Evaluating and Improving the Robustness of Image Classifiers against Adversarial Attacks

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
M.Sc. Francesco Croce
aus Turin / Italien

Tübingen
2023

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	28.09.2023
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter/-in:	Prof. Dr. Matthias Hein
2. Berichterstatter/-in:	Prof. Dr. Matthias Bethge
3. Berichterstatter/-in:	Assistant Prof. Dr. Sijia Liu

Abstract

The decisions of state-of-the-art image classifiers based on neural networks can be easily changed by small perturbations of the input which, at the same time, would not fool humans. These *adversarial* points are ubiquitous and question the deployment of such models in safety-critical systems. Moreover, this phenomenon shows that the networks learn to rely, for classification, on different features of the input images compared to people.

The discovery of the existance of adversarial examples opened up several rich research directions: adversarial attacks want to efficiently generate perturbations to fool a target classifier while preserving the original semantic content of the input. To make sure that the ground truth class is not altered, one typically imposes constraints on some l_p-norm of the perturbations, or on the number (and location) of modified pixels (sparse attacks). Conversely, defensive mechanisms aim at training models which are not (or less) vulnerable to adversarial manipulations. The effectiveness of empirical defenses is in practice evaluated by measuring the success of adversarial attacks against them. Then it is important to design strong and reliable attacking schemes: in fact, it has been shown that some initially successful defenses were still vulnerable to more sophisticated attacks. An alternative approach which avoids relying on attacks for robustness evaluation is represented by certified defenses. These methods provide guarantees that no adversarial perturbation smaller than some threshold exists for a given clean point. The drawback of this approach is that it is either limited to small networks and datasets or incurs in high computational cost (and the decisions of the classifier become stochastic).

In this work, we touch these various aspects of adversarial robustness: first, we derive lower bounds of the adversarial perturbations of ReLU networks, and we use them as regularization during training to obtain provably robust classifiers against either a single or multiple threats.

Second, we propose algorithms to generate adversarial perturbations in different threat models, both in the white- and black-box scenarios, i.e. with and without having access to the target network parameters. We consider various l_p-bounded attacks for $p \in \{\infty, 2, 1\}$, sparse attacks (individual pixels, patches, frames), and imperceivable perturbations which are defined adaptively for each input image.

We then gather a subset of those methods to form AutoAttack, a protocol for reliable evaluation of adversarial robustness which does not require parameter tuning. Exploiting AutoAttack, we introduce RobustBench, a standardized benchmark to evaluate the progress of the robustness of classifiers against adversarial attacks in the l_p-threat models, which also provides access to a larger zoo of adversarially robust models. Since RobustBench imposes some restrictions on the allowed models, we complement it with a study on adaptive test-time defenses which do not satisfy such restrictions but are popular in the literature.

Finally, we analyze how to quickly adapt the type of robustness of a given model: a short fine-tuning, even of single epoch, is sufficient to make a classifier adversarially trained against an l_p-attack robust to a different l_q-bounded threat.

Kurzfassung

Die Entscheidungen moderner Bildklassifizierer, die auf neuronalen Netzen basieren, können leicht durch kleine Störungen der Eingaben verändert werden, die den Menschen jedoch nicht täuschen würden. Diese *adversarial images* sind allgegenwärtig und stellen den Einsatz solcher Modelle in sicherheitskritischen Systemen in Frage. Darüber hinaus zeigt dieses Phänomen, dass die Netze lernen, sich bei der Klassifizierung auf andere Merkmale der Eingabebilder zu verlassen als Menschen.

Die Entdeckung von *adversarial examples* eröffnete mehrere interessante Forschungsrichtungen: Attacken zielen darauf ab, effizient Störungen zu erzeugen, um einen Zielklassifikator zu täuschen, während der ursprüngliche semantische Inhalt der Eingabe erhalten bleibt. Um sicherzustellen, dass der semantische Inhalt der wahren Klasse nicht verändert wird, wird typischerweise die l_p-Norm der Störungen oder die Anzahl (und der Ort) der veränderten Pixel (spärliche Angriffe) beschränkt. Umgekehrt zielen defensive Mechanismen darauf ab, Modelle zu trainieren, die nicht (oder weniger) anfällig für bösartige Manipulationen sind. Die Wirksamkeit empirischer Abwehrmechanismen wird in der Praxis durch Messung des Erfolgs bösartige Angriffe auf das Modell bewertet. In diesem Fall ist es wichtig, starke und zuverlässige Angriffsverfahren zu entwickeln: Es hat sich nämlich gezeigt, dass einige ursprünglich erfolgreiche Verteidigungsmechanismen immer noch anfällig für ausgefeiltere Angriffe waren. Ein alternativer Ansatz, der es vermeidet, sich bei der Bewertung der Robustheit auf Angriffe zu stützen, ist der der zertifizierten Verteidigungsmaßnahmen. Diese Methoden garantieren, dass es für einen gegebenen für ein gegebenes Bild keine bösartige Störung durch einen Angreifer existiert, die kleiner als ein bestimmter Schwellenwert ist. Der Nachteil dieses Ansatzes ist, dass er entweder auf kleine Netze und Datensätze beschränkt ist oder hohe Rechenkosten verursacht (und Klassifikationsentscheidungen stochastisch werden).

In dieser Arbeit bearbeiten wir diese verschiedenen Aspekte der Robustheit gegenüber nachteiligen Einflüssen: Erstens leiten wir untere Schranken für die bösartigen Störungen von ReLU-Netzwerken ab und verwenden sie als Regularisierung während des Trainings, um nachweislich robuste Klassifikatoren gegen ein oder mehrere Störungsmodelle zu erhalten.

Zweitens schlagen wir Algorithmen vor, um bösartige Störungen in verschiedenen Bedrohungsmodellen zu erzeugen, sowohl im White- als auch im Black-Box-Szenario, d.h. mit und ohne Zugriff auf die Parameter des Zielnetzwerks. Wir betrachten verschiedene l_p-begrenzte Angriffe für $p \in \{\infty, 2, 1\}$, spärliche Angriffe (einzelne Pixel, Patches, Frames) und nicht wahrnehmbare Störungen, die adaptiv für jedes Eingabebild definiert werden.

Anschließend fassen wir eine Teilmenge dieser Methoden zu AutoAttack zusammen, einem Protokoll zur zuverlässigen Bewertung der Robustheit gegen bösartige Attacken, das keine Parametereinstellung erfordert. Unter Ausnutzung von AutoAttack entwickeln wir RobustBench, einem standardisierten Benchmark der es erlaubt den Fortschritt der Robustheit von Klassifikatoren gegen bösartige Attacken bezüglich der l_p Bedrohungsmodelle zu evaluieren, und auch Zugang zu einem größeren Zoo widerstandsfähiger Modelle bietet. Da RobustBench einige Beschränkungen an die zulässigen Modelle aufstellt, ergänzen wir es durch eine Studie über adaptive Verteidigung zur Testzeit, die diese Beschränkungen nicht erfüllen, aber in der Literatur beliebt sind.

Schließlich analysieren wir, wie man die Art der Robustheit eines gegebenen Modells schnell anpassen kann: ein kurzes Feintuning, sogar von einer einzigen Epoche, reicht aus, um einen Klassifikator, der gegen einen l_p-Angriff trainiert wurde, robust gegenüber einer anderen l_q-begrenzten Bedrohung zu machen.

Acknowledgements

I am deeply grateful to my advisor Matthias Hein for giving me first the opportunity to explore a new field I knew close to nothing about, and then the time to find my way through it. Matthias supported me during the entire journey, always providing his guidance and priceless insights in all projects we worked on together, and pushing me outside my comfort zone when needed. I could (hopefully) learn from him the principles of doing good research.

Then, a great influence on my work has come from Maksym Andriushchenko, with whom I shared several projects. I must thank Maksym not just for teaching me a lot about coding and Twitter dynamics, but mostly for all the on- and off-line discussions, and always offering new different perspectives and world wisdom, which I very much need.

I would also like to thank Sven Gowal, for both the scientific collaborations and the opportunity of having an amazing time during the internship at DeepMind, which allowed me to take a glance at research outside academia.

I want to thank all collaborators I had during my PhD for the knowledge and experience they shared with me while working together. Moreover, I thank all the colleagues in the ML group, first in Saarbrücken and then in Tübingen, for creating a welcoming and vibrant environment, which always offered occasions for learning about new topics, work-related and not, and having a good time while sharing a coffee.

I would like to thank Shweta and Apratim, who offered me a room to live and a place in their family when I first arrived to Germany, and with whom I shared several fun moments. Also, I must thank my longtime friends Claudia, Elena, Emanuele, Lorenzo, Marco and Nicolò, for creating great memories together, and bearing with my quirks.

Finally, I want to thank my parents and my sister for their love: they taught, and keep teaching, me life lessons which I could not have learnt otherwise.

Details of collaborations

List of references. We list below the references to the papers on which this work is based.

- Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In *AISTATS*, 2019a.
- Francesco Croce and Matthias Hein. Sparse and imperceivable adversarial attacks. In *ICCV*, 2019.
- Francesco Croce and Matthias Hein. Provable robustness against all adversarial l_p-perturbations for $p \geq 1$. In *ICLR*, 2020b.
- Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *ICML*, 2020a.
- Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020c.
- Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *ECCV*, 2020.
- Francesco Croce and Matthias Hein. Mind the box: l_1-apgd for sparse adversarial attacks on image classifiers. In *ICML*, 2021.
- Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. In *NeurIPS Datasets and Benchmarks Track*, 2021.
- Francesco Croce, Maksym Andriushchenko, Naman D Singh, Nicolas Flammarion, and Matthias Hein. Sparse-rs: a versatile framework for query-efficient sparse black-box adversarial attacks. In *AAAI*, 2022a.
- Francesco Croce and Matthias Hein. Adversarial robustness against multiple and single l_p-threat models via quick fine-tuning of robust classifiers. In *ICML*, 2022.
- Francesco Croce, Sven Gowal, Thomas Brunner, Evan Shelhamer, Matthias Hein, and Taylan Cemgil. Evaluating the adversarial robustness of adaptive test-time defenses. In *ICML*, 2022b.

Details of contributions. In all works listed above F.C. contributed as (sole or shared) first author. In particular, M.A. and F.C. equally contributed to Croce et al. [2019a], Andriushchenko et al. [2020], F.C., M.A., V.S. and E.D. equally contributed to Croce et al. [2021], F.C., S.G., T.B. and E.S. equally contributed to Croce et al. [2022b].

Contributions of collaborators. In the following we detail the parts of this work which are partially or solely contributions of collaborators.

- For all papers included in this work, Matthias Hein participated to the development of the projects and writing.
- In Sec. 3.2, based on Croce et al. [2019a], Maksym Andriushchenko conducted the experiments for training robust models and contributed to the visualizations of Sec. 3.4.
- In Sec. 3.3, based on Croce and Hein [2020b], Matthias Hein provided the theoretical analysis in Sec. 3.3.2 and contributed to finalize the various proofs.
- Maksym Andriushchenko initiated the project of Square Attack (Sec. 4.2, based on Andriushchenko et al. [2020]), designed the algorithm of the l_∞-attack, and conducted the relative experiments and ablation studies. He also provided the visualizations in Fig. 4.8. Moreover,

Nicolas Flammarion provided the convergence analysis in Sec. 4.2.4 and the analysis about updates of equal sign in Sec. 4.2.5.

- Matthias Hein contributed to the design of the DLR loss presented in Sec. 4.3.3 [Croce and Hein, 2020c].
- In Sec. 4.4, based on Croce and Hein [2021], Matthias Hein contributed to all theoretical results and analyses in Sec. 4.4.2 and Sec. 4.4.3, in particular providing the results in Lemma 4.4.1 and Proposition 4.4.3.
- In Sec. 5.2 all main authors contributed to the design and implementation of RobustBench [Croce et al., 2021], as well as to the writing of the paper. In the analysis part (Sec. 5.2.4), Maksym Andriushchenko, Vikash Sehwag and Edoardo Debenedetti conducted the experiments about performance across distribution shifts, calibration, out-of-distribution detection, privacy leakage, smoothness and transferability.
- For Sec. 5.3, based on Croce et al. [2022b], all main authors equally contributed to the project design and writing of the paper. Among the evaluations of defenses presented in such section, Sven Gowal and Thomas Brunner conducted those in Sec. 5.3.8, Sec. 5.3.11, Sec. 5.3.13 and Sec. 5.3.15.
- In Sec. 6.1, based on Croce and Hein [2019], Matthias Hein contributed to the formalization of the theoretical results and to the design of the imperceptibility bounds.
- In Sec. 6.2, based on Croce et al. [2022a], Maksym Andriushchenko participated to initiate the project, ran the experiments for a few baselines and participated to formulate the theoretical analysis, Naman D. Singh conducted part of the experiments, Nicolas Flammarion participated to formulate the theoretical analysis.

Contents

Chapter 1

Introduction

The first theoretical bases and implementation of neural networks (NNs) as tools for solving learning problems appeared in the second half of 1900s [Schmidhuber, 2015], but their popularity quickly increased when their training and inference could be successfully combined with hardware advancements, that is the computations were efficiently parallelized on GPUs [Chellapilla et al., 2006, Ciresan et al., 2011, Krizhevsky et al., 2012]. The last decade has seen NNs-based models achieving outstanding performance in several domains, ranging from computer vision [Krizhevsky et al., 2012, He et al., 2016a,b, Zagoruyko and Komodakis, 2016], natural language processing [Amodei et al., 2016, Devlin et al., 2018, Brown et al., 2020] to mastering classical and computer games [Mnih et al., 2013, Silver et al., 2018], getting in most of the cases super-human results. Moreover, NNs are now being integrated in safety-critical systems like autonomous driving, medical diagnosis, malware detection and financial risk assessment.

Despite their success on multiple tasks, it has been early on discovered [Biggio et al., 2013, Szegedy et al., 2014] that the decisions of NNs-based models are not robust to small perturbations of the input if those are crafted in an adversarial way, i.e. with the specific goal of misleading the model. For example, for image classification tasks, it is now well-known that an image can be altered with imperceptible perturbations which make a well-trained classifier, which correctly identifies the original image and has high generalization performance, predict the wrong class, possibly with high confidence. This phenomenon is consistently present across architectures, datasets and even domains, and it is not limited to vision problems as shown in Szegedy et al. [2014] but appears in, among others, natural language processing [Li et al., 2018, Jin et al., 2019, Boucher et al., 2022], malware detection [Grosse et al., 2016, Kolosnjaji et al., 2018], and reinforcement learning [Huang et al., 2017, Lin et al., 2017] tasks. Such inputs are referred to as *adversarial samples*.

While it is hard to provide a comprehensive formal definition, adversarial perturbations should i) modify the original input in a way that either is unnoticeable to humans or does not make humans to change their decision in the considered task, and ii) lead the model to modify its output to a wrong one (e.g. predicting the wrong class). It is worth noting that the vulnerability to adversarial samples is not exclusive of (deep) neural networks, but affects also traditional learning systems like SVMs [Demontis et al., 2016], decision stumps and trees [Andriushchenko and Hein, 2019, Zhang et al., 2020a], k-nearest neighbors [Sitawarin and Wagner, 2020], and it had been already studied by Lowd and Meek [2005] for linear classifiers for spam filtering. This phenomenon is of particular concern when it comes to deploying the decision systems in high stake situations e.g. on autonomous vehicles, medical tasks or malware detection, where it is vital for the user to have a certain degree of trust in the system. In particular, it has been shown that adversarial manipulations are not only realizable in the digital world but even in the physical one, for example by stickers on traffic signs or adversarial T-shirts and glasses [Evtimov et al., 2018, Li et al., 2019b, Lee and Kolter, 2019, Thys et al., 2019, Xu et al., 2020b, Sharif et al., 2019].

While it has been debated whether an adversary would be able to manipulate real-world systems in a practical scenario, the existence of adversarial samples poses by itself interesting questions about which features are actually learnt by neural networks, and how these differ from human perceptions. Consequently, the field of adversarial machine learning has grown rapidly

over the last few years, and counts now more than 6000 papers[1] which analyze a variety of aspects of the problem. The most numerous subsets of such works are probably those which focus on either generating adversarial perturbations (adversarial attacks) [Szegedy et al., 2014, Carlini and Wagner, 2017a] or training models which are robust to them (empirical defenses) [Goodfellow et al., 2015, Madry et al., 2018]. Moreover, important lines of work investigate from a theoretical standpoint the existence of adversarial samples [Allen-Zhu and Li, 2020, Bubeck et al., 2021], how to certify that a model is adversarially robust at a given input (certification methods) [Raghunathan et al., 2018, Wong and Kolter, 2018, Lecuyer et al., 2019] or how to find the minimal modifications which change its decisions (verification methods) [Katz et al., 2017, Tjeng et al., 2019]. Another major subfield consists in the study of the properties of adversarially robust models: it has been in fact shown that those present significantly different characteristics compared to standard ones, e.g. tend to have lower generalization performance on non adversarial data [Tsipras et al., 2019], more interpretable gradients [Santurkar et al., 2019], and are more suitable for transfer learning [Salman et al., 2020]. At the same time, training such robust models presents specific challenges like catastrophic [Wong et al., 2020] and robust [Rice et al., 2020] overfitting, which do not appear in the natural (i.e. non adversarial) setup. A further research direction focuses on defining metrics which are well aligned with human perception [Laidlaw et al., 2021]: in fact, in order to be imperceivable, adversarial changes are often constrained to have small l_p-norm, for some p, but this has been at times criticized since such norms, while practical, are not always good proxies of visual similarity. Finally, a line of work tries to track the progress of the field over time, which is not a simple task given its activity and variety [Kurakin et al., 2018, Brendel et al., 2018, Dong et al., 2020].

Since attacks and (provable and empirical) defenses for image classifiers are the main focus of this work, we will provide a more detailed description of them in the next section, together with an overview of the most relevant related papers.

Contributions. In this work, we focus on different aspects of the robustness of image classifiers. We provide below a summary of our contributions, with an outline of the rest of the manuscript and references to the papers on which the various parts are based.

- **Provable robustness:** First, in Sec. 3 we exploit the structure of piecewise affine function of ReLU networks to derive either the exact value or a lower bound on the size of smallest (wrt an l_p-norm) perturbation which changes the decision of the classifier for a given input. This motivates a regularization scheme which provides models with high provable robustness to l_p-bounded perturbations for a single p (see Croce et al. [2019a]). We then extend such approach to obtain guarantees for multiple threat models, i.e. all $p \geq 1$, simultaneously [Croce and Hein, 2020b].

- **Adversarial attacks:** We present several adversarial attacks in the l_∞- and l_2-threat models, which are the most popular ones in the literature, in both black- and white-box scenarios (i.e. without or with access to the internals of the target classifier). In details, we introduce the Fast Adaptive Boundary Attack [Croce and Hein, 2020a] in Sec. 4.1, the Square Attack [Andriushchenko et al., 2020] in Sec. 4.2, and Auto-PGD [Croce and Hein, 2020c] in Sec. 4.3. Moreover, in Sec. 4.4 we analyze the l_1-threat model, and extend Square Attack and Auto-PGD to that case (see Croce and Hein [2021]). For space constraints we omit the description of the Linear Region Attacks, specialized for ReLU networks, from Croce and Hein [2018], Croce et al. [2019b].

- **Benchmarking adversarial robustness:** In Sec. 5.1 we combine a subset of the attacks previously introduced to create AutoAttack [Croce and Hein, 2020c], a protocol for robustness evaluation without the need of parameter tuning. AutoAttack is then used as a first step to define a standardized benchmark of adversarial robustness, named RobustBench [Croce et al., 2021] (see Sec. 5.2). It aims at systematically tracking the progress of the field, which requires imposing some restrictions on the allowed classifiers. Moreover, its Model Zoo gives easy access to many robust models, which facilitates the analyses of their properties. Finally, we conduct a study on adaptive test-time defenses, a class of methods which is excluded from RobustBench, in Sec. 5.3 (see Croce et al. [2022b]).

[1] https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html

- **Sparse attacks:** Sec. 6 presents algorithms to generate sparse attacks which modify only a small percentage of the pixels of every image, including adversarial patches and frames, and shows how to combine these techniques with imperceptibility constraints. In particular, we introduce methods which either minimize the number of perturbed pixels [Croce and Hein, 2019] or have a fixed budget and aim at query efficiency, i.e. succeeding with the smallest amount of queries to the target classifier [Croce et al., 2022a].
- **Adaptation of robust classifiers via fine-tuning:** In Sec. 7 we study how to achieve with low computational budget empirical robustness in multiple threat models. In particular, we show that one can obtain models robust to a single or multiple l_p-norm bounded attacks by a short fine-tuning, with adversarial training, of classifiers initially robust to an l_q-attack with $q \neq p$ [Croce and Hein, 2022].

Note that in the following we will refer as state-of-the-art to the best methods or results available at the time of writing the papers the corresponding section is based on. Similarly, we might report statistics which reflect the status at that time. We will add for completeness, when necessary, a paragraph commenting on the relevant progress in the field which followed our contributions.

List of references. We list below the references to the papers on which this work is based. Additional experiments and analyses of the presented methods can be found in the corresponding papers.

- Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In *AISTATS*, 2019a.
- Francesco Croce and Matthias Hein. Sparse and imperceivable adversarial attacks. In *ICCV*, 2019.
- Francesco Croce and Matthias Hein. Provable robustness against all adversarial l_p-perturbations for $p \geq 1$. In *ICLR*, 2020b.
- Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *ICML*, 2020a.
- Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020c.
- Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *ECCV*, 2020.
- Francesco Croce and Matthias Hein. Mind the box: l_1-apgd for sparse adversarial attacks on image classifiers. In *ICML*, 2021.
- Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. In *NeurIPS Datasets and Benchmarks Track*, 2021.
- Francesco Croce, Maksym Andriushchenko, Naman D Singh, Nicolas Flammarion, and Matthias Hein. Sparse-rs: a versatile framework for query-efficient sparse black-box adversarial attacks. In *AAAI*, 2022a.
- Francesco Croce and Matthias Hein. Adversarial robustness against multiple and single l_p-threat models via quick fine-tuning of robust classifiers. In *ICML*, 2022.
- Francesco Croce, Sven Gowal, Thomas Brunner, Evan Shelhamer, Matthias Hein, and Taylan Cemgil. Evaluating the adversarial robustness of adaptive test-time defenses. In *ICML*, 2022b.

Omitted works. We list below the works which will not be discussed in this thesis.

- Francesco Croce and Matthias Hein. A randomized gradient-free attack on relu networks. In *GCPR*, 2018.
- Francesco Croce, Jonas Rauber, and Matthias Hein. Scaling up the randomized gradient-free adversarial attack reveals overestimation of robustness using established attacks. *International J. of Computer Vision (IJCV)*, 2019b.
- Francesco Croce and Matthias Hein. On the interplay of adversarial robustness and architecture components: patches, convolution and attention. *arXiv preprint arXiv:2209.06953*, 2022b.
- Valentyn Boreiko, Maximilian Augustin, Francesco Croce, Philipp Berens, and Matthias Hein. Sparse visual counterfactual explanations in image space. In *GCPR*, 2022.
- Maximilian Augustin, Valentyn Boreiko, Francesco Croce, and Matthias Hein. Diffusion visual counterfactual explanations. In *NeurIPS*, 2022.
- Sylvestre-Alvise Rebuffi, Francesco Croce, and Sven Gowal. Revisiting adapters with adversarial training. In *ICLR*, 2023.
- Francesco Croce, Sylvestre-Alvise Rebuffi, Evan Shelhamer, and Sven Gowal. Seasoning model soups for robustness to adversarial and natural distribution shifts. In *CVPR*, 2023a.
- Maksym Andriushchenko, Francesco Croce, Maximilian Müller, Matthias Hein, and Nicolas Flammarion. A modern look at the relationship between sharpness and generalization. In *ICML*, 2023.
- Naman D Singh, Francesco Croce, and Matthias Hein. Revisiting adversarial training for imagenet: Architectures, training and generalization across threat models. *arXiv preprint arXiv:2303.01870*, 2023.
- Francesco Croce, Naman D Singh, and Matthias Hein. Robust semantic segmentation: Strong adversarial attacks and fast training of robust models. *arXiv preprint arXiv:2306.12941*, 2023b.

Chapter 2

Background and Related Work

In the following we first provide the main definitions which we use in the next sections, together with the relative notation. Then we give an overview of adversarial attacks, empirical defenses, and methods for certified robustness.

2.1 Notation

A NN-based classifier can be described as a function $f : X \times \Theta \to \mathbb{R}^K$, where X and Θ are the input and weight space respectively, and K the number of classes of the problem. Let $\theta \in \Theta \subseteq \mathbb{R}^w$ contain the parameters which identify the neural network, and $x \in X \subseteq \mathbb{R}^d$ be a generic input, then $z = f(x, \theta)$ is a vector of dimension the number of classes, referred to as logits, and the decision of f is given by $\arg\max_{r=1,\dots,K} f_r(x, \theta)$, i.e. the index of the maximal logit. When θ is fixed, we might omit to explicitly indicate the dependence of f on θ and write $f(x)$ for brevity. The architecture of a neural network usually consists in a sequence of affine and non-linear, possibly non-parametric, functions, referred to as layers: the vector θ stores all the parameters describing the layers and which are typically learnt during training. Since we consider image classification tasks, the inputs are mostly three dimensional tensors with shape $h \times w \times c$, where c is the number of color channels (e.g. $c = 1$ for grayscale images, $c = 3$ for RGB ones) and $h \cdot w \cdot c = d$. Moreover, x is scaled to be in the image domain, meaning that each of its components belongs to $[0, 1]$ (other types of normalization are possible, but we do not consider them here).

The classification accuracy of f on a set $D = \{(x_i, y_i)\}_{i=1}^N$, whose elements are pairs of an input $x_i \in X$ and its corresponding ground-truth label $y_i \in \{1, \dots, K\}$, is defined, with $\mathbf{1}(\cdot)$ the indicator function, as

$$\frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\arg\max_{r=1,\dots,K} f_r(x_i, \theta) = y_i), \tag{2.1}$$

that is the fraction of data points for which the prediction of the classifier matches the correct label. In particular, when D contains unperturbed images, e.g. the test set, Eq. (2.1) provides the *clean* or *standard accuracy*, while if D is formed by adversarial samples it gives the *robust accuracy* of f under the considered threat model. Additionally, the complementary metrics *clean* and *robust test error* are sometimes used, indicating the percentage of (clean or advervarial) test points which are misclassified. A closely related metric which is mostly used for assessing the effectiveness of adversarial attacks is the success rate, which measures the percentage of points for which f changes its decision (possibly into a specific target class) after they are adversarially perturbed. Moreover, we denote as $B_p(x, \epsilon)$ the l_p-ball of radius $\epsilon \geq 0$ around $x \in \mathbb{R}^d$, i.e. $B_p(x, \epsilon) = \{z \in \mathbb{R}^d : \|z - x\|_p \leq \epsilon\}$.

2.2 Adversarial attacks

The common goal of adversarial attacks is to find a small perturbation $\delta \in \mathbb{R}^d$ which changes the decision of a classifier f at an input $x \in \mathbb{R}^d$ with correct label y: in the untargeted scenario any

misclassification is acceptable, i.e. $\arg\max_{r=1,\dots,K} f_r(x+\delta) \neq y$ is sufficient, while for targeted attacks one wants the decision of f to match a target label t, i.e. $\arg\max_{r=1,\dots,K} f_r(x+\delta) = t$. We recall that adversarial samples should differ from the original ones in such a way that the ground-truth class is still clearly recognizable. In practice, this is reflected by enforcing x and $x+\delta$ to be close wrt some distance measure $\mu : X \times X \to \mathbb{R}_+$ which approximates human perception.

Attack types. There exist two main types of attacks: a first group minimizes the distance between original and manipulated image, i.e., formally, they try to solve the problem

$$\arg\min_{\delta\in\mathbb{R}^d} \mu(x, x+\delta) \quad \text{s. th.} \quad \arg\max_{r=1,\dots,K} f_r(x+\delta) \neq y, \quad x+\delta \in X. \tag{2.2}$$

Another group fixes instead a maximal size $\epsilon > 0$ on the perturbation and approximately solves

$$\arg\max_{\delta\in\mathbb{R}^d} L(f, x, \delta, y) \quad \text{s. th.} \quad \mu(x, x+\delta) \leq \epsilon, \quad x+\delta \in X, \tag{2.3}$$

where L is a surrogate loss which induces misclassification e.g. cross-entropy. Note that Eq. (2.2) is stated for the untargeted case but can be easily extended to the targeted scenario by adapting the first constraint. The first formulation yields an optimization problem which is in general hard to solve directly because of the misclassification constraint. Then, prior works have, for example, rephrased it via penalty methods [Carlini and Wagner, 2017a] or replacing f with its local linear approximation [Moosavi-Dezfooli et al., 2016] to make it tractable. The second formulation, in Eq. (2.3), has the advantage that can be solved with standard methods for constrained optimization, e.g. projected gradient descent (PGD) when $\{\delta \in \mathbb{R}^d : \mu(x, x+\delta) \leq \epsilon\}$ is a set onto which a projection exists [Madry et al., 2018]. Since the optimization problems in Eq. (2.2) and Eq. (2.3) have different objective functions, it is not straightforward how to compare their results. Possible solutions are i) running fixed budget attacks with several values of ϵ to find the smallest δ which fools the classifier, or ii) considering a solution δ^* of Eq. (2.2) successful only if $\mu(x, x+\delta^*) \leq \epsilon$. We will use mostly the latter, since it translates directly into robust accuracy which is the most popular metric for measuring robustness.

Level of knowledge. One can further distinguish adversarial attacks according to the level of knowledge of the target classifier they have. In the white-box scenario, the attacker has access to the weights θ parameterizing the neural network, the training data and the training scheme, and is aware of any defense mechanism adopted to improve the robustness (even at test time). This also means that it can compute the gradient wrt the input of any function applied on top of the logits. White-box algorithms [Goodfellow et al., 2015, Carlini and Wagner, 2017a, Madry et al., 2018] typically provide the strongest (and most computationally efficient) evaluation of robustness since they have full knowledge of the target classifier. Conversely, black-box attacks can only query the victim model, i.e. they can feed it with an arbitrary input and receive back either the values of the corresponding logits (score based attacks) [Fawzi and Frossard, 2016, Ilyas et al., 2018, Moon et al., 2019, Guo et al., 2019] or only the predicted class (decision based attacks) [Brendel et al., 2018, Brunner et al., 2019, Chen and Gu, 2020]. Moreover, some black-box attacks leverage a surrogate model similar to the target one to generate more effective perturbations [Papernot et al., 2016a, Cheng et al., 2019, Huang and Zhang, 2020]: in this case the similarity between substitute and victim models is key for the success of the attack.

Perturbation constraints. The most popular metrics used to enforce similarity between the original and perturbed images are l_p-norms with $p \in \{\infty, 2, 1\}$, since these lead to more tractable optimization problems. Using other (fractional) l_p-norms is also possible [Boreiko et al., 2022] although less common. These attacks typically modify all input components of a small magnitude. An opposite approach is that of sparse attacks, which instead perturb only a small fraction of the input, e.g. a few pixels of an image, but possibly with visible modifications. These include l_0-bounded attacks [Carlini and Wagner, 2017a], adversarial patches [Brown et al., 2017] and frames [Zajac et al., 2019]. Moreover, metrics other than l_p-norms which are better aligned to human perception have been recently proposed, e.g. Wasserstein distance [Wong et al., 2019,

Hu et al., 2020] or neural-network based metrics such as LPIPS [Zhang et al., 2018, Laidlaw et al., 2021]. Finally, Xiao et al. [2018] create adversarial attacks via spatial transformations, while Kang et al. [2019b] propose adversarial corruptions which try to mimic naturally occurring modifications of an image like snow or fog.

2.2.1 l_p-bounded white-box attacks

This is the most popular type of attacks, as it involves a well-defined set of perturbations and allows the attacker to access all elements of the target classifier. Goodfellow et al. [2015] first propose to generate perturbations with fixed l_∞-norm with the Fast Gradient Sign Method (FGSM), which takes a single gradient step to maximize a target loss function (see Eq. (2.3)). Iterative versions of FGSM have been later proposed by Kurakin et al. [2017a] (BIM) and Madry et al. [2018] (PGD), which rely on projecting each iterate onto the target l_∞-ball to ensure that the perturbations remain imperceivable. With small modifications to the gradient step and the projection algorithm, these methods, and PGD in particular, are then adapted to the l_2 [Madry et al., 2018, Rony et al., 2019] and l_1 [Tramèr and Boneh, 2019, Maini et al., 2020] threat models, and enriched by several variants [Dong et al., 2018, Zheng et al., 2019]. The most common choices as objective functions are cross-entropy and margin loss [Madry et al., 2018, Gowal et al., 2019b]. An alternative approach which does not need to project onto the feasible set is represented by attacks based on Frank-Wolfe schemes [Chen et al., 2020a].

Carlini and Wagner [2017a] aim instead at minimizing the l_p-norm of the perturbations, by adding a penalization term to the loss which is optimized by the attack, for $p \in \{\infty, 2, 0\}$, and Chen et al. [2018] later extended it to the l_1 case. Since these methods rely on a bilevel optimization procedure to find the parameter which yields the best performance, they have high computational cost. Efficient l_p-attacks are introduced by Moosavi-Dezfooli et al. [2017], Modas et al. [2019]: they replace at each iteration the target classifier with its first-order approximation for which the smallest adversarial perturbations can be found in closed form. While such algorithms typically find perturbations in a few steps, those have large size. Several more sophisticated techniques of similar spirit have been later proposed [Rony et al., 2021, Brendel et al., 2019, Pooladian et al., 2020, Matyasko and Chau, 2021, Pintor et al., 2021], and manage to improve both quality and runtime of the attacks.

These methods are developed to perform well on a variety of classifiers, which are typically deterministic and differentiable. Therefore they might fail on defenses with some unexpected components which violate such conditions (see Sec. 5.2 for a more detailed discussion). Then, Athalye et al. [2018] introduced tools to, at least partially, adapt common attacks to such cases: Expectation over Transformations (EoT) for randomized defenses and Backward Pass Differentiable Approximation (BPDA) for non-differentiable layers. Evaluating the robustness of such defenses in a standardized way remains however a challenging task, and typically requires adaptive techniques [Tramèr et al., 2019].

2.2.2 l_p-bounded score-based attacks

Score-based black-box attacks have only access to the score predicted by a classifier for each class for a given input. Most of such attacks in the literature are based on gradient estimation through finite differences. The first papers in this direction [Bhagoji et al., 2018, Ilyas et al., 2018, Uesato et al., 2018] propose attacks which approximate the gradient by sampling from some noise distribution around the point. While this approach can be successful, it requires many queries of the classifier, particularly in high dimensional input spaces as in image classification. Thus, improved techniques reduce the dimension of the search space via using the principal components of the data [Bhagoji et al., 2018], searching for perturbations in the latent space of an auto-encoder [Tu et al., 2019] or using a low dimensional noise distribution [Ilyas et al., 2019]. Other attacks exploit evolutionary strategies or random search, e.g. Alzantot et al. [2019] use a genetic algorithm to generate adversarial examples and alleviate gradient masking as they can reduce the robust accuracy on randomization- and discretization-based defenses, while the l_2-attack of Guo et al. [2019] can be seen as a variant of random search which chooses the search directions in an orthonormal basis and tests up to two candidate updates at each step. A recent line of work has pursued black-box attacks which are based on the observation that successful adversarial

perturbations are attained at corners of the l_∞-ball intersected with the image space $[0,1]^d$ [Moon et al., 2019, Al-Dujaili and O'Reilly, 2020, Meunier et al., 2019]. Searching over the corners allows to apply discrete optimization techniques to generate adversarial attacks, significantly improving the query efficiency. Both Moon et al. [2019] and Al-Dujaili and O'Reilly [2020] divide the image according to some coarse grid, perform local search in this lower dimensional space allowing componentwise changes only of $-\epsilon$ and ϵ, then refine the grid and repeat the scheme. Recently, Meunier et al. [2019] proposed several attacks based on evolutionary algorithms, using discrete and continuous optimization, achieving nearly state-of-the-art query efficiency for the l_∞-norm. In order to reduce the dimensionality of the search space, they use the "tiling trick" of Ilyas et al. [2019] where they divide the perturbation into a set of squares and modify the values in these squares with evolutionary algorithms. A related idea also appeared earlier in Fawzi and Frossard [2016] where they introduced black rectangle-shaped perturbations for generating adversarial occlusions. Despite their effectiveness for the l_∞-norm, these discrete optimization based attacks are not straightforward to adapt to e.g. the l_2-norm. Finally, approaches based on Bayesian optimization exist, e.g. Shukla et al. [2019], but show competitive performance only in a low-query regime.

2.2.3 Sparse attacks

Sparse attacks pursue a particular strategy: they perturb only a small portion of the original input but possibly with large modifications. Thus, the perturbations are indeed visible but do not alter the semantic content, and can even be applied in the physical world [Lee and Kolter, 2019, Thys et al., 2019, Li et al., 2019b]. Sparse attacks include l_0-attacks [Narodytska and Kasiviswanathan, 2017, Carlini and Wagner, 2017a, Papernot et al., 2016b, Schott et al., 2019], adversarial patches [Brown et al., 2017, Karmon et al., 2018, Lee and Kolter, 2019] and frames [Zajac et al., 2019], where the perturbations have some predetermined structure. Moreover, sparse attacks generalize to tasks outside computer vision, such as malware detection or natural language processing, where the nature of the domain imposes to modify only a limited number of input features [Grosse et al., 2016, Jin et al., 2019]. Finally, sparsity in the adversarial perturbations has also been linked to better interpretability of the attacks [Xu et al., 2018].

For the l_0-threat model, only a few score-based black-box attacks exist [Narodytska and Kasiviswanathan, 2017, Schott et al., 2019, Zhao et al., 2019]. Since they do not focus on query efficiency, they hardly scale to datasets like ImageNet [Deng et al., 2009] without suffering from prohibitive computational cost. For adversarial patches and frames, black-box methods are mostly limited to transfer attacks, that is a white-box attack is performed on a surrogate model, with the exception of Yang et al. [2020a] who use a predefined dictionary of patches.

2.3 Empirical defenses

Goodfellow et al. [2015] first proposed to increase the robustness of classifiers by adversarial training: adversarial points for the training images are generated with FGSM at every training step for the current network, and the classification loss is minimized at such points. However, the resulting classifiers are still vulnerable to multi-step attacks [Kurakin et al., 2017b]. Then, Madry et al. [2018] used PGD within the adversarial training framework, and this formulation remains the only general method ensuring adversarial robustness across architectures and datasets [Athalye et al., 2018]. Other types of defenses use more sophisticated techniques, typically preventing the direct optimization of the attack: however, adaptive attacks specifically designed for these defenses have often shown that these alternative techniques are non-robust or much less robust than claimed [Tramèr et al., 2020]. Recent improvements of adversarial training have been achieved by using different objectives [Zhang et al., 2019b], unlabeled data [Carmon et al., 2019, Alayrac et al., 2019], adversarial weights perturbations [Wu et al., 2020a] and wider networks [Wu et al., 2021a]. In Gowal et al. [2020] several recent variants were systematically explored and for a very large architecture they obtain the most robust models for l_∞ and l_2 for CIFAR-10. These results were improved later on by using particular data augmentation [Rebuffi et al., 2021a] or synthetic data [Gowal et al., 2021, Wang et al.].

It was early on discovered that adversarial robustness against a specific l_p-threat model does not typically transfer to l_q-threat models for $p \neq q$ (see Kang et al. [2019a], Tramèr and Boneh

[2019] for extensive studies). On the other hand to achieve really reliable machine learning models l_p-robustness wrt all p is necessary. The first approach to general robustness [Schott et al., 2019] uses multiple variational auto-encoders for an analysis by synthesis (ABS) architecture. While ABS is restricted to MNIST, it is robust against l_0, l_2 and l_∞-attacks. Tramèr and Boneh [2019], Maini et al. [2020], Madaan et al. [2021] propose variants of adversarial training to achieve robustness in multiple norms which differ from each other in how the perturbations for different threat models are combined. Madaan et al. [2021] additionally present a meta-learning approach where one learns optimal noise to augment the samples and uses consistency regularization to enforce similar predictions on clean, augmented and adversarial samples. Finally, Stutz et al. [2020] combine adversarial training with a reject option: while that is effective, the comparison to normal adversarially trained models is difficult as their model is non-robust without the reject option.

While fine-tuning of an existing neural network is a commonly used technique in deep learning [Goodfellow et al., 2016] to quickly adapt a model to a different objective e.g. for language models [Howard and Ruder, 2018], it has been recently shown in the area of adversarial robustness that fine-tuning of pre-trained models, possibly using self-supervision, yields better adversarial robustness [Hendrycks et al., 2019, Chen et al., 2020b, Xu and Yang, 2020]. Jeddi et al. [2020] show that fine-tuning of non-robust models with 10 epochs can yield robust models with the caveat that their robustness evaluation is done using only a single run of PGD with 20 steps.

2.4 Certified robustness

The effectiveness of methods which aim at empirical robustness is commonly evaluated by adversarial attacks. However, in some cases such evaluation is not straightforward: for example, it turned out that for many proposed defenses there still exist ways to successfully attack the classifier [Carlini and Wagner, 2017b, Athalye et al., 2018, Mosbach et al., 2018], possibly via adaptive attacks [Tramèr et al., 2020]. Thus, considering the high importance of robustness in safety-critical machine learning applications, we might need robustness guarantees, where one can provide for each test point the radius of a ball on which the classifier will not change the decision, independently of the deployed attack. Therefore recent research has focused on such provable guarantees of a classifier, mostly with respect to the l_p-norms, and existing methods compute either the norm of the minimal perturbation changing the decision at a point (e.g. Katz et al. [2017], Tjeng et al. [2019]) or lower bounds on it [Hein and Andriushchenko, 2017, Raghunathan et al., 2018, Wong and Kolter, 2018]. Several new training schemes and architectures [Hein and Andriushchenko, 2017, Raghunathan et al., 2018, Wong and Kolter, 2018, Mirman et al., 2018, Xiao et al., 2019, Gowal et al., 2019a, Zhang et al., 2020a, Mueller et al., 2023, Leino et al., 2021, Hu et al., 2023, Araujo et al., 2023] aim at both enhancing the robustness of networks and producing models more amenable to certification techniques. However, all of them are only able to prove robustness against a single kind of perturbations, typically either l_2- or l_∞-bounded, and not wrt all the l_p-norms simultaneously.

Randomized smoothing [Lecuyer et al., 2019, Li et al., 2019a, Cohen et al., 2019] provides instead probabilistic guarantees on the robustness of a given classifiers. While it is a computationally expensive, it applies to any type of network and yields guarantees for large datasets which are difficult to handle for deterministic techniques.

Chapter 3

Provable Robustness of ReLU Networks

ReLU networks are feedforward neural networks which use ReLU [Fukushima, 1975] as activation functions, and include convolutional and residual architectures with max- or sum-pooling layers. In the following we propose a regularization scheme for the class of ReLU networks which provably increases the adversarial robustness of the corresponding classifiers.

First, we explicitly rewrite these networks as continuous piecewise affine functions, see Sec. 3.1. Then we leverage this characterization to get either the minimal adversarial perturbation or a lower bound on its size. As a result of this analysis we introduce a new regularization scheme which directly maximizes the lower bound on the robustness guarantee: this allows us to get classifiers with low test error and good robustness guarantees at the same time (Sec. 3.2). While we first focus on robustness with respect to l_2- and l_∞-distance, our approach applies to any l_p-norm. Thanks to this property, in Sec. 3.3 we generalize our analysis and regularization scheme to get robustness guarantees in the union of multiple threat models (l_p-balls in our case). We show in experiments on four datasets that our approach improves lower bounds as well as upper bounds on the robust test error, and can be successfully integrated with adversarial training. Our main observation is that our proposed regularizer significantly improves provable robustness evaluated with the certification method of Wong and Kolter [2018], and especially with the combinatorial solver of Tjeng et al. [2019], since it yields models with properties desirable for applying such techniques. Finally, we provide in Sec. 3.4 visualizations of how the structure of robust models different from that of standard ones.

3.1 Local properties of ReLU networks

Feedforward neural networks which use piecewise affine activation functions (e.g. ReLU, leaky ReLU [Maas et al., 2013]) and are linear in the output layer can be rewritten as continuous piecewise affine functions [Arora et al., 2018].

Definition 3.1.1 *A function $f : \mathbb{R}^d \to \mathbb{R}$ is called* piecewise affine *if there exists a finite set of polytopes $\{Q_r\}_{r=1}^M$ (referred to as* linear regions *of f) such that $\cup_{r=1}^M Q_r = \mathbb{R}^d$ and f is an affine function when restricted to every Q_r.*

In the following we introduce the notation required for the explicit description of the linear regions and decision boundaries of a multi-class ReLU classifier $f : \mathbb{R}^d \to \mathbb{R}^K$ (where d is the input space dimension and K the number of classes) which has no activation function in the output layer. Moreover, we assume that the only non-linear components of the architecture is represented by the activation functions: we show in Sec. 3.1.1 how to derive a similar description for networks with other types of layers e.g. max-pooling. The decision of f at a point x is given by $\arg\max_{r=1,\dots,K} f_r(x)$. The discrimination between linear regions and decision boundaries allows us to share the linear regions across classifier components.

We denote by $\sigma : \mathbb{R} \to \mathbb{R}$, $\sigma(t) = \max\{0, t\}$, the ReLU activation function, $L+1$ is the number of layers, with n_l the numbers of units in the l-th layer and $n_0 = d$. Moreover, $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$

and $b^{(l)} \in \mathbb{R}^{n_l}$ are the weights and offset vectors of layer l, for $l = 1, \dots, L+1$. If $x \in \mathbb{R}^d$ and $g^{(0)}(x) = x$ we define recursively the pre- and post-activation output of every layer as

$$f^{(l)}(x) = W^{(l)} g^{(l-1)}(x) + b^{(l)} \quad \text{and} \quad g^{(l)}(x) = \sigma(f^{(l)}(x)), \quad l = 1, \dots, L,$$

so that the resulting classifier is obtained as $f \equiv f^{(L+1)}(x) = W^{(L+1)} g^{(L)}(x) + b^{(L+1)}$.

We derive the description of the polytope $Q(x)$ in which x lies and the resulting affine function when f is restricted to $Q(x)$. We assume for this that x does not lie on the boundary between two polytopes (which is almost always true as the faces shared by two or more polytopes have dimension strictly smaller than d). Let $\Delta^{(l)}, \Sigma^{(l)} \in \mathbb{R}^{n_l \times n_l}$ for $l = 1, \dots, L$ be diagonal matrices defined elementwise as

$$\Delta^{(l)}(x)_{ij} = \begin{cases} \operatorname{sign}(f_i^{(l)}(x)) & \text{if } i = j, \\ 0 & \text{else.} \end{cases} \quad \text{and} \quad \Sigma^{(l)}(x)_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } f_i^{(l)}(x) > 0, \\ 0 & \text{else.} \end{cases}.$$

This allows us to write $f^{(l)}(x)$ as composition of affine functions, that is

$$f^{(l)}(x) = W^{(k)}\Sigma^{(l-1)}(x)\Big(W^{(l-1)}\Sigma^{(l-2)}(x)\Big(\dots\Big(W^{(1)}x + b^{(1)}\Big)\dots\Big) + b^{(l-1)}\Big) + b^{(l)},$$

We can further simplify the previous expression as $f^{(l)}(x) = V^{(l)}x + a^{(l)}$, with $V^{(l)} \in \mathbb{R}^{n_l \times d}$ and $a^{(l)} \in \mathbb{R}^{n_l}$ given by

$$V^{(l)} = W^{(l)}\Big(\prod_{k=1}^{l-1} \Sigma^{(l-k)}(x) W^{(l-k)}\Big) \quad \text{and}$$
$$a^{(l)} = b^{(l)} + \sum_{k=1}^{l-1}\Big(\prod_{m=1}^{l-k} W^{(l+1-m)}\Sigma^{(l-m)}(x)\Big) b^{(k)}.$$

Note that a forward pass through the network is sufficient to compute $V^{(l)}$ and $a^{(l)}$ for every l, which results in only a small overhead compared to the usual effort necessary to compute the output of f at x. We are then able to characterize the polytope $Q(x)$ as intersection of $N = \sum_{l=1}^{L} n_l$ half spaces given by

$$\Gamma_{l,i} = \big\{ z \in \mathbb{R}^d \,\big|\, \Delta^{(l)}(x)_{ii}\big(V_i^{(l)} z + a_i^{(l)}\big) \geq 0 \big\}$$

for $l = 1, \dots, L$, $i = 1, \dots, n_l$, namely

$$Q(x) = \bigcap_{l=1,\dots,L} \bigcap_{i=1,\dots,n_l} \Gamma_{l,i}.$$

Note that N is also the number of hidden units of the network. Finally, we can write

$$f^{(L+1)}(z)\Big|_{Q(x)} = V^{(L+1)} z + a^{(L+1)},$$

which represents the affine restriction of f to $Q(x)$. One can further distinguish the subset $Q_c(x)$ of $Q(x)$ assigned to a specific class c, among the K available ones, which is given by

$$Q_c(x) = \bigcap_{\substack{s=1,\dots,K \\ s \neq c}} \big\{ z \in \mathbb{R}^d \,\big|\, \Big\langle V_c^{(L+1)} - V_s^{(L+1)}, z \Big\rangle + a_c^{(L+1)} - a_s^{(L+1)} \geq 0 \big\} \cap Q(x),$$

where $V_r^{(L+1)}$ is the r-th row of $V^{(L+1)}$. The set $Q_c(x)$ is again a polytope as it is an intersection of polytopes and it holds $Q(x) = \bigcup_{c=1,\dots,K} Q_c(x)$.

3.1.1 Extension to other types of layer

Modern convolutional architectures use several types of layers, but most of them parameterize affine functions. The main exception is constituted by max-pooling which instead gives a piecewise affine functions. However, even when these layers appear, we can derive a characterization of the network similar to the one above.

Skip connections and normalization layers. Both types layers are already affine operations (with BatchNorm [Ioffe and Szegedy, 2015] used in evaluation mode) and do not affect the polytope.

Pooling layers. Average pooling computes the mean over certain subsets of the input vector. For example, the average of the first four entries of a vector $z \in \mathbb{R}^{n_{l-1}}$ is obtained, introducing

$$w = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, \ldots, 0\right) \in \mathbb{R}^{n_{l-1}},$$

as $\langle w, z\rangle$. In general, each output unit of a pooling layer can be obtained as the inner product of a vector w_i, with entries equal to $1/p$ in the positions of the p elements we want to average and zero else, and the layer input. If we use the vectors w_i as rows of the matrix $A^{(l)}$, then the avg-pooling layer at position l can be written as $f^{(l)}(z) = A^{(l)}z$. We notice that $A^{(l)}$ does not depend on the input x and is already an affine function. The construction of $A^{(l)}$ for max-pooling layers is analogue (as these layers return the maximum instead of the mean). The main difference is that in this case $A^{(l)}$ may change as z does. In fact, if we want to extract the maximum among the first four entries of z and assume that it is realized by the second component, we can set $a = (0, 1, 0, 0, \ldots, 0) \in \mathbb{R}^{n_l}$. If the position of the maximum changes also the vector a changes. Again, $\langle a, z\rangle$ returns the value we are interested in. If $p_1, \ldots, p_{n_l}$ are the positions of the maxima for each of the n_l pools, we can then build $A^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ as

$$A^{(l)}_{rs} = \begin{cases} 1 & \text{if } s = p_r \\ 0 & \text{else} \end{cases}, \quad s = 1, \ldots, n_{l-1}, \quad r = 1, \ldots, n_l,$$

so that $f^{(l)}(z) = A^{(l)}z$. When $f^{(l)}$ is a max-pooling layer, the local linear representation $A^{(l)}$ is preserved as long as the maximum within each pool is realized at the same position. We can denote the n_l pools as the sets $P^1, \ldots, P^{n_l}$, whose elements are the indices of the components of the input (of the max-pooling layer) involved in the pool. Moreover we define for every $r = 1, \ldots, n_l$

$$p^r_{max} = \arg\max_{i \in P^r} x^{(l-1)}_i,$$

that is the index of the component of the input $x^{(l-1)} \in \mathbb{R}^{n_{l-1}}$ of the max-pooling layer attaining the maximum for each pool P^r. Then, for $r = 1, \ldots, n_l$,

$$S^{(l)}_r(x^{(j-1)}) = \{z \in \mathbb{R}^{n_{l-1}} \mid z_{p^r_{max}} \geq z_i, \ \forall i \in P^r\}$$

is the subset of $\mathbb{R}^{n_{l-1}}$ preserving the position of the maximum computed at $x^{(j-1)}$ for pool P^r. Similar to what has been done for ReLU layers above, we define the polytopes where the r-th max-pooling unit can be replaced by a linear function as

$$\Gamma_{l,r} = \left\{z \in \mathbb{R}^d \mid \left(V^{(l-1)}_{p^r_{max}} - V^{(l-1)}_i\right) z + a^{(l-1)}_{p^r_{max}} - a^{(l-1)}_i \geq 0, \ \forall i \in P^r\right\}, \tag{3.1}$$

with $V^{(l-1)}$ and $a^{(l-1)}$ as introduced above. Finally, $\Gamma_l = \bigcap_{r=1}^{n_l} \Gamma_{l,i}$ is the subset of the input space containing x on which $f^{(l)}$ is an affine function.

3.2 Provable robustness in a single threat model

In the following we first derive robustness guarantees for ReLU networks (Sec. 3.2.1), then we introduce a regularization scheme to obtain classifiers with high provable robustness (Sec. 3.2.2). Finally, we test the proposed method in experiments on several datasets (Sec. 3.2.4).

3.2.1 Robustness guarantees for ReLU networks

We call a decision of a classifier f robust at x if small changes of the input do not alter the decision. Formally, this can be described as optimization problem (3.2) [Szegedy et al., 2014].

If the classifier outputs class c for input x, assuming a unique decision, the *robustness* wrt the l_p-norm of f at x is given by

$$\mathbf{r}_p(x) = \min_{\delta \in \mathbb{R}^d} \|\delta\|_p, \quad \text{s.th.} \quad \max_{l \neq c} f_l(x+\delta) \geq f_c(x+\delta), \quad x+\delta \in C \tag{3.2}$$

where C is a constraint set which the generated point $x+\delta$ has to satisfy, e.g., an image has to be in $[0,1]^d$. The complexity of the optimization problem (3.2) depends on the classifier f, but it is typically non-convex, see Katz et al. [2017] for a hardness result for neural networks. For standard neural networks $\|\delta\|_p$ is very small for *almost any* input x of the data generating distribution, which questions the use of such classifiers in safety-critical systems.

For a linear classifier, $f(x) = Wx + b$, with $C = \mathbb{R}^d$ one can compute the solution of (3.2) in closed form [Huang et al., 2016]

$$\|\delta\|_p = \min_{s \neq c} \frac{|\langle W_c - W_s, x \rangle + b_c - b_s|}{\|W_c - W_s\|_q},$$

where $\|\cdot\|_q$ is the dual norm of $\|\cdot\|_p$, that is $\frac{1}{p} + \frac{1}{q} = 1$. In Hein and Andriushchenko [2017] it has been shown that box constraints $C = [a,b]^d$ can be integrated for linear classifiers which results in simple convex optimization problems. In the following we use the intuition from linear classifiers and the particular structures derived in Sec. 3.1 to come up with robustness guarantees, that means lower bounds on the optimal solution of (3.2), for ReLU networks. Moreover, we show that it is possible to compute the minimal perturbation for some of the inputs x even though the general problem is NP-hard [Katz et al., 2017].

Let us start analyzing how we can solve efficiently problem (3.2) inside each linear region $Q(x)$. We first need the definition of two important quantities.

Lemma 3.2.1 *The l_p-distance $d_p^B(x) = \min_{z \in \partial Q(x)} \|z - x\|_p$ of x to the boundary of the polytope $Q(x)$ is given by*

$$d_{p,l,j}^B = \frac{|\left\langle V_j^{(l)}, x \right\rangle + a_j^{(l)}|}{\left\|V_j^{(l)}\right\|_q}, \quad d_p^B = \min_{l=1,\ldots,L} \min_{j=1,\ldots,n_l} d_{p,l,j}^B, \tag{3.3}$$

where $V_j^{(l)}$ is the j-th row of $V^{(l)}$ and $\|\cdot\|_q$ is the dual norm of $\|\cdot\|_p$ ($\frac{1}{p} + \frac{1}{q} = 1$).

Proof. Due to the polytope structure of $Q(x)$ it holds that $d_p^B(x)$ is the minimum distance to the hyperplanes, $\left\langle V_j^{(l)}, z \right\rangle + a_j^{(l)} = 0$, which constitute the boundary of $Q(x)$. It is straightforward to check that the minimum l_p-distance of a hyperplane $H = \{z \mid \langle w, z \rangle + b = 0\}$ to x is given by $\frac{|\langle w,x \rangle + b|}{\|w\|_q}$ where $\|\cdot\|_q$ is the dual norm. This can be obtained as follows

$$\min_{z \in \mathbb{R}^d} \left\{ \|x - z\|_p \mid \langle w, z \rangle + b = 0 \right\} \tag{3.4}$$

Introducing $\delta = z - x$ we get

$$\min_{\delta \in \mathbb{R}^d} \left\{ \|\delta\|_p \mid \langle w, \delta \rangle + \langle w, x \rangle + b = 0 \right\} \tag{3.5}$$

Note that by Hölder inequality one has $-\|w\|_q \|\delta\|_p \leq \langle w, \delta \rangle \leq \|w\|_q \|\delta\|_p$, which yields $\|\delta\|_p \geq \frac{|\langle w,\delta \rangle|}{\|w\|_p}$. Noting that $\langle w, \delta \rangle = -(\langle w, x \rangle + b)$ we get $\|\delta\|_p \geq \frac{|\langle w,x \rangle + b|}{\|w\|_p}$ and equality is achieved when one has equality in the Hölder inequality. □

For the decision boundaries in $Q(x)$, with $c = \arg\max_{r=1,\ldots,K} f_r^{(L+1)}(x)$, we define the quantities

$$d_{p,s}^D(x) = \frac{\left|\left\langle V_c^{(L+1)} - V_s^{(L+1)}, x \right\rangle + a_c^{(L+1)} - a_s^{(L+1)}\right|}{\left\|V_c^{(L+1)} - V_s^{(L+1)}\right\|_q} \quad \text{and} \quad d_p^D(x) = \min_{\substack{s=1,\ldots,K \\ s \neq c}} d_{p,l,j}^D(x).$$

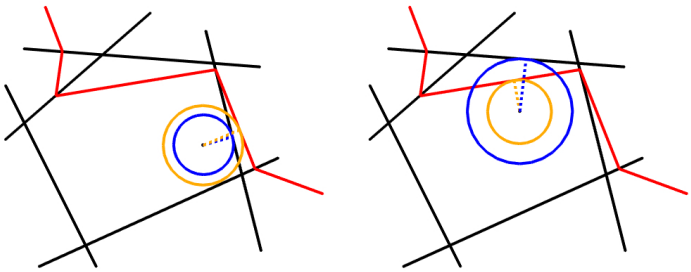

Figure 3.1: **Left**: the input x is closer to the boundary of the polytope $Q(x)$ (black) than to the decision boundary (red). In this case the smallest perturbation that leads to a change of the decision lies outside the linear region $Q(x)$. **Right**: the input x is closer to the decision boundary than to the boundary of $Q(x)$, so that the projection of the point onto the decision hyperplane provides the adversarial example with the smallest norm.

Lemma 3.2.2 *If $d_p^D(x) \leq d_p^B(x)$, then $d_p^D(x)$ is the minimal l_p-distance of x to the decision boundary of the ReLU network $f^{(L+1)}$.*

Proof. First we note that $d_p^D(x)$ is the distance of x to the decision boundary for the linear multi-class classifier $h(x) = V^{(L+1)}x + a^{(L+1)}$ which is equal to $f^{(L+1)}$ on $Q(x)$. If $d_p^D(x) \leq d_p^B(x)$ then the point realizing the minimal distance to the decision boundary $d_p^D(x)$ is inside $Q(x)$ and as $d_p^D(x) \leq d_p^B(x)$ there cannot exist another point on the decision boundary of $f^{(L+1)}$ outside of $Q(x)$ having a smaller l_p-distance to x. Thus $d_p^D(x)$ is the minimal l_p-distance of x to the decision boundary of $f^{(L+1)}$. □

The next theorem combines both results to give a lower bound or the optimal solution $\mathbf{r}_p(x)$ to the optimization problem (3.2).

Theorem 3.2.1 *We get the following robustness guarantees:*

1. *If $d_p^B(x) < d_p^D(x)$, then $d_p^B(x)$ is a lower bound on the minimal l_p-norm of the perturbation necessary to change the class (optimal solution of (3.2)).*
2. *If $d_p^D(x) \leq d_p^B(x)$, then $d_p^D(x)$ is equal to the minimal l_p-norm necessary to change the class (optimal solution of (3.2)).*

Proof. If $d_p^B(x) < d_p^D(x)$, then the ReLU classifier does not change on $B_p(x, d_p^B(x))$ and thus $d_p^B(x)$ is a lower bound on the minimal l_p-norm perturbation necessary to change the class. The second statement follows directly by Lemma 3.2.2. □

In Fig. 3.1 we illustrate the different cases for $p = 2$. On the left hand side $d_p^B(x) < d_p^D(x)$ and thus we get that on the ball $B_2(x, d_B(x))$ the decision does not change, whereas in the rightmost plot we have $d_p^D(x) < d_p^B(x)$ and thus we obtain the minimum distance to the decision boundary. Using Theorem 3.2.1 we can provide robustness guarantees for every point and for some even compute the optimal robustness guarantees. We describe in Sec. 3.2.3 how one can improve the lower bounds by checking neighboring regions of $Q(x)$. Compared to the bounds of Wong and Kolter [2018], Weng et al. [2018] ours are slightly worse (see Table 3.6). However, our bounds can be directly maximized and have a clear geometrical meaning and thus motivate directly a regularization scheme for ReLU networks which we propose in the next section.

3.2.2 Large margin and region boundary regularization

Using the results of Sec. 3.2.1, a classifier with guaranteed robustness requires large distance to the boundaries of the linear region as well as to the decision boundary. Even the optimal guarantee (solution of (3.2)) can be obtained in some cases. Unfortunately, as illustrated in Fig. 3.2 for simple one hidden layer networks, the linear regions $Q(x)$ are small for normally-trained networks and thus no meaningful guarantees can be obtained (first and third plots).

Thus we propose a new regularization scheme which simultaneously aims for each training point to achieve large distance to the boundary of the linear region it lies in, as well as to the decision boundary. Using Theorem 3.2.1 this directly leads to non-trivial robustness guarantees of the resulting classifier.

However, note that just maximizing the distance to the decision boundary might be misleading during training as this does not discriminate between points which are correctly (correct side of the decision hyperplane) or wrongly classified (wrong side of the decision hyperplane). Thus we introduce the signed version of $d_p^D(x)$, where y is the true label of the point x, via

$$\overline{d_{p,s}^D}(x) = \frac{f_y^{(L+1)}(x) - f_s^{(L+1)}(x)}{\left\| V_y^{(L+1)} - V_s^{(L+1)} \right\|_q} \quad \text{and} \quad \overline{d_p^D}(x) = \min_{\substack{s=1,\dots,K \\ s \neq y}} \overline{d_{p,s}^D}(x) \tag{3.6}$$

Please note that if $\overline{d_p^D}(x) \geq 0$, then x is correctly classified, whereas $\overline{d_p^D}(x) < 0$ if x is wrongly classified. If $|\overline{d_p^D}(x)| \leq d_p^B(x)$, then it follows from Lemma 3.2.2 that $|\overline{d_p^D}(x)|$ is the distance to the decision hyperplane. If $|\overline{d_p^D}(x)| > d_p^B(x)$ this does not need to be any longer true, but $|\overline{d_p^D}(x)|$ is at least a good proxy as $\left\| V_y^{(L+1)} - V_s^{(L+1)} \right\|_q$ is an estimate of the local cross Lipschitz constant [Hein and Andriushchenko, 2017]. Finally, we propose to use the following regularization scheme:

Definition 3.2.1 (MMR) *Let x be a point of the training set. We define the Maximum Margin Regularizer (MMR) for x, for some $\gamma_B, \gamma_D \in \mathbb{R}_{++}$, as*

$$\textit{MMR-}l_p(x) = \max\left(0, 1 - \frac{d_p^B(x)}{\gamma_B}\right) + \max\left(0, 1 - \frac{\overline{d_p^D}(x)}{\gamma_D}\right). \tag{3.7}$$

The MMR penalizes distances to the boundary of the polytope if $d_p^B(x) \leq \gamma_B$ and positive distances (x is correctly classified) if $d_p^D(x) \leq \gamma_D$. Notice that wrongly classified points are always penalized. The part of the regularizer corresponding to $d_p^D(x)$ has been suggested in Elsayed et al. [2018] in a similar form as a loss function for general neural networks without motivation from robustness guarantees. They have an additional loss penalizing difference in the outputs with respect to changes at the hidden layers which is completely different from our geometrically motivated regularizer penalizing the distance to the boundary of the linear region. The choice of γ_B, γ_D allows different trade-off between the terms. In particular $\gamma_D < \gamma_B$ (stronger maximization of $d_p^B(x)$) leads to more points for which the optimal robustness guarantee (case $d_p^D(x) \leq d_p^B(x)$) can be proved.

For practical reasons, sorting in increasing order $d_{p,l,j}^B$ and $\overline{d_{p,s}^D}$, that is the l_p-distances to the hyperplanes defining $Q(x)$ and to decision hyperplanes, and denoting them as $d_{p,\pi_i^B}^B$ and $\overline{d_{p,\pi_i^D}^D}$ respectively, we also propose a variation of our MMR regularizer in (3.7):

$$\text{kMMR-}l_p(x) = \frac{1}{k_B}\sum_{i=1}^{k_B} \max\left(0, 1 - \frac{d_{p,\pi_i^B}^B(x)}{\gamma_B}\right) + \frac{1}{k_D}\sum_{i=1}^{k_D} \max\left(0, 1 - \frac{\overline{d_{p,\pi_i^D}^D}(x)}{\gamma_D}\right), \tag{3.8}$$

with $k_B, k_D \leq 1$. Basically, we are optimizing, instead of the closest decision hyperplane, the k_D-closest ones and analogously the k_B-closest hyperplanes defining the linear region $Q(x)$ of x. This speeds up the training time as more hyperplanes are moved in each update. Moreover, when deriving lower bounds using more than one linear region, one needs to consider more than just the closest boundary hyperplane. Finally, many state-of-the-art schemes for proving lower bounds [Tjeng et al., 2019, Wong and Kolter, 2018, Weng et al., 2018] work well only if the activation status of most neurons is constant for small changes of the points. This basically amounts to ensure that all the hyperplanes are sufficiently far away, which is exactly what our regularization scheme is aiming at. Thus our regularization scheme also helps to achieve state-of-the-art provable robustness with other certification methods [Tjeng et al., 2019, Wong and Kolter, 2018]. This is also the reason why the term pushing the polytope boundaries away is essential. Just penalizing the distance to the decision boundary is not sufficient to prove good lower bounds

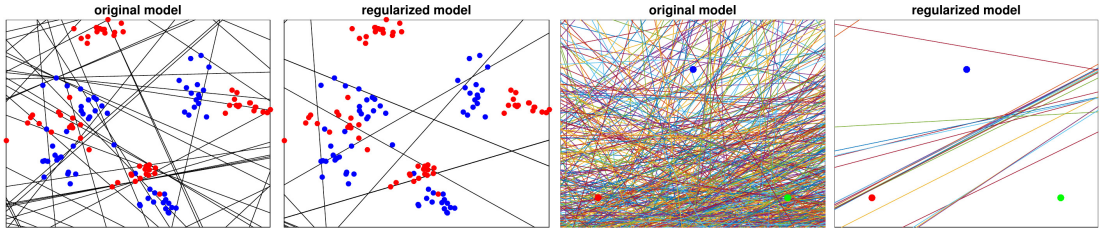

Figure 3.2: **The effects of MMR**. *Fist and second plots*: we train two networks with one hidden layer, 100 units, on 128 points belonging to two classes (red and blue). The first plot shows the points and how the input space is divided in regions on which the classifier is linear. The second one is the analogue for our MMR regularized model. *Third and fourth plots*: we show region boundaries (one hidden layer, 1024 units) on a 2D slice of $\mathbb{R}^{784}$ spanned by three random points from different classes of the MNIST training set. We observe a clear maximization of the linear regions for the MMR-regularized case (right) versus the non-regularized case (left).

on the minimal distance to the decision boundary as we will show in our experiments (Table 3.4). Compared to the regularization scheme in Wong and Kolter [2018] using a dual feasible point of the robust loss, our approach has a direct geometric interpretation and allows us to derive the exact minimal perturbation for some fraction of the test points (it varies from dataset to dataset but it can be as high as 99%). In practice, we gradually decrease k_B and k_D in (3.8) during training so that only the closest hyperplanes of each training point influence the regularizer.

Denoting the cross entropy loss $\mathrm{CE}(f(x), y)$, the final objective of our models is

$$\frac{1}{n}\sum_{i=1}^{n} \mathrm{CE}(f(x_i), y_i) + \lambda \, \text{kMMR-}l_p(x_i), \tag{3.9}$$

where $(x_i, y_i)_{i=1}^n$ is the training data and $\lambda \in \mathbb{R}_+$ the regularization parameter. Fig. 3.2 shows the effect of the regularizer: compared to the unregularized case the size of the linear regions is significantly increased.

3.2.3 Improving lower bounds

Integration of box constraints into robustness guarantees. Note that in many applications the input space is not full $\mathbb{R}^d$ but a certain subset C due to constraints e.g. images belong to $C = [0,1]^d$. These constraints typically increase the norm of the solution of (3.2). Thus it is important to integrate the constraints in the generation of adversarial samples (providing upper bounds on the size of the minimal perturbation) as done e.g. in Carlini and Wagner [2017a], as well as in the computation of the lower bounds, which is in our case based on the computation of distances to hyperplanes. The computation of the l_p-distance of y to a hyperplane (w, b) has a closed form solution:

$$\min\{\|y - x\|_p \mid \langle w, x\rangle + b = 0\} = \frac{|\langle w, y\rangle + b|}{\|w\|_q}, \tag{3.10}$$

The additional box constraints lead to the following optimization problem,

$$\min\{\|y - x\|_p \mid \langle w, x\rangle + b = 0, \quad x \in [0,1]^d\}, \tag{3.11}$$

which is convex but has no analytical solution. However, its dual is just a one-dimensional convex optimization problem which can be solved efficiently. In fact a reformulation of this problem has been considered in Hein and Andriushchenko [2017], where fast solvers for $p \in \{1, 2, \infty\}$ are proposed. Moreover, when computing the distance to the boundary of the polytope or the decision boundaries one does not need to solve always the box-constrained distance problem (3.11). It suffices to compute first the distances (3.10) as they are smaller or equal to the ones of (3.11) and sort them in ascending order. Then one computes the box-constrained distances in the given order and stops when the smallest computed box-constrained distance is smaller than the next original distance in the sorted list. In this way one typically just needs to solve a very small fraction of all box-constrained problems. The integration of the box constraints

is important as the lower bounds improve on average by 20% and this can make the difference between having a certified optimal solution and just a lower bound.

Checking neighboring regions. In order to improve the lower bounds we can use not only the linear region $Q(x)$ where x lies but also some neighboring regions. The following description is just a sketch as one has to handle several case distinctions. Let x be the original point and $H = \{\pi_1, ..., \pi_n\}$ the set of hyperplanes defining the polytope $Q(x)$ sorted so that $d_C(x, \pi_i) < d_C(x, \pi_j)$ if $i < j$, where d_C is the distance including box constraints. If we do not directly get the guaranteed optimal solution, we get an upper bound (u, namely the distance to the decision boundary inside $Q(x)$) and a lower bound for the norm of the adversarial perturbation ($l = d_C(x, \pi_1)$). If $l < u$, we can check the region that we find on the other side of π_1. In order to get the corresponding description of the polytope on the other side, we just have to change the corresponding entry in the activation matrix Σ of the layer where π_1 belongs to and recompute the hyperplanes of the new linear region R. Solving (3.2) on the second region we get a new upper bound if the distance of x to the decision boundary in R is smaller than u. Moreover we update H with the hyperplanes given by the second region. Finally, if $u < d_C(x, \pi_2)$ then u is the optimal solution, otherwise $l = d_C(x, \pi_2)$ and we can repeat this process with the next closest hyperplane. At the moment we stop after checking maximally 5 neighboring linear regions.

3.2.4 Experiments

We provide a variety of experiments aiming to show the ability of MMR to achieve provably robust classifiers. We make our code and models publicly available.[1] We use four datasets: MNIST [LeCun et al., 2010], German Traffic Signs (GTS) [Stallkamp et al., 2012], Fashion MNIST [Xiao et al., 2017] and CIFAR-10 [Krizhevsky, 2012]. We consider in the paper robustness wrt l_2 and l_∞ distances, and measure it with upper and lower bounds on the robust test error for a given threshold ϵ, that is the largest achievable test error if every test input can be modified with a perturbation δ such that $\|\delta\|_p \leq \epsilon$ and δ is chosen so that $x + \delta$ is classified wrongly. Moreover, we show in Sec. 3.2.5 results for lower and upper bounds on the minimal $\|\delta\|_p$ from (3.2).

Upper bounds on the robust test error are computed using the approach of Wong and Kolter [2018]. Additionally, the method of Tjeng et al. [2019] yields upper and lower bounds on the robust test error via mixed integer programming (MIP). We use the solver for the MIP with a timeout of 120s per point, that is if the point has not been verified in this time the solver is stopped. This technique is currently effective for l_∞ but not for l_2, where basically in almost every case the timeout is reached and thus we discard for l_2 the MIP evaluation. Finally, we combine the upper bounds on the robust test error found by the two methods by counting the fraction of points that can be certified with at least one of them. We compare our own guarantees from Theorem 3.2.1 with the ones of Wong and Kolter [2018] in Sec. 3.2.5. Lower bounds on the robust test error are computed using the attack via Projected Gradient Descent (PGD) [Madry et al., 2018], MIP and Linear Region Attack (see Sec. 4).

We compare six training scheme: plain training, adversarial training (AT) of Madry et al. [2018] which has been shown to significantly increase robustness, the robust loss of Wong and Kolter [2018] which supports both l_2 and l_∞ norms (denoted as KW), the method of Xiao et al. [2019], our regularization scheme MMR and MMR together with adversarial training. All schemes are evaluated on a one hidden layer fully connected network with 1024 hidden units (FC1) and a convolutional network (CNN) with 2 convolutional and 2 dense layers as used in Wong and Kolter [2018]. For more details see Sec. 3.2.6. Since code for Xiao et al. [2019] is not available, we just show the l_∞ results from their paper (evaluated on full test sets with PGD lower bounds, and MIP upper bounds), which are available only for MNIST and CIFAR-10 for the same CNN architecture.

Improvement of robustness. The results can be found in Table 3.1. For CIFAR-10 we show only CNN models as fully connected networks do not have good test performance. We report clean test error, lower and upper bounds on robust test error at the threshold indicated in Table 3.1, computed on 1000 points for l_∞, and on the full test sets for l_2 (as we do not do the MIP

[1] https://github.com/max-andr/provable-robustness-max-linear-regions

Table 3.1: **Comparison of different methods regarding robustness.** We here report the statistics of 6 training schemes: plain (usual training), AT (adversarial training Madry et al. [2018]), KW Wong and Kolter [2018], Xiao et al. [2019], MMR (ours), and MMR+AT (MMR plus adversarial training). We show, in percentage, test error (TE), lower (LB) and upper (UB) bounds on the robust test error at the threshold ϵ indicated for each dataset. The robustness statistics are computed on the first 1000 points of the respective test sets for l_∞-robustness, on the full test set for l_2-robustness. KW models marked with * are taken from Wong and Kolter [2018], Wong et al. [2018], while the other are retrained according to the available code (but the bounds are again evaluated with the discussed combination of multiple techniques). The results and bounds of Xiao et al. [2019] are taken directly from the original paper. The model indicated by [1] has been obtained at epoch 50 instead of 100 due to optimization issues. We do not include FC1 on CIFAR-10 since even for plain training it yields poor performance.

	l_∞-norm robustness						**l_2-norm robustness**					
training scheme	FC1			CNN			FC1			CNN		
	TE	LB	UB	TE	LB	UB	TE	LB	UB	TE	LB	UB
MNIST	$\epsilon = 0.1$						$\epsilon = 0.3$					
plain	1.44	93.0	100	0.91	74.0	100	1.73	9.7	66.3	0.85	3.1	100
at	0.92	10.0	99.0	0.82	3.0	100	1.15	2.6	16.9	0.87	1.8	100
KW	3.19	10.9	10.9	1.26*	4.4	4.4	1.19	2.4	5.2	1.11	2.2	6.0
Xiao et al	-	-	-	1.32	4.9	5.7	-	-	-	-	-	-
MMR	2.11	22.5	24.9	1.65	6.0	6.0	2.40	5.9	8.8	2.57	5.8	11.6
MMR+AT	2.04	14.0	14.1	1.19	3.6	3.6	1.77	3.8	6.4	2.12[1]	4.6	9.7
F-MNIST	$\epsilon = 0.1$						$\epsilon = 0.3$					
plain	9.49	100	100	9.50	96.0	100	9.70	42.8	91.8	9.32	57.1	100
at	11.69	29.5	95.5	11.54	21.5	73.0	9.15	19.9	61.4	8.10	20.4	100
KW	21.31	32.8	32.8	21.73*	32.4	32.4	11.24	17.2	22.7	13.08	18.5	21.7
MMR	18.11	37.6	42.0	14.52	33.2	33.6	13.28	25.0	28.0	12.85	25.4	35.3
MMR+AT	15.84	31.3	34.8	14.50	26.6	30.7	12.12	19.7	23.4	13.42	26.2	39.1
GTS	$\epsilon = 4/255$						$\epsilon = 0.2$					
plain	12.72	61.0	77.0	7.11	63.0	99.5	12.24	37.5	44.7	6.78	33.3	99.2
at	9.33	34.5	48.5	6.84	29.5	81.0	13.55	33.1	43.2	8.75	23.8	98.6
KW	13.99	33.0	33.0	15.56	36.1	36.6	16.84	16.9	31.2	14.33	28.7	34.7
MMR	14.29	39.8	39.8	13.31	49.5	49.6	14.55	33.2	34.7	14.22	36.2	36.9
MMR+AT	13.10	33.1	35.4	14.88	38.3	38.4	13.94	29.7	32.1	15.35	32.1	33.2
CIFAR-10	$\epsilon = 2/255$						$\epsilon = 0.1$					
plain	-	-	-	24.63	91.0	100	-	-	-	23.31	47.2	100
at	-	-	-	27.04	52.5	88.5	-	-	-	25.82	35.8	100
KW	-	-	-	38.91*	46.6	48.0	-	-	-	40.24	43.9	49.0
Xiao et al	-	-	-	38.88	50.1	54.1	-	-	-	-	-	-
MMR	-	-	-	34.61	57.5	61.0	-	-	-	40.92	50.6	57.1
MMR+AT	-	-	-	35.38	47.9	54.2	-	-	-	37.75	43.9	53.3

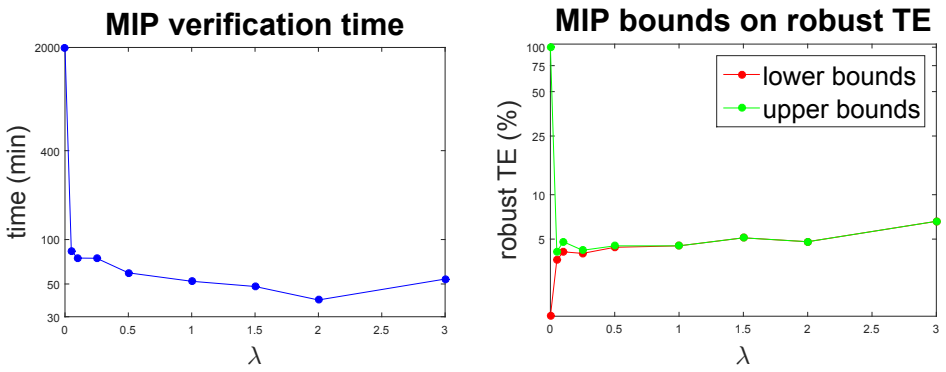

Figure 3.3: **Verifiability of models.** We show the runtime (left) in minutes that MIP [Tjeng et al., 2019] takes to verify 1000 points, setting a timeout of 120s (that is the mixed-integer optimization stops anyway after the time limit is reached), with models trained with different values of λ (see Eq. (3.9)). Note that a logarithmic scale is used on the y-axis. Moreover, we report (right) lower (red) and upper (green) bounds on robust test error (TE). The plain model, trained without MMR ($\lambda = 0$) needs 51 times more to be verified, with only 1% of the points certified. Conversely, even with a light MMR regularization lower and upper bounds are tight.

evaluation there). Both KW and MMR+AT achieve similar performance regarding lower and upper bounds on the robust test error. For l_∞ our MMR+AT achieves overall better performance than KW, sometimes with significantly better clean test error like on F-MNIST. In some cases, e.g. on CIFAR-10, MMR+AT provides slightly worse upper bounds, but instead preserves better test error. For l_2 KW performs better than MMR+AT with the exception of GTS. This is to be expected as we use for the evaluation of the upper bounds on the robust test error the approach of Wong and Kolter [2018] which are directly optimized by their robust training procedure. Note that both KW and MMR/MMR+AT outperform by large margin plain and adversarial training regarding provable robustness (upper bounds on the robust test error). With the exception of the CNN-l_2 models for MNIST and F-MNIST, the upper bound on robust test error of MMR+AT is smaller than the lower bound of the plain model. Thus our models are provably better than the plain models regarding robust test error. Moreover, regarding robustness wrt l_∞ the gaps between lower and upper bounds are often very small for KW and MMR/MMR+AT showing that both techniques lead to models which are easier to check via the MIP which we discuss next in more detail.

Enhancing verifiability. A key aspect of any robust training should be the ability of producing models both resistant to adversarial manipulation and being *verifiable*, in the sense of having guarantees on the minimal perturbation changing the decision for a test input. In fact, even if empirically model seems to be robust, only computing certificates allows to completely trust it. In our experiments, we could use successfully the method of Wong and Kolter [2018] to get meaningful guarantees, which so far, as noted in Raghunathan et al. [2018], could be achieved only on models provided by the specific training of Wong and Kolter [2018]. Moreover, the MIP is too slow to run on standard models, so that ad hoc techniques have been developed to train classifiers verifiable by MIP. Our MMR training produces models which can be checked by MIP. This is due to the fact that the hyperplanes representing the boundary of the polytope are actually the boundary between the different regimes (identity or zero function) of ReLU units, so that pushing the hyperplanes implies that many inputs of ReLU units have constant sign in a wide region around the data points (and having ReLU units with unstable signs is the main slowdown factor for the MIP solver). In this context, Xiao et al. [2019] develop a specific technique aiming at inducing stability of ReLU signs. Therefore, we provide a comparison of MMR to the ReLU-stability loss of Xiao et al. [2019] in Table 3.1. On MNIST our MMR+AT model of the same CNN architecture has much tighter upper bounds on the robust error (3.6% compared to 5.7%). Moreover, we have no gap between the lower and upper bounds on the robust test error, and our upper bounds are *lower* than the lower bounds of Xiao et al. [2019]. This suggests that our training scheme is better at both improving robustness and enhancing verifiability than the approach of Xiao et al. [2019]. Additionally, on CIFAR-10 we have similar upper bounds,

Table 3.2: **Comparison of different methods regarding robustness wrt l_2-norm.** We here report the statistics of the 5 different training schemes: test error (TE), average of lower (LB) and upper (UB) bounds on the robustness $\|\delta\|_2$, where δ is the solution of (3.2). The robustness statistics are computed on the first 1000 points of the respective test sets (including misclassified images) against all the possible target classes.

dataset	*training scheme*	FC1			FC10			CNN		
		TE(%)	LB	UB	TE(%)	LB	UB	TE(%)	LB	UB
MNIST	plain	1.59	0.34	0.98	1.81	0.13	0.70	0.97	0.04	1.03
	at	1.29	0.25	1.23	0.93	0.14	1.59	0.86	0.14	1.67
	KW	1.37	**0.70**	**1.75**	1.69	**0.75**	**1.74**	1.04	0.32	1.84
	MMR	1.51	0.69	1.69	1.87	0.48	1.48	1.17	**0.38**	1.70
	MMR+AT	1.59	**0.70**	1.70	1.35	0.40	1.60	1.14	**0.38**	**1.86**
GTS	plain	12.24	0.33	0.57	11.25	0.08	0.48	6.73	0.06	0.43
	at	13.55	0.34	0.66	13.01	0.10	0.56	8.12	0.06	0.53
	KW	13.06	0.35	0.63	13.56	0.16	0.52	8.44	**0.11**	0.52
	MMR	11.15	0.69	0.69	12.82	**0.64**	**0.67**	7.40	0.09	0.59
	MMR+AT	11.72	**0.72**	**0.72**	13.36	**0.64**	0.66	10.50	**0.11**	**0.62**
F-MNIST	plain	9.61	0.18	0.53	10.53	0.05	0.44	8.86	0.03	0.32
	at	9.89	0.11	1.00	9.89	0.11	1.00	8.77	0.07	0.80
	KW	9.95	0.46	1.11	11.42	0.47	1.22	10.37	0.17	0.96
	MMR	10.22	0.50	0.85	11.73	**0.68**	1.18	10.30	0.17	0.88
	MMR+AT	10.94	**0.66**	**1.45**	11.39	0.67	**1.24**	10.48	**0.21**	**1.14**

but better test error and better lower bounds. To illustrate the speed-up in verification time for MMR models, we show in Fig. 3.3 the time MIP needs to run on 1000 points (with timeout of 120s per point) and lower and upper bounds on robust test error wrt l_∞-distance at $\epsilon = 0.1$. We use CNNs trained on MNIST with $\gamma_B = \gamma_D = 0.15$ and different values of λ representing the weight of our regularizer in the loss (3.9) (for $\lambda = 0$ one gets the plain model). It is clear that MIP performs poorly both in runtime (almost 2000 minutes) and performance (LB=1%, UB=100%) on the plain model. In contrast, the MMR models are verified quickly (between 35 and 79 minutes) and almost completely (the rate of certified points is between 99.3% and 100%), that is lower and upper bounds are close or even equal. Please note that both statistics improve with increasing λ up to 2, while runtime gets worse with a larger value (as at $\lambda = 3$ the classifier becomes less robust and then requires a higher computational effort to be certified).

3.2.5 Additional results

We present a series of experiments for a detailed understanding of how MMR works. For this section we consider models trained to be l_2-robust and evaluate robustness as the average l_2-norm of the perturbations necessary to change the class. Lower bounds on this perturbation are computed either as in Wong and Kolter [2018] or with our method (Theorem 3.2.1), while upper bounds are provided by Carlini-Wagner l_2-attack (CW) [Carlini and Wagner, 2017a]. We also introduce a second fully connected architecture, FC10, with 10 hidden layers (see Sec. 3.2.6 for details).

Analysing l_2-robustness with different metrics. We want here to repeat the experiment of Sec. 3.2.4 wrt l_2-norm but evaluating robustness as the average norm of the perturbation necessary to change the classification of a point. When our technique (Theorem 3.2.1) provides the optimal solution, we set both lower and upper bounds to this value. We report test error of the model and the average lower and upper bounds in Tables 3.2 and 3.3, computed on 1000 points of the test set. For KW, MMR and MMR + adversarial training we report the solutions which achieve similar test error than the plain model. There are several interesting observations. First of all, while adversarial training improves the upper bounds compared to the plain setting often quite significantly, the lower bounds almost never improve, often they get even worse. This is in contrast to the methods, KW and our MMR, which optimize the robustness guarantees. For MMR we see in all cases significant improvements of the lower bounds over plain and adversarial

Table 3.3: **Comparison of different training schemes wrt l_2-norm on CIFAR-10.** For each model, we show test error, average of lower and upper bounds on $\|\delta\|_2$, where δ is the solution of problem (3.2). The statistics are computed on the first 1000 points of the test set (including misclassified images) against all the possible target classes.

training scheme	CNN		
	TE(%)	LB	UB
plain	25.98	0.02	0.16
at	25.36	0.04	0.42
KW	41.52	**0.16**	**0.66**
MMR	41.86	**0.16**	0.39
MMR+AT	41.11	0.13	0.57

Table 3.4: **Full version of MMR is necessary.** We compare the statistics of models trained with the full version of MMR as in (3.7) and (3.8) (MMR-*full*) and with only the second part penalizing the distance to the decision boundary (MMR-d_2^D). While the test error is for the full test set, the lower and upper bounds on the l_2-norm of the optimal adversarial manipulation are compared on the first 1000 points of the test set. One can clearly see that the lower bounds improve significantly when one uses the full MMR regularization.

dataset	*model*	MMR-*full*			MMR-d_2^D		
			average $\mathbf{r}_2(x)$			average $\mathbf{r}_2(x)$	
		TE	LB	UB	TE	LB	UB
MNIST	FC1	1.51%	**0.69**	**1.69**	0.93%	0.35	**1.69**
	FC10	1.87%	**0.48**	1.48	1.21%	0.20	**1.62**
GTS	FC1	11.15%	**0.69**	**0.69**	12.09%	0.48	0.63
	FC10	12.82%	**0.64**	**0.67**	12.41%	0.12	0.48
F-MNIST	FC1	10.22%	**0.50**	0.85	9.83%	0.31	**1.30**
	FC10	11.73%	**0.68**	**1.18**	10.32%	0.13	1.15

training, for KW this is also true but the improvements on GTS are much smaller. Notably, for the fully connected networks FC1 and FC10 on GTS and F-MNIST , the *lower bounds* achieved by MMR and/or MMR+AT are *larger* than the *upper bounds* of the plain training for F-MNIST and better than plain and adversarial training as well as KW on GTS. Thus MMR is provably more robust than the competing methods in these configurations. Moreover, the achieved lower bounds of MMR are only worse than the ones of KW on MNIST for FC10. Also for the achieved upper bounds MMR is most of the time better than KW and always improves over adversarial training. For the CNNs the improvements of KW and MMR over plain and adversarial training in terms of lower and upper bounds are smaller than for the fully connected networks and it is harder to maintain similar test performance. The differences between KW and MMR for the lower bounds are very small so that for CNNs both robust methods perform on a similar level.

Importance of linear regions maximization. In order to highlight the importance of both parts of the MMR regularization, i) penalization of the distance to decision boundary and ii) penalization of the distance to boundary of the polytope, we train, for each dataset/architecture, models penalizing only the distance to the decision boundary, that is the second term in the r.h.s. of (3.7) and (3.8). We call this partial regularizer MMR-d_p^D, in contrast to the full version MMR-*full*. Then we compare the lower and upper bounds on the solution of (3.2) for MMR-d_p^D and MMR-*full* models. For a fair comparison we consider models with similar test error. We clearly see in Table 3.4 that the lower bounds are always significantly better when MMR-*full* is used, while the behavior of the upper bounds does not clearly favor one of the two. This result shows that in order to get good lower bounds one has to increase the distance of the points to the boundaries of the polytope.

Guaranteed optimal solutions via MMR. Theorem 3.2.1 provides a simple and efficient way to obtain in certain cases the solution of (3.2). Although for normally trained networks

Table 3.5: **Occurrence of guaranteed optimal solutions.** For each dataset and architecture we report the percentage of points of the test set for which we can compute the guaranteed optimal solution of (3.2) for models trained without (plain setting) and with MMR regularization. We show the test error of the models as well. In most of the fully connected cases, we achieve certified optimal solutions for a significant fraction of the points without degrading significantly, or sometimes improving, the test error. Moreover, where we have a meaningful number of points with provable minimal perturbation, it is interesting to check how much worse the lower bounds computed by Wong and Kolter [2018] (LB) are. Thus the column *opt vs LB* indicates, in percentage, how much larger the l_2-norm of the optimal solution of (3.2) is compared to its lower bound, explicitly $\left(\|\delta_{opt}\|_2/_{LB} - 1\right) \times 100\%$.

		MMR training			plain training	
dataset	*model*	*test error*	*optimal points*	opt vs LB	*test error*	*optimal points*
MNIST	FC1	1.51%	0.20%	14.11%	1.59%	0.02%
	FC10	1.87%	0.06%	-	1.81%	0.02%
	CNN	1.17%	0.00%	-	0.97%	0.00%
GTS	FC1	11.15%	99.97%	0.59%	12.27%	1.12%
	FC10	12.82%	94.86%	0.66%	11.28%	0.14%
	CNN	7.40%	0.00%	-	6.73%	0.00%
F-MNIST	FC1	10.22%	11.22%	6.53%	9.61%	0.37%
	FC10	11.73%	9.90%	7.27%	10.53%	0.04%
	CNN	10.30%	0.09%	-	8.86%	0.00%

the conditions are rarely satisfied, we show in Table 3.5 that, for the MMR-models with fully connected networks, for a significant fraction of the test set we obtain the globally optimal solution of (3.2), that is the true l_2-robustness. Moreover, we report how much better our globally optimal solutions are compared to the lower bounds of Wong and Kolter [2018]. Interestingly, we can provably get the true robustness for around 10% of points for F-MNIST and for over 98% of the points on GTS for the case of fully connected networks. For these cases the optimal solutions have roughly 7% larger l_2-norm for F-MNIST and 0.5% larger for GTS than the lower bounds. Globally optimal solutions for larger networks achieved via our method can serve as a test both for lower and upper bounds.

Table 3.6: **Lower bounds computed by our method.** We report here for the fully connected models trained with either MMR or MMR+AT the lower bounds computed by our technique, that is exploiting Theorem 3.2.1 and integrating box constraints without and with checking additional neighboring regions (improved lower bounds) versus KW [Wong and Kolter, 2018].

dataset	*model*		*test error*	KW LB	Th. 3.2.1 LB	*improved* LB
MNIST	FC1	MMR	1.51%	0.69	0.22	0.29
	FC1	MMR+AT	1.59%	0.70	0.25	0.33
	FC10	MMR	1.87%	0.48	0.31	0.33
	FC10	MMR+AT	1.35%	0.40	0.21	0.26
GTS	FC1	MMR	11.15%	0.69	0.69	0.69
	FC1	MMR+AT	11.72%	0.72	0.72	0.72
	FC10	MMR	12.82%	0.64	0.63	0.63
	FC10	MMR+AT	13.36%	0.64	0.63	0.63
F-MNIST	FC1	MMR	10.22%	0.50	0.30	0.41
	FC1	MMR+AT	10.94%	0.66	0.33	0.42
	FC10	MMR	11.73%	0.68	0.56	0.64
	FC10	MMR+AT	11.39%	0.67	0.53	0.60

Comparison of lower bounds. In Table 3.6 we compare, for fully connected models, the lower bounds on the distance to the decision boundary computed by Wong and Kolter [2018] and our technique using Theorem 3.2.1 with integration of box constraints once just checking the initial linear region $Q(x)$ where the point x lies versus also checking neighboring linear regions. We see that Wong and Kolter [2018] obtain better lower bounds, and this is why we use their

method for the main evaluation of provable robustness in Table 3.1. Nevertheless, the gap is not too large and while the lower bounds are worse, the achieved robustness using our MMR regularization is mostly better as shown in Table 3.1.

3.2.6 Experimental details

By FC1 we denote a 1 hidden layer fully connected network with 1024 hidden units. By FC10 we denote a 10 hidden layers network that has 1 layer with 124 units, 7 layers with 104 units and 2 layers with 86 units (so that the total number of units is again 1024). The convolutional architecture that we use is identical to Wong and Kolter [2018], which consists of 2 convolutional layers with [16, 32] filters of size 4x4 and 2 fully connected layers with 100 hidden units. For all the experiments we use batch size 100 and we train the FC models for 300 epochs and the CNNs for 100 epochs. Moreover, we use Adam optimizer [Kingma and Ba, 2014] with the default learning rate 0.001. We reduce the learning rate by a factor of 10 for the last 10% of epochs. For training on CIFAR-10 dataset we apply random crops and random mirroring of the images. For the FC models we use MMR regularizer in the formulation in Eq. (3.8) with $k_B, k_D = 10$ for the first 50% epochs and in the formulation in Eq. (3.7) for the rest of epochs. For CNNs we used the formulation (3.8) with fixed $k_D = 10$, and we gradually change k_B from 400 hyperplanes in the beginning to 100 hyperplanes towards the end of training. In order to find the optimal set of hyperparameters we performed a grid search over λ from $\{0.1, 0.25, 0.5, 1.0, 2.0\}$, γ_B and γ_D from $\{0.25, 0.5, 0.75\}$ for CIFAR-10 and GTS, from $\{0.25, 0.5, 1.0\}$ for FMNIST and from $\{1.0, 1.5, 2.0\}$ for MNIST. In order to make a comparison to the robust training of Wong and Kolter [2018] we adapted it for the l_2-norm, and performed a grid search over the radius of the l_2-norm from $\{0.05, 0.1, 0.2, 0.3, 0.4, 0.6\}$ used in their robust loss, aiming at a model with non-trivial lower bounds with a little or no loss in test error.

We perform adversarial training using the PGD attack of Madry et al. [2018]. However, since we focus on l_2-norm, we adapted the implementation from Papernot et al. [2018] to perform the plain gradient update instead of the gradient sign (which corresponds to l_∞-norm and thus irrelevant for l_2 case) on every iteration. We use the following l_2-norm of the perturbation: 2.0 for MNIST, 1.0 for F-MNIST, 0.5 for GTS and CIFAR-10 using the step size of 0.5, 0.25 and 0.125 respectively. We perform 40 iterations of the PGD attack for every batch. During the training, every batch contains 50% of adversarial examples and 50% of clean examples. We use the untargeted formulation of the Carlini-Wagner l_2 attack in order to evaluate the upper bounds on the l_2-norm required to change the class. We use the settings provided in the original paper and in their code, including 20 restarts, 10000 iterations, learning rate 0.01 and initial constant of 0.001.

3.3 Provable robustness in the union of threat models

In the previous section we have considered robustness against a single type of attack. However, we are more broadly interested in guarantees which cover a variety of threat models, since an attacker is not restricted to use only one of them. Then, in the following we aim at providing guarantees that a point is simultaneously robust to multiple l_p-bounded attacks, i.e. robust in the union of l_p-threat models. First, we show in Sec. 3.3.1 which information on the size of smallest adversarial perturbation wrt l_p for $p \in (1, \infty)$ one can obtain by computing only bounds for $p = 1$ and $p = +\infty$, and we analyze it in Sec. 3.3.2. Then, we derive guarantees on the robustness of ReLU networks in the union of threat models, and extend MMR to such scenario, introducing MMR-Universal (Sec. 3.3.3). Finally, we test the effectiveness of MMR-Universal with experiments on four datasets (Sec. 3.3.4).

3.3.1 Minimal l_p-norm of the complement of the union of l_1- and l_∞-ball and its convex hull

Let $B_1 = \{x \in \mathbb{R}^d : \|x\|_1 \leq \epsilon_1\}$ and $B_\infty = \{x \in \mathbb{R}^d : \|x\|_\infty \leq \epsilon_\infty\}$ be the l_1-ball of radius $\epsilon_1 > 0$ and the l_∞-ball of radius $\epsilon_\infty > 0$ respectively, both centered at the origin in $\mathbb{R}^d$. We also assume $\epsilon_1 \in (\epsilon_\infty, d\epsilon_\infty)$, so that $B_1 \nsubseteq B_\infty$ and $B_\infty \nsubseteq B_1$. Suppose we can guarantee that the classifier does not change its label in $U_{1,\infty} = B_1 \cup B_\infty$. Which guarantee does that imply

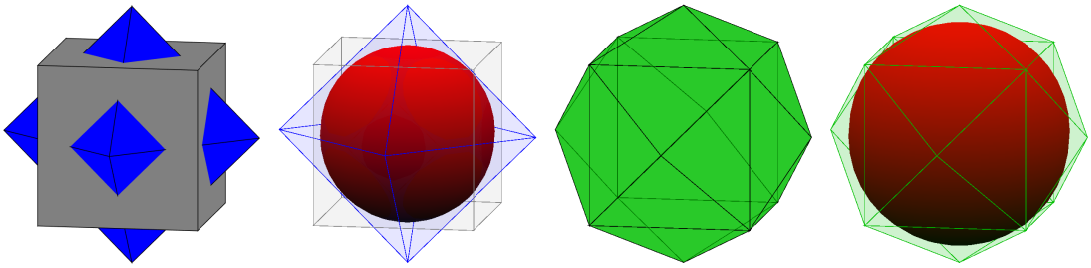

Figure 3.4: Visualization of the l_2-ball contained in the union resp. the convex hull of the union of l_1- and l_∞-balls in $\mathbb{R}^3$. **First column**: co-centric l_1-ball (blue) and l_∞-ball (black). **Second**: in red the largest l_2-ball completely contained in the union of l_1- and l_∞-ball. **Third**: in green the convex hull of the union of the l_1- and l_∞-ball. **Fourth**: the largest l_2-ball (red) contained in the convex hull. The l_2-ball contained in the convex hull is significantly larger than that contained in the union of l_1- and l_∞-ball.

for all intermediate l_p-norms? This question can be simply answered by computing the minimal l_p-norms over $\mathbb{R}^d \setminus U_{1,\infty}$, namely $\min_{x\in\mathbb{R}^d\setminus U_{1,\infty}} \|x\|_p$. By the standard norm inequalities it holds, for every $x \in \mathbb{R}^d$, that

$$\|x\|_p \geq \|x\|_\infty \qquad \text{and} \qquad \|x\|_p \geq \|x\|_1 d^{\frac{1-p}{p}},$$

and thus a naive application of these inequalities yields the bound

$$\min_{x\in\mathbb{R}^d\setminus U_{1,\infty}} \|x\|_p \geq \max\left\{\epsilon_\infty, \epsilon_1 d^{\frac{1-p}{p}}\right\}. \tag{3.12}$$

However, this naive bound does not take into account that we know that $\|x\|_1 \geq \epsilon_1$ *and* $\|x\|_\infty \geq \epsilon_\infty$. Our first result yields the exact value taking advantage of this information.

Proposition 3.3.1 *If $d \geq 2$ and $\epsilon_1 \in (\epsilon_\infty, d\epsilon_\infty)$, then*

$$\min_{x\in\mathbb{R}^d\setminus U_{1,\infty}} \|x\|_p = \left(\epsilon_\infty^p + \frac{(\epsilon_1 - \epsilon_\infty)^p}{(d-1)^{p-1}}\right)^{\frac{1}{p}}. \tag{3.13}$$

Proof. We first note that for $\epsilon_1 < \epsilon_\infty$ it holds $B_1 \subset B_\infty$ and the proof follows from the standard inequality $\|x\|_p \geq \|x\|_\infty$ where equality is attained for $x = \epsilon_\infty e_i$, where e_i are standard basis vectors. Moreover, if $\epsilon_1 > d\epsilon_\infty$ it holds $B_\infty \subset B_1$ as $\max_{x\in B_\infty} \|x\|_1 = d\epsilon_\infty$ and the result follows by $\|x\|_p \geq \|x\|_1 d^{\frac{1-p}{p}}$. The equality is realized by the vector with all the entries equal to $\frac{\epsilon_1}{d}$.

For the second case we first note that using Hölder inequality $|\langle u, v\rangle| \leq \|u\|_p \|v\|_q$ where $\frac{1}{p} + \frac{1}{q} = 1$, it holds

$$\sum_{i=1}^k |x_i| \leq \left(\sum_{i=1}^k |x_i|^p\right)^{\frac{1}{p}} k^{\frac{1}{q}}.$$

Let $x \in \mathbb{R}^d$. Without loss of generality after a potential permutation of the coordinates it holds $|x_d| = \|x\|_\infty$. Then we get

$$\|x\|_p^p = \sum_{i=1}^d |x_i|^p = |x_d|^p + \sum_{i=1}^{d-1} |x_i|^p \geq |x_d|^p + \frac{\left(\sum_{i=1}^{d-1} |x_i|\right)^p}{(d-1)^{\frac{p}{q}}}.$$

We have

$$\min_{\|x\|_\infty \geq \epsilon_\infty, \|x\|_1 \geq \epsilon_1} |x_d|^p + \frac{\left(\sum_{i=1}^{d-1} |x_i|\right)^p}{(d-1)^{\frac{p}{q}}} = \epsilon_\infty^p + \frac{(\epsilon_1 - \epsilon_\infty)^p}{(d-1)^{p-1}},$$

noting that $|x_d| = \|x\|_\infty$ and $\sum_{i=1}^{d-1} |x_i| \geq \epsilon_1 - \epsilon_\infty$.
Finally, we note that the vector

$$v = \sum_{i=1}^{d-1} \frac{\epsilon_1 - \epsilon_\infty}{d-1} e_i + \epsilon_\infty e_d,$$

realizes equality. Indeed, $\|v\|_p^p = (d-1)\frac{(\epsilon_1-\epsilon_\infty)^p}{(d-1)^p} + \epsilon_\infty^p$, which finishes the proof. □

Thus a guarantee both for l_1- and l_∞-ball yields a guarantee for all intermediate l_p-norms. However, for affine classifiers a guarantee for B_1 *and* B_∞ implies a guarantee wrt the convex hull C of their union $B_1 \cup B_\infty$. This can be seen by the fact that an affine classifier generates two half-spaces, and the convex hull of a set A is the intersection of all half-spaces containing A. Thus, inside C the decision of the affine classifier cannot change if it is guaranteed not to change in B_1 *and* B_∞, as C is completely contained in one of the half-spaces generated by the classifier (see Fig. 3.4 for illustrations of B_1, B_∞, their union and their convex hull). With the following theorem, we characterize, for any $p \geq 1$, the minimal l_p-norm over $\mathbb{R}^d \setminus C$.

Theorem 3.3.1 *Let C be the convex hull of $B_1 \cup B_\infty$. If $d \geq 2$ and $\epsilon_1 \in (\epsilon_\infty, d\epsilon_\infty)$, then*

$$\min_{x \in \mathbb{R}^d \setminus C} \|x\|_p = \frac{\epsilon_1}{\left(\epsilon_1/\epsilon_\infty - \alpha + \alpha^q\right)^{1/q}}, \tag{3.14}$$

where $\alpha = \frac{\epsilon_1}{\epsilon_\infty} - \lfloor \frac{\epsilon_1}{\epsilon_\infty} \rfloor$ and $\frac{1}{p} + \frac{1}{q} = 1$.

Proof. We first note that the minimum of the l_p-norm over $\mathbb{R}^d \setminus C$ lies on the boundary of C (otherwise any point on the segment joining the origin and y and outside C would have l_p-norm smaller than y). Moreover, the faces of C are contained in hyperplanes constructed as the affine hull of a subset of d points from the union of the vertices of B_1 and B_∞.

The vertices of B_1 are $V_1 = \{\epsilon_1 e_i, -\epsilon_1 e_i \,|\, i = 1, \ldots, d\}$, where e_i is the i-th element of the standard basis of $\mathbb{R}^d$, and that of B_∞ are V_∞, consisting of the 2^d vectors whose components are elements of $\{\epsilon_\infty, -\epsilon_\infty\}$. Note that $V_1 \cap V_\infty = \emptyset$. Any subset of d vertices from $V_1 \cup V_\infty$ defines a hyperplane which contains a face of C if it does not contain any point of the interior of C.

Let S be a set of vertices defining a hyperplane containing a face of C. We first derive conditions on the vertices contained in S. Let $k = \left\lfloor \frac{\epsilon_1}{\epsilon_\infty} \right\rfloor \in \mathbb{N}$ and $\alpha = \frac{\epsilon_1}{\epsilon_\infty} - k \in [0, 1)$. Note that $k + 1 > \frac{\epsilon_1}{\epsilon_\infty}$. Then no more than k vertices of B_1 belong to S, that is to a face of C. In fact, if we consider $k+1$ vertices of B_1, namely wlog $\{\epsilon_1 e_1, \ldots, \epsilon_1 e_{k+1}\}$, and consider their convex combination $z = \epsilon_1 \sum_{i=1}^{k+1} \frac{1}{k+1} e_i$ then $\|z\|_\infty = \frac{\epsilon_1}{k+1} < \epsilon_\infty$ by the definition of k. Thus S cannot contain more than k vertices of B_1.

Second, assume $\epsilon_1 e_j$ is in S. If any vertex v of B_∞ with $v_j = -\epsilon_\infty$ is also in S then, with $\alpha' = \frac{\epsilon_\infty}{\epsilon_1 + \epsilon_\infty} \in (0, 1)$, we get

$$\|\alpha' \epsilon_1 e_j + (1 - \alpha') v\|_\infty = \max\{|\alpha' \epsilon_1 - (1 - \alpha')\epsilon_\infty|, (1 - \alpha')\epsilon_\infty\},$$

where $(1 - \alpha')\epsilon_\infty < \epsilon_\infty$ and

$$|\alpha' \epsilon_1 - (1 - \alpha')\epsilon_\infty| = |\alpha'(\epsilon_1 + \epsilon_\infty) - \epsilon_\infty| = 0 < \epsilon_\infty.$$

Thus S would not span a face as a convex combination intersects the interior of C. This implies that if $\epsilon_1 e_j$ is in S then all the vertices v of B_∞ in S need to have $v_j = \epsilon_\infty$, otherwise S would not define a face of C. Analogously, if $-\epsilon_1 e_j \in S$ then any vertex v of B_∞ in S has $v_j = -\epsilon_\infty$. However, we note that out of symmetry reasons we can just consider faces of C in the positive orthant and thus we consider in the following just sets S which contain vertices of "positive type" $\epsilon_1 e_j$.

Let now S be a set (not necessarily defining a face of C) containing $h \leq k$ vertices of B_1 and $d - h$ vertices of B_∞ and P the matrix whose columns are these points. The matrix P has the form

$$P = \left(\begin{array}{cccc|ccc} \epsilon_1 & 0 & \ldots & 0 & \epsilon_\infty & \ldots & \epsilon_\infty \\ 0 & \epsilon_1 & \ldots & 0 & \epsilon_\infty & \ldots & \epsilon_\infty \\ \ldots & & & & & & \\ 0 & \ldots & 0 & \epsilon_1 & \epsilon_\infty & \ldots & \epsilon_\infty \\ \hline 0 & \ldots & & 0 & & & \\ \ldots & & & & & A & \\ 0 & \ldots & & 0 & & & \end{array}\right)$$

where $A \in \mathbb{R}^{d-h,d-h}$ is a matrix whose entries are either ϵ_∞ or $-\epsilon_\infty$. If the matrix P does not have full rank then the origin belongs to any hyperplane containing S, which means it cannot be a face of C. This also implies A has full rank if S spans a face of C.

We denote by π the hyperplane generated by the affine hull of S (the columns of P) assuming that A has full rank. Every point b belonging to the hyperplane π generated by S is such that there exists a unique $a \in \mathbb{R}^d$ which satisfies

$$P'a = \begin{pmatrix} \mathbf{1}_{1,d} \\ P \end{pmatrix} a = \left(\begin{array}{cccc|ccc} 1 & & & \dots & & & 1 \\ \hline \epsilon_1 & 0 & \dots & 0 & \epsilon_\infty & \dots & \epsilon_\infty \\ 0 & \epsilon_1 & \dots & 0 & \epsilon_\infty & \dots & \epsilon_\infty \\ \dots & & & & & & \\ 0 & \dots & 0 & \epsilon_1 & \epsilon_\infty & \dots & \epsilon_\infty \\ \hline 0 & \dots & & 0 & & & \\ \dots & & & & & A & \\ 0 & \dots & & 0 & & & \end{array}\right) a = \begin{pmatrix} 1 \\ b \end{pmatrix} = b',$$

where $\mathbf{1}_{d_1,d_2}$ is the matrix of size $d_1 \times d_2$ whose entries are 1.

The matrix $(P', b') \in \mathbb{R}^{d+1,d+1}$ need not have full rank, so that

$$\text{rank}P' = \text{rank}(P', b') = \dim a = d$$

and then the linear system $P'a = b'$ has a unique solution.
We define the vector $v \in \mathbb{R}^d$ as solution of $P^T v = \mathbf{1}_{d,1}$, which is unique as P has full rank. From their definitions we have $Pa = b$ and $\mathbf{1}^T a = 1$, so that

$$1 = \mathbf{1}^T a = (P^T v)^T a = v^T P a = v^T b,$$

and thus

$$\langle b, v \rangle = 1, \tag{3.15}$$

noticing that this also implies that any vector $b \in \mathbb{R}^d$ such that $\langle b, v \rangle = 1$ belongs to π (suppose that $\exists q \notin \pi$ with $\langle q, v \rangle = 1$, then define c as the solution of $Pc = q$ and then $1 = \langle q, v \rangle = \langle Pc, v \rangle = \langle c, P^T v \rangle = \langle c, \mathbf{1} \rangle$ which contradicts that $q \notin \pi$).
Applying Hölder inequality to (3.15) we get for any $b \in \pi$,

$$\|b\|_p \geq \frac{1}{\|v\|_q}, \tag{3.16}$$

where $\frac{1}{p} + \frac{1}{q} = 1$. Moreover, as $p \in (1, \infty)$ there exists always a point b^* for which (3.16) holds as equality.

In the rest of the proof we compute $\|v\|_q$ for any $q > 1$ when S is a face of C and then (3.16) yields the desired minimal value of $\|b\|_p$ over all b lying in faces of C.

Let $v = (v_1, v_2)$, $v_1 \in \mathbb{R}^h$, $v_2 \in \mathbb{R}^{d-h}$ and I_h denotes the identity matrix of $\mathbb{R}^{h,h}$. It holds

$$P^T v = \begin{pmatrix} \epsilon_1 I_h & 0 \\ \epsilon_\infty \mathbf{1}_{d-h,h} & A^T \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} \mathbf{1}_{h,1} \\ \mathbf{1}_{d-h,1} \end{pmatrix} = \mathbf{1}_{d,1},$$

which implies

$$v_1 = \left(\frac{1}{\epsilon_1}, \dots, \frac{1}{\epsilon_1}\right) \quad \text{and} \quad A^T v_2 = \mathbf{1}_{d-h,1} - \frac{h\epsilon_\infty}{\epsilon_1} \mathbf{1}_{d-h,1} = \left(1 - h\frac{\epsilon_\infty}{\epsilon_1}\right) \mathbf{1}_{d-h,1}.$$

Moreover, we have

$$\|v\|_1 = \|v_1\|_1 + \|v_2\|_1 = \frac{h}{\epsilon_1} + \|v_2\|_1, \quad \|v\|_\infty = \max\left\{\frac{1}{\epsilon_1}, \|v_2\|_\infty\right\}. \tag{3.17}$$

If S generates a face, then by definition π does not intersect with the interior of C and thus it holds for all $b \in \pi$: $\|b\|_1 \geq \epsilon_1$ and $\|b\|_\infty \geq \epsilon_\infty$. Suppose $\|v\|_1 = c > \frac{1}{\epsilon_\infty}$. Then there exists

$b^* \in \pi$ such equality in Hölder's equality is realized, that is $1 = \langle b^*, v\rangle = \|b^*\|_\infty \|v\|_1$, and thus $\|b^*\|_\infty = \frac{1}{c} < \epsilon_\infty$, which contradicts $\|b\|_\infty \geq \epsilon_\infty$ for all $b \in \pi$ and thus it must hold $\|v\|_1 \leq \frac{1}{\epsilon_\infty}$. Similarly, one can derive $\|v\|_\infty \leq \frac{1}{\epsilon_1}$. Combining (3.17) with the just derived inequalities we get upper bounds on the norms of v_2,

$$\|v_2\|_1 \leq \frac{1}{\epsilon_\infty} - \frac{h}{\epsilon_1} \quad \text{and} \quad \|v_2\|_\infty \leq \frac{1}{\epsilon_1}. \tag{3.18}$$

Furthermore v_2 is defined as the solution of

$$\frac{A^T}{\epsilon_\infty} v_2 = \left(\frac{1}{\epsilon_\infty} - \frac{h}{\epsilon_1}\right) \mathbf{1}_{d-h,1}.$$

We note that all the entries of $\frac{A^T}{\epsilon_\infty}$ are either 1 or -1, so that the inner product between each row of A^T and v_2 is a lower bound on the l_1-norm of v_2. Since every entry of the r.h.s. of the linear system is $\frac{1}{\epsilon_\infty} - \frac{h}{\epsilon_1}$ we get $\|v_2\|_1 \geq \frac{1}{\epsilon_\infty} - \frac{h}{\epsilon_1}$, which combined with (3.18) leads to $\|v_2\|_1 = \frac{1}{\epsilon_\infty} - \frac{h}{\epsilon_1}$.

This implies that $\frac{A^T}{\epsilon_\infty} v_2 = \|v_2\|_1$. In order to achieve equality $\langle u, v\rangle = \|v\|_1$ it has to hold $u_i = \operatorname{sgn}(v_i)$ for every $v_i \neq 0$. If at least two components of v were non-zero, the corresponding columns of A^T would be identical, which contradicts the fact that A^T has full rank. Thus v_2 can only have one non-zero component which in absolute value is equal to $\frac{1}{\epsilon_\infty} - \frac{h}{\epsilon_1}$ Thus, after a potential reordering of the components, v has the form

$$v = \left(\underbrace{\frac{1}{\epsilon_1}, \ldots, \frac{1}{\epsilon_1}}_{h \text{ times}}, \frac{1}{\epsilon_\infty} - \frac{h}{\epsilon_1}, 0, \ldots, 0\right).$$

From the second condition in (3.18), we have $\frac{1}{\epsilon_\infty} - \frac{h}{\epsilon_1} \leq \frac{1}{\epsilon_1}$ and $h + 1 \geq \frac{\epsilon_1}{\epsilon_\infty} = k + \alpha$. Recalling $h \leq k$, we have

$$h \in [k + \alpha - 1, k] \cap \mathbb{N}.$$

This means that, in order for S to define a face of C, we need $h = k$ if $\alpha > 0$, $h \in \{k-1, k\}$ if $\alpha = 0$ (in this case choosing $h = k - 1$ or $h = k$ leads to the same v, so in practice it is possible to use simply $h = k$ for any α).

Once we have determined v, we can use again (3.15) and (3.16) to see that

$$\|b\|_p \geq \frac{1}{\|v\|_q} = \frac{1}{\left(\frac{k}{\epsilon_1^q} + \left(\frac{1}{\epsilon_\infty} - \frac{k}{\epsilon_1}\right)^q\right)^{\frac{1}{q}}} = \frac{\epsilon_1}{\left({}^{\epsilon_1}\!/_{\epsilon_\infty} - \alpha + \alpha^q\right)^{1/q}}. \tag{3.19}$$

Finally, for any v there exists $b^* \in \pi$ for which equality is achieved in (3.19). Suppose that this b^* does not lie in a face of C. Then one could just consider the line segment from the origin to b^* and the point intersecting the boundary of C would have smaller l_p-norm contradicting the just derived inequality. Thus the b^* realizing equality in (3.19) lies in a face of C. □

3.3.2 Comparison of the robustness guarantee for the union of B_1 and B_∞ and for its convex hull

First, we note that our expression in Theorem 3.3.1 is exact and not just a lower bound. Moreover, the minimal l_p-distance of $\mathbb{R}^d \setminus C$ to the origin in Eq. (3.14) is independent from the dimension d, in contrast to the expression for the minimal l_p-norm over $\mathbb{R}^d \setminus U_{1,\infty}$ in Eq. (3.13) and its naive lower bound in Eq. (3.12), which are both decreasing for increasing d and $p > 1$.

In Fig. 3.4 we visually compare the largest l_2-balls (in red) fitting inside either $U_{1,\infty}$ or the convex hull C in $\mathbb{R}^3$, showing that the one in C is clearly larger. In Fig. 3.5 we provide a quantitative comparison in high dimensions. We plot the minimal l_2-norm over $\mathbb{R}^d \setminus C$ (as in Eq. (3.14), blue curve) and over $\mathbb{R}^d \setminus U_{1,\infty}$ (Eq. (3.13), red) and its naive lower bound (Eq. (3.12),

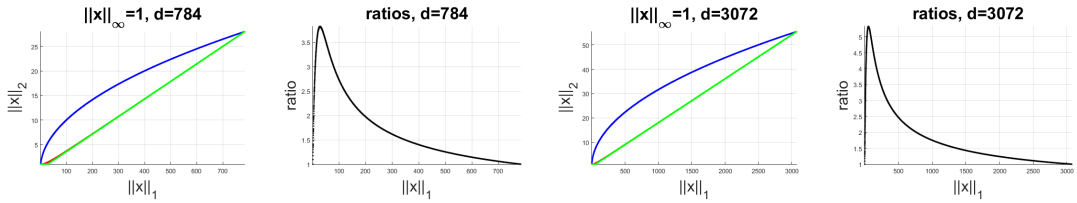

Figure 3.5: Comparison of the minimal l_2-norm over $\mathbb{R}^d \setminus C$ (3.14) (blue), $\mathbb{R}^d \setminus U_{1,\infty}$ (3.13) (red) and its naive lower bound (3.12) (green). We fix $\epsilon_\infty = 1$ and show the results varying $\epsilon_1 \in (1, d)$, for $d = 784$ and $d = 3072$. We plot the value (or a lower bound in case of (3.12)) of the minimal $\|x\|_2$, depending on ϵ_1, given by the different approaches (first and third plots). The red curves are almost completely hidden by the green ones, as they mostly overlap, but can be seen for small values of $\|x\|_1$. Moreover, we report (second and fourth plots) the ratios of the minimal $\|x\|_2$ for $\mathbb{R}^d \setminus \text{conv}(B_1 \cup B_\infty)$ and $\mathbb{R}^d \setminus (B_1 \cup B_\infty)$. The values provided by (3.14) are much larger than those of (3.13).

green). We fix $\|x\|_\infty = \epsilon_\infty = 1$ and vary $\epsilon_1 \in [1, d]$, with either $d = 784$ (left) or $d = 3072$ (right), i.e. the dimensions of the input spaces of MNIST and CIFAR-10. One sees clearly that the blue line corresponding to (3.14) is significantly higher than the other two. In the second and fourth plots of Fig. 3.5 we show, for each ϵ_1, the ratio of the l_2-distances given by (3.14) and (3.13). The maximal ratio is about 3.8 for $d = 784$ and 5.3 for $d = 3072$, meaning that the advantage of (3.14) increases with d.

As further analysis, we want to determine the ratio $\delta = \frac{\epsilon_1}{\epsilon_\infty} \in [1, d]$ for which the gain in the robustness guarantee b_C for the convex hull is maximized compared to just considering the robustness guarantee b_U for the union as a function of the dimension d of the input space. Let us denote

$$b_U = \min_{x \in \mathbb{R}^d \setminus U_{1,\infty}} \|x\|_p = \left(\epsilon_\infty^p + \frac{(\epsilon_1 - \epsilon_\infty)^p}{(d-1)^{p-1}} \right)^{\frac{1}{p}}, \quad b_C = \min_{x \in \mathbb{R}^d \setminus C} \|x\|_p = \frac{\epsilon_1}{\left(\epsilon_1/\epsilon_\infty - \alpha + \alpha^q \right)^{1/q}}$$

the two bounds from (3.13) and (3.14) respectively, which can be rewritten as

$$b_U(\delta) = \epsilon_\infty \left(1 + \frac{(\delta - 1)^p}{(d-1)^{p-1}} \right)^{\frac{1}{p}}, \quad b_C(\delta) = \frac{\epsilon_\infty \delta}{\left(\delta - \alpha + \alpha^q \right)^{1/q}} \sim \epsilon_\infty \delta^{\frac{1}{p}},$$

where $\alpha = \delta - \lfloor \delta \rfloor$. We note that

$$\delta - 1 \leq \delta - \alpha + \alpha^q = \lfloor \delta \rfloor + (\delta - \lfloor \delta \rfloor)^q \leq \delta.$$

As the differences are very small, we use instead the lower bound

$$b_C^*(\delta) = \frac{\epsilon_\infty \delta}{(\delta)^{1/q}} = \epsilon_\infty \delta^{\frac{1}{p}}.$$

We want to find the value δ^* which maximizes $\frac{b_C^*}{b_U}(\delta)$ varying d (a numerical evaluation is presented in Fig. 3.5). Notice first that δ^* maximizes also

$$\frac{(b_C^*(\delta))^p}{(b_U(\delta))^p} = \delta \left(1 + \frac{(\delta - 1)^p}{(d-1)^{p-1}} \right)^{-1} \geq 1$$

and can be found as the solution of

$$\begin{aligned} \frac{\partial}{\partial \delta} \frac{(b_C^*(\delta))^p}{(b_U(\delta))^p} &= \frac{\partial}{\partial \delta} l \left[\delta \left(1 + \frac{(\delta - 1)^p}{(d-1)^{p-1}} \right)^{-1} \right] \\ &= \left(1 + \frac{(\delta - 1)^p}{(d-1)^{p-1}} \right)^{-1} - \delta \left(1 + \frac{(\delta - 1)^p}{(d-1)^{p-1}} \right)^{-2} \frac{p(\delta - 1)^{p-1}}{(d-1)^{p-1}} = 0, \end{aligned}$$

which is equivalent to

$$(d-1)^{p-1} + (\delta - 1)^p - p\delta(\delta - 1)^{p-1} = 0.$$

Restricting the analysis to $p = 2$ for simplicity, we get

$$d - 1 + (\delta - 1)^2 - 2\delta(\delta - 1) = -\delta^2 + d = 0 \quad \Longrightarrow \quad \delta^* = \sqrt{d},$$

and one can check that δ^* is indeed a maximizer. Moreover, at δ^* we have a ratio between the two bounds

$$\frac{b_C}{b_U}\bigg|_{\delta^*} \geq \frac{b_C^*}{b_U}\bigg|_{\delta^*} = d^{\frac{1}{4}} \left(1 + \frac{(\sqrt{d} - 1)^2}{(d - 1)}\right)^{-\frac{1}{2}} \sim \frac{d^{\frac{1}{4}}}{\sqrt{2}}.$$

We observe that the improvement of the robustness guarantee by considering the convex hull instead of the union is increasing with dimension and is ≈ 3.8 for $d = 784$ and ≈ 5.3 for $d = 3072$, as shown by the illustration in Fig. 3.5.

These examples indicate that the l_p-balls contained in C can be a few times larger than those in $U_{1,\infty}$. Recall that we deal with piecewise affine networks. If we could enlarge the *linear regions* on which the classifier is affine so that it contains the l_1- and l_∞-ball of some desired radii, we would automatically get the l_p-balls of radii given by Theorem 3.3.1 to fit in the linear regions. The next section formalizes the resulting robustness guarantees.

3.3.3 Universal provable robustness with respect to all l_p-norms for $p \geq 1$

Combining the results of Theorems 3.2.1 and 3.3.1, in the next theorem we derive lower bounds on the robustness of a continuous piecewise affine classifier f, e.g. a ReLU network, at a point x wrt any l_p-norm with $p \geq 1$ using only $d_1^B(x)$, $\overline{d_1^D}(x)$, $d_\infty^B(x)$ and $\overline{d_\infty^D}(x)$ (see (3.3) and (3.6)).

Theorem 3.3.2 *Let $d_p^B(x)$, $\overline{d_p^D}(x)$ be defined as in (3.3) and (3.6) and $\rho_1 = \min\{d_1^B(x), |\overline{d_1^D}(x)|\}$ and $\rho_\infty = \min\{d_\infty^B(x), |\overline{d_\infty^D}(x)|\}$. If $d \geq 2$ and x is correctly classified, then*

$$\mathbf{r}_p(x) \geq \frac{\rho_1}{\left(\rho_1/\rho_\infty - \alpha + \alpha^q\right)^{1/q}}, \tag{3.20}$$

for any $p \in (1, \infty)$, with $\alpha = \frac{\rho_1}{\rho_\infty} - \lfloor \frac{\rho_1}{\rho_\infty} \rfloor$ and $\frac{1}{p} + \frac{1}{q} = 1$.

Proof. From the definition of $d_p^B(x)$ and $\overline{d_p^D}(x)$ we know that none of the hyperplanes $\{\pi_j\}_j$ (either boundaries of the polytope $Q(x)$ or decision hyperplanes) identified by $V^{(l)}$ and $v^{(l)}$, $l = 1, \ldots, L+1$, is closer than $\min\{d_p^B(x), |\overline{d_p^D}(x)|\}$ in l_p-distance. Therefore the interior of the l_1-ball of radius ρ_1 (namely, $B_1(x, \rho_1)$) and of the l_∞-ball of radius ρ_∞ ($B_\infty(x, \rho_\infty)$) centered in x does not intersect with any of those hyperplanes. This implies that $\{\pi_j\}_j$ are intersecting the closure of $\mathbb{R}^d \setminus \text{conv}(B_1(x, \rho_1) \cup B_\infty(x, \rho_\infty))$. Then, from Theorem 3.3.1 we get

$$\min\{d_p^B(x), |\overline{d_p^D}(x)|\} \geq \frac{\rho_1}{\left(\rho_1/\rho_\infty - \alpha + \alpha^q\right)^{1/q}}.$$

Finally, exploiting Theorem 3.2.1, $\mathbf{r}_p(x) \geq \min\{d_p^B(x), |\overline{d_p^D}(x)|\}$ holds. □

For our goal of simultaneous l_p-robustness guarantees for all $p \geq 1$, we use the insights obtained from Theorem 3.3.2 to propose a combination of MMR-l_1 and MMR-l_∞, called MMR-Universal. It enhances implicitly robustness wrt every l_p-norm without actually computing and modifying separately all the distances $d_p^B(x)$ and $\overline{d_p^D}(x)$ for the different values of p. In the following we keep the notation used in Eq. 3.8.

Definition 3.3.1 (MMR-Universal) *Let x be a training point. We define the regularizer*

$$\begin{aligned} \textit{MMR-Universal}(x) = &\frac{1}{k_B} \sum_{i=1}^{k_B} \lambda_1 \max\left(0, 1 - \frac{d^B_{1,\pi^B_{1,i}}(x)}{\gamma_1}\right) + \lambda_\infty \max\left(0, 1 - \frac{d^B_{\infty,\pi^B_{\infty,i}}(x)}{\gamma_\infty}\right) \\ &+ \frac{1}{K-1} \sum_{i=1}^{K-1} \lambda_1 \max\left(0, 1 - \frac{\overline{d^D_{1,\pi^D_{1,i}}}(x)}{\gamma_1}\right) + \lambda_\infty \max\left(0, 1 - \frac{\overline{d^D_{\infty,\pi^D_{\infty,i}}}(x)}{\gamma_\infty}\right), \end{aligned} \tag{3.21}$$

where $k_B \in \{1, \ldots, N\}$, $\lambda_1, \lambda_\infty, \gamma_1, \gamma_\infty > 0$.

Even if the formulation of MMR-Universal is based on MMR-l_p, it is just thanks to the novel geometrical motivation provided by Theorem 3.3.1 and its interpretation in terms of robustness guarantees of Theorem 3.3.2 that we have a theoretical justification of MMR-Universal. Moreover, we are not aware of any other approach which can enforce simultaneously l_1- and l_∞-guarantees, which is the key property of MMR-Universal.

The loss function which is minimized while training the classifier f is then, with $\{(x_i, y_i)\}_{i=1}^T$ being the training set and CE the cross-entropy loss,

$$L\left(\{(x_i, y_i)\}_{i=1}^T\right) = \frac{1}{T}\sum_{i=1}^{T} \text{CE}(f(x_i), y_i) + \text{MMR-Universal}(x_i).$$

During the optimization our regularizer aims at pushing both the polytope boundaries and the decision hyperplanes farther than γ_1 in l_1-distance and farther than γ_∞ in l_∞-distance from the training point x, in order to achieve robustness close or better than γ_1 and γ_∞ respectively. According to Theorem 3.3.2, this enhances also the l_p-robustness for $p \in (1, \infty)$. Note that if the projection of x on a decision hyperplane does not lie inside $Q(x)$, $\overline{d_p^D}(x)$ is just an approximation of the signed distance to the true decision surface, in which case one can argue that it is an approximation of the local Cross-Lipschitz constant which is also associated to robustness [Hein and Andriushchenko, 2017]. The regularization parameters λ_1 and λ_∞ are used to balance the weight of the l_1- and l_∞-term in the regularizer, and also wrt the cross-entropy loss. Note that the terms of MMR-Universal involving the quantities $\overline{d^D_{p,\pi^D_{p,i}}}(x)$ penalize misclassification, as they take negative values in this case. Similarly to MMR for a single norm, we take into account the k_B closest hyperplanes and not just the closest one as done in Theorem 3.3.2. This has two reasons: first, in this way the regularizer enlarges the size of the linear regions around the training points more quickly and effectively, given the large number of hyperplanes defining each polytope. Second, pushing many hyperplanes influences also the neighboring linear regions of $Q(x)$. This comes into play when, in order to get better bounds on the robustness at x, one wants to explore also a portion of the input space outside of the linear region $Q(x)$, which is where Theorem 3.3.2 holds. As noted in Raghunathan et al. [2018], Xiao et al. [2019], established methods to compute lower bounds on the robustness are loose or completely fail when using normally trained models. In fact, their effectiveness is mostly related to how many ReLU units have stable sign when perturbing the input x within a given l_p-ball. This is almost equivalent to having the hyperplanes far from x in l_p-distance, which is what MMR-Universal tries to accomplish. This explains why in Sec. 3.3.4 we can certify the models trained with MMR-Universal with the methods of Wong and Kolter [2018] and Tjeng et al. [2019].

3.3.4 Experiments

We compare the models obtained via our MMR-Universal regularizer[2] to state-of-the-art methods for provable robustness and adversarial training. Similar to what done for single norm robustness (see Sec. 3.2.4), we compute lower and upper bound on the robust test error, corresponding to empirical and provable robustness, for each l_p-balls with radius ϵ_p for $p \in \{1, 2, \infty\}$. Moreover, we report the results for the union of the three threat models. For computing the upper bounds we additionally use FAB attack, which will be described in Sec. 4.1 (more details in Sec. 3.3.5).

Choice of ϵ_p. In choosing the values of ϵ_p for $p \in \{1, 2, \infty\}$, we try to be consistent with previous literature (e.g. Wong and Kolter [2018]) for the values of ϵ_∞ and ϵ_2. Eq. (3.14) provides, given ϵ_1 and ϵ_∞, a value at which one can expect l_2-robustness (approximately $\epsilon_2 = \sqrt{\epsilon_1 \epsilon_\infty}$). Then we fix ϵ_1 such that this approximation is slightly larger than the desired ϵ_2. We show in Table 3.7 the values chosen for ϵ_p, $p \in \{1, 2, \infty\}$, and used to compute the robust test error in Table 3.8. Notice that for these values no l_p-ball is contained in the others. Moreover, we compute for the plain models the percentage of adversarial examples given by an l_1-attack (we use the PGD-attack) with budget ϵ_1 which have also l_∞-norm smaller than or equal to ϵ_∞, and vice versa. These percentages are zero for all the datasets, meaning that being (provably) robust in the union of these l_p-balls is much more difficult than in just one of them.

Table 3.7: The values chosen for ϵ_p on the different datasets and the expected l_2-robustness level (last column) given ϵ_1 and ϵ_∞, computed according to Eq. (3.14).

dataset	ϵ_1	ϵ_∞	ϵ_2	ϵ_2 by Eq. (3.14)
MNIST / F-MNIST	1	0.1	0.3	0.3162
GTS	3	$4/255$	0.2	0.2170
CIFAR-10	2	$2/255$	0.1	0.1252

Main results. We train CNNs on MNIST, Fashion-MNIST, German Traffic Sign (GTS) and CIFAR-10, with several training schemes: plain training, adversarial training (AT) for individual norms and its extension to multiple l_p-balls in Tramèr and Boneh [2019], the robust training KW, MMR, either alone or with adversarial training (MMR+AT), and the training with MMR-Universal. We use AT, KW, MMR and MMR+AT wrt l_2 and l_∞, as these are the norms for which such methods have been used in the original papers. More details about the architecture and models in Sec. 3.3.5.

In Table 3.8 we report test error (TE) computed on the whole test set and lower (LB) and upper (UB) bounds on the robust test error obtained considering the l_∞-, l_2- and l_1-balls, and the union of the three l_p-balls, indicated by $l_1 + l_2 + l_\infty$ (these statistics are on the first 1000 points of the test set). The lower bounds $l_1 + l_2 + l_\infty$-LB are given by the fraction of test points for which one of the adversarial attacks wrt l_1, l_2 and l_∞ is successful. The upper bounds $l_1 + l_2 + l_\infty$-UB are computed as the percentage of points for which at least one of the three l_p-balls is not certified to be free of adversarial examples (lower is better). This last one is the metric of main interest, since we aim at *universally provably robust* models.

MMR-Universal is the only method which can give non-trivial upper bounds on the robust test error for all datasets, while almost all other methods aiming at provable robustness have $l_1 + l_2 + l_\infty$-UB close to or at 100%. Notably, on GTS the upper bound on the robust test error of MMR-Universal is lower than the lower bound of all other methods except AT-(l_1, l_2, l_∞), showing that MMR-Universal *provably* outperforms existing methods which provide guarantees wrt individual l_p-balls, either l_2 or l_∞, when certifying the union $l_1 + l_2 + l_\infty$. The test error is slightly increased wrt the other methods giving provable robustness, but the same holds true for combined adversarial training AT-(l_1, l_2, l_∞) compared to standard adversarial training AT-l_2/l_∞. We conclude that MMR-Universal is the only method so far being able to provide non-trivial robustness guarantees for multiple l_p-balls in the case that none of them contains any other.

3.3.5 Experimental details

The convolutional architecture is identical to the one used in Sec. 3.2. The AT-l_∞, AT-l_2, KW, MMR and MMR+AT models are the same reported in Sec. 3.2. We trained the AT-(l_1, l_2, l_∞) performing for each batch of 128 images the PGD-attack wrt the three norms (40 steps for MNIST and F-MNIST, 10 steps for GTS and CIFAR-10) and then training on the point realizing the maximal loss (the cross-entropy function is used), for 100 epochs. For all experiments with MMR-Universal we use batch size 128 and we train the models for 100 epochs. Moreover, we use Adam optimizer with learning rate of 5×10^{-4} for MNIST and F-MNIST, 0.001 for the other datasets. We also reduce the learning rate by a factor of 10 for the last 10 epochs. On CIFAR-10 dataset we apply random crops and random mirroring of the images as data augmentation. For training we use MMR-Universal as in (3.21) with k_B linearly (wrt the epoch) decreasing from 20% to 5% of the total number of hidden units of the network architecture. We also use a training schedule for λ_p where we linearly increase it from $\lambda_p/10$ to λ_p during the first 10 epochs. We employ both schemes since they increase the stability of training with MMR. In order to determine the best set of hyperparameters λ_1, λ_∞, γ_1, and γ_∞ of MMR, we perform a grid search over them for every dataset. In particular, we empirically found that the optimal values of γ_p are usually between 1 and 2 times the ϵ_p used for the evaluation of the robust test error, while the values of λ_p are more diverse across the different datasets. Specifically, for the models we reported in Table 3.8 the following values for the $(\lambda_1, \lambda_\infty)$ have been used: (3.0, 12.0) for MNIST, (3.0, 40.0)

[2]Code available at https://github.com/fra31/mmr-universal.

Table 3.8: **Provable robustness against multiple perturbations.** We report, for the different datasets and training schemes, the test error (TE) and lower (LB) and upper (UB) bounds on the robust test error (in percentage) wrt the l_p-norms at thresholds ϵ_p, with $p = 1, 2, \infty$ (that is the largest test error possible if any perturbation of l_p-norm equal to ϵ_p is allowed). Moreover we show the $l_1 + l_2 + l_\infty$-UB, that is the upper bound on the robust error when the attacker is allowed to use the *union* of the three l_p-balls. The training schemes compared are plain training, adversarial training of Madry et al. [2018], Tramèr and Boneh [2019] (AT), robust training of Wong and Kolter [2018], Wong et al. [2018] (KW), MMR regularization, MMR combined with AT (MMR+AT) and our MMR-Universal regularization. One can clearly see that our MMR-Universal models are the only ones which have non trivial upper bounds on the robust test error wrt all the considered norms.

		l_1		l_2		l_∞		$l_1 + l_2 + l_\infty$	
model	TE	LB	UB	LB	UB	LB	UB	LB	UB
MNIST		$\epsilon_1 = 1$		$\epsilon_2 = 0.3$		$\epsilon_\infty = 0.1$			
plain	0.85	2.3	100	3.1	100	88.5	100	88.5	100
AT-l_∞	0.82	1.8	100	1.7	100	4.7	100	4.7	100
AT-l_2	0.87	2.1	100	2.2	100	25.9	100	25.9	100
AT-(l_1, l_2, l_∞)	0.80	2.1	100	1.7	100	4.9	100	4.9	100
KW-l_∞	1.21	3.6	100	2.8	100	4.4	4.4	4.8	100
KW-l_2	1.11	2.4	100	2.3	6.6	10.3	10.3	10.3	100
MMR-l_∞	1.65	10.0	100	5.2	100	6.0	6.0	10.4	100
MMR-l_2	2.57	4.5	62.3	6.7	14.3	78.6	99.9	78.6	99.9
MMR+AT-l_∞	1.19	3.6	100	2.4	100	3.6	3.6	4.1	100
MMR+AT-l_2	1.73	3.6	99.9	3.7	12.1	15.3	76.8	15.3	99.9
MMR-Universal	3.04	6.4	20.8	6.2	10.4	12.4	12.4	12.4	**20.8**
F-MNIST		$\epsilon_1 = 1$		$\epsilon_2 = 0.3$		$\epsilon_\infty = 0.1$			
plain	9.32	31.3	100	65.8	100	100	100	100	100
AT-l_∞	11.54	19.0	100	17.1	100	25.4	73.0	26.3	100
AT-l_2	8.10	15.9	100	20.6	100	98.8	100	98.8	100
AT-(l_1, l_2, l_∞)	14.13	22.2	100	20.3	100	28.3	98.6	29.6	100
KW-l_∞	21.73	42.7	100	30.5	99.2	32.4	32.4	43.6	100
KW-l_2	13.08	15.8	19.8	15.9	19.9	66.7	86.8	66.7	86.8
MMR-l_∞	14.51	28.5	100	23.5	100	33.2	33.6	36.7	100
MMR-l_2	12.85	18.2	39.4	24.8	33.2	95.8	100	95.8	100
MMR+AT-l_∞	14.52	27.3	100	22.9	100	27.5	30.7	31.8	100
MMR+AT-l_2	13.40	17.2	55.4	20.2	37.8	66.5	99.1	66.5	99.1
MMR-Universal	18.57	25.0	52.4	24.3	37.4	43.5	44.3	43.5	**52.9**
GTS		$\epsilon_1 = 3$		$\epsilon_2 = 0.2$		$\epsilon_\infty = 4/255$			
plain	6.77	60.5	100	38.4	99.3	71.1	98.4	71.5	100
AT-l_∞	6.83	64.0	100	24.9	99.2	31.7	82.3	64.0	100
AT-l_2	8.76	44.0	100	27.2	98.4	58.9	97.1	59.0	100
AT-(l_1, l_2, l_∞)	8.80	41.8	100	24.0	93.7	41.2	79.4	45.2	100
KW-l_∞	15.57	87.8	100	41.1	77.7	36.1	36.6	87.8	100
KW-l_2	14.35	46.5	100	30.8	35.3	57.0	63.0	57.6	100
MMR-l_∞	13.32	71.3	99.6	40.9	41.7	49.5	49.6	71.3	99.6
MMR-l_2	14.21	54.6	80.4	36.3	36.6	62.3	63.6	62.6	80.9
MMR+AT-l_∞	14.89	82.8	100	39.9	44.7	38.3	38.4	82.8	100
MMR+AT-l_2	15.34	49.4	84.3	33.2	33.8	57.2	60.2	58.1	84.8
MMR-Universal	15.98	49.7	51.5	34.3	34.6	47.0	47.0	51.6	**52.4**
CIFAR-10		$\epsilon_1 = 2$		$\epsilon_2 = 0.1$		$\epsilon_\infty = 2/255$			
plain	23.29	61.0	100	48.9	100	88.6	100	88.6	100
AT-l_∞	27.06	39.6	100	33.3	99.2	52.5	88.5	52.5	100
AT-l_2	25.84	41.9	100	35.3	99.9	62.1	99.4	62.1	100
AT-(l_1, l_2, l_∞)	35.41	47.7	100	41.7	88.2	57.0	76.8	57.1	100
KW-l_∞	38.91	51.9	100	39.9	66.1	46.6	48.0	51.9	100
KW-l_2	40.24	47.3	100	44.6	49.3	53.6	54.7	54.0	100
MMR-l_∞	34.61	54.1	100	42.3	68.4	57.7	61.0	58.7	100
MMR-l_2	40.93	58.9	98.0	50.4	56.3	72.9	86.1	72.9	98.0
MMR+AT-l_∞	35.38	50.6	100	41.2	84.7	48.7	54.2	50.8	100
MMR+AT-l_2	37.78	50.4	99.9	46.1	54.2	61.3	74.1	61.3	99.9
MMR-Universal	46.96	56.4	63.4	47.9	53.6	63.8	63.8	63.8	**64.6**

for F-MNIST, (3.0, 12.0) for GTS and (1.0, 6.0) for CIFAR-10. In Table 3.8, while the test error which is computed on the full test set, the statistics regarding upper and lower bounds on the robust test error are computed on the first 1000 points of the respective test sets. For the lower bounds we use the FAB-attack with the original parameters, 100 iterations and 10 restarts. For PGD we use also 100 iterations and 10 restarts: the directions for the update step are the sign of the gradient for l_∞, the normalized gradient for l_2 and the normalized sparse gradient suggested by Tramèr and Boneh [2019] with sparsity level 1% for MNIST and F-MNIST, 10% for GTS and CIFAR-10. Finally we use the Liner Region Attack (see Sec. 4). In order to compute the upper bounds on the robust test error in Tables 3.8 we use the method of Wong and Kolter [2018] for all the three l_p-norms and that of Tjeng et al. [2019] only for the l_∞-norm. This second one exploits a reformulation of the problem in (3.2) in terms of mixed integer programming (MIP), which is able to exactly compute the solution of (3.2) for $p \in \{1, 2, \infty\}$. However, such technique is strongly limited by its high computational cost. The only reason why it is possible to use it in practice is the exploitation of some presolvers which are able to reduce the complexity of the MIP. Unfortunately, such presolvers are effective just wrt l_∞. On the other hand, the method of Wong and Kolter [2018] applies directly to every l_p-norm. This explains why the bounds provided for l_∞ are tighter than those for l_1 and l_2.

3.3.6 Recent advancements

After the publication of our works, several new techniques for provable robustness in single threat models have appeared: for l_∞, Interval Bound Propagation (IBP) [Gowal et al., 2019a], with its improvements [Zhang et al., 2020a, Mueller et al., 2023], and Lipschitz networks [Zhang et al., 2022a,b] have achieved state-of-results. Many methods for l_2-robustness exist [Leino et al., 2021, Hu et al., 2023, Araujo et al., 2023], while l_1 remains a less explored threat models [Levine and Feizi, 2021]. These notable advances have been achieved with techniques specialized in the individual threat models. Provable robustness in the union of l_p-balls, which our approach obtains since it can be applied to any l_p-norm, has instead received significantly less attention, with, to our knowledge, only Voráček and Hein [2022] providing a new method.

3.4 Visualizing the structure of provably robust models

Here we analyse the effect of MMR regularization on the gradient of cross entropy loss wrt the input and on the structure of the convolutional filters. We focus on models trained for l_∞-robustness on MNIST, GTS, and CIFAR-10. We provide the plain and adversarially trained model as reference. In Figures 3.6, 3.8, 3.10 we visualize the gradients computed for 10 images of the corresponding test sets (shown in the first row) for the different training procedures. For MNIST, zero values are represented in white, negative in blue and positive in red, where every image is rescaled independently. For GTS and CIFAR-10, we follow Tsipras et al. [2019] and clip the gradient values to ± 3 standard deviations and then rescale them to $[0, 1]^d$. We visualize models obtained with plain training (second row), adversarial training (third), KW robust training (fourth), MMR (fifth), MMR + adversarial training (last row). Figures 3.7, 3.9, 3.11 show the structure of the weights of the filters of the two convolutional layers. The top five rows are the filters from the first layer, either all of them for MNIST, or the first five for GTS and CIFAR-10 (reshaped from $4 \times 4 \times 3$ to 4×12). The bottom five are the filters from the second convolution layer (reshaped from $4 \times 4 \times 16$ to 16×16). We present the filters for plain training (first and sixth row), adversarial training (second and seventh), KW robust training (third and eighth), MMR (fourth and ninth), MMR + adversarial training (fifth and last row). The white color corresponds to zero weights, and the farther the value is from zero, the more intense is the color. Note that each row is scaled independently.

MNIST (Figures 3.6, 3.7): As noted in Tsipras et al. [2019], adversarial training leads to more interpretable gradients that focus on salient features of the digits. All robust training schemes have the effect of creating both interpretable and sparse gradients, which is reasonable considering that a sparse gradient implies that there are less directions along which abrupt variations in the value of the loss are possible. While the KW model seems to highlight more the borders of the images, MMR models have the most concentrated gradients. In line with what has been reported in Madry et al. [2018], Wong and Kolter [2018], the filters of adversarially

trained and provably robust models (Fig. 3.7) have significantly higher sparsity than the plain model. Interestingly, in contrast to the others, MMR models preserve sparsity also in the filters on the second convolutional layer. This seems to be important for obtaining tight certificates by the KW method and for fast verification with the MIP solver.

GTS (Figures 3.8, 3.9): For GTS we also observe that all robust training schemes lead to a qualitatively different behaviour of the gradient. In particular, the models become less sensitive to variations of the background, and instead they highlight more the traffic signs. We observe that the gradients for the MMR and adversarially trained models are the most interpretable. We note that the similarity between the gradients of the KW model and MMR+AT is possibly due to their similar test error and robustness (Table 3.1). Similarly to MNIST, we observe that the convolutional filters are sparser for the adversarially trained model than for the plain model. The filters of KW and MMR models are the sparsest, while MMR+AT is less sparse and resembles more the adversarially trained model.

CIFAR-10 (Figures 3.10, 3.11): First of all, we note that the considered shallow CNNs are not able to achieve very low clean test error (the best is 24.63% for the plain model and 34.61% for provably robust models). This may explain why we cannot observe interpretable gradients as clearly as in Tsipras et al. [2019] even for the adversarially trained model. However, we note that the gradients for the robustly trained models are still qualitatively different from the plain model. While the gradients of the plain model just consist of high-frequency noise, e.g. the MMR model concentrates more on the objects rather than on the background. For the convolutional filters of the CIFAR-10 models, we make conclusions similar to GTS. In particular, the filters of KW and MMR models are the sparsest, while MMR+AT is less sparse and is more similar to the adversarially trained model.

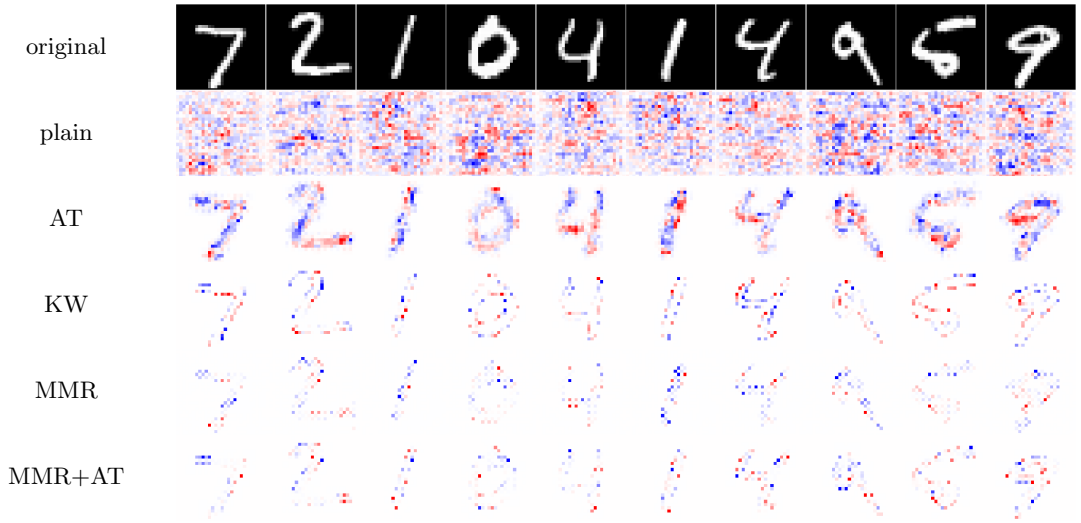

Figure 3.6: **Gradient of l_∞-robust models.** We visualize the gradients of the cross entropy loss wrt the input for different images of MNIST test set for every model: plain training, adversarial training [Madry et al., 2018], KW robust training of Wong and Kolter [2018], MMR, MMR + adversarial training. We can see for robust models the gradients are much sparser, while only for plain training it does not clearly highlights relevant features.

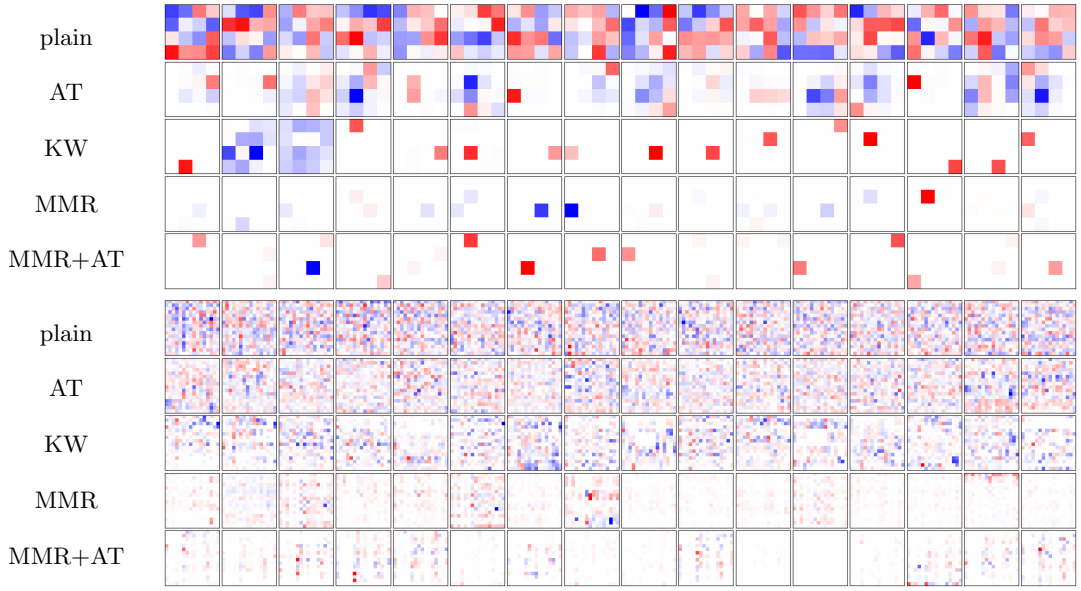

Figure 3.7: **Filters of l_∞-robust models.** We visualize all the filters of the first convolutional layer (first five rows) and 16 filters of the second layer (last five rows) for every MNIST model (trained for l_∞-robustness): plain training, adversarial training [Madry et al., 2018], KW robust training of Wong and Kolter [2018], MMR, MMR + adversarial training. We can see that MMR and MMR+AT leads to very sparse filters, especially in the second layer. Note that each row is rescaled independently.

Figure 3.8: **Gradient of l_∞-robust models.** We visualize the gradients of the cross entropy loss wrt the input for different images of GTS test set (first row) for every model: plain training, adversarial training [Madry et al., 2018], KW robust training of Wong and Kolter [2018], MMR, MMR + adversarial training. We can see for robust models the gradients are much sparser, while only for plain training it does not clearly highlights relevant features.

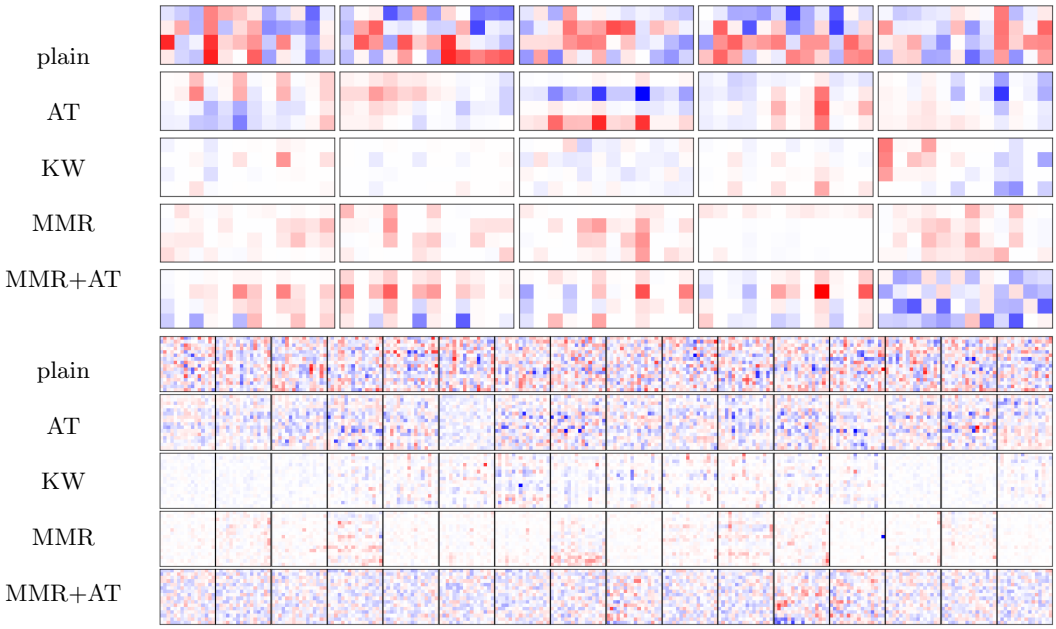

Figure 3.9: **Filters of l_∞-robust models.** We visualize the first five filters of the first convolutional layer (first five rows) and 16 filters of the second layer (last five rows) for every GTS model (trained for l_∞-robustness): plain training, adversarial training [Madry et al., 2018], KW robust training of Wong and Kolter [2018], MMR, MMR + adversarial training. We can see that MMR leads to sparser filters, especially in the second layer (each row is rescaled independently).

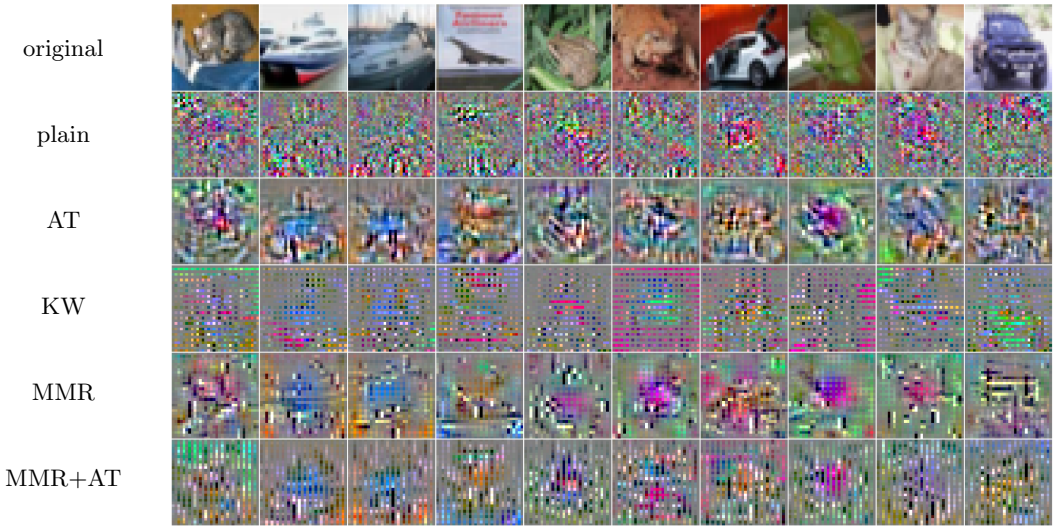

Figure 3.10: **Gradient of l_∞-robust models.** We visualize the gradients of the cross entropy loss wrt the input for different images of CIFAR-10 test set for every model: plain training, adversarial training [Madry et al., 2018], KW robust training of Wong and Kolter [2018], MMR, MMR + adversarial training. We can see for robust models the gradients are sparser than for plain training and they tend to highlight relevant features of the images.

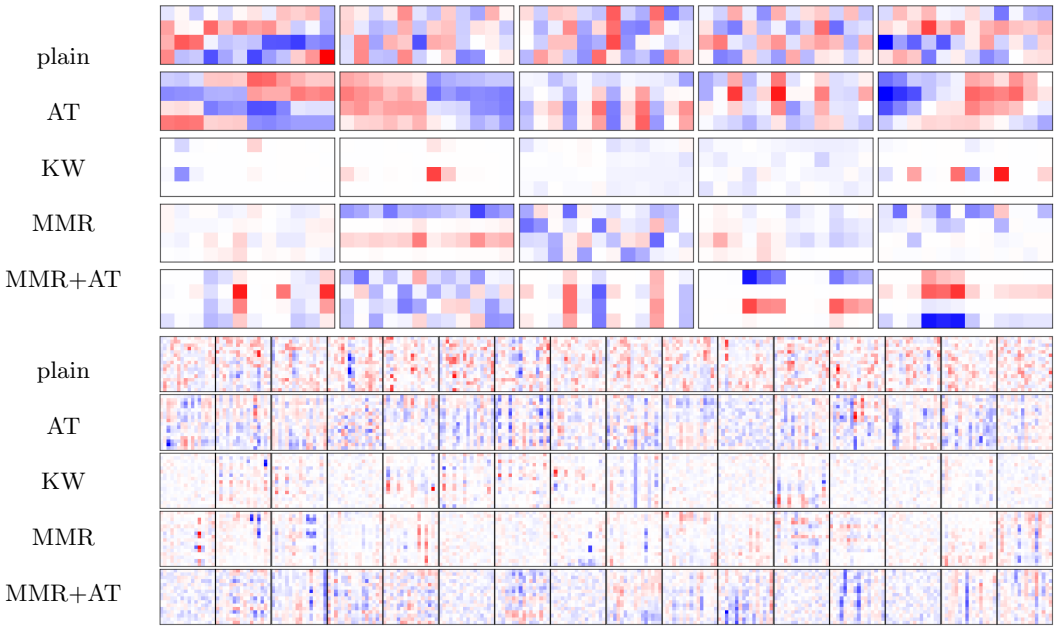

Figure 3.11: **Filters of l_∞-robust models.** We visualize the first five filters of the first convolutional layer (first five rows) and 16 filters of the second layer (last five rows) for every CIFAR-10 model (trained for l_∞-robustness): plain training, adversarial training [Madry et al., 2018], KW robust training of Wong and Kolter [2018], MMR, MMR + adversarial training. We see that MMR and KW lead to sparser filters, especially in the second layer (each row is rescaled independently).

Chapter 4

Adversarial Attacks in l_p-Bounded Threat Models

In the following we present several algorithms to generate adversarial perturbations in l_p-threat models. They differ from each other in either the level of access to the target model, the optimization problem they solve or the strategies they rely on. Table 4.1 summarizes the characteristics of each method: Square Attack is the only black-box attack, while the Linear Region Attack (LRA) is specialized for the l_2-threat model. Both LRA and the Fast Adaptive Boundary Attack (FAB) try to minimize, with different strategies, the l_p-norm of the adversarial perturbations. Conversely, Square Attack and Auto-PGD optimize some loss function in a fixed l_p-ball which determines the set of allowed perturbations. We present FAB in Sec. 4.1, then the l_∞ and l_2 versions of Square Attack (Sec. 4.2) and Auto-PGD (Sec. 4.3), and their extensions to the l_1-threat model in Sec. 4.4. We omit the detailed presentation of LRA since it applies only to ReLU networks and in the l_2-threat model.

Table 4.1: Summary of algorithms for generating l_p-attacks.

attack	*bounds type*	*knowledge*	*goal*
Linear Region Attack	l_2	white-box	norm minimization
Fast Adaptive Boundary Attack	l_∞, l_2, l_1	white-box	norm minimization
Square Attack	l_∞, l_2, l_1	black-box	loss optimization in fixed ball
Auto-PGD	l_∞, l_2, l_1	white-box	loss optimization in fixed ball

4.1 Fast Adaptive Boundary Attack

The first algorithm we present aims at finding adversarial perturbations with minimal l_p-norm for $p \in \{1, 2, \infty\}$. In practice, given a classifier $f : \mathbb{R}^d \to \mathbb{R}^K$ and a point $x_{\text{orig}} \in \mathbb{R}^d$ with true class c, we define the *minimal adversarial perturbation* for x_{orig} wrt the l_p-norm as

$$\delta_{\text{min},p} = \arg\min_{\delta \in \mathbb{R}^d} \|\delta\|_p, \quad \text{s.th.} \quad \max_{l \neq c} f_l(x_{\text{orig}} + \delta) \geq f_c(x_{\text{orig}} + \delta), \quad x_{\text{orig}} + \delta \in C, \tag{4.1}$$

where C represents the domain constraints. Note that the optimization problem (4.1) is non-convex and NP-hard for non-trivial classifiers [Katz et al., 2017], and thus $\delta_{\text{min},p}$ is usually approximated by an *attack algorithm*, which can be seen as a heuristic to solve (4.1).

The high-level idea is that, first, we use the linearization of the classifier at the current iterate $x^{(i)}$ to compute the box-constrained projections of $x^{(i)}$ respectively x_{orig} onto the approximated decision hyperplane and, second, we take a convex combinations of these projections depending on the distance of $x^{(i)}$ and x_{orig} to the decision hyperplane. Finally, we perform an extrapolation step. We explain below the geometric motivation behind these steps. The attack closest in spirit is DeepFool [Moosavi-Dezfooli et al., 2016] which is known to be very fast but suffers from low quality. DeepFool just tries to find the decision boundary quickly but has no incentive to provide

a solution close to x_{orig}. Our scheme resolves this main problem and, together with the exact projection we use, leads to a principled way to track the decision boundary (the surface where the decision of f changes) *close* to x_{orig}.

We first recall the definition and properties of the projection wrt the l_p-norms of a point on the intersection of a hyperplane and box constraints, as they are an essential part of our attack. Then, we present our FAB-attack algorithm to generate minimally distorted adversarial examples.

4.1.1 Projection on a hyperplane with box constraints

Let $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ be the normal vector and the offset defining the hyperplane $\pi : \langle w, x\rangle + b = 0$. Let $x \in \mathbb{R}^d$, we denote by the *box-constrained projection* wrt the l_p-norm of x on π (projection onto the intersection of the box $C = \{z \in \mathbb{R}^d : l_i \leq z_i \leq u_i\}$ and the hyperplane π) the following minimization problem:

$$z^* = \underset{z \in \mathbb{R}^d}{\arg\min} \ \|z - x\|_p \quad \text{s.th.} \quad \langle w, z\rangle + b = 0, \quad l_i \leq z_i \leq u_i, \ i = 1, \dots, d, \tag{4.2}$$

where $l_i, u_i \in \mathbb{R}$ are lower and upper bounds on each component of z. For $p \geq 1$ the optimization problem (4.2) is convex. Hein and Andriushchenko [2017] proved that for $p \in \{1, 2, \infty\}$ the solution can be obtained in $O(d \log d)$ time, that is the complexity of sorting a vector of d elements, as well as determining that there exists no feasible point.

Since this projection is part of our iterative scheme, we need to handle specifically the case of (4.2) being infeasible. In this case, defining $\rho = \text{sign}(\langle w, x\rangle + b)$, we instead compute $z' = \underset{l_i \leq z_i \leq u_i}{\arg\min} \, \rho \cdot (\langle w, z\rangle + b)$, whose solution is

$$z'_i = \begin{cases} l_i & \text{if } \rho w_i > 0, \\ u_i & \text{if } \rho w_i < 0, \\ x_i & \text{if } w_i = 0 \end{cases} \quad \text{for } i = 1, \dots, d. \tag{4.3}$$

Assuming that the point x satisfies the box constraints (as it holds in our algorithm), this is equivalent to identifying the corner of the d-dimensional box, defined by the componentwise constraints on z, closest to the hyperplane π. Note that if (4.2) is infeasible then the objective function of (4.3) stays positive and the points x and z are strictly contained in the same of the two halfspaces divided by π. Finally, we define the projection operator

$$\text{proj}_p : (x, \pi, C) \longmapsto \begin{cases} z^* & \text{if (4.2) is feasible} \\ z' & \text{else} \end{cases} \tag{4.4}$$

which yields the point as close as possible to π without violating the box constraints.

4.1.2 FAB-attack

We now describe the details of the algorithm of our attack. If f was a linear classifier then the closest point to $x^{(i)}$ on the decision hyperplane could be found in closed form. However neural networks are highly non-linear (although ReLU networks, i.e. neural networks which use ReLU as activation function, are piecewise affine functions and thus locally a linearization of the network is an exact description of the classifier). Let $l \neq c$, then the decision boundary between classes l and c can be locally approximated using a first order Taylor expansion at $x^{(i)}$ by the hyperplane

$$\pi_l(z) : f_l(x^{(i)}) - f_c(x^{(i)}) + \left\langle \nabla f_l(x^{(i)}) - \nabla f_c(x^{(i)}), z - x^{(i)} \right\rangle = 0. \tag{4.5}$$

Moreover the l_p-distance $d_p(x^{(i)}, \pi_l)$ of $x^{(i)}$ to π_l is given, assuming $\frac{1}{p} + \frac{1}{q} = 1$, by

$$d_p(x^{(i)}, \pi_l) = \frac{|f_l(x^{(i)}) - f_c(x^{(i)})|}{\left\| \nabla f_l(x^{(i)}) - \nabla f_c(x^{(i)}) \right\|_q}. \tag{4.6}$$

Note that if $d_p(x^{(i)}, \pi_l) = 0$ then $x^{(i)}$ belongs to the true decision boundary. Moreover, if the local linear approximation of the network is correct then the class s with the decision hyperplane closest to the point $x^{(i)}$ can be computed as

$$s = \arg\min_{l \neq c} \frac{|f_l(x^{(i)}) - f_c(x^{(i)})|}{\left\|\nabla f_l(x^{(i)}) - \nabla f_c(x^{(i)})\right\|_q}. \tag{4.7}$$

Thus, given that the approximation holds in some large enough neighborhood, the projection $\text{proj}_p(x^{(i)}, \pi_s, C)$ of $x^{(i)}$ onto π_s lies on the decision boundary (unless (4.2) is infeasible).

Biased gradient step. The iterative algorithm $x^{(i+1)} = \text{proj}_p(x^{(i)}, \pi_s, C)$ would be similar to DeepFool except that our projection operator is exact whereas they project onto the hyperplane and then clip to $[0,1]^d$. This scheme is not biased towards the original target point x_{orig}, thus it goes typically further than necessary to find a point on the decision boundary as basically the algorithm does not aim at the minimal adversarial perturbation. Then we consider additionally $\text{proj}_p(x_{\text{orig}}, \pi_s, C)$ and use instead the iterative step, with $x^{(0)} = x_{\text{orig}}$ and $\alpha \in [0,1]$, defined as

$$x^{(i+1)} = (1-\alpha)\,\text{proj}_p(x^{(i)}, \pi_s, C) + \alpha\,\text{proj}_p(x_{\text{orig}}, \pi_s, C), \tag{4.8}$$

which biases the step towards x_{orig} (see Fig. 4.1). Note that this is a convex combination of two points on π_s and in C and thus also $x^{(i+1)}$ lies on π_s and is contained in C. As we wish a scheme with minimal amount of parameters, our goal is an automatic selection of α based on the available geometric quantities. Let

$$\delta^{(i)} = \text{proj}_p(x^{(i)}, \pi_s, C) - x^{(i)}, \quad \text{and} \quad \delta^{(i)}_{\text{orig}} = \text{proj}_p(x_{\text{orig}}, \pi_s, C) - x_{\text{orig}}.$$

Note that $\left\|\delta^{(i)}\right\|_p$ and $\left\|\delta^{(i)}_{\text{orig}}\right\|_p$ are the distances of $x^{(i)}$ and x_{orig} to π_s (inside C). We propose to use for the parameter α the relative magnitude of these two distances, that is

$$\alpha = \min\left\{\frac{\left\|\delta^{(i)}\right\|_p}{\left\|\delta^{(i)}\right\|_p + \left\|\delta^{(i)}_{\text{orig}}\right\|_p}, \alpha_{\max}\right\} \in [0,1]. \tag{4.9}$$

The motivation for doing so is that if $x^{(i)}$ is close to the decision boundary then we should stay close to this point (note that π_s is the approximation of f computed at $x^{(i)}$ and thus it is valid in a small neighborhood of $x^{(i)}$, whereas x_{orig} is farther away). On the other hand we want to have the bias towards x_{orig} in order not to go too far away from x_{orig}. This is why α depends on the distances of $x^{(i)}$ and x_{orig} to π_s but we limit it from above with $\alpha_{\max}$. Finally, we use a small extrapolation step as we noted empirically, similarly to Moosavi-Dezfooli et al. [2016], that this helps to cross faster the decision boundary and get an adversarial sample. This leads to the final scheme:

$$x^{(i+1)} = \text{proj}_C\Big((1-\alpha)\big(x^{(i)} + \eta\delta^{(i)}\big) + \alpha\big(x_{\text{orig}} + \eta\delta^{(i)}_{\text{orig}}\big)\Big), \tag{4.10}$$

where α is chosen as in (4.9), $\eta \geq 1$ and proj_C is the projection onto the box which can be done by clipping. In Fig. 4.1 we visualize the scheme: in black one can see the hyperplane π_s and the vectors $\delta^{(i)}_{\text{orig}}$ and $\delta^{(i)}$, in blue the step not biased towards x_{orig}, while in red the biased step of FAB-attack, see (4.10). The green vector shows the bias towards the original point we introduce. On the left of Fig. 4.1 we use $\eta = 1$, while on the right we use extrapolation $\eta > 1$.

Interpretation of $\text{proj}_p(x_{\text{orig}}, \pi_s, C)$. The projection of the target point x_{orig} onto the intersection of π_s and C is

$$\arg\min_{z \in \mathbb{R}^d} \left\|z - x_{\text{orig}}\right\|_p \text{ s.th. } \langle w, z\rangle + b = 0,\ l_i \leq z_i \leq u_i,$$

Note that replacing z by $x^{(i)} + \delta$ we can rewrite this as

$$\arg\min_{\delta \in \mathbb{R}^d} \left\|x^{(i)} + \delta - x_{\text{orig}}\right\|_p \text{ s.th. } \quad \left\langle w, x^{(i)} + \delta\right\rangle + b = 0, \quad l_i \leq x_i + \delta_i \leq u_i.$$

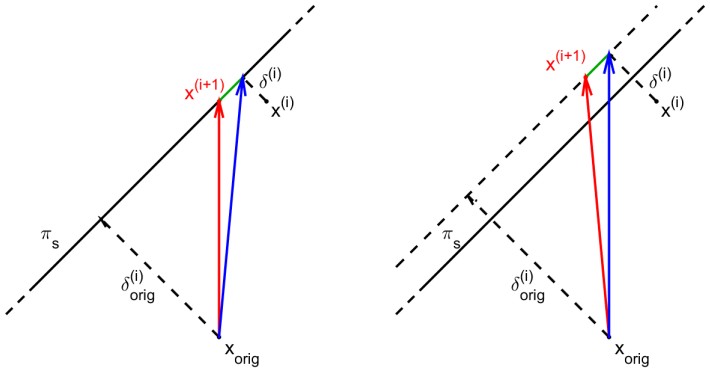

Figure 4.1: **Visualization of FAB-attack scheme.** Left, case $\eta = 1$, right, $\eta > 1$ (extrapolation). In blue we show $\text{proj}_p(x^{(i)}, \pi_s, C)$, the iterate one would get without any bias towards x_{orig}, in green the effect of the bias we introduce and in red the actual iterate $x^{(i+1)}$ of FAB-attack in (4.10). FAB-attack stays closer to x_{orig} compared to the unbiased gradient step with $\text{proj}_p(x^{(i)}, \pi_s, C)$.

This can be interpreted as the minimization of the distance of the next iterate $x^{(i)} + \delta$ to the target point x_{orig} so that $x^{(i)} + \delta$ lies on the intersection of the (approximate) decision hyperplane and the box C. This point of view on the projection $\text{proj}_p(x_{\text{orig}}, \pi_s, C)$ again justifies using a convex combination of the two projections in our scheme in (4.10).

Backward step. The described scheme finds in a few iterations adversarial perturbations. However, we are interested in minimizing their norms. Thus, once we have a new point $x^{(i+1)}$, we check whether it is assigned by f to a class different from c. In this case, we apply

$$x^{(i+1)} = (1 - \beta)x_{\text{orig}} + \beta x^{(i+1)}, \quad \beta \in (0, 1), \tag{4.11}$$

that is we go back towards x_{orig} on the segment $[x^{(i+1)}, x_{\text{orig}}]$, effectively starting again the algorithm at a point which is close to the decision boundary. In this way, due to the bias of the method towards x_{orig} we successively find adversarial perturbations of smaller norm, meaning that the algorithm *tracks* the decision boundary while getting closer to x_{orig}. We fix $\beta = 0.9$ in all experiments.

Final search. Our scheme finds points close to the decision boundary but often they are slightly off as the linear approximation is not exact and we apply the extrapolation step with $\eta > 1$. Thus, after finishing N_{iter} iterations of our algorithmic scheme, we perform a last, fast step to further improve the quality of the adversarial examples. Let x_{out} be the closest point to x_{orig} classified differently from c, say $s \neq c$, found with the iterative scheme. It holds that $f_s(x_{\text{out}}) - f_c(x_{\text{out}}) > 0$ and $f_s(x_{\text{orig}}) - f_c(x_{\text{orig}}) < 0$. This means that, assuming f continuous, there exists a point x^* on the segment $[x_{\text{out}}, x_{\text{orig}}]$ such that $f_s(x^*) - f_c(x^*) = 0$ and $\|x^* - x_{\text{orig}}\|_p < \|x_{\text{out}} - x_{\text{orig}}\|_p$. If f is linear

$$x^* = x_{\text{out}} - \frac{(f_s(x_{\text{out}}) - f_c(x_{\text{out}}))\,(x_{\text{out}} - x_{\text{orig}})}{f_s(x_{\text{out}}) - f_c(x_{\text{out}}) + f_s(x_{\text{orig}}) - f_c(x_{\text{orig}})}. \tag{4.12}$$

Since f is non-linear, we compute iteratively for a few steps

$$x_{\text{temp}} = x_{\text{out}} - \frac{(f_s(x_{\text{out}}) - f_c(x_{\text{out}}))\,(x_{\text{out}} - x_{\text{orig}})}{f_s(x_{\text{out}}) - f_c(x_{\text{out}}) + f_s(x_{\text{orig}}) - f_c(x_{\text{orig}})}, \tag{4.13}$$

each time replacing in (4.13) x_{out} with x_{temp} if $f_s(x_{\text{temp}}) - f_c(x_{\text{temp}}) > 0$ or x_{orig} with x_{temp} if instead $f_s(x_{\text{temp}}) - f_c(x_{\text{temp}}) < 0$. With this kind of modified binary search one can find a better adversarial sample with the cost of a few forward passes (which is fixed to 3 in all experiments).

Random restarts. So far all the steps are deterministic. To improve the results, we introduce the option of random restarts, that is $x^{(0)}$ is randomly sampled in the proximity of x_{orig} instead

Algorithm 1: FAB-attack

Input : x_{orig} original point, c original class, $N_{\text{restarts}}, N_{\text{iter}}, \alpha_{\max}, \beta, \eta, \epsilon, p$
Output: x_{out} adversarial example

$u \leftarrow +\infty$
for $j = 1, \dots, N_{restarts}$ **do**
  if $j = 1$ **then** $x^{(0)} \leftarrow x_{\text{orig}}$;
  else $x^{(0)} \leftarrow$ randomly sampled s.th. $\left\|x^{(0)} - x_{\text{orig}}\right\|_p = {}^{\min\{u,\epsilon\}}/_2$;
  for $i = 0, \dots, N_{iter} - 1$ **do**
    $s \leftarrow \arg\min_{l \neq c} \frac{|f_l(x^{(i)}) - f_c(x^{(i)})|}{\left\|\nabla f_l(x^{(i)}) - \nabla f_c(x^{(i)})\right\|_q}$
    $\delta^{(i)} \leftarrow \text{proj}_p(x^{(i)}, \pi_s, C)$
    $\delta^{(i)}_{\text{orig}} \leftarrow \text{proj}_p(x_{\text{orig}}, \pi_s, C)$
    compute α as in Eq. (4.9)
    $x^{(i+1)} \leftarrow \text{proj}_C\left((1-\alpha)\left(x^{(i)} + \eta\delta^{(i)}\right) + \alpha(x_{\text{orig}} + \eta\delta^{(i)}_{\text{orig}})\right)$
    if $x^{(i+1)}$ *is not classified as* c **then**
      if $\left\|x^{(i+1)} - x_{orig}\right\|_p < u$ **then**
        $x_{\text{out}} \leftarrow x^{(i+1)}$
        $u \leftarrow \left\|x^{(i+1)} - x_{\text{orig}}\right\|_p$
      end
      $x^{(i+1)} \leftarrow (1-\beta)x_{\text{orig}} + \beta x^{(i+1)}$
    end
  end
end
perform 3 steps of final search on x_{out} as in (4.13)

of being x_{orig} itself. Most attacks benefit from random restarts, e.g. Madry et al. [2018], Zheng et al. [2019], especially dealing with models trained for robustness [Mosbach et al., 2018], as it allows a wider exploration of the input space. We choose to sample from the l_p-sphere centered in the original point with radius half the l_p-norm of the current best adversarial perturbation (or a given threshold if no adversarial example has been found yet).

Computational cost. Our attack, in Alg. 1, consists of two main operations: the computation of f and its gradients and solving the projection (4.2). We perform, for each iteration, a forward and a backward pass of the network in the gradient step and a forward pass in the backward step. The projection can be efficiently implemented to run in batches on the GPU and its complexity depends only on the input dimension. Thus, except for shallow models, its cost is much smaller than the passes through the network. We can approximate the computational cost of our algorithm by the total number of calls of the classifier $N_{\text{iter}} \times N_{\text{restarts}} \times (2 \times \text{forward passes} + 1 \times \text{backward pass})$. Per restart one has to add the three forward passes for the final search.

4.1.3 Scale invariance of FAB-attack

For a given classifier f, the decisions and thus adversarial samples do not change if we rescale the classifier $g = \alpha f$ for $\alpha > 0$ or shift its logits as $h = f + \beta$ for $\beta \in \mathbb{R}$. The following proposition states that FAB-attack is invariant under both rescaling and shifting.

Proposition 4.1.1 *Let $f : \mathbb{R}^d \to \mathbb{R}^K$ be a classifier. Then for any $\alpha > 0$ and $\beta \in \mathbb{R}$ the output x_{out} of Algorithm 1 for the classifier f is the same as of the classifiers $g = \alpha f$ and $h = f + \beta$.*

Proof. It holds $\nabla g_i = \nabla \alpha f_i = \alpha \nabla f_i$ and $\nabla h_i = \nabla(f_i + \beta) = \nabla f_i$ for $i = 1, \dots, K$ and thus the definition of the hyperplane $\pi_l(z)$ in (4.5) is not affected by rescaling or shifting. The same holds for the distance of $x^{(i)}$ to the hyperplane $\pi_l(z)$ in (4.7). Note that the rest of the steps just depends on geometric quantities derived from these quantities and are independent of f. Finally, also the iterates of the final search in (4.13) are invariant under rescaling and shifting and thus

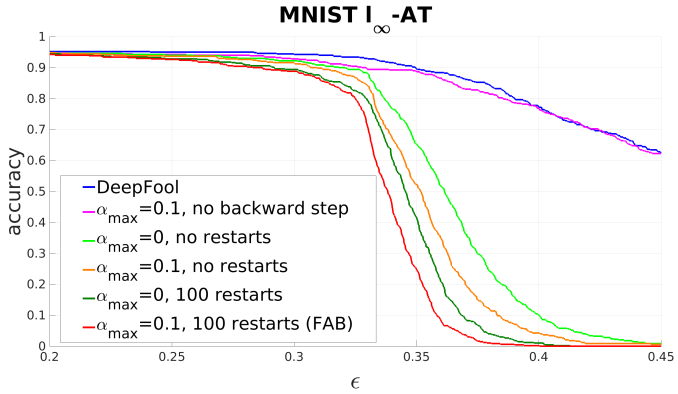

Figure 4.2: **Ablation study to DeepFool for l_∞-attacks.** The curves show the robust accuracy as a function of the threshold ϵ under different attacks on the l_∞-AT model on MNIST (lower values mean stronger attacks). The introduction of the convex combination ($\alpha_{\max} = 0.1$, no backward step) already improves over DeepFool. If one uses the backward step, the case $\alpha_{\max} = 0$ (which can be seen as an improved iterative DF) is worse than $\alpha_{\max} = 0.1$ with the same number of restarts.

the output x_{out} of Alg. 1 is invariant under rescaling of f. □

We note that the cross-entropy loss $\text{CE}(x, y, f) = -\log(e^{f_y(x)} / \sum_{j=1}^K e^{f_j(x)})$ used as objective in the normal PGD attack and its gradient wrt x

$$\nabla_x \text{CE}(x, y, f) = -\nabla_x f_y(x) + \frac{\sum_{j=1}^K e^{f_j(x)} \nabla_x f_j(x)}{\sum_{j=1}^K e^{f_j(x)}}$$

are not invariant under rescaling. Moreover, we observe that the gradient vanishes for αf if $f_y(x) > f_j(x)$ for $j \neq y$ (correctly classified point) as $\alpha \to \infty$. Due to finite precision the gradient becomes zero for finite α and it is obvious that in this case PGD gets stuck. Due to the rescaling invariance FAB-attack is not affected by gradient masking due to this phenomenon as it uses the gradient of the differences of the logits and not the gradient of the cross-entropy loss. The latter runs much earlier into numerical problems when one upscales the classifier due to the exponential function. In the experiments (see below) we show that PGD can catastrophically fail due to a "wrong" scaling whereas FAB-attack is unaffected.

4.1.4 Comparison to DeepFool

The idea of exploiting the first order local approximation of the decision boundary is not novel but the basis of one of the first white-box adversarial attacks, DeepFool (DF). While DF and our FAB-attack share the strategy of using a linear approximation of the classifier and projecting on the decision hyperplanes, we want to point out many key differences: first, DF does not solve the projection (4.2) but its simpler version without box constraints, clipping afterwards. Second, their gradient step does not have any bias towards the original point, that is equivalent to $\alpha = 0$ in (4.10). Third, DF does not have any backward step, final search or restart, as it stops as soon as a misclassified point is found (its goal is to provide quickly an adversarial perturbation of average quality). We perform an ablation study of the differences to DF in Fig. 4.2, where we show robust accuracy as a function of the threshold ϵ (lower is better). We present the results of DeepFool (blue) and FAB-attack with the following variations: $\alpha_{\max} = 0.1$ and no backward step (magenta), $\alpha_{\max} = 0$ (that is no bias in the gradient step) and no restarts (light green), $\alpha_{\max} = 0.1$ and no restarts (orange), $\alpha_{\max} = 0$ and 100 restarts (dark green) and $\alpha_{\max} = 0.1$ and 100 restarts, that is FAB-attack, (red). We can see how every addition we make to the original scheme of DeepFool contributes to the significantly improved performance of FAB-attack when compared to the original DeepFool.

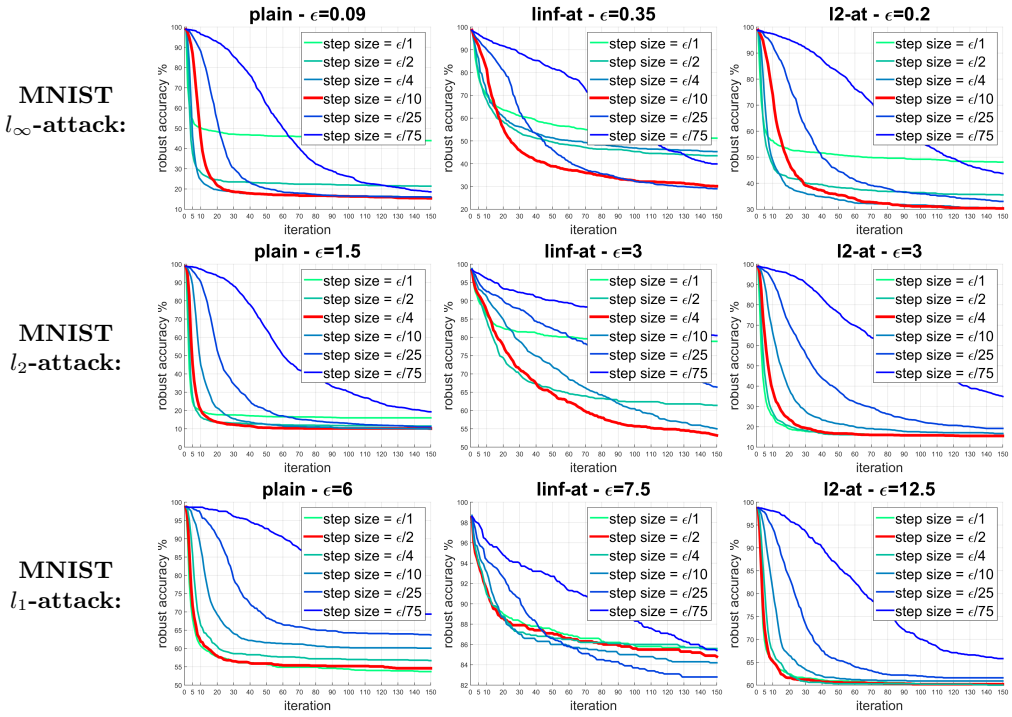

Figure 4.3: Evolution of robust accuracy across iterations for different step sizes for PGD wrt l_1, l_2, l_∞ for three different models on MNIST. In red we highlight the step size we used for each norm in the experiments, which performs on average the best.

4.1.5 Experiments

Models. We run experiments on MNIST, CIFAR-10 and Restricted ImageNet [Tsipras et al., 2019]. For each dataset we consider a normally trained model (*plain*) and two adversarially trained ones as in Madry et al. [2018] wrt the l_∞-norm (l_∞-AT) and the l_2-norm (l_2-AT) (see Sec. 4.1.7 for details).

Attacks. We compare the performance of FAB-attack[1] to those of attacks representing the state-of-the-art in each norm: DeepFool (DF), Carlini-Wagner l_2-attack (CW), Linear Region l_2-Attack (LRA), Projected Gradient Descent on the cross-entropy function (PGD), Distributionally Adversarial Attack (DAA) [Zheng et al., 2019], SparseFool (SF) [Modas et al., 2019], Elastic-net Attack (EAD) [Chen et al., 2018]. We use DF from Rauber et al. [2017], CW and EAD as in Papernot et al. [2018], DAA and LRA with the code from the original papers, while we reimplemented SF and PGD. For MNIST and CIFAR-10 we used DAA with 50 restarts, PGD and FAB with 100 restarts. For Restricted ImageNet, we used DAA, PGD and FAB with 10 restarts (for l_1 we used 5 restarts, since the methods benefit from more iterations). Moreover, we could not use LRA since it hardly scales to models of such scale and CW and EAD for compatibility issues between the implementations of attacks and models. See Sec. 4.1.7 for more details e.g. regarding number of iterations and hyperparameters of all attacks. In order to select the optimal step size for PGD for each norm, we performed a grid search on the step size parameter in ϵ/t for $t \in \{1, 2, 4, 10, 25, 75\}$ for different models and thresholds, and took the values working best on average, see Fig. 4.3 for an illustration of the results on MNIST. As a result we use for PGD wrt l_∞ step size $\epsilon/10$ and the direction is the sign of the gradient of the cross-entropy loss, for PGD wrt l_2 we do a step in the direction of the l_2-normalized gradient with step size $\epsilon/4$, for PGD wrt l_1 we use the gradient step suggested in Tramèr and Boneh [2019] (with sparsity levels of 1% for MNIST and 10% for CIFAR-10 and Restricted ImageNet) with step size $\epsilon/2$. For FAB-attack we use always $\beta = 0.9$ and on MNIST and CIFAR-10: $\alpha_{\max} = 0.1$, $\eta = 1.05$ and on Restricted ImageNet: $\alpha_{\max} = 0.05$, $\eta = 1.3$. These parameters are the same for all norms.

[1] https://github.com/fra31/fab-attack

Table 4.2: Performance summary of all attacks on MNIST and CIFAR-10 (aggregated). We report, for each norm, "avg. rob. acc.", the mean of robust accuracies across all models and datasets (lower is better), "# best", number of times the attack is the best one, "avg. diff. to best" and "max diff. to best", the mean and maximum difference of the robust accuracy of the attack to the robust accuracy of the best attack for each model/threshold (on the first 1000 points for l_∞ and l_1, 500 for l_2, of the test sets). The numbers after the name of the attacks indicate the number of restarts. In total we have 5 thresholds $\times$ 6 models = 30 cases for each of the 3 norms. *Note that for FAB-10 (i.e. with 10 restarts) the "# best" is computed excluding the results of FAB-100.

statistics on MNIST + CIFAR-10

l_∞-norm		DF	DAA-50	PGD-100	FAB-10	FAB-100
avg. rob. acc.		58.81	60.67	46.07	46.18	**45.47**
# best		0	8	12	13*	**17**
avg. diff. to best		14.58	16.45	1.85	1.96	**1.25**
max diff. to best		78.10	49.00	**10.70**	20.30	17.10
l_2-norm	CW	DF	LRA	PGD-100	FAB-10	FAB-100
avg. rob. acc.	45.09	56.10	36.97	44.94	36.41	**35.57**
# best	4	1	9	11	19*	**23**
avg. diff. to best	9.65	20.67	1.54	9.51	0.98	**0.13**
max diff. to best	65.40	91.40	13.60	64.80	8.40	**1.60**
l_1-norm		SF	EAD	PGD-100	FAB-10	FAB-100
avg. rob. acc.		64.47	35.79	49.51	33.26	**29.46**
# best		0	13	0	10*	**17**
avg. diff. to best		35.31	6.63	20.35	4.10	**0.30**
max diff. to best		95.90	58.40	74.00	21.80	**1.60**

Table 4.3: As in Table 4.2 we report statistics of the performance of different attacks on Restricted ImageNet (on the first 500 points of the validation set). In total we consider 5 thresholds $\times$ 3 models = 15 cases for each of the 3 norms.

statistics on Restricted ImageNet

	l_∞-norm				**l_2-norm**			**l_1-norm**		
	DF	DAA-10	PGD-10	FAB-10	DF	PGD-10	FAB-10	SF	PGD-5	FAB-5
avg. rob. acc.	35.61	38.44	**26.91**	27.83	45.69	**31.75**	33.24	71.31	40.64	**38.12**
# best	0	1	**13**	3	0	**14**	1	0	3	**12**
avg. diff. best	8.75	11.57	**0.04**	0.96	13.99	**0.04**	1.53	33.52	2.85	**0.33**
max diff. best	14.60	37.20	**0.40**	2.00	25.40	**0.60**	3.40	59.00	6.20	**2.40**

Evaluation metrics. The *robust accuracy* for a threshold ϵ is the classification accuracy when an adversary is allowed to change every test input with perturbations of l_p-norm smaller than ϵ in order to change the decision. Thus stronger attacks produce lower robust accuracy. For each model and dataset we fix five thresholds at which we compute the robust accuracy for each attack (we choose the thresholds so that the robust accuracy covers the range between clean accuracy and 0). We evaluate the attacks by the following statistics: i) **avg. rob. accuracy**: the mean of the robust accuracy achieved by the attack over all models and thresholds (lower is better), ii) **# best**: how many times the attack achieves the lowest robust accuracy (it is the most effective), iii) **avg. difference to best**: for each model/threshold we compute the difference between the robust accuracy of the attack and the best one across all the attacks, then we average over all models/thresholds, iv) **max difference to best**: as "avg. difference to best", but with the maximum difference instead of the average one.

Results. We summarize the results in Table 4.2 (MNIST and CIFAR-10 aggregated, as we used the same attacks) and Table 4.3 (Restricted ImageNet). Our FAB-attack achieves the best results in all statistics for every norm (with the only exception of "max diff. to best" in l_∞) on MNIST+CIFAR-10. In particular, while on l_∞ the "avg. robust accuracy" of PGD is not

far from that of FAB, the gap is large when considering l_2 and l_1 (we provide in Table 4.6 the aggregate statistics without the result on l_∞-AT MNIST, showing that FAB is still the best attack even if one leaves out this failure case of PGD). Interestingly, the second best attack in terms of average robust accuracy, is different for every norm (PGD for l_∞, LRA for l_2, EAD for l_1), which implies that FAB outperforms algorithms specialized in the individual norms. We also report the results of FAB-10, that is our attack with only 10 restarts, to show that FAB yields high quality results already with a low budget in terms of time/computational cost. In fact, FAB-10 has "avg. robust accuracy" better than or very close to that of the strongest versions of the other attacks (see below for a runtime analysis, where one observes that FAB-10 is the fastest attack when excluding the significantly worse DF and SF attacks). On Restricted ImageNet, FAB-attack gets the best results in all statistics for l_1, while for l_∞ and l_2 PGD performs on average better, but the difference in "avg. robust accuracy" is small. In general, both average and maximum *difference to best* of FAB-attack are small for all the datasets and norms, implying that it does not suffer from severe failures, which makes it an efficient, high quality technique to evaluate the robustness of classifiers for all l_p-norms.

Average perturbations size. In Table 4.4 we report the average l_p-norm of the adversarial perturbations found by the different attacks, computed on the originally correctly classified points on which the attack is successful. Note that we cannot show this statistic for the attacks which do not minimize the distance of the adversarial example to the clean input (PGD and DAA). FAB-attack produces also in this metric the best results in most of the cases, being the best for every model when considering l_∞ and l_2, and the best in 4 out of 6 cases in l_1 (lower values mean a stronger attack).

Resistance to gradient masking. It has been argued [Tramèr and Boneh, 2019] that models trained with first-order methods to be resistant wrt l_∞-attacks on MNIST (adversarial training) give a false sense of robustness in l_1 and l_2 due to *gradient masking*. This means that standard gradient-based methods like PGD have problems to find adversarial examples while they still exist. In contrast, FAB does not suffer from gradient masking. In Table 4.5 we see that it is extremely effective also wrt l_1 and l_2 on the l_∞-robust model, outperforming by a large margin the competitors. The reason is that FAB is not dependent on the norm of the gradient but just its direction matters for the definition of the hyperplane in (4.5). While we believe that resistance to gradient masking is a key property of a solid attack, we recompute the statistics of Table 4.2 excluding l_1 and l_2 attacks on the l_∞-AT model on MNIST in Table 4.6. FAB still achieves in most of the cases the best aggregated statistics, implying that our attack is effective whether or not the attacked classifier tends to "mask" the gradient.

We have shown in Sec. 4.1.3 that FAB-attack is invariant under rescaling of the classifier. We provide an example why this is a desirable property of an adversarial attack. We consider the defense proposed in Pang et al. [2020a], in particular their ResNet-110 (without adversarial training) for CIFAR-10. In Table 5 in Pang et al. [2020a] it is claimed that this model has a robust accuracy of 31.4% for $^8/_{255}$ obtained by a PGD attack on their new loss function, and that a standard PGD attack on the cross-entropy loss performs much worse. We test the performance of PGD on the cross-entropy loss, both using the original classifier and the same scaled down by a factor of 10^6. Moreover, we use the default step size $\epsilon/10$ together with $\epsilon/2$. The results are reported in Table 4.7. We can see that PGD on the original model yields more than 90% robust accuracy which confirms the statement in Pang et al. [2020a] about the cross-entropy loss being unsuitable for this case. However, PGD applied to the rescaled classifier reduces robust accuracy below 13%. The better step size $\epsilon/2$ decreases it to 2.5% which shows that tuning the step size is important for PGD. At the same time, FAB achieves a robust accuracy of 0.3% without any need of parameter tuning or rescaling of the classifier. This exemplifies the benefit of the scaling invariance of FAB. Moreover, as a side result this shows that the new loss alone in Pang et al. [2020a] is an ineffective defense.

Runtime comparison. DF and SF are much faster than the other attacks as their primary goal is to find as fast as possible adversarial examples, without emphasis on minimizing their norms, while LRA is rather expensive. PGD needs a forward and a backward pass of the network per iteration whereas FAB requires three passes for each iteration. Thus PGD is given 1.5 times

Table 4.4: We report mean l_p-norm of the adversarial perturbations found by the attacks (when successful, excluding the already misclassified points) for every model.

dataset	*model*	l_∞-**norm**		l_2-**norm**				l_1-**norm**	
		DF	FAB	DF	CW	LRA	FAB	EAD	FAB
MNIST	plain	0.078	**0.066**	1.13	1.01	**1.00**	**1.00**	6.38	**6.04**
	l_∞-AT	0.508	**0.326**	4.95	1.76	1.25	**1.12**	8.26	**3.36**
	l_2-AT	0.249	**0.170**	3.10	2.35	2.25	**2.24**	12.18	**12.16**
CIFAR-10	plain	0.008	**0.006**	0.28	**0.21**	0.22	**0.21**	3.01	**2.87**
	l_∞-AT	0.032	**0.024**	0.96	0.74	0.74	**0.73**	**5.79**	6.03
	l_2-AT	0.026	**0.019**	0.91	0.71	0.72	**0.70**	**7.94**	8.05

Table 4.5: Comparison of l_∞-, l_2- and l_1-attacks on an l_∞-robust model on MNIST. We report the accuracy on the test set if the attack is allowed to perturb the test points of ϵ in l_p-distance. The statistics are computed on the first 1000 points on the test set for l_∞ and l_1, on 500 points for l_2.

metric	ϵ	DF	DAA-1	DAA-50	PGD-1	PGD-10	PGD-100	FAB-1	FAB-10	FAB-100
l_∞	0.2	95.2	94.6	**93.7**	95.0	94.2	**93.7**	94.6	94.4	93.9
	0.25	94.7	92.7	**91.1**	93.1	91.8	91.4	93.3	92.1	91.7
	0.3	93.9	89.5	**87.2**	91.3	88.3	87.6	91.2	89.2	88.5
	0.325	92.5	72.1	**64.2**	74.9	68.4	64.7	86.2	83.1	81.3
	0.35	89.8	19.7	**11.7**	32.1	19.3	13.8	48.7	32.0	23.8
		CW	DF	LRA	PGD-1	PGD-10	PGD-100	FAB-1	FAB-10	FAB-100
l_2	1	88.8	94.6	73.6	92.2	90.8	89.8	84.2	70.6	**65.4**
	1.5	77.6	93.0	25.8	86.0	81.2	77.0	47.0	20.6	**12.2**
	2	64.4	91.6	3.2	77.8	67.0	57.8	15.6	1.8	**0.2**
	2.5	53.8	89.6	0.4	68.2	49.6	36.4	3.8	**0.0**	**0.0**
	3	46.8	84.6	**0.0**	59.8	29.6	13.4	1.4	**0.0**	**0.0**
		SparseFool	EAD	PGD-1	PGD-10	PGD-100	FAB-1	FAB-10	FAB-100	
l_1	2.5	96.8	92.2	94.1	93.7	93.6	88.0	74.3	**56.9**	
	5	96.5	76.0	90.9	88.9	88.2	78.3	39.4	**17.6**	
	7.5	96.4	49.5	85.2	81.4	79.0	66.1	19.8	**5.0**	
	10	96.4	27.4	80.2	73.5	70.3	49.3	11.9	**2.4**	
	12.5	96.4	14.6	74.9	65.6	58.7	35.6	7.7	**0.5**	

Table 4.6: We recompute the statistics reported in Table 4.2 for l_2 and l_1 attacks excluding the l_∞-AT model on MNIST from Madry et al. [2018] where PGD fails because of gradient masking.

l_2-**norm**	CW	DF	LRA	PGD-100	FAB-10	FAB-100
avg. rob. acc.	40.85	49.18	40.25	42.95	39.98	**39.57**
# best	4	1	8	11	14*	**18**
avg. diff. to best	1.44	9.78	0.84	3.54	0.57	**0.16**
max diff. to best	8.80	44.00	4.00	22.00	4.20	**1.60**
l_1-**norm**		SF	EAD	PGD-100	FAB-10	FAB-100
avg. rob. acc.		58.06	32.56	43.82	33.79	**32.06**
# best		0	**13**	0	5*	12
avg. diff. to best		26.36	0.87	12.12	2.10	**0.36**
max diff. to best		53.10	4.80	31.90	3.90	**1.60**

Table 4.7: We attack the ResNet-110 in Pang et al. [2020a] on CIFAR-10 at $\epsilon = 8/255$. The performance of the PGD attack on the cross entropy loss (CE) heavily depends on both scale of the classifier and the step size. In contrast, the scaling invariant FAB-attack works well even on the original (unscaled) model.

attack	*step size*	*robust accuracy*
PGD-CE	$\epsilon/10$	90.2%
PGD-CE	$\epsilon/2$	90.2%
PGD-CE rescaled	$\epsilon/10$	12.9%
PGD-CE rescaled	$\epsilon/2$	2.5%
FAB	-	**0.3%**

more iterations than FAB, so that overall they have same budget of forward/backward passes (and thus runtime). Below we report the runtimes (for 1000 points on MNIST and CIFAR-10, 50 on R-ImageNet) for the attacks as used in the experiments (if not specified otherwise, it includes all the restarts). For PGD and DAA this is the time for evaluating the robust accuracy at 5 thresholds, while for the other methods a single run is sufficient to compute the robust accuracy for all five thresholds. **MNIST**: DAA 11736s, PGD 3825s for l_∞/l_2 and 14106s for l_1, CW 944s, EAD 606s, FAB-10 161s, FAB-100 1613s. **CIFAR-10**: DAA 11625s, PGD 31900s for l_∞/l_2 and 70110s for l_1, CW 3691s, EAD 3398s, FAB-10 1209s, FAB-100 12093s. **R-ImageNet**: DAA 6890s, PGD 4738s for l_∞/l_2 and 24158s for l_1, FAB 2268s for l_∞/l_2 and 3146s for l_1 (note that for l_1 different numbers of restarts/iterations are used on R-ImageNet). We note that for PGD the robust accuracy for the five thresholds can be computed faster by exploiting the fact that points which are non-robust for a thresholds ϵ are also non-robust for thresholds larger than ϵ. However, even when taking this into account FAB-10 would still be significantly faster than PGD-100 and has better quality on MNIST and CIFAR-10. Moreover, when just considering a fixed number of thresholds, one can stop FAB-attack whenever it finds an adversarial example for the smallest threshold which also leads to a speed-up. However, in real world applications a full picture of robustness as a continuous function of the threshold is the most interesting evaluation scenario.

4.1.6 FAB-attack with a large number of classes

The standard algorithm of FAB-attack requires to compute at each iteration the Jacobian matrix of the classifier f wrt the input x and then the closest approximated decision hyperplane. The Jacobian matrix has dimension $K \times d$, recalling that K is the number of classes and d the input dimension. Although this can be in principle obtained with a single backward pass of the network, it becomes computationally expensive on datasets with many classes. Moreover, the memory consumption of FAB-attack increases with K. As a consequence, using FAB-attack in the normal formulation on datasets like ImageNet which has $K = 1000$ classes may be inefficient.

Then, we propose a *targeted* version of our attack which performs at each iteration the projection onto the linearized decision boundary between the original class and a fixed target class. This means that in (4.8) the hyperplane π_s is not selected via (4.7) as the closest one to the current iterate but rather $s \equiv t$, with t the target class used. Note that in practice we do not constrain the final outcome of the algorithm to be assigned to class t, but any misclassification is sufficient to have a valid adversarial example. The target class t is selected as the second most likely one according to the score given by the model to the target point, and if k restarts are allowed one can use the classes with the $k+1$ highest scores as target (excluding the correct one c). In this way, only the gradient of $f_t - f_c : \mathbb{R}^d \to \mathbb{R}$ needs to be computed, which is a cheaper operation than getting the full Jacobian of f and with computational cost independent of the total number of classes. We will exploit this targeted version of FAB-attack in Sec. 5.1.

4.1.7 Experimental details

Models. The *plain* and l_∞-AT models on MNIST are publicly available available[2] and consist of two convolutional and two fully-connected layers. The architecture of the CIFAR-10 models

[2] https://github.com/MadryLab/mnist_challenge

has 8 convolutional layers (with number of filters increasing from 96 to 384) and 2 dense layers and we make the classifiers available[3], while on Restricted ImageNet we use the models[4] (ResNet-50 [He et al., 2016a]) from Tsipras et al. [2019]. The models on MNIST achieve the following clean accuracy: *plain* 98.7%, l_∞-AT 98.5%, l_2-AT 98.6%. The models on CIFAR-10 achieve the following clean accuracy: *plain* 89.2%, l_∞-AT 79.4%, l_2-AT 81.2%.

Attacks. We use CW with 40 binary search steps, 10000 iterations and confidence 0, EAD with 1000 iterations, l_1 decision rule and $\beta = 0.05$. In both cases we set the parameters to achieve minimally (wrt l_2 for CW and l_1 for EAD) distorted adversarial examples. We could not use these methods on Restricted ImageNet since, to be compatible with the attacks from Papernot et al. [2018], it would be necessary to reimplement from scratch the models of Tsipras et al. [2019], as done in https://github.com/tensorflow/cleverhans/tree/master/cleverhans/model_zoo/madry_lab_challenges for a similar situation. For DAA we use 200 iterations for MNIST, 50 for the other datasets and, given a threshold ϵ, a step size of $\epsilon/30$ for MNIST, $\epsilon/10$ otherwise. We apply PGD with 150 iterations, except for the case of l_1 on Restricted ImageNet where we use 450 iterations. For FAB-attack we set 100 iterations, except for the case of l_1 on Restricted ImageNet where we use 300 iterations.

4.2 Square Attack

In the following we present Square Attack, a score-based adversarial attack, i.e. it can query the probability distribution over the classes predicted by a classifier but has no access to the underlying model. Moreover, unlike transfer-based black-box methods [Papernot et al., 2016a, Yan et al., 2019, Cheng et al., 2019, Du et al., 2020, Suya et al., 2019], it does not leverage any surrogate model similar to the target classifier to craft perturbations. The Square Attack exploits random search[5] [Rastrigin, 1963, Schumer and Steiglitz, 1968] which is one of the simplest approaches for black-box optimization. Due to a particular sampling distribution, it requires significantly fewer queries compared to the state-of-the-art black-box methods in the score-based threat model while outperforming them in terms of *success rate*, i.e. the percentage of successful adversarial examples. This is achieved by a combination of a particular initialization strategy and our square-shaped updates. We motivate why these updates are particularly suited to attack neural networks and provide convergence guarantees for a variant of our method. In an extensive evaluation with untargeted and targeted attacks, three datasets (MNIST, CIFAR-10, ImageNet), normal and robust models, we show that Square Attack outperforms state-of-the-art methods in the l_2- and l_∞-threat model.

4.2.1 General algorithmic scheme of the Square Attack

If $f : [0,1]^d \to \mathbb{R}^K$ is a classifier, we want to find adversarial points $\hat{x}$ with an l_p-norm bound of ϵ for x which satisfy

$$\arg\max_{k=1,\dots,K} f_k(\hat{x}) \neq y, \quad \|\hat{x} - x\|_p \leq \epsilon \quad \text{and} \quad \hat{x} \in [0,1]^d,$$

where we have added the additional constraint that $\hat{x}$ is an image. The task of finding $\hat{x}$ can be rephrased as solving the constrained optimization problem

$$\min_{\hat{x} \in [0,1]^d} L(f(\hat{x}), y), \quad \text{s.th.} \quad \|\hat{x} - x\|_p \leq \epsilon, \tag{4.14}$$

for a loss L. In our experiments, we use $L(f(\hat{x}), y) = f_y(\hat{x}) - \max_{k \neq y} f_k(\hat{x})$.

Square Attack is based on *random search* which is a well known iterative technique in optimization introduced by Rastrigin [1963]. The main idea of the algorithm is to sample a random update δ at each iteration, and to add this update to the current iterate $\hat{x}$ if it improves the objective function. Despite its simplicity, random search performs well in many situations [Zabinsky, 2010] and does not depend on gradient information from the objective function g.

[3] https://github.com/fra31/fab-attack
[4] Available at https://github.com/MadryLab/robust-features-code
[5] It is an iterative procedure different from random sampling inside the feasible region.

Algorithm 2: The Square Attack via random search

Input: classifier f, point $x \in \mathbb{R}^d$, image size w, number of color channels c, l_p-radius ϵ, label $y \in \{1, \dots, K\}$, number of iterations N

Output: approximate minimizer $\hat{x} \in \mathbb{R}^d$ of the problem stated in Eq. (4.14)

1 $\hat{x} \leftarrow init(x)$, $l^* \leftarrow L(f(x), y)$, $i \leftarrow 1$
2 **while** $i < N$ ***and*** $\hat{x}$ *is not adversarial* **do**
3 $h^{(i)} \leftarrow$ side length of the square to modify (according to some schedule)
4 $\delta \sim P(\epsilon, h^{(i)}, w, c, \hat{x}, x)$ (see Alg. 3 and 4 for the sampling distributions)
5 $\hat{x}_{\text{new}} \leftarrow$ Project $\hat{x} + \delta$ onto $\{z \in \mathbb{R}^d : \|z - x\|_p \leq \epsilon\} \cap [0, 1]^d$
6 $l_{\text{new}} \leftarrow L(f(\hat{x}_{\text{new}}), y)$
7 **if** $l_{new} < l^*$ **then** $\hat{x} \leftarrow \hat{x}_{\text{new}}$, $l^* \leftarrow l_{\text{new}}$;
8 $i \leftarrow i + 1$
9 **end**

Many variants of random search have been introduced [Matyas, 1965, Schumer and Steiglitz, 1968, Schrack and Choit, 1976], which differ mainly in how the random perturbation is chosen at each iteration (the original scheme samples uniformly on a hypersphere of fixed radius). For our goal of crafting adversarial examples we come up with two sampling distributions specific to the l_∞- and the l_2-attack (Sec. 4.2.2 and Sec. 4.2.3), which we integrate in the classic random search procedure. These sampling distributions are motivated by both how images are processed by neural networks with convolutional filters and the shape of the l_p-balls for different p. Additionally, since the considered objective is non-convex when using neural networks, a good initialization is particularly important. We then introduce a specific one for better query efficiency.

Our proposed scheme differs from classical random search by the fact that the perturbations $\hat{x} - x$ are constructed such that for every iteration they lie on the boundary of the l_∞- or l_2-ball before projection onto the image domain $[0, 1]^d$. Thus we are using the perturbation budget almost maximally at each step. Moreover, the changes are localized in the image in the sense that at each step we modify just a small fraction of contiguous pixels shaped into **squares**. Our overall scheme is presented in Alg. 2. First, the algorithm picks the side length $h^{(i)}$ of the square to be modified (step 3), which is decreasing according to an a priori fixed schedule. This is in analogy to the step size reduction in gradient-based optimization. Then in step 4 we sample a new update δ and add it to the current iterate (step 5). If the resulting loss (obtained in step 6) is smaller than the best loss so far, the change is accepted otherwise discarded. Since we are interested in query efficiency, the algorithm stops as soon as an adversarial example is found. The time complexity of the algorithm is dominated by the evaluation of $f(\hat{x}_{\text{new}})$, which is performed at most N times, with N total number of iterations. We plot the resulting adversarial perturbations in Fig. 4.5 and additionally in Sec. 4.2.10.

We note that previous works [Ilyas et al., 2019, Moon et al., 2019, Meunier et al., 2019] generate perturbations containing squares. However, while those use a fixed grid on which the squares are constrained, we optimize the position of the squares as well as the color, making our attack more flexible and effective. Moreover, unlike previous works, we motivate squared perturbations with the structure of the convolutional filters (see Sec. 4.2.5).

Size of the squares. Given images of size $w \times w$, let $p \in [0, 1]$ be the percentage of elements of x to be modified. The length h of the side of the squares used is given by the closest positive integer to $\sqrt{p \cdot w^2}$ (and $h \geq 3$ for the l_2-attack). Then, the initial p is the only free parameter of our scheme. With $N = 10000$ iterations available, we halve the value of p at $i \in \{10, 50, 200, 1000, 2000, 4000, 6000, 8000\}$ iterations. For different N we rescale the schedule accordingly.

4.2.2 The l_∞-Square Attack

Initialization. As initialization we use vertical stripes of width one where the color of each stripe is sampled uniformly at random from $\{-\epsilon, \epsilon\}^c$ (c number of color channels). We found that convolutional networks are particularly sensitive to such perturbations, see also Yin et al.

[2019] for a detailed discussion on the sensitivity of neural networks to various types of high frequency perturbations.

Sampling distribution. Similar to Moon et al. [2019] we observe that successful l_∞-perturbations usually have values $\pm\epsilon$ in all the components (note that this does not hold perfectly due to the image constraints $\hat{x} \in [0,1]^d$). In particular, it holds

$$\hat{x}_i \in \{\max\{0, x_i - \epsilon\}, \min\{1, x_i + \epsilon\}\}.$$

Our sampling distribution P for the l_∞-norm described in Alg. 3 selects sparse updates of $\hat{x}$ with $\|\delta\|_0 = h \cdot h \cdot c$ where $\delta \in \{-2\epsilon, 0, 2\epsilon\}^d$ and the non-zero elements are grouped to form a square. In this way, after the projection onto the l_∞-ball of radius ϵ (step 5 of Alg. 2) all components i for which $\epsilon \leq x_i \leq 1 - \epsilon$ satisfy $\hat{x}_i \in \{x_i - \epsilon, x_i + \epsilon\}$, i.e. differ from the original point x in each element either by ϵ or $-\epsilon$. Thus $\hat{x} - x$ is situated at one of the corners of the l_∞-ball (modulo the components which are close to the boundary). Note that all projections are done by clipping. Moreover, we fix the elements of δ belonging to the same color channel to have the same sign, since we observed that neural networks are particularly sensitive to such perturbations (see Sec. 4.2.9).

Algorithm 3: Sampling distribution P for l_∞-norm

Input: maximal norm ϵ, window size h, image size w, color channels c

Output: New update δ

$\delta \leftarrow$ array of zeros of size $w \times w \times c$
sample uniformly $r, s \in \{0, \dots, w-h\} \subset \mathbb{N}$
for $i = 1, \dots, c$ **do**
  $\rho \leftarrow Uniform(\{-2\epsilon, 2\epsilon\})$
  $\delta_{r+1:r+h,\ s+1:s+h,\ i} \leftarrow \rho \cdot \mathbb{1}_{h \times h}$
end

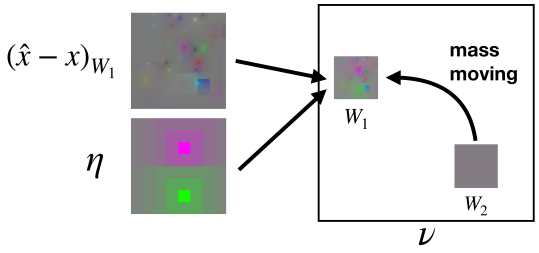

Figure 4.4: Perturbation of the l_2-attack

4.2.3 The l_2-Square Attack

Initialization. The l_2-perturbation is initialized by generating a 5×5 grid-like tiling by squares of the image, where the perturbation on each tile has the shape described next in the sampling distribution. The resulting perturbation $\hat{x} - x$ is rescaled to have l_2-norm ϵ and the resulting $\hat{x}$ is projected onto $[0,1]^d$ by clipping.

Sampling distribution. First, let us notice that the adversarial perturbations typically found for the l_2-norm tend to be much more localized than those for the l_∞-norm [Tsipras et al., 2019], in the sense that large changes are applied on some pixels of the original image, while many others are minimally modified. To mimic this feature we introduce a new update η which has two "centers" with large absolute value and opposite signs, while the other components have lower absolute values as one gets farther away from the centers, but never reaching zero (see Fig. 4.4 for one example with $h = 8$ of the resulting update η). In this way the modifications are localized and with high contrast between the different halves, which we found to improve the query efficiency. Concretely, we define $\eta^{(h_1,h_2)} \in \mathbb{R}^{h_1 \times h_2}$ (for some $h_1, h_2 \in \mathbb{N}_+$ such that $h_1 \geq h_2$) for every $1 \leq r \leq h_1$, $1 \leq s \leq h_2$ as

$$\eta^{(h_1,h_2)}_{r,s} = \sum_{k=0}^{M(r,s)} \frac{1}{(n+1-k)^2}, \quad \text{with} \quad n = \lfloor \frac{h_1}{2} \rfloor,$$

and $M(r,s) = n - \max\{|r - \lfloor \frac{h_1}{2} \rfloor - 1|, |s - \lfloor \frac{h_2}{2} \rfloor - 1|\}$. The intermediate square update $\eta \in \mathbb{R}^{h \times h}$ is then selected uniformly at random from either

$$\eta = \left(\eta^{(h,k)}, -\eta^{(h,h-k)}\right), \quad \text{with } k = \lfloor h/2 \rfloor, \tag{4.15}$$

Algorithm 4: Sampling distribution P for l_2-norm

Input: maximal norm ϵ, window size h, image size w, number of color channels c, current image $\hat{x}$, original image x

Output: New update δ

$\nu \leftarrow \hat{x} - x$

sample uniformly $r_1, s_1, r_2, s_2 \in \{0, \dots, w-h\}$

$W_1 := r_1 + 1 : r_1 + h, s_1 + 1 : s_1 + h$, $W_2 := r_2 + 1 : r_2 + h, s_2 + 1 : s_2 + h$

$\epsilon^2_{\text{unused}} \leftarrow \epsilon^2 - \|\nu\|_2^2$, $\eta^* \leftarrow \eta / \|\eta\|_2$ with η as in (4.15)

for $i = 1, \dots, c$ **do**

  $\rho \leftarrow Uniform(\{-1, 1\})$

  $\nu_{\text{temp}} \leftarrow \rho\eta^* + \nu_{W_1,i} / \|\nu_{W_1,i}\|_2$

  $\epsilon^i_{\text{avail}} \leftarrow \sqrt{\|\nu_{W_1 \cup W_2,i}\|_2^2 + \epsilon^2_{\text{unused}}/c}$

  $\nu_{W_2,i} \leftarrow 0$, $\nu_{W_1,i} \leftarrow (\nu_{\text{temp}} / \|\nu_{\text{temp}}\|_2)\epsilon^i_{\text{avail}}$

end

$\delta \leftarrow x + \nu - \hat{x}$

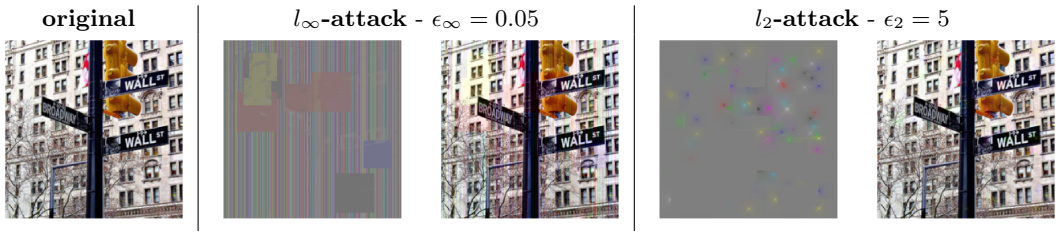

Figure 4.5: Visualization of the adversarial perturbations and examples found by the l_∞- and l_2-versions of the Square Attack on ResNet-50.

or its transpose (corresponding to a rotation of 90°).

Second, unlike l_∞-constraints, l_2-constraints do not allow to perturb each component independently from the others as the overall l_2-norm must be kept smaller than ϵ. Therefore, to modify a perturbation $\hat{x} - x$ of norm ϵ with localized changes while staying on the hypersphere, we have to "move the mass" of $\hat{x} - x$ from one location to another. Thus, our scheme consists in randomly selecting two squared windows in the current perturbation $\nu = \hat{x} - x$, namely ν_{W_1} and ν_{W_2}, setting $\nu_{W_2} = 0$ and using the budget of $\|\nu_{W_2}\|_2$ to increase the total perturbation of ν_{W_1}. Note that the perturbation of W_1 is then a combination of the existing perturbation plus the new generated η. We report the details of this scheme in Alg. 4 where step 4 allows to utilize the budget of l_2-norm lost after the projection onto $[0, 1]^d$. The update δ output by the algorithm is such that the next iterate $\hat{x}_{\text{new}} = \hat{x} + \delta$ (before projection onto $[0, 1]^d$ by clipping) belongs to the hypersphere $B_2(x, \epsilon)$ as stated in the following proposition.

Proposition 4.2.1 *Let δ be the output of Alg. 4. Then $\|\hat{x} + \delta - x\|_2 = \epsilon$.*

Proof. Let δ be the output of Alg. 4. We prove here that $\|\hat{x} + \delta - x\|_2 = \epsilon$. From Step 13 of Alg. 4, we directly have the equality $\|\hat{x} + \delta - x\|_2 = \|\nu\|_2$. Let ν^{old} be the update at the previous iteration, defined in Step 1 and $\overline{W_1 \cup W_2}$ the indices not belonging to $W_1 \cup W_2$. Then,

$$
\begin{aligned}
\|\nu\|_2^2 &= \sum_{i=1}^{c} \|\nu_{W_1 \cup W_2,i}\|_2^2 + \sum_{i=1}^{c} \left\|\nu_{\overline{W_1 \cup W_2},i}\right\|_2^2 = \sum_{i=1}^{c} \|\nu_{W_1,i}\|_2^2 + \sum_{i=1}^{c} \left\|\nu_{\overline{W_1 \cup W_2},i}\right\|_2^2 \\
&= \sum_{i=1}^{c} (\epsilon^i_{\text{avail}})^2 + \sum_{i=1}^{c} \left\|\nu_{\overline{W_1 \cup W_2},i}\right\|_2^2 = \sum_{i=1}^{c} \left\|\nu^{\text{old}}_{W_1 \cup W_2,i}\right\|_2^2 + \epsilon^2_{\text{unused}} + \sum_{i=1}^{c} \left\|\nu_{\overline{W_1 \cup W_2},i}\right\|_2^2 \\
&\overset{(i)}{=} \sum_{i=1}^{c} \left\|\nu^{\text{old}}_{W_1 \cup W_2,i}\right\|_2^2 + \epsilon^2_{\text{unused}} + \sum_{i=1}^{c} \left\|\nu^{\text{old}}_{\overline{W_1 \cup W_2},i}\right\|_2^2 = \left\|\nu^{\text{old}}\right\|_2^2 + \epsilon^2_{\text{unused}} \overset{(ii)}{=} \epsilon^2,
\end{aligned}
$$

where (i) holds since $\nu^{\text{old}}_{\overline{W_1 \cup W_2}} \equiv \nu_{\overline{W_1 \cup W_2}}$ as the modifications affect only the elements in the two

windows, and (ii) holds by the definition of ϵ_{unused} in Step 4 of Alg. 4. □

Next, we provide high-level theoretical justifications and empirical evidence regarding the algorithmic choices in Square Attack, with focus on the l_∞-version (the l_2-version is significantly harder to analyze).

4.2.4 Convergence analysis of random search

First, we want to study the convergence of the random search algorithm when considering an L-smooth objective function g (such as neural networks with activation functions like softplus, swish, ELU, etc) on the whole space $\mathbb{R}^d$ (without projection[6]) under the following assumptions on the update δ_t drawn from the sampling distribution P_t:

$$\mathbb{E}\|\delta_t\|_2^2 \leq \gamma_t^2 C \text{ and } \mathbb{E}|\langle \delta_t, v\rangle| \geq \tilde{C}\gamma_t \|v\|_2, \ \forall v \in \mathbb{R}^d, \tag{4.16}$$

where γ_t is the step size at iteration t, $C, \tilde{C} > 0$ some constants and $\langle \cdot, \cdot \rangle$ denotes the inner product. We obtain the following result, similar to existing convergence rates for zeroth-order methods [Nemirovsky and Yudin, 1983, Nesterov and Spokoiny, 2017, Duchi et al., 2015]:

Proposition 4.2.2 *Suppose that $\mathbb{E}[\delta_t] = 0$ and the assumptions in Eq. (4.16) hold. Then for step-sizes $\gamma_t = \gamma/\sqrt{T}$, we have*

$$\min_{t=0,\dots,T} \mathbb{E}\|\nabla g(x_t)\|_2 \leq \frac{2}{\gamma \tilde{C}\sqrt{T}}\left(g(x_0) - \mathbb{E}g(x_{T+1}) + \frac{\gamma^2 C L}{2}\right).$$

Proof. Using the L-smoothness of the function g, that is it holds for all $x, y \in \mathbb{R}^d$,

$$\|\nabla g(x) - \nabla g(y)\|_2 \leq L \|x - y\|_2 .$$

we obtain (see e.g. Boyd and Vandenberghe [2004]):

$$g(x_t + \delta_t) \leq g(x_t) + \langle \nabla g(x_t), \delta_t \rangle + \frac{L}{2}\|\delta_t\|_2^2,$$

and by definition of x_{t+1} we have

$$\begin{aligned} g(x_{t+1}) &\leq \min\{g(x_t), g(x_t + \delta_t)\} \\ &\leq g(x_t) + \min\{0, \langle \nabla g(x_t), \delta_t \rangle + \frac{L}{2}\|\delta_t\|_2^2\}. \end{aligned}$$

Using the definition of the min as a function of the absolute value ($2\min\{a, b\} = a + b - |a - b|$) yields

$$g(x_{t+1}) \leq g(x_t) + \frac{1}{2}\langle \nabla g(x_t), \delta_t \rangle + \frac{L}{4}\|\delta_t\|_2^2 - \frac{1}{2}|\langle \nabla g(x_t), \delta_t \rangle + \frac{L}{2}\|\delta_t\|_2^2|.$$

And using the triangular inequality ($|a + b| \geq |a| - |b|$), we have

$$g(x_{t+1}) \leq g(x_t) + \frac{1}{2}\langle \nabla g(x_t), \delta_t \rangle + \frac{L}{2}\|\delta_t\|_2^2 - \frac{1}{2}|\langle \nabla g(x_t), \delta_t \rangle|.$$

Therefore taking the expectation and using that $\mathbb{E}\delta_t = 0$, we get

$$\mathbb{E}g(x_{t+1}) \leq \mathbb{E}g(x_t) - \frac{1}{2}\mathbb{E}|\langle \nabla g(x_t), \delta_t \rangle| + \frac{L}{2}\mathbb{E}\|\delta_t\|_2^2.$$

Therefore, together with the assumptions in Eq. (4.16) this yields to

$$\mathbb{E}g(x_{t+1}) \leq \mathbb{E}g(x_t) - \frac{\tilde{C}\gamma_t}{2}\mathbb{E}\|\nabla g(x_t)\|_2 + \frac{LC\gamma_t^2}{2}.$$

[6] Nonconvex constrained optimization under noisy oracles is notoriously harder [Davis and Drusvyatskiy, 2019].

Algorithm 5: Sampling distribution P^{multiple} for l_∞-norm

Input: maximal norm ϵ, window size h, image size w, color channels c
Output: New update δ

$\delta \leftarrow$ array of zeros of size $w \times w \times c$
sample uniformly $r, s \in \{0, \dots, w\} \subset \mathbb{N}$
for $i = 1, \dots, c$ **do**
  $\delta_{r+1:r+h,\, s+1:s+h,\, i} \leftarrow Uniform(\{-2\epsilon, 2\epsilon\}^{h \times h})$
end

and thus

$$\mathbb{E}\,\|\nabla g(x_t)\|_2 \leq \frac{2}{\gamma_t \tilde{C}}\Big(\mathbb{E}g(x_t) - \mathbb{E}g(x_{t+1}) + \frac{LC\gamma_t^2}{2}\Big).$$

Thus for $\gamma_t = \gamma$ we have summing for $t = 0 : T$

$$\min_{0\leq i\leq T} \mathbb{E}\|\nabla g(x_i)\|_2 \leq \frac{1}{T}\sum_{t=0}^{T} \mathbb{E}\|\nabla g(x_t)\|_2 \leq \frac{2}{\tilde{C}\gamma T}\big[g(x_0) - \mathbb{E}g(x_{T+1}) + \frac{TLC\gamma^2}{2}\big].$$

We conclude setting the step-size to $\gamma = \Theta(1/\sqrt{T})$. □

This basically shows for T large enough one can make the gradient arbitrary small, meaning that the random search algorithm converges to a critical point of g (one cannot hope for much stronger results in non-convex optimization without stronger conditions).

Unfortunately, the second assumption in Eq. (4.16) does not directly hold for our sampling distribution P for the l_∞-norm. In fact, let us consider an update δ with a window size $h = 2$ and the direction $v \in \{-1, 1\}^{w\times w\times c}$ defined as

$$v^i_{k,l} = (-1)^{kl} \quad \text{for all } i, k, l.$$

It is easy to check that any update δ drawn from the sampling distribution P is orthogonal to this direction v:

$$\langle v, \delta\rangle = \sum_{i=1}^{c}\sum_{k=r+1}^{r+2}\sum_{l=s+1}^{s+2}(-1)^{kl} = c(-1+1-1+1) = 0.$$

Thus, $\mathbb{E}|\langle v, \delta\rangle| = 0$ and the assumptions in Eq. (4.16) do not hold. This means that the convergence analysis does not directly hold for the sampling distribution P.

However, such assumption holds for a similar sampling distribution, P^{multiple}, where each component of the update δ is drawn uniformly at random from $\{-2\epsilon, 2\epsilon\}$. Let us consider the sampling distribution P^{multiple} where different Rademacher $\rho_{k,l,i}$ are drawn for each pixel of the update window $\delta_{r+1:r+h,\, s+1:s+h,\, i}$. We present it in Alg. 5 with the convention that any subscript $k > w$ should be understood as $k - w$. This technical modification is greatly helpful to avoid side effect.

Let $v \in \mathbb{R}^{w\times w\times c}$ for which we have using the Khintchine inequality [Haagerup, 1981]:

$$\mathbb{E}|\langle \delta, v\rangle| = \mathbb{E}|\sum_{k=r+1}^{r+h}\sum_{l=s+1}^{s+h}\sum_{i=1}^{c}\delta^i_{k,l}v^i_{k,l}| \overset{(i)}{=} \mathbb{E}_{(r,s)}\mathbb{E}_\rho|\sum_{k=r+1}^{r+h}\sum_{l=s+1}^{s+h}\sum_{i=1}^{c}\delta^i_{k,l}v^i_{k,l}|$$
$$\overset{(ii)}{\geq} \frac{2\varepsilon}{\sqrt{2}}\mathbb{E}_{(r,s)}\|V_{(r,s)}\|_2 \overset{(iii)}{\geq} \sqrt{2}\varepsilon\|\mathbb{E}_{(r,s)}V_{(r,s)}\|_2 \geq \frac{\sqrt{2}\varepsilon h^2}{w^2}\|v\|_2,$$

where we define by $V_{(r,s)} = \{v^i_{k,l}\}_{k\in\{r+1,\dots,r+h\}, l\in\{s+1,\dots,s+h\}, i\in\{1,\dots,c\}}$ and (i) follows from the decomposition between the randomness of the Rademacher and the random window, (ii) follows from the Khintchine inequality and (iii) follows from Jensen inequality. In addition we have for the variance:

$$\mathbb{E}\|\delta\|_2^2 = \mathbb{E}_{(r,s)}\sum_{k=r+1}^{r+h}\sum_{l=s+1}^{s+h}\sum_{i=1}^{c}\mathbb{E}_\rho(\delta^i_{k,l})^2 = \mathbb{E}_{(r,s)}\sum_{k=r+1}^{r+h}\sum_{l=s+1}^{s+h}\sum_{i=1}^{c}4\varepsilon^2 = 4c\varepsilon^2h^2.$$

Thus the assumptions in Eq. (4.16) hold for the sampling distribution P^{multiple}. Moreover, while P^{multiple} performs worse than the distribution used in Alg. 3, we show in Sec. 4.2.9 that it already reaches state-of-the-art results.

4.2.5 Why square updates of equal sign?

Why squares? Previous works [Moon et al., 2019, Meunier et al., 2019] build their l_∞-attacks by iteratively adding square modifications. Likewise we change square-shaped regions of the image for both our l_∞- and l_2-attacks, but with the difference that we can sample any square subset of the input, while the grid of the possible squares is fixed in [Moon et al., 2019, Meunier et al., 2019]. This leads naturally to wonder why squares are superior to other shapes, e.g. rectangles.

Let us consider the l_∞-threat model, with bound ϵ, input space $\mathbb{R}^{d\times d}$ and a convolutional filter $w \in \mathbb{R}^{s\times s}$ with entries unknown to the attacker. Let $\delta \in \mathbb{R}^{d\times d}$ be the sparse update with $\|\delta\|_0 = k \geq s^2$ and $\|\delta\|_\infty \leq \epsilon$. We denote by $S(a,b)$ the index set of the rectangular support of δ with $|S(a,b)| = k$ and shape $a \times b$. We want to provide intuition why sparse square-shaped updates are superior to rectangular ones in the sense of reaching a maximal change in the activations of the first convolutional layer. Let $z = \delta * w \in \mathbb{R}^{d\times d}$ denote the output of the convolutional layer for the update δ. The l_∞-norm of z is the maximal componentwise change of the convolutional layer:

$$\begin{aligned}\|z\|_\infty &= \max_{u,v} |z_{u,v}| = \max_{u,v} \Big| \sum_{i,j=1}^{s} \delta_{u-\lfloor \frac{s}{2} \rfloor + i, v-\lfloor \frac{s}{2} \rfloor + j} \cdot w_{i,j} \Big| \\ &\leq \max_{u,v} \epsilon \sum_{i,j} |w_{i,j}| \mathbb{1}_{(u-\lfloor \frac{s}{2} \rfloor + i, v-\lfloor \frac{s}{2} \rfloor + j) \in S(a,b)},\end{aligned}$$

where elements with indices exceeding the size of the matrix are set to zero. Note that the indicator function attains 1 only for the non-zero elements of δ involved in the convolution to get $z_{u,v}$. Thus, to have the largest upper bound possible on $|z_{u,v}|$, for some (u,v), we need the largest possible amount of components of δ with indices in

$$C(u,v) = \Big\{ (u - \lfloor \frac{s}{2} \rfloor + i, v - \lfloor \frac{s}{2} \rfloor + j) : \; i,j = 1, \dots, s \Big\}$$

to be non-zero (that is in $S(a,b)$). Therefore, it is desirable to have the shape $S(a,b)$ of the perturbation δ selected so to maximize the number N of convolutional filters $w \in \mathbb{R}^{s\times s}$ which fit into the rectangle $a \times b$. Let $\mathcal{F}$ be the family of the objects that can be defined as the union of axis-aligned rectangles with vertices on $\mathbb{N}^2$, and $\mathcal{G} \subset \mathcal{F}$ be the squares of $\mathcal{F}$ of shape $s \times s$ with $s \geq 2$. We have the following proposition:

Proposition 4.2.3 *Among the elements of $\mathcal{F}$ with area $k \geq s^2$, those which contain the largest number of elements of $\mathcal{G}$ have*

$$N^* = (a - s + 1)(b - s + 1) + (r - s + 1)^+ \tag{4.17}$$

of them, with $a = \lfloor \sqrt{k} \rfloor$, $b = \lfloor \frac{k}{a} \rfloor$, $r = k - ab$ and $z^+ = \max\{z, 0\}$.

Proof. Let $x \in \mathcal{F}$, and $N(x)$ the number of elements of $\mathcal{G}$ that x contains. Let initialize x as a square of size $s \times s$, so that $N(x) = 1$. We then add iteratively the remaining $k - s^2$ unitary squares to x so to maximize $N(x)$. In order to get $N(x) = 2$ it is necessary to increase x to have size $s \times (s+1)$. At this point, again to get $N(x) = 3$ we need to add s squares to one side of x. However, if we choose to glue them so to form a rectangle $s \times (s+2)$, then $N(x) = 3$ and once more we need other s squares to increase N, which means overall $s^2 + 3s$ to achieve $N(x) = 4$. If instead we glue s squares along the longer side, with only one additional unitary square we get $N(x) = 4$ using $s^2 + 2s + 1 < s^2 + 3s$ unitary squares (as $s \geq 2$), with $x = (s+1) \times (s+1)$. Then, if the current shape of x is $a \times b$ with $a \geq b$, the optimal option is adding a unitary squares to have shape $a \times (b+1)$, increasing the count N of $a - s + 1$. This strategy can be repeated until the budget of k unitary squares is reached. Finally, since we start from the shape $s \times s$, then at each stage $b - a \in \{0, 1\}$, which means that the final a will be $\lfloor \sqrt{k} \rfloor$. A rectangle $a \times b$ in $\mathcal{F}$

contains $(a-s+1)(b-s+1)$ elements of $\mathcal{G}$. The remaining $k-ab$ squares can be glued along the longer side, contributing to $N(x)$ with $(k-ab-s+1)^+$. □

This proposition states that, if we can modify only k elements of δ, then shaping them to form (approximately) a square allows to maximize the number of pairs (u, v) for which $|S(a,b) \cap C(u,v)| = s^2$. If $k = l^2$ then $a = b = l$ are the optimal values for the shape of the perturbation update, i.e. the shape is exactly a square.

Why updates of equal sign? Proposition 4.2.2 underlines the importance of a large inner product $\mathbb{E}[|\langle \delta_t, \nabla g(x_t)\rangle|]$ in the direction of the gradients. This provides some intuition explaining why the update δ^{single}, where a single Rademacher is drawn for each window, is more efficient than the update δ^{multiple} where different Rademacher values are drawn. Following the observation that the gradients are often approximately piecewise constant [Ilyas et al., 2019], we consider, as a heuristic, a piecewise constant direction v for which we will show that

$$\mathbb{E}[|\langle \delta^{\text{single}}, v\rangle|] = \Theta(\|v\|_1) \quad \text{and} \quad \mathbb{E}[|\langle \delta^{\text{multiple}}, v\rangle|] = \Theta(\|v\|_2).$$

Therefore the directions sampled by our proposal are more correlated with the gradient direction and help the algorithm to converge faster. This is also verified empirically in our experiments (see the ablation study in Sec. 4.2.9).

Let us consider the direction $v \in \mathbb{R}^{w \times w}$ composed of different blocks $\{V_{(r,s)}\}_{(r,s) \in \{0,\dots,w/h\}}$ of constant sign. For this direction v we compare two different proposal P^{multiple} and P^{single} where we choose uniformly one random block (r, s) and we either assign a single Rademacher $\rho_{(r,s)}$ to the whole block (this is P^{single}) or we assign multiple Rademacher values $\{\rho_{(k,l)}\}_{k \in \{rh+1,\dots,(r+1)h\}, l \in \{sh+1,\dots,(s+1)h\}}$ (this is P^{multiple}). Using the Khintchine and Jensen inequalities similarly to Sec. 4.2.4, we have

$$\mathbb{E}|\langle \delta^{\text{multiple}}, v\rangle| = \mathbb{E}\Big|\sum_{k=rh+1}^{(r+1)h}\sum_{l=sh+1}^{(s+1)h} \delta_{k,l} v_{k,l}\Big| \geq \frac{2\varepsilon}{\sqrt{2}} \mathbb{E}_{(r,s)} \|V_{(r,s)}\|_2 \geq \frac{\sqrt{2}\varepsilon h^2}{w^2} \|v\|_2.$$

Moreover, we can show the following upper bound using the Khintchine inequality and the inequality between the l_1- and l_2-norms:

$$\begin{aligned}\mathbb{E}|\langle \delta^{\text{multiple}}, v\rangle| &= \mathbb{E}\Big|\sum_{k=rh+1}^{(r+1)h}\sum_{l=sh+1}^{(s+1)h} \delta_{k,l} v_{k,l}\Big| \leq 2\varepsilon \mathbb{E}_{(r,s)} \|V_{(r,s)}\|_2 \\ &= \frac{2\varepsilon h^2}{w^2} \sum_{r=1}^{w/h}\sum_{s=1}^{w/h} \|V_{(r,s)}\|_2 \leq \frac{2\varepsilon h}{w} \|v\|_2\end{aligned}$$

Thus, $\mathbb{E}[|\langle \delta^{\text{multiple}}, v\rangle|] = \Theta(\|v\|_2)$. For the update δ^{single} we obtain

$$\begin{aligned}\mathbb{E}|\langle \delta^{\text{single}}, v\rangle| &= \mathbb{E}\Big|\sum_{k=rh+1}^{(r+1)h}\sum_{l=sh+1}^{(s+1)h} \delta_{r,s} v_{k,l}\Big| = \mathbb{E}|\delta_{r,s}| \sum_{k=rh+1}^{(r+1)h}\sum_{l=sh+1}^{(s+1)h} v_{k,l}\Big| \\ &\overset{(i)}{=} 2\varepsilon \mathbb{E}_{(r,s)} \|V_{(r,s)}\|_1 = \frac{2\varepsilon h^2}{w^2} \|v\|_1\end{aligned}$$

where (i) follows from the fact the $V_{(r,s)}$ has a constant sign. We recover then the l_1-norm of the direction v, i.e. we conclude that $\mathbb{E}[|\langle \delta^{\text{single}}, v\rangle|] = \Theta(\|v\|_1)$.

This implies that for an approximately constant block $\mathbb{E}|\langle \delta^{\text{single}}, v\rangle|$ will be larger than $\mathbb{E}|\langle \delta^{\text{multiple}}, v\rangle|$. For example, in the extreme case of constant binary block $|V_{(r,s)}| = 11^\top$, we have

$$\mathbb{E}|\langle \delta^{\text{single}}, v\rangle| = 2\varepsilon h^2 >> \mathbb{E}|\langle \delta^{\text{multiple}}, v\rangle| \asymp 2\varepsilon h.$$

4.2.6 Experiments

In this section we show the effectiveness of the Square Attack. Here we concentrate on **untargeted** attacks since our primary goal is query efficient robustness evaluation, while the **targeted**

Table 4.8: Results of **untargeted** attacks on ImageNet with a limit of 10,000 queries. For the l_∞-attack we set the norm bound $\epsilon = 0.05$ and for the l_2-attack $\epsilon = 5$. Models: normally trained **I**: Inception v3, **R**: ResNet-50, **V**: VGG-16-BN. The Square Attack outperforms for both threat models all other methods in terms of success rate and query efficiency. The missing entries correspond to the results taken from the original paper where some models were not reported

Norm	Attack	Failure rate			Avg. queries			Med. queries		
		I	R	V	I	R	V	I	R	V
l_∞	Bandits	3.4%	1.4%	2.0%	957	727	394	218	136	36
	Parsimonious	1.5%	-	-	722	-	-	237	-	-
	DFO_c–CMA	0.8%	**0.0%**	0.1%	630	270	219	259	143	107
	DFO_d–Diag. CMA	2.3%	1.2%	0.5%	424	417	211	**20**	20	2
	SignHunter	1.0%	0.1%	0.3%	471	129	95	95	39	43
	Square Attack	**0.3%**	**0.0%**	**0.0%**	**197**	**73**	**31**	24	**11**	**1**
l_2	Bandits	9.8%	6.8%	10.2%	1486	939	511	660	392	196
	SimBA-DCT	35.5%	12.7%	7.9%	**651**	**582**	452	564	467	360
	Square Attack	**7.1%**	**0.7%**	**0.8%**	1100	616	**377**	**385**	**170**	**109**

Table 4.9: Query statistics for untargeted l_2-attacks computed for the points for which all three attacks are successful for fair comparison

Attack	Avg. queries			Med. queries		
	I	R	V	I	R	V
Bandits	536	635	398	368	314	177
SimBA-DCT	647	563	421	552	446	332
Square Attack	**352**	**287**	**217**	**181**	**116**	**80**

attacks are postponed to Sec. 4.2.10. We follow the standard setup [Ilyas et al., 2019, Meunier et al., 2019] of comparing black-box attacks on three ImageNet models in terms of success rate and query efficiency for the l_∞- and l_2-untargeted attacks. We provide more experimental details in Sec. 4.2.8, ablation studies in Sec. 4.2.9 and additional results in Sec. 4.2.10.

Evaluation on ImageNet. We compare the Square Attack to state-of-the-art score-based black-box attacks (without any extra information such as surrogate models) on three pretrained models in PyTorch [Paszke et al., 2019] (Inception v3, ResNet-50, VGG-16-BN) using 1,000 images from the ImageNet validation set. Unless mentioned otherwise, we use the code from the other papers with their suggested parameters. As it is standard in the literature, we give a budget of 10,000 queries per point to find an adversarial perturbation of l_p-norm at most ϵ. We report the *average* and *median* number of queries each attack requires to craft an adversarial example, together with the *failure rate*. All query statistics are computed only for successful attacks on the points which were originally correctly classified.

Tables 4.8 and 4.9 show that the Square Attack, despite its simplicity, achieves in all the cases (models and norms) the **lowest failure rate**, ($< 1\%$ everywhere except for the l_2-attack on Inception v3), and almost always requires **fewer queries** than the competitors to succeed. Fig. 4.6 shows the progression of the success rate of the attacks over queries, with an additional zoom on the first 200 queries. Even in the low query regime the Square Attack outperforms the competitors for both norms. Finally, we highlight that the only hyperparameter of our attack, p, regulating the size of the squares, is set for all the models to 0.05 for l_∞ and 0.1 for l_2-perturbations.

l_∞-attacks. We compare our attack to Bandits [Ilyas et al., 2019], Parsimonious [Moon et al., 2019], DFO_c / DFO_d [Meunier et al., 2019], and SignHunter [Al-Dujaili and O'Reilly, 2020]. In Table 4.8 we report the results of the l_∞-attacks with norm bound of $\epsilon = 0.05$. The Square Attack always has the lowest failure rate, notably 0.0% in 2 out of 3 cases, and the lowest query consumption. Interestingly, our attack has median equal 1 on VGG-16-BN, meaning that the proposed initialization is particularly effective for this model. The closest competitor in terms

of the *average* number of queries is SignHunter, which still needs on average between 1.8 and 3 times more queries to find adversarial examples and has a higher failure rate than our attack. Moreover, the median number of queries of SignHunter is much worse than for our method (e.g. 43 vs 1 on VGG). We note that although DFO$_c$–CMA is competitive to our attack in terms of *median* queries, it has a significantly higher failure rate and between 2 and 7 times worse average number of queries. Additionally, our method is also more effective in the low-query regime (Fig. 4.6) than other methods on all the models. For this comparison we include the success rates of the attacks from Meunier et al. [2019] (DFO$_c$–CMA–50 and DFO$_d$–Diag. CMA–30) and Shukla et al. [2019] (BayesAttack), obtained via personal communication directly from the authors, and were calculated on 500 and 10,000 randomly sampled points, respectively. The gap in the success rate gets larger in the range of 100-1000 queries for the more challenging Inception-v3 model, where we observe over 10% improvement in the success rate of Square Attack over all other methods including SignHunter. Our method also outperforms the BayesAttack in the low query regime, i.e. less than 200 queries, by approximately 20% on every model. We note that DFO$_d$–Diag. CMA–30 method is also quite effective in the low query regime showing results close to BayesAttack. However, it is also outperformed by our Square Attack.

l_2-attacks. We compare our attack to Bandits [Ilyas et al., 2019] and SimBA [Guo et al., 2019] for $\epsilon = 5$, while we do not consider SignHunter since it is not as competitive as for the l_∞-norm, and in particular worse than Bandits on ImageNet (see Fig. 2 in Al-Dujaili and O'Reilly [2020]). As Table 4.8 and Fig. 4.6 show, the Square Attack outperforms by a large margin the other methods in terms of failure rate, and achieves the lowest median number of queries for all the models and the lowest average one for VGG-16-BN. However, since it has a significantly lower failure rate, the statistics of the Square Attack are biased by the "hard" cases where the competitors fail. Then, we recompute the same statistics considering only the points where all the attacks are successful (Table 4.9). In this case, our method improves by at least 1.5× the average and by at least 2× the median number of queries. The l_2-Square Attack outperform the competitors particularly in the low query regime (Fig. 4.6). We note that the success rate of SimBA plateaus after some iteration. This happens due to the fact that their algorithm only adds orthogonal updates to the perturbation, and does not have any way to correct the greedy decisions made earlier. Thus, there is no progress anymore after the norm of the perturbation reaches the $\epsilon = 5$ (note that we used for SimBA the same parameters of the comparison between SimBA and Bandits in Guo et al. [2019]). Contrary to this, both Bandits and our attack constantly keep improving the success rate, although with a different speed.

4.2.7 Square Attack can be more accurate than white-box attacks

Here we test our attack on problems which are challenging for both white-box PGD and other black-box attacks. We use for evaluation *robust accuracy*, defined as the worst-case accuracy of a classifier when an attack perturbs each input in some l_p-ball. We show that our algorithm outperforms the competitors both on state-of-the-art robust models and defenses that induce different types of gradient masking. Thus, our attack is useful to evaluate robustness without introducing adaptive attacks designed for each model separately.

Table 4.10: On the robust models of Madry et al. [2018] and Zhang et al. [2019b] on MNIST l_∞-Square Attack with $\epsilon = 0.3$ achieves state-of-the-art (SOTA) results outperforming white-box attacks in terms of robust accuracy.

Model	SOTA	Square
AT	**88.13%**	88.25%
TRADES	93.33%	**92.58%**

Outperforming white-box attacks on robust models. The models obtained with the adversarial training (AT) of Madry et al. [2018] and TRADES [Zhang et al., 2019b] are standard benchmarks to test adversarial attacks, which means that many papers have tried to reduce their robust accuracy, without limit on the computational budget and primarily via white-box attacks. We test our l_∞-Square Attack on these robust models on MNIST at $\epsilon = 0.3$, using $p = 0.8$, 20k queries and 50 random restarts, i.e., we run our attack 50 times and consider it successful if any of the runs finds an adversarial example (Table 4.10). On the AT model Square

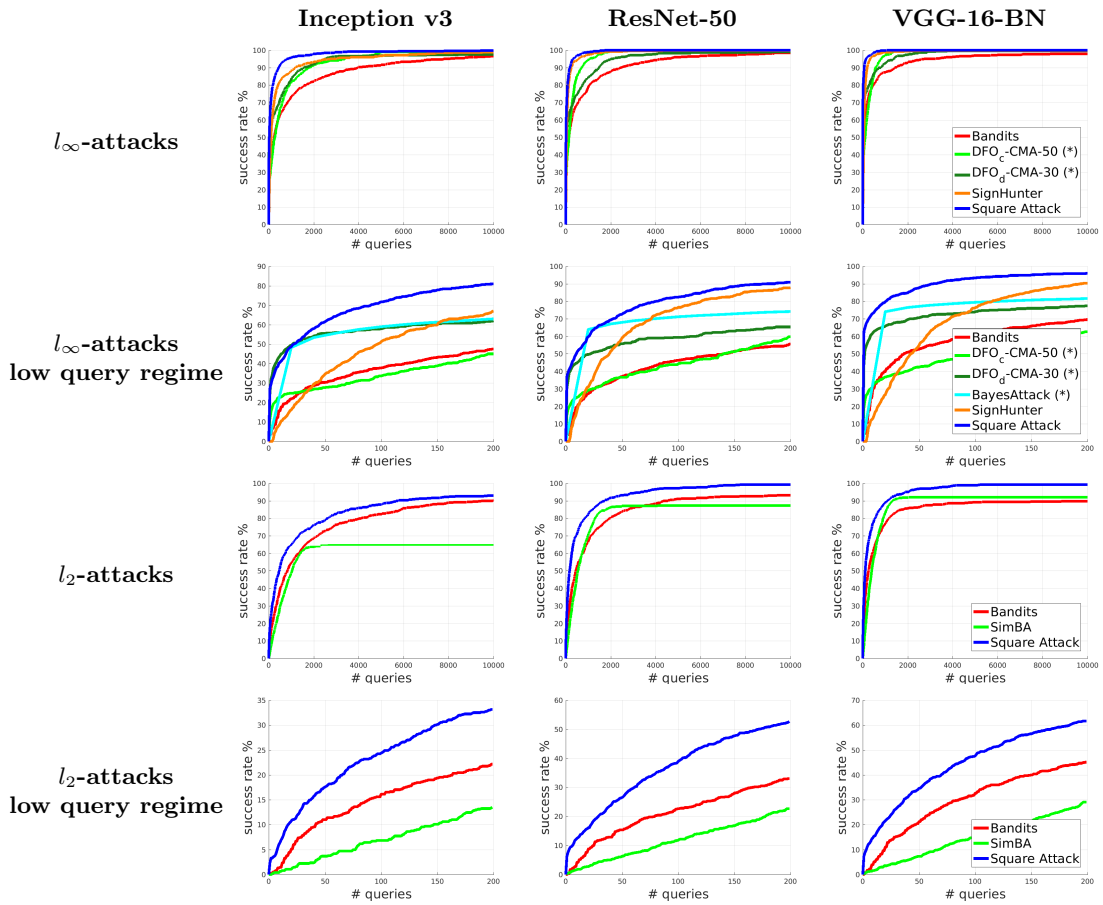

Figure 4.6: Success rate vs number of queries for different attacks on ImageNet on three standardly trained models. The low query regime corresponds to up to 200 queries, while the standard regime corresponds to 10,000 queries. * denotes the results obtained via personal communication with the authors and evaluated on 500 and 10,000 randomly sampled points for DFO [Meunier et al., 2019] and BayesAttack [Shukla et al., 2019] methods, respectively.

Attack is only 0.12% far from the *white-box* state-of-the-art, achieving the second best result (also outperforming the 91.47% of SignHunter by a large margin). On the TRADES benchmark, our method obtains a new SOTA[7] of 92.58% robust accuracy outperforming the white-box FAB attack introduced in Sec. 4.1. In Sec. 5.1 we will show that the Square Attack outperforms other white-box attacks on 9 out of 9 MNIST evaluated models. Thus, our black-box attack can be also useful for robustness evaluation of new defenses in the setting where gradient-based attacks require many restarts and iterations.

Resistance to gradient masking. In Table 4.11 we report the robust accuracy at different thresholds ϵ of the l_∞-adversarially trained models on MNIST of Madry et al. [2018] for the l_2-threat model. It is known that the PGD is ineffective since it suffers from gradient masking (see discussion and results in Sec. 4.1.5) Unlike PGD and other black-box attacks, our Square Attack does not suffer from gradient masking and yields robust accuracy close to zero for $\epsilon = 2.5$, with only a single run. Moreover, the l_2-version of SignHunter fails to accurately assess the robustness because the method optimizes only over the extreme points of the l_∞-ball of radius $\epsilon/\sqrt{d}$ embedded in the target l_2-ball.

Attacking Clean Logit Pairing and Logit Squeezing. These two l_∞ defenses proposed in Kannan et al. [2018] were broken in Mosbach et al. [2018]. However, Mosbach et al. [2018] needed

[7]This refers to SOTA results at the time of publication, and slightly better values have been later reported (see https://github.com/MadryLab/mnist_challenge and https://github.com/yaodongyu/TRADES.

Table 4.11: Robust accuracy wrt l_2 of the l_∞-adversarially trained models of Madry et al. [2018] at different thresholds ϵ. PGD is shown with 1, 10, 100 random restarts. The black-box attacks are given a 10k queries budget.

ϵ_2	White-box			Black-box			
	PGD_1	PGD_{10}	PGD_{100}	SignHunter	Bandits	SimBA	Square
2.0	79.6%	67.4%	**59.8%**	95.9%	80.1%	87.6%	**16.7%**
2.5	69.2%	51.3%	**36.0%**	94.9%	32.4%	75.8%	**2.4%**
3.0	57.6%	29.8%	**12.7%**	93.8%	12.5%	58.1%	**0.6%**

Table 4.12: l_∞-robustness of Clean Logit Pairing (CLP), Logit Squeezing (LSQ) [Kannan et al., 2018]. The Square Attack is competitive to white-box PGD with many restarts (R=10,000, R=100 on MNIST, CIFAR-10 resp.) and more effective than other black-box attacks.

ϵ_∞	Model	White-box		Black-box		
		PGD_1	PGD_R	Bandits	SignHunter	**Square**
0.3	CLP_{MNIST}	62.4%	**4.1%**	33.3%	62.1%	**6.1%**
	LSQ_{MNIST}	70.6%	**5.0%**	37.3%	65.7%	**2.6%**
16/255	CLP_{CIFAR}	2.8%	**0.0%**	14.3%	**0.1%**	0.2%
	LSQ_{CIFAR}	27.0%	**1.7%**	27.7%	13.2%	**7.2%**

up to 10k restarts of PGD which is computationally prohibitive. Using the publicly available models from Mosbach et al. [2018], we run the Square Attack with $p = 0.3$ and 20k query limit (results in Table 4.12). We obtain robust accuracy similar to PGD_R in most cases, but with a *single run*, i.e. without additional restarts. At the same time, although on some models Bandits and SignHunter outperform PGD_1, they on average achieve significantly worse results than the Square Attack. This again shows the utility of the Square Attack to accurately assess robustness.

4.2.8 Experimental details

In this section, we list the main hyperparameters and various implementation details for the experiments done in the main experiments (Sec. 4.2.6).

Experiments on ImageNet. For the untargeted Square Attack on the ImageNet models, we used $p = 0.05$ and $p = 0.1$ for the l_∞- and l_2- versions respectively. For Bandits, we used their code with their suggested hyperparameters (specified in the configuration files) for both l_∞ and l_2. For SignHunter, we used directly their code which does not have any hyperparameters (assuming that the finite difference probe δ is set to ϵ). For SimBA-DCT, we used the default parameters of the original code apart from the following, which are the suggested ones for each model: for ResNet-50 and VGG-16-BN `freq_dims` = 28, `order` = "strided" and `stride` = 7, for Inception v3 `freq_dims` = 38, `order` = "strided" and `stride` = 9. Notice that SimBA tries to minimize the l_2-norm of the perturbations but it does not have a bound on the size of the changes. Then we consider it successful when the adversarial examples produced have norm smaller than the fixed threshold ϵ. The results for all other methods were taken directly from the corresponding papers.

Evaluation of Bandits. The code of Bandits [Ilyas et al., 2018] does not have image standardization at the stage where the set of correctly points is determined.[8] As a result, the attack is run only on the set of points correctly classified by the network *without standardization*, although the network was trained on standardized images. We fix this bug, and report the results in Table 4.8 based on the fixed version of their code. We note that the largest difference of our evaluation compared to the l_∞ results reported in Appendix E of Ilyas et al. [2019] is obtained for the VGG-16-BN network: we get 2.0% failure rate while they reported 8.4% in their paper. Also, we note that the query count for Inception v3 we obtain is also better than reported in Ilyas et al. [2019]: 957 instead of 1117 with a slightly better failure rate. Our l_2 results also differ – we obtain a significantly lower failure rate (9.8%, 6.8%, 10.2% instead of 15.5%, 9.7%, 17.2% for

[8] See https://github.com/MadryLab/blackbox-bandits/issues/3

the Inception v3, ResNet-50, VGG-16-BN networks respectively) with improved average number of queries (1486, 939, 511 instead of 1858, 993, 594).

Comparison to white-box attacks. For the l_∞-Square Attack, we used $p = 0.3$ for all models on MNIST and CIFAR-10. For Bandits on MNIST and CIFAR-10 adversarially trained models we used `exploration` $= 0.1$, `tile_size` $= 16$, `gradient_iters` $= 1$ following Moon et al. [2019]. For the comparison of l_2-attacks on the l_∞-adversarially trained model of Madry et al. [2018] we used the Square Attack with the usual parameter $p = 0.1$. For Bandits we used the parameters `exploration` $= 0.01$, `tile_size` $= 28$, `gradient_iters` $= 1$, after running a grid search over the three of them (all the other parameters are kept as set in the original code). For SimBA we used the "pixel attack" with parameters `order` = "rand", `freq_dims` $= 28$, step size of 0.50, after a grid search on all the parameters.

4.2.9 Ablation study

We present ablation studies about the algorithmic choices for our attacks, and its properties. Unless stated otherwise, we perform these experiments on ImageNet with a standardly trained ResNet-50.

Algorithmic choices in l_∞-Square Attack. We perform an ablation study to show how the individual design decisions for the sampling distribution of the random search improve the performance of l_∞-Square Attack, confirming the theoretical arguments above. The comparison is done for an l_∞-threat model of radius $\epsilon = 0.05$ on $1,000$ test points with a query limit of $10,000$, and the results are shown in Table 4.13. Our sampling distribution is special in two aspects: i) we use localized update shapes in form of squares and ii) the update is constant in each color channel. First, one can observe that our update shape "square" performs better than "rectangle" as we discussed in the previous section, and it is significantly better than "random" (the same amount of pixels is perturbed, but selected randomly in the image). This holds both for c (constant sign per color channel) and $c \cdot h^2$ (every pixel and color channel is changed independently of each other) random signs, with an improvement in terms of average queries of 339 to 73 and 401 to 153 respectively. Moreover, with updates of the same shape, the constant sign over color channels is better than selecting it uniformly at random (improvement in average queries: 401 to 339 and 153 to 73). In total the algorithm with "square-c" needs more than $5\times$ less average queries than "random-$c \cdot h^2$", showing that our sampling distribution is the key to the high query efficiency of Square Attack.

The last innovation of our random search scheme is the initialization, crucial element of every non-convex optimization algorithm. Our method ("square-c") with the vertical stripes initialization improves over a uniform initialization on average by $\approx 25\%$ and, especially, median number of queries (more than halved). We want to also highlight that the sampling distribution "square-$c \cdot h^2$" for which we shown convergence guarantees in Sec. 4.2.4 performs already better in terms of the success rate and the median number of queries than the state of the art (see Sec. 4.2.6). Moreover, we test the performance of using a single random sign for all the elements for the update, "square-1", which turns out to be comparable to "square-$c \cdot h^2$", i.e. every component of the update has sign independently sampled, but worse than keeping the sign constant within each color channel ("square-c"). In order to implement update shape "rectangle", on every iteration and for every image we sample $\alpha, \beta \sim \text{Exp}(1)$ and take a rectangle with sides $\alpha \cdot s$ and $\beta \cdot s$, so that in expectation its area is equal to s^2, i.e. to the area of the original square. This update scheme performs significantly better than changing a random subset of pixels (93 vs 339 queries on average), but worse than changing squares (73 queries on average) as discussed in Sec. 4.2.5. Finally, we show the results with two more initialization schemes: horizontal stripes (instead of vertical), as well as initialization with randomly placed squares. While both solutions lead to the state-of-the-art query efficiency (83 and 90 queries on average) compared to the literature, they achieve worse results than the vertical stripes we choose for our Square Attack.

Algorithmic choices in l_2-Square Attack. We analyze in Table 4.13 the sensitivity of the l_2-attack to different choices of the shape of the update and initialization. In particular, we test

Table 4.13: An ablation study for the performance of the l_∞- and l_2-Square Attack under various algorithmic choices of the attack. The metrics are calculated on 1,000 ImageNet images for a ResNet-50 model. The last row represents our recommended setting. For all experiments we used the best performing p (0.05 for l_∞ and 0.1 for l_2)

l_∞ ablation study

Update shape	# random signs	Initialization	Failure rate	Avg. queries	Median queries
random	$c \cdot h^2$	vert. stripes	0.0%	401	48
random	$c \cdot h^2$	uniform rand.	0.0%	393	132
random	c	vert. stripes	0.0%	339	53
square	$c \cdot h^2$	vert. stripes	0.0%	153	15
square	1	vert. stripes	0.0%	129	18
rectangle	c	vert. stripes	0.0%	93	16
square	c	uniform rand.	0.0%	91	26
square	c	rand. squares	0.0%	90	20
square	c	horiz. stripes	0.0%	83	18
square	c	vert. stripes	0.0%	73	11

l_2 ablation study

Update	Initialization	Failure rate	Avg. queries	Median queries
η^{rand}	η^{rand}-grid	3.3%	1050	324
η^{single}	η^{single}-grid	0.7%	650	171
η	gaussian	0.4%	696	189
η	uniform	0.8%	660	187
η	vert. stripes	0.8%	655	186
η	η-grid	0.7%	616	170

an update with only one "center" instead of two, namely $\eta^{\text{single}} = \eta^{h,h}$ (following the notation of Eq. 4.15) and one, η^{rand}, where the step 7 in Alg. 4 is $\rho \leftarrow Uniform(\{-1, 1\}^{h \times h})$ instead of $\rho \leftarrow Uniform(\{-1, 1\})$, which means that each element of η is multiplied randomly by either -1 or 1 independently (instead of all elements multiplied by the same value). We can see that using different random signs in the update and initialization (η^{rand}) significantly (1.5× factor) degrades the results for the l_2-attack, which is similar to the observation made for the l_∞-attack. Alternatively to the grid described in Sec. 4.2.3, we consider as starting perturbation i) a random point sampled according to $Uniform(\{-\epsilon/\sqrt{d}, \epsilon/\sqrt{d}\}^{w \times w \times c})$, that is on the corners of the largest l_∞-ball contained in the l_2-ball of radius ϵ (uniform initialization), ii) a random position on the l_2-ball of radius ϵ (Gaussian initialization) or iii) vertical stripes similarly to what done for the l_∞-Square Attack, but with magnitude $\epsilon/\sqrt{d}$ to fulfill the constraints on the l_2-norm of the perturbation. We note that different initialization schemes do not have a large influence on the results of our l_2-attack, unlike for the l_∞-attack.

Sensitivity of Square Attack to the hyperparameter p. Fig. 4.7 shows the effect of varying the value p on the results of our attack. For the l_∞-threat model, we note that for *all* values of p (from 0.0125 to 0.4) we achieve 0.0% failure rate, as well as state-of-the-art query efficiency, i.e. we have the average number of queries below 140, and the median below 20 queries. Therefore, we conclude that the attack is robust to a wide range of p, which is an important property of a black-box attack. In fact, since the target model is unknown, and one aims at minimizing the number of queries needed to fool the model, doing even an approximate grid search over p might be prohibitively expensive. Similarly, we observe that the l_2-Square Attack is robust to different choices in the range between 0.05 and 0.4 showing approximately the same failure rate and query efficiency for all values of p in this range, while its performance degrades slightly for very small initial squares $p \in \{0.0125, 0.025\}$.

Stability under different random seeds. Here we study the stability of the Square Attack over the randomness in the algorithm, i.e. in the initialization, in the choice of the locations of square-shaped regions, and in the choice of the values in the updates δ. We repeat 10 times a subset of the experiments reported in Sec. 4.2.6 and Sec. 4.2.7 with different random seeds for our attack, and report all the metrics with standard deviations in Table 4.14. On ImageNet, we

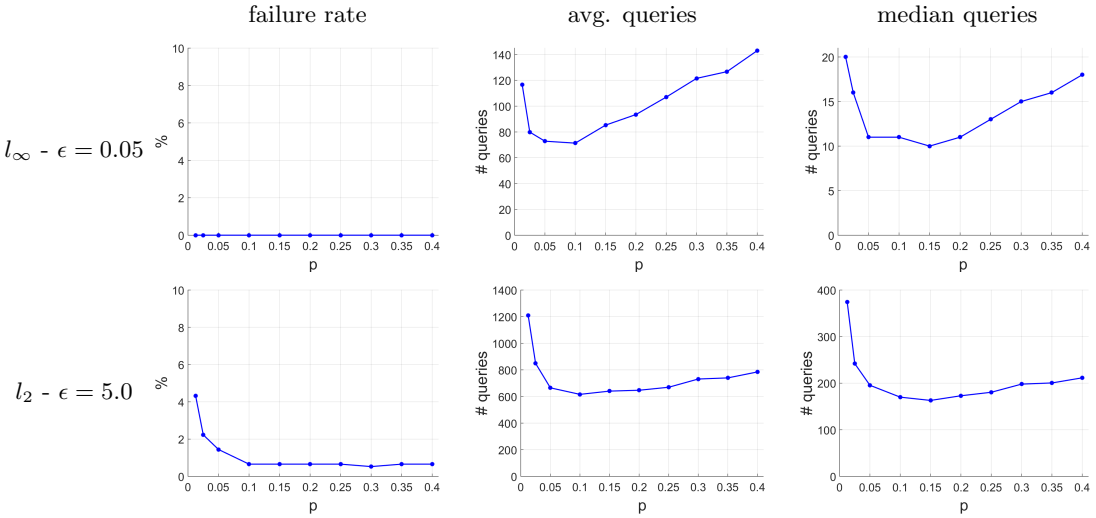

Figure 4.7: Sensitivity of the Square Attack to different choices of $p \in \{0.0125, 0.025, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4\}$, i.e. the initial fraction of pixels changed by the attack, on ImageNet for a ResNet-50 model

Table 4.14: Mean and standard deviation of the main performance metrics of the Square Attack across 10 different runs with different random seeds

ImageNet

Norm	ϵ	Failure rate	Avg. queries	Median queries
l_∞	0.05	$0.0\% \pm 0.0\%$	72 ± 2	11 ± 1
l_2	5	$0.6\% \pm 0.1\%$	638 ± 12	163 ± 8

MNIST

Norm	ϵ	Robust accuracy	Avg. queries	Median queries
l_∞	0.3	$87.0\% \pm 0.1\%$	299 ± 47	52 ± 7
l_2	2	$16.0\% \pm 1.4\%$	1454 ± 71	742 ± 78

evaluate the *failure rate* (over initially correctly classified points) and query efficiency on 1,000 images using ResNet-50. On MNIST, we evaluate the *robust accuracy* (i.e. the failure rate over all points) and query efficiency on 1,000 images using the l_∞-adversarially trained LeNet from Madry et al. [2018]. Note that unlike in Sec. 4.2.7 in both cases we use a single restart for the attack on MNIST, and we compute the statistics on 1,000 points instead of 10,000, thus the final results will differ.

On the ImageNet model, all these metrics are very concentrated for both the l_∞- and the l_2-norms. Moreover, we note that the standard deviations are much smaller than the gap between the Square Attack and the competing methods reported in Table 4.8. Thus we conclude that the results of the attack are stable under different random seeds. On the adversarially trained MNIST model from Madry et al. [2018], the robust accuracy is very concentrated showing only 0.1% and 1.4% standard deviations for the l_∞- and the l_2-norms respectively. Importantly, this is much less than the gaps to the nearest competitors reported in Tables 4.10 and 4.11. We also show query efficiency for this model, although for models with non-trivial robustness it is more important to achieve lower robust accuracy, and query efficiency on successful adversarial examples is secondary. We note that the standard deviation of the mean and median number of queries is higher than for ImageNet, particularly for the l_∞-ball of radius $\epsilon = 0.3$ where the robust accuracy is much higher than for the l_2-ball of radius $\epsilon = 2$. This is possibly due to the fact that attacking more robust models (within a certain threat model) is a more challenging task than, e.g., attacking standardly trained classifiers, as those used on ImageNet, which means that a favorable random initialization or perturbation updates can have more influence on the query efficiency.

Table 4.15: Results of **targeted** attacks on ImageNet for Inception-v3 model using 100k query limit for l_∞, 60k for l_2. The results for the competing methods for l_∞ are taken from Meunier et al. [2019], except SignHunter which we evaluated using their code.

Norm	Attack	Failure rate	Avg. queries	Median queries
l_∞	Bandits	7.5%	25341	18053
	SignHunter	1.1%	8814	5481
	Parsimonious	**0.0%**	7184	5116
	DFO_c – Diag. CMA	6.0%	6768	3797
	DFO_c – CMA	**0.0%**	6662	4692
	Square Attack	**0.0%**	**4584**	**2859**
l_2	Bandits	**24.5%**	20489	17122
	SimBA-DCT	25.5%	30576	30180
	Square Attack	33.5%	**19794**	**15946**

4.2.10 Additional results

Targeted Square Attack. While in Sec. 4.2.6 we considered only untargeted attacks, here we report the results of the different attacks in the *targeted* scenario. In order to adapt our scheme to targeted attacks, where one first choose a target class t and then tries to get the model f to classify a point x as t, we need to modify the loss function L which is minimized (see Eq. (4.14)). For the untargeted attacks we used the margin-based loss $L(f(x), y) = f_y(x) - \max_{k \neq y} f_k(x)$, with y the correct class of x. This loss could be straightforwardly adapted to the targeted case as $L(f(x), t) = -f_t(x) + \max_{k \neq t} f_k(x)$. However, in practice we observed that this loss leads to suboptimal query efficiency. We hypothesize that the drawback of the margin-based loss in this setting is that the maximum over $k \neq t$ is realized by different k at different iterations, and then the changes applied to the image tend to cancel each other. We observed this effect particularly on ImageNet which has a very high number of classes. Instead, we use here as objective function the cross-entropy loss on the target class, defined as

$$L(f(x), t) = -f_t(x) + \log \left(\sum_{i=1}^{K} e^{f_i(x)} \right). \tag{4.18}$$

Minimizing L is then equivalent to maximizing the confidence of the classifier in the target class. Notice that in Eq. (4.18) the scores of all the classes are involved, so that it increases the *relative* weight of the target class respect to the others, making the targeted attacks more effective.

Comparison in the targeted scenario. We present the results for targeted attacks on ImageNet for Inception-v3 model in Table 4.15. We calculate the statistics on 1,000 images (the target class is randomly picked for each image) with query limit of 100,000 for l_∞ and on 200 points and with query limit 60,000 for l_2, as this one is more expensive computationally because of the lower success rate, with the same norm bounds ϵ used in the untargeted case. We use the Square Attack with $p = 0.01$ for l_∞ and $p = 0.02$ for l_2. The results for the competing methods for l_∞ are taken from Meunier et al. [2019], except SignHunter which was not evaluated in the targeted setting before, thus we performed the evaluation using their code on 100 test points using the cross-entropy as the loss function. For l_2, we use Bandits with the standard parameters used in the untargeted scenario, while we ran a grid search over the step size of SimBA (we set it to 0.03) and keep the other hyperparameters as suggested for the Inception-v3 model. The targeted l_∞-Square Attack achieves 100% success rate and requires 1.5 times fewer queries on average than the nearest competitor from Meunier et al. [2019], showing that even in the *targeted* scenario our simple scheme outperforms the state-of-the-art methods. On the other hand, our l_2-attack suffers from higher failure rate than the competitors, but achieves lower average and median number of queries although on a different number of successful points.

Analysis of adversarial examples that require more queries. Here we provide more visualizations of adversarial perturbations generated by the untargeted Square Attack for $\epsilon = 0.05$ on ImageNet. We analyze here the inputs that require more queries to be misclassified. We

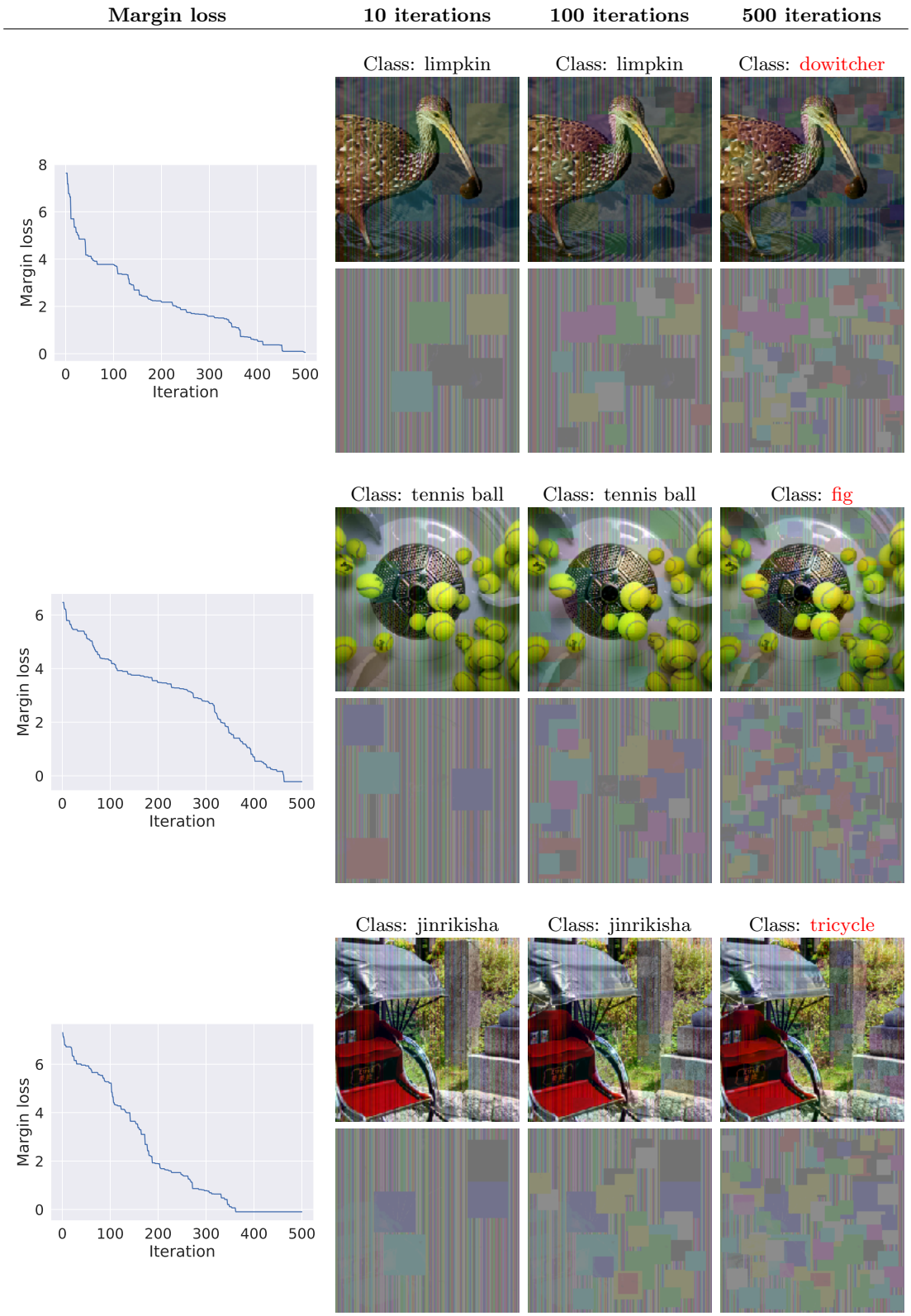

Figure 4.8: Visualization of adversarial examples for which the untargeted l_∞ Square Attack requires more queries. We visualize adversarial examples and perturbations after 10, 100 and 500 iterations of the attack. The experiment is done on ImageNet using ResNet-50 for $\epsilon_\infty = 0.05$. Note that a misclassification is achieved when the margin loss becomes negative

present the results in Fig. 4.8 where we plot adversarial examples after 10, 100 and 500 iterations of our attack. First, we note that a misclassification is achieved when the margin loss becomes negative. We can observe that the loss decreases gradually over iterations, and a single update only rarely leads to a significant decrease of the loss. As the attack progresses over iterations, the size of the squares is reduced according to our piecewise-constant schedule leading to more refined perturbations since the algorithm accumulates a larger number of square-shaped updates.

4.3 Auto-PGD: a budget-aware step size-free variant of PGD

In this section we introduce Auto-PGD, a white-box attack for fixed-size perturbations, which aims at providing an adaptive variant of the commonly used PGD attack [Madry et al., 2018] which does not require parameter tuning when using different classifiers, datasets or threat models.

PGD algorithm. Projected Gradient Descent (PGD) [Madry et al., 2018] is the most popular white-box method to generate norm-bounded adversarial perturbations. Similarly to Square Attack (Sec. 4.2), it aims at finding adversarial points z for a clean point x_{orig} with label c such that

$$\arg\max_{k=1,\dots,K} f_k(z) \neq c, \quad \|x_{\text{orig}} - z\|_p \leq \epsilon \quad \text{and} \quad z \in [0,1]^d,$$

by solving the optimization problem

$$\max_{z \in \mathbb{R}^d} L(f(z), c) \quad \text{such that} \quad \|x_{\text{orig}} - z\|_p \leq \epsilon, \; z \in [0,1]^d \tag{4.19}$$

for some objective function L which induces misclassification. PGD approximately solves (4.19) with the iterative scheme

$$x^{(k+1)} = P_{\mathcal{S}}\left(x^{(k)} + \eta^{(k)} \nabla f(x^{(k)})\right),$$

where $\eta^{(k)}$ is the step size at iteration k, $\mathcal{S}$ denotes the feasible set, in our case $\{z \in \mathbb{R}^d : \|x_{\text{orig}} - z\|_p \leq \epsilon\} \cap [0,1]^d$, and $P_{\mathcal{S}}$ is the projection onto $\mathcal{S}$. In the formulation of Madry et al. [2018], $\eta^{(k)} = \eta$ for every k, i.e. the step size is fixed, and as initial point $x^{(0)}$ either x_{orig} or $x_{\text{orig}} + \zeta$ is used, where ζ is randomly sampled such that $x^{(0)}$ satisfies the constraints. Moreover, instead of the plain gradient, one typically uses steepest descent in the given threat model (e.g. for l_∞ the sign of the gradient) as update direction.

Weaknesses in PGD algorithm. We identify three weaknesses in the standard formulation of the PGD-attack and how it is used in the context of adversarial robustness. First, the **fixed step size** is suboptimal, as even for convex problems this does not guarantee convergence, and the performance of the algorithm is highly influenced by the choice of its value, see e.g. Mosbach et al. [2018]. Second, the overall scheme is in general **agnostic of the budget** given to the attack: as we show, the loss plateaus after a few iterations, except for extremely small step sizes, which however do not translate into better results. As a consequence, judging the strength of an attack by the number of iterations is misleading, see also Carlini et al. [2019a]. Finally, the algorithm is **unaware of the trend**, i.e. does not consider whether the optimization is evolving successfully and is not able to react to this.

4.3.1 Auto-PGD (APGD) algorithm

In our automatic scheme we aim at fixing the weaknesses of PGD highlighed above. The main idea is to partition the available N_{iter} iterations in an initial exploration phase, where one searches the feasible set for *good* initial points, and an exploitation phase, during which one tries to maximize the knowledge so far accumulated. The transition between the two phases is managed by progressively reducing the step size. In fact, a large step size allows to move quickly in $\mathcal{S}$, whereas a smaller one more eagerly maximizes the objective function locally. However, the

Algorithm 6: APGD

Input: f, S, $x^{(0)}$, η, N_{iter}, $W = \{w_0, \dots, w_n\}$
Output: $x_{\max}$, $f_{\max}$

$x^{(1)} \leftarrow P_{\mathcal{S}}\left(x^{(0)} + \eta \nabla f(x^{(0)})\right)$
$f_{\max} \leftarrow \max\{f(x^{(0)}), f(x^{(1)})\}$
$x_{\max} \leftarrow x^{(0)}$ **if** $f_{\max} \equiv f(x^{(0)})$ **else** $x_{\max} \leftarrow x^{(1)}$
for $k = 1, \dots, N_{iter}-1$ **do**
  $z^{(k+1)} \leftarrow P_{\mathcal{S}}\left(x^{(k)} + \eta \nabla f(x^{(k)})\right)$
  $x^{(k+1)} \leftarrow P_{\mathcal{S}}\left(x^{(k)} + \alpha(z^{(k+1)} - x^{(k)}) + (1-\alpha)(x^{(k)} - x^{(k-1)})\right)$
  if $f(x^{(k+1)}) > f_{max}$ **then**
    $x_{\max} \leftarrow x^{(k+1)}$ and $f_{\max} \leftarrow f(x^{(k+1)})$
  end
  if $k \in W$ **then**
    if *Condition 1* ***or*** *Condition 2* **then**
      $\eta \leftarrow \eta/2$ and $x^{(k+1)} \leftarrow x_{\max}$
    end
  end
end

reduction of the step size is not a priori scheduled, but rather governed by the trend of the optimization: if the value of the objective grows sufficiently fast, then the step size is most likely proper, otherwise it is reasonable to reduce it. While the update step in APGD is standard, what distinguishes our algorithm from usual PGD is the choice of the step size across iterations, which is adapted to the overall budget and to the progress of the optimization, and that, once the step size is reduced, the maximization restarts from the best point so far found. We summarize our scheme in Alg. 6 and analyze the main features in the following.

Gradient ascent step. The update of APGD closely follows the classic algorithm and only adds a momentum term. Let $\eta^{(k)}$ be the step size at iteration k, then the update step is

$$\begin{aligned} z^{(k+1)} &= P_{\mathcal{S}}\left(x^{(k)} + \eta^{(k)} \nabla f(x^{(k)})\right) \\ x^{(k+1)} &= P_{\mathcal{S}}\left(x^{(k)} + \alpha \cdot (z^{(k+1)} - x^{(k)}) + (1-\alpha) \cdot (x^{(k)} - x^{(k-1)})\right), \end{aligned} \tag{4.20}$$

where $\alpha \in [0, 1]$ (we use $\alpha = 0.75$) regulates the influence of the previous update on the current one. Since in the early iterations of APGD the step size is particularly large, we want to keep a bias from the previous steps.

Step size selection. We start with step size $\eta^{(0)}$ at iteration 0 (we fix $\eta^{(0)} = 2\epsilon$), and given a budget of N_{iter} iterations, we identify checkpoints $w_0 = 0, w_1, \dots, w_n$ at which the algorithm decides whether it is necessary to halve the current step size. We have two conditions:

1. $\sum_{i=w_{j-1}}^{w_j - 1} \mathbf{1}_{f(x^{(i+1)}) > f(x^{(i)})} < \rho \cdot (w_j - w_{j-1})$,
2. $\eta^{(w_{j-1})} \equiv \eta^{(w_j)}$ and $f_{\max}^{(w_{j-1})} \equiv f_{\max}^{(w_j)}$,

where $f_{\max}^{(k)}$ is the highest objective value found in the first k iterations. If one of the conditions is true, then step size at iteration $k = w_j$ is halved and $\eta^{(k)} := \eta^{(w_j)}/2$ for every $k = w_j + 1, \dots, w_{j+1}$. More explicitly,

Condition 1: counts in how many cases since the last checkpoint w_{j-1} the update step has been successful in increasing f. If this happened for at least a fraction ρ of the total update steps, then the step size is kept as the optimization is proceeding properly (we use $\rho = 0.75$).

Condition 2: holds true if the step size was not reduced at the last checkpoint *and* there has been no improvement in the best found objective value since the last checkpoint. This prevents getting stuck in potential cycles.

Restart from the best point. If at a checkpoint w_j the step size gets halved, then we set $x^{(w_j+1)} := x_{\max}$, that is we restart at the point attaining the highest objective $f_{\max}$ so far. This makes sense as reducing η leads to a more localized search, and this should be done in a neighborhood of the current best candidate solution.

Exploration vs exploitation. We want the algorithm to transit gradually from exploring the whole feasible set $\mathcal{S}$ to a local optimization. This transition is regulated by progressively reducing the step size and by the choice of when to decrease it, i.e. the checkpoints w_j. In practice, we want to allow a relatively long initial exploration stage and then possibly update the step size more often moving toward exploitation. In fact, with smaller step sizes the improvements in the objective function are likely more frequent but also of smaller magnitude, while the importance of taking advantage of the whole input space is testified by the success of random restarts in the usual PGD-attack. We fix the checkpoints as $w_j = \lceil p_j N_{\text{iter}} \rceil \leq N_{\text{iter}}$, with $p_j \in [0, 1]$ defined as $p_0 = 0$, $p_1 = 0.22$ and

$$p_{j+1} = p_j + \max\{p_j - p_{j-1} - 0.03, 0.06\}.$$

Note that the period length $p_{j+1} - p_j$ is reduced in each step by 0.03 but they have at least a minimum length of 0.06.

While the proposed scheme has a few parameters which could be adjusted, we fix them to the values indicated so that the **only free variable is the budget** N_{iter}.

4.3.2 Comparison of APGD and PGD algorithms

We compare our APGD to PGD with and without Momentum in terms of achieved CE loss and robust accuracy, focusing here on l_∞-attacks with perturbation size ϵ. We attack the robust models on MNIST and CIFAR-10 from Madry et al. [2018] and Zhang et al. [2019b]. We run 1000 iterations of PGD with and without Momentum with step sizes ϵ/t with $t \in \{0.5, 1, 2, 4, 10, 25, 100\}$, and APGD with a budget of $N_{\text{iter}} \in \{25, 50, 100, 200, 400, 1000\}$ iterations. In Fig. 4.9 (comparison to PGD with Momentum) and Fig. 4.10 (comparison to plain PGD) we show the evolution of the current best average cross-entropy loss and robust accuracy (i.e. the percentage of points for which the attack could not find an adversarial example) for 1000 points of the test as a function of iterations. In all cases APGD achieves the highest loss (higher is better as it is a maximization problem) and this holds for any budget of iterations. Similarly, APGD attains always the lowest (better) robust accuracy and thus is the stronger adversarial attack on these models (see Sec. 4.3.4 for a comparison across more models and different losses). One can observe the adaptive behaviour of APGD: when the budget of iterations is larger the value of the objective (the CE loss) increases more slowly, but reaches higher values in the end. This is due to the longer exploration phase, which sacrifices smaller improvements to finally get better results. In contrast, the runs of PGD, both with and without Momentum, tend to plateau at suboptimal values, regardless of the choice of the step size.

4.3.3 Difference of Logits Ratio loss

If x has correct class y, the cross-entropy loss at x is

$$\text{CE}(x, y) = -\log p_y = -z_y + \log\Big(\sum_{j=1}^{K} e^{z_j}\Big), \tag{4.21}$$

with $p_i = e^{z_i} / \sum_{j=1}^{K} e^{z_j}$, $i = 1, \dots, K$, which is invariant to shifts of the logits z but not to rescaling, similarly to its gradient wrt x, given by

$$\nabla_x \text{CE}(x, y) = (-1 + p_y)\, \nabla_x z_y + \sum_{i \neq y} p_i \nabla_x z_i. \tag{4.22}$$

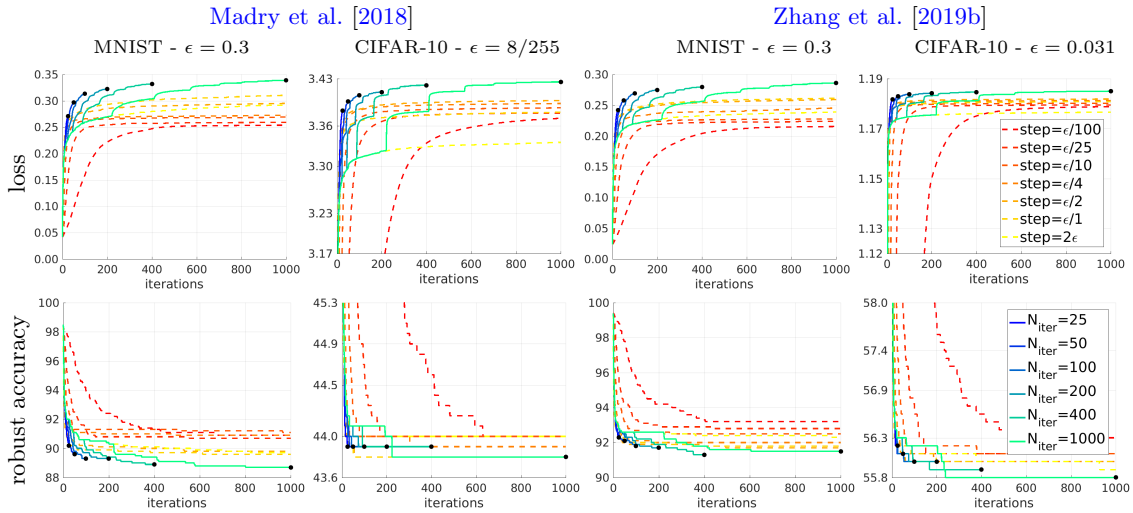

Figure 4.9: **PGD with Momentum vs APGD:** best cross-entropy loss (top) and robust accuracy (bottom) obtained so far as function of iterations for the models of Madry et al. [2018] and TRADES [Zhang et al., 2019b] for PGD with a momentum term ($\alpha = 0.75$, as used in APGD) (dashed lines) with different fixed step sizes (always 1000 iterations) and APGD (solid lines) with different budgets of iterations. APGD outperforms PGD with Momentum for every budget of iterations in achieved loss and almost always in robust accuracy.

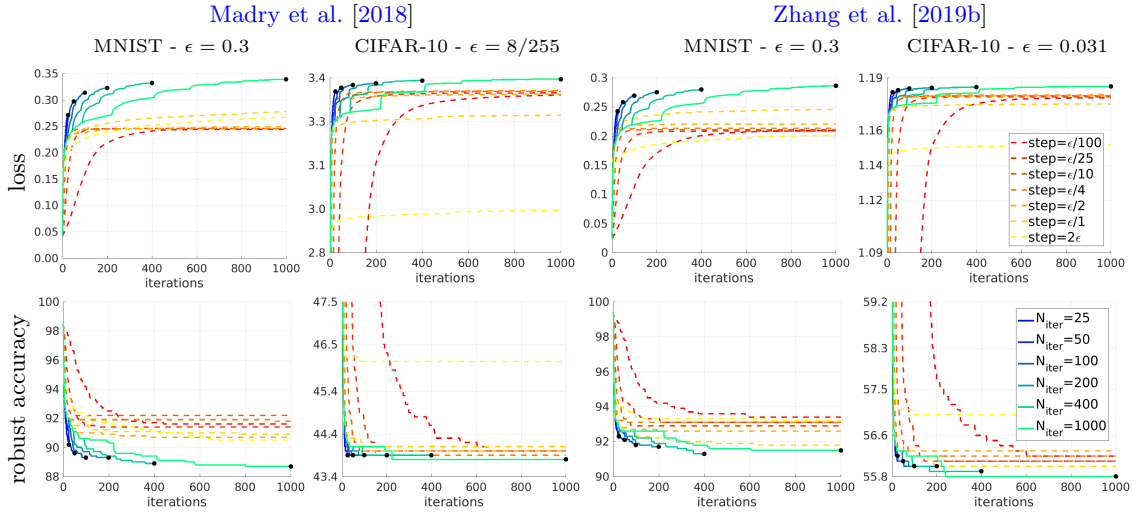

Figure 4.10: **PGD vs APGD:** best cross-entropy loss (top) and robust accuracy (bottom) so far found as function of iterations for the models of Madry et al. [2018] and TRADES [Zhang et al., 2019b] for PGD (dashed lines) with different fixed step sizes (always 1000 iterations) and APGD (solid lines) with different budgets of iterations. APGD outperforms PGD for every budget of iterations in robust accuracy.

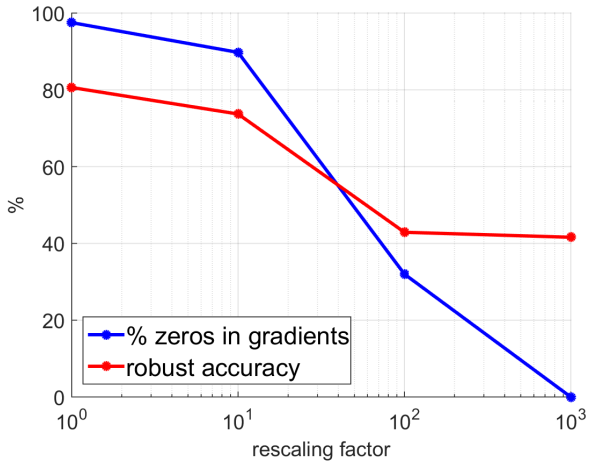

Figure 4.11: Percentage of zeros in the gradients and robust accuracy, computed by PGD on the CE loss, of the classifiers g/α, where g is the CIFAR-10 model of Atzmon et al. [2019] and α a rescaling factor. The performance of PGD depends on the scale of the logits.

If $p_y \approx 1$ and consequently $p_i \approx 0$ for $i \neq y$, then $\nabla_x \text{CE}(x, y) \approx \mathbf{0}$ and finite arithmetic yields $\nabla_x \text{CE}(x, y) = \mathbf{0}$ (this phenomenon of gradient vanishing is observed in Carlini and Wagner [2017a]). As noticed in Sec. 4.1.3, one can achieve $p_y \approx 1$ with a classifier $h = \alpha g$ equivalent to g (i.e. they take the same decision for every x) but rescaled by a constant $\alpha > 0$. To exemplify how this can lead to overestimation of robustness, we run 100 iterations of the l_∞ PGD-attack on the CE loss on the CIFAR-10 model from Atzmon et al. [2019], with $\epsilon = 0.031$, dividing the logits by a factor $\alpha \in \{1, 10^1, 10^2, 10^3\}$. In Fig. 4.11 we show the fraction of entries in the gradients of g/α (g is the original model) equal to zero and the robust accuracy achieved by the attack in dependency on α (we use 1000 test points, the gradient statistic is computed for correctly classified points). Without rescaling ($\alpha = 1$) the gradient vanishes almost for every coordinate, so that PGD is ineffective, but simply rescaling the logits is sufficient to get a much more accurate robustness assessment.

The CW loss [Carlini and Wagner, 2017a] defined as

$$\text{CW}(x, y) = -z_y + \max_{i \neq y} z_i. \tag{4.23}$$

has in contrast to the CE loss a direct interpretation in terms of the decision of the classifier. If an adversarial example exists, then the global maximum of the CW loss is positive. However, the CW loss is not scaling invariant and thus again an extreme rescaling could in principle be used to induce gradient masking.

We propose the **Difference of Logits Ratio (DLR)** loss which is both shift and rescaling invariant and thus has the same degrees of freedom as the decision of the classifier:

$$\text{DLR}(x, y) = -\frac{z_y - \max_{i \neq y} z_i}{z_{\pi_1} - z_{\pi_3}}, \tag{4.24}$$

where π is the ordering of the components of z in decreasing order. The required shift-invariance of the loss is achieved by having a difference of logits also in the denominator. Maximizing DLR wrt x allows to find a point classified not in class y (DLR is positive only if $\text{argmax}_i z_i \neq y$) and, once that is achieved, minimizes the score of class y compared to that of the other classes. If x is correctly classified we have $\pi_1 \equiv y$, so that $\text{DLR}(x, y) = -\frac{z_y - z_{\pi_2}}{z_y - z_{\pi_3}}$ and $\text{DLR}(x, y) \in [-1, 0]$. The role of the normalization $z_{\pi_1} - z_{\pi_3}$ is to push z_{π_2} to $z_y = z_{\pi_1}$ as it prefers points for which $z_y \approx z_{\pi_2} > z_{\pi_3}$ and thus is biased towards changing the decision. Furthermore, we adapt our DLR loss to induce misclassification into a target class t by

$$\text{Targeted-DLR}(x, y) = -\frac{z_y - z_t}{z_{\pi_1} - (z_{\pi_3} + z_{\pi_4})/2}. \tag{4.25}$$

Thus, we preserve both the shift and scaling invariance of DLR loss, while aiming at getting $z_t > z_y$, and modify the denominator in (4.24) to ensure that the loss is not constant.

4.3.4 APGD versus PGD on different losses

We compare PGD, PGD with Momentum (same momentum as in APGD) and APGD with the same budget, i.e. 100 iterations and 5 restarts, on a large number of classifiers trained l_∞-robustness (which we will be also used in the evaluation in Sec. 5.1), optimizing the CE, CW and DLR loss. The complete results are reported in Tables 4.16, 4.17, 4.18. For both PGD and PGD with Momentum we use three step sizes ($\epsilon/10$, $\epsilon/4$, 2ϵ, with ϵ the bound on the norm of the perturbations). APGD outperforms the best among the 6 versions of PGD on 32 of the 43 models with CE, 37/43 with CW and 35/43 with DLR, and the models where APGD is worse are mainly the ones where the extreme step size 2ϵ is optimal as the defenses lead to gradient masking/obfuscation. The version of standard PGD achieving most often the lowest robust accuracy is for all three losses PGD with Momentum and step size $\epsilon/4$, with average robust accuracy (over all models) of 54.84%, 50.42% and 50.47% on the CE, CW and DLR loss respectively. In the same metric, APGD achieves 54.00%, 49.46%, 48.53%. Comparing CW and DLR loss per model (over all PGD versions and APGD) the CW loss is up to 21% worse than the DLR loss, while it is never better by more than 5% and thus the DLR-loss is more stable. Moreover, the only cases where PGD is more than 1% better than APGD consists of cases where the very large step size 2ϵ outperforms all other step sizes. Note that the step size 2ϵ works significantly worse than the smaller step sizes for most of the other models. It is likely that these defenses modify the loss landscape to make it unsuitable for standard gradient-based optimization, and an *informed random search* (as PGD with such a large step size can be seen) works better. The same happens with the CE loss, where PGD yields the best results (with a non-negligible gap to APGD) also on the models of Kim and Wang [2020] and Taghanaki et al. [2019]. However, notice that for these classifiers optimizing the DLR loss is significantly more effective than using the CE loss.

In total these experiments show: i) APGD outperforms PGD/PGD with Momentum consistently regardless of the employed loss, ii) our DLR loss improves upon the CE loss and is comparable to the CW loss, but with less severe failure cases.

Finally, we run a similar comparison for models trained to be robust wrt l_2 and report the results in Tables 4.19, 4.20 and 4.21. APGD again yields most often the best results for all the losses. The difference of the losses is for l_2 marginal.

Table 4.16: **Comparison PGD vs APGD on the cross-entropy loss in the l_∞-threat model.** We run PGD and PGD with Momentum (the same used for APGD) with three different step sizes: $\epsilon/10$, $\epsilon/4$ and 2ϵ. Similarly to APGD, we use 100 iterations and 5 restarts for PGD, and report the resulting robust accuracy on the whole test set (1000 images for ImageNet). For each model we boldface the best attack and underline the best version of PGD (i.e. we exclude APGD).

			PGD			PGD with Momentum			
#	paper	clean	$\epsilon/10$	$\epsilon/4$	2ϵ	$\epsilon/10$	$\epsilon/4$	2ϵ	APGD
CIFAR-10 - $\epsilon = 8/255$									
1	Carmon et al. [2019]	89.69	62.01	61.86	63.15	61.98	61.79	61.84	**61.47**
2	Alayrac et al. [2019]	86.46	60.78	60.72	61.87	60.70	60.58	60.73	**59.86**
3	Hendrycks et al. [2019]	87.11	57.24	57.13	57.88	57.28	57.14	57.13	**57.00**
4	Rice et al. [2020]	85.34	56.87	56.82	57.55	56.88	56.78	56.81	**56.76**
5	Qin et al. [2019]	86.28	55.50	55.48	56.42	55.50	55.47	55.62	**55.45**
6	Engstrom et al. [2019a]	87.03	52.06	51.90	53.30	52.08	51.87	51.85	**51.52**
7	Kumari et al. [2019]	87.80	51.85	51.75	52.95	51.87	51.65	51.68	**51.56**
8	Mao et al. [2019]	86.21	49.71	49.59	50.69	49.72	49.56	49.61	**49.39**
9	Zhang et al. [2019a]	87.20	46.15	46.08	47.18	46.13	46.08	46.04	**45.91**
10	Madry et al. [2018]	87.14	44.93	44.83	46.46	44.94	44.81	44.99	**44.56**
11	Pang et al. [2020a]	80.89	56.02	56.04	56.22	56.02	56.01	56.04	**55.91**
12	Wong et al. [2020]	83.34	46.10	45.93	47.45	46.06	45.90	46.03	**45.60**
13	Shafahi et al. [2019]	86.11	44.45	45.60	48.83	43.63	44.50	47.07	**43.30**
14	Ding et al. [2020]	84.36	50.57	50.36	50.16	50.58	50.41	49.66	**49.36**
15	Moosavi-Dezfooli et al. [2019]	83.11	41.76	41.71	42.33	41.70	41.68	**41.58**	41.59
16	Zhang and Wang [2019]	89.98	67.38	66.79	**57.31**	67.40	66.50	61.15	62.03
17	Zhang and Xu [2020]	90.25	71.69	71.42	**62.52**	71.77	71.48	68.88	69.36
18	Jang et al. [2019]	78.91	38.00	37.88	39.72	37.90	37.73	37.90	**37.40**
19	Kim and Wang [2020]	91.51	59.16	58.24	**51.29**	59.12	58.23	54.17	54.80
20	Moosavi-Dezfooli et al. [2019]	80.41	36.86	36.74	37.92	36.81	36.74	36.61	**36.53**
21	Wang and Zhang [2019]	92.80	61.92	60.61	**54.96**	62.05	60.66	56.89	57.19
22	Wang and Zhang [2019]	92.82	69.63	69.10	**61.03**	69.60	69.06	67.71	67.77
23	Mustafa et al. [2019]	89.16	17.04	13.79	**4.39**	17.06	13.33	5.57	4.48
24	Chan et al. [2020]	93.79	**1.89**	**1.89**	9.67	1.91	1.90	5.93	1.90
25	Pang et al. [2020a]	93.52	85.72	85.73	85.67	85.72	85.72	86.23	**85.58**
CIFAR-10 - $\epsilon = 0.031$									
1	Zhang et al. [2019b]	84.92	55.28	55.24	56.24	55.25	55.21	55.30	**55.08**
2	Atzmon et al. [2019]	81.30	**78.94**	**78.94**	**78.94**	**78.94**	**78.94**	**78.94**	**78.94**
3	Xiao et al. [2020]	79.28	36.32	36.03	34.19	36.27	36.04	35.97	**32.38**
CIFAR-10 - $\epsilon = 4/255$									
1	Song et al. [2019a]	84.81	57.42	**57.40**	57.60	57.42	57.41	57.43	57.43
CIFAR-10 - $\epsilon = 0.02$									
1	Pang et al. [2019]	91.22	7.27	5.96	17.23	7.00	5.44	9.52	**3.53**
2	Pang et al. [2019]	93.44	0.36	0.24	3.37	0.41	0.17	1.22	**0.04**
CIFAR-100 - $\epsilon = 8/255$									
1	Hendrycks et al. [2019]	59.23	33.01	32.91	33.17	33.02	32.83	**32.76**	32.83
2	Rice et al. [2020]	53.83	20.61	20.57	21.08	20.57	20.52	20.47	**20.32**
MNIST - $\epsilon = 0.3$									
1	Zhang et al. [2020b]	98.38	95.25	95.23	95.21	95.06	94.98	94.99	**94.58**
2	Gowal et al. [2019a]	98.34	94.67	94.58	94.39	94.38	94.23	94.19	**93.81**
3	Zhang et al. [2019b]	99.48	94.07	94.10	95.27	93.97	93.80	94.32	**93.14**
4	Ding et al. [2020]	98.95	94.44	94.42	95.96	94.04	93.53	95.43	**93.51**
5	Atzmon et al. [2019]	99.35	**98.79**	98.83	98.94	**98.79**	98.83	98.93	**98.79**
6	Madry et al. [2018]	98.53	90.75	90.88	91.97	90.37	90.29	91.03	**89.40**
7	Jang et al. [2019]	98.47	93.05	93.66	95.34	92.73	92.47	94.47	**92.45**
8	Wong et al. [2020]	98.50	87.16	87.40	90.19	86.51	86.30	87.23	**84.74**
9	Taghanaki et al. [2019]	98.86	20.61	20.27	**20.23**	20.56	20.27	20.43	23.83
ImageNet - $\epsilon = 4/255$									
1	Engstrom et al. [2019a]	63.4	31.5	31.4	34.0	31.5	31.4	31.9	**30.9**

Table 4.17: **Comparison PGD vs APGD on the CW loss in the l_∞-threat model.** We run PGD and PGD with Momentum (the same used for APGD) with three different step sizes: $\epsilon/10$, $\epsilon/4$ and 2ϵ. Similarly to APGD, we use 100 iterations and 5 restarts for PGD, and report the resulting robust accuracy on the whole test set (1000 images for ImageNet). For each model we boldface the best attack and underline the best version of PGD (i.e. we exclude APGD).

			PGD			PGD with Momentum			
#	paper	clean	$\epsilon/10$	$\epsilon/4$	2ϵ	$\epsilon/10$	$\epsilon/4$	2ϵ	APGD
CIFAR-10 - $\epsilon = 8/255$									
1	Carmon et al. [2019]	89.69	60.73	60.72	62.20	60.73	60.69	61.02	**60.48**
2	Alayrac et al. [2019]	86.46	61.78	61.85	63.23	61.69	61.61	62.10	**61.13**
3	Hendrycks et al. [2019]	87.11	56.44	56.42	57.15	56.45	56.41	56.39	**56.20**
4	Rice et al. [2020]	85.34	55.24	55.21	56.04	55.23	55.19	55.26	**55.07**
5	Qin et al. [2019]	86.28	55.10	55.10	55.95	55.10	55.11	55.22	**54.99**
6	Engstrom et al. [2019a]	87.03	52.24	52.17	53.61	52.26	52.16	52.36	**51.94**
7	Kumari et al. [2019]	87.80	51.13	51.07	52.67	51.14	51.03	51.22	**50.88**
8	Mao et al. [2019]	86.21	49.88	49.84	51.18	49.88	49.84	50.02	**49.71**
9	Zhang et al. [2019a]	87.20	47.13	47.09	48.27	47.11	47.06	47.08	**46.89**
10	Madry et al. [2018]	87.14	45.98	45.82	47.47	45.95	45.77	46.11	**45.67**
11	Pang et al. [2020a]	80.89	44.40	44.41	45.83	44.41	44.36	44.66	**44.11**
12	Wong et al. [2020]	83.34	46.04	45.94	47.72	46.03	45.93	46.26	**45.66**
13	Shafahi et al. [2019]	86.11	44.89	45.93	49.27	44.16	45.10	47.43	**43.90**
14	Ding et al. [2020]	84.36	51.60	51.44	51.09	51.59	51.29	50.63	**50.25**
15	Moosavi-Dezfooli et al. [2019]	83.11	40.21	40.18	40.93	40.20	40.16	40.22	**40.10**
16	Zhang and Wang [2019]	89.98	55.98	55.30	**48.59**	55.99	55.27	50.91	51.65
17	Zhang and Xu [2020]	90.25	67.00	66.61	**57.87**	67.02	66.67	64.19	64.46
18	Jang et al. [2019]	78.91	36.85	36.81	39.32	36.85	36.83	37.30	**36.52**
19	Kim and Wang [2020]	91.51	56.85	55.90	**50.26**	56.94	55.92	52.54	53.04
20	Moosavi-Dezfooli et al. [2019]	80.41	35.38	35.37	36.86	35.38	35.35	35.48	**35.29**
21	Wang and Zhang [2019]	92.80	59.93	58.42	**52.56**	60.01	58.46	54.51	54.77
22	Wang and Zhang [2019]	92.82	66.49	65.86	**56.71**	66.60	65.76	63.53	63.80
23	Mustafa et al. [2019]	89.16	18.27	14.75	**4.42**	18.44	14.66	5.46	4.73
24	Chan et al. [2020]	93.79	1.32	1.28	5.51	1.30	1.29	1.29	**1.23**
25	Pang et al. [2020a]	93.52	1.88	0.54	1.42	1.59	0.53	0.47	**0.11**
CIFAR-10 - $\epsilon = 0.031$									
1	Zhang et al. [2019b]	84.92	54.05	54.06	55.16	54.08	54.03	54.22	**53.90**
2	Atzmon et al. [2019]	81.30	40.37	40.28	42.30	40.36	40.21	40.56	**40.05**
3	Xiao et al. [2020]	79.28	35.77	35.53	34.04	36.14	35.31	35.76	**32.09**
CIFAR-10 - $\epsilon = 4/255$									
1	Song et al. [2019a]	84.81	57.54	57.53	57.78	57.52	57.53	57.51	**57.49**
CIFAR-10 - $\epsilon = 0.02$									
1	Pang et al. [2019]	91.22	7.12	6.09	16.15	6.87	5.57	9.49	**3.82**
2	Pang et al. [2019]	93.44	0.47	0.26	3.13	0.42	0.13	1.22	**0.12**
CIFAR-100 - $\epsilon = 8/255$									
1	Hendrycks et al. [2019]	59.23	30.64	30.61	31.19	30.63	30.61	30.56	**30.45**
2	Rice et al. [2020]	53.83	20.22	20.21	21.06	20.22	20.22	20.35	**19.98**
MNIST - $\epsilon = 0.3$									
1	Zhang et al. [2020b]	98.38	95.22	95.21	95.17	95.10	94.97	95.04	**94.61**
2	Gowal et al. [2019a]	98.34	94.63	94.48	94.39	94.49	94.12	94.23	**93.58**
3	Zhang et al. [2019b]	99.48	94.40	94.35	95.23	94.18	94.04	94.33	**93.35**
4	Ding et al. [2020]	98.95	94.49	94.52	95.85	93.98	93.81	95.38	**93.53**
5	Atzmon et al. [2019]	99.35	94.43	94.69	95.98	94.03	93.74	95.21	**92.98**
6	Madry et al. [2018]	98.53	90.82	90.96	91.94	90.39	90.30	91.07	**89.39**
7	Jang et al. [2019]	98.47	93.28	93.64	95.26	92.68	92.71	94.49	**92.40**
8	Wong et al. [2020]	98.50	87.32	87.55	90.20	86.74	86.62	87.25	**84.97**
9	Taghanaki et al. [2019]	98.86	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
ImageNet - $\epsilon = 4/255$									
1	Engstrom et al. [2019a]	63.4	31.7	31.6	34.5	31.7	31.7	32.5	**31.5**

Table 4.18: **Comparison PGD vs APGD on the DLR loss in the l_∞-threat model.** We run PGD and PGD with Momentum (the same used for APGD) with three different step sizes: $\epsilon/10$, $\epsilon/4$ and 2ϵ. Similarly to APGD, we use 100 iterations and 5 restarts for PGD, and report the resulting robust accuracy on the whole test set (1000 images for ImageNet). For each model we boldface the best attack and underline the best version of PGD (i.e. we exclude APGD).

			PGD			PGD with Momentum			
#	paper	clean	$\epsilon/10$	$\epsilon/4$	2ϵ	$\epsilon/10$	$\epsilon/4$	2ϵ	APGD
CIFAR-10 - $\epsilon = 8/255$									
1	Carmon et al. [2019]	89.69	60.98	60.88	62.41	60.93	60.82	61.25	**60.64**
2	Alayrac et al. [2019]	86.46	63.27	63.15	64.71	63.16	63.00	63.55	**62.03**
3	Hendrycks et al. [2019]	87.11	57.27	57.18	57.98	57.27	57.18	57.22	**56.96**
4	Rice et al. [2020]	85.34	55.89	55.86	56.75	55.90	55.85	55.93	**55.72**
5	Qin et al. [2019]	86.28	55.73	55.70	56.47	55.73	55.68	55.70	**55.46**
6	Engstrom et al. [2019a]	87.03	53.13	53.04	54.41	53.09	52.95	53.29	**52.62**
7	Kumari et al. [2019]	87.80	51.97	51.92	53.58	51.98	51.89	52.11	**51.68**
8	Mao et al. [2019]	86.21	50.79	50.57	52.03	50.74	50.52	50.79	**50.33**
9	Zhang et al. [2019a]	87.20	47.70	47.58	48.84	47.71	47.57	47.53	**47.33**
10	Madry et al. [2018]	87.14	46.49	46.30	48.13	46.51	46.30	46.58	**46.03**
11	Pang et al. [2020a]	80.89	44.97	44.91	46.36	44.96	44.87	45.21	**44.56**
12	Wong et al. [2020]	83.34	47.20	47.05	48.62	47.18	47.01	47.22	**46.64**
13	Shafahi et al. [2019]	86.11	45.46	46.43	50.03	44.74	45.51	47.89	**44.56**
14	Ding et al. [2020]	84.36	51.67	51.44	51.46	51.68	51.39	50.93	**50.32**
15	Moosavi-Dezfooli et al. [2019]	83.11	40.36	40.33	41.29	40.37	40.32	40.43	**40.29**
16	Zhang and Wang [2019]	89.98	55.69	54.73	**46.57**	55.68	54.68	48.53	48.96
17	Zhang and Xu [2020]	90.25	59.16	56.77	**46.72**	59.13	56.61	49.07	49.43
18	Jang et al. [2019]	78.91	37.41	37.30	39.77	37.38	37.27	37.69	**37.01**
19	Kim and Wang [2020]	91.51	52.42	51.41	**48.09**	52.43	51.30	48.51	48.41
20	Moosavi-Dezfooli et al. [2019]	80.41	35.57	35.53	37.25	35.57	35.52	35.68	**35.47**
21	Wang and Zhang [2019]	92.80	61.40	53.79	**38.23**	61.31	53.27	39.16	40.69
22	Wang and Zhang [2019]	92.82	54.84	49.01	**35.15**	54.83	48.62	36.05	36.72
23	Mustafa et al. [2019]	89.16	22.16	17.10	4.90	22.18	17.42	5.57	**4.54**
24	Chan et al. [2020]	93.79	9.52	7.11	10.41	7.56	5.38	5.30	**1.20**
25	Pang et al. [2020a]	93.52	1.94	**0.47**	1.75	2.06	0.58	0.75	0.49
CIFAR-10 - $\epsilon = 0.031$									
1	Zhang et al. [2019b]	84.92	54.25	54.20	55.28	54.23	54.16	54.34	**54.04**
2	Atzmon et al. [2019]	81.30	57.45	51.99	44.62	56.19	50.03	**44.45**	44.50
3	Xiao et al. [2020]	79.28	37.74	36.33	33.46	37.49	36.92	35.80	**31.27**
CIFAR-10 - $\epsilon = 4/255$									
1	Song et al. [2019a]	84.81	57.85	57.83	58.07	57.84	57.83	57.81	**57.79**
CIFAR-10 - $\epsilon = 0.02$									
1	Pang et al. [2019]	91.22	15.83	13.33	18.98	15.23	12.38	16.31	**8.61**
2	Pang et al. [2019]	93.44	1.53	0.98	3.58	1.25	0.68	2.59	**0.26**
CIFAR-100 - $\epsilon = 8/255$									
1	Hendrycks et al. [2019]	59.23	32.05	31.91	32.14	32.03	31.92	31.70	**31.68**
2	Rice et al. [2020]	53.83	20.41	20.34	21.41	20.40	20.33	20.47	**20.20**
MNIST - $\epsilon = 0.3$									
1	Zhang et al. [2020b]	98.38	95.65	95.52	95.26	95.41	95.17	95.14	**94.82**
2	Gowal et al. [2019a]	98.34	94.93	94.73	94.28	94.72	94.45	94.22	**93.87**
3	Zhang et al. [2019b]	99.48	94.84	94.78	95.53	94.63	94.50	94.79	**93.89**
4	Ding et al. [2020]	98.95	94.69	94.78	95.80	94.16	94.00	95.40	**93.86**
5	Atzmon et al. [2019]	99.35	94.70	94.99	96.32	94.10	**93.95**	95.53	94.54
6	Madry et al. [2018]	98.53	91.08	91.21	92.15	90.65	90.51	91.51	**89.74**
7	Jang et al. [2019]	98.47	93.19	93.79	94.74	92.72	92.99	94.49	**92.15**
8	Wong et al. [2020]	98.50	87.58	87.65	90.40	86.99	86.87	87.74	**85.39**
9	Taghanaki et al. [2019]	98.86	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
ImageNet - $\epsilon = 4/255$									
1	Engstrom et al. [2019a]	63.4	32.1	32.1	35.0	32.1	32.1	32.8	**32.0**

Table 4.19: **Comparison PGD vs APGD on the cross-entropy loss in the l_2-threat model.** We run PGD and PGD with Momentum (the same used for APGD) with three different step sizes: $\epsilon/10$, $\epsilon/4$ and 2ϵ. Similarly to APGD, we use 100 iterations and 5 restarts for PGD, and report the resulting robust accuracy on the whole test set (1000 images for ImageNet). For each model we boldface the best attack and underline the best version of PGD (i.e. we exclude APGD).

			PGD			PGD with Momentum			
#	paper	clean	$\epsilon/10$	$\epsilon/4$	2ϵ	$\epsilon/10$	$\epsilon/4$	2ϵ	APGD
CIFAR-10 - $\epsilon = 0.5$									
1	Augustin et al. [2020]	91.08	74.81	74.74	**74.60**	74.81	74.75	74.63	74.65
2	Engstrom et al. [2019a]	90.83	69.68	69.65	**69.58**	69.69	69.63	69.62	69.60
3	Rice et al. [2020]	88.67	68.64	68.60	68.55	68.64	68.58	68.56	**68.53**
4	Rony et al. [2019]	89.05	66.60	66.58	66.63	66.59	66.58	66.58	**66.57**
5	Ding et al. [2020]	88.02	66.22	66.21	66.29	66.22	66.21	66.21	**66.19**
ImageNet - $\epsilon = 3$									
1	Engstrom et al. [2019a]	55.3	31.9	31.6	31.6	31.9	**31.5**	31.6	**31.5**

Table 4.20: **Comparison PGD vs APGD on the CW loss in the l_2-threat model.** We run PGD and PGD with Momentum (the same used for APGD) with three different step sizes: $\epsilon/10$, $\epsilon/4$ and 2ϵ. Similarly to APGD, we use 100 iterations and 5 restarts for PGD, and report the resulting robust accuracy on the whole test set (1000 images for ImageNet). For each model we boldface the best attack and underline the best version of PGD (i.e. we exclude APGD).

			PGD			PGD with Momentum			
#	paper	clean	$\epsilon/10$	$\epsilon/4$	2ϵ	$\epsilon/10$	$\epsilon/4$	2ϵ	APGD
CIFAR-10 - $\epsilon = 0.5$									
1	Augustin et al. [2020]	91.08	74.57	74.54	74.55	74.56	74.55	74.53	**74.49**
2	Engstrom et al. [2019a]	90.83	70.08	70.07	**70.06**	70.08	70.07	**70.06**	**70.06**
3	Rice et al. [2020]	88.67	68.73	**68.71**	68.72	68.73	68.72	**68.71**	**68.71**
4	Rony et al. [2019]	89.05	67.00	67.00	67.03	67.00	67.00	**66.99**	**66.99**
5	Ding et al. [2020]	88.02	66.55	66.54	66.64	66.55	66.54	**66.53**	66.54
ImageNet - $\epsilon = 3$									
1	Engstrom et al. [2019a]	55.3	**30.7**	**30.7**	30.8	30.8	**30.7**	**30.7**	**30.7**

Table 4.21: **Comparison PGD vs APGD on the DLR loss in the l_2-threat model.** We run PGD and PGD with Momentum (the same used for APGD) with three different step sizes: $\epsilon/10$, $\epsilon/4$ and 2ϵ. Similarly to APGD, we use 100 iterations and 5 restarts for PGD, and report the resulting robust accuracy on the whole test set (1000 images for ImageNet). For each model we boldface the best attack and underline the best version of PGD (i.e. we exclude APGD).

			PGD			PGD with Momentum			
#	paper	clean	$\epsilon/10$	$\epsilon/4$	2ϵ	$\epsilon/10$	$\epsilon/4$	2ϵ	APGD
CIFAR-10 - $\epsilon = 0.5$									
1	Augustin et al. [2020]	91.08	75.06	75.00	74.99	75.06	75.00	74.99	**74.94**
2	Engstrom et al. [2019a]	90.83	70.20	70.19	**70.17**	70.20	70.19	**70.17**	70.20
3	Rice et al. [2020]	88.67	69.02	69.01	68.96	69.02	69.02	**68.94**	68.95
4	Rony et al. [2019]	89.05	67.03	**67.02**	67.06	67.03	**67.02**	**67.02**	**67.02**
5	Ding et al. [2020]	88.02	66.61	66.56	66.64	66.61	66.55	66.54	**66.53**
ImageNet - $\epsilon = 3$									
1	Engstrom et al. [2019a]	55.3	**30.9**	**30.9**	**30.9**	**30.9**	31.1	**30.9**	**30.9**

4.4 Auto-PGD and Square Attack for the l_1-bounded threat model

While l_∞ and l_2 are the most popular threat models for adversarial attacks, l_1-bounded attacks represent a complementary type of perturbations. In the following we identify reasons why the current versions of l_1-PGD attacks are weaker than SOTA l_1-attacks [Chen et al., 2018, Rony et al., 2021], including FAB (Sec. 4.1). A key issue is that in image classification we have the additional constraint that the input has to lie in the box $[0,1]^d$ and thus the effective threat model is the intersection of the l_1-ball and $[0,1]^d$. However, current l_1-PGD attacks only approximate the correct projection onto this set [Tramèr and Boneh, 2019] and argue for a steepest descent direction without taking into account the box constraints.

We first show that the correct projection onto the intersection can be computed in essentially the same time as the projection onto the l_1-ball, and then we discuss theoretically and empirically that using the approximate projection leads to a worse attack as it cannot access certain parts of the threat model. Moreover, we derive the correct steepest descent step for the intersection of l_1-ball and $[0,1]^d$ which motivates an adaptive sparsity of the chosen descent direction. Then, inspired by Auto-PGD (APGD) for l_2 and l_∞ introduced in Sec. 4.3, we design a novel fully adaptive parameter-free PGD scheme so that the user does not need to do step size selection for each defense separately which is known to be error prone. Interestingly, using our l_1-APGD we are able to train the model with the highest l_1-robust accuracy for $\epsilon = 12$ while standard PGD fails due to catastrophic overfitting [Wong et al., 2020] and/or overfitting to the sparsity of the standard PGD attack. Finally, we use the same insights about the geometry of the l_1-threat model to develop an l_1-adaptation of the black-box Square Attack presented in Sec. 4.2.

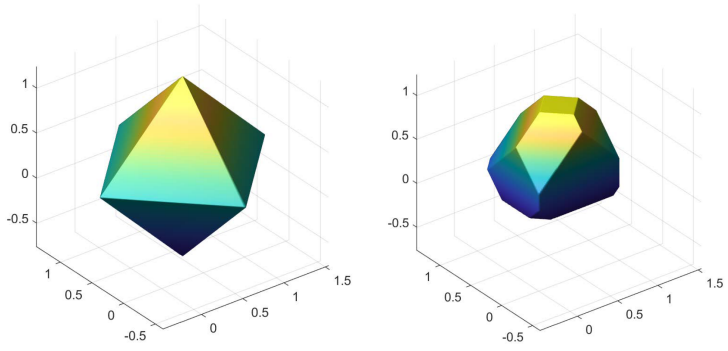

Figure 4.12: **Left:** the l_1-ball $B_1(x, 1)$ centered at the (randomly chosen) target point $x \in [0,1]^3$. **Right:** the intersection $S = B_1(x, 1) \cap [0,1]^3$ of $B_1(x, 1)$ with the box $[0,1]^3$.

4.4.1 Mind the box $[0,1]^d$ in l_1-PGD

We recall that Projected Gradient Descent is a simple first-order method which consists in a descent step followed by a projection onto the feasible set S, that is, given the current iterate $x^{(i)}$, the next iterate $x^{(i+1)}$ is computed as

$$u^{(i+1)} = x^{(i)} + \eta^{(i)} \cdot s(\nabla L(x^{(i)})), \tag{4.26}$$

$$x^{(i+1)} = P_S(u^{(i+1)}), \tag{4.27}$$

where d is the input dimension, $\eta^{(i)} > 0$ the step size at iteration i, $s : \mathbb{R}^d \to \mathbb{R}^d$ determines the descent direction as a function of the gradient of the loss L at $x^{(i)}$ and $P_S : \mathbb{R}^d \to S$ is the projection on S. With an l_1-perturbation model of radius ϵ, we denote by $B_1(x, \epsilon) := \{z \in \mathbb{R}^d \mid \|z - x\|_1 \leq \epsilon\}$ the l_1-ball around a target point $x \in [0,1]^d$ and define $S = [0,1]^d \cap B_1(x, \epsilon)$. The main difference to prior work is that we take explicitly into account the image constraint $[0,1]^d$. Note that the geometry of the effective threat model is actually quite different from $B_1(x, \epsilon)$ alone, see Fig. 4.12 for an illustration. In the following we analyse the projection and

the descent step in this effective threat model S. As ϵ is significantly higher for l_1 (we use $\epsilon = 12$ similar to Maini et al. [2020]) as for l_2 (standard 0.5) and l_∞ (standard $\frac{8}{255}$) the difference of the intersection with $[0,1]^d$ to the l_p-ball alone is most prominent for the l_1-case.

4.4.2 Projection onto S

With $B_1(x,\epsilon)$ as defined above and denoting $H = [0,1]^d$ the image box, we consider the two projection problems:

$$P_S(u) = \operatorname*{arg\,min}_{z\in\mathbb{R}^d} \|u-z\|_2^2 \quad \text{s.th.} \quad \|z-x\|_1 \leq \epsilon, \quad z \in [0,1]^d. \tag{4.28}$$

and

$$P_{B_1(x,\epsilon)}(u) = \operatorname*{arg\,min}_{z\in\mathbb{R}^d} \|u-z\|_2^2 \quad \text{s.th.} \quad \|z-x\|_1 \leq \epsilon. \tag{4.29}$$

It is well known that the l_1-projection problem in (4.29) can be solved in $O(d \log d)$ [Duchi et al., 2008, Condat, 2016]. We show now that also the exact projection onto S can be computed with the same complexity (after this paper has been accepted we got aware of Wang et al. [2019b] who derived also the form of the solution of (4.28) but provided no complexity analysis or an algorithm to compute it).

Proposition 4.4.1 *The projection problem (4.28) onto $S = B_1(x,\epsilon) \cap H$ can be solved in $O(d \log d)$ with solution*

$$z_i^* = \begin{cases} 1 & \text{for } u_i \geq x_i \text{ and } 0 \leq \lambda_e^* \leq u_i - 1 \\ u_i - \lambda_e^* & \text{for } u_i \geq x_i \text{ and } u_i - 1 < \lambda_e^* \leq u_i - x_i \\ x_i & \text{for } \lambda_e^* > |u_i - x_i| \\ u_i + \lambda_e^* & \text{for } u_i \leq x_i \text{ and } -u_i < \lambda_e^* \leq x_i - u_i \\ 0 & \text{for } u_i \leq x_i \text{ and } 0 \leq \lambda_e^* \leq -u_i \end{cases},$$

where $\lambda_e^* \geq 0$. With $\gamma \in \mathbb{R}^d$ defined as

$$\gamma_i = \max\{-x_i \operatorname{sign}(u_i - x_i), (1-x_i)\operatorname{sign}(u_i - x_i)\},$$

it holds $\lambda_e^* = 0$ if $\sum_{i=1}^d \max\{0, \min\{|u_i - x_i|, \gamma_i\} \leq \epsilon$ and otherwise λ_e^* is the solution of

$$\sum_{i=1}^d \max\left\{0, \min\{|u_i - x_i| - \lambda_e^*, \gamma_i\}\right\} = \epsilon.$$

Proof. By introducing the variable $w' = z - x$ we transform (4.28) into

$$\min_{w'\in\mathbb{R}^d} \frac{1}{2}\|u - x - w'\|_2^2 \quad \text{s.th.} \quad \|w'\|_1 \leq \epsilon, \quad w' + x \in [0,1]^d.$$

Moreover, by introducing $w = \operatorname{sign}(u-x)w'$ we get

$$\min_{w\in\mathbb{R}^d} \frac{1}{2}\sum_{i=1}^d (|u_i - x_i| - w_i)^2 \quad \text{s.th.} \quad \sum_{i=1}^d w_i \leq \epsilon, \quad w_i \geq 0, \quad \operatorname{sign}(u_i - x_i)w_i + x_i \in [0,1].$$

The two componentwise constraints can be summarized with

$$\gamma_i := \max\{-x_i \operatorname{sign}(u_i - x_i), (1-x_i)\operatorname{sign}(u_i - x_i)\}$$

as $w_i \in [0, \gamma_i]$. Note that if $\gamma_i = 0$ this fixes the variable $w_i = 0$ and we then remove this variable from the optimization problem. Thus wlog we assume in the following that $\gamma_i > 0$. The corresponding Lagrangian becomes

$$L(w,\alpha,\beta,\lambda_e) = \frac{1}{2}\sum_{i=1}^d (|u_i - x_i| - w_i)^2 + \lambda_e(\langle \mathbf{1}, w\rangle - \epsilon) + \langle \alpha, w - \gamma\rangle - \langle \beta, w\rangle,$$

which yields the KKT optimality conditions for the optimal primal variable w^* and dual variables $\alpha^*, \beta^*, \lambda_e^*$

$$\begin{aligned}\nabla_w L_i = w_i^* - |u_i - x_i| + \lambda_e^* + \alpha_i^* - \beta_i^* = 0 \\ \lambda_e^*(\sum_{i=1}^d w_i^* - \epsilon) = 0 \\ \alpha_i^*(w_i - \gamma_i) = 0, \qquad \beta_i^* w_i = 0 \\ \lambda_e^* \geq 0,\ \alpha_i^* \geq 0,\ \beta_i^* \geq 0\end{aligned}$$

Thus $\beta_i^* > 0$ implies $w_i^* = 0$ and with $\gamma_i > 0$ this yields $\alpha_i^* = 0$ and thus $\beta^* = \lambda_e - |u_i - x_i|$ and we get

$$\beta_i^* = \max\{0, \lambda_e^* - |u_i - x_i|\}.$$

On the other hand $\alpha_i^* > 0$ implies $w_i^* = \gamma_i$ and $\beta_i^* = 0$ and thus $\alpha_i^* = |u_i - x_i| - \gamma_i - \lambda_e^*$ and thus

$$\alpha_i^* = \max\{0, |u_i - x_i| - \gamma_i - \lambda_e^*\}.$$

Thus we have in total

$$w_i^* = \begin{cases} \gamma_i & \text{if } |u_i - x_i| - \gamma_i - \lambda_e^* > 0 \\ 0 & \text{if } \lambda_e^* - |u_i - x_i| > 0 \\ |u_i - x_i| - \lambda_e^* & \text{else.} \end{cases}$$

which can be summarized as $w_i^* = \max\{0, \min\{|u_i - x_i| - \lambda_e^*, \gamma_i\}\}$. Finally, if $\sum_{i=1}^d \max\{0, \min\{|u_i - x_i|, \gamma_i\}\} < \epsilon$ then $\lambda_e^* = 0$ is optimal, otherwise $\lambda_e^* > 0$ and we get the solution from the KKT condition

$$\sum_{i=1}^d \max\{0, \min\{|u_i - x_i| - \lambda_e^*, \gamma_i\}\} = \epsilon. \tag{4.30}$$

Noting that

$$\phi(\lambda_e) = \max\{0, \min\{|u_i - x_i| - \lambda_e, \gamma_i\}\}$$

is a piecewise linear and monotonically decreasing function in λ_e, the solution can be found by sorting the union of $|u_i - x_i| - \gamma_i$ and $|u_i - x_i|$ in non-decreasing order π and then starting with $\lambda_e = 0$ and then going through the sorted list until $\phi(\lambda_e) < \epsilon$. In this case one has identified the interval which contains the optimal solution λ_e^* and computes the solution with the "active" set of components

$$\{i \,|\, 0 < |u_i - x_i| - \lambda_e < \gamma_i\},$$

via Eq. (4.30). The algorithm is provided in Alg. 7, where the main complexity is the initial sorting step $O(2d \log(2d))$ and some steps in $O(d)$ so that the total complexity is $O(2d \log(2d))$. Once we transform back to the original variable of (4.28) we get with the form of γ the solution

$$z_i^* = x_i + \text{sign}(u_i - x_i) w_i^* = \begin{cases} 1 & \text{for } u_i \geq x_i \text{ and } 0 \leq \lambda_e^* \leq u_i - 1 \\ u_i - \lambda_e^* & \text{for } u_i \geq x_i \text{ and } u_i - 1 < \lambda_e^* \leq u_i - x_i \\ x_i & \text{for } \lambda_e^* > |u_i - x_i| \\ u_i + \lambda_e^* & \text{for } u_i \leq x_i \text{ and } -u_i < \lambda_e^* \leq x_i - u_i \\ 0 & \text{for } u_i \leq x_i \text{ and } 0 \leq \lambda_e^* \leq -u_i \end{cases}$$

□

The two prior versions of PGD [Tramèr and Boneh, 2019, Maini et al., 2020] for the l_1-threat model use the approximation $A : \mathbb{R}^d \to S$

$$A(u) = (P_H \circ P_{B_1(x,\epsilon)})(u),$$

instead of the exact projection $P_S(u)$ (see below for a proof that $A(u) \in S$ for any $u \in \mathbb{R}^d$). However, it turns out that the approximation $A(u)$ "hides" parts of S due to the following property.

Algorithm 7: Projection onto $B_1(x,\epsilon) \cap [0,1]^d$

Input: point to be projected u, x and radius ϵ
Output: projection z onto $B_1(x,\epsilon) \cap [0,1]^d$

$\gamma_i = \max\{-x_i \operatorname{sign}(u_i - x_i), (1-x_i)\operatorname{sign}(u_i - x_i)\}$, $\lambda^* = 0$
$z_i = \min\{|u_i - x_i|, \gamma_i\}$
$S = \sum_{i=1}^d z_i$
if $S > \epsilon$ **then**
  sort $t_i = \{|u_i - x_i|, |u_i - x_i| - \gamma_i\}$ in decreasing order π and memorize if it is $|u_i - x_i|$ (category 0) or $|u_i - x_i| - \gamma$ (category 1)
  $M = |\{i \,|\, z_i = |u_i - x_i| \text{ and } |u_i - x_i| > 0\}|$
  $\lambda = 0$
  for $j = 1, \ldots, 2d$ **do**
    $\lambda_{\text{old}} = \max\{0, \lambda\}$
    $\lambda = t_{\pi_j}$
    $S = S - M(\lambda - \lambda_{\text{old}})$
    if *category*$(\pi_j) = 0$ **then**
      $M = M + 1$
    else
      $M = M - 1$
    end
    if $S < \epsilon$ **then**
      if *category*$(\pi_j) = 0$ **then**
        $M = M - 1$
      else
        $M = M + 1$
      end
      $S = S + M(\lambda - \lambda_{\text{old}})$
      $\lambda^* = \lambda_{\text{old}} + (S - \epsilon)/M$
      BREAK
    end
  end
end
$z_i = \max\{0, \min\{|u_i - x_i| - \lambda^*, \gamma_i\}\}$, $i = 1, \ldots, d$
$P_S(u)_i = x_i + \operatorname{sign}(u_i - x_i) \cdot z_i$;

Lemma 4.4.1 *It holds for any $u \in \mathbb{R}^d$,*

$$\|P_S(u) - x\|_1 \geq \|A(u) - x\|_1 .$$

In particular, if $P_{B_1(x,\epsilon)}(u) \notin H$ and $\|u - x\|_1 > \epsilon$ and one of the following conditions holds

- $\|P_S(u) - x\|_1 = \epsilon$
- $\|P_S(u) - x\|_1 < \epsilon$ *and* $\exists u_i \in [0,1]$ *with* $u_i \neq x_i$

then

$$\|P_S(u) - x\|_1 > \|A(u) - x\|_1 .$$

Proof. It holds $A(u) \in H$ but also $A(u) \in B_1(x,\epsilon)$ as with $z_1 := P_{B_1(x,\epsilon)}(u)$

$$\|z_1 - x\| \leq \epsilon,$$

and it holds with $P_H(z) = \max\{0, \min\{z, 1\}\}$ and $z_A := P_H(z_1) = A(u)$ that

$$|z_{1,i} - x_i| = |z_{1,i} - z_{A,i} + z_{A,i} - x_i| = |z_{1,i} - z_{A,i}| + |z_{A,i} - x_i|,$$

which follows as if $z_{1,i} > 1$ then $z_{1,i} - z_{A,i} = z_{1,i} - 1 > 0$ and $1 - x_i \geq 0$ whereas if $z_{1,i} < 0$ then $z_{1,i} - z_{A,i} = z_{1,i} - 0 < 0$ and $0 - x_i \leq 0$. Thus we get

$$\|z_1 - x_1\|_1 = \|z_1 - z_A\|_1 + \|z_A - x\|_1 , \tag{4.31}$$

Thus it holds $\|z_A - x\|_1 \leq \|z_1 - x\|_1 \leq \epsilon$ and we get $A(u) \in H \cap B_1(x, \epsilon)$. and thus it is a feasible point of the optimization problem in (4.28) and by the optimality of $z_e := P_{B_1(x,\epsilon)\cap H}(u)$ it follows

$$\|z_e - u\|_2 \leq \|A(u) - u\|_2 .$$

If $\|z_1 - x\|_1 < \epsilon$ then $z_1 = u$ and thus with (4.31) one gets $\|z_A - x\|_1 < \epsilon$ as well. In this case z_A is the optimal projection if we just had the box constraints. However, as $\|z_A - x\|_1 < \epsilon$ it is also feasible for (4.28) and thus optimal, $z_e = z_A$. Moreover, if $\|u - x\|_1 = \epsilon$ then $z_1 = u$ and thus with the same argument we get $z_e = z_A$. On the other hand if $\|z_1 - x\|_1 = \epsilon$ and $z_1 \in H$, then z_1 is feasible for (4.28) and the minimum over a larger set and thus $z_e = z_1 = z_A$.

Suppose now that $\|u - x\|_1 > \epsilon$ and $z_1 \notin H$. Then $\|z_1 - u\|_1 = \epsilon$ and there exists $\lambda_1^* > 0$ (optimal solution of the l_1-projection problem) such that

$$z_A = \begin{cases} \min\{u_i - \lambda_1^*, 1\} & \text{for } u_i \geq x_i \text{ and } \lambda_1^* \leq |u_i - x_i| \\ x_i & \text{for } \lambda_1^* > |u_i - x_i| \\ \max\{u_i + \lambda_1^*, 0\} & \text{for } u_i \leq x_i \text{ and } \lambda_1^* \leq |u_i - x_i| \end{cases},$$

where we have used that $u_i - \lambda_1^* \geq x_i$ for $u_i \geq x_i$ resp. $u_i + \lambda_1^* \leq x_i$ for $u_i \leq x_i$. Moreover, as $z_1 \notin H$ this implies that $\|z_A - x\|_1 < \|z_1 - x\|_1 = \epsilon$. Then if $\|z_e - x\|_1 = \epsilon$ (which implies $\lambda_e^* \leq \lambda_1^*$) there is nothing to prove (we get $\|z_e - x\|_1 > \|z_A - x\|_1$ and this represents the first condition in the Lemma for strict inequality) so suppose that $\lambda_e^* = 0$ (that is $\|z_e - x\|_1 < \epsilon$) and thus

$$z_e = \begin{cases} \min\{u_i, 1\} & \text{for } u_i \geq x_i \text{ and } |u_i - x_i| \geq 0 \\ x_i & \text{for } |u_i - x_i| < 0 \\ \max\{u_i, 0\} & \text{for } u_i \leq x_i \text{ and } |u_i - x_i| \geq 0 \end{cases}.$$

Then we get

$$|(z_e)_i - x_i| \geq |(z_A)_i - x_i| \quad \forall i = 1, \ldots, d.$$

and thus $\|z_e - x\|_1 \geq \|z_A - x\|_1$. In fact, a stronger property can be derived under an extra condition on u. As $\|z_1 - x\|_1 = \epsilon$ it must hold $\|u - x\|_1 \geq \epsilon$ and thus $u \neq x$. Now suppose that wlog $u_i > x_i$ and $u_i \in [0, 1]$ then $(z_A)_i = u_i - \lambda_1^*$ if $|u_i - x_i| \geq \lambda_1^*$ or $(z_A)_i = x_i$ if $|u_i - x_i| < \lambda_1^*$. However, in both cases we get

$$(z_e)_i - x_i = u_i - x_i > \max\{u_i - x_i - \lambda_1^*, 0\} \geq (z_A)_i - x_i,$$

and thus $\|z_e - x\|_1 > \|z_A - x\|_1$. If $u_i < x_i$ and $u_i \in [0, 1]$ then

$$(z_e)_i - x_i = u_i - x_i < \min\{u_i - x_i - \lambda_1^*, 0\} \leq (z_A)_i - x_i$$

and thus $\|z_e - x\|_1 > \|z_A - x\|_1$. □

The previous lemma shows that the approximation $A(u)$ of $P_S(u)$ used by Maini et al. [2020], Tramèr and Boneh [2019] is definitely suboptimal under relatively weak conditions and has a smaller l_1-distance to the target point x:

$$\|P_S(u) - x\|_1 > \|A(u) - x\|_1 .$$

Effectively, a part of S is hidden from the attack when $A(u)$ instead of $P_S(u)$ is used. In Fig. 4.13 we simulate the projections after the steepest descent step (4.36) (for varying level of sparsity) to get a realistic picture of the influence of the approximation $A(u)$ versus the exact projection $P_S(u)$. For sparse updates (less than 50 non-zero components) the approximation behaves quite poorly in comparison to the exact projection in the sense that the true projection is located at the boundary of the l_1-ball around the target point x whereas $A(u)$ is located far into the interior of the l_1-ball. Note that such sparse updates are used in SLIDE [Tramèr and Boneh, 2019].

This discrepancy between approximate and exact projection in turn leads to suboptimal performance both in the maximization of the loss, which is important for adversarial training, but also in terms of getting low robust accuracy: the plots in Fig. 4.14 show the performance of PGD-based attacks with $A(u)$ (dashed line) vs the same methods with the correct projection $P_S(u)$ (solid line). Our proposed l_1-APGD largely benefits from using $P_S(u)$ instead of $A(u)$, and this even slightly improves the existing l_1-versions of PGD, SLIDE and the one of Maini et al. [2020]. Thus we use in our scheme always the correct projection onto S.

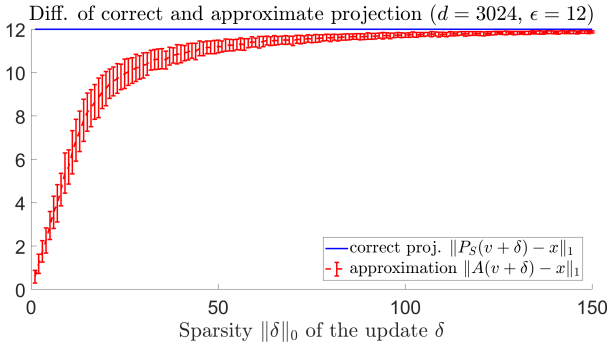

Figure 4.13: In order to simulate update steps of PGD, we sample target points $x \sim \mathcal{U}([0,1]^d)$ and $w \in \mathcal{N}(0,1)$ and define $v = \mathcal{P}_S(w)$ and project the point $u = v + \epsilon\delta$, where δ is generated as the descent step in (4.36) for different sparsity levels k. Then we plot $\|P_S(u) - x\|_1$ and $\|A(u) - x\|_1$ for varying sparsity k of the update (for each level k we show average and standard deviation over 100 samples). It can clearly be seen that the approximate projection is highly biased towards interior regions of the set $S = B_1(x, \epsilon) \cap [0,1]^d$ in particular for small levels of k whereas the correct projection stays on the surface of S. Note that SLIDE uses $k = 31$ (corresponds to 99% quantile) and thus is negatively affected by this strong bias as effectively large portions of the threat model S cannot be easily explored by SLIDE.

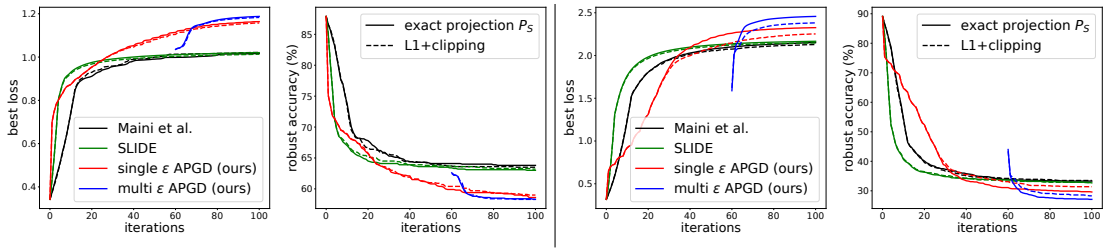

Figure 4.14: Plots of the best robust loss obtained so far (first/third) and robust accuracy (second/fourth) as a function of the iterations for the l_1-PGD of Tramèr and Boneh [2019] (SLIDE), the one of Maini et al. [2020] and our single ϵ-APGD and multi ϵ-APGD for two models (left: our own l_1-robust model APGD-AT, right: the l_2-robust model of Rice et al. [2020]). All of them are once run with the correct projection $P_S(u)$ (solid) and once with the approximation $A(u)$ (dashed). The exact projection improves in almost all cases for all attacks loss and robust accuracy. Moreover, our single- and multi-ϵ APGD improve significantly the robust loss as well as robust accuracy over SLIDE and the l_1-PGD of Maini et al. [2020]. Multi ϵ-APGD is only partially plotted as only the last 40% of iterations are feasible.

4.4.3 Descent direction

The next crucial step in the PGD scheme in (4.26) is the choice of the descent direction which we wrote as the mapping $s(\nabla f(x_i))$ of the gradient. For the l_∞- and l_2-threat models [Madry et al., 2018], the steepest descent direction [Boyd and Vandenberghe, 2004] is used in PGD, that is

$$\delta_p^* = \arg\max_{\delta \in \mathbb{R}^d} \langle w, \delta \rangle \quad \text{s.th.} \quad \|\delta\|_p \leq \epsilon, \tag{4.32}$$

with $w = \nabla f(x^{(i)}) \in \mathbb{R}^d$, which maximizes a linear function over the given l_p-ball. Thus one gets $\delta_\infty^* = \epsilon \operatorname{sign}(w)$ and $\delta_2^* = \epsilon w / \|w\|_2$ for $p = \infty$ and $p = 2$ respectively, which define the function s in (4.26). For $p = 1$, defining $j = \arg\max_i |w_i|$ the dimension corresponding to the component of w with largest absolute value and $\mathcal{B} = \{e_i\}_i$ the standard basis of $\mathbb{R}^d$, we have $\delta_1^* = \epsilon \operatorname{sign}(w_j) e_j$. Obviously, for a small number of iterations this descent direction is not working well and thus in SLIDE suggest to use the top-k components of the gradient (ordered according to their magnitude) and use the sign of these components.

In the following we show that when one takes into account the box-constraints imposed by the image domain the steepest descent direction becomes automatically less sparse and justifies at least partially what has been done by Tramèr and Boneh [2019] and Maini et al. [2020] out of efficiency reasons. More precisely, the following optimization problem defines the steepest

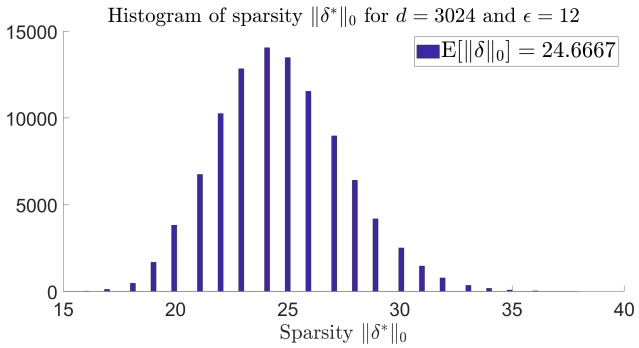

Figure 4.15: Histogram of the sparsity level $\|\delta^*\|_0$ of the steepest descent step from Proposition 4.4.3 for $d = 3024$ and $\epsilon = 12$ (histogram computed using 100.000 samples from $x \in \mathcal{U}([0,1]^d)$ and $w \in \mathcal{N}(0, \mathbb{I})$). The exact expected sparsity can be computed as $\mathbb{E}[\|\delta^*\|_0] \approx 24.6667$.

descent direction:

$$\delta^* = \arg\max_{\delta \in \mathbb{R}^d} \langle w, \delta \rangle \quad \text{s.th.} \quad \|\delta\|_1 \leq \epsilon, \quad x + \delta \in [0,1]^d. \tag{4.33}$$

Proposition 4.4.2 *Let* $z_i = \max\{(1 - x_i)\operatorname{sign}(w_i), -x_i \operatorname{sign}(w_i)\}$, π *the ordering such that* $|w_{\pi_i}| \geq |w_{\pi_j}|$ *for* $i > j$ *and* k *the smallest integer for which* $\sum_{i=1}^{k} z_{\pi_i} \geq \epsilon$, *then the solution of* (4.33) *is given by*

$$\delta^*_{\pi_i} = \begin{cases} z_{\pi_i} \cdot \operatorname{sign}(w_{\pi_i}) & \text{for } i < k, \\ (\epsilon - \sum_{i=1}^{k-1} z_{\pi_i}) \cdot \operatorname{sign}(w_{\pi_k}) & \text{for } i = k, . \\ 0 & \text{for } i > k \end{cases} \tag{4.34}$$

Proof. We introduce the new variable $\alpha_i := \operatorname{sign}(w_i)\delta_i$, with the convention $\operatorname{sign}(t) = 0$ if $t = 0$. Then we get the equivalent optimization problem:

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^d_+} \sum_i |w_i| \alpha_i \quad \text{s.th.} \quad & \sum_{i=1}^{d} \alpha_i \leq \epsilon, \quad \alpha_i \geq 0, \\ & -x_i \leq \operatorname{sign}(w_i)\alpha_i \leq 1 - x_i, \ i = 1, \ldots, d, \end{aligned} \tag{4.35}$$

and thus

$$0 \leq \alpha_i \leq \max\{-x_i \operatorname{sign}(w_i), (1 - x_i)\operatorname{sign}(w_i)\}.$$

Given the positivity of α and the upper bounds the maximum is attained when ordering $|w_i|$ in decreasing order π and setting always α_{π_i} to the upper bound until the budget $\sum_{i=1}^{d} \alpha_i = \epsilon$ is attained. Thus with k being the smallest integer in $[1, d]$ such that $\sum_{i=1}^{k} u_i \leq \epsilon$ we get the solution

$$\alpha^*_{\pi_i} = \begin{cases} z_{\pi_i} & \text{for } \ i < k, \\ (\epsilon - \sum_{i=1}^{k-1} z_{\pi_i}) & \text{for } \ i = k, . \\ 0 & \text{for } \ i > k \end{cases}$$

Noting that $\delta_i = \operatorname{sign}(w_i)\alpha_i$ we get

$$\delta^*_{\pi_i} = \begin{cases} z_{\pi_i} \operatorname{sign}(w_{\pi_i}) & \text{for } \ i < k, \\ (\epsilon - \sum_{i=1}^{k-1} z_{\pi_i}) \operatorname{sign}(w_{\pi_i}) & \text{for } \ i = k, . \\ 0 & \text{for } \ i > k \end{cases}$$

□

Proposition 4.4.2 shows that adding the box-constraints leads to a steepest descent direction δ^* of sparsity level k which depends on the gradient direction w and the target point x. Fig. 4.15 provides an empirical evaluation of the distribution of the sparsity level $\|\delta^*\|_0$. The following proposition computes the expected sparsity of the steepest descent step δ^* for $\epsilon \leq \frac{d-1}{2}$, together with a simple lower bound.

Proposition 4.4.3 *Let $w \in \mathbb{R}^d$ with $w_i \neq 0$ for all $i = 1, \dots, d$ and $x \in \mathcal{U}([0,1]^d)$. Then it holds for any $\frac{d-1}{2} \geq \epsilon > 0$,*

$$\mathbb{E}\big[\,\|\delta^*\|_0\,\big] = \lfloor \epsilon + 1 \rfloor + \sum_{m=\lfloor \epsilon \rfloor + 2}^{d} \sum_{k=0}^{\lfloor \epsilon \rfloor} (-1)^k \frac{(\epsilon - k)^{m-1}}{k!\,(m-1-k)!} \geq \frac{\lfloor 3\epsilon \rfloor + 1}{2}.$$

Proof. We first note that the components z_i defined in Proposition 4.4.2 are independent and have distribution $z_i \sim \mathcal{U}([0,1])$, $i = 1, \dots, d$ as both $1 - x_i$ and x_i are uniformly distributed and the x_i are independent. As the z_i are i.i.d. the distribution of k is independent of the ordering of w and thus we can just consider the probability $\sum_{i=1}^{k} z_i \geq \epsilon$. Note that for any $\epsilon > 0$, we have $1 \leq k \leq d$. Moreover, we note that

$$k > m \iff \sum_{i=1}^{m} z_i < \epsilon \quad \text{and} \quad k = d \iff \sum_{i=1}^{d} z_i \leq \epsilon,$$

and as k is an integer valued random variable we have

$$k \geq m \iff k > m - 1.$$

Thus as k is a non-negative integer valued random variable, we have

$$\begin{aligned}\mathbb{E}[k] &= \sum_{m=1}^{d} \mathrm{P}(k \geq m) = \sum_{m=1}^{d} \mathrm{P}\big(k > m-1\big) \\ &= \sum_{m=1}^{d} \mathrm{P}\Big(\sum_{i=1}^{m-1} z_i < \epsilon\Big) = \sum_{m=1}^{d} \mathrm{P}\Big(\sum_{i=1}^{m-1} z_i \leq \epsilon\Big)\end{aligned}$$

where in the last step we use that $\sum_{i=1}^{m-1} z_i$ is a continuous random variable and thus we add a set of measure zero. The sum of uniformly distributed random variables on $[0,1]$ has the Irwin-Hall distribution with a cumulative distribution function [Irwin, 1927, Hall, 1927] given by

$$\mathrm{P}\Big(\sum_{i=1}^{m-1} z_i \leq \epsilon\Big) = \frac{1}{(m-1)!} \sum_{k=0}^{\lfloor \epsilon \rfloor} (-1)^k \binom{m-1}{k} (\epsilon - k)^{m-1}.$$

Note that the distribution of $\sum_{i=1}^{m-1} z_i$ is symmetric around the mean value $\frac{m-1}{2}$ and thus the median is also $\frac{m-1}{2}$. We note that for $m = 1$ the first sum is empty and thus $\mathrm{P}\Big(\sum_{i=1}^{0} z_i \leq \epsilon\Big) = 1$ and in general as $0 \leq z_1 \leq 1$ it holds for $d - 1 \geq \epsilon \geq m - 1$:

$$\mathrm{P}\Big(\sum_{i=1}^{m-1} z_i \leq \epsilon\Big) = 1$$

Thus

$$\sum_{m=1}^{d} \mathrm{P}\Big(\sum_{i=1}^{m-1} z_i \leq \epsilon\Big) = \lfloor \epsilon + 1 \rfloor + \sum_{m=\lfloor \epsilon + 1 \rfloor + 1}^{d} \mathrm{P}\Big(\sum_{i=1}^{m-1} z_i \leq \epsilon\Big).$$

As the median is given by $\frac{m-1}{2}$ we get for $\epsilon \geq \frac{m-1}{2}$

$$\mathrm{P}\Big(\sum_{i=1}^{m-1} z_i \leq \epsilon\Big) \geq \frac{1}{2}$$

Algorithm 8: Single-ϵ APGD

Input: loss L, initial point x_{init}, feasible set S, N_{iter}, $\eta^{(0)}$, $k^{(0)}$, checkpoints M, input dimension d

Output: approximate maximizer of the loss x_{best}

$x^{(0)} \leftarrow x_{\text{init}}$, $x_{\text{best}} \leftarrow x_{\text{init}}$, $L_{\text{best}} \leftarrow L(x_{\text{init}})$
for $i = 0, \dots, N_{iter} - 1$ **do**
    `// adjust sparsity and step size`
    if $i + 1 \in M$ **then**
        $k^{(i+1)} \leftarrow$ sparsity as in Eq. (4.37)
        $\eta^{(i+1)} \leftarrow$ step size as in Eq. (4.38)
        if $\eta^{(i+1)} = \eta^{(0)}$ **then**
            $x^{(i)} \leftarrow x_{\text{best}}$
        end
    end
    `// update step`
    $u^{(i+1)} \leftarrow x^{(i)} + \eta^{(i)} \cdot s(\nabla L(x^{(i)}), k^{(i+1)} \cdot d)$
    $x^{(i+1)} \leftarrow P_S(u^{(i+1)})$
    `// update best point found`
    if $L(x^{(i+1)}) > L_{best}$ **then**
        $x_{\text{best}} \leftarrow x^{(i+1)}$, $L_{\text{best}} \leftarrow L(x^{(i+1)})$
    end
end

and thus

$$\sum_{m=1}^{d} \mathrm{P}\Big(\sum_{i=1}^{m-1} z_i \leq \epsilon\Big) \geq \lfloor \epsilon + 1 \rfloor + \frac{\lfloor 2\epsilon + 1 \rfloor - \lfloor \epsilon + 1 \rfloor}{2} + \sum_{m=\lfloor 2\epsilon+1 \rfloor + 1}^{d} \mathrm{P}\Big(\sum_{i=1}^{m-1} z_i \leq \epsilon\Big)$$
$$\geq \frac{\lfloor \epsilon + 1 \rfloor + \lfloor 2\epsilon + 1 \rfloor}{2} \geq \frac{\lfloor 3\epsilon \rfloor + 1}{2}$$

□

While the exact expression is hard to access, the derived lower bound $\frac{\lfloor 3\epsilon \rfloor - 1}{2}$ shows that the sparsity is non-trivially bounded away from 1. For a reasonable range of ϵ the expectation is numerically larger than 2ϵ. In this way we provide a justification for the heuristic non-sparse update steps used in Tramèr and Boneh [2019], Maini et al. [2020].

Finally, in our PGD scheme given $g = \nabla L(x^{(i)})$, $t \in \mathbb{N}$ and $T(t)$ the set of indices of the t largest components of $|g|$, we define the function s used in (4.26) via

$$h(t)_i = \begin{cases} \operatorname{sign}(g_i) & \text{if } i \in T(t) \\ 0 & \text{else} \end{cases}, \quad s(g,t) = h(t) / \left\| h(t) \right\|_1 \tag{4.36}$$

defines the function s used in (4.26). The form of the update is the same as in SLIDE which use a fixed k. However, as derived above, the sparsity level k of the steepest descent direction depends on ∇f and x and thus we choose k our scheme in a dynamic fashion depending on the current iterate, as described in the next section.

4.4.4 l_1-APGD minds $[0,1]^d$

The goal of our l_1-APGD is similar to that of APGD for l_2/l_∞. It should be parameter-free for the user and adapt the trade-off between exploration and local fine-tuning to the given budget of iterations.

Proposition 4.4.2 suggests the form of the steepest descent direction for the l_1-ball $\cap\, [0,1]^d$ threat model, which has an expected sparsity on the order of 2ϵ but the optimal sparsity depends

on the target point and the gradient of the loss. Thus fixing the sparsity of the update independent of the target point as in SLIDE is suboptimal. In practice we have $\epsilon \ll d$ (e.g. for CIFAR-10 $d = 3072$ and commonly $\epsilon = 12$) and thus 2ϵ sparse updates would lead to slow progress which is in strong contrast to the tight iteration budget used in adversarial attacks. Thus we need a scheme where the sparsity is adaptive to the chosen budget of iterations and depends on the current iterate. This motivates two key choices in our scheme: 1) we start with updates with low sparsity, 2) the sparsity of the updates is then progressively reduced and adapted to the sparsity of the difference of our currently best iterate (highest loss) to the target point. Thus, initially many coordinates are updated fostering fast progress and exploration of the feasible set, while later on we have a more local exploitation with significantly sparser updates. We observe that, although we do not enforce this actively, the average (over points) sparsity of the updates selected by our adaptive scheme towards the final iterations is indeed on the order of 2ϵ, i.e. around to what theoretically expected, although the exact value varies across models. In the following we describe the details of our l_1-APGD, see Alg. 8, and a multi-ϵ variant which increases the effectiveness, and finally discuss its use for adversarial training.

Single-ϵ l_1-APGD. Our scheme should automatically adapt to the total budget of iterations. Since the two main quantities which control the optimization in the intersection of l_1-ball and $[0, 1]^d$ are the sparsity of the updates and the step size, we propose to adaptively select them at each iteration. In particular, we adjust both every $m = \lceil 0.04 \cdot N_{\text{iter}} \rceil$ steps, with N_{iter} being the total budget of iterations, so that every set of parameters is applied for a minimum number of steps to achieve improvement. In the following we denote by $x^{(i)}_{\max}$ the point attaining the highest loss found until iteration i, and by $M = \{n \in \mathbb{N} \,|\, n \mod m = 0\}$ the set of iterations at which the parameters are recomputed, and describe how the adaptation is performed.

Selection of sparsity. We choose an update step for l_1-APGD whose sparsity is automatically computed by considering the best point found so far. In order to have both sufficiently fast improvements and a good exploration of the feasible set in the first iterations of the algorithm, we start with updates with $d/5$ nonzero elements, that is a sparsity $k^{(0)} = 0.2$ (in practice this implies $k^{(0)} \gg 2\epsilon/d$). Then, the sparsity of the updates is adjusted as

$$k^{(i)} = \begin{cases} \left\| x^{(i-1)}_{\max} - x \right\|_0 / (1.5 \cdot d) & \text{if } i \in M, \\ k^{(i-1)} & \text{else.} \end{cases} \tag{4.37}$$

Note that $k^{(i)}$ is smaller than the sparsity of the current best perturbation since the initial perturbations have much larger l_0-norm than the expected 2ϵ, and we want to refine them.

Selection of the step size. Simultaneously to the sparsity of the updates, we adapt the step size to the trend of the optimization. As high level idea: if the l_0-norm of the best solution is not decreasing significantly for many iterations, this suggests that it is close to the optimal value and that the step size is too large to make progress, then we reduce it. Conversely, when the sparsity of the updates keeps increasing we want to allow large step sizes since the region of the feasible set which can be explored by sparser updates is different from what can be seen with less sparse steps. We set the initial step size $\eta^{(0)} = \epsilon$ (the radius of the l_1-ball), so that the algorithm can search efficiently the feasible set, and then adjust the step size at iterations $i \in M$ according to

$$\eta^{(i)} = \begin{cases} \max\{\eta^{(i-m)}/1.5, \eta_{\min}\} & \text{if } k^{(i)}/k^{(i-m)} \geq 0.95, \\ \eta^{(0)} & \text{else,} \end{cases} \tag{4.38}$$

where $\eta_{\min} = \epsilon/10$ is the smallest value we allow for the step size. Finally, when the step size is set to its highest value, the algorithm restarts from $x^{(i)}_{\max}$.

Multi-ϵ l_1-APGD. In the described l_1-APGD all iterates belong to the feasible set S. However, since the points maximizing the loss are most likely on the low dimensional faces of S, finding them might require many iterations. We notice that the same points are instead in the interior of any l_1-ball with radius larger than ϵ. Thus we propose to split N_{iter} into three phases

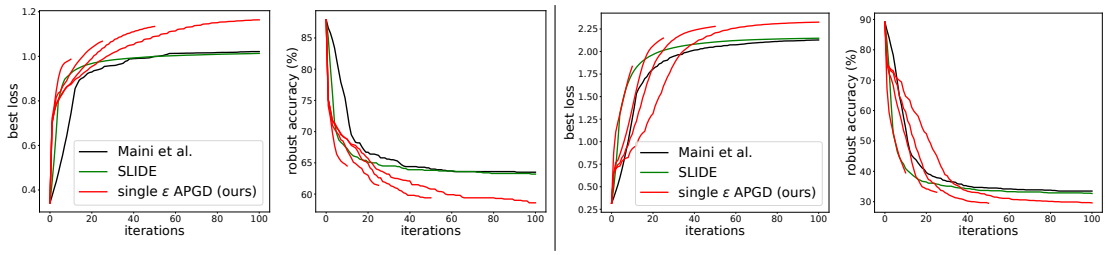

Figure 4.16: Plots of best robust loss obtained so far (first/third) and robust accuracy (second/fourth) over iterations for the l_1-PGD of Tramèr and Boneh [2019] (SLIDE), the one of Maini et al. [2020] and our single-ϵ l_1-APGD with 10 (the version used for adversarial training), 25, 50 and 100 iterations for two models (left: our own l_1-robust model, right: the l_2-robust one of Rice et al. [2020]). Since our method relies on an adaptive scheme, it automatically adjusts the parameters to the number of available iterations, outperforming the competitors.

with 30%, 30% and 40% of the iteration budget, where we optimize the objective L in l_1-balls of radii 3ϵ, 2ϵ and ϵ (always intersected with $[0,1]^d$) respectively. At the beginning of each phase the output of the previous one is projected onto the intersection of the next l_1-ball and $[0,1]^d$ and used as starting point for l_1-APGD with smaller radius. In this way we efficiently find good regions where to start the optimization in the target feasible set, see Fig. 4.14 for an illustration.

Comparison PGD vs single-ϵ and multi-ϵ l_1-APGD. In Fig. 4.14 we compare the performance of the versions of PGD used by Tramèr and Boneh [2019] and Maini et al. [2020] to our l_1-APGD, in both single- and multi-ϵ variants. For two models on CIFAR-10, we plot the best average (over 1000 test points) cross-entropy loss achieved so far and the relative robust accuracy (classification accuracy on the adversarial points), with a total budget of 100 iterations. For multi-ϵ APGD, we report the results only from iteration 60 onward, since before that point the iterates are outside the feasible set and thus the statistics not comparable. We see that the single-ϵ APGD achieves higher (better) loss and lower (better) robust accuracy than the existing PGD-based attacks, which tend to quickly plateau. Also, multi-ϵ APGD provides an additional improvement: exploring the larger l_1-ball yields an initialization in S with high loss, from where even a few optimization steps are sufficient to outperform the other methods. Additionally, we observe that using the exact projection (solid lines) boosts the effectiveness of the attacks compared to the approximated one (dashed lines), especially on the l_2-robust model of Rice et al. [2020] (the two rightmost plots). Moreover, we show in Fig. 4.16 how our single-ϵ APGD adapts to different budgets of iterations: when more steps are available, the loss improves more slowly at the beginning, favoring the exploration of the feasible set, but it finally achieves better values. Note that in this way, even with only 25 steps, l_1-APGD outperforms existing methods with 100 iterations both in terms of loss and robust accuracy attained.

4.4.5 Adversarial training with l_1-APGD

A natural application of a strong PGD-based attack is to maximize the loss in the inner maximization problem of adversarial training (AT) [Madry et al., 2018] and finding points attaining higher loss should lead to more adversarially robust classifiers. Prior works have shown that performing adversarial training wrt l_1 is a more delicate task than for other l_p-threat models: for CIFAR-10, Maini et al. [2020] report that using PGD wrt l_1 in AT led to severe gradient obfuscation, and the resulting model is less than 8% robust at $\epsilon = 12$. A similar effect is reported in Liu et al. [2020], where the B&B attack of Brendel et al. [2019] more than halves the robust accuracy computed by l_1-PGD used for their l_1-AT model. Both report the highest robustness to l_1-attacks when training for simultaneous robustness against different l_p-norms.

In our own experiment, we observe that using the *multi-step* PGD-based attack SLIDE with the standard sparsity of the updates $k = 0.01$ (here and in the following k indicates the percentage of non-zero elements) leads to catastrophic overfitting when training classifiers on CIFAR-10 with $\epsilon = 12$ and 10 steps. Here we mean by catastrophic overfitting that model is robust against the attack used during training even at test time but it fails completely against a stronger attack

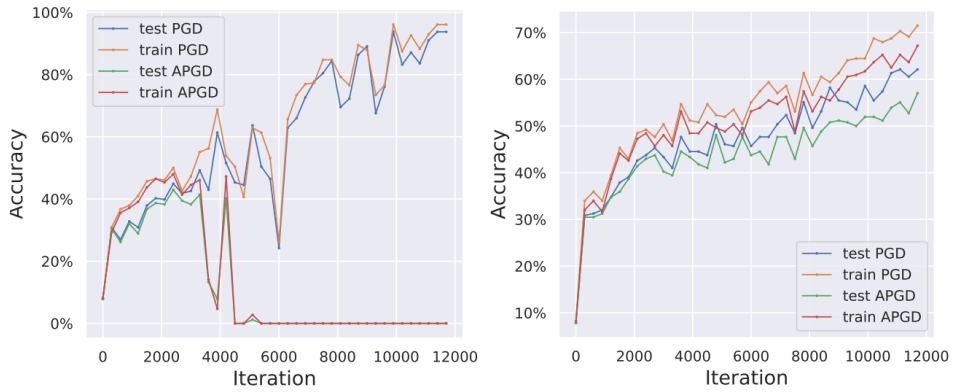

Figure 4.17: **Left:** Illustration of catastrophic overfitting in l_1 adversarial training when using the 10-step PGD-attack SLIDE [Tramèr and Boneh, 2019] with $k = 0.01$ for training. While the model is robust against the 10-step SLIDE attack, it is completely non-robust when running our stronger l_1-APGD attack with 100 iterations. **Right:** Catastrophic overfitting does not happen when using 10-step l_1-APGD for adversarial training to train AT-APGD, even when one attacks the model with much stronger attacks at test time, see the evaluation in Table 5.6. In both plots we report the robust accuracy on training and test set over the 30 epochs of training.

(APGD in the left part of Fig. 4.17). This is similar to what has been reported by Wong et al. [2020] in the l_∞-threat model using FGSM [Goodfellow et al., 2014], i.e. a single step method, in adversarial training might produce classifiers resistant to FGSM but not to a stronger PGD attack. Moreover, the robustness against PGD drops abruptly during training, on both training and test sets. In our scenario, as illustrated in Fig. 4.17 by the left plot, a similar phenomenon happens: when we use SLIDE in adversarial training the classifier is initially robust against both SLIDE with 10 steps (blue and yellow curves) and the stronger l_1-APGD (green and red) with 100 iterations, but around iteration 4000 the robustness computed with l_1-APGD goes close to 0%, while the model is still very robust against the attack used for training (which is a 10-step attack and not a single step attack). Conversely, when l_1-APGD (single-ϵ formulation with initial sparsity $k^{(0)} = 0.05$) with 10 steps is used for training (right plot) the model stays robust against both l_1-APGD with 100 iterations and SLIDE (and other attacks as shown in Sec. 4.4.7).

Moreover, We could prevent catastrophic overfitting by decreasing the sparsity in SLIDE to $k = 0.05$, but this yields poor final robustness (around 50%) compared to what we achieved using l_1-APGD (59.7% against multiple attacks, see Table 5.5 for the detailed evaluation), since a weaker attack is used at training time. In fact, we show in Sec. 4.4.9 that values $k > 0.01$ in SLIDE lead to worse performance in most of the cases. We hypothesize that l_1-APGD, adapting the sparsity of the updates per point, first prevents the model from seeing only perturbations with similar sparsity and second is able to effectively maximize the loss, allowing adversarial training to perform. We note that even Tramèr and Boneh [2019] observed that using the standard fixed value of k leads to overfitting, and proposed as remedies random sparsity levels and 20 steps. We instead keep the usual 10 steps, and have a scheme which adaptively chooses the sparsity rather than randomly. We leave it to future work to do a more thorough investigation of this interesting phenomenon.

4.4.6 l_1-Square Attack

In Sec. 4.2 we have introduced Square Attack, a query efficient score-based black-box adversarial attack for l_∞- and l_2-bounded perturbations. It is based on random search and does not rely on any gradient estimation technique. We adapt its l_2 version to our l_1-threat model, by modifying Alg. 4 (Sec. 4.2.3) so that all normalization operations are computed wrt the l_1-norm. For $p \in \{2, \infty\}$ we create at every iteration perturbations on the surface of the l_p-ball and then clip them to $[0, 1]^d$, but this results in poor performance for the l_1-ball, likely due to the complex structure of the intersection of l_1-ball and $[0, 1]^d$ (see discussion above). Thus, at each iteration, we upscale the square-shaped candidate update by a factor of 3 and then project the resulting iterate onto the intersection of the l_1-ball and $[0, 1]^d$ and accept this update if it increases the

Algorithm 9: Sampling distribution in Square Attack for $l_1 \cap [0,1]^d$

Input: target point x of shape $h \times h \times c$, radius ϵ, current iterate $x^{(i)}$, size of the windows w

Output: new update δ

$\nu \leftarrow x^{(i)} - x$

sample uniformly $r_1, s_1, r_2, s_2 \in \{0, \dots, w-h\}$

$W_1 := r_1 + 1 : r_1 + h, s_1 + 1 : s_1 + h$, $W_2 := r_2 + 1 : r_2 + h, s_2 + 1 : s_2 + h$

$\epsilon_{\text{unused}} \leftarrow \epsilon - \|\nu\|_1$

$\eta^* \leftarrow \eta / \|\eta\|_1$ with η as in Eq. 4.15 (Sec. 4.2)

for $i = 1, \dots, c$ **do**

  $\rho \leftarrow Uniform(\{-1, 1\})$

  $\nu_{\text{temp}} \leftarrow \rho\eta^* + \nu_{W_1,i} / \|\nu_{W_1,i}\|_1$

  $\epsilon^i_{\text{avail}} \leftarrow \|\nu_{W_1 \cup W_2,i}\|_1 + \epsilon_{\text{unused}}/c$

  $\nu_{W_2,i} \leftarrow 0, \quad \nu_{W_1,i} \leftarrow (\nu_{\text{temp}} / \|\nu_{\text{temp}}\|_1)\epsilon^i_{\text{avail}}$

end

$z \leftarrow P_S(x + 3\nu)$

$\delta \leftarrow z - x^{(i)}$

loss. This procedure increases the sparsity of the iterates and in turn the effectiveness of the attack, showing again the different role that the box has in the l_1-threat model compared to the l_∞- and l_2-threat models.

We summarize the sampling distribution for the $l_1 \cap [0,1]^d$ threat model in Alg. 9. The input w controls the size of the update and is progressively reduced according to a piecewise constant schedule, in turn regulated by the only free parameter of the method p. We show below that our resulting scheme, l_1-Square Attack, outperforms the existing black-box methods [Schott et al., 2019, Zhao et al., 2019] on a variety of models, often with margin.

4.4.7 Experiments

In the following we test the effectiveness of our proposed attacks.[9] Here, we compare l_1-APGD (with CE loss) and our l_1-Square Attack to existing white- and black-box attacks in the low budget regime, that is without random restarts. We will include a comparison in the high budget regime in Sec. 5.1.3. All attacks are evaluated on models trained on CIFAR-10, CIFAR-100 and ImageNet, and we report robust accuracy at $\epsilon = 12$ for CIFAR-10 and CIFAR-100 and $\epsilon = 60$ for ImageNet on 1000 test points.

Models. The selected models are (almost all) publicly available and are representative of different architectures and training schemes: the models of Carmon et al. [2019], Engstrom et al. [2019a], Xiao et al. [2020], Kim et al. [2020], Xu and Yang [2020] are robust wrt l_∞, where the model of Xiao et al. [2020] is known to be non-robust but shows heavy gradient obfuscation, those of Augustin et al. [2020], Engstrom et al. [2019a], Rice et al. [2020] wrt l_2, while those of Maini et al. [2020], Madaan et al. [2021] are trained for simultaneous robustness wrt l_∞-, l_2- and l_1-attacks. To our knowledge no prior work has focused on training robust models for solely l_1 (see discussion in Sec. 4.4.5). Additionally, we include the classifier we trained with l_1-APGD integrated in the adversarial training of Madry et al. [2018] and indicated as APGD-AT (see Sec. 4.4.5).

Attacks. We compare our attacks to the existing SOTA attacks for the l_1-threat model using their existing code (see Sec. 4.4.8 for hyperparameters). In detail, we consider SLIDE [Tramèr and Boneh, 2019] (which is based on PGD), EAD [Chen et al., 2018], FAB$^{\text{T}}$ (the targeted version of FAB introduced in Sec. 4.1.6), B&B [Brendel et al., 2019] and the recent ALMA [Rony et al., 2021]. As reported in Rony et al. [2021] B&B crashes as the initial procedure to sample uniform noise to get a decision different from the true class fails. Thus we initialize B&B with

[9] Code available at https://github.com/fra31/auto-attack.

Table 4.22: Robust accuracy achieved by the SOTA l_1 adversarial attacks on various models and datasets. The statistics are computed on 1000 points of the test set. PA and Square are black-box attacks. The budget is 100 iterations for white-box attacks ($\times 9$ for EAD and $+10$ for B&B) and 5000 queries for our l_1-Square-Attack. * indicates retrained models.

model	clean	EAD	ALMA	SLIDE	B&B	FAB$^{\text{T}}$	APGD$_{\text{CE}}$	PA	Square
CIFAR-10 ($\epsilon = 12$)									
APGD-AT **(ours)**	87.1	64.6	65.0	66.6	62.4	67.5	**61.3**	79.7	71.8
Madaan et al. [2021]	82.0	55.3	58.1	56.1	55.2	56.8	**54.7**	73.1	62.8
Maini et al. [2020] - AVG	84.6	51.8	54.2	53.8	52.1	61.8	**50.4**	77.4	68.4
Maini et al. [2020] - MSD	82.1	51.6	55.4	53.2	50.7	54.6	**49.7**	72.7	63.5
Augustin et al. [2020]	91.1	48.9	50.7	48.8	42.1	50.4	**37.1**	73.2	56.8
Engstrom et al. [2019a] - l_2	91.5	40.3	46.4	35.1	36.8	39.9	**30.2**	71.7	52.7
Rice et al. [2020]	89.1	37.7	45.2	32.3	35.2	37.0	**27.1**	70.5	50.3
Xiao et al. [2020]	79.4	44.9	74.5	33.3	72.6	78.9	41.4	36.2	**20.2**
Kim et al. [2020]*	81.9	26.7	31.8	25.1	23.8	32.4	**18.9**	54.9	36.0
Carmon et al. [2019]	90.3	25.1	18.4	19.7	18.7	31.1	**13.1**	60.8	34.5
Xu and Yang [2020]	83.8	20.1	24.0	18.2	14.7	27.8	**10.9**	57.0	32.0
Engstrom et al. [2019a] - l_∞	88.7	14.5	19.4	14.2	12.2	20.9	**8.0**	57.6	28.0
CIFAR-100 ($\epsilon = 12$)									
Rice et al. [2020] - l_2	58.7	19.5	24.4	17.7	19.3	19.6	**14.6**	37.0	23.8
Rice et al. [2020] - l_∞	54.5	8.6	9.9	6.0	6.5	13.0	**4.5**	23.0	10.6
ImageNet ($\epsilon = 60$)									
Engstrom et al. [2019a] - l_2	56.6	45.6	50.8	44.7	44.4	44.8	**43.6**	-	50.2
Engstrom et al. [2019a] - l_∞	61.9	11.7	26.6	11.5	9.5	34.6	**6.3**	-	23.9

random images from CIFAR-100 (the results do not improve when starting at CIFAR-10 images). Besides white-box methods we include the black-box Pointwise Attack (PA) [Schott et al., 2019], introduced for the l_0-threat model but successfully used as l_1-attack by e.g. Maini et al. [2020]. We always use our l_1-APGD in the multi-ϵ version.

CIFAR-10. We compare the attacks with a limited computational budget, i.e. 100 iterations, with the exception of EAD for which we keep the default 9 binary search steps (that is 9×100 iterations), B&B which performs an initial 10 step binary search procedure (10 additional forward passes). Moreover, we add the black-box attacks Pointwise Attack and our l_1-Square Attack with 5000 queries (no restarts). Table 4.22 reports the robust accuracy at $\epsilon = 12$ achieved by every attack: in all but one case l_1-APGD$_{\text{CE}}$ maximizing the cross-entropy loss outperforms the competitors, in 6 out of 11 cases with a gap larger than 4% to the second best method. Note that l_1-APGD$_{\text{CE}}$ consistently achieves lower (better) robustness with a quite significant gap to the non adaptive PGD-based attack SLIDE. Note also that l_1-APGD$_{\text{CE}}$, SLIDE and ALMA are the fastest attacks for the budget of 100 iterations (see Sec. 4.4.8 for more details). The model from Xiao et al. [2020] exemplifies the importance of testing robustness also with black-box attacks: their defense generates gradient obfuscation so that white-box attacks have difficulties to perform well (in particular ALMA, FAB$^{\text{T}}$ and B&B yield a robust accuracy close to the clean one), while Square Attack is not affected and achieves the best results with a large margin. Moreover, it outperforms on all models the other black-box attack PA. The most robust model is APGD-AT trained with our single-ϵ APGD for $\epsilon = 12$. It improves by 7.9% over the second best model Madaan et al. [2021] (see Table 5.5). This highlights how effective our l_1-APGD maximizes the target loss, even with only 10 steps used in AT.

Other datasets. We test the effectiveness of our proposed attacks on CIFAR-100 and ImageNet-1k, with $\epsilon = 12$ and $\epsilon = 60$ respectively, in the same setup above. For CIFAR-100 we use the models (PreAct ResNet-18) from Rice et al. [2020], for ImageNet those (ResNet-50) of Engstrom

Table 4.23: Comparison of black-box attacks in the l_1-threat model on CIFAR-10, $\epsilon = 12$. l_1-Square Attack outperforms both the Pointwise Attack [Schott et al., 2019] and the l_1-ZO-ADMM-Attack [Zhao et al., 2019] by large margin.

model	clean	ADMM	PA	Square
APGD-AT **(ours)**	87.1	86.3	79.7	**71.8**
Maini et al. [2020] - AVG	84.6	81.3	77.4	**68.4**
Maini et al. [2020] - MSD	82.1	77.5	72.7	**63.5**
Madaan et al. [2021]	82.0	78.4	73.1	**62.8**
Augustin et al. [2020]	91.1	88.9	73.2	**56.8**
Engstrom et al. [2019a] - l_2	91.5	89.8	71.7	**52.7**
Rice et al. [2020]	89.1	85.9	70.5	**50.3**
Kim et al. [2020]*	81.9	67.8	54.9	**36.0**
Carmon et al. [2019]	90.3	64.1	60.8	**34.5**
Xu and Yang [2020]	83.8	66.0	57.0	**32.0**
Engstrom et al. [2019a] - l_∞	88.7	69.3	57.6	**28.0**
Xiao et al. [2020]	79.4	78.5	36.2	**20.2**

et al. [2019a]: in both cases one classifier is trained for l_∞-robustness, the other one for l_2, all are publicly available.[10,11] On ImageNet, because of the different input dimension, we use $k = 0.001$ for SLIDE (after tuning it), and we do not run Pointwise Attack since it does not scale. For B&B we use random images not classified in the target class from the respective test or validation sets as starting points. We observe that on ImageNet, the gap in runtime between B&B and the faster attacks increases significantly: for example, to run 100 steps for 1000 test points B&B takes 3612 s, that is around 14 times more than l_1-APGD (254 s). Thus B&B scales much worse to high-resolution datasets. Also, while B&B and l_1-APGD have in principle a similar budget in terms of forward/backward passes, one could do much more restarts for l_1-APGD in the same time as for B&B. We report in Table 4.22 the robust accuracy given by every attack on 1000 points of test set of CIFAR-100 or validation set of ImageNet. Similarly to CIFAR-10, our l_1-APGD achieves the best results for all models in the low budget regime with a significant gap to the second best, either B&B or SLIDE.

Comparison of black-box attacks. While many black-box attacks are available for the l_∞- and l_2-threat model, and even a few have recently appeared for l_0, the l_1-threat model has received less attention: in fact, Tramèr and Boneh [2019], Maini et al. [2020] used the Pointwise Attack, introduced for l_0, to test robustness wrt l_1. To our knowledge only Zhao et al. [2019] have proposed l_1-ZO-ADMM, a black-box method to minimize the l_1-norm of the adversarial perturbations, although only results on MNIST are reported. Since no code is available for ZO-ADMM for l_1, we adapted the l_2 version following Zhao et al. [2019] (see Sec. 4.4.8). As for our l_1-Square Attack, we give to l_1-ZO-ADMM a budget of 5000 queries of the classifier. Table 4.23 shows the robust accuracy on 1000 test points achieved by the three black-box attacks considered on CIFAR-10 models, with $\epsilon = 12$: Square Attack outperforms the other methods on all models, with a significant gap to the second best. Note that l_1-ZO-ADMM does not consider norm-bounded attacks, but minimizes the norm of the modifications. While it is most of the time successful in finding adversarial perturbations, it cannot reduce their l_1-norm below the threshold ϵ within the given budget of queries.

4.4.8 Experimental details

We here report details about the experimental setup used in Sec. 4.4.7.

Models. Almost all the models we used are publicly available: the classifiers from Engstrom et al. [2019a], Carmon et al. [2019], Rice et al. [2020], Augustin et al. [2020] are provided in the library RobustBench[12] (see also Sec. 5.2). Those of Maini et al. [2020], Xu and Yang [2020]

[10] https://github.com/locuslab/robust_overfitting
[11] https://github.com/MadryLab/robustness/tree/master/robustness
[12] https://robustbench.github.io/

Table 4.24: Architecture of the models used in the experimental evaluation on CIFAR-10.

model	*architecture*
APGD-AT **(ours)**	PreAct ResNet-18
Madaan et al. [2021]	WideResNet-28-10
Maini et al. [2020] - AVG	PreAct ResNet-18
Maini et al. [2020] - MSD	PreAct ResNet-18
Augustin et al. [2020]	ResNet-50
Engstrom et al. [2019a] - l_2	ResNet-50
Rice et al. [2020]	PreAct ResNet-18
Xiao et al. [2020]	DenseNet-121
Kim et al. [2020]	ResNet-18
Carmon et al. [2019]	WideResNet-28-10
Xu and Yang [2020]	ResNet-18
Engstrom et al. [2019a] - l_∞	ResNet-50

can be found in the official pages.[13,14] Moreover, Madaan et al. [2021], Xiao et al. [2020] made models available via OpenReview.[15,16] Upon request, Kim et al. [2020] could not give access to the original models out of privacy reasons. Therefore we trained new classifiers using the official code[17] following the suggested parameters. For the models denoted in Kim et al. [2020] by "RoCL" and "RoCL+rLE" we could reproduce both clean and robust accuracy wrt l_∞ with the original evaluation code, while for "RoCL+AT+SS" we could match the robust accuracy but not the clean one (here robust accuracy is the one computed using their code). However, we used this last one in our experiments since it is the most robust one wrt l_1. For APGD-AT we trained a PreAct ResNet-18 [He et al., 2016b] with softplus activation function, using cyclic learning rate with maximal value 0.1 for 100 epochs, random cropping and horizontal flipping as training set augmentations. We set 10 steps of APGD for maximizing the robust loss in the standard adversarial training setup [Madry et al., 2018]. Table 4.24 reports the architecture of every model. As mentioned in Sec. 4.4.7, we chose such models to have different architectures, training schemes and even training data, as Carmon et al. [2019], Augustin et al. [2020] use unlabeled data in their methods.

Attacks. In the following we report the details of the presented attacks. We use ALMA from Adversarial Library [Rony and Ben Ayed, 2020], with the default parameters for the l_1-threat models, in particular $\alpha = 0.5$ with 100 iterations, $\alpha = 0.9$ with 1000. EAD, B&B and Pointwise Attack (PA) are available in FoolBox [Rauber et al., 2017]: for EAD we use the l_1 decision rule and regularization $\beta = 0.01$, for B&B we keep the default setup, while PA does not have tunable parameters. We reimplemented SLIDE following the original code, according to which we set sparsity of the updates $k = 0.01$ for CIFAR-10 and step size $\eta = 3.06$, which is obtained rescaling the one used in Tramèr and Boneh [2019] $\eta = 2$ for $\epsilon = 2000/255$ (see below for a study of the effect of different values of k). Also, since no code is available for ZO-ADMM for l_1, we adapted the l_2 version following Zhao et al. [2019] and then optimized its parameters, using $\rho = 2$, $\gamma = 0.1$. Finally, we use FAB$^{\mathrm{T}}$ with 100 iterations and default parameters used later in also Sec. 5.1. For l_1-APGD we fix the values of all parameters to those mentioned in Sec. 4.4.4. Moreover, we set $p = 0.8$ in l_1-Square Attack.

Attacks runtime. Direct comparison of runtime is not necessarily representative of the computational cost of each method since it depends on many factors including implementation and tested classifier. We gave similar budget to (almost all) attacks: for the low budget comparison (see Table 4.22) we use 100 iterations, which are equivalent to 100 forward and 100 backward passes for ALMA, SLIDE and l_1-APGD, 110 forward and 100 backward passes for B&B (because of the initial binary search), 150 forward and 100 backward passes for FAB$^{\mathrm{T}}$. EAD has instead a

[13] https://github.com/locuslab/robust_union
[14] https://github.com/MTandHJ/amoc
[15] https://openreview.net/forum?id=tv8n52Xb04p
[16] https://github.com/iclrsubmission/kwta
[17] https://github.com/Kim-Minseon/RoCL

Table 4.25: Effect of the sparsity k of the updates in SLIDE Tramèr and Boneh [2019], whose default value is $k = 0.01$.

model	$k = 0.001$	$k = 0.003$	$k = 0.01$	$k = 0.03$	$k = 0.1$	APGD_{CE}
APGD-AT **(ours)**	74.5	70.1	66.6	64.4	65.7	**61.3**
Madaan et al. [2021]	61.9	58.2	56.1	57.0	66.2	**54.7**
Maini et al. [2020] - AVG	71.7	64.4	53.8	51.8	64.7	**50.4**
Maini et al. [2020] - MSD	63.6	58.5	53.2	51.7	62.2	**49.7**
Augustin et al. [2020]	60.9	53.0	48.8	59.3	74.1	**37.1**
Engstrom et al. [2019a] - l_2	49.6	40.0	35.1	47.4	67.4	**30.2**
Xiao et al. [2020]	**28.2**	30.1	33.3	36.1	45.4	41.4
Rice et al. [2020]	47.0	37.6	32.3	45.2	65.2	**27.1**
Kim et al. [2020]*	38.4	30.6	25.1	34.9	58.3	**18.9**
Carmon et al. [2019]	41.4	29.9	19.7	24.2	64.4	**13.1**
Xu and Yang [2020]	35.3	24.9	18.2	21.1	58.2	**10.9**
Engstrom et al. [2019a] - l_∞	34.0	23.6	14.2	17.0	59.4	**8.0**

Table 4.26: **Evaluation at larger threshold:** Robust accuracy achieved by the SOTA l_1-adversarial attacks on models for CIFAR-10 at $\epsilon = 16$.

model	clean	EAD	ALMA	SLIDE	B&B	FAB^{T}	APGD_{CE}	PA	Square
APGD-AT **(ours)**	87.1	55.2	55.3	59.1	52.7	59.7	**50.6**	78.5	66.4
Madaan et al. [2021]	82.0	46.2	50.0	47.7	46.0	48.2	**45.6**	70.4	58.5
Maini et al. [2020] - MSD	82.1	43.9	46.2	44.9	43.2	48.5	**40.9**	69.8	57.9
Maini et al. [2020] - AVG	84.6	43.7	44.9	45.9	43.0	55.3	**38.9**	75.1	62.9

9 times larger budget since we keep the default 9 binary search steps. As an example, when run using a classifier on CIFAR-10 with PreAct ResNet-18 as architecture, 1000 test points, ALMA and SLIDE take around 25 s, l_1-APGD 27 s, FAB^{T} 32 s, EAD 105 s, B&B 149 s.

4.4.9 Additional results

Effect of sparsity in SLIDE. Since the sparsity k of the updates is a key parameter in SLIDE, the PGD-based attack for l_1 proposed in Tramèr and Boneh [2019], we study the effect of varying k on its performance. In Table 4.25 we report the robust accuracy achieved by SLIDE with 5 values of $k \in \{0.001, 0.003, 0.01, 0.03, 0.1\}$ on the CIFAR-10 models used for the experiments in Sec. 4.4.7, with a single run of 100 iterations. As a reference, we also show the results of our l_1-APGD with the same budget (grey column). We observe that while the default value $k = 0.01$ performs best in most of the cases, for 3/12 models the lowest robust accuracy is obtained by $k = 0.03$, for 1/12 by $k = 0.001$. This means that SLIDE would require to tune the value of k for each classifier to optimize its performance. Moreover, l_1-APGD, which conversely automatically adapts the sparsity of the updates, outperforms the best out of the 5 versions of SLIDE in 11 out of 12 cases, with the only exception of the model from Xiao et al. [2020] which is know to present heavy gradient obfuscation and on which the black-box Square Attack achieves the best result (see Sec. 4.4.7).

Larger threshold ϵ. We here test that the performance of our attacks at the larger threshold $\epsilon = 16$. In Table 4.26 we run all the methods, with the lower budget, on the four most robust models: one can observe that even with a larger threshold our l_1-APGD outperforms the competitors on all models.

Chapter 5

Benchmarking Adversarial Robustness

Since the finding that state-of-the-art deep learning models are vulnerable to adversarial examples, achieving adversarially robust models has become one of the most studied topics in the machine learning community. The main difficulty of robustness evaluation is that it is a computationally hard problem even for simple l_p-bounded perturbations [Katz et al., 2017] and exact approaches [Tjeng et al., 2019] do not scale to large enough models, and then one has to rely on adversarial attacks. Due to the many broken defenses, the field is currently in a state where it is very difficult to judge the value of a new defense without an independent test. This limits the progress as it is not clear how to distinguish bad from good ideas. A seminal work to mitigate this issue are the guidelines for assessing adversarial defenses by Carlini et al. [2019a]. However, as we see in our experiments, even papers trying to follow these guidelines can fail in obtaining a proper evaluation. In our opinion the reason is that at the moment there is no protocol which works reliably and autonomously, and does not need the fine-tuning of parameters for every new defense.

AutoAttack. As discussed in Sec. 4.3, the widely used PGD attack, which is computationally cheap and performs well in many cases, may overestimate robustness. Another cause of poor evaluations is the lack of diversity among the attacks used, as most papers rely solely on the results given by PGD or weaker versions of it like FGSM. Of different nature are for example two attacks introduced in the previous sections: the white-box FAB (Sec. 4.1) and the black-box Square Attack (Sec. 4.2). Importantly, these methods have a limited amount of parameters which generalize well across classifiers and datasets. In Sec. 5.1, we combine two versions of Auto-PGD, which aims at overcoming the weaknesses of PGD (see Sec. 4.3), with FAB and Square Attack to form a parameter-free, computationally affordable and user-independent ensemble of complementary attacks to estimate adversarial robustness, named AutoAttack. Note that we do not argue that AutoAttack is the ultimate adversarial attack but rather that it can be used as minimal test for new defenses, since it reliably reaches good performance in all tested models, *without any hyperparameter tuning and at a relatively low computational cost.*

RobustBench. Since we show that AutoAttack is effective across models, datasets and threat models, we can use it as standard evaluation to benchmark the evolution of defensive mechanisms. Then, we introduced RobustBench, a standardized benchmark to track the progress of adversarial defenses (Sec. 5.2). For this, we need to impose some restrictions on the allowed classifiers, to exclude defenses which make standard attacks fail without improving robustness (e.g. randomized defenses, input quantization), and include adaptive attacks to complement the results of AutoAttack when available. Moreover, RobustBench allows us to collect a large pool of robust classifiers which are made easily accessible via its Model Zoo.

Evaluation of adaptive test-time defenses. Since defenses which perform some form of optimization at inference time to counter adversarial perturbations have recently become popular, we provide in Sec. 5.3 a categorization of such methods. Moreover, we discuss their possible

advantages and drawbacks, especially about how to evaluate their robustness. Finally, we test several existing adaptive test-time defenses: based on these experiments, we provide recommendations about how to conduct such evaluations. This complements the results of RobustBench whose restrictions exclude most of adaptive test-time defenses.

5.1 AutoAttack: an ensemble of parameter-free attacks

We combine two parameter-free versions of PGD, i.e. Auto-PGD with cross-entropy loss (APGD_{CE}) and on the targeted version of DLR loss (APGD_{DLR}), with two complementary attacks, FAB and Square Attack, to form the ensemble AutoAttack, which is automatic in the sense that it does not require to specify any free parameters[1].

Since we want our protocol to be effective, computationally affordable and general, we select the following variants of the attacks to compose AutoAttack: APGD_{CE} without random restarts, $\text{APGD}^{\text{T}}_{\text{DLR}}$, i.e. on the Targeted-DLR loss (see Eq. (4.25)), with 9 target classes, the targeted version of FAB, namely FAB^{T}, with 9 target classes and Square Attack with one run of 5000 queries. We use 100 iterations for each run of the white-box attacks. While the runtime depends on the model, its robustness and even the framework of the target network, APGD is the fastest attack, as it requires only one forward and one backward pass per iteration. The computational budget of AutoAttack is similar to what has been used, on average, in the evaluation of the defenses considered.

A key property of AutoAttack is the diversity of its components: APGD is a white-box attack aiming at any adversarial example within an l_p-ball, and we use it with different loss functions. Also, FAB minimizes the norm of the perturbation necessary to achieve a misclassification, and, although it relies on the gradient of the logits, it appears to be effective also on models affected by gradient masking as shown in Sec. 4.1.3. On the other hand, Square Attack is a score-based black-box attack for norm bounded perturbations which uses random search and does not exploit any gradient approximation. It outperforms other black-box attacks in terms of query efficiency and success rate and has been shown to be even competitive with white-box attacks (see Sec. 4.2). Both methods have few parameters which generalize well across models and datasets, so that we will keep them fixed for all experiments. The diversity within the ensemble helps for two reasons: first, there exist classifiers for which some of the attacks dramatically fail, but always at least one of them works well. Second, on the same model diverse attacks might have similar robust accuracy but succeed on different points: then considering the worst case over all attacks, as we do for AutoAttack, improves the performance.

Implementation details. The hyperparameters of all attacks in AutoAttack are fixed for all experiments across datasets, models and norms. In Sec. 5.1.2 we show that AutoAttack evaluates adversarial robustness reliably and cost-efficiently despite its limited budget and running fully automatic, without any hyperparameter tuning. We report all the hyperparameters used in AutoAttack. For APGD, we use as direction in the update step the sign of the gradient for the l_∞-threat model and, as common in literature (see e.g. Tsipras et al. [2019]), the normalized (wrt l_2) gradient for the l_2-threat model. Moreover, we set the momentum coefficient to $\alpha = 0.75$, $\rho = 0.75$ (Condition 1, see Sec. 4.3), initial step size $\eta^{(0)} = 2\epsilon$ where ϵ is the maximum l_p-norm of the perturbations. For FAB we keep the standard hyperparameters, but remove the final search (see Sec. 4.1.2). For Square Attack we keep the algorithm as presented in Sec. 4.2, using as initial value for the size of the squares $p = 0.8$. Moreover, we use the piecewise constant schedule for p as it is suggested for a query limit of 10000 without any rescaling (although we use only up to 5000 queries).

5.1.1 Untargeted vs targeted attacks

We here discuss why we choose the targeted versions of APGD_{DLR} and FAB, considering both the scalability and the effectiveness of the attacks. Note that we keep the untargeted formulation of the CE loss, as it is widely used and achieves the best results for randomized defenses (Table 5.3).

[1] AutoAttack is available at https://github.com/fra31/auto-attack.

Table 5.1: **Comparison untargeted vs targeted attacks.** We report clean test accuracy and robust accuracy achieved by APGD_{DLR}, $\text{APGD}^{\text{T}}_{\text{DLR}}$, FAB and FAB^{T}. Moreover, we show the difference between the results of the targeted and untargeted attacks, and boldface it when is negative (that is the targeted attack is stronger). FAB does not scale to CIFAR-100/ImageNet due to the large number of classes.

#	paper	clean	APGD_{DLR}	$\text{APGD}^{\text{T}}_{\text{DLR}}$	diff.	FAB	FAB^{T}	diff.
CIFAR-10 - l_∞ - $\epsilon = 8/255$								
1	Carmon et al. [2019]	89.69	60.64	59.54	**-1.10**	60.62	60.12	**-0.50**
2	Alayrac et al. [2019]	86.46	62.03	56.27	**-5.76**	58.20	56.81	**-1.39**
3	Hendrycks et al. [2019]	87.11	56.96	54.94	**-2.02**	55.40	55.27	**-0.13**
4	Rice et al. [2020]	85.34	55.72	53.43	**-2.29**	54.13	53.83	**-0.30**
5	Qin et al. [2019]	86.28	55.46	52.85	**-2.61**	53.77	53.28	**-0.49**
6	Engstrom et al. [2019a]	87.03	52.65	49.32	**-3.33**	50.37	49.81	**-0.56**
7	Kumari et al. [2019]	87.80	51.68	49.15	**-2.53**	49.87	49.54	**-0.33**
8	Mao et al. [2019]	86.21	50.33	47.44	**-2.89**	48.32	47.91	**-0.41**
9	Zhang et al. [2019a]	87.20	47.33	44.85	**-2.48**	45.66	45.39	**-0.27**
10	Madry et al. [2018]	87.14	46.03	44.28	**-1.75**	45.41	44.75	**-0.66**
11	Pang et al. [2020a]	80.89	44.56	43.50	**-1.06**	44.47	44.06	**-0.41**
12	Wong et al. [2020]	83.34	46.64	43.22	**-3.42**	44.05	43.74	**-0.31**
13	Shafahi et al. [2019]	86.11	44.56	41.64	**-2.92**	42.90	43.44	0.54
14	Ding et al. [2020]	84.36	50.26	41.74	**-8.52**	47.18	42.47	**-4.71**
15	Moosavi-Dezfooli et al. [2019]	83.11	40.29	38.50	**-1.79**	39.04	38.97	**-0.07**
16	Zhang and Wang [2019]	89.98	48.96	37.29	**-11.67**	40.84	38.48	**-2.36**
17	Zhang and Xu [2020]	90.25	49.40	37.54	**-11.86**	40.36	38.99	**-1.37**
18	Jang et al. [2019]	78.91	37.01	34.96	**-2.05**	35.54	35.50	**-0.04**
19	Kim and Wang [2020]	91.51	48.41	35.93	**-12.48**	38.88	35.41	**-3.47**
20	Moosavi-Dezfooli et al. [2019]	80.41	35.47	33.70	**-1.77**	34.08	34.08	**0.00**
21	Wang and Zhang [2019]	92.80	40.33	33.61	**-6.72**	33.51	31.19	**-2.32**
22	Wang and Zhang [2019]	92.82	36.58	29.73	**-6.85**	31.82	29.10	**-2.72**
23	Mustafa et al. [2019]	89.16	4.54	1.13	**-3.41**	1.12	0.71	**-0.41**
24	Pang et al. [2020a]	93.52	0.49	0.00	**-0.49**	0.03	0.00	**-0.03**
CIFAR-10 - l_∞ - $\epsilon = 0.031$								
1	Zhang et al. [2019b]	84.92	54.04	53.10	**-0.94**	53.79	53.45	**-0.34**
2	Atzmon et al. [2019]	81.30	44.50	41.16	**-3.34**	40.92	40.73	**-0.19**
3	Xiao et al. [2020]	79.28	31.27	32.34	1.07	79.28	79.28	**0.00**
CIFAR-100 - l_∞ - $\epsilon = 8/255$								
1	Hendrycks et al. [2019]	59.23	31.66	28.48	**-3.18**	-	28.74	-
2	Rice et al. [2020]	53.83	20.25	18.98	**-1.27**	-	19.24	-
MNIST - l_∞ - $\epsilon = 0.3$								
1	Zhang et al. [2020b]	98.38	94.77	94.88	0.11	95.60	96.84	1.24
2	Gowal et al. [2019a]	98.34	93.84	93.93	0.09	94.72	97.03	2.31
3	Zhang et al. [2019b]	99.48	93.96	93.58	**-0.38**	94.12	94.62	0.50
4	Ding et al. [2020]	98.95	94.03	94.62	0.59	94.33	95.37	1.04
5	Atzmon et al. [2019]	99.35	94.54	94.16	**-0.38**	94.12	95.26	1.14
6	Madry et al. [2018]	98.53	89.75	90.57	0.82	91.75	93.69	1.94
7	Jang et al. [2019]	98.47	92.15	93.56	1.41	93.24	94.74	1.50
8	Wong et al. [2020]	98.50	85.39	86.34	0.95	87.30	88.28	0.98
9	Taghanaki et al. [2019]	98.86	0.00	0.00	**0.00**	0.02	0.01	**-0.01**
ImageNet - l_∞ - $\epsilon = 4/255$								
1	Engstrom et al. [2019a]	63.4	32.0	27.7	**-4.3**	-	28.4	-
CIFAR-10 - l_2 - $\epsilon = 0.5$								
1	Augustin et al. [2020]	91.08	74.94	72.91	**-2.03**	74.13	73.18	**-0.95**
2	Engstrom et al. [2019a]	90.83	70.20	69.24	**-0.96**	69.54	69.46	**-0.08**
3	Rice et al. [2020]	88.67	68.95	67.68	**-1.27**	68.03	67.97	**-0.06**
4	Rony et al. [2019]	89.05	67.02	66.44	**-0.58**	66.81	66.74	**-0.07**
5	Ding et al. [2020]	88.02	66.53	66.09	**-0.44**	66.43	66.33	**-0.10**
ImageNet - l_2 - $\epsilon = 3$								
1	Engstrom et al. [2019a]	55.3	30.9	28.3	**-2.6**	-	28.5	-

In Table 5.1 we compare the untargeted attacks, APGD_{DLR} and FAB, to their respective targeted versions in terms of the robust accuracy achieved on classifiers trained with recently proposed adversarial defenses (see below for details), for different threat models. We use the untargeted methods with 5 random restarts, the targeted ones with 9 target classes, those attaining the 9 highest scores at the original point (excluding the correct one). One can see that on CIFAR-10, CIFAR-100 and ImageNet in almost all the cases (35/36 for APGD_{DLR}, 29/32 for FAB) the targeted attacks provide lower (better) results, sometimes with a large gap especially for $\text{APGD}^{\text{T}}_{\text{DLR}}$. Although on MNIST the opposite situation occurs, we show in the next section that Square Attack is a stronger adversary than both $\text{APGD}^{\text{T}}_{\text{DLR}}$ and FAB^{T} for this dataset, and thus we favor the targeted attacks when selecting our ensemble in AutoAttack.

5.1.2 Experiments

In order to test the performance of AutoAttack, but also the individual performances of each attack, we evaluate the adversarial robustness in the l_∞- and l_2-threat models of over 50 models of 35 defenses from recent conferences like ICML, NeurIPS, ICLR, ICCV, CVPR, using MNIST, CIFAR-10, CIFAR-100 and ImageNet as datasets. We report first results for deterministic defenses and then for randomized ones, i.e. classifiers with a stochastic component. AutoAttack improves almost all evaluations, showing that its good performance generalizes well across datasets, models and threat models with the same hyperparameters.

Deterministic defenses. In Tables 5.2 and 5.4 we report the results on 49 models, 43 trained for l_∞- and 6 for l_2-robustness, from recent defense papers (for some of them multiple networks are considered, possibly on more datasets and norms). When possible we used the originals models (which are either publicly available or we obtained them via personal communication from the authors). Otherwise we retrained the models with the code released by the authors. For each classifier we report the clean accuracy and robust accuracy, at the ϵ specified in the table, on the whole test set (except for ImageNet where we use 1000 points from the validation set) obtained by the individual attacks APGD_{CE}, $\text{APGD}^{\text{T}}_{\text{DLR}}$, FAB^{T} and Square Attack, together with our ensemble AutoAttack, which counts as a success every point on which *at least one* of the four attacks finds an adversarial example (worst case evaluation). Additionally, we provide the *reported* robust accuracy of the respective papers (please note that in some cases their statistics are computed on a subset of the test set) and the difference between our robust accuracy and the reported one. The reduction is highlighted in red in the last column of Table 5.2 if it is negative (we get a lower robust accuracy). Finally, we boldface the attack which obtains the best individual robust accuracy and underline those achieving a robust accuracy lower than reported.

Notably, in all but one case AutoAttack achieves a lower robust accuracy than reported in the original papers, and the improvement is larger than 10% in 13 out of 49 cases, larger than 30% in 8 cases (AutoAttack yields also significantly lower values on the few models on CIFAR-100 and ImageNet). Thus AutoAttack would almost always have provided a better estimate of the robustness of the models than in the original evaluation, without any adaptation to the specific defense. In the only case where it does not reduce the reported robust accuracy it is only 0.03% far from it, and this result has been obtained with a variant of PGD with 180 restarts and 200 iterations (see Qin et al. [2019]), which is way more expensive than our evaluation.

In most of the cases more than one of the attacks included in AutoAttack achieves a lower robust accuracy than reported (APGD_{CE} improves the reported evaluation in 21/49 cases, $\text{APGD}^{\text{T}}_{\text{DLR}}$ in 45/49, FAB^{T} in 39/49 and Square Attack in 17/49, but 9/9 on MNIST). $\text{APGD}^{\text{T}}_{\text{DLR}}$ most often attains the best result for CIFAR-10, CIFAR-100 and ImageNet, Square Attack on MNIST. Also, $\text{APGD}^{\text{T}}_{\text{DLR}}$ is the most reliable one as it has the least severe failure which we define as the largest difference in robust accuracy to the best performing attack (maximal difference less than 12%, compared to 89% for APGD_{CE}, 59% for FAB^{T} and 70% for Square Attack). Thus our new DLR loss is able to resist gradient masking.

Note that, in Table 5.2 and Table 5.4, the models of Pang et al. [2019] and Pang et al. [2020a] appears twice as they are evaluated both with and without the addition of adversarial training. The models of Wang and Zhang [2019] are the networks named "R-MOSA-LA-4" and "R-MC-LA-4".

Table 5.2: **Robustness evaluation of adversarial defenses by AutoAttack.** We report clean test accuracy, the robust accuracy of the individual attacks as well as the combined one of AutoAttack (AA column). We also provide the robust accuracy reported in the original papers ("rep.") and compute the difference to the one of AutoAttack. If negative (in red) AutoAttack provides lower (better) robust accuracy. Note that APGD uses the (untargeted) cross-entropy loss, APGD^{T} the targeted DLR loss.

#	paper	clean	APGD	APGD^{T}	FAB^{T}	Square	AA	rep.	reduct.
CIFAR-10 - l_∞ - $\epsilon = 8/255$									
1	Carmon et al. [2019]	89.69	61.74	**59.54**	60.12	66.63	59.53	62.5	-2.97
2	Alayrac et al. [2019]	86.46	60.17	**56.27**	56.81	66.37	56.03	56.30	-0.27
3	Hendrycks et al. [2019]	87.11	57.23	**54.94**	55.27	61.99	54.92	57.4	-2.48
4	Rice et al. [2020]	85.34	57.00	**53.43**	53.83	61.37	53.42	58	-4.58
5	Qin et al. [2019]	86.28	55.70	52.85	53.28	60.01	52.84	52.81	0.03
6	Engstrom et al. [2019a]	87.03	51.72	**49.32**	49.81	58.12	49.25	53.29	-4.04
7	Kumari et al. [2019]	87.80	51.80	**49.15**	49.54	58.20	49.12	53.04	-3.92
8	Mao et al. [2019]	86.21	49.65	**47.44**	47.91	56.98	47.41	50.03	-2.62
9	Zhang et al. [2019a]	87.20	46.15	**44.85**	45.39	55.08	44.83	47.98	-3.15
10	Madry et al. [2018]	87.14	44.75	**44.28**	44.75	53.10	44.04	47.04	-3.00
11	Pang et al. [2020a]	80.89	57.07	**43.50**	44.06	49.73	43.48	55.0	-11.52
12	Wong et al. [2020]	83.34	45.90	**43.22**	43.74	53.32	43.21	46.06	-2.85
13	Shafahi et al. [2019]	86.11	43.66	**41.64**	43.44	51.95	41.47	46.19	-4.72
14	Ding et al. [2020]	84.36	50.12	**41.74**	42.47	55.53	41.44	47.18	-5.74
15	Moosavi-Dezfooli et al. [2019]	83.11	41.72	**38.50**	38.97	47.69	38.50	41.4	-2.90
16	Zhang and Wang [2019]	89.98	64.42	**37.29**	38.48	59.12	36.64	60.6	-23.96
17	Zhang and Xu [2020]	90.25	71.40	**37.54**	38.99	66.88	36.45	68.7	-32.25
18	Jang et al. [2019]	78.91	37.76	**34.96**	35.50	44.33	34.95	37.40	-2.45
19	Kim and Wang [2020]	91.51	56.64	35.93	**35.41**	61.30	34.22	57.23	-23.01
20	Moosavi-Dezfooli et al. [2019]	80.41	36.65	**33.70**	34.08	43.46	33.70	36.3	-2.60
21	Wang and Zhang [2019]	92.80	59.09	33.61	**31.19**	64.22	29.35	58.6	-29.25
22	Wang and Zhang [2019]	92.82	69.62	29.73	**29.10**	66.77	26.93	66.9	-39.97
23	Mustafa et al. [2019]	89.16	8.16	1.13	**0.71**	33.91	0.28	32.32	-32.04
24	Chan et al. [2020]	93.79	2.06	**0.53**	58.13	71.43	0.26	15.5	-15.24
25	Pang et al. [2020a]	93.52	89.48	**0.00**	**0.00**	35.82	0.00	31.4	-31.40
CIFAR-10 - l_∞ - $\epsilon = 0.031$									
1	Zhang et al. [2019b]	84.92	55.28	**53.10**	53.45	59.43	53.08	56.43	-3.35
2	Atzmon et al. [2019]	81.30	79.67	41.16	**40.73**	47.99	40.22	43.17	-2.95
3	Xiao et al. [2020]	79.28	39.99	32.34	79.28	**20.44**	18.50	52.4	-33.90
CIFAR-100 - l_∞ - $\epsilon = 8/255$									
1	Hendrycks et al. [2019]	59.23	33.02	**28.48**	28.74	34.26	28.42	33.5	-5.08
2	Rice et al. [2020]	53.83	20.57	**18.98**	19.24	23.57	18.95	28.1	-9.15
MNIST - l_∞ - $\epsilon = 0.3$									
1	Zhang et al. [2020b]	98.38	95.32	94.88	96.84	**93.97**	93.96	96.38	-2.42
2	Gowal et al. [2019a]	98.34	94.79	93.93	97.03	**92.88**	92.83	93.88	-1.05
3	Zhang et al. [2019b]	99.48	93.60	93.58	94.62	**92.97**	92.81	95.60	-2.79
4	Ding et al. [2020]	98.95	94.58	94.62	95.37	**91.42**	91.40	92.59	-1.19
5	Atzmon et al. [2019]	99.35	99.10	94.16	95.26	**90.86**	90.85	97.35	-6.50
6	Madry et al. [2018]	98.53	90.57	90.57	93.69	**88.56**	88.50	89.62	-1.12
7	Jang et al. [2019]	98.47	94.05	93.56	94.74	**88.00**	87.99	94.61	-6.62
8	Wong et al. [2020]	98.50	86.68	86.34	88.28	**83.07**	82.93	88.77	-5.84
9	Taghanaki et al. [2019]	98.86	30.50	**0.00**	0.01	**0.00**	0.00	64.25	-64.25
ImageNet - l_∞ - $\epsilon = 4/255$									
1	Engstrom et al. [2019a]	63.4	31.0	**27.7**	28.4	46.8	27.6	33.38	-5.78
CIFAR-10 - l_2 - $\epsilon = 0.5$									
1	Augustin et al. [2020]	91.08	74.70	**72.91**	73.18	83.10	72.91	73.27	-0.36
2	Engstrom et al. [2019a]	90.83	69.62	**69.24**	69.46	80.92	69.24	70.11	-0.87
3	Rice et al. [2020]	88.67	68.58	**67.68**	67.97	79.01	67.68	71.6	-3.92
4	Rony et al. [2019]	89.05	66.59	**66.44**	66.74	78.05	66.44	67.6	-1.16
5	Ding et al. [2020]	88.02	66.21	**66.09**	66.33	76.99	66.09	66.18	-0.09
ImageNet - l_2 - $\epsilon = 3$									
1	Engstrom et al. [2019a]	55.3	31.5	**28.3**	28.5	46.6	28.3	35.09	-6.79

Table 5.3: **Robustness evaluation of randomized l_∞-adversarial defenses by AutoAttack.** We report the clean test accuracy (mean and standard deviation over 5 runs) and the robust accuracy of the individual attacks as well as the combined on of AutoAttack (again over 5 runs). We also provide the robust accuracy reported in the respective papers and compute the difference to the one of AutoAttack (negative means that AutoAttack is better). The statistics of our attack are computed on the whole test set except for the ones of Yang et al. [2019], which are on 1000 test points due to the computational cost of this defense. The ϵ is the same as used in the papers. APGD^1 and APGD^2 indicate APGD_{CE} and APGD_{DLR} respectively.

#	paper	model	clean	APGD^1	APGD^2	FAB	Square	AA	rep.	red.
CIFAR-10 - $\epsilon = 8/255$										
1	Wang et al. [2019a]	En_5RN	$82.39_{\pm 0.14}$	**48.81**	49.37	-	78.61	$45.56_{\pm 0.20}$	51.48	-5.9
2	Yang et al. [2019]	w/ AT	$84.9_{\pm 0.6}$	**30.1**	31.9	-	-	$26.3_{\pm 0.85}$	52.8	-26.5
3	Yang et al. [2019]	pure	$87.2_{\pm 0.3}$	**21.5**	24.3	-	-	$18.2_{\pm 0.82}$	40.8	-22.6
4	Grathwohl et al. [2020]	JEM-10	$90.99_{\pm 0.03}$	**11.69**	15.88	63.07	79.32	$9.92_{\pm 0.03}$	47.6	-37.7
5	Grathwohl et al. [2020]	JEM-1	$92.31_{\pm 0.04}$	**9.15**	13.85	62.71	79.25	$8.15_{\pm 0.05}$	41.8	-33.6
6	Grathwohl et al. [2020]	JEM-0	$92.82_{\pm 0.05}$	**7.19**	12.63	66.48	73.12	$6.36_{\pm 0.06}$	19.8	-13.4
CIFAR-10 - $\epsilon = 4/255$										
1	Grathwohl et al. [2020]	JEM-10	$91.03_{\pm 0.05}$	**49.10**	52.55	78.87	89.32	$47.97_{\pm 0.05}$	72.6	-24.6
2	Grathwohl et al. [2020]	JEM-1	$92.34_{\pm 0.04}$	**46.08**	49.71	78.93	90.17	$45.49_{\pm 0.04}$	67.1	-21.6
3	Grathwohl et al. [2020]	JEM-0	$92.82_{\pm 0.02}$	**42.98**	47.74	82.92	89.52	$42.55_{\pm 0.07}$	50.8	-8.2

Randomized defenses. Another line of adversarial defenses relies on adding to a classifier some stochastic component. In this case the output (hence the decision) of the model might change across different runs for the same input. Thus we compute in Table 5.3 the mean (standard deviation in brackets) of our statistics over 5 runs. Moreover the results of AutoAttack are given considering, for each point, the attack performing better on average across 5 runs. To counter the randomness of the classifiers, for APGD we compute the direction for the update step as the average of 20 computations of the gradient at the same point (known as Expectation over Transformation [Athalye et al., 2018]) and use the untargeted losses (1 run). We do not run FAB here since it returns points on or very close to the decision boundary, so that even a small variation in the classifier is likely to undo the adversarial change. We modify Square Attack to accept an update if it reduces the target loss on average over 20 forward passes and, as this costs more time we use only 1000 iterations. For the models from Grathwohl et al. [2020], we attack that named JEM-0 with 5 restarts with the deterministic versions (i.e. without averaging across multiple passes of the networks), since the stochastic component has little influence, and then reuse the same adversarial examples on JEM-1 and JEM-10 (the results of FAB confirm that it is not suitable to test randomized defenses). Table 5.3 shows that AutoAttack achieves always lower robust accuracy than reported in the respective papers, with APGD_{CE} being the best performing attack, closely followed by APGD_{DLR}. In 7 out of 9 cases the improvement is significant, larger than 10% (and in 3/9 cases larger than 25%). Thus AutoAttack is also suitable for the evaluation of randomized defenses.

Note that for the models of Grathwohl et al. [2020], only plots of robust accuracy vs size of the perturbation ϵ are available in their paper. Thus the reported values are obtained by extrapolating the values in the tables from the most recent version of the paper, that is the one available at https://arxiv.org/abs/1912.03263v2.

Analysis of adversarial defenses. While the main goal of the evaluation is to show the effectiveness of AutoAttack, at the same time it provides an assessment of the SOTA of adversarial defenses. The most robust defenses rely on variations or fine-tuning of adversarial training introduced in Madry et al. [2018]. One step forward has been made by methods which use additional data for training, like Carmon et al. [2019] and Alayrac et al. [2019]. Moreover, several defenses which claim SOTA robustness turn out to be significantly less robust than Madry et al. [2018]. Interestingly, the most (empirically) resistant model on MNIST is one trained for obtaining provable certificates on the exact robust accuracy, and comes with a verified lower bound on it of 93.32% [Zhang et al., 2020b].

Table 5.4: **Robustness evaluation of additional adversarial defenses by AutoAttack.** We report clean test accuracy, the robust accuracy of the individual attacks and the combined one of AutoAttack (AA column). We also provide the robust accuracy reported in the original papers and compute the difference to that of AutoAttack. If negative (in red) AutoAttack provides lower (better) robust accuracy.

#	paper	clean	APGD_{CE}	$\text{APGD}^{\text{T}}_{\text{DLR}}$	FAB^{T}	Square	AA	rep.	reduct.
CIFAR-10 - l_∞ - $\epsilon = 4/255$									
1	Song et al. [2019a]	84.81	57.43	**56.51**	56.86	64.67	56.51	58.1	-1.59
CIFAR-10 - l_∞ - $\epsilon = 0.02$									
1	Pang et al. [2019]	91.22	5.02	**1.74**	2.45	29.52	1.02	34.0	-32.98
2	Pang et al. [2019]	93.44	0.18	**0.01**	0.06	88.55	0.00	30.4	-30.40

Recent advancements. After the publication of our work, several techniques improved SOTA of adversarial defenses, using for example better data augmentation [Gowal et al., 2021], improved loss functions [Rade and Moosavi-Dezfooli, 2021] or synthetic data [Rebuffi et al., 2021a, Wang et al.]. However all still rely on adversarial training as main component to achieve robustness.

5.1.3 Extension to the l_1-threat model

We here extend AutoAttack to the l_1-threat model. As noted in Sec. 4.4 this case might present some differences to the more common l_∞ and l_2. Then, for AutoAttack wrt l_1 we use our multi-ϵ l_1-APGD with 5 runs (with random restarts) of 100 iterations for the CE and the T-DLR loss (total budget of 1000 steps), FAB^{T} exploits 9 restarts of 100 iterations and l_1-Square Attack has a budget of 5000 queries (the total budget is the same as for the other threat models). Since there are no defenses focusing specifically on l_1, we use the various robust models from Sec. 4.4 to test our l_1-AutoAttack.

Experiments on CIFAR-10. We follow a similar setup to that in Sec. 4.4.7. However, this time we give the attacks a higher budget: we use SLIDE and B&B with 10 random restarts of 100 iterations (the reported accuracy is then the pointwise worst case over restarts). B&B has a default value of 1000 iterations but 10 restarts with 100 iterations each yield much better results. For ALMA and EAD we use 1000 resp. 9×1000 iterations (note that EAD does a binary search) since they do not have the option of restarts. We compare these strong attacks to the combination of l_1-APGD_{CE} and l_1-$\text{APGD}_{\text{T-DLR}}$ denoted as $\text{APGD}_{\text{CE+T}}$ with 100 iterations and 5 restarts each. These runs are also part of our ensemble l_1-AutoAttack which includes additionally, FAB^{T} (100 iterations and 9 restarts as used in l_∞- and l_2-AA) and Square Attack (5000 queries). Note that $\text{APGD}_{\text{CE+T}}$ has the same or smaller budget than the other attacks and it performs very similar to the full l_1-AutoAttack (AA). In Table 5.5 and Table 5.6 we report the results achieved by all methods for $\epsilon = 12$ resp. $\epsilon = 8$. AA outperforms for $\epsilon = 12$ the individual competitors in all cases except one, i.e. B&B on the APGD-AT model, where however it is only 0.5% far from the best. Also, note that all the competitors have at least one case where they report a robust accuracy more than 10% worse than AA. The same also holds for the case $\epsilon = 8$. Note that for $\epsilon = 12$ $\text{APGD}_{\text{CE+T}}$ is the best single attack in 10 out of 12 cases (B&B 3, SLIDE 1) and for $\epsilon = 8$ $\text{APGD}_{\text{CE+T}}$ is the best in 9 out of 12 cases (B&B 2, SLIDE 1).

Since AA is an ensemble of methods, as a stronger baseline we additionally report the worst case robustness across all the methods not included in AA (indicated as "WC" in the tables), that is EAD, ALMA, SLIDE, B&B and Pointwise Attack. l_1-AA achieves better results in most of the cases (7/12 for $\epsilon = 12$, 9/12 for $\epsilon = 8$) even though it has less than half of the total budget of WC. AA is 0.6% worse than WC for the APGD-AT model (at $\epsilon = 12$), which is likely due to the fact that l_1-APGD is used to generate adversarial examples at training time and thus there seems to be a slight overfitting effect to this attack. However, note that standard AT with l_1-PGD has been reported to fail completely. We report the results of the individual methods of l_1-AA below. We also list in Table 5.5 the l_1-robust accuracy reported in the original papers, if available. The partially large differences indicate that a standardized evaluation with AA would lead to a more reliable assessment of l_1-robustness.

Table 5.5: **Evaluation of l_1-robustness:** Robust accuracy achieved by the SOTA l_1 -adversarial attacks on various models and datasets. The statistics are computed on 1000 points of the test set. "WC" denotes the pointwise worst-case over all restarts/runs of EAD, ALMA, SLIDE, B&B and Pointwise Attack. Note that APGD, the combination of APGD_{CE} and $\text{APGD}_{\text{T-DLR}}$ (5 restarts each), yields a similar performance as AA (ensemble of $\text{APGD}_{\text{CE+T}}$, l_1-FAB^T and l_1-Square Attack) with the same or smaller budget than the other individual attacks. AA performs the same or beats the worst case WC of five SOTA l_1-attacks in 8 out of 12 cases. "rep." denotes the reported robust accuracy in the original papers. * the models of Kim et al. [2020] were not available on request and thus are retrained with their code. ** In Madaan et al. [2021] evaluation is done at $\epsilon = \frac{2000}{255}$, but by personal communication with the authors we found that the reported 55.0% corresponds to $\epsilon = 12$.

model	clean	EAD	ALMA	SLIDE	B&B	APGD	WC	AA	rep.
CIFAR-10 ($\epsilon = 12$)									
APGD-AT **(ours)**	87.1	63.3	61.4	65.9	59.9	60.3	**59.7**	60.3	-
Madaan et al. [2021]	82.0	54.5	54.3	55.1	51.9	51.9	**51.8**	51.9	55.0**
Maini et al. [2020] - AVG	84.6	50.0	49.7	52.3	49.0	**46.8**	47.3	**46.8**	54.0
Maini et al. [2020] - MSD	82.1	50.1	49.8	51.7	47.7	**46.5**	46.8	**46.5**	53.0
Augustin et al. [2020]	91.1	46.0	42.9	41.5	32.9	31.1	31.9	**31.0**	-
Engstrom et al. [2019a] - l_2	91.5	36.4	34.7	30.6	27.5	27.0	27.1	**26.9**	-
Rice et al. [2020]	89.1	33.9	32.4	28.1	24.2	24.2	**23.7**	24.0	-
Xiao et al. [2020]	79.4	34.4	75.0	22.5	59.3	27.2	20.2	**16.9**	-
Kim et al. [2020]*	81.9	24.4	22.9	19.9	15.7	15.4	**15.1**	**15.1**	81.18
Carmon et al. [2019]	90.3	26.2	13.6	13.6	10.4	**8.3**	8.5	**8.3**	-
Xu and Yang [2020]	83.8	18.1	14.5	13.9	7.8	7.7	**6.9**	7.6	59.63
Engstrom et al. [2019a] - l_∞	88.7	12.5	10.0	8.7	5.9	**4.9**	5.1	**4.9**	-
CIFAR-100 ($\epsilon = 12$)									
Rice et al. [2020] - l_2	58.7	18.4	17.1	15.5	13.1	**12.1**	12.7	**12.1**	-
Rice et al. [2020] - l_∞	54.5	8.1	5.5	4.5	3.0	3.4	**2.9**	3.1	-
ImageNet ($\epsilon = 60$)									
Engstrom et al. [2019a] - l_2	56.6	44.1	45.6	44.2	**40.3**	40.5	**40.3**	40.5	-
Engstrom et al. [2019a] - l_∞	61.9	9.6	17.1	8.5	6.2	4.6	5.8	**4.4**	-

Experiments on other datasets. We test the effectiveness of our proposed attacks on CIFAR-100 and ImageNet-1k, with $\epsilon = 12$ and $\epsilon = 60$ respectively, in the same setup of Sec. 4.4.7. Table 5.5 l_1-AutoAttack gives the lowest robust accuracy in 2/4 cases, improving up 1.4% over WC, the pointwise worst case over all attacks not included in AA, while in the other cases it is only 0.2% worse than WC, showing that it gives a good estimation of the robustness of the models.

Composition of l_1-AutoAttack. We report in Table 5.7 the individual performance of the 4 methods constituting l_1-AutoAttack, recalling that each version of l_1-APGD, with either cross-entropy or targeted DLR loss, is used with 5 runs of 100 iterations, FAB^{T} exploits 9 restarts of 100 iterations and l_1-Square Attack has a budget of 5000 queries. Note that the robust accuracy given by l_1-AutoAttack is in all cases lower than that of the best individual attack, which varies across models.

Table 5.6: **Evaluation at smaller radius ($\epsilon = 8$):** see Table 5.5 for details, the only change is the evaluation of robust accuracy at the smaller value $\epsilon = 8$.

model	clean	EAD	ALMA	SLIDE	B&B	$\text{APGD}_{\text{CE+T}}$	WC	AA
APGD-AT **(ours)**	87.1	71.6	71.8	72.9	**70.6**	**70.6**	**70.6**	**70.6**
Madaan et al. [2021]	82.0	62.6	63.3	62.7	**60.6**	60.7	**60.6**	**60.6**
Maini et al. [2020] - AVG	84.6	62.9	63.1	63.1	62.4	**60.3**	61.3	**60.3**
Maini et al. [2020] - MSD	82.1	60.2	61.0	61.6	58.6	58.3	**58.2**	**58.2**
Augustin et al. [2020]	91.1	60.9	60.1	60.8	52.6	**50.7**	52.0	**50.7**
Engstrom et al. [2019a] - l_2	91.5	54.0	54.9	52.3	46.0	44.4	45.9	**44.2**
Rice et al. [2020]	89.1	52.5	51.5	49.9	44.3	**42.9**	44.1	**42.9**
Kim et al. [2020]*	81.9	38.7	38.9	36.6	31.8	30.4	31.4	**30.1**
Xiao et al. [2020]	79.4	41.2	75.3	30.7	60.6	33.8	27.9	**22.4**
Carmon et al. [2019]	90.3	37.8	29.7	28.8	25.1	**21.1**	22.6	**21.1**
Xu and Yang [2020]	83.8	33.1	30.2	27.6	22.6	21.4	21.6	**21.0**
Engstrom et al. [2019a] - l_∞	88.7	28.8	24.9	23.1	17.5	16.5	16.4	**16.0**

Table 5.7: Individual performance of the components of l_1-AutoAttack.

model	clean	APGD_{CE}	$\text{APGD}^{\text{T}}_{\text{DLR}}$	FAB^{T}	Square	AA
CIFAR-10 ($\epsilon = 12$)						
APGD-AT **(ours)**	87.1	60.8	60.8	65.9	71.8	**60.3**
Madaan et al. [2021]	82.0	54.2	52.0	54.7	62.8	**51.9**
Maini et al. [2020] - AVG	84.6	48.9	47.5	59.5	68.4	**46.8**
Maini et al. [2020] - MSD	82.1	48.6	47.4	53.5	63.5	**46.5**
Augustin et al. [2020]	91.1	34.7	34.5	42.4	56.8	**31.0**
Engstrom et al. [2019a] - l_2	91.5	27.9	29.3	32.9	52.7	**26.9**
Rice et al. [2020]	89.1	25.5	26.3	30.3	50.3	**24.0**
Xiao et al. [2020]	79.4	32.2	33.4	78.6	20.2	**16.9**
Kim et al. [2020]*	81.9	17.0	16.9	22.2	36.0	**15.1**
Carmon et al. [2019]	90.3	9.9	9.9	21.5	34.5	**8.3**
Xu and Yang [2020]	83.8	9.6	9.3	17.7	32.0	**7.6**
Engstrom et al. [2019a] - l_∞	88.7	6.1	6.7	13.0	28.0	**4.9**
CIFAR-100 ($\epsilon = 12$)						
Rice et al. [2020] - l_2	58.7	13.4	13.8	16.4	23.8	**12.1**
Rice et al. [2020] - l_∞	54.5	3.8	4.2	7.7	10.6	**3.1**
ImageNet ($\epsilon = 60$)						
Engstrom et al. [2019a] - l_2	56.6	42.8	40.8	43.1	50.2	**40.5**
Engstrom et al. [2019a] - l_∞	61.9	5.7	5.5	18.9	23.9	**4.4**

5.2 RobustBench

Rank	Method	Standard accuracy	AutoAttack robust accuracy	Best known robust accuracy	AA eval. potentially unreliable	Extra data	Architecture	Venue
1	Fixing Data Augmentation to Improve Adversarial Robustness *66.56% robust accuracy is due to the original evaluation (AutoAttack + MultiTargeted)*	92.23%	66.58%	66.56%	×	☑	WideResNet-70-16	arXiv, Mar 2021
2	Uncovering the Limits of Adversarial Training against Norm-Bounded Adversarial Examples *65.87% robust accuracy is due to the original evaluation (AutoAttack + MultiTargeted)*	91.10%	65.88%	65.87%	×	☑	WideResNet-70-16	arXiv, Oct 2020
3	Fixing Data Augmentation to Improve Adversarial Robustness *It uses additional 1M synthetic images in training. 64.58% robust accuracy is due to the original evaluation (AutoAttack + MultiTargeted)*	88.50%	64.64%	64.58%	×	×	WideResNet-106-16	arXiv, Mar 2021

Figure 5.1: The top-3 entries of our CIFAR-10 leaderboard hosted at https://robustbench.github.io/ for the l_∞-perturbations of radius $\epsilon_\infty = 8/255$.

As mentioned above, benchmarking the progress of adversarial robustness might be challenging: there are already more than 6000 papers on this topic [Carlini, 2021], but it is often unclear which defenses against adversarial examples indeed improve robustness and which only make the typically used attacks overestimate the actual robustness. As shown in Sec. 5.1, AutoAttack may represent a suitable standardized tool to evaluate robustness, which can be complemented with adaptive attacks. However, recently Tramèr et al. [2020] observe that although several recently published defenses have tried to perform adaptive evaluations, many of them could still be broken by new adaptive attacks. We observe that there are repeating patterns in many of these defenses that prevent standard attacks from succeeding. This motivates us to impose restrictions on the defenses we consider in our proposed benchmark, RobustBench, which aims at *standardized* adversarial robustness evaluation. Specifically, we rule out i) classifiers which have zero gradients with respect to the input [Buckman et al., 2018, Guo et al., 2018], ii) randomized classifiers [Yang et al., 2019, Pang et al., 2020b], and iii) classifiers that use an optimization loop at inference time [Samangouei et al., 2018, Li et al., 2019c]. Often, non-certified defenses that violate these three restrictions only make gradient-based attacks harder but do not substantially improve robustness [Carlini et al., 2019a]. However, we will lift (some of) these constraints if a standardized reliable evaluation method for those defenses becomes available. We start from benchmarking robustness with respect to the l_∞- and l_2-threat models, since they are the most studied settings in the literature. We use AutoAttack as our current standard evaluation which is an ensemble of diverse parameter-free attacks (white- and black-box) that has shown reliable performance over a large set of models that satisfy our restrictions. Moreover, we accept and encourage external evaluations, e.g. with adaptive attacks, to improve our standardized evaluation, especially for the leaderboard entries whose evaluation may be unreliable according to the *flag* that we propose. Additionally, we collect models robust against common image corruptions [Hendrycks and Dietterich, 2019] as these represent another important type of perturbations which should not change the decision of a classifier.

Components of RobustBench. We make following key contributions with our RobustBench benchmark:

- **Leaderboard** (https://robustbench.github.io/): a website with the leaderboard (see Fig. 5.1) based on *more than 120* evaluations where it is possible to track the progress and the current state of the art in adversarial robustness based on a standardized evaluation using AutoAttack *complemented* by (external) adaptive evaluations. The goal is to clearly identify the most successful ideas in training robust models to accelerate the progress in the field.
- **Model Zoo** (https://github.com/RobustBench/robustbench): a collection of the most robust models that are easy to use for any downstream applications. As an example, we expect that this will foster the development of better adversarial attacks by making it easier to perform evaluations on a large set of *more than 80* models.

- **Analysis**: based on the collected models from the Model Zoo, we provide an analysis of how robustness affects the performance on distribution shifts, calibration, out-of-distribution detection, fairness, privacy leakage, smoothness, and transferability. In particular, we find that robust models are significantly *underconfident* that leads to worse calibration, and that not all robust models have higher privacy leakage than standard models.

Threat models. We focus on the fully white-box setting, i.e. the model f is assumed to be fully known to the attacker, and perturbations with bounded l_p-norm since those are the most widely studied ones. We rely on thresholds ϵ_p established in the literature which are chosen such that the true label should stay the same for each in-distribution input within the perturbation set. We note that robustness towards small l_p-perturbations is a necessary but not sufficient notion of robustness which has been criticized in the literature [Gilmer et al., 2018]. It is an active area of research to develop threat models which are more aligned with the human perception such as spatial perturbations [Fawzi and Frossard, 2015, Engstrom et al., 2019c], Wasserstein-bounded perturbations [Wong et al., 2019, Hu et al., 2020], perturbations of the image colors [Laidlaw and Feizi, 2019] or l_p-perturbations in the latent space of a neural network [Laidlaw et al., 2021, Wong and Kolter, 2021]. However, despite the simplicity of the l_p-perturbation model, it has numerous interesting applications that go beyond security considerations [Tramèr et al., 2019, Saadatpanah et al., 2020] and span transfer learning [Salman et al., 2020, Utrera et al., 2020], interpretability [Tsipras et al., 2019, Kaur et al., 2019, Engstrom et al., 2019b], generalization [Xie et al., 2020, Zhu et al., 2019, Bochkovskiy et al., 2020], robustness to unseen perturbations [Kang et al., 2019a, Xie et al., 2020, Laidlaw et al., 2021, Kireev et al., 2021], stabilization of GAN training [Zhong et al., 2020]. Thus, improvements in l_p-robustness have the potential to improve many of these downstream applications.

Additionally, we provide leaderboards for *common image corruptions* [Hendrycks and Dietterich, 2019] that try to mimic modifications of the input images which can occur naturally. Unlike l_p adversarial perturbations, they are not imperceptible and evaluation on them is done in the average-case fashion, i.e. there is no attacker who aims at changing the classifier's decision. In this case, the robustness of a model is evaluated as classification accuracy on the corrupted images, averaged over corruption types and severities.

5.2.1 Comparison to existing benchmarks

Related libraries and benchmarks. There are many libraries that focus primarily on implementations of popular adversarial attacks such as FoolBox [Rauber et al., 2017], Cleverhans [Papernot et al., 2018], AdverTorch [Ding et al., 2019], AdvBox [Goodman et al., 2020], ART [Nicolae et al., 2018], SecML [Melis et al., 2019], DeepRobust [Li et al., 2020]. Some of them also provide implementations of several basic defenses, but they do not include up-to-date state-of-the-art models. The two challenges [Kurakin et al., 2018, Brendel et al., 2018] hosted at NeurIPS 2017 and 2018 aimed at finding the most robust models for specific attacks, but they had a predefined deadline, so they could capture the best defenses only at the time of the competition. Ling et al. [2019] proposed DEEPSEC, a benchmark that tests many combinations of attacks and defenses, but suffers from a few shortcomings as suggested by Carlini [2019]: i) reporting average-case instead of worst-case performance over multiple attacks, ii) evaluating robustness in threat models different from the ones used for training, iii) using excessively large perturbations. Chen and Gu [2020] proposed a new *hard-label* black-box attack, RayS, and evaluated it on a range of models which led to a leaderboard.[2] Despite being a state-of-the-art hard-label black-box attack, the robust accuracy in the leaderboard given by RayS still tends to be overestimated even compared to the original evaluations.

Recently, Dong et al. [2020] have provided an evaluation of a few defenses (in particular, 3 for l_∞- and 2 for l_2-norm on CIFAR-10) against multiple commonly used attacks. However, they did not include some of the best performing defenses [Hendrycks et al., 2019, Carmon et al., 2019, Gowal et al., 2020, Rebuffi et al., 2021b] and attacks [Gowal et al., 2019b, Croce and Hein, 2020a], and in a few cases, their evaluation suggests robustness higher than what was reported in the original papers. Moreover, they do not impose any restrictions on the models they accept

[2] https://github.com/uclaml/RayS

to the benchmark. RobustML[3] aims at collecting robustness claims for defenses together with external evaluations. Their format does not assume running any baseline attack, so it relies entirely on evaluations submitted by the community, which however do not occur often enough. Thus even though RobustML has been a valuable contribution to the community, now it does not provide a comprehensive overview of the recent state of the art in adversarial robustness.

Finally, it has become common practice to test new attacks wrt l_∞ on the publicly available models from Madry et al. [2018] and Zhang et al. [2019b], since those represent widely accepted defenses which have stood many thorough evaluations. However, having only two models per dataset (MNIST and CIFAR-10) does not constitute a sufficiently large testbed, and, because of the repetitive evaluations, some attacks may already overfit to those defenses.

What is different in RobustBench. Learning from these previous attempts, RobustBench presents a few different features compared to the aforementioned benchmarks: i) a baseline worst-case evaluation with an ensemble of *strong*, *standardized* attacks [Croce and Hein, 2020c] which includes both white- and black-box attacks, unlike RobustML which is *solely* based on adaptive evaluations, integrated by external evaluations, ii) we add a *flag* in AutoAttack raised when the evaluation might be unreliable, in which case we do additional adaptive evaluations ourselves and encourage the community to contribute,iii) clearly defined threat models that correspond to the ones used during training of submitted models, iv) evaluation of not only standard defenses [Madry et al., 2018, Zhang et al., 2019b] but also of more recent improvements such as Carmon et al. [2019], Gowal et al. [2020], Rebuffi et al. [2021b]. Moreover, RobustBench is designed as an *open-ended* benchmark that keeps an up-to-date leaderboard, and we welcome contributions of new defenses and evaluations using adaptive attacks. Finally, we open source the Model Zoo for convenient access to the 80+ most robust models from the literature which can be used for downstream tasks and facilitate the development of new standardized attacks.

5.2.2 Leaderboard

Restrictions. We argue that accurate benchmarking adversarial robustness in a standardized way requires some restrictions on the type of considered models. The goal of these restrictions is to prevent submissions of defenses that cause some standard attacks to fail without truly improving robustness. Specifically, we consider only classifiers $f : \mathbb{R}^d \to \mathbb{R}^C$ that

- have in general *non-zero gradients* with respect to the inputs. Models with zero gradients, e.g., that rely on quantization of inputs [Buckman et al., 2018, Guo et al., 2018], make gradient-based methods ineffective thus requiring zeroth-order attacks, which do not perform as well as gradient-based attacks. Alternatively, specific adaptive evaluations, e.g. with Backward Pass Differentiable Approximation [Athalye et al., 2018], can be used which, however, can hardly be standardized. Moreover, we are not aware of existing defenses solely based on having zero gradients for large parts of the input space which would achieve competitive robustness.

- have a *fully deterministic forward pass*. To evaluate defenses with stochastic components, it is a common practice to combine standard gradient-based attacks with Expectation over Transformations [Athalye et al., 2018]. While often effective it might be not sufficient, as shown by Tramèr et al. [2020]. Moreover, the classification decision of randomized models may vary over different runs for the same input, hence even the definition of robust accuracy differs from that of deterministic networks. We note that randomization *can* be useful for improving robustness and deriving robustness certificates [Lecuyer et al., 2019, Cohen et al., 2019], but it also introduces variance in the gradient estimators (both white- and black-box) making standard attacks much less effective.

- do not have an *optimization loop* in the forward pass. This makes backpropagation through it very difficult or extremely expensive. Usually, such defenses [Samangouei et al., 2018, Li et al., 2019c] need to be evaluated adaptively with attacks that rely on a combination of hand-crafted losses.

[3] https://www.robust-ml.org/

Figure 5.2: Visualization of the robustness and accuracy of 54 CIFAR-10 models from the RobustBench l_∞-leaderboard. Robustness is evaluated using l_∞-perturbations with $\epsilon_\infty = {}^8/_{255}$.

Some of these restrictions were also discussed by Brown et al. [2018] for the warm-up phase of their challenge. We refer the reader to Appendix E therein for an illustrative example of a trivial defense that bypasses gradient-based and some of the black-box attacks they consider. We believe that such constraints are necessary at the moment since they allow an accurate standardized evaluation which makes the leaderboard meaningful and sustainable. In fact, for defenses not fulfilling the restrictions there is no standard evaluation which is shown to generalize and perform well across techniques, thus one has to resort to time-consuming adaptive attacks specifically tailored for each case. In the design of our benchmark, we thought that it is more important that the robustness evaluation is reliable, rather than being open to all possible defenses with the risk that the robustness is drastically overestimated. As this can lead to a potential bias in our leaderboard, we will lift the restrictions if reliable standardized evaluation methods for these modalities become available in the literature.

Overall setup. We set up leaderboards for the l_∞, l_2 and common corruption threat models on CIFAR-10, CIFAR-100, and ImageNet datasets (see Table 5.8 for details). We use the fixed budgets of $\epsilon_\infty = 8/255$ and $\epsilon_2 = 0.5$ for the l_∞ and l_2 leaderboards for CIFAR-10 and CIFAR-100. For ImageNet, we use $\epsilon_\infty = 4/255$ and in Sec. 5.2.5 we discuss how we handle that different models use different image resolutions for ImageNet. Most of the models shown in the leaderboards are taken from papers published at top-tier machine learning and computer vision conferences as shown in Fig. 5.2 (left). For each entry we report the reference to the original paper, standard and robust accuracy under the specific threat model (see the next paragraph for details), network architecture, venue where the paper appeared and possibly notes regarding the model. We also highlight when extra data (often, the dataset introduced by Carmon et al. [2019]) is used since it gives a clear advantage for both clean and robust accuracy. If any other attack achieves lower robust accuracy than AutoAttack then we also report it. Moreover, the leaderboard allows to search the entries by their metadata (such as title, architecture, venue) which can be useful to compare different methods that use the same architecture or to search for papers published at some conference.

Evaluation of defenses. The evaluation of robust accuracy on common corruptions [Hendrycks and Dietterich, 2019] involves simply computing the average accuracy on corrupted images over different corruption types and severity levels.[4] To evaluate robustness of l_∞ and l_2 defenses, we currently use AutoAttack (see Sec. 5.1). We choose AutoAttack as it includes both black-box and white-box attacks, does not require hyperparameter tuning (in particular, the step size), and consistently improves the results reported in the original papers for almost all the models (see Fig. 5.2 (middle)). If in the future some new standardized and parameter-free attack is shown to consistently outperform AutoAttack on a wide set of models given a similar computational cost, we will adopt it as standard evaluation. In order to verify the reproducibility of the results, we perform the standardized evaluation independently of the authors of the submitted models. We encourage evaluations of the models in the leaderboard with adaptive or external attacks to reflect the best available upper bound on the true robust accuracy (see a pre-formatted issue

[4] A breakdown over corruptions and severities is also available, e.g. for CIFAR-10 models see: https://github.com/RobustBench/robustbench/blob/master/model_info/cifar10/corruptions/unaggregated_results.csv

template in our repository[5]), in particular in the case where AutoAttack flags that it might not be reliable (see paragraph below). For example, Gowal et al. [2020] and Rebuffi et al. [2021b] evaluate their models with a hybrid of AutoAttack and MultiTargeted attack [Gowal et al., 2019b], that in some cases reports slightly lower robust accuracy than AutoAttack alone. We reflect the additional evaluations in our leaderboard by reporting in a separate column the robust accuracy for the worst case of AutoAttack and all other evaluations. Below we show an example of how one can use our library to easily benchmark a model (either external one or taken from the Model Zoo):

```
from robustbench.eval import benchmark
clean_acc, robust_acc = benchmark(model, dataset='cifar10', threat_model='Linf')
```

Moreover, in Sec. 5.2.5 we also show the variability of the robust accuracy given by AutoAttack over random seeds and report its runtime for a few models from different threat models.

Identifying potential need for adaptive attacks. Although AutoAttack provides an accurate estimation of robustness for most models that satisfy the restrictions mentioned above, there might still be corner cases when AutoAttack overestimates robustness of a model that satisfies the restrictions. Carlini et al. [2019a] suggest that one indicator of possible overestimation of robustness is when black-box attacks are more effective than white-box ones. We noticed that this is the case for the model from Xiao et al. [2020] where the black-box Square Attack improves by *more than* 10% the robust accuracy given by the previous white-box attacks in AutoAttack. We run a simple adaptive attack: Square Attack with multiple random restarts (30 instead of 1) decreases the robust accuracy from the 18.50% of AutoAttack to 7.40%. We note that AutoAttack did not fail completely for this model and correctly revealed a lower level of robustness than reported (52.4%), although the exact robust accuracy was overestimated. Based on this case, we integrate a *flag* in AutoAttack: a warning is output whenever Square Attack reduces of more than 0.2% the robust accuracy compared to the white-box gradient-based attacks in AutoAttack. In this case, AutoAttack evaluation might be not fully reliable and adaptive attacks might be necessary, so we flag the corresponding entries in the leaderboard (currently, only the model of Xiao et al. [2020]). Moreover, for the sake of convenience, we also integrate in AutoAttack flags that automatically inform the user if the restrictions are violated.[6]

Adding new defenses. We believe that the leaderboard is only useful if it reflects the latest advances in the field, so it needs to be constantly updated with new defenses. We intend to include evaluations of new techniques and we welcome contributions from the community which help to keep the benchmark up-to-date. We require new entries to i) satisfy the three restrictions stated above, ii) to be accompanied by a publicly available paper (e.g., an arXiv preprint) describing the technique used to achieve the reported results, and iii) share the model checkpoints (not necessarily publicly). We also allow *temporarily* adding entries without providing checkpoints given that the authors evaluate their models with AutoAttack. However, we will mark such evaluations as *unverified*, and to encourage reproducibility, we reserve the right to remove an entry later on if the corresponding model checkpoint is not provided. It is possible to add a new defense to the leaderboard and (optionally) the Model Zoo by opening an issue with a predefined template in our repository,[7] where more details about new additions can be found.

5.2.3 Model Zoo

We collect the checkpoints of many networks from the leaderboard in a single repository hosted at https://github.com/RobustBench/robustbench after obtaining the permission of the authors (see the website for the information on the licenses). The goal of this repository, the Model Zoo, is to make the usage of robust models as simple as possible to facilitate various downstream applications and analyses of general trends in the field. In fact, even when the checkpoints of the proposed method are made available by the authors, it is often time-consuming and not straightforward to integrate them in the same framework because of many factors such

[5] https://github.com/RobustBench/robustbench/issues/new/choose
[6] See https://github.com/fra31/auto-attack/blob/master/flags_doc.md for details
[7] https://github.com/RobustBench/robustbench

Table 5.8: The total number of models in the Model Zoo and leaderboards per dataset and threat model.

	CIFAR-10		CIFAR-100		ImageNet	
Threat model	Model Zoo	Leaderboard	Model Zoo	Leaderboard	Model Zoo	Leaderboard
l_∞	39	63	14	14	5	6
l_2	17	18	-	-	-	-
Common corruptions	7	15	2	4	5	7

as small variations in the architectures, custom input normalizations, etc. For simplicity of implementation, at the moment we include only models implemented in PyTorch [Paszke et al., 2019]. Below we illustrate how a model can be automatically downloaded and loaded via its identifier and threat model within two lines of code:

```
from robustbench.utils import load_model
model = load_model(model_name='Ding2020MMA', dataset='cifar10', threat_model='L2')
```

We include the most robust models, e.g. those from Rebuffi et al. [2021b], but there are also defenses which pursue additional goals alongside adversarial robustness at the fixed threshold we use: e.g., Sehwag et al. [2020] consider networks which are robust and compact, Wong et al. [2020] focus on computationally efficient adversarial training, Ding et al. [2020] aim at input-adaptive robustness as opposed to robustness within a single l_p-radius. All these factors have to be taken into account when comparing different techniques, as they have a strong influence on the final performance. Thus, we highlight these factors in the footnotes below each paper's title.

A testbed for new attacks. Another important use case of the Model Zoo is to simplify comparisons between different adversarial attacks on a wide range of models. First, the leaderboard already serves as a strong baseline for new attacks. Second, as mentioned above, new attacks are often evaluated on the models from Madry et al. [2018] and Zhang et al. [2019b], but this may not provide a representative picture of their effectiveness. For example, currently the difference in robust accuracy between the first and second-best attacks in the CIFAR-10 leaderboard of Madry et al. [2018] is only 0.03%, and between the second and third is 0.04%. Thus, we believe that a more thorough comparison should involve multiple models to prevent overfitting of the attack to one or two standard robust defenses.

5.2.4 Analysis

With unified access to multiple models from the Model Zoo, one can easily compute various performance metrics to see general trends. We analyze various aspects of robust classifiers for robust models on CIFAR-10.

Performance across various distribution shifts. We test the performance of the models from the Model Zoo on different distribution shifts ranging from common image corruptions (CIFAR-10-C) to dataset resampling bias (CIFAR-10.1 [Recht et al., 2018]) and image source shift (CINIC-10 [Darlow et al., 2018]). For each of these datasets, we measure standard accuracy, and Fig. 5.3 shows that improvement in robust accuracy (which often comes with an improvement in standard accuracy) on CIFAR-10 correlates with an improvement in standard accuracy across distributional shifts. On CIFAR-10-C, robust models (particularly with respect to the l_2-norm) tend to give a significant improvement which agrees with the findings in Ford et al. [2019]. Concurrently with our work, Taori et al. [2020] study the robustness to different distribution shifts of many models trained on ImageNet, including some l_p-robust models. Our conclusions qualitatively agree with theirs, and we hope that our collected set of models will help to provide a more complete picture. Moreover, we measure robust accuracy, in the same threat model used on CIFAR-10, using AutoAttack (see Fig. 5.4), in order to see how l_p adversarial robustness generalizes across the datasets representing different distributions shifts, and observe a clear positive correlation between robust accuracy on CIFAR-10 and its variations.

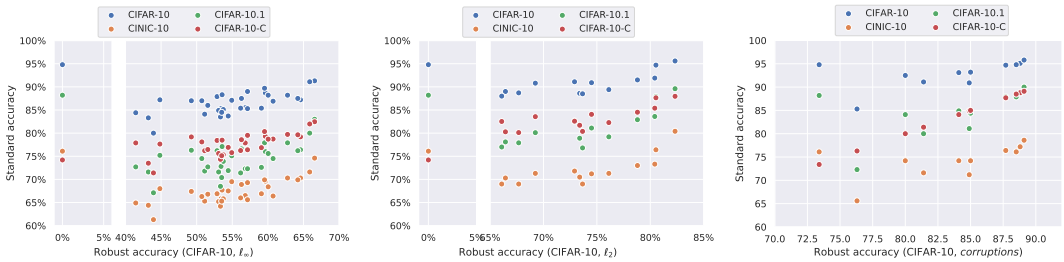

Figure 5.3: Standard accuracy of classifiers trained against l_∞ (**left**), l_2 (**middle**), and common corruption (**right**) threat model respectively, from our Model Zoo on various distribution shifts.

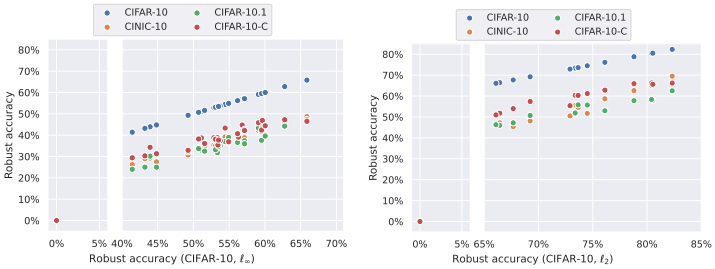

Figure 5.4: Robust accuracy of the robust classifiers, trained against l_∞ and l_2 threat model, respectively, from our Model Zoo on various distribution shifts. The data points with 0% robust accuracy correspond to a standardly trained model.

Calibration. A classifier is *calibrated* if its predicted probabilities correctly reflect the actual accuracy [Guo et al., 2017]. In the context of adversarial training, calibration was considered in Hendrycks et al. [2020] who focus on improving accuracy on common corruptions and in Augustin et al. [2020] who focus mostly on preventing overconfident predictions on out-of-distribution inputs. We instead focus on *in-distribution* calibration. We compute the expected calibration error (ECE) without and with temperature rescaling [Guo et al., 2019] to minimize the ECE (which is a simple but effective post-hoc calibration method) using the code of Guo et al. [2017]. We use their default settings to compute the calibration error which includes, in particular, binning of the probability range onto 15 equally-sized bins. However, we use our own implementation of the temperature rescaling algorithm which is close to that of Augustin et al. [2020]. Since optimization of the ECE over the softmax temperature is a simple *one-dimensional* optimization problem, we can solve it efficiently using a grid search. Moreover, the advantage of performing a grid search is that we can optimize directly the metric of interest, i.e. ECE, instead of the cross-entropy loss as in Guo et al. [2017] who had to rely on a differentiable loss since they used LBFGS [Liu and Nocedal, 1989] to optimize the temperature. We perform a grid search over the interval $t \in [0.001, 1.0]$ with a grid step 0.001 and we test both t and $1/t$ temperatures. Moreover, we check that for all models the optimal temperature t is situated not at the boundary of the grid.

In Fig. 5.5 we plot the expected calibration error (ECE) without and with temperature rescaling together with the optimal temperature for a large set of classifiers. We observe that most of the l_∞ (top row of Fig. 5.5) robust models are significantly *underconfident* since the optimal calibration temperature is less than one for most models. The only two models which are *overconfident* are the standard model and the model of Ding et al. [2020] that aims to maximize the margin. We see that temperature rescaling is even more important for robust models since without any rescaling the ECE is as high as 70% for the model of Pang et al. [2020c] (and 21% on average) compared to 4% for the standard model. Temperature rescaling significantly reduces the ECE gap between robust and standard models but it does not fix the problem completely which suggests that it is worth incorporating calibration techniques also during training of robust models.

We show additional calibration results for l_2-robust models in the bottom row of Fig. 5.5. The overall trend of the ECE is the same as for l_∞-robust models: most of the l_2 models are

underconfident (since the optimal temperature is less than one) and lead to worse calibration before and after temperature rescaling. The main difference compared to the l_∞ threat model is that among the l_2 models there are two models that are *better-calibrated*: one before (Engstrom et al. [2019a] with 1.41% ECE vs 3.71% ECE of the standard model) and one after (Gowal et al. [2020] with 1.00% ECE vs 1.11% ECE of the standard model) temperature rescaling. Moreover, we can see that similarly to the l_∞ case, the only overconfident models are either the standard one or models that maximize the margin instead of using norm-bounded perturbations, i.e. Ding et al. [2020] and Rony et al. [2019].

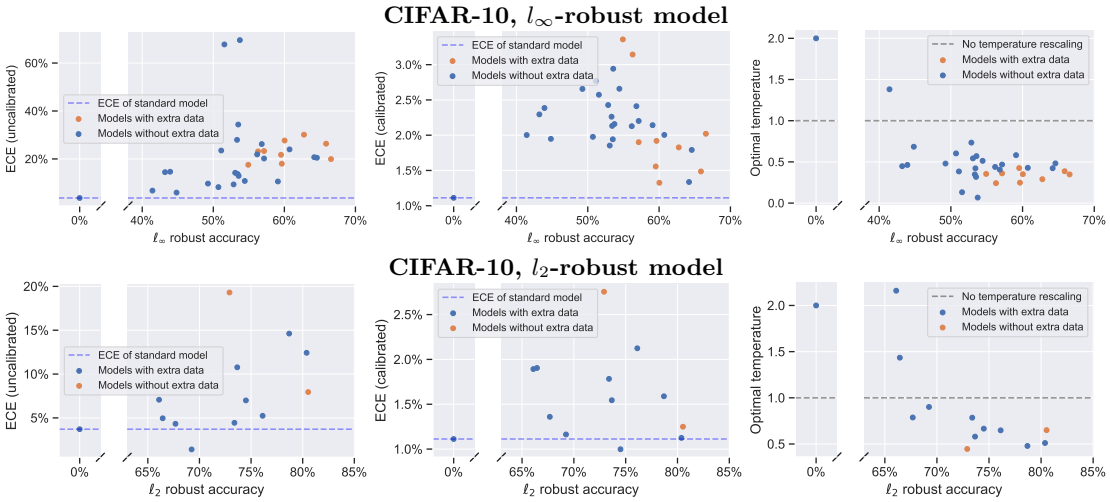

Figure 5.5: Expected calibration error (ECE) before (**left**) and after (**middle**) temperature rescaling, and the optimal rescaling temperature (**right**) for the l_∞-anf l_2-robust models.

Out-of-distribution detection. Ideally, a classifier should exhibit high uncertainty in its predictions when evaluated on *out-of-distribution* (OOD) inputs. One of the most straightforward ways to extract this uncertainty information is to use some threshold on the predicted confidence where OOD inputs are expected to have low confidence from the model [Hendrycks and Gimpel, 2017]. An emerging line of research aims at developing OOD detection methods in conjunction with adversarial robustness [Hein et al., 2019, Sehwag et al., 2019, Augustin et al., 2020]. In particular, Song et al. [2020] demonstrated that adversarial training leads to degradation in the robustness against OOD data. We further test this observation on all l_∞-models trained on CIFAR-10 from the Model Zoo on three OOD datasets: CIFAR-100, SVHN [Netzer et al., 2011], and Describable Textures Dataset [Cimpoi et al., 2014]. We use the area under the ROC curve (AUROC) to measure the success in the detection of OOD data, and show the results in Fig. 5.6. With l_∞ robust models, we find that compared to standard training, various robust training methods indeed lead to degradation of the OOD detection quality. While extra data in standard training can improve robustness against OOD inputs, it fails to provide similar improvements with robust training. We further find that l_2 robust models have in general comparable OOD detection performance to standard models, while the model of Augustin et al. [2020] achieves even better performance since their approach explicitly optimizes both robust accuracy and worst-case OOD detection performance.

Fairness in robustness. Recent works [Benz et al., 2021, Xu et al., 2020a] have noticed that robust training [Madry et al., 2018, Zhang et al., 2019b] can lead to models whose performance varies significantly across subgroups, e.g. defined by classes. We will refer to this performance difference as *fairness*, and here we study the influence of robust training methods on fairness. In Fig. 5.7 we show the breakdown of standard and robust accuracy for the robust models, where one can see how the achieved robustness largely varies over classes. While in general the classwise standard and robust accuracy correlate well, the class "deer" in l_∞-threat model suffers a significant degradation, unlike what happens for l_2, which might indicate that the features of such class are particularly sensitive to l_∞-bounded attacks. Moreover, we measure fairness with

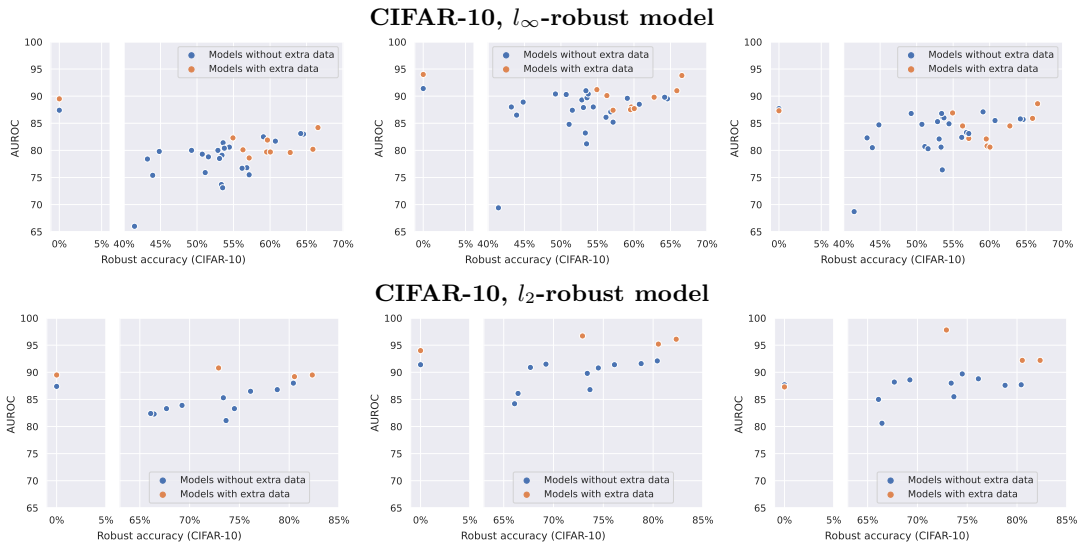

Figure 5.6: Visualization of the OOD detection quality (higher AUROC is better) for the l_∞-robust models trained on CIFAR-10 on three OOD datasets: CIFAR-100 (**left**), SVHN (**middle**), Describable Textures (**right**). We detect OOD inputs based on the maximum predicted confidence [Hendrycks and Gimpel, 2017].

the relative standard deviation (RSD), defined as the standard deviation divided over the average of robust accuracy over classes, for which lower values mean more uniform distribution and higher robustness. We observe that better robust accuracy generally leads to lower RSD values which implies that the disparity among classes is reduced (with a strong linear trend): improving the robustness of the models has then the effect of reducing the disparities among classes. However, some training techniques like MART [Wang et al., 2020] for l_∞ can noticeably increase the RSD and thus *increase the disparity* compared to other methods which achieve similar robustness (around 57%).

To compute the robustness for the experiments about fairness we used APGD on the targeted DLR loss (see Sec. 4.3) with 3 target classes and 20 iterations each on the whole test set. Note that even with this smaller budget we achieve results very close to that of the full evaluation, with an average difference smaller than 0.5%.

Privacy leakage. Deep neural networks are prone to memorizing training data [Shokri et al., 2017, Carlini et al., 2019b]. Recent work has highlighted that robust training exacerbates this problem [Song et al., 2019b]. We closely follow the methodology described in Song and Mittal [2021] to calculate inference accuracy. In particular, we measure the confidence in the correct class for each input image with a pre-trained classifier. We measure the confidence for both training and test set images and calculate the maximum classification accuracy between train and test images based on the confidence values. We refer to this accuracy as *inference accuracy using confidence*. Inference accuracy using confidence measures how accurately we can infer whether a sample was present in the training dataset. Additionally, we follow the recommendation from Song et al. [2019b] where they show that adversarial examples are more successful in estimating inference accuracy on robust networks. In our experiments, we also find that using adversarial examples leads to higher inference accuracy than benign images (Fig. 5.8). Moreover, robust networks in the l_2 threat model have relatively higher inference accuracy than robust networks in the l_∞ threat model.

A key reason behind privacy leakage through membership inference is that deep neural networks often end up overfitting on the training data. One standard metric to measure overfitting is the generalization gap between train and test set. Naturally, this difference in the accuracy on the train and test set is the baseline of inference accuracy. We refer to it as *inference accuracy using label* and report it in Fig. 5.9. Thus one might expect lower privacy leakage in robust networks as previous work explicitly aimed to reduce the generalization gap in robust training e.g. via early stopping [Rice et al., 2020, Zhang et al., 2019b, Gowal et al., 2020]. We consider both

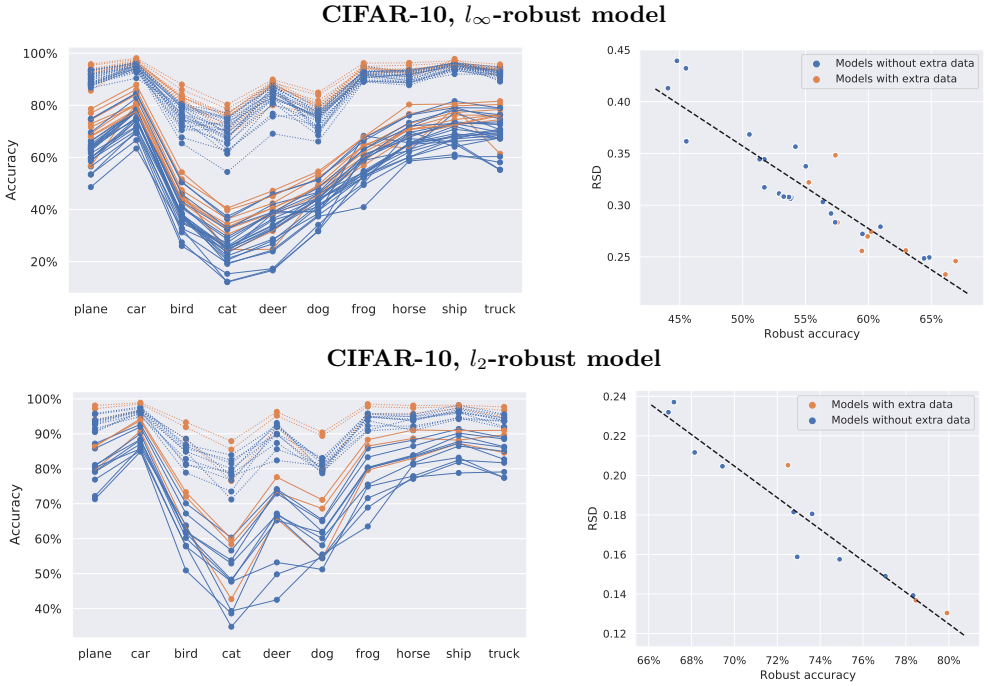

Figure 5.7: Fairness of l_∞-robust models. **Left:** classwise standard (dotted lines) and robust (solid) accuracy. **Right:** relative standard deviation (RSD) of robust accuracy over classes vs its average.

benign and adversarial images. When using benign images, we find confidence information does lead to higher inference accuracy than using only labels. However, with adversarial examples, which achieve higher inference accuracy than benign images, we find that inference accuracy based on confidence information closely follows the inference accuracy calculate from labels.

Smoothness. Previous work [Yang et al., 2020b] has shown that smoothness of a model, together with enough separation between the classes of the dataset for which it is trained, is necessary to achieve both natural and robust accuracy.Yang et al. [2020b] use local Lipschitzness as a measure for model smoothness, and observe empirically that robust models are more smoother than models trained in a standard way. Our Model Zoo enables us to check this fact empirically on a wider range of robust models, trained with a more diverse set of techniques, in particular with and without extra training data. Moreover, as we have access to the model internals, we can also compute local Lipschitzness of the model up to arbitrary layers, to see how smoothness changes between layers.

We compute local Lipschitzness using projected gradient descent (PGD) on the following optimization problem:

$$L = \frac{1}{N}\sum_{i=1}^{N} \max_{\substack{x_1:\|x_1-x_i\|_\infty\leq\varepsilon,\\ x_2:\|x_2-x_i\|_\infty\leq\varepsilon}} \frac{\|f(x_1)-f(x_2)\|_1}{\|x_1-x_2\|_\infty}, \tag{5.1}$$

where x_i represents each sample around which we compute local Lipschitzness, N is the number of samples across which we average ($N = 256$ in all our experiments), and f represents the function whose Lipschitz constant we compute. As mentioned above, this function can be either the full model, or the model up to an arbitrary intermediate layer. Since the models can have similar smoothness behavior, but at a different scale, we also consider normalizing the models outputs. One such normalization we use is given by the projection of the model outputs on the unit l_2-ball. This normalization aims at capturing the angular change of the output, instead of taking in consideration also its magnitude. We compute the "angular" version of the Lipschitz constant as

$$L = \frac{1}{N}\sum_{i=1}^{N} \max_{\substack{x_1:\|x_1-x_i\|_\infty\leq\varepsilon,\\ x_2:\|x_2-x_i\|_\infty\leq\varepsilon}} \frac{\left\| \frac{f(x_1)}{\|f(x_1)\|_2} - \frac{f(x_2)}{\|f(x_2)\|_2} \right\|_1}{\|x_1-x_2\|_\infty}. \tag{5.2}$$

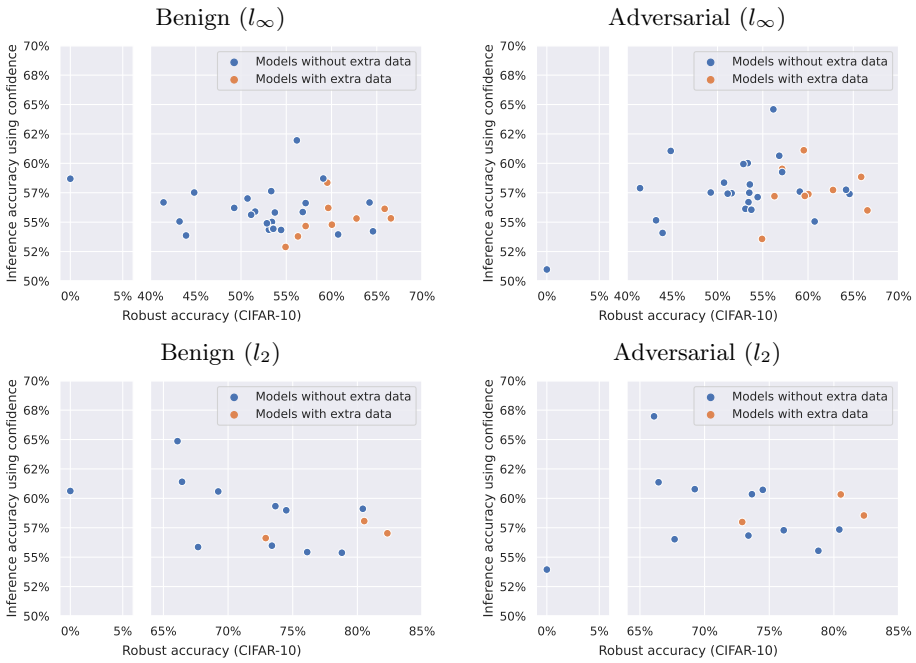

Figure 5.8: **Comparing privacy leakage of different networks.** We compare membership inference accuracy from benign and adversarial images across both l_∞ and l_2 threat model.

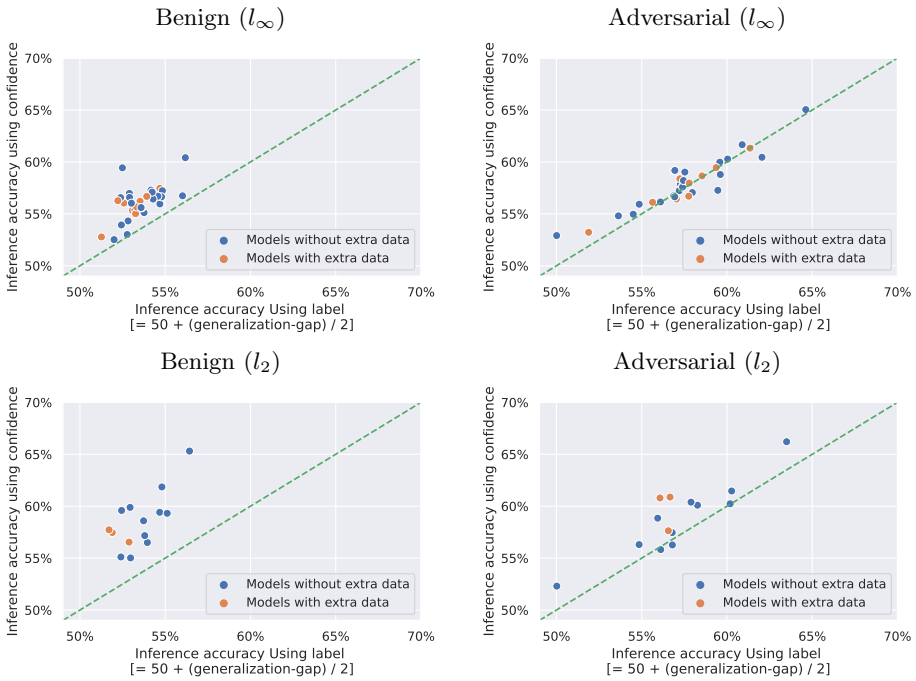

Figure 5.9: **Comparing privacy leakage with different output statistics.** We measure privacy leakage using membership inference accuracy, i.e., classification success between train and test set. We measure it using two baselines 1) based on correct prediction i.e., using predicted class label and 2) based on classification confidence in correct class. We also measure it using both benign and adversarial images.

For both variations of Lipschitzness, we compute it with $\epsilon = 8/255$, running the PDG-like procedure for 50 steps, with a step size of $\epsilon/5$.

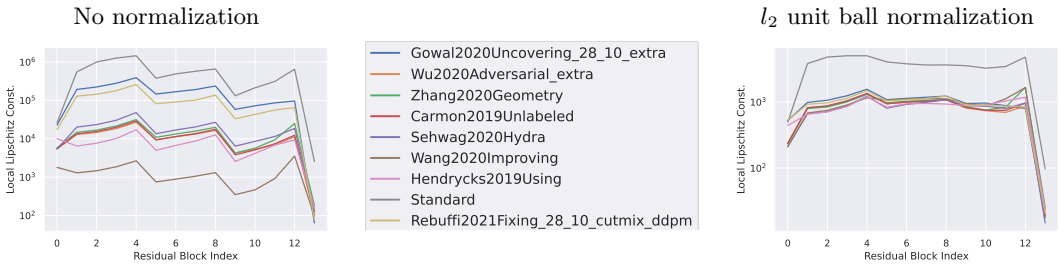

Figure 5.10: **Lipschitzness.** Computation of the local Lipschitz constant of the WRN-28-10 l_∞-robust models in our Model Zoo with $\epsilon = 8/255$. The color coding of the models is the same across both figures. For the correspondence between model IDs (shown in the legend) and papers that introduced them.

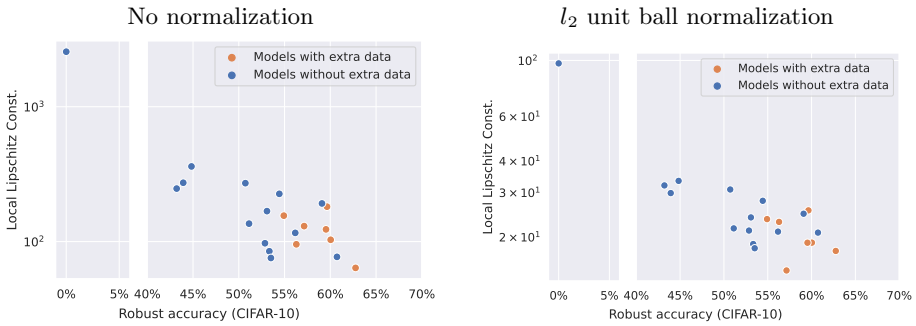

Figure 5.11: **Lipschitzness vs Robustness.** Local Lipschitz constant of the output layer vs. robust accuracy of a subset of the l_∞-robust models in our Model Zoo.

In Fig. 5.10 we compute the layerwise Lipschitzness for all l_∞ models trained on CIFAR-10 from the Model Zoo that have the WideResNet-28-10 architecture. We observe that the standard model is the least smooth at all the layers, and that all the robustly trained models are smoother. Moreover, we can notice that in the left plot of Fig. 5.10 there are two models in the middle ground: these are the models by Gowal et al. [2020] and Rebuffi et al. [2021b], which are the most robust ones, up to the last layer, the smoothest. Nonetheless, in the middle layers, they are the second and third least smooth, according to the unnormalized local Lipschitzness. This can be due to the different activation function used in these models (Swish vs ReLU). For this reason, we also compute "angular" Lipschitzness according to Eq. (5.2). Indeed, in the right plot of Fig. 5.10, all the robust models are in the same order of magnitude at all layers.

Finally, we also show the Lipschitz constants of the output layer for a larger set of l_∞ models from the Model Zoo that are not restricted to the same architecture. We plot the Lipschitz constant vs. the robust accuracy for these models in Fig 5.11. We see that there is a clear relationship between robust accuracy and Lipschitzness, hence confirming the findings of Yang et al. [2020b].

Transferability. We generate adversarial examples for a network, referred to as source network, and measure robust accuracy of every other network, referred to as target network, from the model zoo on them. We also include additional non-robust models[8], to name a few, VGG-19, ResNet-18, and DenseNet-121, in our analysis. We use 10 step PGD attack to generate adversarial examples, with the cross-entropy loss and step size $\epsilon/4$. We present our results in Fig. 5.12 and Fig. 5.13 where the correspondence between model IDs and papers that introduced them can be found in the Model Zoo website. We find that transferability to each robust target network follows a similar trend where adversarial examples transfer equally well from another robust networks. Though slight worse than robust network, adversarial points from non-robust

[8]We train then for 200 epochs and achieve 93-95% clean accuracy for all networks on the CIFAR-10 dataset.

networks also transfer equally well to robust networks. We observe a strong transferability among non-robust networks with adversarial examples generated from PGD attacks. Intriguingly, we observe the weakest transferability from a robust to a non-robust network. This observation holds for all robust source networks in both l_∞ and l_2 threat model.

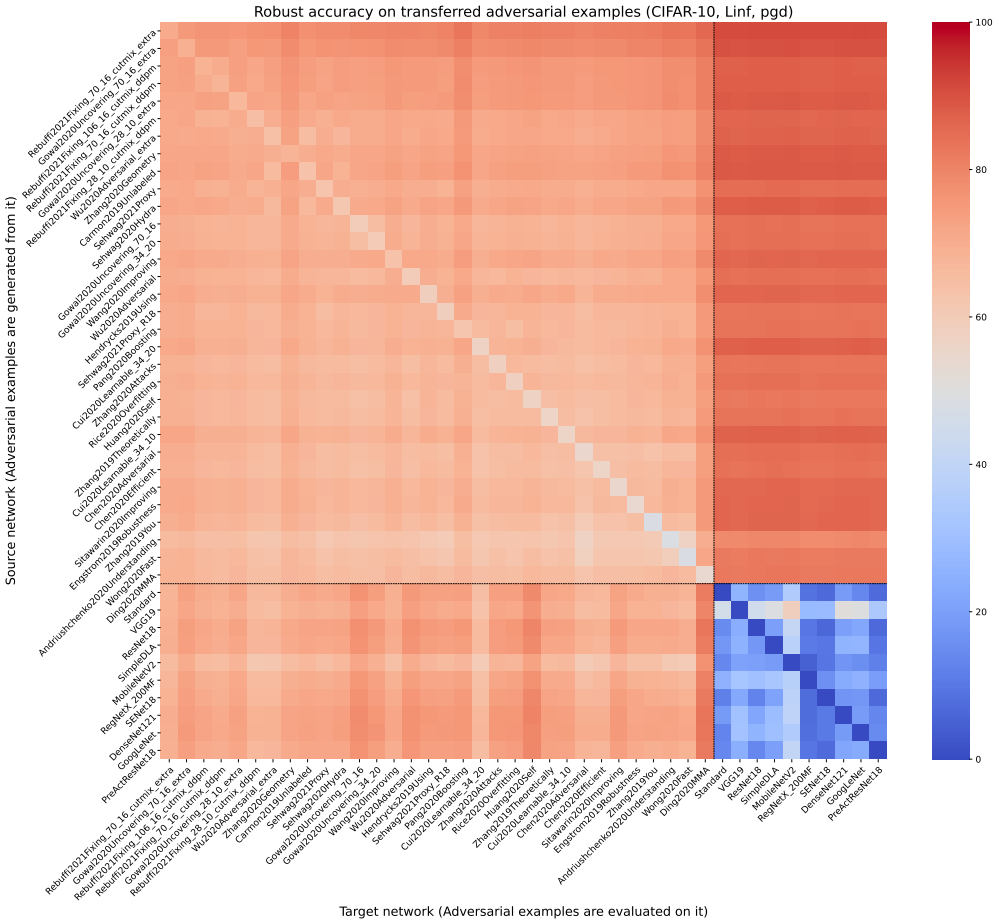

Figure 5.12: Measuring transferability of adversarial examples (l_∞, $\epsilon = 8/255$). Lower robust accuracy implies better transferability.

5.2.5 Additional details

Details of the ImageNet leaderboard. Extending the benchmark to ImageNet presents some challenges compared to CIFAR-10 and CIFAR-100. First, the ImageNet validation set (usually used as the test set) contains 50'000 images which makes it infeasible to run expensive evaluations on it. Thus, we define a fixed subset (5'000 randomly sampled images in our case) for faster evaluation, whose image IDs we make available in the Model Zoo. Second, it is not obvious how to handle the fact that different models may use different preprocessing techniques (e.g., different resolution, cropping, etc) which makes the search space for an attack *not fully comparable* across defenses. For this, we decide to allow models with different preprocessing steps and input resolution, considering them yet another design choice similarly to the choice of the network architecture which also has a large influence on the final results. Since in the l_∞-threat model the constraints are componentwise independent, we use the same threshold $\epsilon_\infty = 4/255$ for every classifier, regardless of the input dimensionality which is used after preprocessing. Note that we use the same set of images for l_p-robustness and for common corruptions, in which case for every point 15 types of corruptions at 5 severity levels are applied, consistently with the other datasets.

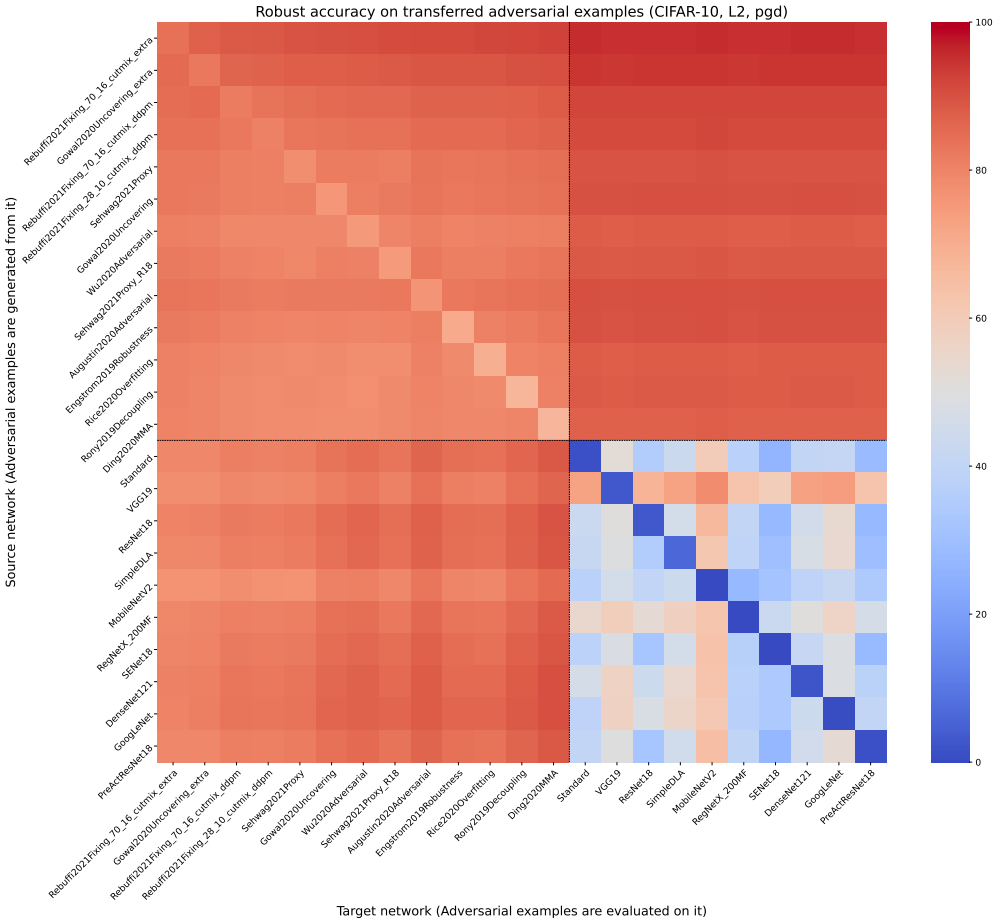

Figure 5.13: Measuring transferability of adversarial examples (l_2, $\epsilon = 0.5$). Lower robust accuracy implies better transferability.

Reproducibility and runtime. Here we discuss the main aspects of the reproducibility of the benchmark. First of all, the code to run the benchmark on a given model is available in our repository, and an example of how to run it is given in the README file. The installation instructions are also provided in the README file and the requirements will be installed automatically. To satisfy other points from the reproducibility checklist[9] which are applicable to our benchmark, we also discuss next the variability of the robust accuracy over random seeds and the average runtime of the benchmark. Evaluation of the accuracy on common corruptions is deterministic if we do not take into account non-deterministic operations on computational accelerators such as GPUs[10] which, however, do not affect the resulting accuracy. On the other hand, robustness evaluation using AutoAttack has an element of randomness since it relies on random initialization of the starting points and also on the randomness in the update of the Square Attack. To show the effect of randomness on the robust accuracy given by AutoAttack, we repeat evaluation over four random seeds on four models available in the Model Zoo from different threat models covering all datasets considered. In Table 5.9, we report the average robust accuracy with its standard deviation and observe that different seeds lead to very similar results. Moreover, we indicate the runtime of each evaluation, which is largely influenced by the size of the model, the computing infrastructure (every run uses a single Tesla V100 GPU), and the dataset. Moreover, less robust models require less time for evaluation which is due to the fact that AutoAttack does not further attack a point if an adversarial example is already found by some preceding attack in the ensemble.

Finally, when we extended the benchmark to ImageNet, we noticed that different versions of PyTorch and `torchvision` may lead to small differences in the standard accuracy (up to

[9] https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf
[10] https://pytorch.org/docs/stable/notes/randomness.html

0.16% on 5000 points for the same model). We suspect this is due to minor variations in the implementation of the preprocessing functions (such as resizing). Thus, we fix in the requirements `torch==1.7.1` and `torchvision==0.8.2` to ensure reproducibility. Note that the overall *ranking* and level of robustness of the defenses should not be influenced by using different versions of these libraries. We have not noticed similar issues for the other datasets.

Table 5.9: Statistics about the standardized evaluation with AutoAttack when repeated for four random seeds. We can see that the robust accuracy has very small fluctuations. We also report the runtime for the different models which is much smaller for less robust models.

dataset	norm	paper	architecture	clean acc.	robust acc.	time
CIFAR-10	l_∞	Gowal et al. [2020]	WRN-28-10	89.48%	62.82% ± 0.016	11.8 h
CIFAR-10	l_2	Rebuffi et al. [2021b]	WRN-28-10	91.79%	78.80% ± 0.000	15.1 h
CIFAR-100	l_∞	Wu et al. [2020a]	WRN-34-10	60.38%	28.84% ± 0.018	6.6 h
ImageNet	l_∞	Salman et al. [2020]	ResNet-18	52.92%	25.31% ± 0.010	1.6 h

5.2.6 Recent developments

After its publication, RobustBench has kept growing, and at the time of writing this thesis it includes more than 200 entries in the leaderboards and over 150 classifiers in the Model Zoo. Moreover, we have expanded the evaluation of robustness on ImageNet with the recently proposed 3D common corruptions dataset [Kar et al., 2022].

5.3 Evaluation of adaptive test-time defenses

Standardizing evaluation allows for the systematic tracking of real progress on adversarial robustness. However, to guarantee that standardized evaluations are accurate, defenses must adhere to some practical restrictions. In fact, RobustBench (Sec. 5.2) rules out i) models which have zero gradients with respect to the input [Buckman et al., 2018, Guo et al., 2018], ii) randomized models [Yang et al., 2019, Pang et al., 2020b], and iii) models that optimize during inference [Samangouei et al., 2018, Li et al., 2019c, Schott et al., 2019]. These restrictions may unnecessarily limit the development of robust models and, as a consequence, several researchers have offered general recommendations on how to evaluate adversarial defenses [Carlini et al., 2019a]. However, the evaluation of each new defense raises non-trivial choices.

In this section, we focus on *adaptive test-time defenses* that apply iterative optimization during inference, which promise to circumvent limitations of *static* defenses. In the context of adversarial robustness, the optimization is designed to "purify" inputs before feeding them to a static model or to adapt the model itself (we elaborate on categorizations of adaptive test-time defenses below). We restrict our analysis to image classification because it is the most common test-bed for studying robustness against l_p-norm bounded attacks.

We foresee adaptive defenses as an important step towards building robust defenses. However, given their novelty, the evaluation of such defenses has not been standardized. Indeed, test-time optimization can prevent the proper operation of standardized attacks established for static defenses, such as AutoAttack. This optimization also renders approximations such as the Backward Pass Differentiable Approximation (BPDA) [Athalye et al., 2018] more difficult to apply. As a result, recent works make strong robustness claims that turn out to be void or significantly weaker than claimed. Then,

- We categorize adaptive test-time defenses and explain their potential benefits and drawbacks;
- We evaluate 9 recent defenses (see Table 5.10), and show that they significantly overestimate their robustness. Relative to static defenses, most provide little improvement, and some are even detrimental. Furthermore all require more inference computation;
- Finally, we provide recommendations for evaluating such defenses and explain under which circumstances standard evaluations, like AutoAttack, are accurate.

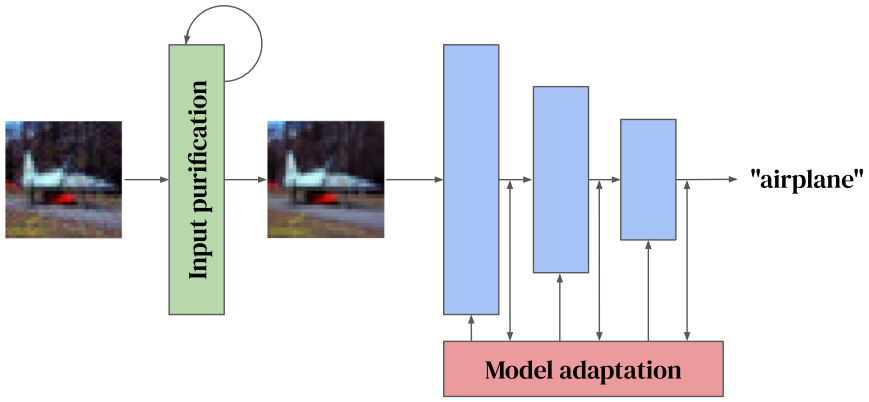

Figure 5.14: Adaptive test-time defenses operate by adapting their computation to the input. A first category of adaptive test-time defenses operate in the input space and aim to "purify" inputs before they are fed to the standard pre-trained model. A second category aims at adapting model parameters or intermediate activations.

In the following we provide a taxonomy of adaptive test-time defenses. We highlight their operating principles in Sec. 5.3.1, identify their building blocks in Sec. 5.3.2, and elaborate on potential pitfalls in Sec. 5.3.3. We also clearly delineate the threat model and adversarial capabilities that we assume during our case study of various techniques in Sec. 5.3.4.

5.3.1 Principles

Static defenses (e.g., standard deep networks) use a trained model for which the inputs and parameters are fixed at inference. In contrast, adaptive test-time defenses can alter the input or parameters for test inputs, either by optimizing an auxiliary loss or conditioning directly on the input. We identify two principles (highlighted in Fig. 5.14).

Input purification (IP): A model (possibly pre-trained for robustness) is augmented with test-time optimization to alter its inputs before the model is applied. The input optimization procedure may rely on hand-crafted [Alfarra et al., 2022, Wu et al., 2021b] or learned objectives [Qian et al., 2021, Mao et al., 2021, Hwang et al., 2021] which may involve auxiliary networks such as generative models [Song et al., 2018, Samangouei et al., 2018, Yoon et al., 2021, Nie et al., 2022].

Model adaptation (MA): A model is made adaptive by applying an optimization procedure to alter its parameters or state (e.g., batch normalization statistics) [Wang et al., 2021a] or activations [Chen et al., 2021] during inference. Other iterative schemes, such as implicit layers [Kang et al., 2021], define their inference as an optimization process.

5.3.2 Building blocks

To carry out input purification or model adaptation, adaptive test-time defenses can rely on distinct building blocks. We identify common blocks that span current defenses.

Iterative algorithm (IA): All defenses in our case study use an iterative algorithm. Five in nine solve their test-time optimization approximately by gradient descent, with some defenses using a single normalized gradient step [Qian et al., 2021] and others using as many as 40 gradient descent steps [Mao et al., 2021]. Other defenses may also use layers defined via implicit functions whose output is computed with iterative algorithms (e.g., neural ODE [Kang et al., 2021]) or may rely on iterative generative models (e.g., energy-based models [Yoon et al., 2021]).

Auxiliary networks (AN): A significant proportion of adaptive test-time defenses (seven in nine defenses studied) rely on an external network besides the underlying static model. This network typically helps to compute the optimization objective (such as a self-supervised objective [Shi et al., 2020]) or directly manipulates inputs (using a generative model for example [Yoon et al., 2021]). It might be bound to [Mao et al., 2021, Hwang et al., 2021] and possibly trained jointly with [Qian et al., 2021, Shi et al., 2020, Chen et al., 2021] the static model, or used as a standalone component [Yoon et al., 2021].

Randomness (R): Three in nine defenses studied are randomized either explicitly (e.g., by adding noise to the input [Wu et al., 2021b]) or implicitly (e.g., by sampling different batches of inputs [Mao et al., 2021]). As such, the same inputs may yield different outputs. We exclude pure randomization defenses (without test-time optimization).

External data (ED): One of the defenses studied exploits a collection of additional images, which are combined with the inputs. As such, inference does not depend only on the given input for classification [Mao et al., 2021].

5.3.3 Advantages, drawbacks and pitfalls

Test-time adaptation by optimization has shown promising results for robustness to natural shifts like common corruptions [Sun et al., 2020, Wang et al., 2021b], and has the potential to likewise improve adversarial robustness. Unlike static defenses, adaptive defenses have the freedom to alter an input (almost) arbitrarily or even update their own parameters on the input. This additional complexity seems to offer an advantage over static defenses that rely on adversarial training but remain fixed during testing. However, complex defenses are often difficult to evaluate [Tramèr et al., 2020], and adaptation might require techniques than those needed for static defenses. The accurate evaluation of adaptive test-time defenses presents several challenges:

Obfuscated gradients: First and foremost, iterative optimizers might be not differentiable (e.g., when using projections). BPDA, which is typically used in such circumstances and usually implemented as an identity function for the backward pass, can be weaker for procedures consisting of many iterations. Even in cases where the optimization is differentiable, gradients might vanish or explode when a large number of iterations is used.

Randomness: The use of randomized elements requires to resort to methods like Expectation over Transformation (EoT) [Athalye et al., 2018] and make the task of the attacker more expensive.

Runtime: The cost of inference is significantly higher than for static models, up to hundreds of times, which clearly impacts the amount of effort required for an accurate evaluation. Such high computational complexity calls into question the applicability of these defenses in the first place. Regardless of running attacks for evaluation, classification in deployment can be extremely slow. Hence, the trade-off in potential improvement versus additional computation should be the subject of further research.

5.3.4 Threat model and adversarial capabilities

We focus exclusively on the robustness of image classifiers to l_p-norm bounded input perturbations. Given a K-way classifier $f : \mathbb{R}^d \to \mathbb{R}^K$, a test input $\boldsymbol{x} \in [0,1]^d$ with label y and a norm bound $\epsilon > 0$, an attack succeeds if it finds a perturbation $\boldsymbol{\delta}$ such that

$$\underset{k \in \{1,\dots,K\}}{\arg\max}\, [f(\boldsymbol{x}+\boldsymbol{\delta})]_k \neq y, \quad \text{with } \|\boldsymbol{\delta}\|_p \leq \epsilon \text{ and } \boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^d. \tag{5.3}$$

where $[a]_i$ represents the i-th coordinate of a. We assume that the attacker has full white-box access to the defense. In other words, the attacker is aware of the test-time optimization in place, and has access to the parameters of the classifier and any auxiliary model(s). For randomized defenses, the attacker does not have access to the state of the random number generator. In some cases, we will fix the random seed to evaluate whether the impact of a proposed defense is mostly driven by optimization or randomization. Finally, when the defense uses external data, the attacker may have access to the set of images used (when fixed and hard-coded by the defense) or may try to infer the distribution from which these external images are drawn. For defenses that operate on a full batch of images, the attacker may be able to influence either one image from the batch or the whole batch. For all defenses, we respect the threat model defined by each defense when clearly defined in the corresponding paper or code.

5.3.5 Case study of adaptive methods

In our case study, we evaluate nine adaptive test-time defenses which rely on the adaptation principles elaborated above. Table 5.10 summarizes the defenses considered in this case study,

Table 5.10: Summary of the nine *adaptive test-time defenses* evaluated in our case study. We categorize each defense by its principles—input purification (IP) and model adaptation (MA)—and its building blocks: iterative algorithm (IA), auxiliary network (AN), randomization (R), and external data (ED). We measure the robust accuracy of each defense (and in parenthesis that of its underlying static model) against l_∞-norm bounded perturbations of size $\epsilon = 8/255$ on CIFAR-10 ("Ours") along with the robust accuracy measured by the respective papers ("Reported"). * This evaluation uses $\epsilon = 2/255$. ** This evaluation uses batch size 50 instead of the original 512.

Defense	Venue	Principles		Building blocks				Infer. time	Evaluation method	Robust Accuracy	
		IP	MA	IA	AN	R	ED			Reported	Ours
Kang et al. [2021]	NeurIPS		•	•	•			2×	Transfer APGD	57.76%	52.2% (53.9%)
Chen et al. [2021]*	ICLR		•	•	•			59×	APGD+BPDA	34.5%	5.6% (0.0%)
Wu et al. [2021b]	ArXiv	•		•		•		46×	Transfer APGD+BPDA+EoT	65.70%	61.0% (63.0%)
Alfarra et al. [2022]	AAAI	•		•				8×	RayS (decision-based)	79.2%	66.6% (66.6%)
Shi et al. [2020]	ICLR	•		•	•			518×	APGD+BPDA (traj.)	51.02%	3.7% (0.0%)
Qian et al. [2021]	ArXiv	•		•	•			4×	APGD	65.07%	12.6% (7.7%)
Hwang et al. [2021]	ICML(W)	•		•	•			40×	APGD+BPDA	52.65%	43.8% (49.3%)
Mao et al. [2021]**	ICCV	•		•	•	•	•	407×	APGD+BPDA+EoT	63.83%	58.4% (59.4%)
Yoon et al. [2021]	ICML	•		•	•	•		176×	APGD+EoT	69.71%	33.7% (0.0%)

categorizes each defense (according to Sec. 5.3.1 and Sec. 5.3.2), and details the corresponding results against l_∞-norm bounded attacks with budget $\epsilon = 8/255$ on CIFAR-10 (which is commonly evaluated by all defenses in their respective papers).[11] In particular, we report the robust accuracy (i.e., the classification accuracy on adversarially-perturbed inputs) originally reported in each work and the result of our evaluation. In parentheses, we report the robust accuracy of the underlying static model. We also measure the cost of inference computation in relation to performing the same inference with the underlying static model. Note that many factors can influence this value (e.g., architecture of the classifier, compute infrastructure, optimized implementation) and we use the runtime observed in our evaluation.

Overall, we find that the reported accuracy is consistently overestimated in all papers. Five defenses [Yoon et al., 2021, Hwang et al., 2021, Qian et al., 2021, Shi et al., 2020, Chen et al., 2021] have robust accuracies well below 50% and are not competitive with state-of-the-art static models (even of moderate size; Gowal et al., 2021, Rade and Moosavi-Dezfooli, 2021). Most importantly, four defenses [Wu et al., 2021b, Kang et al., 2021, Hwang et al., 2021, Mao et al., 2021] weaken the underlying static defense (when it is already robust), while the others provide minor improvements at major computational cost.

In contemporary work, Chen et al. [2022] carry out a complementary analysis of transductive test-time defenses [Wu et al., 2020b, Wang et al., 2021a], which depend on multiple test inputs and even joint optimization across training and testing data.

5.3.6 Evaluation methods

For completeness, we give a short overview of the attacks and techniques used in our case study. More details are described in the evaluation of each defense. **AutoAttack** (Sec. 5.1 is a common benchmark for evaluating static defenses, but it is not designed for adaptive test-time defenses. **AutoPGD (APGD)** (Sec. 4.3 is a variant of Projected Gradient Descent (PGD), one of the most popular technique for l_p-norm bounded adversarial attacks. Its varied surrogate losses include cross-entropy, Carlini-Wagner (CW), margin [Carlini and Wagner, 2017a], or the (targeted) Difference of Logits Ratio (DLR). **RayS** [Chen and Gu, 2020] is a decision-based attack designed for l_∞-norm bounded perturbations. It only requires the label predicted by the model. **BPDA** [Athalye et al., 2018] permits the attack of non-differentiable defenses by approximating them with differentiable functions during gradient computation. The identity is a common approximation. **EoT** [Athalye et al., 2018] permits the attack of randomized defenses. The predictions and gradients are computed in expectation over the randomness of the model, approximated by averaging the results of multiple runs with the same input. **Transfer attacks** generate adversarial perturbations on a surrogate model and use them on the target model.

[11] With the exception of $\epsilon = 2/255$ for Chen et al. [2021].

Table 5.11: Robust accuracy on 1000 points against l_p-norm bounded attacks: TRADES is tested with AutoAttack, and SODEF with transferring APGD from the static TRADES model.

Threat model	Static	with SODEF	
	AA	AA (transf.)	APGD (transf.)
l_∞	53.9%	60.9%	52.2%
l_2	59.1%	66.8%	58.2%

5.3.7 Stable Neural ODE with Lyapunov-Stable Equilibrium Points for Defending against Adversarial Attacks [Kang et al., 2021]

Summary of method. This defense applies a neural Ordinary Differential Equation (ODE) layer to improve robustness against l_p-norm bounded attacks by the stability of equilibrium points of the underlying ODE. The proposed SODEF model consists of a feature extractor followed by a neural ODE layer and finally one or multiple fully connected layers. The feature extractor is a standard convolutional network, while the neural ODE can be solved with numerical methods (Runge-Kutta of order 5 in this case) up to some integration time T.

Evaluation. While Kang et al. consider many scenarios in their experimental evaluation, we test SODEF when the feature extractor consists of a robust classifier, trained with TRADES [Zhang et al., 2019b] against l_∞-norm perturbations of size $8/255$ on CIFAR-10, without the last linear layer (since checkpoints are available online). We use the original implementation with corresponding parameters, including integration time $T = 5$. A significant improvement in robustness against l_∞- and l_2-norm perturbations is reported (57.76% and 67.75%) when compared to the original TRADES model (53.69% and 59.42%), under the evaluation of AutoAttack. In particular, AutoAttack is used both directly on the SODEF model and as the basis for a transfer attack from the TRADES classifier. However, we note that by default AutoAttack returns the original images when they are originally misclassified or when no adversarial perturbation is found. This means that, in practice, most of the transferred points are clean images (since the TRADES model is highly robust). Moreover, AutoAttack does not aim at maximizing the confidence in the misclassification after this is achieved and, as such, it might be not the strongest method for transfer attacks. As a consequence, we use APGD to maximize different target losses on all clean inputs using the TRADES classifier and then test them on SODEF. We run APGD for 100 iterations with cross-entropy, Carlini-Wagner (CW) and targeted DLR (with 3 target classes) loss, and consider the transfer attack successful if any of the resulting points fools the SODEF model. Table 5.11 shows the results of the described experiments for l_∞- and l_2-norm bounded attacks (on 1000 test images with $\epsilon = 8/255$ and $\epsilon_2 = 0.5$, respectively). When transferring inputs that maximize the surrogate losses, the robustness of SODEF is lower than that of the original TRADES classifiers.

Additional evaluation details. We use the original implementation of the defense and the corresponding checkpoints which are publicly available.[12] In the previous experiments, we use a WideResNet-34-10 as the underlying architecture for the static model. As a point of comparison, we also evaluate SODEF when combined with a larger WideResNet-70-16. The base model from Rebuffi et al. [2021b] achieves a robust accuracy of 66.56%. According to Kang et al. [2021], this model combined with SODEF achieves 71.28% when evaluated with AutoAttack. Similarly to what is described above, we use APGD to maximize different loss functions (cross-entropy, margin and targeted DLR with 9 restarts) on the base model, and transfer the obtained perturbations to the SODEF model. This yields a robust accuracy of 65.02%, which can be further decreased to 64.20% by taking the worst-case between the transfer attack and AutoAttack.

[12] https://github.com/kangqiyu/sodef

5.3.8 Towards Robust Neural Networks via Close-loop Control [Chen et al., 2021]

Summary of method. Using the assumption that the representation of an adversarial input becomes progressively more corrupted as it moves through the layers of a deep network, this defense corrects its trajectory by adding offsets to intermediate activations of a pre-trained static model. At each layer i, *control parameters* $\boldsymbol{u}_i$ are added to its output. These control parameters are optimized to reduce the reconstruction error between the original activations and corresponding "reduced" activations. These reduced activations are encoded onto a lower-dimensional manifold and decoded back to the full state space using Principal Component Analysis (PCA) or shallow auto-encoders. At inference time, the parameters $\boldsymbol{u}_i$ are jointly optimized with multiple steps of gradient descent and then used to produce the final prediction.

Evaluation. Chen et al. report 11% robust accuracy against PGD attacks of size $\epsilon = 8/255$. This is not competitive with state-of-art robust classifiers and already hints that the defense may be ineffective. As such, we focus on their strongest result, at the smaller $\epsilon = 2/255$, where they report 50% robust accuracy. Even this reduced claim is voided by further evaluation. Crucially, their attack is not adaptive, as it does not exploit knowledge of the defense. Chen et al. argue that adaptive attacks are infeasible, as they would need to steer the control parameters $\boldsymbol{u}_i$ toward zero. They also argue that calculating the gradients of the optimization process is too difficult. Neither argument is sound, as the defense might still fail for nonzero values of $\boldsymbol{u}_i$, and obfuscated gradients can be approximated. We demonstrate this by combining PGD and BPDA.

As pre-trained models are not publicly available, we train our own ResNet-20 model, and learn the linear embeddings with PCA using the code available. Unfortunately, some hyperparameters are not documented, and so we find a working configuration (using 5 iterations and a learning rate of $5 \cdot 10^{-3}$) by grid search that approximately reproduces the results of the paper. The resulting defense obtains 89.1% accuracy on clean images. We also reproduce their 20-step PGD evaluation by attacking the underlying classifier and testing the adversarial examples against their defense, and measure a robust accuracy of 34.5% (compared with 0% for the static model). The robustness gain is slightly less than reported (i.e., 50%) but still enough for our demonstration.[13] Each prediction takes 4 backward passes and 5 forward passes.

We now mount an adaptive attack. At test-time, the defense runs multiple iterations of gradient descent to optimize the control parameters $\boldsymbol{u}_i$. The optimized $\boldsymbol{u}_i^*$ are then used for the final prediction. Since $\boldsymbol{u}_i^*$ is merely added to the activations of the i-th layer, it is possible to derive an adversarial gradient for this final forward pass. Although this can only approximate the gradient of the full defense, we find that it suffices to drastically improve the attack success rate. We use this gradient with BPDA and repeat the same 20-step PGD attack. The dynamic model now achieves only 9.5% robust accuracy (instead of 34.5%). Finally, the gradient approximation also enables attack by APGD with BPDA, which drops the robust accuracy further to 5.6%.

Additional evaluation details. We use the official implementation of the defense, which is publicly available.[14] At the time of our evaluation, some hyperparameter choices in the code deviated from those in the paper while others were included in neither. Hoping to reproduce the results in the paper, we evaluate the following configuration:

- We use 5000 images from the CIFAR-10 training set to learn the linear projection.
- We set the regularization strength (decaying the magnitude of control parameters) to zero, disabling regularization. This matches the published code. We do not observe any notable effect from using regularization.
- We set the number of defense iterations to $N = 5$ and learning rate to $\alpha = 10^{-3}$, which we find to be the only configuration that is close to the robustness results claimed in the paper, while at the same time retaining good accuracy on clean images.
- We apply the defense to a ResNet-20 trained with the provided code. As no pre-trained checkpoints are available, we refrain from performing costly adversarial training and evaluate only on this normally-trained model.

[13] This is the only configuration matching the reported accuracy on clean images while having non-trivial robustness against PGD.

[14] https://github.com/zhuotongchen/Towards-Robust-Neural-Networks-via-Close-loop-Control

Table 5.12: Robust accuracy on 1000 points of various static models (measured by APGD) and their adaptive version with Hedge Defense (HD) (using 20 steps), where we transfer APGD attacks obtained on the static model and APGD with BPDA and EoT attacks from the model using HD with 5 steps. We also include the worst-case among both transfer attacks.

Underlying model	Static defense	Transfer to HD with 20 steps using as surrogate		
		Static	HD (5 steps)	Worst-case
Gowal et al. [2020]	63.0%	65.9% ± 0.09	61.1% ± 0.14	61.0% ± 0.14
Carmon et al. [2019]	59.2%	65.5% ± 0.34	59.6% ± 0.22	59.2% ± 0.32
Andriushchenko and Flammarion [2020]	45.0%	54.1% ± 0.15	45.8% ± 0.12	45.2% ± 0.28

For the attack, we first run PGD with 20 steps of size $\frac{\epsilon}{4}$, mirroring the evaluation by Chen et al. [2021] but adding BPDA. We then run APGD with BPDA using the cross-entropy loss for 100 iterations (no restarts).

5.3.9 Attacking Adversarial Attacks as a Defense [Wu et al., 2021b]

Summary of method. The HD proposed by Wu et al. aims to defend against adversarial perturbations by *maximizing* the cross-entropy loss of the classifier's predictions summed over all classes. The intuition is that the images that are wrongly classified have stronger gradients, which dominate the optimization and easily reduce the confidence in the initial decision, while correctly classified points are minimally impacted by the defense. The proposed method does not require modifications of the training scheme of the underlying static model. HD solves, for a classifier f,

$$\arg\max_{\boldsymbol{\delta}} \sum_{k=1}^{K} L_{\text{ce}}(f(\boldsymbol{x}+\boldsymbol{\delta}), k), \quad \text{with } \|\boldsymbol{\delta}\|_\infty \leq \epsilon_{\text{d}},\ \boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^d \tag{5.4}$$

with 20 steps of PGD (and random initialization), $\epsilon_{\text{d}} = 8/255$ (the same used by the attacker) and step size $\eta = 4/255$. This preprocesses the input to the classifier f, and therefore applies to any model. Inference with HD costs 21 forward and 20 backward passes of the model.

Evaluation. The main experimental evaluation in Wu et al. [2021b] consists in transferring perturbations generated by several attacks on the underlying classifier f to the model equipped with HD. This results in absolute improvements of 2% to 4% in robust accuracy. An adaptive attack, equivalent to BPDA, is developed on the full defense, but it does not consider the randomness of the defense. This attack is reported to slightly reduce the effectiveness of HD on one model. The official code is not provided. Hence, we implement the defense algorithm ourselves using the details available in the paper.

Wu et al. report that PGD is often more effective than stronger methods like AutoAttack. We hypothesize that this is caused by the original implementation of AutoAttack which returns copies of unperturbed inputs when the attack is unsuccessful or when the original input is already misclassified. However, HD might also turn an originally correctly classified input into a misclassified one, and this is more likely if it is close to the decision boundary (as we expect an unsuccessful adversarial example to be). Ultimately, we use APGD on the targeted DLR loss with 5 target classes (that is 5 restarts of 50 iterations). We use BPDA and reduce the number of steps of HD to 5 when crafting the perturbations. To counter the randomness of the initial step of HD, we use 4 steps of EoT. We evaluate the obtained perturbations on the full defense with 20 steps. In Table 5.12, we report the robust accuracy obtained with our evaluation on static models available in `RobustBench` [Croce et al., 2021]. We also report the robust accuracy obtained by transferring adversarial perturbations from the underlying static models. We observe that HD does not provide clear improvements to the robustness of static models, and in one case it weakens it.

Additional evaluation details. The official code for HD is not available. As such, we implement it by following the details provided by Wu et al. [2021b]. The models used in our evaluation are the WideResNet-28-10 trained with extra data [Gowal et al., 2020, Carmon et al., 2019] and

the PreAct ResNet-18 [Andriushchenko and Flammarion, 2020]. As stated above, the default implementation of APGD (within the AutoAttack library) returns adversarial points only when they are successful (i.e., when misclassification occurs). When we use this default implementation, the HD model which uses the WideResNet-70-16 from Gowal et al. [2020] obtains 69.2% ± 0.10 (compared to 65.9% when using the points maximizing the target loss, see Table 5.12), which is in line with what reported by Wu et al. [2021b] for the same classifier. This gives us confidence that our implementation matches the one from Wu et al..

5.3.10 Online Adversarial Purification based on Self-Supervision [Shi et al., 2020]

Summary of method. This defense purifies adversarial perturbations by minimizing an auxiliary loss before performing inference. The auxiliary loss is connected to a self-supervised task, which may require an additional network which shares some feature representation with the classifier. Since the auxiliary loss is also optimized at training time, the system should learn to perform classification based on robust features that are shared across the supervised and auxiliary tasks. In the evaluation of Shi et al. [2020], Fast Gradient Sign Method (FGSM) is often a stronger attack than PGD, which contradicts the principles of Carlini et al. [2019a].

We focus on the defense variant that uses label consistency as auxiliary task since it does not require an additional network at test-time and yields the best results on CIFAR-10. It uses as auxiliary loss

$$L_{\text{aux}}(f, \boldsymbol{x}) = \|f(a_1(\boldsymbol{x})) - f(a_2(\boldsymbol{x}))\|_2\,, \tag{5.5}$$

with f the classifier and a_1, a_2 augmentations of the input. At test-time, the problem

$$\arg\min_{\boldsymbol{\delta}} L_{\text{aux}}(f, \boldsymbol{x}+\boldsymbol{\delta}), \quad \text{with } \|\boldsymbol{\delta}\|_\infty \leq \epsilon_{\text{d}},\ \boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^d \tag{5.6}$$

is optimized with 5 steps of PGD (without random initialization). The procedure is repeated for eleven values of ϵ_{d} and that attaining the lowest loss is chosen. Overall, inference requires $5 \cdot 11 \cdot 2$ forward and backward passes during purification plus 1 forward pass for the final classification.

Evaluation. We consider the small pre-trained model (referred to as ResNet-18 in Shi et al., 2020). It achieves a clean accuracy of 83.7% (after purification) on the first 1000 test images of CIFAR-10. The original evaluation reports for this model a robust accuracy of 51.02% under transfer from an FGSM attack on the static model. We run APGD with BPDA on the cross-entropy loss, but, instead of using as update direction (the sign of) the gradient of f with respect to the final purified image only (as one would get by approximating the whole purification process with the identity function), we average gradients over intermediate iterates produced by the purification process. Intuitively, this steers all intermediate images towards misclassification and make the attack more effective. Running this attack with 1000 iterations reduces the robust accuracy of the defense to 3.7%.

Additional evaluation details. We use the official implementation of the defense and the pre-trained model both publicly available.[15] Moreover, when transferring FGSM attacks from the static model, the defended adaptive classifier obtains a robust accuracy of 54.5%, which is in line with what is reported by Shi et al. [2020].

5.3.11 Adversarial Attacks are Reversible with Natural Supervision [Mao et al., 2021]

Summary of method. This defense states that images contain intrinsic structure that enables the reversal of adversarial attacks. In particular, they modify the inference step to purify the input using a trained contrastive representation g that leverages the intermediate activations of

[15] https://github.com/Mishne-Lab/SOAP

a pre-trained static model f. Given an input $\boldsymbol{x}$, the defense creates a modified input $\boldsymbol{x} + \boldsymbol{\delta}^\star$ where $\boldsymbol{\delta}^\star$ is the solution to

$$\arg\min_{\boldsymbol{\delta}} L_{\text{aux}}(g(t_1(\boldsymbol{x}+\boldsymbol{\delta})), g(t_2(\boldsymbol{x}+\boldsymbol{\delta}))), \quad \text{with } \|\boldsymbol{\delta}\|_\infty \leq 2\epsilon,\ \boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^d, \tag{5.7}$$

and is found using N PGD steps. L_{aux} is the contrastive loss and t_1, t_2 are two transformations randomly sampled from a set of predefined image augmentations. The authors use either two or four augmentations for each image and leverage a separate set of images to build negative pairs (to be used in the contrastive loss). The paper remains unclear about the provenance of these additional images. The released code uses augmented views of other attacked images to build the negative pairs and thus operates at the level of a batch, which can both weaken the defense and attack. At inference, the modified input is fed to a static model. In practice, Mao et al. set $N = 40$.

Evaluation. According to Mao et al.'s evaluation their best model achieves a robust accuracy of 67.79% against AutoAttack, 64.64% against PGD with 50 steps and 63.83% against the Carlini-Wagner attack with 200 steps [Carlini and Wagner, 2017a] on CIFAR-10 against l_∞-norm bounded perturbations of size $\epsilon = 8/255$. These attacks are performed on the underlying static model and transferred to the full defense. At first sight, it can seem surprising that AutoAttack is weaker since it consists of a suite of attacks with numerous restarts and many steps. However, by default AutoAttack returns the original images when these are originally misclassified or when no adversarial perturbation is found. This means that, in practice, most of the transferred points are clean images (since the underlying static model is already robust).

We replicate the setup from the authors which operates at the batch level. We use four image transformations and set the number of iterations to 40. For the purpose of this demonstration, and to be able to run our evaluation on a single NVIDIA V100 GPU, we reduce the batch size to 50 (instead of 512). While this change may negatively impact the defense, we found that transfer attacks match the results from Mao et al. [2021]: Using AutoAttack from the static model yields a robust accuracy of 67.0% (compared to 67.79%); Using a custom implementation of APGD (which returns worst-case adversarial examples) on the cross-entropy and DLR losses with 10 steps and 10 restarts, we obtain a robust accuracy of 63.9%, which in line with the worst-case robust accuracy obtained by Mao et al. (i.e., 63.83%). We note that under the same APGD attack, the static model obtains a robust accuracy of 59.4%. When attacking the full adaptive defense, we use BPDA and 16 EoT iterations. We obtain a robust accuracy of 58.4% and conclude that the proposed defense weakens the underlying static model. To determine the effect of randomness, we also perform our evaluation by fixing the random seed (and removing EoT). Without randomness, we obtain a robust accuracy of 56.4%.

Additional experimental details. We use the official implementation and pre-trained classifier publicly available.[16] We focus our evaluation on the `Semi-SL` model (from Carmon et al., 2019). The original implementation of the defense retrieves negative pairs of images (to construct the contrastive loss) from the batch of images produced by the attacker. As such, we follow the same protocol and allow the attacker to modify all the images of each batch. Unfortunately, as the setup from Mao et al. [2021] uses a batch size of 512 and requires several GPUs to work in parallel, we are forced to change the default batch size to 50, which may negatively impact the defense. Other than that we keep all hyperparameters identical.

To evaluate whether the attacker obtains an unfair advantage by allowing it to modify the whole batch, we also re-implement our own version of the input purification procedure described by Mao et al. [2021] with the negative images kept separate and hidden from the attacker. In that setup, we were unable to improve upon the robust accuracy of the underlying static model when using transfer attacks (using APGD on the static model) to the contrary of the full-batch setup.

[16] https://github.com/cvlab-columbia/SelfSupDefense

Table 5.13: Robust accuracy on 1000 points against l_∞-norm bounded attacks of static models alone and with AID-purifier.

Dataset	ϵ	Static model	AID-purified model	
			Transfer	Direct
CIFAR-10	8/255	49.3%	56.0%	43.8%
CIFAR-100	8/255	23.3%	32.3%	18.2%
SVHN	12/255	10.5%	62.0%	29.1%

5.3.12 AID-purifier: A Light Auxiliary Network for Boosting Adversarial Defense [Hwang et al., 2021]

Summary of method. This defense, proposed by Hwang et al. [2021], uses a discriminator to purify the input of a pre-trained classifier. The discriminator is trained to distinguish adversarially perturbed from clean inputs. It exploits the intermediate activations of the underlying static model. At inference time, every input is purified by minimizing the probability of perturbation according to the discriminator, with the goal of reducing any adversarial effect the input may have. Given a discriminator g, an input $\boldsymbol{x}$, $\epsilon_\text{d} > 0$, AID-purifier approximately solves the problem

$$\boldsymbol{\delta}^\star \approx \arg\min_{\boldsymbol{\delta}} g(\boldsymbol{x}+\boldsymbol{\delta}) \quad \text{with } \|\boldsymbol{\delta}\|_\infty \leq \epsilon_\text{d},\ \boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^d \tag{5.8}$$

with N steps of PGD and a fixed step-size α (the values of N and α vary across datasets). Inference is performed using the pre-trained classifier on the point $\boldsymbol{x}+\boldsymbol{\delta}^\star$. The inference costs N additional forward and backward passes of the discriminator compared to the standard one (in practice $T = 10$ is used).

Evaluation. The original evaluation relies mostly on transferring perturbations adversarial to the pre-trained classifier to the classifier endowed with purification. The authors also propose an adaptive attack which optimizes a convex combination of the classification loss and the output of the discriminator. This adaptive attack is not effective in decreasing the robust accuracy compared to the non-adaptive counterpart. We focus our analysis on static models trained using adversarial training [Madry et al., 2018] as their respective pre-trained discriminators are available publicly. We note that the reported improvement of AID-purifier on these models is rather small (1 to 2%) for all datasets, with the exception of SVHN where clean and robust accuracy increase by 22% and 27%, respectively. In the following, we consider the l_∞-norm bounded perturbations of size $\epsilon = 8/255$ for CIFAR-10 and CIFAR-100, and of size $\epsilon = 12/255$ for SVHN. We evaluate the robustness of the static models with AutoAttack, and that of the dynamic models with APGD combined with BPDA. Table 5.13 shows that for CIFAR-10 and CIFAR-100 the dynamic classifiers have lower robustness than the original static models. While AID-purifier seems to improve robustness on SVHN, we notice that, unlike on the other datasets, the static model has a low robust accuracy.[17]

Additional evaluation details. In our evaluation, we use the pre-trained classifiers, with WideResNet-34-10 as architecture, and discriminators, as well as the publicly available[18] original implementation of the defense. For the parameters of AID-purifier we use the values provided by Hwang et al. [2021]. Since on SVHN the defense seems to be beneficial, we further test it with APGD with 1000 iterations and 10 restarts (divided between cross-entropy and targeted DLR loss). This reduces the robust accuracy to 25.0%, suggesting that the robustness can be reduced by increasing the budget available to the attacker.

[17] For reference, we can obtain a classifier with more than 40% robust and 86% clean accuracy by standard adversarial training.

[18] https://openreview.net/forum?id=3Uk9_JRVwiF

5.3.13 Combating Adversaries with Anti-Adversaries [Alfarra et al., 2022]

Summary of method. This defense tries to counter gradient-based based attacks during inference by maximizing the classifier's confidence. It uses a static model f to predict pseudo-labels $\hat{y} = \arg\max_k [f(\boldsymbol{x})]_k$ for each input $\boldsymbol{x}$, and optimizes $\boldsymbol{x}' = \arg\min_{\boldsymbol{x}'} L_{\text{ce}}(f(\boldsymbol{x}', \hat{y}))$ with N gradient descent steps to increase the classifier's confidence in $\hat{y}$ (with L_{ce} being the cross-entropy loss). The goal is to cancel out attacks that aim to decrease this confidence score. In practice, N is set to 2, which leads to an increase in inference time of 8× (the original implementation also makes an additional unnecessary forward pass).

This design implies that the defense inherits the decision boundary of the underlying classifier, as increasing a classifier's confidence in its own predictions will not change its decision.[19] Consequently, there is no fundamental gain in robustness and any measured increase can only be the result of obfuscation: forcing the classifier to report high confidence almost everywhere flattens the loss landscape of the cross-entropy loss and causes numerical problems not only for PGD, but also for any score-based attack such as the Square Attack.

Evaluation. We focus on the strongest result reported by Alfarra et al. which consists of applying their defense for two iterations to a robust static CIFAR-10 model pre-trained with Adversarial Weight Perturbation (AWP) [Wu et al., 2020a]. As noted above and as noted by Alfarra et al., the defense can easily be circumvented by transferring adversarial examples from the underlying static model. Using APGD on the cross-entropy loss, we can reduce the robust accuracy of both the static and adaptive model to 63.7% against l_∞-norm bounded perturbations of size $\epsilon = 8/255$ on CIFAR-10.

However, we find that the defense is ineffective even against black-box decision-based attacks that require no knowledge of the static model at all. Testing the defense against a range of attacks (which exclude transfer attacks), Alfarra et al. report a robust accuracy of 79.21%. They also conduct an evaluation against decision-based attacks with inconclusive results as they report a robust accuracy of 86.0% against both the adaptive and static model, which is close the clean baseline of 88.0%. This suggests an incorrect use of the attack, or perhaps not enough iterations. As a consequence, we apply the more efficient RayS attack with 10K queries. We obtain a robust accuracy of 66.6%, which is much lower than the result reported by Alfarra et al. (i.e., 79.2%). We also obtain 66.6% against the static model, which shows that the defense has no effect against an attack that does not use confidence scores.

Additional experimental details. We use the official implementation of the defense, which is publicly available.[20] For the underlying classifier, we use a WideResNet-28-10 trained with AWP, with pre-trained weights obtained from the official AWP code repository.[21] Specifically, we use the RST-AWP checkpoint, which is also referenced by Alfarra et al. [2022] in their evaluation code. Further following the implementation by Alfarra et al. [2022], we set the number of defense iterations to $N = 2$ and the step size to $\alpha = 0.15$.

5.3.14 Adversarial Purification with Score-based Generative Models [Yoon et al., 2021]

Summary of method. This defense preprocesses the input of a classifier with an Energy-Based Model (EBM) trained, with Denoising Score-Matching (DSM), to learn a score function to denoise perturbed images. A particularity is that they allow only a few deterministic updates in the purification process (after adding noise to the initial input).

The proposed defense, named Adaptive Denoising Purification (ADP), purifies an input $\boldsymbol{x}$ with the iterative scheme, for $i = 1, \dots, T$.

$$\boldsymbol{x}_0 = \boldsymbol{x} + \boldsymbol{\xi}, \quad \boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \alpha_{i-i} s(\boldsymbol{x}_{i-1}) \tag{5.9}$$

[19] We observe that the decision boundary can sometimes change due to numerical inaccuracies. This effect is rare and disappears when reducing the defense step-size.

[20] https://github.com/MotasemAlfarra/Combating-Adversaries-with-Anti-Adversaries

[21] https://github.com/csdongxian/AWP

with $\boldsymbol{\xi} \sim \mathcal{N}(0, \sigma^2 I)$ an initial noise addition, α_i an adaptive step-size, and s the score function approximated by the EBM. The procedure is repeated S times, getting S points $\boldsymbol{x}_T^{(1)}, \ldots, \boldsymbol{x}_T^{(S)}$. Given a classifier f, the final classification is obtained as

$$\arg\max_{k=1,\ldots,K} \frac{1}{S} \sum_{s=1}^{S} [\text{softmax}(f(x_T^{(s)})]_k. \tag{5.10}$$

In practice, Yoon et al. set $T = 10$, $S = 10$ and $\sigma = 0.25$, and f is a pre-trained classifier. Hence, the inference costs S forward passes of f and $2 \cdot T \cdot S$ forward passes of s (since an additional call of the score networks is necessary for the computation of α_i).

Evaluation. In their experimental evaluation, Yoon et al. [2021] report that ADP achieves 69.71% against l_∞-norm bounded perturbations of size $\epsilon = 8/255$ on CIFAR-10. The strongest attack is reported to be PGD with BPDA and EoT, where BPDA approximates the iterative purification process with an identity. However, we note BPDA is not necessary since the defense consists only in forward passes of the score network and of the classifier from which gradients can be computed. As such, we apply 10 steps of APGD combined with 200 EoT iterations to alleviate the randomness induced by $\boldsymbol{\xi}$. This attack yields a robust accuracy of $33.7\% \pm 0.54$ averaged over 1000 images.[22]

Additional experimental details. As base classifier we use the WideResNet-28-10 model (as evaluated by Yoon et al., 2021) from RobustBench. The score network is provided by Song and Ermon [2020] online[23] (we use the checkpoint named `best_checkpoint_with_denoising`). The ADP classifier attains a clean accuracy of $86.9\% \pm 0.44$ on the 1000 points used for testing robustness. In our evaluation, we exclude the computation of the adaptive step size when getting the gradient of the target loss via backpropagation for simplicity (including it did not yield any improvements).

5.3.15 Improving Model Robustness with Latent Distribution Locally and Globally [Qian et al., 2021]

Summary of method. This defense introduces a new technique called Adversarial Training with Latent Distribution (ATLD). ATLD builds adversarial examples by fooling a discriminator network g. The discrimator has $C + 1$ outputs, where the zero-th output distinguishes between clean and adversarial images, while the remaining outputs predict the class. In a process similar to Madry et al. [2018], adversarial examples are used within a cross-entropy loss to train a static model. The claimed benefit of this approach is that adversarial examples are not biased towards the decision boundary of the classifier. The experiments demonstrate that the resulting model (denoted ATLD-) is more robust to weak adversarial attacks such as PGD with 10 steps. In addition to ATLD, the authors also propose a modification to the inference process called Inference with Manifold Transformation (IMF). They modify inference by repairing inputs $\boldsymbol{x}$ to $\boldsymbol{x} + \boldsymbol{\delta}^\star$ such that the trained discrimator is more likely to classify them as clean examples. The optimal purification offset $\boldsymbol{\delta}^\star$ is found using PGD with N steps to solve the problem

$$\arg\min_{\boldsymbol{\delta}} L_{\text{softmax}}[g(\boldsymbol{x} + \boldsymbol{\delta})]_0, \quad \text{with } \|\boldsymbol{\delta}\|_\infty \leq \epsilon_{\text{d}}, \ \boldsymbol{x} + \boldsymbol{\delta} \in [0, 1]^d. \tag{5.11}$$

The resulting models, dubbed ATLD and ATLD+, are seemingly more robust to evaluation pipelines like AutoAttack (reaching 65.07% robust accuracy against l_∞-norm bounded perturbations of size $\epsilon = 8/255$). In practice, Qian et al. set $N = 1$ and $\epsilon_{\text{d}} = 2\epsilon$.

Evaluation. As noted by Qian et al., the ATLD- model is only robust to weak attacks and completely breaks under AutoAttack. In our evaluation, using APGD on cross-entropy and targeted DLR losses (with 9 targets), the robust accuracy of the ATLD- model reduces to 7.7%

[22] Average and standard deviation are computed over 5 repeated evaluations over ADP of the adversarially perturbed points.

[23] https://github.com/ermongroup/ncsnv2

(compared to 65.40% against CW with 100 steps). The authors evaluate their full defense, which includes IMF, by transferring attacks from various static models. Surprisingly, transferring from the underlying ATLD- model results in weaker attacks than transferring from another model trained through adversarial training [Madry et al., 2018]. This suggests that the underlying static model obfuscates gradients. Consequently, we increase the number of restarts to 100. Since the defense consists of a single differentiable step, we attack the full defense with APGD on the cross-entropy and targeted DLR losses. Overall, we obtain a robust accuracy of 12.6% for the ATLD+ model, which is significantly lower than reported (i.e., 65.07%).

Additional evaluation details. We use the official implementation of the defense and the pre-trained models publicly available.[24] All hyperparameters are set as specified by Qian et al. [2021]. Our only modification consists in allowing gradients to flow through the optimization steps. We use the standard implementation of AutoAttack.

5.3.16 Discussion and recommendations

Our case study shows that evaluating adaptive test-time defenses is more challenging than evaluating static defenses because of several complicating factors. These factors include the possibly intricate optimization process, randomness, and the large computational requirements of running these defenses. Moreover, each defense has its own peculiarities, which frustrates attempts at standardizing evaluation. In fact, Table 5.10 shows that the most effective attack differs across cases. At the same time, the main elements constituting such attacks consist of techniques proposed in prior works, which can be adaptively combined depending on the elements of the defense. The main challenge is therefore to find the attack setup which is the most suitable for each case.

Carlini et al. [2019a] proposed guidelines to evaluate adversarial robustness, and applied their principles to build a strong attack methodology. Moreover, they identify signs of overestimated robustness (e.g., single-step attacks stronger than multi-step ones), which also hold for adaptive test-time defenses. Our case study suggests taking the following steps to extend their foundational guidance:

❶ Transfer attacks that are effective against the underlying static model. Consider using other models as surrogates, such as those available in the `RobustBench` model zoo. When transferring attacks, make sure to transfer unsuccessful adversarial attacks (i.e., perturbations that attain high loss but do not lead to misclassification) rather than transferring the unperturbed input (as a library may do by default, e.g., AutoAttack).

❷ Verify with various black-box attacks that the adaptive test-time defense is more robust than the underlying static model. Consider Square Attack which is score-based and RayS [Chen and Gu, 2020] which is decision-based for this purpose.

❸ Apply strong white-box attacks to the full defense when possible. We found APGD (with multiple losses and restarts) to be a reliable gradient-based attack in most cases. Modern frameworks such as PyTorch [Paszke et al., 2019] and JAX [Bradbury et al., 2018] enable easy gradient computation.

❹ When gradient attacks fail, due to non-differentiability or vanishing gradients caused by excessive iteration, consider combining APGD with BPDA and try various approximations for the backward pass. In particular, BPDA can simply replace the backward pass for the optimization process with the identity, or make use of intermediate iterates (see Sec. 5.3.10).

❺ When randomness is present, explicitly or implicitly, use EoT, or remove randomness altogether by fixing the seed at each attack step.

❻ Ultimately, although they can serve as baselines, attacks developed for static defenses are not guaranteed to be effective for adaptive test-time defenses. Consequently, always try to implement adaptive attacks that are specific to adaptive defenses. These attacks should be stronger than their non-adaptive counterparts, unlike the results reported in Mao et al. [2021] and Hwang et al. [2021].

Adaptive test-time defenses complicate robustness evaluation due to their complexity and computational cost. Despite these complications, our evaluation succeeds in reducing the apparent robustness of the studied defenses, with a relative reduction of more than 50% for four of

[24] https://github.com/LitterQ/ATLD-pytorch

the nine defenses. In all cases with an adversarially robust static model (five out of nine), the adaptive test-time defense does not improve upon it and might even weaken it. These results are disappointing, but we foresee that adaptive test-time defenses can still potentially lead to significant robustness gains, and clearly our results do not challenge the whole idea. While we could not provide a standardized evaluation protocol for adaptive test-time defenses, we hope that our case study and recommendations can guide future evaluations as new defenses are developed. eFurthermore, we emphasize the need to measure gains in robustness against the computation required. This weakness may be turned into a potential strength, if the additional computation used by the defender can impose even more computation on the attacker. Finally, we make that the code developed for our case study available.[25]

Recent advancements. After the appearance of our work, many techniques which can be categorized in our taxonomy for adaptive test-time defenses. While we could not expand our evaluation to include such more recent methods, this highlights the relevance of guidelines for evaluating the robustness of this type of defenses.

[25] https://github.com/fra31/evaluating-adaptive-test-time-defenses

Chapter 6

Sparse Adversarial Attacks

In Sec. 4 we have presented several algorithms to generate adversarial perturbations in the l_p-threat models. These methods have in common the tendency to perturb all pixels in an image: however, the pixelwise changes are, in most cases, of small magnitude. This ensures that the semantic content, and then the ground truth label, of the original image is preserved by the attack. Sparse attacks take instead a different approach: only a small fraction of input dimensions can be perturbed, but with very large modifications. Depending on the structure imposed on the perturbed pixels, sparse threat models include l_0-perturbations [Carlini and Wagner, 2017a] (pixels at any position can be modified), adversarial patches [Brown et al., 2017] (including only neighboring pixels) and frames [Zajac et al., 2019] (only the borders of the image are perturbed). In Sec. 6.1 and Sec. 6.2, we introduce several algorithms for such threat models.

Since sparse attacks do not constrain the size of the perturbations on a single pixel, they are localized but often very visible (e.g. introducing bright colors which stick out on uniform backgrounds). One option to limit this phenomenon is to add bounds on the l_∞-norm of the perturbations. Alternatively, we propose to adaptively decide how much each pixel can be perturbed, depending on e.g. how different it is from its neighbors or if it lays along a straight edge: this yields a set (named σ-map) of pixelwise constraints. We propose attacks for both cases in Sec. 6.1. We summarize the proposed algorithms in Table 6.1.

Table 6.1: Summary of algorithms for generating sparse attacks.

attack	*bounds type*	*knowledge*	*goal*
CornerSearch	l_0, $l_0 + l_\infty$, $l_0 + \sigma$-map	black-box	sparsity maximization
PGD_0	l_0, $l_0 + l_\infty$, $l_0 + \sigma$-map	white-box	fixed sparsity
Sparse-RS	l_0, patches, frames	black-box	fixed sparsity

6.1 Sparse and imperceivable attacks

In this section we focus on sparse adversarial attacks which modify the smallest amount of pixels in order to change the decision. There are currently white-box attacks based on variants of gradient based methods integrating the l_0-constraint [Carlini and Wagner, 2017a, Papernot et al., 2016b] or mainly black-box attacks which use either local search or evolutionary algorithms [Narodytska and Kasiviswanathan, 2017, Su et al., 2019, Schott et al., 2019]. In particular, 1) we suggest a novel black-box attack based on local search which outperforms all existing l_0-attacks, 2) we present closed form expressions or simple algorithms for the projections onto the l_0-ball (or intersection of l_0-ball and componentwise constraints) in order to extend the PGD attack of Madry et al. [2018] to the considered scenario, and 3) since sparse attacks are often clearly visible and thus, at least in some cases, easy to detect (see second image of Fig. 6.1), we combine the sparsity constraint (l_0-ball) with componentwise constraints, and we extend the two l_0-attacks mentioned above to produce sparse and imperceivable adversarial perturbations. Compared to Modas et al. [2019] who introduce global componentwise constraints (see third image of Fig. 6.1) we propose to use locally adaptive componentwise constraints. These local constraints ensure

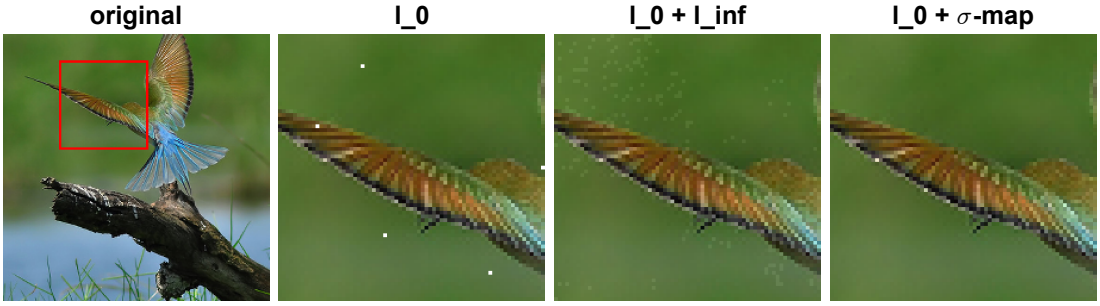

Figure 6.1: First image: original image with box for zoom, Second: our l_0-attack, very few pixels, only 0.04%, are changed, but the modified pixels are clearly visible, Third: the result of the $l_0 + l_\infty$-attack as proposed in Modas et al. [2019], the modifications are sparse, 2.7% of the pixels are changed, but clearly visible, Fourth: our sparse, 2.7% of the pixels are changed, but imperceivable attack ($l_0 + \sigma$-map).

that the change is typically not visible, that is we neither change color too much, nor we change pixels along edges aligned with the coordinate axis (see Sec. 6.1.6 for a visualization) or in regions which have uniform color (see fourth image of Fig. 6.1). This is in line with, and significantly improves upon, Luo et al. [2018], who suggest to perturb pixels in regions of high variance to have less recognizable modifications. In fact the often employed l_∞-attacks which modify each pixel only slightly but have to manipulate all pixels seem not to model perturbations which could actually occur. We think that our sparse and imperceivable attacks could happen in practice and correspond to modifications which do not change the semantics of the images even on very small scales. The good news of our paper is that the success rate of such attacks (50-70% success rate for standard models) is smaller than that of the commonly used ones. Nevertheless we find it disturbing that such manipulations are possible at all. Thus we also test if adversarial training can reduce the success rate of such attacks. We find that adversarial training wrt l_2 partially decreases the effectiveness of l_0-attacks, while adversarial training wrt either l_2 or l_∞ helps to be more robust against sparse and imperceivable attacks. Finally, we introduce adversarial training aiming specifically at robustness wrt both our attack models.

6.1.1 Input constraints for sparse (and imperceivable) attacks

We recall that, given a classifier f, minimal adversarial perturbation y^* of $x \in \mathbb{R}^d$ with respect to a distance function $\gamma : \mathbb{R}^d \longrightarrow \mathbb{R}_+$ is given as the solution $y^* \in \mathbb{R}^d$ of the optimization problem

$$\min_{y \in \mathbb{R}^d} \gamma(y - x) \quad \text{s.th.} \quad \arg\max_{r=1,\dots,K} f_r(y) \neq \arg\max_{r=1,\dots,K} f_r(x), \quad y \in C, \tag{6.1}$$

where C is a set of constraints valid inputs need to satisfy (e.g. images are scaled to be in $[0,1]^d$).

Sparse l_0-attacks. In an l_0-attack one is interested in finding the smallest number of pixels which need to be changed so that the decision changes. We write in the following gray-scale images x with d pixels as vectors in $[0,1]^d$ and color images x with d pixels as matrices x in $[0,1]^{d\times 3}$, and x_i denotes the i-th pixel with the three color channels in RGB. The corresponding distance function γ is thus given for gray-scale images as the standard l_0-norm

$$\gamma(y - x) = \sum_{i=1}^{d} \mathbb{1}_{|y_i - x_i| \neq 0}, \tag{6.2}$$

and for color images as

$$\gamma(y - x) = \sum_{i=1}^{d} \max_{j=1,\dots,3} \mathbb{1}_{|y_{ij} - x_{ij}| \neq 0}, \tag{6.3}$$

where the inner maximization checks if any color channel j of the pixel i is changed. From a practical point of view the l_0-attack tests how vulnerable the model is to failure of pixels or large localized changes on an object e.g. a sticker on a refrigerator or dirt/dust on a windshield.

Introducing imperceptibility constraints. The problem of l_0-attacks is that they are completely unconstrained in the way how they change each pixel. Thus the perturbed pixels have usually completely different color than the surrounding ones and thus are easily visible. On the other hand l_∞-attacks, using the distance function

$$\gamma(y-x) = \max_{i=1,\dots,d} \max_{j=1,\dots,3} |y_{ij} - x_{ij}|,$$

are known to result in very small changes per pixel but have to modify every pixel and color channel. This seems to be a quite unrealistic perturbation model from a practical point of view. A more realistic attack model which could happen in a practical scenario is when the changes are sparse but also imperceivable. In order to achieve this we come up with additional constraints on the allowed channelwise change. In Modas et al. [2019] they suggest to have global bounds, for some fixed $\delta > 0$, in the form

$$x_{ij} - \delta \leq y_{ij} \leq x_{ij} + \delta,$$

which should ensure that the changes are not visible (we call an attack with l_0-norm and these global componentwise bounds an $l_0 + l_\infty$-attack in the following). However, these global bounds are completely agnostic of the image and thus δ has to be really small so that the changes are not visible even in regions of homogeneous color, e.g. sky, where almost any variation is easily spotted. We suggest image-specific local bounds taking into account the image structure. We have two specific goals:

1. We do not want to make changes along edges which are aligned with the coordinate axis as they can be easily spotted and detected.

2. We do not want to change the color too much and rather just adjust its intensity and keep approximately also its saturation level.

In order to achieve this we compute the standard deviation of each color channel in x- and y-axis directions with the two immediate neighboring pixels and the original pixel. We denote the corresponding values as $\sigma_{ij}^{(x)}$ and $\sigma_{ij}^{(y)}$ and define $\sigma_{ij} = \sqrt{\min\{\sigma_{ij}^{(x)}, \sigma_{ij}^{(y)}\}}$. Since $\sigma_{ij}^{(x)}, \sigma_{ij}^{(y)} \in [0,1]$ the square root increases more significantly, in relative value, smaller $\min\{\sigma_{ij}^{(x)}, \sigma_{ij}^{(y)}\}$. In this way we both enlarge the space of the possible adversarial examples and prevent perturbations in areas of zero variance. In fact we allow the changed image y just to have values given by

$$y_{ij} = (1 + \lambda_i \sigma_{ij}) x_{ij}, \quad \text{with } -\kappa \leq \lambda_i \leq \kappa, \tag{6.4}$$

where $\kappa > 0$. Additionally, we enforce box constraints $y \in [0,1]^{d \times 3}$. Note that the parameter λ_i corresponds to a change in intensity of pixel i by maximally plus/minus $\kappa \sum_{j=1}^3 \sigma_{ij} x_{ij}$ as

$$\sum_{j=1}^3 y_{ij} = \sum_{j=1}^3 x_{ij} + \lambda_i \sum_{j=1}^3 \sigma_{ij} x_{ij}.$$

Thus we are just changing intensity of the pixel instead of the actual color. Moreover, note that this change also preserves the saturation of the color value[1] if the σ_{ij} are equal for $j = 1, \dots, 3$. Thus we fulfill the second requirement from above. Moreover, the first requirement is satisfied as $\sigma_{ij} = \sqrt{\min\{\sigma_{ij}^{(x)}, \sigma_{ij}^{(y)}\}}$, meaning that if along one of the coordinates there is no change in all color channels then the pixel cannot be modified at all. Thus pixels along a coordinate-aligned edge showing no change in color will not be changed. The attack model of sparse and imperceivable attacks will be abbreviated as $l_0 + \sigma$-map. For gray-scale images $x \in [0,1]^d$ we use instead

$$y_i = x_i + \lambda_i \sigma_i, \quad \text{with } -\kappa \leq \lambda_i \leq \kappa. \tag{6.5}$$

as there the approximate preservation of color saturation is not needed.

[1] In the HSV color space the saturation of a color is defined as $1 - \frac{\min\{R,G,B\}}{\max\{R,G,B\}}$, where R, G, B are the red/green/blue color channels in RGB color space.

6.1.2 Score-based attacks

Most of the existing black-box l_0-attacks either start with perturbing a small set of pixels and then enlarge this set until they find an adversarial example [Papernot et al., 2016b, Narodytska and Kasiviswanathan, 2017] or, given a successful adversarial manipulation, try to progressively reduce the number of pixels exploited to change the classification [Carlini and Wagner, 2017a, Schott et al., 2019]. Instead we introduce a flexible attack scheme where at the beginning one checks pixelwise targeted attacks and then sorts them according to the resulting gap in the classifier outputs. Then we introduce a probability distribution on the sorted list and sample one-pixel changes to generate attacks where more pixels are manipulated simultaneously. The distribution we use is biased towards the one-pixel perturbations which produce, when applied individually, already large changes in the classifier output. In this non-iterative scheme there is thus no danger to get stuck in suboptimal points. Moreover, while the attack has to test many points, its non-iterative nature allows to check the perturbed points in large batches which is thus much faster than an evolutionary attack. Even if the scheme is simple it outperforms all existing methods including white-box attacks.

One-pixel modifications. In the first step we check all one pixel modifications of the original image $x \in [0,1]^{d\times 3}$ (color) or $x \in [0,1]^d$ (gray-scale). The tested modifications depend on the attack model.

1. **l_0-attack:** for each pixel i we generate $8 = 2^3$ images changing the original color value to one of the 8 corners of the RGB color cube. Thus we name our method **CornerSearch**. This results in a set of $8d$ images, all one pixel modifications of the original image x, which we denote by $(z^{(j)})_{j=1}^{8d}$. For gray-scale images one just checks the extreme gray-scale values (black and white) and gets $(z^{(j)})_{j=1}^{2d}$.

2. **$l_0 + l_\infty$-attack:** for each pixel i we generate 8 images changing the original color value of $(x_{ij})_{j=1}^3$ by the corners of the cube $[-\epsilon, \epsilon]^3$ resulting again in $(z^{(j)})_{j=1}^{8d}$ images. For gray-scale we use $x_i \pm \epsilon$ resulting in total in $(z^{(j)})_{j=1}^{2d}$ images. If necessary we clip to satisfy the constraint $z^{(j)} \in [0,1]^{d\times 3}$ or $z^{(j)} \in [0,1]^d$.

3. **$l_0 + \sigma$-map attack:** for color images we generate for each pixel i two images by setting

$$y_{ij} = (1 \pm \kappa\sigma_{ij})x_{ij}, \quad j = 1, \dots, 3,$$

   where κ and σ_{ij} are as defined in Sec. 6.1.1. For gray scale images $x \in [0,1]^d$ we use

$$y_i = x_i \pm \kappa\sigma_i.$$

   Finally, we clip y_{ij} and y_i to $[0,1]$. Thus, this results in $(z^{(j)})_{j=1}^{2d}$ images. We call it **σ-CornerSearch**.

After the generation of all the images we get the classifier output $f(z^{(j)})_{j=1}^M$ for each of them, where M is the total number of generated images, either $M = 2d$ or $M = 8d$. Then, separately for each class $r \neq c$, where $c = \arg\max_{r=1,\dots,K} f_r(x)$, we sort the values of $f_r(z^{(j)}) - f_c(z^{(j)})$ in decreasing order $\pi^{(r)}$. That means for all $1 \leq s \leq M-1$

$$f_r(z^{(\pi_s^{(r)})}) - f_c(z^{(\pi_s^{(r)})}) \geq f_r(z^{(\pi_{s+1}^{(r)})}) - f_c(z^{(\pi_{s+1}^{(r)})}).$$

We introduce also an order $\pi^{(c)}$, sorting in decreasing order the quantities

$$\max_{r\neq c} f_r(z^{(j)}) - f_c(z^{(j)}).$$

The idea behind generating these one-pixel perturbations is to identify the pixels which push most the decision towards a particular class r or in case of the set $\pi^{(c)}$ towards an unspecific change. If $f_r(z^{(\pi_1^{(r)})}) - f_c(z^{(\pi_1^{(r)})}) > 0$ for some r, then the decision has changed by only modifying one pixel. In this case the algorithm stops immediately. Otherwise, one could try to iteratively select the most effective change and repeat the one-pixel perturbations. However, this is overly expensive and again suffers if suboptimal pixel modifications are chosen in the initial steps of the iterative scheme. Thus we suggest in the next paragraph a sampling scheme based on the obtained orderings, where one randomly selects k one-pixel modifications to combine in order to produce a multi-pixels attack.

Algorithm 10: CornerSearch

Input : x original image classified as class c, K number of classes, $N, k_{\max}, N_{\text{iter}}$
Output: y adversarial example

$y \leftarrow \emptyset$
create *one-pixels modifications* $(z^{(i)})_{i=1}^{M}$
if *exists* $u \in (z^{(i)})_{i=1}^{M}$ *classified not as* c **then**
  $y \leftarrow u$, **return**
end
compute orderings $\pi^{(1)}, \ldots, \pi^{(K)}$,
$k \leftarrow 2$
while $k \leq k_{max}$ **do**
  for $r = 1, \ldots, K$ **do**
    create the set $Y^{(r)}$ of N_{iter} "k-pixels modifications" towards class r (see paragraph above)
    if $\exists u \in Y^{(r)}$ *classified not as* c **then**
      $y \leftarrow u$, **return**
    end
  end
  $k \leftarrow k + 1$
end

Multi-pixels modifications. Most of the times the modifications of one pixel are not sufficient to change the decision. Suppose we want to generate a candidate for a targeted adversarial sample towards class r by changing at most k pixels, choosing among the first N one-pixel perturbations according to the ordering $\pi^{(r)}$. We do this by sampling k indices $(s_1, \ldots, s_k)$ in $\{1, \ldots, N\}$ from the probability distribution on $\{1, \ldots, N\}$ defined as

$$P(Z = i) = \frac{2N - 2i + 1}{N^2}, \quad i = 1, \ldots, N. \tag{6.6}$$

The candidate image $y^{(r)}$ is generated by applying all the k one-pixel changes defined in the images $z^{(\pi_{s_1}^{(r)})}, \ldots, z^{(\pi_{s_k}^{(r)})}$ to the original image x. Please note that we only sample from the top N one-pixel changes found in the previous paragraph and that the distribution on $\{1, \ldots, N\}$ is biased towards sampling on the top of the list e.g. $P(Z = 1) = \frac{2N-1}{N^2}$ is $2N - 1$ larger than $\mathrm{P}(Z = N) = \frac{1}{N^2}$. This bias ensures that we are mainly accumulating one-pixel changes which have led individually already to a larger change of the decision towards the target class r. We produce candidate images $y^{(1)}, \ldots, y^{(K)}$ for all K classes, having $K - 1$ candidate images targeted towards changes in a particular class and one image where the attack is untargeted (for $r = c$). In total we repeat this process N_{iter} times. The big advantage of the sampling scheme compared to an iterative scheme is that all these images can be fed into the classifier in batches in parallel which compared to a sequential processing is significantly faster. Moreover, it does not depend on previous steps and thus cannot get stuck in some suboptimal regions. As shown in the experiments this relatively simple sampling scheme performs better than sophisticated evolutionary algorithms (black-box attacks) and even white-box attacks.

Since we want to find adversarial examples differing from x in as few pixels as possible, we generate the batches $y^{(1)}, \ldots, y^{(K)}$ of candidate images as described above, gradually increasing k, up to a threshold $k_{\max}$, until we get a classification different from the original class c. Alg. 10 summarizes the main steps.

6.1.3 PGD-based attacks

The projected gradient descent (PGD) attack of Madry et al. [2018] is not aiming at finding the smallest adversarial perturbation but instead argues from the viewpoint of robust optimization about maximizing the loss

$$\max_{z \in C(x)} L(c, f(z)),$$

where $L : \{1, \dots, K\} \times \mathbb{R}^K \to \mathbb{R}_+$ is usually chosen to be the cross-entropy loss, c is the correct label of the point x and the set $C(x) \subset [0,1]^{d \times 3}$ (color images with d pixels) or $C(x) \subset [0,1]^d$ (gray-scale images). The interpretation in terms of robust optimization [Madry et al., 2018] has led to a now well-accepted way of adversarial training with the goal of getting robust wrt a fixed set of perturbations. The usage of PGD attacks during training is the de facto standard for adversarial training, which we will also use later on in Sec. 6.1.4. Commonly used as the set of allowed perturbations is the l_∞-ball: $C(x) = \{z \,|\, \|z - x\|_\infty \leq \epsilon,\, z \in [0,1]^d\}$ as the projection can be done analytically.

In order to extend PGD to l_0, $l_0 + l_\infty$ and $l_0 + \sigma$-map attacks, we first have to capture the sets allowed in our attack models in Sec. 6.1.1 and then find fast algorithms for the projections onto these sets. In the following we derive such projection algorithms, then define the gradient step used in PGD: in this way we introduce PGD_0 for the l_0 and $l_0 + l_\infty$-threat models, and σ-PGD for imperceivable manipulations.

Projection onto the l_0-ball and l_0+l_∞-ball. Given an original color image $x \in [0,1]^{d \times 3}$ we want to project a given point $y \in \mathbb{R}^{d \times 3}$ onto the set

$$C(x) = \big\{ z \in \mathbb{R}^{d \times 3} \,\big|\, \sum_{i=1}^{d} \max_{j=1,2,3} \mathbb{1}_{|z_{ij} - x_{ij}| > 0} \leq k,\; l_{ij} \leq z_{ij} \leq u_{ij} \big\}.$$

We can write the projection problem onto $C(x)$ as

$$\min_{z \in \mathbb{R}^{d \times 3}} \sum_{i=1}^{d} \sum_{j=1}^{3} (y_{ij} - z_{ij})^2 \quad \text{s.th.} \quad l_{ij} \leq z_{ij} \leq u_{ij}, \quad i = 1, \dots, d,\; j = 1, \dots, 3$$
$$\sum_{i=1}^{d} \max_{j=1,2,3} \mathbb{1}_{|z_{ij} - x_{ij}| > 0} > 0 \leq k$$

Ignoring the combinatorial constraint, we first solve for each pixel i the problem

$$\min_{z_i \in \mathbb{R}^3} \sum_{i=1}^{d} \sum_{j=1}^{3} (y_{ij} - z_{ij})^2 \quad \text{s.th.} \quad l_{ij} \leq z_{ij} \leq u_{ij}, \quad i = 1, \dots, d,\; j = 1, \dots, 3.$$

The solution is given by $z^*_{ij} = \max\{l_{ij}, \min\{y_{ij}, u_{ij}\}\}$. We note that each pixel can be optimized independently from the other pixels. Thus we sort in decreasing order π the gains

$$\phi_i := \sum_{j=1}^{3} (y_{ij} - x_{ij})^2 - \sum_{j=1}^{3} (y_{ij} - z^*_{ij})^2.$$

achieved by each pixel i. Thus the final solution differs from x in the k pixels (or less if there are less than k pixels with positive ϕ_i) which have the largest gain and is given by

$$z_{\pi_i j} = \begin{cases} z^*_{\pi_i j} & \text{for } i = 1, \dots, k,\; j = 1, \dots, 3, \\ x_{\pi_i j} & \text{else.} \end{cases}.$$

Using $l_{ij} = 0$ and $u_{ij} = 1$ we recover the projection onto the intersection of l_0-ball and $[0,1]^{d \times 3}$. For $l_0 + l_\infty$ note that the two constraints

$$0 \leq z_{ij} \leq 1, \quad -\epsilon \leq z_{ij} - x_{ij} \leq \epsilon,$$

are equivalent to:

$$\max\{0, -\epsilon + x_{ij}\} \leq z_{ij} \leq \min\{1, x_{ij} + \epsilon\}.$$

Thus by using

$$l_{ij} = \max\{0, -\epsilon + x_{ij}\}, \quad u_{ij} = \min\{1, x_{ij} + \epsilon\},$$

the set $C(x)$ is equal to the intersection of the l_0-ball of radius k, the l_∞-ball of radius ϵ around x and $[0,1]^{d \times 3}$.

Projection onto the intersection of the l_0-ball and the σ-map constraints: color images. We here present the projection step which is used for the σ-PGD attack, where we allow perturbations on at most k pixels and respecting the σ-map constraints which are defined in (6.4). Given a color image $x \in [0,1]^{d\times 3}$, we want to project a given point $y \in \mathbb{R}^{d\times 3}$ onto the set

$$C(x) = \big\{ z \in \mathbb{R}^{d\times 3} \,\big|\, \sum_{i=1}^{d} \max_{j=1,\dots,3} \mathbb{1}_{|z_{ij}-x_{ij}|>0} \leq k,\ (1-\kappa\sigma_{ij})x_{ij} \leq z_{ij} \leq (1+\kappa\sigma_{ij})x_{ij},\ 0 \leq z_{ij} \leq 1 \big\},$$

where d is the number of pixels, σ_{ij} are the pixelwise, channel-specific bounds defined in Sec. 6.1.1 and $\kappa > 0$ a given parameter. We can write the projection problem as

$$\begin{aligned} \min_{\lambda\in\mathbb{R}^d} \sum_{i=1}^{d}\sum_{j=1}^{3}(y_{ij} - (1+\lambda_i\sigma_{ij})x_{ij})^2 \quad \text{s.th.} \quad & -\kappa \leq \lambda_i \leq \kappa, \quad i = 1,\dots,d \\ & 0 \leq (1+\lambda_i\sigma_{ij})x_{ij} \leq 1,\ i = 1,\dots,d,\ j = 1,\dots,3 \\ & \sum_{i=1}^{d} \mathbb{1}_{|\lambda_i|>0} \leq k \end{aligned}$$

Ignoring the combinatorial constraint, we first solve for each pixel the problem

$$\min_{\lambda_i\in\mathbb{R}} \sum_{j=1}^{3}(y_{ij} - (1+\lambda_i\sigma_{ij})x_{ij})^2 \text{s.th.} \quad -\kappa \leq \lambda_i \leq \kappa, \quad 0 \leq (1+\lambda_i\sigma_{ij})x_{ij} \leq 1, \quad j = 1,\dots,3.$$

We first note that the last constraint is always fulfilled if $x_{ij} = 0$ or $\sigma_{ij} = 0$. In the other case we can rewrite the constraint as

$$-\frac{1}{\sigma_{ij}} \leq \lambda_i \leq \frac{1}{\sigma_{ij}}\Big(\frac{1}{x_{ij}} - 1\Big), \quad j = 1,\dots,3.$$

Combining all constraints yields

$$\lambda_i^{(l)} := \max\Big\{-\kappa, \max_{\substack{j \\ x_{ij}\neq 0, \sigma_{ij}\neq 0}} -\frac{1}{\sigma_{ij}}\Big\} \leq \lambda_i \leq \min\Big\{\kappa, \min_{\substack{j \\ x_{ij}\neq 0, \sigma_{ij}\neq 0}} \frac{1}{\sigma_{ij}}\Big(\frac{1}{x_{ij}} - 1\Big)\Big\} := \lambda_i^{(u)}.$$

The unconstrained solution is given by

$$\lambda_i' = \frac{\sum_{j=1}^{3} \sigma_{ij}x_{ij}(y_{ij} - x_{ij})}{\sum_{j=1}^{3} \sigma_{ij}^2 x_{ij}^2}.$$

Thus the optimal solution for each pixel i is given by

$$\lambda_i^* = \max\{\lambda_i^{(l)}, \min\{\lambda_i', \lambda_i^{(u)}\}\}.$$

The final solution of the original problem allows only to choose k pixels to be changed. For each pixel i the quantity

$$\phi_i := \sum_{j=1}^{3}(y_{ij} - x_{ij})^2 - \sum_{j=1}^{3}(y_{ij} - (1+\lambda_i^*\sigma_{ij})x_{ij})^2$$

represents the difference in how much the objective increases between the cases $\lambda_i = 0$ (that is y_i is projected to x_i) and $\lambda_i = \lambda_i^*$. Since we want to minimize the objective function, the optimal solution is obtained by sorting $(\phi_i)_{i=1}^d$ in decreasing order π and setting

$$\lambda_{\pi_i}^{(final)} = \begin{cases} \lambda_{\pi_i}^* & \text{if } i = 1,\dots,k, \\ 0 & \text{else .} \end{cases}.$$

Finally, the point belonging to $C(x)$ onto which y is projected is $z \in \mathbb{R}^{d\times 3}$ defined componentwise by

$$z_{ij} = (1+\lambda_i^{(final)}\sigma_{ij})x_{ij},\ i = 1,\dots,d,\ j = 1,\dots,3.$$

Table 6.2: **Comparison of different l_0-attacks.** While SparseFool is always successful it requires significantly more pixels to be changed. Our method CornerSearch requires out of all attacks the least *median* amount of pixels to be changed.

	LocSearchAdv	PA 10x	CW	SparseFool	JSMA	CornerSearch
black-box	Yes	Yes	No	No	No	Yes
MNIST						
success rate	91.39%	92.35%	87.9%	100%	99.6%	97.38%
mean (pixels)	17.56	8.82	46.04	19.44	83.92	9.21
median (pixels)	-	8	44	12	46	7
CIFAR-10						
success rate	97.32%	100%	100%	100%	100%	99.56%
mean (pixels)	38.4	4.63	16.55	16.10	54.5	2.75
median (pixels)	-	3	11	12	47	2

Projection onto the intersection of the l_0-ball and the σ-map constraints: color images: gray-scale images. Since gray-scale images have only one color channel and, to get imperceivable manipulations, we use additive modifications as defined in (6.5), we project onto the set, given the original image x,

$$C(x) = \left\{ z \in \mathbb{R}^d \,\middle|\, \sum_{i=1}^{d} \mathbb{1}_{|z_i - x_i| > 0} \leq k,\ x_i - \kappa\sigma_i \leq z_i \leq x_i + \kappa\sigma_i,\ 0 \leq z_i \leq 1 \right\}.$$

Defining

$$l_i := \max\{x_i - \kappa\sigma_i, 0\}, \quad u_i := \min\{x_i + \kappa\sigma_i, 1\},$$

we can see that in this case the problem is equivalent to the projection onto the intersection of an l_0-ball and box constraints and then solved as illustrated above.

PGD$_0$ algorithm. Using the derivation above of the projection onto the l_0- resp. $l_0 + l_\infty$-ball, we introduce an l_0 version of the well-known PGD attack on the cross-entropy function L, namely PGD$_0$. The iterative scheme, to be repeated for a fixed number of iterations, is, given an input x assigned to class c,

$$\begin{aligned} z^{(i)} &= x^{(i-1)} + \eta \cdot \nabla L(c, f(x^{(i-1)})) / \left\| \nabla L(c, f(x^{(i-1)})) \right\|_1 \\ x^{(i)} &= P_k(z^{(i)}), \end{aligned} \tag{6.7}$$

where $\eta \in \mathbb{R}_+$, $x^{(0)} = x$, $P_k(z)$ represents the projection onto the l_0-ball, with the radius fixed at k, and the l_∞-ball defined by the box constraint $x \in [0,1]^d$. Note that PGD$_0$ needs k to be specified and thus does not aim at the minimal modification to change the decision. Finally, changing the projection as described above, we obtain σ-PGD for the $l_0 + \sigma$-map threat model.

6.1.4 Experiments

In the experimental section, we evaluate the effectiveness of our score-based l_0-attack CornerSearch and our white-box attack PGD$_0$. Moreover, we give illustrative examples of our sparse and imperceivable $l_0 + \sigma$-map attacks σ-CornerSearch and σ-PGD (the latter in the Appendix). Finally, we test adversarial training wrt various norms as a defense against our l_0- and $l_0 + \sigma$-map-attacks.[2]

Evaluation of l_0-attacks. We compare CornerSearch with state-of-the-art attacks for sparse adversarial perturbations: LocSearchAdv [Narodytska and Kasiviswanathan, 2017], Pointwise Attack (PA) [Schott et al., 2019], Carlini-Wagner l_0-attack (CW) [Carlini and Wagner, 2017a],

[2]The code is available at https://github.com/fra31/sparse-imperceivable-attacks.

Table 6.3: **l_0-attacks on Restricted ImageNet.** We attack the 89 correctly classified points out of 100 points from the validation set with SparseFool and our algorithm CornerSearch. Due to the limit on the allowed number of pixel changes, CornerSearch is not always successful, but requires many less pixels to be changed.

	SparseFool	CornerSearch
black-box	No	Yes
success rate	100%	93.26%
mean (pixels)	143.2	106.7
median (pixels)	101	50

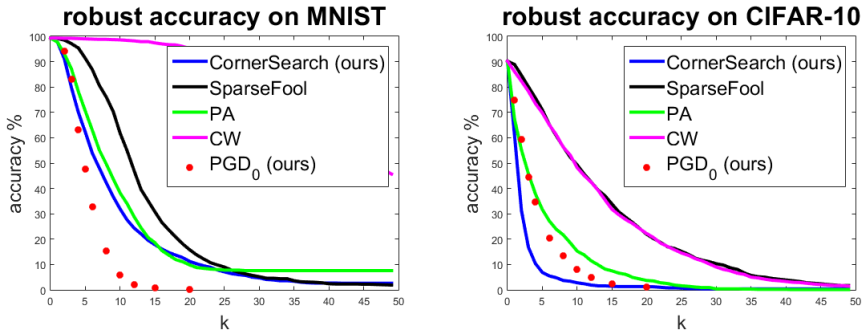

Figure 6.2: **Evaluation of PGD_0.** We compute for 1000 test points the robust accuracy of the classifier when the attack is allowed to perturb at most k pixels. We can see that PGD_0 (red dots) outperforms SparseFool, thus being the best "cheap" attack, and it is even the best one on MNIST for $k \geq 4$.

SparseFool (SF) [Modas et al., 2019], JSMA [Papernot et al., 2016b]. The first two operate in a black-box scenario, exploiting only the classifier output, like our method, while the latter three require access to the network itself (white-box attacks). Note that SparseFool is actually an l_1-attack, that means it uses the l_1-norm as distance measure in (6.1) in order to avoid the combinatorial problem arising from the usage of the l_0-norm. However, SparseFool can produce sparse attacks and in Modas et al. [2019] has been shown to outperform l_0-attacks in terms of sparsity. We use the implementation of the Pointwise Attack in Rauber et al. [2017] with 10 restarts as done in Schott et al. [2019], CW and JSMA from Papernot et al. [2018], while we reimplemented SparseFool. Since neither the code nor the models used in Narodytska and Kasiviswanathan [2017] are available (the results for LocSearchAdv are taken from Narodytska and Kasiviswanathan [2017]), we decided to compare the performance of the different attacks on one of the architectures reported in Narodytska and Kasiviswanathan [2017], the Network in Network [Lin et al., 2014] with batch normalization, retrained on MNIST and CIFAR-10.

We run the attacks on the first 1000 points of the corresponding test sets. We use CornerSearch with $k_{\max} = 50$, $N = 100$ and $N_{\text{iter}} = 1000$. In Table 6.2 we report the *success rate* of each method, that is the fraction of correctly classified points which can be successfully attacked, *mean* and *median* number of pixels that every attack needs to modify to change the decision. Please recall that MNIST consists of images with 784 pixels and CIFAR-10 of images with 1024 pixels. Although CornerSearch does not find an adversarial example for each test point, since we fix the maximum number of pixels that can be modified, both the average and median number of changed pixels are lower than those of the other methods, that is less pixels need to be perturbed by our method to change the decision (with the only exception of the mean on MNIST, where anyway CornerSearch has higher success rate and lower median than PA). On MNIST CornerSearch requires for at least 50% of all test images 0.89% of the pixels to be changed and for CIFAR-10 it is even just 0.2%.

Since PGD_0 needs k to be specified, in order to evaluate the robust accuracy, that is the accuracy of the classifier when the goal of the attacker is to change the decision of all correctly classified images using k-pixels modifications, one needs to evaluate PGD_0 for each value of k separately, whereas all other attacks yield the robust accuracy for all levels of sparsity in one run. For comparison we run PGD_0, using 20 iterations and 10 random restarts, with 10 sparsity

values k on the networks of Table 6.2 (see Sec. 6.1.5 for more details). In Fig. 6.2 we show the robust accuracy of the different attacks. PGD_0 achieves the best results on MNIST for $k \geq 4$, outperforms SparseFool and is even close to CornerSearch on CIFAR-10. As PGD_0 is very fast, it is a valuable alternative to our more expensive score-based attack.

l_0-attacks on Restricted ImageNet. We further test CornerSearch on Restricted ImageNet, that is a subset of ImageNet where some of the classes are grouped to form 9 distinct macro-classes. We use the ResNet-50 from Tsipras et al. [2019] and compare our attack to SparseFool (we do not run the other methods as either no code is available or they do not scale to the size of the images). The images have 50176 pixels. In Table 6.3 we report the statistics on 100 points for SparseFool and our attack with $k_{\max} = 1000, N_{\text{iter}} = 1000$. As for the other datasets, SparseFool always finds an adversarial example, whereas the smallest *mean* and *median* adversarial modification is achieved by CornerSearch, although with an inferior success rate. The runtime for SparseFool is around 55 times smaller than for CornerSearch. The runtime of our attack directly scales with the number of pixels and the time of a forward pass of the network, both large in this case. However, please note that SparseFool is a white-box attack, whereas ours is a black-box attack. For a comparison to PGD_0, given 100 pixels of budget, SF achieves a success rate of 49.4%, CS 64.0%, PGD_0 39.3%.

Sparse and imperceivable manipulations. We illustrate the differences of the adversarial modifications found by l_0-, $l_0 + l_\infty$- and $l_0 + \sigma$-map attacks. In Figures 6.3 and 6.4 we show some examples. As discussed before the adversarial modifications produced wrt only the l_0-norm are the sparsest but also the easiest to recognize. The $l_0 + l_\infty$-attack provides images where, although the absolute value of the individual modification is bounded (we use here $\delta = 0.1$ for CIFAR-10, $\delta = 0.05$ for ImageNet), some perturbations are visible since either colors are not homogeneous with the neighbors (second rows of the top part of Fig. 6.3 and bottom part of Fig. 6.4) or modifications of an uniform background are introduced (second row of the bottom part of Fig. 6.3 and top part of Fig. 6.4). On the other hand, the adversarial modifications of σ-CornerSearch are imperceivable while still being very sparse (third rows of Figures 6.3 and 6.4), showing that the σ-map, also shown in the Figures rescaled so that the largest component is equal to 1, is able to correctly identify the area where a change is difficult to perceive (see in particular the zoomed images). We provide a comparison of the adversarial examples crafted by σ-CornerSearch and σ-PGD in Sec. 6.1.6.

Table 6.4: **Evaluation of adversarial training.** Robust accuracy (%) given by l_0- and $l_0 + \sigma$-attacks (changing at most k pixels and for $l_0 + \sigma$-attacks fixing $\kappa = 0.8$ for MNIST and $\kappa = 0.4$ for CIFAR-10) on models adversarially trained wrt different metrics.

	l_0				$l_0 + \sigma$	
training	PA	SF	PGD_0	CS	σPGD	σCS
MNIST	$k = 15$				$k = 50$	
plain	25.6	41.2	**3.6**	9.2	49.2	80.4
l_∞*-at*	1.6	96.0	60.0	**0.0**	88.2	90.0
l_2*-at*	73.0	85.0	**34.0**	55.4	80.2	89.2
l_0*-at*	55.8	63.6	74.6	**39.8**	57.8	73.8
$l_0 + \sigma$*-at*	19.4	26.8	**9.4**	15.4	93.6	95.4
CIFAR-10	$k = 10$				$k = 100$	
plain	15.2	57.0	7.0	**2.2**	27.6	52.4
l_∞*-at*	28.6	57.6	22.6	**10.8**	50.4	63.6
l_2*-at*	37.6	60.6	25.4	**13.8**	53.2	66.0
l_0*-at*	64.2	63.8	54.8	**46.0**	34.6	58.2
$l_0 + \sigma$*-at*	41.6	54.4	6.0	**5.6**	62.8	67.6

Adversarial training. In order to increase robustness of the models to sparse adversarial manipulations, we adapt adversarial training to our cases. We use PGD_0 presented above for

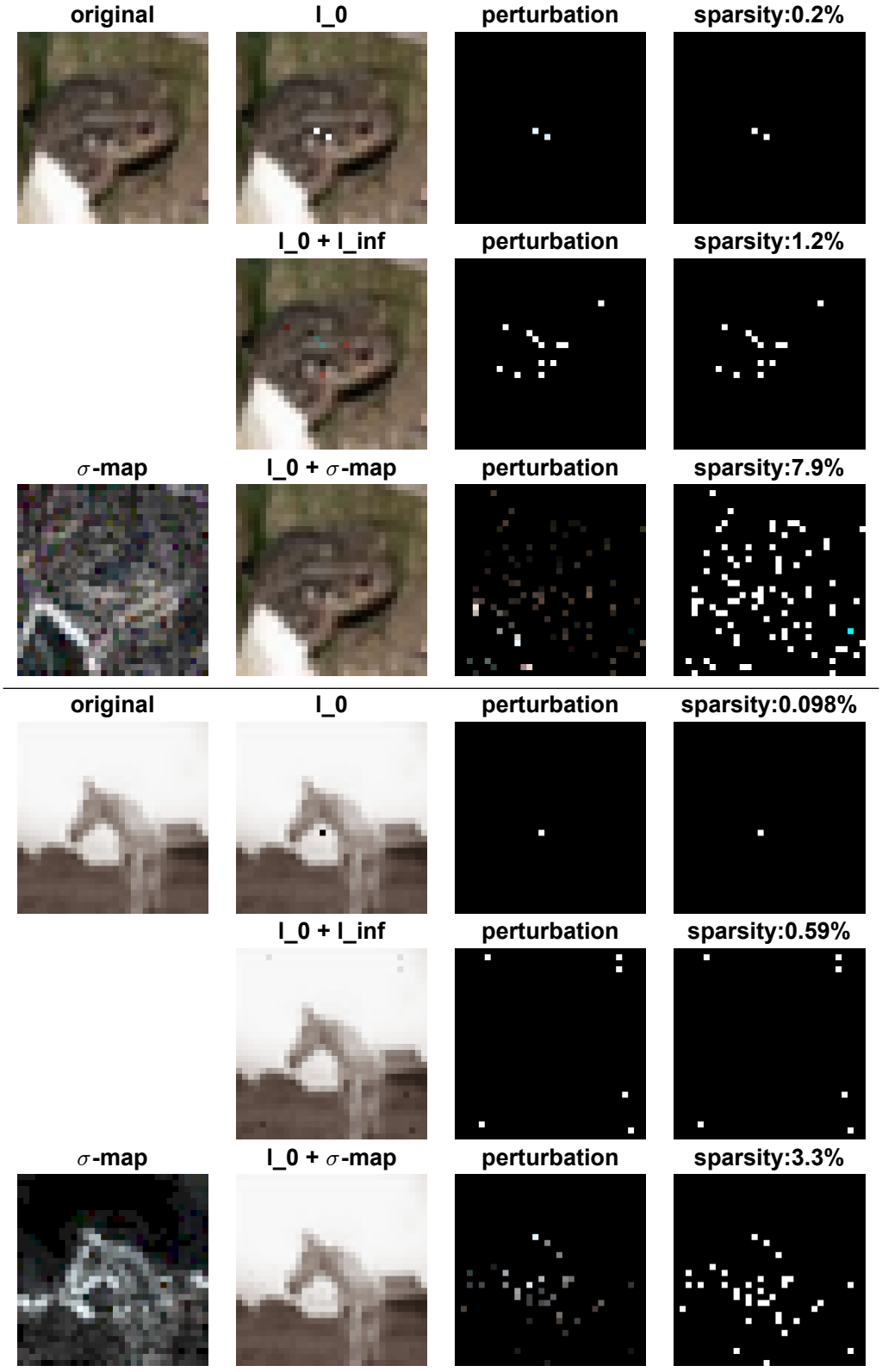

Figure 6.3: **Different attacks on CIFAR-10.** We illustrate the differences of the adversarial examples (second column) found by CornerSearch (l_0), l_0+l_∞-attack and σ-CornerSearch respectively first, second and third row. The third column shows the adversarial perturbations rescaled to $[0, 1]$, the fourth the map of the modified pixels (*sparsity* column). The original image can be found top left and the RGB representation of the σ-map bottom left.

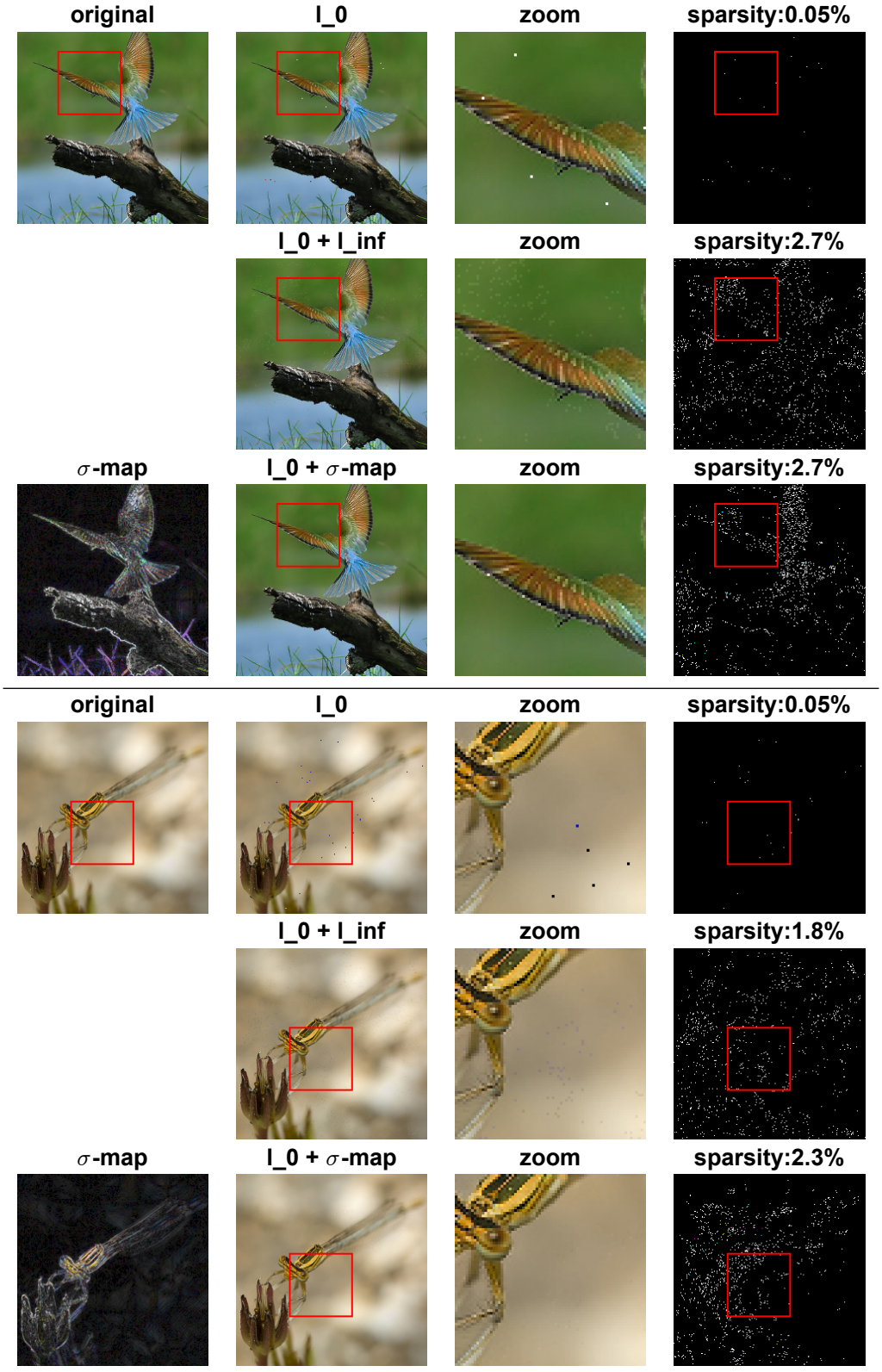

Figure 6.4: **Different attacks on Restricted ImageNet.** We illustrate the differences of the adversarial examples (second column, zoom in third column) found by CornerSearch (l_0), $l_0 + l_\infty$-attack and σ-CornerSearch, respectively first, second and third row. The fourth column shows the map of the modified pixels (*sparsity* column). The original image is in the top left and the RGB representation of the rescaled σ-map in the bottom left.

Table 6.5: **Accuracy of the attacked models.** We here report the accuracy on the test (validation set for Restricted ImageNet) set of the models introduced in Section 5.

experiments	*dataset*	*model*	*accuracy*
Comparison of l_0-attacks	MNIST	NiN [Lin et al., 2014]	99.66%
	CIFAR-10	NiN [Lin et al., 2014]	90.62%
	R-ImageNet	ResNet-50 [Tsipras et al., 2019]	94.46%
Sparse and imperceivable manipulation, Adversarial training	MNIST	*plain* [Madry et al., 2018]	99.17%
		l_∞-*at* [Madry et al., 2018]	98.53%
		l_2-*at*	98.95%
		l_0-*at*	96.38%
		$l_0 + \sigma$-*at*	99.29%
	CIFAR-10	*plain*	88.38%
		l_∞-*at*	79.90%
		l_2-*at*	80.44%
		l_0-*at*	82.31%
		$l_0 + \sigma$-*at*	76.24%
	R-ImageNet	ResNet-50 [Tsipras et al., 2019]	94.46%

adversarial training in order to achieve robustness against l_0-attacks (l_0-*at*), while we use σ-PGD to enhance robustness against sparse and imperceivable attacks ($l_0 + \sigma$-*at*). With these two techniques we train models on MNIST and CIFAR-10 (more details about the architectures and hyerparameters in Sec. 6.1.5). We compare them to the models trained on the *plain* training set and with adversarial training wrt the l_∞- and l_2-norm (l_∞-*at* and l_∞-*at*). In Table 6.4 we report the robust accuracy on 500 points (we fix the maximum number of pixels to be modified to k, and the parameter of the $l_0 + \sigma$ attacks defined in (6.4) and (6.5) to $\kappa = 0.8$ for MNIST and $\kappa = 0.4$ for CIFAR-10).

On MNIST the models trained on l_2 and l_0 perturbations are the most robust against l_0-attacks, while on CIFAR-10 the l_0-*at* model is more than 3 times more resistant than all the others. Similarly to Modas et al. [2019] we find that l_∞-*at* does not help for l_0-robustness. Notably, on both dataset our attacks PGD_0 and CornerSearch (CS) achieve the best results and then are the most suitable to evaluate robustness. Regarding the $l_0 + \sigma$-map attacks, we see that the $l_0 + \sigma$-*at* models are the least vulnerable, but also l_∞-*at* and l_2-*at* show some robustness. Note that σ-PGD is more successful than σ-CornerSearch but produces less sparse perturbations as it always fully exploits the budget of k pixels to modify while σ-CS mostly uses just a few of them, making the modifications even more difficult to spot (see Sec. 6.1.6).

6.1.5 Experimental details

We here report the details about the attacks, the attacked models and the parameters used in Sec. 6.1.4. The test accuracy (validation accuracy for Restricted ImageNet) of every model introduced in the paper is reported in Table 6.5.

Evaluation of l_0-attacks. The architecture used for this experiment is the Network in Network from Lin et al. [2014], which we trained according to a publicly available implementation,[3] adapting it also to the case of MNIST (which has different input dimension). For Restricted ImageNet, the attacked model is the ResNet-50 from Tsipras et al. [2019] (both weights and code are available[4]). We run PGD_0 with ten thresholds k (that is the maximum number of pixels that can be modified), which are $k \in \{2, 3, 4, 5, 6, 8, 10, 12, 15, 20\}$ for MNIST and $k \in \{1, 2, 3, 4, 6, 8, 10, 12, 15, 20\}$ for CIFAR-10.

Running times. We report the average running times for one image for the experiments in Sec. 6.1.4 (the times for SparseFool are those given by our reimplementation, which uses DeepFool as implemented in Rauber et al. [2017]). MNIST: LocSearchAdv 0.6s (from [25]), PA

[3] https://github.com/BIGBALLON/cifar-10-cnn
[4] https://github.com/MadryLab/robust-features-code

Figure 6.5: Left: original. Right: l_0-adversarial example with clearly visible changes along axis-aligned edges.

21s, CW 300s, SparseFool 2.5s, CornerSearch 9.8s, PGD_0 0.06s (one threshold). CIFAR-10: LocSearchAdv 0.7s [25], PA 22s, CW 283s, SparseFool 1.0s, CornerSearch 3.6s, PGD_0 0.19s (one threshold). ImageNet: SparseFool 17s, CornerSearch 953s, PGD_0 13s (one threshold).

Adversarial training. For MNIST, the architecture used is the same as in Madry et al. [2018], consisting of 2 convolutional layers, each followed by a max-pooling operation, and 2 dense layers. We trained our classifiers for 100 epochs with Adam [Kingma and Ba, 2014]. The *plain* and l_∞-*at* models are publicly available[5] while we trained l_2-*at* using the plain gradient as direction of the update for PGD, in contrast to the sign of the gradient which is used for adversarial training wrt the l_∞-norm. For adversarial training wrt l_0-norm we use $k = 20$ (maximal number of pixels to be changed), 40 iterations and step size $\eta = {}^{30000}/_{255}$. For $l_0 + \sigma$-*at* we set $k = 100$, $\kappa = 0.9$ (for the bounds given by the σ-map), 40 iterations of gradient descent with step size $\eta = {}^{30000}/_{255}$. On CIFAR-10, we use a CNN with 8 convolutional layers, consisting of 96, 96, 192, 192, 192, 192, 192 and 384 feature maps respectively, and 2 dense layers of 1200 and 10 units. ReLU activation function is applied on the output of each layer, apart from the last one. We perform the training with data augmentation (in particular, random crop and random mirroring are applied) for 100 epochs and with Adam optimizer. For adversarial training wrt l_0-norm we use $k = 20$ (number of pixels to be changed), 10 iterations of PGD with step size $\eta = {}^{30000}/_{255}$. For $l_0 + \sigma$-*at*, we use $k = 120$, $\kappa = 0.6$, 10 iterations of PGD with step size $\eta = {}^{30000}/_{255}$.

6.1.6 Additional experiments

Stability of CornerSearch. Since Alg. 10 involves a component of random sampling, we want to analyse here how the performance of CornerSearch depends on it. Then, we run CornerSearch for 10 times on the models used in Table 6.2 and get the following statistics: MNIST, success rate (%) 97.37 ± 0.13, *mean* 9.12 ± 0.05, *median* 7 ± 0. CIFAR-10, success rate (%) 99.33 ± 0.12, *mean* 2.71 ± 0.02, *median* 2 ± 0. This means that our attack is stable across different runs.

An example of visible changes. In Fig. 6.5 we show an example of how changes along axis-aligned edges are evident and easy to detect, even if the color is similar to that of some of the neighboring pixels. This provides a further justification of the heuristic we use to decide where the images can be perturbed in an invisible way.

Sparse and imperceivable manipulations on MNIST. In Fig. 6.6 we show the differences among the adversarial examples found by our attacks CornerSearch (l_0), black-box $l_0 + l_\infty$-attack and σ-CornerSearch. We see that our $l_0 + \sigma$-map attack does not modify pixels in the background, far from the digit or in areas of uniform color in the interior of the digit itself. Moreover, while, in the example shown in the lower left quadrant of Fig. 6.6, the original image is correctly classified as a "4", we notice that the adversarial example found by $l_0 + l_\infty$-attack shows features of a new class. The modified pixels bridge the gap between the vertical segments in the upper part of the digit, making it similar to a "9". This means that this image does not clearly belong to any class and thus cannot be considered as an obvious adversarial sample. Conversely, the $l_0 + \sigma$-attack does not change the shape of the digit, which can still be clearly recognized as belonging to the

[5] https://github.com/MadryLab/mnist_challenge

original class "4". The attacked model is the *plain* model. For the l_0+l_∞-attack we use a bound on the l_∞-norm of the perturbation of $\delta = 0.2$.

Additional figures. We show in Fig. 6.7 more examples on CIFAR-10 built as those in Fig. 6.3 (same attacked model). For the l_0+l_∞-attack we use a bound on the l_∞-norm of the perturbation of $\delta = 0.1$. For Restricted ImageNet we show in Fig. 6.8 more examples created as those in Fig. 6.4. For the $l_0 + l_\infty$-attack we use $\delta = 0.05$ as bound on the l_∞-norm of the perturbation.

Adversarial examples of σ-PGD. We want here to compare the adversarial examples generated by our two methods, σ-CornerSearch and σ-PGD. In Fig. 6.9 (MNIST) and Fig. 6.10 (CIFAR-10) we show the perturbed images crafted by the two attacks, as well as the original images and the modifications rescaled so that each component is in $[0, 1]$ and the largest one equals 1. Moreover, for σ-PGD we report the results obtained with a smaller κ. The gray images indicate unsuccessful cases. It is clear that σ-CornerSearch produces sparser perturbations. Moreover, σ-PGD with the same κ used for σ-CS gives more visible manipulations. We think that this is due to two reasons: first, the whole budget of k pixels to modify is always used by σ-PGD, while this does not happen with σ-CS, and second, σ-PGD aims at maximizing the loss inside the space of the allowed perturbations. This is possibly achieved by modifying neighboring pixels, which sometimes have slightly different colors, in opposite directions (that is with λ_i with different signs for different i). Conversely, σ-CornerSearch does not consider spatial relations among pixels, and thus it does not show this behaviour. However, as showed in the Figures, it is possible to recover less visible changes also for σ-PGD by decreasing κ, at the cost of a smaller success rate.

6.1.7 Recent advancements

Many new methods for sparse l_0-attacks have appeared after the publication of our work [Pooladian et al., 2020, Césaire et al., 2020, Fan et al., 2020, Matyasko and Chau, 2021, Pintor et al., 2021]. We discuss these attacks in the next section, where we also introduce a new framework for black-box sparse attacks.

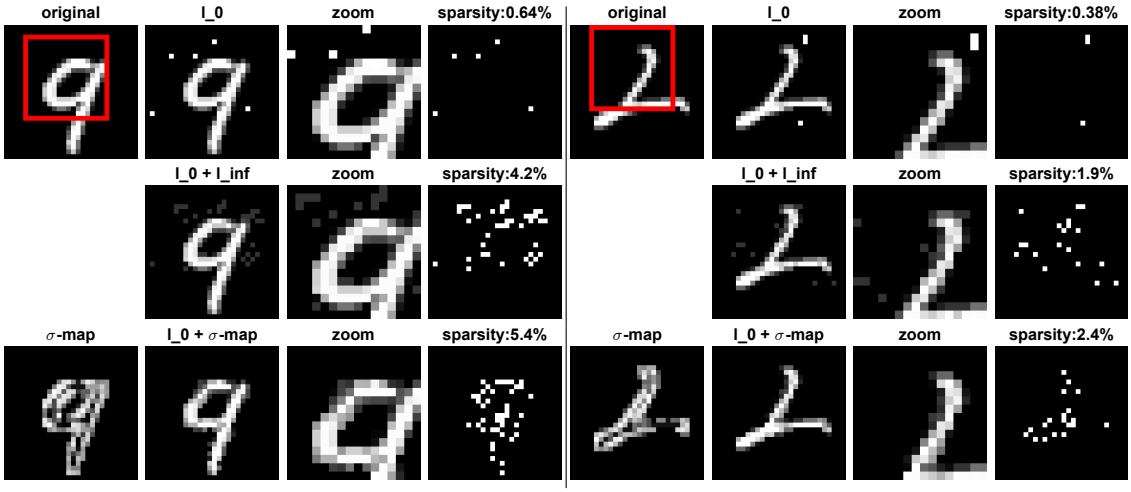

Figure 6.6: **Different attacks on MNIST.** We illustrate the differences of the adversarial examples (second column) found by CornerSearch (l_0), $l_0 + l_\infty$- and σ-CornerSearch, respectively first, second and third row. The third column shows the zoom of the area highlighted by the red box while the fourth column contains the map of the modified pixels (*sparsity* column). The original image can be found top left and the visualization of the σ-map (rescaled so that $\max_i \sigma_i = 1$) bottom left.

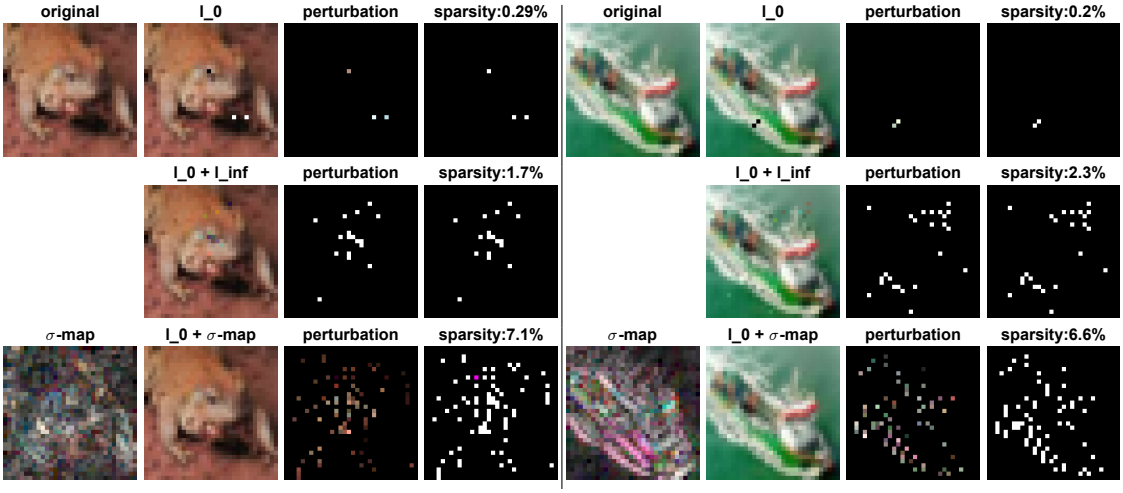

Figure 6.7: **Different attacks on CIFAR-10.** We illustrate the differences of the adversarial examples (second column) found by CornerSearch (l_0), $l_0 + l_\infty$-attack and σ-CornerSearch, respectively first, second and third row. The third column shows the adversarial perturbations rescaled to $[0, 1]$, the fourth the map of the modified pixels (*sparsity* column). The original image can be found top left and the RGB representation of the σ-map, rescaled so that $\max_{i,j} \sigma_{ij} = 1$, bottom left.

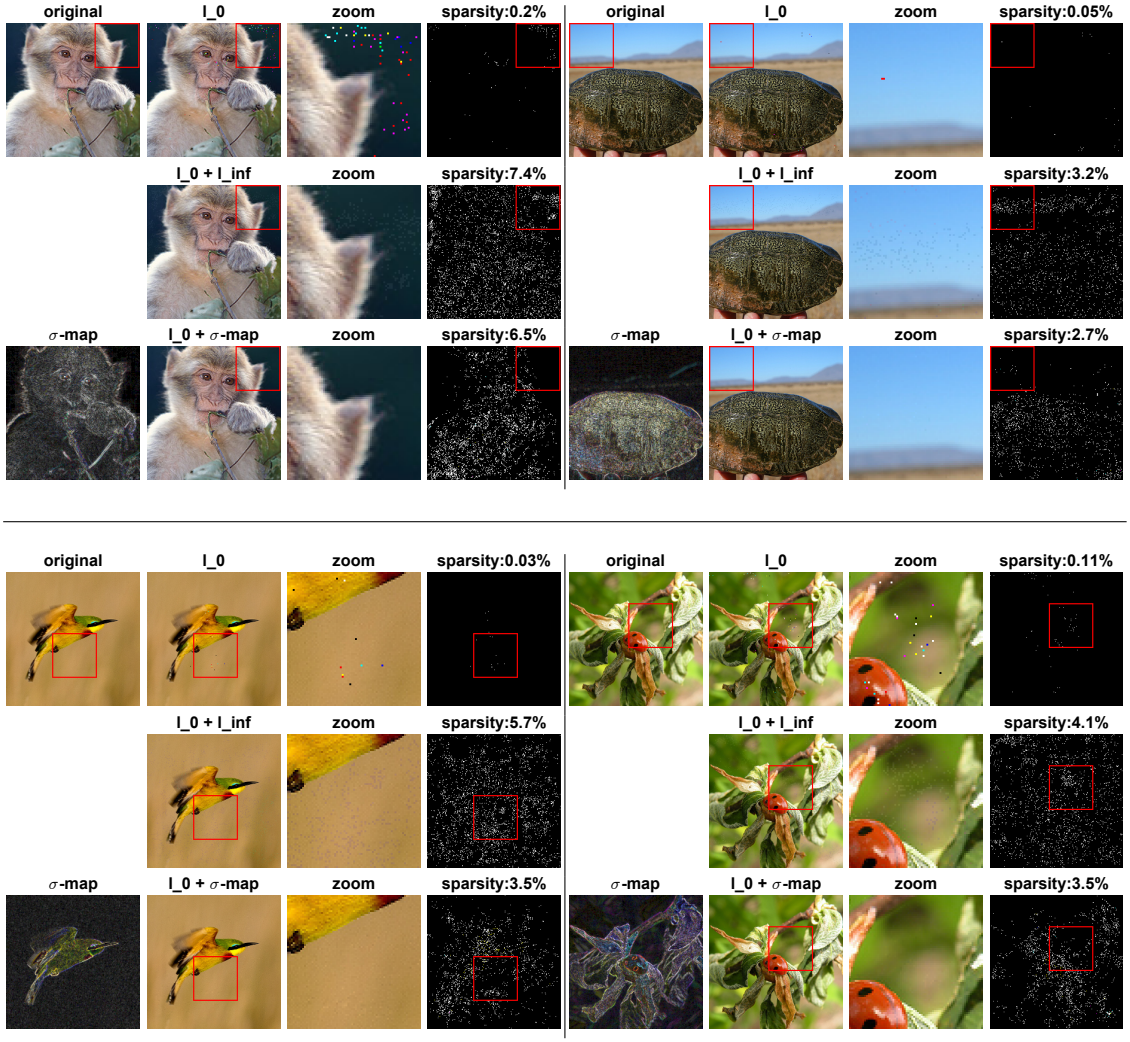

Figure 6.8: **Different attacks on Restricted ImageNet.** We illustrate the differences of the adversarial examples (second column, zoom in third column) found by CornerSearch (l_0), $l_0 + l_\infty$-attack and σ-CornerSearch, respectively first, second and third row. The fourth column shows the map of the modified pixels (*sparsity* column). The original image is in the top left and the RGB representation of the σ-map, rescaled so that $\max_{i,j} \sigma_{ij} = 1$, bottom left.

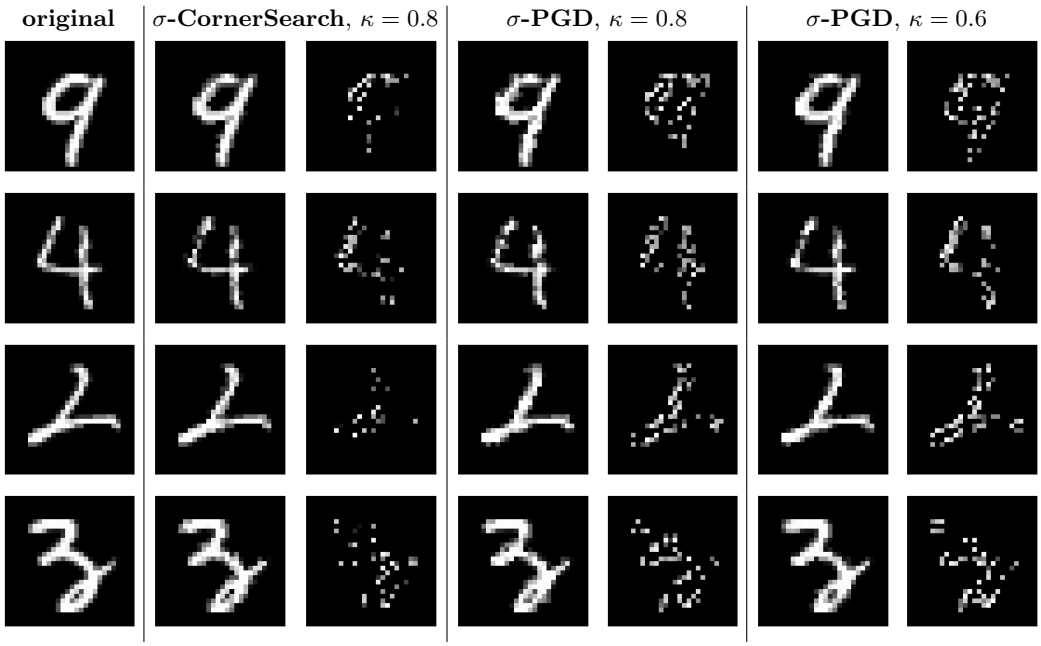

Figure 6.9: **Comparison σ-CornerSearch and σ-PGD on MNIST**. We show the adversarial examples generated by σ-CornerSearch with $\kappa = 0.8$, σ-PGD with $\kappa = 0.8$ and σ-PGD with $\kappa = 0.6$, together with the respective perturbations rescaled to [0,1]. The sparsity level used for σ-PGD is $k = 50$.

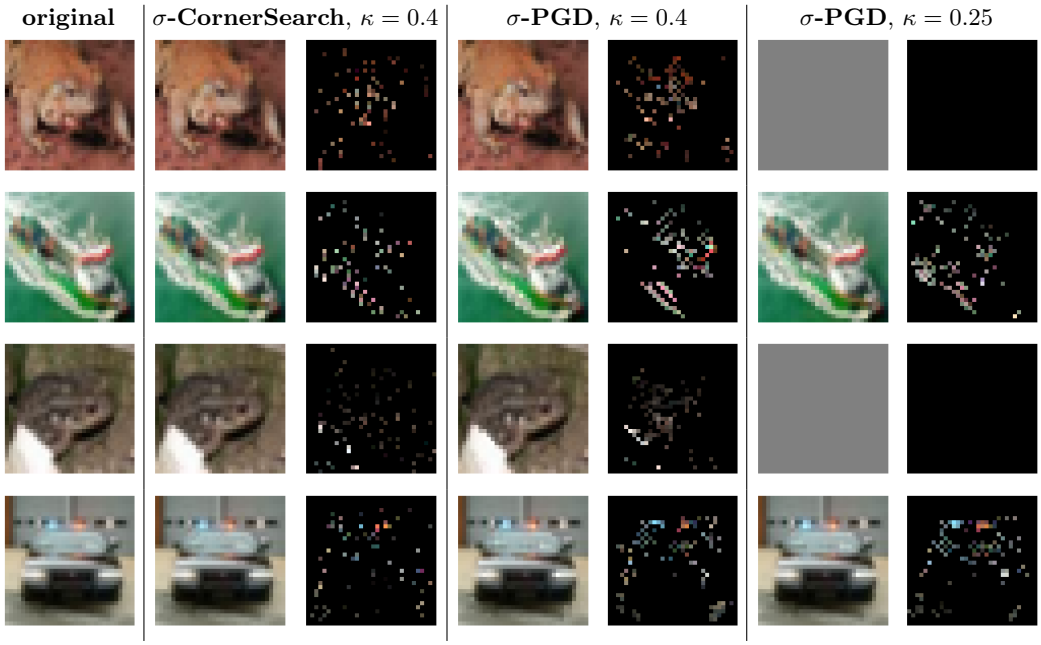

Figure 6.10: **Comparison σ-CornerSearch and σ-PGD on CIFAR-10**. We show adversarial examples generated by σ-CornerSearch with $\kappa = 0.4$, σ-PGD with $\kappa = 0.4$ and σ-PGD with $\kappa = 0.25$, together with the respective perturbations rescaled to [0,1]. The sparsity level used is $k = 100$. The gray images means the method could not find an adversarial manipulation.

6.2 Sparse-RS: a versatile framework for query-efficient sparse black-box adversarial attacks

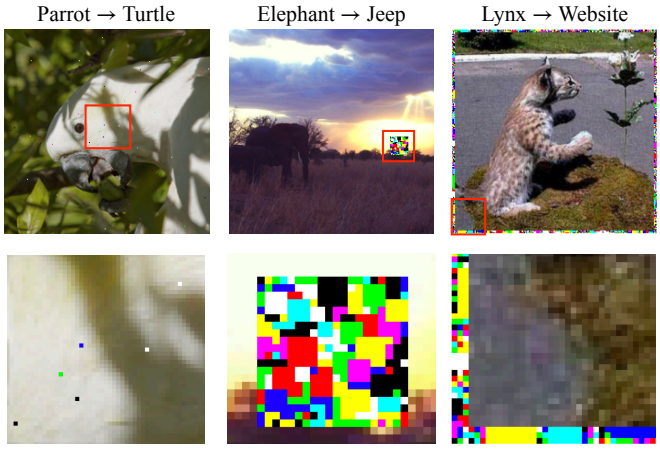

Figure 6.11: Adversarial examples for sparse threat models (l_0-bounded, patches, frames) generated with our black-box `Sparse-RS` framework which does not require surrogate models and is more query efficient.

We now consider sparse black-box adversarial attacks where the number of pixels which can be perturbed is fixed. Similarly to the setup used in Sec. 4.2, we compare attacks in term of both success rate and query efficiency, since we seek methods which are able to efficiently fool a target model. We can generate such attacks by solving

$$\min_{\delta \in \mathbb{R}^d} L(f(x+\delta), c) \quad \text{s.t.} \quad x + \delta \in [0,1]^d \quad \text{and} \quad \delta \in \mathcal{T} \tag{6.8}$$

by choosing a label c, a loss function L whose minimization leads to the desired classification, and a set of allowed perturbations $\mathcal{T}$. In our case $\mathcal{T}$ captures the various sparse constraints: for l_0-attacks any set of at most k pixels can be perturbed, for adversarial patches we restrict the perturbations to be squares of size $s \times s$ pixels, and for adversarial frames of width w, $\mathcal{T}$ contains the pixels which are at most w pixels away from the image borders. We illustrate the three threat models in Fig. 6.11.

Random search is particularly suitable for zeroth-order optimization in presence of complicated combinatorial constraints, as those of sparse threat models. Then, we design specific sampling distributions for the random search algorithm to efficiently generate sparse black-box attacks. The resulting `Sparse-RS` is a simple and flexible framework which handles

- **l_0-perturbations**: `Sparse-RS` significantly outperforms the existing black-box attacks in terms of the query efficiency and success rate, and leads to a better success rate even when compared to the state-of-the-art *white-box* attacks on standard and robust models.
- **Adversarial patches**: `Sparse-RS` achieves better results than both TPA [Yang et al., 2020a] and a black-box adaptations of projected gradient descent (PGD) attacks via gradient estimation.
- **Adversarial frames**: `Sparse-RS` outperforms the existing adversarial framing method [Zajac et al., 2019] with gradient estimation and achieves a very high success rate even with 2-pixel wide frames.

Our approach is close in spirit to that of Square Attack (Sec. 4.2) for l_p-bounded perturbations, which shows how random search can be effective to build adversarial attacks. In the following we show that random search is ideally suited for sparse attacks, where the non-convex, combinatorial constraints are not easily handled even by gradient-based white-box attacks.

Algorithm 11: `Sparse-RS`

Input: loss L, input x_{orig}, max query N, sparsity k, input space constraints $\mathcal{S}$
Output: approximate minimizer of L

1 $M \leftarrow k$ indices of elements to be perturbed
2 $\Delta \leftarrow$ values of the perturbation to be applied
3 $z \leftarrow x_{\text{orig}}, \quad z_M \leftarrow \Delta$ `// set elements in` M `to values in` Δ
4 $L^* \leftarrow L(z), \quad i \leftarrow 0$ `// initialize loss`
5 **while** $i < N$ **and** *success not achieved* **do**
6 $\quad M' \leftarrow$ sampled modification of M `// new set of indices`
7 $\quad \Delta' \leftarrow$ sampled modification of Δ `// new perturbation`
8 $\quad z \leftarrow x_{\text{orig}}, \quad z_{M'} \leftarrow \Delta'$ `// create new candidate in` $\mathcal{S}$
9 $\quad$ **if** $L(z) < L^*$ **then**
10 $\quad\quad L^* \leftarrow L(z), \quad M \leftarrow M', \quad \Delta \leftarrow \Delta'$ `// if loss improves, update sets`
11 $\quad$ **end**
12 $\quad i \leftarrow i + 1$
13 **end**
14 $z \leftarrow x_{\text{orig}}, \quad z_M \leftarrow \Delta$ `// return best` Δ
15 **return** z

6.2.1 `Sparse-RS` framework

As mentioned in Sec. 4.2, random search (RS) is a well known scheme for derivative free optimization [Rastrigin, 1963]. Given an objective function L to minimize, a starting point $x^{(0)}$ and a sampling distribution $\mathcal{D}$, an iteration of RS at step i is given by

$$\delta \sim \mathcal{D}(x^{(i)}), \quad x^{(i+1)} = \underset{y \in \{x^{(i)}, x^{(i)}+\delta\}}{\arg\min} L(y). \tag{6.9}$$

At every step an update of the current iterate $x^{(i)}$ is sampled according to $\mathcal{D}$ and accepted only if it decreases the objective value, otherwise the procedure is repeated. Although not explicitly mentioned in Eq. (6.9), constraints on the iterates $x^{(i)}$ can be integrated by ensuring that δ is sampled so that $x^{(i)} + \delta$ is a feasible solution. Thus even complex, e.g. combinatorial, constraints can easily be integrated as RS just needs to be able to produce feasible points in contrast to gradient-based methods which depend on a continuous set to optimize over. While simple and flexible, RS is an effective tool in many tasks [Zabinsky, 2010], with the key ingredient for its success being a task-specific sampling distribution $\mathcal{D}$ to guide the exploration of the space of possible solutions.

We summarize our general framework based on random search to generate sparse adversarial attacks, `Sparse-RS`, in Alg. 11, where the sparsity k indicates the maximum number of features that can be perturbed. A sparse attack is characterized by two variables: the set of components to be perturbed M and the values Δ to be inserted at M to form the adversarial input. To optimize over both of them we first sample a random update of the locations M of the current perturbation (step 6) and then a random update of its values Δ (step 7). In some threat models (e.g. adversarial frames) the set M cannot be changed, so $M' \equiv M$ at every step. How Δ' is generated depends on the specific threat model, so we present the individual procedures in the next sections. We note that for all threat models, the runtime is dominated by the cost of a forward pass through the network, and all other operations are computationally inexpensive.

Common to all variants of `Sparse-RS` is that the whole budget for the perturbations is fully exploited both in terms of number of modified components and magnitude of the elementwise changes (constrained only by the limits of the input domain $\mathcal{S}$). This follows the intuition that larger perturbations should lead faster to an adversarial example. Moreover, the difference of the candidates M' and Δ' with M and Δ shrinks gradually with the iterations which mimics the reduction of the step size in gradient-based optimization: initial large steps allow to quickly decrease the objective loss, but smaller steps are necessary to refine a close-to-optimal solution at the end of the algorithm. Finally, we impose a limit N on the maximum number of queries of the classifier, i.e. evaluations of the objective function.

As objective function L to be minimized, we use in the case of untargeted attacks the margin loss $L_{\text{margin}}(f(\cdot), y) = f_y(\cdot) - \max_{r \neq y} f_r(\cdot)$, where y is the correct class, so that $L < 0$ is equivalent to misclassification, whereas for targeted attacks we use the cross-entropy loss L_{CE} of the target class t, namely $L_{\text{CE}}(f(\cdot), t) = -f_t(\cdot) + \log\left(\sum_{r=1}^{K} e^{f_r(\cdot)}\right)$.

The code of the `Sparse-RS` framework is available at https://github.com/fra31/sparse-rs.

6.2.2 `Sparse-RS` for l_0-bounded attacks

The first threat model we consider is that of l_0-bounded adversarial examples where only up to k pixels or k features/color channels of an input $x_{\text{orig}} \in [0,1]^{h \times w \times c}$ (width w, height h, color c) can be modified, but there are no constraints on the magnitude of the perturbations except for those of the input domain. Note that constraining the number of perturbed pixels or features leads to two different threat models which are not directly comparable. Due to the combinatorial nature of the l_0-threat model, this turns out to be quite difficult for continuous optimization techniques which are more prone to get stuck in suboptimal maxima.

l_0-`RS` algorithm. We first consider the threat model where up to k pixels can be modified. Let U be the set of the $h \cdot w$ pixels. In this case the set $M \subset U$ from Alg. 11 is initialized sampling uniformly k elements of U, while $\Delta \sim \mathcal{U}(\{0,1\}^{k \times c})$, that is random values in $\{0,1\}$ (every perturbed pixel gets one of the corners of the color cube $[0,1]^c$). Then, at the i-th iteration, we randomly select $A \subset M$ and $B \subset U \setminus M$, with $|A| = |B| = \alpha^{(i)} \cdot k$, and create $M' = (M \setminus A) \cup B$. Δ' is formed by sampling random values from $\{0,1\}^c$ for the elements in B, i.e. those which were not perturbed at the previous iteration. The quantity $\alpha^{(i)}$ controls how much M' differs from M and decays following a predetermined piecewise constant schedule rescaled according to the maximum number of queries N. The schedule is completely determined by the single value α_{init}, used to calculate $\alpha^{(i)}$ for every iteration i, which is also the only free hyperparameter of our scheme. We provide details about the algorithm, schedule, and values of α_{init} in Sec. 6.2.10, and ablation studies for them in Sec. 6.2.11. For the feature based threat model each color channel is treated as a pixel and one applies the scheme above to the "gray-scale" image ($c = 1$) with three times as many "pixels".

6.2.3 Comparison of query efficiency of l_0-RS

We compare pixel-based l_0-`RS` to other black-box untargeted attacks in terms of success rate versus query efficiency. The results of targeted attacks are in Sec. 6.2.5. Here we focus on attacking normally trained VGG-16-BN and ResNet-50 models on ImageNet, which contains RGB images resized to shape 224×224, that is 50,176 pixels, belonging to 1,000 classes. We consider perturbations of size $k \in \{50, 150\}$ pixels to assess the effectiveness of the untargeted attacks at different thresholds with a limit of 10,000 queries. We evaluate the success rate on the initially correctly classified images out of 500 images from the validation set.

Competitors. Many existing black-box pixel-based l_0-attacks [Narodytska and Kasiviswanathan, 2017, Schott et al., 2019] do not aim at query efficiency and rather try to minimize the size of the perturbations. Among them, only CornerSearch (Sec. 6.1.2) and ADMM attack [Zhao et al., 2019] scale to ImageNet. However, CornerSearch requires $8 \times \#pixels$ queries only for the initial phase, exceeding the query limit we fix by more than 40 times. The ADMM attack tries to achieve a successful perturbation and then reduces its l_0-norm. Moreover, we introduce black-box versions of PGD_0 (Sec. 6.1.3) with the gradient estimated by finite difference approximation as done in prior work, e.g., see Ilyas et al. [2018]. As a strong baseline, we introduce JSMA-CE which is a version of the JSMA algorithm [Papernot et al., 2016b] that we adapt to the black-box setting: 1) for query efficiency, we estimate the gradient of the cross-entropy loss instead of gradients of *each* class logit, 2) on each iteration, we modify the *pixels* with the highest gradient contribution. More details about the attacks can be found in Sec. 6.2.5.

Results. We show in Fig. 6.12 the success rate vs the number of queries for all black-box attacks. In all cases, l_0-`RS` outperforms its competitors in terms of the final success rate by a large margin, e.g. the second best method (PGD_0 w/ GE) is at least 30% worse. Moreover,

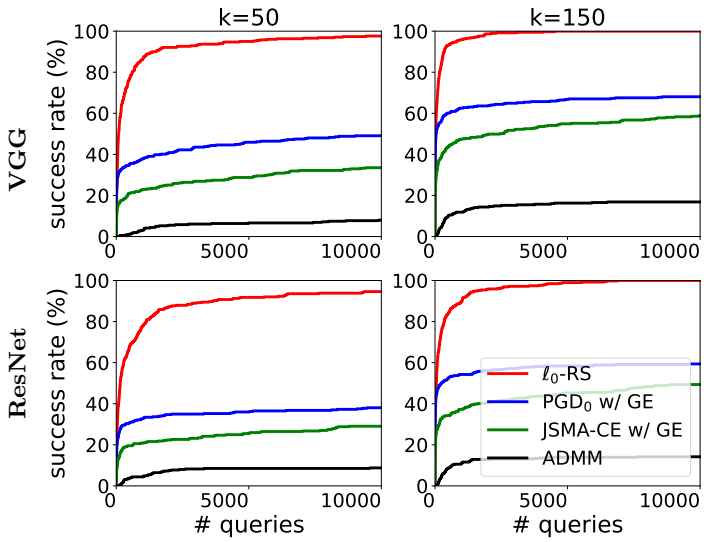

Figure 6.12: Progression of the success rate vs number of queries for black-box pixel-based l_0-attacks on ImageNet in the untargeted setting. At all sparsity levels l_0-RS (red) outperforms PGD$_0$ (blue) and JSMA-CE (green) with gradient estimation and ADMM attack (black).

Table 6.6: **ImageNet:** Robust test error of l_0-attacks. The entries with * are evaluated on 100 points instead of 500 because of their high computational cost. All black-box attacks use 10k queries except CornerSearch which uses 600k. l_0-RS outperforms all black- *and* white-box attacks.

attack	*type*	VGG	ResNet
l_0-bound in **pixel space** $k = 50$			
JSMA-CE	white-box	42.6%	39.6%
PGD$_0$	white-box	87.0%	81.2%
ADMM	black-box	30.3%	29.0%
JSMA-CE with GE	black-box	49.6%	44.8%
PGD$_0$ with GE	black-box	61.4%	51.8%
CornerSearch*	black-box	82.0%	72.0%
l_0-RS	black-box	**98.2%**	**95.8%**

attack	*type*	VGG	ResNet
l_0-bound in **feature space** $k = 50$			
SAPF*	white-box	21.0%	18.0%
ProxLogBarrier	white-box	33.0%	28.4%
EAD	white-box	39.8%	35.6%
SparseFool	white-box	43.6%	42.0%
VFGA	white-box	58.8%	55.2%
FMN	white-box	83.8%	77.6%
PDPGD	white-box	89.6%	87.2%
ADMM	black-box	32.6%	29.0%
CornerSearch*	black-box	76.0%	62.0%
l_0-RS	black-box	**92.8%**	**88.8%**

l_0-RS is query efficient as it achieves results close to the final ones already with a low number of queries. For example, on VGG with $k = 150$, l_0-RS achieves 100% of success rate using on average only 171 queries, with a median of 25. Unlike other methods, l_0-RS can achieve almost 100% success rate by perturbing 50 pixels which is *only* 0.1% of the total number of pixels. We visualize the adversarial examples of l_0-RS in Fig. 6.13.

6.2.4 Using l_0-RS for accurate robustness evaluation

In this section, our focus is the accurate evaluation of robustness in the l_0-threat model. For this, we evaluate existing white-box methods and black-box methods together. Instead of the success rate taken only over correctly classified examples, here we rather consider *robust error* (similarly to Madry et al. [2018]), which is defined as the classification error on the adversarial examples crafted by an attacker.

White-box attacks on ImageNet. We test the robustness of the ImageNet models introduced in the previous section to l_0-bounded perturbations. As competitors we consider multiple white-box attacks which minimize the l_0-norm in *feature space*: SAPF [Fan et al., 2020], ProxLogBarrier [Pooladian et al., 2020], EAD [Chen et al., 2018], SparseFool [Modas et al., 2019],

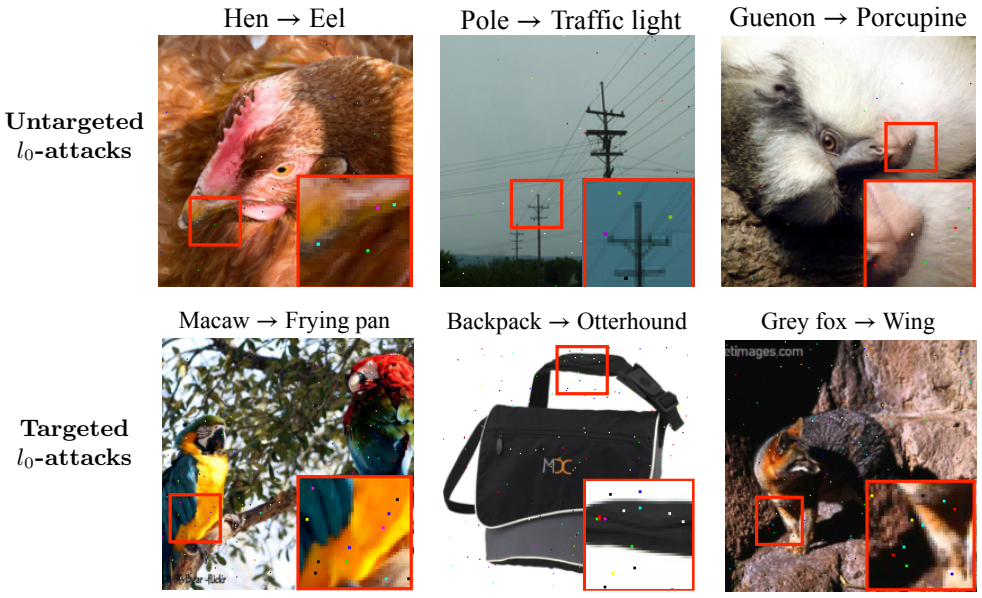

Figure 6.13: Untargeted ($k = 50$ pixels) and targeted ($k = 150$ pixels) l_0-adversarial examples generated by our pixel-based l_0-`RS` algorithm.

VFGA [Césaire et al., 2020], FMN [Pintor et al., 2021] and PDPGD [Matyasko and Chau, 2021]. For the l_0-threat model in *pixel space* we use two white-box baselines: PGD_0 and JSMA-CE [Papernot et al., 2016b] (where we use the gradient of the cross-entropy loss to generate the saliency map). Moreover, we show the results of the black-box attacks from the previous section (all with a query limit of 10,000), and additionally use the black-box CornerSearch for which we use a query limit of 600k and which is thus only partially comparable. Details of the attacks are available in Sec. 6.2.10. Table 6.6 shows the robust error given by all competitors: l_0-`RS` achieves the best results for pixel and feature based l_0-threat model on both VGG and ResNet, outperforming black- *and* white-box attacks. We note that while the PGD attack has been observed to give accurate robustness estimates for l_∞- and l_2-norms [Madry et al., 2018], this is not the case for the l_0 constraint set. This is due to the discrete structure of the l_0-ball which is not amenable for continuous optimization.

Table 6.7: **CIFAR-10:** Robust test error of untargeted l_0-attacks in pixel and feature space on a l_2- resp. l_1-AT model.

attack	*type*	l_2-AT	l_1-AT
l_0-bound in **pixel space** $k = 24$			
PGD_0	white-box	68.7%	72.7%
CornerSearch	black-box	59.3%	64.9%
l_0-`RS`	black-box	**85.7%**	**81.0%**
l_0-bound in **feature space** $k = 24$			
VFGA	white-box	40.5%	27.5%
FMN	white-box	52.9%	28.2%
PDPGD	white-box	46.4%	26.9%
CornerSearch	black-box	43.2%	29.4%
l_0-`RS`	black-box	**63.4%**	**38.3%**

Table 6.8: **MNIST:** Robust test error on robust models from Schott et al. [2019] on MNIST by different attacks on 100 test points.

attack	*type*	$k = 12$ (pixels) ABS	$k = 12$ (pixels) Binary ABS
Pointwise Attack	black-box	31%	23%
CornerSearch	black-box	29%	28%
l_0-`RS`	black-box	**55%**	**51%**

Comparison on CIFAR-10. In Table 6.7 we compare the strongest white- and black-box attacks on l_1- resp. l_2-adversarially trained PreAct ResNet-18 on CIFAR-10 from Sec. 4.4 and Rebuffi et al. [2021b]. We keep the same computational budget used on ImageNet. As before, we consider perturbations with l_0-norm $k = 24$ in pixel or feature space: in both cases l_0-RS achieves the highest robust test error outperforming even all white-box attacks. Note that, as expected, the model robust wrt l_1 is less vulnerable to l_0-attacks especially in the feature space, whose l_1-norm is close to that used during training.

Robust generative models on MNIST. Schott et al. [2019] propose two robust generative models on MNIST, ABS and Binary ABS, which showed high robustness against multiple types of l_p-bounded adversarial examples. These classifiers rely on optimization-based inference using a variational auto-encoder (VAE) with 50 steps of gradient descent for each prediction (times 1,000 repetitions). It is too expensive to get gradients with respect to the input through the optimization process, thus Schott et al. [2019] evaluate only black-box attacks, and test l_0-robustness with sparsity $k = 12$ using their proposed Pointwise Attack with 10 restarts. We evaluate on both models CornerSearch with a budget of 50,000 queries and l_0-RS with an equivalent budget of 10,000 queries and 5 random restarts. Table 6.8 summarizes the robust test error (on 100 points) achieved by the attacks (the results of Pointwise Attack are taken from Schott et al. [2019]). For both classifiers, l_0-RS yields the strongest evaluation of robustness suggesting that the ABS models are less robust than previously believed. This illustrates that despite we have *full access* to the attacked VAE model, a strong *black-box* l_0-attack can still be useful for an accurate robustness evaluation.

6.2.5 Additional results

Targeted attacks. Here we test l_0-bounded attacks in the targeted scenario using the sparsity level $k = 150$ (maximum number of perturbed pixels or features) on ImageNet. For black-box attacks including l_0-RS, we set the query budget to $100,000$, and show the success rate on VGG and ResNet in Table 6.9 computed on 500 points. We also give additional budget to white-box attacks compared to the untargeted case (see details in Sec. 6.2.10). The target class for each point is randomly sampled from the set of labels which excludes the true label. We report the success rate in Table 6.9 instead of robust error since we are considering targeted attacks where the overall goal is to cause a misclassification towards a particular class instead of an arbitrary misclassification. We can see that targeted attacks are more challenging than untargeted attacks for many methods, particularly for methods like CornerSearch that were designed primarily for untargeted attacks. When considering the pixel space, l_0-RS outperforms both black- and white-box attacks, while in the feature space it is second only to PDPGD although achieving similar results on the VGG model. Additionally, we report the query efficiency curves in Fig. 6.14 for l_0-RS, PGD_0 and JSMA-CE with gradient estimation. We omit the ADMM attack since it has shown the lowest success rate in the untargeted setting. We can see that l_0-RS achieves a high success rate (90%/80% for VGG/ResNet) even after 20,000 queries. At the same time, the targeted setting is very challenging for other methods that have nearly 0% success rate even after 100,000 queries. Finally, we visualize targeted l_0 adversarial examples generated by pixel-based l_0-RS in Fig. 6.13 (bottom row).

Additional statistics. In Table 6.10, we report the details of success rate and query efficiency of the black-box attacks on ImageNet for sparsity levels $k \in \{50, 150\}$ in the pixel space and 10,000/100,000 query limit for untargeted/targeted attacks respectively.

l_0-RS on malware detection. In the following we apply l_0-RS in a different domain, i.e. malware detection, showing its versatility. We consider the Drebin dataset [Arp et al., 2014] which consists of $129,013$ Android applications, among which $123,453$ are benign and $5,560$ malicious, with $d = 545,333$ features divided into 8 families. Data points are represented by a binary vector $x \in \{0,1\}^d$ indicating whether each feature is present or not in x (unlike image classification the input space is in this case discrete). As Grosse et al. [2016], we restrict the attacks to only adding features from the first 4 families, that is modifications to the manifest of the applications, to preserve the functionality of the samples (no feature present in the clean data

Table 6.9: Success rate of targeted l_0-attacks on ImageNet. The entry with * is evaluated on 100 points instead of 500 because of their high computational cost. All black-box attacks use 100k queries except CornerSearch which uses 450k. l_0-RS outperforms all black- *and* white-box attacks in the pixel space scenario, and is competitive with white-box attacks in the feature space one.

attack	*type*	VGG	ResNet
l_0-bound in **pixel space** $k = 150$			
JSMA-CE	white-box	0.4%	1.4%
PGD_0	white-box	62.8%	67.8%
JSMA-CE w/ GE	black-box	0.0%	0.0%
PGD_0 w/ GE	black-box	0.4%	1.4%
CornerSearch*	black-box	3.0%	2.0%
l_0-RS	black-box	**98.2%**	**95.6%**

attack	*type*	VGG	ResNet
l_0-bound in **feature space** $k = 150$			
FMN	white-box	73.0%	79.8%
PDPGD	white-box	**92.8%**	**94.2%**
l_0-RS	black-box	90.8%	80.0%

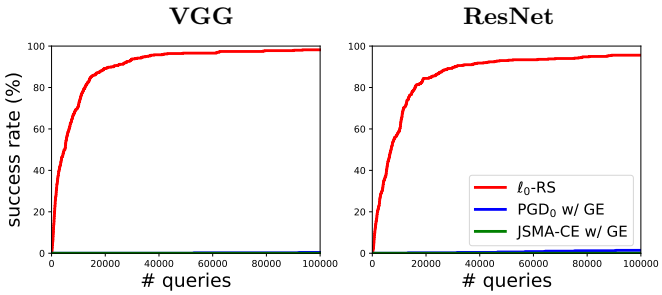

Figure 6.14: Success rate over queries for targeted black-box attacks in the pixel-based l_0-threat model with $k = 150$.

is removed), which leaves a maximum of 233,727 alterable dimensions. We trained the classifier, which has 1 fully-connected hidden layer with 200 units and uses ReLU as activation function, with 20 epochs of SGD minimizing the cross-entropy loss, with a learning rate of 0.1 reduced by a factor of 10 after 10 and 15 epochs. We use batches of size 2000, consisting of 50% of benign and 50% of malicious examples. For training, we merged the training and validation sets of one of the splits provided by Arp et al. [2014]. It achieves a test accuracy of 98.85%, with a false positive rate of 1.01%, and a false negative rate of 4.29%.

We apply l_0-RS as described at the beginning of Sec. 6.2.2 for images as input by setting $h = d$ and $w = c = 1$ such that $x \in \{0, 1\}^{d \times 1 \times 1}$. Only adding features means that all values in Δ equal 1, thus $\Delta' \equiv \Delta$ at every iteration (step 7 in Alg. 11) and only the set M is updated. For our attack we use $\alpha_{\text{init}} = 1.6$ and the same schedule of $\alpha^{(i)}$ of l_0-RS on image classification tasks (see Sec. 6.2.10). Grosse et al. [2016] successfully fooled similar models with a variant of the white-box JSMA [Papernot et al., 2016b], and Podschwadt and Takabi [2019] confirm that it is the most effective technique on Drebin, compared to the attacks of Stokes et al. [2017], Al-Dujaili et al. [2018], Hu and Tan [2017] including adaptations of FGSM and PGD. We use JSMA and PGD_0 in a black-box version with gradient estimation (details below). Liu et al. [2019] propose a black-box genetic algorithm with prior knowledge of the importance of the features for misclassification (via a pretrained random forest) which is not comparable to our threat model.

We compare the attacks allowed to modify $k \in \{80, 160\}$ features and with a maximum of 10000 queries. Fig. 6.15 shows the progression of success rate (computed on the initially correctly classified apps) of the attacks over number of queries used. At both sparsity levels, our l_0-RS attack (in red) achieves very high success rate and outperforms JSMA (green) and PGD_0 (blue) in the low query regime, where the approximation of the gradient is likely not sufficiently precise to identify the most relevant feature for the attacker to perturb.

6.2.6 Theoretical analysis of l_0-RS

Given the empirical success of l_0-RS, here we analyze it theoretically for a binary classifier. While the analysis does not directly transfer to neural networks, most modern neural network architectures result in piecewise linear classifiers [Arora et al., 2018], so that the result approximately

Table 6.10: We report the success rate and statistics about query consumption of black-box l_0-bounded attacks in the pixel space on ImageNet. We consider sparsity levels $k \in \{50, 150\}$ and compute the average and median number of queries needed to generate adversarial examples on all correctly classified points.

			VGG			ResNet		
	pixels modified	*attack*	*success rate*	*mean queries*	*median queries*	*success rate*	*mean queries*	*median queries*
untarg.	$k = 50$	ADMM	8.3%	9402	10000	8.8%	9232	10000
		JSMA-CE w/ GE	33.5%	7157	10000	29.0%	7509	10000
		PGD_0 w/ GE	49.1%	5563	10000	38.0%	6458	10000
		l_0-RS	**97.6%**	**737**	**88**	**94.6%**	**1176**	**150**
	$k = 150$	ADMM	16.9%	8480	10000	14.2%	8699	10000
		JSMA-CE w/ GE	58.8%	4692	2460	49.4%	5617	10000
		PGD_0 w/ GE	68.1%	3447	42	59.4%	4263	240
		l_0-RS	**100%**	**171**	**25**	**100%**	**359**	**49**
targeted	$k = 150$	JSMA-CE w/ GE	0.0%	100000	100000	0.0%	100000	100000
		PGD_0 w/ GE	0.4%	99890	100000	1.4%	99357	100000
		l_0-RS	**98.2%**	**9895**	**4914**	**95.6%**	**14470**	**6960**

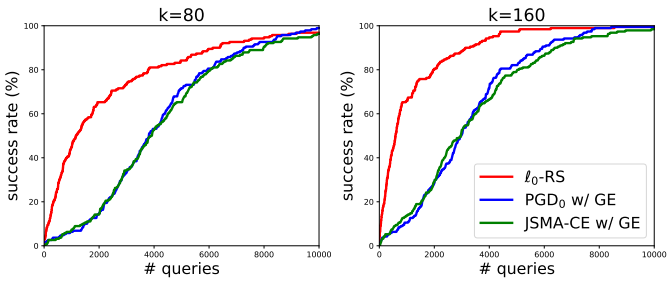

Figure 6.15: Success rate vs number of queries in fooling a malware detector for sparsity levels $k \in \{80, 160\}$. l_0-RS outperforms the competitors, with a large gap in the low query regime.

holds in a sufficiently small neighborhood of the target point x.

As in the malware detection task in Sec. 6.2.5, we assume that the input x has binary features, $x \in \{0,1\}^d$, and we denote the label by $y \in \{-1, 1\}$ and the gradient of the linear model by $w_x \in \mathbb{R}^d$. Then the Problem (6.8) of finding the optimal l_0 adversarial example is equivalent to:

$$\underset{\substack{\|\delta\|_0 \leq k \\ x_i + \delta_i \in \{0,1\}}}{\arg\min}\ y \langle w_x, x + \delta \rangle = \underset{\substack{\|\delta\|_0 \leq k \\ \delta_i \in \{0, 1-2x_i\}}}{\arg\min} \langle y w_x, \delta \rangle = \underset{\substack{\|\delta\|_0 \leq k \\ \delta_i \in \{0,1\}}}{\arg\min} \langle \underbrace{y w_x \odot (1 - 2x)}_{\hat{w}_x}, \delta \rangle,$$

where $\odot$ denotes the elementwise product. In the white-box case, i.e. when w_x is known, the solution is to simply set $\delta_i = 1$ for the k smallest weights of $\hat{w}_x$. The black-box case, where w_x is unknown and we are only allowed to query the model predictions $\langle \hat{w}_x, z \rangle$ for any $z \in \mathbb{R}^d$, is more complicated since the naive weight estimation algorithm requires $O(d)$ queries to first estimate $\hat{w}_x$ and then to perform the attack by selecting the k minimal weights. This naive approach is prohibitively expensive for high-dimensional datasets (e.g., $d = 150{,}528$ on ImageNet assuming $224 \times 224 \times 3$ images). However, the problem of generating adversarial examples does not have to be always solved exactly, and often it is enough to find an approximate solution. Therefore we can be satisfied with only identifying k among the m smallest weights. Indeed, the focus is not on exactly identifying the solution but rather on having an algorithm that in expectation requires a *sublinear* number of queries. With this goal, we show that l_0-RS satisfies this requirement for large enough m.

Proposition 6.2.1 *The expected number t_k of queries needed for l_0-RS with $\alpha^{(i)} = 1/k$ to find a set of k weights out of the smallest m weights of a linear model is:*

$$\mathbb{E}[t_k] = (d-k)k \sum_{i=0}^{k-1} \frac{1}{(k-i)(m-i)} < (d-k)k \frac{\ln(k) + 2}{m-k}.$$

Proof. According to the l_0-RS algorithm, we have d features grouped in a set U and the goal is to find a set $M \subset U$ containing k elements among the m smallest elements of U. Since $\alpha^{(i)} = 1/k$, at every iteration we pick one element $p \in M$ to remove from M and one element $q \in U \setminus M$ to add to M. This results in a binary vector $z_{new} \in \{0,1\}^d$ indicating which are the features in M. Then we query the black-box linear model to determine whether the loss at a new point z_{new} improves compared to the point on the previous iteration $z_{current}$, i.e. $L(z_{new}) < L(z_{current})$, which gives us information whether $q < p$.

If the current set M contains i elements which belong to the smallest m, the probability of increasing it to $i+1$ elements with the next pick is equal to

$$\mathcal{P}[i \to i+1] = \mathcal{P}[p \notin \text{smallest } m, q \in \text{smallest } m] = \frac{k-i}{k} \cdot \frac{m-i}{d-k}.$$

Then the expected number of iterations for the step $i \to i+1$ is equal to

$$\mathbb{E}[t_{i+1} - t_i] = \frac{(d-k)k}{(k-i)(m-i)}$$

since all the draws are independent. Assuming that we begin with none of the smallest m elements in M, the expected number of iterations t_k needed to find a set of k weights out of the smallest m weights is given by

$$\mathbb{E}[t_k] = \sum_{i=0}^{k-1} \mathbb{E}[t_{i+1} - t_i] = \sum_{i=0}^{k-1} \frac{(d-k)k}{(k-i)(m-i)} = (d-k)k \sum_{i=0}^{k-1} \frac{1}{(k-i)(m-i)}. \tag{6.10}$$

Now assuming that $m > k$, we can write the summation as:

$$\begin{aligned} \sum_{i=0}^{k-1} \frac{1}{(k-i)(m-i)} &= \sum_{j=1}^{k} \frac{1}{j(j+m-k)} = \frac{1}{m-k} \sum_{j=1}^{k} \left(\frac{1}{j} - \frac{1}{j+m-k} \right) \\ &= \frac{1}{m-k} \left(H_k - H_m + H_{m-k} \right), \end{aligned} \tag{6.11}$$

where H_k is the k-th harmonic number. Using the fact that $\ln(k) < H_k \leq \ln(k) + 1$, we have

$$H_k - H_m + H_{m-k} < \ln(k) - \ln(m) + \ln(m-k) + 2$$

and

$$\ln(k) - \ln(m) + \ln(m-k) + 2 < \ln(k) + 2.$$

If we combine this result with Eq. (6.10) and Eq. (6.11), we obtain the desired upper bound on $\mathbb{E}[t_k]$:

$$\mathbb{E}[t_k] < (d-k)k \frac{\ln(k)+2}{m-k}.$$

□

For non-linear models, l_0-RS uses $\alpha^{(i)} > 1/k$ for better exploration initially, but then progressively reduces it. The main conclusion from Proposition 6.2.1 is that $\mathbb{E}[t_k]$ becomes sublinear for large enough gap $m-k$, as we illustrate in Fig. 6.16.

Remark 6.2.1 *We further remark that for $m = k$ we get that*

$$\mathbb{E}[t_k] = (d-k)k \sum_{j=1}^{k} \frac{1}{j^2} < \frac{\pi^2}{6}(d-k)k,$$

however we are interested in the setting when the gap $m-k$ is large enough so that $\mathbb{E}[t_k]$ becomes sublinear.

The main conclusion from Proposition 6.2.1 is that $\mathbb{E}[t_k]$ becomes sublinear for large enough gap $m-k$ which we further illustrate in Fig. 6.16 where we plot the expected number of queries needed by l_0-RS for $d = 150,528$ and $k = 150$ which is equivalent to our ImageNet experiments with 50 pixels perturbed.

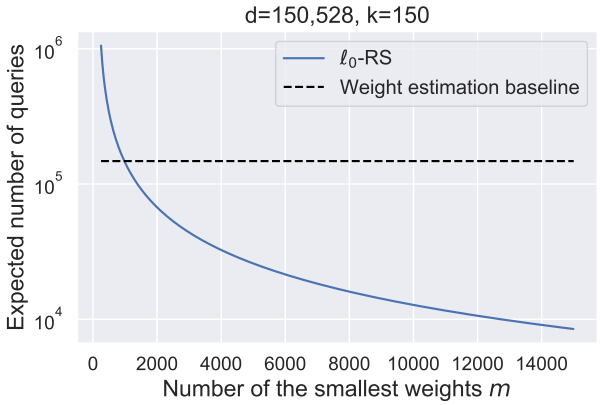

Figure 6.16: Comparison of the query efficiency of l_0-`RS` to that of the naive weight estimation baseline which requires $O(d)$ queries to estimate the gradient.

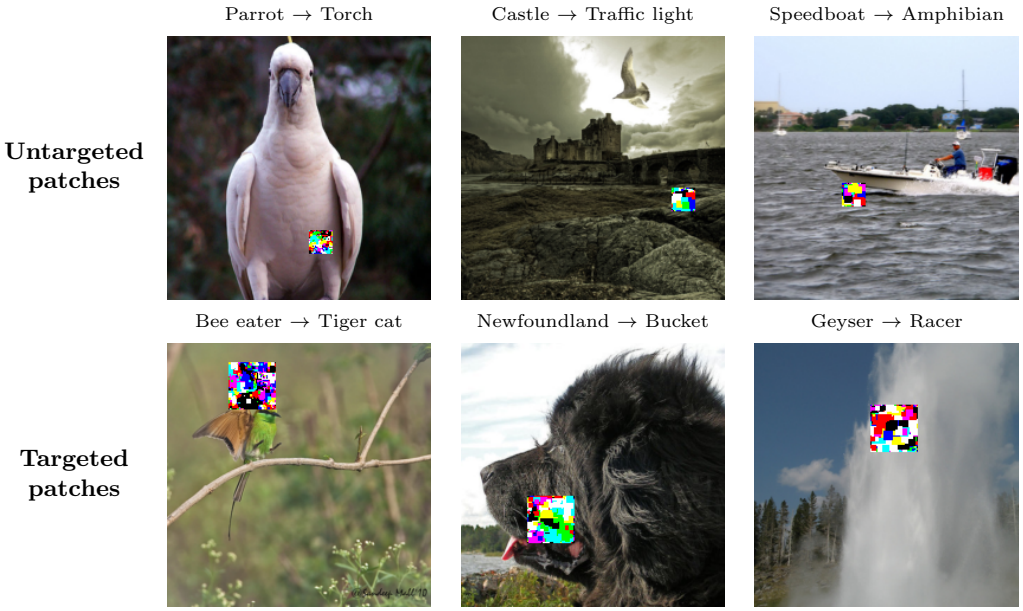

Figure 6.17: Image-specific untargeted (20×20 pixels) and targeted (40×40) patches generated by our `Patch-RS` algorithm.

6.2.7 `Sparse-RS` for adversarial patches

Another type of sparse attacks which recently received attention are adversarial patches introduced by Brown et al. [2017]. There the perturbed pixels are localized, often square-shaped, and limited to a small portion of the image but can be changed in an arbitrary way (see examples in Fig. 6.17). While some works Brown et al. [2017], Karmon et al. [2018] aim at universal patches, which fool the classifier regardless of the image and the position where they are applied, which we consider in Sec. 6.2.9, we focus first on image-specific patches as in Yang et al. [2020a], Rao et al. [2020] where one optimizes both the content and the location of the patch for each image independently.

General algorithm for patches. Note that both location (step 6 in Alg. 11) and content (step 7 in Alg. 11) have to be optimized in `Sparse-RS`, and on each iteration we check only one of these updates. We test the effect of different frequencies of location/patch updates in an ablation study in Sec. 6.2.11. Since the location of the patch is a discrete variable, random search is particularly well suited for its optimization. For the location updates in step 6 in Alg. 11, we randomly sample a new location in a 2D l_∞-ball around the current patch position (using

Table 6.11: Success rate and query statistics of image-specific patches. Black-box attacks are given 10k/50k queries for untargeted/targeted case. SH is a deterministic method. The query statistics are computed on *all* images with 5 random seeds. * TPA uses an early stopping mechanism to save queries, thus might not use all queries. `Patch-RS` outperforms all other methods in success rate and query efficiency.

	attack	VGG			ResNet		
		succ. rate	*mean qr.*	*med. qr.*	*succ. rate*	*mean qr.*	*med. qr.*
untargeted, 20×20 pixels							
black-box	LOAP w/ GE	$55.1\%_{\pm 0.6}$	$5879_{\pm 51}$	$7230_{\pm 377}$	$40.6\%_{\pm 0.1}$	$6870_{\pm 10}$	$10000_{\pm 0}$
black-box	TPA	$46.1\%_{\pm 1.1}$	$6085^{*}_{\pm 83}$	$8080^{*}_{\pm 1246}$	$49.0\%_{\pm 1.2}$	$5722^{*}_{\pm 64}$	$5280^{*}_{\pm 593}$
black-box	`Sparse-RS` + SH	82.6%	2479	514	75.3%	3290	549
black-box	`Sparse-RS` + SA	$85.6\%_{\pm 1.1}$	$2367_{\pm 83}$	$533_{\pm 40}$	$78.5\%_{\pm 1.0}$	$2877_{\pm 64}$	$458_{\pm 43}$
black-box	`Patch-RS`	**87.8%$_{\pm 0.7}$**	**2160$_{\pm 44}$**	**429$_{\pm 22}$**	**79.5%$_{\pm 1.4}$**	**2808$_{\pm 89}$**	**438$_{\pm 68}$**
	White-box LOAP	98.3%	-	-	82.2%	-	-
targeted, 40×40 pixels							
black-box	LOAP w/ GE	$23.9\%_{\pm 0.9}$	$44134_{\pm 71}$	$50000_{\pm 0}$	$18.4\%_{\pm 0.9}$	$45370_{\pm 88}$	$50000_{\pm 0}$
black-box	TPA	$5.1\%_{\pm 1.2}$	$29934^{*}_{\pm 462}$	$34000^{*}_{\pm 0}$	$6.0\%_{\pm 0.5}$	$31690^{*}_{\pm 494}$	$34667^{*}_{\pm 577}$
black-box	`Sparse-RS` + SH	63.6%	25634	19026	48.6%	31250	50000
black-box	`Sparse-RS` + SA	$70.9\%_{\pm 1.2}$	$23749_{\pm 346}$	$15569_{\pm 568}$	$53.7\%_{\pm 0.9}$	$32290_{\pm 239}$	$40122_{\pm 2038}$
black-box	`Patch-RS`	**72.7%$_{\pm 0.9}$**	**22912$_{\pm 207}$**	**14407$_{\pm 866}$**	**55.6%$_{\pm 1.5}$**	**30290$_{\pm 317}$**	**34775$_{\pm 2660}$**
	White-box LOAP	99.4%	-	-	94.8%	-	-

clipping so that the patch is fully contained in the image). The radius of this l_∞-ball shrinks with increasing iterations in order to perform progressively more local optimization (see Sec. 6.2.10 for details). For the update of the patch itself in step 7 in Alg. 11, the only constraints are given by the input domain $[0, 1]^d$. Thus in principle any black-box method for an l_∞-threat model can be plugged in there. We use Square Attack (SA) (Sec. 4.2) and SignHunter (SH) [Al-Dujaili and O'Reilly, 2020] as they represent the state-of-the-art in terms of success rate and query efficiency. We integrate both in our framework and refer to them as `Sparse-RS` + SH and `Sparse-RS` + SA. Next we propose a novel random search based attack motivated by SA which together with our location update yields our novel `Patch-RS` attack.

`Patch-RS` algorithm. While SA and SH are state-of-the-art for l_∞-attacks, they have been optimized for rather small perturbations whereas for patches all pixels can be manipulated arbitrarily in $[0, 1]$. Here, we design an initialization scheme and a sampling distribution specific for adversarial patches. As initialization (step 2 of Alg. 11), `Patch-RS` uses randomly placed squares with colors in $\{0, 1\}^3$, then it samples updates of the patch (step 7) with shape of squares, of size decreasing according to a piecewise constant schedule, until a refinement phase in the last iterations, when it performs single-channel updates (exact schedule in Sec. 6.2.10). This is in contrast to SA where random vertical stripes are used as initialization and always updates for all three channels of a pixel are sampled. The ablation study in Sec. 6.2.11 shows how both modifications contribute to the improved performance of `Patch-RS`.

Experiments. In addition to `Sparse-RS` + SH, `Sparse-RS` + SA, and `Patch-RS`, we consider two existing methods: i) TPA [Yang et al., 2020a] which is a black-box attack aiming to produce image-specific adv. patches based on reinforcement learning. While Yang et al. [2020a] allows multiple patches for an image, we use TPA in the standard setting of a single patch, ii) Location-Optimized Adversarial Patches (LOAP) [Rao et al., 2020], a white-box attack that uses PGD for the patch updates, which we combine with gradient estimation in order to use it in the black-box scenario (see Sec. 6.2.10 for details). In Table 6.11 we report success rate, mean and median number of queries used for untargeted attacks with patch size 20×20 and query limit of 10,000 and for targeted attacks (random target class for each image) with patch size 40×40 and maximally 50,000 queries. We attack 500 images of ImageNet with VGG and ResNet as target models. The query statistics are computed on *all* 500 images, i.e. without restricting to only successful

Table 6.12: Success rate and query statistics of image-specific frames computed for 5 seeds. Black-box attacks are given 10k/50k queries for untargeted/targeted case. SH is a deterministic method. `Frame-RS` achieves the best success rate and efficiency in all settings.

	attack	VGG			ResNet		
		succ. rate	*mean qr.*	*med. qr.*	*succ. rate*	*mean qr.*	*med. qr.*
	untargeted, 2 pixels wide						
black-box	AF w/ GE	$57.6\%_{\pm 0.4}$	$5599_{\pm 23}$	$6747_{\pm 127}$	$70.6\%_{\pm 0.7}$	$4568_{\pm 13}$	$3003_{\pm 116}$
black-box	`Sparse-RS` + SH	79.9%	3169	882	88.7%	2810	964
black-box	`Sparse-RS` + SA	$77.8\%_{\pm 2.0}$	$3391_{\pm 215}$	$1292_{\pm 220}$	$80.7\%_{\pm 1.4}$	$3073_{\pm 111}$	$1039_{\pm 93}$
black-box	`Frame-RS`	$\mathbf{90.2\%_{\pm 0.3}}$	$\mathbf{2366_{\pm 26}}$	$\mathbf{763_{\pm 32}}$	$\mathbf{94.0\%_{\pm 0.3}}$	$\mathbf{1992_{\pm 34}}$	$\mathbf{588_{\pm 10}}$
	White-box AF	100%	-	-	100%	-	-
	targeted, 3 pixels wide						
black-box	AF w/ GE	$16.1\%_{\pm 1.5}$	$45536_{\pm 328}$	$50000_{\pm 0}$	$40.6\%_{\pm 1.4}$	$39753_{\pm 459}$	$50000_{\pm 0}$
black-box	`Sparse-RS` + SH	52.0%	32223	41699	73.2%	25929	20998
black-box	`Sparse-RS` + SA	$32.9\%_{\pm 1.9}$	$38463_{\pm 870}$	$50000_{\pm 0}$	$47.5\% \pm 2.8$	$33321_{\pm 980}$	$48439_{\pm 3500}$
black-box	`Frame-RS`	$\mathbf{65.7\%_{\pm 0.4}}$	$\mathbf{28182_{\pm 99}}$	$\mathbf{27590_{\pm 580}}$	$\mathbf{88.0\%_{\pm 0.9}}$	$\mathbf{19828_{\pm 130}}$	$\mathbf{15325_{\pm 370}}$
	White-box AF	100%	-	-	100%	-	-

adversarial examples, as this makes the query efficiency comparable for different success rates. Our `Sparse-RS` + SH, `Sparse-RS` + SA and `Patch-RS` outperform existing methods by a large margin, showing the effectiveness of our scheme to optimize both location and patch. Among them, our specifically designed `Patch-RS` achieves the best results in all metrics. We visualize its resulting adversarial examples in Fig. 6.17.

6.2.8 `Sparse-RS` for adversarial frames

Adversarial frames introduced in Zajac et al. [2019] are another sparse threat model where the attacker is allowed to perturb only pixels along the borders of the image. In this way, the total number of pixels available for the attack is small (3%-5% in our experiments), and in particular the semantic content of the image is not altered by covering features of the correct class. Thus, this threat model shows how even peripheral changes can influence the classification.

General algorithm for frames. Unlike patches (Sec. 6.2.7), for frames the set of pixels M which can be perturbed (see Alg. 11) is fixed. However, similarly to patches, the pixels of the frame can be changed arbitrarily in $[0,1]^3$ and thus we can use l_∞-black-box attacks for step 7 in Alg. 11 which we denote as `Sparse-RS` + SH and `Sparse-RS` + SA. While SH is agnostic to the shape of the perturbations defined by adversarial frames and can be successfully applied for step 7 without further modifications, `Sparse-RS` + SA leads to suboptimal performance (see Table 6.12). Indeed, in the frame setting, the sampling distribution of SA requires its square updates to be entirely contained in the frame which due to its small width strongly limits the potential updates. Thus, to achieve better performance, we propose a new sampling distribution specific to frames which, combined with `Sparse-RS` framework, we call `Frame-RS`.

`Frames-RS` algorithm. Our new sampling distribution (used for step 7 of Alg. 11) returns squares that only *partially* overlap with the frame. Effectively, we have more flexible updates, often of non-squared shape. Moreover, similarly to `Patch-RS`, we introduce single-channel updates in the final phase. We also use a different initialization in step 2: uniformly random sampling instead of vertical stripes. As a result, `Frame-RS` shows significantly better results than both `Sparse-RS` + SH and `Sparse-RS` + SA. We initialize the components of the frame with random values in $\{0,1\}$. Then, at each iteration i we sample uniformly a pixel of the frame to be a random corner of a square of size $s^{(i)} \times s^{(i)}$: all the pixels in the intersection of such square with the frame are set to the same randomly chosen color in $\{0,1\}^c$ to create the candidate update. The length of the side of the square is regulated by a linearly decreasing schedule ($s^{(i)} = 3\lceil \alpha_{\text{init}} \cdot w^2 \cdot (0.5 - i/N) \rceil$, with N the total number of queries available and w the width

Figure 6.18: Image-specific untargeted (2 pixels wide) and targeted (3 pixels) frames generated by our `Frame-RS` algorithm.

of the frame) for the first 50% of queries, after which $s^{(i)} = 1$. Finally, for the last 75% of iterations, only single-channel updates are performed. Similarly to the other threat models, the only hyperparameter is α_{init}.

Competitors. Similarly to the case of patches, we can design `Sparse-RS` + SH and `Sparse-RS` + SA for frames: SignHunter can be directly used to generate adversarial frames as this method does not require a particular shape of the perturbation set. Square Attack is specifically designed for rectangular images and we adapted it to this setting by constraining the sampled updates to lay inside the frame, as they are contained in the image for the l_∞- and l_2- threat models. Since the perturbation set has a different shape than in the other threat models a new schedule for the size of squares specific for this case: at iteration i of N and with frames of width w, we have $s^{(i)} = \lceil \alpha_{\text{init}} \cdot w \cdot (0.5 - i/N) \rceil$ until $i < N/2$, then $s^{(i)} = 1$ (we set $\alpha_{\text{init}} = 2$).

Moreover, Zajac et al. [2019] propose adversarial framing (AF), a method based on gradient descent to craft universal adversarial frames. We here use standard PGD on the cross-entropy loss to generate image-specific perturbations by updating only the components corresponding to the pixels belonging to the frame. In details, we use 1000 iterations with step size 0.05 using the sign of the gradient as direction and each element is projected on the interval $[0, 1]$. This white-box attack achieves 100% of success in all settings we tested (consider that 1176 and 2652 pixels be perturbed for the untargeted and targeted scenarios respectively). For the black-box version, we use the gradient estimation technique as for patches, that is Alg. 12 with the current iterate as input x, $m = 5$, $\eta = 10/\sqrt{d}$ and the gradient estimated at the previous iteration as prior p, where d represents the number of features in the frame. Moreover, we use step size 0.1 for the untargeted case, 0.01 for the targeted one. Note that we found the mentioned values via grid search to optimize the success rate. We iterate the procedure, which costs 10 queries of the classifier, until reaching the maximum number of allowed queries or finding a successful adversarial perturbation.

Experimental evaluation. For untargeted attacks we use frames of width 2 and a maximum of 10,000 queries, and for targeted attacks we use frames of width 3 and maximally 50,000 queries. In Table 6.12 we show that our `Frame-RS` achieves the best success rate (at least 10% higher than the closest competitor) and much better query efficiency in all settings. In particular, `Frame-RS` significantly outperforms `Sparse-RS` + SA, showing that our new sampling distribution is crucial to obtain an effective attack. For the untargeted case the success rate of our black-box `Frame-RS`

is close to that of the white-box AF method which achieves 100% success rate. Finally we illustrate some examples of the resulting images in Fig. 6.18.

6.2.9 Targeted universal attacks: patches and frames

A challenging threat model is that of a black-box, targeted universal adversarial patch attack where the classifier should be fooled into a chosen target class when the patch is applied inside any image of some other class. Previous works rely on transfer attacks: in Brown et al. [2017] the universal patch is created using a white-box attack on surrogate models, while the white-box attack of Karmon et al. [2018] directly optimizes the patch for the target model on a set of training images and then only tests generalization to unseen images. Our goal is a targeted black-box attack which crafts universal patches that generalize to unseen images when applied at random locations (see examples in Fig. 6.19). Similarly, one can consider black-box targeted universal frames, which are however applied at a fixed position (the borders of the images), see Fig. 6.20. To our knowledge, this is the first method for this threat model which does not rely on a surrogate model.

`Sparse-RS` for targeted universal attacks. Targeted universal attacks have to generalize to unseen images and, in the case of patches, random locations on the image. We propose to generate them in a black-box scenario with the following scheme, with a budget of 100k iterations. We select a small batch of training images $\{x_i\}_{i=1}^n$ ($n = 30$) and apply the patch at random positions on them (for frames the location is determined by its width). These positions are kept fixed for 10k iterations during which the patch is updated (step 7 of Alg. 11) with some black-box attack (we use the same as for image-specific attacks in Sec. 6.2.7 and Sec. 6.2.8) to minimize the loss

$$L_{\text{targ}}(\{x_i\}_{i=1}^n) = \sum_{i=1}^{n} L_{\text{CE}}(f(x_i), t), \tag{6.12}$$

where t is the target class. Then, to foster generalization, we resample both the batch of training images and the locations of the patch over them (step 6 in Alg. 11). In this way it is possible obtain black-box attacks without relying on surrogate models. Note that 30 queries of the classifiers are performed at every iteration of the algorithm.

In this scheme, as mentioned, we integrate either SignHunter or Square Attack to have `Sparse-RS` + SH and `Sparse-RS` + SA, and our novel `Patch-RS` introduced for image-specific attacks. In the case of patches, for `Sparse-RS` + SA we set $p = 0.05$ and for `Patch-RS` $\alpha_{\text{init}} = 0.05$ (recall that both parameters control the schedule for sampling the updates in a similar way for the two attacks, details in Sec. 6.2.10). For frames, for SA we use the schedule for image-specific attacks (Sec. 6.2.8) with $\alpha_{\text{init}} = 0.7$, while in `Frame-RS` we adapt the schedule for image-specific attacks to the specific case, so that the size of the squares decrease to $s^{(i)} = 1$ at $i = 25,000$ and annel updates are sampled from $i = 62,500$.

Transfer-based attacks. We evaluate the performance of targeted universal patches and frames generated by PGD [Madry et al., 2018] and MI-FGSM [Dong et al., 2018] on a surrogate model and then transferred to attack the target model. In particular, at every iteration of the attack we apply on the training images (200) the patch at random positions (independently picked for every image), average the gradients of the cross-entropy loss at the patch for all images, and take a step in the direction of the sign of the gradients (we use 1000 iterations, step size of 0.05 and momentum coefficient 1 for MI-FGSM). For patches, we report the success rate as the average over 500 images, 10 random target classes, and 100 random positions of the patches for each image. For frames, we average over 5000 images and the same 10 random target classes. In Table 6.13 we show the success rate of the transfer based attacks on the surrogate model but with images unseen when generating the perturbation and on the target model (we use VGG and ResNet alternatively as the source and target models). We see that the attacks achieve a very high success rate when applied to new images on the source model, but are not able to generalize to another architecture. We note that Brown et al. [2017] also report a similarly low success rate for the same size of the patch (50×50 which corresponds to approximately 5% of the input image, see Fig. 3 in Brown et al. [2017]) on the target class they consider. Finally, we

Table 6.13: Success rate of transfer-based targeted universal patches, with VGG (V) and ResNet (R) used as either source or target model averaged over 10 classes. When source and target model are the same network, the attack is only used on unseen images. These targeted universal attacks are very effective when transferred to unseen images on the source model, but do not generalize to the other architecture.

	attack	*succ. rate*			
		V → V	V → R	R → R	R → V
patches	PGD	99.6%	0.0%	94.9%	3.3%
	MI-FGSM	99.1%	0.0%	92.6%	1.3%
frames	PGD	99.4%	0.0%	99.8%	0.0%
	MI-FGSM	99.3%	0.0%	99.7%	0.0%

Table 6.14: Average success rate of targeted universal attacks over 10 target classes on VGG and ResNet. We use patches of size 50×50 and frames of width 6 pixels and repeat the attacks based on random search with 3 random seeds.

attack	**patches**		**frames**	
	VGG	ResNet	VGG	ResNet
Transfer PGD	3.3%	0.0%	0.0%	0.0%
Transfer MI-FGSM	1.3%	0.0%	0.0%	0.0%
PGD w/ GE	35.1%	15.5%	22.7%	52.4%
ZO-AdaMM	45.8%	6.0%	41.8%	53.5%
`Sparse-RS` + SH	63.9%	13.8%	48.8%	**65.5%**
`Sparse-RS` + SA	**72.9% ± 3.6**	29.6% ± 5.6	43.0% ± 1.2	63.6% ± 2.1
`Patch-RS` / `Frame-RS`	70.8% ± 1.3	**30.4% ± 8.0**	**55.5% ± 0.3**	65.3% ± 2.3

note that Dong et al. [2018] show that the perturbations obtained with MI-FGSM have better transferability compared to PGD, while in our experiments they are slightly worse, but we highlight that such observation was done for image-specific l_∞- and l_2-attacks, which are significantly different threat models from what we consider here.

Targeted universal attacks via gradient estimation. Consistently with the other threat models, we test a method based on PGD with gradient estimation. In particular, in the scheme introduced above for `Sparse-RS`, we optimize the loss with the PGD attack and the gradient estimation technique via finite differences as in Alg. 12, similarly to what is done for image-specific perturbations in Sec. 6.2.7 and Sec. 6.2.8 (we also use the same hyperparameters, step size 0.1 for PGD). Moreover, Chen et al. [2019] propose ZO-AdaMM, a black-box method for universal attacks of minimal l_2-norm able to fool a batch of images, without testing them on unseen images. We adapt it to the case of patches and frames removing the penalization term in the loss which minimizes the l_2-norm of the perturbations since it is not necessary in our threat models, and integrate it in our scheme (see above). We used the standard parameters and tuned learning rate, setting 0.1 for patches, 0.5 for frames. In particular, 10 random perturbations, each equivalent to one query, of the current universal perturbation are sampled to estimate the gradient at each iterations, recalling that the total budget is 100,000 queries.

Results. To evaluate the success of a targeted universal patch, we apply to each of 500 images the patch at 100 random locations and compute the percentage of the perturbed images which are classified in the target class. For targeted universal frames, we compute the success rate applying the perturbation to 5,000 images (unseen during training). The results in Table 6.14 are computed as mentioned above for 10 randomly chosen target classes and then averaged (the same set of target classes is used for patches and frames). For the random search based attacks, we repeat the attacks for 3 random seeds. We report the complete results for both threat models in Table 6.14: one can see that our `Sparse-RS` + SH/SA, `Patch-RS` and `Frame-RS` outperform other methods by large margin. Visualizations of the perturbations found by our scheme are shown in Fig. 6.19 and Fig. 6.20.

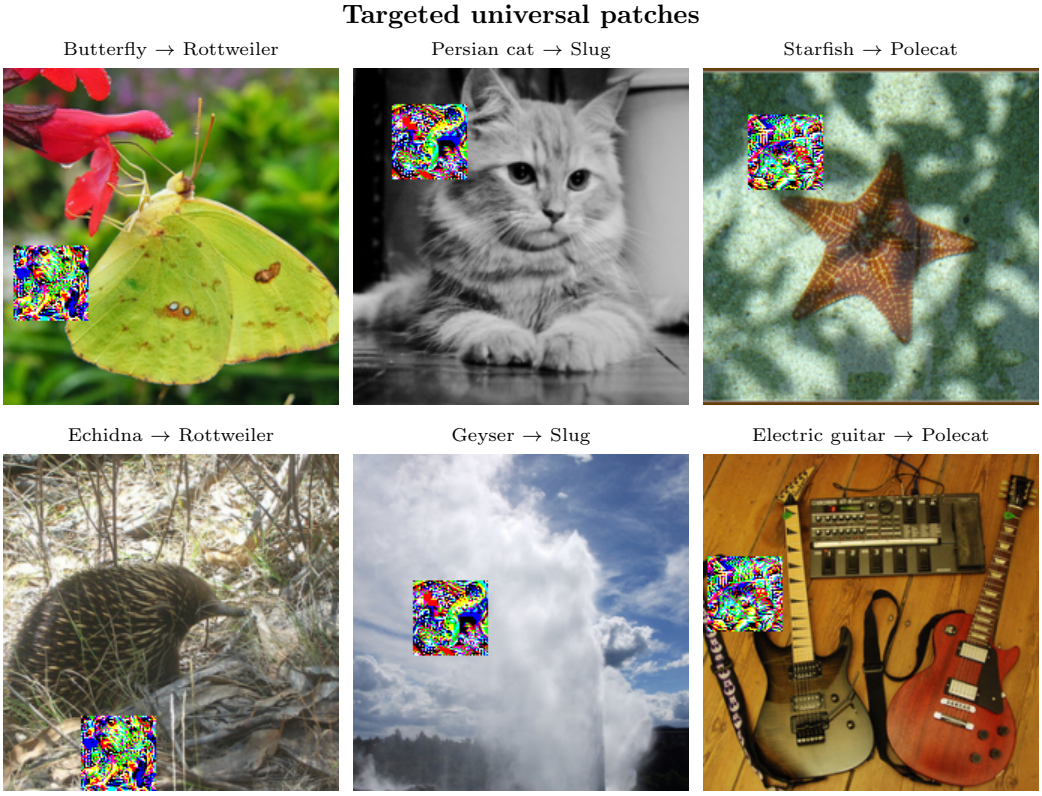

Figure 6.19: We visualize two images in each column with the *same* targeted universal patch generated by `Patch-RS` that changes the predictions to the desired target class.

Figure 6.20: Targeted universal frames generated by our `Frame-RS`. We show two images in each column with the same universal frame that is able to change the decision of the classifier into the desired target class.

6.2.10 Experimental details

Details of l_0-bounded attacks for image classification. We here provide details about the implementation of both l_0-`RS` algorithm and the competitors, many of which are used as available in Adversarial Library [Rony and Ben Ayed, 2020].

l_0-**`RS`:** As mentioned in Sec. 6.2.2, at iteration i the new set M' is formed modifying $\alpha^{(i)} \cdot k$ (rounded to the closest positive integer) elements of M containing the currently modified dimensions (see step 6 in Alg. 11). Inspired by the step-size reduction in gradient-based optimization methods, we progressively reduce $\alpha^{(i)}$. Assuming $N = 10,000$, the schedule of $\alpha^{(i)}$ is piecewise constant where the constant segments start at iterations $j \in \{0, 50, 200, 500, 1000, 2000, 4000, 6000, 8000\}$ with values $\alpha_{\text{init}}/\beta_j$, $\beta_j \in \{2, 4, 5, 6, 8, 10, 12, 15, 20\}$. For a different maximum number of queries N, the schedule is linearly rescaled accordingly. In practice, we use $\alpha_{\text{init}} = 0.3$ and $\alpha_{\text{init}} = 0.1$ for the untargeted and targeted scenario respectively on ImageNet (for both pixel and feature space), $\alpha_{\text{init}} = 0.3$ and $\alpha_{\text{init}} = 0.1$ when considering the pixel and feature space respectively on CIFAR-10.

SparseFool: We use SparseFool [Modas et al., 2019] in the original implementation and optimize the hyperparameter which controls the sparsity of the solutions (called λ, setting finally $\lambda = 1$) to achieve the best results. we use 60 and 100 steps for ImageNet and CIFAR-10 (default value is 20), after checking that higher values do not lead to improved performance but significantly increase the computational cost. Note that SparseFool has been introduced for the l_1-threat model but can generate sparse perturbations which are comparable to l_0-attacks.

PGD$_0$: we use 2,000 iterations (5,000 for targeted attacks) and step size $0.05 \cdot d$ ($0.025 \cdot d$ for targeted attacks) for ImageNet ($0.5 \cdot d$ for CIFAR-10) where $d = 224 \times 224 \times 3$ (resp. $d = 32 \times 32 \times 3$) is the input dimension. PGD$_0$ with gradient estimation uses the same gradient step of PGD$_0$ but with step size $5 \cdot d$ ($1 \cdot d$ for targeted attacks). To estimate the gradient we use finite differences, similarly to Ilyas et al. [2018], as shown in Alg. 12, with the current iterate as input x, $\eta = 0.01/\sqrt{d}$ (in line with what suggested in Ilyas et al. [2019] for this algorithm), $m = 1$ and a zero vector as p. In case of targeted attacks we instead use $m = 50$ (the budget of queries is also 10 times larger) and the current estimated gradient as p. We optimized the aforementioned parameters to achieve the best results and tested that sampling more points to better approximate the gradient at each iteration leads to similar success rate with worse query consumption.

CornerSearch: on ImageNet, we use the following hyperparameters: `n_max` $= 500$, `n_iter` $= 200$, `n_classes` $= 1,000$ (i.e. all classes, which is the default option) for untargeted attacks and `n_max` $= 1,500$, `n_iter` $= 50,000$ for targeted attacks. We note that CornerSearch has a limited scalability to ImageNet as it requires $8 \cdot 224 \cdot 224 = 401,408$ queries only for the initial phase of the attack to obtain the orderings of pixels, and then for the second phase `n_iter` $\cdot$ `n_classes` $= 200,000$ queries for untargeted attacks and `n_iter` $= 50,000$ queries for targeted attacks. On CIFAR-10 we set `n_max` $= 450$, `n_iter` $= 1000$, `n_classes` $= 10$, which amounts to more than $17,000$ queries of budget. Thus, CornerSearch requires significantly more queries compared to l_0-`RS` while achieving a lower success rate. Finally, we note that CornerSearch was designed to minimize the number of modified pixels, while we use it here for l_0-*bounded* attacks.

SAPF: we use the hyperparameters suggested in Fan et al. [2020] and the second-largest logit as the target class for untargeted attacks. With these settings, the algorithm is still very expensive computationally, so we could evaluate it only on 100 points. Note that Fan et al. [2020] report that SAPF achieves the average l_0-norm of at least 30,000 which is orders of magnitudes more than what we consider here. But we evaluate it for completeness as their goal is to obtain a sparse adversarial attack measured in terms of l_0.

ADMM: we perform a grid search over the parameters to optimize the success rate and set `ro=10`, `gama=4`, `binary_search_steps=500` and `Query_iterations=20` (other parameters are used with the default values). Note that Zhao et al. [2019] report results for the l_0-norm only on MNIST. We apply it to both pixel and feature space scenarios by minimizing the corresponding metric.

JSMA-CE: as mentioned in Sec. 6.2.3 we adapt the method introduce in Papernot et al. [2016b] to scale it to attack models on ImageNet. We build the saliency map as the gradient of the cross-entropy loss instead of using the gradient of the logits because ImageNet has 1000 classes and it

Algorithm 12: `GradEst`: Gradient estimation via finite differences

input : loss function L, input point x, number of iterations m, step size η, prior estimation p
output: estimated gradient g

$g \leftarrow p$
for $i = 1, \dots, m$ **do**
  $s \leftarrow \mathcal{N}(0, I)$
  $l_1 \leftarrow L(x + \eta \cdot s)$
  $l_2 \leftarrow L(x - \eta \cdot s)$
  $g \leftarrow g + (l_1 - l_2)/(2\eta) \cdot s$
end

Algorithm 13: JSMA-CE with gradient estimation (malware detection)

input : cross-entropy loss L, input point x_{orig}, number of iterations N, pixel budget k, number of iterations for gradient estimation m, gradient estimation step size η
output: z

$g \leftarrow \mathbf{0}, \quad q \leftarrow 2 \cdot m$
while $q \leq N$ **do**
  $g \leftarrow \texttt{GradEst}(L, x_{\text{orig}}, m, \eta, g)$
  $z \leftarrow x_{\text{orig}}$
  $A \leftarrow$ indices of k largest positive components of g (if there are)
  $z_A \leftarrow 1$
  $q \leftarrow q + 2 \cdot m$
  if z *is adversarial* **then return**;
end

would be very expensive to get gradients of each logit separately (particularly in the black-box settings where one has to first estimate them). Moreover, at each iteration we perturb the pixel which has the gradient with the largest l_1-norm, setting each channel to either 0 or 1 according to the sign of the gradient at it. We perform these iterations until the total sparsity budget is reached. For the black-box version, we modify the algorithm as follows: the gradient is estimated via finite difference approximation as in Alg. 12, the original image x_{orig} as input throughout the whole procedure (this means that the prior p is always the current estimation of the gradient). We found that this gives stronger results compared to an estimation of the gradient at a new iterate. We use the gradient estimation step size $\eta = 0.01$. Then every 10 queries we perturb the k pixels with the largest gradient as described above for the white-box attack and check whether this new image is misclassified (we do not count this extra query for the budget of 10k). This is in line with the version of the attack we use for the malware detection task described in detail in Sec. 6.2.5 and below.

EAD: we use EAD [Chen et al., 2018] from Foolbox [Rauber et al., 2017] with l_1 decision rule, regularization parameter $\beta = 0.1$ optimized to achieve highest success rate in the l_0-threat model and 1000 iterations (other parameters as default). Note that EAD is a strong white-box attack for l_1 and we include it as additional baseline since it can generate sparse attacks.

VFGA: we use it as implemented in Adversarial Library with the default parameters.

FMN: we use the implementation of Adversarial Library with $1,000$ iterations for the untargeted case (as in the original paper), $10,000$ for the targeted one. We set the hyperparameter α_0 to 10 which was optimized with a grid search. We also note that Pintor et al. [2021] do not report results for the l_0-version of their attack on ImageNet.

PDPGD: we run it with the implementation of Adversarial Library with $1,000$ and $10,000$ iterations for untargeted and targeted scenarios respectively, $l_{2/3}$-proximal operator and other parameters with default values. While Matyasko and Chau [2021] introduce also a version of their attack for the pixel space, it is not available in Adversarial Library, then we only consider the method for the comparison in the feature space.

Details of attacks on malware detection. We provide additional details of the attacks used for malware detection tasks.

JSMA-CE with gradient estimation: The idea of the white-box attack of Grosse et al. [2016] is, given a sparsity level of k, to perturb iteratively the feature of the iterate $x^{(i)}$ which corresponds to the largest value (if positive) of $\nabla_x L_{\text{CE}}(f(x^{(i)}), y)$, with f the classifier, y the correct label and L_{CE} the cross-entropy loss, until either the maximum number of modifications are made or misclassification is achieved. With only approximate gradients this approach is not particularly effective. However, since $k \ll d$ and only additions can be made, we aim at estimating the gradient of the cross-entropy loss at x_{orig} and then set to 1 the k elements (originally 0) of x_{orig} with the largest component in the approximated gradient. With the goal of query efficiency, every m iterations of gradient estimation through finite differences we check if an adversarial example has been found (we do not count these queries in the total for the query limit). Alg. 13 shows the procedure, and we set $m = 5$ and $\eta = 1$ (we tested other values which achieved worse or similar performance).

PGD_0 with gradient estimation: we use PGD_0 with the gradient estimation technique presented in Alg. 12 with the original point as input x, $m = 10$, $\eta = 100$ and the current estimate of the gradient as prior p (unlike on image classification tasks where the gradient is estimated at the current iterate and no prior used). Moreover, we use step size $4 \cdot \sqrt{d}$ and modify the projection step the binary input case and so that features are only added and not removed (see above).

Details of `Patch-RS` algorithm. As mentioned in Sec. 6.2.7, in this scenario we optimize via random search the attacks for each image independently. In `Patch-RS` we alternate iterations where a candidate update of the patch is sampled with others where a new location is sampled.

We have a location update every m iterations, with $m = 5$ ($1 : 4$ scheme Sec. 6.2.11, four patch updates then one location update) for untargeted attacks and $m = 10$ for targeted ones: the latter have more queries available (50k vs 10k) and wider patches (40×40 vs 20×20), therefore it is natural to dedicate a larger fraction of iterations to optimizing the content rather than the position (which has also a smaller feasible set). We present in Sec. 6.2.11 an ablation study the effect of different frequencies of location updates in the untargeted scenario. The position of the patch is updated with an uniformly sampled shift in $[-h^{(i)}, h^{(i)}]$ for each direction (plus clipping to the image size if necessary), where $h^{(i)}$ is linearly decreased from $h^{(0)} = 0.75 \cdot s_{\text{image}}$ to $h^{(N)} = 0$ (s_{image} indicates the side of the squared images). In this way, initially, the patch can be easily moved on the image, while towards the final iterations it is kept (almost) fixed and only its values are optimized.

The patch is initialized by superposing at random positions on a black image 1000 squares of random size and color in $\{0, 1\}^3$. Then, we update the patch following the scheme of Square Attack, that is sampling random squares with color one of the corners of the color cube and accept the candidate if it improves the target loss. The size of the squares is decided by a piecewise constant schedule, which we inherit from Andriushchenko et al. [2020]. Specifically, at iteration i the square-shaped updates of a patch with size $s \times s$ have side length $s^{(i)} = \sqrt{\alpha^{(i)}} \cdot s$, where the value of $\alpha^{(i)}$ starts with $\alpha^{(0)} = \alpha_{\text{init}}$ and then is halved at iteration $j \in \{10, 50, 200, 500, 1000, 2000, 4000, 6000, 8000\}$ if the query limit is $N = 10,000$, otherwise the values of j are linearly rescaled according to the new N. Hence, α_{init} is the only tunable parameters of `Patch-RS`, and we set $\alpha_{\text{init}} = 0.4$ for untargeted attacks and $\alpha_{\text{init}} = 0.1$ for targeted ones. In contrast to Square Attack, in the second half of the iterations dedicated to updates of size 1×1, `Patch-RS` performs a refinement of the patch applying only single-channel updates. We show in Sec. 6.2.11 that both the initialization tailored for the specific threat model and the single-channel updates allow our algorithm to achieve the best success rate and query efficiency.

Details of other patch attacks. We provide the implementation details of other attacks for the patch threat model.

`Sparse-RS` + SH and `Sparse-RS` + SA: When integrating SignHunter [Al-Dujaili and O'Reilly, 2020] and Square Attack [Andriushchenko et al., 2020] in our framework for adversarial patches, the updates of the locations are performed as described above, while the patches are modified with the l_∞ attacks with constraints given by the input domain $[0, 1]^d$ (in practice one can fix

a sufficiently large l_∞ threshold $\epsilon_\infty = 1$). SH does not have free parameters, while we tune the only parameter p of SA (which has the same role as α_{init} of `Patch-RS`) and use $p = 0.4$ and $p = 0.2$ for the untargeted and targeted case respectively.

LOAP: Location-Optimized Adversarial Patches (LOAP) [Rao et al., 2020] is a white-box PGD-based attack which iteratively first updates the patch with a step in the direction of the sign of the gradient to maximize the cross-entropy function at the current iterate, then it tests if shifting of `stride` pixels the patch in one of the four possible directions improves the loss and, if so, the location is updated otherwise kept (Rao et al. [2020] have also a version where only one random shift is tested). We use LOAP with the original implementation[6] with 1,000 iterations, learning rate 0.05 and the `full` optimization of the location with `stride=5`. To adapt LOAP to the black-box setup, we use the gradient estimation via finite differences shown in Alg. 12 restricted to the patch to optimize (this means that the noise sampled in step 3 is applied only on the components of the image where the patch is). In particular we use the current iterate as input x, $m = 5$ iterations, $\eta = 10/(s \cdot \sqrt{c})$ if the patch has dimension $s \times s \times c$ and the gradient estimation at the previous iteration as prior p. Moreover, we perform the update of the location by sampling only 1 out of the 4 possible directions (as more queries are necessary for better gradient estimation) with `stride=5`, and learning rate 0.02 for the gradient steps (the sign of the gradient is used as a direction). Note that we optimized these hyperparameters to achieve the best final success rate. Moreover, each iteration costs 11 queries (10 for Alg. 12 and 1 for location updates) and we iterate the procedure until the budget of queries is exhausted.

TPA: The second method we compare to is TPA [Yang et al., 2020a], which is based on reinforcement learning and exploits a dictionary of patches. Note that in Yang et al. [2020a] TPA was used primarily putting multiple patches on the same image to achieve misclassification, while our threat model does not include this option. We obtained the optimal values for the hyperparameters for untargeted attacks (`rl_batch=400` and `steps=25`) via personal communication with the authors and set those for the targeted scenario (`rl_batch=1000` and `steps=50`) similarly to what reported in the original code[7] doubling the value of `rl_batch` to match the budget of queries we allow. TPA has a mechanism of early stopping, which means that it might happen that not the whole budget of queries is exploited even for unsuccessful points. Finally, Yang et al. [2020a] show that TPA significantly outperforms the methods of Fawzi and Frossard [2016], which also generates image-specific patches, although optimizing the shape and the location but not the values of the perturbation. Thus we do not compare directly to Fawzi and Frossard [2016] in our experiments.

6.2.11 Ablation studies

We here present a series of ablation studies to better understand the robustness of our algorithms to different random seeds, how its performance changes depending on α_{init}, and to justify the algorithmic choice of the piecewise decaying schedule for $\alpha^{(i)}$. We focus the ablation study only on l_0`-RS` and `Patch-RS` methods since for other threat models the setting is less challenging as there is no need to optimize the location of the perturbation.

Different random seeds in l_0`-RS`. First, we study how the performance of l_0`-RS` in the pixel space varies when using different random seeds, which influence the stochastic component inherent in random search. In Table 6.15 we report mean and standard deviation over 10 runs (with different seeds) of success rate, average and median queries of l_0`-RS` on VGG and ResNet with sparsity levels $k \in \{50, 100, 150\}$ (the same setup of Sec. 6.2.3, with the additional $k = 100$). One can observe that the success rate is very stable in all the cases and the statistics of query consumption consistent across different runs.

Different values of α_{init} in l_0`-RS`. Then we analyze the behaviour of our algorithm with different values of α_{init}, since it is the only free hyperparameter of `Sparse-RS`. Let us recall that it is used to regulate how much M' and Δ' differ from M and Δ respectively in steps 6-7 of Alg. 11 at each iteration. Fig. 6.21 shows the success rate and query usage (computed on the successful

[6] https://github.com/sukrutrao/Adversarial-Patch-Training
[7] https://github.com/Chenglin-Yang/PatchAttack

Table 6.15: Mean and standard deviation of the success rate and query efficiency of untargeted l_0-`RS` repeated with 10 different random seeds. The success rate and query efficiency are very stable over random seeds.

model	k	*success rate (%)*	*successful points*		*all points*	
			avg. queries	*med. queries*	*avg. queries*	*med. queries*
VGG	50	97.4 ± 0.42	497 ± 28	80 ± 4	745 ± 43	86 ± 5
	100	99.8 ± 0.14	320 ± 20	42 ± 3	335 ± 17	43 ± 3
	150	100.0 ± 0.00	193 ± 16	26 ± 2	193 ± 16	26 ± 2
ResNet	50	94.6 ± 0.70	686 ± 47	135 ± 6	1187 ± 48	166 ± 11
	100	99.7 ± 0.15	544 ± 36	73 ± 5	571 ± 26	74 ± 5
	150	100.0 ± 0.00	365 ± 19	49 ± 3	365 ± 19	49 ± 3

Table 6.16: Ablation study on the effect of initialization and single-channel updates for untargeted image-specific adversarial patches (size 20×20 pixels) within our proposed framework. We highlight in grey our modified schemes that lead to a higher success rate and query efficiency. Note that the single-channel updates come later in the optimization process when 50% success rate is reached, and thus they do not influence the median. We also report the results of `Sparse-RS` + SH for reference.

attack	*init.*	*single ch.*	VGG			ResNet		
			succ. rate	*mean qr.*	*med. qr.*	*succ. rate*	*mean qr.*	*med. qr.*
`Sparse-RS` + SH	-	-	82.6%	2479	514	75.3%	3290	549
`Sparse-RS` + SA	stripes	no	$85.6\%_{\pm 1.1}$	$2367_{\pm 83}$	$533_{\pm 40}$	$78.5\%_{\pm 1.0}$	$2877_{\pm 64}$	$458_{\pm 43}$
	stripes	yes	$85.7\%_{\pm 1.4}$	$2359_{\pm 89}$	$533_{\pm 40}$	$79.1\%_{\pm 1.1}$	$2861_{\pm 58}$	$458_{\pm 43}$
	squares	no	$87.1\%_{\pm 0.9}$	$2174_{\pm 50}$	$429_{\pm 22}$	$78.9\%_{\pm 1.3}$	$2820_{\pm 91}$	$438_{\pm 68}$
`Patch-RS`	squares	yes	$87.8\%_{\pm 0.7}$	$2160_{\pm 44}$	$429_{\pm 22}$	$79.5\%_{\pm 1.4}$	$2808_{\pm 89}$	$438_{\pm 68}$

samples) of our untargeted l_0-bounded attack on VGG at the usual three sparsity levels k for runs with $\alpha_{\text{init}} \in \{0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.6, 0.8, 1\}$ (for our experiments in Sec. 6.2.3 we use $\alpha_{\text{init}} = 0.3$). We observe that the success rate is similar (close to 100%) for all the values, with a slight degradation for the largest ones. In order to minimize the queries of the classifier, the runs with α_{init} between 0.1 and 0.4 are comparably good, with small differences in the tradeoff between average and median number of queries.

Constant vs decaying $\alpha^{(i)}$ in l_0-`RS`. In order to demonstrate the role of decaying the difference between the candidate updates M' and Δ' and the current iterates M and Δ over iterations (see steps 6-7 of Alg. 11) to achieve good performance, we run our attack with constant $\alpha^{(i)}$ schedule instead of the piecewise constant schedule with decreasing values. We fix $\alpha^{(i)} = c \in [0, 1]$ for every i so that for the whole algorithm M' and M differ in $\max\{c \cdot k, 1\}$ elements. In Fig. 6.22 we report the results achieved by $c \in \{0, 0.05, 0.15, 0.3, 0.5, 1\}$ on VGG at $k \in \{50, 100, 150\}$, together with the baseline (black dashed line) of the standard version of l_0-`RS`. One can observe that small constant values c for $\alpha^{(i)} = c$ achieve a high success rate but suffer in query efficiency, in particular computed regarding the median and for larger k, while the highest values of c lead to significantly worse success rate (note that average and median queries are computed only on the successful points) than the baseline. These results highlight how important it is to have an initial exploration phase, with a larger step size, and at a later stage a more local optimization.

Initialization and single-channel updates in `Patch-RS`. The original Square Attack for l_∞- and l_2-bounded perturbations applies as initialization vertical stripes on the target image, and it does not consider updates of a single channel (these can be randomly sampled but are not explicitly performed). Table 6.16 shows the success rate and query efficiency (computed on all points) for untargeted adversarial attacks with patches of size 20×20 on VGG and ResNet, averaged over 5 random seeds for methods based on random search. Both the initialization with squares and the single-channel updates already individually improve the performance of `Sparse-RS` + SA and, when combined, allow `Patch-RS` to outperform the competitors (see also Sec. 6.2.7).

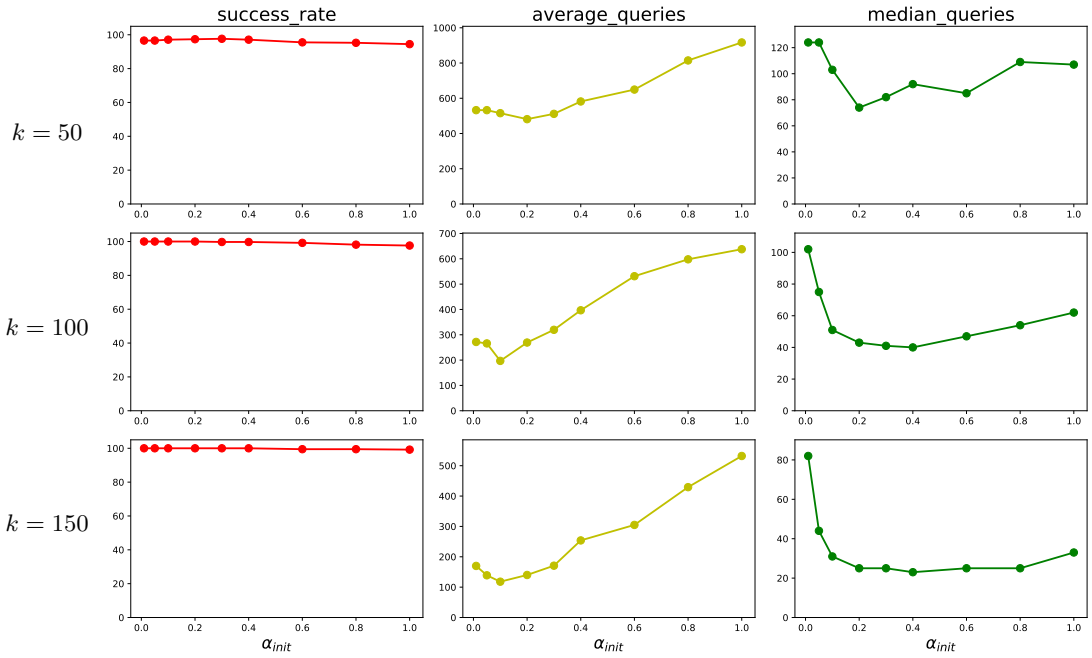

Figure 6.21: Ablation study on the influence of α_{init}, the hyperparameter which regulates the size of the updates at each iteration. We show success rate (first column), average (second) and median queries (third) achieved by l_0-`RS` on VGG at sparsity levels $k = \{50, 100, 150\}$. Considering jointly the three statistics values in $[0.1, 0.4]$ are preferable for this threat model and schedule.

Ratio location to patch updates. We study here the effect of different ratios between the number of location and patch updates in our framework for image-specific untargeted patches (size 20×20). In Table 6.17 we report the performance `Sparse-RS` + SH, `Sparse-RS` + SA and our final method `Patch-RS` with various schemes: alternating an update of the location to one of the patch (1:1), doing 4 or 9 location updates for one patch update (4:1, 9:1) and vice versa (1:4, 1:9). Considering the results on both networks, the scheme with 1:4 ratio yields for every attack the highest success rate or very close. In query consumption, the schemes 1:1 and 1:4 are similar, with some trade-off between average and median. Moreover, we observe that the ranking of the three methods is preserved for all ratios of location and patch updates we tested, with `Patch-RS` outperforming the others. Also, its effectiveness is less affected by suboptimal ratios, showing the stability of our algorithm.

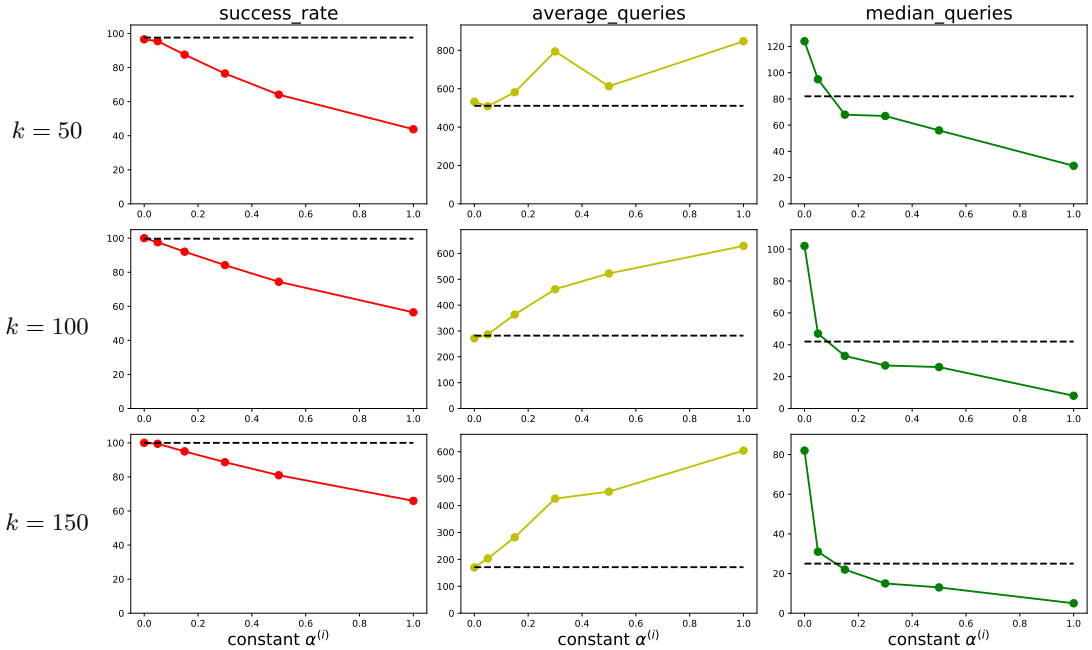

Figure 6.22: Performance of untargeted l_0-`RS` on VGG when using a constant schedule for $\alpha^{(i)}$, that is the size of $|A| = |B| = c \cdot k$ (see Sec. 6.2.2) at every iteration, or equivalently $\alpha^{(i)} = c$ for every $i = 1, \ldots, N$. The dashed line is the reference of the results achieved with the piecewise constant schedule to decay $\alpha^{(i)}$. While constantly small updates lead to good success rate, the median number of queries used increases notably, especially with larger sparsity levels k.

Table 6.17: Performance of different attacks for untargeted image-specific adversarial patches of size 20×20 varying the ratio between updates of the location and of the patch. We use 1:4 ratio in our framework which is the best choice for `Sparse-RS` + SA and `Patch-RS` (for `Sparse-RS` + SH 1:1 would work equally well). Note that our novel attack `Patch-RS` outperforms the competitors consistently in terms of the success rate and query efficiency across all choices of the ratio of the location and patch updates. For random search based methods, we repeat the attack with 5 random seeds.

ratio	*attack*	VGG			ResNet		
		succ. rate	*mean qr.*	*med. qr.*	*succ. rate*	*mean qr.*	*med. qr.*
9:1	`Sparse-RS` + SH	74.9%	3897	1331	68.0%	4038	1121
	`Sparse-RS` + SA	$71.4\%_{\pm 0.9}$	$3927_{\pm 104}$	$1361_{\pm 224}$	$68.7\%_{\pm 1.0}$	$3931_{\pm 76}$	$789_{\pm 65}$
	`Patch-RS`	$76.8\%_{\pm 0.8}$	$3414_{\pm 48}$	$779_{\pm 69}$	$72.0\%_{\pm 0.8}$	$3655_{\pm 57}$	$731_{\pm 97}$
4:1	`Sparse-RS` + SH	77.1%	3332	781	72.9%	3639	701
	`Sparse-RS` + SA	$77.7\%_{\pm 1.2}$	$3236_{\pm 106}$	$735_{\pm 41}$	$73.6\%_{\pm 0.6}$	$3408_{\pm 35}$	$548_{\pm 65}$
	`Patch-RS`	$82.4\%_{\pm 0.8}$	$2816_{\pm 28}$	$485_{\pm 74}$	$75.3\%_{\pm 0.5}$	$3248_{\pm 51}$	$443_{\pm 41}$
1:1	`Sparse-RS` + SH	82.9%	2718	500	75.3%	3168	451
	`Sparse-RS` + SA	$84.6\%_{\pm 1.2}$	$2485_{\pm 92}$	$504_{\pm 20}$	$78.2\%_{\pm 0.7}$	$2943_{\pm 142}$	$389_{\pm 46}$
	`Patch-RS`	$86.8\%_{\pm 0.8}$	$2301_{\pm 61}$	$352_{\pm 28}$	$79.6\%_{\pm 1.1}$	$2764_{\pm 86}$	$316_{\pm 10}$
1:4	`Sparse-RS` + SH	82.6%	2479	514	75.3%	3290	549
	`Sparse-RS` + SA	$85.6\%_{\pm 1.1}$	$2367_{\pm 83}$	$533_{\pm 40}$	$78.5\%_{\pm 1.0}$	$2877_{\pm 64}$	$458_{\pm 43}$
	`Patch-RS`	$87.8\%_{\pm 0.7}$	$2160_{\pm 44}$	$429_{\pm 22}$	$79.5\%_{\pm 1.4}$	$2808_{\pm 89}$	$438_{\pm 68}$
1:9	`Sparse-RS` + SH	80.7%	2719	564	72.2%	3451	843
	`Sparse-RS` + SA	$85.9\%_{\pm 0.8}$	$2419_{\pm 60}$	$542_{\pm 34}$	$77.6\%_{\pm 0.8}$	$3000_{\pm 92}$	$617_{\pm 41}$
	`Patch-RS`	$87.5\%_{\pm 1.6}$	$2261_{\pm 44}$	$542_{\pm 39}$	$78.4\%_{\pm 0.6}$	$2961_{\pm 34}$	$533_{\pm 54}$

Chapter 7

Adaptation of Robust Classifiers via Fine-tuning

As mentioned in the previous sections, adversarial training [Madry et al., 2018] and its variants are the standard method to obtain adversarially robust models. A main drawback of adversarial training is the increased computational cost compared to standard training, especially when one considers that larger networks yield significantly better results than smaller ones [Gowal et al., 2020]. Moreover, when it comes to training for robustness in multiple threat models, the cost of obtaining robust classifiers might further increase, since several l_p-attacks need to be generated. In the following we show that it possible to adapt models trained against l_p-attacks to be robust in either another l_q-threat model or the union of multiple threat models by a short fine-tuning with adversarial training. In this way one can exploit e.g. the large l_∞-robust models which are publicly available to obtain robust classifiers in new threat models with small computational effort, even for large datasets like ImageNet.

7.1 Adversarial robustness against multiple and single l_p-threat models via quick fine-tuning of robust classifiers

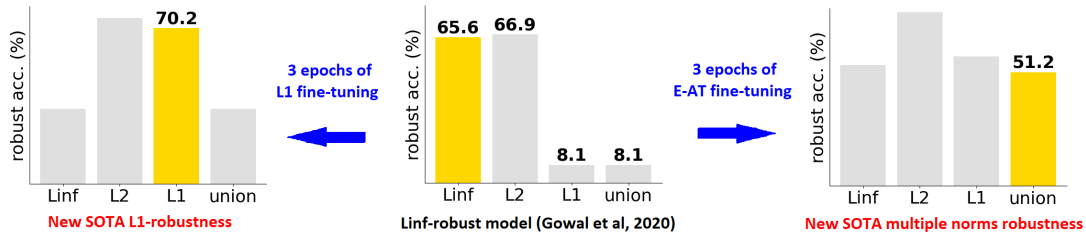

Figure 7.1: We fine-tune for 3 epochs the WideResNet-70-16 on CIFAR-10 from Gowal et al. [2020] with highest l_∞-robustness to be either robust wrt l_1 (left) or with our E-AT to be robust wrt the union of the l_∞-, l_2-, and l_1-threat models (right). We achieve state-of-the-art results in both threat models. The plots show the robust accuracy in the individual threat models and in their union for the initial l_∞-robust classifier (middle) and the fine-tuned ones, with the target threat model highlighted.

In the following we show that, using the geometry of the l_p-balls similarly to what done in Sec. 3.3, the computationally expensive multiple norm training procedures of Tramèr and Boneh [2019], Maini et al. [2020], which costs up to three times as much as normal adversarial training, can be replaced by a very effective and simple form of adaptively alternating between the two extreme norms, namely l_1 and l_∞. This scheme achieves similar robustness for the union of the threat models to more costly previous approaches. Additionally, we show that on CIFAR-10 three epochs and on ImageNet even just a single epoch of fine-tuning with our extreme norms adversarial training (E-AT) are sufficient to turn any l_p-robust model for $p \in \{1, 2, \infty\}$ into a model which is robust against all l_p-threat models for $p \in \{1, 2, \infty\}$, even if the original classifier

was completely non-robust against one of them. Fine-tuning one of the currently most robust networks in the l_∞-threat model [Gowal et al., 2020] for CIFAR-10 we significantly improve the current state-of-the-art performance for multiple norm robustness (i.e. robustness over the union of l_1, l_2 and l_∞-balls) without the need of training from scratch this large network with existing expensive techniques. Also, our E-AT fine-tuning scheme yields in just one epoch the first ImageNet model which is robust against multiple l_p-bounded attacks. Finally, we show that fine-tuning, again with only 3 epochs for CIFAR-10 and one epoch for ImageNet, is sufficient to transfer robustness from one threat model to another one, which very quickly yields baselines for all threat models. In this way, starting from l_∞-robust classifiers we achieve SOTA l_1-robust classifiers on CIFAR-10 and ImageNet. These results are quite remarkable as the original classifiers show no or very little l_1-robustness. Fig. 7.1 summarizes the results of fine-tuning of robust classifiers on CIFAR-10 (see Table 7.2 and Table 7.6 for details), and Table 7.3 for ImageNet shows that our results hold also for fine-tuning of transformer architectures.

Note that since Sec. 5.1 showed that on models defended with adversarial training the two versions of APGD (with budget as in AutoAttack) already give an accurate robustness evaluation, and as we have to evaluate very large models always for three threat models, we use those as a strong standard attack in our experiments.

7.1.1 Multiple-norm robustness via fast fine-tuning of existing robust models

Adversarial training for the union of l_1-, l_2- and l_∞-balls. Let us denote by $f_\theta : \mathbb{R}^d \to \mathbb{R}^K$ the classifier parameterized by $\theta \in \mathbb{R}^n$, with input $x \in \mathbb{R}^d$ and $f_\theta(x) \in \mathbb{R}^K$ where K is the number of classes of the task. Let further $\mathcal{D} = \{(x_i, y_i)\}_i$ be the training set, with y_i the correct label of x_i, and $\mathcal{L} : \mathbb{R}^K \times \mathbb{R}^K \to \mathbb{R}$ a given loss function. The aim is to enforce adversarial robustness in all multiple l_p-balls simultaneously, i.e., defining $B_p(\epsilon_p) = \{x \in \mathbb{R}^d : \|x\|_p \leq \epsilon_p\}$, the threat model is the union of the individual l_p-balls, i.e. $\Delta = B_1(\epsilon_1) \cup B_2(\epsilon) \cup B_\infty(\epsilon_\infty)$. Δ is a non convex set, since in practice the ϵ_p are chosen such that no l_p-ball contains any of the others. In adversarial training the worst case loss for input perturbations in the threat model, $\max_{\delta \in \Delta} \mathcal{L}(f_\theta(x_i + \delta), y_i)$, is minimized. Efficiently maximizing the loss $\mathcal{L}$ in the union of threat models is non-trivial and different approaches have been proposed.

MAX: Tramèr and Boneh [2019] suggest to run the three attacks for each $B_p(\epsilon_p)$ for $p \in \{1, 2, \infty\}$ independently and then use the one which realizes the highest loss, that is

$$\max_{p \in \{1,2,\infty\}} \max_{\delta \in B_p(\epsilon_p)} \mathcal{L}(f_\theta(x_i + \delta), y_i).$$

This training optimizes directly the worst case in the union but comes at the price of being nearly three times as expensive as normal adversarial training wrt a single l_p-ball.

AVG: Additionally, Tramèr and Boneh [2019] suggest to run the three attacks independently but replace the inner maximization problem with

$$\sum_{p \in \{1,2,\infty\}} \max_{\delta \in B_p(\epsilon_p)} \mathcal{L}(f_\theta(x_i + \delta), y_i),$$

and thus averaging the updates of all l_p-balls with the motivation of not "wasting" the computed attacks, in particular when the attained loss values are rather similar and thus the max is ambiguous. This has similar cost to MAX.

MSD: Maini et al. [2020] argue that the correct way to maximize the loss in the union is to test during the PGD attacks all the three steepest ascent updates corresponding to the three norms (sign of the gradient for l_∞, normalized gradient for l_2 and a smoothed l_1-step by using the top-k components in magnitude of the gradient) and then take the step which yields the highest loss. Maini et al. [2020] report that MSD outperforms both AVG and MAX, also in terms of a more stable training. As all three updates (forward passes) are tested but only one backward pass is needed (gradient is the same) this costs roughly two times as much as normal adversarial training.

SAT: Madaan et al. [2021] introduce Stochastic Adversarial Training (SAT) which randomly samples $p \in \{1, 2, \infty\}$ for each batch and performs PGD only for the corresponding l_p-norm.

Table 7.1: **CIFAR-10 - Other methods vs E-AT for fine-tuning:** We fine-tune with different methods for multiple norms for 3 epochs the RN-18 robust wrt l_∞ (mean and standard deviation of the clean and robust accuracy over 5 seeds is reported). We report clean performance, robust accuracy in each l_p-threat model, their average and the robust accuracy in their union (all values in percentage).

model	*clean*	l_∞ ($\epsilon_\infty = \frac{8}{255}$)	l_2 ($\epsilon_2 = 0.5$)	l_1 ($\epsilon_1 = 12$)	*average*	*union*	*time / epoch*
RN-18 l_∞-AT	83.7	48.1	59.8	7.7	38.5	7.7	151 s
+ SAT	83.5 ± 0.2	43.5 ± 0.2	68.0 ± 0.4	47.4 ± 0.5	53.0 ± 0.2	41.0 ± 0.3	161 s
+ AVG	84.2 ± 0.4	43.3 ± 0.4	68.4 ± 0.6	46.9 ± 0.6	52.9 ± 0.4	40.6 ± 0.4	479 s
+ MAX	82.2 ± 0.3	45.2 ± 0.4	67.0 ± 0.7	46.1 ± 0.4	52.8 ± 0.3	42.2 ± 0.6	466 s
+ MSD	82.2 ± 0.4	44.9 ± 0.3	67.1 ± 0.6	47.2 ± 0.6	53.0 ± 0.4	42.6 ± 0.2	306 s
+ E-AT	82.7 ± 0.4	44.3 ± 0.6	68.1 ± 0.5	48.7 ± 0.5	53.7 ± 0.3	42.2 ± 0.8	160 s

While SAT has the same cost as standard adversarial training, Madaan et al. [2021] report that it does not perform very well.

Multiple-norm robustness via fast fine-tuning of robust models. Prior works [Tramèr and Boneh, 2019, Kang et al., 2019a] observed that models adversarially trained wrt l_∞ give non trivial robustness to l_2-attacks, although lower than what one gets directly training against such attacks, and vice versa. This is confirmed by our evaluation in Sec. 7.1.6, where we also notice that l_1-AT provides good robust accuracy in l_2. On the other hand, training for l_∞ resp. l_1 does not yield particular robustness to the dual norm, which is reasonable since the perturbations generated in the two threat models are very different, while the l_2-threat model is an intermediate case which yields partial robustness against l_∞ and l_1. Therefore, we propose to use models trained for robustness wrt a single norm as good initializations to achieve, within a small computational budget, multiple norms robustness. We test this by fine-tuning for 3 epochs an l_∞-robust model on CIFAR-10 with the existing methods for multiple norms robustness and our E-AT. Table 7.1 shows that a short fine-tuning (details in Sec. 7.1.3) of a PreAct ResNet-18 [He et al., 2016b] trained with adversarial training wrt l_∞ is effective in achieving competitive robustness in the union, not far from those of full training from random initialization (see Table 7.5). However, the most effective methods, MAX and MSD, are 2-3x slower than standard adversarial training, while SAT is as fast as l_∞-AT but performs slightly worse. E-AT aims at achieving the same results as MAX and MSD in the union while having complexity on par with SAT. Moreover, we report the average robustness in the 3 threat models, as done in prior works, where E-AT achieves the best results. In Sec. 7.1.6 we show that similar observations can be made when fine-tuning models initially l_2- or l_1-robust, and (Sec. 7.1.3) are not specific to CIFAR-10 but generalize to ImageNet.

All previous methods assume that for achieving robustness to multiple norms each threat models has to be used at training time. In the following we first present an argument, using results from Sec. 3.3.1, suggesting that this need not be the case. Based on this analysis, we introduce our extreme norms adversarial training (E-AT) which achieves multiple norm robustness at the same price as training for a single l_p. Finally, we fine-tune with E-AT large robust models on CIFAR-10 and ImageNet.

7.1.2 Extreme norms adversarial training (E-AT)

Geometry of the union of l_p-balls and their convex hull. The main insight we use for E-AT is that a linear classifier which is robust in both an l_1- and an l_∞-ball is also robust wrt the largest l_p-ball for $1 \leq p \leq \infty$ which fits into the convex hull of the union of the l_1- and l_∞-ball. This ball is significantly larger than the largest l_p-ball contained into the union of the l_1- and l_∞-ball (see Fig. 3.4 and Fig. 7.2). Thus it is sufficient to be robust wrt the two "extreme" norms l_1 and l_∞ to ensure robustness. While this is exact for affine classifiers, we conjecture that for neural networks this will at least hold approximately true (note that typical ReLU-networks yield piecewise affine classifiers [Arora et al., 2018]) and for the model it is the most efficient way in terms of capacity to be l_1- and l_∞-robust.

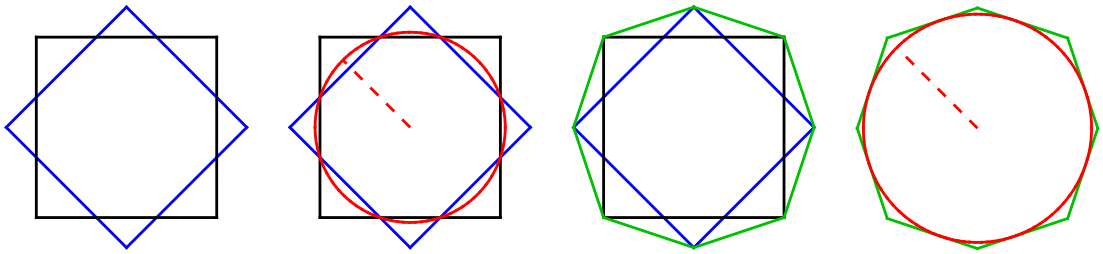

Figure 7.2: Visualization of the l_2-ball contained in the union resp. the convex hull of the union of l_1- and l_∞-balls in $\mathbb{R}^2$. **First:** co-centric l_1-ball (blue) and l_∞-ball (black). **Second:** in red the largest l_2-ball contained in the union of l_1- and l_∞-ball. **Third:** in green the convex hull of the union of the l_1- and l_∞-ball. **Fourth:** the largest l_2-ball (red) contained in the convex hull. The l_2-ball contained in the convex hull is significantly larger than that in the union of l_1- and l_∞-ball.

We work in the non-trivial setting, where the balls are not included in each other as otherwise the problem of enforcing multiple norms (including l_1 or l_∞) robustness boils down again to single norm robustness. For this, it has to hold $\epsilon_1 \in (\epsilon_\infty, d\epsilon_\infty)$: for CIFAR-10, $\epsilon_\infty = \frac{8}{255}$ and dimension $d = 3072$ yield an upper bound $\epsilon_1 \leq 96.38$ which is far higher than $\epsilon_1 = 12$ commonly used for l_1-threat models. We denote by $U_{1,\infty}(\epsilon_1, \epsilon_\infty) = B_1(\epsilon_1) \cup B_\infty(\epsilon_\infty)$ the union of the l_1- and l_∞-balls, and Proposition 3.3.1 provides the l_p-robustness, for $1 < p < \infty$, of a classifier robust in $U_{1,\infty}(\epsilon_1, \epsilon_\infty)$. For us the case $p = 2$ is most interesting, and, given that $\epsilon_1 \gg \epsilon_\infty$, the l_2-robustness can be tightly upper bounded as

$$\epsilon_2 := \min_{x \in \mathbb{R}^d \setminus U_{1,\infty}(\epsilon_1, \epsilon_\infty)} \|x\|_2 \;\leq\; \sqrt{\epsilon_\infty^2 + \frac{\epsilon_1^2}{d-1}}. \tag{7.1}$$

For CIFAR-10 one gets $\epsilon_2 \leq 0.2188$. As the radius of the l_2-threat model is usually chosen as $\epsilon_2 = 0.5$, robustness in the union alone would not be sufficient to achieve the desired robustness wrt l_p for $p \in \{1, 2, \infty\}$. However, if we consider affine classifiers then a guarantee for $B_1(\epsilon_1)$ *and* $B_\infty(\epsilon_\infty)$ implies a guarantee with respect to the convex hull C of their union $B_1(\epsilon_1) \cup B_\infty(\epsilon_\infty)$ as an affine classifier generates a half-space and thus only the extreme points of $B_1(\epsilon_1)$ resp. $B_\infty(\epsilon_\infty)$ matter (see Fig. 3.4 and Fig. 7.2 for illustrations). As standard architectures using ReLU activation function yield a piecewise affine classifier one can anticipate that this result gives at least a rule of thumb on the expected l_p-robustness when one is l_1- and l_∞-robust. Following Theorem 3.3.1 with the choice of $\epsilon_1, \epsilon_\infty$ from above, one gets for the radius of the l_2-ball that fits into the convex hull C of the union of $B_1(\epsilon_1)$ and $B_\infty(\epsilon_\infty)$

$$\epsilon_2 := \min_{x \in \mathbb{R}^d \setminus C} \|x\|_2 = \frac{\epsilon_1}{\sqrt{\epsilon_1/\epsilon_\infty - \alpha + \alpha^2}} \approx 0.6178. \tag{7.2}$$

Thus for a desired l_2-robustness with radius less than 0.6178 it is sufficient for an affine classifier, and at least plausible for a ReLU network, to enforce l_1-robustness with $\epsilon_1 = 12$ and l_∞-robustness with $\epsilon_\infty = \frac{8}{255}$. This motivates our extreme norms adversarial training (E-AT) and fine-tuning.

E-AT algorithm. In light of the geometrical argument presented in the previous section, we propose to train only on adversarial perturbations for the l_∞- and l_1-threat models if the l_2-radius obtained from Theorem 3.3.1 is larger than the radius ϵ_2 of the l_2-threat model. In this case it is sufficient to just train for the extremes l_1 and l_∞ in order to achieve robustness also to the intermediate l_p-attacks with $p \in (1, \infty)$. Since we seek a method as expensive as standard adversarial training, for each batch we do either the l_1- or the l_∞-attack. For full training from a random initialization simply alternating or sampling uniformly at random from the l_1- and l_∞-attack works already well. However, for very quick fine-tuning, e.g. just one epoch in the case of ImageNet, for multiple norm robustness from an existing classifier robust wrt a single threat model, one has to take into account the existing robustness of the model. Thus we use an adaptive sampling strategy based on the running averages, reset at every epoch, of the robust training errors rerr_1 and rerr_∞ (note that these running averages are computed just from

Table 7.2: **CIFAR-10 - 3 epochs of E-AT fine-tuning on l_p-robust models:** We fine-tune with E-AT models robust wrt a single l_p-norm, and report the robust accuracy on 1000 test points for all threat models and the difference to the initial classifier. (*) uses extra data.

model		*clean*		l_∞ ($\epsilon_\infty = \frac{8}{255}$)		l_2 ($\epsilon_2 = 0.5$)		l_1 ($\epsilon_1 = 12$)		*union*	
Fine-tuning l_∞-robust models											
RN-50 - l_∞		88.7		50.9		59.4		5.0		5.0	
[Engstrom et al., 2019a]	+ FT	86.2	-2.5	46.0	-4.9	70.1	10.7	49.2	44.2	43.4	38.4
WRN-34-20 - l_∞		87.2		56.6		63.7		8.5		8.5	
[Gowal et al., 2020]	+ FT	88.3	1.1	49.3	-7.3	71.8	8.1	51.2	42.7	46.2	37.7
WRN-28-10 - l_∞ (*)		90.3		59.1		65.7		8.0		8.0	
[Carmon et al., 2019]	+ FT	90.3	0.0	52.6	-6.5	74.7	9.0	54.0	46.0	48.7	40.7
WRN-28-10 - l_∞ (*)		89.9		62.9		67.2		10.8		10.8	
[Gowal et al., 2020]	+ FT	91.2	1.3	53.9	-9.0	76.0	8.8	56.9	46.1	50.1	39.3
WRN-70-16 - l_∞ (*)		90.7		65.6		66.9		8.1		8.1	
[Gowal et al., 2020]	+ FT	91.6	0.9	54.3	-11.3	78.2	11.3	58.3	50.2	51.2	43.1
Fine-tuning l_2-robust models											
RN-50 - l_2		91.5		29.7		70.3		27.0		23.0	
[Engstrom et al., 2019a]	+ FT	87.8	-3.7	43.1	13.4	70.8	0.5	50.2	23.2	41.7	18.7
RN-50 - l_2 (*)		91.1		37.7		73.4		31.2		28.8	
[Augustin et al., 2020]	+ FT	87.0	-4.1	47.2	9.5	70.4	-3.0	54.1	22.9	46.0	17.2
WRN-70-16 - l_2 (*)		94.1		43.1		81.7		34.6		32.4	
[Gowal et al., 2020]	+ FT	91.2	-2.9	51.9	8.8	79.2	-2.5	58.8	24.2	49.7	17.3
Fine-tuning l_1-robust models											
RN-18 - l_1		87.1		22.0		64.8		60.3		22.0	
	+ FT	83.5	-3.6	40.3	18.3	68.1	3.3	55.7	-4.6	40.1	18.1

averaging the robust error on the batches where the respective attack has been performed, thus no extra attacks are necessary), such that the probability for sampling the l_p-threat model is

$$\frac{\text{rerr}_p}{\text{rerr}_1 + \text{rerr}_\infty}, \quad \text{for } p \in \{1, \infty\}. \tag{7.3}$$

The motivation for this sampling scheme is that the robust error in the union Δ is mainly influenced by the worst threat model. We show in more details the effect of the biased sampling in E-AT fine-tuning in Sec. 7.1.6.

7.1.3 Scaling up multiple-norm robust models

Fine-tuning robust classifiers. The fact that multiple-norm robustness can be achieved via a short fine-tuning allows to use large architectures, which would be hard and expensive to train from random initialization in this more difficult threat model. Moreover, fine-tuning with E-AT has similar computational cost per epoch as standard adversarial training, and since we aim at efficiency, we therefore use it as main tool on large models.

Experimental details: In the following we fine-tune with E-AT[1] for 3 epochs on CIFAR-10 and 1 epoch on ImageNet-1k, starting with learning rate 0.05 or 0.01, depending on the model, and decreasing by a factor of 10 every 1/3 of the total number of finetuning epochs. We do 10 steps of APGD in adversarial training for CIFAR-10, while 5 and 15 with l_∞ and l_1 respectively on ImageNet as optimizing in the l_1-ball requires more iterations in that case. When the model was originally trained with extra data beyond the training set on CIFAR-10, we use the 500k images introduced by Carmon et al. [2019] as additional data for fine-tuning (see also Sec. 7.1.5).

CIFAR-10: RobustBench (see Sec. 5.2) provides a collection of the currently most robust classifiers. We took a subset of the most robust models, among those which do not use synthetic

[1]Code available at https://github.com/fra31/robust-finetuning.

Table 7.3: **ImageNet - Results of one epoch of E-AT fine-tuning of existing robust models:** We use existing models trained to be robust wrt a single l_p-ball (either l_∞ or l_2) and fine-tune them for a single epoch for multiple-norm robustness with our E-AT scheme.

model	*clean*		l_∞ ($\epsilon_\infty = \frac{4}{255}$)		l_2 ($\epsilon_2 = 2$)		l_1 ($\epsilon_1 = 255$)		union	
Fine-tuning l_∞-robust models										
RN-50 - l_∞	62.9		29.8		17.7		0.0		0.0	
[Engstrom et al., 2019a] + FT	58.0	-4.9	27.3	-2.5	41.1	23.4	24.0	24.0	21.7	21.7
RN-50 - l_∞	68.2		36.7		15.6		0.0		0.0	
[Bai et al., 2021] + FT	60.1	-8.1	29.2	-7.5	42.1	26.5	24.5	24.5	22.6	22.6
DeiT-S - l_∞	66.4		35.6		40.1		3.1		3.1	
[Bai et al., 2021] + FT	62.6	-3.8	32.2	-3.4	46.1	6.0	24.8	21.7	23.6	20.5
XCiT-S - l_∞	72.8		41.7		45.3		2.7		2.7	
[Debenedetti, 2022] + FT	68.0	-4.8	36.4	-5.3	51.3	6.0	28.4	25.7	26.7	24.0
Fine-tuning l_2-robust models										
RN-50 - l_2	58.7		25.0		40.5		14.0		13.5	
[Engstrom et al., 2019a] + FT	56.7	-2.0	26.7	1.7	41.0	0.5	25.4	11.4	23.1	9.6

Table 7.4: **ImageNet - 1 epoch of fine-tuning of existing robust models:** We use existing models trained to be robust wrt a single l_p-ball (either l_∞ or l_2) and fine-tune them for a single epoch for multiple-norm robustness with SAT and E-AT.

model	*clean*	l_∞	l_2	l_1	*union*
RN-50 - l_∞-AT	62.9	29.8	17.7	0.0	0.0
+ SAT	59.4	26.5	38.8	21.1	19.4
+ E-AT	58.0	27.3	41.1	24.0	21.7
RN-50 - l_2-AT	58.7	25.0	40.5	14.0	13.5
+ SAT	57.7	25.9	41.6	23.2	21.1
+ E-AT	56.7	26.7	41.0	25.4	23.1

data, for l_2- and l_∞-norm and the l_1-robust one introduced in Sec. 4.4.5 (all are trained with the same radii ϵ_p as in our experiment). Note that we use the classifiers from Gowal et al. [2020] (instead of those from Rebuffi et al. [2021b]) since those were the best available ones at the time of the start of this project. We present in Table 7.2 the results. First of all the fine-tuning works for all tested architectures and results in many cases in stronger robustness in the union than for the specifically trained WideResNet-28-10 models (see Table 7.14). In particular, the most robust l_∞-model from Gowal et al. [2020] with 65.6% l_∞-robustness and only 8.1% l_1-robustness can be fine-tuned to a multiple-norm robust model with 51.2% robustness which is up to our knowledge the best reported multiple-norm robustness. While in general it is expected that larger architectures and extra data improve robustness (see e.g. RobustBench leaderboards), we could achieve such improvement without the high computational cost (and potential instabilities) of training large networks on an extended dataset from scratch. Very interesting is that the l_2-robustness of 78.2% is quite close to the 81.7% l_2-robustness of the specifically l_2-trained model from Gowal et al. [2020]. Moreover, the l_1-robustness of 58.3% is close to the best reported one of 60.3% in Sec. 4.4 (however we improve this a lot in the next section) and the model has even higher clean accuracy. Clearly, this comes at the price of a significant loss in l_∞ but this is to be expected. Striking is that fine-tuning the l_2-robust model from Gowal et al. [2020] results in a very similar result. Finally, we observe that the l_∞-threat model is the most challenging one, and fine-tuning l_∞-robust models yields the best robust accuracy in the union (when comparing models with the same architecture). In a nutshell, E-AT fine-tuning of existing l_p-robust models yields very efficient and competitive baselines for future research in this area.

ImageNet: We start with the l_2- resp. l_∞-robust models from Engstrom et al. [2019a], Bai et al. [2021] and Debenedetti [2022] including the vision transformers DeiT small [Touvron et al., 2021] and XCiT small [El-Nouby et al., 2021] We use $\epsilon_2 = 2$ for the experiments as the robust accuracy is still in a reasonable range of 40% and together with our choice of $\epsilon_1 = 255$ and the

Table 7.5: **CIFAR-10 - Training from random initialization:** For full training of PreAct ResNet-18 with different methods for multiple norm robustness, we show robust accuracy in each l_p-threat model, their average, and the robustness in their union.

model	*clean*	l_∞	l_2	l_1	*avg.*	*union*
l_∞-AT	84.0	48.1	59.7	6.3	38.0	6.3
l_2-AT	88.9	27.3	68.7	25.3	40.5	20.9
l_1-AT	85.9	22.1	64.9	59.5	48.8	22.1
SAT	83.9	40.7	68.0	54.0	54.2	40.4
AVG	84.6	40.8	68.4	52.1	53.8	40.1
MAX	80.4	45.7	66.0	48.6	53.4	44.0
MSD	81.1	44.9	65.9	49.5	53.4	43.9
E-AT	81.9	43.0	66.4	53.0	54.2	42.4

standard $\epsilon_\infty = \frac{4}{255}$ the l_2-radius from Theorem 3.3.1 is almost exactly 2. The initial l_∞-models are completely non-robust for l_1 but achieve, after fine-tuning, over 24% l_1-robust accuracy and also the l_2-robust accuracy improves, at the price of a relatively small loss in l_∞-robust and clean accuracy. Interestingly, the DeiT-S and XCiT-S models has already high robustness wrt l_2, unlike the RN-50s, which further improves thanks to E-AT, and the latter attains the best robustness in the union. For the l_2-robust model all robust accuracies improve as the original model was trained for $\epsilon = 3$. Up to our knowledge no multiple-norm robustness has been reported before for ImageNet and thus these results are an important baseline. Finally, we show in Table 7.4 that both SAT and E-AT are effective for fine-tuning on ImageNet, and E-AT achieves the best robustness in the union (we omit the other methods since they are computationally more expensive). We also observe that in this case l_1 is the most challenging threat model, and the best robustness in the union is achieved when fine-tuning the classifier (among those using RN-50 as architecture) trained wrt l_2 which already has non-trivial robustness wrt l_1.

Additional experiments: Sec. 7.1.6 contains further studies and details about fine-tuning with E-AT, e.g. we report runtime and show that fine-tuning a naturally trained model does not provide competitive robustness and leads to low clean accuracy. Moreover, we show the stability of the scheme over random seeds, that increasing the number of epochs progressively improves the robustness in the union, and that even models trained to be robust wrt perceptual metrics can be used for E-AT fine-tuning.

Full training for multiple norm robustness from random initialization. We evaluate the performance of the different methods for multiple-norm robustness when applied for full training from random initialization on CIFAR-10 (using the same ϵ_p as above). Table 7.5 reports the results, averaged over 3 runs, in every threat model: MAX and MSD attain the best robustness in the union, and E-AT is close to them and outperforms SAT. We recall that E-AT is 2-3x less expensive than MSD and MAX (see Table 7.1). We also include the performance of models trained to be robust for single norms, which do not show high robustness in the union of the threat models. More details and further experiments with WideResNet-28-10 as architecture can be found in Sec. 7.1.5 and Sec. 7.1.6.

7.1.4 Fine-tuning l_p-robust models to become l_q-robust for $p \neq q$

Motivated by our results for the multiple-norm threat model we study to which extent we can fine-tune an l_p-robust model to a l_q-robust model with $p \neq q$. Again the emphasis is on an extremely short fine-tuning time so that this is much faster than full adversarial training.

CIFAR-10: We fine-tune for 3 epochs the most l_∞-robust model at $\epsilon_\infty = \frac{8}{255}$ of Gowal et al. [2020] with adversarial training wrt l_2 and l_1 with $\epsilon_2 = 0.5$ and $\epsilon_1 = 12$. Table 7.6 shows that fine-tuning for l_1-robustness yields 70.2% l_1-robust accuracy which is 9.9% more than the previously most robust model with a small computational cost (to be fair with a larger architecture and using extra data). Also we get a strikingly high l_2-robust accuracy for the l_2-fine-tuned model of 80.0% not far away from the 81.7% which ones gets by training for l_2 from scratch. Surprisingly, fine-tuning the l_2-robust model of Gowal et al. [2020] wrt l_1 does not outperform the l_1-robustness achieved by fine-tuning their l_∞-robust model. Interestingly, fine-tuning the l_1-robust PreAct

Table 7.6: **Fine-tuning l_p-robust models to another threat model:** For each norm we fine-tune the most robust models wrt the other ones for 3 epochs for CIFAR-10 and 1 epoch for ImageNet and report clean and robust accuracy for all threat models. Even for the threat models where the robustness of the original model is low, the fine-tuning is sufficient to yield robustness almost at the same level of the specialized models with same architecture. For each threat model (column) we highlight in blue the model trained for the specific norm, in orange those only fine-tuned in the target norm. The values of the thresholds ϵ are the same used the multiple norms experiments.

	CIFAR-10				**ImageNet**				
	clean	l_∞	l_2	l_1		*clean*	l_∞	l_2	l_1
WRN-70-16 [Gowal et al., 2020] - l_∞ (*)					DeiT-S [Bai et al., 2021] - l_∞				
original	90.7	65.6	66.9	8.1	original	66.4	35.6	40.1	3.1
+ FT wrt l_2	92.8	47.4	80.0	34.0	+ FT wrt l_2	66.5	31.2	46.1	9.6
+ FT wrt l_1	92.4	33.9	74.7	**70.2**	+ FT wrt l_1	61.0	23.9	42.9	30.1
WRN-70-16 [Gowal et al., 2020] - l_2 (*)					XCiT-S [Debenedetti, 2022] - l_∞				
original	94.1	43.1	81.7	34.6	original	72.8	41.7	45.3	2.7
+ FT wrt l_∞	92.3	58.5	73.5	11.4	+ FT wrt l_2	71.5	35.9	51.4	9.5
+ FT wrt l_1	92.8	29.2	75.7	68.9	+ FT wrt l_1	65.8	25.2	47.1	**33.9**
RN-18 - l_1					RN-50 [Engstrom et al., 2019a] - l_2				
original	87.1	22.0	64.8	60.3	original	58.7	25.0	40.5	14.0
+ FT wrt l_∞	82.7	44.2	66.6	25.4	+ FT wrt l_∞	59.1	31.5	40.1	7.5
+ FT wrt l_2	88.0	31.0	69.8	39.7	+ FT wrt l_1	56.8	18.0	37.1	28.7

ResNet-18 for l_2 yields a better l_2-robustness than l_2-training from scratch, see Table 7.14. This shows again that fine-tuning of existing models requires minimal effort and already provides strong baselines for adversarial robustness obtained by adversarial training from scratch and in some cases even outperforms them.

ImageNet: We fine-tune the l_2- RN-50 and the l_∞-robust transformers DeiT-S and XCiT-S used in the previous section to the other threat model respectively and wrt l_1. The results are in Table 7.6 (those for a RN-50 robust wrt l_∞ in Sec. 7.1.6). For all models it is possible to achieve within one epoch of fine-tuning a non-trivial robustness in every threat model. In particular, note that those originally trained for the l_∞-threat model have very low robustness wrt l_1, while after fine-tuning XCiT-S achieves 33.9% robust accuracy, even higher than what is obtained from the l_2-robust RN-50 i.e. 28.7%. Note that compared to fine-tuning for multiple-norm robustness (see Table 7.3), fine-tuning XCiT-S specifically for l_2 yields almost the same robust accuracy but 3.5% better clean performance, while, when fine-tuning it for l_1, the l_1-robust accuracy is 5.5% higher. Up to our knowledge our ImageNet models are the first ones for which l_1-robustness is reported.

7.1.5 Experimental details

For the comparison of training schemes we use for multiple-norm robustness we train PreAct ResNet-18 with softplus activation function for 80 epochs with initial learning rate of 0.05 reduced by a factor of 10 after 70 epochs. When training WideResNet-28-10 we use a cyclic schedule for the learning rate with maximum value 0.1 for 30 epochs. We use SGD optimizer with momentum of 0.9 and weight decay of $5 \cdot 10^{-4}$, batch size of 128. We use random cropping and horizontal flipping as augmentation techniques. For adversarial training of models robust wrt a single norm and with SAT and our novel scheme E-AT we use APGD with default parameters, while for the retrained AVG, MAX, and MSD we use PGD for l_∞ (step size $\epsilon_\infty/4$) and l_2 (step size $\epsilon/3$), SLIDE [Tramèr and Boneh, 2019] for l_1 (standard parameters). For all methods we use 10 steps for the inner maximization problem in adversarial training (note that AVG and MAX repeat the attack for all threat models, and MSD tests multiple steps, thus they are more expensive). For all schemes we select the best performing checkpoint for the comparison (that is the one with the highest robustness) when using the piecewise schedule, the final checkpoint with the cyclic schedule. Moreover, we use the TRADES-XENT loss (TRADES loss [Zhang et al., 2019b] with adversarial points maximizing the cross-entropy loss) since Gowal et al. [2020] show that this

gives slightly better robustness on CIFAR-10 without additional data, while we use standard adversarial training [Madry et al., 2018] for PreAct ResNet-18. We train the classifiers of MNG-AC [Madaan et al., 2021] with the original code, where we set $\epsilon_1 = 12$ and rescale the step size linearly. Finally, for the runtime comparison we run each method on a single Tesla V100 GPU.

For fine-tuning on CIFAR-10 we use 3 epochs and the same setup as for full training except for the learning rate schedule, since in this case we use as initial value the best performing one in $\{0.01, 0.05\}$ (the larger value works best for the smaller networks) and reduce it by a factor of 10 at the beginning of each epoch. When the model was originally trained with extra data beyond the training set on CIFAR-10, we use the 500k images introduced by Carmon et al. [2019] as additional data for fine-tuning, and each batch is split equally between standard and extra images, and we count 1 epoch when the whole standard training set has been used: note that in this way, using only 3 epochs not the whole pseudo-labelled dataset is exploited.

For fine-tuning on ImageNet we use 1 epoch, initial learning rate of 0.01 (0.0005 for XCiT-S), reduced by a factor of 10 every 1/3 of training steps. We follow the setup of Engstrom et al. [2019a] for data augmentation and setting batch size to 256 (except for the RN-50 from Bai et al. [2021] and XCiT-S for which we use 192 to fit into the GPU memory) and weight decay to 10^{-4}. For adversarial training we use APGD with 5 steps for l_∞ and l_2, 15 steps for l_1 since optimizing in the l_1-ball intersected with the box constraints is more challenging, see Sec. 4.4.

7.1.6 Addtional analysis and experiments

We here analyze in more details our E-AT scheme and expand the comparison to existing methods presented above.

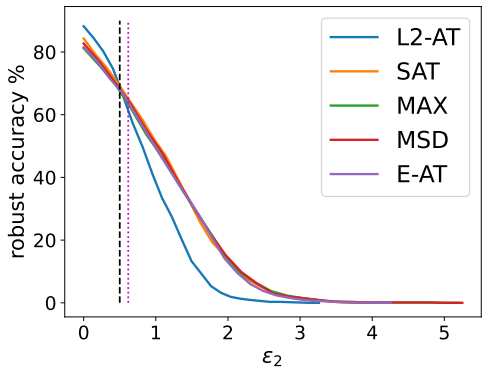

Figure 7.3: l_2-robustness curves of a model trained with l_2-adversarial training (AT) for $\epsilon_2 = 0.5$ and methods for multiple norm robustness on CIFAR-10. Our E-AT is expected to yield robustness at $\epsilon_2 = 0.62$. **Although l_2-attacks are not used for training, our extreme-adv. training scheme E-AT yields l_2-robustness similar to the other methods and even to the one obtained with specific l_2-adversarial training.**

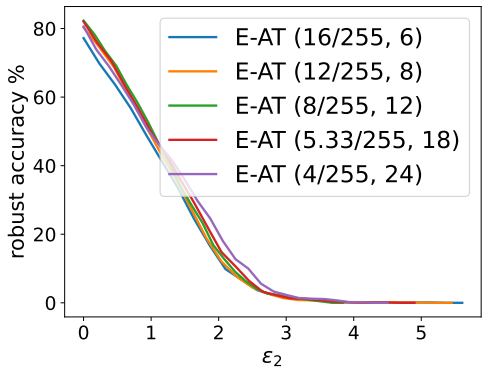

Figure 7.4: l_2-robustness curves of models obtained fine-tuning an l_∞-robust RN-18 with E-AT on CIFAR-10 with different combinations $(\epsilon_\infty, \epsilon_1)$ of radii of the l_∞- and l_1-ball.

Robustness wrt l_2 of E-AT. To show the effect of E-AT on l_2-robustness we plot in Fig. 7.3 the robust accuracy wrt l_2 computed with FAB (Sec. 4.1), which minimizes the size of the perturbations, as a function of the threshold ϵ_2 for a PreAct ResNet-18 trained with either l_2-AT at $\epsilon_2 = 0.5$ or methods for multiple norm robustness (see complete results for such models below). Theorem 3.3.1 suggests that the extreme norms training provides robustness at $\epsilon_2 \approx 0.62$, which is confirmed by the plots. Although no l_2-attack has been used during training by the E-AT model, it has robustness wrt l_2 similar to that both of the techniques for multiple norms which use l_2-perturbations at training time and of the classifier specifically trained for such threat model.

Effect of varying the radii of the l_∞- and l_1-ball on the robustness wrt l_2. We study the effect of varying the radii of the l_∞- and l_1-balls in E-AT on the robustness wrt l_2 of the

resulting classifier. We fine-tune with E-AT the l_∞-robust RN-18 (trained with $\epsilon_\infty = 8/255$) for 3 epochs with different pairs $(\epsilon_\infty, \epsilon_1)$ such that $\sqrt{\epsilon_\infty \cdot \epsilon_1}$ is constant at ≈ 0.62, i.e. inducing, ignoring α, the same ϵ_2 according to Eq. (7.2) (convex hull of the union), but very different ϵ_2 for Eq. (7.1) (union only), ranging from 0.13 (16/255, 6) to 0.44 (4/255, 24). Fig. 7.4 shows the l_2-robustness curves computed with FAB. For all combinations the l_2-robustness curves are similar, even for the smaller ϵ_1 (the small drops are related to the lower clean accuracy). Thus the statement of Theorem 3.3.1 (convex hull) formulated for linear classifiers holds empirically also for deep networks.

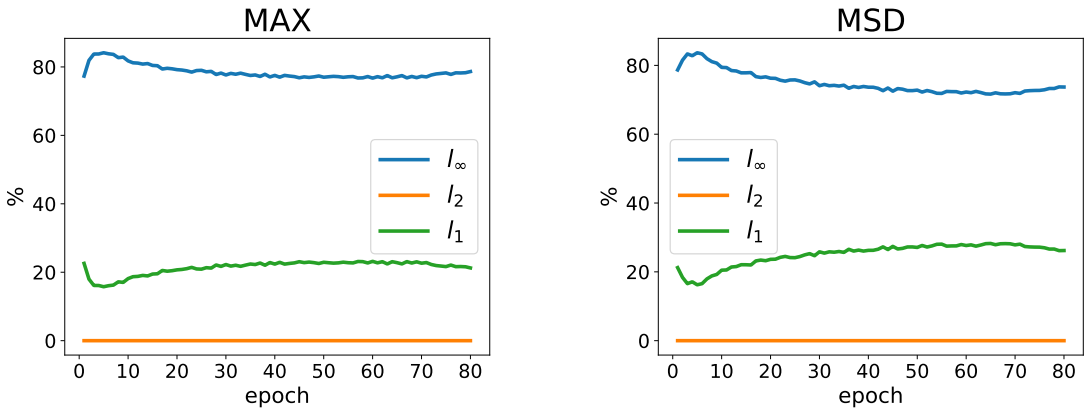

Figure 7.5: **CIFAR-10, ResNet18. Left:** For MAX-training we show for each epoch during training the percentage of points attaining for the indicated l_p-threat model ($p \in \{1, 2, \infty\}$) the highest loss over the the three threat models. **Right:** For MSD-training we show the percentage of steps taken wrt each threat model over epochs (note that MSD does the steepest descent step for each l_p-threat model and then realizes the one yielding maximal loss).

Confirmation that adversarial training with the extreme norms is sufficient: Analysis of MAX and MSD training. Both MAX and MSD schemes perform adversarial training considering all the threat models simultaneously, and we analyze here their training procedure in more detail. Fig. 7.5 shows for the MAX strategy how many times, in percentage, for each epoch the points computed for each threat model realize the maximal loss over the three attacks (l_1, l_2, l_∞), and then are subsequently used for the update of the model. Similarly, for MSD, we show the frequency with which a step wrt each l_p-norm is taken when computing the adversarial points (average over all iterations and training points). In both cases the l_∞-threat model is the most used one with the l_1-threat model being used $3-4$ times less often. However, the l_2-threat model is almost never chosen. This empirically confirms the analysis from Theorem 3.3.1 which shows that training only wrt l_∞ and l_1 is (at least for a linear classifier) sufficient to achieve l_2-robustness for the chosen ϵ_2 and thus during training no extra updates wrt the l_2-threat model are necessary. This is in line with the results reported in Fig. 7.3 which show that for different thresholds the l_2-robustness achieved by E-AT-training is similar to that of standard l_2-training.

Robustness against unseen non l_p-bounded adversarial attacks. We investigate on CIFAR-10 to which extent adversarial robustness achieved with multiple norms training generalizes to unseen and possibly very different threat models. We select three sparse attacks (l_0-bounded, patches and frames) and five adversarial corruptions (fog, snow, Gabor noise, elastic, l_∞-JPEG) from Kang et al. [2019b]. Additionally, we compute the accuracy of the classifiers on the common corruptions, i.e. not adversarially optimized, of CIFAR-10-C [Hendrycks and Dietterich, 2019]. Table 7.7 shows the results against such attacks of WRN-28-10 adversarially trained either wrt single norms or for multiple norm robustness. Additionally, we include the PAT model from Laidlaw et al. [2021], which uses a RN-50 as architecture, is obtained with perceptual adversarial training, and has been shown to be robust to unseen attacks, and a normally trained model (NAT) as baseline. The models trained wrt multiple norms show much higher robustness in both the union (worst case robustness over all threat models, excluding

Table 7.7: **CIFAR-10 - Robustness against non l_p-bounded attacks:** We test the robustness of WRN-28-10 trained in different threat models against unseen types of attacks. Moreover, we add the PAT model from Laidlaw et al. [2021], which uses RN-50 as architecture.

model	*clean*	*comm. corr.*	l_0	*patches*	*frames*	*fog*	*snow*	*gabor*	*elastic*	*jpeg*	*avg.*	*union*
NAT	94.4	71.6	0.1	8.1	2.6	47.3	3.9	35.0	0.2	0.0	12.2	0.0
l_∞-AT	81.9	72.6	7.3	21.6	26.2	36.0	35.9	52.5	59.4	5.1	30.5	2.0
l_2-AT	87.8	79.2	13.2	25.0	17.7	44.9	22.1	43.5	56.6	14.0	29.6	4.5
l_1-AT	83.5	75.0	40.9	41.3	21.1	35.6	20.6	41.2	53.3	25.5	34.9	8.6
PAT	82.6	76.9	23.3	37.9	21.7	53.5	25.6	41.8	53.5	13.7	33.9	8.0
SAT	80.5	72.0	38.7	36.7	29.3	33.5	29.0	49.8	57.0	37.4	38.9	13.8
AVG	82.0	73.6	39.7	36.8	30.8	37.2	21.1	49.9	58.1	30.4	38.0	10.9
MAX	80.1	71.3	35.1	34.6	32.7	34.5	35.0	53.4	58.5	33.5	39.7	15.3
MSD	81.0	71.7	36.9	35.0	31.8	34.6	26.4	51.5	59.7	33.4	38.7	12.9
E-AT	79.1	71.3	39.5	37.7	30.5	34.8	33.4	50.2	58.6	38.7	40.4	15.9

common corruptions) and average robustness against these attacks: in particular, MAX and E-AT achieve almost twice as high robustness in the union than the best of the models not trained for multiple-norm robustness, with only little loss in clean accuracy compared to the other robust classifiers. Moreover, for almost all threat models, these classifiers attain the highest robustness or are close to it, while the best among the single l_p-robust models varies. Therefore training for multiple norm robustness allows to generalize to some extent beyond the threat model used during training. Finally, E-AT outperforms even the PAT model which is trained wrt LPIPS and aims at generalization to unseen attacks. More details in Sec. 7.1.6.

To test robustness against l_0-attacks we use Sparse-RS (Sec. 6.2) with a budget of 18 pixels and 10k queries. We adopt patches of size 5×5 pixels, optimized with the PGD-based attack from Rao et al. [2020] (without constraints on the position of the patch on the images), and frames of width 1 pixel [Zajac et al., 2019], again optimized with PGD: in both cases we use 10 random restarts of 100 iterations. For the adversarial corruptions we use the original implementation [Kang et al., 2019b] with 100 iterations and search for a budget ϵ for which the models show different levels of robustness (in details, for fog $\epsilon = 128$, snow $\epsilon = 0.5$, Gabor noise $\epsilon = 60$, elastic $\epsilon = 0.125$, l_∞-JPEG $\epsilon = 0.25$). Finally, we average the classification accuracy over the 5 severities of the common corruptions. All the statistics are on 1000 test points.

Multiple-norm robustness via fast fine-tuning. We here show that with short fine-tuning of an l_p-robust model it is possible to achieve competitive multiple-norm robustness for any $p \in \{\infty, 2, 1\}$. Table 7.8 shows that all methods for robustness in the union are able to improve in 3 epochs the robustness of l_p-robust classifiers to multiple threat models on CIFAR-10. Moreover, one can see that the l_∞-threat model is the most challenging, and fine-tuning robust models wrt it yields the best robust accuracy in the union. Moreover, E-AT outperforms SAT which has roughly the same computational cost (the training time per epoch can be found in Table 7.14).

Fine-tuning natural models. We fine-tune with E-AT for 3 epochs PreAct ResNet-18 either naturally trained or robust wrt a single l_p-norm. Table 7.9 shows clean and robust accuracy for each threat model for the initial classifier and after E-AT fine-tuning: while for all robust models the fine-tuning yields values competitive with the full training for multiple norms (see Table 7.14), starting from a standard model leads to significantly lower both clean performance and robustness in the union of the three l_p-balls.

Runtime with large models. We reported in Table 7.1 and Table 7.14 the runtime per epoch of E-AT. For larger architectures the computational cost increases significantly, and adversarial training with the WideResNet-70-16, the largest one we consider, on CIFAR-10 takes, in our experiments, around 6100 s per epoch when using only the training set and over 10000 s if the unlabelled data is used (since twice more training steps are effectively used). Moreover, fine-tuning on ImageNet takes around 6 h per epoch for l_∞ and l_2 (5 steps of adversarial training), 20 h per epoch for l_1 (15 steps), 13.5 h per epoch with E-AT (with the architectures we consider).

Table 7.8: **CIFAR-10 - Fine-tuning for multiple-norm robustness:** We fine-tune with different methods for multiple norms for 3 epochs the PreAct ResNet-18 robust wrt individual norms (mean and standard deviation of the clean and robust accuracy over 5 seeds is reported).

model	*clean*	l_∞ ($\epsilon_\infty = \frac{8}{255}$)	l_2 ($\epsilon_2 = 0.5$)	l_1 ($\epsilon_1 = 12$)	*average*	*union*
RN-18 - l_∞-AT	83.7	48.1	59.8	7.7	38.5	7.7
+ SAT	83.5 ± 0.2	43.5 ± 0.2	68.0 ± 0.4	47.4 ± 0.5	53.0 ± 0.2	41.0 ± 0.3
+ AVG	84.2 ± 0.4	43.3 ± 0.4	68.4 ± 0.6	46.9 ± 0.6	52.9 ± 0.4	40.6 ± 0.4
+ MAX	82.2 ± 0.3	45.2 ± 0.4	67.0 ± 0.7	46.1 ± 0.4	52.8 ± 0.3	42.2 ± 0.6
+ MSD	82.2 ± 0.4	44.9 ± 0.3	67.1 ± 0.6	47.2 ± 0.6	53.0 ± 0.4	42.6 ± 0.2
+ E-AT unif.	82.6 ± 0.6	44.4 ± 0.4	68.0 ± 0.3	48.5 ± 1.0	53.6 ± 0.4	42.2 ± 0.3
+ E-AT	82.7 ± 0.4	44.3 ± 0.6	68.1 ± 0.5	48.7 ± 0.5	53.7 ± 0.3	42.2 ± 0.8
RN-18 - l_2-AT	88.2	29.8	68.6	27.5	42.0	23.1
+ SAT	86.8 ± 0.3	38.2 ± 0.4	69.6 ± 0.7	49.1 ± 0.6	52.3 ± 0.3	37.2 ± 0.4
+ AVG	86.9 ± 0.2	37.8 ± 0.4	70.2 ± 0.6	48.3 ± 0.5	52.1 ± 0.4	36.8 ± 0.4
+ MAX	85.1 ± 0.8	42.1 ± 0.3	69.4 ± 0.2	45.6 ± 0.5	52.4 ± 0.2	40.0 ± 0.3
+ MSD	85.3 ± 0.4	42.0 ± 0.7	69.2 ± 0.2	44.0 ± 0.4	51.8 ± 0.4	39.0 ± 0.5
+ E-AT unif.	85.9 ± 0.5	40.0 ± 0.6	69.4 ± 0.7	50.3 ± 0.4	53.2 ± 0.3	38.9 ± 0.8
+ E-AT	85.8 ± 0.7	40.7 ± 0.9	69.5 ± 0.5	49.5 ± 0.5	53.3 ± 0.6	39.4 ± 0.7
RN-18 - l_1-AT	87.1	22.0	64.8	60.3	49.0	22.0
+ SAT	85.1 ± 0.3	38.4 ± 0.6	68.3 ± 0.4	55.0 ± 0.7	53.9 ± 0.2	38.2 ± 0.6
+ AVG	86.0 ± 0.2	38.7 ± 0.5	68.4 ± 0.3	55.6 ± 0.6	54.2 ± 0.2	38.5 ± 0.5
+ MAX	82.2 ± 0.3	42.7 ± 0.5	66.8 ± 0.5	48.1 ± 0.4	52.5 ± 0.3	41.5 ± 0.4
+ MSD	81.8 ± 0.7	42.7 ± 0.5	66.8 ± 0.4	47.9 ± 0.6	52.5 ± 0.2	41.5 ± 0.5
+ E-AT unif.	83.5 ± 0.7	39.8 ± 0.5	68.0 ± 0.2	55.8 ± 0.5	54.6 ± 0.2	39.6 ± 0.5
+ E-AT	83.6 ± 0.6	40.5 ± 0.4	68.1 ± 0.1	55.3 ± 0.4	54.6 ± 0.2	40.3 ± 0.3

This shows how transferring robustness with fine-tuning might allow to obtain classifiers robust wrt different threat models fast and at much lower computational cost.

Effect of biased sampling scheme. We compare the biased sampling scheme of E-AT introduced in Eq. (7.3) to a uniform one, where one chooses uniformly at random which l_p, with $p \in \{1, \infty\}$, attack to use for adversarial training for each batch. This is referred to as E-AT unif. in Table 7.8, where one case see that the biased sampling schemes yields slightly better results when fine-tuning the l_2- and in particular the l_1-robust model: we hypothesize that, since l_∞ is the most challenging threat model, it is important to use it more often at training time when the initial model is non robust wrt l_∞. Moreover, we test E-AT unif. for full training (see Table 7.14).

Results using different number of epochs. Table 7.10 shows the effect of our E-AT-fine-tuning for different numbers of epochs on a RN-18 either robust wrt l_∞ or naturally trained. For the robust model, with longer training the clean accuracy progressively improves, as well as the robustness in the union of the threat models. In particular, the models fine-tuned for 15 epochs has robust accuracy similar to that achieved by the MAX-training (43.2% compared to 43.3%, see Table 7.14), while still being significantly faster even considering the training time of the initial model. When starting from a standard model, E-AT fine-tuning leads to a large drop in clean accuracy while the robustness in the union remains lower than what can be achieved with robust models, even when using 15 epochs. Similarly, we fine-tune for 3 epochs, instead of 1 as done above, the l_2-robust model on ImageNet from Engstrom et al. [2019a] with our E-AT. Table 7.11 shows that the longer fine-tuning improves all the performance metrics between 0.6% and 1.3%. Moreover, since we use a single epoch of fine-tuning on ImageNet, we test its effect on CIFAR-10. In Table 7.12 we fine-tune with E-AT models adversarially trained wrt a single norm for 1 epoch: this is sufficient to significantly increase the robustness in the union of the threat models, which gets close to that obtained with the standard 3 epochs (differences are in the range 0.6% to 2.5%). In particular, the l_∞-robust classifier is again the most suitable for the fine-tuning, since it has been trained in the most challenging threat model.

Table 7.9: **CIFAR-10 - 3 epochs of fine-tuning with E-AT :** We report the results of fine-tuning PreAct ResNet-18 models to become robust wrt the union of the threat models. Fine-tuning any l_p-robust model leads to competitive clean and robust accuracy to full training, differently from using a naturally trained model.

model	*clean*		l_∞ ($\epsilon_\infty = \frac{8}{255}$)		l_2 ($\epsilon_2 = 0.5$)		l_1 ($\epsilon_1 = 12$)		*union*	
RN-18 - standard	94.4		0.0		0.0		0.0		0.0	
+ FT	66.6	-27.8	29.9	29.9	50.1	50.1	38.5	38.5	29.8	29.8
RN-18 - l_∞	83.7		48.1		59.8		7.7		7.7	
+ FT	82.3	-1.4	43.4	-4.7	68.0	8.2	48.0	40.3	41.2	33.5
RN-18 - l_2	88.2		29.8		68.6		27.5		23.1	
+ FT	85.4	-2.8	40.6	10.8	69.8	1.2	48.7	21.2	39.1	16.0
RN-18 - l_1	87.1		22.0		64.8		60.3		22.0	
+ FT	83.5	-3.6	40.3	18.3	68.1	3.3	55.7	-4.6	40.1	18.1

Table 7.10: **CIFAR-10 - Fine-tuning for more epochs:** We show the effect of fine-tuning for different number of epochs (3 is the standard we use) the PreAct ResNet-18 (standard or robust wrt l_∞).

model	*clean*		l_∞ ($\epsilon_\infty = \frac{8}{255}$)		l_2 ($\epsilon_2 = 0.5$)		l_1 ($\epsilon_1 = 12$)		*union*	
RN-18 - l_∞	83.7		48.1		59.8		7.7		7.7	
+ 3 epochs FT	82.3	-1.4	43.4	-4.7	68.0	8.2	48.0	40.3	41.2	33.5
+ 5 epochs FT	83.0	-0.7	45.2	-2.9	68.8	9.0	50.1	42.4	43.1	35.4
+ 7 epochs FT	83.1	-0.6	44.6	-3.5	68.7	8.9	50.4	42.7	42.6	34.9
+ 10 epochs FT	84.0	0.3	44.9	-3.2	69.2	9.4	51.0	43.3	42.8	35.1
+ 15 epochs FT	84.6	0.9	44.9	-3.2	69.5	9.7	52.1	44.4	43.2	35.5
RN-18 - standard	94.4		0.0		0.0		0.0		0.0	
+ 3 epochs FT	66.6	-27.8	29.9	29.9	50.1	50.1	38.5	38.5	29.8	29.8
+ 5 epochs FT	70.6	-23.8	33.8	33.8	55.2	55.2	44.4	44.4	33.4	33.4
+ 7 epochs FT	72.1	-22.3	36.1	36.1	58.9	58.9	45.9	45.9	35.6	35.6
+ 10 epochs FT	75.4	-19.0	37.1	37.1	61.0	61.0	47.9	47.9	36.9	36.9
+ 15 epochs FT	76.0	-18.4	40.2	40.2	61.6	61.6	49.2	49.2	40.0	40.0

Fine-tuning perceptually robust models We test the effect of E-AT fine-tuning on a model trained to be robust to perturbations which are aligned with human perception. In particular, we use the classifier obtained with perceptual adversarial training (PAT), i.e. wrt the LPIPS metric, from Laidlaw et al. [2021], and compare it to two models with the same architecture (ResNet-50) adversarially trained wrt l_∞ and l_2. Table 7.13 shows the robustness in every threat model for the original models and those obtained with 3 epochs of E-AT fine-tuning. The PAT classifier has initially the highest robustness in the union, confirming the observation of Laidlaw et al. [2021] that PAT provides some robustness to unseen attacks. After fine-tuning, all three models achieve similar worst-case robustness, with the classifier originally l_∞-robust being slightly better. This shows that our E-AT fine-tuning is effective even when applied to models adversarially trained not wrt an l_p-norm.

Full training for multiple norm robustness. We report in Table 7.14 the detailed results of training for multiple norms robustness from random initialization with PreAct ResNet-18 and WideResNet-28-10. For both architectures MAX and MSD attain the best results in the union. However, E-AT is only slightly worse while being 2-3 times less expensive. We include MNG-AC from Madaan et al. [2021] for the larger architecture as that is used also in the original paper. Moreover, since it has PreAct ResNet-18 as architecture, we additionally include the original MSD model of Maini et al. [2020], marked with (*), which obtains 41.4% robustness in the union, whereas with our reimplementation we get 43.9%, significantly improving their results. Note that they reported in their paper 47.0% robustness in the union, while our APGD-based evaluation reduces this to 41.4% which shows that our robustness evaluation is significantly stronger. Finally, we add the results of E-AT without biased sampling scheme (E-AT unif.): this achieves worse results than E-AT on the WRN-28-10, showing the effectiveness of the biased

Table 7.11: **ImageNet - Fine-tuning for more epochs:** We fine-tune the l_2-robust model from Engstrom et al. [2019a] for either 1 or 3 epochs with our E-AT scheme.

model	*clean*		l_∞ ($\epsilon_\infty = \frac{4}{255}$)		l_2 ($\epsilon_2 = 2$)		l_1 ($\epsilon_1 = 255$)		union	
RN-50 - l_2	58.7		25.0		40.5		14.0		13.5	
+ 1 epochs FT	56.7	-2.0	26.7	1.7	41.0	0.5	25.4	11.4	23.1	9.6
+ 3 epochs FT	57.4	-1.3	27.8	2.8	41.6	1.1	26.7	12.7	23.7	10.2

Table 7.12: **CIFAR-10 - Fine-tuning for 1 epoch:** We show the effect of fine-tuning for a single (compared to the standard 3) models robust wrt a single norm. We use the classifiers robust wrt l_∞ and l_2 from Engstrom et al. [2019a], and the l_1-robust one from Sec. 4.4.

model	*clean*		l_∞ ($\epsilon_\infty = \frac{8}{255}$)		l_2 ($\epsilon_2 = 0.5$)		l_1 ($\epsilon_1 = 12$)		*union*	
RN-50 - l_∞	88.7		50.9		59.4		5.0		5.0	
+ 1 epochs FT	84.8	-3.9	46.6	-4.3	68.3	8.9	47.2	42.2	42.8	37.8
+ 3 epochs FT	86.2	-2.5	46.0	-4.9	70.1	10.7	49.2	44.2	43.4	38.4
RN-50 - l_2	91.5		29.7		70.3		27.0		23.0	
+ 1 epochs FT	85.9	-5.6	41.8	12.1	69.6	-0.7	47.6	20.6	39.7	16.7
+ 3 epochs FT	87.8	-3.7	43.1	13.4	70.8	0.5	50.2	23.2	41.7	18.7
RN-18 - l_1	87.1		22.0		64.8		60.3		22.0	
+ 1 epochs FT	78.9	-8.2	37.7	15.7	62.9	-1.9	51.3	-9.0	37.6	15.6
+ 3 epochs FT	83.5	-3.6	40.3	18.3	68.1	3.3	55.7	-4.6	40.1	18.1

sampling.

Experiments on MNIST. We further test the different techniques on the MNIST dataset. We use the same CNN of Maini et al. [2020] as architecture and $\epsilon_\infty = 0.3$, $\epsilon_2 = 2$ and $\epsilon_1 = 10$ as thresholds at which evaluating robustness, as done by Maini et al. [2020]. We note that while it is an easier dataset, MNIST is challenging when it comes to adversarial training since it presents unexpected phenomena: e.g. Tramèr and Boneh [2019] noted that l_∞-adversarial training induces gradient obfuscation when using attack wrt l_2 and l_1, and both Tramèr and Boneh [2019], Maini et al. [2020] had to use many PGD-steps (up to 100), and Tramèr and Boneh [2019] even a ramp-up schedule for the ϵ during training. While other modifications to the training setup might be beneficial for some or all the methods, we just increased the number of APGD-steps to 50 for l_1 (see more details below). In Table 7.15 we compare E-AT to SAT, AVG, MAX and MSD (for the last three we use the models provided by Maini et al. [2020]). First, E-AT outperforms the available classifiers trained with AVG, MAX and MSD, meaning that even on MNIST it is a strong baseline. However, in this case, SAT, which trains on all types of perturbations, achieves better results than E-AT on average: E-AT has higher variance over runs but the best run (over multiple seeds) is close to the best one of SAT in terms of robustness in the union (55.3% vs 54.4%). Interestingly, SAT has much higher robustness wrt l_2 compared to E-AT, but this is somehow expected since Eq. (3.14) would "predict" robustness for E-AT at $\epsilon_2 \approx 1.7$ while $\epsilon_2 = 2$ is used for testing, and this is precise only for linear models. Thus the slightly worse performance of E-AT compared to SAT for the chosen radii of the threat models is to be expected from our geometric analysis. Moreover, since we have shown that fine-tuning an l_p-robust model with E-AT yields high multiple norms robustness, and given that E-AT from random initialization is weak mostly wrt l_2, we fine-tune the l_2-AT classifier with E-AT. This, with just 3 or 5 epochs, significantly outperforms SAT (up to +2.3% robustness in the union), while preserving l_2-robustness. In total, we improve the previous SOTA for multiple-norm robustness for MNIST from 48.7% (MSD) to 57.5% (E-AT fine-tuning of an l_2-robust model with 5 epochs) which is a significant improvement. Note that in this case we increase the radii ϵ_∞ and ϵ_1 to 0.33 and 14 respectively to preserve the l_2-robustness: in fact, with such values Eq. (3.14) yields $\epsilon_2 \approx 2.16$. However, training from random initialization, in the standard setup, with the larger thresholds leads to worse robustness in the union. We hypothesize that this is due to the increased difficulty of the task to learn: it is known that even single norm adversarial training is problematic when increasing the value of ϵ [Ding et al., 2020].

Table 7.13: **CIFAR-10 - E-AT fine-tuning of perceptually robust models:** We use E-AT to fine-tune for 3 epochs the PAT model, robust wrt LPIPS, and compare to fine-tuning l_p-robust models.

model	*clean*		l_∞ ($\epsilon_\infty = \frac{8}{255}$)		l_2 ($\epsilon_2 = 0.5$)		l_1 ($\epsilon_1 = 12$)		*union*	
RN-50 - l_∞	88.7		50.9		59.4		5.0		5.0	
[Engstrom et al., 2019a] + FT	86.2	-2.5	46.0	-4.9	70.1	10.7	49.2	44.2	43.4	38.4
RN-50 - l_2	91.5		29.7		70.3		27.0		23.0	
[Engstrom et al., 2019a] + FT	87.8	-3.7	43.1	13.4	70.8	0.5	50.2	23.2	41.7	18.7
RN-50 - PAT	82.6		31.1		62.4		33.6		27.7	
[Laidlaw et al., 2021] + FT	83.7	1.1	43.7	12.6	68.5	6.1	50.7	17.1	42.3	14.6

For training we use 30 epochs with cyclic learning rate (maximum value 0.05, also used for fine-tuning) and no data augmentation (other settings as for CIFAR-10). As mentioned, we use in adversarial training for multiple norms (SAT, E-AT unif. and E-AT) 50 steps of APGD for l_1, and to reduce the training cost we decrease to 5 those for l_∞. Moreover, for training, in l_1-APGD we increase the parameter to control the initial sparsity of the updates to 0.1 (default is 0.05). For evaluation, we use the full AutoAttack, since on MNIST FAB and Square Attack (see Sec. 5.1) are at times stronger than PGD-based attacks, as shown in the original papers.

Table 7.14: **CIFAR-10 - Comparison of different full training schemes:** We compare methods for multiple-norm robustness on training from random initialization.

method	*clean*	l_∞ ($\epsilon_\infty = \frac{8}{255}$)	l_2 ($\epsilon_2 = 0.5$)	l_1 ($\epsilon_1 = 12$)	*average*	*union*	*time/ep.*
WideResNet-28-10							
l_∞-AT	82.6 ± 0.5	52.0 ± 0.7	59.7 ± 0.2	9.1 ± 0.2	40.3 ± 0.4	9.1 ± 0.2	922 s
l_2-AT	88.2 ± 0.4	35.9 ± 0.2	70.9 ± 0.4	36.1 ± 0.2	47.6 ± 0.2	31.3 ± 0.2	928 s
l_1-AT	83.7 ± 0.2	30.7 ± 0.7	65.1 ± 0.5	61.6 ± 0.3	52.5 ± 0.5	30.7 ± 0.7	949 s
SAT	80.5 ± 0.6	45.9 ± 0.5	66.7 ± 0.3	55.9 ± 0.5	56.2 ± 0.4	45.7 ± 0.6	925 s
MNG-AC	81.3 ± 0.3	43.5 ± 0.7	66.9 ± 0.2	57.6 ± 0.8	56.0 ± 0.4	43.3 ± 0.7	1500 s
AVG	82.5 ± 0.4	45.4 ± 1.1	68.0 ± 0.9	55.0 ± 0.2	56.1 ± 0.7	45.1 ± 1.1	2771 s
MAX	79.9 ± 0.1	48.4 ± 0.7	65.3 ± 0.3	50.2 ± 0.6	54.6 ± 0.5	47.4 ± 0.8	2479 s
MSD	80.6 ± 0.3	48.0 ± 0.2	65.6 ± 0.3	51.7 ± 0.4	55.1 ± 0.2	46.9 ± 0.1	1554 s
E-AT unif.	79.7 ± 0.2	45.4 ± 0.5	66.0 ± 0.5	55.6 ± 0.5	55.7 ± 0.4	45.1 ± 0.7	939 s
E-AT	79.9 ± 0.7	46.6 ± 0.2	66.2 ± 0.6	56.0 ± 0.4	56.3 ± 0.3	46.4 ± 0.3	921 s
PreAct ResNet-18							
l_∞-AT	84.0 ± 0.3	48.1 ± 0.2	59.7 ± 0.4	6.3 ± 1.0	38.0 ± 0.3	6.3 ± 1.0	151 s
l_2-AT	88.9 ± 0.6	27.3 ± 1.8	68.7 ± 0.1	25.3 ± 1.6	40.5 ± 1.1	20.9 ± 1.8	153 s
l_1-AT	85.9 ± 1.1	22.1 ± 0.1	64.9 ± 0.5	59.5 ± 0.8	48.8 ± 0.4	22.1 ± 0.1	195 s
SAT	83.9 ± 0.8	40.7 ± 0.7	68.0 ± 0.4	54.0 ± 1.2	54.2 ± 0.8	40.4 ± 0.7	161 s
AVG	84.6 ± 0.3	40.8 ± 0.7	68.4 ± 0.7	52.1 ± 0.4	53.8 ± 0.1	40.1 ± 0.8	479 s
MAX	80.4 ± 0.5	45.7 ± 0.9	66.0 ± 0.4	48.6 ± 0.8	53.4 ± 0.5	44.0 ± 0.7	466 s
MSD (*)	82.1	43.1	64.5	46.5	51.4	41.4	-
MSD	81.1 ± 1.1	44.9 ± 0.6	65.9 ± 0.6	49.5 ± 1.2	53.4 ± 0.4	43.9 ± 0.8	306 s
E-AT unif.	82.2 ± 1.8	42.7 ± 0.7	67.5 ± 0.5	53.6 ± 0.1	54.6 ± 0.2	42.4 ± 0.6	163 s
E-AT	81.9 ± 1.4	43.0 ± 0.9	66.4 ± 0.6	53.0 ± 0.3	54.2 ± 0.4	42.4 ± 0.7	160 s

Table 7.15: **MNIST - Comparison of full training schemes and fine-tuning with E-AT for multiple norm robustness:** We train classifier (architecture as in Maini et al. [2020]) on MNIST with different training scheme. For SAT and E-AT we report, together with the statistics over multiple random seeds, the results of the best run. Additionally, we show the results of fine-tuning the l_2-AT model with E-AT for different numbers of epochs, which achieves the best results. (*) AVG, MAX and MSD classifiers are those provided by Maini et al. [2020].

model	*clean*	l_∞ ($\epsilon_\infty = 0.3$)	l_2 ($\epsilon_2 = 2$)	l_1 ($\epsilon_1 = 10$)	*union*
l_∞-AT	98.9 ± 0.12	90.0 ± 0.45	8.4 ± 1.49	6.0 ± 0.65	4.2 ± 0.73
l_2-AT	98.8 ± 0.17	0.0 ± 0.05	70.7 ± 0.26	59.1 ± 0.37	0.0 ± 0.05
l_1-AT	98.8 ± 0.09	0.0 ± 0.00	45.8 ± 0.42	77.2 ± 0.14	0.0 ± 0.00
SAT	98.6 ± 0.17	62.0 ± 0.86	65.7 ± 1.69	61.3 ± 1.29	53.9 ± 1.25
AVG (*)	99.1	58.6	60.8	22.5	21.1
MAX (*)	98.6	39.4	59.9	25.6	20.3
MSD (*)	98.2	63.7	66.6	51.0	48.7
E-AT unif.	98.8 ± 0.12	67.1 ± 3.03	50.2 ± 4.93	62.0 ± 4.59	45.9 ± 4.43
E-AT	98.7 ± 0.12	69.0 ± 3.66	56.4 ± 3.94	61.1 ± 4.03	50.6 ± 3.89
l_2-AT + E-AT (3 ep.)	96.9 ± 0.32	57.5 ± 0.92	67.8 ± 0.58	62.0 ± 1.16	54.6 ± 0.59
l_2-AT + E-AT (5 ep.)	97.4 ± 0.21	60.6 ± 2.19	65.9 ± 0.56	63.8 ± 0.93	56.2 ± 1.16
best run					
SAT	98.8	62.7	67.2	62.6	55.3
E-AT unif.	98.9	67.4	56.3	63.6	51.6
E-AT	98.8	71.0	58.4	62.0	54.4
l_2-AT + E-AT (3 ep.)	97.3	58.0	67.9	62.9	55.7
l_2-AT + E-AT (5 ep.)	97.5	61.6	66.2	64.0	57.5

Chapter 8

Conclusion and Outlook

8.1 Conclusion

In this work we have analyzed different aspects of the adversarial robustness of image classifiers. From the attackers perspective, in Sec. 4 and Sec. 6 we have proposed several algorithms to generate perturbations which are able to fool the target model, using a variety of constraints on the type of allowed modifications (l_p-bounds, sparse attacks, perceptual bounds) and on the level of access to the classifier. This shows that existing models are vulnerable to many different attacks: a broadly robust classifier should be able to perform well in presence of any of them, since an attacker is in general not limited to any specific type of threat as long as the ground truth class is preserved.

While it is possible to train networks with provable robustness against multiple l_p-threat models (Sec. 3), such techniques are currently limited to small networks and datasets. Then, we have developed a framework to benchmark the empirical robustness of adversarial defenses (Sec. 5): this allows us to track the advancements achieved by newly proposed methods, and analyze which are the relevant elements contributing to the most effective defenses. Also, in Sec. 7 we have suggested how to adapt the type of robustness of existing models at low computational cost, which can yield baseline robust classifiers for new threat models.

Finally, while we have focused on image classification tasks (except for a brief analysis of malware detection in Sec. 6.2), adversarial perturbations might affect virtually every domain where NN-based models are employed. The techniques developed in this work could be then extended to test adversarial robustness in such cases.

8.2 Outlook

While most of the existing methods consider l_p-norms to limit the size, and then visibility, of the attacks, an active research line consists in defining new spaces of adversarial perturbations which are both aligned with human perception and allow for fast generation, e.g. Wasserstein or LPIPS-bounded attacks. Developing such methods could enrich the set of perturbations which an ideal classifier should be robust to and tested on.

Despite the significant progress of defensive techniques (see Sec. 5.2), especially via improvements of adversarial training, obtaining classifiers which are simultaneously resistant against several attacks is still a challenging and elusive goal, especially if one wants such robustness to generalize to threats not seen at training time. An even more ambitious goal is having certified robustness, which is typically more challenging than the empirical one, in these more complex scenarios. Training on multiple l_p-threat models can provide some robustness to other types of adversarial perturbations e.g. fog or elastic attacks as shown in Sec. 7.1. However, we expect more refined techniques, and possibly different architectures, to be necessary for achieving classifiers robust in a broader sense.

Moreover, it is in a open question whether adversarial robustness can naturally emerge, for example from using larger (and better) datasets or more complex systems which rely on both vision and text data, or is an inevitable phenomenon with the current deep learning paradigm.

The vulnerability of neural networks to adversarial perturbations has often been associated with security concerns: this still represents a major research direction in the field, especially with the growing deployment of such systems in the real world. However, studying the adversarial robustness of the existing models, and how to counter it, has also led to a better understanding of the classifiers and their learning dynamics. The same techniques can be used to analyze the new generation of (foundation) models, and the deriving insights might help to build more reliable systems.

Bibliography

Abdullah Al-Dujaili and Una-May O'Reilly. There are no bit parts for sign bits in black-box attacks. In *ICLR*, 2020.

Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. Adversarial deep learning for robust detection of binary encoded malware. In *IEEE Security and Privacy Workshops (SPW)*, pages 76–82, 2018.

Jean-Baptiste Alayrac, Jonathan Uesato, Po-Sen Huang, Alhussein Fawzi, Robert Stanforth, and Pushmeet Kohli. Are labels required for improving adversarial robustness? In *NeurIPS*. 2019.

Motasem Alfarra, Juan Camilo Perez, Ali Thabet, Adel Bibi, Philip Torr, and Bernard Ghanem. Combating adversaries with anti-adversaries. In *AAAI*, 2022.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature purification: How adversarial training performs robust deep learning. *arXiv preprint arXiv:2005.10190*, 2020.

Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B Srivastava. Genattack: practical black-box attacks with gradient-free optimization. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2019.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *ICML*, 2016.

Maksym Andriushchenko and Nicolas Flammarion. Understanding and improving fast adversarial training. In *NeurIPS*, 2020.

Maksym Andriushchenko and Matthias Hein. Provably robust boosted decision stumps and trees against adversarial attacks. In *NeurIPS*, 2019.

Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *ECCV*, 2020.

Alexandre Araujo, Aaron J Havens, Blaise Delattre, Alexandre Allauzen, and Bin Hu. A unified algebraic perspective on lipschitz neural networks. In *ICLR*, 2023.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear unit. In *ICLR*, 2018.

Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, 2014.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, 2018.

Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, and Yaron Lipman. Controlling neural level sets. In *NeurIPS*, 2019.

Maximilian Augustin, Alexander Meinke, and Matthias Hein. Adversarial robustness on in- and out-distribution improves explainability. In *ECCV*, 2020.

Yutong Bai, Jieru Mei, Alan Yuille, and Cihang Xie. Are transformers more robust than CNNs? In *NeurIPS*, 2021.

Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Robustness may be at odds with fairness: An empirical study on class-wise accuracy. *Proceedings of Machine Learning Research*, 2021.

Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In *ECCV*, 2018.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *ECML/PKKD*, 2013.

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

Valentyn Boreiko, Maximilian Augustin, Francesco Croce, Philipp Berens, and Matthias Hein. Sparse visual counterfactual explanations in image space. In *GCPR*, 2022.

Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. Bad characters: Imperceptible nlp attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Wieland Brendel, Jonas Rauber, Alexey Kurakin, Nicolas Papernot, Behar Veliqi, Marcel Salathé, Sharada P Mohanty, and Matthias Bethge. Adversarial vision challenge. In *NeurIPS Competition Track*, 2018.

Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge. Accurate, reliable and fast robustness evaluation. In *NeurIPS*, 2019.

Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. In *NeurIPS 2017 Workshop on Machine Learning and Computer Security*, 2017.

Tom B Brown, Nicholas Carlini, Chiyuan Zhang, Catherine Olsson, Paul Christiano, and Ian Goodfellow. Unrestricted adversarial examples. *arXiv preprint arXiv:1809.08352*, 2018.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Thomas Brunner, Frederik Diehl, Michael Truong Le, and Alois Knoll. Guessing smart: biased sampling for efficient black-box adversarial attacks. *ICCV*, 2019.

Sébastien Bubeck, Yeshwanth Cherapanamjeri, Gauthier Gidel, and Rémi Tachet des Combes. A single gradient step finds adversarial examples on random two-layers neural networks. In *NeurIPS*, 2021.

Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *ICLR*, 2018.

Nicholas Carlini. A critique of the deepsec platform for security analysis of deep learning models. *arXiv preprint arXiv:1905.07112*, 2019.

Nicholas Carlini. A complete list of all (arxiv) adversarial example papers. https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html, 2021. Accessed: 2021-06-08.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017a.

Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM Workshop on Artificial Intelligence and Security*, 2017b.

Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. version from May 14, 2019, 2019a. URL https://github.com/evaluating-adversarial-robustness/adv-eval-paper.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*, 2019b.

Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and Percy S Liang. Unlabeled data improves adversarial robustness. In *NeurIPS*. 2019.

Manon Césaire, Hatem Hajri, Sylvain Lamprier, and Patrick Gallinari. Stochastic sparse adversarial attacks. *arXiv preprint arXiv:2011.12423*, 2020.

Alvin Chan, Yi Tay, Yew Soon Ong, and Jie Fu. Jacobian adversarially regularized networks for robustness. In *ICLR*, 2020.

Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

Jiefeng Chen, Xi Wu, Yang Guo, Yingyu Liang, and Somesh Jha. Towards evaluating the robustness of neural networks learned by transduction. In *ICLR*, 2022.

Jinghui Chen and Quanquan Gu. Rays: A ray searching method for hard-label adversarial attack. In *KDD*, 2020.

Jinghui Chen, Dongruo Zhou, Jinfeng Yi, and Quanquan Gu. A frank-wolfe framework for efficient and effective adversarial attacks. In *AAAI*, 2020a.

Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. In *AAAI*, 2018.

Tianlong Chen, Sijia Liu, Shiyu Chang, Yu Cheng, Lisa Amini, and Zhangyang Wang. Adversarial robustness: From self-supervised pre-training to fine-tuning. In *CVPR*, 2020b.

Xiangyi Chen, Sijia Liu, Kaidi Xu, Xingguo Li, Xue Lin, Mingyi Hong, and David Cox. Zo-adamm: Zeroth-order adaptive momentum method for black-box optimization. In *NeurIPS*, 2019.

Zhuotong Chen, Qianxiao Li, and Zheng Zhang. Towards robust neural networks via close-loop control. In *ICLR*, 2021.

Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. In *NeurIPS*, 2019.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *CVPR*, 2014.

Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI*, 2011.

Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. In *ICML*, 2019.

Laurent Condat. Fast projection onto the simplex and the l_1 ball. *Mathematical Programming*, 158: 575–585, 2016.

Francesco Croce and Matthias Hein. A randomized gradient-free attack on relu networks. In *GCPR*, 2018.

Francesco Croce and Matthias Hein. Sparse and imperceivable adversarial attacks. In *ICCV*, 2019.

Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *ICML*, 2020a.

Francesco Croce and Matthias Hein. Provable robustness against all adversarial l_p-perturbations for $p \geq 1$. In *ICLR*, 2020b.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020c.

Francesco Croce and Matthias Hein. Mind the box: l_1-apgd for sparse adversarial attacks on image classifiers. In *ICML*, 2021.

Francesco Croce and Matthias Hein. Adversarial robustness against multiple and single l_p-threat models via quick fine-tuning of robust classifiers. In *ICML*, 2022.

Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In *AISTATS*, 2019a.

Francesco Croce, Jonas Rauber, and Matthias Hein. Scaling up the randomized gradient-free adversarial attack reveals overestimation of robustness using established attacks. *International J. of Computer Vision (IJCV)*, 2019b.

Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. In *NeurIPS Datasets and Benchmarks Track*, 2021.

Francesco Croce, Maksym Andriushchenko, Naman D Singh, Nicolas Flammarion, and Matthias Hein. Sparse-rs: a versatile framework for query-efficient sparse black-box adversarial attacks. In *AAAI*, 2022a.

Francesco Croce, Sven Gowal, Thomas Brunner, Evan Shelhamer, Matthias Hein, and Taylan Cemgil. Evaluating the adversarial robustness of adaptive test-time defenses. In *ICML*, 2022b.

Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

Damek Davis and Dmitriy Drusvyatskiy. Stochastic model-based minimization of weakly convex functions. *SIAM Journal on Optimization*, 29(1):207–239, 2019.

Edoardo Debenedetti. Adversarially robust vision transformers. Master's thesis, Swiss Federal Institute of Technology, Lausanne (EPFL), 2022.

Ambra Demontis, Paolo Russu, Battista Biggio, Giorgio Fumera, and Fabio Roli. On security and sparsity of linear classifiers for adversarial settings. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. AdverTorch v0.1: An adversarial robustness toolbox based on pytorch. *arXiv preprint arXiv:1902.07623*, 2019.

Gavin Weiguang Ding, Yash Sharma, Kry Yik Chau Lui, and Ruitong Huang. Mma training: Direct input space margin maximization through adversarial training. In *ICLR*, 2020.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *CVPR*, 2018.

Yinpeng Dong, Qi-An Fu, Xiao Yang, Tianyu Pang, Hang Su, Zihao Xiao, and Jun Zhu. Benchmarking adversarial robustness on image classification. In *CVPR*, 2020.

Jiawei Du, Hu Zhang, Joey Tianyi Zhou, Yi Yang, and Jiashi Feng. Query-efficient meta attack to deep neural networks. In *ICLR*, 2020.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *ICML*, 2008.

John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, 2015.

Alaaeldin El-Nouby, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. Xcit: Cross-covariance image transformers. *arXiv preprint arXiv:2106.09681*, 2021.

Gamaleldin Elsayed, Dilip Krishnan, Hossein Mobahi, Kevin Regan, and Samy Bengio. Large margin deep networks for classification. In *NeurIPS*, 2018.

Logan Engstrom, Andrew Ilyas, Hadi Salman, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019a. URL https://github.com/MadryLab/robustness.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. Adversarial robustness as a prior for learned representations. *arXiv preprint arXiv:1906.00945*, 2019b.

Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. In *ICML*, 2019c.

Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, 2018.

Yanbo Fan, Baoyuan Wu, Tuanhui Li, Yong Zhang, Mingyang Li, Zhifeng Li, and Yujiu Yang. Sparse adversarial attack via perturbation factorization. In *ECCV*, 2020.

Alhussein Fawzi and Pascal Frossard. Manitest: Are classifiers really invariant? In *BMVC*, 2015.

Alhussein Fawzi and Pascal Frossard. Measuring the effect of nuisance variables on classifiers. In *BMVC*, 2016.

Nicolas Ford, Justin Gilmer, Nicolas Carlini, and Dogus Cubuk. Adversarial examples are a natural consequence of test error in noise. In *ICML*, 2019.

Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136, 1975.

Justin Gilmer, Ryan P Adams, Ian Goodfellow, David Andersen, and George E Dahl. Motivating the rules of the game for adversarial example research. *arXiv preprint arXiv:1807.06732*, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Dou Goodman, Hao Xin, Wang Yang, Wu Yuesheng, Xiong Junfeng, and Zhang Huan. Advbox: a toolbox to generate adversarial examples that fool neural networks. *arXiv preprint arXiv:2001.05574*, 2020.

Sven Gowal, Krishnamurthy (Dj) Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *ICCV*, 2019a.

Sven Gowal, Jonathan Uesato, Chongli Qin, Po-Sen Huang, Timothy Mann, and Pushmeet Kohli. An alternative surrogate loss for pgd-based adversarial testing. *arXiv preprint, arXiv:1910.09338*, 2019b.

Sven Gowal, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. Uncovering the limits of adversarial training against norm-bounded adversarial examples. *arXiv preprint arXiv:2010.03593v2*, 2020.

Sven Gowal, Sylvestre-Alvise Rebuffi, Olivia Wiles, Florian Stimberg, Dan Andrei Calian, and Timothy Mann. Improving robustness using generated data. In *NeurIPS*, 2021.

Will Grathwohl, Kuan-Chieh Wang, Joern-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *ICLR*, 2020.

Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. In *ICLR*, 2018.

Chuan Guo, Jacob Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Weinberger. Simple black-box adversarial attacks. In *ICML*, 2019.

Uffe Haagerup. The best constants in the Khintchine inequality. *Studia Math.*, 70(3):231–283, 1981.

Philip Hall. The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable. *Biometrika*, 19:240—245, 1927.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016b.

Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NeurIPS*, 2017.

Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *CVPR*, 2019.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*, 2017.

Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *ICML*, 2019.

Dan Hendrycks, Norman Mu, Ekin D. Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *ICLR*, 2020.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *ACL*, 2018.

J Edward Hu, Adith Swaminathan, Hadi Salman, and Greg Yang. Improved image wasserstein attacks and defenses. *ICLR Workshop: Towards Trustworthy ML: Rethinking Security and Privacy for ML*, 2020.

Kai Hu, Andy Zou, Zifan Wang, Klas Leino, and Matt Fredrikson. Scaling in depth: Unlocking robustness certification on imagenet. *arXiv preprint arXiv:2301.12549*, 2023.

Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on GAN. *arXiv preprint arXiv:1702.05983*, 2017.

Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. In *ICLR*, 2016.

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

Zhichao Huang and Tong Zhang. Black-box adversarial attack with transferable model-based embedding. In *ICLR*, 2020.

Duhun Hwang, Eunjung Lee, and Wonjong Rhee. Aid-purifier: A light auxiliary network for boosting adversarial defense. In *ICML Workshop on Adversarial Machine Learning*, 2021.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *ICML*, 2018.

Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In *ICLR*, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Oscar Irwin. On the frequency distribution of the means of samples from a population having any law of frequency with finite moments. *Biometrika*, 19:225—239, 1927.

Yunseok Jang, Tianchen Zhao, Seunghoon Hong, and Honglak Lee. Adversarial defense via learning to generate diverse attacks. In *ICCV*, 2019.

Ahmadreza Jeddi, Mohammad Javad Shafiee, and Alexander Wong. A simple fine-tuning is all you need: Towards robust deep learning via adversarial fine-tuning. *arXiv preprint, arXiv:2012.13628*, 2020.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is BERT really robust? natural language attack on text classification and entailment. In *AAAI*, 2019.

Daniel Kang, Yi Sun, Tom Brown, Dan Hendrycks, and Jacob Steinhardt. Transfer of adversarial robustness between perturbation types. *arXiv preprint, arXiv:1905.01034*, 2019a.

Daniel Kang, Yi Sun, Dan Hendrycks, Tom Brown, and Jacob Steinhardt. Testing robustness against unforeseen adversaries. *arXiv preprint arXiv:1908.08016*, 2019b.

Qiyu Kang, Yang Song, Qinxu Ding, and Wee Peng Tay. Stable neural ode with lyapunov-stable equilibrium points for defending against adversarial attacks. In *NeurIPS*, 2021.

Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.

Oğuzhan Fatih Kar, Teresa Yeo, Andrei Atanov, and Amir Zamir. 3d common corruptions and data augmentation. In *CVPR*, 2022.

Danny Karmon, Daniel Zoran, and Yoav Goldberg. Lavan: Localized and visible adversarial noise. In *ICML*, 2018.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV*, 2017.

Simran Kaur, Jeremy Cohen, and Zachary C Lipton. Are perceptually-aligned gradients a general property of robust classifiers? In *NeurIPS Workshop: Science Meets Engineering of Deep Learning*, 2019.

Jungeum Kim and Xiao Wang. Sensible adversarial learning, 2020. URL https://openreview.net/forum?id=rJlf_RVKwr.

Minseon Kim, Jihoon Tack, and Sung Ju Hwang. Adversarial self-supervised contrastive learning. In *NeurIPS*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *preprint, arXiv:1412.6980*, 2014.

Klim Kireev, Maksym Andriushchenko, and Nicolas Flammarion. On the effectiveness of adversarial training against common corruptions. *arXiv preprint arXiv:2103.02325*, 2021.

Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *European signal processing conference (EUSIPCO)*, 2018.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2012.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Nupur Kumari, Mayank Singh, Abhishek Sinha, Harshitha Machiraju, Balaji Krishnamurthy, and Vineeth N Balasubramanian. Harnessing the vulnerability of latent layers in adversarially trained models. In *IJCAI*, 2019.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *ICLR Workshop*, 2017a.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *ICLR*, 2017b.

Alexey Kurakin, Ian Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, et al. Adversarial attacks and defences competition. In *NeurIPS Competition Track*, 2018.

Cassidy Laidlaw and Soheil Feizi. Functional adversarial attacks. In *NeurIPS*, 2019.

Cassidy Laidlaw, Sahil Singla, and Soheil Feizi. Perceptual adversarial robustness: Defense against unseen threat models. In *ICLR*, 2021.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE S&P*, 2019.

Mark Lee and Zico Kolter. On physical adversarial patches for object detection. *ICML Workshop on Security and Privacy of Machine Learning*, 2019.

Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In *ICML*, 2021.

Alexander J Levine and Soheil Feizi. Improved, deterministic smoothing for l_1 certified robustness. In *ICML*, 2021.

Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. In *NeurIPS*, 2019a.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271*, 2018.

Juncheng Li, Frank Schmidt, and Zico Kolter. Adversarial camera stickers: A physical camera-based attack on deep learning systems. In *ICML*, 2019b.

Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. Deeprobust: A pytorch library for adversarial attacks and defenses. *arXiv preprint arXiv:2005.06149*, 2020.

Yingzhen Li, John Bradshaw, and Yash Sharma. Are generative classifiers more robust to adversarial attacks? In *ICML*, 2019c.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *ICLR*, 2014.

Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. In *IJCAI*, 2017.

Xiang Ling, Shouling Ji, Jiaxu Zou, Jiannan Wang, Chunming Wu, Bo Li, and Ting Wang. Deepsec: A uniform platform for security analysis of deep learning model. In *IEEE S&P*, 2019.

Aishan Liu, Shiyu Tang, Xianglong Liu, Xinyun Chen, Lei Huang, Zhuozhuo Tu, Dawn Song, and Dacheng Tao. Towards defending multiple adversarial perturbations via gated batch normalization. *arXiv preprint arXiv:2012.01654v1*, 2020.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

Xiaolei Liu, Xiaojiang Du, Xiaosong Zhang, Qingxin Zhu, Hao Wang, and Mohsen Guizani. Adversarial samples on android malware detection systems for IoT systems. *Sensors*, 19(4):974, 2019.

Daniel Lowd and Christopher Meek. Adversarial learning. In *KDD*, 2005.

Bo Luo, Yannan Liu, Lingxiao Wei, and Qiang Xu. Towards imperceptible and robust adversarial example attacks against neural networks. In *AAAI*, 2018.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

Divyam Madaan, Jinwoo Shin, and Sung Ju Hwang. Learning to generate noise for multi-attack robustness. In *ICML*, 2021.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

Pratyush Maini, Eric Wong, and Zico Kolter. Adversarial robustness against the union of multiple perturbation models. In *ICML*, 2020.

Chengzhi Mao, Ziyuan Zhong, Junfeng Yang, Carl Vondrick, and Baishakhi Ray. Metric learning for adversarial robustness. In *NeurIPS*. 2019.

Chengzhi Mao, Mia Chiquier, Hao Wang, Junfeng Yang, and Carl Vondrick. Adversarial attacks are reversible with natural supervision. In *ICCV*, 2021.

J. Matyas. Random optimization. *Automation and Remote control*, 26(2):246–253, 1965.

Alexander Matyasko and Lap-Pui Chau. Pdpgd: Primal-dual proximal gradient descent adversarial attack. *arXiv preprint arXiv:2106.01538*, 2021.

Marco Melis, Ambra Demontis, Maura Pintor, Angelo Sotgiu, and Battista Biggio. secml: A python library for secure and explainable machine learning. *arXiv preprint arXiv:1912.10013*, 2019.

Laurent Meunier, Jamal Atif, and Olivier Teytaud. Yet another but more efficient black-box adversarial attack: tiling and evolution strategies. *arXiv preprint, arXiv:1910.02244*, 2019.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *ICML*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Apostolos Modas, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Sparsefool: a few pixels make a big difference. In *CVPR*, 2019.

Seungyong Moon, Gaon An, and Hyun Oh Song. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. In *ICML*, 2019.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2016.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *CVPR*, 2017.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. Robustness via curvature regularization, and vice versa. In *CVPR*, 2019.

Marius Mosbach, Maksym Andriushchenko, Thomas Trost, Matthias Hein, and Dietrich Klakow. Logit pairing methods can fool gradient-based attacks. In *NeurIPS 2018 Workshop on Security in Machine Learning*, 2018.

Mark Niklas Mueller, Franziska Eckert, Marc Fischer, and Martin Vechev. Certified training: Small boxes are all you need. In *ICLR*, 2023.

Aamir Mustafa, Salman Khan, Munawar Hayat, Roland Goecke, Jianbing Shen, and Ling Shao. Adversarial defense by restricting the hidden space of deep neural networks. In *ICCV*, 2019.

Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *CVPR Workshops*, 2017.

A. S. Nemirovsky and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons, 1983.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *Technical Report*, 2011.

Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.2.0. *arXiv preprint arXiv:1807.01069*, 2018.

Weili Nie, Brandon Guo, Yujia Huang, Chaowei Xiao, Arash Vahdat, and Anima Anandkumar. Diffusion models for adversarial purification. In *ICML*, 2022.

Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In *ICML*, 2019.

Tianyu Pang, Kun Xu, Yinpeng Dong, Chao Du, Ning Chen, and Jun Zhu. Rethinking softmax cross-entropy loss for adversarial robustness. In *ICLR*, 2020a.

Tianyu Pang, Kun Xu, and Jun Zhu. Mixup inference: Better exploiting mixup to defend adversarial attacks. In *ICLR*, 2020b.

Tianyu Pang, Xiao Yang, Yinpeng Dong, Kun Xu, Hang Su, and Jun Zhu. Boosting adversarial training with hypersphere embedding. In *NeurIPS*, 2020c.

Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016a.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016b.

Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Maura Pintor, Fabio Roli, Wieland Brendel, and Battista Biggio. Fast minimum-norm adversarial attacks through adaptive norm constraints. In *NeurIPS*, 2021.

Robert Podschwadt and Hassan Takabi. On effectiveness of adversarial examples and defenses for malware classification. In *International Conference on Security and Privacy in Communication Systems*, 2019.

Aram-Alexandre Pooladian, Chris Finlay, Tim Hoheisel, and Adam M. Oberman. A principled approach for generating adversarial images under non-smooth dissimilarity metrics. In *AISTATS*, 2020.

Zhuang Qian, Shufei Zhang, Kaizhu Huang, Qiufeng Wang, Rui Zhang, and Xinping Yi. Improving model robustness with latent distribution locally and globally. *arXiv preprint arXiv:2107.04401*, 2021.

Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial robustness through local linearization. In *NeurIPS*, 2019.

Rahul Rade and Seyed-Mohsen Moosavi-Dezfooli. Helper-based adversarial training: Reducing excessive margin to achieve a better accuracy vs. robustness trade-off. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *ICLR*, 2018.

Sukrut Rao, David Stutz, and Bernt Schiele. Adversarial training against location-optimized adversarial patches. In *ECCV Workshop on the Dark and Bright Sides of Computer Vision: Challenges and Oppurtinities for Privacy and Security*, 2020.

L. Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automaton & Remote Control*, 24:1337–1342, 1963.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *ICML Reliable Machine Learning in the Wild Workshop*, 2017.

Sylvestre-Alvise Rebuffi, Sven Gowal, Dan A. Calian, Florian Stimberg, Olivia Wiles, and Timothy Mann. Data augmentation can improve robustness. In *NeurIPS*, 2021a.

Sylvestre-Alvise Rebuffi, Sven Gowal, Dan A. Calian, Florian Stimberg, Olivia Wiles, and Timothy Mann. Fixing data augmentation to improve adversarial robustness. *arXiv preprint arXiv:2103.01946*, 2021b.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

Leslie Rice, Eric Wong, and J. Zico Kolter. Overfitting in adversarially robust deep learning. In *ICML*, 2020.

Jérôme Rony and Ismail Ben Ayed. Adversarial library, 2020. URL https://github.com/jeromerony/adversarial-library.

Jérôme Rony, Luiz G Hafemann, Luiz S Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *CVPR*, 2019.

Jérôme Rony, Eric Granger, Marco Pedersoli, and Ismail Ben Ayed. Augmented Lagrangian adversarial attacks. In *ICCV*, 2021.

Parsa Saadatpanah, Ali Shafahi, and Tom Goldstein. Adversarial attacks on copyright detection systems. In *ICML*, 2020.

Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust imagenet models transfer better? In *NeurIPS*, 2020.

Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *ICLR*, 2018.

Shibani Santurkar, Dimitris Tsipras, Brandon Tran, Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Image synthesis with a single (robust) classifier. In *NeurIPS*, 2019.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on MNIST. In *ICLR*, 2019.

Günther Schrack and Mark Choit. Optimized relative step size random searches. *Mathematical Programming*, 10:230–244, 1976.

Micheal A Schumer and Kenneth Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13(3):270–276, 1968.

Vikash Sehwag, Arjun Nitin Bhagoji, Liwei Song, Chawin Sitawarin, Daniel Cullina, Mung Chiang, and Prateek Mittal. Analyzing the robustness of open-world machine learning. In *12th ACM Workshop on Artificial Intelligence and Security*, 2019.

Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. On pruning adversarially robust neural networks. In *NeurIPS*, 2020.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *NeurIPS*. 2019.

Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. A general framework for adversarial examples with objectives. *ACM Transactions on Privacy and Security (TOPS)*, 2019.

Changhao Shi, Chester Holtz, and Gal Mishne. Online adversarial purification based on self-supervised learning. In *ICLR*, 2020.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

Satya Narayan Shukla, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Black-box adversarial attacks with Bayesian optimization. *arXiv preprint arXiv:1909.13857*, 2019.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Chawin Sitawarin and David Wagner. Minimum-norm adversarial examples on knn and knn based models. In *IEEE Security and Privacy Workshops (SPW)*, 2020.

Chuanbiao Song, Kun He, Liwei Wang, and John E. Hopcroft. Improving the generalization of adversarial training with domain adaptation. In *ICLR*, 2019a.

Liwei Song and Prateek Mittal. Systematic evaluation of privacy risks of machine learning models. In *USENIX Security Symposium*, 2021.

Liwei Song, Reza Shokri, and Prateek Mittal. Privacy risks of securing machine learning models against adversarial examples. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 241–257, 2019b.

Liwei Song, Vikash Sehwag, Arjun Nitin Bhagoji, and Prateek Mittal. A critical evaluation of open-world machine learning. *arXiv preprint arXiv:2007.04391*, 2020.

Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. In *NeurIPS*, 2020.

Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *ICLR*, 2018.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.

Jack W Stokes, De Wang, Mady Marinescu, Marc Marino, and Brian Bussone. Attack and defense of dynamic analysis-based, adversarial neural malware classification models. *arXiv preprint arXiv:1712.05919*, 2017.

David Stutz, Matthias Hein, and Bernt Schiele. Confidence-calibrated adversarial training: Generalizing to unseen attacks. In *ICML*, 2020.

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23:828–841, 2019.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *ICML*, 2020.

Fnu Suya, Jianfeng Chi, David Evans, and Yuan Tian. Hybrid batch attacks: Finding black-box adversarial examples with limited queries. *arXiv preprint arXiv:1908.07000*, 2019.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Saeid Asgari Taghanaki, Kumar Abhishek, Shekoofeh Azizi, and Ghassan Hamarneh. A kernelized manifold mapping to diminish the effect of adversarial perturbations. In *CVPR*, 2019.

Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. *arXiv preprint arXiv:2007.00644*, 2020.

Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: adversarial patches to attack person detection. In *CVPR Workshops*, 2019.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *ICLR*, 2019.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

Florian Tramèr, Pascal Dupré, Gili Rusak, Giancarlo Pellegrino, and Dan Boneh. Adversarial: Perceptual ad blocking meets adversarial machine learning. In *ACM SIGSAC CCS*, 2019.

Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *NeurIPS*, 2019.

Florian Tramèr, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. In *NeurIPS*, 2020.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *ICLR*, 2019.

Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *AAAI*, 2019.

Jonathan Uesato, Brendan O'donoghue, Pushmeet Kohli, and Aaron Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *ICML*, 2018.

Francisco Utrera, Evan Kravitz, N Benjamin Erichson, Rajiv Khanna, and Michael W Mahoney. Adversarially-trained deep nets transfer better. *arXiv preprint arXiv:2007.05869*, 2020.

Václav Voráček and Matthias Hein. Provably adversarially robust nearest prototype classifiers. In *ICML*, 2022.

Bao Wang, Zuoqiang Shi, and Stanley Osher. Resnets ensemble via the feynman-kac formalism to improve natural and robust accuracies. In *NeurIPS*, 2019a.

Dequan Wang, An Ju, Evan Shelhamer, David Wagner, and Trevor Darrell. Fighting gradients with gradients: Dynamic defenses against adversarial attacks. *arXiv preprint arXiv:2105.08714*, 2021a.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *ICLR*, 2021b.

Jianyu Wang and Haichao Zhang. Bilateral adversarial training: Towards fast training of more robust models against adversarial attacks. In *ICCV*, 2019.

Jingkang Wang, Tianyun Zhang, Sijia Liu, Pin-Yu Chen, Jiacen Xu, Makan Fardad, and Bo Li. Towards a unified min-max framework for adversarial exploration and robustness. *arXiv preprint arXiv:1906.03563*, 2019b.

Yisen Wang, Difan Zou, Jinfeng Yi, James Bailey, Xingjun Ma, and Quanquan Gu. Improving adversarial robustness requires revisiting misclassified examples. In *ICLR*, 2020.

Zekai Wang, Tianyu Pang, Chao Du, Min Lin, Weiwei Liu, and Shuicheng Yan. Better diffusion models further improve adversarial training. In *ICML*.

Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *ICML*, 2018.

Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018.

Eric Wong and J Zico Kolter. Learning perturbation sets for robust machine learning. In *ICLR*, 2021.

Eric Wong, Frank R Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *NeurIPS*, 2018.

Eric Wong, Frank R Schmidt, and J Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. In *ICML*, 2019.

Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *ICLR*, 2020.

Boxi Wu, Jinghui Chen, Deng Cai, Xiaofei He, and Quanquan Gu. Do wider neural networks really help adversarial robustness? *arXiv preprint arXiv:2010.01279v2*, 2021a.

Boxi Wu, Heng Pan, Li Shen, Jindong Gu, Shuai Zhao, Zhifeng Li, Deng Cai, Xiaofei He, and Wei Liu. Attacking adversarial attacks as a defense. *arXiv preprint arXiv:2106.04938*, 2021b.

Dongxian Wu, Shu tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In *NeurIPS*, 2020a.

Yi-Hsuan Wu, Chia-Hung Yuan, and Shan-Hung Wu. Adversarial robustness via runtime masking and cleansing. In *ICML*, 2020b.

Chang Xiao, Peilin Zhong, and Changxi Zheng. Enhancing adversarial defense by k-winners-take-all. In *ICLR*, 2020.

Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. In *ICLR*, 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *preprint, arXiv:1708.07747*, 2017.

Kai Y Xiao, Vincent Tjeng, Nur Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. In *ICLR*, 2019.

Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *CVPR*, 2020.

Cong Xu and Min Yang. Adversarial momentum-contrastive pre-training. *arXiv preprint, arXiv:2012.13154v2*, 2020.

Han Xu, Xiaorui Liu, Yaxin Li, and Jiliang Tang. To be robust or to be fair: Towards fairness in adversarial training. *arXiv preprint arXiv:2010.06121*, 2020a.

Kaidi Xu, Sijia Liu, Pu Zhao, Pin-Yu Chen, Huan Zhang, Quanfu Fan, Deniz Erdogmus, Yanzhi Wang, and Xue Lin. Structured adversarial attack: Towards general implementation and better interpretability. In *ICLR*, 2018.

Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world. In *ECCV*, 2020b.

Ziang Yan, Yiwen Guo, and Changshui Zhang. Subspace attack: Exploiting promising subspaces for query-efficient black-box attacks. In *NeurIPS*, 2019.

Chenglin Yang, Adam Kortylewski, Cihang Xie, Yinzhi Cao, and Alan Yuille. Patchattack: A black-box texture-based attack with reinforcement learning. In *ECCV*, 2020a.

Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Ruslan Salakhutdinov, and Kamalika Chaudhuri. A closer look at accuracy vs. robustness. In *NeurIPS*, 2020b.

Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. ME-net: Towards effective adversarial robustness with matrix estimation. In *ICML*, 2019.

Dong Yin, Raphael Gontijo Lopes, Jon Shlens, Ekin Dogus Cubuk, and Justin Gilmer. A Fourier perspective on model robustness in computer vision. In *NeurIPS*, 2019.

Jongmin Yoon, Sung Ju Hwang, and Juho Lee. Adversarial purification with score-based generative models. In *ICML*, 2021.

Zelda B Zabinsky. Random search algorithms. *Wiley encyclopedia of operations research and management science*, 2010.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

Michał Zajac, Konrad Zołna, Negar Rostamzadeh, and Pedro O Pinheiro. Adversarial framing for image and video classification. In *AAAI*, 2019.

Bohang Zhang, Du Jiang, Di He, and Liwei Wang. Boosting the certified robustness of l-infinity distance nets. In *ICLR*, 2022a.

Bohang Zhang, Du Jiang, Di He, and Liwei Wang. Rethinking lipschitz neural networks and certified robustness: A boolean function perspective. In *NeurIPS*, 2022b.

Chong Zhang, Huan Zhang, and Cho-Jui Hsieh. An efficient adversarial attack for tree ensembles. In *NeurIPS*, 2020a.

Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In *NeurIPS*. 2019a.

Haichao Zhang and Jianyu Wang. Defense against adversarial attacks using feature scattering-based adversarial training. In *NeurIPS*. 2019.

Haichao Zhang and Wei Xu. Adversarial interpolation training: A simple approach for improving model robustness, 2020. URL https://openreview.net/forum?id=Syejj0NYvr.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *ICML*, 2019b.

Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *ICLR*, 2020b.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

Pu Zhao, Sijia Liu, Pin-Yu Chen, Nghia Hoang, Kaidi Xu, Bhavya Kailkhura, and Xue Lin. On the design of black-box adversarial examples by leveraging gradient-free optimization and operator splitting method. In *ICCV*, 2019.

Tianhang Zheng, Changyou Chen, and Kui Ren. Distributionally adversarial attack. In *AAAI*, 2019.

Jiachen Zhong, Xuanqing Liu, and Cho-Jui Hsieh. Improving the speed and quality of gan by adversarial training. *arXiv preprint arXiv:2008.03364*, 2020.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding. In *ICLR*, 2019.