

Robust Image Classification with 1-Lipschitz Networks

by

Bernd Prach

March, 2025

*A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy*

Committee in charge:
Michael Sammler, Chair
Christoph H. Lampert
Marco Mondelli
Andrea Vedaldi



The thesis of Bernd Prach, titled *Robust Image Classification with 1-Lipschitz Networks*, is approved by:

Supervisor: Prof. Christoph H. Lampert, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Prof. Marco Mondelli, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Prof. Andrea Vedaldi, University of Oxford, Oxford, UK

Signature: _____

Defense Chair: Prof. Michael Sammler, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

© by Bernd Prach, March, 2025
All Rights Reserved

ISTA Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I accept full responsibility for the content and factual accuracy of this work, including the data and their analysis and presentation, and the text and citation of other work.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Bernd Prach
March, 2025

Abstract

Despite generating remarkable results in various computer vision tasks, deep learning comes with some surprising shortcomings. For example, tiny perturbations, often imperceptible to the human eye, can completely change the predictions of image classifiers. Despite a decade of research, the field has made limited progress in developing image classifiers that are both accurate and robust. This thesis aims to address this gap.

As our first contribution, we aim to simplify the process of training certifiably robust image classifiers. We do this by designing a convolutional layer that does not require executing an iterative procedure in every forward pass, but relies on an explicit bound instead. We also propose a loss function that allows optimizing for a particular margin more precisely.

Next, we provide an overview and comparison of various methods that create robust image classifiers by constraining the Lipschitz constant. This is important since generally longer training times and more parameters improve the performance of robust classifiers, making it challenging to determine the most practical and effective methods from existing literature.

In 1-Lipschitz classification, the performance of current methods is still much worse than what we expect on the simple tasks we consider. Therefore, we next investigate potential causes of this shortcoming. We first consider the role of the activation function. We prove a theoretical shortcoming of the commonly used activation function, and provide an alternative without it. However this theoretical improvement does barely translate to the empirical performance of robust classifiers, suggesting a different bottleneck.

Therefore, in the final chapter, we study how the performance depends on the amount of training data. We prove that in the worst case, we might require far more data to train a robust classifier compared to a normal one. We furthermore find that the amount of training data is a key determinant of the performance current methods achieve on popular datasets. Additionally, we show that linear subspaces exist with tiny data variance, and yet we can still train very accurate classifiers after projecting into those subspaces. This shows that on the datasets considered, enforcing robustness in classification makes the task strictly more challenging.

Acknowledgements

First and foremost, I want to thank my supervisor Christoph, for his support, his patience and his continuous efforts to make me keep the bigger picture in mind. Furthermore I want to thank him for the time and dedication he invested in teaching me many valuable skills, in particular academic writing. I also want to thank Marco and Andrea for being part of my thesis committee and for providing useful feedback and advice throughout my PhD.

I really enjoyed all the interesting discussions and the opportunity to learn from other members of our group. Many thanks to Amélie, Mary, Niko, Alex, Paul, Jonny, Hossein, Peter, Egor, Nikita, Max and Edwige as well as our interns. Special thanks also to my co-author Fabio for the enjoyable collaboration.

Finally, I would like to thank my family, friends and especially my wife Ishita for the endless support and encouragement, and for giving meaning to this journey.

About the Author

Bernd Prach completed both a bachelor's degree and a master's degree in mathematics at the University of Cambridge. After his studies, he joined the group of Christoph Lampert at IST Austria as an intern. Following a great experience, he decided to pursue his PhD with the group. During his research, he initially focused on interpretable computer vision models, but soon switched his focus to robust classification. He worked on 1-Lipschitz models, and aimed to develop better methods and a deeper understanding of the challenges of robust classification.

List of Collaborators and Publications

This thesis is written based on the following first-author publications and preprints of Bernd Prach. For each, we provide a summary of the contribution of each author, referred to by initials.

1. [PL22]. Bernd Prach and Christoph H. Lampert. Almost-orthogonal layers for efficient general-purpose Lipschitz networks. In *European Conference on Computer Vision (ECCV)*, 2022. Reproduced with permission from Springer Nature
 - BP had the idea and designed and proved the bounds.
 - BP implemented the layers and conducted the experiments.
 - CHL supervised the project.
 - CHL did the main part of the writing, BP wrote the proofs.

This paper was reused in full in Chapter 3.

2. [PBBL24]. © 2024 IEEE. Reprinted, with permission, from Bernd Prach, Fabio Brau, Giorgio Buttazzo, and Christoph H. Lampert. 1-Lipschitz layers compared: Memory speed and certifiable robustness. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024
 - CHL suggested the work.
 - FB and BP developed the approach together.
 - BP led the theoretical part and the memory and training time experiments.
 - FB led the implementation of methods, and the experiments for robust classification.
 - CHL advised the project.
 - BP and FB wrote most of the paper as joint first authors, CHL and GB edited the manuscript.

This paper was reused in full in Chapter 4.

3. [PL23]. Bernd Prach and Christoph H. Lampert. 1-Lipschitz neural networks are more expressive with N-activations. *arXiv preprint arXiv:2311.06103*, 2023
 - BP had the idea and developed the theorems and proofs.
 - CHL advised throughout the project.
 - BP wrote the paper and CHL edited it.

This paper was reused in full in Chapter 5.

4. [PL25]. © 2025 IEEE. Reprinted, with permission, from Bernd Prach and Christoph H. Lampert. Intriguing properties of robust classification. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2025

- BP had the idea and designed the experiments.
- CHL advised throughout the project.
- BP wrote the paper and CHL edited it.

This paper was reused in full in Chapter 6.

Table of Contents

Abstract	vii
Acknowledgements	viii
About the Author	ix
List of Collaborators and Publications	x
Table of Contents	xiii
List of Figures	xiv
List of Tables	xv
List of Abbreviations	xv
1 Introduction	1
2 Background	3
2.1 Supervised Learning	3
2.2 Robust Classification	3
2.3 1-Lipschitz Networks	6
3 Almost Orthogonal Lipschitz Layers	9
3.1 Motivation	9
3.2 Related Work	9
3.3 Almost-Orthogonal Lipschitz (AOL) Layers	10
3.4 Offset Cross-Entropy Loss Function	14
3.5 Experimental Setup	14
3.6 Results	16
3.7 Discussion	18
4 1-Lipschitz Layers Compared	19
4.1 Background and Existing Work	19
4.2 Theoretical Comparison	22
4.3 Experimental Setup	26
4.4 Experimental Results	28
4.5 Potential Future Directions	33
4.6 Summary and Guidelines	34

5	1-Lipschitz Networks are more expressive with \mathcal{N}-activations	37
5.1	Background & Definitions	38
5.2	Related Work	38
5.3	Theoretical Results	39
5.4	Experimental Setup	46
5.5	Empirical Results	47
5.6	Discussion	49
6	Intriguing Properties of Robust Classifiers	51
6.1	Robust Classification Needs More Data	52
6.2	Experimental Setup	56
6.3	Experimental Results	57
6.4	Related Work	65
6.5	Summary and Discussion	66
7	Conclusion and future work	69
7.1	Thesis Summary	69
7.2	Future Directions	70
7.3	Larger Context	71
	Bibliography	73
A	Full Results for 1-Lipschitz Layer Comparison	79

List of Figures

3.1	OffsetCrossEntropy with different temperature parameters	15
3.2	AOL has an almost orthogonal Jacobian.	17
4.1	Comparison of training time and inference throughput	29
4.2	Comparison of memory requirements	30
4.3	Comparison of CRA	31
4.4	Performance summary	35
5.1	\mathcal{N} -function & \mathcal{N} -activation	39
5.2	Fitting the \mathcal{N} -function	48
5.3	Comparing performance of MaxMin and \mathcal{N} -activation	49
5.4	Comparing \mathcal{N} -activation initializations	50
6.1	Scaling behavior on CIFAR-10.	58
6.2	Scaling behavior on MNIST and CIFAR-100.	59
6.3	Scaling compute.	60
6.4	Scaling behavior on generated data.	61
6.5	Performance on different subsets of the principal components.	62
6.6	We can robustly overfit the CIFAR-10 training set.	63

6.7	1-Lipschitz model can generalize.	64
A.1	Comparison of model accuracy in decreasing order.	80

List of Tables

3.1	Patchwise architecture	15
3.2	Certified Robust Classification on CIFAR-10.	16
3.3	Certified Robust Classification on CIFAR-100.	16
3.4	Accuracy-robustness trade-off with OffsetCrossEntropy	18
3.5	Generality of AOL.	18
4.1	Computational complexity and memory requirements	23
4.2	Architecture	26
4.3	ConvBlock	27
4.4	Model size guide	27
4.5	Epoch Budgets	30
4.6	Comparison of accuracy and CRA.	32
5.1	Comparing performance with MaxMin and \mathcal{N} -activation	49
5.2	Comparing \mathcal{N} -activation initializations	50
6.1	Performance on different subsets of principal components	62
A.1	Full comparison result on CIFAR-10.	81
A.2	Full comparison result on CIFAR-100.	82
A.3	Full comparison result on Tiny ImageNet.	83
A.4	Full comparison result on Imagenette.	84

List of Abbreviations

- 1-LPL** 1-Lipschitz, piece-wise linear. 38–45
- AOL** Almost Orthogonal Lipschitz. 9–12, 14–18, 21–25, 29, 31, 33, 34, 46, 56, 63
- BCOP** Block Orthogonal Convolution Parameterization. 10, 21, 23–25, 29–31, 33, 34
- Cayley** Cayley Layers. 21, 23–25, 29–33
- CPL** Convex Potential Layer. 21–25, 29, 31, 33, 46, 48, 56, 61, 63

CRA Certified Robust Accuracy. 16, 18, 49, 64

i.i.d. independent and identically distributed. 3

LOT Layer-wise Orthogonal Training. 21–26, 29–31, 33

MACs multiply-accumulate operations. 23, 24, 29

MLP multilayer perceptron. 17, 18, 56, 58, 63

SGD stochastic gradient descent. 15, 28, 46, 57

SLL SDP-based Lipschitz Layers. 21–25, 29, 31, 33

SOC Skew Orthogonal Convolution. 10, 21, 23, 25, 29, 31, 33, 34, 46, 47

Introduction

In the last ten years we have achieved unprecedented results in various computer vision tasks by using *deep learning*. However, despite their impressive performance, deep neural networks still have some surprising shortcomings. For example, there are tiny perturbations, usually imperceptible to the human eye, that can completely change the predictions of state-of-the-art models. This phenomenon is known as *Adversarial Examples* [SZS⁺14].

We believe that adversarial examples are an important issue to address for multiple reasons. Firstly, adversarial examples can negatively affect the trust of user into a system. Secondly, it could pose a potential safety risk for systems that incorporate deep networks. Finally, the existence of adversarial examples makes it impossible to develop faithful interpretable models, as they appear as identical images with very different predictions to humans.

In order to obtain models that are less affected by such adversarial examples, researchers developed multiple solutions, including *adversarial training* [SZS⁺14, GSS15], *randomized smoothing* [CRK19] and *Lipschitz networks* [CBG⁺17]. In adversarial training, adversarial examples are constructed during training, and the model is optimized to classify those correctly. This procedure makes it harder to find adversarial examples, however, it cannot *guarantee* that none exist. As a result we can never be certain about a model's performance, as a stronger attack could in the future find adversarial perturbations of images we considered robustly classified.

In order to obtain guarantees of robustness, we can use randomized smoothing. This procedure can provide a probabilistic guarantee that no adversarial example exists for a particular input. In randomized smoothing, noise is added to an input before passing it through the network in order to limit the impact of adversarial perturbations. This step is repeated, often thousands of times, and the image is classified based on the majority vote among all predictions. While this procedure offers robustness guarantees, it is impractical as it requires a huge amount of forward passes for every example.

Therefore, in settings where the inference time is an important property, we believe that the most promising approach is to use Lipschitz networks. These networks restrict the *Lipschitz constant* of the network, typically to be at most 1. Networks with a Lipschitz constant of 1 have the property that any two model outputs are at most as far apart as the original two inputs. Consequently, tiny perturbations to an input can only cause tiny changes in the model's predictions, and we can use this property to provide guarantees of robustness.

In this thesis we aim to explore robust classification using 1-Lipschitz models. In the first chapter we introduce new components (a linear layer and a loss function) that simplify training 1-Lipschitz models and improve their robust performance. In the following chapter we compare various existing methods of creating 1-Lipschitz convolutions, with a focus on how many resources (memory and training time) the methods require. In the third chapter we investigate why current 1-Lipschitz models still lack performance by analyzing the role of activation function. In the final chapter we continue to examine the performance limitations of robust classifiers, and present theoretical arguments and experimental evidence showing that the amount of training data plays a crucial role in determining the performance of robust classifiers.

Background

This chapter is based on our publications [PL22, PL25].

2.1 Supervised Learning

In this thesis we exclusively consider supervised learning. In supervised learning, we usually have some input \mathbf{x} , and we are interested in predicting a label y for it. We will assume $\mathbf{x} \in \mathbb{R}^d$ for some data dimension d , and $y \in [c]$, where c is the number of classes and $[c] := \{1, \dots, c\}$. We write \mathcal{D} for the joint distribution of inputs and labels, and \mathcal{D}_x for the marginal distribution of the inputs. In order to learn how to predict a label y based on an input \mathbf{x} we have access to some training data, $D_{\text{train}} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, that is independent and identically distributed (i.i.d.) and sampled from \mathcal{D} . Here, n is the size of the training dataset. Our goal in this setting is to predict the label y of a new input \mathbf{x} , where again $(\mathbf{x}, y) \sim \mathcal{D}$. For this, we use a learning algorithm, that takes as input D_{train} and produces a classifier $f : \mathbb{R}^d \rightarrow [c]$. Empirically, we measure the performance of a classifier f on some (unseen) test dataset, D_{test} , that is also i.i.d. and sampled from \mathcal{D} . In standard (non-robust) supervised learning we aim to maximize the accuracy on this test data.

Definition 1. For a classifier f and data D the *accuracy* is given as

$$\text{acc}(f, D) := \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \mathbb{1}[f(\mathbf{x}) = y]. \quad (2.1)$$

2.2 Robust Classification

The focus of this thesis is on robust classification. Robustness is an important property in many different settings. For example, plenty of research focused on generating models that still perform well if the underlying data distribution, \mathcal{D} , changes between training and inference [QCSSL09]. However, in this work we focus on robustness to perturbations of the input, and we furthermore consider the worst-case setting: We assume an adversary perturbs the inputs in order to maximally decrease the accuracy of our model. Our goal in this setting is to reach high accuracy despite those perturbations.

Mathematically, we not only want classifiers that are correct on inputs \mathbf{x} sampled from \mathcal{D}_x , but we furthermore want to predict the same label for every $\tilde{\mathbf{x}}$ (not necessary in the support

of \mathcal{D}_x) that is close to \mathbf{x} . This property guarantees that small perturbations applied to \mathbf{x} cannot change classifier’s prediction. In this thesis, we usually measure closeness using the Euclidean norm, denoted by $\|\cdot\|_2$. Restricting perturbations to small Euclidean norm prevents the adversary to change the label of an image as perceived by humans.

We consider an input \mathbf{x} as *robustly classified* by a model under input perturbations up to size ϵ , if $\mathbf{x} + \boldsymbol{\delta}$ is correctly classified for all $\boldsymbol{\delta}$ with $\|\boldsymbol{\delta}\|_2 \leq \epsilon$:

Definition 2. The *robust accuracy* of a classifier f for perturbations of size at most ϵ on data D is given as

$$\text{RA}(f, D, \epsilon) := \frac{1}{|D|} \sum_{(x,y) \in D} \mathbb{1} \left[f(\tilde{\mathbf{x}}) = y \quad \forall \tilde{\mathbf{x}} : \|\tilde{\mathbf{x}} - \mathbf{x}\|_2 \leq \epsilon \right]. \quad (2.2)$$

There are many approaches that aim to generate classifiers with high robust accuracy, including adversarial training, randomized smoothing and 1-Lipschitz networks.

2.2.1 Adversarial Training

The most common approach is adversarial training [SZS⁺14, GSS15]. In adversarial training, we want to optimize our model to not only work well on standard inputs, but also on adversarially perturbed versions. In order to achieve this we rely on an *attack* η that aims to produce adversarial examples given $(\mathbf{x}, y) \sim \mathcal{D}$ and a model f . At every training step we first use η to find adversarial examples against the current model f . Afterwards we improve the model to perform better on those adversarial examples. We repeat this many times.

When training this way, it becomes harder to find adversarial examples for the final model. However, it is not possible to guarantee that none exist, so we cannot evaluate the robust accuracy or even a (non-trivial) lower bound of it.

2.2.2 Randomized Smoothing

Randomized Smoothing [CRK19] is an alternative to adversarial training that does provide some guarantees of robustness. Based on a classifier $f : \mathbb{R}^d \rightarrow [c]$, for c the number of classes, the *smoothed classifier* h is defined as

$$h(\mathbf{x}) = \arg \max_{i \in [c]} \mathbb{P}(f(\mathbf{x} + \boldsymbol{\epsilon}) = i), \quad (2.3)$$

where $\boldsymbol{\epsilon}$ follows a certain multivariate Gaussian distribution: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$ for some fixed standard deviation σ . Suppose $f(\mathbf{x} + \boldsymbol{\epsilon})$ returns the two most likely classes with probabilities p_1 and p_2 . Then, its smoothed classifier is robust to perturbations of size $\frac{\sigma}{2} (F_G^{-1}(p_1) - F_G^{-1}(p_2))$, for F_G^{-1} the inverse of a standard Gaussian cumulative distribution function. Notably, we cannot evaluate this classifier ¹. However, we can approximate it by sampling $\boldsymbol{\epsilon}$. Furthermore, we can also estimate the robust performance of this (theoretical) classifier using those samples.

Because the evaluation of this classifier relies on sampling, randomized smoothing can fail to provide the exact robust accuracy of the classifier. However, it can give a good approximation of it. This does usually require a high number of samples though, often many thousands are

¹When f is a piecewise-affine classifier, we could in theory evaluate the probability. This is usually not possible in practice though.

used to classify a single input, and we need to evaluate the base classifier f on every such sample. Therefore, randomized smoothing classifiers require a lot of compute for every input, especially when evaluating f is already compute intensive.

2.2.3 1-Lipschitz Classifiers

We can also obtain guarantees of robustness by using 1-Lipschitz classifiers [CBG⁺17]. Those provide guarantees by enforcing the K -Lipschitz property:

Definition 3. We call a function, a layer or a network $f : \mathbb{R}^a \rightarrow \mathbb{R}^b$ K -Lipschitz if for all input vectors \mathbf{x} and \mathbf{y} ,

$$\|f(\mathbf{x}) - f(\mathbf{y})\|_2 \leq K \|\mathbf{x} - \mathbf{y}\|_2. \quad (2.4)$$

We extend this definition to functions taking tensors as input and/or output by considering the L_2 norm of the flattened tensor. The *Lipschitz constant* of f is the smallest such K .

With this definition we can define 1-Lipschitz classifiers:

Definition 4. A *1-Lipschitz classifier* is a function of the form $f(\mathbf{x}) = \arg \max_{i \in \{1, \dots, c\}} g(\mathbf{x})_i$, where $g : \mathbb{R}^d \rightarrow \mathbb{R}^c$ is a 1-Lipschitz function.

We will refer to the output of g as the *class scores* predicted by classifier f . It turns out that for 1-Lipschitz classifiers, for an input x we can give guarantees of robustness based on those outputs of g in terms of the classification margin [TSS18]:

Definition 5. For a vector of class scores $\mathbf{s} \in \mathbb{R}^c$ and a label $y \in [c]$ we define the *classification margin* as

$$\text{margin}(\mathbf{s}, y) := s_y - \max_{i \neq y} s_i. \quad (2.5)$$

Proposition 1. [TSS18] For an input \mathbf{x} , the 1-Lipschitz classifier f based on 1-Lipschitz function g is robust to perturbation of size up to $\frac{1}{\sqrt{2}} \text{margin}(g(\mathbf{x}), f(\mathbf{x}))$.

Proof. In order to prove this statement, we will use the elementary inequality that $a^2 + b^2 \geq 2ab$ and therefore $(|a| + |b|)^2 \leq 2(a^2 + b^2)$, as well as the 1-Lipschitz property of g . Write $y = f(\mathbf{x})$, then we have that for any $i \neq y$ and any $\tilde{\mathbf{x}}$:

$$g(\tilde{\mathbf{x}})_y - g(\tilde{\mathbf{x}})_i = g(\mathbf{x})_y - (g(\mathbf{x})_y - g(\tilde{\mathbf{x}})_y) - g(\mathbf{x})_i + (g(\mathbf{x})_i - g(\tilde{\mathbf{x}})_i) \quad (2.6)$$

$$\geq g(\mathbf{x})_y - g(\mathbf{x})_i - |g(\mathbf{x})_y - g(\tilde{\mathbf{x}})_y| - |g(\mathbf{x})_i - g(\tilde{\mathbf{x}})_i| \quad (2.7)$$

$$\geq \text{margin}(g(\mathbf{x}), y) - \sqrt{2} \sqrt{(g(\mathbf{x})_y - g(\tilde{\mathbf{x}})_y)^2 + (g(\mathbf{x})_i - g(\tilde{\mathbf{x}})_i)^2} \quad (2.8)$$

$$\geq \text{margin}(g(\mathbf{x}), y) - \sqrt{2} \|g(\mathbf{x}) - g(\tilde{\mathbf{x}})\|_2 \quad (2.9)$$

$$\geq \text{margin}(g(\mathbf{x}), y) - \sqrt{2} \|\mathbf{x} - \tilde{\mathbf{x}}\|_2. \quad (2.10)$$

Therefore, as long as $\|\mathbf{x} - \tilde{\mathbf{x}}\|_2 \leq \frac{1}{\sqrt{2}} \text{margin}(g(\mathbf{x}), y)$, we have that $g(\tilde{\mathbf{x}})_y \geq g(\tilde{\mathbf{x}})_i$ for all $i \neq y$, and therefore $f(\tilde{\mathbf{x}}) = y$. So f predicts label y for all $\tilde{\mathbf{x}}$ within the specified margin. \square

This result allows us to compute a lower bound to the robust accuracy defined in Equation (2.2):

Definition 6. For a 1-Lipschitz classifier f based on a 1-Lipschitz function g , we define the *certified robust accuracy* on data D for perturbations of size up to ϵ as

$$\text{CRA}(f, D, \epsilon) := \frac{1}{|D|} \sum_{(x,y) \in D} \mathbb{1} \left[\text{margin}(g(\mathbf{x}), y) > \sqrt{2}\epsilon \right]. \quad (2.11)$$

By Proposition 1 we know that $\text{RA}(f, D, \epsilon) \geq \text{CRA}(f, D, \epsilon)$, therefore, 1-Lipschitz classifiers do allow us to provide guarantees about the robust accuracy. Furthermore, we can simply evaluate those guarantees by computing g a single time. Therefore, as long as we have a suitable 1-Lipschitz function g , we get robustness guarantees without additional computational overhead. We will describe a way of obtaining g in the following section.

2.3 1-Lipschitz Networks

Deep learning has proven that we can approximate complicated functions well by training a neural network to represent them. A neural network is usually the concatenation of a set of layers, where each layer is a simple function parameterized by some parameter tensor. We can *train* a neural network by iteratively updating those parameters.

In order to obtain a 1-Lipschitz function g , we also chose to represent it as a neural network. We do require a way of controlling the Lipschitz constant of g . For standard neural networks, it is practically impossible to evaluate the Lipschitz constant. Therefore, in lack of any better methods, we choose to restrict the Lipschitz constant of every single layer to be at most 1. As concatenations of 1-Lipschitz functions are also 1-Lipschitz, this is sufficient to ensure to 1-Lipschitz property on the whole network.

In order to construct 1-Lipschitz networks, many different kinds of 1-Lipschitz layers are useful. This includes convolutions, fully connected layers, activation functions and auxiliary layers to change the tensor size during the forward pass. We will discuss them below.

2.3.1 1-Lipschitz layers

We will first describe fully connected layers. For those, the Lipschitz constant is equal to the *spectral norm* of the weight matrix, which is defined as

$$\|M\|_{\text{spec}} := \sup_{\mathbf{v} \neq 0} \frac{\|M\mathbf{v}\|_2}{\|\mathbf{v}\|_2}. \quad (2.12)$$

Any algorithm that evaluates the spectral norm of a matrix can therefore be used to find the Lipschitz constant of a fully connected layer. For example, $\|M\|_{\text{spec}}$ is equal to the largest eigenvalue of $M^T M$, so we can use power iteration [MPG29] to find the Lipschitz constant of linear layers. (See Section 4.1.1 for details.) Once we have found the spectral norm of the weight matrix, we can divide the outputs of the layer by that value to enforce the 1-Lipschitz property.

Another commonly used layer is a convolution (see e.g. Equation (3.8)). Any convolution is a linear map from input to output, therefore we can rewrite it as matrix multiplication applied to the flattened input. The matrix we need to use for this is equal to the Jacobian matrix J , the matrix of partial derivatives of the flattened outputs with respect to the flattened inputs.

For a layer f it is given as

$$J_{ij} := \frac{\partial f(\mathbf{x})_i}{\partial x_j}. \quad (2.13)$$

By using the Jacobian, in theory we could rely on algorithms such as power iteration to make convolutions 1-Lipschitz. However, that is usually not feasible because of computational restrictions. In particular, for a convolution with input size $s \times s \times c$, the Jacobian will be of size $ssc \times ssc$, it is usually not practical to work with this matrix explicitly. Therefore, plenty of other methods have been proposed to generate 1-Lipschitz convolutions, we will discuss them in Chapter 3 and Chapter 4.

For the two layers described above, it is actually also possible to enforce that Equation (2.4) holds with equality for all \mathbf{x} and \mathbf{y} . For fully connected layers, this is the case if the weight matrix is *orthogonal*, where a matrix M is orthogonal if $MM^\top = I$, for I the identity matrix. For convolutions, we require the Jacobian to be orthogonal in order to enforce equality in Equation (2.4). We will refer to such a layer as an *orthogonal convolution*. We describe some methods that create those in Chapter 4.

For 1-Lipschitz networks, the choice of activation function also has an important influence on the performance. We could use ReLU as activation function, as it has the 1-Lipschitz property. However previous work has shown that using ReLU restricts the kind of functions a network can express, therefore usually MaxMin [CN16, ALG19] is used. This activation function acts on vectors by sorting every pair of entries by size:

$$\begin{aligned} \text{MaxMin}(\mathbf{x})_{2i+1} &:= \max(x_{2i+1}, x_{2i+2}) \\ \text{MaxMin}(\mathbf{x})_{2i+2} &:= \min(x_{2i+1}, x_{2i+2}). \end{aligned} \quad (2.14)$$

In order to train 1-Lipschitz networks, often the choice of initialization is very important. We can make initialization slightly simpler by enforcing each linear layer to preserve the tensor size. In order to do this, we need some additional layers that change the number of channels. We use a layer we call *ZeroChannelPadding*, to append channels with value 0 to the input. The layer *FirstChannels(c)* outputs only the first c input channels and omits the rest.

Finally, in 1-Lipschitz networks we also use specific pooling layers. Using the otherwise popular MaxPooling layer usually leads to low performance. Therefore, in 1-Lipschitz networks we usually use *ConcatenationPooling* or *PixelUnshuffle* to change the resolution. Those layers reshape patches of size $2 \times 2 \times c$ to shape $1 \times 1 \times 4c$, amongst them, they differ in the order the elements appear in the output.

Almost Orthogonal Lipschitz Layers

This chapter appears in full in [PL22], Bernd Prach and Christop H. Lampert, Almost-orthogonal layers for efficient general-purpose Lipschitz networks, first published in the *Proceedings of the 17th European Conference on Computer Vision (ECCV 2022)*, pp. 350-365 by Springer Nature. Reproduced with permission from Springer Nature.

In this work we proposed *Almost Orthogonal Lipschitz (AOL)*, a method of parameterizing 1-Lipschitz linear layers (both fully connected layers as well as convolutions). Compared to existing methods at the time, AOL was the first competitive method that **guarantees** a low Lipschitz constant and can be computed **efficiently**, without the need for inner iterative procedures or expensive operations.

3.1 Motivation

As described in Chapter 1, 1-Lipschitz networks are a very promising approach to guarantee robust classification. However, the existing methods often come with at least one of two shortcomings: The first one is a lack of guarantees, many methods only encourage a low Lipschitz constant using regularization, but cannot guarantee that it is small. The second shortcoming is high computational requirements. Many methods require evaluating an iterative procedure in every forward pass, adding a computational overhead to each layer. This also makes implementing those methods more complex and error-prone. We solve this problem, and introduce a rescaling-based method that guarantees a Lipschitz constant of 1 at all times and allows us to train networks to good robust accuracy. The rescaling is calculated explicitly from the parameter values, allowing for simple implementation.

3.2 Related Work

We will discuss related work at the time of publication in this section. For an overview of more recent 1-Lipschitz convolutions, and more details to convolutions mentioned in this section, see e.g. Chapter 4.

Early attempts of constructing networks with a small Lipschitz constant relied on regularization techniques that do not provide tight bounds on the Lipschitz constant. Regularization techniques include weight clipping [ACB17], or using an additional regularization term in the

loss function encouraging orthogonal weight matrices [CBG⁺17] or orthogonal convolutions [WCCY20]. Unfortunately, regularization methods do not provide the formal guarantees we require for certified robust classification, therefore other methods have been more popular.

A few methods rely on the power iteration algorithm [MPG29] in order to give guarantees of the Lipschitz constant of a layer. When run to convergence, power iteration gives the Lipschitz constant of any linear layer, so it is a very useful tool for robust classification. Some methods that use power iteration directly divide weight matrices by their spectral norm [MKKY18], which can also be done efficiently and tightly for convolutions [FZT18]. Other methods control the Lipschitz constant of a network by including it in the loss function [TSS18, LWF21].

Another set of approaches uses explicit orthogonalization in order to create 1-Lipschitz layers. There are plenty of (differentiable) ways of generating orthogonal matrices, for example the iterative procedure by Björck and Bowie [BB71]. It was used to create orthogonal fully connected layers [ALG19], and also to parameterize kernels of orthogonal convolutions in *Block Orthogonal Convolution Parameterization (BCOP)* [LHA⁺19]. Another method, *Orthogonalization by Newton's Iteration (ONI)* [HLZ⁺20], uses the relation $W = (VV^T)^{-1/2}V$ in order to parameterize an orthogonal weight matrix W . The authors use an iterative Newton method to compute W . Another known way of parameterizing orthogonal matrices is by using the *Cayley Transform* [Cay46]. In [TK21] the authors further use the discrete Fourier transform in order to parameterize orthogonal convolutions. Finally, a *Skew Orthogonal Convolution (SOC)* [SF21b] is an orthogonal layer parameterized using the exponential map, this method can be used for fully connected layers as well as convolutions. See Chapter 4 for more details on methods mentioned in this paragraph.

3.3 Almost-Orthogonal Lipschitz (AOL) Layers

In this section, we introduce our main contribution, Almost-Orthogonal Lipschitz (AOL) layers, which provide guaranteed Lipschitz bounds without the need for inner iterative methods such as power iteration. Specifically, we introduce a weight-dependent rescaling technique for the weights of a linear neural network layer that guarantees the layer to be 1-Lipschitz. It can be easily computed in closed form and is applicable to fully-connected as well as convolutional layers.

The main ingredient is the following theorem, which provides an elementary formula for controlling the spectral norm of a matrix by rescaling its columns.

Theorem 1. *For any matrix $P \in \mathbb{R}^{n \times m}$, define $D \in \mathbb{R}^{m \times m}$ as the diagonal matrix with*

$$D_{ii} = \left(\sum_j |P^\top P|_{ij} \right)^{-1/2} \quad (3.1)$$

if the expression in the brackets is non-zero, or $D_{ii} = 0$ otherwise. Then the spectral norm of PD is bounded by 1.

Proof. The upper bound of the spectral norm of PD follows from an elementary computation. By definition of the spectral norm, we have

$$\|PD\|_{\text{spec}}^2 = \max_{\|v\|_2=1} \|PDv\|_2^2 = \max_{\|v\|_2=1} v^\top D^\top P^\top PDv. \quad (3.2)$$

We observe that for any symmetric matrix $M \in \mathbb{R}^{n \times n}$ and any $\mathbf{w} \in \mathbb{R}^n$:

$$\mathbf{w}^T M \mathbf{w} \leq \sum_{i,j=1}^n |M_{ij}| |w_i| |w_j| \leq \sum_{i,j=1}^n \frac{1}{2} |M_{ij}| (w_i^2 + w_j^2) = \sum_{i=1}^n \left(\sum_{j=1}^n |M_{ij}| \right) w_i^2, \quad (3.3)$$

where the second inequality follows from the general relation $2ab \leq a^2 + b^2$. Using (3.3) with $M = P^\top P$ and $\mathbf{w} = D\mathbf{v}$, we obtain that for all \mathbf{v} with $\|\mathbf{v}\|_2 = 1$

$$\mathbf{v}^\top D^\top P^\top P D \mathbf{v} \leq \sum_{i=1}^n \left(\sum_{j=1}^n |P^\top P|_{ij} \right) (D_{ii} v_i)^2 \leq \sum_{i=1}^n v_i^2 = 1, \quad (3.4)$$

which proves the bound. \square

Note that the bound in Equation (3.4) was constructed in a way such that it is tight when P has orthogonal columns of full rank. In this case we have that $P^\top P$ is diagonal, and $D_{ii} = (P^\top P)_{ii}^{-1/2}$, so PD is an orthogonal matrix.

In the rest of this section, we demonstrate how Theorem 1 allows us to control the Lipschitz constant of any linear layer in a neural network.

3.3.1 Fully-Connected Lipschitz Layers

We first discuss the case of fully-connected layers.

Corollary 1 (Fully-Connected AOL Layers). *Let $P \in \mathbb{R}^{n \times m}$ be an arbitrary parameter matrix. Then, the fully-connected network layer*

$$f(x) = Wx + b \quad (3.5)$$

is guaranteed to be 1-Lipschitz, when $W = PD$ for D defined as in Theorem 1.

Proof. The Corollary follows directly from Theorem 1 as the Lipschitz constant of f is equal to the spectral norm of its weight matrix W . \square

Discussion. Despite its simplicity, there are a number of aspects of Corollary 1 that are worth a closer look. First, we observe that a layer of the form $f(x) = PDx + b$ can be interpreted in two ways, depending on how we (mentally) put brackets into the linear term. In the form $f(x) = P(Dx) + b$, we apply an arbitrary weight matrix to a suitably rescaled input vector. In the form $f(x) = (PD)x + b$, we apply a column-rescaling operation to the weight matrix before applying it to the unchanged input.

The two views highlight different aspects of AOL. The first view reflects the flexibility and high capacity of learning with an arbitrary parameter matrix, with only an intermediate rescaling operation to prevent the growth of the Lipschitz constant. The second view shows that AOL layers can be implemented without any overhead at prediction time, because the rescaling factors can be absorbed in the parameter matrix itself, even preserving potential structural properties such as sparsity patterns.

As a second insight from Corollary 1 we obtain how AOL relates to prior methods that rely on orthogonal weight matrices. As derived after Theorem 1, if the parameter matrix P has orthogonal columns of full rank, then $W = PD$ is an orthogonal weight matrix. In particular,

when P is already an orthogonal matrix, then D will be the identity matrix, so no rescaling will be applied. Therefore, our method can express any linear map based on an orthogonal matrix, but it can also express other linear maps. If the columns of P are approximately orthogonal, in the sense that $P^\top P$ is approximately diagonal, then the entries of D are dominated by the diagonal entries of the product. The multiplication by D acts mostly as a normalization of the length of the columns of P , and the resulting W is again an almost-orthogonal matrix.

Finally, observe that Corollary 1 does not put any specific numeric or structural constraints on the parameter matrix. Consequently, there are no restrictions on the optimizer or objective function when training AOL-networks.

3.3.2 Convolutional Lipschitz Layers

An analog of Corollary 1 for convolutional layers can, in principle, be obtained by applying the same construction as above: Convolutions are linear operations, so we can compute their Jacobian matrix and determine an appropriate rescaling matrix from it. However, this naive approach is computationally very expensive, because it requires working with matrices that are of size $ssc \times ssc$, for inputs of size $s \times s \times c$. Instead, by a more refined analysis, we obtain the following result.

Theorem 2 (Convolutional AOL Layers). *Let $P \in \mathbb{R}^{k \times k \times c_i \times c_o}$, be a convolution kernel tensor, where $k \times k$ is the kernel size and c_i and c_o are the number of input and output channels, respectively. Then, the convolutional layer*

$$f(x) = P * R(x) + b \quad (3.6)$$

is guaranteed to be 1-Lipschitz, where $R(x)$ is a channel-wise rescaling that multiplies each channel $c \in [c_i]$ of the input by

$$d_c = \left(\sum_{i=1}^{2k-1} \sum_{j=1}^{2k-1} \sum_{a=1}^{c_i} \left| P^\top * P_{:, :, c, :} \right|_{i, j, a} \right)^{-1/2}. \quad (3.7)$$

*We can equivalently write f as $f(x) = W * x + b$, where W has entries given by $W_{i, j, a, b} = d_a P_{i, j, a, b}$.*

Here, the transpose in Equation (3.7) is applied to the last two (channel) dimensions of the kernel and $*$ in the same equation denotes the discrete 2-dimensional multi-channel cross-correlation operation, which for kernel K of size $k \times k \times c_i \times c_o$ is given as

$$(K * x)_{i, j, a} := \sum_{p=1}^k \sum_{q=1}^k \sum_{b=1}^{c_i} K_{p, q, b, a} x_{(i-k+p), (j-k+q), b}. \quad (3.8)$$

Throughout this section we will define $x_{i, j} = 0$ when i or j are outside the valid indices.

For the proof we first apply Theorem 1 to the Jacobian of the convolution in order to obtain a rescaling for every entry of the input (expressed as a function of the entries of the Jacobian). Then we express those rescaling values in terms of the entries of the kernel. Finally, we show that these terms are lower bounded by the term in Equation (3.7).

Proof. For the proof we will first assume *maximal* padding of the input: We pad the input $x \in \mathbb{R}^{s \times s \times c_i}$ (with values independent of x) to a size of $(s + 2k - 2) \times (s + 2k - 2) \times c_i$, and

then apply the convolution to the padded input. From this we obtain an output of size $t \times t \times c_o$ for $t = s + k - 1$. We will derive a rescaling of the input that ensures that this convolution has a Lipschitz constant of 1. Then, any convolution with a different kind of padding (such as *same* size or *valid*) can be considered as first doing a maximally padded convolution, followed by a center cropping operation. Since cropping has a Lipschitz constant of 1, this also shows that convolutional layers with a different kind of padding have the 1-Lipschitz property. Note that this maximally padded convolution is given by Equation (3.8).

In order to obtain our rescaling, we first consider this convolution as a linear map, with weight matrix equal to the Jacobian, J . The entries of J are given by

$$J_{(i_1, j_1, a_1), (i_2, j_2, a_2)} := \frac{\partial (P * x)_{i_1, j_1, a_1}}{\partial x_{i_2, j_2, a_2}} \quad (3.9)$$

$$= \sum_{p=1}^k \sum_{q=1}^k \sum_{b=1}^{c_i} P_{p, q, b, a_1} \frac{\partial x_{(i_1-k+p), (j_1-k+q), b}}{\partial x_{i_2, j_2, a_2}} \quad (3.10)$$

$$= P_{(i_2-i_1+k), (j_2-j_1+k), a_2, a_1} \quad (3.11)$$

for any $i_1, j_1 \in [t]$, $a_1 \in [c_o]$, $i_2, j_2 \in [s]$ and $a_2 \in [c_i]$.

Applying Theorem 1 with $P = J$ gives us the necessary rescaling: In order to guarantee that the convolution has Lipschitz constant 1, we need to multiply input x_{i_2, j_2, a_2} by

$$\left(\sum_{i_1=1}^s \sum_{j_1=1}^s \sum_{a_1=1}^{c_i} |J^\top J|_{(i_1, j_1, a_1), (i_2, j_2, a_2)} \right)^{-1/2} \quad (3.12)$$

We use Equation (3.11) to obtain an expression for $J^\top J$ (for any $i_1, i_2, j_1, j_2 \in [s]$ and $a_1, a_2 \in [c_i]$):

$$[J^\top J]_{(i_1, j_1, a_1), (i_2, j_2, a_2)} \quad (3.13)$$

$$= \sum_{i=1}^t \sum_{j=1}^t \sum_{b=1}^{c_o} J_{(i, j, b), (i_1, j_1, a_1)} J_{(i, j, b), (i_2, j_2, a_2)} \quad (3.14)$$

$$= \sum_{i=1}^t \sum_{j=1}^t \sum_{b=1}^{c_o} P_{(i_1-i+k), (j_1-j+k), a_1, b} P_{(i_2-i+k), (j_2-j+k), a_2, b} \quad (3.15)$$

$$= \sum_{i=1}^k \sum_{j=1}^k \sum_{b=1}^{c_o} P_{i, j, a_1, b} P_{(i+i_2-i_1), (j+j_2-j_1), a_2, b} \quad (3.16)$$

$$= \left(P^\top * P_{:::, a_2, :} \right)_{(i_2-i_1+k), (j_2-j_1+k), a_1} \quad (3.17)$$

where the transpose operation is again over the last two (channel) dimensions.

Using this in in Equation (3.12) the rescaling becomes

$$\left(\sum_{i_1=1}^s \sum_{j_1=1}^s \sum_{a_1=1}^{c_i} \left| P^\top * P_{:::, a_2, :} \right|_{(i_2-i_1+k), (j_2-j_1+k), a_1} \right)^{-1/2} \quad (3.18)$$

A lower bound of this expression (that is tight for most values of i_2 and j_2) is given by

$$\left(\sum_{i=1}^{2k-1} \sum_{j=1}^{2k-1} \sum_{a=1}^{c_i} \left| P^\top * P_{:::, c, :} \right|_{i, j, a} \right)^{-1/2} \quad (3.19)$$

This term is independent of both i_2 and j_2 , and it is exactly the channel rescaling in Theorem 2, so that completes our proof for maximally padded convolutions. This further implies that convolutions with less padding are also 1-Lipschitz when the input is rescaled as described. \square

Discussion. We now discuss some favorable properties of Theorem 2. First, as in the fully-connected case, the rescaling operation again can be viewed either as acting on the inputs, or as acting on the parameter matrix. Therefore, the convolutional layer also combines the properties of high capacity and no overhead at inference time. In fact, for 1×1 convolutions, the construction of Theorem 2 reduces to the fully-connected situation of Corollary 1.

Second, computing the scaling factors is efficient, because the necessary operations scale only with the size of the convolution kernel regardless of the image size. The rescaling preserves the structure of the convolution kernel, e.g. sparsity patterns. In particular, this means that constructs such as dilated convolutions are automatically covered by Theorem 2 as well, as these can be expressed as ordinary convolutions with specific zero entries.

Furthermore, Theorem 2 requires no strong assumption on the padding type. Also, the computation of the scale factors is easy to implement in all common deep learning frameworks using batch-convolution operations.

3.4 Offset Cross-Entropy Loss Function

In order to train the network to achieve good certified robust accuracy we want the score of the correct class to be bigger than any other score by a margin (see e.g. Proposition 1). In order to encourage a large margin during training, we extend the loss function from *Lipschitz-margin training* [TSS18], and add an temperature parameter t . Our proposed loss function *OffsetCrossEntropy*, depends on an offset u and a temperature parameter t . It takes the class score vector s generated by the 1-Lipschitz network as well as a one-hot encoding y of the true label as an input, and it is given as

$$\mathcal{L}(s, y) = \text{cross-entropy} \left(y, \text{softmax} \left(\frac{s - uy}{t} \right) \right) t. \quad (3.20)$$

We designed this loss function with the goal that its value should be close to 0 as soon as the network achieves the desired offset. This is not the case for the loss from Lipschitz-margin training, which is equivalent to our loss function with $t = 1$. However, our temperature parameter allows us to achieve this, as we demonstrate in Figure 3.1. There we show the value of our loss function for offset $\sqrt{2}$ and temperature $t \in \{\frac{1}{16}, \frac{1}{4}, 1\}$, for a score vector of dimension 10, with the score of the correct class varied along the x-axis and all other scores set to 0. We can see that with temperature 1, the margin needs to be much higher than the chosen offset before the loss is small, whereas with small temperature the value of the loss function is close to 0 when the margin is equal to our goal offset.

3.5 Experimental Setup

We compare our method to related work in certified robust classification. Following prior work in the field, we evaluate the certified robust accuracy for perturbations up to ϵ for different values of ϵ on CIFAR-10 as well as the CIFAR-100. We also provide ablation studies that illustrate that AOL can be used in a variety of network architectures, and that it indeed learns

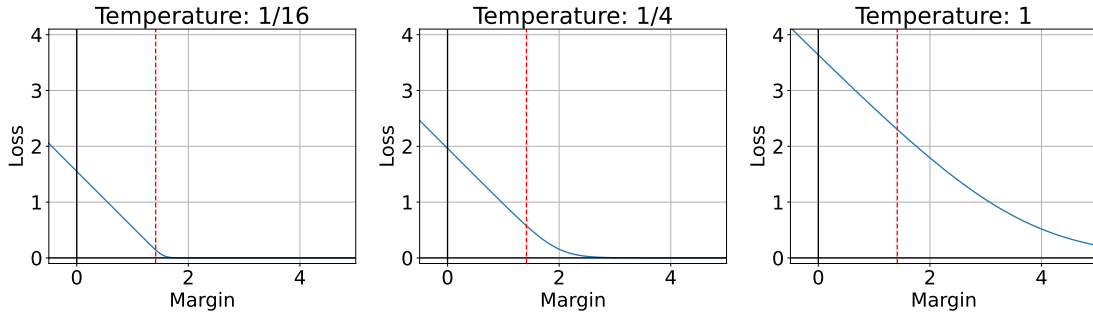


Figure 3.1: $\text{OffsetCrossEntropy}(u, t)$ encourages a much larger margin than goal offset u for temperature parameter $t \geq 1$. Here, the offset $u = \sqrt{2}$ is shown in the plot by the red line.

Table 3.1: **Patchwise architecture** for c classes and $w \in \{16, 32, 48\}$.

Layer name	Kernel size	Stride	Activation	Output size	Amount
Concatenation Pooling	4×4	4×4	-	$8 \times 8 \times 48$	1
AOL Conv	1×1	1×1	MaxMin	$8 \times 8 \times 192$	1
AOL Conv	3×3	1×1	MaxMin	$8 \times 8 \times 192$	12
AOL Conv	1×1	1×1	None	$8 \times 8 \times 192$	1
First Channels	-	-	-	$8 \times 8 \times w$	1
Flatten	-	-	-	$64w$	1
AOL FC	-	-	MaxMin	$64w$	13
AOL FC	-	-	None	$64w$	1
First Channels	-	-	-	c	1

matrices that are approximately orthogonal. In the following we describe our experimental setup. All hyperparameters were determined on validation sets.

Architecture: Our main model architecture is loosely inspired by the *ConvMixer* architecture [TK22]: We first subdivide the input image into 4×4 patches, which are processed by 14 convolutional layers, usually with kernel size 3×3 . This is followed by 14 fully connected layers. We report results for three different model sizes. For AOL-Small we set w in Table 3.1 to 16, and we choose $w = 32$ and $w = 48$ for AOL-Medium and AOL-Large. For all layers we use zero padding to keep the size the same. We use the MaxMin activation function. The full architectural details can be found in Table 3.1.

Initialization: In order to ensure stable training we initialize the parameter matrices so that our bound is tight. In particular, for layers preserving the size between input and output we initialize the parameter matrix so that the Jacobian is the identity matrix. For the first convolution, that increases the size, we initialize the parameter matrix so that it has random orthogonal columns.

Optimization: We train our models to minimize the loss function proposed in Equation (3.20). We use $u = \sqrt{2}$, which encourages the model to learn to classify the training data certifiably robustly to perturbations of norm ≤ 1 . We set the temperature to $t = 1/4$, and optimize using stochastic gradient descent (SGD) with Nesterov momentum of 0.9 for 1000 epochs. The

Table 3.2: Certified Robust Classification on CIFAR-10.

Method	Accuracy	Certified Robust Accuracy			
		$\epsilon = \frac{36}{255}$	$\epsilon = \frac{72}{255}$	$\epsilon = \frac{108}{255}$	$\epsilon = 1$
BCOP [LHA ⁺ 19] _{Large}	72.2%	58.3%	-	-	-
GloRo [LWF21] _{6C2F}	77.0%	58.4%	-	-	-
Cayley [TK21] _{Large}	75.3%	59.2%	-	-	-
SOC [SF21b] ₂₀	76.4%	61.9%	-	-	-
SOC + CR [SF22]	76.4%	63.0%	48.5%	35.5%	-
AOL-Small	69.8%	62.0%	54.4%	47.1%	21.8%
AOL-Medium	71.1%	63.8%	56.1%	48.6%	23.2%
AOL-Large	71.6%	64.0%	56.4%	49.0%	23.7%

Table 3.3: Certified Robust Classification on CIFAR-100.

Method	Accuracy	Certified Robust Accuracy (CRA)			
		$\epsilon = \frac{36}{255}$	$\epsilon = \frac{72}{255}$	$\epsilon = \frac{108}{255}$	$\epsilon = 1$
SOC [SF21b] ₃₀	43.1%	29.2%	-	-	-
SOC + CR [SF22]	47.8%	34.8%	23.7%	15.8%	-
AOL-Small	42.4%	32.5%	24.8%	19.2%	6.7%
AOL-Medium	43.2%	33.7%	26.0%	20.2%	7.2%
AOL-Large	43.7%	33.7%	26.3%	20.7%	7.8%

batch size is 250. The learning rate starts at 10^{-3} and is reduced by a factor of 10 at epochs 900, 990 and 999. For all AOL layers we also use weight decay with coefficient 5×10^{-4} .

Data augmentation We use data augmentation in all our experiments. We first do some color augmentation of the images, followed by some spatial transformations. For the color augmentation, we first randomly adjust the hue of the image ($\delta \in [-0.02, 0.02]$), then the saturation (by a factor in $[\cdot 3, 2]$), then the brightness ($\delta \in [-0.1, 0.1]$), and finally the contrast (by a factor in $[\cdot 5, 2]$). After this we clip the pixel values to $[0., 1.]$. For the spatial transformations, we first apply a random rotation (up to 5 degrees), then random shifts (up to 10% of the image size), and finally we flip the image with a probability of 50%. We rely on the TensorFlow layers *RandomRotation* and *RandomTranslation* for the spatial transformations, and leave all hyperparameters such as the fill mode as the default values. All hyperparameters specifying the amount of augmentation were chosen based on visual inspection of the augmented training images.

3.6 Results

The main results can be found in Table 3.2 and Table 3.3, where we compare the certified robust accuracy of our method to those reported in previous works on orthogonal networks. For methods that are presented in multiple variants, such as different networks depths, we include the best performing variant.

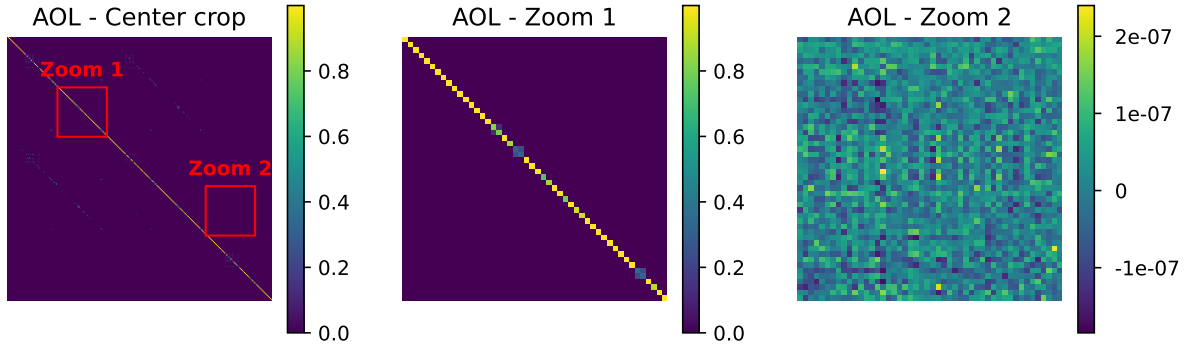


Figure 3.2: A center crop of $J^\top J$, for J the Jacobian, as well as two further crops.

The table shows that our proposed method achieves results competitive with the current state-of-the-art. Focusing on medium or higher robustness thresholds, AOL even achieves certified robust accuracy that is better than previous methods. Note that this is in part due to our choice of loss function. As a reference for future work, we also report values for an even higher robustness threshold than what appeared in the literature so far, $\epsilon = 1$.

Another observation is that on the CIFAR-10 dataset the accuracy of AOL is somewhat below other methods. We attribute this to the fact that we mainly focused our training towards high robustness. The accuracy-robustness trade-off can in fact be influenced by the choice of margin at training time, see our ablation study in Section 3.6.1.

3.6.1 Ablation Studies

Approximate orthogonality: As our first ablation study, we demonstrate that AOL indeed learns almost-orthogonal weight matrices, thereby justifying its name. In order to do that, we evaluate $J^\top J$ for J the Jacobian of an AOL convolution (more specifically, the third layer of the AOL-small model), and visualize parts of it in Figure 3.2. We see that indeed, $J^\top J$ is very close to the identity matrix, with a lot of elements on the diagonal close to 1, and most elements off the diagonal close to 0.

Accuracy-robustness trade-off: The loss function we introduced in Section 3.4 allows trading off between accuracy and certified robust accuracy by changing the size of the enforced margin. We demonstrate this by an ablation study that varies the offset parameter u in the loss function, and also scales t proportional to u . The results can be found in Table 3.4. Using a small margin allows us to train an AOL network with high accuracy, but decreases the certified robust accuracy for larger input perturbations, whereas choosing a higher offset allows us to reach state-of-the-art accuracy for larger input perturbations but reduce accuracy. Therefore, varying this offset gives us an easy way to prioritize the measure that is important for a specific problem.

Generality: One of the main advantages of AOL is that it is not restricted to a specific architecture or layer type. To demonstrate this, we conducted additional experiments with other architectures: *AOL-multilayer perceptron (MLP)* consists of 9 fully connected layers of width 4096. *AOL-Conv* resembles a standard convolutional architecture, where the number of channels starts as 32 and doubles whenever the resolution is reduced. It consists of 5 blocks with 5 convolutions each and uses *ConcatenationPooling* (described in Section 2.3). *AOL-alt* is an alternative convolutional architecture that quadruples the number of channels whenever

Table 3.4: Varying u and t in our loss function. AOL-small on CIFAR-10.

u	t	Accuracy	CRA			
			$\epsilon = \frac{36}{255}$	$\epsilon = \frac{72}{255}$	$\epsilon = \frac{108}{255}$	$\epsilon = 1$
$\sqrt{2}/16$	1/64	79.8%	45.3%	16.7%	3.3%	0.0%
$\sqrt{2}/4$	1/16	77.4%	63.0%	47.6%	33.0%	2.5%
$\sqrt{2}$	1/4	70.4%	62.6%	55.0%	47.9%	22.2%
$4\sqrt{2}$	1	59.8%	55.5%	50.9%	46.5%	30.8%
$16\sqrt{2}$	4	48.2%	45.2%	42.2%	39.4%	28.6%

Table 3.5: **Generality of AOL.** Experimental results with different architectures.

Method	Accuracy	CRA			
		$\epsilon = \frac{36}{255}$	$\epsilon = \frac{72}{255}$	$\epsilon = \frac{108}{255}$	$\epsilon = 1$
AOL-MLP	67.9%	59.1%	51.1%	43.1%	17.6%
AOL-Conv	65.0%	56.4%	48.2%	39.9%	15.8%
AOL-alt	68.4%	60.3%	52.4%	44.8%	19.9%

the resolution is reduced. This keeps the number of activations constant throughout the forward pass. We present experimental results in Table 3.5. They confirm that we can use AOL in various architectures without need for adaptation to the architecture, and we obtain reasonable certified robust accuracy with such architectures.

3.7 Discussion

In this section we proposed AOL, a method for constructing linear layers with Lipschitz constant of 1. It does not require any inner iterative methods, making AOL easier to use and implement than existing methods. We also proposed `OffsetCrossEntropy`, a loss function that allows us to optimize 1-Lipschitz networks to a chosen level of robustness more precisely. Together, these methods allow us to get state-of-the-art performance among 1-Lipschitz method (at the time of publication) for larger perturbation radii. We are also competitive to existing methods for small perturbations. However, it is also worth noting that we are still far away from our goal of robust classification with 1-Lipschitz networks. Even for a simple data such as CIFAR-10 and small perturbations, the certified robust accuracy we get is a lot worse than the (non-robust) accuracy we can achieve.

1-Lipschitz Layers Compared

In the last chapter we proposed a method of creating 1-Lipschitz linear layers. One observation we had there is that usually, larger models and longer training times do improve performance. Since we introduced AOL, more methods of creating 1-Lipschitz convolutions have been proposed, however, it is hard to compare them from the literature alone, as different papers use different model sizes, training times and different setups in general. Therefore, in this section we aim to fairly compare existing methods, both theoretically as well as experimentally. We evaluate the methods in terms of training time per epoch, inference time, memory requirement (training & inference), accuracy as well as certified robust accuracy.

This chapter appears in full in [PBBL24], Bernd Prach, Fabio Brau, Giorgio Buttazzo, and Christoph H. Lampert, 1-Lipschitz layers compared: Memory speed and certifiable robustness, first published in the *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

4.1 Background and Existing Work

In this section we will first introduce basic methods that allow parameterizing orthogonal matrices, and then describe all the 1-Lipschitz convolutions compared in this chapter.

4.1.1 Algorithms for controlling the spectral norm

A lot of recently proposed methods of creating 1-Lipschitz convolutions rely on a way of parameterizing either orthogonal matrices or matrices with bounded spectral norm. We present some useful methods below:

Bjorck & Bowie [BB71] introduced an iterative algorithm that approximates the closest orthogonal matrix to the given input matrix. In the commonly used form, this is achieved by computing a sequence of matrices using

$$A_{k+1} = A_k \left(I + \frac{1}{2} Q_k \right), \quad \text{for } Q_k = I - A_k^\top A_k, \quad (4.1)$$

where A_0 is equal to the input matrix. The algorithm is usually truncated after a fixed number of steps, during training often 3 iterations are enough, and for inference 15 iterations are used

to ensure a good approximation. Since the algorithm is differentiable, it can be directly used to construct 1-Lipschitz networks [ALG19] or also as an auxiliary method for more complex strategies [LHA⁺19].

Cayley Transform [Cay46] The *Cayley Transform* maps a skew-symmetric matrix A to an orthogonal matrix Q using the relation

$$Q = (I - A)(I + A)^{-1}, \quad (4.2)$$

where we call a matrix is skew-symmetric if $A^\top = -A$. We can use the Cayley Transform to parameterize orthogonal matrices, however, it does require the (often costly) inversion of a matrix.

Exponential Map For any skew-symmetric matrix A ,

$$\exp(A) := I + \sum_{k=1}^{\infty} \frac{A^k}{k!} = I + \frac{A}{1!} + \frac{A^2}{2!} + \frac{A^3}{3!} + \dots \quad (4.3)$$

is orthogonal. This series has the useful property that it always converges. Furthermore, if we can bound the operator norm of A , even just using a few terms gives us a good approximation of $\exp(A)$. In order to apply $\exp(A)$ to an input x we do not require (costly) matrix-matrix multiplication, but can instead compute the terms using only matrix-vector multiplication.

Power Method The *power method* [MPG29] was used in [MKKY18, FZT18, TSS18] in order to bound the spectral norm of matrices. The power method starts from a random initial vector \mathbf{b}_0 , and iteratively computes

$$\mathbf{b}_{k+1} = \frac{A\mathbf{b}_k}{\|A\mathbf{b}_k\|_2}, \quad \mu_k = \frac{\mathbf{b}_k^\top A\mathbf{b}_k}{\|\mathbf{b}_k\|_2}. \quad (4.4)$$

With this, the sequence (μ_k) converges to the eigenvalue of A with largest magnitude.

We can use the power method applied to $A = W^\top W$ in order to find the spectral norm of W . Note that we do not actually need to calculate the result of the matrix multiplication $W^\top W$ in order to apply the power method, but instead we can do 2 matrix-vector multiplications in every step.

The power method has been extended to efficiently calculate the Lipschitz constant of convolutional layers [FZT18]. The authors use the fact that the transpose of a convolution operation is a convolution as well, with a kernel that can be constructed from the original one by transposing the channel dimensions and flipping the spatial dimensions of the kernel. So for a kernel K and a random initial tensor \mathbf{u}_0 of shape $s \times s \times c$, we can define

$$\mathbf{v}_{k+1} = \frac{K * \mathbf{u}_k}{\|K * \mathbf{u}_k\|_2}, \quad \mathbf{u}_{k+1} = \frac{K *^\top \mathbf{v}_{k+1}}{\|K *^\top \mathbf{v}_{k+1}\|_2}, \quad \sigma_k = \sum_{i,j,c} (\mathbf{u}_k)_{i,j,c} (K *^\top \mathbf{v}_k)_{i,j,c} \quad (4.5)$$

for $*^\top$ denoting a transpose convolution. Then (σ_k) converges to the Lipschitz constant of the convolution.

When the power method is used on a parameter matrix of a layer during training, we can make it even more efficient. As we usually expect the parameter matrix to change only slightly during each update, we can store the result \mathbf{u}_k during each training step, and start the power method with this vector as \mathbf{u}_0 during the following training step. When using this, empirically it seems to be enough to do a single iteration of the power method at each training step. We usually do not use backpropagate through the steps of the power method.

4.1.2 Convolutions in Fourier Space

When computing the result of a convolution, in certain settings we can rely on simpler operations. By the *2D convolutional theorem* [Jai89], we can calculate the circular convolution of two matrices by their element-wise product in the Fourier domain. [TK21] extended this idea and showed that multi-channel convolutions with circular padding can be evaluated in the Fourier domain as a set of matrix-vector products. This gives us a much easier way of enforcing orthogonality of such convolutions, we can simply enforce orthogonality in the Fourier domain. As the discrete Fourier transform is also an orthogonal map, this allows us to parameterize orthogonal convolutions.

4.1.3 1-Lipschitz convolutions

This section provides an overview of the existing methods for providing 1-Lipschitz layers. The rest of this section describes 7 methods from the literature that construct 1-Lipschitz convolutions: BCOP, Cayley, SOC, AOL, LOT, CPL and SLL.

Block Orthogonal Convolution Parameterization (BCOP). *BCOP* [LHA⁺19] is based on a previous work [XBSD⁺18] that proposed a way of initializing kernels of orthogonal convolutions. In order to construct a $k \times k$ convolution, BCOP uses a set of $(2k - 1)$ parameter matrices. Each of these matrices is orthogonalized using the algorithm by Björck and Bowie [BB71]. Then, a $k \times k$ kernel is constructed from those matrices in a way that guarantees that the resulting convolution is orthogonal.

Cayley Layers (Cayley). Another family of orthogonal convolutional and fully connected layers has been proposed in [TK21] by leveraging the *Cayley Transform* [Cay46]. The transformation can be used to parameterize orthogonal weight matrices for linear layers in a straightforward way. For convolutions, the authors make use of the fact that circular padded convolutions are vector-matrix products in the Fourier domain. As long as all those vector-matrix products have orthogonal matrices, the full convolution will have an orthogonal Jacobian. For Cayley Layers, those matrices are orthogonalized using the Cayley transform.

Skew Orthogonal Convolution (SOC). SOC [SF21b] is an orthogonal convolutional layer obtained by using the exponential convolution [HGSTW20]. Analogously to the matrix case, given a kernel $K \in \mathbb{R}^{c \times c \times k \times k}$, the exponential convolution can be defined as

$$\exp(K)(x) := x + \frac{K \star x}{1} + \frac{K \star^2 x}{2!} + \dots + \frac{K \star^k x}{k!} + \dots, \quad (4.6)$$

where \star^k denotes a convolution applied k -times. The authors proved that any exponential convolution has an orthogonal Jacobian matrix as long as K is skew-symmetric, providing a way of parameterizing 1-Lipschitz layers. In their work, the sum of the infinite series is approximated by computing only the first 5 terms during training and the first 12 terms during the inference, and K is normalized to have unitary spectral norm using the method *Fantastic Four* [SF21a].

Almost Orthogonal Lipschitz (AOL). We also compare our own method [PL22], introduced in Chapter 3. To summarize our method, for any matrix P , we define a diagonal

rescaling matrix D with

$$D_{ii} = \left(\sum_j |P^\top P|_{ij} \right)^{-1/2}, \quad (4.7)$$

and prove that the spectral norm of PD is bounded by 1. We use this result to define a 1-Lipschitz linear layer given by $l(x) = PDx + b$, where P is the learnable matrix and D is given by Equation (4.7). Furthermore, we extend the idea so that it can also be efficiently applied to convolutions. This is done by applying the rescaling in Equation (4.7) to the Jacobian J of the convolution, expressing the elements in $J^\top J$ explicitly in terms of the kernel values, and using a lower bound of those rescaling values that is more efficient to compute.

Layer-wise Orthogonal Training (LOT). LOT [XLL22] extends the idea of [HLZ⁺20] to use the *Inverse Square Root* of a matrix in order to orthogonalize it. Indeed, for any matrix V , the matrix $Q = (VV^\top)^{-\frac{1}{2}}V$ is orthogonal. As in Cayley Convolutions, for LOT the convolution is applied in the Fourier frequency domain. To find the inverse square root, the authors rely on an iterative *Newton Method*. In details, defining $Y_0 = VV^\top$, $Z_0 = I$, and

$$Y_{k+1} = \frac{1}{2}Y_k(3I - Z_kY_k), \quad Z_{k+1} = \frac{1}{2}(3I - Z_kY_k)Z_k, \quad (4.8)$$

it can be shown that Z_k converges to $(VV^\top)^{-\frac{1}{2}}$. In their proposed layer, the authors apply 10 iterations of the method for both training and evaluation.

Convex Potential Layer (CPL). Given a non-decreasing 1-Lipschitz function σ (usually ReLU), CPL [MDAA22] is constructed as

$$l(x) = x - \frac{2}{\|W\|_2^2} W^\top \sigma(Wx + b), \quad (4.9)$$

which is 1-Lipschitz by design. The spectral norm required to calculate $l(x)$ is approximated using the power method (see Section 4.1.1).

SDP-based Lipschitz Layers (SLL). SLL [AHD⁺23], combine the CPL layer with the upper bound on the spectral norm from AOL. The SLL layer can be written as

$$l(x) = x - 2W^\top Q^{-2} D^2 \sigma(Wx + b), \quad (4.10)$$

where Q is a learnable diagonal matrix with positive entries and D is deduced by applying Equation (3.1) to $P = W^\top Q^{-1}$.

Remark Both CPL and SLL are non-linear by construction, so they can be used to construct a network without any further use of activation functions. However, carrying out some preliminary experiments, we empirically found that alternating CPL (and SLL) layers with MaxMin activation layers allows achieving a better performance.

4.2 Theoretical Comparison

As illustrated in the last section, various ideas and methods have been proposed to parameterize 1-Lipschitz layers. This section aims at highlighting the properties of different algorithms, focusing on the algorithmic complexity and the required memory.

Table 4.1: **Computational complexity and memory requirements of different methods**, for batch size b , input size $s \times s \times c$ and t inner iterations. We use $C = \mathcal{O}(bs^2c^2)$ and $M = \mathcal{O}(bs^2c)$. © 2024 IEEE.

Method	Input Transformations $\mathcal{O}(\cdot)$			Parameter Transformations $\mathcal{O}(\cdot)$		
	Operation	MACs	Memory	Operation	MACs	Memory
Standard	CONV	C	M	-	-	c^2
AOL	CONV	C	M	CONV	c^3	c^2
BCOP	CONV	C	M	BnB	c^3t	c^2t
Cayley	MVs	C	M	INVs	s^2c^3	s^2c^2
CPL	CONVs	C	M	PM	s^2c^2	$c^2 + s^2c$
LOT	MVs	C	M	MMs	s^2c^3t	s^2c^2t
SLL	CONVs	C	M	CONVs	c^3	c^2
SOC	CONVs	Ct_1	Mt_1	F4	c^2t_2	c^2

Table 4.1 provides an overview of the computational complexity and memory requirements for the different layers considered in the previous section. We report multiply-accumulate operations (MACs) as well as memory requirements per layer for batch size b and input size $s \times s \times c$. We denote the number of inner iterations by t . (Often, t is different at training and inference time.) We use C and M to denote MACs and memory requirement of a standard convolution, $C = \mathcal{O}(bs^2c^2)$ and $M = \mathcal{O}(bs^2c)$. In order to simplify some terms, we assume that $c > \log_2(s)$. Also, for Cayley, where the algorithm described in the paper and the version provided in the supplied code differed, we considered the algorithm implemented in the code.

The transformations reported in the table are convolutions (CONV), matrix-vector multiplications (MV), matrix-matrix multiplications (MM) and matrix inversions (INV). The application of algorithms such as Björck and Bowie (BnB), power method (PM), and Fantastic 4 (F4) is also reported.

For the sake of clarity, we perform the analysis by considering the transformations applied to the input of the layers separately from those applied to the weights to ensure the 1-Lipschitz constraint. Table 4.1 contains three columns for both of the transformations. These contain the most costly transformation, its computational complexity expressed in MACs, as well as the memory required by the transformation during the training phase.

At training time, both input and weight transformations are required, thus the training complexity of the forward pass can be computed as the sum of the two corresponding MACs columns of the table. Similarly, the training memory requirements can be computed as the sum of the two corresponding Memory columns of the table. For the considered operations, the cost of the backward pass during training has the same computational complexity as the forward pass, and therefore increases the overall complexity by a constant factor. At inference time, all the parameter transformations can be computed just once and cached afterward. Therefore, the inference complexity is equal to the complexity due to the input transformation (column 3 in the table). At inference time, the intermediate variables are not stored in memory, hence, the memory requirements are much lower than during training. The values cannot directly be inferred from Table 4.1, we reported them separately in Section 4.2.4.

4.2.1 Details and derivation

In this section we give some intuition of the values in Table 4.1. We will denote the kernel size as $k \times k$ for the detailed analysis.

AOL: In order to compute the rescaling matrix for AOL, we need to convolve the kernel with itself. This operation has complexity $\mathcal{O}(c^3k^4)$. It outputs a tensor of size $(2k - 1) \times (2k - 1) \times c \times c$, so in total we require memory of about $5c^2k^2$ for storing the parameters as well as the intermediate results from transformations.

BCOP: For BCOP we only require computing a single convolution as long as we know the kernel. However, we do require a lot of computation to create the kernel for this convolution. In particular, we require $2k - 1$ matrix orthogonalizations (usually done with Björck and Bowie), as well as $\mathcal{O}(k^3)$ matrix multiplications for building up the kernel. These require about $c^3kt + c^3k^3$ MACs as well as $c^2kt + c^2k^3$ memory.

Cayley: Cayley Convolutions make use of the fact that circular padded convolutions are vector-matrix products in the Fourier domain. Applying the fast Fourier transform to inputs and weights has complexity of $\mathcal{O}(bcs^2 \log(s^2))$ and $\mathcal{O}(c^2s^2 \log(s^2))$. Then, we need to orthogonalize $\frac{1}{2}s^2$ matrices. Note that the factor of $\frac{1}{2}$ appears due to the fact that the Fourier transform of a real matrix has certain symmetry properties, and we can use that fact to skip half the computations. Doing the matrix orthogonalization with the Cayley Transform requires taking the inverse of a matrix, as well as matrix multiplication, the whole process has a complexity of about s^2c^3 . The final steps consist of doing $\frac{1}{2}bs^2$ matrix-vector products, requiring $\frac{1}{2}bs^2c^2$ MACs, as well as another fast Fourier transform. Note that under our assumption that $c > \log(s^2)$, the fast Fourier transform operation is dominated by other operations. Cayley Convolutions require padding the kernel from a size of $k \times k \times c \times c$ to a (usually much larger) size of $s \times s \times c \times c$ requiring a lot of extra memory. In particular, we need to keep the output of the (real) fast Fourier transform, the matrix inversion as well as the matrix multiplication in memory, each requiring storing about $\frac{1}{2}s^2c^2$ floats.

CPL: For each CPL, we need to evaluate two convolutions as well as an activation function. We also need to do one step of the convolutional power method, however, the computational cost of this is dominated by convolutions. Therefore, the method requires $2C$ MACs and $3M$ memory.

LOT: Similar to Cayley, LOT performs the convolution in Fourier space. However, instead of using the Cayley transform, they parameterize orthogonal matrices as $V(V^TV)^{-\frac{1}{2}}$. To find the inverse square root, authors rely on an iterative *Newton Method*. Executing these iterations, includes computing $4s^2t$ matrix multiplications, requiring about $4s^2c^3t$ MACs as well as $4s^2c^2t$ memory.

SLL: Similar to CPL, each SLL layer also requires evaluating two convolutions as well as one activation. However, SLL also needs to compute the AOL rescaling, resulting in total computational cost of $2C + \mathcal{O}(c^3k^4)$.

SOC: For each SOC layer we require applying t convolutions. Other required operations (application of Fantastic 4 for an initial bound, as well as parameterizing the kernel such that the Jacobian is skew-symmetric) are cheap in comparison.

4.2.2 Analysis of the computational complexity

It is worth noting that the complexity of the input transformations (in Table 4.1) is similar for all methods. This implies that a similar scaling behavior is expected during **inference** for the models. Cayley and LOT apply an FFT-based convolution, therefore we expect the inference time to be different from other methods. CPL and SLL require two convolutions, which make forward passes slightly more expensive. Notably, each layer of SOC requires evaluating multiple convolutions, making forward passes using this method more expensive.

During **training**, parameter transformations need to be applied in addition to the input transformations during every forward pass. For SOC and CPL, the input transformations always dominate the parameter transformations in terms of computational complexity. This means the complexity scales like c^2 , just like a regular convolution, with a further factor of 2 and 5 respectively. All other methods require parameter transformations that scale like c^3 , making them more expensive for wider architectures. In particular, we do expect Cayley and LOT to require long training times for larger models, since the complexity of their parameter transformations further depends on the input size.

4.2.3 Analysis of the training memory requirements

The memory requirements of the different layers are important, since they determine the maximum batch size and the type of models we can train on a particular infrastructure. At training time, typically all intermediate results are kept in memory to perform backpropagation. This includes intermediate results for both input and parameter transformations. The input transformations usually preserve the size, and therefore the memory required is usually of $\mathcal{O}(M)$. Therefore, for the input transformations, all methods require memory not more than a constant factor worse than standard convolutions, with the worst method being SOC, with a constant t_1 , typically equal to 5.

In addition to the input transformation, we also need to store intermediate results of the parameter transformations in memory in order to evaluate the gradients. Again, most methods approximately preserve the sizes during the parameter transformations, and therefore the memory required is usually of order $\mathcal{O}(c^2)$. Exceptions to this rule are Cayley and LOT, with a much larger $\mathcal{O}(s^2c^2)$ term, as well as BCOP.

4.2.4 Memory requirements at inference time

The amount of memory required at inference time is much lower than what is needed during training. However, it might still be important, for example for efficient deployment of models. During inference time, we do not need to keep the activations in memory, but only need to store them while we are doing computations. Furthermore, all transformations on the weights can be cached once, and then we never need to perform them again. For this reason, most methods do not have any memory overhead at all over standard convolutions at inference time. In particular, a layer of methods AOL, BCOP, CPL, SLL or SOC does require memory of $M + c^2k^2$, just like a standard convolution ($M = bs^2c$). The methods that apply the convolution in the Fourier domain do require more memory. In particular, applying a Cayley

Table 4.2: **Architecture** for width parameter w and c classes. © 2024 IEEE.

Layer name	Output size
ZeroChannelPadding	$32 \times 32 \times w$
1×1 Convolution	$32 \times 32 \times w$
MaxMin	$32 \times 32 \times w$
ConvBlock($k = 3$)	$16 \times 16 \times 2w$
ConvBlock($k = 3$)	$8 \times 8 \times 4w$
ConvBlock($k = 3$)	$4 \times 4 \times 8w$
ConvBlock($k = 3$)	$2 \times 2 \times 16w$
ConvBlock($k = 1$)	$1 \times 1 \times 32w$
Flatten	$32w$
Linear	$32w$
FirstChannels(c)	c

convolutions requires storing $M + \frac{1}{2}s^2c^2$ floats, and applying a LOT convolutions requires memory to store $M + s^2c^2$ floats. In order to evaluate a forward pass on the model, we usually need to store all (transformed) model parameters, and in addition we require memory to store the largest activation during the forward pass.

4.3 Experimental Setup

This section presents an experimental study aimed at comparing the performance of the considered layers with respect to different metrics. Before presenting the results, we first summarize the setup used in our experiments. To have a fair and meaningful comparison among the various models, all the proposed layers have been evaluated using the same architecture, loss function, and optimizer. Since, according to the data reported in Table 4.1, different layers may have different throughput, to ensure a fair comparison with respect to the tested metrics, we limited the total training time instead of fixing the number of training epochs. Results are reported for training times of 2h, 10h, and 24h on one A100 GPU.

4.3.1 Architecture

Our architecture is a standard convolutional network that doubles the number of channels whenever the resolution is reduced. We show the architecture used for our experiments in Tables 4.2 and 4.3. Note that we exclusively use convolutions with the same input and output size as an attempt to make the model less dependent on the initialization, by using layers described in Section 2.3.

For each method, we tested architectures of different sizes. We refer to them as XS, S, M and L. See Table 4.4 for the number of parameters for each size, as well as the *width parameter* w that ensures our architecture has the correct number of parameters.

4.3.2 Training Time

One of our main goals is to evaluate what is the best model to use given a certain time budget. In order to do this, as a first experiment, we measure the time per epoch on an A100 GPU

Table 4.3: **ConvBlock**(k) with input size $s \times s \times t$. © 2024 IEEE.

	Layer name	Output size
$5 \times \left\{ \begin{array}{l} \\ \\ \\ \end{array} \right.$	$k \times k$ Convolution	$s \times s \times t$
	MaxMin	$s \times s \times t$
	First Channels	$s \times s \times t/2$
	Pixel Unshuffle	$s/2 \times s/2 \times 2t$

Table 4.4: Number of parameters for different model sizes, as well as the matching *width parameter* w . © 2024 IEEE.

Size	Parameters (millions)	w
XS	$1 < p < 2$	16
S	$4 < p < 8$	32
M	$16 < p < 32$	64
L	$64 < p < 128$	128

with 80GB memory for different methods and different model sizes. We use this to estimate the number of epochs we can do in our chosen time budget of either 2h, 10h or 24h, and use that many epochs to train our models.

4.3.3 Datasets and augmentation

We evaluate on four different datasets: CIFAR-10, CIFAR-100 [Kri09], Tiny ImageNet [LY15] and Imagenette [How19]. Tiny ImageNet consist of 200 ImageNet [DDS⁺09] classes, with images scaled to have size 64×64 . As a higher resolution dataset we use Imagenette. It is a subset of 10 classes from Imagenet, chosen to be easily distinguishable. The version we consider contains images with their shorter side resized to 320 pixels.

As preprocessing, on all datasets we subtract the channel means from each image. On Imagenette we also crop the images to size 256×256 . We use center cropping for inference and random cropping during training.

On CIFAR-10, CIFAR-100, as data augmentation at training time we apply random crops (4 pixels) and random flipping. On Tiny ImageNet we use *RandAugment* [CZSL20] with 2 transformations of magnitude 9 out of 31. On Imagenette we use random crops as well as *RandAugment* (magnitude 9 again).

In order to adjust to the larger input size of Tiny ImageNet (64×64) and Imagenette (256×256), we add additional *ConvBlocks* to our model. We also decrease the width parameter w in Table 4.2 to keep the amount of parameters in the range described in Table 4.4.

We use a batch size of 64 of Imagenette and 256 for all other experiments.

4.3.4 Loss, optimizer and hyperparameters

We use the loss function proposed in Chapter 3, with the same temperature of $\frac{1}{4}$. We aim to maximize the robust accuracy for $\epsilon = \frac{36}{255}$ in this work, in order to achieve this we set the

margin to $2\sqrt{2}\epsilon$ where the $\sqrt{2}$ factor comes from the L_2 norm, and we added an additional factor 2 to help with generalization.

As optimizer we use SGD with a momentum of 0.9 for all experiments. We also used a learning rate schedule. We choose to use *OneCycleLR*, as described by [ST19], with default values as in *PyTorch*.

We aim to obtain a good (peak) learning rate and weight decay amount for each setting (model size, method and dataset). We used a validation set composed of 10% of the images of the original training data. For each setting we did a hyperparameter search by training multiple runs for $2h$ each. We did 16 runs with learning-rate of the form 10^x , where x is sampled uniformly in the interval $[-4, -1]$, and with weight-decay of the form 10^x , where x is sampled uniformly in the interval $[-5.5, -3.5]$. Finally, we selected the learning rate and weight decay corresponding to the run with the highest validation certified robust accuracy for radius $36/255$. We use these hyperparameters found also for the experiments with longer training time.

4.3.5 Metrics

All the considered models were evaluated based on three main metrics: the *throughput*, the required memory, and the certified robust accuracy.

Throughput and epoch time The *throughput* of a model is the average number of examples that the model can process per second. It determines how many epochs are processed in a given time frame. The evaluation of the throughput was performed on an *NVIDIA A100 80GB PCIe GPU* based on the average time of 100 mini-batches. In order to estimate the inference throughput, we first evaluate and cache all parameter transformations.

Memory required Layers that require less memory allow for larger batch size, and the memory requirements also determine the type of hardware we can train a model on. For each model, we measured and reported the maximal GPU memory occupied by tensors using the function `torch.cuda.max_memory_allocated()` provided by the *PyTorch* framework. This is not exactly equal to the overall GPU memory requirement but gives a fairly good approximation of it. Note that the model memory measured in this way also includes additional memory required by the optimizer (e.g. to store the momentum term) as well as by the activation layers in the forward pass. However, this additional memory should be at most of order $\mathcal{O}(M + c^2k^2)$. As for the throughput, we evaluated and cached all calculations independent of the input at inference time.

Certified robust accuracy We also evaluate the performance of the trained networks. For that we evaluate the accuracy as well as the certified robust accuracy for perturbations of size $36/255$, as defined in Equation (2.11).

4.4 Experimental Results

This section presents the results of the comparison performed by applying the methodology discussed in Section 4.3. The results related to the different metrics are discussed in dedicated subsections and the key takeaways are summarized in the radar-plot illustrated in Figure 4.4.

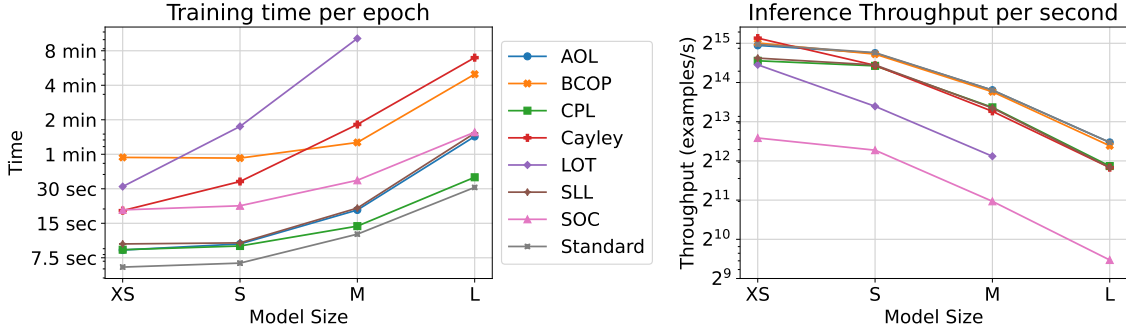


Figure 4.1: **Training time** per epoch (on CIFAR-10) (left) and **inference throughput** (right) for different methods and different model sizes. All parameter transformations have been evaluated and cached before measuring inference throughput. © 2024 IEEE.

4.4.1 Training and inference times

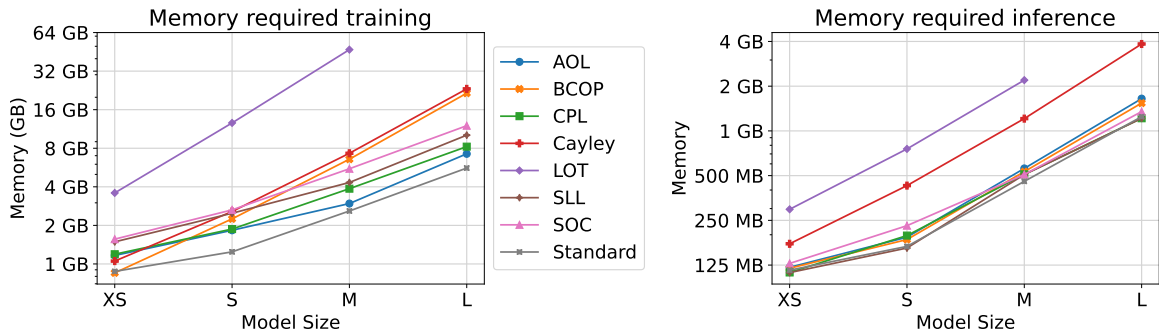
Figure 4.1 plots the training time per epoch of the different models as a function of their size, as well as the corresponding inference throughput for the various sizes. As described in Table 4.2, the model base width, referred to as w , is doubled from one model size to the next. We expect the training and inference time of a model to scale with w similarly to how individual layers scale with their number of channels, c (in Table 4.1). This is because the width of each of the 5 blocks of our architecture is a constant multiple of the base width, w .

Training time: First note that for the standard convolution, when we increase the size of a model (and therefore double the width parameter), the time also (at worst) doubles. However, the computational complexity of a standard convolutions scales like c^2 , implying a four-fold increase in MACs. This suggests that loading weights into GPU memory or the overhead from other operations (activations, parameter updates, etc.) are important factors determining the training time. This also explains why CPL (doing two convolutions, with identical kernel parameters) is only slightly slower than a standard convolution, and SOC (doing 5 convolutions) is only about 3 times slower than the standard convolution. The methods AOL and SLL also require times comparable to a standard convolution for small models, but the c^3 term in the computation of the rescaling makes them slower for larger models. Finally, Cayley, LOT, and BCOP methods take much longer training times per epoch. For Cayley and LOT this behavior was expected, as they have a large $\mathcal{O}(s^2c^3)$ term in their computational complexity, see Table 4.1.

Inference time During inference, we can compute weight transformations a single time in the beginning, and afterwards use cached values, therefore some methods (AOL, BCOP) have no overhead at all compared to a standard convolution. As expected, other methods (CPL, SLL, and SOC) that apply additional convolutions to the input suffer from a corresponding overhead. Finally, Cayley and LOT have a slightly different throughput due to their FFT-based convolution. Among them, Cayley is about twice as fast because it involves a real-valued FFT rather than a complex-valued one. In Figure 4.1, Cayley and CPL have the same inference time, even though CPL uses twice the number of convolutions. We believe this is due to the fact that the conventional FFT-based convolution is quite efficient for large kernel sizes, but for smaller ones PyTorch implements a faster algorithm, *Winograd*, [LG16].

Table 4.5: Number of epochs different methods can complete in 2h on an A100 GPU. © 2024 IEEE.

	CIFAR				TinyImageNet				Imagenette		
	XS	S	M	L	XS	S	M	L	S	M	L
AOL	837	763	367	83	223	213	123	34	136	108	54
BCOP	127	125	94	24	50	50	39	11	63	51	16
CPL	836	797	522	194	240	194	148	63	136	108	62
Cayley	356	214	70	17	138	86	30	8	81	47	-
LOT	222	68	11	-	83	29	5	-	33	-	-
SLL	735	703	353	79	242	194	118	32	134	110	59
SOC	371	336	201	77	122	87	63	27	98	68	28

Figure 4.2: **Memory required** during training (left) and inference (right) on CIFAR-10. © 2024 IEEE.

Epoch budgets: Based on the results of these experiments, we also determine the number of epochs that can be completed in a certain time window. We show them in Table 4.5. For CIFAR-10 and CIFAR-100 the architectures used are identical, and the datasets have the same number of images, so the budgets are the same.

4.4.2 Training memory requirements

The training and inference memory requirements of the various models (measured as described in Section 4.3.5) are reported in Figure 4.2 as a function of the model size. The results of the theoretical analysis reported in Table 4.1 suggest that the training memory requirements always have a term linear in the number of channels c (usually the activations from the forward pass), as well as a term quadratic in c (usually the weights and all transformations applied to the weights during the forward pass). This behavior can also be observed from Figure 4.2. For some of the models, the memory required approximately doubles from one model size to the next one, just like the width of the layers. This means that the linear term dominates (for those sizes), which makes those models relatively cheap to scale up. For the BCOP, LOT, and Cayley methods, the larger coefficients of the c^2 term cause this term to dominate. This makes it much harder to scale those methods to more parameters. Method LOT requires huge amounts of memory, in particular LOT-L is too large to fit in 80GB GPU memory.

At test time, the memory requirements are much lower, because the intermediate activation values do not need to be stored, as there is no backward pass. Therefore, for inference, most methods require a very similar amount of memory as a standard convolution. The Cayley and

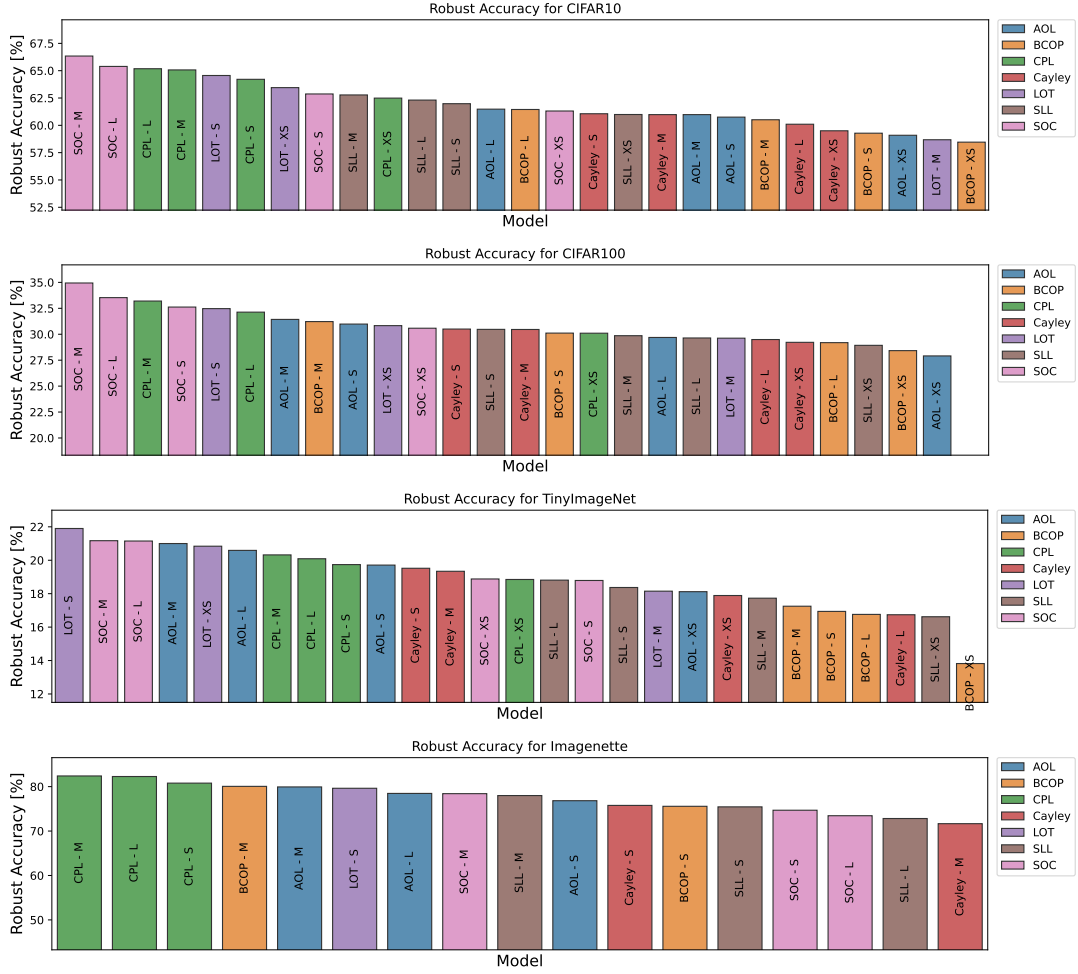


Figure 4.3: **Certified robust accuracy** ($\epsilon = 36/255$) in decreasing order. Note that the axes do not start at 0. © 2024 IEEE.

LOT methods require more memory since they perform the calculation in the Fourier space, creating an intermediate representation of the weight matrices of size $\mathcal{O}(s^2c^2)$.

4.4.3 Certified robust accuracy

We present the accuracy and certified robust accuracy (radius $36/255$) for the different methods, model sizes, and datasets in Table 4.6. For those results we trained all models for 24 hours. The results are also visualized in Figure 4.3. As a first observation, it is often not the largest model that performs best. Instead, it is usually the model of size M that reaches the best certified robust accuracy. This is because we restricted training based on time, and not by the number of epochs. This means that bigger models are trained for fewer epochs, explaining the drop in performance.

In our experiments SOC performs best, especially on lower resolution datasets, reaching the highest certified robust accuracy twice. CPL models consistently rank in top positions, and are the clear winner on Imagenette. LOT performed well, in particular on Tiny ImageNet where it performs the best. AOL did not reach high accuracy on CIFAR-10, but reached more competitive results on Tiny ImageNet. An opposite effect can be observed for SLL, which performance seems to decrease for datasets with more classes. BCOP rarely reaches the top positions, and Cayley is consistently outperformed by the other methods.

Table 4.6: **Certified robust accuracy** for radius $\epsilon = 36/255$. Training is performed for 24 hours. © 2024 IEEE.

Methods	Accuracy [%]				Robust Accuracy [%]			
	XS	S	M	L	XS	S	M	L
CIFAR-10								
AOL	71.7	73.6	73.4	73.7	59.1	60.8	61.0	61.5
BCOP	71.7	73.1	74.0	74.6	58.5	59.3	60.5	61.5
CPL	74.9	76.1	76.6	76.8	62.5	64.2	65.1	65.2
Cayley	73.1	74.2	74.4	73.6	59.5	61.1	61.0	60.1
LOT	75.5	76.6	72.0	-	63.4	64.6	58.7	-
SLL	73.7	74.2	75.3	74.3	61.0	62.0	62.8	62.3
SOC	74.1	75.0	76.9	76.9	61.3	62.9	66.3	65.4
CIFAR-100								
AOL	40.3	43.4	44.3	41.9	27.9	31.0	31.4	29.7
BCOP	41.4	42.8	43.7	42.2	28.4	30.1	31.2	29.2
CPL	42.3	-	45.2	44.3	30.1	-	33.2	32.1
Cayley	42.3	43.9	43.5	42.9	29.2	30.5	30.5	29.5
LOT	43.5	45.2	42.8	-	30.8	32.5	29.6	-
SLL	41.4	42.8	42.4	42.1	28.9	30.5	29.9	29.6
SOC	43.1	45.2	47.3	46.2	30.6	32.6	34.9	33.5
Tiny ImageNet								
AOL	26.6	29.3	30.3	30.0	18.1	19.7	21.0	20.6
BCOP	22.4	26.2	27.6	27.0	13.8	16.9	17.2	16.8
CPL	28.3	29.3	29.8	30.3	18.9	19.7	20.3	20.1
Cayley	27.8	29.6	30.1	27.2	17.9	19.5	19.3	16.7
LOT	30.7	32.5	28.8	-	20.8	21.9	18.1	-
SLL	25.1	27.0	26.5	27.9	16.6	18.4	17.7	18.8
SOC	28.9	28.8	32.1	32.1	18.9	18.8	21.2	21.1
Imagenette								
AOL		80.8	83.7	82.8		76.8	79.9	78.5
BCOP		81.2	84.5	9.8		75.6	80.1	9.8
CPL		85.5	86.5	86.4		80.8	82.4	82.3
Cayley		81.2	77.9	-		75.8	71.7	-
LOT		84.4	-	-		79.6	-	-
SLL		80.8	83.4	79.3		75.4	78.0	72.8
SOC		80.6	83.6	79.0		74.7	78.4	73.5

We repeated the analysis for the (non-robust) accuracy. See Figure A.1 for the results. They are similar, with one difference that Cayley performs slightly better for that metric.

Interpretation of the results

All layer types considered in this paper have comparable expressive power. Therefore, all networks could reach comparable accuracy if trained for enough time. However, training Lipschitz networks to convergence is typically not possible, because the training loss decreases very slowly (often logarithmic in the number of epochs). This is the reason why we adopt the setting of time budgets. Our results confirm empirically what was suspected in [XLL22]: Layers that naturally include a skip connections (CPL, SLL, SOC) generally perform better than layers that do not have this ability. Furthermore, we noticed that layers that are initialized close to the identity map (AOL, LOT) perform better than layers that do neither (BCOP, Cayley). Presumably this is because MaxMin activations reduce the variance in the forward pass when alternated with non-identity layers.

Note that the three methods CPL, SLL and SOC all rely on an upper bound of the Lipschitz constant in their calculation. Furthermore, all three methods have the property that if the upper bound is loose, the layer approximates the identity map. Since the upper bounds are usually only tight for a small subspace of parameter matrices, those methods should allow larger learning rates, as the individual layers will end up approximating the identity map in the worst case. This seems like a very useful property for a network to possess, and we expect that this is one important reason behind the good performance of those methods.

Our results also allow ruling out some other possible explanation: One might suspect that AOL, CPL, and SLL suffer from vanishing gradients, as they do not enforce orthogonality but only restrict the Lipschitz constant. Our experiments, however, do not show evidence of this. Also, one might suspect that slower methods perform worse, because they allow fewer epochs for a given time budget. Again, our experiments do not support this. Two relative slow methods (SOC, LOT) are among the best ones.

The results on Imagenette give us an indication how well methods scale to higher image resolution. In our experiments, CPL clearly performs best. BCOP, AOL and LOT also perform well. Note that the calculations performed by those methods are independent of the image resolution; the methods rescale or construct the kernel directly. So in relation to applying the convolution, parameter transformations become much cheaper to calculate. This is not the case for SOC, and whilst it was a top performer on the datasets with low image resolution, it is much worse compared to other methods on Imagenette.

4.5 Potential Future Directions

There are plenty of interesting directions to extend this work.

One very important aspect is the influence of the batch size. In 1-Lipschitz networks, the parameter transformation is only done once per batch, so the batch size has a large influence on the performance a model can reach in a certain training time. So, in particular for computationally expensive methods, we expect that increasing the batch size could increase performance. For this then the required memory becomes an important factor.

Also, as mentioned above, we do believe that initialization is a very important factor for 1-Lipschitz networks. We are curious to see the influence being explored in future work.

Apart from that, we also leave adapting properties like model depth, choice of optimizer and data augmentation to future work. Finally, different methods seem to have different strengths,

it might be useful to look into combining them, for example by using different methods for early (high resolution) layers and later layers (with more channels).

4.6 Summary and Guidelines

This work presented a comparative study of state-of-the-art 1-Lipschitz layers under the lens of different metrics: time and memory requirements, accuracy, and certified robust accuracy, all evaluated at training and inference time. We also presented a theoretical comparison of the methods in terms of time and memory complexity.

Taking all metrics into account (summarized in Figure 4.4), the results are in favor of CPL, due to its highest performance and lower consumption of computational resources. When large computational resources are available the SOC layer could be used, due to its slightly better performance. Finally, applications in which the inference time is crucial may take advantage of AOL or BCOP, which do not introduce additional runtime overhead during inference compared to a standard convolution.

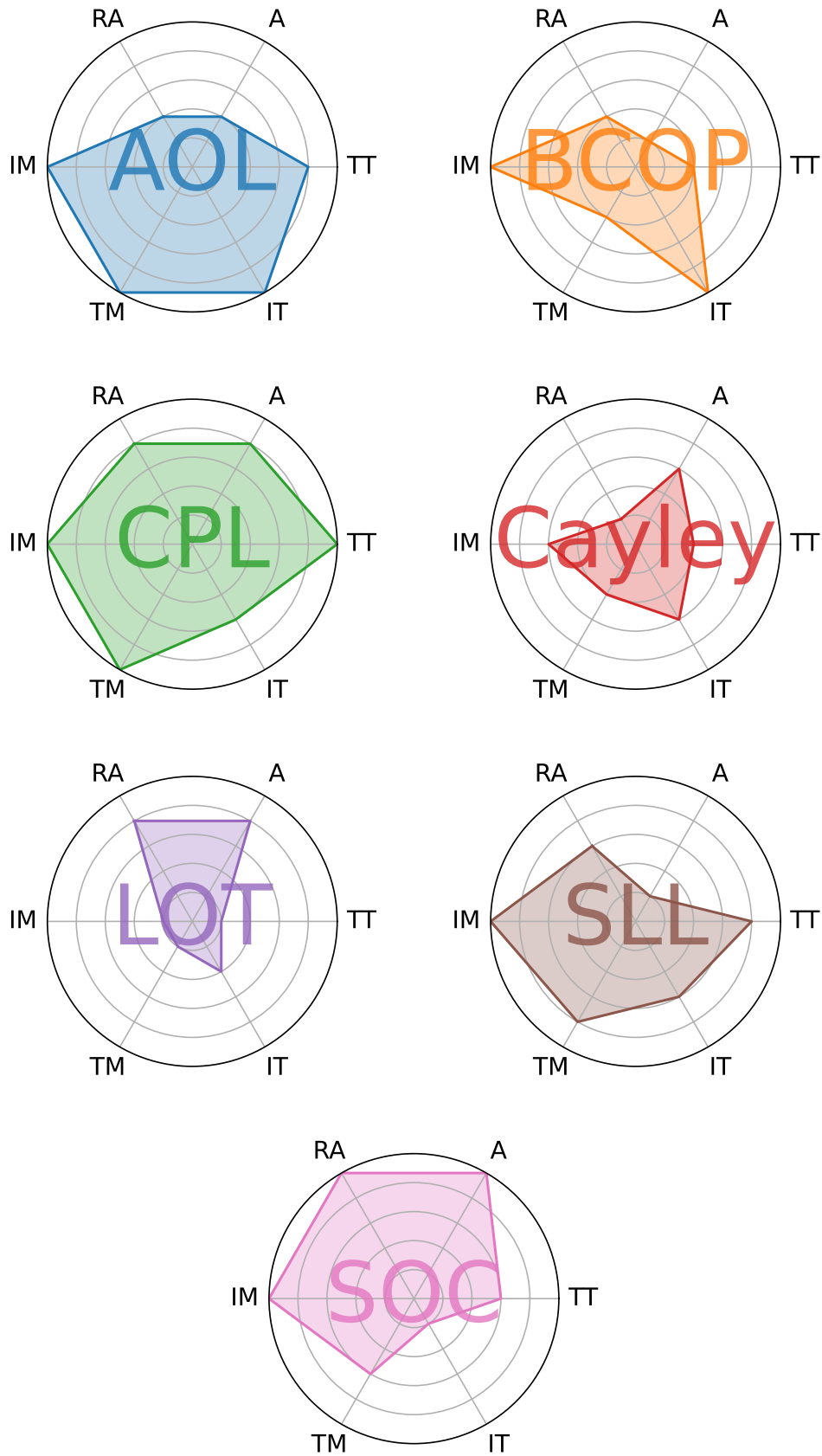


Figure 4.4: **Performance summary** of certified robust accuracy (RA), accuracy (A), training time (TT), inference time (IT), training memory (TM) and inference memory (IM). Larger radius is better. © 2024 IEEE.

1-Lipschitz Networks are more expressive with \mathcal{N} -activations

Unfortunately, despite many years of research, robust networks still achieve results far worse than what one might hope for. Even on fairly simple datasets and for fairly small perturbations the robust accuracy is much worse than what we believe is possible. Furthermore, in the previous chapter we found that even architectures and training techniques that differ strongly in terms of their memory and computational demands ultimately achieve quite similar robust accuracy values. This raises the question if the task of robust classification is just inherently very difficult, or if there is a theoretical shortcoming in our current approaches that prevents them from reaching better performance.

In this chapter, we will show that there actually is a shortcoming with current architectures. We prove that the activation function commonly used in 1-Lipschitz networks, MaxMin [ALG19], does not allow them to express all functions that we might expect such networks to be able to express. In particular, we show (theoretically and experimentally) that 1-Lipschitz networks with MaxMin activation functions can not even express a very simple 1-dimensional (1-Lipschitz) function.

To overcome the shortcoming, we introduce an activation function that (provably) can express any reasonable 1-dimensional function. We call it the \mathcal{N} -activation. Apart from providing a theoretical guarantee, we also show empirical results. In particular, we show that the \mathcal{N} -activation is a competitive replacement for the MaxMin activation in the task of certified robust classification.

Note that in this chapter in the theoretical analysis we consider regression problems, not classification problems. However, in particular for 1-Lipschitz classifiers, there is a strong connection between regression problems and classification problems: In order to classify inputs with the maximal possible margin, we require the scores produced by the 1-Lipschitz networks to follow a particular 1-Lipschitz function of the input. Therefore, we essentially need to solve a regression problem.

This chapter appears in full in [PL23], Bernd Prach and Christoph H. Lampert, 1-Lipschitz neural networks are more expressive with \mathcal{N} -activations, *arXiv preprint arXiv:2311.06103*, 2023.

5.1 Background & Definitions

In this chapter, we are interested in the predictive power of 1-Lipschitz networks. Clearly, there are principled restrictions on the kinds of functions that these can express. In particular, if the activation function is piece-wise linear, the whole network will also only be able to express piece-wise linear functions. Also, by construction, the network can of course only express 1-Lipschitz functions. Within those two restrictions, though, the networks should ideally be able to express any function.

Formally, we call a function 1-Lipschitz, piece-wise linear (1-LPL), if it is 1-Lipschitz and piece-wise linear with finitely many segments. We say an activation function can *express* a function f if we can write f using only 1-Lipschitz linear operations and the activation function. We call an activation function that can express all 1-LPL functions *1-LPL-universal* (short: universal).

5.2 Related Work

Whilst a lot of work has focused on parameterizing 1-Lipschitz linear layers so far, much less work has focused on the problems and shortcomings of 1-Lipschitz networks in general, as well as shortcomings with the architectural decisions often made in current 1-Lipschitz networks.

For general ReLU networks it has been shown that we require about $\log \log k$ layers to represent the function that computes maximum of k numbers [GHL25]. So the depth required for universality does increase with the dimension of the input when there is no restriction on the Lipschitz constant. In this chapter we show that for 1-Lipschitz networks, already with 1-dimensional input there are functions that cannot be expressed, independent of the depth of the network.

Some works have looked into general limitations of robust networks. For example, [BS21, BKM23] showed that in order to perfectly interpolate noise in the training data in a robust way, one does require many times more parameters than data points. However, we are not interested in achieving zero training loss, but we care about the generalization performance, so their theory is not applicable in our scenario. Similarly, [Lei23] showed that there exist data distributions for which a simple robust classifier exists, however, any 1-Lipschitz score-based classifier requires to fit a much more complicated function with many more non-linearities.

There is some work (such as [CHC19]) that provides results for different norms (like L_∞), however, in this thesis, we are only interested in 1-Lipschitz networks with respect to the L_2 norm.

As shown by [Eck20] any 1-Lipschitz function (with one output) can be approximated (arbitrarily well) by a neural network in a way that the approximation is also 1-Lipschitz. However, we are interested in 1-Lipschitz neural network with the commonly used design where every single layer is 1-Lipschitz, and all intermediate representations are 1-Lipschitz transformations of the inputs as well.

For the L_2 norm and for networks consisting purely of 1-Lipschitz layers, early on, [ALG19] observed that the otherwise popular ReLU activation function is not a good choice in this context. They showed that 1-Lipschitz ReLU networks are not able to fit even some simple functions, including the absolute value function. As an improvement, the authors suggested

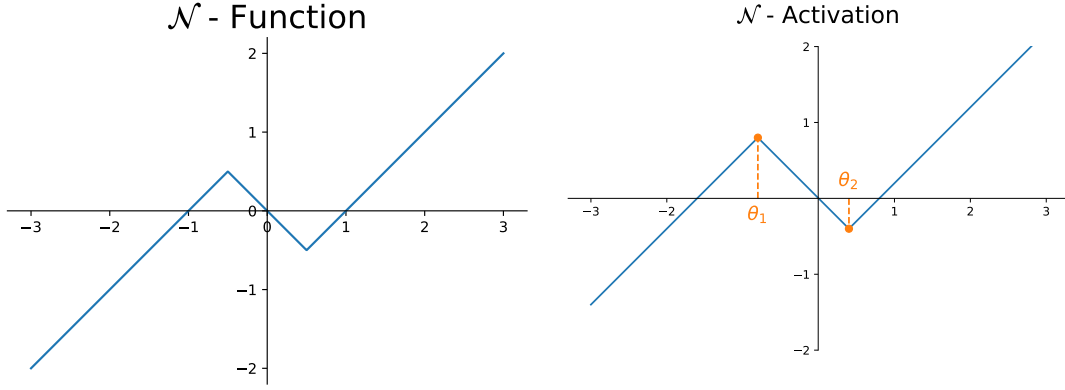


Figure 5.1: A plot of the \mathcal{N} -function (left) and \mathcal{N} -activation with parameters θ_1 and θ_2 (right).

GroupSort with its special case of *MaxMin*, given as

$$\begin{aligned} \text{MaxMin}(\mathbf{x})_{2i+1} &:= \max(x_{2i+1}, x_{2i+2}) \\ \text{MaxMin}(\mathbf{x})_{2i+2} &:= \min(x_{2i+1}, x_{2i+2}). \end{aligned} \quad (5.1)$$

and proved a result of universal approximation. However, for this result they constraint the operator norm of the weight matrices using the L_∞ norm instead of the L_2 norm.

Other activation functions such as *Householder activations* [SSF21] have been proposed to improve results with 1-Lipschitz networks, however, e.g. [DGB⁺22] showed that the corresponding networks do represent the same set of functions as *MaxMin* networks.

Finally, as a very promising direction, [AGCU20] used (parameterized) linear splines as an activation function in 1-Lipschitz networks. Building on this work, [NGBU22] showed that 1-Lipschitz linear splines with 3 linear regions are universal as an activation function in 1 dimension. However, training spline networks with a general parameterization seems difficult in practice, and [DGB⁺22] use three different learning rates as well as auxiliary loss functions in order to do that. As one of our contributions, we will show that we can further restrict the set of splines that can be learned to have a specific structure, without restricting the class of functions the network can express.

5.3 Theoretical Results

In this section, we present our main results, namely that even for functions with one-dimensional input and output, commonly used activation functions are not universal. We also introduce a new activation function, which we call \mathcal{N} -activation, and show that this activation function is universal.

5.3.1 Limitations of existing activation functions

Our first result is that a whole class of activation functions, namely 1-LPL functions with 2 segments are not universal. In order to proof this we will first prove that the absolute value activation ($x \rightarrow |x|$) is not universal:

Lemma 1. *The absolute value activation is not universal.*

Proof. In order to prove Lemma 1, it is enough to show that there exists a 1-dimensional, 1-LPL function that can not be expressed by any 1-Lipschitz absolute value network. There is in fact a simple function that can not be expressed, we call it the \mathcal{N} -function. It is given as:

$$\mathcal{N}(x) = \begin{cases} x + 1 & \dots & x \leq -\frac{1}{2} \\ -x & \dots & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ x - 1 & \dots & \frac{1}{2} \leq x. \end{cases} \quad (5.2)$$

It is visualized in Figure 5.1. We will show that absolute value networks can not express this function.

Suppose there is an absolute value network f that can express the \mathcal{N} -function. First, note that the \mathcal{N} -function is not linear, so we need to apply an activation function at some point. Consider the first time the absolute value-activation is applied non-trivially. We will consider the input and the output of this first activation, and denote them by $v(x) \in \mathbb{R}^k$ and $w(x) \in \mathbb{R}^k$ (as a function of input $x \in \mathbb{R}$). We obtain $w(x)$ from $v(x)$ by applying the absolute value-activation to at least some of the elements. Furthermore, we can obtain the output of the network, $f(x)$, from $w(x)$ by applying the remaining layers of the network. The only assumption we make about those layer is that they are all 1-Lipschitz.

By our assumption, we know that v is a linear function of the input, write

$$v(x) = \mathbf{a}x + \mathbf{b}, \quad (5.3)$$

for some $\mathbf{a} \in \mathbb{R}^k$ and $\mathbf{b} \in \mathbb{R}^k$. Note that $\|\mathbf{a}\|_2 \leq 1$ by the 1-Lipschitz property.

We assumed that the activation was applied non-trivially, so there is an index t such that $w(x)_t = |v(x)_t|$, and $a_t \neq 0$, and therefore we can define $x_0 \in \mathbb{R}$ so that $v(x_0)_t = 0$. Then we have that for any $\delta \in \mathbb{R}$:

$$w\left(x_0 + \frac{\delta}{2}\right)_t = \left|v\left(x_0 + \frac{\delta}{2}\right)_t\right| = \left|a_t \frac{\delta}{2}\right| = \left|v\left(x_0 - \frac{\delta}{2}\right)_t\right| = w\left(x_0 - \frac{\delta}{2}\right)_t \quad (5.4)$$

and we can use the fact that the transformation $v(x)_i \mapsto w(x)_i$ is 1-Lipschitz for all i to get that

$$\left\|w\left(x_0 + \frac{\delta}{2}\right) - w\left(x_0 - \frac{\delta}{2}\right)\right\|_2^2 \quad (5.5)$$

$$= \sum_{i=1}^k \left(w\left(x_0 + \frac{\delta}{2}\right)_i - w\left(x_0 - \frac{\delta}{2}\right)_i\right)^2 \quad (5.6)$$

$$= \sum_{i \neq t} \left(w\left(x_0 + \frac{\delta}{2}\right)_i - w\left(x_0 - \frac{\delta}{2}\right)_i\right)^2 \quad (5.7)$$

$$\leq \sum_{i \neq t} \left(v\left(x_0 + \frac{\delta}{2}\right)_i - v\left(x_0 - \frac{\delta}{2}\right)_i\right)^2 \quad (5.8)$$

$$= \left\|v\left(x_0 + \frac{\delta}{2}\right) - v\left(x_0 - \frac{\delta}{2}\right)\right\|_2^2 - |a_t \delta|^2. \quad (5.9)$$

Then, for f the output of the full network we can use the Lipschitz property of the layers in order to get the following:

$$\left(f\left(x_0 + \frac{\delta}{2}\right) - f\left(x_0 - \frac{\delta}{2}\right) \right)^2 \quad (5.10)$$

$$\leq \left\| w\left(x_0 + \frac{\delta}{2}\right) - w\left(x_0 - \frac{\delta}{2}\right) \right\|_2^2 \quad (5.11)$$

$$\leq \left\| v\left(x_0 + \frac{\delta}{2}\right) - v\left(x_0 - \frac{\delta}{2}\right) \right\|_2^2 - |a_t \delta|^2 \quad (5.12)$$

$$\leq \left\| \left(x_0 + \frac{\delta}{2}\right) - \left(x_0 - \frac{\delta}{2}\right) \right\|_2^2 - |a_t \delta|^2 \quad (5.13)$$

$$= (1 - a_t^2) \delta^2. \quad (5.14)$$

By our assumption, $a_t \neq 0$, so $(1 - a_t^2) < 1$. However, the \mathcal{N} -function has the property that

$$\lim_{\delta \rightarrow \infty} \left(\frac{\mathcal{N}\left(x_0 + \frac{\delta}{2}\right) - \mathcal{N}\left(x_0 - \frac{\delta}{2}\right)}{\delta} \right)^2 = 1, \quad (5.15)$$

and this contradicts that f is equal to the \mathcal{N} -function. \square

This result will allow us to prove our first theorem:

Theorem 3. *No 2-piece 1-LPL activation is universal.*

Proof. Any 2-piece piecewise linear activation function can be written as a linear (and 1-Lipschitz) combination of an absolute value-activation and the identity map (and some bias terms). Therefore, Theorem 3 follows directly from Lemma 1. \square

This theorem shows that activations such as ReLU, leaky ReLU and absolute value are not universal. Furthermore, we will also show that the commonly used MaxMin-activation is not universal.

Corollary 2. *MaxMin-activations are not universal.*

Proof. It has been shown before that MaxMin networks can express exactly the same class of functions as absolute value networks on bounded inputs [ALG19]. Similarly, on unbounded input, any absolute value network can be expressed as a network consisting of absolute value activations as well as identity connections. One way of doing this is by expressing the MaxMin-activation as

$$\text{MaxMin}\left(\begin{pmatrix} x \\ x \end{pmatrix}\right) = M\sigma\left(M\begin{pmatrix} x \\ y \end{pmatrix}\right), \quad (5.16)$$

where

$$M = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{and} \quad \sigma\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = \begin{pmatrix} x \\ |y| \end{pmatrix}. \quad (5.17)$$

Therefore, any function that can be expressed by a MaxMin network can also be expressed by a network with absolute value activations (and identity connections), and Theorem 3 implies that also MaxMin is not universal. \square

Finally, we can also express activations such as Householder activations [SSF21] as a concatenation of rotations and MaxMin-activations (see e.g. [DGB⁺22]). Therefore, these are not universal either.

5.3.2 The \mathcal{N} -activation is universal in 1D.

In order to be able to express the \mathcal{N} -function, we will define an activation that is more expressive than MaxMin. Inspired by the \mathcal{N} -function, we define the (parameterized) \mathcal{N} -activation:

$$\mathcal{N}(x; \theta_1, \theta_2) = \begin{cases} x - 2\theta_{\min} & \dots & x \leq \theta_{\min} \\ -x & \dots & \theta_{\min} \leq x \leq \theta_{\max} \\ x - 2\theta_{\max} & \dots & \theta_{\max} \leq x, \end{cases} \quad (5.18)$$

where $\theta_{\max} = \max(\theta_1, \theta_2)$ and $\theta_{\min} = \min(\theta_1, \theta_2)$. For a visualization of the \mathcal{N} -activation see Figure 5.1.

In our arguments, we will allow θ_1 to be $-\infty$. In this case, the \mathcal{N} -activation is equal to the absolute value activation when $\theta_2 = 0$, so this is equivalent to allowing a network to use a combination of \mathcal{N} -activations (with θ_1 and θ_2 finite) and absolute value activations. Note that for bounded inputs, setting θ_1 to some negative value with high magnitude will have the same effect as (theoretically) setting it to $-\infty$.

A network using the \mathcal{N} -activation is trivially able to fit the \mathcal{N} -function, but furthermore, the \mathcal{N} -activation is universal: It can express any 1-dimensional, 1-LPL function:

Theorem 4. *\mathcal{N} -activations are universal in 1D. Specifically, any 1-LPL, 1-dimensional function with k non-linearities can be represented by a network of width 2 consisting of $(k + 5)$ linear layers and $(\frac{3}{2}k + 5)$ \mathcal{N} -activations.*

In the proof below we first show that we can express *increasing* 1-LPL functions using k linear layers and $(k - 1)$ \mathcal{N} -activations. Then we show that each function with $2l$ local extreme points can be expressed by applying the \mathcal{N} -activation to some function with $2l - 2$ local extreme points, and that allows us to use induction to show that we can express functions with any number of extreme points. Finally, we need 3 absolute value activations to adjust the slope of a function before the first and after the last non-linearity.

In order to prove Theorem 4, we will first state and proof some useful lemmas. The first one is about expressing increasing functions:

Lemma 2. *Any increasing, 1-LPL, 1-dimensional function f with gradient 1 before the first and after the last non-linearity and with k non-linearities can be expressed with k 1-Lipschitz, linear layers and $(k - 1)$ \mathcal{N} -activations.*

Proof. Suppose f has non-linearities at t_1, \dots, t_k . Then for $i = 1, \dots, k$ we will construct a continuous function $g_i : \mathbb{R} \rightarrow \mathbb{R}$ that has i layers such that

$$\begin{aligned} g'_i(x) &= f'(x) & \dots & x < t_i \\ g_i(x) &= x & \dots & x \geq t_i. \end{aligned} \quad (5.19)$$

Then up to a bias term $g_k = f$ and we are done.

In order to construct such functions g_i , we will define s_i to be the slope of f between t_i and t_{i+1} , and furthermore define α_i and β_i such that $\alpha_i^2 - \beta_i^2 = s_i$ and $\alpha_i^2 + \beta_i^2 = 1$. We also define $\sigma_i : \mathbb{R} \rightarrow \mathbb{R}$ as the (element-wise) application of the \mathcal{N} -activation, with certain parameters:

$$\sigma_i \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ \mathcal{N}(x_2; \beta_i t_i, \beta_i t_{i+1}) \end{pmatrix}. \quad (5.20)$$

Note that the first element is given as the identity, which is equal to the \mathcal{N} -activation with both parameters equal to 0.

Having defined α_i, β_i as well as σ_i , we can define g_i . Set $g_1(x) = x$ and

$$g_{i+1}(x) = \begin{pmatrix} \alpha_i & \beta_i \end{pmatrix} \sigma_i \left(\begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} g_i(x) \right). \quad (5.21)$$

Note that in this definition we use 2 matrix multiplications, which seems to be contradicting the fact that g_i has i layers. However, we can merge two adjacent matrix multiplications into one, allowing us to in fact express g_i using i layers.

Furthermore, since $\alpha_i^2 + \beta_i^2 = 1$ holds, both the matrices have a spectral norm of at most one. To see this note that the matrix

$$\begin{pmatrix} \alpha_i & \beta_i \\ \beta_i & -\alpha_i \end{pmatrix} \quad (5.22)$$

is orthogonal, and the matrices in Equation (5.21) are either a row or a column of the matrix in Equation (5.22).

From the definition of g_{i+1} we get that

$$g_{i+1}(x) = \alpha_i^2 g_i(x) + \beta_i \mathcal{N}(\beta_i g(x); \beta_i t_i, \beta_i t_{i+1}) \quad (5.23)$$

$$= \alpha_i^2 g_i(x) + \beta_i^2 \begin{cases} g_i(x) - 2g_i(t_i) & \dots & g(x) \leq t_i \\ -g_i(x) & \dots & t_i \leq g(x) \leq t_{i+1} \\ g_i(x) - 2g_i(t_{i+1}) & \dots & t_{i+1} \leq g(x), \end{cases} \quad (5.24)$$

and therefore

$$g'_{i+1}(x) = \begin{cases} (\alpha_i^2 - \beta_i^2) g'_i(x) & \dots & t_i \leq x \leq t_{i+1} \\ (\alpha_i^2 + \beta_i^2) g'_i(x) & \dots & \text{otherwise.} \end{cases} \quad (5.25)$$

Noting that $\alpha_i^2 - \beta_i^2 = s_i$ and $\alpha_i^2 + \beta_i^2 = 1$ completes the proof. \square

We can use the lemma above as an inductive basis for the proof of the following lemma:

Lemma 3. *Any 1-LPL, 1-dimensional function f with gradient 1 before the first and after the last non-linearity and with k non-linearities and $2l$ extreme points can be expressed with k 1-Lipschitz, linear layers and $(k + l - 1)$ \mathcal{N} -activations.*

Proof. We will proof this lemma by induction on l . If $l = 0$, f is increasing, so we proved this case in the previous lemma.

Now suppose we want to fit a function f with $2l$ local extreme points. Define k_s as position of highest local maxima, and k_t as the position of lowest local minima with $k_t > k_s$. Then, by the definition of k_s and k_t , we have that $f(k_s) > f(k_t)$ and

$$\begin{array}{rccccccc} & f(x) & \leq & f(k_s) & \dots & & x \leq k_s \\ f(k_t) & \leq & f(x) & \leq & f(k_s) & \dots & k_s \leq x \leq k_t \\ f(k_t) & \leq & f(x) & & & \dots & k_t \leq x. \end{array} \quad (5.26)$$

We will define a function g , that is similar to f but with the part between k_s and k_t flipped. Formally,

$$g = \begin{cases} f(x) - 2f(k_s) & \dots & x \leq k_s \\ -f(x) & \dots & k_s \leq x \leq k_t \\ f(x) - 2f(k_t) & \dots & k_t \leq x. \end{cases} \quad (5.27)$$

Then g does not have local extreme points at k_s and k_t , and it keeps all other local extreme points of f (and does not have any new ones). Therefore, g has $2l - 2$ local extreme points, so by induction we can represent it by a network with k layers and $(k + (l - 1) - 1)$ \mathcal{N} -activations.

Furthermore, the properties we stated above for f imply similar properties for g :

$$\begin{array}{rccccccc} & g(x) & \leq & g(k_t) & \dots & & x \leq k_s \\ g(k_s) & \leq & g(x) & \leq & g(k_t) & \dots & k_s \leq x \leq k_t \\ g(k_s) & \leq & g(x) & & & \dots & k_t \leq x. \end{array} \quad (5.28)$$

This implies that

$$g(k_s) \leq g(x) \leq g(k_t) \iff k_s \leq x \leq k_t. \quad (5.29)$$

Define a function h as applying the \mathcal{N} -activation with certain parameters to the output of g :

$$h(x) = \mathcal{N}(g(x); g(k_s), g(k_t)). \quad (5.30)$$

Then

$$h'(x) = \begin{cases} -g'(x) & \dots & k_s \leq x \leq k_t \\ g'(x) & \dots & \text{otherwise,} \end{cases} \quad (5.31)$$

so $h'(x) = f'(x)$, and therefore $h(x) = f(x)$ up to a bias term. Therefore, we know we can express f by using g and an additional \mathcal{N} -activation. So, by induction we know we can express f using a network with k layers and $(k + l - 1)$ \mathcal{N} -activations. This completes the proof. \square

Corollary 3. Any 1-LPL function f with bounded input and k non-linearities can be expressed using $(k + 2)$ linear 1-Lipschitz layers and $(\frac{3}{2}k + 2)$ \mathcal{N} -activations.

Proof. Consider the continuous function that agrees with f on the bounded input interval, and has derivative 1 outside of it. Call it g . This function has (at most) $(k + 2)$ non-linearities. Therefore, by our previous theorem, we can express this function by a network with $(k + 2)$ layers and $(k + 2 + l - 1)$ \mathcal{N} -activations, where $2l$ is the number of local extreme points of g . Now note that $2l \leq k + 2$, since any local extreme point must be at a non-linearity. This completes the proof. \square

With this, we proved that \mathcal{N} -activations are universal as long as the inputs are bounded. We will extend this result below to show that in fact any 1-LPL, 1-dimensional function, even with potentially unbounded inputs can be expressed by a \mathcal{N} -activation network, as long as we are allowed to use 3 absolute value-activations (or set $\theta_1 = -\infty$).

Lemma 4. *Any 1-LPL function f with potentially unbounded input region and k non-linearities can be expressed by a network of width 2, with $k + 5$ linear 1-Lipschitz layers, $(\frac{3}{2}k + 2)$ \mathcal{N} -activations as well as 3 absolute value activations.*

Proving this lemma will also finally proof Theorem 4, since setting $\theta_1 = -\infty$ and $\theta_2 = 0$ makes an \mathcal{N} -activation identical to an absolute value-Activation.

Proof. For this theorem, the gradient of f outside the regions where all the non-linearities lie will be important. Define s_s and s_t as the gradient of f before the first and after the last non-linearity. We will also again denote the non-linearities of f as t_1, \dots, t_k .

As a first case, suppose $s_s \geq 0$ and $s_t \geq 0$. Note that either (or both) of them being 0 is a bit of a special case, but we have designed our proof so that it also works in this case. Now if $s_s > 0$, define v_s such that $v_s \leq t_1$ and $f(v_s) = \min_{1 \leq i \leq k} f(t_i)$. With this definition $f(x) \leq f(v_s) \iff x \leq v_s$. If $s_s = 0$, set $v_s = t_1$. Similarly, if $s_t > 0$, define v_t such that $v_t \geq t_k$, and furthermore $f(v_t) = \max_{1 \leq i \leq k} f(t_i)$. If $s_t = 0$, set $v_t = t_k$.

Now define g as the (continuous) function that has $g(x) = f(x)$ as long as $v_s \leq x \leq v_t$, and g has slope 1 otherwise. By Lemma 3 (and the fact that $2l \leq k$) we can express g using $(k + 2)$ layers and $(\frac{3}{2}k + 2)$ \mathcal{N} -activations.

Further define h such that $h(x) = x$ as long as $v_s \leq x \leq v_t$, and h has slope s_s and s_t outside this region.

Then, note that $f(x) = g(h(x))$, as they agree when $v_s \leq x \leq v_t$, and the derivatives of f and $g \circ h$ also agree outside that region.

It turns out that h itself can be expressed using 3 layers and 2 Absolute Value activations. For example, we can use a linear combination of x and $-|x - v_t|$ in order to obtain a function with slope 1 if $x \leq v_t$ and slope s_t otherwise, and similarly change the slope when $x \leq v_s$ to s_s . The coefficients are defined similarly to Equation 5.21. This completes the proof for the case where $s_s \geq 0$ and $s_t \geq 0$.

The case $s_s \leq 0$ and $s_t \leq 0$ can be reduced to the one above by changing the sign in the first layer.

Finally, there are the cases that either $s_s < 0$ and $s_t > 0$ or $s_s > 0$ and $s_t < 0$. Again, by a sign change we only need to consider one of them, say $s_s > 0$ and $s_t < 0$. In this case f does have a global minimum at one of the non-linearities, call this point v_m . We consider a function g that has $g'(x) = f'(x)$ when $x \leq v_m$, and $g'(x) = -f'(x)$ otherwise. The case above us tells us that g can be expressed, and applying an Absolute Value Activation to the output of g (and a suitable bias before and after that) allows us to express f . \square

This finally also proves Theorem 4, so we know for 1-dimensional functions, \mathcal{N} -activation networks can express any 1-LPL functions.

5.4 Experimental Setup

5.4.1 Fitting the \mathcal{N} -function

We first designed a toy experiment to show that not only is it theoretically impossible to exactly express the \mathcal{N} -function, but also approximating the \mathcal{N} -function well with MaxMin networks seems impossible, even when the inputs are bounded. To show this, we train networks to fit the \mathcal{N} -function on the interval $[-3, 3]$. As a training set, we sample 1000 points uniformly from $[-3, 3]$, and obtain the targets by applying the \mathcal{N} -function. We try to fit this data using networks with different activation functions. Each network consists of 3 fully connected layers with activation functions in between. We use AOL to constrain the linear layers to be 1-Lipschitz. The layers are of width 40. We optimize each network using Nesterov SGD with learning rate 0.01 and momentum of 0.9, with a batch size of 100 for 1000 epochs. We fit each of the networks to our training set using mean squared error, and also report the mean squared error on the training set for the different models.

5.4.2 Certified Robust Classification

In order to confirm that our activation is not only useful in theory and in constructed toy problems, we also compare our activation function to MaxMin in the task of certified robust classification. In the previous section we showed that \mathcal{N} -activations are more expressive than MaxMin activations. It is not clear that higher expressiveness actually leads to better certified robust accuracy. However, we do use a loss function that encourages high robust accuracy on the training data. We hope that the fact that \mathcal{N} -activation networks are more expressive than MaxMin networks will allow \mathcal{N} -activation networks to obtain a lower loss, and that this lower loss also translates to better performance on the test data.

For our experimental setup, we mostly reuse the setup from Chapter 4. We describe the details of our setup below.

Architecture Our architecture is similar to the medium-sized (M) architecture from the previous chapter. We change the pooling function to *ConcatenationPooling*. In order to compare the activations, in half of the experiments, we replace the MaxMin activation with the \mathcal{N} -activation. In this chapter we show experimental results for methods AOL, CPL and SOC.

Optimization Our main focus is again on the certified robust accuracy for $\epsilon = 36/255$. As in the previous chapter we use *OffsetCrossEntropy* as defined in Equation (3.20) with an offset of $2\sqrt{2}\epsilon$ and temperature parameter $t = \frac{1}{4}$. As hyperparameter search, for every type of convolutional layer used, we select the best learning rate on CIFAR-10 with MaxMin activation, and use this learning rate also with the \mathcal{N} -activation and on other datasets. This ensures that we do not give an unfair advantage to our proposed method. We again use SGD with a momentum of 0.9 and *OneCycleLR* as a learning rate scheduler. We train for 1000 epochs with batch size 256. We subtract the dataset means before training. As data augmentation, we use random crops and random flips on CIFAR, and *RandAugment* [CZSL20] on Tiny ImageNet.

Initialization of the \mathcal{N} -activation Initialization can be a very important aspect of 1-Lipschitz networks, since those usually do not have residual connections or batch normalization

layers that help against vanishing gradients. We also found that the initialization of the \mathcal{N} -activation is crucial to ensure good performance. We took inspiration from the MaxMin activation. In particular, we noticed that the MaxMin activation is equivalent (up to a rotation of input and output) to applying absolute value and identity map in an alternating way (see Equation (5.16)). We initialize the \mathcal{N} -activation accordingly: For each pair of channels, for the first channel we set $\theta_1 = -100$ and $\theta_2 = 0$ (to initialize as the absolute value function), for the second channel we set $\theta_1 = \theta_2 = 0$ (to initialize as the identity map).

Learning rate for the \mathcal{N} -activation Unfortunately, we observed that using the same learning rate for the parameters of linear layers and for the parameters of the \mathcal{N} -activation does hurt performance. In order to overcome that we rescale the parameters of the \mathcal{N} -activation by a factor of $1/10$, effectively reducing the learning rate.

5.5 Empirical Results

5.5.1 Fitting the \mathcal{N} -function

We plot the mean squared errors on the training set for different activation functions during the 1000 training epochs in Figure 5.2. In this figure we also visualize the actual functions that were learned by the models.

We observe that even though the \mathcal{N} -function is 1-Lipschitz, neither the MaxMin network nor the absolute value network manage to fully reduce the training loss, even when trained for 1000 epochs. In contrast, the \mathcal{N} -activation network achieves much lower loss, as predicted by our theory. We also show in Figure 5.2 that the function learned when using MaxMin or absolute value networks is visibly different to the \mathcal{N} -function.

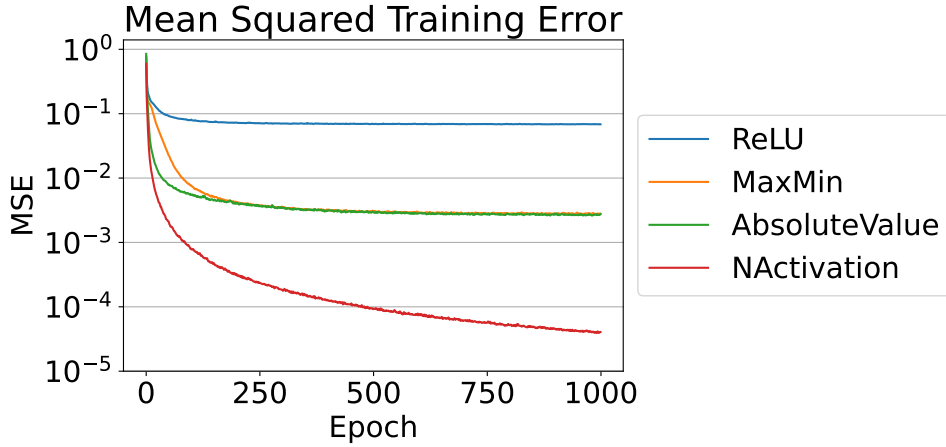
5.5.2 Certified Robust Classification

We show results where we compare our proposed \mathcal{N} -activation to the MaxMin activation in Figure 5.3 as well as Table 5.1. We can see that it is in fact possible to replace the commonly used MaxMin activation with our \mathcal{N} -activation without reducing the certified robust accuracy. In 7 out of 9 settings, the certified robust accuracy (for $\epsilon = 36/255$) is actually slightly higher when using the \mathcal{N} -activation. This shows that the \mathcal{N} -activation is in fact competitive to the MaxMin activation. However, it also shows that the additional expressive power that comes with the \mathcal{N} -activation does not improve performance on certified robust classification by a larger amount, at least in the setup we consider. This indicates that there might be other fundamental restrictions of 1-Lipschitz networks, and the lack of expressiveness is not that important in practice.

Interestingly, both cases where MaxMin performs better than the \mathcal{N} -activation are with SOC layers. So the best choice of activation function might depend on the 1-Lipschitz layer used.

5.5.3 Ablation Experiment

We conducted an ablation experiment to evaluate how important the initialization strategy for the \mathcal{N} -activation is. We compare 3 different ways of initializing: The first way is the initialization described in section 5.4.2, we refer to it as *AbsId-Initialization*. The second way is to initialize the \mathcal{N} -activation as the identity map, by setting all parameters initially to



(a) Mean squared training error throughout training.

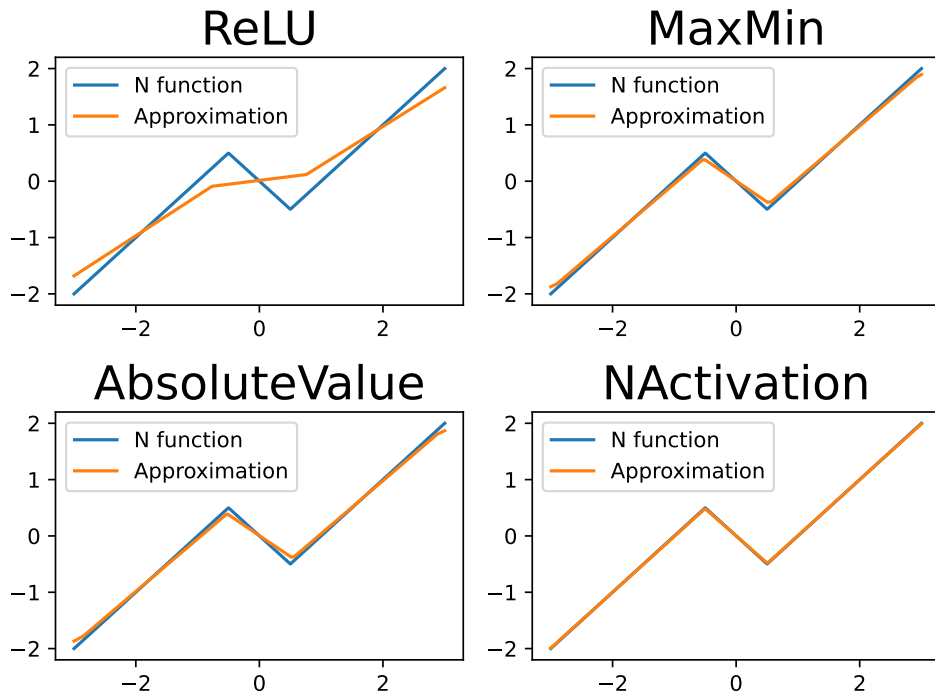
(b) Approximation of the \mathcal{N} -function on the interval $[-3, 3]$.

Figure 5.2: ReLU networks, MaxMin networks and absolute value networks can not fit the \mathcal{N} -function, whereas \mathcal{N} -activation networks can.

zero (*Zero-Initialization*). The third way, *Random-Initialization* is to choose the initialization values randomly. We set $\theta_1 = -10^{u_1}$ and $\theta_2 = +10^{u_2}$, for u_1 and u_2 randomly sampled in the interval $[-5, 0]$.

The results of this ablation study can be found in Figure 5.4 and Table 5.2. We see that the kind of initialization does influence the certified robust accuracy a model achieves. For CPL, that is already a non-linear layer, Zero-Initialization performs comparable to AbsId-Initialization. The Random-Initialization does perform worse than AbsId-Initialization in both settings.

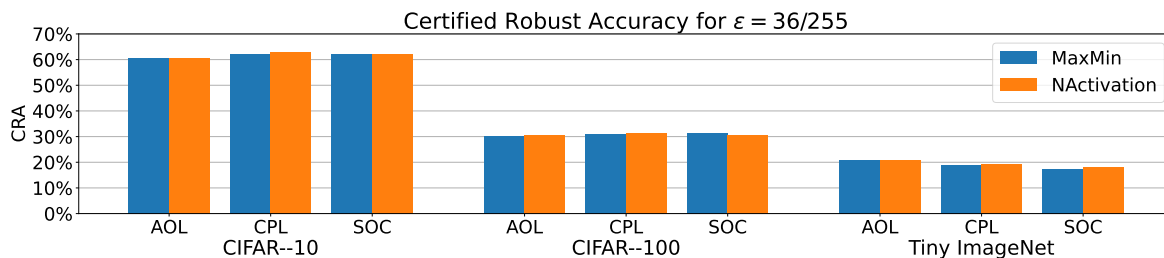


Figure 5.3: **Comparing performance of MaxMin and \mathcal{N} -activation** on different datasets with different 1-Lipschitz layers.

Table 5.1: **Comparing performance with MaxMin and \mathcal{N} -activation.** Test set accuracy and certified robust accuracy for different values of ϵ .

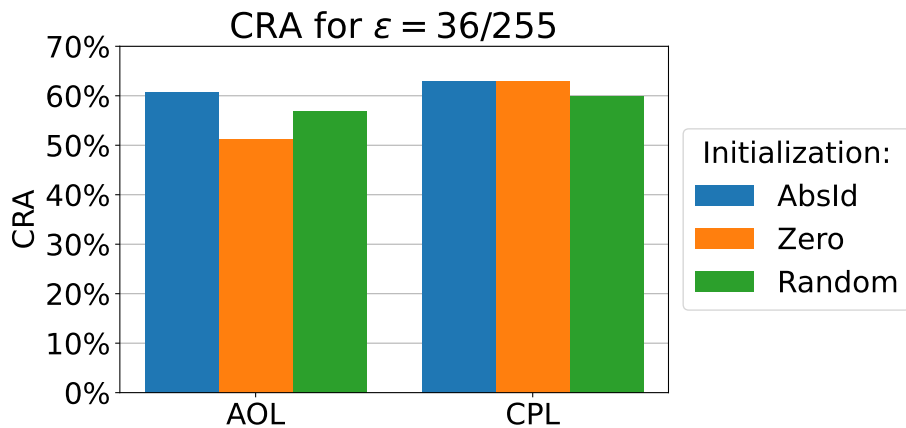
Dataset	Layer Type	Activation	Accuracy	Certified Robust Accuracy			
				$\epsilon = \frac{36}{255}$	$\epsilon = \frac{72}{255}$	$\epsilon = \frac{108}{255}$	$\epsilon = 1$
CIFAR-10	AOL	MaxMin	72.7%	60.6%	47.0%	34.1%	4.3%
		NActivation	72.9%	60.7%	46.9%	34.7%	4.5%
CIFAR-10	CPL	MaxMin	74.9%	62.2%	47.9%	35.2%	4.0%
		NActivation	75.0%	63.0%	48.6%	35.6%	4.2%
CIFAR-10	SOC	MaxMin	74.5%	62.3%	49.2%	36.0%	5.0%
		NActivation	74.5%	62.2%	49.6%	36.7%	5.1%
CIFAR-100	AOL	MaxMin	43.1%	30.4%	20.7%	14.0%	2.1%
		NActivation	43.2%	30.6%	20.9%	14.1%	2.1%
CIFAR-100	CPL	MaxMin	43.8%	31.1%	21.3%	14.3%	2.3%
		NActivation	43.3%	31.2%	21.6%	14.8%	2.4%
CIFAR-100	SOC	MaxMin	43.0%	31.2%	21.3%	14.3%	2.1%
		NActivation	42.8%	30.6%	21.2%	14.4%	2.3%
Tiny ImageNet	AOL	MaxMin	30.4%	20.8%	13.6%	9.0%	1.0%
		NActivation	30.8%	20.9%	13.5%	9.1%	1.1%
Tiny ImageNet	CPL	MaxMin	28.6%	19.0%	12.4%	8.0%	0.8%
		NActivation	28.3%	19.3%	12.4%	8.0%	0.8%
Tiny ImageNet	SOC	MaxMin	26.4%	17.2%	10.9%	6.9%	0.6%
		NActivation	27.1%	18.1%	10.9%	7.1%	0.8%

5.6 Discussion

In this chapter, we analyze the expressive power of 1-Lipschitz networks. We show that the use of previously proposed activation functions, such as *MaxMin*, causes an unnecessary restriction of the class of functions that 1-Lipschitz networks can represent, and we propose the \mathcal{N} -activation as a more expressive alternative. In particular, we prove that in the one-dimensional setting, \mathcal{N} -activation networks are universal, in the sense that they can represent any 1-Lipschitz piece-wise linear function with finitely many segments. Our experiments show that \mathcal{N} -activation networks are not only theoretically appealing but are also a reasonable

Table 5.2: **Comparing \mathcal{N} -activation initializations** on CIFAR-10.

Layer Type	\mathcal{N} -activation Initialization	Accuracy	Certified Robust Accuracy			
			$\epsilon = \frac{36}{255}$	$\epsilon = \frac{72}{255}$	$\epsilon = \frac{108}{255}$	$\epsilon = 1$
AOL	AbsId	72.9%	60.7%	46.9%	34.7%	4.5%
	Zero	65.3%	51.2%	36.5%	24.4%	1.2%
	Random	70.1%	56.8%	41.8%	29.1%	2.3%
CPL	AbsId	75.0%	63.0%	48.6%	35.6%	4.2%
	Zero	75.4%	63.0%	49.6%	36.2%	4.5%
	Random	73.1%	59.9%	45.2%	32.3%	2.8%

Figure 5.4: **Comparing \mathcal{N} -activation initializations.** Certified robust accuracy on CIFAR-10.

replacement for the MaxMin activation in experiments, reaching comparable certified robust accuracy in standard benchmarks. However, the improvements we achieve are tiny, which indicates that the lack of expressiveness is not the current bottleneck limiting performance of 1-Lipschitz classifiers. In fact, as demonstrated in Figure 5.2, MaxMin networks can approximate the \mathcal{N} -function fairly well on a bounded domain. This also makes us believe that MaxMin networks should be able to find solutions as good as \mathcal{N} -activation networks for practical problems.

Intriguing Properties of Robust Classifiers

This chapter appears in full in [PL25], Bernd Prach and Christoph H. Lampert, Intriguing properties of robust classification, first published in the *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2025.

In the previous chapter we investigated whether the activation function is causing the lack of performance of 1-Lipschitz classifiers. Whilst we found a theoretical limitation of the currently used activation function, it barely seems to influence the performance in experiments. Therefore, in this chapter, we investigate another factor that might limit the performance of robust classifier: the amount of training data.

We will first show that in certain settings robust generalization is only possible with unrealistically large amounts of data. Specifically, we find a setting where a robust classifier exists, it is easy to learn an accurate classifier, yet it requires an exponential amount of data to learn a robust classifier. Based on this theoretical result, we evaluate the influence of the amount of training data on datasets such as CIFAR-10. Our findings indicate that the the amount of training data is the main factor determining the robust performance. Furthermore we show that there are low magnitude directions in the data which are useful for non-robust generalization but are not available for robust classifiers. This implies that robust classification is a strictly harder tasks than normal classification, thereby providing an explanation why robust classification requires more data. Finally, we show that we can obtain great robust accuracy on the training data itself, showing that current models are expressive enough to fit the currently available training data.

Our main contributions in this chapter are three *insights* that we hope will clear up some misconceptions and hopefully guide future research on training robust networks in new directions.

Insight 1: There are settings in which learning robust accurate classifiers requires much more data than learning just accurate classifiers. Specifically, we present a learning problem in which any learning algorithm requires an amount of training data exponential in the data dimension, otherwise it cannot learn a robust classifier that is meaningfully better than chance level. Our construction is based on the fact that non-robust classifiers are able to exploit low-magnitude features in the data, while robust classifiers have to rely on

high-magnitude features. The exponential gap between robust and non-robust learning opens up when the former generalize well but the latter ones do not.

Insight 2: Even for real data, robust learning can require a lot more data than non-robust learning. We provide evidence that the problem of Insight 1 is not just theoretical, but happens in (less drastic) form also for real datasets. Specifically, for MNIST, CIFAR-10 and CIFAR-100 we demonstrate that the amount of data is a major determinant of performance. We further show that linear subspaces of the input space exist that only contain a tiny amount of the variance of the data, so those directions cannot be used for robust classification. However, when projecting our data into those subspaces, we can still obtain great (non-robust) accuracy. This implies that enforcing robustness makes classification a strictly hard task on CIFAR-10, providing an explanation why robust classification requires more data.

Insight 3: Robust architectures can fit and generalize non-robustly. Training robust models requires certain architectural choices that are different from standard networks. We show that this is not the reason for the lack of performance on test data. Architectures built for robust classification are expressive enough to robustly overfit the training data, and we can also learn classifiers that generalize well, we just struggle to learn robust classifiers that generalize well.

We believe these insights show how important the amount of training data is for robust classification. In the remainder of the chapter, we state our insights more formally and report in detail on our theoretical and empirical findings.

6.1 Robust Classification Needs More Data

We start our discussion by two observations. First, deep learning has been so successful for many classical computer vision tasks that it has become a routine task to train classifiers on a training dataset in a way so that the classifier also performs well on unseen test data afterwards. Second, for many such tasks, classifiers of high *robust* accuracy are provably possible. Namely, the human visual system provides proof for this, as human perception is typically not just highly accurate (we use it to generate the “ground truth” of our datasets), but also robust, in the sense that it is unaffected by small perturbations of its input.

It is tempting to assume that those two observations (classifiers learned from data generalize well, and high-accuracy robust classifiers do exist) imply that it is also possible to *learn* a high-accuracy robust classifier. However, in the following we show that this conclusion does not hold. Informally, we show that for any dataset size there exists a family of data distributions such that robust classification is possible, learning an accurate classifier is easy, but learning an accurate robust classifier is impossible.

Our result is formalized in the following Theorem.

Theorem 5 (No Free Robustness). *For any dataset size n there exists a family, \mathcal{F} , of binary classification problems such that the following 3 properties hold:*

1. *For any $\mathcal{D} \in \mathcal{F}$, there exists a classifier with 100% robust accuracy.*
2. *There is a learning algorithm that for any $\mathcal{D} \in \mathcal{F}$ and $S \sim \mathcal{D}$ finds a (linear) classifier with 100% test accuracy.*

3. For any learning algorithm, \mathcal{L} , on average over $\mathcal{D} \in \mathcal{F}$ and $S \stackrel{iid}{\sim} \mathcal{D}$, the learned classifier $\mathcal{L}(S)$ achieves robust accuracy less than 51% on \mathcal{D} .

Proof. This proof consist of an explicit construction of a family of data distributions that fulfills the three conditions.

Defining \mathcal{F} . We first need to define our family of classification problems, \mathcal{F} . For that, we first set the data dimension to $d = \lceil \log_2 n \rceil + 7$. We denote the set of all binary functions on the $(d - 1)$ -dimensional hypercube as Φ ,

$$\Phi = \left\{ \phi : \{\pm 1\}^{d-1} \rightarrow \{\pm 1\} \right\}. \quad (6.1)$$

Note that this is a very large set with size $|\Phi| = 2^{2^{d-1}}$. For every $\phi \in \Phi$ we will define a data distribution \mathcal{D}_ϕ , then our family of distribution is given as

$$\mathcal{F} = \{ \mathcal{D}_\phi : \phi \in \Phi \}. \quad (6.2)$$

In order to sample a pair (\mathbf{x}, y) from \mathcal{D}_ϕ , we will sample x_i uniformly from $\{+1, -1\}$ for $i = 1, \dots, (d - 1)$. Then we will set $x_d = \delta \phi(x_1, \dots, x_{d-1})$ for some small scalar δ , and $y = \text{sign}(x_d)$. Here, x_1, \dots, x_{d-1} are *robust* (large magnitude) features. Their relation to the ground truth label y is deterministic ($y = \phi(x_1, \dots, x_{d-1})$), but it is hard to learn because of the size of Φ . In contrast, x_d is a useful, non-robust feature: It is perfectly correlated with the label, but because of its small magnitude it can be easily perturbed.

Proof of statement 1. For any $\mathcal{D}_\phi \in \mathcal{F}$ with associated mapping ϕ , consider the classifier $f(x_1, \dots, x_d) = \phi(\text{sign}(x_1), \dots, \text{sign}(x_{d-1}))$. It is robust against any perturbations of size $\epsilon < 1$, because any such perturbation of the robust features is undone by the sign function, and it has perfect accuracy, because it coincides with the labeling function ϕ .

Proof of statement 2. Consider a learning algorithm that always outputs the classifier $f(x_1, \dots, x_d) = \text{sign}(x_d)$. Then, because $y = \text{sign}(x_d)$ holds for all data distributions, it follows that f has perfect accuracy on future data.

Proof of statement 3. The third property requires a slightly longer proof, resembling the *No Free Lunch* theorem, e.g. [SSBD14, Theorem 5.1]. Intuitively, it is based on the fact that a robust classifier cannot rely on the value of feature x_d , because that can be set to 0 by a perturbation of size δ . Furthermore, the functional relation between (x_1, \dots, x_{d-1}) and y , is hard to learn because of the size of Φ .

In order to proof the statement, first note that the average adversarial test error for perturbation of size $\leq \delta$ is given as

$$\mathbb{E}_{\mathcal{D}_\phi \in \mathcal{F}} \mathbb{E}_S \mathbb{E}_{S \stackrel{iid}{\sim} \mathcal{D}_\phi} \mathbb{P}_{(x,y) \sim \mathcal{D}_\phi} \left[\exists \mathbf{x}' \text{ s.t. } \|\mathbf{x} - \mathbf{x}'\|_2 \leq \delta \text{ s.t. } \mathcal{L}(S)(\mathbf{x}') \neq y \right]. \quad (6.3)$$

We can get a lower bound to this quantity by considering only a single attack that sets the “non-robust” feature x_d to 0. We will write $\tilde{\mathbf{x}}$ for the result of applying this attack to an input \mathbf{x} . With this we can lower bound the average adversarial test error by

$$\mathbb{E}_{\mathcal{D}_\phi \in \mathcal{F}} \mathbb{E}_S \mathbb{E}_{S \stackrel{iid}{\sim} \mathcal{D}_\phi} \mathbb{P}_{(x,y) \sim \mathcal{D}_\phi} \mathcal{L}(S)(\tilde{\mathbf{x}}) \neq y. \quad (6.4)$$

Next we will rewrite sampling $\mathcal{D}_\phi \in \mathcal{F}$ and $S \stackrel{iid}{\sim} \mathcal{D}_\phi$ as sampling $\phi \in \Phi$, and once we know ϕ , sampling from \mathcal{D}_ϕ is equal to sampling the robust features from the hypercube $\{\pm 1\}^{d-1}$, as the non-robust feature x_d and the label depend deterministically on the robust features. We can furthermore change the order of sampling the robust features and sampling $\phi \in \Phi$. We will write X^r and \mathbf{x}^r for these robust features. Using this, the lower bound on the average adversarial test error becomes:

$$\mathbb{E}_{X^r} \mathbb{E}_{\mathbf{x}^r} \mathbb{E}_{\phi \in \Phi} \mathbb{1} \left[\mathcal{L}(X^r, \phi(X^r))(\tilde{\mathbf{x}}^r) \neq \phi(\mathbf{x}^r) \right] \quad (6.5)$$

Now note that when $\mathbf{x}^r \notin X^r$, then $\phi(\mathbf{x}^r)$ becomes independent of $\phi(X^r)$. Therefore, since we assumed a uniform distribution on Φ , any learner will be correct exactly $\frac{1}{2}$ of the time. When $\mathbf{x}^r \in X^r$, any reasonable learner should be able to predict correctly, we will bound the error in this case by 0. Using this, the lower bound on the average adversarial test error becomes

$$\mathbb{E}_{X^r} \mathbb{E}_{\mathbf{x}^r} \mathbb{1} \left[\mathbf{x}^r \notin X^r \right] \frac{1}{2}. \quad (6.6)$$

Now the probability that $\mathbf{x}^r \in X^r$ is at most $\frac{n}{2^{d-1}}$, so we know that the average adversarial test error is at least $\frac{1}{2} - \frac{n}{2^d}$, and therefore at least 49% by our choice of $d = \lceil \log_2 n \rceil + 7$. \square

The previous result established a lower bound on the worst case behavior, by showing that in certain settings we do require exponentially many data points (exponentially in the dataset dimension). Next, we provide a matching upper bound: As long as the input domain is bounded and some robust classifier exists, exponentially many data points suffice to learn an accurate and robust classifier. For this we do need to assume that there exists a robust classifier that is robust to perturbations with bounded L_∞ norm. Note that here is the only part of the chapter where we use L_∞ norm, everywhere else we assume L_2 distances. More precisely:

Theorem 6. *Assume that there exists a L_∞ robust classifier (margin δ) on data distribution \mathcal{D} , where the data points are in $[0, 1]^d$. Then as long as we have $n \geq 37 \left\lceil \frac{1}{\delta} \right\rceil^d$ training points independently sampled from \mathcal{D} , for margin $\delta/2$, we can achieve average L_∞ robust test accuracy of at least 99% (average over sampling training sets).*

We will prove that in the setting above, the one-nearest-neighbor classifier will get 99% robust accuracy. In order to show this, we first show that for any test point, the nearest point of a different class is at least 2δ away. Therefore, if there is a training point within distance δ of a test point, no perturbation of size at most $\delta/2$ applied to the test point can change the prediction of the one-nearest-neighbor classifier algorithm. Finally, we show that the probability (over sampling training set and test point) of having a training example within δ is $\geq 99\%$.

Proof. In order to prove this theorem we will show that in this setting, the 1-nearest neighbor algorithm achieves robust accuracy of at least 99%.

In order to show this, we first show that for any test point, the nearest point of a different class is at least an L_∞ distance of 2δ away. Assume that f is a robust classifier on \mathcal{D}_ϕ . Consider a test point \mathbf{x} and the nearest training point \mathbf{x}_j that is of a different class. We know that f robustly classifies both \mathbf{x} and \mathbf{x}_j with radius δ . This implies that any point of distance $\leq \delta$ to either of the points must share a label with that point, and therefore \mathbf{x} and \mathbf{x}_j must be at least 2δ apart.

For a test point \mathbf{x} with label y , suppose there exists a training point \mathbf{x}_i that is less than δ away from \mathbf{x} . By our assumption, this training point also has label y . Consider any other point $\tilde{\mathbf{x}}$ with $\|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty \leq \delta/2$. Furthermore, consider any training point \mathbf{x}_j of a different class than \mathbf{x} . Using the triangle inequality we get that

$$\|\tilde{\mathbf{x}} - \mathbf{x}_i\|_\infty \leq \|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty + \|\mathbf{x} - \mathbf{x}_i\|_\infty < \frac{\delta}{2} + \delta = \frac{3\delta}{2} \quad (6.7)$$

$$\|\tilde{\mathbf{x}} - \mathbf{x}_j\|_\infty \geq \|\mathbf{x} - \mathbf{x}_j\|_\infty - \|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty \geq 2\delta - \frac{\delta}{2} = \frac{3\delta}{2}. \quad (6.8)$$

Therefore, we know that the nearest training point to $\tilde{\mathbf{x}}$ must have label y . This shows that the one-nearest-neighbor classifier is robust to perturbation of size $\leq \delta/2$ of \mathbf{x} .

With this established it just remains to be shown that with enough training examples, for 99% of test points \mathbf{x} , there will be a training point close to \mathbf{x} . In order to prove that this is the case, we will split the hypercube into a set of disjoint boxes. The probability of a test point being close to a training point is at least as big as the probability of the test point being in a box that has at least one training point inside. We define the boxes by defining a set of *box centers*: For $D = \lceil 1/\delta \rceil$, set $\mathcal{C} = [\frac{1}{2}\delta, \frac{3}{2}\delta, \dots, \frac{2D-1}{2}\delta]^d$. We define $B_r(C)$ as the L_∞ ball with radius r around C . Further, we set \mathcal{B} to be the set of all boxes, $\mathcal{B} = \{B_{\delta/2}(C) : C \in \mathcal{C}\}$. We have that $|\mathcal{B}| = D^d = \lceil 1/\delta \rceil^d$. We will write p_B for the probability of a data point lying in box B under distribution \mathcal{D} . With this, we can write the probability of having at least 1 training point in box B as

$$\mathbb{P}(\exists i : \mathbf{x}_i \in B) = 1 - \mathbb{P}(\mathbf{x}_i \notin B \forall i) \quad (6.9)$$

$$= 1 - \prod_{i=1}^n (1 - \mathbb{P}(\mathbf{x}_i \in B)) \quad (6.10)$$

$$= 1 - (1 - p_B)^n. \quad (6.11)$$

We further have that $(1 - p)^n = (1 - p)^{\frac{1}{p}np} \leq \exp(-np)$, and therefore

$$\mathbb{P}(\exists i : \mathbf{x}_i \in B) \geq 1 - \exp(-np_B). \quad (6.12)$$

We will also use that $\exp(x) \geq 1 + x$ for all x , and therefore $\exp(x - 1) \geq x$, and

$$x \exp(-x) \leq \exp(-1). \quad (6.13)$$

Then, putting everything together, for p the probability that a test point \mathbf{x} is classified robustly

we have that:

$$p \geq \mathbb{P}\left(\min_i \|\mathbf{x} - \mathbf{x}_i\|_\infty \leq \delta\right) \quad (6.14)$$

$$\geq \sum_{B \in \mathcal{B}} \mathbb{P}(\mathbf{x} \in B) \mathbb{P}(\exists i : \mathbf{x}_i \in B) \quad (6.15)$$

$$\geq \sum_{B \in \mathcal{B}} p_B (1 - \exp(-np_B)) \quad (6.16)$$

$$= 1 - \sum_{B \in \mathcal{B}} p_B \exp(-np_B) \quad (6.17)$$

$$= 1 - \frac{1}{n} \sum_{B \in \mathcal{B}} np_B \exp(-np_B) \quad (6.18)$$

$$\geq 1 - \frac{1}{n} \sum_{B \in \mathcal{B}} \frac{1}{e} \quad (6.19)$$

$$= 1 - \frac{|\mathcal{B}|}{ne} \quad (6.20)$$

Now since $|\mathcal{B}| = \lceil 1/\delta \rceil^d$ and we assumed that $n \geq 37 \lceil 1/\delta \rceil^d$ we get that $p \geq 99\%$. \square

Note that we can adapt the proof to work for any margin $\delta' < \delta$, and not just for $\delta/2$. Furthermore, if we only assume L_2 robustness we might need many more data points, namely $\mathcal{O}(c^d d^{d/2})$ for some constant c .

6.2 Experimental Setup

In order to quantify robust classifiers on datasets such as MNIST and CIFAR-10, we train various robust and standard (non-robust) models. We provide some background and describe architectures and training setup below.

6.2.1 SimpleConvNet

In this chapter, we want to compare the performance of standard machine learning architectures with 1-Lipschitz ones. For this we of course require a model and training pipeline that allows achieving competitive accuracy on CIFAR-10. We started with the setup described in [Win22], and simplified model and training pipeline further. For example, we remove techniques such as label smoothing or weight decay that do not seem influence performance.

Our final model, *SimpleConvNet*, consists of 8 convolutional layers and one linear layer. Each convolutional layer uses *BatchNorm* [IS15] and ReLU as the activation function. The model uses *MaxPooling* in order to reduce the resolution between convolutional layers, and also as a global pooling before the linear layer. For details see [Pra24]. We usually use *OffsetCrossEntropy* (Section 3.4) with offset 0 and temperature 8 to train this model.

When we train this model for 24 epochs with optimizer as described in Section 6.2.4 (and learning rate 0.1), we get about 93.7% accuracy on a CIFAR-10 test set.

6.2.2 1-Lipschitz models

For the robust 1-Lipschitz models, we either use an 8 layer MLP or the ConvNet architecture from Chapter 4. As 1-Lipschitz layers we use AOL (from Chapter 3) or CPL [MDAA22].

Unless mentioned otherwise, we use the loss function defined in Section 3.4, with offset and temperature both set to $\frac{1}{4}$. The only hyperparameter we tune is the peak learning rate, we train models with randomly sampled learning rates for 100 epochs each, and pick the learning rate of the model with the highest certified robust accuracy on a validation set. We usually train 1-Lipschitz models for 3000 epochs.

6.2.3 Randomized Smoothing

We also estimate the robust performance of a *Randomized Smoothing* classifier [CRK19] (see Equation (2.3)). Since in this chapter we are purely interested in estimating robust performance, we directly report the robustness approximation based on random samples. We use a SimpleConvNet (Section 6.2.1) for the base classifier f and during training we add Gaussian noise to the images in addition to the data augmentation described in Section 6.2.4. We found that setting the standard deviation σ to $\frac{1}{8}$ and training for 100 epochs gave us good results on a validation set, therefore we used this values for our evaluation runs. We approximate the class probabilities by sampling ϵ 1000 times.

6.2.4 Optimization.

We train all our models using SGD with Nesterov momentum of 0.9 and batch size of 256 with *OneCycleLR* as a learning rate scheduler. As data pre-processing we subtract the training data mean from every channel, we do not rescale the data. We use the same data augmentation as [Win22, Pra24]. It consists of random crops, random flips and setting a random patch of the image to zero.

6.3 Experimental Results

6.3.1 Robust scaling behavior

Recent work on robust image classification has shown that additional data can greatly increase robust accuracy on CIFAR-10. Also, in Theorem 5 we showed that the amount of training data can be an important limiting factor for robust classification. Therefore, in this section, we want to explore how the size of the training data influences the performance of a robust classifier on real data.

Scaling dataset size. Therefore, for our first result, we subsampled CIFAR-10 in order to get datasets of different sizes, and evaluated the performance of models trained on those datasets. In order to keep the amount of compute the same for all settings, when we divide the dataset size by some value k we also multiply the number of epochs by k . We trained four models on different amounts of training data this way.

The results can be found in Figure 6.1. We found that the amount of training data indeed has a big influence on the (test) performance of these current classifiers: Doubling the amount of data seems to reliably increase the certified robust accuracy by about 5%. As an additional result, Figure 6.1 also shows that the training data allows higher certified robust accuracy than what 1-Lipschitz currently achieve.

We repeated the experiment described above on MNIST and CIFAR-100, see Figure 6.2 for the results. Considering all results together, we can nicely see that the certified robust accuracy

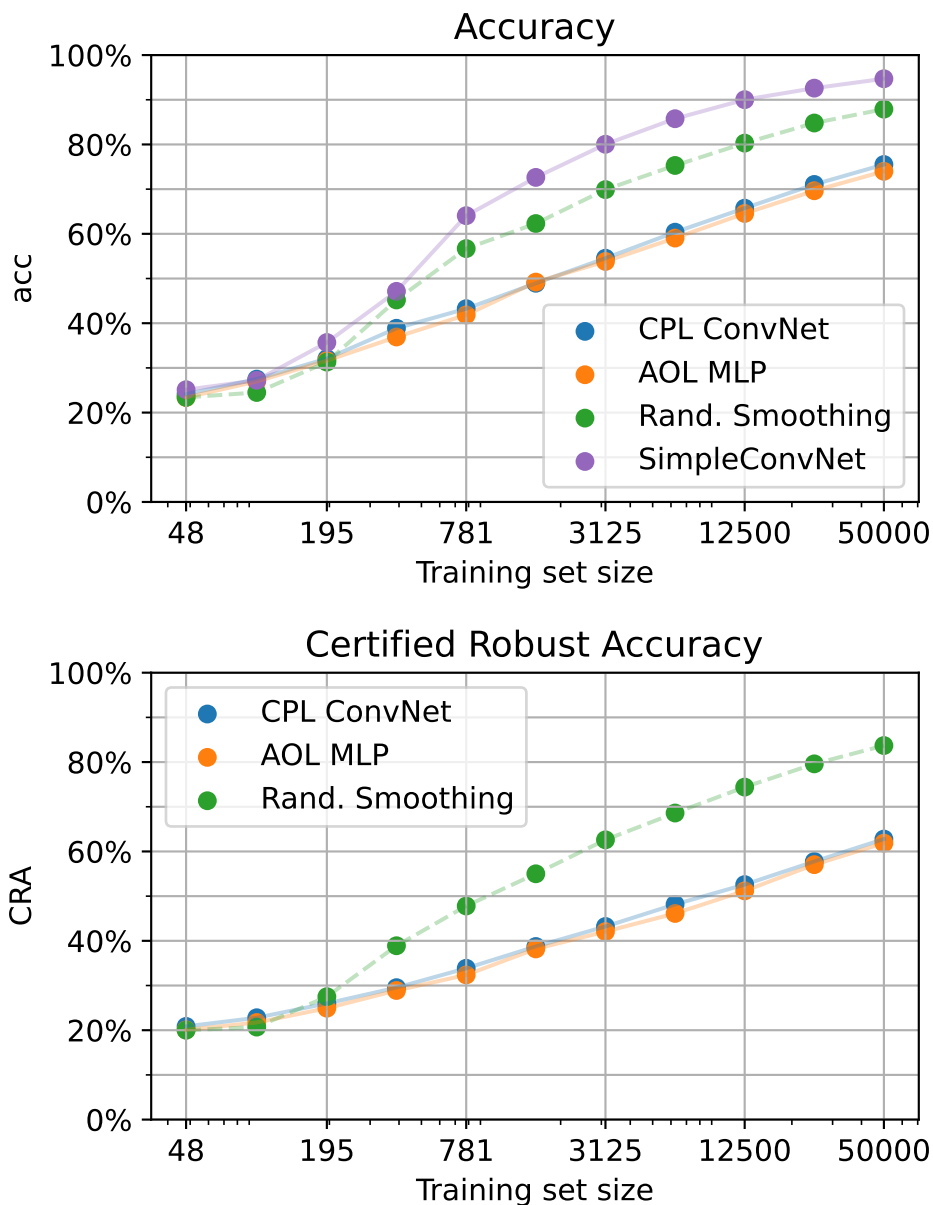


Figure 6.1: Scaling behavior on CIFAR-10. © 2025 IEEE.

follows a sigmoid curve when plotted against the logarithm of the training set size. No matter how little data, we can always robustly classifier at chance level. This is visible on CIFAR-100, as the curve starts out almost flat for little training data. With more data, the certified robust accuracy increases about linearly with the logarithm of the training set size. We can of course never get above 100% certified robust accuracy, so for larger dataset sizes the curve flattens again, which can be seen in the MNIST experiment.

Interestingly, we also observe that across datasets, when training long enough, the convolutional architecture and the MLP have a very similar performance. It does seem that for robust classification, the inductive biases from the convolutional architecture are not very helpful, which is a big difference to non-robust classification.

Scaling compute. We did a similar set of experiments to evaluate the influence of the amount of compute. In Figure 6.3 we show how the certified robust accuracy changes with

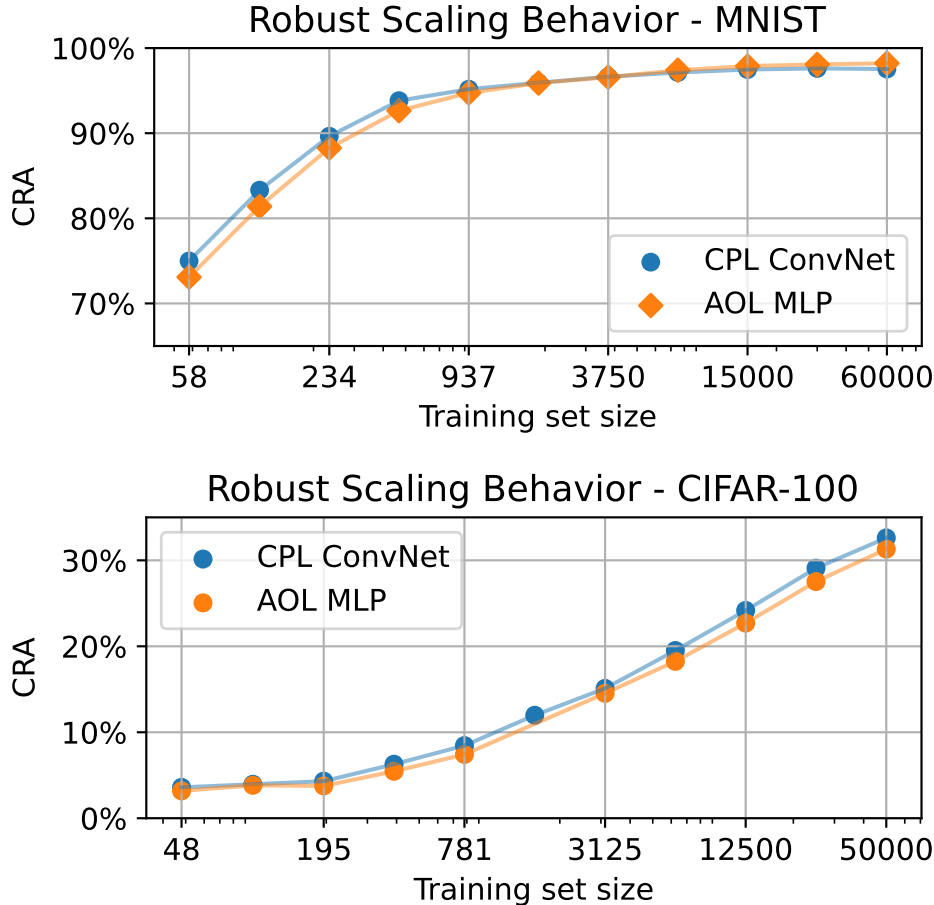


Figure 6.2: Scaling behavior on MNIST and CIFAR-100. © 2025 IEEE.

the number of training epochs. Whilst just increasing the number of epochs does also improve the performance, it has much less of an effect, and the curves flatten out with increasing compute. This indicates that currently the amount of data is the main bottleneck preventing great robust classifiers. Note that of course with more data, additional compute will be more useful. So we do expect that ideally we scale up both, dataset size and amount of compute.

Training on additional data. We are also interested whether the scaling law from Figure 6.1 also extends to larger training datasets, larger than the $50k$ images from the CIFAR-10 training dataset itself. Recently, there have been a lot of works that used data generated in the style of CIFAR-10 by a diffusion model [GRW⁺21, WPD⁺23, ADE⁺23, HZW⁺23, HLWF24].

We also followed this approach, and we used 1 million images from [WPD⁺23]. We subsampled this large dataset to obtain training sets of different sizes. We always trained for a fixed number of epochs for this experiment (3000 for 1-Lipschitz models, 100 for randomized smoothing), so we did use more compute with larger datasets this time.

In addition to our own results, we also report the performance of the current best 1-Lipschitz model [HLWF24]. The authors generated 1 million additional CIFAR-10 style images with a diffusion model, using a model trained on 940 million images for data filtering. Training with this additional data allows them to achieve 78.1% certified robust accuracy.

We show the results in Figure 6.4. The scaling behavior from Figure 6.1 extends to larger datasets sizes and generated data for all models considered. However, the improvements from

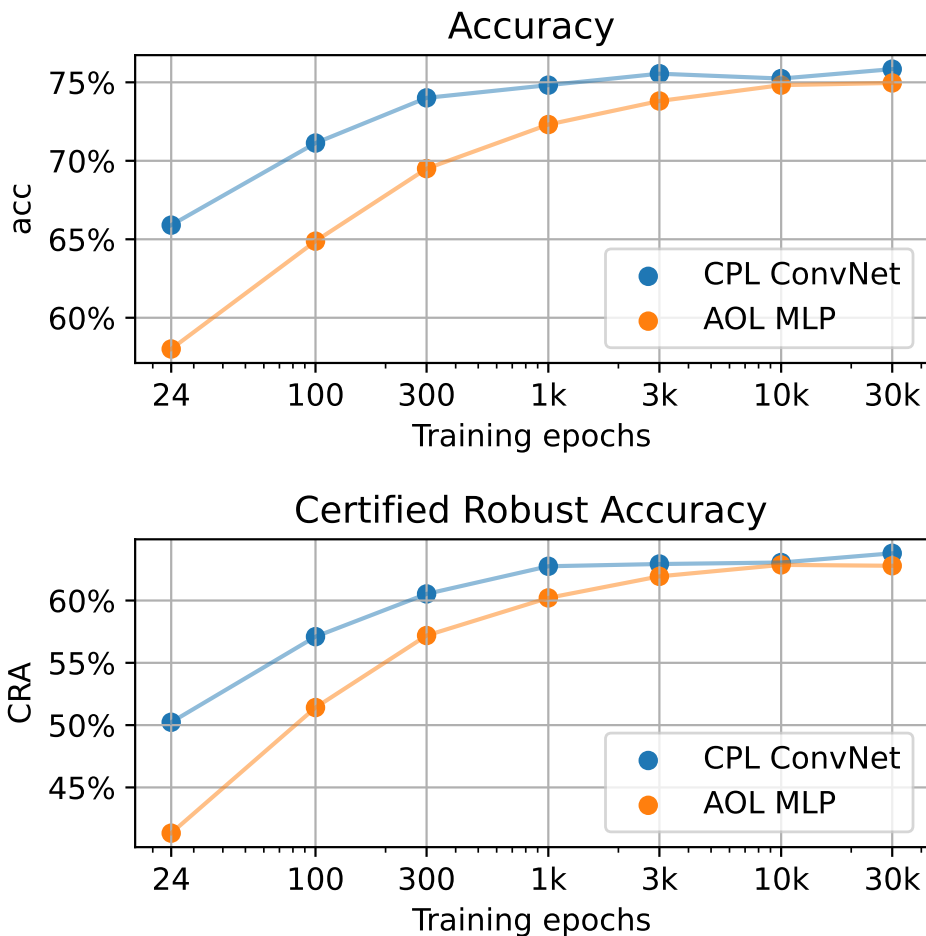


Figure 6.3: Scaling compute. © 2025 IEEE.

training 1-Lipschitz models naively on more data, without increasing the model size eventually diminish. The results from related work that carefully designs the training setup to maximize test performance show that it is possible to extend the improvement. This effect might also in part be due to a difference in quality of the generated data. For randomized smoothing our estimate of performance does scale very well with the amount of data.

6.3.2 Robust and non-robust features

In our theoretical section we have established that robust classification can be much harder than non-robust classification, which is also what we observe in experiments. In our theoretical example this hardness comes from a feature of small magnitude and high predictive power. In this section we aim to evaluate whether CIFAR-10 has similar properties: We want to know whether there is a subspace of the input space, such that the data has very low variance when projected onto that subspace, yet the projection is useful for classification. It turns out that it is the case. For example, we can find such a subspace by considering the principal components [Pea01] of the dataset. The principal components of a data set are orthogonal basis vectors, such that principal component i maximizes the variance of the data that lies in the subspace spanned by the first i principal components.

For our experiments we flatten the training images in order to evaluate the principal components. Then in different experiments we project the flattened (train and test) images onto different

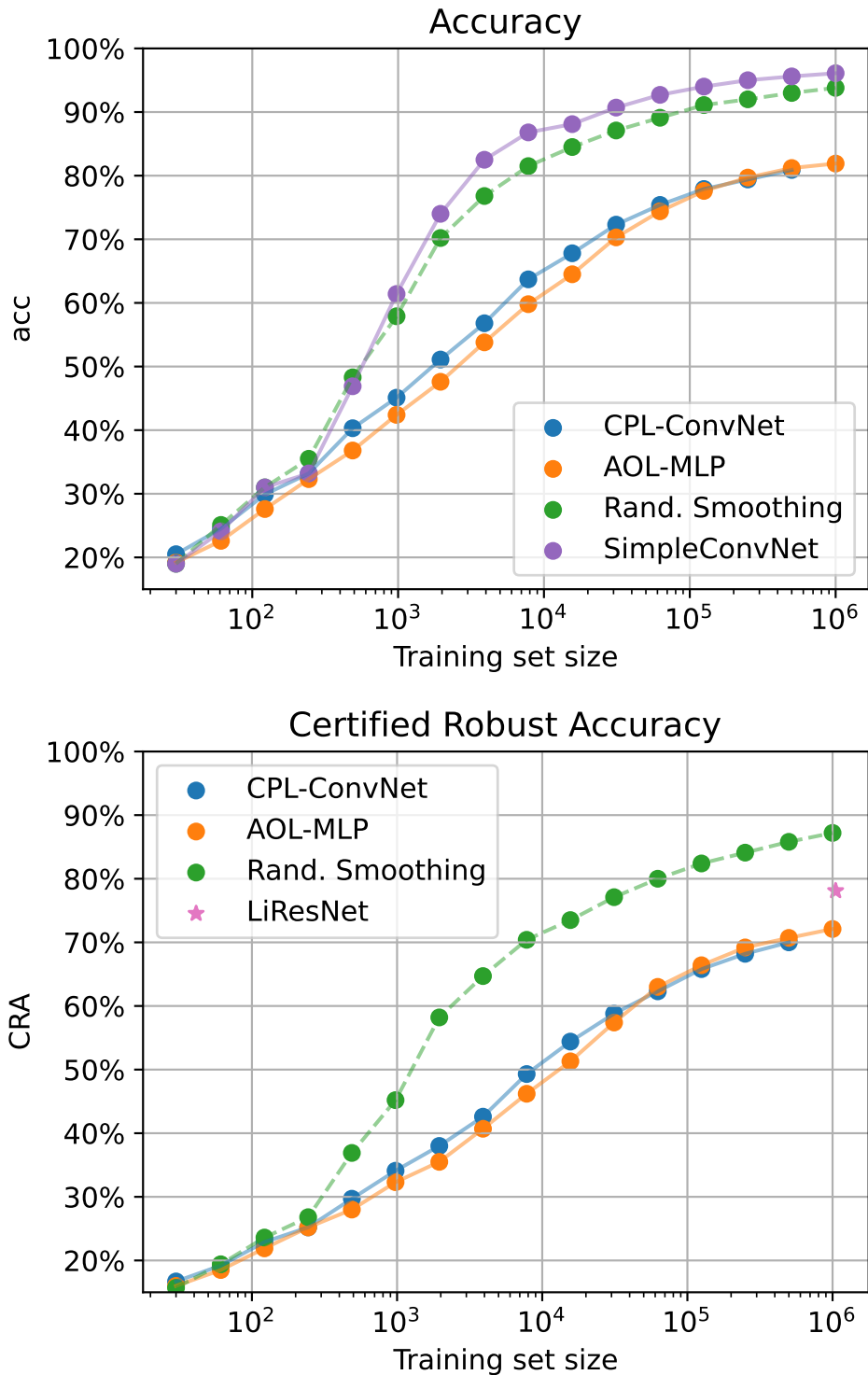


Figure 6.4: Scaling behavior on generated data. © 2025 IEEE.

subsets of principal components. After the projection, we transform the vectors back into the image space, so that we can train a standard and a 1-Lipschitz convolutional CPL network on the data without modifications to the setup.

Find our results in Table 6.1 and Figure 6.5. It turns out that when considering the subspace spanned by all but the first 512 components, only about 2% of the data variance are in this subspace. However, when training a standard network on this data, we obtain about 85% test

PCs	Variance Explained %	Accuracy %		CRA %	
		Train	Test	Train	Test
1-16	72	99	43	64	31
1-512	98	100	91	89	61
513-3072	2	100	85	9	9
2049-3072	0.02	99	39	0	0
1-16 & 513-3072	74	100	86	65	35
1-3072	100	100	94	93	62

Table 6.1: Performance on different subsets of principal components, as well as the proportion of variance explained by it. © 2025 IEEE.

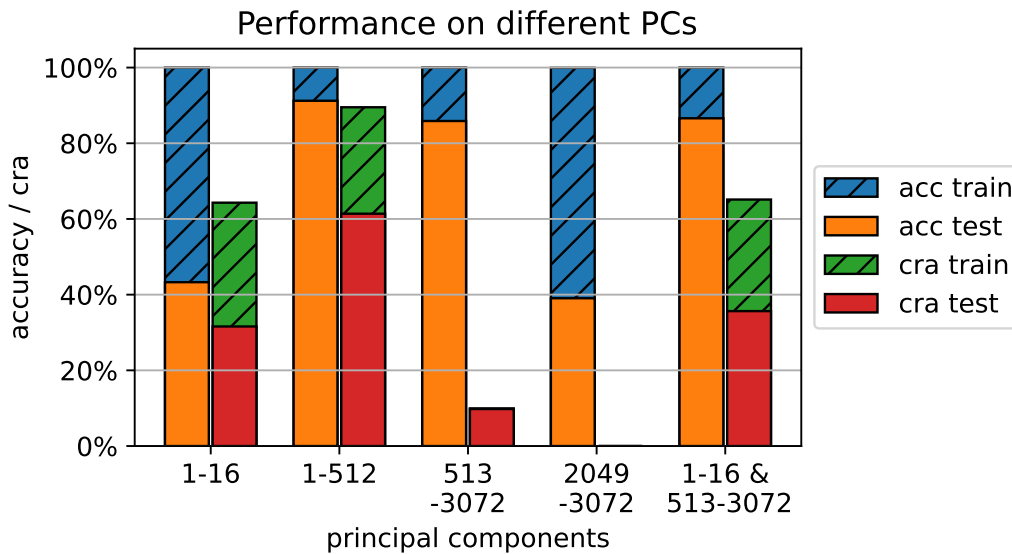


Figure 6.5: Performance on different subsets of the principal components. © 2025 IEEE.

accuracy. Similarly, when projecting on the last 1024 (out of 3072) components, we get only 0.02% of the variance, not enough to allow any robust classification, not even on the training data. However, we trained a standard network to achieve about 39% accuracy on the test set, so there is still some weak signal in the last principal component, a signal that we cannot use for robust classification due to its low magnitude.

From this we conclude that there are in fact low magnitude directions on CIFAR-10 that are useful for classification, yet because of the small magnitude they are not useful for robust classification. This is a property that CIFAR-10 shares with our example from Section 6.1.

We are also interested in the principal components of high variance. It turns out that the first principal components are not very useful for generalizing. When using only the first 16 principal components for example, we do get about 72% of the total variance, and we can fit standard networks to get $\geq 99\%$ training accuracy. Furthermore we can also train a 1-Lipschitz ConvNet to get good certified robust training accuracy (64% with augmentation, 97% without) on this low-dimensional subspace. However, the standard convolution network only achieves about 43% accuracy on the test set, suggesting that there is only a weak signal in those features despite high variance.

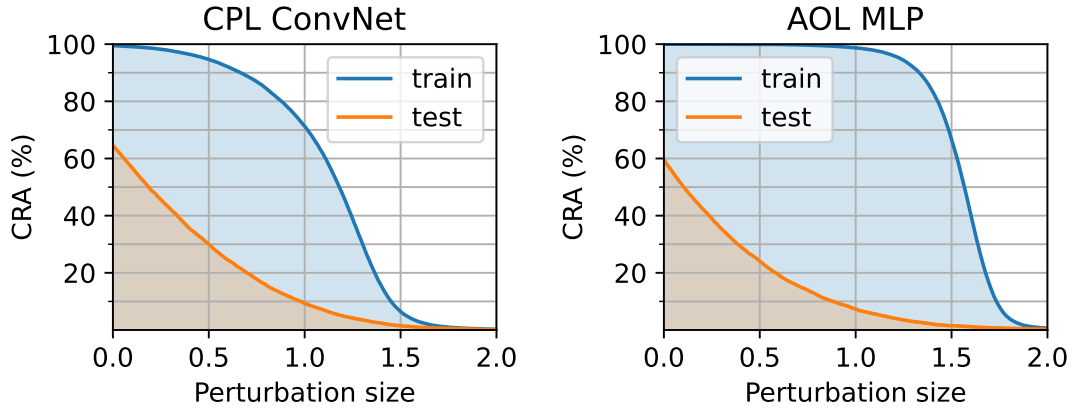


Figure 6.6: We can robustly overfit the CIFAR-10 training set. © 2025 IEEE.

Based on this observation, we created another dataset which contains the PCA components 1–16 together with the components 513–3072, that is, high magnitude and low magnitude features but not the intermediates. This data suffices for non-robust learning (86% test accuracy), but robust learning fails (35% robust test accuracy). We take this result as an indication that CIFAR-10 as a real dataset shares some characteristics with the hypercube example in Theorem 5: it contains high-magnitude features, which do not allow generalizing, and features of tiny magnitude, which generalize, but which no robust classifier is able to exploit.

6.3.3 Robust overfitting

Previous works have suspected that the lack of robust performance might be due to underfitting: Our models might not have enough capacity to fit the data robustly. In our next experiment we will show that this is not the case on CIFAR-10, we can train a 1-Lipschitz networks to perfectly fit the training data, and do so in a robust way. We train an CPL ConvNet and an AOL MLP. In order to overfit robustly, we set the offset in our loss function to $\sqrt{2}$, and we train without augmentation for 3000 epochs.

We show the results in Figure 6.6. First note that we can clearly fit the training data robustly. For the MLP, even for perturbations of size 1, we get almost perfect certified robust accuracy on the training set. However, it is also visible that the classifiers do not generalize well. The performance on the test set is much worse for any perturbation size.

Importantly, this result does not imply that our models have enough capacity to fit the data distribution robustly, only that they have enough capacity for the amount of training data we currently have. We believe that when scaling up to amount of training data by a few magnitudes, we will benefit from larger models. However, the results in Figure 6.6 show that the capacity is not the current bottleneck.

6.3.4 Robust architectures can generalize

In this final experimental section we want to explore whether it is the model architecture that prevents 1-Lipschitz models from generalizing. In order to prevent vanishing gradients and other problems when training 1-Lipschitz networks, there are a few important adaptations from standard convolutional networks. For example, in 1-Lipschitz model we use a different activation function, no MaxPooling and no BatchNorm. We hypothesize that the absence of

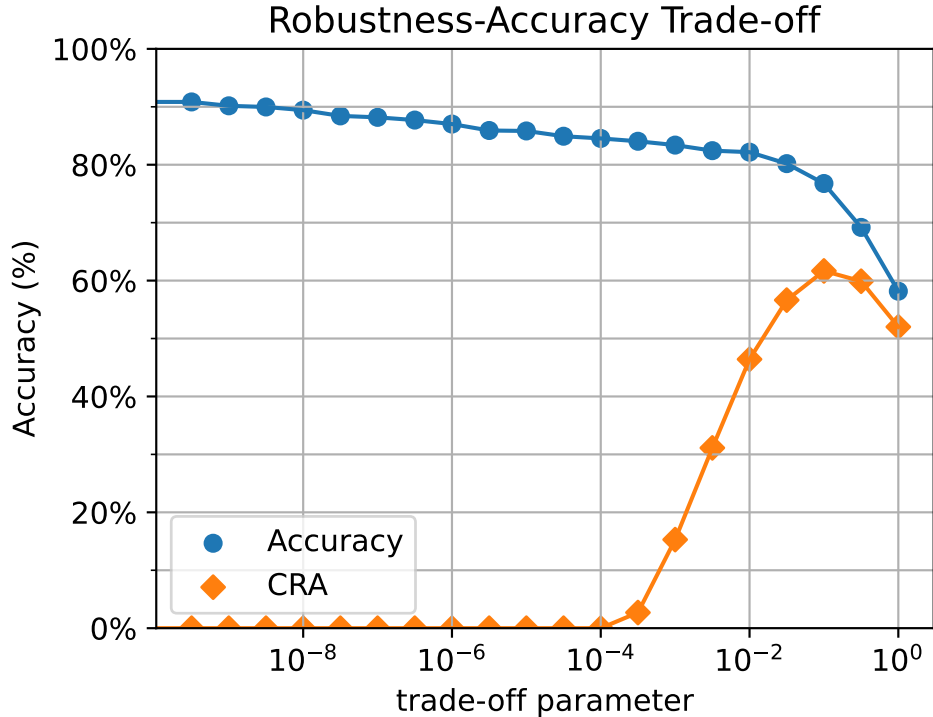


Figure 6.7: **1-Lipschitz model can generalize.** One model can be trained either to good accuracy (with $t \sim 0$) or to good certified robust accuracy (with $t \sim \frac{1}{10}$). © 2025 IEEE.

those layers affect the ability of models to perform well on the test set. This seems plausible especially since 1-Lipschitz training usually generates classifier with relatively low accuracy.

However, we found that this hypothesis is not true, at least not on CIFAR-10. In order to show this, we adapted a 1-Lipschitz architecture to archive competitive accuracy on CIFAR-10, whilst preserving the 1-Lipschitz property.

We managed to achieve this by changing the initialization strategy and the loss function of the model. Our preliminary experiments suggested that random initialization is important for high accuracy, so we chose to initialize the layers as random orthogonal transformations. Furthermore, our initial experiments suggested that we required at least a single normalization layer to obtain accurate models, however, such a layer is unfortunately not 1-Lipschitz.

In order to solve this problem, we define a loss function that includes this normalization term: For trade-off parameter t we define our loss as

$$\text{CrossEntropy} \left(\text{Softmax} \left(\frac{s}{\text{std}(s) + t} - y \right), y \right), \quad (6.21)$$

where s is the vector of scores predicted by the model, y is a one-hot encoding of the label and $\text{std}(s)$ denotes the standard deviation of s .

With this loss function we get the desired results: Setting $t = 0$ we can train our architecture to competitive accuracy (93.2%), when setting $t = \frac{1}{10}$ we get competitive certified robust accuracy (61.7%). We also visualize the accuracy and CRA the AOL-ConvNet achieves for different values of the trade-off parameter t in Figure 6.7.

Our results show that current 1-Lipschitz architectures can generalize comparably to traditional ConvNets, so the architectural restrictions of 1-Lipschitz models are not the reason why those

models fail to generalize well. This is more evidence that the task of robust classification is harder and solving it requires more data, and the lack of robustness is not just an artifact of current methods.

6.4 Related Work

In the following section we will provide an overview of related work on robust generalization as well as other attempts of explaining the lack of robustness of current models.

We will first describe related work on **robust generalization**. One closely related piece of work is [SST⁺18]. In their work, the authors show that even in a very simple example on Gaussian distributions, robust classification can require many more training examples than non-robust classification. In their example, we can construct an accurate classifier from just a single training example, but for ϵ -robust classification with L_∞ -norm we need about $\Omega(\epsilon^2 \sqrt{d} / \log(d))$ examples, for d the data dimension.¹ This is a very interesting results, that could explain part of the gap in performance between standard classification and robust classification. Our paper differs in that we are mainly interested in L_2 robustness, and in our example we show that the gap in samples required could potentially be much larger, and we might require $\Omega(2^d)$ training examples.

The work of [SST⁺18] has been extended by [BCM19] and [DWR20], where the authors also consider different L_p norms, and prove some bounds about the excess risk. We are more interested in distribution where the optimal robust classifier actually achieves 0 risk, and furthermore we think that the convergence rate to this optimal risk is not that important, but the sample complexity of getting within (e.g.) 1% of the optimal risk is a more useful quantity to study.

Other works have also produced results about robust generalization. For example, in [RXY⁺19] the authors introduce a data distribution where adversarial training can hurt the test performance of a (regression) model. The authors also argue that we do not actually need labeled data in order to improve performance. As long as we have unlabeled data, we can use a standard network to produce pseudo-labels, and (as long as the data is not adversarial) those labels should be fairly accurate. In [MCK21] the authors show that for Gaussian data the test loss of a linear classifier might actually get worse with additional data, or might show some double descent behavior. In [BJC21] the authors consider data distribution where a perfectly accurate robust classifier exists. They show that in this scenario, learning a robust classifier with maximal possible margin can need d times more samples. We show in our paper that robust classification can be hard not just when aiming for maximum possible margin, but also when the goal is robustness to smaller perturbations.

A different attempt of explaining the lack of performance of current robust models is by blaming the **robustness-accuracy trade-off** [TSE⁺19]. It has been observed theoretically as well as empirically that on certain data distributions a classifier can be either very accurate or reasonably robust, but not both [TSE⁺19, RXY⁺20, ZYJ⁺19, Doh18, BBS⁺22]. Whilst this trade-off offers interesting insights in general, we think it is not the most promising way to study the lack of performance in image classification tasks, where a classifier that is both accurate and robust at the same time does exist. However, we also see a robustness-accuracy trade-off in our experiments in this chapter.

¹Here, Ω is the Bachmann–Landau notation: for functions f and g , we write $f(d) = \Omega(g(d))$ if for some M and d_0 it holds that $|f(d)| \geq M|g(d)|$ for all $d \geq d_0$.

Other authors argue that robust classification might require much more complex models, where complexity can refer to the hypothesis class [FFF18, Nak19, PL23], the amount of compute required [DNV19, BLPR19] or the size of the model required [BLN21, LJZ⁺22]. The distributions used to prove results are often similar to our example distribution, there is a map that is hard to learn (*e.g.* from the hypercube to the label), but the data comes with an additional feature of small magnitude that allows us read off the label. These results are further related to our work as in order to process exponentially (in d) many examples, one definitely also requires compute exponentially in d , at either training (*e.g.* for a neural network) or at inference time (*e.g.* for a 1-nearest neighbor classifier). So our result also implies the result that on certain distributions we do require exponential amount of compute.

Another related concept are **robust and non-robust features** [IST⁺19]. The authors introduce the idea that adversarial examples might not directly be artifacts of the way we train the models, but exist because of the data distribution. Furthermore, they exist because of features that are useful and generalize well but are not robust. In [IST⁺19] the authors use a very general definition of features, and consider any map from the input space to the real numbers a feature. We believe this definition is too general to give us insights about the datasets. We show that even linear subspaces of the input space exist with tiny variance, and yet projecting into these subspaces still allows achieving great (non-robust) performance.

There is also a recent piece of work [BDPK24] that studied **robust scaling laws**, however they consider perturbations with bounded L_∞ norm. In their setting they concluded that (with current techniques) it will require unreasonable amounts of compute (much more than to train recent LLMs) to match human performance on CIFAR-10.

6.5 Summary and Discussion

Even 10 years after adversarial examples have entered the community's attention, robust classification is far from solved. Furthermore it is also not clear why the problem of robust classification is so hard to solve, and we still struggle on very simple datasets with robustness to fairly small perturbations. In our paper we have aimed to collect theoretical facts and empirical evidence about robust classification, in particular about robust generalization, in order to give the field a better understanding of the phenomena.

We first showed that there are data distributions on which it is not possible to train a robust classifier, unless the amount of data is unreasonably large. Moreover, this can be the case even for distribution where we can easily learn a good (non-robust) classifier, and where a perfect robust classifier exists.

Based on this insight, we evaluated whether similar results also hold on real data. We showed that on popular datasets including CIFAR-10, the performance of current models seems to be mainly determined by the amount of training data. Furthermore, we showed that as in our theoretical example, CIFAR-10 does have low-magnitude directions that cannot be used for robust classification, yet they are useful for training a standard classifier.

Finally, we showed that the lack of performance of 1-Lipschitz classifiers is not a consequence of the architecture. In particular, 1-Lipschitz models are expressive enough to fit the training data very robustly. Furthermore, 1-Lipschitz architecture are also able to do (non-robust) generalization very well and a single architecture can be trained either to good test accuracy, or to good certified robust accuracy based on the choice of loss function. We just currently fail

do both (robust fitting and generalizing) at the same time, indicating that robust classification is in fact a much more difficult task than normal classification.

Overall we highlighted how important the amount of data is for training a robust classifier. We presented arguments suggesting that training a robust image classifier on current training data could be an impossible task. We hope that future research will explore the effect of increasing the amount of training data on the certified robust accuracy further, and that awareness of the intriguing properties of robust classification we presented will allow the community to create better robust image classifiers in the future.

Conclusion and future work

7.1 Thesis Summary

In this work, we studied the problem of robust image classification with 1-Lipschitz classifiers. Our aim was to develop methods to improve the performance of 1-Lipschitz classifier, but also to simplify them and make them easier to use.

To achieve this, in Chapter 3 we introduced a simple and efficient approach to generate 1-Lipschitz layers. Our layer does not require evaluating any iterative procedure, which makes it simpler to use compared to existing methods at the time, whilst reaching comparable performance. We also proposed a loss function that allows optimizing for a specific margin more precisely. Our loss function has proven useful in the task of robust classification, and was adopted by the community.

In Chapter 4 we compared many different methods of creating 1-Lipschitz convolutions. We provided a theoretical analysis of computational complexity and memory requirements of different methods. We also conducted a large-scale empirical comparison on multiple datasets. Before the release of our work, it was hard to compare existing methods, as most papers used different setups, including differences in model sizes, training times, choice of optimizer and the use of caching during inference. Furthermore, usually properties such as computational complexity were not reported. Therefore, we believe our comparison is a helpful and important point of reference for choosing a 1-Lipschitz method.

One additional insight from our comparison is that the methods achieve fairly similar robust accuracy despite large differences in the approaches and computational complexities, so the choice of linear layer does not seem to be very relevant. Therefore we believe that instead of focusing on introducing new linear layers it is more promising to explore other aspects of 1-Lipschitz classifiers in order to understand their shortcomings and to improve their performance.

Therefore, in Chapter 5 we analyzed the role of the activation function. We managed to (negatively) answer the previously open question of whether MaxMin networks can express any reasonable function. We empirically showed that MaxMin networks also struggle to approximate a certain function very well. We further introduced an activation function that solved this problem. However, it turned out that the expressiveness is not currently the main limitation when training 1-Lipschitz networks.

Therefore, in Chapter 6 we investigated the influence of the dataset size on robust performance in various experiments. We first showed that there are data distributions for which enforcing robustness requires a huge amount of additional examples. We also found that on real data the amount of training data seems to be the main determinant of performance for current models. Next we investigated the data distribution, and showed the existence of directions in the input space that contain a tiny amount of data variance, yet they are very useful for classification. This shows that just like in our theoretical examples, there are features that a standard classifier can rely on, but those features are not helpful to a robust classifier because of their magnitude. Finally, we showed that 1-Lipschitz architectures can fit the training data very robustly, and generalize well, just not both of the same time.

7.2 Future Directions

7.2.1 1-Lipschitz models

In recent years, a high amount of attention went into parameterizing 1-Lipschitz linear layers. We believe this is now a solved problem, as we have various different methods at our disposal, including ones that scale well with the model size and input resolution. Therefore, we believe that future research should consider other aspects of 1-Lipschitz classification.

There are many other aspects that we do not yet understand well. For example, we believe the influence of the choice of activation function should be explored further. Whilst we have presented some results that hold for 1-dimensional functions, there is not much understanding for higher dimensions. In particular, we are interested in the kind of (multi-dimensional) functions that 1-Lipschitz networks with different activation functions can express.

Another important aspect is generalization of 1-Lipschitz classifiers. There is a huge gap between the performance of current 1-Lipschitz classifiers and the robust accuracy of randomized smoothing, so we know it is possible to train better models on limited amounts of data. A key direction for future research will be to understand this gap, and hopefully decrease it.

Finally, we are very curious to see the effect of initialization explored further. We have shown in this thesis that identity or near-identity initialization is common between well performing methods, and suspected that this initialization is a reason for good robust accuracy. However, we also found that using identity initialization decreases the (non-robust) accuracy of classifiers. Therefore, we believe that exploring this seemingly conflicting findings and understanding the role of initialization better is a promising future direction.

7.2.2 Robust classification

In certified robust classification, we currently have to choose one of two limitations. We can train 1-Lipschitz models, which usually come with a large computational overhead at training time and lack performance. However, many approaches have no overhead at all for inference. Alternatively, we can use randomized smoothing, and enjoy quick training and great performance, but suffer from a huge computational overhead for inference. We believe that combining the strengths of both approaches is an important direction for future research. For example, it might be possible to use distillation, and train a 1-Lipschitz model to match the (1-Lipschitz) function created by randomized smoothing.

We also hope to see more exploration into what makes the problem of robust classification so hard, even on very simple datasets. Whilst we have shown some analysis and evidence, such

as the existence of useful features with tiny magnitude, we believe there is much more to be done in the future.

Another interesting direction is to work on scaling up models, and better understanding the effects of this. We have shown that the amount of training data is currently a crucial determinant of performance, we are curious to find out how much improvement is possible by using more data.

Overall, we believe that the main challenge in robust classification is still to create classifiers and training pipelines that are more efficient and generalize better. Whilst scaling up is a possible solution on CIFAR-10, on more relevant tasks (such as classification of high resolution images), this seems less practical.

7.3 Larger Context

This thesis explores robustness to perturbations with small Euclidean norm in computer vision. We choose to restrict the Euclidean norm of the perturbations because it provides the simplest meaningful problem in robust classification that we struggle to solve. While our focus is on this specific constraint, we do expect that many of our findings will generalize to broader settings and contribute to improving robustness against a wider range of perturbations. In particular, our hardness results are highly relevant in these settings.

In order to obtain robust models, adversarial training remains more widely used than methods that provide guaranteed robustness. We believe this is due to the simplicity of adversarial training, as well as its higher estimated performance. In recent years, the field has developed techniques and insights that simplify the training of 1-Lipschitz models. However, at least under current known attacks, there still remains a large gap in performance between the two approaches. We hope further research will help to close this gap in order to make 1-Lipschitz methods competitive in practical applications.

Our research has only limited relevance to other robustness-related topics, such as robustness to distribution shifts or training-time attacks. Those are distinct research areas that only share part of the name with adversarial robustness. However we do believe robust networks can have applications beyond adversarial defense. In particular, robust networks are required for faithfully interpretable models, a topic of growing interest in the machine learning community.

Overall, robustness is currently not widely used in practical computer vision applications. We attribute this to an absence of attacks on deployed system that use machine learning. However, if such attacks become more common as a method to circumvent automatic filtering systems, it will be important to have access to robust models. Therefore, we believe developing them is an important tasks, as there might be a future need to deploy them alongside standard models.

Bibliography

- [ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning (ICML)*, 2017.
- [ADE⁺23] Thomas Altstidl, David Dobre, Björn Eskofier, Gauthier Gidel, and Leo Schwinn. Raising the bar for certified adversarial robustness with diffusion models. *arXiv preprint arXiv:2305.10388*, 2023.
- [AGCU20] Shayan Aziznejad, Harshit Gupta, Joaquim Campos, and Michael Unser. Deep neural networks with trainable activations and controlled Lipschitz constant. *IEEE Transactions on Signal Processing*, 2020.
- [AHD⁺23] Alexandre Araujo, Aaron J Havens, Blaise Delattre, Alexandre Allauzen, and Bin Hu. A unified algebraic perspective on Lipschitz neural networks. In *International Conference on Learning Representations (ICLR)*, 2023.
- [ALG19] Cem Anil, James Lucas, and Roger Grosse. Sorting out Lipschitz function approximation. In *International Conference on Machine Learning (ICML)*, 2019.
- [BB71] Å. Björck and C. Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 1971.
- [BBS⁺22] Louis Béthune, Thibaut Boissin, Mathieu Serrurier, Franck Mamalet, Corentin Friedrich, and Alberto Gonzalez Sanz. Pay attention to your loss: understanding misconceptions about Lipschitz neural networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [BCM19] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. Lower bounds on adversarial robustness from optimal transport. *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
- [BDPK24] Brian R. Bartoldson, James Diffenderfer, Konstantinos Parasyris, and Bhavya Kailkhura. Adversarial robustness limits via scaling-law and human-alignment studies. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ ICML 2024)*, 2024.
- [BJC21] Robi Bhattacharjee, Somesh Jha, and Kamalika Chaudhuri. Sample complexity of robust linear classification on separated data. In *International Conference on Machine Learning (ICML)*, 2021.
- [BKM23] Simone Bombari, Shayan Kiyani, and Marco Mondelli. Beyond the universal law of robustness: Sharper laws for random features and neural tangent kernels. In *International Conference on Machine Learning (ICML)*, 2023.

- [BLN21] Sébastien Bubeck, Yanzhi Li, and Dheeraj M Nagaraj. A law of robustness for two-layers neural networks. In *Conference on Learning Theory (COLT)*, 2021.
- [BLPR19] Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints. In *International Conference on Machine Learning (ICML)*, 2019.
- [BS21] Sébastien Bubeck and Mark Sellke. A universal law of robustness via isoperimetry. *Advances in Neural Information Processing Systems*, 2021.
- [Cay46] Arthur Cayley. About the algebraic structure of the orthogonal group and the other classical groups in a field of characteristic zero or a prime characteristic. *Journal für die reine und angewandte Mathematik*, 1846.
- [CBG⁺17] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning (ICML)*, 2017.
- [CHC19] Jeremy EJ Cohen, Todd Huster, and Ra Cohen. Universal Lipschitz approximation in bounded depth neural networks. *arXiv preprint arXiv:1904.04861*, 2019.
- [CN16] Artem Chernodub and Dimitri Nowicki. Norm-preserving orthogonal permutation linear unit activation functions (OPLU). *arXiv preprint arXiv:1604.02313*, 2016.
- [CRK19] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning (ICML)*, 2019.
- [CZSL20] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020.
- [DDS⁺09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [DGB⁺22] Stanislas Ducotterd, Alexis Goujon, Pakshal Bohra, Dimitris Perdios, Sebastian Neumayer, and Michael Unser. Improving Lipschitz-constrained neural networks by learning activation functions. *arXiv preprint arXiv:2210.16222*, 2022.
- [DNV19] Akshay Degwekar, Preetum Nakkiran, and Vinod Vaikuntanathan. Computational limitations in robust classification and win-win results. In *Conference on Learning Theory (COLT)*, 2019.
- [Doh18] Elvis Dohmatob. Limitations of adversarial robustness: strong no free lunch theorem. *arXiv preprint arXiv:1810.04065*, 2018.
- [DWR20] Chen Dan, Yuting Wei, and Pradeep Ravikumar. Sharp statistical guarantees for adversarially robust gaussian classification. In *International Conference on Machine Learning (ICML)*, 2020.
- [Eck20] Stephan Eckstein. Lipschitz neural networks are dense in the set of all Lipschitz functions. *arXiv preprint arXiv:2009.13881*, 2020.

- [FFF18] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. *Machine Learning*, 2018.
- [FZT18] Farzan Farnia, Jesse Zhang, and David Tse. Generalizable adversarial training via spectral normalization. In *International Conference on Learning Representations (ICLR)*, 2018.
- [GHL25] Moritz Grillo, Christoph Hertrich, and Georg Loho. Depth-bounds for neural networks via the braid arrangement. *arXiv preprint arXiv:2502.09324*, 2025.
- [GRW⁺21] Sven Gowal, Sylvestre-Alvise Rebuffi, Olivia Wiles, Florian Stimberg, Dan Andrei Calian, and Timothy A Mann. Improving robustness using generated data. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- [GSS15] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.
- [HGSTW20] Emiel Hoogeboom, Victor Garcia Satorras, Jakub Tomczak, and Max Welling. The convolution exponential and generalized Sylvester flows. In *Advances in Neural Information Processing Systems*, 2020.
- [HLWF24] Kai Hu, Klas Leino, Zifan Wang, and Matt Fredrikson. A recipe for improved certifiable robustness. In *International Conference on Learning Representations (ICLR)*, 2024.
- [HLZ⁺20] Lei Huang, Li Liu, Fan Zhu, Diwen Wan, Zehuan Yuan, Bo Li, and Ling Shao. Controllable orthogonalization in training DNNs. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [How19] Jeremy Howard. Imagenette: A smaller subset of 10 easily classified classes from imagenet. <https://github.com/fastai/imagenette/>, 2019. Accessed: 01.02.2024.
- [HZW⁺23] Kai Hu, Andy Zou, Zifan Wang, Klas Leino, and Matt Fredrikson. Unlocking deterministic robustness certification on Imagenet. *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.
- [IST⁺19] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
- [Jai89] Anil K Jain. Fundamentals of digital image processing. *Prentice-Hall google scholar*, 1989.
- [Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.
- [Lei23] Klas Leino. Limitations of piecewise linearity for efficient robustness certification. *arXiv preprint arXiv:2301.08842*, 2023.

- [LG16] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.
- [LHA⁺19] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Joern-Henrik Jacobsen. Preventing gradient attenuation in Lipschitz constrained convolutional networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
- [LJZ⁺22] Binghui Li, Jikai Jin, Han Zhong, John Hopcroft, and Liwei Wang. Why robust generalization in deep learning is difficult: Perspective of expressive power. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [LWF21] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning (ICML)*, 2021.
- [LY15] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 2015.
- [MCK21] Yifei Min, Lin Chen, and Amin Karbasi. The curious case of adversarially robust models: More data can help, double descend, or hurt generalization. In *Uncertainty in Artificial Intelligence (UAI)*, 2021.
- [MDAA22] Laurent Meunier, Blaise J Delattre, Alexandre Araujo, and Alexandre Allauzen. A dynamical system perspective for Lipschitz neural networks. In *International Conference on Machine Learning (ICML)*, 2022.
- [MKKY18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018.
- [MPG29] RV Mises and Hilda Pollaczek-Geiringer. Praktische verfahren der gleichungsaufloesung. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 1929.
- [Nak19] Preetum Nakkiran. Adversarial robustness may be at odds with simplicity. *arXiv preprint arXiv:1901.00532*, 2019.
- [NGBU22] Sebastian Neumayer, Alexis Goujon, Pakshal Bohra, and Michael Unser. Approximation of Lipschitz functions using deep spline neural networks. *arXiv preprint arXiv:2204.06233*, 2022.
- [PBBL24] Bernd Prach, Fabio Brau, Giorgio Buttazzo, and Christoph H. Lampert. 1-Lipschitz layers compared: Memory speed and certifiable robustness. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [Pea01] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1901.
- [PL22] Bernd Prach and Christoph H. Lampert. Almost-orthogonal layers for efficient general-purpose Lipschitz networks. In *European Conference on Computer Vision (ECCV)*, 2022. Reproduced with permission from Springer Nature.

- [PL23] Bernd Prach and Christoph H. Lampert. 1-Lipschitz neural networks are more expressive with N-activations. *arXiv preprint arXiv:2311.06103*, 2023.
- [PL25] Bernd Prach and Christoph H. Lampert. Intriguing properties of robust classification. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2025.
- [Pra24] Bernd Prach. SimpleConvNet. <https://github.com/berndprach/SimpleConvNet>, 2024.
- [QCSSL09] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil Lawrence. *Dataset Shift in Machine Learning*. MIT Press, 2009.
- [RXY⁺19] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John Duchi, and Percy Liang. Adversarial training can hurt generalization. In *ICML Workshop on Identifying and Understanding Deep Learning Phenomena*, 2019.
- [RXY⁺20] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John Duchi, and Percy Liang. Understanding and mitigating the tradeoff between robustness and accuracy. In *International Conference on Machine Learning (ICML)*, 2020.
- [SF21a] Sahil Singla and Soheil Feizi. Fantastic four: Differentiable bounds on singular values of convolution layers. In *International Conference on Learning Representations (ICLR)*, 2021.
- [SF21b] Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *International Conference on Machine Learning (ICML)*, 2021.
- [SF22] Sahil Singla and Soheil Feizi. Improved techniques for deterministic l_2 robustness. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
- [SSF21] Sahil Singla, Surbhi Singla, and Soheil Feizi. Improved deterministic l_2 robustness on CIFAR-10 and CIFAR-100. In *International Conference on Learning Representations (ICLR)*, 2021.
- [SST⁺18] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.
- [ST19] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, 2019.
- [SZS⁺14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.
- [TK21] Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the Cayley transform. In *International Conference on Learning Representations (ICLR)*, 2021.

- [TK22] Asher Trockman and J Zico Kolter. Patches are all you need? *Transactions on Machine Learning Research*, 2022.
- [TSE⁺19] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations (ICLR)*, 2019.
- [TSS18] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.
- [WCCY20] Jiayun Wang, Yubei Chen, Rudrasis Chakraborty, and Stella X. Yu. Orthogonal convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [Win22] Johan Sokrates Wind. 94% on CIFAR-10 in 94 lines and 94 seconds. https://johanwind.github.io/2022/12/28/cifar_94.html, 2022. Accessed: 2024-11-12.
- [WPD⁺23] Zekai Wang, Tianyu Pang, Chao Du, Min Lin, Weiwei Liu, and Shuicheng Yan. Better diffusion models further improve adversarial training. In *International Conference on Machine Learning (ICML)*, 2023.
- [XBSD⁺18] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning (ICML)*, 2018.
- [XLL22] Xiaojun Xu, Linyi Li, and Bo Li. LOT: Layer-wise orthogonal training on improving ℓ_2 certified robustness. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [ZYJ⁺19] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning (ICML)*, 2019.

Full Results for 1-Lipschitz Layer Comparison

In this section we report the full experimental results. In addition to the training budget of $24h$ used in the main chapter, we also report results for $2h$ and $10h$. See Table A.1 (CIFAR-10), Table A.2 (CIFAR-100), Table A.3 (Tiny ImageNet) and Table A.4 (Imagenette). Each of those tables also reports the best learning rate (LR) and weight decay (WD) we found for each setting. We also show a comparison of the model accuracy of different models in Figure A.1.

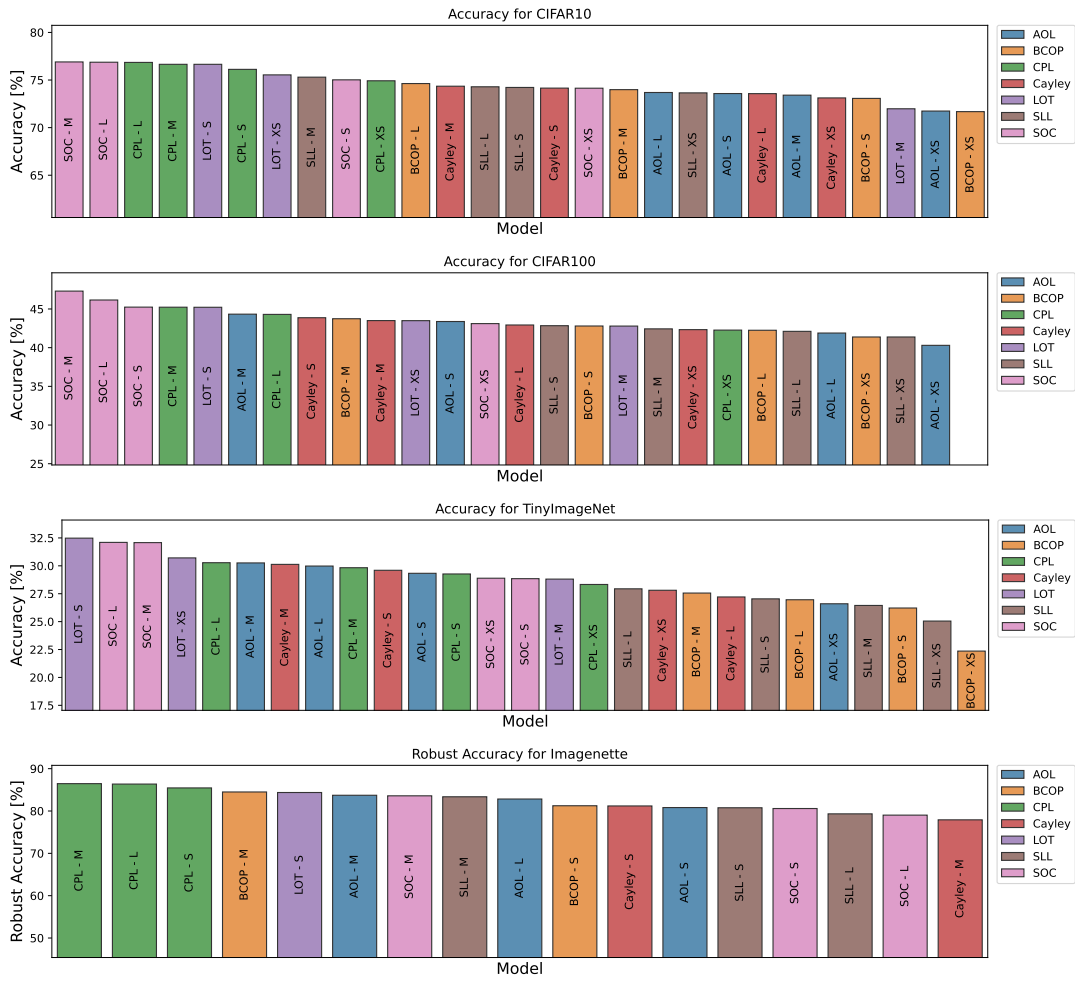


Figure A.1: **Comparison of model accuracy in decreasing order.** Note that the axes do not start at 0. © 2024 IEEE.

Table A.1: Full comparison result on CIFAR-10. © 2024 IEEE.

Layer	Model	LR	WD	Accuracy (%)			Robust Accuracy (%)		
				2h	10h	24h	2h	10h	24h
AOL	XS	$6 \cdot 10^{-2}$	$4 \cdot 10^{-5}$	68.5	71.7	71.7	55.6	58.8	59.1
	S	$1 \cdot 10^{-2}$	$5 \cdot 10^{-5}$	70.9	73.0	73.6	57.9	61.0	60.8
	M	$2 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	70.3	73.4	73.4	57.0	60.7	61.0
	L	$2 \cdot 10^{-2}$	$7 \cdot 10^{-5}$	64.7	71.8	73.7	51.4	59.0	61.5
BCOP	XS	$7 \cdot 10^{-3}$	$3 \cdot 10^{-4}$	69.0	70.8	71.7	55.0	57.6	58.5
	S	$4 \cdot 10^{-3}$	$8 \cdot 10^{-5}$	70.6	72.5	73.1	57.5	59.1	59.3
	M	$6 \cdot 10^{-3}$	$7 \cdot 10^{-6}$	71.8	73.6	74.0	57.9	59.9	60.5
	L	$1 \cdot 10^{-3}$	$9 \cdot 10^{-6}$	67.6	73.2	74.6	52.5	59.7	61.5
CPL	XS	$3 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	71.7	74.3	74.9	58.2	61.7	62.5
	S	$7 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	73.9	74.1	76.1	61.1	61.1	64.2
	M	$8 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	74.3	76.4	76.6	61.6	64.7	65.1
	L	$5 \cdot 10^{-2}$	$3 \cdot 10^{-4}$	72.6	76.5	76.8	59.1	64.5	65.2
Cayley	XS	$2 \cdot 10^{-2}$	$2 \cdot 10^{-5}$	71.3	73.2	73.1	57.2	59.4	59.5
	S	$1 \cdot 10^{-2}$	$1 \cdot 10^{-5}$	71.9	73.6	74.2	58.1	60.0	61.1
	M	$1 \cdot 10^{-2}$	$6 \cdot 10^{-6}$	70.2	73.5	74.4	55.8	60.2	61.0
	L	$8 \cdot 10^{-3}$	$2 \cdot 10^{-4}$	61.3	71.1	73.6	45.9	57.0	60.1
LOT	XS	$8 \cdot 10^{-2}$	$4 \cdot 10^{-6}$	73.5	75.2	75.5	59.6	62.7	63.4
	S	$3 \cdot 10^{-2}$	$7 \cdot 10^{-5}$	70.5	75.0	76.6	56.4	62.3	64.6
	M	$2 \cdot 10^{-2}$	$9 \cdot 10^{-6}$	61.4	69.3	72.0	45.7	54.7	58.7
SLL	XS	$9 \cdot 10^{-2}$	$2 \cdot 10^{-5}$	69.9	73.1	73.7	56.4	59.9	61.0
	S	$4 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	72.9	74.0	74.2	59.8	61.4	62.0
	M	$9 \cdot 10^{-2}$	$9 \cdot 10^{-5}$	70.5	74.4	75.3	57.4	61.5	62.8
	L	$6 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	56.7	72.7	74.3	38.9	60.0	62.3
SOC	XS	$5 \cdot 10^{-2}$	$7 \cdot 10^{-6}$	68.9	72.9	74.1	55.1	60.0	61.3
	S	$2 \cdot 10^{-2}$	$2 \cdot 10^{-5}$	67.3	73.3	75.0	53.2	60.8	62.9
	M	$7 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	73.1	77.0	76.9	60.3	66.0	66.3
	L	$2 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	62.3	73.8	76.9	46.2	60.8	65.4

Table A.2: Full comparison result on CIFAR-100. © 2024 IEEE.

Layer	Model	LR	WD	Accuracy (%)			Robust Accuracy (%)		
				2h	10h	24h	2h	10h	24h
AOL	XS	$3 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	37.9	40.1	40.3	26.6	28.0	27.9
	S	$2 \cdot 10^{-2}$	$2 \cdot 10^{-5}$	40.5	43.4	43.4	29.0	30.8	31.0
	M	$7 \cdot 10^{-2}$	$6 \cdot 10^{-6}$	40.5	43.5	44.3	28.4	31.1	31.4
	L	$6 \cdot 10^{-2}$	$4 \cdot 10^{-5}$	34.5	41.1	41.9	23.1	29.2	29.7
BCOP	XS	$8 \cdot 10^{-3}$	$3 \cdot 10^{-4}$	35.4	40.0	41.4	22.9	27.8	28.4
	S	$6 \cdot 10^{-3}$	$3 \cdot 10^{-4}$	37.8	42.1	42.8	24.5	29.5	30.1
	M	$2 \cdot 10^{-3}$	$2 \cdot 10^{-4}$	37.6	43.8	43.7	24.6	30.4	31.2
	L	$4 \cdot 10^{-3}$	$1 \cdot 10^{-5}$	29.2	40.3	42.2	17.3	27.2	29.2
CPL	XS	$9 \cdot 10^{-2}$	$3 \cdot 10^{-5}$	39.7	42.0	42.3	27.9	29.8	30.1
	S	$9 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	42.1	1.0	1.0	29.8	0.0	0.0
	M	$4 \cdot 10^{-2}$	$4 \cdot 10^{-5}$	40.5	44.5	45.2	27.4	32.4	33.2
	L	$9 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	40.1	43.9	44.3	27.3	31.5	32.1
Cayley	XS	$4 \cdot 10^{-2}$	$2 \cdot 10^{-5}$	41.1	41.6	42.3	27.9	29.2	29.2
	S	$2 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	42.3	43.1	43.9	28.8	30.4	30.5
	M	$6 \cdot 10^{-3}$	$4 \cdot 10^{-6}$	38.6	43.3	43.5	25.3	29.9	30.5
	L	$5 \cdot 10^{-3}$	$2 \cdot 10^{-5}$	26.3	40.3	42.9	14.3	27.0	29.5
LOT	XS	$6 \cdot 10^{-2}$	$3 \cdot 10^{-4}$	42.7	43.8	43.5	29.4	30.9	30.8
	S	$5 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	40.3	45.2	45.2	27.2	31.8	32.5
	M	$4 \cdot 10^{-2}$	$9 \cdot 10^{-5}$	28.4	38.9	42.8	15.5	25.9	29.6
SLL	XS	$5 \cdot 10^{-2}$	$6 \cdot 10^{-5}$	37.9	40.9	41.4	25.9	29.0	28.9
	S	$1 \cdot 10^{-1}$	$7 \cdot 10^{-5}$	40.0	41.9	42.8	28.4	29.9	30.5
	M	$7 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	39.3	41.9	42.4	27.0	30.0	29.9
	L	$9 \cdot 10^{-2}$	$8 \cdot 10^{-5}$	22.0	38.5	42.1	10.8	26.4	29.6
SOC	XS	$7 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	40.7	43.1	43.1	27.7	29.7	30.6
	S	$9 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	42.2	44.5	45.2	29.3	31.9	32.6
	M	$4 \cdot 10^{-2}$	$3 \cdot 10^{-4}$	41.8	46.2	47.3	28.8	33.5	34.9
	L	$7 \cdot 10^{-2}$	$3 \cdot 10^{-5}$	34.7	43.1	46.2	21.5	30.2	33.5

Table A.3: Full comparison result on Tiny ImageNet. © 2024 IEEE.

Layer	Model	LR	WD	Accuracy (%)			Robust Accuracy (%)		
				2h	10h	24h	2h	10h	24h
AOL	XS	$3 \cdot 10^{-2}$	$2 \cdot 10^{-5}$	24.5	26.9	26.6	16.5	18.4	18.1
	S	$7 \cdot 10^{-2}$	$5 \cdot 10^{-6}$	24.9	27.3	29.3	16.8	18.5	19.7
	M	$4 \cdot 10^{-2}$	$8 \cdot 10^{-6}$	26.2	29.5	30.3	18.1	20.4	21.0
	L	$2 \cdot 10^{-2}$	$8 \cdot 10^{-6}$	22.1	27.7	30.0	12.8	18.8	20.6
BCOP	XS	$6 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	11.7	20.4	22.4	4.7	11.6	13.8
	S	$7 \cdot 10^{-4}$	$4 \cdot 10^{-5}$	19.0	25.3	26.2	10.1	15.7	16.9
	M	$3 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	20.0	25.4	27.6	10.5	15.8	17.2
	L	$1 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	9.6	23.6	27.0	2.6	14.0	16.8
CPL	XS	$1 \cdot 10^{-1}$	$9 \cdot 10^{-6}$	26.5	27.5	28.3	16.3	17.5	18.9
	S	$4 \cdot 10^{-2}$	$3 \cdot 10^{-5}$	26.2	29.3	29.3	16.7	19.4	19.7
	M	$4 \cdot 10^{-2}$	$2 \cdot 10^{-5}$	26.3	29.4	29.8	16.7	19.5	20.3
	L	$6 \cdot 10^{-2}$	$1 \cdot 10^{-5}$	24.7	29.8	30.3	15.0	19.2	20.1
Cayley	XS	$8 \cdot 10^{-3}$	$8 \cdot 10^{-5}$	25.3	27.5	27.8	15.7	17.9	17.9
	S	$5 \cdot 10^{-3}$	$7 \cdot 10^{-5}$	25.6	29.3	29.6	15.8	19.1	19.5
	M	$4 \cdot 10^{-3}$	$4 \cdot 10^{-5}$	20.2	29.2	30.1	9.9	18.8	19.3
	L	$1 \cdot 10^{-3}$	$3 \cdot 10^{-4}$	5.7	22.1	27.2	0.9	11.9	16.7
LOT	XS	$3 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	28.1	31.1	30.7	18.2	20.4	20.8
	S	$5 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	27.1	32.0	32.5	16.3	21.6	21.9
	M	$2 \cdot 10^{-2}$	$6 \cdot 10^{-6}$	12.6	25.2	28.8	4.6	14.5	18.1
SLL	XS	$7 \cdot 10^{-2}$	$3 \cdot 10^{-4}$	24.2	25.3	25.1	14.9	16.7	16.6
	S	$4 \cdot 10^{-2}$	$2 \cdot 10^{-4}$	24.4	25.7	27.0	15.6	16.8	18.4
	M	$5 \cdot 10^{-2}$	$5 \cdot 10^{-5}$	17.0	26.2	26.5	7.8	17.0	17.7
	L	$7 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	11.4	26.2	27.9	2.9	16.7	18.8
SOC	XS	$9 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	25.5	28.7	28.9	15.7	18.7	18.9
	S	$7 \cdot 10^{-2}$	$2 \cdot 10^{-5}$	23.9	28.4	28.8	14.0	18.5	18.8
	M	$7 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	25.7	31.1	32.1	15.5	20.4	21.2
	L	$6 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	20.3	30.1	32.1	10.1	19.7	21.1

Table A.4: **Full comparison result on Imagenette.** © 2024 IEEE.

Layer	Model	LR	WD	Accuracy (%)			Robust Accuracy (%)		
				2h	10h	24h	2h	10h	24h
AOL	S	$9 \cdot 10^{-3}$	$1 \cdot 10^{-5}$	71.0	78.9	80.8	62.5	74.3	76.8
	M	$5 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	73.2	80.7	83.7	64.3	75.8	79.9
	L	$5 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	69.7	79.2	82.8	59.7	74.2	78.5
BCOP	S	$9 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	37.3	75.4	81.2	15.1	66.9	75.6
	M	$2 \cdot 10^{-3}$	$2 \cdot 10^{-4}$	36.4	77.8	84.5	15.6	71.3	80.1
	L	$7 \cdot 10^{-2}$	$4 \cdot 10^{-6}$	9.8	9.8	9.8	9.8	9.8	9.8
CPL	S	$9 \cdot 10^{-2}$	$5 \cdot 10^{-5}$	75.8	83.5	85.5	68.3	78.8	80.8
	M	$9 \cdot 10^{-2}$	$3 \cdot 10^{-5}$	79.5	84.8	86.5	73.5	80.3	82.4
	L	$7 \cdot 10^{-2}$	$3 \cdot 10^{-5}$	76.0	85.1	86.4	68.5	80.3	82.3
Cayley	S	$6 \cdot 10^{-4}$	$3 \cdot 10^{-5}$	60.9	78.0	81.2	45.4	71.5	75.8
	M	$1 \cdot 10^{-4}$	$1 \cdot 10^{-5}$	48.6	69.7	77.9	30.1	59.4	71.7
LOT	S	$7 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	66.2	80.7	84.4	55.7	75.6	79.6
SLL	S	$7 \cdot 10^{-2}$	$5 \cdot 10^{-6}$	70.6	77.6	80.8	60.6	70.1	75.4
	M	$7 \cdot 10^{-2}$	$6 \cdot 10^{-6}$	71.7	80.4	83.4	62.0	74.3	78.0
	L	$5 \cdot 10^{-2}$	$5 \cdot 10^{-6}$	64.7	75.4	79.3	51.0	67.2	72.8
SOC	S	$9 \cdot 10^{-3}$	$7 \cdot 10^{-5}$	60.7	77.2	80.6	45.1	69.7	74.7
	M	$2 \cdot 10^{-2}$	$9 \cdot 10^{-5}$	63.4	79.5	83.6	50.4	72.2	78.4
	L	$8 \cdot 10^{-3}$	$3 \cdot 10^{-4}$	52.5	71.7	79.0	34.1	62.2	73.5