1-Lipschitz Layers Compared: Memory, Speed, and Certifiable Robustness

Bernd Prach[1,*] Fabio Brau[2,*] Giorgio Buttazzo[2] Christoph H. Lampert[1]
[1] ISTA, Klosterneuburg, Austria
[2] Scuola Superiore Sant'Anna, Pisa, Italy
{bprach, chl}@ist.ac.at, {fabio.brau, giorgio.buttazzo}@santannapisa.it

Abstract

The robustness of neural networks against input perturbations with bounded magnitude represents a serious concern in the deployment of deep learning models in safety-critical systems. Recently, the scientific community has focused on enhancing certifiable robustness guarantees by crafting 1*-Lipschitz neural networks that leverage Lipschitz bounded dense and convolutional layers. Different methods have been proposed in the literature to achieve this goal, however, comparing the performance of such methods is not straightforward, since different metrics can be relevant (e.g., training time, memory usage, accuracy, certifiable robustness) for different applications. Therefore, this work provides a thorough comparison between different methods, covering theoretical aspects such as computational complexity and memory requirements, as well as empirical measurements of time per epoch, required memory, accuracy and certifiable robust accuracy. The paper also provides some guidelines and recommendations to support the user in selecting the methods that work best depending on the available resources. We provide code at* github.com/berndprach/1LipschitzLayersCompared.

1. Introduction

Modern artificial neural networks achieve high accuracy and sometimes superhuman performance in many different tasks, but it is widely recognized that they are not robust to tiny and imperceptible input perturbations [4, 39] that, if properly crafted, can cause a model to produce the wrong output. Such inputs, known as *Adversarial Examples*, represent a serious concern for the deployment of machine learning models in safety-critical systems [26]. To overcome this issue, *adversarial training* has been proposed in [14, 30, 39]. It uses adversarial examples during the training to correct the model prediction. This strategy does improves the empirical robustness of the model, however, it does not

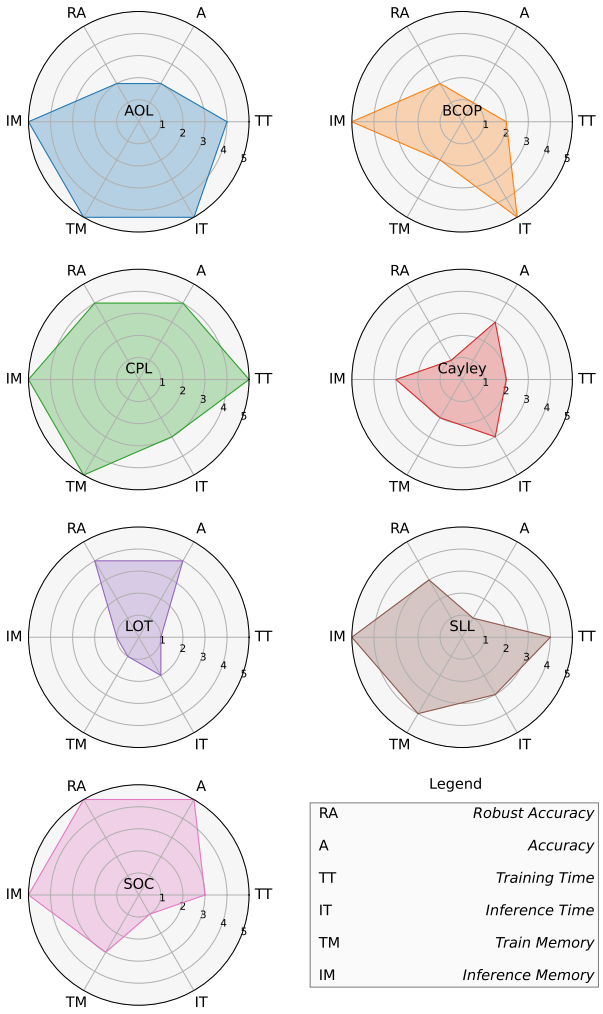

Figure 1. Evaluation of 1-Lipschitz methods on different metrics. Scores are assigned from 1 (worst) to 5 (best) to every method based on the results reported in Sections 3 and 5.

∗Joined first authors.

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

provide any guarantees of robustness.

However, for many application a *guarantee* of robustness is desired. Roughly speaking, a model f is said to be ε-robust for a given input x if no perturbation of magnitude bounded by ε can change its prediction. Recently, in the context of image classification, various approaches have been proposed to achieve certifiable robustness, including *Verification, Randomized Smoothing*, and *Lipschitz Bounded Neural Networks*.

Verification strategies aim to establish, for any given model, whether all samples contained in a l_2-ball with radius ε and centered in the tested input x are classified with the same class as x. In the exact formulation, verification strategies involve the solution of an NP-hard problem [20]. Nevertheless, even in a relaxed formulation, [44], these strategies require a huge computational effort [43].

Randomized smoothing strategies, initially presented in [10], represent an effective way of crafting a certifiable-robust classifier g based on a base classifier f. If combined with an additional denoising step, they can achieve state-of-the-art levels of robustness, [7]. However, since they require multiple evaluations of the base model (up to 100k evaluations) for the classification of a single input, they cannot be used for real-time applications.

Finally, Lipschitz Bounded Neural Networks [6, 9, 24, 27, 29, 34, 40] represent a valid alternative to produce certifiable classifiers, since they only require a single forward pass of the model at inference time to deduce guarantees of robustness. Indeed, for such models, a lower-bound of the minimal adversarial perturbation capable of fooling the classifier can be evaluated by considering the difference between the two largest class scores predicted by the model.

Lipschitz-bounded neural networks can be obtained by the composition of 1-Lipschitz layers [2]. The process of parameterizing 1-Lipschitz layers is fairly straightforward for fully connected layers. However, for convolutions — with overlapping kernels — deducing an effective parameterization is a hard problem. Indeed, the Lipschitz condition can be essentially thought of as a condition on the Jacobian of the layer. However, the Jacobian matrix can not be efficiently computed.

In order to avoid the explicit computation of the Jacobian, various methods have been proposed, including parameterizations that cause the Jacobian to be (very close to) orthogonal [27, 36, 40, 46] and methods that rely on an upper bound on the Jacobian instead [34]. Those different methods differ drastically in training and validation requirements (in particular time and memory) as well as empirical performance. Furthermore, increasing training time or model sizes very often also increases the empirical performance. This makes it hard to judge from the existing literature which methods are the most promising. This becomes even worse when working with specific computation requirements, such as restrictions on the available memory. In this case, it is important to choose the method that better suits the characteristics of the system in terms of evaluation time, memory usage as well and certifiable-robust-accuracy.

This work aims at giving a comprehensive comparison of different strategies for crafting 1-Lipschitz layers from both a theoretical and practical perspective. For the sake of fairness, we consider several metrics such as *Time* and *Memory* requirements for both training and inference, *Accuracy*, as well as *Certified Robust Accuracy*. The main contributions are the following:

- An empirical comparison of 1-Lipschitz layers based on six different metrics, and four different datasets on four architecture sizes with three time constraints.
- A theoretical comparison of the runtime complexity and the memory usage of existing methods.
- A review of the most recent methods in the literature, including implementations with a revised code that we will release publicly for other researchers to build on.

2. Existing Works and Background

In recent years, various methods have been proposed for creating artificial neural networks with a bounded Lipschitz constant. The *Lipschitz constant* of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with respect to the l_2 norm is the smallest L such that for all $x, y \in \mathbb{R}^n$

$$\|f(x) - f(y)\|_2 \leq L\|x - y\|_2. \tag{1}$$

We also extend this definition to networks and layers, by considering the l_2 norms of the flattened input and output tensors in Equation (1). A layer is called 1-Lipschitz if its Lipschitz constant is at most 1. For linear layers, the Lipschitz constant is equal to the *spectral norm* of the weight matrix that is given as

$$\|M\|_2 = \sup_{\mathbf{v} \neq 0} \frac{\|M\mathbf{v}\|_2}{\|\mathbf{v}\|_2}. \tag{2}$$

A particular class of linear 1-Lipschitz layers are ones with an orthogonal Jacobian matrix. The Jacobian matrix of a layer is the matrix of partial derivatives of the flattened outputs with respect to the flattened inputs. A matrix M is orthogonal if $MM^\top = I$, where I is the identity matrix. For layers with an orthogonal Jacobian, Equation (1) always holds with equality and, because of this, a lot of methods aim at constructing such 1-Lipschitz layers.

All the neural networks analyzed in this paper consist of 1-Lipschitz parameterized layers and 1-Lipschitz activation

functions, with no skip connections and no batch normalization. Even though the commonly used ReLU activation function is 1-Lipschitz, Anil *et al*. [2] showed that it reduces the expressive capability of the model. Hence, we adopt the MaxMin activation proposed by the authors and commonly used in 1-Lipschitz models. Concatenations of 1-Lipschitz functions are 1-Lipschitz, so the networks analyzed are 1-Lipschitz by construction.

2.1. Parameterized 1-Lipschitz Layers

This section provides an overview of the existing methods for providing 1-Lipschitz layers. We discuss fundamental methods for estimating the spectral norms of linear and convolutional layers, i.e. *Power Method* [32] and *Fantastic4* [35], and for crafting orthogonal matrices, i.e. Bjorck & Bowie [5], in Appendix A. The rest of this section describes 7 methods from the literature that construct 1-Lipschitz convolutions: BCOP, Cayley, SOC, AOL, LOT, CPL, and SLL. Further 1-Lipschitz methods, [19, 42, 47], and the reasons why they were not included in our main comparison can be found in Appendix B.

BCOP *Block Orthogonal Convolution Parameterization (BCOP)* was introduced by Li *et al*. in [27] to extend a previous work by Xiao *et al*. [45] that focused on the importance of orthogonal initialization of the weights. For a $k \times k$ convolution, BCOP uses a set of $(2k-1)$ parameter matrices. Each of these matrices is orthogonalized using the algorithm by Bjorck & Bowie [5] (see also Appendix A). Then, a $k \times k$ kernel is constructed from those matrices in a way that guarantees that the resulting layer is orthogonal.

Cayley Another family of orthogonal convolutional and fully connected layers has been proposed by Trockman and Kolter [40] by leveraging the *Cayley Transform* [8], which maps a skew-symmetric matrix A into an orthogonal matrix Q using the relation

$$Q = (I - A)(I + A)^{-1}. \tag{3}$$

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 Convolutions*, those matrices are orthogonalized using the Cayley transform.

SOC *Skew Orthogonal Convolution* is an orthogonal convolutional layer presented by Singla *et al*. [36], obtained by leveraging the exponential convolution [15]. Analogously to the matrix case, given a kernel $L \in \mathbb{R}^{c \times c \times k \times k}$, the exponential convolution can be defined as

$$\exp(L)(x) := x + \frac{L \star x}{1} + \frac{L \star^2 x}{2!} + \cdots + \frac{L \star^k x}{k!} + \cdots, \tag{4}$$

where $\star^k$ denotes a convolution applied k-times. The authors proved that any exponential convolution has an orthogonal Jacobian matrix as long as L 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 L is normalized to have unitary spectral norm following the method presented in [35] (see Appendix A).

AOL Prach and Lampert [34] introduced *Almost Orthogonal Lipschitz (AOL)* layers. For any matrix P, they defined a diagonal rescaling matrix D with

$$D_{ii} = \left(\sum_j \left| P^\top P \right|_{ij} \right)^{-1/2} \tag{5}$$

and proved that the spectral norm of PD is bounded by 1. This result was used to show that the linear layer given by $l(x) = PDx + b$ (where P is the learnable matrix and D is given by Eq. (5)) is 1-Lipschitz. Furthermore, the authors extended the idea so that it can also be efficiently applied to convolutions. This is done by calculating the rescaling in Equation (5) with the Jacobian J of a convolution instead of P. In order to evaluate it efficiently the authors express the elements of $J^\top J$ explicitly in terms of the kernel values.

LOT The layer presented by Xu *et al*. [46] extends the idea of [19] to use the *Inverse Square Root* of a matrix in order to orthogonalize it. Indeed, for any matrix V, the matrix $Q = V(V^T V)^{-\frac{1}{2}}$ is orthogonal. Similarly to the *Cayley* method, for the *layer-wise orthogonal training (LOT)* the convolution is applied in the Fourier frequency domain. To find the inverse square root, the authors relay on an iterative *Newton Method*. In details, defining $Y_0 = V^T V$, $Z_0 = I$, and

$$Y_{i+1} = \frac{1}{2} Y_i \left(3I - Z_i Y_i\right), \; Z_{i+1} = \frac{1}{2} \left(3I - Z_i Y_i\right) Z_i, \tag{6}$$

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

CPL Meunier *et al*. [31] proposed the *Convex Potential Layer*. Given a non-decreasing 1-Lipschitz function σ (usually ReLU), the layer is constructed as

$$l(x) = x - \frac{2}{\|W\|_2^2} W^\top \sigma(Wx + b), \tag{7}$$

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

SLL The *SDP-based Lipschitz Layers (SLL)* proposed by Araujo *et al*. [3] combine the CPL layer with the upper bound on the spectral norm from AOL. The layer can be written as

$$l(x) = x - 2W^\top Q^{-2} D^2 \sigma \left(Wx + b\right), \tag{8}$$

where Q is a learnable diagonal matrix with positive entries and D is deduced by applying Equation (5) to $P = W^\top Q^{-1}$.

Remark 1. 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.

3. Theoretical Comparison

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

Table 1 provides an overview of the computational complexity and memory requirements for the different layers considered in the previous section. For the sake of clarity, the analysis is performed by considering separately the transformations applied to the input of the layers and those applied to the weights to ensure the 1-Lipschitz constraint. Each of the two sides of the table contains three columns: i) *Operations* contains the most costly transformations applied to the input as well as to the parameters of different layers; ii) *MACS* reports the computational complexity expressed in multiply-accumulate operations (MACS) involved in the transformations (only leading terms are presented); iii) *Memory* reports 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 1, we reported them separately in Appendix C.1.

Note that all the terms reported in Table 1 depend on the batch size b, the input size $s \times s \times c$, the number of inner iterations of a method t, and the kernel size $k \times k$. (Often, t is different at training and inference time.) For the sake of clarity, the MACS of a naive convolution implementation is denoted by C ($C = bs^2c^2k^2$), the number of inputs of a layer is denoted by M ($M = bs^2c$), and the size of the kernel of a standard convolution is denoted by P ($P = c^2k^2$). Only the leading terms of the computations are reported in Table 1. In order to simplify some terms, we assume that $c > \log_2(s)$ and that rescaling a tensor (by a scalar) as well as adding two tensors does not require any memory in order to do backpropagation. We also assume that each additional activation does require extra memory. All these assumptions have been verified to hold within *PyTorch*, [33]. Also, when 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), Fast Fourier Transformations (FFT), matrix-vector multiplications (MV), matrix-matrix multiplications (MM), matrix inversions (INV), as well as applications of an activation function (ACT). The application of algorithms such as Bjorck & Bowie (BnB), power method, and Fantastic 4 (F4) is also reported (see Appendix A for descriptions).

3.1. Analysis of the computational complexity

It is worth noting that the complexity of the input transformations (in Table 1) is similar for all methods. This implies that a similar scaling behaviour is expected at inference time for the models. Cayley and LOT apply an FFT-based convolution and have computational complexity independent of the kernel size. CPL and SLL require two convolutions, which make them slightly more expensive at inference time. Notably, SOC requires multiple convolutions, making this method more expensive at inference time.

At training time, 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

Table 1. **Computational complexity and memory requirements of different methods.** We report multiply-accumulate operations (MACS) as well as memory requirements (per layer) for batch size b, image size $s \times s \times c$, kernel size $k \times k$ and number of inner iterations t. We use $C = bs^2c^2k^2$, $M = bs^2c$ and $P = c^2k^2$. For a detailed explanation on what is reported see Section 3. For some explanation on how the entries of this table were derived, see Appendix C.

Method	Input Transformations			Parameter Transformations		
	Operations	MACS $\mathcal{O}(\cdot)$	Memory	Operations	MACS $\mathcal{O}(\cdot)$	Memory $\mathcal{O}(\cdot)$
Standard	CONV	C	M	-	-	P
AOL	CONV	C	M	CONV	c^3k^4	$5P$
BCOP	CONV	C	M	BnB & MMs	$c^3kt + c^3k^3$	$c^2kt + c^2k^3$
Cayley	FFTs & MVs	bs^2c^2	$\frac{5}{2}M$	FFTs & INVs	s^2c^3	$\frac{3}{2}s^2c^2$
CPL	CONVs & ACT	$2C$	$3M$	power method	$s^2c^2k^2$	$P + s^2c$
LOT	FFTs & MVs	bs^2c^2	$3M$	FFTs & MMs	$4s^2c^3t$	$4s^2c^2t$
SLL	CONVs & ACT	$2C$	$3M$	CONVs	c^3k^4	$5P$
SOC	CONVs	Ct_1	Mt_1	F4	$c^2k^2t_2$	P

require parameter transformations that scale like c^3, making them more expensive for larger 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.

3.2. 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 $\mathcal{O}(P)$. Exceptions to this rule are Cayley and LOT, with a larger $\mathcal{O}(s^2c^2)$ term, as well as BCOP.

4. 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. For a detailed description see Appendix E. 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 1, different layers may have different throughput, to have 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.

Our architecture is a standard convolutional network that doubles the number of channels whenever the resolution is reduced [6, 40]. For each method, we tested architectures of different sizes. We denoted them as XS, S, M and L, depending on the number of parameters, according to the criteria in Table 7, ranging from 1.5M to 100M parameters.

Since different methods benefit from different learning rates and weight decay, for each setting (model size, method and dataset), we used the best values resulting from a random search performed on multiple training runs on a validation set composed of 10% of the original training set. More specifically, 16 runs were performed for each configuration of randomly sampled hyperparameters, and we selected the configuration maximizing the certified robust accuracy w.r.t. $\epsilon = 36/255$ (see Appendix E.4 for details).

The evaluation was carried out using four different datasets: CIFAR-10, CIFAR-100 [21], Tiny ImageNet [23], and Imagenette [16] for large images. Augmentation was used during the training (Random crops and flips on CIFAR-10 and CIFAR-100, *RandAugment* [11] on Tiny ImageNet, and random crop as well as *RandAugment* on Imagenette), details in Appendix E.5. We use the loss function proposed by [34], with same temperature 0.25, and where we tuned the margin to maximize the robust accuracy for $\epsilon = \frac{36}{255}$. In detail, we considered a margin of $2\sqrt{2}\epsilon$ where

the $\sqrt{2}$ factor comes from the L_2 norm [41], and the factor 2 has been added to help with generalization.

4.1. 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 80GB-A100-GPU based on the average time of 100 mini-batches. We measured the inference throughput with cached 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 $\mathcal{O}(M+P)$. As for the throughput, we evaluated and cached all calculations independent of the input at inference time.

Certified robust accuracy In order to evaluate the performance of a 1-Lipschitz network, the standard metric is the *certified robust accuracy*. An input is classified certifiably robustly with radius ϵ by a model, if no perturbations of the input with norm bounded by ϵ can change the prediction of the model. Certified robust accuracy measures the proportion of examples that are classified correctly as well as certifiably robustly. For 1-Lipschitz models, a lower bound of the certified ϵ-robust accuracy is the portion of correctly classified inputs such that $\mathcal{M}_f(x_i, l_i) > \epsilon\sqrt{2}$ where the *margin* $\mathcal{M}_f(x, l)$ of a model f at input x with label l, given as $\mathcal{M}_f(x, l) = f(x)_l - \max_{j \neq l} f_j(x)$, is the difference between target class score and the highest score of a different class. For details, see [41].

5. Experimental Results

This section presents the results of the comparison performed by applying the methodology discussed in Section 4. 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 1.

5.1. Training and inference times

Figure 2 plots the training time per epoch of the different models as a function of their size, and Figure 3 plots the corresponding inference throughput for the various sizes as described in Section 4. As described in Table 5, the model base width, referred to as w, is doubled from one model size to the next. We expect the training and inference time to scale with w similarly to how individual layers scale with their number of channels, c (in Table 1). This is because the width of each of the 5 blocks of our architecture is a constant multiple of the base width, w.

The training time increases (at most) about linearly with w for standard convolutions, whereas the computational complexity of each single convolution scales like c^2. This suggests that parallelism on the GPU and 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

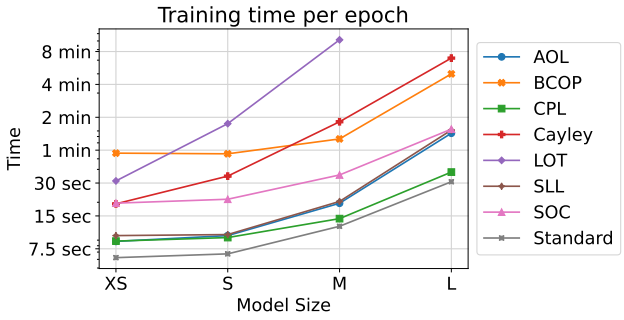

Figure 2. **Training time** per epoch (on CIFAR-10) for different methods and different model sizes.

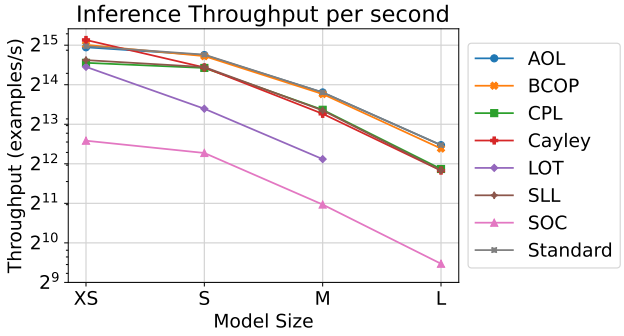

Figure 3. **Inference throughput** for different methods as a function of their size for CIFAR-10 sizes input images. All parameter transformations have been evaluated and cached beforehand

convolution, and SOC (doing 5 convolutions) is only about 3 times slower than the standard convolution. The AOL and SLL methods also require times comparable to a standard convolution for small models, although eventually, 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 1 for further details.

At inference time transformations of the weights are cached, therefore some methods (AOL, BCOP) do not have any overhead 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. From Figure 3, it can be noted that cached 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, *i.e.*, *Winograd*, [22], that can be up to 2.5 times faster.

5.2. Training memory requirements

The training and inference memory requirements of the various models (measured as described in Section 4.1) are reported in Figure 4 as a function of the model size. The results of the theoretical analysis reported in Table 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. For some of the models, the memory required approximately doubles from one model size to the next one, just like the width. 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 in the c^2 term (for LOT and Cayley the coefficient is even dependent on the input size, s^2) 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.

Note that 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, at inference time, most methods require a very similar

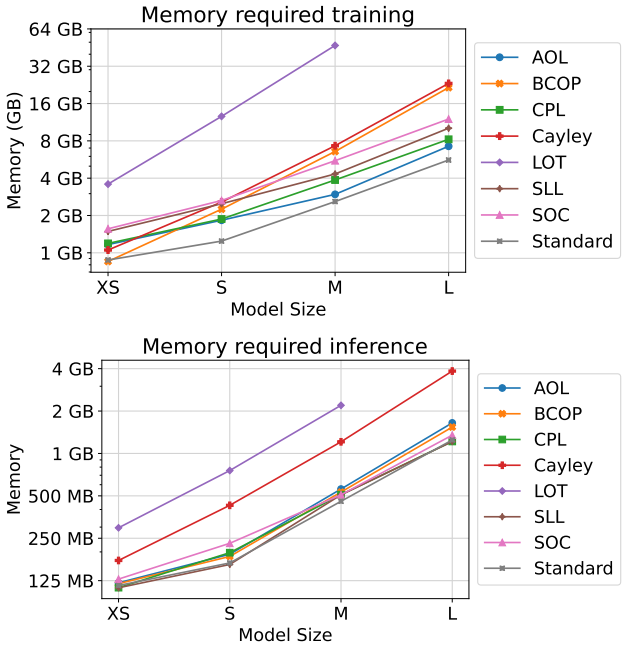

Figure 4. **Memory required** at training and inference time for input size 32×32.

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

5.3. Certified robust accuracy

The results related to the accuracy and the certified robust accuracy for the different methods, model sizes, and datasets measured on a 24h training budget are summarized in Table 2. The differences among the various model sizes are also highlighted in Figure 6 by reporting the sorted values of the certified robust accuracy. Further tables and plots relative to different training budgets can be found in Appendix G. The reader can compare our results with the state-of-the-art certified robust accuracy summarized in Appendix D. However, it is worth noting that, to reach state-of-the-art performance, authors often carry out experiments using large model sizes and long training times, which makes it hard to compare the methods themselves. On the other hand, the evaluation proposed in this paper allows a fairer comparison among the different methods, since it also considers timing and memory aspects. This restriction based on time, rather than the number of epochs, ensures that merely enlarging the model size does not lead to improved performance, as bigger models typically process fewer epochs of data. Indeed, in our results in Figure 6 it is usually the M (and not the L) model that performs best.

Experiments show that SOC performs best, reaching the

Table 2. Certified robust accuracy for radius $\epsilon = 36/255$ on the evaluated datasets. Training is performed for 24 hours.

Methods	Accuracy [%] XS	S	M	L	Robust Accuracy [%] 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	-
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

highest certified robust accuracy on two datasets. CPL models consistently rank in top-10 position among the three datasets. LOT performed well, in particular on Tiny ImageNet dataset 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 strongly depend on the number of classes. BCOP only reach the top-10 once, while Cayley is consistently outperformed by the other methods. The very same analysis can be applied to the clean accuracy, whose sorted bar-plots are reported in Appendix G, where the main difference is that Cayley performs slightly better for that metric. Furthermore, it is worth highlighting that CPL is sensitive to weight initialization. We faced numerical errors during the 10h and 24h training of the small model on CIFAR-100. On Imagenette, CPL clearly performs best, followed by BCOP and AOL. Note that these methods all construct a kernel so that the convolution is 1-Lipschitz. This seems to be good strategy for higher resolution datasets. E.g. SOC, that instead applies multiple convolutions has a drop in performs compared to other datasets.

5.3.1 Interpretation of the results

We confirm empirically what suspected in [46]: 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 with an identity initialization (AOL, LOT) perform better than layers that do neither (BCOP, Cayley). Presumably this is due to the MaxMin activation reducing the variance in the forward pass when alternated with non-identity layers.

Our results also allow ruling out some other possible explanation: one might suspect that pure contractive layers (AOL, CPL, and SLL) would suffer from vanishing gradients, differently from orthogonal ones, however, our experiments do not show any evidence of this fact. Furthermore, one might suspect that slower methods perform worse, because they allow fewer epochs for a given time budget, however, our experiments do not support this fact; two relative slow methods (SOC, LOT) are among the best ones.

6. Conclusions and Guidelines

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

Taking all metrics into account (summarized in Figure 1), the results are in favor of CPL, due to its highest performance and lower consumption of computational resources. When large computational resources are available and the application does not impose stringent timing constraints during inference and training, the SOC layer could be used, due to its slightly better performance. Finally, those 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. For higher resolution images, it also seems that CPL is the most promising method.

References

[1] 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. 14, 22

[2] Cem Anil, James Lucas, and Roger Grosse. Sorting out Lipschitz function approximation. In *International Conference on Machine Learing (ICML)*, 2019. 2, 3, 11

[3] 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. 4, 14, 22

[4] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases*, 2013. 1

[5] Å. Björck and C. Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 1971. 3, 11

[6] Fabio Brau, Giulio Rossolini, Alessandro Biondi, and Giorgio Buttazzo. Robust-by-design classification via unitary-gradient neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023. 2, 5

[7] Nicholas Carlini, Florian Tramer, Krishnamurthy Dj Dvijotham, Leslie Rice, Mingjie Sun, and J Zico Kolter. (Certified!!) adversarial robustness for free! In *International Conference on Learning Representations (ICLR)*, 2023. 2

[8] 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. 3

[9] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International conference on machine learning*, 2017. 2, 11

[10] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. 2

[11] 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. 5, 16

[12] 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. 16

[13] Farzan Farnia, Jesse Zhang, and David Tse. Generalizable adversarial training via spectral normalization. In *International Conference on Learning Representations*, 2018. 11

[14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *stat*, 2015. 1

[15] 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. 3

[16] Jeremy Howard. Imagenette. https://github.com/fastai/imagenette/. Accessed: 01.02.2024. 5, 16

[17] Kai Hu, Klas Leino, Zifan Wang, and Matt Fredrikson. Effectively leveraging capacity for improved deterministic robustness certification. In *International Conference on Learning Representations (ICLR)*, 2024. 14, 22

[18] Kai Hu, Andy Zou, Zifan Wang, Klas Leino, and Matt Fredrikson. Unlocking deterministic robustness certification on imagenet. *Conference on Neural Information Processing Systems (NeurIPS)*, 2024. 14

[19] 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. 3, 11, 12

[20] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International conference on computer aided verification*, 2017. 2

[21] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 5, 16

[22] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 7

[23] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 2015. 5, 16

[24] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning*, 2021. 2, 11

[25] Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learing (ICML)*, 2019. 11

[26] Linyi Li, Tao Xie, and Bo Li. Sok: Certified robustness for deep neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*, 2023. 1

[27] 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. 2, 3, 11, 14

[28] Shuai Li, Kui Jia, Yuxin Wen, Tongliang Liu, and Dacheng Tao. Orthogonal deep neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 11

[29] Max Losch, David Stutz, Bernt Schiele, and Mario Fritz. Certified robust models with slack control and large Lipschitz constants. *arXiv preprint arXiv:2309.06166*, 2023. 2

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018. 1

[31] Laurent Meunier, Blaise J Delattre, Alexandre Araujo, and Alexandre Allauzen. A dynamical system perspective for Lipschitz neural networks. In *International Conference on Machine Learing (ICML)*, 2022. 3, 11, 14, 22

[32] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018. 3, 11

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Conference on Neural Information Processing Systems (NeurIPS)*. 2019. 4

[34] Bernd Prach and Christoph H Lampert. Almost-orthogonal layers for efficient general-purpose Lipschitz networks. In *European Conference on Computer Vision (ECCV)*, 2022. 2, 3, 5, 14

[35] S Singla and S Feizi. Fantastic four: Differentiable bounds on singular values of convolution layers. In *International Conference on Learning Representations (ICLR)*, 2021. 3, 11

[36] Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *International Conference on Machine Learing (ICML)*, 2021. 2, 3, 14

[37] Sahil Singla and Soheil Feizi. Improved techniques for deterministic l2 robustness. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022. 14

[38] 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. 15

[39] 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. 1

[40] Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the Cayley transform. In *International Conference on Learning Representations (ICLR)*, 2021. 2, 3, 5, 14, 23

[41] 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. 6

[42] Ruigang Wang and Ian Manchester. Direct parameterization of Lipschitz-bounded deep networks. In *International Conference on Machine Learing (ICML)*, 2023. 3, 11, 12

[43] Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learing (ICML)*, 2018. 2

[44] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learing (ICML)*, 2018. 2

[45] 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 Learing (ICML)*, 2018. 3

[46] Xiaojun Xu, Linyi Li, and Bo Li. Lot: Layer-wise orthogonal training on improving l2 certified robustness. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022. 2, 3, 8, 14, 23

[47] Tan Yu, Jun Li, Yunfeng Cai, and Ping Li. Constructing orthogonal convolutions in an explicit manner. In *International Conference on Learning Representations (ICLR)*, 2021. 3, 11, 12