Spectrum Extraction and Clipping for Implicitly Linear Layers

Ali Ebrahimpour Boroojeny
University of Illinois,
Urbana-Champaign

Matus Telgarsky
New York University

Hari Sundaram
University of Illinois,
Urbana-Champaign

Abstract

We show the effectiveness of automatic differentiation in efficiently and correctly computing and controlling the spectrum of *implicitly* linear operators, a rich family of layer types including all standard convolutional and dense layers. We provide the first clipping method which is correct for general convolution layers, and illuminate the representational limitation that caused correctness issues in prior work. We study the effect of the batch normalization layers when concatenated with convolutional layers and show how our clipping method can be applied to their composition. By comparing the accuracy and performance of our algorithms to the state-of-the-art methods, using various experiments, we show they are more precise and efficient and lead to better generalization and adversarial robustness. We provide the code for using our methods at `https://github.com/Ali-E/FastClip`.

1 INTRODUCTION

Implicitly linear layers are key components of deep learning models and include any layer whose output can be written as an affine function of their input. This affine function might be trivial, such as a dense layer, or non-trivial, such as convolutional layers. These layers inherit appealing properties of linear transformations; not only are they flexible and easy to train, but also they have the same Jacobian as the transformation that the layer represents and a Lipschitz constant equal to its largest singular value. Therefore controlling the largest singular value of these layers, which is the same as the largest singular value of their Jacobians, not only contributes to the generalization of the model (Bartlett et al., 2017), but makes the model more robust to adversarial perturbations (Szegedy et al., 2013; Weng et al., 2018), and prevents the gradients from exploding or vanishing during backpropagation. Although efficient algorithms have been introduced to bound the spectral norm of dense layers (Miyato et al., 2018), computing and bounding them efficiently and correctly has been a challenge for the general family of *implicitly* linear layers, such as convolutional layers.

Convolutional layers are a major class of implicitly linear layers that are used in many models in various domains. They are compressed forms of linear transformations with an effective rank that depends on the dimensions of input rather than their filters. A 2d-convolutional layer with a stride of 1 and zero padding, whose kernel has dimensions (c_{out}, c_{in}, k, k) and is applied to an input of size $n \times n$ represents a linear transformation of rank $\min(c_{in}, c_{out})n^2$. This compression is beneficial for training large and deep models as it reduces the number of parameters drastically while keeping the flexibility of performing the higher-rank transformation. Nevertheless, this compression makes it challenging to compute and control the spectrum. Clearly, a straightforward way of computing the spectrum of the convolutional layers is to unroll them to build the explicit matrix form of the transformation and compute its singular values. In addition to the complexity of computing the matrix representation for various padding types and sizes and different strides, this procedure is very slow, as represented in prior work (Sedghi et al., 2018), and forfeits the initial motivation for using them. The problem becomes much more challenging if one decides to bound the spectral norms of these layers during training without heavy computations that make this regularization impractical.

There has been an ongoing effort to design methods for controlling the spectrum of convolutional layers in either trained models or during the training; however, neither of these methods work correctly for all stan-

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238. Copyright 2024 by the author(s).

dard convolutional layers. Moreover, they are either computationally extensive or rely on heuristics. In this work, we study both problems of extracting the spectrum and controlling them for the general family of implicitly linear layers. We give efficient algorithms for both tasks that scale to large models without incurring noticeable computational barriers. We also unveil some previously neglected limitations of convolutional layers in representing arbitrary spectrums. Up to this point, researchers have assumed they can modify the spectrum of convolutional layers in an arbitrary way. Although the experiments center around dense layers and convolutional layers, as we will discuss, our algorithms are general and do not assume properties that are specific to these layers. The only assumption is that their transformation can be represented as an affine function $f(x) = M_W x + b$, in which W represents the parameters of the layer, but the transformation matrix M_W is not necessarily explicit and might need further computations to be derived.

In detail, the contributions are as follows:

Efficient algorithm for extracting the spectrum: We use auto-differentiation to implicitly perform shifted subspace iteration algorithm on any implicitly linear layer (Section 2.1). Our method is correct for all convolutions and can be applied to the composition of layers as well. It is also more efficient than prior work for extracting top-k singular values (Section 3.1).

Studying the limitations of convolutional layers: We are the first to reveal the limitation of convolutional layers with circular padding in representing arbitrary spectrums (Section 2.3).

Fast and precise clipping algorithm: We present a novel algorithm for exact clipping of the spectral norm to arbitrary values for implicitly linear layers in an iterative manner (Section 2.2). Unlike prior methods, our algorithm works for any standard convolutional layer (Figure 1) and can be applied to the composition of layers (Section 2.4). We show the effectiveness of our method to enhance the generalization and robustness of the trained models in **Table 1**.

1.1 Related Work

Miyato et al. (2018) approximate the original transformation by reshaping the kernel to a $n_{in}k^2 \times n_{out}$ matrix and perform the power-iteration they designed for the dense layers to compute the spectral norm of the convolutional layers. To clip the spectral norm, they simply scale all the parameters to get the desired value for the largest singular value. Farnia et al. (2018) and Gouk et al. (2021) use the transposed convolution layer to perform a similar power iteration method to the one introduced by Miyato et al. (2018) for dense layers, and then perform the same scaling of the whole spectrum. Virmaux & Scaman (2018) perform the power method implicitly using the automatic differentiation that is similar to our implicit approach; however, they do not provide any algorithm for clipping the spectral norms and leave that for future work. Also, for extracting multiple singular values, they use successive runs of their algorithm followed by deflation, which is much less efficient than our extraction method. Some other works consider additional constraints. Cisse et al. (2017) constrained the weight matrices to be orthonormal and performed the optimization on the manifold of orthogonal matrices. Sedghi et al. (2018) gave a solution that works only for convolutional layers with circular padding and no stride. For other types of convolutional layers, they approximate the spectral norm by considering the circular variants; however, the approximation error can be large for these methods. (Senderovich et al., 2022) extends the method of Sedghi et al. (2018) to support convolutional layers with strides. It also provides an optional increase in speed and memory efficiency, albeit with the cost of expressiveness of convolutions, through specific compressions. Delattre et al. (2023) use Gram iteration to derive an upper-bound on the spectral norm of the circular approximation in a more efficient way. However, they do not provide a solution for strides other than 1. Therefore, in the best case, they will have the same approximation error as (Sedghi et al., 2018). A parallel line of research on controlling the spectrum of linear layers seeks to satisfy an additional property, gradient norm preserving, in addition to making each of the dense and convolutional layers 1-Lipschitz Anil et al. (2019); Singla & Feizi (2021); Yu et al. (2021); Trockman & Kolter (2021); Singla et al. (2021); Xu et al. (2022); Achour et al. (2022). Some state-of-the-art methods in this line of work are also based on the clipping method introduced by Sedghi et al. (2018) (Yu et al., 2021; Xu et al., 2022). Therefore, those approaches are even slower and less efficient.

2 METHODS

This section is organized as follows. We introduce our algorithm, PowerQR, for extracting top-k singular values and vectors of any implicitly linear layer or their composition in Section 2.1. In Section 2.2, we introduce our accurate and efficient algorithm, FastClip, for clipping the spectral norm of implicitly linear layers, and explain how it can be efficiently used with the PowerQR algorithm during training. In Section 2.3, we establish the inability of convolutions to attain any arbitrary spectrum. The reader can find the proofs in Appendix A. Next, we introduce notation.

Notation. An implicitly linear layer can be written as an affine function $f(x) = M_W x + b$, where $x \in \mathbb{R}^n$, $M_W \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$. M_W is not necessarily the explicit form of the parameters of the layer, in which case the explicit form is shown with W (e.g., the kernel of convolutional layers); otherwise, M_W is the same as W. The spectral norm of M_W, which is the largest singular value of M_W, is shown with $\|M_W\|_2$. $\sigma_i(W)$ is used to represent the i-th largest singular value of matrix M_W. The largest singular value might be referred to as either $\|W\|_2$ or $\|M_W\|_2$. We use $\omega = \exp(2\pi i/n)$ (the basic n-th roots of unity), where n is the rank of the linear transformation. $\Re(.)$ returns the real part of its input, and we also define the symmetric matrix A_W as $M_W^\top M_W$. Hereinafter $[k]$ will be used to show the set $\{0, 1, 2, \ldots, k\}$. When studying the composition of affine functions, we use the word "concatenation" to refer to the architecture of the network (i.e., succession of the layers), and "composition" to refer to the mathematical form of this concatenation.

2.1 Spectrum Extraction

We introduce our *PowerQR* algorithm, which performs an implicit version of the shifted subspace iteration algorithm on any implicitly linear layer. Shifted subspace iteration is a common method for extracting the spectrum of linear transformations; the shift parameter makes the algorithm more stable and faster than the regular power method. Also, it is much more efficient when multiple singular values and vectors are of interest, compared to iterative calls to the regular power method followed by deflation (see chapter 5 of Saad (2011) for more details of this algorithm). However, the direct application of this algorithm requires the explicit matrix form of the transformation. In Algorithm 1, we show how auto-differentiation can be used to perform this algorithm in an implicit way, without requiring the explicit matrix form. Regarding the complexity of Algorithm 1, other than the $O(n^2k)$ complexity of QR decomposition in line 6, it has an additional cost of computing the gradients of the layer at line 4 for a batch of k data points, which is dominated by the former cost.

Algorithm 1 PowerQR $(f, X, N, \mu = 1)$

```
1: **Input:** Affine function $f$, Initial matrix $X \in \mathbb{R}^{n\times k}$, Number of iterations $N$, Shift value $\mu$
2: **Output:** Top $k$ singular values and corresponding right singular vectors
3: **for** $i = 1$ **to** $N$ **do**
4:     $X' \leftarrow \nabla_X \frac{1}{2}\|f(X) - f(0)\|^2$
           // $\nabla_x \frac{1}{2}\|f(x) - f(0)\|^2 = M^\top M x$
5:     $X \leftarrow \mu X + X'$
6:     $(Q, R) \leftarrow \mathrm{QR}(X)$   // QR *decomposition of* $X$
7:     $X \leftarrow Q$
8: **end for**
9: $S \leftarrow \sqrt{\mathrm{diag}(R - \mu I)}$   // *Singular values*
10: $V \leftarrow X$   // *Right singular vectors*
11: **Return** $S, V$
```

Proposition 2.1. *Let $f(x) = Mx + b$. Then Algorithm 1 correctly performs the shifted subspace iteration algorithm on M, with μ as the shift value.*

Note that by subtracting the value of f at 0 in line 4, we remove the bias term from the affine function, which does not affect the singular values and vectors. This technique works even for the composition of multiple affine functions (e.g., concatenation of implicitly linear layers). The convergence rate in the subspace iteration algorithm depends on the initial matrix X (line 1), and if there are vectors s_i in the column space of X for which $\|s_i - v_i\|_2$ is small, then it takes fewer steps for the i-th eigenvector to converge. Meanwhile, we know that SGD makes small updates to W in each training step. Therefore, by reusing the matrix V in line 10 (top-k right singular vectors) computed for W_{i-1} as initial X for computing the spectrum of W_i, we can get much faster convergence. Our experiments show that by using this technique, it is enough to perform only one iteration of PowerQR per SGD step to converge to the spectrum of W after a few steps. Using a warm start (performing more iterations of PowerQR for the initial W) helps to correctly follow the top-k singular values during the training. The prior work on estimating and clipping the singular values has exploited these small SGD updates for the parameters in a similar manner (Farnia et al., 2018; Gouk et al., 2021; Senderovich et al., 2022).

2.2 Clipping the Spectral Norm

In this section, we introduce our algorithm for clipping the spectral norm of an implicitly linear layer to a specific target value. To project a linear operator $M = USV^\top$ to the space of operators with a bounded spectral norm of c, it is enough to construct $A' = US'V^\top$, where $S'_{i,i} = \min(S_{i,i}, c)$ (Lefkimmiatis et al., 2013). Therefore, for this projection, we will not need to extract and modify the whole spectrum of A (as in (Sedghi et al., 2018; Senderovich et al., 2022)), and only clipping the singular values that are larger than c to be exactly c would be enough. Note that after computing the largest spectral norm (e.g., using the PowerQR method) by simply dividing the parameters of the affine model by the largest singular value, the desired bound on the spectral norm would be achieved, and that is the basis of some prior work (Miyato et al., 2018; Farnia et al., 2018; Gouk et al., 2021); however, this procedure scales the whole spectrum rather than

Algorithm 2 Clip (f_W, c, N, P)

```
1:  Input: Affine function $f_W = M_W x + b$, Clip value $c$, Number of iterations $N$, Learning rate $\lambda$, Number of iterations of PowerQR $P$
2:  Output: Affine function $f_{W'}$ with singular values clipped to $c$
3:  $\sigma_1, v_1 \leftarrow$ PowerQR$(f_W, v, P)$ ($v$: Random vector)
4:  while $\sigma_1 > c$ do
5:      $W' \leftarrow W$                                  // $W' = \sum_{i=1}^{n} u_i \sigma_i v_i^\top$
6:      for $i = 1$ to $N$ do
7:          $W'_\delta \leftarrow \nabla_{W'} \frac{1}{2} \|f_{W'}(v_1) - f_W(c\sigma_1^{-1} v_1)\|^2$
                                                           // $W'_\delta = u_1(\sigma_1 - c)v_1^\top$
8:          $W' \leftarrow W' - \lambda W'_\delta$
9:      end for
10:     $W \leftarrow W'$
11:     $\sigma_1, v_1 \leftarrow$ PowerQR$(f_W, v, P)$ ($v$: Random vector)   // Update $\sigma_1$ and $v_1$
12: end while
13: Return $f_{W'}, \sigma_1, v_1$
```

projecting it to a norm ball. Therefore, it might result in suboptimal optimization of the network due to replacing the layer with one in the norm ball that behaves very differently. This adverse effect can be seen in Table 1 as well. Thus, to resolve both aforementioned issues, our algorithm iteratively shrinks the largest singular value by substituting the corresponding rank 1 subspace in an implicit manner.

Algorithm 2 shows our stand-alone clipping method. The outer `while` loop clips the singular values of the linear operator one by one. After clipping each singular value, the PowerQR method in line `11` computes the new largest singular value and vector, and the next iteration of the loop performs the clipping on them. The clipping of a singular value is done by the `for` loop. Considering that in the `for` loop $M := M_W = M_{W'} = \sum_{i=1}^{n} u_i \sigma_i v_i^\top$, since v_is are orthogonal and $v_i^\top v_i = 1$, for line `7` we have:

$$\begin{aligned} W'_\delta &= \nabla_{W'} \frac{1}{2} \|f_{W'}(v_1) - f_W(c\sigma_1^{-1} v_1)\|^2 \\ &= \left(f_{W'}(v_1) - f_W(c\sigma_1^{-1} v_1)\right) \nabla_{W'} f_{W'}(v_1) \\ &= \left(M v_1 + b - c\sigma_1^{-1} M v_1 - b\right) v_1^\top \\ &= \left(u_1 \sigma_1 - u_1 c\right) v_1^\top = u_1 \sigma_1 v_1^\top - u_1 c v_1^\top, \end{aligned}$$

where $\nabla_{W'} f_{W'}(v_1) = v_1^\top$ because it is as if we are computing the gradient of the linear operator with respect to its transformation matrix. Therefore, in line `7` if $\lambda = 1$ we compute the new transformation matrix as $\sum_{i=1}^{n} u_i \sigma_i v_i^\top - u_1 \sigma_1 v_1^\top + u_1 c v_1^\top = u_1 c v_1^\top + \sum_{i=2}^{n} u_i \sigma_i v_i^\top$. Notice that W'_δ is in the same format as the parameters of the linear operator (e.g., convolutional filter in convolutional layers).

If we set $\lambda = 1$, the largest singular value will be clipped to the desired value c, without requiring the `for` loop. That is indeed the case for dense layers. The reason that we let λ be a parameter and use the `for` loop is that for convolutional layers, we noticed that a slightly lower value of λ is required for the algorithm to work stably. The `for` loop allows this convergence to singular value c.

To derive our fast and precise clipping method, *FastClip*, we intertwine Algorithm 1 and Algorithm 2 with the outer SGD iterations used for training the model, as shown in Algorithm 3. As we mentioned in Section 2.1, the PowerQR method with warm start is able to track the largest singular values and corresponding vectors by running as few as one iteration per SGD step (lines 4 and 8 in Algorithm 3). Whenever the clipping method is called, we use this value and its corresponding vector as additional inputs to Algorithm 2 (rather than line 3 in Algorithm 2). By performing this clipping every few iterations of SGD, since the number of calls to this method becomes large and the changes to the weight matrix are slow, we do not need to run its `while` loop many times. Our experiments showed performing the clipping method every 100 steps and using only 1 iteration of `while` and `for` loops is enough for clipping the trained models (lines 9 and 10). Because after the clipping, the corresponding singular value of v has shrunk, we need to perform a few iterations of PowerQR on a new randomly chosen vector (since v is orthogonal to the other right singular vectors) to find the new largest singular value and corresponding right singular vector, and line 11 of Algorithm 2 takes care of this task. Obviously, the constants used in our algorithms are hyperparameters, but we did not tune them to find the best ones and the ones shown in this algorithm are what we used in all of our experiments. Also, any optimizer can be used for the SGD updates in line 7 (e.g., Adam (Kingma & Ba, 2014)).

2.3 Limitations of Convolutional Layers

In this section, we shed light on the formerly overlooked limitation of convolutional layers to represent any arbitrary spectrum. In some prior work, the proposed method for clipping computes the whole spectrum, clips the spectral norm, and then tries to form a new convolutional layer with the new spectrum (Sedghi et al., 2018; Senderovich et al., 2022). We also introduce a new simple optimization method that uses our PowerQR algorithm and adheres to this procedure in Appendix B.1. In the following, we start with a simple example that shows an issue with this procedure, and then, present our theoretical results that show a more general and fundamental limitation for a family of convolutional layers.

Consider a 2d convolutional layer whose kernel is 1×1

Algorithm 3 FastClip (f_W, X, c, N, η)

```
1: Input: Affine function $f_W$, Dataset $X$, Clip value $c$, Number of SGD iterations $N$, Learning rate $\eta$
2: Output: Trained affine function $f_{W'}$ with singular values clipped to $c$
3: $v \leftarrow$ Random input vector
4: $\sigma_1, v_1 \leftarrow$ PowerQR$(f_W, v_1, 10)$
5: for $i = 1$ to $N$ do
6:     $X_b \leftarrow$ SampleFrom$(X)$   // Sample a batch
7:     $W = W - \eta \nabla_W \ell(f_W(X_b))$   // SGD step
8:     $\sigma_1, v_1 \leftarrow$ PowerQR$(f_W, v_1, 1)$
9:     if $i$ isDivisibleBy 100 then
10:        $f_W, \sigma_1, v_1 \leftarrow$ Clip$(f_W, \sigma_1, v_1, c, 1, 10)$
11:    end if
12: end for
13: Return $f_W$
```

with a value of c. Applying this kernel to any input of size $n \times n$ scales the values of the input by c. The equivalent matrix form of this linear transformation is an $n \times n$ identity matrix scaled by c. We know that this matrix has a rank of n, and all the singular values are equal to 1. Therefore, if the new spectrum, S', does not represent a full-rank transformation, or if its singular values are not all equal, we cannot find a convolutional layer with a 1×1 kernel with S' as its spectrum.

In the following theorem, we compute the closed form of the singular values of convolutional layers with circular padding, and in Remark 2.3, we mention one of the general limitations that it entails.

Theorem 2.2. *For a convolutional layer with* 1 *input channel and* m *output channels (the same result holds for convolutions with* 1 *output channel and* m *input channels) and circular padding, if the vectorized form of the* j*-th channel of the filter is given by* $\boldsymbol{f}^{(j)} = [f_0^{(j)}, f_1^{(j)}, \dots, f_{k-1}^{(j)}]$, *the singular values are:*

$$\mathcal{S}(\omega) = \left\{ \sqrt{\sum_{j=1}^{m} \mathcal{S}_k^{(j)^2}(\omega)},\ k \in [n-1] \right\}, \qquad (1)$$

where for $j \in 1, \dots, m$*:*

$$\mathcal{S}^{(j)}(\omega) = \left[\sqrt{c_0^{(j)} + 2\sum_{i}^{k-1} c_i^{(j)} \Re(\omega^{k \times i})},\ k \in [n-1] \right]^\top$$

in which $c_i^{(j)}$*'s are defined as:*

$$c_0^{(j)} := f_0^{(j)^2} + f_1^{(j)^2} + \cdots + f_{k-1}^{(j)^2},$$
$$c_1^{(j)} := f_0^{(j)} f_1^{(j)} + f_1^{(j)} f_2^{(j)} + \cdots + f_{k-2}^{(j)} f_{k-1}^{(j)},$$
$$\vdots$$
$$c_{k-1}^{(j)} := f_0^{(j)} f_{k-1}^{(j)}.$$

Remark 2.3. Note that in the closed form, we only have the real parts of the roots of unity. So, for any non-real root of unity, we get a duplicate singular value because mirroring that root around the real axis (i.e., flipping the sign of the imaginary part) gives another root of unity with the same real component. Only the roots that are real might derive singular values that do not have duplicates. The only such roots with real parts are 1 and -1 (-1 is a root only for even n). Therefore, except for at most two singular values, other ones always have duplicates. This shows a more general limitation in the representation power of convolutional layers in representing arbitrary spectrums.

As Theorem 2.2 shows the singular values of convolutional layers with circular padding have a specific structure. This makes the singular values of the convolutional layers connected. This structure among the singular values shrinks the space of the linear transformations they can represent. One such limitation was mentioned in Remark 2.3. Using this theorem, we also derive an easy-to-compute upper bound and lower bound based on the values of the filter in Corollary A.2, which become equalities when the filter values are all non-negative.

One implication of this result is that the alternative approach for clipping the spectral norm of convolutional layers that assigns arbitrary values to all the singular values at once (in contrast to our iterative updates which do not leave the space of convolutional operators), which has been used in several prior works, cannot be correct and effective because convolutional layers cannot represent arbitrary spectrums.

2.4 Batch Normalization Layers

Batch Normalization (Ioffe & Szegedy, 2015) has proved to successfully stabilize and accelerate the training of deep neural networks and is thus by now standard in many architectures that contain convolutional layers. However, the adverse effect of these layers on the adversarial robustness of models has been noted in previous research Xie & Yuille (2019); Benz et al. (2021); Galloway et al. (2019). As we will show in our experiments, not controlling the spectral norm of the batch normalization layers might forfeit the benefits of merely controlling the spectral norm of convolutional layers. We also point out an interesting compensation behavior that the batch normalization layer exhibits when the spectral norm of the convolutional layer is clipped; As the clipping value gets smaller, the spectral norm of the batch normalization layers increases (see Figure 2a).

The clipping method introduced by Gouk et al. (2021) (explained in Appendix B.2), which is also used by

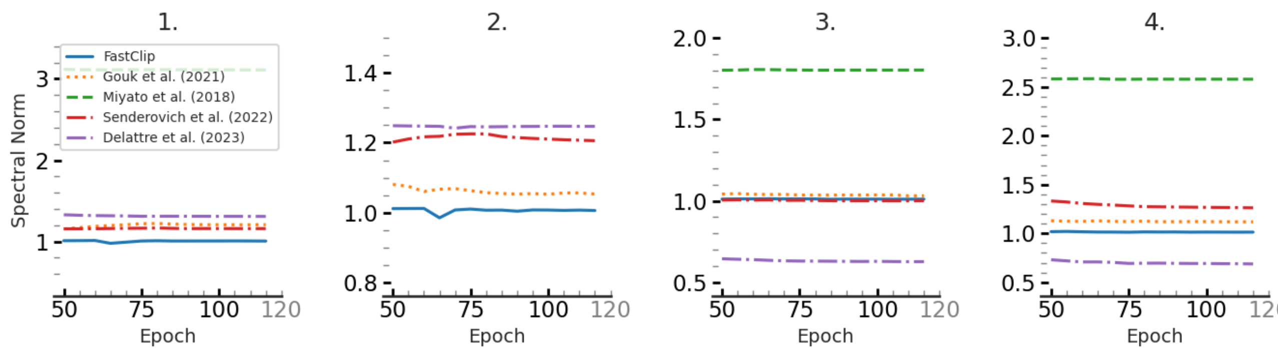


Figure 1: Comparison of the clipping methods in a simple network with only one convolutional layer and one dense layer, where **the target value is 1**. Our method is the only one that clips this layer correctly for all different settings: 1. Kernel of size 3 with reflect padding, 2. Kernel of size 3 with same padding, 3. Kernel of size 3 and zeros padding with stride of 2, and 4. Kernel of size 5 with same replicate padding and stride of 2.

the follow-up works (Senderovich et al., 2022; Delattre et al., 2023), performs the clipping of the batch norm layer separately from the preceding convolutional layer, and therefore upper-bounds the Lipschitz constant of their concatenation by the multiplication of their individual spectral norms. This upper-bound, although correct, is not tight, and the actual Lipschitz constant of the concatenation might be much smaller, which might lead to unwanted constraining of the concatenation such that it hurts the optimization of the model. Also, the purpose of the batch normalization layer is to control the behavior of its preceding convolutional layer, and therefore, clipping it separately does not seem to be the best option. As explained in Section 2.2, our clipping method can be applied directly to the concatenation of the convolutional layer and the batch normalization layer. This way, we can control the spectral norm of the concatenation without tweaking the batch normalization layer separately. We will show in our experiments that this method can be effective in increasing the robustness of the model, without compromising its accuracy.

3 EXPERIMENTS

We perform various experiments to show the effectiveness of our algorithms and compare them to the state-of-the-art methods to show their advantages. Since the method introduced by Senderovich et al. (2022) extends the method of Sedghi et al. (2018) and (Farnia et al., 2018) use the same method as (Gouk et al., 2021), the methods we use in our comparisons are the methods introduced by Miyato et al. (2018); Gouk et al. (2021); Senderovich et al. (2022); Delattre et al. (2023). For all these methods we used the settings recommended by the authors in all experiments. Implementations of our methods and experiments can be found in the supplementary material. All the experiments were performed on a single node with an NVIDIA A40 GPU. Some details of our experiments along with further results are moved to Appendix C because of the space limitation.

3.1 PowerQR

As pointed out previously, the methods introduced by Senderovich et al. (2022) and Delattre et al. (2023), compute the exact spectral norm only when circular padding is used for the convolutional layers. For the other types of common paddings for these layers, these methods can result in large errors. To show this, we computed the average absolute value of the difference in their computed values and the correct value for a 100 randomly generated convolutional filters for different types of padding (see Appendix C.1). As the results show, the error dramatically increases as the number of channels increases. Also, the method introduced by Delattre et al. (2023) assumes a stride of 1 and therefore leads to even larger errors when a stride of 2 is used. We also show that PowerQR is much more efficient than k successive runs of the power method followed by deflation (as suggested in Virmaux & Scaman (2018)) for extracting the top-k singular values in Appendix C.1. We also have utilized the capability of our method to extract the spectral norm of the concatenation of multiple implicitly linear layers for analyzing the spectral norm of the concatenation of convolutional and batch norm layers in Figure 2 (see Appendix C.4 for more use-cases).

3.2 Clipping Method

We start by comparing the correctness of the clipping models for different types of convolutional layers. Then, we show the effectiveness of each of these methods on the generalization and robustness

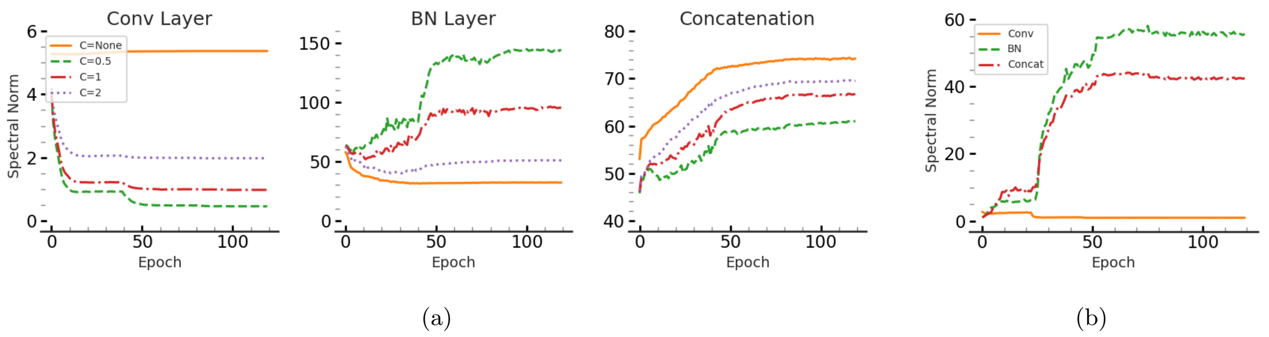


Figure 2: **(a)** The first three plots show the clipping of the convolutional layer in a simple two-layer network to various values on MNIST. As the clipping target gets smaller, the spectral norm of the batch norm layer compensates and becomes larger. Meanwhile, the spectral norm of their concatenation slightly decreases. **(b)** The right-most plot shows the spectral norm of a convolutional layer, its succeeding batch norm layer, and their concatenation from the clipped ResNet-18 model trained on CIFAR-10. Although the convolutional layer is clipped to 1, the spectral norm of the concatenation is much larger due to the presence of the batch norm layer.

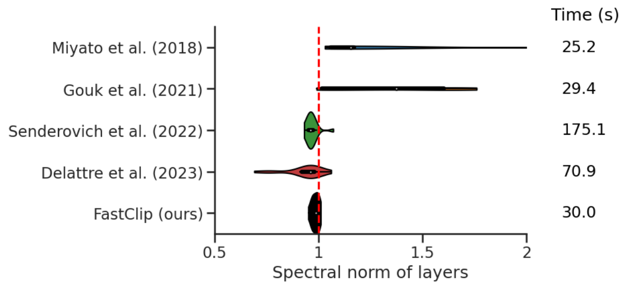


Figure 3: The layer-wise spectral norm of a ResNet-18 model trained on CIFAR-10 using each of the clipping methods. The time column shows the training time per epoch for these methods. As the plot shows, by using our method, all the layers have a spectral norm very close to the **target value 1**. Our method is much faster than the relatively accurate alternatives.

of ResNet-18 (He et al., 2016), which is the same model used in the experiments of prior work (Gouk et al., 2021; Senderovich et al., 2022; Delattre et al., 2023) and Deep Layer Aggregation (DLA) (Yu et al., 2018) model, which is more complex with more layers and parameters. We train the models on the same datasets used in prior works, MNIST (LeCun, 1998) and CIFAR-10 (Krizhevsky et al., 2009).

3.2.1 Precision and Efficiency

We use a simple model with one convolutional layer and one dense layer and use each of the clipping methods on the convolutional layer while the model is being trained on MNIST and the target clipping value is 1. We compute the true spectral norm after each epoch. Figure 1 shows the results of this experiment for 4 convolutional layers with different settings (e.g., kernel size and padding type). This figure shows our method is the only one that correctly clips various convolutional layers.

We also compared the spectral norm of all the layers of the ResNet-18 model trained using each of the clipping methods on CIFAR-10 and MNIST and presented them in Figure 3. This figure also shows the average wall-clock time per epoch for each of the methods. As the figure shows, our method is the most precise one while being among the fastest ones. The results are similar for the ResNet-18 model trained on MNIST (see Appendix C.2).

3.2.2 Generalization and Robustness

Since the Lipschitz constant of a network may be upper-bounded by multiplying together the spectral norms of its constituent dense and convolutional layers, it is intuitive that regularizing per-layer spectral norms improves model generalization (Bartlett et al., 2017), and also adversarial robustness (Szegedy et al., 2013). Therefore, we tested all the clipping models by using them during the training and computing the accuracy of the corresponding models on the test set and adversarial examples generated by two common adversarial attacks, Projected Gradient Descent (PGD) (Madry et al., 2017) and Carlini & Wanger Attack (CW) (Carlini & Wagner, 2017).

As Table 1 shows, our model leads to the best improvement in test accuracy while making the models more robust to adversarial attacks (similar results for DLA are presented in Appendix C.3). The reason for the lack of the expected boost in the robustness of the models when clipping their spectral norms is shown in Figure 2b. This figure shows that although the models are clipped, the concatenation of some convolu-

Table 1: The accuracy on the test set and adversarial examples generated with PGD-(50) and CW for ResNet-18 trained on MNIST and CIFAR-10, for the models with their convolutional and dense layers clipped to 1 (`With BN`) and clipped models with their batch norm layers removed (`No BN`). The accuracy of the original model on the test set, PGD-generated examples, and CW-generated examples for MNIST are **99.37 ± 0.02**, **15.30 ± 9.11**, and **77.46 ± 5.89**, respectively. For CIFAR-10, these values are **94.77 ± 0.19**, **22.15 ± 0.54**, and **13.85 ± 0.74**.

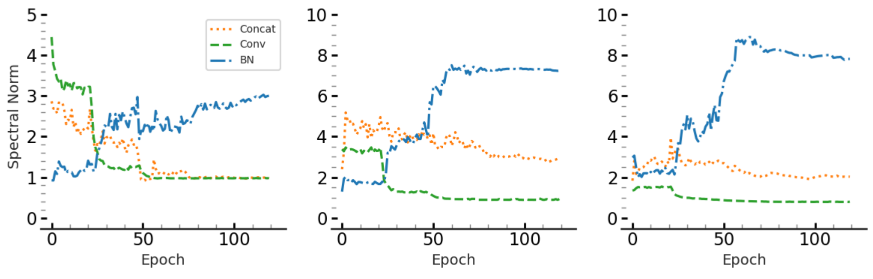
	MNIST		**CIFAR-10**	
Method	With BN	No BN	With BN	No BN
	Accuracy on the test set			
Miyato et al. (2018)	99.40 ± 0.06	98.00 ± 0.22	94.82 ± 0.11	88.83 ± 1.41
Gouk et al. (2021)	99.25 ± 0.04	21.94 ± 6.01	89.98 ± 0.38	19.80 ± 5.55
Senderovich et al. (2022)	99.40 ± 0.03	62.63 ± 24.01	94.19 ± 0.13	68.29 ± 10.63
Delattre et al. (2023)	99.29 ± 0.05	97.27 ± 0.03	93.17 ± 0.13	39.35 ± 9.84
FastClip (Algorithm 3)	$\mathbf{99.41 \pm 0.04}$	$\mathbf{99.31 \pm 0.02}$	$\mathbf{95.28 \pm 0.07}$	$\mathbf{92.08 \pm 0.28}$
	Accuracy on samples from PGD attack			
Miyato et al. (2018)	21.77 ± 12.98	32.67 ± 14.08	23.48 ± 0.11	35.18 ± 7.72
Gouk et al. (2021)	2.40 ± 2.94	8.41 ± 3.03	16.13 ± 1.28	14.66 ± 3.99
Senderovich et al. (2022)	30.99 ± 9.28	15.97 ± 4.84	21.74 ± 0.72	39.84 ± 7.87
Delattre et al. (2023)	30.87 ± 4.77	71.75 ± 1.49	21.08 ± 0.84	16.22 ± 3.17
FastClip (Algorithm 3)	$\mathbf{47.90 \pm 5.49}$	$\mathbf{78.50 \pm 2.85}$	$\mathbf{24.48 \pm 0.32}$	$\mathbf{41.37 \pm 0.95}$
	Accuracy on samples from CW attack			
Miyato et al. (2018)	86.25 ± 2.18	73.56 ± 10.38	16.68 ± 0.95	48.48 ± 6.40
Gouk et al. (2021)	66.59 ± 21.91	21.94 ± 6.01	18.79 ± 2.99	12.63 ± 4.33
Senderovich et al. (2022)	87.72 ± 2.75	58.71 ± 20.67	20.53 ± 0.77	43.82 ± 9.57
Delattre et al. (2023)	83.97 ± 1.79	$\mathbf{96.93 \pm 0.06}$	24.05 ± 1.72	11.92 ± 5.34
FastClip (Algorithm 3)	$\mathbf{90.21 \pm 1.80}$	95.35 ± 1.06	$\mathbf{24.31 \pm 0.96}$	$\mathbf{56.28 \pm 0.96}$

tional layers with batch normalization layers forms linear operators with large spectral norms. As Figure 2a suggests, clipping the convolutional layer to smaller values will further increase the spectral norm of the batch norm layer. Still, as this figure suggests, there might be an overall decrease in the spectral norm of their concatenation which causes the slight improvement in the robustness of the clipped models. Therefore, we also trained a version of the model with all the batch norm layers removed. As Table 1 shows, this leads to a huge improvement in the robustness of the clipped models; however, the clipped models achieve worse accuracy on the test set.

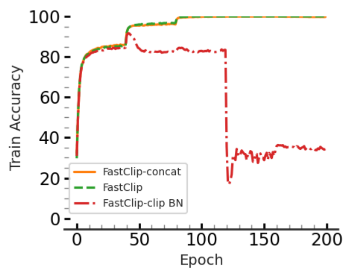
3.3 Clipping Batch Norm

As Table 1 shows, the presence of batch normalization can be essential for achieving high test accuracy. In fact, as explained in Appendix C.3, DLA models cannot be trained without batch norm. So, instead of removing them, we are interested in a method that allows us to control their adverse effect on the robustness of the model. In Appendix C.4 we show the results for the models with their batch norm layers clipped to strictly less than 1 by utilizing the method that was used by prior work (Gouk et al., 2021; Senderovich et al., 2022). As the results show, our method still achieves the best results with this technique; however, this method for clipping the batch norm, although leads to bounded Lipschitz constant for the model, does not lead to a significant improvement in the robustness of the models and leads to low test accuracy, which is due to over-constraining the models and hindering their optimization as discussed in Section 2.4. In fact, as Figure 4b shows, using this method for controlling batch norm layers even hinders the training process (we show this version of our method by `FastClip-clip BN` in our experiments).

The capability of our clipping method to be applied to the concatenation of implicitly linear layers provides an alternative approach to control the spectral norm of the concatenation of convolutional layers and batch norm layers. This still leads to the desired Lipschitz constants for the model, without over-constraining each individual layer. For this purpose, we clip the convolutional layers of the model to the target value, and meanwhile, we pass the modules that represent the composition of batch norm layers and



(a)



(b)

Figure 4: **(a)** Each of these three subplots shows the spectral norms of a convolutional layer, its succeeding batch norm layer, and their concatenation in a ResNet-18 model trained on CIFAR-10. The convolutional layers in this model are clipped to 1. Instead of clipping the batch normalization layer, our method has been applied to the concatenation to control its spectral norm. **(b)** The rightmost subplot shows the training accuracy for the ResNet-18 model that is trained on CIFAR-10. One curve belongs to the model with the convolutional layers clipped to 1 using FastClip and the batch norm layers clipped using the direct method used by prior works (FastClip-clip BN). The other two belong to FastClip and FastClip-concat.

their preceding convolutional layers to our clipping method while leaving the batch norm layers themselves unclipped. We will refer to this version of our clipping method as `FastClip-concat` in our experiments. In Figure 4a, we show the effect of this approach on 3 of the layers from the ResNet-18 model trained using `FastClip-concat`. As the plot shows, the convolutional layers are correctly clipped to 1, and the spectral norm of the concatenations are approaching the target value 1, while the spectral norm of the batch norm layers might increase up to an order of magnitude larger than the target clipping value. The convergence of the spectral norm of the concatenation to the target value is slower because we used a much smaller λ value (see Algorithm 2) to make the clipping method stable without changing the other hyperparameters.

Figure 4b shows the trajectory of the training accuracies for `FastClip`, `FastClip-concat`, and `FastClip-clip BN` (which uses `FastClip` together with the direct batch norm clipping method used in prior works Gouk et al. (2021)). As the figure shows, direct clipping of the batch norm layers hinders the optimization of the model and hence leads to poor results presented in Table 3, while `FastClip-concat` follows the same trajectory as `FastClip` in terms of the training accuracy, which shows less interference with the optimization of the model. In Appendix C.4 we elaborate on the results for each of these modifications to the clipping methods for controlling the spectral norm of the batch norm layers. As these results show, `FastClip-concat` is more successful in achieving its goal; it bounds the spectral norm of the concatenation and leads to improved robustness without incurring as much loss to the test accuracy compared to removing the batch normalization layers. Further optimization of the hyperparameters (e.g., clipping value of the convolutional layer, clipping value of the concatenation, λ, number of steps for clipping, etc.) for the convolutional layers and the concatenations, and finding the best combination is left for future work.

4 CONCLUSIONS

We introduced efficient and accurate algorithms for extracting and clipping the spectrum of implicitly linear layers. We showed they are more accurate and effective than existing methods. Also, our algorithms are unique in that they can be applied to not only convolutional and dense layers, but also the concatenation of these layers with other implicitly linear ones, such as batch normalization. This opens up the new possibilities in controlling the Lipschitz constant of models and needs to be explored further in future work. Through various experiments, we showed that our clipping method can be used as an effective regularization method for the neural networks containing convolutional and dense layers, which helps the model to generalize better on unseen samples and makes it more robust against adversarial attacks.

References

El Mehdi Achour, François Malgouyres, and Franck Mamalet. Existence, stability and scalability of orthogonal convolutional neural networks. *The Journal of Machine Learning Research*, 23(1):15743–15798, 2022.

Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. In *Interna-*

tional Conference on Machine Learning, pp. 291–301. PMLR, 2019.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, 30, 2017.

Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Revisiting batch normalization for improving corruption robustness. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 494–503, 2021.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57. Ieee, 2017.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pp. 854–863. PMLR, 2017.

Blaise Delattre, Quentin Barthélemy, Alexandre Araujo, and Alexandre Allauzen. Efficient bound of lipschitz constant for convolutional layers by gram iteration. *arXiv preprint arXiv:2305.16173*, 2023.

Farzan Farnia, Jesse M Zhang, and David Tse. Generalizable adversarial training via spectral normalization. *arXiv preprint arXiv:1811.07457*, 2018.

Angus Galloway, Anna Golubeva, Thomas Tanay, Medhat Moussa, and Graham W Taylor. Batch normalization is a cause of adversarial vulnerability. *arXiv preprint arXiv:1905.02161*, 2019.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Anil K Jain. *Fundamentals of digital image processing.* Prentice-Hall, Inc., 1989.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Stamatios Lefkimmiatis, John Paul Ward, and Michael Unser. Hessian schatten-norm regularization for linear inverse problems. *IEEE transactions on image processing*, 22(5):1873–1888, 2013.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition.* SIAM, 2011.

Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*, 2018.

Alexandra Senderovich, Ekaterina Bulatova, Anton Obukhov, and Maxim Rakhuba. Towards practical control of singular values of convolutional layers. *Advances in Neural Information Processing Systems*, 35:10918–10930, 2022.

Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *International Conference on Machine Learning*, pp. 9756–9766. PMLR, 2021.

Sahil Singla, Surbhi Singla, and Soheil Feizi. Improved deterministic l2 robustness on cifar-10 and cifar-100. *arXiv preprint arXiv:2108.04062*, 2021.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the cayley transform. *arXiv preprint arXiv:2104.07167*, 2021.

Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

Cihang Xie and Alan Yuille. Intriguing properties of adversarial training at scale. *arXiv preprint arXiv:1906.03787*, 2019.

Xiaojun Xu, Linyi Li, and Bo Li. Lot: Layer-wise orthogonal training on improving l2 certified robustness. *Advances in Neural Information Processing Systems*, 35:18904–18915, 2022.

Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2403–2412, 2018.

Tan Yu, Jun Li, Yunfeng Cai, and Ping Li. Constructing orthogonal convolutions in an explicit manner. In *International Conference on Learning Representations*, 2021.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes, they are included in the supplementary material.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes
   (b) Complete proofs of all theoretical results. Yes
   (c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes, they are included in the suplementary material.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Not Applicable
   (b) The license information of the assets, if applicable. Not Applicable
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable
   (d) Information about consent from data providers/curators. Not Applicable
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

A Omitted Proofs

A.1 Proofs of Section 2.1

We start by proving Proposition 2.1:

Proof. Let $g(X) = f(X) - f(0) = M_W X + b - b = M_W X$. Notice that $\nabla_X \frac{1}{2}\|g(X)\|^2 = M_W^T M_W X = A_W X$. Let define $A' = A_W + \mu I$. Putting steps 5 and 6 of the algorithm together, we have $(Q, R) = \mathrm{QR}(A_W X + \mu X) = \mathrm{QR}(A' X)$. Therefore, the algorithm above can be seen as the same as the subspace iteration for matrix A'. Hence, the diagonal values of R will converge to the eigenvalues of A', and columns of Q converge to the corresponding eigenvectors. If λ is an eigenvalue of $A' = A_W + \mu I$, then $\lambda - \mu$ is an eigenvalue of A_W. Therefore, by subtracting μ from the diagonal elements of R, we get the eigenvalues of A_W, which are squared of singular values of M_W. The eigenvectors of A' are the same as the eigenvectors of A_W and, therefore, the same as the right singular vectors of M_W.

□

A.2 Proofs of Section 2.3

For proving Theorem 2.2, we first present and prove Lemma A.1 which proves a similar result for convolutional layers with only 1 input and output channel.

Lemma A.1. *Given a convolutional layer with single input and output channel and circular padding, if the vectorized form of the kernel is given by $\boldsymbol{f} = [f_0, f_1, \ldots, f_{k-1}]$, the singular values are:*

$$\mathcal{S}(\omega) = \left\{ \sqrt{c_0 + 2\sum_{i}^{k-1} c_i \Re(\omega^{j\times i})},\ j = 0, 1, 2, \ldots, n-1 \right\} \tag{2}$$

where c_i's are defined as:

$$\begin{aligned} c_0 &:= f_0^2 + f_1^2 + \cdots + f_{k-1}^2, \\ c_1 &:= f_0 f_1 + f_1 f_2 + \cdots + f_{k-2} f_{k-1}, \\ &\vdots \\ c_{k-1} &:= f_0 f_{k-1}. \end{aligned}$$

Proof. The above convolutional operator is equivalent to a circulant matrix A with $[f_m, f_{m+1}, \ldots, f_{k-1}, 0, \ldots, 0, f_0, f_1, \ldots, f_{m-1}]^T$ as its first row, where $m = \lfloor \frac{k}{2} \rfloor$ Jain (1989); Sedghi et al. (2018). Singular values of A are the eigenvalues of $M = A^T A$. Circulant matrices are closed with respect to multiplication, and therefore M is a symmetric circulant matrix. It is easy to check that the first row of M is $r = [c_0, c_1, \ldots, c_{k-1}, 0, \ldots, 0, c_{k-1}, \ldots, c_1]^T$.

Now, we know the eigenvalues of a circulant matrix with first row $v = [a_0, a_1, \ldots, a_{n-1}]^T$ are given by the set $\Lambda = \{v\Omega_j, j = 0, 1, \ldots, n-1\}$, where $\Omega_j = [\omega^{j\times 0}, \omega^{j\times 1}, \ldots, \omega^{j\times n-1}]$. So eigenvalues of M are given by the set below:

$$\begin{aligned} \{r\Omega_j,\ j = 0, 1, \ldots, n-1\} = \{&c_0 + c_1\omega^{j\times 1} + c_2\omega^{j\times 2} + \cdots + c_{k-1}\omega^{j\times(k-1)} \\ &+ c_{k-1}\omega^{j\times(n-k+1)} + c_{k-2}\omega^{j\times(n-k+2)} + \cdots + c_1\omega^{j\times(n-1)},\ j = 0, 1, \ldots, n-1\}. \end{aligned}$$

Note that $\omega^{j\times i}$ has the same real part as $\omega^{j\times(n-i)}$, but its imaginary part is mirrored with respect to the real axis (the sign is flipped). Therefore, $\omega^{j\times i} + \omega^{j\times(n-i)} = 2\Re(\omega^{j\times i})$. Using this equality the summation representing the eigenvalues of M can be simplified to:

$$\{c_0 + 2\sum_{i}^{k-1} c_i \Re(\omega^{j\times i}),\ j = 0, 1, \ldots, n-1\},$$

and therefore the singular values can be derived by taking the square roots of these eigenvalues. □

Now using Lemma A.1, we can easily prove Theorem 2.2.

Proof. The convolutional layer with m output channels and one input channel can be represented by a matrix $M = [M_1, M_2, \ldots, M_m]^T$, where each M_i is an $n \times n$ circulant matrix representing the $i-$th channel. Therefore, $A = M^T M$ (or MM^T when there are multiple input channels) can be written as $\sum_{j=1}^m M_j^T M_j$. So, for circulant matrix A we have $c_i = \sum_{j=1}^m c_i^{(j)}$, and the proof can be completed by using Lemma A.1. □

Next, we focus on Corollary A.2, which uses the results of Theorem 2.2 to give an easy-to-compute lower and upper bounds for the spectral norm of the convolutional layers with either one input or output channel. When the filter values are all positive, these bounds become equalities and provide and easy way to compute the exact spectral norm.

Corollary A.2. *Consider a convolutional layer with* 1 *input channel and* m *output channels or* 1 *output channel and* m *input channels and circular padding. If the vectorized form of the kernel of the* j*-th channel is given by* $\boldsymbol{f^{(j)}} = [f_0^{(j)}, f_1^{(j)}, \ldots, f_{k-1}^{(j)}]$*, and the largest singular value of the layer is* σ_1*, then:*

$$\sqrt{\sum_{j=1}^{m}\left(\sum_{i=0}^{k-1} f_i^{(j)}\right)^2} \leq \sigma_1 \leq \sqrt{\sum_{j=1}^{m}\left(\sum_{i=0}^{k-1} |f_i^{(j)}|\right)^2}, \tag{3}$$

and therefore the equalities hold if all $f_i^{(j)}$*s are non-negative.*

To make the proof more clear, we first present a similar corollary for Lemma A.1, which proves similar results for convolutions with only 1 input and output channel.

Corollary A.3. *Consider a convolutional layer with single input and output channels and circular padding. If the vectorized form of the kernel is given by* $\boldsymbol{f} = [f_0, f_1, \ldots, f_{k-1}]$*, and the largest singular value of the layer is* σ_1*, then:*

$$\sum_{i=0}^{k-1} f_i \leq \sigma_1 \leq \sum_{i=0}^{k-1} |f_i|, \tag{4}$$

and therefore, the equalities hold if all f_i*s are non-negative.*

Proof. From Lemma A.1, we can write:

$$\mathcal{S}(\omega) = \sqrt{c_0 + 2\sum_{i}^{k-1} c_i \Re(\omega^i)} = \sqrt{|c_0 + 2\sum_{i}^{k-1} c_i \Re(\omega^i)|}$$
$$\leq \sqrt{|c_0| + 2\sum_{i}^{k-1} |c_i \Re(\omega^i)|} \leq \sqrt{|c_0| + 2\sum_{i}^{k-1} |c_i|},$$

where the last inequality is due to the fact that $\Re(\omega^i) \leq 1$. Now, for c_is we have:

$$\begin{aligned}|c_0| &= f_0^2 + f_1^2 + \cdots + f_{k-1}^2, \\ 2|c_1| &\leq 2\left(|f_0||f_1| + |f_1||f_2| + \cdots + |f_{k-2}||f_{k-1}|\right), \\ &\vdots \\ 2|c_{k-1}| &\leq 2\left(|f_0||f_{k-1}|\right).\end{aligned}$$

Note that the summation of the right-hand sides of the above inequalities is equal to $(\sum_{i=0}^{k-1} |f_i|)^2$ Therefore:

$$\mathcal{S}(\omega) \leq \sum_{i=0}^{k-1} |f_i| \implies \sigma_1 \leq \sum_{i=0}^{k-1} |f_i|.$$

Since 1 is always one of the $n-$th roots of unity, if all the f_is are non-negative, all the above inequalities hold when $\omega = 1$ is considered. Now, note that when $\omega = 1$:

$$\begin{aligned}\mathcal{S}^2(1) &= c_0 + 2\sum_{i}^{k-1} c_i \\ &= f_0^2 + f_1^2 + \cdots + f_{k-1}^2 \quad (c_0) \\ &\quad + 2f_0f_1 + 2f_1f_2 + \cdots + 2f_{k-2}f_{k-1} \quad (c_1) \\ &\quad \vdots \\ &\quad + 2f_0f_{k-1} \quad (c_{k-1}) \\ &= (\sum_{i=0}^{k-1} f_i)^2\end{aligned}$$

Therefore, $\sum_{i=0}^{k-1} f_i$ is a singular value of the convolution layer, which gives a lower bound for the largest one and this completes the proof.

□

Now we give the proof for Corollary A.2.

Proof. From Theorem 2.2, we can write:

$$\mathcal{S}(\omega) = \sqrt{\sum_{j=1}^{m} \mathcal{S}^{(j)^2}(\omega)} \leq \sqrt{\sum_{j=1}^{m}(\sum_{i=0}^{k-1} |f_i^{(j)}|)^2},$$

where the inequality is a result of using Corollary A.3 for each $\mathcal{S}^{(j)}(\omega)$. For showing the left side of inequality, we set $\omega = 1$ (1 is one of the n-th roots of unity), and again use Theorem 2.2 to get:

$$\mathcal{S}(1) = \sqrt{\sum_{j=1}^{m} \mathcal{S}^{(j)^2}(1)} = \sqrt{c_0^{(j)} + 2\sum_{i}^{k-1} c_i^{(j)} \Re(1)} = \sqrt{\sum_{j=1}^{m}(\sum_{i=0}^{k-1} f_i^{(j)})^2},$$

where the last equality was shown in the proof of Corollary A.3. This shows the left side of the inequality in (4) is one of the singular values of the layer, and hence is a lower bound for the spectral norm (the largest singular value of the layer).

□

B Other Algorithms

B.1 Modifying the Whole Spectrum

If we use PowerQR to extract singular values and right singular vectors S and V from $M_W = USV^T$, then given new singular values S', we can modify the spectrum of our function to generate a linear operator $f' : x \to M'_W x + b$, where $M'_W = US'V^T$, without requiring the exact computation of U or M_W:

$$f_W(VS^{-1}S'V^Tx) - f(0) = USV^TVS^{-1}S'V^Tx = US'V^Tx = f'(x) - b.$$

Although f' is a linear operator with the desired spectrum, it is not necessarily in the desired form. For example, if f_W is a convolutional layer with W as its kernel, $f_W(VSS'V^Tx)$ will not be in the form of a convolutional layer. To find a linear operator of the same form as f_W, we have to find new parameters W' that give us f'. For this, we can form a convex objective function and use SGD to find the parameters W' via regression:

$$\min_{W'} \mathbb{E}_x \left\| f_{W'}(x) - f_W(VS^{-1}S'V^Tx) \right\|_{\mathrm{F}}^2, \tag{5}$$

where x is randomly sampled from the input domain. Note that we do not need many different vectors x to be sampled for solving 5. If the rank of the linear operator $f_W(.)$ is n, then sampling as few as n random data points would be enough to find the optimizer $f_{W'}$. This procedure can be seen as two successive projections, one to the space of linear operators with the desired spectrum and the other one to the space of operators with the same form as f_W (e.g., convolutional layers). This method can be very slow because the matrix V can be very large. One might reduce the computational cost by working with the top singular values and singular vectors and deriving a low-ranked operator. This presented method is not practical for controlling the spectral norm of large models during training, similar to the other algorithms that need the whole spectrum for controlling the spectral norm (Sedghi et al., 2018; Senderovich et al., 2022).

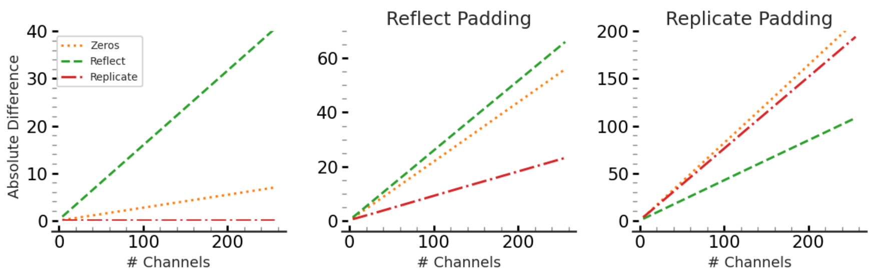
Another important point to mention here is that depending on the class of linear operators we are working with, the objective 5 might not reach zero at its minimizer. For example, consider a 2d convolutional layer whose kernel is 1×1 with a value of c. Applying this kernel to any input of size $n \times n$ scales the value of the input by the c. The equivalent matrix form of this linear transformation is an $n \times n$ identity matrix scaled by c. We know that this matrix has a rank of n, and all the singular values are equal to 1. Therefore, if the new spectrum, S', does not represent a full-rank transformation, or if its singular values are not all equal, we cannot attain a value of 0 in optimization 5. As we showed in Section 2.3, this problem is more general for the convolutional layers, as they are restricted in the spectrums they can represent.

B.2 Clipping Batch Norm

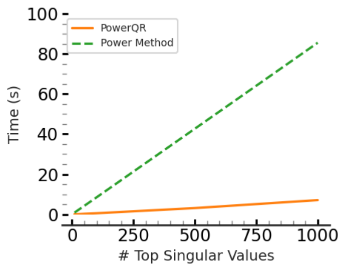
Batch Normalization (Ioffe & Szegedy, 2015) has proved to successfully stabilize and accelerate the training of deep neural networks and is thus by now standard in many architectures that contain convolutional layers. However, the adverse effect of these layers on the adversarial robustness of models has been noted in previous research Xie & Yuille (2019); Benz et al. (2021); Galloway et al. (2019). As we showed in Figure 2b, not controlling the spectral norm of the batch normalization layers might forfeit the benefits of merely controlling the spectral norm of convolutional layers. We also revealed the compensating behavior of these layers as the spectral norm of convolutional layers are clipped to smaller values in Figure 2a. The presented results in Table 1 confirm this adverse effect of batch norm layers by showing a boost in the adversarial robustness of the models when these layers are removed from the network; however, as the results show, removing these layers from the model will incur a noticeable loss to the performance of the model on the test set and hinders the optimization. As pointed out in Appendix C.3, removing these layers from more complex modes such as DLA might completely hinder the optimization of the model. Therefore, it is crucial to find a better way to mitigate the adverse effect of these layers on adversarial robustness while benefiting from the presence of these layers in improving the optimization and generalization of the models.

Batch normalization layers perform the following computation on the output of their preceding convolution layer:

$$y = \frac{x - \mathbb{E}(x)}{\sqrt{\mathrm{Var}(x) + \epsilon}} * \gamma + \beta.$$



(a)



(b)

Figure 5: **a.** The absolute difference in the spectral norm of convolutional layers with different padding types and their circulant approximates for various kernel sizes (3, 5, and 7) and numbers of channels. The values are computed by averaging over 100 convolutional filters drawn from a normal distribution for each setting. **b.** Comparison of the run-time of PowerQR (algorithm 1) to that of the pipeline used by Virmaux & Scaman (2018) for computing the top-k singular values. We considered a 2d-convolutional layer with 3×3 filters and 32 input/output channels. The convolution is applied to a 32×32 image.

As pointed out by Gouk et al. (2021), by considering this layer as a linear transformation on $x - \mathbb{E}(x)$, the transformation matrix can be represented as a diagonal matrix with $\gamma_i / \sqrt{\mathrm{Var}(x_i) + \epsilon}$ values, and therefore its largest singular value is equal to $\max_i \left(|\gamma_i| / \sqrt{\mathrm{Var}(x_i) + \epsilon} \right)$. So, by changing the magnitude of γ_i values we can clip the spectral norm of the batch norm layer. This approach has been followed in prior work (Gouk et al., 2021; Senderovich et al., 2022; Delattre et al., 2023). In Appendix C.4, we show the results for the application of this clipping method and point out its disadvantages when used in practice. We will also show that the capability of our presented clipping method to work on the concatenation of convolutional and batch norm layers provides us with a better alternative. The full exploration of its potential, however, is left for future work.

C Experiments

In this section, we elaborate on the experiments and results pointed out in section 3 and present some additional experiments to show the advantages of our proposed methods. The models used for the experiments are ResNet-18, as it was used by prior work Gouk et al. (2021); Senderovich et al. (2022); Delattre et al. (2023), which has a small modification compared to the original model; for the residual connections in the model, they simply divide the output of the layer by 2 so that if both of the layers are 1-Lipschitz, the output of the module will remain 1-Lipschitz. Similarly, for the optimization of these models, we use SGD optimizer and a simple scheduler that decays the learning rate by 0.1 every 40 steps. We train each of the models for 200 epochs. In addition to ResNet-18 models, we perform the comparisons on DLA, as it is presented in a publicly available GitHub repository [1] as a model that achieves better results on CIFAR-10. We use the same training procedure for these models as the ones used for the ResNet-18 model. For generating adversarial examples and evaluating the models, we use a publicly available repository [2], along with the strongest default values for the attacks ($c = \epsilon = 0.02$ and $c = \epsilon = 0.1$ for CW and PGD attack on CIFAR-10 and MNIST, respectively), and also add their training procedure for MNIST to evaluate both ResNet-18 and DLA models on an additional dataset. We also use a simple model which consists of a single convolutional layer and a dense layer on the MNIST dataset for some of our experiments regarding the correctness of clipping methods (Figure 1) and showing the compensation phenomenon (Figure 2a).

C.1 Spectrum Extraction

Our introduced method can be used to compute the exact spectrum of any implicitly linear layer, including different types of convolutional layers. However, the method introduced by Sedghi et al. (2018) only computes the spectrum of the convolutional layers when they have circular padding and no strides. This method was

[1] https://github.com/kuangliu/pytorch-cifar
[2] https://github.com/AI-secure/Transferability-Reduced-Smooth-Ensemble/tree/main

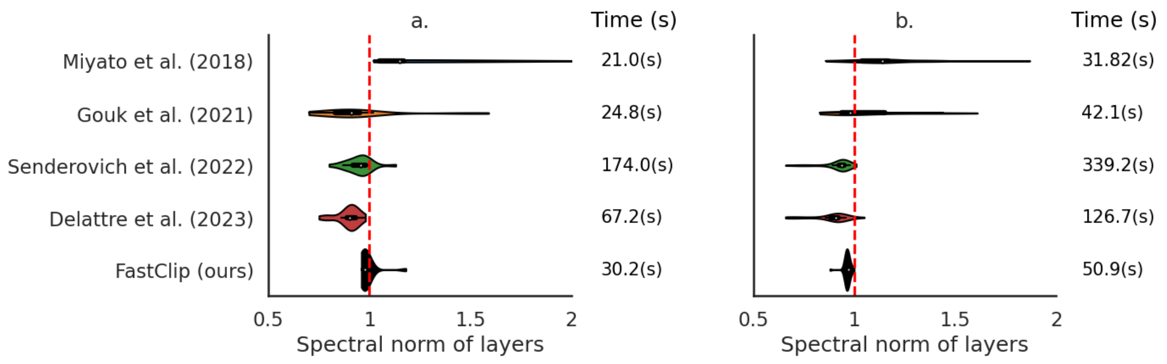


Figure 6: **a.** The layer-wise spectral norm of a ResNet-18 model trained on MNIST using each of the clipping methods. The time column shows the training time per epoch for these methods. **b.** The layer-wise spectral norm of a DLA model trained on CIFAR-10 using each of the clipping methods. The time column shows the training time per epoch for these methods. As both of the plots show, by using our method, all the layers have a spectral norm very close to the **target value 1**. Our method is also much faster than the relatively accurate alternatives and shows a slower increase in running time as the model gets larger.

extended to convolutional layers with circular padding with strides other than 1 by Senderovich et al. (2022). These methods, as shown in Section 3.2.1 and Appendix C.2, are very slow and not practical for large models. These methods are also very memory consuming compared to other methods. More recently, Delattre et al. (2023) introduced an algorithm that uses Gram iteration to derive a fast converging upper-bound for the convolutional layers with circular padding with a stride of 1. As shown in Figure 3 and Figure 6, this method is still much slower than our method for clipping and less accurate than its prior work on circulant convolutional layers. In Figure 5a we show that the approximation error using these methods can be large on random convolutional layers. We use convolutional layers with different standard padding types with various filter sizes and choose the filter values from a normal distribution. Then we use the circulant approximation to approximate the spectral norm of these convolutional layers, and present the absolute difference with the true spectral norm. The presented values are averaged over 100 trials for each setting. As the figure shows, the approximation error can be large, especially as the number of channels or the kernel size increases.

We also compared the run-time of the PowerQR with 100 steps for extracting the top-k eigenspace to the run-time of k successive runs of the power method (as suggested in Virmaux & Scaman (2018)). We did not include the deflation step (required for their proposed method to work), which makes the running time of their method even worse. Figure 5b in Appendix B shows that our implicit implementation of subspace iteration is dramatically more efficient.

C.2 Precise and Efficient Clipping

In Section 3.2.1 we presented the results of an experiment that showed our method is the only one that performs a correct clipping for all standard convolutional layers (Figure 1). In Figure 3, we also evaluated the distribution of the layer-wise spectral norms of the ResNet-18 models trained on CIFAR-10 using each of the clipping methods (for which the performance can also be found in Table 1) and showed our method is the most precise one, while being much more efficient than the relatively accurate alternatives (Senderovich et al., 2022; Delattre et al., 2023). To further evaluate the precision of our clipping method in comparison to the other methods, we also did the same experiment with the ResNet-18 model trained on the MNIST dataset. As Figure 6a shows the precision results are similar to the plot we had for CIFAR-10, with the times per epoch slightly smaller for all the methods due to the smaller input size. To check if the same precision results hold for other models and how the running time changes for larger models, we performed the same experiment for the DLA models that are trained using each of the clipping methods (for which the performance on the test set and adversarial examples can be found in Table 2). As Figure 6b shows, our method is still the most precise one among the clipping methods, while still being much faster than the more accurate alternatives. Another interesting point is that the factor by which the running times have increased for the larger model is smaller for our method compared to the methods introduced by Senderovich et al. (2022) and Delattre et al. (2023).

Table 2: The accuracy on the test set and adversarial examples generated with PGD-(50) and CW for DLA trained on MNIST and CIFAR-10, for the models with their convolutional and dense layers clipped to 1. The accuracy of the original model on the test set, PGD-generated examples, and CW-generated examples for MNIST are $\mathbf{99.39 \pm 0.08}$, $\mathbf{0.35 \pm 0.48}$, and $\mathbf{50.78 \pm 6.58}$, respectively. For CIFAR-10, these values are $\mathbf{94.82 \pm 0.12}$, $\mathbf{21.67 \pm 1.42}$, and $\mathbf{11.19 \pm 1.35}$.

Method	**Test Set**	**PGD-(50)**	**CW**
		CIFAR-10	
Miyato et al. (2018)	95.06 ± 0.09	$\mathbf{23.62 \pm 1.27}$	$\mathbf{17.75 \pm 1.34}$
Gouk et al. (2021)	92.44 ± 0.14	19.39 ± 1.34	13.41 ± 1.08
Senderovich et al. (2022)	93.39 ± 2.30	20.37 ± 2.67	15.39 ± 0.87
Delattre et al. (2023)	93.66 ± 0.46	19.90 ± 2.47	14.92 ± 2.77
FastClip	$\mathbf{95.53 \pm 0.10}$	22.54 ± 1.02	17.14 ± 0.76
		MNIST	
Miyato et al. (2018)	99.40 ± 0.05	3.26 ± 2.95	78.59 ± 5.34
Gouk et al. (2021)	99.26 ± 0.06	3.13 ± 2.49	79.16 ± 7.07
Senderovich et al. (2022)	99.43 ± 0.03	14.03 ± 5.27	83.92 ± 2.33
Delattre et al. (2023)	99.34 ± 0.04	4.28 ± 2.11	68.02 ± 5.37
FastClip	$\mathbf{99.44 \pm 0.04}$	$\mathbf{16.74 \pm 7.09}$	$\mathbf{84.90 \pm 1.59}$

C.3 Generalization and Robustness

In Section 3.2.2 we showed that our clipping method helps to improve the generalization and robustness of the ResNet-18 model for both CIFAR-10 and MNIST datasets. As Table 1 shows, our clipping method works much better than the other clipping methods in this regard. In this section, we show the results for the DLA model on both of these datasets. An interesting observation with the DLA model was that they could not be trained at all without batch normalization layers with the default settings, even if the spectral norm is clipped using any of the clipping methods. Therefore, unlike the results for ResNet-18 we do not show the numbers for the `No BN` case (the test accuracies were almost the same as the random classifier in all cases). As Table 2 shows, our clipping method makes the most improvement in generalization on both datasets, and in terms of adversarial robustness, it is either better than the other methods or achieves competitive results. Note that unlike ResNet-18 models in which according to the experiments by Gouk et al. (2021) we divided the sum of the two layers in the residual modules to make the whole output 1-Lipschits at each layer, we used DLA models as they are designed without further modification. Since the DLA model contains modules in which the output of two layers are summed up to generate the inputs of the succeeding layer, making each one of the layers 1-Lipshitz does not make each module 1-Lipschitz. This might affect the robustness of models; however, we wanted to use this model as it is to show the benefits of our method applied to an original model without any modification.

C.4 Clipping Batch Norm

As explained in Appendix B.2, it is important to control the spectral norm of batch normalization layers. In this section, we investigate the performance of both ResNet-18 and DLA models on the test set, as well as their adversarial robustness, when the clipping method for batch norm layers introduced by Gouk et al. (2021) is used. These results are presented in Table 3. Moreover, we investigate the application of our clipping method to the concatenation of convolutional layers and their succeeding batch norm layers, rather than controlling the batch norm layers in isolation. We present the latter results, which is unique to our method, as `FastClip-concat` in Table 3. In this setting, we use a smaller value for λ (see Algorithm 2), and apply the clipping every 500 steps. As the results show, with the regular batch norm clipping introduced by Gouk et al. (2021) our method still achieves the best test accuracy among the models and increases the robustness compared to the original model, however, neither of the clipping methods can achieve generalization or adversarial robustness better than the `FastClip` model with the batch norm layers removed (see Table 1 and Table 2). This shows the previously suggested clipping method for batch norm layers, although theoretically bounding the Lipschitz constant of the

Table 3: The accuracy on the test set and adversarial examples generated with PGD (50), $\epsilon = 0.02$ of ResNet-18 trained on CIFAR-10, for the models with their convolutional and dense layers clipped to 1 (`With BN`). For all the models, except `FastClip-concat`, the batch norm layers are clipped to strictly 1 using the method by Gouk et al. (2021). `FastClip-concat` uses the FastClip method for controlling the batch norm of the concatenation of convolutional and batch norm layers, as described in Section 3.3. As the results show, this method does not impede the optimization of the model and leads to a much better test accuracy while making the models more robust compared to when batch norm layers are not taken into account during the clipping process (see Table 1 and Table 2).

Method	**Test Set**	**PGD-50** $\epsilon = 0.02$	**CW** $c = 0.02$
		ResNet-18 Model	
Miyato et al. (2018)	85.91 ± 0.27	17.63 ± 0.44	$\mathbf{49.94 \pm 2.25}$
Gouk et al. (2021)	27.33 ± 2.11	13.89 ± 0.68	11.23 ± 3.53
Senderovich et al. (2022)	69.27 ± 4.35	16.22 ± 1.86	25.27 ± 3.43
Delattre et al. (2023)	30.44 ± 3.59	12.99 ± 4.16	10.17 ± 6.63
FastClip	90.59 ± 0.36	$\mathbf{25.97 \pm 0.88}$	41.50 ± 2.02
FastClip-concat	$\mathbf{94.63 \pm 0.08}$	25.02 ± 1.56	33.77 ± 3.59
		DLA Model	
Miyato et al. (2018)	80.91 ± 0.95	16.29 ± 2.79	$\mathbf{40.64 \pm 5.34}$
Gouk et al. (2021)	29.35 ± 1.19	13.07 ± 3.13	11.95 ± 3.97
Senderovich et al. (2022)	74.94 ± 1.03	13.92 ± 1.25	34.14 ± 3.52
Delattre et al. (2023)	32.31 ± 4.38	12.18 ± 3.14	12.07 ± 5.45
FastClip	89.27 ± 0.25	23.40 ± 1.46	39.16 ± 2.76
FastClip-concat	$\mathbf{95.02 \pm 0.07}$	$\mathbf{25.93 \pm 1.31}$	28.16 ± 0.81

model, does not help us in practice. On the other hand, `FastClip-concat` helps us improve the robustness compared to the original model and `FastClip`, but still achieves an accuracy on the test set which is close to `FastClip` and much better than the models with their batch norm layers removed.