Towards Practical Control of Singular Values of Convolutional Layers

Alexandra Senderovich[*†] HSE University
Ekaterina Bulatova[*] HSE University
Anton Obukhov ETH Zürich
Maxim Rakhuba HSE University

Abstract

In general, convolutional neural networks (CNNs) are easy to train, but their essential properties, such as generalization error and adversarial robustness, are hard to control. Recent research demonstrated that singular values of convolutional layers significantly affect such elusive properties and offered several methods for controlling them. Nevertheless, these methods present an intractable computational challenge or resort to coarse approximations. In this paper, we offer a principled approach to alleviating constraints of the prior art at the expense of an insignificant reduction in layer expressivity. Our method is based on the tensor-train decomposition; it retains control over the actual singular values of convolutional mappings while providing structurally sparse and hardware-friendly representation. We demonstrate the improved properties of modern CNNs with our method and analyze its impact on the model performance, calibration, and adversarial robustness. The source code is available at: https://github.com/WhiteTeaDragon/practical_svd_conv

1 Introduction

Over the past decade, empirical advances in Deep Learning have made machine learning ubiquitous to researchers and practitioners from various fields of science and industry. However, the theory of Deep Learning lags behind its practical applications, resulting in unexpected outcomes or lack of explainability of the models. These adverse effects are becoming more prominent with many models put into customer-facing products, such as perception systems of self-driving cars, chatbots, and other applications. The demand for tighter control over deployed models has given rise to several subfields of Deep Learning, such as the study of adversarial robustness and model calibration, to name a few.

Convolutional neural networks (CNNs) and, in particular, residual CNNs have become a benchmark for various computer vision tasks. The key component of CNNs is convolutional layers, representing linear transformations that account for the image data structure. The singular values of these linear transformations are key to the properties of the whole network, such as the Lipschitz constant and, hence, generalization error and robustness to adversarial examples. Moreover, bounding the Lipschitz constant of convolutional layers can also improve training, as it prevents gradients from exploding.

Nevertheless, finding and controlling singular values of convolutional layers is challenging. Indeed, computing and then clipping exact singular values of a layer [Sedghi et al., 2019], given by a kernel tensor of the size $k \times k \times c_{\text{in}} \times c_{\text{out}}$, has time and space complexities of $\mathcal{O}(n^2c^2(c + \log n))$ and $\mathcal{O}(n^2c^2)$ respectively, where $n \times n$ is the input image size, c_{in} and c_{out} are the numbers of input and output channels respectively, $c = \max\{c_{\text{in}}, c_{\text{out}}\}$, and $k \times k$ is the filter size. Since typically $k^2 \ll c$, computing singular values requires many more operations than computing a single convolution with

[*]Equal contribution.

[†]Corresponding author: Alexandra Senderovich (AlexandraSenderovich@gmail.com)

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

its asymptotic complexity $\mathcal{O}(n^2c^2k^2)$. When computing singular values, the storage of an arising kernel tensor of the size $n \times n \times c_{\text{in}} \times c_{\text{out}}$ (padded to image dimensions) can also be an issue for larger networks and inputs. This effect becomes even more pronounced in three-dimensional convolutions used for volumetric data processing. Alternative methods that parametrize the convolutional layers are also computationally demanding [Singla and Feizi, 2021b, Li et al., 2019].

In this paper, we propose a practical approach to constraining the singular values of the convolutional layers based on intractable techniques from the prior art. Our approach relies on the assumption that modern over-parameterized neural networks can be made sparse using tensor decompositions, incurring insignificant degradation of the downstream task performance. We investigate the impact of using our approach on model performance, calibration, and adversarial robustness. More specifically, our contributions are as follows:

1. We propose a new framework for reducing the computational complexity of controlling singular values of a convolutional layer by imposing the Tensor-Train (TT) decomposition constraint on a kernel tensor. It allows for substituting the computation of singular values of the original layer by a smaller layer. As a result, it reduces the complexity of singular values control by using a method of choice.
2. We extend the formula of exact singular values of a convolutional layer [Sedghi et al., 2019] to the case of strided convolutions, which is ubiquitous in CNN architectures.
3. We apply the proposed framework to different methods of controlling singular values and test them on several CNN architectures. Contrary to measuring just the downstream task performance as in the prior art, we additionally demonstrate improvements in adversarial robustness and model calibration of the networks.

The paper is organized as follows: Sec. 2 provides an overview of prior art on singular values and Lipschitz constant estimation, related methods employing these techniques, and assumptions used to develop our approach; Sec. 3 revisits computation of singular values of convolutional layers and describes our approach to dealing with computational complexity; Sec. 4 walks through the steps of our algorithm to computing singular values of compressed convolutional layers; Sec. 5 contains the empirical study of our method; Sec. 6 wraps up the paper.

2 Related Work

2.1 Computing Singular Values of Convolutions

Traditional discrete convolutional layers used in signal processing and computer vision [LeCun et al., 1989] can be seen from two perspectives. On the one hand, a d-dimensional convolution linearly maps (potentially overlapping) windows of an c-channel input signal with side k into vectors of c output values (*window map*). This perspective is related to the one implementation of convolutional layers, decomposing the operation into window extraction termed `im2col` [Chellapilla et al., 2006], followed by matrix multiplication via `GEMM` [Blackford et al., 2002]. From this perspective, the transformation is defined by the convolution weight tensor unfolded into a matrix of size $k^2c_{\text{in}} \times c_{\text{out}}$ (`Conv2D` case); its singular values can be computed and controlled through SVD-based schemes over the said unfolding matrix. For example, Yoshida and Miyato [2017] proposed a method of Spectral Normalization to stabilize GAN training [Goodfellow et al., 2020]. Their method effectively estimates the largest singular value of the map via the power method. In a similar vein, Liu et al. [2019], Obukhov et al. [2021] explicitly parametrized the SVD decomposition of the mapping to control multiple singular values during training.

Nevertheless, the above perspective lacks the generality of treating the input signal as a whole, which is especially important when chaining maps, such as seen in deep CNNs. To this end, a discrete convolutional layer can be viewed as a mapping from and onto the space defined by the signal shape (*signal map*). The map is defined by an N-ly block-circulant matrix, where N is the number of spatial dimensions of the convolution. A straightforward way to compute singular values of such a matrix by applying a full SVD algorithm is prohibitively expensive, even for small numbers of channels, as well as signal and window sizes. To overcome this issue, Sedghi et al. [2019] proposed a method based on the Fast Fourier Transform (FFT) for computing exact singular values of convolutional layers that has a much better complexity than the naive approach. Nevertheless, it is still quite demanding in space

and time complexities, as it deals with a padded kernel of the size $n \times n \times c \times c$. Alternatively, Singla and Feizi [2021b] directly impose nearly orthogonal constraints on convolutional layers using Taylor expansion of a matrix exponential of a skew-Hermitian matrix, which appears to be more efficient than a Block Convolution Orthogonal Parameterization approach of Li et al. [2019] proposed earlier. Singla and Feizi [2021a] drew connections between both perspectives and proved that the Lipschitz constant of the window map could serve as a genuine upper bound of the Lipschitz constant of the signal map if multiplied by a certain constant.

2.2 Effects of Singular Values

The study of singular values and the closely related Lipschitzness of convolutional neural networks impacts many domains of Deep Learning research. One essential expectation about neural networks is to generalize input data instead of memorizing it. Bartlett et al. [2017] present the generalization bound that depends on the product of the largest singular values of layers' Jacobians. Gouk et al. [2021] performed a detailed empirical study of the influence of bounding individual Lipschitz constants of each layer and its effect on the generalization error of the whole CNN. Singular values of Jacobians greater or smaller than 1.0 can also be responsible for the growth or decay of gradients. Therefore, controlling all singular values can also be helpful to avoid exploding or vanishing gradients. The decay of singular values in layers also plays a role in the performance of GAN generators [Liu et al., 2019].

Apart from boosting the accuracy of predictions, researchers have been working on improving the robustness of neural networks to adversarial attacks, which is also affected by the Lipschitz constant [Cisse et al., 2017]. Adversarial attacks aim to find a negligible (to human perception) perturbation of input data that would sabotage the predictions of a model. In this paper, we use the AutoAttack Robust Accuracy module [Croce and Hein, 2020] to evaluate results. We additionally use a metric of classification model calibration (Expected Calibration Error, ECE) [Naeini et al., 2015]. Connections between model Lipschitz constraints, calibration, and out-of-distribution (OOD) model performance have been drawn in recent literature [Postels et al., 2022]. Various training techniques can also affect model properties [Kodryan et al., 2022] and efficiency of training [Gusak et al., 2022].

2.3 Tensor Decompositions in Deep Learning

Our proposed framework for CNN weights reparameterization relies on the well-studied property of neural network overparameterization [Frankle and Carbin, 2019]. Many network compression approaches [Anwar et al., 2017, Lee et al., 2019] agree that it is often possible to approach the uncompressed model performance by imposing some form of sparsity on the network weights. Tensor decompositions have been used previously for compression [Novikov et al., 2015, Garipov et al., 2016, Wang et al., 2018, Obukhov et al., 2020], multitask learning [Kanakis et al., 2020], neural rendering [Obukhov et al., 2022] and reinforcement learning [Sozykin et al., 2022]. The idea of this paper is to utilize tensor decomposition closely related to the SVD decomposition and to take advantage of its sparsity to reduce the complexity of controlling singular values. For that, we resort to the TT decomposition [Oseledets, 2011], which in the outlined context is also equivalent to the Tucker-2 decomposition [Kim et al., 2016]. As opposed to alternative tensor decompositions, this one allows us to naturally access singular values of the convolution layer with any method of choice.

3 Method-independent Complexity Reduction

This section explains how to reduce the complexity of calculating the singular values of a convolutional layer (in the *signal map* sense explained in Sec. 2.1), regardless of the chosen method. For the sake of generality, we consider layers acting on $(d{+}1)$-dimensional input arrays (tensors) with d spatial dimensions and 1 channel dimension. Of particular interest are cases $d = 2$ and $d = 3$, corresponding to regular images and volumetric data.

We bootstrap our approach based on the recent research on neural network sparsity by compressing convolutional layers with the TT decomposition. We focus on dealing with the channel-wise complexity of the methods employed. In what follows, Sec. 3.1 revisits the convolutional operator in neural networks; Sec. 3.2 introduces our principled approach to compressing convolutional layers and reducing the complexity of the problem by a significant margin.

3.1 Regular Convolutional Layers

To introduce a convolutional layer formally, let $\mathcal{K} \in \mathbb{R}^{k\times\cdots\times k\times c_{\text{in}}\times c_{\text{out}}}$ be a $(d{+}2)$-dimensional kernel tensor, where k is a filter size and $c_{\text{in}}, c_{\text{out}}$ are the numbers of input and output channels, respectively. A convolutional layer is given by a linear map $\mathscr{C}_{\mathcal{K}}$ – multichannel convolution with the kernel tensor $\mathcal{K}$ such that $\mathscr{C}_{\mathcal{K}}\colon \mathbb{R}^{c_{\text{in}}\times n\times\cdots\times n} \to \mathbb{R}^{c_{\text{out}}\times f(n,k)\times\cdots\times f(n,k)}$ and

$$(\mathscr{C}_{\mathcal{K}}(\mathcal{X}))_{j,:,\ldots,:} = \sum_{i=0}^{c_{\text{in}}-1} \mathcal{K}_{:,\ldots,:,i,j} * \mathcal{X}_{i,:,\ldots,:}, \qquad j = 1,\ldots,c_{\text{out}}, \tag{1}$$

where $f(n,k)$ is an integer, depending on the type of the convolution $'*'$ used and convolution parameters, e.g., strides. The key assumption needed for our derivations is the bilinearity of $'*'$, which covers different convolution types, e.g., linear, periodic, and correlation. For example, a widely used correlation-type convolution with strides equal to one, reads

$$\mathcal{Y}_{jq_1\ldots q_d} = \sum_{i=0}^{c_{\text{in}}-1} \sum_{p_1,\ldots,p_d=0}^{k-1} \mathcal{K}_{p_1\ldots p_d ij}\mathcal{X}_{i,q_1+p_1,\ldots,q_d+p_d},$$

for all $j = 0,\ldots,c_{\text{out}}-1$ and $q_\alpha = 0,\ldots,f(n,k)-1$, $\alpha = 1,\ldots,d$, where $f(n,k) = n-k+1$. In what follows in this section, we write

$$\mathcal{Y} = \mathscr{C}_{\mathcal{K}}(\mathcal{X}), \tag{2}$$

implying one of the convolution types mentioned above.

Using the linearity of $\mathscr{C}_{\mathcal{K}}$, we can rewrite (2) as a matrix-vector product:

$$\texttt{vec}(\mathcal{Y}) = T_{\mathcal{K}}\,\texttt{vec}(\mathcal{X}), \tag{3}$$

where $\texttt{vec}$ is a row-major reshaping of a multidimensional array into a column vector. In turn, $T_{\mathcal{K}} \in \mathbb{R}^{c_{\text{out}}n^d\times c_{\text{in}}n^d}$ is a matrix with an additional block structure: its $n^d \times n^d$ blocks are d-level Toeplitz matrices (see Sedghi et al. [2019] for $d = 2$). Representation (3) allows us to replace singular values of a linear map $\mathscr{C}_{\mathcal{K}}$ with singular values of a matrix $T_{\mathcal{K}}$, which is handy for analysis. In turn, access to singular values allows for controlling the Lipschitz constant of $\mathscr{C}_{\mathcal{K}}$ in terms of the largest singular value of $T_{\mathcal{K}}$, denoted as $\sigma_1(T_{\mathcal{K}})$. Indeed, since $\sigma_1(T_{\mathcal{K}}) = \|T_{\mathcal{K}}\|_2$, we have

$$\|\mathscr{C}_{\mathcal{K}}(\mathcal{X}) - \mathscr{C}_{\mathcal{K}}(\mathcal{Z})\|_F = \|T_{\mathcal{K}}(\texttt{vec}(\mathcal{X}) - \texttt{vec}(\mathcal{Z}))\|_2 \le \|T_{\mathcal{K}}\|_2\|\mathcal{X}-\mathcal{Z}\|_F = \sigma_1(T_{\mathcal{K}})\|\mathcal{X}-\mathcal{Z}\|_F.$$

3.2 Compressed Convolutional Layers

Recall that even if $d = 2$, finding the SVD of $T_{\mathcal{K}}$ requires $\mathcal{O}(n^2(c^2+\log n))$ FLOPs, which is too much for practical use with large networks. To reduce the computational cost of SVD of $T_{\mathcal{K}}$, we propose a low-rank compressed layer representation based on the following tensor decomposition:

$$\mathcal{K}_{p_1\ldots p_d ij} = \sum_{\alpha=0}^{r_1-1}\sum_{\beta=0}^{r_2-1} \mathcal{K}^{(1)}_{i\alpha}\,\mathcal{K}^{(2)}_{p_1\ldots p_d\alpha\beta}\,\mathcal{K}^{(3)}_{\beta j}, \tag{4}$$

where $\mathcal{K}^{(1)} \in \mathbb{R}^{c_{\text{in}}\times r_1}$, $\mathcal{K}^{(3)} \in \mathbb{R}^{r_2\times c_{\text{out}}}$ and $\mathcal{K}^{(2)} \in \mathbb{R}^{k\times\cdots\times k\times r_1\times r_2}$ are some small tensors and integers r_1, r_2: $1 \le r_1 \le c_{\text{in}}$, $1 \le r_2 \le c_{\text{out}}$ are called ranks. This decomposition is essentially the TT decomposition [Oseledets, 2011] of the kernel tensor with filter modes $p_1,\ldots,p_d$ merged into a single index. We also note that it is equivalent to the so-called Tucker-2 decomposition, successfully used by Kim et al. [2016] to compress large convolutional networks. The proposed decomposition is visualized in Fig. 1. We note that we chose the TT decomposition as it gives us access to the SVD decomposition of the convolution (Theorem 1). It is, however, unclear how to obtain similar results with other decompositions, such as tensor-ring or canonical tensor decomposition, as they do not admit SVD-like form, see, e.g., [Grasedyck et al., 2013] for more details of these formats.

By substituting (4) into (1), using bilinearity of $'*'$ and omitting summation limits for clarity:

$$(\mathscr{C}_{\mathcal{K}}(\mathcal{X}))_{j,:,\ldots,:} = \sum_i \mathcal{K}_{:,\ldots,:,i,j} * \mathcal{X}_{i,:,\ldots,:} = \sum_i \left(\sum_{\alpha,\beta} \mathcal{K}^{(1)}_{i\alpha}\,\mathcal{K}^{(2)}_{:,\ldots,:,\alpha,\beta}\,\mathcal{K}^{(3)}_{\beta j}\right) * \mathcal{X}_{i,:,\ldots,:} =$$

$$\sum_i\sum_{\alpha,\beta} \mathcal{K}^{(1)}_{i\alpha}\mathcal{K}^{(3)}_{\beta j}\left(\mathcal{K}^{(2)}_{:,\ldots,:,\alpha,\beta} * \mathcal{X}_{i,:,\ldots,:}\right) = \sum_\beta \mathcal{K}^{(3)}_{\beta j}\sum_\alpha \mathcal{K}^{(2)}_{:,\ldots,:,\alpha,\beta} * \left(\sum_i \mathcal{K}^{(1)}_{i\alpha}\mathcal{X}_{i,:,\ldots,:}\right),$$

we obtain that the convolution with kernel given by (4) is equivalent to a sequence of three convolutions without nonlinear activation functions between them:

1. $1 \times \cdots \times 1$ convolution with c_{in} input and r_1 output channels (kernel tensor – $\mathcal{K}^{(1)}$);
2. $k \times \cdots \times k$ convolution with r_1 input and r_2 output channels (kernel tensor – $\mathcal{K}^{(2)}$);
3. $1 \times \cdots \times 1$ convolution with r_2 input and c_{out} output channels (kernel tensor – $\mathcal{K}^{(3)}$);

or equivalently,

$$\mathscr{C}_{\mathcal{K}} = \mathscr{T}_{\mathcal{K}^{(3)}} \circ \mathscr{C}_{\mathcal{K}^{(2)}} \circ \mathscr{T}_{\mathcal{K}^{(1)}}, \tag{5}$$

where $\mathscr{A} \circ \mathscr{B}$ denotes the composition of two maps $\mathscr{A}$ and $\mathscr{B}$; $\mathscr{T}_{\mathcal{K}^{(i)}}$ denotes $1 \times \cdots \times 1$ convolution map given by a convolution kernel $\mathcal{K}^{(i)}$.

In the rest of this section, we show that singular values of the original convolutional layer with kernel $\mathcal{K}$ correspond to singular values of the compressed kernel $\mathcal{K}^{(2)}$. As a first step, we demonstrate that we can impose orthogonality constraints on $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(3)}$ without loss of expressivity.

Lemma 1. *Let $\mathcal{K} \in \mathbb{R}^{k \times \cdots \times k \times c_{\text{in}} \times c_{\text{out}}}$ be given by (4). Then there exist matrices $\mathcal{Q}^{(1)} \in \mathbb{R}^{c_{\text{in}} \times r_1}$, $\mathcal{Q}^{(3)} \in \mathbb{R}^{r_2 \times c_{\text{out}}}$ satisfying $\mathcal{Q}^{(1)\top}\mathcal{Q}^{(1)} = I_{r_1}$, $\mathcal{Q}^{(3)}\mathcal{Q}^{(3)\top} = I_{r_2}$ and a tensor $\mathcal{Q}^{(2)} \in \mathbb{R}^{k \times \cdots \times k \times r_1 \times r_2}$, such that*

$$\mathcal{K}_{p_1 \ldots p_d i j} = \sum_{\alpha=0}^{r_1-1} \sum_{\beta=0}^{r_2-1} \mathcal{Q}^{(1)}_{i\alpha} \, \mathcal{Q}^{(2)}_{p_1 \ldots p_d \alpha \beta} \, \mathcal{Q}^{(3)}_{\beta j}. \tag{6}$$

Proof. See Sec. A.1 of the Appendix. □

Now we have all preliminaries in place to formulate the key result that our framework is based on.

Theorem 1. *Let $\mathcal{K} \in \mathbb{R}^{k \times \cdots \times k \times c_{\text{in}} \times c_{\text{out}}}$ be given by (4). Let also $\mathcal{K}^{(1)} \in \mathbb{R}^{c_{\text{in}} \times r_1}$, $\mathcal{K}^{(3)} \in \mathbb{R}^{r_2 \times c_{\text{out}}}$:*

$$\mathcal{K}^{(1)\top}\mathcal{K}^{(1)} = I_{r_1}, \quad \mathcal{K}^{(3)}\mathcal{K}^{(3)\top} = I_{r_2}. \tag{7}$$

Then the multiset of nonzero singular values of $\mathscr{C}_{\mathcal{K}}$ defined in (1) equals the multiset of nonzero singular values of $\mathscr{C}_{\mathcal{K}^{(2)}}$.

Proof. See Sec. A.2 of the Appendix. □

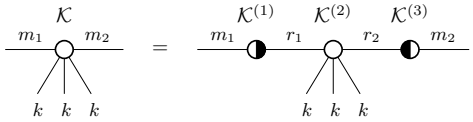

Figure 1: Tensor diagram representation of (4) for $d = 3$ with the constraints (7) as in Theorem 1. A node with D "legs" represents a D-dimensional tensor; individual legs represent indices; connected legs represent summation over the respective indices; half-filled nodes are orthogonal matrices.

Theorem 1 suggests that after forcing the orthogonality of $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(3)}$, one only needs to find singular values of the layer corresponding to the smaller kernel tensor $\mathcal{K}^{(2)}$ of the size $n \times n \times r_1 \times r_2$ instead of the original kernel tensor $\mathcal{K}$ of the size $n \times n \times r_1 \times r_2$. At the same time, thanks to Lemma 1, one may always set $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(3)}$ to be orthogonal without additionally losing the expressivity of $\mathscr{C}_{\mathcal{K}}$. Fig. 1 demonstrates the resulting decomposition using tensor diagram notation.

4 Application of the Proposed Framework

This section discusses how to apply the proposed framework to various methods. We start with the explicit formula for singular values computation and then discuss how to apply the framework when the convolutions are parametrized, for example, as in [Singla and Feizi, 2021b, Li et al., 2019].

Algorithm 1 Singular values of a TT-compressed periodic convolutional layer ($d = 2$).

Input:
$\mathcal{K}^{(1)} \in \mathbb{R}^{c_{\text{in}} \times r_1}$ – left core tensor of the TT-compressed kernel $\mathcal{K}$,
$\mathcal{K}^{(2)} \in \mathbb{R}^{k \times k \times r_1 \times r_2}$ – middle core tensor of the TT-compressed kernel $\mathcal{K}$,
$\mathcal{K}^{(3)} \in \mathbb{R}^{r_2 \times c_{\text{in}}}$ – right core tensor of the TT-compressed kernel $\mathcal{K}$,
n – input image size,
s – stride of the convolution defined by kernel $\mathcal{K}$ s.t. $n \equiv 0 \pmod{s}$.

Output:
Non-zero singular values of the convolutional layer, defined by the kernel tensor $\mathcal{K}$.

1: $Q^{(1)}, R^{(1)} \leftarrow \texttt{QR}\left(\mathcal{K}^{(1)}\right)$ ▷ Perform QR decomposition of the left core
2: $Q^{(3)}, R^{(3)} \leftarrow \texttt{QR}\left(\mathcal{K}^{(3)\top}\right)$ ▷ Perform QR decomposition of the right core
3: $\widehat{\mathcal{K}}^{(2)}_{p_1,p_2,:,:} \leftarrow R^{(1)} \mathcal{K}^{(2)}_{p_1,p_2,:,:} R^{(3)\top}, \ \forall p_1, p_2$ ▷ Absorb non-orthogonal factors in the middle core
4: $\widehat{\mathcal{K}}^{(2)} \leftarrow \texttt{pad_zeros}\left(\widehat{\mathcal{K}}^{(2)}, \ \texttt{shape=}(n \times n \times r_1 \times r_2)\right)$ ▷ Pad middle core to the image size
5: $R \leftarrow \texttt{reshape}\left(\widehat{\mathcal{K}}^{(2)}, \ \texttt{shape=}\ (8)\right)$ ▷ Reshape middle core as in (8). See also Sec. B.1.
6: $P^{(:,:)}_{ij} \leftarrow \texttt{fft2}\left(R_{j \bmod s^2,:,:,i,\lfloor j/s^2 \rfloor}\right), \ \forall\, i, j$ ▷ Perform 2-dimensional FFT of slices of R
7: **return** $\bigcup\limits_{p_1,p_2 \in \{1,\ldots,\frac{n}{s}\}} \sigma\left(P^{(p_1,p_2)}\right)$ ▷ Return a union of all singular values of $P^{(\cdot,\cdot)}$

4.1 Explicit Formulas for Strided Convolutions

Sedghi et al. [2019] consider periodic convolutions, allowing them to express singular values through discrete Fourier transforms and several SVDs. We extend the main result of their work to the case of strided convolutions and summarize it in the following theorem.

Theorem 2. *Let $\widehat{\mathcal{K}} \in \mathbb{R}^{n \times n \times c_{\text{in}} \times c_{\text{out}}}$ be a kernel $\mathcal{K} \in \mathbb{R}^{k \times k \times c_{\text{in}} \times c_{\text{out}}}$ of a periodic convolution, padded with zeros along the filter modes. Let s be a stride of the convolution $\mathscr{C}_{\mathcal{K}}$: $n \equiv 0 \pmod{s}$ and let the reshaped kernel $R \in \mathbb{R}^{s^2 \times \frac{n}{s} \times \frac{n}{s} \times c_{\text{in}} \times c_{\text{out}}}$ be such that:*

$$R_{q,a,b,i,j} = \widehat{\mathcal{K}}_{\lfloor q/s \rfloor + as,(q \bmod s)+bs,i,j}. \tag{8}$$

Let us consider $\left(\frac{n}{s}\right)^2$ matrices $P^{(p_1,p_2)} \in \mathbb{R}^{c_{\text{in}} \times s^2 c_{\text{out}}}$ with entries:

$$P^{(p_1,p_2)}_{ij} = \left(F^\top R_{j \bmod s^2,:,:,i,\lfloor j/s^2 \rfloor} F\right)_{p_1 p_2}, \tag{9}$$

where F is an $n \times n$ Fourier matrix. Then the multiset of singular values of $T_{\mathcal{K}}$ is as follows:

$$\sigma\left(T_{\mathcal{K}}\right) = \bigcup_{p_1,p_2 \in \{1,\ldots,\frac{n}{s}\}} \sigma\left(P^{(p_1,p_2)}\right). \tag{10}$$

Proof. See Sec. B.2 of the Appendix. □

The algorithm for finding the singular values of a TT-compressed convolutional layer based on Theorem 2 is summarized in Alg. 1. This algorithm consists of several major parts. We start with a layer with the imposed TT structure as in (4). We reduce this decomposition to the form with orthogonality conditions (7) by using the QR decomposition (steps 1–3). The second part is the application of Theorem 2 to the smaller kernel tensor $\widehat{\mathcal{K}}^{(2)}$ (steps 4–7). For detailed pseudocode of applying Theorem 2, see Sec. B.1.

Computing singular values using (9) and (10) has the complexity $\mathcal{O}(n^2 c^2 (cs^2 + \log \frac{n}{s}))$, $c = \max\{c_{\text{in}}, c_{\text{out}}\}$, which depends cubically on c. Thus, reducing c can significantly reduce the computational cost of finding the singular values. Let us now use the TT-representation (4) for kernel tensor with the ranks $r_1 = r_2 = r$. Theorem 1 suggests that we only need to apply Theorem 2 to the core $\mathcal{K}^{(2)} \in \mathbb{R}^{k \times k \times r_1 \times r_2}$, which leads to the complexity $\mathcal{O}(n^2 r^2 (r + \log \frac{n}{s}))$. For example, $r = c/2$ provides a theoretical speedup of up to 8 times (see Fig. 2 for practical evaluation). Memory consumption can also be high with the method of Sedghi et al. [2019]. Processing the padded kernel tensor of the size $n \times n \times c \times c$ with our method requires 4 times less memory for $r = c/2$ and 9 times less for $r = c/3$ (see also Fig. 2).

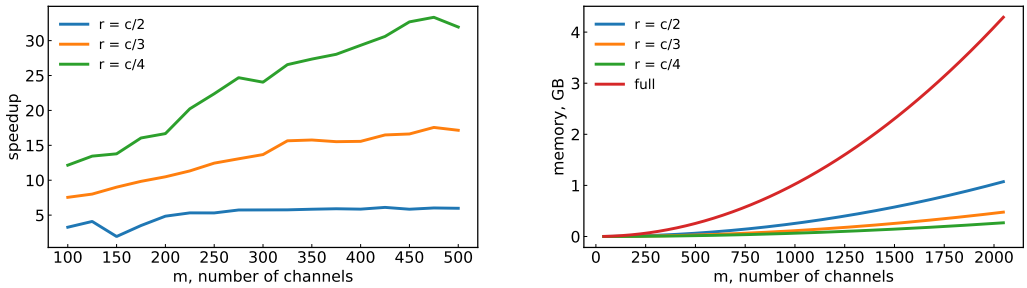

Figure 2: Left: the speedup of computing singular values of a TT-compressed layer using Alg. 1 relative to an uncompressed one. Right: Memory (single precision) to store the full $n \times n \times c \times c$ kernel and its TT-compressed form. Hyperparameter values: $n = 16$ and $s = 1$.

4.2 Orthogonality Constraints

To ensure that the frame matrices $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(3)}$ are orthogonal, we apply QR decomposition to the kernel in Alg. 1. However, we can use other methods to maintain the orthogonality of frame matrices; for example, we can impose regularization on the layers. Regularization is quite helpful in case $\mathcal{K}^{(2)}$ is not a standard convolution but has a special complex structure, such as the SOC layer from [Singla and Feizi, 2021b]. For that, let n_l denote the number of TT-decomposed layers in a network, $\mathcal{K}_i^{(1)}$ and $\mathcal{K}_i^{(3)}$ – left and right TT-cores of the ith layer, and $r_{1,i}, r_{2,i}$ – layer ranks. Then, the orthogonal loss reads as:

$$\text{loss}_{\text{ort}} = \left(\sum_{i=0}^{n_l} \left\| \mathcal{K}_i^{(1)\top} \mathcal{K}_i^{(1)} - I_{r_{1,i}} \right\|_F^2 + \left\| \mathcal{K}_i^{(3)} \mathcal{K}_i^{(3)\top} - I_{r_{2,i}} \right\|_F^2 \right) \Bigg/ \left(\sum_{i=0}^{n_l} r_{1,i}^2 + r_{2,i}^2 \right). \quad (11)$$

In the case of applying this regularization, we sum this loss with cross-entropy:

$$\text{loss} = \text{loss}_{\text{CE}} + \lambda_{\text{ort}} \text{loss}_{\text{ort}}. \quad (12)$$

The value of loss_{ort} is the MSE measure of non-orthogonality of all $\mathcal{K}_i^{(1)}$ and $\mathcal{K}_i^{(3)}$, so the coefficient λ_{ort} tends to be large. We set it to 1e5 for experiments with WideResNets.

5 Experiments

To study the effects of the proposed regularization on the performance of CNNs, we combined it with various methods of controlling singular values and applied it during training on the image classification task on the CIFAR-10 and CIFAR-100 datasets [Krizhevsky et al., 2009]. The code was implemented in PyTorch [Paszke et al., 2019] and executed on a NVIDIA Tesla V100, 32GB. In our experiments, we used the LipConvNet architecture [Singla and Feizi, 2021b], a WideResNet [Zagoruyko and Komodakis, 2016], and VGG-19 [Simonyan and Zisserman, 2015]. Furthermore, the ranks are tested in the range $[c/4, c/2]$, where c is the largest number of channels among all convolutions (except for 1×1), which we found to give the better performance/speedup trade-off.

Evaluation Metrics We consider four main metrics: standard accuracy on the test split, accuracy on the CIFAR-C dataset (CC) [Hendrycks and Dietterich, 2019][1], accuracy after applying the AutoAttack module (AA) [Croce and Hein, 2020], and the Expected Calibration Error (ECE) [Guo et al., 2017]. The last columns of tables with the results of our experiments contain compression ratios of all layers in the network. In addition, we reported a dedicated metric for the LipConvNet architecture: p.r. – provable bound-norm robustness, introduced by Li et al. [2019]. This metric shows the percentage of input images guaranteed to be predicted correctly despite any perturbation within radius $36/255$.

Table 1: Comparison between SOC [Singla and Feizi, 2021b] and SOC combined with the proposed framework using LipConvNet-N on CIFAR-10. "Speedup" is the speedup of training w.r.t. the SOC baseline. "Comp." (compression) is the ratio between the number of parameters of convolutional layers in the original and the decomposed networks. Despite speedups not being as substantial as in other experiments, we observe consistent improvement in all metrics, which is discussed in Fig. 3.

RANK	N	ACC. ↑	AA ↑	CC ↑	ECE ↓	P.R. ↑	SPEEDUP↑	COMP.↑
–	5	75.6 ± 0.3	31.1 ± 0.2	67.3 ± 0.1	8.6 ± 0.4	59.1 ± 0.1	1.0	1.0
128	5	76.9 ± 0.2	32.8 ± 0.5	68.8 ± 0.2	6.7 ± 0.1	62.7 ± 0.1	1.4	2.9
256	5	**78.3 ± 0.1**	**34.7 ± 0.6**	**69.7 ± 0.2**	**5.3 ± 0.2**	**65.4 ± 0.3**	0.9	1.2
–	20	76.3 ± 0.5	33.4 ± 0.5	68.1 ± 0.2	7.5 ± 0.1	61.4 ± 0.2	1.0	1.0
128	20	76.8 ± 0.2	33.0 ± 0.1	68.3 ± 0.1	5.9 ± 0.2	62.4 ± 0.2	1.2	3.3
256	20	**78.4 ± 0.2**	**35.4 ± 0.5**	**70.4 ± 0.1**	**4.7 ± 0.4**	**65.6 ± 0.2**	1.1	1.4
–	30	76.3 ± 1.0	32.0 ± 1.6	67.9 ± 1.0	7.3 ± 1.0	61.9 ± 1.2	1.0	1.0
128	30	76.0 ± 0.1	32.6 ± 0.1	68.0 ± 0.1	5.0 ± 0.6	61.7 ± 0.6	1.3	3.5
256	30	**77.9 ± 0.1**	**34.7 ± 0.3**	**69.7 ± 0.3**	**4.8 ± 0.1**	**64.8 ± 0.2**	1.1	1.5

5.1 Experiments on LipConvNet

Singla and Feizi [2021b] introduced a new architecture called Lipschitz Convolutional Networks, or LipConvNet for short. LipConvNet-N consists of N convolutional layers and activations. This architecture is provably 1-Lipschitz by contrast to popular residual network architectures. In Singla and Feizi [2021b], the authors also propose an orthogonal layer called Skew Orthogonal Convolution (SOC) and applied it in LipConvNet.

We modify the SOC layer in correspondence with the proposed framework by adding a $1 \times 1 \times c_{\text{in}} \times r$ convolution before applying it, and finally applying the last convolution of size $1 \times 1 \times r \times c_{\text{out}}$, thus reducing the number of channels to $r \times r$ in the bottleneck. To maintain the orthogonality of this new layer that we call SOC-TT, we add orthogonal loss (Sec. 4.2) to keep these two frame matrices orthogonal.

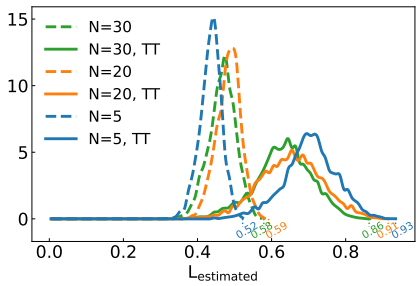

Figure 3: Histograms of empirically estimated Lipschitz constants of LipConvNet-N architectures (see Sec. C.1). When applying the TT method to LipConvNet architecture, the peaks are wider and closer to 1, giving more flexibility to the model but still keeping constants below 1. Rank parameter: $r = 256$.

Tab. 1 demonstrates metrics improvement after replacing SOC with SOC-TT. For rank 256, we performed a grid search of the λ_{ort} coefficient. The training protocol is the same as in Singla and Feizi [2021b] except for the learning rate of LipConvNet-20 with the SOC-TT layer, which we set to 0.05 instead of 0.1. To investigate the observed increase in metrics, we present histograms of empirically estimated Lischitz constants by using adversarial attacks in Fig. 3 (see Sec. C.1) with optimally chosen λ_{ort} from (12). We observed that increasing λ_{ort} forces the frame matrices in the TT decomposition to be closer to orthogonal and, as a result, the histogram converges to that of the original SOC method. At the same time, decreasing λ_{ort} may lead to networks with constants larger than 1 but with better metrics.

5.2 Experiments with WideResNet-16-10

Next, we consider WRN-16-10 [Zagoruyko and Komodakis, 2016] with almost 17M parameters. The results of our experiments on CIFAR-10 are summarised in Tab. 2. Additional experiments on CIFAR-100 are presented in Sec. C.3. The experimental setup and training schedule are taken from Gouk et al. [2021]. We train our models with batches of 128 elements for 200 epochs, using SGD optimizer and a learning rate of 0.1, multiplied by 0.1 after 60, 120, and 160 epochs. The hyperparameters for

[1]Distributed under Apache License 2.0

Table 2: Performance metrics for different constraints applied to WRN-16-10 trained on CIFAR-10. Both clipping and division combined with TT layers lead to an increase in metrics (except for ECE), with 256 being the best overall rank. Clipping tends to be a better option in terms of metrics but is substantially slower than the division approach. Moreover, clipping the singular values to 2 results in better performance than clipping to 1. "Speedup" is a speedup of the overhead resulting from singular values control in all the layers in the network. Clipping the whole network w/o decomposition takes 6.2 min while applying division – 0.6 sec. "Comp." (compression) is the ratio between the number of parameters of convolutional layers in the original (≈ 16.8M) and decomposed networks.

Method	Rank	Acc. ↑	AA ↑	CC ↑	ECE ↓	Speedup ↑	Comp. ↑
Baseline	–	**95.52**	54.22	72.91	**2.06**	–	1.0
	192	95.21	54.97	**73.94**	2.84	–	3.6
	256	95.39	**55.07**	73.73	2.73	–	2.4
	320	95.02	52.32	72.86	2.85	–	1.8
Clip to 1	–	**95.27**	53.27	**73.44**	2.45	1.0	1.0
	192	95.25	51.7	73.37	2.49	4.1	3.6
	256	95.12	51.94	72.47	**2.43**	3.3	2.4
	320	95.05	**54.44**	73.38	2.7	2.3	1.8
Clip to 2	–	**95.99**	55.77	73.27	**1.92**	1.0	1.0
	192	95.45	55.45	73.26	2.55	4.1	3.6
	256	95.73	**55.83**	73.35	2.28	3.3	2.4
	320	95.5	54.63	**74.34**	2.53	2.3	1.8
Division	–	95.17	53.39	72.94	2.54	1.0	1.0
	192	**95.71**	53.4	**74.68**	**2.4**	2.3	3.6
	256	95.7	**56.28**	73.43	2.51	1.5	2.4
	320	94.93	52.35	70.89	2.97	1.3	1.8

the experiments with division were taken from Gouk et al. [2021]. The orthogonal loss (11) was used in all experiments with decomposed networks.

Clipping Following Sedghi et al. [2019], we apply the so-called clipping of singular values after computing the singular values of a layer using Theorem 2. In particular, we first fix a threshold parameter δ, chosen to be 1 or 2 in numerical experiments. Then all singular values greater than δ are replaced with δ. This procedure allows for maintaining maximal singular values at the desired level. After the clipping, we reconstruct the expanded kernel $\widehat{\mathcal{K}}_\delta$ of the size $n \times n \times c_{\text{in}} \times c_{\text{out}}$, which is no longer sparse. Finally, we revert to its original shape by setting $\mathcal{K} = \widehat{\mathcal{K}}_\delta(1{:}k, 1{:}k, :, :)$. This operation is performed every 100 training iterations; we provide the code for computing singular values in Sec. B.1 of the Appendix. The results suggest that applying TT decomposition not only compresses the network by decreasing the number of parameters of its convolutional layers, but also speeds up the clipping operation. Network robustness increased when training decomposed layers with clipping, which is confirmed by the results of the AutoAttack module.

Division Even though we can bound the Lipschitz constants of all convolutional layers, the Lipschitz constant of the whole network can be arbitrary. In particular, for a WideResNet architecture, the Lipschitz constant would still be unconstrained due to residual connections, fully connected layers, batch normalization, and average pooling layers of the network. Gouk et al. [2021] derived a formula for the Lipschitz constants of batch normalization layers and proposed regularizing them too. After each training step, they set the desired values for Lipschitz constants of convolutional, fully connected, and batch normalization layers. To control the Lipschitz constant of convolutional and fully connected layers, they first compute the estimate of the largest singular value of a layer via power iteration. Then they divide the layer by the estimate and multiply it by the desired Lipschitz constant. A similar operation is performed for batch normalization layers, but the Lipschitz constant is computed directly.

In the original paper, this approach proved to be successful for WideResNet-16-10. The method itself already increases the robust metrics and accuracy substantially. The pattern remains intact after applying our decomposition to the layers: the robust accuracy of the AutoAttack module improves. Overall, the TT decomposition, applied without any other regularization, performs poorly. However,

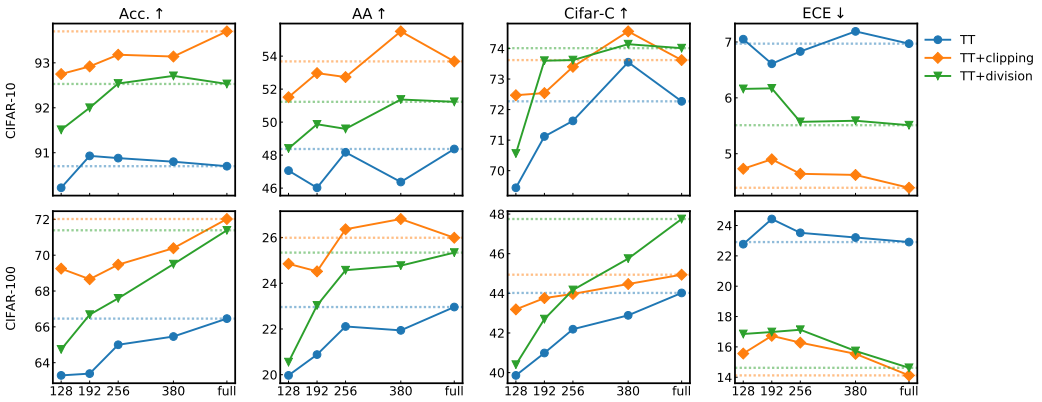

Figure 4: Metrics for different ranks on CIFAR-10 and CIFAR-100; "full" stands for the uncompressed case, i.e., "full" label with TT method is simply the baseline with no TT decomposition and no control of singular values, similarly "full" TT+clipping is [Sedghi et al., 2019] and "full" TT+division is [Gouk et al., 2021]. Horizontal dotted lines additionally represent uncompressed baselines.

when used to facilitate the application of a particular method of control over the singular values, the decomposition only slightly decreases the accuracy, which might still exceed that of the baseline.

5.3 Experiments with VGG-19

This section presents the results of controlling singular values on the VGG-19 network and two datasets: CIFAR-10 and CIFAR-100. As previously, we consider the clipping method with the exact computation of singular values [Sedghi et al., 2019] and the application of one iteration of the power method [Gouk et al., 2021]. The clipping parameter is set to 0.5. The results are presented in Fig. 4. Firstly, we observe that all the considered robust metrics benefit from controlling singular values. This effect is more pronounced than for the WRN-16-10 and was also observed in [Gouk et al., 2021]. We conjecture that this happens due to the lack of residual connections in VGG-type architectures, which makes them more sensitive to singular values of convolutional layers; in residual networks, convolutional layers are only corrections to the signal. In almost all scenarios, we observed that the clipping method performed better than the division with the power method.

6 Conclusion

We proposed a principled sparsity-driven approach to controlling the singular values of convolutional layers. We investigated two different families of CNNs and analyzed their performance and robustness under the proposed singular values constraint. The limitations of our approach are partly inherited from the encapsulated methods of computing singular values (e.g., the usage of periodic convolutions) and partly stem from the core assumption of CNN layer sparsity. Overall, our approach has proven effective for large-scale networks with hundreds of input and output channels, making it stand out compared to intractable methods from the prior art.

Societal Impact As our work is concerned with improving the robustness of neural networks, it positively impacts their reliability. Nevertheless, due to the generality of our approach, it can be used in various application domains, including malicious purposes.

Acknowledgments

The publication was supported by the grant for research centers in the field of AI provided by the Analytical Center for the Government of the Russian Federation (ACRF) in accordance with the agreement on the provision of subsidies (identifier of the agreement 000000D730321P5Q0002) and the agreement with HSE University №70-2021-00139. The calculations were performed in part through the computational resources of HPC facilities at HSE University [Kostenetskiy et al., 2021].

References

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.*, 13(3), feb 2017. ISSN 1550-4832. doi: 10.1145/3005348.

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.

L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006.

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

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry P. Vetrov. Ultimate tensorization: compressing convolutional and FC layers alike. *CoRR*, abs/1611.03214, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2015.

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.

Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

Julia Gusak, Daria Cherniuk, Alena Shilova, Alexandr Katrutsa, Daniel Bershatsky, Xunyi Zhao, Lionel Eyraud-Dubois, Oleh Shliazhko, Denis Dimitrov, Ivan Oseledets, and Olivier Beaumont. Survey on efficient training of large neural networks. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5494–5501, 7 2022. doi: 10.24963/ijcai.2022/769.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. On manifolds of tensors of fixed tt-rank. *Numerische Mathematik*, 120(4):701–731, 2012.

A. K. Jain. *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Menelaos Kanakis, David Bruggemann, Suman Saha, Stamatios Georgoulis, Anton Obukhov, and Luc Van Gool. Reparameterizing convolutions for incremental multi-task learning without task interference. In *Computer Vision - ECCV 2020*, volume 12365, pages 689–707. Springer, 2020.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *4th International Conference on Learning Representations, ICLR 2016*, 05 2016.

Maxim Kodryan, Ekaterina Lobacheva, Maksim Nakhodnov, and Dmitry Vetrov. Training scale-invariant neural networks on the sphere can happen in three regimes. In *Advances in Neural Information Processing Systems, 35*, 2022.

PS Kostenetskiy, RA Chulkevich, and VI Kozyrev. Hpc resources of the higher school of economics. In *Journal of Physics: Conference Series*, volume 1740, page 012050, 2021.

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

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Jörn-Henrik Jacobsen. Preventing gradient attenuation in lipschitz constrained convolutional networks. *Advances in neural information processing systems*, 32, 2019.

Kanglin Liu, Wenming Tang, Fei Zhou, and Guoping Qiu. Spectral regularization for combating mode collapse in gans. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6382–6390, 2019.

Mahdi Pakdaman Naeini, Gregory F Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *AAAI*, page 2901–2907, 2015.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Anton Obukhov, Maxim Rakhuba, Stamatios Georgoulis, Menelaos Kanakis, Dengxin Dai, and Luc Van Gool. T-basis: a compact representation for neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 7392–7404. PMLR, 2020.

Anton Obukhov, Maxim Rakhuba, Alexander Liniger, Zhiwu Huang, Stamatios Georgoulis, Dengxin Dai, and Luc Van Gool. Spectral tensor train parameterization of deep learning layers. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130, pages 3547–3555. PMLR, 2021.

Anton Obukhov, Mikhail Usvyatsov, Christos Sakaridis, Konrad Schindler, and Luc Van Gool. Tt-nf: Tensor train neural fields, 2022.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5): 2295–2317, 2011.

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 H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Janis Postels, Mattia Segù, Tao Sun, Luca Daniel Sieber, Luc Van Gool, Fisher Yu, and Federico Tombari. On the practicality of deterministic epistemic uncertainty. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 17870–17909. PMLR, 2022.

Amartya Sanyal, Philip H. Torr, and Puneet K. Dokania. Stable rank normalization for improved generalization in neural networks and gans. In *International Conference on Learning Representations*, 2020.

Hanie Sedghi, Vineet Gupta, and Philip M. Long. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Sahil Singla and Soheil Feizi. Fantastic four: Differentiable and efficient bounds on singular values of convolution layers. In *International Conference on Learning Representations*, 2021a.

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

Konstantin Sozykin, Andrei Chertkov, Roman Schutski, Anh-Huy Phan, Andrzej Cichocki, and Ivan Oseledets. TTOpt: A maximum volume quantized tensor train-based optimization and its application to reinforcement learning. In *Advances in Neural Information Processing Systems, 35*, 2022.

Wenqi Wang, Yifan Sun, Brian Eriksson, Wenlin Wang, and Vaneet Aggarwal. Wide compression: Tensor ring nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9329–9338, 2018.

Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning, 2017.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Subset of experiments
   (d) Did you include the total amount of compute and the type of resources 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...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Towards Practical Control of Singular Values of Convolutional Layers: Supplementary Materials

Alexandra Senderovich[*†] HSE University
Ekaterina Bulatova[*] HSE University
Anton Obukhov ETH Zürich
Maxim Rakhuba HSE University

A Compressed Convolutional Layers

A.1 Proof of Lemma 1

The result essentially follows from different ways to represent the TT decomposition [Holtz et al., 2012]. By applying the QR decomposition to $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(3)\top}$, we obtain:

$$\mathcal{K}^{(1)} = \mathcal{Q}^{(1)} R^{(1)}, \quad \mathcal{K}^{(3)\top} = \mathcal{Q}^{(3)\top} R^{(3)},$$

where $R^{(1)}$, $R^{(3)}$ are upper triangular. Substituting these formulas into (4) yields (6) with

$$\mathcal{Q}^{(2)}_{p_1 \dots p_d \alpha \beta} = \sum_{\alpha'=0}^{r_1-1} \sum_{\beta'=0}^{r_2-1} R^{(1)}_{\alpha\alpha'} \mathcal{K}^{(2)}_{p_1 \dots p_d \alpha' \beta'} R^{(3)}_{\beta'\beta},$$

which completes the proof.

A.2 Proof of Theorem 1

Using (5), we have

$$T_{\mathcal{K}} = T_{\mathcal{K}^{(3)}} T_{\mathcal{K}^{(2)}} T_{\mathcal{K}^{(1)}}.$$

Let us first show that given $\mathcal{K}^{(1)\top}\mathcal{K}^{(1)} = I_{r_1}$, the matrix $T_{\mathcal{K}^{(1)}}$ has orthonormal rows, i.e., it satisfies

$$T_{\mathcal{K}^{(1)}} T_{\mathcal{K}^{(1)}}^{\top} = I_{r_1 n^d}. \tag{13}$$

To do so, let us find $T_{\mathcal{K}^{(1)}}$ in terms of $\mathcal{K}^{(1)}$. For any $\mathcal{X} \in \mathbb{R}^{c_{\text{in}} \times n \times \dots \times n}$ and its row-major reshaping into a matrix $X \in \mathbb{R}^{c_{\text{in}} \times n^d}$, we have

$$\begin{aligned} T_{\mathcal{K}^{(1)}} \texttt{vec}(\mathcal{X}) &\equiv \texttt{vec}\left(\mathscr{C}_{\mathcal{K}^{(1)}}(\mathcal{X})\right) \\ &= \texttt{vec}\left(\mathcal{K}^{(1)} X\right) = \left(\mathcal{K}^{(1)} \otimes I_{n^d}\right) \texttt{vec}\left(X\right) \\ &= \left(\mathcal{K}^{(1)} \otimes I_{n^d}\right) \texttt{vec}\left(\mathcal{X}\right). \end{aligned}$$

Therefore, $T_{\mathcal{K}^{(1)}} = \mathcal{K}^{(1)} \otimes I_{n^d}$, where $\otimes$ denotes the Kronecker product of matrices. Using basic Kronecker product properties and the orthogonality of $\mathcal{K}^{(1)}$, we arrive at (13). Analogously, we may obtain $T_{\mathcal{K}^{(3)}} = \mathcal{K}^{(3)\top} \otimes I_{f(n,k)^d}$ and

$$T_{\mathcal{K}^{(3)}}^{\top} T_{\mathcal{K}^{(3)}} = I_{c_{\text{out}} f(n,k)^d}.$$

[*]Equal contribution.
[†]Corresponding author: Alexandra Senderovich (AlexandraSenderovich@gmail.com)

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Finally, using SVD of $T_{\mathcal{K}^{(2)}}$: $T_{\mathcal{K}^{(2)}} = U\Sigma V^\top$, we get:

$$\begin{aligned} T_{\mathcal{K}} &= \left(\mathcal{K}^{(3)\top} \otimes I_{f(n,k)^d}\right) U\Sigma V^\top \left(\mathcal{K}^{(1)\top} \otimes I_{n^d}\right) \\ &= \left(\left(\mathcal{K}^{(3)\top} \otimes I_{f(n,k)^d}\right) U\right) \Sigma \left(\left(\mathcal{K}^{(1)} \otimes I_{n^d}\right) V\right)^\top = \widetilde{U}\Sigma\widetilde{V}^\top, \end{aligned}$$

where $\widetilde{U}, \widetilde{V}$ have orthonormal columns as a product of matrices with orthonormal columns. Hence, $T_{\mathcal{K}} = \widetilde{U}\Sigma\widetilde{V}^\top$ is in the compact SVD form, which completes the proof.

B Periodic Strided Convolutions

B.1 Code for Computing Singular Values of a Strided Convolution

According to Theorem 2, the following code computes the singular values of a transform encoded by a strided convolution. Note that neither the full SVD nor the clipping operation is included in the code. The code for computing the singular vectors and the new kernel with constrained singular values can be found in the source code repository.

```
def SingularValues(kernel, input_shape, stride):
 1  kernel_tr = np.transpose(kernel, axes=[2, 3, 0, 1])
 2  d1 = input_shape[0] - kernel_tr.shape[2]
 3  d2 = input_shape[1] - kernel_tr.shape[3]
 4  kernel_pad = np.pad(kernel_tr, ((0, 0), (0, 0), (0, d1), (0, d2)))
 5  str_shape = input_shape // stride
 6  r1, r2 = kernel_pad.shape[:2]
 7  transforms = np.zeros((r1, r2, stride**2, str_shape[0], str_shape[1]))
 8  for i in range(stride):
 9      for j in range(stride):
10          transforms[:, :, i*stride+j, :, :] = \
11                  kernel_pad[:, :, i::stride, j::stride]
12      transforms = np.fft.fft2(transforms)
13      transforms = transforms.reshape(r1, -1, str_shape[0], str_shape[1])
14      transpose_for_svd = np.transpose(transforms, axes=[2, 3, 0, 1])
15      sing_vals = svd(transpose_for_svd, compute_uv=False).flatten()
16      return sing_vals
```

B.2 Proof of Theorem 2

In this section, we prove Theorem 2 from Sec. 4.1. Firstly, we analyze the structure of the matrix corresponding to a strided convolution. Secondly, we show that the columns of this matrix can be permuted to make matrix structure similar to that of a non-strided convolution. Therefore, the new theorem can be reduced to the already proven theorem.

Let us denote a circulant with each row shifted by the value of stride as a "strided circulant". The shape of such a strided circulant is $\frac{n}{s} \times n$, where n is the number of elements in the first row. Here is an example of a strided circulant with $n = 4$, $s = 2$:

$$\begin{pmatrix} a & b & c & d \\ c & d & a & b \end{pmatrix}.$$

One can think of this strided circulant as a block-circulant matrix with block sizes $1 \times s$. At the same time, slicing this matrix by taking columns with a step equal to the stride, e.g., columns $\left(0, s, 2 \cdot s, \dots \left(\frac{n}{s} - 1\right) \cdot s\right)$, gives a standard circulant matrix. This fact can be stated as

$$A = \sum_{i=0}^{s-1} A_i \left(I_{\frac{n}{s}} \otimes e_i^T\right),$$

where $\otimes$ denotes the Kronecker product, A is a strided circulant, $e_i \in \mathbb{R}^s$ is the i-th standard basis vector in $\mathbb{R}^s$, and A_i is a circulant obtained by slicing columns of A.

The regular 2D convolutional transform matrix with a single input and single output channels has a doubly block-circulant structure (see Section 5.5 in Jain [1989]). However, for strided convolutions, the structure is different. For a fixed pair of input and output channels, the output of a strided convolution is a submatrix of an output of a convolution with the same kernel but without the stride. More specifically, it is a slice with the stride s by both dimensions (in Python, it would be written as `[::s, ::s]`). For the matrix encoding the transformation, it means that only every s-th block row (simulating the slice by the first dimension) and every s-th regular row of a block (simulating the slice by the second dimension) are considered. This means that the doubly block-circulant structure of the initial matrix turns into a doubly block-strided circulant structure of shape $\left(\frac{n}{s}\right)^2 \times n^2$, where n is the size of a kernel.

Each block of this matrix is a strided circulant, and the block structure is that of a strided circulant as well. The matrix consists of $\frac{n}{s} \times n$ blocks, and each of them has the shape $\frac{n}{s} \times n$. If we denote a strided circulant of a row vector a as $\mathrm{circ}_s(a)$, then the matrix of the transform is as follows:

$$B = \begin{pmatrix} \mathrm{circ}_s(\mathcal{K}_{0,:}) & \dots & \mathrm{circ}_s(\mathcal{K}_{n-1,:}) \\ \mathrm{circ}_s(\mathcal{K}_{n-s,:}) & \dots & \mathrm{circ}_s(\mathcal{K}_{n-s-1,:}) \\ \vdots & \ddots & \vdots \\ \mathrm{circ}_s(\mathcal{K}_{s,:}) & \dots & \mathrm{circ}_s(\mathcal{K}_{s-1,:}) \end{pmatrix}.$$

It can be noted that, in the same way as in strided circulants, we can take block columns of the matrix with a step equal to the stride and get block circulants (where each block is a strided circulant). This block structure can be described as follows:

$$B = \sum_{i=0}^{s-1} B_i \left(\left(I_{\frac{n}{s}} \otimes e_i^T \right) \otimes I_n \right).$$

Here B is a doubly block-strided circulant matrix, and $B_i \in \mathbb{R}^{(\frac{n}{s})^2 \times \frac{n^2}{s}}$ is a block-circulant matrix. This formula is similar to our previous sum representation for a strided circulant; however, the dimensions of the right term of each element are larger to account for the block structure of the left term. Note that this term is needed to describe how the columns of B_i are positioned in the matrix B, similar to the 1D case.

Let us consider B_i. It consists of block columns $\left(i, s+i, 2s+i, \dots \left(\frac{n}{s}-1\right) s + i\right)$:

$$B_i = \begin{pmatrix} \mathrm{circ}_s(\mathcal{K}_{i,:}) & \dots & \mathrm{circ}_s(\mathcal{K}_{n-s+i,:}) \\ \mathrm{circ}_s(\mathcal{K}_{n-s+i,:}) & \dots & \mathrm{circ}_s(\mathcal{K}_{n-2s+i,:}) \\ \vdots & \ddots & \vdots \\ \mathrm{circ}_s(\mathcal{K}_{s+i,:}) & \dots & \mathrm{circ}_s(\mathcal{K}_{i,:}) \end{pmatrix}$$

As a block-circulant with $\frac{n}{s} \times \frac{n}{s}$ blocks of $\frac{n}{s} \times n$, it can be expanded as follows:

$$B_i = \sum_{k=0}^{\frac{n}{s}-1} P^k \otimes C_{ik},$$

where $C_{ik} \in \mathbb{R}^{\frac{n}{s} \times n}$ is a strided circulant block, $C_{ik} = \mathrm{circ}_s\left(\mathcal{K}_{i-sk,:}\right)$ and P is a permutation matrix:

$$P = \begin{pmatrix} 0 & \dots & 0 & 1 \\ 1 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 1 & 0 \end{pmatrix} \in \mathbb{R}^{\frac{n}{s} \times \frac{n}{s}}.$$

We can expand C_{ik} as a strided circulant:

$$C_{ik} = \sum_{j=0}^{s-1} A_{ikj} \left(I_{\frac{n}{s}} \otimes e_j^T \right),$$

where $A_{ikj} \in \mathbb{R}^{\frac{n}{s} \times \frac{n}{s}}$ is a regular circulant matrix that can be acquired as $\mathrm{circ}_1(\mathcal{K}_{i-sk,j::s})$, i.e. as a circulant built from the slice of a string $\mathcal{K}_{i-sk}$, taken with a step s starting from the index j. Finally,

$$
\begin{aligned}
B_i &= \sum_{k=0}^{\frac{n}{s}-1} P^k \otimes C_{ik} = \sum_{k=0}^{\frac{n}{s}-1} P^k \otimes \left(\sum_{j=0}^{s-1} A_{ikj} \left(I_{\frac{n}{s}} \otimes e_j^T \right) \right) = \sum_{k=0}^{\frac{n}{s}-1} \sum_{j=0}^{s-1} P^k \otimes \left(A_{ikj} \left(I_{\frac{n}{s}} \otimes e_j^T \right) \right) \\
&= \sum_{k=0}^{\frac{n}{s}-1} \sum_{j=0}^{s-1} P^k \otimes \left(A_{ikj} \otimes e_j^T \right) = \sum_{k=0}^{\frac{n}{s}-1} \sum_{j=0}^{s-1} \left(\left(P^k \otimes A_{ikj} \right) \otimes e_j^T \right) = \sum_{j=0}^{s-1} \left(\sum_{k=0}^{\frac{n}{s}-1} P^k \otimes A_{ikj} \right) \otimes e_j^T \\
&= \sum_{j=0}^{s-1} \left(\sum_{k=0}^{\frac{n}{s}-1} P^k \otimes A_{ikj} \right) \left(I_{(\frac{n}{s})^2} \otimes e_j^T \right).
\end{aligned}
$$

This reformulation helps us see that the slices of B_i by columns with step s are, in fact, doubly block-circulant matrices defined by spatial slices of the kernel. There are s matrices B_i, each containing s column slices. The jth column slice of B_i is defined by $A_{i,:,j}$, which, in turn, is defined by the kernel slice $\mathcal{K}_{i::s,j::s}$.

In order to reduce the task of computing singular values of the matrix B, corresponding to the strided convolution, to the simpler task of computing singular values of a matrix corresponding to a regular convolution, let us permute the columns of B_i:

$$
B_i' = \sum_{j=0}^{s-1} \left[e_j^T \otimes \left(\sum_{k=0}^{\frac{n}{s}-1} P^k \otimes A_{ikj} \right) \right].
$$

This matrix consists of s consecutive doubly block-circulant matrices $\mathrm{circ}^2\left(\mathcal{K}_{i::s,j::s}\right)$. The first dimension of B_i is the same as the first dimension of B. Therefore, for B, this is also just a permutation of the columns. The next step is to permute the block columns of B:

$$
B' = \sum_{i=0}^{s-1} \left(e_i^T \otimes \sum_{j=0}^{s-1} \left[e_j^T \otimes \left(\sum_{k=0}^{\frac{n}{s}-1} P^k \otimes A_{ikj} \right) \right] \right).
$$

After this permutation, matrix B consists of s^2 consecutive doubly block-circulant matrices. Note that the particular order of these blocks is not important, as it is just a matter of column order.

Finally, let us look at the matrix associated with the multiple-channel convolution. The matrix of the convolutional transform with c_{in} input channels and c_{out} output channels is as follows (equivalent to the structure described in Sedghi et al. [2019]):

$$
M = \begin{pmatrix} B_{0,0} & B_{0,1} & \dots & B_{0,(c_{\mathrm{out}}-1)} \\ B_{1,0} & B_{1,1} & \dots & B_{1,(c_{\mathrm{out}}-1)} \\ \vdots & \vdots & \ddots & \vdots \\ B_{(c_{\mathrm{in}}-1),0} & B_{(c_{\mathrm{in}}-1),1} & \dots & B_{(c_{\mathrm{in}}-1),(c_{\mathrm{in}}-1)} \end{pmatrix},
$$

where

$$
B_{c,d} = \mathrm{circ}_s(\mathcal{K}_{:,:,c,d}).
$$

However, now we know that we can permute the columns of this matrix to get a matrix comprised of doubly block-circulant blocks. After the permutation described above, we get a matrix

$$
M' = \begin{pmatrix} B'_{0,0} & B'_{0,1} & \dots & B'_{0,(s^2 c_{\mathrm{out}}-1)} \\ B'_{1,0} & B'_{1,1} & \dots & B'_{1,(s^2 c_{\mathrm{out}}-1)} \\ \vdots & \vdots & \ddots & \vdots \\ B'_{(c_{\mathrm{in}}-1),0} & B'_{(c_{\mathrm{in}}-1),1} & \dots & B'_{(c_{\mathrm{in}}-1),(s^2 c_{\mathrm{out}}-1)} \end{pmatrix},
$$

where

$$
B'_{c,d} = \mathrm{circ}^2\left(\mathcal{K}_{i::s,j::s,c,\lfloor d/s^2 \rfloor}\right),
$$

Table 3: Various metrics for the proposed framework applied to the SOC method (SOC-TT) and the LipConvNet-N architectures. λ_{ort} denotes the regularization parameter of the orthogonal loss (11). We chose a range of lambda values to allow us to keep the Lipschitz constant under 1. The rank is set to 256.

PARAMETERS		METRICS				
-N	λ_{ort}	ACC. ↑	CIFAR-C ↑	ECE ↓	AA ↑	LIP. ↓
5	5E3	**78.37**	**69.84**	**5.42**	**34.65**	**0.93**
	8E3	77.68	69.57	6.57	33.73	0.79
	1E4	77.17	68.99	6.61	32.13	0.76
	3E4	76.25	67.9	8.35	30.96	0.6
	4E4	75.8	67.67	7.96	30.69	0.59
	5E4	75.74	67.66	8.71	30.77	0.58
20	7E4	**78.41**	**70.47**	**4.89**	**35.89**	**0.91**
	8E4	78.34	70.27	5.18	35	0.89
	1E5	77.45	69.81	5.43	34.1	0.8
	2E5	76.56	68.4	6.62	32.19	0.64
	3E5	76.03	68	7.23	32.52	0.58
	4E5	75.66	67.56	7.08	31.54	0.57
30	2E5	**77.82**	**69.94**	**4.78**	**34.91**	**0.86**
	3E5	77.86	69.38	5.86	33.96	0.75
	5E5	76.57	68.23	6.79	32.73	0.65
	7E5	75.93	68.03	6.93	32.65	0.59
	9E5	75.88	67.71	6.81	31.49	0.59
	1.2E6	76.22	67.6	7.23	31.85	0.56

i, j – some integer indices from 0 to $s-1$. The exact relationship between i, j and c, d is not important, as it depends only on the order of the columns. The only important thing is that it has to be the same in all the rows. We choose the following functional form:

$$i = \left\lfloor \left(d \bmod s^2\right)/s \right\rfloor, \quad j = d \bmod s.$$

This matrix M' can be perceived as the matrix of convolution for a new kernel $K' \in \mathbb{R}^{\frac{n}{s} \times \frac{n}{s} \times c_{\text{in}} \times c_{\text{out}} s^2}$, defined by this equation:

$$K_{a,b,c,d} = \widehat{\mathcal{K}}_{\lfloor (d \bmod s^2)/s \rfloor + as, d \bmod s + bs, c, d \bmod s}.$$

Alternatively, to make things simpler, we can use the additional tensor R, defined in (8). It is easier to use this tensor for implementing the formula in Python.

To conclude, we reduced the task of computing the singular values of M to the task of computing the singular values of the M', solved by Sedghi et al. [2019]. The reduction is made possible via columns permutation, meaning that the singular values of the matrix did not change.

C Additional Empirical Studies

C.1 Plotting Empirical Lipschitz Constant

In order to estimate the Lipschitz constants of LipConvNet networks and their TT-compressed counterparts, we plot histograms of empirical Lipschitz constants inspired by Sanyal et al. [2020]. The Lipschitz constants are evaluated by attacking each image from the test set using the FGSM attack of a fixed radius (0.5 in our experiments) [Goodfellow et al., 2015]. We compute a ratio that is upper-bounded by the true Lipschitz constant L:

$$L_{\text{estimated}}(\mathcal{X}) \equiv \frac{\|f(\mathcal{X}) - f(\mathcal{X}_{\text{attacked}})\|_2}{\|\mathcal{X} - \mathcal{X}_{\text{attacked}}\|_2} \leqslant L \tag{14}$$

for each image and plot the results as a histogram. Here, $f(\mathcal{X})$ is the output of a model f for an input image $\mathcal{X}$; $\mathcal{X}_{\text{attacked}}$ is $\mathcal{X}$ perturbed, as described in the first paragraph. Then, by computing

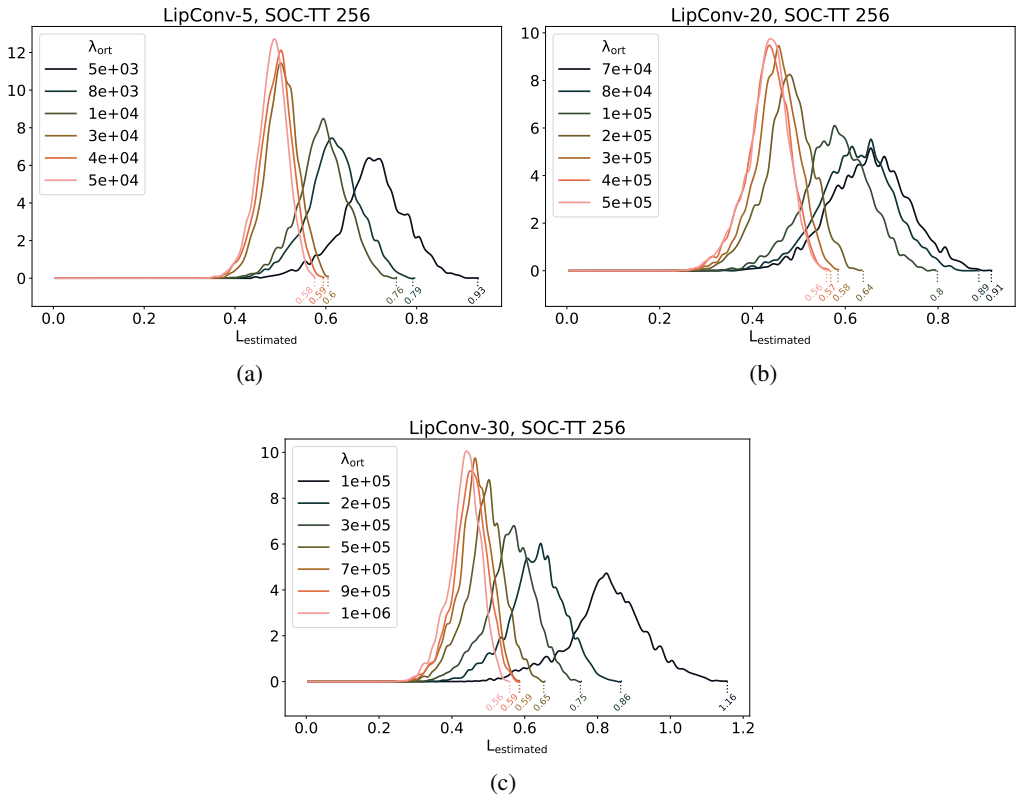

Figure 5: Histograms of empirical constants (14) of networks for different regularization parameters λ_{ort} and LipConvNet-N architectures.

$L_{\text{estimated}}(\mathcal{X})$ for different $\mathcal{X}$ from the dataset, we plot the histograms (Fig. 5). These histograms give us insights into the true Lipschitz constant of the network.

If all network layers are 1-Lipschitz, then we expect each $L_{\text{estimated}}(\mathcal{X})$ on the histogram to be strictly bounded by 1. However, for the LipConvNet architectures, the bound on the Lipschitz constant might not be exact in practice due to truncating Taylor expansion. In our case, the frame matrices $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(3)}$ are trained with the regularization loss, so they are not precisely orthogonal either. By increasing the coefficient λ_{ort} of this loss component, though, we can force the matrices to become orthogonal by the end of the training process.

To support this claim, we provide the distributions of the empirical Lipschitz constants for several LipConvNet architectures trained with different λ_{ort}. The trained models are presented in Tab. 3. The last metric, "Lip.", or Maximum Empirical Lipschitz Constant, is a lower bound on the real Lipschitz constant of the corresponding model. It is obtained as the maximum value of all calculated empirical Lipschitz constants. As we can see, by increasing λ_{ort}, we can balance between the 1-lipschitzness and the quality of metrics: the higher λ_{ort}, the lower the metrics are and the more constrained the Lipschitz constant is. Numbers selected with bold correspond to models that were selected as best for the corresponding LipConvNet-N architectures: they demonstrate high performance in terms of accuracy and robust metrics while at the same time maintaining the Maximum Empirical Lipschitz Constant that does not exceed 1. We also note that LipConvNet architectures do not converge if the convolutional layers are too far from being 1-Lipschitz. For example, this effect was observed on LipConvNet-5 with $\lambda_{\text{ort}} = 4e3$, where the λ_{ort} turns out to be not big enough.

Fig. 5 demonstrates the shifts in distribution depending on λ_{ort}. We can observe ranges of empirical Lipschitz constants on the x-axis. Maximum values for each model, denoted with dotted vertical lines, are the discussed Maximum Empirical Lipschitz Constants in Tab. 3.

Table 4: Performance metrics for different constraints applied to WideResNet-16-10 trained on CIFAR-100. Clipping to 1 increases the baseline performance only without TT decomposition, while the other methods (clipping to 2 and division) yield an increase in some of the metrics. "Speedup" is the speedup of an overhead resulting from singular values control in all the layers in the network. Clipping the whole network w/o decomposition takes 6.2 min, while the application of division takes 0.6 sec. "Comp." (compression) is the ratio between the number of parameters of convolutional layers in the original (~ 16.8M) and decomposed networks.

Method	Rank	Acc. ↑	AA ↑	CC ↑	ECE ↓	Clip (s) ↓	Comp. ↑
Baseline	–	77.97	**27.24**	**48.95**	**8.27**	–	1.0
	192	78.53	25.5	47.28	11.78	–	3.6
	256	**78.55**	26.11	47.72	11.76	–	2.4
	320	76.51	24.81	46.18	12.5	–	1.8
Clip to 1	–	**78.99**	**27.84**	**47.89**	**10.09**	1.0	1.0
	192	77.7	25.02	46.77	11.9	4.1	3.6
	256	77.71	27.15	46.24	11.88	3.3	2.4
	320	77.79	26.34	47.51	11.75	2.3	1.8
Clip to 2	–	**79.92**	**28.15**	**47.73**	**9.4**	1.0	1.0
	192	78.74	27.21	46.08	11.41	4.1	3.6
	256	79.39	27.54	47.26	11.07	3.3	2.4
	320	79.52	26.82	47.68	10.43	2.3	1.8
Division	–	78.59	**26.85**	**48.38**	10.97	1.0	1.0
	192	78.65	25.4	47.86	8.29	4.1	3.6
	256	**78.98**	26.26	47.77	**8.21**	3.3	2.4
	320	77.58	25.69	46.28	12.18	2.3	1.8

C.2 Inference Time of a SOC-TT Layer

In this section, we present the inference time of our framework when applied to the SOC method. In particular, we consider the application of a single SOC layer with various numbers of channels. Fig. 6 illustrates speedups when a single SOC layer is accelerated using the proposed method with different rank values. The figure suggests that for larger residual networks that contain layers with a number of channels up to 1000, the speedup can be up to ≈ 6 times for $c/4$ and up to ≈ 3 times for $c/2$. However, for networks where the number of channels is less than 500, the speedup is less than 2.

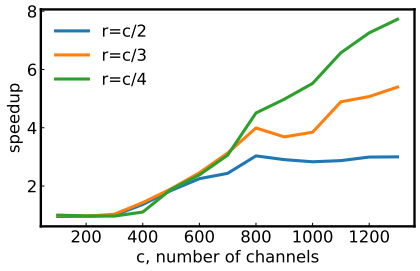

Figure 6: Speedups (w.r.t. uncompressed layer) of the application of a SOC-TT layer, $n = 16$.

C.3 WideResNet16-10 on CIFAR-100

In addition to evaluating our framework on CIFAR-10, we conducted experiments on another classic vision dataset, CIFAR-100. We trained WideResNet-16-10 on CIFAR-100 with the same experimental setup as in Gouk et al. [2021] and compared clipping and division as in the main body of the paper. The results are presented in Tab. 4. Compared with CIFAR-10 experiments from 2, we observe that both clipping and division do not give gain on CIFAR-C (CC). The other metrics tend to improve with the control of singular values. Except for clipping-1, the TT-compressed versions with singular value control outperform the baseline accuracy in most cases. The AA metric of the baseline is outperformed when using clipping to 2 and $r = 256$.