Towards Practical Computation of Singular Values of Convolutional Layers

Anonymous Author(s)
Affiliation
Address
email

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 a significant 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.

1 Introduction

Empirical advances in Deep Learning of the past decade have made machine learning ubiquitous to researchers and practitioners from various domains of science and industry. Yet, the theory of Deep Learning is lagging behind its practical applications, resulting in unexpected outcomes or lack of explainability pertaining to the machine learning 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 to tame such models has given rise to several subfields of Deep Learning research, 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 are the convolutional layers, representing linear transformations that account for the structure of image data. The singular values of these linear transformations are key to the properties of the whole network such the Lipschitz constant and, hence, generalization error and robustness to adversarial examples. Moreover, setting singular values of the convolutional layers close to 1 can also improve training, as it allows for avoiding exploding/vanishing gradients.

Nevertheless, finding and controlling singular values of convolutional layers is a challenging task. Indeed, computing exact singular values [Sedghi et al., 2019] of a layer, given by a kernel tensor of the size $k \times k \times m \times m$ has the complexity $\mathcal{O}(n^2(m^2 + \log n))$, where $n \times n$ is the input image size, m is the number of input and output channels and $k \times k$ is the filter size. Since typically $k^2 \ll m$, computing singular values requires much more operations than computing a single convolution, which has the asymptotic complexity $\mathcal{O}(n^2m^2k^2)$. Moreover, when computing singular values, the storage of an arising padded kernel tensor of the size $n \times n \times m \times m$ can also be an issue for

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

larger networks. For example, setting $n = 64$ and $m = 640$ (last layers of WideResNet-28-10 on ImageNet 256×256) leads to more than 13 GB of storage, which becomes even more pronounced for three-dimensional convolutions. Alternative methods that parametrize the convolutional layers are also known to be 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 is based on the assumption that modern over-parameterized neural networks can be made sparse by using tensor decompositions and with insignificant degradation of the downstream task performance. We investigate the impact of using our approach on the model performance, its calibration, and adversarial robustness.

More specifically, our contributions are as follows:

1. We propose a new framework for reducing the computational complexity of singular values of a convolutional layer using the Tensor-Train (TT) decomposition. 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

The study of singular values and the closely related Lipschitzness of convolutional neural networks impacts many domains of Deep Learning research.

Generalization An essential expectation about neural networks is to generalize input data instead of memorizing it. Much research has been conducted to determine which parts of a neural network lead to its overfitting. For example, the authors of [Hardt et al., 2016] studied the impact of early stopping on the generalization of models. It was proven [Bartlett et al., 2017] that the generalization error of a neural network could be bounded by its Lipschitz constant, defined as min L: $\forall x, y$: $|f(x) - f(y)| \leqslant L \cdot |x - y|$. The Lipschitz constant, in turn, could be bounded by the product of largest singular values of the Jacobian of the layers. Numerous attempts to constrain the operator norms have been made. For instance, [Gouk et al., 2020] proposed to estimate the Lipschitz constant and scale the transformation using this estimate.

Spectral Normalization Another constraint was proposed in [Yoshida and Miyato, 2017]. First, the authors of the paper propose to reshape the kernel of a convolutional layer into a matrix. Then, the largest singular value estimate of the reshaped matrix is used as an upper bound for the operator norm of the layer. However, this estimation is rarely accurate, which has been studied by other researchers. For example, [Singla and Feizi, 2021a] proved that the estimation can serve as a genuine upper bound if multiplied by a certain constant. Moreover, access and control of *all singular values* are crucial in some applications, e.g., when dealing with the mode collapse in GANs [Liu et al., 2019].

Vanishing and Exploding Gradients One of the significant challenges in training neural networks is the problem of gradients magnitude control. In some cases, changes need to be applied to the architecture in order to solve the problem, for example, by using residual connections [He et al., 2015]. The LSTM architecture, introduced by [Hochreiter and Schmidhuber, 1997], was designed to overcome the problem. Another way to solve the problem of exploding and vanishing gradients

is to limit the singular values of the Jacobian matrices of the transformations defined by the layers of a network. Moreover, [Bartlett et al., 2017, Cisse et al., 2017] explore the relation between the spectral norm of the Jacobian matrix and model performance and its robustness. Thus, it is evident that studying the singular values of Jacobian matrices is of great importance.

Lipschitz Architectures To resolve the issue with bounding the Lipschitz constant, [Li et al., 2019] proposed a new Lipschitz-constrained convolutional neural network: LipConv. One of the major challenges when training a Lipschitz-constrained neural network is the problem of vanishing gradients [Anil et al., 2019]. To overcome the issue, the authors parameterize standard convolutional layers with Block Convolution Orthogonal Parameterization. However, this approach has a notable flaw: it is too expensive. Addressing the problem, [Singla and Feizi, 2021b] proposed to use another approach: Skew-Symmetric Orthogonal Convolutional layers. Although it reduced the time needed for training a model, it is still much slower than training with standard convolutions.

Exact Singular Values of Convolutions [Sedghi et al., 2019] proposed an elegant method for computing exact singular values of convolutional layers. The authors of [Singla and Feizi, 2021b] impose nearly orthogonal constraints on convolutional layers using Taylor expansion of a matrix exponential of a skew-Hermitian matrix. The usage of Taylor expansion leads to multiple applications of a convolution operator for a single layer, thus, slowing down computations. In the present work, we resort to using the results developed by [Sedghi et al., 2019] and show how to use them within our compressed convolutions framework with reduced computational complexity. Speaking of compression,

Compression Over-parameterization of neural networks has been the driving force of many applied domains of Deep Learning research, such as Computer Vision [He et al., 2015] and Natural Language Processing [Devlin et al., 2019]. Several recent works paved the way for discovering such models through several techniques countering over-parameterization, such as inducing structured [Anwar et al., 2017] or unstructured sparsity [Lee et al., 2019]. However, most of the approaches agree that it is often possible to approach the uncompressed model performance by imposing some form of sparsity on the network weights. In this paper, we rely on the sparsity assumption to make the computation of singular values of convolutional layers tractable.

Decompositions Tensor and matrix decompositions facilitate network compression with [Obukhov et al., 2020] and without weight sharing [Novikov et al., 2015, Garipov et al., 2016, Wang et al., 2018], training stability under compression [Obukhov et al., 2021], and weight sharing schemes for multi-task learning [Kanakis et al., 2020]. Tucker-2 decomposition, which is equivalent to the TT decomposition we use in this paper, was considered in [Kim et al., 2016].

Robustness Apart from boosting the accuracy of predictions, researchers have been working on improving the robustness of neural networks to adversarial attacks. Adversarial attacks aim to find a negligible to human perception perturbation of input data that would sabotage the predictions of a model. Various attacks have been proposed in recent years. For example, [Croce and Hein, 2020] consider four types of attacks: two modifications of Projected Gradient Descent (PGD) [Madry et al., 2019], FAB-attack [Croce et al., 2019], and Square Attack [Andriushchenko et al., 2019]. In this paper, we use the AutoAttack Robust Accuracy to evaluate the results.

Calibration Confidence calibration is the problem of establishing trust in a model via its predictive uncertainty. Expected Calibration Error (ECE) [Naeini et al., 2015] is a metric of a classification model calibration, widely studied in recent years for various purposes [Guo et al., 2017, Kull et al., 2019, Wen et al., 2020, Luo et al., 2021, Qin et al., 2020, Küppers et al., 2020]. As stated in [Deng-Bao Wang, 2021], a stronger regularization of a model helps achieve better calibration. Connections between model Lipschitz constraints, calibration, and out-of-distribution (OOD) model performance have been drawn in recent literature [Postels et al., 2021]. In this paper, we utilize ECE as another proxy metric for improving model Lipschitzness.

3 Method-independent Complexity Reduction

This section explains how to reduce the complexity of the calculation of the singular values of a convolutional layer 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, respectively, to regular images and 3D data.

The computational complexity of methods from prior art [Sedghi et al., 2019, Singla and Feizi, 2021b] explicitly depends on both the number of channels (m) and the actual input size (n). The dependence on the input size is $O(n^2 \log n)$ in [Sedghi et al., 2019] and $O(n^2)$ for a single term of Tailor expansion in [Singla and Feizi, 2021b]. For the number of channels, [Sedghi et al., 2019] has $O(m^3)$ and [Singla and Feizi, 2021b] has $O(m^2)$, again for a single term. Due to the non-linear dependency on both the number of channels and input size, the task of finding singular values in convolutional layers of modern neural networks with hundreds of channels and large input sizes (let alone 3D data) is intractable with these methods.

Therefore, reducing the computational cost of finding singular values in these cases is an important task. We bootstrap our approach based on the recent research on neural network sparsity by compressing convolutional layers with the TT decomposition. Our approach focuses on dealing with channel-wise complexity of the employed methods. The rest of the sections are organized as 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

Let us formally introduce a convolutional layer. Let $\mathcal{K} \in \mathbb{R}^{k \times \cdots \times k \times m_1 \times m_2}$ be a $(d+2)$-dimensional kernel tensor, where k is a filter size and m_1, m_2 are the numbers of input and output channels respectively. A convolutional layer is a linear map defined by the kernel tensor $\mathcal{K}$:

$$\mathscr{C}_{\mathcal{K}} \colon \mathbb{R}^{m_1 \times n \times \cdots \times n} \to \mathbb{R}^{m_2 \times f(n,k) \times \cdots \times f(n,k)},$$

where $f(n,k)$ is an integer, depending on the type of the convolution used (linear, periodic, correlation) and convolution parameters, e.g., strides. For example, a widely used correlation-type convolution with strides equal to one, reads

$$\mathcal{Y}_{jq_1 \ldots q_d} = \sum_{p_1,\ldots,p_d=0}^{k-1} \sum_{i=0}^{m_1-1} \mathcal{K}_{p_1 \ldots p_d ij} \mathcal{Y}_{i,q_1+p_1,\ldots,q_d+p_d},$$

for all $j = 0, \ldots, m_2$ 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

$$\mathcal{Y} = \mathscr{C}_{\mathcal{K}}(\mathcal{X}), \tag{1}$$

implying one of the convolution types mentioned above. Using the isomorphism between multidimensional arrays and column vectors of appropriate size and the linearity of $\mathscr{C}_{\mathcal{K}}$, we may rewrite (1) as a matrix-vector product:

$$\texttt{vec}(\mathcal{Y}) = T_{\mathcal{K}} \, \texttt{vec}(\mathcal{X}), \tag{2}$$

where `vec` is a row-major reshaping of a multidimensional array into a column vector and $T_{\mathcal{K}} \in \mathbb{R}^{m_2 n^d \times m_1 n^d}$ is a matrix with additional block structure: its $n^d \times n^d$ blocks are d-level Toeplitz matrices (see [Sedghi et al., 2019] for $d = 2$). Representation (2) allows us to replace singular values of a linear map $\mathscr{C}_{\mathcal{K}}$ by singular values of a matrix $T_{\mathcal{K}}$, which is handy for analysis. In turn, the 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,

$$\begin{aligned} \|\mathscr{C}_{\mathcal{K}}(\mathcal{X}) - \mathscr{C}_{\mathcal{K}}(\mathcal{Z})\|_F &= \|T_{\mathcal{K}} \texttt{vec}(\mathcal{X}) - T_{\mathcal{K}} \texttt{vec}(\mathcal{Z})\|_2 \\ &\leq \|T_{\mathcal{K}}\|_2 \, \|\texttt{vec}(\mathcal{X}) - \texttt{vec}(\mathcal{Z})\|_2 = \sigma_1(T_{\mathcal{K}}) \|\mathcal{X} - \mathcal{Z}\|_F, \end{aligned}$$

since $\sigma_1(T_{\mathcal{K}}) = \|T_{\mathcal{K}}\|_2$.

3.2 Compressed Convolutional Layers

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

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

with smaller tensors $\mathcal{K}^{(1)} \in \mathbb{R}^{m_1 \times r_1}$, $\mathcal{K}^{(3)} \in \mathbb{R}^{r_2 \times m_2}$ and $\mathcal{K}^{(2)} \in \mathbb{R}^{k \times \dots \times k \times r_1 \times r_2}$ and integers r_1, r_2: $1 \leq r_1 \leq m_1$, $1 \leq r_2 \leq m_2$, called ranks. This decomposition is essentially the TT decomposition [Oseledets, 2011] of the kernel tensor with filter modes $p_1, \dots, p_d$ merged together into a single long index. We also note that it is equivalent to the so-called Tucker-2 decomposition, which was successfully used in [Kim et al., 2016] to compress large convolutional networks.

By substituting (3) into (1), one can easily verify that convolution with the kernel given by (3) is equivalent to a sequence of three convolutions without nonlinear activation functions between them:

1. $1 \times \dots \times 1$ convolution with m_1 input and r_1 output channels (kernel tensor[1] — $\mathcal{K}^{(1)}$);
2. $k \times \dots \times k$ convolution with r_1 input and r_2 output channels (kernel tensor — $\mathcal{K}^{(2)}$);
3. $1 \times \dots \times 1$ convolution with r_2 input and m_2 output channels (kernel tensor — $\mathcal{K}^{(3)}$);

or equivalently,

$$\mathscr{C}_{\mathcal{K}} = \mathscr{C}_{\mathcal{K}^{(3)}} \circ \mathscr{C}_{\mathcal{K}^{(2)}} \circ \mathscr{C}_{\mathcal{K}^{(1)}}, \tag{4}$$

where $\mathscr{A} \circ \mathscr{B}$ denotes composition of two maps $\mathscr{A}$ and $\mathscr{B}$.

In the rest of this section, we show that singular values of the original convolutional layer with kernel $\mathcal{K}$ correspond to the 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 $\mathcal{K} \in \mathbb{R}^{k \times \dots \times k \times m_1 \times m_2}$ be given by (3). Then there exist matrices $\mathcal{Q}^{(1)} \in \mathbb{R}^{m_1 \times r_1}$, $\mathcal{Q}^{(3)} \in \mathbb{R}^{r_2 \times m_2}$ satisfying*

$$\mathcal{Q}^{(1)\top} \mathcal{Q}^{(1)} = I_{r_1}, \quad \mathcal{Q}^{(3)} \mathcal{Q}^{(3)\top} = I_{r_2}$$

and a tensor $\mathcal{Q}^{(2)} \in \mathbb{R}^{k \times \dots \times k \times r_1 \times r_2}$, such that

$$\mathcal{K}_{p_1 \dots p_d ij} = \sum_{\alpha=0}^{r_1-1} \sum_{\beta=0}^{r_2-1} \mathcal{Q}^{(1)}_{i\alpha} \, \mathcal{Q}^{(2)}_{p_1 \dots p_d \alpha\beta} \, \mathcal{Q}^{(3)}_{\beta j}. \tag{5}$$

Proof. 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 (3) yields (5) 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. □

Now we have all preliminaries in place to formulate one of the main results of this paper.

Theorem 1. *Let $\mathcal{K} \in \mathbb{R}^{k \times \dots \times k \times m_1 \times m_2}$ be given by (3). Let also*

$$\mathcal{K}^{(1)\top} \mathcal{K}^{(1)} = I_{r_1}, \quad \mathcal{K}^{(3)} \mathcal{K}^{(3)\top} = I_{r_2}. \tag{6}$$

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

[1] When using $\mathcal{K}^{(1)} \in \mathbb{R}^{m_1 \times r_1}$ as a convolution kernel, we actually imply that it is a tensor of the size $\mathcal{K}^{(1)} \in \mathbb{R}^{1 \times \dots \times 1 \times m_1 \times r_1}$. Similarly, for $\mathcal{K}^{(3)}$.

Proof. For proof see Appendix A. □

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 $\mathcal{K}^{(2)}$. At the same time, thanks to Lemma 1, one may always set $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(3)}$ to be orthogonal without losing expressivity of $\mathscr{C}_{\mathcal{K}}$.

4 Application of the Proposed Framework

In this section, we discuss how to apply the proposed framework to different methods. We start with the explicit formula for singular values computation and then discuss how to apply the framework when the convolutions are non-trivially parametrized.

4.1 Explicit Formulas for Strided Convolutions

[Sedghi et al., 2019] considers periodic convolutions, which allow them to express singular values in terms of 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 $\widehat{\mathcal{K}} \in \mathbb{R}^{n \times n \times m_1 \times m_2}$ be a kernel $\mathcal{K} \in \mathbb{R}^{k \times k \times m_1 \times m_2}$ of a periodic convolution, padded with zeros along the filter modes. Let s be the stride of the convolution $\mathscr{C}_{\mathcal{K}}$: $n \equiv 0 \pmod{s}$ and let the reshaped kernel $R \in \mathbb{R}^{s^2 \times \frac{n}{s} \times \frac{n}{s} \times m_1 \times m_2}$ be such that:*

$$R_{q,a,b,i,j} = \widehat{\mathcal{K}}_{\lfloor q/s \rfloor + as, (q \bmod s) + bs, i, j}. \tag{7}$$

Let us consider $(\frac{n}{s})^2$ matrices $P^{(p_1 p_2)} \in \mathbb{R}^{m_1 \times s^2 m_2}$ with the entries:

$$P_{ij}^{(p_1 p_2)} = (F^\top R_{j \bmod s^2, :, :, i, \lfloor j/s^2 \rfloor} F)_{p_1 p_2}, \tag{8}$$

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

$$\sigma\left(T_{\mathcal{K}}\right) = \bigcup_{p_1, p_2 \in \{1, \ldots, \frac{n}{s}\}} \sigma\left(P^{(p_1 p_2)}\right). \tag{9}$$

Proof. For proof see Appendix B. □

The algorithm for finding the singular values of a TT-compressed convolutional layer based on Theorem 2 is summarized in Alg. 1.

Computing singular values using (8) and (9) has the complexity $\mathcal{O}(n^2 m^2 (ms^2 + \log \frac{n}{s}))$, $m = \max\{m_1, m_2\}$, which depends cubically on m. Thus, reducing m can significantly reduce the computational cost of finding the singular values. Let us now use the TT-representation (3) 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, $r = m/2$ provides a theoretical speedup of up to 8 times (see Figure 1 for practical evaluation).

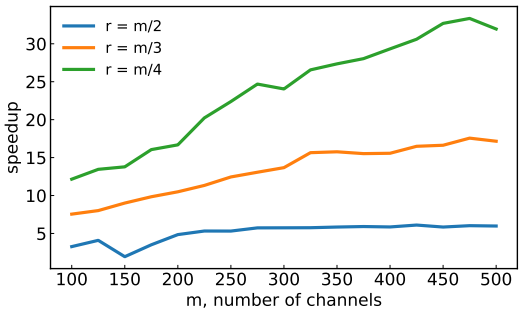

Figure 1: The speedup (w.r.t. uncompressed layer) to compute singular values with $n = 16$ and $s = 1$.

Note that to apply Theorem 1, we also need to orthogonalize $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(3)}$, which can be done in various ways as is described in the following section.

Algorithm 1 Singular values of a TT-compressed periodic convolutional layer

Input: $K^{(1)} \in \mathbb{R}^{m_1 \times r_1}$, $K^{(2)} \in \mathbb{R}^{k \times k \times r_1 \times r_2}$ and $K^{(3)} \in \mathbb{R}^{r_2 \times m_1}$ – cores of the TT-compressed kernel $\mathcal{K}$; n – input image size; s – stride of the convolution defined by kernel $\mathcal{K}$; $n \equiv 0 \pmod{s}$.
Output: non-zero singular values of the convolutional layer, defined by the kernel tensor $\mathcal{K}$.

$Q^{(1)}, R^{(1)} \coloneqq \texttt{QR}(\mathcal{K}^{(1)})$ *# QR-decompositions*
$Q^{(3)}, R^{(3)} \coloneqq \texttt{QR}({\mathcal{K}^{(3)}}^\top)$

$\widehat{\mathcal{K}}^{(2)}_{p_1 p_2 \alpha \beta} \coloneqq R^{(1)}_{\alpha,:} \mathcal{K}^{(2)}_{p_1,p_2,:,:} R^{(3)}_{:,\beta} \; \forall \alpha, \beta$ *# compute $\widehat{\mathcal{K}}^{(2)}$*

Zero-pad $\widehat{\mathcal{K}}^{(2)}$ to the size $n \times n \times r_1 \times r_2$ *# pad $\widehat{\mathcal{K}}^{(2)}$*

$R \coloneqq$ reshape $\widehat{\mathcal{K}}^{(2)}$ into R as in (7) *# reshape $\widehat{\mathcal{K}}^{(2)}$*

$P^{(:,:)}_{ij} \coloneqq \texttt{fft2}(R_{j \bmod s^2,:,:,i,\lfloor j/s^2 \rfloor}) \; \forall \, i, j$ *# 2-dimensional FFT for computing P*

Return: $\bigcup_{p_1,p_2 \in \{1,\ldots,\frac{n}{s}\}} \sigma\left(P^{(p_1 p_2)}\right)$

4.2 Orthogonality constraints

In order 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 orthogonality of the frame matrices; for example, we can impose regularization on the layers. This is also extremely helpful in case if $\mathcal{K}^{(2)}$ is not a standard convolution, but has a special complex structure as, for example, the SOC layer from [Singla and Feizi, 2021b].

Let us define the orthogonal loss we use to regularize the frame matrices in the TT decomposition of our convolutional layers instead of QR. Let n_l denote the number of TT-decomposed layers in a network, $K_1^{(i)}$ and $K_3^{(i)}$ — frame matrices, and $r_1^{(i)}$ with $r_3^{(i)}$ — ranks of the frame matrices. Then, the orthogonal loss is defined as follows:

$$\text{loss}_{ort} = \Big(\sum_{i=0}^{n_l} \left\| K_1^{(i)T} K_1^{(i)} - I_{r_1^{(i)}} \right\|_F^2 + \left\| K_3^{(i)} K_3^{(i)T} - I_{r_3^{(i)}} \right\|_F^2 \Big) / \Big(\sum_{i=0}^{n_l} \left(r_1^{(i)2} + r_3^{(i)2} \right) \Big). \quad (10)$$

In case of applying this regularization, we sum this loss with crossentropy:

$$\text{loss} = \text{loss}_{CE} + \lambda_{ort} \text{loss}_{ort},$$

loss_{ort} itself is the mean square error for each element in all $K_1^{(i)}$ and $K_3^{(i)}$, so the λ_{ort} coefficient tends to be large. We set it to 1e5 for experiments on WideResNets.

5 Experiments

To study the effects the proposed regularization has 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 CIFAR-10 [Krizhevsky et al., 2009] dataset. The code was implemented in PyTorch [Paszke et al., 2019] and executed on a single NVIDIA Tesla V100, 32GB. Reproducible source code will be made available at *redacted*.

Models In our experiments, we used the LipConv architecture [Singla and Feizi, 2021b] and WideResNet-28-10 [Zagoruyko and Komodakis, 2016].

Evaluation Metrics We consider four main metrics: standard accuracy on the test split, accuracy on the CIFAR-C dataset [Hendrycks and Dietterich, 2019][2], accuracy after applying the AutoAttack module [Croce and Hein, 2020], and the Expected Calibration Error (ECE) [Guo et al., 2017]. The

[2] Distributed under Apache License 2.0

Table 1: LipConv-n for different hyperparameters; comparison between SOC and SOC-TT, which is SOC method combined with the proposed framework. Best configurations are highlighted.

Parameters			Metrics					
Layer	Rank	-n	Ac. ↑	CIFAR-C ↑	ECE ↓	AA ↑	P.R. ↑	Comp. ↑
SOC		5	75.9	67.32	9.03	31.35	59.08	1
SOC-TT	256	5	**78.37**	**69.84**	**5.42**	**34.65**	**65.24**	1.2
	128	5	76.82	68.49	6.62	32.65	62.72	2.9
SOC		20	75.81	67.94	7.34	32.87	61.3	1
SOC-TT	256	20	**78.41**	**70.47**	**4.89**	**35.89**	**65.52**	1.4
	128	20	75.57	67.76	6.54	31.9	60.66	3.3
SOC		30	75.1	67.31	6.42	31.18	60.51	1
SOC-TT	256	30	**77.82**	**69.94**	**4.78**	**34.91**	**64.9**	1.5
	128	30	75.19	66.46	6.05	32.65	60.05	3.5

Table 2: Performance metrics for different constraints of WideResNet-28-10. Computations with clipping and no compression are intractable.The column "Clip" contains the time needed for processing of all the layers in the network. For further details see Section 5.2.

Method	Rank	Clip	Metrics					
			Acc. ↑	CIFAR-C ↑	AA ↑	ECE ↓	Clip (s) ↓	Comp. ↑
Baseline	–	–	**95.43**	**75.26**	**58.7**	**3.12**	–	1.0
	256	–	91.35	67.39	50.08	4.88	–	2.4
Clipping	–	1	–	–	–	–	2760	1.0
	256	0.5	94.23	72.94	54.2	3.83	342.5	2.4
	256	1	**95.65**	**74.7**	**62.47**	**3.25**	342.5	2.4
Division	–	–	**95.89**	**76.08**	60.6	**2.68**	0.084	1.0
	256	–	95.68	75.52	**64.12**	2.84	0.069	2.4

last column of both tables with the results of our experiments contains the compression ratio of all layers in the network. In addition, we also have a dedicated metric for the LipConv architecture: p.r. — provable bound-norm robustness, introduced in [Li et al., 2019]. This metric shows the percentage of input images that are guaranteed to be predicted correctly on any perturbation with radius 36/255.

5.1 Experiments on LipConv

The authors of [Singla and Feizi, 2021b] introduced a new architecture that they call Lipschitz Convolutional Networks, or LipConv for short. LipConv-n consists of n convolutional layers and activations. This architecture is provably 1-Lipschitz by constrast to popular residual network architectures. In [Singla and Feizi, 2021b], the authors also developed an orthogonal layer called Skew Orthogonal Convolution (SOC) and applied it in LipConv networks. We modify the SOC layer in correspondence with the proposed framework by adding a $1 \times 1 \times c_{in} \times R$ convolution before applying SOC, reducing the number of channels in SOC to $R \times R$, and finally applying the last convolution of size $1 \times 1 \times R \times c_{out}$. To maintain the orthogonality of this new layer that we call SOC-TT, we add orthogonal loss (Section 4.2) to keep these two frame matrices orthogonal.

Table 1 demonstrates metrics improvement after replacing SOC with SOC-TT. For rank 256, we performed grid search of the coefficient λ_{ort} in front of the orthogonal component of the loss. The training protocol is the same as in [Singla and Feizi, 2021b] except for the learning rate of LipConv-20 with the SOC-TT layer, which we set to 0.05 instead of 0.1.

5.2 Experiments with WideResNet-28-10

To experiment with a more common and widely used network, we chose WideResNet-28-10 [Zagoruyko and Komodakis, 2016] with 36M parameters. It produces the best results on CIFAR-10

of all networks in the WideResNet family and provides enough room for compression. The results of our experiments with WideResNet-28-10 are summarised in Table 2.

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 0.5 or 1 in numerical experiments. Then all singular values greater than δ are replaced with δ. This procedure allows for maintaining maximal singular values on the desired level. After the clipping, we reconstruct the expanded kernel $\widehat{\mathcal{K}}_\delta$ of the size $n \times n \times m_1 \times m_2$, which is no longer sparse. Finally, we return its original shape by setting $\mathcal{K} = \widehat{\mathcal{K}}_\delta(1{:}k, 1{:}k, :, :)$. This operation is performed every 100 training iterations.

Due to the abundance of channels, clipping the singular values of uncompressed convolutional layers of WideResNet-28-10 using the method from [Sedghi et al., 2019] is intractable. In our experiments, one clipping operation of the entire network takes well over 45 minutes, while clipping of the compressed network is 8 times faster. The robustness of the network increases when trained 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.

The authors of [Gouk et al., 2020] derived a formula for the Lipschitz constants of batch normalization layers and proposed to regularize them too. They set the desired values for Lipschitz constants of convolutional, fully connected, and batch normalization layers after each training step. 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. Similar operation is performed for batch normalization layers, but the Lipschitz constant is computed directly instead of the singular value estimate.

This approach proved to be even more successful for WideResNet-28-10. 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, 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.

Hyperparameters The training schedule is adapted from the original paper. We train our models with batches of 128 elements for 200 epochs, using SGD optimizer and learning rate of 0.1, multiplied by 0.1 after 60, 120, and 160 epochs. The hyperparameters for the experiments with division were taken from [Gouk et al., 2020]. The orthogonal loss (10) was used only in the last experiment with division, while the experiments with clipping used the QR decomposition.

6 Conclusion

We proposed a principled sparsity-driven approach to constraining 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 imrpoving robustness of neural networks, it has an overall positive impact on their reliability. Nevertheless due to the generality of our approach, it can be used in different applications domains, including malevolent purposes.

References

Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. *CoRR*, abs/1912.00049, 2019.

Cem Anil, James Lucas, and Roger Grosse. Sorting out Lipschitz function approximation. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 291–301. PMLR, 09–15 Jun 2019.

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 Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks, 2017.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples, 2017.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. *CoRR*, abs/2003.01690, 2020.

Francesco Croce, Jonas Rauber, and Matthias Hein. Scaling up the randomized gradient-free adversarial attack reveals overestimation of robustness using established attacks. *CoRR*, abs/1903.11359, 2019.

Min-Ling Zhang Deng-Bao Wang, Lei Feng. Rethinking calibration of deep neural networks: Do not be afraid of overconfidence. *NeurIPS*, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423.

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

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. Regularisation of neural networks by enforcing lipschitz continuity, 2020.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. *ArXiv*, abs/1706.04599, 2017.

Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

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 Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XX*, volume 12365 of *Lecture Notes in Computer Science*, 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 Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 05 2016.

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

Meelis Kull, Miquel Perello Nieto, Markus Kängsepp, Telmo Silva Filho, Hao Song, and Peter Flach. Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Fabian Küppers, Jan Kronenberger, Amirhossein Shantia, and Anselm Haselhoff. Multivariate confidence calibration for object detection. *CoRR*, abs/2004.13546, 2020.

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. *CoRR*, abs/1911.00937, 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.

Rachel Luo, Aadyot Bhatnagar, Huan Wang, Caiming Xiong, Silvio Savarese, Yu Bai, Shengjia Zhao, and Stefano Ermon. Localized calibration: Metrics and recalibration. *CoRR*, abs/2102.10809, 2021.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 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 C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *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 Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7392–7404. PMLR, 13–18 Jul 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 Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3547–3555. PMLR, 13–15 Apr 2021.

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, Luc Van Gool, Fisher Yu, and Federico Tombari. On the practicality of deterministic epistemic uncertainty. *CoRR*, abs/2107.00649, 2021.

Yao Qin, Xuezhi Wang, Alex Beutel, and Ed H. Chi. Improving uncertainty estimates through the relationship with adversarial robustness. *CoRR*, abs/2006.16375, 2020.

Hanie Sedghi, Vineet Gupta, and Philip M. Long. The singular values of convolutional layers, 2019.

Sahil Singla and Soheil Feizi. Fantastic four: Differentiable bounds on singular values of convolution layers, 2021a.

Sahil Singla and Soheil Feizi. Skew orthogonal convolutions, 2021b.

Wenqi Wang, Yifan Sun, Brian Eriksson, Wenlin Wang, and Vaneet Aggarwal. Wide compression: Tensor ring nets. *CoRR*, abs/1802.09052, 2018.

Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: An alternative approach to efficient ensemble and lifelong learning. *CoRR*, abs/2002.06715, 2020.

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

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *ArXiv*, abs/1605.07146, 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] See section 5 with the experiments.
   (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] See sections 3 and 4.
   (b) Did you include complete proofs of all theoretical results? [Yes] See supplementary material.
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)? [No]
   (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] See section 5.
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] See section 5.
   (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]