# Contributions on Deep Neural Networks with Toeplitz Matrices: Efficiency and Robustness

*by*

Alexandre Araujo

Université Paris-Dauphine

# Table of Contents

# List of Figures

# List of Tables

# List of Symbols

| | |
|---|---|
| $\mathbb{N}$ | The set of natural numbers |
| $\mathbb{R}$ | The set of real numbers |
| $\mathbb{C}$ | The set of complex numbers |
| $[n]$ | The set of natural numbers from 1 to n |
| $\mathbf{i} = \sqrt{-1}$ | Imaginary number |
| $\Re(z)$ | Real part of complex number z |
| $\Im(z)$ | Imaginary part of complex number z |
| $\|z\|$ | Modulus of complex number z |
| $\mathbf{M} = (a_{ij})$ | Matrix $\mathbf{M}$ with $(i, j)$-entry $a_{ij}$ |
| $\mathbf{x} = [\mathbf{x}_1 \ldots \mathbf{x}_n]$ | Vector $\mathbf{x}$ of size $n$ with $\mathbf{x}_i$ entries |
| $\mathbf{M}^\top$ | Transpose of matrix $\mathbf{M}$ |
| $\mathbf{M}^*$ | Conjugate transpose of matrix $\mathbf{M}$ |
| $\mathbf{1}_n$ | $n$-vector of ones |
| $\mathbf{I}_{n\times n}$ | Identity matrix of size $n \times n$ |
| $\mathbf{Q}_{n\times n}$ | Anti-identity matrix $n \times n$ |
| $\\|\mathbf{M}\\|_p$ | Norm $p$ of matrix $\mathbf{M}$, *i.e.* $\\|\mathbf{M}\\|_p = \sup_{\\|\mathbf{x}\\|_p \neq 0} \frac{\\|\mathbf{Mx}\\|_p}{\\|\mathbf{x}\\|_p}$ |
| $\\|\mathbf{x}\\|_p$ | Norm $p$ of vector $\mathbf{x}$, *i.e.* $\\|\mathbf{x}\\|_p = (\sum_{i=0}^{n} \mathbf{x}_i^p)^{\frac{1}{p}}$ |
| $\sigma_1(\mathbf{M})$ | Largest singular value of matrix $\mathbf{M}$, *i.e.* $\sigma_1(\mathbf{M}) = \\|\mathbf{M}\\|_2$ |
| $\lambda_1(\mathbf{M})$ | Largest eigenvalue of matrix $\mathbf{M}$ |
| $\mathbf{M} \geq 0$ | $\mathbf{M}$ is positive semidefinite, *i.e.* $\mathbf{x}^*\mathbf{Mx} \geq 0$ for all $\mathbf{x} \in \mathbb{C}^n$ |
| $\mathbf{M} > 0$ | $\mathbf{M}$ is positive definite, *i.e.* $\mathbf{x}^*\mathbf{Mx} > 0$ for all $\mathbf{x} \in \mathbb{C}^n$ |

# 1 Introduction

## Contents

## 1.1 Context

Neural networks are now present all around us and in all areas. Without knowing it, an average person uses them every day; automatic translators, chatbots, voice assistants, recommendations, etc. The applications are very diverse and can have a profound impact on entire sectors (transport, health care, etc.). If the use of neural networks is so important, it is because they are so effective in *learning* any type of task with results that are close to perfection, surpassing man on many points.

However, although widely in use, Neural Networks are not perfect, a lot of open questions remain and many problems have emerged. This thesis attempts to highlight and answer some of the problems that Neural Networks face.

## 1.2 Introduction to Neural Networks

Neural Networks find their roots, in 1958, in the works of Frank Rosenblatt (Rosenblatt, 1958) where for the first time, the Perceptron, an electronic device inspired by the human brain, showed ability to *learn* from multiple examples. In essence, the Perceptron is an algorithm for learning a binary classifier, this classifier can be analytically described as a composition of a linear function $\phi$ and the Heaviside step function $\rho$ as follows:

$$f(\mathbf{x}) = \rho \circ \phi(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \phi(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{1.1}$$

where $\phi(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ is a linear function and the values of the vector $\mathbf{w}$ are the learned parameters. However, a few years after its introduction, Minsky & Papert (1969) demonstrated important limitations of the Perceptron. Indeed, although theoretically capable of classifying any linear separable problem, it is not able to correctly classify simple non-linear functions (see Figure 1.1). To circumvent these limitations, Minsky

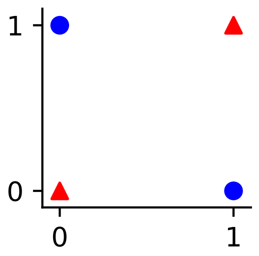


Figure 1.1: Graphical representation of the XOR function. This function cannot be separated by a linear classifier.

& Papert (1969) considered that an *other layer of logic* could be added to the function allowing the classification to be done on another representation (see Figure 1.2). Let $\phi_1$ and $\phi_2$ be two linear functions, we could define a *Multi-Layer Perceptron* as follows:

$$f(\mathbf{x}) = \rho \circ \phi_2 \circ \rho \circ \phi_1(\mathbf{x}) \tag{1.2}$$

$$\begin{array}{c|c} \multicolumn{2}{c}{\mathbf{x}} \\ \hline 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{array} \quad \rightarrow \quad \begin{array}{c|c} \multicolumn{2}{c}{\rho \circ \phi_1(\mathbf{x})} \\ \hline 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{array} \quad \rightarrow \quad \begin{array}{c} f(\mathbf{x}) \\ \hline 0 \\ 1 \\ 1 \\ 0 \end{array}$$

Input representation — Intermediate representation — Final representation

Figure 1.2: Classification of the XOR function with a Multi-Layer Perceptron

One of the first *Multi-Layer Perceptron*, or now more commonly known as *Deep Neural Network*, was introduced by Ivakhnenko & Lapa in 1967. More precisely, a

Neural Neural can be analytically described as a composition of linear function interlaced with non-linear function (also called activation function):

$$f_{\mathbf{W}}(\mathbf{x}) = \phi_{\mathbf{W}_L} \circ \rho \circ \phi_{\mathbf{W}_{L-1}} \circ \cdots \circ \phi_{\mathbf{W}_2} \circ \rho \circ \phi_{\mathbf{W}_1}(\mathbf{x}) \tag{1.3}$$

where the function $\phi_{\mathbf{W}_i}$ is a linear function parameterized by $\mathbf{W}_i$, $\rho$ is a non-linear function (also called activation function), $\mathbf{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$ and $L$ correspond to the *depth* of the network (*i.e.* the number of layers).

In recent years, Deep Neural Networks have achieved state-of-the-art performances in a variety of domains such as image recognition (LeCun et al. 1998; Krizhevsky et al. 2012; He et al. 2016a; Tan & Le, 2019), image detection (Redmon et al. 2016), natural language processing (Radford et al. 2018), speech recognition Hinton et al. (2012), etc. In order for Deep Neural Networks to achieve such performance, specific architectures have been devised for each application. More precisely, each of these architectures relies on specific *structured linear transformations.*

## 1.3 Introducing Structure into Deep Neural Networks

A neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ composed of at least two linear functions (layers) and one non-linear function (activation function). The input space $n$ is usually large and the output space $m$ corresponds to the number of classes the network has to classify; therefore, we have $m \ll n$. We can write two layers neural network as follows:

$$f(\mathbf{x}) = \mathbf{W}_2 \rho(\mathbf{W}_1 \mathbf{x}) \tag{1.4}$$

where $\mathbf{W}_1 \in \mathbb{R}^{n \times n}$, $\mathbf{W}_2 \in \mathbb{R}^{m \times n}$ are dense matrices. This architecture is called *Fully Connected Neural Network* because all the neurons from the first activation are connected to all the neurons from the second activation. This type of network can have a large number of parameters, for example, with the MNIST dataset (LeCun et al. 1998) which consists of $28 \times 28$ images of hand-written digits from 0 to 9, the network from equation 1.4 will have more than $6 \times 10^5$ parameters.

Training such a large network has a number of significant drawbacks: they are hard to train, subject to overfitting and are computationally expensive. To overcome these drawbacks, researchers have devised neural networks with structure linear operations in order to reduce the number or parameters needed.

$$\begin{pmatrix} a & & & \\ & b & & \\ & & c & \\ & & & d \end{pmatrix} \qquad \begin{pmatrix} a & b & c & d \\ e & a & b & c \\ f & e & a & b \\ d & f & e & a \end{pmatrix} \qquad \begin{pmatrix} ae & af & ag & ah \\ be & bf & bg & bh \\ ce & cf & cg & ch \\ de & df & dg & dh \end{pmatrix} \qquad \begin{pmatrix} a & a^2 & a^3 & a^4 \\ b & b^2 & b^3 & b^4 \\ c & c^2 & c^3 & c^4 \\ d & d^2 & d^3 & d^4 \end{pmatrix}$$

diagonal Toeplitz Low Rank Vandermonde

Figure 1.3: Examples of structured matrices.

A $n \times n$ structure matrix can be represented with less than $n^2$ parameters, Figure 1.3 shows an example of structured matrices. In addition to offering a more compact representation, the structure of certain matrices can be leveraged to obtain better algorithms for matrix-vector product leading in memory and computationally operations.

Important questions remain, which structure is better suited for neural networks ? Does the reduction of parameters reduce the performance of the network ? Which properties can we leverage from these structures to improve the training and performance of neural networks ?

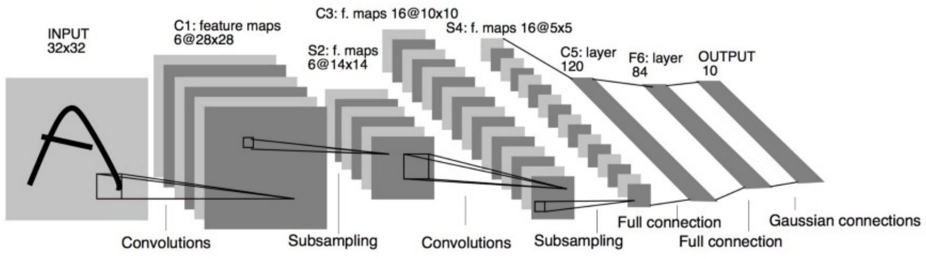


Figure 1.4: Graphical representation of the LeNet architecture proposed by LeCun et al. (1998)

A perfect example of such efficient linear operation is the discrete convolution which consists of a kernel sliding over the image and acting as a filter. Let $\mathbf{a}$ and $\mathbf{b}$ be two complex-valued vectors indexed by the set $M = \{-m, -m+1, \ldots, m-1, m\}$, the discrete convolution between the signals $\mathbf{a}$ and $\mathbf{b}$ is given by:

$$(\mathbf{a} * \mathbf{b})[n] \triangleq \sum_{m \in M} \mathbf{a}[m] \cdot \mathbf{b}[n-m] \tag{1.5}$$

The convolution operation has translation invariance characteristics (Zhang et al. 1990) which is perfectly suited for image classification (objects can be at different positions in images). LeCun et al. (1998) was one of the first to successfully train a *convolutional neural network* (CNN), achieving state-of-the-art performance on the MNIST dataset. Convolution neural networks achieve such a great result for two main reasons: first, CNNs are similar to the connectivity pattern of neurons in the visual cortex of the human brain. Secondly, CNNs are very efficient due to the sharing of parameters. While a classical linear operation with dense matrix has $n \times n$ parameters, a convolution only has $k \times k$ parameters where $k$ is the kernel size and is usually small (*e.g.* 3 or 5 for classical convolution layers use in neural networks). This weight sharing makes the network very small with respect to the Fully Connected neural network. The LeNet architecture (see Figure 1.4) has only $6 \times 10^4$ parameters.

## 1.4 Main contributions of the Thesis

The contributions of this Thesis are based on structured matrices from the Toeplitz family. More specifically, in Chapter 2, we use circulant matrices, which are a particular case of Toeplitz matrices, to devise a new compact architecture replacing Fully Connected Neural Networks. In Chapter 3, we leverage the structure of convolutional layers to devise a new regularization scheme for neural networks.

In Chapter 2, we study deep diagonal circulant neural networks, which are deep neural networks in which weight matrices are the product of diagonal and circulant ones. Besides making a theoretical analysis of their expressivity, we introduce principled techniques for training these models: we devise an initialization scheme and propose a smart use of non-linearity functions in order to train deep diagonal circulant networks. Furthermore, we show that these networks outperform recently introduced deep networks with other types of structured layers. We conduct a thorough experimental study to compare the performance of deep diagonal circulant networks with state-of-the-art models based on structured matrices and with dense models. We show that our models achieve better accuracy than other structured approaches while requiring 2x fewer weights than the next best approach. Finally, we train compact and accurate deep diagonal circulant networks on a real world video classification dataset with over 3.8 million training examples.

Finally, in Chapter 3, we tackle the problem of Lipschitz regularization of Convolutional Neural Networks. Lipschitz regularity is now established as a key property of modern deep learning with implications in training stability, generalization, robustness against adversarial examples, etc. However, computing the exact value of the Lipschitz constant of a neural network is known to be NP-hard. Recent attempts from the literature introduce upper bounds to approximate this constant that are either efficient but loose or accurate but computationally expensive. In this work, by leveraging the theory of Toeplitz matrices, we introduce a new upper bound for convolutional layers that is both tight and easy to compute. Based on this result we devise an algorithm to train Lipschitz regularized Convolutional Neural Networks.

# 2 Diagonal Circulant Neural Networks

## Contents

## 2.1 Introduction

The deep learning revolution has yielded models of increasingly large size. In recent years, designing compact and accurate neural networks with a small number of trainable parameters has been an active research topic. It is motivated by practical applications in embedded systems (to reduce memory footprint (Tara Sainath, 2015)), federated and distributed learning (to reduce communication (Konečný et al. 2016)), derivative-free optimization in reinforcement learning (to simplify the computation of the approximated gradient (Choromanski et al. 2018)), etc. Besides a number of practical applications, it is also an important research question whether or not models really need to be this large or if smaller networks can achieve similar accuracy (Ba & Caruana, 2014).

Structured matrices are at the very core of most of the work on compact networks. In these models, dense weight matrices are replaced by matrices with a prescribed structure (*low rank matrices, Toeplitz matrices, circulant matrices, LDR, etc.*). Despite substantial efforts (Cheng et al. 2015; Moczulski et al. 2015), the performance of compact models is still far from achieving an acceptable accuracy motivating their use in real-world scenarios. This raises several questions about the effectiveness of such models and about our ability to train them. In particular two main questions call for investigation:

**Q1** *How to efficiently train deep neural networks with a large number of structured layers?*

**Q2** *What is the expressive power of structured layers compared to dense layers?*

In this paper, we provide principled answers to these questions for the particular case of deep neural networks based on diagonal and circulant matrices (*a.k.a* Diagonal-circulant neural networks or DCNNs).

The idea of using diagonal and circulant matrices together comes from a series of results in linear algebra by Müller-Quade et al. (1998) and Huhtanen & Perämäki (2015). The most recent result from Huhtanen & Perämäki (2015) demonstrates that any matrix $\mathbf{A} \in \mathbb{C}^{n \times n}$ can be decomposed into the product of $2n-1$ alternating diagonal and circulant matrices. The diagonal-circulant decomposition inspired Moczulski et al. (2015) to design the *Structured Efficient Linear Layers* (SELL), which is the building block of DCNNs. However, they were not able to train deep neural networks based on these layers.

To answer **Q1**, we first describe a theoretically sound initialization procedure for DCNN which allows the signal to propagate through the network without vanishing or exploding. Furthermore, we provide a number of empirical insights to explain the behaviour of DCNNs and show the impact of the number of the non-linearities in the network on the convergence rate and the accuracy of the network. By combining all these insights, we are able (for the first time) to train large and deep DCNNs and demonstrate the good performance of these networks on a large scale application (the *YouTube-8M* video classification problem) and obtain very competitive accuracy.

To answer **Q2**, we propose an analysis of the expressivity of DCNNs by extending the results by Huhtanen & Perämäki (2015). We introduce a new bound on the number of diagonal-circulant products required to approximate a matrix that depends on its rank. Building on this result, we demonstrate that a DCNN with bounded width and small depth can approximate any dense networks with ReLU activations.

Outline of the chapter: We present in Section 2.2 the related work on structured neural networks and several compression techniques. Section 2.3 introduces circulant matrices, our new result extending the one from Huhtanen & Perämäki (2015). Section 2.4 proposes a theoretical analysis on the expressivity on DCNNs. Section 2.5 describes two efficient techniques for training deep diagonal circulant neural networks. Finally, Section 2.6 presents extensive experiments to compare the performance of deep diagonal circulant neural networks in different settings with respect to other state of the art approaches. Section 2.7 provides a discussion and concluding remarks.

## 2.2 Related Work

Structured matrices exhibit a number of good properties which have been exploited by deep learning practitioners, mainly to compress large neural networks architectures into smaller ones. For example, Hinrichs & Vybiral (2011) have demonstrated that a single circulant matrix can be used to approximate the Johnson-Lindenstrauss transform, often used in machine learning to perform dimensionality reduction. Building upon this result, Cheng et al. (2015) proposed to replace the weight matrix of a fully connected layer by a circulant matrix effectively replacing the complex transform modeled by the fully connected layer by a simple dimensionality reduction. Despite the reduction of expressivity, the resulting network demonstrated good accuracy using only a fraction of its original size (90% reduction).

Comparison with ACDC. Moczulski et al. (2015) have introduced two *Structured Efficient Linear Layers* (SELL) called AFCF and ACDC, where $\mathbf{A}$ and $\mathbf{D}$ are diagonal matrices and $\mathbf{F}$ and $\mathbf{C}$ are the Fourier and cosine transform respectively. The AFCF structured layer benefits from the theoretical results introduced by Huhtanen & Perämäki (2015) and can be seen as the building block of DCNNs. However, Moczulski et al. (2015) only experiment using ACDC, a different type of layer that does not involve circulant matrices. As far as we can tell, the theoretical guarantees available for the AFCF layer do not apply on the ACDC layer since the cosine transform does not diagonalize circulant matrices (Sanchez et al. 1995). Another possible limit of the ACDC paper is that they only train large neural networks involving ACDC layers combined with many other expressive layers. Although the resulting network demonstrates good accuracy, it is difficult the characterize the true contribution of the ACDC layers in this setting.

Comparison with Low displacement rank structures. More recently, Thomas et al. (2018) have generalized these works by proposing neural networks with low-displacement rank matrices (LDR), that are structured matrices encompassing a large family of structured matrices, including Toeplitz-like, Vandermonde-like, Cauchy-like and more notably DCNNs. To obtain this result, LDR represents a structured matrix using two displacement operators and a low-rank residual. Despite being elegant and general, we found that the LDR framework suffers from several limits which are inherent to its generality and makes it difficult to use in the context of large and deep neural networks. First, the training procedure for learning LDR matrices is highly involved and implies many complex mathematical objects such as Krylov matrices. Then, as acknowledged by the authors, the number of parameters required to represent a given structured matrix (a *Toeplitz matrix*) in practice is unnecessarily high (higher than required in theory).

Other compression techniques. Besides structured matrices, a variety of techniques have been proposed to build more compact deep learning models. These include *model distillation* (Hinton et al. 2015), Tensor Train (Novikov et al. 2015), Low-rank decomposition (Denil et al. 2013), to mention a few. However, circulant networks show good performances in several contexts (the interested reader can refer to the results reported by Moczulski et al. (2015) and Thomas et al. (2018)).

## 2.3 A Primer on Circulant Matrices and a New Result

An $n$-by-$n$ circulant matrix $\mathbf{C}$ is a matrix where each row is a cyclic right shift of the previous one as illustrated below.

$$\mathbf{C} = \mathrm{circ}(\mathbf{c}) = \begin{pmatrix} c_0 & c_{n-1} & c_{n-2} & \cdots & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ c_2 & c_1 & c_0 & & c_3 \\ \vdots & & & \ddots & \vdots \\ c_{n-1} & c_{n-2} & c_{n-3} & & c_0 \end{pmatrix} \tag{2.1}$$

Circulant matrices exhibit several interesting properties from the perspective of numerical computations. Most importantly, any $n$-by-$n$ circulant matrix $\mathbf{C}$ can be represented using only $n$ coefficients instead of the $n^2$ coefficients required to represent classical unstructured matrices. In addition, the matrix-vector product is simplified from $O(n^2)$ to $O(n \log n)$ using the convolution theorem.

As we will show in this chapter, circulant matrices also have a strong expressive power. So far, we know that a single circulant matrix can be used to represent a variety of important linear transforms such as random projections (Hinrichs & Vybiral, 2011). When they are combined with diagonal matrices, they can also be used as building blocks to represent any linear transform (Schmid et al. 2000; Huhtanen & Perämäki, 2015) with an arbitrary precision. Huhtanen & Perämäki (2015) were able to bound the number of factors that is required to approximate any matrix $\mathbf{A}$ with arbitrary precision.

## Relation between diagonal circulant matrices and low rank matrices

We recall this result in Theorem 1 as it is the starting point of our theoretical analysis.

**Theorem 1** (Reformulation from Huhtanen & Perämäki (2015))**.** *For every matrix $\mathbf{A} \in \mathbb{C}^{n \times n}$, for any $\epsilon > 0$, there exists a sequence of matrices $\mathbf{B}_1 \dots \mathbf{B}_{2n-1}$ where $\mathbf{B}_i$ is a circulant matrix if $i$ is odd, and a diagonal matrix otherwise, such that $\|\mathbf{B}_1\mathbf{B}_2 \dots \mathbf{B}_{2n-1} - \mathbf{A}\| < \epsilon$.*

Unfortunately, this theorem is of little use to understand the expressive power of diagonal-circulant matrices when they are used in deep neural networks. This is because: 1) the bound only depends on the dimension of the matrix $\mathbf{A}$, not on the matrix itself, 2) the theorem does not provide any insights regarding the expressive power of $m$ diagonal-circulant factors when $m$ is much lower than $2n-1$ as it is the case in most practical scenarios we consider in this chapter.

In the following theorem, we enhance the result by Huhtanen & Perämäki (2015) by expressing the number of factors required to approximate $\mathbf{A}$, *as a function of the rank of* $\mathbf{A}$. This is useful when one deals with low-rank matrices, which is common in machine learning problems.

**Theorem 2** (Rank-based circulant decomposition)**.** *Let $\mathbf{A} \in \mathbb{C}^{n \times n}$ be a matrix of rank at most $k$. Assume that $n$ can be divided by $k$. For any $\epsilon > 0$, there exists a sequence of $4k+1$ matrices $\mathbf{B}_1, \dots, \mathbf{B}_{4k+1}$, where $\mathbf{B}_i$ is a circulant matrix if $i$ is odd, and a diagonal matrix otherwise, such that $\|\mathbf{B}_1\mathbf{B}_2 \dots \mathbf{B}_{4k+1} - \mathbf{A}\| < \epsilon$.*

***Proof of Theorem** 2.* Let $\mathbf{U\Sigma V}^T$ be the SVD decomposition of $\mathbf{M}$ where $\mathbf{U}, \mathbf{V}$ and $\mathbf{\Sigma}$ are $n \times n$ matrices. Because $\mathbf{M}$ is of rank $k$, the last $n-k$ columns of $\mathbf{U}$ and $\mathbf{V}$ are null. In the following, we will first decompose $\mathbf{U}$ into a product of matrices

**WRO**, where **R** and **O** are respectively circulant and diagonal matrices, and **W** is a matrix which will be further decomposed into a product of diagonal and circulant matrices. Then, we will apply the same decomposition technique to **V**. Ultimately, we will get a product of $4k+2$ matrices alternatively diagonal and circulant.

Let $\mathbf{R} = \mathrm{circ}(r_1 \dots r_n)$. Let $\mathbf{O}$ be a $n \times n$ diagonal matrix where $\mathbf{O}_{i,i} = 1$ if $i \leq k$ and 0 otherwise. The $k$ first columns of the product **RO** will be equal to that of **R**, and the $n-k$ last colomns of **RO** will be zeros. For example, if $k=2$, we have:

$$\mathbf{RO} = \begin{pmatrix} r_1 & r_n & 0 & \cdots & 0 \\ r_2 & r_1 & & & \\ r_3 & r_2 & \vdots & & \vdots \\ \vdots & \vdots & & & \\ r_n & r_{n-1} & 0 & \cdots & 0 \end{pmatrix} \tag{2.2}$$

Let us define $k$ diagonal matrices $\mathbf{D}_i = \mathrm{diag}(d_{i1} \dots d_{in})$ for $i \in [k]$. For now, the values of $d_{ij}$ are unknown, but we will show how to compute them. Let $\mathbf{W} = \sum_{i=1}^{k} \mathbf{D}_i \mathbf{S}^{i-1}$ where **S** is the *cyclic shift* matrix define as follows:

$$\mathbf{S} = \begin{pmatrix} 0 & & & & 1 \\ 1 & 0 & & & \\ & 1 & \ddots & & \\ & & \ddots & 0 & \\ & & & 1 & 0 \end{pmatrix} \tag{2.3}$$

Note that the $n-k$ last columns of the product **WRO** will be zeros. For example, with $k=2$, we have:

$$\mathbf{W} = \begin{pmatrix} d_{1,1} & & & & d_{2,1} \\ d_{2,2} & d_{1,2} & & & \\ & d_{2,3} & \ddots & & \\ & & \ddots & & \\ & & & d_{2,n} & d_{1,n} \end{pmatrix} \tag{2.4}$$

$$\mathbf{WRO} = \begin{pmatrix} r_1 d_{11} + r_n d_{21} & r_n d_{11} + r_{n-1} d_{21} & 0 & \cdots & 0 \\ r_2 d_{12} + r_1 d_{22} & r_1 d_{12} + r_n d_{22} & & & \\ & & \vdots & & \vdots \\ \vdots & \vdots & & & \\ r_n d_{1n} + r_{n-1} d_{2n} & r_{n-1} d_{1n} + r_{n-2} d_{2n} & 0 & \cdots & 0 \end{pmatrix} \tag{2.5}$$

We want to find the values of $d_{ij}$ such that $\mathbf{WRO} = \mathbf{U}$. We can formulate this as linear equation system. In case $k = 2$, we get:

$$\begin{pmatrix} r_n & r_1 & & & & & & \\ r_{n-1} & r_n & & & & & & \\ & & r_1 & r_2 & & & & \\ & & r_n & r_1 & & & & \\ & & & & r_2 & r_3 & & \\ & & & & r_1 & r_2 & & \\ & & & & & & \ddots & \\ & & & & & & & \ddots \end{pmatrix} \times \begin{pmatrix} d_{2,1} \\ d_{1,1} \\ d_{2,2} \\ d_{1,2} \\ d_{2,3} \\ d_{1,3} \\ \vdots \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{U}_{1,1} \\ \mathbf{U}_{1,2} \\ \mathbf{U}_{2,1} \\ \mathbf{U}_{2,2} \\ \\ \\ \vdots \\ \\ \end{pmatrix} \tag{2.6}$$

The $i^{th}$ bloc of the bloc-diagonal matrix is a Toeplitz matrix induced by a subsequence of length $k$ of $(r_1, \ldots r_n, r_1 \ldots r_n)$. Set $r_j = 1$ for all $j \in \{k, 2k, 3k, \ldots n\}$ and set $r_j = 0$ for all other values of $j$. Then it is easy to see that each bloc is a permutation of the identity matrix. Thus, all blocs are invertible. This entails that the block diagonal matrix above is also invertible. So by solving this set of linear equations, we find $d_{1,1} \ldots d_{k,n}$ such that $\mathbf{WRO} = \mathbf{U}$. We can apply the same idea to factorize $\mathbf{V} = \mathbf{W}'.\mathbf{R}.\mathbf{O}$ for some matrix $\mathbf{W}'$. Finally, we get

$$\mathbf{A} = \mathbf{U\Sigma V}^\top = \mathbf{WRO\Sigma O}^\top \mathbf{R}^\top \mathbf{W}'^\top \tag{2.7}$$

Thanks to Theorem 1, $\mathbf{W}$ and $\mathbf{W}'$ can both be factorized in a product of $2k - 1$ circulant and diagonal matrices. Note that $\mathbf{O\Sigma O}^\top$ is diagonal, because all three are diagonal. Overall, $\mathbf{A}$ can be represented with a product of $4k + 2$ matrices, alternatively diagonal and circulant. ■

A direct consequence of Theorem 2, is that if the number of diagonal-circulant factors is set to a value $K$, we can represent all linear transform $\mathbf{A}$ whose rank is $\frac{K-1}{4}$.

Compared to Huhtanen & Perämäki (2015), this result shows that structured matrices with fewer than $2n$ diagonal-circulant matrices (as it is the case in practice) can still represent a large class of matrices. As we will show in the following section, this result will be useful to analyze the expressivity of neural networks based on diagonal and circulant matrices.

## 2.4 Analysis of Diagonal Circulant Neural Networks

Zhao et al. (2017) have shown that circulant networks with 2 layers and unbounded width are universal approximators. However, results on unbounded networks offer weak guarantees and two important questions have remained open until now:

1. Can we approximate any function with a bounded-width circulant networks?
2. What function can we approximate with a circulant network that has a bounded width and a small depth?

We answer these two questions in this section. First, we introduce some necessary definitions regarding neural networks and we provide a theoretical analysis of their approximation capabilities.

**Definition 1** (Complex ReLU function Trabelsi et al. (2018))**.** *Let us define the complex ReLU function* $\mathrm{ReLU} : \mathbb{C}^n \to \mathbb{C}^n$ *by:* $\mathrm{ReLU}(\mathbf{z}) = \max(0, \Re(\mathbf{z})) + \mathbf{i}\max(0, \Im(\mathbf{z}))$

**Definition 2** (Deep ReLU network)**.** *Given $L$ weight matrices $\mathbf{W} = (\mathbf{W}_1, \ldots, \mathbf{W}_L)$ with $\mathbf{W}_i \in \mathbb{C}^{n\times n}$ and $L$ bias vectors $\mathbf{b} = (\mathbf{b}_1, \ldots, \mathbf{b}_L)$ with $\mathbf{b}_i \in \mathbb{C}^n$, a* deep ReLU network *is a function $f_{\mathbf{W}_L,\mathbf{b}_L} : \mathbb{C}^n \to \mathbb{C}^n$ such that $f_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = (f_{\mathbf{W}_L,\mathbf{b}_L} \circ \ldots \circ f_{\mathbf{W}_1,\mathbf{b}_1})(\mathbf{x})$ where $f_{\mathbf{W}_i,\mathbf{b}_i}(\mathbf{x}) = \phi(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i)$ and $\phi(\;\cdot\;)$ is a* ReLU *non-linearity In the rest of this chapter, we call $L$ and $n$ respectively the depth and the width of the network.*

**Definition 3** (Total Rank)**.** *The Total Rank $k$ of the Neural Network $f_{\mathbf{W},\mathbf{b}}$ corresponds to the sum of the ranks of the matrices $W_1 \ldots W_L$. i.e. $k = \sum_{i=1}^{L} \mathrm{rank}(W_i)$.*

We also need to introduce DCNNs, similarly to Moczulski et al. (2015).

**Definition 4** (Diagonal Circulant Neural Networks)**.** *Given $L$ diagonal matrices $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_L)$ with $\mathbf{D}_i \in \mathbb{C}^{n\times n}$, $L$ circulant matrices $\mathbf{C} = (\mathbf{C}_1, \ldots, \mathbf{C}_L)$ with $\mathbf{C}_i \in \mathbb{C}^{n\times n}$ and $L$ bias vectors $\mathbf{b} = (\mathbf{b}_1, \ldots, \mathbf{b}_L)$ with $\mathbf{b}_i \in \mathbb{C}^n$, a* Diagonal Circulant Neural Networks *(DCNN) is a function $f_{\mathbf{W}_L,\mathbf{b}_L} : \mathbb{C}^n \to \mathbb{C}^n$ such that $f_{\mathbf{D},\mathbf{C},\mathbf{b}}(\mathbf{x}) =$*

$(f_{\mathbf{D}_L,\mathbf{C}_L,\mathbf{b}_L} \circ \ldots \circ f_{\mathbf{D}_1,\mathbf{C}_1,\mathbf{b}_1})(\mathbf{x})$ *where* $f_{\mathbf{D}_i,\mathbf{C}_i,\mathbf{b}_i}(\mathbf{x}) = \phi_i(\mathbf{D}_i\mathbf{C}_i\mathbf{x}+\mathbf{b}_i)$ *and where* $\phi_i(\,\cdot\,)$ *is a* ReLU *non-linearity or the identity function.*

We can now show that bounded-width DCNNs can approximate any Deep ReLU Network, and as a corollary, that they are universal approximators.

**Lemma 1.** *Let* $W_L, \ldots W_1 \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$ *and let* $\mathcal{X} \subset \mathbb{C}^n$ *be a bounded set. There exists* $\beta_L \ldots \beta_1 \in \mathbb{C}^n$ *such that for all* $x \in \mathcal{X}$ *we have* $f_{W_L,\beta_L} \circ \ldots \circ f_{W_1,\beta_1}(x) = \mathrm{ReLU}(W_L W_{L-1} \ldots W_1 x + b)$.

***Proof of Lemma 1.*** Define $S = \left\{ \left( \left( \prod_{k=1}^{j} \mathbf{W}_k \right) \mathbf{x} \right)_t : \mathbf{x} \in \mathcal{X}, t \in [n], j \in [L] \right\}$. Let $\Omega = \max\{\Re(v) : v \in S\} + \mathbf{i}\max\{\Im(v) : v \in S\}$. Intuitively, the real and imaginary parts of $\Omega$ are the largest any activation in the network can have. Define $h_j(\mathbf{x}) = \mathbf{W}_j\mathbf{x}+\beta_j$. Let $\beta_1 = \Omega\mathbf{1}_n$. Clearly, for all $\mathbf{x} \in \mathcal{X}$ we have $h_1(\mathbf{x}) \geq 0$, so $\mathrm{ReLU} \circ h_1(\mathbf{x}) = h_1(\mathbf{x})$. More generally, for all $j < n-1$ define $\beta_{j+1} = \mathbf{1}_n\Omega - \mathbf{W}_{j+1}\beta_j$. It is easy to see that for all $j < n$ we have $h_j \circ \ldots \circ h_1(\mathbf{x}) = \mathbf{W}_j\mathbf{W}_{j-1} \ldots \mathbf{W}_1 x + \mathbf{1}_n\Omega$. This guarantees that for all $j < n$, $h_j \circ \ldots \circ h_1(\mathbf{x}) = \mathrm{ReLU} \circ h_j \circ \ldots \circ \mathrm{ReLU} \circ h_1(\mathbf{x})$. Finally, define $\beta_L = b - A_L\beta_{L-1}$. We have, $\mathrm{ReLU} \circ h_L \circ \ldots \circ \mathrm{ReLU} \circ h_1(\mathbf{x}) = \mathrm{ReLU}(\mathbf{W}_j \ldots \mathbf{W}_1\mathbf{x} + b)$. ■

**Lemma 2.** *Let* $\mathcal{N}$ *be a deep ReLU network of width* $n$ *and depth* $L$*, and let* $\mathcal{X} \subset \mathbb{C}^n$ *be a bounded set. For any* $\epsilon > 0$*, there exists a DCNN* $\mathcal{N}'$ *of width* $n$ *and of depth* $(2n-1)L$ *such that* $\|\mathcal{N}(\mathbf{x}) - \mathcal{N}'(\mathbf{x})\| < \epsilon$ *for all* $\mathbf{x} \in \mathcal{X}$.

***Proof of Lemma 2.*** Assume $\mathcal{N} = f_{W_L,b_L} \circ \ldots \circ f_{W_1,b_1}$. By Theorem 1, for any $\epsilon' > 0$, any matrix $\mathbf{W}_i$, there exists a sequence of $2n-1$ matrices $\mathbf{C}_{i,n}\mathbf{D}_{i,n-1}\mathbf{C}_{i,n-1} \ldots \mathbf{D}_{i,1}\mathbf{C}_{i,1}$ such that

$$\left\| \prod_{j=0}^{n-1} \mathbf{D}_{i,n-j}\mathbf{C}_{i,n-j} - \mathbf{W}_i \right\| < \epsilon', \tag{2.8}$$

where $D_{i,1}$ is the identity matrix. By Lemma 1, we know that there exists $\{\beta_{ij}\}_{i\in[L],j\in[n]}$ such that for all $i \in [L]$,

$$f_{\mathbf{D}_{in}\mathbf{C}_{in},\beta_{in}} \circ \ldots \circ f_{\mathbf{D}_{i1}\mathbf{C}_{i1},\beta_{i1}}(\mathbf{x}) = \mathrm{ReLU}(\mathbf{D}_{in}\mathbf{C}_{in} \ldots \mathbf{C}_{i1}\mathbf{x} + \mathbf{b}_i). \tag{2.9}$$

Now if $\epsilon'$ tends to zero, $\|f_{\mathbf{D}_{in}\mathbf{C}_{in},\beta_{in}} \circ \ldots \circ f_{\mathbf{D}_{i1}\mathbf{C}_{i1},\beta_{i1}} - \mathrm{ReLU}(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i)\|$ will also tend to zero for any $\mathbf{x} \in \mathcal{X}$, because the ReLU function is continuous and $\mathcal{X}$ is bounded. Let $\mathcal{N}' = f_{\mathbf{D}_{1n}\mathbf{C}_{1n},\beta_{1n}} \circ \ldots \circ f_{\mathbf{D}_{i1}\mathbf{C}_{i1},\beta_{i1}}$. Again, because all functions are continuous, for all $\mathbf{x} \in \mathcal{X}$, $\|\mathcal{N}(\mathbf{x}) - \mathcal{N}'(\mathbf{x})\|$ tends to zero as $\epsilon'$ tends to zero. ■

We can now state the universal approximation corollary:

**Corollary 1.** *Bounded width DCNNs are universal approximators in the following sense: for any continuous function $f : [0,1]^n \rightarrow \mathbb{R}_+$ of bounded supremum norm, for any $\epsilon > 0$, there exists a DCNN $\mathcal{N}_\epsilon$ of width $n+3$ such that $\forall \mathbf{x} \in [0,1]^{n+3}$, $|f(\mathbf{x}_1 \ldots \mathbf{x}_n) - (\mathcal{N}_\epsilon(\mathbf{x}))_1| < \epsilon$, where $(\,\cdot\,)_i$ represents the $i^{th}$ component of a vector.*

***Proof of Corollary** 1.* It has been shown recently by Hanin (2017) that for any continuous function $f : [0,1]^n \rightarrow \mathbb{R}_+$ of bounded supremum norm, for any $\epsilon > 0$, there exists a dense neural network $\mathcal{N}$ with an input layer of width $n$, an output layer of width 1, hidden layers of width $n+3$ and ReLU activations such that $\forall x \in [0,1]^n, |f(\mathbf{x}) - \mathcal{N}(\mathbf{x})| < \epsilon$. From $\mathcal{N}$, we can easily build a deep ReLU network $\mathcal{N}'$ of width exactly $n+3$, such that $\forall x \in [0,1]^{n+3}$, $|f(\mathbf{x}_1 \ldots \mathbf{x}_n) - (\mathcal{N}'(\mathbf{x}))_1| < \epsilon$. Thanks to Lemma 2, this last network can be approximated arbitrarily well by a DCNN of width $n+3$. ■

This is a first result, however $(2n+5)L$ is not a small depth (in our experiments, $n$ can be over 300 000), and a number of work provided empirical evidences that DCNN with small depth can offer good performances (Araujo et al. 2018; Cheng et al. 2015). To improve our result, we introduce our main theorem which studies the approximation properties of these small depth networks.

**Theorem 3** (Rank-based expressive power of DCNNs)**.** *Let $\mathcal{N}$ be a deep ReLU network of width $n$, depth $L$ and a total rank $k$ and assume $n$ is a power of 2. Let $\mathcal{X} \subset \mathbb{C}^n$ be a bounded set. Then, for any $\epsilon > 0$, there exists a DCNN with ReLU activation $\mathcal{N}'$ of width $n$ such that $\|\mathcal{N}(\mathbf{x}) - \mathcal{N}'(\mathbf{x})\| < \epsilon$ for all $\mathbf{x} \in \mathcal{X}$ and the depth of $\mathcal{N}'$ is bounded by $9k$.*

***Proof of Theorem** 3.* Let $k_1 \ldots k_L$ be the ranks of matrices $\mathbf{W}_1 \ldots \mathbf{W}_L$, which are $n$-by-$n$ matrices. For all $i$, there exists $k'_i \in \{k_i \ldots 2k_i\}$ such that $k'_i$ is a power of 2. Due to the fact that $n$ is also a power of 2, $k'_i$ divides $n$. By Theorem 2, for all $i$ each matrix $\mathbf{W}_i$ can be decomposed as an alternating product of diagonal-circulant matrices $\mathbf{B}_{i,1} \ldots \mathbf{B}_{i,4k'_i+1}$ such that $\left\|\mathbf{W}_i - \mathbf{B}_{i,1} \ldots \mathbf{B}_{i,4k'_i+1}\right\| < \epsilon$. Using the exact same technique as in Lemma 2, we can build a DCNN $\mathcal{N}'$ using matrices $\mathbf{B}_{1,1} \ldots \mathbf{B}_{L,4k'_L+1}$, such that $\|\mathcal{N}(\mathbf{x}) - \mathcal{N}'(\mathbf{x})\| < \epsilon$ for all $\mathbf{x} \in \mathcal{X}$. The total number of layers is $\sum_i (4k'_i + 1) \leq L + 8 \sum_i k_i \leq L + 8.\text{total rank} \leq 9.\text{total rank}$. ■

Remark that in the theorem, we require that $n$ is a power of 2. We conjecture that the result still holds even without this condition. This result refines Lemma 2, and answer our second question: a DCNN of bounded width and small depth can approximate a Deep ReLU network of low total rank. Note that the converse is not true: because $n$-by-$n$ circulant matrix can be of rank $n$, approximating a DCNN of depth 1 can require a deep ReLU network of total rank equals to $n$.

Finally, what if we choose to use small depth networks to approximate deep ReLU networks where matrices are not of low rank? To answer this question, we first need to show the negative impact of replacing matrices by their low rank approximators in neural networks:

**Proposition 1.** *Let $\mathcal{N} = f_{\mathbf{W}_L,\mathbf{b}_L} \circ \ldots \circ f_{\mathbf{W}_1,\mathbf{b}_1}$ be a Deep ReLU network, where $\mathbf{W}_i \in \mathbb{C}^{n\times n}, \mathbf{b}_i \in \mathbb{C}^n$ for all $i \in [L]$. Let $\tilde{\mathbf{W}}_i$ be the matrix obtained by an SVD approximation of rank $k$ of matrix $\mathbf{W}_i$. Let $\sigma_{i,j}$ be the $j^{th}$ singular value of $\mathbf{W}_i$. Define $\tilde{\mathcal{N}} = f_{\tilde{\mathbf{W}}_L,\mathbf{b}_L} \circ \ldots \circ f_{\tilde{\mathbf{W}}_1,\mathbf{b}_1}$. Then, for any $\mathbf{x} \in \mathbb{C}^n$, we have:*

$$\left\|\mathcal{N}(\mathbf{x}) - \tilde{\mathcal{N}}(\mathbf{x})\right\| \leq \frac{\left(\sigma_{max,1}^L - 1\right) R \sigma_{max,k}}{\sigma_{max,1} - 1} \tag{2.10}$$

*where $R$ is an upper bound on norm of the output of any layer in $\mathcal{N}$, and $\sigma_{max,j} = \max_i \sigma_{i,j}$.*

***Proof of Proposition 1.*** Let $\mathbf{x}_0 \in \mathbb{C}^n$ and $\tilde{\mathbf{x}}_0 = \mathbf{x}_0$. For all $i \in [L]$, define $\mathbf{x}_i = \mathrm{ReLU}(\mathbf{W}_i \mathbf{x}_{i-1} + \mathbf{b})$ and $\tilde{\mathbf{x}}_i = \mathrm{ReLU}\left(\tilde{\mathbf{W}}_i \tilde{\mathbf{x}}_{i-1} + \mathbf{b}\right)$. By Lemma 3, we have

$$\|\mathbf{x}_i - \tilde{\mathbf{x}}_i\| \leq \sigma_{i,k+1}\|\mathbf{x}_{i-1}\| + \sigma_{i,1}\|\mathbf{x}_{i-1} - \tilde{\mathbf{x}}_{i-1}\| \tag{2.11}$$

Observe that for any sequence $a_0, a_1 \ldots$ defined recurrently by $a_0 = 0$ and $a_i = r a_{i-1} + s$, the recurrence relation can be unfold as follows: $a_i = \frac{s(r^i - 1)}{r-1}$. We can apply this formula to bound our error as follows:

$$\|x_l - \tilde{x}_l\| \leq \frac{\left(\sigma_{max,1}^l - 1\right)\sigma_{max,k} \max_i \|x_i\|}{\sigma_{max,1} - 1} \tag{2.12}$$

■

**Lemma 3.** *Let $\mathbf{W} \in \mathbb{C}^{n\times n}$ with singular values $\sigma_1 \ldots \sigma_n$, and let $\mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{C}^n$. Let $\tilde{\mathbf{W}}$ be the matrix obtained by a SVD approximation of rank $k$ of matrix $\mathbf{W}$. Then we have:*

$$\left\|\mathrm{ReLU}(\mathbf{W}\mathbf{x} + \mathbf{b}) - \mathrm{ReLU}\left(\tilde{\mathbf{W}}\tilde{\mathbf{x}} + \mathbf{b}\right)\right\| \leq \sigma_{k+1}\|\mathbf{x}\| + \sigma_1\|\tilde{\mathbf{x}} - \mathbf{x}\| \tag{2.13}$$

***Proof of Lemma 3.*** Recall that $\|\mathbf{W}\|_2 = \sup_{\mathbf{z}} \frac{\|\mathbf{W}\mathbf{z}\|_2}{\|\mathbf{z}\|_2} = \sigma_1 = \left\|\tilde{\mathbf{W}}\right\|_2$, because $\sigma_1$ is the greatest singular value of both $\mathbf{W}$ and $\tilde{\mathbf{W}}$. Also, note that $\left\|\mathbf{W} - \tilde{\mathbf{W}}\right\|_2 = \sigma_{k+1}$. Let us bound the formula without ReLUs:

$$\begin{aligned}
\left\|(\mathbf{W}\mathbf{x} + \mathbf{b}) - \left(\tilde{\mathbf{W}}\tilde{\mathbf{x}} + \mathbf{b}\right)\right\| &= \left\|(\mathbf{W}\mathbf{x} + \mathbf{b}) - \left(\tilde{\mathbf{W}}\tilde{\mathbf{x}} + \mathbf{b}\right)\right\| && (2.14)\\
&= \left\|\mathbf{W}\mathbf{x} - \tilde{\mathbf{W}}\mathbf{x} - \tilde{\mathbf{W}}(\tilde{\mathbf{x}} - \mathbf{x})\right\| && (2.15)\\
&\le \left\|\left(\mathbf{W} - \tilde{\mathbf{W}}\right)\mathbf{x}\right\| + \left\|\tilde{\mathbf{W}}\right\|_2 \|\tilde{\mathbf{x}} - \mathbf{x}\| && (2.16)\\
&\le \|\mathbf{x}\|\sigma_{k+1} + \sigma_1\|\tilde{\mathbf{x}} - \mathbf{x}\| && (2.17)
\end{aligned}$$

Finally, it is easy to see that for any pair of vectors $\mathbf{a}, \mathbf{b} \in \mathbb{C}^n$, we have

$$\|\mathrm{ReLU}(\mathbf{a}) - \mathrm{ReLU}(\mathbf{b})\| \le \|\mathbf{a} - \mathbf{b}\|. \tag{2.18}$$

This concludes the proof. ■

**Corollary 2.** *Consider any deep ReLU network $\mathcal{N} = f_{W_L,b_L} \circ \ldots \circ f_{W_1,b_1}$ of depth $L$ and width $n$. Let $\sigma_{max,j} = \max_i \sigma_{i,j}$ where $\sigma_{i,j}$ is the $j^{th}$ singular value of $W_i$. Let $\mathcal{X} \subset \mathbb{C}^n$ be a bounded set. Let $k$ be an integer dividing $n$. There exists a DCNN $\mathcal{N}' = f_{D_m C_m, b'_m} \circ \ldots \circ f_{D_1 C_1, b'_1}$ of width $n$ and of depth $m = L(4k+1)$, such that for any $x \in \mathcal{X}$:*

$$\left\|\mathcal{N}(x) - \mathcal{N}'(x)\right\| < \frac{\left(\sigma_{max,1}^L - 1\right) R \sigma_{max,k}}{\sigma_{max,1} - 1} \tag{2.19}$$

*where $R$ is an upper bound on the norm of the outputs of each layer in $\mathcal{N}$.*

***Proof of Corollary 2.*** Let $\tilde{\mathcal{N}} = f_{\tilde{\mathbf{W}}_L,\mathbf{b}_L} \circ \ldots \circ f_{\tilde{\mathbf{W}}_1,\mathbf{b}_1}$, where each $\tilde{\mathbf{W}}_i$ is the matrix obtained by an SVD approximation of rank $k$ of matrix $\mathbf{W}_i$. With Proposition 1, we have an error bound on $\left\|\mathcal{N}(\mathbf{x}) - \tilde{\mathcal{N}}(\mathbf{x})\right\|$. Now each matrix $\tilde{\mathbf{W}}_i$ can be replaced by a product of $k$ diagonal-circulant matrices. By Theorem 3, this product yields a DCNN of depth $m = L(4k+1)$, strictly equivalent to $\tilde{\mathcal{N}}$ on $\mathcal{X}$. This conludes the proof. ■

Expressivity of DCNNs For the sake of clarity, we highlight the significance of these results with the two following properties.

Properties. Given an arbitrary fixed integer $n$, let $\mathcal{R}_k$ be the set of all functions $f : \mathbb{R}^n \to \mathbb{R}^n$ representable by a deep ReLU network of total rank at most $k$ and let $\mathcal{C}_l$ the set of all functions $f : \mathbb{R}^n \to \mathbb{R}^n$ representable by deep diagonal-circulant networks of depth at most $l$, then:

$$\forall k, \exists l \quad \mathcal{R}_k \subsetneq \mathcal{C}_l \tag{2.20}$$

$$\forall l, \nexists k \quad \mathcal{C}_l \subseteq \mathcal{R}_k \tag{2.21}$$

We illustrate the meaning of this properties using Figure 2.1. As we can see, the set $\mathcal{R}_k$ of all the functions representable by a deep ReLU network of total rank $k$ is strictly included in the set $\mathcal{C}_{9k}$ of all DCNN of depth $9k$ (as by Theorem 3).

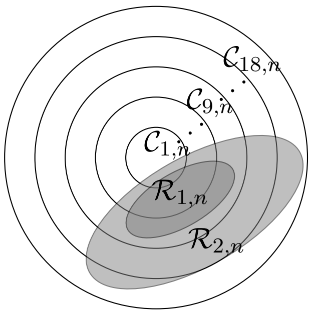
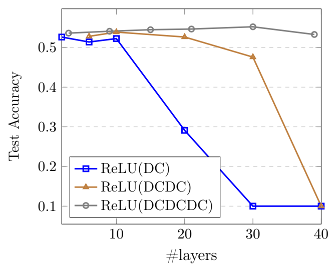


Figure 2.1: Illustration of Properties 2.20 and 2.21.

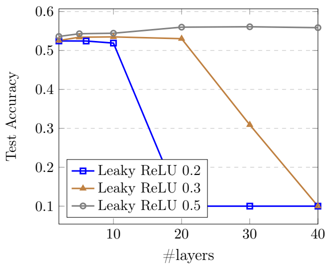
These properties are interesting for many reasons. First, Property 2.21 shows that diagonal-circulant networks are *strictly more expressive* than networks with low total rank. Second and most importantly, in standard deep neural networks, it is known that the most of the singular values are close to zero (see *e.g.* Sedghi et al. (2018) and Arora et al. (2019)). Property 2.20 shows that these networks can efficiently be approximated by diagonal-circulant networks. Finally, several publications have shown that neural networks can be trained explicitly to have low-rank weight matrices (C. Li & Shi, 2018; Goyal et al. 2019). This opens the possibility of learning compact and accurate diagonal-circulant networks.

## 2.5 How to Train Deep Diagonal Circulant Neural Networks

Training DCNNs has revealed to be a challenging problem. We devise two techniques to facilitate the training of deep DCNNs. First, we propose an initialization procedure which guarantee the signal is propagated across the network without vanishing nor exploding. Secondly, we study the behavior of DCNNs with different non-linearity functions and determine the best parameters for different settings.



(a) Impact of increasing the number of ReLU activations in a DCNN. Deep DCNNs with fewer ReLUs are easier to train.



(b) Impact of increasing the slope of a Leaky-ReLU in DCNNs. Deep DCNNs with a larger slope are easier to train.

Figure 2.2: Experiments on training DCNNs and other structured neural networks on CIFAR-10.

Initialization scheme The following initialization procedure which is a variant of Xavier initialization. First, for each circulant matrix $\mathbf{C} = \operatorname{circ}(c_1 \dots c_n)$, each $c_i$ is randomly drawn from $\mathcal{N}\big(0, \sigma^2\big)$, with $\sigma = \sqrt{\frac{2}{n}}$. Next, for each diagonal matrix $\mathbf{D} = \operatorname{diag}(d_1 \dots d_n)$, each $d_i$ is drawn randomly and uniformly from $\{-1, 1\}$ for all $i$. Finally, all biases in the network are randomly drawn from $\mathcal{N}\big(0, \sigma'^2\big)$, for some small value of $\sigma'$. The following proposition states that the covariance matrix at the output of any layer in a DCNN, independent of the depth, is constant.

**Proposition 2** (Initialization of DCNNs)**.** *Let $\mathcal{N}$ be a DCNN of depth $L$ initialized according to our procedure, with $\sigma' = 0$. Assume that all layers $1$ to $L-1$ have ReLU activation functions, and that the last layer has the identity activation function. Then, for any $\mathbf{x} \in \mathbb{R}^n$, the covariance matrix of $\mathcal{N}(\mathbf{x})$ is $\frac{2.Id}{n}\|\mathbf{x}\|_2^2$. Moreover, note that this covariance does not depend on the depth of the network.*

***Proof of Proposition 2.*** Let $\mathcal{N} = f_{\mathbf{D}_L,\mathbf{C}_L} \circ \dots \circ f_{\mathbf{D}_1,\mathbf{C}_1}$ be a $L$ layer DCNN. All matrices are initialized as described in the statement of the proposition. Let $\mathbf{y} = \mathbf{D}_1\mathbf{C}_1\mathbf{x}$. Lemma 4 shows that $\operatorname{Cov}(\mathbf{y}_i, \mathbf{y}_{i'}) = 0$ for $i \neq i'$ and $\operatorname{Var}(\mathbf{y}_i) = \frac{2}{n}\|x\|_2^2$. For any $j \leq L$, define $\mathbf{z}^j = f_{\mathbf{D}_j,\mathbf{C}_j} \circ \dots \circ f_{\mathbf{D}_1,\mathbf{C}_1}(\mathbf{x})$. By a recursive application of Lemma 4, we get that then $\operatorname{Cov}(\mathbf{z}_i^j, \mathbf{z}_{i'}^j) = 0$ and $\operatorname{Var}(\mathbf{z}_i^j) = \frac{2}{n}\|\mathbf{x}\|_2^2$. ■

**Lemma 4.** *Let $c_1 \dots c_n, d_1 \dots d_n, b_1 \dots b_n$ be random variables in $\mathbb{R}$ such that $c_i \sim \mathcal{N}(0, \sigma^2)$, $b_i \sim \mathcal{N}(0, \sigma'^2)$ and $d_i \sim \{-1, 1\}$ uniformly. Define $\mathbf{C} = \operatorname{circ}(c_1 \dots c_n)$ and $\mathbf{D} = \operatorname{diag}(d_1 \dots d_n)$. Define $\mathbf{y} = \mathbf{DCu}$ and $\mathbf{z} = \mathbf{CDu}$ for some vector $\mathbf{u}$ in $\mathbb{R}^n$. Also define $\bar{\mathbf{y}} = \mathbf{y} + \mathbf{b}$ and $\bar{\mathbf{z}} = \mathbf{z} + \mathbf{b}$. Then, for all $i$, the p.d.f. of $\mathbf{y}_i$, $\bar{\mathbf{y}}_i$, $\mathbf{z}_i$ and $\bar{\mathbf{z}}_i$ are symmetric. Also:*

- *Assume $u_1 \dots u_n$ is fixed. Then, we have for $i \neq i'$:*

$$\begin{aligned}\mathrm{Cov}(\mathbf{y}_i, \mathbf{y}_{i'}) = \mathrm{Cov}(\mathbf{z}_i, \mathbf{z}_{i'}) = \mathrm{Cov}(\bar{\mathbf{y}}_i, \bar{\mathbf{y}}_{i'}) &= \mathrm{Cov}(\bar{\mathbf{z}}_i, \bar{\mathbf{z}}_{i'}) = 0 \\ \mathrm{Var}(\mathbf{y}_i) = \mathrm{Var}(\mathbf{z}_i) &= \sum_j u_j^2 \sigma^2 \\ \mathrm{Var}(\bar{\mathbf{y}}_i) = \mathrm{Var}(\bar{\mathbf{z}}_i) &= \sigma'^2 + \sum_j \mathbf{u}_j^2 \sigma^2\end{aligned}$$

- *Let $x_1 \dots x_n$ be random variables in $\mathbb{R}$ such that the p.d.f. of $x_i$ is symmetric for all $i$, and let $u_i = \mathrm{ReLU}(x_i)$. We have for $i \neq i'$ :*

$$\begin{aligned}\mathrm{Cov}(\mathbf{y}_i, \mathbf{y}_{i'}) = \mathrm{Cov}(\mathbf{z}_i, \mathbf{z}_{i'}) = \mathrm{Cov}(\bar{\mathbf{y}}_i, \bar{\mathbf{y}}_{i'}) &= \mathrm{Cov}(\bar{\mathbf{z}}_i, \bar{\mathbf{z}}_{i'}) = 0 \\ \mathrm{Var}(\mathbf{y}_i) = \mathrm{Var}(\mathbf{z}_i) &= \frac{1}{2} \sum_j \mathrm{Var}(x_i) \cdot \sigma^2 \\ \mathrm{Var}(\bar{\mathbf{y}}_i) = \mathrm{Var}(\bar{\mathbf{z}}_i) &= \sigma'^2 + \frac{1}{2} \sum_j \mathrm{Var}(x_i) \cdot \sigma^2\end{aligned}$$

***Proof of Lemma 4.*** By an abuse of notation, we write $c_0 = c_n, c_{-1} = c_{n-1}$ and so on. First, note that: $y_i = \sum_{j=1}^n c_{j-i} u_j d_j$ and $z_i = \sum_{j=1}^n c_{j-i} u_j d_i$. Observe that each term $c_{j-i} u_j d_j$ and $c_{j-i} u_j d_i$ have symmetric p.d.f. because of $d_i$ and $d_j$. Thus, $y_i$ and $z_i$ have symmetric p.d.f. Now let us compute the covariance.

$$\begin{aligned}\mathrm{Cov}(y_i, y_{i'}) &= \sum_{j,j'=1}^n \mathrm{Cov}\big(c_{j-i} u_j d_j, c_{j'-i'} u_{j'} d_{j'}\big) && (2.22) \\ &= \sum_{j,j'=1}^n \mathbb{E}\big[c_{j-i} u_j d_j c_{j'-i'} u_{j'} d_{j'}\big] - \mathbb{E}[c_{j-i} u_j d_j] \mathbb{E}\big[c_{j'-i'} u_{j'} d_{j'}\big] && (2.23)\end{aligned}$$

Observe that $\mathbb{E}[c_{j-i} u_j d_j] = \mathbb{E}[c_{j-i} u_j] \mathbb{E}[d_j] = 0$ because $d_j$ is independent from $c_{j-i} u_j$. Also, observe that if $j \neq j'$ then $\mathbb{E}\big[d_j d_{j'}\big] = 0$ and thus $\mathbb{E}\big[c_{j-i} u_j d_j c_{j'-i'} u_{j'} d_{j'}\big] = \mathbb{E}\big[d_j d_{j'}\big] \mathbb{E}\big[c_{j-i} u_j c_{j'-i'} u_{j'}\big] = 0$. Thus, the only non null terms are those for which $j = j'$. We get:

$$\begin{aligned}\mathrm{Cov}(y_i, y_{i'}) &= \sum_{j=1}^n \mathbb{E}\big[c_{j-i} u_j d_j c_{j-i'} u_j d_j\big] \\ &= \sum_{j=1}^n \mathbb{E}\big[c_{j-i} c_{j-i'} u_j^2\big]\end{aligned}$$

Assume $u$ is a fixed vector. Then, $\mathrm{Var}(y_i) = \sum_{j=1}^{n} u_j^2 \sigma^2$ and $\mathrm{Cov}(y_i, y_{i'}) = 0$ for $i \neq i'$ because $c_{j-i}$ is independent from $c_{j-i'}$. Now assume that $u_j = \mathrm{ReLU}(x_j)$ where $x_j$ is a r.v. Clearly, $u_j^2$ is independent from $c_{j-i}$ and $c_{j-i'}$. Thus:

$$\mathrm{Cov}(y_i, y_{i'}) = \sum_{j=1}^{n} \mathbb{E}\left[c_{j-i} c_{j-i'}\right] \mathbb{E}\left[u_j^2\right]$$

For $i \neq i'$, then $c_{j-i}$ and $c_{j-i'}$ are independent, and thus $\mathbb{E}\left[c_{j-i} c_{j-i'}\right] = \mathbb{E}[c_{j-i}]\mathbb{E}\left[c_{j-i'}\right] = 0$. Therefore, $\mathrm{Cov}(y_i, y_{i'}) = 0$ if $i \neq i'$. Let us compute the variance. We get $\mathrm{Var}(y_i) = \sum_{j=1}^{n} \mathrm{Var}(c_{j-i}).\mathbb{E}\left[u_j^2\right]$. Because the p.d.f. of $x_j$ is symmetric, $\mathbb{E}\left[x_j^2\right] = 2\mathbb{E}\left[u_j^2\right]$ and $\mathbb{E}[x_j] = 0$. Thus, $\mathrm{Var}(y_i) = \frac{1}{2} \sum_{j=1}^{n} \mathrm{Var}(c_{j-i}).\mathbb{E}\left[x_j^2\right] = \frac{1}{2} \sum_{j=1}^{n} \mathrm{Var}(c_{j-i}).\mathrm{Var}(x_j)$.

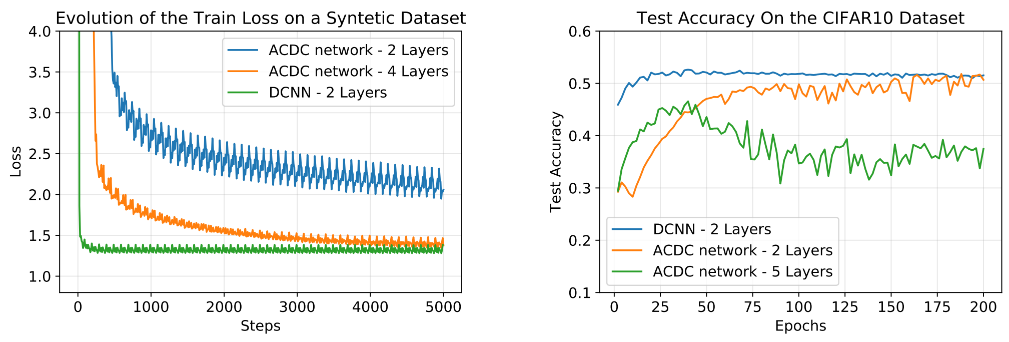
Finally, note that $\mathrm{Cov}(\bar{y}_i, \bar{y}_{i'}) = \mathrm{Cov}(y_i, y_{i'}) + \mathrm{Cov}(b_i, b_{i'})$. This yields the covariances of $\bar{y}$.

To derive $\mathrm{Cov}(z_i, z_{i'})$ and $\mathrm{Cov}(\bar{z}_i, \bar{z}_{i'})$, the required calculus is nearly identical. We let the reader check by himself/herself. ■

Non-linearity function We empirically found that reducing the number of non-linearities in the networks simplifies the training of deep neural networks. To support this claim, we conduct a series of experiments on various DCNNs with a varying number of ReLU activations (to reduce the number of non-linearities, we replace some ReLU activations with the identity function). In a second experiment, we replace the ReLU activations with Leaky-ReLU activations and vary the slope of the Leaky ReLU (a higher slope means an activation function that is closer to a linear function). The results of this experiment are presented in Figure 2.2a and 2.2b. In Figure 2.2a, "ReLU(DC)" means that we interleave ReLU activation functions between every diagonal-circulant matrix, whereas ReLU(DCDC) means we interleave a ReLU activation every other block etc. In both Figure 2.2a and Figure 2.2b, we observe that reducing the non-linearity of the networks can be used to train deeper networks. This is an interesting result, since we can use this technique to adjust the number of parameters in the network, without facing training difficulties. We obtain a maximum accuracy of 0.56 with one ReLU every three layers and leaky-ReLUs with a slope of 0.5. We hence rely on this setting in the experimental section.

## 2.6 Empirical evaluation

This experimental section aims at answering the following questions:



(a) Evolution of the training loss on a regression task with synthetic data.

(b) Test accuracy on the CIFAR-10 dataset.

Figure 2.3: Comparison of DCNNs and ACDC networks on two different tasks.

How do DCNNs compare to other approaches such as ACDC, LDR or other structured approaches?

How do DCNNs compare to other compression based techniques?

How do DCNNs perform in the context of large scale real-world machine learning applications?

### 2.6.1 Comparison with other structured approaches

Comparison with ACDC (Moczulski et al. 2015). In Section 2.2, we have discussed the differences between the ACDC framework and our approach from a theoretical perspective. In this section, we conduct experiments to compare the performance of DCNNs with neural networks based on ACDC layers. We first reproduce the experimental setting from Moczulski et al. (2015), and compare both approaches using only linear networks (*i.e.* networks without any ReLU activations). The synthetic dataset has been created in order to reproduce the experiment on the regression linear problem proposed by Moczulski et al. (2015). We draw $\mathbf{X}$ and $\mathbf{W}$ from a uniform distribution between [-1, +1] and $\epsilon$ from a normal distribution with mean 0 and variance 0.01. The relationship between $\mathbf{X}$ and $\mathbf{Y}$ is define by $\mathbf{Y} = \mathbf{X}\mathbf{W} + \epsilon$. The results are presented in Figure 2.3a. On this simple setting, while both architectures demonstrate good performance, we can observe that DCNNs offer a better convergence rate. In Figure 2.3b, we compare neural networks with ReLU activations on CIFAR-10.

We found that networks which are based only on ACDC layers are difficult to train and offer poor accuracy on CIFAR-10 (we have tried different initialization

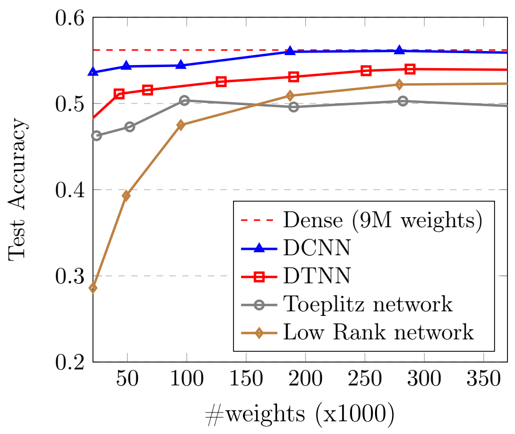


Figure 2.4: Network size vs. Accuracy compared on Dense networks, DCNNs (our approach), DTNNs (our approach), neural networks based on Toeplitz matrices and neural networks based on Low Rank-based matrices. DCNNs outperforms alternatives structured approaches.

schemes including the one from the original paper, and the one we introduce in this chapter). Moczulski et al. (2015) manage to train a large VGG network however these networks are generally highly redundant and the contribution of the structured layer is difficult to quantify. We also observe that adding a single dense layer improves the convergence rate of ACDC in the linear case, which explains the good results of Moczulski et al. (2015). However, it is difficult to characterize the true contribution of the ACDC layers when the network has a large number of expressive layers.

In contrast, deep DCNNs can be trained and offer good performance without additional dense layers (these results are in line with our experiments on the *YouTube-8M* dataset). We can conclude that DCNNs are able to model complex relations at a low cost.

**Comparison with Dense networks, Toeplitz networks and Low Rank networks.** We now compare DCNNs with other state-of-the-art structured networks by measuring the accuracy on a flattened version of the CIFAR-10 dataset. Our baseline is a dense feed-forward network with a fixed number of weights (9 million weights). We compare with DCNNs and with DTNNs (see below), Toeplitz networks, and Low-Rank networks (X. Yu et al. 2017). We first consider Toeplitz networks which are stacked Toeplitz matrices interleaved with ReLU activations since Toeplitz matrices are closely related to circulant matrices. However, Toeplitz networks have a different structure than DCNNs (they do not include diagonal matrices), therefore, we

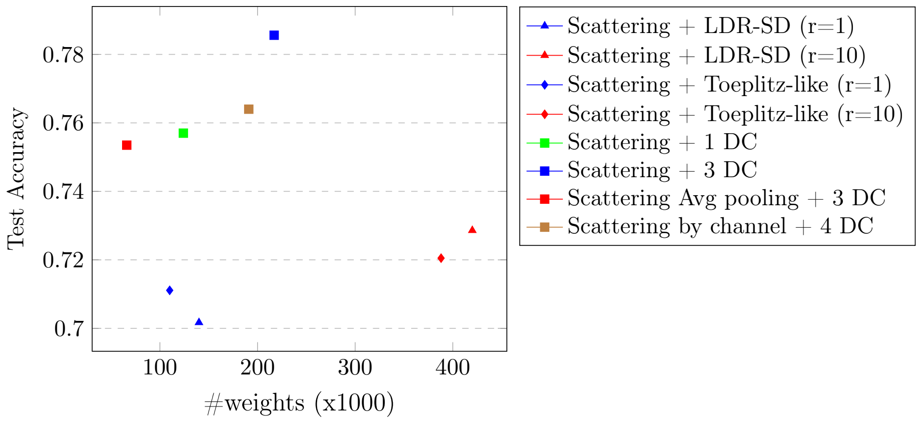


Figure 2.5: Accuracy of different structured architecture given the number of trainable parameters.

also experiment using DTNNs, a variant of DCNNs where all the circulant matrices have been replaced by Toeplitz matrices. Finally we conduct experiments using networks based on low-rank matrices as they are also closely related to our work. For each approach, we report the accuracy of several networks with a varying depth ranging from 1 to 40 (DCNNs, Toeplitz networks) and from 1 to 30 (from DTNNs). For low-rank networks, we used a fixed depth network and increased the rank of each matrix from 7 to 40. We also tried to increase the depth of low rank matrices, but we found that deep low-rank networks are difficult to train so we do not report the results here. We compare all the networks based on the number of weights from 21K (0.2% of the dense network) to 370K weights (4% of the dense network) and we report the results in Figure 2.4. First we can see that the size of the networks correlates positively with their accuracy which demonstrate successful training in all cases. We can also see that the DCNNs achieves the maximum accuracy of 56% with 20 layers ($\sim$ 200K weights) which is as good as the dense networks with only 2% of the number of weights. Other approaches also offer good trade-off but they are not able to reach the accuracy of a dense network.

Comparison with LDR networks (Thomas et al. 2018). We now compare DCNNs with the LDR framework using the network configuration experimented in the original paper: a single LDR structured layer followed by a dense layer. In the LDR framework, we can change the size of a network by adjusting the rank of the residual matrix, effectively capturing matrices with a structure that is close to a known structure but not exactly (in the LDR framework, Toeplitz matrices can be

| **Architectures** | **#Params** | **Acc.** |
|---|---|---|
| *Dense* | *9.4M* | *0.562* |
| ***DCNN*** (5 *layers*) | **49K** | **0.543** |
| ***DCNN*** (2 *layers*) | **21K** | **0.536** |
| LDR–TD ($r = 2$) | 64K | 0.511 |
| LDR–TD ($r = 3$) | 70K | 0.473 |
| Toeplitz-like ($r = 2$) | 46K | 0.483 |
| Toeplitz-like ($r = 3$) | 52K | 0.496 |

Table 2.1: LDR networks compared with DCNNs on a flattend version of CIFAR-10. DCNNs outperform all LDR configurations with fewer weights. Remark: the numbers may differ from the original experiments by Thomas et al. (2018) because we use the original dataset instead of a monochrome version.

| **Architectures** | **#Params** | **Acc.** |
|---|---|---|
| **DC** (1 *layers*) | **124K** | **0.757** |
| **DC** (3 *layers*) | **217K** | **0.785** |
| **Ensemble x5 DC** (3 *layers*) | **1.08M** | **0.811** |
| LDR-SD ($r = 1$) | 140K | 0.701 |
| LDR-SD ($r = 10$) | 420K | 0.728 |
| Toeplitz-like ($r = 1$) | 110K | 0.711 |
| Toeplitz-like ($r = 10$) | 388K | 0.720 |

Table 2.2: Two depths scattering on CIFAR-10 followed by LDR or DC layer. Networks with DC layers outperform all LDR configurations with fewer weights.

encoded with a residual matrix with rank=2, so a matrix that can be encoded with a residual of rank=3 can be seen as Toeplitz-like.). The results are presented in Table 2.1 and demonstrate that DCNNs outperforms all LDR networks both in terms in size and accuracy.

Exploiting image features. Dense layers and DCNNs are not designed to capture task-specific features such as the translation invariance inherently useful in image classification. We can further improve the accuracy of such general purpose architectures on image classification without dramatically increasing the number of trained parameters by stacking them on top of fixed (ie non-trained) transforms such as the scattering transform (Mallat, 2010). In this section we compare the accuracy of various structured networks, enhanced with the scattering transform, on an image classification task, and run comparative experiments on CIFAR-10.

Table 2.3: Comparison with compression based approaches

| **Architecture** | **#Params** | **Error (%)** |
|---|---|---|
| *LeNet (Lecun et al. 1998)* | *4 257 674* | *0.61* |
| **DCNN** | **25 620** | **1.74** |
| | **31 764** | **1.60** |
| HashNet (Chen et al. 2015) | 46 875 | 2.79 |
| | 78 125 | 1.99 |
| Dark Knowledge (Hinton et al. 2015) | 46 875 | 6.32 |
| | 78 125 | 2.16 |

Our test architecture consists of 2 depth scattering on the RGB images followed by a batch norm and LDR or DC layer. To vary the number of parameters of Scattering+LDR architecture, we increase the rank of the matrix (stacking several LDR matrices quickly exhausted the memory). The Figure 2.5 and 2.2 shows the accuracy of these architectures given the number of trainable parameters.

First, we can see that the DCNN architecture very much benefits from the scattering transform and is able to reach a competitive accuracy over 78%. We can also see that scattering followed by a DC layer systematically outperforms scattering + LDR or scattering + Toeplitz-like with less parameters.

### 2.6.2 Comparison with other compression based approaches

We provide a comparison with other compression based approaches such as HashNet (Chen et al. 2015), Dark Knowledge (Hinton et al. 2015) and Fast Food Transform (FF) (Yang et al. 2015). Table 2.3 shows the test error of DCNN against other know compression techniques on the MNIST datasets. We can observe that DCNN outperform easily HashNet (Chen et al. 2015) and Dark Knowledge (Hinton et al. 2015) with fewer number of parameters. The architecture with Fast Food (FF) (Yang et al. 2015) achieves better performance but with convolutional layers and only 1 Fast Food Layer as the last Softmax layer.

### 2.6.3 Large-scale video classification on the *YouTube-8M* dataset

To understand the performance of deep DCNNs on large scale applications, we conducted experiments on the *YouTube-8M* video classification with 3.8 training examples introduced by Abu El Haija et al. (2016). Notice that we favour this experiment over ImageNet applications because modern image classification architectures involve

Table 2.4: This table shows the GAP score for the *YouTube-8M* dataset with DCNNs. We can see a large increase in the score with deeper networks.

| **Architecture** | **#Weights** | **GAP@20** |
|---|---|---|
| *original* | *5.7M* | *0.773* |
| 4 DC | 25 410 (***0.44***) | 0.599 |
| 32 DC | 122 178 *(2.11)* | 0.685 |
| 4 DC + 1 FC | 4.46M *(77)* | **0.747** |

a large number of convolutional layers, and compressing convolutional layers is out of our scope. Also, as mentioned earlier, testing the performance of DCNN architectures mixed with a large number of expressive layers makes little sense. The *YouTube-8M* includes two datasets describing 8 million labeled videos. Both datasets contain audio and video features for each video. In the first dataset (*aggregated*) all audio and video features have been aggregated every 300 frames. The second dataset (*full*) contains the descriptors for all the frames. To compare the models we use the GAP metric (Global Average Precision) proposed by Abu El Haija et al. (2016). On the simpler *aggregated* dataset we compared off-the-shelf DCNNs with a dense baseline with 5.7M weights. On the full dataset, we designed three new compact architectures based on the state-of-the-art architecture introduced by Abu El Haija et al. (2016).

Experiments on the aggregated dataset with DCNNs: We compared DCNNs with a dense baseline with 5.7 millions weights. The goal of this experiment is to discover a good trade-off between depth and model accuracy. To compare the models we use the GAP metric (Global Average Precision) following the experimental protocol in (Abu El Haija et al. 2016), to compare our experiments.

Table 2.4 shows the results of our experiments on the *aggrgated YouTube-8M* dataset in terms of number of weights, compression rate and GAP. We can see that the compression ratio offered by the circulant architectures is high. This comes at the cost of a little decrease of GAP measure. The 32 layers DCNN is 46 times smaller than the original model in terms of number of parameters while having a close performance.

Experiments with DCNNs Deep Bag-of-Frames Architecture: The Deep Bag-of-Frames architecture can be decomposed into three blocks of layers, as illustrated in Figure 2.6. The first block of layers, composed of the Deep Bag-of-Frames embedding (DBoF), is meant to model an embedding of these frames in order to

Table 2.5: This table shows the GAP score for the *YouTube-8M* dataset with different layer represented with our DC decomposition.

| **Architecture** | **#Weights** | **GAP@20** |
|---|---|---|
| *original* | *45M* | *0.846* |
| DBoF with DC | 36M (*80*) | 0.838 |
| FC with DC | 41M (*91*) | **0.845** |
| MoE with DC | 12M (***26***) | 0.805 |

make a simple representation of each video. A second block of fully connected layers (FC) reduces the dimensionality of the output of the embedding and merges the resulting output with a concatenation operation. Finally, the classification block uses a combination of Mixtures-of-Experts (MoE) (Jordan & Jacobs, 1993; Abu-El-Haija et al. 2016) and Context Gating (Miech et al. 2017) to calculate the final class probabilities. Table 2.5 shows the results in terms of number of weights, size of the model (MB) and GAP on the full dataset, replacing the DBoF block reduces the size of the network without impacting the accuracy. We obtain the best compression ratio by replacing the MoE block with DCNNs (26%) of the size of the original dataset with a GAP score of 0.805 (95% of the score obtained with the original architecture). We conclude that DCNN are both theoretically sound and of practical interest in real, large scale applications.

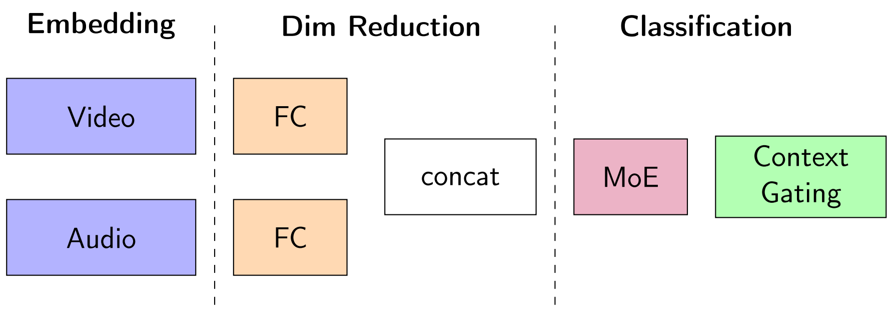


Figure 2.6: This figure shows the state-of-the-art neural network architecture, initially proposed by Abu El Haija et al. (2016) and later improved by Miech et al. (2017), used in our experiment.

Architectures & Hyper-Parameters: For the first set of our experiments (*experiments on CIFAR-10*), we train all networks for 200 epochs, a batch size of 200, Leaky ReLU activation with a different slope. We minimize the Cross Entropy

Loss with Adam optimizer and use a piecewise constant learning rate of $5 \times 10^{-5}$, $2.5 \times 10^{-5}$, $5 \times 10^{-6}$ and $1 \times 10^{-6}$ after respectively 40K, 60K and 80K steps. For the *YouTube-8M* dataset experiments, we built a neural network based on the SOTA architecture initially proposed by Abu El Haija et al. (2016) and later improved by Miech et al. (2017). Remark that no convolution layer is involved in this application since the input vectors are embeddings of video frames processed using state-of-the-art convolutional neural networks trained on ImageNet. We trained our models with the CrossEntropy loss and used Adam optimizer with a 0.0002 learning rate and a 0.8 exponential decay every 4 million examples. All fully connected layers are composed of 512 units. DBoF, NetVLAD and NetFV are respectively 8192, 64 and 64 of cluster size for video frames and 4096, 32, 32 for audio frames. We used 4 mixtures for the MoE Layer. We used all the available 300 frames for the DBoF embedding. In order to stabilize and accelerate the training, we used batch normalization before each non linear activation and gradient clipping.

## 2.7 Conclusion

This chapter deals with the training of diagonal circulant neural networks. To the best of our knowledge, training such networks with a large number of layers had not been done before. We also endowed this kind of models with theoretical guarantees, hence enriching and refining previous theoretical work from the literature. More importantly, we showed that DCNNs outperform their competing structured alternatives, including the very recent general approach based on LDR networks. Our results suggest that stacking diagonal circulant layers with non linearities improves the convergence rate and the final accuracy of the network. Formally proving these statements constitutes the future directions of this work. We would like to generalize the good results of DCNNs to convolutional neural networks. We also believe that circulant matrices deserve a particular attention in deep learning because of their strong ties with convolutions: a circulant matrix operator is equivalent to the convolution operator with circular paddings. This fact makes any contribution to the area of circulant matrices particularly relevant to the field of deep learning with impacts beyond the problem of designing compact models. As future work, we would like to generalize our results to deep convolutional neural networks.

# 3 Lipschitz Bound of Convolutional Neural Network

## Contents

## 3.1 Introduction

The last few years have witnessed a growing interest in Lipschitz regularization of neural networks, with the aim of improving their generalization (Bartlett et al. 2017), their robustness to adversarial attacks (Tsuzuku et al. 2018; Farnia et al. 2019), or their generation abilities (*e.g.* for GANs: (Miyato et al. 2018; Arjovsky et al. 2017)). Unfortunately computing the exact Lipschitz constant of a neural network is NP-hard (Virmaux & Scaman, 2018) and in practice, existing techniques such as (Virmaux & Scaman, 2018; Fazlyab et al. 2019) or (Latorre et al. 2020) are difficult to implement for neural networks with more than one or two layers, which hinders their use in deep learning applications.

To overcome this difficulty, most of the work has focused on computing the Lipschitz constant of *individual layers* instead. The product of the Lipschitz constant of each layer is an upper-bound for the Lipschitz constant of the entire network, and it can be used as a surrogate to perform Lipschitz regularization. Since most common activation functions (such as ReLU) have a Lipschitz constant equal to one, the main bottleneck is to compute the Lipschitz constant of the underlying linear application which is equal to its maximal singular value. The work in this line of research mainly relies on the celebrated iterative algorithm by Golub & Van der Vorst (2000) used to approximate the maximum singular value of a linear function. Although generic and accurate, this technique is also computationally expensive, which impedes its usage in large training settings.

In this paper we introduce a new tight and easy to compute upper bound on the largest singular value of convolution layers. We rely on Toeplitz matrix theory and its links with Fourier analysis. It is known that the sequence of elements that characterizes a Toeplitz matrix is the Fourier transform of a given trigonometric polynomial. We extend this result to doubly-block Toeplitz matrices (*i.e.* block Toeplitz matrices where each block is Toeplitz) and devise a bound on the singular values of these matrices as a function of a multivariate trigonometric polynomial. Implementing this bound in the context of convolutional neural networks, requires further to consider a sequence of doubly-block Toeplitz matrices, since the entries are commonly chosen to be tensors of more than one dimension. As a main result, we devise an upper bound for a stacked sequence of doubly-block Toeplitz matrices, which fits well the requirements of CNNs, both in terms of applicability and efficiency. From our analysis immediately follows an algorithm for bounding the Lipschitz constant of a convolutional layer, and by extension the Lipschitz constant of the whole network.

Finally, we illustrate our approach on adversarial robustness. Recent work has shown that empirical methods such as adversarial training offer poor generalization (Schmidt et al. 2018), and can be improved by applying Lipschitz regularization (Farnia et al. 2019). To illustrate the benefit of our new method, we train a large, state-of-the-art Wide ResNet architecture with Lipschitz regularization and show that it offers a significant improvement over adversarial training alone, and over other methods for Lipschitz regularization. To sum up, we make the following contributions:

1. We devise an upper bound on the singular values of the operator matrix of convolutional layers by leveraging Toeplitz matrix theory and its links with Fourier analysis.

2. We propose an efficient algorithm to compute this upper bound which enables its use in the context of convolutional neural networks.

3. We use our method to regularize the Lipschitz constant of neural networks in the area of adversarial robustness and show that it offers a significant improvement over adversarial training alone.

## 3.2 Related Work

A popular technique for approximating the maximal singular value of a matrix is the power method (Golub & Van der Vorst, 2000), an iterative algorithm which yields a good approximation of the maximum singular value when the algorithm is able to run for a sufficient number of iterations. Yoshida & Miyato (2017) and Miyato et al. (2018) have used the power method to normalize the spectral norm of each layer of a neural network, and showed that the resulting models offered improved generalization performance and generated better examples when they were used in the context of GANs. Farnia et al. (2019) built upon the work of Miyato et al. (2018) and proposed a power method specific for convolutional layers that leverages the deconvolution operation and avoid the computation of the gradient. They used it in combination with adversarial training. In the same vein, Gouk et al. (2018) demonstrated that regularized neural networks using the power method also offered improvements over their non-regularized counterparts. Furthermore, Tsuzuku et al. (2018) have shown that a neural network can be more robust to some adversarial attacks, if the prediction margin of the network (*i.e.* the difference between the first and the second maximum logit) is higher than a minimum threshold that depends on the global Lipschitz constant of the network. Building on this observation, they use the power method to compute an upper bound on the global Lipschitz constant, and maximize the prediction margin during training. Finally, Virmaux & Scaman (2018) have used automatic differentiation combined with the power method to compute a tighter bound on the global Lipschitz constant of neural networks. Despite a number of interesting results, using the power method is expensive and results in prohibitive training times.

Other approaches to regularize the Lipschitz constant of neural networks have been proposed by Sedghi et al. (2018) and Singla & Feizi (2019). The method of Sedghi et al. (2018) exploits the properties of circulant matrices to approximate the maximal singular value of a convolutional layer. Although interesting, this method results in a loose approximation of the maximal singular value of a convolutional layer. Furthermore, the complexity of their algorithm is dependent on the convolution input which can be high for large datasets such as ImageNet. More recently, Singla & Feizi (2019) have successfully bounded the operator norm of the Jacobian matrix of a convolution layer by the Frobenius norm of the reshaped kernel. This technique has the advantage to be very fast to compute and to be independent of the input size but it also results in a loose approximation.

To build robust neural networks, Cisse et al. (2017) and Q. Li et al. (2019) have proposed to constrain the Lipschitz constant of neural networks by using orthogonal convolutions. Cisse et al. (2017) use the concept of *parseval tight frames*, to constrain their networks. Q. Li et al. (2019) built upon the work of Cisse et al. (2017) to propose an efficient construction method of orthogonal convolutions.

Finally, recent work Fazlyab et al. (2019) and Latorre et al. (2020) has proposed a tight bound on the Lipschitz constant of the full network with the use of semi-definite programming. These works are theoretically interesting but lack scalability (*i.e.* the bound can only be computed on small networks).

## 3.3 A Primer on Toeplitz and block Toeplitz matrices

### 3.3.1 Convolution as Matrix Multiplication

A discrete convolution between a signal $\mathbf{x}$ and a kernel $\mathbf{k}$ can be expressed as a product between the vectorization of $\mathbf{x}$ and a doubly-block Toeplitz matrix $\mathbf{M}$, whose coefficients have been chosen to match the convolution $\mathbf{x} * \mathbf{k}$. For a 2-dimensional signal $\mathbf{x} \in \mathbb{R}^{n \times n}$ and a kernel $\mathbf{k} \in \mathbb{R}^{m \times m}$ with $m$ odd, the convolution operation can be written as follows:

$$\operatorname{vec}(\mathbf{y}) = \operatorname{vec}(\operatorname{pad}(\mathbf{x}) * \mathbf{k}) = \mathbf{M} \operatorname{vec}(\mathbf{x}) \tag{3.1}$$

where $\mathbf{M}$ is a $n^2$-by-$n^2$ doubly-block Toeplitz matrix, *i.e.* a block Toeplitz matrix where the blocks are also Toeplitz (Note that this is not a doubly-block circulant matrix because of the padding.), $\mathbf{y}$ is the output of size $q \times q$ with $q = n - m + 2p + 1$, (see *e.g.* (Dumoulin & Visin, 2016)). The vec : $\mathbb{R}^{n \times n} \to \mathbb{R}^{n^2}$ operator is defined as follows: $\operatorname{vec}(\mathbf{x})_q = \mathbf{x}_{\lfloor q/n \rfloor,\ q \mod n}$. The pad : $\mathbb{R}^{n \times n} \to \mathbb{R}^{(n+2p)\times(n+2p)}$ operator is a zero-padding operation which takes a signal $\mathbf{x}$ of shape $\mathbb{R}^{n \times n}$ and adds 0 on the edges so as to obtain a new signal $\mathbf{y}$ of shape $\mathbb{R}^{(n+2p)\times(n+2p)}$. In order to have the same shape between the convoluted signal and the signal, we set $p = \lfloor m/2 \rfloor$ [1].

We now present an example of the convolution operation with doubly-block Toeplitz matrix. Let us define a kernel $\mathbf{k} \in \mathbb{R}^{3\times 3}$ as follows:

$$\mathbf{k} = \begin{pmatrix} k_0 & k_1 & k_2 \\ k_3 & k_4 & k_5 \\ k_6 & k_7 & k_8 \end{pmatrix} \tag{3.2}$$

If we set the padding to 1, then, the matrix $\mathbf{M}$ is a tridiagonal doubly-block Toeplitz matrix of size $n \times n$ and has the following form:

$$\mathbf{M} = \begin{pmatrix} \mathbf{T}_0 & \mathbf{T}_1 & & & 0 \\ \mathbf{T}_2 & \mathbf{T}_0 & \mathbf{T}_1 & & \\ & \mathbf{T}_2 & \ddots & \ddots & \\ & & \ddots & \mathbf{T}_0 & \mathbf{T}_1 \\ 0 & & & \mathbf{T}_2 & \mathbf{T}_0 \end{pmatrix} \tag{3.3}$$

where $\mathbf{T}_j$ are banded Toeplitz matrices and the values of $\mathbf{k}$ are distributed in the Toeplitz blocks as follow:

$$\mathbf{T}_0 = \begin{pmatrix} k_4 & k_3 & & & 0 \\ k_5 & k_4 & k_3 & & \\ & k_5 & \ddots & \ddots & \\ & & \ddots & k_4 & k_3 \\ 0 & & & k_5 & k_4 \end{pmatrix} \quad \mathbf{T}_1 = \begin{pmatrix} k_7 & k_6 & & & 0 \\ k_8 & k_7 & k_6 & & \\ & k_8 & \ddots & \ddots & \\ & & \ddots & k_7 & k_6 \\ 0 & & & k_8 & k_7 \end{pmatrix} \quad \mathbf{T}_2 = \begin{pmatrix} k_1 & k_0 & & & 0 \\ k_2 & k_1 & k_0 & & \\ & k_2 & \ddots & \ddots & \\ & & \ddots & k_1 & k_0 \\ 0 & & & k_2 & k_1 \end{pmatrix} \tag{3.4}$$

Remark 1: Note that the size of the operator matrix $\mathbf{M}$ of a convolution operation depends on the size of the signal. If a signal $\mathbf{x}$ has size $n \times n$, the vectorized signal will be of size $n^2$ and the operator matrix will be of size $n^2 \times n^2$ which can be very large. Indeed, in deep learning practice the size of the images used for training can range

[1] We take a square signal and an odd size square kernel to simplify the notation but the same applies for any input and kernel size. Also, we take a specific padding in order to have the same size between the input and output signal. But everything in the paper can be generalized to any paddings.

from 32 (CIFAR-10) to hundred for high definition images (ImageNet). Therefore, with classical methods, computing the singular values of this operator matrix can be very expensive.

Remark 2: In the particular case of zero padding convolution operation, the operator matrix is a Toeplitz block with circulant block (i.e. each block of the Toeplitz block is a circulant matrix) which is a particular case of doubly-block Toeplitz matrices.

### 3.3.2 Generating a Toeplitz matrix and block Toeplitz matrix from a trigonometric polynomial

An $n \times n$ Toeplitz matrix $\mathbf{A}$ is fully determined by a two-sided sequence of scalars: $\{a_h\}_{h \in N}$, whereas an $nm \times nm$ block Toeplitz matrix $\mathbf{B}$ is fully determined by a two-sided sequence of blocks $\{\mathbf{B}_h\}_{h \in N}$ and where each block $\mathbf{B}_h$ is an $m \times m$ matrix.

The trigonometric polynomial that *generates* the Toeplitz matrix $\mathbf{A}$ can be defined as follows:

$$f_{\mathbf{A}}(\omega) \triangleq \sum_{h \in N} a_h e^{\mathbf{i} h \omega} \tag{3.5}$$

The function $f_{\mathbf{A}}$ is said to be the *generating function* of $\mathbf{A}$. To recover the Toeplitz matrix from its generating function, we have the following operator presented in Section 3.3.3 of the main paper:

$$(\mathbf{T}(f))_{i,j} \triangleq \frac{1}{2\pi} \int_0^{2\pi} e^{-\mathbf{i}(i-j)\omega} f(\omega)\, d\omega. \tag{3.6}$$

We can now show that $\mathbf{T}(f_{\mathbf{A}}) = \mathbf{A}$:

$$\begin{aligned}
(\mathbf{T}(f_{\mathbf{A}}))_{i,j} &= \frac{1}{2\pi} \int_0^{2\pi} e^{-\mathbf{i}(i-j)\omega} f_{\mathbf{A}}(\omega)\, d\omega && (3.7)\\
&= \frac{1}{2\pi} \int_0^{2\pi} e^{-\mathbf{i}(i-j)\omega} \sum_{h \in N} a_h e^{\mathbf{i} h \omega}\, d\omega && (3.8)\\
&= \frac{1}{2\pi} \int_0^{2\pi} \sum_{h \in N} a_h e^{\mathbf{i}(j-i+h)\omega}\, d\omega && (3.9)\\
&= \sum_{h \in N} a_h \frac{1}{2\pi} \int_0^{2\pi} e^{\mathbf{i}(j-i+h)\omega}\, d\omega = a_{j-i}. && (3.10)
\end{aligned}$$

Because:

$$\frac{1}{2\pi}\int_0^{2\pi} e^{\mathbf{i}k\omega}\, d\omega = \begin{cases} 1, & \text{if } k = 0, \\ 0, & \text{if } k \text{ is a non-zero integer number.} \end{cases} \tag{3.11}$$

The same reasoning can be applied to block Toeplitz matrices. Instead of being complex-valued, the trigonometric polynomial that *generates* the block Toeplitz $\mathbf{B}$ is matrix-valued and can be defined as follows:

$$f_{\mathbf{B}}(\omega) \triangleq \sum_{h\in N} \mathbf{B}_h e^{\mathbf{i}h\omega} \tag{3.12}$$

The function $f_{\mathbf{B}}$ is said to be the *generating function* of $\mathbf{B}$. To recover the block Toeplitz matrix from its generating function, we use the Toeplitz operator defined in Equation 3.6. We can show that $\mathbf{T}(f_{\mathbf{B}}) = \mathbf{B}$:

$$\begin{aligned}
(\mathbf{T}(f_{\mathbf{B}}))_{i,j} &= \frac{1}{2\pi}\int_0^{2\pi} e^{-\mathbf{i}(i-j)\omega} f_{\mathbf{B}}(\omega)\, d\omega && (3.13)\\
&= \frac{1}{2\pi}\int_0^{2\pi} e^{-\mathbf{i}(i-j)\omega} \sum_{h\in N} \mathbf{B}_h e^{\mathbf{i}h\omega}\, d\omega && (3.14)\\
&= \frac{1}{2\pi}\int_0^{2\pi} \sum_{h\in N} \mathbf{B}_h e^{\mathbf{i}(j-i+h)\omega}\, d\omega && (3.15)\\
&= \sum_{h\in N} \mathbf{B}_h \frac{1}{2\pi}\int_0^{2\pi} e^{\mathbf{i}(j-i+h)\omega}\, d\omega = \mathbf{B}_{j-i}. && (3.16)
\end{aligned}$$

In order to devise a bound on the Lipschitz constant of a convolution layer as used by the Deep Learning community, we study the properties of doubly-block Toeplitz matrices. In this section, we first introduce the necessary background on Toeplitz and block Toeplitz matrices, and introduce a new result on doubly-block Toeplitz matrices.

Toeplitz matrices and block Toeplitz matrices are well-known types of structured matrices. A Toeplitz matrix (respectively a block Toeplitz matrix) is a matrix in which each scalar (respectively block) is repeated identically along diagonals.

An $n \times n$ Toeplitz matrix $\mathbf{A}$ is fully determined by a two-sided sequence of scalars: $\{a_h\}_{h\in N}$, whereas an $nm \times nm$ block Toeplitz matrix $\mathbf{B}$ is fully determined by a two-sided sequence of blocks $\{\mathbf{B}_h\}_{h\in N}$, where $N = \{-n+1, \dots, n-1\}$ and where each block $\mathbf{B}_h$ is an $m \times m$ matrix.

$$\mathbf{A} = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} \\ a_{-1} & a_0 & \ddots & \vdots \\ \vdots & \ddots & a_0 & a_1 \\ a_{-n+1} & \cdots & a_{-1} & a_0 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_{n-1} \\ \mathbf{B}_{-1} & \mathbf{B}_0 & \ddots & \vdots \\ \vdots & \ddots & \mathbf{B}_0 & \mathbf{B}_1 \\ \mathbf{B}_{-n+1} & \cdots & \mathbf{B}_{-1} & \mathbf{B}_0 \end{pmatrix}.$$

Finally, a doubly-block Toeplitz matrix is a block Toeplitz matrix in which each block is itself a Toeplitz matrix. In the remainder, we will use the standard notation $(\,\cdot\,)_{i,j\in\{0,\ldots,n-1\}}$ to construct (block) matrices. For example, $\mathbf{A} = (a_{j-i})_{i,j\in\{0,\ldots,n-1\}}$ and $\mathbf{B} = (\mathbf{B}_{j-i})_{i,j\in\{0,\ldots,n-1\}}$.

### 3.3.3 Bound on the singular value of Toeplitz and block Toeplitz matrices

A standard tool for manipulating (block) Toeplitz matrices is the use of Fourier analysis. Let $\{a_h\}_{h\in N}$ be the sequence of coefficients of the Toeplitz matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and let $\{\mathbf{B}_h\}_{h\in N}$ be the sequence of $m \times m$ blocks of the block Toeplitz matrix $\mathbf{B}$. The complex-valued function $f(\omega) = \sum_{h\in N} a_h e^{\mathbf{i}h\omega}$ and the matrix-valued function $F(\omega) = \sum_{h\in N} \mathbf{B}_h e^{\mathbf{i}h\omega}$ are the *inverse Fourier transforms* of the sequences $\{a_h\}_{h\in N}$ and $\{\mathbf{B}_h\}_{h\in N}$, with $\omega \in \mathbb{R}$. From these two functions, one can recover these two sequences using the standard Fourier transform:

$$a_h = \frac{1}{2\pi}\int_0^{2\pi} e^{-\mathbf{i}h\omega} f(\omega) d\omega \qquad \mathbf{B}_h = \frac{1}{2\pi}\int_0^{2\pi} e^{-\mathbf{i}h\omega} F(\omega) d\omega. \tag{3.17}$$

From there, similarly to the work done by Gray (2006) and Gutiérrez Gutiérrez & Crespo (2012), we can define an operator $\mathbf{T}$ mapping integrable functions to matrices:

$$\mathbf{T}(g) \triangleq \left(\frac{1}{2\pi}\int_0^{2\pi} e^{-\mathbf{i}(i-j)\omega} g(\omega)\, d\omega\right)_{i,j\in\{0,\ldots,n-1\}}. \tag{3.18}$$

Note that if $f$ is the inverse Fourier transform of $\{a_h\}_{h\in N}$, then $\mathbf{T}(f)$ is equal to $\mathbf{A}$. Also, if $F$ is the inverse Fourier transform of $\{\mathbf{B}_h\}_{h\in N}$ as defined above, then the integral in Equation 3.18 is matrix-valued, and thus $\mathbf{T}(F) \in \mathbb{R}^{mn\times mn}$ is the block matrix $\mathbf{B}$.

Now, we can state two known theorems which upper bound the maximal singular value of Toeplitz and block Toeplitz matrices with respect to their generating functions. In the rest of the paper, we refer to $\sigma_1(\,\cdot\,)$ as the maximal singular value.

**Theorem 4** (Bound on the singular values of Toeplitz matrices)**.** *Let $f : \mathbb{R} \to \mathbb{C}$, be continuous and $2\pi$-periodic. Let $\mathbf{T}(f) \in \mathbb{R}^{n \times n}$ be a Toeplitz matrix generated by the function $f$, then:*

$$\sigma_1(\mathbf{T}(f)) \leq \sup_{\omega \in [0,2\pi]} |f(\omega)|. \tag{3.19}$$

Theorem 4 is a direct application of Lemma 4.1 by Gray (2006) for real Toeplitz matrices.

**Theorem 5** (Bound on the singular values of Block Toeplitz matrices Gutiérrez Gutiérrez & Crespo (2012))**.** *Let $F : \mathbb{R} \to \mathbb{C}^{m \times m}$ be a matrix-valued function which is continuous and $2\pi$-periodic. Let $\mathbf{T}(F) \in \mathbb{R}^{mn \times mn}$ be a block Toeplitz matrix generated by the function $F$, then:*

$$\sigma_1(\mathbf{T}(F)) \leq \sup_{\omega \in [0,2\pi]} \sigma_1(F(\omega)). \tag{3.20}$$

### 3.3.4 Bound on the singular value of Doubly-Block Toeplitz matrices

We extend the reasoning from Toeplitz and block Toeplitz matrices to doubly-block Toeplitz matrices (*i.e.* block Toeplitz matrices where each block is also a Toeplitz matrix). A doubly-block Toeplitz matrix can be generated by a function $f : \mathbb{R}^2 \to \mathbb{C}$ using the 2-dimensional inverse Fourier transform. For this purpose, we define an operator $\mathbf{D}$ which maps a function $f : \mathbb{R}^2 \to \mathbb{C}$ to a doubly-block Toeplitz matrix of size $nm \times nm$. For the sake of clarity, the dependence of $\mathbf{D}(f)$ on $m$ and $n$ is omitted. Let $\mathbf{D}(f) = (\mathbf{D}_{i,j}(f))_{i,j \in \{0,\ldots,n-1\}}$ where $\mathbf{D}_{i,j}$ is defined as:

$$\mathbf{D}_{i,j}(f) = \left( \frac{1}{4\pi^2} \int_{[0,2\pi]^2} e^{-\mathbf{i}((i-j)\omega_1 + (k-l)\omega_2)} f(\omega_1, \omega_2) \, d(\omega_1, \omega_2) \right)_{k,l \in \{0,\ldots,m-1\}} . \tag{3.21}$$

We are now able to combine Theorem 4 and Theorem 5 to bound the maximal singular value of doubly-block Toeplitz matrices with respect to their generating functions.

Note that in the following, we only consider generating functions as trigonometric polynomials with real coefficients therefore the matrices generated by $\mathbf{D}(f)$ are real. We can now combine Theorems 4 and 5 to bound the maximal singular value of a doubly-block Toeplitz Matrix.

**Theorem 6** (Bound on the maximal singular value of a Doubly-Block Toeplitz Matrix)**.** *Let* $\mathbf{D}(f) \in \mathbb{R}^{nm \times nm}$ *be a doubly-block Toeplitz matrix generated by the function* $f$*, then:*

$$\sigma_1(\mathbf{D}(f)) \leq \sup_{\omega_1, \omega_2 \in [0, 2\pi]^2} |f(\omega_1, \omega_2)| \tag{3.22}$$

*where the function* $f : \mathbb{R}^2 \to \mathbb{C}$*, is a multivariate trigonometric polynomial of the form:*

$$f(\omega_1, \omega_2) \triangleq \sum_{h_1 \in N} \sum_{h_2 \in M} d_{h_1, h_2} e^{\mathbf{i}(h_1 \omega_1 + h_2 \omega_2)}, \tag{3.23}$$

*where* $d_{h_1,h_2}$ *is the* ${h_2}^{th}$ *scalar of the* ${h_1}^{th}$ *block of the doubly-Toeplitz matrix* $\mathbf{D}(f)$*, and where* $M = \{-m+1, \ldots, m-1\}$*.*

***Proof of Theorem 6.*** A doubly-block Toeplitz matrix is by definition a block matrix where each block is a Toeplitz matrix. We can then express a doubly-block Toeplitz matrix with the operator $\mathbf{T}(F)$ where the matrix-valued generating function $F$ has Toeplitz coefficient. Let us define a matrix-valued trigonometric polynomial $F : \mathbb{R} \to \mathbb{C}^{n \times n}$ of the form:

$$F(\omega_1) = \sum_{h_1 \in N} \mathbf{A}_{h_1} e^{\mathbf{i} h_1 \omega_1} \tag{3.24}$$

where $\mathbf{A}_{h_1}$ are Toeplitz matrices of size $m \times m$ determined by the sequence $\{d_{h_1,-m+1}, \ldots, d_{h_1,m-1}\}$. From Theorem 5 we have:

$$\sigma_1(\mathbf{T}(F)) \leq \sup_{\omega_1 \in [0, 2\pi]} \sigma_1(F(\omega_1)) \tag{3.25}$$

Because Toeplitz matrices are closed under addition and scalar product, $F(\omega_1)$ is also a Toeplitz matrix of size $m \times m$. We can thus define a function $f : \mathbb{R}^2 \to \mathbb{C}$ such that $f(\omega_1, \cdot\,)$ is the generating function of $F(\omega_1)$. From Theorem 4, we can write:

$$\sigma_1(F(\omega_1)) \leq \sup_{\omega_2 \in [0, 2\pi]} |f(\omega_1, \omega_2)| \tag{3.26}$$

$$\Leftrightarrow \sup_{\omega_1 \in [0, 2\pi]} \sigma_1(F(\omega_1)) \leq \sup_{\omega_1, \omega_2 \in [0, 2\pi]^2} |f(\omega_1, \omega_2)| \tag{3.27}$$

$$\Leftrightarrow \sigma_1(\mathbf{T}(F)) \leq \sup_{\omega_1, \omega_2 \in [0, 2\pi]^2} |f(\omega_1, \omega_2)| \tag{3.28}$$

where the function $f$ is of the form:

$$f(\omega_1, \omega_2) = \sum_{h_1 \in N} \sum_{h_2 \in M} d_{h_1,h_2} e^{\mathbf{i}(h_1\omega_1 + h_2\omega_2)} \tag{3.29}$$

Because the function $f(\omega_1, \,\cdot\,)$ is the generating function of $F(\omega_1)$ is it easy to show that the function $f$ is the generating function of $\mathbf{T}(F)$. Therefore, $\mathbf{T}(F) = \mathbf{D}(f)$ which concludes the proof. ■

## 3.4 Bound on the Singular Values of Convolutional Layers

From now on, without loss of generality, we will assume that $n = m$ to simplify notations. It is well known that a discrete convolution operation with a 2d kernel applied on a 2d signal is equivalent to a matrix multiplication with a doubly-block Toeplitz matrix (Jain, 1989). However, in practice, the signal is most of the time 3-dimensional (RGB images for instance). We call the channels of a signal *channels in* denoted *cin*. The input signal is then of size $cin \times n \times n$. Furthermore, we perform multiple convolutions of the same signal which corresponds to the number of channels the output will have after the operation. We call the channels of the output *channels out* denoted *cout*. Therefore, the kernel, which must take into account *channels in* and *channels out*, is defined as a 4-dimensional tensor of size: $\text{cout} \times \text{cin} \times s \times s$.

The operation performed by a 4-dimensional kernel on a 3d signal can be expressed by the concatenation (horizontally and vertically) of doubly-block Toeplitz matrices. Hereafter, we bound the singular value of multiple vertically stacked doubly-block Toeplitz matrices which corresponds to the operation performed by a 3d kernel on a 3d signal.

**Theorem 7** (Bound on the maximal singular value of stacked Doubly-block Toeplitz matrices)**.** *Consider doubly-block Toeplitz matrices* $\mathbf{D}(f_1), \dots, \mathbf{D}(f_{\text{cin}})$ *where* $f_i : \mathbb{R}^2 \to \mathbb{C}$ *is a generating function. Construct a matrix* $\mathbf{M}$ *with* $\text{cin} \times n^2$ *rows and* $n^2$ *columns, as follows:*

$$\mathbf{M} \triangleq \left(\mathbf{D}^\top(f_1), \dots, \mathbf{D}^\top(f_{\text{cin}})\right)^\top. \tag{3.30}$$

*Then, with $f_i$ a multivariate polynomial of the same form as Equation 3.23, we have:*

$$\sigma_1(\mathbf{M}) \leq \sup_{\omega_1,\omega_2 \in [0,2\pi]^2} \sqrt{\sum_{i=1}^{\text{cin}} |f_i(\omega_1,\omega_2)|^2}. \tag{3.31}$$

In order to prove Theorem 7, we will need the following lemmas:

**Lemma 5** (Zhang (2011))**.** *Let $\mathbf{A}$ and $\mathbf{B}$ be Hermitian positive semi-definite matrices. If $\mathbf{A}-\mathbf{B}$ is positive semi-definite, then:*

$$\lambda_1(\mathbf{B}) \leq \lambda_1(\mathbf{A})$$

**Lemma 6** (Serra (1994))**.** *If the doubly-block Toeplitz matrix $\mathbf{D}(f)$ is generated by a non-negative function $f$ not identically zero, then the matrix $\mathbf{D}(f)$ is positive definite.*

**Lemma 7** (Serra (1994))**.** *If the doubly-block Toeplitz matrix $\mathbf{D}(f)$ is generated by a function $f : \mathbb{R}^2 \to \mathbb{R}$, then the matrix $\mathbf{D}(f)$ is Hermitian.*

**Lemma 8** (Gutiérrez Gutiérrez & Crespo (2012))**.** *Let $f : \mathbb{R}^2 \to \mathbb{C}$ and $g : \mathbb{R}^2 \to \mathbb{C}$ be two continuous and $2\pi$-periodic functions. Let $\mathbf{D}(f)$ and $\mathbf{D}(g)$ be doubly-block Toeplitz matrices generated by the function $f$ and $g$ respectively. Then:*

- $\mathbf{D}^\top(f) = \mathbf{D}(f^*)$
- $\mathbf{D}(f) + \mathbf{D}(g) = \mathbf{D}(f+g)$

Before proving Theorem 7, we generalize the famous Widom identity (Widom, 1976) that express the relation between Toeplitz and Hankel matrix to doubly-block Toeplitz and Hankel matrices. We will need to generalize the doubly-block Toeplitz operator presented in Section 3.3.4. From now on, without loss of generality, we will assume that $n = m$ to simplify notations.

Let $\mathbf{H}^{\alpha_p}(f) = \left(\mathbf{H}^{\alpha_p}_{i,j}(f)\right)_{i,j \in \{0 \dots n-1\}}$ where $\mathbf{H}^{\alpha_p}_{i,j}$ is defined as:

$$\mathbf{H}^{\alpha_p}_{i,j}(f) = \left(\frac{1}{4\pi^2}\int_{[0,2\pi]^2} e^{-\mathbf{i}\alpha_p(i,j,k,l,\omega_1,\omega_2)} f(\omega_1,\omega_2)\, d(\omega_1,\omega_2)\right)_{k,l \in \{0,\dots,n-1\}}. \tag{3.32}$$

Note that as with the operator $\mathbf{D}(f)$ we only consider generating functions as trigonometric polynomials with real coefficients therefore the matrices generated by $\mathbf{H}(f)$ are real.

We will use the following $\alpha$ functions:

$$\alpha_0(i,j,k,l,\omega_1,\omega_2) = (-j-i-1)\omega_1 + (k-l)\omega_2$$

$$\alpha_1(i,j,k,l,\omega_1,\omega_2) = (i-j)\omega_1 + (-l-k-1)\omega_2$$

$$\alpha_2(i,j,k,l,\omega_1,\omega_2) = (-j-i-1)\omega_1 + (-l-k-1)\omega_2$$

$$\alpha_3(i,j,k,l,\omega_1,\omega_2) = (-j-i+n)\omega_1 + (-l-k-1)\omega_2$$

As with the doubly-block Toeplitz operator $\mathbf{D}(f)$, the matrices generated by the operator $\mathbf{H}^{\alpha_p}$ are of size $n^2 \times n^2$.

We now present the generalization of the Widom identity for Doubly-Block Toeplitz matrices below:

**Lemma 9** (Generalization of Widom Identity)**.** *Let $f : \mathbb{R}^2 \to \mathbb{C}$ and $g : \mathbb{R}^2 \to \mathbb{C}$ be two continuous and $2\pi$-periodic functions. We can decompose the Doubly-Block Toeplitz matrix $\mathbf{D}(fg)$ as follows:*

$$\mathbf{D}(fg) = \mathbf{D}(f)\mathbf{D}(g) + \sum_{p=0}^{3} \mathbf{H}^{\alpha_p\top}(f^*)\mathbf{H}^{\alpha_p}(g) + \mathbf{Q}\left(\sum_{p=0}^{3} \mathbf{H}^{\alpha_p\top}(f)\mathbf{H}^{\alpha_p}(g^*)\right)\mathbf{Q}. \quad (3.33)$$

*where $\mathbf{Q}$ is the anti-identity matrix of size $n^2 \times n^2$.*

***Proof of Lemma 9.*** Let $(i,j)$ be matrix indexes such $(\,\cdot\,)_{i,j}$ correspond to the value at the $i^{\text{th}}$ row and $j^{\text{th}}$ column, let us define the following notation:

$$i_1 = \lfloor i/n \rfloor \qquad j_1 = \lfloor j/n \rfloor$$
$$i_2 = i \mod n \qquad j_2 = j \mod n$$

Let us define $\hat{f}$ as the 2 dimensional Fourier transform of the function $f$. We refer to $\hat{f}_{h_1,h_2}$ as the Fourier coefficient indexed by $(h_1,h_2)$ where $h_1$ correspond to the index of the block of the doubly-block Toeplitz and $h_2$ correspond to the index of the value inside the block. More precisely, we have

$$(\mathbf{D}(f))_{i,j} = \hat{f}_{(\lfloor j/n \rfloor - \lfloor i/n \rfloor),((j \mod n)-(i \mod n))} \quad (3.34)$$
$$(\mathbf{H}^{\alpha_0}(f))_{i,j} = \hat{f}_{(\lfloor j/n \rfloor + \lfloor i/n \rfloor + 1),((j \mod n)-(i \mod n))} \quad (3.35)$$
$$(\mathbf{H}^{\alpha_1}(f))_{i,j} = \hat{f}_{(\lfloor j/n \rfloor - \lfloor i/n \rfloor),((j \mod n)+(i \mod n)+1)} \quad (3.36)$$
$$(\mathbf{H}^{\alpha_2}(f))_{i,j} = \hat{f}_{(\lfloor j/n \rfloor - \lfloor i/n \rfloor),((j \mod n)-(i \mod n))} \quad (3.37)$$
$$(\mathbf{H}^{\alpha_3}(f))_{i,j} = \hat{f}_{(\lfloor j/n \rfloor + \lfloor i/n \rfloor + n),((j \mod n)+(i \mod n)+1)} \quad (3.38)$$

We simplify the notation of the expressions above as follow:

$$(\mathbf{D}(f))_{i,j} = \hat{f}_{(j_1-i_1),(j_2-i_2)} \tag{3.39}$$
$$(\mathbf{H}^{\alpha_0}(f))_{i,j} = \hat{f}_{(j_1+i_1+1),(j_2-i_2)} \tag{3.40}$$
$$(\mathbf{H}^{\alpha_1}(f))_{i,j} = \hat{f}_{(j_1-i_1),(j_2+i_2+1)} \tag{3.41}$$
$$(\mathbf{H}^{\alpha_2}(f))_{i,j} = \hat{f}_{(j_1-i_1),(j_2-i_2)} \tag{3.42}$$
$$(\mathbf{H}^{\alpha_3}(f))_{i,j} = \hat{f}_{(j_1+i_1+n),(j_2+i_2+1)} \tag{3.43}$$

The convolution theorem states that the Fourier transform of a product of two functions is the convolution of their Fourier coefficients. Therefore, one can observe that the entry $(i,j)$ of the matrix $\mathbf{D}(fg)$ can be express as follows:

$$(\mathbf{D}(fg))_{i,j} = \sum_{k_1=-2n+1}^{2n-1} \sum_{k_2=-2n+1}^{2n-1} \hat{f}_{(k_1-i_1),(k_2-i_2)} \hat{g}_{(j_1-k_1),(j_2-k_2)}.$$

By splitting the double sums and simplifying, we obtain:

$$\begin{aligned}(\mathbf{D}(fg))_{i,j} = \sum_{k_1,k_2\in P} \Big( & \hat{f}_{(k_1-i_1),(k_2-i_2)}\hat{g}_{(j_1-k_1),(j_2-k_2)} + \hat{f}_{(-k_1-i_1-1),(k_2-i_2)}\hat{g}_{(j_1+k_1+1),(j_2-k_2)} \\ & + \hat{f}_{(k_1-i_1),(-k_2-i_2-1)}\hat{g}_{(j_1-k_1),(j_2+k_2+1)} + \hat{f}_{(-k_1-i_1-1),(-k_2-i_2-1)}\hat{g}_{(j_1+k_1+1),(j_2+k_2+1)} \\ & + \hat{f}_{(k_1-i_1+n),(-k_2-i_2-1)}\hat{g}_{(j_1-k_1-n),(j_2+k_2+1)} + \hat{f}_{(k_1-i_1+n),(k_2-i_2)}\hat{g}_{(j_1-k_1-n),(j_2-k_2)} \\ & + \hat{f}_{(k_1-i_1),(k_2-i_2+n)}\hat{g}_{(j_1-k_1),(j_2-k_2-n)} + \hat{f}_{(k_1-i_1+n),(k_2-i_2+n)}\hat{g}_{(j_1-k_1-n),(j_2-k_2-n)} \\ & + \hat{f}_{(-k_1-i_1-1),(k_2-i_2+n)}\hat{g}_{(j_1+k_1+1),(j_2-k_2-n)}\Big) \end{aligned} \tag{3.44}$$

where $P = \{(k_1,k_2) \mid k_1,k_2 \in \mathbb{N} \cup 0, 0 \leq k_1 \leq n-1, 0 \leq k_2 \leq n-1\}$.

Furthermore, we can observe the following:

$$(\mathbf{D}(f)\mathbf{D}(g))_{i,j} = \sum_{k=0}^{n^2} (\mathbf{D}(f))_{i,k}(\mathbf{D}(g))_{k,j} = \sum_{k_1,k_2\in P} \hat{f}_{(k_1-i_1),(k_2-i_2)}\hat{g}_{(j_1-k_1),(j_2-k_2)}$$

$$\begin{aligned}\Big(\mathbf{H}^{\alpha_1\top}(f^*)\mathbf{H}^{\alpha_1}(g)\Big)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}^*_{(k_1+i_1+1),(i_2-k_2)}\hat{g}_{(j_1+k_1+1),(j_2-k_2)} \\ &= \sum_{k_1,k_2\in P} \hat{f}_{(-k_1-i_1-1),(k_2-i_2)}\hat{g}_{(j_1+k_1+1),(j_2-k_2)} \\ \Big(\mathbf{H}^{\alpha_2\top}(f^*)\mathbf{H}^{\alpha_2}(g)\Big)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}^*_{(i_1-k_1),(k_2+i_2+1)}\hat{g}_{(j_1-k_1),(j_2+k_2+1)}\end{aligned}$$

$$
\begin{aligned}
&= \sum_{k_1,k_2\in P} \hat{f}_{(k_1-i_1),(-k_2-i_2-1)}\hat{g}_{(j_1-k_1),(j_2+k_2+1)} \\
\left(\mathbf{H}^{\alpha_3\top}(f^*)\mathbf{H}^{\alpha_3}(g)\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}^*_{(k_1+i_1+1),(k_2+i_2+1)}\hat{g}_{(j_1+k_1+1),(k_2+j_2+1)} \\
&= \sum_{k_1,k_2\in P} \hat{f}_{(-k_1-i_1-1),(-k_2-i_2-1)}\hat{g}_{(j_1+k_1+1),(k_2+j_2+1)} \\
\left(\mathbf{H}^{\alpha_4\top}(f^*)\mathbf{H}^{\alpha_4}(g)\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}^*_{(i_1-k_1-n),(k_2+i_2+1)}\hat{g}_{(j_1-k_1-n),(j_2+k_2+1)} \\
&= \sum_{k_1,k_2\in P} \hat{f}_{(k_1-i_1+n),(-k_2-i_2-1)}\hat{g}_{(j_1-k_1-n),(j_2+k_2+1)}
\end{aligned}
$$

Let us define the matrix $\mathbf{Q}$ of size $n^2 \times n^2$ as the anti-identity matrix. We have the following:

$$
\begin{aligned}
\left(\mathbf{H}^{\alpha_1\top}(f)\mathbf{H}^{\alpha_1}(g^*)\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}_{(k_1+i_1+1),(i_2-k_2)}\hat{g}^*_{(j_1+k_1+1),(j_2-k_2)} \\
&= \sum_{k_1,k_2\in P} \hat{f}_{(k_1+i_1+1),(i_2-k_2)}\hat{g}_{(-j_1-k_1-1),(k_2-j_2)} \\
\Leftrightarrow \left(\mathbf{Q}\mathbf{H}^{\alpha_1\top}(f)\mathbf{H}^{\alpha_1}(g^*)\mathbf{Q}\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}_{(k_1-i_1+n),(k_2-i_2)}\hat{g}_{(j_1-k_1-n),(j_2-k_2)} \\
\left(\mathbf{H}^{\alpha_2\top}(f)\mathbf{H}^{\alpha_2}(g^*)\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}_{(i_1-k_1),(k_2+i_2+1)}\hat{g}^*_{(j_1-k_1),(j_2+k_2+1)} \\
&= \sum_{k_1,k_2\in P} \hat{f}_{(i_1-k_1),(k_2+i_2+1)}\hat{g}_{(k_1-j_1),(-j_2-k_2-1)} \\
\Leftrightarrow \left(\mathbf{Q}\mathbf{H}^{\alpha_2\top}(f)\mathbf{H}^{\alpha_2}(g^*)\mathbf{Q}\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}_{(k_1-i_1),(k_2-i_2+n)}\hat{g}_{(j_1-k_1),(j_2-k_2-n)} \\
\left(\mathbf{H}^{\alpha_3\top}(f)\mathbf{H}^{\alpha_3}(g^*)\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}_{(k_1+i_1+1),(k_2+i_2+1)}\hat{g}^*_{(j_1+k_1+1),(k_2+j_2+1)} \\
&= \sum_{k_1,k_2\in P} \hat{f}_{(k_1+i_1+1),(k_2+i_2+1)}\hat{g}_{(-j_1-k_1-1),(-k_2-j_2-1)} \\
\Leftrightarrow \left(\mathbf{Q}\mathbf{H}^{\alpha_3\top}(f)\mathbf{H}^{\alpha_3}(g^*)\mathbf{Q}\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}_{(k_1-i_1+n),(k_2-i_2+n)}\hat{g}_{(j_1-k_1-n),(-k_2+j_2-n)} \\
\left(\mathbf{H}^{\alpha_4\top}(f)\mathbf{H}^{\alpha_4}(g^*)\right)_{i,j} &= \sum_{k_1,k_2\in P} \hat{f}_{(-k_1+i_1-n),(k_2+i_2+1)}\hat{g}^*_{(j_1-k_1-n),(j_2+k_2+1)}
\end{aligned}
$$

$$= \sum_{k_1,k_2 \in P} \hat{f}_{(-k_1+i_1-n),(k_2+i_2+1)} \hat{g}_{(-j_1+k_1+n),(-j_2-k_2-1)}$$

$$\Leftrightarrow \Big(\mathbf{Q}\mathbf{H}^{\alpha_4 \top}(f)\mathbf{H}^{\alpha_4}(g^*)\mathbf{Q}\Big)_{i,j} = \sum_{k_1,k_2 \in P} \hat{f}_{(-k_1-i_1-1),(k_2-i_2+n)} \hat{g}_{(j_1+k_1+1),(j_2-k_2-n)}$$

Now, we can observe from Equation 3.44 that:

$$\mathbf{D}(fg) = \mathbf{D}(f)\mathbf{D}(g) + \sum_{p=0}^{3} \mathbf{H}^{\alpha_p \top}(f^*)\mathbf{H}^{\alpha_p}(g) + \mathbf{Q}\left(\sum_{p=0}^{3} \mathbf{H}^{\alpha_p \top}(f)\mathbf{H}^{\alpha_p}(g^*)\right)\mathbf{Q}. \tag{3.45}$$

which concludes the proof. ■

Now we can prove Theorem 7 which bounds the maximal singular value of vertically stacked doubly-block Toeplitz matrices with their generating functions.

***Proof of Theorem* 7.** First, let us observe the following:

$$\sigma_1^2(\mathbf{M}) = \lambda_1\Big(\mathbf{M}^\top \mathbf{M}\Big) = \lambda_1\left(\sum_{i=1}^{\text{cin}} \mathbf{D}^\top(f_i)\mathbf{D}(f_i)\right). \tag{3.46}$$

And the fact that:

$$\lambda_1\left(\sum_{i=1}^{\text{cin}} \mathbf{D}\big(|f_i|^2\big)\right) \overset{\text{by Lemma 8}}{=} \lambda_1\left(\mathbf{D}\left(\sum_{i=1}^{\text{cin}} |f_i|^2\right)\right) \tag{3.47}$$

$$\overset{\text{by Lemma 7}}{=} \sigma_1\left(\mathbf{D}\left(\sum_{i=1}^{\text{cin}} |f_i|^2\right)\right) \tag{3.48}$$

$$\overset{\text{by Theorem 6}}{\leq} \sup_{\omega_1,\omega_2 \in [0,2\pi]^2} \sum_{i=1}^{\text{cin}} |f_i(\omega_1,\omega_2)|^2. \tag{3.49}$$

To prove the Theorem, we simply need to verify the following inequality:

$$\lambda_1\left(\sum_{i=1}^{\text{cin}} \mathbf{D}^\top(f_i)\mathbf{D}(f_i)\right) \leq \lambda_1\left(\mathbf{D}\left(\sum_{i=1}^{\text{cin}} |f_i|^2\right)\right). \tag{3.50}$$

From the positive definiteness of the following matrix:

$$\mathbf{D}\left(\sum_{i=1}^{\text{cin}} |f_i|^2\right) - \sum_{i=1}^{\mathbf{i}} \mathbf{D}^\top(f_i)\mathbf{D}(f_i), \tag{3.51}$$

one can observe that the r.h.s is a real symmetric positive definite matrix by Lemma 6 and 7. Furthermore, the l.h.s is a sum of positive semi-definite matrices. Therefore, if the subtraction of the two is positive semi-definite, one could apply Lemma 5 to prove the inequality 3.50.

We know from Lemma 9 that

$$\mathbf{D}(fg) - \mathbf{D}(f)\mathbf{D}(g) = \sum_{p=0}^{3} \mathbf{H}^{\alpha_p \top}(f^*)\mathbf{H}^{\alpha_p}(g) + \mathbf{Q}\left(\sum_{p=0}^{3} \mathbf{H}^{\alpha_p \top}(f)\mathbf{H}^{\alpha_p}(g^*)\right)\mathbf{Q}. \tag{3.52}$$

By instantiating $f = f^*$, $g = f$ and with the use of Lemma 8, we obtain:

$$\mathbf{D}(f^*f) - \mathbf{D}(f^*)\mathbf{D}(f) = \mathbf{D}(|f|^2) - \mathbf{D}^\top(f)\mathbf{D}(f) \tag{3.53}$$

$$= \sum_{p=0}^{3} \mathbf{H}^{\alpha_p \top}(f)\mathbf{H}^{\alpha_p}(f) + \mathbf{Q}\left(\sum_{p=0}^{3} \mathbf{H}^{\alpha_p \top}(f^*)\mathbf{H}^{\alpha_p}(f^*)\right)\mathbf{Q}. \tag{3.54}$$

From Equation 3.54, we can see that the matrix $\mathbf{D}(|f|^2) - \mathbf{D}^\top(f)\mathbf{D}(f)$ is positive semi-definite because it can be decomposed into a sum of positive semi-definite matrices. Therefore, because positive semi-definiteness is closed under addition, we have:

$$\sum_{i=1}^{\text{cin}} \Big(\mathbf{D}\big(|f_i|^2\big) - \mathbf{D}^\top(f_i)\mathbf{D}(f_i)\Big) \geq 0 \tag{3.55}$$

By re-arranging and with the use Lemma 8, we obtain:

$$\sum_{i=1}^{\text{cin}} \big(\mathbf{D}\big(|f_i|^2\big)\big) - \sum_{i=1}^{\text{cin}} \Big(\mathbf{D}^\top(f_i)\mathbf{D}(f_i)\Big) \geq 0 \tag{3.56}$$

$$\mathbf{D}\left(\sum_{i=1}^{\text{cin}} |f_i|^2\right) - \sum_{i=1}^{\text{cin}} \Big(\mathbf{D}^\top(f_i)\mathbf{D}(f_i)\Big) \geq 0 \tag{3.57}$$

We can conclude that the inequality 3.50 is true and therefore by Lemma 5 we have:

$$\lambda_1\left(\sum_{i=1}^{\mathrm{cin}} \mathbf{D}^\top(f_i)\mathbf{D}(f_i)\right) \leq \lambda_1\left(\mathbf{D}\left(\sum_{i=1}^{\mathrm{cin}} |f_i|^2\right)\right) \tag{3.58}$$

$$\Leftrightarrow \sigma_1^2(\mathbf{M}) \leq \sup_{\omega_1,\omega_2\in[0,2\pi]^2} \sum_{i=1}^{\mathrm{cin}} |f_i(\omega_1,\omega_2)|^2 \tag{3.59}$$

$$\Leftrightarrow \sigma_1(\mathbf{M}) \leq \sup_{\omega_1,\omega_2\in[0,2\pi]^2} \sqrt{\sum_{i=1}^{\mathrm{cin}} |f_i(\omega_1,\omega_2)|^2} \tag{3.60}$$

which concludes the proof. ■

To have a bound on the full convolution operation, we extend Theorem 7 to take into account the number of output channels. The matrix of a full convolution operation is a block matrix where each block is a doubly-block Toeplitz matrices. Below, we present our main result:

**Theorem 8** (Bound on the maximal singular value on the convolution operation)**.** *Let us define doubly-block Toeplitz matrices* $\mathbf{D}(f_{11}),\ldots,\mathbf{D}(f_{\mathrm{cin}\times\mathrm{cout}})$ *where* $f_{ij}:\mathbb{R}^2\to\mathbb{C}$ *is a generating function. Construct a matrix* $\mathbf{M}$ *with* $\mathrm{cin}\times n^2$ *rows and* $\mathrm{cout}\times n^2$ *columns such as Then, with* $f_{ij}$ *a multivariate polynomial of the same form as Equation 3.23, we have:*

$$\sigma_1(\mathbf{M}) \leq \sqrt{\sum_{i=1}^{\mathrm{cout}} \sup_{\omega_1,\omega_2\in[0,2\pi]^2} \sum_{j=1}^{\mathrm{cin}} |f_{ij}(\omega_1,\omega_2)|^2}. \tag{3.61}$$

First, in order to prove Theorem 8, we will need the following lemma which bound the singular values of a matrix constructed from the concatenation of multiple matrix.

**Lemma 10** (Bound on the singular values of concatenation of matrices)**.** *Let us define matrices* $\mathbf{A}_1,\ldots,\mathbf{A}_p$ *with* $\mathbf{A}_i\in\mathbb{R}^{n\times n}$*. Let us construct the matrix* $\mathbf{M}\in\mathbb{R}^{n\times pn}$ *as follows:*

$$\mathbf{M} \triangleq (\mathbf{A}_1,\ldots,\mathbf{A}_p) \tag{3.62}$$

*where* $(\ \cdot\ )$ *define the concatenation operation. Then, we can bound the singular values of the matrix* $\mathbf{M}$ *as follows:*

$$\sigma_1(\mathbf{M}) \leq \sqrt{\sum_{i=1}^{p} \sigma_1(\mathbf{A}_i)^2} \tag{3.63}$$

***Proof of Lemma 10.***

$$\sigma_1(\mathbf{M})^2 = \lambda_1\left(\mathbf{M}\mathbf{M}^\top\right) \tag{3.64}$$

$$= \lambda_1\left(\sum_{i=1}^{p} \mathbf{A}_i\mathbf{A}_i^\top\right) \tag{3.65}$$

$$\leq \sum_{i=1}^{p} \lambda_1\left(\mathbf{A}_i\mathbf{A}_i^\top\right) \tag{3.66}$$

$$\leq \sum_{i=1}^{p} \sigma_1(\mathbf{A}_i)^2 \tag{3.67}$$

$$\Leftrightarrow \sigma_1(\mathbf{M}) \leq \sqrt{\sum_{i=1}^{p} \sigma_1(\mathbf{A}_i)^2} \tag{3.68}$$

which concludes the proof. ■

***Proof of Theorem 8.*** Let us define the matrix $\mathbf{M}_i$ as follows:

$$\mathbf{M}_i = \left(\mathbf{D}(f_{1,i})^\top, \dots, \mathbf{D}(f_{\text{cin},i})^\top\right)^\top. \tag{3.69}$$

We can express the matrix $\mathbf{M}$ as the concatenation of multiple $\mathbf{M}_i$ matrices:

$$\mathbf{M} = (\mathbf{M}_1, \dots, \mathbf{M}_{\text{cout}}) \tag{3.70}$$

Then, we can bound the singular values of the matrix $\mathbf{M}$ as follows:

$$\sigma_1(\mathbf{M}) \overset{\text{by Lemma 10}}{\leq} \sqrt{\sum_{i=1}^{\text{cout}} \sigma_1(\mathbf{M}_i)^2} \tag{3.71}$$

$$\sigma_1(\mathbf{M}) \overset{\text{by Theorem 7}}{\leq} \sqrt{\sum_{j=1}^{\text{cout}} \sup_{\omega_1,\omega_2\in[0,2\pi]^2} \sum_{i=1}^{\text{cin}} |f_{ij}(\omega_1,\omega_2)|^2} \tag{3.72}$$

which concludes the proof. ■

We can easily express the bound in Theorem 8 with the values of a 4-dimensional kernel. Let us define a kernel $\mathbf{k} \in \mathbb{R}^{\mathrm{cout} \times \mathrm{cin} \times s \times s}$, a padding $p \in \mathbb{N}$ and $d = \lfloor s/2 \rfloor$ the degree of the trigonometric polynomial, then:

$$f_{ij}(\omega_1, \omega_2) = \sum_{h_1=-d}^{d} \sum_{h_2=-d}^{d} k_{i,j,h_1,h_2} e^{\mathbf{i}(h_1\omega_1 + h_2\omega_2)}. \tag{3.73}$$

where $k_{i,j,h_1,h_2} = (\mathbf{k})_{i,j,a,b}$ with $a = s - p - 1 + i$ and $b = s - p - 1 + j$.

In the rest of the paper, we will refer to the bound in Theorem 8 applied to a kernel as LipBound and we denote LipBound($\mathbf{k}$) the Lipschitz upper bound of the convolution performed by the kernel $\mathbf{k}$.

## 3.5 Computation and Performance Analysis of LipBound

This section aims at analyzing the bound introduced in Theorem 8. First, we present an algorithm to efficiently compute the bound, we analyze its tightness by comparing it against the true maximal singular value. Finally, we compare the efficiency and the accuracy of our bound against the state-of-the-art.

### 3.5.1 Computing the maximum modulus of a trigonometric polynomial

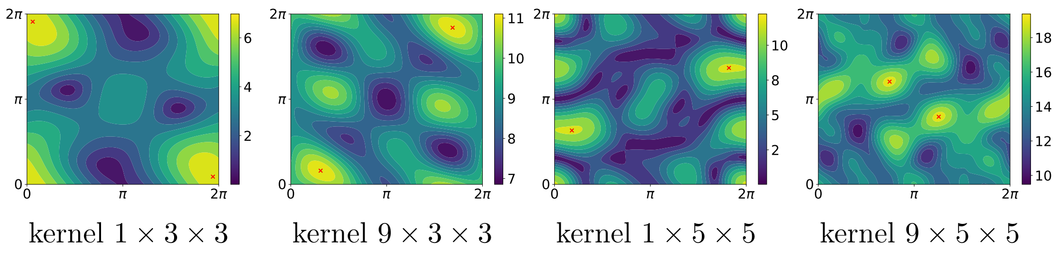


Figure 3.1: These figures represent the contour plot of multivariate trigonometric polynomials where the values of the coefficient are the values of a random convolutional kernel. The red dots in the figures represent the maximum modulus of the trigonometric polynomials.

In order to compute LipBound from Theorem 8, we have to compute the maximum modulus of several trigonometric polynomials. However, finding the maximum modulus of a trigonometric polynomial has been known to be NP-Hard (Pfister &

**Algorithm 1** PolyGrid Algorithm

```
1: procedure PolyGrid($f, S$)                ▷ polynomial $f$, number of samples $S$
2:     $\sigma \leftarrow 0$,
3:     $\omega_1 \leftarrow 0$
4:     $\omega_2 \leftarrow 0$
5:     $\epsilon \leftarrow \frac{2\pi}{S}$
6:     for $i = 0$ to $S - 1$ do
7:         for $j = 0$ to $S - 1$ do
8:             $\omega_2 \leftarrow \omega_2 + \epsilon$
9:             $\sigma \leftarrow \max(\sigma, f(\omega_1, \omega_2))$
10:        end for
11:    end for
12:    return $\sigma$                        ▷ approximated maximum modulus of $f$
13: end procedure
```

Bresler, 2018), and in practice they exhibit low convexity (see Figure 3.1). We found that for 2-dimensional kernels, a simple grid search algorithm such as PolyGrid (see Algorithm 1), works better than more sophisticated approximation algorithms (*e.g.* Green (1999) and De La Chevrotiere (2009)). This is because the complexity of the computation depends on the degree of the polynomial which is equal to $\lfloor s/2 \rfloor$ where $s$ is the size of the kernel and is usually small in most practical settings (*e.g.* $s = 3$). Furthermore, the grid search algorithm can be parallelized effectively on CPUs or GPUs and runs within less time than alternatives with lower asymptotic complexity.

To fix the number of samples $S$ in the grid search, we rely on the work of (Pfister & Bresler, 2018), who has analyzed the quality of the approximation depending on $S$. Following, this work we first define $\Theta_S$, the set of $S$ equidistant sampling points as follows:

$$\Theta_S \triangleq \left\{ \omega \mid \omega = k \cdot \frac{2\pi}{S} \text{ with } k = 0, \dots, S-1 \right\}. \tag{3.74}$$

Then, for a trigonometric polynomial $f : [0, 2\pi]^2 \to \mathbb{C}$, we have:

$$\max_{\omega_1, \omega_2 \in [0,2\pi]^2} |f(\omega_1, \omega_2)| \leq (1 - \alpha)^{-1} \max_{\omega_1', \omega_2' \in \Theta_S^2} \left| f(\omega_1', \omega_2') \right|, \tag{3.75}$$

where $d$ is the degree of the polynomial and $\alpha = 2d/S$. For a $3 \times 3$ kernel which gives a trigonometric polynomial of degree 1, we use $S = 10$ which gives $\alpha = 0.2$. Using this result, we can now compute LipBound for a convolution operator with *cout* output channels as per Theorem 7.

### 3.5.2 Analysis of the tightness of the bound

In this section, we study the tightness of the bound with respect to the dimensions of the doubly-block Toeplitz matrices. For each $n \in \mathbb{N}$, we define the matrix $\mathbf{M}^{(n)}$ of size $kn^2 \times n^2$ as follows:

$$\mathbf{M}^{(n)} \triangleq \left(\mathbf{D}^{(n)\top}(f_1), \dots, \mathbf{D}^{(n)\top}(f_k)\right)^\top \tag{3.76}$$

where the matrices $\mathbf{D}^{(n)}(f_i)$ are of size $n^2 \times n^2$. To analyze the tightness of the bound, we define the function $\Gamma$, which computes the difference between LipBound and the maximal singular value of the function $\mathbf{M}^{(n)}$:

$$\Gamma(n) = \text{LipBound}(\mathbf{k}_{\mathbf{M}^{(n)}}) - \sigma_1(\mathbf{M}^{(n)}) \tag{3.77}$$

where $\mathbf{k}_{\mathbf{M}^{(n)}}$ is the convolution kernel of the convolution defined by the matrix $\mathbf{M}^{(n)}$.

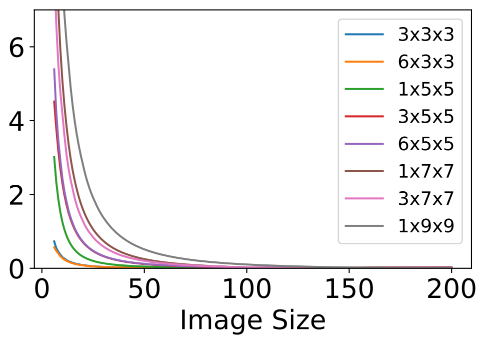


Figure 3.2: This graph represents the function $\Gamma(n)$ defined in Section 3.5.2 for different kernel size.

To compute the exact largest singular value of $\mathbf{M}^{(n)}$ for a specific $n$, we use the Implicitly Restarted Arnoldi Method (IRAM) (Lehoucq & Sorensen, 1996) available in SciPy. The results of this experiment are presented in Figure 3.2. We observe that the difference between the bound and the actual value (approximation gap) quickly decreases as the input size increases. For an input size of 50, the approximation gap is as low as 0.012 using a standard $6 \times 3 \times 3$ convolution kernel. For a larger input size such as ImageNet (224), the gap is lower than $4.10^{-4}$. Therefore LipBound gives an almost exact value of the maximal singular value of the operator matrix for most realistic settings.

### 3.5.3 Comparison of LipBound with other state-of-the-art approaches

Table 3.1: The following table compares different approaches for computing an approximation of the maximal singular value of a convolutional layer. It shows the ratio between the approximation and the true maximal singular value. The approximation is better for a ratio close to one.

| | **1x3x3** | | **32x3x3** | |
|---|---|---|---|---|
| | **Ratio** | **Time (ms)** | **Ratio** | **Time (ms)** |
| Sedghi et al. | $0.431 \pm 0.042$ | $1088 \pm 251$ | $0.666 \pm 0.123$ | $1729 \pm 399$ |
| Singla & Feizi | $1.293 \pm 0.126$ | $1.90 \pm 0.48$ | $1.441 \pm 0.188$ | $1.90 \pm 0.46$ |
| Farnia et al. (10 iter) | $0.973 \pm 0.006$ | $4.30 \pm 0.64$ | $0.972 \pm 0.004$ | $4.93 \pm 0.67$ |
| **LipBound (Ours)** | $\mathbf{0.992} \pm 0.012$ | $\mathbf{0.49} \pm 0.05$ | $\mathbf{0.984} \pm 0.021$ | $\mathbf{0.63} \pm 0.46$ |

In this section we compare our PolyGrid algorithm with the values obtained using alternative approaches. We consider the 3 alternative techniques by Sedghi et al. (2018), Singla & Feizi (2019) and Farnia et al. (2019) which have been described in Section 3.2.

To compare the different approaches, we extracted 20 kernels from a trained model. For each kernel we construct the corresponding doubly-block Toeplitz matrix and compute its largest singular value. Then, we compute the ratio between the approximation obtained with the approach in consideration and the exact singular value obtained by SVD, and average the ratios over the 20 kernels. Thus good approximations result in approximation ratios that are close to 1. The results of this experiment are presented in Table 3.1. The comparison has been made on a Tesla V100 GPU. The time was computed with the PyTorch CUDA profiler and we warmed up the GPU before starting the timer.

The method introduced by Sedghi et al. (2018) computes an approximation of the singular values of convolutional layers. We can see in Table 3.1 that the value is off by an important margin. This technique is also computationally expensive as it requires computing the SVD of $n^2$ small matrices where $n$ is the size of inputs. Singla & Feizi (2019) have shown that the singular value of the reshape kernel is a bound on the maximal singular value of the convolution layer. Their approach is very efficient but the approximation is loose and overestimate the real value. As said previously, the power method provides a good approximation at the expense of the efficiency. We use the special Convolutional Power Method from Farnia et al. (2019) with 10 iterations. The results show that our proposed technique: PolyGrid algorithm can

Table 3.2: This table shows the Accuracy under $\ell_2$ and $\ell_\infty$ attacks of CIFAR10 dataset. We compare vanilla Adversarial Training with the combination of Lipschitz regularization and Adversarial Training. We also compare the effectiveness of the power method by Farnia et al. (2019) and LipBound. The parameters $\lambda_2$ from Equation 3.78 is equal to 0.008 for AT+PM and AT+LipReg. It has been chosen from a grid search among 10 values. The attacks below are computed with 200 iterations.

| | **Accuracy** | **PGD-$\ell_\infty$** | **C&W-$\ell_2$ 0.6** | **C&W-$\ell_2$ 0.8** |
|---|---|---|---|---|
| **Baseline** | $\mathbf{0.953} \pm 0.001$ | $0.000 \pm 0.000$ | $0.002 \pm 0.000$ | $0.000 \pm 0.000$ |
| **AT** | $0.864 \pm 0.001$ | $0.426 \pm 0.000$ | $0.477 \pm 0.000$ | $0.334 \pm 0.000$ |
| **AT+PM** | $0.788 \pm 0.010$ | $0.434 \pm 0.007$ | $0.521 \pm 0.005$ | $0.419 \pm 0.003$ |
| **AT+LipReg** | $0.808 \pm 0.022$ | $\mathbf{0.457} \pm 0.002$ | $\mathbf{0.547} \pm 0.022$ | $\mathbf{0.438} \pm 0.020$ |

get the best of both worlds. It achieves a near perfect accuracy while being very efficient to compute.

We provide in the supplementary material a benchmark on the efficiency of LipBound on multiple convolutional architectures.

## 3.6 Application: Lipschitz Regularization for Adversarial Robustness

One promising application of Lipschitz regularization is in the area of adversarial robustness. Empirical techniques to improve robustness against adversarial examples such as Adversarial Training only impact the training data, and often show poor generalization capabilities (Schmidt et al. 2018). Farnia et al. (2019) have shown that the adversarial generalization error depends on the Lipschitz constant of the network, which suggests that the adversarial test error can be improved by applying Lipschitz regularization in addition to adversarial training.

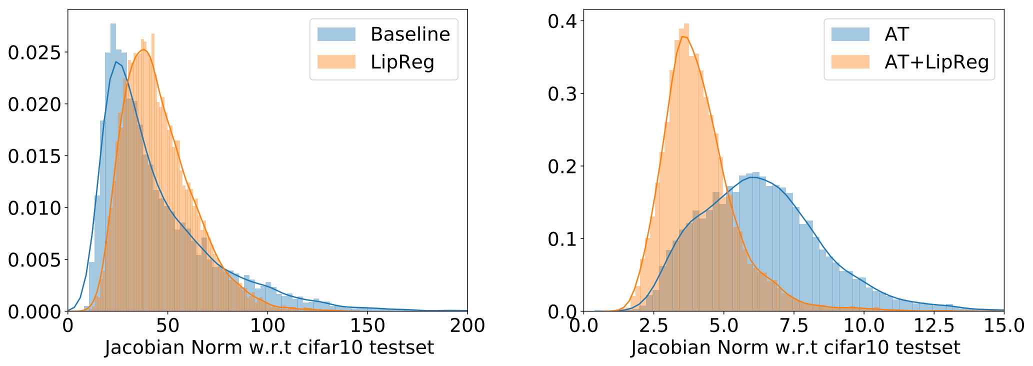
In this section, we illustrate the usefulness of LipBound by training a state-of-the-art Wide ResNet architecture (Zagoruyko & Komodakis, 2016) with Lipschitz regularization and adversarial training. Our regularization scheme is inspired by the one used by Yoshida & Miyato (2017) but instead of using the power method, we use our **PloyGrid** algorithm presented in Section 3.5.1 which efficiently computes an upper bound on the maximal singular value of convolutional layers.

We introduce the **AT+LipReg** loss to combine Adversarial Training and our Lipschitz regularization scheme in which layers with a large Lipschitz constant are penalized. We consider a neural network $\mathcal{N}_\theta : \mathcal{X} \to \mathcal{Y}$ with $\ell$ layers $\phi^{(1)}_{\theta_1}, \dots, \phi^{(\ell)}_{\theta_\ell}$

where $\theta^{(1)}, \ldots, \theta^{(\ell-1)}$ are the kernels of the first $\ell - 1$ convolutional layers and $\theta_\ell$ is the weight matrix of the last fully-connected layer $\phi^{(\ell)}_{\theta_\ell}$. Given a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, we can train the parameters $\theta$ of the network by minimizing the AT+LipReg loss as follows:

$$\min_{\theta} \mathbb{E}_{x,y\sim\mathcal{D}} \left[ \max_{\|\tau\|_\infty \leq \epsilon} \mathcal{L}(\mathcal{N}_\theta(x+\tau), y) + \lambda_1 \sum_{i=1}^{\ell} \|\theta_i\|_{\mathrm{F}} + \lambda_2 \sum_{i=1}^{\ell-1} \log(\mathrm{LipBound}(\theta_i)) \right] \tag{3.78}$$

where $\mathcal{L}$ is the cross-entropy loss function, and $\lambda_1$, $\lambda_2$ are two user-defined hyper-parameters. Note that regularizing the sum of logs is equivalent to regularizing the product of all the LipBound which is an upper bound on the global Lipschitz constant. In practice, we also include the upper bound on the Lipschitz of the batch normalization because we can compute it very efficiently (see C.4.1 of Tsuzuku et al. (2018)) but we omit the last fully connected layer.



(a) (b)

Figure 3.3: These figures show the distribution of the norm of the Jacobian matrix w.r.t the CIFAR10 test set from a Wide Resnet trained with different schemes. Although Lipschitz regularization is not a Jacobian regularization, we can observe a clear shift in the distribution. This suggests that our method does not only work layer-wise, but also at the level of the entire network.

In this section, we compare the robustness of Adversarial Training (Goodfellow et al. 2015; Madry et al. 2018) against the combination of Adversarial Training and Lipschitz regularization. To regularize the Lipschitz constant of the network, we use the objective function defined in Equation 3.78. We train Lipschitz regularized neural network with LipBound (Theorem 8) implemented with PolyGrid (Algorithm 1) (AT+LipBound) with $S = 10$ or with the specific power method for convolutions introduced by Farnia et al. (2019) with 10 iterations (AT+PM).

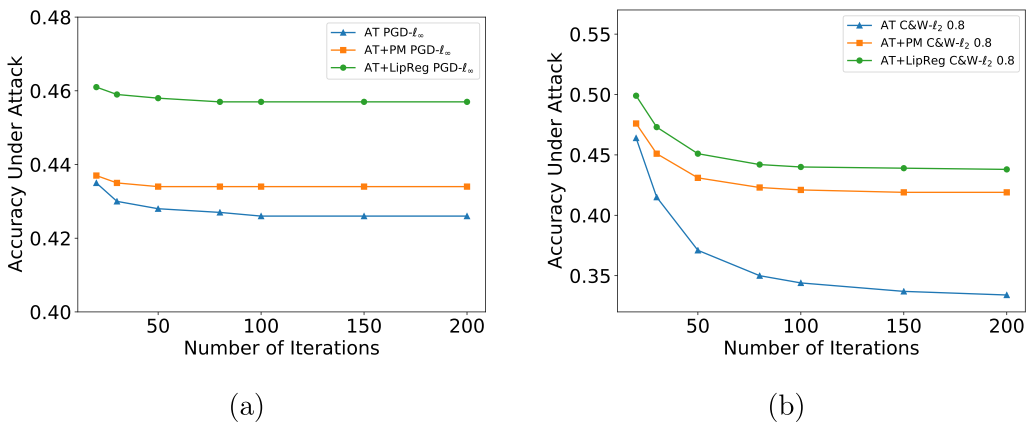


Figure 3.4: These figures show the Accuracy under attack on CIFAR10 test set with PGD-$\ell_\infty$ and C&W-$\ell_2$ attacks for several classifiers trained with Adversarial Training given the number of iterations.

Table 3.2 shows the gain in robustness against strong adversarial attacks. We can observe that both AT+LipBound and AT+PM offer a better defense against adversarial attacks and that AT+LipBound offers a further improvement over the Power Method. The Figure 3.4 (b) shows the Accuracy under attack with different number of iterations.

Finally, we also conducted an experiment to study the impact of the regularization on the gradients of the whole network by measuring the norm of the Jacobian matrix, averaged over the inputs from the test set. The results of this experiment are presented in Figure 3.3 (a) and show more concentrated gradients with Lipschitz regularization, which is the expected effect. This suggests that our method does not only work layer-wise, but also at the level of the entire network. A second experiment, using Adversarial Training, presented in Figure 3.3 (b) demonstrates that the effect is even stronger when the two techniques are combined together. This corroborates the work by (Farnia et al. 2019). It also demonstrates that Lipschitz regularization and Adversarial Training (or other Jacobian regularization techniques) are complementary. Hence they offer an increased robustness to adversarial attacks as demonstrated above.

Experimental Settings We use the Wide ResNet architecture introduced by Zagoruyko & Komodakis (2016) to train our classifiers (for details of the hyper-parameters used, see the supplementary material). For Adversarial Training (Madry et al. 2018), we use Projected Gradient Descent with an $\epsilon = 8/255(\approx 0.031)$, a step size of $\frac{\epsilon}{5}(\approx 0.0062)$

and 10 iterations, we use a random initialization but run the attack only once. To evaluate the robustness of our classifiers, we rigorously followed the experimental protocol proposed by Tramer et al. (2020) and Carlini, Athalye et al. (2019). More precisely, as an $\ell_\infty$ attack, we use PGD with the same parameters ($\epsilon = 8/255$, a step size of $\frac{\epsilon}{5}$) but we increase the number of iterations up to 200 with 10 restarts. For each image, we select the perturbation that maximizes the loss among all the iterations and the 10 restarts. As $\ell_2$ attacks, we use a bounded version of the (Carlini & Wagner, 2017) attack. We choose 0.6 and 0.8 as bounds for the $\ell_2$ perturbation. Note that the $\ell_2$ ball with a radius of 0.8 has approximately the same volume as the $\ell_\infty$ ball with a radius of 0.031 for the dimensionality of CIFAR10.

## 3.6.1 Results on CIFAR100 dataset & Hyper-parameters

Table 3.3: This table shows the Accuracy under $\ell_2$ and $\ell_\infty$ attacks of CIFAR100 dataset. We compare Adversarial Training with the combination of Lipschitz regularization and Adversarial Training (Madry et al. 2018). The $\lambda_2$ parameters which control the Lipschitz regularization is 0.008, it has been chosen among a grid search of 10 values. The attacks below are computed with 200 iterations.

| | **Accuracy** | **PGD-$\ell_\infty$** | **C&W-$\ell_2$ 0.6** | **C&W-$\ell_2$ 0.8** |
|---|---|---|---|---|
| **Baseline** | $\mathbf{0.792} \pm 0.000$ | $0.000 \pm 0.000$ | $0.001 \pm 0.000$ | $0.000 \pm 0.000$ |
| **AT** | $0.591 \pm 0.000$ | $0.199 \pm 0.000$ | $0.263 \pm 0.000$ | $0.183 \pm 0.000$ |
| **AT+LipReg** | $0.552 \pm 0.019$ | $\mathbf{0.215} \pm 0.004$ | $\mathbf{0.294} \pm 0.010$ | $\mathbf{0.226} \pm 0.008$ |

Hyper-parameters used for training classifiers on CIFAR10 & CIFAR100 dataset For all our experiments, we use the Wide Resnet architecture (Zagoruyko & Komodakis, 2016) with 28 layers and a width factor of 10. We train our networks for 200 epochs with a batch size of 200. We use Stochastic Gradient Descent with a momentum of 0.9, an initial learning rate of 0.1 with exponential decay of 0.1 (MultiStepLR gamma = 0.1) after the epochs 60, 120 and 160.

## 3.6.2 Results on ImageNet dataset & Hyper-parameters

Hyper-parameters used for training and attacking classifiers on ImageNet dataset For all our experiments, we use the Resnet-101 architecture (He et al. 2016b). We have used Stochastic Gradient Descent with a momentum of 0.9, a weight decay of 0.0001, label smoothing of 0.1, an initial learning rate of 0.1 with exponential decay of 0.1 (MultiStepLR gamma = 0.1) after the epochs 30 and 60. We

Table 3.4: This table shows the accuracy and accuracy under attack of ImageNet dataset with different training schemes. We compare Adversarial Training with the combination of Lipschitz regularization and Adversarial Training (Madry et al. 2018).

| | $\lambda_2$ | **Natural** | **PGD-$\ell_\infty$** | | **C&W-$\ell_2$** | | |
|---|---|---|---|---|---|---|---|
| | | | 0.02 | 0.031 | 1.00 | 2.00 | 3.00 |
| **AT** | - | 0.509 | 0.251 | 0.118 | 0.307 | 0.168 | 0.099 |
| **AT+LipReg** | 0.0006 | **0.515** | **0.255** | **0.121** | **0.316** | **0.177** | **0.105** |
| **AT+LipReg** | 0.0010 | **0.519** | **0.259** | **0.123** | **0.338** | **0.204** | **0.129** |

have used Exponential Moving Average over the weights with a decay of 0.999. We have trained our networks for 80 epochs with a batch size of 4096. For Adversarial Training, we have used PGD with 5 iterations, $\epsilon = 8/255 (\approx 0.031)$ and a step size of $\frac{\epsilon}{5} (\approx 0.0062)$.

To evaluate the robustness of our classifiers in ImageNet, we have used an $\ell_\infty$ and an $\ell_2$ attacks. More precisely, as an $\ell_\infty$ attack, we use PGD with an epsilon of 0.02 and 0.031, a step size of $\frac{\epsilon}{5}$) but we increase the number of iterations to 30 with 5 restarts. For each image, we select the perturbation that maximizes the loss among all the iterations and the 10 restarts. As $\ell_2$ attacks, we use a bounded version of the (Carlini & Wagner, 2017) attack. We have used 1, 2 and 3 as bounds for the $\ell_2$ perturbation.

## 3.7 Conclusion

In this chapter, we introduced a new bound on the Lipschitz constant of convolutional layers that is both accurate and efficient to compute. We used this bound to regularize the Lipschitz constant of neural networks and demonstrated its computational efficiency in training large neural networks with a regularized Lipschitz constant. As an illustrative example, we combined our bound with adversarial training, and showed that this increases the robustness of the trained networks to adversarial attacks. The scope of our results goes beyond this application and can be used in a wide variety of settings, for example, to stabilize Generative Adversarial Networks, or improve generalization capabilities of classifiers, to mention a few. Our future work will focus on investigating these fields.

# References

Abu El Haija, S., N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan (2016). "Youtube-8m: A large-scale video classification benchmark". *arXiv preprint arXiv:1609.08675* (see pages 27–30).

Abu-El-Haija, S., N. Kothari, J. Lee, A. ( Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan (2016). "YouTube-8M: A Large-Scale Video Classification Benchmark". In: *arXiv:1609.08675* (see page 29).

Araujo, A., B. Negrevergne, Y. Chevaleyre, and J. Atif (2018). "Training compact deep learning models for video classification using circulant matrices". In: *The 2nd Workshop on YouTube-8M Large-Scale Video Understanding at ECCV 2018* (see page 16).

Arjovsky, M., S. Chintala, and L. Bottou (2017). "Wasserstein gan". *arXiv preprint arXiv:1701.07875* (see page 32).

Arora, S., N. Cohen, W. Hu, and Y. Luo (2019). "Implicit Regularization in Deep Matrix Factorization". In: *Neurips* (see page 19).

Ba, J. and R. Caruana (2014). "Do deep nets really need to be deep?" In: *Advances in neural information processing systems*, pages 2654–2662 (see page 7).

Bartlett, P. L., D. J. Foster, and M. J. Telgarsky (2017). "Spectrally-normalized margin bounds for neural networks". In: *Advances in Neural Information Processing Systems*, pages 6240–6249 (see page 31).

Carlini, N., A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, and A. Madry (2019). "On Evaluating Adversarial Robustness". *arXiv preprint arXiv:1902.06705* (see page 57).

Carlini, N. and D. Wagner (2017). "Towards evaluating the robustness of neural networks". In: *2017 ieee symposium on security and privacy (sp)*. IEEE, pages 39–57 (see pages 57, 58).

Chen, W., J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen (2015). "Compressing Neural Networks with the Hashing Trick". In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. JMLR.org, Lille, France, pages 2285–2294 (see page 27).

Cheng, Y., F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S. F. Chang (2015). “An Exploration of Parameter Redundancy in Deep Networks with Circulant Projections”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2857–2865 (see pages 8, 9, 16).

Choromanski, K., M. Rowland, V. Sindhwani, R. E. Turner, and A. Weller (2018). “Structured Evolution with Compact Architectures for Scalable Policy Optimization”. In: *ICML* (see page 7).

Cisse, M., P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier (2017). “Parseval Networks: Improving Robustness to Adversarial Examples”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. JMLR.org, Sydney, NSW, Australia, pages 854–863 (see page 34).

De La Chevrotiere, G. (2009). “Finding the maximum modulus of a polynomial on the polydisk using a generalization of steckins lemma”. *SIAM Undergraduate Research Online* (see page 51).

Denil, M., B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas (2013). “Predicting Parameters in Deep Learning”. In: *Advances in Neural Information Processing Systems 26*. Editor C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., pages 2148–2156 (see page 10).

Dumoulin, V. and F. Visin (2016). “A guide to convolution arithmetic for deep learning”. *arXiv preprint arXiv:1603.07285* (see page 35).

Farnia, F., J. Zhang, and D. Tse (2019). “Generalizable Adversarial Training via Spectral Normalization”. In: *International Conference on Learning Representations* (see pages 31–33, 53–56).

Fazlyab, M., A. Robey, H. Hassani, M. Morari, and G. Pappas (2019). “Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks”. In: *Advances in Neural Information Processing Systems 32*. Editor H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pages 11423–11434 (see pages 32, 34).

Golub, G. H. and H. A. Van der Vorst (2000). “Eigenvalue computation in the 20th century”. *Journal of Computational and Applied Mathematics* 123:1-2, pages 35–65 (see pages 32, 33).

Goodfellow, I., J. Shlens, and C. Szegedy (2015). “Explaining and Harnessing Adversarial Examples”. In: *International Conference on Learning Representations* (see page 55).

Gouk, H., E. Frank, B. Pfahringer, and M. Cree (2018). “Regularisation of neural networks by enforcing lipschitz continuity”. *arXiv preprint arXiv:1804.04368* (see page 33).

Goyal, S., A. Roy Choudhury, and V. Sharma (2019). “Compression of Deep Neural Networks by Combining Pruning and Low Rank Decomposition”. In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 952–958 (see page 19).

Gray, R. (2006). “Toeplitz and circulant matrices: A review”. *Foundations and Trends® in Communications and Information Theory* 2:3, pages 155–239 (see pages 38, 39).

Green, J. (1999). “Calculating the maximum modulus of a polynomial using Steckin’s lemma”. *SIAM journal on numerical analysis* 36:4, pages 1022–1029 (see page 51).

Gutiérrez Gutiérrez, J. and P. Crespo (2012). “Block Toeplitz matrices: Asymptotic results and applications”. *Foundations and Trends® in Communications and Information Theory* 8:3, pages 179–257 (see pages 38, 39, 42).

Hanin, B. (2017). “Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations”. *ArXiv e-prints.* arXiv: `1708.02691 [stat.ML]` (see page 16).

He, K., X. Zhang, S. Ren, and J. Sun (2016a). “Deep Residual Learning for Image Recognition”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (see page 3).

– (2016b). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778 (see page 57).

Hinrichs, A. and J. Vybiral (2011). “Johnson-Lindenstrauss lemma for circulant matrices”. *Random Structures & Algorithms* 39:3, pages 391–398 (see pages 9, 11).

Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath et al. (2012). “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. *IEEE Signal processing magazine* 29:6, pages 82–97 (see page 3).

Hinton, G., O. Vinyals, and J. Dean (2015). “Distilling the Knowledge in a Neural Network”. In: *NIPS Deep Learning and Representation Learning Workshop* (see pages 10, 27).

Huhtanen, M. and A. Perämäki (2015). “Factoring Matrices into the Product of Circulant and Diagonal Matrices”. *Journal of Fourier Analysis and Applications* 21:5, pages 1018–1033. ISSN: 1531-5851 (see pages 8, 9, 11, 14).

Ivakhnenko, A. G. and V. G. Lapa (1967). “Cybernetics and forecasting techniques” (see page 2).

Jain, A. K. (1989). *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall (see page 41).

Jordan, M. I. and R. A. Jacobs (1993). “Hierarchical mixtures of experts and the EM algorithm”. In: *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*. Volume 2, 1339–1344 vol.2 (see page 29).

Konečný, J., H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon (2016). “Federated Learning: Strategies for Improving Communication Efficiency”. In: *NIPS Workshop on Private Multi-Party Machine Learning* (see page 7).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*, pages 1097–1105 (see page 3).

Latorre, F., P. Rolland, and V. Cevher (2020). “Lipschitz constant estimation for Neural Networks via sparse polynomial optimization”. In: *International Conference on Learning Representations* (see pages 32, 34).

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86:11, pages 2278–2324 (see pages 3–5).

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*, pages 2278–2324 (see page 27).

Lehoucq, R. B. and D. C. Sorensen (1996). “Deflation techniques for an implicitly restarted Arnoldi iteration”. *SIAM Journal on Matrix Analysis and Applications* 17:4, pages 789–821 (see page 52).

Li, C. and C. J. R. Shi (2018). “Constrained Optimization Based Low-Rank Approximation of Deep Neural Networks”. In: *Computer Vision – ECCV 2018*. Editor V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss. Springer International Publishing, Cham, pages 746–761 (see page 19).

Li, Q., S. Haque, C. Anil, J. Lucas, R. B. Grosse, and J.-H. Jacobsen (2019). “Preventing Gradient Attenuation in Lipschitz Constrained Convolutional Networks”. In: *Advances in Neural Information Processing Systems 32*. Editor H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pages 15364–15376 (see page 34).

Madry, A., A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu (2018). “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *International Conference on Learning Representations* (see pages 55–58).

Mallat, S. (2010). “Recursive interferometric representation”. In: *Proc. of EUSICO conference, Danemark* (see page 26).

Miech, A., I. Laptev, and J. Sivic (2017). “Learnable pooling with context gating for video classification”. *arXiv preprint arXiv:1706.06905* (see pages 29, 30).

Minsky, M. and S. Papert (1969). *Perceptrons.* MIT Press (see page 2).

Miyato, T., T. Kataoka, M. Koyama, and Y. Yoshida (2018). “Spectral normalization for generative adversarial networks”. *arXiv preprint arXiv:1802.05957* (see pages 32, 33).

Moczulski, M., M. Denil, J. Appleyard, and N. de Freitas (2015). “ACDC: A structured efficient linear layer”. *arXiv preprint arXiv:1511.05946* (see pages 8–10, 14, 23, 24).

Müller-Quade, J., H. Aagedal, T. Beth, and M. Schmid (1998). “Algorithmic design of diffractive optical systems for information processing”. *Physica D: Nonlinear Phenomena* 120:1-2, pages 196–205 (see page 8).

Novikov, A., D. Podoprikhin, A. Osokin, and D. P. Vetrov (2015). “Tensorizing neural networks”. In: *Advances in Neural Information Processing Systems*, pages 442–450 (see page 10).

Pfister, L. and Y. Bresler (2018). “Bounding multivariate trigonometric polynomials with applications to filter bank design”. *arXiv preprint arXiv:1802.09588* (see pages 50, 51).

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever (2018). *Language Models are Unsupervised Multitask Learners.* Technical report. OpenAi (see page 3).

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016). “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788 (see page 3).

Rosenblatt, F. (1958). “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review* 65:6, page 386 (see page 1).

Sanchez, V., P. Garcia, A. M. Peinado, J. C. Segura, and A. J. Rubio (1995). “Diagonalizing properties of the discrete cosine transforms”. *IEEE transactions on Signal Processing* 43:11, pages 2631–2641 (see page 9).

Schmid, M., R. Steinwandt, J. Müller-Quade, M. Rötteler, and T. Beth (2000). “Decomposing a matrix into circulant and diagonal factors”. *Linear Algebra and its Applications* 306:1-3, pages 131–143 (see page 11).

Schmidt, L., S. Santurkar, D. Tsipras, K. Talwar, and A. Madry (2018). “Adversarially robust generalization requires more data”. In: *Advances in Neural Information Processing Systems*, pages 5014–5026 (see pages 32, 54).

Sedghi, H., V. Gupta, and P. Long (2018). “The Singular Values of Convolutional Layers”. In: *ICLR* (see pages 19, 34, 53).

Serra, S. (1994). “Preconditioning strategies for asymptotically ill-conditioned block Toeplitz systems”. *BIT Numerical Mathematics* 34:4, pages 579–594 (see page 42).

Singla, S. and S. Feizi (2019). “Bounding Singular Values of Convolution Layers”. *arXiv preprint arXiv:1911.10258* (see pages 34, 53).

Tan, M. and Q. Le (2019). “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *International Conference on Machine Learning*, pages 6105–6114 (see page 3).

Tara Sainath, C. P. (2015). “Convolutional Neural Networks for Small-Footprint Keyword Spotting”. In: *Interspeech* (see page 7).

Thomas, A., A. Gu, T. Dao, A. Rudra, and C. Ré (2018). “Learning Compressed Transforms with Low Displacement Rank”. In: *Advances in Neural Information Processing Systems 31*. Editor S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pages 9066–9078 (see pages 10, 25, 26).

Trabelsi, C., O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal (2018). “Deep Complex Networks”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings* (see page 14).

Tramer, F., N. Carlini, W. Brendel, and A. Madry (2020). “On adaptive attacks to adversarial example defenses”. *arXiv preprint arXiv:2002.08347* (see page 57).

Tsuzuku, Y., I. Sato, and M. Sugiyama (2018). “Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks”. In: *Advances in Neural Information Processing Systems*, pages 6541–6550 (see pages 31, 33, 55).

Virmaux, A. and K. Scaman (2018). “Lipschitz regularity of deep neural networks: analysis and efficient estimation”. In: *Advances in Neural Information Processing Systems 31*. Editor S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pages 3835–3844 (see pages 32, 33).

Widom, H. (1976). “Asymptotic behavior of block Toeplitz matrices and determinants. II”. *Advances in Mathematics* 21:1, pages 1–29 (see page 42).

Yang, Z., M. Moczulski, M. Denil, N. d. Freitas, A. Smola, L. Song, and Z. Wang (2015). "Deep Fried Convnets". In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1476–1483 (see page 27).

Yoshida, Y. and T. Miyato (2017). "Spectral norm regularization for improving the generalizability of deep learning". *arXiv preprint arXiv:1705.10941* (see pages 33, 54).

Yu, X., T. Liu, X. Wang, and D. Tao (2017). "On Compressing Deep Models by Low Rank and Sparse Decomposition". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 67–76 (see page 24).

Zagoruyko, S. and N. Komodakis (2016). "Wide residual networks". *arXiv preprint arXiv:1605.07146* (see pages 54, 56, 57).

Zhang, F. (2011). *Matrix theory: basic results and techniques*. Springer Science & Business Media (see page 42).

Zhang, W., K. Itoh, J. Tanida, and Y. Ichioka (1990). "Parallel distributed processing model with local space-invariant interconnections and its optical architecture". *Applied optics* 29:32, pages 4790–4797 (see page 4).

Zhao, L., S. Liao, Y. Wang, Z. Li, J. Tang, and B. Yuan (2017). "Theoretical Properties for Neural Networks with Weight Matrices of Low Displacement Rank". In: *Proceedings of the 34th International Conference on Machine Learning*. Editor D. Precup and Y. W. Teh. Volume 70. Proceedings of Machine Learning Research. PMLR, International Convention Centre, Sydney, Australia, pages 4082–4090 (see page 14).