Using Toeplitz Matrices to obtain 2D convolution

Michał Gnacik (michal.gnacik@port.ac.uk)
University of Portsmouth
Krystian Łapa
Czestochowa University of Technology

Research Article

Keywords: Convolution, Toeplitz matrix, Convolutional Neural Networks, Fourier Transform, Deep Learning

Posted Date: October 27th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2195496/v1

Using Toeplitz Matrices to obtain 2D convolution

Michał Gnacik[1*] and Krystian Łapa[2]

[1*]School of Mathematics and Physics, University of Portsmouth, Lion Gate Building, Lion Terrace, Portsmouth, PO1 3HF, United Kingdom.

[2]Department of Computational Intelligence, Czestochowa University of Technology, Al. Armii Krajowej, Czestochowa, 42-200, Poland.

*Corresponding author(s). E-mail(s): m.gnacik@gmail.com; michal.gnacik@port.ac.uk;
Contributing authors: krystian.lapa@pcz.pl;

Abstract

Optimising the discrete convolution operations is important due to the fast growing interest and successful applications of deep learning to various fields and industries. In response to that, we propose an algorithm that views the 2D convolution operation between matrices as a matrix multiplication that involves a Toeplitz matrix; our algorithm is based on the mathematical result that we prove. We will present the complexity of the resulting algorithm and benchmark it against other 2D convolution algorithms in known Python computational libraries. The current implementations of our algorithm suggest a potential speed-up of convolution operation if the sparsity of the Toeplitz matrix was utilised.

Keywords: Convolution, Toeplitz matrix, Convolutional Neural Networks, Fourier Transform, Deep Learning

1 Introduction

Convolution operations are often employed in signal processing with sound or image inputs. They may be used, for example, to generate reverb in audio tracks [1] and blurring/smoothing images [2]. Convolving an image with a filter/kernel can enhance certain image properties, which may lead to finding a pattern within a collection of images that belong to the same category. Finding this pattern is a part of a process that convolutional neural networks (CNNs) do behind a black box of hidden layers. Due to a large number of data images that are required in this recognition process it is desired to improve the efficiency of the convolution operations in CNNs. We will dedicate the majority of this introduction to discuss CNNs and their applications. This will justify the importance of investigating various algorithms for the convolution operation. Then we will link the convolution operation with special objects of linear algebra such as Toeplitz matrices, and attempt to design efficient algorithms that yield the convolution.

Convolution for Artificial Neural Networks

To justify the importance of the convolution operation for artificial neural network, we will briefly discuss how these were developed and applied. The first example of a CNN was introduced in 1989 by Le Cun et al [3] for recognition of handwritten zip-codes. Despite further successful applications, e.g., in recognising handwritten digits [4], CNNs did not take off immediately in the field of computer vision, mainly due to lack of computational power and data [5]. In the last decade CNNs were extensively studied and successfully applied in image recognition (e.g., [6–10]), speech recognition (e.g., [11, 12]) and many other fields (e.g., [13–16]). Unlike feed-forward neural network (e.g., multilayer perceptrons [17]), CNNs do not require a flattened input (a vector), but allow the input to be a matrix (or more generally a tensor, meaning a multidimensional array). Typical CNNs architecture consists of convolutional layers, pooling layers (that perform down-sampling) and fully-connected layers (also known as dense layers) [18]. The input image of the CNN is processed via convolving it with each element of a collection of kernel/filter tensors (this may result in enhancing certain image properties, e.g., edge detection, sharpening, blurring). This process of convolving is performed via the convolution operation between the tensor (or matrix) objects (e.g., the input and kernels). For an overview of CNNs, their architecture and applications we refer the reader to the surveys [5, 18, 19].

Recall that during the training (or learning) process for neural networks the network weights are adjusted (thanks to the back-propagation by gradient-descent type methods) to minimise the cost function of the network (e.g., the categorical cross-entropy if the main task is a classification or the mean square error if the main task is a regression problem). It is well-known that neural networks are computationally hard to train [20, 21]. For CNNs this is caused,

[0] **Abbreviations:** CNN, convolutional neural networks

for example, by their complex and deep form (large number of hidden layers or blocks of layers) and often requires GPU implementation [6, 22] or optimised CPU architectures [23] in order to process large data sets of images. Reducing the time of training of neural networks by introducing new more efficient gradient-descent type algorithms [24, 25] or optimising their architecture (e.g., reformulation of layers via residual functions [26], approximating kernel tensors by using tensor rank decomposition [27]) has been a popular subject of research. It has been demonstrated that reducing computation and memory consumption can be achieved by using smaller kernels/filters (e.g., bottleneck in ResNet [26] or low rank decomposition [27–29]). However, large filters/kernels does also matter, for example, in semantic segmentation, large kernels play an important role for simultaneous classification and localisation tasks [30–32].

In recent years, the researchers started proposing to restructure the hidden layers in the form of Toeplitz ([33]) and Toeplitz-like matrices ([34, 35]), these include, for example, Toeplitz, Hankel, Vandermonde and Cauchy matrices (fast algorithms involving these matrices are presented in [36]) and their inverses. Using Toeplitz matrices is attractive due to its lower space complexity $\mathcal{O}(n)$ than in arbitrary matrices $\mathcal{O}(n^2)$, where n is the width and height of the corresponding matrix. It is also worth mentioning the enhancement of performance in a matrix-vector multiplication (algorithm of cost $\mathcal{O}(n \log n)$ [37], where n is the width of a Toeplitz matrix).

In this note we demonstrate that the convolution between matrices can be obtained as the matrix multiplication involving a Toeplitz matrix resulting from the kernel/filter matrix. It is rather simple to show that the convolution between two vectors can be presented as the multiplication of a Toeplitz matrix and a column vector (see, e.g., [38, Problem 2.4.1]). It is natural then to ask if this relation extends to matrix inputs. We answer this question affirmatively and explicitly construct the suitable Toeplitz matrix.

Outline

The paper is organised as follows, Section 2 contains a short overview of existing 2D convolution algorithms. It is followed by the main section of the paper in which we provide the construction of matrices (among which one is Toeplitz) that yields the 2d convolution as matrix multiplication; this mathematical result is proved. The final part of the main section provides an algorithm that arises from the construction and the mathematical result. Section four demonstrates how to embed the rectangular Toeplitz matrix into a square circulant matrix so that one may perform efficient Toeplitz matrix-matrix multiplication. This method is then applied to the matrices constructed in the previous section to yield the 2d convolution. After that the complexity of the proposed algorithm is discussed. In section five we present the benchmarks from our various implementations of the algorithm compared to the existing methods

for computing 2d convolution in Numpy, Scipy and Pytorch. The final section contains the conclusions and plans for the future work.

Notation and conventions

Throughout the paper we will use cross-correlation rather than convolution, however we will refer to it as convolution due to the terminology used in deep learning. In CNNs their inputs and building blocks are represented by tensors (multidimensional arrays). A typical input consists of N images of size $m \times n$ having h channels (i.e., RGB channels when $h = 3$ or gray-scale with $h = 1$) and are stored in an input tensor. The CNNs filters are also stored in a tensor, each filter has the matching number of channels which consist of a $p \times q$ filter matrices, where $p \leq m$ and $q \leq n$. Given the a-th image $X^{(a)}$ we convolve it with the b-th filter $K^{(b)}$ as follows

$$Y_{i,j}^{(a,b)} = \sum_{c=1}^{h} \sum_{k=1}^{p} \sum_{l=1}^{q} X_{c,i+k-1,j+l-1}^{(a)} K_{c,k,l}^{(b)}, \tag{1}$$

where $1 \leq i \leq m-p+1$ and $1 \leq j \leq n-q+1$. Therefore, a single convolution $X^{(a)} * K^{(b)}$ is stored in a $(m-p+1) \times (n-q+1)$ matrix $[Y_{i,j}^{(a,b)}]$. For simplicity, we ignored to pad the image X, this is often referred to as a 'valid' mode of the convolution (see e.g. [39]). There are also other modes of convolution such as 'same' and 'full'; 'full' requires padding of X prior of taking the convolution and 'same' extracts the submatrix from the center of 'full' convolution so that the output result preserves the original size of X. We have also set strides (the number that kernel moves on an image in each step along one direction) to 1. Often one wants to pad the image with zeros to dimension $(m+p-1)\times(n+q-1)$ so that result of the convolution preserves the dimension of the original image, that is, $m \times n$. The essence of the convolution operation is captured by the latter two sums in equation (1). Hence, at c-th channel the convolution is expressed as a $(m-p+1) \times (n-q+1)$ matrix $Y_c = X_c * K_c$ so that its elements are given by

$$Y_{c,i,j} = \sum_{k=1}^{p} \sum_{l=1}^{q} X_{c,i+k-1,j+l-1} K_{c,k,l}, \tag{2}$$

where X_i and K_i are $m \times n$ and $p \times q$ matrices, respectively. Note that given the image $X^{(a)} = (X_1, \ldots, X_h)$ and filter $K^{(b)} = (K_1, \ldots, K_h)$ we link (1) and (2) by

$$X^{(a)} * K^{(b)} = \sum_{c=1}^{k} X_c * K_c.$$

The matrices studied throughout the paper have real entries, unless stated otherwise, e.g., the entries of the discrete Fourier transform (DFT) matrices are complex numbers. The transpose of matrix A is denoted by A^T. Let $n \in \mathbb{N} := \{1, 2, 3, \ldots\}$, for simplicity we denote the modulo operation $a \mod n$

[0] **Abbreviations:** CNN, convolutional neural networks; RGB, red green blue

by $a\%n$; this notation is used in some programming languages (e.g., C, Java, Python). An $n \times n$ Toeplitz matrix $T = [t_{i,j}]_{i,j}$ is determined by a sequence of scalars $a_{-n+1}, \ldots, a_0, \ldots, a_{n-1}$ so that $t_{ij} = a_{j-i}$ for all $i, j \in \{1, \ldots, n\}$ [37]. This can be generalised to a not necessarily square $m \times n$ matrix by replacing the sequence with $a_{-m+1}, \ldots, a_0, \ldots, a_{n-1}$ and keeping the same relation $t_{ij} = a_{j-i}$ with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. The DFT matrix $\mathcal{F}$ is a $N \times N$ matrix given by $\mathcal{F} = \left[\frac{\omega^{(j-1)(k-1)}}{\sqrt{N}}\right]_{j,k=1,\ldots,N}$, where $\omega = e^{\frac{-2\pi\iota}{N}}$ is a primitive N-th root of unity and ι is the imaginary unit ($\iota^2 = -1$).

2 Overview of 2D convolution algorithms

In this section we reference and briefly discuss some well-known efficient algorithms to calculate 2D convolution. For an elaborate mathematical review of various convolution algorithms (particularly, focusing on Winograd's convolution) we refer the reader to [40]. For an nice exposition comparing different convolution algorithms when directly applied in CNNs we refer the reader to [41] .

First method that we already mentioned is using the formula (2) we refer to this as the *direct convolution*. For an $m \times n$ matrix X and $p \times q$ kernel K. The complexity of calculating this convolution is $\mathcal{O}(mnpq)$.

The next method arises from the Convolution Theorem and it is can be efficiently calculated by using the Fast Fourier Transform (FFT) [42]. Let $\mathcal{F}_2$ denote 2 dimensional discrete Fourier transform (DFT2), that is, for an $m \times n$ matrix X the entry in k-th row and l-th column of DFT2 is given by

$$\mathcal{F}_2\{X\}(k,l) = \sum_{s=1}^{m}\sum_{r=1}^{n} X_{k,l} e^{-2\pi\iota\left(\frac{k(s-1)}{m} + \frac{l(r-1)}{n}\right)}$$

for all $k \in \{0, \ldots, m-1\}$, $l \in \{0, \ldots, n-1\}$. The inverse transform is denoted by $\mathcal{F}_2^{-1}$ and entries are given by

$$\mathcal{F}_2^{-1}\{X\}(k,l) = \frac{1}{mn}\sum_{s=1}^{m}\sum_{r=1}^{n} X_{k,l} e^{2\pi\iota\left(\frac{k(s-1)}{m} + \frac{l(r-1)}{n}\right)}.$$

Let P be an anti-diagonal matrix with its anti-diagonal to be the vector of 1s (P is a permutation matrix). The convolution (cross-correlation) of a matrix X and kernel K can be obtained via the Convolution Theorem (see, e.g., [43, pages 33 and 72]) as a sub-matrix of

$$\mathcal{F}_2^{-1}\{\mathcal{F}_2\{\widehat{X}\} \odot \mathcal{F}_2\{\widehat{PKP}\}\}, \tag{3}$$

where $\widehat{X}$ and $\widehat{PKP}$ are obtained from X and the flipped K (by PKP) so that they are padded with 0s to have dimension $(n+p-1)\times(m+q-1)$, and $\odot$ denotes the point-wise (Hadamard) product of two matrices. The fast algorithm arises

from the application of Fast Fourier Transform (FFT) , when applying FFT the complexity of calculating (3) is $\mathcal{O}(nm\log((n+p-1)(m+q-1)))$. In particular, it is more efficient than direct convolution in networks with larger filters/kernels.

Another fast convolution algorithm is to rewrite the input and the kernel so that the result can be achieved via matrix multiplication [44]. This is often referred to as the unrolling of the convolution. In this method all the channels of the input matrix are combined so that the window of shape of the kernel from each channel of the input is flattened to form a new row. The new kernel matrix is obtained by flattening all kernel channels and stacking them into a column. Let us demonstrate this on an input image with one channel, that is, a matrix of size $m \times n$. To convolve this input with a single kernel $p \times q$ we will form two matrices. First matrix arises from unrolling the image and has the size $(m-p+1)(n-q+1) \times pq$ the reshaped kernel matrix (it is rather a vector) has size $pq \times 1$. The result contains all the entries of the 2d convolution that are stacked together in a column. The resulting matrix multiplication has complexity $\mathcal{O}((m-p+1)(n-q+1)pq) = \mathcal{O}(mnpq)$. Even though the algorithm complexity matches the complexity for the direct convolution, due to the existing optimised matrix multiplication libraries, for example, via Basic linear algebra subroutines (BLAS) [45], this yields a significant speedup. The benchmarks in [44] with and without BLAS on a CPU and separately on a GPU, show a speedups of $3\times$ on the CPU (with BLAS) and over $4\times$ on the GPU. None of the new matrices that are obtained in this algorithm belong to the class of special Toeplitz-like matrices, hence the multiplication of matrices can be computationally expensive given the dimensions of the resulting new matrices.

We also would like to make a note of an existing matrix-multiplication based algorithm in which one of the matrix is block Toeplitz with Toeplitz blocks. This means that the matrix is not necessarily Toeplitz itself, but its blocks are Toeplitz matrices, and block-wise it behaves like a Toeplitz matrix. One may think about a Toeplitz matrix as 1-block Toeplitz matrix, meaning that if you treat single entries as blocks that it yields a block Toeplitz matrix. The result is presented in [46, 4.1.2 Two-Dimensional Problems p. 37] for 3×3 matrices. Given an image X, the kernel K so that

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}, K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix},$$

their 'same' mode convolution

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

is a submatrix of $\widehat{X} * K$, where $\widehat{X}$ is obtained from X by (zero-)padding from top, bottom, left and right, and it can be obtained via

$$\begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \\ b_{21} \\ b_{22} \\ b_{32} \\ b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} = \left[\begin{array}{ccc|ccc|ccc} k_{22} & k_{32} & 0 & k_{23} & k_{33} & 0 & 0 & 0 & 0 \\ k_{12} & k_{22} & k_{32} & k_{13} & k_{23} & k_{33} & 0 & 0 & 0 \\ 0 & k_{12} & k_{22} & 0 & k_{13} & k_{23} & 0 & 0 & 0 \\ \hline k_{21} & k_{31} & 0 & k_{22} & k_{32} & 0 & k_{23} & k_{33} & 0 \\ k_{11} & k_{21} & k_{31} & k_{12} & k_{22} & k_{32} & k_{13} & k_{23} & k_{33} \\ 0 & k_{11} & k_{21} & 0 & k_{12} & k_{22} & 0 & k_{13} & k_{23} \\ \hline 0 & 0 & 0 & k_{21} & k_{31} & 0 & k_{22} & k_{32} & 0 \\ 0 & 0 & 0 & k_{11} & k_{21} & k_{31} & k_{12} & k_{22} & k_{32} \\ 0 & 0 & 0 & 0 & k_{11} & k_{21} & 0 & k_{12} & k_{22} \end{array}\right] \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ x_{12} \\ x_{22} \\ x_{32} \\ x_{13} \\ x_{23} \\ x_{33} \end{bmatrix}.$$

We found that this result lacks generality and it only considers the 'same' mode convolutions. Also the output of the algorithm requires further transformations to coincide with the dimensions of the convolution. Our proposed algorithm fills all these gaps.

The other algorithms for 2D convoltuion that are very efficient for small fixed-size kernels include variants of Winograd's convolution algorithm [40, 47, 48] based on the Winograd's minimal filtering method [49]. It is also worth to mention low-rank approximation [27, 29] methods allowing to approximate the convolution.

3 Construction, the main result and the algorithm

We demonstrate that $X * K$ can be obtained as matrix multiplication by a Toeplitz matrix, namely,

$$X * K = R_p(X^F) \cdot T_n(K), \tag{4}$$

where $T_n(K)$ is an $np \times (n - q + 1)$ Toeplitz matrix constructed from K and $R_p(X^F)$ is a $(m - p + 1) \times np$ matrix whose columns consists of certain rows of X that were flatten together. For simplicity, we drop the subscripts so that $R(X^T) \equiv R_p(X^F)$ and $T(K) \equiv T_n(K)$.

3.1 Construction

First we construct a (non-necessarily square) Toeplitz matrix $T(K)$ from K given the dimension n of X. To construct it is sufficient to provide only the first row and the first column of $T(K)$. The first row of $T(K)$ is obtained from a vector r so that

$$r = \begin{bmatrix} K_{1,1} & 0 & \dots & 0 \end{bmatrix}^T \in \mathbb{R}^{n-q+1}. \tag{5}$$

The first column is a vector of length $p \cdot n$ consisting of the following blocks of vectors

$$c = [\ B_1^T \quad B_2^T \quad \dots \quad B_p^T\ ]^T \in \mathbb{R}^{p \cdot n}, \tag{6}$$

so that the i-th block vector is

$$B_i = [\ K_{i,1} \quad K_{i,2} \quad \dots \quad K_{i,q} \quad \underbrace{0 \quad \dots \quad 0}_{n-q \text{ times}}\ ]^T \in \mathbb{R}^n.$$

Remark 1. Note that if $q < \frac{n}{2}$ then $T(K)$ is a sparse matrix, that is, most of its elements are zero.

Remark 2. As Toeplitz matrices are constructed by specifying the first row and column. If $T_{r,c}$ is a Toeplitz matrix with first row r and first column c then

$$(T_{r,c})^T = T_{c,r}. \tag{7}$$

Hence, our construction yields two Toeplitz matrices $T(K) \equiv T_{r,c}$ and $T(K)^T \equiv T_{c,r}$. To calculate the RHS of (4) more efficiently, re-write it to use the transpose of $T(K)$ instead, this will allow to use fast Toeplitz matrix-matrix multiplication (see section 4.1).

Now we construct the $R(X^F)$ matrix given X and the dimension p of K. First let us denote by X^F a vector that is obtained by flattening X, namely

$$X^F = [X^{(1)} \quad X^{(2)} \quad \dots \quad X^{(m)}]^T \in \mathbb{R}^{m \cdot n},$$

where $X^{(i)} = [X_{i,1} \quad \dots \quad X_{i,n}]$ for $i \in \{1, \dots, m\}$. The k-th row of $R(X^F)$ is

$$[(X^F)_{(k-1)n+1} \quad \dots \quad (X^F)_{(k-1+p)n}], \tag{8}$$

where $(X^F)_i$ is the i-th coordinate of X^F, for $k \in \{1, \dots, m-p+1\}$. Hence, $R(X^F)$ is an $(m-p+1) \times np$ matrix.

Remark 3. Note that only $T(K)$ is a Toeplitz matrix, $R(X^T)$ in general does not preserve the same values on the diagonals.

The following example will illustrate the above construction.

Example 1. Let

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} \quad \text{and} \quad K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}$$

for some real numbers x_{ij}, k_{rs}, where $i, j \in \{1, 2, 3, 4\}$, $r, s \in \{1, 2, 3\}$. Then

$$(T(K))^T = \begin{bmatrix} k_{11} & k_{12} & k_{13} & 0 & k_{21} & k_{22} & k_{23} & 0 & k_{31} & k_{32} & k_{33} & 0 \\ 0 & k_{11} & k_{12} & k_{13} & 0 & k_{21} & k_{22} & k_{23} & 0 & k_{31} & k_{32} & k_{33} \end{bmatrix},$$

$$R(X^F) = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{21} & x_{22} & x_{23} & x_{24} & x_{31} & x_{32} & x_{33} & x_{34} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{31} & x_{32} & x_{33} & x_{34} & x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}.$$

3.2 The main result

Now we prove the main result of this paper:

Main Theorem *It holds that*

$$X * K = R(X^F) \cdot T(K). \tag{9}$$

Proof First note that both matrices on the LHS and the RHS of equation (9) have the same dimensions, namely, $(m-p+1) \times (n-q+1)$. Let $Y = [Y_{i,j}]$ denote $X * K$. The definition of 2d convolution yields the form

$$Y_{i,j} = \sum_{k=1}^{p} \sum_{l=1}^{q} X_{i+k-1, j+l-1} K_{k,l}.$$

Before we expand the RHS of equation (9) note that $T(K)$ can be written as

$$T(K) = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & \cdots & a_{n-q} \\ a_{-1} & a_0 & a_1 & \ddots & & \vdots \\ a_{-2} & a_{-1} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_1 & a_2 \\ \vdots & & \ddots & a_{-1} & a_0 & a_1 \\ a_{-(np-1)} & \cdots & \cdots & a_{-2} & a_{-1} & a_0 \end{bmatrix}, \tag{10}$$

where $a_k = 0$ for $k \geq 1$ and and for $k \in \mathbb{Z} \cap [(l-1)n, ln-1]$ we have

$$a_{-k} = \begin{cases} K_{l,k\%n} & \text{if } k\%n < q, \\ 0 & \text{if } k\%n \geq q \end{cases} \tag{11}$$

for all $l \in \{1, \ldots, n\}$, where $\%n$ denotes the modulo operation yielding an element of $\mathbb{Z}_n$. Moreover, the element of $T(K)$ in the i-th row and j-th column is

$$(T(K))_{i,j} = a_{j-i}.$$

We also note that for $i \in [(l-1)n+1, ln]$, where $l = 1, \ldots, m$ the k-th coordinate of X^F is given by

$$(X^F)_i = X_{l,(i-1)\%n+1}.$$

Now denote by $C_{i,j}$ an element of $R(X^F) \cdot T(K)$ matrix in the i-th row and j-th column. We have

$$C_{i,j} = \sum_{k=1}^{np} (R(X^F))_{i,k} (T(K))_{k,j}$$

$$
\begin{aligned}
&= \sum_{k=1}^{np} (X^F)_{(i-1)n+k} a_{j-k} \\
&= \sum_{k=1}^{p} \sum_{l=(k-1)n+1}^{kn} (X^F)_{(i-1)n+l} a_{j-l} \\
&= \sum_{k=1}^{p} \sum_{l=(k-1)n+1}^{kn} X_{k+i-1,(l-1)\%n+1} a_{j-l} \\
&= \sum_{k=1}^{p} \sum_{l=1}^{n} X_{k+i-1,l} a_{j-(k-1)n-l}.
\end{aligned}
$$

Now we use the definition of a_k to obtain

$$a_{j-(k-1)n-l} = K_{k,(l-j+1)\%n}$$

whenever $(l-j)\%n < q$. Therefore,

$$
\begin{aligned}
C_{i,j} &= \sum_{k=1}^{p} \sum_{\substack{l\in\{1,\ldots,n\} \\ (l-j)\%n<q}} X_{k+i-1,l} K_{k,(l-j+1)\%n} \\
&= \sum_{k=1}^{p} \sum_{\substack{l\in\{-j+2,\ldots,n-j+1\} \\ (l-1)\%n<q}} X_{k+i-1,l+j-1} K_{k,l\%n}.
\end{aligned}
$$

Taking into account the condition $(l-1)\%n < q$ and that the index l runs over the set of n elements which are consecutive integers, we may drop the j dependence. Hence,

$$
\begin{aligned}
C_{i,j} &= \sum_{k=1}^{p} \sum_{\substack{l\in\{1,\ldots,n\} \\ (l-1)\%n<q}} X_{k+i-1,l+j-1} K_{k,l} \\
&= \sum_{k=1}^{p} \sum_{l=1}^{q} X_{i+k-1,j+l-1} K_{k,l} \\
&= Y_{i,j}
\end{aligned}
$$

□

Let us demonstrate the theorem on a particular example in the spirit of Example 1. Given

$$X = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 2 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 2 & 1 & 3 & 1 \end{bmatrix} \quad \text{and} \quad K = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

the construction yields the form

$$(T(K))^T = \begin{bmatrix} 1 & 2 & 1 & 0 & 2 & 4 & 2 & 0 & 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 & 2 & 4 & 2 & 0 & 1 & 2 & 1 \end{bmatrix},$$

$$R(X^F) = \begin{bmatrix} 1 & 2 & 1 & 1 & 2 & 1 & 1 & 1 & 0 & 1 & 2 & 3 \\ 2 & 1 & 1 & 1 & 0 & 1 & 2 & 3 & 2 & 1 & 3 & 1 \end{bmatrix}.$$

The convolution of matrices X and K is

$$X * K = \begin{bmatrix} 20 & 21 \\ 20 & 28 \end{bmatrix} = R(X^F) \cdot T(K).$$

Remark 4. As $T(K)$ is a Toeplitz matrix to employ a fast matrix-matrix multiplication it may be beneficial to obtain (9) as

$$X * K = (T(K)^T \cdot R(X^F)^T)^T. \tag{12}$$

This is further eleaborated in section 4.

3.3 Convolution Algorithm – pseudocode

The algorithm

INPUT:

1. matrix X (or an array or arrays), representing an input image.
2. matrix K (or an array of arrays), representing the filter/kernel.

OUTPUT:

- matrix Y (or an array of arrays), resulting by convolving X with K.

PROCEDURES:

1. From K construct the Toeplitz matrix $T(K)$ with row R and column C as follows
   a) row R consists of $n - q + 1$ elements which are all zeros apart from its first element which is $K[0, 0]$.
   b) First extend each row of K by adding $n - q$ zeros, then obtain C by flattening the new extended K.
2. From X construct the matrix $R(X^F)$ as follows
   (a) First flatten matrix X to obtain X^F.
   (b) Then the k-th row (for k from 0 to $m - p$) consists of sliced XF from index $k \cdot n$ to index $(k + p) \cdot n - 1$.
3. Obtain Y through matrix multiplication in either of the forms $R(X^F) \cdot T(K)$ or $(T(K)^T \cdot R(X^F)^T)^T$.

The diagram with the pseudocode is captured in Algorithm 1.

4 Toeplitz matrix-matrix mutlipication

This section introduces the 2D Convolution via Toeplitz and Fast Toeplitz matrix-matrix multiplication methods.

4.1 Embedding rectangular Toeplitz matrix into a square circulant matrix

To construct a square circulant matrix from a rectangular Toeplitz matrix we will use the methods described in [50]. Recall that a circulant $M \times M$ matrix (see [51]) is defined via a vector $c = (c_0, \ldots, c_{M-1})$ so that

$$C = \begin{bmatrix} c_0 & c_{M-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{M-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{M-2} & & \ddots & \ddots & c_{M-1} \\ c_{M-1} & c_{M-2} & \cdots & c_1 & c_0 \end{bmatrix}$$

In particular, it is a Toeplitz matrix. The advantage of circulant matrices is its eigendecomposition form [51, Theorem 3.2.2], that is,

$$C = \mathcal{F}^{-1}\Lambda\mathcal{F}, \tag{13}$$

where $\mathcal{F}$ is an $M \times M$ DFT matrix and $\Lambda = \mathrm{diag}(\mathcal{F}c)$ is a diagonal matrix that contains the eigenvalues of C.

Let T be an $m \times n$ Toeplitz matrix so that

$$T = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & \cdots & a_{n-1} \\ a_{-1} & a_0 & a_1 & \ddots & & \vdots \\ a_{-2} & a_{-1} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_1 & a_2 \\ \vdots & & \ddots & a_{-1} & a_0 & a_1 \\ a_{-m-1} & \cdots & \cdots & a_{-2} & a_{-1} & a_0 \end{bmatrix}. \tag{14}$$

First we construct the top-left of a $M \times M$ circulant matrix, where $M = m + n - 1$. The top-left of a circulant is given by $\begin{bmatrix} \widetilde{T} \\ T \end{bmatrix}$, where $\widetilde{T}$ is a $n-1 \times n$ Toeplitz matrix given by

$$\widetilde{T} = \begin{bmatrix} a_{n-1} & a_{-m+1} & a_{-m+2} & \cdots & \cdots & a_{-m+n-1} \\ a_{n-2} & a_{n-1} & a_{-m+1} & \ddots & & \vdots \\ a_{n-3} & a_{n-2} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-m+1} & a_{-m+2} \\ a_1 & & \ddots & a_{n-2} & a_{n-1} & a_{-m+1} \end{bmatrix}.$$

Algorithm 1 2D Convolution via Toeplitz

```
procedure CONVOLVE(X, K)
    p, q = Shape(K)                    ▷ extract shape (dimensions) from array K
    m, n = Shape(X)
    R = Zeros(n-q+1)                   ▷ Create an array of n-q+1 zeros
    R[0] = K[0, 0]                     ▷ Set the first element of row R to be K[0, 0]
    K = Pad(K, PadWidth=((0, 0), (0, n-q)))  ▷ Pad an array width with n-q zeros. PadWidth determines the number of zeros padded to the edges of each axis in format (before, after)
    C = Flatten(K)                     ▷ Flatten K array
    TK = Toeplitz(C, R)                ▷ Create a Toeplitz matrix from column C and row R
    XF = Flatten(X)                    ▷ Create X^F matrix
    for i=1 do m-p
        for j=k*n do (k+p)*n-1
            RX[i][j] = XF[j];          ▷ Create R(X^F) matrix by looping over XF
        end for
    end for
    Y = MatrixMultiplication(RX, TK)   ▷ Create convolution of X and K by matrix multiplication
    return Y
end procedure
```

The $M \times M$ circulant C whose top-left we constructed above is given by vector $c = (c_0, c_1, \ldots, c_{M-1})$ where

$$c_i = a_{n-1-i}, \tag{15}$$

for $i \in \{0, \ldots, M-1\}$. To efficiently multiply T by a vector $v = [v_1, v_2, \ldots, v_n] \in \mathbb{R}^n$ first denote $\widehat{v} = [v, 0 \ldots, 0] \in \mathbb{R}^M$. Then

$$Tv = JC\widehat{v}, \tag{16}$$

where $J = [J_{i,j}]$ is an $m \times M$ so that

$$J_{i,j} = \begin{cases} 1 & \text{if } i = j - n + 1 \\ 0 & \text{otherwise} \end{cases}.$$

Alternatively, one could present equation (16) directly with a DFT, by employing (13), that is,

$$Tv = J\mathcal{F}^{-1}\Lambda\mathcal{F}\widehat{v}. \tag{17}$$

One may view $J\mathcal{F}^{-1}$ as the slice of $\mathcal{F}^{-1}$ that contains its n-th to M-th rows.

4.2 2D convolution via Toeplitz-matrix-matrix multiplication

Consider an image (that for simplicity has only 1 colour channel) X of size $m \times n$ and a kernel K of size $p \times q$. To obtain the convolution $X * K$ via formula (12) we need $T_n(K)^T$ which is an $(n-q+1) \times np$ matrix and $\left(R_p(X^F)\right)^T$ which is $np \times (m-p+1)$, then we wish to perform

$$T_n(K)^T \cdot \left(R_p(X^F)\right)^T$$

before taking the transpose. Here, $T_n(K)^T$ is a Toeplitz matrix, hence it is generated by a sequence $a_{-(n-q+1)+1}, \ldots, a_0, \ldots, a_{np-1}$ so that the entries t_{ij} of $T_n(K)^T$ yield the relation $t_{ij} = a_{j-i}$ for all indices i, j. Let $M = n(p+1)-q$ following the technique from section 4.1, from $T_n(K)^T$ we construct an $M \times M$ circulant matrix C induced by the vector $c = (c_0, c_1, \ldots, c_M)$ so that $c_i = a_{n-1-i}$ for $i \in \{0, \ldots, M-1\}$. Note that multiplying $T_n(K)^T$ by $\left(R_p(X^F)\right)^T$ is equivalent to performing $m-p+1$ matrix-vector multiplications when viewing $\left(R_p(X^F)\right)^T$ as

$$[v_1 \ v_2 \ \ldots \ v_{m-p+1}],$$

where each $v_i \in \mathbb{R}^{np}$ represents the i-th column of $\left(R_p(X^F)\right)^T$ for each $i \in \{1, \ldots, m-p+1\}$.

Then it is enough to demonstrate how to efficiently calculate the matrix-vector multiplication. We observe that multiplying $T_n(K)^T$ by a vector $v \in \mathbb{R}^n$ is equivalent to

$$T_n(K)^T v = JC\widehat{v},$$

where $\widehat{v} = [v, 0 \ldots, 0] \in \mathbb{R}^M$ and $J = [J_{i,j}]$ is an $m \times M$ so that

$$J_{i,j} = \begin{cases} 1 & \text{if } i = j-n+1 \\ 0 & \text{otherwise} \end{cases}.$$

Furthermore, using the spectral decomposition of C we obtain that

$$T_n(K)^T v = J\mathcal{F}^{-1}\Lambda\mathcal{F}\widehat{v}, \tag{18}$$

where $\Lambda = \text{diag}\{\mathcal{F}c\}$. Hence, we will need to calculate three DFTs:

$$\mathcal{F}(c), \mathcal{F}(\widehat{v}) \text{ and } \mathcal{F}^{-1}(\Lambda\mathcal{F}(\widehat{v})). \tag{19}$$

4.3 Algorithm complexity of 2D convolution via Toeplitz

The complexity of the regular convolution between $m \times n$ matrix X and $p \times q$ filter K is of order $\mathcal{O}(mnpq)$. By analysing our algorithm we observe that the most expensive operation is the matrix multiplication followed by creating $R(X^F)$ matrix. Creating $R(X^F)$ matrix is of complexity $\mathcal{O}((m-p)np) =$

$\mathcal{O}(mnp)$. The original matrix multiplication algorithm would yield the cost $\mathcal{O}((m-p+1)np(n-q+1)) = \mathcal{O}(n^2mp)$ which is greater than for the original convolution. However, $p(n-q)(n-q+1)$ elements of $T(K)$ are zeros, and if $q < \frac{n}{2}$ then $T(K)$ is sparse, for speedups one may use the sparse matrix multiplication libraries such as SPBLAS (Sparse Basic Linear Algebra Subprograms) [52].

We exploit the fact that $T(K)$ is Toeplitz matrix, hence the algorithm complexity of the matrix multiplication can be reduced via applying Fast Fourier Transform methods. Calculating any of the Fourier Transforms in (19) is of complexity $\mathcal{O}(M\log(M)) = \mathcal{O}(np\log(n(p+1)-q))$. To apply it to matrix-matrix multiplication rather than matrix-vector multiplication we obtain the complexity $\mathcal{O}((m-p+1)M\log(M)) = \mathcal{O}(mnp\log(n(p+1)-q))$.

Algorithm 2 Fast Toeplitz matrix-matrix multiplication

procedure TOEPLITZ_MATMUL(col, row, B) ▷ col and row are 1D arrays, B is a 2D array
    B_row, B_col = Shape(B) ▷ extract shape (dimensions) from array B
    mm, nn = SIZE(row), SIZE(col) ▷ extract size (dimensions) from arrays row and col
    M = mm + nn - 1
    Bpad = PAD(B, PadWidth=((0, M-B_row), (0, 0))) ▷ Pad an array width with M-B_row zeros. PadWidth determines the number of zeros padded to the edges of each axis in format (before, after)
    row_col_vec = CONCATENATE(row[1:], col) ▷ Concatenating row vector with column vector, excluding the common first value
    indices = RANGE(M-1, -1, -1) ▷ Producing an array of inverted range from M-1 to 0
    c_vector = row_col_vec[indices] ▷ Constructing a vector to generate a circulant matrix according to (15)
    FFTB = FFT(Bpad, axis=0) ▷ Using Fast Fourier Transform (FFT) along the first axis on the input array
    FFTB = TRANSPOSE(FFTB) ▷ Transposing the array
    Lambda = FFT(c_vector) ▷ Using FFT on the input vector
    LambdaFB = PRODUCT(FFTB, Lambda) ▷ Producing a point-wise product between two arrays
    MultMat = IFFT(LambdaFB, axis=1) ▷ Using Inverse FFT along the second axis on the input array
    return TRANSPOSE(MultMat)
end procedure

5 Benchmarks

We compare the performance of our new suggested algorithm with the current implementations of 2D convolution in known Python libraries such as Numpy, Scipy and PyTorch. Our benchmarks will be performed with and without GPU support. We generate randomly a hundred 100×100 grey-scale images (each pixel is drawn uniformly from $[0, 1]$) and a hundred kernels (each element of the matrix is drawn from a standard normal distribution) of size from 1 to 100. For each size of a kernel we measure the average time of computing 2D convolution of the image and the kernel. We have used either numpy or torch optimised fast matrix multiplication rather than our Toeplitz matrix-matrix multiplication.

5.1 Numpy/Scipy implementation (CPU only)

The results of a benchmark are captured by figure 1. Our Toeplitz algorithm outperforms the scipy and numpy 2D convolution for larger kernels (over 85).

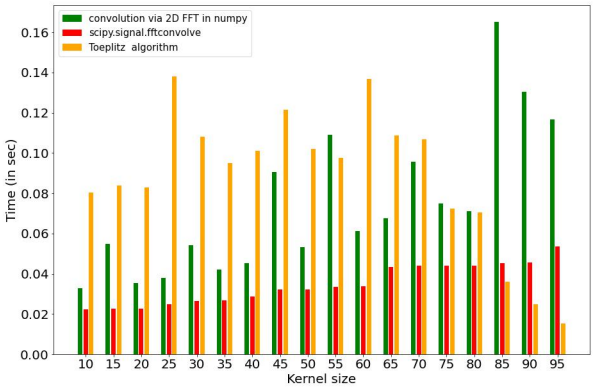

Fig. 1 The average time-performance of our Toeplitz 2D convolution algorithm versus the current implementation of 2D convolution in scipy *fftconvolve* function and the numpy implementation of 2D convolution via Convolution Theorem with numpy FFT library.

5.2 GPU implementations

The results of a benchmark are captured by figure 2. Both implementations have similar running times, apart from larger kernels over 85, where our algorithm has longer running times.

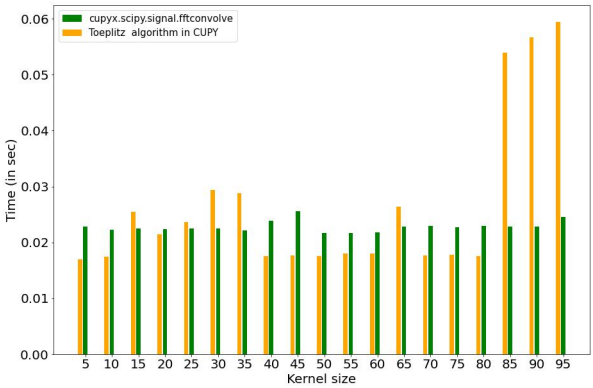

Fig. 2 The average time-performance of our Toeplitz 2D convolution algorithm implemented in cupy versus the current implementation of 2D convolution in *cupy.scipy.signal.fftconvolve*.

5.3 PyTorch implementations

The results of a benchmark are captured by figure 3. It appears that Pytorch 2D convolution is only optimised for small and very large kernels. Our implementation appears to have much less variation in running times for all kernel sizes .

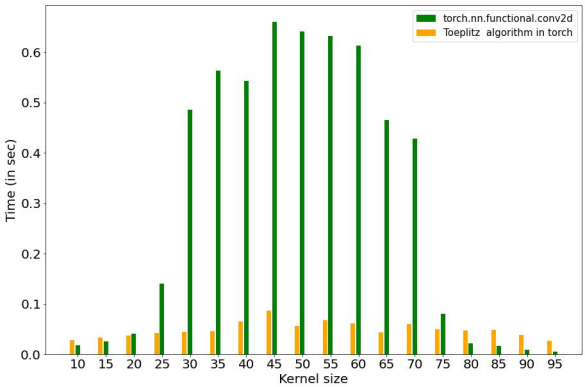

Fig. 3 The average time-performance of our Toeplitz 2D convolution algorithm versus the current implementation of 2D convolution in Pytorch.

The benchmarks were run on Ubuntu LTS 20.04 with CPU Intel i7-11800H and GPU GeForce RTX3070 8 GB RAM.

6 Conclusions

We have obtained an alternative representation of 2D convolution operation through Toeplitz matrix-matrix multiplication and provided a mathematical proof of this result. The current limitation of our approach is the complexity of our algorithm; when we apply Toeplitz matrix-matrix multiplication method the time complexity is lower than either the direct convolution or the convolution obtained through 2D FFT for some kernel sizes, it is never lower than both of the them simultaneously for the same image size and kernel size.

The future improvements of the algorithm should exploit the sparsity of the resulting Toeplitz matrix and the fact that we do not need the full matrix from the inverse Fourier transform in (19). Given the fact that we presented 2D convolution as a matrix multiplication allows to employ well optimised linear algebras libraries such as BLAS or SPBLAS.

Supplementary Materials

The code is available in the dedicated GitHub repository https://github.com/gnacikm/Conv2D_Toeplitz

Ethical Approval

Not Applicable

Availability of supporting data

Not Applicable

Competing interests

No, we declare that the authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

Funding

There is no funding/grants associated with this project.

Authors' contributions

M.G. wrote the main manuscript, proposed and proved the main results. K.Ł. has contributed to the benchmarks of the algorithm. All authors reviewed the manuscript.

Acknowledgements

M.G. would like to thank to his colleague James Burridge for helpful comments and suggestions.

References

[1] Valimaki, V., Parker, J.D., Savioja, L., Smith, J.O., Abel, J.S.: Fifty years of artificial reverberation. IEEE Transactions on Audio, Speech, and Language Processing **20**(5), 1421–1448 (2012). https://doi.org/10.1109/TASL.2012.2189567

[2] Bradski, G., Kaehler, A.: Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media, (2008)

[3] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation **1**(4), 541–551 (1989). https://doi.org/10.1162/neco.1989.1.4.541

[4] Cun, L., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: Advances in Neural Information Processing Systems, pp. 396–404. Morgan Kaufmann, (1990)

[5] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., Chen, T.: Recent advances in convolutional neural networks. Pattern Recognition **77**, 354–377 (2018). https://doi.org/10.1016/j.patcog.2017.10.013

[6] Krizhevsky, A., Sutshever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In Proc. 26th Annual Conf. on Neural Information Processing Systems (3–5), 1090–1098 (2012)

[7] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9 (2015). https://doi.org/10.1109/CVPR.2015.7298594

[8] Russakovsky, O., Deng, J., Su, H., *et al.*: Imagenet large scale visual recognition challenge. Int J Comput Vis **115**, 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y

[9] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)

[10] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)

[11] Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., Deng, L., Penn, G., Yu, D.: Convolutional neural networks for speech recognition. IEEE/ACM Transactions on Audio, Speech, and Language Processing **22**(10), 1533–1545 (2014). https://doi.org/10.1109/TASLP.2014.2339736

[12] Palaz, D., Magimai.-Doss, M., Collobert, R.: Convolutional neural networks-based continuous speech recognition using raw speech signal. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4295–4299 (2015). https://doi.org/10.1109/ICASSP.2015.7178781

[13] Chattopadhyay, A., Hass, P.: Predicting clustered weather patterns: A test case for applications of convolutional neural networks to spatio-temporal climate data. Sci Rep **10**, 1317 (2020)

[14] Borges Oliveira, D.A., Viana, M.P.: Fast cnn-based document layout analysis. In: 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), pp. 1173–1180 (2017). https://doi.org/10.1109/ICCVW.2017.142

[15] Truong, N.D., Nguyen, A.D., Kuhlmann, L., Bonyadi, M.R., Yang, J., Ippolito, S., Kavehei, O.: Convolutional neural networks for seizure prediction using intracranial and scalp electroencephalogram. Neural Networks **105**, 104–111 (2018). https://doi.org/10.1016/j.neunet.2018.04.018

[16] Wang, W., Gang, J.: Application of convolutional neural network in natural language processing. In: 2018 International Conference on Information Systems and Computer Aided Education (ICISCAE), pp. 64–70 (2018). https://doi.org/10.1109/ICISCAE.2018.8666928

[17] Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, (2006)

[18] A, K., Sohail, A., Zahoora, U., *et al.*: survey of the recent architectures of deep convolutional neural networks. A. Artif Intell Rev **53**, 5455–5516 (2020)

[19] Mallat, S.: Understanding deep convolutional networks. Trans. R. Soc. A **372**, 0203 (2015)

[20] Blum, A.L., Rivest, R.L.: Training a 3-node neural network is np-complete. Neural Networks **5**(1), 117–127 (1992). https://doi.org/10.1016/S0893-6080(05)80010-3

[21] Livni, R., Shalev-Shwartz, S., Shamir, O.: On the computational efficiency of training neural networks. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1. NIPS'14,

pp. 855–863. MIT Press, Cambridge, MA, USA (2014)

[22] Oh, K.-S., Jung, K.: Gpu implementation of neural networks. Pattern Recognition **37**(6), 1311–1314 (2004). https://doi.org/10.1016/j.patcog.2004.01.013

[23] Vanhoucke, V., Senior, A., Mao, M.Z.: Improving the speed of neural networks on cpus. (2011)

[24] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2015)

[25] Luo, L., Xiong, Y., Liu, Y.: Adaptive gradient methods with dynamic bound of learning rate. In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=Bkg3g2R9FX

[26] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90

[27] Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned cp-decomposition. CoRR **abs/1412.6553** (2015)

[28] Rigamonti, R., Sironi, A., Lepetit, V., Fua, P.: Learning separable filters. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2754–2761 (2013). https://doi.org/10.1109/CVPR.2013.355

[29] Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. ArXiv **abs/1405.3866** (2014)

[30] Hou, W., Tao, X., Xu, D.: Combining prior knowledge with cnn for weak scratch inspection of optical components. IEEE Transactions on Instrumentation and Measurement **70**, 1–11 (2021). https://doi.org/10.1109/TIM.2020.3011299

[31] Lioutas, V., Guo, Y.: Time-aware large kernel convolutions. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 6172–6183. PMLR, (2020). https://proceedings.mlr.press/v119/lioutas20a.html

[32] Peng, C., Zhang, X., Yu, G., Luo, G., Sun, J.: Large kernel matters – improve semantic segmentation by global convolutional network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)

[33] Liao, S., Samiee, A., Deng, C., Bai, Y., Yuan, B.: Compressing deep neural networks using toeplitz matrix: Algorithm design and fpga implementation. In: ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1443–1447 (2019). https://doi.org/10.1109/ICASSP.2019.8683556

[34] Sindhwani, V., Sainath, T., Kumar, S.: Structured transforms for small-footprint deep learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28. Curran Associates, Inc., (2015). https://proceedings.neurips.cc/paper/2015/file/851300ee84c2b80ed40f51ed26d866fc-Paper.pdf

[35] Zhao, L., Liao, S., Wang, Y., Tang, J., Yuan, B.: Theoretical properties for neural networks with weight matrices of low displacement rank. ArXiv **abs/1703.00144** (2017)

[36] Pan, V.Y.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Birkhäuser, (2001)

[37] Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, (1996)

[38] Pan, V.Y.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Springer, (2012)

[39] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.fftconvolve.html

[40] Ju, C., Solomonik, E.: Derivation and analysis of fast bilinear algorithms for convolution. SIAM Review **62**(4), 743–777 (2020). https://doi.org/10.1137/19M1301059

[41] Kim, H., Nam, H., Jung, W., Lee, J.: Performance analysis of cnn frameworks for gpus. In: 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 55–64 (2017). https://doi.org/10.1109/ISPASS.2017.7975270

[42] Frigo, M., Johnson, S.G.: The design and implementation of fftw3. Proceedings of the IEEE **93**(2), 216–231 (2005). https://doi.org/10.1109/JPROC.2004.840301

[43] Dudgeon, D., Mersereau, R.: Multidimensional Digital Signal Processing. Prentice-Hall, (1983)

[44] Chellapilla, K., Puri, S., Simard, P.: High Performance Convolutional Neural Networks for Document Processing. In: Lorette, G. (ed.) Tenth

International Workshop on Frontiers in Handwriting Recognition. Suvisoft, La Baule (France) (2006). Université de Rennes 1. https://hal.inria.fr/inria-00112631

[45] BLAS (Basic Linear Algebra Subprograms). http://www.netlib.org/blas/

[46] Hansen, P.C., Nagy, J.G., O'Leary, D.P.: Deblurring Images Matrices, Spectra, and Filtering. Siam, (2006)

[47] Lavin, A., Gray, S.: Fast algorithms for convolutional neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4013–4021. IEEE Computer Society, Los Alamitos, CA, USA (2016). https://doi.org/10.1109/CVPR.2016.435. https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.435

[48] Barabasz, B., Gregg, D.: Winograd convolution for dnns: Beyond linear polynomials. In: Alviano, M., Greco, G., Scarcello, F. (eds.) AI*IA 2019 – Advances in Artificial Intelligence, pp. 307–320. Springer, (2019)

[49] Winograd, S.: Arithmetic Complexity of Computations vol. 33. Siam, (1980)

[50] Van Barel, M., Heinig, G., Kravanja, P.: A superfast method for solving toeplitz linear least squares problems. Linear Algebra and its Applications **366**, 441–457 (2003). https://doi.org/10.1016/S0024-3795(02)00495-0. Special issue on Structured Matrices: Analysis, Algorithms and Applications

[51] Davies, P.J.: Circulant Matrices. Wiley, (1979)

[52] SPBLAS (SPARSE Basic Linear Algebra Subprograms). https://math.nist.gov/spblas/