Fast Convolutional Neural Networks with Fine-Grained FFTs

Yulin Zhang
Dept. of Electrical and Computer Engineering
University of Delaware
Newark, DE, USA
yzhan@udel.edu

Xiaoming Li
Dept. of Electrical and Computer Engineering
University of Delaware
Newark, DE, USA
xli@udel.edu

ABSTRACT

The Convolutional Neural Networks (CNNs) architecture is one of the most widely used deep learning tools. The execution time of CNNs is dominated by the time spent on the convolution steps. Most CNNs implementations adopt an approach that lowers the convolution into a matrix-based operation through the im2col (image to column) process. The transformed convolution then can be easily parallelized with highly efficient BLAS libraries. The contribution of this paper is that we observe significant but intricately patterned data redundancy in this matrix representation of convolution. This redundancy has not been exploited before to improve the performance of CNNs. In this paper, we analyze the origin of the redundancy generated by the im2col process, and reveal a new data pattern to more mathematically concisely describe the matrix representation for convolution. Based on this redundancy-minimized matrix representation, we implement a FFT-based convolution with finer FFT granularity. It achieves on average 23% and maximum 50% speedup over the regular FFT convolution, and on average 93% and maximum 286% speedup over the Im2col+GEMM method from NVIDIA's cuDNN library, one of the most widely used CNNs libraries. Moreover, by replacing existing methods with our new convolution method in a popular deep-learning programming framework Caffe, we observe on average 74% speedup for multiple synthetic CNNs in closer-to-real-world application scenarios and 25% speedup for a variant of the VGG network.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Theory of computation** → *Parallel computing models*; • **Software and its engineering** → *Massively parallel systems*.

KEYWORDS

convolutional neural network, algorithm optimization, fast Fourier transform, GPU

ACM Reference Format:
Yulin Zhang and Xiaoming Li. 2020. Fast Convolutional Neural Networks with Fine-Grained FFTs. In *Proceedings of the 2020 International Conference on Parallel Architectures and Compilation Techniques (PACT '20), October 3–7, 2020, Virtual Event, GA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3410463.3414642

1 INTRODUCTION

Deep convolutional neural networks (CNNs) have become one of the most successful deep learning tools in recent years. One of the first successful CNN models can be traced back to [25] and later improved in [26]. As Krizhevsky et.al. made significant breakthrough in vision recognition with CNNs [21] in 2012, deeper and more complicated network structures are proposed to increase CNN expressiveness [34] [35] [37] [15]. ResNet [15] is 20 times deeper and 8 times deeper than AlexNet [21] and VGGNet [35], respectively. As the complexity of CNNs increase, their computation complexity increases exponentially. Furthermore, for real-time interactive applications where the CNN is deployed, e.g., autonomous driving [3][5] and video surveillance systems [9], long latency is not acceptable. Thus, the computation efficiency of CNN becomes an essential factor in its research and application.

A typical convolutional neural network consists of many different layers. Among these layers, the bulk of the computation is performed on convolutional (CONV) layers [14] [8] [31] [38] [18] [28][33]. It is reported in [18] that the convolution layer accounts for 92% of the execution time of forward pass on a Nvidia GPU for the AlexNet model. The backward pass has similar breakdown of computation time. It is also shown in [28] that convolutional layer consumes 86%, 89%, 90% and 94% of the total execution time for GooleNet, VGG, OverFeat and AlexNet CNN models respectively in one forward and one backward propagation. Convolution layer requires a substantial amount of computation resources, especially for modern advanced CNN models with deeper and complicated architecture. Naturally, prior research on CNNs' performance has been focused on optimizing the convolution process.

One of the most widely adopted approaches is to exploit GPU's massive parallel computing capability to accelerate CNNs. Among many GPU-based CNN implementations, Nvidia has developed a highly efficient drop-in deep learning acceleration library cuDNN [6] with optimized routines on GPUs. Most deep learning frameworks support GPU by default [19] [40] [1] [2] [7].

Another line of work is to optimize the backend implementation of convolution. There are two representative approaches: (1) Use FFT to carry out convolutions in the Fourier domain [30] [4] [23]. Experimental results [38] have shown that larger convolution kernels yield more performance gain. As CovNets utilized smaller kernels [35] [17], Winograd algorithm has been proposed to reduce the amount of multiplication at cost of performing more additions. For large kernels, however, the saving on multiplication may not be enough to compensate the extra additions. (2) Use matrix-matrix multiplication to directly calculate convolution. The approach [4] is representative as it transforms convolution into matrix multiplication, and then highly tuned GEMM libraries can be invoked to compute matrix multiplication.

This paper presents our work on optimizing CNN's convolution process at the backend-implementation level. Our key insight, and also the main contribution of this paper is that we observe significant yet intricate-patterned redundancy hidden in the matrix-based representation of CNN's convolution process. This redundancy has been largely overlooked in prior work, but can be transformed to reduce the computational complexity, and therefore to improve the performance of CNN's convolution. We present a systematic study of the redundancy and reveal a doubly block Hankel matrix data pattern for an unrolled input feature map. Based on this data pattern, contrary to the regular FFT convolutions that take 2D FFT over the entire feature map, we implement a new FFT-based convolution with finer granularity, which yields notable performance improvements compared to existing state-of-the-art implementations. We conduct various comparisons of these two algorithms. The experimental results suggest that the fine-grained FFT approach outperforms the regular FFT method for both synthetic and real-world benchmarks.

2 OVERVIEW AND BACKGROUND

In this section, we present an overview of CNN from the point-of-view of computation complexity and performance with a particular highlight on CNN's most time-consuming step—convolution. Then we briefly discuss im2col+MM convolution and FFT-based convolution to which our proposed method is closely related. To help relate to other work, the deep-learning domain notations used in this paper are summarized in Table 1.

2.1 Convolutional Neural Network

A typical CNN is composed of multiple stages [24]. Each stage consists of a CONV layer, a non-linearity layer and a pooling layer. Notably non-linearity is introduced into CNNs using non-linearity layers and pooling layers to help the output feature map to be robust and invariant to small shifts and distortions in the previous layer [27]. Convolutional Networks (ConvNets) consists of multiple convolutional (CONV) layers to extract features from the input. They act as a feature extractor that extract feature maps by different convolutional kernels. Each kernel extracts a feature from input images, and a convolutional layer typically uses multiple kernels to extract multiple feature maps. Low-level features extracted at lower convolutional layers are combined to more abstract features at higher layers. In CNNs, the input and output to the convolutional layers are often referred to as feature maps. Each neuron at the current convolutional layer is connected to a local region of the previous layer with the full depth dimension, which is referred as local receptive field. It results in a reduced number of connections between layers by only connecting to the local receptive field of previous layers. The property associated with the sparsely connected layers is called sparse connectivity [10]. Another key idea of CNNs is parameter sharing [10]. It is accomplished by applying the same weights over the input feature maps at all positions to extract the output feature maps. It further reduces the storage requirement for the parameters. The output feature maps represent a particular feature extracted from the input. Convolutional layer produces a feature map for each filter, thus the number of output feature maps depends on the number of filters. The last layer of CNNs is a fully-connected (FC) layer which combines the results of convolutions of a set of relatively high-level features for classification purpose.

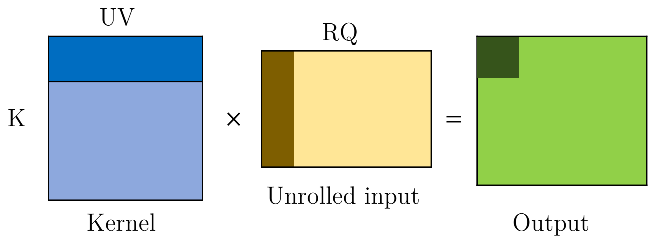


Figure 1: Illustration of im2col+MM convolution, which transforms a convolution into a matrix multiplication. Refer to Table 1 for notations.

A two-dimensional (2D) convolution performs the dot product between the filter and the corresponding values in the feature map. In fact the convolution operation in CNNs is essentially batched 2D convolution. To convolve the multiple channels in images with the filter, CNNs perform a 2D convolution separately in corresponding channels and sum results across all the channels. For multiple filters, the output can be viewed as the concatenation of the resultant matrices generated by the input and corresponding filters.

2.2 im2col+MM Convolution

Im2col+GEMM, also known as lowering or unrolling convolution, is a straightforward and efficient approach to compute convolution. Im2col (image to column) is the step where image patches based on the kernel size are rearranged into columns and further reorganized into a concatenated matrix. Im2col-based convolution first unrolls/lowers input images to 2d matrices. Since kernels are already stored as the kernel matrix, convolution is converted to a general matrix multiply (GEMM). Each row of the output matrix corresponds to an output feature map, whose width is determined by the following formula.

$$Q = (W - V + 2P)/S + 1 \qquad (1)$$

Where Q, W, V, P, and S are explained in Table 1. The latter three hyper-parameters control the width of output feature map. We only consider the width here, but it can be trivially applied to the height. Figure 1 presents the im2col+MM version of convolution. It takes a $H \times W$ input feature map and K $U \times V$ kernels. Each column in the unrolled input matrix contains the 2D patch and each row of the kernel matrix corresponds to a $U \times V$ kernel. Multiplying these two matrices results in an output matrix that each row is an output feature map.

It is first observed in [4] that im2col-based convolution to matrix multiplication has high performance in CNNs. Yanqing et al. [19] independently found that it is efficient in Caffe deep learning framework. By transforming a convolution into a general matrix multiplication, it takes the advantage of highly-tuned linear algebra libraries, e.g., cuBLAS. However, the main disadvantage is that it causes significant memory overhead during the im2col process.

Table 1: Summary of Notations

Name	Description
N	Mini-batch size
K	Number of kernels
H	Input height
W	Input width
R	Output height
Q	Output width
C	Input channels
U	Kernel height
V	Kernel width
S	Stride
P	Padding

2.3 FFT Convolution

FFT-based convolution [30] [38] makes use of Fast Fourier Transform (FFT) to compute pointwise products in frequency domain, which are equivalent to spatial convolutions based on the convolution theorem.

$$f * g = \mathcal{F}^{-1}(\overline{\mathcal{F}(f)} \cdot \mathcal{F}(g)) \tag{2}$$

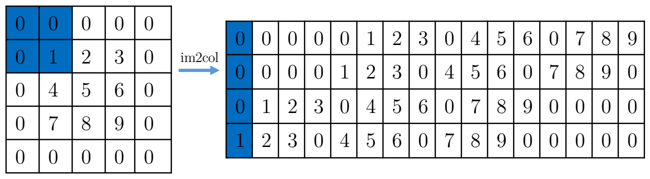
Where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier Transform and inverse Fourier Transform, and $*$, $\cdot$, and $\overline{}$ are convolution, complex pointwise product, and complex conjugation, respectively. In this approach, the sizes of both input tensor and weight tensor have to be equal through the padding with zeros prior the transformation. Then they are transformed from the spatial domain to the frequency domain with FFT. Following the FFT transformation, a pointwise multiplication is performed between the resulting input FFT transform and the complex conjugate of the filter FFT result. The last step is to apply inverse FFT transform to return to the spatial domain.

FFT-based approach greatly reduces the algorithmic complexity of convolution in the spatial domain. However, one major drawback is the need for large temporary memory. First, the weight tensor needs to be padded to the same size as input tensor. The memory overhead is high if input tensor is much larger than weight tensor. When the weight size is significantly smaller than the input, too much padding to the input could occur and FFT-based convolution is less efficient. As a consequence, a tiling strategy [38] [16] is used to decompose a large convolution into smaller ones, which can be used to reduce the memory overhead. Second, additional memory is required to store the FFT coefficients. In our work, we find the symmetry property of real inputs in the Fourier space can reduce the storage of FFT coefficients to half. The symmetry is also exploited to reduce the pointwise product computation cost.

2.4 Other Related Work

Besides Im2col+MM and FFT-based convolutions, some other efficient convolution methods have been proposed.

Direct convolution, as the name implies, directly perform convolution. However, due to its across-channel dependency, it will not expose sufficient parallelism to fully utilize the resources on GPUs. Cuda-convnet [20] is one of the earliest CNN frameworks with direct convolution implementation. It achieves high efficiency when batch size is large [6], but the efficiency generalizes poorly once batch size is 64 or below. Lavin et al. [22] develop an efficient convolution maxDNN based on SGEMM implementations created by an open source assembler for NVIDIA Maxwell GPUs [12]. maxDNN is written in low-level Maxwell assembler, and it takes advantage of hardware specific optimizations and obtains high performance. However the optimization technique is not portable for other architectures. Cong et.al. [8] employ the Strassen matrix multiplication [36] to reduce the arithmetic complexity of the redefined matrix multiplication, where each element in the matrix is a feature map, reducing the number of operations by up to 47%. Lavin et al. [23] introduced a fast convolutional algorithm to reduce the complexity using Winograd's minimal filtering algorithm [39]. However, its number of operations grows quadratically with kernel size. Additionally, the numerical accuracy of Winograd convolution decreases as larger kernels are used.



0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

0	0	0	0	0	1	2	3	0	4	5	6	0	7	8	9
0	0	0	0	1	2	3	0	4	5	6	0	7	8	9	0
0	1	2	3	0	4	5	6	0	7	8	9	0	0	0	0
1	2	3	0	4	5	6	0	7	8	9	0	0	0	0	0

Figure 2: A 3×3 input with zero-padding of size 1 convolves with a 2×2 kernel (shown in blue on the left). On the right is a larger matrix generated by the im2col process with redundancy.

3 MOTIVATION

In this section we explain how data redundancy is discovered from the implementation details of the im2col-based convolution, and how this novel observation motivates the proposed more efficient implementation of convolution.

Recall that the im2col operation reshapes the input feature map as a concatenation of columns stretched by the local patches of the input feature map. The kernels are already stored as a kernel matrix. Therefore, the convolution is transformed to a matrix multiplication to take advantage of highly optimized GEMM libraries. During the im2col process, since the receptive fields overlap, the elements in the overlapped area are duplicated into multiple distinct columns. The duplication of the overlapped elements incurs lots of redundancy. The exact degree of redundancy is hard to predict analytically, however can be measured once all parameters are known. In addition to the redundancy of overlapped elements, there is another kind of redundancy due to the treatment of the input feature map. Sometimes the convolution layer would pad the input feature map with zero to keep the spatial size constant as well as preserve the information at the border. Thus the im2col process also duplicates zeros. We could skip multiplications that always give zero in matrix multiply if we know the distribution of zero element after the im2col operation. For example, Figure 2 shows a 3×3 input with zero-padding of size of 1 with respect to a 2×2 kernel are processed to generate an unrolled output by the im2col operation.

To exploit the redundancies in im2col and develop more efficient convolution algorithm, we need to answer three questions. (1) How

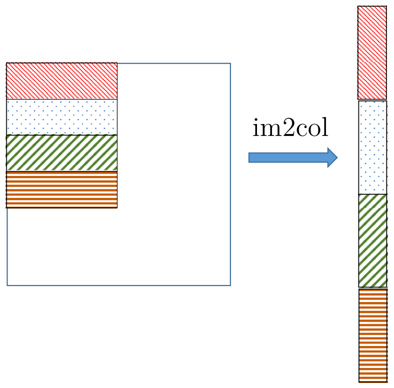


Figure 3: The close look of im2col process by reshaping the input patch overlapped with a kernel of four rows into a column.

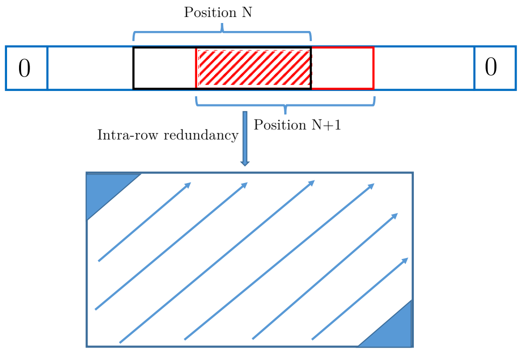


Figure 4: The row kernel slides on the one-row feature map, and it incurs intra-row redundancy. The skew diagonals denoted by the blue arrows are constant, and blue triangles represent zero elements.

to mathematically describe the duplication of the input feature map in im2col? (2) How to mathematically describe the distribution of zero elements introduced in zero-padding? And (3) Where are the absolutely necessary elements in the resultant matrix after the im2col step? In the remainder of this paper, we develop a recursive data pattern to describe the redundancy we find. With the redundancy pattern description, we are able to transform the convolution algorithm to avoid computation and storage on those redundant elements. The final product is our fine-grained FFT convolution algorithm.

4 A NEW DATA PATTERN

In this section, we first take a close look at how im2col works for each row of the kernel, and then show why the particular visiting process introduces two types of redundancy when im2col's kernels slide on CNNs' feature map both horizontally and vertically. We then develop a concise mathematic presentation to describe the redundancy mechanism. Particularly we reveal the connection between the data pattern and the doubly block Hankel matrix [32], and present a new way to express the data pattern unique to the im2col process. With all these, we present the theoretical foundation of how the convolution in CNNs can be asymbolitically optimized.

4.1 im2col Process

Let us first take a closer look at the im2col operation since the new data pattern is dependent on the im2col operation. As shown in Figure 3, the kernel has four rows indicated by different colors, and the im2col operation transposes and concatenates each row of the overlapping patch in the feature map into one long column. As the kernel slides horizontally and vertically, each row in the kernel works independently. Alternately, we can think of this 2D filter as a set of 1D row kernel, and the final resultant matrix generated by im2col is the composition of results of each row kernel. As the kernel slides horizontally, it incurs the redundancy within the row kernel, which we name it intra-row redundancy.

4.2 Intra-row Redundancy

Figure 4 shows an example of taking a row from both the feature map and the filter. The one row feature map is then padded one zero on both sides. We assume the length of the feature map and kernel are m, n, respectively. As the filter slides along the feature map, the current position $N + 1$ and the previous position N (denoted by black rectangular) are overlapped by $n - 1$ (indicated by red striped pattern) and only the leftmost element in the previous position and the rightmost element in the current position are distinct and contain new information. As the filter slides horizontally, im2col operation transposes the elements in these positions to columns and concatenate them in a shoulder by shoulder manner. Once the row filter finishes sliding to the rightmost, elements in each neighboring of columns generated by im2col are overlapped by n-1 elements and shifted up by one. Because zeros are padded symmetrically on the border, the upper left and the lower right blue triangles are zeros, where their length of sides along height and width directions is equal to p (p is padding size).

4.3 Inter-row Redundancy

In the convolution process, the kernel visits the feature map starting from the top left corner and moving a stride size step to the right. In this process, rows in the kernel independently incurs the redundancy pattern as shown in Figure 4. When the kernel reaches the right border of feature map, it shifts one row down, where another type of redundancy is incurred because each row-kernel will traverse the elements the previous row-kernel just traversed. Hence, we name it inter-row redundancy. The kernel continues to go through the same process until it finishes at the bottom right corner of the feature map. In the example shown in Figure 5, the kernel has four rows indicated by different colors and the feature map is padded zeros with size one. As the kernel reaches the bottom right corner, k_1, k_2, k_3 and k_4 independently generate rows from one to four on the output matrix and each block has intra-redundancy. Blocks on skew diagonals are the same, which is indicated by blue dotted arrows. Since we only pad zeros with size one, the upper left and lower right blocks are zero matrices.

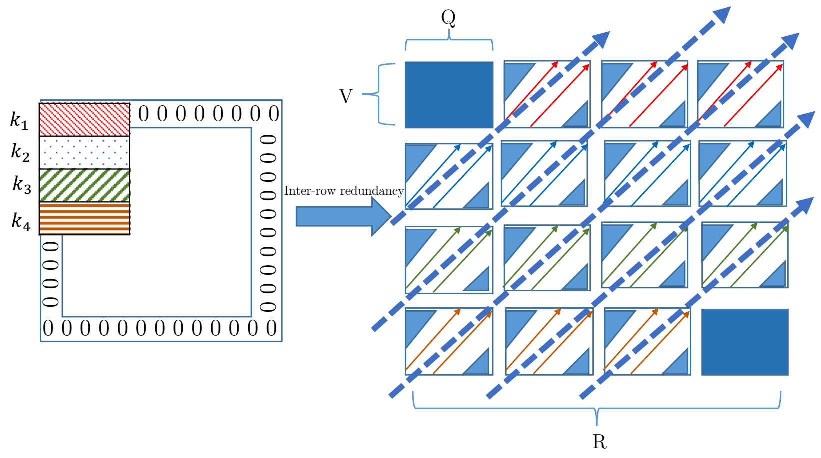


Figure 5: As the kernel traverses the entire feature map, the output matrix generated by im2col has inter-row redundancy denoted by dotted blue arrows. Blue blocks are zero matrices. Each block is $V \times Q$ with total number of $U \times R$. Refer to table 1 for notations.

4.4 Doubly Block Hankel Matrices

With the intra-redundancy and the inter-redundancy revealed, we can now answer the questions asked in section 3 as to how to express and use the redundancy patterns.

The data pattern, which features both intra-row and inter-row redundancy, can be qualitatively summarized as following. Each block along skew-diagonals in Figure 5 are identical, while each block has intra-row redundancy that elements along skew-diagonals are constant. Let us now quantify the data pattern's composition.

The im2col process converts the input patches to columns and forms an unrolled matrix, which has the inter-row redundancy, whereas each block has intra-row redundancy, as demonstrated in figure 6. The padded zero elements are distributed in the upper-left and lower-right blocks of the unrolled matrix. Additionally, they are distributed within the upper-left and lower-right corners of each non-zero blocks. Each row in the input feature map is distributed to the first column and last row within the blocks depicted in figure 6, because a new element is shifted up in intra-row redundancy.

We now are able to track all the elements from the input feature map and fully reveal the data pattern introduced by the im2col process. Let us formally define it. We set the feature map and kernel sizes to be $m \times m$ and $n \times n$, padding the feature map with zeros of size p around the borders. The im2col operation rearranges the feature map into dimensions of $n^2 \times (m - n + 2p + 1)^2$, and each block in the output matrix is $n \times (m - n + 2p - 1)$.

$$O_{inter|intra}[i][j] = O_{inter|intra}[i-1][j+1] \tag{3}$$

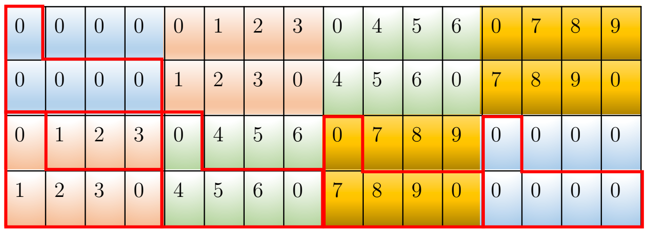
$i > 0, j > 0$ and $i < n, j < m - n + 2p - 1$, where O denotes output matrix. The padded zero distribution follows the equation.

$$O_{inter|intra}[i][j] = 0 \tag{4}$$

such that $i < p, j < p$ or $i > n - p - 1, j > m - n + p$. i and j denote the indices for the elements and blocks in intra-row and inter-row redundancy respectively.

We make a key finding here. The equation 3 that summarizes the intra-row and inter-row redundancy patterns is in fact the definition for a Hankel matrix [32] where each value along skew-diagonals are constant. The output matrix generated by im2col for an input feature map is actually a *doubly block Hankel matrix*. Each individual block in the matrix is a Hankel matrix, and the whole matrix with respect to its blocks is also a Hankel one. In CNNs, the kernel matrix has dimensions $K \times CUV$, and the input data with dimension $NCHW$ is unrolled to N matrices with dimensions $CUV \times RQ$. Each matrix has C sub-matrices ,and each of them is a doubly block Hankel matrix with size of $UV \times RQ$ (see table 1 for notations).

Using the doubly block Hankel matrix to represent the feature map matrix generated by the im2col, the convolution actually become the multiplication between Hankel matrix and vector. Due to the intrinsic data redundancy in Hankel matrices, such Hankel-matrix-vector multiplication has theoretically lower computational complexity than a generic matrix-vector multiplication. We use the Fast Fourier Transform (FFT) to asymbolitically optimize the Hankel-matrix-vector multiplication and therefore improve the performance of the convolution in CNNs. In the next section, we will present our proposed fine-grained FFT-based convolution in details.



0	0	0	0	0	1	2	3	0	4	5	6	0	7	8	9
0	0	0	0	1	2	3	0	4	5	6	0	7	8	9	0
0	1	2	3	0	4	5	6	0	7	8	9	0	0	0	0
1	2	3	0	4	5	6	0	7	8	9	0	0	0	0	0

Figure 6: A new data pattern revealed in figure 2 that each block with the same color are the same, and element along the skew diagonals are constant. The elements bounded by red lines correspond to individual rows from the input feature map.

5 FINE-GRAIN-FFT-BASED CONVOLUTION ALGORITHM

We have shown that im2col-based convolution unrolls each input feature map to a doubly block Hankel matrix, and the convolution is transformed to a matrix multiplication between the kernel matrix and doubly block Hankel matrices. Disregarding such redundancy, existing implementations of the im2col+MM convolution approach such as that in cuDNN all directly compute the kernel and unrolled input matrices multiplication using BLAS libraries such as cuBLAS. In this section, we show how the Hankel matrix data pattern enables the use of FFT to more efficiently compute the specific matrix multiplication. We then introduce the complete FFT Hankel matrix vector multiplication algorithm in the context of convolution. At last, we demonstrate analytically that our fine-grain FFT based convolution algorithm not only reduces the computational complexity by FFT, but also reduces the memory overhead because it eliminates the needs to fully unroll the input data and replaces unrolling with the implicit element-wise matrix multiplication.

5.1 Hankel Matrix Vector Multiplication

Recall that for N $H \times W$ C-channel input images, im2col transforms them to NC doubly block Hankel matrices of size $UV \times RQ$ with Hankel blocks of size $V \times Q$. The corresponding dimensions of the kernel matrix are $K \times UVC$ for K kernels with spatial dimensions $U \times V$ since it is required that the input tensor and the set of K kernels have the same depth size C in a CNN convolution layer. Here we first show a fast multiplication to multiply a vector v of size $1 \times V$ from the kernel matrix with the Hankel block $\mathcal{H}$ of size $V \times Q$, and then in section 5.2 extend it to the doubly block Hankel matrix method.

Hankel matrices are referred as structured matrices, which can be described without loss of information much more concisely than the n^2 elements in $n \times n$ matrices. The immediate benefit is that the storage complexity can be significantly reduced. More importantly for the convolution algorithm, much lower computational complexity for structured matrix-vector product can be obtained via fast matrix-vector products by FFTs. The Hankel block $\mathcal{H}$ can be embedded into a $2Q \times 2Q$ circulant matrix X, and the multiplication by the kernel vector v can be achieved by FFTs. The circulant matrix X can be completely specified with only the first row, which is also known as the generating vector x. Each subsequent row is obtained by doing a right-shift of the previous row by one, wrapping around at the edges. It is diagonalized by the Discrete Fourier Transform (DFT) matrix regardless of vector x [11]. It can be expressed as

$$X = \mathbb{F} \triangle \mathbb{F}^{-1}$$

where $\mathbb{F}$ is the DFT matrix and $\triangle$ is a diagonal matrix containing the eigenvalues of X such that $\Delta = \text{diag}\,(\mathbb{F}x)$. Therefore, multiplying the circulant matrix X with the kernel vector v is as follows.

$$X\hat{v} = \mathbb{F} \triangle \mathbb{F}^{-1}\hat{v} = \mathbb{F}(\mathbb{F}x \circ \mathbb{F}^{-1}\hat{v})$$

where $\circ$ denotes the Hadamard element-wise vector multiplication and $\hat{v} = (v_V, v_{V-1}, \dots, v_1, 0, \dots, 0)$. It first computes a DFT $\mathbb{F}x$ and an IDFT $\mathbb{F}^{-1}\hat{v}$ and then a final DFT $\mathbb{F}(\mathbb{F}x \circ \mathbb{F}^{-1}\hat{v})$. These three DFTs can be computed efficiently by applying FFTs. Their computational complexity is $O(2Q \log 2Q)$, thus our fine-grained FFT algorithm works at $O(2Q \log 2Q)$ granularity.

5.2 Implicit Element-Wise Matrix Multiplication

The previous section explains the transformation of the unrolled input feature map into a doubly block Hankel matrix data pattern and the use of FFT to optimize the Hankel matrix vector multiplication. In this section we develop an efficient implementation of the Hankel block matrix multiplication. In particular, we present a unique optimization technique that is derived from the linearity of the DFT. Because DFT is a linear transformation, the linearity allows the sum of the element-wise product directly in the Fourier domain, which leads to considerable savings of FFT time.

We first briefly overview the existing techniques used in the implementation of the im2col-based convolution. Caffe's default implementation calls matrix multiplication iteratively for each image in the mini-batch. In contrast, Gu et al [13] have showed a performance boost of around 4-5 times is obtained by using batched im2col over multiple images. The batched scheme increases data parallelism and moves the unrolled matrix size to a more favorable region in BLAS. Hadjis et.al. [14] demonstrated that batching up multiple input images against the same kernel matrix once can improve the performance. In this work we adopt the batched scheme that the kernel matrix multiply all the input feature maps in a batch in the Fourier domain.

Comparing to the batching mode, a more important performance issue is the handling of the data redundancy and its derivative memory/computation overhead. All prior work requires unrolling the feature map completely or partially. In our work, we find that the redundant data pattern make it possible to not unroll the input matrix at all for the computing of convolution. It is because a doubly block Hankel matrix can be fully specified by only the distinct elements, e.g., elements bounded by red lines are distinct elements in Figure 6. The key insight is that the Hankel block matrix has an interesting property that the generating vector for the circulant matrix happens to contain all the distinct elements, and the rest elements are padded. This interesting structure of the circulant matrix makes it possible to compute the matrix vector product by only using the generating vector. Furthermore, these distinct elements are extracted by the im2col process from each individual row of the original feature map, as demonstrated by the red lines in Figure 6 and the original feature map in Figure 2. Therefore we don't need the completely unrolled input matrix, as required in existing approaches, for the computing of convolution. All these reduce not only the memory foot-print but also the computational complexity of the proposed convolution method.

Specifically in implementation, we develop an implicit element-wise matrix multiplication strategy in the Fourier domain. It uses an indexing arithmetic to load the corresponding Fourier coefficients from input feature maps without unrolling them. Accordingly, our approach avoids the FFT computations of redundant Hankel blocks, which in turn reduces the storage of the Fourier coefficients for the redundant Hankel blocks. Compared with a fully unrolled input feature map with size of $H \times W$, it requires $U \times R$ FFT computations of Hankel blocks, whereas our approach only needs $U + R - 1$.

5.3 FFT Hermitian Symmetry

We take advantage of another property of FFT to further optimize both the memory storage and the operation complexity of our convolution method. The Fourier transform of a real-valued input is Hermitian symmetric (conjugate complex symmetry). The symmetry allows us to store roughly half Fourier representations to carry out the complete FFT. For even N real-valued input x_i, $i \in \{0, \dots, N-1\}$, the Fourier representations of X_0 and $X_{N/2}$ are real-valued, and X_1 through $X_{N/2-1}$ are the complex conjugates of $X_{N/2+1}$ through X_{N-1}. Thus, we only need to store $N/2+1$ complex numbers. Furthermore, we can use the symmetry property to reduce the number of products in element-wise multiplication by almost half. Specifically, the second half of element-wise multiplication can be constructed by simply taking the complex conjugate of the first half. Each element-wise product in Fourier domain requires four multiplications for two complex numbers. Using Gauss' multiplication algorithm [29], the number of multiplications is reduced from four to three. For two complex numbers $a+ib$ and $c+id$, it first computes $t_1 = c*(a+b)$, $t_2 = a*(d-c)$ and $t_3 = b*(c+d)$, and the real and imaginary parts of the result can be computed as $t_1 - t_3$ and $t_1 + t_2$, respectively. Similarly, each element-wise multiplication of complex numbers is replaced by three element-wise multiplication of real numbers.

5.4 Overall Working Flow

Overall the proposed convolution method is implemented in four steps:

Step 1 Input transform. Since the generating vector of circulant matrix is already contained in each row of input feature maps, we apply 1D FFTs to each row to transform the input. For the best performance in cuFFT, it is worth noting that cuFFT performance is sensitive to FFT size. Very often slight changes in size result in large performance differences due to different implementations used in cuFFT. Therefore we use input padding to find the best cuFFT case for our particular FFT problem instances.

Step 2 Kernel transform. To transform the kernel into Fourier domain, we decompose the kernel matrix into $K \cdot U \cdot C$ tiles, and perform $K \cdot U \cdot C$ 1D FFTs using the batch mode provided by cuFFT.

Step 3 Element-wise computation. For this step, where block matrix multiplication with element-wise product is performed since the doubly Hankel matrix is already partitioned into Hankel blocks. Within each block, we perform element-wise product.

Step 4 Inverse transform. Lastly, inverse FFT transform is performed on the output matrix from Step 3. Only $1 \times Q$ element in the $1 \times 2Q$ output is valid for the $1 \times V$ vector and $V \times Q$ Hankel matrix multiplication, the rest is discarded.

5.5 Arithmetic Complexity Analysis

Next we compare the computation complexity of the proposed fine-grained FFT based convolution against the existing regular FFT based approach.

Both regular and fine-grained FFT approaches perform convolutions in four basic steps: input transform, kernel transform, element-wise multiplication, and inverse transform. It worth noting that the element-wise multiplication step in the fine-grained FFT approach is in fact matrix multiplication with element-wise product. As a point of comparison, we could treat it as the element-wise multiplication. The first two steps transform inputs and kernels from a spatial domain to Fourier domain. The third step can be converted to a batched complex general matrix multiplication (Cgemm) for the regular FFT approach. The inverse transform step converts the results back to the spatial domain. Assume we have (N, C, H, W) inputs and (K, C, U, V) kernels, the RegularFFT approach needs $2W^2 \cdot \log W \cdot K \cdot C$ to perform the most expensive Kernel Transform step, and the fine-grained FFT approach needs only $2W \cdot \log 2W \cdot K \cdot C \cdot V$ operations, a reduction by a factor of W/V. Notably, the complexity of the regular FFT convolution does not depend on the kernel size. It is expected that the regular FFT approach performs the same regardless of the kernel size since it zero-pads the kernel to be the same size as the input image before applying the FFT. On the other hand, the complexity of our method depends on the kernel size.

5.6 Autotuning

We apply a simple autotuning strategy to tune our implementation on GPU. Basically our autotuning selects the best configuration of CUDA thread and block parameters for given constraints of input settings and hardware resources. The result of the autotuning can be stored locally and re-used when a similar configuration of inputs passed to the implementation. More specifically, the fine-grained FFT consists of four major steps, and each step can have different values for the CUDA thread and block parameters. As an example, we autotune the point-wise multiplication step to find the optimal combination of CUDA parameters, BLOCK_SIZE and NUM_BLOCKS, which represent the thread block size and the number of thread blocks, respectively. The autotuning strategy explore different BLOCK_SIZE and NUM_BLOCKS combinations, where BLOCK_SIZE $\in [32, r]$, where r is NextPowerTwo(2Q)/2+1 and NextPowerTwo is a function to find the next power of two number. NUM_BLOCKS's range is defined as $[1, N \times K \times R]$. Thus we execute and measure only reasonable parameter combinations to avoid the exhaustive coverage of the search space. As a result, the time spent on autotuning is reduced. Overall autotuning gives us the speedup of about 5% compared with our pre-autotuned version.

6 EVALUATION AND PERFORMANCE ANALYSIS

We evaluate the proposed convolution method from four aspects: (1) accuracy of result, (2) kernel-level performance comparison with NVIDIA's cuDNN library [6], (3) application-scenario performance with networks developed in Caffe[19], a leading deep learning programming framework, by replacing Caffe's own convolution method, and (4) performance profiling to analyze and understand the source of performance improvement.

Each performance point is the average of five runs. Since the performance of convolution is independent to input values, we randomly generate inputs and use the same input for each data point. The versions of cuDNN and Caffe are 7.1 and 1.0, respectively. Hardware-wise, all the experiments are performed on Nvidia Titan XP GPU. In our experiments, CPU only serves as the command processor and has a negligible impact on performance.

6.1 Accuracy:

We first measure the accuracy of our method by comparing the results of our convolution with those computed by the im2col+MM approach. Table 2 shows the numeric accuracy of fine-grained FFT convolution using the convolution configurations listed in the top row. The error is in the order of 10^{-11}, which means that our optimization technique maintains the numerical integrity of convolution.

(U, K, S, P)	*(3, 10, 1, 2)*	*(5, 32, 1, 2)*	*(4, 10, 1, 2)*
Error	4.73E − 11	3.00E − 11	2.38E − 11

Table 2: FineGrainedFFT convolution absolute element error. Ground truth is computed by cuDNN's im2col+MM method.

6.2 Kernel Performance Comparison:

We measure the pure GPU kernel execution times in order to compare head-to-head how our method performs in terms of the kernel-level performance against the FFT-based convolution methods in cuDNN. cuDNN is generally considered as a library that is deeply optimized by NVIDIA and provide some state-of-the-art and fastest convolution implementations. We vary kernel sizes, and batch sizes to analyze strengths and weaknesses for these algorithms in the parameter space. We organize these parameters into a 2-tuple (U, N), and we assign a set of values to the other parameters (K, C, H) that is commonly used in benchmarking convolution performance. The experiment is then categorized into two groups. Each group fixes the value of one parameter and varies the other one. Thus, we can study how this parameter impact the overall performance of the algorithm. Please note that the performance of our method is dependent on the parameters of convolution. However it is insensitive to the value of inputs and weights. Due to the performance's insensitivity to input, the performance we observe at the kernel level is going to be consistent with that with real-world data.

In Figure 7, we compare the fine-grained FFT algorithm with the regular FFT algorithm from cuDNN on a synthetic benchmark and the 2017 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) object localization benchmark. We use random input data and kernels from $[-1, 1]$ for the synthetic benchmark. The execution time of regular FFT convolution tends to be constant and insensitive to the kernel size, because it zero-pads the kernel to be the same size as the input image and kernel size has nearly no effect on the performance. In contrast, the performance of our method decreases with the kernel size since our algorithm is based on the padded matrix. Although the matrix is not unrolled, the algorithmic complexity is still dependent on the kernel size. Figures 7a and 7b have similar trend. Our fine-grained FFT convolution maintains its high performance when the kernel size is small. It is shown that our algorithm is insensitive to the value of inputs and kernels.

In Figure 8, we also compare the fine-grained FFT algorithm against the im2col+GEMM algorithm from cuDNN. Although the execution time for both approaches increases as the kernel size increases from 5 to 30, the im2col+GEMM approach increases more rapidly. Our approach outperforms the im2col+GEMM method mainly because the im2col+GEMM method fully unroll matrices, and the matrix sizes grow quadratically with the kernel size. In contrast, our approach does not fully unroll the input. In addition, the kernel and input matrix multiplication performed by FFTs has lower algorithmic complexity. Compared with the im2col+GEMM algorithm, our method has better performance when the kernel size is large. However, when the kernel size is small, im2col+GEMM has better performance, because the unrolled matrix is small and the high performance of matrix multiplication routine in cuDNN outweighs the saving of algorithmic complexity in the fine-grained FFT method.

Conv. Layers	L1	L2	L3	L4	L5
Network1	(3, 10)	(3, 5)	(3, 8)	(3, 7)	(3, 10)
Network2	(3, 5)	(4, 10)	(3, 5)	(5, 5)	(3, 5)
Network3	(3, 10)	(3, 8)	(5, 5)	(3, 10)	(3, 5)

Table 3: Layer configurations for the three synthetic CNNs. Their performance evaluation is shown in Figure 9. Each element in the table indicates (U, K).

6.3 Performance in Applications:

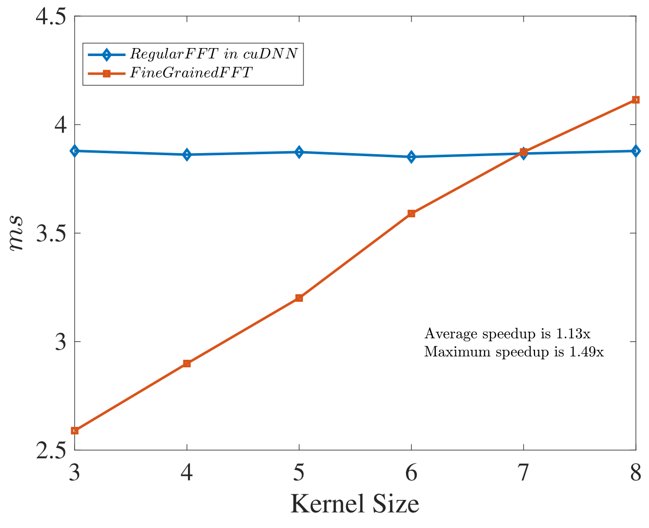
Caffe is one of the most popular frameworks that people use to develop deep-learning applications. In this experiment, we replace the convolution implementation in Caffe with our method, compose several CNNs with Caffe, and compare the performance before/after the replacement.

We compose three CNNs by five convolutional layers with parameter configurations listed in Table 3. The CNNs are inserted pooling and rectified linear units layers, and the last layer is a fully connected layer for prediction with 10 outputs. The inputs to these CNNs are $128 \times 128 \times 3$, $254 \times 254 \times 3$ and $254 \times 254 \times 3$ images with batch sizes of 5, 10 and 1, respectively. As it is shown in Figure 9, our fine-grained-FFT convolution outperforms the RegularFFT one in all configurations for one iteration of CNN inference. Specifically, it achieves speedups of 2.12×, 1.19× and 1.92× over the RegularFFT convolution, respectively. Additionally, our method perform faster for the layer-wise comparison, except for L2 in Network2.

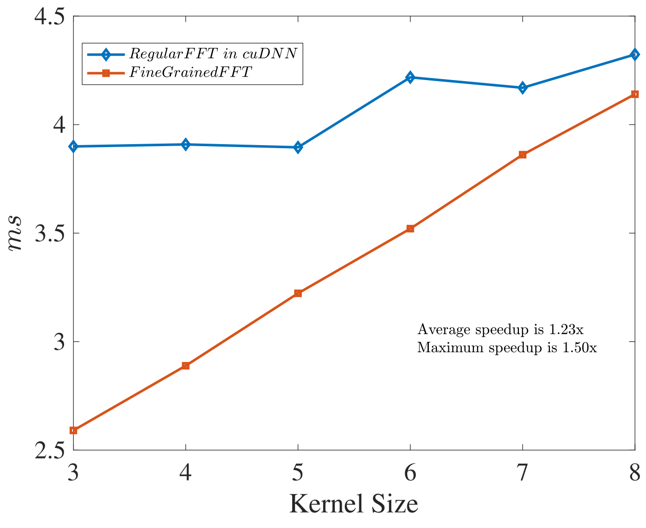
VGG-16 Benchmark:

We also evaluate the effectiveness of our method on a variant of VGG-16 [35]. VGG is frequently used as a CNNs benchmark because it makes improvement over prior-art configurations of CNNs by having 13 convolution layers of 3x3 filter. In our evaluation, we measure the performance of VGG-16 with on our FineGrainedFFT based convolution and that with the RegularFFT based convolution. We choose the width of convolution layers (the number of kernels) to be 38 and the batch size is set to 3. The FineGrainedFFT version takes 65.96 ms to run a forward-only inference, and the RegularFFT convolution based version takes 82.76 ms, a speedup of 1.25X.

To analyze the source of speedup in VGG-16, we conduct a layer-wise performance comparison of FineGrainedFFT vs. cuDNN's RegularFFT for the seven most time-consuming layers in VGG-16. The seven layers together account for about 60% of VGG-16's execution



(a) Synthetic benchmark



(b) ILSVRC2017 benchmark

Figure 7: RegularFFT and FineGrainedFFT performance comparison as the kernel size varies from 3 to 8.

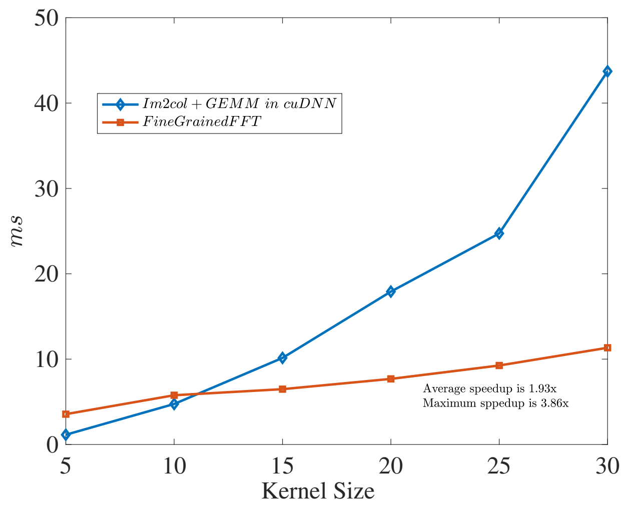


Figure 8: Im2col+GEMM and FineGrainedFFT performance comparison.

time. The layer-wise comparison is shown in Figure 10. Layer-by-layer benchmarked, FineGrainedFFT outperforms RegularFFT by the average speedup of 1.7 and the maximum speedup of 2.86.

6.4 Performance Analysis:

In order to empirically explain the performance gain of our algorithm, we use nvprof to profile the GPU kernels and compare the FFTs and element-wise multiplication execution time for each algorithm side by side in Table 4. This performance profiling provides a detailed performance breakdown and enable us to see which steps contribute most to the performance gain.

The FFT_r and MM_r stages have almost constant performance as kernel size increases because the kernel zero-pads to be the same size as the input feature maps, thus the amount of computations does not change. In contrast, the element-wise multiplication for the fine-grained-FFT convolution MM_f grows as the kernel size increases because our method is dependent on the im2col process; as kernel size increases the unrolled matrix becomes larger. FFT_f also tends to be a constant since the FFTs size is a power of two depending on the input size. In this case, it is 512. The matrix multiplication MM_r implementation is in fact batched matrix multiplications and perform transpositions to prepare tensors for matrix multiplications, which incurs extra operations. Thus MM_r is larger than MM_f except the last row in the table. Additionally, FFT_r and FFT_f almost have the same values. For 3×3 kernel, the speedup is 1.74×, which also means fine grained FFT convolution is 33% faster than the regular FFT one.

KernelSize	FFT_r	FFT_f	MM_r	MM_f	Speedup
3	0.732	0.784	3.272	1.510	1.74×
4	0.730	0.681	3.267	1.812	1.60×
5	0.725	0.770	3.271	2.200	1.34×
6	0.732	0.772	3.283	2.599	1.19×
7	0.728	0.648	3.267	2.848	1.14×
8	0.724	0.787	3.257	3.085	1.02×
9	0.691	0.764	3.249	3.307	0.96×

Table 4: Profiling results with varying kernel size. A subscript of r, f indicates RegularFFT and FineGrainedFFT convolutions. FFT and MM represent the FFTs and element-wise multiplication execution time. (unit ms)

7 CONCLUSIONS

In this paper, we have thoroughly analyzed how data redundancy is incurred in the im2col operation that is the foundation of today's high performance implementation of convolution. Our observation

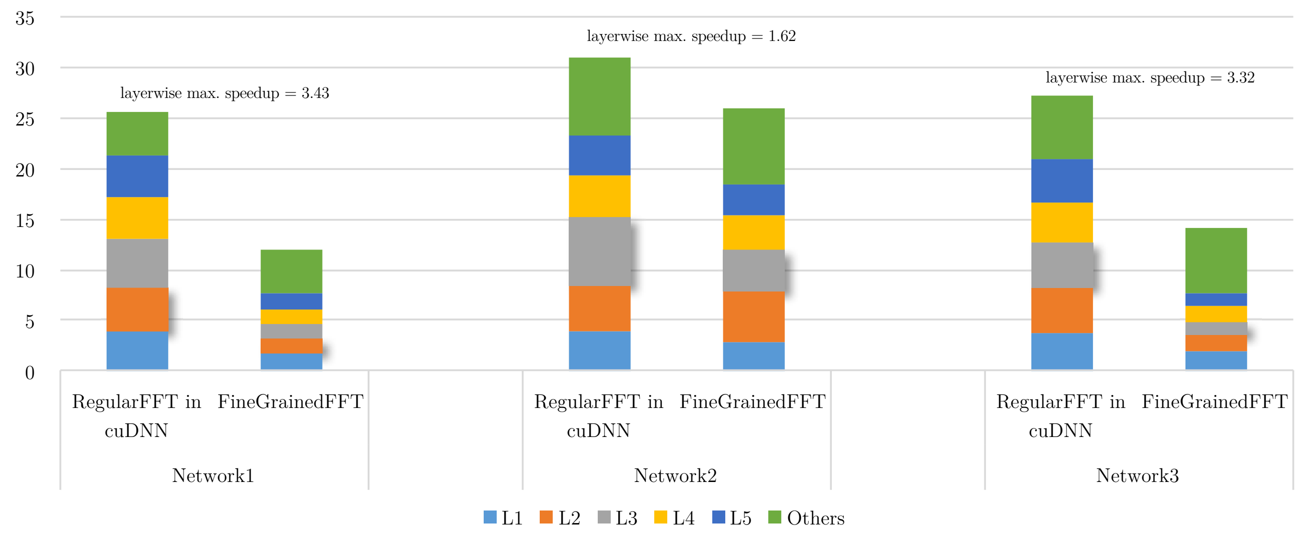


Figure 9: Layer-wise performance benchmark for the networks composed by five convolutional layers. Others in the legend represents pooling, ReLU and fully connected layers. Conv. layers with maximum speedup are highlighted as shaded rectangles. The average speedup for the three networks is 1.74×.

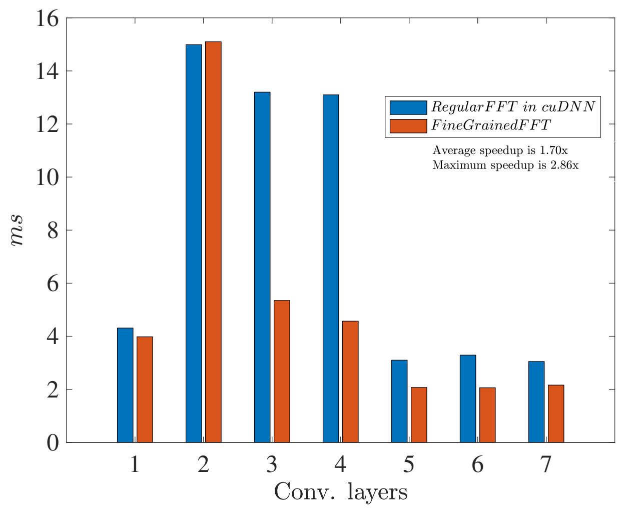


Figure 10: Layer-wise comparison of FineGrainedFFT vs. RegularFFT with the seven most time consuming layers in VGG-16.

and analysis lead to two novel discoveries: the intra-row redundancy and the inter-row redundancy in the matrix-based representation of convolution.

These discoveries motivate us to develop a more mathematically concise description of the data layout in convolution. We develop a doubly block Hankel matrix data pattern description. This unique data pattern enables us to design and implement a new fine-grained FFT-based convolution. This paper presents the theoretical arithmetic complexity analysis for both our fine-grained FFT convolution and the regular FFT convolution from NVIDIA's cuDNN library. The empirical results are consistent with the theoretical analysis. This fine-grained FFT convolution outperforms the regular FFT one in terms of speed in most parts of the parameter space of the convolutions. More specifically, compared with NVIDIA's cuDNN library, our method achieves on average 23% and maximum 50% speedup over its regular FFT convolution method, and on average 93% and maximum 286% speedup over its Im2col+GEMM method.

Our efforts add to a wide spectrum of convolution approaches in CNNs. Moreover, since there is no one "one-size-fits-all" convolution implementation across all the parameter space, our work raises the overall performance pareto-curve of convolution. One possible future work is to develop heuristics to select the fine-grained FFT convolution when the parameters, i.e., kernel sizes and batch sizes, are favorable.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI '16)*. USENIX Association, USA, 265–283.

[2] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. 2016. Theano: A Python framework for fast computation of mathematical expressions. *CoRR* abs/1605.02688 (2016). arXiv:1605.02688 http://arxiv.org/abs/1605.02688

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *CoRR* abs/1604.07316 (2016). arXiv:1604.07316 http://arxiv.org/abs/1604.07316

[4] Kumar Chellapilla, Sidd Puri, and Patrice Simard. 2006. High Performance Convolutional Neural Networks for Document Processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, Guy Lorette (Ed.). Université de Rennes 1, Suvisoft, La Baule (France). https://hal.inria.fr/inria-00112631 http://www.suvisoft.com.

[5] Chenyi Chen, Ari Seff, Alain L. Kornhauser, and Jianxiong Xiao. 2015. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving.

In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2722–2730. https://doi.org/10.1109/ICCV.2015.312

[6] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient Primitives for Deep Learning. *CoRR* abs/1410.0759 (2014). arXiv:1410.0759 http://arxiv.org/abs/1410.0759

[7] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. 2011. *Torch7: A Matlab-like Environment for Machine Learning*. Technical Report.

[8] Jason Cong and Bingjun Xiao. 2014. Minimizing Computation in Convolutional Neural Networks. In *Artificial Neural Networks and Machine Learning – ICANN 2014*. Springer International Publishing, Cham, 281–290.

[9] Christophe Garcia and Manolis Delakis. 2004. Convolutional Face Finder: A Neural Architecture for Fast and Robust Face Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 11 (2004), 1408–1423. https://doi.org/10.1109/TPAMI.2004.97

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press.

[11] Robert M Gray et al. 2006. Toeplitz and circulant matrices: A review. *Foundations and Trends® in Communications and Information Theory* 2, 3 (2006), 155–239.

[12] Scott Gray. 2014. Maxas: Assembler for nvidia maxwell architecture.

[13] Junli Gu, Yibing Liu, Yuan Gao, and Maohua Zhu. 2016. OpenCL caffe: Accelerating and enabling a cross platform machine learning framework. In *Proceedings of the 4th International Workshop on OpenCL, IWOCL 2016, Vienna, Austria, April 19-21, 2016*. ACM, 8:1–8:5. https://doi.org/10.1145/2909437.2909443

[14] Stefan Hadjis, Firas Abuzaid, Ce Zhang, and Christopher Ré. 2015. Caffe con Troll: Shallow Ideas to Speed Up Deep Learning. In *Proceedings of the Fourth Workshop on Data analytics in the Cloud, DanaC 2015, Melbourne, VIC, Australia, May 31 - June 4, 2015*, Asterios Katsifodimos (Ed.). ACM, 2:1–2:4. https://doi.org/10.1145/2799562.2799641

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90

[16] Tyler Highlander and Andres Rodriguez. 2016. Very Efficient Training of Convolutional Neural Networks using Fast Fourier Transform and Overlap-and-Add. *CoRR* abs/1601.06815 (2016). arXiv:1601.06815 http://arxiv.org/abs/1601.06815

[17] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015 (JMLR Workshop and Conference Proceedings)*, Francis R. Bach and David M. Blei (Eds.), Vol. 37. JMLR.org, 448–456. http://proceedings.mlr.press/v37/ioffe15.html

[18] Yangqing Jia. 2014. *Learning Semantic Image Representations at a Large Scale*. Ph.D. Dissertation. University of California, Berkeley, USA. http://www.escholarship.org/uc/item/64c2v6sn

[19] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, Kien A. Hua, Yong Rui, Ralf Steinmetz, Alan Hanjalic, Apostol Natsev, and Wenwu Zhu (Eds.). ACM, 675–678. https://doi.org/10.1145/2647868.2654889

[20] Alex Krizhevsky. 2012. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. *Source code available at https://github.com/akrizhevsky/cuda-convnet2 [March, 2017]* 7 (2012).

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[22] Andrew Lavin. 2015. maxDNN: An Efficient Convolution Kernel for Deep Learning with Maxwell GPUs. *CoRR* abs/1501.06633 (2015). arXiv:1501.06633 http://arxiv.org/abs/1501.06633

[23] Andrew Lavin and Scott Gray. 2016. Fast Algorithms for Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 4013–4021. https://doi.org/10.1109/CVPR.2016.435

[24] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature Cell Biology* 521, 7553 (27 may 2015), 436–444. https://doi.org/10.1038/nature14539

[25] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1, 4 (1989), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2323. https://doi.org/10.1109/5.726791

[27] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. 2010. Convolutional networks and applications in vision. In *International Symposium on Circuits and Systems (ISCAS 2010), May 30 - June 2, 2010, Paris, France*. IEEE, 253–256. https://doi.org/10.1109/ISCAS.2010.5537907

[28] Xiaqing Li, Guangyan Zhang, H. Howie Huang, Zhufan Wang, and Weimin Zheng. 2016. Performance Analysis of GPU-Based Convolutional Neural Networks. In *45th International Conference on Parallel Processing, ICPP 2016, Philadelphia, PA, USA, August 16-19, 2016*. IEEE Computer Society, 67–76. https://doi.org/10.1109/ICPP.2016.15

[29] M Donald MacLaren. 1970. The art of computer programming. Volume 2: Seminumerical algorithms (Donald E. Knuth). *SIAM Rev.* 12, 2 (1970), 306–308.

[30] Michaël Mathieu, Mikael Henaff, and Yann LeCun. 2014. Fast Training of Convolutional Networks through FFTs. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1312.5851

[31] Hyunsun Park, Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. 2016. Zero and data reuse-aware fast convolution for deep neural networks on GPU. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES 2016, Pittsburgh, Pennsylvania, USA, October 1-7, 2016*. ACM, 33:1–33:10. https://doi.org/10.1145/2968456.2968476

[32] J.R. Partington. 1988. *An Introduction to Hankel Operators*. Cambridge University Press. https://books.google.com/books?id=bG03AAAAIAAJ

[33] Hugh Perkins. 2016. cltorch: a Hardware-Agnostic Backend for the Torch Deep Neural Network Library, Based on OpenCL. *CoRR* abs/1606.04884 (2016). arXiv:1606.04884 http://arxiv.org/abs/1606.04884

[34] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. 2014. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1312.6229

[35] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556

[36] Volker Strassen. 1969. Gaussian Elimination is Not Optimal. *Numer. Math.* 13, 4 (aug 1969), 354âĂŞ356. https://doi.org/10.1007/BF02165411

[37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[38] Nicolas Vasilache, Jeff Johnson, Michaël Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. 2015. Fast Convolutional Nets With fbfft: A GPU Performance Evaluation. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.7580

[39] Shmuel Winograd. 1980. *Arithmetic Complexity of Computations*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611970364 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611970364

[40] Dong Yu, Adam Eversole, Michael L. Seltzer, Kaisheng Yao, Brian Guenter, Oleksii Kuchaiev, Frank Seide, Huaming Wang, Jasha Droppo, Zhiheng Huang, Geoffrey Zweig, Christopher J. Rossbach, and Jon Currey. 2014. An introduction to computational networks and the computational network toolkit (invited talk). In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, Haizhou Li, Helen M. Meng, Bin Ma, Engsiong Chng, and Lei Xie (Eds.). ISCA. http://www.isca-speech.org/archive/interspeech_2014/i14_4001.html