Hardware-efficient convolution algorithms for CNN accelerators: A brief review

Ricardo Di Curzio Lera[1], **Bruno de Carvalho Albertini**[2]

[1]Centro de Ciência dos Dados (C2D) – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

[2]Departamento de Engenharia de Computação – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

ricardodclera@usp.br, balbertini@usp.br

Abstract. *The Convolutional Neural Network (CNN) is a technology of vast importance in image processing and computer vision applications. The bottleneck of CNNs is the multidimensional convolution, which often demands accelerator hardware. The convolution algorithms these accelerators use directly affect the ratio between speed increase and hardware resource consumption during scaling, a metric known as hardware efficiency. The lower this metric, the more power and area are spent on minor performance improvements. In this review, we analyze the potential for hardware efficiency in the current proven algorithms used in convolutional layers:* `im2col` *convolution used by most modern applications, Toom-Cook convolution, and FFT convolution. Our analysis reveals the inefficiency of* `im2col` *convolution regarding hardware scaling and confirms the potential for hardware-efficient applications using Toom-Cook and FFT convolutions, each with its caveats. Further, we identify possible hardware applications for these algorithms, which may be expanded upon in future works.*

Keywords*: convolutional neural network, CNN, hardware efficiency, TPU, im2col, Toom-Cook, Winograd, FFT, GEMM*

1. Introduction

Convolutional Neural Networks (CNNs) have a long history of constant development and adaptation to advancements in machine learning. The original LeNet-5 implementation [LeCun et al. 1998] constituted the basis of the CNN architecture: pairs of convolutional and subsampling layers (e.g., Max Pooling) followed by a set of fully connected layers.

With recent advancements in computational processing power in the form of GPUs, CNNs have become the state-of-art for image processing and computer vision, starting with AlexNet [Krizhevsky et al. 2012], which uses a similar architecture to LeNet-5, adding features such as Dropout, Normalization, and ReLU. Since then, architectures utilizing small kernels (e.g., 3x3, stride=1) across multiple layers have been shown to keep their large receptive field intact while severely reducing the number of parameters required [Simonyan and Zisserman 2014].

However, CNN computation is notoriously complex, traditionally requiring massive amounts of multiplication and addition operations which are not

easily parallelized by common architectures and consume large amounts of memory [Moolchandani et al. 2021], increasing the demand for accelerator architectures.

Modern CNN accelerators such as Google's TPU [Jouppi et al. 2020] and NVIDIA's Tensor Core [Choquette et al. 2021] are capable of substantial speed improvements, leveraging the parallelization of matrix multiplication algorithms and systolic array architectures.

Beyond the commonly studied metrics, however, there is the matter of *hardware efficiency*, defined as the ratio between speed and hardware utilization (i.e., area, components). This metric is critical to the process of scaling hardware, as it measures the speed increase derived from the hardware investment. Li et al. (2022) demonstrated the importance of hardware efficiency in CNN accelerators via their implementation in FPGA technology. A more hardware-efficient application can be scaled to perform better than a less hardware-efficient one at the same hardware cost. Alternatively, it can be scaled *down* to be applied to resource-constrained systems.

While the algorithms that govern the basic operations of modern CNN accelerators are highly optimized, the underlying convolution algorithms are still under active research and are what govern the raw number of operations that must be performed. A larger number of required operations usually equates to a longer processing time but can be compensated with hardware scaling. Thus, an algorithm requiring fewer operations will require less hardware to sustain speed.

In this review, we analyze the potential for hardware efficiency in the current algorithms for convolution processing. Section 2 instantiates and compares the relevant algorithms, Section 3 discourses on significant aspects of each algorithm relating to hardware efficiency, and Section 4 describes our conclusions and comments on conceivable future developments.

2. Convolution Algorithms

We can characterize two parameters inherent to an algorithm that significantly impact hardware efficiency: *complexity* and *sequentiality*.

Complexity relates to the total number of *elementary operations*[1] that must be performed to complete the task. It is often displayed in *Big O* notation, but asymptotic notation such as this also hides the information about the algorithm's early growth. The notations used depend on the application.

Sequentiality relates to the presence of dependency chains in the algorithm. Given sufficient memory and components, independent operations can be performed simultaneously in hardware. This is known in hardware design as *parallelization* and is a core facet of virtually every current processor to some degree. Low sequentiality in an algorithm enables parallelization.

Other metrics exist, such as an algorithm's memory footprint, which are essential

[1] An *elementary operation* is defined as a computing operation that takes a fixed amount of time to be processed by a particular computing machine. Consequentially, the set of elementary operations used for complexity analysis depends on the model of computation used, as different abstract and physical machines will have widely different sets of instructions. For a fair comparison between different architectures, this project will refer to any *single-bit arithmetic or logical operations* as elementary operations.

for architecture design but are not as discrepant between the relevant convolution algorithms, and thus not critical to this analysis.

2.1. Convolution-over-Space

Signal convolution over sets of Euclidean-organized data (e.g., arrays, images) can be represented by the "sliding product" of two inputs. The filter (or *kernel*) "slides" across the input, multiplying each pair of coincident points and adding the products together. Each of these results represents a convolution centered on a different input point. All of them are organized jointly as the output of the convolution. In image processing, we refer to this operation as "convolution-over-space".

The output can have varying sizes depending on how we handle the sliding of the kernel out-of-bounds [Alzubaidi et al. 2021]:

- If we ignore every position of the kernel in which any of its points coincide with out-of-bounds data: the output size is smaller, proportional to the kernel's size, and referred to as *valid* convolution.
- If we require only the center point of the kernel to coincide with valid data and the out-of-bounds data equal to zero: the output size is the same as the input, referred to as *same* convolution.
- If we consider every position of the kernel that coincides with at least one valid point of data and the out-of-bounds data equal to zero: the output size is larger, proportional to the kernel's size, and is referred to as *full* convolution.

Let s be the size of the output of a single-dimension full convolution such that $s = i + k - 1$, where i is the dimension of the input, and k is the dimension of the kernel. The number of multiplication operations required by full convolution-over-space is $s.k$. This becomes problematic in higher dimensions, where the number of multiplications equals the input's total size times the kernel's. For instance, convolving a 252x252 image with three color channels by a 3x3 kernel, also with three color channels, would require $252^2.3^4$ or about 5 million multiplications and additions.

2.2. `im2col` Convolution

Since convolution is essentially a matrix operation, it is possible to arrange the input and kernel into matrices such that their matrix multiplication becomes equivalent to their convolution [Vasudevan et al. 2017]. We can use set theory to define this property independently of any particular algorithm:

$$\forall[i \in \mathbb{R}^a, k \in \mathbb{R}^b] \; \exists[I_{m\times n} \subseteq i, K_{n\times p} \subseteq k] \mid I_{m\times n}K_{n\times p} = i * k \tag{1}$$

where $*$ denotes the convolution operator, and $a, b, m, n, p \in \mathbb{R}$. The $\subseteq$ operator refers to *improper subsets*, which in this context denotes permutation (e.g., $I_{m\times n} \subseteq i$ denotes a matrix I which contains all elements, and only the elements, of the vector i). This property can be expanded for higher dimensions of the inputs, generating wider matrices.

Note that Equation 1 is written for *valid* convolution in one dimension but can be expanded to *valid* and *full* as well as to higher dimensions by adjusting the sizes of its inputs and matrices. This article will evaluate all algorithms as full convolution for consistency.

Converting convolution into matrix multiplication enables algorithms such as BLAS' General Matrix-to-Matrix Multiplication (GEMM) [Kågström et al. 1998], which can also benefit from parallelization. Zhou et al. (2021) implemented the first open-source implicit `im2col` algorithm of the same type used by GEMM-based accelerators (such as the TPU and Tensor Core) to convert inputs and kernels into matrices. Using this algorithm followed by GEMMs, while highly optimized for hardware, **does not** reduce the number of operations required to complete a convolution. As such, the number of multiplications required by the IK matrix multiplication (in full convolution) always equals $s.k$. In the case of multiple dimensions, channels, and filters, this number becomes the total size of the input times the total size of all kernels.

GEMM-based hardware accelerators use *systolic array* technology, a topological assortment of processing units commonly referred to as *nodes*. Each node is designed to process fractions of the desired operation and send information to each neighbor, forming a *stream* that eventually leads to a complete operation. In particular, the input of the systolic array can be shared between nodes, removing the need to re-read the input value from a register. As such, computationally complex operations can be subdivided and performed within a single machine cycle, provided the array is filled with data from the stream. In the TPU and Tensor Core accelerators, the matrix multiplication is subdivided into multiply-add operations, and tens of thousands of these operations are calculated per cycle, leading to a significant decrease in latency.

2.3. Toom-Cook Convolution

Lavin & Grey, in their 2016 seminal paper, defined the *fast algorithms for convolution* based on the theory of Finite Impulse Response (FIR) filters, assisted by Winograd's algorithms. The resulting algorithm is known as Toom-Cook convolution[2] and it can be used in two distinct ways.

The core of the algorithm samples the input and kernel at a given set of points using transform matrices. In this way, the Hadamard product (element-wise multiplication) of the transformed signals, followed by the inverse transform, is equal to the convolution of the original signals, as shown by the equation:

$$Y = A^T \left[\left[GgG^T \right] \odot \left[B^T dB \right] \right] A \tag{2}$$

where g and d are the signals; A, B, and G are transform matrices; Y is the resulting convolution in matrix form; and $\odot$ represents the Hadamard product.

In short, Equation 2 uses transformations to convert convolution into pointwise multiplication. The sizes of the input and kernel are critical factors in this method's efficiency, so the input is usually divided into *blocks* of specific sizes. Each block is convolved with the kernel, and the results are added together. Toom-Cook convolution gains its efficiency by computing multiple outputs simultaneously [Alam et al. 2022].

[2]Lavin & Grey originally referred to this algorithm as Winograd convolution; however, recent sources [Barabasz et al. 2020, Alam et al. 2022] have disputed this term as the article uses the Toom-Cook method, which is closely related but distinct from Winograd. This article will use the term Toom-Cook convolution for consistency.

As the input block size *increases*, the number of multiplications required by the Hadamard product *decreases* to a theoretical minimum of s. However, the number of additions and constant multiplications required by the transform grows quadratically with the *size of the convolution* (block + kernel - 1) [Barabasz et al. 2020], as does the floating-point precision loss due to the magnitude of these constants [Barabasz and Gregg 2019]. Therefore, to minimize the convolution's size while maximizing block size, the *size of the kernel* has to be lower. As such, Toom-Cook convolution is more efficient in applications with very small kernels, which is convenient for modern CNNs, as they tend to have many layers with small kernels.

However, Lavin & Grey (2016) went one step further. CNNs are usually processed with a third dimension, the channels, as well as multiple kernels. We label the channels c, the number of channels C, the blocks p, the number of blocks P, the number of kernels K, and the input block size m. If we now label the resulting transform matrices $U = GgG^T$ and $V = B^T dB$, we can scatter U and V such that their Hadamard product is equivalent to $(m+k-1)(m+k-1)$ matrix multiplications of type $(K \times C)(C \times P)$. Substituting in Equation 2 we have:

$$Y_{f,t} = A^T \left[U_{k,c}^{(\xi,\nu)} V_{c,p}^{(\xi,\nu)} \right] A \tag{3}$$

which maps Toom-Cook convolution to matrix multiplications across multiple channels and kernels, simultaneously amortizing the cost of the inverse transform A and allowing the use of GEMM operations. Both approaches have applications depending on the CNNs in question, and both significantly impact the operation's complexity.

2.3.1. Complexity of Matrix Multiplication

Lavin & Grey (2016) also pointed out the possibility of applying algorithms for complexity reduction of matrix multiplication to CNNs. Such algorithms have been studied since at least 1969 with the Strassen Algorithm [Strassen et al. 1969]. By reducing the complexity of the matrix multiplication itself, the scaling of a convolution accelerator would cost less hardware, thus increasing hardware efficiency. Additionally, these algorithms must be capable of parallelism and not incur other caveats that overcome these benefits.

Finding the matrix multiplication algorithm with the smallest possible asymptotic complexity is an open problem in modern mathematics. In technical terms, the problem is defined as "finding the smallest ω such that the number of operations required by an $N \times N$ matrix multiplication equals $N^{\omega+O(1)}$" [Gao et al. 2020]. The standard matrix multiplication algorithm (i.e., the naive method derived from the definition of matrix multiplication) has $\omega = 3$, and the current lowest known value for ω is approximately 2.37 [Duan et al. 2022] using a method based on the Coppersmith-Winograd algorithm.

It may be possible to apply these algorithms, particularly Strassen due to its relative simplicity, to reduce the complexity of various methods cited in this project. BLAS has implemented Strassen on some of its algorithms, but GEMM still uses standard matrix multiplication.

2.4. FFT Convolution

The Theorem of Convolution defines the operation as follows:

$$[g * h](x) \Leftrightarrow F^{-1}[G \odot H](y) \tag{4}$$

where G and H are the *Fourier Transforms* of the respective signals g and h; F^{-1} is the *Inverse Fourier Transform*; $*$ denotes convolution; x denotes an arbitrary domain and y denotes the respective Fourier domain of x.

The theorem states that signal convolution in a particular domain is equivalent to pointwise multiplication in its Fourier domain. In a manner similar to Toom-Cook convolution, the theorem allows us to compute convolution by calculating the Discrete Fourier Transforms (DFT) of the input and kernel in the domain of space (represented by σ), applying the Hadamard product in the respective Fourier domain of space (represented by ζ), and then calculating the Inverse Discrete Fourier Transform (IDFT) as such:

$$[i * k](\sigma) = F^{-1}[F(i) \odot F(k)](\zeta) \tag{5}$$

The traditional DFT (matrix multiplication between the signal and a twiddle factor matrix) has complexity $O(s^2)$. Instead, we can use the Fast Fourier Transform (FFT) algorithm, which exploits the symmetry in the Fourier Transform and has complexity $O(s.log_2 s)$. The same relationship applies to the IDFT and IFFT (Inverse Fast Fourier Transform).

The FFT convolution scales almost linearly with the image's size, significantly increasing processing speed when using larger kernels [Jha 2007]. Furthermore, it is possible to subdivide the input into kernel-sized pieces, perform FFT convolution in each one, and add the overlapping values, resulting in the full convolution of the undivided input [Highlander and Rodriguez 2016]. This method, known as *overlap-add*, reduces the asymptotic complexity of the operation to $O(s.log_2 k)$.

However, the arrays transformed by the FFT are in the *Complex Domain*[3] and thus require Complex operations. Most prominently, the Hadamard product becomes a series of Complex products, usually calculated as four Real products and two additions but can be simplified to three Real products using a less-known method [Lavin and Gray 2016]. Regardless, the Complex product can drastically increase computational requirements depending on the hardware used.

2.5. Comparison

Table 1 summarizes the analyzed algorithms regarding the number of multiplications required by the core algorithm and their transform complexities, considering a single dimension. These values can be expanded to two or more dimensions by increasing the exponent of s and k.

[3]Whenever the word "complex" refers to the Complex Domain ($\mathbb{C}$), we shall capitalize it to differentiate it from the concept of *complexity* used in other parts of the text, for the sake of cohesion. Similarly, the word "real" is capitalized when referring to the Real Domain ($\mathbb{R}$).

Table 1. Computational complexity of convolution algorithms.

Convolution Algorithm	Multiplication Operations	Transform Complexity
Convolution-over-Space	$s.k$	-
`im2col` Convolution	$s.k$	Unknown/Undisclosed
Toom-Cook Convolution	$\geqslant s$	$O(s^2)^*$
FFT Convolution	s ($\mathbb{C}$)	$O(s.log_2 k)^*$

The transform complexities of Toom-Cook and FFT convolution (marked with an asterisk in Table 1) depend on the size of the array they are applied to. If a transform is applied to a block subdivided from the input, the complexity of this operation is lower, but the total complexity still depends on the input size s. In Toom-Cook convolution, the total complexity remains at $O(s^2)$ [Barabasz et al. 2020], while in FFT convolution, the overlap-add method reduces the complexity of the logarithmic term to $O(s.log_2 k)$ [Highlander and Rodriguez 2016].

3. Analysis

3.1. Matrix Multiplication

Current accelerators rely heavily on systolic array technology, which itself relies on the `im2col` algorithm. We can infer that if latency is to be preserved, the number of nodes of a systolic array is directly proportional to the number of operations required. For instance, if the number of operations required is doubled, the systolic array must contain about twice the number of nodes to preserve latency.

Recall that the number of operations required by convolution-over-space is preserved in its respective matrix multiplication through the use of `im2col`. Furthermore, while modern accelerators rely on very optimized algorithms (e.g., BLAS), these algorithms are primarily optimized in terms of processor hardware (e.g., efficient use of instructions and registers; parallelism) and rarely use unconventional algorithms such as Strassen. Thus, they retain the complexity of standard matrix multiplication.

This reveals a conflict between systolic array technology and the convolution operation: convolution-over-space has a high complexity that grows proportional to the dimensions of its inputs (e.g., quadratically for 2D inputs, cubically for 3D inputs), and the systolic array's hardware requirements have to grow directly proportionally to this complexity in order to maintain latency.

3.2. Toom-Cook Convolution

The Hadamard approach to Toom-Cook convolution is inefficient in CNNs with many channels and kernels due to the number of inverse transforms required. By using the matrix multiplication approach and amortizing the cost of the inverse transform over the channels and kernels, the core multiplication complexity may increase, but the overall complexity decreases by a more significant amount.

This method was demonstrated by Kala et al. (2019) by implementing a GEMM-based accelerator in FPGA using the Toom-Cook convolution algorithm. Their

implementation achieved a performance improvement of 1.40x to 4.02x with only 13% more hardware resources compared to Shen et al. (2018) systolic array matrix multiplier.

This comparison exemplifies the importance of hardware efficiency in modern hardware design. A single Toom-Cook convolution "core" provides lower throughput than a systolic array in a single cycle. However, when more cores are added and thus the hardware is scaled, Toom-Cook overcomes performance at an increased rate due to its hardware efficiency. This characteristic is valid not only for programmable hardware (i.e., FPGAs) but also for ASICs.

3.3. FFT Convolution

The low complexity of the current Fourier Transform algorithms and the relatively small Hadamard product make FFT convolution a promising solution. Its most significant drawback is its reliance on Complex operations, which severely increases the cost of multiplication and is not generally supported by common architectures.

A possible solution would be to employ the polar representation of Complex numbers in which multiplication is equivalent to one Real multiplication (the amplitude) and one Real addition (the phase), as such:

$$A\angle\alpha_1 \cdot B\angle\alpha_2 = AB\angle\alpha_1 + \alpha_2 \tag{6}$$

The conversion of rectangular to polar coordinates usually requires trigonometric operations, but there exist algorithms with mature hardware implementations that can execute such operations quickly and hardware-efficiently, such as the Coordinate Rotation Digital Computer (CORDIC) [Meher et al. 2009].

A more extreme but very efficient idea is the *Fourier Convolutional Neural Network* (FCNN) [Pratt et al. 2017] in which the Fourier transform happens at the start of the network, and the entire system is processed in the Fourier domain, essentially turning every convolutional layer into a multiplication. This effectively nullifies the transform latency (which could perhaps lead to a similar approach for Toom-Cook convolution), but FCNNs are limited by the need to describe every layer of the network in the Fourier domain, which do not necessarily perform adequately [Han and Hong 2021].

FFT convolution is also more efficient for larger kernels than Toom-Cook, as it has a less complex transform. Application-constrained systems might benefit from this advantage; for instance, extremely-low-latency applications often use CNNs with fewer layers to reduce sequentiality, which incurs larger filters to maintain accuracy.

We could not find published results about FFT-based CNN accelerators which have explicitly measured hardware efficiency. However, recent attempts at such accelerators have shown remarkable improvements in energy-efficiency [Lee et al. 2020] and hardware resource utilisation [Chitsaz et al. 2020].

4. Conclusions & Future Work

This project provided a short review of the most relevant convolution algorithms used today from the point of view of their expected hardware efficiency.

We based our analysis on the numerical definitions of the algorithms and supported it with the experimental results of recent works in the field. Using this method, we could derive valuable conclusions about the applicability of the three most used algorithms in convolution processing, responsible for considerable computational power demand in CNNs.

Nevertheless, we cannot directly measure hardware efficiency using disconnected experimental results performed in distinct platforms. For an accurate measurement of this nature, we suggest, as future work, directly implementing each algorithm in FPGA technology.

The following list summarizes the key points addressed in this project:

- **Accelerators such as the TPU are fast but hardware-inefficient** due to the large inherent complexity of the convolution operation, which is conserved to the matrix multiplication through `im2col`, and thus scales the systolic array at the same rate.
- **Toom-Cook is the most promising solution for standardized hardware**, as it only requires Real operations. It provides a way to map large CNNs into fewer GEMMs, increasing hardware efficiency. The transform limits the application to small kernels and requires amortizing the cost of the transform over many channels and kernels, but current CNNs are well suited for these limitations.
- **FFT convolution is a promising solution for specialized hardware**, as its reliance on Complex numbers can be amortized using targeted hardware-based solutions, such as CORDIC, implemented on FPGAs or ASICs. The fast transforms and Hadamard product inherent to the algorithm could overcome Toom-Cook in hardware efficiency. FFT convolution is also more suitable in environments that require larger kernels.

Based on our conclusions, our next recommendation for future works is the implementation of an FFT-convolution-specialized architecture in FPGA or ASIC, using CORDIC (or similar technologies) to amortize the cost of complex multiplication. Another possible research topic would be identifying applications that require large kernels and implementing FFT convolution as a hardware-efficient solution.

Finally, we must emphasize the importance of a metric such as hardware efficiency in a time when hardware scaling is both necessary and often wasteful. We expect the use of such metrics to increase, as hardware design paradigms slowly shift from the performance-centered stance we see today.

References

Alam, S. A., Anderson, A., Barabasz, B., and Gregg, D. (2022). Winograd convolution for deep neural networks: Efficient point selection. *ACM Transactions on Embedded Computing Systems*, 21(6):1–28.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., and Farhan, L. (2021). Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8:1–74.

Barabasz, B., Anderson, A., Soodhalter, K. M., and Gregg, D. (2020). Error analysis and improving the accuracy of winograd convolution for deep neural networks. *ACM Transactions on Mathematical Software (TOMS)*, 46(4):1–33.

Barabasz, B. and Gregg, D. (2019). Winograd convolution for dnns: Beyond linear polynomials. In *AI* IA 2019–Advances in Artificial Intelligence: XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19–22, 2019, Proceedings*, pages 307–320. Springer.

Chitsaz, K., Hajabdollahi, M., Karimi, N., Samavi, S., and Shirani, S. (2020). Acceleration of convolutional neural network using fft-based split convolutions. *arXiv preprint arXiv:2003.12621*.

Choquette, J., Gandhi, W., Giroux, O., Stam, N., and Krashinsky, R. (2021). Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35.

Duan, R., Wu, H., and Zhou, R. (2022). Faster matrix multiplication via asymmetric hashing. *arXiv preprint arXiv:2210.10173*.

Gao, J., Ji, W., Chang, F., Han, S., Wei, B., Liu, Z., and Wang, Y. (2020). A systematic survey of general sparse matrix-matrix multiplication. *ACM Computing Surveys*.

Han, Y. and Hong, B.-W. (2021). Deep learning based on fourier convolutional neural network incorporating random kernels. *Electronics*, 10(16):2004.

Highlander, T. and Rodriguez, A. (2016). Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add. *arXiv preprint arXiv:1601.06815*.

Jha, N. K. (2007). Ieee transactions on very large scale integration (vlsi) systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(3):249.

Jouppi, N. P., Yoon, D. H., Kurian, G., Li, S., Patil, N., Laudon, J., Young, C., and Patterson, D. (2020). A domain-specific supercomputer for training deep neural networks. *Communications of the ACM*, 63(7):67–78.

Kågström, B., Ling, P., and Van Loan, C. (1998). Gemm-based level 3 blas: high-performance model implementations and performance evaluation benchmark. *ACM Transactions on Mathematical Software (TOMS)*, 24(3):268–302.

Kala, S., Jose, B. R., Mathew, J., and Nalesh, S. (2019). High-performance cnn accelerator on fpga using unified winograd-gemm architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2816–2828.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS'2012)*, pages 1106—-1114.

Lavin, A. and Gray, S. (2016). Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4013–4021.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324.

Lee, Y.-C., Chi, T.-S., and Yang, C.-H. (2020). A 2.17-mw acoustic dsp processor with cnn-fft accelerators for intelligent hearing assistive devices. *IEEE Journal of Solid-State Circuits*, 55(8):2247–2258.

Li, X., Huang, H., Chen, T., Gao, H., Hu, X., and Xiong, X. (2022). A hardware-efficient computing engine for fpga-based deep convolutional neural network accelerator. *Microelectronics Journal*, 128:105547.

Meher, P. K., Valls, J., Juang, T.-B., Sridharan, K., and Maharatna, K. (2009). 50 years of cordic: Algorithms, architectures, and applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(9):1893–1907.

Moolchandani, D., Kumar, A., and Sarangi, S. R. (2021). Accelerating cnn inference on asics: A survey. *Journal of Systems Architecture*, 113:101887.

Pratt, H., Williams, B., Coenen, F., and Zheng, Y. (2017). Fcnn: Fourier convolutional neural networks. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part I 17*, pages 786–798. Springer.

Shen, J., Qiao, Y., Huang, Y., Wen, M., and Zhang, C. (2018). Towards a multi-array architecture for accelerating large-scale matrix multiplication on fpgas. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Strassen, V. et al. (1969). Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356.

Vasudevan, A., Anderson, A., and Gregg, D. (2017). Parallel multi channel convolution using general matrix multiplication. In *2017 IEEE 28th international conference on application-specific systems, architectures and processors (ASAP)*, pages 19–24. IEEE.

Zhou, Y., Yang, M., Guo, C., Leng, J., Liang, Y., Chen, Q., Guo, M., and Zhu, Y. (2021). Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pages 214–225. IEEE.