2025 38th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI) | 979-8-3315-5555-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/SBCCI66862.2025.11218694

Fast and Energy-Efficient 2D Convolution: Inspection-Based Methods for Hardware Acceleration

Társio Onofrio da Silva and Fernando Gehm Moraes
School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil
tarsio.onofrio@edu.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—This paper revisits Winograd's inspection-based factorization (IF) method for generating hardware-efficient 2D convolution accelerators. The objective is to reduce the number of multipliers, execution time, and energy consumption without increasing area. First, we apply IF to reduce the total number of multiplications compared to a naïve implementation. Unlike Toom-Cook 3 (TC), IF does not require any constant-factor multiplications. Second, we propose a parameterizable RTL architecture that leverages IF to instantiate only a subset of multipliers, rather than mapping each multiplication directly to dedicated hardware. This approach avoids area overhead while maintaining performance, achieving a balanced trade-off between resource utilization and throughput. Finally, we synthesized the designs using 28 nm technology at 500 MHz. We compared our approach with fast convolution methods previously employed in CNN accelerators. Compared to TC, IF reduces energy consumption by 19% and silicon area by 9% while delivering equivalent performance. Compared to the Winograd minimal filter, IF improves performance by 49% and reduces energy consumption by 24%. These results demonstrate that IF is a promising alternative for CNN hardware on power- and performance-constrained platforms.**

***Index Terms*—Fast Convolution, Inspection-based, Winograd, Toom-Cook, hardware acceleration.**

I. Introduction

Convolutional neural networks (CNNs) are increasingly adopted in computer vision tasks such as object detection, image classification, and semantic segmentation. Their widespread success stems from the ability to automatically learn salient image features, outperforming traditional approaches on numerous benchmarks [1, 2]. The integration of CNNs into resource-constrained environments such as edge-constrained devices has become increasingly relevant [3]. These scenarios demand low-latency, energy-efficient inference, often under strict hardware limitations. At the same time, the increased depth of these architectures comes with increased computational demands, driven primarily by the extensive multiply-accumulate (MAC) operations within convolutional layers. As a result, accelerating the execution time of these layers is critical to improving the performance of CNN-based systems [4]–[6].

A fundamental part of optimizing CNNs lies in how convolution is performed. Conventional methods map convolution to matrix multiplications or execute it directly in the spatial domain. Although efficient, these approaches do not reduce the number of multiplications. On the other hand, fast convolution algorithms, such as Winograd and FFT [7]–[10], have been increasingly investigated for significantly reducing the computational load, especially for small filters (3×3) found in popular networks [11]. Implementing CNNs in hardware offers challenges, requiring the mapping of the hardware resources and an effective balance between data throughput and computational capacity to avoid memory bottlenecks. Exploring parameters such as the number of processing elements, tile sizes, and pipeline depth can affect the efficiency of the design [12]–[14].

Except for the FFT, all proposals that implement fast convolution techniques in hardware employ either the Winograd Minimum Filter (Toom-Cook 2.5) or the Toom-Cook 3, and Nested as the multidimensional method [12, 15]–[19]. Toom-Cook 3 presents higher throughput but increases the silicon area due to a larger number of additions and constant multiplications [12]. On the other hand, the Winograd Minimum Filter reduces power and area but sacrifices throughput. The Nested method is generally presented using the equation proposed by Lavin et al. [20].

Our work *proposes* an alternative to established fast-convolution techniques by integrating Winograd's inspection-based factorization into our accelerator design. This approach effectively balances the PPA targeting resource-constrained devices. We validate these advantages through a comparative study against the standard (naïve) convolution, the Winograd Minimal Filtering, and Toom-Cook 3 algorithms. Results demonstrate that our IF-enabled accelerator delivers area and energy savings with the same performance as the Toom-Cook 3 implementation.

II. Background Knowledge

The naïve convolution computation for two sequences, d and g produces s (Table I). If d and g have the same width, W, then $K = W^2$. Equation 1 presents the naïve 1D convolution.

$$s[i] = (d * g)[i] = \sum_{k=0}^{W-1} d[i+k] \cdot g[k] \tag{1}$$

TABLE I
Glossary of terms used in the convolution methods.

Term	Descripton	Term	Descripton
d	Input feature map (IFMAP)	L	g (weights) width $\|g\|$
g	Weights	W	Width when $M = L$
s	Output feature map (OFMAP)	N	s (output feature map) width $\|s\|$
M	d width $\|d\|$	K	Number of multiplications

Winograd [7] developed a suite of algorithms to optimize convolution, establishing the fast convolution algorithm with the general formulation shown in Equation 2, where: Q: matrix of quotients; A, B, G and C are transform matrices; $\odot$: the Hadamard product [21]–[23].

$$s = A^T[(QBg) \odot (C^T d)] \tag{2}$$

One of the key algorithms developed by Winograd for fast convolution is the Toom-Cook algorithm [7]–[9, 21], initially designed for multiplication. This algorithm enables fast convolution for 1D data and guarantees the minimum number of multiplications. The number of multiplications is given by $K = L + N - 1$, a linear growth, differing from the quadratic growth of the naïve 1D convolution ($K = W^2$).

We use two sets of matrices generated by the Toom-Cook algorithm with $L = 3$. Equation 3 presents a set of matrices for parameters $\{N, L, K\} = \{2, 3, 4\}$, corresponding to 4 multiplications and two output values.

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad C^T = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} -1 \\ \frac{1}{2} \\ \frac{1}{2} \\ 1 \end{bmatrix} \tag{3}$$

Equation 4 shows another set of matrices for $\{N, L, K\} = \{3, 3, 5\}$, corresponding to 5 multiplications and three output values.

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & 0 \\ 0 & 1 & 1 & 4 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \quad C^T = \begin{bmatrix} 2 & -1 & -2 & 1 & 0 \\ 0 & -2 & -1 & 1 & 0 \\ 0 & 2 & -3 & 1 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{6} \\ \frac{1}{6} \\ 1 \end{bmatrix} \tag{4}$$

Multidimensional arrays are standard in signal processing, particularly in image processing and CNNs. These arrays extend the complexity of operations such as convolutions into multiple dimensions. Equation 5 presents the naïve 2D convolution.

$$s[i_1 i_2] = (d * * g)[i_1 i_2] = \sum_{k_1=0}^{W_1-1} \sum_{k_2=0}^{W_2-1} d[i_1 - k_1, i_2 - k_2] g[k_1 k_2] \tag{5}$$

Multidimensional fast convolution combines multiple 1D algorithms. A common strategy is to nest two fast 1D convolutions, permuting rows and columns between stages. This nested formulation optimizes the 2D case by reordering indices, as shown in Equation 6 [20, 24].

$$s = A_1^T \Big\{ \left[(Q_1 B_1) g (Q_2 B_2)^T \right] \odot (C_1^T d C_2) \Big\} A_2 \tag{6}$$

Figure 1 compares the number of multiplications for 2D-naïve and 2D Toom-Cook convolutions, for $L = 9$ (3×3).

Existing studies on convolution and fast multidimensional transformations [24, 25] propose a Kronecker product-based method to bind fast convolution algorithms. Equation 7 expresses our method.

$$s = (A_1 \otimes A_2)^t [G \odot (C_1 \otimes C_2)^t d] \tag{7}$$

The Kronecker product, denoted by $\otimes$, combines two matrices into a larger composite matrix while preserving structural properties. Our approach uses it to bind two fast convolution matrices, one for the first and the other for the second dimensions, resulting in matrices $A = A_1 \otimes A_2$ and $C = C_1 \otimes C_2$.

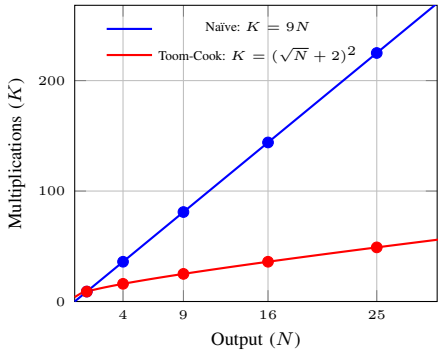

Fig. 1. Multiplications required for 2D-naïve (Eq. 5) and 2D Toom-Cook (Eq. 6) convolutions for $L = 9$ (3×3 weights).

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 & 0 & 1 & -1 & 1 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & -1 & -1 & -1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 & -1 & 1 & -1 & 0 & 1 & -1 & 1 \end{bmatrix} \tag{8}$$

Equation 8 shows an example of the Kronecker product of Toom-Cook matrices for $\{N, L, K\} = \{2, 3, 4\}$ (Equation 3).

To reduce the cost of constant multiplication, we adopt the Multiplierless Constant Multiplication (MCM) technique, which avoids increasing the multiplication count by decomposing constants into sums of powers of two. For example, the integer 18 can be represented as $2^4 + 2^1$, enabling multiplication by 18 to be implemented using bit shifts and additions instead of hardware multipliers. This decomposition facilitates efficient implementation of constant multiplication in both scalar and matrix operations, reducing hardware complexity and resource usage.

III. Fast Convolution Method

To reduce the number of multiplications in the Toom-Cook algorithms compared to the naïve method, it is essential to increase N, as shown in Figure 1. However, with $N > 4$ in 2D or $N > 2$ in 1D convolution, matrices A and C contain values other than 1, -1, and 0, as shown in Equation (4). This behavior contrasts with the case when $N \leq 4$ in 2D or $N \leq 2$ in 1D convolution, as depicted in Equation 3.

Although fast convolution methods significantly reduce the number of multiplications present in the Hadamard product ($\odot$) (Equation 2), for larger values of N, the matrix operations involving A and C, particularly when constants that are not powers of two are involved, introduce trade-offs. These trade-offs must be carefully managed, as they diminish the initial reduction in multiplications offered by fast convolution techniques.

Winograd's inspection-based factorization (IF) method, whose derivation is presented in [7, 8, 10], and parameterized by $\{N, L, K\} = \{3, 3, 6\}$, achieves the exact input-output mapping of the Toom–Cook 3 while eliminating the constant-multiplication overhead, as shown in Equation 4. In IF, the transform matrices A and B are binary, containing only zeros and ones, thus avoiding multiplications by constants. Moreover, the matrix Q consists entirely of ones, eliminating rounding errors in the weight transform and facilitating straightforward online weight updates.

TABLE II
NAIVE AND FAST CONVOLUTIONS ALGORITHMS AND ARCHITECTURES (WORD SIZE: 20 BITS, 20X20 MULTIPLIERS).

Name	Fast	Multidim.	Input	Output	Multip.	Matrix A	Matrix C	Cycles p/ batch	Multipliers	Origin
N^9	Naïve	-	5×5	3×3	9	-	-	27	9	Previous work
MFn^4	Winog. M. Filter	Nested	4×4	2×2	16	$[(2,4),(4,2)]$	$[(4,4),(4,4)]$	6	4	Previous work
TCk^9	Toom Cook	Kronecker	5×5	3×3	25	$(9,25)$	$(25,25)$	9	5	Our work
TCn^9	Toom Cook	Nested	5×5	3×3	25	$(9,25)$	$(25,25)$	9	5	Previous work
IFk^9	Inspect. Factor.	Kronecker	5×5	3×3	36	$(9,36)$	$(36,36)$	9	6	Our work
IFn^9	Inspect. Factor.	Nested	5×5	3×3	36	$[(3,6),(6,3)]$	$[(6,6),(6,6)]$	9	6	Our work

The IF method for convolution, or polynomial multiplication, employs two second-degree polynomials, resulting in a fourth-degree output. Equivalently, this corresponds to one input vector of size 5 and another of size 3, yielding an output vector of size 3, as illustrated in Equation 9.

$$A^T=\begin{bmatrix}1&0&0&1&1&0\\0&1&0&1&0&1\\0&0&1&0&1&1\end{bmatrix}\quad B=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\1&1&0\\1&0&1\\0&1&1\end{bmatrix}\quad C^T=\begin{bmatrix}1&-1&-1&0&0\\0&-1&1&-1&0\\0&0&-1&-1&1\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\end{bmatrix}\quad Q=\begin{bmatrix}1\\1\\1\\1\\1\\1\end{bmatrix} \tag{9}$$

IV. ACCELERATORS' IMPLEMENTATION

This section details the hardware implementation of the algorithms from Sections II and III. Table II summarizes the architectures for both naïve and fast convolution accelerators used in this work. The table categorizes convolution designs by the multidimensional algorithm (Kronecker or Nested), input/output size, total multiplications, matrix sizes (A and C), cycles per convolution batch, and number of multipliers in the accelerator.

The baseline architecture is the naïve accelerator N^9, which consists of nine 20×20-bit multipliers and a summation of the nine outputs using a Carry Save Adder (CSA). A finite state machine (FSM) iterates nine times to generate the 3×3 result. This represents a performance-driven implementation of the naïve convolution, using a set of parallel multipliers and an adder instead of a MAC matrix. Each output requires three clock cycles, resulting in 27 cycles to obtain a 3×3 output (one batch).

Figure 2 illustrates the block diagram of the hardware to execute the fast convolution algorithms.

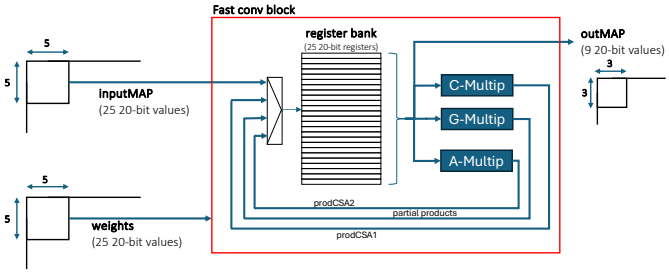

Fig. 2. High-level block diagram of the fast convolution accelerator. Input and output sizes correspond to TCk^9, TCn^9, IFk^9, and IFn^9.

The accelerator has three main blocks: (i) **C-Multip** – performs $D=Cd$ for the Kronecker case and $C_1^T dC_2$ for the Nested case, where d denotes an input or IFMAP window; (ii) **G-Multip** – computes $S=G\odot D$, representing the block with the multipliers; (iii) **A-Multip** – executes AS for the Kronecker case and $A_1^T SA_2$ for the Nested case, generating the output or OFMAP window.

This accelerator operates in a pipelined manner: 1 clock cycle (cc) to load the IFMAP, 1 cc to compute D, 5 cc to perform the multiplications, 1 cc to produce the output s, and 1 cc to send s to the memory. This yields nine cc to produce a batch (3×3 output).

Listing 1 shows a snippet of the **C-Multip** block for a matrix C (25×25) of TCk^9 that contains powers of two constants. The code defines the terms to be summed using shifts, where each P value is an element in d. Two CSAs produce a row product: one for positive terms (*cp[x]*) and one for negative terms (*cn[x]*). The result, *sum[x]*, is computed as *cp[x] - cn[x]*. This approach limits the critical path to the delay from one CSA and the final subtraction, enabling completion in a single clock cycle.

```
1   // Left-shifting input values P to define the terms to be summed
2   s2P0 = {P[0][NBITS-3:0], 2'b00};
3   s1P1 = {P[1][NBITS-2:0], 1'b0};
4   s1P23 = {P[23][NBITS-2:0], 1'b0};
5
6   CSA_8 sp0 (s2P0, s1P3, P[6], s1P7, s1P11, s2P12, s1P15, P[18], cp[0]);
7   CSA_8 sn0 (s1P1, s2P2, s1P5, P[8], s2P10, s1P13, P[16], s1P17, cn[0]);
8   assign sum[0] = cp[0] - cn[0];
9
10  CSA_6 sp21 (s1P8, s1P11, P[12], s2P16, s1P17, P[23], cp[21]);
11  CSA_6 sn21 (s2P6, s1P7, P[13], s1P18, s1P21, P[22], cn[21]);
12  assign sum[21] = cp[21] - cn[21];
```

Listing 1. Snippet of the C-Multipl block ($D=Cd$). The size of the C and d matrices are detailed in Table II. Matrix multiplication is implemented using Carry-save Adders (CSA) to minimize delay. The IF methods does not have shifts (lines 2-4).

The multiplication of matrix D of TCk^9 with the quantized weights requires 25 multiplications. We implemented **G-Multip** with five 20×20-bit multipliers (20-bit results, shifting left the eight less significant bits to remove the quantization). The five multipliers process both matrices row by row, requiring five clock cycles to produce the 25 products. Although it is possible to implement the G-multip block with 25 multipliers, our design aims to minimize silicon area. The **A-Multip** block follows the same approach as the C-Multip block, using shifts and CSAs, and is also executed in a single clock cycle.

For the implementation of the Nested algorithm, the process is similar for both the **C-** and **A-Multip** blocks; however, the difference lies in the fact that each of these blocks is divided into two modules. The input to **C1-Multip** is processed, and its output is fed into **C2-Multip**, which then passes its result to **G-Multip**. The output of **G-Multip**, denoted as S, serves as the input to **A2-Multip**, whose output is subsequently directed to **A1-Multip**, and finally to the output. This is also executed in a single clock cycle.

V. Results

Table III presents results obtained after logic synthesis for the accelerators.

TABLE III
AREA (μm^2) AND POWER AFTER LOGICAL SYNTHESIS (CADENCE GENUS 22.15, 28NM@500MHZ).

Name	Cell Count	Cell Area	Net Area	Total Area	Flop Count	Power (mW)
N^9	7424	7718	3736	11454	193	6.5
MFn^4	5460	6456	2923	9379	324	6.9
IFk^9	10848	14625	5977	20601	725	11.1
IFn^9	9625	11920	5201	17121	725	9.7
TCk^9	14568	22142	8057	30199	1005	14.2
TCn^9	10339	13346	5519	18865	505	12

The **area** and **power** evaluation follow the architectural parameters presented in Table II. The reference implementation (N^9) presents smaller area and power values at the cost of the smallest throughput (27 clock cycles for a batch). Among the fast convolution architectures that produce a 3×3 output, the IF architectures (IFk^9 and IFn^9) present advantages in area and power compared to the Toom-Cook method with output 3×3 (TCk^9 and TCn^9), as the matrices in the IF architectures do not require shifts. The need for multiple shifts and larger matrices (25×25) penalizes the TCk^9 and TCn^9 method (Listing 1).

Figure 3 evaluates the **performance** of the convolution methods by varying the IFMAP size. The advantage of fast convolution methods is noticeable. The TCk^9 method is the fastest, followed by the architectures based on the IF approach and TCn^9, with a performance difference of 7%. Although TCk^9 outperforms the IF based architectures in speed, this advantage incurs an area overhead of 76% and a power increase of 28% relative to IFn^9.

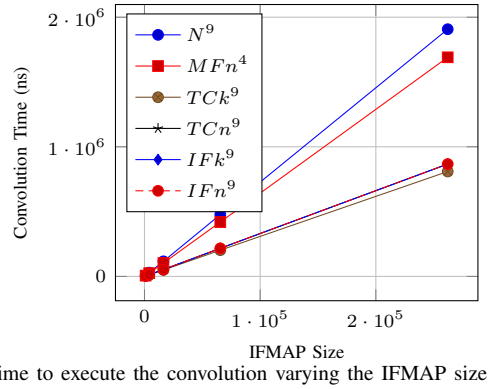

Fig. 3. Time to execute the convolution varying the IFMAP size (32×32, 64×64, 128×128, 256×256 and 512×512). 512×512 has 262,144 input points.

Figure 4 evaluates the **energy** consumption of the convolution methods as a function of the IFMAP size, corresponding to the product power (Table III) $\times$ performance (Figure 3). IFn^9 and IFk^9 emerge as the most energy-efficient solutions, achieving the lowest values and, consequently, the best pixel-per-joule ratio. Although MFn^4, TCn^9 and TCk^9 also reduce energy consumption relative to N^9, the savings are less pronounced.

Table IV compares N^9 in relation to other designs and summarizes the obtained results.

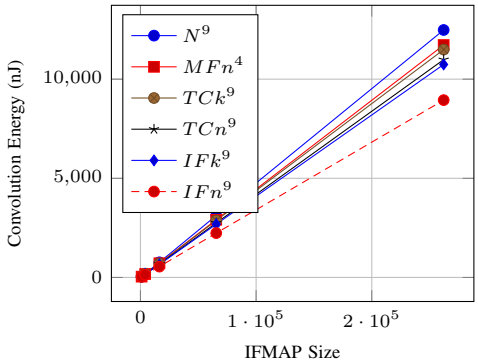

Fig. 4. Energy consumption to execute the convolution varying the IFMAP size, computed as per-cycle power × total execution cycles.

TABLE IV
RELATIVE TO N^9

Name	MFn^4	TCk^9	TCn^9	IFk^9	IFn^9
Total Area	**82%**	264%	165%	180%	149%
Performance	89%	**42%**	45%	45%	45%
Energy	94%	92%	88%	86%	**72%**

IFn^9 reduces energy consumption and enhances performance compared to the naïve convolution and the leading fast-convolution techniques for CNN accelerators, namely MFn^4 and Toom–Cook (TCn^9). Against TCn^9, IFn^9 achieves a 19% energy reduction and a 9% decrease in area while preserving identical throughput. Compared with MFn^4, the proposed design delivers a 49% performance improvement alongside a 24% reduction in energy consumption. Relative to the naïve baseline, IFn^9 yields a 28% reduction in energy and a 55% increase in performance.

VI. Conclusions and Future Work

Hardware acceleration for two-dimensional convolution using fast convolution methods effectively reduces the time required to perform the convolution operation compared to naïve convolution approaches based on MAC operations. In addition to demonstrating improved performance, we also show reduced energy consumption. These benefits compensate for the area overhead. The best area-performance-energy trade-off architecture combines the IFn^9 method with the nested approach for 2D convolutions.

Having demonstrated that convolution can be performed using methods not based on MACs, the primary direction for future work is the development of a complete accelerator that includes memory interfacing and integration between the CNN layers. The main challenge is to achieve memory throughput that matches the throughput of the arithmetic module responsible for the convolution operation.

Additionally, we plan to extend our empirical evaluation by conducting comprehensive benchmarks across CNN models and by prototyping the accelerator on FPGA hardware to obtain real-world performance, power, and energy measurements.

Acknowledgments

This work was financed in part by: CNPq (grant 305621/2024-6), FAPERGS (grant 23/2551-0002200-1), and a Carrefour grant.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with Deep Convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017, https://doi.org/10.1145/3065386.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[3] J. Jang, H. Lee, and H. Kim, "EDeN: Enabling Low-Power CNN Inference on Edge Devices Using Prefetcher-assisted NVM Systems," in *ISLPED*, 2024, pp. 1–6, https://doi.org/10.1145/3665314.3670801.

[4] L. R. Juracy, M. T. Moreira, A. de Morais Amory, A. F. Hampel, and F. G. Moraes, "A High-Level Modeling Framework for Estimating Hardware Metrics of CNN Accelerators," *IEEE Transactions on Circuits and Systems – I*, vol. 68, no. 11, pp. 4783–4795, 2021, https://doi.org/10.1109/TCSI.2021.3104644.

[5] L. R. Juracy, A. de Morais Amory, and F. G. Moraes, "A Fast, Accurate, and Comprehensive PPA Estimation of Convolutional Hardware Accelerators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 12, pp. 5171–5184, 2022, https://doi.org/10.1109/TCSI.2022.3204932.

[6] L. R. Juracy, A. M. Amory, and F. G. Moraes, "A Comprehensive Evaluation of Convolutional Hardware Accelerators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 3, pp. 1149–1153, 2023, https://doi.org/10.1109/TCSII.2022.3223925.

[7] S. Winograd, *Arithmetic Complexity of Computations*. Society for Industrial Mathematics, 1987.

[8] R. E. Blahut, *Fast Algorithms for Signal Processing*. Cambridge University Press, 2010, https://doi.org/10.1017/CBO9780511760921.

[9] R. Tolimieri, M. An, and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution*. Springer Science & Business Media, 2013, https://doi.org/10.1007/978-1-4757-2767-8.

[10] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Wiley-Interscience, 1999.

[11] A. Younesi, M. Ansari, M. Fazli, A. Ejlali, M. Shafique, and J. Henkel, "A Comprehensive Survey of Convolutions in Deep Learning: Applications, Challenges, and Future Trends," *IEEE Access*, vol. 12, pp. 41 180–41 218, 2024, 10.1109/ACCESS.2024.3376441.

[12] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 857–870, 2020, https://doi.org/10.1109/TCAD.2019.2897701.

[13] S. Kala, B. R. Jose, J. Mathew, and S. Nalesh, "High-Performance CNN Accelerator on FPGA Using Unified Winograd-GEMM Architecture," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 27, no. 12, pp. 2816–2828, 2019, https://doi.org/10.1109/TVLSI.2019.2941250.

[14] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs," in *Design Automation Conference (DAC)*, 2017, pp. 1–6, https://doi.org/10.1145/3061639.3062244.

[15] S. Kala, D. Paul, B. Jose, and S. Nalesh, "Design Space Exploration of Convolution Algorithms to Accelerate CNNs on FPGA," in *International Symposium on Embedded Computing and System Design (ISED)*, 2018, pp. 21–25, https://doi.org/10.1109/ISED.2018.8704043.

[16] J. Shen, Y. Huang, Z. Wang, Y. Qiao, M. Wen, and C. Zhang, "Towards a Uniform Template-based Architecture for Accelerating 2D and 3D CNNs on FPGA," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018, pp. 97–106, https://doi.org/10.1145/3174243.3174257.

[17] A. Ahmad and M. A. Pasha, "Towards Design Space Exploration and Optimization of Fast Algorithms for Convolutional Neural Networks (CNNs) on FPGAs," in *ACM/IEEE Design, Automation Test in Europe Conference (DATE)*, 2019, pp. 1106–1111, https://doi.org/10.23919/DATE.2019.8715272.

[18] X. Wang, C. Wang, J. Cao, L. Gong, and X. Zhou, "WinoNN: Optimizing FPGA-Based Convolutional Neural Network Accelerators Using Sparse Winograd Algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4290–4302, 2020, https://doi.org/10.1109/TCAD.2020.3012323.

[19] H. Ye, X. Zhang, Z. Huang, G. Chen, and D. Chen, "HybridDNN: A Framework for High-Performance Hybrid DNN Accelerator Design and Implementation," in *ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6, https://doi.org/10.1109/DAC18072.2020.9218684.

[20] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4013–4021, https://doi.org/10.1109/CVPR.2016.435.

[21] D. G. Myers, *Digital Signal Processing: Efficient Convolution and Fourier Tranform Techniques*. Prentice Hall, 1990.

[22] R. Tolimieri, M. An, and C. Lu, *Mathematics of Multidimensional Fourier Transform Algorithms*. Springer, 1997, https://doi.org/10.1007/978-1-4612-1948-4.

[23] J. H. McClellan and C. M. Rader, *Number Theory in Digital Signal Processing*. Prentice-Hall, 1979.

[24] S. Winograd, "On Computing the Discrete Fourier Transform," *Mathematics of Computation*, vol. 32, no. 141, pp. 175–199, 1978, https://doi.org/10.1090/S0025-5718-1978-0468306-4.

[25] R. Agarwal and J. Cooley, "New Algorithms for Digital Convolution," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 5, pp. 392–410, 1977, https://doi.org/10.1109/TASSP.1977.1162981.