Case Study on the Use of Winograd-Based Convolution for CNN Inference in FPGA

Artur H. C. Pereira[1], **Marcus V. Lamar**[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Caixa Postal 4466 – 70.910-900 – Brasília – DF – Brazil

arturhcpereira@aluno.unb.br, lamar@unb.br

Abstract. *This work explores the implementation of Convolutional Neural Networks (CNNs) on FPGAs. Two circuits are investigated: one based on the spatial 2-D convolution algorithm and another based on the Winograd algorithm for convolution. The Winograd algorithm simplifies the convolution process through linear transformations that reduce the number of multiplications required. However, there is an increase in circuit complexity due to the additional logic to perform the transformations. The two implementations are compared in a case study of a simple architecture to solve the handwritten digits classification problem in the MNIST dataset.*

Resumo. *Este trabalho explora a implementação de Redes Neurais Convolucionais (CNNs) em FPGAs. Dois circuitos são investigados: um baseado no algoritmo de convolução espacial 2-D e outro baseado no algoritmo de Winograd para convolução. O algoritmo de Winograd simplifica o processo de convolução por meio de transformações lineares que reduzam o número de multiplicações necessárias. No entanto, há um aumento na complexidade do circuito devido à lógica adicional para realizar as transformações. As duas implementações são comparadas em um estudo de caso de uma arquitetura simples para resolver o problema de classificação de dígitos manuscritos no conjunto de dados MNIST.*

1. Introduction

Convolutional Neural Networks (CNNs) are a class of deep artificial neural networks designed to preserve the spatial relationship within their inputs. Therefore, they are highly suitable and effective for processing and analyzing images in computer vision tasks. However, CNNs come with a high computational cost. This makes it interesting, if not necessary in some cases, to use dedicated hardware to accelerate model processing.

Nowadays, the most widely adopted approach for accelerating CNNs is the use of Graphics Processing Units (GPUs). However, GPUs come with a high cost in terms of energy consumption, making this solution unsuitable for more constrained scenarios such as embedded systems. Other technologies that can be explored for CNN acceleration are Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). The implementation of CNNs on FPGAs has been consistently studied in recent times due to the greater flexibility this approach provides through the ease of reconfiguring the designed circuit. To harness this flexibility, this work aims to study the implementation of the convolution circuit based on the Winograd algorithm. A circuit based on classic spatial convolution has also been implemented for reference comparison.

2. Related Work

A relatively recent literature review on FPGA acceleration of neural networks shows that there is still relevance to this field [Wu et al. 2021]. An argument presented is the greater flexibility the FPGA platform provides and its efficient use of energy. In that work, the optimization techniques are divided into categories, one of which is the reduction of computation complexity. That class of optimization involves using fast algorithms, such as the Winograd algorithm.

The Winograd algorithm has been evaluated as a technique for convolution acceleration and found to be more efficient for small filter convolutions when compared to the Fast Fourier Transform (FFT), other fast algorithm [Lavin and Gray 2016]. Some following works, [Wang et al. 2017] and [Liang et al. 2019], present FPGA implementations that incorporate this algorithm in their designs, being able to achieve performances comparable to GPU acceleration in the state of the art models.

3. Winograd Algorithm

Let denote the computation of m outputs of an input tile convolved with an r-tap filter $F(m, r)$. The conventional convolution requires $m \times r$ multiplications to perform the computation. For instance, $F(2, 3)$ requires $2 \times 3 = 6$ multiplications. Winograd documents the following algorithm in [Winograd 1987] that uses $m+r-1 = 4$ multiplications, according

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}, \tag{1}$$

where

$$\begin{aligned} m_1 &= (d_0 - d_2)g_0, & m_2 &= (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}, \\ m_4 &= (d_1 - d_3)g_2, & m_3 &= (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}. \end{aligned} \tag{2}$$

The 1-D algorithm can be nested with itself to perform the 2-D convolution $F(m \times m, r \times r)$. In matrix form, the 2-D algorithm can be written as

$$Y = A^T \left[(GgG^T) \odot (B^T dB) \right] A, \tag{3}$$

where g and d are the filter and the input data tile, G and B^T are their respective transformation matrices and A^T is the inverse transformation matrix. The purpose of the transformations is to have the filter and inputs with the same dimensions so that an element-wise multiplication can be performed. The inverse transformation then brings the output back to the pixel space.

4. Circuit modeling and implementation

Figure 1 illustrates the workflow for generating the proposed circuit and obtaining its outputs. The process begins with implementing and training a CNN model in software.

This step was carried out using Python with the Keras library. This library allows the model to be saved in an HDF5 format file, from which the Memory Initialization Files (MIFs) are generated for each layer that will be implemented in hardware. Similarly, the input image to be processed is also converted into a MIF file and provided for circuit compilation. Once compiled, the circuit is loaded onto the FPGA. Finally, the results are written to an output memory, which can be analyzed after the time required for the circuit to perform inference.

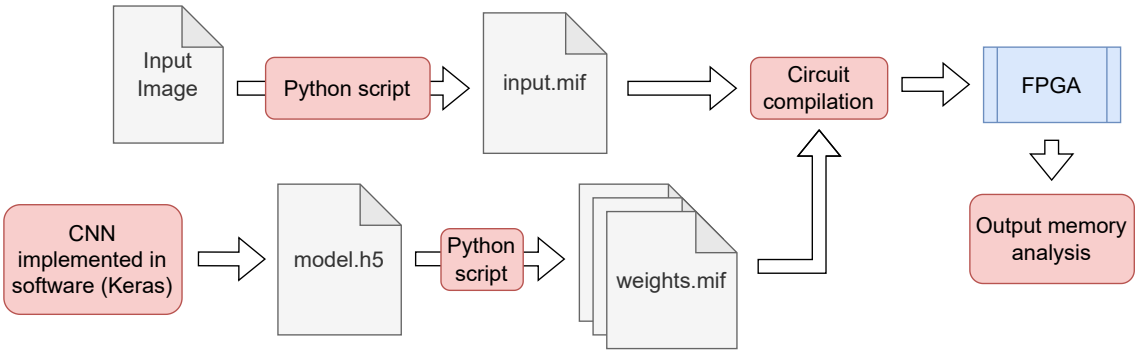


Figure 1. Flow of the proposed system. Processes are highlighted in red, and the generated circuit's test device is highlighted in blue. The remaining blocks represent intermediate files used in circuit compilation.

An internal memory module is instantiated for each layer implemented in the circuit to store its weights. The input image of the circuit is also loaded into a memory module, and to evaluate the output results, the values of the neurons in the Fully Connected (FC) layer are also written to an output memory. Figure 2 illustrates the schematic of the overall developed circuit, including the relationship between layers and weight memories, using a two-layer convolutional architecture as an example. The input of the convolutional cores is stored in a module called window buffer. As illustrated in Figure 3, data is sequentially received at the input and placed in the last row of registers within the value window used in the convolution computation. As a value enters this window, the previously entered values are shifted to the adjacent register. If no adjacent register is available, the value is stored in the appropriate line buffer.

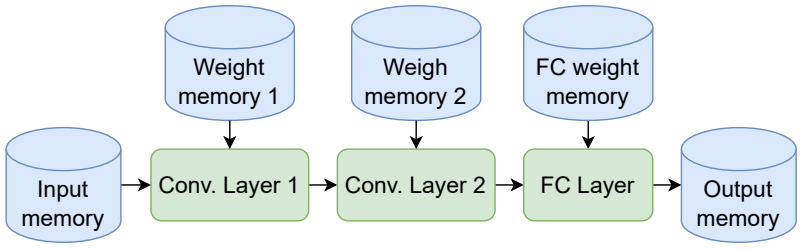


Figure 2. Diagram of the general circuit structure using two convolutional layers for illustration. Internal memory elements are shown in blue and the circuits responsible for processing each layer are represented in green.

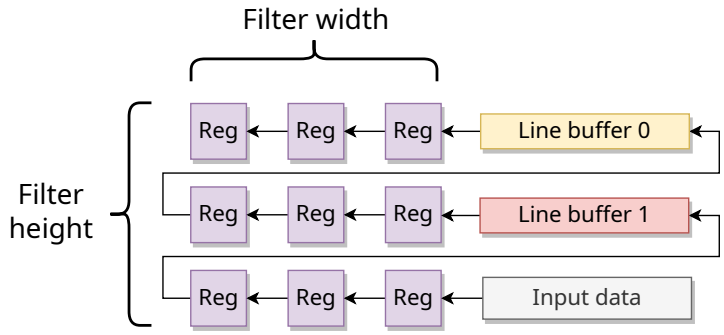


Figure 3. Diagram of two line buffers being used to form a sliding window for convolution with a 3 × 3 filter.

The schematic of the internal components of the spatial convolution core is shown in Figure 4. This circuit employs a module called a *kernel*, which implements the multiplication between the values of the input window from one channel and the values of a channel in the filter. The *kernel* module performs these multiplications in parallel. The results of these multiplications are summed together and accumulated in the channel_acc register.

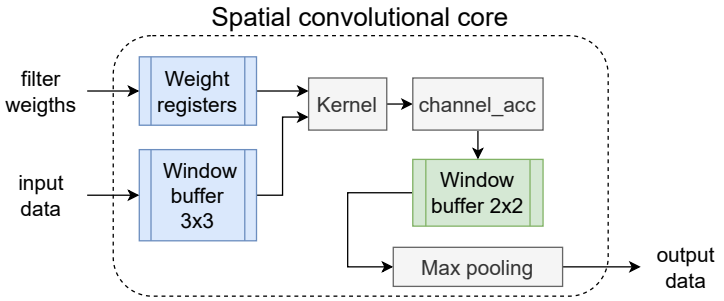


Figure 4. Diagram of the internal components of a spatial convolutional core. Elements in blue are replicated for each input channel. Elements in green are replicated for each layer's filter, i.e., output channels.

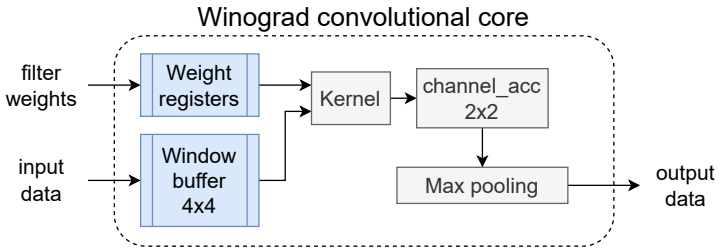


Figure 5. Diagram of the internal components of a Winograd convolutional core. Elements in blue are replicated for each input channel.

The implementation of Winograd convolution is somewhat more constrained, as it is necessary to establish the transformation matrices to be used. As discussed in Section 3, the $F(2 \times 2, 3 \times 3)$ algorithm was implemented, meaning a convolution with a 3×3 filter and a 4×4 input window to produce a 2×2 output window. Due to this, the *kernel* module

of the Winograd convolution core does not have parameterizable input dimensions. This module takes a 4×4 input window in the spatial domain, performs linear transformation to obtain values in the Winograd domain, multiplies them with the filter values in parallel, and then inversely transforms to get the 2×2 result in the spatial domain. These values are stored and accumulated in four channel_acc registers, as shown in Figure 5. In the implementation of this module, it was assumed that the weights read from memory correspond to the coefficients of the already-transformed 4×4 filters. This increases the memory space required for weight storage but reduces circuit complexity.

5. Results

This work was developed in Verilog HDL, using the Quartus Prime Lite Edition tool, version 21.1, for circuit synthesis and FPGA programming. The Quartus' *In-System Memory Content Editor* tool was utilized for analyzing the output memory. The machine employed in the development was a computer with an AMD Ryzen 5 5600G processor, 16GiB of RAM, and the Ubuntu 22.04.2 LTS operating system.

The development board used for implementing the project circuits was the DE1-SoC, which features an Intel Cyclone V FPGA model 5CSEMA5F31C6 with 32,070 Adaptive Logic Modules (ALMs), 85,000 Adaptive Look-Up Tables (ALUTs), and 87 Digital Signal Processors (DSPs). DSPs refer to specialized hardware circuits for performing multiplications. The numerical representation precision adopted in the circuit implementation was 32-bit Q16 fixed-point, meaning 16 fractional bits. Some spatial characterization results couldn't be compiled for loading onto the board, thus displaying the usage of Logic Elements in ALUTs instead of ALMs. The source code for this project is publicly available in the GitHub repository[1].

5.1. Case Study

A simple problem of handwritten digit classification was chosen to assess the error introduced by fixed-point precision and Winograd transformations. The choice of this problem stems from the fact that the complexity of the required CNN to achieve good results is relatively lower compared to more general classification problems and detection and segmentation tasks. The implemented network was trained with the MNIST dataset, and its architecture is illustrated in Figure 6.

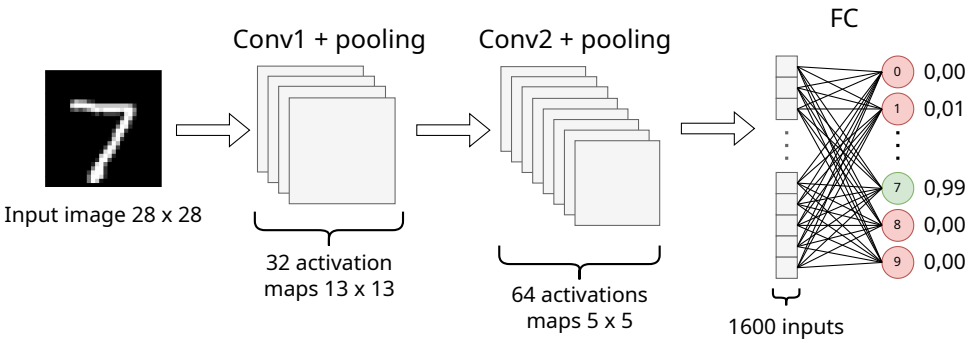


Figure 6. Diagram of the implemented CNN. The neurons of the FC layer pass through the *softmax* activation function to provide the classification probabilities of the input.

[1] https://github.com/ArturHugo/cnn_fpga.git

The network consists of two convolutional layers with 32 and 64 filters respectively, followed by a FC layer with 10 neurons, one for each digit from 0 to 9. This architecture is an example presented in the Keras library documentation. It was chosen as a case study due to its accuracy of approximately 99% on the test set of the used dataset [Sim].

5.2. Spatial Analysis

The implementation of a fully combinational one-layer FC network with 1,600 inputs and 10 outputs, needed in the case study, needs physical requirements estimated in 7,379,857 ALUTs and for the circuit synthesis 4,128,064 ALMs. These numbers of Logic Elements do not fit in any commercial FPGA in 2023. Thus, a sequential version of the FC network was developed to implement in the DE1-SoC development kit. However, the sequential implementation might have potentially introduced delays in the temporal performance of inference.

Table 1 presents the physical requirements after mapping the circuit onto the target device. This provides an insight into the proportion of resources utilized, as the ALMs of the FPGA are limited to 32,070 units. From the presented data, it is immediately noticeable that the Winograd approach consumes slightly more resources than the spatial approach, particularly the number of DSPs, which exceeds the target device's limit of 87.

Table 1. Spatial requirements obtained for the case study for both spatial and Winograd implementations, with and without the FC layer.

Requirements	Spatial Conv.		Winograd Conv.	
	Without FC	**With FC**	**Without FC**	**With FC**
ALMs	21,887 (68%)	22,831 (71%)	22,554 (70%)	23,359 (73%)
Registers	34,656	34,455	37,029	36,822
Memory Bits	2,589,424 (64%)	3,091,552 (76%)	2,590,394 (64%)	3,091,392 (76%)
DSPs	56 (64%)	86 (99%)	87 (100%)	87 (100%)

5.3. Time Analysis

Both circuits were simulated in order to measure the number of cycles necessary for completing the inference of one input image. The spatial convolution approach took a total o $6{,}127{,}753$ cycles, while the Winograd's circuit took $2{,}334{,}853$. Winograd's convolution approach of simultaneously calculating four outputs provides a significant temporal advantage to this implementation, needing only 38.10% of the cycles used by the spatial convolution.

Table 2. Maximum frequencies, cycle counts, and total processing times for processing input with and without a sequential FC layer.

Results	Spatial Convolution		Winograd Convolution	
	Without FC	**With FC**	**Without FC**	**With FC**
f_{max} (MHz)	29.54	22.15	21.25	19.97
Cycles	6,125,385	6,127,753	2,334,853	2,334,853
Total Time (ms)	207.36	276.65	109.88	116.92

In addition to the simulations, to evaluate the delay introduced by the sequential FC layer, the number of cycles required to compute the output of the last convolutional layer without a connected FC layer was also measured. Table 2 presents the values obtained for each implemented circuit with and without the sequential FC layer.

The values clearly indicate that the Winograd circuit was not impacted by the sequential nature of the layer, meaning that the time it takes for the FC layer to read the next weights from memory is shorter than the time the previous layer takes to calculate the next output of the same channel. However, in the case of spatial convolution, in some instances, the convolutional layer had to wait for a `hold` signal from the FC layer to fall, resulting in a difference of $2{,}368$ cycles.

From a delay analysis, it was possible to extrapolate the propagation delay of the $1{,}600$ inputs combinational FC needed for the case study. The result was $t_{pd} = 2{,}721$ ns. Considering the maximum frequency value of 29.54 MHz obtained for the circuit without FC in Table 2, the delay introduced by the combinational FC would be 81 cycles. Therefore, for the spatial convolution circuit, using the combinational FC would be advantageous. The same is true for the Winograd circuit, which would have a delay of 58 cycles considering the frequency of 21.25 MHz.

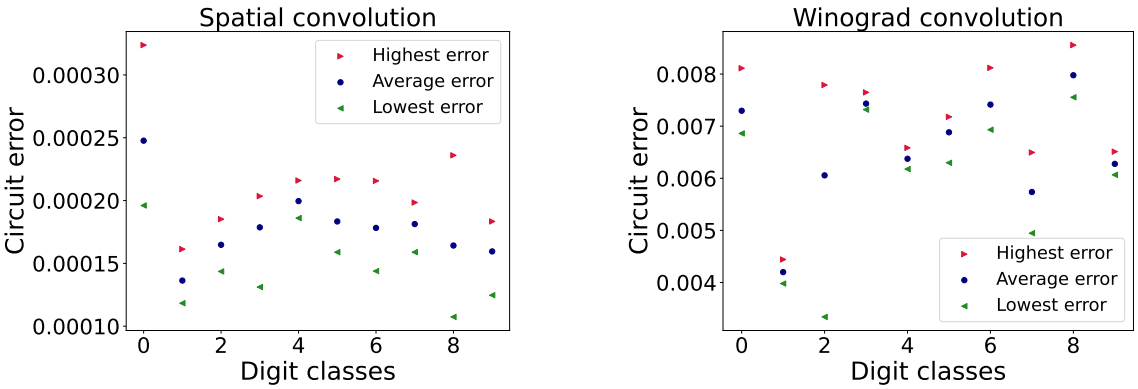
5.4. Output Error Analysis

To test the implementations of the example CNN, a random sub-sampling of 30 images from the test set was conducted, with 3 images from each class. This subset of the test dataset was utilized to evaluate the magnitude of errors introduced by fixed-point precision and Winograd algorithm transformations. Since the hardware circuit does not implement the *softmax* activation of the FC layer, this activation function was removed from the software implementation to obtain reference values for the neurons.

The graphs presented in Figure 7 illustrate the ratio of errors relative to the smallest and largest errors obtained for each class. In Figure 7a, it can be observed that the most significant error obtained with the spatial convolution algorithm implementation did not exceed 3.5×10^{-4}. Meanwhile, in Figure 7b, it can be seen that the highest error from the Winograd circuit does not exceed 9×10^{-3}. Consequently, the errors introduced by the transformations in the Winograd implementation are an order of magnitude greater than the errors introduced solely by fixed-point precision. However, errors of that magnitude do not affect the classification accuracy of the CNN.

6. Conclusion

This work aimed to explore implementations of CNNs on Field-Programmable Gate Arrays (FPGAs). Two implementations were investigated: one based on the conventional 2-D convolution algorithm, also called spatial convolution, and another based on the Winograd algorithm. To analyze the performance of both approaches and evaluate the error introduced by fixed-point numerical representation and the linear transformations of the Winograd algorithm, a case study was physically implemented on the DE1-SoC development kit. Due to the resource limitations of the FPGA, the CNN used as a case study was a network with two convolutional layers, solving the simple handwritten digit classification problem from the MNIST dataset.



(a) Errors for each class obtained by the spatial convolution circuit.

(b) Errors for each class obtained by the Winograd-based circuit.

Figure 7. Errors in the outputs of the FC layer calculated by hardware implementations relative to the reference values of the software implementation.

In the conducted case study, it was evident that the time performance of the Winograd circuit was superior, completing the processing of a sample CNN input in almost one-third of the cycles required by the spatial convolution circuit. This outcome was anticipated, as the Winograd algorithm computes four output elements of convolution simultaneously for each processed input window. Consequently, four spatial convolution filter modules would need to be parallelized to achieve this performance, which would undoubtedly increase the spatial requirements of the circuit. For future work, it would be interesting to improve some aspects of the presented implementation. For instance, increasing the filter parallelism of the circuit by utilizing multiple convolutional cores per layer. Additionally, it would be important to explore the use of external memory resources or a more robust FPGA to enable the implementation of more complex architectures.

References

Simple MNIST convnet. `https://keras.io/examples/vision/mnist_convnet/`. Acessado em: 26/07/2023.

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.

Liang, Y., Lu, L., Xiao, Q., and Yan, S. (2019). Evaluating fast algorithms for convolutional neural networks on FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(4):857–870.

Wang, D., Xu, K., and Jiang, D. (2017). Pipecnn: An OpenCL-based open-source FPGA accelerator for convolution neural networks. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 279–282.

Winograd, S. (1987). *Arithmetic complexity of computations*. CBMS-NSF regional conference series in applied mathematics 33. Society for Industrial and Applied Mathematics.

Wu, R., Guo, X., Du, J., and Li, J. (2021). Accelerating neural network inference on FPGA-based platforms—a survey. *Electronics*, 10(9):1025.