Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

COMPREHENSIVE EXPLORATION AND DESIGN IMPLEMENTATION OF AN FPGA-BASED CONVOLUTIONAL NEURAL NETWORK

MOHAMED FEKIR
Laboratory of Research in Electrical Engineering and Automation (LREA), University Yahia Feres of Medea, Algeria. Corresponding author, Email: m.fekir@univ-dbkm.dz

Pr. ABDELHAFIDH MOUALDIA
Laboratory of Research in Electrical Engineering and Automation (LREA), University Yahia Feres of Medea, Algeria.

Dr. MOHAMED DALI
Laboratory of Research in Electrical Engineering and Automation (LREA), University Yahia Feres of Medea, Algeria.

Abstract

This paper explores the use of a cost-effective Field Programmable Gate Array (FPGA) to deploy a Convolutional Neural Network (CNN) while ensuring optimal performance. CNNs are a type of Artificial Neural Network (ANN) designed for image processing and computer vision tasks. They are inspired by the intricate structure of the biological visual cortex and can identify complex patterns in large datasets. Compared to traditional deep learning models, CNNs are better at pattern recognition and require fewer computational resources for training and deployment. However, implementing CNNs on an FPGA presents challenges that require a thorough evaluation of performance metrics. Our research has two main objectives: first, we focus on training the CNN model to achieve high accuracy, and second, we optimize the hardware design to suit the FPGA platform. We use the LeNet5 CNN model and the Modified National Institute of Standards (MNIST) dataset for experimentation. High-level synthesis (HLS) is used to assess the CNN's performance on a VC707 FPGA board. Our results show an accuracy rate of over 97% and a latency of 299.3µs, demonstrating the effectiveness of our FPGA implementation in achieving robust CNN performance

Keywords: Convolutional neural networks (CNNs), Field Programmable Gate Array (FPGA)), hardware implementation, optimization.

1. INRODUCTION

Convolutional neural networks (CNNs) have emerged as a prominent class of models resulting from the recent advancement of artificial neural networks (ANNs). Over time, they have consistently fulfilled their potential, capturing the attention of an expanding community of researchers across diverse scientific domains. Consequently, CNNs have found extensive application in image processing tasks like classification [1], segmentation [2], and vision [3,4], along with their utilization in the realms of the Internet of Things (IoT) [5], remote sensing [6], and robotics [7]. As a consequence, numerous pre-trained CNN models have been developed, such as LeNet [8], AlexNet [9], VGGNet [10], ResNet [11], and CaffeNet [12], among others. These models exhibit variations in depth and efficiency; those with deeper architectures possess more parameters, resulting in increased computational complexity.

The implementation of Convolutional Neural Networks (CNNs) poses significant challenges concerning computational power and large-scale general-purpose operation.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

Numerous implementations have been proposed for various platforms, including Central Processing Units (CPUs) [13], Graphical Processing Units (GPUs) [14], Application-Specific Integrated Circuits (ASICs) [15], and Field-Programmable Gate Arrays (FPGAs) [16]. Among these, FPGAs exhibit superior performance in terms of power efficiency and the availability of reconfigurable internal resources [17]. The paramount principles for achieving optimal CNN hardware optimization on FPGA involve data reuse, internal hardware resource minimization, and data transmission reduction. The ultimate objective is to narrow the gap between the required performance and the performance capabilities offered by the hardware platform. This is a complex topic that has sparked the interest of many researchers. As a result, different approaches have been proposed in various studies [17-19]. According to these authors, over 90% of calculations are found in the convolutional layer [20]. Therefore, most approaches focus on optimizing this layer.

Many optimization approaches for CNN have been suggested in the literature, each with its unique solutions and ideas. Some try to reduce the memory consumption of numerous parameters, while others aim to decrease the computational latency and the throughput of nested loops in convolutional layers. However, all these approaches strive to achieve the best possible optimization.

In their study, Han et al. [21] utilized the network pruning technique to simplify and mitigate the overfitting of the CNN model. Subsequently, in another study, Han et al. [22] proposed deep compression to further decrease the memory requirements of CNNs by enforcing weight distribution. Chen et al. [23] employed fixed feature maps and weight representations to minimize computational resources and memory usage for the same purpose. Denton et al. [24], in a separate paper, also used the technique of singular value decomposition (SVD) for this purpose. Van Loan et al. [25] likewise reduced multiple weights and biases in the fully connected layers using the same technique.

Zhang et al. [26], introduced a customized computation method for nested loops that involves loop unrolling, loop tilling, and loop exchange. This approach is specifically designed for convolutional layers, which require multiply and accumulate (MAC) operations of kernel weights and a sliding window of future maps in three dimensions. Over the years, various methods have been proposed in the literature to refine each level of these loops and study their impact on resources and performance. Ma et al. [20] conducted a detailed examination of the aforementioned nested loops of the convolutional layer. To build on their earlier work, the same authors attempted to improve the unrolled and tilling loops in subsequent work [27]. However, this approach improved the throughput of the convolutional layer at the expense of maximizing the utilization of internal resources.

Based on the aforementioned works, we can note the following points:

The hardware description language (HDL) has been the main basis for hardware implementation, while the recent emergence of new FPGA-compatible high-level programming tools, such as High-Level Synthesis (HLS), OpenCV, and OpenCL, has opened the door to a much more efficient development cycle, allowing engineers and researchers to rapidly prototype their hardware designs;

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

The focus is solely on the hardware design phase of the CNN models, ignoring the training phase, which is a separate but equally important step in implementing an accurate and optimized CNN mode.

In this context, the objective of the work is to present a comprehensive exploration and design implementation of the CNN model, which has been carried out in two sequential phases: training implementation using the latest library optimization tools, followed by hardware design implementation on an FPGA platform using a high-level synthesis design methodology.

The contributions of the proposed work are briefly listed below:

We implemented the LeNet5 CNN model in Python, then compiled and accurately validated the training phase, and finally saved the weights and biases of the different filters for further optimization and implementation in the hardware phase;

Using Vivado HLS, we implemented the basic architecture of the LeNet5 CNN model as a register transfer level (RTL) design using the saved weights and biases of the trained filters;

Using Vivado HLS synthesis and analysis tools, we tested the hardware performance and then optimized the architecture by applying techniques such as loop unrolling, pipeline optimization, loop interchange, and folding techniques to decrease the latency;

We then compared the performance of the basic architecture with the optimized architecture solutions using metrics such as latency, throughput, and resource utilization.

This research paper is organized into several sections. First, Section 2 provides an overview of Convolutional Neural Networks (CNNs) as background information. Then, in Section 3, the approach for implementing a CNN model is explained. In Section 4, the results of both the hardware implementation and training phases are presented. Finally, the paper concludes in the last section.

2. BACKGROUND

This section provides a background on CNN basics in general and a detailed description of the LeNet5 CNN model architecture in particular.

2.1 CNN Basics

The CNN models comprise multiple layers, each tasked with learning distinct facets of the data. The primary layers encompass the convolutional layer, tasked with feature extraction from the input data; the activation layer, governing neuron behavior; the pooling layer, reducing convolutional output dimensionality; the fully connected layer, functioning as an input data classifier; and the output layer, generating the intended outcome.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

2.2 LeNet5 Architecture

The fundamental prerequisite for implementing transfer learning is the presence of both a pre-trained network and a suitable test dataset. Figure 1 shows the architecture of LeNet5.

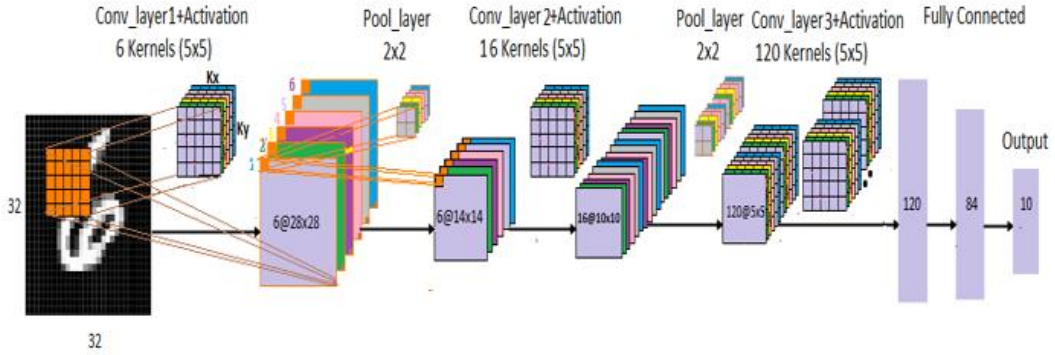


Fig. 1: LeNet5 architecture

3. OVERVIEW OF THE PROPOSED IMPLEMENTATION

Figure 2 provides a comprehensive overview of the proposed implementation, delineating its principal constituents encompassing both the training and hardware implementation phases of the CNN model's intellectual property (IP) core. Each of these two phases is segmented into a sequence of sub-stages, which are further dissected into tasks expounded upon extensively in the ensuing sections. The training process is executed utilizing the Python programming language, whereas the hardware implementation is realized through employment of a High-Level Synthesis (HLS) tool. At inception of the training phase, a dataset is imported and subjected to preprocessing procedures to facilitate the training of the CNN model. Upon culmination of the validation process, the model's weight parameters are saved and subsequently transmitted to the dynamic DDR3 Random Access Memory (RAM) component situated within the FPGA, earmarked for utilization within the HLS-based implementation. The entirety of the hardware system is encompassed within a singular FPGA chip; wherein external memory is furnished by the DDR3 DRAM.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

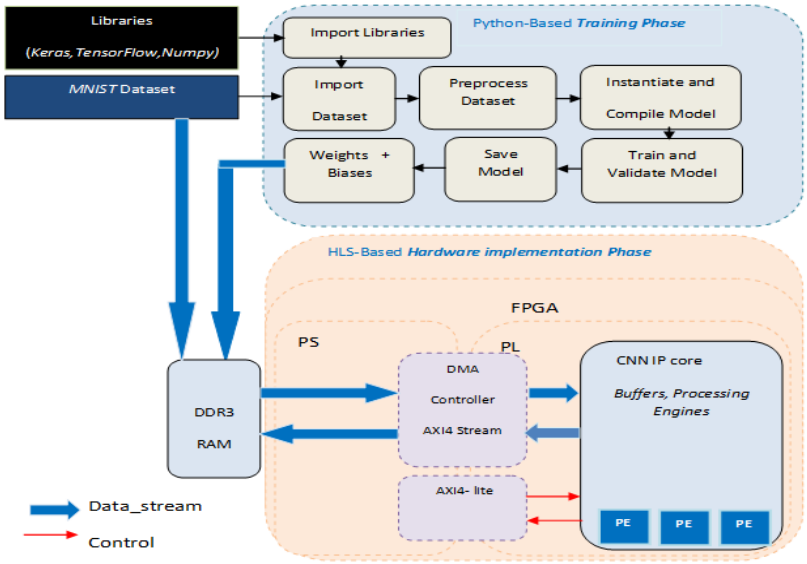


Fig. 2: The proposed implementation overview

3.1.1 Importing libraries

Libraries are required for a Python implementation to perform many specialized tasks related to a given topic. In our case, the following libraries were imported and used for our FPGA-based CNN implementation:

Keras [28], which is an open-source neural network library written in Python;

Numpy, which is a library used for scientific computing and working with multidimensional arrays;

TensorFlow, which is an open-source software library for numerical computation.

3.1.2 Importing Dataset

In this work, we used the MNIST open dataset available for training. This dataset consists of 60,000 training samples and 10,000 label samples. After importing, we printed 25 random samples to confirm the success of the import, as shown in Figure 3.

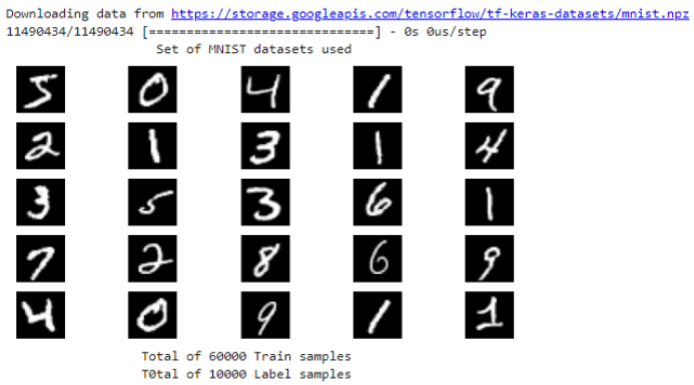


Fig. 3: Sample from the imported MNIST dataset

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

3.1.3. Preprocessing of the Dataset

Upon import, the dataset is categorized into three distinct subsets: 1) a training dataset employed for training the CNN model, 2) a validation dataset used during training to assess CNN performance at varying iteration epochs, and 3) a test dataset employed subsequent to the conclusion of the training phase for performance evaluation. Additionally, the initial 28×28 dataset undergoes resizing with a padding value of 2 to conform to the 32×32 input layer specifications of the LeNet5 architecture. It is then transformed into a 1D vector, with pixel values of the dataset normalized to a range between 0 and 1. The sequence of steps is illustrated in Figure 4.

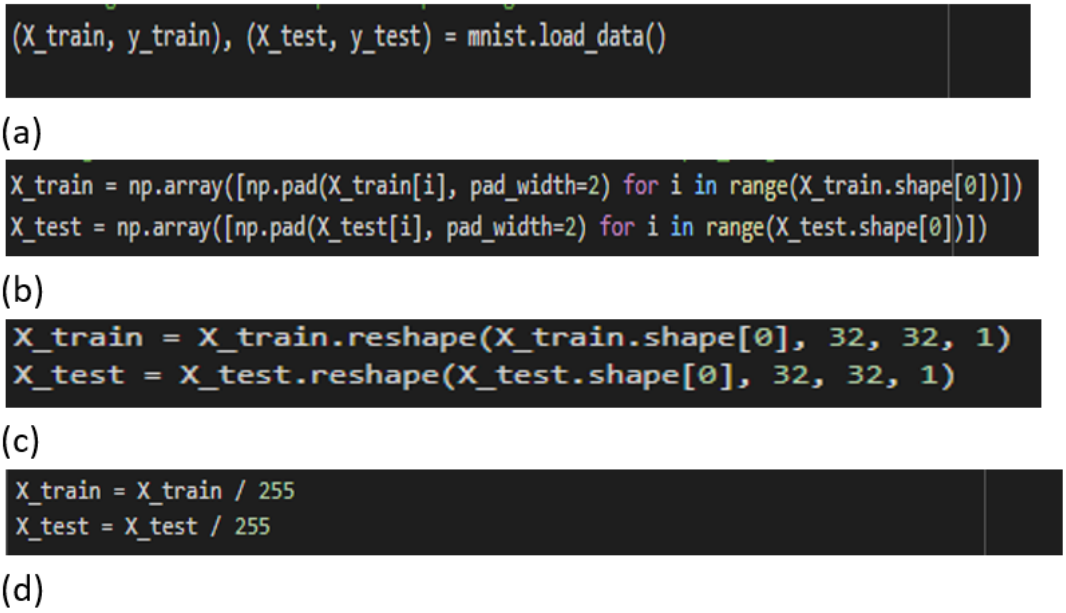


Fig. 4: Samples codes used to preprocess the dataset: (a) loading and splitting the dataset; (b) resizing the dataset to 32×32 using padding values of 2; (c) transforming the dataset into a 1-D vector; (d) normalizing the pixel values of the training and testing datasets

3.1.4. Building the CNN's Model

This work focuses on the LeNet-5 CNN model, which was discussed in Section 2. The LeNet-5 was designed specifically to recognize handwritten numerical digits. Each of its convolutional layers uses a 5x5 kernel with a stride of 1 and is activated with Rectified Linear Units (ReLU). The pooling layers use a 2x2 kernel with a stride of 2. The convolutional layers have 6, 16, and 120 kernels, and the output layer has ten neurons, one for each digit from 0 to 9.

3.1.5. Compilation of the Model

Before starting the training, we introduced the essential compilation parameters during the learning phase. These parameters are the number of epochs, the batch size, the loss

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

function, the optimizer, and the evaluation metric function. Figure 5 illustrates the compilation parameter settings.

```
EPOCHS = 10
BATCH_SIZE = 32
LOSS = keras.metrics.categorical_crossentropy
OPTIMIZER = keras.optimizers.SGD()
METRIC = ['accuracy']
```

Fig. 5: Settings for compilation parameters

3.1.6 Saving the Model

After validating the CNN model, the weights and biases are saved in H5 format for future use in the hardware implementation phase.

3.2. Hardware design implementation

This section will cover the design process of the CNN IP core HLS-based accelerator. Firstly, we will provide a thorough overview of its top-level design and methodology. Then, we will optimize convolutional layer operations step-by-step to reduce latency. Lastly, we will discuss the architecture and design parameters of an HLS-based accelerator to identify the best design space that meets project requirements.

3.2.1. CNN IP Core Overview

The CNN IP core is designed in a data flow structure that employs sequential tasks to form an architecture that promotes parallel processing. This, in turn, enhances throughput and reduces latency, as depicted in Figure 6. The output data buffers of each layer provide sufficient data for the next layer to process without any delay. As soon as data exits the buffer, fresh data replaces it instantly. As a result, this data flow structure permits optimization to synchronize the computational process between the convolutional and pooling layers. However, fully connected layers cannot benefit from this optimization as they require the processing of all data from the previous layer, which accounts for less than 1% of the overall computation and has no impact on latency.

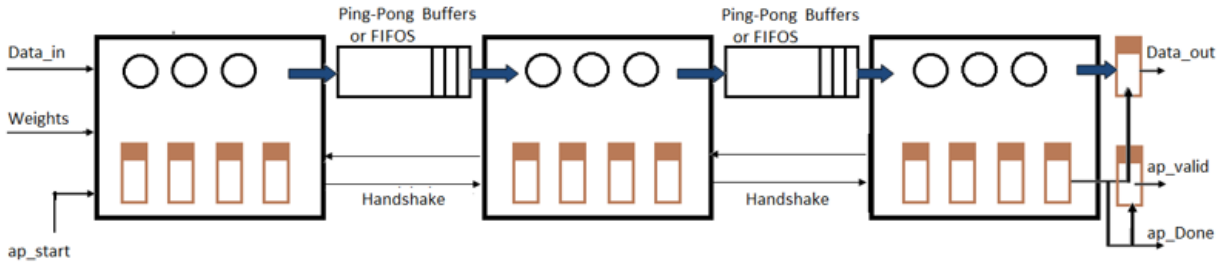


Fig. 6: Dataflow structure of the CNN IP core

3.2.2 Convolution layer optimization

The convolutional layer plays a pivotal role in image processing by employing filters to discern fundamental attributes within the input data of an image. This process entails a

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

tridimensional linear operation involving the multiplication of an array kernel with the input array window. The kernel is iteratively applied to the input array window across various positions, leading to the creation of tridimensional arrays referred to as feature maps (FMs), which the subsequent layer will use. As shown in Figure 7, Nix and Niy represent the numbers of inpDut feature maps (IFMs) and output feature maps (OFMs), respectively. The dimensions of the IFMs are denoted as Nix and Niy, while the dimensions of the OFMs are denoted as Nox and Noy. Additionally, Nk signifies the number of kernels with dimensions x×y.

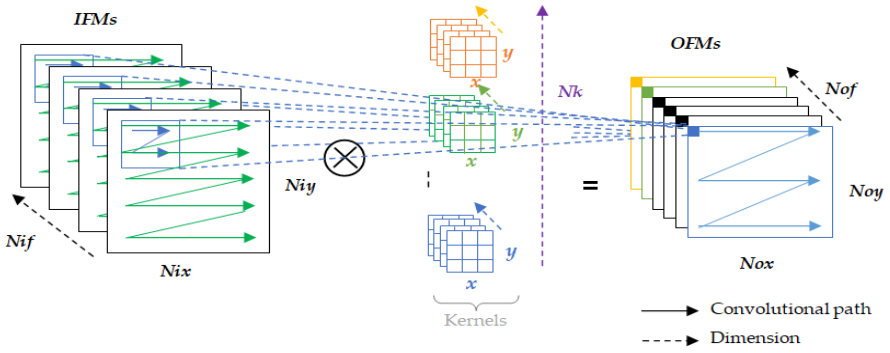


Fig. 7: Convolutional operations and dimensions

Within the convolutional layer, a hierarchical arrangement of four nested loops is employed: Loop 1 undertakes the convolutional operation by convolving the kernel values with their corresponding window values within the Input Feature Maps (IFMs). Subsequently, Loop 2 traverses through the depth dimension, denoted as Nif, of the IFMs. Loop 3 systematically traverses the spatial dimensions Nix and Niy of the Feature Maps (FMs), while Loop 4 comprehensively scans the dimension Nof pertaining to the Output Feature Maps (OFMs). The comprehensive depiction of the convolutional process is elucidated through Equation (1) and Algorithm 1.

$$(\boldsymbol{no}, \boldsymbol{k}, \boldsymbol{m}) = \sum_{\boldsymbol{ni}=1}^{\boldsymbol{Nif}} \sum_{\boldsymbol{J}=1}^{\boldsymbol{y}} \sum_{\boldsymbol{I}=1}^{\boldsymbol{x}} \boldsymbol{Out}_{\boldsymbol{p}-1}(\boldsymbol{ni}, \boldsymbol{m} \times \boldsymbol{S} + \boldsymbol{x},\ \boldsymbol{k} \times \boldsymbol{S} + \boldsymbol{y}) \times \boldsymbol{weight}(\boldsymbol{ni}, \boldsymbol{no}, \boldsymbol{x}, \boldsymbol{y}) + \boldsymbol{Bias}(\boldsymbol{no}) \quad \textbf{(1)}$$

In the given equation, "no" represents a value between 1 and Nof, while "S" denotes the sliding stride. For a better understanding of the computational operations involved in the convolutional layer, refer to Algorithm 1, which provides a detailed pseudo-code.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

Convolutional Pseudo-code

```
Nof_FMs:                                    // Loop 4
  For (no=0;  no<Nof ; no++)   {
   Row_output:                              // Loop 3 Rows
     For (k=0 ; k<Noy ; k=+S)     {
       Col_output:                          // Loop 3 Cols
         For (m=0 ; m<Nox ; m=+ S)   {
          Nif_FMs :                         // Loop 2
           For (ni=0 ; ni<Nif ; ni++)     {
             Row_Kernel:                    // Loop 1 Rows
               For (J=0 ; J < y ; J++)       {
                 Col_Kernel:                // Loop 1 Cols
                   For (I=0 ; I < x ; I++)       {
        Outp(no; k, m) += Outp-1(ni;  k+ I, m + J) × weightp-1(ni, no, I, J)
    }
                                        }
                                    }
                            Outp (no; k, m) = Outp (no; k, m) + bias(no)
                                }
                            }
                        }
```

Algo. 1: The pseudo-code of convolutional layer

As shown in the previous Figure and algorithm, the operations of the convolutional layer are based on nested loops that perform additions and multiplications over arrays. Nested loop and array partitioning optimization techniques are used to increase the computational efficiency of the convolutional layer, which improves computational throughput and latency while optimizing hardware resources, and then accelerates data access to enable pipelining. Nested loop optimization techniques include loop unrolling and function pipelining techniques. Therefore, a trade-off between latency and hardware resources must be made with the above optimizations.

Figure 8.a shows the entire unrolling of Loop 1, which is the innermost loop corresponding to the parallel processing of multiple hardware units of the same size as the $x \times y$ of the kernel. However, the data flow must be parallel to ensure that these units have enough processing data. Overuse of unrolling loops and loop pipelining optimizations, on the other hand, not only speeds up the network, but also creates bottlenecks that require data access organization. Figure 8.b illustrates the functions and units have enough processing data. Overuse of unrolling loops and loop pipelining optimizations, on the other hand, not only speeds up the network, but also creates bottlenecks that require data access organization. Figure 8.b illustrates the functions and loops pipelining.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

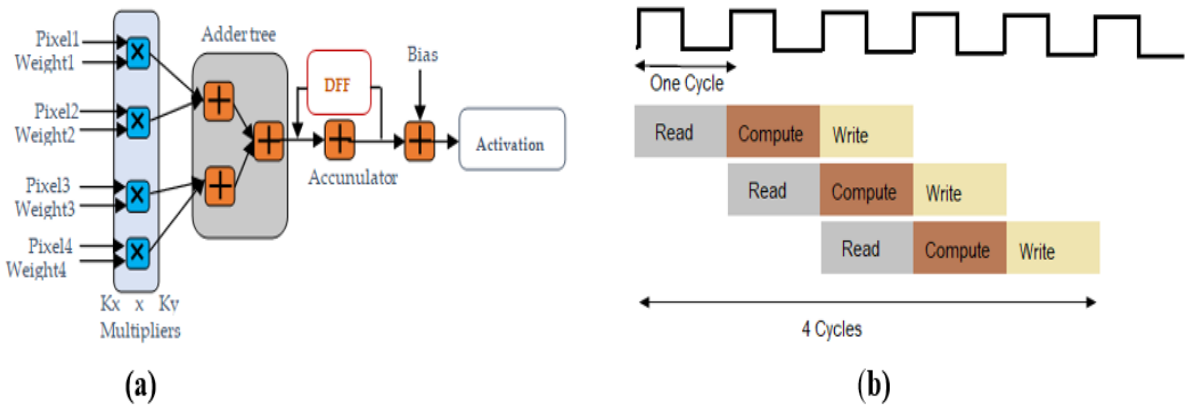


Fig. 8: Optimization techniques; (a) Unrolling Loop 1; (b) pipelining functions and loops

Based on the above-mentioned information, our approach is to first implement the IP core of the CNN in a sequential structure and then in a dataflow structure to highlight the contribution of each one. Then we use the dataflow structure to progressively apply optimization techniques using the ***# Pragma*** optimization directives available in the HLS tools [29]. The different design options are compared using metric analysis and synthesis tools. Table 1 summarizes the different design options adopted for CNN's IP core design.

Table 1: Design optimization steps

	Unroll	Pipeline	Unroll	Pipeline	Unroll	Pipeline	Unroll	Pipeline	Pipeline	Portioning	Sequential	Dataflow
	Loop1		Loop2		Loop3		Loop4		Inputs	Arrays	Structure	Structure
Baseline											✓	
Dataflow									✓			✓
Design_1										✓		✓
Design_2	✓									✓		✓
Design_3	✓								✓	✓		✓
Design_4	✓	✓	✓				✓		✓	✓		✓
Design_5	✓	✓	✓		✓	✓			✓	✓		✓
Design_6	✓	✓	✓		✓			✓	✓	✓		✓

4. RESULTS AND DISCUSSION

In this section, the primary outcomes of the project are presented and analyzed. The first sub-section covers the results obtained during the training phase, while the second sub-section delves into the main results of the implementation of the CNN IP core design.

4.1. Training Phase

We conducted ten epochs of training for the CNN model and utilized the learning history to evaluate both models by plotting the accuracy and loss curves, as illustrated in Figure 9. According to the curves, the model displays rapid learning with a linear increase in

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

accuracy during the initial two epochs. Subsequently, the curve starts to flatten, indicating that the learning process stabilizes and requires fewer epochs to complete the model's training.

Additionally, the validation demonstrates linear growth over the first two epochs and then starts to level off until it follows the accuracy curve entirely from the seventh epoch onwards. This suggests that the model is functioning effectively, learning quickly in the initial stages before stabilizing. However, in the test set, the loss drops gradually for the first four epochs and then flattens out for the remaining epochs, indicating that our CNN model is generalizing well to data that it has not encountered before.

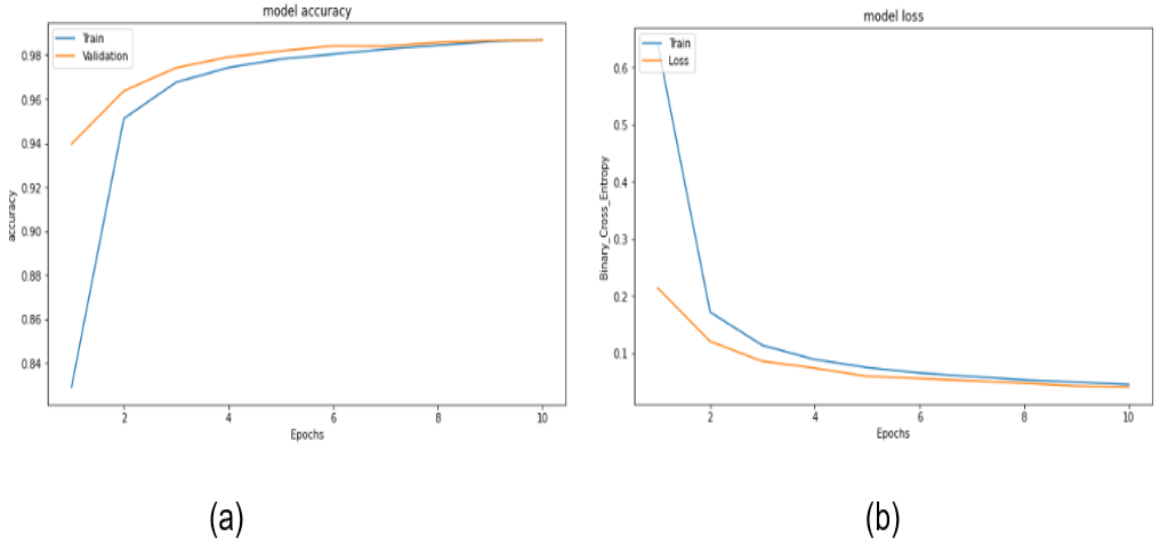


Fig. 9: Training performance curves within 10 epochs: (a) model accuracy evolution within 10 epochs; (b) model loss evolution within 10 epochs

We also generated a concise overview of the training phase's outcomes using the confusion matrix, visually depicted in Figure 10. This facilitated a comprehensive statistical evaluation of the F-Score parameter, encompassing precision and recall metrics. Figure 11 illustrates that while labels 2, 3, 6, and 9 exhibit F-Scores below the mean, they still achieve F-Scores surpassing 97%, indicating a highly satisfactory performance level.

For instance, the prediction pertaining to label 8, manifesting the weakest F-Score, can be attributed to the occurrence of 20 false positives in relation to this label within the entirety of true labels.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

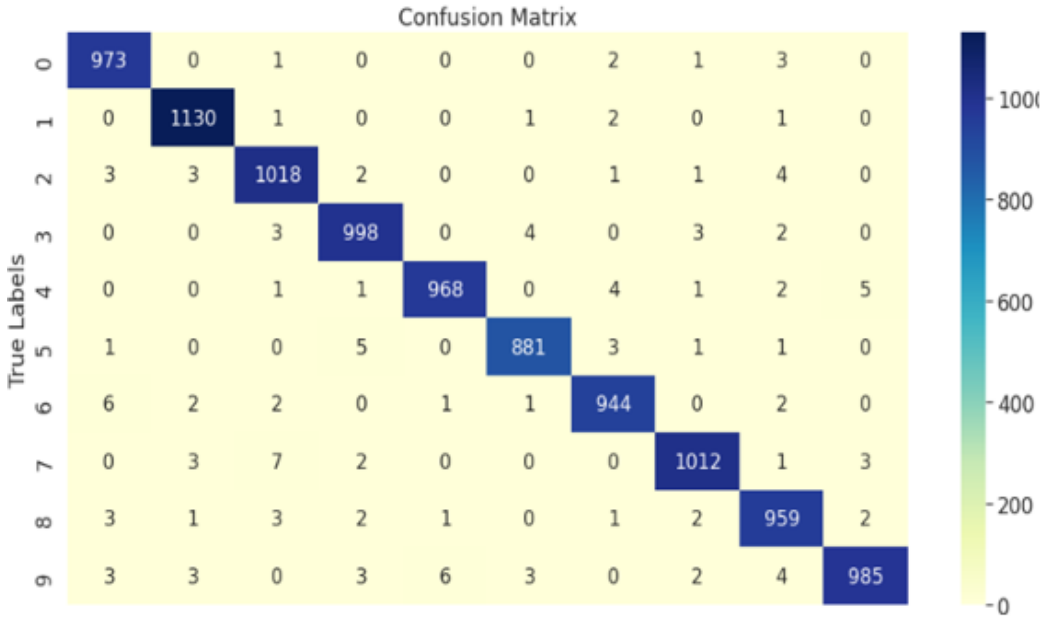


Fig. 10: The confusion matrix

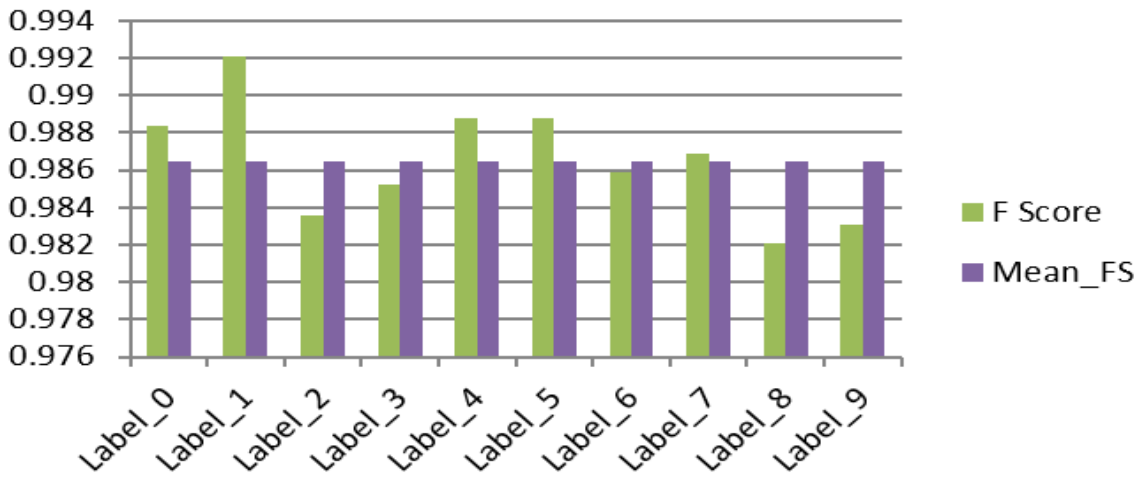


Fig. 11: F-Score histogram

4.2 Hardware Implementation Phase

In Figure 12.a, the analysis metrics for the baseline implementation's performance are presented in a sequential structure without optimization. On the other hand, Figure 12.b displays the analysis metrics of the design_1 implementation's performance in a dataflow structure without optimization. By comparing the two figures, it becomes evident that design_1 performs better than the baseline implementation in terms of latency. This is because design_1 leveraged parallelism between layers due to the dataflow structure, while the baseline implementation was constrained by its sequential structure.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
lenet5	-	145	9	48132	19089	4880274~4899874	4880275 ~ 4899875	none
conv2D_layer3	-	45	1	23243	4100	3729722~3735482	3729722 ~ 3735482	none
conv2D_layer2	-	3	1	11929	3710	581854~587854	581854 ~ 587854	none
conv2D_layer1	-	1	1	4390	3312	299028~306868	299028 ~ 306868	none
pool_layer2	-	0	0	5177	3147	14061	14061	none
pool_layer1	-	0	0	2761	3189	12274	12274	none
fc_layer1	-	81	2	176	363	208753	208753	none
fc_layer2	-	12	2	170	361	30649	30649	none
fc_layer3	-	1	2	145	330	1869	1869	none

(a)

	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
lenet5	-	145	9	48124	18976	3735112~3740872	3729723 ~ 3735483	dataflow
conv2D_layer3	-	45	1	23244	4111	3729722~3735482	3729722 ~ 3735482	none
conv2D_layer2	-	3	1	11928	3720	581854~587854	581854 ~ 587854	none
conv2D_layer1	-	1	1	4391	3323	299028~306868	299028 ~ 306868	none
pool_layer2	-	0	0	5178	3158	14061	14061	none
pool_layer1	-	0	0	2762	3200	12274	12274	none
fc_layer1	-	81	2	177	372	208753	208753	none
fc_layer2	-	12	2	171	370	30649	30649	none
fc_layer3	-	1	2	146	339	1869	1869	none
Loop_lenet5_label0_p	-	1	0	26	81	2049	2049	none

(b)

Fig. 12: Analysis metrics of sequential and dataflow Structure: (a) analysis metrics of the design in sequential structure; (b) analysis metrics of the design in dataflow structure

We used HLS to produce various reports such as analysis, synthesis, and comparison reports. These reports helped us gain a complete understanding of each design option and make well-informed decisions based on their advantages and disadvantages. Upon analyzing the reports, we arrived at a conclusion based on the statistical analysis illustrated in Figure 15. The figures demonstrate that designs 4, 5, and 6 outperform the others in terms of latency and hardware resource utilization. Designs 6 and 7 have better latency but still use more resources than design 4. Therefore, when choosing a design option, it's crucial to consider the balance between latency and resource usage. Despite the improvements in latency for designs 5 and 6, design 4 remains the most efficient overall for our requirements. We used HLS to produce various reports such as analysis, synthesis, and comparison reports. These reports helped us gain a complete understanding of each design option and make well-informed decisions based on their advantages and disadvantages. Upon analyzing the reports, we arrived at a conclusion based on the statistical analysis illustrated in Figure 13. The figures demonstrate that designs 4, 5, and 6 outperform the others in terms of latency and hardware resource utilization. Designs 6 and 7 have better latency but still use more resources than design

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

4. Therefore, when choosing a design option, it's crucial to consider the balance between latency and resource usage. Despite the improvements in latency for designs 5 and 6, design 4 remains the most efficient overall for our requirements.

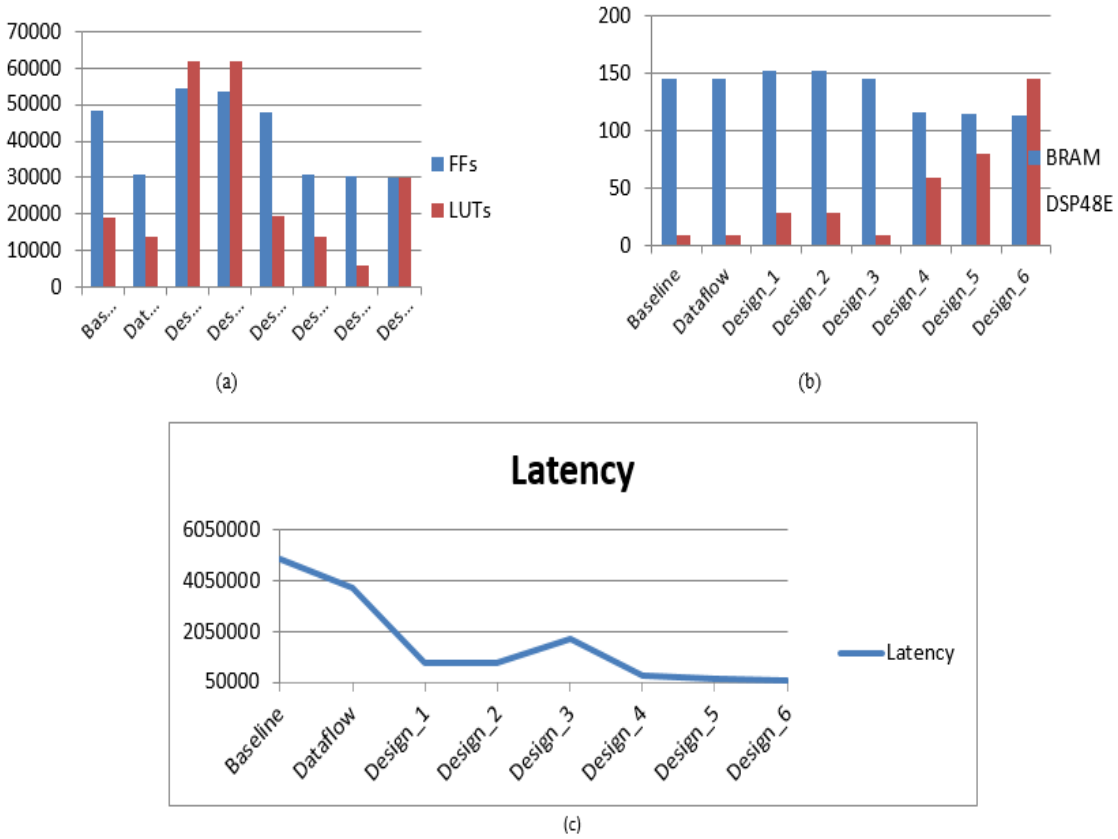


Fig. 13: Utilization and performance estimates of all implemented options: (a) utilization estimates of FFs and LUTS; (b) utilization estimates of BRAMs and DSP48E; (c) Performance estimates.

Table 2 presents a comparative study of the latency performance obtained by our approach for a single image with those of [30], [31], and [32]. We achieve 14.3x and 6.3x speedups for the first two works, respectively, by applying optimization directives such as function pipelining, loop unrolling, and array partitioning. Concerning the third work, the results are close, and the difference is explained by the fact that they used a fixed-16 representation while ours was a fixed-25.

Table 2: Performance comparison of the LeNet5 with previous works

	[30]	[31]	[32]	[Ours]
CNN Model	Lenet5	Lenet5	Lenet5	Lenet5
Platform	ZC706	ZC709	ZC706	VC707
Precision	Fixed-25	Fixed-8-16	Fixed-16	Fixed-25
Frequency (Mhz)	100	100	100	100
Latency	3ms	1.318ms	175.7µs	299.3µs

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

5. CONCLUSIONS

Our work involved exploring and implementing Convolutional Neural Networks (CNNs) on a Field-Programmable Gate Array (FPGA). To achieve this, we followed a two-phase approach. The first phase was initial training, which we carried out using relevant libraries in Python. The second phase was hardware implementation on an FPGA-based High-Level Synthesis (HLS) platform. We used the open MNIST dataset for our experimental results, which showed that our training phase implementation was highly effective, with an accuracy rate exceeding 97%. For hardware implementation, we applied filters derived from the MNIST dataset during the training phase. We leveraged the FPGA's reconfigurability to apply various HLS optimization directives to the design, and through careful assessment and comparison of various metrics using the synthesis and analysis capabilities of the HLS platform, we identified the most optimal performance configuration, with a latency of 299.3μs. Our research presents opportunities for improvement and expansion. We propose integrating dynamic data representation into deeper neural networks to reduce memory usage and increase latency and throughput. We also see potential in exploring the synergies of combining OpenCV with HLS for video processing applications, such as object detection and person tracking.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Acknowledgements

This work didn't receive any external funding.

References

1) Abbas, A., Abdelsamea, M. M., Gaber, M. M. "Classification of COVID-19 in chest X-ray images using DeTraC deep convolutional neural network", Applied Intelligence, 51(2), pp. 854–864, 2021. https://doi.org/10.3390 /s21041492

2) Kervadec, H., Dolz, J., Tang, M., Granger, E., Boykov, Y., Ben Ayed, I. "Constrained-CNN losses for weakly supervised segmentation", Medical Image Analysis, 54, pp. 88-99, 2019. https://doi.org/10.1016/j.media.2019.02.009

3) Li, G., Huang, Y., Chen, Z., Chesser, G. D., Jr.; Purswell, J. L., Linhoss, J., Zhao, Y. "Practices and Applications of Convolutional Neural Network-Based Computer Vision Systems in Animal Farming: A Review", Sensors, 21, 1492., 2021. https://doi.org/10.3390 /s21041492

4) Adjabi, I., Ouahabi, A., Benzaoui, A., Taleb–Ahmed, "A. Past, present, and future of face recognition, A Review", Electronics, 9, 1188, 2020. https://doi.org/10.3390/electronics9081.188

5) Abu Al-Haija, Q., Zein-Sabatto, S. "An Efficient Deep-Learning-Based Detection and Classification System for Cyber-Attacks", in IoT Communication Networks. Electronics, 9, 2152, 2020. https://doi.org/10.339 0/electronics9122152

6) Kattenborn,T., Leitloff, J., Schiefer, F., Hinz, S. "Review on Convolutional Neural Networks (CNN) in vegetation remote sensing", ISPRS Journal of Photogrammetry and Remote Sensing, 173,24-49., 2021. https://doi.org/10.1016/j.isprsjprs.2020.12.010

7) Wan, S., Goudos, S. "Faster R-CNN for multi-class fruit detection using a robotic vision system.Computer Networks", 168, 107036., 2020 https://doi.org/10.1016/j.comnet.2019.107036

ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

8) LeCun, Y., Bottou, L., Bengio,Y., Haffner, P. "Gradient-based learning applied to document recognition", Proceedings of the IEEE, 86(11), 2278-2324., 1998. https://doi.org/10.1109/5.726791

9) Krizhevsky, A.; Sutskever, I.; Hinton, G.E. "ImageNet classification with deep convolutional neural networks", In Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS), Lake Tahoe, NV, USA, pp. 1097–1105, 3–6 December 2012. https://doi.org/10.1145/3065386

10) Simonyan, K., Zisserman, A. "Very deep convolutional networks for large-scale image recognition", In Proceedings of the 2nd International Conference on Learning Representations (ICLR), Ban, AB, Canada,14–16 April 2014, https://doi.org/10.1109/ACPR.2015.7486599

11) He, K., Zhang, X., Ren, S., Sun, J. "Identity mappings in deep residual networks", In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 9908 LNCS,pp. 630–645, 2016. https://doi.org/10.48550/arXiv.1603.05027

12) Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Darrell, T. "Caffe: Convolutional architecture for fast feature embedding", In Proceedings of the 22ndACM international conference on Multimedia., Orlando, FL, USA, November 3-7, pp. 675-678, 2014. https://doi.org/10.48550/arXiv.1408.5093

13) Liu, Y., Wang, Y., Yu, R., Li, M., Sharma, V., Wang, Y. "Optimizing CNN model inference on CPUs", In Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC 19), Renton, WA, USA, ;pp. 1025-1040, July 10-12, 2019. https://doi.org/10.48550/arXiv.1809.02697

14) Zhang, C., Hosseini, S. A. H., Weingärtner, S., Uğurbil, K., Moeller, S., Akçakaya, M. "Optimized fast GPU implementation of robust artificial-neural-networks for k-space interpolation (RAKI) reconstruction", PLoS One,14(10), e0223315, 2019. https://doi.org/10.1371/journal.pone.0223315

15) Moolchandani, D., Kumar, A., Sarangi, S. R. "Accelerating CNN Inference on ASICs: A Survey",. Journal of Systems Architecture 113, 101887, 2021. https://doi.org/10.1016/j.sysarc.2020.101887

16) Ma, Y., Cao, Y., Vrudhula, S., Seo, J. S. "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks", In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, pp. 45-54, February 22-24, 2017. https://dl.acm.org/doi/10.1145/3020078.3021736

17) Shahshahani, M., Goswami, P., Bhatia, D. "Memory optimization techniques for FPGA based CNN implementations", In Proceedings of the 2018 IEEE 13thDallas Circuits and Systems Conference (DCAS), Dallas, TX, USA, pp. 1-6, 12 November 2018. https://doi.org/10.1109/TVLSI.2018.2815603

18) Zhang, N., Wei, X., Chen, H., Liu, W. "FPGA implementation for CNN-based optical remote sensing object detection", Electronics 10(3), 282, 2021. https://doi.org10.3390/electronics10030282

19) Zhao, R., Song, W., Zhang, W., Xing, T., Lin, J. H., Srivastava, M., Zhang, Z. "Accelerating binarized convolutional neural networks with software-programmable FPGAs", In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, pp. 15-24, February 22-24, 2017. https://doi.org/10.1145/3020078.3021741

20) Ma, Y. "Hardware Acceleration of Deep Convolutional Neural Networks on FPGA", Doctoral dissertation, Arizona State University, USA, 2018.

21) Han, S., Pool, J., Tran, J., Dally, W. J. "Learning both weights and connections for efficient neural networks", In Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS), Montreal, Canada, pp.1135–1143, 07-12 December 2015. https://doi.org/10.48550/arXiv.1506.02626

22) Han, S., Mao, H., Dally, W. J. "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding", In Proceedings of the 2016 International Conference on

ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 10-2023
DOI: 10.5281/zenodo.10016126

Learning Representations (ICLR), San Juan, Puerto Rico, 02-04 May 2016. https://doi.org/10.48550/arXiv.1510.00149

23) Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., Temam, O. "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning", ACM SIGARCH Computer Architecture News, 42(1), pp. 269-284, 2014. https://doi.org/10.1145/2541940.2541967

24) Denton, E., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R. "Exploiting linear structure within convolutional networks for efficient evaluation", In Proceedings of the 2014 27th International Conference on Neural Information Processing System(NIPS), Montreal, Canada, pp. 1269–1277, 8-13 December 2014. https://doi.org/10.5555/2968825.2968968

25) Golub, G. H., Van Loan, C. F. "Matrix Computations", Johns Hopkins University Press, The Mathematical Gazette, 83(498), pp. 556-557, 1999.

26) Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., Cong, J. "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks", In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 161–170, February 2015. https://doi.org/10.1145/2684746.2689060

27) Ma, Y., Cao, Y., Vrudhula, S., Seo, J. S. "An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks", In Proceedings of the 2017 27thInternational IEEE Conference on Field Programmable Logic and Applications (FPL), pp. 1-8, 04-08 September 2017. **https://doi.org/**10.23919/FPL.2017.8056824

28) Allaire, J. Chollet, F. "keras: R Interface to 'Keras. Version V2.4. 0, J. Open Source Softw", 2(4), 296. 2017.

29) Xilinx Inc., "Vivado Design Suite User Guide," Ug902, vol. 4, pp. 1–173, 2015. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug902-vivado-high-level-synthesis

30) Ghaffari, S., Sharifian, S. "FPGA-based convolutional neural network accelerator design using high level synthesis", In Proceedings of the 2016 2nd IEEE International Conference of Signal Processing and Intelligent Systems (ICSPIS), Tehran, Iran, pp. 1-6, 14-15 December 2016. **https://doi.org/**10.1109/ICSPIS.2016.7869873

31) Liu, Z., Dou, Y., Jiang, J., Xu, J. "Automatic code generation of convolutional neural networks in FPGA implementation", In Proceedings of the 2016 IEEE International Conference on Field-Programmable Technology (FPT), Xi'an, China, pp. 61-68, December 7-9, 2016. **https://doi.org/10.1109/FPT.2016.7929190**

32) Liu, X., Liu, D. H., Chen, D., Wu, C. "Resource and data optimization for hardware implementation of deep neural", In Proceedings of the 20th System Level Interconnect Prediction Workshop, San Francisco, California, USA, pp. 1-8, June 23, 2018. https://doi.org/10.1145/3225209.3225214