International Journal of Multidisciplinary Studies and Innovative Technologies

e-ISSN: 2602-4888
dergipark.org.tr/en/pub/ijmsit
Research Article

2022, 6 (1), 38-44
DOI: DOI: 10.36287/ijmsit.6.1.38
Received: April 25, 2022; Accepted: June 20, 2022

Efficient Hardware Optimization for CNN

Seda GUZEL AYDIN[1*] and Hasan Sakir BILGE [2]

[1*]*Electrical and Electronics Engineering, Bingol University, Bingol, Turkey (sgaydin@bingol.edu.tr) (ORCID: 0000-0001-8875-9705)*
[2] *Electrical and Electronics Engineering, Gazi University, Ankara, Turkey (bilge@gazi.edu.tr) (ORCID: 0000-0002-4945-0884)*

Abstract – Convolutional Neural Networks (CNN) architectures have been increasingly well-known for image processing applications such as object detection, and remote sensing. Some applications like these systems need to adopt CNN methods for real-time implementation. Embedded devices like Field Programmable Gate Arrays (FPGA) technologies are a favorable alternative to implementing CNN-based algorithms. However, FPGA has some drawbacks such as limited resources and bottlenecks, it is difficult and so crucial to map the whole CNN that has a high number of layers, on FPGA without any optimization. Therefore, hardware optimization techniques are compulsory. In this study, an FPGA-based CNN architecture using high-level synthesis (HLS) is demonstrated, and a synthesis report is created for Xilinx Zynq-7000 xc7z020-clg484-1 target FPGAs. By implementing the CNN architecture on an FPGA platform, the implemented architecture has been fastened. To improve the throughput, the proposed design is optimized for convolutional layers. The most important contribution of this study is to perform optimization on the convolution layer by unrolling kernels and input feature maps and examine the effects on throughput, latency, and hardware resources. In this study, throughput is 15.6 GOP/s for the first convolution layer. With the proposed method in the study, approximately x2.6 acceleration in terms of latency and throughput was achieved compared to the baseline design.

Keywords *– FPGA, Deep learning, Convolutional neural networks, Hardware optimization, HLS*

Citation: Guzel Aydin, S. and Bilge, H. Ş. (2022). Efficient Hardware Optimization for CNN, International Journal of Multidisciplinary Studies and Innovative Technologies, 6(1): 38-44.

I. INTRODUCTION

Because CNN-based methods have presented successful results over traditional methods, these networks consider a powerful tool in many areas, especially in image processing applications. They are just adapted to address numerous problems. Starting with the 7-layer Lenet-5 [1], the deep learning (DL) approaches now continue with more than a hundred layers. In the early days of deep learning networks, researchers aimed to design deeper and wider networks to increase the accuracy of the network [2,3]. By increasing the number of layers and establishing larger networks, it was possible to design networks with high accuracy. However, increasing the number of layers and designing larger networks has increased the workload of the platform used to run the network. In addition, the number of parameters in the used network has increased, which has increased the training and inference stages. This, in addition to the success of CNN's in solving problems, caused an increase in the workload of the undesirable platforms and caused the general-purpose platforms to be insufficient to operate such deep networks. As a consequence of the large number of parameters and the computational burden of CNNs, researchers began to explore ways to design sparse and smaller networks using more compact and fewer parameters [4].

For some applications such as vehicle tracking systems, mobile applications, and object tracking, networks designed for real-time operation are required. CNN-based networks require a high volume of parameters and lots of calculation processes. However, central processing units (CPU) are not much preferred for real-time applications. To address these challenges, new solutions for efficient hardware implementations have been researched recently. To meet this requirement, numerous research tries to CNN to adopt high-performance devices such as Graphics Processing Unit (GPU), and FPGA [5-7]. It is so hard to reveal the parallelism feature in CNN for real-time applications in the CPU. In recent years, DL-based algorithms have been implemented and accelerated on GPU platforms so that notable parallel computing capability and high memory bandwidth. Nevertheless, studies on GPU show that GPU usually consumes more power than FPGA which makes it inefficient and difficult to use in battery-powered devices. In conclusion, it can be said that the computing performance of GPU is noticeable but on the other hand power consumption is high [7]. Therefore, FPGA could be addressed as a potential solution approach for optimizing and accelerating CNNs.

In recent years, significant research has investigated the utilization of accelerated CNN employment on FPGA. The FPGA implementation for CNNs has concerned much regarded a consequence of its reconfigurability, high performance, and energy efficiency. However, CNNs networks are computation-intensive and memory-intensive algorithms so, these features have brought many challenges to CNN implementations on hardware. FPGA has some disadvantages such as limited resources and limited bandwidth. Therefore, implementing complex structures like DL algorithms on FPGA is so challenge problem. To address these problems, hardware optimization methods can be applied to network design on the platforms.

In the last few years, researchers have been investigating different optimization methods for the development of efficient acceleration of CNN systems on FPGAs. Studies are generally carried out for increasing parallelism, reducing power consumption, and reducing the number of resources used. The majority of the CNN application accelerators are intended on optimizing computation processing engines. Researchers in [8] used implemented YOLO2 on FPGA for optical remote sensing. They used a uniform module to implement multiple types of convolutions for the network. Researchers in [9] proposed a Roofline model to improve throughput for CNN. The Roofline model tries to determine the maximum performance the hardware can achieve for implemented algorithm according to two bounds. They investigate architecture for throughput and required memory bandwidth using unrolling and tilling techniques for optimizing design. They implement an accelerator using Vivado HLS with uniform loop unroll factors for different convolutional layers. The architecture purposed by [10] implemented a deep convolutional neural network accelerator on FPGA to run in real-time. Their platform consists of programmable logic and a programmable system. Both platforms shared the same memory (DDR3). They stated that the FPGA-based accelerator system they designed allows ~25x faster execution faster than the CPU-based implemented architecture. They focused on computation engine optimization; they were not optimized memory access problems. More recently, researchers in [11] used the operation chaining between the layer for data processing architecture for implementing CNN on De2i-150 board that received x18.04 acceleration rate than the baseline design. DSE was explored by different parameters such as processing time, and power consumption. In [12], researchers have employed loop optimization techniques for speeding up convolution operation to design CNN accelerator. They present the quantitative analysis for many design variables to optimize the design. In [13], researchers optimized their design by implementing multiple convolutional layer processors (CLP) to meet different conv layers for better resource utilization. In [14], researchers recommend two types of dedicated hardware structures. The first structure is suitable for the small-size CNN implementation that is designed with specific hardware for each layer. The second structure is one hardware model for each layer that is used multiple times for different layers. This approach uses low resource consumption; however, the throughput is low. There is a control block determining to use which operation. They reported that this structure can be modified for large networks because many layers use the same resources. In [15], researchers recommended an end-to-end CNN accelerator implementation based on FPGA. Their architecture maps the whole layer on one chip. In these model different layers can work concurrently in a pipelined structure therefore throughput can be increased. Streaming design architecture could reach the best throughput through a high parallelism strategy for each layer however this approach uses high resource-consuming In [16], researchers used the tilling technique to accelerate their deep neural network. The tilling technique is employed for reducing the memory bandwidth requirement at the inference state. Their proposed approaches reduce the memory bandwidth by 46.7% at the inference state.

FPGA-based CNN accelerator designs have some limitations that cause considerable challenges in performance and flexibility. The first of them is the limited number of resources such as Look up tables (LUTs), flip flops (FFs), Block Random Access Memory (BRAM), and Digital signal processing (DSP). CNNs have several computations that need a great number of hardware resources. Another challenge is the bandwidth. Limited bandwidth will cause a bottleneck during data exchange between off-chip memory and FPGA, preventing the system from operating at high performance. These challenges motivate researchers to design more optimal systems through a series of hardware optimization approaches considering accelerated CNN on FPGA for designing the high-performance inference phase of CNN. The main contribution of this paper is to perform optimization on the convolution layer by unrolling kernels and input feature maps and examine the effects on throughput, latency, and hardware resources. In this study, throughput is 15.6 GOP/s for the first convolution layer. With the proposed method in the study, approximately x2.6 acceleration in terms of latency and throughput was achieved compared to the baseline design. The rest of this paper is organized as follows. In section 2, brief information about CNN is given. Parallelism methods that can be used for hardware designs are explained in section 3. Section 4 shows the details of the methods used in the study, the hardware implementation, and the results. Section 5 deals with the conclusions.

II. BACKGROUND

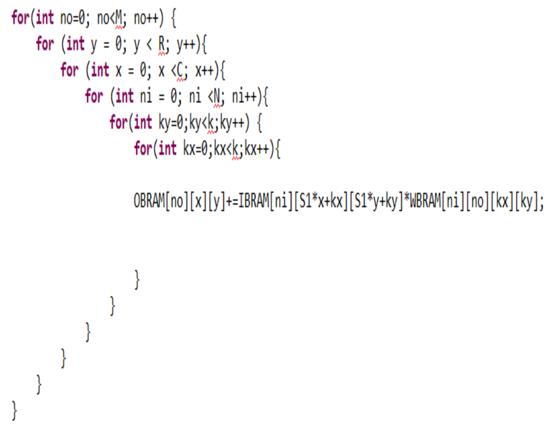
In this section, a typical CNN architecture is briefly described.

A. The architecture of Convolutional Neural Networks

A classical CNN is a multi-layer pattern that contains the convolutional layer (Conv layer), activation layer, pooling layer (PL), and fully connected (FC) layer.

1) Convolutional layer

Conv layer is the layer where the convolution process takes place, so this layer is the most important layer of the CNN. The convolution operation performed on the image takes different kernel filters, shifts them on the input feature maps (IFM) and creates an output feature map (OFM). OFM data resulting from the operations performed in this layer form IFMs for the next layer.



```
for(int no=0; no<M; no++) {
    for (int y = 0; y < R; y++){
        for (int x = 0; x <C; x++){
            for (int ni = 0; ni <N; ni++){
                for(int ky=0;ky<k;ky++) {
                    for(int kx=0;kx<k;kx++){

                    OBRAM[no][x][y]+=IBRAM[ni][S1*x+kx][S1*y+ky]*WBRAM[ni][no][kx][ky];


                    }
                }
            }
        }
    }
}
```

Fig. 1. Pseudo-code of convolution operation for CNN

The filter values used here are determined during the training. Fig. 1 shows the pseudo-code of convolution operation for

CNN. Fig. 2 is demonstrated convolution operation with multiple channel input and multiple kernels. The number of convolution filters is denoted by M, equivalent to the number of output feature maps. The size of the kernels is demonstrated by kxk. The number of channels, height, and width for IFM are denoted by $N, H, and\ W$ respectively. The height and width for OFM are denoted by $C, and\ R$ respectively.

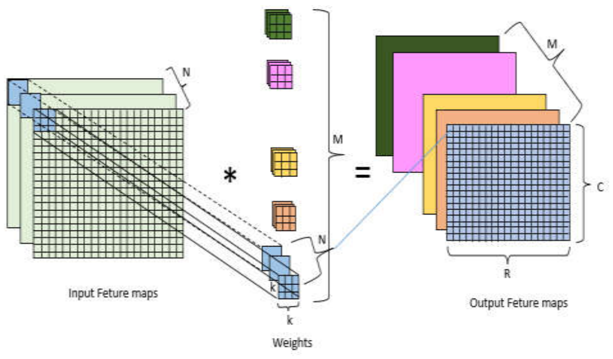


Fig. 2. Convolution operation with multiple channel input

Stride operation enables a dimension reduction of the convolution result by avoiding filters implemented in the whole of the IFMs. Stride (s) parameter defined how many steps are moved in each step-in convolution operation. The padding (p) operation allows remaining the same dimension of the resulting image through the process of adding zeros to the input matrix. The following OFM dimension can be calculated in (1):

$$OFM_{R,C} = \frac{W,H+(2\ x\ p)-k}{s} + 1 \qquad (1)$$

The computational complexity (CC) can be denoted by #OP for each convolutional layer of the network could be measured based (2).

$$\#\mathrm{OP}^{l} = M^{l}xN^{l}xR^{l}xC^{l}xk^{l}xk^{l} \qquad (2)$$

2) Activation layer

After the convolution process is finished, the results obtained are passed through the activation function. In the convolution layer, the inputs are multiplied by the weights, and then added together with the bias values. The output signal generated as a result of this process is a simple linear function. This result is converted into a non-linear structure by applying the activation function. Usually, nonlinear and differentiable activation functions are preferred, such as step function, sigmoid, hyperbolic tangent (Tanh), Rectified Linear Unit (ReLU), Softmax etc. The sigmoid function is one of the most commonly used activation functions. It is differentiable. However, due to the gradient vanish problem, the maximum performance of the network using this function could not be achieved so, an alternative activation function was sought to find a solution to this problem. The Tanh converts the input value to the hyperbolic tangent of the angle it. The interval of this function, which has a very similar structure to the sigmoid function, is defined as (-1,+1). However, since the problem of gradients dying at the ends of this function continues, different functions are preferred as an alternative solution. In addition, since the constant Euler's number (e) is used in these two functions, it causes computational complexity in hardware designs and this is a problem in hardware implementations. The ReLU is the most used activation function in accelerating hardware. It rectifies the linear unit, and activating some neurons. The computational load is less than the sigmoid and hyperbolic tangent functions, making it more preferred in multilayer networks.

3) Pooling layer

Usually used after the convolution layer, the purpose of this layer is to reduce the input size for the next convolution layer to reduce the complexity of further layers. The depth (channel) of the input data does not affect the pooling process. The pooling operation occurs as follows: The input feature map is partitioned into smaller rectangles and transformed the defined function value inside that small rectangle [23]. Frequently, max-pooling and average pooling methods are used for pooling operations. Max-pooling returns the max value of the sub-region, and average pooling returns the average of values in the sub-region.

4) Fully connected layer

Like the layers in classical neural networks, each neuron in the FC layer is connected to all the neurons after it. Therefore, these layers are also referred to as densely connected layers. Since there are too many parameters in these layers, they are the layers that consume the most memory for storing these parameters [24].

III. ACCELERATION METHODS FOR CNN

A. Different parallelism structures

CNN architecture has streaming structures. That is, architecture consists of interconnected layers that work one after the other. Therefore, different parallelism methods can be applied to these architectures. [18-20] have exploited several levels of parallelism methods that can be applied to CNN architecture. These are task-level parallelism (batch parallelism), layer-level parallelism, and loop-level parallelism. Task-level parallelism can be defined as the simultaneous execution of two or more inference prediction tasks during the inference phase of the designed model by using efficiently on-chip memory. Layer-level parallelism (Inter-layer) can be achieved depending on the pipeline strategy. In the inference phase, each layer receives data from the previous layer as input. Because the layers are data-dependent on each other, the layers can't run completely in parallel. The model can be accelerated by using the pipeline structure instead of parallelism through launching layer (l) before ending the execution of $(l-1)$. Loop-level parallelism can be defined as kernel-level parallelism. To implement convolution operation MxN kernels are employed. Each of the kernels operate can be executed in parallel ways theoretically because all the convolutions' operations are independent. However, practically, limited computation resources and memory bandwidth does not allow all processes to be performed in parallel on FPGA. On account of this, loop-level parallelism can be implemented in many different ways according to different loop unrolling strategies. Loop-level parallelism is explored in detail in the next section.

The implementation of convolution operation enables numerous techniques for parallelism. However, due to the FPGA resource limitation, exploiting a full parallelism pattern for overall CNN is impossible. In some cases, even just one convolution layer can't run completely in parallel. Therefore, partial parallelism is employed by using the *unrolling* factor and *tilling* factor.

Due to the fact that convolution operations take occur in the convolution layer, most operations, about 90% of all operations are performed in the convolutional layers. Convolution operation involves multiple multiplies and accumulates (MAC) operations with six nested loops. Therefore, an effective convolution acceleration optimization approaches considerably affects the performance of a hardware-based CNN accelerator. Nested loop optimization techniques, e.g. loop unrolling, tiling, and interchange, or only tune some of the design variables after the accelerator architecture and dataflow are already fixed. Without fully studying the convolution loop optimization before the hardware design phase, the resulting accelerator can hardly exploit the data reuse and manage data movement efficiently.

IV. IMPLEMENTATION OF ACCELERATED CNN

In this study, optimization in the Conv layer is done by unrolling kernel and IFM to decrease latency and increase throughput. Baseline design is used to compare the result of the suggested design in RTL syntheses. Comparison is made by means of latency, throughput, and resource utilization.

The framework of created accelerator system (AC) is shown in Fig. 3. AC system consists of two parts that are Input layer and Conv layer. The input layer is the layer responsible for storing data in BRAM units. The conv layer is the unit where the calculations are made. The Conv layer consists of Processing Elements (PEs) units, buffers, and OFMs. PE is the basic computation unit to perform convolution operations. The number of PE can be determined by unrolling factor N. Unrolling kernels determine the multiplication and adder units.

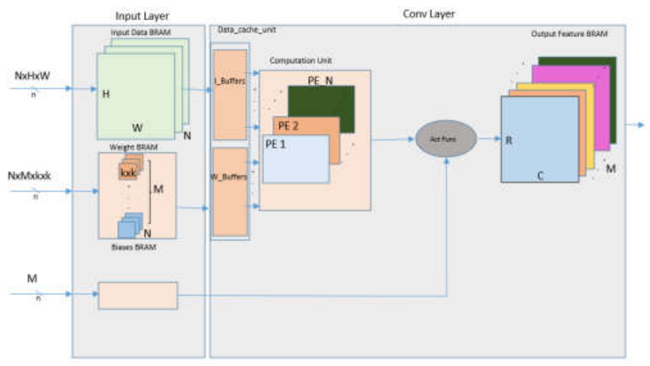


Fig. 3. Proposed accelerator framework

The convolution operation for a three-dimensional image consists of six nested for loops. All of these loops can be unrolled individually or together. Unrolling separately or together causes different parallelism operations. Loop-level parallelism can be implemented in many ways according to different loop unrolling strategies. In this section, the unroll operation performed in different ways for the convolution operation and the hardware equivalent of these operations are examined.

A. Data cache structure

During the inference state of a CNN, it is often necessary to read a large amount of data. The limitation of bandwidth and on-chip memory capacity makes it hard to obtain all data simultaneously. Modern FPGA provides large amounts of on-chip memory which of it is implemented as block RAM (BRAM) where embedded within the fabric. For the inference stage, CNN required memory required for parameters and data which are input data, bias, weight data, and the output of intermediate layers data. The required memory (RM) for the input data can be calculated

$$RM\_IFM = HxWxNxdata\_bit\_widht$$
$$RM\_OFM = RxCxMxdata\_bit\_widht$$
$$RM\_W = kxkxMxNxdata\_bit\_widht$$

The block RAM in Xilinx Zynq-7000 target FPGAs store up to 36 Kbits of data and each 36 Kb block RAM can be constructed as a 64K x 1 (when cascaded with an adjacent 36 Kb block RAM), 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, 1K x 36, or 512 x 72 in simple dual-port mode.

Data access time is one of the main challenges in FPGA implementation, therefore, data access pattern is crucial. The data feeding the parallel processing units should also come to these units in parallel. BRAM partitioning method is used for the parallel reception of data on BRAM [22,23]. In order to cache required data in parallel, the BRAMs must be completely partitioned and all data must be recorded in registers. By means of partitioning BRAM completely, the data can be fed to the computation unit in parallel and at the same time the use of BRAM will be reduced, but there may not be enough resources to save this much data to the registers. Therefore, using this method is not suitable for large networks. Using a temporal memory storage controller provides using low resource and low latency. In this study, the data are given to the computational units in parallel using the buffer structure. Data cache structure consists of three units as shown in Fig. 4. The first is the BRAMs where all acquired data are stored on. The second unit is the line buffer (LB) unit. The last is the window buffer (WB) unit. This unit is used to temporarily store data needed for convolution operation in the PE units or computation units. PE units access the same data multiple times because the convolution operation is the iterative algorithm. To avoid reading the same data multiple times the WB could shift all data and replaces the used data which are not required anymore with new data [21]. WB has a kxk register for kernel data and kxk register for the IFM data. It stores the data coming from the LB. LBs are temporary memory storage buffers that can cache rows or columns of IFM data. LB units are created based on shift register logic which shifts all data via WB.

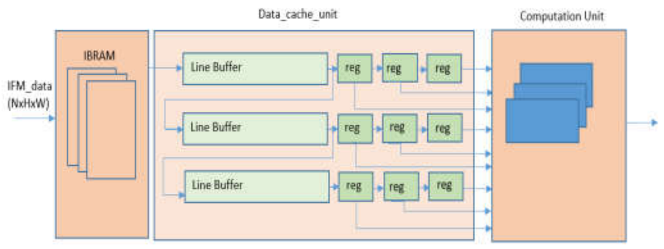


Fig. 4. Data cache structure

B. Baseline design

In the baseline design loops are rolled. That means that one copy of the loop body is synthesized for iterations and re-used for each iteration. This causes all operations to occur in a sequential manner. Sequencing of processes will ensure that the number of resources used is less than designs with parallel processing that are made by unrolling loops. However, latency will be very high than in unrolled loop architecture because the operations are done sequentially. Through the unrolling nested loops, new hardware resources will be assigned for all operations that can be done in parallel, so that operations can be performed faster and in parallel.

Table 1. Parameters for first conv layer

N	M	R	C	kxk
3	6	25	25	5x5

The baseline design process and hardware structure corresponding to baseline design are shown in Fig. 5 and Fig. 6 respectively. In the study, the baseline model represents the situation in which no optimization has been made yet. Table 1 shows the parameters for implemented convolution layer. Only one module is used, and this module is used sequentially for all MAC operations to be made. In this design, the latency is very high, but the number of resources used is very low.

Trip count (TC) is used to define a minimum number of times a loop executes.

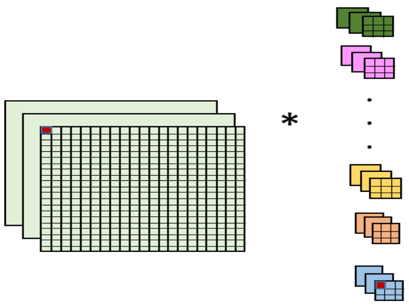


Fig. 5. Baseline design process

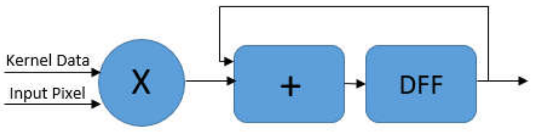
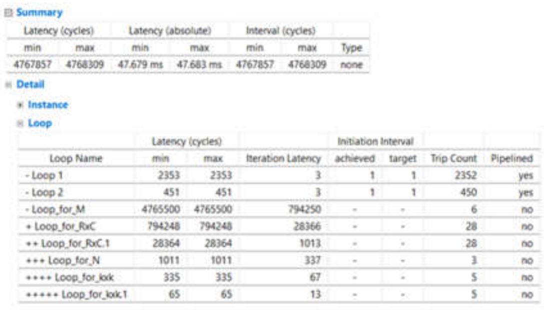


Fig. 6. Corresponding hardware architecture for the baseline design



Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
4767857	4768309	47.679 ms	47.683 ms	4767857	4768309	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	2353	2353	3	1	1	2352	yes
- Loop 2	451	451	3	1	1	450	yes
- Loop_for_M	4765500	4765500	794250	-	-	6	no
+ Loop_for_RxC	794248	794248	28366	-	-	28	no
++ Loop_for_RxC.1	28364	28364	1013	-	-	28	no
+++ Loop_for_N	1011	1011	337	-	-	3	no
++++ Loop_for_kxk	335	335	67	-	-	5	no
+++++ Loop_for_kxk.1	65	65	13	-	-	5	no

Fig. 7. A part of the synthesis report for the baseline design

In the baseline model, operations are performed sequentially. There is only one multiplication that can perform this structure therefore, it processes by taking one pixel from IFM set and one data from kernel sets at a time. It takes Nxkxk trip count to generate one pixel of one OFM. It takes NxRxkxk TC to generate one row of one OFM and NxRxCxkxk TC to generate one OFM. To generate all OFM it takes MxNxRxCxkxk TC.

#PE=1, #multipliers=1, #adders=1

$$Throughput = \frac{MxNxRxCxkxk}{latency}$$

The latency for baseline design is 47.683 ms. The baseline design uses 25 BRAMs and 5 DSP resources. Throughput is calculated 5.9GOP/s for baseline design. Fig. 7 shows the part of the synthesis report for baseline design.

C. Proposed Design: Unrolling kernels and IFM channels

In this case, operations are performed partially parallel to result in one pixel of OFM sets. The proposed design process and hardware structure corresponding to it are shown in Fig. 8 and Fig. 9 respectively.

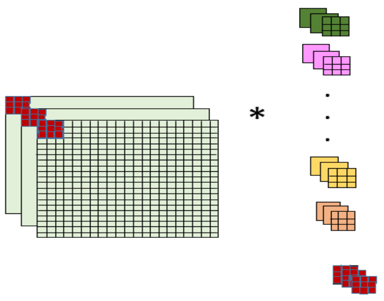


Fig. 8. Proposed design process

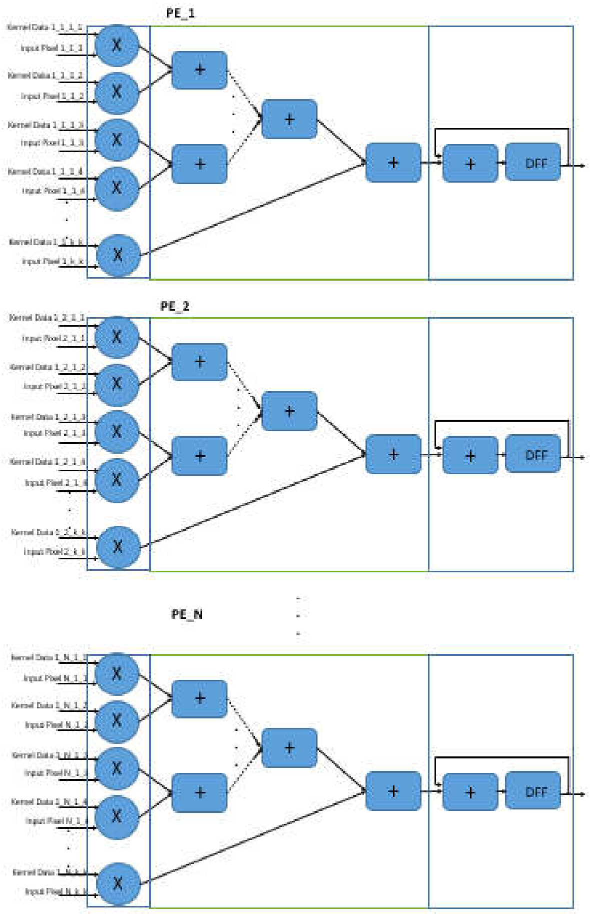
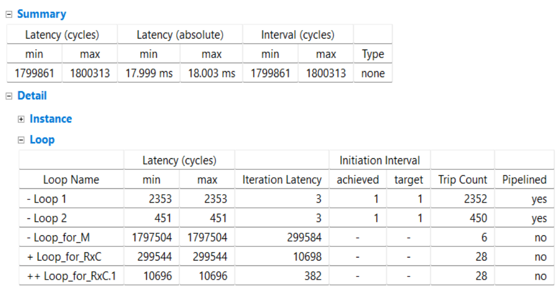


Fig. 9. Corresponding hardware architecture for the proposed design

In the proposed method, input data channels and kernels are unrolled. As a result of this process, N PE units are created for hardware implementation. In each PE unit, kxk multiplication is performed. Fig. 10 shows the part of the synthesis report for the proposed design.



Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1799861	1800313	17.999 ms	18.003 ms	1799861	1800313	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	2353	2353	3	1	1	2352	yes
- Loop 2	451	451	3	1	1	450	yes
- Loop_for_M	1797504	1797504	299584	-	-	6	no
+ Loop_for_RxC	299544	299544	10698	-	-	28	no
++ Loop_for_RxC.1	10696	10696	382	-	-	28	no

Fig. 10. A part of the synthesis report for the proposed design

There are Nxkxk multiplication can perform this structure therefore, it processes by taking Nxkxk pixel from IFM set and Nxkxk data from kernel sets at a time. It takes 1 TC to generate one pixel of one OFM. It takes R TC to generate one row of one OFM and RxC TC to generate one OFM. To generate all OFM it takes MxRxC TC.
#PE=N=3, #multipliers=kxkxN#adders=(kxk-1)xN

The latency for the proposed design is 18 ms. Proposed design use 26 BRAMs and 8 DSP resources. Throughput is calculated 15.6 GOP/s for the first convolution layer in proposed design.

V. CONCLUSION

FPGA technologies are a favorable alternative to implementing CNN-based algorithms due to their reconfigurability, high performance, and low energy usage characteristics. However, it is difficult to implement the whole CNN architecture on FPGA without any optimization because of limitations in terms of resources and bottlenecks. In this paper, an FPGA-based CNN architecture using high-level synthesis (HLS) is demonstrated and the implemented architecture has been fastened. To improve the throughput, the proposed design is optimized for convolutional layers by unrolling the kernel and IFM channel. The most important contribution of this study is to perform optimization on the convolution layer by unrolling kernels and input feature maps and examine the effects on throughput, latency, and hardware resources.

ACKNOWLEDGMENT

This research was supported by a grant from (121E393) TUBITAK (Türkiye Bilimsel ve Teknolojik Araştirma Kurumu). We thank the TUBITAK for their support of our research.

Authors' Contributions

The authors' contributions to the paper are equal.

Statement of Conflicts of Interest

There is no conflict of interest between the authors.

Statement of Research and Publication Ethics

The authors declare that this study complies with Research and Publication Ethics

REFERENCES

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv.org*, 2014, doi: 10.48550/arXiv.1409.1556.

[3] 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, May 2017, doi: 10.1145/3065386.

[4] M. Mikaeili and H. S. Bilge, "Estimating Rotation Angle and Transformation Matrix Between Consecutive Ultrasound Images Using Deep Learning," 2020 Medical Technologies Congress (TIPTEKNO), Nov. 2020, doi: 10.1109/tiptekno50054.2020.9299237.

[5] C. Huang, S. Ni and G. Chen, "A layer-based structured design of CNN on FPGA," *2017 IEEE 12th International Conference on ASIC (ASICON), 2017*, pp. 1037-1040, doi: 10.1109/ASICON.2017.8252656.

[6] W. A. Haque, S. Arefin, A. S. M. Shihavuddin, and M. A. Hasan, "DeepThin: A novel lightweight CNN architecture for traffic sign recognition without GPU requirements," *Expert Systems with Applications,* vol. 168, p. 114481, Apr. 2021, doi: 10.1016/j.eswa.2020.114481.

[7] Y. Hu, Y. Liu, and Z. Liu, "A Survey on Convolutional Neural Network Accelerators: GPU, FPGA and ASIC," *2022 14th International Conference on Computer Research and Development (ICCRD),* Jan. 2022, doi: 10.1109/iccrd54409.2022.9730377.

[8] N. Zhang, X. Wei, H. Chen, and W. Liu, "FPGA Implementation for CNN-Based Optical Remote Sensing Object Detection," *Electronics*, vol. 10, no. 3, p. 282, Jan. 2021, doi: 10.3390/electronics10030282.

[9] C, Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks." *In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays,* Monterey, CA, USA, 22–24 February 2015; pp. 161–170.

[10] A. Dundar, J. Jin, V. Gokhale, B. Krishnamurthy, A. Canziani, B. Martini, & E. Culurciello," Accelerating deep neural networks on mobile processor with embedded programmable logic." *In Neural information processing systems conference (NIPS).* 2013

[11] M. Arredondo-Velázquez, J. Diaz-Carmona, C. Torres-Huitzil, A. Padilla-Medina, and J. Prado-Olivarez, "A streaming architecture for Convolutional Neural Networks based on layer operations chaining," *Journal of Real-Time Image Processing,* vol. 17, no. 5, pp. 1715–1733, Jan. 2020, doi: 10.1007/s11554-019-00938-y.

[12] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 26, no. 7, pp. 1354–1367, Jul. 2018, doi: 10.1109/tvlsi.2018.2815603.

[13] Y. Shen, M. Ferdman and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA),* 2017, pp. 535-547, doi: 10.1145/3079856.3080221.

[14] S. Ghaffari and S. Sharifian, "FPGA-based convolutional neural network accelerator design using high level synthesize," *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS),* 2016, pp. 1-6, doi: 10.1109/ICSPIS.2016.7869873.

[15] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou and Lingli Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," *2016 26th International Conference on Field Programmable Logic and Applications (FPL),* 2016, pp. 1-9, doi: 10.1109/FPL.2016.7577308.

[16] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, Y. Xu, "Throughput-optimized fpga accelerator for deep convolutional neural networks." *ACM Trans. Reconfgurable Technol. Syst. (TRETS)* 10(3), 17, 2017

[17] Y. Zhou, J. Jiang, " An FPGA-based accelerator implementation for deep convolutional neural networks." *In Proceedings of the 2015 4th International Conference on Computer Science and Network Technology, ICCSNT 2015,* Harbin, China, 19–20 December2015; Volume 1, pp. 829–832.

[18] K. Abdelouahab, M. Pelcat, J. Serot, & F. Berry, "Accelerating CNN inference on FPGAs: A survey." arXiv preprint arXiv:1806.01683. 2018.

[19] K. Guo, S. Zeng, J. Yu, Y. Wang, & H. Yang" [DL] A survey of FPGA-based neural network inference accelerators." *ACM Transactions on Reconfigurable Technology and Systems (TRETS),* 12(1), 1-26.2019.

[20] R. Ayachi, Y. Said, & A. Abdelali, "Optimizing Neural Networks for Efficient FPGA Implementation: A Survey." *Archives of Computational Methods in Engineering,* 28(7), 4537–4547. 2021.

[21] G. Muhsin "A Comparative Study between RTL and HLS for Image Processing Applications with FPGAs" thesis, University of California, San Diego, Master of Science.

[22] Vivado Design Suite User Guide High-Level Synthesis Documentation Portal. (2022). Retrieved May 17, 2022, from Xilinx.com website: https://docs.xilinx.com/v/u/2018.3-English/ug902-vivado-high-level-synthesis

[23] S. Guzel Aydin and H. S. Bilge, "FPGA -Based Implementation of Convolutional Layer Accelerator Part for CNN," *2021 Innovations in Intelligent Systems and Applications Conference (ASYU), 2021*, pp. 1-6, doi: 10.1109/ASYU52992.2021.9599029.

[24] F. Uysal, F. Hardalaç, O. Peker, T. Tolunay, and N. Tokgöz, "Classification of Shoulder X-ray Images with Deep Learning Ensemble Models," Applied Sciences, vol. 11, no. 6, p. 2723, Mar. 2021, doi: 10.3390/app11062723.