ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

Büşra BÜLBÜL

CONVOLUTIONAL NEURAL NETWORK DESIGN WITH NEW MAX POOLING CIRCUITS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ADANA-2022

ABSTRACT

MSc THESIS

CONVOLUTIONAL NEURAL NETWORK DESIGN WITH NEW MAX POOLING CIRCUITS

Büşra BÜLBÜL

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Supervisor : Prof. Dr. Mustafa GÖK
Year: 2022, Pages: 35
Jury : Prof. Dr. Mustafa GÖK
: Assoc. Prof. Dr. Murat AKSOY
: Asst. Prof. Dr. Fatih KILIÇ

In this thesis, max pooling unit designs, which is an important process block of Convolutional Neural Networks (CNN), are presented. The max pooling layer is in the critical delay path of the CNN design and is important to influence the main conversion rate of a pipeline integrated circuit. The total frame processing times of the proposed designs are much shorter than the Standard Design. The proposed designs can be integrated into different pipeline structures. All designs are modeled with VHDL and synthesized on a current FPGA platform. The synthesis results show that the fastest of the proposed designs processes a $128x128$ frame around 8.1 times faster than the Standard Design.

The first max pooling circuit design presented in this thesis is used in the design of a fully functional pipelined CNN. The presented design has six layers. The main focus of the implementation is performance efficiency, to double the speed it divides the input images by half and simultaneously processes them in two data paths. The CNN design has reduced latency compared to a standard implementation. Also, the design fits on a medium size FPGA platform.

Keywords: Max Pooling, Convolutional Neural Network, Digital Design

ÖZ

YÜKSEK LİSANS TEZİ

YENİ MAKSİMUM ORTAKLAMA DEVRELERİYLE EVRİŞİMLİ SİNİR AĞI TASARIMI

Büşra BÜLBÜL

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK- ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Prof. Dr. Mustafa GÖK
Yıl: 2022, Sayfa: 35
Jüri : Prof. Dr. Mustafa GÖK
: Doç. Dr. Murat AKSOY
: Dr. Öğr. Üyesi Fatih KILIÇ

Bu tezde gelişkin Evrişimli Sinir Ağı (ESA) mimarilerinin önemli bir işlem bloğu olan maksimum ortaklama ünite tasarımları sunulmuştur. Maksimum ortaklama katmanı ESA tasarımlarının kritik gecikme yolunda olup, boru hatlı bir tümleşik devrenin ana çevrim hızını etki edebilecek önemdedir. Önerilen tasarımların toplam çerçeve işleme süreleri Standart Tasarıma göre çok daha kısadır. Önerilen tasarımlar farklı boru hatlı yapılara entegre edilebilecektir. Tasarımlar VHDL ile modellenmiş ve güncel bir FPGA platformu üzerinde sentezlenmiştir. Sentez sonuçları, önerilen tasarımların en hızlısının Standart Tasarımla karşılaştırıldığında $128x128$'lik bir çerçeveyi yaklaşık 8.1 kat daha hızlı işlediğini göstermiştir.

Bu tezde ayrıca sunulan ilk maksimum ortaklama devresi tam fonksiyonel boru hatlı bir ESA donanım tasarımına entegre edilmiştir. Mevcut tasarım altı katmandan meydana gelmektedir. Uygulamanın ana amacı performans verimliliğine sahip basit bir tasarım oluşturmaktır. Bu amaç doğrultusunda ESA basit tutulmuş ve hızı artırmak için işlenen görüntüler iki eşit parçaya ayrılarak aynı anda iki veri yolu ile işlenir. ESA tasarımı standart tasarıma göre gecikmeyi azaltmıştır. Ayrıca, tasarım orta boyutlu bir FPGA platformu üzerinde gerçeklenmiştir.

Anahtar Kelimeler: Maksimum Ortaklama, Evrişimli Sinir Ağı, Sayısal Tasarım

EXTENDED ABSTRACT

In recent years, deep learning become de facto tool for solving complicated image processing problems. Many deep learning applications require real time response. In order to provide that type of performance the models are mapped on hardware.

CNN models are widely used in image processing tasks such as object recognition, detection and classification problems. FPGA based implementations for CNNs are widely studied in speech recognition (Abdel-Hamid, et al., 2012), real-time classification (Hwang, et al., 2017), object detection (Ren, et al., 2017) ,target detection (Li, et al., 2018), computer-assisted diagnostics (Rajaraman, et al., 2018), face recognition (Qiao, et al., 2018). Application-oriented circuit designs to improve the processing performance of these networks have attracted much attention in recent years.

Major processing layers in a CNN model are convolutional layers, pooling layers and fully connected layers. All these layers are on the critical path of the hardware. This study focused on improving max pooling hardware in the hope of that improving the processing speed as well as fully connected layer implementation that fits our proposed designs.

The convolution layer is designed by $3x3$ sliding window with no stride. This layer is similar to previous studies in (Hamdan, et al., 2017) and (Qiao, et al., 2018). The convolution layer consists of convolution operation and activation operation.

The convolution operation consists of 3 FIFOs, 9 shift registers, and 9 multipliers to perform $3x3$ sliding window. After FIFOs are full, the outputs of the shift registers are multiplied by weights and multiplication results are accumulated by the adder tree. In this way, the outputs of the convolution operation are computed.

ReLU that is a nonlinear activation function is applied to the outputs of the convolution operation to determine the neuron output values in this thesis. If the

output of the convolution operation is less than zero, the neuron is deactivated. Thus, all the neurons are not activated at the same time. That is the main advantage of using ReLU over other activation functions.

Max pooling operation is an important part of CNN hardware designs and there is a growing interest on the design of area and energy efficient max pooling circuits. These circuits are used in (Hamdan, et al., 2017), (Hwang, et al., 2017), (Qiao, et al., 2018), and (Zhao, et al., 2020). Different architectures of $2x2$ max pooling operation with 2 stride is performed in (Hamdan, et al., 2017), (Hwang, et al., 2017),and (Qiao, et al., 2018) while $3x3$ max pooling operation with 2 stride is presented in (Zhao, et al., 2020). At the end of the max pooling operation, the input frame size is decreased with max pooling filter size and stride. In this way, the max pooling operation finds more general features from higher precision frames.

This thesis presented pipelined and sequential max pooling designs that improves total processing time as well as main clock frequency. These designs can process 128 by 128 frame. The size of the FIFOs is directly affected by the frame size. The data used in max pooling circuit is fixed point data sent by ReLU layer. In order to find the maximum value in a given window must be compared.

The conventional sequential $2x2$ max pooling circuit design with stride 2 is presented 2 FIFOs, 4 registers, 3 comparator circuits. The data coming from ReLU layer are loaded into the FIFOs. After FIFOs are full, the conventional sequential max pooling circuit compares 4 data points in the windows at each step and it finds the maximum value at each 2 clock cycles. The conventional max pooling design processes mxm frame in $(3m^2 + m)\ /2$ clock cycles.

Proposed max pooling design 1 proposed in the thesis assumes that a single value comes from convolution layer at each iteration (Bülbül, et al., 2020). The design performs $2x2$ max pooling operation with stride 2. Proposed max pooling design 1 contains a 16-bit register, a FIFO, a multiplexer, and a comparator. The size

of the FIFO is m bytes for an $n\ by\ m$ frame and max pooling operation takes $n.m$ clock cycles. The design can support quite high frequency execution.

Proposed max pooling design 2 is proposed for networks that can process a frame in parallel. In those cases, an efficient way of max pooling is also required. Proposed max pooling design 2 can process two rows of a frame is processed (Bülbül, et al., 2020). In this circuit, 2 registers, 2 comparators are used. This design can process $m\ by\ m$ frame in $m^2/2$ clock cycles.

All the max pooling circuits presented in this study is written in VHDL and mapped on a low cost FPGA platform. The comparators used in the max pooling circuits presented in this thesis are designed by using a comparator and subtractor in order to observe the FPGA performance. The proposed max pooling design 2 is the best in terms of resource usage while the conventional sequential max pooling circuit uses more FPGA resources. Maximum frequency results obtain from 85C and 0C models of FPGA platform. The maximum frequency of the conventional sequential max pooling circuit is better than the proposed max pooling design 1 but slower than the proposed max pooling design 2. Besides all these, maximum frame processing time is significant. The proposed max pooling design 1 processes $m\ by\ m$ frame in m^2 clock cycles, the proposed max pooling design 2 in $m^2/2$ clock cycles, the conventional sequential max pooling circuit in $(3m^2 + m)/2$ clock cycles. Even though the proposed max pooling design 1's clock frequency is less than the conventional max pooling circuit's, its frame processing speed is faster than the conventional max pooling circuit. For example, the conventional max pooling circuit can process $128\ by\ 128$ frame in $154\ \mu s$ while the proposed max pooling design 1 can do the same job in $141\ \mu s$. On the other hand, this frame can be processed by the proposed max pooling design 2 in $19\ \mu s$.

Fully connected layer implementation presented in this thesis are designed for sequential execution and adapted to max pooling hardware. The outputs of max pooling layers are processed in parallel and generates a layer output in sync with the

pooling operations. Every neuron on fully connected layer is connected to every other neuron of the next layer. This thesis consists of 2 fully connected layers. The first fully connected layer includes 10 units. There is a single unit in the second fully connected layer. The second fully connected layer processes the outputs of the fully connected layer nodes and this process contains 10 multipliers and a multi-operand adder in this thesis.

All the designs presented in this study are modeled using VHDL and mapped on a state of the art FPGA. Max pooling design 1 is integrated into a fully functional CNN. Both of standard CNN design and proposed CNN design processes an $m\ by\ m$ grayscale image. While standard architecture consists of only one data path, proposed architecture is implemented two identical data paths to decrease the latency and increase the efficiency.

Syntheses are performed and compared with standard implementations. Compared to the standard implementation the proposed design generates the result faster. While the final output is obtained after $m^2 + 3m$ by the standard design, the proposed design produces $m^2 - 9m$. For example, a 30 by 30 frame is processed in 630 cycles by the proposed design while the standard implementation generates the same result in 990 cycles. The proposed design is approximately 36% faster for this size.

The clock frequency of the standard design is 51.19 MHz and the clock frequency of the proposed design is 50.39 MHz at slow 1100mV 85C model of FPGA model. The latency per image with the proposed design is calculates 0,0125 milliseconds for a 30 by 30 frame.

Compared to the standard implementation, the proposed implementation uses roughly 40% more DSP elements which is main parameter for selecting the FPGA. The differences among the other resource categories do not affect the platform selection decision.

The proposed designs currently can process 30 by 30 grayscale images but the same design can be used to process larger images on different FPGA platforms.

ACKNOWLEDGMENTS

I would like to express my deep and sincere gratitude to, my supervisor, Prof. Dr. Mustafa GÖK for his guidance, supervision, encouragement, and for leading my interests to the research and scholarship. It has been a great honor to have Prof. Dr. Mustafa GÖK as a supervisor.

I also want to thank my colleague Halit ERİŞ for his encouragement, and support during my thesis.

My sincere thanks go to my family for their everlasting encouragement, supporting me spiritually throughout my life, and for their patience.

TABLE OF CONTENTS **PAGE**

ABSTRACT I
ÖZ II
EXTENDED ABSTRACT III
ACKNOWLEDGMENTS IX
TABLE OF CONTENTS X
LIST OF TABLES XII
LIST OF FIGURES XIV
ABBREVIATIONS XVI
1. INTRODUCTION 1
2. PREVIOUS WORK 5
2.1. Conventional Max Pooling Operation 5
2.1.1. What is The Effect of Max Pooling Operation? 6
2.2. Conventional Average Pooling Operation 6
2.3. The Meaning of Stride Operation 7
2.4. Design of Conventional Sequential Max Pooling Circuits 8
2.4.1. Design of Conventional Sequential $2x2$ Max Pooling Circuit 8
2.4.2. Design of Conventional Sequential $3x3$ Max Pooling Circuit 9
2.5. Max Pooling Circuit Designs in Previous Studies 10
3. PROBLEM DEFINITION 17
3.1. Convolutional Neural Network Architecture 17
3.2. The Convolutional Layer 17
3.3. The Activation Function 19
3.4. The Pooling Layer 20
3.5. The Fully Connected Layer 20
4. PROPOSED DESIGN 21
4.1. The Convolution Layer 21
4.2. Proposed Max Pooling Design 1 22

4.3. Proposed Max Pooling Design 2 23
4.4. Proposed Fully Connected Layer 24
5. RESULTS 27
6. CONCLUSION 31
REFERENCES 33
CURRICULUM VITAE 35

LIST OF TABLES **PAGE**

Table 5.1. Full design latency in clock cycles. 27
Table 5.2. The usage of the resources. 28
Table 5.3. FPGA resource utilization..... 29
Table 5.4. FPGA maximum clock cycles. 30

LIST OF FIGURES **PAGE**

Figure 2.1. Example of max pooling operation. 6
Figure 2.2. Example of average pooling operation. 7
Figure 2.3. The block diagram of a conventional sequential 2x2 max pooling circuit. 9
Figure 2.4. The block diagram of a conventional sequential 3x3 max pooling circuit. 10
Figure 2.5. The block diagram of a 2x2 max pooling circuit adapted from..... 11
Figure 2.6. The block diagram of a 2x2 max pooling circuit adapted from..... 12
Figure 2.7. The block diagram of a 2x2 max pooling circuit adapted from..... 13
Figure 2.8. The block diagram of 2 level max pooling adapted from 14
Figure 2.9. The block diagrams of (a) first level max pooling unit and (b) second level of max pooling unit adapted from 15
Figure 3.1. The block diagram of a CNN. 17
Figure 3.2. Convolution operation on an input image of size 5x5x3 using a 3x3x3 filter..... 18
Figure 3.3. The block diagram of a convolutional filter. 19
Figure 4.1. The block diagram of proposed CNN architecture. 21
Figure 4.2. Standard convolutional layer. 22
Figure 4.3. The block diagram of max pooling design 1. 23
Figure 4.4. The block diagram of max pooling design 2. 24
Figure 4.5. The block diagram of FC1 layer unit..... 25

ABBREVIATIONS

ALM	: Adaptive Logic Module
ALUT	: Adaptive Look-Up Table
ASIC	: Application Specific Integrated Circuit
CNN	: Convolutional Neural Network
D1	: Proposed Max Pooling Design 1
D2	: Proposed Max Pooling Design 2
DSP	: Digital Signal Processing
EQ	: Equation
FC1	: Fully Connected Layer 1
FC2	: Fully Connected Layer 2
FIFO	: First Input First Output
FIG	: Figure
FPGA	: Field Programmable Gate Array
GFLOPS	: Giga Floating Point Operations per Second
GPU	: Graphics Processing Unit
HLS	: High Level Synthesis
I/O	: Input/ Output
MHz	: Megahertz
MLAB	: Memory Logic Array Block
MP	: Max Pooling
RAM	: Random Access Memory
ReLU	: Rectified Linear Unit
SD	: Standard Max Pooling Design
VHDL	: VHSIC Hardware Description Language
VHSIC	: Very High-Speed Integrated Circuits Program

1. INTRODUCTION

Deep learning-based practices are about to become an indispensable element of our modern life. Recent advances in areas such as image recognition, voice recognition, time series analysis have been made possible through the use of multi-layer deep networks (Goodfellow, et al., 2016). Convolutional Neural Networks (CNNs) are widely used in image recognition tasks. Application-oriented circuit designs to improve the processing performance of these networks have attracted much attention in recent years. These designs are used in real-time classification (Hwang, et al., 2017), target detection (Li, et al., 2018), computer-assisted diagnostics (Rajaraman, et al., 2018), object detection (Ren, et al., 2017), speech recognition (Abdel-Hamid, et al., 2012), face recognition (Qiao, et al., 2018).

Hardware implementations of CNNs can be roughly classified in three categories. In the first category are the Graphics Processor based cards, which are generally used in algorithm development and test phase due to their ease of use. Graphics cards can be used in personal computers or can be accessed as a cloud service. However, they have serious disadvantages in terms of size and power consumption compared to the custom solutions. Application Specific Integrated Circuit (ASIC) implementations can be considered in the second category. These chips offer the highest performance and low cost when mass production is on the table. However, they have high engineering and maintenance cost. ASIC implementations can be preferred when the demand for high performance dominates the cost concerns (as in military, health, etc.). The third category includes the Field Programmable Gate Array (FPGA) based implementations. They can be a good alternative to the previous categories since they have performance advantage over Graphics Processing Units (GPU) and cost advantage over ASIC implementations. FPGAs offer reduction in the cost of engineering and design updates. They can be used in the development and test phase as well as the production phase.

Nowadays, FPGA based implementations for CNNs are widely studied (Hamdan, et al., 2017), (Hwang, et al., 2017), (Qiao, et al., 2018), and (Zhao, et al., 2020).

In Hamdan 2017, a VHDL generator tool proposed to automatically produced VHDL code for CNN. The tool used to generate a fairly small CNN model, known as LeNet. And implemented on Xilinx Virtex-7 architecture. The main contribution of this work it frees the user from the complex development process of HLS tools. The executable code was made available in GitHub (Hamdan).

In Hwang 2017, an FPGA based CNN implementation that can be used for character recognition, face recognition, and hand posture recognition were presented. The hardware can process floating point numbers and mapped on Stratix 4 FPGA device. The proposed design is pipelined and can achieve up to 21.70 GFLOPS, which is suitable for real time classification applications.

In Qiau 2018, another FPGA implementation for face recognition is proposed. The study explored parallel execution options to perform kernel operations. The design consists of 4 processing layers where each layer contains a convolutional layer and a pooling layer. The design is modelled using Verilog and mapped on CycloneIV E device. The design achieved 60 MHz frequency and used 74% of total logic elements of the device.

In Zhao 2020, in particular max pooling circuit design is studied with the goal of area and energy efficiency. This work shares the goal of this thesis and for this reason in the previous work section the details of the implementation will be discuss.

The literature review shows that max pooling operation is an important part of CNN hardware designs and there is a growing interest on the design of area and energy efficient max pooling circuits. This thesis main goal is to present new methods that achieves this goal and increase the performance of CNN designs. The rest of the thesis is organized as follows. Section 2 presents previous work on max pooling circuit design, Section 3 presents problem definition, Section 4 introduces

proposed design, Section 5 shows the synthesis results and Section 6 gives the conclusions.

2. PREVIOUS WORK

Pooling layers also known as subsampling layers decrease the frame size and help to extract more important features. In general, there are two types of pooling methods used in the literature. This section presents recent previous work on the design of circuits that perform these operations.

2.1. Conventional Max Pooling Operation

Conventional max pooling operation finds the maximum of a number in a given window. Fixed point data sent by Rectified Linear Unit (ReLU) layer are received by max pooling layer. In order to find the maximum all the numbers in the window must be compared.

Figure 2.1 shows a $6x6$ frame that enters max pooling operation. In this example max pooling operation is performed using $2x2$ windows. At each step 4 data points in the windows are compared. Often the pooling operation is performed on non-overlapping windows. To achieve this for $2x2$ windows the stride amount should be determined as 2. In the example 9 non-overlaping windows and the new $3x3$ frame generated by applying max pooling are shown. All the values shown in this example are positive fixed-point numbers. Because, they are received from ReLU layer which does not generate non negative output.

Although $2x2$ windows are the most used window size, there exists applications that used $3x3$ windows size. In the case of $3x3$ windows size, the stride amount is set to 1 which causes the overlapping of windows.

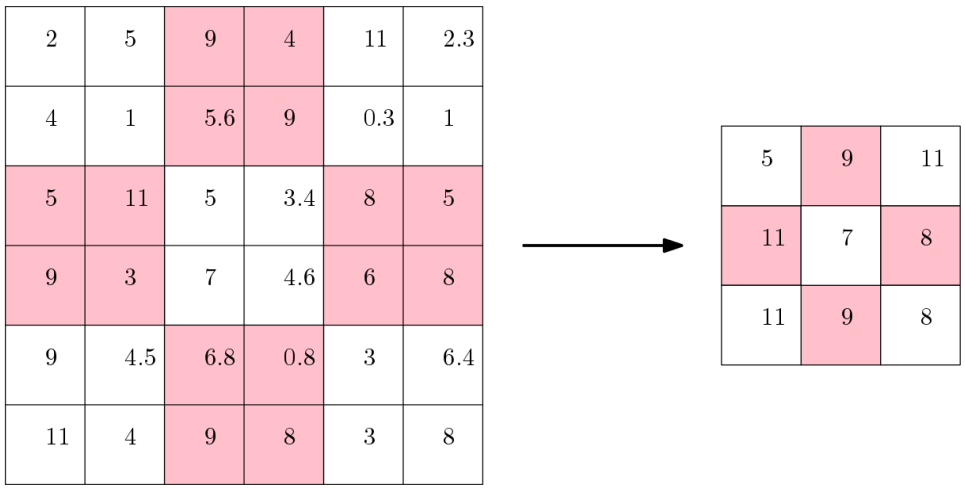

Figure 2.1. Example of max pooling operation.

2.1.1. What is The Effect of Max Pooling Operation?

The max pooling operation carries the information coming from previous convolutional layers such that at each layer information from a larger previous window is contracted and passed to next layer. This way more general features can be found from higher precision frames.

2.2. Conventional Average Pooling Operation

The average pooling operation calculates the average of data points in a given window size. Figure 2.2 shows a $6x6$ frame that enters average pooling operation. In this example average pooling operation is performed using $2x2$ windows with stride 2. At each step the average of 4 data points in the windows are computed. For example, the first window consists of $2,5,4,1$ and the average of them is 3.

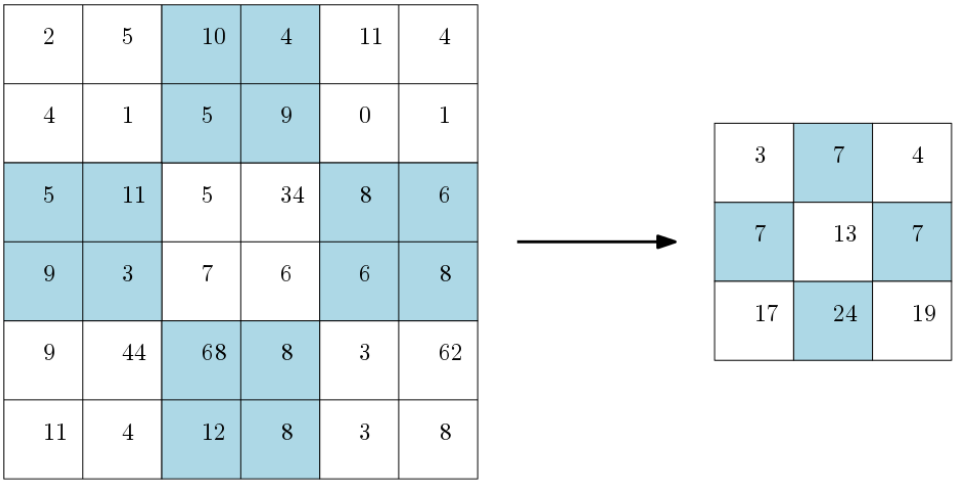

Figure 2.2. Example of average pooling operation.

Nowadays, average pooling operation is not preferred for two reasons. The first reason is it requires more computation compare to max pooling operation. The second reason is the max pooling operation does a better job on finding the special features spread on more pixels. The reason for that is it is more beneficial to detect the existence of a feature by finding its maximum value than using its average.

2.3. The Meaning of Stride Operation

Stride is the hop distance between filter windows. A new filter operation starts from the offset point determined by the stride amount. For convolution operations, stride is usually taken as 1 because, it is better to apply convolution in high precision. However, some convolution layers use stride 2 just for the purpose of down sampling but this is rarely.

2.4. Design of Conventional Sequential Max Pooling Circuits

2.4.1. Design of Conventional Sequential $2x2$ Max Pooling Circuit

In figure 2.3 a conventional sequential $2x2$ max pooling circuit design with FIFOs is presented. The circuit consists of 2 FIFOs, 4 registers, 3 comparators circuits. Assuming the size of the frame is mxm the size of the FIFOs is $m-2$ bytes and each register is 1 byte, and the comparators can process 2 values at the same time.

The data flow on this circuit is as follows:

- The data output from ReLU layer are loaded into the first and second FIFOs in $2m$ clock cycles.
- After FIFOs are filled, the processing begins. The data read at each cycle from the FIFOs are compared in the Comparators 1 and 2. The outputs of the comparators are sent to Comparator 3.

The conventional design processes mxm frame in $(3m^2+m)/2$ clock cycles. Ones the FIFOs are full this system can process a $2x2$ windows at every 2 clock cycles and generates a maximum value. Most of the previous work cited in this thesis are using this standard design.

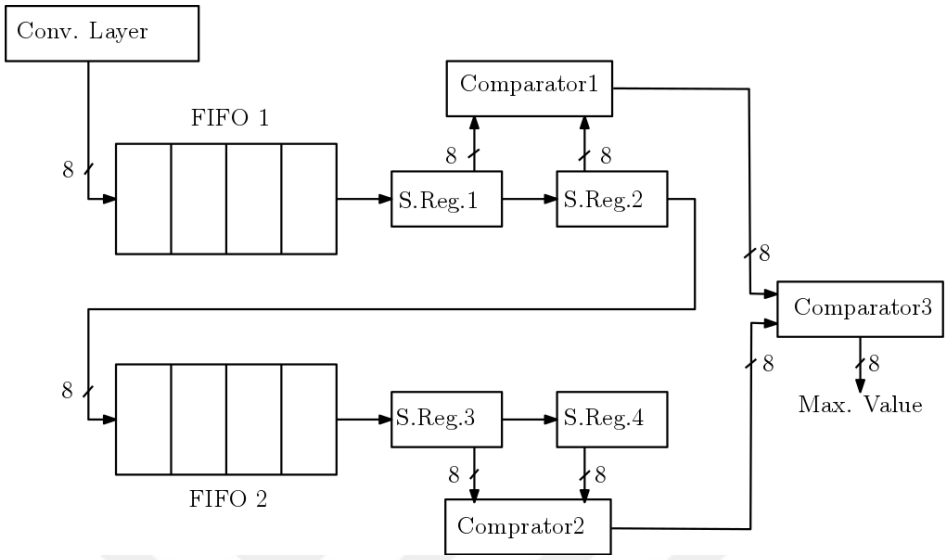

Figure 2.3. The block diagram of a conventional sequential $2x2$ max pooling circuit.

2.4.2. Design of Conventional Sequential $3x3$ Max Pooling Circuit

In figure 2.4 a conventional sequential $3x3$ max pooling circuit design with FIFOs is presented. The circuit consists of 3 FIFOs, 9 registers, 8 comparators circuits. Assuming the size of the frame is mxm the size of the FIFOs is $m-3$ bytes and each register is 1 byte, and the comparators can process 2 values at the same time.

Working principle of the circuit in figure 2.4 is similar to the $2x2$ max pooling circuit in figure 2.3. The data flow on this circuit is as follows:

- All FIFOs are loaded with data output from ReLU layer into in $3m$ clock cycles.
- After FIFOs are filled, the processing begins. The data coming from the registers at each cycle are compared in the comparators.

When the FIFOs are full this system can process a $3x3$ windows and generates a maximum value.

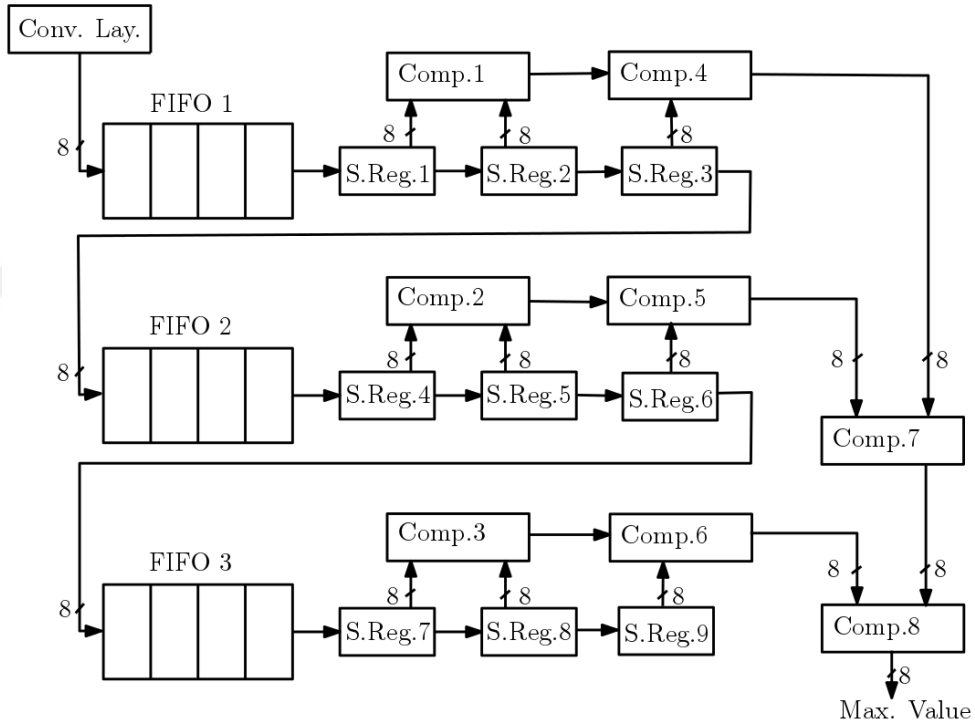

Figure 2.4. The block diagram of a conventional sequential $3x3$ max pooling circuit.

2.5. Max Pooling Circuit Designs in Previous Studies

In this section max pooling circuit designs from 4 study are explained. Figure 2.5 presents the block diagram of a $2x2$ max pooling circuit adapted from (Hamdan, et al., 2017). In this study, max pooling circuit is named as processing elements (PE). PEs contains 7 registers, 1 FIFO, and 3 comparators. PE takes up the values come from the ReLU layer and performs a $2x2$ max pooling operation with the specified stride value. The stride size is determined by Stride Enable input. When the stride is 2, a $2x2$ max pooling operation is performed.

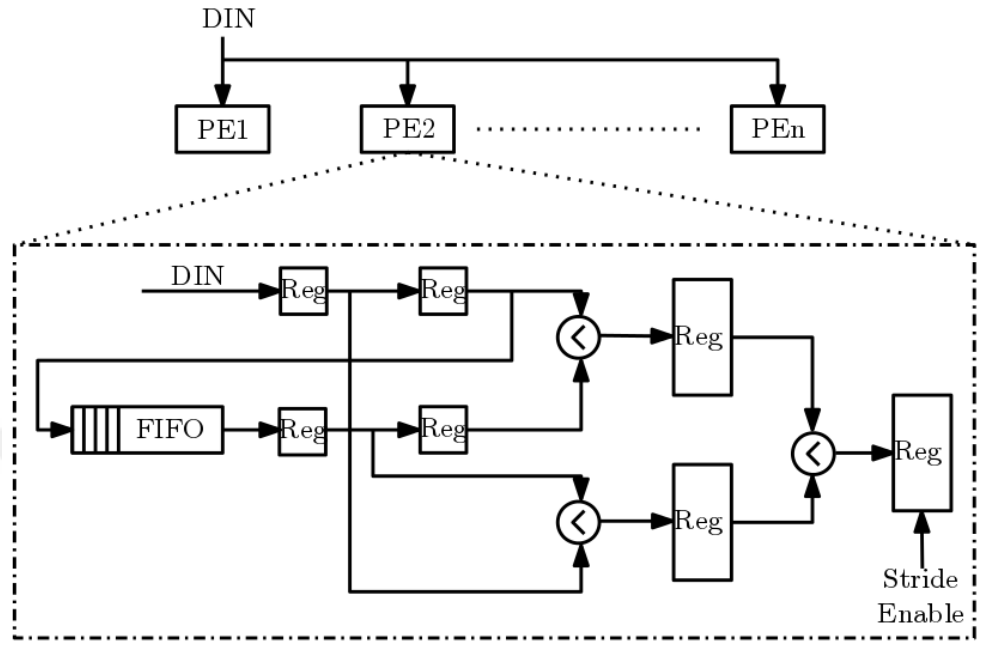

Figure 2.5. The block diagram of a $2x2$ max pooling circuit adapted from (Hamdan, et al., 2017).

Figure 2.6 presents the block diagram of a $2x2$ max pooling circuit adapted from (Hwang, et al., 2017). This circuit performs the subsampling operations with stride 2. The size of the input frame is assumed LxL. The architecture in figure 2.6 consists of Shift Register, Comparator Core and Controller parts. If the size of the input frame is assumed LxL, the Shift Register part contains $L + 2$ shift registers. The Comparator Core part involves 3 comparators to find the maximum value in the $2x2$ window. The Comparator Core part is activated merely for every other column and every other row by Controller part.

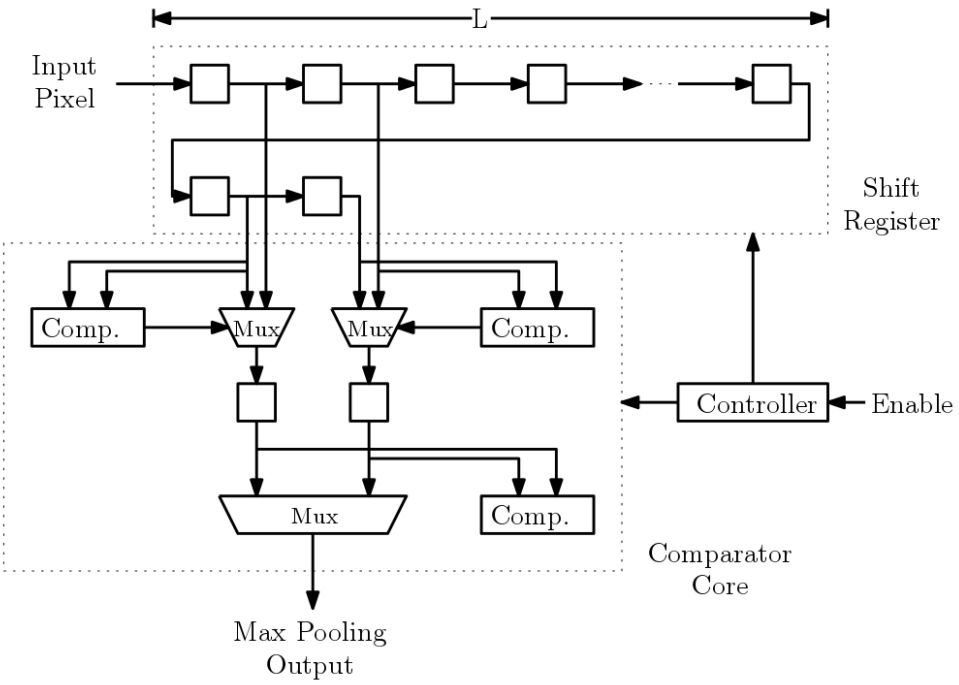

Figure 2.6. The block diagram of a $2x2$ max pooling circuit adapted from (Hwang, et al., 2017).

Figure 2.7 presents the block diagram of a $2x2$ max pooling circuit adapted from (Qiao, et al., 2018).The architecture in figure 2.7 is similar to the block diagram of a conventional sequential $2x2$ max pooling circuit in figure 2.3. The main difference between these two max pooling circuits is latter uses Shift Register instead of FIFOs. The other components of this circuit are 3 comparators and 4 registers. The circuit finds the maximum value of the four pixels using 3 comparators.

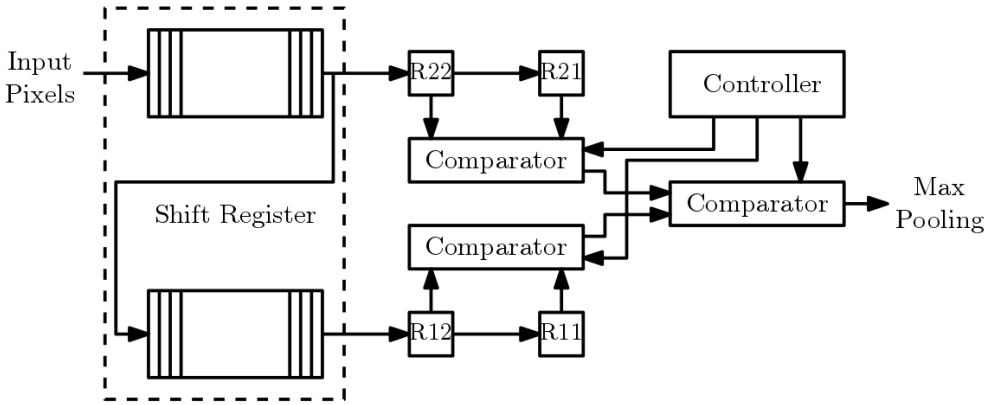

Figure 2.7. The block diagram of a $2x2$ max pooling circuit adapted from (Qiao, et al., 2018).

In (Zhao, et al., 2020) a 2 level max pooling design is used. Figure 2.8 presents the organization of this 2 level max pooling design (adapted from (Zhao, et al., 2020)). The max pooling layer is implemented to perform $3x3$ max pooling operation with stride 2. Figure 2.9 presents the details of the units in the first and second level. The advantages of 2 level max pooling circuit in figure 2.9 are area and energy efficiency.

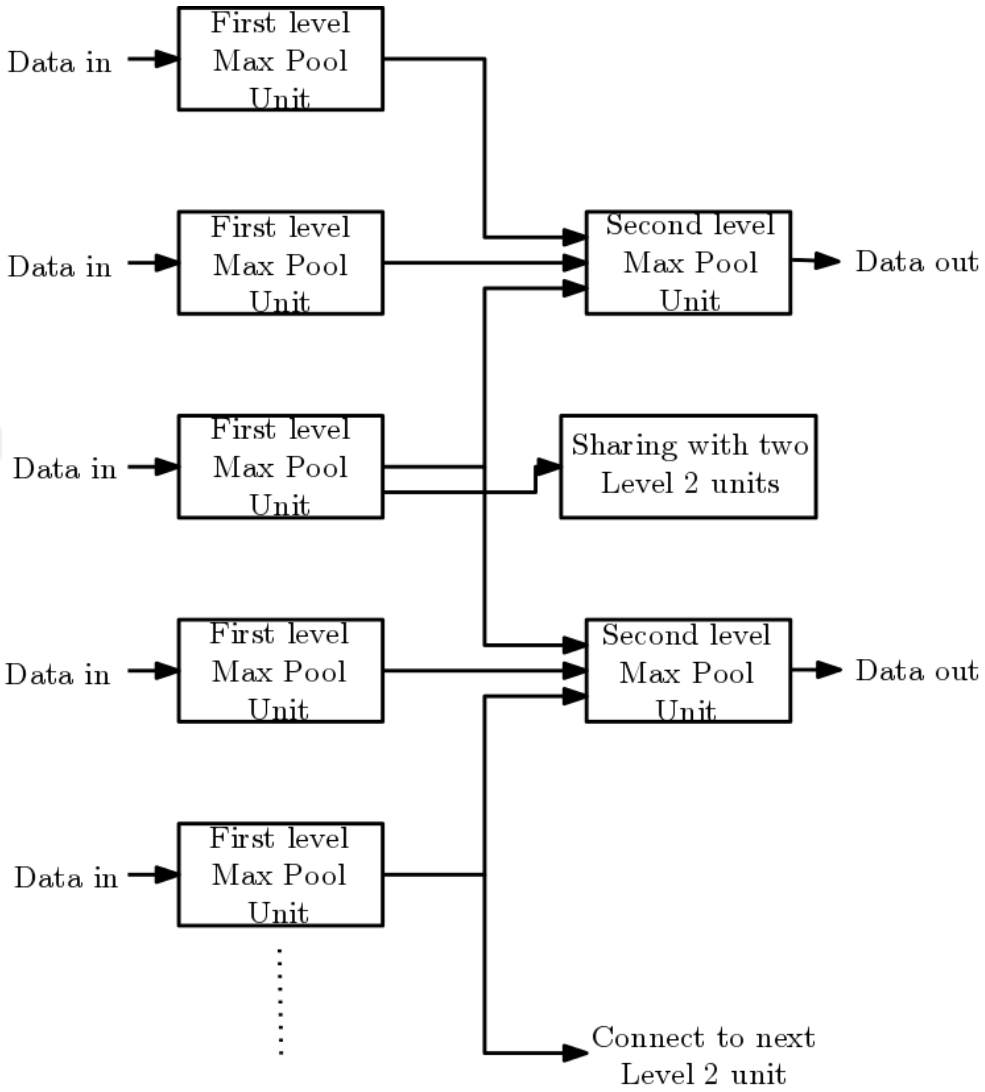

Figure 2.8. The block diagram of 2 level max pooling adapted from (Zhao, et al., 2020).

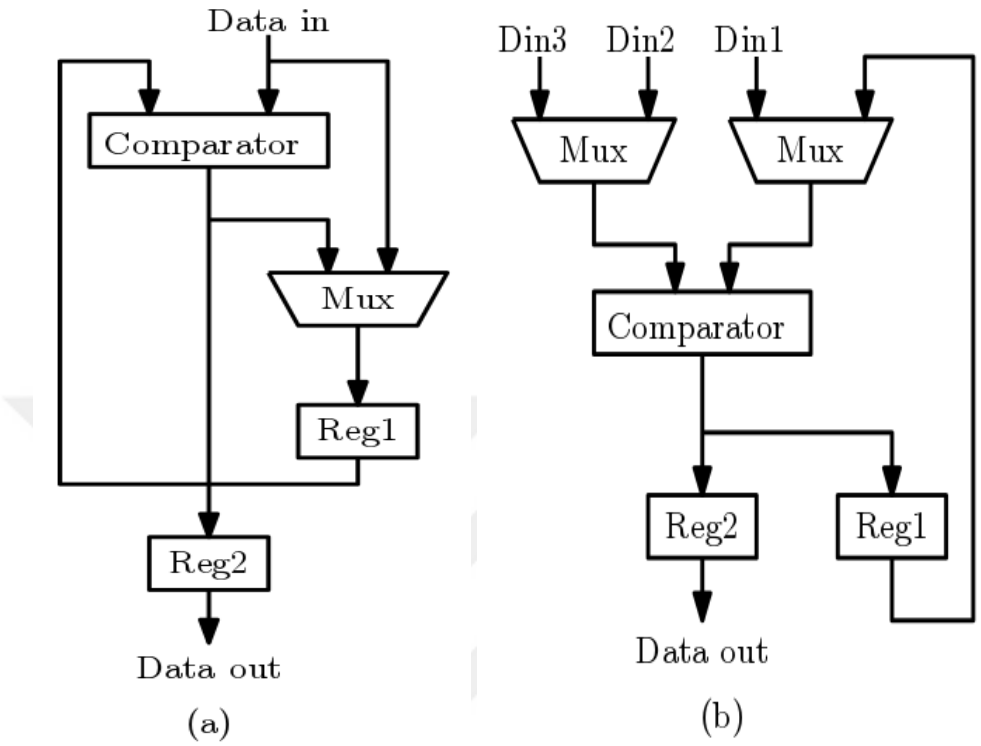

Figure 2.9. The block diagrams of (a) first level max pooling unit and (b) second level of max pooling unit adapted from (Zhao, et al., 2020).

3. PROBLEM DEFINITION

3.1. Convolutional Neural Network Architecture

A very generic CNN architecture consists of an input layer, convolutional layers, pooling layers and fully-connected layers. The output of each convolution layer is called a feature map. The layers decrease the size of the frame while the number of feature maps increases as data passes through the layers. Convolutional and pooling layers are cascaded many times in deep CNN implementations.

In figure 3.1, the block diagram of a CNN implementation that process a $30\ by\ 30$ grayscale image is shown as running example. The grayscale image data flows through the input layer to the first convolutional layer C1, then the first max pooling layer MP1, then the second convolutional layer C2, then the second max pooling layer MP2, then a fully connected layer with 10 units and another one with a single unit. This CNN is designed to identify if image contains an object or not.

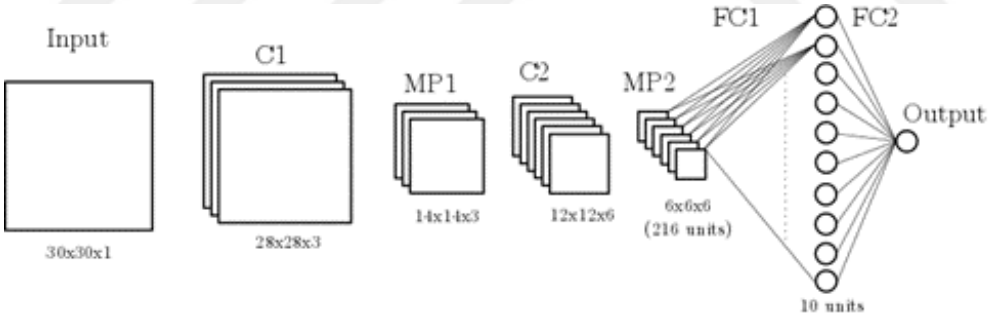

Figure 3.1. The block diagram of a CNN.

3.2. The Convolutional Layer

The convolutional layer computes the following filter equation for each filter

$$\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}^{l}.a_{ij} + b \tag{3.1}$$

, where w_{ij}^{l} are the convolution filter coefficients of channel l, a_{ij} are the image pixel values, b is the bias, and n is the filter size. An example of Eq. 3.1 is shown in figure 3.2.

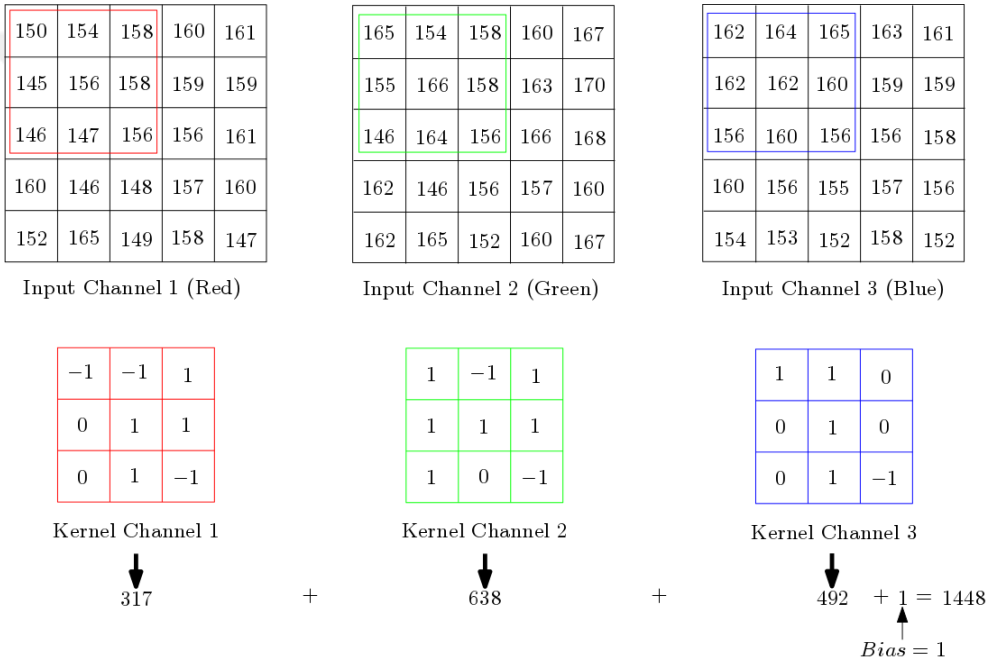

Figure 3.2. Convolution operation on an input image of size $5x5x3$ using a $3x3x3$ filter.

For the maximum speed Eq. 3.1 should be implemented by using n^2 multipliers, and n^2 adders. Figure 3.3 presents a convolutional $3x3$ filter implementation. Pixel values are shifted into the FIFOs and when they are full at each cycle a new window is processed by the circuit.

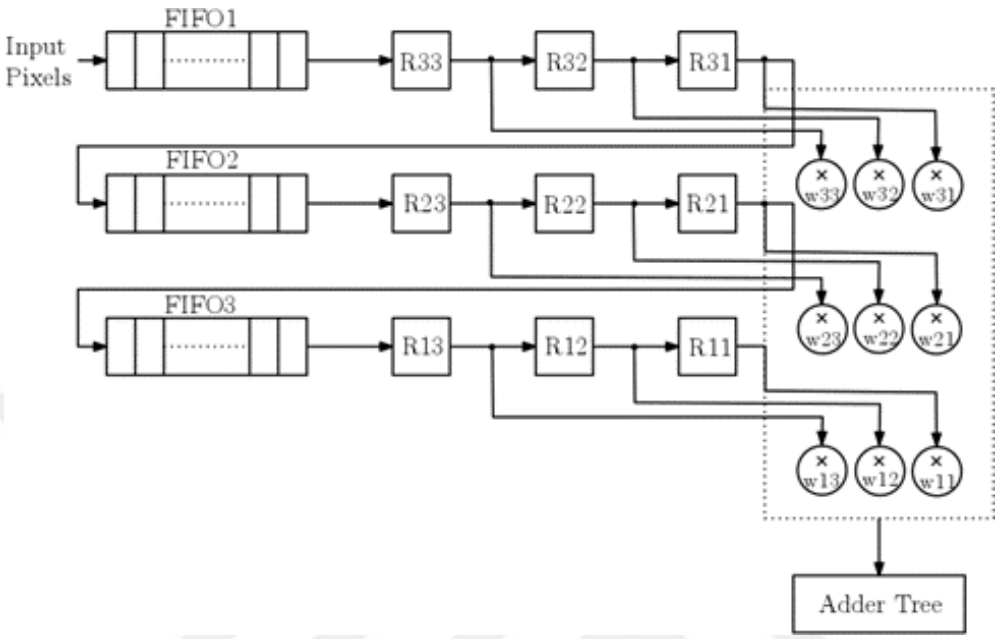

Figure 3.3. The block diagram of a convolutional filter.

3.3. The Activation Function

The activation function determines the neuron output value. The current research on CNN design favors Rectified Linear Unit (ReLU). ReLU(x) is a nonlinear activation function, which simply returns x if it is positive, otherwise zero.

$$ReLU(x) = \max(0, x) \tag{3.2}$$

ReLU(x) can be implemented using a multiplexer, which selects 0 or x based on the sign bit. This circuit is assumed as a part of the filter hardware in the rest of this work.

3.4. The Pooling Layer

A stated before, the two common pooling operations used to subsample the feature maps are the average pooling and the max pooling. The average pooling gets the average of pixels, while the max pooling outputs the maximum value in a fixed size window. In the state of the art implementations the max pooling operation is preferred.

The most commonly used max pooling window is $2\ by\ 2$. This operation reduces the frame size by two when stride of 2 is used.

3.5. The Fully Connected Layer

Fully connected layer is placed after the convolution layers before the output layer. Every neuron on this layer is connected to every other neuron of the next layer. The direct parallel implementation of this layer requires as many multipliers and adders as the neurons because of that it is best to implement this layer as a sequential circuit. A sequential implementation requires at least 6 multipliers and a multi-operand adder.

4. PROPOSED DESIGN

Proposed CNN architecture, processes an $m\ by\ m$ grayscale image. The data flow is presented in figure 4.1, which is slightly different than the flow of the standard design depicted in Fig. 4.1. In the figure, the grayscale image is assumed as a $30\ by\ 30$ image, which is divided into two equal grayscale images size of $18\ by\ 30$ to decrease the latency. Both half images are independently processed by using two convolutional layers, and two pooling layers. C1 layers on each path has three channels, the MP1 layers reduces the output of C1 by half, C2 layers have six channels, the outputs of MP2 layer are send to a fully-connected layer of ten nodes.

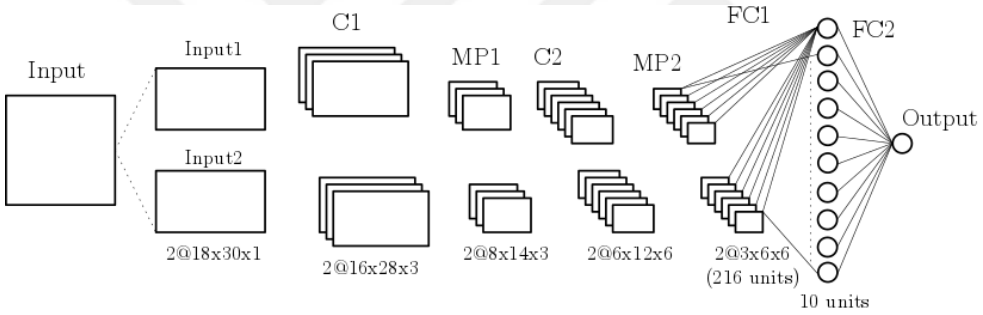

Figure 4.1. The block diagram of proposed CNN architecture.

4.1. The Convolution Layer

The convolution layer is designed as shown in figure 4.2. This is a widely used efficient implementation of Eq.3.1. In our implementation the convolution units consist of 3 FIFOs and 9 shift registers. FIFOs temporarily store values until the shift registers are ready to operate. The $3x3$ sliding window operation is realized using the shift registers. The design starts generating valid outputs after $3m + 3$ cycles. The outputs of the shift registers are multiplied by weights and multiplication results are accumulated by the adder tree. For the design in figure 4.1, 6 convolutional filters at the first level, and 36 convolutional filters at the second level are required.

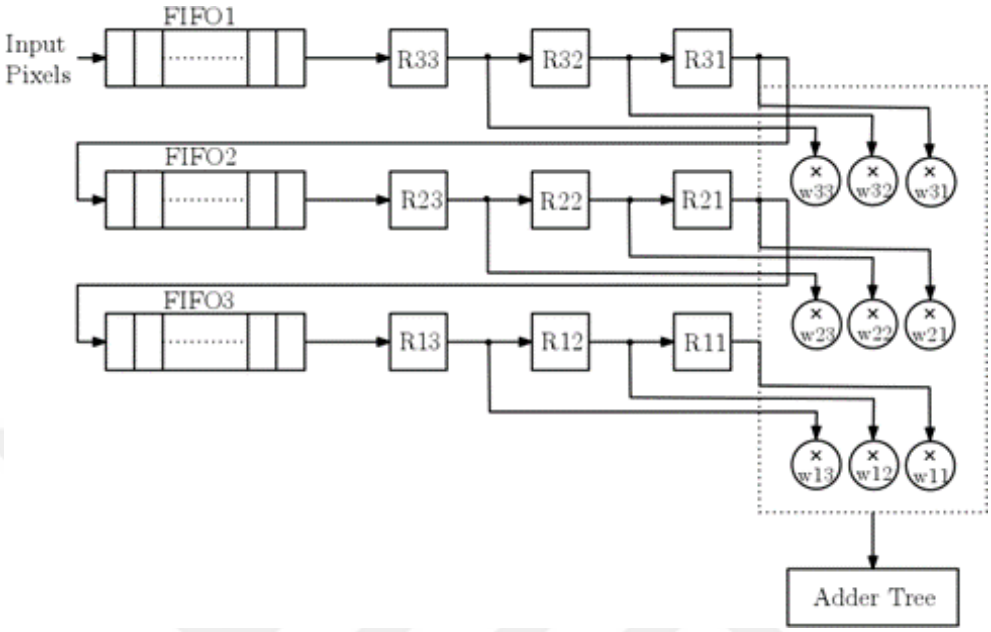

Figure 4.2. Standard convolutional layer.

4.2. Proposed Max Pooling Design 1

The max pooling layer is designed assuming a single value comes from convolution layer at each iteration (Bülbül, et al., 2020). The design performs $2x2$ max pooling operation with stride 2. Max pooling design contains a 16-bit register, a FIFO, a multiplexer, and a comparator. The size of the FIFO is m bytes for an $n\ by\ m$ frame. The block diagram of the max pooling unit design is shown in figure 4.3. The data flow is described as follows:

- The output of the convolution layer is sent to the comparator. Input '00' is chosen while processing odd numbered columns, the output of the comparator is stored into the register, and the write control of FIFO is disabled.
- The even numbered columns are compared with the content of the register by selecting '01' input of the multiplexer. Then, the largest of

them is written on the FIFO. First and second steps are successively executed until FIFO is full, which takes m clock cycles.

- When the second row starts, input '10' is selected, the data is compared with value in FIFO and in the next iteration the new value is compared with value in the register. Third step takes another m clock cycles.

In brief, the design processes two rows at $2m$ cycles and generates a new frame row input for the next convolution layer. Max pooling operation takes $n.m$ clock cycles for an $n\ by\ m$ frame. For the design in Fig. 4, 6 max pooling units at the first level and 12 at the second level are required.

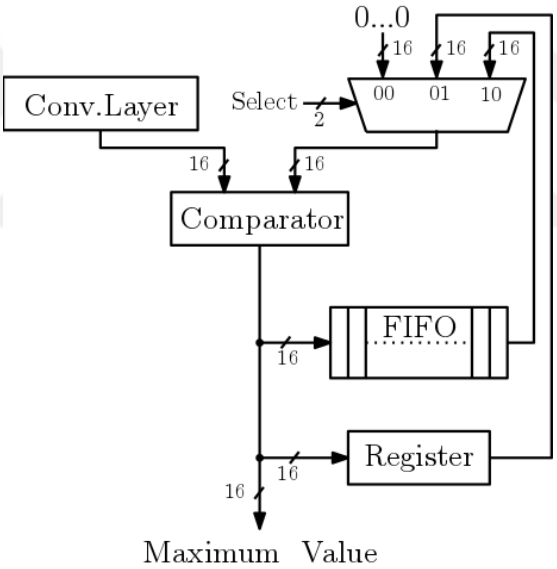

Figure 4.3. The block diagram of max pooling design 1.

4.3. Proposed Max Pooling Design 2

Design 2 can be preferred whenever two rows of a frame is processed (Bülbül, et al., 2020). In figure 4.4 the block diagram of the second design is presented. In this circuit, 2 registers, 2 comparators are used.

The data flow in this unit is explained as follows.

- Two values are compared in the comparator and the larger one is stored into the register. The register can be written only on odd cycles.
- In the second cycle, two new values are compared and then the larger of these values is compared with the value stored in the register. This design can process $m\ by\ m$ frame in $m^2/2$ clock cycles.

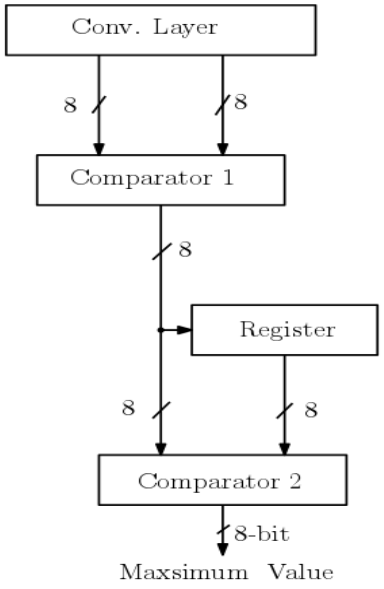

Figure 4.4. The block diagram of max pooling design 2.

4.4. Proposed Fully Connected Layer

The proposed fully connected layer implementation in the FC1 layer consists of sequential fully connected layer designs for each node. In the proposed design there are 10 nodes at this layer. There is a single node in FC2 and its design is similar to a combinational convolutional layer design. The block diagram for the FC1 node is shown in figure 4.5. The data flow of the design is described as follows:

- FC1 gets data from each MP2 channel (six for this design). These values are multiplied by 8-bit weight values and the products are summed by the multi-operand adder.
- The input ‘10’ of the multiplexer is selected and the bias value is added with the output of the multi-operand adder. In the later iterations, the ‘01’ input is selected to add the accumulated sum with the output of the multi-operand adder.
- The input ‘00’ is selected when no new operand arrives.

The outputs of FC1 nodes are processed by the design in FC2, which contains ten multipliers and a multi-operand adder for the example system. For the design in figure 4.1, 10 multipliers and one multi-operand adders are required.

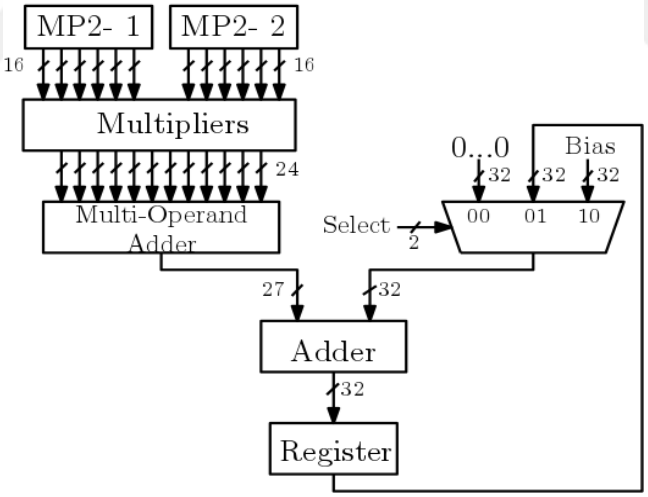

Figure 4.5. The block diagram of FC1 layer unit.

5. RESULTS

Table 5.1 presents the latency of each layer of the standard and proposed implementations based on m, which is the number of input image columns. The proposed implementation uses max pooling design 1 presented in section 4.2. In table 5.1, the first column shows the label of the layer, the second column shows the latency of the first output from each layer and the third column shows the last output from the layer for the standard implementation, the fourth and fifth columns show the same content for the proposed implementation. The imbalance of latencies among layers are due to the reduction of frames by max pooling layers. The frame latency of first output in is important for quick response. The final output is obtained after $m^2 - 9m$ cycles by the proposed implementation. Compared to the standard implementation (shown in figure 3.1), the proposed design generates the result faster. For example, a $30\ by\ 30$ frame is processed in 630 cycles by the proposed design while the standard implementation generates the same result in 990 cycles. The proposed design is approximately 36% faster for this size.

Table 5.1. Full design latency in clock cycles.

Layer	Standard Implementation		Proposed Implementation	
	First Output	Last Output	First Output	Last Output
C1	$3m + 3$	$m^2 + m$	$3m + 3$	$m^2 - 11m$
MP1	$4m + 5$	$m^2 + m + 1$	$4m + 5$	$m^2 - 11m + 1$
C2	$9m + 3$	$m^2 + 3m - 3$	$9m + 3$	$m^2 - 9m - 3$
MP2	$12m + 8$	$m^2 + 3m - 2$	$12m + 8$	$m^2 - 9m - 2$
FC1	$12m + 9$	$m^2 + 3m - 1$	$12m + 9$	$m^2 - 9m - 1$
FC2	$m^2 + 3m$	$m^2 + 3m$	$m^2 - 9m$	$m^2 - 9m$

The proposed implementation and standard implementation are modeled using VHDL and synthesized by using Quartus II synthesis tool on Intel Cyclone V

5CGTFD9E5F35C7 FPGA model. The clock frequency 51.19 MHz and 50.39 MHz at slow 1100mV 85C setting for the standard and proposed implementations, respectively. Table 5.2 presents the resource usages obtained by using the tool. Compared to the standard implementation, the proposed implementation uses roughly 40% more DSP elements which is main parameter for selecting the FPGA. The differences among the other resource categories do not affect the platform selection decision.

Table 5.2. The usage of the resources.

Resource	**Standard Implementation**		**Proposed Implementation**	
	Usage	%	Usage	%
Logic utilization (in ALMs)	4092	4	7781	7
Combinational ALUT usage for logic	5806	-	10498	-
Dedicated logic registers	7357	-	14056	-
I/O pins	53	9	61	10
Total MLAB memory bits	0	-	0	-
Total block memory bits	12696	< 1	25392	< 1
Total RAM Blocks	72	6	144	12
Total DSP Blocks	155	45	290	85
Maximum fan-out	8437	-	16216	-
Total fan-out	65626	-	125020	-
Average fan-out	4.52	-	4.61	-

We also design systems that can process 128 by 128 frame. The frame size is noted since it has a direct effect on the size of the FIFOs. The design can be modified to process smaller or larger frames just by resizing the FIFO.

The comparators are designed by using a subtractor and comparator in order to observe the FPGA performance of the designs since they are on the critical path

of the system. The designs that use comparators outperformed the ones with the subtractor.

In Table 5.3, FPGA resource usage data are presented. In this table, the first column shows the type of resources used in the design. They are logic utilization, combinational ALUT usage for logic, dedicated logic registers, I/O pins, maximum fan-out, total fan-out, average fan-out. In columns 2,3,5 and 6 the resources used for the proposed designs are shown. In columns 4 and 7 the resources used in the standard design are shown.

The standard design (SD) uses more FPGA resources since they require larger FIFOs. The proposed design 2 (D2) is the best in terms of resource usage. The reason for this is it does not use FIFO and has more regular inter connections.

Table 5.3. FPGA resource utilization.

Resource	**Magnitude Comparators**			**Subtractors**		
	D1	**D2**	**SD**	**D1**	**D2**	**SD**
Logic Utilization (in ALMs)	74	20	129	70	19	125
Combinational ALUT usage for logic	97	28	172	100	37	178
Dedicated logic registers	59	10	138	59	10	138
I/O Pins	18	26	18	18	26	18
Maximum fan-out	67	10	154	67	17	154
Total fan-out	734	193	1426	705	192	1415
Average fan-out	3.67	2.14	3.94	3.47	1.94	3.85

In Table 5.4, maximum frequency results obtain from 85C and 0C models. The maximum frequency of SD is better than the proposed design 1 (D1) but slower than D2. On the other hand, maximum frame processing time is important. D1 processes $m\ by\ m$ frame in m^2 clock cycles, D2 in $m^2/2$ clock cycles, standart design in $(3m^2 + m)/2$ clock cycles. Even though D1's clock frequency is less than standard designs', its frame processing speed is faster than the standard design. For

example, D1 can process 128 by 128 frame in 141 μs while standart design can do the same job in 154 μs. On the other hand, this frame can be processed by D2 in 19 μs.

Table 5.4. FPGA maximum clock cycles.

Design	Magnitude Comparators		Subtractors	
	Slow 1100mV 85C (MHz)	Slow 1100mV 0C (MHz)	Slow 1100mV 85C (MHz)	Slow 1100mV 0C (MHz)
D1	115.89	114.09	114.78	111.20
D2	426.08	442.87	408.50	419.46
SD	160.03	161.92	150.22	150.15

6. CONCLUSION

This paper presented max pooling designs that can be used in any deep learning hardware. Max pooling hardware designs often neglected however, it is on the critical path of deep learning processing. The improved circuits presented in this thesis may decrease the performance of various applications mapped on FPGAs. These designs are modelled in VHDL and mapped on a state of the art FPGAs. Through synthesis resource usage and delay of the designs are obtained. The main focus in the design process was increasing the performance while keeping the area of the design small enough to fit on a low cost FPGA platform.

Max pooling designs are integrated into a fully functional Convolutional Neural Network. The proposed designs currently can process $30\ by\ 30$ grayscale images but the same design can be used to process larger images on different FPGA platforms. The model of this design is also written in VHDL and synthesis are performed.

Another contribution of this thesis is the design of a custom fully connected layer with the goal of increasing processing performance.

The presented designs and methods may be used as a guide for more specialized hardware units. The future work would include more advanced versions of max pooling operations.

REFERENCES

Abdel-Hamid, Ossama, et al. 2012. Applying Convolutional Neural Networks concepts to hybrid NN-HMM model for speech recognition. Kyoto, Japan : IEEE, 2012. pp. 4277-4280.

Bülbül, Büşra and Gök, Mustafa. 2020. Evrişimli Sinir Ağları için Maksimum Ortaklama Devre Tasarımları. s.l. : Çukurova Üniversitesi Mühendislik-Mimarlık Dergisi, 2020. pp. 477-484. Vol. 35.

Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron. 2016. Deep Learning. s.l. : MIT Press, 2016. pp. 326-366.

Hamdan, Muhammad K. and Rover, Diane T. 2017. VHDL Generator for A High Performance Convolutional Neural Network FPGA- Based Accelerator. Cancun, Mexico : IEEE, 2017. pp. 1-6.

Hamdan, Muhammed K. GİtHub. [Online] [Cited: April 15, 2022.] https://github.com/mhamdan91/cnn_vhdl_generator.

Hwang, Wen Jyi, Jhang, Yun Jie and Tai, Tsung Ming. 2017. An Efficient FPGA-Based Architecture for Convolutional Neural Networks. Barcelona, Spain : IEEE, 2017. pp. 582-588.

Li, Yinghua, et al. 2018. Vehicle-Type Detection Based on Compressed Sensing and Deep Learning in Vehicular Networks. s.l. : Sensors, 2018. Vol. 18.

Qiao, Shijie and Ma, Jie. 2018. FPGA Implementation of Face Recognition System Based on Convolutional Neural Network. 2018. pp. 1-6.

Rajaraman, Sivaramakrishnan, et al. 2018. Visualization and Interpretation of Convolutional Neural Network Predictions in Detecting Pneumonia in Pediatric Chest Radiographs. s.l. : Applied Sciences, 2018. Vol. 8.

Ren, Shaoqing, et al. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposes Networks. s.l. : IEEE, 2017. pp. 1137-1149. Vol. 39.

Zhao, Bin, Chong, Yi Sheng and Do, Anh Tuan. 2020. Area and Energy Effficient 2D Max-Pooling For Convolutional Neural Network Hardware Accelerator. Singapore : IEEE, 2020. pp. 423-427.

CURRICULUM VITAE

Büşra BÜLBÜL received her Bachelor of Science (B.S) degree in Electrical-Electronics Engineering Department from Dokuz Eylül University in 2017. She started Master of Science (MSc) education in Electrical-Electronics Engineering Department of Çukurova University. She has been working as a Research Assistant in Electical-Electronics Engineering Department of Çukurova University since 2020. Her research areas are Computer Vision, FPGA Design, and Embedded Systems.