Efficient hardware implementation of VGG neural network

By:

Aya Emara

zewailcity for science and technology

s-ayaemara@zewailcity.edu.eg

Marihan Amein

zewailcity for science and technology

marihan.amein@gmail.com

Osama Yousef

zewailcity for science and technology

s-osamayousef@zewailcity.edu.eg

Yomnah Hassan

zewailcity for science and technology

s-yomna.hassan2015@zewailcity.edu.eg

Under the Supervision of:

Prof. Hassan Mostafa

hmostafa@zewailcity.edu.eg

Prof. Amr Helmy

ahelmy@zewailcity.edu.eg

Graduation Project Report Submitted to

Nano-technology Engineering Faculty at UST in ZewailCity

June 2019

Abstract

Convolutional neural networks (CNN) are the closest image recognition systems to humans' eyes with rising classification accuracy above 90%. This is motivating the development of their overall performance to fit in real time applications. Graphics processing units (GPUs) are very commonly used in the acceleration of CNNs due to their high speed and accuracy, but their high-power consumption is a very serious limitation. The most promising replacement is the Field Programmable Gate Arrays (FPGA) but with speed and accuracy limitations. This thesis presents a compromising FPGA based implementation to speed up the CNNs and keep a low power consumption.

Acknowledgement

we want to express our sincere gratitude to all the ones who supported and guided us all over the way.

First, we want to express our deepest gratitude to Dr. Ahmed Zewail -may his soul rest in peace- for giving us this opportunity to get better education in such helpful, productive, supportive and inspiring atmosphere. If it was not for his efforts, we would have never reached this success.

We would like to show our greatest appreciation to our supervisors Dr. Amr Helmy and Dr. Hassan Mustafa for their support and guidance to us since the very beginning of the Project and for their continuous advices in all the problems we faced.

A great thanks to all our supportive professors and teaching assistants for all their efforts that brought us to this stage especially Eng. Nehad Mansour, Eng. Kareem Abu El-Makarem -may his soul rest in peace.

We owe special thanks to our families and friends who were always pushing us forward and supporting us all the way.

Table of contents

List of Tables

List of Figures

List of Acronyms

CNN	Convolutional neural networks
DCNs	Deep convolution networks
DNN	Deep neural networks
Conv	Convolutional layer
FC	Fully connected layer
LRN	Local Response normalization
ReLu	Rectified Linear Unit layers
RGB	Red-Green-Blue the color model based on additive color
ANN	Artificial neural network
ASIC	Application-specific integrated circuit
FPGA	Field programmable gate array
HDL	Hardware descriptive language
VHSIC	Very High Speed Integrated Circuit hardware
RAM	Random access memory
BRAM	Block Random access memory
CLB	Configurable Logic Blocks
CPUs	General purpose processors
GPU	Graphics processing units

DRAM	Dynamic Random-access memory
DSP	Digital signal processing unit
FF	Flip-flop
IOBs	Input/output Blocks
MUX	Multiplexer
LUT	Look up table
MAC	Multiply and accumulate

Chapter 1.
Introduction

This thesis introduces optimized hardware implementation of the VGG convolution neural network on FPGA platform. In addition, the optimizations and design techniques used can further get modified to build a platform for all CNNs architectures.

1.1. Future of neural networks

Human brain never failed to impress all sciences with its abilities and powers. Although its complexity and the relatively slow switching activity of its neurons, it can process information incredibly faster than any electrical machine. This is due to the highly parallel processing techniques performed by the neural networks which is based on dividing the information into many pieces that gets processed individually.

Artificial Neural Networks (ANN) are brain inspired systems designed to solve highly-complex problems. They use same processing techniques as the human brains in which weights are given to the pieces of information based on how much they affect the final results. This is achieved by dividing the network into interconnected layers of processing, each is built of large number of simple processing block. This makes ANNs excel in feature extraction and classifications of complex nonlinear functions. ANN weights are developed through long process of training on previously known datasets. More surprisingly, ANNs can successfully extract main features from data not used in training process. In other words, ANNs can do accurate predictions of unknown data without any restrictions on the size of distribution of it. As a result, ANNs became widely used in many applications as Self-driving cars, Weather forecasting, image recognition and data analysis.

Convolution Neural Networks (CNNs) are the most used type of ANNs in visual recognition and image classification. Among all ANNs, CNNs use special processing units that sweep many convolution kernels with different weights over the input image to capture the main two-dimensional features existed. In addition, these convolution kernels have small dimensions compared with the dimensions of input images which results in

having fewer parameters to be learned and stored compared with the huge parameters of interconnected neurons of ANNs. As a result, CNNs gained large popularity recently especially with the availability of many powerful processing units and training datasets as ImageNet Large Scale Visual Recognition Competition (ILSVRC) [1] and COCO dataset [2].

Although the convolution kernels reduced the overall number of parameters of the network, CNNs still use over 100 million parameters to achieve high accuracies [3]. Also, they need powerful processing units to be able to perform the huge parallel computations and data reuse processes. Graphic Processing Units (GPUs) are the most powerful platform for such complex algorithms but with very high cost in terms of power. Recently, CNNs become widely used in real-time embedded systems requiring platforms with high speed, small area and very low power consumption which are not achievable in GPUs. Power problems of GPUs and CPUs arise from their generality, no matter how direct the process is, it must be translated into set of instructions, then fetched from memory, executed by the general hardware and the stored back in the memory. Therefore, specific platforms for CNNs with higher resources for parallelism and pipelining are needed.

Dedicated hardware implementations are always faster than GPUs which made Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA) platforms more promising for neural networks. Although ASICs are better than FPGA based implementations in terms of area, power and speed, their high cost and complex, time consuming design flow makes them less favorable in this area. In addition, FPGAs have a great degree of flexibility as they can be reconfigured -statically and dynamically.

In addition to their flexibility, modern FPGAs have very powerful features that incredibly boost the performance CNNs. First, they have embedded CPUs that could be used to implement the area hungry parts of the hardware. In addition, they offer high degree of freedom for parallelism and pipelining. Also, they have relatively large embedded memories along with the flexibility to use external memories -but with speed cost. Those features gave FPGA based CNNs an increasing popularity recently.

However, there are limiting tradeoffs in FPGA designs that need to be solved. Firstly, usage of embedded CPU greatly boosts the speed but is very power inefficient for embedded systems. Also, speed is very degraded by the slow connections to external memories which are needed due to the limited internal memories. Those bottlenecks are making it vital for CNN designs to adopt optimization techniques that reduce the number of computations and parameters without affecting the overall accuracy of the network.

1.2. Problems to be solved

Deep CNNs are greatly needed in embedded systems, smartphones, wearable electronics and self-driving cars. In order to achieve efficient real-time performance, they should have small latency, high throughput along with low power consumption. In addition, Higher accuracy is required, and this is achieved by the long training process that need huge amount of data. Also, high accuracies limit the possibility of parameter pruning which means massive memories are needed.

Memory and Power problems are not a big concern in the training process as it is done only once to evaluate the parameters, so it could take place on GPUs or computers, then the resulting parameters get transferred to the optimized CNN platform. However, during the classification phase these problems are very critical. Three main platforms are the candidates for accelerating CNNs: GPUs, ASICs and FPGAs.

ASIC designs are fully customized to the computations needed so they achieve the highest speed with minimum power consumption. In addition, massive memories could be designed with very high speed to account for the deepest CNN. However, the design process, synthesis and fabrication are very complex and expensive in terms of time and finances.

On the other hand, Soft-Ware based platforms (GPUs and CPUs) require very easy design flow and are not very expensive. Also, their high speed and large memories makes them very powerful engines for training process. However, their massive power consumption is critical in classification process as mainly the target systems are battery-dependent.

FPGAs seem to be the solution for this trade-off. In comparison with ASICs and GPUs, they are less expensive and simpler to program and reconfigure. In addition, they provide low power consumption and high throughput. also, FPGAs offer many resources like Block Random Access Memories (BRAMs), Arithmetic Logic Units (ALUs) and CPUs.

The main problem of FPGAs is that their internal memories are not large enough to store all the parameters needed for such huge parallel processing units. External memories solve this problem but their limited bus size, read and write speed greatly degrade the throughput of the CNN which is the main target for real-time applications.

1.3. introduced solution

FPGAs are programmable and very flexible which allow implementing several optimization techniques to reach the target performance. This thesis is targeting lower power consumption, low external memory access along with high speed. In order to achieve this purpose, several optimization techniques were used.

First, parallel processing is used to speed up the performance of all layers. Along with pipelining techniques to allow for higher clock frequency and in turn higher throughput.

Second, data reuse techniques were used to lower the internal BRAMs access which will save much power and in turn lower the external memory access which is also time and power consuming.

In order to lower the computation s power, approximate computations were used. Floating numbers are of constant length of minimum of 32 bits with no restrictions of the length of fractions. This makes floating representation accurately represent the extremely small of large numbers which leave no option for simplifying the operations on floating numbers. Instead of accurate, floating-point representation of numbers, fixed-point representation was used. In which, all numbers are represented wit constant lengths for fraction and for the whole number. This quantization saves a lot of computational power with very small reduction in the accuracy of the overall network. Along with the Rectified Linear Unite layers which also help in power reduction by setting the negative

values to zero after each convolution layer. Negative values are commonly existed especially in deeper layers of the CNN, as main features get represented with positive values and higher weights while absence of features is represented by negative, close-to-zero values. By setting these values to zero, ReLU layers help in skipping large amount of computations [4].

Moreover, we used same Hardware architecture to perform all the layers of the same type no matter the differences in their parameters. This helped to save a lot of power and area to be used in building distributed memories to reduce external memory access.

1.4. Organization of coming chapters

This thesis introduces new accelerated FPGA based implementation of the VGG neural network but first we will discuss some literature view of CNNs and move to the VGG algorithms and implementation and then to the proposed implementation and the results.

Chapter 2. discusses an overview about ANNs and CNNs and their building layers then it moves to the VGG network and provide a comparative study between it and other popular CNN algorithms. After that, the specifications and mathematical functions of different layers of the VGG are explained. Also, it explains how the training process takes place and then move to an overview about the FPGAs, their architecture, design flow, available resources and how they are used to accelerate the CNNs.

Chapter 3. explains the software-based algorithm of VGG which was used for the training process. It also discussed the approximate representation of numbers used, datasets used in training and the achieved accuracy. In addition, it provides information on the format of the input parameters to the Hard-ware implementation.

Chapter 4. starts with the proposed optimization techniques used in the design and then explain in details the architectures of each layer and memory and the top integration of them all.

Chapter 5. represents the verification of the designed hard-ware in comparison with the SW results. In addition to explaining the reached performance of the network in terms

of power, speed and area. Also, a comparative study with the previous VGG implementation is presented.

Chapter 6. provide a brief overview of the thesis and the reached conclusions. It introduces some ideas for future works in the same field.

Chapter 2.

Background and previous work

2.1. CNNs

2.1.1 Artificial Neural Networks overview

ANN are mathematically modeled by a set of algorithms to mimic the human brain. It is inspired by the human brain interactions and designed mainly for patterns' recognition. The human brain can recognize voices and images in few milliseconds and we are usually do this without even knowing how it works but this is not the case when it comes to machine learning. Those patterns must be numerically expressed in vectors so; all real-life date must be translated into numbers and vectors so that the network can deal with them and recognize them. Foe example, RGB images consist of pixels, each pixel is a number describing its color code from 1 which is white to 256 which is dark green [5]. Any change of any number will lead to another image. ANN can classify and group data according to similarities among the inputs after training them on some specific datasets [5]. This can be used on many applications like face detection, image classifications, figure print pattern recognition, voice recognition and so many others.

2.1.2. Convolutional Neural Networks overview

CNN is one of the most popular ANN and named after the mathematical operation, convolution [5]. It is built by main blocks called layers each layer is responsible for some cascading mathematical operations that lead to finally recognize whatever the input is, depending on the training and the dataset. Those layers are: convolution and ReLU layer, pooling layer, fully connected layer and soft-max layer. We will explain them in detail in the next section.

2.2. VGG Network overview

The VGG neural network is the first deep network, more than 8 layers, among the convolutional neural networks as shown in figure (1). It was done by Karen Simonyan and Andrew Zisserman and named after Visual Geometry Group (VGG), Department of Engineering Science, University of Oxford. The main objective of this work was to study the effect of convolutional network depth on its accuracy in the large-scale image recognition setting [5]. They got the first and the second places in the localization and classification tracks respectively in ImageNet Challenge 2014 after their improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. Which is the smallest size to capture the notion of left/right, up/down, center. They evaluated 2 models, 16 and 19 weight layers which can classify images into 1000 object categories, such as cars, pens, T-shirts, and many animals. We used VGG16 model which was trained for weeks using NVIDIA Titan Black GPU's by Simonyan and Andrew Zisserman.

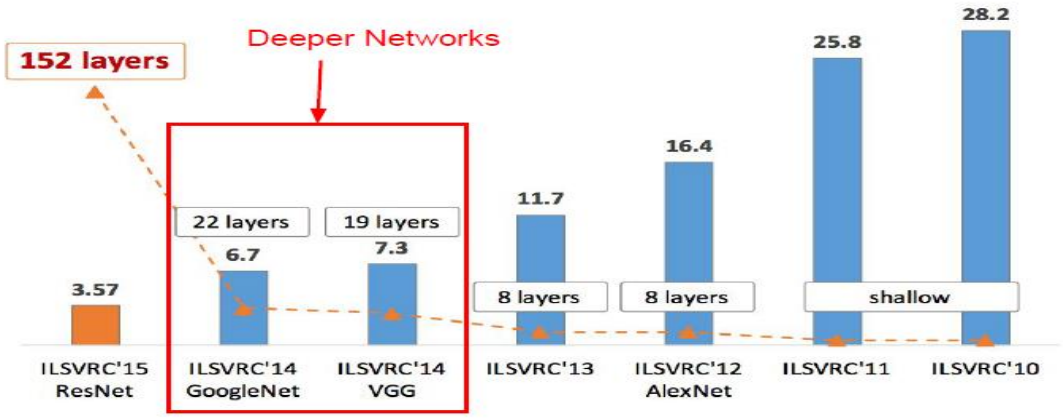

Figure 1: ImageNet Large Scale Visual Recognition Challenge. [5]

2.3. comparative study

Among all CNNs, VGG network has many advantages in terms of accuracy, speed and power.

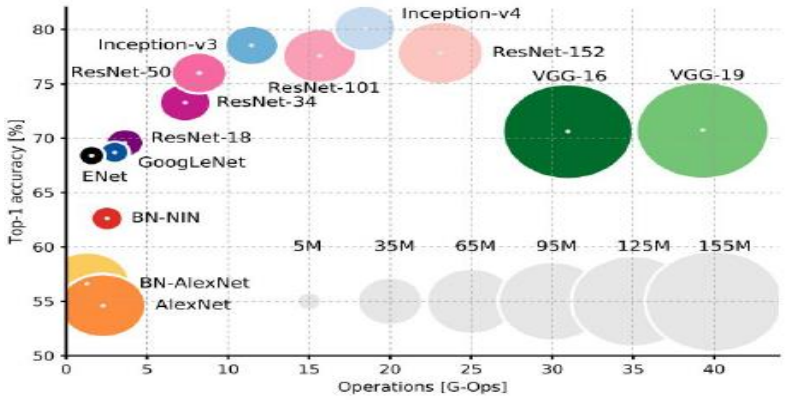

Figure 2: Comparison of top-1 accuracy and operations between the different architectures [6].

As shown in figure (2), VGG network has the highest memory and the most operations [G-Ops] compared with other architectures and has a satisfying top-1 accuracy percentage compared to other deep networks as shown in figure (3) which can be considered as disadvantages, according to [6].

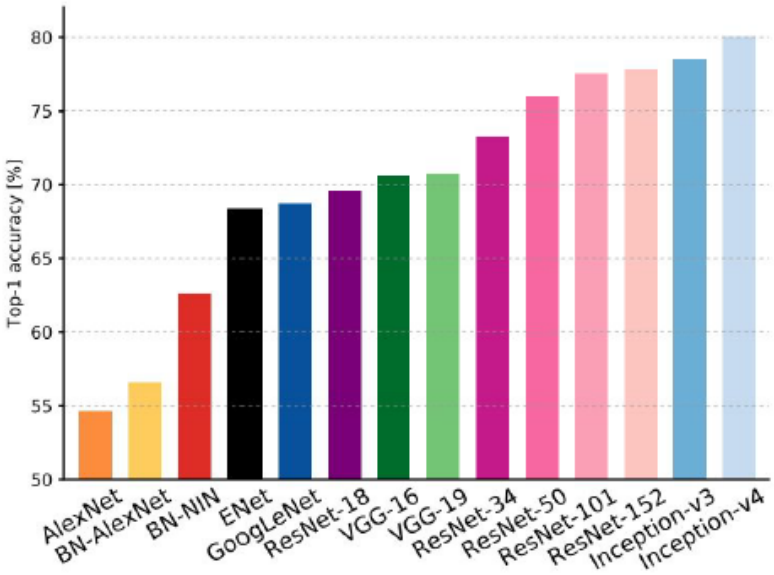

Figure 3: Comparison of top-1 accuracy between the different architectures adopted from[6].

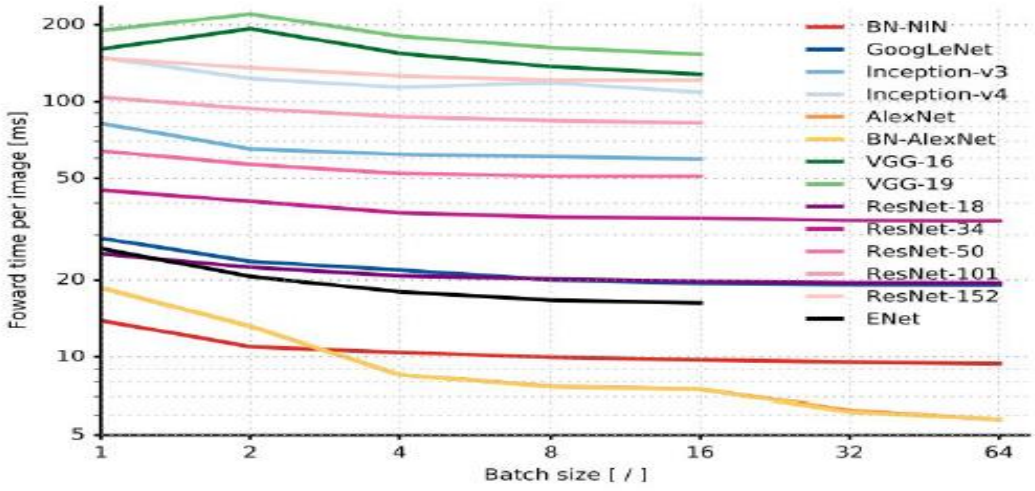

Figure 4: Comparison of forward time per image and batch size between the different architectures adopted from [6].

As shown in figure (4), VGG network has the longest forward time per image (delay) compared with other architectures with the same batch size, which can be considered as disadvantages, according to [6].

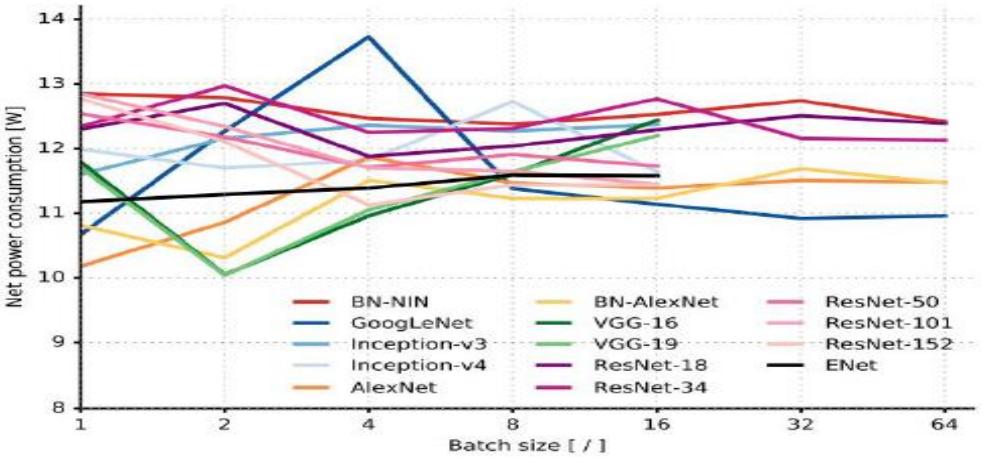

Figure 5: Comparison of net power consumption and batch size between the different architectures adopted from [6].

As shown in figure (5), VGG network has the lowest power consumption compared with other architectures with the same batch size, which can be considered as one of its advantages, according to [6].

2.4. VGG architecture

Karen Simonyan and Andrew Zisserman in [5] worked on six configurations for the VGG architecture listed below in table (1) named from A to E. Starts with configuration A with 11 weight layers (8 conv. layers and 3 FC. Layers) then configuration B with 13 weight layers (10 conv. layers and 3 FC. Layers) increasing the number of conv. layers as they go until they reach deeper configuration which is E with 19 weight layers (16 conv. layers and 3 FC. Layers). The number of channels is increased by a factor of 2 as they go deeper staring from 64 channel in the very first conv. layers until they reach 512 channels. The number of parameters is also increased by increasing the number of layers as shown in table (2). [5]

Table 1. ConvNet configurations [5]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2. Number of parameters [5].

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

The main building blocks of VGG or any CNN are the feature extractor part and recognizer part. This thesis proposes an optimized implementation for the VGG-16 architecture shown in figure (6) below:

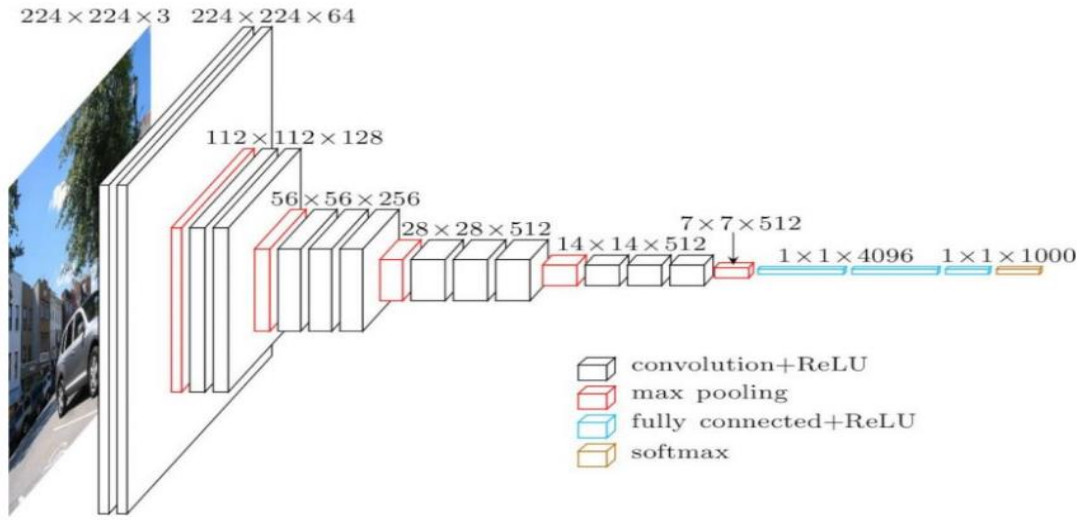

Figure 6: VGG-16 layers. Adopted from [11]

As the architecture in figure (6) shows, ReLU layer is not shown for brevity, the first stack of two 3x3 conv. layers act as an effective receptive field of 5x5, without the spatial pooling in between, and the next stack of three 3x3 conv. layers act as an effective receptive field of 7x7. -So, this architecture does the same job of the other architectures and achieved top-1 accuracy of 70.5% and top-5 accuracy of 90% with only 3x3 filters which reduces the number of parameters by 81%. i.e. 3x3 filter has $3(3^2C^2) = 27C^2$ parameters but with 7x7 filter, the number of parameters will be $7^2C^2 = 49C^2$ while C is the number of channels in each conv. layer[5]. They also incorporate three non-linear rectification layers instead of a single one, which makes the decision function more discriminative. [5] Those are the main reasons why we selected this architecture to study.

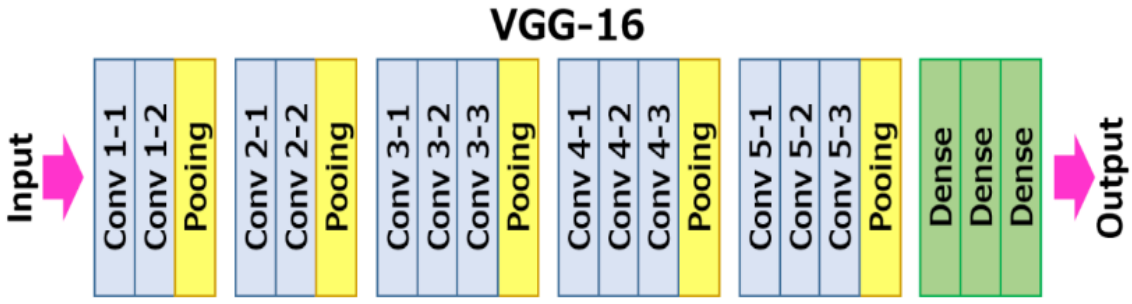

Figure 7: VGG layers adopted from [11]

2.4.1. Convolutional layer

Convolution layer is the very first layer in the network and works as the feature extractor. The 3x3 weight matrix is initialized to extract certain features from the image and run across the image pixels or matrix so that it reaches each pixel once to result into a convolved output that is ready to be cascaded to the nest layer for further operations [9]. Each output pixel is obtained by adding the values obtained by element wise multiplication of the weight matrix and its number of moving strides, for example, highlighted 3*3 part of the input image for the first output pixel as shown in figure (8). [9] The convolution layer output dimensions differ from the input due to the convolution process as the output depth is dependent of the number of filters of the layer and the 2D dimensions depend on the stride movement. All filters sweep over the entire input image in all the dimensions. If the filter window moves by one pixel at a time this is called 1x1 stride.

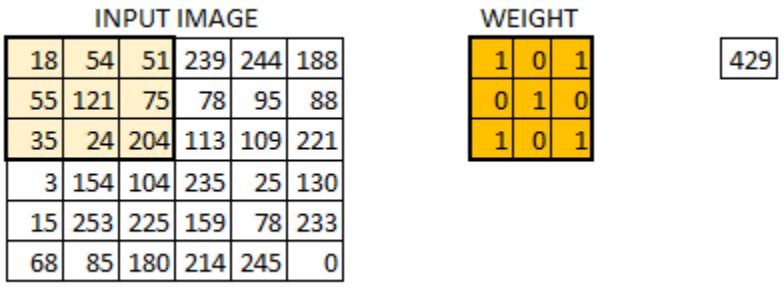

Figure 8: Convolution operation. [9]

As claimed, the main purpose of the convolution is to extract the main features and edges from image. For visualization of the process, Figure (9) shows the outputs of many serial convolution functions on the test image in figure (10).

Figure 9: Test Image of ImageNet dataset

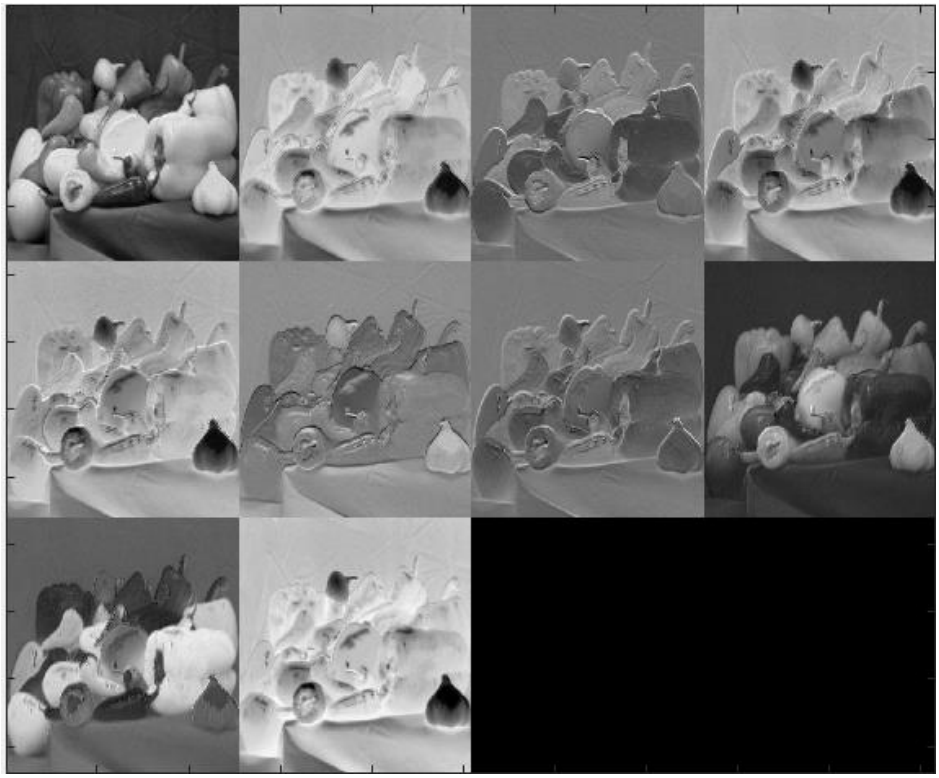

Figure 10: extraction of main features by consecutive convolutions

The CONV layer's parameters consist of a set of learnable filters. The VGG-16 network consists of 13 conv/ layers. The input to the first conv layer is a fixed-size 224 x 224 x 3 (RGB) image which was preprocessed by subtracting the mean RGB value from each pixel [5]. The image is then passed through the stack of conv. layers where they used 3x3 filters or 1x1 filters which can be considered as a linear transformation of the input channels followed by non-linearity [5]. The stride movement of all layers is fixed to be 1 pixel. The spatial padding is chosen such that the resolution is kept preserved i.e. 1 pixel for 3x3 convolution layers [5]. The most memory consumption is in the early layers; as their feature maps contain 224x224x46pixel = 3.2M pixel. However, it has the smallest number of parameters due to small number of filters; the first layer for example has 3x3x64=1728 parameter [6].

2.4.2. ReLU Layer

ReLU stands for Rectified Linear Unit and it is used to introduce non-linearity to the system after each convolution layer because the element-wise matrix multiplication and addition that was done in the convolution stage are linear operation and each real-life date is non-linear. So, this will make it easier for the model to generalize or adapt with variety of data to best fit its representation and to differentiate between the outputs. The ReLU function is done by eliminating all the unwanted negative-valued features and replace them with zero by applying the activation function max (0, x). It is now used instead of other non-linear functions like sigmoid and tanh for more computational efficiency. The ReLU function plot is shown in figure (11).

As the convolution layers in CNN only output positive values for existence of features, a large amount of RELU outputs will be zero and do not have to be used for further computations [14]. As a result, ReLU function has a powerful role in reducing the computational power by skipping many computations due to the zero-valued pixels.

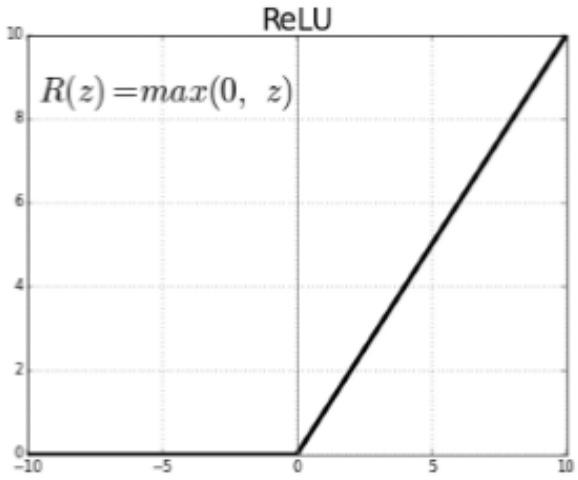

Figure 11: Plot of ReLU function outputs

2.4.3 Pooling layer

This layer follows some of the conv. layers to reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting. It can be done by more than one way but the most commonly used one is by taking the maximum of each filter size of the input volume, in our case it is 2x2 applied with a stride of 2 which down samples the input volume by 2 in both width and height but the depth remain unchanged, decreasing the 75% of the activation. Figure (12) illustrates the max pooling operation.

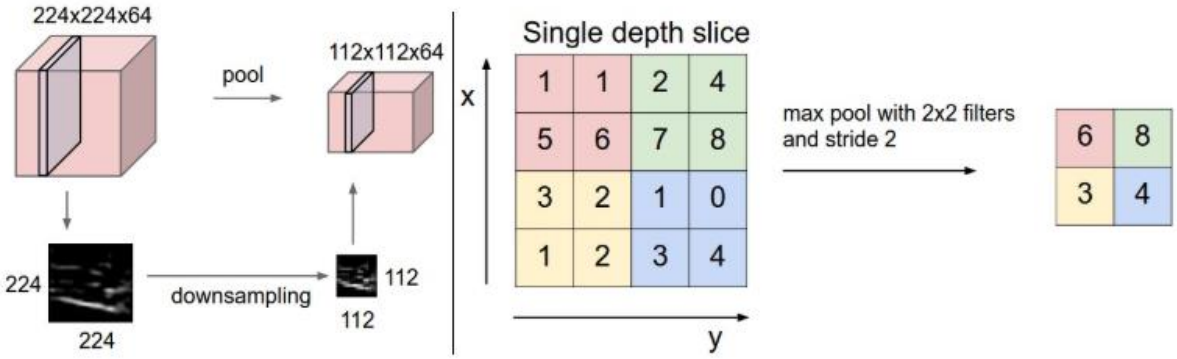

Figure 12: Max-Pooling operation explanation.

This is the following layer after the conv. layers. The VGG network contains 5 max pooling layers each one is between two successive conv. layers. The pooling here is done

by the most common way in CNN which is taking the maximum of each number of pixels which is 4 pixels because filter size is 2x2 with a stride of 2. The input is down sampled by 2 with the same depth dimensions discarding 75% of the activations.

Max-pooling layer also contributes a lot to power-consumption optimization by reducing the feature-map dimensions without any loss of the important features. Figure (13) shows the max-pooling output of the image in figure (9) and as seen, pooling successfully kept all the important features and edges that are needed for classifying the image.

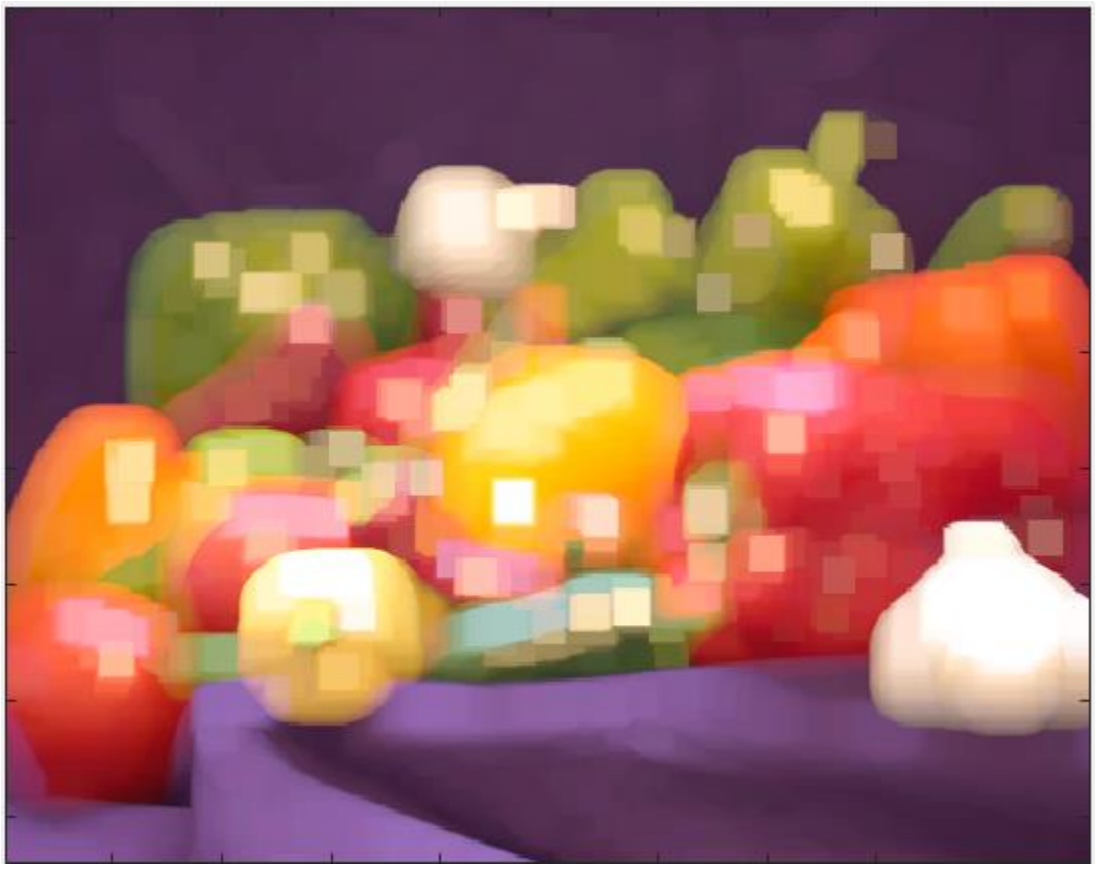

Figure 13: output of Max-Pooling of the image in figure (9)

2.4.4. Fully connected layer

In FC layers, every node is fully connected to every activation node in both the next layer and the previous layer just similar to the way that the neurons are fully connected in the brain neural network as shown in figure (14).

The most common drawback in this layer is that it has the highest number of parameters that need a lot of complex computations in both modes of operation; training and classification so that, the most commonly used number of FC layers is three [8]. The only difference between FC layers and conv. layers is that FC layers have more parameters than conv. layers and the neurons in the conv. layer is connected only to a local region in the input. However, both have identical functionality of dot products and addition, there for the can be implemented with same hardware [8].

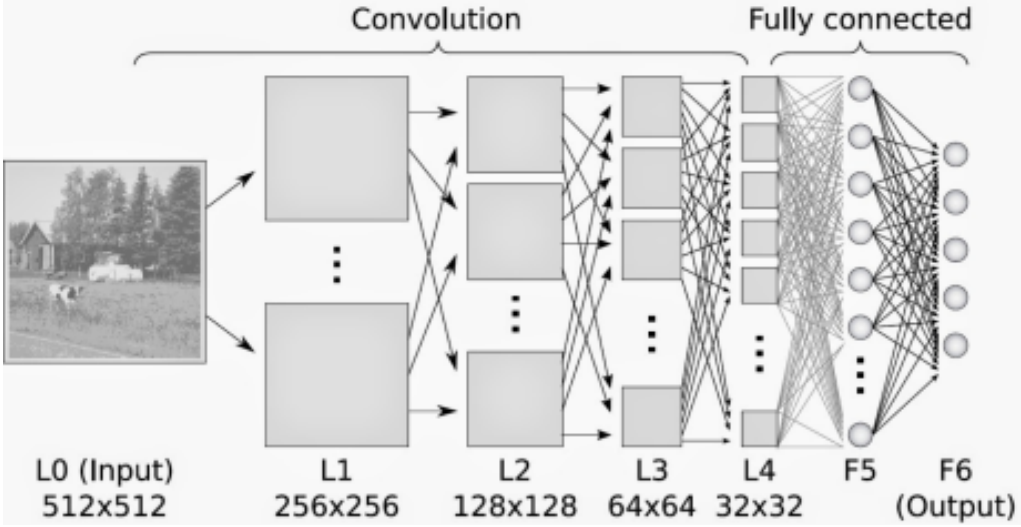

Figure 14: Fully-Connected layers structure.

In VGG-16, the first two FC layers have 4096 neurons each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class) [5]. Unlike the earlier conv. layers; those last layers have the highest number of parameters reaching around 4096x1000=4096000 parameters per layer. However, they have the lowest memory consumption for feature maps of 1x1x4096=4096. All numbers of parameters and memory consumption are reported in table (2).

2.4.5. Soft Max layer

Soft-max is the layer that decides the class which the input image belongs to. It takes its inputs from the Fully Connected layer's outputs and assign a value for each class that differ in range from 0 to 1.0 and corresponds to the class probability. The summation of those probabilities will add up to 1.

For an input vector $X = (x_1 + x_2 + \cdots + x_k) \in R$ where k is the total number of classes.

The Softmax function S is calculated through the following equation

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{j=k} e^{x_j}} \quad \text{Where} \quad j = 1,2,3, \dots, k \qquad (1)$$

Figure (15) shows an example of normalization of the inputs [0.1, 1, 2] into the probability distribution that predicts the category or the class of the input image. In the shown example we have an output probability of [0.1, 0.2, 0.7] which means that the input image belongs to the third class by a 70 percent probability.

Figure 15: Soft-Max probability distribution

Softmax function is a normalized exponential function that enlarges the probability of the highest score class in order to be distinguishable from other classes, and this relation is clearly shown in figure (16).

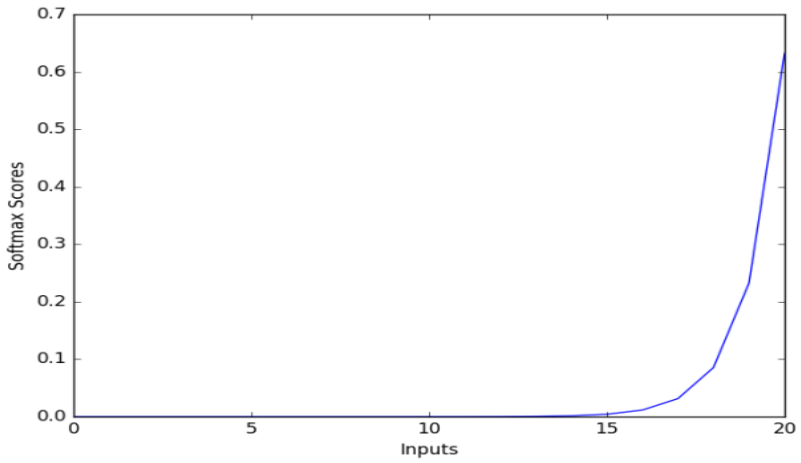

Figure 16: Plot of Soft-Max function score. [10]

In VGG network, Soft-max layer consists of 1000 channels. It gives the classification depending on the highest probability using the soft max function which is $\sigma(z)_i \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}\ for\ j = 1, \ldots., k\ and\ z = (z_1, \ldots \ldots, z_k) \epsilon\ R^k$ that gives a very high probability for the most commonly appeared classes and gives a very small probability for the rarely appeared classes.

2.4.6. Classification experiments results

Karen Simonyan and Andrew Zisserman reported the accuracies of VGG implementations with different depths in [5]. As observed in table (3), top-1 and top-5 error are decreased as we go deeper from 11 layers in configuration A until they reach 19 layers in configuration E. That is configuration D was selected in this thesis to be implement, with top-1 error that reaches 25.6% and top-5 error of 8.1%.

Table 3. top 1 and top 5 error percentages of different VGGs [5]

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

From table 4. TOTAL memory of 24M * 4 bytes ~= 96MB / image needed is to store 138M parameters without accounting for biases.

Table 4. Number of parameters and needed memories for the layers outputs

Layer	Type	Channels	Filter size	Memory	Parameters
1	Conv.	64	3x3	224*224*64=3.2M	(3*3*3) *64 = 1,728
2	Conv. + max	64	3x3	224*224*64=3.2M +112*112*64=800K	(3*3*64) *64 = 36,864
3	Conv.	128	3x3	112*112*128=1.6M	(3*3*64) *128 = 73,728
4	Conv. + max	128	3x3	112*112*128=1.6M + 56*56*128=400K	(3*3*128) *128 = 147,456
5	Conv.	256	3x3	56*56*256=800K	(3*3*128) *256 = 294,912
6	Conv.	256	3x3	56*56*256=800K	(3*3*256) *256 = 589,824
7	Conv. + max	256	3x3	56*56*256=800K +28*28*256=200K	(3*3*256) *256 = 589,824
8	Conv.	512	3x3	28*28*512=400K	(3*3*256) *512 = 1,179,648
9	Conv.	512	3x3	28*28*512=400K	(3*3*512) *512 = 2,359,296
10	Conv. + max	512	3x3	28*28*512=400K +14*14*512=100K	(3*3*512) *512 = 2,359,296
11	Conv.	512	3x3	4*14*512=100K	(3*3*512) *512 = 2,359,296
12	Conv.	512	3x3	4*14*512=100K	(3*3*512) *512 = 2,359,296
13	Conv. + max	512	3x3	4*14*512=100K +7*7*512=25K	(3*3*512) *512 = 2,359,296
14	FC.	4096	-	4096	7*7*512*4096 = 102,760,448
15	FC.	4096	-	4096	4096*4096 = 16,777,216
16	Output + softmax	1000	-	1000	4096*1000 = 4,096,000

2.5. Training process

Like human brains, CNNs need a training period in order to formulate accurate parameter values for better classification. The training process is done by back propagation. Back propagation can be done in 4 steps: The first step is to take the training image and pass it through the whole network layers. This is called forward pass. All the initialized weights are randomly generated with no preferences to any of them, so the network need more information so that it can tell the classification. This leads us to the next step of the pack propagation which is loss function. The most common way to make the loss function is by mean squared error which is $E = \sum \frac{1}{2}(\text{actual} - \text{predicted})^2$. The predicted output of the CNN must have a minimized error in order to function correctly. This can be done by optimizing the weight values that directly contribute in the loss function. The next step is back-word pass that get the most contributing weights that affect the loss or the error and reduce them. Then the last step is to update the weights.

2.6 FPGAs

Field Programmable Gate Array (FPGA) is a customer configurable Integrated Circuit (IC) that can be programmed and reprogrammed after manufacturing, consequently; it takes an advantage over the Application Specific Integrated Circuit (ASIC) as it can be updated and reconfigured. FPGAs configured by hardware description languages (HDL) which are Verilog and VHDL. This configuration is stored in a volatile Random-Access Memory (RAM), consequently, it is lost every time the power supply disconnected, and the FPGA needs to be configured again when the power supply is available.

2.6.1. FPGA architecture

FPGA consists of three main components shown in Figure (17):

- **Programmable Logic Blocks**

  The programmable logic block provides basic computation and storage elements used to implement the needed function. It is further decomposed to sub-components like Lookup tables (LUTs), Multiplexers and Flipflops.

Modern FPGAs contain a heterogeneous mixture of different blocks like RAM blocks, ALUs, and digital signal processing (DSP) blocks which perform operations such as multiplication, Moreover DSP is widely used in CNN.

- **Programmable Interconnects**

  The programmable routing establishes a connection between logic blocks and Input/Output blocks to complete a user-defined design unit.

- **Programmable Input/Output Blocks**

  The programmable I/O pads are used to interface the logic blocks and routing architecture to the external components. These cells consume a large portion of the FPGA's area.

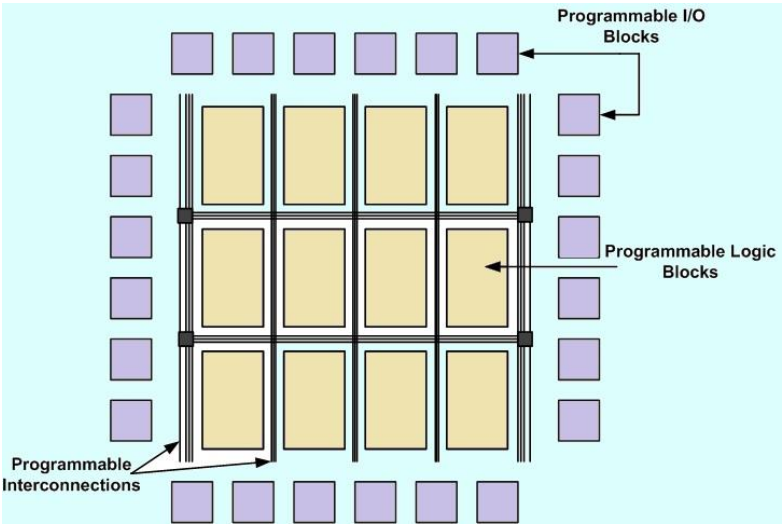

Figure 17: FPGA architecture. adopted from [15]

2.6.2. Design Flow

The FPGA design flow is graphically represented in Figure (18) and it consists of six stages: design entry, design synthesis, design implementation, device programming and design verification.

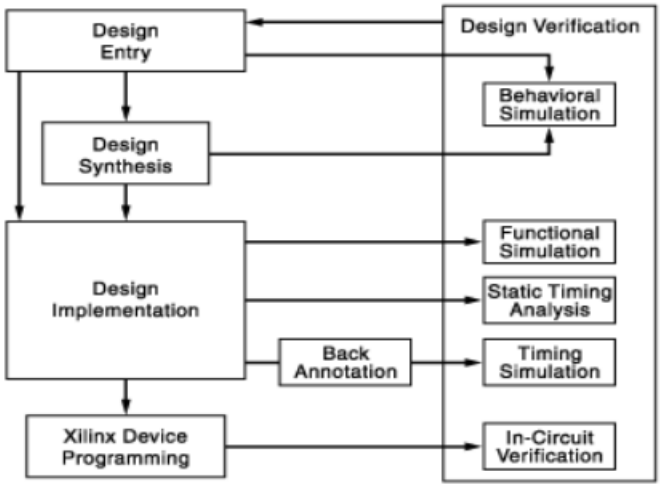

Figure 18: FPGA design flow. adopted from [16]

Design Entry

The design entry is done in different techniques like schematic based, hardware description language (HDL) and a combination of both. Selection of a method depends on the design and designer, if the designer wants to deal with hardware, then the schematic entry is a good choice, but if the designer thinks the design in an algorithmic way, then the HDL is the better choice. The schematic based entry gives the designer a greater visibility and control over the hardware.

Design Synthesis

This process translates VHDL code into a device netlist format. The design synthesis process will check the code syntax and analyze the hierarchy of the design architecture. This ensures the design optimized for the design architecture. The netlist is saved as Native Generic Circuit (NGC) file.

Design Implementation

The design Implementation consists of three consecutive processes illustrated in Figure (19):

1) Translate

This process combines all the input netlists to the logic design file which is saved as NGD (Native Generic Database) file. Here the ports are assigned to the physical elements like pins, switches in the design.

2) Map

Mapping divides the circuit into sub-blocks such that they can be fit into the FPGA logic blocks. Thus this process fits the logic defined by NGD into the combinational Logic Blocks, Input-Output Blocks and then generates an NCD file, which represents the design mapped to the components of FPGA.

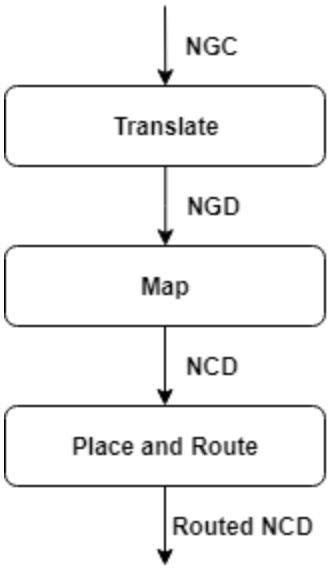

Figure 19: design implementation

3) Place and Route

The routing process places the sub-blocks from the mapping process into the logic block according to the constraints and then connects the logic blocks.

Device Programming

The routed design must be loaded into the FPGA. This design must be converted into a format supported by the FPGA. The routed NCD file is given to the BITGEN program, which generates the BIT file. This BIT file is configured to the FPGA.

Design Verification

The design is verified through the following steps:

1) Behavioral Simulation

Behavioral simulation is the first of all the steps that occur in the hierarchy of the design. This is performed before cheap lace dresses the synthesis process to verify the RTL code.

In this process, the signals and variables are observed and further, the procedures and functions are traced, and breakpoints are set.

2) Functional Simulation

Functional simulation is performed post-translation simulation. It gives the information about the logical operation of the circuit.

3) Static Time Analysis

This is done post mapping. Post map timing report gives the signal path delays. After place and route, timing report takes the timing delay information. This provides a complete timing summary of the design.

Chapter 3.

Software implementation

Training a hardware configuration as large as a 16-layer deep VGG network can be quite exhaustive computationally and in terms of area. A clever approach to cut down the costs is to perform training on the exact software version of the required network, then plug-in the results directly in the hardware memories. Therefore, in order to evaluate the performance and accuracy of our VGG hardware implementation, we had implemented a pre-trained model [3] and employed its weights and biases in our design.

3.1. Datasets used for training

ImageNet is a large dataset of approximately 22k classes, amounting in total to more than 15M labeled images. The images were collected from the web and manually labelled by the aid of Amazon Mechanical Turk. An annual competition, called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) is being held since 2010 to test the performance of the different ConvNets designed every year. The competition is based on a sub-set of the ImageNet dataset, where the ConvNet performances are evaluated for top-1 and top-5 errors.

Training and evaluation of the pre-trained VGG ConvNet architecture was carried out on the ILSVRC-2012 dataset. The dataset is composed of 1000 classes and partitioned into 3 categories; 1.3M training images, 50k validation images, and 100k test images with class labels. The architecture was tested in the ILSVRC 2012 to 2014 challenges.

3.1.1 Image padding

As we apply the convolution, the output volume shrinks by a factor of the stride number, in case of using a 3x3 stride it will be reduced by a factor of 3x3. There are some considerations that should be taken, like making the receptive field fit the diminutions of the input volume to make sure that there is no any missed data. In case of miss fitting, zero padding is added as a border around the image pixels as indicated in figure (20) and its size is determined by fitting the resolution and the stride size of the convolution. The

padding size is determined by the next equation $Zero\ padding = \frac{(k-1)}{2}$, while k is the filter size. In our case of 3x3 filter size, the padding size is 1x1.

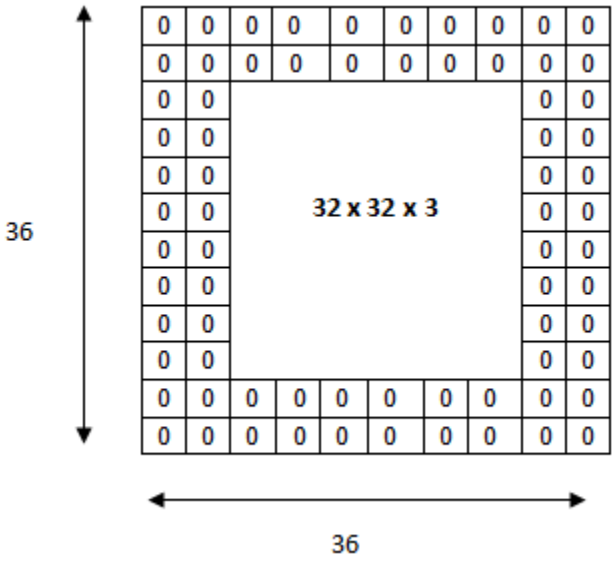

Figure 20: visualization of zero-padding

The formula of the output size for any conv. layer is given by $O = \frac{(W-K+2P)}{S} + 1$ while O is the output size, w is the output height/length, p is the padding, k is the filter size and s is stride.

3.2. Training details

The VGG in [3]was trained using mini-batch gradient descent and backpropagation with momentum. Batch size and momentum of 256 and 0.9, respectively, were chosen for training with an L2 regularization coefficient of 5.10^{-4} for weight decay. Momentum is normally used to keep the fluctuations in weight change to a minimal during training to prevent a noisy output. The L2 regularization coefficient is an important hyper-parameter, as well, in determining the proper fitting of the results to the input data. If this coefficient is set to zero, the results may over-fit. On the other hand, if it is set to a very large value, under-fitting would occur.

The learning rate, which defines almost the most important hyper-parameter, was initially set to 10^{-2} and was manually decreased each time the accuracy saturated until

the accuracy reached a maximum value after 3 external interferences. Also, for the fully-connected layers drop-out regularization was performed with a drop-out ratio of 0.5, which entails the optimum fitting for the results. Learning was eventually stopped after 74 epochs, constituting to 370k iterations in total.

Weight initialization can be challenging in deep networks, as bad initialization can set the gradient to unsuitable values at the early stages which may lead to the exploration of undesired solution spaces that are far from the optimal weights. This situation could result in slow convergence or even no convergence at all to the desired solution space with the optimal results. In order to address this problem, the authors in [3] designed multiple ConvNet architectures with roughly the same structure but removed some replicated layers to produce shallower versions of the same network. They designed an 11-layer VGG, which was shallow enough to initialize randomly and still produce satisfying results. The weights that gave the best accuracy for the 11-layer VGG were used to initialize the first 4 Convolution layers and the last 3 fully-connected layers in the 16-layer VGG, while the intermediate layers were given a random normal sampling. The biases, however, were initialized with zero, which was enough to allow good convergence.

3.3. Fixed point back ground

With the increasing complexity of ConvNet architectures, bigger computational capacity and memory resources are required for hardware implementation. To mitigate this problem, fixed-point notation is used for memory storage and arithmetic computations of pixels and weights instead of floating-point notation. As seen in figure (21) , the fixed-point representation divides a word length (n) into a sign bit, a fixed integer component and a fixed fractional component.

Figure 21: Fixed-Point representation of numbers

The maximum and minimum values that can be stored in a word of x bits and y scale factor (exponent) are $-2^{x-1}/2^{y}$, and $(2^{x-1}-1)/2^{y}$, respectively. In short, a word is stored in two's complement format but with fixed radix point positioning. This gives the advantage of storing values in memory and performing arithmetic operations in integer form, thus enhancing the hardware performance.

3.3.1. Fixed point arithmetic operations

The choice of a radix point for the results of an arithmetic operation is a designer's choice. Therefore, we preprocessed the input weights and biases to our ConvNet design to a standard fixed-point representation having a word length of 17 bits and a fractional length of 9 bits. For fixed-point addition, this approach allows the radix point of 2 different words to align easily, giving the result the same radix point. A fixed-point adder is employed to truncate additional bits and put the result in standard form, which is the same as the input's, for additional computations. For fixed-point multiplication, however, the operation performed is exactly the same as two's complement multiplication. A fixed-point multiplier is employed to determine the radix point of the result and then truncate it to put it in standard form for further operations. Here, truncation also serves as a means to simplify computation and save area and power in our design.

3.2. Software Accuracy results

The 16-layer pre-trained VGG we used from [3] was evaluated in the ILSVRC-2014 challenge and won second place with a top-5 test error of 7.3%, as referenced in Table (5). A test error of 7.3% was obtained from the combination and averaging of the results from the 7 VGG networks of different number of layers designed in the paper, which had enhanced the results due to the compatibility and similarity of the structures of the 7 topologies. The error was enhanced to 6.8% the following year by combining the results of 2 nets only. As seen in table 1, the VGG network surpasses the performance of the all the ConvNets designed before the time of the competition in 2014, except for GoogleLeNet. The best performance for GoogleLeNet outperformed the best performance for VGG by only 0.1%. On the contrary, if the test errors of both

configurations are to be compared relative to the performance of only one net, VGG would surpass by 0.9%.

Table 5. As represented in [3], A comparison with the state of the art in ILSVRC-2014 classification

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

3.3. Preparing data for Hardware implementation

The Pre-trained VGG in [3] takes in images of fixed dimensions 224x224. In order to do that we re-scaled some ImageNet images isotopically until the smaller side of the images reached 224 pixels, then we cropped them with the required fixed dimension from the center of the images. Also, to be able to store an image in the RAM memory of an FPGA, the image has to undergo flattening, such that it is stored in a one-dimensional array, each containing one pixel. An image with dimensions 224x224x3 becomes of dimensions 150,528x1 in the memory cells. Using Matlab, we unfolded the input image by unrolling the pixels of each of the 3 RGB channels, one after the other, as shown in

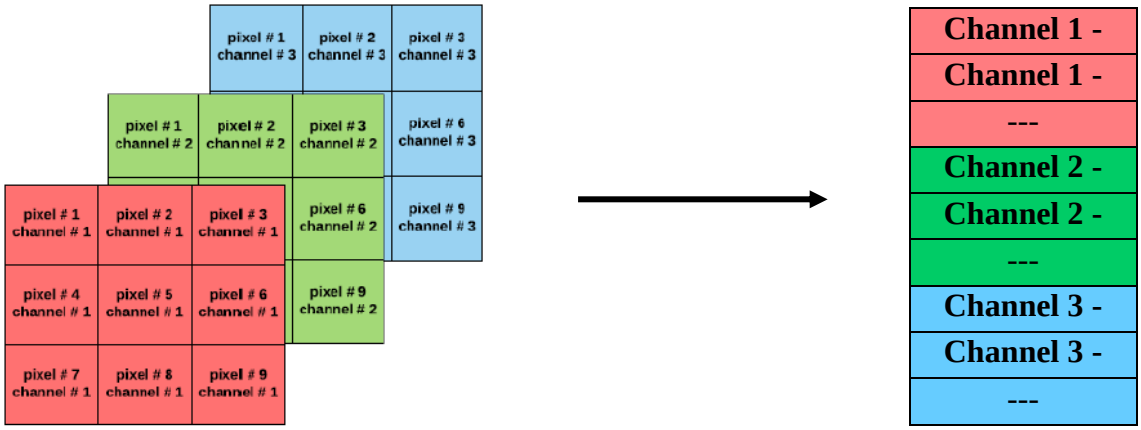

Figure 22: Image flattening in memory

3.4. Summary

This chapter presented the Soft-Ware implementation of the VGG network which this thesis aims to implement on FPGA. The soft-ware implementation was used for the training process as it has more memory resources to account for the massive-parameters and FMs. All the mathematical operations were held in Fixed-Point representation in order to expect same accuracy results as the HW .

Chapter 4.

Hardware Architecture

4.1. Proposed approach

Our proposed design aims to reach optimization between speed and power consumption through the following techniques:

4.1.1 use of local memory architectures

The proposed design takes one input picture at a time and does not support real time flow of images. The input image is read from external memory and stored in local memory and all the generated Feature maps from all the layers will get stored in local cashes to be used as input for the following layer. All weights and biases of convolution filters and Fully connected layer neurons are Previously loaded from the external memory to local different memories.

The use of local memories limited the external memory access to only the loading state at the start. External memory access consumes a lot of energy and connections and limits the speed of the CNN. Despite the larger area introduced by local memories, they saved a lot of energy and time as their accessing is much easier and faster.

4.1.2 reusing feature map pixels

As the convolution and Fully connected layers require reading the feature map once for each filter or neuron, Reuse technique was applied to lower the memory access. Each pixel is used for multiple filters' kernel and neurons at the same time to lower the number of recalling the data. This helped reducing the energy consumption of those large layers.

4.1.3 stationery outputs

All the intermediate calculations results are stored inside the layers not in the memory caches. Only the final outputs of the layers are stored in the memory. This saved a lot memory access -for intermediate results and partial sums- and hence saved a lot of power.

4.1.4 fixed point representations of numbers

All the calculations of the entire network are carried in fixed point format to avoid numbers inflation and lower the memory capacity. All numbers are represented as 17 bits binary numbers and all the mathematical operations results are quantized to 17 bits.

4.2. Top architecture

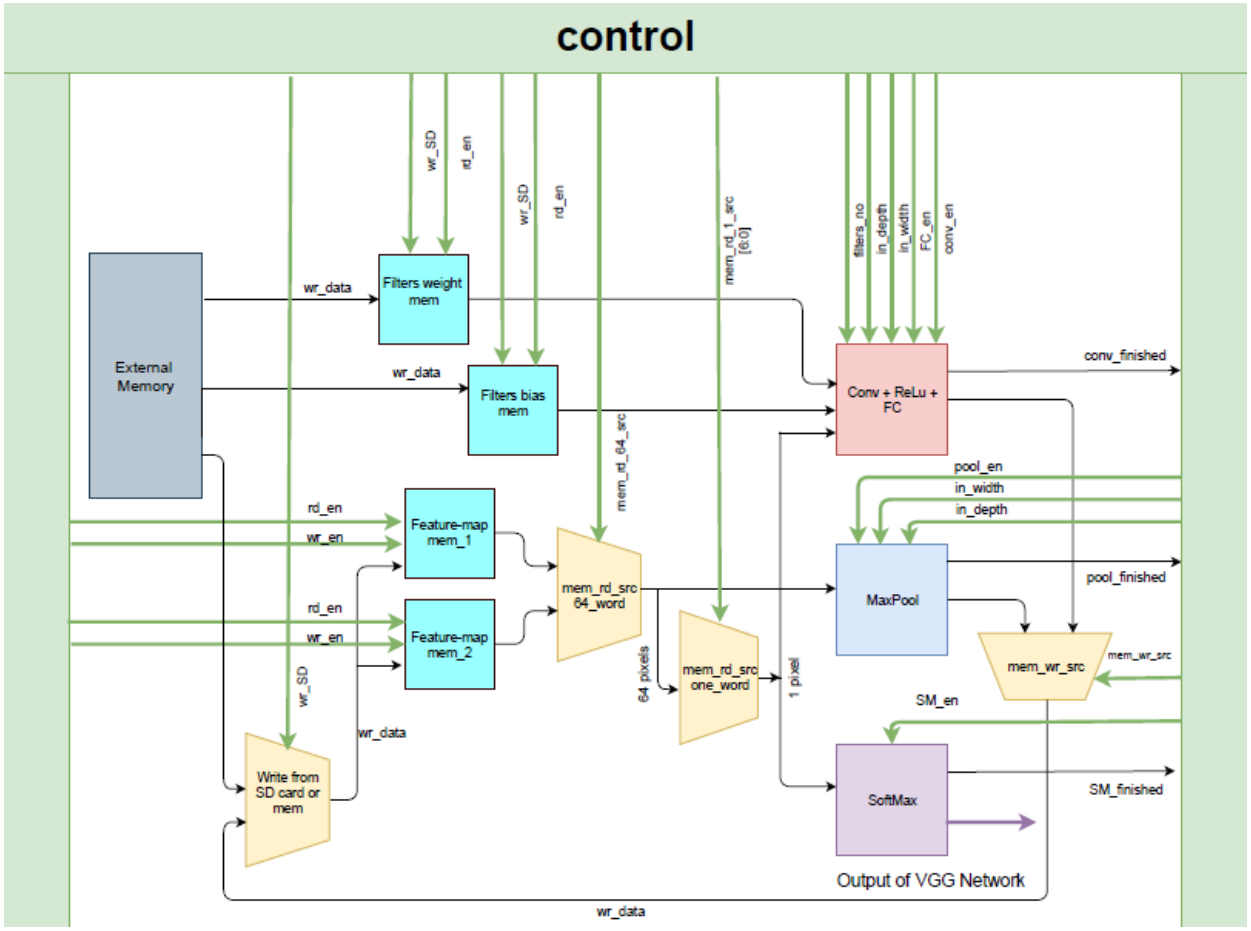

Figure 23: System-level Design of VGG-16

Figure (23) depicts the top architecture for our VGG-16 design. It consists of 3 main blocks, which perform convolution, max-pooling and soft-max operation. Since the operation of the fully-connected layers is similar to that of the convolution's but with different data arrangement, both are designed to operate on the same hardware and switching between them is governed by the controller.

The data path contains 2 main memories, in which the feature maps are stored. The image to be evaluated is initially installed in one of these memories through muxing between the input from the external memory and the output of the previous network stages. The feature map memory to be read from is selected through a mux for max-pooling and further selection is undergone for extracting only one pixel, if the operation to be performed is convolution, fully-connected or soft-max. Two Additional memories are used to store the weights and biases of the filters.

Due to the large memory requirement for the filters of the 16 layers, new weights and biases need to be written periodically from an external memory.

4.2.1 Control unit

The top control block is responsible for controlling the flow of data between CNN layers and memories. It is implemented as a finite state machine with one state for each layer of the VGG including one start state in which all the weights, bias and FM memories get initialized from the external memory. When all computation of a layer is finished, it sends a finished signal to the controller to move on to the next layer. It also controls the sources of weights and bias whether it comes from filters memory or FC memory.

4.3. Convolution layer and ReLU

Convolution layer is the main building block of all CNNs and the most critical layer in the VGG network. As illustrated in section (2.4.1.) the mechanism of the layer to convolve the input feature map with number of filters, each of them sweeps over the entire feature map. Each filter has specific kernel dimension, stride, weights and bias. In VGG network, all the filters have window of 3 x 3 x depth of the input feature map and they all have one stride movement and zero padding. All the other parameters of convolution layers are illustrated in table 6.

Table 6. The Parameters of the convolution layers of VGG-16 network

Convolution layers	Input feature map	Output feature map	Number of filters	Filter size
Conv1	224 x 224 x 3	224 x 224 x 64	64	3 x 3 x 3
Conv2	224 x 224 x 64	224 x 224 x 64	64	3 x 3 x 64
Conv3	112 x 112 x 64	112 x 112 x 128	128	3 x 3 x 64
Conv4	112 x 112 x 128	112 x 112 x 128	128	3 x 3 x 128
Conv5	65 x 65 x 128	56 x 65 x 256	256	3 x 3 x 128
Conv6	65 x 65 x 256	56 x 65 x 256	256	3 x 3 x 256
Conv7	65 x 65 x 256	56 x 65 x 256	256	3 x 3 x 256
Conv8	28 x 28 x 256	28 x 28 x 512	512	3 x 3 x 256
Conv9	28 x 28 x 512	28 x 28 x 512	512	3 x 3 x 512
Conv10	28 x 28 x 512	28 x 28 x 512	512	3 x 3 x 512
Conv11	14 x 14 x 512	14 x 14 x 512	512	3 x 3 x 512
Conv12	14 x 14 x 512	14 x 14 x 512	512	3 x 3 x 512
Conv13	14 x 14 x 512	14 x 14 x 512	512	3 x 3 x 512

Despite the differences in parameters of convolution layers, they could be implemented using same building blocks and a control unite as indicated in figure (24).

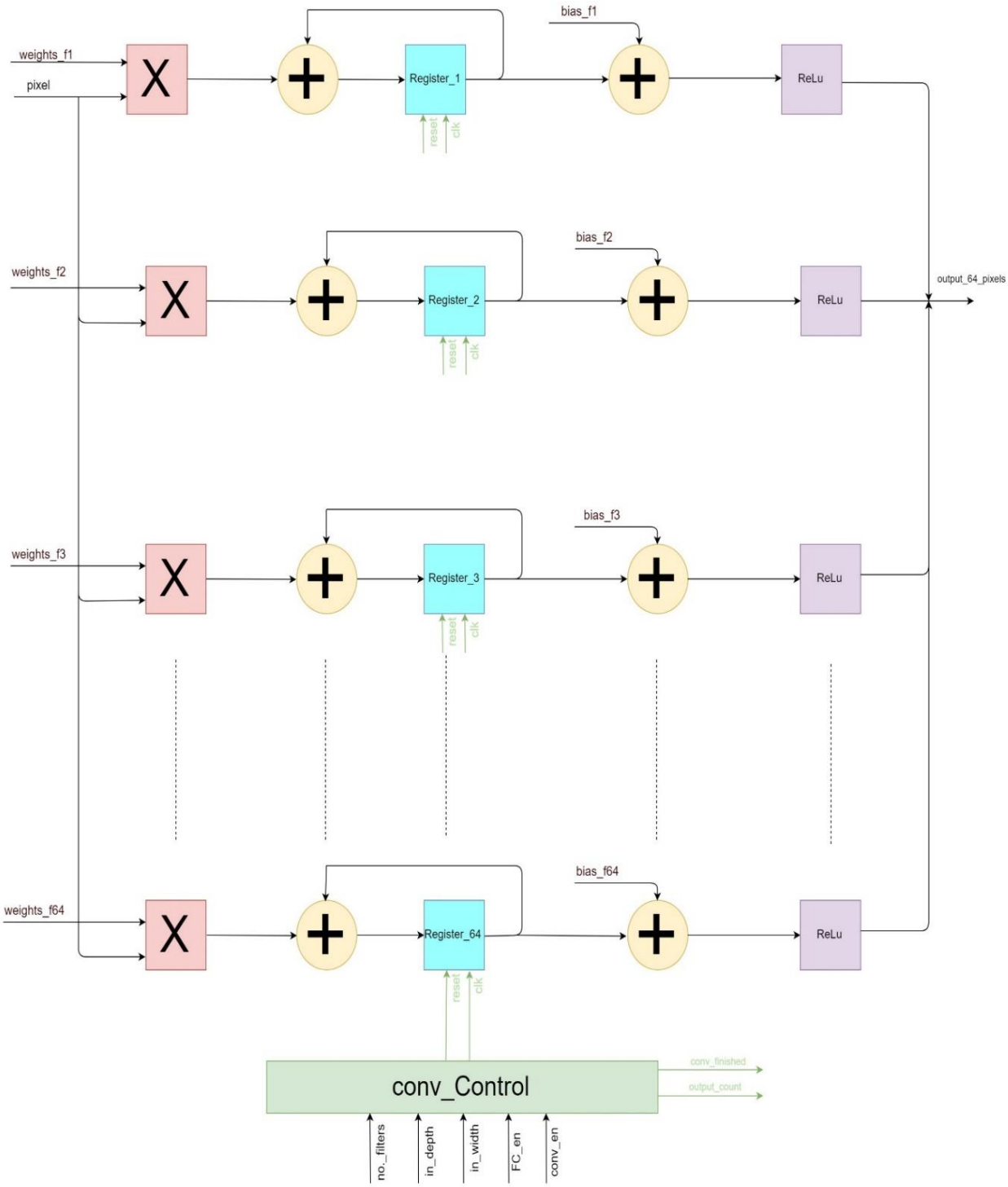

Figure 24: architecture of convolution+ReLu layer. all green signals are outputs of the conv_control unit

4.3.1 Convolution Building Block

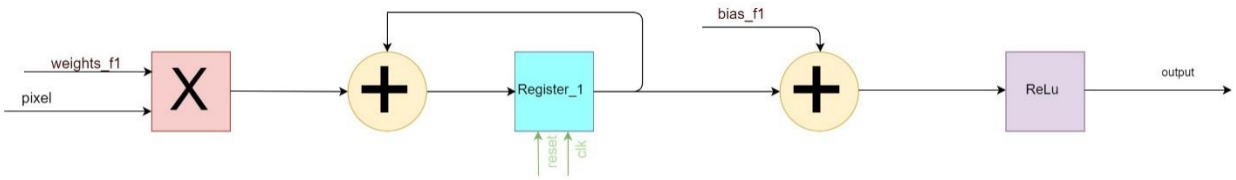

Figure 25 convolution building lock

The building block consists of a multiplier and accumulative adder. Together they perform the process of multiplying each pixel with its corresponding weight and add all the multiplication results together. When all the multiplications are finished, the accumulated result is added to the bias and get passed to the ReLU block which pass the positive results and pass zero for any negative result. During the accumulation, the reset signal is set to low and is set to high once the accumulation is finished to get the block ready for the next window. This building block calculates the output of one filter window convolution and is swept over the input feature map to produce one layer of the output feature map.

4.3.2 Reuse convolution architecture for FC

As noticed from sections (2.4.1 and 2.4.4.) the operation of of fully connected layers is similar to that of the convolution. The first FC layer for example, We can think of it as a convolution layer with 4096 filters and kernel window of 7x7. For this reason, In the proposed design we used the same hardware for both the convolution and FC layers which saved huge amount of area and power.

In addition, Parallelism and pipelining techniques are applied to the hardware architecture to speed up the CNN.

4.3.3 Convolution Control unit

The conv_en and FC_en signals control the operation of the architecture as indicated in the table 7.

Table 7. Enable signals values corresponding to which layer of FC or convolution layer is to be performed

Conv_en	FC_en	function
0	00	Layer disabled (reset = 1)
0	01	First FC layer
0	10	Second FC layer
0	11	Third FC layer
1	xx	Convolution layer

When the architecture operates as a convolution layer. The conv_control block takes as input the input FM depth to control the reset signal of the register after accumulation of one window and adds one to the output count. It takes also the width of input FM and number of filters to be calculated to calculate when the layer will be finished and output the conv_finished signal to enable the next layer.

when it operates as FC layer the conv_control calculate the accumulate and repeat conditions according to which layer of FC is to operate.

4.3.4 Convolution layer architecture

In the architecture of the Convolution layer in figure (24), we used 64 parallel units of the building block to reuse the feature map input pixels for 64 filters at the same time. Together, they correspond to 64 filter convolutions. Their number was chosen as it is the greatest common divisor of all convolution layers filters number. As a result, one hardware architecture is used for all the convolution layers but with different accumulate and repeat conditions.

The number of building blocks could be increased to speed up large convolution layers, but this of course will add more area that is unused when it operates for small convolution layers with smaller number of filters.

4.4. Max Pooling layer

As illustrated before, Pooling layers are used to down convert the feature map 2-D size without major loss of the features used for classification. In the case of VGG network, the pooling is done by taking the maximum between each 4 inputs as illustrated in figure (12).

The max-pool building block takes the four pixels' window in a serial way. So that, it consists of comparator that compare two words and output the maximum of the, with the output of the comparator connected to one of its inputs as in figure (26). In, this way, after four clock cycles the value stored in the register is equal to the maximum of the input window. Reset signal is used to reset the register value to zero before each window processing.

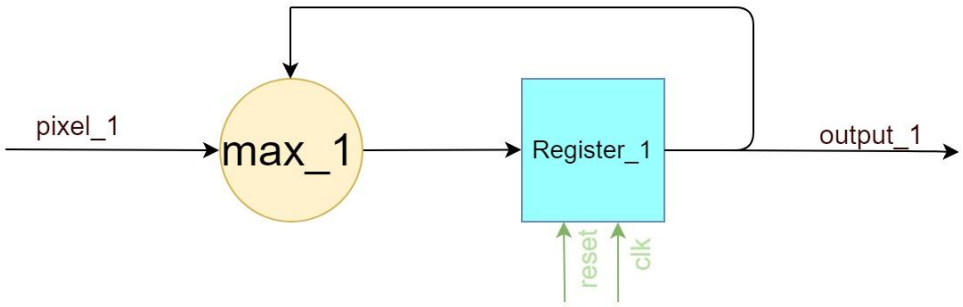

Figure 26: Max-Pool Building Block

4.4.1. Max-pool Parallelism

In order to speed up the process which is our target of this design. Also, in order to match the output bus with that of the convolution layer to be able to store them in same memory, 64 parallel max-pool blocks were used as shown in figure(27).

The max-pool layer also output an output_count signal to allow the memory control to increment the address of the memories for storing the next outputs. In addition to finished_signal which enable the next layer of the network. The Max-Pool control block output these two signals using input information about the layer input FM dimensions.

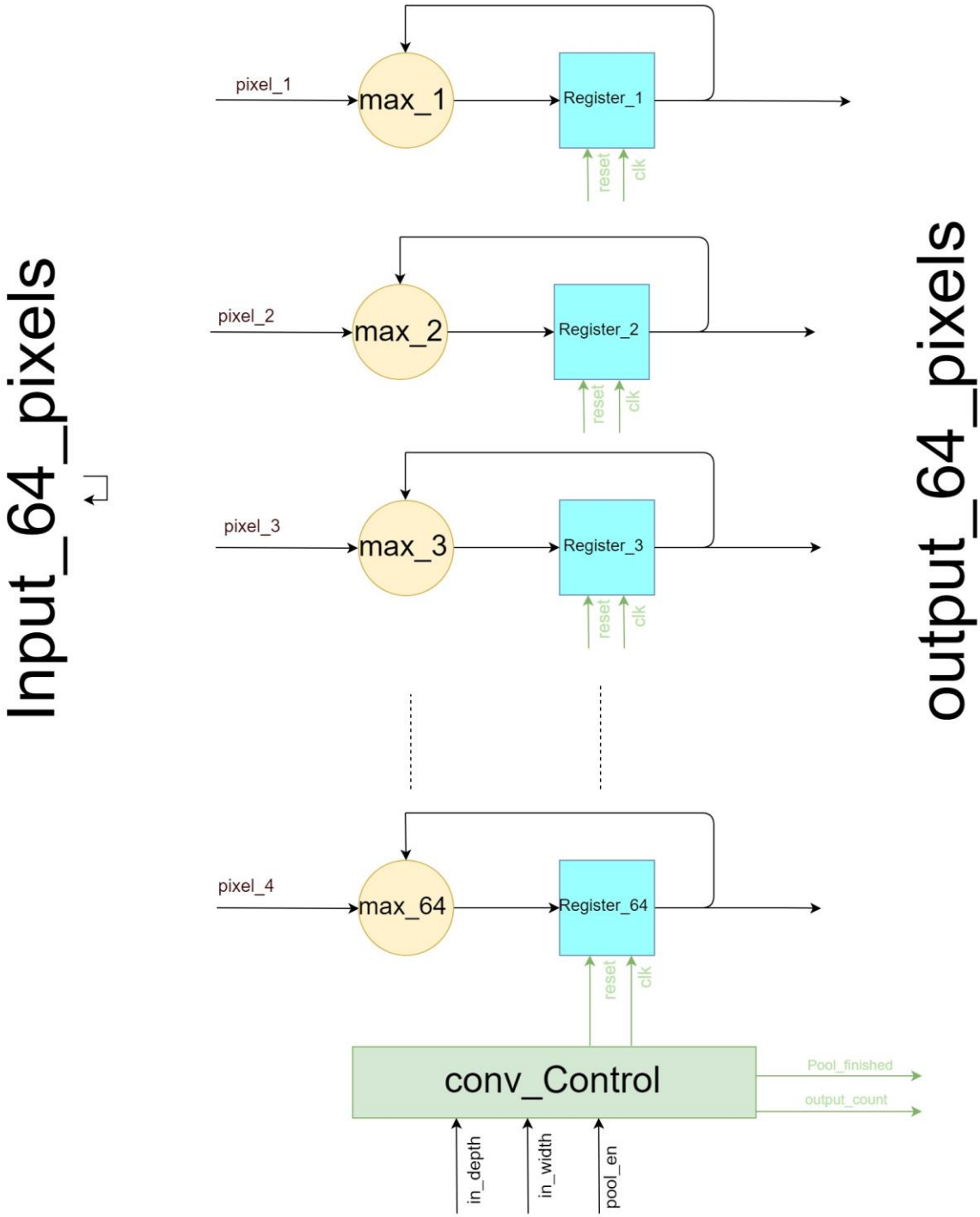

Figure 27: Max-Pool architecture

4.5. Soft-max Layer

Soft-max function is simply consists of three main mathematical operations which are exponential, addition and division. The hardware implementation of those mathematical operations is not as simple as their mathematical meaning especially for both exponential and division functions. Moreover, in order to achieve those functions, valid approximations and simpler functions are used to get a high accuracy output with low power consumption and high speed. The hardware implementation of those functions is briefly explained in the following sections.

4.5.1 Exponential implementation

A low power hardware implementation of exponential is proposed by Shaik [16] using Taylor Series expansion. Taylor series represents any function by an infinite addition of diffrentiated terms of the function at any given value. Equation (2) shows the representation of any real function $f(x)$ at an arbitrary value $x = a$.

$$f(\boldsymbol{x}) = f(a) + f'(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^n(a)}{n!}(x-a)^n \qquad (2)$$

Substituting $f(x) = e^x$ in Equation (2) leads to Equation (23

$$e^x = e^a[1 + (x-a) + \frac{1}{2}(x-a)^2 + \frac{1}{6}(x-a)^3 + \frac{1}{24}(x-a)^4 + \frac{1}{120}(x-a)^5 + \frac{1}{720}(x-a)^6] \qquad (3)$$

Taking $(x-a)$ as a common factor and rearrange will result Equation (4)

$$e^x = e^a\left[1 + (x-a)\left(1 + (x-a)\left(\frac{1}{2} + (x-a)\left(\frac{1}{6} + (x-a)\left(\frac{1}{24} + (x-a)\left(\frac{1}{120} + \frac{1}{720}(x-a)\right)\right)\right)\right)\right)\right] \qquad (4)$$

Using many reduction steps, we got Equation (5)

$$e^x = e^a(c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5) \qquad (5)$$

Where the coefficient values are calculated in table 8.

Table 8. Coefficient values as function of the constant a

Constant	Coefficient
C0	$1 - a + \frac{a^2}{2} - \frac{a^3}{6} + \frac{a^4}{24} - \frac{a^5}{120} + \frac{a^6}{720}$
C1	$1 - \frac{2a}{2} + \frac{3a^2}{6} - \frac{4a^3}{24} + \frac{31a^4}{720} - \frac{6a^5}{720}$
C2	$\frac{1}{2} - \frac{3a}{6} + \frac{6a^2}{24} - \frac{64a^3}{720} + \frac{14a^4}{720}$
C3	$\frac{1}{6} - \frac{4a}{24} + \frac{66a^2}{720} - \frac{16a^3}{720}$
C4	$\frac{1}{24} - \frac{34a}{720} + \frac{9a^2}{720}$
C5	$\frac{7}{720} - \frac{2a}{720}$

Equation (5) can be simplified into Equation (6)

$$e^x = e^a(c_0 + x(c_1 + x(c_2 + x(c_3 + x(c_4 + c_5x))))) \qquad (6)$$

As noticed from Equation (6)we can implement the exponent by only 6 multipliers and 5 adders as illustrated in the architecture in figure (28)

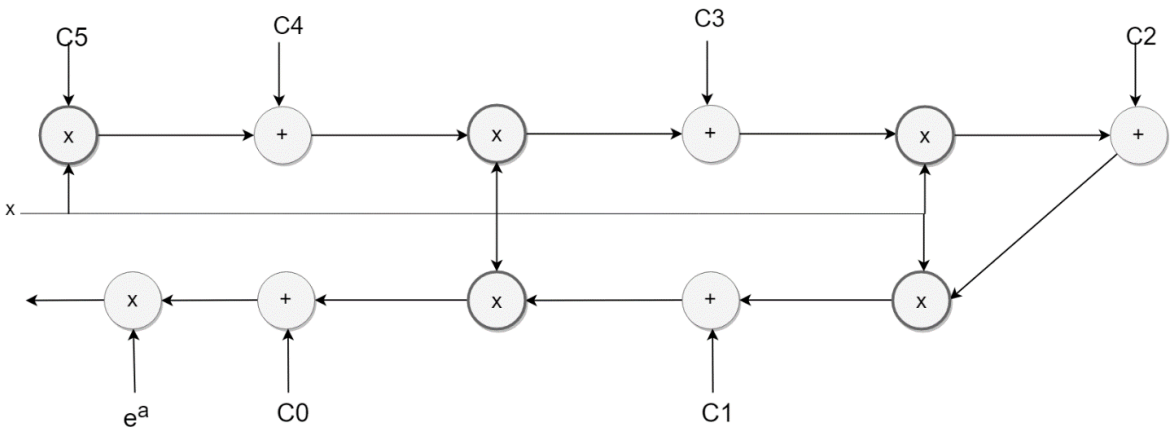

Figure 28: Architecture of The Exponential module using Taylor Series

4.5.2. Fixed-Point Divider

The basic idea of fixed-point divider is that the division is occurred using shifters, comparators, counters, and subtractors. The floating-point input dividend and divisor consist of three parts which are the sign bit, integer bits, and the floating bits. In our case, the sign bit is always positive because of the previous RELU layer is vanishing any negative numbers. The integer and floating bits of the quotient is calculated through the module in the following steps. First, the inputs are stored in registers with larger number of bits, consequently, an enlargement in the value of the inputs occurred, but the divisor has a bigger enlargement because it is stored in a larger register. Second, the comparator takes the previous values and checks if the dividend is bigger than or equal the divisor, and if so, the counter determines the index of the bit of the quotient that will be assigned to 1, Besides, the divisor will be subtracted from dividend and overwritten in the dividend [18]

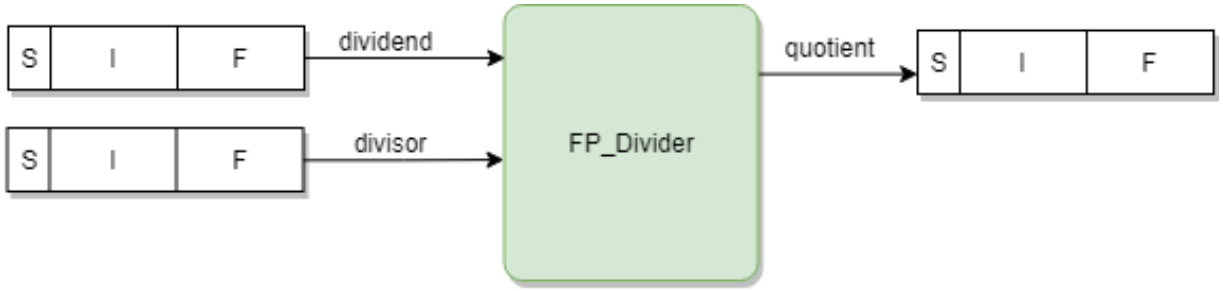

Figure 29: divider module

2.5.3. Soft-Max top architecture

Finally, the Soft-max architecture as in figure (30) contains data path and controller. The data path consists of four blocks; exponential block (EXP), memory block (MEM), fixed point adder block (adder), and the division block (DIV). The exponential module takes one input at a time and pass its output to be stored in predefined address in the memory, Moreover, the exponential output is added to all other exponent outputs by the adder in order to obtain the denominator of the Soft-max function. Thereafter, the divider takes each element in the memory and divides it on the summation of all exponentials that resulted from the adder. The controller module is working in two consecutive states. The first state disables the divider module and the reading from memory until all the inputs are went through the exponential, and EXP outputs are completely written in the specified address at MEM besides getting the sum of those outputs from adder. The second state disables EXP, writing on MEM, and adder in order to save power and keep the values in MEM and adder as they are. Furthermore, second state enables DIV until it calculates all probability distributions of the original inputs.

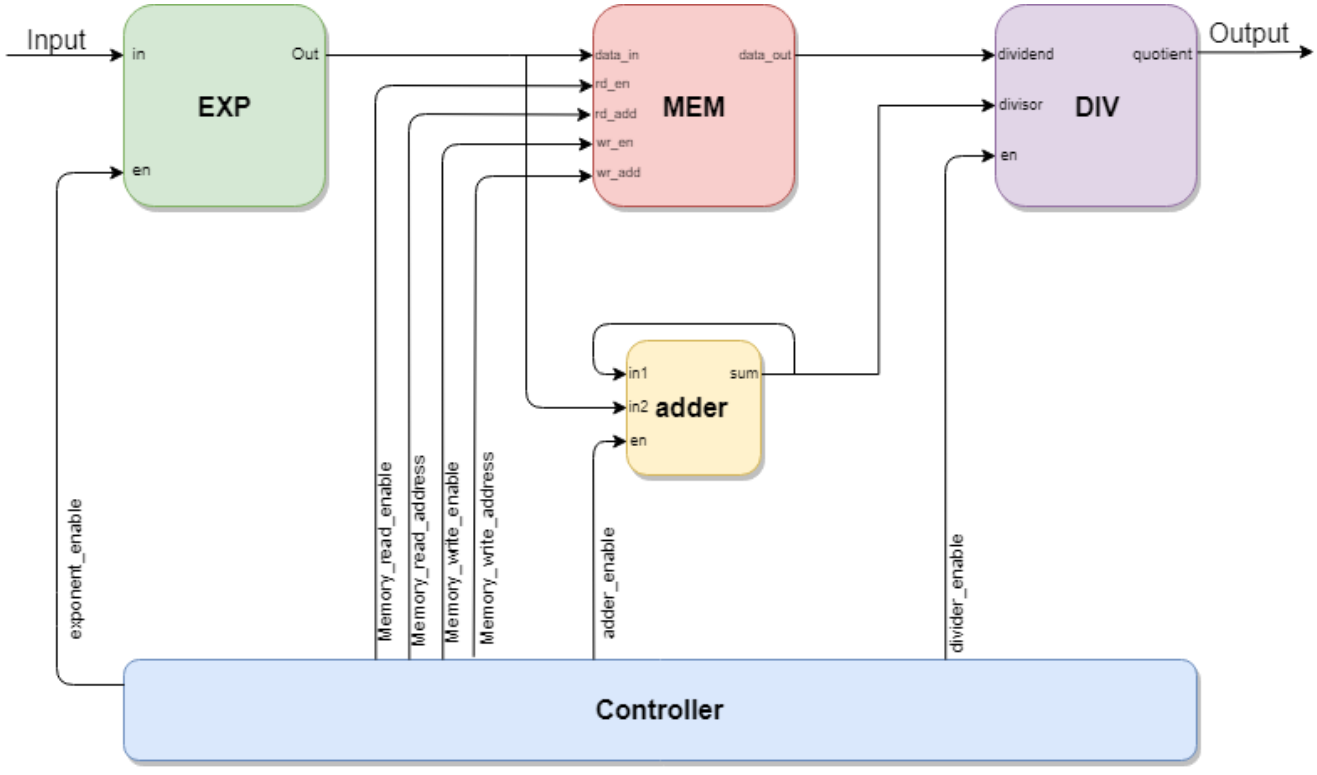

Figure 30: Soft-Max top architecture

4.6. Memory architecture

Neural networks require very large memory storage capacity and especially the fully-connected layers are considered the most exhaustive in terms of memory use due to the large number of weights per filter. For the feature map memory, the largest feature map produced during the entire VGG operation is the product of the first convolution layer, giving 224x 224 x 64 words, 17 bits each. Therefore, 2 memories of this size are required, one to be read from and the other to be written to simultaneously.

In this paper, Virtex UltraScale + FPGA is targeted for the design implementation. Therefore, there are 3 available on-chip memory types; distributed memory, Block RAM (BRAM) and Ultra RAM (URAM). Distributed memories are LUT-based, which makes them not in optimum design for data storage. Therefore, they are mostly designated to store relatively smaller data sizes. Here, BRAMs and URAMs are mainly used to store the feature maps and filter weights.

4.6.1. Memory Types

There are two main types of FPGA memory used:

4.6.1.1. Block RAM

According to [12], A BRAM in Virtex UltraScale + has a memory capacity of up to 36Kbits. It can also be set up as 2 independent 18Kbits blocks or cascaded into a 64x1 block using 2 blocks. Each block can be configured as 16Kx2, 8Kx4, 4Kx9, 2Kx18, 1Kx36 or 512x72. BRAMs can be set up as synchronous/asynchronous and single/dual port, but in this design synchronous dual-port BRAMs were employed.

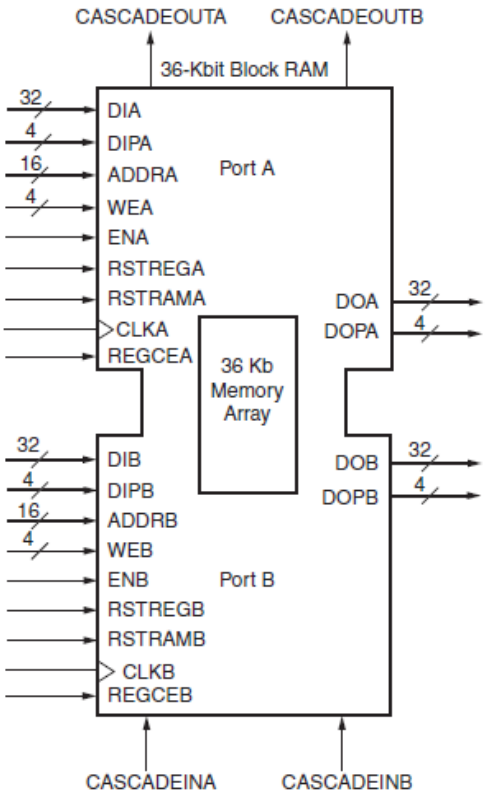

Figure 31: As represented in [12, Fig 1-1], a true-dual port design for RAM36

As shown in figure (31), a dual-port RAM takes in an input bus of 36 bits at DOA+DOPA, and outputs 36 bits on another independent port DOB+DOPB according the input addresses (ADDRA/ADDRB). Its synchronous behavior is governed by the clock (clkA/clkB) and the memory is enabled by (enA/enB), which is determined by the designer.

4.6.1.2. Ultra-RAM

As stated in [13], A URAM block is a synchronous, single clock memory with two-independent read and write ports. It has a fixed width of 72 bits and can store up to 288Kbits. A URAM has nearly 8 times the storage capacity of a BRAM and multiple URAMs can be cascaded together along a column using designated cascading paths to form a larger memory. Also, Cascades within different columns can be linked together, while consuming minimal logic and delay if pipelined efficiently.

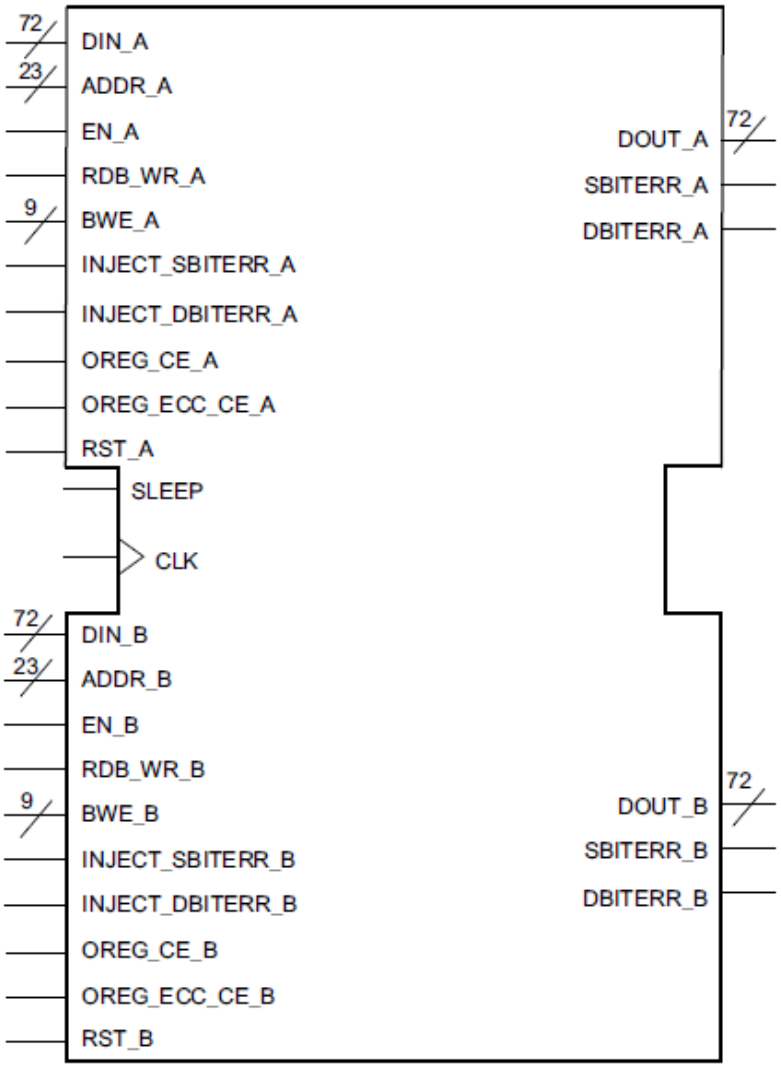

Figure 32: As represented in [13, Fig 2-1], a true-dual port design for RAM36

As seen in figure (32), Input is written into the memory from port DIN and output is read from DOUT. The memory operation enabled from EN and the addresses are recognized from ADDR ports. There is additionally an automatic power saving mode embedded in URAMs that gets activated by the SLEEP port.

4.6.1.3 Memory design

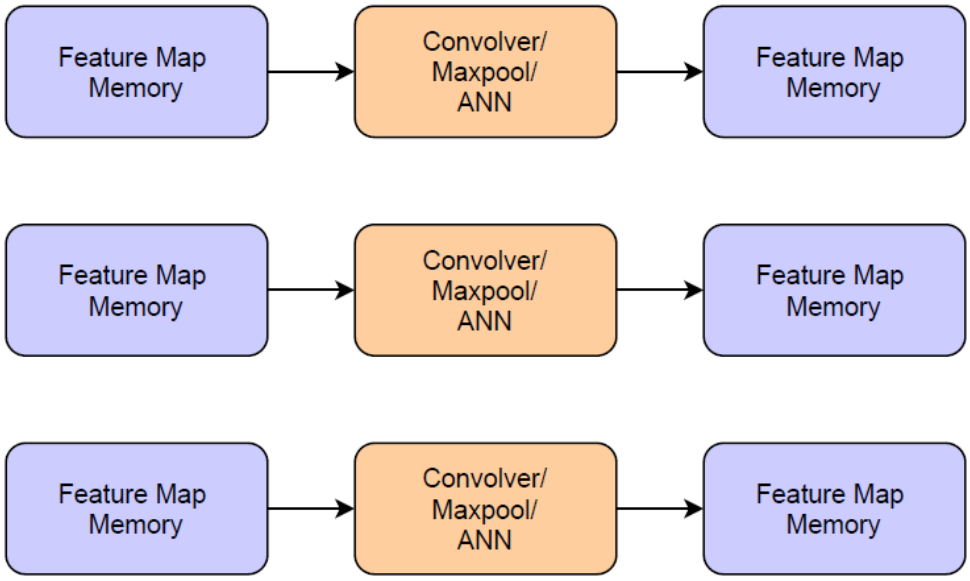

Figure 33: 64 Processing engines with their adjacent feature map memories

In this thesis, parallelism of 64 processing engines is designed in order to perform convolution and max-pooling on 64 filters simultaneously. Accordingly, each processing engine is designed to have a memory at the input and one at the output each with a storage capacity of 224x224. In total 1664 BRAMs would be needed for one feature map memory. Since, the total number of BRAMs in this version of Virtex 7 UltraScale + is 2688, it was not enough to accommodate both feature maps. The other feature map was, therefore, designed using URAMs, which required 832 blocks to build 64 memory units. The rest of the BRAMs were used to store the weights and biases of the 64 filters in use. An external memory would also be needed, as the data of one fully-connected layer alone can get as large as 7x7x512x4096.

4.6.2. Memory optimizations

4.6.2.1. Pipelining

In order to achieve the large memory requirement for VGG, each memory block was constructed from a cascade of multiple URAMs. URAMs are divided into columns and the routing paths of every cascade in contained within the column, saving area and

delays. Also, Pipelining of 3 cycles has been designed in order to enhance the performance of the cascade and reduce delays.

4.6.2.2. memory Reuse

The feature map memories are overwritten after each layer is done, which allows the operation to run mostly on-chip, preventing delays and enhancing memory access rate. Filter weights only need to be updated by the end of each layer, which does not require interruption in the middle of the process.

4.6.2.3. Parallelism

Sixty-four parallel memories are used in order to speed up the memory access rate and enhance the performance of the entire network. It is also needed to support the outputs of the processing elements, so that no processes need to be interrupted in order to save or read data.

Chapter 5.
Implementation and Results

5.1. Verification of RTL functionality

5.2. FPGA implementation Results

5.2.1. Utilization of resources

5.2.2. Power analysis

5.2.3. Timing Results

5.3. Discussion of Results

5.4. Comparative study

Chapter 6.
Conclusion and future work

6.1. Conclusion

6.2. Future work

6.2.1. Implementation on Large FPGA

6.2.2. Use of external memory

6.2.3. FC weights Pruning

6.2.4. Increasing parallelism

6.2.5. General platform for CNNs

References

[1] "ImageNet Large Scale Visual Recognition Competition (ILSVRC)", *Image-net.org*, 2017. [Online]. Available: http://www.image-net.org/challenges/LSVRC/.

[2] "COCO - Common Objects in Context", *Cocodataset.org*, 2019. [Online]. Available: http://cocodataset.org/#home.

[3] S. Liu and W. Deng, "Very deep convolutional neural network-based image classification using small training sample size", 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), 2015. Available: 10.1109/acpr.2015.7486599

[4] B. Moons, B. De Brabandere, L. Van Gool and M. Verhelst, "Energy-efficient ConvNets through approximate computing", *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016. Available: 10.1109/wacv.2016.7477614

[5] S. Albawi, T. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network", *2017 International Conference on Engineering and Technology (ICET)*, 2017. Available: 10.1109/icengtechnol.2017.8308186

[6] Justin Johnson & Serena Yeung, "Lecture 9: CNN Architectures", 2017.

[7] A. Ur Rahman Shaik, "HARDWARE IMPLEMENTATION OF THE EXPONENTIAL FUNCTION USING TAYLOR SERIES AND LINEAR INTERPOLATION", Sweden, 2014.

[8] S. Albawi, T. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network", 2017 International Conference on Engineering and Technology (ICET), 2017. Available: 10.1109/icengtechnol.2017.8308186

[9] D. Learning and A. demystified, "Architecture of Convolutional Neural Networks (CNNs) demystified", Analytics Vidhya, 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

[10] "Difference Between Softmax Function and Sigmoid Function", Dataaspirant, 2019. [Online]. Available: http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/

[11] "VGG16 - Convolutional Network for Classification and Detection", Neurohive.io, 2019. [Online]. Available: https://neurohive.io/en/popular-networks/vgg16/.

[12] “7 Series FPGAs Memory Resources Xilinx User Guide”Xilinx.com, 2011. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ug586_7Series_MIS.pdf.

[13] “UltraScale Architecture Memory Resources Xilinx User Guide” *Xilinx.com*, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascale-memory-resources.pdf.

[14] E. Abd El-sattar, R. Ahmed and S. Abd El-Wahab, "ACCELERATED DEEP NEURAL NETWORKS USING FPGA", 2018.

[15]T. Agarwal, "Basic FPGA Architecture and its Applications", *Edgefx.in*, 2019. [Online]. Available: https://www.edgefx.in/fpga-architecture-applications/. [Accessed: 09- Jun- 2019].

[16]"FPGA Design Flow Overview", *Xilinx.com*, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm#ise_c_configuration_overview_ise_c_configuration_overview139_44. [Accessed: 09- Jun- 2019].

[17] A. Shaik, Hardware Implementation of the Exponential Function Using Taylor Series and Linear Interpolation. 2014, pp. 29-30.

[18] T. Burke, "Overview :: Fixed Point Arithmetic Modules :: OpenCores", Opencores.org, 2019. [Online]. Available: http://opencores.org/project,fixed_point_arithmetic_parameterized. [Accessed: 08- Jun- 2019].