DESIGN SPACE EXPLORATION OF FPGA - BASED CNN

PROJECT REPORT

Submitted by

BL.EN.P2EBS15022 VIBHA PANT

In partial fulfillment for the award of the degree

of

MASTER OF TECHNOLOGY

IN

EMBEDDED SYSTEMS

AMRITA SCHOOL OF ENGINEERING, BENGALURU

AMRITA VISHWA VIDYAPEETHAM

BENGALURU 560 035

JULY-2017

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, BENGALURU, 560035

BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**DESIGN SPACE EXPLORATION OF FPGA - BASED CNN**" submitted by "**VIBHA PANT** (BL.EN.P2EBS15022)" in partial fulfillment of the requirement for the award of the Degree of **Master of Technology** in **"EMBEDDED SYSTEMS"** is a bonafide record of the work carried out under our guidance and supervision at Amrita School of Engineering, Bengaluru.

PROJECT GUIDE

Dr. Madhura Purnaprajna
Associate Professor
Dept. of Computer Science & Engineering
Amrita School of Engineering, Bengaluru

CHAIRPERSON

Dr. Amudha J
Professor & Chairperson
Dept. of Computer Science & Engineering
Amrita School of Engineering, Bengaluru

This project report was evaluated by us on .....

EVALUATORS

Dr. Prashanth Athri
Associate Professor
Dept. of Computer Science & Engg.
Amrita School of Engineering, Bengaluru

Dr. Rajath Vasudevamurthy
Assistant Professor
Dept. of Electronics & Communication Engg.
Amrita School of Engineering, Bengaluru

ACKNOWLEDGEMENT

I wish to acknowledge all the people who have helped me in this project. Foremost, I humbly bow before AMMA who has been driving force and a source of inspiration for all educational activities and extracurricular activities at AMRITA SCHOOL OF ENGINEERING, BENGALURU.

I would like to express my sincere gratitude to our Director Br. Dhanaraj Swamiji for the successful completion of the project. I would like to extend my sincere thanks to our Associate Dean Dr.Rakesh S.G for inspiring us.

I am highly indebted to Dr. Amudha.J, Chairperson, Department of Computer Science and Engineering and also Dr. T. S. B. Sudarshan, former Chairman, Department of Computer Science and Engineering for motivating us towards the successful completion of the project.

I express my sincere gratitude to Mihir Mody and Manu Matthew, Texas Instruments, Bengaluru for their invaluable support and guidance throughout this project.

I am deeply indebted to my project guide, Dr. Madhura Purnaprajna, Associate Professor, Amrita School of Engineering, Bengaluru for her support and guidance and also for her initiatives which have opened up new avenues for me.

I thank Dr. Prashanth Athri , Associate Professor, Dept. of Computer Science Engineering and Dr. Rajath V, Assistant Professor, Dept. of Electronics & Communication Engg. of Amrita School of Engineering, Bengaluru for their support and taking time to evaluate my work.

Lastly and most importantly, I thank my husband and our parents for their continuous love and support, without which none of this would be possible.

Vibha Pant

BL.EN.P2EBS15022

ABSTRACT

Convolution Neural Networks (CNNs) have become very popular for advanced driver assistance systems (ADAS) and autonomous driving for object detection and image recognition. The choice of CNNs for ADAS necessitates a high-throughput in the order of about few 10's of TeraMACs per second (TMACS). In addition to throughput, accuracy is also of very high importance. Existing implementations become unusable with performance ranging only in the order of a few Giga-ops. This paper presents a novel tiled architecture for CNNs with ternarized weights.

In this context, it has been observed that ternarization of weight vectors results in a minimal loss of accuracy, while reducing the memory utilization and eliminating multipliers in the inference phase. Our tiled architecture is generic and scalable across variations in network organization and device configurations. Hence, our objective is to trade-off accuracy to achieve high performance for CNNs to be applied in ADAS scenario.

Our implementation results in 13.76 TOPS throughput for TileNET implementation of AlexNet on Virtex-7 FPGA device. The results from our estimation model show that the performance can be tuned across various application needs by catering to different CNN architectures and by providing flexibility across devices. The post implementation power results for AlexNet are around 16 W for the largest layer and are much lower than GPU power consumption.

Table of Contents

List of Figures

List of Tables

Nomenclature

Acronyms / Abbreviations

ADAS Advanced Driver Assistance Systems

AI Artificial Intelligence

ANN Artificial Neural Network

BLE Basic Logic Element

BRAM Block Random Access Memory

CLB Configurable Logic Block

CNN Convolutional Neural Network

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

LUT Look Up Table

ReLU Rectified Linear Unit

RSB Row Storage Buffer

Chapter 1
INTRODUCTION

Artificial Intelligence is the branch of Computer Science which aims at making computers and machines think intelligently as humans do. One of the research areas in AI is Artificial Neural Networks. Artificial Neural Network (ANN) is an information processing system modelled after the biological neural networks. The ANN consists of multiple layers which ultimately perform a classification task as shown in Figure 1.1. The basic processing element of the ANN is a neuron which applies an activation function to the sum of the weighted inputs and passes it along via weighted connections to other layers or neurons as shown in Figure 1.2.

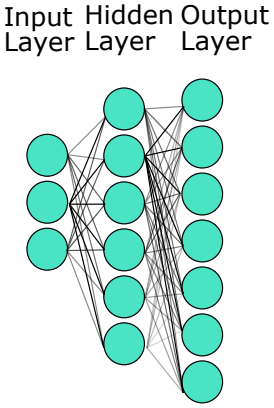


Figure 1.1: Artificial Neural Network

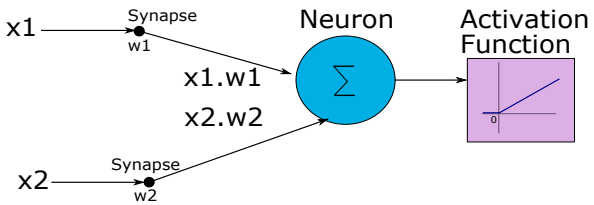


Figure 1.2: Neuron

1.1 Convolution Neural Network

Convolution Neural Networks (CNN) are very like ANNs and are made up of nodes with learnable weights and biases and with the nodes performing dot product of the inputs they receive and like the ANNs are mainly used for problems involving classification. The architecture of a CNN is designed to take advantage of the 3D structure of an input image as shown in Figure 1.3 (or other 3D input such as a speech signal).CNNs therefore can be considered a special case of ANNs dealing exclusively with input as images. In CNNs, the input is arranged as a 3-D volume with height, depth and width. Here, each layer is connected to the next through a window or a receptive field unlike in ANN where each layer is fully connected to all the neurons of the next one.

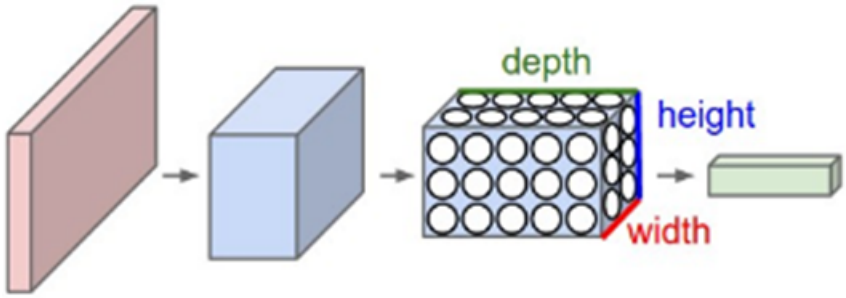


Figure 1.3: Convolution Neural Network[14]

1.1.1 CNN Architecture

CNNs are deep neural networks which consist of layers on layers of different types.The main types of layers in a CNN are:

- Convolution Layer
- Pooling Layer
- ReLu Layer
- Fully Connected Layer

A CNN architecture consists of a number of convolution and sub sampling layers optionally followed by fully connected layers. The input to a convolution layer is a $NxR_{in}xC_{in}$ image where R_{in} is the height, C_{in} width of the image and N is the number of channels, e.g. an RGB image has N=3. The convolution layer will have KxK filters (or weight vectors or

kernels) of size $M \times N \times K \times K$ where K is smaller than the dimension of the image and M is the number of output channels. The size of the weight vectors gives rise to a receptive field or a locally connected structure which are each convolved with the image to produce output of Dimension $M \times R_{out} \times C_{out}$.

Each output feature map is then sub sampled typically with mean or max pooling over $d \times d$ contiguous regions where d ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the sub sampling layer an additive bias and sigmoidal non-linearity is applied to each feature map. Traditionally,the sigmoid or hyperbolic tangent function is used for the non linearity introduction. However, these classic non-linear functions have several disadvantages. First, their gradient becomes very small for large values, which means the error gradient during training vanishes. Moreover, these two non-linear functions are relatively costly in terms of computing power.As an alternative, nearly all state of the art CNN use Rectified Linear Units(ReLU). The work by Krizhevsky et al [15] was the first to apply this simplified activation to deep neural networks.Deep networks trained with this activation function converge much faster.The function of a ReLU layer maps negative values to zero: $f(x) = max(0,x)$.

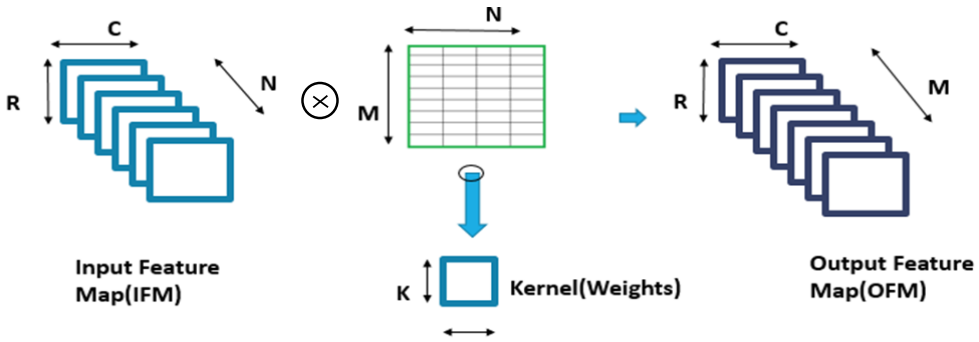


Figure 1.4: Convolution Layer

The layers are stacked in various combinations to form the CNN architecture (E.g. LeNet, AlexNet). The basic functionality of the CNN is to classify the input image to one the classes determined during training phase. The convolution layer is the core building block of the CNN. As shown in Figure 1.4 the convolution layer accepts N depth of input images of size $R_{in} \times C_{in}$ and performed a convolution with $M \times N$ weights of size $K \times K$ to give M depth of output images of dimension $R_{out} \times C_{out}$. In essence, the kernel or the filters is moved across the input feature maps to perform multiply and accumulate operations and get the Output feature map. When the hyper parameter of Stride is considered, which is how much the kernel can skip while moving across the input volume, then if the output feature maps are to have the same dimensions as that of $R_{in} \times C_{in}$ then zeros have to be padded

by $(kernel - 1/2)$. Convolution Layers are the feature extractors as they look at the input image through receptive fields and identify the features. As the image passes through each convolution layer, certain features are extracted.The convolution layer is a feature extractor where the filters or weight vectors are convolved with the Input images to extract the low level features like edges and curves. As the input passes through multiple convolution layers more complex features are detected.

Fully connected layers, as the name implies as fully connected to the next layer. The neurons are connected to each and every neuron of the next layer. This means we have a proportional increase in the number of weights or kernels that are required. Convolution layer is the most computationally intensive of all the layers while the fully connected layer is the most memory intensive. The main aim of the research in CNNs is to exploit parallelism to reduce the memory and computational load to improve the performance in terms of accuracy, power, area, throughput and cost. In this thesis, we look at designing and implementing a CNN on an FPGA and analyzing and improving its performance. FPGAs are inherently parallel and come close to mimicking the cortex of the brain in regard to the parallelism. FPGAs are flexible and allow quick prototyping as well.

1.2 Deep CNNs

There are different ways in which the layers can be arranged in a deep CNN. Researchers are continually exploring these deep networks to come up with better state of the art networks which can revolutionize the way image classification works. ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) is an annual challenge in which researchers from across the world compete to see who has the best computer vision model for tasks such as classification, localization, detection, and more. Some of the more popular CNN architectures are elaborated in the following sections.

1.2.1 LeNet

The LeNet architecture was first introduced by LeCun et al. [17] and is still considered the pioneer in Deep Neural Network Architectures. LeNet was used primarily for OCR and character recognition in documents.LeNet works on the MNIST dataset.The MNIST dataset is arguably the most well-studied, most understood dataset in the computer vision and machine learning literature. An example of an image in MNIST dataset is shown in Figure 1.5.

Figure 1.5: MNIST Dataset

The LeNet architecture is straightforward and small as shown in Figure 1.6 . LeNet can be considered small in terms of memory footprint.The goal of LeNet to classify the handwritten digits from 0-9. There are a total of 70,000 images, with normally 60,000 images used for training and 10,000 used for evaluation.Each digit is represented as a 28 x 28 gray scale image (examples from the MNIST dataset can be seen in the figure 1.5). The gray scale pixel intensities are unsigned integers, with the values of the pixels falling in the range [0, 255]. All digits are placed on a black background with a light foreground (i.e., the digit itself) being white and various shades of gray.

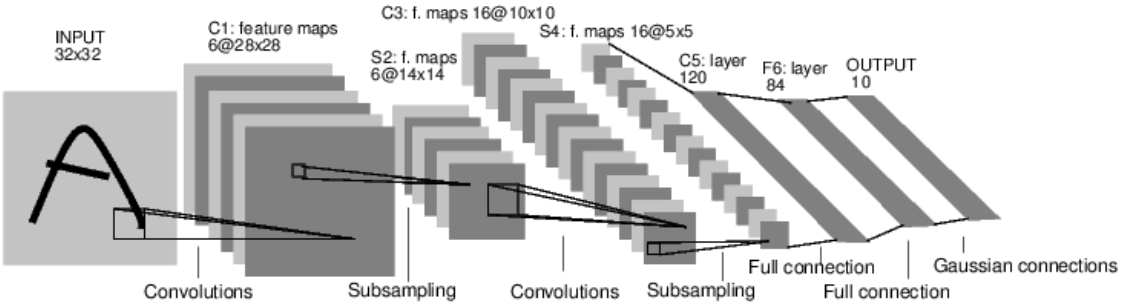


Figure 1.6: LeNet Architecture[17]

The key component is the convolution layer in the CNN. These layers exploit the structure of the data which is an image. A convolution layer connects each output to only a few close inputs, as shown in the illustration above. Intuitively, this means the layer will learn local features. The pooling layer then combines nearby inputs. This model has 91190 learnable weights.

1.2.2 AlexNet

AlexNet [15] is widely regarded as one of the most influential publications in the field of DNNs. Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton created a “large, deep convolutional neural network” that was used to win the 2012 ILSVRC.Figure 1.7 shows the network that was used by AlexNet. It is made up of 5 convolution layers, max-pooling

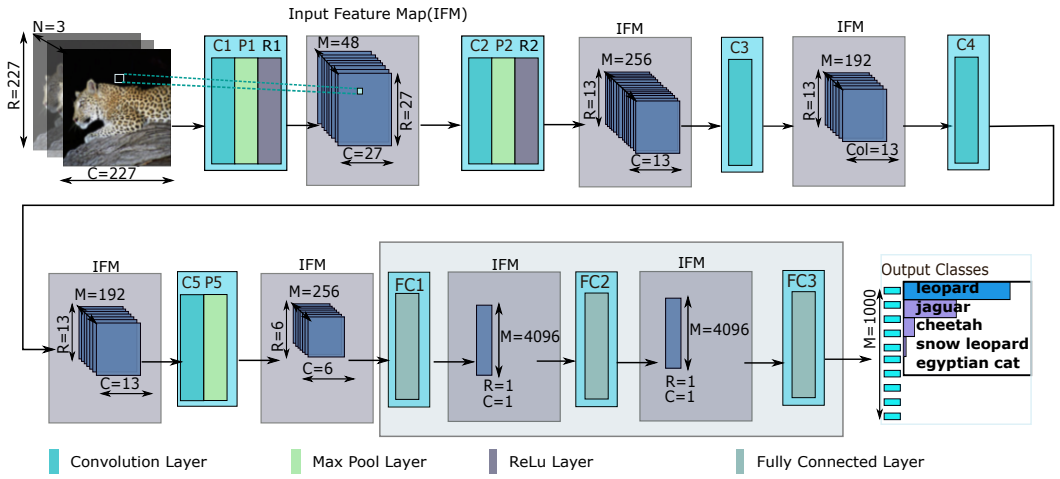


Figure 1.7: AlexNet Architecture

layers and 3 fully connected layers. The network designed was used for classification with 1000 possible categories. The dataset used for training is a subset of ImageNet is shown in Figure 1.8 which contains 15 million with over 22,000 categories.

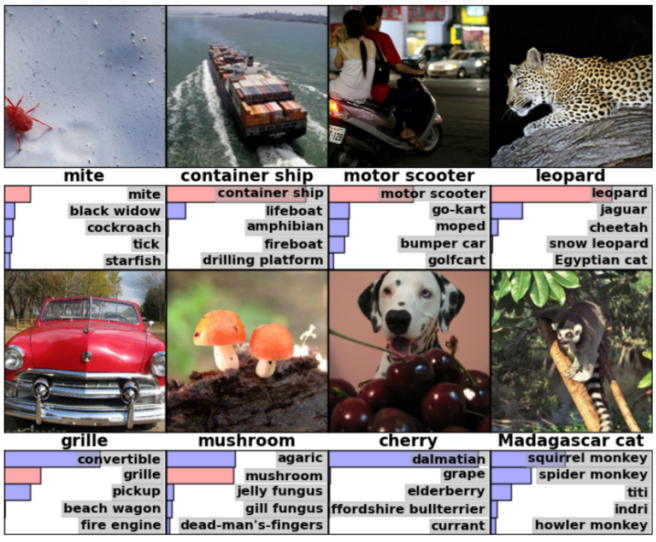


Figure 1.8: ImageNet Dataset Example[15]

1.2.3 VGG-Net

VGG-Net [27] is a 19 layer CNN that strictly uses 3x3 filters with stride and pad of 1, along with 2x2 max pooling layers with stride 2.The authors' reasoning is that the combination of two 3x3 convolution layers has an effective receptive field of 5x5. This in turn simulates a larger filter while keeping the benefits of smaller filter sizes. One of the benefits is a

decrease in the number of parameters VGG-Net is one of the most influential papers because it reinforced the notion that convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work. It is a deep but simple network architecture. The architecture is elaborated in Figure 1.9.

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					

Figure 1.9: VGG-Net Architecture[27]

1.2.4 Microsoft ResNet

ResNet is a relatively new 152 layer very deep network architecture that set new records in classification, detection, and localization through one incredible architecture. Aside from the new record in terms of number of layers, ResNet won ILSVRC 2015 with an error rate of 3.6%[12].The idea behind a residual block is that the input x is sent through Conv-ReLU-Conv layers. This will produces $F(x)$. That result is then added to the original input x. $H(x) = F(x) + x$. In traditional CNNs, $H(x)$ would just be equal to $F(x)$.Instead of just computing that transformation (straight from x to $F(x)$), the authors compute the term that has to be added, $F(x)$, to the input x. Basically, the mini residual module computes a "delta" or a slight change to the original input x to get a slightly altered representation. The authors believe that "it is easier to optimize the residual mapping than to optimize the original, un-referenced mapping". It is an Ultra-deep Network with up to 152 layers as shown in Figure 1.10.

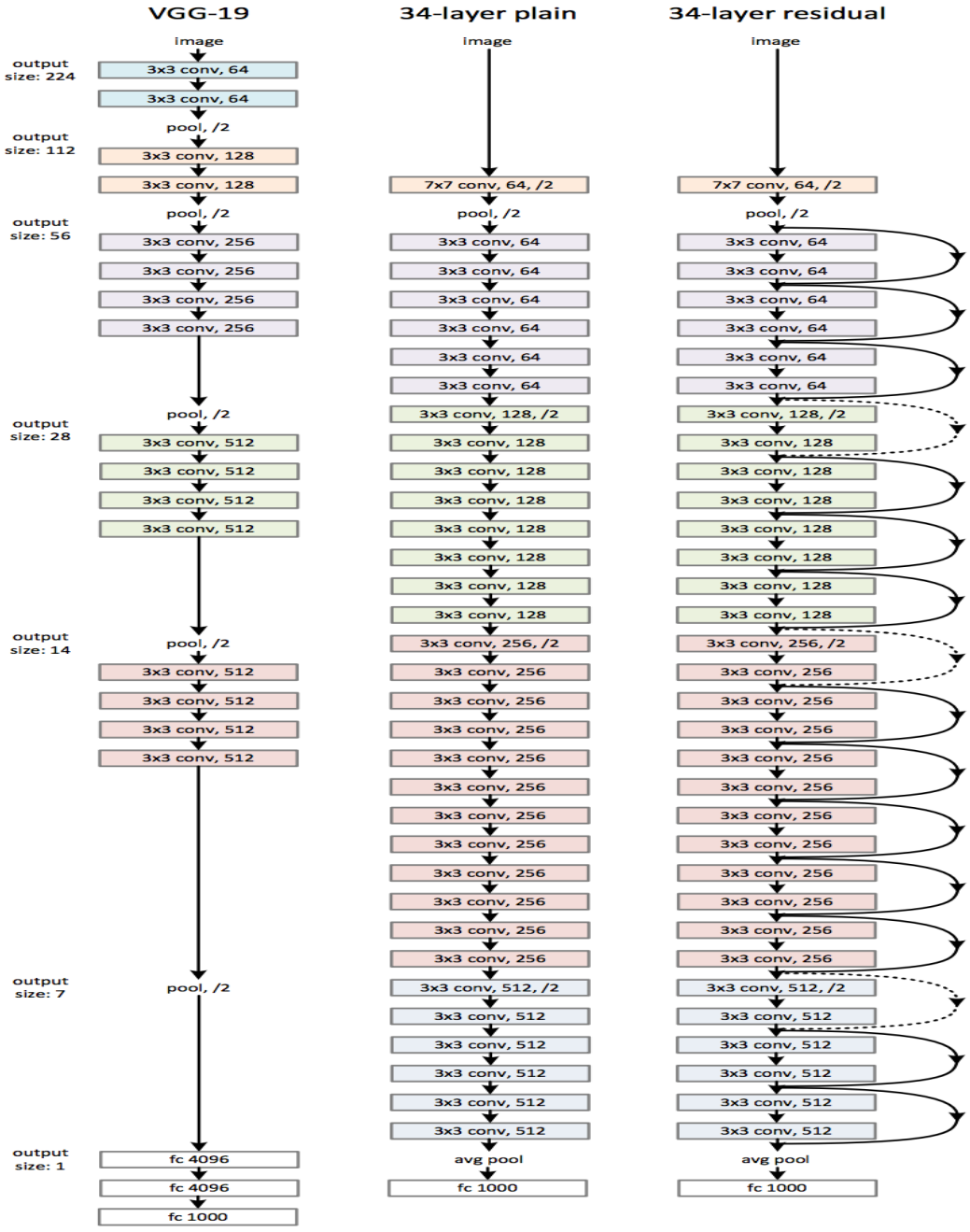
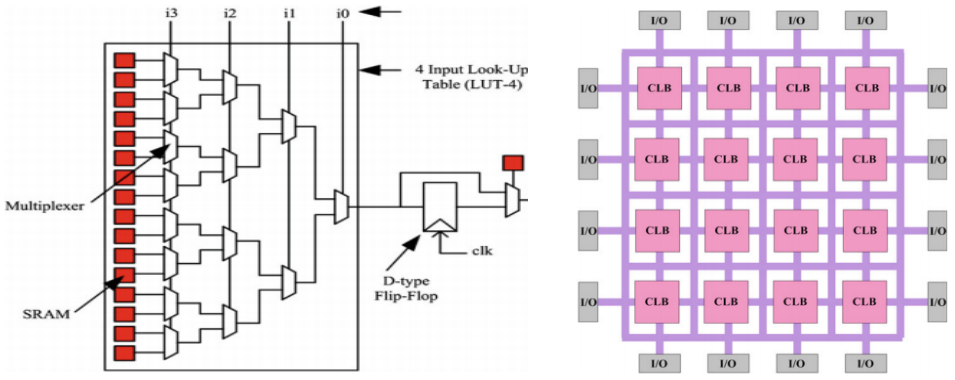


Figure 1.10: ResNet vs VGG-Net Architecture [12]

1.3 FPGA

FPGAs are inherently parallel devices which make them an ideal fit for CNN computations. The architecture of a typical FPGA device is shown in Figure 1.11b. The Configurable Logic Block(CLB) is a cluster of a few Basic Logic Elements(BLE).A simple BLE consists of a LUT and a Flip-Flop as shown in A LUT with k inputs (LUT-k) contains 2k configuration bits and it can implement any k-input boolean function. FPGA devices have resources in terms of memory BRAMs and computation in terms of LUTs. Additionally in heterogeneous FPGAs, there are hard blocks for implementing specific functions. For example, DSP blocks are often used for dedicated logic optimized for large (18×18 bits) floating point multiply or multiply add operators. [9] . In a ternary scenario, the MAC can be fully accommodated onto the LUTs which are faster and less expensive in terms of area. GPUs are known to do well on data parallel computation that exhibits regular parallelism and demands high floating point compute throughput. Across generations, GPUs offer increased FLOP/s, by incorporating more floating-point units, on-chip RAMs, and higher memory bandwidth. However, GPUs are extremely power hungry and expensive. This brings down performance per Watt.FPGAs have provided superior energy efficiency (Performance/Watt) than GPUs for DNNs. The main challenge is to provide high throughput in terms of TOPS, which we seek to address through our thesis.



(a) Basic Logic Element [9] **(b) FPGA Overview Architecture [9]**

Figure 1.11: FPGA

Chapter 2
LITERATURE SURVEY

2.1 Related Work

Convolution layers are the most computationally intensive of all the layers in the Convolutional Neural Network(CNN).[1][8],[24]. The basic operation of the convolution layer is Multiply Accumulate(MAC). Any neuron on the convolution layer receives the weighted inputs from the previous layer through a filter or a receptive field. The neuron performs the summation of all the weighted inputs and forwards it to the next layer. The cost of MACs depends on the precision on the inputs and the kernels. We need to reduce the memory size and accesses and replace most floating point arithmetic with bit-wise OR low precision operations which improves power efficiency. Computational optimization can be done by the following methods which make the training and inference in deep neural networks more efficient:

- Compression after training: Pruning redundant, non-informative weights in a previously trained network reduces the size of the network at inference time : (e.g. RISTRETTO)
- Quantizing parameters : High precision parameters quantized to lower precision to reduce usage of floating point units(e.g. XNORNETS, Binary Connect, Binary Neural Network, ternary networks)

The papers on Binary connect and Binary neural networks and XNOR nets, binarize weights. The authors in BinaryConnect [5] constraints the weights to +1 or -1 during propagations. The binarization can be done by a deterministic approach or by using a stochastic approach. The deterministic approach directly converts the real valued weight using a sign function as shown below

$$w_b = \left\{ \begin{array}{c} \text{+1 if w>=0} \\ \text{-1 otherwise} \end{array} \right\} \tag{2.1}$$

. The stochastic approach to binarization allows finer and more accurate averaging and is given by

$$w_b = \left\{ \begin{array}{c} +1 \text{ with probability p=}\sigma(w) \\ -1 \text{ with probability 1-p} \end{array} \right\} \tag{2.2}$$

where $\sigma$is a hard sigmoid function. The authors use the binary weights for forward and backward propagations but not during parameter update. The real valued weights are used during the update phase with the Stochastic Gradient Descent(SGD) during training. SGD updates the weights through infinitesimal changes during training phase to fine tune the network. The weights are represented by one bit as -1, 1. This leads to immense savings in terms of memory as weights form a large chunk of the parameters in the CNN. Also, by changing the weights to +1, -1 the MAC operation can be eliminated and approximated as just addition and subtraction. The paper on XNORnet [25] approximates this operation further by a XNOR operation and looks at the performance in this regard. The authors look at using binary inputs and weights in convolution and fully connected layers.The papers differ in regard to the method of Binarization. Binarized Neural Networks and Binary connect prefer deterministic Binarization where the weights are binarized by the sign of the weights after training (hard Sigmoid). Whereas, Binary Weighted networks and XNOR nets, prefer to estimate an optimum scaling factor which can closely approximate the weights.

The Ternarization scheme approximates the weights as +1, 0 and -1 [18].The Ternarization of the weights is performed by finding the optimal scaling factor by optimizing the L2 distance between the full precision weights. The approximated optimal solution of the Ternarization is to find an optimal threshold. The ternary valued weights are used during the forward and backward pass while keeping the full precision weights during update phase. In TWNs, 2-bit storage requirement is needed for a unit of weight. Thus, TWNs achieve up to 16× or 32× model compression rate compared with the float (32-bit) or double (64-bit) precision counterparts.Their ternary Weighted network has reported better accuracy results on MNIST Dataset as shown the following Table 2.1.

CNN	Weight	Activation	Scheme	Accuracy(MNIST)
Binary Connect[7]	-1,1	Full Precision	Deterministic Binarization	98.82%
Ternary WN[1]	-1,0,1	Full Precision	Optimization with threshold-based Ternarization function	99.35%
Planned Ternary Weight Network	-1,0,1	8 bit	Optimization with threshold-based Ternarization function	To be Analyzed

Figure 2.1: Comparison: Binary vs Ternary Neural Networks

Table 2.1: Comparison of existing schemes of CNN Acceleration

Research	Technique	Performance					Platform
FINN, Feb,2017[29]	Weightsand Activation function are binarized		MNIST	CIFAR	SVHN	AlexNet	Zynq family, Xilinx.
		Accuracy	95.8%	80.1%	94.9%	-	
		Latency	12.3M/,0.31 us	21906/ 283 us	21906/ 283 us	-	
		Power	11.3W	3.6W	3.6W	-	
		GOPS	11600	2465	2465	-	
Ternary Neural Networks for Resource-Efficient AI Applications[2]	Weights and Inputs are ternarized		MNIST	CIFAR	SVHN	AlexNet	Kintex 7,,Xilinx
		Accuracy	96.58	-	-	-	
		Latency	195K images/s20.5us	-	-	-	
		Power	3.8W	-	-	-	
		GOPS	-	-	-	-	
A high performanceCNN accelerator[19]	16 bit weights,input and intermediate results		MNIST	CIFAR	SVHN	AlexNet	Virtex 7, Xilinx
		Accuracy	-	-	-	-	
		Latency	-	-	-	0.39K images/s	
		Power	-	-	-	30.2W	
		GOPS	-	-	-	565.94	

In FINN [29], the authors implement a Binary neural network with binary inputs and weights on a Xilinx Zynq Ultrascale FPGA and report a 66 TOPS computation upper bound for the binary computations.The authors have adopted a heterogeneous architecture with multiple compute engines where the input, weights and output activation have been fully binarized. The authors also report the accuracy and throughput of FINN on MNIST, CIFAR and cropped SVN datasets. They have implemented using Vivado HLS tool and they report a maximum throughput is about 2 TOPS.

Ternary AI [2] proposes a student-teacher approach for Ternary Neural Networks. The Teacher NN is trained on full precision weights and the outputs are ternarized. The student network then performs Ternarization of the weights of the teacher network based on optimal thresholds. The authors propose a teacher-student approach for training. The teacher network is trained with full precision weight and the student network uses a ternarized architecture to mimic the teacher's behaviour.Even though this model is not a CNN, we still get an insight on the hardware performance in terms of throughput, Latency and energy efficiency.

High performance FPGA based accelerator [19] , has a FPGA based CNN accelerator for AlexNet with all the layers mapped onto the single chip. In this approach, the weights for all the layers are stored onto an off-chip memory and intermediate results in between the layers are stored in on-chip buffers (BRAMs). These buffers facilitate data reuse. The architecture uses parallelism within the convolution operation, parallelism among multiple outputs feature maps and parallelism within multiple input feature maps. Many authors have tried to maintain a trade off between throughput and accuracy [21][16]however their throughput is in the range of Gigaops[28][7][3][10][23][26][20].

Ristretto [11] is an automated framework for neural network approximation. The open-source framework experiments with varying bit-widths of input operators to reduce the resource requirement per operator. The objective in reducing bit-width precision is to lower the area requirement for the processing elements, memory and reducing off-chip

memory communication. The authors analyze the impact of quantization on both training and inference for networks. The authors show 32-bit floating point data types in CNN can be approximated well by using 8 -bit and 4-bit representations as shown in Figure 2.2. For a hardware implementation, this reduces the size of multiplication units by about one order of magnitude. The required memory bandwidth is reduced by four to eight times. The savings in memory utilization in turn imply that more parameters can be held in the on chip buffers.

Network	Layer outputs	CONV parameters	FC parameters	32-bit baseline	Fixed point accuracy
LeNet (Exp 1)	4-bit	4-bit	4-bit	99.15%	98.95% (98.72%)
LeNet (Exp 2)	4-bit	2-bit	2-bit	99.15%	98.81% (98.03%)
Full CIFAR-10	8-bit	8-bit	8-bit	81.69%	81.44% (80.64%)
CaffeNet	8-bit	8-bit	8-bit	56.90%	56.00% (55.77%)
SqueezeNet	8-bit	8-bit	8-bit	57.68%	57.09% (55.25%)
GoogLeNet	8-bit	8-bit	8-bit	68.92%	66.57% (66.07%)

Figure 2.2: Ristretto: Bit Width Comparison

Binary quantization has been used in neural networks constrain the weights and/or activation to +1 and -1 during training and inference, which results in significant speedup and reduction in memory requirement [13][5][25].

Ternary CNNs on FPGAs and GPUs have been compared by the authors Nurvitadhi et al [22][4][6]. In the paper, the weights and input neurons are loaded into on-chip buffers in ODM from memory.The convolution and fully-connected layers are computed by dynamically flattening the weights and input feature maps (neurons) onto blocked matrix operations. The GEMM Unit performs such matrix operations and outputs the result to non linear unit for ReLU/BatchNorm/Pooling layers, as dictated by the desired Deep Neural Network configuration. The output goes into the on-chip buffer to be read by the next convolution/FC layer. If there is not enough buffer the output is spilled out to off chip memory.The authors conclude that their FPGA version of ResNET is 60% better than state-of-the-art GPUs. However, the best case performance for ResNET is still below 16 TOPS.

2.2 Challenges

The current implementations are limited in performance and/or with experimented problem size. In contrast, our approach to accelerating CNNs is in exploring parallelism through the use of Ternary Weighted Networks (TWN) [18]. These ternary networks provide good accuracy and huge memory and computation savings.These are better than Binary Weighted Networks in the accuracy levels achieved across a range of networks. The FPGA implementation of Ternary CNNs described by authors Nurvitadhi et al [22] also presents similar scheme of ternarization for the multiplication units. However, the adders in the implementation are of 32-bit floating point precision, which limits the best case performance to about 16 TOPS for for ResNET. Unlike the authors,Nurvitadhi et al [22], we propose to use 16-bit fixed point adders in our implementation.This further reduces the memory requirement and helps in improving the parallelism that can be achieved.

Chapter 3
TILENET ARCHITECTURE

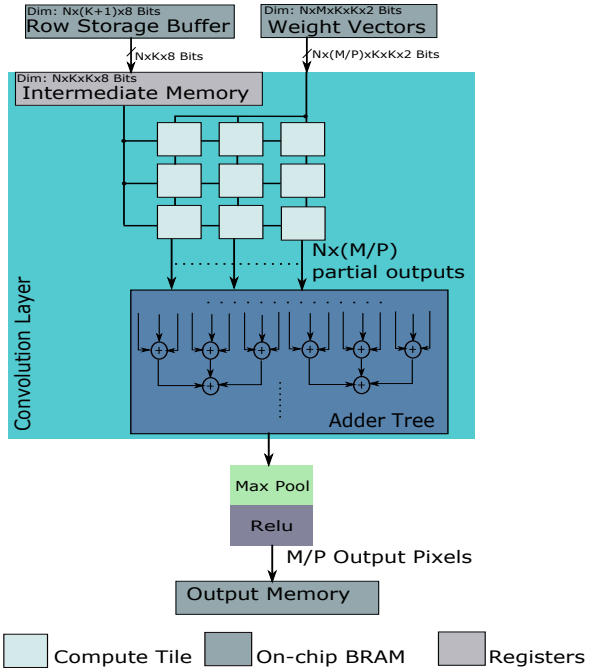


Figure 3.1: TileNET Accelerator Template

In this thesis, we introduce an architecture for realizing ternary CNN on FPGAs.TileNET is a generic architecture which is independent of network architecture and device specifics. TileNET architecture introduces an approach of tiling of the input feature maps and the weight vectors. This tiling in turn implies the tiling of computations involved. A high-level representation of the TileNET accelerator template is shown in Figure 3.1. Any convolution neural network architecture consists of many convolution layers. The dimensions of the input feature maps and the weight vectors vary with position of the layer in a network as well as change with the CNN architecture considered. Hence, our approach is to breakdown a single convolution layer into multiple tiles. This modular design makes it scalable and portable across CNN architectures.

The TileNET architecture organized as follows:

- Compute
- Memory
- Control

3.1 Compute

A single compute tile accommodates the entire computation required to process an input tile size of 3×3. The Tile size (T) of 3×3 has been considered because most of the layers in our reference CNN architecture(AlexNet), the weight vector of size 3×3 is used. In addition, the choice of 3×3 is a most frequently used weight vector size in newer CNN architectures. Weight vector sizes higher than 3×3 can be implemented by partitioning it into multiples of of 3x3 tiles. For example, to implement a weight vector of size 11×11, 16 such 3×3 tiles can be used.

Each compute tile performs multiple MAC operations to process an input kernel size of $N \times 3 \times 3$ and corresponding weight vectors. The input feature map represented by pixels, each as a 8-bit fixed point vector. As weights are ternarized, they are represented by values -1, 0, 1. These are represented in 2 bits as (00, 01, 11 for 0,1 and -1 respectively) are sufficient for ternary representation as shown in Figure 3.2

Weight Vector	Output
"00"	0
"01"	Input
"10"	0
"11"	2's compl (Input)

Figure 3.2: Ternary Weights

Ternarization of weights can be done by using optimal threshold values. This can be done deterministic or stochastic fashion. This process is generally done in the training phase. For our thesis, we consider the CNN to be trained and ready with ternary weights. For the ternarized weights, the multiplication operation is simply reduced to a multiplexing structure, as shown in Figure 3.5. For a weight value of '0', the output is a '0'. For a weight of '1', the input vector value itself is the result of multiplication. For a weight of '-1', the result of multiplication is a two's complement representation of the input vector value to represent subtraction.

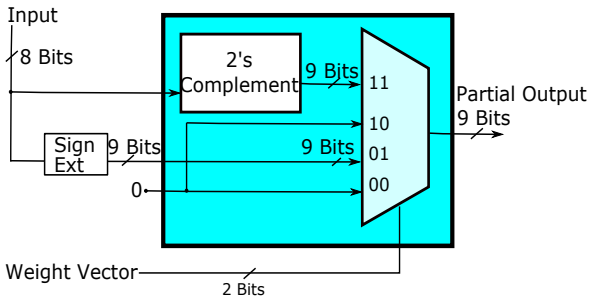


Figure 3.3: Ternary Multiplier

This form of simplification for the multiplication reduces the resource requirement for a multiplier, which is simply replaced by a 4x1 multiplexer with a two's complement computation as shown in Figure 3.3.As a result, the multiplier with ternarized weight vector has significantly lower resource requirements as compared to an 8-bit fixed-point multiplier. This reduction in resource requirement gives way to accommodating higher number of MAC units that can be realized in parallel.As shown in Figure 3.5, a single tile consists of 9 ternary multipliers followed by 3-input adder tree. Multiple such tiles are instantiated in parallel for computing a part of a convolution layer.

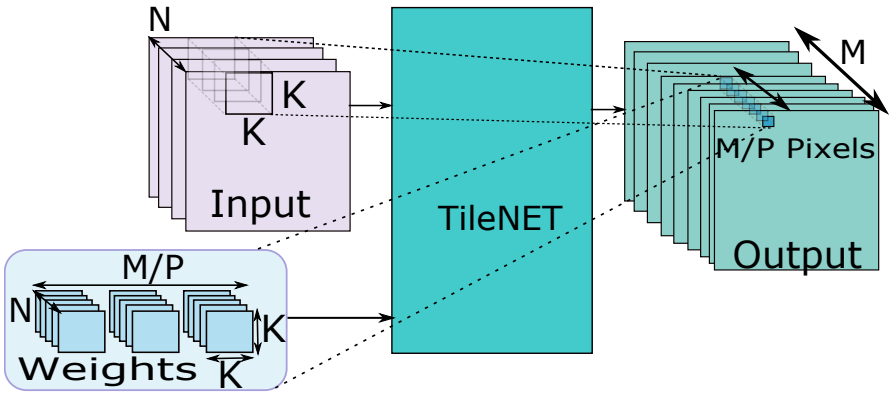


Figure 3.4: Parallel Computation of M/P Output Pixels

In any convolution layer, the input image size is considered to be of depth N with R rows and C Columns of pixel intensities(e.g. Layer 1 input image or the input feature map has dimensions: $3 \times 227 \times 227$ where N=3 for RGB channels and $R \times C$ as 227×227). Due to resource constraints, all the output pixels cannot be computed in parallel for any convolution layer. So we introduce a parallelization factor P which determines the maximum number of pixels that can be generated at a time. For a total depth of M of the output feature maps or output vectors of dimension $R_{out} \times C_{out}$ in any convolution layer, M/P output pixels in cycle are generated by ensuring that these compute operations require one clock cycle as shown in Figure 3.4

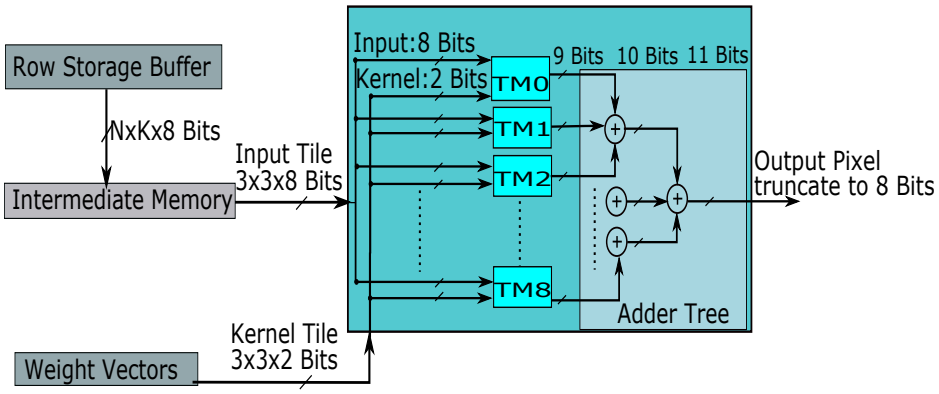


Figure 3.5: Processing Element

The convolution layer is followed by Rectified Linear Unit (ReLU) and Max Pooling. The non-linearity in the CNN is introduced through the ReLU layer which performs $max(0, input)$. The size of the input vectors remains unchanged. The pooling layer is introduced between different convolution layers to reduce the dimension by downsizing. Max pooling is performed by representing a neighborhood of values by their maximum value. The neighbourhood is defined by the stride considered as shown in Figure 3.6

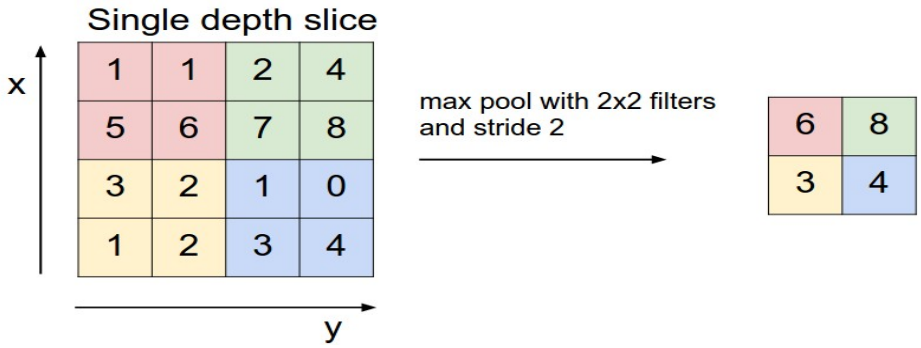


Figure 3.6: Max Pooling[14]

3.2 Memory Organization

The inputs to the CNN are represented by $R \times C \times N$ array of 8-bit fixed point values. The ternary weight vector has the dimensions of $M \times N \times K \times K$.The Row Storage Buffer (RSB) is the first level of storage for the input vectors.In RSB $(K+1)$ rows of the entire input feature map are stored.The RSB is implemented in Block RAMs(BRAMs) within each tile for close coupling with the ternary multipliers. The input vectors of dimension $N \times 3 \times 3$ that is required to feed the compute elements in a tile, is stored in an intermediate memory . The tiling and partitioning of input data is shown in Figure 3.8. At the beginning , the entire input image is stored in the off-chip memory. $(K+1)$ rows of input kernel are copied to the on-chip Block RAM memory (BRAM) in RSB, where K is Kernel or weight vector width of the layer. As shown in Figure 3.8 , in the first cycle T_i,the first column of the top K BRAMs in RSB is transferred onto the intermediate memory. In the next cycle T_{i+1} the $(K+1)^{th}$ column data is fetched from the input memory RSB to the next tile in the intermediate memory. The intermediate memory stores only the $N \times K \times K$ input vector values. Each time this chunk of data is processed in the convolution layer, the columns in the intermediate memory are left-shifted and left-most column is flushed out and a new column of K values is fetched from the RSB into the intermediate memory as shown in Figure 3.8. After the parallel convolution operations are performed on the input vectors, the data from the convolution layers is stored into output memory. As only M/P output pixel values are computed at a time, the input vector values in the intermediate memory are reused P times till all the computation with the weight vectors have been completed. After P iterations, M output pixels in depth are stored to the output memory.The data from the output memory serves as the input for the compute of the next layer. The output memory is implemented as another RSB which is shared between the two layers. As the computation for all the convolution layers is tiled, the next layer can begin its compute after the required tile T is available.

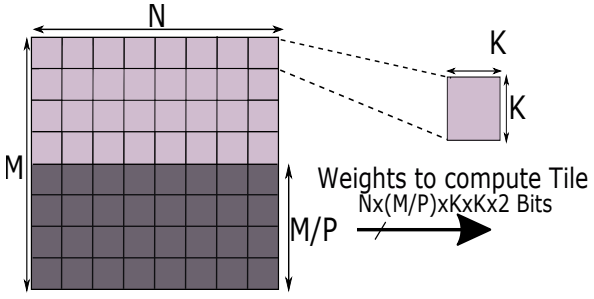


Figure 3.7: Weight Memory Organization

For example,if we consider a convolution layer $ConvLayer_i$ with the parameters as N, M, R, C. The parallelization factor P determines the number of output pixels that are computed in parallel in one cycle,i.e.,M/P output pixels in depth are generated. In the next cycle, the next M/P output pixels in depth are generated. After P such iterations, all the output pixel in the depth of the first row and column will be generated. The new column of the input vector is fetched into the intermediate memory and the process is repeated. The weight vectors are each represented by 2 Bits on account of ternarization and are smaller in size. As a result,all the weight vectors are stored in on-chip BRAMs. By moving the weight vectors closer to the ternary multiplier through the use of distributed memories results in further performance improvement. $K \times K$ weight vectors are stored in a single location with depth M/P and $P \times N$ such memories are used as shown in Figure 3.7

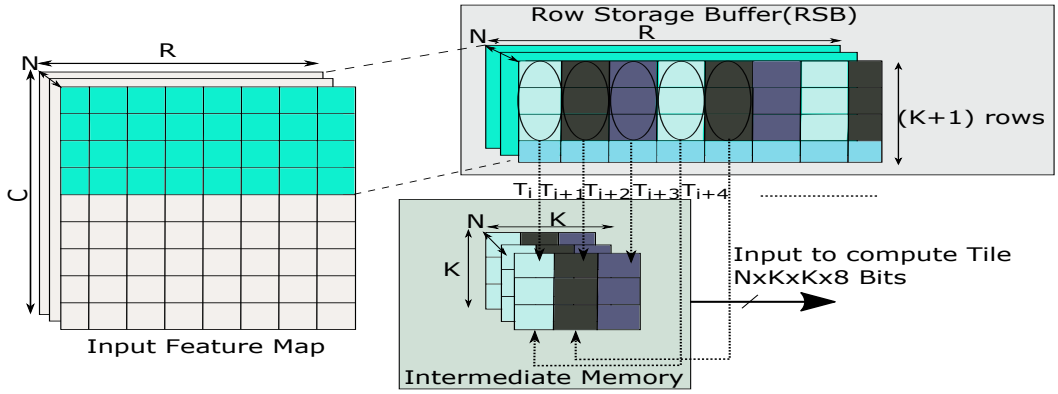


Figure 3.8: Memory tiling and Organization

3.3 Control Unit

The movement of data from memory to the compute units in the tile is orchestrated by the Control unit. The pipe-lining of all the units in the tiled organization necessitates the controlled input of data (both from input and weight vectors) to each tile. The resource constraints in terms of number of computation through LookUp Tables (LUTs) and the memory (BRAMs) available dictate the number of parallel tiles that can be accommodated on the FPGA. The factor P can be adjusted to accommodate parallel computations. The Finite State Machine for the control path is shown in the Figure 3.9 Each of the state transitions occurs at the rising edge of the clock.The input vectors are loaded column wise from the RSB into the intermediate memory.

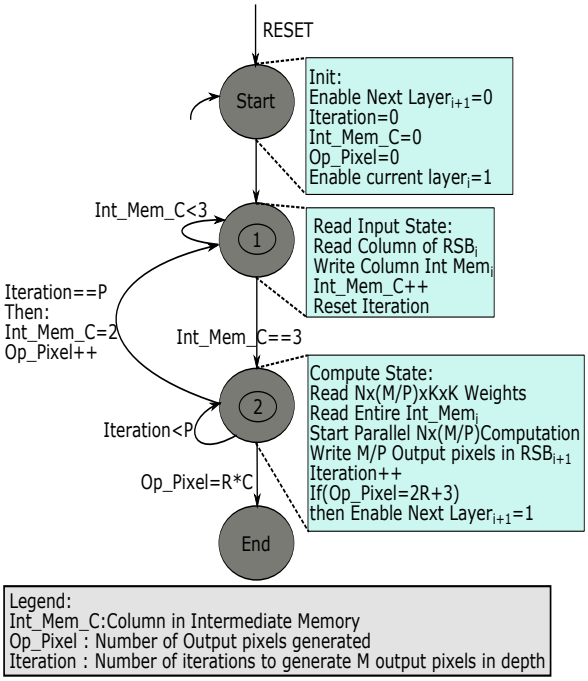


Figure 3.9: Control FSM for Convolution Layer $Convlayer_i$

Once the N such 3×3 input tiles are ready in the intermediate memory and the corresponding $N \times (M/P) \times K \times K$ weight vectors are available, the parallel computation of M/P output pixels is started. The Compute state is repeated P times, each time reading a new set of M/P weight vectors. The input vectors in the intermediate memory are reused for these P iterations. Once this is completed, the next N columns of RSB are brought into the intermediate memory and the process is repeated. When a $M\times3\times3$ tile is available in the output memory(RSB), the next layer compute is enabled. A similar control path is followed for the next state.

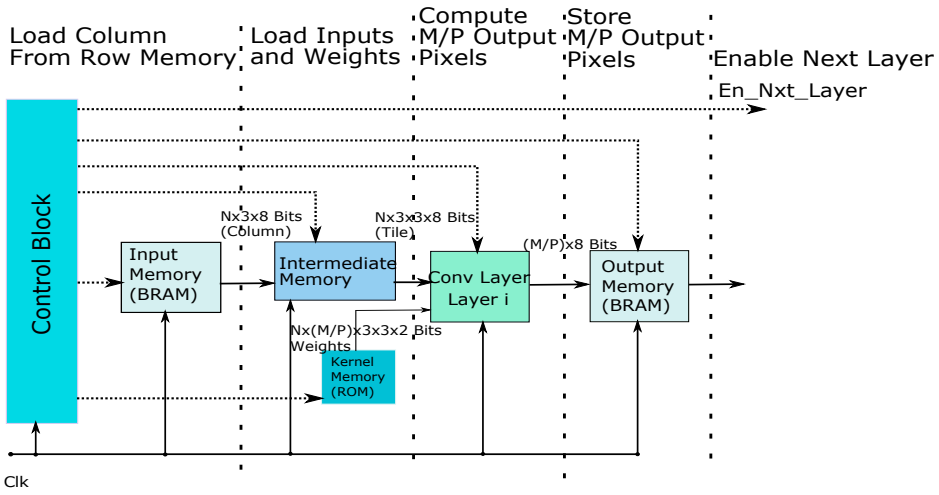


Figure 3.10: Control Path For Convolution $ConvLayer_i$

All the stages are pipelined in the TileNET Streaming architecture for a given CNN network. Once the pipeline is full, each stage performs the computation of M/P output pixels at every cycle. All the layers operate in parallel in the streaming architecture mode.

3.4 TileNET Variants

TileNet is a generic tiled architecture which can be tailored to suit the application and the device. The customization of the TileNet is to meet the following:

- Computational demands of any application-specific network architecture
- Device specific constraints on compute and memory resources

The variations of TileNET mainly depend on the parallelization factor P which determines the number of tiles instantiated in parallel at a time. In addition, there are two more architectural variants that TileNET supports :

- Streaming Mode
- Systolic Mode

These architectural variants differ in the number of convolution layers processed at a time and are detailed in the following sections.

3.4.1 Streaming Architecture

In streaming architecture mode, the data flows from one layer to another in a stream lined fashion. All the convolution layers in a given CNN architecture exist together on the given device, as shown in Figure 3.11. At steady state, in this pipe-lined approach, each layer provides enough output to trigger computations in the subsequent layers. Consequently, all the layers perform computations in parallel. As all the layers in the architecture need to be accommodated, each of the layer is provided with limited amount of parallelization.The partial output feature maps with M/P output pixels which are generated at every cycle from each $layer_i$ are stored in RSB_{i+1} BRAMs and passed to the next $layer_{i+1}$.

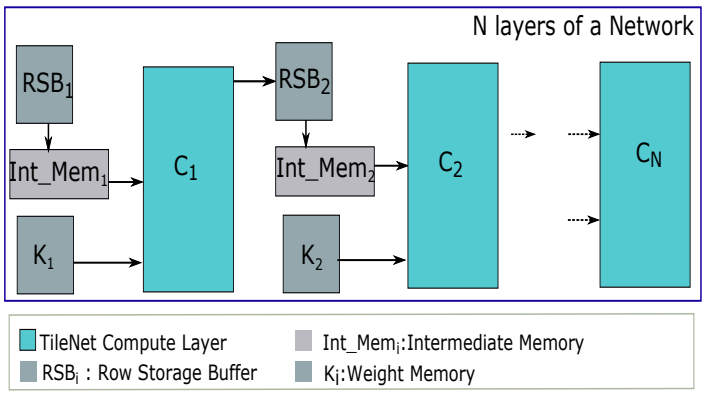


Figure 3.11: TileNET in Streaming Mode

3.4.2 Systolic Architecture

In the systolic architecture, all the resources on the device are used to implement a particular convolution layer. The maximum resources required by any of the layers that can be accommodated on the device is found and given for the implementation as shown in Figure 3.12. The execution of the network architecture is serialized. Each layer finishes all the computations that are mandated by the architecture and only after the completion of the layer can the computation for the next layer be instantiated. This type of computation in a staggered fashion is known as systolic.

As a result, layers are executed one after the other with the output of each layer stored onto on-chip memory. The contents of this memory act as the input for the next layer once the layer has finished the computation. Once the layer completes its execution a signal to enable the next layer is generated. This signal can be used to select the input weights for the next layer.and In case of an overflow, the excess output will be accommodated on an external memory. However, this adds onto the total delay for processing the entire image as well as

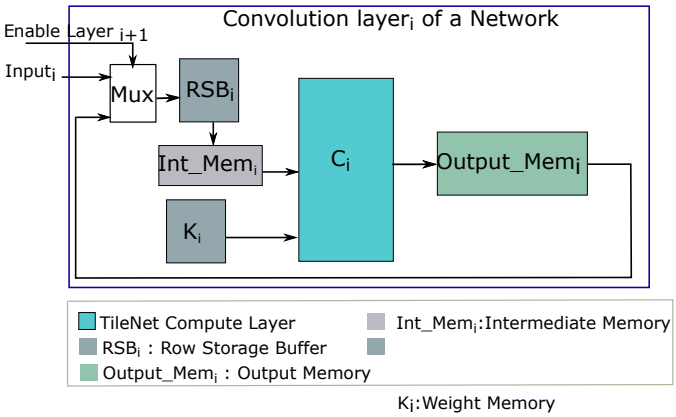


Figure 3.12: TileNET in Systolic Mode

time spent on each layer. As only one layer operates at a time, more number of output pixels can be generated in parallel as compared to streaming architecture.

3.5 Scalability

For processing $N\times$ 3 $\times$ 3 input vector tiles to generate M/P output pixels in parallel,$N \times (M/P)$ tile computations have to be done in parallel. The parallelization factor(P) in Figure 3.4 determines the number of tiles that can be operated in parallel. M/P also determines the number of operations have to be done in a layer. Scalability across different network architectures can be achieved by changing the parameter P.Given a set of resource constraints for a particular device, different network architectures can be accommodated in a streaming fashion as per the maximum permissible compute and memory by required it. In smaller networks like LeNet, as the convolution layers are fewer and the dimensions of parameters are smaller, maximum number of output pixels can be computed in depth,i.e. keeping P as 1. We can even compute pixels in parallel across rows and columns improve the compute utilization. ResNet-50 architecture has the maximum number of convolution layers of all the architectures considered in this paper. To accommodate a streaming architecture, initial layers compute a pixel at a time and then later layers can produce only partial outputs every clock . So we introduce parallelization factor(F) for input vector depth of N to ensure that the compute utilization doesn't exceed the resources available.Portability across devices is ensured by varying P as per the resource constraints for each FPGA device.

Chapter 4
PERFORMANCE ESTIMATION MODEL

The need for a performance estimation model arises as the amount of parallel computations that can be performed depends on the device resource constraints and the network architecture chosen. As in this thesis, we propose a generic scalable architecture for TileNET, a performance estimation model is a pre-requisite for the implementation. Our performance estimation model is elaborated in Figure 4.1.The model based on TileNET uses device-level resource information and estimates the performance of a given network in terms of TOPS. The input for the estimation model are the following device specific constraints and the CNN architectural parameters.

- Device: Memory Constraints in terms of number of BRAMs available for the particular FPGA Device
- Device: Compute Constraints in terms of number of LookUp Tables(LUTs) available for the particular FPGA Device
- Network Architecture in terms of number of convolution layers and the dimensions for each layers inputs, outputs and weights.

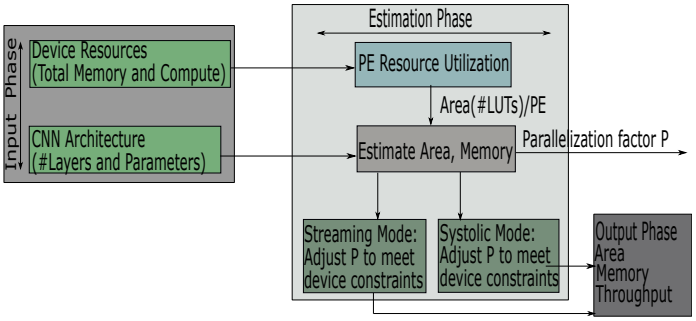


Figure 4.1: Estimation Model

The estimation model uses the resource requirement for a single computation tile which is obtained through the implementation reports for the selected specific FPGA device family.The model then estimates the number of tiles that can be accommodated on the specific device while ensuring maximum utilization in terms of memory and computation. For the streaming

architecture, the maximum number of tiles for all the layers combined is adjusted to obtain the parallelization factor *P*.

For the systolic architecture, the maximum number of tiles for the largest possible layer to achieve maximum utilization is considered. Thus the parallelization factor for both streaming architecture and the systolic architecture are estimated. Based on this parallelization factor, the total number of computations that can happen simultaneously in an ideal scenario can be calculated. The generic nature of TileNet, allows for a customized arrangement, while aiming at the high throughput. The estimations are done as a pre-processing step, before implementing on a device. This allows for a good idea of how much utilization can be achieved while maintaining a particular high throughput.The estimation model has been modeled in Microsoft excel.

Chapter 5
RESULTS AND ANALYSIS

5.1 Experimental Setup

The estimation model has been modeled using Microsoft excel. The family of devices considered are from Xilinx FPGAs. The TileNET architecture has been implemented in Verilog using Vivado Design Suite. The inputs for the Estimation Model are the CNN network parameters as well as the FPGA device constraints. The parallelization factor P is adjusted for maximum resource utilization and based on this the throughput,area and memory are estimated.

5.1.1 CNN

Deep CNNs have multiple convolution layers with varying network parameters. Each convolution layer has different parameters like N, M, R, C which determine the amount of parallelism possible.The number of convolution layers and the weight vectors used in the some of the network architectures is elaborated in Table 5.1.

Table 5.1: CNN Architectures

CNN	#Conv Layers	Weight Vectors
LeNet	2	5x5
VGG-16	13	3x3
AlexNet	5	11x11,5x5,3x3
ResNet-50	70	7x7,3x3,1x1

5.1.2 FPGA

FPGA Device constraints in terms of number of compute elements or LUTs and the on-chip memory in terms of BRAMs limit the amount of parallel computations that can occur at a time. Virtex FPGA have abundant resources while smaller devices like Artix are resource constrained.In this thesis, we have considered different devices like Artix, Virtex, Zync and Kintex of the Xilinx family of FPGAs. The resource constraints for the different devices considered in this thesis have been elaborated in the Table 5.2.

Table 5.2: FPGA Device Constraints [30]

Device	#LUTs	#BRAMs(36Kb)
Artix	134600	13140
Kintex	298600	34380
Zync	277400	27180
Virtex	712000	67680

5.2 Performance Estimation Model

The performance estimation model estimates the area, memory and throughput based on the implementation results for a single PE. A single PE is the core of the TileNET architecture, as it forms the basic computaion module as elaborated in the previous chapters. The estimation model requires the implemenation result values for the compute and memory for the PE across various FPGA families to accurately determine the estimations.The validation of the estimation model is carried out by performing the implementation of the different convolution layers in the AlexNet architecutre. The estimation for AlexNet layers in terms of memory and LUT usage is elaborated further in the following sections.

5.2.1 PE Results

For a single PE, the implementation is done on the different devices like Virtex, Kintex, Zync and Artix. The results for the implementation are shown in Table 5.3 show that the utilization remains similar across devices.The total on-chip power consumption for each tile is also shown in the Table 5.3.

Table 5.3: Implementation of PE on different devices

Device	#LUTs	#FFs	Delay(ns)	Power(W)
Virtex-7	318	600	1.581	0.752
Kintex	317	600	1.529	0.310
Zync	319	600	1.587	0.363
Artix	319	600	2.398	0.236

From the implementation results on the different devices, it shows that Virtex 7 have a better frequency of operation while having the maximum resources. This is ideal for higher throughput when compared to other devices.

5.2.2 AlexNet:Compute Estimation

The AlexNet network architecture for the different convolution layers is shown in Table 5.4.The estimation for the maximum compute for AlexNet based on TileNET template considering the device as Virtex 7 is shown in Table 5.5.The amount of parallelization for all the convolution layers in AlexNet is adjusted by varying P to achieve the maximum utilization of the device.

Table 5.4: Network Architecture of AlexNet

Layer	N	M	K	#Tiles	Rin	Cin	Rout	Cout	Max Pooling
Conv1	3	96	11x11	16	224	224	55	55	3x3
Conv2	48	256	5x5	4	55	55	27	27	3X3
Conv3	256	384	3x3	1	13	13	13	13	NA
Conv4	192	384	3x3	1	13	13	13	13	NA
Conv5	192	256	3x3	1	13	13	13	13	3X3

As shown in Table 5.4, the different parameters for the convolution layers in AlexNet are elaborated. The parameters for the input image are: N,M,R_{in},C_{in} and for output image are: M,R_{out} ,C_{out}. The parameters for the weight vectors are M,N and K. As seen in the Table 5.4, the third layer is the largest in terms of computation required. The Table 5.5, shows the estimations with different parallelization factor P for each of the layers. The maximum parallelization is for the smallest convolution layer i.e. layer 1, and here 6 output pixels will be computed at a time. In order to accommodate the other layers in a streaming architectural scheme, the parallelization in terms of number of pixels computed for the output have to be limited. In this scenario, the parallelization is limited to 2 output pixels in parallel.

Table 5.5: Estimation of Resource(LUT) for AlexNet on Virtex 7

Layer	P	M/P	#Estimated MACs	#Estimated LUTs
Conv1	16	6	2592	97344
Conv2	128	2	3456	129792
Conv3	192	2	4608	173056
Conv4	192	2	3456	129792
Conv5	128	2	3456	129792
Total	17568	659776		

The maximum number of parallel computations are limited to the total number of compute elements that are available in the target device. In Table 5.5, the estimation bound is calculated with the target device as Xilinx FPGA: Virtex 7 series.

5.2.3 AlexNet: Memory Estimation

The memory requirements for any network depend on the parameters in each layer and the number of layers. In this estimation model, we only consider the convolution layers in any deep CNN. The BRAMs are the on-chip memory resources in Xilinx 7 FPGA series. Each BRAM can be configured to store up to 36Kb of data. The 36Kb BRAM can be configured as two independent 18Kb BRAMs or a single 36 KB BRAM depending on the memory requirement. The maximum number of available BRAMs form the constraint for the memory that can be accommodated.The input and weight vectors across layers are stored in RSB and weight memory storage which are implemented as BRAMs.The Table 5.6 shows the memory requirement for different Layers in AlexNet. The weight vectors of size 3×3 can be accommodated using multiple 18Kb BRAMs. These 18Kb BRAMs utilize only half of the 36Kb BRAM as seen in the implementation experiments. Larger weight vectors like 11×11, require the entire 36Kb BRAM as a basic memory element.

Table 5.6: Memory Requirement in AlexNet(BRAMs(36Kb))

Layer	**M/P**	**#Input**	**#Weight**	**#Total**
Conv1	6	6	72	78
Conv2	2	36	48	84
Conv3	2	96	64	160
Conv4	2	72	48	120
Conv5	2	72	48	120

As the layer 3 in AlexNet is the largest and requires the maximum number of BRAMs. By the ternarization of weights, the weight memory requirement is greatly reduced by almost 93% as shown in Table 5.7.

Table 5.7: Comparison:Weight Memory Requirement With and Without Ternarization(BRAMs(36Kb))

Layer	**M/P**	**#Ternary Weights**	**#Full Precision Weight**
Conv1	6	72	1152
Conv2	2	48	768
Conv3	2	64	1024
Conv4	2	48	768
Conv5	2	48	768
#Total	-	280	4480

5.2.4 Resource-Throughput Analysis

The maximum throughput achievable for any CNN architecture is dependent on the resource constraints of the target FPGA device. Specifically, the resources in terms of memory(BRAM) and compute(LUT) on an FPGA device are the main constraints to achieve the maximum throughput for a CNN architecture.

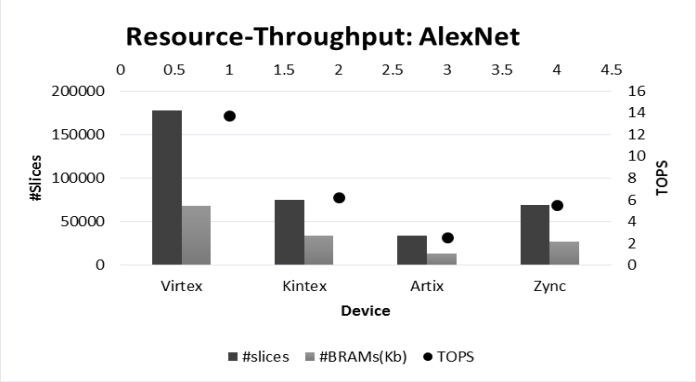


Figure 5.1: AlexNet: Resource - Throughput across devices

The relationship between the throughput and the resources available on the different FPGA devices is shown in Figure 5.1.Virtex-7 FPGA device has the maximum resources in terms of available BRAMs and LUT for computation.The throughput achieved is the maximum for TileNET implementation of AlexNet at 13.76 TOPs. Artix FPGA devices are extremely constrained in terms of resources available. This constraint directly effects the maximum throughput that can be achieved and it is reduced to 2.6 TOPS. The plot in Figure 5.1, thus clearly illustrates the relationship between the resource constraints on devices and the maximum achievable throughput considering the ALexNet CNN architecture.

5.2.5 Portability across Devices

The portability of any CNN network architecture across devices is ensured by using the generic and modular TileNET template. As seen in the previous section ,different FPGA devices, have different resource availability. The plot of variation in (in TOPS) with the maximum device size for the systolic and streaming for various networks is shown in Figure 5.2. Virtex FPGA have the maximum resources in terms of LUTs and BRAMs and therefore are able to accommodate more parallel computations, hence increasing the maximum throughput achieved.

Resource constraints vary across different devices and the utilization for AlexNet is shown in Figure 5.3. The utilization for memory and compute have to be balanced to

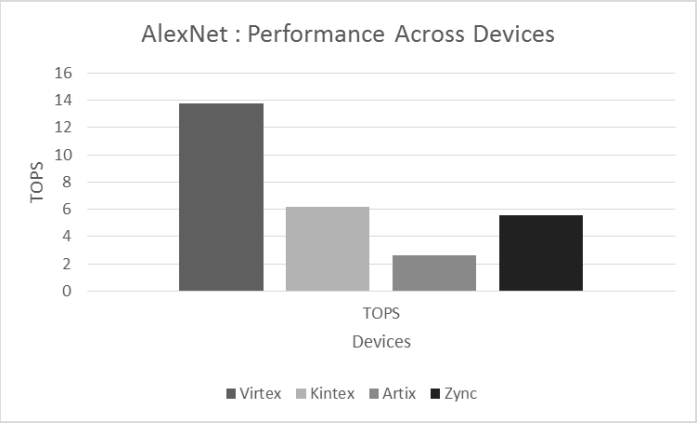


Figure 5.2: AlexNet:Throughput Across Devices

accommodate the maximum possible parallel computations. Virtex-7 FPGA devices have the maximum resources in terms of memory and compute LUTs amond the devices considered in this thesis. But the additional resources make the devices more expensive. Zync FPGAs are equipped with an additional processor core which is usually an ARM core. This opens up avenues of offloading some operations on to the micro controller core. Artix FPGAs on the other hand are resource constrained and the number of parallel computations are severely limited. The limitation is not only because of the compute elements available but also the available memory is not sufficient.The application of this device may thus be more suitable for smaller networks.

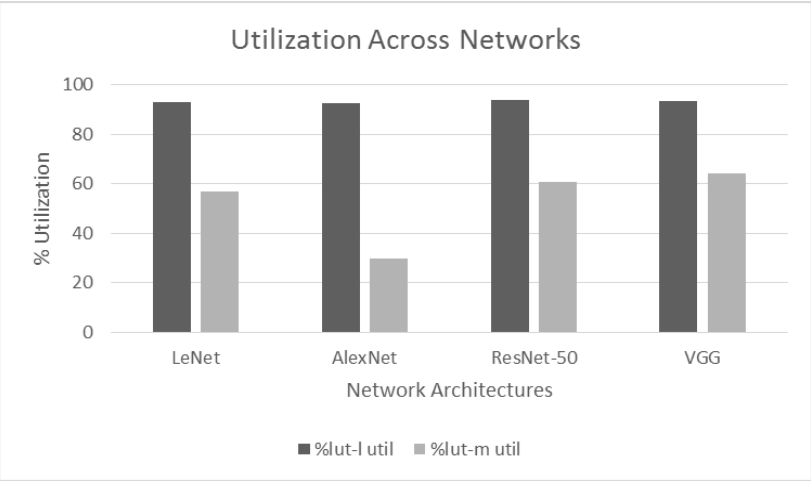


Figure 5.3: AlexNet:LUT and BRAM Utilization Across Devices

5.2.6 Scalability across Deep Networks

Deep CNN have varying number of layers and the arrangement of the convolution layers. In this section, we compare the performance and utilization across different CNN architectures like AlexNet,LeNet,VGGNet-16 and ResNet-50. The Figure 5.4 shows the performance in TOPs across CNN architectures while keeping a similar resource utilization threshold on Virtex-7 device. Deeper networks like ResNet with many convolution layers increase the number of operations that are performed at a time and show slightly higher throughput values.LeNet is a small network with only 2 convolution layers. Despite of generating more number of output pixels, the throughput is slightly lower. The performance is comparable across different network architectures and exceeds 13 TOPs.

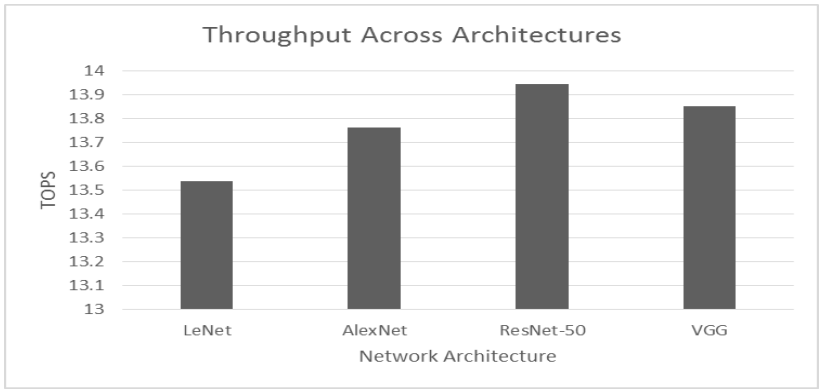


Figure 5.4: Estimated Performance Across Networks

5.3 Validation

5.3.1 AlexNet: Implementation Results

The performance estimation model provides the area and memory estimates based on the maximum amount of parallelization.Using the P values from Table 5.5. TileNet was customized for AlexNet network which has 5 convolution layers. The ternary computes are the total number of ternary operations that happen at a time and are dependent on the parallelization factor as :

$No.TernaryCompute = N * (M/P) * K * K$

Through implementation of the basic compute tile onto the Virtex-7 FPGA device, we can find the number of LUTs required. So based on this, the total no of LUTs can be computed. The total number of LUTs for the layer would be based on the number of parallel computations that can be done, which depends on the the factor P. Based on the parallelization factor

P, the maximum throughput that can be achieved can be estimated. As the throughput is calculated in number of operations that can be performed in one second.The throughput can be calculated by:

$Throughput = (TernComp * 2) * F_{max})$

Where the Total number of ternary computations($TernComp$) is the summation of the compute requirement across all the layers in streaming mode.The ternary computation is in essence two operations,Multiply and Addition.So the total number of operations is twice the $TernComp$. Also, F_{max} is the maximum frequency of operation for the particular layer.

Table 5.8: Implementation Results for AlexNet on Virtex 7

Layers	#MACs	#LUTs	LUTs Util %	#FFs	FF Util %	#BRAMs	BRAM Util%
Conv1	2592	94177	13.22711	187488	13.16629	78	4.148936
Conv2	3456	125535	17.63132	249882	17.54789	84	4.468085
Conv3	4608	167391	23.50997	333290	23.4052	160	8.510638
Conv4	3456	125535	17.63132	249882	17.54789	120	6.382979
Conv5	3456	125535	17.63132	249882	17.54789	120	6.382979
Total	17568	638173	89.63104	1270424	89.21517	562	29.89362

The convolution layers for AlexNet using the TileNET accelerator template have been implemented on Virtex 7 FPGA device using Vivado Design Suite. The coding has been done in Verilog and the implementation results have been obtained. The implementation results are summarized the Table 5.8. The implementation results are promising as the maximum achievable frequency of operation is around 390 MHz for the slowest layer. This will help in achieving high throughputs.

Table 5.9: Timing Implementation Results for AlexNet on Virtex 7

Layers	DELAY(ns)	Freq(MHz)	Power (W)
Conv1	1.964	509.165	12.395
Conv2	2.481	403.0633	12.563
Conv3	2.553	391.696	16.008
Conv4	2.021	494.8046	15.247
Conv5	2.021	494.8046	15.247
Total	2.553	391.696	71.46

The implementation results validate what the estimation model predicts.The comparison between the performance estimation model and the implementation results for the layers in AlexNet is detailed in the Table 5.10. The results show that our estimation model gives conservative estimates for memory and compute usage for Virtex-7 device.

Table 5.10: AlexNet: Estimation vs Implementation Results

Layer	#Est. BRAMs	#Impl. BRAMs	#Est. LUTs %)	#Impl. LUTs %)
CONV1	78	78	13.672	13.227
CONV2	84	84	18.229	17.631
CONV3	160	160	24.306	23.510
CONV4	120	120	18.229	17.631
CONV5	120	120	129792	17.631

5.3.2 Latency

The systolic and the streaming architectures differ mainly in the latency of operation.The latency comparison for streaming and systolic architectures for AlexNet is shown in Table 5.11. The latency is in terms of the number of clock cycles required to complete the processing of an entire image. As shown in the Table 5.11, the systolic architecture requires more cycles to process the entire image for AlexNet. In a scenario, of a very small CNN architecture, this difference may not be so significant. The choice between the two architecture is a designer's decision while considering the resource utilization, throughput and latency.

Table 5.11: Latency(Clock cycles)

Mode	Latency
Streaming	50832
Systolic	242525

Chapter 6
CONCLUSION & FUTURE WORK

In this thesis, we have designed a novel CNN architecture template which achieves high accuracy and throughput of 13 TOPS through a scalablility and re-configurablility. We have designed a performance estimation model which computes the memory, logic utilization and performance for a particular CNN architecutre and a target device based on the TileNET template.
As a part of the preliminary analysis and experiments, we have an estimate on the resources in terms of memory and the logic that is required for a large networks like AlexNet,VGG-16 and ResNet-50. As the resources required are quite large and cannot be accommodated for all the layers on the on board memory of the FPGA, we have to look into a highly parallel architecture that will fulfill these requirements. As a part of this thesis, we have devised a novel tiled architecture with tiling for the input and pipe-lining all the layers so that at a given time a part of the image can be processed. Systolic architectural approach where only one layer at a time is processed has also been explored in this thesis.

Hybrid architecture with systolic and streaming architectures may be considered for achieving high throughput and high resource utilization, as a part of future work. Techniques like zero skipping might help in freeing up more resources and improving parallelization will also help in improving the performance of the proposed model.

TileNET opens up bright avenues for implementing Ternary CNNs on FPGAs to achieve high throughput and accuracy as required for ADAS and other image processing applications.

Bibliography

[1] Albericio, J., Judd, P., Hetherington, T., Aamodt, T., Jerger, N. E., and Moshovos, A. (2016). Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 1–13.

[2] Alemdar, H., Caldwell, N., Leroy, V., Prost-Boucle, A., and Pétrot, F. (2016). Ternary neural networks for resource-efficient AI applications. *CoRR*, abs/1609.00222.

[3] Bekkerman, R., Bilenko, M., and Langford, J. (2011). *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, New York, NY, USA.

[4] Belwal, M., Purnaprajna, M., and Sudarshan, T. (2015). Enabling seamless execution on hybrid cpu/fpga systems: Challenges & directions. pages 1–8.

[5] Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc.

[6] et. al, N. P. J. (2017). In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760.

[7] Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., and Culurciello, E. (2010). Hardware accelerated convolutional neural networks for synthetic vision systems. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 257–260.

[8] Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., and LeCun, Y. (2011). Neuflow: A runtime reconfigurable dataflow processor for vision. In *CVPR 2011 WORKSHOPS*, pages 109–116.

[9] Farooq, U., Marrakchi, Z., and Mehrez, H. (2012). *Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*. Springer Science & Business Media.

[10] Gokhale, V., Jin, J., Dundar, A., Martini, B., and Culurciello, E. (2014). A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 682–687.

[11] Gysel, P. (2016). Ristretto: Hardware-oriented approximation of convolutional neural networks. *CoRR*, abs/1605.06402.

[12] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[13] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc.

[14] Karpathy, A. (2012). Cs231n: Convolutional neural networks for visual recognition. *http://cs231n.github.io/ Systems*.

[15] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*.

[16] Kumar, J., Kumar, J., Bhakthavatchalu, R., et al. (2016). Design and implementation of hodgkin and huxley spiking neuron model on fpga. pages 1483–1487.

[17] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[18] Li, F. and Liu, B. (2016). Ternary weight networks. *CoRR*, abs/1605.04711.

[19] Li, H., Fan, X., Jiao, L., Cao, W., Zhou, X., and Wang, L. (2016). A high performance fpga-based accelerator for large-scale convolutional neural networks. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–9.

[20] Mody, M., Nandan, N., and Sanghvi, H. (2016). Efficient vlsi architecture for sao decoding in 4k ultra-hd hevc video codec. pages 81–84.

[21] Murali, S., Kumar, J., Kumar, J., and Bhakthavatchalu, R. (2016). Design and implementation of izhikevich spiking neuron model on fpga. pages 946–951.

[22] Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J., Liew, Y. T., Srivatsan, K., Moss, D., Subhaschandra, S., and Boudoukh, G. (2017). Can fpgas beat gpus in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 5–14, New York, NY, USA. ACM.

[23] Ovtcharov, K., Ruwase, O., Kim, J.-Y., Fowers, J., Strauss, K., and Chung, E. (2015). Accelerating deep convolutional neural networks using specialized hardware.

[24] Peemen, M., Setio, A. A. A., Mesman, B., and Corporaal, H. (2013). Memory-centric accelerator design for convolutional neural networks. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 13–19.

[25] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279.

[26] Shin, D., Lee, J., Lee, J., and Yoo, H. J. (2017). 14.2 dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 240–241.

[27] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

[28] Tanomoto, M., Takamaeda-Yamazaki, S., Yao, J., and Nakashima, Y. (2015). A cgra-based approach for accelerating convolutional neural networks. In *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 73–80.

[29] Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P. H. W., Jahre, M., and Vissers, K. A. (2016). FINN: A framework for fast, scalable binarized neural network inference. *CoRR*, abs/1612.07119.

[30] Xilinx (2017). All programmable 7 series selection guide.