

COMPRESSED CONVOLUTIONAL NEURAL NETWORK FOR
AUTONOMOUS SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Durvesh Pathak

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2018

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Mohamed El-Sharkawy, Chair

Department of Engineering and Technology

Dr. Maher Rizkalla

Department of Engineering and Technology

Dr. Brian King

Department of Engineering and Technology

Approved by:

Dr. Brian King

Head of the Graduate Program

This is dedicated to my parents
Bharati Pathak and Ramesh Chandra Pathak,
my brother Nikhil Pathak
and my amazing friends
Jigar Parikh and Ankita Shah.

ACKNOWLEDGMENTS

This wouldn't have been possible without the support of Dr. Mohamed El-Sharkawy. It was his constant support and motivation during the course of my thesis, that helped me to pursue my research. He has been a constant source of ideas. I would also like to thank the entire Electrical and Computer Engineering Department, especially Sherrie Tucker, for making things so easy for us and helping us throughout our time at IUPUI.

I would also like to thank people I collaborated with during my time at IUPUI, Dewant Katare, Akash Gaikwad, Surya Kollazhi Manghat, Raghavan Naresh Sarangapani without your support it would be very difficult to get things done.

A special thank to my friends and mentors Alope, Nilesh, Irtzam, Shantanu, Harshal, Vignesh, Mahajan, Arminta, Yash, Kunal, Priyan, Gauri, Rajas, Mitra, Monil, Ania, Poorva, and many more for being around.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vii
LIST OF FIGURES	viii
ABBREVIATIONS	x
ABSTRACT	xii
1 INTRODUCTION	1
1.1 Motivation	3
1.2 Contribution	5
2 OVERVIEW OF CONVOLUTION NEURAL NETWORK	7
2.1 Network and Architecture	9
2.2 Convolutional Neural Networks (CNN)	10
2.2.1 Number of Parameters Calculation	13
2.2.2 Pooling	13
2.2.3 Non Linearity or Activation Function	14
3 OBJECT DETECTION	15
3.1 Informative Region Selection	15
3.2 Feature Extraction	16
3.3 Classification	16
3.4 Deep Neural Network for Object Detection	16
3.4.1 Faster R-CNN	19
3.4.2 YOLO	22
3.4.3 Motivation for Object Detection	23
4 BLUEBOX 2.0	25
4.1 S32V234 - Vision Processor	26
4.2 LS-2084A	27

	Page
4.3 Platform Overview	28
5 REAL-TIME MULTI SENSOR APPLICATIONS (RTMAPS)	30
6 REDUCEDSQNET ARCHITECTURAL ADVANCES	32
6.1 Accuracy	32
6.2 Computation Time	33
6.3 ReducedSqNet	37
6.3.1 Baseline Architecture	39
6.3.2 ReducedSqNet Architecture	47
6.4 Conclusion	56
7 SQUEEZED CNN - A NEW ARCHITECTURE FOR CIFAR-10	58
7.1 VGG-16 Architecture	59
7.2 Squeezed CNN	61
7.2.1 Hardware Deployment of Squeezed CNN	65
7.2.2 Bluebox 2.0 by NXP	66
7.2.3 Real-time Multisensor Applications (RTMaps)	66
7.2.4 Deployment of Squeeze CNN	66
7.2.5 Conclusion	67
8 INTEGRATING CNN WITH FASTER RCNN AND YOLO	70
8.1 Squeezenet integrated with Faster R-CNN	70
8.1.1 Modification for model compatibility	71
8.1.2 Detection Results	73
8.2 SqueezeNet integrated with You Look Only Once (YOLO)	75
8.2.1 YOLO Architecture	75
8.2.2 Detection Results	76
9 SUMMARY	78
REFERENCES	79

LIST OF TABLES

Table	Page
6.1	The table summarizes the base squeezeNet architecture 42
6.2	Table represents the number of parameters in reducedSqNet architecture, the table also highlights the position of pooling layers and batch normalization layers used in the architecture. 49
6.3	Results for ReducedSqNet vs squeezeNet v 1.1 57
7.1	Table highlights the number of parameters for each layer of VGG-16 architecture 59
7.2	Table represents Squeezed CNN architecture and Number of parameters in each layer. 62
7.3	Summary for VGG-16 vs Squeezed CNN 64
8.1	Model Comparison for Faster R-CNN vs Modified Architectures 73

LIST OF FIGURES

Figure	Page
1.1 Deep Learning a subset of Machine Learning	4
2.1 Biological Neurons [11]	8
2.2 Artificial Neural Network	9
2.3 Multi Layer Perceptron	10
2.4 Convolutional Neural Network [12]	11
2.5 Represent the output dimension calculation based on <i>Eqn 2.2</i>	12
2.6 Convolution Kernel and Receptive fields [18]	12
2.7 Max pooling using 2×2 Kernel	14
3.1 Object Detection Pipeline	15
3.2 Boost in Deep learning [19].	17
3.3 Boost in GPU performance [20].	18
3.4 Two train of thoughts for object detection are represented in the figure is inspired by [13] SPP-net [21], FRCN [22], Faster-RCNN [23], R-FCN [24], YOLO [25], SSD [26].	19
3.5 Faster R-CNN Architecture [23]	20
3.6 Bounding boxes generated using YOLO detection.	23
4.1 Hardware Architecture for Bluebox 2.0 [Pic Courtesy NXP].	25
4.2 Hardware Architecture for Bluebox 2.0 [Pic Courtesy NXP].	26
4.3 Hardware Architecture for S32V234 [Pic Courtesy NXP].	27
4.4 Hardware Architecture for LS2084A [Pic Courtesy NXP].	28
4.5 System overview of NXP BLBX2.	29
5.1 RTMap setup with Bluebox 2.0.	31
6.1 Accuracy measured on test batch.	33
6.2 Fire Module from squeezeNet [2].	39

Figure	Page
6.3 SqueezeNet architecture v1.1 adapted for CIFAR-10 dataset.	40
6.4 Test Loss with SGD (<i>Orange</i>) and Adam (<i>Blue</i>).	44
6.5 The plot represents the multiple runs with different hyper-parameters.	45
6.6 Test loss at learning rate of 0.0001.	46
6.7 Proposed ReducedSqNet architecture for CIFAR-10 dataset.	48
6.8 Accuracy curves for different hyper-parameters configurations.	50
6.9 Loss curves for different hyper-parameters configurations.	51
6.10 Testing accuracy of baseline model vs reducedSqNet.	52
6.11 Testing loss of baseline model vs reducedSqNet.	53
6.12 Experimentation Results with Exponential Linear Unit.	55
6.13 This figure shows the accuracy comparison of the base squeezeNet vs the ReducedSqnet vs ReducedSqnet with ELU.	56
7.1 This figure shows VGG-16 Architecture (<i>Left</i>) Proposed Architecture (<i>Right</i>).	60
7.2 Test Accuracy of squeezed CNN Architecture.	65
7.3 Flow for application development using RTMaps for Bluebox 2.0.	67
7.4 Graphical interface in RTMaps.	68
7.5 Console output on RTMaps for displaying results.	69
8.1 This figure represents the SqueezeNet Architecture	71
8.2 This figure represents the integration of SqueezeNet with Faster R-CNN	72
8.3 Detection Result for Faster R-CNN where CNN is VGG-16.	74
8.4 Detection Result for R-SqueezeNet (Faster R-CNN + SqueezeNet).	74
8.5 Detection Result for R-Squeezed CNN (Faster R-CNN + Squeezed CNN).	75
8.6 Representation of how YOLO works.	76
8.7 Detection result for SqueezeNet + YOLO on Real-Time Videos	76
8.8 Detection result for SqueezeNet + YOLO on Real-Time Videos	77
8.9 Detection result for SqueezeNet + YOLO on Real-Time Videos	77
8.10 Detection result for SqueezeNet + YOLO on Real-Time Videos	77

ABBREVIATIONS

CNN	Convolution Neural Network
NN	Neural Network
YOLO	You Only Look Once
SSD	Single Shot Detector
DNN	Deep Neural Network
SGD	Stochastic Gradient Descent
LR	Learning Rate
GPU	Graphic Processing Unit
TPU	Tensor Processing Unit
CPU	Central Processing Unit
FPGA	Field Programmable Gate Array
RTMaps	Real-Time Multisensor Applications
TCP/IP	Transmission Control and Internet Protocol
DSE	Design Space Exploration
ReLU	Rectified Linear Unit
ELU	Exponential Linear Unit
VGG	Visual Geometry Group
UAVs	Unmanned Ariel Vehicles
CV	Computer Vision
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
KSIZE	Kernel Size
SVM	Support Vector Machines
RPN	Region Proposal Network

ROI	Region Of Interest
NMS	Non-Maximal Suppression
IOU	Intersection Over Union
FCN	Fully Convolutional Neural Network

ABSTRACT

Pathak, Durvesh. M.S.E.C.E., Purdue University, December 2018. Compressed Convolutional Neural Network for Autonomous Systems. Major Professor: Mohamed El-Sharkawy.

The word “Perception” seems to be intuitive and maybe the most straightforward problem for the human brain because as a child we have been trained to classify images, detect objects, but for computers, it can be a daunting task. Giving intuition and reasoning to a computer which has mere capabilities to accept commands and process those commands is a big challenge. However, recent leaps in hardware development, sophisticated software frameworks, and mathematical techniques have made it a little less daunting if not easy. There are various applications built around to the concept of “Perception”. These applications require substantial computational resources, expensive hardware, and some sophisticated software frameworks. Building an application for perception for the embedded system is an entirely different ballgame. Embedded system is a culmination of hardware, software and peripherals developed for specific tasks with imposed constraints on memory and power. Therefore, the applications developed should keep in mind the memory and power constraints imposed due to the nature of these systems.

Before 2012, the problems related to “Perception” such as classification, object detection were solved using algorithms with manually engineered features. However, in recent years, instead of manually engineering the features, these features are learned through learning algorithms. The game-changing architecture of Convolution Neural Networks proposed in 2012 by Alex K [1], provided a tremendous momentum in the direction of pushing Neural networks for perception. This thesis is an attempt to develop a convolution neural network architecture for embedded systems, i.e. an

architecture that has a small model size and competitive accuracy. Recreate state-of-the-art architectures using fire module's concept to reduce the model size of the architecture. The proposed compact models are feasible for deployment on embedded devices such as the Bluebox 2.0. Furthermore, attempts are made to integrate the compact Convolution Neural Network with object detection pipelines.

1. INTRODUCTION

The Convolutional Neural Network (CNN) has completely revolutionized the "Perception" domain. It has proven to be a dominant technology in tasks such as image classification and object detection. Convolutional neural network surpasses the performance when compared to the existing algorithms like SIFT, HOG, etc. in terms of accuracy and detection time. In convolution neural network, instead of manually engineering the features, supervised learning helps to learn these features through learning algorithms and optimization techniques. Though the convolutional neural networks are a great tool to attack "Perception" problems, it is computationally and memory intensive. Due to high computational resources required for deploying the convolution neural network, it is challenging to deploy such models on embedded devices with memory and power constraints. Memory and power constrained device requires architecture with a small model size for deployment. Deeper architectures demand more computational resources than the shallow network. Hence it is vital to develop a deep architecture with low computational cost with competitive accuracy. Thriving research in the field of Design Space Exploration (DSE) of neural networks for developing compact architectures for deployment on memory constrained or embedded devices have made it feasible to deploy a convolution neural network on constrained devices. These advancements have led to accelerated growth in the design space exploration of architectures, where the focus is on developing architectures with a less number of parameters.

SqueezeNet [2] is one such architecture, and the paper provides some great insight into developing an architecture with compact model size. Also, deeper network with multiple hidden layers such as RESNET [3], SqueezeNet [2], VGG16, VGG19 [4] and developing new optimization techniques, various training methodology and implementation of non-linearity like ReLU [6], ELU [7], to tackle problems of vanishing

gradients, has substantially aided in developing and training deeper neural network for image recognition or object detection challenge. The increasing complexity of neural network has also led to accelerated development of various hardware architectures like graphics processing unit, tensor processing unit and large-scale distributed deep networks [8], which uses parallel architecture and multiple computation units for example boards such as S32V234, BlueBox (embedded systems) by NXP and NVIDIA's TITAN, TESLA, GTX 1080 (GPUs) and Jetson TK1 (embedded systems) are widely used for deploying or accelerating training process of various deep Convolutional Neural Networks. Deep Convolution Neural Networks have become the focus of "computer vision" field to improve the performance [1]. Currently, all the state-of-the-art model have a similar underlying structure. As discussed earlier, having a deep convolutional neural network has better performance, but at the same time, it is computationally expensive. Hence, to counter this issue, the concept of squeezing the network and controlling the dimension of activation maps can be used to develop an architecture with competitive accuracy and compact model size. This compact model can be further used for deployment on embedded systems with memory and power constraints.

One of the techniques to design a compact architecture is the fire modules used in SqueezeNet [1]. This approach controls the dimension of activation maps and reduces the number of parameters by using a convolution filter of kernel size 1×1 , instead of convolution filter of size 3×3 . The implementation for fire modules in the architecture compresses the architecture and saves memory. This technique also helps to reduce the inference time of the network. Tasks such as object detection and image classification are required to be implemented in real-time for application such as autonomous driving, in such application inference time or detection time is very critical to safety a lag due to a computationally expensive model can have a detrimental effect. Hence reducing the model size is critical, this reduces the computational time and in turn, reduces inference time and can help to develop real-time applications.

1.1 Motivation

As discussed earlier, classification, detection, and segmentation are challenging tasks for a computer. Nevertheless, there are computer vision algorithms that are capable of image classification, object recognition, face detection these algorithms are fundamental to build applications for surveillance, autonomous cars, UAVs and various other embedded systems. As the autonomy of the system increases so is the requirement to develop an algorithm with intuitions. An algorithm with a hint intelligence, so to say. Past few years have been the golden era for computer vision with significant progress made in developing more sophisticated computer vision system. Machine Learning supersede the historical approaches like manual feature engineering and complex hard-coded algorithms, where rather than just searching the image for hard-coded features, the computer tries to learn the features looking at the images during the training process. This approach is similar to a child learning to recognize different objects by going through the examples.

Due to the powerful and large computing platform, it has become easy to store and compute large dataset for the training of such algorithms. The advanced learning algorithms, training deep architectures with multiple layers on the massive data and powerful computing platform is referred to as deep learning. Deep learning is a subset of machine learning.

The exponential increase in hardware development and computing platforms has shaped and accelerated the research areas related to computer vision. As discussed before concepts of deep learning are replacing historical approaches like manual feature engineering, and complex hard-coded algorithms, where rather than just searching the image for hard-coded features, the model tries to learn the features looking at the images.

The Convolution Neural Network consists of multiple layers that learn high-level and low-level features in the image propagated through the network. High-level features are those features that concern with finding shapes and objects in the image

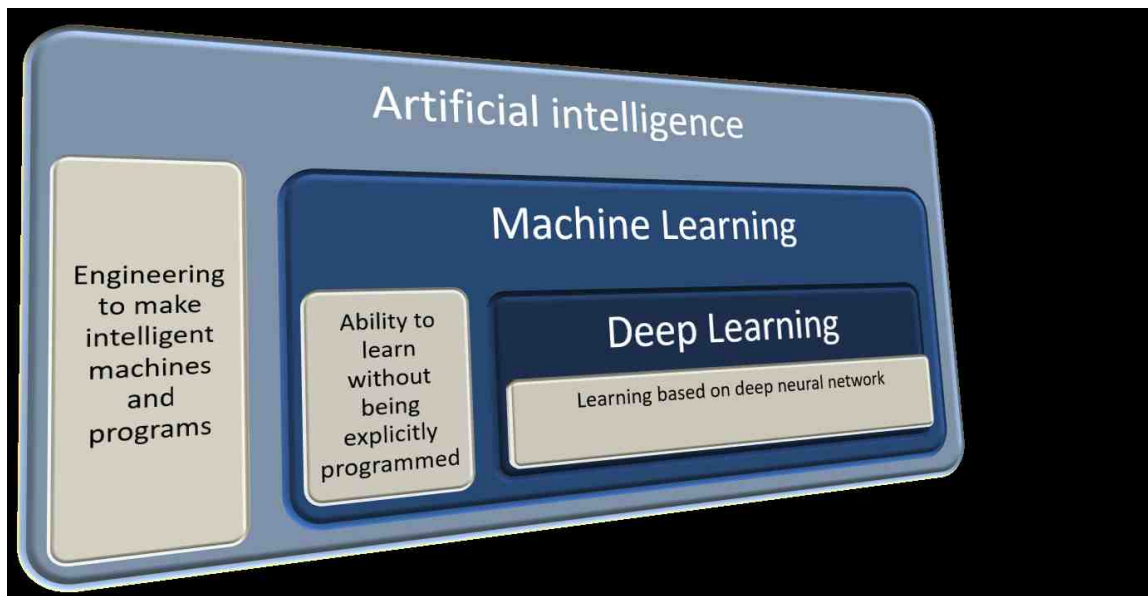


Fig. 1.1. Deep Learning a subset of Machine Learning

and is based on low-level features. Whereas, low-level features are descriptor extracted from an image containing information about visual properties. These layers are adapted and optimized by using various optimization techniques used in deep learning [8]. Though the neural network has been around for several years, there has been tremendous growth in using Convolution Neural Network after 2012. Accelerated research in the past few years has led to the development of state-of-the-art network architecture for Image Classification. This state-of-the-art model rivals the accuracy of a human for classification tasks.

However, real-time deployment of these model becomes a challenge when model size and the number of computations are in the order of billions. The tasks like classification can take considerable time. Though the power of state-of-the-art graphics processing units can be used to implement these tasks, there is a growing need to embed these applications on devices with power, size and memory constraints. Therefore, there is a need to compress the model size while maintaining the competitive accuracy of CNNs.

1.2 Contribution

This thesis examines the Design Space of convolution neural network, benchmark the proposed network architecture on the CIFAR-10 dataset and modify the CNN architecture to reduce the model size while maintaining a competitive level of accuracy. This new architecture further integrates with the object detection pipeline for real-time object detection. Furthermore, the trained CNN and the object detector were deployed on autonomous development platforms such as Bluebox 2.0 by NXP using RTMaps remote studio software.

This thesis also aims to discuss well established deep convolution neural networks like VGG-16 [4] and recreate similar architecture using fire modules [2] to compress the network. This approach leads to a model which has a competitive accuracy and a compact model size. VGG-16 has 16 hidden layers, and these hidden layers consist of convolution filters with a kernel size of 3x3, pooling layer 2x2 and three fully connected layers at the end of the network, this model has a model size of 528 MB for the ImageNet dataset and 385 MB for the CIFAR-10 dataset. The difference in model size is due to less number of classes in the CIFAR-10 dataset, so the output layer has a lesser number of parameters to learn. The approach followed provides a squeezed version of VGG-16 with a model size of 12.9 MB, which is a 96% reduction in model size and reduces the inference time of the model. The model architecture trained had the same number of convolution layers as VGG-16, but the fully connected layers were removed from the network architecture to reduce the weight of the model. Reduction in model size also leads to less overhead when uploading new models to an embedded device many companies in the domain of autonomous driving need to update the model wirelessly to the end device this has improved the reliability and safety of autonomous driving systems [1]. The design target is to reduce the model size using compact convolution filters, keeping in mind the competitive accuracy of VGG16. As a result, two network architectures were developed the ReducedSqNet

with model size of 1.6MB and 2.9MB and the Squeezed CNN with model size of 12.9MB and benchmarked on the CIFAR-10 dataset.

2. OVERVIEW OF CONVOLUTION NEURAL NETWORK

This chapter introduces the concept of the Neural Network and the Convolution Neural Networks and gives a brief overview of the various architectures available. Since the first paper written in the field neural network to gain a mathematical understanding of biological neurons, this field has observed multiples waves of research. The paper by Mc Culloch and Pitts [10] "*A Logical Calculus of the ideas immanent in nervous activity*", was an early attempt to understand the mathematics behind biological neurons. The working of neurons in the brain was explained using a network of simple electrical circuits.

Definition of Artificial Neural Network:

"A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." – **Dr Robert Hecht-Nielsen**

Biological nervous systems inspire the neural network. There are approximately 80-90 billion neurons connected by $1e14$ - $1e15$ synapses. The basic unit of computation in a neural network is neuron also called nodes. neurons have a highly parallel structure and very complex inter connectivity. Each neuron receives an input signal at its dendrites of another neuron via synapses. These synapses transfer information from one neuron to the other by either amplifying the signal or attenuating is. These of neurons with the simple capability of amplifying or attenuating the signal when clubbed together form a very complex architecture which enables the human body to see, hear, move, remember etcetera.

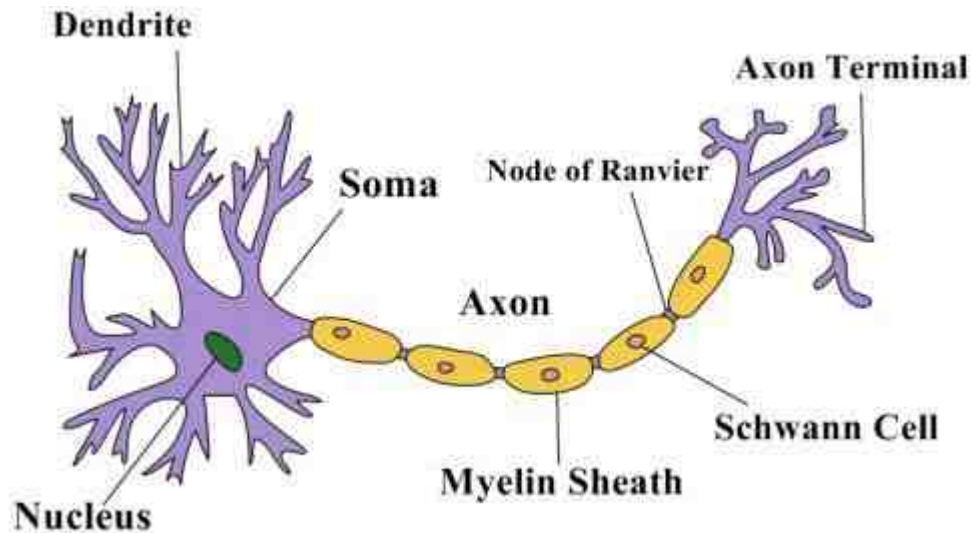


Fig. 2.1. Biological Neurons [11]

There is a decade of research available to replicate what the human nervous system is capable of, and engineers have come up with a model that mimics the neurons. Artificial neurons are loosely biologically inspired. This forms a fundamental building block artificial neural network similar to the biological neuron artificial neural network receives input signals X_i (*Input Signal*) from other neurons and these input signals multiplied by the weights W_i (*Synaptic Weights*) and added with a bias b_i (*Bias*) term finally the output passes through a non-linear function or activation functions. The following equation gives the output of the neurons.

$$Y_i = \sum W_i \times X_i + b_i \quad (2.1)$$

W_i is a factor that decides neurons reaction to the given signal. These weights are analogous to slope in linear regression. Weights are the learnable parameter which can be optimized using various numerical methods like Newtons method, Gradient descent, Adam optimizer to model a dataset. *Activation functions* of a node defines the output of that node given the input. Activation functions are a significant factor

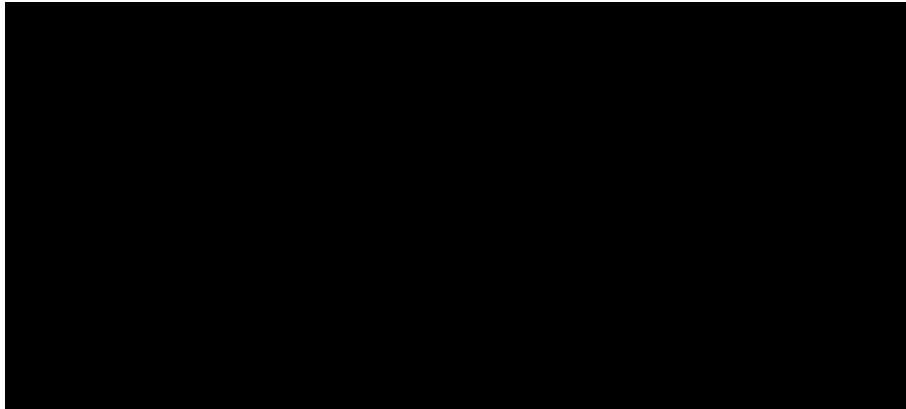


Fig. 2.2. Artificial Neural Network

to be considered while developing a neural network architecture. It is the activation function that helps to model non-linearity in the dataset. Without activation function, the entire network would be just an affine function, and a linear function is incapable of modeling complex data.

2.1 Network and Architecture

Interconnection of multiple artificial neurons forms a network. Usually, connected in a directed acyclic graph to form a neural network. Why is network necessary? An artificial neuron is capable of approximating simple dataset. When it comes to more complex and high dimensional datasets, it is required to have a collection of neurons to model the complexities and to have a better approximation. There are various types of neural network architecture used today to tackle specific problems. Applications like natural language processing, speech to text processing use different types of neural networks such as Feedforward network, Single layer perceptron, Multi-layer perceptron, Convolutional Neural network. Some Neural networks are cyclic and are called Recurrent Neural Network. The most popular network is RNN currently being used is LSTM (Long short-term memory). The focus of this thesis is convolution neural networks, and other architectures are out of the scope of this report.

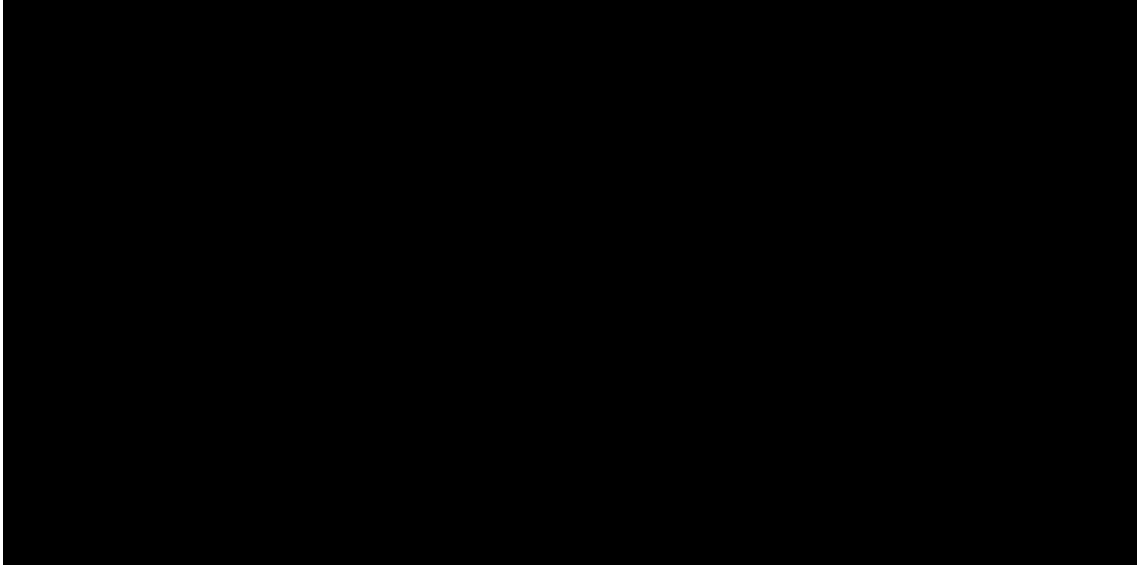


Fig. 2.3. Multi Layer Perceptron

There are a few terminologies used while discussing artificial neural networks. ***Input layer***: This layer consists of nodes which transfers the information to the hidden layers. ***Hidden layer***: This layer performs computation and transfers the information to the next layer that can be either hidden or output layer. ***Output layer***: This layer is responsible for the final output. To this layer activation functions are applied e.g. SoftMax.

2.2 Convolutional Neural Networks (CNN)

Convolution Neural Networks are a deep feed-forward artificial neural network, commonly used for image processing. The concept is slightly different from the fully connected network. In a fully connected network, every neuron in the previous layer is connected to every other neuron in the next layer whereas, the convolution layer takes into account the spatial resolution and takes advantage of the fact that the input is an image. This underlying assumption helps to reduce the number of parameters as the neuron from the previous layer are not connected to all the neuron in the next

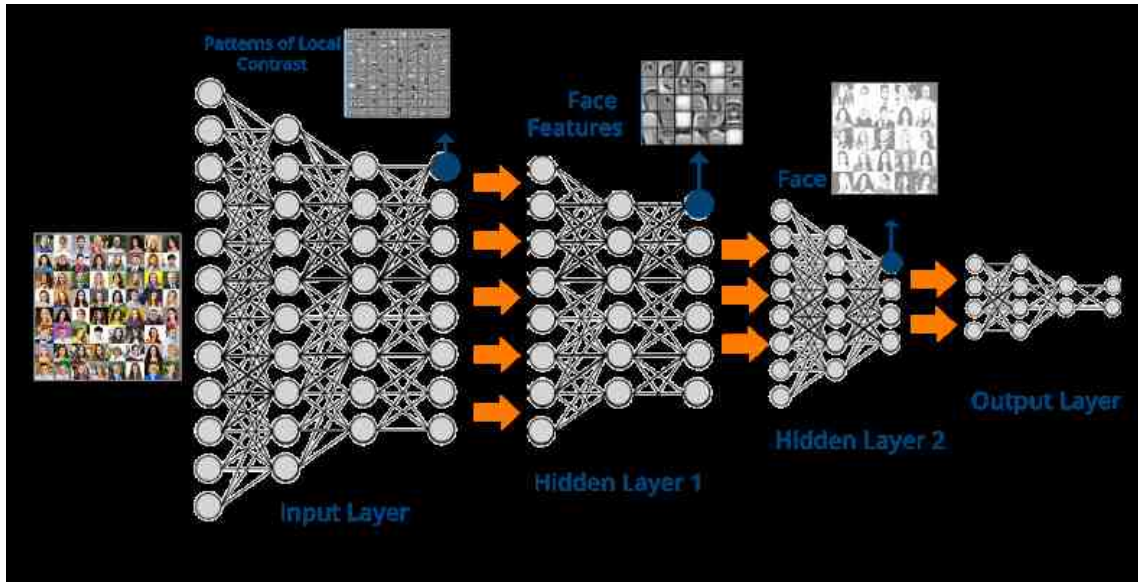


Fig. 2.4. Convolutional Neural Network [12]

layer. Due to high dimensional input, it is futile to have a fully connected layer. Instead, we look at the spatial information of the image, i.e., the receptive field. This receptive field is also called the kernel size. Convolutions performed in a CNN are the dot product of filters with local regions in the input.

In a CNN there are three hyperparameters that control the performance of the convolution neural network such as: *depth*, *stride* and *zero padding*. **Depth** provides the output volume of the convolution layer. It corresponds to the number of filters that the model learns. These filters learn specific features to provide an activation map. **Stride** is a value which decides how the filter slides through the image while performing kernel convolution. **Zero Padding** allows us to control the dimension of the output by padding zeros. The output dimension can be controlled using the following relation.

$$OutputDimension = (W - F + 2P)/S + 1 \quad (2.2)$$

Where, W is the input volume dimension, F is the filter size, P is the number of zero padding to be used and S refers to the stride used. Let us assume we have

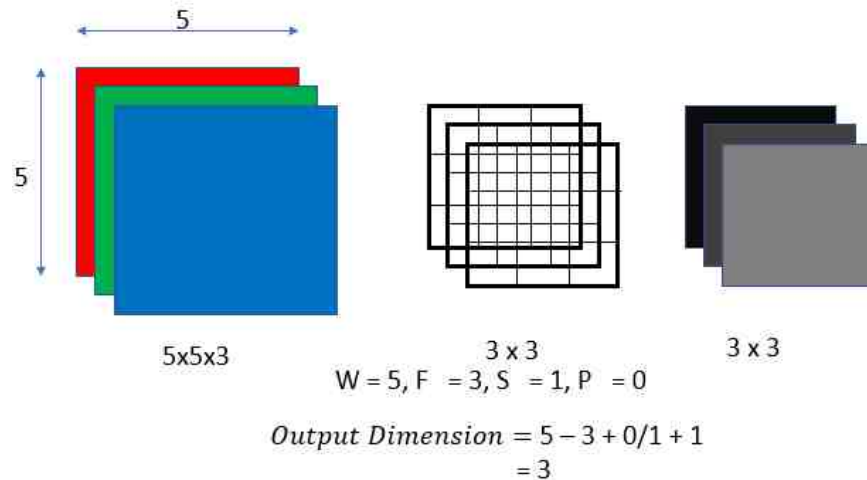


Fig. 2.5. Represent the output dimension calculation based on Eqn 2.2

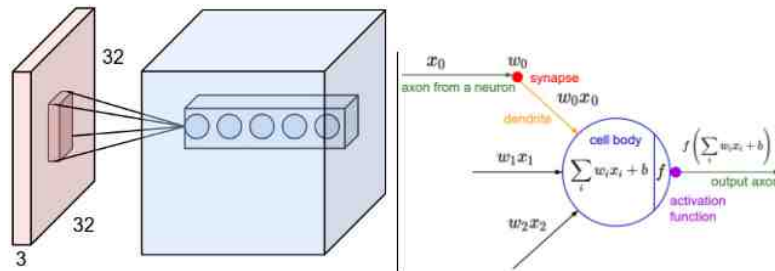


Fig. 2.6. Convolution Kernel and Receptive fields [18]

$W = 5, F = 3, P = 0, S = 1$ in Fig. 2.5, On the left, we have an example of input volume in red (e.g., a $32 \times 32 \times 3$ CIFAR-10 image), and an example volume of neurons in the first Convolution layer. Each neuron in the convolution layer is connected only to a local region in the input volume spatially but to the full depth (i.e., all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input - see discussion of depth columns in the text below. On the right, we have The neurons from the Neural Network which compute

a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially [18].

2.2.1 Number of Parameters Calculation

The number of parameters is a significant factor to keep in mind while designing a CNN. As model size and processing time is directly proportional to the number of parameters. Hence it is critical to constrain the number of parameters in a model.

$$\text{Inference Time \& Model Size} \propto \text{Number of Parameters} \quad (2.3)$$

As number of parameters plays an important role in developing a compact model. It is important to compute the number of parameters for a convolution layers. Number of parameters are calculated based on dimension of activation map and kernel size of the filter for e.g if the input layer contains $5 \times 5 \times 3$ number of neurons and the kernel size is 3 and number of filter is 3 we need $3 \times 3 \times 3 \times 5 \times 5 \times 3 = 675$ number of parameters. The assumption in the example is that, the input image size is 5×5 and channel depth is 3 but in real life situations the image size is in the order of 227×227 with channel depth of 3 if the same kernel size of 3 is used the total number of parameter explodes to $3 \times 3 \times 3 \times 227 \times 227 \times 3 = 4173849$.

2.2.2 Pooling

In the CNN for downsampling the activation map, as it propagates through the network, requires pooling layers. There are various types of pooling layer the most commonly used is Max Pooling and Avg Pooling. The following figure is a good representation of how pooling layer works.

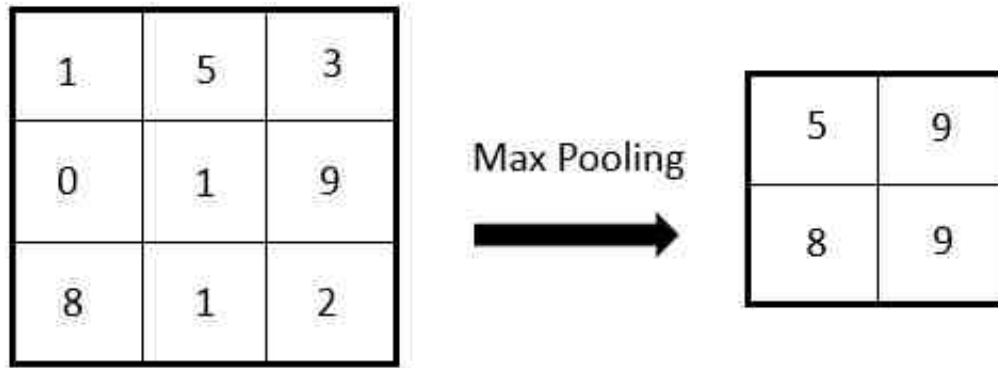


Fig. 2.7. Max pooling using 2×2 Kernel

2.2.3 Non Linearity or Activation Function

The non-linearity function or the activation functions are used when an affine function of input variables cannot model the output. Activation function helps to provide a better model. There are various activation functions used in CNN, but the most popular is rectifying linear unit (ReLU).

$$F(x) = \max(x, 0) \quad (2.4)$$

3. OBJECT DETECTION

The task of classifying images and estimating the location of an object constrained in the image is called object detection. The most fundamental problem of computer vision is object detection. Object detection can provide a semantic understanding of images and videos. The pipeline of object detection model has three stages: Informative region selection, feature extraction, and classification.

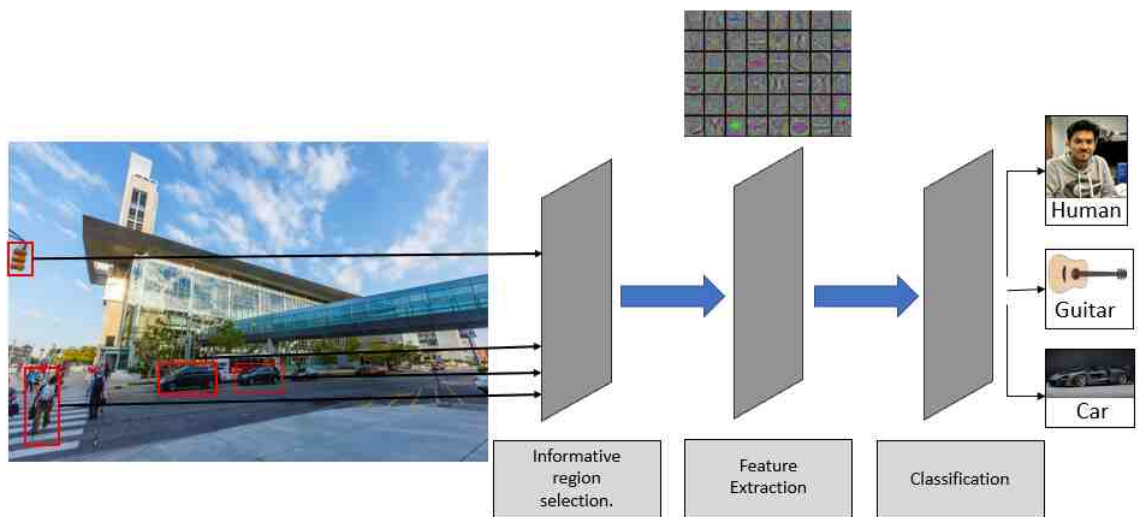


Fig. 3.1. Object Detection Pipeline

3.1 Informative Region Selection

As the image propagates through the layers of Convolutional Neural Network, the objects in the image may appear at any position. Using a sliding window technique on the entire image and search for features is intuitive. This approach is computationally costly and produces many redundant windows [13]. As sliding window techniques

using a scalable window is a brute force algorithm developing efficient algorithm or techniques to select Informative region or region of interest is instrumental in reducing the computational complexity. In architectures such as the faster R-CNN the region Proposal network is responsible for generating proposals. A concept of anchors is used to reduce the time complexity of the architecture.

3.2 Feature Extraction

Recognition of objects in the image requires low-level and high-level features that can provide a semantic representation [13]. Before the advent of convolution neural networks these features were manually engineered, e.g., Haar-like features [14]. However, with optimization techniques and learning algorithms, these features can now be learned by convolution filters.

3.3 Classification

Due to the presence of multiple objects in the image, a classifier is used to predict class probabilities and differentiate between objects. The classifier uses models like the support vector machine [15], Deformable part model [16] and AdaBoost [17]. Also, convolution neural networks due to its performance are quite popular choice to perform classification tasks.

3.4 Deep Neural Network for Object Detection

Since 2012 when Alex Kizhevsky et al. proposed the convolution neural network [1], the field of the deep neural network has experienced accelerated growth. Convolution neural network reduced the top 1 and top 5 classification error to 37.5% and 17% respectively on ImageNet [1] challenge. In 2012, stacked against the traditional approaches of image classification, the AlexNet outperformed the traditional approaches with a top 5 error rate of 15.3% vs. 26.2% [1]. These results have proved

the dominance of Convolution Neural Network to attack image classification tasks. Historically, developing and training the architecture has also been dependent on hardware architecture. The advances in hardware architectures and software frameworks have aided in the training of the deep neural network architecture. There has been a tremendous increase in compute capability of graphics processing units like Tesla, GTX 1080 and tensor processing units by Google. These advances have drastically reduced the training time for deep neural networks. Figure 3.2 and 3.3 show the boosts in software frameworks and GPU performances.

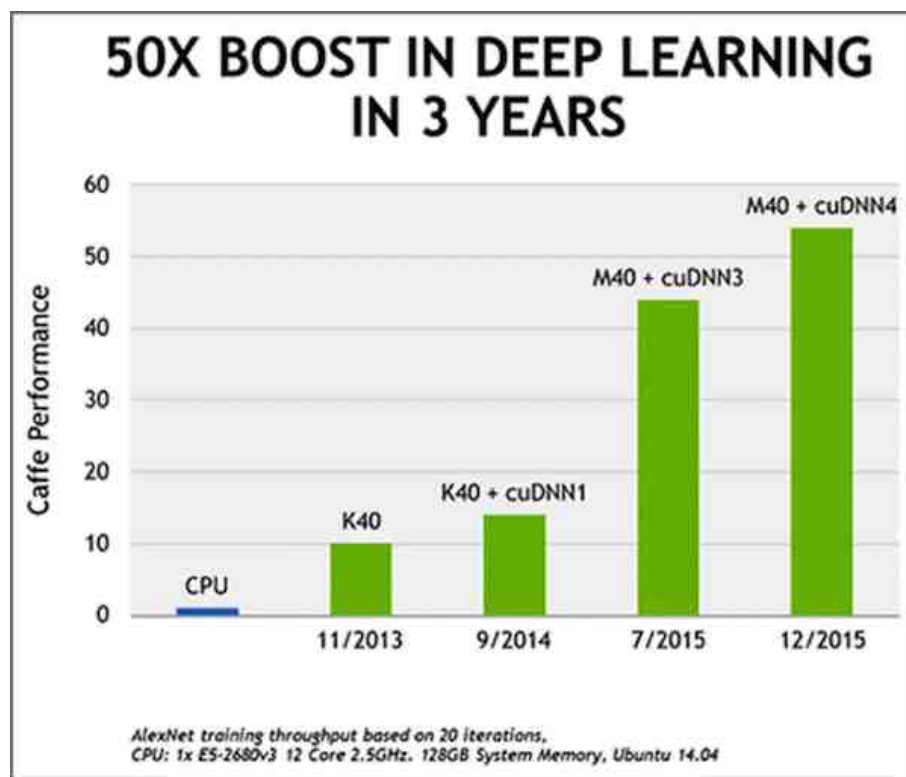


Fig. 3.2. Boost in Deep learning [19].

These advancements have substantially helped developing object detection algorithm based on deep neural networks. Object detection has two types of frameworks region proposal based and regression/classification based. Figure 3.4 helps to put

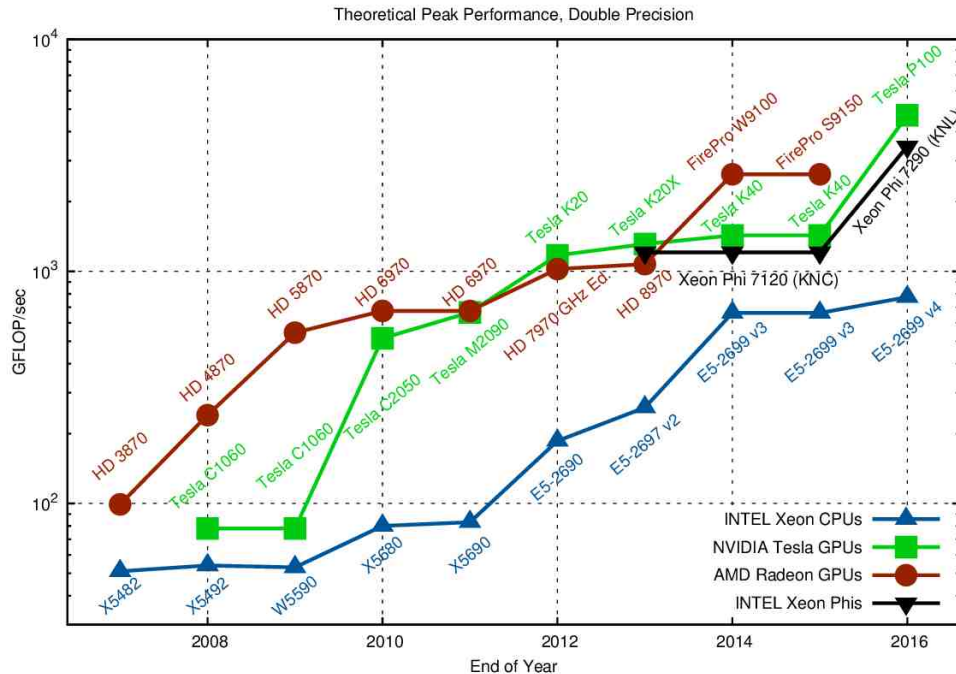


Fig. 3.3. Boost in GPU performance [20].

things in perspective the evolution of object detection framework. These frameworks have their pros and cons for, e.g., Region Proposal based algorithm has better accuracy but high computational cost when compared to box regression-based techniques. Whereas the regression/classification-based algorithm has low computational time but low accuracy. As observed there is a trade-off between accuracy and computational time and the choice depends on the developer to give precedence to the model accuracy or faster inference time. In the case of embedded system development which is the focus of this thesis, the precedence is to having a low computational cost and competitive accuracy.

Motivated to develop an object detection algorithm for real-time Autonomous embedded systems. We discuss two models from two different frameworks Faster R-CNN a region proposal-based architecture and YOLO a regression/classification-based architecture. Both the architectures are modified by replacing the CNN with a

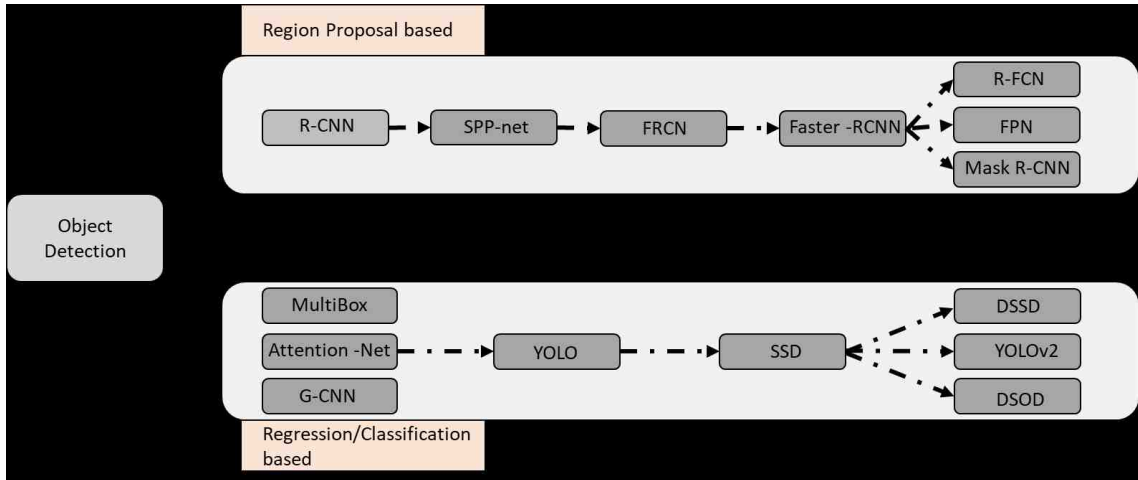


Fig. 3.4. Two train of thoughts for object detection are represented in the figure is inspired by [13] SPP-net [21], FRCN [22], Faster-RCNN [23], R-FCN [24], YOLO [25], SSD [26].

squeezed version of CNN created using fire modules to reduce the model size. These architectures are compared with its baseline architectures concerning computation time and accuracy. These results are discussed later in the chapter.

3.4.1 Faster R-CNN

Faster R-CNN consists of two networks, region proposal network and a convolution neural network. Region proposal network is responsible for proposing the regions or the bounding box whereas the CNN is responsible for the classification. The faster R-CNN architecture makes use of feature maps to generate region proposals. As this approach reuses the feature map generated through CNN and does not use a brute force technique such as the sliding window the computation cost is low. The further section discusses the components of faster R-CNN.

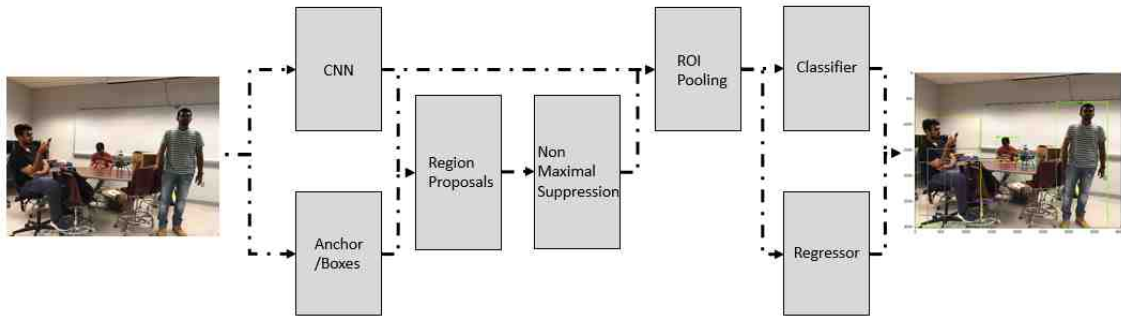


Fig. 3.5. Faster R-CNN Architecture [23]

Anchor:

The anchor is the most critical components in faster R-CNN. Anchors are boxes in faster R-CNN configuration with different aspect ratios of 1:1, 1:2, 2:1. It is a very straightforward approach to train a neural network with four output X_i , Y_i , H_i , W_i to detect the bounding box of the object but this approach fails when there are multiple objects in the image. This problem can be resolved by running the anchors at each spot on feature maps generated by CNN. Measure for the object detection accuracy is Intersection Over Union (IOU), so when the IOU of a specific anchor and ground truth label have a large intersection, the regression associated with that anchor provides fine-tuned bounding box.

Region Proposal network:

The feature learned from the CNN is propagated through the Region Proposal Network. The RPN has 2 objectives; firstly, it is responsible for providing two class scores, for the object present or not present for each anchor. Secondly, it is responsible for predicting the bounding box coordinates for the object present in the image for each anchor, e.g., if the input feature map is 100×100 . RPN generates $2 \times 9 \times 100 \times 100$

class scores for whether the object is present or not and also generate $4 \times 9 \times 100 \times 100$ coordinates for bounding boxes X_i, Y_i, H_i, W_i .

$$L_{loc}(t^u, v) = \sum Smooth_{L_i}(t^u - v_i) \quad (3.1)$$

Where,

$$Smooth_{L_i}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases} \quad (3.2)$$

The above formula is the loss function for regressors [23].

The Mighty Classifier:

The classifier is the essential part of the Faster R-CNN network, and The CNN is responsible for extracting features from the image and classifying the image. This part of faster R-CNN can be modified, and various other CNN architectures can be used to extract features for RPN. This chained network of CNN and RPN can be trained jointly to classify images and predict bounding box. This architecture provides the flexibility of using custom-made CNN architectures with the existing RPN framework.

Region Of Interest Pooling:

Object detection pipeline uses the region of interest pooling layer. For finding out the region of interest in the images, it takes a section of input feature map and scales it to some predefined size. This fixed size is achieved by dividing the region proposal into equal-sized sections (the number of which is the same as the dimension of the output), finding the largest value in each section and copying these max values to the output buffer [27]. The reason for doing this is since ROI layer is preceded with RPN layer which can provide regions proposal with varying dimensions ROI layer is required to convert it into a fixed dimension tensor. This is because the fully connected layer after ROI layer expects a fixed size input tensor.

3.4.2 YOLO

Another common architecture in object detection pipeline proposed by Redmon et al. [25]. YOLO stands for You Only Look Once. In YOLO the CNN is responsible for classification and predicting the bounding box. YOLO poses the object detection as a single regression problem [25]. There exist no region proposal networks. The input image is divided into $S \times S$ grids, and each grid cell predicts N bounding boxes with classification scores. Scores in YOLO is defined as $Pr(Object) \times IOU_{Pred}^{Truth}$ where $Pr(Object)$ is the softmax class possibilities and IOU_{Pred}^{Truth} is the intersection over the union of ground truth boxes and predicted boxes. Apart from these the C class probabilities is predicted for each grid. The only grid containing an object is calculated not the background. Since YOLO uses a CNN to compute both the bounding boxes and the class scores it computationally very efficient but at the same time it is not as accurate as Faster R-CNN.

Loss Function:

Classification Loss:

$$\sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (3.3)$$

$1_i^{obj} \rightarrow 1$ if the object appears in cell i, else it is 0

$\hat{p}_i(c) \rightarrow$ denotes conditional class probability for class c in cell i

Localization Loss:

$$\begin{aligned} & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{i,j}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{i,j}^{obj} [(\sqrt{W_i} - \sqrt{\hat{W}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \end{aligned} \quad (3.4)$$

$1_{i,j}^{obj} \rightarrow 1$ if the j^{th} boundary box in cell i is responsible for detecting the object, otherwise 0.

λ_{coord} weight factor for penalizing the co-ordinate loss function.

Confidence Loss:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{i,j}^{noobj} (C_i - \hat{C}_i)^2 \quad (3.5)$$

Confidence loss function when no object is detected in the box.

$$\sum_{i=0}^{S^2} \sum_{j=0}^B 1_{i,j}^{obj} (C_i - \hat{C}_i)^2 \quad (3.6)$$

Confidence loss function when object is detected in the box.

$\hat{C}_i \rightarrow$ is the box confidence score in cell i .

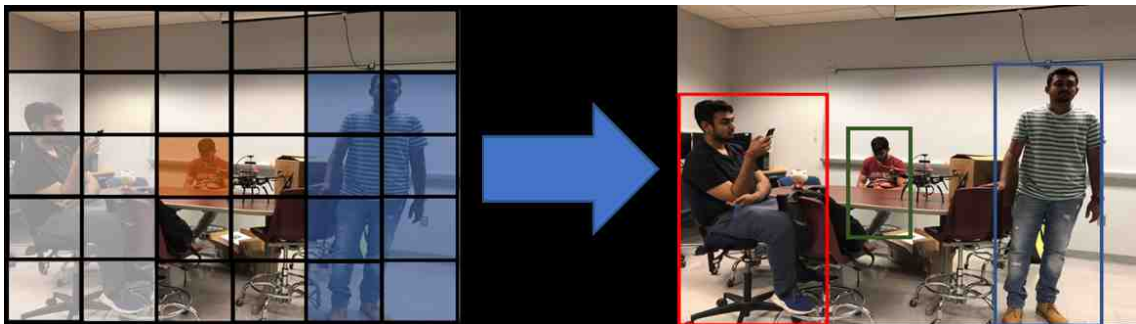


Fig. 3.6. Bounding boxes generated using YOLO detection.

3.4.3 Motivation for Object Detection

There are various fields of research in the deep learning domain. This thesis explores the Network optimization path [13] using compact convolution filters. In deep learning, it is essential to strike a balance between accuracy and inference time which is dependent on model size [28][29]. Though the accuracy of the model is an essential factor, it is essential for real-time systems to learn compact models with a

fewer number of parameters [30]. This balance between accuracy and model size can be achieved with multiple approaches one of them being using fire modules to create a network with a lower number of parameters. Also, in 2018 Akash Gaikwad et al. also achieved good results by pruning the network based on Taylor expansion based criterion [31]. This can be further used in series with the approach developed in this paper to obtain a compact model for hardware deployment.

4. BLUEBOX 2.0

The major challenge in porting an algorithm on the vision system is to optimally map it to various units in the system to achieve an overall boost in the performance. This boost in the performance requires an intimate knowledge of the individual processors, their capabilities, and limitations. For example, APEX processors are highly parallel computing units, with Single Instruction Multiple Data (SIMD) architecture, and can handle data level parallelism quite good. One of the significant requirements of this thesis is to analyze the capability of NXP Bluebox 2.0 (BLBX2) as an autonomous embedded system for real-time applications. Bluebox is one of the development platforms designed for the advanced driver assistance system feature for autonomous vehicles. The bluebox development platform is an integrated package for creating applications for autonomous driving and is comprised of three independent systems on chip (SoCs: S32V234, LS2084A, S32R274).

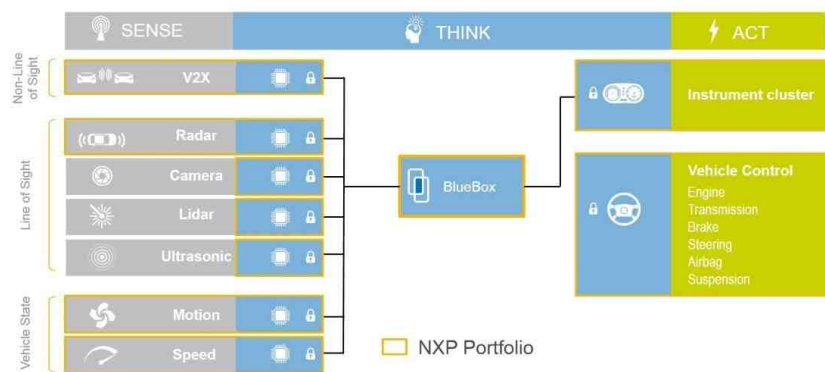


Fig. 4.1. Hardware Architecture for Bluebox 2.0 [Pic Courtesy NXP].

functionality of image conditioning allowing to integrate multiple cameras. It also contains APEX-2 vision accelerators and 3D GPU designed to accelerate computer vision functions such as object detection, recognition, surround view, machine learning and sensor fusion applications. It also contains four ARM Cortex-A53 core, an ARM M4 core designed for embedded related applications.

The processor can operate on the software such as Linux Board Support Packages (BSP), the Linux OS (Ubuntu 16.04 LTS) and NXP vision SDK. The Processor boots up from the SD card interface available at the front panel of the bluebox. A complete overview of the S32V234 processor is shown in figure 4.3.

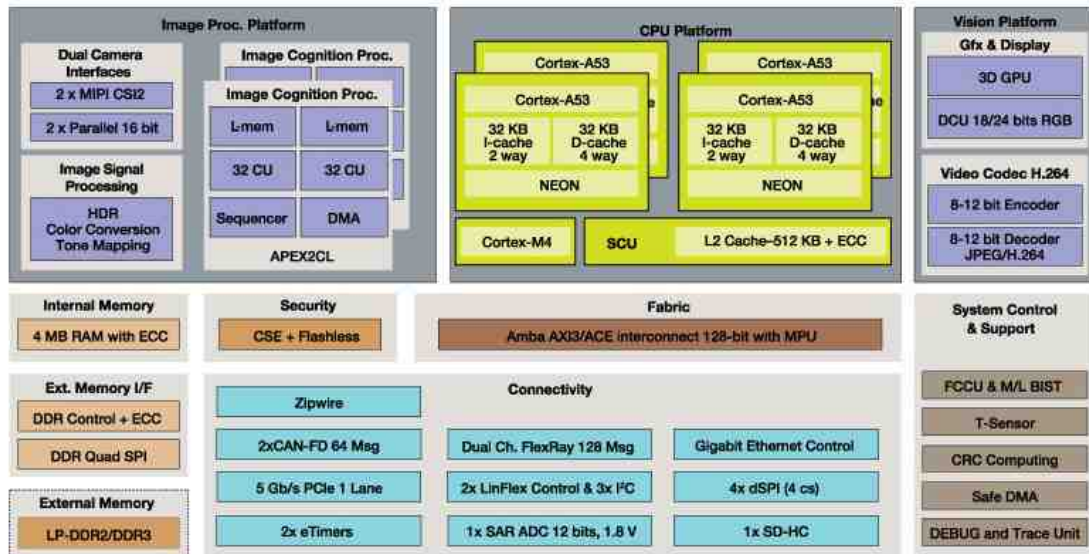


Fig. 4.3. Hardware Architecture for S32V234 [Pic Courtesy NXP].

4.2 LS-2084A

The LS2 processor in the BLBX2 is a general-purpose high-performance computing platform. The processor consists of eight ARM Cortex-A72 cores, 10Gb Ethernet ports, supports a high total capacity of DDR4 memory, and features a PCIe expansion

slot for any additional hardware such as GPUs or FPGAs, thus making it especially suitable for applications that demand high performance or high computation, or support for multiple concurrent threads with low latency. In addition to being suitable

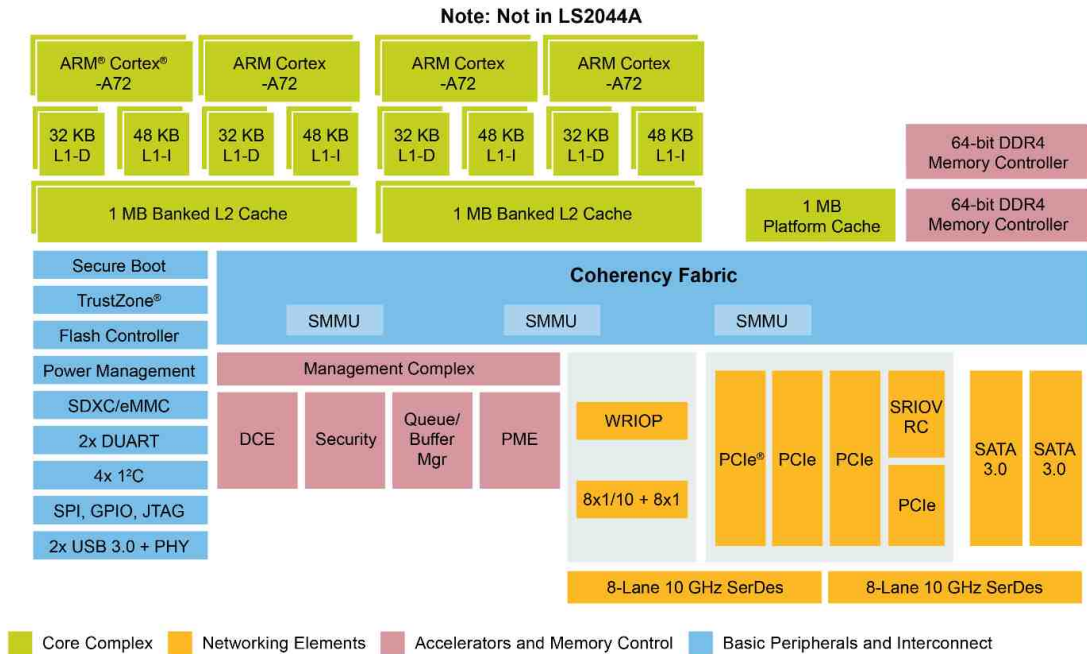


Fig. 4.4. Hardware Architecture for LS2084A [Pic Courtesy NXP].

for high-performance computing, the LS2 is also a convenient platform to develop the ARMV8 code. The LS2 is connected to a Lite-On Automotive Solid State Drive via SATA, to provide large memory size for software installation, it also consists of SD card interface which allows the processor to run: Linux Board Support Packages (BSP), the Linux OS (Ubuntu 16.04 LTS) as OS

4.3 Platform Overview

For this thesis, the software enablement on the LS2084A and S32V234 SoC is deployed using the Linux board support package which is built using the Yocto frame-

work. The LS2084A and S32V234 SoC are installed with Ubuntu 16.04 LTS which is a complete, developer-supported system and contains the complete kernel source code, compilers, toolchains, with ROS kinetic and Docker package. Figure 4.5 represents the platform overview.

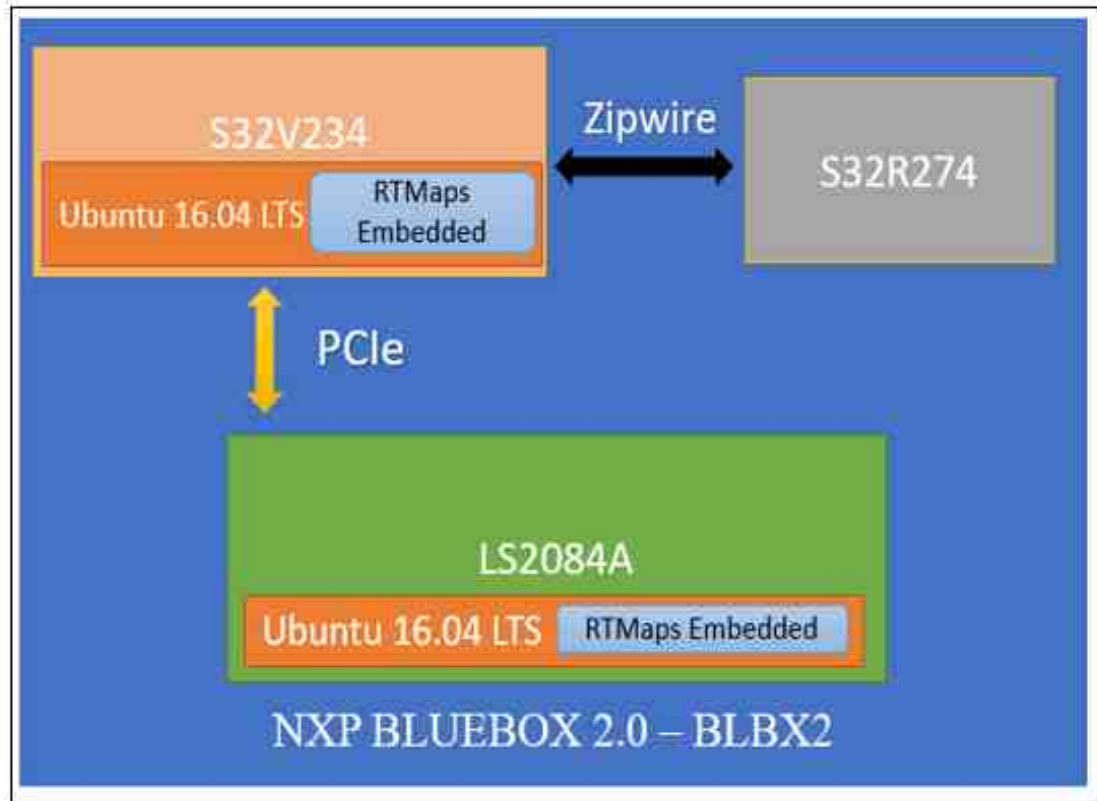


Fig. 4.5. System overview of NXP BLBX2.

5. REAL-TIME MULTI SENSOR APPLICATIONS (RTMAPS)

RTMaps: RTMaps is designed for the development of multimodal based applications, thus providing the feature of incorporating multiple sensors such as camera, lidar, radar. It has been tested for processing and fusing the data streams in the real-time or even in the post-processing scenarios. The software architecture consists of several independent modules that can be used for different situation and circumstance.

RTMaps Runtime Engine: The Runtime Engine is an easily deployable, multithreaded, highly optimized module designed in a context to be integrated with third-party applications and is accountable for all base services such as component registration, buffer management, time stamping threading, and priorities.

RTMaps Component Library: It consists of the software module which is easily interfaceable with the automotive and other related sensors and packages such as Python, C++, Simulink models, and 3-d viewers, etc. responsible for the development of an application.

RTMaps Studio: It is the graphical modeling environment with the functionality of programming using Python packages. The development interface is available for the windows and ubuntu based platforms. Applications are developed by using the modules and packages available from the RTMaps Component library.

RTMaps Embedded: It is a framework which comprises of the component library and the runtime engine with the capability of running on an embedded x86 or ARM capable platform such as NXP Bluebox, Raspberry Pi, DSpace MicroAutobox, etc.

For this paper the RTMaps embedded v4.5.3 platform is tested with NXP Bluebox, it is used independently on the Bluebox, and with the RTMaps remote studio operating on a computer thus providing the graphical interface for the development and testing purpose. The connection between the Computer running RTMaps Remote studio and the Embedded platform can be accessed via a static TCP/IP as shown in Figure.

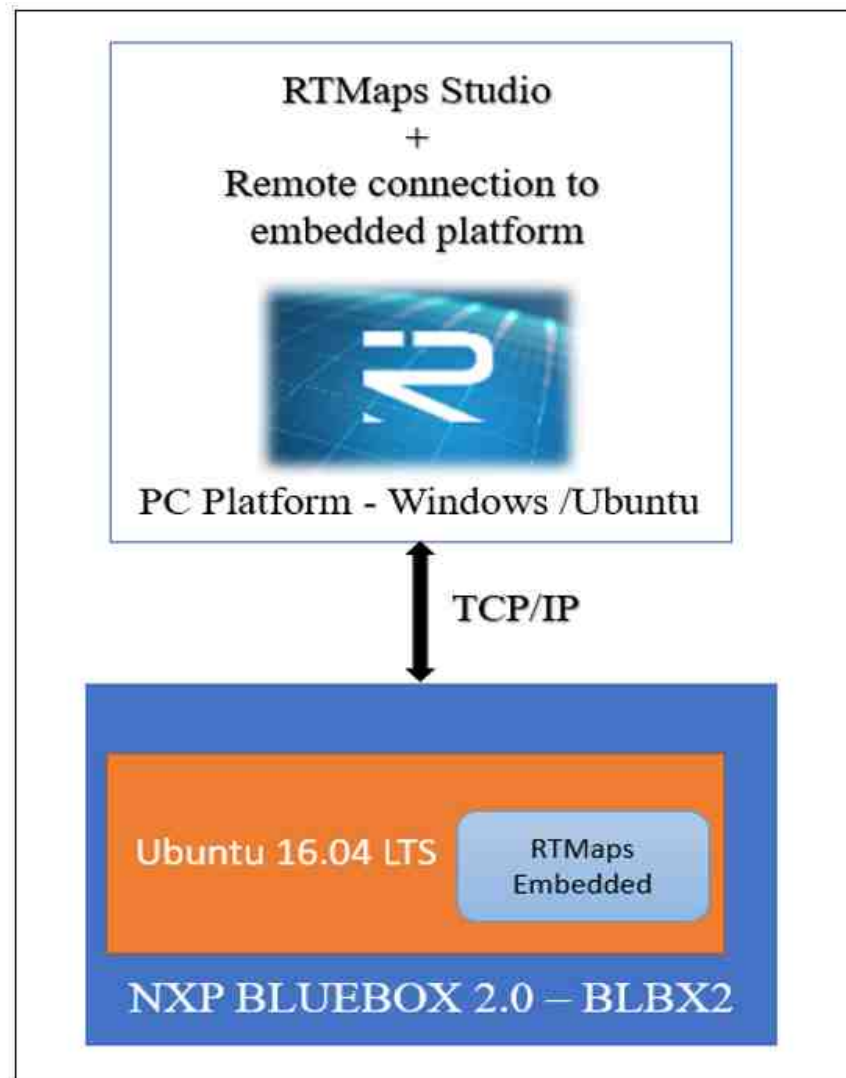


Fig. 5.1. RTMap setup with Bluebox 2.0.

6. REDUCEDSQNET ARCHITECTURAL ADVANCES

As discussed in previous chapters, there is various on-going research in the field of deep learning to name a few, Multi-task joint optimization and multimodal information fusion, scale adaptation, network optimization, network compression. The objective for these researches is to improve on the performance measure of the existing state-of-the-art models. The performance measures are the accuracy or error rate of the model architecture on the test set (the data it has never seen before), computation time (time it takes to make a forward pass) and space complexity (Memory require to store the optimized weights). As these three factors are critical to evaluating the performance of the architecture, striking the right balance between these three performance measures is critical. This chapter discusses the development of a compact architecture proposed in this thesis. ReducedSqNet is an attempt to adapt a state-of-the-art for a specific dataset (CIFAR-10).

6.1 Accuracy

For a task such as classification, accuracy is the performance measure of the deep neural network to test the performance of the architecture. The performance can also be measured by using the error rate, i.e., the number of miss-classifications per batch. Accuracy is measured on the training and test batch. However, using the training dataset to evaluate the final accuracy of the model architecture has absolutely zero advantages. Hence it is a good practice to evaluate the model architecture accuracy with test dataset. Accuracy for a classifier is calculated for the batch size used for testing, e.g., If there are 100 images in a batch and the model classified 70 images correctly, that means the accuracy for the model is 70%. The following chart provides

a good idea about the accuracy calculation for a classifier. The class prediction is calculated taking the argmax of the class score vector.

$$\text{class prediction} = \text{Argmax}(\text{class score}) \quad (6.1)$$

	Ground Truth			Prediction			Cross-Entropy	Accuracy
	CAR	TRUCK	CYCLE	CAR	TRUCK	CYCLE		
Test Image 1	1	0	0	0.33	0.33	0.34	1.109	0
Test Image 2	1	0	0	0.33	0.33	0.34	1.109	0
Test Image 3	1	0	0	0.33	0.33	0.34	1.109	0
Test Image 4	1	0	0	0.99	0.01	0.00	0.01	1
Test Image 5	1	0	0	0.99	0.01	0.00	0.01	1
Test Image 6	1	0	0	0.98	0.02	0.00	0.02	1
Test Image 7	1	0	0	0.98	0.02	0.00	0.02	1
Test Image 8	1	0	0	0.98	0.02	0.00	0.02	1

Fig. 6.1. Accuracy measured on test batch.

6.2 Computation Time

Computation time depend on how fast the network can provide inference from the given input, e.g. if the input image contains a cat how fast would the network be able to infer that the give input image has a cat. The number of parameters is proportional to the computational time and the Multiply And Accumulate Operations (MAAC) in the model. This performance measure is critical for developing a model for low power and low-cost embedded targets. The reason to focus on network optimization using compact convolution filter is that for an embedded device it is essential to have a right balance between accuracy and inference time. Having a highly accurate model but having very high inference time is no good for a real-time system. Therefore, it is essential to work on optimizing the network to reduce the model size and still

maintain a competitive accuracy. The architecture developed in this thesis qualifies both these requirements it is fast and can be deployed on embedded targets, and if tailored for a specific dataset it can outperform the baseline architectures.

Convolutional Factorization

The idea is to divide the convolution kernels into smaller size kernels to reduce the number of parameters and computation time required for generating an output activation map. One method of achieving this is using flattening proposed by Jin et al. (2014) [38]. Flattening turns an original 3D kernel ($N_C \times K_H \times K_W$) into three 1D kernels ($N_C \times 1 \times 1, 1 \times K_H \times 1, 1 \times 1 \times K_W$). This method can reduce parameters by a factor of $(N_C \times K_H \times K_W)/(N_C + K_H + K_W)$ for each output channel results show that, both training and inference processes can be accelerated, and the accuracy can also be better on some datasets. However, the result should be further evaluated on large-scale datasets.

Convolution Module

The idea behind the compact convolution modules is to replace the large convolution kernels by a group of smaller convolution kernels. This approach has no hardware or software dependencies and by far the least complicated solution to reduce the number of parameters in the convolution layer. The major motivation of this approach is to reduce the total number of parameters required for a large convolution layer and increase the computational efficiency for e.g., if the kernel size for convolution layer is $7 \times 7 \times C$ with the input size of $100 \times 100 \times 3$ then the total number of parameters are $100 \times 100 \times 3 \times 49 \times C = 1470000 \times C$ but if the kernel size is constrained to $3 \times 3 \times C$ the total number of parameters is $270000 \times C$.

Strassen's Algorithm

Convolution operation in a CNN boils down to matrix multiplications of input feature maps with convolution kernels to produce an output feature maps. Assume the input feature map has a spatial dimension of $I_H \times I_W$ and channel width of I_D . The filter size of $K_x \times K_x$ with N_k number of filters. The following gives the time complexity of the convolution layer:

$$T(n) = \mathcal{O}(I_H \times I_W \times I_D \times K_x \times K_x \times N_k) \quad (6.2)$$

We can optimize the matrix multiplication techniques using Strassen's algorithm for matrix multiplication. Assume that matrix $Q_{H_i \times W_o}$, and there are 3 channels representing input feature map. Let the $W_{H_f \times W_f}$ represent the kernel size, and $O_{H_o \times W_o}$ matrix represent the output feature map after convolution.

$$O_{H_o \times W_o} = Q_{H_i \times W_i} * W_{H_f \times W_f} \quad (6.3)$$

The output feature map is computed by the following method:

$$\begin{aligned} O_{1,1} &= W_{1,1} \times Q_{1,1} + W_{1,2} \times Q_{1,2} + \dots + W_{1,W_n} \times Q_{1,Q_n} \\ O_{2,1} &= W_{2,1} \times Q_{2,1} + W_{2,2} \times Q_{2,2} + \dots + W_{2,W_n} \times Q_{2,Q_n} \\ &\vdots \\ &\vdots \\ &\vdots \\ O_{O_n,1} &= W_{W_n,1} \times Q_{Q_n,1} + W_{W_n,2} \times Q_{Q_n,2} + \dots + W_{W_n,W_n} \times Q_{Q_n,Q_n} \end{aligned} \quad (6.4)$$

We can represent the convolution as follows:

$$O = W \times Q \quad (6.5)$$

where,

$O \rightarrow$ Output feature map.

$W \rightarrow$ Weight matrix given by filter kernel.

$Q \rightarrow$ Input feature map.

Strassen's algorithm works on simple divide and conquer principle, the assumption is that the W and Q matrix are evenly divisible i.e. the matrix can be divided into equal size smaller matrix.

$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \quad (6.6)$$

$$O_{1,1} = W_{1,1} \times Q_{1,1} + W_{1,2} \times Q_{1,2}$$

$$O_{1,2} = W_{1,1} \times Q_{1,2} + W_{1,2} \times Q_{2,2}$$

$$O_{2,1} = W_{2,1} \times Q_{1,1} + W_{2,2} \times Q_{2,1}$$

$$O_{2,2} = W_{2,1} \times Q_{1,2} + W_{2,2} \times Q_{2,2}$$

to solve the equation 6.6 we need 8 multiplication but using Strassen's algorithm we can reduce the number of multiplication to 7.

$$M_1 = (W_{1,1} + W_{2,2}) \times (Q_{1,1} + Q_{2,2})$$

$$M_2 = (W_{2,1} + W_{2,2}) \times Q_{1,1}$$

$$M_3 = W_{1,1} \times (Q_{1,2} - Q_{2,2})$$

$$M_4 = W_{2,2} \times (Q_{2,1} - Q_{1,1})$$

$$M_5 = (W_{1,1} + W_{1,2}) \times Q_{2,2}$$

$$M_6 = (W_{2,1} - W_{1,1}) \times (Q_{1,1} - Q_{1,2})$$

$$M_7 = (W_{1,2} - W_{2,2}) \times (Q_{2,1} - Q_{2,2})$$

$$O_{1,1} = M_1 + M_4 - M_5 + M_7$$

$$O_{1,2} = M_3 + M_5$$

$$O_{2,1} = M_2 + M_4$$

$$O_{2,2} = M_1 - M_2 + M_3 + M_6$$

Since we have a very large number of multiplication in convolution neural network a single reduction in multiply operation has huge impact on computation time.

6.3 ReducedSqNet

Though the Strassen's algorithm optimizes matrix multiplication, it is a very sophisticated approach, and current deep learning framework does not have support for such algorithms. So the approach of the compact convolution module was used to develop a compact architecture. The architecture developed in this thesis qualifies both of these requirements it is fast and can be deployed on embedded targets, and if tailored for a specific dataset it can outperform the baseline architectures.

The architecture proposed, is developed in line with concepts of squeezeNet [2]. ReducedSqNet is a shallow network explicitly trained for the CIFAR-10 dataset. The idea behind developing the ReducedSqNet model is to generate an architecture which is easily deployable on embedded targets such as the Bluebox 2.0. Specific strategies are put in place to achieve a compact model size.

Strategy 1:

Replace 3x3 filters with a point-wise 1x1 filter. This strategy reduces the number of parameters by 9 times, thereby reducing the model size.

Strategy 2:

Decrease the number of input channels by using a squeeze layer which is a 1x1 point-wise filter.

Strategy 3:

Perform Down-sample at the later layers of the network to keep the dimension of activation map large. Each convolution filter in the network produces an output activation map with some spatial resolution. Height and width of this activation map are controlled by the input size, the stride of convolution filter or a pooling layer. So, to keep the dimension of activation map large, the down-sampling is implemented

towards the end of the network, i.e. use a convolution filter with a stride greater than 1 or use a pooling layers later in the network . The intuition behind this approach is large activation maps as they corresponds to more features and provides better classification accuracy [2][31].

Firstly, the depth of the baseline architecture is reduced. The intuition behind this is approach is that, since the CIFAR-10 dataset is a smaller dataset using a model with more number of hidden layers would model the training data too well and perform poorly on the test set (overfitting). Secondly, since the images in the dataset have a small dimension $32 \times 32 \times 3$ a deep network is not required. Thirdly, the position of pooling layers plays a crucial role in down-sampling the image. Therefore, in line with strategy 3 discussed before, the down-sampling is performed later in the network to keep the dimension of the activation map large. The down-sampling is controlled using the max pooling layer and stride of the convolution layer. Instead of using max pooling layer early in the network, the max pooling layer is implemented later in the network. Fig. 6.7 represents the reducedSqNet architecture with just 2 max-pooling layers. In the architecture, the max pooling layer is used after the first convolution filter and then before the last convolution filter as shown in Fig. 6.7. These changes are necessary to keep the dimension of activation maps large. To understand specific terminologies, the network architecture for squeezeNetv1.1 is discussed in further sections.

The primary focus of this chapter is to; understand the impact of the modifications implemented to the baseline architecture to create a compact architecture with less number of parameters adapted for the CIFAR-10 [42] dataset. This chapter also discusses hyper-parameter used during training process and focuses on implementing above mentioned strategies in conjunction with batch normalization techniques to achieve better performance than the baseline architecture. The two variant of proposed architectures of ReducedSqNet are compact models of 1.6MB and 2.9MB which are easy to deploy on an embedded targets and thus achieves better accuracy than the baseline architecture of 2.9MB on the CIFAR-10 dataset. Fig: 6.3 represents the

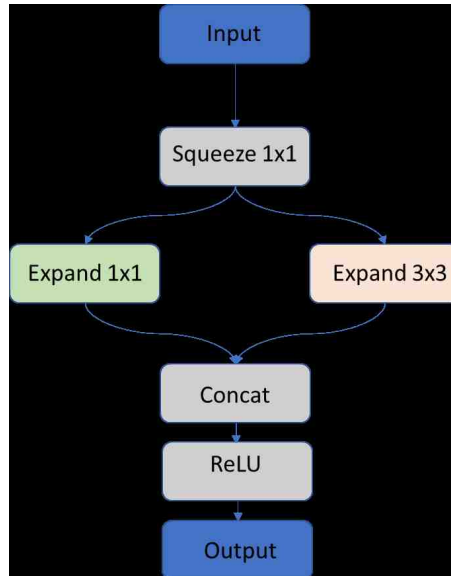


Fig. 6.2. Fire Module from squeezeNet [2].

baseline architecture of squeezeNet and further we discuss the approach is to develop a network architecture which is smaller than the existing squeezeNet baseline and suitable for the CIFAR-10 dataset.

6.3.1 Baseline Architecture

SqueezeNet is a type of CNN with similar architecture to NiN (Network in Network) [41]. The baseline model used is squeezeNet model v1.1. As shown in Fig. 6.3 the model consists of 10 hidden layers, the Fire modules, which are the building block of the network and consists of a squeeze layer which is a convolution filter of kernel size 1 and expand layers with a kernel size of 1 and kernel size of 3. This is followed by a concatenation layer and a non-linearity, stacked together in a specific arrangement shown in Fig 6.2. This can be termed as a sub-network. The model architecture can be visualized as stacks of these multiple sub-networks.

SqueezeNet has quite a few advantages when compared to other deep neural networks. The most prominent advantage is the reduced number of parameters. The

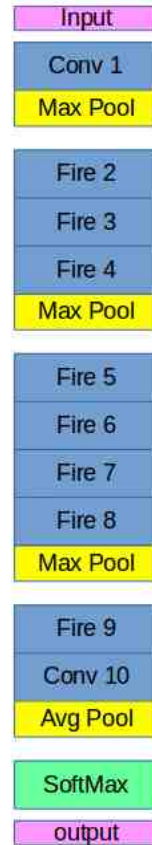


Fig. 6.3. SqueezeNet architecture v1.1 adapted for CIFAR-10 dataset.

reduction in number of parameters is due to the squeeze layer which controls the depth of the activation map of the layer and feeds into the expanding layers. The choice of replacing 3x3 with 1x1 (strategy 1) and using the squeeze layer to reduce the input channels, makes the squeezeNet ideal for embedded system and FPGA deployment. SqueezeNet has achieved the AlexNet level of accuracy with a model size of 4.8 MB on the ImageNet dataset for 1000 classes [1]. The SqueezeNet version 1.1 architecture with no bypass is replicated in Tensor-flow framework for training and testing. The baseline model is trained from scratch on the CIFAR-10 dataset.

Various algorithms and training methodologies exists for training the SqueezeNet on the CIFAR-10 and some of them have achieved an accuracy of more than 80% which is more than the 73.62% accuracy achieved in this paper. However, all such

algorithms use transfer learning techniques. In transfer learning, the model is first trained on a very large and generic dataset like ImageNet (138GB)[11] and then the pre-trained model is fine-tuned on a smaller dataset like CIFAR-10. The transfer learning technique provides better accuracy than a network, which is trained from scratch. However, training the network using the CIFAR-10 dataset takes less time as compared to training the network using the ImageNet dataset. Furthermore, the comparison between two models, one with pre-trained weights and the other trained from scratch, will be biased. Therefore the models mentioned in this chapter, are networks which are trained from scratch, so that they can be fairly compared in terms of size and accuracy.

The squeezeNet v1.1 adapted for the CIFAR-10 is first trained to establish a baseline results. This baseline results are compared with results of reducedSqNet. As mentioned earlier, squeezeNet is built using 8 fire modules. Necessary changes are made to the baseline network so that the model can be trained using the CIFAR-10 dataset. Table 6.1 summarizes the details regarding filter kernel size and strides used in the baseline architecture and provide details on the number of squeeze layers and expand layers used in a specific fire module. Number of parameter for each layer can be seen in Table 6.1.

The calculation for number of parameter is done in the following way for e.g Convolution layer 1 with input of 3 channels and output of 64 channels and a kernel size of 3x3 the total number of parameter are equal to $64 \text{ (Input Channels)} * 3 \text{ (Output Channels)} * 3 * 3 \text{ (Kernel size)} + 64 \text{ (Biases)} = 1792$ (The formatting of Table 6.1 was inspired by squeezeNet paper [2]).

Weight Initialization

One of the crucial factors while training a deep neural network is the initialization of weights before starting the training process. The main problem that researchers face, is highly non-convex objective functions. Optimization of the function requires

Table 6.1.
The table summarizes the base squeezeNet architecture

Layer Type	Output Size	Filter Size	Filter Stride	Number of s 1x1 (Squeeze)	Number of e 1x1 (Expand)	Number of e 3x3 (Expand)	Number of parameters
Input Image	32 x 32 x 3	-	-	-	-	-	0
Conv 1	16 x 16 x 64	3 x 3	2	-	-	-	1792
Max Pool 1	8 x 8 x 64	3 x 3	2	-	-	-	0
Fire 2	8 x 8 x 128		-	16	64	64	11408
Fire 3	8 x 8 x 128		-	16	64	64	12432
Fire 4	8 x 8 x 256		-	32	128	128	45344
Max Pool 2	4 x 4 x 256	3 x 3	2	-	-	-	0
Fire 5	4 x 4 x 256		-	32	128	128	49440
Fire 6	4 x 4 x 384		-	48	192	192	104880
Fire 7	4 x 4 x 384		-	48	192	192	111024
Fire 8	4 x 4 x 512		-	64	256	256	188992
Max Pool 3	2 x 2 x 512	3 x 3	2	-	-	-	0
Fire 9	2 x 2 x 512		-	64	256	256	197184
Conv 10	2 x 2 x 10	3 x 3	1	-	-	-	46090
Avg Pool 1	1 x 1 x 10	1 x 1	2	-	-	-	0
Total							768586

weights to be initialized in the right way, If the weights are too small, the signal attenuates as it propagates through the network and if the weights are too big, the signal becomes too large as it propagates through the network. Both conditions negatively impacts the performance of the network. Xaviers initialization is used to initialize the weight parameters. Xavier initializes the weights in the network by drawing them from a distribution with zero mean and a fixed variance. Weights can also be initialized based on data statistics of the dataset used for training [39].

Training

After weight initialization, we proceed with the training process where the choice of learning rate, optimizer, and the loss function plays a crucial role in reducing the generalization error by optimizing the weights of the model. The initial choice of optimizer for training is Stochastic gradient descent to update the weights. But, after running optimization for nearly 33000 iterations (i.e. approximately 84 epochs at the learning rates of 0.04 and 0.0001), the model did not converge. This is due to a saddle point (local minimum). Hence the Adam optimizer is used and the training is conducted at a lower learning rate. The loss curve can be seen in Fig 6.4. The accuracy observed is 20% after 1000 iterations.

In stochastic gradient descent, a single learning rate parameter is maintained for all weight updates and the learning rate does not change during the training process. However, Adam computes individual adaptive learning rates for different parameters from the estimate of the first and second moments of the gradients [43]. The following optimization parameters are used: the decay rate of the 1st moment estimate is 0.9 and the decay rate of the 2nd moment estimate is 0.99. The learning rate is initially kept at 0.04. In practice, Adam is currently recommended as the default algorithm to use as it often works better than RMSProp [44] and AdaGrad [45]. At the learning rate of $2e-4$, the baseline model starts converging with Adam optimizer and the accuracy increases. The base model is trained for 100,000 iterations which is approximately

255 epochs with batch sizes of 64, 128 and 256. It can be seen from Fig 6.5 that the batch size of 256 images with learning rate of $2e-4$ gives the best result for baseline model with the mean of 63.41%. The worst performance is with the batch size of 64 with a mean of 59.09%. With the averages of multiple runs, the accuracy observed is 63.41% for the baseline network. As seen from loss plot Fig. 6.6 that the loss decreases up to 35000-40000 iterations and then starts increasing. This phenomenon is over-fitting. The network starts to over-fit on the dataset and loses generality. This is due to using a small dataset for training. It is a strong indication that particular analysis is required to prevent over-fitting of data. To avoid over-fitting an early stopping algorithm is implemented.

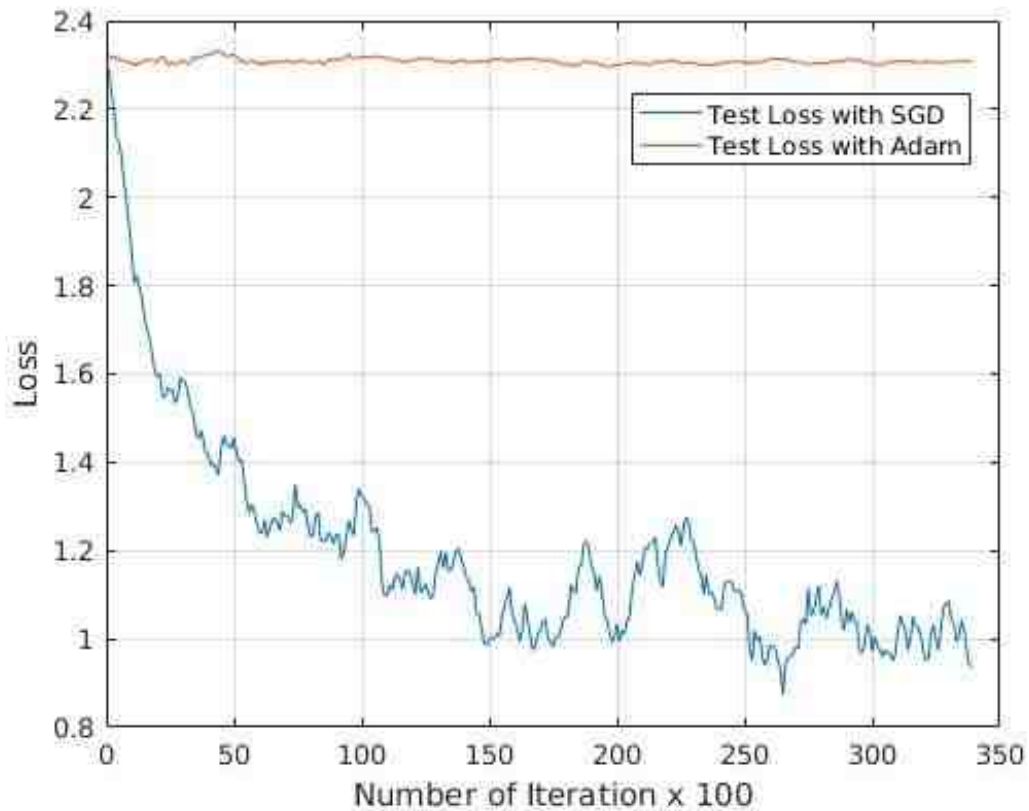


Fig. 6.4. Test Loss with SGD (*Orange*) and Adam (*Blue*).

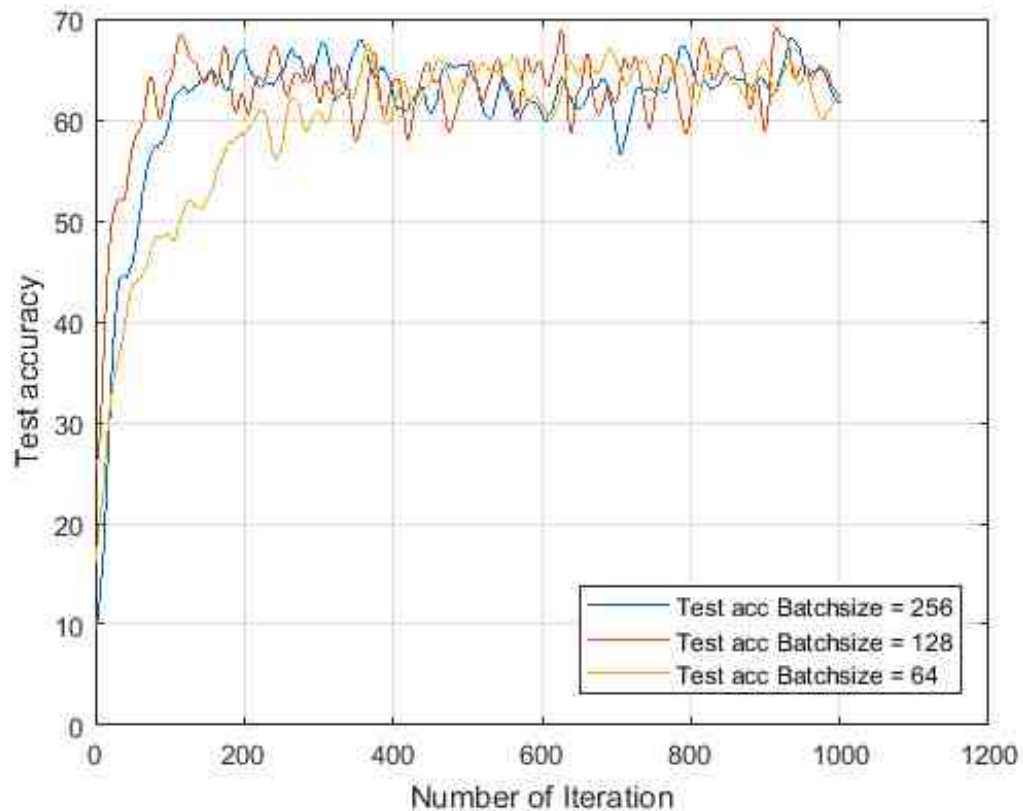


Fig. 6.5. The plot represents the multiple runs with different hyper-parameters.

It is observed in Fig. 6.5 that the batch size of 256 with the learning rate of $2e-4$ gives the best result for the base squeezeNet with the mean of 69.96% and the worst performance is with the batch size of 64 with a mean of 50.09%. With multiple runs, the average accuracy observed was 63.41% for the base network. From the loss plot Fig. 6.6 it is observed that the loss decreases up to 35000-40000 iterations and then starts increasing. This phenomenon is known as over-fitting. The network starts to over-fit on the dataset and loses generality. This is due to using a small dataset for training. It is a strong indication that particular analysis is required to prevent over-fitting of data. To avoid over-fitting an early stopping algorithm was implemented.

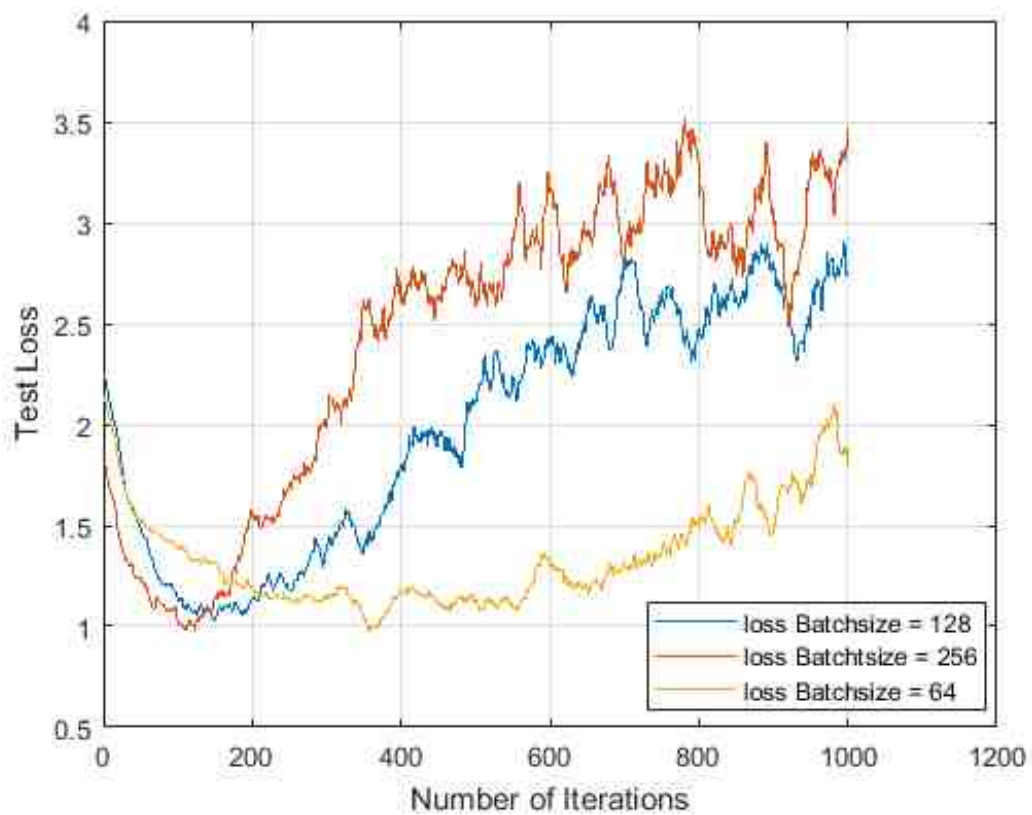


Fig. 6.6. Test loss at learning rate of 0.0001.

Result: Stops the training process when validation error increases a specific threshold.

Let θ be the initial parameters;

$i = 0$;

$j = 0$;

$v = \infty$;

$P = //$ number of times to observe validation error before stopping;

while $j < P$ **do**

 Keep training for n steps;

$i = i+n$;

$\hat{v} =$ validation error;

if $\hat{v} < v$ **then**

$j = 0$;

$v = \hat{v}$;

else

$j = j+1$;

end

end

Algorithm 1: Early Stopping Criterion

6.3.2 ReducedSqNet Architecture

It is always important to design an architecture specific to the dataset as it provides better performance. ReducedSqNet architecture is designed for a specific dataset (CIFAR-10). The motivation is to generate a model with less number of parameters than the baseline model and with a competitive accuracy. The modifications were done to the network to reduce the model size and the number of parameters. The design choices were also dependent on the dataset that was used.

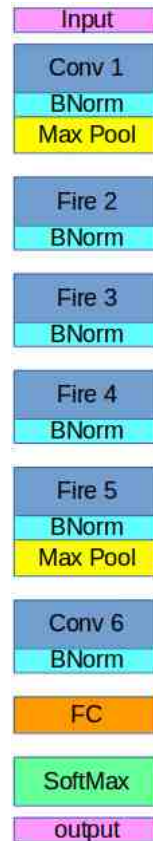


Fig. 6.7. Proposed ReducedSqNet architecture for CIFAR-10 dataset.

The size of baseline model is 2.9MB, the proposed architecture achieves a compact model and performs equivalently to the baseline network (squeezeNet v1.1) on the CIFAR-10 dataset. This reduction in model size is due to change in depth of architecture. The intuition behind reducing the depth is as follows: since the CIFAR-10 dataset consists of the images of size $32 \times 32 \times 3$ which is very small, a deeper network may not be required to model the dataset. It is observed during the training of the baseline network that, the test loss starts to increase after few number of iterations which is because of over-fitting, as small dataset is used. Reducing the depth of the the model and adding the batch-normalization layer improved the performance of the network. Also, Reducing the depth of architecture results in less number of parameters and reduces the model complexity which helps to counter the over-fitting

Table 6.2.

Table represents the number of parameters in reducedSqNet architecture, the table also highlights the position of pooling layers and batch normalization layers used in the architecture.

Layer Types	Output Size	Filter Size	Filter Stride	Number of s 1x1 (Squeeze)	Number of e 1x1 (Expand)	Number of e 3x3 (Expand)	Number of Parameters
Input Image	32 x 32 x 3	-	-	-	-	-	0
Conv 1	32 x 32 x 64	3 x 3	1	-	-	-	1792
Batch Norm	32 x 32 x 64	-	-	-	-	-	2
Max Pool 1	16 x 16 x 64	2 x 2	2	-	-	-	0
Fire 2	16 x 16 x 128		-	16/32	64	64	11408/22688
Batch Norm	16 x 16 x 128	-	-	-	-	-	2
Fire 3	16 x 16 x 128		-	16/32	64	64	12432/24736
Batch Norm	16 x 16 x 128	-	-	-	-	-	2
Fire 4	16 x 16 x 256		-	32/64	128	128	45344/90432
Batch Norm	16 x 16 x 256	-	-	-	-	-	2
Fire 5	16 x 16 x 256		-	32/64	128	128	49440/98624
Batch Norm	16 x 16 x 256	-	-	-	-	-	2
Max Pool 2	8 x 8 x 256	2 x 2	2	-	-	-	0
Conv 6	8 x 8 x 10	1 x 1	1	-	-	-	2570
Batch Norm	8 x 8 x 10	-	-	-	-	-	2
Fully Connected	[640->10]	-	-	-	-	-	6410
Total							129408/247264

problem as observed in Fig 6.6. Inline with strategy 3 discussed before, the down-sampling is performed later in the network to keep the dimension of the activation map large. The down-sampling is controlled using the max pooling layer and stride of the convolution layer. Instead of using max pooling layer early in the network, the max pooling layer is implemented later in the network. The Fig. 6.7 represents the reducedSqNet architecture with just 2 max pooling layers. In the architecture the max pooling layer is used after the first convolution filter and then before the last convolution filter as shown in Fig. 6.7. This is necessary to keep the dimension of activation maps large. Two variants of the model are created: one with squeeze ratio of 0.125 and other with the squeeze ratio of 0.25. Calculations of the squeeze ratio is as follows:

$$\text{SqueezeRatio} = \frac{s_{i,1x1}}{e_{i,1x1} + e_{i,3x3}} \quad (6.7)$$

$s_{i,1x1} \rightarrow 1x1$ convolution filter, squeeze layer i

$e_{i,1x1} \rightarrow 1x1$ convolution filter, expand layer i

$e_{i,3x3} \rightarrow 3x3$ convolution filter, expand layer i

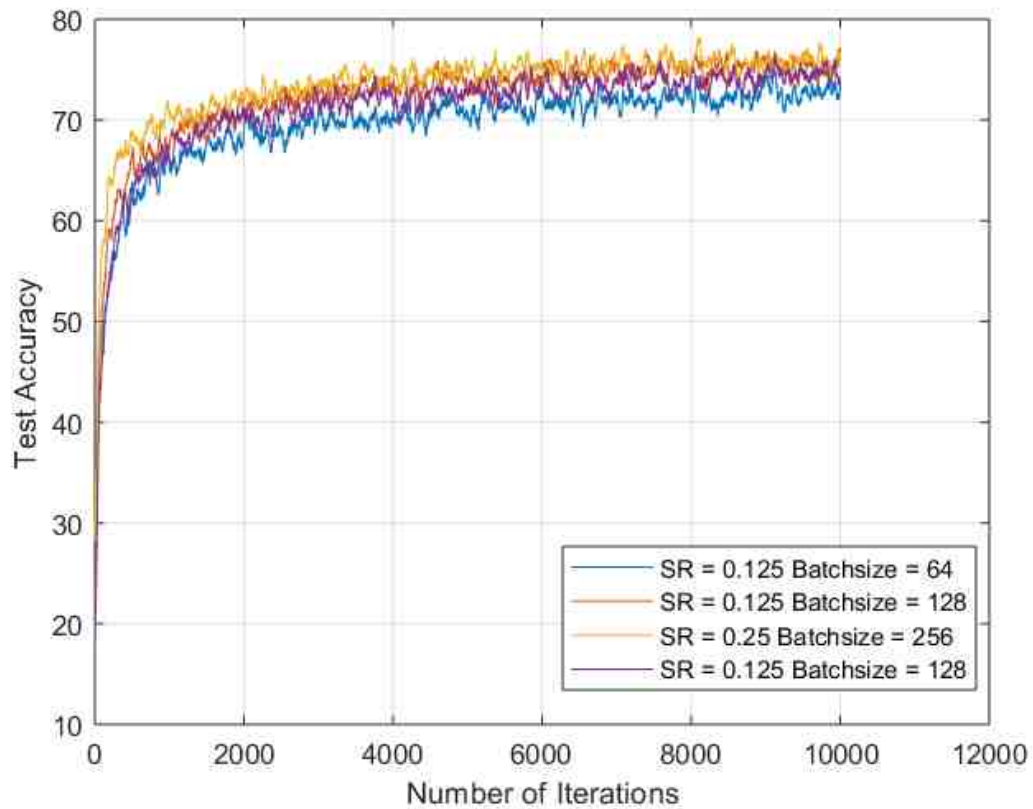


Fig. 6.8. Accuracy curves for different hyper-parameters configurations.

Finally, a very small fully connected layer of $8 \times 8 \times 10 = 640$ neurons is added at the end of the network to enhance the performance of the reducedSqNet.

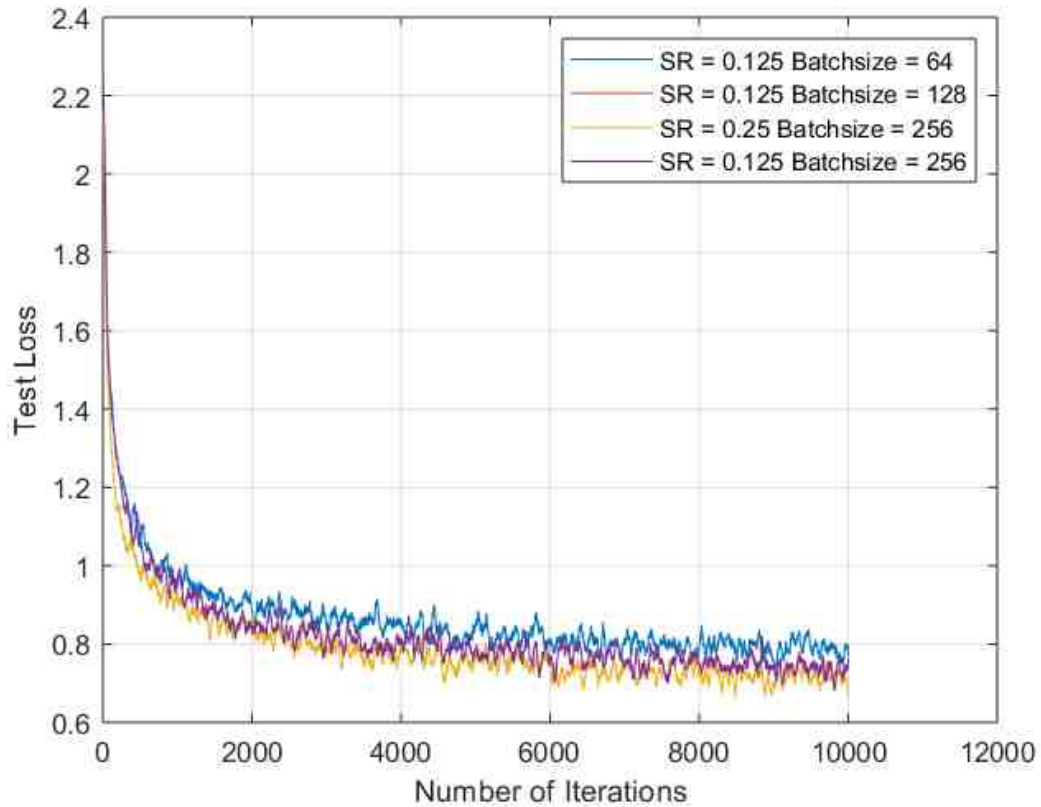


Fig. 6.9. Loss curves for different hyper-parameters configurations.

While training the reducedSqNet, data augmentation techniques such as random flip, hue, contrast, brightness and saturation are used which introduces co-variate shifts in the data to counter the overfitting problems which can be seen in Fig 6.6. Use of the batch-Normalization layer, helped in the optimization process and reduced the training time of the network. The Table 6.2 provides the details for the reducedSqNet layers and the number of parameters per layer.

The two variants of model with different squeeze ratios are trained on tensor-flow. The optimizer used is Adam with learning rate of 0.002 with batch sizes of 64, 128 and 256. The proposed models converged with better accuracy than the baseline network. The Fig 6.8 shows accuracy plot for multiple runs of the reducedSqNet with various

batch sizes and Squeeze ratios. After multiple runs with different hyper-parameters, it is observed that the best accuracy is achieved at a learning rate of 0.002 and squeeze ratio of 0.25, batch size of 256 images, with a mean of 79.86% and standard deviation of 1.5. The worst accuracy achieved is with the batch size of 64, with a mean 69.46% accuracy. With averages of multiple runs the average accuracy observed is 74%.

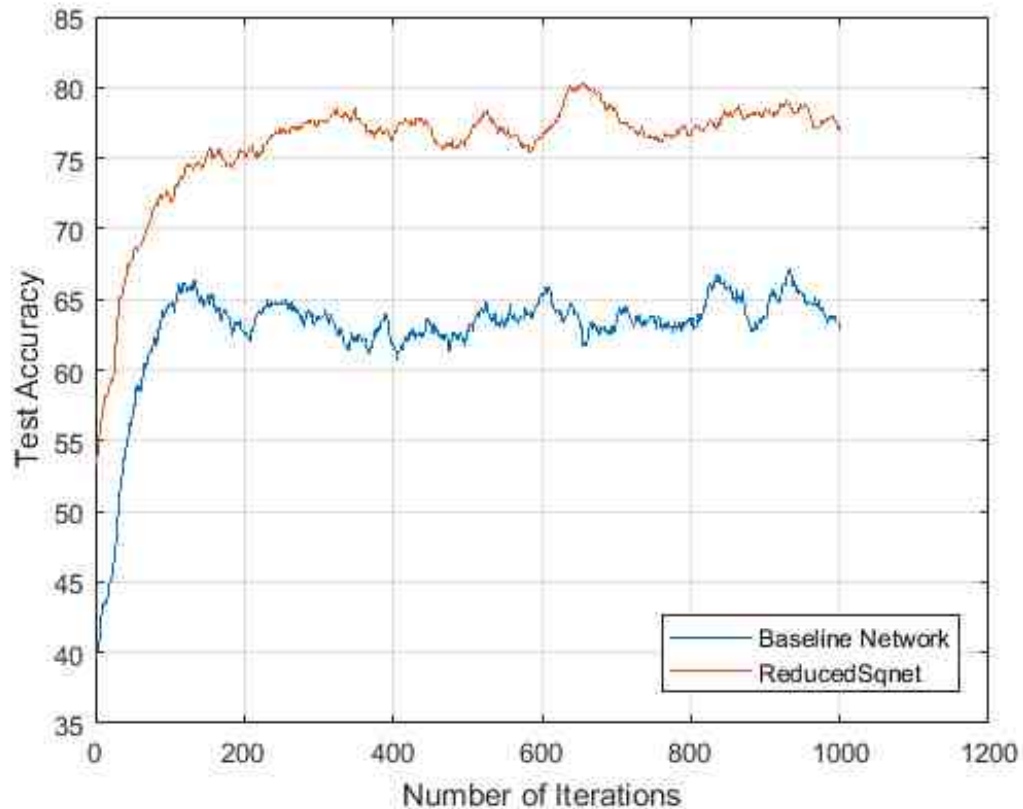


Fig. 6.10. Testing accuracy of baseline model vs reducedSqNet.

The Fig 6.10 shows the test accuracy for the baseline network and the proposed reducedSqNet, which is trained with the same hyper-parameters. It is observed that the accuracy of the proposed network is higher than the baseline architecture and the test loss is much lower when compared to the baseline model as shown in Fig 6.11. It can also be seen from Fig. 6.11 that the test loss for baseline network increases after

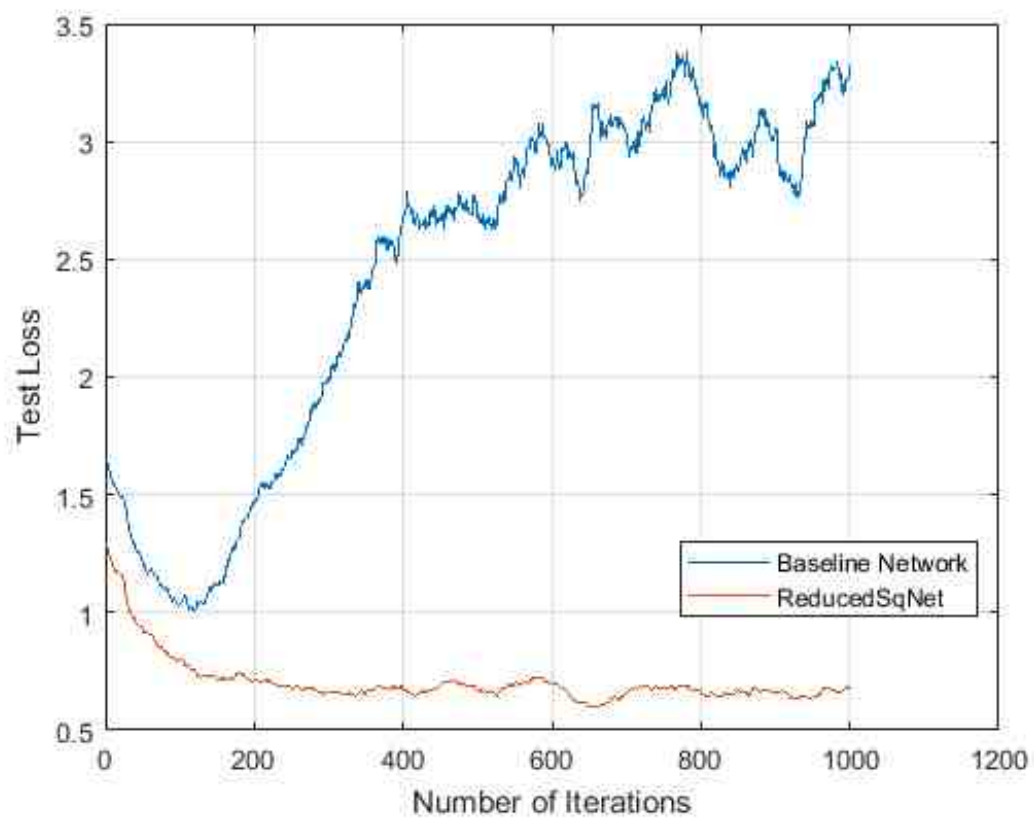


Fig. 6.11. Testing loss of baseline model vs reducedSqNet.

a few number of iterations whereas for the reducedSqNet the test losses continuously decreases. This is due to the reduction in the depth of the architecture and data augmentation techniques put in place to combat the overfitting problems.

Fig. 6.10 concludes that, reducing the number of parameters still maintains the same accuracy. Hence, it would be appropriate in this scenario to conclude that, reducing the depth of the network did not have a negative impact on the accuracy of the model. This highlights the fact that, a compact network can be trained to perform tasks with similar accuracy compared to its deep counterpart. Another conclusion regarding these changes is that since the image size on the CIFAR-10 dataset is 32x32, having a deep network is futile. Because, towards the end of the network the image will be too down-sampled, and the network will not be able to learn a lot of high-level features. Therefore, a tailored architecture for the specific dataset provides a better performance.

Note: *During the training process, some experiments were conducted with non-linearity in the network, ReLU corresponding to convolution 1 and 10 were removed and replaced with exponential linear unit. Due to the removal of the above mentioned ReLUs an increase in accuracy was observed. Multiple runs were recorded and observed using three different batch sizes: 256, 128 and 64 with the learning rate of $2e-3$ and results observed were consistent.*

Results:

Replacing the ReLU with ELU caused the learning to be faster than the previous model and had slightly better accuracy than the previously trained model. Exponential Linear units alleviate the dead ReLU problem. In contrast to ReLUs, ELUs have negative values which allow them to push mean unit activations closer to zero like batch normalization but with lower computational complexity.

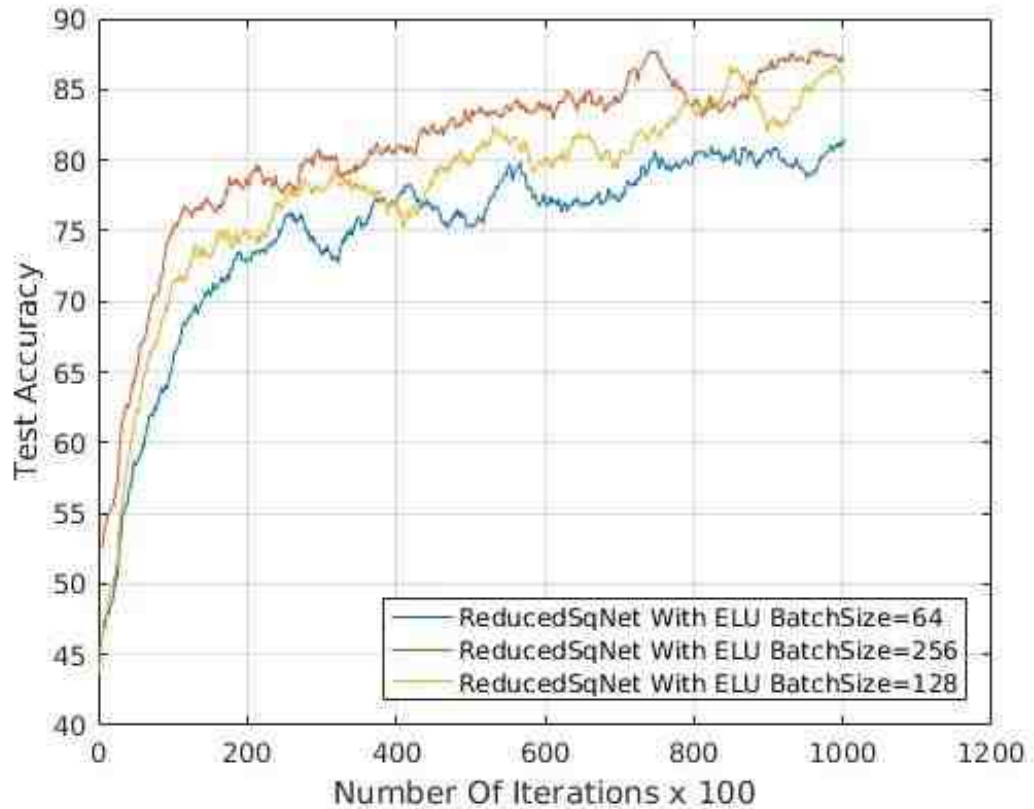


Fig. 6.12. Experimentation Results with Exponential Linear Unit.

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \quad (6.8)$$

Also, ELU has mean activation close to zero and does not saturate for large inputs. It behaves similarly to a ReLU unit if the input is positive, but for negative values, it is a function bounded by a fixed value of -1 , for $\alpha=1$. This behavior pushes the mean activation of neurons closer to zero which is beneficial for learning representations that are more robust to noise. The following graphs highlight the improvement over the baseline network.

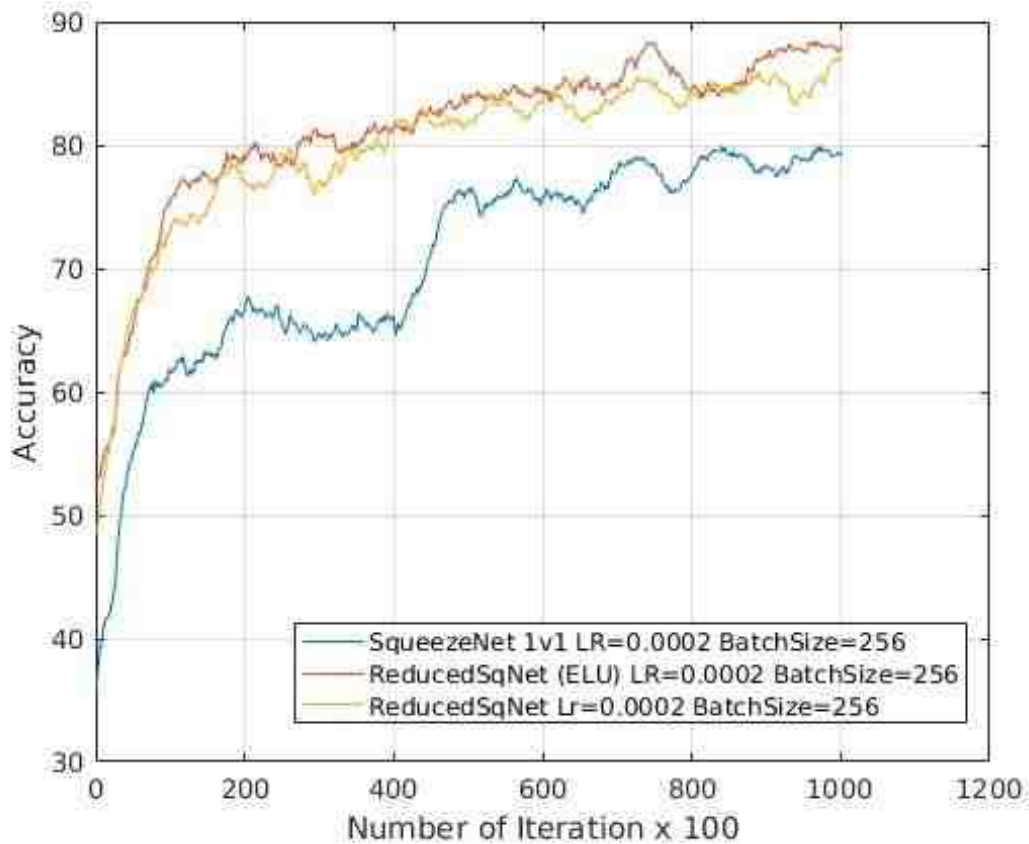


Fig. 6.13. This figure shows the accuracy comparison of the base squeezeNet vs the ReducedSqnet vs ReducedSqnet with ELU.

6.4 Conclusion

This chapter presents a compact architecture build on the principle of fire modules. The proposed network architecture is adapted for CIFAR-10 dataset and has a smaller model size than the baseline architecture and has a better performance, if trained from scratch on the CIFAR-10 dataset. This paper also highlights the choice for number of pooling layers to be used and their positions in the network. It is observed that using the pooling layer towards the end of the network results in a better accuracy. Next, use of batch normalization layer is also highlighted in the paper. It is observed that

Table 6.3.
Results for ReducedSqNet vs squeezeNet v 1.1

Model name	Squeeze Ratio	Model Size(MB)	Accuracy
SqueezeNet v 1.1	0.25	2.9	63.4%
ReducedSqNet	0.125	1.4	70%
ReducedSqNet	0.25	2.8	74%

by adding a batch normalization layer after every convolution and fire layer reduces the fluctuation in the loss and accuracy curves of the proposed architecture and also results in the faster training time as compared to baseline network. The Table 6.3 provides a summary of various architectures trained and their performance measures. Also, from Fig 6.10 and 6.11, It is observed that the accuracy of the reducedSqNet has a better performance than the baseline model. This compact model can be used on embedded targets for real-time classifications. It can be concluded that tailoring the architecture for specific dataset will result in the overall improved performance.

In this chapter, we discussed an approach to reduce the model size of the existing architecture by reducing the depth of the model and maintain the same level of accuracy by increasing the width of the network. Table 6.3 consists of collective information of comparison between baseline network, ReducedSqNet with squeeze ratio of 0.125 and 0.25. It is observed that the accuracy for the modified architecture surpasses the baseline architecture if trained from scratch. Hence it is essential to develop specific architectures for a specific dataset.

7. SQUEEZED CNN - A NEW ARCHITECTURE FOR CIFAR-10

In the previous chapter, a new architecture for CIFAR-10 was proposed by slightly modifying the baseline architecture by adding a batch normalization layer after the convolution filters and adding a fully connected layer at the end of the network. This chapter focus on the same principle of developing an architecture with a compact model size. Here we create a deeper architecture with 16 hidden layers of convolution inspired by VGG-16 [4], with the concepts of squeeze layers discussed in previous chapters and adding batch normalization layers. Though the proposed architecture is very different from the VGG architecture, we use VGG-16's performance measure as the baseline for development. VGG-16 has 16 hidden layers, these hidden layers consist of convolution filters with the kernel size of 3x3, pooling layer 2x2 and three fully connected layers at the end of the network, the model has a size of 528 MB for ImageNet dataset [32] and 385 MB for CIFAR-10 dataset this difference is due to a smaller number of classes in the CIFAR-10 dataset. The approach highlighted in this chapter provides a compressed version of CNN with a model size of 12.9 MB, which is a 96% reduction in the model size. The model is trained to have the same number of layers as VGG-16, but the fully connected layers are removed from the network architecture to reduce the model size, and batch normalization layer is added to the network. Reduction in the model size leads to less overhead when uploading new models to an embedded device. Many companies in the domain of autonomous driving need to update the model wirelessly to the end device, and this has improved the reliability and safety of autonomous driving systems [2]. Attempts have been made in this paper to reduce the model size of the architecture keeping in mind the competitive accuracy of VGG16.

Table 7.1.

Table highlights the number of parameters for each layer of VGG-16 architecture

Layer Type	Output Size	Filter Stride	Filter Size	Number of s 1x1 (Squeeze)	Number of e 1x1 (Expand)	Number of e 3x3 (Expand)	Number of Parameters
Conv 1.1	32x32x64	1	3x3	-	-	-	1792
Conv 1.2	32x32x64	1	3x3	-	-	-	36928
Max Pool	16x16x64	2	2x2	-	-	-	0
Conv 2.1	16x16x128	1	3x3	-	-	-	73856
Conv 2.2	16x16x128	1	3x3	-	-	-	147584
Max Pool	8x8x128	2	2x2	-	-	-	0
Conv 3.1	8x8x256	1	3x3	-	-	-	295268
Conv 3.2	8x8x256	1	3x3	-	-	-	590080
Conv 3.3	8x8x256	1	3x3	-	-	-	590080
Max Pool	4x4x256	2	2x2	-	-	-	0
Conv 4.1	4x4x512	1	3x3	-	-	-	1180160
Conv 4.2	4x4x512	1	3x3	-	-	-	2359808
Conv 4.3	4x4x512	1	3x3	-	-	-	2359808
Max Pool	2x2x512	2	2x2	-	-	-	0
Conv 5.1	2x2x512	1	3x3	-	-	-	2359808
Conv 5.2	2x2x512	1	3x3	-	-	-	2359808
Conv 5.3	2x2x512	1	3x3	-	-	-	2359808
Max Pool	2x2x512	2	2x2	-	-	-	0
Dense Layer	FC 4096	-	-	-	-	-	2101248
Dense Layer	FC 4096	-	-	-	-	-	16781312
Dense Layer	FC 10	-	-	-	-	-	40970
Total							33,683,218

7.1 VGG-16 Architecture

The Visual geometry group of the University of Oxford proposed a 16 layer architecture known as VGG-16 [4]. It is a state-of-the-art network architecture and evaluated on ILSVRC-2012 with 28.1% top-1 error and 9.3% top-5 error [4]. VGG architecture has multiple variants the most famous ones are 16 layers and 19 layers deep network. Table 7.1 provides a summary of total number of parameters in VGG-16 architecture. The VGG architecture is the most preferred architecture to extract features from images, the pre-trained network for VGG-16 is widely available

online. VGG-16 is an obvious choice as many object detection architectures use VGG architecture for classification. The VGG networks have small receptive fields of 3×3 instead of using a large receptive field of 7×7 , the stride of convolution filters is 1. The approach is to use multiple convolution filters stacked together instead of using a single large filter. This approach also reduces the number of parameters considerably, e.g., 3 convolution filters of kernel size 3×3 stacked together have $3 \times (3 \times 3 \times K) = 27K$ parameters whereas 1 convolution filter of 7×7 has in total $49K$ parameters. This reduction in parameters has a significant impact on the model size. Fig 7.1 represents the architecture of the VGG-16 networks.

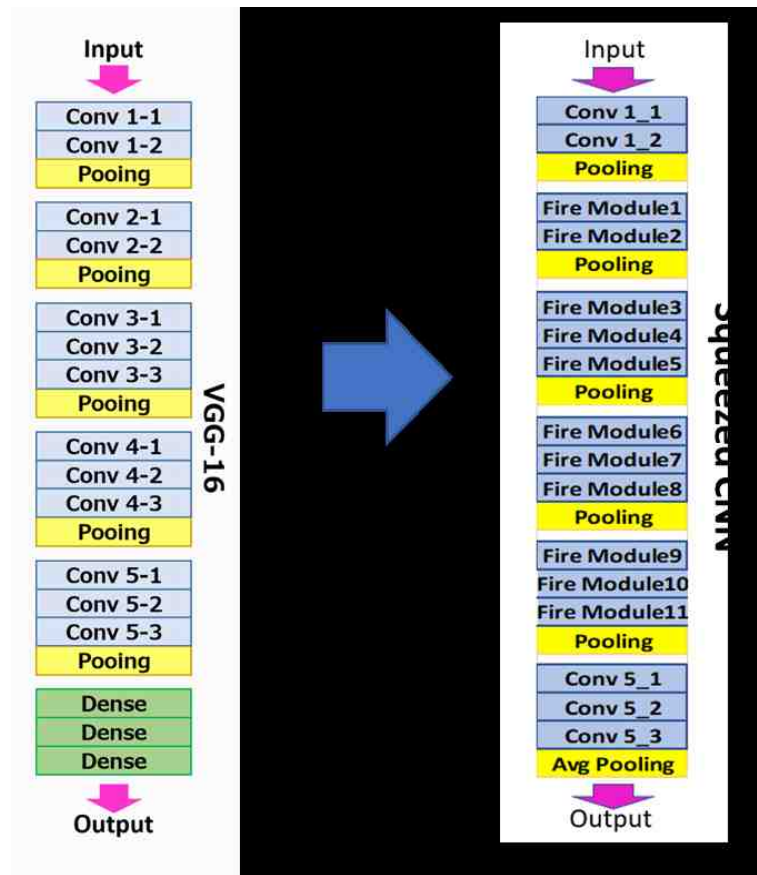


Fig. 7.1. This figure shows VGG-16 Architecture (*Left*) Proposed Architecture (*Right*).

This architecture has in total 16 layers with 13 convolution filter and 3 fully connected networks. Each convolution filter has a kernel size of 3x3 and a stride of 1 with 5 pooling layers with the stride of 2 to down sample the image. Towards the end of the network, there are 3 fully connected layers for computing the class scores.

7.2 Squeezed CNN

Previous chapters discuss the approach to develop a compact model and an architecture for the hardware targets. Inline with these approaches, a new fully convolution architecture is proposed. This new architecture incorporates the positives of both the networks (VGG-16 and SqueezeNet). The VGG-16's depth and the stack of convolution layers are recreated using the concept of fire modules and varying the squeeze ratio to reduce the model size and achieve architecture compression. The batch normalization layers are used after every convolution filters to handle the covariate shifts introduced during data augmentation and reduce the training time. The proposed architecture is a fully convolutional network with 16 convolution layers and no fully connected layer. The idea behind using a fully-convolution network is motivated because the convolution neural networks are sparse and have less number of parameters. The last layer of the convolution is modified to output class scores for the number of classes in the CIFAR-10 dataset the changes made to the model with competitive accuracy and minimal model size. The proposed model was trained on the CIFAR-10 dataset and could achieve the accuracy of 76.32% with a model size of 12.9 MB as compared to 92.2% accuracy with a pre-trained model size of 385MB and 82% with a model trained from scratch. Significant reduction in the model size of 96.78% is achieved. Fig 7.1 (*Right*) represents the squeezed CNN model architecture, and table 7.2 represents the total number of parameters in Squeezed CNN architecture. The proposed architecture consists of two convolution filters of 3x3 kernel size stacked together followed by a max pooling layer. The convolutional filters are cascaded with two fire modules (FireModule1 and FireModule2) followed by a max pooling layer,

Table 7.2.
Table represents Squeezed CNN architecture and Number of parameters in each layer.

Layer Type	Output Size	Filter Stride	Filter Size	Number of s 1x1 (Squeeze)	Number of e 1x1 (Expand)	Number of s 3x3 (Expand)	Number of Parameters
Conv 1.1	32x32x64	1	3x3	-	-	-	1792
Batch Norm	32x32x64	-	-	-	-	-	2
Conv 1.2	32x32x64	1	3x3	-	-	-	36928
Batch Norm	32x32x64	-	-	-	-	-	2
Max Pool	16x16x64	2	2x2	-	-	-	0
Fire Module 1	16x16x128	-	-	32	64	64	22688
Batch Norm	16x16x128	-	-	-	-	-	2
Fire Module 2	16x16x128	-	-	32	64	64	24736
Batch Norm	16x16x128	-	-	-	-	-	2
Max Pool	8x8x128	2	2x2	-	-	-	0
Fire Module 3	8x8x256	-	-	32	128	128	45344
Batch Norm	8x8x256	-	-	-	-	-	2
Fire Module 4	8x8x256	-	-	64	128	128	98624
Batch Norm	8x8x256	-	-	-	-	-	2
Fire Module 5	8x8x256	-	-	64	128	128	98624
Batch Norm	8x8x256	-	-	-	-	-	2
Max Pool	4x4x256	2	2x2	-	-	-	0
Fire Module 6	4x4x512	-	-	64	256	256	180800
Batch Norm	4x4x512	-	-	-	-	-	2
Fire Module 7	4x4x512	-	-	64	256	256	197184
Batch Norm	4x4x512	-	-	-	-	-	2
Fire Module 8	4x4x512	-	-	64	256	256	197184
Batch Norm	4x4x512	-	-	-	-	-	2
Max Pool	2x2x512	2	2x2	-	-	-	0
Fire Module 9	2x2x512	-	-	16	64	64	18576
Batch Norm	2x2x512	-	-	-	-	-	2
Fire Module 10	2x2x512	-	-	32	64	64	24736
Batch Norm	2x2x512	-	-	-	-	-	2
Fire Module 11	2x2x512	-	-	32	64	64	24736
Batch Norm	2x2x512	-	-	-	-	-	2
Conv 2.1	2x2x64	1	3x3	-	-	-	73792
Batch Norm	2x2x512	-	-	-	-	-	2
Conv 2.2	2x2x32	1	3x3	-	-	-	18464
Batch Norm	2x2x32	1	3x3	-	-	-	2
Conv 3.3	2x2x10	1	3x3	-	-	-	2890
Batch Norm	2x2x10	-	-	-	-	-	2
Avg Pool	1x1x10	2	2x2	-	-	-	0
						Total	1,067,128

and similarly, 3 stacks of 3 Fire modules are cascaded to the architecture. Finally, in the end, one more stack of 3x3 convolution filters are added and an average pooling layer to produce class scores. As discussed earlier the idea behind adding a fire module is to reduce the number of parameters and model size. Table 7.2 contains the accuracy and the model size for the base architecture of VGG-16 and Squeezed CNN which is a fully convolutional network. As represented in Fig. 7.2 the accuracy of the model goes down by nearly 16%, but the model size is reduced by a considerable factor of 96%. This reduction in model size increases the scope of adding more layers to the network to increase the accuracy of the architecture. This drop in size gives room to add more hidden layers to the network to increase the accuracy but adding layer also increases the complexity of optimization and may create problems of vanishing gradient, which can be tackled by injecting gradients through techniques like auxiliary optimization techniques used in GoogleNet [33] and Residual learning [34]. This approach can be used in future to develop a compressed architecture with much better accuracy.

The proposed model is trained in tensorflow on the CIFAR-10 dataset. Since the dataset contains a small set of examples, and the complexity of the model is high (more number of the hidden layer) there is a high probability that the model will overfit the dataset. To prevent overfitting of the model, the concept of dropout layers were used. The dropout layer work in the following way: during the training process some percentage of neurons are deactivated, due to this the model is forced to learn the same feature with different connections. This approach helps in reducing the generalization error. In the architecture, the dropout is implemented after fire module 6 and fire module 11 both dropout layer had 50 dropout percentage. Batch size used was 512. Therefore 97 batches were required to complete 1 epoch and the network was trained for 100 epochs approx. The test accuracy evaluated at 100 epochs had a mean of 76.32%.

Table 7.3.
Summary for VGG-16 vs Squeezed CNN

#	VGG-16	Squeezed CNN	VGG-16 (without pretrained model)
Accuracy	92.20%	76.32%	82%
Model Size	385MB	12.9MB	385MB

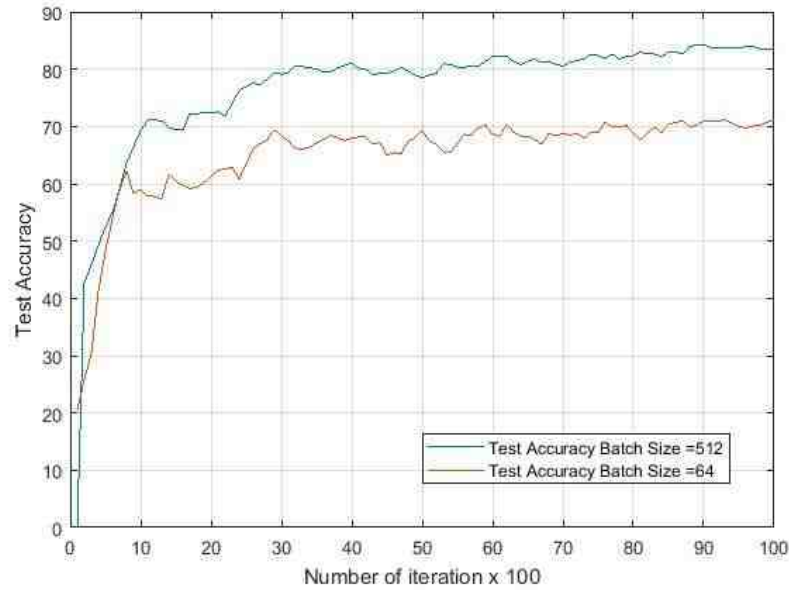


Fig. 7.2. Test Accuracy of squeezed CNN Architecture.

7.2.1 Hardware Deployment of Squeezed CNN

The squeezed CNN model is trained on a desktop pc with a core i7 processor, 32 GB RAM and GTX 1080 by Nvidia using the tensorflow framework and deployed on BlueBox by NXP. BlueBox is a development platform for self-driving cars. It incorporates S32V234 which is a vision and sensor fusion processing unit, LS2084A embedded computer processor and S32R27 radar micro-controller. BlueBox2.0 provides the required performance for deploying convolutional neural networks and can be embedded with Realtime Multisensor Application software (RTMaps)[35]. RTMaps is a high-performance platform with support for frameworks such as tensorflow, opencv, sklearn, etcetera. These packages are instrumental in developing, training and deploying neural network architectures. RTMaps studio provides an easy to use graphical user interface and component libraries to tackle sensor fusion and computer vision problems. The out of the box packages provided with RTMaps facilitates algorithm prototyping and development.

7.2.2 Bluebox 2.0 by NXP

NXP is the leader in developing semiconductor devices for the automotive domain. NXP's Bluebox 2.0 is a development platform targeted for developing software and application for autonomous systems. Bluebox 2.0 incorporates S32V234 for vision processing, LS2084A for heavy computation and S3VR27 a radar micro-controller. S32V234 within Bluebox contains image processing platform with dual camera interfaces, image signal processors and APEX cores for image cognition processing and four A-53 ARM cores. LS2084A the CPU platform contains eight ARM A-72 processors for data crunching and provides high-performance data path and peripheral network interfaces for networking. S32R27 is a 32-bit power architecture-based microcontroller for automotive and radar applications [36].

7.2.3 Real-time Multisensor Applications (RTMaps)

RTMaps is an asynchronous, high-performance platform to design a multisensory framework and prototype sensor fusion algorithm. RTMaps has of several modules: RTMaps Runtime Engine, RTMaps Studio, RTMaps Component Library and RTMaps SDK [37].

RTMaps studio provides a development interface for prototyping and building real-time applications. The block diagrams in workspace made of connected components from RTMaps libraries represents the application built in RTMaps studio. The studio provides a user-friendly interface to set up the applications [35]. The RTMap runtime engine takes care of base services such as component registration, time stamp threading, and priorities.

7.2.4 Deployment of Squeeze CNN

As discussed earlier this thesis aims at developing a pipeline to design, train and deploy CNN classifiers on Bluebox 2.0 using RTMaps studio. RTMaps provides sup-

port for deep learning framework such as tensorflow. Fig 7.3 provides the process of deploying a trained network on Bluebox 2.0. The checkpoint files generated after the

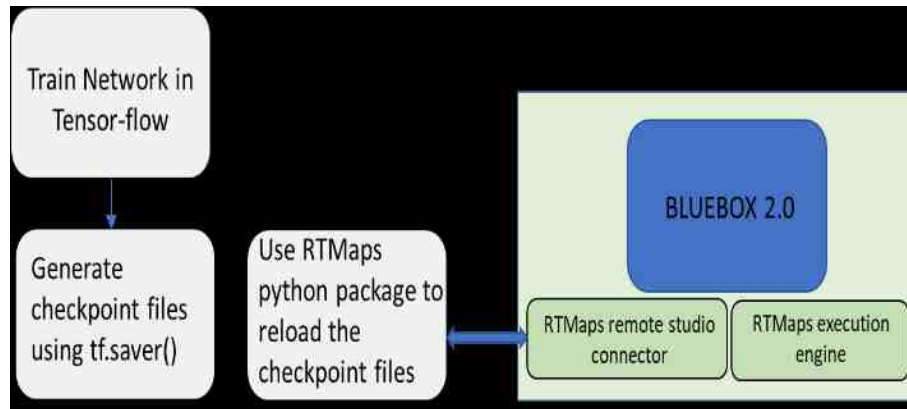


Fig. 7.3. Flow for application development using RTMaps for Bluebox 2.0.

training process was saved on the Bluebox 2.0 and the RTMaps studio was connected to the execution engine using TCP/IP which ran the software on the Linux operating system installed on Bluebox2.0.

RTMaps provides a python structure to create and deploy python codes. The python code for RTMaps contains three function definition Birth(), Core() and Death(). The Birth() is executed once so all the setup code can be called and implemented within this section. Core() runs in an infinite loop. Therefore, the part of the code that runs continuously can be defined in this section. Death() runs after the program is halted so all the cleanup functions can be implemented under this section. This defined structure makes it easy for prototyping and developing a modular code. Below is a Fig 7.4 representing the graphical interface that RTMaps provides for developing the algorithm.

7.2.5 Conclusion

The focus of this chapter was to design and deploy a compact version of convolution neural networks on Bluebox 2.0 development platform. The approach followed

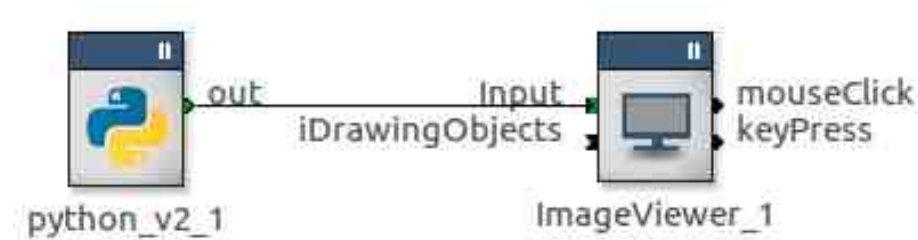


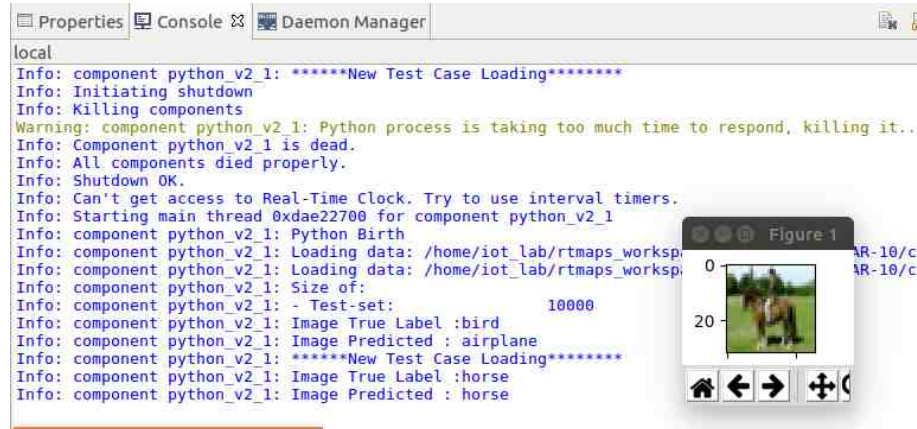
Fig. 7.4. Graphical interface in RTMaps.

has two parts. Firstly, this chapter highlights the process to define a compressed architecture using the concept of fire modules train the new architecture on the CIFAR-10 dataset. Secondly, use techniques such as batch normalization and data augmentation to improve the performance of the compressed architecture. Thirdly, deploy the network on Bluebox 2.0 using RTMaps studio.

Initially, the chapter discusses the VGG-16 architecture, types of layers used and the number of parameters in each layer of VGG-16. This architecture serves as a baseline architecture for developing the compact architecture. Next, this chapter discusses the proposed compressed architecture Squeezed CNN and the number of parameters and types of the layer used in the architecture and intuition behind using these layers. Further, This chapter discusses the process of training the squeezed CNN architecture and highlights the reduction in model size from 385 MB to 12.9 MB which is a reduction of 96%.

The reduction techniques using the fire modules discussed in the chapter suggests a generic technique to create convolution neural networks for memory and power constrained devices. The second part of the chapter is specific to Bluebox 2.0 and RTMaps. It proposes a process that can be followed to build and deploy convolution neural networks on Bluebox development platforms using RTMaps studio. It also explains the code structure used in the RTMaps studio for deploying python packages. Finally, the chapter highlights and shares the result of classifier deployed on Bluebox 2.0 in Fig 7.5. During this process of deploying convolution neural networks,

it was observed that RTMaps provides easy to use component libraries and the integration with the tensorflow framework is seamless. It reduces the prototyping effort for deployment on BlueBox 2.0 and provides a very user-friendly environment for the developer.



```

local
Info: component python_v2_1: *****New Test Case Loading*****
Info: Initiating shutdown
Info: Killing components
Warning: component python_v2_1: Python process is taking too much time to respond, killing it..
Info: Component python_v2_1 is dead.
Info: All components died properly.
Info: Shutdown OK.
Info: Can't get access to Real-Time Clock. Try to use interval timers.
Info: Starting main thread 0xdae22700 for component python_v2_1
Info: component python_v2_1: Python Birth
Info: component python_v2_1: Loading data: /home/iot_lab/rtmaps_worksp
Info: component python_v2_1: Loading data: /home/iot_lab/rtmaps_worksp
Info: component python_v2_1: Size of:
Info: component python_v2_1: - Test-set:          10000
Info: component python_v2_1: Image True Label :bird
Info: component python_v2_1: Image Predicted : airplane
Info: component python_v2_1: *****New Test Case Loading*****
Info: component python_v2_1: Image True Label :horse
Info: component python_v2_1: Image Predicted : horse
  
```

The screenshot shows a terminal window with RTMaps console output. The output details the lifecycle of a Python component (python_v2_1), including its shutdown, restart, and execution. It also shows the loading of a test set and the results of image predictions for two images: a bird (true label) and a horse (true label). A small window titled 'Figure 1' is overlaid on the terminal, displaying a horse image with a vertical axis labeled '0' and '20'.

Fig. 7.5. Console output on RTMaps for displaying results.

8. INTEGRATING CNN WITH FASTER RCNN AND YOLO

As discussed earlier the motivation behind this thesis is to create a compact convolution neural network for memory and power constrained devices, that is useful for developing real-time applications. The approaches discussed in the previous chapters helps to generate compact models for constrained devices. However, this chapter discusses the integration of compact architectures such as squeezenet and Squeezed CNN with the object detection pipelines based on region proposal networks such as faster R-CNN and classification or regression based networks such as YOLO (You Only Look Once). The basics of object detection are discussed in chapter 3. Hence, this chapter skips all the pieces of information provided in chapter 3.

8.1 Squeezenet integrated with Faster R-CNN

To implement object detection in real-time, it is crucial to integrate the convolution neural network with the object detection pipeline. The choice of CNN is squeezeNet baseline architecture integrated with the region proposal network in faster R-CNN. This integration is achieved by replacing the VGG-16 convolution neural network with squeezeNet convolutional Neural Network. Since the VGG-16 is a convolution neural network which has fully connected layers at the end of the network, and the squeezeNet is a fully convolutional network. Therefore, the squeezeNet was modified so that it can be compatible with the existing faster R-CNN pipeline. The modification required for compatibility is the is discussed in later sections.

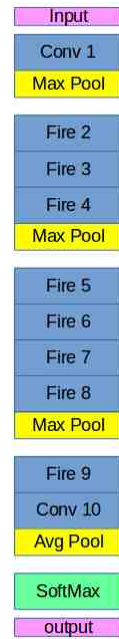


Fig. 8.1. This figure represents the SqueezeNet Architecture .

8.1.1 Modification for model compatibility

Since the squeezenet architecture is trained on a larger dataset which is the imagenet. Due to the pre-training on the imagenet, the model has already learned the low-level features, and it is essential to preserve these features and modify the last layers so that the high-level feature can be learned from the new dataset. To achieve this, the last convolution layer Conv 10 is removed from the squeezenet architecture which is shown in Fig 8.1. The fire module 9 layer is attached to the region proposal network of faster-RCNN object detection pipeline Fig 8.2 represents the default architecture of faster R-CNN and the changed architecture where squeezenet CNN replaces the VGG-16 CNN. As discussed, the focus of the research is to reduce the model size of the architecture. The changes made to the architecture made it possible to develop a compact model and reduce the inference time of the trained network by a factor of 2 the detection time for faster R-CNN with the VGG-16 CNN is 0.09 ms and the detection time for faster R-CNN is approximately 0.04 ms on GTX 1080

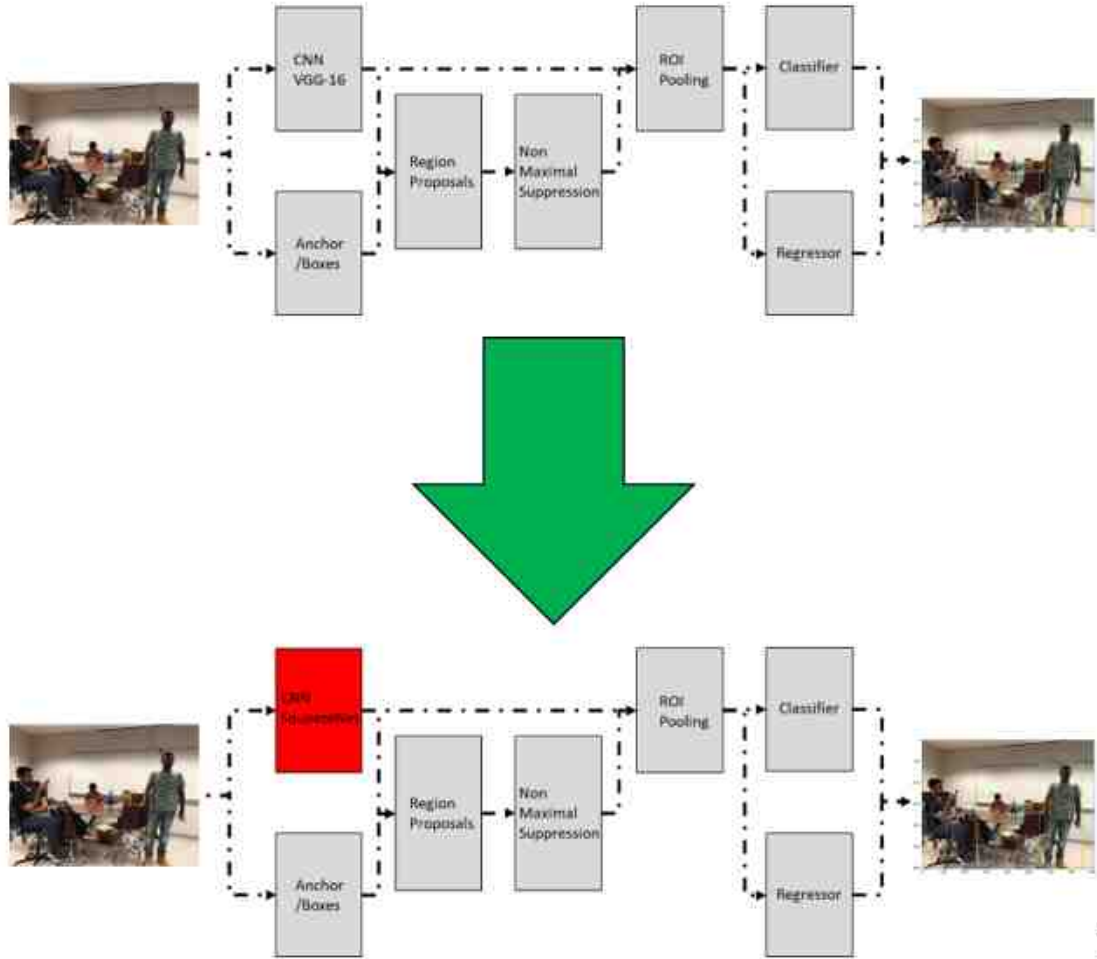


Fig. 8.2. This figure represents the integration of SqueezeNet with Faster R-CNN .

GPU. The model size reduction is from 553.4 MB to 80.7 MB. Though we can reduce the model size and detection time for the architecture the Mean Average Precision (MAP) of the current model is not at par with the benchmark model. The MAP for the baseline architecture tested is 73%MAP whereas the MAP for the modified architecture is 27%MAP which is less than half of the benchmarked model, but there is a significant reduction in the detection time it is reduced by a factor of 2 and a reduction in the model size is approximately 85%.

Table 8.1.
Model Comparison for Faster R-CNN vs Modified Architectures

Architecture	mAP	Speed	Model Size
Faster R-CNN(Baseline)	73	95ms	553MB
R-SqueezeNet	33	30ms	81MB
R-Squeezed CNN	10	45ms	181MB

These modifications help to create a model feasible for deployment on embedded targets. Also, create real-time applications on memory constrained and power constrained devices. The following table presents the data regarding the various combination of CNN architectures used. From the table, it is apparent that reduction in model size the boosts the frame per secs. With the R-Squeezenet we can perform object detection with a mAP of 27% and model size of 80.7MB. It is also observed the the R-Squeezed CNN developed has mAP lesser that R-Squeezenet this is because the R-Squeezed CNN is not trained on ImageNet dataset but on Tiny imageNet dataset. Training this architecture on ImageNet will have a better mAP.

8.1.2 Detection Results

As observed in Fig. 8.5 the detection results are not accurate for R-Squeezed CNN one way to overcome this is to train the Squeezed CNN architecture on ImageNet data set. This approach might increase the MAP of the proposed network.

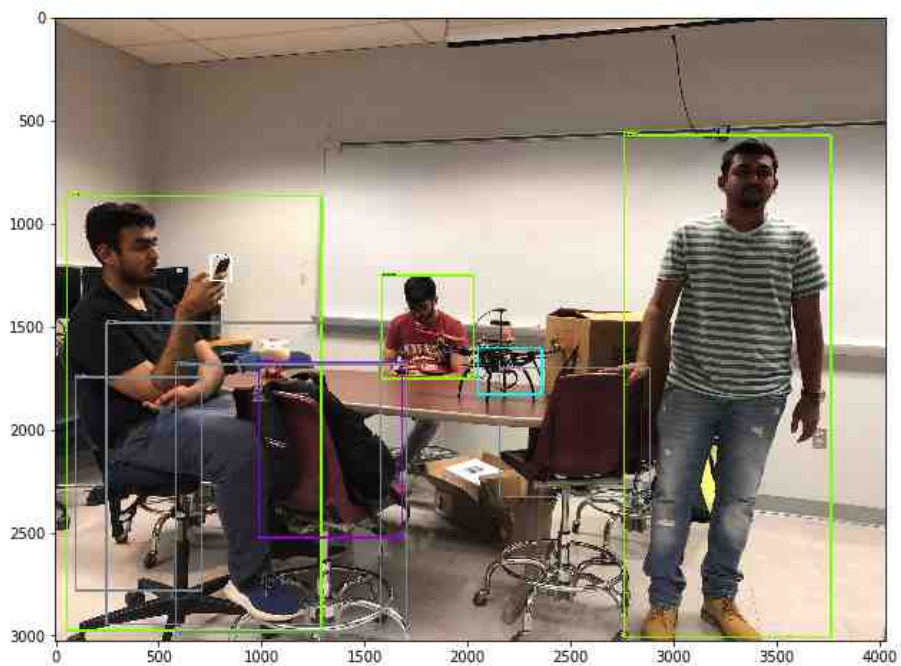


Fig. 8.3. Detection Result for Faster R-CNN where CNN is VGG-16.

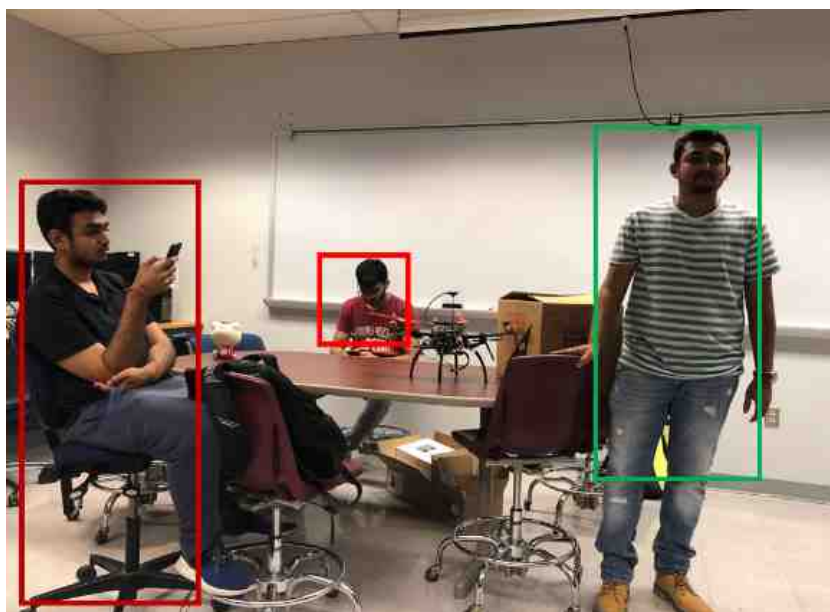


Fig. 8.4. Detection Result for R-SqueezeNet (Faster R-CNN + SqueezeNet).

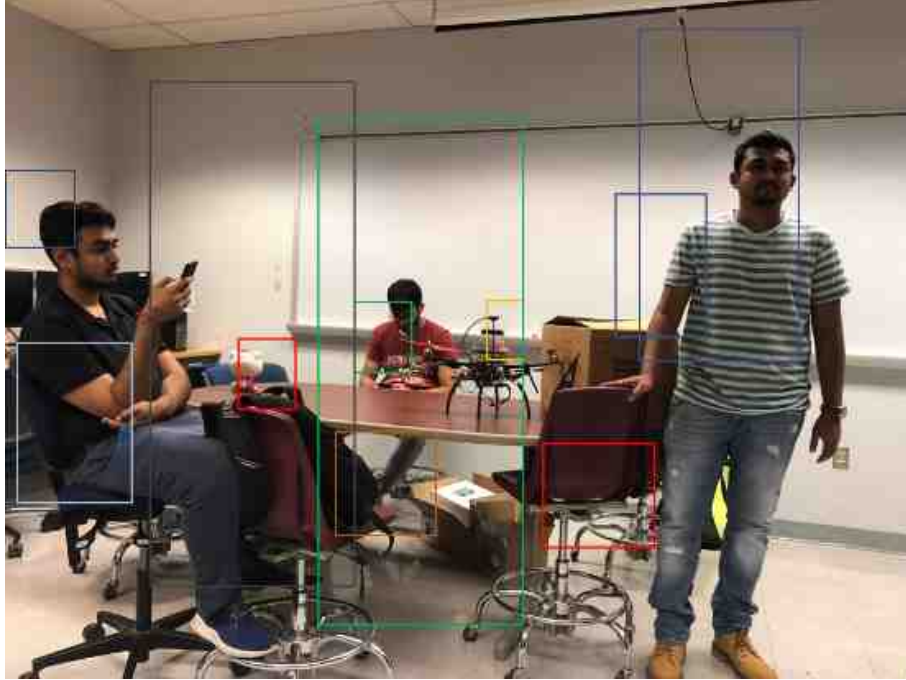


Fig. 8.5. Detection Result for R-Squeezed CNN (Faster R-CNN + Squeezed CNN).

8.2 SqueezeNet integrated with You Look Only Once (YOLO)

YOLO architecture poses the object detection as a regression problem. The single convolution neural network with the fully connected layer at the end is responsible for classifying class scores and the co-ordinate for bounding boxes. YOLO divides the image into an $S \times S$ grid, and each grid is responsible for predicting the bounding boxes. For YOLO we were able to train the model on Pascal VOC dataset and KITTI dataset this idea was inspired by squeezeDet [46].

8.2.1 YOLO Architecture

YOLO consist of a CNN which is for classification, and the last layer is modified to provide a tensor of $S \times S \times (B * 5 + C)$, where S = grid size, B = number of bounding boxes per grid and C is confidence for the bounding box. The modified architecture

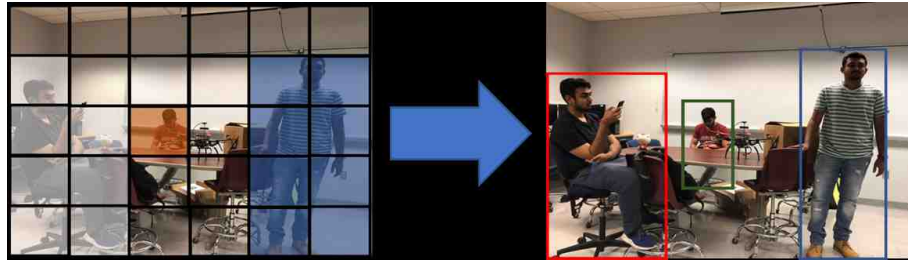


Fig. 8.6. Representation of how YOLO works.

integrates the squeezeNet[2] with the YOLO pipeline with a simple modification. An additional layer was added after the last layer, i.e. fire 9 and to provide the bounding box and confidence score for the bounding box. Adding SqueezeNet to YOLO pipeline reduced the model size to 10MB and the detection time reduced to 20ms.

8.2.2 Detection Results

Below are few detection results on real-time video.



Fig. 8.7. Detection result for SqueezeNet + YOLO on Real-Time Videos



Fig. 8.8. Detection result for SqueezeNet + YOLO on Real-Time Videos



Fig. 8.9. Detection result for SqueezeNet + YOLO on Real-Time Videos



Fig. 8.10. Detection result for SqueezeNet + YOLO on Real-Time Videos

9. SUMMARY

The motivation behind this thesis has been the development of convolution neural network architecture designed specifically for embedded systems. During this thesis, multiple CNN architectures were developed using fire modules. Firstly, a compact architecture called ReducedSqNet was developed for the CIFAR-10 dataset this network was trained and compared to the baseline model. It was observed that the model performed better than the baseline architecture on CIFAR-10 even though the depth and the model size was lower than the baseline network. The results for the reducedSqNet network are highlighted in chapter 6 of this thesis. These results proved that the fire modules could be used to recreate the existing architectures and with fine-tuning, a compressed convolutional neural network with competitive accuracy can be obtained, which is feasible for deployment on embedded systems. Next, a convolution neural network with 16 layers was proposed named squeezed CNN inspired by VGG-16. Squeezed CNN was created using compact convolution filters called fire modules, which helped to reduce the model size of the architecture. Chapter 7 in the thesis highlights the results for squeezed CNN architecture.

Both the compact architectures are feasible for deployment on embedded systems and were deployed on an autonomous development platform Bluebox 2.0. Furthermore, the squeezed CNN and the squeezeNet architecture were integrated with the object detection pipelines Faster R-CNN and YOLO this gives a proof of concept that any CNN trained can be integrated with these pipelines. Results for these integrated models are discussed in Chapter 8. Also, the process of deploying the trained model on the Bluebox using RTMaps software is discussed in chapter 7.

REFERENCES

REFERENCES

- [1] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems(NIPS)*(pp. 1097-1105). [online] [<http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>] [Accessed on 6th October 2018].
- [2] Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J. and Keutzer, K., 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. arXiv preprint arXiv:1602.07360, Cornell University.[online] [<https://arxiv.org/pdf/1602.07360.pdf>] [Accessed on 6th October 2018].
- [3] Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A.A., 2017, February. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI* (Vol. 4, p. 12).
- [4] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, Cornell University. [online] [<https://arxiv.org/pdf/1409.1556.pdf>] [Accessed on 6th October 2018].
- [5] Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y. and Azumi, T., 2018, April. Autoware on board: enabling autonomous vehicles with embedded systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems* (pp. 287-296). IEEE Press.
- [6] Nair, V. and Hinton, G.E., 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (pp. 807-814).
- [7] Clevert, D.A., Unterthiner, T. and Hochreiter, S., 2015. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289, Cornell University. [online] [<https://arxiv.org/pdf/1511.07289.pdf>] [Accessed on 6th October 2018].
- [8] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V. and Ng, A.Y., 2012. Large scale distributed deep networks. In *Advances in neural information processing systems(NIPS)* (pp. 1223-1231). [online] [<http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf>] [Accessed on 6th October 2018].
- [9] Conti, F. and Benini, L., 2015, March. A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters. In *Proceedings of the 2015 Design, Automation and Test in Europe Conference and Exhibition* (pp. 683-688). EDA Consortium.

- [10] McCulloch, W.S. and Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), pp.115-133, Springer.
- [11] Marek R, Thoughts on Machine Learning and Natural Language Processing., [online] [<http://www.marekrei.com/blog/neural-networks-part-2-the-neuron/>] [Accessed on 5th October 2018].
- [12] Deep Neural Network: What is Deep Learning Edureka? [online] [<https://cdn.edureka.co/blog/wp-content/uploads/2017/05/Deep-Neural-Network-What-is-Deep-Learning-Edureka.png>] [Accessed on 5th October 2018].
- [13] Zhao, Z.Q., Zheng, P., Xu, S.T. and Wu, X., 2018. Object detection with deep learning: A review. arXiv preprint arXiv:1807.05511, Cornell University. [online] [<https://arxiv.org/pdf/1807.05511.pdf>] [Accessed on 6th October 2018].
- [14] Lienhart, R. and Maydt, J., 2002. An extended set of haar-like features for rapid object detection. In *Proceedings of 2002 International Conference on Image Processing (Vol. 1, pp. I-I)*. IEEE.
- [15] Cortes, C. Vapnik, V. *Machine Learning* (1995) 20: 273. [online] [<https://doi.org/10.1023/A:1022627411411>] [Accessed on 5th October 2018].
- [16] Felzenszwalb, P.F., Girshick, R.B., McAllester, D. and Ramanan, D., 2010. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9), pp.1627-1645.
- [17] Freund, Y. and Schapire, R.E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), pp.119-139.
- [18] CS231n: Convolutional Neural Networks for Visual Recognition. [online] [<http://cs231n.github.io/convolutional-networks/>] [Accessed on 6th October 2018]
- [19] Kayid, A., and Yasmeeen K., Performance of CPUs/GPUs for Deep Learning workloads. [online] [<https://www.researchgate.net>] [Accessed on 6th October 2018]
- [20] Karl R., CPU, GPU and MIC Hardware Characteristics over Time [online] [<https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/>] [Accessed on 6th October 2018].
- [21] He, K., Zhang, X., Ren, S. and Sun, J., 2014, September. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European conference on computer vision* (pp. 346-361). Springer, Cham.
- [22] Girshick, R., 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448).
- [23] Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91-99). [online] [<http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf>] [Accessed on 6th October 2018].

- [24] Dai, J., Li, Y., He, K. and Sun, J., 2016. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems* (pp. 379-387). [online] [<https://papers.nips.cc/paper/6465-r-fcn-object-detection-via-region-based-fully-convolutional-networks>] [Accessed on 6th October 2018].
- [25] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).
- [26] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016, October. Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21-37). Springer, Cham.
- [27] Tomasz G, in *Data science, Deep learning, Machine learning*, [online][<https://deepsense.ai/region-of-interest-pooling-explained/>][Accessed on 6th October 2018].
- [28] Hong, S., Roh, B., Kim, K.H., Cheon, Y. and Park, M., 2016. PVANet: lightweight deep neural networks for real-time object detection. arXiv preprint arXiv:1611.08588, Cornell University. [online] [<https://arxiv.org/pdf/1611.08588.pdf>] [Accessed on 6th October 2018].
- [29] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S. and Murphy, K., 2017, July. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE International Conference on Computer Vision and Pattern Recognition* (Vol. 4).
- [30] Tome, D., Bondi, L., Baroffio, L., Tubaro, S., Plebani, E. and Pau, D., 2016, September. Reduced memory region based deep Convolutional Neural Network detection. In *2016 IEEE 6th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)* (pp. 15-19). IEEE.
- [31] Gaikwad, A., 2018 "Pruning the Convolution Neural Network (SqueezeNet) using Taylor Expansion Based Criterion", *International Symposium on Signal Processing and Information Technology, International Symposium on Signal Processing and Information Technology IEEE*.
- [32] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026-1034).
- [33] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. Imagenet: A large-scale hierarchical image database, In *Computer Vision and Pattern Recognition, IEEE* (pp. 248-255).
- [34] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).
- [35] Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A.A., 2017, February. Inception-v4, inception-resnet and the impact of residual connections on learning. In *American Association for Artificial Intelligence* (Vol. 4, p. 12).

- [36] Venkitachalam, S., Manghat, S.K., Gaikwad, A.S., Ravi, N., Bhamidi, S., El-Sharkawy, M., Realtime Applications with RTMaps and Bluebox 2.0, International Conference Artificial Intelligence 2018 ICAI'18.
- [37] Nxp.com. (2018). NXP BlueBox Autonomous Driving—NXP. [online] [<https://www.nxp.com/products/processors-andmicrocontrollers/arm-based-processors-and-mcus/qoriq-layerscape-armprocessors/nxp-bluebox-autonomous-driving-developmentplatform:BLBX>] [Accessed 6th June. 2018]
- [38] Intempora.com. (2018). Intempora - RTMaps - A component-based framework for rapid development of multi-modal applications. [online][<https://intempora.com/>] [Accessed 6th September. 2018].
- [39] Jin, J., Dundar, A. and Culurciello, E., 2014. Flattened convolutional neural networks for feedforward acceleration. arXiv preprint arXiv:1412.5474, Cornell University. [online] [<https://arxiv.org/pdf/1412.5474.pdf>] [Accessed on 6th October 2018].
- [40] Koturwar, S. and Merchant, S., 2017. Weight Initialization of Deep Neural Networks (DNNs) using Data Statistics. arXiv preprint arXiv:1710.10570, Cornell University. [online] [<https://arxiv.org/pdf/1710.10570.pdf>] [Accessed on 6th October 2018].
- [41] SLin, M., Chen, Q. and Yan, S., 2013. Network in network. arXiv preprint arXiv:1312.4400, Cornell University. [online] [<https://arxiv.org/pdf/1312.4400.pdf>] [Accessed on 6th October 2018].
- [42] Krizhevsky, A., Nair, V. and Hinton, G., 2014. The CIFAR-10 dataset. [online] [<http://www.cs.toronto.edu/kriz/cifar.html>] [Accessed 6th September. 2018].
- [43] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, Cornell University. [online] [<https://arxiv.org/pdf/1412.6980.pdf>] [Accessed on 6th October 2018].
- [44] Tieleman, T. and Hinton, G., 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), pp.26-31.
- [45] Zeiler, M.D., 2012. ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701, Cornell University. [online] [<https://arxiv.org/pdf/1212.5701.pdf>] [Accessed on 6th October 2018].
- [46] Wu, B., Iandola, F.N., Jin, P.H. and Keutzer, K., 2017, July. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving, In Proceedings of the IEEE conference on computer vision and pattern recognition workshop (pp. 446-454).