PRUNING CONVOLUTION NEURAL NETWORK (SQUEEZENET) FOR EFFICIENT HARDWARE DEPLOYMENT

A Thesis

Submitted to the Faculty

of

Purdue University

by

Akash S. Gaikwad

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

Department of Electrical and Computer Engineering

Dr. Maher Rizkalla

Department of Electrical and Computer Engineering

Dr. Brian King

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

Head of the Graduate Program

This Thesis is dedicated to my Parents,

Jayashree and Sunil Gaikwad,

and my family,

Alisha and Ragini.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my adviser Dr. Mohamed El-Sharkawy for his patience, motivation and constant guidance throughout my masters education and research. Besides my advisor, I would like to thank my fellow labmates, Durvesh, Dewant, Surya, Shreeram and Raghavan for valuable advice and stimulating sleepless discussion and for all the fun we had in the last two years.

My sincere thanks to IoT Collaboratory and department of Electrical and Computer Engineering, who provided the access to the laboratory and research facilities. Special thanks to Sherrie and Dr. Brian King for their constant support and motivation throughout my time at IUPUI. Last but not the least, I would like to thank and acknowledge, Government of Maharashtra and India, for sponsoring my education.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

SYMBOLS

$\circledast$	Convolution
r_i	Feature map
$C(.)$	Cost function of the network

ABBREVIATIONS

CNN	Convolution Neural Network
NN	Neural Network
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
ReLU	Rectified Linear Unit
CV	Computer Vision
FCNN	Fully Convolutional Neural Network
SIMD	Single Instruction, Multiple Data
BLBX2	BlueBox Version 2
RTOS	Real-Time Operating System
ISP	Image Signal Processor
LTS	Long Term Support
BSP	Board Support Package
FPGA	Field Programmable Gate Array
SATA	Serial Advanced Technology Attachment
SoC	System on Chip
ROS	Robot Operating System

ABSTRACT

Gaikwad, Akash S. M.S.E.C.E., Purdue University, December 2018. Pruning Convolution Neural Network (SqueezeNet) for Efficient Hardware Deployment. Major Professor: Mohamed El-Sharkawy.

In recent years, deep learning models have become popular in the real-time embedded application, but there are many complexities for hardware deployment because of limited resources such as memory, computational power, and energy. Recent research in the field of deep learning focuses on reducing the model size of the Convolution Neural Network (CNN) by various compression techniques like Architectural compression, Pruning, Quantization, and Encoding (e.g., Huffman encoding). Network pruning is one of the promising technique to solve these problems.

This thesis proposes methods to prune the convolution neural network (SqueezeNet) without introducing network sparsity in the pruned model.
This thesis proposes three methods to prune the CNN to decrease the model size of CNN without a significant drop in the accuracy of the model.

1. Pruning based on Taylor expansion of change in cost function ΔC.
2. Pruning based on L_2 normalization of activation maps.
3. Pruning based on a combination of method 1 and method 2.

The proposed methods uses various ranking methods to rank the convolution kernels and prune the lower ranked filters afterwards SqueezeNet model is fine-tuned by backpropagation. Transfer learning technique is used to train the SqueezeNet on the CIFAR-10 dataset. Results show that the proposed approach reduces the SqueezeNet model by 72% without a significant drop in the accuracy of the model (optimal pruning efficiency result). Results also show that Pruning based on a combination of Taylor expansion of the cost function and L_2 normalization of activation

maps achieves better pruning efficiency compared to other individual pruning criteria and most of the pruned kernels are from mid and high-level layers. The Pruned model is deployed on BlueBox 2.0 using RTMap software and model performance was evaluated.

1. INTRODUCTION

1.1 Motivation

Deep Convolutional Neural Networks (CNN) have a variety of applications, and are extensively used in the field of computer vision. In recent years, advancement in computation power and active research in the deep learning field has resulted in improvement of the benchmark architectures such as SqueezeNet [1], Alex-Net [2], and VGG16 [3] for feature extraction from images. Understanding an image or feature extraction from the image has been one of the difficult challenges for researches in recent years. Previous approaches used to extract the features, are the traditional computer vision algorithm or hard-coded algorithms. The deep learning architectures are replacing the conventional computer vision algorithms for feature extraction methods with high accuracy (ex. Object detection with Canny or HOG are replaced by CNN). As the network grows more in-depth, the model size and inference time of the model is also increasing. For embedded devices, resources such as computation power, memory, and energy are limited [4] as shown in Figure 1.1. To deploy these modern models in small edge-embedded devices such as smart-phone, raspberry pi, $S32V234$ [5], the parameters such as computational efficiency, limited memory size, and power efficiency must be considered [6]. For the distributed training on parallel servers, the overhead to train the model is directly proportional to the number of parameters and complexity of the model [7]. Table 1.1 shows the number of parameters in various modern deep CNN.

The model size of CNN can play a vital role when it comes to transporting the model wirelessly in terms of the firmware or software updates. Table 1.2 shows the percentage of convolution and fully connected layer in various Convolution Neural Networks. Many redundant parameters do not contribute towards the output in the

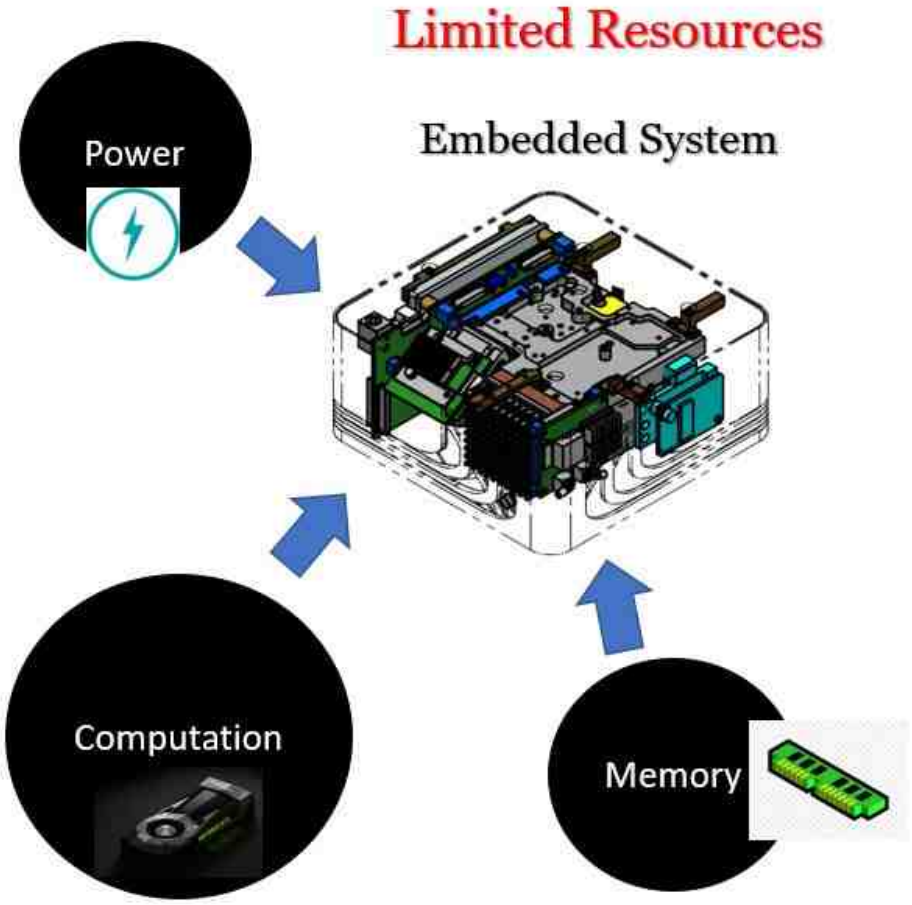

Fig. 1.1. Limited resources for embedded devices.

Table 1.1.
Types of layers in modern Convolution Neural Network.(Percentage wise)

Network Architecture	Convolution layer	Fully Connected layer
VGG16	99%	1%
Alexnet	89%	11%
R3DCNN	90%	10%
SqueezeNet	100%	0%

deep convolution neural network [8]. SqueezeNet [1] is chosen to prune because of its modern architecture and fully convolutional nature (It consist of 2292 number of parameters and zero fully connected layers in SqueezeNet).

Table 1.2.
Number of parameters in the modern Convolution Neural Network architecture.

Network Architecture	Number of Parameters (M=Million)
VGG16	138M
Alexnet	61M
SqueezeNet	1.24M

The CNN consists of various layers such as the convolution layer, pooling layer, ReLU layers, and Fully Connected layer(FC). The convolution layers takes high computational power as compared to the other layers because of complex matrix multiplication. Several network compression techniques are available such as architectural compression, pruning [9], quantization and encoding techniques (Huffman encoding). However, many of these methods reduce the model size but results in a significant accuracy drop of the model [10]. In the pruning method, there are no proper ranking methods available to prune the unimportant parameters or convolutional filters from the model to reduce the model size. Moreover, in the case of fine pruning, there are many complexities to restructure the model due to network sparsity [11].

1.2 Contribution

This master thesis proposes an efficient methods to prune a CNN without a significant drop in the accuracy of the model. This thesis mainly focuses on pruning the entire convolution filters from the model as it takes an substantial computational power for inference. For real time application in embedded and low power devices the model size and inference time play a vital role. Therefore, getting faster and smaller CNN is essential for running these deep learning models on embedded devices. To compress the model, the proposed solutions uses coarse pruning techniques to reduce the model size. The pruning method uses various ranking methods to rank the unimportant or repentant parameters of the model.
Three different ranking methods are proposed to prune the SqueezeNet architecture.

1. Taylor expansion of cost value based ranking.
2. L2 normalization of activation maps based ranking.
3. Combination of above: Taylor expansion-based and L2 normalization of activation.

This thesis documents the comparison of these ranking methods for pruning the SqueezeNet model. The results shows that the combination of Taylor expansion-based and L2 normalization of the activation-based ranking method achieves more compression without a significant drop in the accuracy of the original model. The Transfer learning technique is used to train the SqueezeNet model on with CIFAR-10 dataset on PyTorch framework. By deploying this ranking methods, The proposed approach achieves 72% model reduction without drop in the accuracy of the SqueezeNet model. Finally, the pruned SqueezeNet model is deployed on specialized embedded hardware: BlueBox 2.0, and performance evaluated.

2. BACKGROUND

The concept of pruning the Neural Network has been in existence, since the 1990s. Yann LeCun et al. discussed, finding of the non-essential and redundant parameters that do not contribute to the output of the network. The second order derivatives of the Taylor expansion is used to find the importance of the neurons for their optimal brain damage research [12].

For a fully connected layer, Mariet and Sara (2016) discussed a method to reduce the network redundancy by identifying the essential neurons that do not require retraining. However, this method works only for fully connected layers [13]. To reduce the overhead of the convolution, Mathieu et al. (2013) used Fast Fourier Transform (FFT) based convolution [14], while Lavin and Gray (2016) discussed Winograd algorithms to reduce the convolution overhead [15]. Polyak and Wolf (2015) presented an idea to prune the non-correlated parameters or filters based on the activation frequency of the activation maps [16]. Sajid Anwar et al.(2015) discussed the complex pruning criteria, the particle score is calculated based on network accuracy and validation dataset. Iterative pruning takes place based on the ranking of these particle scores [17]. The work, "Pruning filters for efficient covnets" discuses pruning the entire convolution filters. The L_1 normalization of feature map is used as the pruning criteria [18]. However, work done on "Structured Pruning of Deep Convolutional Neural Networks" uses comparatively complex pruning criteria. The particle based masking techniques are implemented to prune the network. Based on the validation set and network accuracy, each particle received a score. Finally, based on the calculated masked particle score the filters are pruned iteratively [17]. Another approach to reduce the number of parameters by squeezing the model using the fire module is proposed by Pathak et al. Thus making the architecture hardware deployable [19]. This thesis discusses ways to find the nonessential filter maps from each convolution

layer and remove the entire feature map from the model to decrease the model size. To increase the performance of the architecture, non-correlated weights are removed from the CNN.

3. CONCEPTS

3.1 Introduction to Neural Network

Artificial Neural Networks (ANN) are inspired by the biological neural network system. It is a programming paradigm with which a system can learn from available observational data. Neural Network (NN) is formed by arranging the directed acyclic graph to form a feed-forward neural network. These neurons are grouped to form layers. Figure 3.1 shows the biological neurons, the main computational unit of the brain, these units are connected by synapse. In this system, the input signal comes via dendrites branches and the output signals are sent out via axon branches. The axon branches are further connected to synapses dendrites of other neurons. Artificial neurons get the input signal from x_0 of previous neurons, and then this signal is multiplied by weight w_i to simulate the interaction of dendrites. After that, the weighted input signals are summed up, and fixed bias is added and fed to the non-linear activation function which gives the output signal. Weights are adjusted according to labels of data to learn and approximate the inference.

$$y = f(P[x_i \times w_i] + b) \tag{3.1}$$

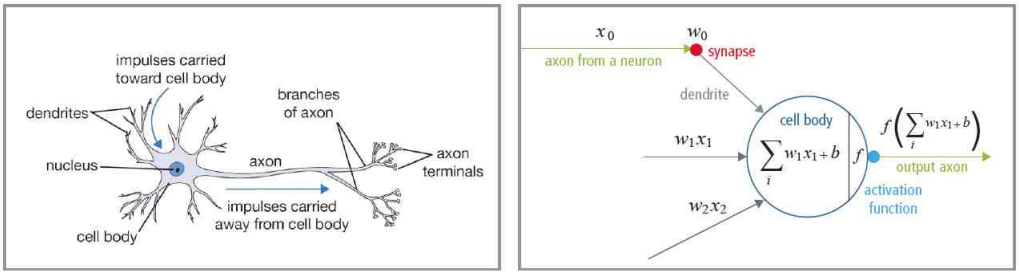

Fig. 3.1. Biological neuron (left) and its Mathematical model (right) [20].

Figure 3.1 shows the comparison between the mathematical model and the biological model. Artificial Neural network (ANN) is a system which is interconnected by neurons that can transfer messages from one node to another. The connection between neurons have weights, and they are fine-tuned in the training process. A Neural Network consists of many layers which has feature detector neurons. Figure 3.2 shows an example of a Neural Network with four Fully-Connected Layers, three Inputs and five Outputs.

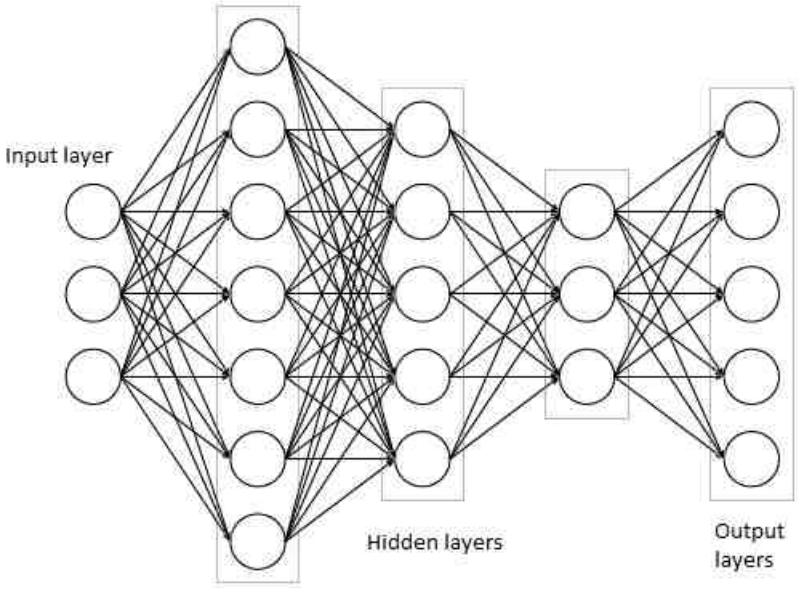

Fig. 3.2. Artificial Neural Network which has three layers: Input, Hidden and Output layers.

3.2 Network training

Neural Network (NN) consists of a large number of parameters. In NN, these parameters are not hard coded neither they are manually predefined. It learns the parameter values based on the labels provided during the training. There are various training approaches available: Supervised learning, Unsupervised learning and Reinforcement learning. In supervised learning, a function map is generated to achieve

the desired output, from the inputs which comprises of dataset (data and labels). Generally at the start of training, the network parameters are initialized by random values which are close to zero but not exactly zero. In the training phase, the data is passed one by one through the network (forward pass) and after that, the output of the forward pass is compared with the provided labels of the data and loss function. Loss function can be defined as the deviation of actual output from the ground truth or label. The Aim of this training process is to minimize the loss value by optimizing the network parameters. There are many optimization techniques available such as Stochastic Gradient Descent (SGD), Mini Batch Gradient Descent, Adagrad, Adam, Momentum, Adadelta. Stochastic Gradient Descent (SGD) optimization is used to train the SqueezeNet model. SGD uses gradient descent algorithms which calculates the gradient vector that specifies the contribution of weights towards the error. These gradients are calculated by backpropagation of the output through the network (backward pass). The SGD repeatedly passes the training data to model as forward pass and calculates the gradient vector (backward pass) and based on that, SGD adjusts the weight of parameters by a small amount in opposite direction of the gradient vector. This loop repeats until it converges to the global cost minimum. The magnitude of these update steps is called the learning rate. Figure 3.3 shows SGD, $J(w)$ = Cost function $J_{min}(W)$ = minimum cost function.

3.3 Introduction to CNN

Convolution Neural Networks are similar to general Neural Network, they have learnable weights and biases as shown in Figure 3.4. The main difference between the two is that, the CNN architectures only takes input as images. CNN is designed to extract the properties or features from the input images and encode it in the architecture which results in large parameter reduction compared to a normal neural network. The problem in the traditional Fully Connected Neural Network for an image is that each pixel of the image is mapped to one neuron and it is the Fully

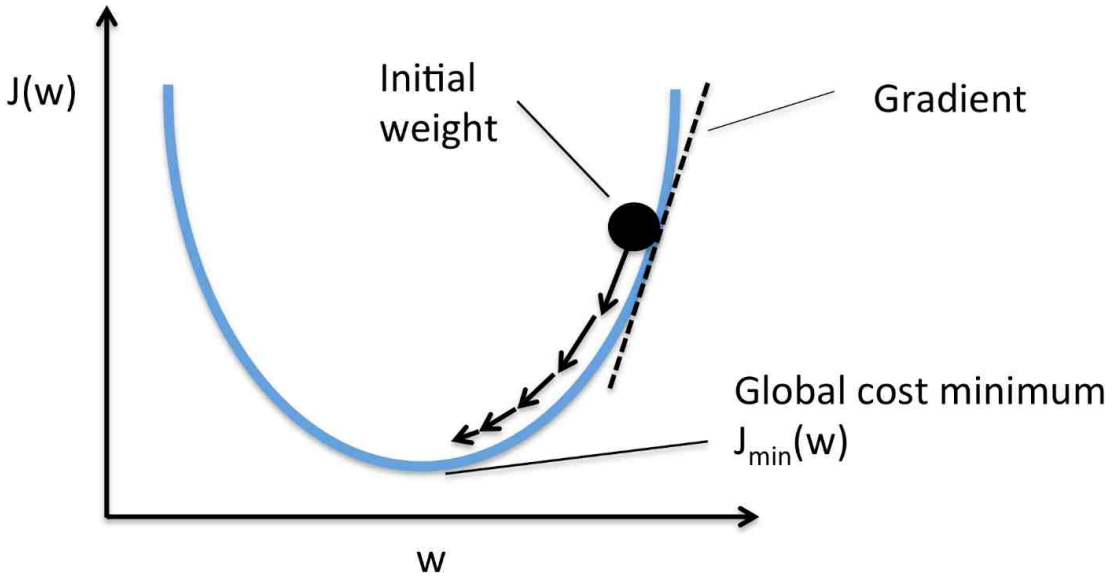

Fig. 3.3. Cost function of CNN

Connected (FC) layer which results in large number of parameters and large model size, as shown in Figure 3.5.

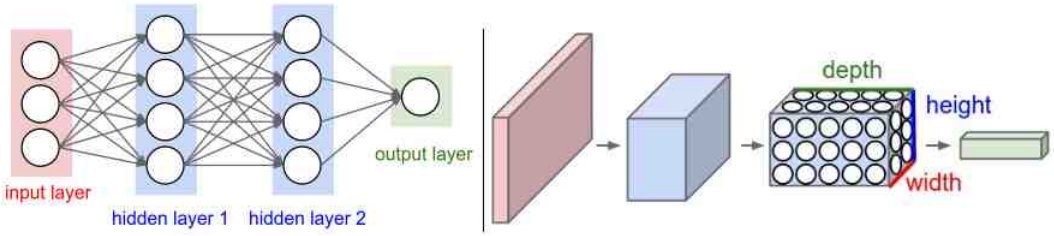

Fig. 3.4. Comparison of NN and CNN.

If there is an image with size 200×200 and 40,000 hidden parameter in Fully Connected Neural Network (FCNN). This will result in around two billion parameters. There are not enough resources for the computation of such large number of parameters because of the resulting model size (generally large in size). Even if there is use of a locally connected layer, there will be a still significant number of parameters to

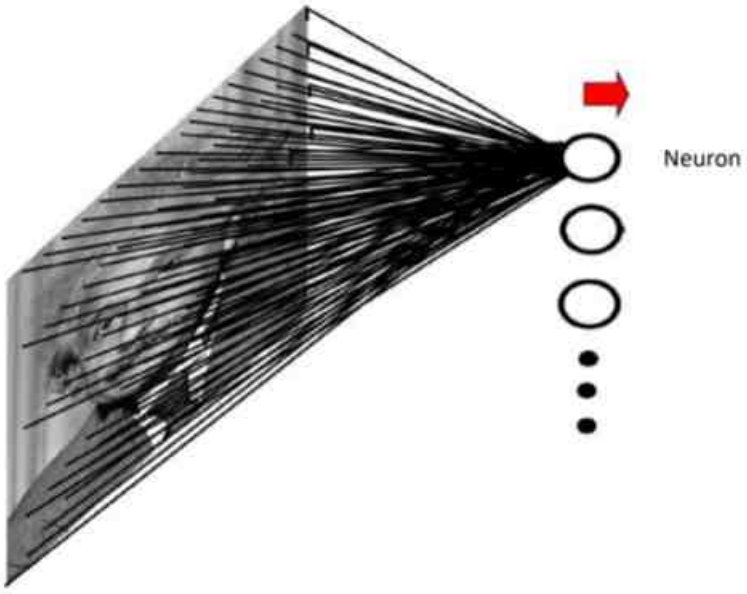

Fig. 3.5. Fully Connected NN for Images.

train the model, and this will affect the model size as well as computation cost. The solution of these problem is to use the convolution layer.

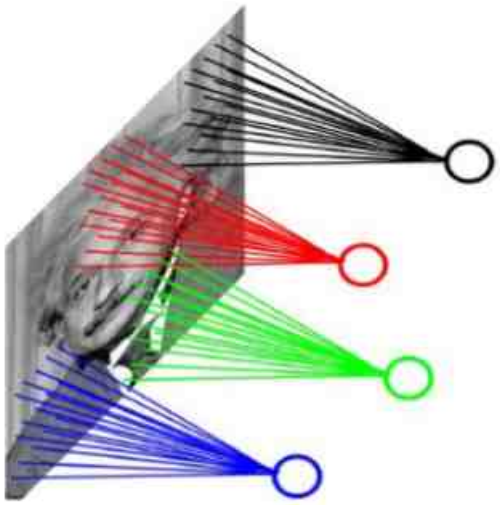

Fig. 3.6. Locally Connected NN for Image

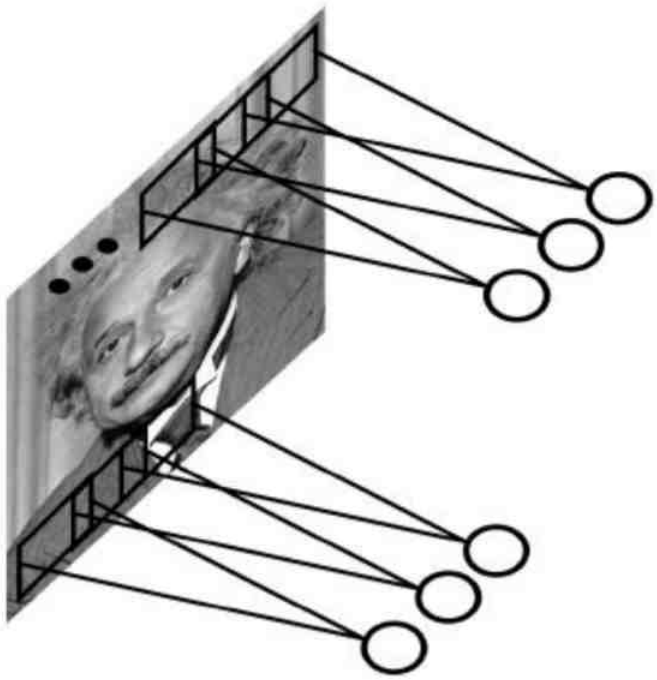

Fig. 3.7. Convolution layer for Image

In CNN, instead of finding the correlation of one pixel to all other pixels, CNN tries to learn the hierarchy of local and spatially independent features (low-level independent features). This approach helps to reduce the computation cost and parameter reduction.

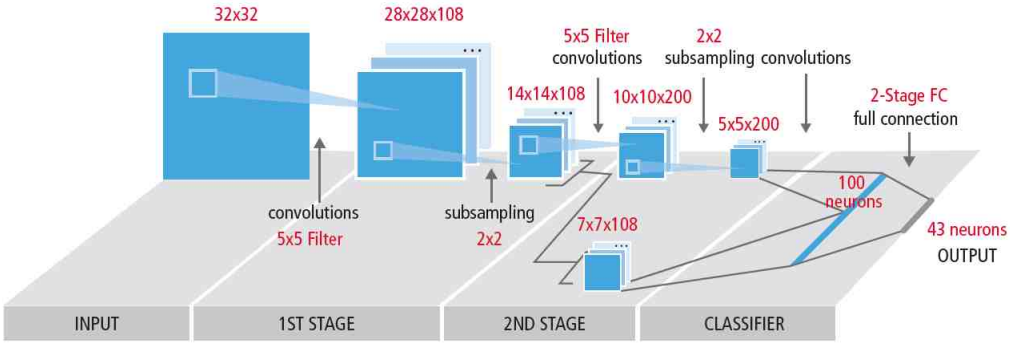

Fig. 3.8. CNN classifier example [20].

Convolution Neural Network consists of various types of generic layers, which transforms the input feature map ($I_h \times I_w \times I_{ch}$) into output feature map ($O_h \times O_w \times O_{ch}$). Important building blocks of CNN are as follows:

1. Convolution layer
2. Nonlinear layer
3. Pooling layer
4. Fully connected layers
5. Normalization layers
6. Dropout layers

3.3.1 Convolution layer

Convolutional layers take many feature maps as input and produce N feature maps as output. N is the number of convolutional kernels in convolution layer.
The convolution layer has following types of hyperparameters:

1. Kernel size ($k_w \times k_h \times d$)
2. Number of kernels N or depth of output
3. Stride S,
4. Padding P
5. Activation function

1. Kernel size ($k_w \times k_h \times d$)

The kernels in convolution layers are learnable parameters which are updated in the training phase based on labels of the training dataset.

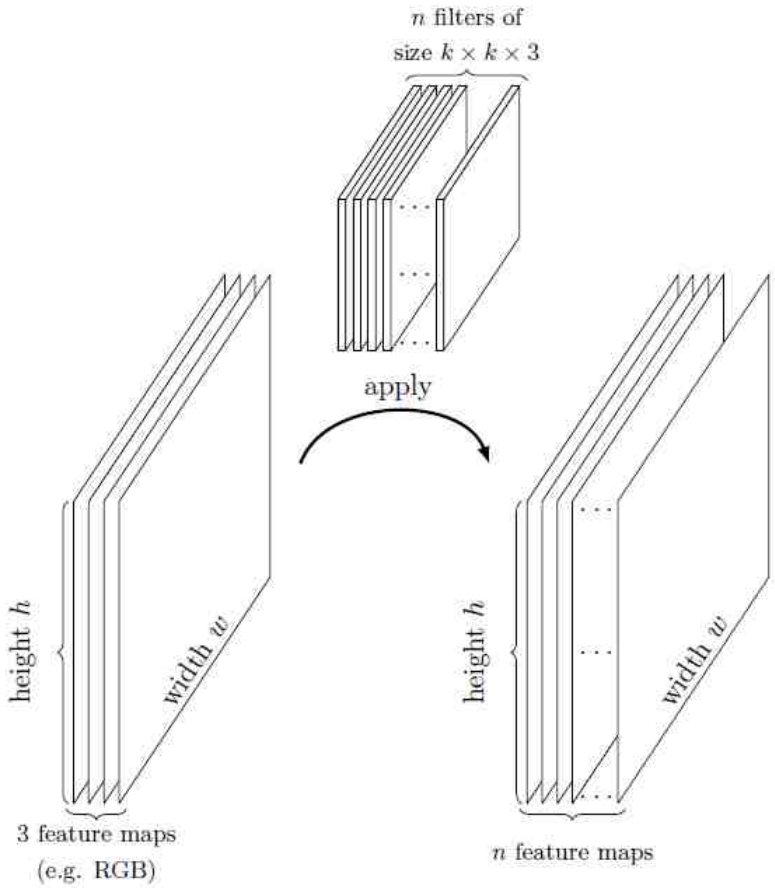

Fig. 3.9. Application of a single convolutional layer with N filters of size $k \times k \times 3$ with stride $S = 1$ to input data of size width height with three channels.

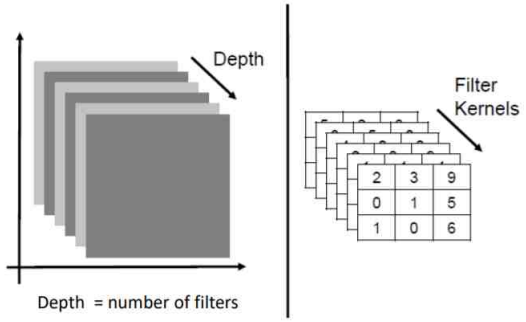

Fig. 3.10. Kernels in convolution layer.

2. Number of kernels N

The number of kernels N is used to define the number of learnable kernels in a convolution layer. This value also corresponds to the output volume.

3. Stride S

The stride number describes the kernel sliding by the number of pixels, of the images. For example, If stride is one, then the kernel is moved one pixel at a time. It controls the convolving of the kernel with the input features.

4. Padding P

The padding P is a process of symmetrically adding any number to the input image matrix to preserve the feature. It is a modification method to adjust the input size depending upon the requirement. Example: Zero padding.

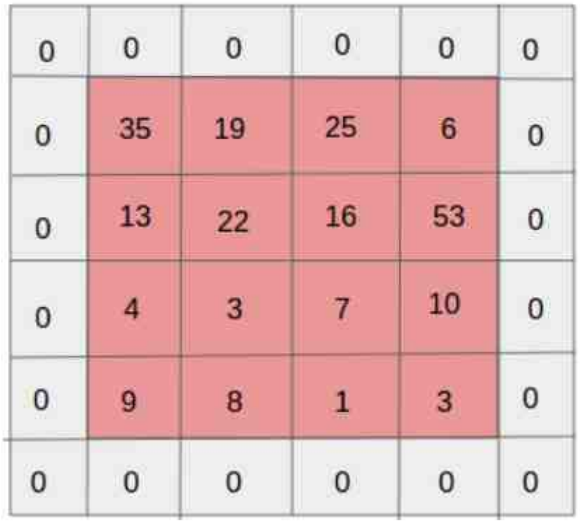

Fig. 3.11. Zero Padding

5. Activation function

It is a function which is used to introduce the non-linearity in the model. This function helps to train the model faster and more accurately, this also helps to reduce the overfitting of the model on the training dataset. Rectified linear unit (ReLU) is one of the most widely used activation functions in CNN. It converts all negative numbers to 0 and keeps the positive number unchanged. The output volume of convolution depends on the four parameters: input size (W), Kernel size (F), stride (S) and padding (P).

$$\text{Convolution output volume} = \frac{(W - F + 2P)}{S} + 1$$

3.3.2 Nonlinearity layers

These layers are used to introduce the non-linearity in the model. This helps to train the model faster and more accurately, and it helps to reduce the overfitting of the model on the training dataset. Rectified linear unit (ReLU) is one of the favorite activation functions used in CNN. Figure 3.13 shows a ReLU function which converts all negative numbers to 0 and keeps the positive number. ReLU helps to reduce the computational complexity, and it helps in convergence while training.

3.3.3 Normalization layer

Normalization layer is used to normalize the responses from the adjacent output channels. It normalizes the output distribution to zero mean with unit variance.

3.3.4 Dropout layers

These layers are very popular in the training phase. These layers introduce the non-linearity in the model by randomly dropping a certain percentage of connection

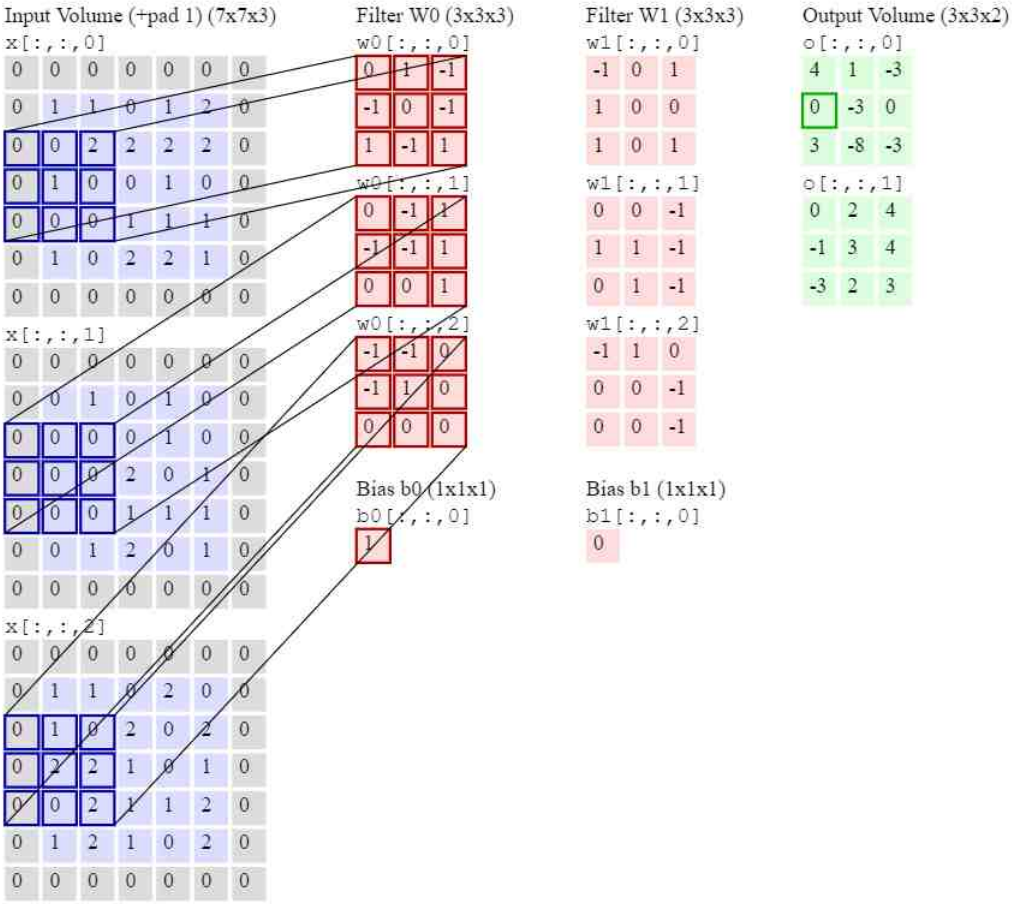

Fig. 3.12. Convolution example with following parameters $W = 7, K = 2, F = 3, S = 2, P = 1$ [21]

during training. This method avoids in learning of the redundant filters and thus combats the overfitting of the large CNN.

3.3.5 Pooling layers

In CNN, it is used to downsample the input features with non-linearity. It reduces the resolution of the input feature map but keeps the features intact. There are two popular choices of pooling.

1. Max pooling.

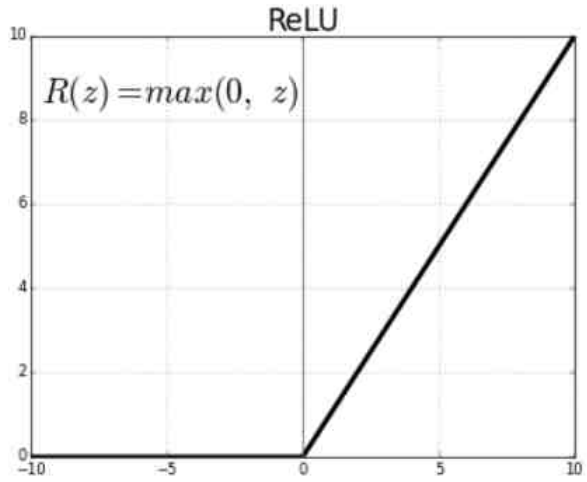

Fig. 3.13. Rectified linear Unit function (ReLU)

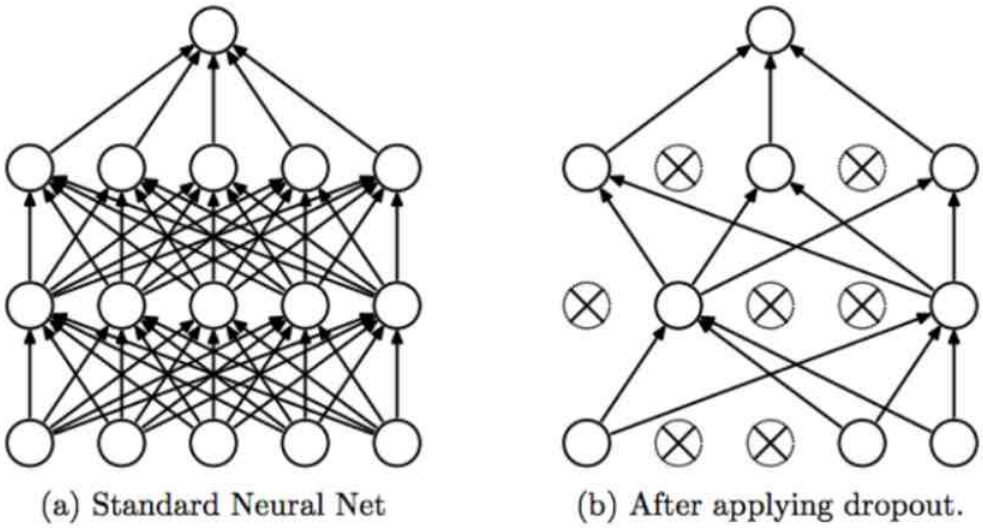

Fig. 3.14. Dropout layer

2. Average pooling.

In Figure 3.15, the input volume $224 \times 224 \times 64$ is pooled with kernel size 2×2 and with stride 2, which produces the output volume of size $112 \times 112 \times 64$ in this case the output depth is preserved.

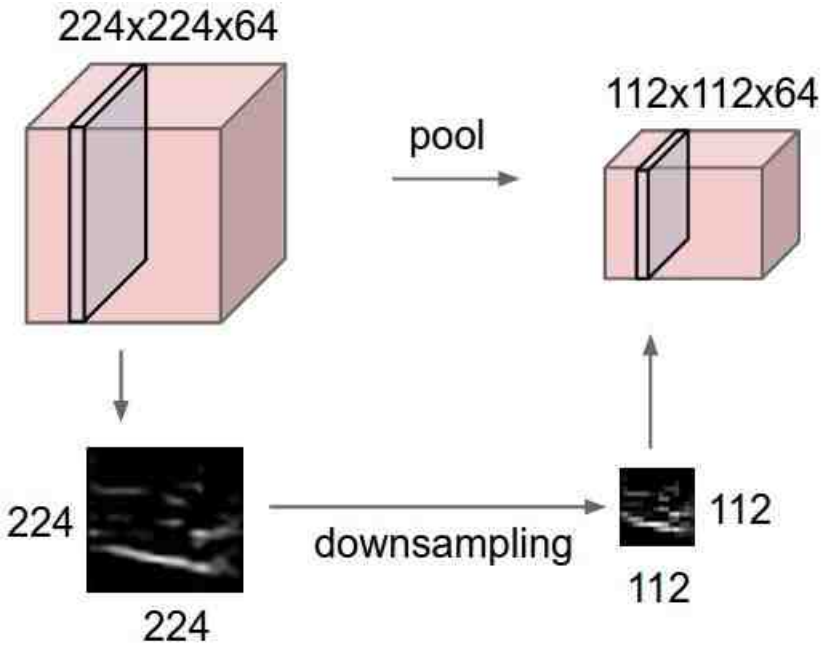

Fig. 3.15. Pooling layer

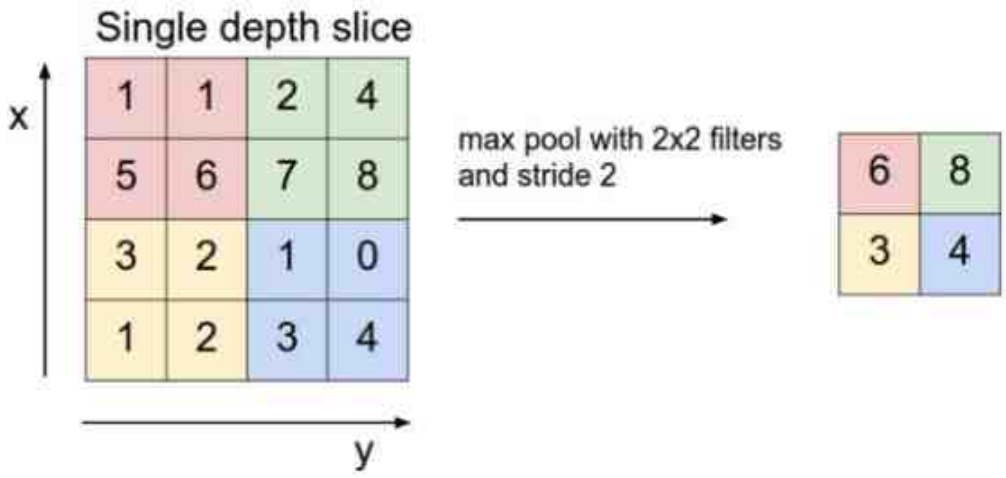

Fig. 3.16. Maxpool with kernel size = 2 and stride = 2

3.3.6 Fully connected layers (FC)

These layers are often used as the last layers in the CNN to compute the class score of the classification. This layer has a full connection to all previous activations. FC layers act as a classifier on the top of high-level features.

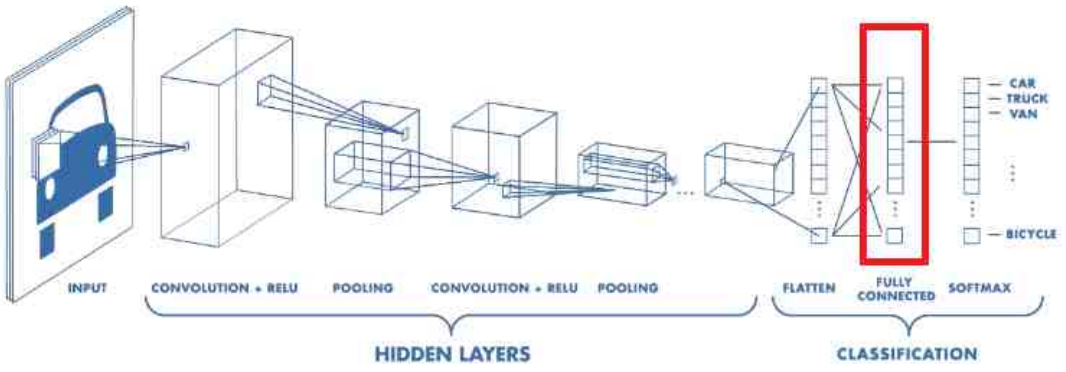

Fig. 3.17. Fully connected layer in CNN architecture.

3.4 Benchmarked CNN architectures

In this section, some popular CNN architectures are discussed. The section also describes the reason for choosing the SqueezeNet architecture for pruning.

3.4.1 LeNet

LeNet is one of the first successful applications of convolution neural network for classification of digits to read the zip code.

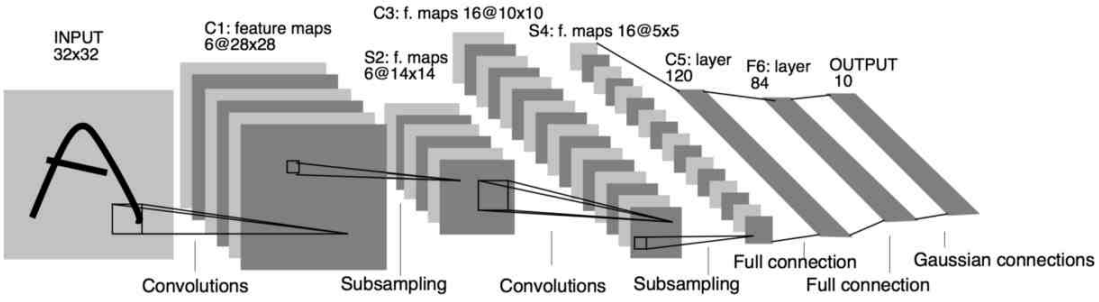

Fig. 3.18. LeNet-5 architecture [22].

3.4.2 AlexNet

AlexNet is one of the famous CNN architectures for ImageNet ILSVRC challenges in 2012 AlexNet outperforms all CNN architectures with top 5% error is only 16%. AlexNet uses ReLU instead of *tanh* to add non-linearity, which accelerates the speed up to six times. It uses dropout layers instead of regularization to deal with overfitting of the network with a dropout rate of 0.5.

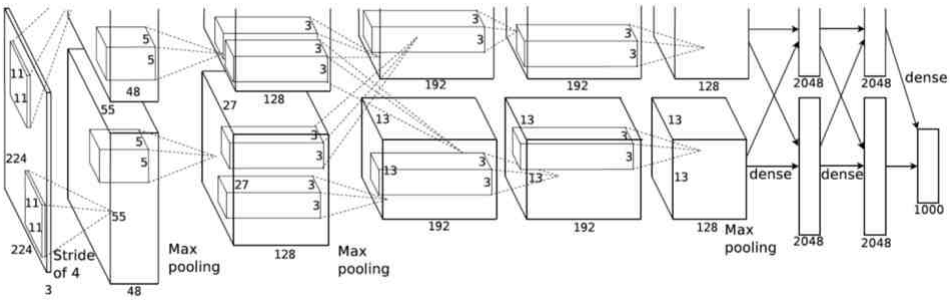

Fig. 3.19. AlexNet architecture.

3.4.3 GoogLeNet

This CNN architecture is by Google. It was the winner of ILSVRC 2014 challenge. The main highlight of this architecture was the inception model which dramatically reduced the number of parameters of the model. It has 4 million parameters compared to AlexNet with 60 Million. It uses average pooling instead of fully connected layers at the top layer. The most recent version of GoogLeNet is inception-v4.

3.4.4 VGGNet

This architecture was runner-up in ILSVRC 2014 from Karen Simonyan and Andrew Zisserman. This model proved that the depth of the network plays a vital role in the performance of the network. It contains 16 convolution layers which are very

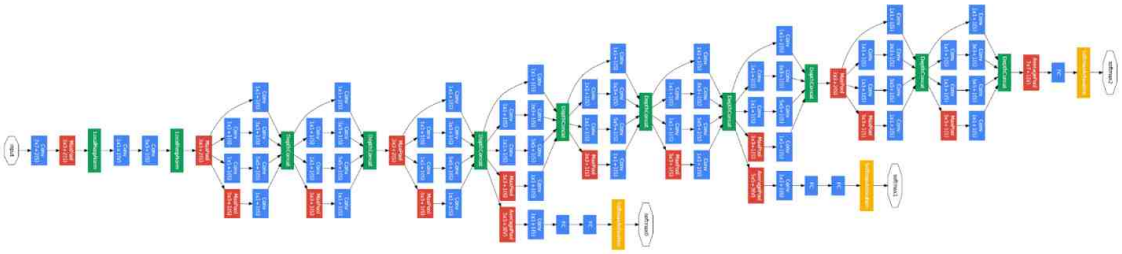

Fig. 3.20. GoogLeNet Network [2] (From Left to Right)

deep, and it has homogenous architecture throughout the model, it only performs 3×3 convolution and 2×2 pooling from beginning to end. It contains 140 million parameters, and it is computationally expensive [3].

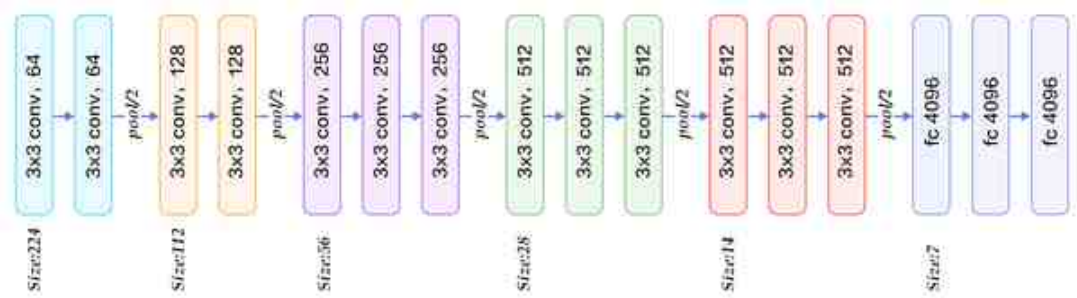

Fig. 3.21. VGG16 Architecture (From Left to Right)

3.4.5 SqueezeNet

This is a small CNN architecture which achieves the AlexNet-level accuracy on ImageNet with 50× fewer parameters [1]. This type of small architecture is beneficial for efficient distributed training; it has very little overhead while exporting new models to clients and it is feasible for FPGA and embedded deployment.

In the SqueezeNet paper [1], the author outlines three main strategies:

1. Replace 3×3 kernels with 1×1 for smaller network: To reduce the network parameters, they used 1×1 filters instead of 3×3. Since 1×1 kernels has $9\times$ less parameters than 3×3 kernels.

2. Decrease the number of input channels to 3×3 kernels: Using the 1×1 kernels as bottleneck layer called squeeze layer as shown in the Figure 5.5 to reduce the depth of the model and reduce the computation for the following 3×3 kernels.

3. Downsampling later in the network to keep the large activation maps to preserve the feature maps: The intuition is that, the large activation map is kept due to delayed downsampling in the network, which results in higher classification accuracy.

To implement the above strategies Iandola, et.al. [1] used fire module in the SqueezeNet architecture, which contain two types layers.

1. Squeeze layer

2. Expand layer

Fire module comprises two types of convolution layers: 1×1 filter size (squeeze layer) and $(1 \times 1 + 3 \times 3)$ filter size (expand layer). 1×1 (point-wise) filters are used as the bottleneck layer instead of 3×3 filters to reduce the computation [1].

3.5 Datasets

This section describes the datasets that were experimented with the network model. Dataset is essential to benchmark the network model performance after training.

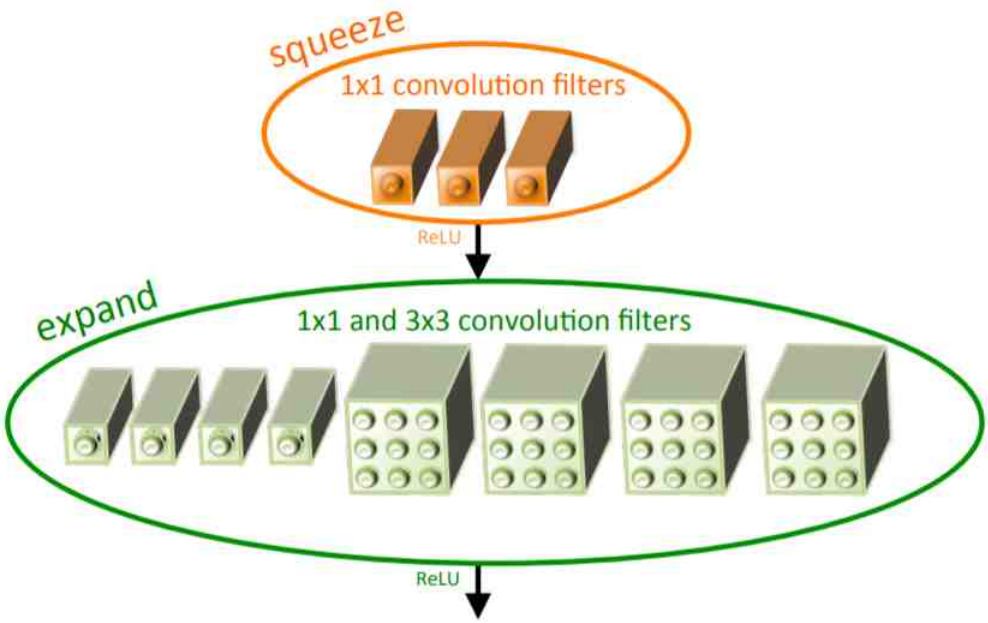

Fig. 3.22. Fire layer of SqueezeNet Architecture [1].

3.5.1 MNIST

This Dataset consists of 6000 training images and 10000 test images. Each image is 28×28 black and white image with only one channel. The classes of this dataset are a handwritten digit from 0 to 9.

3.5.2 CIFAR-10

This dataset consists of $50,000$ training images and $10,000$ testing images. Each image has size 32×32 of RGB color with ten different classes. The classes are cat, deer, dog, frog, horse, ship, airplane, automobile, bird, and truck.

3.5.3 ImageNet

ImageNet is an image database organized by WordNet hierarchy. It contains on an average of five hundred images per node. It is designed for visual object detection

and classification for research. ImageNet has around 20 thousand different image classes. Generally, to benchmark any CNN architecture researchers use the ImageNet as a standard dataset. The most popular ImageNet Large Scale Visual Recognition Challenge (ILSVRC) uses ImageNet with 1000 classes.

4. COMPARISON BETWEEN DIFFERENT DEEP LEARNING FRAMEWORK

In this chapter, different kinds of framework for deep learning are discussed, which can provide a different level of abstraction depending upon the difficulty of the programming languages and application. Each framework has an advantage and purpose. PyTroch 4.0 version on Python 2.7 is used to train and generate the SqueezeNet model on CIFAR-10 dataset and for implementing the pruning algorithm and the same configuration are used.

Fig. 4.1. Deep learning framework logos [23] [24] [25] [26] [27].

4.1 TensorFlow

TensorFlow is one of the best deep learning frameworks from Google, which is used widely by researches from Google, IBM, Twitter etc. It is popular because of its highly flexible architecture. TensorFlow supports Python, C++, and R with wrapper libraries. It comes with two highly useful tools.

1. Tensor Board:
   It is mainly used for data visualization of network architecture and performance.

2. TensorFlow:
   It is used for rapid prototyping of models, and it has various standard API that are used to design modular networks.

4.2 Keras

Keras is another widely used deep learning framework for fast prototyping. It supports the only python as the programming language with TensorFlow or Theno as backend support. Keras is part of TensorFlow's core API, and it is primarily used for classification, speech recognition, and text generation and summarization.

4.3 Caffe

It is famous for its high speed, transportability, and modularity. It was developed by Berkeley Artificial Intelligence Research. It is mainly used for CNN. It is famous for the vision recognition algorithm. Caffe uses libraries which are written in low-level languages such as C++. Caffe is popular in research industries for its quick prototype deployment. It can process over 60 million images per day with a single NVIDIA K40 GPU. Caffe supports C, C++, Python, and MATLAB and command line interface.

4.4 Theano

Theano was created in 2007 by Yoshua Bengio and the research team at the University of Montreal. It was one of the first widely used deep learning frameworks. It contains a highly optimized python library which is extremely fast, but it is not very popular because of its complex and low-level interface.

4.5 PyTorch

PyTorch is an open source deep learning framework which is developed for GPU based algorithms. It was developed by Facebook, and is based on the scripting language Lua which is flexible and easy to build models. PyTroch is popular because of its dynamic computational graph and efficient memory usage. Modular prototyping can be performed using the object-oriented programming (OPPS) concept. Other framework uses static graphs which sometimes takes time to compute, but PyTorch use a dynamic graph which helps with debugging and faster processing.

5. TRAINING THE SQUEEZENET WITH THE CIFAR-10 DATASET ON PYTORCH

5.1 Transfer learning

The common practice used by researches is not to train the Convolution Neural Network from scratch (with random initialization), because training the efficient and accurate model is a tedious task as it takes significant of time and computation power to train any network from scratch. Another major issue is non-availability of huge labeled datasets. The conventional approach is to use a pre-trained model for any specific application. This type of approach is called transfer learning. There are mainly two types of transfer learning approaches:

1. CNN as fixed feature extractor: In this approach, a pre-trained model is imported (Example CNN trained on ImageNet with 1000 classes). The preferred method is to freeze all the layers of the CNN except the last layer. These layers are considered as fixed feature extractor. The last layer is modified depending upon the application and the datasets. The last layer is fine tuned and thus model is deployed.

2. Fine -tune the pre-trained model: In this approach, a pre-trained model is imported and it is retrained with different datasets. All the weights are fine-tuned by continuing the backpropagation with a new datasets.

Transfer learning is efficient as many low-level features are common in lot of classification tasks. Transfer learning saves a lot of computation power and time for training. Observation suggests that the pretrained model performs better than the model trained from scratch. There are few things to take care while doing transfer learning. The pretrained model which was trained on the different dataset cannot be

used because there will not be any common features in two totally different datasets. Moreover, the learning rate while retraining the model should be small.

5.2 Training the SqueezeNet

In this experiment, The SqueezeNet was trained from scratch (with random initialization) with a CIFAR-10 dataset on Pytorch framework. The accuracy of the model after 110 epochs was 64% as shown in Figure 5.1. The Figure 5.5 shows the modified SqueezeNet architecture used for training with the CIFAR-10 dataset. On the other hand, the pre-trained SqueezeNet model was imported, which was trained on ImageNet with 1000 classes. The pre-trained model was modified by removing the last convolution layer, and finally, the application-specific convolution layer was added. The CNN as a fixed feature extractor approach was used to retain the model, and 74% network accuracy was achieved as shown in Figure 5.2.

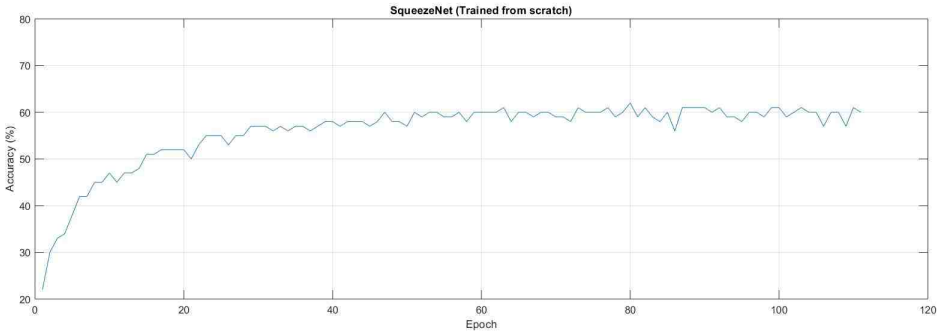

Fig. 5.1. SqueezeNet model accuracy - trained from scratch on CIFAR-10 dataset.

Figure 5.3 shows the training loss of SqueezeNet model which is trained from scratch with the CIFAR-10 dataset and Figure 5.4 shows the training loss of SqueezeNet model which is trained from pretrained model with the CIFAR-10 dataset.

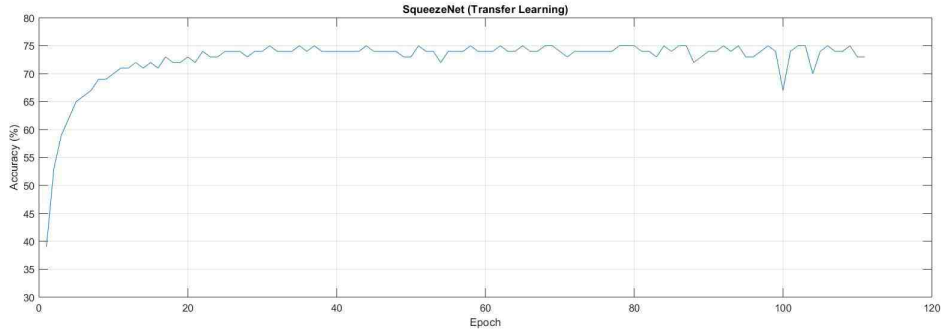

Fig. 5.2. SqueezeNet model accuracy - (Pretrained model- ImageNet) on CIFAR-10 dataset.

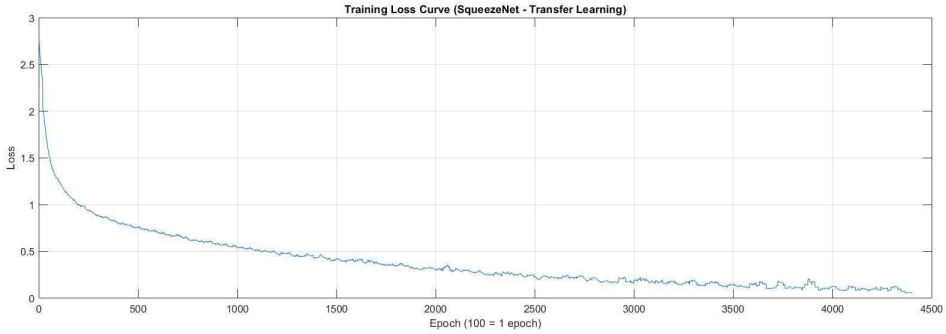

Fig. 5.3. Training loss for SqueezeNet model - (Trained from scratch)

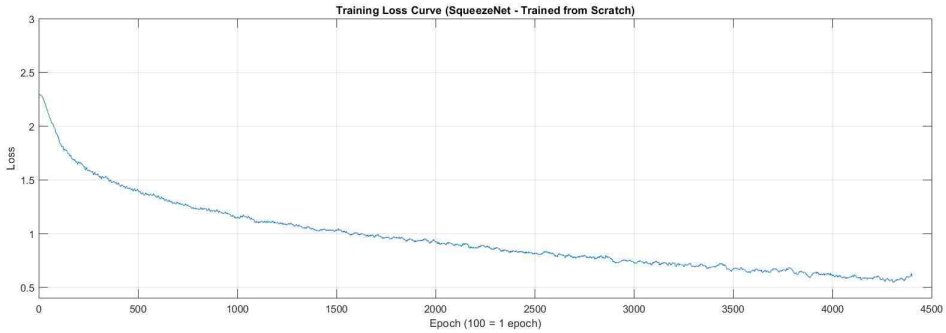

Fig. 5.4. Training loss for SqueezeNet model - (Pretrained model)

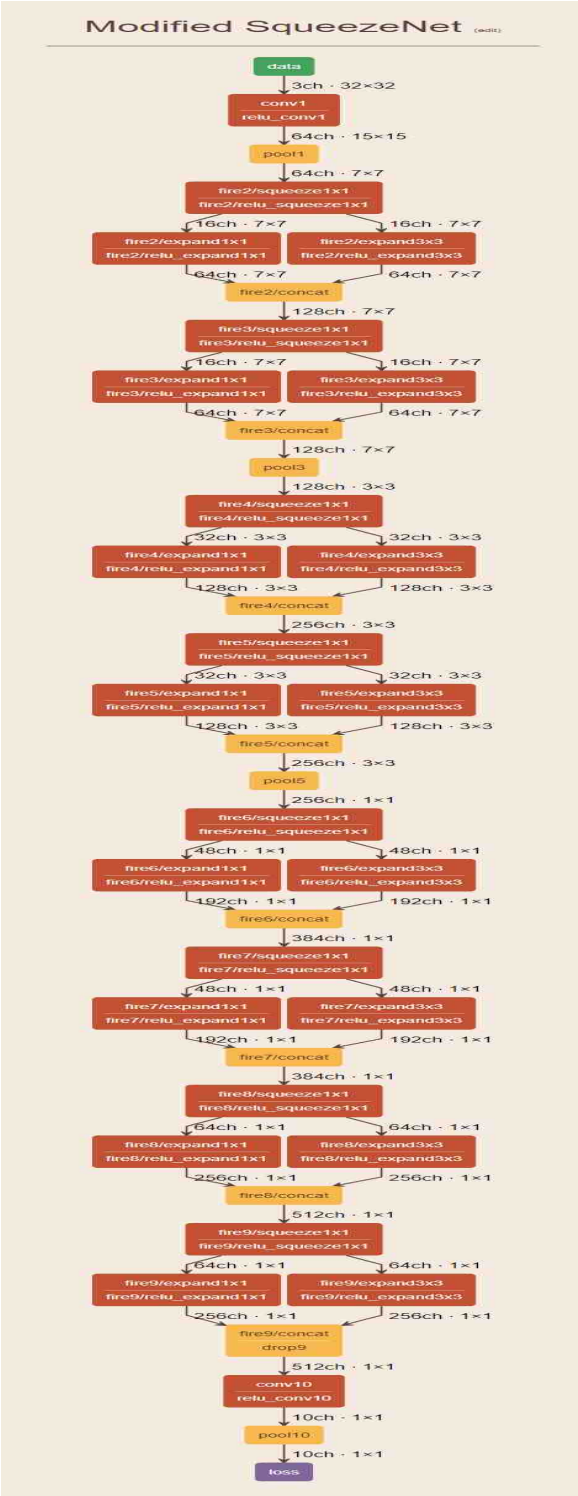

Fig. 5.5. Modified SqueezeNet architecture for CIFAR-10 dataset

6. PRUNING METHODS

The previous chapters discusses the building blocks to generate and train the CNN model. This chapter, introduces some methods to reduce the model size and computation cost without affecting the accuracy of the model. The implementation, experiments and pruning results are discussed. Pruning is one of a the methods that can compress the model. The main idea of pruning is to find unimportant parameters or redundant parameters that do not contribute towards the output of the model. After finding these parameters, pruning takes place and model is restructured. Finally, the model is checked for any accuracy drop, and if there is any drop, the model is retrained using the same dataset to recover the accuracy drop. This chapter discusses a method to find the uncorrelated filters or kernels from each convolution layer and remove the entire feature map from the model to increase the memory, computation and power efficiency.

Consider a case for CNN training, (Example - Vehicle classification.) in Figure 6.1, a small CNN architecture is used to classify the vehicle. Due to a limited number of parameters, small CNN can't extract enough features from the image which will results in low accuracy however the model size will be small. On the other hand, if massive CNN architecture is used for the same classification, the accuracy of the model will be good due to large number of features, but the model size will be huge, as shown in Figure 6.2. To solve this problem, Pruning compression technique can be used. This approach can help to keep proper trade-off between accuracy and model size, as shown in Figure 6.3.

6.1 Types of Pruning

There are mainly two types of pruning:

1. Fine Pruning:
   As shown in Figure 6.4, this type of pruning is implemented by zeroing out the entire feature map or removing the connections between feature maps. This type of pruning may require software or hardware acceleration.

2. Coarse Pruning:
   As shown in Figure 6.5 and 6.6, the entire feature map is removed from the model to increase the computation speed to get a faster inference and a reduced model size.

This thesis concentrates on Coarse pruning, where the entire feature map or kernel is removed from the convolution layer to avoid the network sparsity. Identifying and pruning the right uncorrelated parameters from the network and then pruning them is a crucial task to improve the computation and power efficiency of the model. To preserve the accuracy of the model with a reduced model size, three proposed methods are discussed in this thesis:

1. Pruning based on Taylor expansion of the cost function.

2. Pruning based on L2 normalization of activation maps.

3. Pruning based on a combination of the two above mentioned pruning criteria.

Figure 6.7 show the proposed steps for the pruning model:

1. Choose any standardized CNN architecture and dataset based on specific application.

2. Train the network either from scratch or use a pretrained model (use of a pretrained model is recommended), in this case the pretrained SqueezeNet model was used, which was trained on ImageNet with 1000 classes.

3. Find the important convolutional kernels based on proposed pruning criteria.
4. Rank the kernels according to proposed algorithm.
5. Remove these low ranked noncontributing kernels from the model.
6. Check for any accuracy drop and retrain the model to recover this accuracy drop, if necessary.
7. Check the trade-off between accuracy and the pruning objective and decide if further pruning is required.

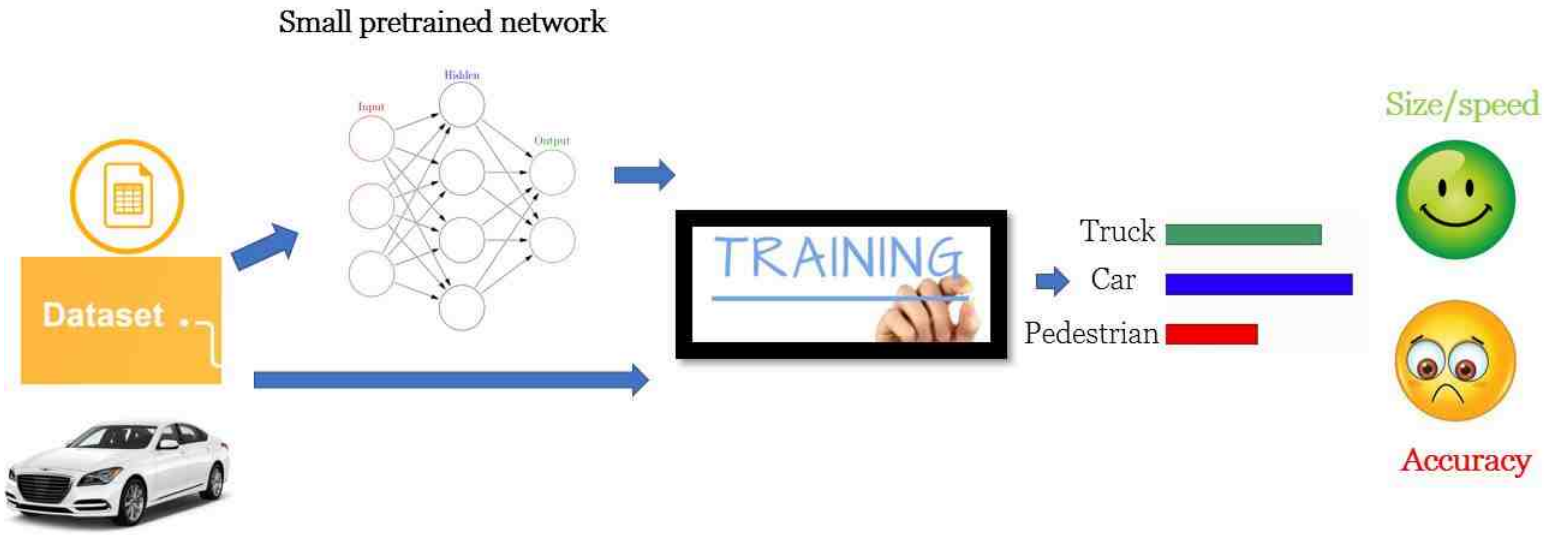

Fig. 6.1. Vehicle classification with small CNN.

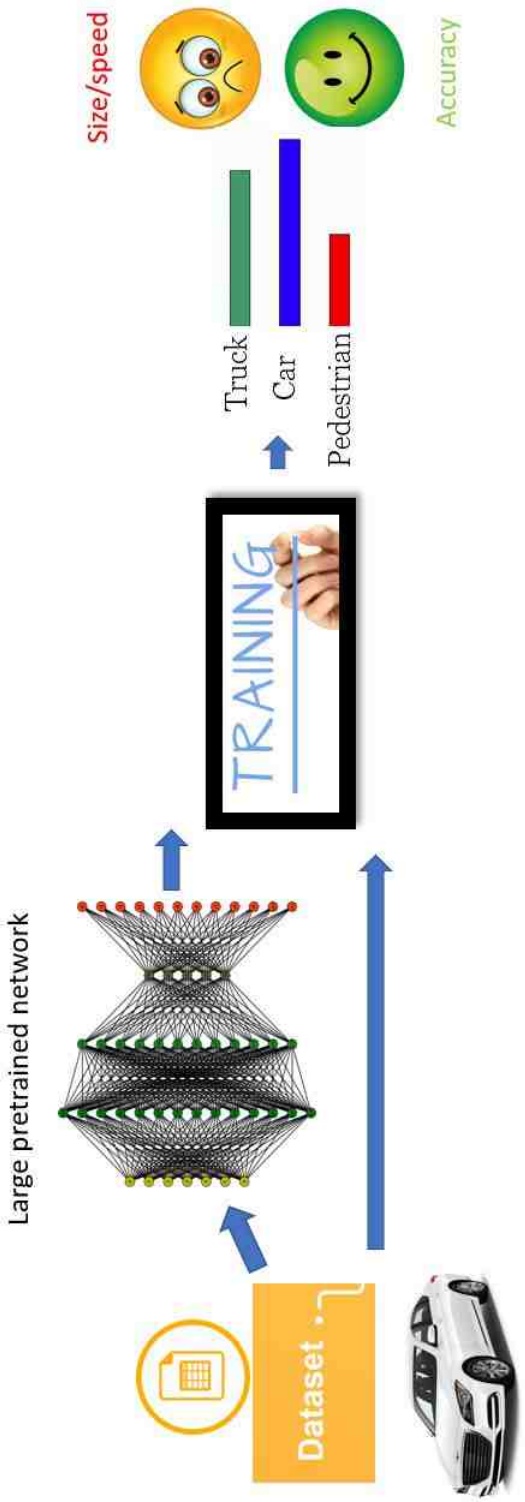

Fig. 6.2. Vehicle classification with large CNN.

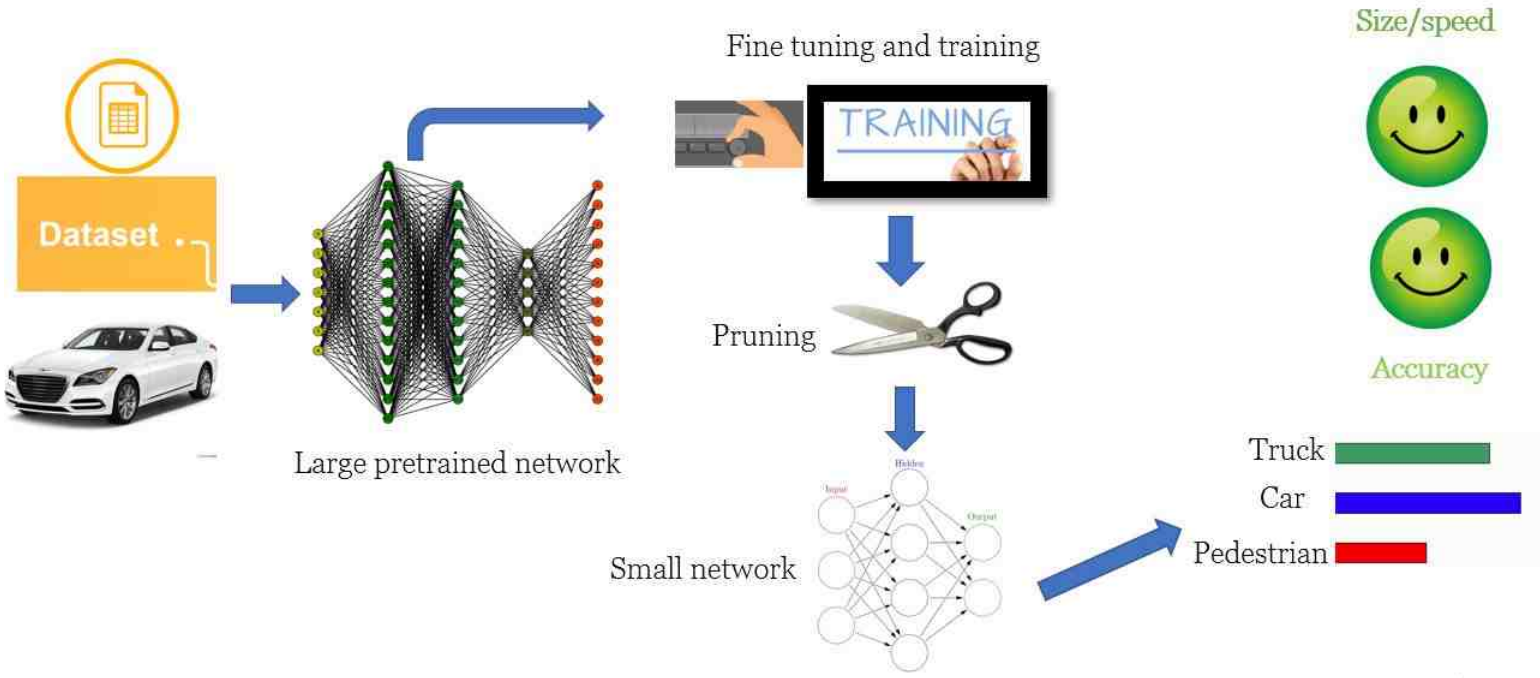

Fig. 6.3. Vehicle classification with large CNN and pruning.

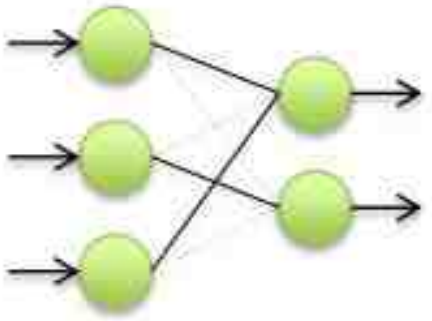

Fig. 6.4. Fine Pruning.

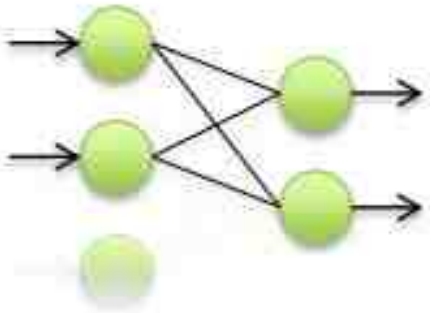

Fig. 6.5. Coarse Pruning.

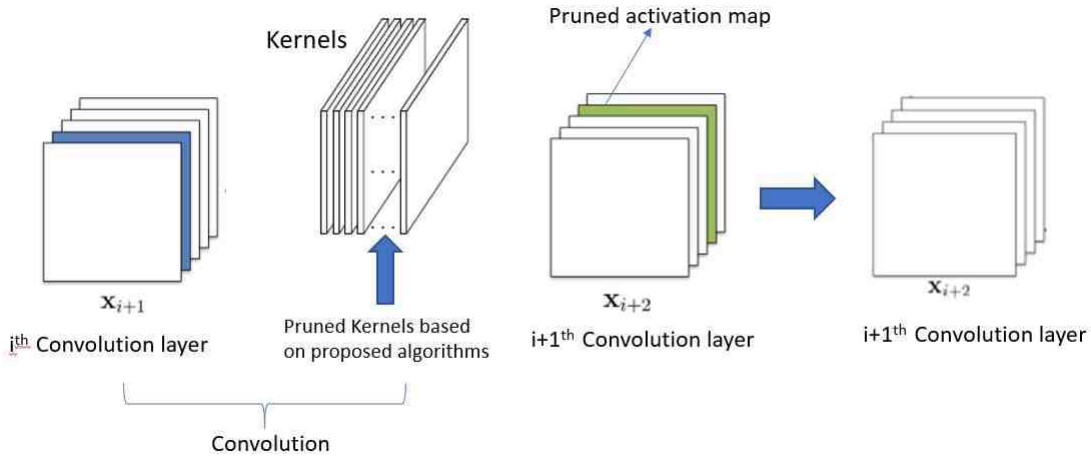

Fig. 6.6. Coarse Pruning.

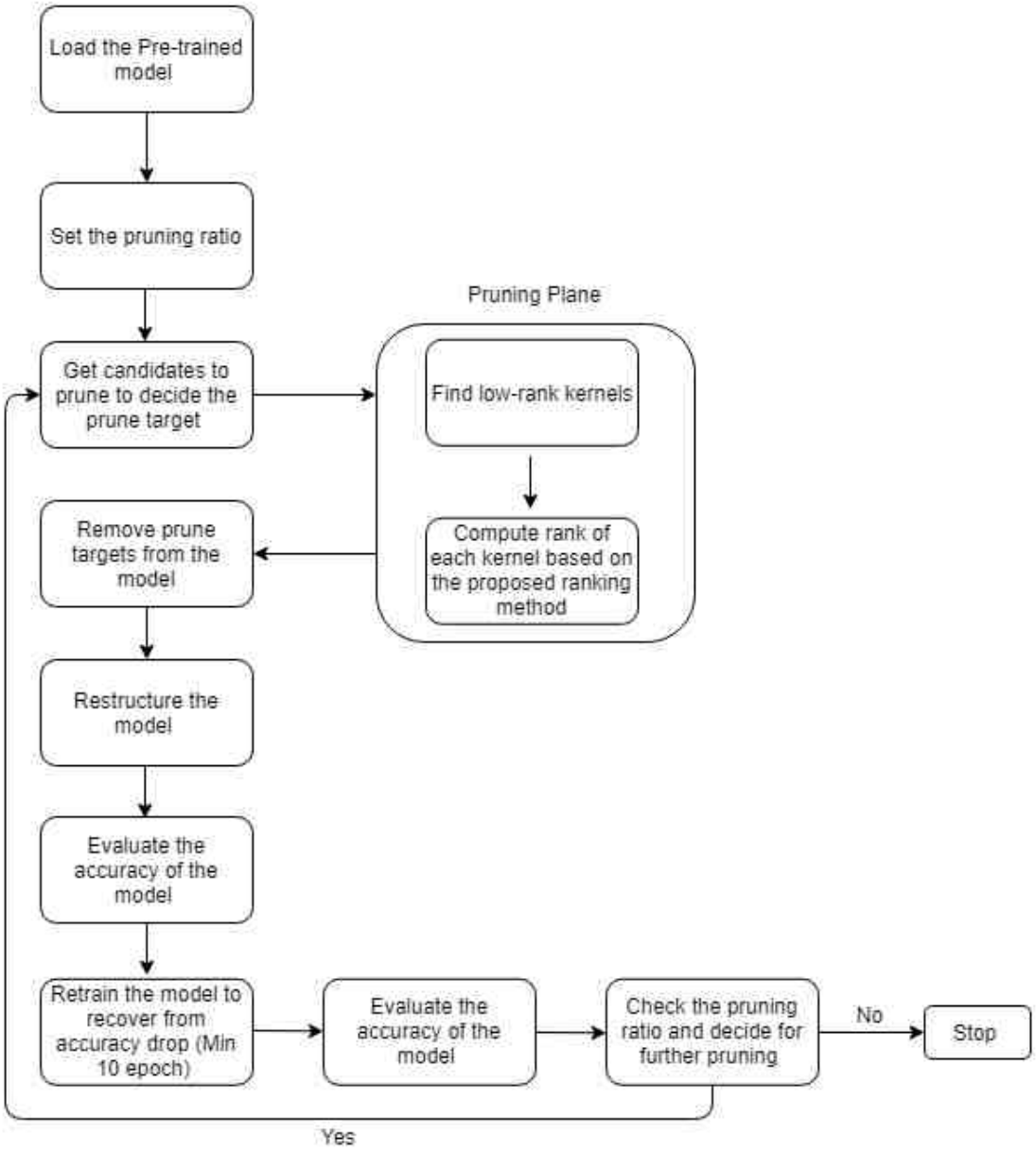

Fig. 6.7. Pruning steps to reduce the model based on pruning criteria.

7. PRUNING BASED ON L_2 NORMALIZATION OF ACTIVATION MAPS

7.1 Implementation algorithm

This chapter discusses a method to find the uncorrelated filters or kernels from each convolution layer and remove the entire feature map from the model to increase the memory, computation and power efficiency, and thus further focuses on the coarse pruning, where the entire feature map or kernel from the convolution layer is pruned to avoid network sparsity. As shown in Figure 7.2, this approach uses activation-based features map pruning. First step is to identify the weak activation filters.

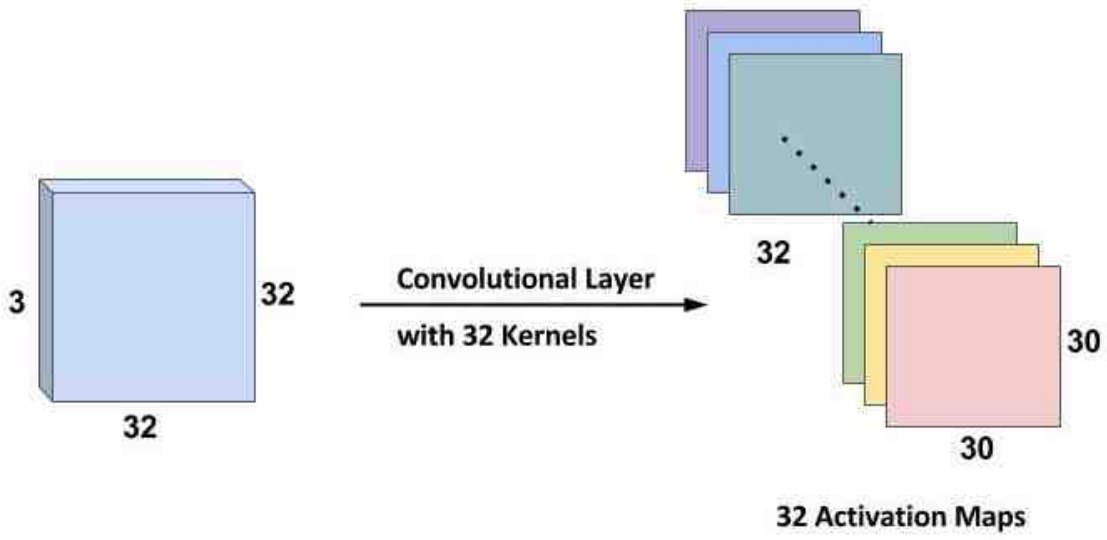

Fig. 7.1. Activation maps generated by convolution

Table 7.1.
Sensitivity to pruning the SqueezeNet model on CIFAR-10 dataset.

Pruning Ratio (%wise)	Accuracy of model
0	74
5	74
10	74
15	74
20	74
25	74
30	74
35	74
40	74
45	74
50	74
55	74
60	74
65	74
67	73
70	70
75	72
80	63
85	58
90	52
95	42
100	0

In Figure 7.1, the activation maps A_i is generated by convolving with the kernel.

$$F_i \circledast C_i \rightarrow A_i$$

$$F_i(h_i \times w_i) \circledast C_i(h_i \times w_i) \rightarrow A_i(h_i \times w_i)$$

$F_i(h_i \times w_i)$ = Input feature map with height h and width w.
$C_i(h_i \times w_i)$ = Convolution kernel height h and width w.
A_i = Activation map generated by convolving input feature map and convolution kernel.
This leads to L_2 normalization of each activation in each layer

$$L_2 \text{ norm is} \rightarrow ||A_i||_2 = \sqrt{\sum_{i=1}^{n} A_i^2} \tag{7.1}$$

The L_2 norm of each activation is calculated and corresponding convolutional filters are ranked based on the L_2 norm of the activation map. Finally, the lower ranked filters or kernels are pruned iteratively.

Figure 4 shows the steps for pruning are as follows:

1. The Network was trained from scratch or a pre-trained network is used (recommended). In this paper pre-trained SqueezeNet which was trained on ImageNet was used. The last layer is removed and a CIFAR-10 dataset specific sequential layer was added.

2. Based on the L_2 normalization, from each of the activation maps, rank the corresponding convolutional filters or kernels.

3. Remove the non-contributing lower ranked filters.

4. Fine tune the model by retraining it with the CIFAR-10 dataset.

5. Check the trade-off between accuracy and the pruning objective and decide if further pruning is required.

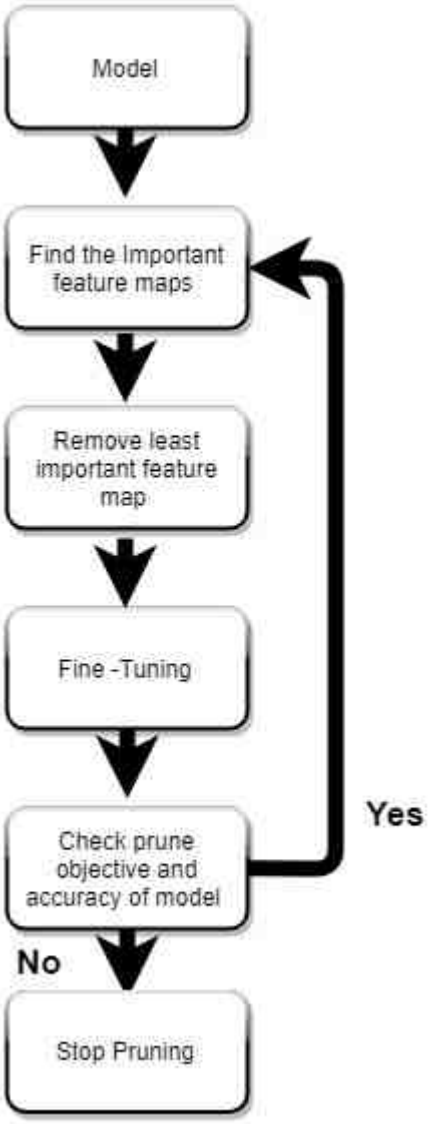

Fig. 7.2. Pruning steps to reduce the model based on L_2 normalization of activation map.

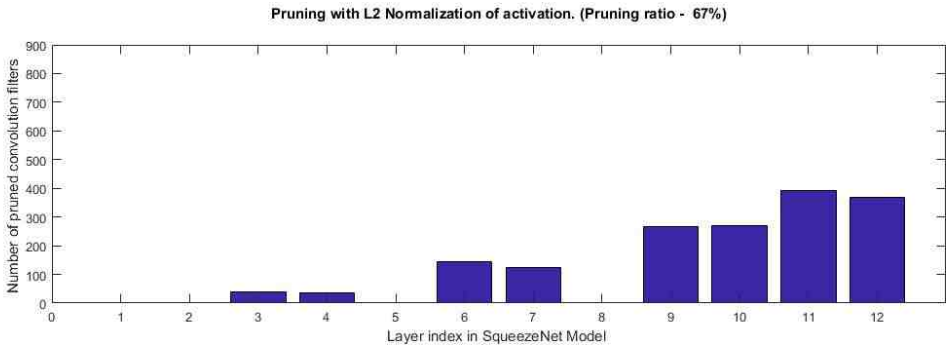

Fig. 7.3. Pruning with L_2 Normalization of activation with pruning ratio - 67%

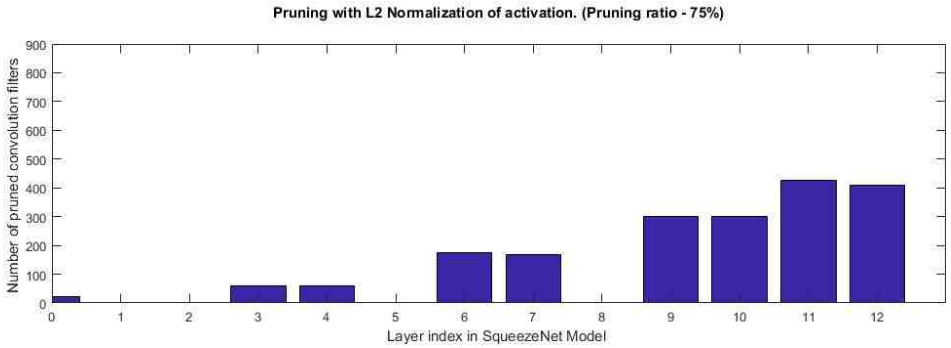

Fig. 7.4. Pruning with L_2 Normalization of activation with pruning ratio - 75%

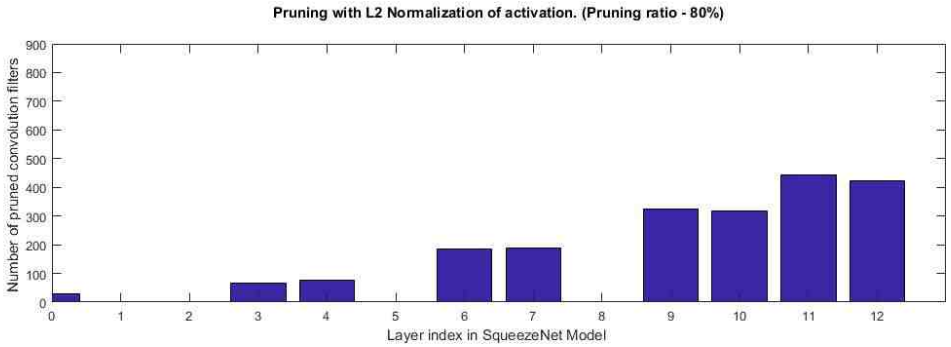

Fig. 7.5. Pruning with L_2 Normalization of activation with pruning ratio - 80%

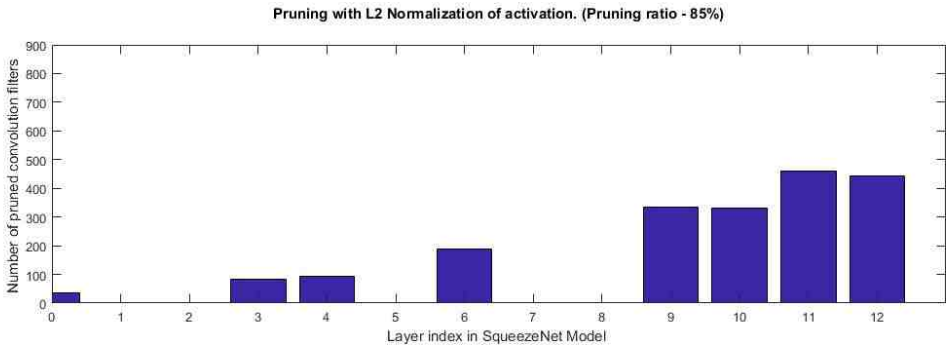

Fig. 7.6. Pruning with L_2 Normalization of activation with pruning ratio - 85%

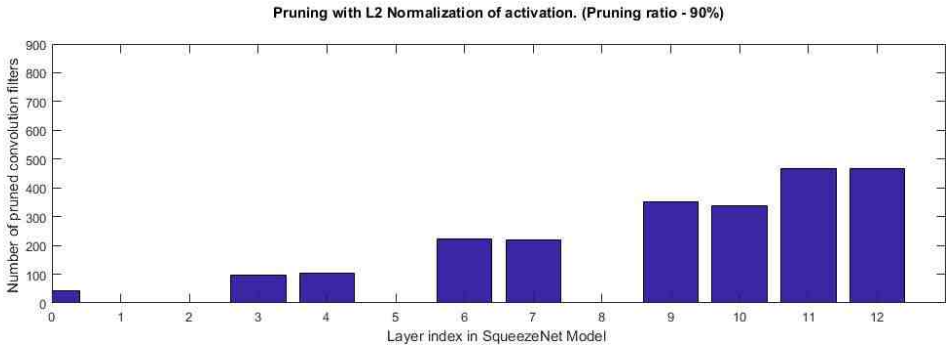

Fig. 7.7. Pruning with L_2 Normalization of activation with pruning ratio - 90%

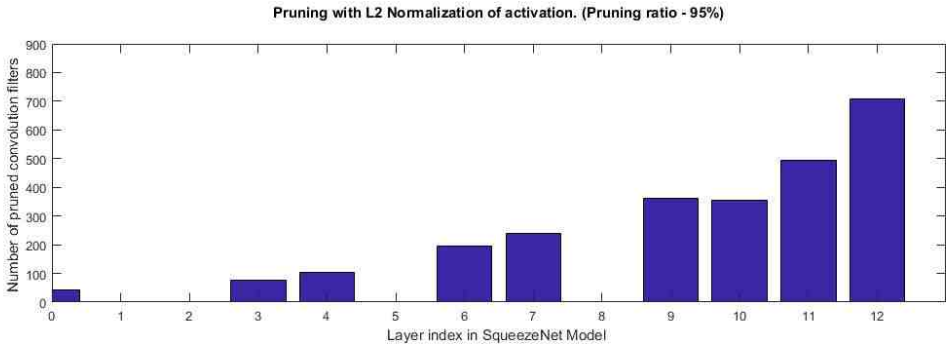

Fig. 7.8. Pruning with L_2 Normalization of activation with pruning ratio - 95%

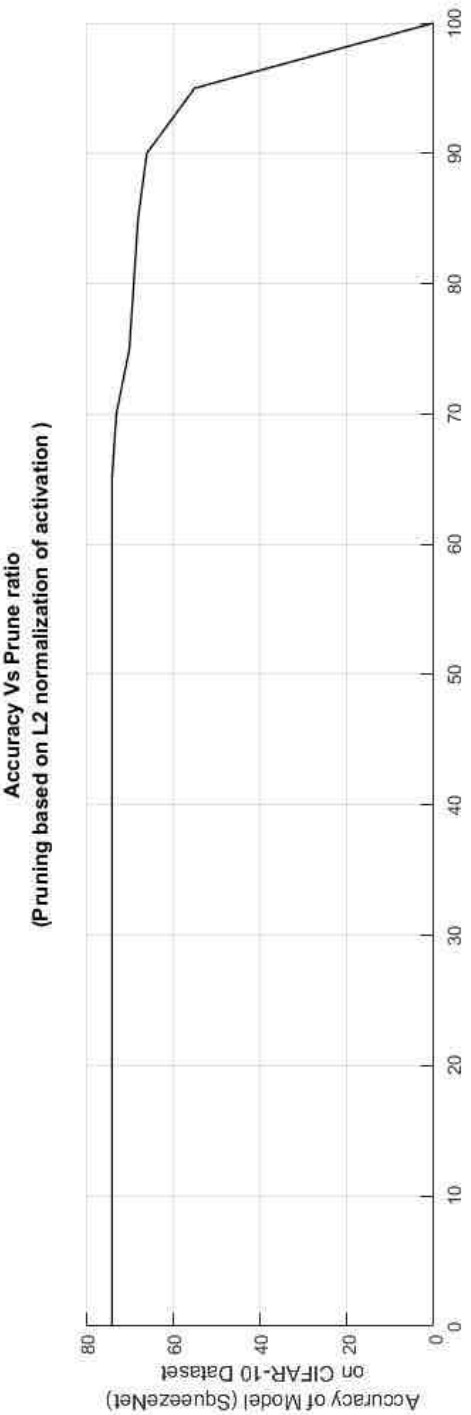

Fig. 7.9. Accuracy vs Pruning ratio for L_2 Normalization of activation map based pruning.

7.2 Results

SqueezeNet architecture was chosen for pruning because of its small modern and fully convolutional nature. It was trained on the CIFAR-10 dataset that resulted in the model size of 2.9 MB and with an accuracy of 74%.

The proposed pruning algorithm on SqueezeNet was implemented.
Pruning objectives:

1. Number of filters to prune per iteration = 128
2. Model size to prune (in %) = 67%

The different pruning ratio verses accuracy of the model is presented in Figure 7.9.

$$\begin{aligned}\text{Iterations} &= \frac{\frac{2}{3} \times \text{Total number of convolution filters}}{\text{Number of filters to prune per iteration}} \\ &= 11.92 \sim 12 \text{ iterations}\end{aligned}$$

Then L_2 norm values were calculated for each filter map. See Equation 7.1

Finally, iterative pruning took place based on the lower ranked filters. Figure 7.9 shows total number of pruned filters for each convolution layer of the SqueezeNet model with a different pruning ratio. Figure 7.9 and 5 shows that, there are many high-level features from the deeper layers of SqueezeNet, that are redundant or not critical for the accurate inference. Figure 7.9 suggests that pruning based on the proposed solution works well up to 70% of the pruning ratio without any significant drop in the accuracy of the model as compared to original model. If the network is pruned further, the accuracy of model decreases drastically. In conclusion, to get the optimal pruning efficiency, there should be a proper trade-off between network accuracy and pruning ratio. For the SqueezeNet, it was 67% pruning ratio with only a noted 1% of drop in the accuracy compared to original model.

8. PRUNING BASED ON TAYLOR EXPANSION OF COST FUNCTION

8.1 Implementation algorithm

In this chapter, the Taylor expansion of the cost function based pruning criteria is discussed, followed by the experiments with different pruning ratios and pruning results. Suppose there is a dataset D that consists of the training and testing data. The training data consist of training images and labels.

$$D \Rightarrow \begin{cases} X = x_0, x_1, ..., x_n \\ Y = y_0, y_1, ..., y_n \end{cases} \tag{8.1}$$

Where, X = input images, Y = labels, and W is the network parameters (Filter weights).

The pretrained model has initialized weights as W_0, $C(.)$ = Cost function of the network. Cost function can be any negative log-like functions (example: Cross-entropy function.). $C(D/W)$ = Cost function of the network on dataset D with network parameters W, W' = Network parameters that change due to pruning, as shown in Figure 8.1

The goal is to find the critical feature maps, that have a strong correlation with the output. In this chapter, the pruning problem was considered as a combinatorial optimization problem. The goal was to prune the filters, which minimizes the change in the cost function (ΔC).

$$\Delta C = \min |C(D/W) - C(D/W')| \tag{8.2}$$

To implement the above approach, It takes 2^{2292} computations to get the an optimal solution. Currently, there is not enough computation power to evaluate this

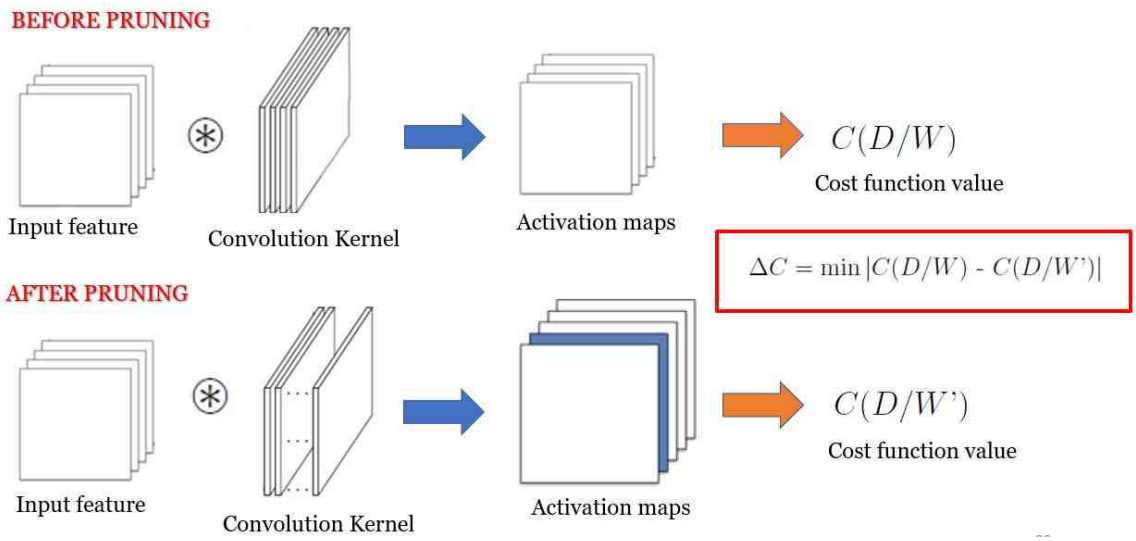

Fig. 8.1. Cost function value before pruning and after pruning.

kind of computation. The solution is to follow the greedy approach. The idea is to prune the feature maps iteratively by minimizing the drop in the accuracy to reach the pruning target. The cost function rely on the parameters and inference from these parameters. In this approach, the parameters are considered independent of each other. However in practical, these parameters are dependent on each other, as a result, the same assumption holds during each step of gradient calculation during the training process.

Let us consider the cost function value with pruning is $C(D, r_i = 0)$ and without pruning $C(D, r_i)$. Here r_i is the feature map produced by the i^{th} parameter. To evaluate $C(D, r_i = 0)$, first order Taylor expansion is used:
For a function $f(x)$, the Taylor expansion at point $x = k$ is

$$f(x) = \left| \sum_{m=1}^{M} \frac{f^m(k)}{m!}(x-k) + R_m(x) \right| \tag{8.3}$$

$$|\Delta C| = \min |C(D/W) \text{ - } C(D/W')| \tag{8.4}$$

$f^m(k)$ is the m^{th} derivative of $f(x)$ evaluated at point k

$R_m(x)$ is the m^{th} order remainder.

Approximating $C(D, r_i = 0)$ with a first-order Taylor polynomial near $r_i = 0$,

$$C(D, r_i = 0) = C(D, r_i) - \frac{\partial C}{\partial r_i} r_i + R_m(r_i = 0) \tag{8.5}$$

The remainder term can be calculated by the Lagrange formula:

$$R_m(r_i = 0) = \frac{\partial^2 C}{\partial (r_i^2 = \xi)} \frac{r_i^2}{2}$$

Where $\xi \in (0, r_i)$, Finally, by substituting Eq. (3) into Eq. (2) and ignoring the remainder to reduce the complexity,

$$|\Delta C(r_i)| = \left| C(D, r_i) - \frac{\partial C}{\partial r_i} r_i - C(D, r_i) \right| = \left| \frac{\partial C}{\partial r_i} r_i \right| \tag{8.6}$$

This approach calculates the ΔC for each feature map of each layer. The proposed algorithm ranks each filter based on ΔC value. Subsequently, each layer is normalized by L2 normalization using L2 norm of the ranks in that filter.

First, prune ratio per iteration is decided based on the pruning target. Both activation and gradient w.r.t activation are used to find out the change in the cost function of the model. The high value of ΔC indicates, that the feature map is correlates strongly with the output. After that, the lowest ranking filters or feature maps are pruned based on ranking. When the network is pruned, the accuracy of the network drops, then the network must be retrained to recover the accuracy. Finally, the trade-off between the pruning objective and drop in accuracy is checked for the model and then based on the results the further pruning is decided.

8.2 Results

The proposed pruning algorithm is experimented with different pruning ratios and thus analysis on the sensitivity of the pruning ratio to network accuracy is carried out.

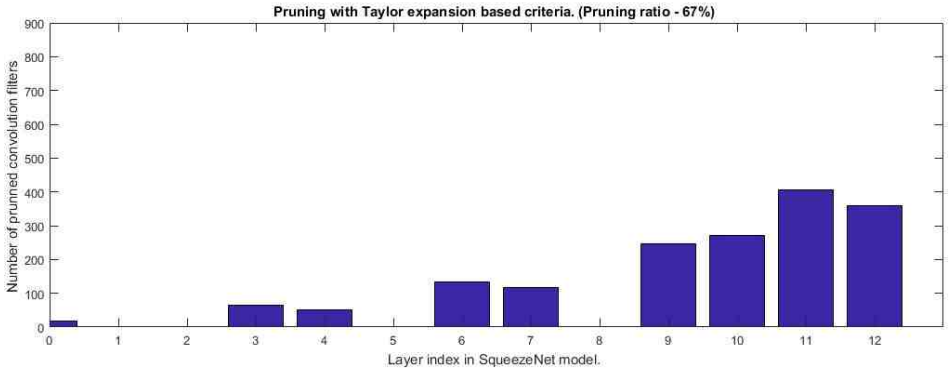

Fig. 8.2. Pruning with Taylor expansion based criteria with pruning ratio - 67%

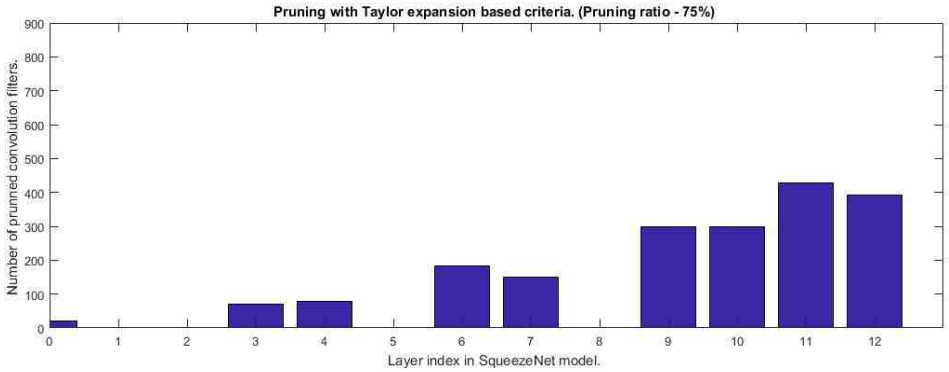

Fig. 8.3. Pruning with Taylor expansion based criteria with pruning ratio - 70%

Figure 8.8 shows the sensitivity of Taylor expansion based pruning criteria to accuracy of SqueezeNet model. Observation indicates that this pruning algorithm works well until 69% pruning ratio. If the network is pruned more than 69% then the

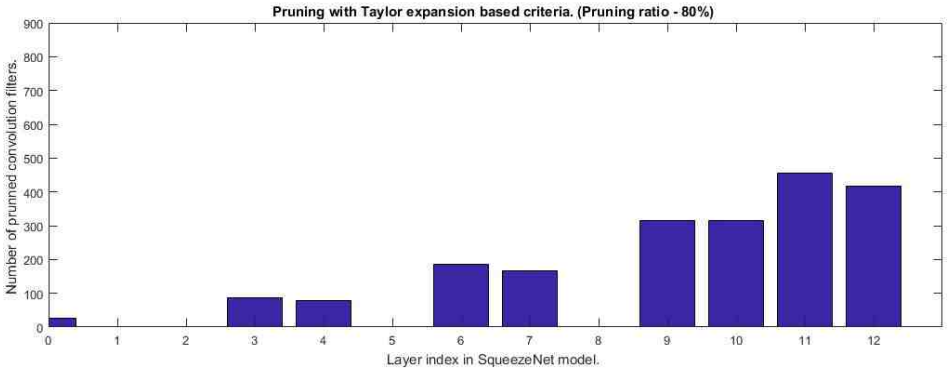

Fig. 8.4. Pruning with Taylor expansion based criteria with pruning ratio - 80%

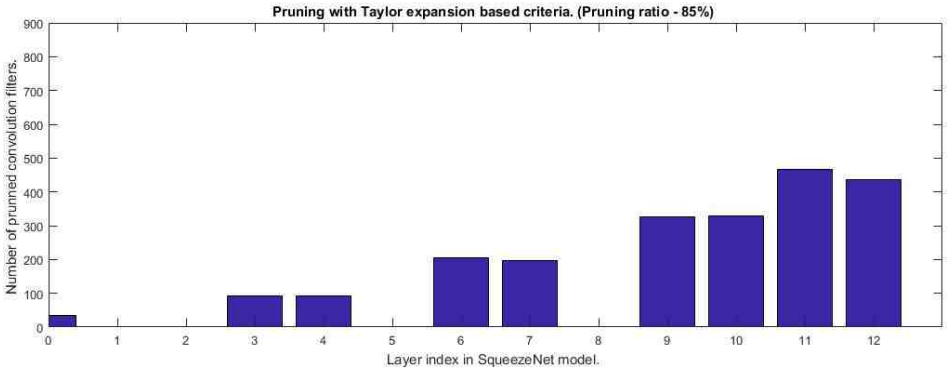

Fig. 8.5. Pruning with Taylor expansion based criteria with pruning ratio - 85%

accuracy of the model decreases drastically. The Proposed Taylor expansion-based criteria prunes the SqueezeNet model by the $\sim$ 69%. The pretrained SqueezeNet model was trained on the CIFAR-10 dataset and had an accuracy of 74% and model size was 2.95$-$3.0 MB. After that, the proposed algorithm was implemented iteratively to prune the network.

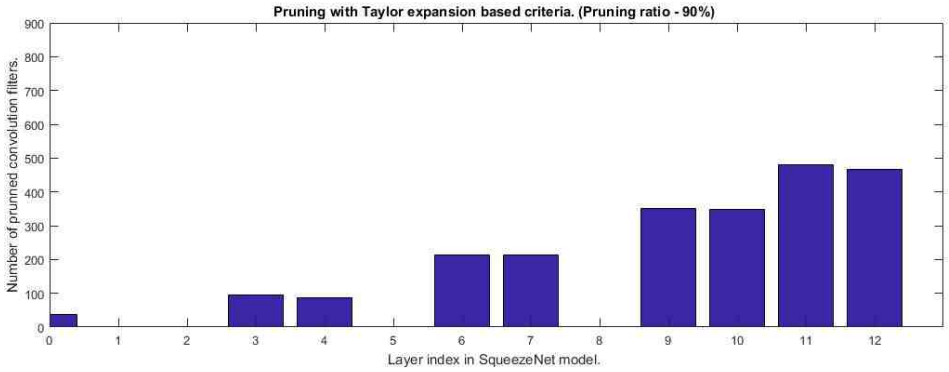

Fig. 8.6. Pruning with Taylor expansion based criteria with pruning ratio - 90%

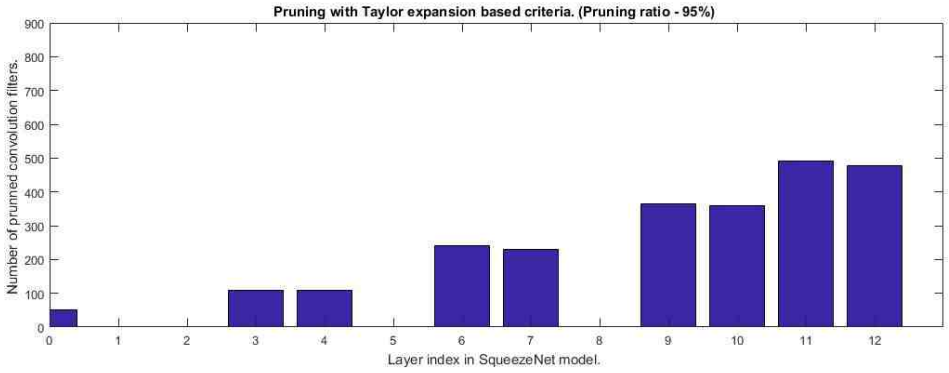

Fig. 8.7. Pruning with Taylor expansion based criteria with pruning ratio - 95%

Pruning objectives decided are as follows:

1. Number of filters to prune per iteration = 128
2. Model size to prune (Percentage wise) = 67%

$$\text{Number of iterations} = \frac{\frac{2}{3} \text{ x Total number of convolution filters}}{\text{Number of filters to prune per iteration}}$$
$$= 11.92 \sim 12 \text{ iterations}$$

Then, ΔC was calculated for each feature map based on the proposed Taylor expansion-based criteria.

$$|\Delta C(r_i)| = \left| \frac{\partial C}{\partial r_i} r_i \right|$$

Consider, layers from the index (0-2) as initial layers, (3-7) as middle layers, and (9-12) as a last layers for SqueezeNet architecture (See table 8.2). It is observed that 77.47 % filters were pruned from the last layers, 21.7% from the middle layer, and 0.7% from initial layers. Most of the pruned feature maps were found to be from the last layers. Observation suggests that many of high level features do not contribute towards the inference. With this approach, The SqueezeNet model size is reduced from 2.9 MB to 984 KB which is the total reduction of 69% in the model size and with an accuracy drop of only $\sim 1\%$, i.e. 73% for the CIFAR-10 dataset (Refer Table: 8.2).

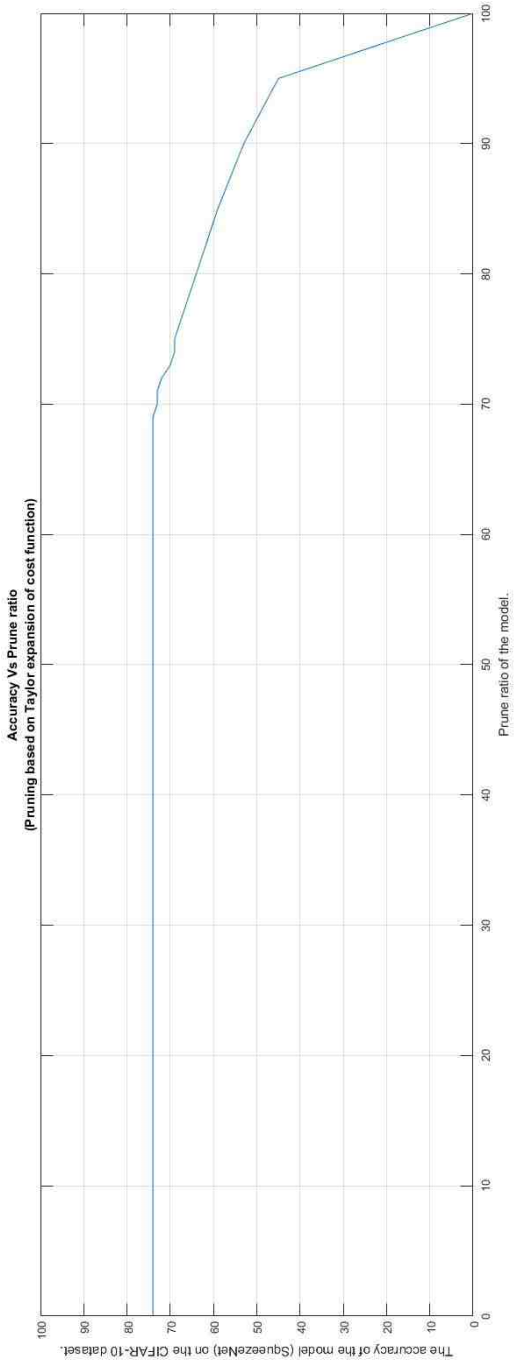

Fig. 8.8. Accuracy Vs pruning ratio for Taylor expansion based criteria

Table 8.1.
Pruning Pattern per iteration of SqueezeNet Model using Taylor expansion-based criterion with pruning ratio = 67%.

Type of Layers	Convolution	ReLU	Pooling	Fire	Fire	Pooling	Fire	Fire	Pooling	Fire	Fire	Fire	Fire	Total
Layer Index	0	1	2	3	4	5	6	7	8	9	10	11	12	
Iteration 1	0	0	0	1	1	0	2	0	0	28	19	27	50	128
Iteration 2	0	0	0	1	0	0	4	1	0	11	18	33	60	128
Iteration 3	0	0	0	1	0	0	8	5	0	15	18	33	48	128
Iteration 4	1	0	0	2	3	0	12	12	0	18	19	37	24	128
Iteration 5	0	0	0	3	1	0	9	10	0	13	28	34	30	128
Iteration 6	0	0	0	4	4	0	15	18	0	25	17	37	8	128
Iteration 7	1	0	0	7	5	0	17	10	0	20	26	29	13	128
Iteration 8	2	0	0	8	1	0	10	4	0	22	18	38	25	128
Iteration 9	1	0	0	5	3	0	10	13	0	27	22	26	21	128
Iteration 10	2	0	0	9	4	0	14	13	0	20	19	23	24	128
Iteration 11	3	0	0	4	8	0	14	13	0	8	27	39	12	128
Iteration 12	1	0	0	7	3	0	19	17	0	19	16	21	25	128
Total Filters pruned	11	0	0	52	33	0	134	116	0	226	247	377	340	1536

Table 8.2.
Sensitivity to pruning the SqueezeNet model on CIFAR-10 dataset.

Pruning Ratio (%wise)	Accuracy of model
0	74
5	74
10	74
15	74
20	74
25	74
30	74
35	74
40	74
45	74
50	74
55	74
60	74
65	74
67	74
70	73
75	69
80	65
85	59
90	53
95	45
100	0

9. PRUNING BASED ON A COMBINATION OF TAYLOR EXPANSION OF COST FUNCTION & L_2 NORMALIZATION OF ACTIVATION MAPS

9.1 Implementation algorithm

In this chapter, the combination of the two pruning criteria is discussed: Pruning based on Taylor expansion of cost function and L_2 normalization of activation maps. Following that experiments with different pruning ratio is discussed, and lastly Pruning results are discussed.

The experimentation is performed with two pruning criteria by combining ranking methods of Taylor expansion of cost function and L_2 normalization of activation maps. This experiment is carried out to find out the optimal pruning efficiency of the modified SqueezeNet architecture on the CIFAR-10 dataset. The goal of this proposed algorithm is to find out the uncorrelated convolution filters towards the inference by ranking methods, and the pruning of the low ranked filters or kernels form the architecture is performed. Pruning the kernels will introduce the sparsity in the network. To combat this problem, the network architecture is restructured, and then to reach the pruning target, the pruning algorithm is performed iteratively. The Figure 9.1 shows the pruning steps to implement the proposed algorithms.

1. Import the model (SqueezeNet):

2. Find the critical convolution kernels:

   To find these essential filters towards the output, the first step is to calculate the individual ranking for every convolution filter in the model and finally combine the two ranking methods:

(a) Ranking based on Taylor expansion of cost function:

This ranking method calculate the ΔC for each feature maps using the following formula.

$$|\Delta C(r_i)| = \left| C(D, r_i) - \frac{\partial C}{\partial r_i} r_i - C(D, r_i) \right| = \left| \frac{\partial C}{\partial r_i} r_i \right|$$

This algorithm ranks each convolution filter based on δC value, following that each layer is normalized by L_2 normalization.

(b) Ranking based on L_2 normalization of activation maps.:

This ranking methods calculate L_2 normalization value for each activation map and decide the ranking of the corresponding convolution filter based on the following formula.

$$L_2 \text{ norm is} \rightarrow ||A_i||_2 = \sqrt{\sum_{i=1}^{n} A_i^2}$$

After that two individual ranking are combined to get the final ranking.

3. Lower ranked filters are calculated using the min heap algorithm and finally, the model is pruned iteratively.

4. Fine tune the model by retraining the model with a CIFAR-10 dataset.

5. Check the trade-off between accuracy and the pruning objective and decide if further pruning is required.

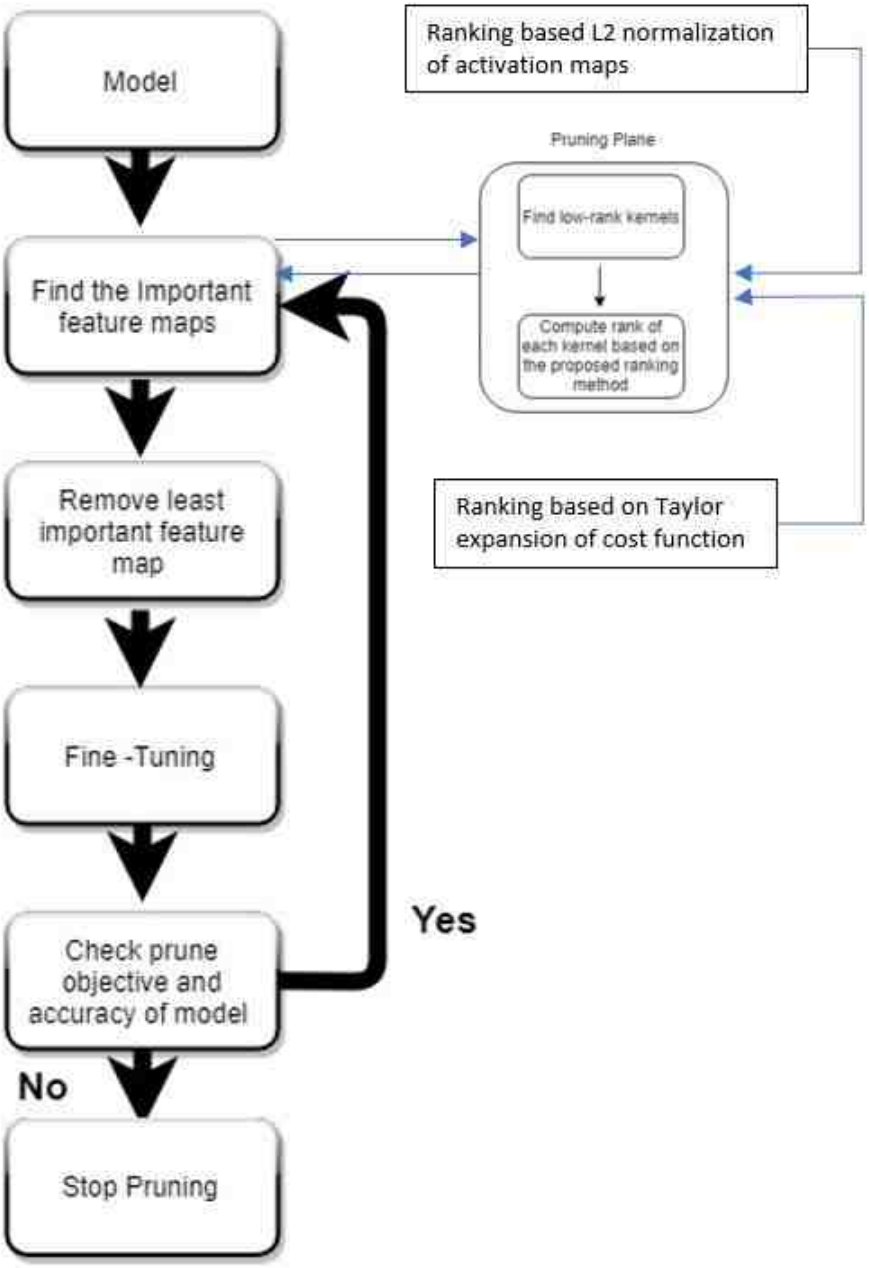

Fig. 9.1. Pruning steps to reduce the model based on combination of Taylor expansion of cost function and L_2 normalization of activation maps.

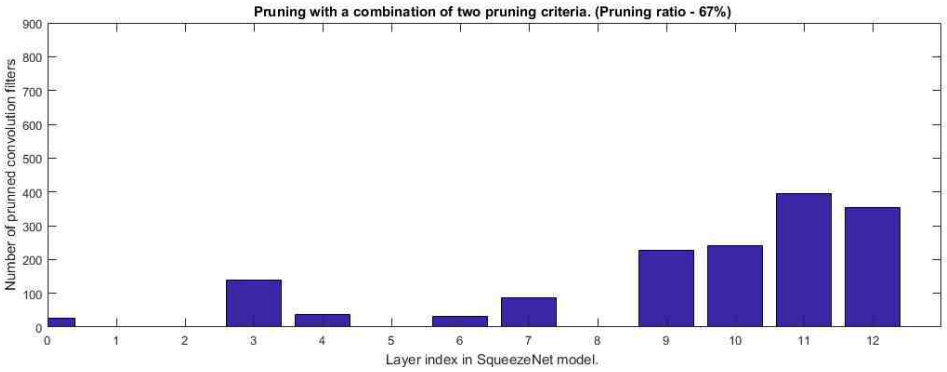

Fig. 9.2. Pruning based on a combination of Taylor expansion of cost function and L_2 normalization of activation map with pruning ratio - 67%

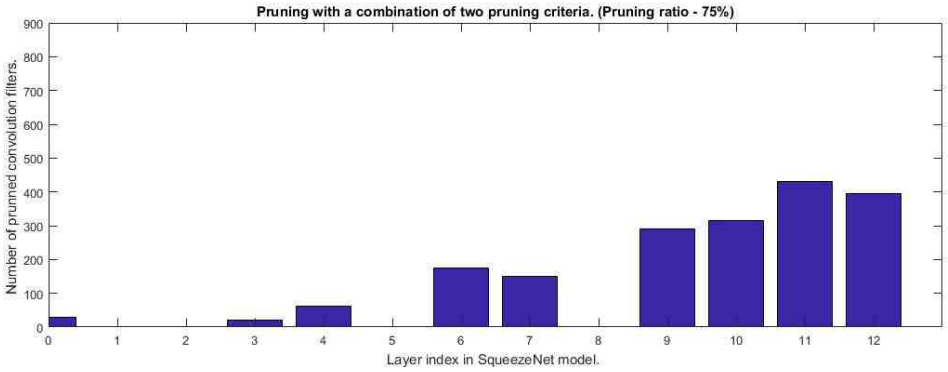

Fig. 9.3. Pruning based on a combination of Taylor expansion of cost function and L_2 normalization of activation map with pruning ratio - 75%

9.2 Results

SqueezeNet architecture is chosen for implementing this pruning algorithm because of its fully convolutional and modern architecture inspired by inception module. It is trained on CIFAR-10 dataset using the transfer learning technique with the model size around 3 MB and accuracy of 74%. The proposed pruning algorithm on

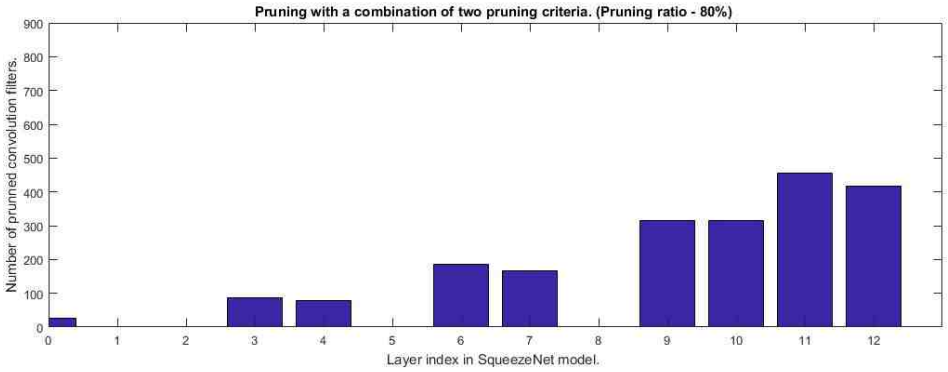

Fig. 9.4. Pruning based on a combination of Taylor expansion of cost function and L_2 normalization of activation map with pruning ratio - 80%

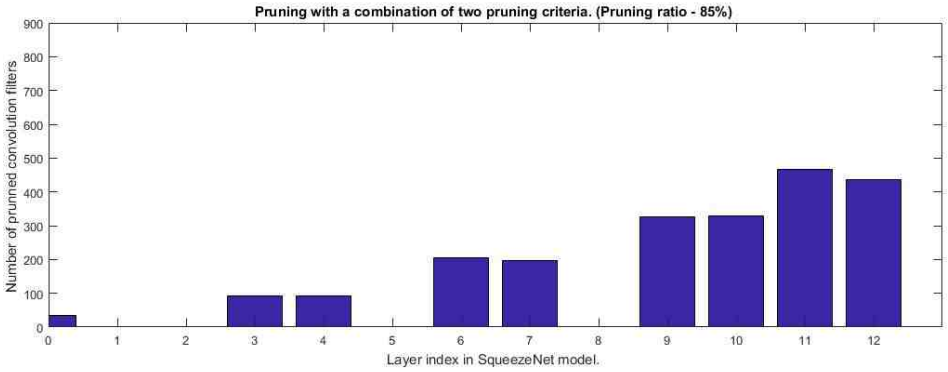

Fig. 9.5. Pruning based on a combination of Taylor expansion of cost function and L_2 normalization of activation map with pruning ratio - 85%

SqueezeNet is implemented.

The pruning objectives decided are as follows:

1. Number of filters to prune per iteration = 128

2. Model size to prune (in Percentage) = 67%

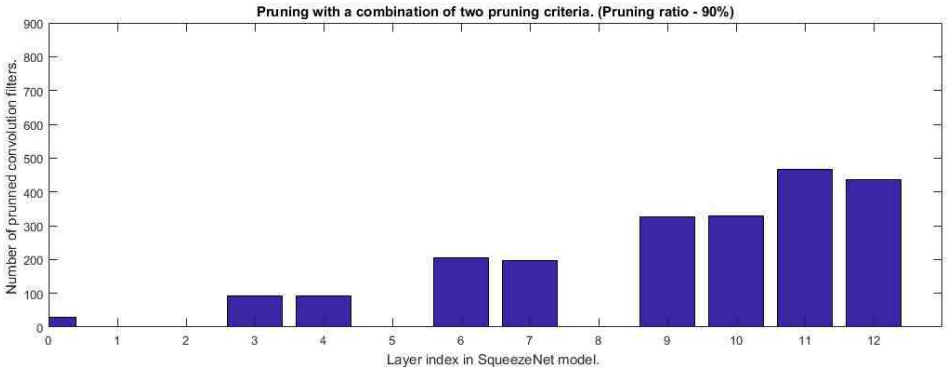

Fig. 9.6. Pruning based on a combination of Taylor expansion of cost function and L_2 normalization of activation map with pruning ratio - 90%

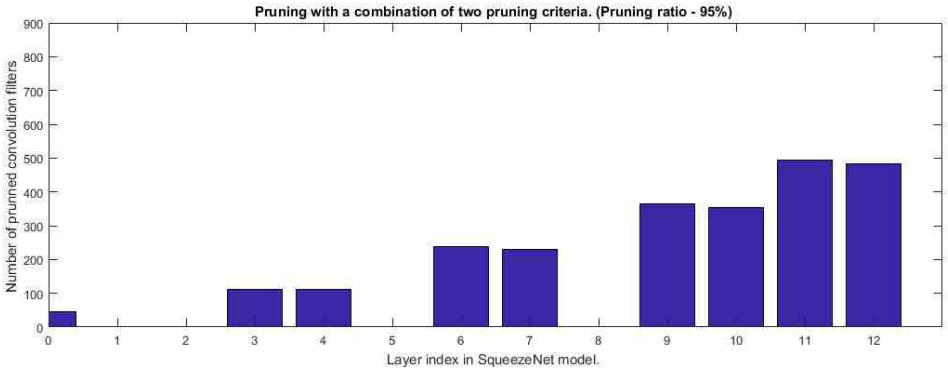

Fig. 9.7. Pruning based on a combination of Taylor expansion of cost function and L_2 normalization of activation map with pruning ratio - 95%

The number of iterations required to achieve the pruning target is calculated. The different pruning ratio verses accuracy of the model is presented in Figure 9.9.

$$\text{Iterations} = \frac{\frac{2}{3} \times \text{Total number of convolution filters}}{\text{Number of filters to prune per iteration}}$$
$$= 11.92 \sim 12 \text{ iterations}$$

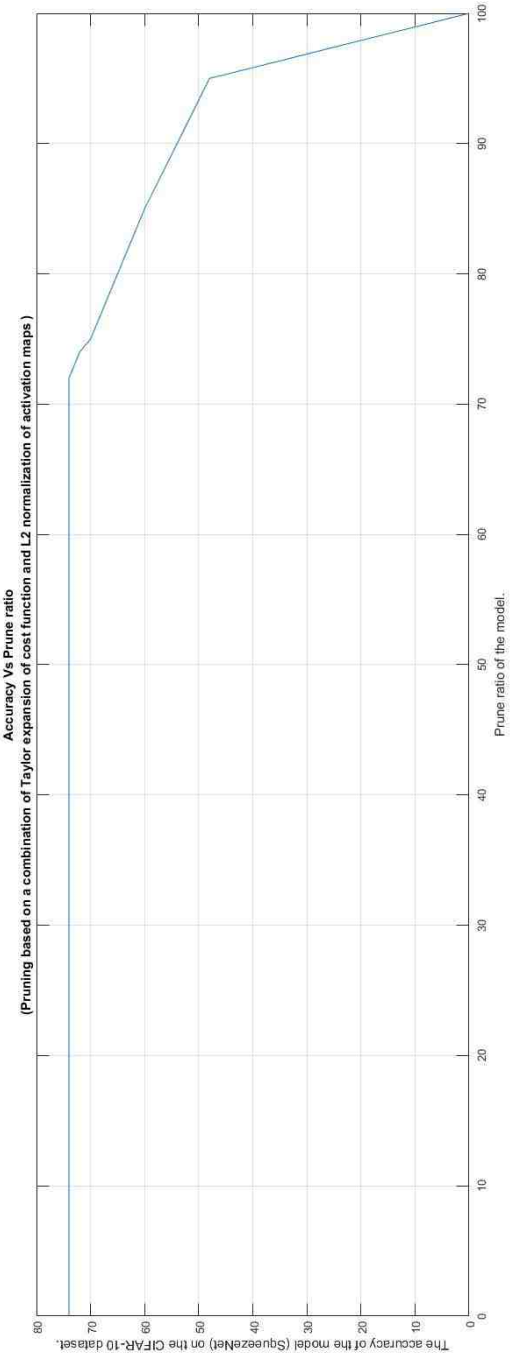

Fig. 9.8. Accuracy vs Pruning ratio for combination of Taylor expansion of cost function and L_2 normalization of activation map with pruning ratio .

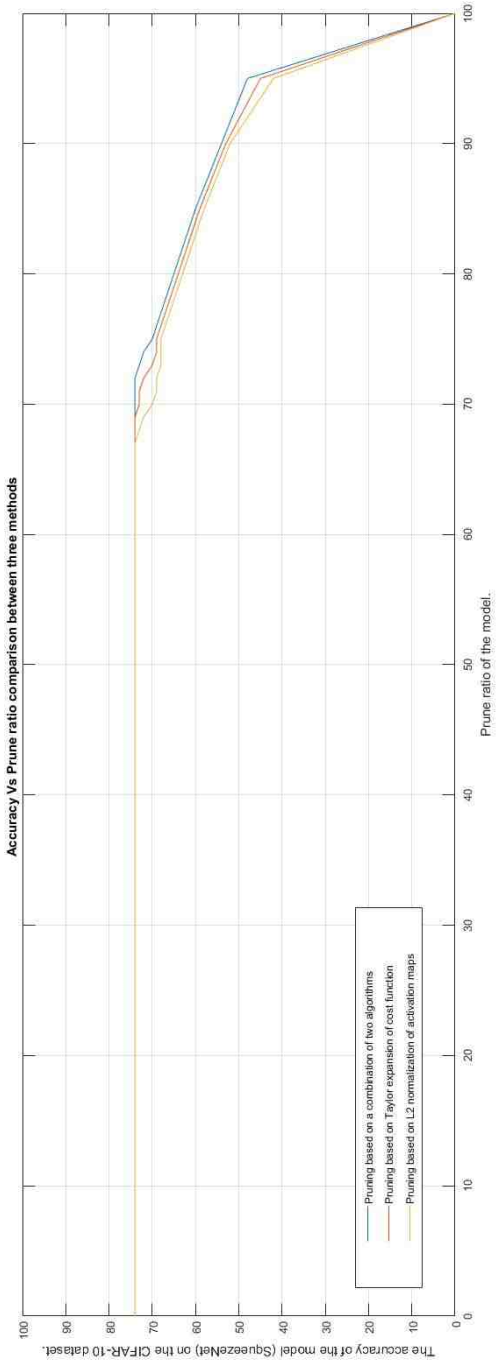

Fig. 9.9. Accuracy vs prune ratio comparison between three pruning methods.

Finally, iterative pruning takes place based on the lower ranked filters. Figure 9.2 to 9.7 shows the total number of pruned filters for each convolution layers of the SqueezeNet model with different pruning ratios. Figure 9.2 to 9.7 and 5.5 shows that, there are many mid and high-level features from the deeper layers of SqueezeNet, that are redundant or not critical for the accurate inference, so this pruning algorithm concentrates in these mid and high-level filters for pruning. Figure 9.8 suggests that pruning based on the proposed solution works well up to 72% of the pruning ratio without any significant drop in the accuracy of the model as compared to the original model. If the network is pruned further, the accuracy of the model decreases drastically. In conclusion, to achieve the optimal pruning efficiency, there should be a proper trade-off between network accuracy and pruning ratio. For the SqueezeNet, it is 72% pruning ratio with only a 1% of the drop in the accuracy compared to the original model accuracy as shown in the Figure 9.9. The three pruning algorithms is compared and the results shows that, the combination of Taylor expansion of cost function and L_2 normalization of the activation maps based pruning performed is better than other two individual pruning criteria, as shown in Figure 9.9.

10. HARDWARE DEPLOYMENT OF PRUNED SQUEEZENET MODEL ON BLUEBOX USING RTMAPS

One of the main goals of thesis is to deploy the pruned model on specialized hardware such as BlueBox [28]. In the previous chapters, The model compression was discussed with pruning. This chapter focuses on hardware deployment of the pruned model. The modified SqueezeNet is trained and pruned on an i7 processor, 32 GB RAM and GTX 1080 by Nvidia using the PyTorch Framework and the final pruned model was generated.

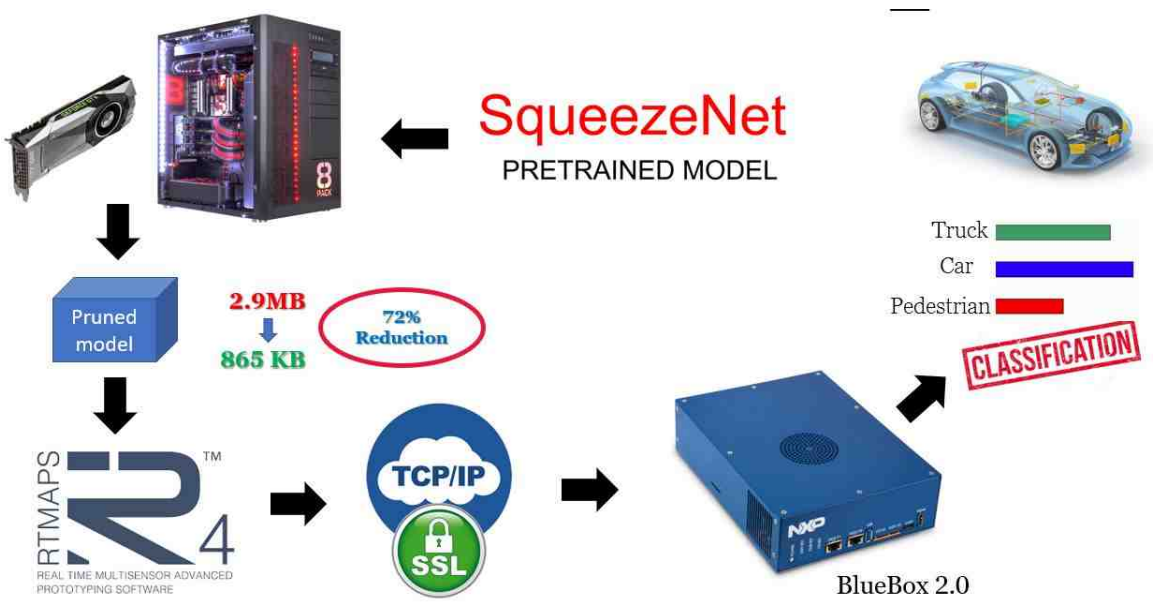

Fig. 10.1. Hardware deployment overview with BlueBox 2.0 and RTMaps.

RTMaps provides supports for deploying deep learning architectures. PyTorch framework can be installed on RTMaps desktop studio and RTMaps runtime studio. 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 framework. 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. The deployment overview is shown in Figure 10.1

10.1 NXP BlueBox

The major challenge in porting an algorithm on the vision system is to optimally map it to various units of the system in order to achieve an overall boost in the performance. This requires detailed 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 well. One of the significant requirements of this thesis is to analyze the capability of NXP Bluebox 2.0 (BLBX2) [29] as an autonomous embedded system for the real-time applications. Bluebox is one of the development platforms designed for the advanced driver assistance system feature for the autonomous vehicles. The bluebox development platform is an integrated package for automated driving and is comprised of three independent systems on chip (SoCs: S32V234, LS2084A, S32R274).

The BLBX2 operates on the independent embedded Linux OS for both the S32V and LS2 processors, the S32R typically runs bare-metal code or an RTOS. BlueBox as shown in the Figure 10.2 functions as the central computing unit (as a brain) of the system thus providing the capability to control the car through actions based on the inputs collected from the surrounding. This section details the information related to the components incorporated within the bluebox.

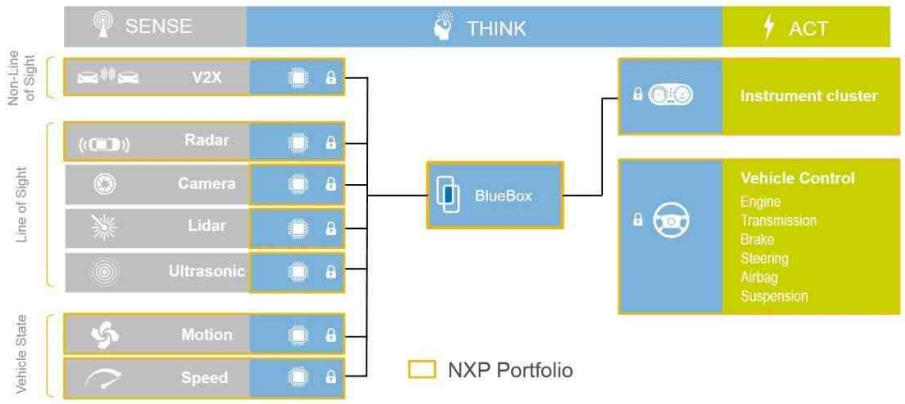

Fig. 10.2. Hardware Architecture for Bluebox 2.0 [Pic Courtesy NXP].

10.1.1 S32V234

The S32V234 [5] is a vision-based processor designed for computationally intensive application related to vision and image processing. The processor comprises of Image Signal Processor (ISP) available on all MIPI-CSI camera inputs, providing the 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 be operated on 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 10.3 and 10.4.

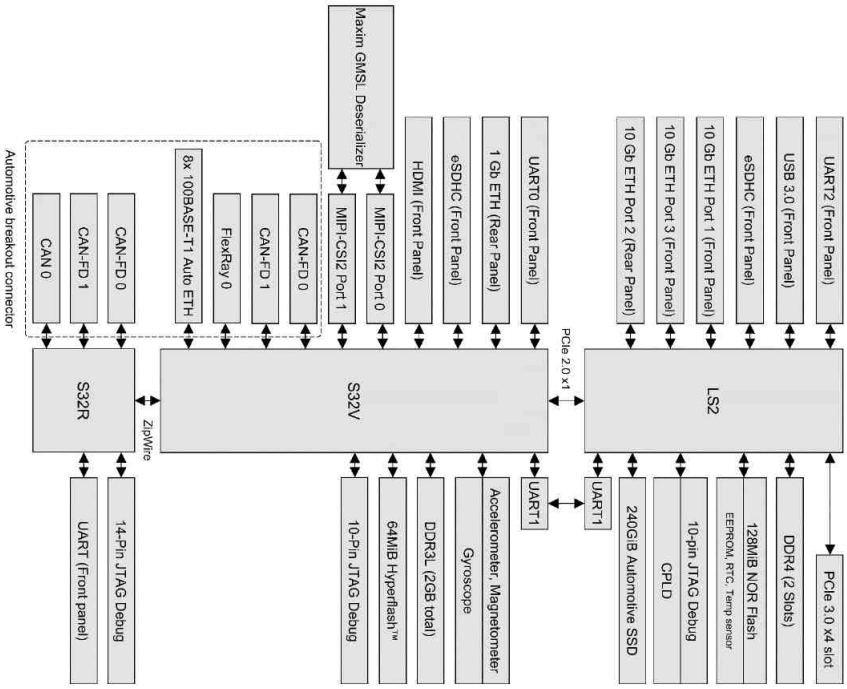

Fig. 10.3. Hardware Architecture for Bluebox 2.0 [Pic Courtesy NXP].

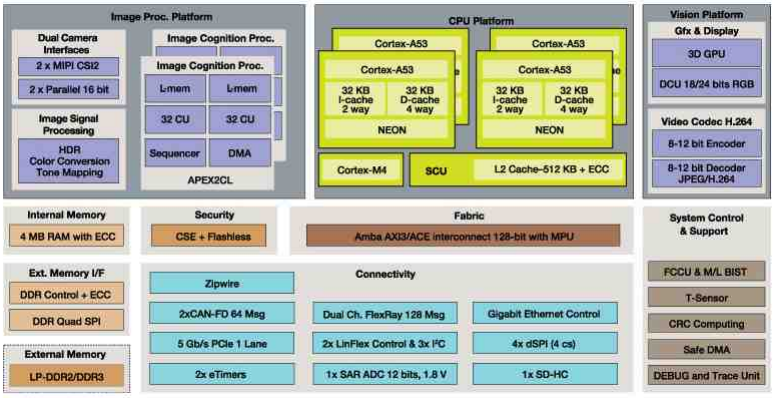

Fig. 10.4. Hardware Architecture for S32V234 [Pic Courtesy NXP].

10.1.2 LS2084A

The LS2 processor in the BLBX2 is designed as 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 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.

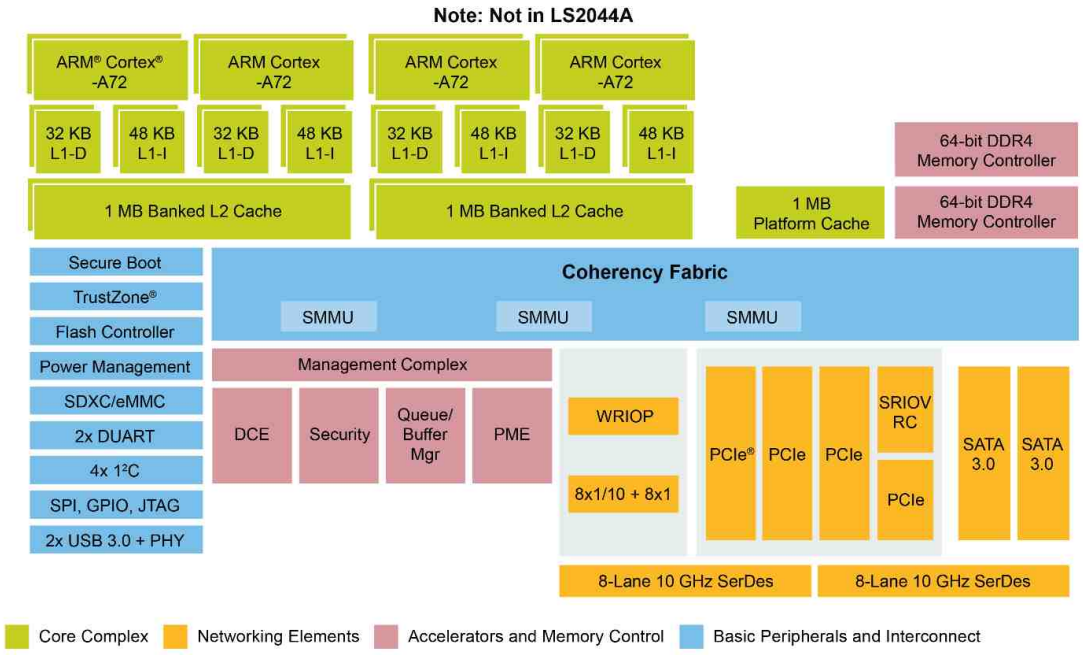

Fig. 10.5. Hardware Architecture for LS2084A [Pic Courtesy NXP].

10.2 Real-Time Multisensor applications (RTMaps)

RTMaps is designed for the development of multimodal based applications [30], thus providing the feature of incorporating multiple sensors such as camera, lidar,

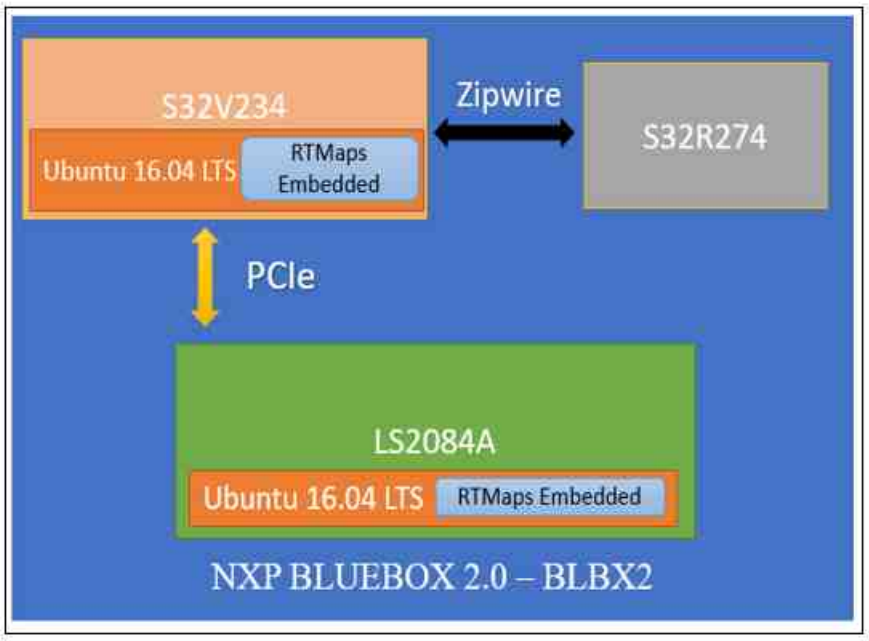

Fig. 10.6. System overview of NXP BlueBox 2.0 BLBX2.

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

10.2.1 RTMaps Runtime Engine

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

10.2.2 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.

10.2.3 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 Linux based platforms. Applications are developed by using the modules and packages available from the RTMaps Component library.

10.2.4 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 thesis 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 purposes. 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 10.7.

10.3 Hardware Implementation

Figure 10.7 shows the process of hardware deployment of the pruned model using RTMaps and BlueBox 2.0. Training and pruning take place on the desktop PC on PyTorch framework and after the successful implementation of pruning algorithms, a final pruned model is generated. RTMaps desktop studio is used to load the model parameters. TCP/IP connection is established between RTMaps desktop studio and RTMaps runtime studio using RTMaps runtime engine. Finally, the performance of the model is evaluated using the BlueBox 2.0.

To evaluate the performance of the pruned model with an original model, two experiments were conducted. First, The original model (Model size - 2.9 MB) was deployed on PC to get the inference for single image and inference time was calculated (inference time - 2 ms) then the same model was deployed on BlueBox 2.0 with RTMaps software and inference time was calculated (Inference time - 1.8 ms). Finally, the same steps were repeated for the pruned model and results were calculated, for PC the inference time was 1 ms and for BlueBox 2.0 it was 0.9 ms with the model size 846KB, as shown in Figure 10.8

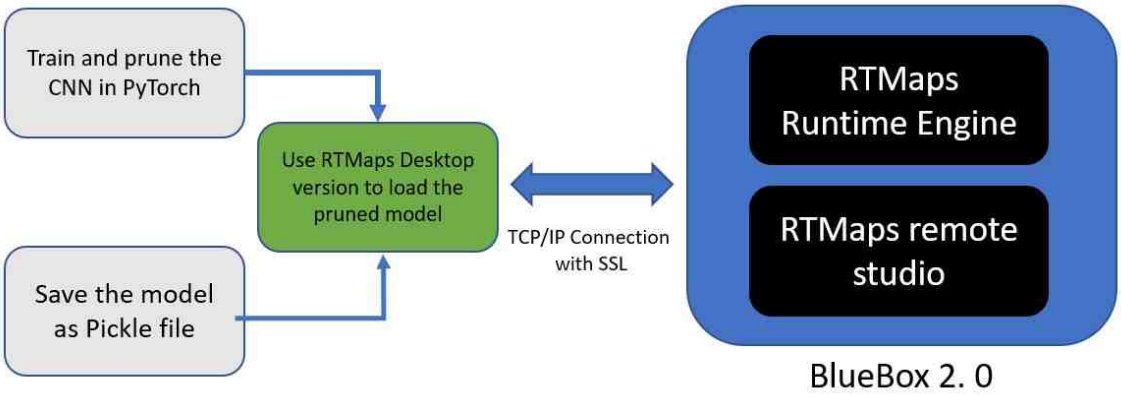

Fig. 10.7. Hardware deployment of the pruned model using the RTMaps and BlueBox 2.0

Prune Ratio	Model size	Model Accuracy	Inference time (PC)	Inference time (BlueBox)
Original model	2.9 MB	74%	2ms	1.8ms
Pruned Model	846 KB	74%	1ms	0.9ms

OPTIMAL PRUNING EFFICIENCY

Fig. 10.8. Model performances with PC and BlueBox 2.0.

11. SUMMARY

Modern CNN architecture has a large number of parameters and considerable training time and inference cost. This thesis proposes a algorithms to prune the Convolution Neural Network (SqueezeNet) based three criteria:

1. Pruning based on Taylor expansion of the cost function.
2. Pruning based on L_2 normalization of activation maps.
3. Pruning based on a combination of Taylor expansion of the cost function and L_2 normalization of activation maps.

The proposed algorithms achieves 72% optimal pruning efficiency on SqueezeNet architecture without a significant loss in accuracy as compared to the original model on the CIFAR-10 dataset. These pruning algorithms prune the model without introducing the network sparsity. Using the proposed pruning technique, the sensitive or robust filters in the model can be identified, this will help to understand and improve the modern architecture better. Results show that most of the pruned kernels are from mid and high-level layers. Results also show that Pruning based on a combination of Taylor expansion of the cost function and L_2 normalization of activation maps archives better pruning efficiency compared to other individual pruning criteria. Finally, the Pruned model is deployed on BlueBox 2.0 using RTMap software and model performance was calculated. the model size was reduced from 2.9 MB to 846 KB which is 72% reduction and the inference time was reduced by 50% for an embedded target (BlueBox 2.0).

REFERENCES

REFERENCES

[1] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016 (Accessed June 18, 2018). [Online]. Available: https://arxiv.org/abs/1602.07360

[2] P. Ballester and R. M. de Araújo, "On the performance of googlenet and alexnet applied to sketches." in *AAAI*, 2016, pp. 1124–1128.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014 (Accessed July 20, 2018). [Online]. Available: https://arxiv.org/abs/1409.1556

[4] X. Li, G. Zhang, H. H. Huang, Z. Wang, and W. Zheng, "Performance analysis of gpu-based convolutional neural networks," in *2016, 45th International Conference on Parallel Processing (ICPP)*. IEEE, 2016, pp. 67–76.

[5] NXP, "Nxp-s32v234," 2018 (Accessed August 20, 2018). [Online]. Available: https://www.nxp.com/products///processors-and-microcontrollers/arm-based-processors-and-mcus/s32-automotive-platform:S32

[6] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *arXiv preprint arXiv:1611.05128*, 2016 (Accessed June 1, 2018). [Online]. Available: https://arxiv.org/abs/1611.05128

[7] J. Gildenblat, "Pruning deep neural networks to make them fast and small," 2009 (Accessed August 18, 2018). [Online]. Available: https://jacobgil.github.io/deeplearning/pruning-deep-learning

[8] B. O. Ayinde and J. M. Zurada, "Building efficient convnets using redundant feature pruning," *arXiv preprint arXiv:1802.07653*, 2018 (Accessed September 21, 2018). [Online]. Available: https://arxiv.org/abs/1802.07653

[9] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *International Conference on Computer Vision (ICCV)*, vol. 2, no. 6, 2017.

[10] J.-H. Luo and J. Wu, "An entropy-based pruning method for cnn compression," *arXiv preprint arXiv:1706.05791*, 2017 (Accessed June 19, 2018). [Online]. Available: https://arxiv.org/abs/1706.05791

[11] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR, abs/1611.06440*, 2016 (Accessed June 20, 2018). [Online]. Available: https://arxiv.org/abs/1611.06440

[12] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990 (Accessed June 10, 2018), pp. 598–605.

[13] Z. Mariet and S. Sra, "Diversity networks: Neural network compression using determinantal point processes," *arXiv preprint arXiv:1511.05077*, 2015 (Accessed July 9, 2018). [Online]. Available: https://arxiv.org/abs/1511.05077

[14] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *arXiv preprint arXiv:1312.5851*, 2013 (Accessed July 28, 2018). [Online]. Available: https://arxiv.org/abs/1312.5851

[15] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.

[16] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," *IEEE Access*, vol. 3, pp. 2163–2175, 2015.

[17] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[18] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016 (Accessed July 21, 2018). [Online]. Available: https://arxiv.org/abs/1608.08710

[19] P. Durvesh, "Squeezed cnn: A real-time classifier with rtmaps and bluebox 2.0," 2018 (Accessed August 30, 2018). [Online]. Available: Unpublished paper, Submitted to CCWC 2019: IEEE Annual Computing and Communication Workshop and Conference

[20] S. Hijazi, R. Kumar, and C. Rowen, "Using convolutional neural networks for image recognition," *Cadence Design Systems Inc.: San Jose, CA, USA*, 2015.

[21] CS231n, "Convolutional neural networks for visual recognition," 2018 (Accessed September 1, 2018). [Online]. Available: http://cs231n.github.io/convolutional-networks/

[22] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," p. 20, 2015 (Accessed August 26, 2018). [Online]. Available: http://yann. lecun. com/exdb/lenet

[23] PyTroch, "Pytroch," 2018 (Accessed August 21, 2018). [Online]. Available: https://pytorch.org/

[24] Keras, "Keras," 2018 (Accessed August 21, 2018). [Online]. Available: https://keras.io/

[25] Caffe, "Caffe," 2018 (Accessed August 21, 2018). [Online]. Available: http://caffe.berkeleyvision.org/

[26] Theano, "Theano," 2018 (Accessed August 21, 2018). [Online]. Available: http://deeplearning.net/software/theano/

[27] TensorFlow, "Tensorflow," 2018 (Accessed August 21, 2018). [Online]. Available: https://www.tensorflow.org/

[28] S. Venkitachalam and A. Gaikwad, "Realtime applications with rtmaps and bluebox 2.0," 2018 (Accessed August 28, 2018). [Online]. Available: https://csce.ucmss.com/cr/books/2018/LFS/CSREA2018/ICA4218.pdf

[29] Nxp.com.(2018), "Nxp bluebox autonomous driving—nxp," 2018 (Accessed June 2, 2018). [Online]. Available: https://www.nxp.com/products/processors-andmicrocontrollers/arm-based-processors-and-mcus/qoriq-layerscape-armprocessors/nxp-bluebox-autonomous-driving-developmentplatform:BLBX

[30] Intempora, "Intempora - rtmaps - a component-based frame-work for rapid development of multi-modal applications," 2018 (Accessed September 21, 2018). [Online]. Available: https://intempora.com/