Sub-clustering based Neurogenetic Pruning

A thesis submitted

in partial fulfillment for the award of the degree of

Master of Technology

in

Digital Signal Processing

by

Soumya Sara John

Department of Avionics

Indian Institute of Space Science and Technology

Thiruvananthapuram, India

May, 2019

Certificate

This is to certify that the thesis titled '**Sub-clustering based Neurogenetic Pruning**' submitted by **Soumya Sara John**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the degree of **Master of Technology** in **Digital Signal Processing**, is a bonafide record of the research work done by her under our supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Deepak Mishra
Supervisor-1
Associate Professor
Dept. of Avionics
IIST

Dr. Sheeba Rani J.
Supervisor-2
Associate Professor
Dept. of Avionics
IIST

Dr. B. S. Manoj
Head of the Department
Dept. of Avionics
IIST

Place: Thiruvananthapuram
Date: May, 2019

Declaration

I declare that this thesis titled '**Sub-clustering based Neurogenetic Pruning**' submitted in partial fulfillment for the award of the degree of **Master of Technology** in **Digital Signal Processing** is a record of original work carried out by me under the supervision of **Dr. Deepak Mishra** and **Dr. Sheeba Rani J.**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited.

Place: Thiruvananthapuram
Date: May, 2019

Soumya Sara John
(SC17M048)

To my wonderful parents.

Acknowledgements

I express my sincere gratitude to my Supervisors Dr. Deepak Mishra, Associate Professor, IIST and Dr. Sheeba Rani J., Associate professor, IIST for their guidance and support. They consistently allowed this thesis to be my own work and steered me in the right direction whenever I needed it. I am grateful for their assistance throughout the project.

I would like to thank Dr. Lakshmi Narayanan R., Assistant Professor, IIST for his encouragement through words and inspirational books, which helped me through hard times. I would also like to thank Dr. Vineeth B.S., Assistant Professor, IIST, for his timely support, that helped me in completing this project work.

I want to thank my fellow labmates Priya Mariam Raju and Aswathy P., Research Scholars and Jinu Joseph, Project Fellow for the stimulating discussions and 'tea-time' chats, that helped let off the steam. I am also grateful to my friends Minha and Piruthvi, for being really good listeners and making these two years less stressful and more fun.

I am immensely grateful to my parents, my sisters and my grandmother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. Thank you for all the long conversations, blessings and prayers.

I thank God Almighty for being with me the whole time, especially the past two years and giving me all that I needed at the right time.

Soumya Sara John

Abstract

Despite the recent advances in embedded GPUs and memory technologies, porting the huge number of parameters of the present Neural Network models make them less flexible for real-time inference applications. Application of model compression or network pruning techniques is one of the many possibilities for reducing the effective number of parameters. Restoring the desired performance of a pruned model requires a fine-tuning step, which lets the network relearn using the training data, except that the parameters are initialised to the pruned parameters. This relearning procedure is a key component in deciding the time taken in building a hardware-friendly architecture.

This thesis studies and derives the fine-tuning or retraining conditions for pruning various neural network architectures. A mathematical analysis is carried out to obtain the lower bounds on the number of epochs(conditions for convergence) while carrying out pruning at various layers of an architecture, based on the amount of pruning done. Analyses on the propagation of errors through the layers while performing layer-wise pruning, is also performed and a new parameter named 'Net Deviation' is proposed, that can be used as an alternative to other metrics for the ranking of various pruning algorithms. Similar to the test accuracy degradation for different amounts of pruning, the net deviation curves help compare the pruning methods. In order to validate this claim, a comparison between Random pruning, Weight magnitude based pruning and Clustered pruning is performed on LeNet-300 and LeNet-5 architectures using Net Deviation. The results indicate clustered pruning to be a better option than random approach, for obtaining higher model compression.

Numerous algorithms and approaches exist, that aid in model compression. However, it is observed that the methods that provide high compression are mostly computationally very expensive and require a significant number of retraining epochs, which in a way increases the overall deployment time of the models to the desired embedded platform. In order to address this problem, a biologically inspired pruning method, i.e. 'Sub-clustering based Neurogenetic Pruning' is proposed, that provides significantly higher compression in Neural Networks with fewer computations of retraining to gain the desired performance. This feature-based technique finds the required number of nodes and filters based on the

initial clustering set by the user. Using this approach, LeNet-300-100 could be compressed by 76.803% which required just 4 epochs to attain the same performance as the original unpruned network. LeNet-5 could be compressed to 23.03% of the initial size and a LeNet-5 like architecture with more filters and nodes, could be compressed to 29.63% of the initial size with only 2 epochs for retraining. A 12-layer network trained on CIFAR-10 data set was also trained and pruned by 71.067% with 5 epochs of retraining. A modified VGG-16 architecture, also trained on CIFAR-10 was compressed by 493$\times$ with just 5% reduction in accuracy.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

NN	Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
MLP	Multi-Layer Perceptron
SVD	Singular Value Decomposition

Nomenclature

l	NN layer
L	Total number of layer in a NN
n_l	Number of nodes or filters in layer l
$\mathbf{W}^{(l)}$	Weight matrix in layer l, with size $n_l \times n_{l+1}$
$\mathbf{B}^{(l)}$	Bias vectors added in layer l, with size $n_l \times B$
B	Batch size of input
x	Input sample
K	Number of clusters
C_i	Cluster i
c_i	Cluster center of cluster i
thresh	variance threshold set using HoH algorithm
V	Intra-cluster covariance matrix
p	Percentage energy

Chapter 1

Introduction

Neural Networks have been widely used to solve various real world problems in different arenas because of its remarkable function approximation capability[3],[4]. Neural Networks are non-linear signal processors that have attained much awaited attention across various discipline ranging from stock market prediction to object recognition to object tracking. They are widely used in Natural Language Processing, Speech processing, Image and Video Processing applications. The part of Machine Learning that applies Neural Networks is defined as Deep Learning. One of the things that increased the popularity of Deep Learning is the massive amount of data that is available, which has been gathered over the last several years and decades. This enables Neural Networks to really demonstrate its potential since they always get better with more data. The recent breakthroughs in the development of algorithms help in making them run much faster than before, which makes it possible to use more and more data.

Despite of all the advances in the field of Deep Learning, including the arrival of better and improved optimization algorithms and GPUs, several questions related to optimal architecture, are still puzzling. They are computationally expensive than traditional machine learning algorithms, like Naive Bayes[5], SVM[6], Random Forest[7], etc. State of the art deep learning algorithms can take several weeks to train completely from scratch. The amount of computational power needed for a Neural Network depends heavily on the size of the data it has to process and also on how deep and complex the network architecture is. Consider a real-time application of DNNs, say in an aeroplane. Suppose a neural network trained for object recognition is used in an aeroplane to identify birds that are approaching the plane. This is designed to avoid bird-strike, which refers to the contact between a moving aircraft and a bird. Consider the plane to be moving at 900 km/hrs and the bird to be 1000 metres away from the plane. This implies that the plane has an approximate time of 4s to identify the bird, to take the decision of changing direction and to signal the relevant con-

trols to let the plane move away. The neural network is used here in the identification part only. This implies that the inference or test time of a neural network must be very small, even below 2s. However, the inference time of the state-of-the-art networks is not within 2s. This calls for Model Compression, which aims at reducing the size of the network in terms of the parameters that need to be stored. Lower the number of parameters, lower would be the number of computations required for inference and this lets the network have smaller inference time. All throughout the process, the one thing that must not be compromised is the network performance (here, it would be the bird recognition accuracy). A proper model compression method, should provide good and reasonable compression, which complies with the available memory technologies, and the desired network performance.

1.1 Computational Complexity of Neural Networks

As per complexity theory, the general learning problem for neural networks is NP complete. The space complexity denote the memory required for obtaining the algorithmic solution once a computational model has been adopted. The time complexity refers to the number of algorithmic steps executed by the computational model. Normally, space complexity is not considered to be a primary issue, since the computational models are provided with an infinitely large memory. However, the time complexity is very crucial for all algorithms and will be considered here. The term 'computational complexity' that is used in this thesis refers to time complexity alone.

The total cost per iteration of a mini-batch update, involved in the training of a DNN, usually vary linearly as per the total number of trainable weights and connections. Weight-sharing can complicate the exact computations of cost for a specific network architecture. Moreover, the most commonly used optimization procedure today is the stochastic gradient update with Nesterov momentum and Adam oprimization, due to its faster convergence and stability. Gradients are computed using backpropagation. These steps have linear cost.

Evaluating a trained neural network on a training sample is usually relatively cheap, and the cost is also linear in the number of weights. In general the runtime is proportional to the number of weights and the memory usage is proportional to the number of weights which need to be stored. So the runtime will scale linearly in the number of output nodes, quadratically in the number of hidden states, and linearly in the number of output nodes.

1.1.1 Computational Complexity of a typical Multi-layer perceptron

Consider a MLP with L layers. Let n_l denote the number of nodes in layer l, N to be the number of input patterns and e refers to the epochs for training. The time complexity in training the network is given by

$$O\big(eN \sum_{l=1}^{L} n_l n_{l-1}\big) \tag{1.1}$$

Similarly, testing the network after proper training involves a time complexity of

$$O\big(\sum_{l=1}^{L} n_l n_{l-1}\big) \tag{1.2}$$

1.1.2 Computational Complexity of a typical Convolutional Neural Network

Consider a CNN with d number of convolutional layers, with n_l number of filters in layer l, each of size s_l x s_l giving an output feature map of size m_l x m_l. The convolutional layers will then have a computational complexity of

$$O(\sum_{l=1}^{d} n_l n_{l-1} s_l^2 m_l^2) \tag{1.3}$$

Eq.(1.3) is valid for both the training and the testing. The cost of fully connected and pooling layers, which often take up only 5-10% of the computational time, is not considered in the above formulation. It is experimentally noted that the CNN performance improves with increasing number of layers and filters. This further increases the time complexity of the network. The number of parameters involved in some of the state-of-the-art Deep Neural Network architectures are given in Table 1.1.

1.2 Contributions & Scope

Pruning is an important area of research and is of much importance when it comes to real-time implementation of trained models. The trained NNs are to be used as a final product in various applications. This can done by either downloading the weights to a suitable hardware or by an online inference method. The online inference method makes

NN model	No.of parameters
Lenet300	26.6k
Lenet5[1]	60k
AlexNet[8]	62M
VGGNet[9]	138M
Inception-v3[10]	23M
ResNet-50[11]	25.5M

Table 1.1: No. of parameters in popular DNN architectures

use of Internet of things (IoT) technology, where the final product would be connected to a server that performs the computations required for inference. In both of these cases, faster inference could only be possible with fewer matrix computations. The matrix computations involved with the current state-of-the-art architectures is huge. It is therefore the pruning of weights and nodes that reduce the dimensions of the parameter matrices which leads to faster inference. The current state-of-the-art pruning procedures are very time-consuming and requires a large amount of retraining or fine-tuning procedure. Any error in the NN model, during real-time usage, could only be rectified by retraining the whole network from scratch (i.e., training the network as it was before pruning). The time taken for obtaining this rectification can then be divided into two: the first one is for training the big network and the other one for pruning. The rectification time can be reduced, if pruning time is also reduced. This thesis is an attempt to devise an appropriate pruning procedure that gives reasonable model compression in a shorter span of time.

1.3 Datasets

This section describes the data sets that were used in the simulation experiments that were conducted to obtain the results stated in this thesis.

1.3.1 Digit Recognition - MNIST

The MNIST database(Modified National Institute of Standards and Technology database) of handwritten digits[12], consists of a training set of 60,000 examples, and a test set of 10,000 examples. A sample image of the digits in this data set is given in the Fig.1.1.

Figure 1.1: Example MNIST images

1.3.2 Object Recognition - CIFAR-10

CIFAR-10 [13] is a computer-vision dataset used for object recognition. It consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. Some example images in this data set is shown in Fig.1.2.

1.4 Popular Neural Network Architectures

This section explains the network architectures that were used in the simulations done to prepare the results given in this thesis.

1.4.1 LeNet-300-100

LeNet-300-100 is a simple Multi Layer Perceptron with two hidden layers, with 300 and 100 nodes respectively. It is trained using MNIST digit data set and has a total of 2,66,610 parameters. It was trained for 4 epochs with a learning rate of 0.01, using Adam optimizer. The network gave a test accuracy of 97.769%. The name LeNet-300 used in this thesis also refers to the same network.

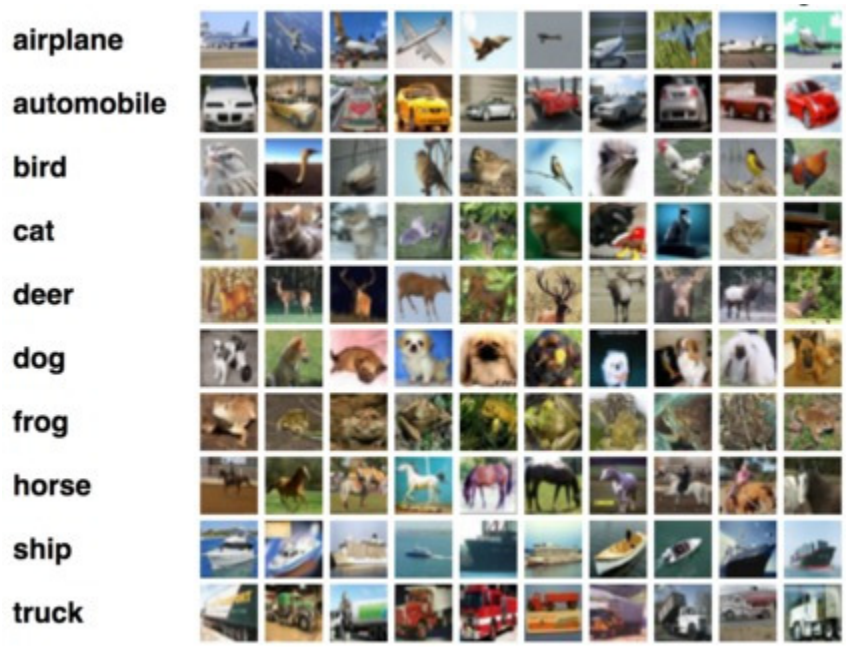

Figure 1.2: Example CIFAR-10 dataset images

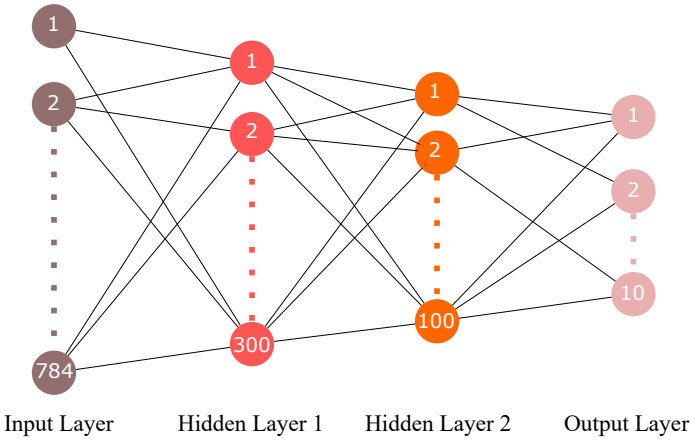

Figure 1.3: LeNet-300-100 architecture

1.4.2 LeNet-5

The LeNet-5 architecture consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, two fully-connected layers and a softmax classifier.

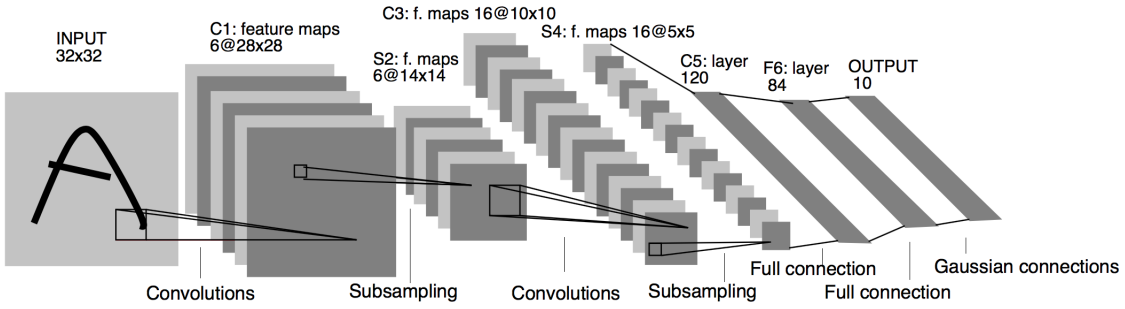

Figure 1.4: LeNet-5 architecture[1]

Each layer is briefly explained below:

First Layer: The input for LeNet-5 is a 32×32 grayscale image which passes through the first convolutional layer with 6 feature maps or filters having size 5×5 and a stride of one. The image dimensions changes from $32 \times 32 \times 1$ to $28 \times 28 \times 6$.

No. of parameters $= 5 * 5 * 6 + 6 = 156$

Second Layer: LeNet-5 applies average pooling layer or sub-sampling layer with a filter size 2×2 and a stride of two. The resulting image dimensions will be reduced to $14 \times 14 \times 6$.

No. of parameters $= 2 * 6 = 12$

Third Layer: This is a second convolutional layer with 16 feature maps having size 5×5 and a stride of one. The image dimensions change from $14 \times 14 \times 6$ to $10 \times 10 \times 16$.

No. of parameters $= 5 * 5 * 6 * 16 = 2400$

In this layer, only 10 out of 16 feature maps are connected to 6 feature maps of the previous layer as shown below. This is to break the symmetry in the network and keeps the number of connections within reasonable bounds, hence reducing the number of connections from 2000 to **1516**.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I
EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Figure 1.5: Snapshot of feature map from [1]

Fourth Layer: This is an average pooling layer with filter size 2×2 and a stride of 2. This layer is the same as the second layer (S2) except it has 16 feature maps so the output will

be reduced to $5 \times 5 \times 16$.
No. of parameters $= 2 * 16 = 32$
Fifth Layer: This layer is a fully connected convolutional layer with 120 feature maps each of size 1×1. Each of the 120 units in C5 is connected to all the 400 nodes (5*5*16) in the fourth layer.
No. of parameters $= 400 * 120 + 120 = 45120$
Sixth Layer: This is another fully connected layer with 84 nodes.
No. of parameters $= 120 * 84 + 84 = 10164$
Output layer: The output layer is also a fully connected layer with 10 nodes. This layer applies softmax function to predict the output using probabilities.
No. of parameters $= 84 * 10 + 10 = 850$
Total number of parameters in LeNet-5 = 60k
For simulations, a network similar to LeNet-5 was used. The difference lies in the number of filters and nodes. There are 32 and 64 filters in the first and second layers respectively. There are two fully-connected layers with 1024 nodes and 10 nodes after the convolutional layers. Total number of parameters for this LeNet-5 like architecture is 3.2M. It was trained on MNIST digit data for 200 epochs with a learning rate of 0.01. It was compressed by 89.4707% with retraining for 2 epochs with the same learning rate using Adam optimizer.

1.4.3 Deeper CNN

Two deep CNN architectures were designed to train and test on the CIFAR-10 data set. These networks are explained in the following sub sections.

1.4.3.1 12 layer CNN

The 12-layer CNN was trained for 10 epochs on CIFAR-10 data set and attained a test accuracy of 71.75% and validation set accuracy of 72.34%. The total number of parameters in this network is approximately 4.6M. This network layers are explained below:
CONV 1 : This layer is a convolutional layer with 64 3×3 filters with a stride of one. The activation used was ReLU function. The input was same-padded, and hence the output image size is same as that of input image. The input given was of size $32 \times 32 \times 3$ and the output is a tensor of size $32 \times 32 \times 64$
No. of parameters: $3 * 3 * 3 * 64 + 64 = 1792$
MAX POOL 1 : The output of layer 1 is then passed through a Max Pool layer with stride 2. Batch normalization is then done on the output. The output of this layer has a size of

$16 \times 16 \times 64$

CONV 2 : The third layer is a convolutional layer with 128 fliters of size 3×3 with stride one. The activation function applied is ReLU function. This layer outputs have a size of $16 \times 16 \times 128$

No. of parameters: $3 * 3 * 64 * 128 + 128 = 73856$

MAX POOL 2 : The output of layer 3 is passed through a Max Pool layer with stride 2 and then batch normalized. The output of this layer has a size of $8 \times 8 \times 128$.

CONV 3 : This is another convolutional layer with 256 5×5 filters with a stride of one. ReLU activation function was applied on the output. The output of this layer has a size of $8 \times 8 \times 256$.

No. of parameters: $5 * 5 * 128 * 256 + 256 = 819456$

MAX POOL 3 : This layer performs Max Pooling on the output from Layer 5 with a stride of 2. Batch Normalization is also done in this layer. The output of this layer has a size of $4 \times 4 \times 256$

CONV 4 : The seventh layer is a convolutional layer with 512 5×5 filters with a stride of one. ReLU activation is applied on the outputs. The output of this layer has a size of $4 \times 4 \times 512$.

No. of parameters: $5 * 5 * 256 * 512 + 512 = 3277312$

MAX POOL 4 : This layer performs Max Pooling on the output from Layer 7 with a stride of 2. Batch Normalization is also done in this layer. The output of this layer has a size of $2 \times 2 \times 512$

FC 1 : The output of layer 8 is flattened and then connected to a fully connected layer with 128 nodes and ReLU activation. Dropout is also used for training and the output is batch normalized.

No. of parameters: $2048 * 128 + 128 = 262272$

FC 2 : This is another fully connected layer with 256 nodes, ReLU activation. Dropout is also used for training and the output is batch normalized.

No. of parameters: $128 * 256 + 256 = 33024$

FC 3 : This is another fully connected layer with 512 nodes, ReLU activation. Dropout is also used for training and the output is batch normalized.

No. of parameters: $256 * 512 + 512 = 131584$

Output layer : The output layer consists of 10 nodes and softmax function is applied to get the class probabilities.

No. of parameters: $512 * 10 + 10 = 5130$

1.4.3.2 Modified VGG-16

A modified VGG-16 architecture[2] was trained on CIFAR-10 for 50 epochs with a learning rate of 0.01 and Adam optimizer. It had a validation data accuracy of 67.84% and test data accuracy of 67.207%. All the convolutional layer filters are of size 3×3 with stride 1. Max pooling was done with stride 2. The network layer details are given in Table 1.2. The total number of parameters for this network is around 15.08 million.

LAYER	**No. of filters/nodes**
CONV1	64
CONV2	64
MAXPOOL1	64
CONV3	128
CONV4	128
MAXPOOL2	128
CONV5	256
CONV6	256
CONV7	256
MAXPOOL3	256
CONV8	512
CONV9	512
CONV10	512
MAXPOOL4	512
CONV11	512
CONV12	512
CONV13	512
MAXPOOL5	512
FLATTEN	
FC1	512
OUTPUT	10

Table 1.2: Modified VGG16 architecture[2]

1.5 Hardware & Software

Simulations were performed on a 64-bit system that runs on Ubuntu 16.04 OS with 44GB RAM. It has a Intel Xeon(R) CPU X5675 and GeForce GTX 1080 Ti processors. The simulations were run using the TensorFlow library, which is an end-to-end open source platform for machine learning.

1.6 Outline of the thesis

The thesis is structured as follows. The present chapter explains the requirement of model compression and details of the experimental set ups used to validate the proposed theory and the ideas given in this thesis. Chapter 2 gives an introduction to Pruning and its importance. A brief overview about the prior art in model compression and pruning is also given in Chapter 2. Chapter 3 focuses mainly on two important aspects of this thesis: the first one is a theoretical analysis of the error propagation through the various layers as a result of pruning and the second one is to derive a theoretical bound on the relative number of epochs that a pruned network will take to achieve near identical performance as that of the unpruned network. A new parameter 'Net Deviation' is proposed which can be used as a measure of pruning efficiency. Chapter 4 delves into the clustered pruning approach, using different clustering algorithms. Chapter 5 explains the proposed method 'Subclustering based Neurogenetic Pruning' which does pruning by clustering the features from each layer. This method does not require any knowledge of the distribution of the features. It automatically discovers unique clusters from the feature data. Conclusions and future work are given in Chapter 6. Finally, theoretical proofs for all the theorems stated in Chapter 3 are give in the Appendix section.

Chapter 2

Pruning of Neural Networks

2.1 Introduction

Model Compression refers to the reduction of NN size or dimension, in terms of the number of parameters involved with it. Common model compression techniques are inspired by the fault tolerance property to network damage conditions, seen in large networks. Pruning is one such model compression technique that introduces damage to the network purposefully (by eliminating the nodes or filters or weight connections), which compromises its performance in terms of accuracy. However, a procedure of retraining can be used to regain the original performance. In general, the percentage reduction in accuracy is proportional to the amount of damage made. When the damage to the network is very large, the network requires more retraining to regain the desired level of accuracy on the particular data set. The authors in [14] conducted experiments comparing the accuracy of large, but pruned models (large-sparse) with their smaller, but dense (small-dense) counterparts and concluded that the large- sparse models outperform the small-denser models with $10\times$ reduction in the number of non zero parameters with minimal reduction in accuracy.

Pruning refers to the elimination of certain parts of a network, mainly the weight connections or the nodes in a layer or layers as such. Neural Networks have grown in terms of applications and size. Performing pruning on a pre-trained network serves the purpose of model compression. A process named Synaptic Pruning happens in the brain between early childhood and the onset of puberty in most mammals, including humans[15]. Synaptic pruning is defined as the process of synapse elimination. Synapse refers to the structure that helps one neuron communicate with other neurons, using the help of electrical or chemical signals. The weights present in neural networks are indeed inspired from these synapses. The infant brain grows in terms of synapses reaching a size of atleast 85 billion

neurons. The total number of neurons remain the same; but there would be a growth in the number of synaptic connections between them.

Pruning is widely thought to represent an effective way of learning. There are three models explaining synaptic pruning : axon degeneration, axon retraction, and axon shedding. In all these cases, the synapses are formed by a transient axon terminal, and synapse elimination is caused by the axon pruning. Each model offers a different method in which the axon is removed to delete the synapse. After adolescence, the volume of the synaptic connections decrease again due to synaptic pruning[16].

The pruning associated with learning is termed as small-scale axon terminal arbor pruning. The selection of the arbor terminal is based on synaptic plasticity. Synaptic plasticity is defined as the ability of the synapses to become stronger or weaker, based on its frequency of being active. This means synapses that are frequently used have stronger connections while the rarely used synapses are eliminated.

2.2 Categorization of Pruning Methods

Pruning can be divided into three, based on the element eliminated from the network. It could be either the connections between neurons or the nodes in MLPs and fully connected layers or the filters itself from convolutional layers.

(i) Connection Pruning or Non-Structured pruning : This aims at removing the connections or making the weight matrix sparse. The degree of compression depends on the sparsity introduced in the weight matrices. A node as a whole can only be removed when all its corresponding preceding and succeeding connections are zeroed out.

(ii) Node Pruning or Structured Pruning: Node pruning results in lower rank tensors and matrices by removing nodes as a whole, which includes all its connections. It could be seen as a more structured sparsity or structured pruning.

(iii) Filter Pruning: Filter pruning refers to the elimination of filters from the convolutional layers. This reduces the number of features extracted by the CNN. Filter pruning reduces the computational complexity more, compared to the other methods, as the complexity involved with convolutions are higher compared to those involved with matrix multiplication.

A comparative analysis between the first two methods can be seen in [17] and results show that node pruning is better at model compression than connection pruning, as the former provides stronger and structured dimension reduction.

2.3 Geometric View of Pruning

Pruning nodes of a layer l of a NN with total 'L' number of layers, imply projecting the corresponding weight matrix on a hyper plane of lower dimensionality passing through the origin. The following retraining procedure then involves the optimization of the same loss function in this hyper plane; except that the search space has now reduced to the projected hyper plane.

The number of retraining steps required for the network to reach the original testing accuracy depends on the selection of the appropriate hyper plane for projection. Projecting the weight matrices to a hyper plane indirectly reduces the degrees of freedom for weight updation (while fine-tuning) as the search space has shrunk to a subspace. This makes the retraining step to be complex, in terms of the epochs taken.

Also, the hyper planes constructed using the dimensions with higher projection will have lower retraining time. Projecting the weight matrix to a hyper plane constructed in this way will have less projection error and thus will require fewer number of epochs for retraining. Such projections will have only a small reduction in testing accuracy even without retraining.

2.4 Literature Survey related to Model Compression

Deep Neural Networks are computationally very expensive. The search for a method to find the optimum architecture for a particular problem statement and dataset, started from a time when DNNs were not that popular[18]. Different solutions to this age-old problem have been proposed and one technique is to train a large network and then eliminate connections and nodes, the technique termed as pruning. This section aims to condense this widely studied area in terms of the key concept used and the drawbacks involved.

Significant contributions started in the early 90s, when the researchers tried to define an appropriate measure to clearly distinguish between the required connections and the

ones whose absence doesn′t deteriorate the network performance. [18] defined parameters “Consuming energy" and “Weights power" to select the appropriate units and weights. Similar measure is proposed in [19], where the authors define the parameters using both the weights as well as the activation outputs. [20] proposed a pruning method based on the idea of iteratively eliminating units and adjusting the remaining weights, by solving a set of linear equations. [21], [22], [23] use second order derivatives in the Taylor Series expansion to choose the nodes to be pruned. [24],[25],[26] prune irrelevant nodes based on the sensitivity of the nodes. The sensitivity of a node is defined as the expectation of its output deviation due to expected deviation in the input, with respect to the overall inputs in a continuous interval. In [26], sensitivity is determined by analysing the Fourier decomposition of the variance of the model output. The authors in [27] proposed a new relevance measure based on the mutual information between the models output and the node output. All these methods, except [23] were performed on small datasets and smaller networks. The measures defined using second order derivatives and sensitivity measures, are difficult to compute for larger datasets and networks, even though they give better compression with comparatively less performance deterioration.

A simple pruning approach is given in [28], where the pruning is seen to be a three step process. The first step is the usual training of a large network on a particular dataset. The low weight connections in the network are then pruned , making the network sparse. The pruned network is then retrained till the required accuracy is achieved. [28] implements iterative pruning where pruning followed by retraining is considered to be a single iteration. The minimum number of connections can be found only after many such iterations. A method of model compression that can be applied on both the pre-trained model and untrained model is given in [29]. This gives flexibility in trying different combinations of the possible compressed models while the network tries to learn. This might not be a right technique as the designer may not be aware of the accuracy of the network after proper training.

Pruning can also be done in the frequency domain, as seen in [30]. This would also have the added advantage of reducing the amount of computations in the convolutional layers, as convolution in spatial domain is equivalent to multiplication in frequency domain. Complex domain makes the whole pruning process a bit cumbersome, when it comes to fine-tuning the pruned network.

An alternative approach to pruning is to keep just the unique nodes and connections in the network. One approach to find the set of unique nodes is to exploit the redundancy present in the features. To implement the same, [31] applies k-means clustering on the features, that group nodes in each layer to different clusters. The node closest to the cluster centre is allowed to stay and the rest are removed. The number of unique clusters (k) is an unknown parameter. A similar approach can be seen in [32], where the authors propose a core-set based compression. The method is a search for the appropriate core-sets of the parameters so that the network performance will not be degraded. This method has the added advantage of no requirement of retraining step after pruning. The paper [33] looks into the similarity between the connecting weight matrices of each node and wire similar nodes by simple addition of parameters. [34] applies a hashing trick on the weights, where certain connections are randomly selected into a hash bucket and all those connections are expressed by the same representation. This helps to reduce the memory overhead. [35] proposed a new method named NoiseOut, which is also based on the same concept, it finds out the correlation between the activations of each neuron after adding new neurons(noise) to the original trained model. These additional neurons increase the correlation between the corresponding activations, which helps in better compression.

Another interesting way of pruning is to learn a compressed model from the trained model in a teacher-student manner. The original trained model acts as the teacher and the compressed model as the student. Popularly known as Knowledge Distillation, this work can be found in [36] and is one of the state-of-the-art techniques of model compression. This student network learns the parameters from the teacher network in such a way that the soft targets remain unchanged. [37] removes redundancy by embedding a small network named “D-Pruner", which evaluates the importance of filters and removes them in the fine-tuning phase without much reduction in the accuracy of the original model. It essentially contains a separate neural network that predicts which filters must be removed within the final network architecture. The algorithm works in multiple pruning iterations. A neural network can be trained to generate model descriptions from scratch as seen in [38], where a Recurrent Neural Network is trained using Reinforcement Learning to help in architecture design. Similar approaches are seen in [39] and [40]. The pruning procedure is modelled as a Markov Decision Process in [39] and an agent is trained using reinforcement learning. The agent conducts channel wise pruning based on the importance of the filters in the convolutional layers. [40] introduces a try and learn algorithm, where pruning agents are trained for each layer. Unnecessary filters are removed in a data-driven manner. All the

mentioned methods require extra hardware to train an agent and only increases the computations involved with creating a compressed NN.

Pruning can be done by matrix decomposition methods like SVD or Fastfood transform or CP decomposition and this idea is seen to be implemented in [41],[42] and [43]. These projection methods are capable of choosing the higher energy nodes, but finding the proper projection matrix is a difficult task. As seen in [43], such methods are more useful to reduce the computations involved in convolutional layers, where the fully connected layers remain untouched. [43] applies Adaptive Fastfood transform on both convolutional and fully connected layers, but this approach involves learning of certain matrices, using back propagation. Learning of matrices involve huge computations, especially when the data and the network is large.

Deeper compression can be achieved by combining pruning with quantization of the parameters as can be seen in [44] and [45]. The idea “Deep Compression" was introduced first in [44], where the proposed method is a combination of pruning, trained quantization and Huffman coding, which reduces the storage requirement by 35x to 49x, without affecting the accuracy. This method is another popularly used approach, even in mobile applications. On the other hand [45] proposes in-parallel pruning and quantization, which takes advantage of the complementary nature of the two methods pruning and quantization. This allows self healing to the compressed model due to independent errors brought by parallel pruning and quantization. But this method requires certain pruning-quantization hyper parameters that are computed using Bayesian optimization and hence turns out to be computationally very expensive.

A recent trend in model compression is the application of optimization theory, as could be seen in [46], [47], [48], [49] and [32]. Pruning is implemented as a search algorithm to find the compressed weight parameters by running an optimization problem defined on the trained weight matrices. Interestingly, the work in [32] has the added advantage of no requirement of fine-tuning or retraining step after pruning. Even though this approach results in very high compression results, the pruning process is computationally very expensive, especially for larger networks as the search might take more time than it took for training the network in the first place.

2.5 Discussion

The different methods to prune Neural Networks have been explained briefly in Section 2.4. A common procedure in all of these methods is the requirement of the fine-tuning step after pruning the network, to maintain the desired performance level. The computational complexity involved in general for each of the category of the above explained pruning procedures is listed below:

- Weight magnitude based methods:
  The basic concept of weight magnitude based methods calculate the sum of absolute magnitude of weight connections to each node. For any node i in layer l, there would be $n_{l-1} + n_{l+1}$ connections and hence the number of additions involved would be $O(n_{l-1} + n_{l+1})$. n_l such additions need to be done for layer l. Therefore, the whole computational complexity involved is $O(\max_l n_l(n_{l-1} + n_{l+1}))$.

- Matrix decomposition methods:
  The simplest matrix decomposition method is Singular Value Decomposition(SVD), where SVD of the weight matrices are calculated and approximated, to compress the network. SVD of a $m \times n$ matrix takes $O(mn^2)$. If SVD is done on a MLP, the computational complexity becomes $O(\max_l n_l n_{l+1}^2)$. There are methods in this category with reduced computational complexity, like [43], which proposes Adaptive Fastfood Transform. Here, for a layer l with d nodes and with square weight matrix, the complexity involved for pruning using Fastfood transform was $O(d \log d)$.

- Clustered pruning:
  Authors in [31] proposed a feature-based pruning method, that uses k-means clustering to collect clusters. k-means algorithm has a computational complexity of $O(n)$, where n is the number of nodes. The node closer to the cluster center is kept and the rest eliminated. Here, the distance of the features from the cluster center needs to be calculated, which has a complexity of $O(\sum_{k=1}^{K} n_k) = O(n)$. Sorting needs to be done which has a complexity of $O(\sum_{k=1}^{K} n_k log(n_k))$, $O(n \log n)$ in the worst case. The overall complexity for 'k' clusters is [31] implements this for many 'k' and chooses the best one, and if T such searches are done, the overall computational complexity is $O(T \max_l n_l \log (n_l))$.

- Agent based method:
  These methods involve learning another neural network from scratch, to see which nodes are relevant and important. This learning procedure is itself a tedious work, as it involves training a NN. The computational complexity involved will be based on the size of the learning network and the data.

- Optimization techniques:
  [32] mentions the computational complexity to be $O(n.A.\log N_k.(1+s))$, where n is the number of layers, A denotes the complexity to do one feed-forward operation on the entire training set, N_k denotes the number of filters and s is the fraction of random training points used for coreset construction set. The complexity involved to do one feed-forward operation on the entire training set is itself very large, for larger networks and data sets. While in [47], at every layer, an individual Net-trim program is solved for every output neuron. After compression, fine-tuning is also done to better the performance.

2.6 Summary

The best compression on a DNN currently reported is that of [47], which use optimization techniques ($200\times$ and $409.5\times$ of compression in LeNet-300-100 and LeNet-5 respectively). But, these techniques are computationally very expensive. The methods with lower computational complexity does not give high compression, with a low number of retraining epochs. A good compression technique hence requires two things: higher compression at lower computational cost. Compromising between the requirement of better compression, lower degradation in accuracy and the computational complexity involved, we choose clustered pruning to be a proper approach to prune deep neural networks. 'Sub-clustering based Neurogenetic Pruning' is an extension to clustered pruning, that provides higher model compression, with fewer computations.

Chapter 3

Analysis of the fine-tuning step after pruning a Neural Network

3.1 Overview

The pruning of neural network changes the overall response of the network. The amount of the resulting change, depends on the number of connections and nodes removed. Restoring the desired performance on the pruned model requires a fine-tuning step which can be defined as the process of letting the network learn again, using the training data, except that the parameters are initialised to the pruned parameters. This chapter aims to look into two aspects:

1. Deriving the theoretical bounds for determining the number of epochs required in a pruned sparse network to reach the original performance, relative to the number of epochs the original un-pruned network had taken to reach the desired performance.

2. A rigorous analysis of pruning error propagation through the layers, which explains the contribution of the pruning of early layers to the final layer. A parameter defined as ‘Net Deviation" is also introduced, that helps to select the best pruning procedure for a particular trained network and data.

Detailed proofs for the proposed theorems are given in Appendix A and B.

3.2 Theoretical Bounds on the number of epochs for fine-tuning a Sparse Neural Network

Consider a Multi-Layer Perceptron with 'L' layers, with $n^{(l)}$ nodes in layer l. Corresponding weights and biases are denoted as $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \ldots, \mathbf{W}^{(L-1)}$ and $\mathbf{B}^{(2)}, \mathbf{B}^{(3)}, \ldots, \mathbf{B}^{(L)}$ respectively, where $\mathbf{W}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l+1)}}$ and $\mathbf{B}^{(l)} \in \mathbb{R}^{n^{(l)} \times 1}$. The loss function is denoted as $L(\mathbf{W})$. This could be mean-squared error or cross-entropy loss, defined on both the weights and biases, using the labels and the predicted outputs.

The initial weight matrices and initial loss are denoted as $\mathbf{W}^{(1)}_{initial}, \mathbf{W}^{(2)}_{initial}, \ldots, \mathbf{W}^{(L-1)}_{initial}$ and $L(\mathbf{W}_{initial})$, respectively. Similarly, the sparse weight matrices are denoted as $\mathbf{W}^{(1)}_{sparse}$, $\mathbf{W}^{(2)}_{sparse}, \ldots, \mathbf{W}^{(L-1)}_{sparse}$ and corresponding loss is denoted as $L(\mathbf{W}_{sparse})$. The weight matrix after training is denoted as $\mathbf{W}^*$. The number of epochs the network had taken to train till the desired performance from randomly initialised weights $\mathbf{W}_{initial}$ is denoted as $t_{initial}$. Let t_{sparse} be the number of epochs the pruned or sparse network took to attain the same performance as the earlier un-pruned network.

If the total number of parameters in the network is M, the parameters can be made p-sparse in $\frac{M!}{p!(M-p)!}$ number of ways. Hence, the bounds provided below must be understood in the average sense (average over a large number of runs). Assume that the loss function is continuously differentiable and strictly convex. The losses decide the number of epochs the network takes to reach convergence and hence, the number of epochs to reach convergence can be viewed to be directly proportional to the loss.

Theorem 1. *Given a trained network $N(\{\boldsymbol{W}_l\}_{l=1}^{L}, \boldsymbol{X})$, trained from initial weights $\boldsymbol{W}_{initial}$ using $t_{initial}$ epochs. For fine-tuning the sparse network $N_{sparse}(\{\boldsymbol{W}_l\}_{l=1}^{L}, \boldsymbol{X})$, there exists a positive integer γ that lower bounds the number of epochs (t_{sparse}) to attain the original performance as,*

$$t_{sparse} \geq \frac{\gamma \mu_1}{\mu_2} \left[\frac{||\nabla L(\mathbf{W}_{initial})||^2}{||\nabla L(\mathbf{W}_{sparse})||^2} \right] t_{initial} \tag{3.1}$$

The variables μ_1 and μ_2 are defined in differently for connection and node pruning and is given in eqns. (3.2) and (3.3). It is to be noted that the definition for node pruning is valid for connection pruning as well. The value of γ is an unknown parameter in the bound and the designer has to try and keep different values of γ to see a range of possible epochs the

network could take. It might be possible to find a appropriate relation for γ using manifold learning. The values of γ given in Section 3.4.1 is obtained through careful observation of the epochs obtained in the simulation. γ depends on the data batch size used for retraining, the optimizer used and the learning rate and the amount of pruning done.

Connection Pruning:

As described in Section 2.2, connection pruning results in a sparse parameter matrix of the same size as that of the unpruned network and hence μ_1 and μ_2 can be defined as

$$\mu_1 \leq \frac{||\nabla L(\mathbf{W}_{initial}) - \nabla L(\mathbf{W}^*)||}{||\mathbf{W}_{initial} - \mathbf{W}^*||} \tag{3.2}$$

$$\mu_2 \leq \frac{||\nabla L(\mathbf{W}_{sparse}) - \nabla L(\mathbf{W}^*)||}{||\mathbf{W}_{sparse} - \mathbf{W}^*||} \tag{3.3}$$

Node and Filter Pruning:

Node and filter pruning reduces the rank of the parameter matrix and hence eqns. (3.2 and 3.3) cannot be used. PL inequality is used instead.

$$\mu_1 \leq \frac{||\nabla L(\mathbf{W}_{initial})||^2}{||L(\mathbf{W}_{initial}) - L(\mathbf{W}*)||} \tag{3.4}$$

$$\mu_2 \leq \frac{||\nabla L(\mathbf{W}_{sparse})||^2}{||L(\mathbf{W}_{sparse}) - L(\mathbf{W}*)||} \tag{3.5}$$

When there are different hidden layers, the gradients would follow the chain rule and the following equation can be incorporated in eqn.(3.1) to obtain the bound.

$$||\nabla L(\mathbf{W})||^2 = ||\nabla L(\mathbf{W}^{(1)})||^2 + ||\nabla L(\mathbf{W}^{(2)})||^2 + \ldots + ||\nabla L(\mathbf{W}^{(L-1)})||^2 \tag{3.6}$$

3.2.1 Loss function with a regularisation term

Adding a regularisation term still keeps the loss function strongly convex, if the initial loss function is strongly convex. This makes the above theorem and relations valid even for loss functions with regularisation terms.

3.3 Theoretical Analysis of the Error Propagation in Sparse Neural Network

Pruning process can be made parallel if it is done layer wise. Consider pruning to be done with the constraint that the output change doesn't go beyond some ϵ. The information regarding how much pruning can be done in each layer would be useful to keep the constraint intact while pruning. For the same reason, the layer wise error bound(ϵ_l), with respect to the overall allowed error(ϵ) needs to be known. This section hence looks into the individual contributions of the change in the parameter matrices in each layer to the final output error.

Consider a Multi-Layer Perceptron with 'L' layers, with n_l nodes in layer l. Corresponding weights and biases are denoted as $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \ldots, \mathbf{W}^{(L-1)}$ and $\mathbf{B}^{(2)}, \mathbf{B}^{(3)}, \ldots, \mathbf{B}^{(L)}$ respectively, where $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ and $\mathbf{B}^{(l)} \in \mathbb{R}^{n_l \times 1}$. The output of the neural network is given as

$$\mathbf{Y}^{(L)} = f(\mathbf{W}^{(L-1)^T}\mathbf{Y}^{(L-1)} + \mathbf{B}^{(L)}) \tag{3.7}$$

If pruning connections in $\mathbf{W}^{(L-1)}$ introduced a change of $\delta\mathbf{W}^{(L-1)}$ and pruning the earlier parameter matrices introduced a deviation of $\delta\mathbf{Y}^{(L-1)}$, then the output error introduced can be bounded as

$$||\delta\mathbf{Y}^{(L)}|| \leq ||\frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{W}^{(L-1)}}\delta\mathbf{W}^{(L-1)}|| + ||\frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{Y}^{(L-1)}}\delta\mathbf{Y}^{(L-1)}|| \tag{3.8}$$

Theorem 2. *Assuming that the input layer is left untouched, the output error introduced by pruning the trained network $N(\{\boldsymbol{W}_l\}_{l=1}^{L}, \boldsymbol{X})$ will always be bounded by the following relation,*

$$||\delta\mathbf{Y}^{(L)}|| \leq \sum_{l=2}^{L}\left[\prod_{\substack{i=l+1 \\ (l\neq L)}}^{L}||\frac{\partial\mathbf{Y}^{(i)}}{\partial\mathbf{Y}^{(i-1)}}||\right]||\frac{\partial\mathbf{Y}^{(l)}}{\partial\mathbf{W}^{(l-1)}}||||\delta\mathbf{W}^{(l-1)}|| \tag{3.9}$$

The above relation essentially explains the accumulation of the error in each layer to produce the error in the final layer i.e., if ϵ is the total allowed error in the final layer, then it can be bounded as the sum of different layer errors as shown below:

$$\epsilon \leq \epsilon_2 + \epsilon_3 + \ldots + \epsilon_L \tag{3.10}$$

where ϵ_l is the individual layer errors for $l = 2, 3, \ldots, L$.
The above equation sets apart error bounds on different layers and will be of much help in

optimisation-based pruning techniques.

$\epsilon_1 = 0$
$\epsilon_L = ||\frac{\partial \mathbf{Y}_L}{\partial \mathbf{W}_L}\delta\mathbf{W}_L||$

$$\epsilon_l = \left[\prod_{i=l+1}^{L} ||\frac{\partial \mathbf{Y}^{(i)}}{\partial \mathbf{Y}^{(i-1)}}||\right] ||\frac{\partial \mathbf{Y}^{(l)}}{\partial \mathbf{W}^{(l-1)}}|| ||\delta\mathbf{W}^{(l-1)}||, \text{ for } l = 2, \ldots, L-1$$

Assume if the layer wise error $\epsilon_l = 0$. This results into the simple equation

$$\delta\mathbf{W}^{(l-1)^T}\mathbf{Y}^{(l-1)} = \mathbf{0} \tag{3.11}$$

Two design practices can be explained theoretically using the above equation.

1. *An optimised structure of Multi-Layer Perceptrons will have $n_l \geq n_{l+1}$, where n_l denotes the number of nodes in layer l and $l = 2, 3, \ldots, L$.*
   Since $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ and $\mathbf{Y}^{(l)} \in \mathbb{R}^{n_l \times 1}$, for the above equation to have a solution, $n_l \geq n_{l+1}$. Thus the minimum number of nodes the hidden layers can have equally, so that the network trains from the data, is the number of nodes in the output layer.

2. *Data dependent approaches results in better compression models.*
   Each neural network is unique because of its architecture and the data it was trained on. Any pruning approach must not change the behaviour of the network with respect to the application it was destined to perform. Consider $\mathbf{Y}^{(l)} \in \mathbb{R}^{n_l \times B}$, where B is the batch size and $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$. For the above equation to be satisfied, the column space of $\mathbf{Y}^{(l)}$ must lie in the null space of $\delta\mathbf{W}^{(l)}$ and vice-versa. This implies that the entries for the appropriate $\delta\mathbf{W}^{(l)}$ can be obtained when the pruning measure is coined based on the features obtained from that layer.

Net Deviation(D):
The normalized difference between the obtained error difference and the bound (from eqn. (3.8)) is defined as Net Deviation. The Net Deviation curves for different sparsity ratios could be used as an alternate parameter to test accuracy, to help find the better pruning approach, for the desired compression and network. Net Deviation uses the validation data that was used for pruning. This hence, doesn't require reloading of the test data each time, to check on the pruning efficiency, in terms of test accuracy. The use of Net Deviation is explained in Section 3.4.2, where two pruning methods, Random, Weight magnitude based and Clustered Pruning are evaluated for different pruning ratios on LeNet-300 and LeNet-5

networks.

$$dev = ||\delta \mathbf{Y}^{(L)}|| - \sum_{l=2}^{L} \left[\prod_{\substack{i=l+1 \\ (l \neq L)}}^{L} || \frac{\partial \mathbf{Y}^{(i)}}{\partial \mathbf{Y}^{(i-1)}} || \right] || \frac{\partial \mathbf{Y}^{(l)}}{\partial \mathbf{W}^{(l-1)}} || || \delta \mathbf{W}^{(l-1)} || \tag{3.12}$$

$$D = \frac{dev}{||dev||} \tag{3.13}$$

3.4 Simulation Results and Discussion

This section delves into two main results: the first part proves the theoretical bound on the number of retraining epochs, stated in Theorem 1 and the second part explains the application and advantages of the parameter 'Net Deviation' in ranking pruning algorithms for a particular dataset.

3.4.1 Theoretical Bounds to the number of epochs required for fine-tuning a pruned network

Both the networks LeNet-300 and LeNet-5 were pruned randomly with the same seed in two ways: Connection Pruning and Node Pruning. The results of connection pruning is given in table 3.1, while those of node pruning is shown in table 3.2.

Sparsity Ratio	LeNet-300-100		LeNet-5	
	Theoretical Lower Bound	Average Epochs taken	Theoretical Lower Bound	Average Epochs taken
0.1	0.24	2.2	1.26	2.3
0.6	1.48	2.4	2.783	2.8
0.9	2.26	2.8	1.62	2.9

Table 3.1: The number of epochs taken for fine-tuning connection-pruned networks at different sparsity ratios.

Pruning Ratio	LeNet-300-100		LeNet-5	
	Theoretical Lower Bound	Average Epochs taken	Theoretical Lower Bound	Average Epochs taken
0.1	0.23	1.7	0.52	2
0.5	2.6	2.6	5.049	13
0.9	19.71	50	18.623	25.6

Table 3.2: The number of epochs taken for fine-tuning node-pruned networks at different pruning ratios.

The value of γ was experimentally found out in all the cases and are as follows. For LeNet-300 pruned at pruning ratios 0.1, 0.5 and 0.9, the suitable value of γ was found to be 0.01, 1 and 100 respectively. For sparsity ratios of 0.1, 0.6 and 0.9, for the same architecture, the value of γ was found to be 1e-4, 0.2 and 0.5, respectively. Similarly for LeNet-5, γ was found to be 1e-5, 2e-4 and 5e-5 for pruning ratios 0.1, 0.5 and 0.9 and 3e-5, 1e-6 and 1e-5 for sparsity ratios 0.1, 0.6 and 0.9 respectively. The results validate the bound provided in Theorem 1.

3.4.2 Analysis of Net Deviation

To explain the application of the parameter 'Net Deviation', LeNet-300 and LeNet-5 were pruned using Random Pruning, Weight magnitude based Pruning and Clustered Pruning approach. In random pruning, the connections were made sparse randomly, while in clustered pruning, the features of each layer were clustered to the required pruning level. One out of each node or filter in the cluster is kept and the rest pruned. Weight magnitude based pruning method ranked the connections based on magnitude and pruned lower magnitude connections. The results for different percentage of pruning in an average sense, are given in the figures 3.1 and 3.2.

It could be inferred from Fig 3.1 and 3.2 that for lower level of pruning, the value of net deviation is lower for random pruning approach. But for higher pruning or higher model compression, random pruning is not a good pruning method to look into. Net deviation is calculated using the same batch of data used for pruning. Test data was not used and hence Net Deviation could be used as an alternate measure to compare pruning methods, other than the Test Accuracy, where, test data needs to be used.

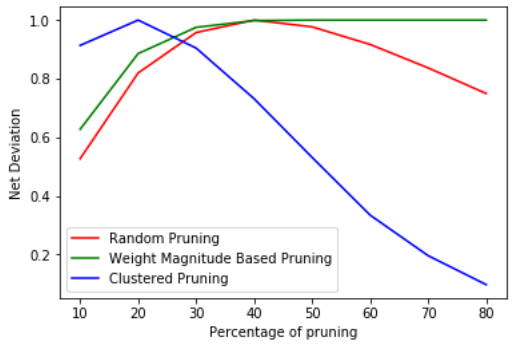

Figure 3.1: Comparison of pruning methods on LeNet-300 for different percentage of pruning, using Net Deviation.

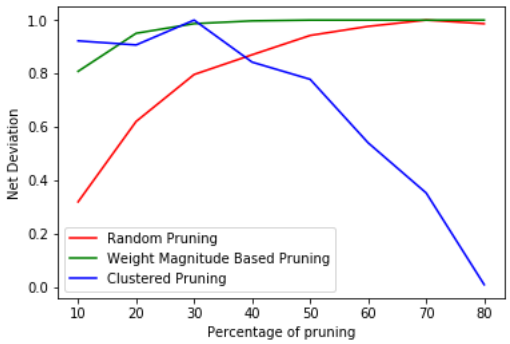

Figure 3.2: Comparison of pruning methods on LeNet-5 for different percentage of pruning, using Net Deviation.

Comparison between Net Deviation and Test Accuracy:

Consider pruning LeNet-300 network using three pruning methods- Random pruning, Weight magnitude based pruning and Clustered pruning. Net Deviation has the advantage of not using the test data as test accuracy does. D can be calculated using the same data that was used for pruning. Apart from this, consider the comparison between the two, for judging the best method for the desired compression.

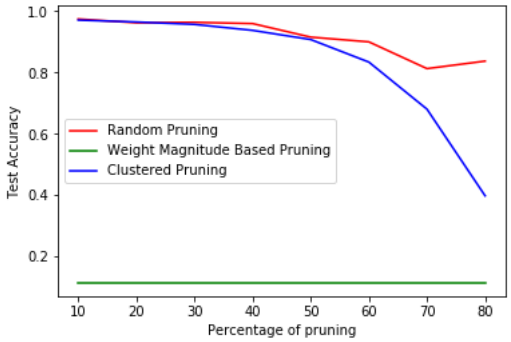

Figure 3.3: Test Accuracy curves for different pruning methods on LeNet-300.

The net deviation and test accuracy values are calculated on an average for all the three methods and plotted in figures 3.1 and 3.3. It could be seen that when choosing the appropriate method for pruning, for a particular data set and network, test accuracy does not give much information with respect to the amount of compression or the percentage of pruning. Random and clustered pruning may look equally good for lower compression. The pruning efficiency would be judged to be very bad for weight magnitude based method. But, from fig 3.3, it could be understood that, random pruning is better than clustered pruning for

lower compression, as they have lower D value. Similarly, for higher compression, clustered pruning is the best, compared to the other two methods. This inference is intuitively more sensible as well. Because of similar inferences, random pruning could be applied by a user, who wants smaller compression, with lower computational complexity as random pruning is computationally less expensive than clustered pruning. On the other hand, the weight magnitude based method is good for lower compression, which perfectly fits its usage for smaller data sets, that require only small compression[18]. Although this parameter is very promising, it is difficult to calculate for large networks, due to the involvement of gradients.

3.5 Summary

This chapter explored the theoretical bounds on the number of epochs that a pruned network would take to reach the original performance as the unpruned one. Since node pruning is the most preferred approach of pruning, the value of γ will always be between 0 and 1. This would be useful for a NN designer who after setting a suitable γ value, knows what to expect during retraining of the pruned network (in terms of the epochs it would take for retraining). Third section explained the error propagation through the layers, where the term 'error' indicates the error due to pruning each layer. A new parameter 'Net Deviation (D)' was also introduced and its advantages and uses are explained through an example application of the same on three pruning methods. It turns out that, D curves on different pruning percentages can help compare different pruning methods and choose the appropriate pruning method for the required compression, without the test data. Additionally, it could be noted from this chapter that, data-dependent methods (like clustered pruning) are better than data-independent methods (like random and weight magnitude based pruning). This is the reason why, in the next chapter, we focus more on clustered pruning method.

Chapter 4

Clustered Pruning

4.1 Introduction

Clustered pruning approach tries to preserve only the unique features of the network. Each node outputs certain features of the input image. Pruning of nodes remove the use of certain features which might be required by the network, and hence the deterioration in network performance explained. Clustering techniques group similar features into clusters. Keeping a single node from the cluster removes the redundancy present among the features.

4.2 Review of Clustering Algorithms

Clustering is a method of unsupervised learning. It is a common tool used in statistical analysis of data. Many clustering algorithms exist and a few that are relevant to our work are explained in this chapter. The theory is explained for clustering unlabelled data $\mathbf{X}$, with features $\mathbf{x}_i$, where $i = 1, 2, ..., N$. Each feature $\mathbf{x}_i$ is d-dimensional. A cluster is a collection of data points of minimum distance. Each cluster is defined by its cluster center.

4.2.1 k-Means Clustering

The k-means clustering algorithm [54], [55] is the most popular and widely used algorithm. This algorithm clusters the data points into k clusters, where the k is unknown and must be selected by the designer. k cluster-centers are initialised randomly. Each feature point $\mathbf{x}_i$ is assigned to a cluster based on

$$\underset{C_j \in C}{\operatorname{argmin}} \operatorname{dist}(\mathbf{c}_j, \mathbf{x}_i) \tag{4.1}$$

The cluster centres are then updated as

$$\mathbf{c}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i \tag{4.2}$$

$|C_j|$ is defined as the number of feature points in cluster j. The k-means algorithm iterates between the above two steps to create proper clusters with high inter cluster variance. Each cluster C_j is defined as the set of points $\mathbf{x}_i$ such that

$$C_j = \{\mathbf{x}_i : dist(\mathbf{c}_j, \mathbf{x}_i) < dist(\mathbf{c}_l, \mathbf{x}_i), \forall j \neq l\} \tag{4.3}$$

$\mathbf{c}_j$ and $\mathbf{c}_l$ are the cluster centres of the clusters C_j and C_l respectively and $C_j, C_l \in C$.

Computational Complexity:

The algorithm has an original time complexity of $O(N^2)$ but [56] gives an alternate approach that implements k-means clustering in linear complexity, i.e., $O(N)$. This is the fastest clustering method available so far.

Advantages:

- Fast convergence

Disadvantages:

- Unknown value of number of clusters k
- Random initialisation of cluster centers. This makes the results that may not be repeatable and less consistent.

4.2.2 Spectral Clustering

In spectral clustering[57], the data points are treated as nodes of a graph. Clustering is then considered to be a graph partitioning problem. The nodes are mapped to a low-dimensional space that can be easily segregated to form clusters. Spectral clustering techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the dataset. Spectral Clustering requires the number of clusters to be specified. It works well for a small number of clusters but is not advised when using many clusters. The clustering algorithm consists of three steps:

(i) *Creation of the similarity graph between the N data points to cluster*
A graph consists of edges and vertices. The edges can be calculated using either ϵ or k nearest neighbourhood method. In ϵ-neighborhood method, each vertex is connected to vertices falling inside a ball of radius ϵ. In k nearest neighbor method, each vertex is connected to its k-nearest neighbors.

(ii) *Compute the first k eigen vectors of its Laplacian matrix*
Project each data point onto the k eigen vectors to get lower-dimensional feature points.

(iii) *Apply k-means on the projected data*
Apply k-means clustering algorithm on the k-dimensional data points to get the clusters.

Advantages:

- Easy to implement and gives good clustering results
- Reasonably fast for sparse data points

Disadvantages:

- Use of k-means in the algorithm implies that the results may not be consistent all the time
- Computationally expensive due to the calculation of eigen values, especially when the data set is large.

4.2.3 Hierarchical Clustering

Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. A proximity matrix is created between the nodes based on some distance metric. The distance metric could be min, max, group average, distance between centroids or ward′s method. Based on the metric, there are different types of hierarchical clustering. Some of them are explained below:

(i) **Single linkage algorithm:**
This algorithm uses the MIN similarity criterion, where similarity between clusters C_j and C_k is defined as the minimum distance between any two points from cluster j and k. In simple words, pick the two closest points such that one point lies in cluster

C_j and the other point lies in cluster C_k and take their similarity and declare it as the similarity between two clusters. This approach can separate non-elliptical shapes as long as the gap between two clusters is not small, but it fails in the presence of noise.

(ii) **Complete linkage algorithm:**
This algorithm uses the MAX similarity criterion, where similarity between clusters C_j and C_k is defined as the maximum distance between any two points from cluster j and k. In other words, pick the two farthest points such that one point lies in cluster C_j and the other point lies in cluster C_k and take their similarity and declare it as the similarity between two clusters. This approach works well even in the presence of noise, but tends to break large clusters and is biased towards globular clusters.

(iii) **Group-average based similarity:**
This algorithm finds the average of all the pairwise similarities between the points in two clusters and declare it as the similarity between the two clusters. Similarity between clusters C_j and C_k is given as

$$sim(C_j, C_k) = \frac{1}{|C_j||C_k|} \sum_{x_{jm} \in C_j, x_{kn} \in C_k} sim(x_{jm}, x_{kn}) \tag{4.4}$$

Group Average approach does well in separating clusters if there is noise between clusters, but is biased to globular clusters.

(iv) **Distance between centroids:**
This algorithm computes the centroids of two clusters C_j and C_k and takes the similarity between the two centroids as the similarity between two clusters. This technique is not used much.

(v) **Ward′s method:**
This method [58] minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function. This method works well even in noisy data.

Hierarchical clustering is less efficient and has a time complexity of $O(N^3)$. It has a space complexity of $O(N^2)$. Hence, this clustering technique cannot be used for large data sets.

4.2.4 Agglomerative Clustering

Agglomerative clustering performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together. The algorithm consists of the following steps:

(i) Calculation of the proximity matrix, based on suitable distance metric

(ii) With each data point seen as a cluster, merge two closest clusters

(iii) Update the proximity matrix

(iv) Repeat steps (ii) and (iii) until fewer clusters are left.

4.2.5 Density-based Spatial Clustering of applications with noise

DBSCAN or Density-based Spatial Clustering of applications with noise [59] groups together points based on a distance measurement and the minimum number of points that defines a dense cluster. The points in lower density are marked as noise. It can attain arbitrary cluster shapes as well. The steps of the clustering algorithm are summarised below:

(i) Choose an arbitrary point from the database as a seed.

(ii) Retrieve all points that are density-reachable from the seed, which results in the cluster containing the seed. Density - reachable points are defined as the set of points with a size higher than the minimum number of points required to define it as a cluster with each point in the ϵ-neighborhood of the seed. Points that create a less dense cluster, are noted as outliers.

The above two steps are iteratively performed to get the clusters.
Advantages:

- Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.
- Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.
- Good efficiency on large databases

- Identifies outliers

Disadvantages:
It doesn't perform as well as the other clustering algorithms as the clusters are of varying density. It is due to the parameters ϵ and minimum number of points for identifying neighborhood points, which will vary from cluster to cluster as the density varies. Hence, this method is not suitable for higher dimensional data.

4.2.6 EM clustering

Expectation Maximization clustering method is a technique similar to k-means clustering. The number of clusters must be defined beforehand. The EM clustering algorithm computes probabilities of cluster memberships based on one or more probability distributions. The goal of the clustering algorithm then is to maximize the overall probability or likelihood of the data, given the clusters. The details of the basic EM algorithm can be found in [60]. EM algorithm does clustering by fitting a Gaussian Mixture Model(GMM) to the data points, which inherently finds the probability distributions behind the data. The clusters could then be defined using the parameters defining the probability distributions. The steps involved in EM algorithm are briefly explained below to find the GMM defined as:

$$p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k N(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.5}$$

(i) Define the number of clusters K. Initialize the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$ and mixing coefficients π_k for cluster k. Evaluate the initial value of the log likelihood. Log-likelihood is given by

$$lnp(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} ln\left\{ \sum_{k=1}^{K} \pi_k N(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \tag{4.6}$$

(ii) **E step:** Evaluate the responsibility matrix γ for $n = 1, ..., N$ data points as rows and $k = 1, ..., K$ clusters as columns, using the equation

$$\gamma(z_{nk}) = \frac{\pi_k N(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j N(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{4.7}$$

(iii) **M step:** Estimate the new mean, covariance and mixing coefficients using the following equations.

Define

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{4.8}$$

Then

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{4.9}$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{new})(\mathbf{x}_n - \boldsymbol{\mu}_k^{new})^T \tag{4.10}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{4.11}$$

(iv) Evaluate the log likelihood to check for convergence. If not converged, go to step 2 and repeat the steps.

(v) Once converged, assign the data points to clusters based on higher probability value.

4.3 Clustered Pruning Algorithm

Motivated from the comparison results in Chapter 3, we implement clustered pruning, which is a feature-based pruning approach, that clusters features and removes redundancy present among them. Clustered pruning using k-means clustering is implemented in [31]. The general algorithm for pruning using the clustering approach is given in Algorithm 1. It includes three steps:

(i) Cluster the features of each layer to k clusters

(ii) Keep one node from each cluster. Eliminate the remaining nodes.

(iii) Fine-tune the pruned network

Algorithm 1 Clustered Pruning

Input: Data **X**, pre-trained model, weights **W**, biases **B**, no. of clusters k
Calculate the original layer outputs using validation data.
for $l = 2$ **to** $L - 1$ **do**
  Apply clustering on each layer output.
  From each cluster, keep just one node picked at random.
end for

4.3.1 How to select the node to be kept from each cluster

There are several criteria for selecting the node from each cluster like the following:

- **Random Selection**: If the clusters are tightly connected, each node in the cluster would be equal in terms of the network performance after pruning. This means that all the nodes in the cluster are equiprobable to be selected. The selection is then very random. This reduces additional computations involved in the selection procedure as there is no specific parameter to be chosen.

- **Closest to cluster center**: The node closest to the cluster center can be selected to be the ones that remain in the pruned network. This is used in [31].

- **Higher connecting weights norm**: Intuitively, the contributions of each node to the output is primarily defined on the magnitude of weights connecting to the node. This parameter can be defined as follows:

$$node_j = \underset{n \in C_j}{\text{argmax}} \left[\sum_{p=1}^{n_{l+1}} |w_{np}| + \sum_{q=1}^{n_{l-1}} |w_{qn}| \right] \tag{4.12}$$

  $node_j$ denotes the node to be selected from cluster C_j in hidden layer l. w_{np} is the weight value connecting node n in layer l to node p in layer $l+1$. Similarly, w_{qn} is the weight value connecting node q in layer $l-1$ to node n in layer l. |x| denotes the absolute value of x.

- **Higher bias value:** Bias of a node is highly sensitive to pruning. Selecting the node with higher bias value is a good option especially for filter pruning.

$$node_j = \underset{n \in C_j}{\text{argmax}}\ b_n \tag{4.13}$$

  where b_n denotes the bias of node n from cluster C_j of hidden layer l.

4.4 Simulation Results and Discussion

The following analyses were conducted on LeNet-300-100 and LeNet-5 networks, both trained on MNIST digit data. LeNet-300-100 had a test accuracy of 97.769%, while LeNet-5 had an accuracy of 99.218%. Both were trained using Adam optimizer with a learning rate of 0.01. Simulations were done on the Tensor Flow Machine Learning platform and Scikit-learn library.

4.4.1 Analysis on clustering algorithms

For the analysis on clustering methods on LeNet-300 and LeNet-5 architectures, nodes from each cluster were selected based on a combination of higher bias magnitude and correlation based weight update. This update is explained in the next chapter. Fig 4.1 shows the comparison between the clustering algorithms for different pruning percentages on LeNet-300 trained on MNIST data. It could be inferred that k-means and Agglomerative Ward clustering have almost the same performance for different pruning percentages.

Fig 4.2 does the same comparison, but on LeNet-5 CNN architecture, trained on MNIST data. EM clustering cannot be applied to large data sets and hence could not be used here and hence the other three methods are compared for different pruning percentage in this case. In both the plots, the X axis denotes the percentage number of non-zero parameters, which implies the percentage pruning that is performed. The plots for both the networks show that k-Means and Agglomerative Clustering methods are better than the other techniques in clustering features in Neural Networks.

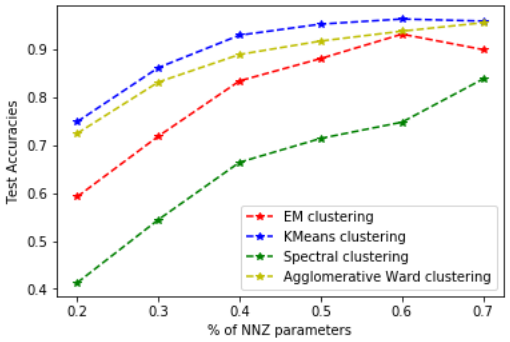

Figure 4.1: Comparison between different clustering techniques on LeNet-300.

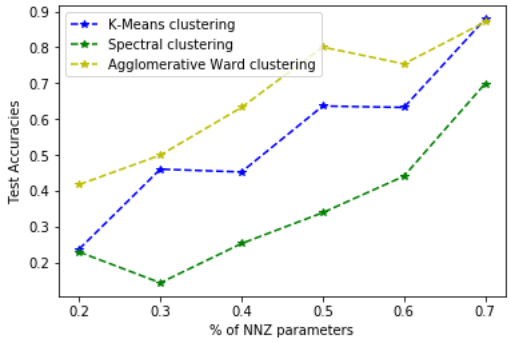

Figure 4.2: Comparison between different clustering techniques on LeNet-5.

4.4.2 Analysis on which node to select

This analysis aims at finding the best practice to select the node or filter from each cluster. Nodes were selected from the cluster using all the four methods given in Section 4.3.1 on LeNet-300. Based on the result obtained in Section 4.4.1 for LeNet-300, the base clustering algorithm was chosen to be k-means clustering algorithm. The results are plotted for different percentage of nonzero values in the parameter matrices and is shown in Fig. 4.3. Note that the percentage of nonzero values refer to the percentage of nonzero values left after pruning. Selecting the node with higher weight magnitude turns out to be better than cluster centers, in terms of the reduction in accuracy, which was used in [31].

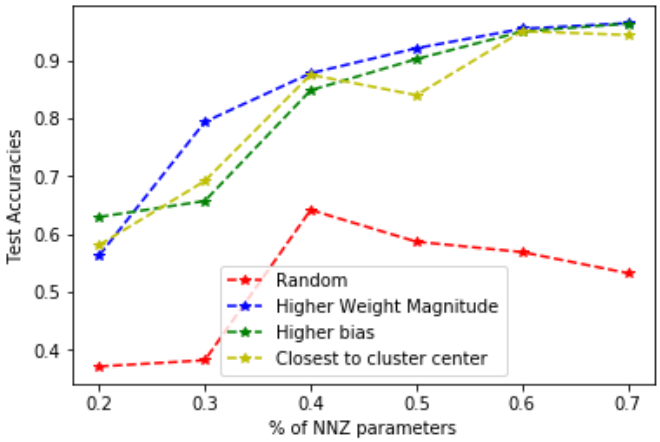

Figure 4.3: Comparison of different methods to select a node from the cluster on LeNet-300 over different percentage of pruning.

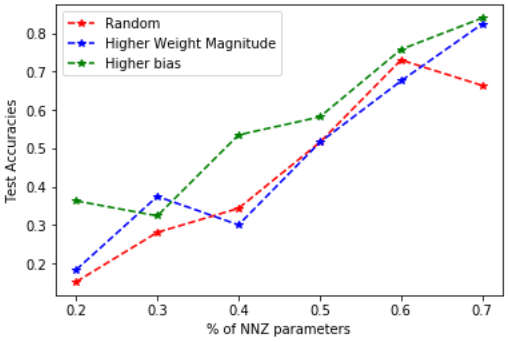

Figure 4.4: Clustered pruning using Agglomerative ward clustering and selecting the node using different criteria, for different pruning percentages on LeNet-5.

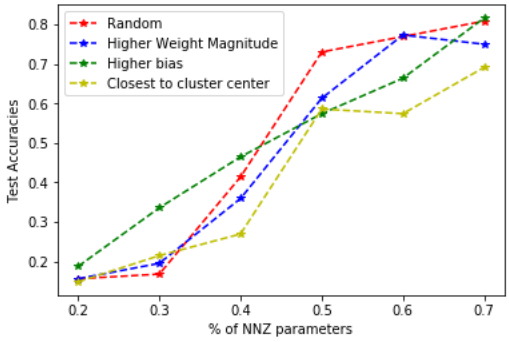

Figure 4.5: Clustered pruning using k-Means clustering and selecting the node using different criteria, for different pruning percentages on LeNet-5.

From Fig.4.2, it can be inferred that agglomerative clustering is better than k-means for LeNet-5. And hence, the comparative study on LeNet-5 to find the best method to select the node from the cluster has been done for both agglomerative and k-means clustering algorithms and the results are given in figures 4.4 and 4.5. It could be seen that on LeNet-5 CNN architecture, when using agglomerative clustering, selecting the node and filter with higher weight magnitude gives comparatively lower performance degradation (see Fig 4.4.). While, selecting the node and filter with higher bias works best when the clustering algorithm used is k-Means.

4.5 Summary

This chapter looked into the different clustering algorithms that can help in Clustered Pruning approach. Different methods can be applied to select the node or filter from each cluster and rigorous study had been done to select the best approach. k-means algorithm works well for MLP architectures while both agglomerative and k-means work better in CNN pruning. From the results obtained, it can be concluded that both k-means and agglomerative clustering techniques are equally good for clustered pruning in DNNs. Comparing the computational complexity involoved with these two approaches, k-means clustering is less computationally expensive, as agglomerative clustering (being a branch of hierarchical clustering technique) has a computational complexity of $O(N^3)$. Combining the best of both worlds, the idea of agglomerative clustering can still be used along with k-means, which gives way to the proposed pruning method, 'Subclustering based Neurogenetic Pruning', that is explained in the next chapter.

Chapter 5

Sub-clustering based Neurogenetic Pruning

5.1 Introduction

Pruning of the nodes in a neural network is equivalent to damaging the network. Node pruning reduces the number of features considered for classification. In chapter 4, we explained the use of k-means clustering to extract filters with similar features in a CNN, for a known k. However, the number of nodes that provide unique features is unknown to the designer and hence the amount of acceptable pruning is to be determined by a trial and error method. The value of 'k' must be such that the network is not severely damaged, with a large amount of performance degradation. If the value of 'k' is too small, the network might overfit.

The approach presented in this chapter, starts with smaller number of nodes in each layer. Based on a growth factor, the network grows in size, and finds the optimum number of nodes that keeps the unique features in each layer. Even when the starting 'k' is small, the network grows to an optimal architecture that doesn't overfit. Retraining this pruned model for a few epochs, restores the network to its original performance level, without overfitting.

5.2 Neurogenesis and the Sub-clustering algorithm

Neurogenesis is the process of the creation of neurons in the brain. It is most crucial during the development of an embryo, but also continues in certain brain regions after birth and even throughout the lifespan. It is of much importance during recovery from stroke. Experimental evidences exist that proves the generation of new neurons in the brain as a

part of recovery from stroke, even in regions where neurogenesis normally doesn't occur. Stroke is caused as a result of blockage in the cerebral artery. This reduces blood flow and causes focal ischemia, which induces the growth of new neurons. Ischemia-induced neurogenesis is triggered both in areas where new neurons are normally formed, such as the dentate gyrus, and in areas that are non-neurogenetic in the intact brain (e.g., the striatum). [61] summarizes the current status of research on neurogenesis after stroke.

k-means clustering algorithm is the fastest clustering algorithm currently present, useful even for larger data sets. These advantages make k-means to be a good option to cluster features, but the value of k is unknown. Clusters must be formed such that, they have high inter-cluster variance and low intra-cluster variance. k-means subclustering algorithm is a combination of k-means and agglomerative clustering algorithm. k-means clustering increases the inter-class variance, while having it in an iterative fashion (based on some criterion), reduces the intra-cluster variance, making the cluster tight. The criterion used here is the intra-cluster covariance matrix.

Similar to the generation of neurons in a stroke affected brain, new neurons are created from old by performing continuous clustering of initial clusters. The generation of neurons happen based on a growth factor[62]. Similar to this, continuous clustering happens only when the intra-cluster variance is higher than a threshold. This continuous clustering of clusters, by dividing them into two, is referred to as 'Sub-clustering'. The Sub-clustering algorithm runs in two steps:

(a) Cluster the features to very low 'k' clusters

(b) For each cluster

  (i) Calculate the intra-cluster covariance matrix

  (ii) Calculate the variance threshold using HoH algorithm (refer Section 5.3.3) for the initial cluster

  (iii) Subcluster till the intra-cluster covariance is less than the variance threshold

HoH or Histogram of Histogram algorithm runs on the initial intra-cluster covariance matrix to set a variance threshold. As the name refers, the algorithm takes the histogram of the histogram of the covariance matrix. The algorithm is explained more in the next section.

5.3 Algorithm

Algorithm 2 Sub-clustering based Neurogenetic Pruning

Input: Pretrained weights and biases
Output: Pruned model

for $l = 2$ to $L - 1$ **do**
  Choose an initial number of nodes, nc
  Cluster the nodes to nc clusters
  Calculate the variance threshold $thresh$, using HoH algorithm
  for $i = 1$ to nc **do**
    Calculate the intra-cluster variance, var
    if $(var > thresh)$ **then**
      Divide the cluster into two
    end if
  end for
end for
Fine-tune the pruned network
return Pruned model

5.3.1 Initial Clustering

The nodes in each layer is clustered initially using k-means clustering method, where the value of k is chosen to be very small compared to the total number of nodes in the particular layer. Each feature point x is assigned to a cluster based on

$$\underset{C_i \in C}{\operatorname{argmin}} \operatorname{dist}(\mathbf{c}_i, \mathbf{x}) \tag{5.1}$$

The cluster centres are updated as

$$\mathbf{c}_i = \frac{1}{|C_i|} \sum_{\mathbf{x}_i \in C_i} \mathbf{x}_i \tag{5.2}$$

$|C_i|$ is defined as the number of feature points in cluster i. The k-means algorithm iterates between the above two steps to create proper clusters with high inter cluster variance. Each cluster C_i is defined as the set of points $\mathbf{x}$ such that

$$C_i = \{\mathbf{x} : dist(\mathbf{c}_i, \mathbf{x}) < dist(\mathbf{c}_j, \mathbf{x}), \forall i \neq j\} \tag{5.3}$$

$\mathbf{c}_i$ and $\mathbf{c}_j$ are the cluster centres of the clusters C_i and C_j respectively and $C_i, C_j \in C$.

5.3.2 Sub-clustering and Neurogenesis

Neurogenesis is the process of creation of neurons in the brain using neural stem cells. Since, pruning the nodes in a neural network is equal in concept to damaging the neurons in the brain, the idea of neurogenesis can be adopted to repair the damage. Sub-clustering is defined as the process of dividing a cluster into two. An initial k-means clustering on the nodes (with k being a small value), yields clusters that are defined by the cluster centres and the labelled nodes. Based on a growth factor, the network decides whether to expand the pruned layer or not through sub-clustering. This can be related to the recovery of a brain after a stroke through neurogenesis. The growth factor chosen in this thesis is the intra-cluster covariance.
Suppose there are m points in cluster C_i.

$$C_i = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$$

Sub-clustering of cluster i results in two clusters C_{i1} and C_{i2}, defined as:

$$C_{i1} = \{\mathbf{x} : \text{dist}(\mathbf{x}, \mathbf{c}_{i1}) < \text{dist}(\mathbf{x}, \mathbf{c}_{i2})\} \tag{5.4}$$

$$C_{i2} = \{\mathbf{x} : \text{dist}(\mathbf{x}, \mathbf{c}_{i2}) < \text{dist}(\mathbf{x}, \mathbf{c}_{i1})\} \tag{5.5}$$

where $\mathbf{c}_{i1}$ and $\mathbf{c}_{i2}$ are the two new cluster centres. The intra-cluster variance of cluster i is defined using the covariance matrix between all the feature points in cluster i. To differentiate the uniqueness in the layer, each of the resulting cluster must be tightly bound. Thus, sub-clustering is done when the covariance values are higher than the threshold set in the beginning.

5.3.3 Growth factor and HoH algorithm

As mentioned above, the growth factor used in this thesis work is the intra-cluster covariance. If the intra-cluster covariance is higher than a pre-specified threshold, then the cluster is subjected to sub-clustering. The threshold is set differently for each hidden layer, using the Histogram of Histogram or HoH algorithm. The HoH algorithm is given in Algorithm 3.

The basic intuition in setting the threshold through two histograms is explained as follows: The aim of sub-clustering is to have tight clusters. Just like in agglomerative ward clustering, which performed better in CNNs, we try to impart a similar effect through k-means clustering, so that the computational complexity reduces from $O(N^3)$. The p value contains information regarding the amount of sub-clustering that the designer aims to provide, i.e., if the value of p is higher, the variance threshold set would be higher and not much sub-clustering would happen. This means that, the designer aims at keeping the number of clusters close to the initial number of clusters. Having two histograms would help in finding a common threshold value for all the clusters uniformly. This is done to reduce the computations required in setting the variance threshold.

The two steps involved with the above mentioned algorithm are:

1. Calculation of the histogram of the covariance matrix:
   This gives a probability curve (when normalized) of the possible covariance values of the nodes grouped into one cluster. Higher covariance value implies that the nodes are eligible to be separated as another unique feature providing node. Depending on the percentage of pruning desired, find the bin that sums up to at least 'p%' of the total energy under the curve.
   Let $h(.)$ be the function that calculates the histogram and v be the covariance values in the covariance matrix **V**, i.e., $v \in \mathbf{V}$. Then the set of possible thresholds for all the clusters **t**, is defined as :

$$\mathbf{t} = \left\{ b : \int_0^b h_i(v)dv \not> p \int_{-\infty}^{\infty} h_i(v)dv \text{ for } i = 1, \ldots, nc \right\} \tag{5.6}$$

2. Calculation of the histogram of the possible threshold values on all the clusters:
   From the earlier step, for each layer, a set of possible thresholds over all the clusters in that layer is obtained. Histogram over this set is then calculated. The bin with maximum frequency is selected as the variance threshold, $thresh$. Note that as the variance threshold that is set reduces, sub-clustering increases.

$$\text{thresh} = \underset{t}{\text{argmax}}\ h(\mathbf{t}) \tag{5.7}$$

Algorithm 3 HoH function

Input: Initial clusters, output of each layer for a batch input, percentage energy p
Output: variance threshold, $thresh$

Initialise ***t*** *= []*

for $c = 1$ to nc **do**
  Calculate the covariance matrix **V**
  Find the histogram of **V**
  Find the bin b, that covers p% of the total energy under the histogram
  Append b to **t**
end for
Calculate the histogram of **t**, H
Find the bin h with highest frequency in H
thresh $\leftarrow$ h
return thresh

5.3.4 How to set the initial number of clusters?

Setting the initial number of clusters is important for pruning, because the network grows based on the initial clustering result. As a rule of thumb, for fully connected layers, the designer could start with atleast 5% of the total number of nodes in that particular layer. For convolutional filters, there should be atleast 30 - 40% of filters initially.

5.3.5 How to select the best node from the cluster?

There are different ways to select the best node from the cluster. This part is crucial in the sense that, it affects the performance of the network. The nodes with highest weight norms are selected in fully connected layer, while filters with higher bias norm is selected in convolutional layers.

5.3.6 Fine-tuning

Fine-tuning is the step where the network is made to learn once again, with the pruned parameters as the initialisation, so that, any degradation due to pruning is compensated. The fine-tuning can be done in two steps, such that the number of computations can be reduced. These are explained below:

1. Correlation-based update:
   This step is inspired from the work in [63], where in an optimization procedure is used to calculate the weight updates, based on the concept of correlation. Consider

the cluster C_i with n_i number of nodes, in layer l. Let the node that is selected from the cluster, to remain in the pruned network be s. Let all other nodes in the cluster, be denoted as u, in general. Let the corresponding features be denoted as f_s and f_u. The same set of features that were used for clustering can be used in this step. The weight and bias updates are defined as follows:
For each neuron k in layer $l+1$:

$$w_{s,k}^{(l)} = w_{s,k}^{(l)} + \alpha w_{u,k}^{(l)} \tag{5.8}$$

$$b_{k}^{(l+1)} = b_{k}^{(l+1)} + \beta b_{u}^{(l)} \tag{5.9}$$

The values of α and β can be calculated from the optimisation problem:

$$\alpha, \beta = \underset{\alpha,\beta}{\operatorname{argmin}} ||f_u^{(l)} - \alpha f_s^{(l)} - \beta||^2 \tag{5.10}$$

Closed form solutions for α and β can be found out iteratively using the following equations:

$$\alpha^{(t+1)} = \frac{f_u f_s^T - \beta^{(t)} f_s^T}{(f_s f_s^T + \epsilon)} \tag{5.11}$$

$$\beta^{(t+1)} = f_u - \alpha^{(t+1)} f_s \tag{5.12}$$

To avoid dividing by zero, a small value ϵ is added in the denominator of eqn.(5.11).

2. Back-propogation:
   Similar to the training procedure, the whole data is used for this second part of fine-tuning. Due to the correlation based update, the number of epochs required for re-training is much lower compared to not having such an update.

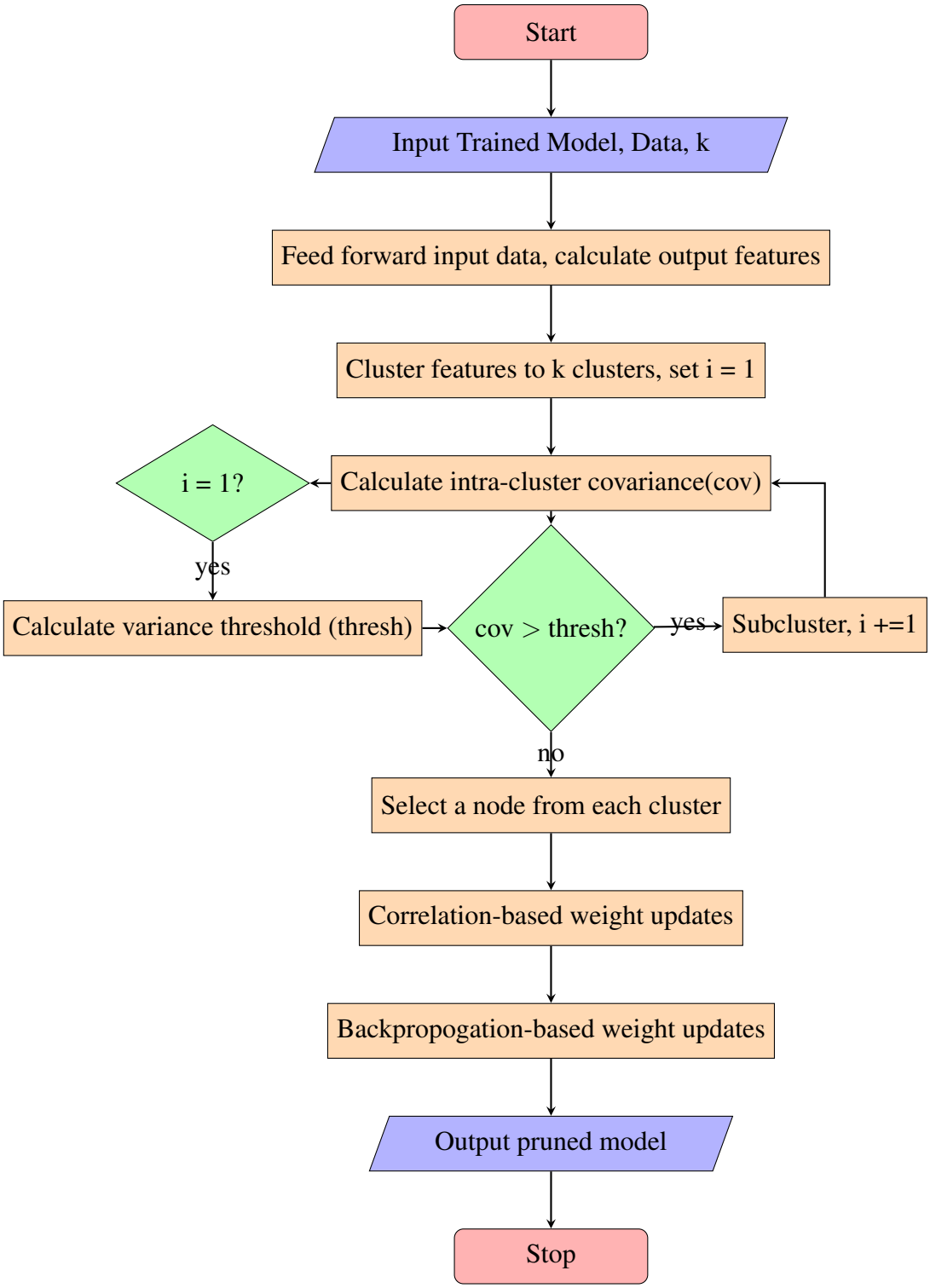

Figure 5.1: Flowchart of the proposed pruning algorithm

5.4 Computational Complexity

The initial clustering of n_l number of nodes or filters in layer l to k_l clusters (where k_l is very small), has a complexity of $O(n_l k_l)$ [53],[55]. The calculation of intra-cluster covariance matrix of cluster i with z_i number of nodes, has a complexity of $O(z_i^2)$. The t_i number of sub-clustering on cluster i has a complexity of $O(2t_i z_i)$. The HoH algorithm includes the calculation of two histograms for each cluster and hence has a time complexity of $O(z_i^2) + O(k_l)$. The selection of node from each cluster, using weight magnitude based method, has a complexity of $O(z_i d_l)$ for calculating the sum of connecting weights and $O(z_i \log z_i)$, for sorting using quick sort [64]. Here, d_l denotes the number of connections of node i in layer l (for fully connected networks, $d_l = n_{l-1} + n_{l+1}$). Similarly, selection of node from each cluster, using higher bias method will have a complexity of $O(z_i \log z_i)$. The complexity involved in a cluster i with z_i nodes is then

$$O(z_i^2) + O(2t_i z_i) + O(z_i^2) + O(k_l) + O(z_i \log z_i) + O(z_i d_l)$$

which can be approximated to $O(z_i d_l)$ as $d_l >> z_i$. The total computational complexity involved for pruning using the proposed method on each layer is given as $O(n_l k_l) + \sum_{i=1}^{k_l} O(z_i d_l)$. The overall complexity involved with pruning a MLP using the proposed method depends on the product of maximum number of nodes in a layer and its connections and is $O((\max_l n_l(n_{l-1} + n_{l+1})))$. For a CNN, the overall computational complexity is $O(\max_l n_l^2)$.

5.5 Experimental Evaluation

5.5.1 LeNet-300-100

The proposed method selected 76 nodes out of 300 in the first hidden layer and 25 nodes out of 100 in the second hidden layer of LeNet-300-100. And hence this network could be compressed by 76.803% (i.e. 431.09×) which in turn required just 4 epochs to attain the same performance as the original unpruned network (Refer Table 5.2). Retraining was done using the same learning rate and optimizer. It has been compared with some state-of-the-art methods and tabulated in Table 5.1. It can be observed from Table 5.1 that, the proposed method gives higher compression than all the state-of-the-art methods. The loss and accuracy curves for validation and test data obtained while retraining, are also given below.

Method	**Top-1 error**	$\Delta\%$	**Compression**
Wang *et al.*[65]	1.64→1.62	-0.02%	32.43×
Han *et al.*[44]	1.64→ 1.58	-0.06%	40×
Guo *et al.*[66]	2.28→1.99	-0.29%	56×
Ullrich *et al.*[67]	1.89→1.94	0.05%	64×
Aghasi *et al.*[49]	1.35→1.24	-0.11%	200×
Proposed method	2.23→2.23	0%	**431.09**×

Table 5.1: Compression results on LeNet-300-100

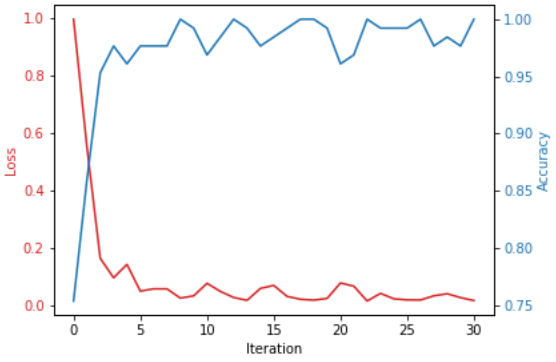

Figure 5.2: Loss and accuracy curves on validation data for different iterations while fine-tuning the pruned LeNet-300-100 network

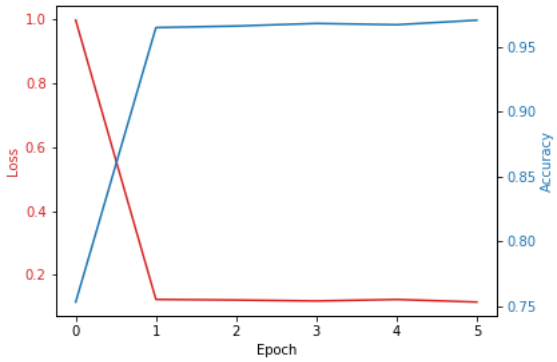

Figure 5.3: Loss and accuracy curves on test data for the epochs while fine-tuning the pruned LeNet-300-100 network

Layer	No. of nodes before Pruning	No. of nodes after Pruning	Compression obtained
Hidden Layer 1	300	76	394.73×
Hidden Layer 2	100	25	400×

Table 5.2: Compression obtained in LeNet-300-100

5.5.2 LeNet-5

The LeNet-5 [1] architecture with a test accuracy of 99.218% was pruned using the proposed method with a test accuracy of 99.218% and 70.37% compression. It required almost 25 epochs of retraining. The initial number of clusters taken was 4 filters in the first and second layers and 15 nodes in the fully connected layer. Percentage energies were set to 30% in the convolutional layers and 40% in the fully connected layer. Sub-clustering selected 4 out of 6 filters in layer 1, 5 out of 16 filters in layer 2 and 19 out of 84 nodes in the fully connected layer (Refer Table 5.4). It can be observed from Table 5.3 that, the proposed method outperforms the other state-of-the-art methods.

Method	**Top-1 error**	Δ%	**Compression**
Wang *et al.*[65]	0.88→ 0.93	0.05%	15.78×
Han *et al.*[44]	0.8→ 0.74	-0.23%	39×
Guo *et al.*[66]	0.97→0.91	-0.06%	101×
SVD[42]	0.97→0.92	-0.05%	118×
Ullrich *et al.*[67]	0.97→0.97	0%	162×
Dubey *et al.*[32]	0.97→0.96	-0.01%	193×
Aghasi *et al.*[49]	0.43→0.4	-0.03%	409.5×
Proposed method	0.78→0.78	0%	**434.22**×

Table 5.3: Compression results on LeNet-5

Layer	No. of nodes/Filters before Pruning	No. of nodes/Filters after Pruning	Compression obtained
Conv Layer 1	6	4	150×
Conv Layer 2	16	5	320×
FC layer 1	84	19	442.1 ×

Table 5.4: Compression obtained in LeNet-5

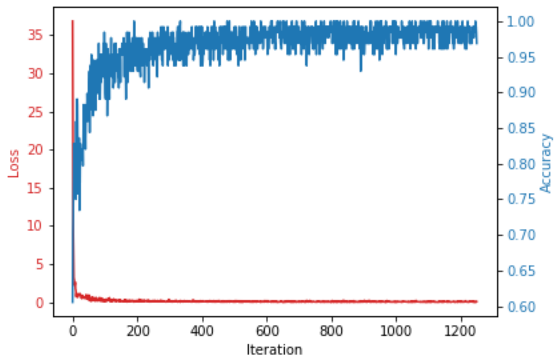

Figure 5.4: Loss and accuracy curves on validation data for different iterations while fine-tuning the pruned LeNet-5 network

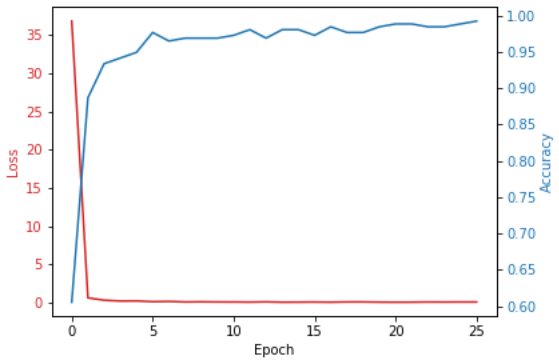

Figure 5.5: Loss and accuracy curves on test data for the epochs while fine-tuning the pruned LeNet-5 network

A network similar to LeNet-5 was trained on MNIST digit data for 200 epochs with a learning rate of 0.01. It was compressed by 90.103% with retraining for 3 epochs with the same learning rate using Adam optimizer. Initial number of clusters was set to 10, 15 out of 32, 64 filters in the convolutional layers and 50 out of 1024 nodes in the fully connected layer. Sub-clustering resulted in 17 out of 32 filters in the first convolutional layer, 19 out of 64 filters in the second layer and 98 out of 1024 nodes in the fully connected layer. Percentage energy was set to 50% for both the convolutional layers and 40% for the fully connected layer. The loss and accuracy curves while retraining for validation and test data are given in figures 5.6 and 5.7.

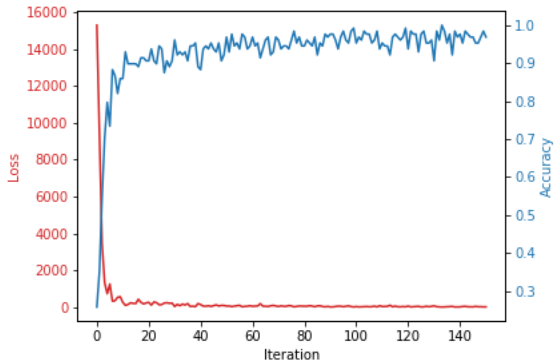

Figure 5.6: Loss and accuracy curves on validation data for different iterations while fine-tuning the pruned LeNet-5 like network

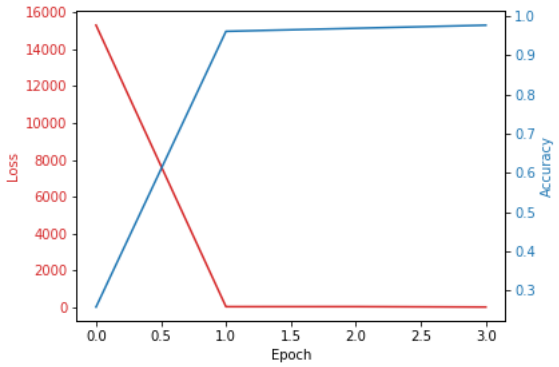

Figure 5.7: Loss and accuracy curves on test data for the epochs while fine-tuning the pruned LeNet-5 like network

It took almost 25 epochs of retraining in LeNet-5 and only 3 for LeNet-5 like architecture. A comparison between the two initial network architectures show that pruning a very large network with large number of filters and nodes, will require lower number of epochs for retraining.

5.5.3 12-layer CNN

The 12-layer CNN was trained for 10 epochs on CIFAR-10 data set and attained a test accuracy of 71.75% and validation set accuracy of 72.34%. It could be compressed by 68.137% and the compressed model required just 8 epochs for retraining, on an average. Retraining was done with the same learning rate that was used for training. It had an accuracy of 71.03% after retraining. The value of p was set as 80% for fully connected layers and 30% for convolutional layers. The layer-wise compression details are given in Table 5.5. The loss and accuracy curves while retraining for validation and test data are shown in figures 5.8 and 5.9 respectively.

Layer	Nodes/Filters before Pruning	Nodes/Filters before subclustering	Nodes/Filters after subclustering	Compression obtained
Conv Layer 1	64	16	50	128×
Conv Layer 2	128	32	45	284×
Conv Layer 3	256	64	66	387.88×
Conv Layer 4	512	128	135	379.25×
FC layer 1	128	64	88	145.45 ×
FC layer 2	256	64	86	297.67 ×
FC layer 3	512	128	128	400 ×

Table 5.5: Compression obtained in 12-layer CNN

Network	**Acc%**	**Parameters**	**Compression**
12-layer network Ref	71.75%	4.6M	1×
12-layer network Compressed	71.03%	1.46M	313.8×
VGG16 Ref [2]	67.21%	15.08M	1×
VGG16 Compressed	62.81%	3.06M	492.88×

Table 5.6: Compression obtained in deeper CNN architectures

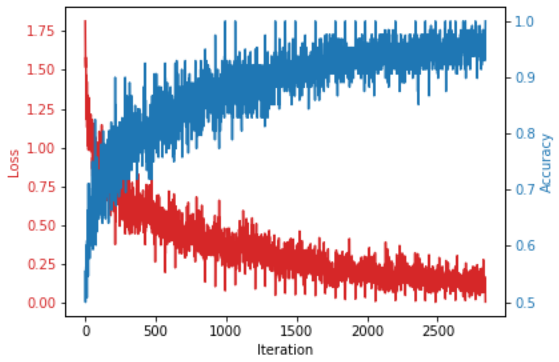

Figure 5.8: Loss and accuracy curves on validation data for different iterations while fine-tuning the pruned 12-layer CNN

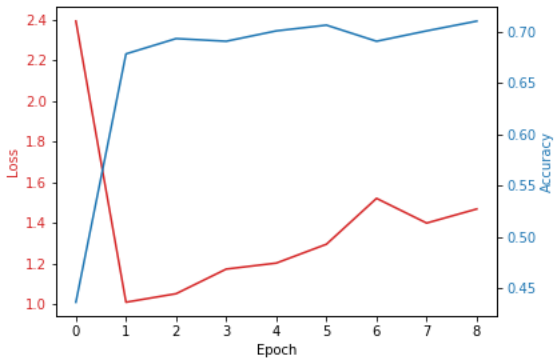

Figure 5.9: Loss and accuracy curves on test data for the epochs while fine-tuning the pruned 12-layer CNN

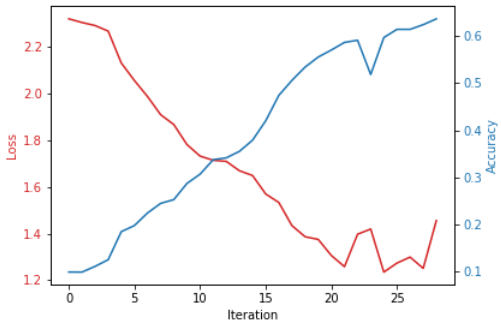

Figure 5.10: Loss and accuracy curves on validation data for different iterations while fine-tuning the pruned VGG16 CNN

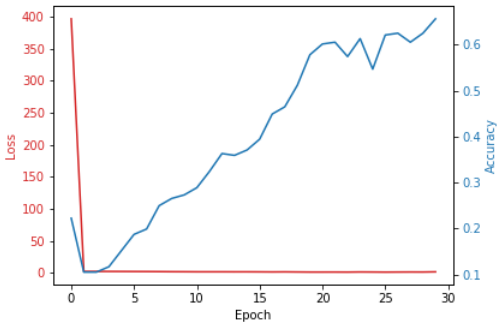

Figure 5.11: Loss and accuracy curves on test data for the epochs while fine-tuning the pruned VGG16 CNN

5.5.4 Modified VGG16 architecture

A modified VGG16 [2] architecture trained on CIFAR-10 data was pruned using the proposed method and got a compression of the order of 492.88% with an average of 25 to 50 epochs of retraining. Due to this model compression, the test accuracy had dropped from 67.208% to 62.81%. The test and validation curves while retraining is given in figures 5.10 and 5.11. The compression results of both the deep CNN architectures are tabulated in Table 5.6. Individual layer compression details are given in Table 5.7. From all the above experiments, it could be concluded that, the proposed method is applicable to deeper CNN architectures as well.

Layer	Nodes/Filters before Pruning	Nodes/Filters before subclustering	Nodes/Filters after subclustering	Compression obtained
Conv Layer 1	64	20	38	168.42×
Conv Layer 2	64	20	26	246.15×
Conv Layer 3	128	25	53	241.51×
Conv Layer 4	128	25	46	278.26×
Conv Layer 5	256	45	86	297.67×
Conv Layer 6	256	45	81	316.05×
Conv Layer 7	256	45	73	350.68×
Conv Layer 8	512	50	80	640×
Conv Layer 9	512	50	78	656.41×
Conv Layer 10	512	50	85	602.35×
Conv Layer 11	512	50	93	550.54×
Conv Layer 12	512	50	95	538.95×
Conv Layer 13	512	50	132	387.87×
FC layer 1	512	64	107	758.87 ×

Table 5.7: Compression obtained in modified VGG-16

5.5.5 Analysis of hyperparameter

The hyper parameter involved in the realization of this approach is the percentage of energy p. As p increases, the threshold value, will also increase. This results in reduced subclustering, which imposes more sparsity. Experimental results on the same performed on LeNet-300-100 is tabulated in Table 5.4. The initial number of clusters were taken to be 20 and 25 out of 300 and 100 nodes in hidden layers 1 and 2 respectively.

p%	Percentage Sparsity	Test Accuracy After Pruning	No. of epochs for Retraining
0.1	63.10%	84.32%	2
0.5	76.34%	76.07%	11
0.8	91.65%	53.08%	>25

Table 5.8: Performance analysis on LeNet-300-100 pruned using the proposed algorithm for different p% of energy.

A similar analysis was performed on the LeNet-5-like architecture with initial number of clusters as 10, 15 and 50 out of 32, 64 filters and 1024 nodes respectively. The results are tabulated below. From Table 5.2, it could be seen that, as the percentage of energy increases, the sparsity increases.

p%	Percentage Sparsity	Test Accuracy After Pruning	No. of epochs for Retraining
0.1	89.42%	42.57%	3
0.5	91.73%	36.71%	2
0.8	94.11%	31.65%	>25

Table 5.9: Performance analysis on LeNet-5 pruned using the proposed algorithm for different p% of energy.

Hyper parameter($p\%$) analysis on the modified VGG-16 model is also performed and is tabulated in Table 5.10. It could be observed that, as p increases, sub-clustering decreases and sparsity increases. Larger networks work better at smaller p (< 0.4), where more subclustering happens.

p%	Percentage Sparsity	Test Accuracy After Pruning	No. of epochs for Retraining
0.2	83.88%	16.11%	27
0.5	84.64%	15.36%	>50
0.8	84.96%	15.04%	>50

Table 5.10: Performance analysis on modified VGG-16 architecture pruned using the proposed algorithm for different p% of energy.

5.5.6 Comparison with Clustered pruning using Net Deviation parameter

LeNet-300-100 network was pruned with different initial clusters, based on percentage of pruning, using the proposed method. The pruning results are compared with Clustered Pruning, using k-means clustering using the 'Net Deviation' parameter defined in Chapter 3. The resulting plot is given in Figure 5.12. It could be seen that, for higher compression, sub-clustering based approach would be more complex than clustered pruning method, which is intuitively true. For higher compression, the number of clusters set initially would be very very small and the amount of sub-clustering required would be high, so that the intra-cluster variance is lower than the set threshold. Based on the complexity involved, clustered pruning would better, but the corresponding performance in terms of Test Accu-

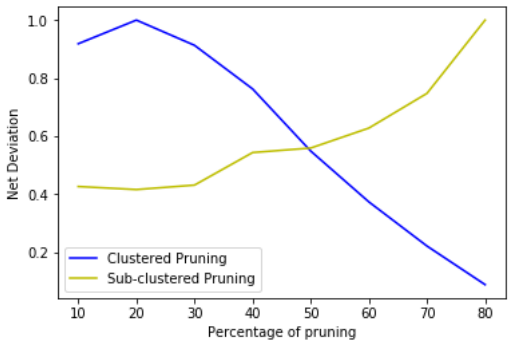

Figure 5.12: Comparison between the proposed method and clustered pruning based on Net Deviation

racy will be lower. And the network might not even regain the earlier performance, justified in [31], where the authors have mentioned that the performance would drop heavily when the number of clustering centroids become less. The inference is simple. Sub-clustering increases the number of clusters, when the initial number of clusters is small. The maximum observed number of sub-clustering is 4. And hence, the complexity is not that high, if the proposed method is used. But compared to Clustered Pruning, the proposed method would help jump to a new 'k' value, even without any prior knowledge about the data. At present, higher compression is reported using optimization techniques, and compared to these methods, the proposed method is able to achieve high compression with lower number of computations.

5.6 Summary

This chapter explained a novel biologically inspired node pruning method named as 'Sub-clustering based Neurogenetic Pruning'. Experimental results show that this method can attain high compression with comparatively low computational complexity, even for larger networks. It has been observed that, although it provides high compression, it does get stuck in local minimum sometimes, during fine-tuning, which is a common problem that is found in training neural networks. And hence, care must be taken while retraining, so that the network converges properly.

Chapter 6

Conclusion and Future Work

Model Compression is an important area research which focuses on the reduction of the NN size so that the network can provide faster inference. A popular approach of model compression is Pruning, which eliminates some of the network connections or nodes, based on certain conditions. The current pruning methods that provide high compression, are computationally very expensive. While, the methods with lower computations, cannot achieve high compression, with significantly low epochs of retraining.

This thesis has theoretically derived and experimentally validated the amount of retraining that would be required after pruning, in terms of the relative number of epochs. Also, the propagation of errors through the layers, due to pruning different layers is analysed and a bound to the amount of error that the layers contribute was derived. The parameter 'Net Deviation' can be used to study different pruning approaches and hence can be used as a criteria for ranking different pruning approaches. If not completely avoided, reducing the number of epochs linked to retraining the network will reduce the computational complexity involved with training a Neural Network. A new pruning approach inspired from the recovery of a brain from stroke, was proposed that compresses the model with less error, so that the retraining procedure is computationally less expensive.

As a future work:

- An empirical relation for the γ given in Eqn.(3.1), that will help in easy determination of the expected number of epochs for retraining the pruned model.
- Experimental results have shown that Subclustering based pruning method gives more than 70% compression, in general. Further compression can obtained by applying quantization of parameters after pruning.

- Larger networks like VGG16, ResNet, etc. can be pruned using the proposed method. And how much pruning process can help in real-time performance could be analysed by implementing them on hardware, like FPGA. The effect of pruning and quantization errors on the performance can be studied which can explain how useful model compression is, or not.

- An optimization problem can be coined using the definition of Net Deviation, which could also be used for pruning. The solution to such an optimization problem, the computational complexity involved and the performance in terms of compression and the amount of retraining required, could be another future work.

- A real-time application of pruning could be tried out even in Object Tracking using Deep Learning. Subclustered based Neurogenetic Pruning removes the redundancy present among the features. Since object tracking requires only the features, will pruning based on the proposed method be useful in faster tracking?

Bibliography

[1] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*. IEEE, 2015, pp. 730–734.

[3] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.

[4] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, p. 48, 2001.

[5] R. O. Duda and P. E. Hart, "Pattern recognition and scene analysis," 1973.

[6] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[7] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

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

[10] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[13] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[14] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[15] G. Chechik, I. Meilijson, and E. Ruppin, "Synaptic pruning in development: a computational account," *Neural computation*, vol. 10, no. 7, pp. 1759–1777, 1998.

[16] F. I. Craik and E. Bialystok, "Cognition through the lifespan: mechanisms of change," *Trends in cognitive sciences*, vol. 10, no. 3, pp. 131–138, 2006.

[17] A. Bondarenko, A. Borisov, and L. Aleksejeva, "Neurons vs weights pruning in artificial neural networks," in *Proceedings of the 10th International Scientific and Practical Conference. Volume III*, vol. 22, 2015, p. 28.

[18] M. Hagiwara, "Removal of hidden units and weights for back propagation networks," in *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, vol. 1. IEEE, 1993, pp. 351–354.

[19] M. G. Augasta and T. Kathirvalavakumar, "A novel pruning algorithm for optimizing feedforward neural network of classification problems," *Neural processing letters*, vol. 34, no. 3, p. 241, 2011.

[20] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE transactions on Neural networks*, vol. 8, no. 3, pp. 519–531, 1997.

[21] J. D. 1. Y. Cun and S. Solla, "Optimal brain damage," *Advances in Neural Information Processing Systems, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann*, vol. 2, pp. 598–605, 1990.

[22] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*. IEEE, 1993, pp. 293–299.

[23] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 2017, pp. 4857–4867.

[24] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE transactions on Neural Networks*, vol. 12, no. 6, pp. 1386–1399, 2001.

[25] D. S. Yeung and X.-Q. Zeng, "Hidden neuron pruning for multilayer perceptrons using a sensitivity measure," in *Proceedings. International Conference on Machine Learning and Cybernetics*, vol. 4. IEEE, 2002, pp. 1751–1757.

[26] P. Lauret, E. Fock, and T. A. Mara, "A node pruning algorithm based on a fourier amplitude sensitivity test method," *IEEE transactions on neural networks*, vol. 17, no. 2, pp. 273–293, 2006.

[27] H.-J. Xing and B.-G. Hu, "Two-phase construction of multilayer perceptrons using information theory," *IEEE Transactions on Neural Networks*, vol. 20, no. 4, pp. 715–721, 2009.

[28] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[29] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866*, 2018.

[30] Z. Liu, J. Xu, X. Peng, and R. Xiong, "Frequency-domain dynamic pruning for convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 1043–1053.

[31] L. Li, Y. Xu, and J. Zhu, "Filter level pruning based on similar feature extraction for convolutional neural networks," *IEICE TRANSACTIONS on Information and Systems*, vol. 101, no. 4, pp. 1203–1206, 2018.

[32] A. Dubey, M. Chatterjee, and N. Ahuja, "Coreset-based neural network compression," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 454–470.

[33] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.

[34] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[35] M. Babaeizadeh, P. Smaragdis, and R. H. Campbell, "Noiseout: A simple way to prune neural networks," *arXiv preprint arXiv:1611.06211*, 2016.

[36] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[37] L. N. Huynh, Y. Lee, and R. K. Balan, "D-pruner: Filter-based pruning method for deep convolutional neural network," in *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning*. ACM, 2018, pp. 7–12.

[38] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[39] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Advances in Neural Information Processing Systems*, 2017, pp. 2181–2191.

[40] Q. Huang, K. Zhou, S. You, and U. Neumann, "Learning to prune filters in convolutional neural networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 709–718.

[41] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang, "Deep fried convnets," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1476–1483.

[42] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, 2014, pp. 1269–1277.

[43] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[44] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[45] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7873–7882.

[46] M. A. Carreira-Perpinán, "Model compression as constrained optimization, with application to neural nets. part i: General framework," *arXiv preprint arXiv:1707.01209*, 2017.

[47] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, "Net-trim: Convex pruning of deep neural networks with performance guarantee," in *Advances in Neural Information Processing Systems*, 2017, pp. 3177–3186.

[48] M. A. Carreira-Perpinán and Y. Idelbayev, "âĂIJlearning-compressionâĂİ algorithms for neural net pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8532–8541.

[49] A. Aghasi, A. Abdi, and J. Romberg, "Fast convex pruning of deep neural networks," *arXiv preprint arXiv:1806.06457*, 2018.

[50] M. G. Augasta and T. Kathirvalavakumar, "Pruning algorithms of neural networksâĂŤa comparative study," *Central European Journal of Computer Science*, vol. 3, no. 3, pp. 105–115, 2013.

[51] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[52] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[53] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[54] S. Lloyd, "Least square quantization in pcm. bell telephone laboratories paper. published in journal much later: Lloyd, sp: Least squares quantization in pcm," *IEEE Trans. Inform. Theor.(1957/1982)*, vol. 18, 1957.

[55] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[56] M. K. Pakhira, "A linear time-complexity k-means algorithm using cluster shifting," in *2014 International Conference on Computational Intelligence and Communication Networks*. IEEE, 2014, pp. 1047–1051.

[57] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[58] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[59] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[60] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[61] O. Lindvall and Z. Kokaia, "Neurogenesis following stroke affecting the adult brain," *Cold Spring Harbor perspectives in biology*, vol. 7, no. 11, p. a019034, 2015.

[62] B. Kolb, C. Morshead, C. Gonzalez, M. Kim, C. Gregg, T. Shingo, and S. Weiss, "Growth factor-stimulated generation of new cortical tissue and functional recovery after stroke damage to the motor cortex of rats," *Journal of cerebral blood flow & metabolism*, vol. 27, no. 5, pp. 983–997, 2007.

[63] M. Babaeizadeh, P. Smaragdis, and R. H. Campbell, "A simple yet effective method to prune dense layers of neural networks," 2016.

[64] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[65] S. Wang, H. Cai, J. Bilmes, and W. Noble, "Training compressed fully-connected networks with a density-diversity penalty," 2016.

[66] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, 2016, pp. 1379–1387.

[67] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," *arXiv preprint arXiv:1702.04008*, 2017.

[68] B. T. Polyak, "Gradient methods for the minimisation of functionals," *USSR Computational Mathematics and Mathematical Physics*, vol. 3, no. 4, pp. 864–878, 1963.

[69] J. R. Taylor, "Error analysis," *Univ. Science Books, Sausalito, California*, 1997.

List of Publications

Conferences

1. Soumya Sara John, Deepak Mishra and Sheeba Rani J., "Retraining conditions: How much to retrain a network after pruning?' in 8th International Conference on Pattern Recognition and Machine Intelligence (PReMI), 2019 $[Submitted]$

2. Soumya Sara John, Deepak Mishra and Sheeba Rani J., "Sub-clustering based Neurogenetic Pruning" in Thirty-third Conference on Neural Information Processing Systems (NeurIPS), 2019 $[Submitted]$

Appendix A

Theoretical Bounds on the number of epochs for fine-tuning a Sparse Neural Network

Assume that the loss function is continuously differentiable and strictly convex. By the strong convexity assumption, the Polyak-Lojasiewicz inequality [68] can be implied, which is stated as follows:
For a continuously differentiable and strongly convex function f, over parameter x,

$$\frac{1}{2}||\nabla f(x)||^2 \geq \mu(f(x) - f(x^*)), \forall x \tag{A.1}$$

where x^* is the minimum point.
Using the above relation on the loss function $L(W)$, eq.(4.1) can be rewritten for two cases:

1. Initial training of the network, till convergence

$$\frac{1}{2}||\nabla L(W_{initial})||^2 \geq \mu_1(L(W_{initial}) - L(W^*)) \tag{A.2}$$

2. Fine-tuning the sparse network, to reach back to the original performance

$$\frac{1}{2}||\nabla L(W_{sparse})||^2 \geq \mu_2(L(W_{sparse}) - L(W^*)) \tag{A.3}$$

The number of epochs to reach convergence is directly proportional to the difference in losses and can be written as:

$$t_{initial} \propto ||L(W_{initial} - L(W^*)|| \tag{A.4}$$

$$t_{sparse} \propto ||L(W_{sparse} - L(W^*)|| \tag{A.5}$$

Combining eqns (A.2 - A.5),

$$t_{sparse} \geq \frac{\gamma\mu_1}{\mu_2}\left[\frac{||\nabla L(W_{initial})||^2}{||\nabla L(W_{sparse})||^2}\right] t_{initial} \tag{A.6}$$

where γ is introduced so as to accommodate the proportionality given in eqns.(A.4) and (A.5).

Appendix B

Theoretical Analysis of the Error Propagation in Sparse Neural Network

B.1 Proof of Theorem 2

From [69], for a function with more than one variable $q(x, y)$, when the uncertainties in x and y are independent and random, the uncertainty in q can be written as

$$\delta q^2 = \left[\frac{\partial q}{\partial x}\delta x\right]^2 + \left[\frac{\partial q}{\partial y}\delta y\right]^2 \tag{B.1}$$

Applying triangular inequality, the following equation is valid.

$$\delta q \leq |\frac{\partial q}{\partial x}|\delta x + |\frac{\partial q}{\partial y}|\delta y \tag{B.2}$$

In the concept of pruning of neural networks, the weight changes and output changes in earlier layers are independent to each other. The output of the neural network is

$$\mathbf{Y}^{(L)} = f(\mathbf{W}^{(L-1)^T}\mathbf{Y}^{(L-1)} + \mathbf{B}^{(L)}) \tag{B.3}$$

The function $f(.)$ can be sigmoid or relu or softmax function as per the layer considered. Softmax function is used in the output layer usually. Assuming no change in the bias of the layers,

$$||\delta\mathbf{Y}^{(L)}||^2 = ||\frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{W}^{(L-1)}}\delta\mathbf{W}^{(L-1)}||^2 + ||\frac{\partial\mathbf{Y}^{(L)}}{\partial\mathbf{Y}^{(L-1)}}\delta\mathbf{Y}^{(L-1)}||^2 \tag{B.4}$$

and

$$||\delta \mathbf{Y}^{(L)}|| \leq ||\frac{\partial \mathbf{Y}^{(L)}}{\partial \mathbf{W}^{(L-1)}} \delta \mathbf{W}^{(L-1)}|| + ||\frac{\partial \mathbf{Y}^{(L)}}{\partial \mathbf{Y}^{(L-1)}} \delta \mathbf{Y}^{(L-1)}|| \tag{B.5}$$

Similarly for other layers,

$$||\delta \mathbf{Y}^{(L-1)}|| \leq ||\frac{\partial \mathbf{Y}^{(L-1)}}{\partial \mathbf{W}^{(L-2)}} \delta \mathbf{W}^{(L-2)}|| + ||\frac{\partial \mathbf{Y}^{(L-1)}}{\partial \mathbf{Y}^{(L-2)}} \delta \mathbf{Y}^{(L-2)}||$$

Accumulating the effect of weight changes in all the layers and assuming no change in the input layer, we get

$$||\delta \mathbf{Y}^{(L)}|| \leq ||\frac{\partial \mathbf{Y}^{(L)}}{\partial \mathbf{W}^{(L-1)}} \delta \mathbf{W}^{(L-1)}|| + ||\frac{\partial \mathbf{Y}^{(L)}}{\partial \mathbf{Y}^{(L-1)}} \frac{\partial \mathbf{Y}^{(L-1)}}{\partial \mathbf{W}^{(L-2)}} \delta \mathbf{W}^{(L-2)}|| + ... + \left[\prod_{i=3}^{L} ||\frac{\partial \mathbf{Y}^{(i)}}{\partial \mathbf{Y}^{(i-1)}}|| \right] ||\frac{\partial \mathbf{Y}^{2}}{\partial \mathbf{W}^{1}} \delta \mathbf{W}^{1}||$$

This is given in Theorem 2.

$$||\delta \mathbf{Y}^{(L)}|| \leq \sum_{l=2}^{L} \left[\prod_{\substack{i=l+1 \\ (l \neq L)}}^{L} ||\frac{\partial \mathbf{Y}^{(i)}}{\partial \mathbf{Y}^{(i-1)}}|| \right] ||\frac{\partial \mathbf{Y}^{(l)}}{\partial \mathbf{W}^{(l-1)}}|| ||\delta \mathbf{W}^{(l-1)}|| \tag{B.6}$$

Hence the proof.

B.2 Proof of Eqn. (3.10)

Suppose the output error is bounded by ϵ, which corresponds to the LHS in eqn.(B.6). Expanding the RHS,

$$\epsilon_1 = 0$$

$$\epsilon_2 = \left[\prod_{i=3}^{L} ||\frac{\partial \mathbf{Y}^{(i)}}{\partial \mathbf{Y}^{(i-1)}}|| \right] ||\frac{\partial \mathbf{Y}^{2}}{\partial \mathbf{W}^{1}} \delta \mathbf{W}^{1}||$$

$$\epsilon_3 = \left[\prod_{i=4}^{L} ||\frac{\partial \mathbf{Y}^{(i)}}{\partial \mathbf{Y}^{(i-1)}}|| \right] ||\frac{\partial \mathbf{Y}^{3}}{\partial \mathbf{W}^{2}} \delta \mathbf{W}^{2}||$$

$$\epsilon_L = ||\frac{\partial \mathbf{Y}^{(L)}}{\partial \mathbf{W}^{(L-1)}} \delta \mathbf{W}^{(L-1)}||$$

Combining all the above,

$$\epsilon \leq \epsilon_1 + \epsilon_2 + ... + \epsilon_L \tag{B.7}$$

An alternative expression can be found in a similar pattern of derivation starting with eqn.(B.4), which is given in [47] and [49] as

$$\epsilon^2 = \epsilon_1^2 + \epsilon_2^2 + ... + \epsilon_L^2 \tag{B.8}$$

B.3 Proof of Eqn. (3.11)

The proof of eqn. (3.11) is quite obvious. Assuming that the error in each layer is approximately zero,

$$\begin{aligned}
\epsilon_L &= 0 \\
||\frac{\partial \mathbf{Y}^{(L)}}{\partial \mathbf{W}^{(L-1)}} \delta \mathbf{W}^{(L-1)}|| &= 0 \\
||\mathbf{Y}^{(L)} \odot (\mathbf{1} - \mathbf{Y}^{(L)})||||\delta \mathbf{W}^{(L-1)^T} \mathbf{Y}^{(L-1)}|| &= 0 \\
||\delta \mathbf{W}^{(L-1)^T} \mathbf{Y}^{(L-1)}|| &= 0 \\
\delta \mathbf{W}^{(L-1)^T} \mathbf{Y}^{(L-1)} &= \mathbf{0}
\end{aligned}$$

$\odot$ refers to the Hadamard product of matrices. Similar results can be obtained for other layers as well. This proves eqn.(3.11).