2019

Using Multi Population Cultural Algorithms to prune Deep Neural Networks

Anish Saurabh Desai
University of Windsor

Using Multi Population Cultural Algorithms to prune Deep Neural Networks

By

Anish Saurabh Desai

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2019

Using Multi Population Cultural Algorithms to prune Deep Neural Networks

Anish Saurabh Desai

APPROVED BY:

_____

R. J. Urbanic

Department of Mechanical, Automotive and Materials Engineering

_____

J. Lu

School of Computer Science

_____

Z. Kobti, Advisor

School of Computer Science

April 16, 2019

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION

1. Co-Authorship

I hereby declare that this thesis incorporates material that is the result of research conducted under the supervision of Dr. Ziad Kobti (Advisor). In all cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author, and the contribution of co-authors was primarily through the proofreading of the published manuscripts.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

2. Previous Publication

This thesis includes one original paper that has been previously submitted for publication in peer reviewed journals, as follows:

Section	Publication title/full citation	Publication status
1.2, 3.3, 4.5, 4.6, 4.7, 6.1, 6.2 and 6.3	Anish Desai and Ziad Kobti. "Using Multi-Population Cultural Algorithms to prune Deep Neural Networks. In *Proceedings of the Genetic and Evolutionary Computation Conference 2019 (GECCO'19).*	Submitted

I certify that the above material describes work completed during my registration as a graduate student at the University of Windsor.

3. General

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

The success of Deep Neural Networks (DNN) in classification is accompanied by a drastic increase in weight parameters which also increases the computational and storage costs. Pruning of DNN involves identifying and removing redundant parameters with little or no loss of accuracy. Layer-wise pruning of weights by their magnitude has shown to be an efficient method to prune neural networks. However, finding the optimal values of the threshold for each layer is a challenging task given the large search space. To solve this problem, we use multi population cultural algorithm which is an evolutionary algorithm that takes advantage of knowledge domains and faster convergence and is used in many optimization problems. We experiment it on LeNet-style models and measure the level of pruning through the pruning ratio. Results show that our method achieves the best pruning ratio (864 on LeNet5) compared with some state-of-the-art DNN pruning methods. By removing redundant parameters, the computational and storage costs are reduced significantly.

DEDICATION

I would like to dedicate this thesis to my parents.

Father: Saurabh Desai

Mother: Dipti Desai

ACKNOWLEDGEMENTS

There are many people whom I would like to thank for the successful completion of this thesis. First and foremost, I would like to thank my parents, whose endless support kept me motivated throughout my research journey.

I would also like to express my gratitude to my advisor, Dr. Kobti, who helped me in accomplishing my goals and has provided valuable guidance improving my research skills and knowledge. I highly appreciate the amount of time he dedicated for me, the knowledge he imparted to me and the funding he provided to me. Without his support, this thesis wouldn't be complete.

I would also thank my committee members – Dr. Lu and Dr. Urbanic – whose inputs and feedbacks have given a better shape to my research. I also admire the support of Mrs. Melissa and Mrs. Gloria for helping me in various academic issues.

Last, but not the least, I would like to thank God for giving me this ability and opportunity to undertake this research and complete it satisfactorily.

TABLE OF CONTENTS

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION..... iii

ABSTRACT..... vi

DEDICATION..... vii

ACKNOWLEDGEMENTS viii

LIST OF TABLES xii

LIST OF FIGURES xiii

Introduction.....1

1.1 Background 1

1.2 Problem Definition.....7

1.3 Thesis Motivation.....8

1.4 Thesis Statement 10

1.5 Thesis Contribution.....11

1.6 Thesis Organization.....12

Literature Review 14

2.1 Neuron-based pruning.....14

2.2 Weight-based pruning 16

2.3 Evolutionary Pruning 21

Evolutionary Computation 25

3.1 Evolutionary Algorithms 25
3.2 Genetic Algorithms 27
3.2.1 Crossover Operation 28
3.2.2 Mutation Operation 29
3.3 Cultural Algorithms 30
3.3.1 Belief Space 31
3.3.2 Multi-Population Cultural Algorithms 32

Proposed Approach 35

4.1 Proposed Strategies to prune DNN 35
4.2 Evolutionary Pruning of DNN 36
4.3 Individual Representation 37
4.4 Fitness Evaluation 37
4.5 Pruning Neurons 38
4.6 Adjusting Cultures 39
4.6.1 Situational Component 39
4.6.2 Spatial Component 40
4.6.3 Normative Component 41
4.7 Influence Functions 42
4.8 Using CA to prune DNN 45
4.9 Using MPCA to prune DNN 47

Experiments and Results 50

5.1 Dataset 50
5.2 Models 51
5.3 Setting Hyperparameters 51
5.3 Using CA to prune LeNet300-100 52

5.4 Using MPCA to prune LeNet300-100 53
5.5 Using CA to prune LeNet5.....55
5.6 Using MPCA to prune LeNet5.....56
5.7 Error Analysis 58

Comparison, Analysis and Discussion.....59

6.1 Comparison between CA and MPCA 59
6.2 Comparisons with the LeNet300-100 model 61
6.3 Comparisons with the LeNet5 model.....63

Conclusion and Future Work 67

7.1 Limitations 67
7.2 Future Work 68

REFERENCES.....69

APPENDIX.....74

VITA AUCTORIS 75

LIST OF TABLES

Table 1 - Development of synapses in human brain 8
Table 2 - Comparison of various methods to prune neural networks 21
Table 3 - Recent use of evolutionary techniques to prune DNN 22
Table 4 - Comparison of PR achieved by different methods for pruning LeNet300-100 61
Table 5 - Pruning details by each layer of the LeNet300-100 network 62
Table 6 - Comparison of PR achieved by different methods for pruning LeNet5 63
Table 7 - Pruning details by each layer of the LeNet5 network 65

LIST OF FIGURES

Figure 1 - Architecture of a Feedforward Neural Network [1] 2
Figure 2 - Architecture of a CNN 3
Figure 3 – Kernel computations in a CNN 4
Figure 4 - Max-Pooling 5
Figure 5 - Space reduction due to a sparse matrix of a pruned network. 9
Figure 6 - Since the first and third neuron of the hidden layer do not have any input and output weights respectively, it can be pruned. 17
Figure 7 - Pruning Feature Map and its associated filters [14] 17
Figure 8 - Weights and neurons before and after pruning [7] 19
Figure 9 - The OLMP method of pruning network [10] 23
Figure 10 - Relation between Genomes and Phenomes [24] 27
Figure 11 - Types of crossover operations 28
Figure 12 - Types of mutations 29
Figure 13 - Components of cultural algorithm [30] 31
Figure 14 - Architecture of a variant of CA having multiple population space but a single belief space [32] 32
Figure 15 - Architecture of a variant of CA having multiple population and belief spaces [33] 33
Figure 16 - Representation of an individual. 37
Figure 17 - Example of the crossover operation. 44
Figure 18 - Example of the mutation operation 45
Figure 19 - Example of knowledge migration in MPCA 48
Figure 20 - Images in MNIST dataset that are difficult even for humans to recognize ... 51
Figure 21 - Graph showing the best cost achieved at each pruning epoch for pruning LeNet300-100 by CA 52
Figure 22 - Improvement in accuracy after finetuning. 53

Figure 23 - Graph showing the best cost achieved at each pruning epoch for pruning LeNet300-100 by MPCA 54
Figure 24 - Finetuning LeNet300-100 after pruning with MPCA 54
Figure 25 - Graph showing the best cost achieved at each pruning epoch for pruning LeNet5 by CA 55
Figure 26 - Finetuning LeNet5 after pruning with CA 56
Figure 27 - Graph showing the best cost achieved at each pruning epoch for pruning LeNet5 by MPCA 57
Figure 28 - Finetuning LeNet5 after pruning with MPCA 57
Figure 29 - Comparison of CA and MPCA for pruning LeNet300-100 60
Figure 30 - Comparison of CA and MPCA for pruning LeNet5 60

Chapter 1

Introduction

Classification is the process of classifying data according to shared qualities or characteristics. It is the problem of finding a function by training over a given set of instances whose category membership is known, also known as training data, to classify or predict new instances. Although there exist many models for classification problems, Deep Neural Networks have emerged to be the most promising of them all.

1.1 Background

Neural Networks are an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information [1]. Various architectures of neural networks have successfully achieved high accuracy rates in many classification problems. A simple Feedforward Neural Network (FNN) consists of an input layer, one or more hidden layers and an output layer as shown in Figure 1 [2]. Every hidden layer contains one or more neurons. Every neuron of a layer is connected to all the neurons of the next layer through weight parameters. In addition, every layer except the output layer contain one bias parameter for each neuron of the next layer. Each neuron calculates its

activation value by computing the sum of the bias parameter with the product of the weight parameters with their corresponding activation values of the input neurons. Mathematically,

$$a_j^l = \sigma\left(\sum_k w_{jk}^{l-1} a_k^{l-1} + b_j^{l-1}\right) \quad (1)$$

Where a_j^l is the activation value of the j^{th} neuron of the l^{th} layer, b_j^{l-1} is the bias parameter of the $l-1^{th}$ layer connected to the j^{th} neuron of the l^{th} layer, w_{jk}^{l-1} is the weight parameter connecting the k^{th} neuron of the $l-1^{th}$ layer to the j^{th} neuron of the l^{th} layer and σ (z) is known as an activation function. One of the commonly used activation functions is the sigmoid function which is formulated as,

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad (2)$$

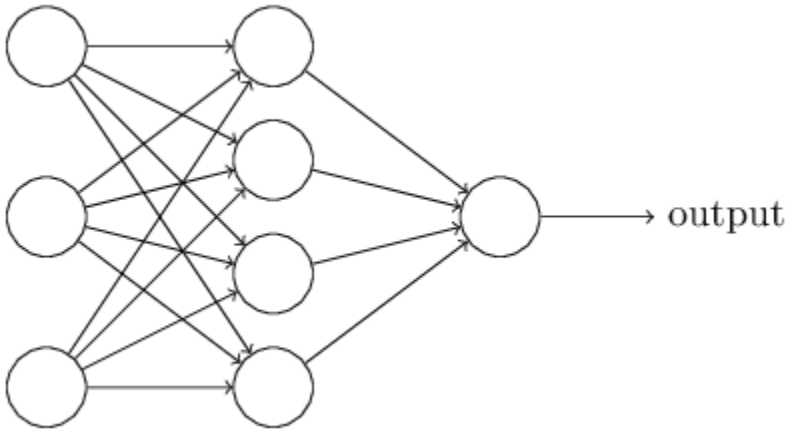

Figure 1 - Architecture of a Feedforward Neural Network [1]

The hidden layers of FNN are known as dense layers since every neuron of one layer is connected to every other neuron of the next layer. It has been shown that FNN are universal approximators, i.e., FNN can in principle approximate any measurable function to any desired accuracy, if the network contains enough "hidden" neurons between the input and output layers [3]. There exist models of Neural Networks with different architecture which have shown to be more efficient when dealing with complex data. One of the models is Convolutional Neural Network (CNN) which are used extensively to classify images [4]. CNN consists of one input layer, one or more convolutional layer followed by one or more dense layer and an output layer as shown in Figure 2.

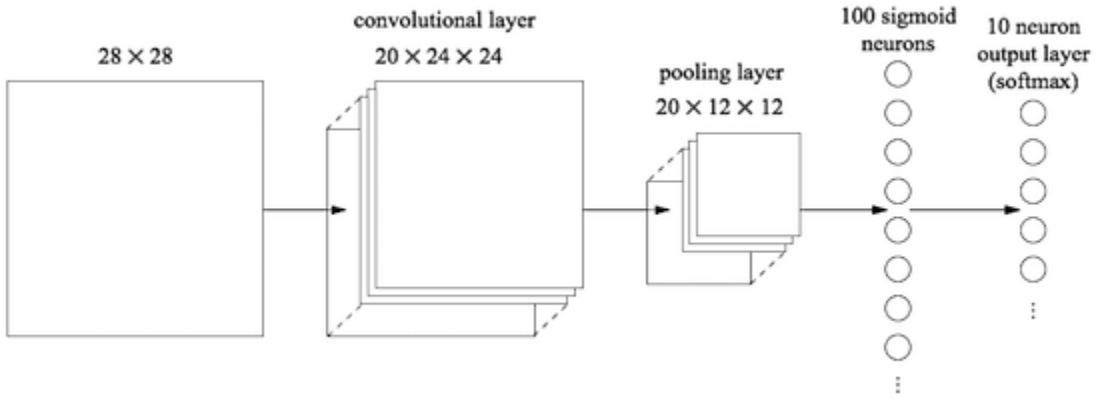

Figure 2 - Architecture of a CNN

For the convolutional layers, the neurons are arranged in a two-dimensional array known as a feature map. The weight parameters are also arranged in a two-dimensional array known as the kernel. The kernel maps a region of the input feature maps to produce the activation value of a single neuron of the next output feature map as shown in Figure 3. The kernel then slides through the input layer to produce the entire feature map. An array of kernels, also known as a filter, produces an array of feature maps which forms the next convolutional layer. Due to massive weight replication, relatively few weight parameters are necessary to describe the behavior of a convolutional layer, resulting in small kernel sizes [5].

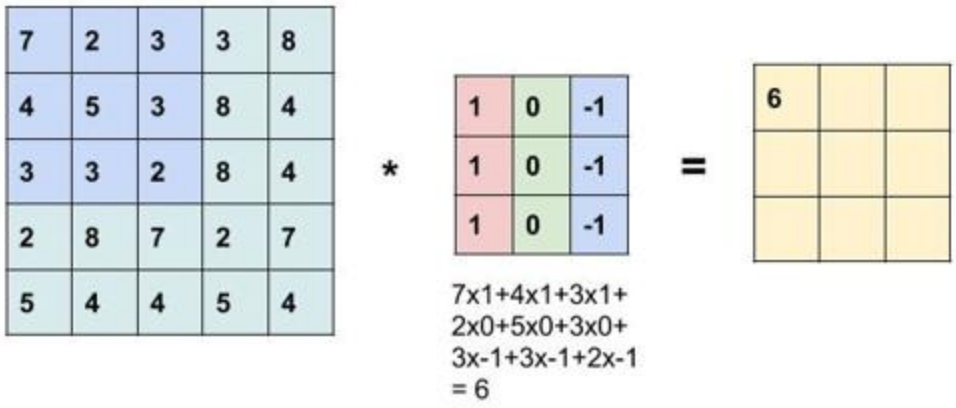

Figure 3 – Kernel computations in a CNN

There also exists a shared bias parameter for each kernel in the convolutional layer. The output activation value is expressed as

$$a_{j,k}^{l} = \sigma\left(b^{l-1} + \sum_{m=0}^{p}\sum_{n=0}^{q} w_{m,n}^{l-1}\, a_{j+m,k+n}^{l-1}\right) \tag{3}$$

where $a_{j,k}^{l}$ is the activation value of the k^{th} neuron in the j^{th} row of the l^{th} layer, b is the shared bias, w is the weight parameter of the *pxq* kernel and $\sigma(z)$ is the activation function.

Pooling layers are usually used immediately after convolutional layers. They replace the output at a certain location with a summary statistic of the nearby outputs [6]. One common pooling operation is the max-pooling operation which simply outputs the maximum activation in a *mxn* input region as shown in Figure 4. Other popular pooling functions include the average of a rectangular neighborhood, the L2-norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel [6]. The last convolutional layer is then flattened into a dense layer.

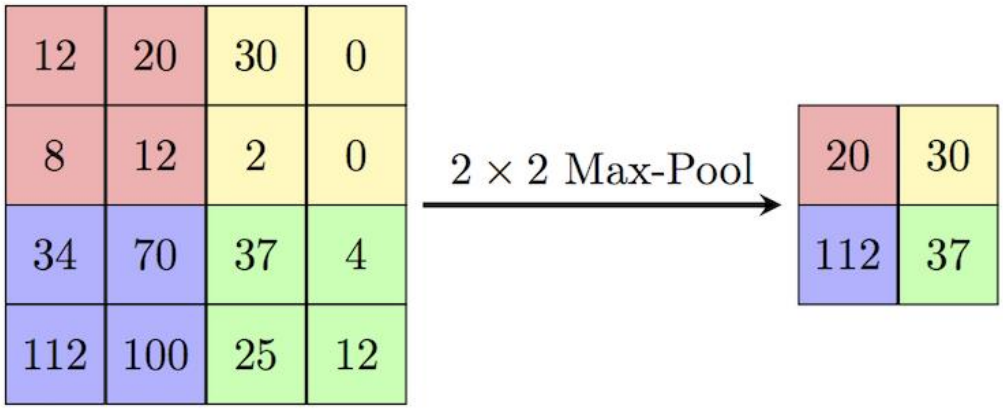

Figure 4 - Max-Pooling

Convolution helps in detecting the same pattern at different locations in the input image. Thus, convolutional networks are well adapted to the translation invariance of images.

Neural Networks learn through backpropagation where the cost is propagated back to each neuron of the hidden layer. Based on this cost, the value of each weight and bias parameter attached to that neuron is updated. This process continues until the values converge giving high accuracy.

First, the predicted output is compared with the desired output and the cost is calculated. We have used the cross-entropy cost function shown below -

$$C = -\frac{1}{n}\sum_{x}[y\, ln\, a + (1-y)\, ln(1-a)] \tag{4}$$

where n is the number of instances, y is the desired output and a is the activation output. The advantage of cross-entropy function over quadratic cost is that it prevents slow

learning of sigmoidal function [1]. The error to be backpropagated is also to be distributed amongst the weight and bias parameters. This distribution is given by the rate of change of cost with respect to the weight and bias parameters. This rate of change is given as follows –

$$\frac{\partial C}{\partial b_j^l} = \frac{1}{n}\sum_x (\sigma(z) - y) \tag{5}$$

$$\frac{\partial C}{\partial w_j^l} = \frac{1}{n}\sum_x x_j \,.\, (\sigma(z) - y) \tag{6}$$

The above two partial derivatives signify the amount of change required to the corresponding weight and bias parameters. After some iterations the cost tends to approach zero signifying that the activation output and the desired output are the same.

As the number of layers and neurons in each layer increases, the number of parameters also increase drastically. Certain parameters of these networks are redundant and removing them does not affect the accuracy of the network. Hence, pruning a neural network involves identifying and removing the connections that are not relevant in classifying the data. However, searching for these redundant parameters is difficult as the search space is huge. Modern networks like LeNet5 contains 430500 parameters [7] whereas AlexNet contains 60 million parameters [8]. There also exists networks having more than a billion parameters [9]. Hence, we need to implement an optimization technique that can quickly search for maximum number of redundant parameters and prune them.

Here, we shall define a general problem statement for pruning a neural network.

1.2 Problem Definition

Let a Neural Network model with L+1 layers be represented as

$$W = \{w \mid w \neq 0, w \in W^l, 1 \leq l \leq L\} \tag{7}$$

where W^l is a set of weight parameters of the l^{th} layer. For dense layers, it consists of all the weight parameters that exists between the two layers. Mathematically,

$$W^l = \left\{w_{j,k}^l : 1 \leq j \leq n^l, 1 \leq k \leq n^{l+1}\right\} \tag{8}$$

where $w_{j,k}^l$ is the weight parameter connecting the j^{th} neuron of the l^{th} layer to the k^{th} neuron of the $l + 1^{th}$ layer and n^l is the number of neurons present in the l^{th} layer.

For convolutional layers, it is the union of all the weight parameters that exists in all the kernels between the two convolutional layers. Mathematically,

$$W^l = \bigcup_{k^l} \{ w_{j,k}^l \mid w_{j,k}^l \in k^l, 1 \leq j \leq p, 1 \leq k \leq q\} \tag{9}$$

where k^l is a kernel of size *pxq* present in the l^{th} layer and $w_{j,k}^l$ is the weight parameter present in that kernel.

Then, the problem can be stated as [10]:

$$W^* = \underset{W' \subseteq W}{argmin} \ |W'| \ s.t. \ \ f(W) - f(W') \leq \ \delta \tag{10}$$

where $|W'|$ is the size of W', δ is the tolerance, $f(W)$ is the accuracy of the network before pruning and $f(W')$ is the accuracy of the network after pruning.

W^* denotes the best pruned model. Thus, pruning of neural network can be considered as a constraint optimization problem.

1.3 Thesis Motivation

Since the neural networks are modelled after the human biological nervous system, it is interesting to see how the brain develops its synapses with the passage of time. Table 1 shows that during the first few months of the birth, trillions of synapses are generated in the human brain. It peaks at 1000 trillion synapses for a one-year old baby. However, at the age of 10 there are only 500 trillion synapses left in the brain. This is due to a natural pruning mechanism that removes redundant synapses from the brain [11]. This pruning reduces the complexity and fastens the information processing in our brain.

Age	Number of Connections	Stage
At birth	50 Trillion	Newly formed
1 year old	1000 Trillion	Peak
10 years old	500 Trillion	Pruned and stabilized

Table 1 - Development of synapses in human brain

Also, neural networks are generally regarded as black box algorithm due to the fact that it is hard to produce interpretable rules from the network of weights. Hence, many rule-extraction algorithms have been built that produce simple human readable rules from the neural network. In these algorithms [12], pruning the network is usually a preceding step to rule extraction. It is done as pruning leads to creation of simple rules compared to unpruned networks that create large number of complex rules.

Moreover, in [13] the authors have stated that the pruned network can further achieve a space compression up to 40 times the original network using quantization and Huffman encoding as shown in Figure 5. Thus, pruning a neural network helps in reducing the space required to store large networks.

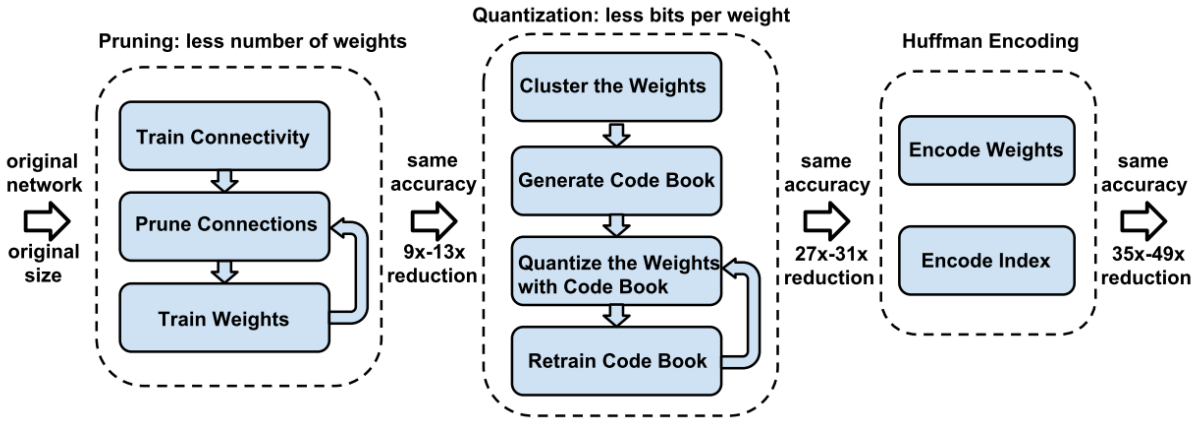

Figure 5 - Space reduction due to a sparse matrix of a pruned network.

The most beneficial outcome of pruning a neural network is the reduction in the computational costs associated it. In dense layers of a neural network, removing neurons drastically reduces the size and computational cost of the network. Let n_i denote the number of neurons in the i^{th} dense layer. The dense layer transforms the input activation

layer $x_i \epsilon R^{n_i}$ into the output activation layer $x_{i+1} \epsilon R^{n_{i+1}}$, which is used as the activation layer for the next dense layer. The number of operations of the dense layer is $n_{i+1}n_i$. When a neuron in the $(i+1)^{th}$ layer is pruned, its corresponding input weights are removed, which reduces n_i operations. The output weights of the pruned neuron are also removed, which saves an additional n_{i+2} operations.

Similarly, in convolutional layers removing a feature map reduces the size and the computational cost associated with the network. Let the number of input feature maps for the i^{th} convolutional layer be denoted by n_i , h_i be the height and w_i be the width of the input feature maps. In the convolutional layer, the input layer $a_i \in R^{n_i * h_i * w_i}$ is transformed into the output layer $a_{i+1} \in R^{n_{i+1} * h_{i+1} * w_{i+1}}$, which is used as the input layer for the next convolutional layer. To achieve this, n_{i+1} 3D filters $F_{i,j} \in R^{n_i * k * k}$ are applied on the n_i input feature maps $x_i \in R^{h_i * w_i}$. One filter generates one output feature map $x_{i+1} \in R^{h_{i+1} * w_{i+1}}$. Each filter is composed by n_i 2D kernels $K \in R^{k*k}$. Together, all these filters constitute the filter matrix $F_i \in R^{n_i * n_{i+1} * k * k}$. The number of operations of the convolutional layer is $n_{i+1}n_i k^2 h_{i+1} w_{i+1}$ [14]. When a feature map x_{i+1} is pruned, its corresponding input filter $F_{i,j}$ is also pruned, which saves $n_i k^2 h_{i+1} w_{i+1}$ operations [14]. The kernels associated with the pruned feature maps from the filters of the next convolutional layer are also pruned, saving an additional $n_{i+1} k^2 h_{i+2} w_{i+2}$ operations [14].

Thus, pruning deep neural networks saves space and computational costs and is crucial for other applications including rule extraction algorithms.

1.4 Thesis Statement

The objective of this research is to prune the neural network by removing as many redundant weight parameters as possible. This will reduce the size of the network. Along

with the weight parameters, neurons and feature maps can also be removed for dense and convolutional layers respectively.

Our approach to prune the network is to remove all the weight parameters whose absolute value lie below a certain threshold level. However, finding optimal threshold values is a challenge due the large search space. To find the optimal values quickly, we have used the modern evolutionary algorithms like Cultural Algorithms (CA) and Multi Population Cultural Algorithms (MPCA). These algorithms use domain knowledge that will lead to faster convergence. Moreover, MPCA uses a different population for each parameter to be optimized which we propose would give better results that a single population used in CA. In Section 4.2 , we will restate the problem definition taking the thresholds into account.

We measure the quality of our algorithm with pruning ratio which will be explained later. We expect to see high pruning ratio as that would signify that the network is highly pruned.

1.5 Thesis Contribution

This thesis presents the problem of pruning deep neural networks as an optimization problem. Two different models of neural networks (LeNet300-100 and LeNet5) were pruned using the standard Cultural Algorithms (CA). It also uses a novel idea to use the topography of the network as the situational knowledge of the belief space in the CA.

Moreover, a more sophisticated variant of the standard CA, known as Multi-Population Cultural Algorithms (MPCA), is also used to prune the network. A comparison is done on the effectiveness of MPCA over CA.

Also, dynamic pruning is implemented where an untrained network is pruned and trained simultaneously. Thus, this thesis contributes by implementing the following strategies –

- Using CA to prune LeNet300-100
- Using MPCA to prune LeNet300-100
- Using CA to prune LeNet5
- Using MPCA to prune LeNet5

1.6 Thesis Organization

The rest of the thesis/research work is organized in the following manner.

In chapter II, we discuss the related work/literature review in the field of pruning neural network using different techniques.

In chapter III, we introduce Evolutionary Computation and explain its working in detail. We also introduce CA and its variants like MPCA that are used in this research.

Chapter IV, we explain our proposed approach which makes it possible to utilize evolutionary techniques to reduce the search space and prune neural networks.

In Chapter V, we present the experimental setup and results with its assumptions.

In Chapter VI, we compare our work with other state-of-the-art methods and analyze the result. We also compare the results of CA and MPCA.

Chapter VII concludes the research, explaining the insights received during the work and setting up a wide range of opportunities for the future work.

Chapter 2

Literature Review

Many pruning methods are widely used to prune neural networks [15]. A network is pruned either by pruning the neurons or by pruning the weight parameters. There exist certain criteria, which when fulfilled leads to the pruning of neurons or weight parameters. For many methods, the criterion is incorporated in the form of a threshold which is either fixed or determined during training. There can also be a different threshold value for each layer.

2.1 Neuron-based pruning

Neuron based pruning methods prune neurons of the layer based on its relevance and the threshold. All the input and output weights connected with the pruned neuron are also pruned. For convolutional layers, the feature map is pruned instead of the neuron.

In [16], the sensitivity of a neuron is the criterion of pruning. The researchers calculate the sensitivity of each neuron and keep only the top k_l neurons for the layer l where k_l is a hyper-parameter. It then updates the remaining weights by using the non-linear

reconstruction error as the distance between the activation values of the unpruned model. The sensitivity of each neuron is calculated by the formula -

$$\delta_i^{(l)} \approx \sum_h \left(W_{ih}^{(l)}\right)^2 . \sum_j \left(W_{ji}^{(l+1)}\right)^2 \tag{11}$$

where $\delta_i^{(l)}$ is the sensitivity of the i^{th} neuron of the l^{th} layer and $W_{ih}^{(l)}$ is the weight parameter that connects the i^{th} neuron of the l^{th} layer to the h^{th} neuron of the $l + 1^{th}$ layer. Thus, weight parameters having high absolute values will increase the sensitivity of the neurons to which it is connected. Intuitively, it keeps neurons that are connected to weight parameters having high absolute value and prunes neurons that are connected to weight parameters having low absolute value.

One limitation of this approach is that it depends upon the user defined hyper-parameter k_l. If the user gives a large value for k_l, then there would not be optimal pruning of neurons. On the other hand, if the user gives a small value of k_l, then even the important neuron will get pruned from the network. Hence, the efficiency of the network depends upon the optimal values of k_l provided by the user.

Another limitation of all neuron based pruning methods is that, although it prunes redundant neurons, the unpruned neurons are still fully connected. There exist weight parameters from each neuron of one layer to every other neuron of the next layer. Hence, the network is still dense. For this reason, weight-based pruning methods have become more popular.

2.2 Weight-based pruning

In the weight based pruning approach, the weight parameters of a network are pruned based on certain criterion. The criterion can be the magnitude of the weight parameters or it can be calculated iteratively based on regularization.

Although we prune the weight parameters of a network, the neurons can still be pruned. If all the input weight parameters of a neuron are pruned, then that neuron will never get activated. Hence it is better to prune that neuron. Similarly, if all the output weight parameters of a neuron are pruned, then that neuron plays no role in the classification of the network. Even if such a neuron gets activated, it will not be able to activate any neuron of the next layer. Hence, it should be pruned.

As shown in Figure 6, the first and the third neuron will be pruned as it does not have any input or output weights respectively. When a neuron is pruned in a dense layer, the weights connected to the neuron are also removed. In Figure 6, when the first neuron is pruned, the output weights connected to it are also pruned.

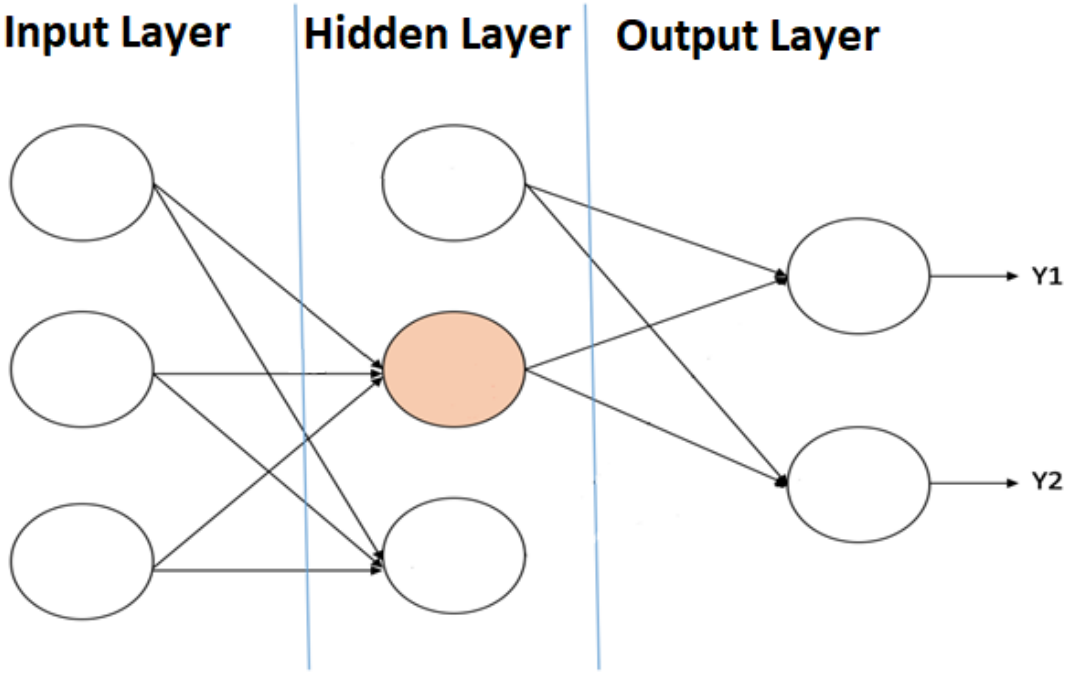

Figure 6 - Since the first and third neuron of the hidden layer do not have any input and output weights respectively, it can be pruned.

In case of a convolutional layer, the weights are arranged in a kernel. When all the weights within a kernel are pruned, the kernel is assumed to be pruned. When all the input or output kernels associated with a feature map are pruned, the feature map is also pruned. Similar to a neuron, when a feature map is pruned, all the remaining kernels associated with that feature map are also pruned.

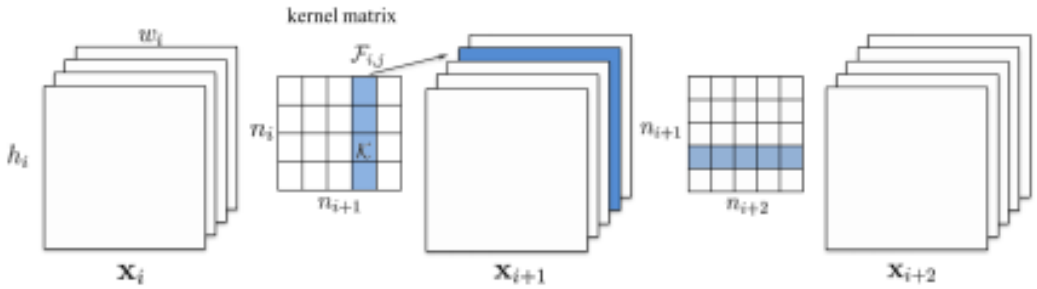

Figure 7 - Pruning Feature Map and its associated filters [14]

As shown in Figure 7, when all the input kernels in the filter $F_{i,j}$ are pruned, the corresponding feature map is also pruned. Then, all the output kernels associated with the feature map are also pruned.

The researchers in [14] pruned the filters of a convolutional layer in the neural network by calculating the sum of the absolute kernel weight parameters for each filter. This sum is the criterion for pruning and is formulated as –

$$s_j = \sum_{l=1}^{n_l} \sum |K_l| \tag{12}$$

where s_j is the sum for the j^{th} filter, n_l is the number of kernels in that filter and $|K_l|$ is the kernel with absolute values of weight parameters. Once the sum is calculated for all the filters, the filters having the *m* smallest sum are pruned where *m* is a user defined hyper parameter. Clearly, weight parameters having small absolute value will lead to a small sum and will be pruned.

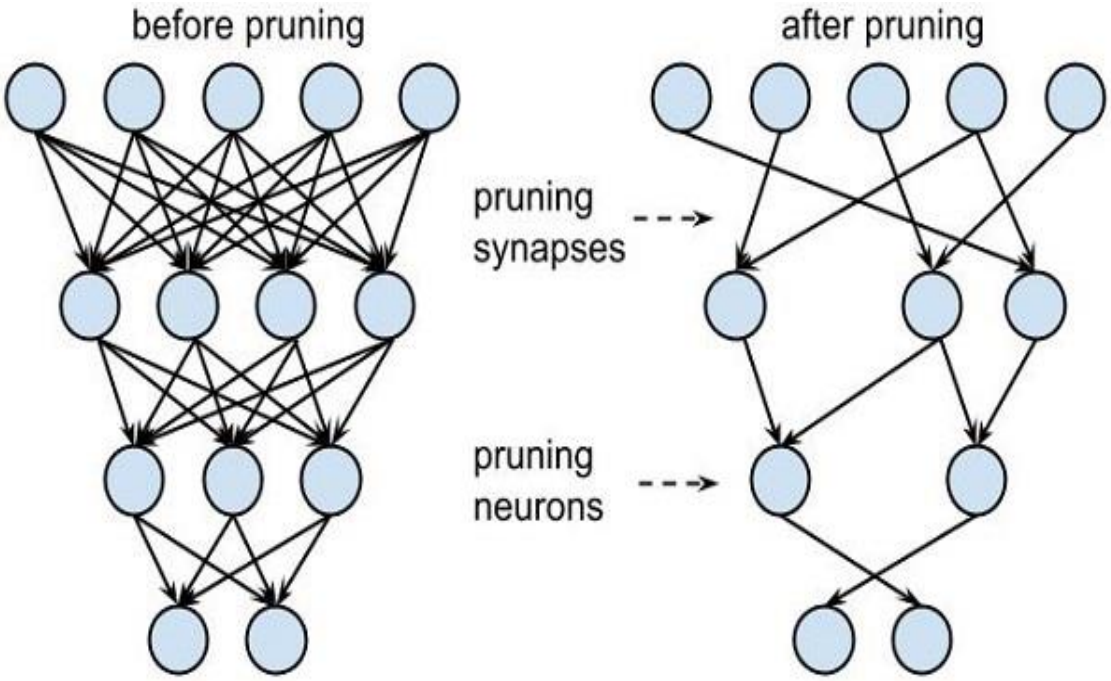

Figure 8 - Weights and neurons before and after pruning [7]

This idea of pruning weight parameters having small absolute value is carried forward in [7] where the researchers prune weight parameters with absolute value below a certain threshold level in the trained neural network. It further removes neurons that have zero input or output weight parameters. This process is repeated iteratively followed by retraining. Figure 8 shows the sparsity in the network after pruning.

Similarly, Dynamic Network Surgery (DS) [17] calculates the importance of each weight parameter and prunes the weights if its importance falls below certain threshold. The importance is calculated as –

$$h_k\left(W_k^{i,j}\right) = \begin{cases} 0, & if\ a_k > W_k^{i,j} \\ T_k^{i,j}, & if\ a_k \le W_k^{i,j} \le b_k \\ 1, & if\ W_k^{i,j} > b_k \end{cases} \tag{13}$$

where $h_k(z)$ is the importance of the z^{th} weight parameter of the k^{th} layer, $W_k^{i,j}$ is the weight parameter connecting the i^{th} weight parameter of the k^{th} layer to the j^{th} weight parameter of the $k+1^{th}$ layer, a_k and b_k are user defined thresholds for the k^{th} layer and T_k is a binary matrix of the k^{th} layer which gets updated during each iteration.

There are two novel approaches used in this method. First is the existence of two thresholds for each layer. This incorporates a small margin which is argued to have increased the robustness of the method. Another novelty is the dynamic property of the method as against the greedy approach. This means that a weight parameter that is once pruned can be re-established if they appear to be more important in the later iterations.

Method	**Approach**	**Prunes**
Jiang *et al.* [16]	Layer-wise pruning of neurons that are less sensitive.	Neurons
Dynamic Network Surgery [17]	Prunes weights based on their importance and threshold levels.	Weights
Li *et al.* [14]	Prunes filters having lowest sum of absolute kernel weights.	Filters

Han *et al.* [7]	Iterative Greedy search that finds the best connections and prunes the rest.	Weights

Table 2 - Comparison of various methods to prune neural networks

All the methods described above for pruning neural networks, through neurons or weight parameters, revolve around the central idea that weight parameters having low absolute value and neurons associated with such weight parameters are more likely to be pruned. The idea is that weight parameters with small absolute value do not activate the neurons of the next layer and hence, pruning them will not affect the result of classification. Recent researches [10] have found that having different values of threshold for each layer is better than having a single threshold value for every layer..

2.3 Evolutionary Pruning

In 1990, Genetic Algorithm was used to prune a trained network [18]. A binary representation was used for the weight parameters with the bits set to 0 to 1 depending if a weight parameter is pruned or not. Moreover, heavily trained networks are given more training cycles as a reward for fewer weights.

Since then, various evolutionary techniques have been used for evolving the architecture of neural networks. Table 3 shows the list of all the recent uses of evolutionary techniques to prune DNN.

Author	Approach
Li *et al.* 2018 [10]	Used Negative Correlated Search (NCS) to prune Deep Neural Networks
Jaddi *et al.* 2015 [19]	Used modified bat algorithm to optimize weight and structure of neural networks
Alencar *et al.* 2016 [20]	Used Genetic Algorithms to prune Extreme Learning Machines (ELM)
Samala *et al.* 2018 [21]	Used Evolutionary pruning for deep convolutional neural networks
Wong *et al.* 2016 [22]	Used Evolutionary Algorithms for optimizing and pruning neural networks

Table 3 - Recent use of evolutionary techniques to prune DNN

A recent and efficient method to prune networks is the Optimization-based Layer-wise Magnitude Pruning (OLMP) [10], which tries to automatically find the optimal threshold values using Negatively Correlated Search (NCS) technique. NCS is a population based heuristic optimization algorithm. The flowchart of the entire process is shown in Figure 9. It starts with a population of randomly initialized threshold levels for each layer. It prunes the network using these threshold values to calculate the fitness value, which determines how much the network has been pruned, incorporating the accuracy constraint. The thresholds are then updated, and the process is repeated resulting in iterative pruning and adjusting. The best threshold values are returned to prune the trained model and the pruned model is then retrained for a better accuracy.

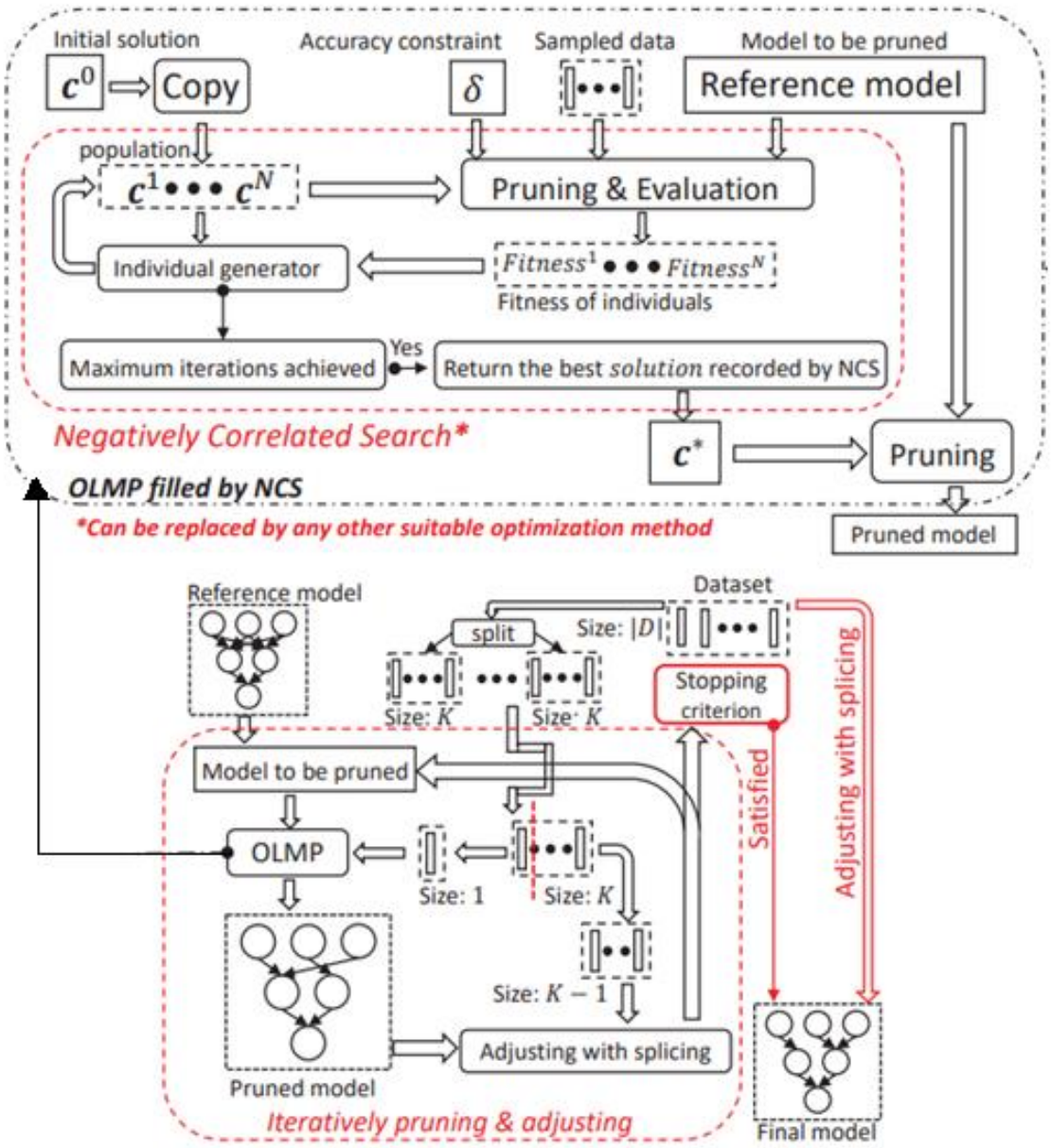

Figure 9 - The OLMP method of pruning network [10]

OLMP extends magnitude-based pruning where the fitness function is calculated based on the layer wise error and a threshold value. It achieves a better pruning ratio than other magnitude-based pruning methods.

The model of OLMP has been further investigated and improved in our research. We have used cultural algorithms and its variants to find the optimal threshold values. Also, we are pruning and training our network simultaneously, saving the time required by the network during the training phase.

Chapter 3

Evolutionary Computation

Optimization is a process which is executed to minimize or maximize an objective function until an optimum or a satisfactory solution is found. There exist many optimization problems where the computational time required to find the optimal solution is exponentially high. Evolutionary Computation contains a set of evolutionary algorithms (EA) that can find optimal or near-optimal solutions in polynomial time.

3.1 Evolutionary Algorithms

Evolutionary algorithms are metaheuristic optimization algorithms which use mechanisms inspired by the Darwin's theory of biological evolution [23]. They are population-based algorithms using the concepts of mutation, crossover, natural selection, and survival of the fittest, in order to refine a set of candidate solutions iteratively in a cycle [24].

Optimization using evolutionary algorithms involve understanding the concepts of phenotypes, genotypes, objective function, fitness function and search operations. The following definitions are stated in [24].

Definition 1. (Phenome)

The set of all the elements x that can be the solution of the optimization problem is known as the problem space or the phenome X.

Definition 2. (Phenotype)

The elements $x \in X$ of the phenome are known as the phenotypes.

Although we need to find the optimal phenotypes, the phenotypes are represented in mathematical terms so that it is possible to compute their score and execute different search operations. This representation of phenomes is known as genomes. For example, in pruning the neural networks, we need to optimize the architecture of the network so that it has a smaller number of weight parameters and neurons. For this, we find the optimal threshold values that help in pruning the weight parameters. Thus, although we need to optimize the architecture of the neural network (phenotypes), we actually optimize the threshold values (genotypes). This is because the threshold values are a better representation for the computation purposes as compared to the architectural design. The genotypes are mapped into their corresponding phenotypes using a mapping function.

Definition 3. (Genome)

The set of all elements g which can be processed by the search operations in an optimization problem is known as the search space or the genome G.

Definition 4. (Genotype)

The elements $g \in G$ of the genome are known as genotypes.

A genotype may consist of many parameters, where each parameter may represent a certain property of the genotype. These parameters are known as genes. Genes can be binary, where it's value can be either 0 or 1, or real coded, where its value is a real number. The value of a gene is known as an allele.

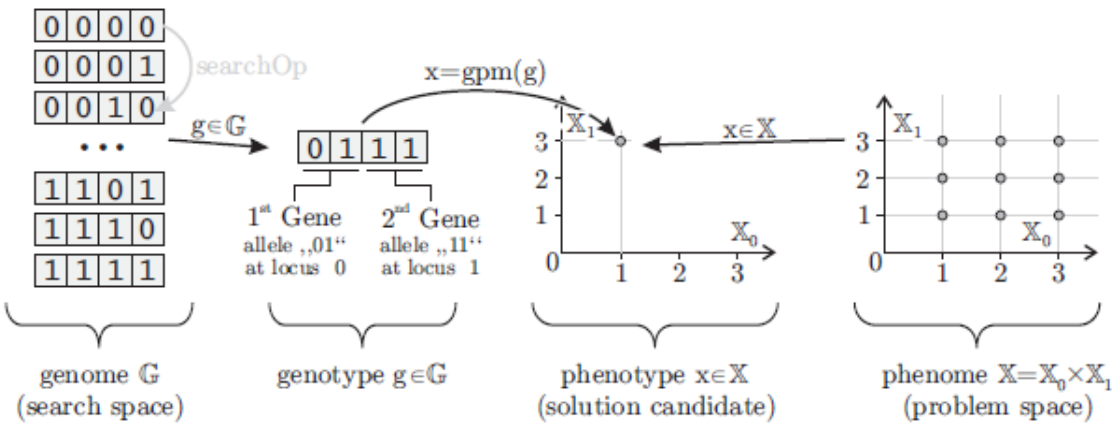

Figure 10 - Relation between Genomes and Phenomes [24]

Figure 10 shows the relation between the genomes and phenomes. The phenomes (problem space) contains a set of point on the Cartesian plane from which an optimum point is to be found for a particular optimization problem. This problem space is represented through genomes (search space) which is computationally easier to optimize. Each genotype present in the genome has binary genes. Once the optimal genotype is found, it is mapped into the corresponding optimal phenotype using a genotype phenotype mapping (gpm) function.

3.2 Genetic Algorithms

One of the most standard evolutionary algorithms is the Genetic Algorithms (GA). Genetic Algorithms, first proposed by John Holland [25] and popularized by the works of Goldberg [26], are able find good solutions to problems that were otherwise computationally intractable. They are heuristic search techniques that starts with a random population and, based on the fitness evaluation, selects individuals that will produce the successor population. This process is iterated until a stopping criterion is reached. GA helps in searching for solutions even when the domain knowledge is minimum [27].

A population is a group of individuals, where a population corresponds to a genome and each individual is a genotype. The genes of the individuals can be either binary or real coded. The search operations - crossover and mutation - directly modify these genes [28].

3.2.1 Crossover Operation

In a crossover over operation, certain genes of one individual are exchanged with the genes present at the same position of the other individual to produce two new individuals. Figure 11 shows various types of crossover operations. The simplest of all is the single point crossover where the genes after a particular point are interchanged with the genes of another genotype. In multi-point crossover, two or more points are used, and every alternate gene sequence is interchanged. In uniform crossover, there exists a probability distribution for each gene. This distribution indicates the probability with which a gene should be exchanged.

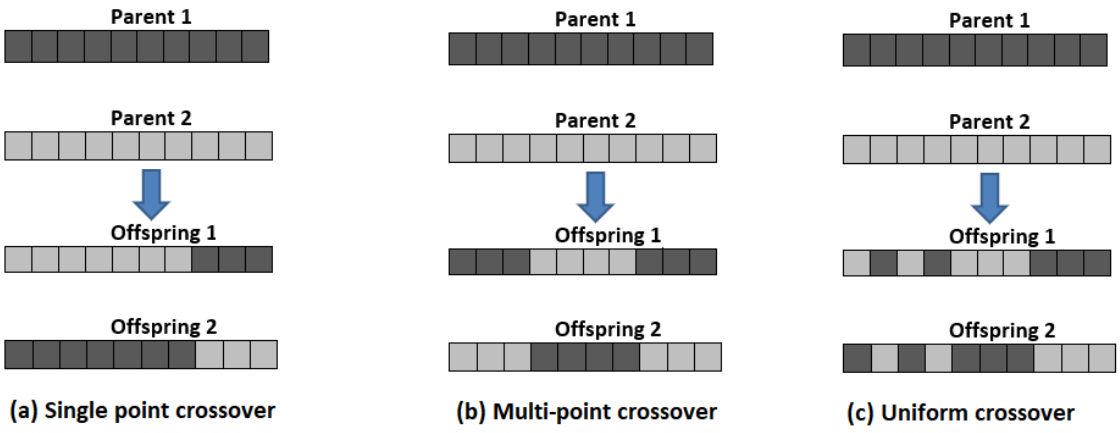

Figure 11 - Types of crossover operations

3.2.2 Mutation Operation

Crossover operation may converge all the individuals to a particular genotype. To maintain diversity and to explore new genotypes, the mutation operation is applied on a genotype. Figure 12 shows various types of mutation operations. The operation varies from a single gene mutation, which mutates only one gene of the genotype, to a complete mutation, which mutates all the genes of the genotype. Consecutive multi-gene mutation mutates a sequence of genes whereas uniform multi-gene mutation applies a probability distribution on each gene indicating the probability with which a gene is to be mutated.

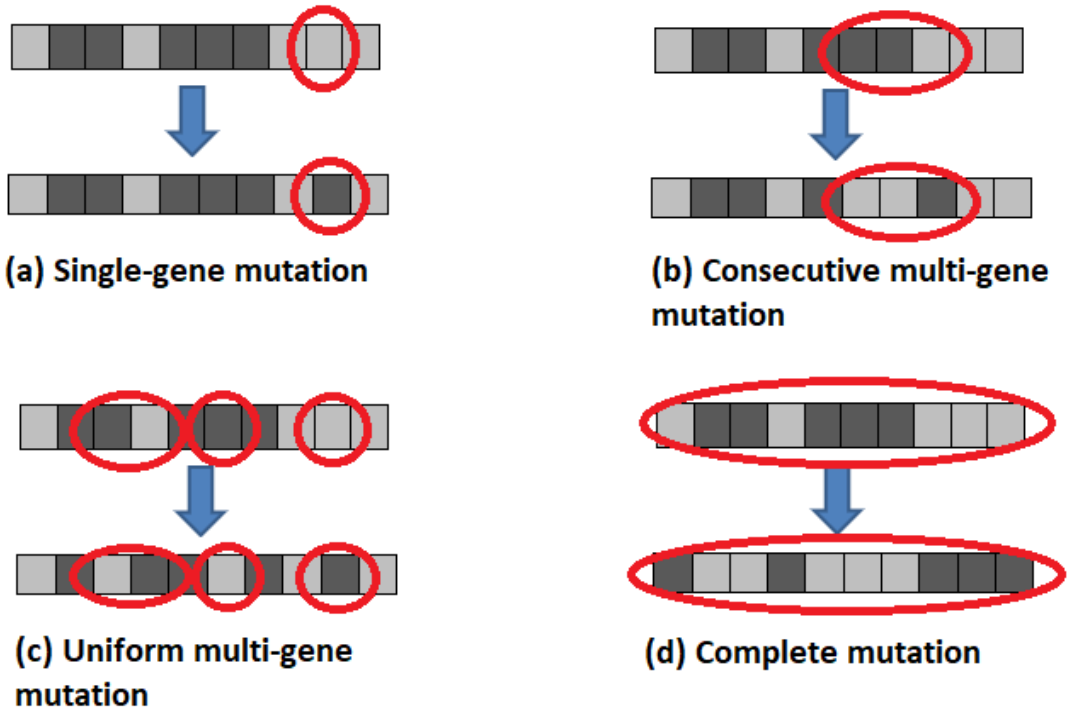

Figure 12 - Types of mutations

3.3 Cultural Algorithms

The conventional GA have little or no domain knowledge due to which it does not make full use of the historical or domain information and lack prediction about the search space [29]. However, if some domain knowledge is incorporated into the search process, then the search space is drastically reduced. Thus, domain knowledge reduces the search space by removing undesirable parts of the solution space, and by promoting desirable parts. Reynolds [30] in 1994 proposed Cultural Algorithms where the search process incorporates domain knowledge as well as knowledge acquired due to evolution to yield a better result. Unlike GA, CA enables societies to adapt to their changing environments at rates that exceed that of biological evolution. Engelbrecht [31] defines a culture as "*Cumulative deposit of knowledge, experience, beliefs, values, attitudes, meanings, hierarchies, religion, notions of time, roles, spatial relations, concepts of the universe, and material objects and possessions acquired by a group of people in the course of generations through individual and group striving*".

Cultural algorithm maintains two search spaces - the population representing the genetic component and the belief space representing the cultural component. Both these search spaces evolve in parallel and exert significant influence over one another. The experiences of individuals in the population space, identified through an acceptance function, are used for the creation of knowledge residing within the belief space. An acceptance function determines which individual's experiences should be considered to contribute to the current beliefs. This knowledge is stored and manipulated in the belief space – also known as adjusting the belief space. These adjusted beliefs then influence the evolution of the population. The communication between the two components, population space and belief space, is shown in Figure 13.

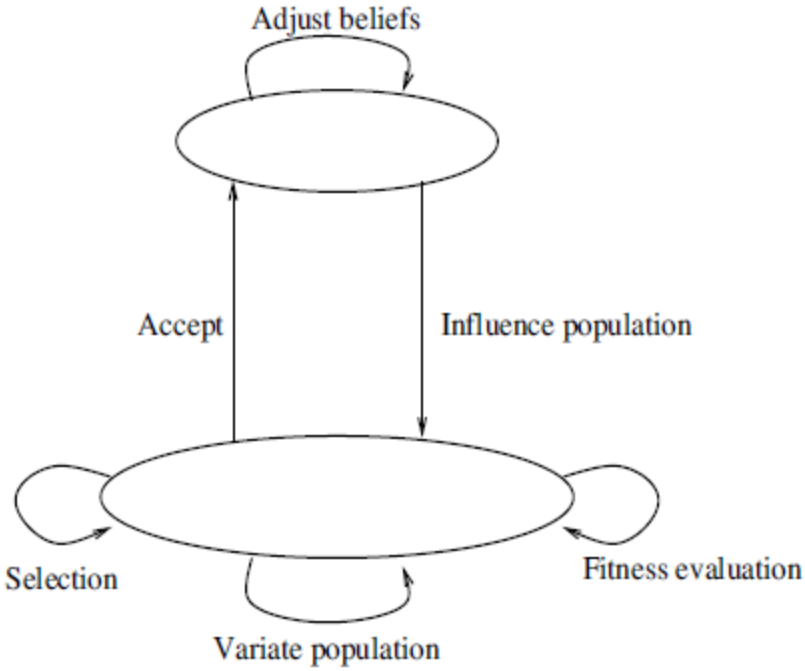

Figure 13 - Components of cultural algorithm [31]

3.3.1 Belief Space

The belief space is the central component where knowledge or beliefs of the individuals in the population space is stored. This knowledge makes the search biased towards a particular direction, resulting in the significant reduction of the search space. The belief space is updated after each iteration by the most fit individuals. The belief space has been classified into five basic categories [32]:

- Normative Knowledge: This knowledge represents a set of desirable value ranges residing within the population space. It indicates the acceptable behavior for the individuals in the population.
- Domain Specific Knowledge: This reflects some knowledge pertaining to the problem being optimized. It is also called “prior” in the Bayesian statistics.

- Situational Knowledge: This knowledge refers to the beliefs pertaining to the vital individuals in the search space.
- Historical/Temporal Knowledge: This knowledge represents the historical or the temporal patterns of the search space.
- Spatial Knowledge: This knowledge represents the landscape or topography of the search space.

3.3.2 Multi-Population Cultural Algorithms

Standard CA have only one population space where all the individuals reside. However, there have been variants of this approach. In [33], the researchers have created multiple population spaces with a single belief space as shown in Figure 14. The advantage of having multiple population spaces is that each population will try to optimize a certain parameter of the problem which is better than a single population trying to optimize all the parameters. This fastens the search process as the optimal values of the parameters are found quickly.

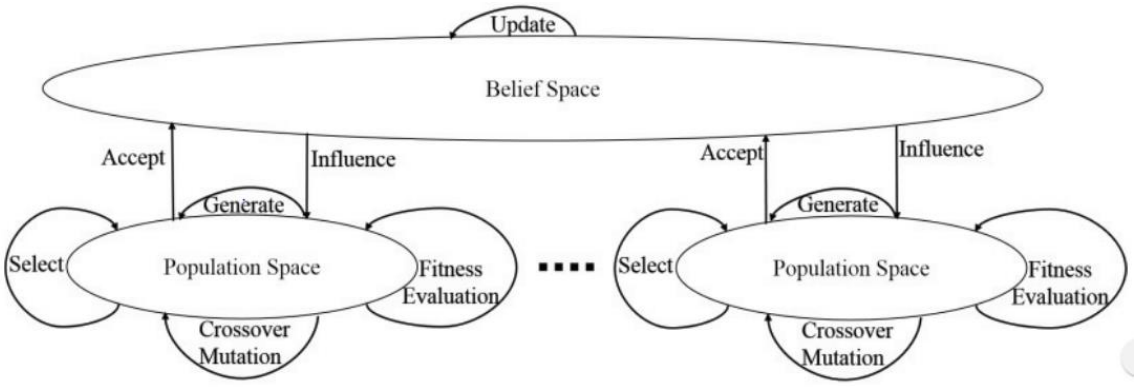

Figure 14 - Architecture of a variant of CA having multiple population space but a single belief space [33]

A more sophisticated variant of CA is having multiple population and belief spaces and transferring the implicit knowledge through another component as shown in Figure 15.

Since the belief space influences the population, it is better to have different belief spaces for different population spaces. By doing so, each belief space can influence the population space at a rate and direction that is suitable for the optimization of the parameter of that population space. Thus, rather than having one belief space influencing all the population spaces towards optimal value, this architecture would increase the speed of the search process. Such an architecture is also known as Multi-Population Cultural Algorithms (MPCA).

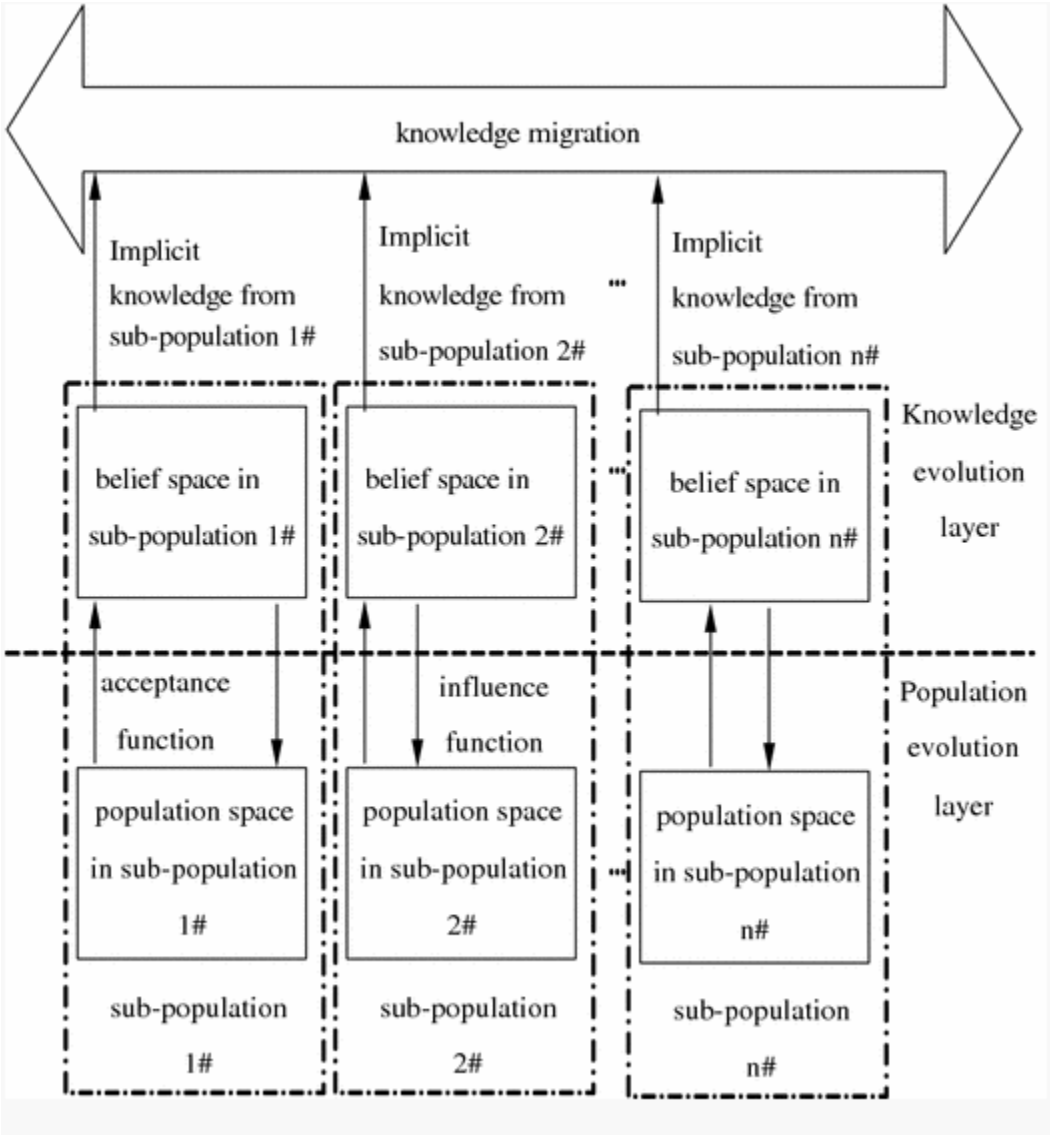

Figure 15 - Architecture of a variant of CA having multiple population and belief spaces [34]

In this research, Normative, Situational and Spatial knowledge have been used for pruning the neural networks. A comparison is also made between CA and MPCA by pruning the network through both the approaches.

Chapter 4

Proposed Approach

In this section, a description of the methods used in the processing of neural networks and in applying the cultural algorithm to prune them will be given.

4.1 Proposed Strategies to prune DNN

There are two different strategies used to prune DNN. Each strategy is used to prune two different model of DNN – one Feedforward Neural Network (FNN) and other Convolutional Neural Network (CNN). Hence there are four approaches as follows –

- Using CA to prune FNN
- Using CA to prune CNN
- Using MPCA to prune FNN
- Using MPCA to prune CNN

4.2 Evolutionary Pruning of DNN

We follow a weight-based pruning approach where the weights having a value below a certain threshold are pruned. In the layer-wise magnitude-based pruning (LMP) approach [10], there exists thresholds for each layer.

Let $c = (c_1, c_2 \dots c_L)$ be the set of thresholds for each layer. Then, we can redefine our model from our problem definition in Section 1.2 by incorporating the thresholds as

$$LMP(W, c) = \{w \mid |w| > c_l,\ w \in W^l, 1 \le l \le L\} \tag{14}$$

where $|w|$ is the absolute value of w . Now, we need to find the best values of threshold for each layer so that the pruned network has minimum number of weight parameters. Thus, the problem can be restated as [10] -

$$c^* = \underset{c \in R^L,\ W' = LMP(W, c)}{argmin} |W'|\ s.t.\ \ f(W) - f(W') \le\ \delta \tag{15}$$

where $|W'|$ is the size of W' and δ is the user defined tolerance.

c^* denotes the best threshold values which is found by running the CA and MPCA over a population of networks. After getting the hyper-parameter vector c^*, we can get a corresponding pruned model $LMP(W, c^*)$.

4.3 Individual Representation

The individual is represented with the threshold values. Each gene of an individual is a threshold value for a layer of the neural network. This value evolves during the epochs. Figure 16 shows the genes of an individual. The gene values are uniformly initialized from the range of 0 to 1.

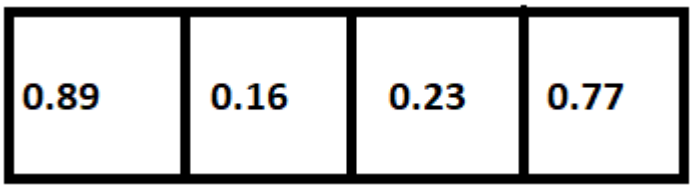

Figure 16 - Representation of an individual.

4.4 Fitness Evaluation

The objective of pruning the network is to have a high pruning ratio. The pruning ratio is the ratio of the size of the network before pruning to the size of the network after pruning. The size of the network is the number of non-zero weight parameters. This ratio indicates the extent to which a network is pruned. Higher the pruning ratio, more is the network pruned.

However, there exists a constraint on the objective. We do not want the accuracy to be drastically less than the accuracy of the unpruned network. Hence, we have defined δ,

which is the tolerance of the network. Thus, if the accuracy of the network falls by more than δ, the pruning ratio will be simply ignored.

As before, let $f(W)$ be the accuracy of the initial network and $f(W')$ be the accuracy of the pruned network. Let $|W|$ denoted the initial network size and $|W'|$ denotes the pruned network size. Then,

$$fitness = \begin{cases} f(W') - 100, & if\ f(W') < f(W) - \delta \\ \frac{|W|}{|W'|}, & otherwise \end{cases} \quad (16)$$

4.5 Pruning Neurons

In the LMP approach, we prune the weights of the network. If all the input weights of a neuron are pruned, then that neuron will never get activated. Hence it is better to prune that neuron. Similarly, if all the output weights of a neuron are pruned, then that neuron plays no role in the classification of the network and is thus to be pruned. Thus, in our algorithm, we check if there exists any neuron whose input or output weights are all pruned. If such a neuron exists, then that neuron is immediately pruned. Pruning of neurons takes place in the spatial component of the culture.

Thus, in our algorithm, we check if there exists any neuron whose input or output weights are all pruned. If such a neuron exists, then that neuron is immediately pruned. Similarly, a feature map is pruned if all the input or output kernels have been pruned. Pruning of neurons and filters take place in the spatial component of the culture.

When a feature map is pruned, the kernels associated with the pruned feature maps are also removed. Once a neuron or a filter is pruned, it is not considered again for the next

generations. Also, when a neuron or a filter is pruned for one individual network, then it is also pruned for all the networks in the population. This reduces the search space and increases the pruning ratio quickly.

4.6 Adjusting Cultures

The belief space is the most important aspect of the cultural algorithm. In our method, we have used normative, situational and spatial knowledge to adjust the belief space and influence the networks. Mathematically, the belief space is a tuple represented as

$$B(t) = [N(t), S(t), Sp(t)] \tag{17}$$

where $B(t)$ represents the belief space at generation t, $N(t)$, $S(t)$ and $Sp(t)$ represent the Normative, Situational and Spatial components respectively. Each of these components get updated simultaneously and influence every individual of the next generation.

4.6.1 Situational Component

Let $x_{best}(t)$ represent the individual having the best fitness value at generation t. Then, we update the situational component as follows -

$$S(t+1) = \begin{cases} x_{best}(t) & if\ x_{best}(t) > S(t) \\ S(t) & otherwise \end{cases} \tag{18}$$

This property of storing the best individual is known as Elitism.

Definition 5. (Elitism)

An elitist approach in an evolutionary algorithm ensures that at least one copy of the best individual(s) of the current generation is propagated on to the next generation. [24]

Elitism guarantees that the evolutionary algorithm will converge. Hence, once a global optimum basin is discovered, the algorithm will converge to that basin. However, the chances of converging to a local optimum also increases due to elitism. At the beginning of the first generation, the situational component is an empty individual, i.e., an individual which has all the genes of the value 0. Since the gene value represents the threshold value below which a weight parameter will be pruned, a gene value of 0 would indicate no pruning as there cannot be a weight parameter with an absolute value of less than 0.

4.6.2 Spatial Component

The Spatial component stores the dimension of the network. For dense layers, it is the number of unpruned neurons in that layer. For convolutional layers, it is the number of unpruned feature maps in that layer. Whenever a neuron or a feature map in any layer is pruned, the Spatial component is updated. Thus,

$$Sp(t+1) = \begin{cases} [n^1(t), n^2(t), n^3(t), \dots\ n^L(t)\ ], & if\ f(W) - f(W') \leq\ \delta \\ Sp(t), & otherwise \end{cases} \tag{19}$$

where $n^l(t)$ represents the number of neurons for the l^{th} dense layer or the number of feature maps for the l^{th} convolutional layer at generation t. This shows that the spatial component is updated only when the accuracy after pruning is within the tolerance level.

At the beginning of the first generation, the spatial component is the given input model of the neural network.

4.6.3 Normative Component

The Normative component stores the lower and upper bounds which decide the size of the search space. Its representation is similar to [31] –

$$N(t) = \left[x_{min,j}(t), x_{max,j}(t), L_j(t), U_j(t)\right] \tag{20}$$

where,

$$x_{min,j}(t+1) = \begin{cases} x_{lj}(t), & if\ x_{lj}(t) \le x_{min,j}(t)\ or\ f\left(x_l(t)\right) < L_j(t) \\ x_{min,j}(t), & otherwise \end{cases}$$

$$x_{max,j}(t+1) = \begin{cases} x_{lj}(t), & if\ x_{lj}(t) \ge x_{max,j}(t)\ or\ f\left(x_l(t)\right) < U_j(t) \\ x_{max,j}(t), & otherwise \end{cases}$$

$$L_j(t+1) = \begin{cases} f(x_l(t)), & if\ x_{lj}(t) \le x_{min,j}(t)\ or\ f\left(x_l(t)\right) < L_j(t) \\ L_j(t), & otherwise \end{cases}$$

$$U_j(t+1) = \begin{cases} f(x_l(t)), & if\ x_{lj}(t) \ge x_{max,j}(t)\ or\ f\left(x_l(t)\right) < U_j(t) \\ U_j(t), & otherwise \end{cases}$$

For each $x_l(t), 1 \le l \le n_p$.

In the above set of equations, $x_l(t)$ represents the l^{th} individual at generation t, n_p is the number of individuals in the populations (also known as the population size), $f(x_l(t))$ is the fitness value of the l^{th} individual at generation t and $x_{lj}(t)$ is the value of the j^{th} gene of the l^{th} individual at generation t. Thus, $x_{min,j}(t)$ would signify the smallest value of the j^{th} gene in the population or the value of the j^{th} gene of the l^{th} individual whose fitness value is less than that of the individual with the smallest j^{th} gene at generation t. $L_j(t)$ would represent the fitness value of the smallest j^{th} gene in the population or the fitness value of the individual that is less that the fitness value of the individual having the smallest j^{th} gene at generation t. Similarly, $x_{max,j}(t)$ would signify the highest value of the j^{th} gene in the population or the value of the j^{th} gene of the l^{th} individual whose fitness value is less than that of the individual with the highest j^{th} gene at generation t. $U_j(t)$ would represent the fitness value of the largest j^{th} gene in the population or the fitness value of the individual that is less that the fitness value of the individual having the largest j^{th} gene at generation t.

4.7 Influence Functions

Once the belief space is updated, it is used to influence the population of the next generation. Let the size of the normative component be represented as-

$$\sigma_j = x_{max,j}(t+1) - x_{min,j}(t+1) \tag{21}$$

The individuals of the next generation are updated by using both normative and situational components. The change in direction is determined by the normative component whereas the step sizes are determined by the situational component.

$$x_{ij}(t+1) = \begin{cases} x_{ij}(t) + \left|\sigma_{ij} N(0,1)\right| & if\ x_{ij}(t) < S_j(t+1) \\ x_{ij}(t) - \left|\sigma_{ij} N(0,1)\right| & if\ x_{ij}(t) > S_j(t+1) \\ x_{ij}(t) + \sigma_{ij} N(0,1) & otherwise \end{cases} \quad (22)$$

$N(0,1)$ represents the normal distribution, $x_{ij}(t)$ represents the value of the j^{th} gene of the l^{th} individual at generation t and $S_j(t+1)$ is the value of the j^{th} gene of the situational component.

The situational component also plays a role in the crossover operation. Figure 17 shows the crossover operation which is defined as –

$$x_j(t+1) = x_j(t) * 0.5 + S(t+1) * 0.5 \quad (23)$$

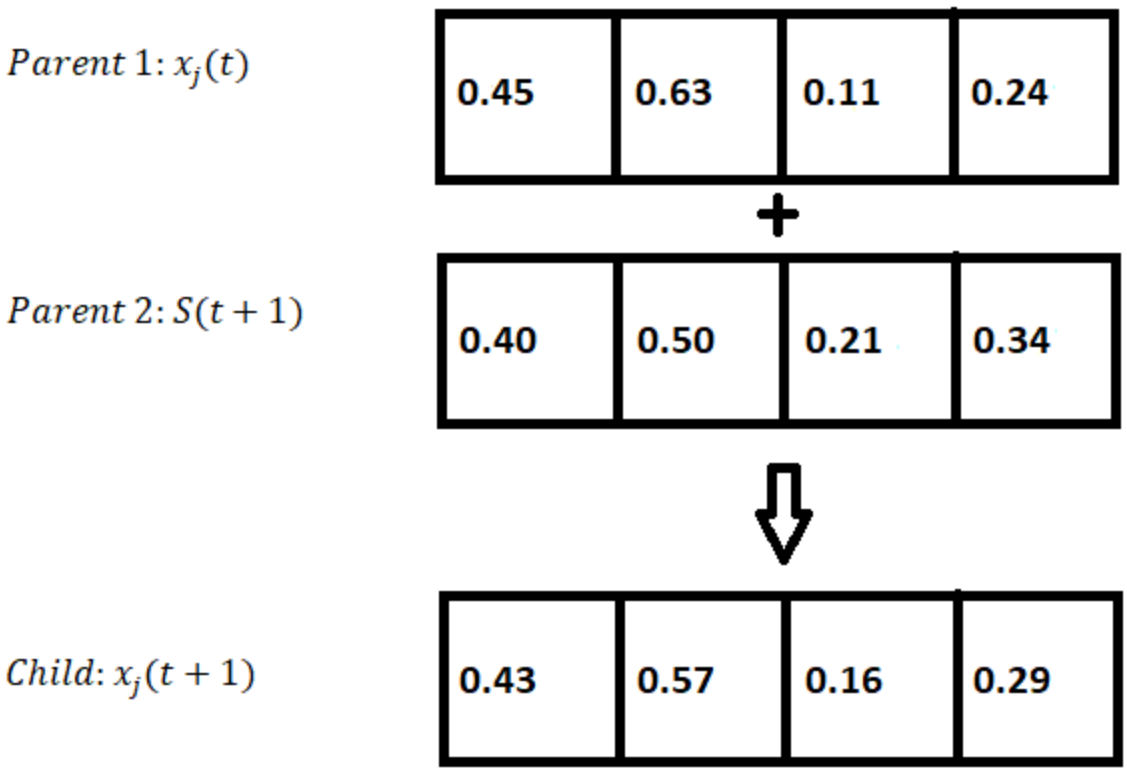

Figure 17 - Example of the crossover operation.

This is followed by a mutation operation where the gene values of an individual are increased by a factor of 1.1. The probability of a gene getting mutated is set to 10%. This is illustrated in Figure 18.

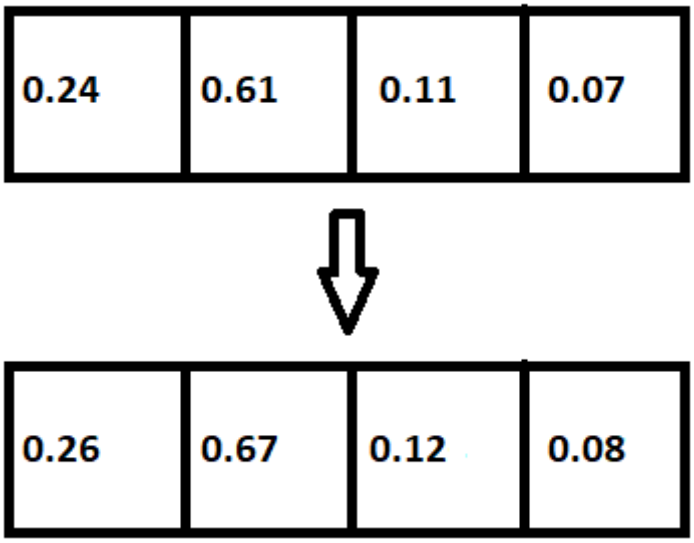

Figure 18 - Example of the mutation operation

Finally, the spatial component, if updated during a generation, influences all the individuals of the generation by updating the dimensions of every individual network. Thus, every individual network prunes the neurons and the features maps that were pruned by one individual in the previous generation. This drastically reduces the search space and leads to convergence quickly.

4.8 Using CA to prune DNN

The standard CA starts with a population of neural networks. Each network is initialized with random weight and bias parameters. Apart from these weight and bias parameters, each network also has a threshold array which would signify the individual in the evolutionary algorithm. The size of the array corresponds to the number of layers in the network. The thresholds for each layer of each network is also randomly initialized. These threshold values form the genes of the individual.

Each neural network is then trained for one epoch and the weight parameters whose absolute value lie below the threshold value are then pruned. This pruned network is further retrained for one more epoch and the accuracy of the network on the test data is then calculated. If the accuracy falls by a margin greater than δ, the fitness value is $accuracy - 100$, else the pruning ratio is calculated and is returned as the fitness value. The value $accuracy - 100$ represents the negative error since $100 - accuracy$ is the error of the network. Thus, an individual will have a positive fitness value only if the accuracy is within the tolerance levels.

The culture of the population is then adjusted based on the fitness values of the individual networks. The network having the best fitness value gets stored in the situational component of the belief space. If there exists any individual with a positive fitness value whose neurons or feature maps can be pruned, then the spatial component is updated with the new dimension of the network. The normative component is updated to find the maximum and minimum gene value for each gene of the threshold of the individual. It also stores the fitness values achieved by such genes.

Once the belief space is updated, it is used to influence the individual to create a new population. If the spatial component is updated, then the dimension of all the individual networks are also updated. The situational and normative components are used together to influence the population as shown in Equation 22. The new population of networks is created using the crossover and mutation operations. The new threshold values will now be used in the next generation of the networks.

This process is repeated over a number of generations, which is also the number of pruning epochs. Once the last epoch ends, the network having the best fitness value from the population is selected and is further finetuned for another 100 epochs. The weight parameters, bias parameters and threshold values of this network are then recorded.

4.9 Using MPCA to prune DNN

MPCA works in a similar way but has a different population for each threshold value. Thus, a network having four layers of weight parameters would need four threshold values – one for each layer – and hence would have four populations in the MPCA method. The individuals in the populations would have only one gene.

Each population has its own belief space. The situational and the normative components work the same way as they work in the standard CA. Similarly, the spatial component, if updated for any one of the individuals in any population, will influence all the individuals in all the populations. The dimensions of the network would thus change for all the individual network.

An important aspect of the MPCA method is the knowledge migration. Figure 19 shows the flow of the knowledge migration that happen in MPCA. Initially we start with a threshold array having all values set to zero. Each population has individuals with only one gene. Initially these gene values are randomly initialized. The individuals only change that value of the threshold array to which its population belongs to. Hence, the individuals of population 1 would only change the value of the first element of the threshold array, individuals of population 2 would change the second value of the threshold array and so on. This is done for all the individuals and the fitness value is calculated. The gene value of the individuals having the best fitness value in each population is then selected. These best gene values are then set to the threshold array to be used for the next epochs.

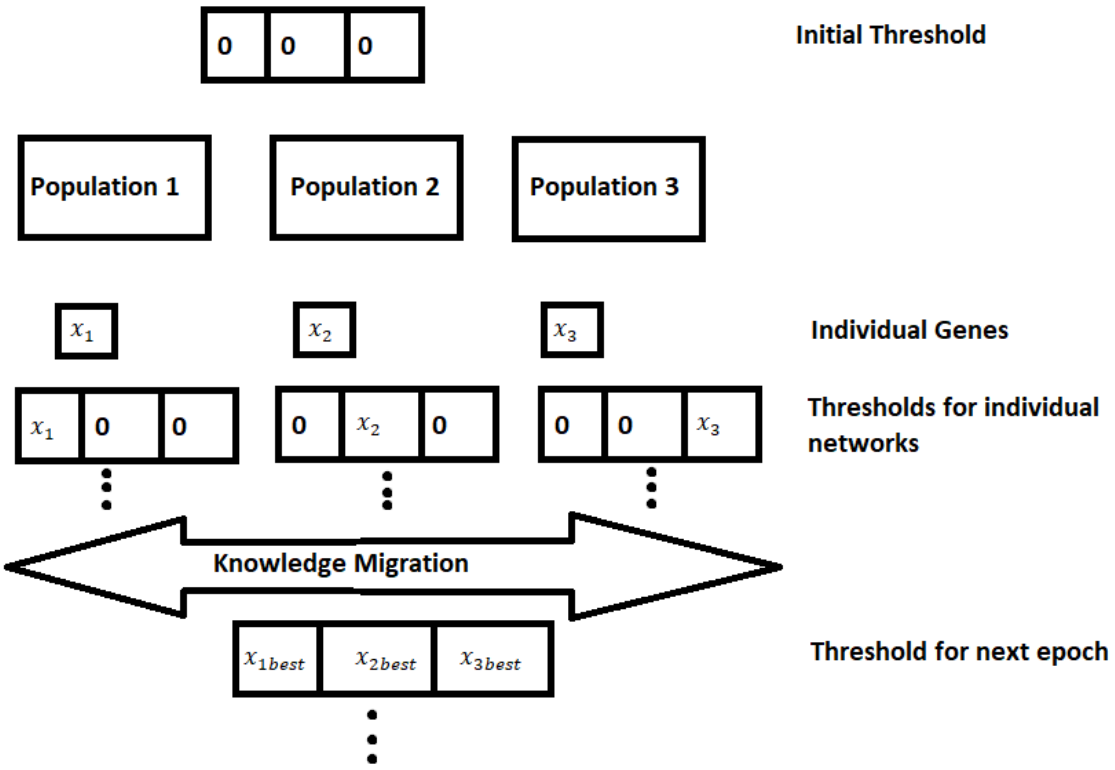

Figure 19 - Example of knowledge migration in MPCA

Algorithm 1 Algorithm for pruning using MPCA

Input: inputModel, populationSize, δ, pruningEpochs
Output: outputModel

Begin :

Initialize a population of neural network of size populationSize with random weights for each threshold.
gen = 0
while $gen < pruningEpochs$ **do**
  for each population **do**
    for each network in the population **do**
      Train network for 1 epoch
      Prune weights whose absolute value lie below the threshold value for each layer.
      Retrain for 1 epoch and test it with test data
      if $(accuracy < f(W) - \delta)$ **then**
        $fitness = accuracy - 100$
      else
        $fitness = \frac{|W|}{|W'|}$
      end if
    end for
    Transfer the spatial knowledge and the best threshold value in each population.
    Adjust culture based on the fitness values
    Perform crossover and mutation operations to generate new population
    Influence the new population based on the culture
  end for
  $gen = gen + 1$
end while
Select the best network and retrain it for 100 epochs
return the best network

End

Chapter 5

Experiments and Results

In this chapter, we give the details about the experimental setup and the results obtained from those experiments.

5.1 Dataset

The algorithm was tested on the standard MNIST handwritten digits dataset [35]. The dataset consists of 60000 training images and 10000 testing images. Each image is a grayscale handwritten digit of size 28*28 pixels. Each pixel has a value ranging from 0 to 255 with 0 indicating pure black and 255 indicating pure white.

Before using the pixel values directly into our algorithm, we have to normalize the dataset. To normalize the data, we divide each pixel value by 255. This changes the range of the pixel values from 0 - 255 to 0 – 1.

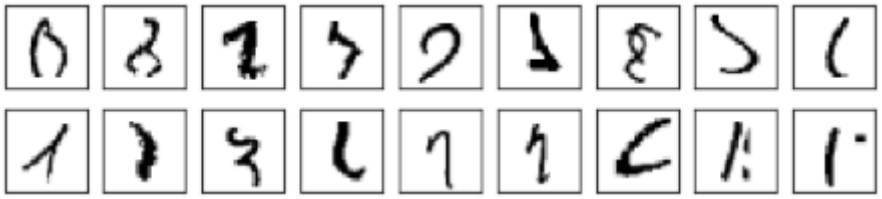

Figure 20 - Images in MNIST dataset that are difficult even for humans to recognize

5.2 Models

We have used the LeNet300-100 and the LeNet5 model which was tested by DS [17] and OLMP [10]. The LeNet300-100 contains two dense layers. The first dense layer has 300 neurons and the second dense layer has 100 neurons. Thus, the model contains a total of 266200 weight parameters. The LeNet5 model contains two convolutional layers followed by a dense layer. The first convolutional layer contains 20 feature maps. The second convolutional layer contains 50 feature maps. The first dense layer contains 500 neurons. Thus, the model contains 430500 weight parameters. Both the models were tested on the MNIST dataset.

5.3 Setting Hyperparameters

In the next chapter, we have compared our methods with ITR [7], DS [17] and OLMP [10]. To make our methods comparable with those of other author's work, the values of various hyper parameters are same as that of other methods. The value of δ was set to 6%. In CA, the population size was 10. In MPCA, each population had 4 individuals. The number of populations depended on the number of thresholds required in pruning the model. In LeNet300-100 model, there were three populations while in LeNet5 model, there were four populations. The number of pruning epochs was 15. After pruning, the network was finetuned for another 100 epochs.

The Pruning Ratio (PR) is the ratio of the size of the unpruned network to the size of the pruned network, i.e., $\frac{|W|}{|W'|}$.

5.3 Using CA to prune LeNet300-100

Figure 21 shows the best cost achieved by an individual network of the population during each pruning epoch. The best cost is the Pruning Ratio (PR) if the accuracy of the pruned network is within the tolerance level, else it is 0. The pruning ratio increased significantly to end at 200 at the end of the fifteenth epoch.

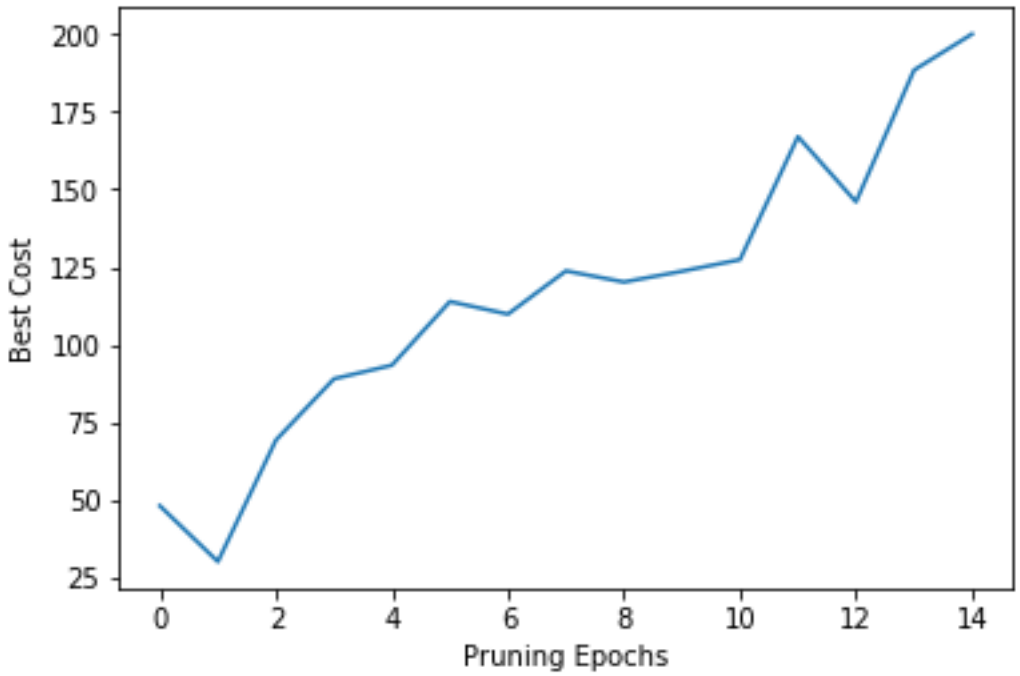

Figure 21 - Graph showing the best cost achieved at each pruning epoch for pruning LeNet300-100 by CA

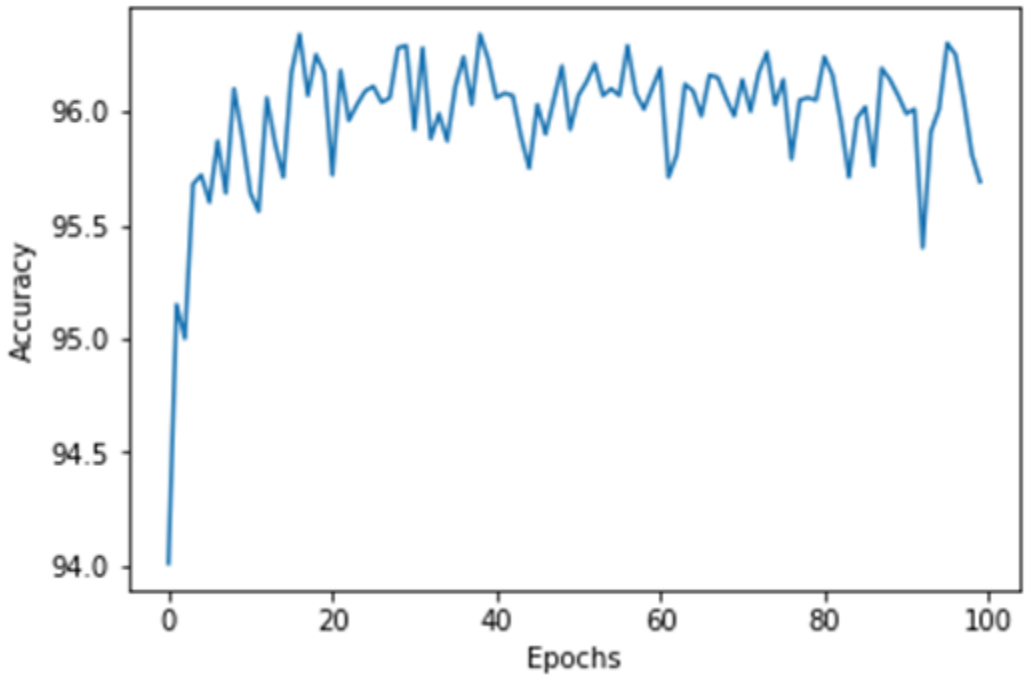

Figure 22 - Improvement in accuracy after finetuning.

Figure 22 shows an improvement of 2% in the accuracy of the network at the end of finetuning. However, the accuracy remains the same after 20 epochs of finetuning. Hence, the finetuning is done only for a hundred epochs.

5.4 Using MPCA to prune LeNet300-100

Figure 23 shows the pruning ratio achieved at each epoch after pruning the LeNet300-100 model by using MPCA. At the end of the 14th epoch, the pruning ratio achieved was 277. Figure 24 shows the effect of finetuning the model after pruning. Similar to the case before, the accuracy increases by 2% during finetuning.

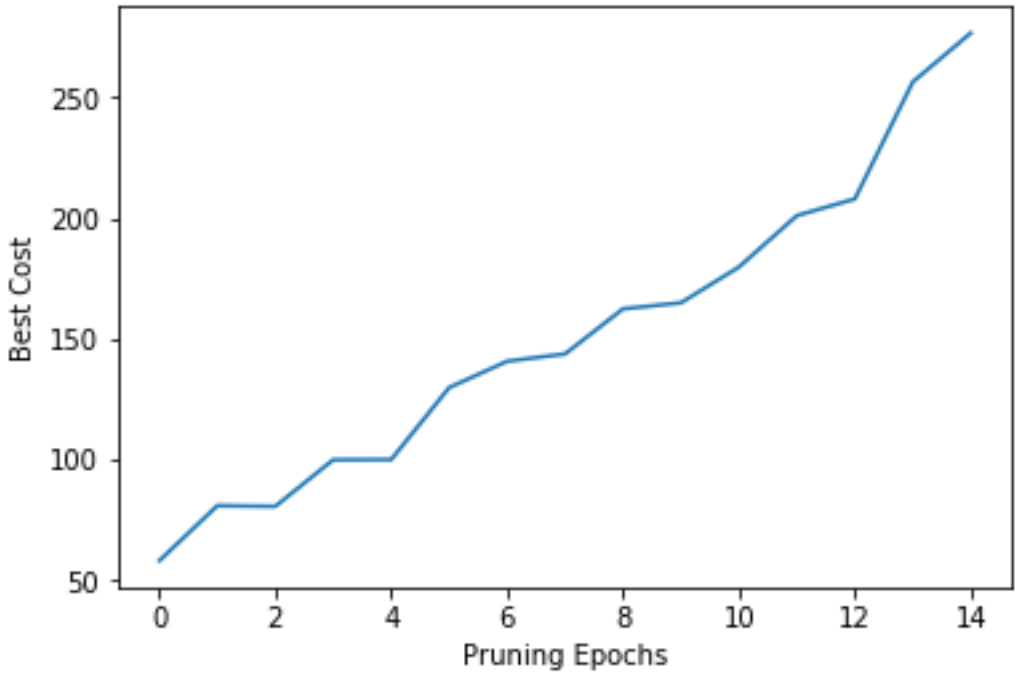

Figure 23 - Graph showing the best cost achieved at each pruning epoch for pruning LeNet300-100 by MPCA

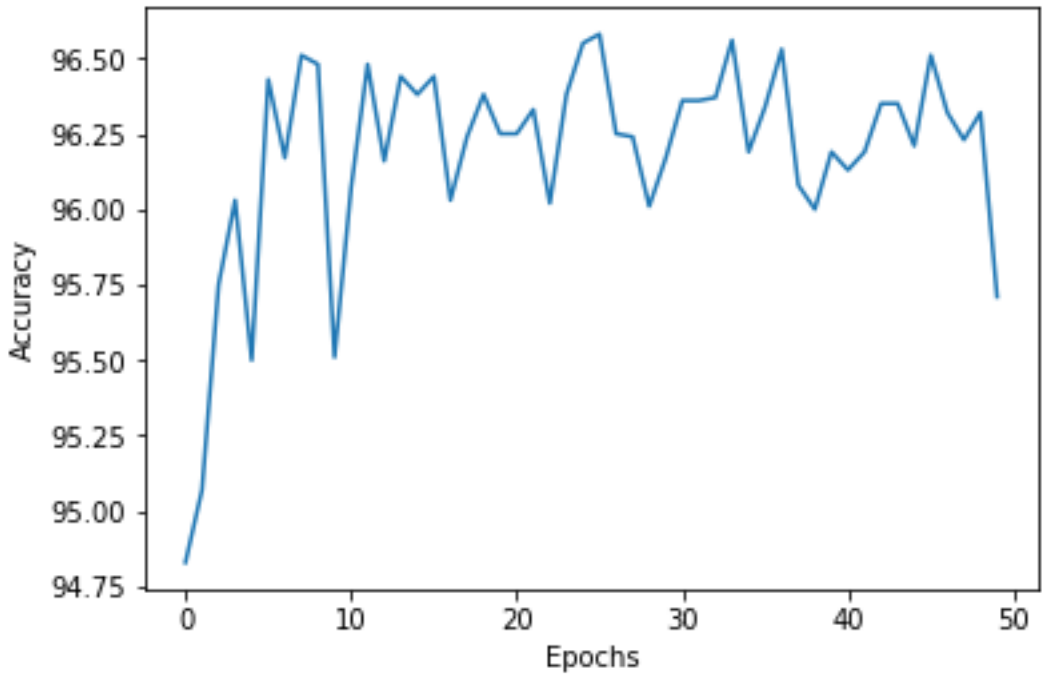

Figure 24 - Finetuning LeNet300-100 after pruning with MPCA

5.5 Using CA to prune LeNet5

Figure 25 shows the pruning ratio achieved by pruning LeNet5 with the standard CA. The pruning ratio was 592. This cannot be compared with the pruning ratio achieved in pruning the LeNet300-100 model since the two models are different. The LeNet5 model contains more layers and weight parameters as compared to the LeNet300-100 model. Figure 26 shows a similar increase of 2% in the accuracy after finetuning the model.

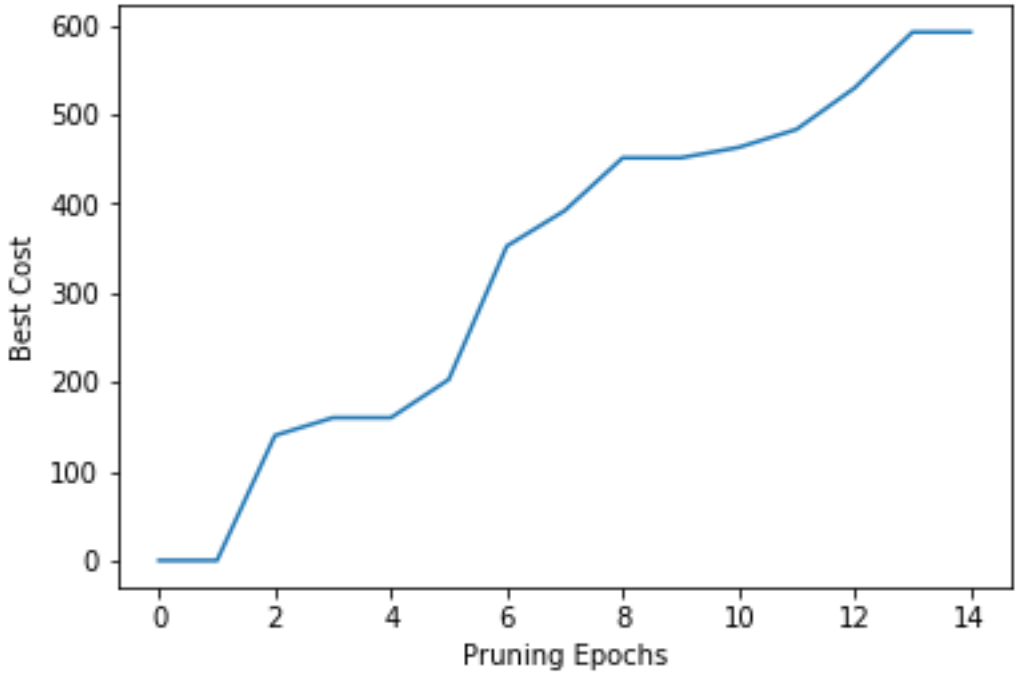

Figure 25 - Graph showing the best cost achieved at each pruning epoch for pruning LeNet5 by CA

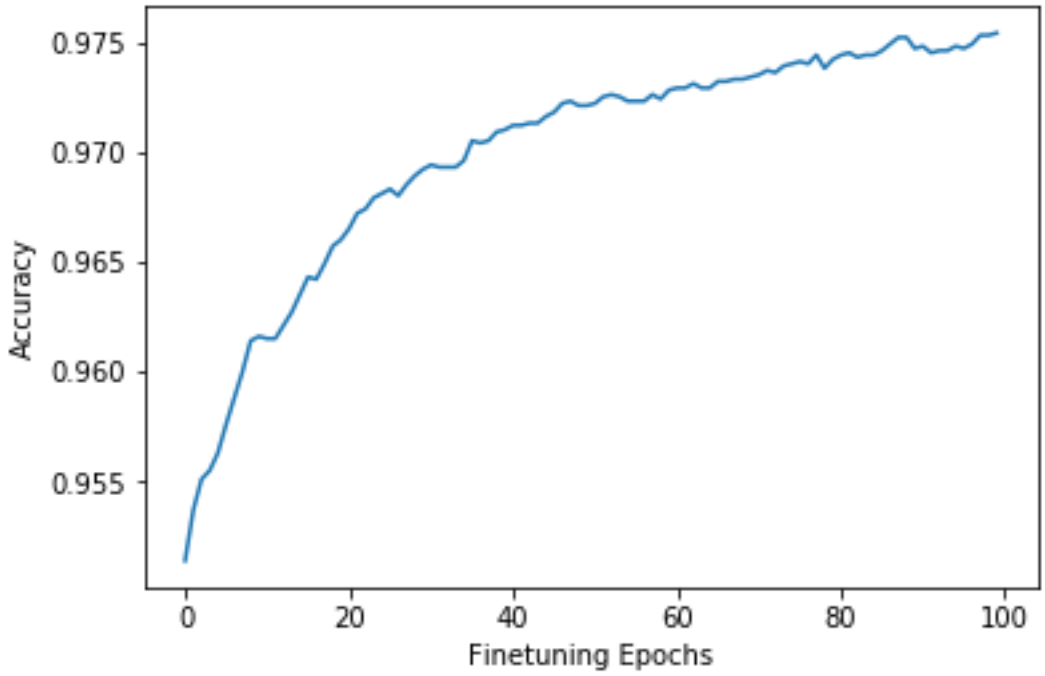

Figure 26 - Finetuning LeNet5 after pruning with CA

5.6 Using MPCA to prune LeNet5

Figure 27 shows a pruning ratio of 864 achieved after pruning the LeNet5 model with the MPCA method. This clearly shows that MPCA achieves higher pruning ratio quicker as compared to the standard CA. Similar to all the cases mentioned above, Figure 28 shows an increase of roughly 2% in accuracy after finetuning the model.

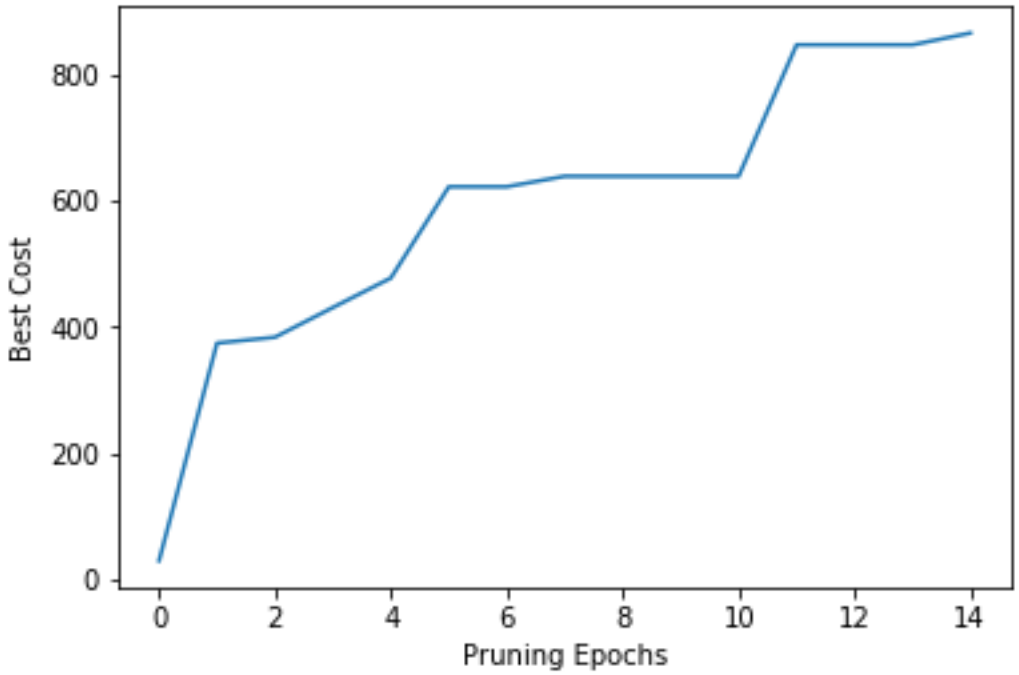

Figure 27 - Graph showing the best cost achieved at each pruning epoch for pruning LeNet5 by MPCA

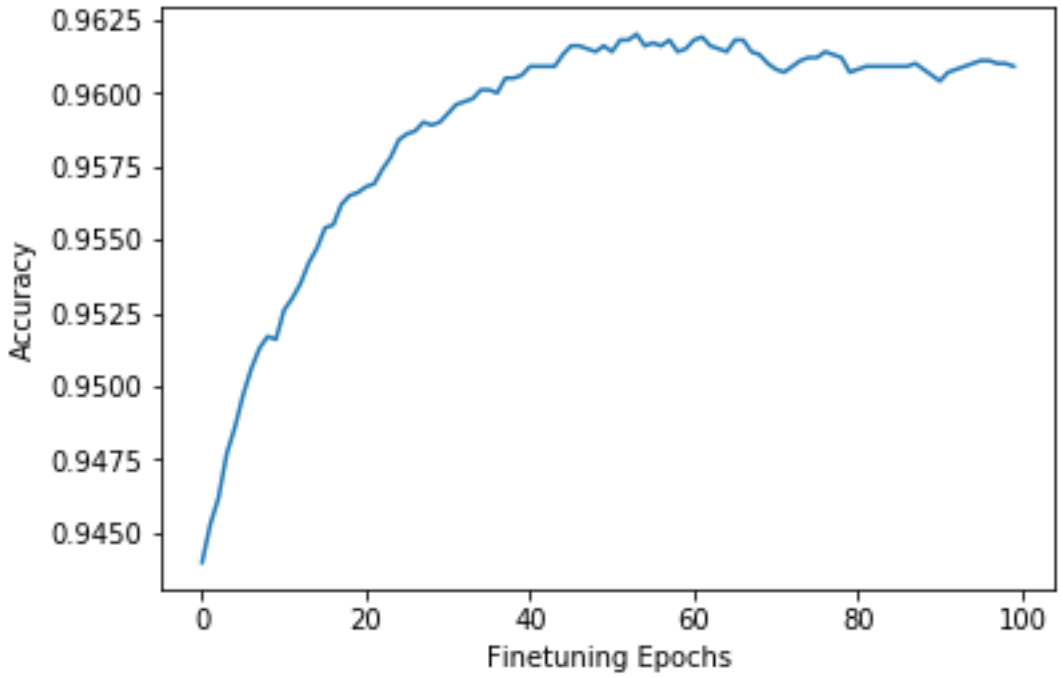

Figure 28 - Finetuning LeNet5 after pruning with MPCA

5.7 Error Analysis

All the above experiments were run three times and the best result was recorded. However, the three results didn't differ by a huge margin. The difference in the error recorded was less than 1% for each run of the algorithm. The standard deviation in the error was 0.16. This shows that the results given by the algorithm was consistent throughout the experiment.

Chapter 6

Comparison, Analysis and Discussion

In this chapter, we compare and analyze our results with that of other methods.

6.1 Comparison between CA and MPCA

Figure 29 and Figure 30 show the comparison in the performance of CA and MPCA in pruning the LeNet300-100 and LeNet5 model respectively. The graphs show the best fitness value, which is the pruning ratio if accuracy is within the tolerance levels, achieved by both the methods at each epoch. The graphs show that the MPCA prunes the network quickly as compared to the CA. For LeNet5, it takes thirteen epochs for CA to reach a fitness value of approximately 600 whereas MPCA achieves that fitness value within the fifth epoch.

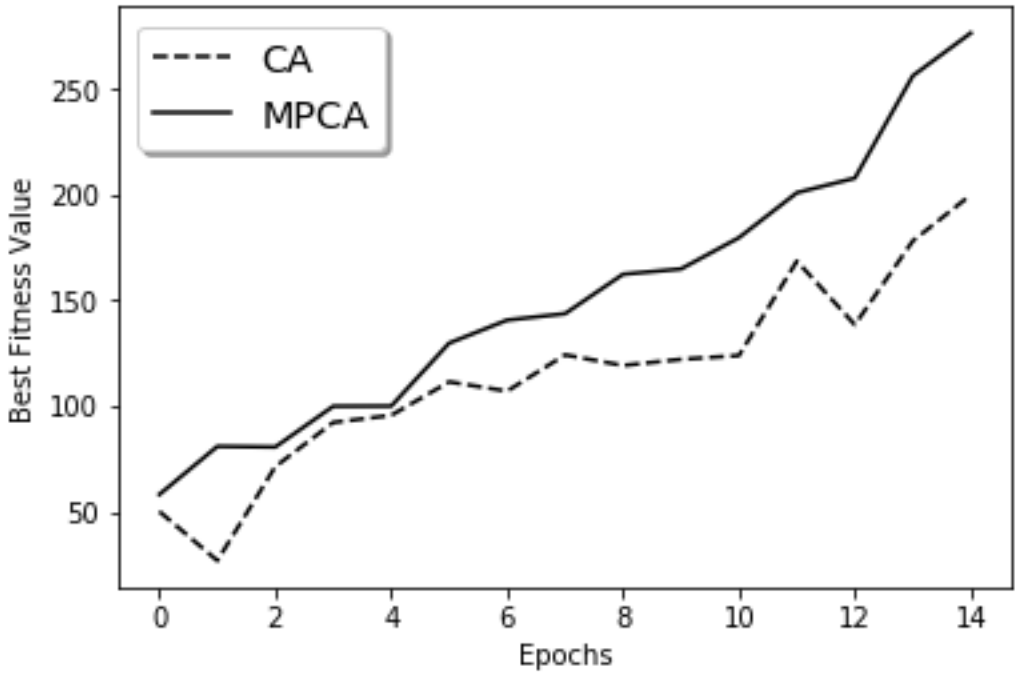

Figure 29 - Comparison of CA and MPCA for pruning LeNet300-100

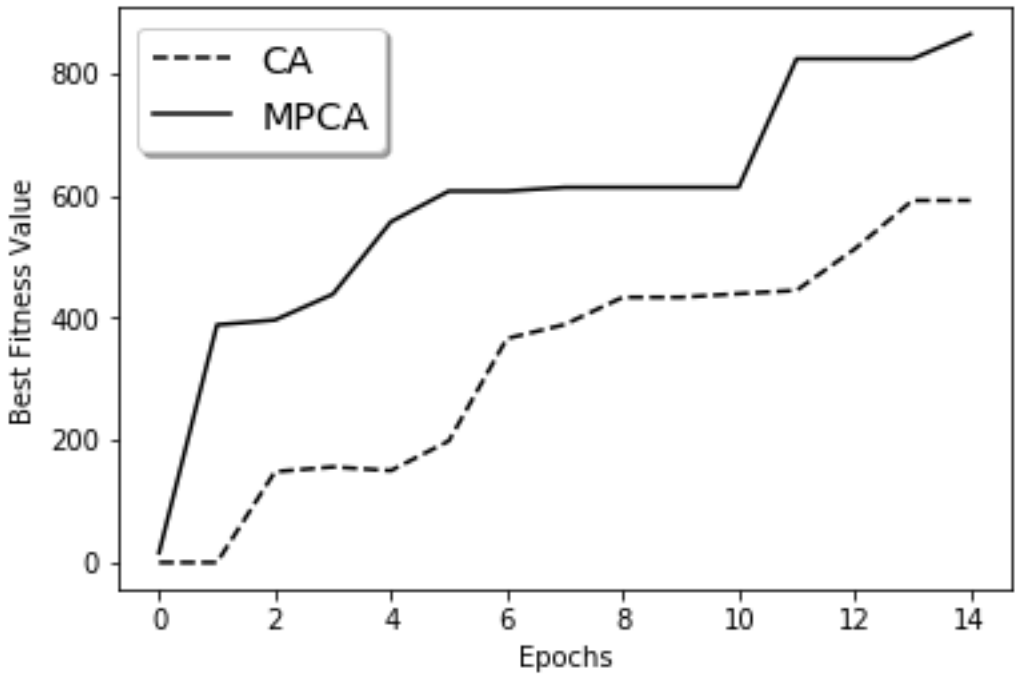

Figure 30 - Comparison of CA and MPCA for pruning LeNet5

6.2 Comparisons with the LeNet300-100 model

Table 4 compares CA and MPCA with other methods for pruning LeNet300-100. It can be seen that although our methods slightly increase the error, the PR achieved is way higher compared to other methods. While ITR achieves the best error rate of 1.59%, the pruning ratio it achieves is a mere 12. As the various methods increase the pruning ratio, they do so at the expense of the error. CA further improves the pruning ratio to 200 with a simultaneous increase in error to 3.75%. The best pruning ratio of 298 is achieved by MPCA with a corresponding error of 5.17%. It is to be noted that the tolerance level was set to 6% and hence the error posted by both - CA and MPCA – fall with the tolerance level.

Method	Error (%)	Pruning Ratio
ITR	**1.59**	12
DS	1.99	56
OLMP	2.18	114
CA	3.75	200
MPCA	3.44	**277**

Table 4 - Comparison of PR achieved by different methods for pruning LeNet300-100

Since the LeNet300-100 model contains two hidden layers, it will have three fully connected (fc) layers of weight parameters. The first fully connected layer (fc1) is between the input layer and the first hidden layer, the second fully connected layer (fc2) is between the first hidden layer and the second hidden layer, and the third fully connected layer (fc3) is between the second hidden layer and the output layer.

Since each input image has a size of 28*28 pixels, the total number of neurons in the input layer is 28*28 = 784. The hidden layers have 300 and 100 neurons and the output layer has 10 neurons representing the 10 output categories of digits (0-9). In dense layers, there exist weight parameters that connect every neuron of one layer to each neuron of the next layer. Thus, the number of weight parameters is simply the product of the number of neurons present in the two layers. Hence, the calculations of the number of weight parameters in each fully connected layer is as follows –

The fc1 layer connects the input layer with 784 neurons to the first hidden layer with 300 neurons. Hence, the total number of weight parameters in fc1 is 784*300 = 235200. The fc2 layer connects the first hidden layer with 300 neurons to the second hidden layer with 100 neurons. Hence, the total number of weight parameters in fc2 is 300*100 = 30000. The fc3 layer connects the second hidden layer with 100 neurons to the output layer with 10 neurons. Hence, the total number of weight parameters in fc3 is 100*10 = 1000. Thus, the total number of weight parameters in the model is 235200 + 30000 + 1000 = 266200.

Table 5 displays the number of initial weight parameters and the percentage of weight parameters remaining after pruning in each layer of the LeNet300-100 model.

Layer	**Params**	**Params (%) ITR**	**Params (%) DS**	**Params (%) CA**	**Params (%) MPCA**
fc1	235.2K	8%	1.8%	0.38%	0.28%
fc2	30K	9%	1.8%	0.98%	0.49%
fc3	1K	26%	5.5%	15.20%	15.30%
Total	266.2K	8%	1.8%	0.50%	**0.36%**

Table 5 - Pruning details by each layer of the LeNet300-100 network

6.3 Comparisons with the LeNet5 model

Table 6 compares CA and MPCA with other methods for pruning LeNet5. Once again, despite a slight increase in the error, the pruning ratio achieved by CA and MPCA is considerably higher compared to other methods. Similar to the LeNet300-100 pruning, ITR achieves the best error rate of 0.77%. However, the pruning ratio is only 12. OLMP has achieved a significant improvement with a pruning ratio of 298 at the expense of increasing the error to 0.91%. CA doubles the pruning ratio to 598 with a moderate increase in error to 2.50%. Finally, the best pruning ratio of 864 is achieved by MPCA with an error of 3.75%.

Method	Error (%)	Pruning Ratio
ITR	**0.77**	12
DS	0.91	108
OLMP	0.91	298
CA	2.50	592
MPCA	3.75	**864**

Table 6 - Comparison of PR achieved by different methods for pruning LeNet5

The LeNet5 model contains two convolutional hidden layers and one dense hidden layer. Thus, it will have four layers of weight parameters – two convolutional (conv) and two fully connected (fc) layers of weight parameters. The first convolutional layer (conv1) of weight parameters is between the input layer and the first layer of feature maps and the second convolutional layer (conv2) of weight parameters is between the first layer of feature maps and the second layer of feature maps. The second layer is then flattened. The first fully connected layer (fc1) is between the flattened layer of neurons and the first dense hidden layer, and the second fully connected layer (fc2) is between the first dense hidden layer and the output layer.

The convolutional layer of weight parameters is an array of filters, with each filter being an array of kernels. The kernel size is 5*5. Hence, the number of weight parameters is 25 is each kernel. The number of filters and the number of kernels in a filter depends on the number of feature maps present in the input and output layers of feature maps respectively. The input layer is the input image of size 28*28 pixels. Thus, the number of feature maps is 1. The first hidden output layer has 20 feature maps. Thus, the number of weight parameters in the first convolutional layer (conv1) is 25*1*20 = 500. The second hidden output layer has 50 feature maps. Hence, the number of weight parameters in the second convolutional layer (conv2) is 25*20*50 = 25000.

When a kernel of size $n * n$ is applied on a feature map of size $p * q$, then the output feature map has size $(p - n + 1) * (q - n + 1)$. Also, in a pooling layer when a max-pool with size $n * m$ is applied on a feature map of size $p * q$, then the output feature map has size $(p/n) * (q/m)$. Hence, when we apply the kernel of size 5*5 on the input feature map of size 28*28, then the size of the feature maps of the first hidden output layer is 24*24. After this, we apply max-pool of size 2*2 which reduces the size of the feature maps to 12*12. When the kernel is again applied on these feature maps, then the size of the feature maps in the second hidden output layer is 8*8. Again, max-pool is applied which further reduces the size of the feature maps to 4*4. This layer of 50 feature maps, with each feature map of size 4*4, is then flattened. The total number of neurons in this flattened layer is 4*4*50 = 800.

Then, the fc1 layer connects the flattened layer with 800 neurons to the first hidden layer with 500 neurons. Hence, the total number of weight parameters in fc1 is 800*500 = 400000. The fc2 layer connects the first hidden layer with 500 neurons to the output layer with 10 neurons. Hence, the total number of weight parameters in fc2 is 500*10 = 5000. Thus, the total number of weight parameters in the entire LeNet5 model is 500 + 25000 + 400000 + 5000 = 430500.

Table 7 displays the number of initial weight parameters and the percentage of weight parameters remaining after pruning in each layer of the LeNet5 model.

Layer	Params	Params (%) ITR	Params (%) DS	Params (%) CA	Params (%) MPCA
conv1	500	66%	14.2%	19.00%	16.80%
conv2	25K	12%	3.1%	0.66%	0.32%
fc1	400K	8%	0.7%	0.06%	0.06%
fc2	5K	19%	4.3%	4.68%	1.74%
Total	430.5K	8%	0.9%	0.17%	**0.12%**

Table 7 - Pruning details by each layer of the LeNet5 network

Since our objective function gives more importance to pruning, the layers having a greater number of weight parameters will be pruned heavily as compared to layers having a smaller number of weight parameters. This effect can be seen in Table 5 and Table 7. In LeNet300-100, the fc3 layer has the least number of parameters – only 1000. Hence, it is the least pruned layer and retains approximately fifteen percent of all the weight parameters. Similarly, in LeNet5, the conv1 layer has only 500 parameters and retains more than sixteen percent of the parameters. In contrast, the fc1 layer has 400K parameters and is heavily pruned, retaining only 0.06% of the total weight parameters. Thus, layers having more parameters will be heavily pruned compared to layers having less parameters.

Thus, we have made various comparisons of pruning different models of neural networks with CA, MPCA and other widely accepted methods. In each model, we have found that both CA and MPCA achieve a very high pruning ratio compared to its counterparts. Also, MPCA achieves a higher pruning ratio more quickly than the standard CA. The only

disadvantage is a slight increase in the error. Despite this increase, the error is still under the defined tolerance levels.

Chapter 7

Conclusion and Future Work

We proposed pruning of neural networks as a constraint optimization problem. We solved this problem by iteratively pruning and retraining the network. The pruning is based on threshold values which are different for each layer. Cultural Algorithms (CA) and Multi-Population Cultural Algorithms (MPCA) are used to find the best threshold values for each layer. Our results show that MPCA outperforms CA in pruning neural networks. Also, it has achieved better pruning ratio (864 for LeNet5 and 277 for LeNet300-100) compared to other state-of-the-art methods.

7.1 Limitations

One major limitation is the training of large neural networks multiple times. Consider a population of ten individuals. This would mean that a large neural network would be trained for ten times in each generation. However, as the network becomes more smaller, the time taken to train the network reduces significantly. This may be advantageous for large networks that have high number of training epochs.

Moreover, the pruned weight parameters are not realistically pruned but are set to 0 forming a sparse weight matrix. This matrix needs to be further encoded using techniques like Huffman encoding as shown by Han *et al.* [13] to truly achieve space reduction.

7.2 Future Work

We have used our algorithms to prune sequential neural netwoks, where the output of one layer become the input of the next layer. However, we would also like to extend the use of CA and MPCA to prune non-sequential deep neural network models such as Recurrent Neural Networks (RNN), ResNets [36] and Inception [37].

We would also like to test the algorithms on more complex data like the Cifar dataset [38] and ImageNet [39]. Also, we shall incorporate the idea of pruning layers in a given model of neural network.

We would also like to have a sensitivity analysis of the effect of the tolerance. We could change the tolerance value and note the pruning ratio and the error for different values of tolerance and plot a pareto front.

There exists a lot of scope for the implementation of these algorithms in pruning deep neural networks. Smaller, simpler and faster neural networks would be highly used in many applications in various fields.

REFERENCES

[1] M. A. Neilsen, Neural Networks and Deep Learning, Determination Press, 2015.

[2] C. Stergiou and D. Siganos, "NEURAL NETWORKS," [Online]. Available: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html. [Accessed 21 March 2019].

[3] K. Hornik, M. Stinchcombe and H. White, "Multilayer Feedforward Neural Networks are Universal Approximators," *Neural Networks,* vol. 2, pp. 359-366, 1989.

[4] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature,* vol. 521, pp. 436-444, 2015.

[5] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," 2014.

[6] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[7] S. Han, J. Pool, J. Tran and W. Dally, "Learning both Weights and Connections for Efficient Neural Network," in *Neural Information Processing Systems (NIPS)*, 2015.

[8] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Neural Information Processing Systems (NIPS)*, 2012.

[9] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton and J. Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," in *International Conference on Learning Representations (ICLR)*, 2017.

[10] G. Li, C. Qian, C. J. X. Lu and K. Tang, "Optimization based Layer-wise Magnitude-based Pruning for DNN Compression," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[11] P. R. Huttenlocher and A. S. Dabholkar, "Regional differences in synaptogenesis in human cerebral cortex," *Journal of Comparative Neurology*, vol. 387, no. 2, pp. 167-178, 1997.

[12] R. Sentiono and H. Liu, "Understanding neural networks via rule extraction," in *Proc. of the 14th International Joint Conference on Artificial Intelligence, (pp. 480- 485)*, 1995.

[13] S. Han, H. Mao and W. Dally, "Deep Compression: Compressing Deep Neural Networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations (ICLR)*, 2016.

[14] H. Li, A. Kadav, I. Durdanovic, H. Samet and H. P. Graf, "Pruning Filters for Efficient ConvNets," in *International Conference on Learning Representations (ICLR)*, 2017.

[15] R. Reed, "Pruning Algorithms - A Survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740-747, 1993.

[16] C. Jiang, G. Li, C. Q. and K. Tang, "Efficient DNN Neuron Pruning by Minimizing Layer-wise Nonlinear Reconstruction Error," in *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018.

[17] Y. Guo, A. Yao and Y. Chen, "Dynamic Network Surgery for Efficient DNNs," in *Advances in Neural Information Processing Systems 29 (NIPS'16)*, Barcelona, Spain, 2016.

[18] D. Whitley and C. Bogart, "The evolution of connectivity: Pruning neural networks using genetic algorithms," in *International Joint Conference on Neural Networks*, Washington DC, USA, 1990.

[19] N. S. Jaddi, S. Abdullah and A. R. Hamdan, "Optimization of neural network model using modified bat-inspired algorithm," *Applied Soft Computing*, vol. 37, pp. 71-86, 2015.

[20] A. S. Alencar, A. R. R. Neto and J. P. P. Gomes, "A new pruning method for extreme learning machines via genetic algorithms," *Applied Soft Computing*, vol. 44, pp. 101-107, 2016.

[21] R. K. Samala, H.-P. Chan, L. M. Hadjiiski, M. A. Helvie, C. Richter and K. Cha, "Evolutionary pruning of transfer learned deep convolutional neural network for breast cancer diagnosis in digital breast tomosynthesis," *Physics in Medicine & Biology*, vol. 63, no. 9, 2018.

[22] W. Wong, C. Ali, W. K. Ing, L. K. Haw and V. Lee, "Optimisation of Neural Network with Simultaneous Feature Selection and Network Prunning using Evolutionary Algorithm," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 8, no. 12, 2016.

[23] S. Upadhyayula, "Dominance in multi-population cultural algorithms," 2015.

[24] T. Weise, Global Optimization Algorithms – Theory and Application, www.it-weise.de, 2011.

[25] J. Holland, Adaptation in Natural and Artificial Systems, Ann Arbor, MI: The University of Michigan Press, 1975.

[26] L. B. Booker, D. E. Goldberg and J. H. Holland, "Classifier systems and genetic algorithms," 1989.

[27] A. Bhullar, "Improving Quality of the Solution for the Team Formation Problem in Social Networks Using SCAN Variant and Evolutionary Computation," 2018.

[28] P. Parikh, "Knowledge migration strategies for optimization of multipopulation cultural algorithm," 2017.

[29] Z. Xue and Y. Guo, "Improved Cultural Algorithm based on Genetic Algorithm," in *IEEE International Conference on Integration Technology*, 2007.

[30] R. Reynolds, "An introduction to cultural algorithms," in *Proceedings of the 3rd Annual Conference on Evolutionary*, SebaldRiver Edge, NJ, 1994.

[31] A. P. Engelbrecht, Computational Intelligence: An Introduction 2nd Edition, John Wiley Publication, 2007.

[32] Z. Kobti, R. Reynolds and T. Kohler, "A multi-agent simulation using cultural algorithms: the effect of culture on the resilience of social systems," in *Proceedings of Congress on Evolutionary Computation*, 3:1988-95, 2003.

[33] P. M. Zadeh and Z. Kobti, "A Multi-Population Cultural Algorithm for Community Detection in Social Networks," *Procedia Computer Science,* vol. 52, pp. 342-349, 2015.

[34] Y.-n. Guo, J. Cheng, Y.-y. Cao and Y. Lin, "A novel multi-population cultural algorithm adopting knowledge migration," *Soft Computing*, vol. 15, no. 5, pp. 897-905, 2011.

[35] Y. LeCun, C. Cortes and C. J. Burges, "THE MNIST DATABASE of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed 01 12 2018].

[36] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper With Convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[38] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large

Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211-252, 2015.

[40] G. v. Rossum and F. Drake, "Python Reference Manual," PythonLabs, 2001. [Online]. Available: https://www.python.org/. [Accessed 21 March 2019].

[41] O. Travis E, "A guide to NumPy," Trelgol Publishing, USA, 2006.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher and M. Perrot, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[43] M. Abadi, A. Agarwal, P. Barham and E. Brevdo, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: http://tensorflow.org/. [Accessed 21 March 2019].

[44] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, pp. 90-95, 2007.

[45] Yarpiz, "Cultural Algorithm (CA) in MATLAB," [Online]. Available: http://yarpiz.com/425/ypea125-cultural-algorithm. [Accessed 21 March 2019].

APPENDIX

All the experiments were done in Python [40]. Python is an easy to use programming language highly used in machine learning. It can be download from https://www.python.org/downloads/. To develop the above models in python, I would highly recommend installing the following python libraries –

- NumPy [41]: It is used for scientific computing in python. The installation steps are provided in http://www.numpy.org/.
- Scikit-Learn [42]: It consists of various machine learning model embedded in it. Creating and using these models are easy using this library.
- Tensorflow [43]: It is used to build and train complex machine learning models with relative ease of coding. The installation steps are listed in https://www.tensorflow.org/install.
- Matplotlib [44]: It is used to plot graphs.

The Cultural Algorithm was created using the ideas from Yarpiz [45] who developed CA in Matlab. The Multi-Population Cultural Algorithm was developed by creating different belief spaces and populations and implementing knowledge migration in the standard CA. One may need to change the hyper-parameters, activation functions or even the code used in the above references to better suit their dataset and experiments.

The code was executed on an Intel® Core™ i5-7200U CPU @ 2.50GHz 2.70GHz processor.

VITA AUCTORIS

NAME:	Anish Desai
PLACE OF BIRTH:	Mumbai, India
YEAR OF BIRTH:	1994
EDUCATION:	Bachelor of Technology in Computer Science, Veermata Jijabai Technological Institute (VJTI), Mumbai, India, 2016
	Master of Science in Computer Science, University of Windsor, Windsor, ON, Canada, 2019