Deep Learning Methods Using Levenberg-Marquardt with Weight Compression and Discrete Cosine Transform Spectral Pooling

by

James Smith

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 5, 2018

Keywords: deep learning, discrete cosine transformation, spectral pooling,
Levenberg-Marquardt (LM) algorithm, weight compression, diminishing gradient

Copyright 2018 by James Smith

Approved by

Bogdan Wilamowski, Chair, Professor of Electrical and Computer Engineering
Michael Baginski, Associate Professor of Electrical and Computer Engineering
Stan Reeves, Professor of Electrical and Computer Engineering

Abstract

State-of-the-Art Machine Learning methods achieve powerful results on practical problems in all data-driven industries. As these methods are improved, the hardware and training time necessary to implement them have grown expensive. It is desired to find Machine Learning methods that achieve near-optimal results in practical time using affordable hardware. This thesis addresses two separate artificial neural network issues resulting in practical regression, classification, and image compression algorithms. First, this thesis proposes a second-order Weight Compression Algorithm, implemented in Levenberg-Marquardt with Weight Compression (LM-WC), which combats the flat spot problem by compressing neuron weights to push neuron activation out of the saturated region and close to the linear region. The presented algorithm requires no additional learned parameters and contains an adaptable compression parameter, which is adjusted to avoid training failure and increase the probability of neural network convergence. Second, this thesis implements spectral pooling with the discrete cosine transform for image classification and compression. By using these pooling methods, greater information can be retained compared to spatial pooling while using the same network parameter reduction. These improved convolutional neural networks are found to converge faster and achieve high performance on standard benchmarks.

Acknowledgments

I would like to thank my advisor, Dr. Bogdan Wilamowski, for introducing me to the field of deep learning and guiding me in my research methods. I would also like to thank Dr. Michael Baginski for advising me on additional projects and teaching me habits of a successful scientist. Finally, I would like to thank Dr. Stan Reeves for being an outstanding instructor and a source to consult for image processing problems.

Thanks to the Auburn Electrical and Computer Engineering department and the chair, Dr. R. Mark Nelms, for funding and supporting my graduate studies in a productive and encouraging environment. I would also like to express gratitude to my friends and family, whose support has kept me going throughout my journey.

Table of Contents

Abstract ii

Acknowledgments iii

List of Abbreviations ix

1 Introduction 1

1.1 Artificial Neural Networks 1

1.2 Problem Formulation 2

1.2.1 Failing Second-Order Methods 2

1.2.2 Lossy Pooling Method 3

1.3 Contribution 3

2 Background 5

2.1 Levenberg-Marquardt Optimization and the Flat Spot Problem 5

2.2 Neural Network Architecture 9

2.2.1 Connections Across Layers 9

2.2.2 Convolutional Neural Networks 11

2.2.3 Deep Convolutional Autoencoder 13

2.3 Spectral Representation 14

2.3.1 Spectral Convolution 14

2.3.2 Spectral Pooling 15

2.3.3 Spectral Upsampling 18

3 Weight Compression Algorithm 20
3.1 Choice of WC parameters 22
3.2 LM Implementation 23
4 Discrete Cosine Transform Spectral Pooling 24
5 Experiment Results 28
5.1 Weight Compression for LM Regression 28
5.1.1 Experiment Setups 28
5.1.2 Further Considerations 29
5.1.3 Results on benchmark datasets 30
5.1.4 Additional Experiments 32
5.2 DCT Pooling for CNN Classification 32
5.3 Spectral Up-Sampling for CAE Compression 41
6 Final Remarks 51
References 53

List of Figures

2.1 Neural network as seen by a neuron . . . 5

2.2 Neuron activation for saturation scenario . . . 8

2.3 Neuron activation for non-saturation scenario . . . 8

2.4 FCC architecture . . . 10

2.5 MLP architecture . . . 10

2.6 Sucess of LM on parity-7 for varying network sizes . . . 12

2.7 Average epochs of LM on parity-7 for varying network sizes . . . 12

2.8 Training time of LM on parity-7 for varying network sizes . . . 12

2.9 Training time vs image size (in pixels) for various convolution techniques with filter dimensions as 5% of image dimensions . . . 16

2.10 Training time vs image size (in pixels) for various convolution techniques with filter dimensions as 10% of image dimensions . . . 16

3.1 Weight compression of bipolar neuron with $\Omega = 1$ . . . 23

4.1 Image subsampling for image of bird using max pooling (first row), mean pooling (second row), DFT (third row) DCT (fourth row) . . . 26

4.2 Spectral representation of bird image using (a) Discrete Fourier Transform and (b) Discrete Cosine Transfrom . . . 27

5.1 Spirals dataset . . . 31

5.2 Spirals results with LM-WC . . . 31

5.3 Convolution Neural Network Architecture Implementation for Various Pooling Methods . . . 36

5.4 Training curves with max pooling for experiment 1 (top) and experiment 2 (bottom) . . . 37

5.5 Training curves with mean pooling for experiment 1 (top) and experiment 2 (bottom) . . . 38

5.6 Training curves with FFT spectral pooling for experiment 1 (top) and experiment 2 (bottom) 39

5.7 Training curves with DCT spectral pooling for experiment 1 (top) and experiment 2 (bottom) 40

5.8 CAE network structure 42

5.9 Original MNIST digits 44

5.10 MNIST digits using Max–NN and middle dimension 10 45

5.11 MNIST digits using Max–BL and middle dimension 10 45

5.12 MNIST digits using Max–SU and middle dimension 10 46

5.13 MNIST digits using SP–NN and middle dimension 10 46

5.14 MNIST digits using SP–BL and middle dimension 10 47

5.15 MNIST digits using SP–SU and middle dimension 10 47

5.16 MNIST digits using Max–NN and middle dimension 50 48

5.17 MNIST digits using Max–BL and middle dimension 50 48

5.18 MNIST digits using Max–SU and middle dimension 50 49

5.19 MNIST digits using SP–NN and middle dimension 50 49

5.20 MNIST digits using SP–BL and middle dimension 50 50

5.21 MNIST digits using SP–SU and middle dimension 50 50

List of Tables

5.1 Training Results for Parity-7 Dataset 33

5.2 Training Results for Spirals Dataset 33

5.3 Training Results for Housing Dataset 33

5.4 Training Results for Concrete Dataset 33

5.5 Training Results for Iris Dataset 34

5.6 Training Results for Abalone Dataset 34

5.7 Training Results for Various Pooling Methods 35

5.8 Training Results for Various CAE Networks 43

List of Abbreviations

ANN	Artificial Neural Network
BMLP	Bridged Multi-Layer Perceptron
CAE	Convolutional AutoEncoder
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DCTSPL	DCT Spectral Pooling Layers
DFT	Discrete Fourier Transform
EBP	Error Backpropagation
FCC	Fully–Connected Cascading
FFT	Fast–Fourier Transform
LM	Levenberg-Marquardt
ML	Machine Learning
MLP	Multilayer Perceptron
OA	Overlap–Add
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error

RR	Random Restarts
SOTA	State-of-the-Art
WC	Weight Compression

Chapter 1

Introduction

It is often desired to use practical Machine Learning (ML) algorithms as tools to solve data-driven engineering problems, but the State-of-the-Art (SOTA) tools may not be practical due to memory and time constraints. The widely used artificial neural networks (ANNs) provide opportunity for advances in practical network designs.

1.1 Artificial Neural Networks

ANNs are utilized by researchers in a variety of fields as a supervised learning technique to predict outcomes (by regression or classification) based on knowledge from previous events (training data). Important uses include computer vision, speech recognition, and natural language processing [1]. Neural networks are loosely inspired by the biologic principles that neurons of the brain adhere to and are popular because of their minimal design requirements, problem-solving accuracy, and ability to adapt to new problems (generalization ability).

Common algorithms used to train standard feedforward neural networks are the gradient-descent algorithm, Error Backpropagation (EBP) [2], and second-order algorithms such as Newtons method and Levenberg-Marquardt (LM) [3]. second-order algorithms are advantageous to the gradient descent algorithm in that they converge at a higher rate and can handle more complex problems while retaining compact network sizes, but they often are ignored when training high-dimensional models due to heavy computations.

1.2 Problem Formulation

This thesis focuses on two related problems dealing with ANNs and presents practical solutions to each problem that can be used for everyday engineering problems without requiring expensive hardware or excessive time. The two problems are presented in the next two subsections.

1.2.1 Failing Second-Order Methods

LM is the most widely used second-order algorithm for SOTA purposes such as breast cancer diagnosis [4], cyber-physical systems [5], water supply system management [6], and facial recognition [7]. Because of its important industrial uses, any improvement made to the algorithm has transformation implications to a variety of applications.

When using algorithms such as LM to train neural networks, a typical trade-off faced by the designer is that of network size versus ease of training. It is desired to train networks using a minimum number of neurons (compact architecture) given that they reduce the required number of parameters to be learned. Furthermore, they are less likely to be over-trained, therefore retaining generalization ability for new scenarios [8]. The emergence of the Bridged Multi-Layer Perceptron (BMLP) network architecture and efficient means of its computation [9] has reduced the required number of neurons needed for many scenarios. Although these compact networks are desired, they are often more difficult to train; there are likely few existing solutions and, assuming initial random weights, it can take many attempts to perform successful training.

Training difficult problems such as those implementing compact architectures often fails because of what is known as the flat spot problem [10, 11]. This is where gradient-descent algorithms (such as LM) converge on non-optimal solutions due to diminishing gradients that disable the learning process. In neural networks, neuron outputs reach points of the activation function such that the derivative of that function is at or near zero, disabling the learning process.

This thesis presents Levenberg-Marquardt with Weight Compression (LM-WC) , an algorithm to combat the flat-spot problem in the widely-used LM algorithm. The reason we focus

on the second-order LM algorithm is that first order-algorithms are able to learn transformations with additional parameters as in the popular batch-normalization and layer-normalization methods [12, 13]. However, a solution that increases the number of trained parameters is not desired in the LM algorithm that must compute a large Jacobian matrix. Rather, we have developed a solution that requires no further parameters to be trained in LM-WC.

1.2.2 Lossy Pooling Method

In the surging big data era, convolutional neural networks (CNNs) are used with tremendous success as a deep learning technique to process grid–like data such as images [1]. Recently, several innovations have been used to improve efficiency of the convolution process using domain transforms such as the Fast Fourier Transform (FFT) [14]. A more efficient FFT–based convolution process is implemented in [15] by using the overlap–and–add method. Rippel et al. proposed frequency–domain pooling operations (spectral pooling) in [16] by utilizing the FFT. The work in [17] further extends this concept by mapping the operations and parameters of the entire convolution into the frequency domain.

This works proposed network reduces training time while retaining prediction accuracy of current state–of–the–art CNN techniques by thoroughly examining and comparing various implementations of two spectral-based techniques, 2–dimensional convolution operations using the FFT and spectral pooling using both the FFT and Discrete Cosine Transformation (DCT). The purpose of these experiments is to find the most effective methods that create a practical network that can be used for common applications such as computer vision, health care, weather forecasting, and natural language, image, and video processing. In addition to presenting DCT pooling layers, this work implements spectral up–sampling for deep convolutional autoencoders.

1.3 Contribution

This thesis presents practical solutions to the flat-spot problem of second-order algorithms and inefficient CNN pooling strategies. Specifically, this thesis makes the following contributions:

1. LM-WC is presented and used to solve high-failing benchmark problems, with comparisons to other second-order methods.

2. DCT pooling is presented and CNNs are compared using spatial pooling, spectral pooling, DCT pooling, and other relevant methods for several benchmark datasets on both image classification and compression.

Chapter 2

Background

2.1 Levenberg-Marquardt Optimization and the Flat Spot Problem

Consider a neural network as seen by an individual neuron in Figure 2.1. At pattern p, the neuron sees inputs $\boldsymbol{x_p}$ and produces the output z_p by calculating the dot product between $\boldsymbol{x_p}$ and the neurons weight vector $\mathbf{w}$. The resulting product, net_p, is acted upon by the neurons activation function f. The function F models the remaining neural network, which produces the final output o_p.

The neural network weight update rule for LM is given in (2.1):

$$\Delta \boldsymbol{w} = (\boldsymbol{H} + \mu \boldsymbol{I})^{-1} \boldsymbol{g} \tag{2.1}$$

where:

$\boldsymbol{g}$ is the gradient vector

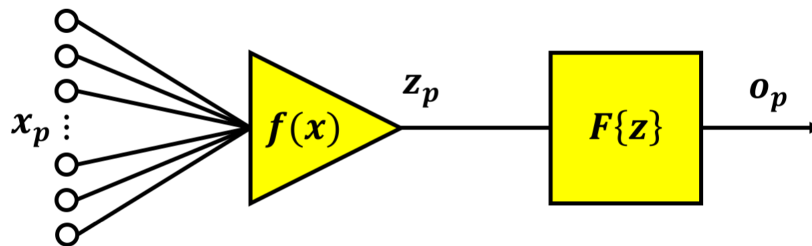


Figure 2.1: Neural network as seen by a neuron

$\boldsymbol{H}$ is the approximated Hessian matrix

$\boldsymbol{I}$ is the identity matrix

μ is a learning parameter.

Both $\boldsymbol{g}$ and $\boldsymbol{H}$ are given with respect to the cumulative (for all patterns and outputs) error vector and are calculated using the Jacobian matrix $\boldsymbol{J}$ as shown in (2.2–2.3):

$$\boldsymbol{g} = \boldsymbol{J}^T \boldsymbol{e} \tag{2.2}$$

$$\boldsymbol{H} \approx \boldsymbol{J}^T \boldsymbol{J} \tag{2.3}$$

The Jacobian is calculated by (2.4):

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} & \cdots & \frac{\partial e_{21}}{\partial w_N} \\ \frac{\partial e_{21}}{\partial w_1} & \frac{\partial e_{21}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{M1}}{\partial w_1} & \frac{\partial e_{M1}}{\partial w_2} & \cdots & \frac{\partial e_{1P}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{2P}}{\partial w_1} & \frac{\partial e_{2P}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_N} \\ \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{MP}}{\partial w_1} & \frac{\partial e_{MP}}{\partial w_2} & \cdots & \frac{\partial e_{MP}}{\partial w_N} \end{bmatrix} \tag{2.4}$$

where:

M is the total number of outputs

N is the total number of neurons

P is the total number of patterns

The error vector is given as (2.5):

$$\boldsymbol{e} = \begin{bmatrix} e_{11} \\ e_{21} \\ \vdots \\ e_{M1} \\ \vdots \\ e_{1P} \\ e_{2P} \\ \vdots \\ e_{MP} \end{bmatrix} \tag{2.5}$$

The elements of the Jacobian are calculated by (2.6):

$$\frac{\partial e_{mp}}{\partial w_i} = -[(d_p - o_{op})F'(z_p)f'(net_p)net_p] \tag{2.6}$$

where:

d_p is the desired output given pattern p

The derivative of the neuron activation function f' is used in calculating the elements of the Jacobian matrix. If f' approaches zero for any input net_p, the weight change will also approach zero, essentially labeling the neuron as stuck and unable to update its weight towards an optimal solution. This process is demonstrated in Figure 2.2 (assume bipolar neuron with tangent hyperbolic activation function). This is expected to happen if the neuron approaches its desired solution; however, if the neuron is in a high-error scenario and needing to update its weights, the neuron will remain stuck in the saturated region. The ideal neuron activation for this high error scenario is towards the linear gradient region, as presented in Figure 2.3.

A common scenario in which the flat spot problem can occur is when low error patterns that were favored by the initial weight initialization are able to give a stronger pull to the gradient-influenced weight update and prevent other patterns from contributing to the weight

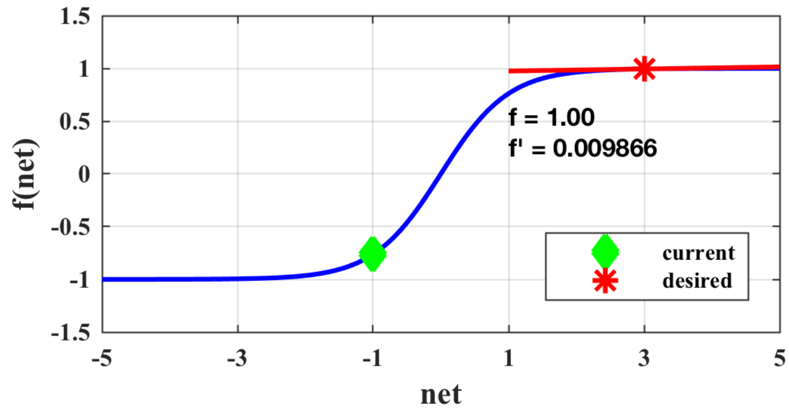


Figure 2.2: Neuron activation for saturation scenario

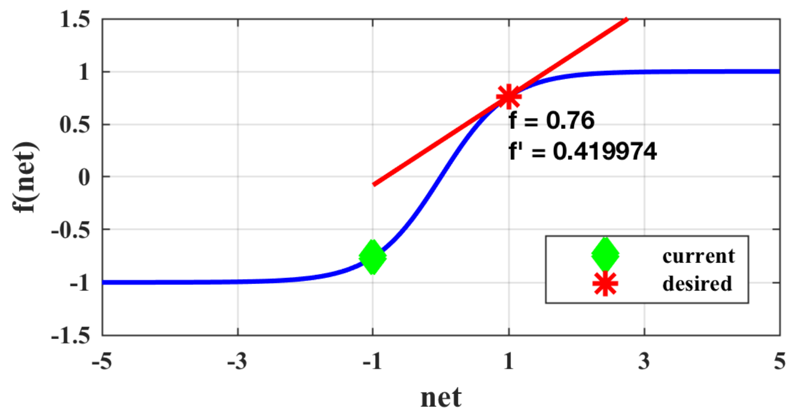


Figure 2.3: Neuron activation for non-saturation scenario

update process, due to a diminished gradient. However, if the neuron were to compress its weights back into the active region, all patterns would have an equal chance to influence the networks gradient.

Normally when an algorithm fails to converge, the user will restart the training process with new weight initialization [18, 19]. This thesis verifies that it is more effective to compress the weights than to reinitialize weights; even problems stuck in a non-optimal solution have made progress towards an optimal solution. Rather than restarting failed training attempts, the algorithm proposed in this thesis will push neuron activation towards the active region, allowing the network to retain important information already learned while incorporating information previously unreachable because of the flat spot problem.

2.2 Neural Network Architecture

2.2.1 Connections Across Layers

In [8], it is shown that compact neural network architectures outperform larger networks in that they take less memory to train and retain the ability to generalize to new patterns. It is also shown that the architecture which can perform best to these standards (i.e., successfully train to a desired training error with the smallest computation time and greatest generalization ability) is the Fully–Connected Cascading (FCC) architecture, a special case of the BMLP architecture. Unlike the standard Multilayer Perceptron (MLP) architecture, BMLP networks allow connections across layers. For the FCC architecture, each neuron is its own layer and is connected to each of the previous layers.

Consider the widely used benchmark test for neural networks, the parity-N problem. Parity-N is considered one of the most difficult problems to be solved by a neural network. According to [20], the parity-N problem can be solved with significantly fewer neurons using an FCC architecture compared to an MLP architecture. For example, Figure 2.4 shows the solution to parity-7 using an FCC architecture, which requires only 3 neurons. In comparison, the widely popular MLP architecture requires 8 neurons to solve the same problem, as shown in Figure 2.5. MLP networks have retained popularity for their relative ease of training; however,

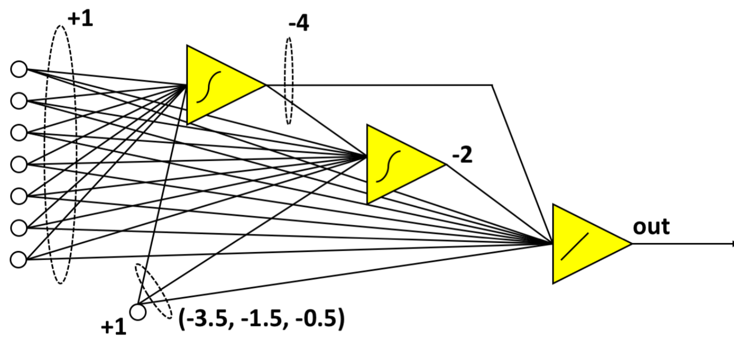


Figure 2.4: FCC architecture

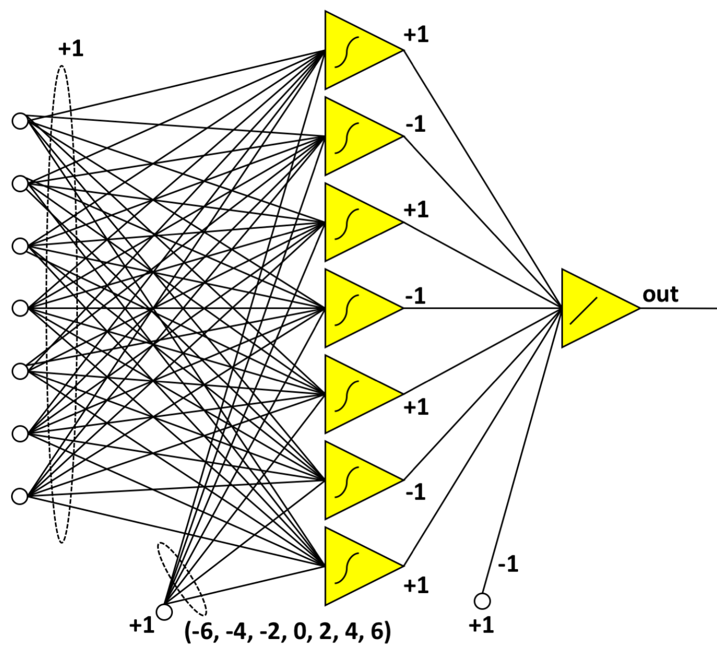


Figure 2.5: MLP architecture

as explained in Section III, our algorithm implements a second-order method which is able to handle training the superior FCC architecture.

In Figure 2.6–2.8, a case study is shown analyzing the parity 7 dataset. In this study, the LM algorithm is used to train both MLP and FCC networks of various sizes towards a desired Root Mean Squared Error (RMSE) of 0.1 with a maximum 100 epochs for 100 trials. Only successful trials contribute data for training epochs and training time. It is seen that as network size increases, the success rate (percentage of trials passing the desired error criterion) for each network increases, at the expense of increasing training time. The deeper 2-layer MLP

architectures is more difficult to train compared to the shallow single-layer MLP architecture, even though each network contains the same number of neurons. FCC networks of similar parameter size remain more accurate and computationally efficient than the MLP networks. The FCC network is able to achieve desirable results at a higher percentage of trials and a lower number of training epochs, while maintaining comparable training time.

The case study results may lead a researcher to believe that four or five hidden neurons are needed to implement a network capable of producing an RMSE of 0.1 for this problem, but, as stated previously, it is possible that a network with two hidden neurons can achieve this same goal. For 2% of the test cases, the network of two hidden neurons was able to achieve successful trials. This is significant in that it shows a very difficult problem that, if solved, can gain the researcher computation time, memory, and generalization ability.

This describes a key difficulty in training low success-rate networks: a neural network designer may not know a possible solution exists. For example, imagine a situation where a training process took days to complete and the success rate was 0.1%. The designer would likely move towards implementing a larger network before finding positive results in the smaller network. Because of this issue, it is ideal to find a learning process that can find solutions to difficult problems such as the three neuron FCC parity-7 problem, bringing a low, impractical success rate into a higher, searchable range.

2.2.2 Convolutional Neural Networks

A CNN is a supervised learning technique that applies kernel operators (convolution) as a feature extraction method to data that is then fed into a deep, fully connected MLP neural network. Two high payoff areas to improve convolutional neural networks are computational efficiency and prediction accuracy.

Methods and operations on data in CNNs are implemented as individual layers. Each layer takes as input the output from the previous layer and feeds its processed output into the next layer. The initial layer is the input data and the final layer is class predictions. To train a convolutional neural network, a forward pass is conducted on the entire network starting with the initial layer and ending with the final layer. Loss is calculated and a gradient is passed

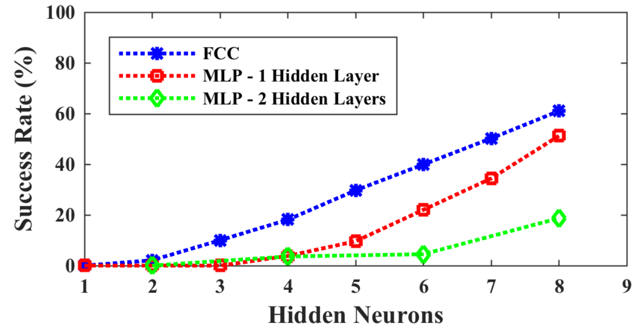


Figure 2.6: Sucess of LM on parity-7 for varying network sizes

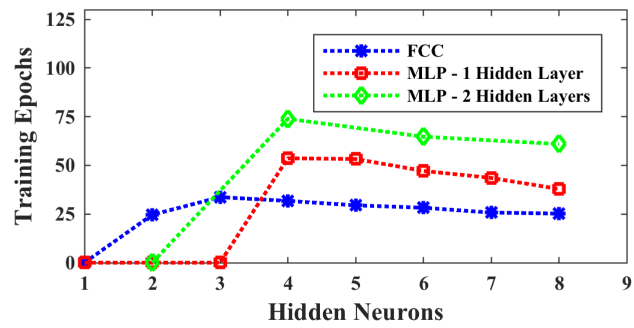


Figure 2.7: Average epochs of LM on parity-7 for varying network sizes

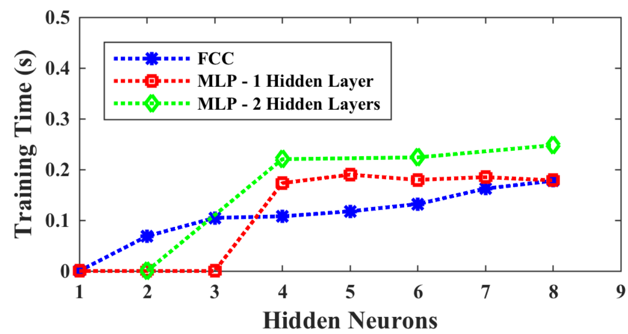


Figure 2.8: Training time of LM on parity-7 for varying network sizes

backwards to each layer in reverse order. During this process, each layer updates its parameters and propagates the gradient backwards to the next layer.

Convolutional layers, used to extract features from data, are the key distinguishing layer that separates CNNs from other artificial neural networks. In a convolutional layer, several kernels are convolved with the input data, with the results being fed into the next layer. The kernel parameters are updated as the gradient is propagated in the backward pass. Several other layers are regularly included in CNNs to improve performance. Rectified Linear Units (ReLUs) are used to avoid the diminishing gradient phenomenon during the backpropagation pass of the training process [21]. Dropout is a method used to prevent networks from overfitting data [22]. Pooling layers, such as the most commonly used max pooling layer, are used to down–sample data as a means for parameter reduction and over–training avoidance [23]. Pooling layers are especially important for CNNs because the convolution layer passes a large amount of data forward.

2.2.3 Deep Convolutional Autoencoder

Introduced by [24], a convolutional autoencoder (CAE) is a combination of two neural networks, an encoder and decoder, purposed to extract image features and compress the features into a small space. Specifically, the encoder is a CNN that learns a mapping from an image to a reduced vector and the decoder is a CNN that learns a mapping from the reduced vector back to the original image. Compared to a classic ANN autencoder architecture [25], deep CAEs use convolutional neural networks (CNNs) to abstract image features to be encoded and then deconvolution operations to decode the reduced vector.

The implementation of a deep CAE is rather intuitive. The encoder is a classic CNN composed of convolutional layers for feature extraction, pooling layers to down-sample data as a means of parameter reduction and generalization improvement [23], Rectified Linear Units (ReLUs) to prevent the vanishing gradient problem, and feedforward multilayer perceptron (MLP) layers. Techniques such as dropout are used to prevent networks from over-fitting data.

The decoder is generally made of the same building blocks as the encoder but without pooling techniques. Pooling (especially max pooling) is an irreversible operation, with discarded data being irretrievable. Rather, a nearest-neighbor method is used in the state-of-the-art deep CAE [26]. Another option used by SOTA methods is bilinear interpolation [27]. The purpose of both of these methods is to ”reverse” the max pooling operation and interpolate into a larger-dimensional space without creating undesirable artifacts associated with using convolutional or fully-connected layers.

2.3 Spectral Representation

The spectral domain offers two advantages for training CNNs. The first advantage is that the convolution operation can be sped considerably by performing the convolution in the spectral domain as element–wise multiplication. This is given as the convolution theorem (2.7):

$$\mathscr{F}\{f * g\} = \mathscr{F}\{f\} \cdot \mathscr{F}\{g\} \tag{2.7}$$

The second advantage is that the spectral domain can be used for parameter reduction while preserving great amounts of energy. The reason for this is that energy is often concentrated in only a portion of the entire spectra of the transformed data. Not all of the transformed coefficients are needed to represent the original data for practical purposes.

2.3.1 Spectral Convolution

The cause for convolution in the spectral domain being faster than convolution in the spatial domain is that the transform to the spectral domain can take advantage of the radix–2 based Cooley–Tukey FFT algorithm [28]. By using the FFT, the time complexity for the respective convolution techniques assuming an l_1 x l_2 image and an m_1 x m_2 filter are given in (2.8–2.10):

$$T_{\text{spatial}} = O\left(l_1 l_2 m_1 m_2\right) \tag{2.8}$$

$$T_{\text{spectral}} = O\left(n_1 n_2 log\left(n_1 n_2\right)\right) \tag{2.9}$$

where:

$$n_i = l_i + m_1 + 1 \quad (2.10)$$

Note that the time complexity for spectral convolution is not always better than that of spatial convolution. For very small filter sizes (such as a few pixels), the time complexity for spatial convolution can be advantageous. The primary reason for this is that the FFT algorithm requires both the image and filter to be zero-padded to the dimensions given in (4). Furthermore, it may be faster to use spatial convolution for small image sizes given the overhead computations that must be accounted for in spectral convolution. With these drawbacks being considered, spectral convolution is still powerful because it will outperform spatial convolution for large, practical–sized problems.

In the case of dealing with large images accompanied by small filters, the overlap–add (OA) convolution method can be used to prevent padding the filter to size n_1 x n_2. This is not practical for the case of Fourier–spectral pooling, but it is practical for using spectral convolution without Fourier–spectral pooling. This method will also be used in the later presented DCT–based spectral pooling.

In order to compare 2D convolution methods, Figures 2.9 and 2.10 show computation time for images of increasing sizes using spatial convolution, spectral convolution implemented with an FFT, and the OA method using MATLAB commands. The kernel dimensions for each image are 5% and 10% of the image dimensions, respectively, to demonstrate typical convolution times seen in CNNs, where the filter size is typically considerably smaller than the image size. Pay careful attention to the log scale on the time axis; the spatial convolution grows exponentially faster than the other two methods. OA is more efficient than using the FFT (keep in mind the log scale); however, this advantage is diminished as the filter size increases.

2.3.2 Spectral Pooling

Spectral pooling was developed by Rippel et al. [16] to take advantage of the powerful frequency-domain properties. Spectral pooling applies a low-pass filter to the spectral representation of

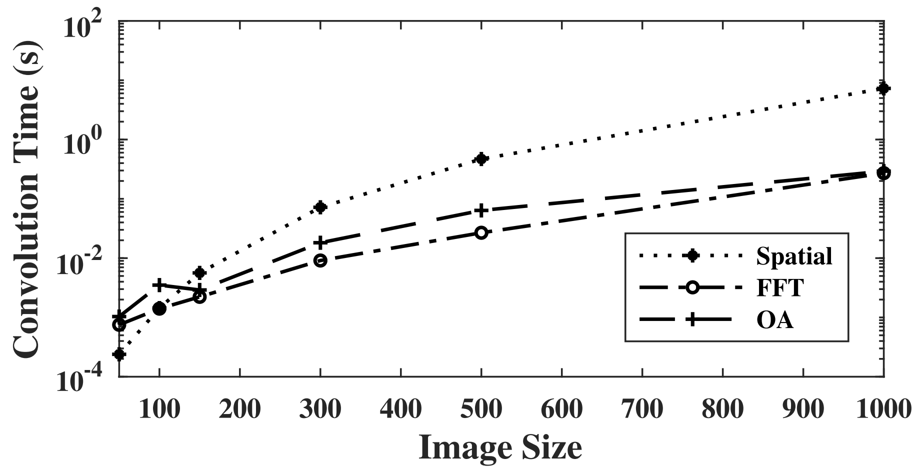


Figure 2.9: Training time vs image size (in pixels) for various convolution techniques with filter dimensions as 5% of image dimensions

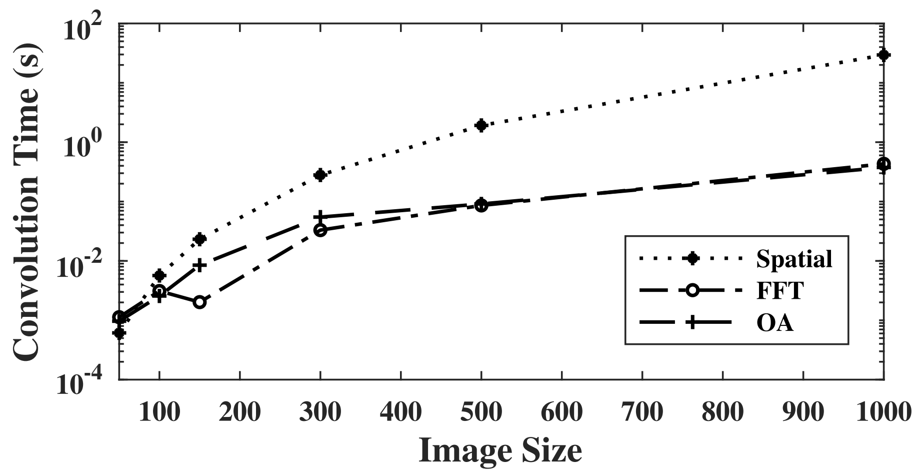


Figure 2.10: Training time vs image size (in pixels) for various convolution techniques with filter dimensions as 10% of image dimensions

an image-kernel product. Simply put, spectral pooling is truncating data in the frequency domain, as described in Algorithm 1. Assume an image with *c* color channels and and a desired dimsinon size of p_1 x p_2

Algorithm 1 Spectral Pooling

Input: $X \in c \times m_1 \times m_2$ image
Output: $Y \in c \times p_1 \times p_2$
compute FFT of X
truncate X to size $c \times p_1 \times p_2$
return IFFT of X

Spectral pooling is desirable because it can be combined with the convolution theorem to achieve faster results. Specifically, one can combine convolution and pooling by performing the truncation step immediately after the element–wise multiplication in the frequency domain. This is described in the full spectral pooling convolutional layer in Algorithm 2, where k_1 is the total number of images in a batch and k_2 is the number of filters. The CNN can be further optimized by considering the kernels to be in the frequency domain initially. That is, there would be no need to transform the kernels at each pass and the kernels would be updated with the transformed propagated gradient. This thesis does not include this implementation, but it is a theoretically valid method with considerable potential advantages for convolutional layers with larger filter sizes.

Spectral pooling can retain much more information about the image compared to the commonly used pooling operators such as max pooling and average pooling, which reduce dimensionality by applying the max and average operator, respectively, over small sub matrices contained in each image. Spectral pooling also allows much more variability in pooling amount given that the truncation amount does not have to be a factor of the image size. Another pooling method that utilizes a different type of transformation will be presented in the next section.

A few implementation details must be observed to guarantee correct implementation of this method. One must be careful that the FFT shifts the zero-frequency components to the center so that the cropped spectra can be symmetric about the origin. Otherwise, the results will not represent the original data. In order to calculate the gradient in the backwards pass,

Algorithm 2 Spectral Pooling Convolution Layer

Input: $X \in k_1 \times c \times m_1 \times m_2$ image matrix, $F \in k_2 \times c \times l_1 \times l_2$ filter matrix
Output: $Y \in (k_1 \cdot k_2) \times c \times p_1 \times p_2$

```
n1 = l1 + m1 - 1
n2 = l2 + m2 - 1
zero-pad filters and images to ki × c × n1 × n2
for all images i do
    compute and store FFT of i
end for
for all filters f do
    compute FFT of f
    for all images i do
        multiply FFT(i) with FFT(f)
        truncate result to size c × p1 × p2
        store truncated results
    end for
end for
```

the passed gradient from the layer ahead should be zero padded to the original image size (pre–truncation) and then applied the same operator that would be applied in a standard convolutional layer.

2.3.3 Spectral Upsampling

In addition to implementing spectral convolution in the CAE encoder, this thesis presents spectral up-sampling to be used in the decoder. This is the most important contribution given that deconvolutional layers typically struggle to invert the pooling operation as images are up-sampled. Simply put, spectral up-sampling transforms inputs into the frequency domain and then uses these components to create an image of greater, pre-defined dimensions. This is given in Algorithm 3:

Algorithm 3 Spectral Up-sampling

Input: $X \in c$ x p_1 x p_2 image
Output: $Y \in c$ x m_1 x m_2

```
compute FFT of X
return IFFT of spectral X with size c x m1 x m2
```

Intuitively, it can be seen that the back-propagated gradient will be calculated by applying forward spectral pooling on the gradient backwards through the layer. That is, the gradient will

be transformed into the spectral domain and truncated to the number of coefficients required to down-sample into the resulting layer.

Chapter 3

Weight Compression Algorithm

We present a solution to the problem of neuron saturation with the proposed Weight Compression Algorithm, implemented in Levenberg-Marquardt with Weight Compression (LM-WC). LM-WC compress neuron weights by adjusting the output at each neuron to an active region of the activation function. The updated weight vector, $\boldsymbol{w_c}$, can be realized as a linear operator to $\boldsymbol{w}$ in (3.1):

$$\boldsymbol{w_c} = diag(\boldsymbol{c})\boldsymbol{w} \tag{3.1}$$

$\boldsymbol{c}$ is given as (3.2):

$$\boldsymbol{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_1 \\ c_2 \\ \vdots \\ c_2 \\ c_N \\ \vdots \\ c_N \end{bmatrix} \tag{3.2}$$

Each element of c is calculated by (3.3):

$$c_n = \frac{\Omega \cdot w_n \cdot M \cdot P}{\sum_{o=1}^{M} \sum_{p=1}^{P} net_{op}} \tag{3.3}$$

where:

Ω is an adaptable compression parameter

This operator is applied when the network has reached a point of saturation and the learning process is no longer updating network weights. This is determined by calculating the RMSE by (3.4):

$$RMSE = \sqrt{\frac{\sum_{p=1}^{P} (d_p - 0_p)^2}{P}} \tag{3.4}$$

and comparing the change in RMSE to the previous epoch, or (3.5):

$$\Delta RMSE = \frac{RMSE_{\text{k}} - RMSE_{\text{k-1}}}{RMSE_{\text{k-1}}} \tag{3.5}$$

to a threshold, β. This compression forces neurons out of the saturated activation region and closer to, but not in, the linear activation region.

In the case where weight compression pushes the networks activation functions too much or too little, the network can either find itself stuck in the same or worse position in the next compression instance. To avoid this scenario, Ω is adjusted during the training process by an adapting constant ρ normally given as (3.6):

$$\rho \cong 1.1 \tag{3.6}$$

Generally, the improved training algorithm addition can be organized in Algorithm 4. Our methodology is similar to methods that involve normalization of either the inputs or input weights for deep networks trained by first and second-order backpropagation [29, 30]. Our method differs in that it is not an iterative transformation of the inputs or weight parameters; rather, it is a transformation of each neuron's weights to be only applied when convergence has been reached; this is significant in that it can specifically target local optima (solutions reached by pre-converging network of diminished gradient) and adapt its transformation based on success relative to the previous transformation.

Algorithm 4 Weight Compression

```
if ΔRMSE < β then
    if error_prev < error then
        w ← w_prev
        error ← error_prev
        Ω = Ω/ρ
    else
        w_prev ← w
        error_prev ← error
        Ω = Ω · ρ
    end if
    update w by (10)
end if
```

3.1 Choice of WC parameters

The saturation threshold β is chosen in a similar manner to an exit threshold used to finish training for neural networks no longer exhibiting significant weight updates. A successful strategy is to start at a threshold likely to be too large, such as $\beta = 0.1$, and then iterate through smaller β values, decreasing β by a power of 10 each time. Generally, this strategy can be organized in Algorithm 5.

Algorithm 5 β Selection

```
β = 0.1
train neural network
while success improves do
    β = β/10
    train neural network
end while
```

The compression parameter Ω is chosen as a value near but not in the linear activation region of the networks activation function. For the first bipolar networks used later in this thesis, a value of $\Omega = 1$ is used. After a neurons weights are compressed to the Ω value, its net should be in the most effective region of the neuron output, avoiding both the diminishing gradient in the saturation region and high gain in the linear region. This is demonstrated in Figure 3.1.

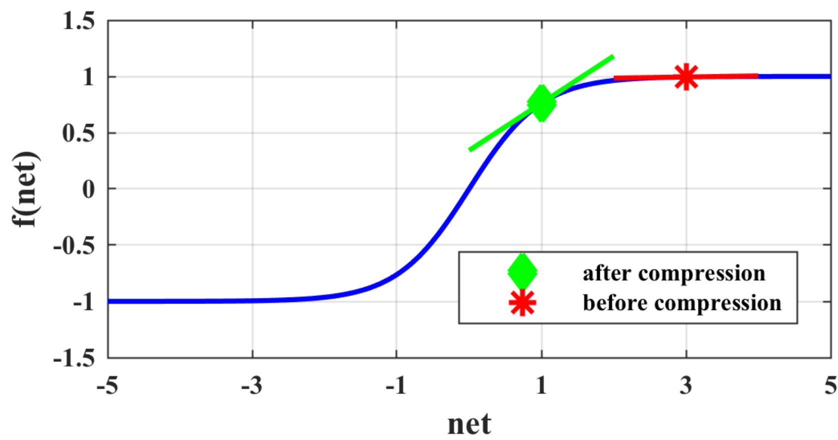


Figure 3.1: Weight compression of bipolar neuron with $\Omega = 1$

3.2 LM Implementation

When implementing weight compression in the LM algorithm, μ must be set to its initial training value when compression occurs, given that it is likely seeing a maximum value when the saturation criterion is met. The network error must be recalculated after weight compression to allow the LM algorithm to correctly implement its next training epoch, which includes an error comparison to the previous epoch.

Chapter 4

Discrete Cosine Transform Spectral Pooling

As an alternative to the spectral pooling algorithm, a DCT Spectral Pooling Layer (DCTSPL) is presented to compare to the Fourier–based spectral pooling layer. The DCT gives a spectral representation of data using only cosine functions instead of complex sinusoids. As a result, only real-valued spectral coefficients must be stored; furthermore, more energy is concentrated in even fewer spectra compared to the Fourier representation. It is because of this that the DCT is commonly used for lossy data compression [31].

DCT has been successfully used in computational intelligence as both a function approximation method [32] and a network compression method [33]. The network compression method has many similarities to this work's approach; however, this work does not implement the method in a layer-based approach to interact with standard CNNs and is not compared to similar pooling methods. Rather than subsampling data to improve network generalization, the work gives a practical CNN implementation to compress large network parameters.

In order to visualize the motivation behind both Fourier (as done in [16]) and cosine-based spectral pooling, the previously discussed pooling methods are compared by applying max pooling, mean pooling, Discrete Fourier Transform (DFT) pooling (implemented with an FFT), and DCT pooling to a 2048 x 2048 sized image of a bird on the gulf coast. Figure 4.1 shows the reconstructed images from the subsampling pooling methods while Figure 4.2 show the spectral representations of the bird image using DFT and DCT, respectively. Notice that the compliment image (reverse white and black) is given so that the spectral density can be easily observed. Figure 4.1 shows that, for varying degrees of pixel subsampling, the spectral pooling techniques greatly outperform the spatial pooling techniques in terms of information preservation. This

can be explained by closely examining Figure 4.2. Observe that most spectral power is focused in the center of the DFT spectrum and in the upper left corner for the DCT spectrum. The significance of this is that the image can be greatly preserved even by truncating many of the spectral components, which is the motivation behind spectral pooling.

Our DCT is implemented by using FFTs, which is slower than performing a single FFT but much faster than the vanilla DCT implementation. A significant drawback for the purpose of this work is that convolution cannot be performed using the DCT unless the filter is both real and symmetric. Therefore, the DCT convolutional layer must separate the convolution and pooling processes. The extra computation cost is not too severe given that the OA method can be used for convolution.

The DCT based convolutional layers is given as Algorithm 6. When truncating the spectral representation for DCT, the first p_i desired indexes are kept (this is different than DFT, as shown in Figure 4.2). Compared to the Fourier–based convolutional layer, the DCT convolutional layer is roughly double the number of operations. However, it is still of the same time complexity, and its information preservation makes up for what it loses in training time.

Algorithm 6 DCT Spectral Pooling Convolution Layer

Input: $X \in k_1 \times c \times m_1 \times m_2$ image matrix, $F \in k_2 \times c \times l_1 \times l_2$ filter matrix
Output: $Y \in (k_1 \cdot k_2) \times c \times p_1 \times p_2$

```
n_1 = l_1 + m_1 - 1
n_2 = l_2 + m_2 - 1
zero-pad filters and images to k_i × c × n_1 × n_2
for all filters f do
   for all images i do
      convolve i with f using OA method
      compute DCT
      truncate result to size c × p_1 × p_2
      store truncated results
   end for
end for
```

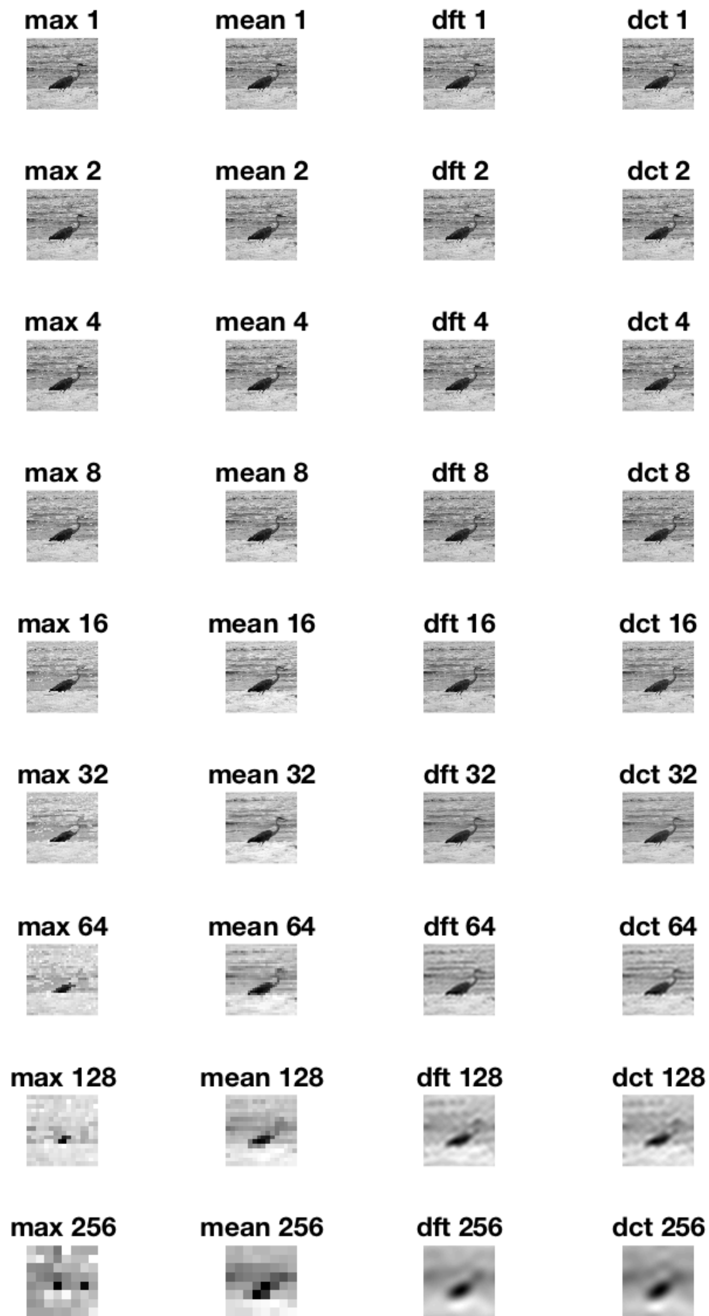
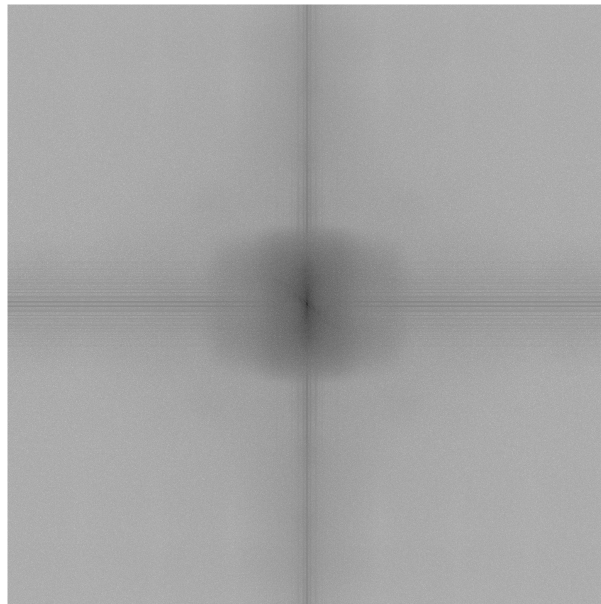
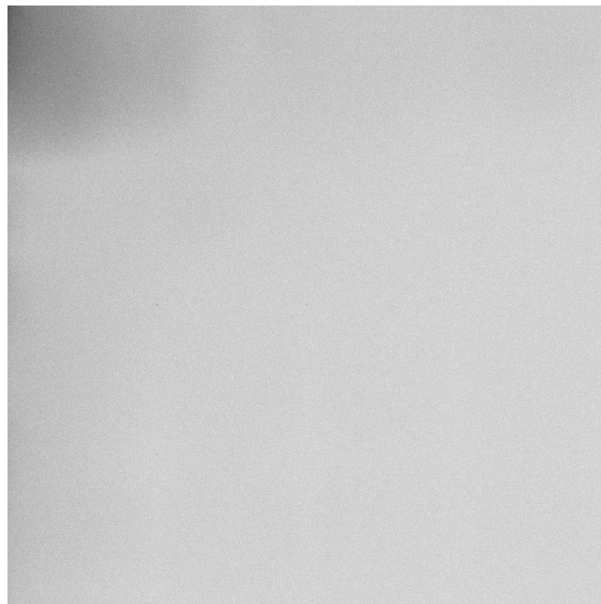


Figure 4.1: Image subsampling for image of bird using max pooling (first row), mean pooling (second row), DFT (third row) DCT (fourth row)



(a) DFT



(b) DCT

Figure 4.2: Spectral representation of bird image using (a) Discrete Fourier Transform and (b) Discrete Cosine Transfrom

Chapter 5

Experiment Results

5.1 Weight Compression for LM Regression

In this section, four experiments are analyzed to demonstrate the effectiveness of weight compression implemented in LM-WC. The experiments compare the effectiveness of LM-WC against the original State-of-the-Art LM algorithm and LM algorithm with random restarts.

5.1.1 Experiment Setups

Experiments will be compared on the following criterion: success rate for a determined success threshold of RMSE, average number of epochs per successful experiment trial, and average training time of successful experiment trials. Specifically, algorithms with a higher success rate and lower average epochs are considered the most successful, but only if the cost of additional training time is insignificant. In these experiments, the LM and LM-WC algorithms are implemented with MATLAB using only standard MATLAB statements.

Each experiment is performed with both the standard MLP network and an FCC network to demonstrate the algorithm's ability to enhance performance on a variety of architectures. The network sizes are found by finding the smallest network size (for both MLP and FCC) that succeeds at a given error threshold. Because of these, results from each network size can be thought of as separate experiments.

For the first experiment, it was experimentally determined that the optimal value of the compression parameter Ω and the adapting constant ρ should be $\Omega = 1$ and $\rho = 1.1$. For the remaining experiments, we use $\Omega = 0.1$ because the data set is of high dimensionality and

contains more training patterns relative to the first three datasets. Experiments were conducted with varying values of the saturation threshold β, consistent with the method described in Algorithm 5; results with the optimal β value are presented. μ is initialized to be 10^{-3} and is updated with a classic Levenberg update method with maximum and minimum values of 10^{-15} and 10^{15}, respectfully.

In order to demonstrate that LM-WC is more powerful than randomly resetting the weights, we compare LM-WC to LM Random Restart (LM-RR) , which follows the same algorithm as Algorithm 4, but instead randomly initializes new weights when non-optimal convergence occurs. As discussed earlier, the usage of random restarts is a common technique to improve neural network convergence by escaping local optima; our algorithm performs better than random restarts in every case because it retains useful information gained in the failed training attempt.

5.1.2 Further Considerations

When analyzing results, we must consider potential effects of overfitting. Because these experiments test the capabilities of successfully training to a target error, validation and testing data are not included. We are not wanting to show effects of over-training datasets; rather, we want to demonstrate that, for this given architecture and second-order algorithm, we can successfully train to a low target error. Because these experiments consist of training difficult, low-succeeding networks, the method of using validation and testing data would potentially present results in which what we deem as successful (low training error) is seen as less to what we deem as unsuccessful (high training error) due to the effects of overfitting. Experiments in the next section will investigate training vs. testing RMSE.

It is important to realize that each network is trained to a specific target error and finishes training after reaching the desired value (success) or the maximum iterations (failure). Only successful trials contribute to the reported training epoch and training time values. Because they do not consider the failing cases, these two criterion may be lower for the classic LM algorithm compared to LM-WC.

5.1.3 Results on benchmark datasets

For the following experiments, desired error is set to be 0.1. For the first experiment, the training algorithms are allowed to run for a maximum of 100 epochs for 100 trials each. 1000 iterations are allowed on the remaining experiments because of higher data dimensionality and larger size. The data is normalized to have zero mean and be contained in the range [-1 1]. In each trial, weights are randomly initialized in the range [-1 1], excluding zero.

The first experiment is conducted on the previously discussed parity-7 problem. There is a total of 128 training patterns for this dataset. The first architecture compared is a MLP architecture containing two hidden layers of two neurons each and a linear output neuron. The second architecture compared is an FCC architecture composed of two hidden layer neurons and a linear output neuron. Results, in Table 5.1, demonstrate the effectiveness of the LM-WC algorithm over LM and LM-RR.

The second experiment is conducted on a benchmark dataset known as the two-spiral problem, shown in Figure 5.1. The Spirals dataset contains output classes of either -1 or 1, designated as xs and os. The first architecture compared is a MLP architecture containing two hidden layers of eight neurons each and a linear output neuron. The second architecture compared is an FCC architecture composed of nine hidden layer neurons and a linear output neuron. The results of the experiment are shown in Table 5.2; Figure 5.2 shows the resulting output. Normally, an output of this accuracy would require 100 trials on average to achieve; the Weight Compression Algorithm allows us to achieve these results with only 5 trials on average, a significant improvement. Similar to the parity problem, the performance of LM-WC is best.

The third and fourth experiments are conducted on regression datasets from the UCI Machine Learning Repository [34]. The first dataset, Housing, contains data from the Boston Housing Market and predicts median value of owner-occupied homes based on various housing attributes [35]. The dataset contains 506 patterns of thirteen attributes. The first architecture compared is a MLP architecture containing two hidden layers of three neurons each and a linear output neuron. The second architecture compared is an FCC architecture composed of three

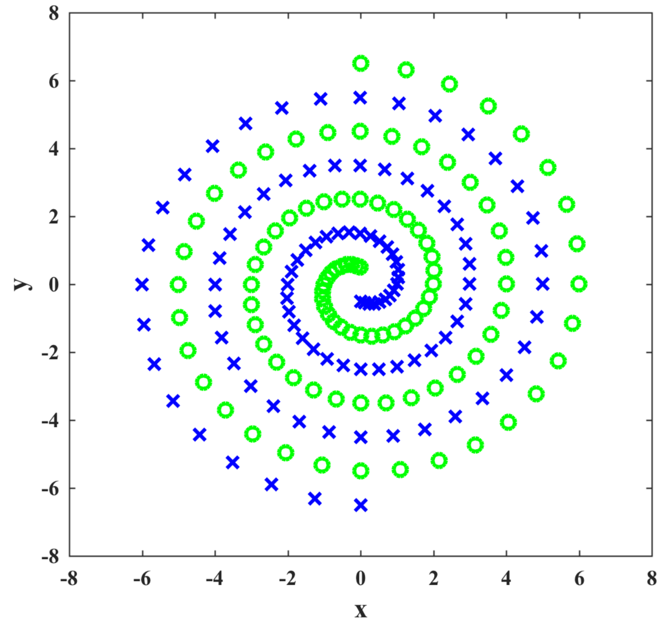


Figure 5.1: Spirals dataset

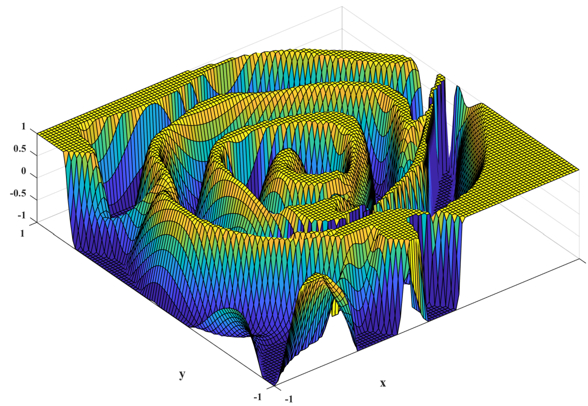


Figure 5.2: Spirals results with LM-WC

hidden layer neurons and a linear output neuron. The results in Table 5.3 demonstrate strong performance of LM-WC.

The fourth dataset, Concrete, approximates concrete compressive strength from eight quantitative input variables [36]. There is a total of 1030 instances for this dataset. The first architecture compared is a MLP architecture containing two hidden layers of six neurons each and a linear output neuron. The second architecture compared is an FCC architecture composed of seven hidden layer neurons and a linear output neuron. Results in Table 5.4 once again favor the LM-WC algorithm.

To summarize the results, the LM-WC algorithm is able to reach a desired training error at a much higher success rate compared to classic LM and LM with random restarts. The results show a greater time to achieve these error rates for our LM-WC algorithm, but this time does not consider the frequently occurring failing cases in the LM and LM-RR results.

5.1.4 Additional Experiments

In order to evaluate LM-WC in a classical approach, additional experiments using the iris flower and abalone datasets (also retrieved from the UCI Machine Learning Repository) are presented to compare LM-WC against LM (random restarts are left out because they were shown to be no better than LM in the previous experiments). The iris dataset [37] is used frequently as a benchmark dataset and contains 3 classes of 50 instances each (4 attributes). The abalone dataset [38] contains 4177 instances of 8 attributes predicting the age of abalone. Both experiments use the same parameters as the previous experiments, but the data is split into training and testing data (70/30 split) and evaluated for RMSE after 100 training iterations. The results of these experiments for various architectures (found using the same methodology as the first four experiments) are given in Tables 5.5 and 5.6. Using this evaluation method, LM-WC is able to achieve lower training and testing RMSE with similar training times.

5.2 DCT Pooling for CNN Classification

In order to compare the presented implementations of spectral convolution and pooling, experiments were conducted on both the original (experiment 1) and a modified version (experiment

Table 5.1: Training Results for Parity-7 Dataset

Algorithm	β	SR (%)	Training Epochs	Training Time (s)	Hidden Neurons	Hidden Layers	Network Type
LM	~	2.00	82.00	0.2355	4	2	MLP
LM-RR	1E-5	4.00	71.50	0.2355	4	2	MLP
LM-WC	1E-5	14.00	81.50	0.2426	4	2	MLP
LM	~	2.00	18.00	0.0468	2	~	FCC
LM-RR	1E-3	7.00	51.57	0.1332	2	~	FCC
LM-WC	1E-3	26.00	55.19	0.1448	2	~	FCC

Table 5.2: Training Results for Spirals Dataset

Algorithm	β	SR (%)	Training Epochs	Training Time (s)	Hidden Neurons	Hidden Layers	Network Type
LM	~	12.00	447.67	5.5140	16	2	MLP
LM-RR	1E-5	12.00	384.58	3.9582	16	2	MLP
LM-WC	1E-5	24.00	447.58	4.0723	16	2	MLP
LM	~	1.00	293.00	2.0171	9	~	FCC
LM-RR	1E-4	1.00	103.00	2.7262	9	~	FCC
LM-WC	1E-4	19.00	627.81	4.5166	9	~	FCC

Table 5.3: Training Results for Housing Dataset

Algorithm	β	SR (%)	Training Epochs	Training Time (s)	Hidden Neurons	Hidden Layers	Network Type
LM	~	10.00	81.30	1.3106	6	2	MLP
LM-RR	1E-5	12.00	234.67	3.2774	6	2	MLP
LM-WC	1E-5	52.00	434.5577	6.5499	6	2	MLP
LM	~	1.00	57.00	0.9465	3	~	FCC
LM-RR	1E-3	2.00	181.50	2.5092	3	~	FCC
LM-WC	1E-3	16.00	522.69	7.2620	3	~	FCC

Table 5.4: Training Results for Concrete Dataset

Algorithm	β	SR (%)	Training Epochs	Training Time (s)	Hidden Neurons	Hidden Layers	Network Type
LM	~	49.00	142.14	7.6002	12	2	MLP
LM-RR	1E-6	50.00	134.50	9.4457	12	2	MLP
LM-WC	1E-6	59.00	222.1356	12.1849	12	2	MLP
LM	~	5.00	114.80	5.0420	7	~	FCC
LM-RR	1E-4	4.00	259.00	11.5835	7	~	FCC
LM-WC	1E-4	34.00	522.32	24.3557	7	~	FCC

Table 5.5: Training Results for Iris Dataset

Algorithm	β	Training RMSE	Testing RMSE	Training Time (s)	Hidden Neurons	Hidden Layers	Network Type
LM	˜	0.0845	0.1435	0.3880	4	2	MLP
LM-WC	1E-4	0.0819	0.1348	0.3942	4	2	MLP
LM	˜	0.1248	0.1479	0.3440	3	˜	FCC
LM-WC	1E-4	0.1057	0.1183	0.3327	3	˜	FCC

Table 5.6: Training Results for Abalone Dataset

Algorithm	β	Training RMSE	Testing RMSE	Training Time (s)	Hidden Neurons	Hidden Layers	Network Type
LM	˜	0.1526	0.1541	5.5032	4	2	MLP
LM-WC	1E-5	0.1513	0.1517	5.3944	4	2	MLP
LM	˜	0.1547	0.1546	6.2234	3	˜	FCC
LM-WC	1E-4	0.1505	0.1481	6.2234	3	˜	FCC

2) of the MNIST benchmark classification dataset [39]. The original MNIST dataset is composed of 70,000 hand–drawn grey–scale digits of 28 x 28 pixels in 10 classes ranging from 0 to 9 (60,000 are used for training and 10,000 are used for testing). The second experiment stack 16 identical digits in the same image to create images of 112 x 112 pixels. The purpose of this is to effectively demonstrate the power of spectral–based techniques because, as previously mentioned, the advantage grows exponentially with image size (with an upfront additional computation cost that outweighs the benefit for small image size).

The CNN was implemented with standard MATLAB statements in order to facilitate a rapid development and visualization process. A vanilla CNN library was developed from scratch to enable a fair comparison of new layers (i.e., to not compare a vanilla custom layer to a highly optimized layer from a language such as Tensorflow). The spectral convolution and pooling layers were developed based on the given algorithms, using optimized MATLAB functions when available. A fully–working network trainer software was built on top of the library and used for these experiments.

It is important to observe that this preliminary work is implemented with basic MATLAB statements and not all processes are optimized; therefore, the important measurement for this work is classification rate. Many operations in the spectral layers include computationally

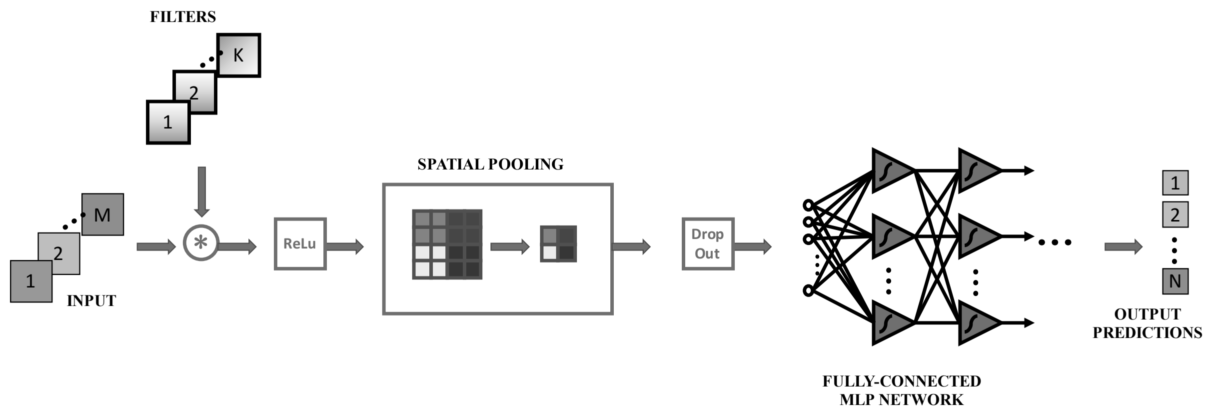
Table 5.7: Training Results for Various Pooling Methods

Pooling	Stride	Size	Truncation (%)	Reduction (%)	Training Time (s)	Classifaction Accuracy (%)
Max	2	2	~	75	182.3	79.0
Mean	2	2	~	75	199.5	78.0
FFT	~	~	75	75	290.6	78.6
DCT	~	~	75	75	10002.2	83.8
Max	4	4	~	93.75	4236.5	70.4
Mean	4	4	~	93.75	4251.1	72.7
FFT	~	~	93.75	93.75	2438.5	73.4
DCT	~	~	93.75	93.75	4056.5	82.5

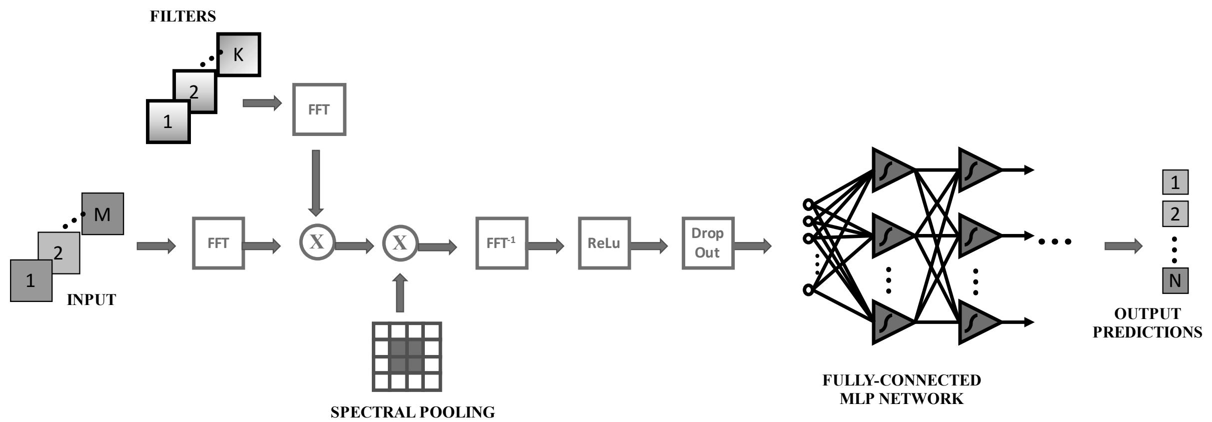
expensive loops that will be avoided in a future tensor-based implementation. The purpose of these experiments is to show the powerful classification abilities of these layers, given that the theoretical speedup factors have already been presented.

All experiments were implemented with the networks in Figure 5.3 with experiment variations being confined to the convolution and pooling layers. The network was trained with the stochastic gradient decent algorithm (batch size = 256) with a momentum rate of 0.9. A probability of 0.05% was used in the dropout layers. The convolutional layer contained 32 kernels of size 3 x 3. For each experiment trial, the network is trained on 50 epochs of training data and then evaluated on testing data; final results are averaged from ten trials total. It is important to note that hyper-parameters for these experiments were tuned with simple grid searches for each strategy, but this tuning is not presented in these results because it is irrelevant to the purpose of this thesis. Rather, these experiments are to compare spectral techniques on a network holding all other settings the same.

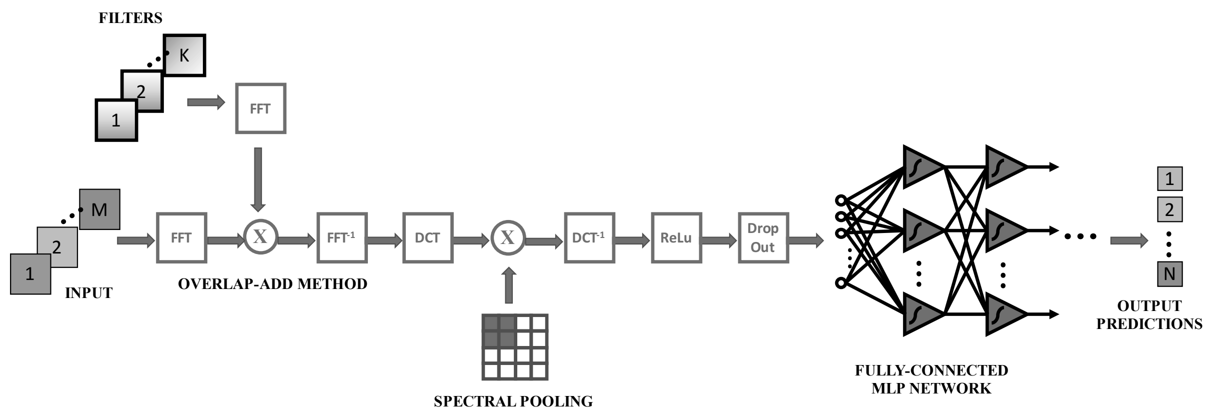
The first experiment uses pooling techniques that result in data reduction of 75% (i.e., the number of elements being produced by the pooling layer is 75% of the number of elements being fed into the layer) and the second experiment uses pooling techniques that result in data reduction of 93.75%. Note that the networks are identical if the reduction is 0%. Both experiments were evaluated on classification accuracy and training time. The layer parameters and results are given in Table 5.7 while the learning curves are given in Figures 5.4 - 5.7.



(a) Spatial Pooling

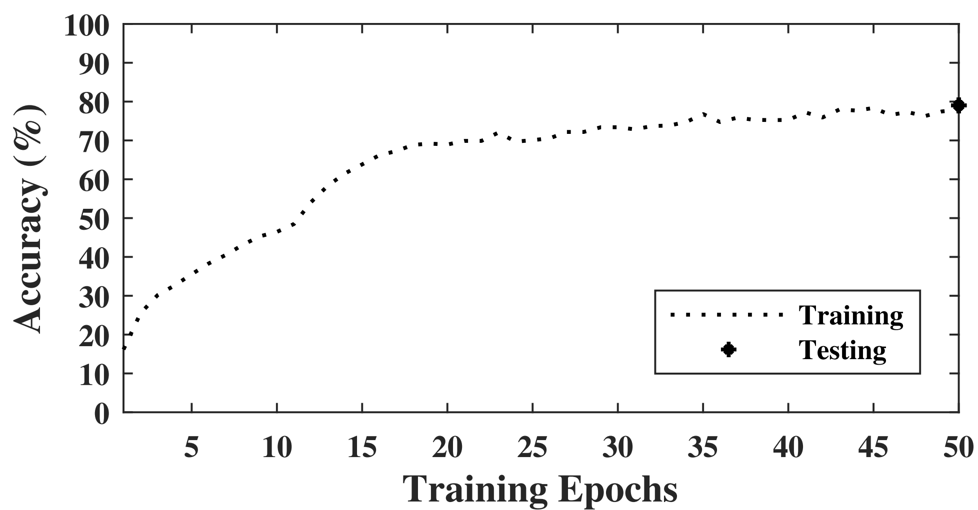


(b) DFT Spectral Pooling



(c) DCT Spectral Pooling

Figure 5.3: Convolution Neural Network Architecture Implementation for Various Pooling Methods



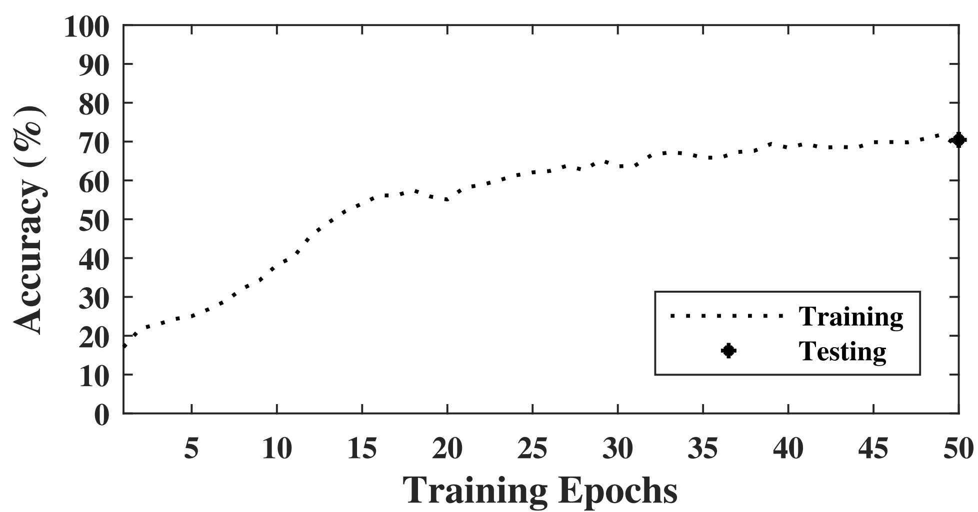
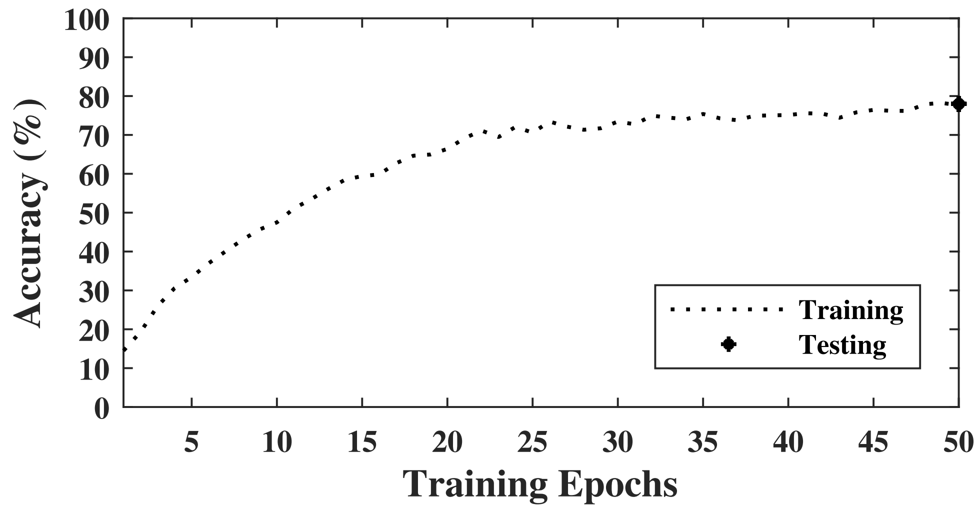


Figure 5.4: Training curves with max pooling for experiment 1 (top) and experiment 2 (bottom)



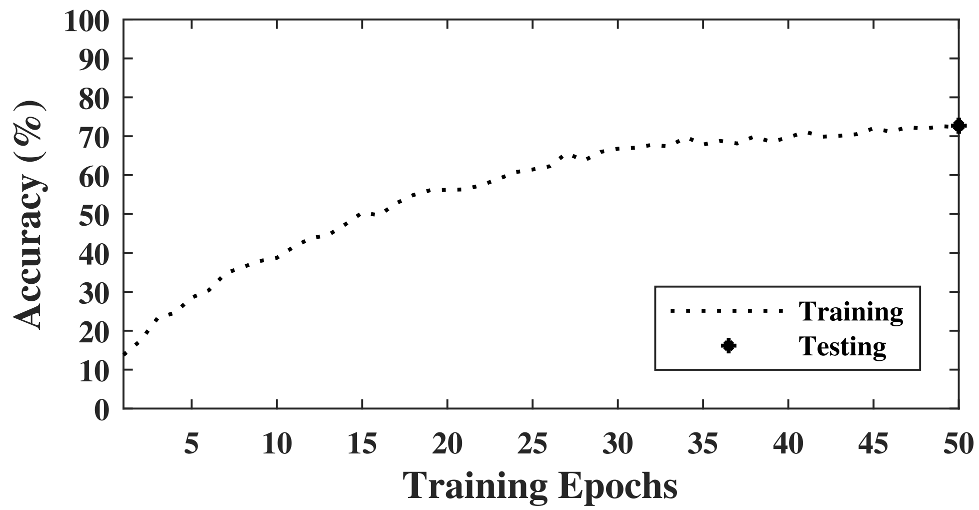
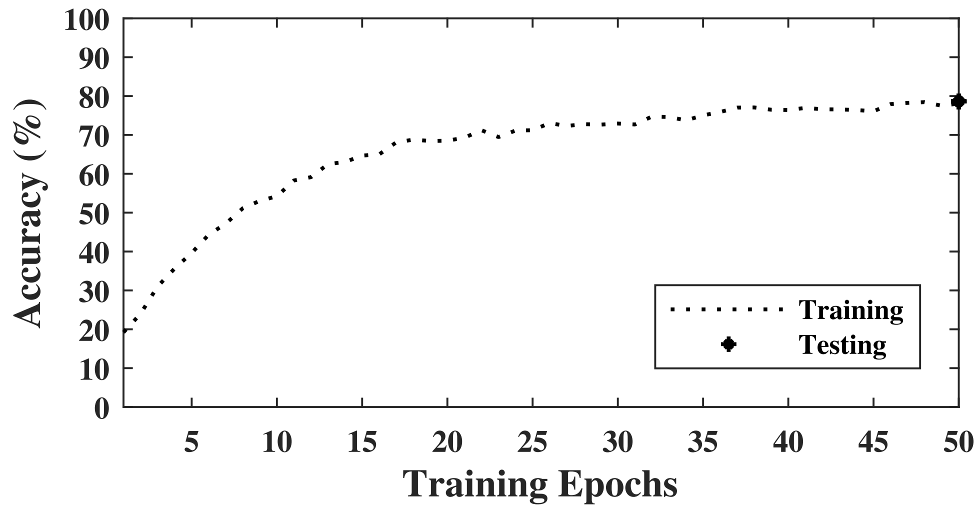


Figure 5.5: Training curves with mean pooling for experiment 1 (top) and experiment 2 (bottom)



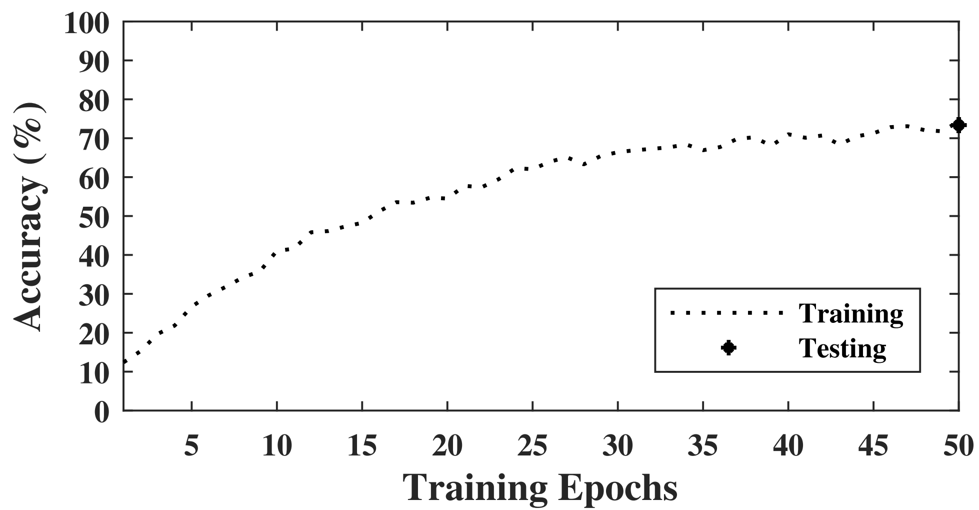
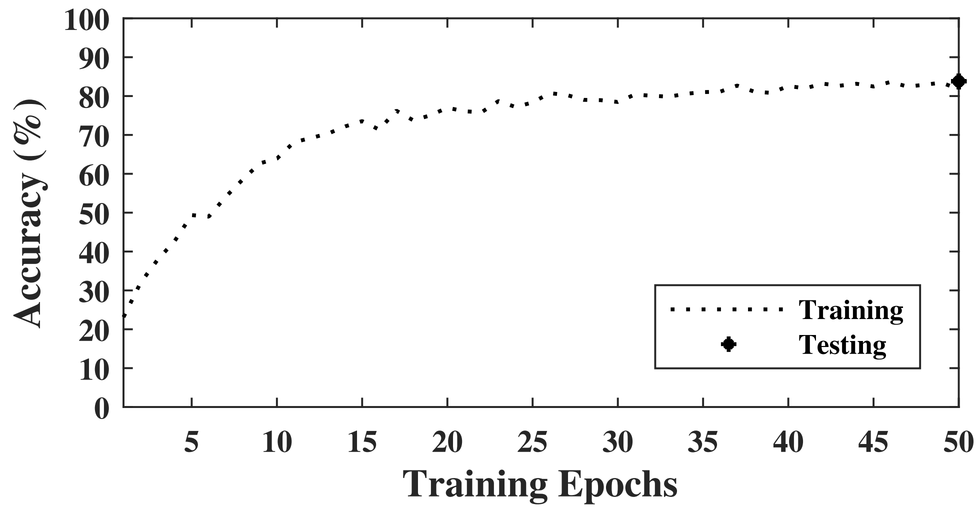


Figure 5.6: Training curves with FFT spectral pooling for experiment 1 (top) and experiment 2 (bottom)



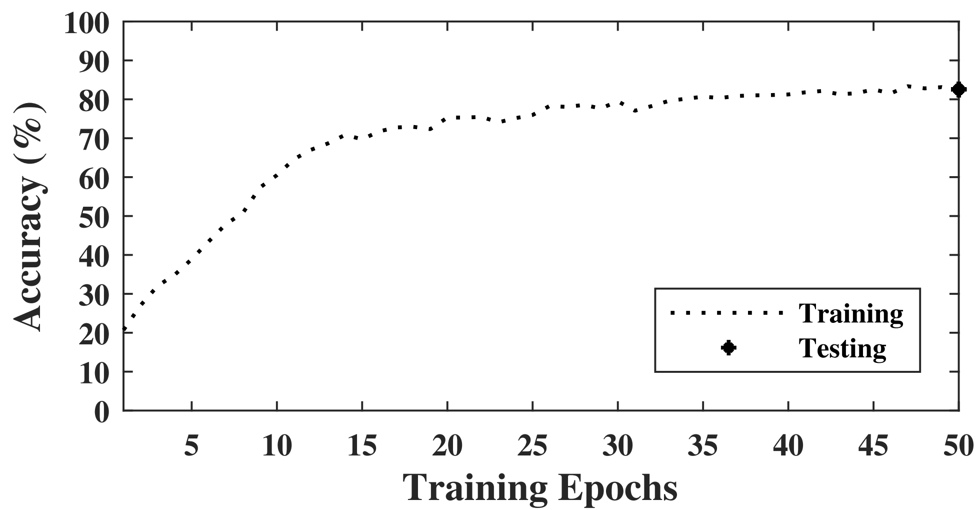


Figure 5.7: Training curves with DCT spectral pooling for experiment 1 (top) and experiment 2 (bottom)

When comparing results, the FFT spectral pooling is the fastest method as the problem size scales up (experiment 2) while the DCT is the most accurate method. The DCT is able to converge faster and to a higher classification rate than the other methods, especially in the second experiment. For this size data, the training times of the spatial and spectral networks are comparable; however, as discussed earlier in Figures 2.9 and 2.10, the training time of the spatial networks will grow exponentially compared to that of the spectral networks. Furthermore, it was earlier shown that the DCT network will take longer than the FFT network but will grow at a linear rate in comparison. This means that the high relative training time is not a major concern; rather, the high classification rate for large layer reduction is a significant achievement.

5.3 Spectral Up-Sampling for CAE Compression

Finally, spectral pooling and up-sampling are implemented in a CAE to demonstrate the capability of using these spectral methods for lossy compression networks. A basic network shown in Figure 5.8 is created and tested for two pooling methods, max and spectral pooling (SP), as well as three up-sampling methods, nearest-neighbor (NN), bilinear (BL), and spectral up-sampling (SU), giving six different architectures in total.

For this experiment, the images are embedded into a vectors of size 10 and 30 (compared to 784 features in the input and output images). Each network is trained using the SOTA Adam optimization method (batch size = 64) [40] rather than stochastic gradient descent due to the difficulty in training an autoencoder (compared to a classic CNN). Similar to the previous experiment, loss and training times were averaged over ten separate trials and given in Table 5.8. These experiments use pixel-by-pixel mean squared error and thus cannot estimate the probability distribution of the training data, resulting in blurry pictures ion many cases. However, these proof of concept experiments show the powers of spectral pooling for both encoders and decoders, demonstrating the need for these methods to be implemented in SOTA Variational Autoencoders [41] and Generative Adversarial Networks [42] to generate images of similar quality compared to the training dataset.

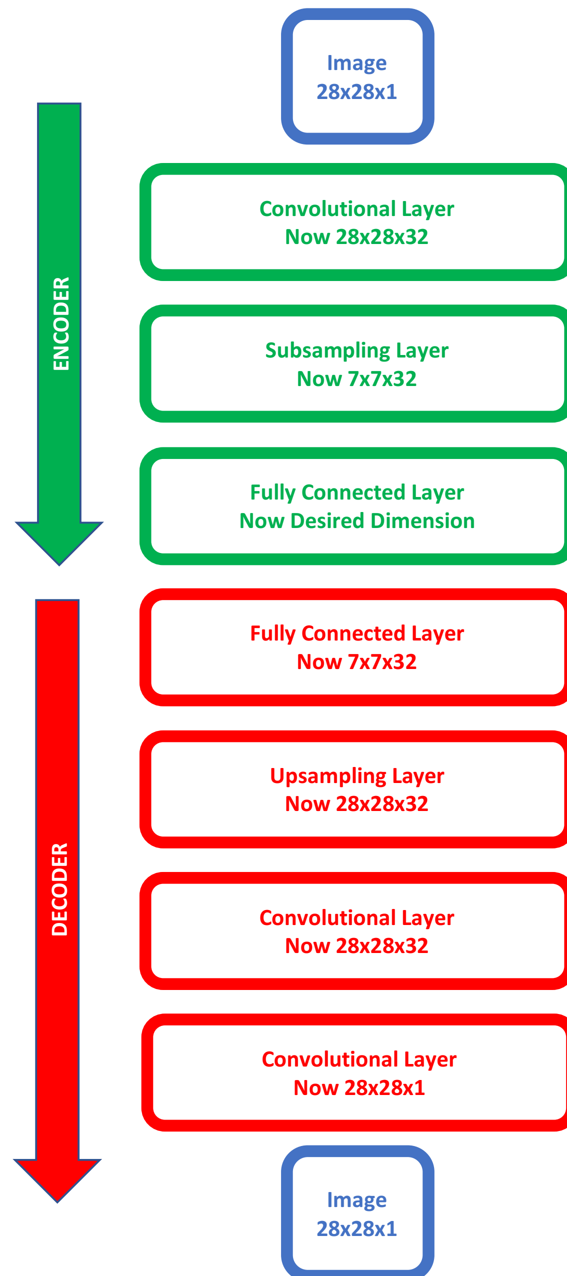


Figure 5.8: CAE network structure

Table 5.8: Training Results for Various CAE Networks

Pooling	Upsampling	Dim Middle	Training Time (s)	Validation Error	Testing Error
Max	NN	10	3114.58	0.2253	0.2293
Max	BL	10	3195.52	0.1683	0.1730
Max	SU	10	7947.34	0.1805	0.1822
SP	NN	10	5840.06	0.1481	0.1504
SP	BL	10	5506.64	0.1334	0.1360
SP	SU	10	9457.59	0.1734	0.1771
Max	NN	30	2684.74	0.1382	0.1413
Max	BL	30	2859.32	0.0907	0.0913
Max	SU	30	7222.52	0.1118	0.1133
SP	NN	30	5821.34	0.1021	0.1034
SP	BL	30	6972.01	0.0923	0.0937
SP	SU	30	11248.88	0.0993	0.1003

When comparing results, it is clear that spectral pooling technique reduces training loss. However, spectral upsampling is not necessarily as successful as the nearest-neighbor and bilinear transform methods. When examining the generated images in Figures 5.9–5.21, it is clear that the mean squared error may not necessarily correlate to what the viewer deems as the greatest recovery method. All of the digits look readable, save the spectral upsampling. The results suggest that spectral upsampling in the decoder is not as effective as the other interpolation methods, at least when training on mean squared error; however, spectral pooling in the encoding outperforms max pooling.

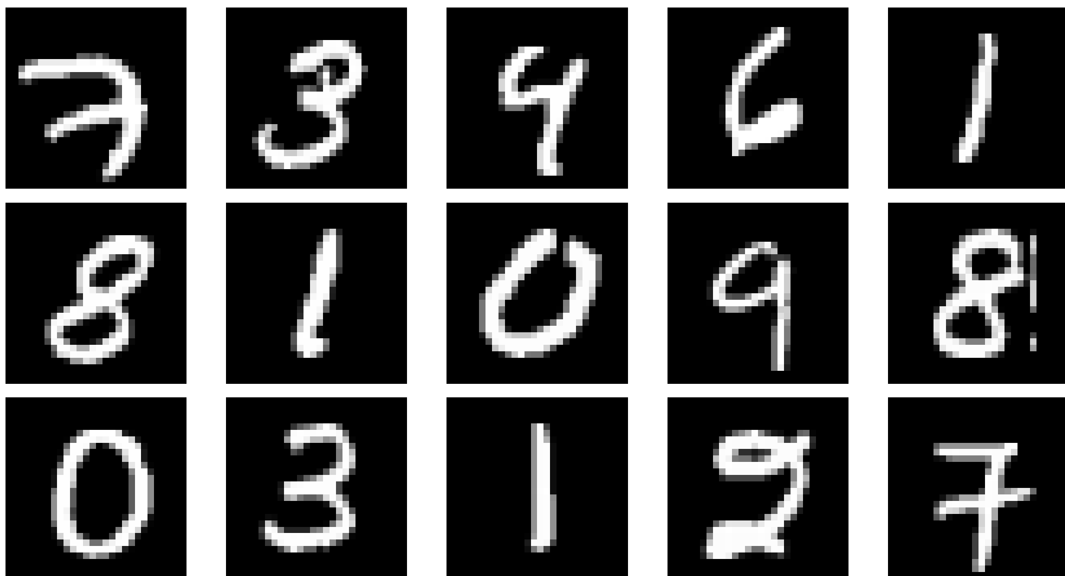


Figure 5.9: Original MNIST digits

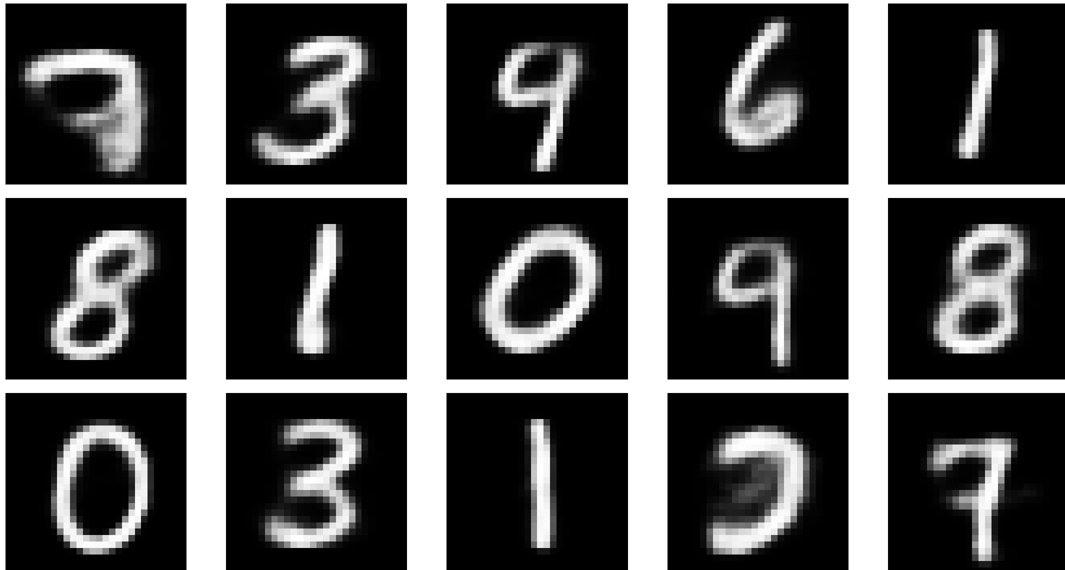


Figure 5.10: MNIST digits using Max–NN and middle dimension 10

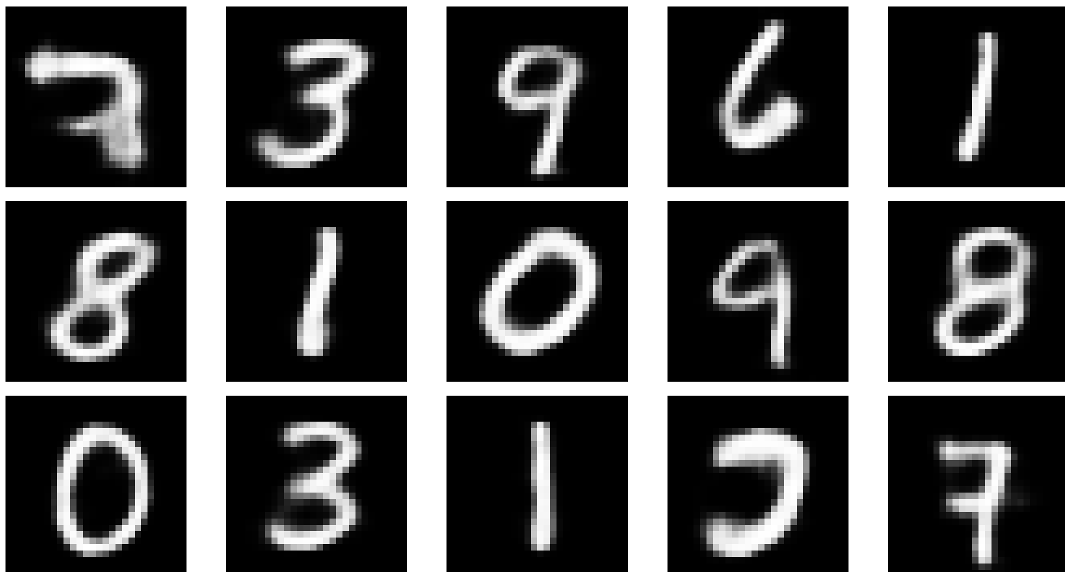


Figure 5.11: MNIST digits using Max–BL and middle dimension 10

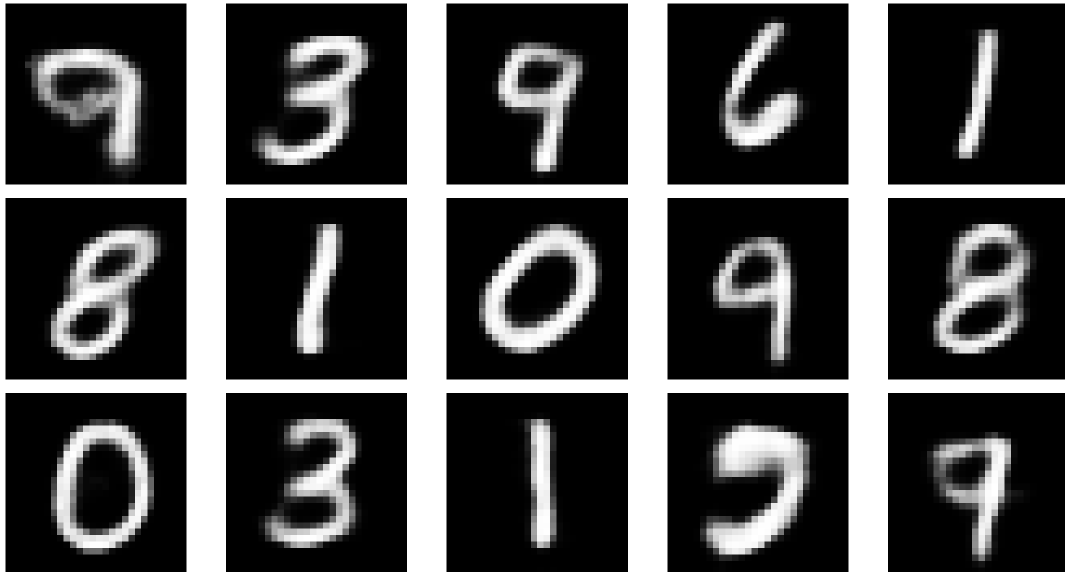


Figure 5.12: MNIST digits using Max–SU and middle dimension 10

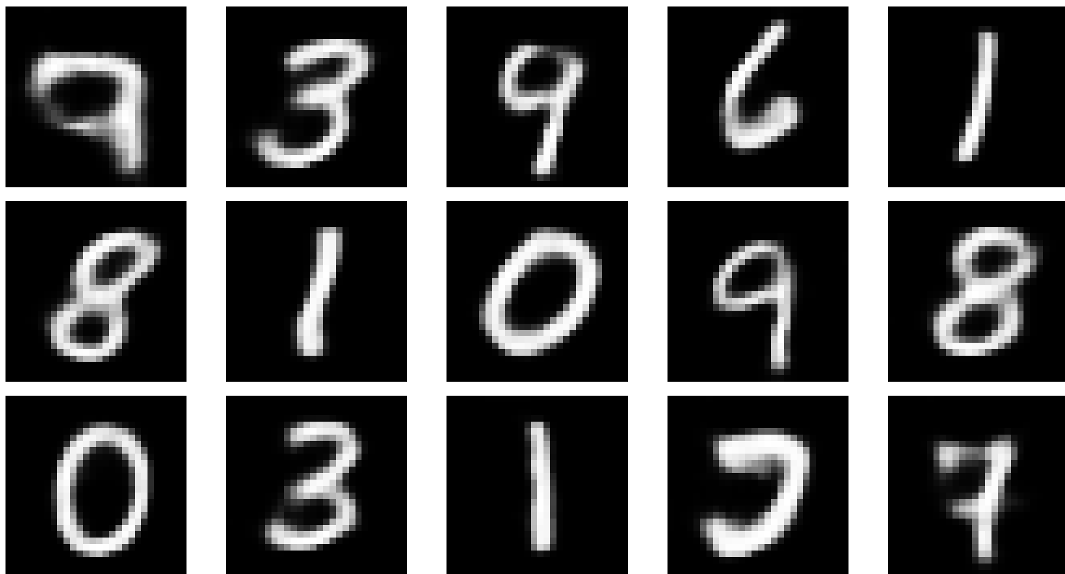


Figure 5.13: MNIST digits using SP–NN and middle dimension 10

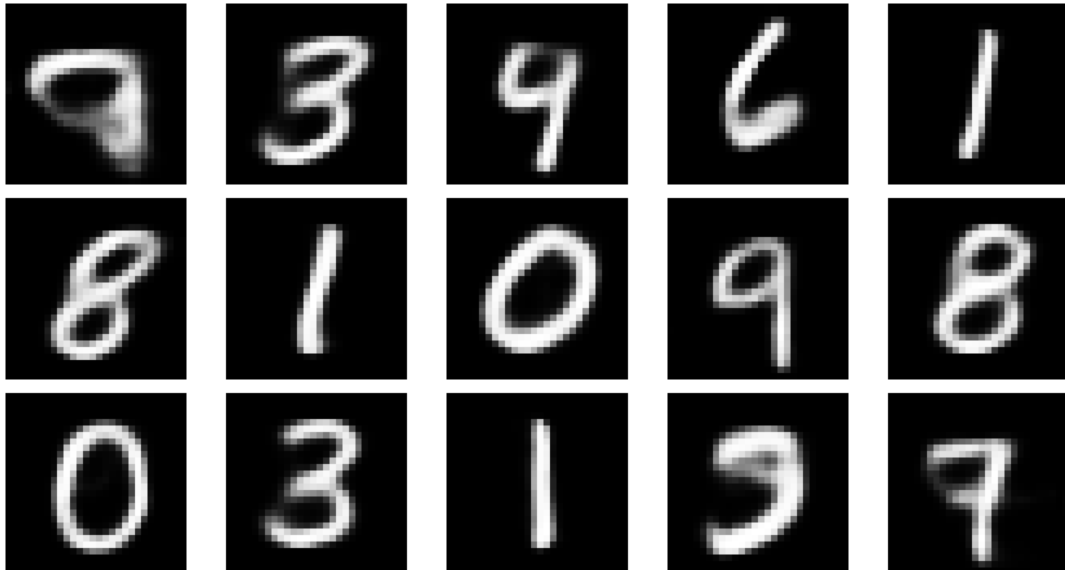


Figure 5.14: MNIST digits using SP–BL and middle dimension 10

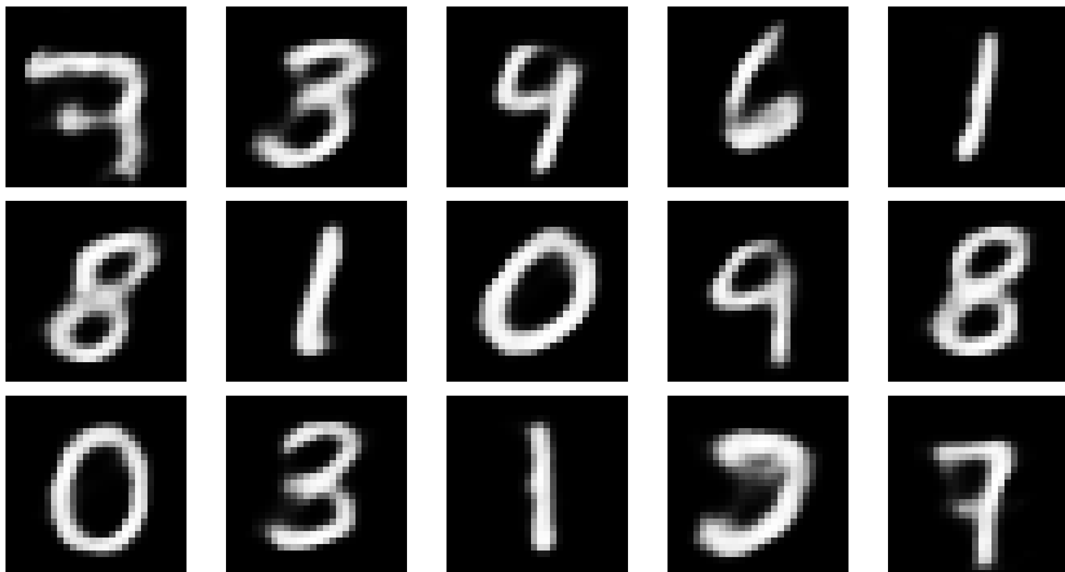


Figure 5.15: MNIST digits using SP–SU and middle dimension 10

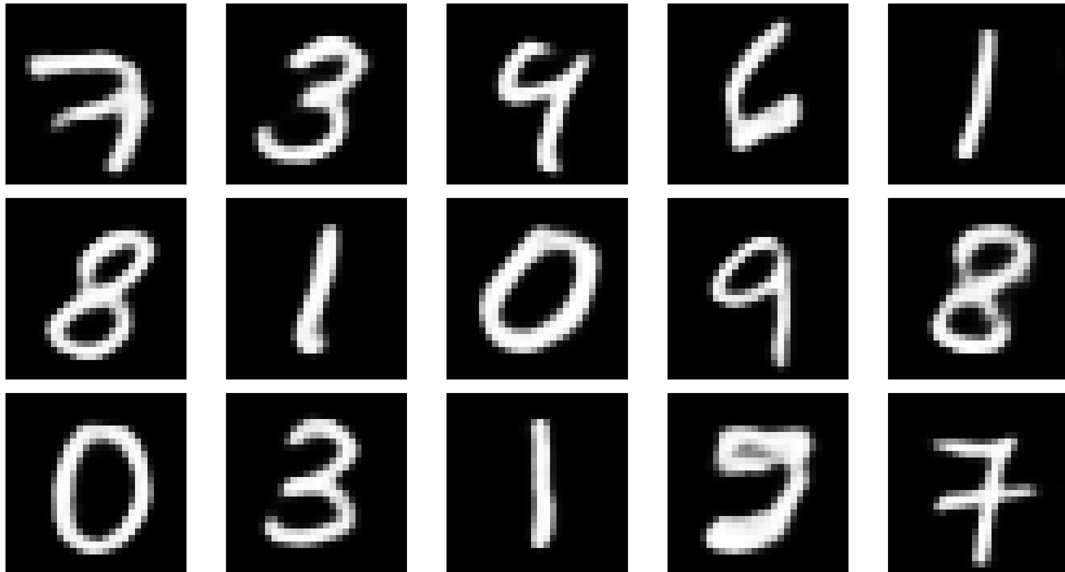


Figure 5.16: MNIST digits using Max–NN and middle dimension 50

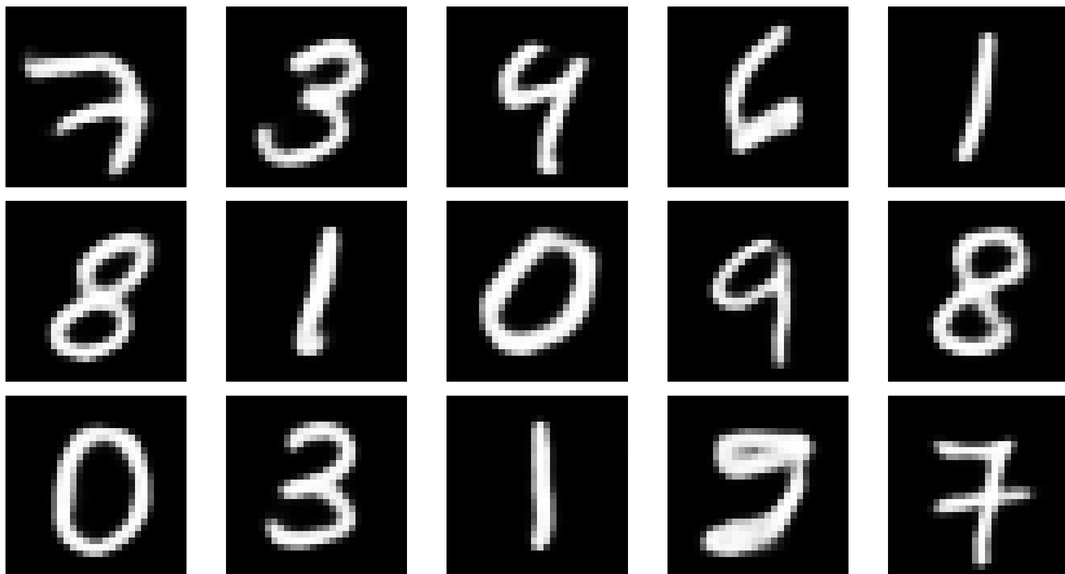


Figure 5.17: MNIST digits using Max–BL and middle dimension 50

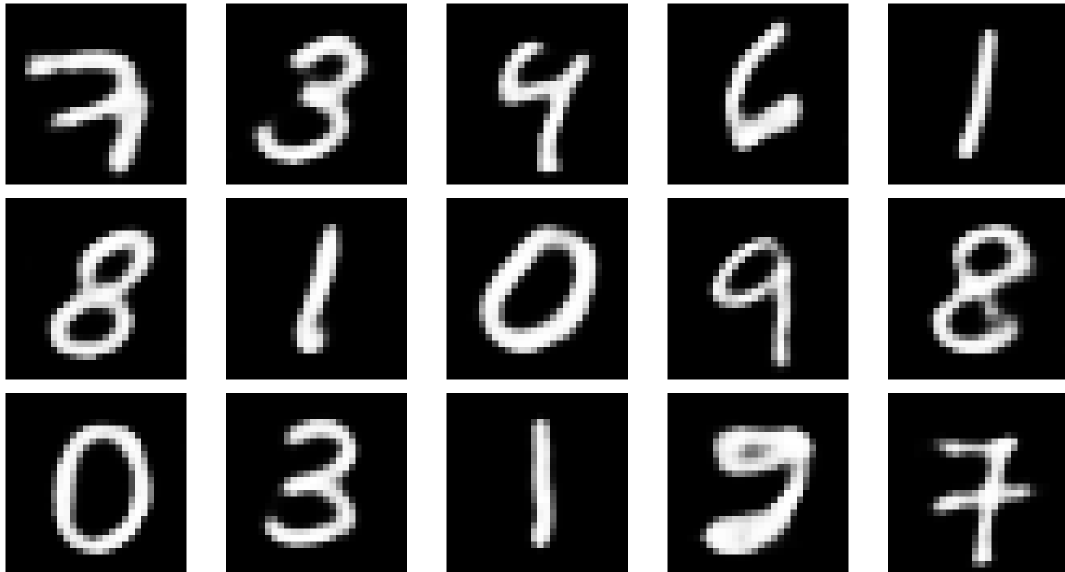


Figure 5.18: MNIST digits using Max–SU and middle dimension 50

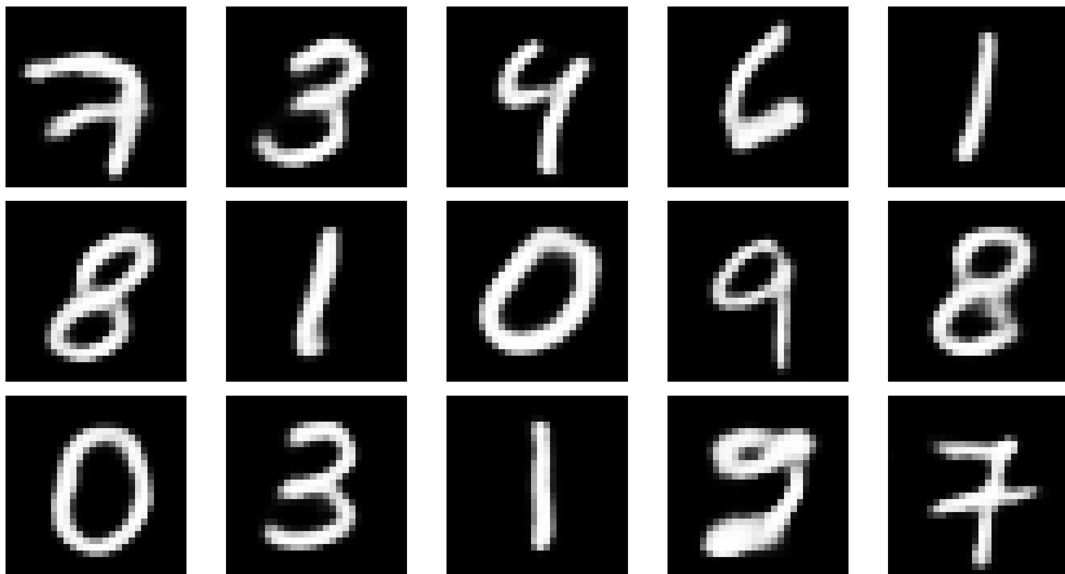


Figure 5.19: MNIST digits using SP–NN and middle dimension 50

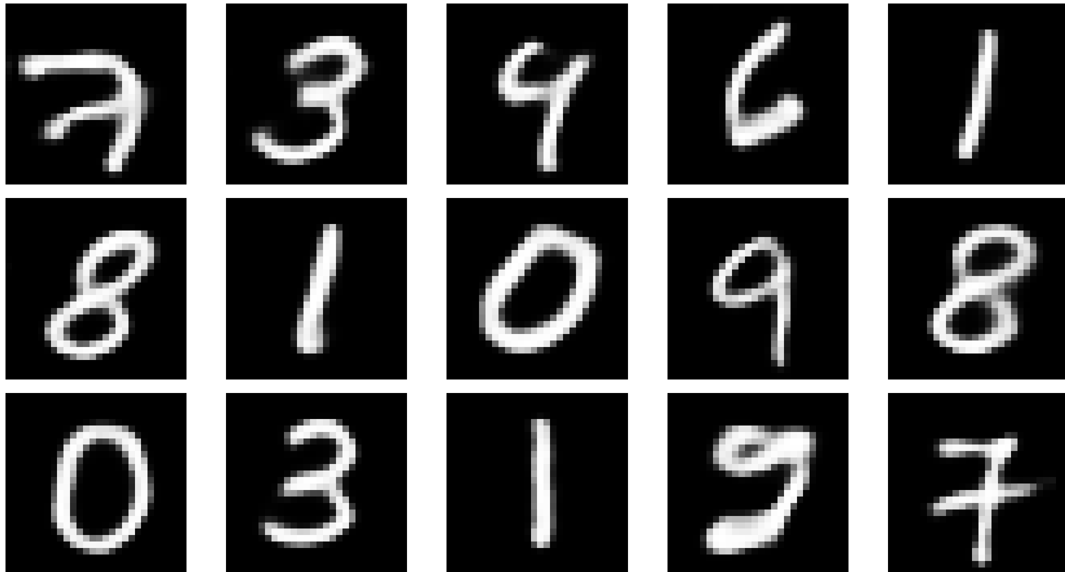


Figure 5.20: MNIST digits using SP–BL and middle dimension 50

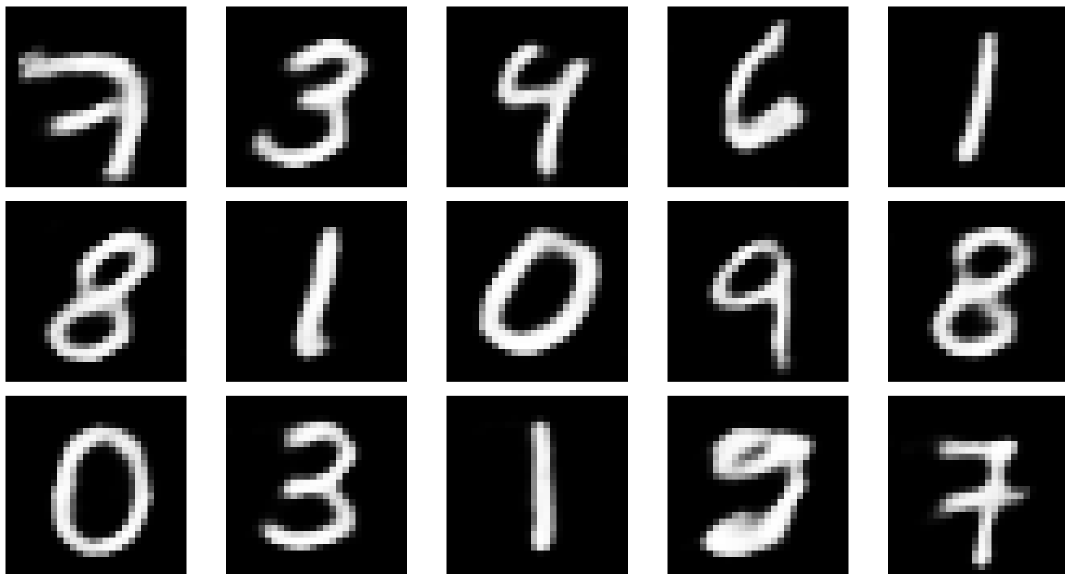


Figure 5.21: MNIST digits using SP–SU and middle dimension 50

Chapter 6

Final Remarks

This work presented practical solutions to two ANN problems focused on designing network architectures capable of being implemented for practical usage without expensive hardware and excessive training time. We first proposed a new Weight Compression (WC) Algorithm, implemented in LM-WC, to avoid the flat-spot problem. The advantage of the WC algorithm is the ability to find solutions for difficult problems characterized by low rates of success that may not be feasible to explore with normally used second-order algorithms, allowing the practical usage of compact neural network architectures capable of high memory efficiency and great generalization abilities.

Performance of the proposed algorithm was compared with similar algorithms for several benchmark problems using both FCC and MLP neural network architectures. LM-WC improved difficult problems characterized by low training success, especially for deeper network architectures such as the FCC network. This thesis presented instances where the LM-WC algorithm improved problems that would take 100 trials on average to succeed, finding solutions in as little as 5 trials.

The LM-WC algorithm has transformative implications for training deep neural networks using fewer neurons, thus reducing the number of parameters to be learned and retaining greater generalization abilities. Our results demonstrated the ability to find compact solutions to high-failure problems in what should be considered a practical number of attempts for even a computationally expensive deep learning problem.

In addition to the LM-WC algorithm, this thesis presented a discrete cosine transform spectral pooling layer (DCTSPL) to be used for subsampling large layer data while retaining

maximum information. Convolution times for practical image and filter size combinations were investigated to conclude that spectral convolution outperforms spatial convolution, even for very small filter sizes. The advantage of spectral pooling over max and mean pooling for information retention was visually demonstrated. Experimental results demonstrated that the DCTSPL achieved higher classification rates on a practical dataset compared to networks using the same amount of data reduction. Whereas other pooling methods take a significant loss in classification accuracy for data reduction through the pooling layer, the DCT is able to retain greater accuracy.

Finally, this thesis implemented spectral pooling and upsampling techniques in a deep convolutional autoencoder. Compared to nearest-neighbor and bilinear interpolation techniques, the spectral upsampling (combined with spectral pooling to replace max pooling) had the lowest error rates and the highest quality images produced on a lossy (high sub/upsampling) network compressing handwritten digits, thus demonstrating the potential for these methods to be used for more than just classification purposes.

The natural next step for these spectral pooling networks is to continue testing the techniques on even more datasets using a wider variety of network parameters. It could be that our algorithms do not always outperform competing strategies, and therefore it should be determined when it is most practical to use. As these methods are optimized for Tensorflow [43] and other libraries, it is expected that they will be desired for problems where a sharp reduction in training time and memory costs is desired for only a small loss in accuracy.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[2] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[3] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, November 1994.

[4] Z. Min, L. Xiao, L. Cao, and Hangcheng. Application of the neural network in diagnosis of breast cancer based on levenberg-marquardt algorithm. In *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pages 268–272, December 2017.

[5] C. Lv, Y. Xing, J. Zhang, X. Na, Y. Li, T. Liu, D. Cao, and F. Y. Wang. Levenberg-Marquardt Backpropagation Training of Multilayer Neural Networks for State Estimation of A Safety Critical Cyber-Physical System. *IEEE Transactions on Industrial Informatics*, PP(99):1–1, 2017.

[6] L. Zhang, W. J. Li, and J. F. Qiao. The prediction of daily water demand based on fuzzy neural network with an improved Levenberg-Marquardt algorithm. In *2017 36th Chinese Control Conference (CCC)*, pages 3925–3930, July 2017.

[7] R. Y. Dillak, S. Dana, and M. Beily. Face recognition using 3d GLCM and Elman Levenberg recurrent Neural Network. In *2016 International Seminar on Application for Technology of Information and Communication (ISemantic)*, pages 152–156, August 2016.

[8] D. Hunter, H. Yu, M. S. Pukish III, J. Kolbusz, and B. M. Wilamowski. Selection of Proper Neural Network Sizes and Architectures #x2014;A Comparative Study. *IEEE Transactions on Industrial Informatics*, 8(2):228–240, May 2012.

[9] B. M. Wilamowski and H. Yu. Improved Computation for Levenberg #x2013;Marquardt Training. *IEEE Transactions on Neural Networks*, 21(6):930–937, June 2010.

[10] Javier E. Vitela and Jaques Reifman. Premature Saturation in Backpropagation Networks: Mechanism and Necessary Conditions. *Neural Networks*, 10(4):721–735, June 1997.

[11] B. M. Wilamowski. Neural network architectures and learning. In *IEEE International Conference on Industrial Technology, 2003*, volume 1, pages TU1–T12 Vol.1, December 2003.

[12] Tim Salimans and Diederik P. Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. *arXiv:1602.07868 [cs]*, February 2016. arXiv: 1602.07868.

[13] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016. arXiv: 1607.06450.

[14] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast Training of Convolutional Networks through FFTs. *arXiv:1312.5851 [cs]*, December 2013. arXiv: 1312.5851.

[15] Tyler Highlander and Andres Rodriguez. Very Efficient Training of Convolutional Neural Networks using Fast Fourier Transform and Overlap-and-Add. *arXiv:1601.06815 [cs]*, January 2016. arXiv: 1601.06815.

[16] Oren Rippel, Jasper Snoek, and Ryan P. Adams. Spectral Representations for Convolutional Neural Networks. *arXiv:1506.03767 [cs, stat]*, June 2015. arXiv: 1506.03767.

[17] Jong Hwan Ko, Burhan Mudassar, Taesik Na, and Saibal Mukhopadhyay. Design of an Energy-Efficient Accelerator for Training of Convolutional Neural Networks Using Frequency-Domain Computation. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 59:1–59:6, New York, NY, USA, 2017. ACM.

[18] M. Magdon-Ismail and A. F. Atiya. The early restart algorithm. *Neural Computation*, 12(6):1303–1312, June 2000.

[19] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv:1608.03983 [cs, math]*, August 2016. arXiv: 1608.03983.

[20] B. M. Wilamowski. Neural network architectures and learning algorithms. *IEEE Industrial Electronics Magazine*, 3(4):56–63, December 2009.

[21] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, September 2009.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[23] Dominik Scherer, Andreas Mller, and Sven Behnke. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*, ICANN'10, pages 92–101, Berlin, Heidelberg, 2010. Springer-Verlag.

[24] Jonathan Masci, Ueli Meier, Dan Cirean, and Jrgen Schmidhuber. Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In *Artificial Neural Networks and Machine Learning ICANN 2011*, Lecture Notes in Computer Science, pages 52–59. Springer, Berlin, Heidelberg, June 2011.

[25] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

[26] Xianxu Hou, Linlin Shen, Ke Sun, and Guoping Qiu. Deep Feature Consistent Variational Autoencoder. *arXiv:1610.00291 [cs]*, October 2016. arXiv: 1610.00291.

[27] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and Checkerboard Artifacts. *Distill*, 1(10):e3, October 2016.

[28] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965.

[29] Tapani Raiko, Harri Valpola, and Yann Lecun. Deep Learning Made Easier by Linear Transformations in Perceptrons. In *PMLR*, pages 924–932, March 2012.

[30] X. Cai, K. Tyagi, and M. T. Manry. Training multilayer perceptron by using optimal input normalization. In *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, pages 2771–2778, June 2011.

[31] Andrew B. Watson. Image Compression Using the Discrete Cosine Transform. *Mathematica Journal*, 4(1):81–88, 1994.

[32] Angelika Olejczak, Janusz Korniak, and Bogdan M. Wilamowski. Discrete Cosine Transformation as Alternative to Other Methods of Computational Intelligence for Function Approximation. In Leszek Rutkowski, Marcin Korytkowski, Rafa Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada, editors, *Artificial Intelligence and Soft Computing*, pages 143–153. Springer International Publishing, 2017.

[33] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. CNNpack: Packing Convolutional Neural Networks in the Frequency Domain. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 253–261. Curran Associates, Inc., 2016.

[34] M. Lichman. *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences, 2013.

[35] David Harrison and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, March 1978.

[36] I. C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, December 1998.

[37] Fisher R. A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, August 2012.

[38] Warwick J Nash and Tasmania Marine Research Laboratories. The Population biology of abalone (Haliotis species) in Tasmania. 1, Blacklip abalone (H. rubra) from the north coast and the islands of Bass Strait. Article; Article/Report, Taroona, Tas : Sea Fisheries Division, Marine Research Laboratories - Taroona, Department of Primary Industry and Fisheries, Tasmania, 1994.

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

[40] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. arXiv: 1412.6980.

[41] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, December 2013. arXiv: 1312.6114.

[42] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. arXiv: 1406.2661.

[43] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association.