Fall 2020

Experiments on the Neural Network Approach to the Handwritten Digit Classification Problem

William Meissner

EXPERIMENTS ON THE NEURAL NETWORK APPROACH TO THE HANDWRITTEN DIGIT CLASSIFICATION PROBLEM

by

WILLIAM MEISSNER

(Under the Direction of Kai Wang)

ABSTRACT

When the MNIST dataset was introduced in 1998, training a network was a multiple week problem in order to receive results far less accurate than an average CPU can produce within a couple of hours today. While this indicates that training a network on such a dataset is not the complicated problem it may have been twenty years ago, the MNIST dataset makes a good tool for study and testing with beginner and medium complexity neural networks. This paper follows along with the work presented in the online textbook “Neural Networks and Deep Learning” by Michael Nielson and an updated repository of his python code examples made current for the most recent version of python. In this paper, the convolutional neural networks outlined in chapter 6 of "Neural networks and deep learning" will be built, run and the results will be analyzed and compared to the results shown in Nielson’s work to see how convolutional layers improve accuracy of the network. Making use of Nielson’s network3.py, I will conduct several experiments on this example starting with a single hidden layer to get a baseline accuracy which will be used to compare with the further tests in order to determine if the more complex and additional layers have improved the network’s accuracy. The second test will build on the first network by additionally utilizing a 5x5 local receptive field, 20 feature maps, and a max-pooling layer 2x2. The third test will use the previous features and insert a second convolutional-pooling layer identical in design to the first layer. The fourth test will build the network using rectified linear units and some L2 regularization (lmbda=0.1). Using the four different networks which each seek to correctly determine the appropriate digit from a dataset of thousands of handwritten images, I will compare how the networks are impacted by these modifications and compare my results with those presented in Michael Nielson’s textbook.

INDEX WORDS: Neural network, Deep learning, Machine learning, MNIST

EXPERIMENTS ON THE NUERAL NETWORK APPROACH TO THE HANDWRITTEN DIGIT CLASSIFICATION PROBLEM

by

WILLIAM MEISSNER

B.S., James Madison University, 2012

M.S., Georgia Southern University, 2020

A Thesis Submitted to the Graduate Faculty of Georgia Southern University

in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

EXPERIMENTS ON THE NUERAL NETWORK APPROACH TO THE HANDWRITTEN DIGIT CLASSIFICATION PROBLEM

by

WILLIAM MEISSNER

Major Professor:	Kai Wang
Committee:	Gursimran Singh Walia
	Wen-Ran Zhang

Electronic Version Approved:
December 2020

ACKNOWLEDGMENTS

I wanted to thank Dr. Kai Wang for advising my project progress over the past year and helping improve my project with his feedback and suggestions for revisions. Thank you to Dr. Walia and Dr. Zhang for serving on my defense committee. Thank you to my wife and family for supporting me as I worked on this project and on my Master's degree over the last three years.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1
INTRODUCTION

Deep learning and neural networks are becoming further and further integrated and applicable across just about every sector of business and education. Pattern recognition, object detection, language processing, segmentation, video analysis, spam detection, recognition and classification of speech and images are now being used everywhere from classroom education to assisting businesses to make critical data driven decisions. Neural networks are driving this field of data analytics and while handwritten digit recognition is not going to inform a board of directors the direction to take their company, it serves as a test case to demonstrate achievable network accuracy on a well-known dataset. A basic artificial neural network has an input layer, an output layer, and one or more hidden layers in between. Together, the whole network is used to classify a given dataset with an accurate classification. For example, in this paper and similar studies, neural networks are trained to classify whether a handwritten digit is one of the specific numbers 0 through 9.

Purpose of the Study

In this paper, the convolutional neural networks outlined in chapter 6 of "Neural Networks and Deep Learning" will be built, run and the results will be analyzed and compared to the results shown in Nielson's work to see how convolutional layers improve accuracy of the network. The four neural networks that will be analyzed in this paper are as follows: The first network is a shallow model with a single hidden layer containing 100 hidden neurons. The second network contains a convolutional layer consisting of 5x5 local receptive fields, a stride length of 1, and 20 feature maps along with a max-pooling layer, which combines the features using 2x2 pooling windows. The third network is the same as the second network, but will have a second convolutional and pooling layer with the same features as outlined above. The fourth network will utilize rectified linear units instead of using the sigmoid activation function which was used in the previous models. This network is a little too susceptible to overfitting, so L2 regularization will be used to address this. The regularization parameter will be $\lambda=0.1$.

CHAPTER 2
RELATED WORK

Convolutional Neural Networks

Architecture

Traditional models for pattern recognition face several problems in their structure so that hardware and processing capability put limitations on the success of the models. In [4], the authors laid the foundation for the modern understanding of convolutional networks. They outlined that traditional pattern recognition networks faced several problems when applied to character recognition which limited their capacity to successfully classify the data. In the traditional model, LeCun et al. explained that, "a fully connected first layer with, say one hundred hidden units in the first layer, would already contain several tens of thousands of weights. Such a larger number of parameters increases the capacity of the system and therefore requires a larger training set. In addition, the memory requirement to store so many weights may rule out certain hardware implementations" [4]. Additionally, the authors went on to explain that the lack of ability for image applications to have inbuilt invariance to account for local distortion of the image set was the main limitation of such an unstructured network. Variance is inherent in a dataset such as one made up of numbers, letters or both numbers and letters as different writing styles will change placement of the distinctive features which are important for the network to learn. Convolutional networks address these constraints by forcing replication of weight configurations across space which creates a shift invariance that reduces the burden on the network to learn the dataset. The three architectural ideas that comprise a convolutional network were also laid out in their paper: local receptive fields, shared weights, and pooling.

Local receptive fields

A local receptive field connects the input pixels to a hidden layer, but every input pixel is not assigned to its' own neuron. For example, in the second model outlined in the next chapter, a five by five local receptive field maps a group of twenty-five input pixels. That region in the input image is called the local receptive field for the hidden neuron. It's a little window on the input pixels. Each connection learns

a weight. And the hidden neuron learns an overall bias as well. You can think of that particular hidden neuron as learning to analyze its particular local receptive field [7].

Shared weights and biases

Sharing weights and biases across all neurons is an important aspect of the convolutional network. Besides making it easier to allow a hidden neuron to determine certain aspects of an image such as the vertical/horizontal edges of an image, it significantly reduces the total number of parameters involved in the network. Considering the network being analyzed in this paper, each 5x5 feature map requires 25 shared weights and a single shared bias. Assuming the twenty feature maps similar to the models outlined in the next section, that is 26 parameters x 20 maps for a total of 520 parameters making up the convolutional layer. A similar fully connected layer of 28x28 input neurons is already 784 parameters (one for each neuron in the 28 by 28 field) before factoring in the weights and biases which would be multiplicative ballooning the number of parameters comprising the fully-connected layer.

Pooling Layers

Convolutional networks make use of an additional type of layer known as pooling layers. Pooling layers are used after convolutional layers in order to simplify the output of the convolutional layer. In the tests run below, using a procedure known as max-pooling each neuron of the max-pool output is representative of a 2x2 block of hidden neurons that were the output of a feature map from the convolutional layer. As the convolutional layer used in the tests built twenty feature maps of 24x24 neurons output, after the pooling layer is completed, there will be twenty feature maps of 12x12 neurons.

Considering these three features together, the four different layers at use by the four neural networks in this paper can be explained. We start with a grid of input neurons in a configuration of 28x28. These input neurons are used to encode the pixel intensities for the MNIST image. This is then followed by a convolutional layer using a 5×5 local receptive field and 3 feature maps. The result is a layer of 20×24×24 hidden feature neurons. The next step is a max-pooling layer, applied to 2×2 regions, across each of the twenty feature maps. The result is a layer of 20×12×12 hidden feature neurons. The final layer

of connections in the network is a fully-connected layer. That is, this layer connects every neuron from the max-pooled layer to every one of the 10 output neurons [7].

SoftMax Layer

A SoftMax layer is utilized to normalize the network's output from the given data to a probability distribution. The function will take the input as a vector of the real numbers and normalize the distribution so that the components will add up to one and the larger inputs will correspond to larger probabilities. The SoftMax function is shown below. Given that the exponential function will always be positive and that Z_j^L is representative of the current neuron while k represents the final neuron, then the sum of all neurons after normalization has been done will be 1. This can be represented mathematically as $a_1^L + a_2^L + \cdots + a_{k-1}^L + a_k^L = 1$.

$$a_{Lj} = \frac{e^{Z_j^L}}{\sum_k e^{Z_k^L}}$$

As all values of the output from a SoftMax layer add up to 1, each individual output node is equivalent to or can be considered as representative of the probability distribution of the dataset.

Sigmoid Activation Function

A Sigmoid Activation Function is appropriate for use in a neural network where inputs have been normalized between 0 to 1. In this paper, this was achieved through the SoftMax Layer explained above. Similar to a perceptron, sigmoid neurons have inputs in the range of 0 to 1, but instead of being limited to only values of 0 or 1, sigmoid neurons can have inputs of any value between 0 and 1. Each has its own weight and bias. The sigmoid activation function is used in models where the goal is predicting the probability as an output as the probability exists only between the range of 0 and 1. The first three networks looked at in this paper will make use of the sigmoid activation function and the final network will utilize rectified linear units which are described below.

Rectified Linear Units

Rectified linear units are an activation function similar to the Sigmoid activation function. For any positive value inputted to the rectified linear unit function, the same exact value will be returned by the function. The function is defined as: $f(x)=max(0,x)$

Rectified Linear Units are preferable and perform better than sigmoid neurons in some situations, but not all and there is not a clearly defined set of circumstances when Rectified Linear units will outperform sigmoid neurons. Sigmoid neurons will reach a point where their output approaches either zero or one and will stop learning and be considered saturated. Rectified linear units will never saturate through increasing the weighted input, which means that learning of the network does not slowdown, but when the weighted input becomes negative, the neuron will not learn at all. So, it is not as simple deciding to always use rectified linear units over other activation functions and increasing the weighted inputs incrementally in order to collect ever increasing results in the capacity of the network to learn.

MNIST Dataset

The MNIST dataset is from the National Institute of Standards and Technology (NIST). The training set consists of handwritten numbers from 250 different people, pooled from high school students and employees at the U.S. Census Bureau. The MNIST dataset in total contains 60,000 images in the training set and 10,000 patterns in the testing set [5]. The dataset can be downloaded online and some examples from the MNIST dataset are shown in Figure 1.

Fig. 1. Example Images from the MNIST Dataset, Including 60,000 Images in the Training Set and 10,000 Patterns in the Testing Set.

CHAPTER 3
CNN MODELS

Michael Nielsen's network3.py used in his textbook, "Neural Networks and Deep Learning" provided the models used below. His GitHub repository is written for Python 2.7 and is incompatible with Python 3. Michal Daniel Dobrzanski created a repository for Nielsen's code that was updated to work for Python 3, and Dobrzanski's repository was used to build the networks tested in this paper. Network3.py is used as a library to build the convolutional networks explained below. Along with this existing code, the machine learning library Theano was used as it makes for an easy implementation of backpropagation in convolutional neural networks.

Baseline Network

In order to inform how the models in this paper perform, it is helpful to create a baseline for comparison. A shallow model with a single hidden layer was used to accomplish this goal. The layer contained 100 hidden neurons which are passed through a SoftMax Layer in order to normalize the inputs and the model trained for 60 epochs, using a learning rate of η=0.1, a mini-batch size of 10, and no regularization [7].

```
>>> import network3
>>> from network3 import Network
>>> from network3 import ConvPoolLayer, FullyConnectedLayer, SoftmaxLayer
>>> training_data, validation_data, test_data = network3.load_data_shared()
>>> mini_batch_size = 10
>>> net = Network([
      FullyConnectedLayer(n_in=784, n_out=100),
      SoftmaxLayer(n_in=100, n_out=10)], mini_batch_size)
>>> net.SGD(training_data, 60, mini_batch_size, 0.1,
         validation_data, test_data)
```

One Convolutional Layer and One Pooling Layer

The second network builds on the first network. On top of the 100 hidden neuron layer which was passed through a SoftMax layer, a convolutional pooling layer utilizing 5x5 local receptive fields, a stride length of 1, and 20 feature maps were added. Along with this, a max-pooling layer, which combines the features using 2x2 pooling windows, was inserted [7].

```
>>> net = Network([
    ConvPoolLayer(image_shape=(mini_batch_size, 1, 28, 28),
                  filter_shape=(20, 1, 5, 5),
                  poolsize=(2, 2)),
    FullyConnectedLayer(n_in=20*12*12, n_out=100),
    SoftmaxLayer(n_in=100, n_out=10)], mini_batch_size)
>>> net.SGD(training_data, 60, mini_batch_size, 0.1,
        validation_data, test_data)
```

Inserting a Second Convolutional-Pooling Layer into the Previous Model

Next, a second convolutional-pooling layer was inserted onto the existing framework from the second model to see if this additional layer would be able to further improve the accuracy of the network. The new layer was inserted between the existing convolutional-pooling layer and the fully-connected hidden layer. Using the same parameters as the first convolutional-pooling layers, and matching hyper parameters, the model looks like this [8]:

```
>>> net = Network([
    ConvPoolLayer(image_shape=(mini_batch_size, 1, 28, 28),
                  filter_shape=(20, 1, 5, 5),
                  poolsize=(2, 2)),
    ConvPoolLayer(image_shape=(mini_batch_size, 20, 12, 12),
                  filter_shape=(40, 20, 5, 5),
                  poolsize=(2, 2)),
    FullyConnectedLayer(n_in=40*4*4, n_out=100),
    SoftmaxLayer(n_in=100, n_out=10)], mini_batch_size)
>>> net.SGD(training_data, 60, mini_batch_size, 0.1,
        validation_data, test_data)
```

Rectified Linear Units and Some L2 Regularization (lmbda=0.1)

The previous models are variations of one of the networks introduced in [4], referred to in that paper as LeNet-5. These previous networks all make use of a sigmoid activation function. For the fourth network, rectified linear units will be used instead of using the sigmoid activation function. This network will also be trained for 60 epochs, with a learning rate of $\eta=0.03$. This network is a little too susceptible to overfitting, so L2 regularization will be used to address this. The regularization parameter will be $\lambda=0.1$:

```
>>> from network3 import ReLU
>>> net = Network([
    ConvPoolLayer(image_shape=(mini_batch_size, 1, 28, 28),
                  filter_shape=(20, 1, 5, 5),
                  poolsize=(2, 2),
                  activation_fn=ReLU),
    ConvPoolLayer(image_shape=(mini_batch_size, 20, 12, 12),
                  filter_shape=(40, 20, 5, 5),
                  poolsize=(2, 2),
                  activation_fn=ReLU),
    FullyConnectedLayer(n_in=40*4*4, n_out=100, activation_fn=ReLU),
    SoftmaxLayer(n_in=100, n_out=10)], mini_batch_size)
>>> net.SGD(training_data, 60, mini_batch_size, 0.03,
      validation_data, test_data, lmbda=0.1)
```

CHAPTER 4
RESULTS

Discussion of the Obtained Classification Results

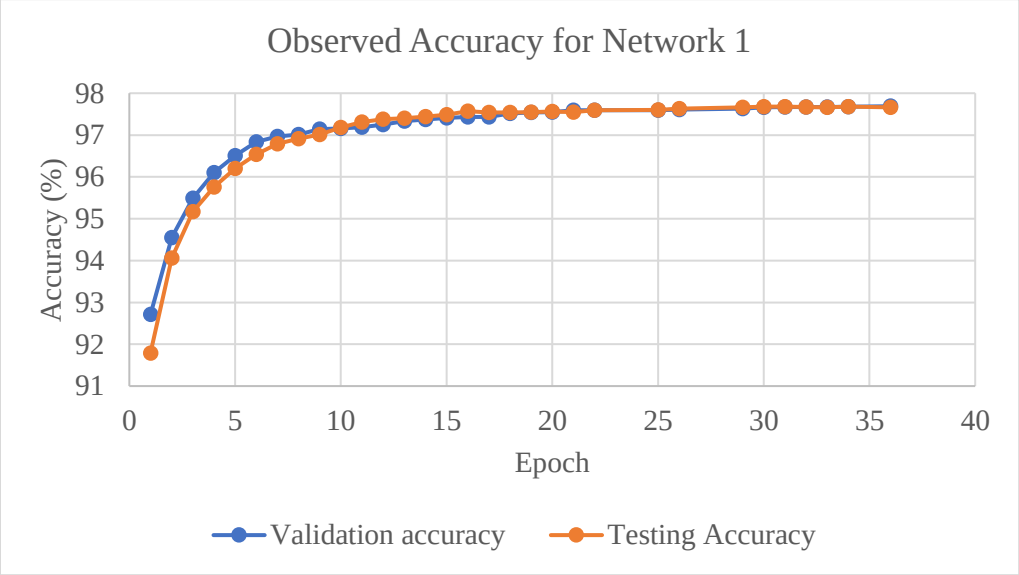
Four neural networks have been trained on the MNIST dataset in order to observe the variation of accuracies for handwritten digits. Test and validation accuracy for 60 different epochs were observed on the four neural networks in order to test and examine how these changes impact the accuracy of the network. Networks 1 and 2 were run for a full 60 epochs, while networks 3, and 4 were stopped early as the network was no longer showing improvement in accuracy. These networks were stopped after 30 epochs and 42 epochs respectively. Figures 2, 3, 4 and 5 show the performance of the neural network for different combinations of hidden layers. Table 1 shows the minimum and maximum test and validation accuracies of the network found after the training for the four different cases by varying number of hidden layers for the recognition of handwritten digits.

Table 1. Results of Four Neural Networks Over the Training Epochs

Neural Network	Minimum Validation Accuracy (%)	Minimum Validation Epoch	Corresponding Test Accuracy (%)	Maximum Validation Accuracy (%)	Maximum Validation Epoch	Corresponding Test Accuracy (%)
1	92.71	1	91.79	97.69	36	97.66
2	93.80	1	93.19	98.69	59	98.68
3	90.97	1	90.54	98.92	24	98.89
4	97.56	1	97.39	99.19	37	99.20

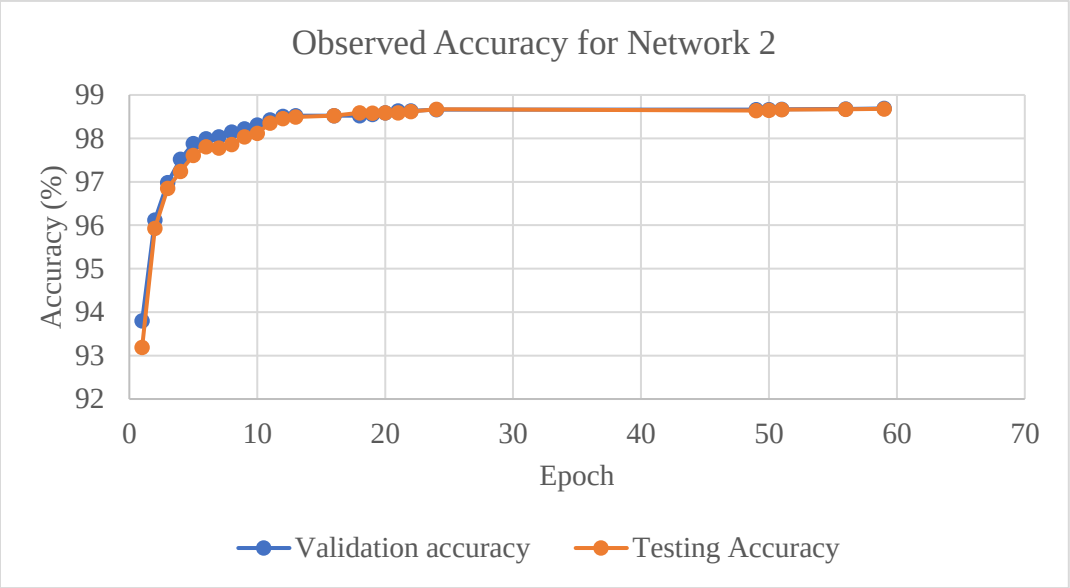
Looking at the results of the first network, at epoch 1 the minimum test accuracy of 91.79% is found and a validation accuracy of 92.71% is found. At epoch 36, the maximum validation accuracy is found to be 97.69% and at epoch 31, the maximum testing accuracy is found to be 97.68%.

Fig. 2. Observed Accuracy for Network 1



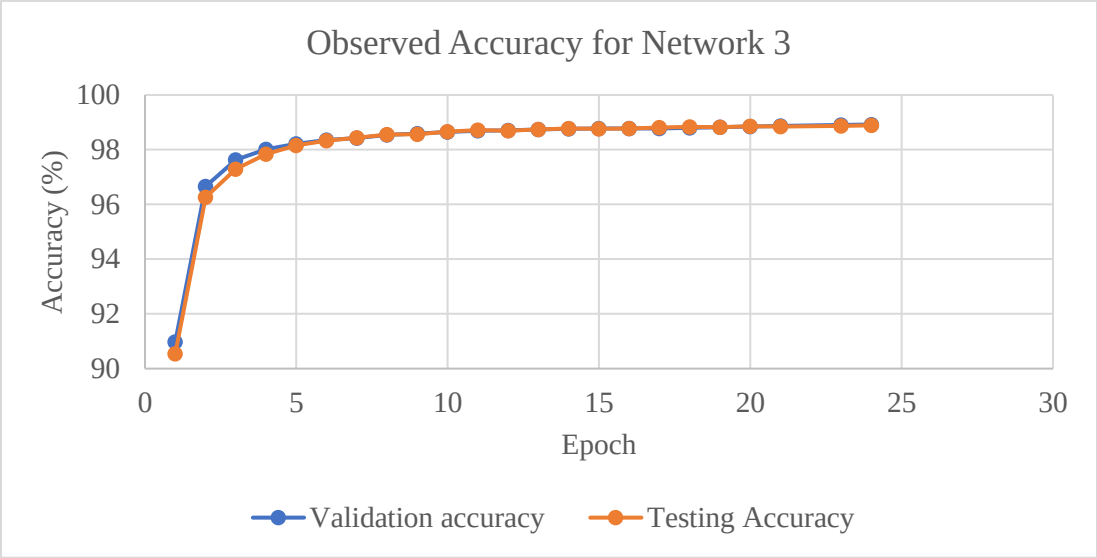
In the second network, at epoch 1 the minimum test accuracy of 93.19% is found and the minimum validation accuracy of 93.80% is found. At epoch 59, the maximum validation accuracy is found to be 98.69% and the maximum testing accuracy was also found at this epoch with a value of 98.68%.

Fig. 3. Observed Accuracy for Network 2



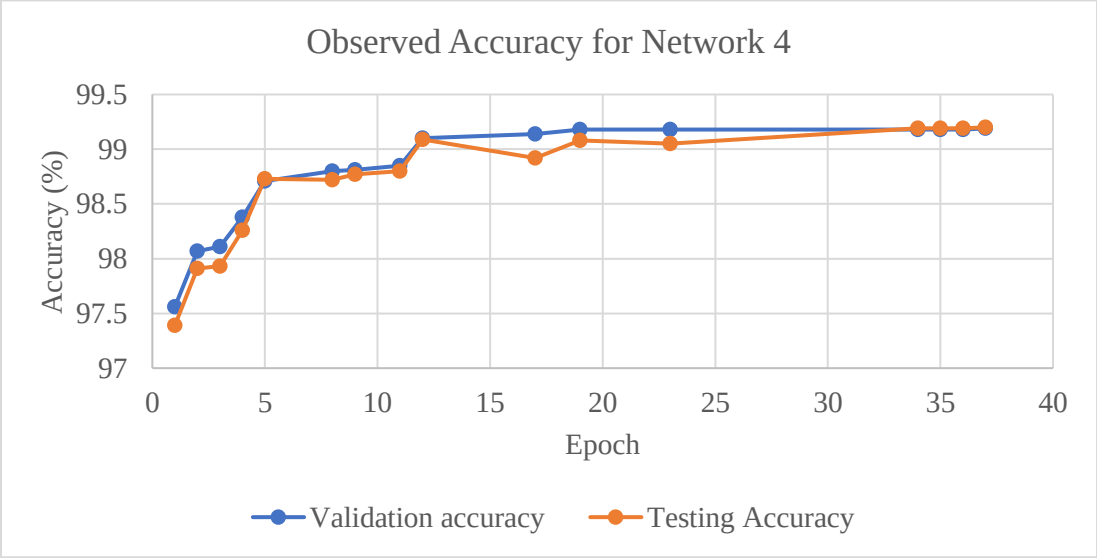
For the third network, epoch 1 produced the minimum validation accuracy of 90.97% and the corresponding minimum test accuracy was found to be 90.54%. At epoch 24, the maximum validation accuracy is found to be 98.92% and the maximum testing accuracy was also found at this epoch with a value of 98.89%.

Fig. 4. Observed Accuracy for Network 3



The fourth network at epoch 1 found the minimum validation accuracy of 97.56% and the minimum test accuracy of 97.39%. At epoch 37, the maximum validation accuracy is found to be 99.19% and the maximum testing accuracy was also found at this epoch with a value of 99.20%.

Fig. 5. Observed Accuracy for Network 4



CHAPTER 5
CONCLUSION

The tests performed on the neural networks in this paper had closely similar results to the figures reported by Nielsen in his textbook. Nielsen only reported the obtained classification accuracies of his tests of the networks, but comparatively his results closely mirror the results obtained in tests run for this paper. Nielsen's classification accuracies for networks one through four were 97.80%, 98.78%, 99.06%, and 99.23% respectively. While the testing accuracies reported above are marginally lower, they increase by nearly identical margins. This shows an observable and repeatable impact on network accuracy through the additional hidden layers and complexity added to the network over the course of the four network tests.

REFERENCES

[1] Rigouste, L., Cappé, O. and Yvon, F. 2007. Inference and evaluation of the multinomial mixture model for text clustering. Information processing & management. 43(5): 1260-1280.

[2] LeCun, Y., Cortes, C., and Burges, C. 2013. THE MNIST DATABASE of handwritten digits. http://yann.lecun.com/exdb/mnist/

[3] Ciregan, D., Meier, U. and Schmidhuber, J. 2012. Multi-column deep neural networks for image classification. In Computer vision and pattern recognition. 2012 IEEE conference: 3642-3649.

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

[5] Kussul, E. and Baidyk, T. 2004. Improved method of handwritten digit recognition tested on MNIST database, Image and Vision Computing. 22 (12): 971–981.

[6] Steppan, J. 2017. Sample images from MNIST test dataset. CC BY-SA 4.0. https://commons.wikimedia.org/w/index.php?curid=64810040

[7] Nielsen, M. 2019. Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/