**Program Structures and Algorithms(INFO6205)**

Final Project Report

**Digit Recognition using Neural Networks**

**Introduction**

Neural networks are used as a method of deep learning, one of the many subfields of artificial intelligence. They were first proposed around 70 years ago as an attempt at simulating the way the human brain works, though in a much more simplified form. Individual ‘neurons’ are connected in layers, with weights assigned to determine how the neuron responds when signals are propagated through the network. Previously, neural networks were limited in the number of neurons they were able to simulate, and therefore the complexity of learning they could achieve. But in recent years, due to advancements in hardware development, we have been able to build very deep networks, and train them on enormous datasets to achieve breakthroughs in machine intelligence.

**Problem Statement**

In this project, we have implemented a small subsection of object recognition. Using Java, we have taken hand-drawn images of the numbers 0-9 and built and trained a neural network to recognize and predict the correct label for the digit displayed.

The following steps have been identified as a part of the solution to the problem statement:

* Defining the Neural Network Architecture
* Importing the Image and label database (MNIST)
* Programming the Neural Network and associated methods
* Writing Unit tests and checking the validity of methods
* Training the Neural Network using MNIST database
* Testing the Neural Network with new images
* Performing variations in training and architecture to observe changes.

**Neural Network Architecture**

The architecture of the neural network refers to elements such as the number of layers in the network, the number of units in each layer, and how the units are connected between layers.  
Different architectures can yield dramatically different results, as the performance can be thought of as a function of the architecture among other things, such as the parameters, the data, and the duration of training.

For the neural network we are designing, we have decided to use the following parameters:

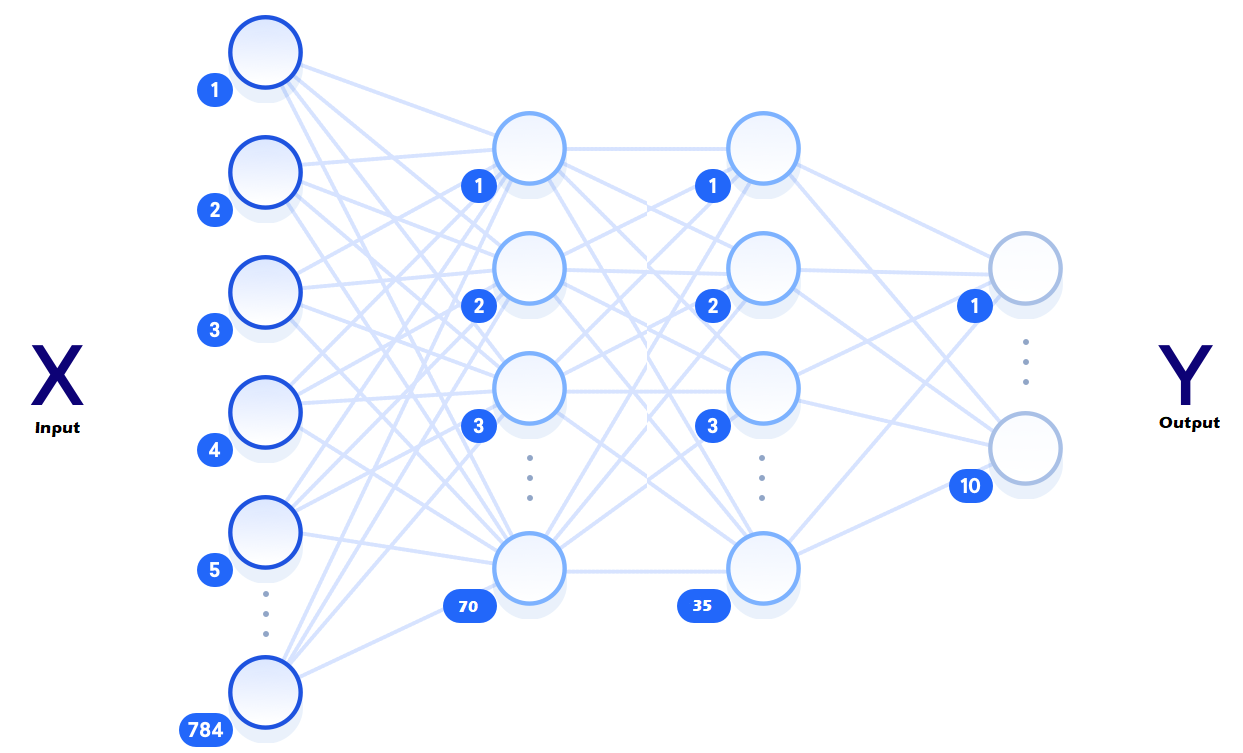
* Number of hidden layers: **2**
* Number of neurons in Hidden layer 1: **70**
* Number of neurons in Hidden layer 2: **35**

For this problem we are considering a **28x28** pixel image which is compatible with the images in the MNIST database and we would eventually classify it as a digit from **0-9** appropriately.

This implies we would have the following input and output parameters:

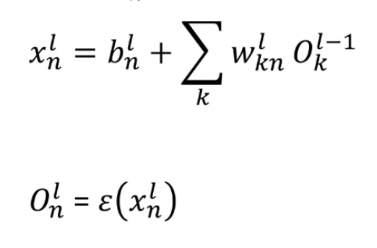
* Neurons in Input layer: **784**
* Neurons in Output layer: **10**

Here is a visualization of the designed neural network:



**Neuron Characteristics**

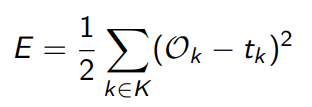
Each neuron has the following parameters:

And the equations used to compute the output of each neuron are given below:

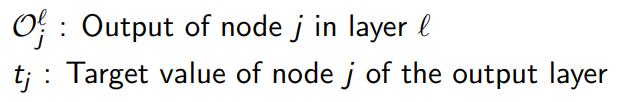
All the neurons have been programmed as formulated above.

**Backpropagation Algorithm**

The error or the loss is given by the following equation

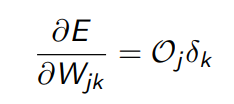


Where

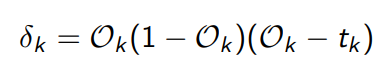


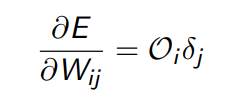
We let the error of the network for a single training iteration be denoted by E. We want to calculate , the rate of change of the error with respect to the given connective weight, so we can minimize it.

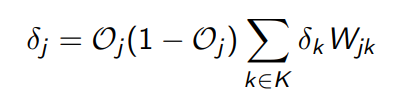
For the output layer nodes, we have the following equation:

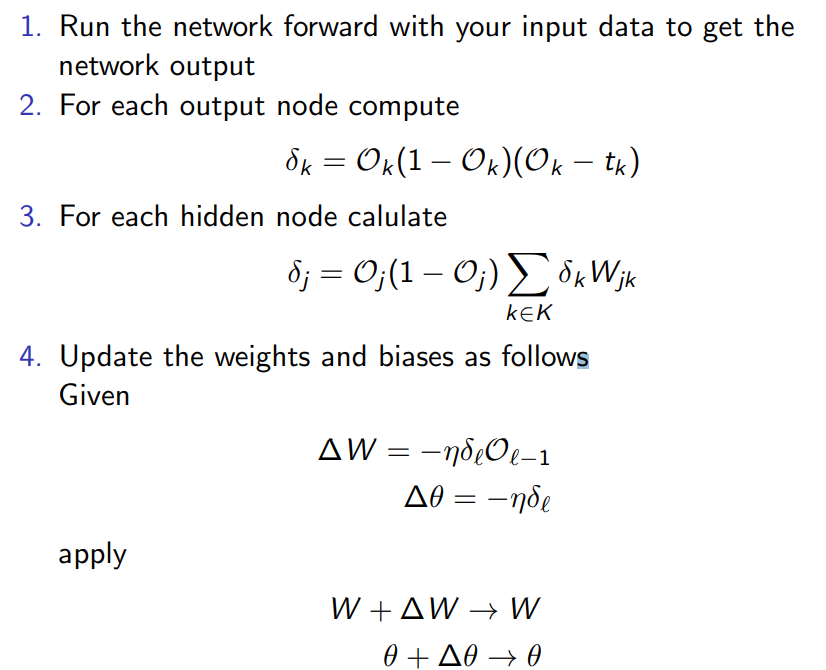


Where



And for the hidden layer node we have the following equations: 

Where 

The following procedure is programmed which would execute the backpropagation algorithm in our code: 

**Training Process**

The essence of the training process in deep learning using neural networks is to optimize the loss function. Here we are aiming to minimize the difference between the predicted labels of the images, and the true labels of the images. The process involves four steps which are repeated for a set number of iterations:

* Propagate values forward through the network
* Compute the loss
* Propagate values backward through the network
* Update the parameters

At each training step, the parameters are adjusted slightly to try and reduce the loss for the next step. As the learning progresses, we should see a reduction in loss, and eventually we can stop training and use the network as a model for testing our new data.

When training our network, we have set the following parameters to certain values :

Epoch: An epoch is one entire computation of a dataset being passed forward and backward one single time.

Batch size: Total number of training samples present in a single batch.

Loops: Loops are the number of batches which are computed to complete a single epoch.

**Image Extraction from MNIST Database**

The first and most important step in any machine learning task is to prepare the data. Since, any errors introduced during this step will cause the learning algorithm to learn incorrect patterns. We have leveraged the hard work of the National Institute of Standards and Technology and the hardest parts have been done for us. The data has been collected and is already well-formatted for processing.

The code to extract the MNIST database files has been written based on the information given in the MNIST website and appropriate methods have been written to extract images as well as the associated labels.

**Code**

The code has been divided into the following packages based on the functionality of the files inside the package :

fullyconnectednetwork : This package deals with the specifics of the components of a neuron and the associated network like output computation and backpropagation.

mnist: This package contains all the necessary methods to access the MNIST database and extract images from it. It also contains the methods to process an external image.

trainingset: This package contains the methods which involve the training of the network based on the batch size and the epoch number.

test: This package contains all the unit tests that can be executed on the methods which are used in the computation of the neural network.

UI: This package contains all the files associated with the User Interface which was made to train and test the network.

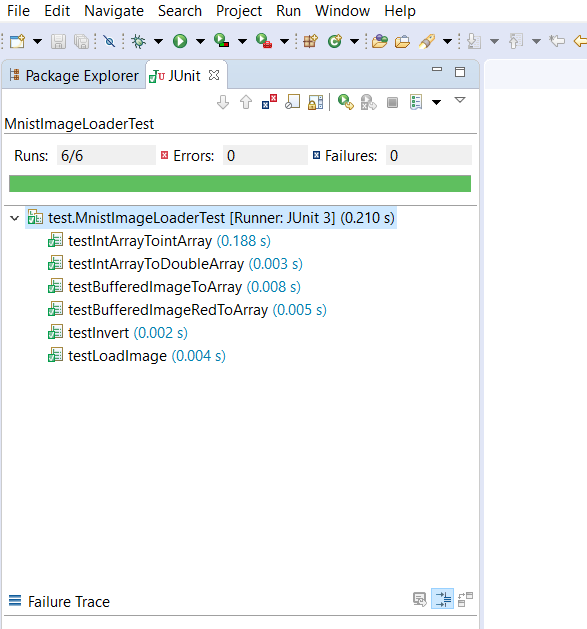
**Unit Tests**

The unit tests have been written to ensure the creation and computation of network related parameters are correct.

We have written unit tests using the JUnit 3.0 framework. The written tests cover the main computations involved in the neurons and the network.

Also, unit tests have been written to check if the image pre-processing being performed before using it in the network is as accurate.

Here is a screenshot that the written functions are correct:

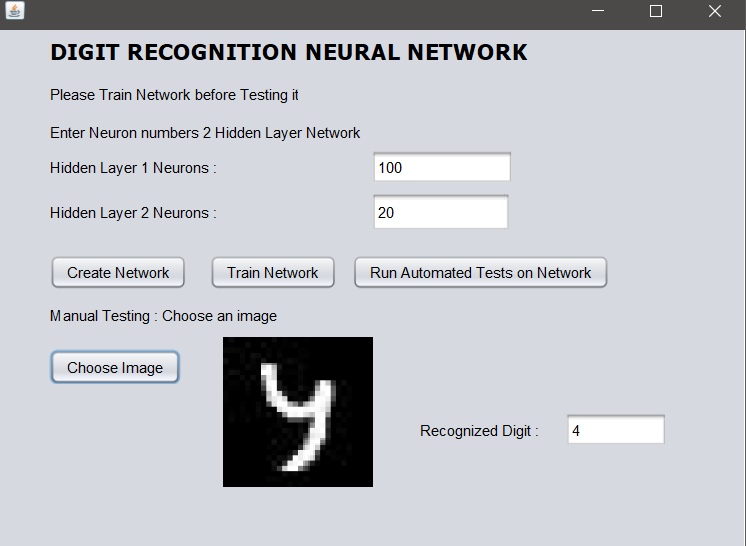


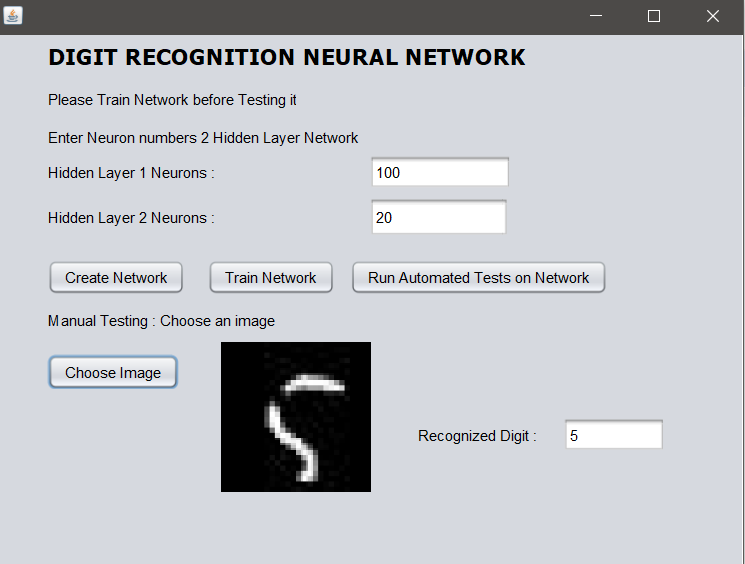
**UI**

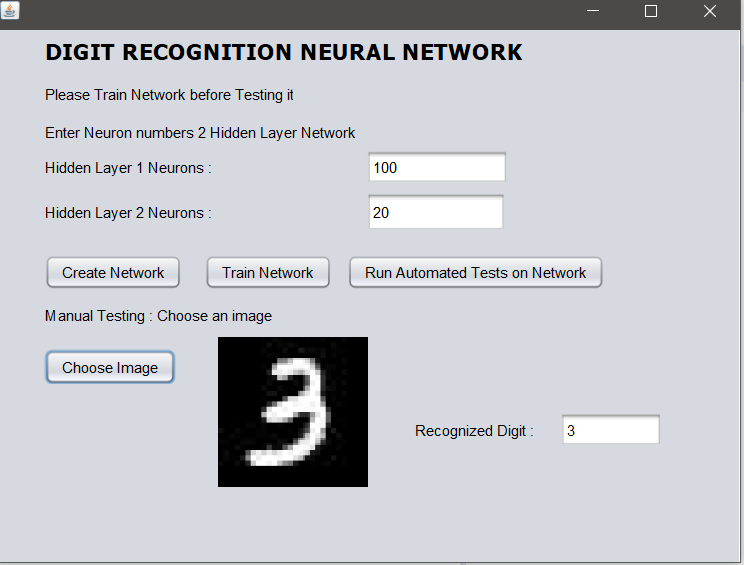
A user interface has been made by us to set the number of hidden layers and for the creation, training and testing of our network. For the UI we have fixed the number of hidden layers to 2.

Also we have created a button to run automated tests, where we take the images from the MNIST database and recognize the digits using our trained neural network and compare it against the label of the images to give us the accuracy percentage of our network.

We have added an option to perform manual testing on images which are also included with the project to showcase the functionality, where it shows the image which was selected and the output computed by network. The UI design is shown below:







**Confusion Matrix:**

Confusion Matrix helps in calculating the accuracy of the classification model which indirectly helps us to describe the performance of the classification model.  
  
Accuracy = currently classified items/ all classified items

4957 / 5000 -> 99.14%

4966 / 5000  -> 99.32 %

4954 / 5000  -> 99.08 %

4931 / 5000  -> 98.62 %

**Results**

Base case: For the base case as given in the neural network architecture we have chosen the number of hidden layers as 2. The first and second containing 70 and 35 neurons respectively.

We have trained this neural network, using the following training parameters :

Number of Images used for training : 5000

Number of Epoch : 50

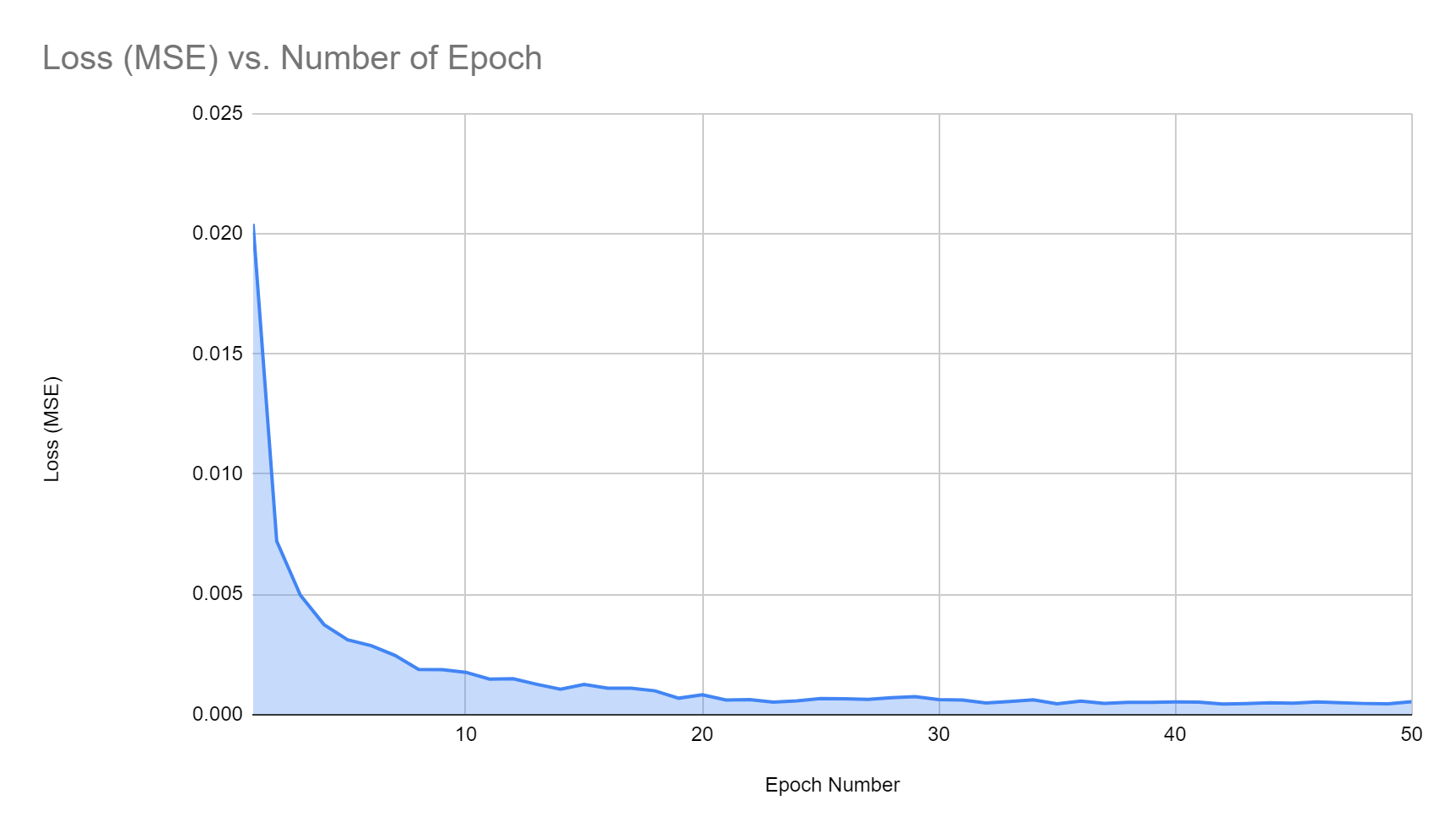
Batch size : 100

Number of Loops : 50

The results are as follows:

* The time taken for processing images from the database and training by optimizing weights of the neural network : **65 seconds**
* Accuracy of test cases : **98.98%**

The variation of the Loss which is measured by the mean square error between the computed probability of a digit and the labelled probability of a digit has been captured during the training of each epoch and the resulting graph is given below :



From this graph we can clearly see that the loss reduces to the order of 0.001 by the end of 20 epoch and kind of saturates after that to a value of 0.00045 which suggests that the training has been done quite efficiently.

We can also infer that the number of epoch which are computed in a training run give us to what extent the network is trained, that is higher the number of epoch the higher would be the accuracy.

**Alternate Experiments**

To understand the implication of changing the various training related parameters we have performed the following experiments:

Experiment 1: In this experiment we have reduced the number of loops (one fifth of original) by keeping the rest of the parameters as the same values as base case, the training parameters are given below and the network architecture is kept the same:

Number of Images used for training : 5000

Number of Epoch : 50

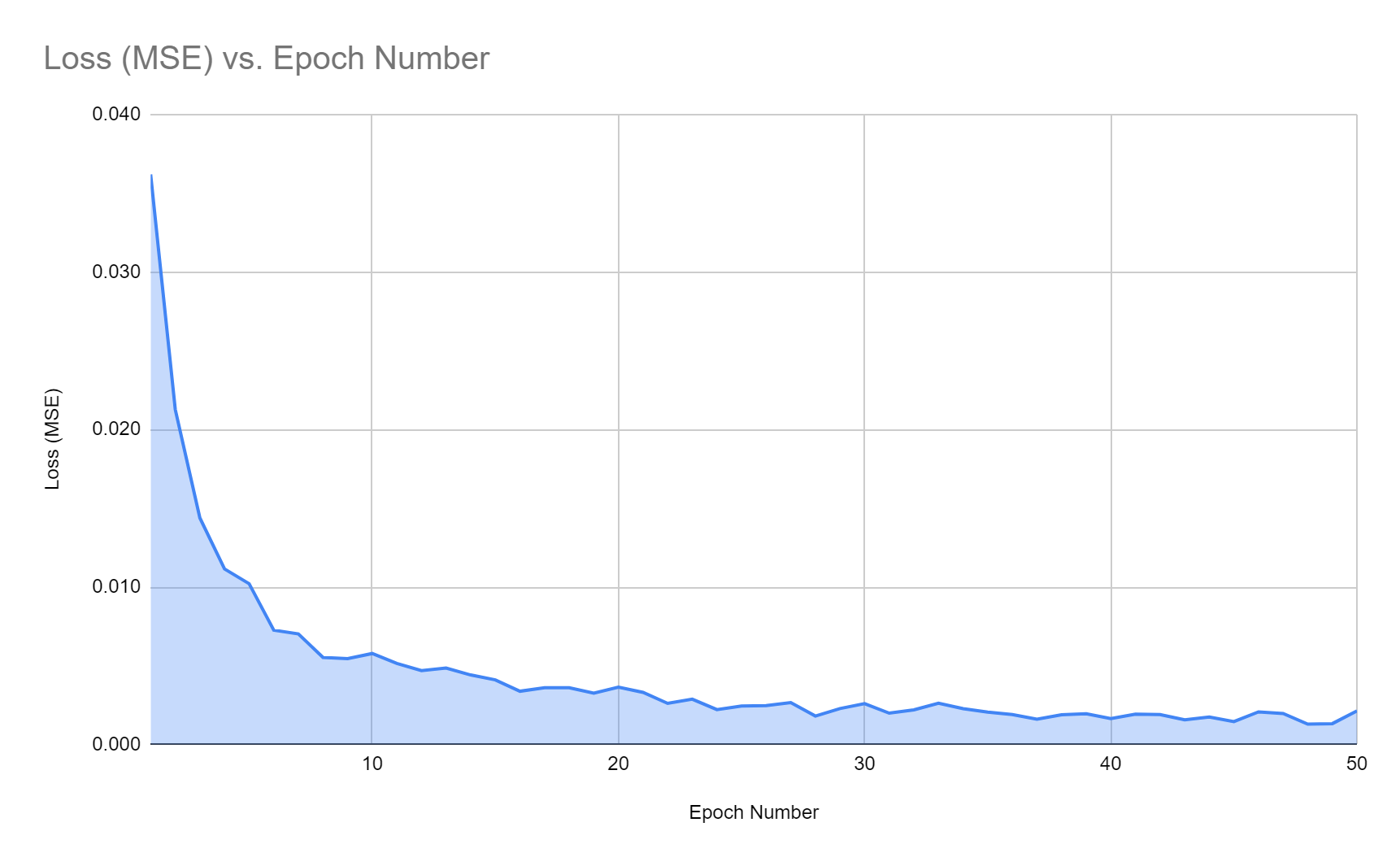
Batch size : 100

Number of Loops : 10

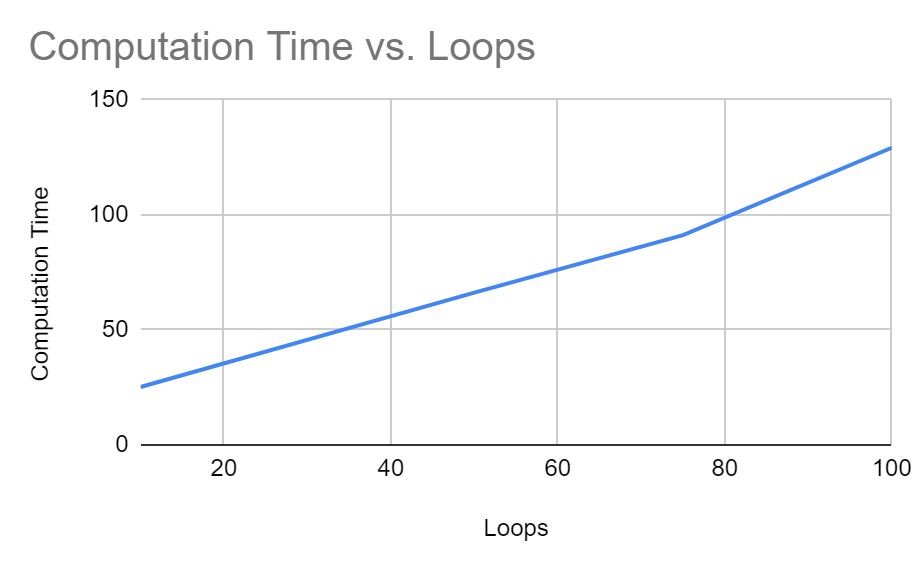
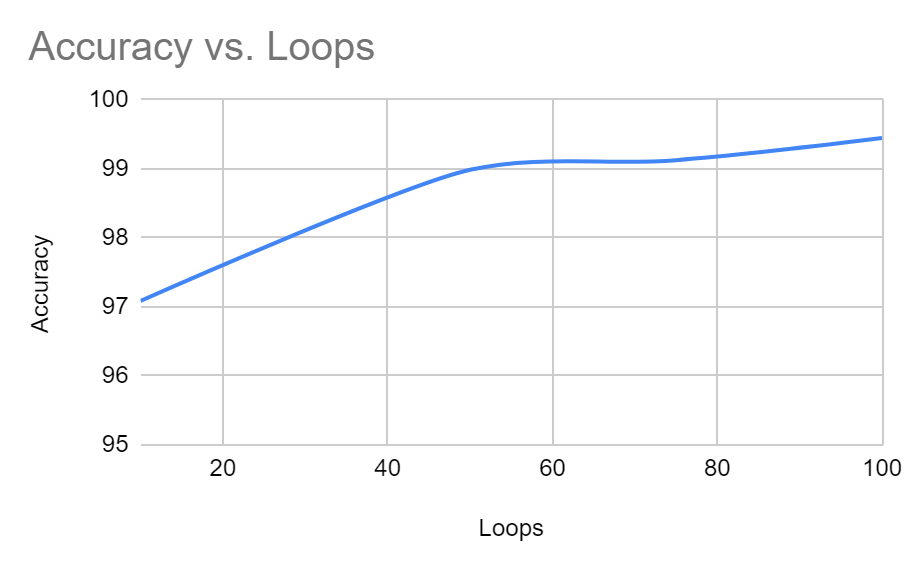
The results are as follows:

* The time taken for processing images from the database and training by optimizing weights of the neural network : **25 seconds**
* Accuracy of test cases : **97.08%**

The loss vs number of epoch graph is given below :



From this experiment we can see that there is a clear increase in the loss ( Mean square error ) which saturates at 0.002 as compared to the base case which saturates at 0.00045, also the accuracy significantly decreases as well as the computation time has a major decrease. So it would be a trade-off to reduce between accuracy and training time when we have to choose the number of loops.

The computation time and the accuracy measured are compared as we vary the number of loops by conducting various other experiments while keeping other parameters the same: 

We can clearly see from the above graphs that the computation time steadily increases while accuracy kind of saturates which gives us a trade-off decision to make while choosing number of loops.

Experiment 2: In the second experiment we observe the change in the network computation time and accuracy due to the change in the batch size while keeping the other parameters the same as the base case. The parameters used are given below and the network architecture is kept the same:

Number of Images used for training : 5000

Number of Epoch : 50

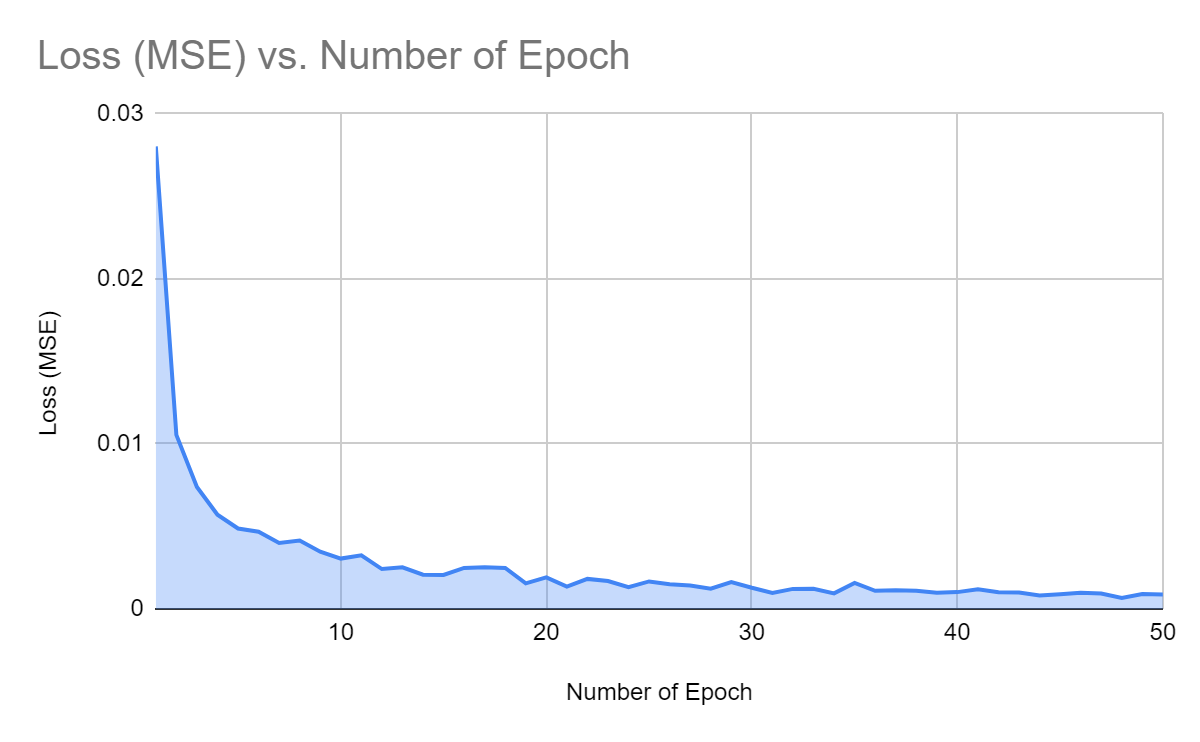
Batch size : 40

Number of Loops : 50

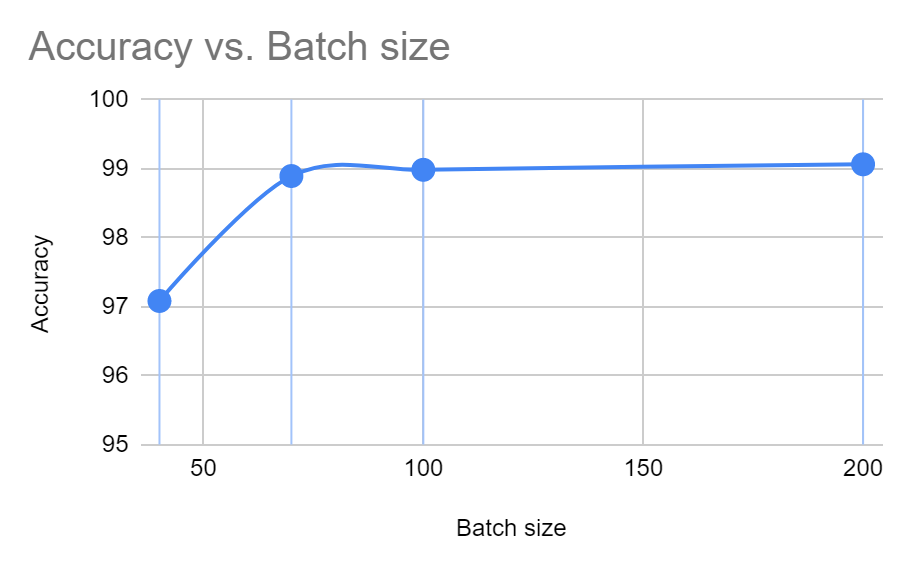
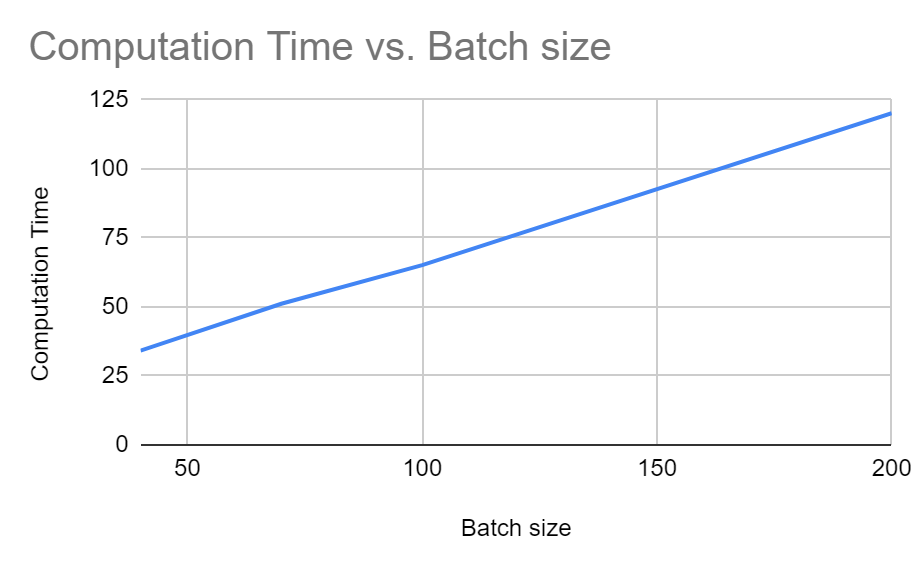
The results are as follows:

* The time taken for processing images from the database and training by optimizing weights of the neural network : **34 seconds**
* Accuracy of test cases : **97.08%**

The loss vs number of epoch graph is given below :



The computation time and the accuracy measured are compared as we vary the batch size by conducting various other experiments by only varying the batch size while keeping other parameters the same:



From the above graphs we can clearly see that the accuracy saturates but the computation time increases steadily as we increase the batch size, hence we have to find a good batch size which is within the computational constraints.

One more interesting observation is the batch size increase is less efficient as compared to number of loops increase in terms of increasing accuracy while keeping the computation time low, which is clearly visible from the results.

Experiment 3: In this experiment we change the number of training images (double them) and observe the change in performance and accuracy while the rest of the parameters are kept the same as base case :

Number of Images used for training : 10000

Number of Epoch : 50

Batch size : 100

Number of Loops : 10

The results are as follows:

* The time taken for processing images from the database and training by optimizing weights of the neural network : **83 seconds**
* Accuracy of test cases : **99.01%**

We can see that there is a slight increase in the accuracy but a significant increase in the time taken as there is an increase in the amount of time taken for processing images as well as training the network.

Experiment 4: The final experiment is to vary the architecture by changing the width of the network that is changing the number of neurons in the hidden layer. The following parameters are changed in the hidden layers :

* Number of neurons in Hidden layer 1: **128**
* Number of neurons in Hidden layer 2: **64**

The training parameters used are as follows:

Number of Images used for training : 5000

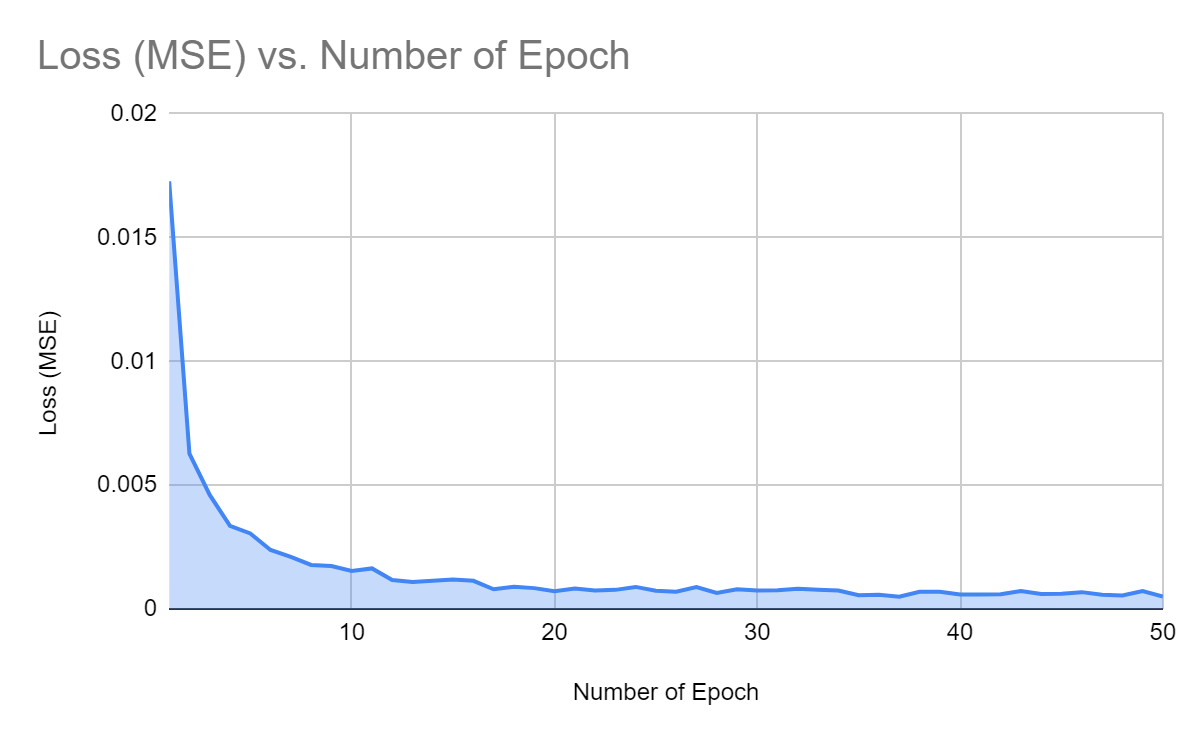
Number of Epoch : 50

Batch size : 100

Number of Loops : 10

The results are as follows:

* The time taken for processing images from the database and training by optimizing weights of the neural network : **113 seconds**
* Accuracy of test cases : **99.00%**



We can see that there is a very slight increase in accuracy but a very significant increase in the computation time. We can see that the loss has saturated at around 0.00025 which is almost the same as the base case.

**Conclusion**

From the above experiments we can conclude the following :

* The architecture of the network is pivotal to the performance of the network i.e number of layers and width of each layer greatly affects the accuracy and the training time of the network.
* The number of loops in the training process is directly proportional to the computation time and accuracy, but the accuracy saturates after reaching a certain value.
* The batch size in every epoch is also directly proportional to computation time and accuracy but the accuracy in this case also saturates as it reaches the high values (99%).
* From the experiments it looks like increasing the number of loops in training is more efficient than increasing batch size in reducing errors while keeping the computational time of training under control.
* Increasing the number of training images, the amount of time required to train the network increases as well as the accuracy which is not very proportional but kind of flattens out.
* The more the number of Epoch, the better the training is done, and the training accuracy saturates after a certain number of Epochs although the computational time increases proportionally.
* Overfitting is also a real problem where the network only works well on the training set whereas does not perform well on the test set, which would happen when the depth of the network is small.

**Alternate Approaches**

Some of the other alternate approaches to solve the problem of digit recognition using image

* Convolutional Neural Network
* K-Nearest Neighbours
* Support Vector Machines
* OCR Python

**Reference Links**:

http://yann.lecun.com/exdb/mnist/  
http://ramok.tech/2017/11/29/digit-recognizer-with-neural-networks/  
<https://www.youtube.com/watch?v=d3OtgsGcMLw&list=PLgomWLYGNl1dL1Qsmgumhcg4HOcWZMd3k&index=1>

**Jahnavi Peri(001388737),**

**Mansi Agrawal(001356676),**

**Anish Singh(001388384)**