DeepLearning (CPSC-8430)
Homework -1(Report)

GitHub Link:https://github.com/sangaraju-01/DeepLearning-CPSC8430-.git

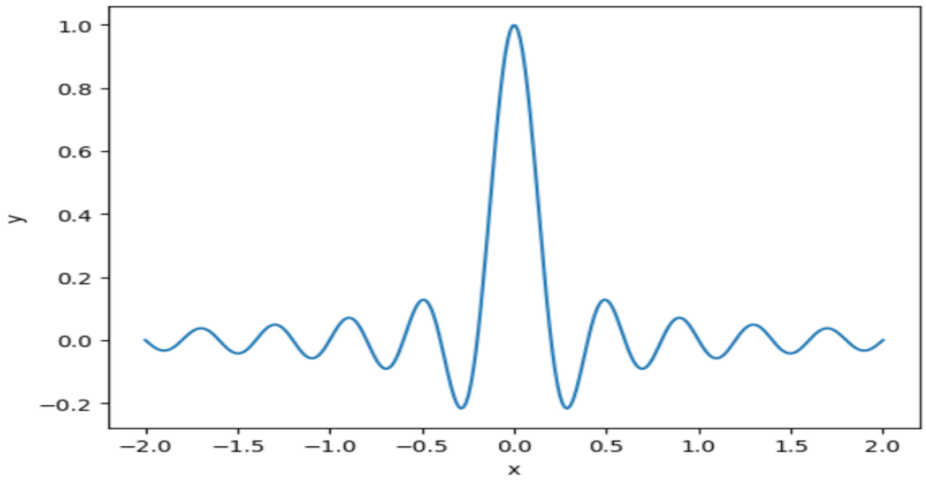
(1-1) Deep vs Shallow:

Task1-Simulate a Function

Three distinct models, each with a unique set of parameters, were trained on two different functions. The First model, Second model, and Third model are fully connected neural networks with seven, four, and one layers, respectively, with 571, 572, and 572 parameters. Both models exhibited loss function convergence after a reasonable number of training sessions. Figure 1 illustrates the mean squared error of the three models (First model, second, and Third model) during the arcsinh(x) simulation training. Convergence is achieved after just 250 iterations. These two approximated functions were y = cos(x) and y = arcsinh (x). There were **0.001** learning rates for each.

Function: sin(5*pi*x) / 5*pi*x

Below is the function plot for the same function:



Within a sufficient number of training rounds, the loss functions of all models for the two functions converged. Figure 1 below shows the Mean Squared Error (MSE) of three models (deep, intermediate, and shallow) for each epoch during the training for the simulation **(sin(5*(pi*x))/((5*(pi*x))**. After **1750 iterations**, convergence is reached. The anticipated and actual values for the function (sin(5*(pi*x))/((5*(pi*x)) are shown in Figure 2.

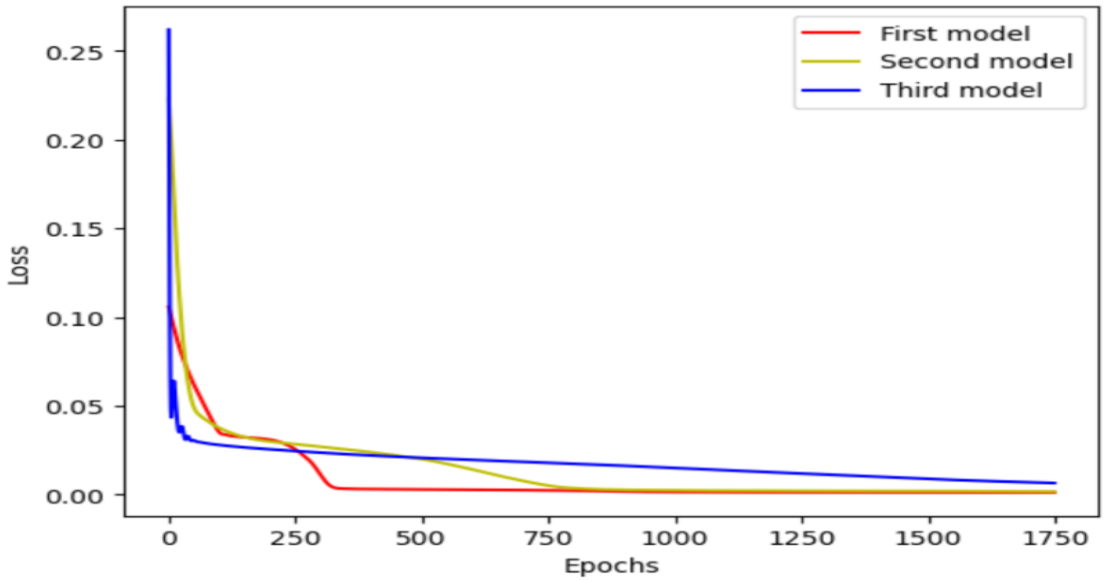


Figure.1

In the above Figure, we can observe that It takes just 250 iterations to reach convergence.

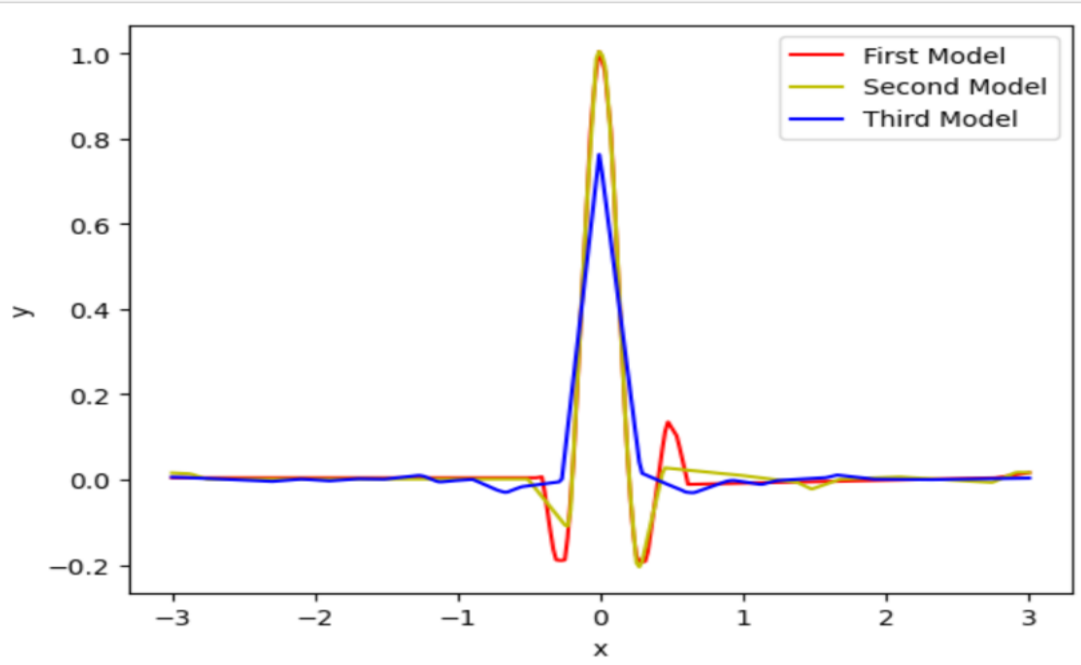
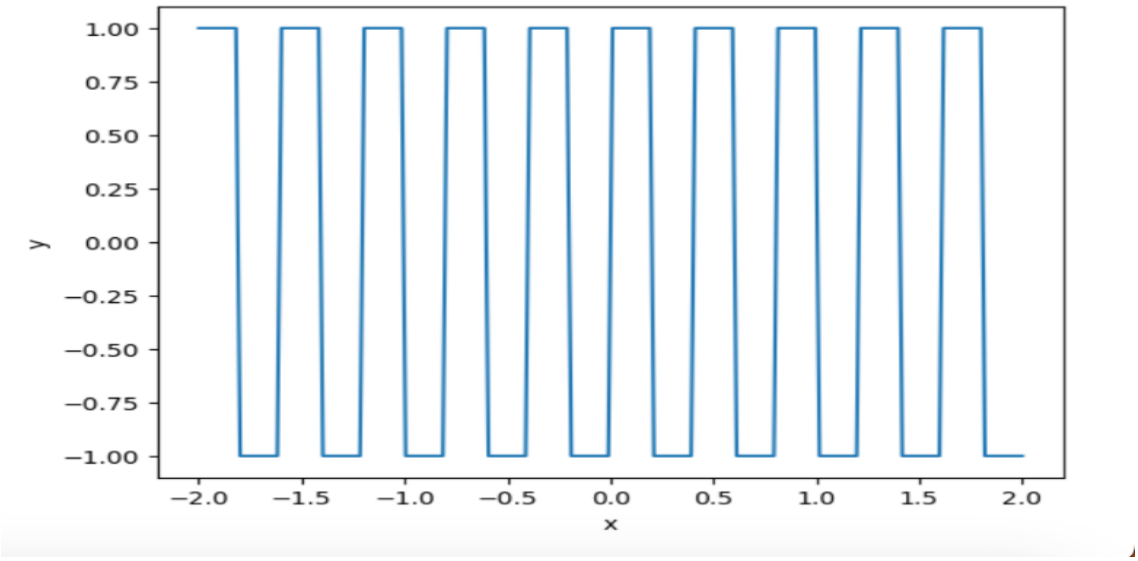


Figure.2

Result: The first and second models converge more quickly than the third model, which approaches its maximum number of epochs before converging. The first and second models learn the function significantly better than Model 3 and have lower loss values, as seen by the graph. This is evidence of the various models' layer counts, which allowed the models with more layers to learn more quickly and effectively.



Function.2 Plot

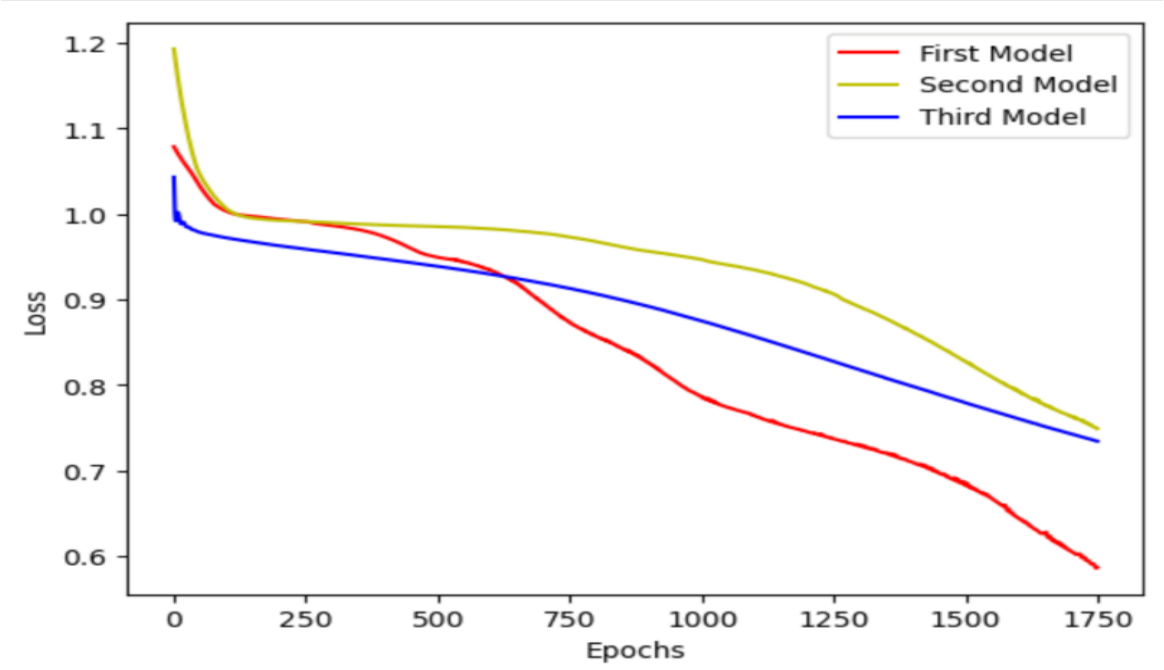


Figure.3

In the above figure, we can observe that the first and second model has more convergence than the Third model.

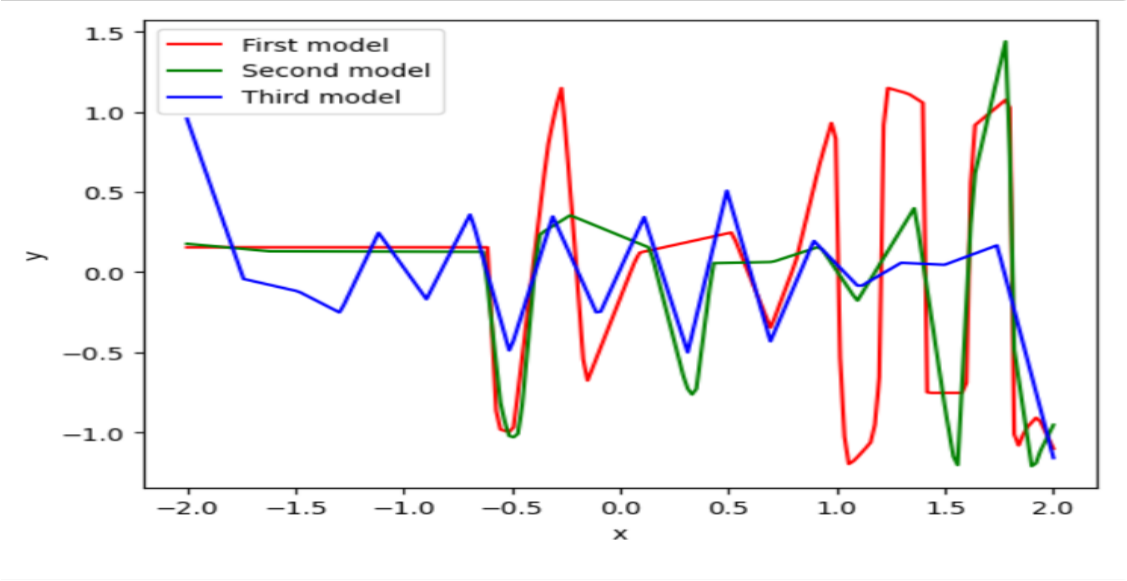


Figure.4

Result:

Since the function for the three models is challenging to train, all models approach the maximum number of epochs before convergence. First Model slightly outperforms the Second model, has the lowest loss and appears to be the best learner. The third Model fails to converge throughout the range and minimize its loss to a low value.

Task 1-2: Train on Actual Tasks

The models in this Task are trained on MNIST Dataset.
(https://www.kaggle.com/datasets/hojjatk/mnist-dataset)
Training Set: 60,000 Testing set: 10,000

Convolutional neural networks (CNN) were used to create two models utilizing 60,000 training and 10,000 test data from the MNIST dataset, with a batch size of 10. Testing data is not jumbled, but training data is. The two models are completely interconnected, and to pool the data, we utilized the max pool function using RELU(**RECTIFIED LINEAR ACTIVATION UNIT**) as the activation function.
Each tier of every network has a different number of nodes. The average **learning rate was 0.001**. The loss that happened during the training of the First models and the Second is shown in figure 5. For two models, convergence is often attained after **40 epochs**.

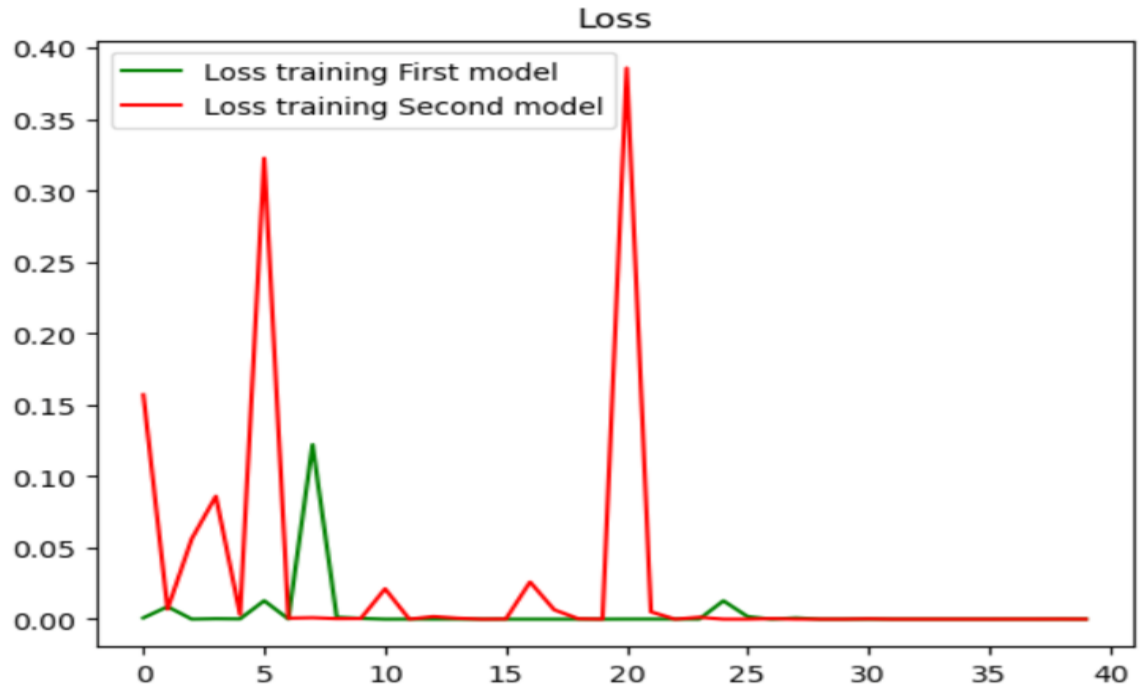


Figure.5

Figure 5 shows the learning process for the Two models. All models reach convergence after around 40 epochs. The graph still contains some noise, though.

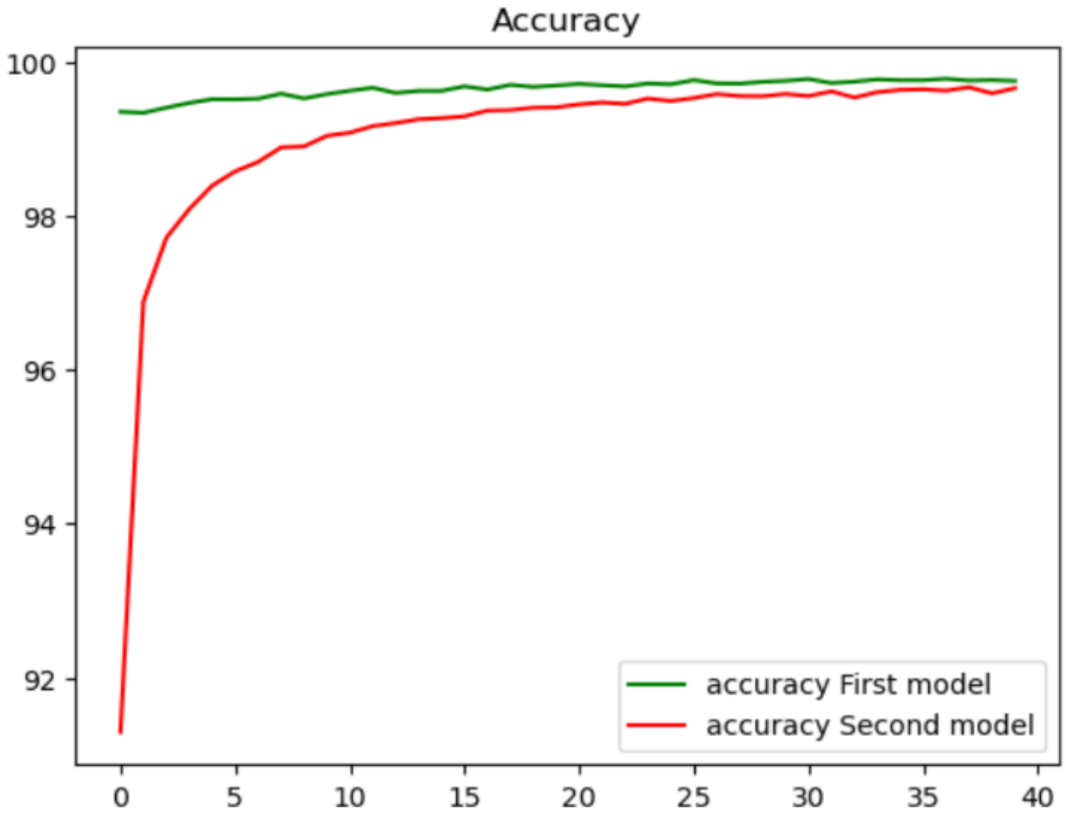


Figure.6

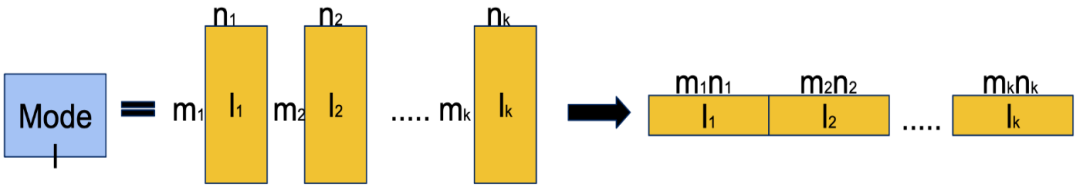
Result:

The accuracy of the two models on training and test sets is shown in Figure 6 during model training. Two models achieve 99% accuracy on training data while Maintaining around **90-96 percent accuracy** on testing data. As predicted, we can see that the two models regularly outperform each other on training data compared to test data. Naturally, the accuracy shows a similar tendency, with model 1 outperforming model 2 with greater training accuracy.

Part 2: Optimization
Task 1: Visualize the Optimization Process

The models in this Task are trained on MNIST Dataset.
(https://www.kaggle.com/datasets/hojjatk/mnist-dataset)
Training Set: 60,000 Testing set: 10,000

To simulate the function (sin(5*(pi*x))/((5*(pi*x)), a deep neural network with three fully connected layers and **57 parameters** was trained. The network was optimized using the Adam optimizer. The model's weights were regularly gathered throughout eight training series, each with **30 training epochs**. The Adam optimization approach is a stochastic gradient descent variant that has lately gained more popularity for deep learning applications in computer vision and natural language processing.



The above Picture shows the Collecting Parameters of the Model.

PCA application results in dimension reduction. All models have a **learning rate of 0.001**.

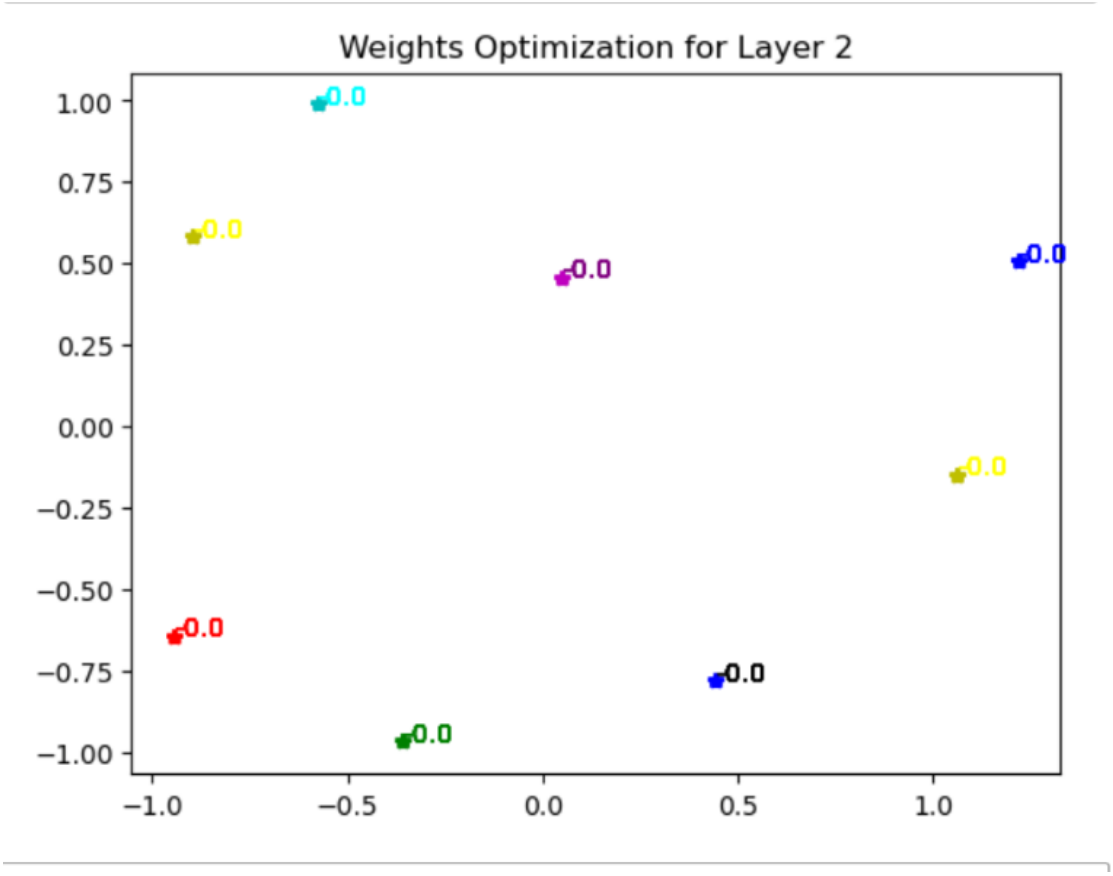


Figure.7

The above Figure demonstrates how the weights inside the network learned during training are optimized through the backpropagation of Layer 2.

Figure.8

The above figure shows how the network's weights are optimized for the entire model after learning them during training. Fine-tuning takes place over time when the weights are gradually adjusted during training.

Task 2: **Observe Gradient Norm during Training**

The models in this Task are trained on MNIST Dataset.
(https://www.kaggle.com/datasets/hojjatk/mnist-dataset)
Training Set: 60,000 Testing set: 10,000

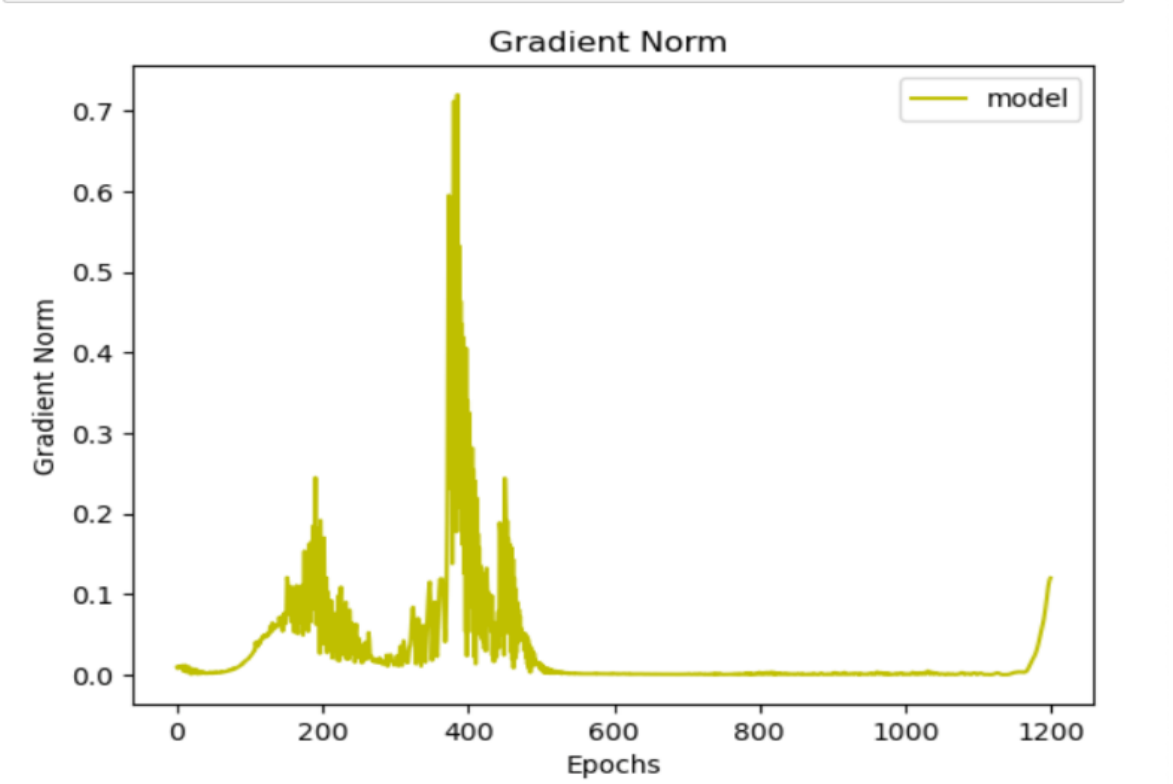


Figure.9

The gradient norm for each epoch is displayed in figure 9. The slope variations in figure 10 and each of the spikes in figure 9 are related. Graph 9 exhibits an anomaly.

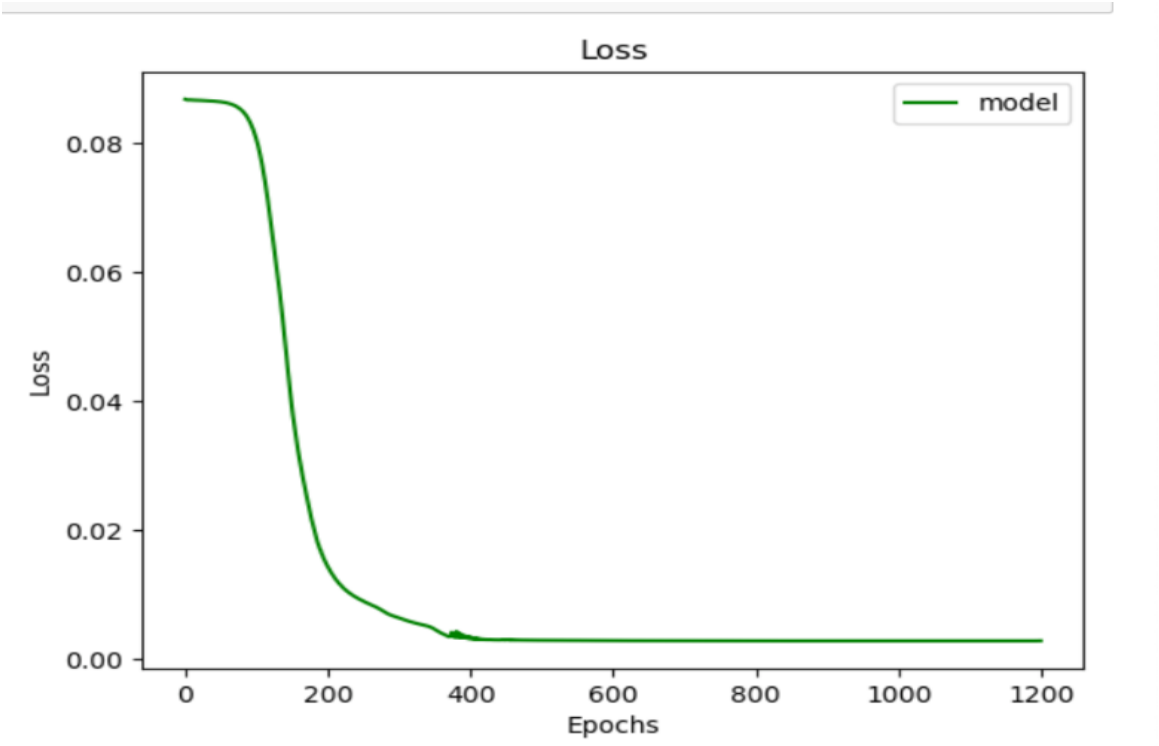


Figure.10

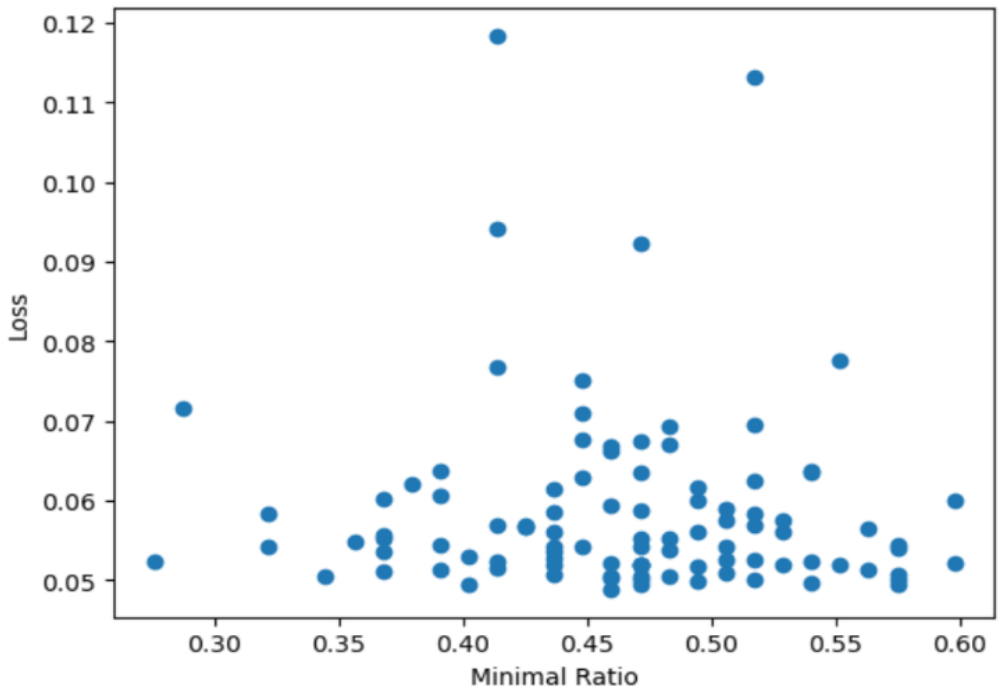
Each training epoch's loss for the model is depicted in Figure 10. Here we have trained with 1200 epochs. The slope of the line in Figure 10 varies when the model starts to learn more quickly or more slowly, and this shift is represented by a spike or other irregularity in the line graph in Figure 9.

Task 3: What Happens when Gradient is Almost Zero?

A function's slope is flat, and the gradient is close to zero when this occurs. This indicates that the function is changing very little in that direction. This is crucial in optimization algorithms since it indicates if the process is getting near to the function's minimum or maximum. An example of this is the gradient descent method.

In this, we have used two hidden layers and 32 layers for the neural network training I have used Adam Optimization.

Output:



Part 3: Generalization

Task 1: Can the Network fit Random Variables

The models in this Task are trained on MNIST Dataset.
(https://www.kaggle.com/datasets/hojjatk/mnist-dataset)
Training Set: 60,000 Testing set: 10,000

For this Task, the MNIST(**Modified National Institute of Standards and Technology database)** dataset was selected as the data for both training and testing. A two-layer feed-forward Deep Neural Network was designed and trained 35 times on the MNIST dataset. The learning rate for all models was set at 0.001, and the Adam optimization algorithm was utilized to optimize the neural network.

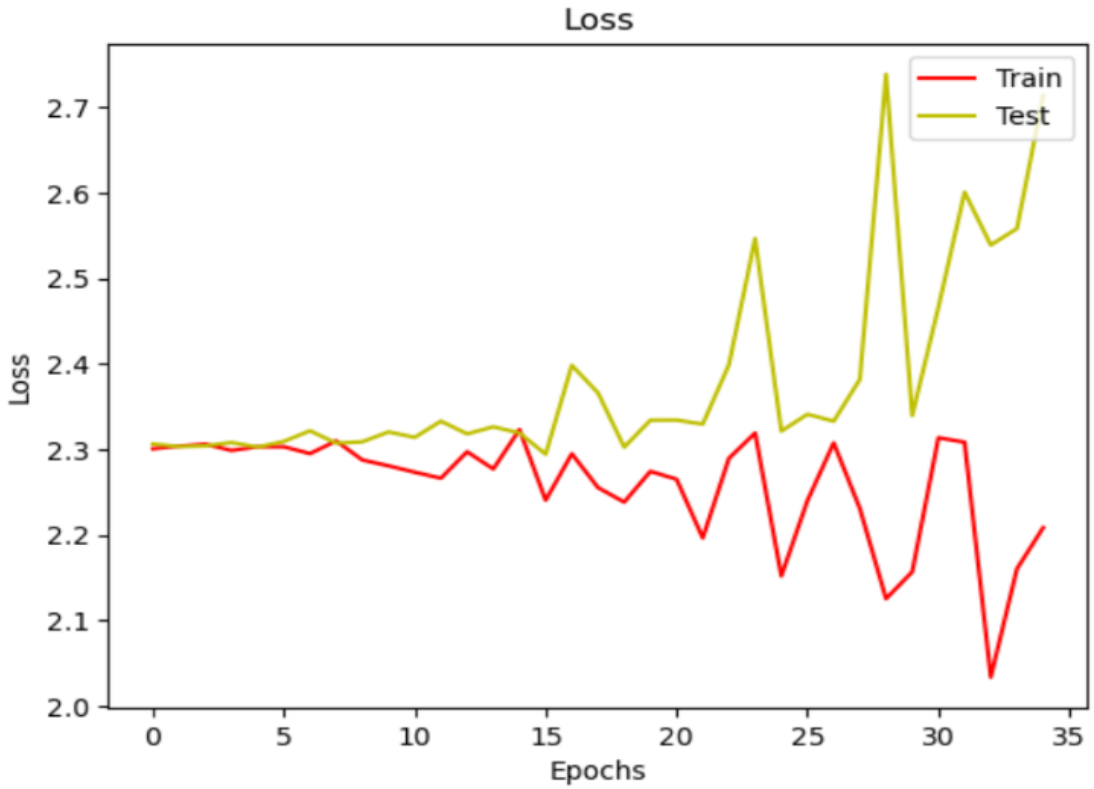


Figure.11

In the above figure, the network may perform well on the training set and have a low loss, but it fails to extend this success to new inputs, as evidenced by the increase in loss for the testing set, as seen in Figure 11. This means that the network is not capable of fitting all labels.

Task 2: **Number of Parameters VS. Generalization**
The models in this Task are trained on MNIST Dataset.
(https://www.kaggle.com/datasets/hojjatk/mnist-dataset)
Training Set: 60,000 Testing set: 10,000

Both training and testing were done using the MNIST dataset. Ten FeedForward DNNs with two hidden layers each were constructed. The model has between a few thousand to a million parameters. A learning rate of 0.001 was chosen for each. The Pytorch Adam optimizer was used to make the network more efficient.

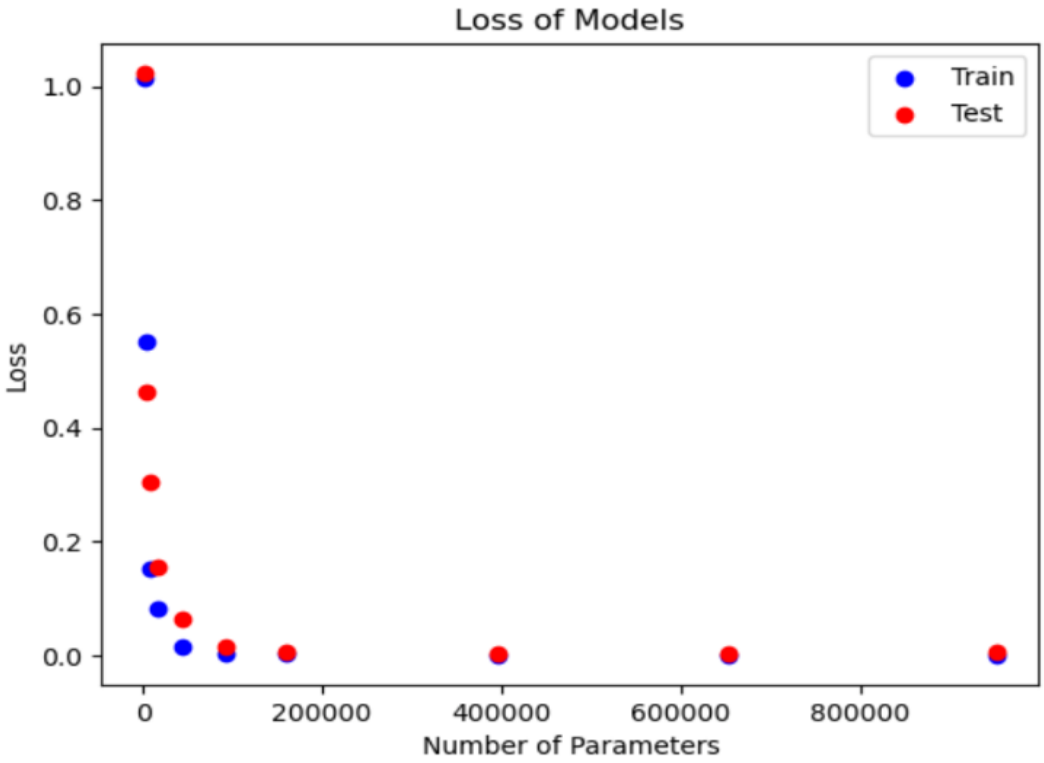


Figure.12

In the above figure, we can observe the loss of models. In this, we used approximately One million Parameters.

Figure 12 illustrates how adding more parameters to a model improves accuracy while lowering loss. At this point, our model won't get any better. After this point, a model barely slightly benefits from adding new parameters. But after a certain number of parameters, perhaps 200,000, the model's progress from one training cycle to the next becomes negligible.

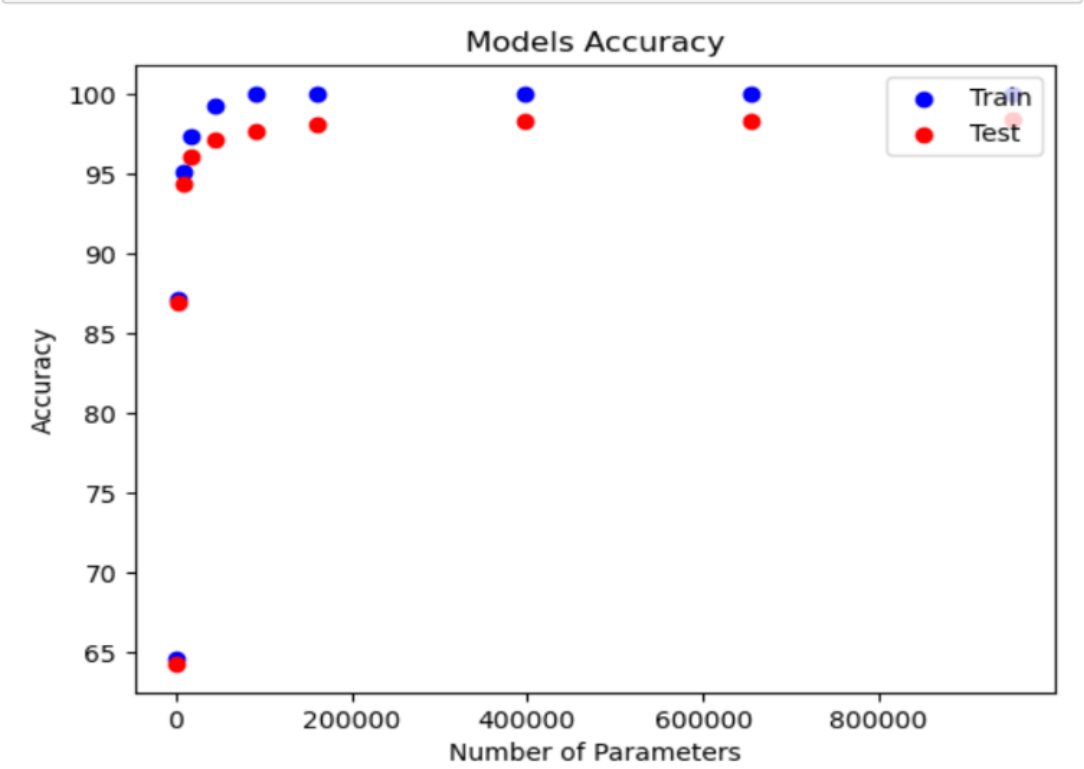


Figure.13

The above figure shows the accuracy of the models. When the models are applied to the training set compared to the testing set, a difference in the performance of the models can be seen. When the models are tested against the training set, they have greater accuracy and lower loss values. Given that the test set comprises data the models have never seen before, this is a typical occurrence. However, increasing the model's parameters partially closes the performance difference between the training and testing sets.

Task 3: Flatness VS Generalization

The models in this Task are trained on MNIST Dataset.
(https://www.kaggle.com/datasets/hojjatk/mnist-dataset)
Training Set: 60,000 Testing set: 10,000

The data for training and testing was chosen from the MNIST collection. The batch sizes used to generate two Deep Neural Networks were 64 and 1024. The Adam Optimizer was used to optimize the Neural Network, and the learning rate for all models was set at 0.001.Theta1 for the First model and Theta2 for the Second model is represented by the parameter "Alpha," which represents the linear interpolation between the two sets of parameters.

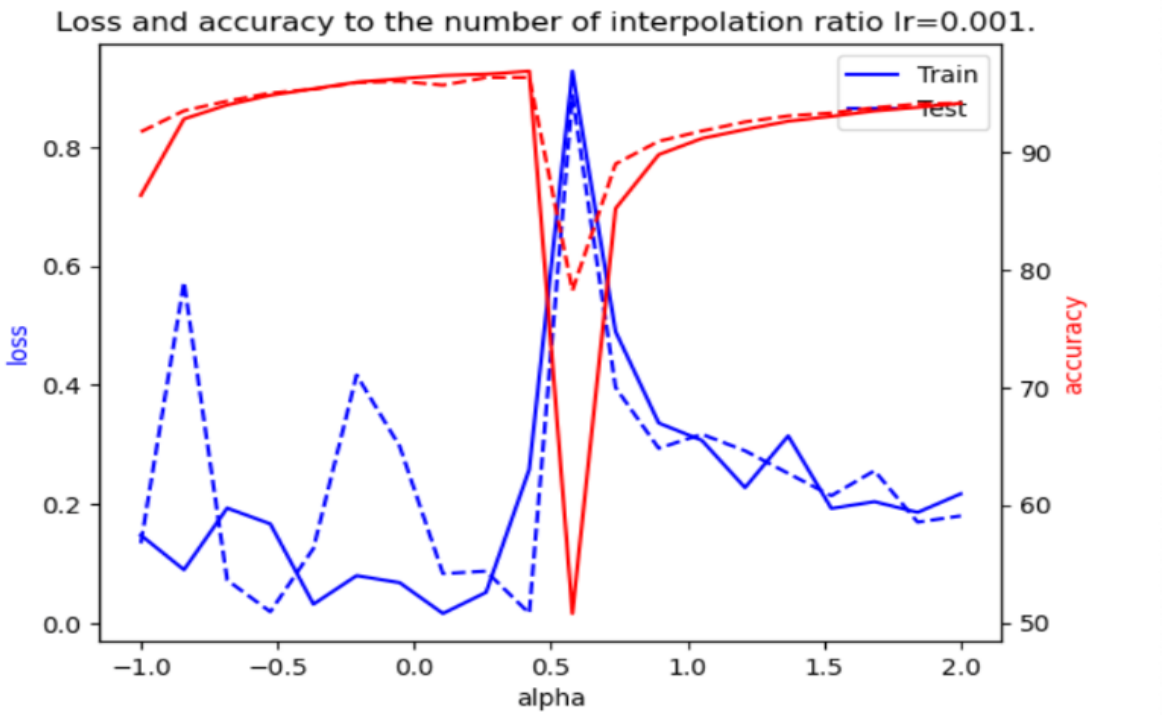


Figure.14

In figure 14 above, the loss, accuracy, and linear interpolation alpha during the training of two models with a learning rate of 0.001 are depicted.

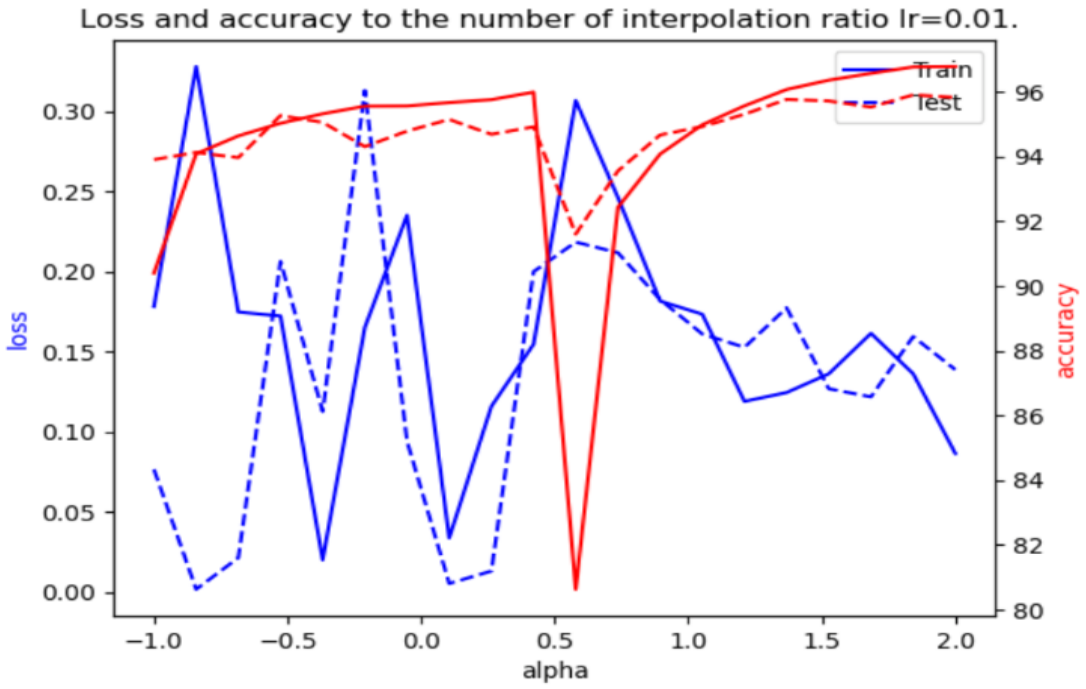


Figure.15

whereas, Figure 15 shows the Loss and accuracy with a learning rate of 0.01.

Part 2: Flatness Vs Generalization

The module was tested and trained using the MNIST data set. Five identical Deep Neural Networks, each with two hidden layers and 16330 parameters, were trained in batches ranging from five to one thousand. For the optimization process, the Adam Optimizer is utilized, and all models have a learning rate of 0.001.

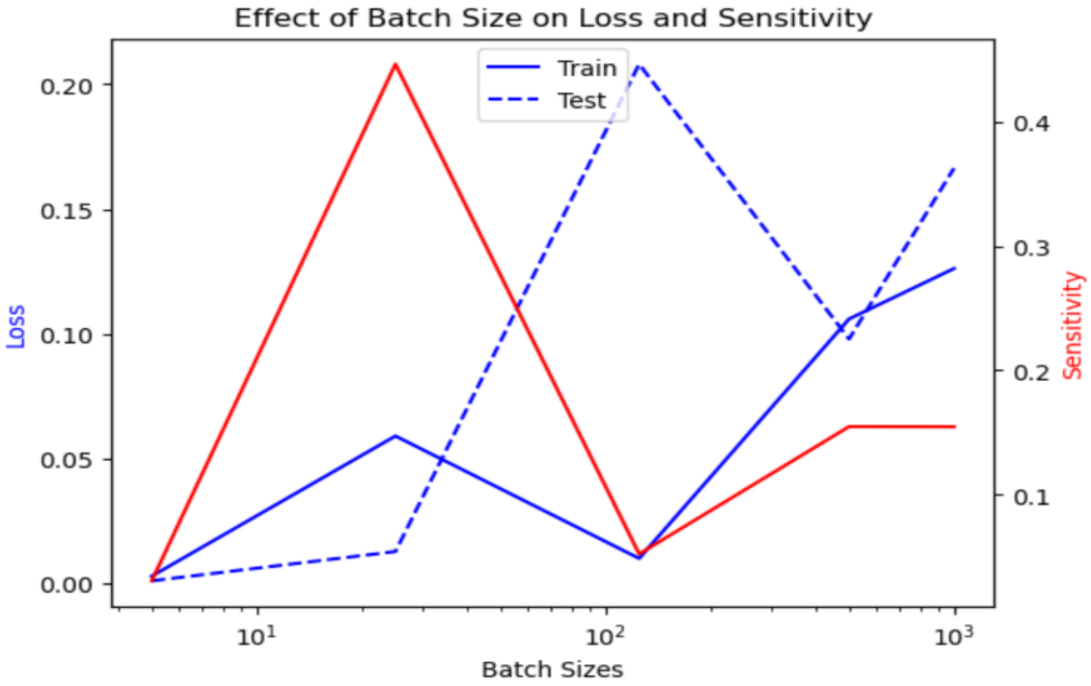


Figure.16

In the above figure, we can Observe the effect of batch size on Loss and Sensitivity.

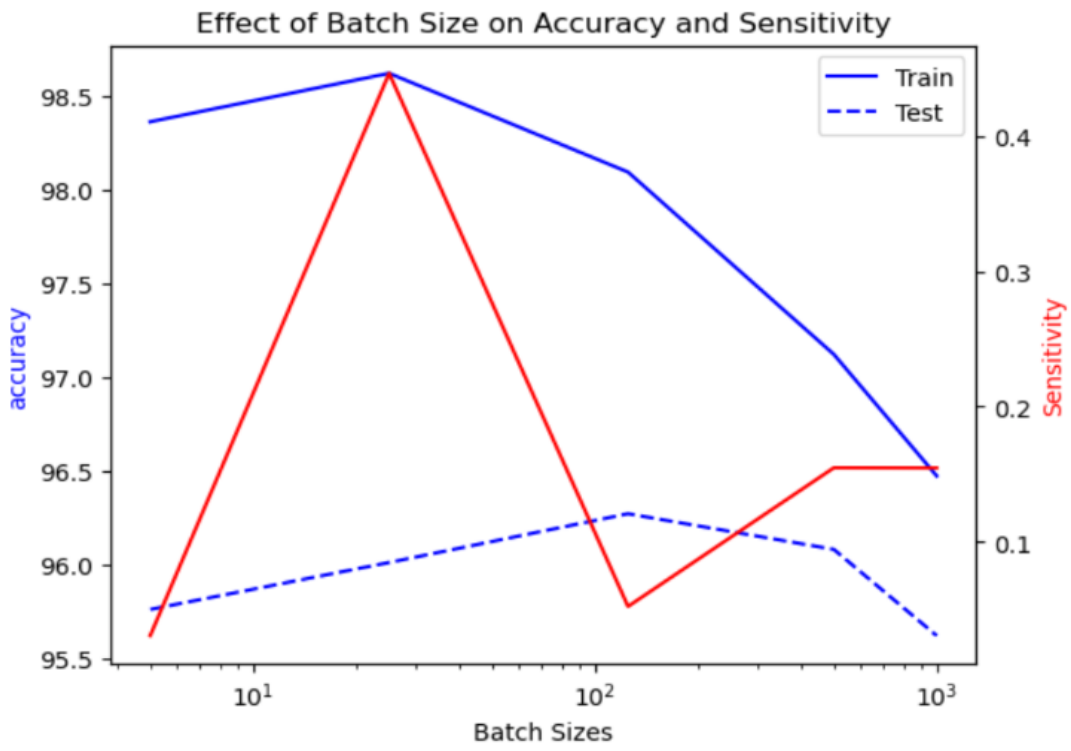


Figure.17

Above Plot shows the effect of Batch Size on Accuracy and Sensitivity Following training, the accuracy and loss for the training dataset and test dataset for each of the five models were determined. Following that, the norm of the gradient approach was used to assess the models' sensitivity.

Therefore, we conclude that a batch size of between 102 and 103 will provide the network with the greatest outcomes. The accuracy and sensitivity are shown to be impacted by the batch size, loss, and sensitivity in Figures 16 and 17. Sensitivity declines as batch size rise.