\

Name : Mithilesh Biradar
CUID : C12093342

CPSC-8430 Deep Learning HW 1

---

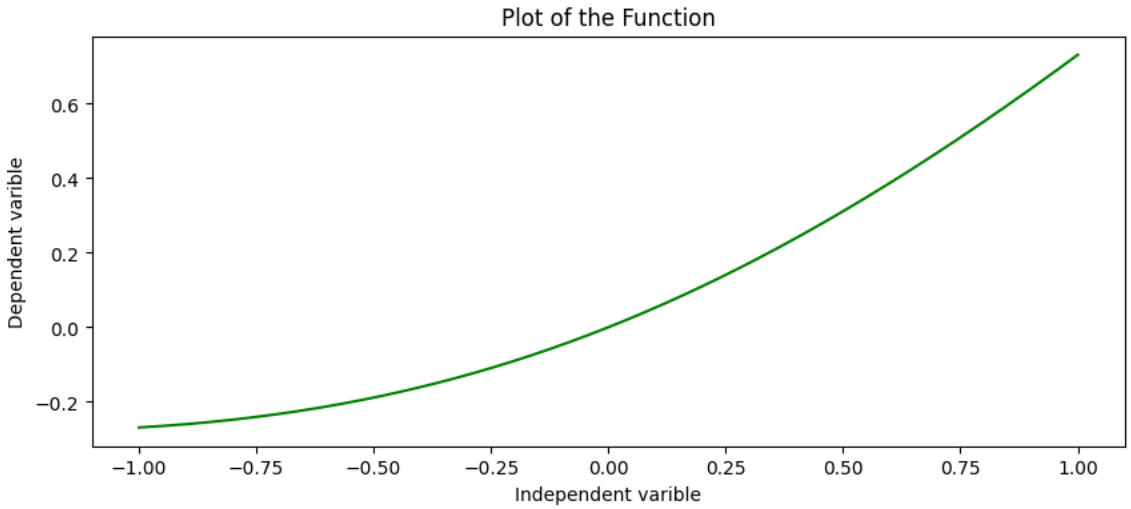
1 Deep vs. Shallow

1.1 Simulate a function:

First simulation function: $x \cdot \frac{1}{1+e^{-x}}$

Second simulation function: $\text{sgn}\left(\frac{\cos(10x)}{x}\right)$

First Simulation function: $x \cdot \frac{1}{1+e^{-x}}$



Model 1: Architecture:

7 dense layers with sizes:

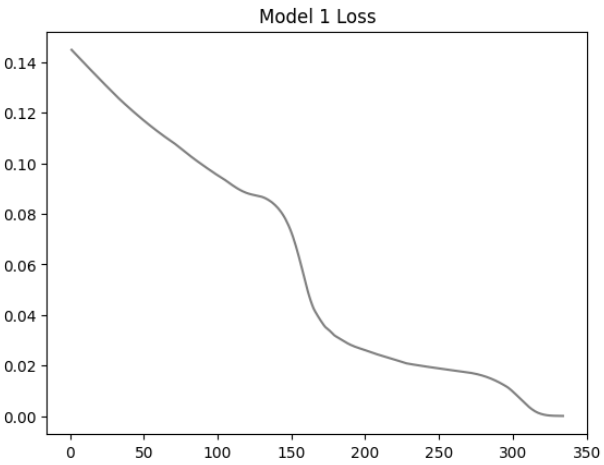
- Layer 1: 1 → 5
- Layer 2: 5 → 10
- Layer 3: 10 → 10
- Layer 4: 10 → 10
- Layer 5: 10 → 10

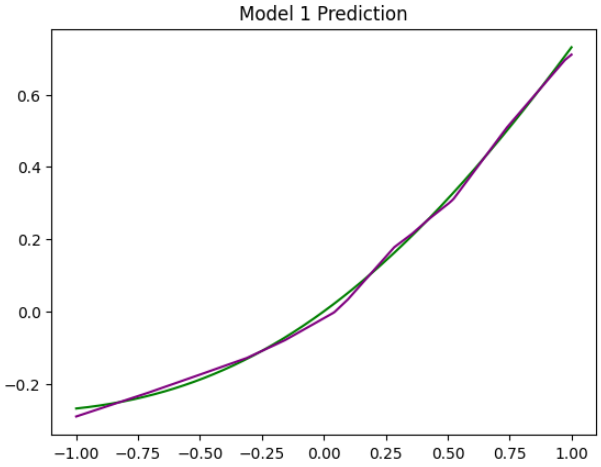
\

        - Layer 6: 10 → 10
        - Layer 7: 10 → 5

    Final prediction layer: 5 → 1

- Activation function: Leaky ReLU
- Total number of parameters: 571
- Loss function: "MSELoss"
- Optimizer: Adam
- Hyperparameters: Learning rate: 0.0011





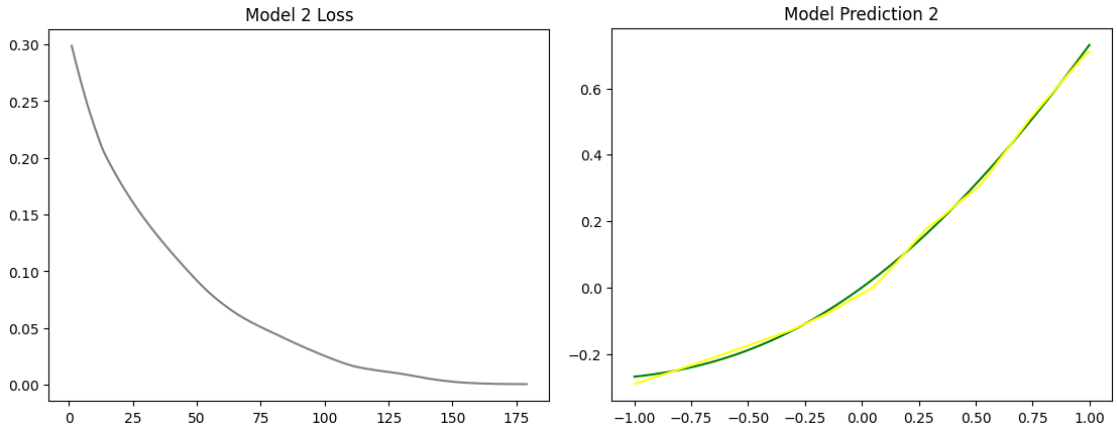
Model 2: Architecture:

- 4 dense layers with sizes:
    - Layer 1: 1 → 10
    - Layer 2: 10 → 18
    - Layer 3: 18 → 15
    - Layer 4: 15 → 4
- Final prediction layer: 4 → 1

- Activation function: Leaky ReLU
- Total number of parameters: Calculated based on the layer structure.
- Loss function: MSELoss
- Optimizer: Adam
- Hyperparameters:

\

- Learning rate: 0.0011



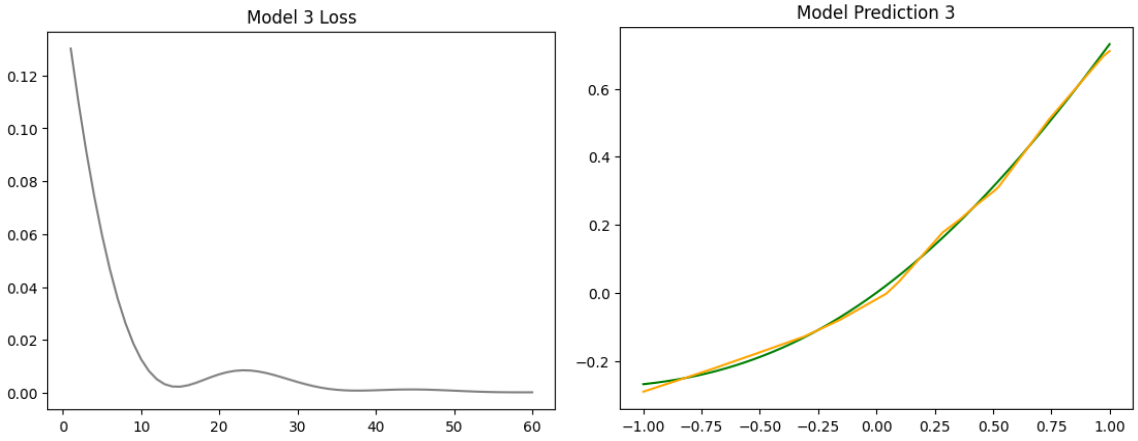
Model 3: Architecture:

1 dense layer: 1 → 190

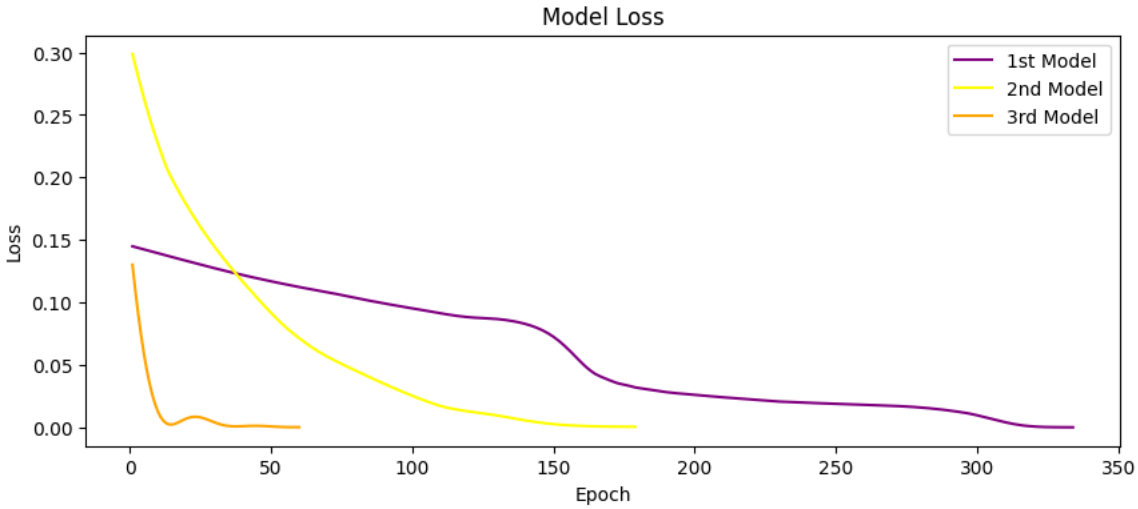
Final prediction layer: 190 → 1

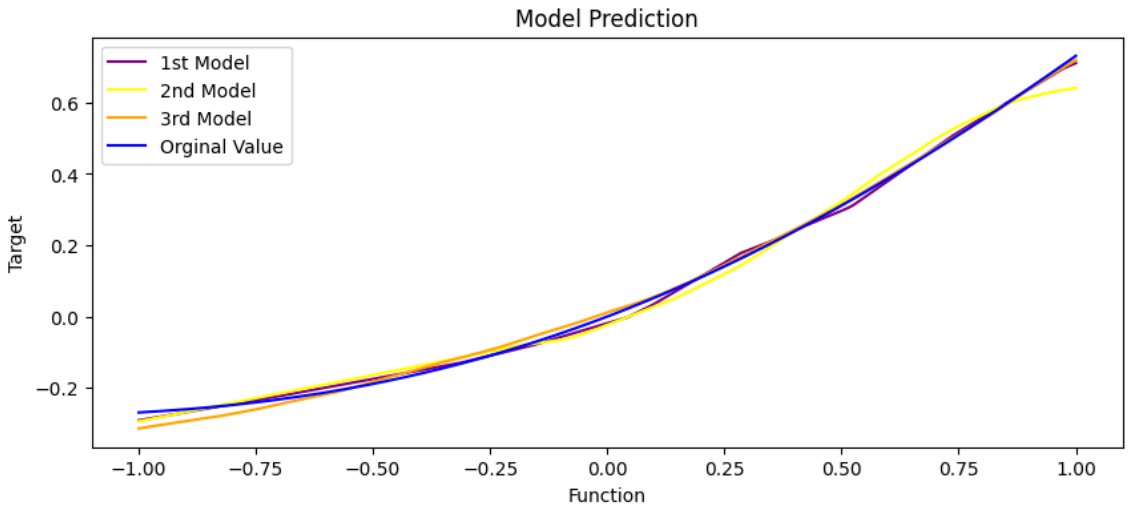
- Activation function: "Leaky ReLU"
- Total number of parameters: 190
- Loss function: "MSELoss"
- Optimizer: "Adam"
- Hyperparameters:

  Learning rate: 0.0011



\

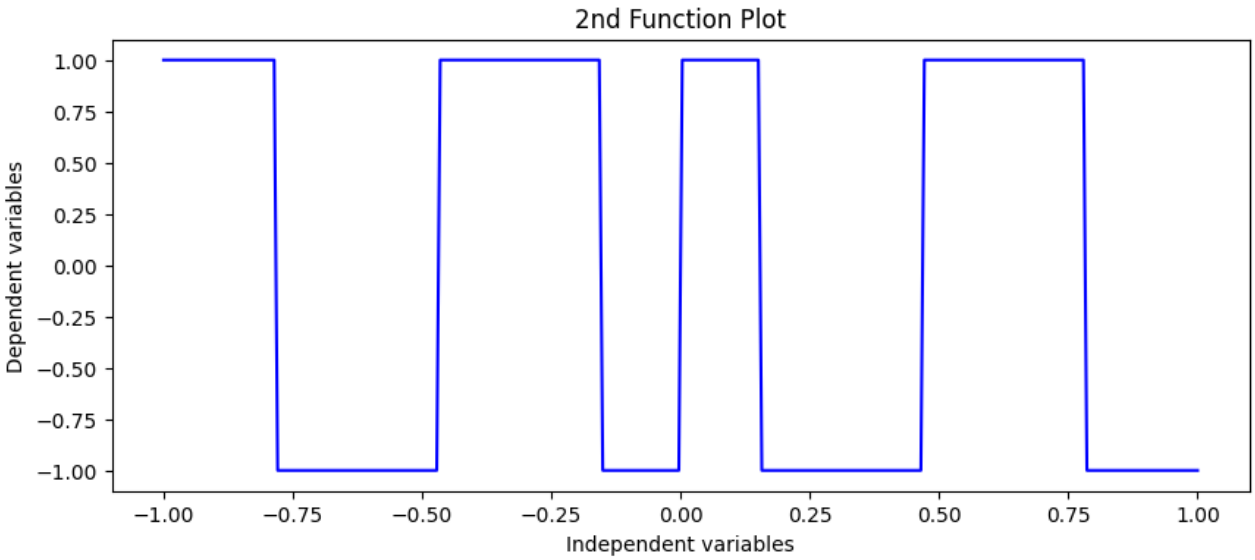




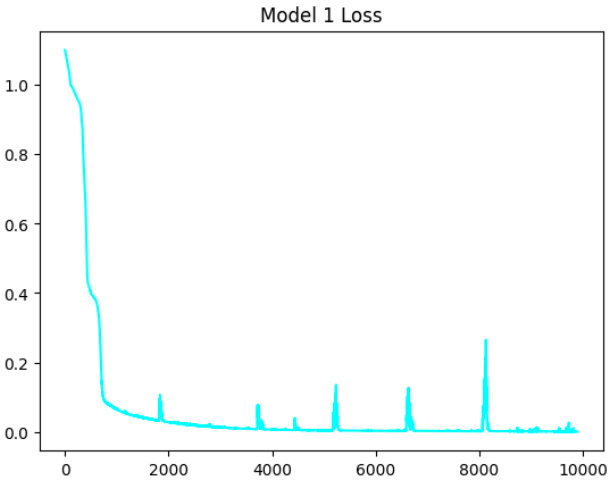
The first model shows a steady but slower decrease in loss compared to the others, with slight deviations in its predictions at the higher input range. This indicates it captures the function well but struggles with certain aspects. The second model has rapid loss convergence and strong alignment with the target function. The third model converges the fastest but shows more pronounced deviations, especially at the input extremes.

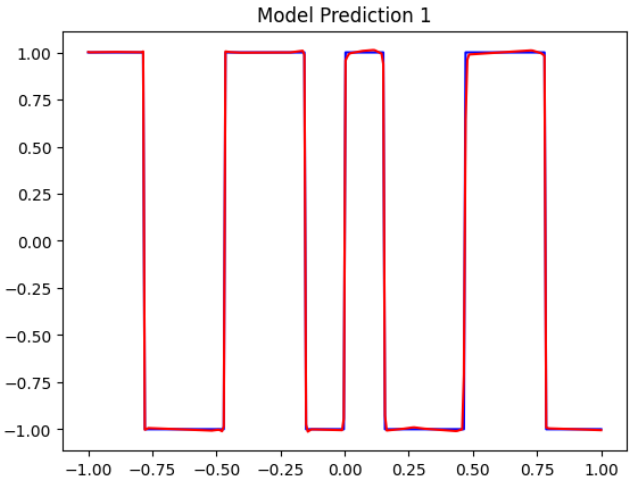
\

Function 2: $\text{sgn}\left(\frac{\cos(10x)}{x}\right)$



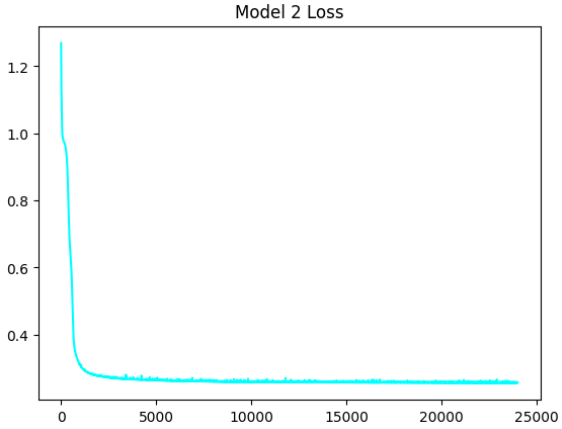
Model 1 :

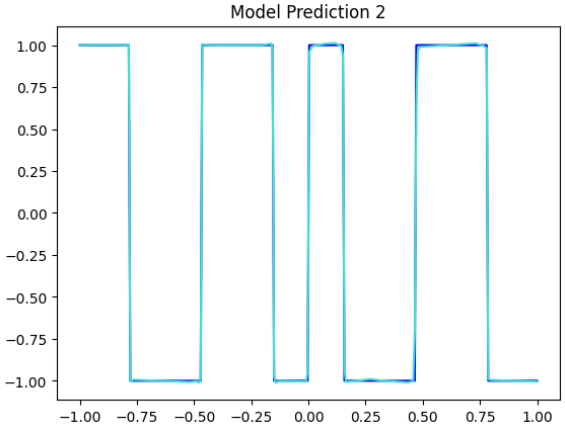




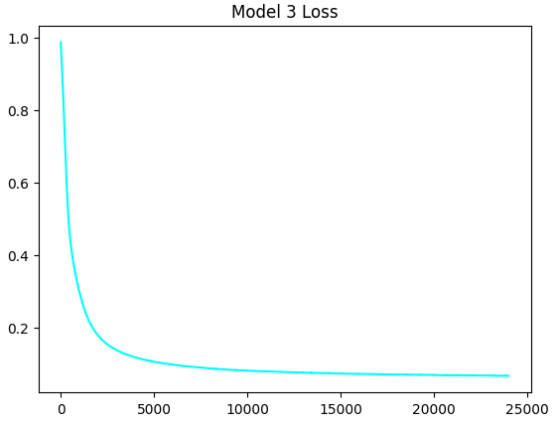
\

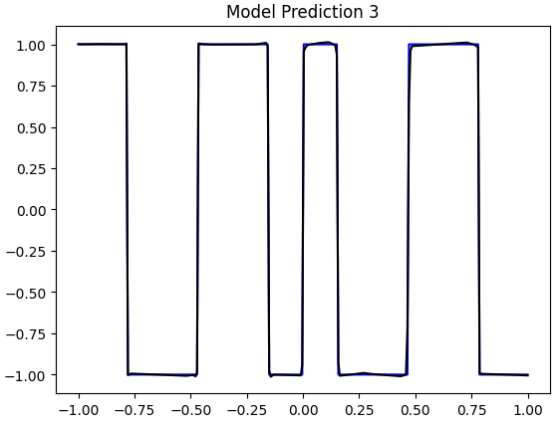
Model 2 :



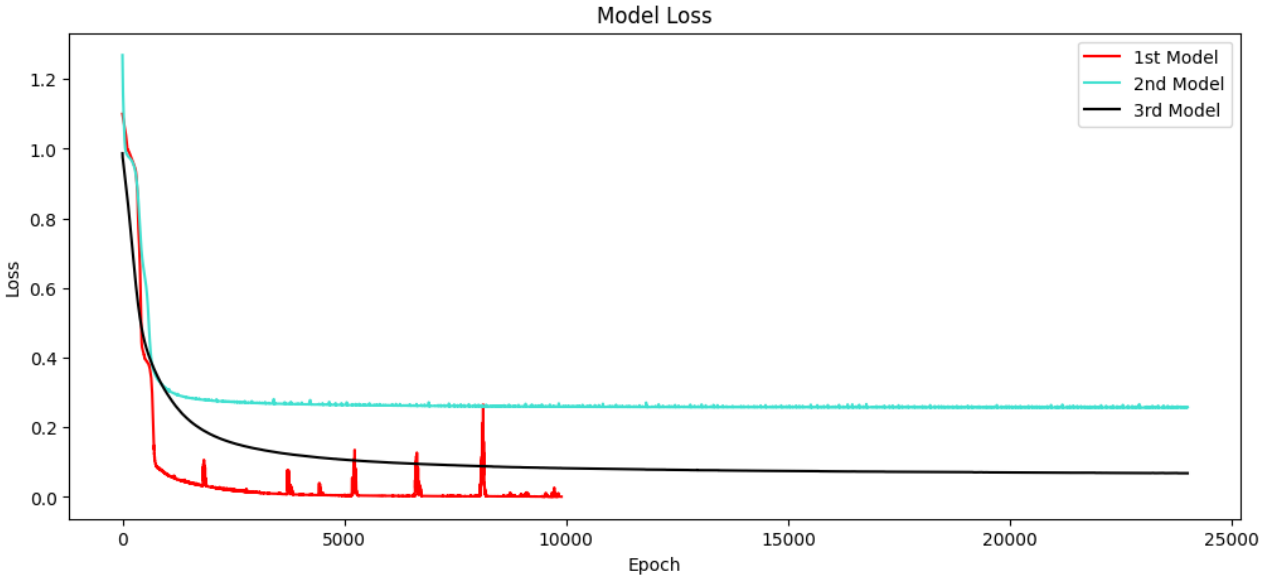


Model 3:

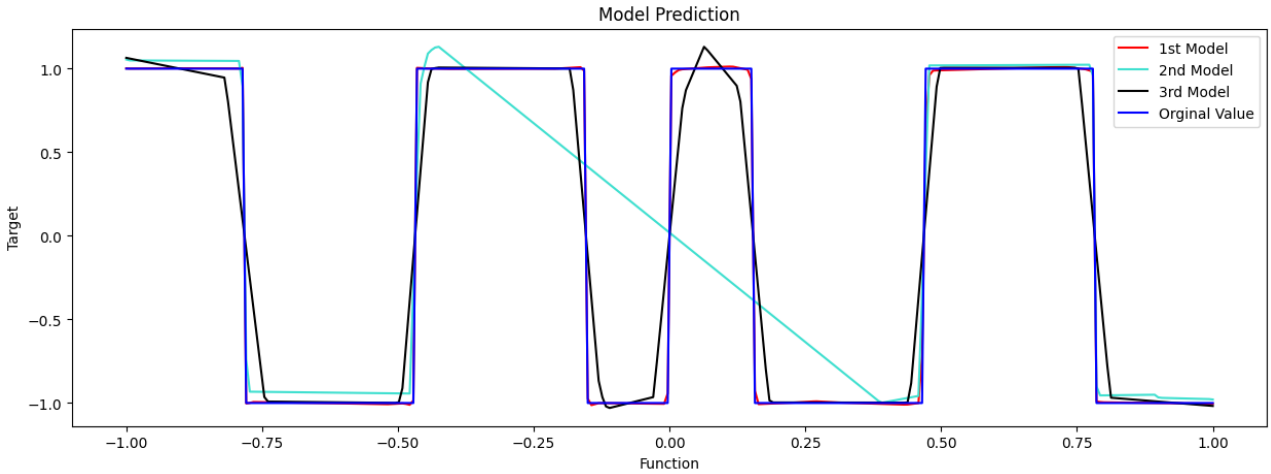




Comparative Analysis:

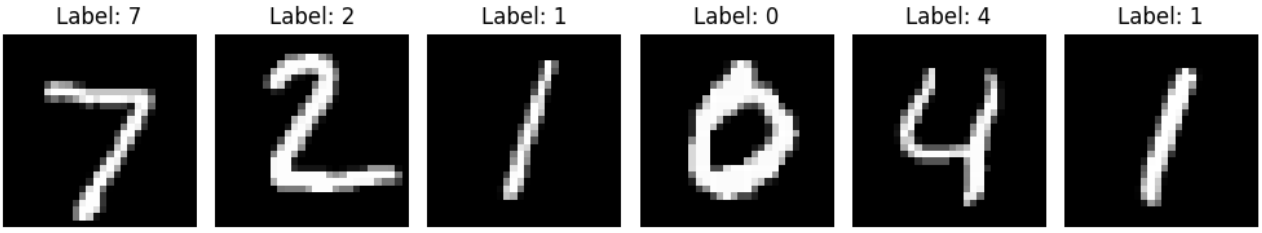


\


Model Prediction
1st Model
2nd Model
3rd Model
Orginal Value
Target
Function
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

\

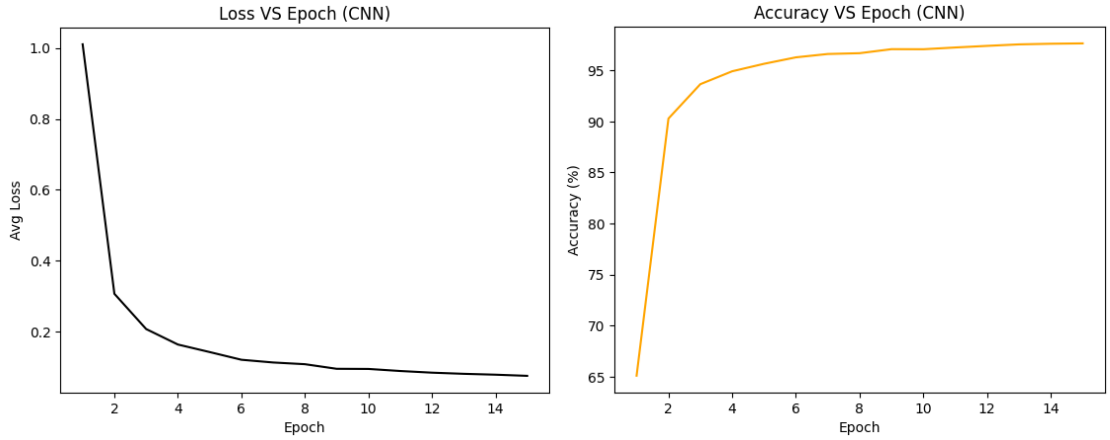
1.2 Train on Actual Task



CNN Model 1:

- Architecture: This model consists of convolutional layers followed by fully connected layers. It includes:
  - Conv2d layers = 2
  - Pooling layers = 2
  - Activation functions like ReLU
- Parameter numbers: 25,550.
- Performance:
  - Maximum accuracy: 97.64%
  - Final loss: 0.0580

Observations: This model converges relatively well, achieving high accuracy after multiple epochs of training.

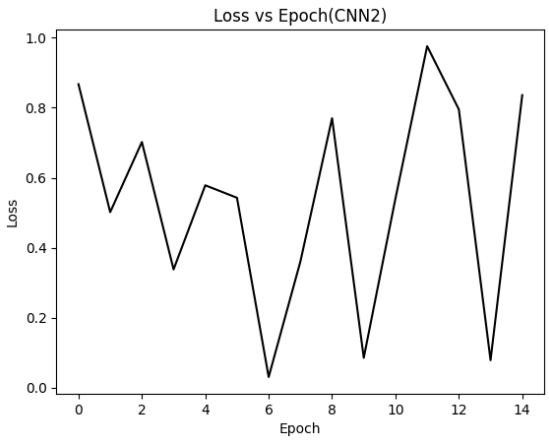


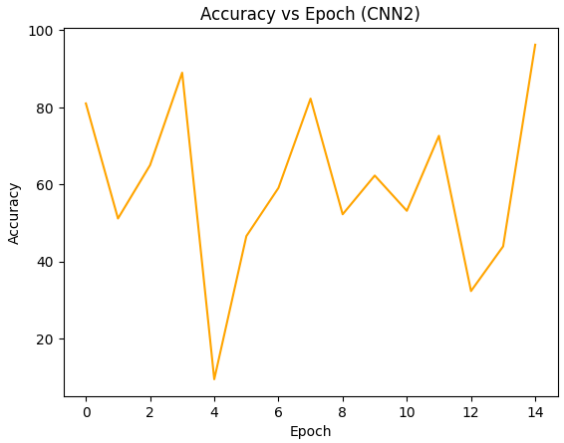
\

CNN Model 2:

- Architecture: Slightly modified convolutional architecture compared to CNN1.
- Performance:

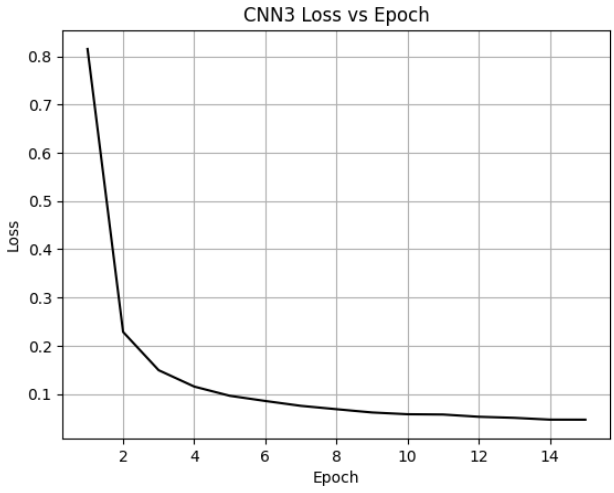
  This model also shows good convergence, with a loss decrease over epochs, and accurate predictions. The training logs indicate gradual improvements over the epochs.

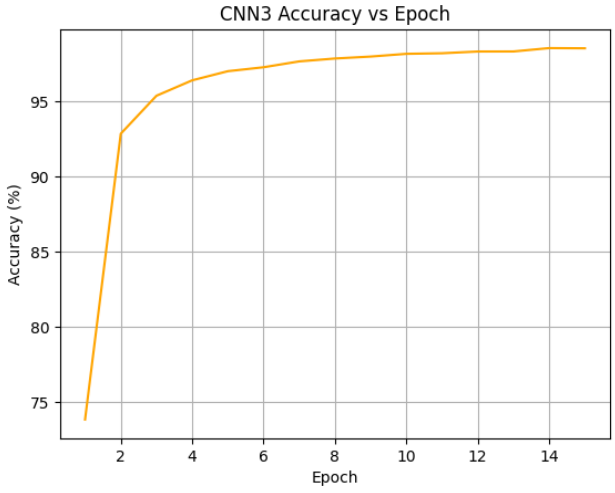




CNN Model 3:

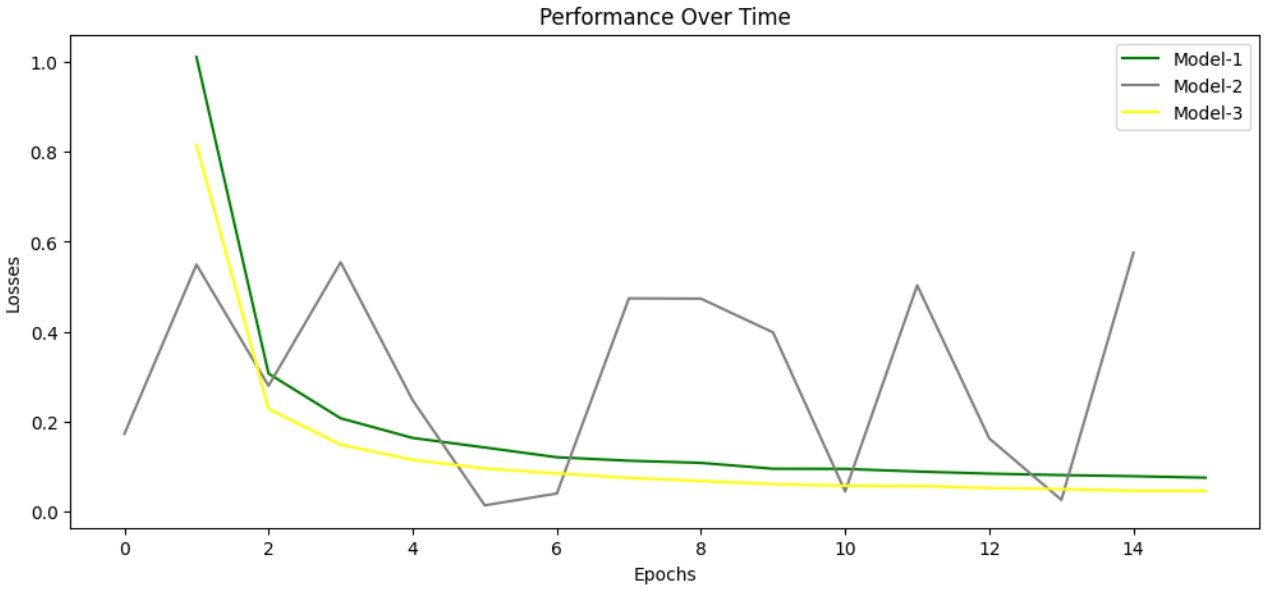
- Architecture: Includes convolutional layers with slightly larger capacity, utilizing more filters.
- Total number of parameters: 25,621.
- Performance:
  - Achieves the best results among the models, with an accuracy of 98.56% and a lower final loss (0.0451).





\

Model evaluations:



CNN Model 3 is the best performing model in terms of accuracy and loss, benefiting from its increased complexity. CNN Model 1 provides a solid baseline with fewer parameters and reasonable performance, making it a good candidate if computational resources are limited.

2 Optimization

2.1 Visualize the Optimization Process

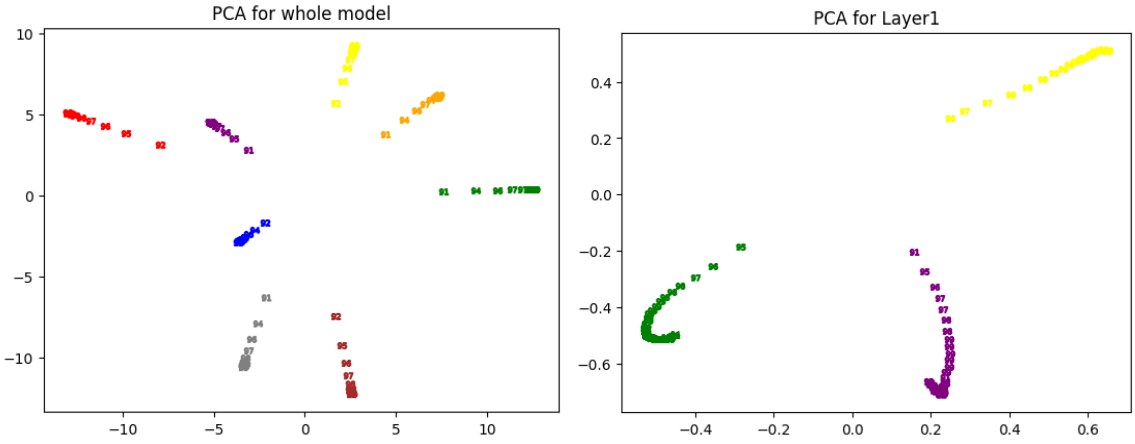
Model Description:

Model Parameters: The model is trained with Adam optimizer and for the loss function MSE is used.

Learning Rate: 0.0005

The model was trained for 8 cycles with 40 epochs, and I focused on tracking the parameters of one selected layer and the entire model over iterations.

\

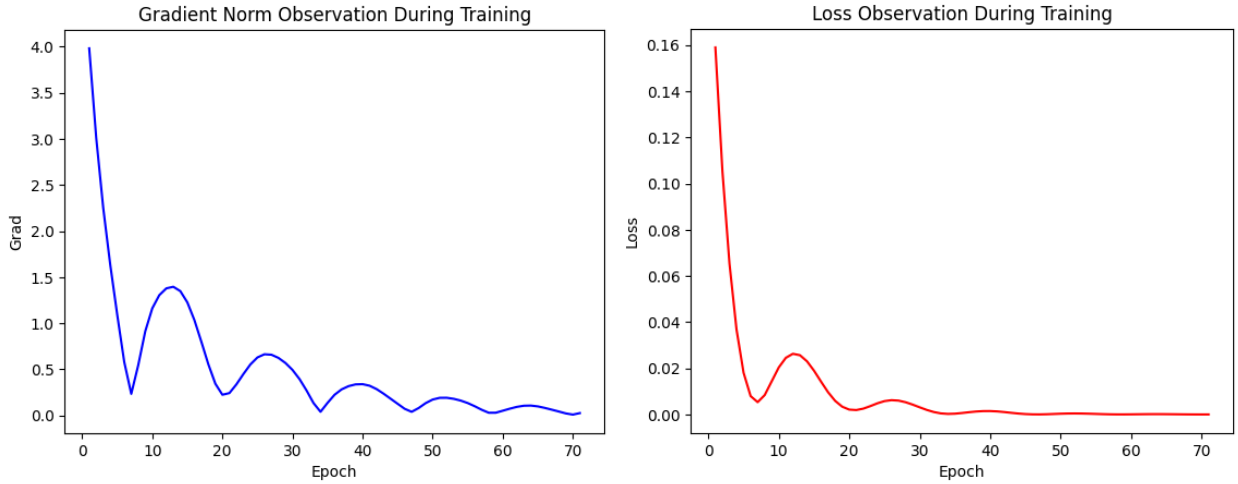


Observations:

The model parameters move noticeably at the training phase of the model but when the points are coming more closer suggests that model id finetuning. At the early stages there is larger changes in gradient, but at the end model reaches minimum for loss function. The same scenario can be seen in layer 1 wherein model's loss function is less.

2.2 Observe Gradient Norm During Training: $\text{sgn}\left(\frac{\cos(10x)}{x}\right)$

A plot was generated showing the gradient norm at each iteration and its relationship to the loss function over time. The gradient norm decreases significantly in the initial iterations, indicating rapid learning, before stabilizing as the model converges.



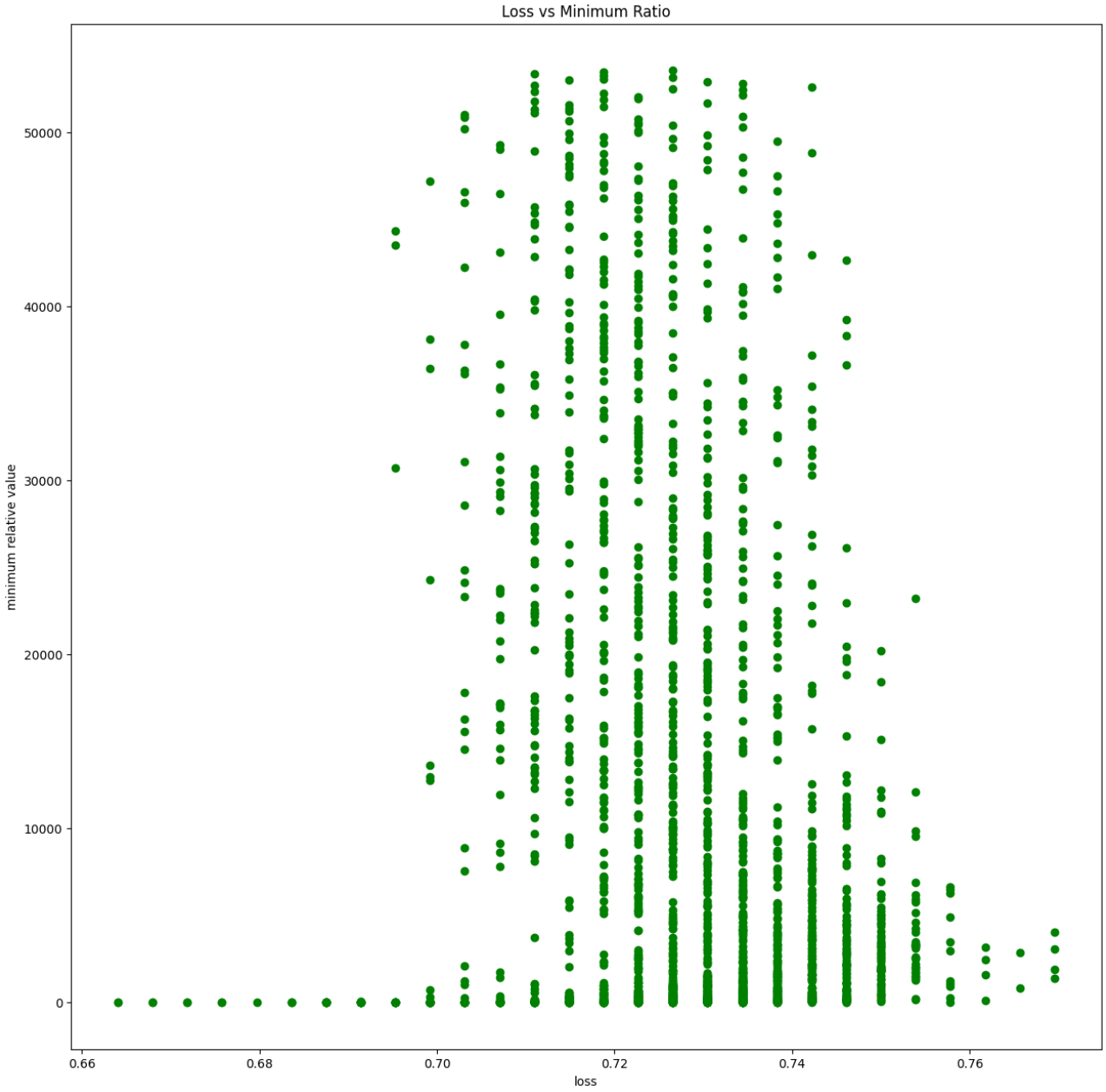
Observations: The plot shows that the gradient norm decreases quickly early in training as the model adjusts weights to reduce loss. As the loss settles, the gradient norm approaches to zero, indicating that the model is no longer making large parameter updates.

\

2.3 When gradient is zero:

To examine what happens when the gradient norm approaches zero, I identified the weights where the gradient norm becomes negligible. These were used to compute the minimal ratio

The loss stabilizes and the model achieves a point of minimal updates when the gradient norm approaches zero. The minimal ratio helps in understanding the nature of the local minimum.



3 Generalization

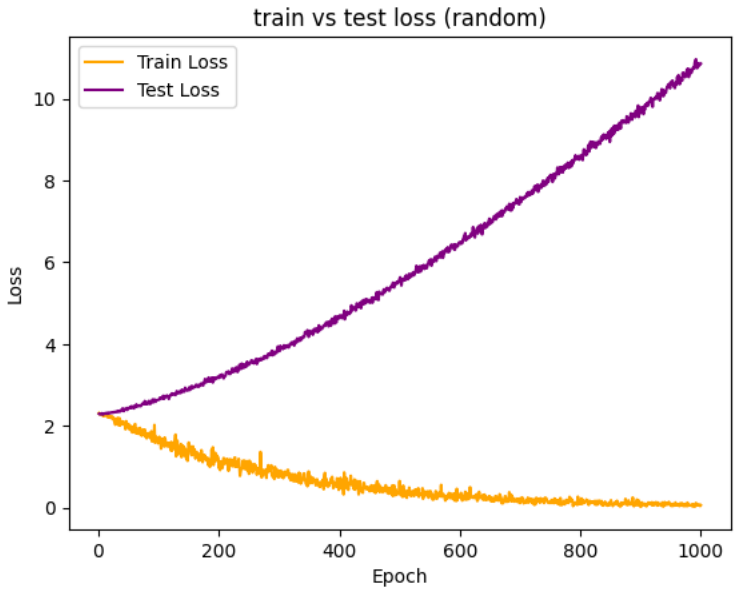
\

3.1 Can network fit random labels?

Model Architecture:

A single dense layer
Total number of parameters: 397519
Activation Function: ReLU
Loss Function: Cross Entropy Loss
Optimizer: Adam
Learning Rate: 0.0001
Random Labels: For this experiment, the labels in the MNIST training set were randomly shuffled.

The model trained with random labels was evaluated for both training and test losses over 1000 epochs.



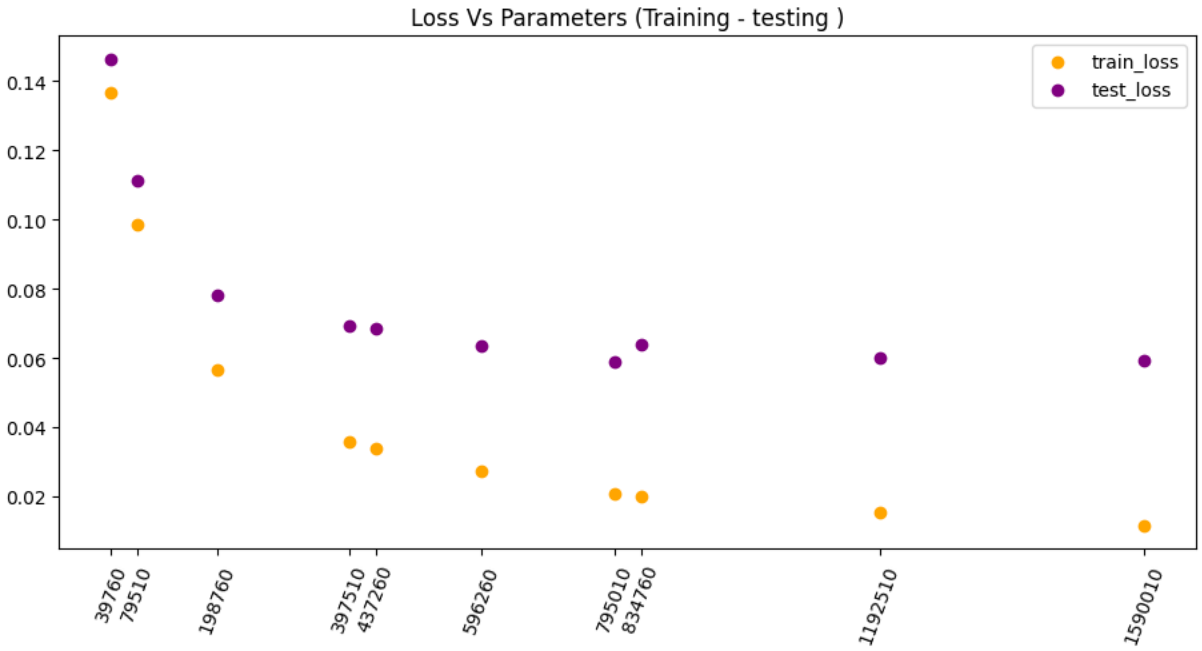
The training loss decreases over time, which suggests that the model is successfully learning to fit the random labels. The test loss increases continuously, this indicates that the model is not generalizing to the test data, this is due to random labels. This opposite behavior between training loss and testing loss is leading to overfitting. Concluding that meaningful data is crucial for training models,

\

3.1 Number of parameters v.s. Generalization

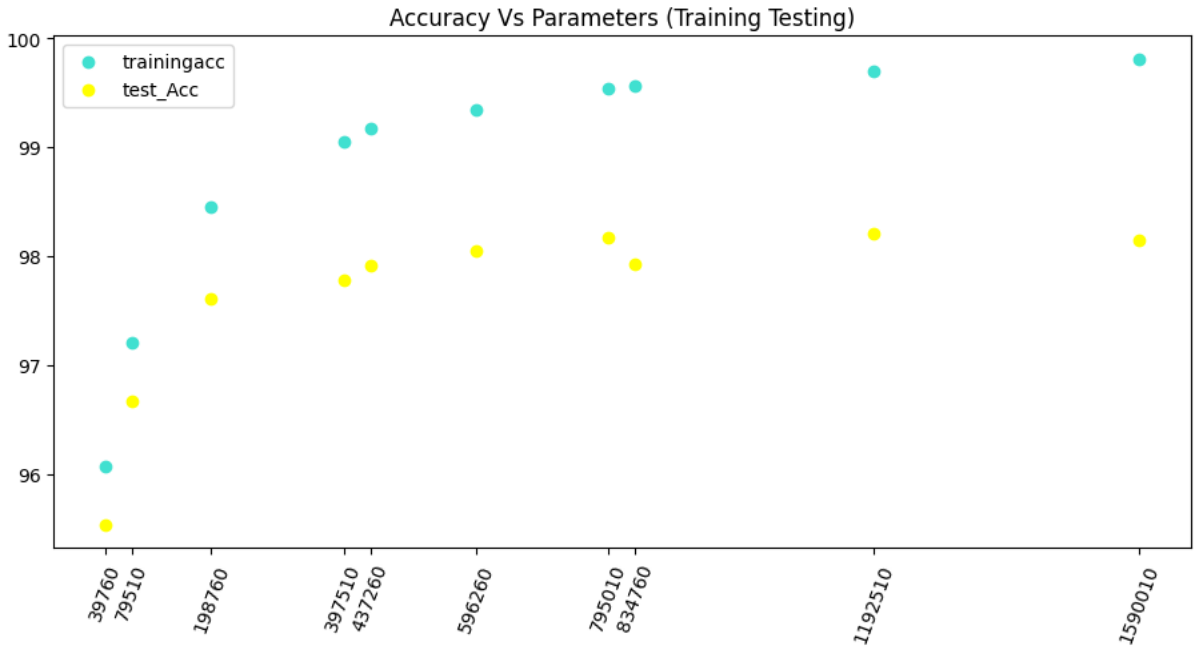
Model Architecture:

Two convolution layers with kernel size 4, max pooling, two dense layers.
Number of parameters was varied across 10 different models ranging from 39,760 to over 397,510 parameters.
Loss Function: Cross Entropy Loss
Optimizer: Adam



Training Loss: As the number of parameters increases, the training loss generally decreases, indicating that larger models (with more parameters) are better at fitting the training data.
Test Loss: The test loss decreases initially but starts fluctuating for larger models. This suggests that after a certain number of parameters, the model begins to overfit the training data.

\



Training Accuracy: As the number of parameters increases, the training accuracy improves, reaching almost 100% for models with more parameters. This suggests that larger models have more capacity to memorize the training data. Test accuracy improves with the number of parameters initially but starts to plateau and fluctuate for larger models.

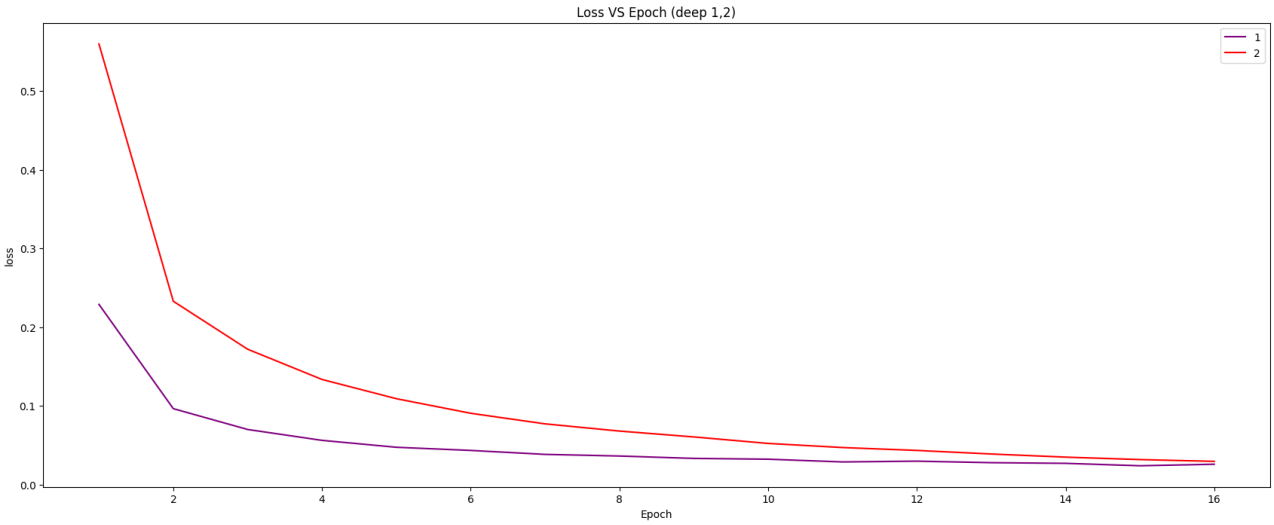
Flatness vs generalization

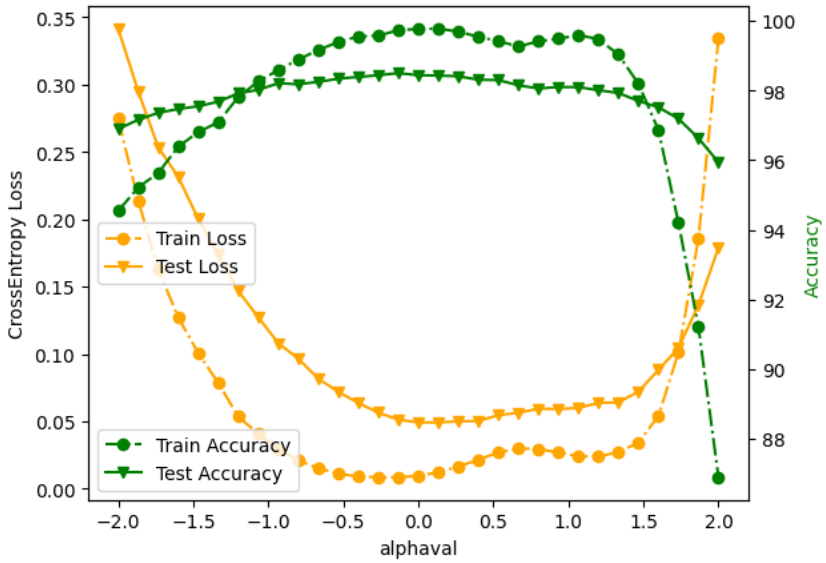
Part 1 :

Create models with different batch sizes

Model Architecture:
One dense layer
Activation Function: ReLU
Loss Function: Cross Entropy Loss
Optimizer: Adam
Batch Size: Two models were trained, one with a batch size of 64, and another with a batch size of 1024.

\

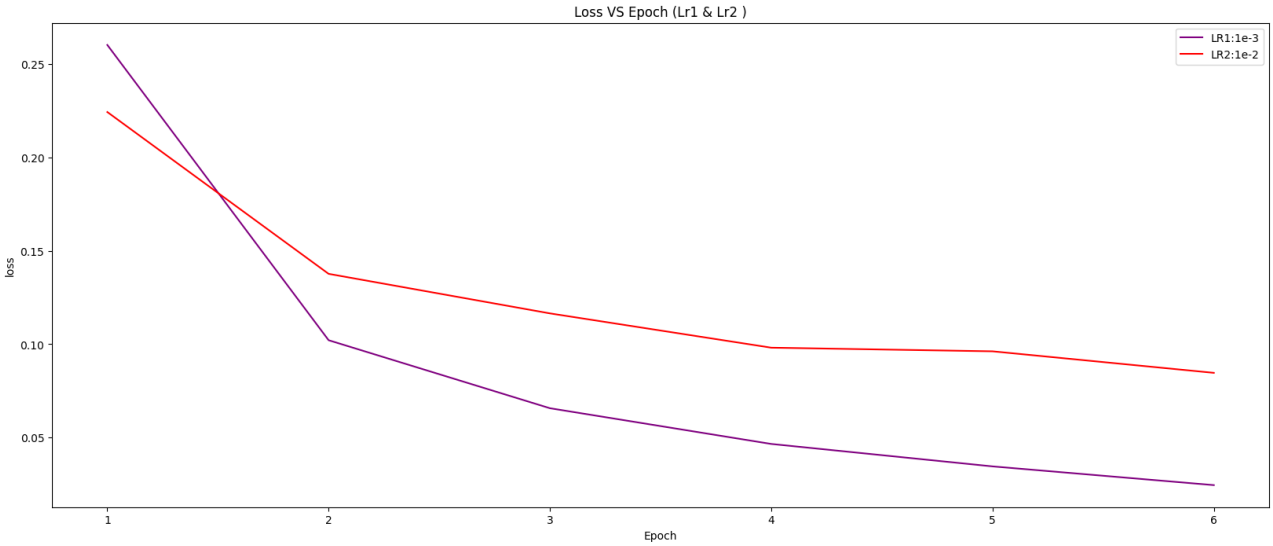


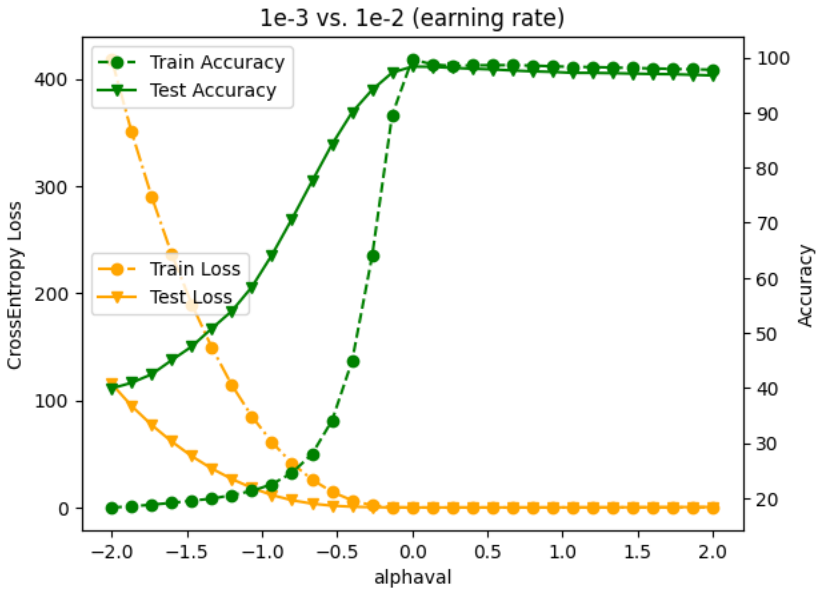


Based on the experiment, the loss and accuracy vary across the interpolation ratios. For certain values of \alpha, the interpolated model generalizes better (lower test loss, higher test accuracy) than either of the individual models.

\

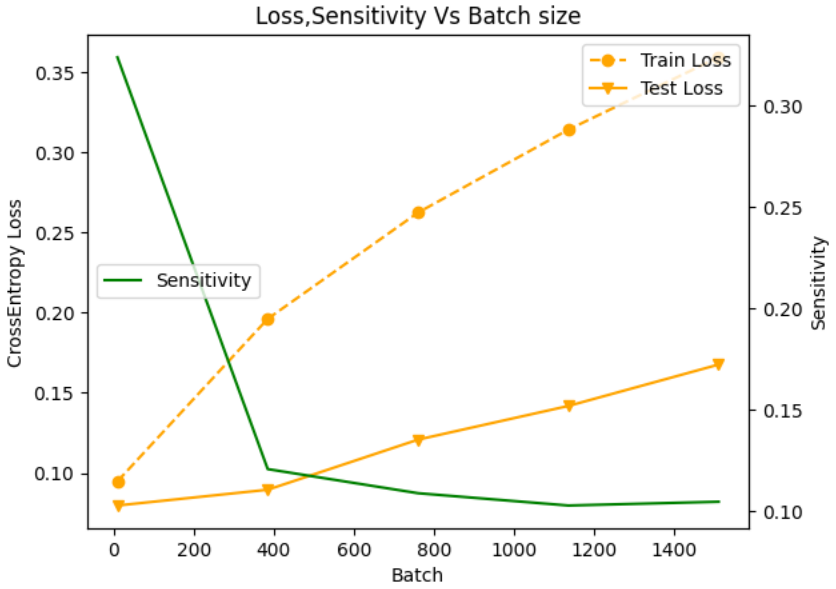
Part 2 –
Sensitivity to batch sizes





The results show that models trained on smaller batches typically achieve higher accuracy and reduced test and training losses. Greater losses and decreased accuracy on the training and test sets, on the other hand, indicate that larger batch sizes are less responsive to changes in the data.

\



Smaller batch training results in lower test and train losses and higher sensitivity for the models. This implies that smaller batch sizes improve the model's ability to generalize, probably because they let the model to explore flatter areas of the loss landscape. Greater train and test losses, together with decreased sensitivity, are associated with larger batch sizes. This suggests that larger batches tend to overfit to sharper minima, which results in worse generalization, even though they may accelerate convergence.

\

\