PRML ASSIGNMENT - 6

Name : Aditya Sahani
Roll No : B22CS003
Google Colab :
https://colab.research.google.com/drive/1LCpG2SMC7jqB_4KmrGBkD_sB4ffpfG6a?usp=sharing

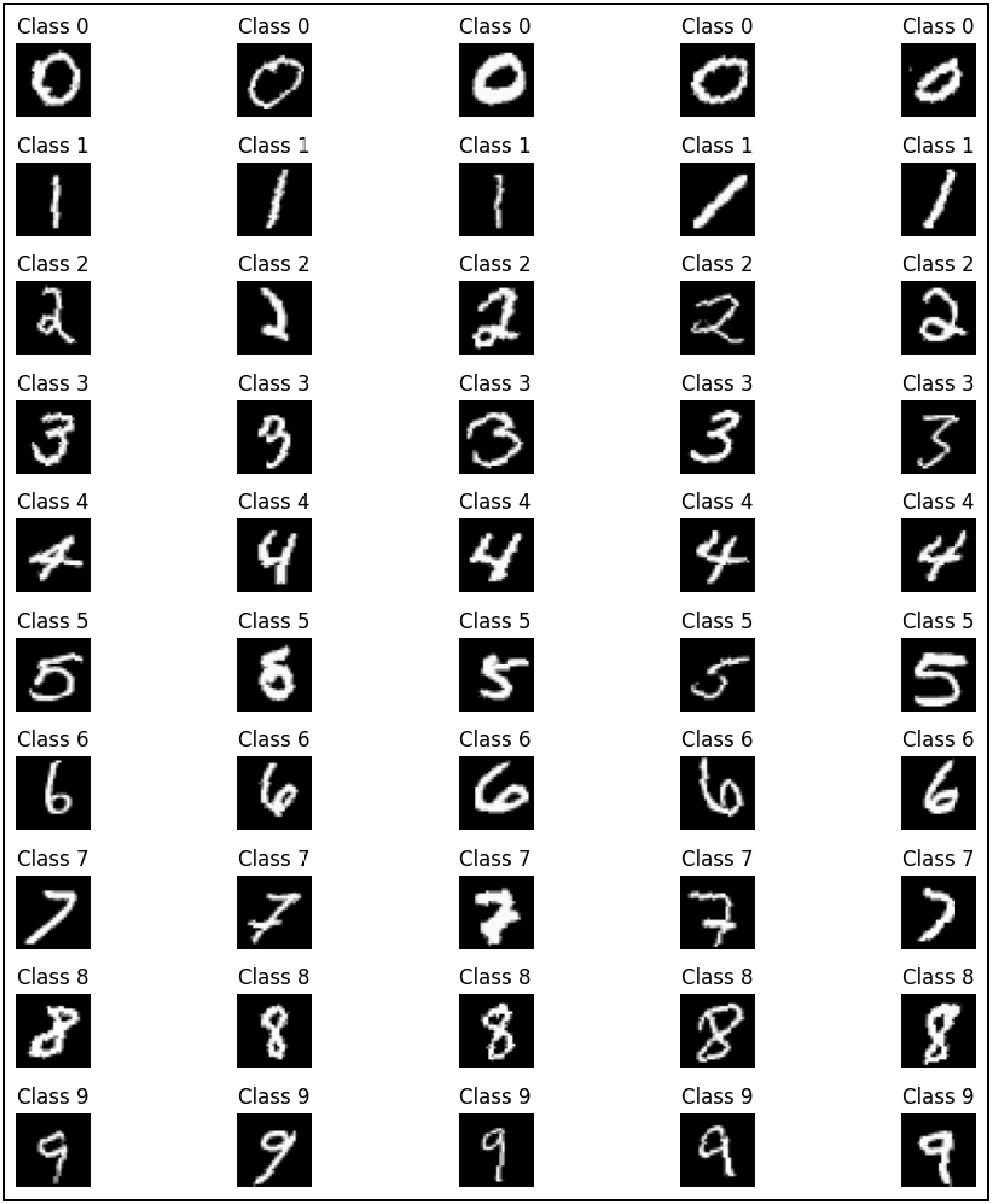
Neural Network

Task 0: Initialization and Data Loading

- **Libraries:** Essential libraries imported, including torch, torchvision, torch.nn, torch.optim, and cv2.
- **Device Configuration:** Checks if CUDA is available, setting the appropriate device.
- **Model and Training Parameters:** Sets input size, output classes, learning rate, batch size, and epochs.

Task 1: Data Preparation and Visualization

- **Transformation:** Applies data augmentation with random rotations and crops, and normalizes with the MNIST-specific mean and standard deviation.
- **Data Loading:** Loads the MNIST training and test datasets, applying the transformations.
- **Dataset Splitting:** Splits the training dataset into training and validation subsets.
- **Visualization:** Plots images from each class to visualize the dataset.



Task 2: Model Definition

- **Model Structure:** Defines a 3-layer MLP with two hidden layers and an output layer, using linear layers and ReLU activation.

- **Model Parameters:** Calculates and prints the total number of trainable parameters in the model.
- **Loss Function and Optimizer:** Sets CrossEntropyLoss as the loss function and Adam as the optimizer with a specified learning rate.

$$H(y) = -\sum_{i}^{n} y_i log(y_i)$$

Number of trainable parameters : 52650

Calculation :

First Layer:

Weights = 64 x 784 = 50176

Biases = 64

Second Layer:

Weights = 64 x 32 = 2048

Biases = 32

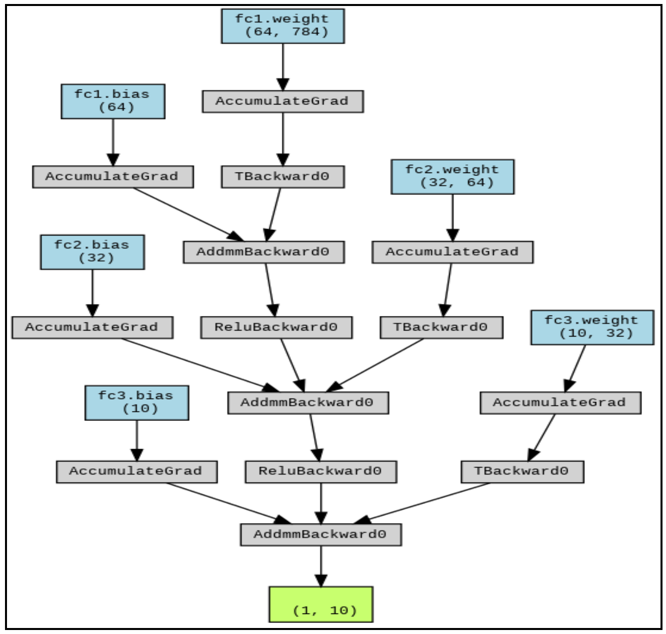
Third Layer:

Weights = 32 x 10 = 320

Biases = 10

Total trainable parameters = 50176 + 64 + 2048 + 32 + 320 + 10 = 52650

Plot of the model tree of Neural Network :



Task 3: Training and Validation

- **Training Loop:** Implements a training loop for a specified number of epochs, with loss calculation, optimizer steps, and accuracy tracking.
- **Metrics:** Tracks training and validation accuracy and loss, logging results after each epoch.
- **Model Checkpointing:** Saves the best model based on validation accuracy.

Training Loss vs Validation Loss :

Epoch	Training Loss	Validation Loss
1	0.3699	0.2553
2	0.2582	0.2492
3	0.2328	0.2382
4	0.2167	0.2246

5	0.2118	0.2245

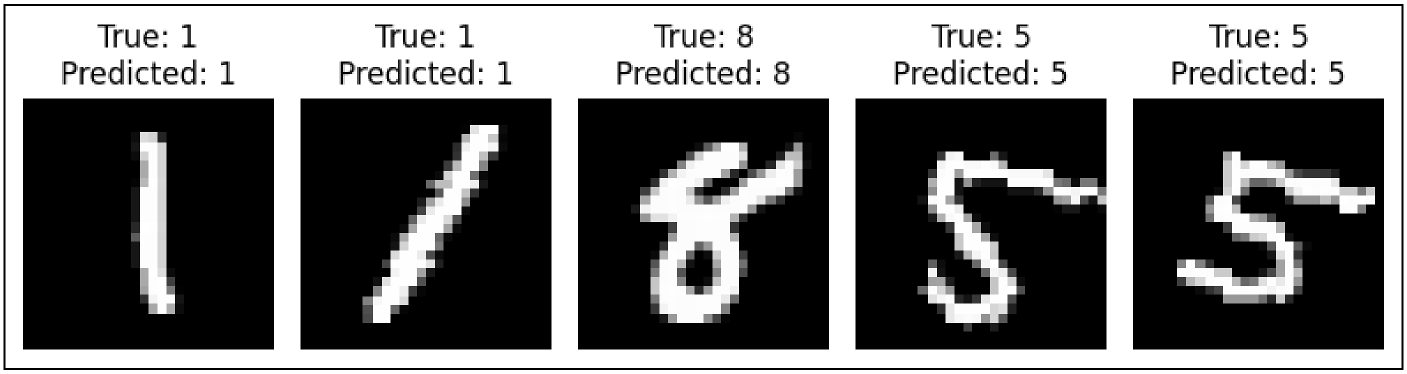
Training Accuracy vs Validation Accuracy:

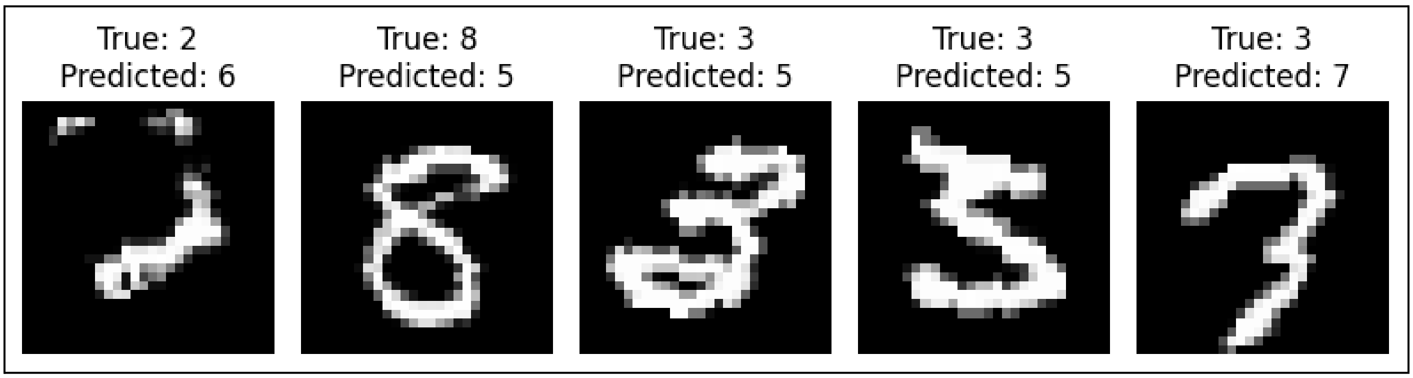
Epoch	Training Accuracy	Validation Accuracy
1	0.8862	0.9247
2	0.9246	0.9286
3	0.9338	0.9358
4	0.9401	0.9396
5	0.9414	0.9361

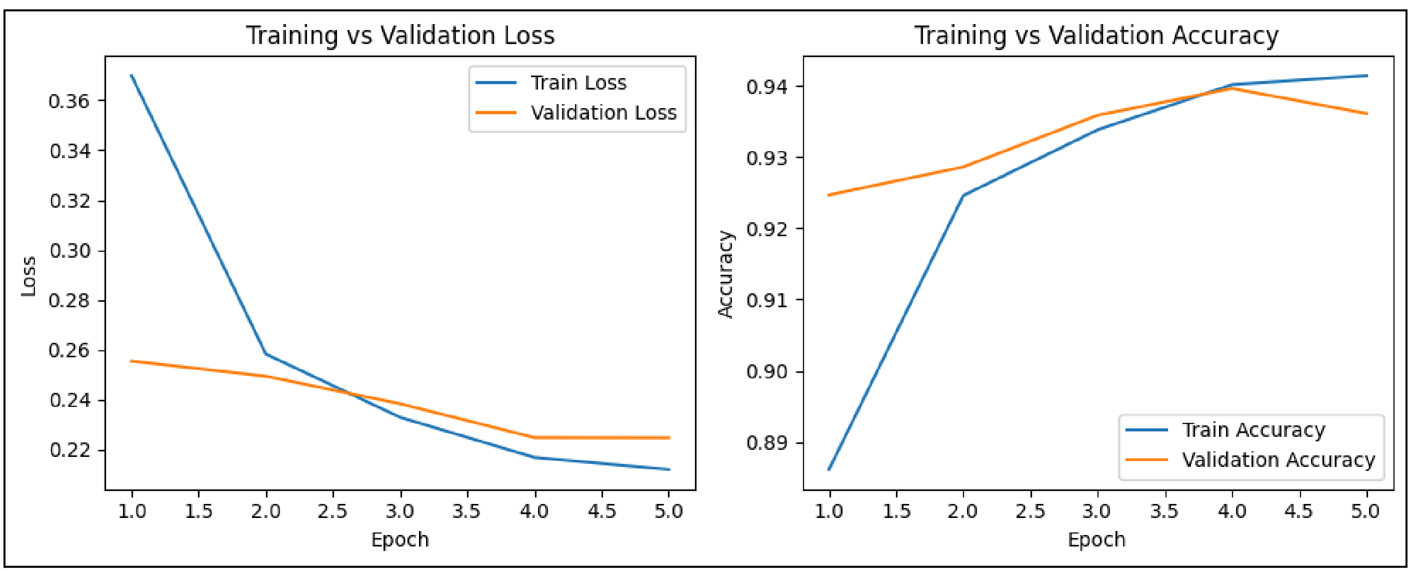
Accuracy on test data : 95.70 %

Task 4: Visualization and Analysis

- **Training and Validation Results:** Plots training and validation loss/accuracy after training.
- **Correct and Incorrect Predictions:** Visualizes examples of correct and incorrect predictions.
- **Model Accuracy:** Calculates model accuracy on the test set using check_accuracy.
- **PCA Scatter Plot:** Applies PCA to reduce dimensions and plots a scatter plot of MNIST classes with colors for each class.







Observations :

Final Results: Displays a scatter plot of MNIST classes after applying PCA.

Why low accuracy ?

Simple Model Structure:
A 3-layer MLP might not capture all complex patterns in the MNIST dataset. Lack of depth or neuron count could limit the model's learning capacity.

Insufficient Regularization:
Overfitting is a common problem when training with smaller datasets. If the model has memorized the training data but doesn't generalize well, validation and test accuracy can suffer.

Limited Data Augmentation:
Data augmentation introduces variability in training samples, enhancing the model's ability to generalize. Minimal augmentation might lead to overfitting.

Suboptimal Hyperparameters:
Incorrect learning rates, batch sizes, or other training parameters could affect model convergence and performance.

Inadequate Training Time:
Insufficient epochs or premature convergence can lead to incomplete training. The number of epochs is 5. We can increase it .

How to improve accuracy ?

Enhance Model Complexity:
Consider adding more layers to the MLP or increasing the number of neurons per layer.Alternatively, use more complex models like Convolutional Neural Networks (CNNs), which are well-suited for image data like MNIST.

Regularization Techniques:

Introduce dropout in hidden layers to reduce overfitting.Apply L2 regularization (weight decay) to the optimizer to penalize large weights.

Hyperparameter Tuning:
Experiment with different learning rates, batch sizes, and optimization algorithms.Use learning rate schedulers to adapt the learning rate during training.

Increase Training Time:
Train for more epochs to allow the model to fully converge.Implement early stopping to avoid overfitting.