PRML PA 6 REPORT

ANUSHKA DADHICH

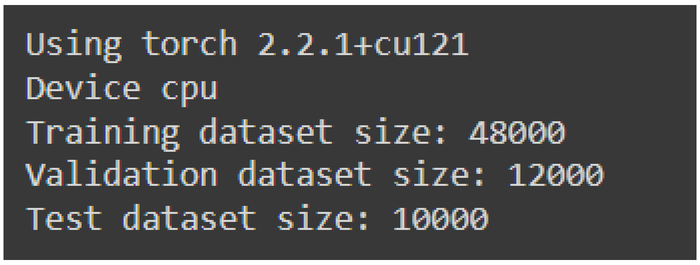
B22CS097

Colab link -
https://colab.research.google.com/drive/1SQdDZwIKv3kZOYqybeqcEE5tyujVEwsD?usp=sharing

TASK 0:

- The code uses torchvision from pytorch to download the MNIST dataset.
- Then splitting the dataset into training, validation, and testing sets.
- Augmentations such as RandomRotation, RandomCrop, ToTensor, and Normalize are applied to the images during preprocessing. These transformations help in augmenting the dataset and preparing it for training a neural network.

Output:

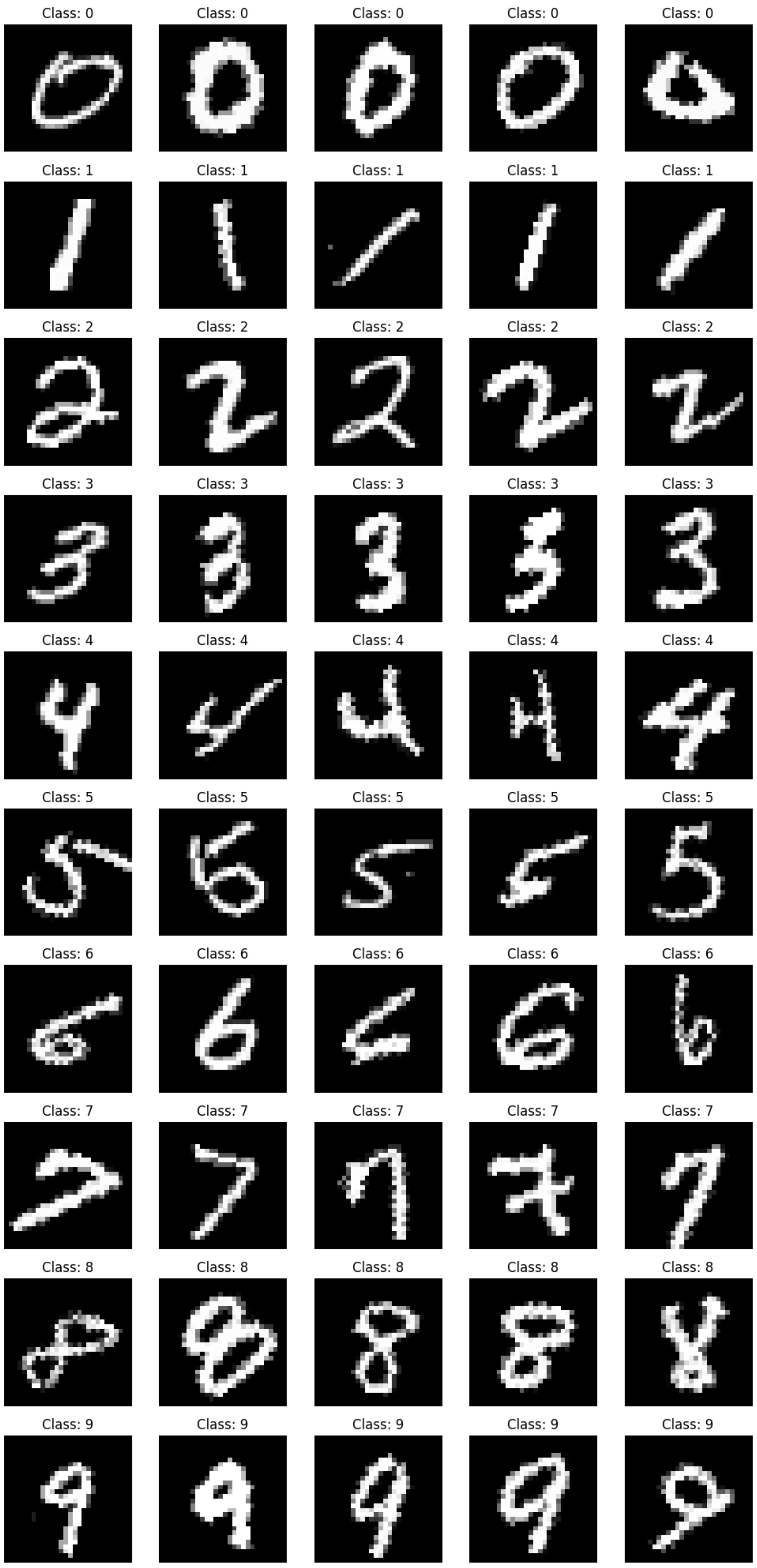


```
Using torch 2.2.1+cu121
Device cpu
Training dataset size: 48000
Validation dataset size: 12000
Test dataset size: 10000
```

TASK 1:

- Defined a function plot_images to visualize a few images from each class in a dataset. It utilizes Matplotlib to create subplots for each class and then iterates through the classes to select a few images for display.
- Finally, it plots the images using imshow and displays the class label as the title of each subplot. After defining the plotting function, it is called to visualize a few images from each class in the training dataset.
- Use of torch.utils.data.DataLoader to create data loaders for efficient batching and loading of data during training and testing.

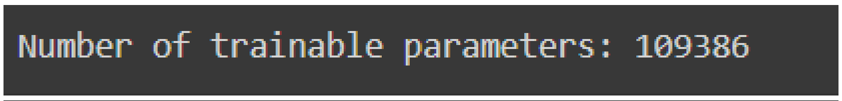
Output:


Class: 0
Class: 0
Class: 0
Class: 0
Class: 0
Class: 1
Class: 1
Class: 1
Class: 1
Class: 1
Class: 2
Class: 2
Class: 2
Class: 2
Class: 2
Class: 3
Class: 3
Class: 3
Class: 3
Class: 3
Class: 4
Class: 4
Class: 4
Class: 4
Class: 4
Class: 5
Class: 5
Class: 5
Class: 5
Class: 5
Class: 6
Class: 6
Class: 6
Class: 6
Class: 6
Class: 7
Class: 7
Class: 7
Class: 7
Class: 7
Class: 8
Class: 8
Class: 8
Class: 8
Class: 8
Class: 9
Class: 9
Class: 9
Class: 9
Class: 9

TASK 2:

- A 3-layer Multilayer Perceptron (MLP) is implemented using PyTorch's nn.Module class.
- The MLP consists of three linear layers (fully connected layers) with ReLU activation functions between them. The model architecture is defined in the **MLP** class, where the constructor initializes the linear layers (**f1**, **f2**, and **f3**) with the specified input, hidden, and output sizes.
- The **forward** method defines the forward pass of the model, where the input tensor is flattened and passed through each linear layer with ReLU activation. Finally, the output tensor is returned.
- After defining the model, the total number of trainable parameters is calculated by summing the number of elements in all trainable parameters of the model.
- This count includes the weights and biases of each linear layer. The number of trainable parameters provides insight into the model's complexity and capacity to learn from the data.

Output:

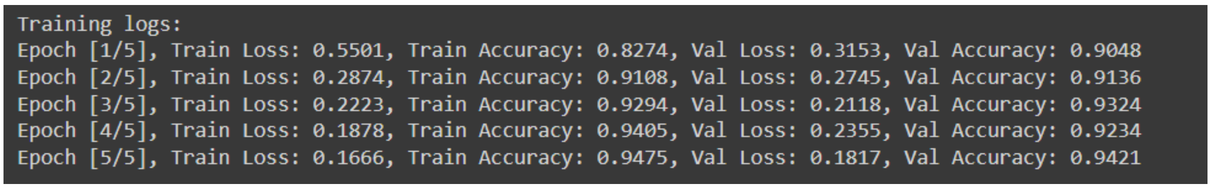


```
Number of trainable parameters: 109386
```

TASK 3:

- The code performs the training of the previously defined MLP model (three_layer_model).
- First, it defines the loss function (CrossEntropyLoss) and the optimizer (Adam) to optimize the model parameters during training as given in the question.
- Next, it defines a function to calculate the accuracy of the model's predictions. This function compares the predicted labels with the ground truth labels and calculates the accuracy as the fraction of correct predictions.
- Another function, evaluate, is defined to evaluate the model's performance on a given dataset. This function computes both the loss and accuracy of the model on the dataset.
- Then it enters a training loop for a specified number of epochs. Within each epoch, the model is set to training mode, and the training data is iterated over in batches. For each batch, the optimizer is zeroed, the forward pass is computed, the loss is calculated, gradients are backpropagated, and the optimizer is updated.
- After each epoch, the model is evaluated on the validation dataset using the evaluate function to compute the validation loss and accuracy. The best model based on validation accuracy is saved.
- Finally, the training logs, including training loss, training accuracy, validation loss, and validation accuracy, are printed after each epoch. The best model is saved to a file named 'best_model.pth' for later use.

Output:



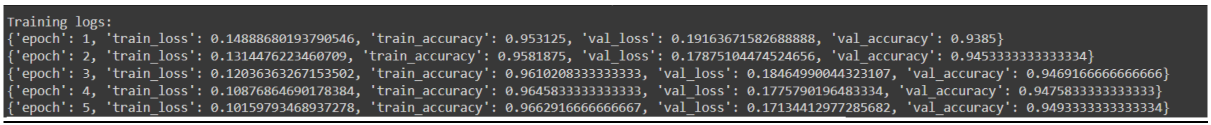
```
Training logs:
Epoch [1/5], Train Loss: 0.5501, Train Accuracy: 0.8274, Val Loss: 0.3153, Val Accuracy: 0.9048
Epoch [2/5], Train Loss: 0.2874, Train Accuracy: 0.9108, Val Loss: 0.2745, Val Accuracy: 0.9136
Epoch [3/5], Train Loss: 0.2223, Train Accuracy: 0.9294, Val Loss: 0.2118, Val Accuracy: 0.9324
Epoch [4/5], Train Loss: 0.1878, Train Accuracy: 0.9405, Val Loss: 0.2355, Val Accuracy: 0.9234
Epoch [5/5], Train Loss: 0.1666, Train Accuracy: 0.9475, Val Loss: 0.1817, Val Accuracy: 0.9421
```

TASK 4:

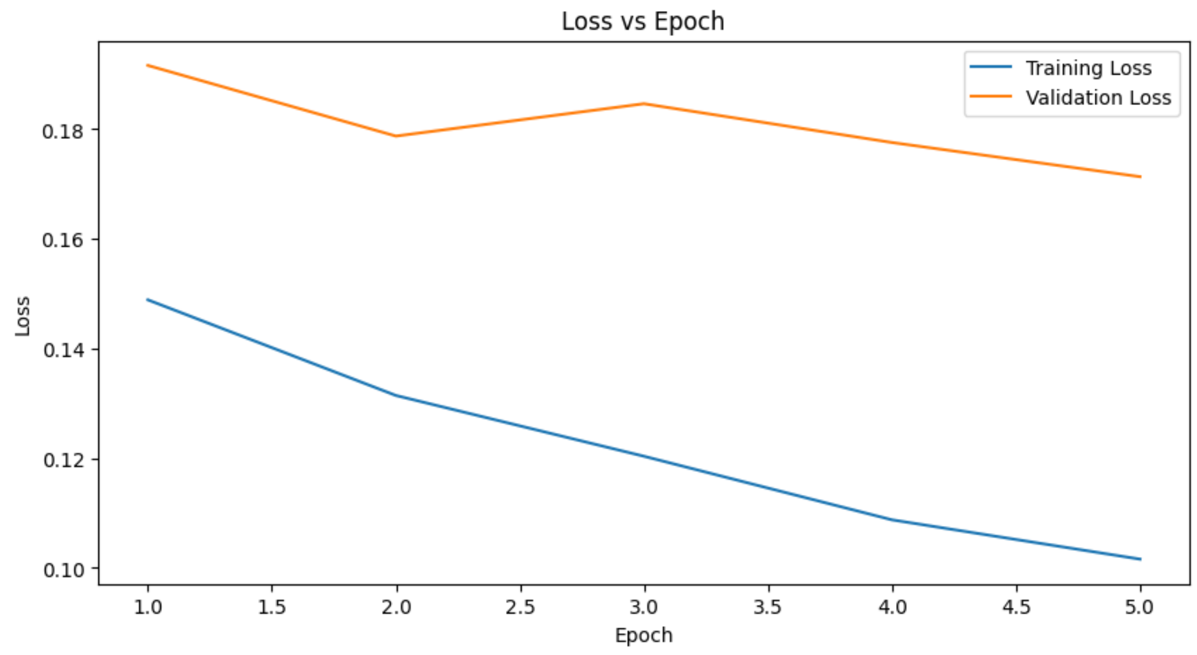
The code performs several operations to evaluate and visualize the performance of the trained model:

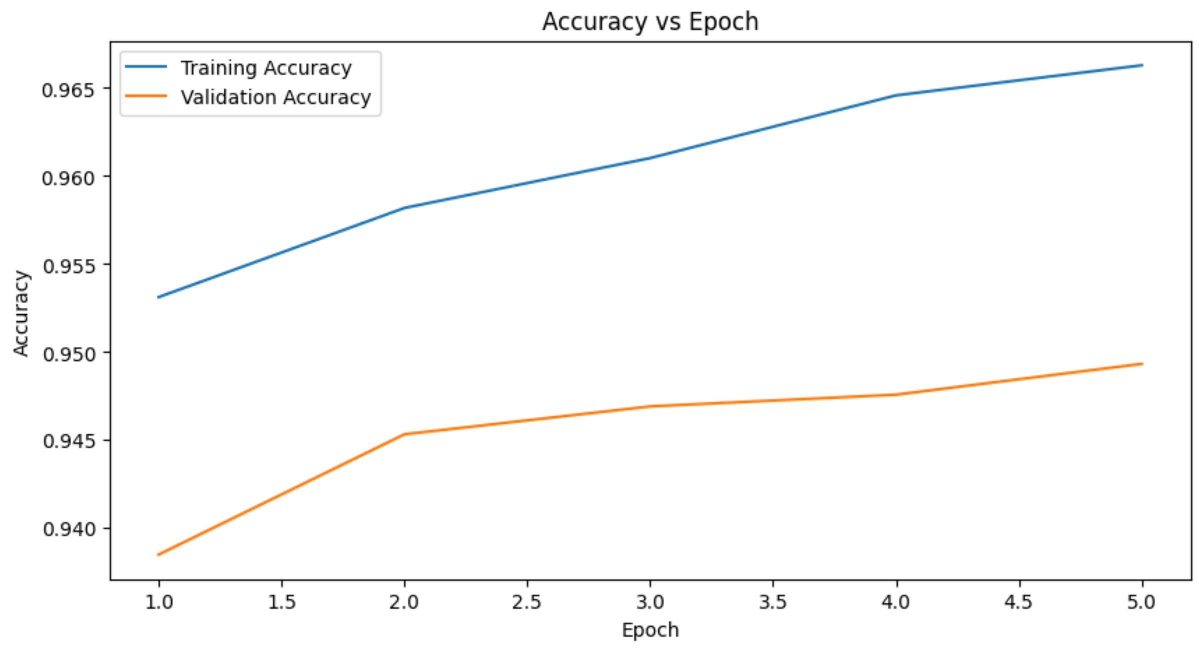
1. Evaluation and Logging:
    - The trained model is evaluated on the validation dataset after each epoch to compute the validation loss and accuracy.
    - The validation loss, validation accuracy, training loss, and training accuracy are logged for each epoch and stored in a list called training_logs.
    - The best model based on validation accuracy is saved to a file named 'best_model.pth'.
2. Plotting Loss and Accuracy:
    - The training and validation loss are plotted against the number of epochs to visualize their trends over training.
    - Similarly, the training and validation accuracy are plotted against the number of epochs to observe how the accuracy changes over training.
3. Visualizing Predictions:
    - The function visualize_predictions is defined to visualize correct and incorrect predictions made by the model.
    - For correct predictions, a subset of correctly classified images from the validation dataset is displayed.
    - For incorrect predictions, a subset of incorrectly classified images from the validation dataset is displayed.

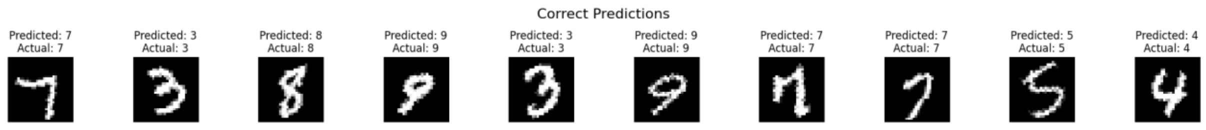
Output:

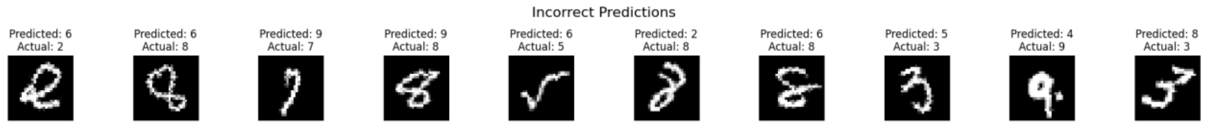


```
Training logs:
{'epoch': 1, 'train_loss': 0.14888680193790546, 'train_accuracy': 0.953125, 'val_loss': 0.19163671582688888, 'val_accuracy': 0.9385}
{'epoch': 2, 'train_loss': 0.1314476223460709, 'train_accuracy': 0.9581875, 'val_loss': 0.17875104474524656, 'val_accuracy': 0.9453333333333334}
{'epoch': 3, 'train_loss': 0.12036363267153502, 'train_accuracy': 0.9610208333333333, 'val_loss': 0.18464990044323107, 'val_accuracy': 0.9469166666666666}
{'epoch': 4, 'train_loss': 0.10876864690178384, 'train_accuracy': 0.9645833333333333, 'val_loss': 0.1775790196483334, 'val_accuracy': 0.9475833333333333}
{'epoch': 5, 'train_loss': 0.10159793468937278, 'train_accuracy': 0.9662916666666667, 'val_loss': 0.17134412977285682, 'val_accuracy': 0.9493333333333334}
```


Loss vs Epoch
Training Loss
Validation Loss
Loss
Epoch
0.18
0.16
0.14
0.12
0.10
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Accuracy vs Epoch
Training Accuracy
Validation Accuracy
Accuracy
Epoch
0.965
0.960
0.955
0.950
0.945
0.940
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Correct Predictions
Predicted: 7 Actual: 7
Predicted: 3 Actual: 3
Predicted: 8 Actual: 8
Predicted: 9 Actual: 9
Predicted: 3 Actual: 3
Predicted: 9 Actual: 9
Predicted: 7 Actual: 7
Predicted: 7 Actual: 7
Predicted: 5 Actual: 5
Predicted: 4 Actual: 4


Incorrect Predictions
Predicted: 6 Actual: 2
Predicted: 6 Actual: 8
Predicted: 9 Actual: 7
Predicted: 9 Actual: 8
Predicted: 6 Actual: 5
Predicted: 2 Actual: 8
Predicted: 6 Actual: 8
Predicted: 5 Actual: 3
Predicted: 4 Actual: 9
Predicted: 8 Actual: 3