PATTERN RECOGNITION AND MACHINE LEARNING
LAB REPORT
ASSIGNMENT-4

Lavangi Parihar
B22EE044

Question 1 NEURAL NETWORKS

Task 0– Getting the dataset

- The MNIST dataset consists of handwritten digits and is commonly used in image processing tasks..
- a series of image transformations, applied to preprocess the images before feeding them into a neural network. The transformation steps are as follows:

  1. Random Rotation: Images are rotated by a random angle within ±10 degrees. This introduces rotational variance to the dataset, helping the model generalize better to new, unseen data.
  2. Random Crop: Images are cropped to 28x28 pixels with padding of 4 pixels on each side before cropping. This transformation is used to make the model invariant to small translations of the input image.
  3. ToTensor: Converts the images to PyTorch tensors, which are a suitable input format for training models in PyTorch.
  4. Normalize: The pixel values are normalized using a mean of 0.5 and a standard deviation of 0.5. Normalization helps in stabilizing the learning process and improving the convergence speed of gradient descent.

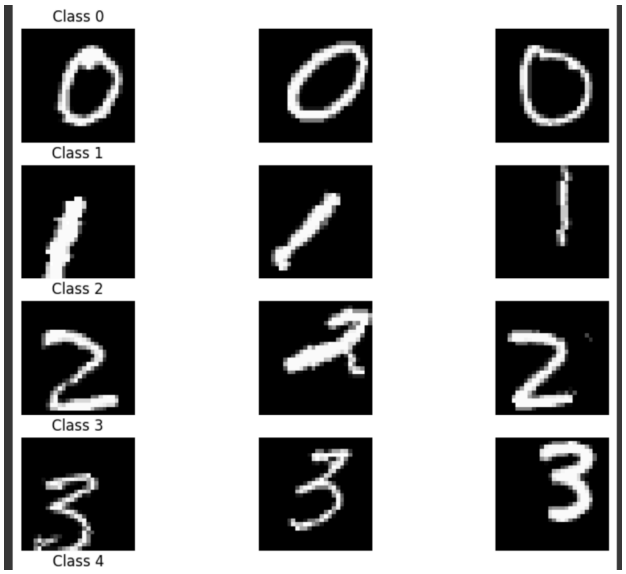
- The MNIST dataset is split into training and testing datasets using the torchvision dataset utility. Additionally, the training dataset is further split into training and validation datasets:

  - Training Dataset: Contains 90% of the MNIST training data..
  - Validation Dataset: Contains the remaining 10% of the MNIST training data.
  - Testing Dataset: Contains the remaining 10% of the MNIST training data.

Task 1– Plotting the dataset

- The script initializes data loaders for the training, validation, and test datasets with the following parameters:
  - Shuffle: Enabled for all data loaders to ensure the data is shuffled at every epoch, which helps prevent the model from learning unintended patterns from the order of the data.

- The script first collects a set of images from the training dataset loader using the `get_images_for_each_class` function. It then visualizes these images using the `plot_class_images` function.



Task 2- Writing 3 layer MLP

- Function: `create_three_layer_mlp`
  The purpose of this function is to define a neural network using PyTorch's `nn.Sequential` container, which chains together a series of layers and activations. Here is a breakdown of the network architecture:

  1. First Layer: An input layer that flattens the 28x28 pixel images into a single 784-element vector and passes it through a fully connected (linear) layer with 128 output nodes.
  2. First Activation: A ReLU (Rectified Linear Unit) activation function that introduces non-linearity, helping the network learn more complex patterns.
  3. Second Layer: Another linear layer that reduces the dimension from 128 nodes to 64 nodes.
  4. Second Activation: A ReLU function to maintain non-linearity in the model.
  5. Output Layer: A final linear layer that maps the 64 nodes to 10 output nodes, corresponding to the 10 possible classes (digits 0-9).

  The function returns the constructed model.

- Model Creation
  The script instantiates the model by calling `create_three_layer_mlp` and prints the structure of the model. This output allows verification of the model architecture, ensuring that the layers are correctly defined as per the specifications.

- Function: `count_parameters`
  This function calculates the total number of trainable parameters in the model. It iterates through all parameters of the model that require gradients (trainable parameters) and sums up their elements using:
  - p.numel(): This returns the number of elements in the parameter tensor `p`.
  - p.requires_grad: Ensures that only parameters which contribute to model learning are counted.

  This function is crucial for understanding the complexity and capacity of the neural network.

```
Sequential(
  (0): Linear(in_features=784, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=10, bias=True)
)
Number of trainable parameters: 109386
```

Task 3- Training and Validating the model

- The script covers the setup of a loss function and optimizer, defines training and validation procedures, manages device allocation for GPU or CPU, and tracks the performance of the model across epochs.

- Additionally, the script visualizes correct and incorrect predictions, saves the best-performing model, and reloads it to confirm its performance

- Setup of Loss Function and Optimizer
  - Loss Function: The script uses `CrossEntropyLoss` This loss function combines softmax activation and cross-entropy loss in one single class.
  - Optimizer: `Adam` optimizer is chosen for its efficiency in handling sparse gradients and adaptive learning rate capabilities, which makes it suitable for quickly converging on training deep learning models.

- Training and Validation Functions
  - Function: `train_model`: This function handles the training loop for the model. It performs the following steps:
    1. Set the model to training mode.
    2. Iteratively processes batches of data, computing outputs, and loss, performing backpropagation, and updating model parameters.
    3. Calculates total loss and accuracy for the epoch and returns these metrics.

- Function: `validate_model`: Runs the model in evaluation mode to avoid changing model parameters, and computes validation loss and accuracy. It also collects a small number of correctly and incorrectly classified samples for visualization purposes, providing insights into model performance and potential areas for improvement.

- Training Procedure
  The script orchestrates the training over multiple epochs, handling both training and validation within each epoch. Key components include:
  - Epoch-wise Training and Validation: For each epoch, it calls the `train_model` and `validate_model` functions, appending loss and accuracy metrics to respective lists.
  - It tracks and prints the loss and accuracy for both training and validation phases within each epoch, providing a clear view of the model's learning progress.

```
Epoch 1: Train Loss: 1.0494, Train Acc: 64.93%, Val Loss: 0.6281,
Val Acc: 80.37%
Epoch 2: Train Loss: 0.5339, Train Acc: 83.08%, Val Loss: 0.4577,
Val Acc: 85.17%
Epoch 3: Train Loss: 0.4290, Train Acc: 86.69%, Val Loss: 0.4176,
Val Acc: 87.03%
Epoch 4: Train Loss: 0.3746, Train Acc: 88.51%, Val Loss: 0.3594,
Val Acc: 88.50%
Epoch 5: Train Loss: 0.3411, Train Acc: 89.46%, Val Loss: 0.3516,
Val Acc: 88.77%
Training complete.
```

- The best model (based on validation loss) is saved during the training process, ensuring that the best-performing parameters are retained.

```
The best model was found at Epoch 5 with Validation Loss: 0.3516 and
Validation Accuracy: 88.77%.
```

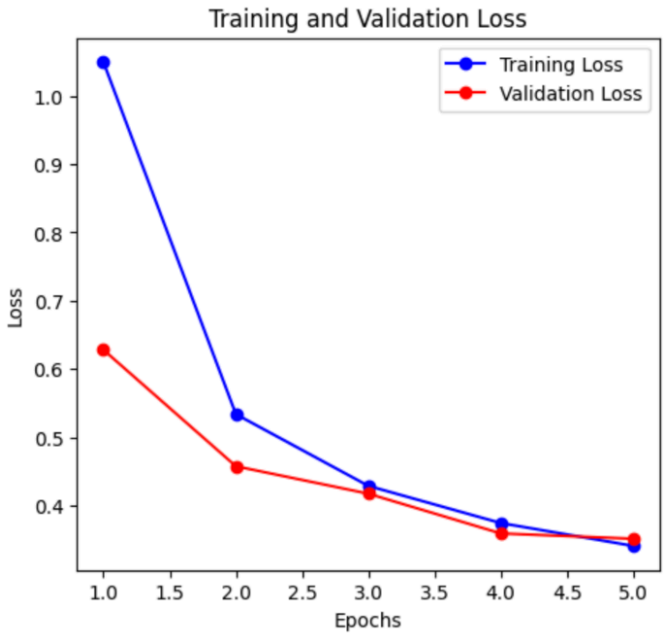
Task 4- Visualizing the results

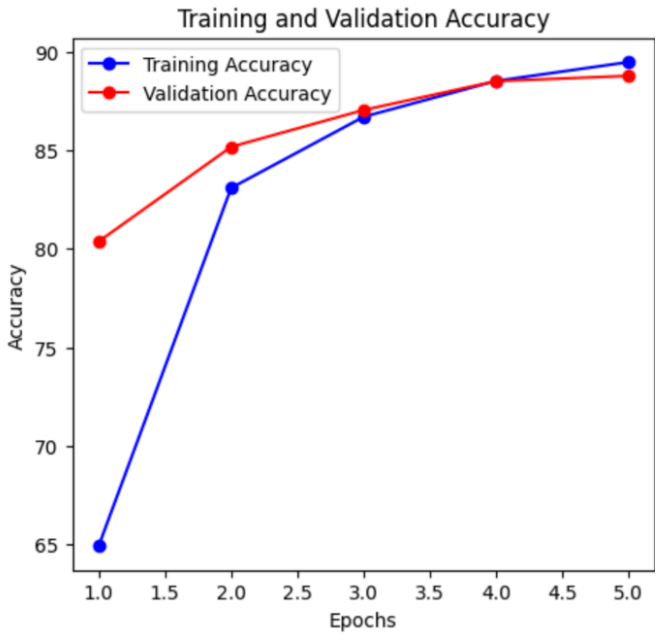
- Visualization of Training and Validation Metrics

The script uses matplotlib, to create visual representations of training dynamics. The key components of this visualization include:

1. Loss Graph:
   - Function: Plots both training and validation loss over epochs.
   - Purpose: Helps in identifying trends such as overfitting or underfitting. For example, diverging training and validation loss might indicate overfitting.
   - Design: Uses blue lines for training loss and red lines for validation loss, with markers on each epoch, enhancing clarity and visual appeal.

2. Accuracy Graph:
   - Function: Displays training and validation accuracy over the same epochs.
   - Purpose: Allows assessment of how well the model is learning and generalizing to unseen data.
   - Design: Follows the same color scheme and style as the loss graph, ensuring consistency in visualization.





- Visualization of Prediction Samples
  In addition to performance metrics, the script visualizes specific prediction samples from the validation set:

  1. Correct Predictions:
     - Function: Displays images that were correctly classified by the model.
     - Purpose: Provides confidence in the model's predictive capabilities and showcases areas of strength.

2. Incorrect Predictions:
   - Function: Shows images that were incorrectly classified.
   - Purpose: Identifies potential weaknesses in the model, highlighting areas for improvement such as feature extraction or class imbalance handling.