Assignment – 2

Name: Ashmita Das

UID: TNU2022053200054

Dept: CSE (AI &ML)

Sem: 6th

Title: Handwritten Digit Classification

Objective:

- To classify handwritten digits (0-9) using a neural network.
- To implement the model using PyTorch.
- To train the model on the MNIST dataset and evaluate its performance.
- To save the trained model for future inference.

Theory:

PyTorch is an open-source deep learning framework developed by Facebook. It provides Tensor operations with GPU acceleration, improving performance as well as pre-built libraries for datasets, neural networks, and optimizers.

The ***MNIST*** dataset is a collection of 60,000 training and 10,000 testing grayscale images of handwritten digits (0-9), each of size 28×28 pixels.

1. Neural Network Architecture:
A neural network consists of three layers:

- **Input Layer:** Takes the pixel values of the image (28×28 = 784 features for MNIST).
- **Hidden Layers:** Performs feature extraction using weights and activation functions.
- **Output Layer:** Produces 10 outputs (one for each digit 0-9) using the softmax function.

2. Activation Function (ReLU):
The Rectified Linear Unit (ReLU) is used as an activation function to introduce non-linearity: ReLU (x) = max (0 , x)

3. Training Parameters: Training parameters define how the neural network learns

- **Learning Rate:** Controls the step size of weight updates to minimize loss.
- **Batch Size:** Number of samples processed before updating the model weights.
- **Epochs:** Number of times the model iterates over the entire dataset.

4. Data Processing: Prepares data for training

- **Transforms.ToTensor():** Converts images into PyTorch tensors (normalized between 0 and 1).
- **DataLoader:** Loads the dataset in mini-batches for efficient training.

5. Training Process: Steps involved in training a neural network

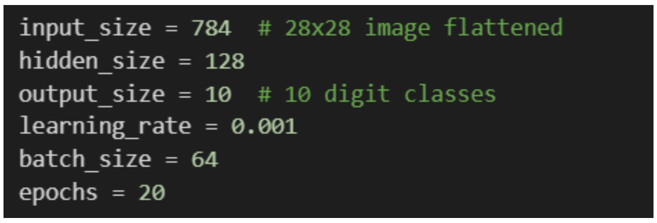
- **Forward Pass:** Input data is passed through the model and predictions are made.
- **Criterion (Loss Function):** Measures how far the predicted outputs are from actual labels.
- **Backpropagation:**
    - optimizer.zero_grad(): Clears previous gradients.
    - loss.backward(): Computes gradients.
- **Optimizer:** Updates the weights to minimize loss (e.g., Adam, SGD).
    - optimizer.step(): Updates weights using computed gradients.
- This process continues for multiple epochs until the model reaches good accuracy.

6. Evaluation and Model Saving:

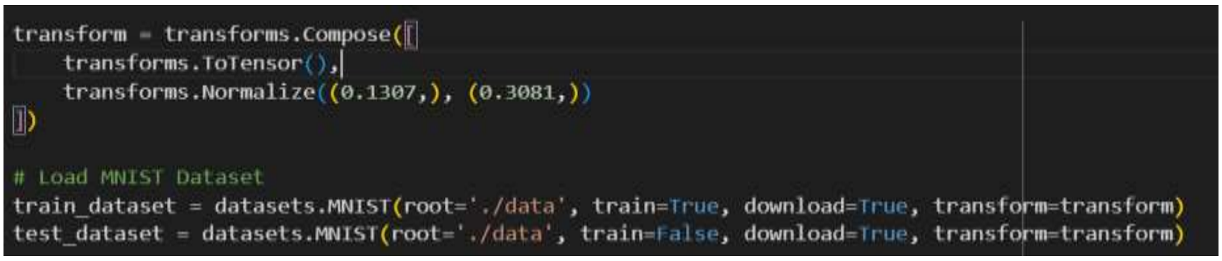
- **Accuracy Calculation:** Measures the percentage of correct predictions on test data.
- **Test Loader:** Used to evaluate the trained model.
- **Model Saving:** Saves the trained model by using *torch.save(model.state_dict(), 'model.pth')*

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
```

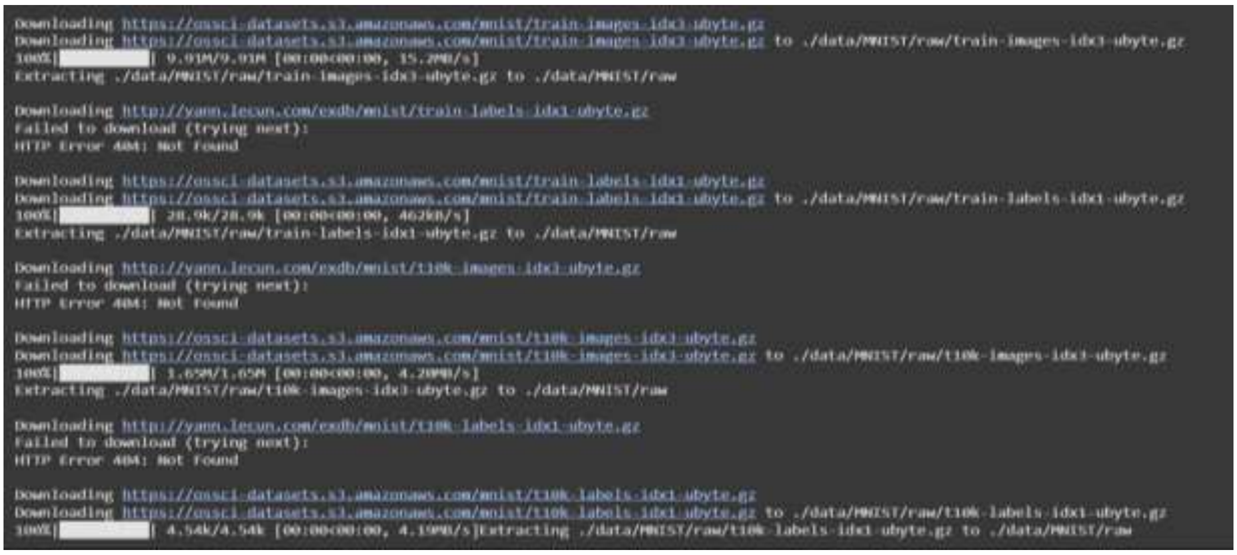
```python
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, output_size)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```



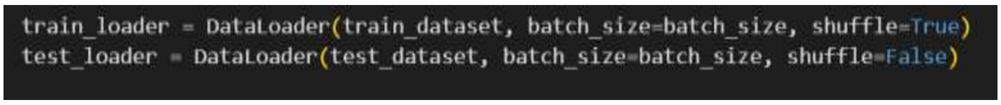
```python
input_size = 784  # 28x28 image flattened
hidden_size = 128
output_size = 10  # 10 digit classes
learning_rate = 0.001
batch_size = 64
epochs = 20
```



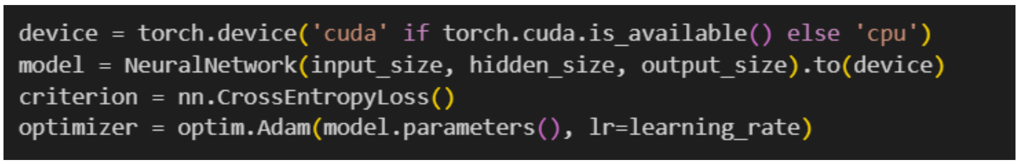
```python
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# Load MNIST Dataset
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)
```



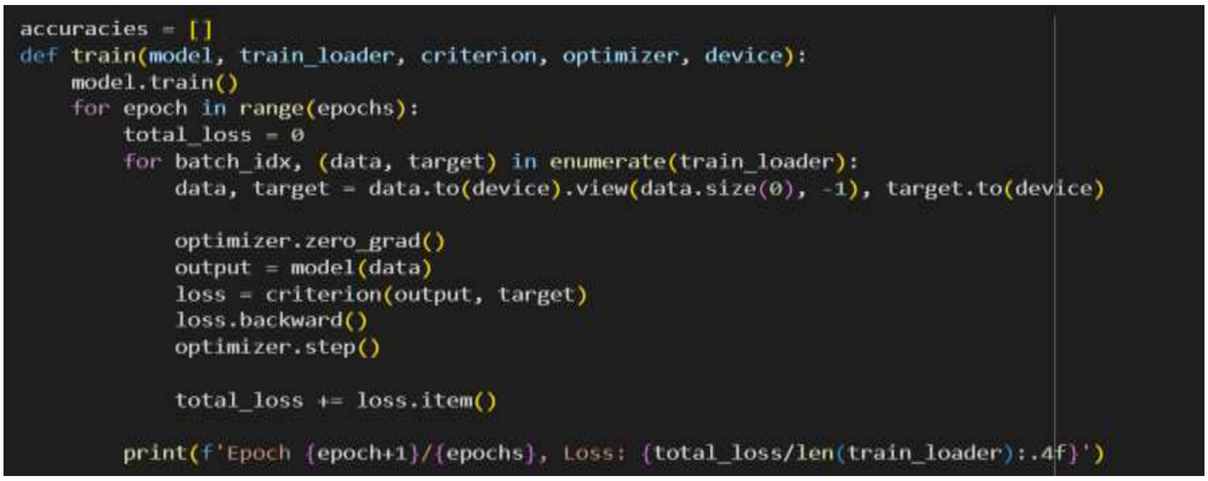
```
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9.91M/9.91M [00:00<00:00, 15.2MB/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 462kB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.20MB/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4.54k/4.54k [00:00<00:00, 4.19MB/s]Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```



```python
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```



```python
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = NeuralNetwork(input_size, hidden_size, output_size).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```



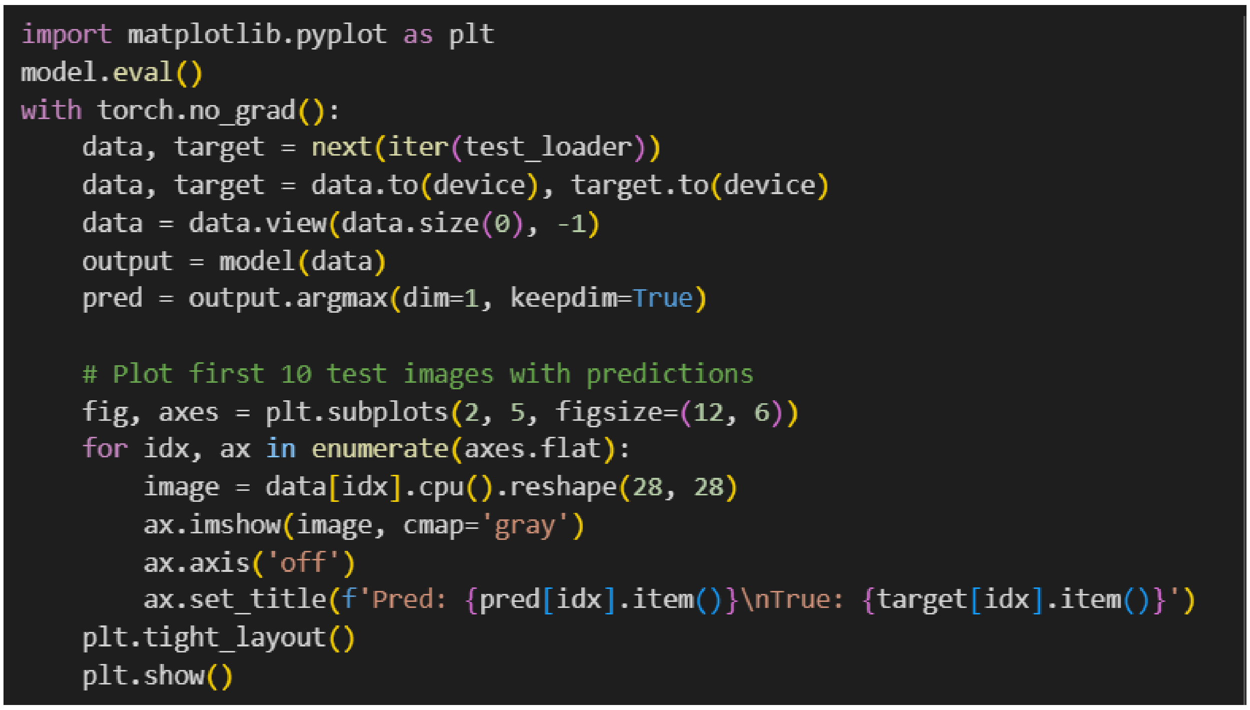
```python
accuracies = []
def train(model, train_loader, criterion, optimizer, device):
    model.train()
    for epoch in range(epochs):
        total_loss = 0
        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = data.to(device).view(data.size(0), -1), target.to(device)

            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

        print(f'Epoch {epoch+1}/{epochs}, Loss: {total_loss/len(train_loader):.4f}')
```

```python
def evaluate(model, test_loader, device):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device).view(data.size(0), -1), target.to(device)
            outputs = model(data)
            _, predicted = torch.max(outputs.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()

    accuracy = 100 * correct / total
    print(f'Test Accuracy: {accuracy:.2f}%')
    return accuracy
```

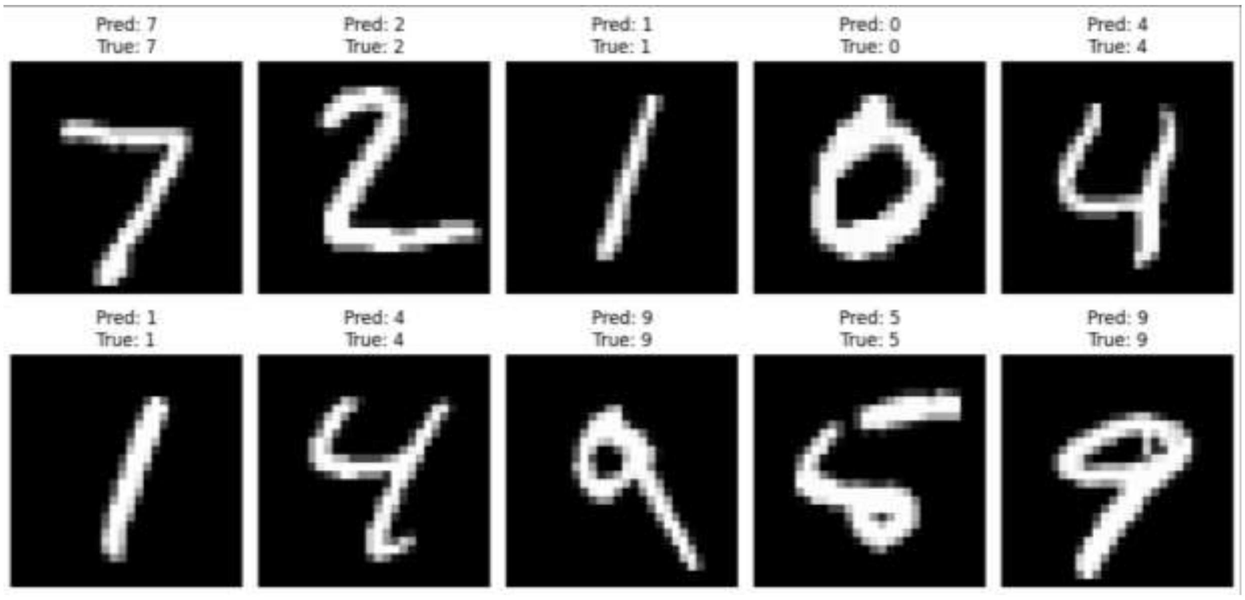
```python
train(model, train_loader, criterion, optimizer, device)
accuracy = evaluate(model, test_loader, device)
accuracies.append(accuracy)
```

```
Epoch 1/20, Loss: 0.0137
Epoch 2/20, Loss: 0.0125
Epoch 3/20, Loss: 0.0119
Epoch 4/20, Loss: 0.0142
Epoch 5/20, Loss: 0.0129
Epoch 6/20, Loss: 0.0111
Epoch 7/20, Loss: 0.0124
Epoch 8/20, Loss: 0.0080
Epoch 9/20, Loss: 0.0086
Epoch 10/20, Loss: 0.0107
Epoch 11/20, Loss: 0.0130
Epoch 12/20, Loss: 0.0073
Epoch 13/20, Loss: 0.0109
Epoch 14/20, Loss: 0.0067
Epoch 15/20, Loss: 0.0122
Epoch 16/20, Loss: 0.0102
Epoch 17/20, Loss: 0.0079
Epoch 18/20, Loss: 0.0104
Epoch 19/20, Loss: 0.0083
Epoch 20/20, Loss: 0.0116
Test Accuracy: 98.16%
```

```python
torch.save(model.state_dict(), 'mnist_model.pth')
```



```python
import matplotlib.pyplot as plt
model.eval()
with torch.no_grad():
    data, target = next(iter(test_loader))
    data, target = data.to(device), target.to(device)
    data = data.view(data.size(0), -1)
    output = model(data)
    pred = output.argmax(dim=1, keepdim=True)

    # Plot first 10 test images with predictions
    fig, axes = plt.subplots(2, 5, figsize=(12, 6))
    for idx, ax in enumerate(axes.flat):
        image = data[idx].cpu().reshape(28, 28)
        ax.imshow(image, cmap='gray')
        ax.axis('off')
        ax.set_title(f'Pred: {pred[idx].item()}\nTrue: {target[idx].item()}')
    plt.tight_layout()
    plt.show()
```



Conclusion:

Handwritten digit classification using neural networks effectively recognizes digits from images by achieving an accuracy of **98.16%.** PyTorch simplifies model building, training, and evaluation with its flexible and dynamic framework. Proper data processing and optimized training parameters improve model accuracy and efficiency. The model learns to classify digits through iterative training using loss functions and optimizers. A well-trained model can be saved and used for real-world applications like digit recognition in banking and automation.