REQURIED LIB

```
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import torchvision
from torchvision.datasets import MNIST
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

```

```
# Check if CUDA (GPU) is available, otherwise use CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
device
```

```
device(type='cuda')
```

Data Preprocessing and Loading Dataset

```
# transformations for the images
transform = transforms.ToTensor()

# Load the MNIST dataset and create data loaders
train_dataset = MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = MNIST(root='./data', train=False, transform=transform, download=True)

train_dataloader = DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_dataloader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 104129574.91it/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
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 41670345.31it/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
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 31281877.83it/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
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 14848424.60it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

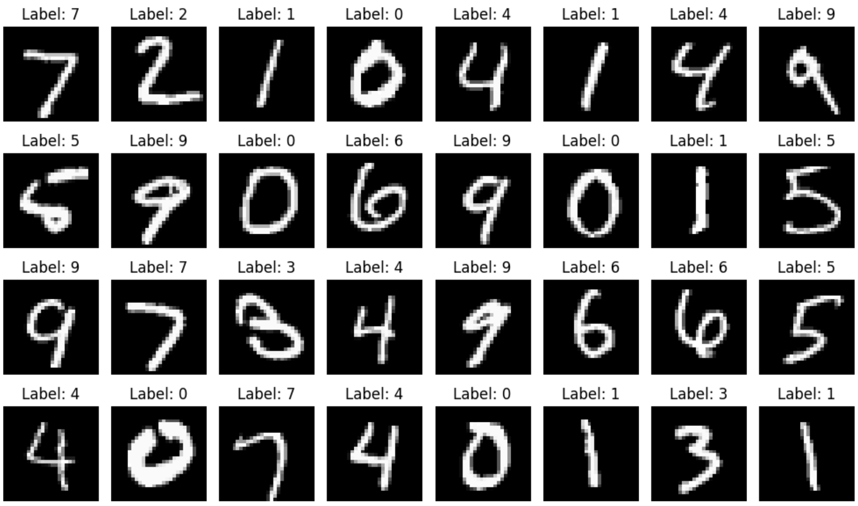
```
train_dataset
```

```
Dataset MNIST
    Number of datapoints: 60000
    Root location: ./data
    Split: Train
    StandardTransform
Transform: ToTensor()
```

Visualizing the Data

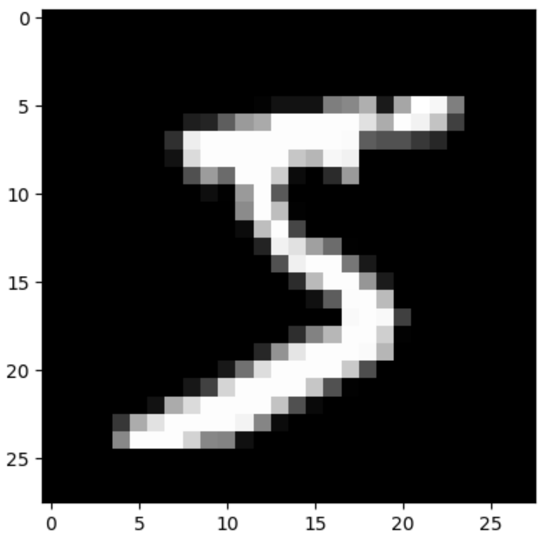
Let's visualize a few examples from the dataset to understand the input images better.

```
import matplotlib.pyplot as plt
import numpy as np

# Function to display a grid of images
def show_images(images, labels, nrows, ncols):
    fig, axes = plt.subplots(nrows, ncols, figsize=(10, 6))
    for i, ax in enumerate(axes.flat):
        ax.imshow(np.squeeze(images[i]), cmap='gray')
        ax.set_title(f"Label: {labels[i]}")
        ax.axis('off')
    plt.tight_layout()
    plt.show()

# Get a batch of images and labels from the test dataset
images, labels = next(iter(test_dataloader))

# Display the images and their labels
show_images(images, labels, 4, 8)
```




```
img , label = train_dataset[0]
plt.imshow(np.squeeze(img), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x790162f065c0>
```



Model Architecture

Next, we'll define our CNN model architecture using PyTorch.

```

class DigitClassification(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5)
        self.pool = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=8, kernel_size=3)
        self.L1 = nn.Linear(in_features=8*5*5, out_features=64)
        self.L2 = nn.Linear(in_features=64, out_features=10)

    def forward(self, x):
        x = self.conv1(x)  # 1*28*28 --> 16*24*24
        x = self.pool(x)
        x = F.relu(x)

        x = self.conv2(x)
        x = self.pool(x)
        x = F.relu(x)

        x = x.flatten(start_dim=1, end_dim=-1)
        x = F.relu(self.L1(x))
        x = F.relu(self.L2(x))
        return x
```

Training the Model

Define Loss Function and Optimizer We need to specify the loss function and optimizer for training the model.

```
# Instantiate the model and move it to the appropriate device
model = DigitClassification().to(device)

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Training Loop

Now, we'll implement the training loop to train our model on the MNIST dataset.

```
# Define the training function
def train(model, train_dataloader, criterion, optimizer):
    total_loss = 0
    for batch in train_dataloader:
        data, label = batch[0].to(device), batch[1].to(device)  # Move data to device

        optimizer.zero_grad()

        predicted = model(data)
        loss = criterion(predicted, label)

        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss

```

TEST loop

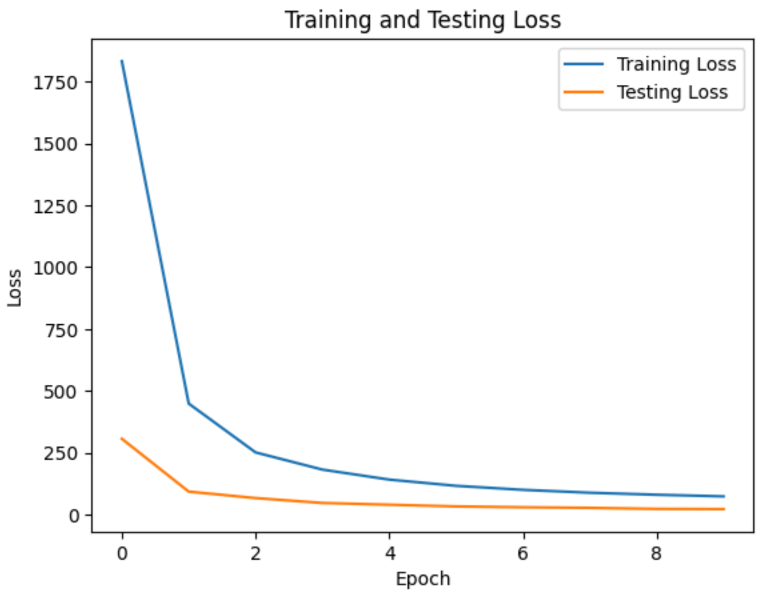
```
def test(model, test_dataloader, criterion, optimizer):
    total_loss = 0
    with torch.no_grad():  # No need to track gradients during testing
        for batch in test_dataloader:
            data, label = batch[0].to(device), batch[1].to(device)  # Move data to device

            predicted = model(data)
            loss = criterion(predicted, label)

            total_loss += loss.item()
    return total_loss
```

```
# Train and test the model for a certain number of epochs
num_epochs = 10
training_losses = []
testing_losses = []
```

```
for epoch in tqdm(range(num_epochs), desc='Training Progress'):

    training_loss = train(model, train_dataloader, criterion, optimizer)
    testing_loss = test(model, test_dataloader, criterion, optimizer)

    # Append losses for visualization
    training_losses.append(training_loss)
    testing_losses.append(testing_loss)

    print(f"Training Loss: {training_loss}, Testing Loss: {testing_loss}")
```

```
Training Progress:  10%|█         | 1/10 [00:11<01:41, 11.26s/it]Training Loss: 1833.5078232884407, Testing Loss: 305.89
Training Progress:  20%|██        | 2/10 [00:20<01:20, 10.04s/it]Training Loss: 448.25220078229904, Testing Loss: 92.100
Training Progress:  30%|███       | 3/10 [00:29<01:07,  9.60s/it]Training Loss: 250.6873406395316, Testing Loss: 65.9009
Training Progress:  40%|████      | 4/10 [00:39<00:58,  9.72s/it]Training Loss: 181.1206422690302, Testing Loss: 46.1270
Training Progress:  50%|█████     | 5/10 [00:48<00:46,  9.38s/it]Training Loss: 140.67512695118785, Testing Loss: 39.108
Training Progress:  60%|██████    | 6/10 [00:57<00:37,  9.28s/it]Training Loss: 115.48206906672567, Testing Loss: 32.162
Training Progress:  70%|███████   | 7/10 [01:06<00:27,  9.23s/it]Training Loss: 99.26806882582605, Testing Loss: 28.6702
Training Progress:  80%|████████  | 8/10 [01:15<00:18,  9.21s/it]Training Loss: 87.5746425203979, Testing Loss: 26.31545
Training Progress:  90%|█████████ | 9/10 [01:24<00:08,  8.97s/it]Training Loss: 79.41148270107806, Testing Loss: 22.1120
Training Progress: 100%|██████████| 10/10 [01:33<00:00,  9.31s/it]Training Loss: 72.75542741268873, Testing Loss: 21.402
```

```
# Plotting training and testing losses
plt.plot(training_losses, label='Training Loss')
plt.plot(testing_losses, label='Testing Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Testing Loss')
plt.legend()
plt.show()
```



Predictions and Visualization

Making Predictions

Let's make predictions on new data using the trained model.

```
# Function to make predictions on new data
def predict(model, dataloader):
    predictions = []
    with torch.no_grad():
        for data in dataloader:
            inputs = data[0].to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            predictions.extend(predicted.cpu().numpy())
    return predictions

```

```
# Make predictions on the test dataset
test_predictions = predict(model, test_dataloader)
```

Visualizing Predictions

We'll visualize a random sample of test images along with their predicted labels.

```
# Function to plot a random sample of images along with their predicted labels
def visualize_predictions(dataset, predictions, num_samples=5):
    plt.figure(figsize=(15, 7))
    samples = np.random.choice(len(dataset), num_samples, replace=False)
    for i, idx in enumerate(samples):
        plt.subplot(1, num_samples, i + 1)
        image, label = dataset[idx]
        plt.imshow(image.squeeze(), cmap='gray')
        plt.title(f"Predicted: {predictions[idx]}, Actual: {label}")
        plt.axis('off')
    plt.show()

```

```
# Visualize predictions on a random sample of test images
visualize_predictions(test_dataset, test_predictions)
```