```
from torchvision import datasets
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch


# Load MNIST dataset for training and testing
train_data = datasets.MNIST(
    root='data',          # Directory to store the dataset
    train=True,           # Load the training set
    transform=ToTensor(), # Apply ToTensor image transformation
    download=True         # Download the dataset if not already present
)

test_data = datasets.MNIST(
    root='data',
    train=False,          # Load the test set
    transform=ToTensor(),
    download=True
)

# Check the size of the datasets (number of samples, width, height)
print(train_data.data.size())  # Output: torch.Size([60000, 28, 28])
print(test_data.data.size())   # Output: torch.Size([10000, 28, 28])

# Check the labels for the training data
print(train_data.targets)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

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
100%|██████████| 9912422/9912422 [00:00<00:00, 13167454.61it/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 403: Forbidden

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
100%|██████████| 28881/28881 [00:00<00:00, 394998.25it/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 403: Forbidden

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 t
100%|██████████| 1648877/1648877 [00:00<00:00, 3437466.99it/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 403: Forbidden

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 t
100%|██████████| 4542/4542 [00:00<00:00, 9937678.02it/s]Extracting data/MNIST/raw/t10

torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
tensor([5, 0, 4,  ..., 5, 6, 8])
```

```
# Create data loaders for training and test sets
loader = {
    'train': DataLoader(train_data,
                        batch_size=125,  # Number of samples per batch
                        shuffle=True,    # Shuffle the data at every epoch
                        num_workers=1),  # Use 1 worker for data loading
    'test': DataLoader(test_data,
                       batch_size=125,   # Number of samples per batch
                       shuffle=False,    # No need to shuffle test data
                       num_workers=1)    # Use 1 worker for data loading
}
```

```
# Define the Convolutional NN model class
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # First convolutional layer: 1 input channel (grayscale), 10 output channels, ker
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)

        # Second convolutional layer: 10 input channels, 20 output channels, kernel size
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)

        # Dropout layer after the second convolution
        self.conv2_dropout = nn.Dropout2d()

        # Fully connected layer: 320 input features, 50 output features
        self.fc1 = nn.Linear(320, 50)

        # Fully connected output layer: 50 input features, 10 output features (10 digits
        self.fc2 = nn.Linear(50, 10)
```

```
    def forward(self, x):
        # Apply first convolution, followed by ReLU and max pooling
        x = F.relu(F.max_pool2d(self.conv1(x), 2))

        # Apply second convolution, followed by dropout, ReLU, and max pooling
        x = F.relu(F.max_pool2d(self.conv2_dropout(self.conv2(x)), 2))

        # Flatten the tensor into a 1D vector (for input to fully connected layer)
        x = x.view(-1, 320)

        # Apply first fully connected layer followed by ReLU
        x = F.relu(self.fc1(x))

        # Apply dropout (during training only)
        x = F.dropout(x, training=self.training)

        # Apply second fully connected layer (output layer)
        x = self.fc2(x)

        # Use log_softmax as the final activation since we're using CrossEntropyLoss
        return F.log_softmax(x, dim=1)



# Set the device to use (GPU if available, otherwise CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Initialize the model and move it to the appropriate device (CPU or GPU)
model = CNN().to(device)

# Define the optimizer (Adam) and the learning rate
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Define the loss function (CrossEntropyLoss, which combines softmax and negative log lik
loss_fn = nn.CrossEntropyLoss()



# Define the training loop
def train(epoch):
    model.train()  # Set the model to training mode
    for batch_idx, (data, target) in enumerate(loader['train']):
        data, target = data.to(device), target.to(device)  # Move data to the correct dev
        optimizer.zero_grad()  # Zero out gradients from previous step
        output = model(data)   # Forward pass
        loss = loss_fn(output, target)  # Compute the loss
        loss.backward()  # Backpropagate the gradients
        optimizer.step()  # Update the model's weights
        if batch_idx % 20 == 0:  # Print progress every 20 batches
            print(f'Train epoch: {epoch} [{batch_idx * len(data)} / {len(loader["train"].
```

```
# Define the testing loop
def test():
    model.eval()  # Set the model to evaluation mode
    test_loss = 0
    correct = 0

    with torch.no_grad():  # Disable gradient calculation for evaluation
        for data, target in loader['test']:
            data, target = data.to(device), target.to(device)  # Move data to device
            output = model(data)  # Forward pass
            test_loss += loss_fn(output, target).item()  # Accumulate test loss
            pred = output.argmax(dim=1, keepdim=True)  # Get the index of the max log-proba
            correct += pred.eq(target.view_as(pred)).sum().item()  # Count correct predicti

    avg_loss = test_loss / len(loader['test'])  # Compute average loss per batch
    accuracy = 100 * correct / len(loader['test'].dataset)  # Compute accuracy percentage
    print(f'\nTest set: Average loss: {avg_loss:.4f}, Accuracy: {correct} / {len(loader["te
```

```
# Run the training and testing loops for 10 epochs
for epoch in range(1, 11):
    train(epoch)  # Train the model for one epoch
    test()        # Test the model after each epoch
```

```
 Train epoch: 1 [0 / 60000]
 Train epoch: 1 [2500 / 60000]
 Train epoch: 1 [5000 / 60000]
 Train epoch: 1 [7500 / 60000]
 Train epoch: 1 [10000 / 60000]
 Train epoch: 1 [12500 / 60000]
 Train epoch: 1 [15000 / 60000]
 Train epoch: 1 [17500 / 60000]
 Train epoch: 1 [20000 / 60000]
 Train epoch: 1 [22500 / 60000]
 Train epoch: 1 [25000 / 60000]
 Train epoch: 1 [27500 / 60000]
 Train epoch: 1 [30000 / 60000]
 Train epoch: 1 [32500 / 60000]
 Train epoch: 1 [35000 / 60000]
 Train epoch: 1 [37500 / 60000]
 Train epoch: 1 [40000 / 60000]
 Train epoch: 1 [42500 / 60000]
 Train epoch: 1 [45000 / 60000]
 Train epoch: 1 [47500 / 60000]
 Train epoch: 1 [50000 / 60000]
 Train epoch: 1 [52500 / 60000]
 Train epoch: 1 [55000 / 60000]
```

```
Train epoch: 1 [57500 / 60000]

Test set: Average loss: 0.1631, Accuracy: 9486 / 10000, 95%

Train epoch: 2 [0 / 60000]
Train epoch: 2 [2500 / 60000]
Train epoch: 2 [5000 / 60000]
Train epoch: 2 [7500 / 60000]
Train epoch: 2 [10000 / 60000]
Train epoch: 2 [12500 / 60000]
Train epoch: 2 [15000 / 60000]
Train epoch: 2 [17500 / 60000]
Train epoch: 2 [20000 / 60000]
Train epoch: 2 [22500 / 60000]
Train epoch: 2 [25000 / 60000]
Train epoch: 2 [27500 / 60000]
Train epoch: 2 [30000 / 60000]
Train epoch: 2 [32500 / 60000]
Train epoch: 2 [35000 / 60000]
Train epoch: 2 [37500 / 60000]
Train epoch: 2 [40000 / 60000]
Train epoch: 2 [42500 / 60000]
Train epoch: 2 [45000 / 60000]
Train epoch: 2 [47500 / 60000]
Train epoch: 2 [50000 / 60000]
Train epoch: 2 [52500 / 60000]
Train epoch: 2 [55000 / 60000]
Train epoch: 2 [57500 / 60000]

Test set: Average loss: 0.1063, Accuracy: 9656 / 10000, 97%

Train epoch: 3 [0 / 60000]
Train epoch: 3 [2500 / 60000]
Train epoch: 3 [5000 / 60000]
Train epoch: 3 [7500 / 60000]
```