```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as dsets
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
'''
LOADING DATASET
'''
train = dsets.MNIST(root='./data',
                            train=True,
                            transform=transforms.ToTensor(),  # Normalize the image to [0-1] from [0-255]
                            download=True)

test = dsets.MNIST(root='./data',
                           train=False,
                           transform=transforms.ToTensor())
```

```
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 to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 11517123.79it/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 to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 344696.61it/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 to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 2740280.05it/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 to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 9398386.17it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

```
print(len(train))
```

```
60000
```

```
from torch import tensor
traindata = [train[i] for i in range(len(train))]
train = torch.stack([d[0] for d in traindata], dim=0)
train=train[0:59999]
ys = [d[1] for d in traindata]
train_y = tensor(ys)

testdata = [test[i] for i in range(len(test))]
test = torch.stack([d[0] for d in testdata], dim=0)
test=test[0:9999]
ys = [d[1] for d in testdata]
test_y = tensor(ys)
```

Seeding

```
torch.manual_seed(120)
```

```
<torch._C.Generator at 0x7b0e9854dc90>
```

Dataset Loading

```
# train = torch.rand(6000, 28, 28, 1)
# test = torch.rand(1000, 28, 28, 1)
# train_y = torch.randint(0,9, (6000,))
# test_y = torch.randint(0,9, (1000,))
```

```
print(train_y)
```

```
tensor([5, 0, 4,  ..., 5, 6, 8])
```

Model Architeture

```
class NeuralNetwork(nn.Module):
    def __init__(self, input_dim):
        super(NeuralNetwork, self).__init__()
        self.cnn_layer_1 = nn.Conv2d(in_channels=1, out_channels=16,kernel_size=5, stride=1, padding=2)
        self.cnn_layer_2 = nn.Conv2d(in_channels=16, out_channels=32,kernel_size=5, stride=1, padding=2)

        self.flatten = nn.Flatten()
        self.maxpool = nn.MaxPool2d(2,2)

        self.linear_layer_1 = nn.Linear(32*7*7, 512)
        self.linear_layer_2 = nn.Linear(512, 128)
        self.linear_layer_3 = nn.Linear(128, 10)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(.2)

        # self.flatten = nn.Flatten()

    def forward(self, x):

        x = self.cnn_layer_1(x)
        x = self.dropout(x)
        x = self.relu(x)
        x = self.maxpool(x)

        #print(x.shape)

        x = self.cnn_layer_2(x)
        x = self.dropout(x)
        x = self.relu(x)
        x = self.maxpool(x)

        #print(x.shape)

        x = self.flatten(x)
        #print(x.shape)

        x = self.linear_layer_1(x)
        x = self.dropout(x)
        x = self.relu(x)

        x = self.linear_layer_2(x)
        x = self.dropout(x)
        x = self.relu(x)

        x = self.linear_layer_3(x)
        #logits = self.sigmoid(x)
        return x
```

Model Creation

```
model = NeuralNetwork(28*28)
print(model.to(device))
```

```
NeuralNetwork(
  (cnn_layer_1): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (cnn_layer_2): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (linear_layer_1): Linear(in_features=1568, out_features=512, bias=True)
  (linear_layer_2): Linear(in_features=512, out_features=128, bias=True)
  (linear_layer_3): Linear(in_features=128, out_features=10, bias=True)
  (relu): ReLU()
  (sigmoid): Sigmoid()
  (dropout): Dropout(p=0.2, inplace=False)
)
```

```
from torchsummary import summary
summary(model,(1, 28,28))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 28, 28]             416
```

```
           Dropout-2          [-1, 16, 28, 28]               0
              ReLU-3          [-1, 16, 28, 28]               0
         MaxPool2d-4          [-1, 16, 14, 14]               0
            Conv2d-5          [-1, 32, 14, 14]          12,832
           Dropout-6          [-1, 32, 14, 14]               0
              ReLU-7          [-1, 32, 14, 14]               0
         MaxPool2d-8            [-1, 32, 7, 7]               0
           Flatten-9                [-1, 1568]               0
           Linear-10                 [-1, 512]         803,328
          Dropout-11                 [-1, 512]               0
             ReLU-12                 [-1, 512]               0
           Linear-13                 [-1, 128]          65,664
          Dropout-14                 [-1, 128]               0
             ReLU-15                 [-1, 128]               0
           Linear-16                  [-1, 10]           1,290
================================================================
Total params: 883,530
Trainable params: 883,530
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.49
Params size (MB): 3.37
Estimated Total Size (MB): 3.87
----------------------------------------------------------------
```

Optimizer

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=.001)
```

```
def trainModel(model, loss_fn, optimizer):
    model.train()

    batch = 100

    loss = 0
    for i in range(train.shape[0]):
      x, y = torch.reshape(train[i],(1,1,28,28)).to(device), torch.tensor([train_y[i]], dtype=torch.float).to(device)

      label=torch.zeros([1,10,], dtype=torch.float32).to(device)
      label[0,int(y.item())]=1
      # Compute prediction error
      pred = model(x)
      #print(pred)
      #print(label)
      loss += loss_fn(pred, label)

      if i>0 and (i+1)%batch == 0:
          # Backpropagation
          optimizer.zero_grad()
          loss.backward()
          optimizer.step()
          print(f'Training Loss: {loss.item():.4f}', end="\r")
          loss = 0
    print()
```

Model testing

```
def testModel(model, loss_fn):
    model.eval()

    size = test.shape[0]
    correct=0
    loss = 0
    total =10000
    with torch.no_grad():
      for i in range(test.shape[0]):
        x, y = torch.reshape(test[i],(1,1,28,28)).to(device), torch.tensor([test_y[i]], dtype=torch.float).to(device)
        label=torch.zeros([1,10,], dtype=torch.float32).to(device)
        label[0,int(y.item())]=1
        pred = model(x)
        #print(pred)
        predicted = torch.argmax(pred)
        #print(predicted)
        #print(y)

epochs = 5
for t in range(epochs):
```