```python
import numpy as np
import matplotlib.pyplot as plt
import torch
from torchvision import transforms, datasets
```

```python
transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])

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

```python
data_path = './MNIST'

data_test   = datasets.MNIST(root = data_path, train= True, download=True, transform= transform)
data_train  = datasets.MNIST(root = data_path, train= False, download=True, transform= transform)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./MNIST/MNIST/raw/train-images-idx3-ubyte.gz
90%                                   8880128/9912422 [00:01<00:00, 1470817.77it/s]
```

```python
print("the number of your training data (must be 10,000) = ", data_train.__len__())
print("hte number of your testing data (must be 60,000) = ", data_test.__len__())
```

```
the number of your training data (must be 10,000) =  10000
hte number of your testing data (must be 60,000) =  60000
```

```python
train_loader = torch.utils.data.DataLoader(data_train, batch_size = 64, num_workers=0)
test_loader = torch.utils.data.DataLoader(data_test, batch_size=64, num_workers=0)
```

```python
import torch.nn as nn

def calc_out(in_layers, stride, padding, kernel_size, pool_stride):
    """
    Helper function for computing the number of outputs from a
    conv layer
    """
    return int((1+(in_layers - kernel_size + (2*padding))/stride)/pool_stride)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        # Some helpful values
        inputs      = [1,32,64,64] # MNIST data shape
        kernel_size = [5,5,3]
        stride      = [1,1,1]
        pool_stride = [2,2,2]

        # Layer lists
        layers = []

        self.out   = 28
        self.depth = inputs[-1]
        for i in range(len(kernel_size)):
```

```
        for i in range(len(kernel_size)):
            # Get some variables
            padding = int(kernel_size[i]/2)

            # Define the output from this layer
            self.out = calc_out(self.out, stride[i], padding,
                                kernel_size[i], pool_stride[i])

            # convolutional layer 1
            layers.append(nn.Conv2d(inputs[i], inputs[i+1], kernel_size[i],
                                       stride=stride[i], padding=padding))
            layers.append(nn.BatchNorm2d(inputs[i+1]))
            layers.append(nn.ReLU())

            # convolutional layer 2
            layers.append(nn.Conv2d(inputs[i+1], inputs[i+1], kernel_size[i],
                                       stride=stride[i], padding=padding))
            layers.append(nn.BatchNorm2d(inputs[i+1]))
            layers.append(nn.ReLU())
            # maxpool layer
            layers.append(nn.MaxPool2d(pool_stride[i],pool_stride[i]))
            layers.append(nn.Dropout(p=0.2))

        self.cnn_layers = nn.Sequential(*layers)

        print(self.depth*self.out*self.out)

        # Now for our fully connected layers
        layers2 = []
        layers2.append(nn.Dropout(p=0.2))
        layers2.append(nn.Linear(self.depth*self.out*self.out, 512))
        layers2.append(nn.Dropout(p=0.2))
        layers2.append(nn.Linear(512, 256))
        layers2.append(nn.Dropout(p=0.2))
        layers2.append(nn.Linear(256, 256))
        layers2.append(nn.Dropout(p=0.2))
        layers2.append(nn.Linear(256, 10))

        self.fc_layers = nn.Sequential(*layers2)
```

```
        self.fc_layers = nn.Sequential(*layers2)

    def forward(self, x):
        x = self.cnn_layers(x)
        x = x.view(-1, self.depth*self.out*self.out)
        x = self.fc_layers(x)
        return x

# create a complete CNN
model = Net()
model
```

```
tLoss, vLoss = [], []
tacc, vacc = [], []
```

```
import torch.optim as optim

# specify loss function
criterion = nn.CrossEntropyLoss()

# specify optimizer
optimizer = optim.Adam(model.parameters(), lr=0.0005)
```

```
# number of epochs to train the model
n_epochs = 30

# Get the device
model.to(device)

for epoch in range(n_epochs):
    # keep track of training and validation loss
    train_loss = 0.0
    test_loss = 0.0
    train_correct = 0
    test_correct = 0

```

```
    # train #
    model.train()
    for data, target in train_loader:
        # move tensors to GPU if CUDA is available
        data   = data.to(device)
        target = target.to(device)

        # clear the gradients of all optimized variables
        optimizer.zero_grad()
        output = model(data)

        # calculate the batch loss
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()*data.size(0)

        prediction = output.data.max(1)[1]   # first column has actual prob.
        correct = prediction.eq(target.data).sum()
        train_correct += correct


    # test #
    model.eval()
    for data, target in test_loader:
        # move tensors to GPU if CUDA is available
        data   = data.to(device)
        target = target.to(device)

        output = model(data)

        # calculate the batch loss
        loss = criterion(output, target)
        test_loss += loss.item()*data.size(0)

        prediction = output.data.max(1)[1]   # first column has actual prob.
        correct = prediction.eq(target.data).sum()
        test_correct += correct
```

```
        test_correct += correct

    # calculate average losses
    train_loss = train_loss/len(train_loader.dataset)
    test_loss = test_loss/len(test_loader.dataset)
    tLoss.append(train_loss)
    vLoss.append(test_loss)

    # calculate average accuracy
    train_accuracy = train_correct / len(train_loader.dataset)
    test_accuracy = test_correct / len(test_loader.dataset)
    tacc.append(train_accuracy)
    vacc.append(test_accuracy)

    # print training/validation statistics
    print('Epoch: {} ₩tTraining Loss: {:.5f} ₩tTesting Loss: {:.5f}'.format(
        epoch, train_loss, test_loss))
    print('Epoch: {} ₩tTraining acc: {:.5f} ₩tTesting acc: {:.5f}'.format(
        epoch, train_accuracy, test_accuracy))
```

```
cuda
Epoch: 0        Training Loss: 0.48889  Testing Loss: 0.24120
Epoch: 0        Training acc: 0.83620   Testing acc: 0.93440
Epoch: 1        Training Loss: 0.12925  Testing Loss: 0.19014
Epoch: 1        Training acc: 0.95930   Testing acc: 0.94888
Epoch: 2        Training Loss: 0.09355  Testing Loss: 0.15203
Epoch: 2        Training acc: 0.96970   Testing acc: 0.96112
Epoch: 3        Training Loss: 0.06981  Testing Loss: 0.09111
Epoch: 3        Training acc: 0.97840   Testing acc: 0.97465
Epoch: 4        Training Loss: 0.05037  Testing Loss: 0.10475
Epoch: 4        Training acc: 0.98300   Testing acc: 0.97467
Epoch: 5        Training Loss: 0.05559  Testing Loss: 0.10922
Epoch: 5        Training acc: 0.98190   Testing acc: 0.97373
Epoch: 6        Training Loss: 0.05032  Testing Loss: 0.08810
Epoch: 6        Training acc: 0.98460   Testing acc: 0.97850
Epoch: 7        Training Loss: 0.04541  Testing Loss: 0.09403
Epoch: 7        Training acc: 0.98440   Testing acc: 0.97637
Epoch: 8        Training Loss: 0.03695  Testing Loss: 0.08212
Epoch: 8        Training acc: 0.98780   Testing acc: 0.97995
Epoch: 9        Training Loss: 0.03521  Testing Loss: 0.11271
```

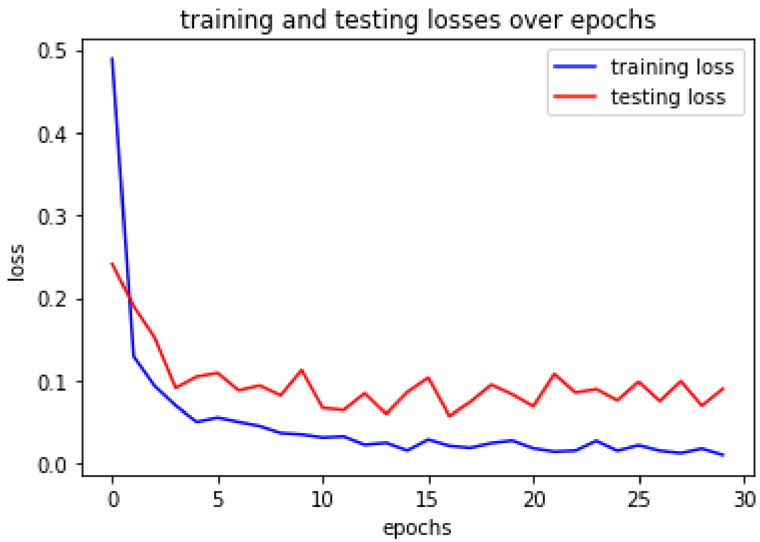
```
Epoch: 9        Training acc: 0.98900   Testing acc: 0.97473
Epoch: 10       Training Loss: 0.03152  Testing Loss: 0.06703
Epoch: 10       Training acc: 0.99000   Testing acc: 0.98387
Epoch: 11       Training Loss: 0.03275  Testing Loss: 0.06449
Epoch: 11       Training acc: 0.98950   Testing acc: 0.98433
Epoch: 12       Training Loss: 0.02276  Testing Loss: 0.08469
Epoch: 12       Training acc: 0.99160   Testing acc: 0.98163
Epoch: 13       Training Loss: 0.02510  Testing Loss: 0.05986
Epoch: 13       Training acc: 0.99270   Testing acc: 0.98657
Epoch: 14       Training Loss: 0.01584  Testing Loss: 0.08609
Epoch: 14       Training acc: 0.99500   Testing acc: 0.98363
Epoch: 15       Training Loss: 0.02910  Testing Loss: 0.10357
Epoch: 15       Training acc: 0.99080   Testing acc: 0.97780
Epoch: 16       Training Loss: 0.02150  Testing Loss: 0.05728
Epoch: 16       Training acc: 0.99360   Testing acc: 0.98757
Epoch: 17       Training Loss: 0.01902  Testing Loss: 0.07424
Epoch: 17       Training acc: 0.99360   Testing acc: 0.98402
Epoch: 18       Training Loss: 0.02480  Testing Loss: 0.09501
Epoch: 18       Training acc: 0.99380   Testing acc: 0.97943
Epoch: 19       Training Loss: 0.02785  Testing Loss: 0.08328
Epoch: 19       Training acc: 0.99300   Testing acc: 0.98238
Epoch: 20       Training Loss: 0.01852  Testing Loss: 0.06883
Epoch: 20       Training acc: 0.99460   Testing acc: 0.98535
Epoch: 21       Training Loss: 0.01451  Testing Loss: 0.10794
Epoch: 21       Training acc: 0.99540   Testing acc: 0.98095
Epoch: 22       Training Loss: 0.01561  Testing Loss: 0.08543
Epoch: 22       Training acc: 0.99400   Testing acc: 0.98485
Epoch: 23       Training Loss: 0.02758  Testing Loss: 0.08960
Epoch: 23       Training acc: 0.99160   Testing acc: 0.98168
Epoch: 24       Training Loss: 0.01557  Testing Loss: 0.07596
Epoch: 24       Training acc: 0.99420   Testing acc: 0.98552
Epoch: 25       Training Loss: 0.02221  Testing Loss: 0.09837
Epoch: 25       Training acc: 0.99300   Testing acc: 0.98017
Epoch: 26       Training Loss: 0.01557  Testing Loss: 0.07513
Epoch: 26       Training acc: 0.99570   Testing acc: 0.98502
Epoch: 27       Training Loss: 0.01268  Testing Loss: 0.09921
Epoch: 27       Training acc: 0.99650   Testing acc: 0.98045
Epoch: 28       Training Loss: 0.01817  Testing Loss: 0.06952
Epoch: 28       Training acc: 0.99450   Testing acc: 0.98652
```

Output

1. Plot the training and testing losses over epochs [2pt]

```
plt.plot([i for i in range(len(tLoss))], tLoss, label = 'training loss' , c = 'blue')
plt.plot([i for i in range(len(vLoss))], vLoss, label = 'testing loss' , c = 'red')

plt.legend()
plt.xlabel("epochs")
plt.ylabel("loss")
plt.title("training and testing losses over epochs")
plt.show()
```

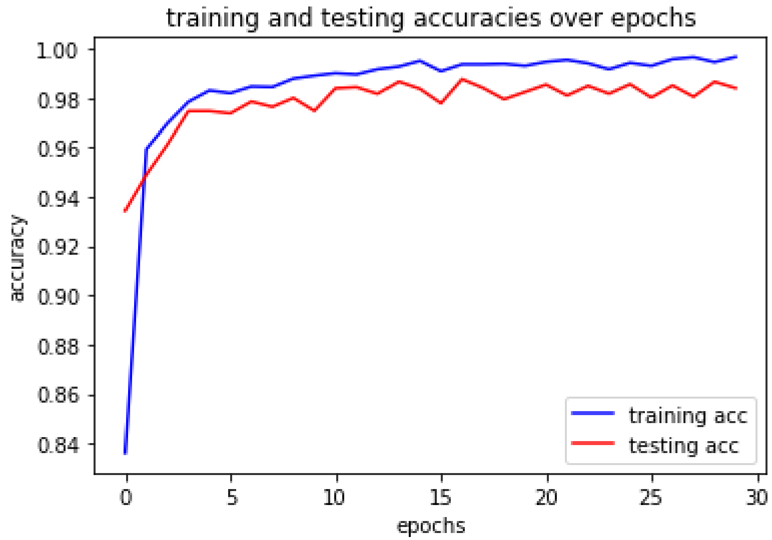


2. Plot the training and testing accuracies over epochs [2pt]

```
plt.plot([i for i in range(len(tacc))], tacc, label = 'training acc' , c = 'blue')
plt.plot([i for i in range(len(vacc))], vacc, label = 'testing acc' , c = 'red')

```

```
plt.legend()
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.title("training and testing accuracies over epochs")
plt.show()
```



3. Print the final training and testing losses at convergence [2pt]

```
print('Training loss: {:.5f} ₩nTesting loss: {:.5f}'.format(tLoss[-1], vLoss[-1]))
```

```
Training loss: 0.01050
Testing loss: 0.08998
```

4. Print the final training and testing accuracies at convergence [20pt]

```
print('Training acc: {:.5f} ₩nTesting acc: {:.5f}'.format(tacc[-1], vacc[-1]))
```

```
Training acc: 0.99660
```

```
Testing acc: 0.98395
```

5. Print the testing accuracies within the last 10 epochs [5pt]

```
vacc_last_10_epochs = vacc[n_epochs - 10:]

for i in range(len(vacc_last_10_epochs)):
    print("[epoch = {}] ₩tTesting acc : {:.5f}".format(n_epochs - 10 + i, vacc_last_10_epochs[i]))
```

```
[epoch = 20]    Testing acc : 0.98535
[epoch = 21]    Testing acc : 0.98095
[epoch = 22]    Testing acc : 0.98485
[epoch = 23]    Testing acc : 0.98168
[epoch = 24]    Testing acc : 0.98552
[epoch = 25]    Testing acc : 0.98017
[epoch = 26]    Testing acc : 0.98502
[epoch = 27]    Testing acc : 0.98045
[epoch = 28]    Testing acc : 0.98652
[epoch = 29]    Testing acc : 0.98395
```