**DL- CNN Sample code 2**

import torch

import torch.nn as nn

import torchvision

import torch.nn.functional as F

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

device

# apply transformation

transform = transforms.Compose([transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

# download the data

training\_data = torchvision.datasets.EMNIST(root='contents/',download=True, transform=transform, train=True,split='balanced')

test\_data = torchvision.datasets.EMNIST(root='contents/',download=True, transform=transform, train=False,split='balanced')

# build the data loader

train\_loader = torch.utils.data.DataLoader(dataset = training\_data,

batch\_size = 128,

shuffle = True)

test\_loader = torch.utils.data.DataLoader(dataset = test\_data,

batch\_size = 128,

shuffle = True)

for i in range(6):

plt.subplot(2,3,i+1)

plt.imshow(training\_data.data[i])

plt.title('Ground Truth: {}'.format(training\_data.classes[training\_data.targets[i]]))

plt.axis('off')

class CNNModel(nn.Module):

def \_\_init\_\_(self):

super(CNNModel, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(in\_channels=1, out\_channels=32, kernel\_size= 5,stride = 1)

self.conv2 = nn.Conv2d(32, 64, 5, 1)

self.fc = nn.Linear(64\*20\*20, 47)

def forward(self, x):

x = F.relu(self.conv1(x))

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

x = F.max\_pool2d(x, 1)

x = torch.flatten(x, 1)

x = self.fc(x)

output = F.log\_softmax(x, dim=1)

return output

# initiating the model

model = CNNModel()

# loss function

criterion = nn.CrossEntropyLoss()

# Optimizer

optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

# fetch model to the working device

model.to(device)

# training loss

train\_loss = []

test\_losses =[]

def train(e):

#Load in the data

for i, (images, labels) in enumerate(train\_loader):

# load data on to device

images = images.to(device)

labels = labels.to(device)

# Forward pass

outputs = model(images)

loss = criterion(outputs, labels)

# Backward and optimize

optimizer.zero\_grad()

loss.backward()

optimizer.step()

train\_loss.append(loss.item())

print('Epoch [{}/{}], Train Loss: {:.4f}'.format(e+1, 10, loss.item()))

def test():

test\_loss = 0

with torch.no\_grad():

correct = 0

total = 0

for images, labels in test\_loader:

images = images.to(device)

labels = labels.to(device)

outputs = model(images)

test\_loss += F.nll\_loss(outputs, labels, size\_average=False).item()

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

test\_loss /= len(test\_loader.dataset)

test\_losses.append(test\_loss)

print('Test Accuracy: {:4f} %, Test loss: {:4f}'.format((100 \* correct / total),test\_loss))

for i in range(10):

train(i)

test()

plt.figure(figsize=(7,5))

plt.plot(train\_loss,label='Train loss')

plt.plot(test\_losses, label='Test loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Loss vs Epochs')

plt.legend()