import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

import numpy as np

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

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

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

train\_dataset = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False)

class Net(nn.Module):

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

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

self.fc1 = nn.Linear(28\*28, 256)

self.fc2 = nn.Linear(256, 128)

self.fc3 = nn.Linear(128, 10)

self.relu = nn.ReLU()

def forward(self, x):

x = x.view(-1, 28\*28)

x = self.relu(self.fc1(x))

x = self.relu(self.fc2(x))

x = self.fc3(x)

return x

model = Net().to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 10

train\_acc\_list, test\_acc\_list, loss\_list = [], [], []

for epoch in range(epochs):

model.train()

correct, total, running\_loss = 0, 0, 0

for images, labels in train\_loader:

images, labels = images.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(images)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

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

total += labels.size(0)

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

train\_acc = 100 \* correct / total

loss\_list.append(running\_loss / len(train\_loader))

train\_acc\_list.append(train\_acc)

model.eval()

correct, total = 0, 0

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

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

total += labels.size(0)

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

test\_acc = 100 \* correct / total

test\_acc\_list.append(test\_acc)

print(f'Epoch {epoch+1}/{epochs}, Loss: {loss\_list[-1]:.4f}, Train Acc: {train\_acc:.2f}%, Test Acc: {test\_acc:.2f}%')

plt.plot(loss\_list, label='Loss')

plt.plot(train\_acc\_list, label='Train Acc')

plt.plot(test\_acc\_list, label='Test Acc')

plt.legend()

plt.show()

model.eval()

y\_true, y\_pred = [], []

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

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

y\_true.extend(labels.cpu().numpy())

y\_pred.extend(predicted.cpu().numpy())

cm = confusion\_matrix(y\_true, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=train\_dataset.classes)

disp.plot(cmap=plt.cm.Blues, xticks\_rotation='vertical')

plt.show()

examples = enumerate(test\_loader)

batch\_idx, (example\_data, example\_targets) = next(examples)

with torch.no\_grad():

output = model(example\_data.to(device))

pred = output.argmax(dim=1, keepdim=True).cpu().numpy()

plt.figure(figsize=(12,4))

for i in range(10):

plt.subplot(2,5,i+1)

plt.imshow(example\_data[i][0], cmap='gray')

color = 'green' if pred[i][0] == example\_targets[i].item() else 'red'

plt.title(train\_dataset.classes[pred[i][0]], color=color)

plt.axis('off')

plt.show()