*# Step 1: Import Libraries*

import torch

import torch.nn as nn import torch.optim as optim

from torch.utils.data import DataLoader import torchvision

import torchvision.transforms as transforms import matplotlib.pyplot as plt

from sklearn.decomposition import PCA from sklearn.manifold import TSNE import numpy as np

*# Step 2: Set Device (GPU if available)*

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

*# Step 3: Load Fashion MNIST Dataset*

transform = transforms.Compose([transforms.ToTensor(), transforms.Lambda(lambda x: x.view(-1))]) *# Flatten images*

trainset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)

testset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

train\_loader = DataLoader(trainset, batch\_size=128, shuffle=True) test\_loader = DataLoader(testset, batch\_size=128, shuffle=False)

*# Step 4: Define VAE Model*

class VAE(nn.Module):

def init (self, latent\_dim=2): super(VAE, self). init () self.latent\_dim = latent\_dim

*# Encoder*

self.fc1 = nn.Linear(784, 512)

self.fc21 = nn.Linear(512, latent\_dim) *# Mean*

self.fc22 = nn.Linear(512, latent\_dim) *# Log variance*

*# Decoder*

self.fc3 = nn.Linear(latent\_dim, 512) self.fc4 = nn.Linear(512, 784)

def encode(self, x):

h1 = torch.relu(self.fc1(x))

return self.fc21(h1), self.fc22(h1) def reparameterize(self, mu, logvar):

std = torch.exp(0.5 \* logvar) eps = torch.randn\_like(std) return mu + eps \* std

def decode(self, z):

h3 = torch.relu(self.fc3(z)) return torch.sigmoid(self.fc4(h3))

def forward(self, x):

mu, logvar = self.encode(x.view(-1, 784)) z = self.reparameterize(mu, logvar) return self.decode(z), mu, logvar

def loss\_function(self, recon\_x, x, mu, logvar):

BCE = nn.functional.binary\_cross\_entropy(recon\_x, x.view(-1, 784), reduction='sum')

*# KL divergence*

KL = -0.5 \* torch.sum(1 + logvar - mu.pow(2) - logvar.exp()) return BCE + KL

*# Step 5: Set Up Training Loop*

model = VAE(latent\_dim=2).to(device)

optimizer = optim.Adam(model.parameters(), lr=1e-3)

num\_epochs = 10 train\_losses = []

for epoch in range(num\_epochs): model.train()

train\_loss = 0

for batch\_idx, (data, \_) in enumerate(train\_loader): data = data.to(device)

optimizer.zero\_grad()

recon\_batch, mu, logvar = model(data)

loss = model.loss\_function(recon\_batch, data, mu, logvar) loss.backward()

train\_loss += loss.item() optimizer.step()

avg\_train\_loss = train\_loss / len(train\_loader.dataset) train\_losses.append(avg\_train\_loss)

print(f"Epoch {epoch+1}/{num\_epochs}, Average Training Loss:

{avg\_train\_loss:.4f}")

*# Save reconstructed images at intervals*

if (epoch + 1) % 5 == 0: model.eval()

test\_data, \_ = next(iter(test\_loader)) test\_data = test\_data.to(device)

with torch.no\_grad():

recon\_images, \_, \_ = model(test\_data) comparison = torch.cat([test\_data[:8],

recon\_images.view(128, 784)[:8]])

comparison = comparison.view(16, 28, 28).cpu() plt.figure(figsize=(10, 5))

for i in range(16): plt.subplot(4, 8, i + 1)

plt.imshow(comparison[i], cmap="gray") plt.axis("off")

plt.show()

*# Plot training loss* plt.plot(train\_losses) plt.title("Training Loss") plt.xlabel("Epochs") plt.ylabel("Loss") plt.show()

*# Step 6: Latent Space Visualization (PCA + t-SNE)*

model.eval()

with torch.no\_grad(): data\_points = [] labels = []

for data, target in test\_loader: data = data.to(device)

mu, \_ = model.encode(data.view(-1, 784)) data\_points.append(mu.cpu().numpy()) labels.append(target.numpy())

data\_points = np.concatenate(data\_points) labels = np.concatenate(labels)

*# PCA for dimensionality reduction to 2D*

pca = PCA(n\_components=2)

reduced\_data = pca.fit\_transform(data\_points)

*# t-SNE for better visualization*

tsne = TSNE(n\_components=2, random\_state=0) reduced\_data\_tsne = tsne.fit\_transform(data\_points)

plt.figure(figsize=(8, 6))

plt.scatter(reduced\_data\_tsne[:, 0], reduced\_data\_tsne[:, 1], c=labels, cmap="tab10")

plt.colorbar()

plt.title("Latent Space Visualization (t-SNE)") plt.show()

*# Step 7: Generate New Images from Latent Space*

model.eval()

with torch.no\_grad():

z = torch.randn(64, 2).to(device) *# Sample from the standard normal distribution*

generated\_images = model.decode(z).cpu()

*# Display the generated images*

generated\_images = generated\_images.view(64, 28, 28) fig, ax = plt.subplots(8, 8, figsize=(10, 10))

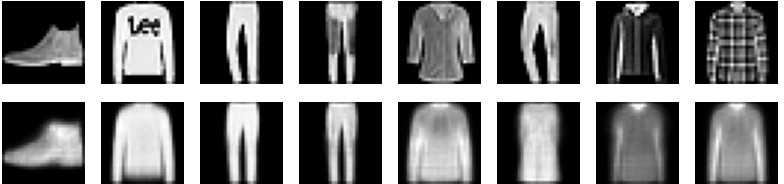
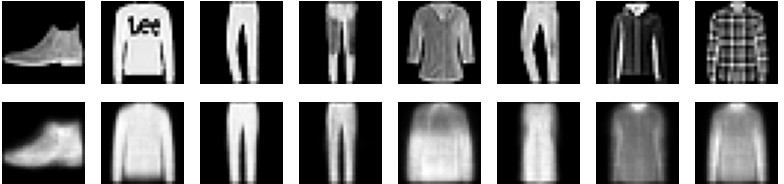
for i in range(8):

for j in range(8):

ax[i, j].imshow(generated\_images[i \* 8 + j], cmap="gray") ax[i, j].axis('off')

plt.show()

Epoch 1/10, Average Training Loss: 287.6970 Epoch 2/10, Average Training Loss: 271.6543 Epoch 3/10, Average Training Loss: 268.9053 Epoch 4/10, Average Training Loss: 267.2170 Epoch 5/10, Average Training Loss: 266.1489



Epoch 6/10, Average Training Loss: 265.2597 Epoch 7/10, Average Training Loss: 264.6352 Epoch 8/10, Average Training Loss: 264.1574 Epoch 9/10, Average Training Loss: 263.7446 Epoch 10/10, Average Training Loss: 263.3032

