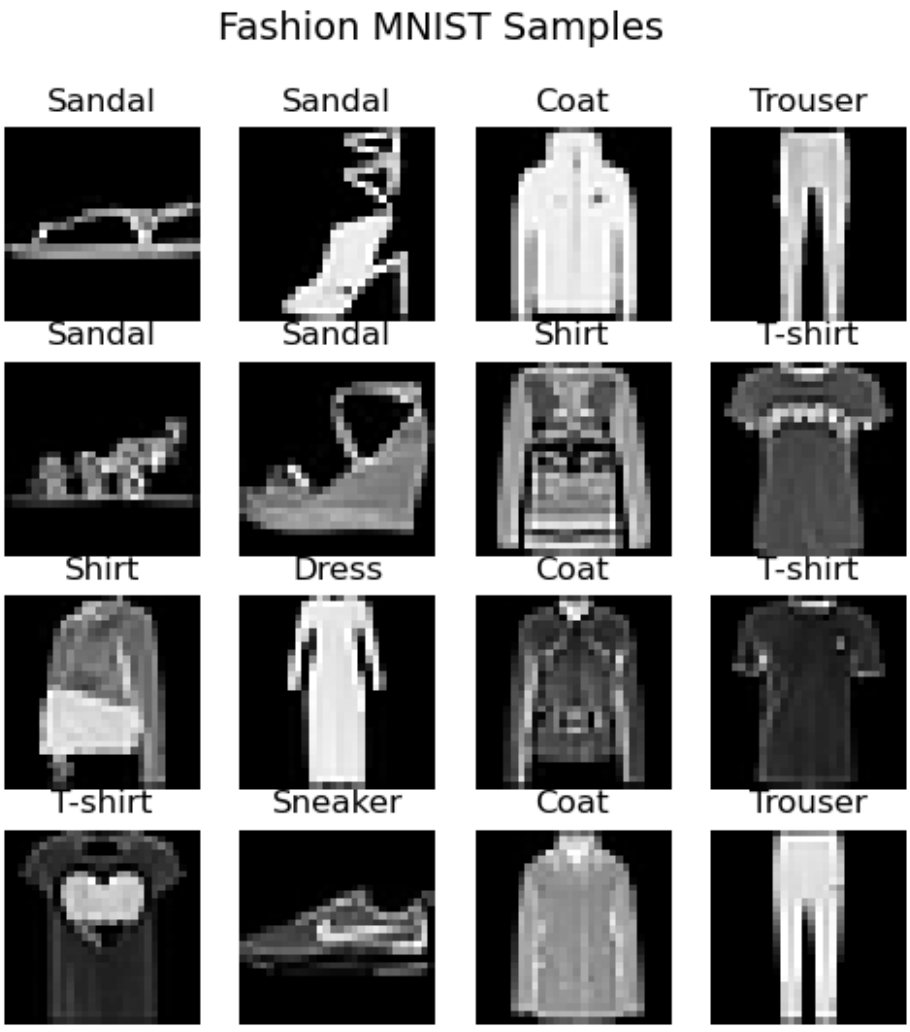
```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torchvision.utils import make_grid
from torch.utils.data import DataLoader

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA

# Define transformations (convert to tensor & normalize)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Load Fashion MNIST dataset
train_dataset = torchvision.datasets.FashionMNIST(root="./data",
train=True, transform=transform, download=True)
test_dataset = torchvision.datasets.FashionMNIST(root="./data",
train=False, transform=transform, download=True)

# Create data loaders
batch_size = 128
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)

# Print dataset info
print(f"Training samples: {len(train_dataset)}, Testing samples:
{len(test_dataset)}")
```

```
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/train-images-idx3-ubyte.gz to ./data\FashionMNIST\raw\
train-images-idx3-ubyte.gz

100%|██████████| 26.4M/26.4M [14:25<00:00, 30.5kB/s]

Extracting ./data\FashionMNIST\raw\train-images-idx3-ubyte.gz to
./data\FashionMNIST\raw

Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/train-labels-idx1-ubyte.gz
```

```
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/train-labels-idx1-ubyte.gz to ./data\FashionMNIST\raw\
train-labels-idx1-ubyte.gz

100%|██████████| 29.5k/29.5k [00:00<00:00, 68.5kB/s]

Extracting ./data\FashionMNIST\raw\train-labels-idx1-ubyte.gz to
./data\FashionMNIST\raw

Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/t10k-images-idx3-ubyte.gz to ./data\FashionMNIST\raw\
t10k-images-idx3-ubyte.gz

100%|██████████| 4.42M/4.42M [01:40<00:00, 43.9kB/s]

Extracting ./data\FashionMNIST\raw\t10k-images-idx3-ubyte.gz to
./data\FashionMNIST\raw

Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ./data\FashionMNIST\raw\
t10k-labels-idx1-ubyte.gz

100%|██████████| 5.15k/5.15k [00:00<?, ?B/s]

Extracting ./data\FashionMNIST\raw\t10k-labels-idx1-ubyte.gz to
./data\FashionMNIST\raw

Training samples: 60000, Testing samples: 10000
```

```python
import matplotlib.pyplot as plt
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

# Define Transform (Convert to Tensor)
transform = transforms.Compose([transforms.ToTensor()])

# Load Fashion MNIST Dataset
dataset = datasets.FashionMNIST(root='./data', train=True,
transform=transform, download=True)
dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

# Get a Batch of Images
images, labels = next(iter(dataloader))

# Class Labels for Fashion MNIST
```

```
class_names = ['T-shirt', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot']

# Plot Images in a Grid
fig, axes = plt.subplots(4, 4, figsize=(6, 6))
for i, ax in enumerate(axes.flat):
    ax.imshow(images[i].squeeze(), cmap='gray')
    ax.set_title(class_names[labels[i].item()])
    ax.axis("off")

plt.suptitle("Fashion MNIST Samples", fontsize=14)
plt.show()
```




```
class VAE(nn.Module):
    def __init__(self, latent_dim=20):
        super(VAE, self).__init__()
```

```
        # Encoder
        self.encoder = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU()
        )

        # Latent space
        self.fc_mu = nn.Linear(256, latent_dim)  # Mean
        self.fc_logvar = nn.Linear(256, latent_dim)  # Log Variance

        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 28*28),
            nn.Tanh()
        )

    def encode(self, x):
        x = self.encoder(x)
        mu = self.fc_mu(x)
        logvar = self.fc_logvar(x)
        return mu, logvar

    def reparameterize(self, mu, logvar):
        """Reparameterization Trick"""
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        return self.decoder(z)

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        x_recon = self.decode(z)
        return x_recon, mu, logvar

def vae_loss(recon_x, x, mu, logvar):
    recon_loss = F.mse_loss(recon_x, x.view(-1, 28*28),
reduction='sum')
    kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return recon_loss + kl_loss
```

```python
# Initialize model, optimizer
latent_dim = 20
vae = VAE(latent_dim=latent_dim).to("cuda" if
torch.cuda.is_available() else "cpu")
optimizer = optim.Adam(vae.parameters(), lr=1e-3)

# Training loop
num_epochs = 200
device = "cuda" if torch.cuda.is_available() else "cpu"

vae.train()
train_losses = []
for epoch in range(num_epochs):
    running_loss = 0.0
    for images, _ in train_loader:
        images = images.to(device)
        optimizer.zero_grad()

        recon_images, mu, logvar = vae(images)
        loss = vae_loss(recon_images, images, mu, logvar)

        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    avg_loss = running_loss / len(train_loader.dataset)
    train_losses.append(avg_loss)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}")
```

```
Epoch [1/200], Loss: 112.4648
Epoch [2/200], Loss: 75.3325
Epoch [3/200], Loss: 69.3400
Epoch [4/200], Loss: 66.6757
Epoch [5/200], Loss: 64.8368
Epoch [6/200], Loss: 63.7372
Epoch [7/200], Loss: 62.7573
Epoch [8/200], Loss: 62.0826
Epoch [9/200], Loss: 61.5363
Epoch [10/200], Loss: 61.0110
Epoch [11/200], Loss: 60.6343
Epoch [12/200], Loss: 60.2369
Epoch [13/200], Loss: 59.9359
Epoch [14/200], Loss: 59.6626
Epoch [15/200], Loss: 59.4359
Epoch [16/200], Loss: 59.2161
Epoch [17/200], Loss: 59.0207
Epoch [18/200], Loss: 58.7982
Epoch [19/200], Loss: 58.5954
Epoch [20/200], Loss: 58.4686
Epoch [21/200], Loss: 58.3565
```

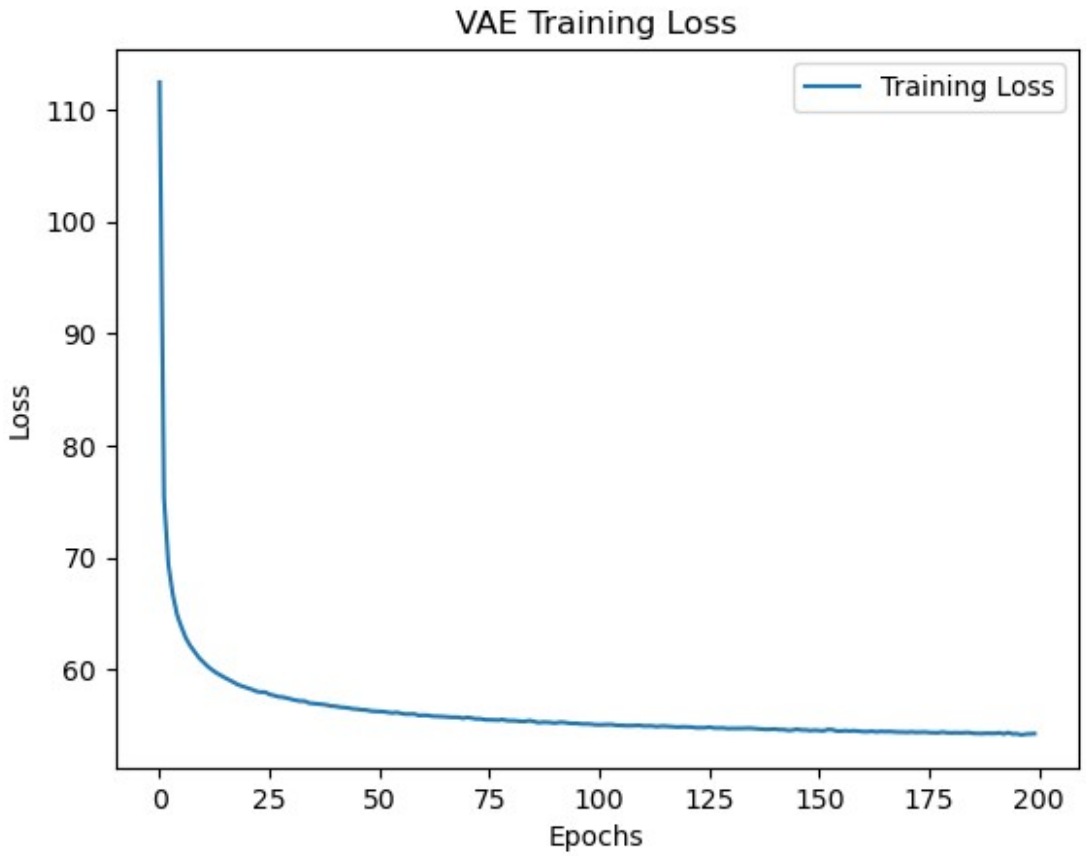
```
Epoch [22/200], Loss: 58.2008
Epoch [23/200], Loss: 58.0357
Epoch [24/200], Loss: 57.9640
Epoch [25/200], Loss: 57.9650
Epoch [26/200], Loss: 57.7632
Epoch [27/200], Loss: 57.6705
Epoch [28/200], Loss: 57.5454
Epoch [29/200], Loss: 57.5196
Epoch [30/200], Loss: 57.4211
Epoch [31/200], Loss: 57.3149
Epoch [32/200], Loss: 57.2305
Epoch [33/200], Loss: 57.1632
Epoch [34/200], Loss: 57.1590
Epoch [35/200], Loss: 56.9842
Epoch [36/200], Loss: 56.9493
Epoch [37/200], Loss: 56.9063
Epoch [38/200], Loss: 56.8688
Epoch [39/200], Loss: 56.8183
Epoch [40/200], Loss: 56.7141
Epoch [41/200], Loss: 56.7027
Epoch [42/200], Loss: 56.6173
Epoch [43/200], Loss: 56.5915
Epoch [44/200], Loss: 56.5417
Epoch [45/200], Loss: 56.4663
Epoch [46/200], Loss: 56.4419
Epoch [47/200], Loss: 56.3827
Epoch [48/200], Loss: 56.3604
Epoch [49/200], Loss: 56.2792
Epoch [50/200], Loss: 56.2494
Epoch [51/200], Loss: 56.2425
Epoch [52/200], Loss: 56.2195
Epoch [53/200], Loss: 56.1470
Epoch [54/200], Loss: 56.1049
Epoch [55/200], Loss: 56.1498
Epoch [56/200], Loss: 56.0527
Epoch [57/200], Loss: 55.9923
Epoch [58/200], Loss: 56.0048
Epoch [59/200], Loss: 55.9978
Epoch [60/200], Loss: 55.8768
Epoch [61/200], Loss: 55.8975
Epoch [62/200], Loss: 55.8832
Epoch [63/200], Loss: 55.8115
Epoch [64/200], Loss: 55.7957
Epoch [65/200], Loss: 55.7960
Epoch [66/200], Loss: 55.7658
Epoch [67/200], Loss: 55.7237
Epoch [68/200], Loss: 55.7177
Epoch [69/200], Loss: 55.7142
Epoch [70/200], Loss: 55.6241
```

```
Epoch [71/200], Loss: 55.7100
Epoch [72/200], Loss: 55.6351
Epoch [73/200], Loss: 55.5695
Epoch [74/200], Loss: 55.5795
Epoch [75/200], Loss: 55.4930
Epoch [76/200], Loss: 55.4965
Epoch [77/200], Loss: 55.4816
Epoch [78/200], Loss: 55.4570
Epoch [79/200], Loss: 55.5155
Epoch [80/200], Loss: 55.4187
Epoch [81/200], Loss: 55.4042
Epoch [82/200], Loss: 55.4024
Epoch [83/200], Loss: 55.3544
Epoch [84/200], Loss: 55.3454
Epoch [85/200], Loss: 55.4158
Epoch [86/200], Loss: 55.3631
Epoch [87/200], Loss: 55.2405
Epoch [88/200], Loss: 55.2685
Epoch [89/200], Loss: 55.2766
Epoch [90/200], Loss: 55.2365
Epoch [91/200], Loss: 55.2020
Epoch [92/200], Loss: 55.2725
Epoch [93/200], Loss: 55.2568
Epoch [94/200], Loss: 55.1919
Epoch [95/200], Loss: 55.1604
Epoch [96/200], Loss: 55.1679
Epoch [97/200], Loss: 55.1145
Epoch [98/200], Loss: 55.1243
Epoch [99/200], Loss: 55.1087
Epoch [100/200], Loss: 55.0576
Epoch [101/200], Loss: 55.0402
Epoch [102/200], Loss: 55.0613
Epoch [103/200], Loss: 55.0716
Epoch [104/200], Loss: 55.0635
Epoch [105/200], Loss: 54.9962
Epoch [106/200], Loss: 54.9946
Epoch [107/200], Loss: 54.9683
Epoch [108/200], Loss: 54.9626
Epoch [109/200], Loss: 54.9880
Epoch [110/200], Loss: 54.9957
Epoch [111/200], Loss: 54.9401
Epoch [112/200], Loss: 54.9478
Epoch [113/200], Loss: 54.9438
Epoch [114/200], Loss: 54.8781
Epoch [115/200], Loss: 54.9248
Epoch [116/200], Loss: 54.9178
Epoch [117/200], Loss: 54.8871
Epoch [118/200], Loss: 54.8453
Epoch [119/200], Loss: 54.8399
```

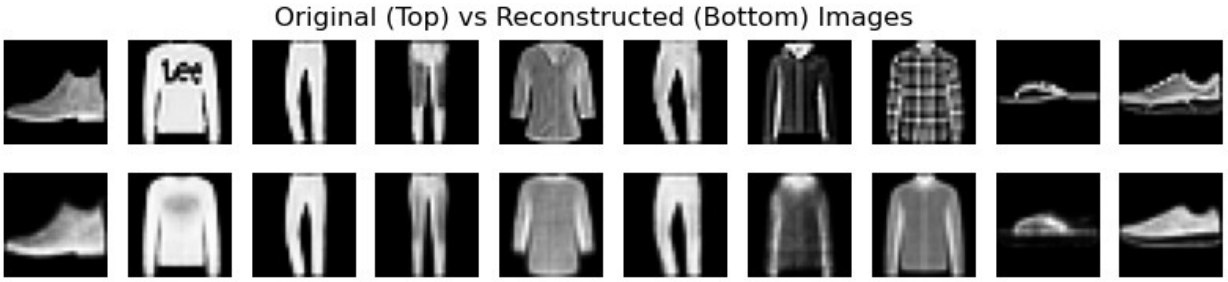
```
Epoch [120/200], Loss: 54.8672
Epoch [121/200], Loss: 54.8487
Epoch [122/200], Loss: 54.8371
Epoch [123/200], Loss: 54.7772
Epoch [124/200], Loss: 54.7777
Epoch [125/200], Loss: 54.7837
Epoch [126/200], Loss: 54.8361
Epoch [127/200], Loss: 54.7664
Epoch [128/200], Loss: 54.7530
Epoch [129/200], Loss: 54.7669
Epoch [130/200], Loss: 54.7136
Epoch [131/200], Loss: 54.7157
Epoch [132/200], Loss: 54.7367
Epoch [133/200], Loss: 54.7027
Epoch [134/200], Loss: 54.7252
Epoch [135/200], Loss: 54.7480
Epoch [136/200], Loss: 54.7140
Epoch [137/200], Loss: 54.6916
Epoch [138/200], Loss: 54.6583
Epoch [139/200], Loss: 54.6275
Epoch [140/200], Loss: 54.6505
Epoch [141/200], Loss: 54.6650
Epoch [142/200], Loss: 54.6069
Epoch [143/200], Loss: 54.6119
Epoch [144/200], Loss: 54.5418
Epoch [145/200], Loss: 54.5724
Epoch [146/200], Loss: 54.6511
Epoch [147/200], Loss: 54.5746
Epoch [148/200], Loss: 54.5789
Epoch [149/200], Loss: 54.5270
Epoch [150/200], Loss: 54.5620
Epoch [151/200], Loss: 54.5339
Epoch [152/200], Loss: 54.5013
Epoch [153/200], Loss: 54.6267
Epoch [154/200], Loss: 54.6074
Epoch [155/200], Loss: 54.4818
Epoch [156/200], Loss: 54.4858
Epoch [157/200], Loss: 54.5189
Epoch [158/200], Loss: 54.4841
Epoch [159/200], Loss: 54.5115
Epoch [160/200], Loss: 54.4714
Epoch [161/200], Loss: 54.4353
Epoch [162/200], Loss: 54.4283
Epoch [163/200], Loss: 54.4737
Epoch [164/200], Loss: 54.4070
Epoch [165/200], Loss: 54.4539
Epoch [166/200], Loss: 54.4315
Epoch [167/200], Loss: 54.4469
Epoch [168/200], Loss: 54.4228
```

```
Epoch [169/200], Loss: 54.3909
Epoch [170/200], Loss: 54.3956
Epoch [171/200], Loss: 54.3719
Epoch [172/200], Loss: 54.4046
Epoch [173/200], Loss: 54.3739
Epoch [174/200], Loss: 54.3729
Epoch [175/200], Loss: 54.3946
Epoch [176/200], Loss: 54.3537
Epoch [177/200], Loss: 54.3468
Epoch [178/200], Loss: 54.3245
Epoch [179/200], Loss: 54.3926
Epoch [180/200], Loss: 54.3438
Epoch [181/200], Loss: 54.3164
Epoch [182/200], Loss: 54.3425
Epoch [183/200], Loss: 54.3196
Epoch [184/200], Loss: 54.3409
Epoch [185/200], Loss: 54.3551
Epoch [186/200], Loss: 54.2818
Epoch [187/200], Loss: 54.2750
Epoch [188/200], Loss: 54.2564
Epoch [189/200], Loss: 54.2877
Epoch [190/200], Loss: 54.2887
Epoch [191/200], Loss: 54.2710
Epoch [192/200], Loss: 54.3295
Epoch [193/200], Loss: 54.2214
Epoch [194/200], Loss: 54.3538
Epoch [195/200], Loss: 54.2233
Epoch [196/200], Loss: 54.2367
Epoch [197/200], Loss: 54.1444
Epoch [198/200], Loss: 54.2099
Epoch [199/200], Loss: 54.2272
Epoch [200/200], Loss: 54.2530
```

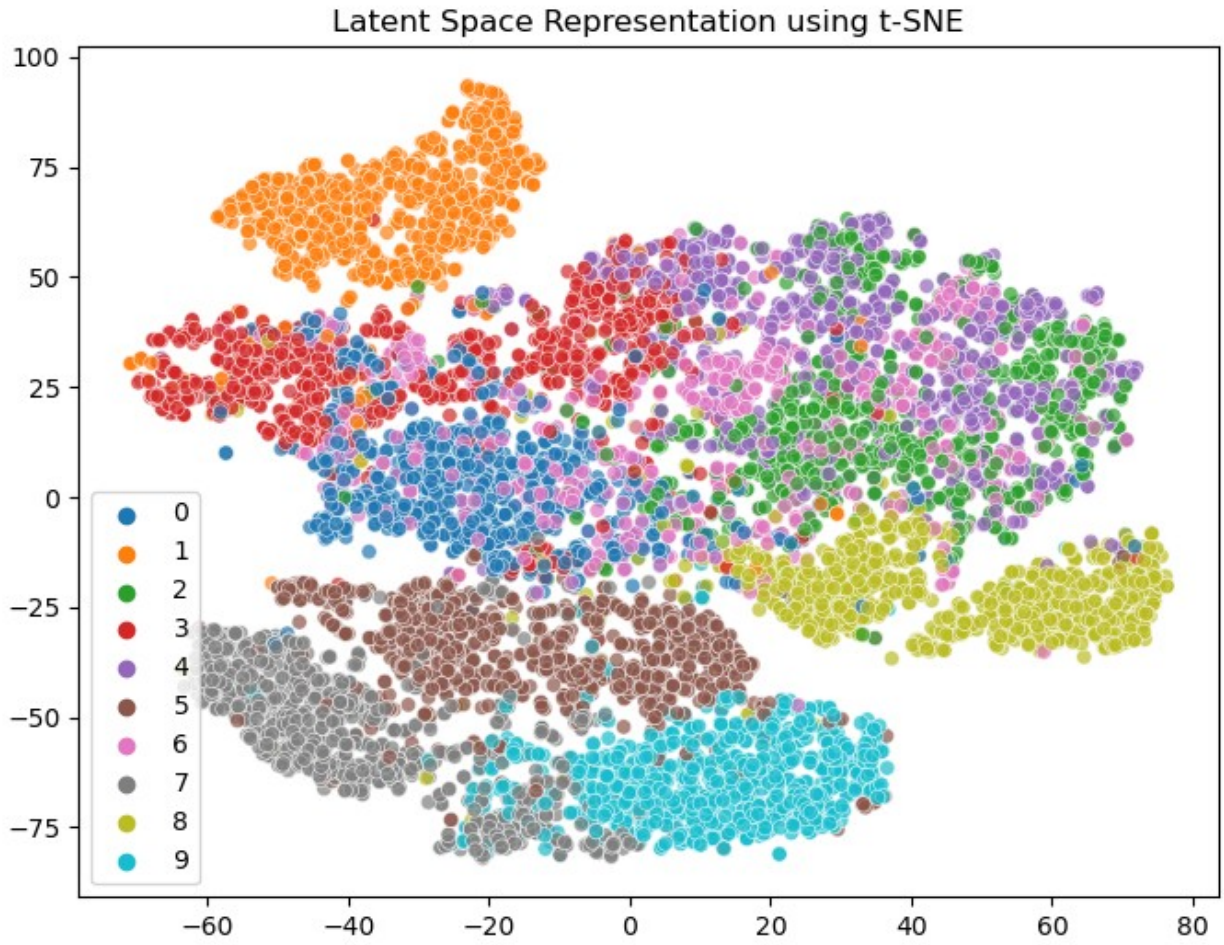
```
plt.plot(train_losses, label="Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("VAE Training Loss")
plt.legend()
plt.show()
```




```
vae.eval()
with torch.no_grad():
    images, _ = next(iter(test_loader))
    images = images.to(device)
    recon_images, _, _ = vae(images)

    fig, axes = plt.subplots(2, 10, figsize=(10, 2))
    for i in range(10):
        axes[0, i].imshow(images[i].cpu().numpy().squeeze(),
cmap="gray")
        axes[0, i].axis("off")

        axes[1, i].imshow(recon_images[i].cpu().numpy().reshape(28,
28), cmap="gray")
        axes[1, i].axis("off")

    plt.suptitle("Original (Top) vs Reconstructed (Bottom) Images")
    plt.show()
```




```
vae.eval()
all_mu = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        _, mu, _ = vae(images)
        all_mu.append(mu.cpu().numpy())
        all_labels.append(labels.numpy())

all_mu = np.concatenate(all_mu)
all_labels = np.concatenate(all_labels)

# Apply t-SNE for visualization
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
latent_2d = tsne.fit_transform(all_mu)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=latent_2d[:, 0], y=latent_2d[:, 1], hue=all_labels,
palette="tab10", alpha=0.7)
plt.title("Latent Space Representation using t-SNE")
plt.show()
```




```
vae.eval()
num_samples = 10
latent_samples = torch.randn(num_samples, latent_dim).to(device)

with torch.no_grad():
    generated_images = vae.decode(latent_samples).cpu()

fig, axes = plt.subplots(1, num_samples, figsize=(10, 2))
for i in range(num_samples):
    axes[i].imshow(generated_images[i].numpy().reshape(28, 28),
cmap="gray")
    axes[i].axis("off")

plt.suptitle("Generated Images from Latent Space")
plt.show()
```

Generated Images from Latent Space

```
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

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

# Load Fashion MNIST Test Dataset
test_dataset = torchvision.datasets.FashionMNIST(root="./data",
train=False, transform=transform, download=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

print("□ Test dataset loaded successfully!")

import torch

# Define device (use GPU if available, otherwise use CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"□ Using device: {device}")


□ Test dataset loaded successfully!
□ Using device: cpu
```

MODEL WASNT SAVED SO RUNNING AGAIN TO SAVE IT THIS TIME FOR FURTHER EVALUATION

```
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

# Load Fashion MNIST Dataset
transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])
train_dataset = torchvision.datasets.FashionMNIST(root="./data",
train=True, transform=transform, download=True)
```

```
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)

# Define Autoencoder (AE) Model
ae_model = Autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(ae_model.parameters(), lr=0.001)

# Train the Autoencoder
num_epochs = 10
for epoch in range(num_epochs):
    total_loss = 0
    for images, _ in train_loader:
        images = images.view(images.size(0), -1).to(device)  # Flatten
images
        optimizer.zero_grad()
        outputs = ae_model(images)
        loss = criterion(outputs, images)  # MSE Loss
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss /
len(train_loader):.4f}")

# Save the model for future use
torch.save(ae_model.state_dict(), "autoencoder.pth")
print("□ Autoencoder training completed and model saved!")
```

```
Epoch [1/10], Loss: 0.6520
Epoch [2/10], Loss: 0.6071
Epoch [3/10], Loss: 0.6031
Epoch [4/10], Loss: 0.6006
Epoch [5/10], Loss: 0.5986
Epoch [6/10], Loss: 0.5971
Epoch [7/10], Loss: 0.5959
Epoch [8/10], Loss: 0.5950
Epoch [9/10], Loss: 0.5942
Epoch [10/10], Loss: 0.5936
□ Autoencoder training completed and model saved!
```

```
# Load the trained autoencoder model
loaded_ae_model = Autoencoder().to(device)
loaded_ae_model.load_state_dict(torch.load("autoencoder.pth"))
loaded_ae_model.eval()  # Set to evaluation mode
print("□ Autoencoder model loaded successfully!")
```

```
□ Autoencoder model loaded successfully!
```

```
C:\Users\palak\AppData\Local\Temp\ipykernel_9844\1595940052.py:3:
FutureWarning: You are using `torch.load` with `weights_only=False`
(the current default value), which uses the default pickle module
```

```
implicitly. It is possible to construct malicious pickle data which
will execute arbitrary code during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  loaded_ae_model.load_state_dict(torch.load("autoencoder.pth"))

import matplotlib.pyplot as plt

# Get a batch of test images
test_dataset = torchvision.datasets.FashionMNIST(root="./data",
train=False, transform=transform, download=True)
test_loader = DataLoader(test_dataset, batch_size=10, shuffle=True)
test_images, _ = next(iter(test_loader))

# Flatten images before passing them into the AE model
test_images_flatten = test_images.view(test_images.size(0), -
1).to(device)

# Get the reconstructed images from the AE model
with torch.no_grad():
    ae_reconstructed =
loaded_ae_model(test_images_flatten).cpu().view(-1, 28, 28)

# Plot the original and reconstructed images side-by-side
fig, axes = plt.subplots(2, 10, figsize=(15, 4))

for i in range(10):
    # Original Image (Top Row)
    axes[0, i].imshow(test_images[i].squeeze(), cmap="gray")
    axes[0, i].axis("off")

    # AE Reconstructed Image (Bottom Row)
    axes[1, i].imshow(ae_reconstructed[i], cmap="gray")
    axes[1, i].axis("off")

axes[0, 0].set_title("Original Images", fontsize=10)
axes[1, 0].set_title("AE Reconstructed Images", fontsize=10)
plt.show()
```