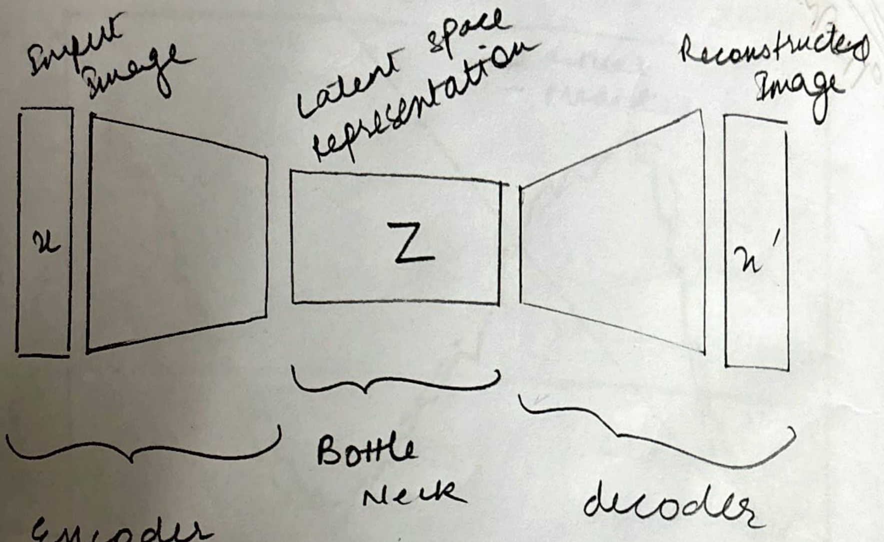
Autoencoder

Architecture.



Experiment 10
9/10/25

Compressing MNIST Dataset using autoencoders

Aim: The aim of this experiment is to compress the MNIST data set and plot the loss curve.

Objective:

→ Data compression
→ Reconstruction
→ Training the autoencoder
→ Performance evaluation
→ Visualization of latent space:

Pseudocode

→ Start
→ Import Required libraries
→ Load and preprocess MNIST Dataset
→ Define the Autoencoder model

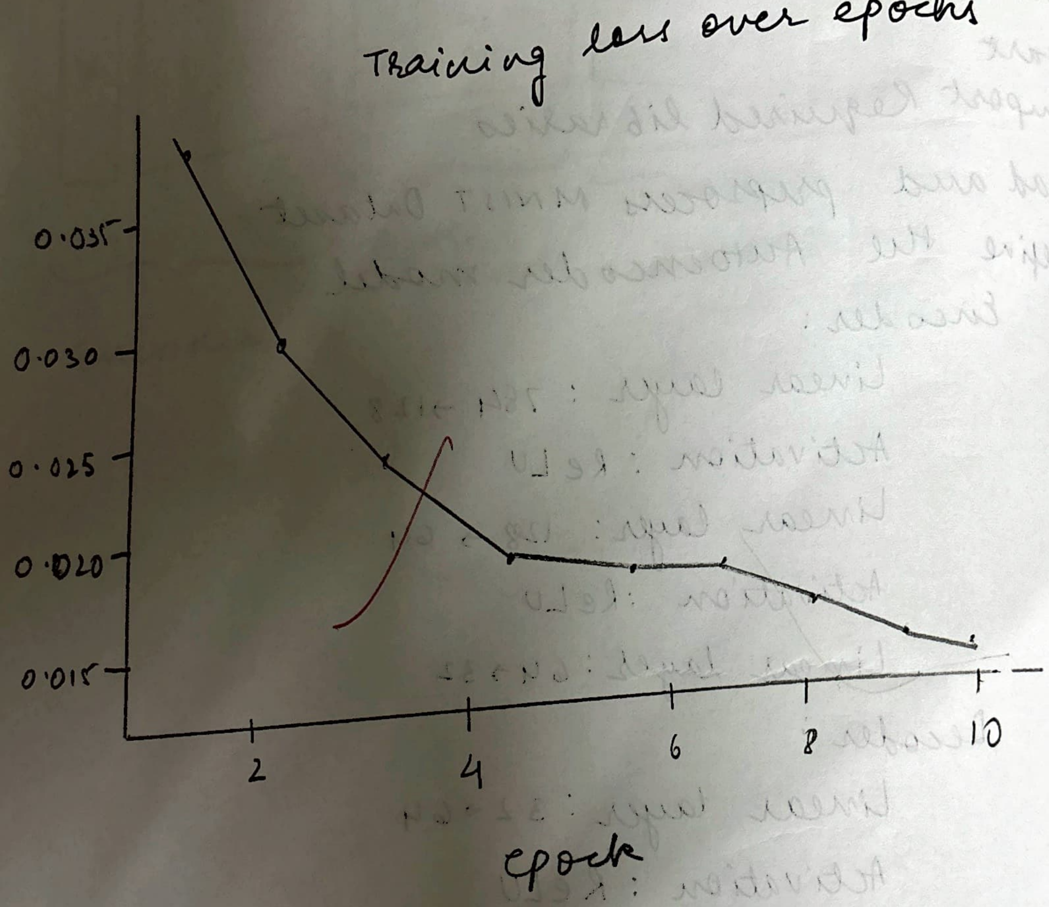
Encoder:
- Linear layer : 784 → 128
- Activation : ReLU
- Linear layer: 128 → 64
- Activation : ReLU
- Linear layer: 64 → 32

Decoder:
- Linear layer : 32 → 64
- Activation : ReLU
- Linear layer: 64 → 128
- Activation : ReLU
- Linear layer: 128 → 784
- Activation : Sigmoid.

→ Initialize Model, Loss Function & optimizer
→ Train the mode.

Epoch [1/10] : loss : 0.0378
epoch [2/10] : loss : 0.0290
epoch [3/10] : loss : 0.0229
⋮
epoch [9/10] : loss : 0.0137
epoch [10/10] : loss : 0.0133

The reconstructed images closely resembled the original images, especially for digits with simple shapes.



Visualize the loss curve

Result: successfully implemented and compressed the MNIST data set

```python
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

# ----------------------------
# 1. Load MNIST dataset
# ----------------------------
transform = transforms.Compose([
    transforms.ToTensor(),
])

train_data = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_data = datasets.MNIST(root='./data', train=False, transform=transform, download=True)

train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=False)

# ----------------------------
# 2. Define Autoencoder model
# ----------------------------
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        # Encoder: compress from 784 → 32
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True),
            nn.Linear(64, 32)
        )
```

7:14 PM ✓

```python
        return decoded

# ----------------------------
# 3. Train the model
# ----------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

num_epochs = 10
for epoch in range(num_epochs):
    for data, _ in train_loader:
        img = data.to(device)
        output = model(img)
        loss = criterion(output, img)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

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

# ----------------------------
# 4. Visualize results
# ----------------------------
model.eval()
with torch.no_grad():
    for data, _ in test_loader:
        img = data.to(device)
        output = model(img)
        break  # just one batch for visualization

# Compare original vs reconstructed images
img = img.cpu().numpy()
```

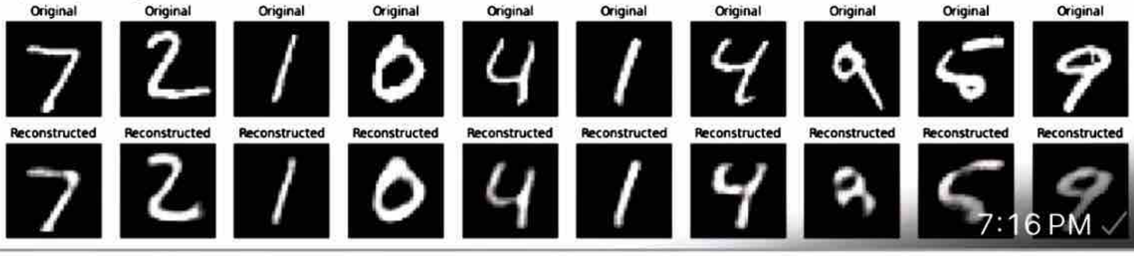
7:15 PM ✓

```python
# Compare original vs reconstructed images
img = img.cpu().numpy()
output = output.cpu().numpy()

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i+1)
    plt.imshow(img[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis("off")

    # Reconstructed
    ax = plt.subplot(2, n, i + n + 1)
    plt.imshow(output[i].reshape(28, 28), cmap='gray')
    plt.title("Reconstructed")
    plt.axis("off")
plt.show()
```

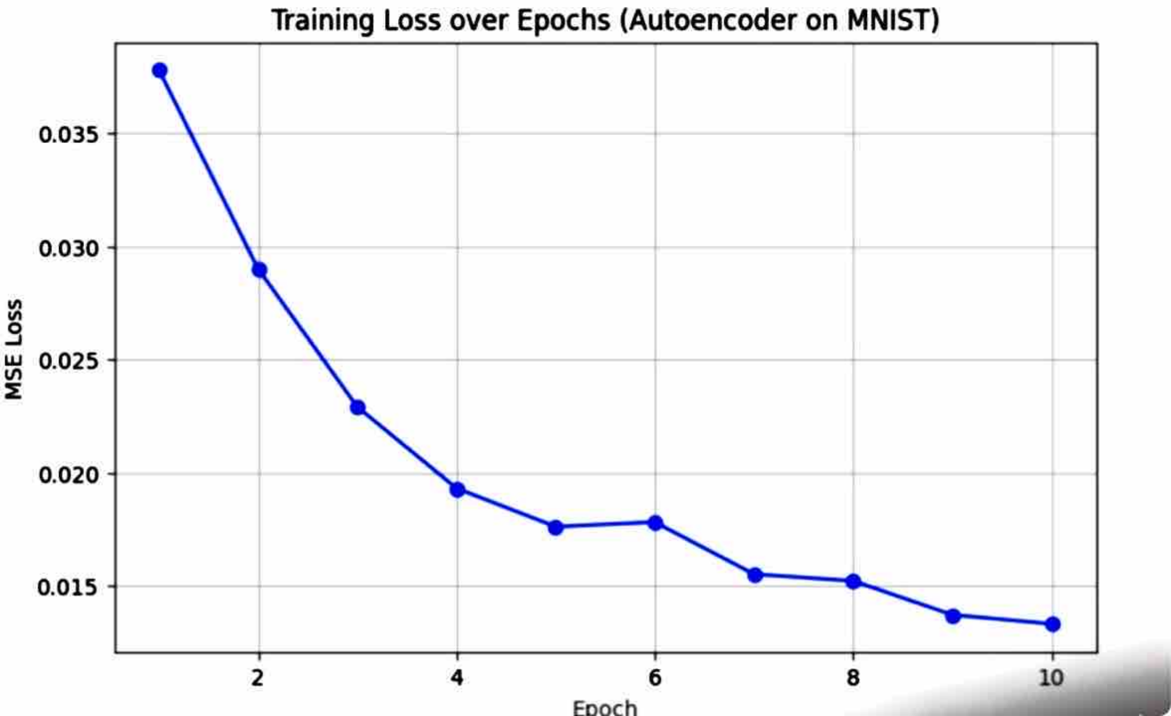
```
00%|██████████| 9.91M/9.91M [00:00<00:00, 38.9MB/s]
00%|██████████| 28.9k/28.9k [00:00<00:00, 1.15MB/s]
00%|██████████| 1.65M/1.65M [00:00<00:00, 9.93MB/s]
00%|██████████| 4.54k/4.54k [00:00<00:00, 9.70MB/s]
poch [1/10], Loss: 0.0378
poch [2/10], Loss: 0.0290
poch [3/10], Loss: 0.0229
poch [4/10], Loss: 0.0193
poch [5/10], Loss: 0.0176
poch [6/10], Loss: 0.0178
poch [7/10], Loss: 0.0155
poch [8/10], Loss: 0.0152
```

7:15 PM

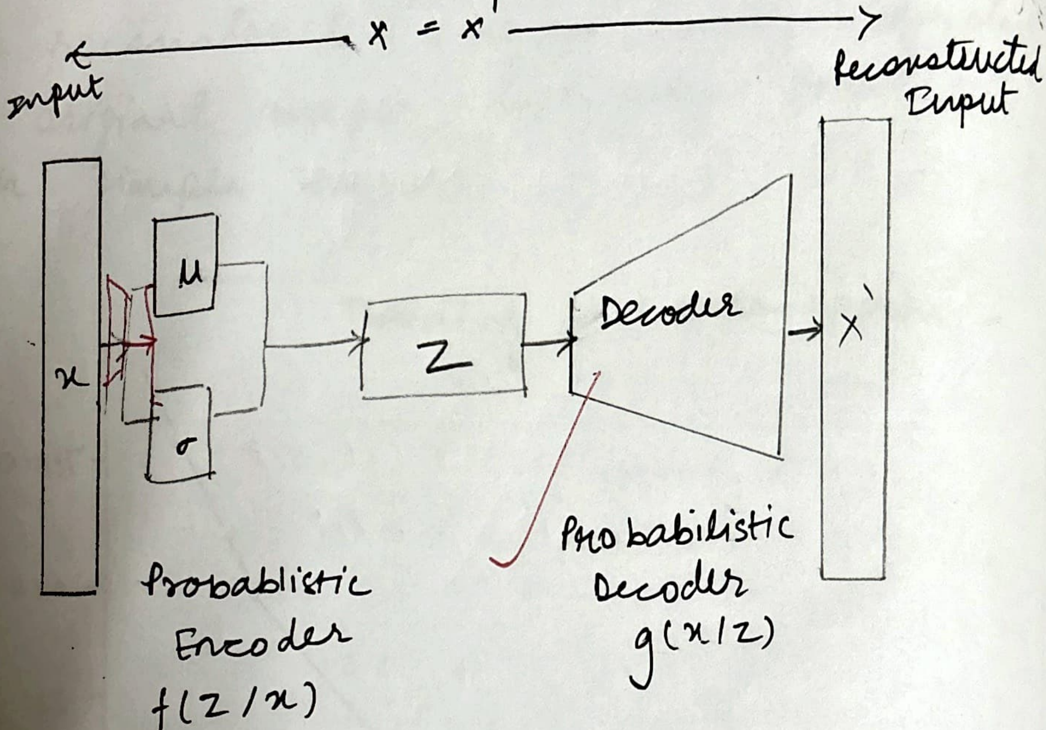
```
Epoch [3/10], Loss: 0.0229
Epoch [4/10], Loss: 0.0193
Epoch [5/10], Loss: 0.0176
Epoch [6/10], Loss: 0.0178
Epoch [7/10], Loss: 0.0155
Epoch [8/10], Loss: 0.0152
Epoch [9/10], Loss: 0.0137
Epoch [10/10], Loss: 0.0133
```



7:16 PM



VAE Architecture



Experiment 11 Variational
9/10/25 Auto Encoder

Aim : Implement VAE for the MNIST dataset

Objective:
→ Learn a low dimensional latent representation of handwritten digit images.

Pseudocode:
→ Setup parameters and device
→ Load MNIST dataset
→ Define VAE
Encoder : input image → hidden layer → output latent mean (mu)
Reparameterization : sample latent vector
Decoder : Latent vector z → hidden layer → reconstructed
→ Define loss function
Reconstruction loss
KL divergence.
→ Initialize model and optimizer.
→ For each epoch:
Set model to train mode
For each batch in training data:
Flatten images
Forward pass
- Compute Loss
Backpropagate and update model parameters.

Observation

epoch 1: Avg Loss 162.2229

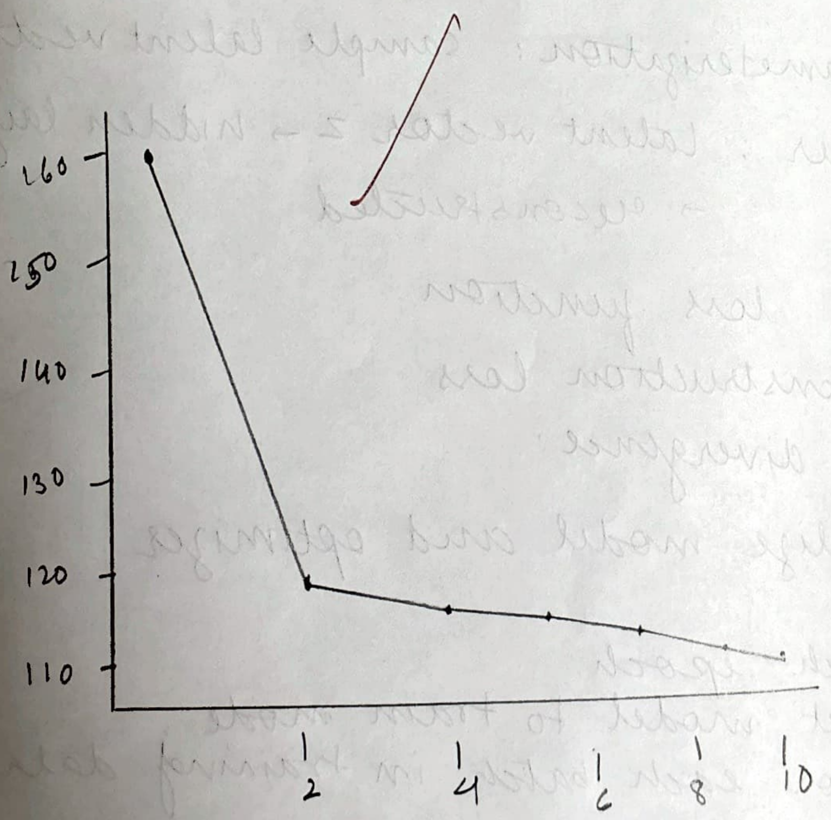
epoch 2: Avg Loss 124.9240

epoch 3: Avg Loss 119.5600

epoch 4: Avg Loss 116.8461

⋮

epoch 10 Avg Loss 111.1155.



→ After training :

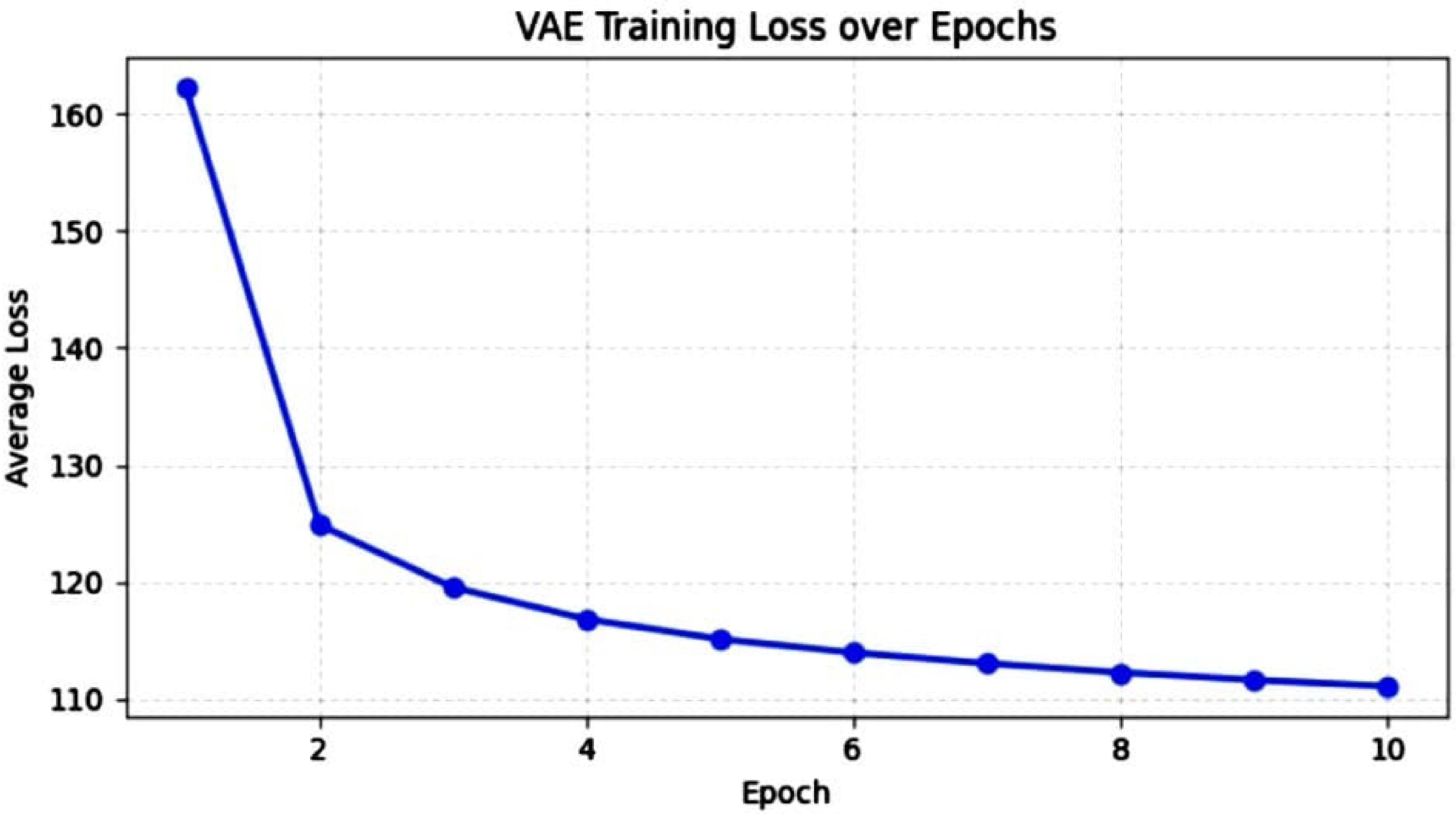
- Set model to eval mode
- Take a batch from test data
- Get reconstructed images
- Plot original & reconstructed images side by side

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

# --- Config ---
batch_size = 128
epochs = 10
lr = 1e-3
latent_dim = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# --- Data ---
transform = transforms.ToTensor()
train_loader = DataLoader(
    datasets.MNIST("data", train=True, download=True, transform=transform),
    batch_size=batch_size, shuffle=True
)
test_loader = DataLoader(
    datasets.MNIST("data", train=False, download=True, transform=transform),
    batch_size=batch_size, shuffle=False
)

# --- Model ---
class VAE(nn.Module):
    def __init__(self, z_dim):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 400)
        self.fc21 = nn.Linear(400, z_dim)   # μ
        self.fc22 = nn.Linear(400, z_dim)   # logσ²
        self.fc3 = nn.Linear(z_dim, 400)
        self.fc4 = nn.Linear(400, 28*28)
```

```python
        self.fc4 = nn.Linear(400, 28*28)

    def encode(self, x):
        h = F.relu(self.fc1(x))
        return self.fc21(h), self.fc22(h)

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

    def decode(self, z):
        h = F.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h))

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

def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x, reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

model = VAE(latent_dim).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=lr)

# --- Training ---
def train(epoch):
    model.train()
    train_loss = 0
    for data, _ in train_loader:
        data = data.to(device).view(-1, 784)
        optimizer.zero_grad()
```

```python
        loss = loss_function(recon, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
    print(f"Epoch {epoch}: Avg loss {train_loss / len(train_loader.dataset):.4f}")

# --- Run training ---
for epoch in range(1, epochs+1):
    train(epoch)

# --- Visualize ---
model.eval()
with torch.no_grad():
    data, _ = next(iter(test_loader))
    data = data.to(device).view(-1, 784)
    recon, _, _ = model(data)
    n = 8
    comparison = torch.cat([data[:n], recon[:n]])
    comparison = comparison.cpu().view(-1, 1, 28, 28)
    grid = torch.cat([comparison[:n], comparison[n:]])
    plt.figure(figsize=(8, 4))
    for i in range(n):
        plt.subplot(2, n, i+1)
        plt.imshow(data[i].cpu().view(28, 28), cmap="gray")
        plt.axis("off")
        plt.subplot(2, n, n+i+1)
        plt.imshow(recon[i].cpu().view(28, 28), cmap="gray")
        plt.axis("off")
    plt.show()
```

```
Epoch 1: Avg loss 162.2229
Epoch 2: Avg loss 124.9240
Epoch 3: Avg loss 119.5600
Epoch 4: Avg loss 116.8461
Epoch 5: Avg loss 115.1469
Epoch 6: Avg loss 113.9818
Epoch 7: Avg loss 113.0682
Epoch 8: Avg loss 112.2899
Epoch 9: Avg loss 111.6539
Epoch 10: Avg loss 111.1155
```


VAE Training Loss over Epochs
Average Loss
Epoch
160
150
140
130
120
110
2
4
6
8
10