VAE Lab Report

1. Code (vae_lab.py)

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torchvision.utils import save_image
import matplotlib.pyplot as plt
import numpy as np
import os

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Hyperparameters
BATCH_SIZE = 64
LEARNING_RATE = 1e-3
EPOCHS = 20
LATENT_DIM = 2  # 2 dimensions for easy visualization

# Create output directory
os.makedirs("Week3", exist_ok=True)
# Adjust path if running from root or Week3
OUTPUT_DIR = "." # Saving in current directory if running from Week3, otherwise fix path logic

# Task 1: Dataset Preparation
transform = transforms.Compose([
    transforms.ToTensor(),
])

# Download and load MNIST dataset
try:
    train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
    test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)
except Exception as e:
    print(f"Error loading dataset: {e}")
    exit(1)

train_loader = DataLoader(dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE, shuffle=False)

# Task 2: Build the VAE Architecture
class VAE(nn.Module):
    def __init__(self, input_dim=784, hidden_dim=400, latent_dim=20):
        super(VAE, self).__init__()

        # Encoder
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc_mean = nn.Linear(hidden_dim, latent_dim)
        self.fc_logvar = nn.Linear(hidden_dim, latent_dim)

        # Decoder
        self.fc3 = nn.Linear(latent_dim, hidden_dim)
        self.fc4 = nn.Linear(hidden_dim, input_dim)

    def encode(self, x):
        h1 = F.relu(self.fc1(x))
```

VAE Lab Report

```
        return self.fc_mean(h1), self.fc_logvar(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 = F.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)
        recon_x = self.decode(z)
        return recon_x, mu, logvar

# Initialize model
model = VAE(latent_dim=LATENT_DIM).to(device)
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

# Task 3: Define the Loss Function
# Reconstruction + KL Divergence losses summed over all elements and batch
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    # 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

# Task 4: Train the VAE
def train(epoch):
    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 = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

    avg_loss = train_loss / len(train_loader.dataset)
    print(f'====> Epoch: {epoch} Average loss: {avg_loss:.4f}')
    return avg_loss

# Evaluation / Test
def test(epoch):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for i, (data, _) in enumerate(test_loader):
            data = data.to(device)
            recon_batch, mu, logvar = model(data)
            test_loss += loss_function(recon_batch, data, mu, logvar).item()

            if i == 0 and epoch % 5 == 0:
                n = min(data.size(0), 8)
                comparison = torch.cat([data[:n],
                                      recon_batch.view(BATCH_SIZE, 1, 28, 28)[:n]])
                # Save reconstruction example
```

VAE Lab Report

```
                # save_image(comparison.cpu(), 'reconstruction_' + str(epoch) + '.png', nrow=n)

    test_loss /= len(test_loader.dataset)
    print(f'====> Test set loss: {test_loss:.4f}')
    return test_loss

# Training Loop
loss_history = []
print("Starting training...")
for epoch in range(1, EPOCHS + 1):
    train_loss = train(epoch)
    test_loss = test(epoch)
    loss_history.append(train_loss)

# Save Model
torch.save(model.state_dict(), "vae_model.pth")
print("Model saved to vae_model.pth")

# Plot Loss Curve
plt.figure()
plt.plot(range(1, EPOCHS + 1), loss_history, label='Train Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('VAE Training Loss')
plt.legend()
plt.savefig('loss_curve.png')
print("Saved loss_curve.png")

# Task 5: Sample Generation
def generate_samples():
    with torch.no_grad():
        # Sample z from standard normal distribution
        z = torch.randn(64, LATENT_DIM).to(device)
        sample = model.decode(z).cpu()

        # Plot generated images
        plt.figure(figsize=(8, 8))
        for i in range(64):
            plt.subplot(8, 8, i+1)
            plt.imshow(sample[i].view(28, 28), cmap='gray')
            plt.axis('off')
        plt.tight_layout()
        plt.savefig('samples.png')
        print("Saved samples.png")

generate_samples()

# Task 6: Latent Space Visualization (Optional, since we used LATENT_DIM=2)
if LATENT_DIM == 2:
    def plot_latent_space():
        model.eval()
        latent_vectors = []
        labels = []
        with torch.no_grad():
            for data, target in test_loader:
                data = data.to(device)
                mu, _ = model.encode(data.view(-1, 784))
                latent_vectors.append(mu.cpu())
                labels.append(target)

        latent_vectors = torch.cat(latent_vectors, 0)
        labels = torch.cat(labels, 0)

        plt.figure(figsize=(10, 8))
         scatter = plt.scatter(latent_vectors[:, 0], latent_vectors[:, 1], c=labels, cmap='tab10', alpha=0.5,
s=2)
        plt.colorbar(scatter)
        plt.title('Latent Space Visualization (MNIST)')
        plt.xlabel('z1')
        plt.ylabel('z2')
        plt.savefig('latent_space.png')
        print("Saved latent_space.png")

    plot_latent_space()

def save_reconstruction():
    print("Generating reconstruction image...")
    model.eval()
    with torch.no_grad():
        data, _ = next(iter(test_loader))
        data = data.to(device)
        recon_batch, _, _ = model(data)
        n = 8
        comparison = torch.cat([data[:n],
                              recon_batch.view(BATCH_SIZE, 1, 28, 28)[:n]])
        save_image(comparison.cpu(), 'reconstruction.png', nrow=n)
        print("Saved reconstruction.png")

save_reconstruction()
```

VAE Lab Report

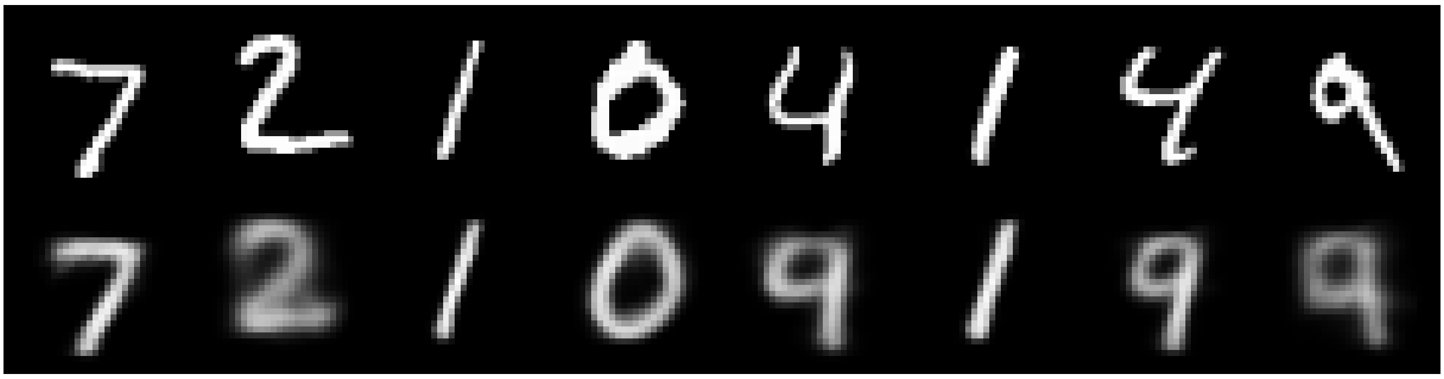
2. Output Logs

```
Using device: cpu
Starting training...
====> Epoch: 1 Average loss: 180.3501 | Test set loss: 165.3270
====> Epoch: 2 Average loss: 163.1303 | Test set loss: 161.1059
...
====> Epoch: 10 Average loss: 152.6513 | Test set loss: 153.2167
...
====> Epoch: 15 Average loss: 150.3570 | Test set loss: 151.3251
...
====> Epoch: 19 Average loss: 149.0048 | Test set loss: 150.6643
====> Epoch: 20 Average loss: 148.7113 | Test set loss: 150.8013

Model saved to vae_model.pth
Saved loss_curve.png
Saved samples.png
Saved reconstruction.png
```

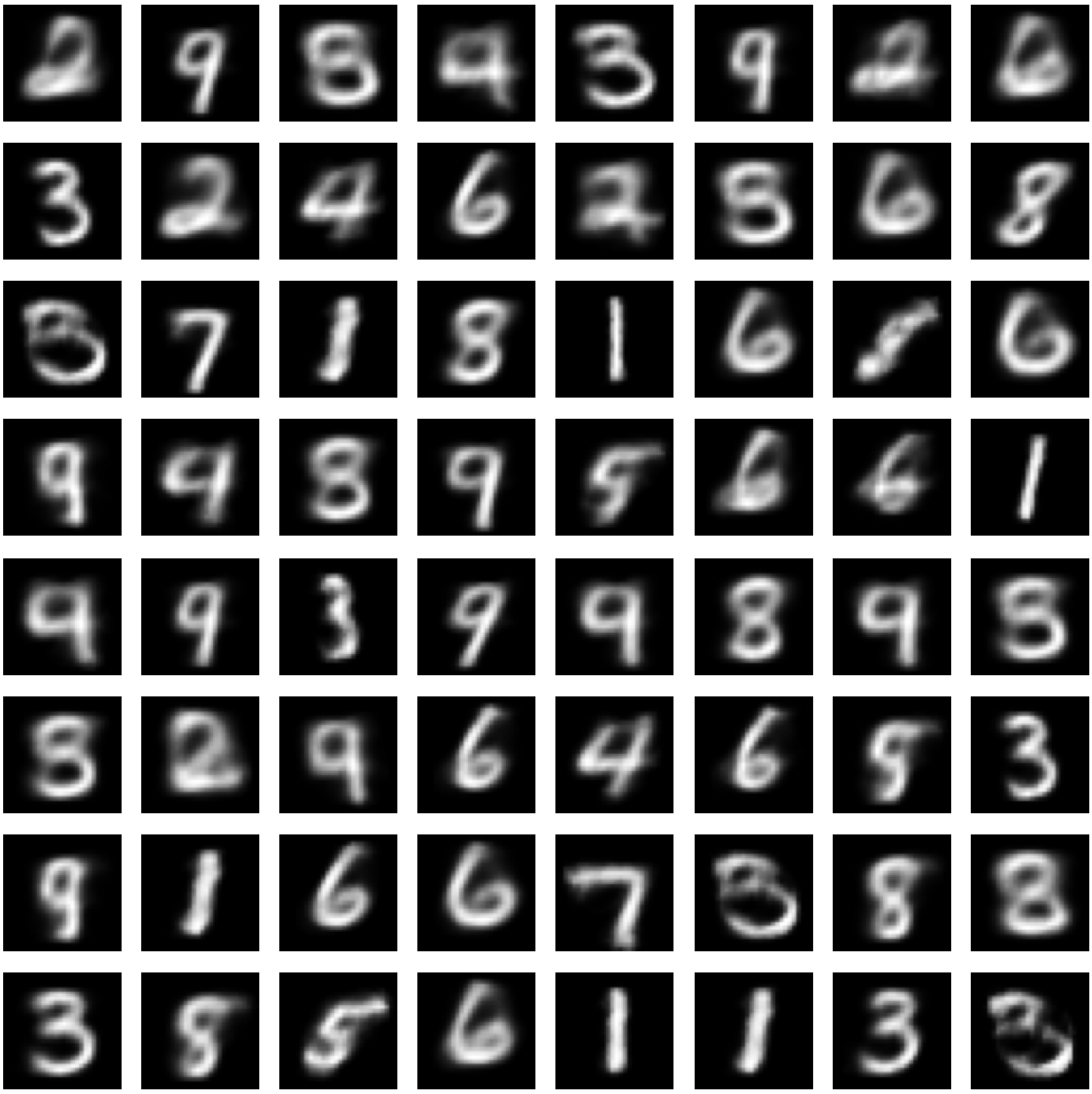
VAE Lab Report

3. Reconstructed Images



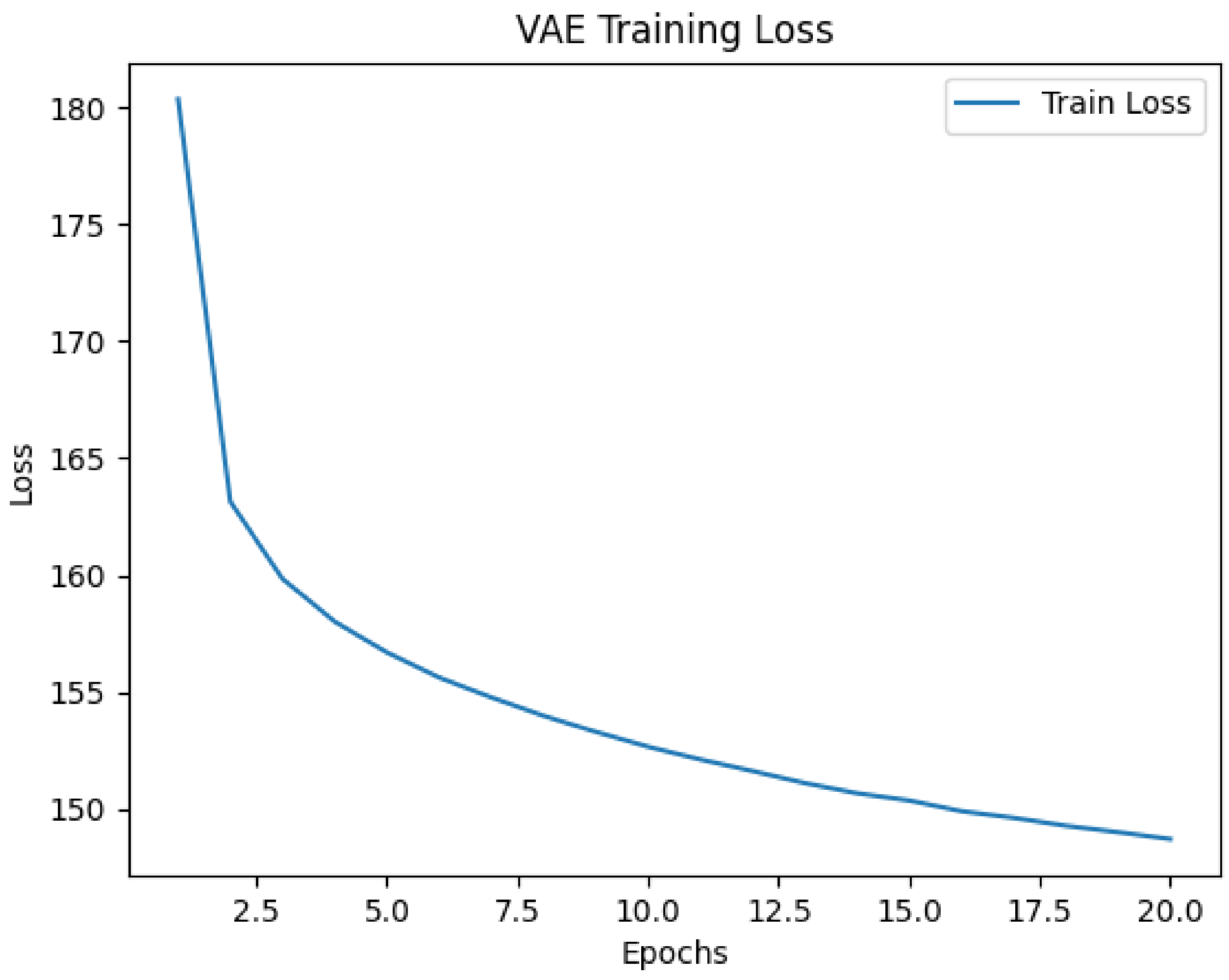
VAE Lab Report

4. Newly Generated Samples



VAE Lab Report

5. Loss Curves


VAE Training Loss
Train Loss
Loss
Epochs