LAB 4 > lab11.py > ...

```python
# ----------------------------------------
# LAB 11: VARIATIONAL AUTOENCODER (VAE)
# ----------------------------------------

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import matplotlib.pyplot as plt

# ----------------------------------------
# Step 1: Load and Preprocess Dataset
# ----------------------------------------
# FIX: Removed normalization to keep pixel values in [0,1]
transform = transforms.Compose([
    transforms.ToTensor()
])

train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)

# ----------------------------------------
# Step 2: Define the VAE Architecture
# ----------------------------------------
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        # Encoder
        self.fc1 = nn.Linear(28 * 28, 400)
        self.fc_mu = nn.Linear(400, 20)      # mean
        self.fc_logvar = nn.Linear(400, 20)  # log variance

        # Decoder
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 28 * 28)

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

LAB 4 > lab11.py > ...

```python
class VAE(nn.Module):
        self.fc4 = nn.Linear(400, 28 * 28)

    def encode(self, x):
        h1 = torch.relu(self.fc1(x))
        mu = self.fc_mu(h1)
        logvar = self.fc_logvar(h1)
        return mu, logvar

    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))  # outputs in [0,1]

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

# ---------------------------------------
# Step 3: Define Loss Function
# ---------------------------------------
def vae_loss_function(recon_x, x, mu, logvar):
    BCE = nn.functional.binary_cross_entropy(recon_x, x, reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

# ---------------------------------------
# Step 4: Initialize Model, Optimizer
# ---------------------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

model = VAE().to(device)
```

LAB 4 > lab11.py > ...

```python
# ----------------------------------------
# Step 4: Initialize Model, Optimizer
# ----------------------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

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

# ----------------------------------------
# Step 5: Train the VAE
# ----------------------------------------
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.view(-1, 28 * 28).to(device)
        recon_batch, mu, logvar = model(data)
        loss = vae_loss_function(recon_batch, data, mu, logvar)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    print(f'Epoch [{epoch+1}/{num_epochs}], Average loss: {train_loss / len(train_loader.dataset):.4f}')

print("Training complete!")

# ----------------------------------------
# Step 6: Visualize Reconstructed Images
# ----------------------------------------
model.eval()
with torch.no_grad():
    sample_data, _ = next(iter(train_loader))
    sample_data = sample_data.view(-1, 28 * 28).to(device)
    recon_data, _, _ = model(sample_data)

```

LAB 4 > lab11.py > ...

```python

# ----------------------------------------
# Step 6: Visualize Reconstructed Images
# ----------------------------------------
model.eval()
with torch.no_grad():
    sample_data, _ = next(iter(train_loader))
    sample_data = sample_data.view(-1, 28 * 28).to(device)
    recon_data, _, _ = model(sample_data)

sample_data = sample_data.cpu()
recon_data = recon_data.cpu()

fig, axes = plt.subplots(2, 8, figsize=(12, 3))
for i in range(8):
    axes[0, i].imshow(sample_data[i].view(28, 28), cmap='gray')
    axes[0, i].axis('off')
    axes[1, i].imshow(recon_data[i].view(28, 28), cmap='gray')
    axes[1, i].axis('off')
plt.suptitle("Top: Original Images | Bottom: Reconstructed by VAE", fontsize=10)
plt.show()

# ----------------------------------------
# Step 7: Generate New Samples
# ----------------------------------------
with torch.no_grad():
    z = torch.randn(16, 20).to(device)
    generated = model.decode(z).cpu()

fig, axes = plt.subplots(2, 8, figsize=(12, 3))
for i in range(16):
    axes[i // 8, i % 8].imshow(generated[i].view(28, 28), cmap='gray')
    axes[i // 8, i % 8].axis('off')
plt.suptitle("Generated Images from Random Latent Vectors", fontsize=10)
plt.show()

```

```
PS C:\Users\aarus\OneDrive\Desktop\SRM\DLT> & C:/Python313/python.exe "c:/Users/aarus/OneDrive/Desktop/SRM/DLT/LAB 4/lab11.py"
Using device: cpu
Epoch [1/10], Average loss: 164.6451
Epoch [2/10], Average loss: 121.3886
Epoch [3/10], Average loss: 114.5304
Epoch [4/10], Average loss: 111.6806
Epoch [5/10], Average loss: 110.0243
Epoch [6/10], Average loss: 108.8747
Epoch [7/10], Average loss: 108.0075
Epoch [8/10], Average loss: 107.3867
Epoch [9/10], Average loss: 106.9194
Epoch [10/10], Average loss: 106.4618
Training complete!
```