```
!ls
```

```
sample_data
```

```
from google.colab import drive
drive.mount('/content/drive')

import os

base_dir = "/content/drive/MyDrive/VAE_Project_lab3"
plots_dir = base_dir + "/plots"
samples_dir = base_dir + "/samples"
models_dir = base_dir + "/models"

os.makedirs(plots_dir, exist_ok=True)
os.makedirs(samples_dir, exist_ok=True)
os.makedirs(models_dir, exist_ok=True)

print("Folders created successfully!")
```

```
Mounted at /content/drive
Folders created successfully!
```

```
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
import numpy as np
```

```
batch_size = 128
learning_rate = 1e-3
epochs = 20
latent_dim = 2   # keep 2 for visualization
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
transform = transforms.Compose([
    transforms.ToTensor(),
])

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

print("Dataset loaded!")
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 12.9MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 342kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 3.24MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 3.11MB/s]Dataset loaded!
```

```
class Encoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 400)
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)

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

```
class Decoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(latent_dim, 400)
        self.fc2 = nn.Linear(400, 784)

    def forward(self, z):
        z = torch.relu(self.fc1(z))
        x_recon = torch.sigmoid(self.fc2(z))
        return x_recon
```

```
class VAE(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

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

    def forward(self, x):
        mu, logvar = self.encoder(x)
        z = self.reparameterize(mu, logvar)
        x_recon = self.decoder(z)
        return x_recon, mu, logvar
```

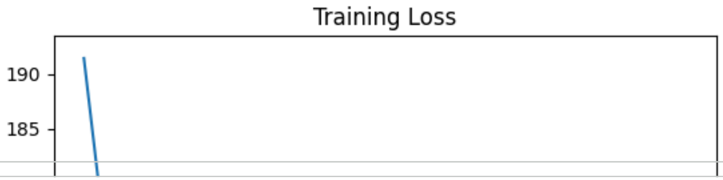
```
def loss_function(x_recon, x, mu, logvar):
    recon_loss = nn.functional.binary_cross_entropy(x_recon, x, reduction='sum')
    kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return recon_loss + kl_loss
```

```
model = VAE().to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

losses = []
```

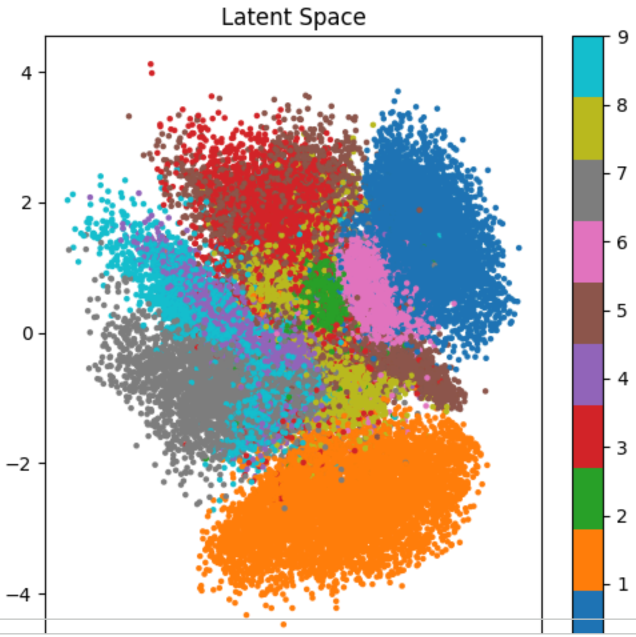
```
for epoch in range(epochs):
    total_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.view(-1, 784).to(device)

        optimizer.zero_grad()
        x_recon, mu, logvar = model(data)
        loss = loss_function(x_recon, data, mu, logvar)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

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

```
Epoch [1/20], Loss: 191.4277
Epoch [2/20], Loss: 166.8163
Epoch [3/20], Loss: 162.8067
Epoch [4/20], Loss: 160.5320
Epoch [5/20], Loss: 158.9362
Epoch [6/20], Loss: 157.6650
Epoch [7/20], Loss: 156.6727
Epoch [8/20], Loss: 155.8859
Epoch [9/20], Loss: 155.1744
Epoch [10/20], Loss: 154.5691
Epoch [11/20], Loss: 154.0429
Epoch [12/20], Loss: 153.4944
Epoch [13/20], Loss: 153.0785
Epoch [14/20], Loss: 152.6224
Epoch [15/20], Loss: 152.2083
Epoch [16/20], Loss: 151.8351
Epoch [17/20], Loss: 151.4760
Epoch [18/20], Loss: 151.1208
Epoch [19/20], Loss: 150.8457
Epoch [20/20], Loss: 150.5420
```

```python
plt.figure()
plt.plot(losses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss")

plot_path = plots_dir + "/training_loss.png"
plt.savefig(plot_path)
plt.show()

print("Saved:", plot_path)
```

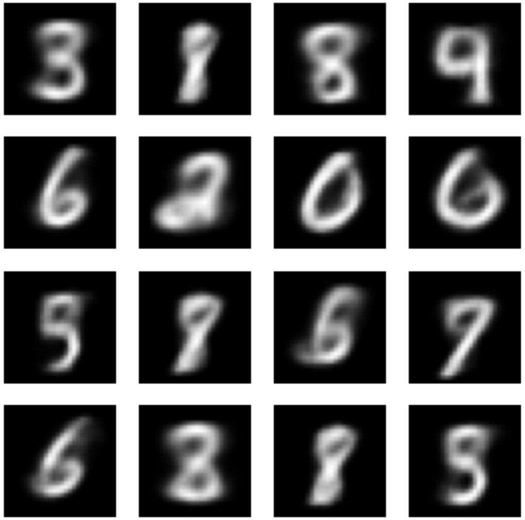


```
model.eval()
latent_vectors = []
labels = []

with torch.no_grad():
    for data, target in train_loader:
        data = data.view(-1, 784).to(device)
        mu, logvar = model.encoder(data)
        latent_vectors.append(mu.cpu())
        labels.append(target)

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

plt.figure(figsize=(6,6))
plt.scatter(latent_vectors[:,0], latent_vectors[:,1], c=labels, cmap='tab10', s=5)
plt.colorbar()
plt.title("Latent Space")

latent_plot_path = plots_dir + "/latent_space.png"
plt.savefig(latent_plot_path)
plt.show()

print("Saved:", latent_plot_path)
```



```
with torch.no_grad():
    z = torch.randn(16, latent_dim).to(device)
    samples = model.decoder(z).cpu()

samples = samples.view(16, 1, 28, 28)

fig, axes = plt.subplots(4,4, figsize=(5,5))
for i, ax in enumerate(axes.flat):
    ax.imshow(samples[i][0], cmap="gray")
    ax.axis("off")

sample_path = samples_dir + "/generated_samples.png"
```

```
    plt.savefig(sample_path)
    plt.show()

    print("Saved:", sample_path)
```

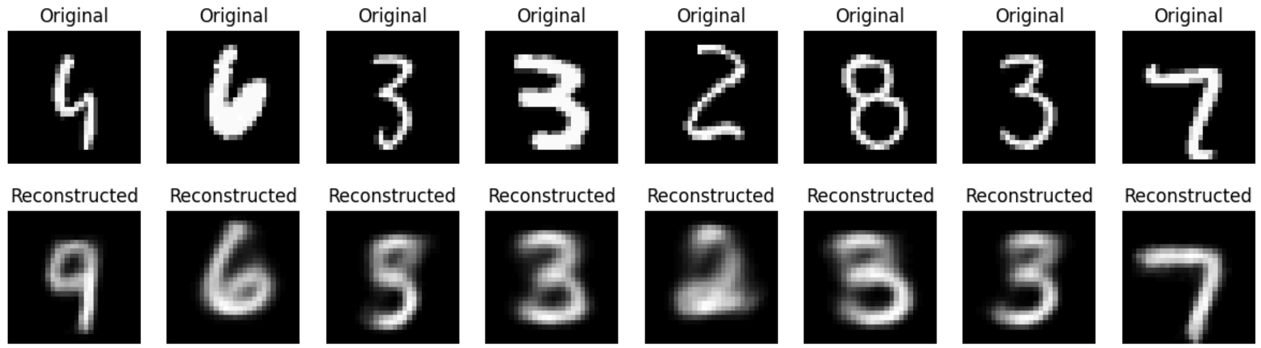


```
Saved: /content/drive/MyDrive/VAE_Project_lab3/samples/generated_samples.png
```

```
model_path = models_dir + "/vae_model.pth"
torch.save(model.state_dict(), model_path)
print("Model saved at:", model_path)
```

```
Model saved at: /content/drive/MyDrive/VAE_Project_lab3/models/vae_model.pth
```

```
model.eval()

data_iter = iter(train_loader)
images, _ = next(data_iter)

images = images.view(-1, 784).to(device)

with torch.no_grad():
    recon_images, mu, logvar = model(images)

images = images.cpu().view(-1,1,28,28)
recon_images = recon_images.cpu().view(-1,1,28,28)

n = 8  # number of samples to display

fig, axes = plt.subplots(2, n, figsize=(15,4))

for i in range(n):
    axes[0,i].imshow(images[i][0], cmap="gray")
    axes[0,i].set_title("Original")
    axes[0,i].axis("off")

    axes[1,i].imshow(recon_images[i][0], cmap="gray")
    axes[1,i].set_title("Reconstructed")
    axes[1,i].axis("off")

save_path = samples_dir + "/original_vs_reconstruction.png"
plt.savefig(save_path)
plt.show()

print("Saved:", save_path)
```



```
Saved: /content/drive/MyDrive/VAE_Project_lab3/samples/original_vs_reconstruction.png
```

```
import imageio

model.eval()

frames = []

grid_x = np.linspace(-3, 3, 10)
grid_y = np.linspace(-3, 3, 10)

for xi in grid_x:
    for yi in grid_y:
        z = torch.tensor([[xi, yi]]).float().to(device)

        with torch.no_grad():
            sample = model.decoder(z).cpu().view(28,28)

        fig, ax = plt.subplots()
        ax.imshow(sample, cmap='gray')
        ax.axis('off')

        # Save frame temporarily
        frame_path = "/content/frame.png"
        plt.savefig(frame_path)
        plt.close()

        frames.append(imageio.imread(frame_path))

gif_path = samples_dir + "/latent_space_animation.gif"
imageio.mimsave(gif_path, frames, fps=5)

print("GIF saved at:", gif_path)
```

```
/tmp/ipython-input-2049433730.py:26: DeprecationWarning: Starting with ImageIO v3 the behavior of this function w
  frames.append(imageio.imread(frame_path))
GIF saved at: /content/drive/MyDrive/VAE_Project_lab3/samples/latent_space_animation.gif
```

Start coding or generate with AI.