VAE_Implementation

February 1, 2025

[ ]:
```
import os
import torch
import tarfile
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
from torch.optim.lr_scheduler import LinearLR
from sklearn.decomposition import PCA
```

[ ]:
```
# Dataset setup
data_path = '/home/student/102flowers.tgz'
unzipped_path = '/home/student/flowers64'
```

[ ]:
```
os.makedirs(unzipped_path, exist_ok=True)
# Extract the .tgz file
try:
    with tarfile.open(data_path, "r:gz") as tar:
        tar.extractall(path=unzipped_path)
    print(f"Files have been extracted to: {unzipped_path}")
except FileNotFoundError:
    print(f"The file {unzipped_path} does not exist.")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Files have been extracted to: /home/student/flowers64
```

[ ]:
```
# Transform and DataLoader
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
transform = transforms.Compose([
    transforms.Resize((96, 96)),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std)
])
batch_size = 128
```

```
dataset = datasets.ImageFolder(unzipped_path, transform=transform)
data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

[ ]:
```
class VAE(nn.Module):
    def __init__(self, latent_dim=128):  # Increased latent dim for more␣
↪expressive power
        super(VAE, self).__init__()

        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),  # 96 -> 48
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 48 -> 24

            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),  # 24 -> 12
            nn.ReLU(),

            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),  # 12 -> 6
            nn.ReLU(),

            nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),  # 6 -> 3
            nn.ReLU(),

            nn.Flatten()
        )

        # Adjusted input size to match encoder output
        self.fc_mu = nn.Linear(512 * 3 * 3, latent_dim)
        self.fc_var = nn.Linear(512 * 3 * 3, latent_dim)

        # Decoder
        self.fc_decode = nn.Linear(latent_dim, 512 * 3 * 3)
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1), ␣
↪# 3 -> 6
            nn.ReLU(),

            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1), ␣
↪# 6 -> 12
            nn.ReLU(),

            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),  #␣
↪12 -> 24
            nn.ReLU(),

            nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1),  #␣
↪24 -> 48
```

```python
            nn.ReLU(),

            nn.ConvTranspose2d(32, 3, kernel_size=4, stride=2, padding=1),  #␣
↪48 -> 96
        )

    def encode(self, x):
        h = self.encoder(x)
        mu = self.fc_mu(h)
        var = self.fc_var(h)
        return mu, var

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

    def decode(self, z):
        h = self.fc_decode(z).view(-1, 512, 3, 3)  # Adjusted for new encoder␣
↪output
        x = self.decoder(h)
        return x

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

[ ]:
```python
# Loss function
def vae_loss(reconstructed, original, mu, var):
    recon_loss = nn.functional.mse_loss(reconstructed, original,␣
↪reduction='sum') / reconstructed.size()[0]
    kld_loss = -0.5 * torch.sum(1 + var - mu.pow(2) - torch.exp(var))
    return recon_loss + kld_loss, recon_loss, kld_loss
```

[ ]:
```python
# Training setup
latent_dim = 128
epochs = 30
learning_rate = 0.001
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
vae = VAE(latent_dim=latent_dim).to(device)
optimizer = optim.AdamW(vae.parameters(), lr=learning_rate)
scheduler = LinearLR(optimizer, start_factor=1.0, end_factor=0.01,␣
↪total_iters=epochs)

loss_history = []
recon_loss_history = []
```

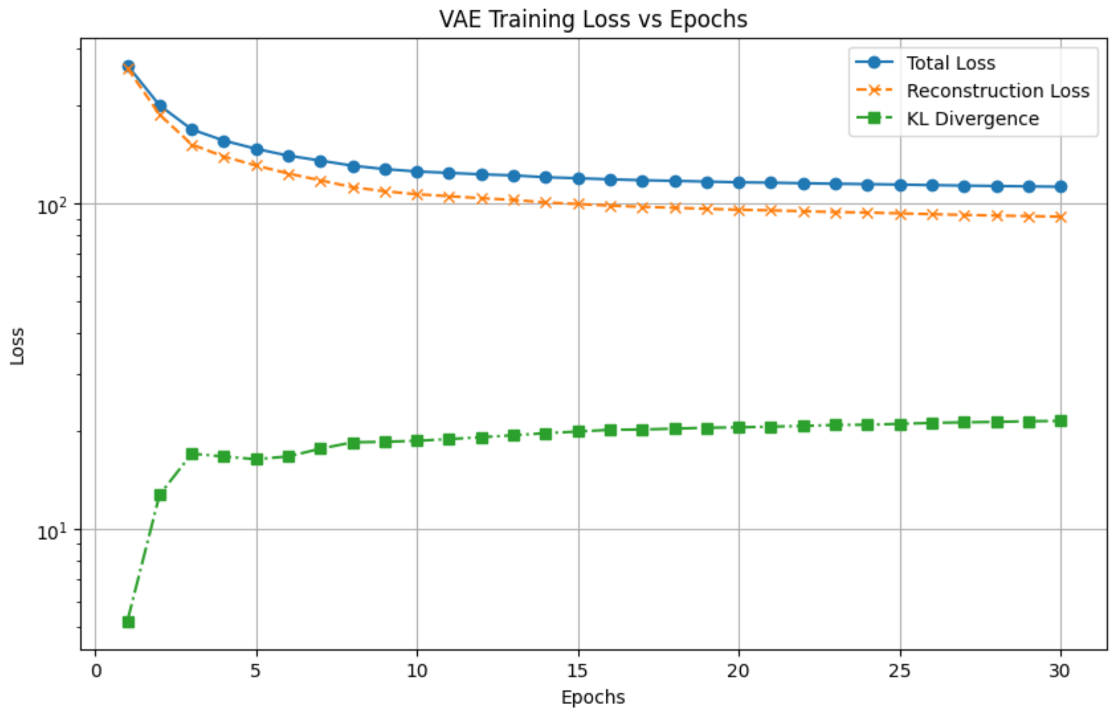
```
kld_loss_history = []

# Training loop
vae.train()
for epoch in range(epochs):
    train_loss = 0
    recon_loss, kld_loss = 0, 0
    for batch in data_loader:
        images, _ = batch
        images = images.to(device)

        optimizer.zero_grad()
        reconstructed, mu, var = vae(images)
        loss, recon, kld = vae_loss(reconstructed, images, mu, var)
        loss.backward()
        optimizer.step()
        kld_loss += kld.item()
        recon_loss += recon.item()
        train_loss += loss.item()

    scheduler.step()
    recon_loss_history.append(recon_loss / len(dataset))
    kld_loss_history.append(kld_loss / len(dataset))
    loss_history.append(train_loss / len(dataset))

    print(f"Epoch [{epoch + 1}/{epochs}], Total Loss: {train_loss /␣
↪len(dataset):.2f}, "
          f"Recon Loss: {recon_loss / len(dataset):.2f}, KL Loss: {kld_loss /␣
↪len(dataset):.2f}", flush=True)
```

```
Epoch [1/50], Total Loss: 266.35, Recon Loss: 261.14, KL Loss: 5.22
Epoch [2/50], Total Loss: 200.67, Recon Loss: 187.93, KL Loss: 12.75
Epoch [3/50], Total Loss: 169.17, Recon Loss: 152.11, KL Loss: 17.06
Epoch [4/50], Total Loss: 156.34, Recon Loss: 139.61, KL Loss: 16.72
Epoch [5/50], Total Loss: 147.48, Recon Loss: 131.08, KL Loss: 16.40
Epoch [6/50], Total Loss: 140.56, Recon Loss: 123.83, KL Loss: 16.73
Epoch [7/50], Total Loss: 135.70, Recon Loss: 118.03, KL Loss: 17.67
Epoch [8/50], Total Loss: 130.89, Recon Loss: 112.40, KL Loss: 18.49
Epoch [9/50], Total Loss: 127.79, Recon Loss: 109.23, KL Loss: 18.55
Epoch [10/50], Total Loss: 125.64, Recon Loss: 106.95, KL Loss: 18.69
Epoch [11/50], Total Loss: 124.40, Recon Loss: 105.48, KL Loss: 18.91
Epoch [12/50], Total Loss: 123.15, Recon Loss: 103.99, KL Loss: 19.17
Epoch [13/50], Total Loss: 122.20, Recon Loss: 102.76, KL Loss: 19.44
Epoch [14/50], Total Loss: 120.62, Recon Loss: 100.89, KL Loss: 19.73
Epoch [15/50], Total Loss: 119.80, Recon Loss: 99.85, KL Loss: 19.95
Epoch [16/50], Total Loss: 118.85, Recon Loss: 98.65, KL Loss: 20.20
Epoch [17/50], Total Loss: 118.11, Recon Loss: 97.87, KL Loss: 20.24
```

```
Epoch [18/50], Total Loss: 117.56, Recon Loss: 97.19, KL Loss: 20.37
Epoch [19/50], Total Loss: 117.00, Recon Loss: 96.51, KL Loss: 20.49
Epoch [20/50], Total Loss: 116.39, Recon Loss: 95.83, KL Loss: 20.56
Epoch [21/50], Total Loss: 116.09, Recon Loss: 95.43, KL Loss: 20.66
Epoch [22/50], Total Loss: 115.61, Recon Loss: 94.86, KL Loss: 20.75
Epoch [23/50], Total Loss: 115.21, Recon Loss: 94.30, KL Loss: 20.91
Epoch [24/50], Total Loss: 114.90, Recon Loss: 93.97, KL Loss: 20.93
Epoch [25/50], Total Loss: 114.51, Recon Loss: 93.44, KL Loss: 21.07
Epoch [26/50], Total Loss: 114.09, Recon Loss: 92.88, KL Loss: 21.21
Epoch [27/50], Total Loss: 113.71, Recon Loss: 92.41, KL Loss: 21.30
Epoch [28/50], Total Loss: 113.38, Recon Loss: 92.01, KL Loss: 21.36
Epoch [29/50], Total Loss: 113.09, Recon Loss: 91.63, KL Loss: 21.46
Epoch [30/50], Total Loss: 112.77, Recon Loss: 91.27, KL Loss: 21.51
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[8], line 28
     26 loss.backward()
     27 optimizer.step()
---> 28 kld_loss += kld.item()
     29 recon_loss += recon.item()
     30 train_loss += loss.item()

KeyboardInterrupt:
```

[ ]:
```
# Plot loss vs epochs
plt.figure(figsize=(10, 6))
plt.plot(range(1, epochs + 1), loss_history, marker='o', linestyle='-',
 ↪label='Total Loss')
plt.plot(range(1, epochs + 1), recon_loss_history, marker='x', linestyle='--',
 ↪label='Reconstruction Loss')
plt.plot(range(1, epochs + 1), kld_loss_history, marker='s', linestyle='-.',
 ↪label='KL Divergence')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.yscale('log')
plt.title('VAE Training Loss vs Epochs')
plt.legend()
plt.grid()
plt.show()
```




```
[ ]: def plot_results(vae, data_loader, mean=[0.485, 0.456, 0.406], std=[0.229, 0.
      ↪224, 0.225]):
        vae.eval()
        # Convert mean and std to tensors
        mean = torch.tensor(mean).view(1, 3, 1, 1).to(device)
        std = torch.tensor(std).view(1, 3, 1, 1).to(device)
        # Plot sample images
        with torch.no_grad():
            sample = torch.randn(8, latent_dim).to(device)
            generated_images = vae.decode(sample)

            # Denormalize images
            generated_images = generated_images * std + mean

            fig, axes = plt.subplots(2, 4, figsize=(10, 10))
            for i, ax in enumerate(axes.flatten()):
                img = generated_images[i].cpu().permute(1, 2, 0)
                ax.imshow(img)
                ax.axis("off")
            plt.suptitle("Rnadomly Samaple from Laten Space")
            plt.tight_layout()
            plt.show()

        # Show results for 5 images
```

```
vae.eval()
with torch.no_grad():
    images, _ = next(iter(data_loader))
    images = images[:5].to(device)
    reconstructed, _, _ = vae(images)

    # Visualization
    fig, axes = plt.subplots(5, 2, figsize=(10, 15))
    den_images = images * std + mean
    reconstructed = reconstructed * std + mean
    for i in range(5):
        axes[i, 0].imshow(den_images[i].cpu().permute(1, 2, 0))
        axes[i, 0].set_title("Original")
        axes[i, 0].axis("off")

        axes[i, 1].imshow(reconstructed[i].cpu().permute(1, 2, 0))
        axes[i, 1].set_title("Reconstructed")
        axes[i, 1].axis("off")
    plt.tight_layout()
    plt.show()

# Linear interpolation between 3 pairs
with torch.no_grad():
    pairs = [(0, 1), (1, 2), (2, 3)]  # Indices of pairs
    fig, axes = plt.subplots(len(pairs), 8, figsize=(20, 8))

    for idx, (i, j) in enumerate(pairs):
        z1, _ = vae.encode(images[i].unsqueeze(0))
        z2, _ = vae.encode(images[j].unsqueeze(0))

        # Decode and normalize the original images
        img1 = images[i] * std.squeeze(0) + mean.squeeze(0)
        img2 = images[j] * std.squeeze(0) + mean.squeeze(0)

        # Plot the original images at the first and last positions
        axes[idx, 0].imshow(img1.cpu().permute(1, 2, 0))
        axes[idx, 0].axis("off")
        axes[idx, 0].set_title("Original Image 1")

        axes[idx, -1].imshow(img2.cpu().permute(1, 2, 0))
        axes[idx, -1].axis("off")
        axes[idx, -1].set_title("Original Image 2")

        for alpha_idx, alpha in enumerate(torch.linspace(0, 1, steps=6)):
            z_interp = (1 - alpha) * z1 + alpha * z2
            img_interp = vae.decode(z_interp)
            img_interp = img_interp * std + mean
```

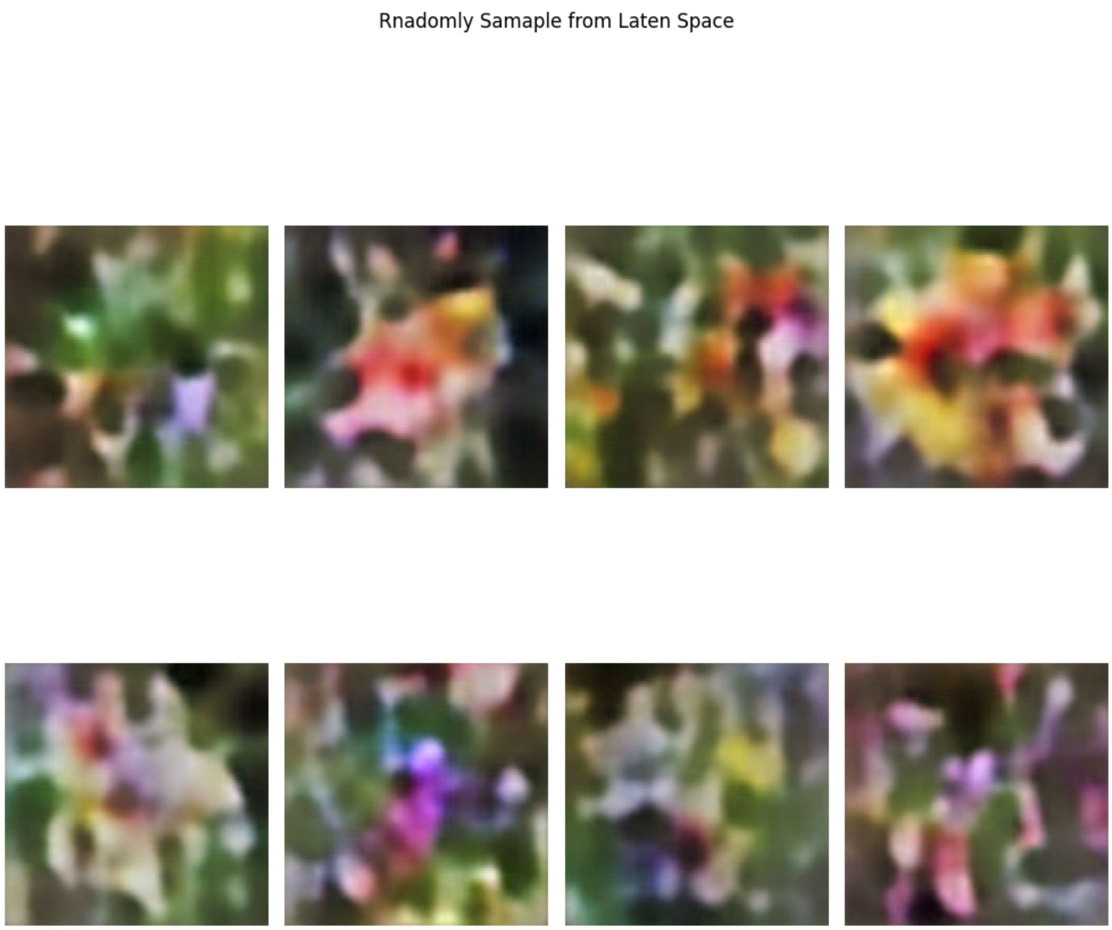
```
                img_interp = img_interp.squeeze(0)
                axes[idx, alpha_idx+1].imshow(img_interp.cpu().permute(1, 2, 0))
                axes[idx, alpha_idx+1].axis("off")

        plt.tight_layout()
        plt.suptitle("Move through Latent Space")
        plt.show()
```

[ ]:
```
def encode_and_visualize_pca(vae, dataloader, num_images=10,␣
↪num_samples_per_image=50, device="cuda"):
    """
    Encodes `num_images` images, samples multiple points from their␣
↪distributions,
    applies PCA for dimensionality reduction, and plots the latent space.

    Args:
        vae (nn.Module): Trained VAE model.
        dataloader (DataLoader): DataLoader to get images.
        num_images (int): Number of images to encode.
        num_samples_per_image (int): Number of samples per latent distribution.
        device (str): Device for computation ("cuda" or "cpu").
    """

    vae.eval()  # Set model to evaluation mode
    images, _ = next(iter(dataloader))  # Get a batch of images
    images = images[:num_images].to(device)  # Select the first `num_images`

    # Encode images to get latent distributions
    with torch.no_grad():
        mu, var = vae.encode(images)

    # Sample from each latent distribution
    sampled_points = []
    labels = []  # Track which image each sample belongs to
    for i in range(num_images):
        mu_i = mu[i]  # Mean for image i
        var_i = var[i]  # Variance for image i
        std_i = torch.exp(0.5 * var_i)  # Convert log-variance to std deviation

        # Sample multiple points from the latent distribution
        samples = mu_i + std_i * torch.randn(num_samples_per_image, mu.
↪shape[1]).to(device)
        sampled_points.append(samples.cpu())
        labels.extend([i] * num_samples_per_image)  # Assign a unique label per␣
↪image

    # Convert to tensor for PCA
```

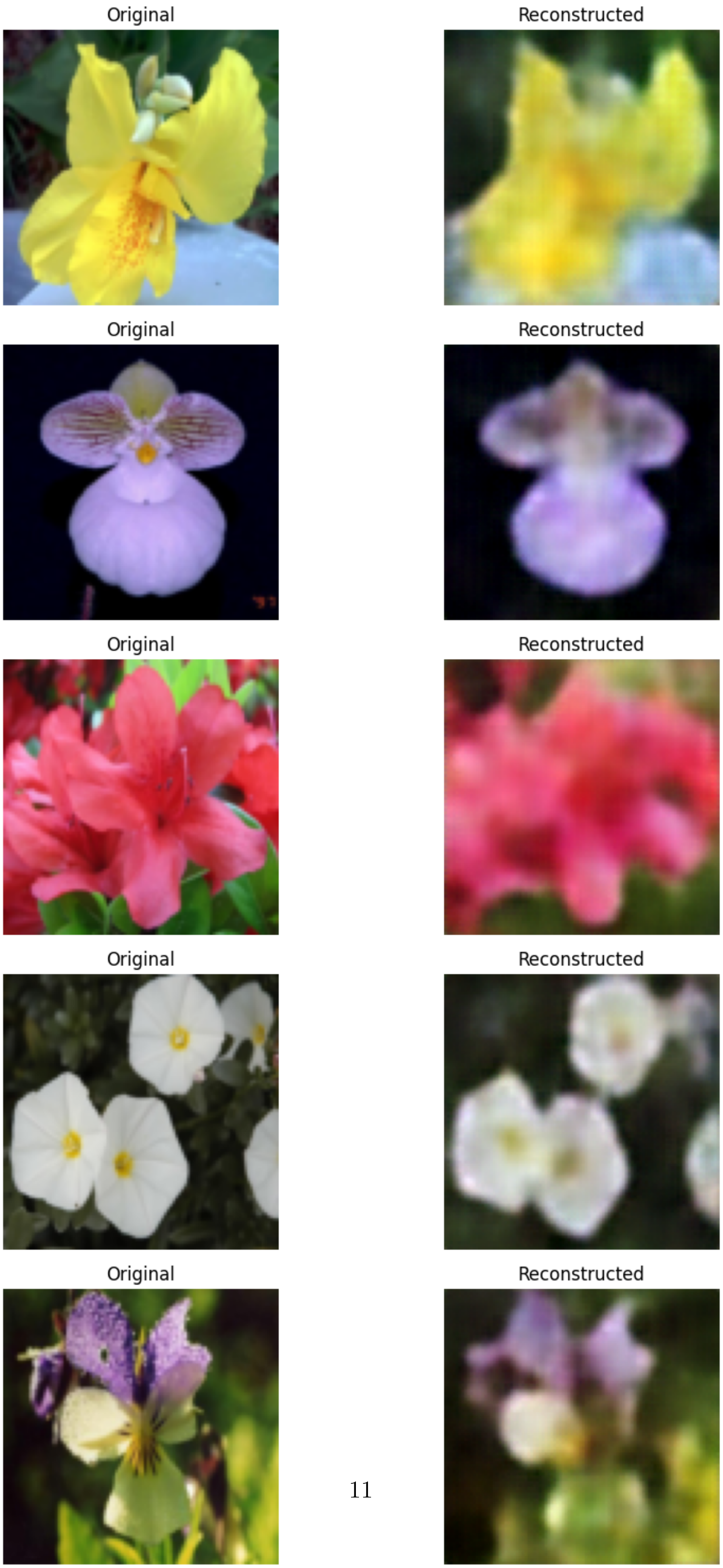
```
    sampled_points = torch.cat(sampled_points, dim=0).numpy()

    # Apply PCA to reduce dimensions to 2 for visualization
    pca = PCA(n_components=2)
    reduced_points = pca.fit_transform(sampled_points)

    # Plot the PCA-transformed latent space
    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(reduced_points[:, 0], reduced_points[:, 1], c=labels,␣
↪cmap="tab10", alpha=0.5, marker="o")
    plt.xlabel("PCA Dimension 1")
    plt.ylabel("PCA Dimension 2")
    plt.title("PCA Visualization of Sampled Latent Space")
    plt.grid()
    plt.show()
```

[ ]:
```
plot_results(vae, data_loader)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.016188323..0.9600401].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [0.029251188..1.0415324].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.0003222525..0.9045547].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [0.057617247..1.2267807].
```

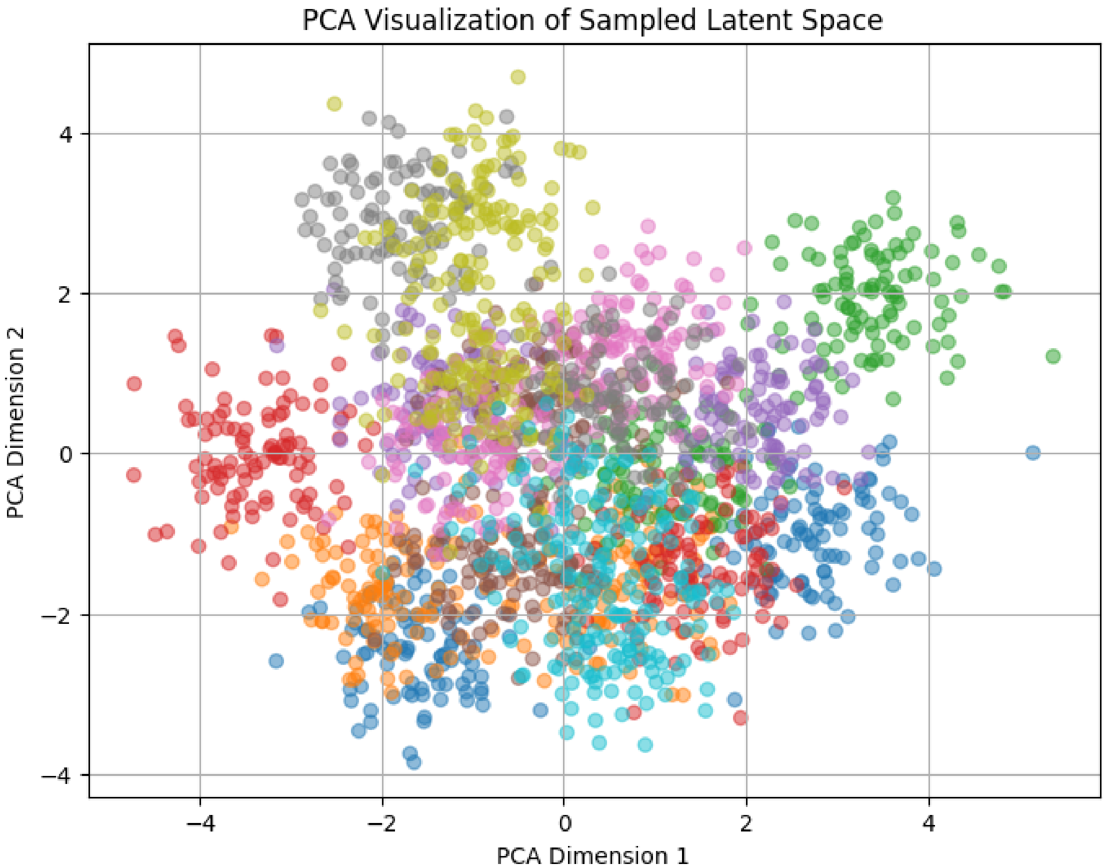



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [0.03396797..1.0535845].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.14875087..1.0503289].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [0.027055323..1.0375571].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.01017794..1.0056936].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.038351208..0.9730686].
```


Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [0.03186816..1.0529616].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.035125792..0.9318745].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.1503264..1.0436699].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.15032634..1.0436699].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.029792756..0.9056579].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [0.02486372..1.0342867].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [0.02486372..1.0342867].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.012179524..0.99380255].
```

[ ]:
```
encode_and_visualize_pca(vae, data_loader, num_images=20,
    num_samples_per_image=100, device="cuda")
```



[ ]:
```
# Save trained weights
torch.save(vae.state_dict(), '/home/student/model_weights.pkl')

print("Training complete. Model saved as vae_trained_weights.pkl.")
```

```
Training complete. Model saved as vae_trained_weights.pkl.
```