

```

1 # !pip install torch
2 # !pip install torchvision

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torchvision import datasets, transforms

1 class VAE(nn.Module):
2     def __init__(self, input_dim, hidden_dim, latent_dim):
3         super(VAE, self).__init__()
4
5         self.fc1 = nn.Linear(input_dim, hidden_dim)
6         self.fc_mu = nn.Linear(hidden_dim, latent_dim)
7         self.fc_logvar = nn.Linear(hidden_dim, latent_dim)
8
9         self.fc2 = nn.Linear(latent_dim, hidden_dim)
10        self.fc3 = nn.Linear(hidden_dim, input_dim)
11
12    def encode(self, x):
13        h1 = torch.relu(self.fc1(x))
14        return self.fc_mu(h1), self.fc_logvar(h1)
15
16    def reparameterize(self, mu, logvar):
17        std = torch.exp(0.5 * logvar)
18        eps = torch.randn_like(std)
19        return mu + eps * std
20
21    def decode(self, z):
22        h2 = torch.relu(self.fc2(z))
23        return torch.sigmoid(self.fc3(h2))
24
25    def forward(self, x):
26        mu, logvar = self.encode(x.view(-1, 784))
27        z = self.reparameterize(mu, logvar)
28        return self.decode(z), mu, logvar
29

1 def loss_function(recon_x, x, mu, logvar):
2     BCE = nn.functional.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
3     KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
4     return BCE + KLD
5

1 def train(model, train_loader, optimizer, epoch):
2     model.train()
3     train_loss = 0
4     for batch_idx, (data, _) in enumerate(train_loader):
5         optimizer.zero_grad()
6         recon_batch, mu, logvar = model(data)
7         loss = loss_function(recon_batch, data, mu, logvar)
8         loss.backward()
9         train_loss += loss.item()
10        optimizer.step()
11    print(f'Epoch {epoch}, Loss: {train_loss / len(train_loader.dataset)}')

1 transform = transforms.ToTensor()
2 train_dataset = datasets.MNIST('./data', train=True, download=True, transform=transform)
3 train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)
4
5 vae = VAE(input_dim=784, hidden_dim=400, latent_dim=20)
6 optimizer = optim.Adam(vae.parameters(), lr=1e-3)
7
8 for epoch in range(1, 11):
9     train(vae, train_loader, optimizer, epoch)
10

```

⚡ Downloading <http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz>
 Failed to download (trying next):
 HTTP Error 404: Not Found

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz>

Downloading <https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz> to ./data/MNIST/raw/train-images-idx3-u

```

100%|██████████| 9.91M/9.91M [00:00<00:00, 42.8MB/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-uby
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.25MB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-uby
100%|██████████| 1.65M/1.65M [00:00<00:00, 10.9MB/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-uby
100%|██████████| 4.54k/4.54k [00:00<00:00, 2.73MB/s]Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/ra

```

```

Epoch 1, Loss: 164.8485989420573
Epoch 2, Loss: 121.42219446614584
Epoch 3, Loss: 114.47182703450521
Epoch 4, Loss: 111.65288460286459
Epoch 5, Loss: 109.90579399414062
Epoch 6, Loss: 108.74398286132812
Epoch 7, Loss: 107.92627141927083
Epoch 8, Loss: 107.26012202148438
Epoch 9, Loss: 106.74345738932291
Epoch 10, Loss: 106.36903325195313

```

```

1 import matplotlib.pyplot as plt
2
3
4 def generate_images(model, num_images=10, latent_dim=20):
5     model.eval()
6     with torch.no_grad():
7         z = torch.randn(num_images, latent_dim)
8         generated_images = model.decode(z).cpu()
9
10    fig, axs = plt.subplots(1, num_images, figsize=(num_images, 1.5))
11    for i in range(num_images):
12        axs[i].imshow(generated_images[i].view(28, 28), cmap='gray')
13        axs[i].axis('off')
14    plt.show()
15
16
17

```

```
1 generate_images(vae, num_images=5, latent_dim=20)
```



Task 1: Modify the VAE architecture to use convolutional layers for both the encoder and decoder, and train it on the CIFAR-10 dataset. This modification will allow the model to capture spatial relationships within images more effectively, improving its ability to generate high-quality images. After training, compare the generated images with those from a fully connected VAE.

```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 import torchvision
5 import torchvision.transforms as transforms
6 import matplotlib.pyplot as plt

```

```

7
8
9 class ConvVAE(nn.Module):
10     def __init__(self, latent_dim=128):
11         super(ConvVAE, self).__init__()
12
13
14         self.encoder = nn.Sequential(
15             nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1),
16             nn.ReLU(),
17             nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),
18             nn.ReLU(),
19             nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
20             nn.ReLU(),
21         )
22         self.fc_mu = nn.Linear(128 * 4 * 4, latent_dim)
23         self.fc_logvar = nn.Linear(128 * 4 * 4, latent_dim)
24
25
26         self.fc_decode = nn.Linear(latent_dim, 128 * 4 * 4)
27         self.decoder = nn.Sequential(
28             nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
29             nn.ReLU(),
30             nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1),
31             nn.ReLU(),
32             nn.ConvTranspose2d(32, 3, kernel_size=4, stride=2, padding=1),
33             nn.Sigmoid()
34         )
35
36     def encode(self, x):
37         x = self.encoder(x)
38         x = x.view(x.size(0), -1)
39         return self.fc_mu(x), self.fc_logvar(x)
40
41     def reparameterize(self, mu, logvar):
42         std = torch.exp(0.5 * logvar)
43         eps = torch.randn_like(std)
44         return mu + eps * std
45
46     def decode(self, z):
47         x = self.fc_decode(z).view(-1, 128, 4, 4)
48         x = self.decoder(x)
49         return x
50
51     def forward(self, x):
52         mu, logvar = self.encode(x)
53         z = self.reparameterize(mu, logvar)
54         return self.decode(z), mu, logvar
55
56
57
58 def loss_function(recon_x, x, mu, logvar):
59     BCE = nn.functional.mse_loss(recon_x, x, reduction='sum')
60     KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
61     return BCE + KLD
62
63
64 transform = transforms.Compose([
65     transforms.ToTensor(),
66 ])
67 train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
68 train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)
69
70
71 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
72 vae = ConvVAE(latent_dim=128).to(device)
73 optimizer = optim.Adam(vae.parameters(), lr=1e-3)
74
75 def train(model, train_loader, optimizer, num_epochs=10):
76     model.train()
77     for epoch in range(num_epochs):
78         train_loss = 0
79         for batch_idx, (data, _) in enumerate(train_loader):
80             data = data.to(device)
81             optimizer.zero_grad()
82             recon_batch, mu, logvar = model(data)
83             loss = loss_function(recon_batch, data, mu, logvar)

```

```

84         loss.backward()
85         train_loss += loss.item()
86         optimizer.step()
87         print(f'Epoch {epoch+1}, Loss: {train_loss / len(train_loader.dataset)}')
88
89 train(vae, train_loader, optimizer, num_epochs=10)
90
91
92 def generate_images(model, num_images=5, latent_dim=128):
93     model.eval()
94     with torch.no_grad():
95         z = torch.randn(num_images, latent_dim).to(device)
96         generated_images = model.decode(z).cpu()
97     fig, axs = plt.subplots(1, num_images, figsize=(num_images, 2))
98     for i in range(num_images):
99         axs[i].imshow(generated_images[i].permute(1, 2, 0))
100        axs[i].axis('off')
101    plt.show()
102
103
104 generate_images(vae, num_images=5, latent_dim=128)

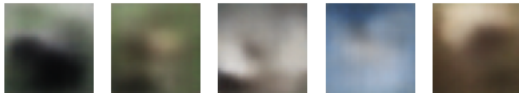
```

Files already downloaded and verified

```

Epoch 1, Loss: 117.75153705078125
Epoch 2, Loss: 84.09979545898437
Epoch 3, Loss: 79.74381274414063
Epoch 4, Loss: 78.43982765625
Epoch 5, Loss: 77.69290485351563
Epoch 6, Loss: 77.05875447265625
Epoch 7, Loss: 76.54311198242188
Epoch 8, Loss: 76.22927495117187
Epoch 9, Loss: 75.82949255859376
Epoch 10, Loss: 75.61358473632812

```



Task 2: Using the trained VAE, interpolate between two images in the latent space and generate intermediate images. This demonstrates how smoothly the model can transition between different data points. Visualize and display the results, showing the interpolated images in a grid format to observe the transformation.

```

1 def interpolate_images(model, img1, img2, num_steps=10):
2     model.eval()
3     with torch.no_grad():
4         mu1, _ = model.encode(img1.unsqueeze(0).to(device))
5         mu2, _ = model.encode(img2.unsqueeze(0).to(device))
6
7         interpolated_images = []
8
9         interpolation_factors = torch.linspace(0, 1, steps=num_steps)
10
11        for alpha in interpolation_factors:
12            z = (1 - alpha) * mu1 + alpha * mu2
13            recon = model.decode(z)
14            interpolated_images.append(recon.squeeze(0).cpu())
15
16    return interpolated_images
17
18
19
20 test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
21
22 img1, label1 = test_dataset[0]
23 img2, label2 = test_dataset[1]
24
25
26 fig, axs = plt.subplots(1, 2, figsize=(6, 3))
27 axs[0].imshow(img1.permute(1, 2, 0))
28 axs[0].set_title(f'Image 1 (Label: {label1})')
29 axs[0].axis('off')
30 axs[1].imshow(img2.permute(1, 2, 0))
31 axs[1].set_title(f'Image 2 (Label: {label2})')
32 axs[1].axis('off')
33 plt.suptitle("Original Images")

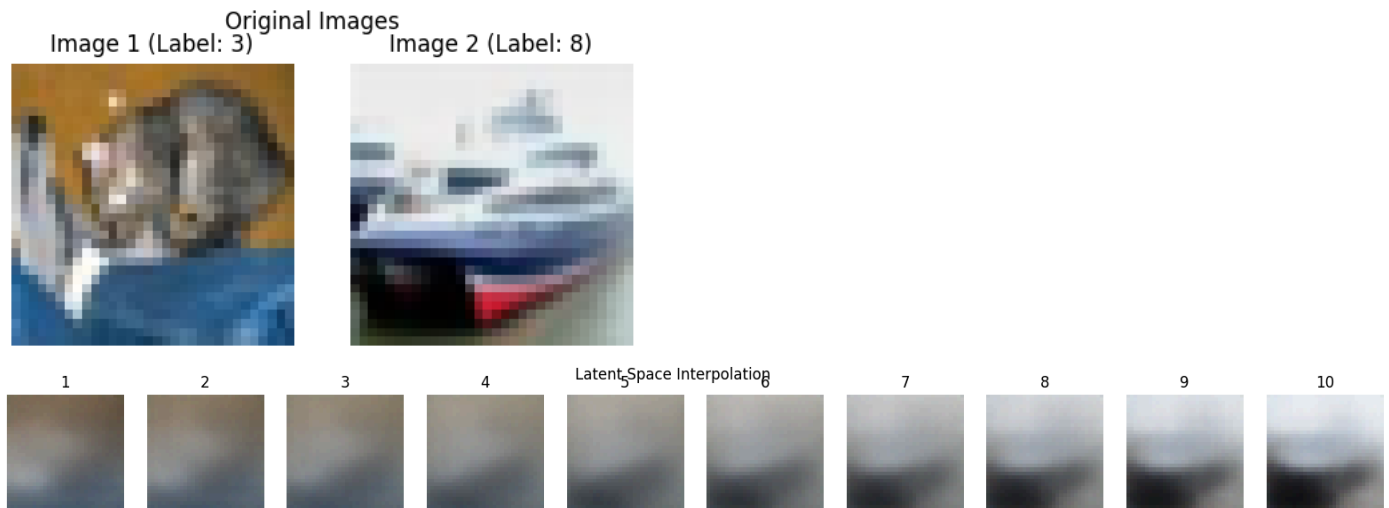
```

```

34 plt.show()
35
36
37 num_steps = 10
38 interpolated_images = interpolate_images(vae, img1, img2, num_steps=num_steps)
39
40
41 fig, axes = plt.subplots(1, num_steps, figsize=(num_steps * 2, 2))
42 for idx, img in enumerate(interpolated_images):
43     axes[idx].imshow(img.permute(1, 2, 0))
44     axes[idx].set_title(f'{idx+1}')
45     axes[idx].axis('off')
46 plt.suptitle("Latent Space Interpolation")
47 plt.show()

```

Files already downloaded and verified



Task 3: Train the VAE on a new dataset of your choice (e.g., CelebA for faces), and visualize generated samples. Experiment with sampling from different regions of the latent space and analyze how the generated outputs vary based on different latent vectors.

```

1 from google.colab import files
2 files.upload()
3
4 !mkdir -p ~/.kaggle
5 !cp kaggle.json ~/.kaggle/
6 !chmod 600 ~/.kaggle/kaggle.json
7
8
9 !kaggle datasets download -d therealcyberlord/50k-celeba-dataset-64x64
10 !unzip -q 50k-celeba-dataset-64x64.zip -d celeba_64
11
12
13 import os
14 import torch
15 import torch.nn as nn
16 import torch.optim as optim
17 import torchvision.transforms as transforms
18 import matplotlib.pyplot as plt
19 from torch.utils.data import Dataset, DataLoader
20 from PIL import Image
21
22
23 class CelebADataset(Dataset):
24     def __init__(self, root, transform=None):
25         self.root = root
26         self.transform = transform
27         self.image_files = []
28         for subdir, _, files in os.walk(self.root):
29             for file in files:
30                 if file.lower().endswith(('.jpg', '.jpeg', '.png')):
31                     self.image_files.append(os.path.join(subdir, file))

```

```

32     print(f"Found {len(self.image_files)} images in {self.root}")
33
34     def __len__(self):
35         return len(self.image_files)
36
37     def __getitem__(self, idx):
38         img_path = self.image_files[idx]
39         image = Image.open(img_path).convert('RGB')
40         if self.transform:
41             image = self.transform(image)
42         return image
43
44
45 transform = transforms.Compose([
46     transforms.Resize((64, 64)),
47     transforms.ToTensor(),
48 ])
49
50 dataset = CelebADataset(root='./celeba_64', transform=transform)
51 train_loader = DataLoader(dataset, batch_size=128, shuffle=True, num_workers=2)
52
53
54 class ConvVAE(nn.Module):
55     def __init__(self, latent_dim=128):
56         super(ConvVAE, self).__init__()
57         self.encoder = nn.Sequential(
58             nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1), nn.ReLU(),
59             nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1), nn.ReLU(),
60             nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1), nn.ReLU(),
61             nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1), nn.ReLU(),
62         )
63
64         self.fc_mu = nn.Linear(256 * 4 * 4, latent_dim)
65         self.fc_logvar = nn.Linear(256 * 4 * 4, latent_dim)
66         self.fc_decode = nn.Linear(latent_dim, 256 * 4 * 4)
67
68         self.decoder = nn.Sequential(
69             nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1), nn.ReLU(),
70             nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1), nn.ReLU(),
71             nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1), nn.ReLU(),
72             nn.ConvTranspose2d(32, 3, kernel_size=4, stride=2, padding=1), nn.Sigmoid(),
73         )
74
75     def encode(self, x):
76         x = self.encoder(x)
77         x = x.view(x.size(0), -1)
78         mu = self.fc_mu(x)
79         logvar = self.fc_logvar(x)
80         return mu, logvar
81
82     def reparameterize(self, mu, logvar):
83         std = torch.exp(0.5 * logvar)
84         eps = torch.randn_like(std)
85         return mu + eps * std
86
87     def decode(self, z):
88         x = self.fc_decode(z)
89         x = x.view(-1, 256, 4, 4)
90         x = self.decoder(x)
91         return x
92
93     def forward(self, x):
94         mu, logvar = self.encode(x)
95         z = self.reparameterize(mu, logvar)
96         out = self.decode(z)
97         return out, mu, logvar
98
99
100 def loss_function(recon_x, x, mu, logvar):
101     mse = nn.functional.mse_loss(recon_x, x, reduction='sum')
102     kld = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
103     return mse + kld
104
105
106 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
107 vae = ConvVAE(latent_dim=128).to(device)
108 optimizer = optim.Adam(vae.parameters(), lr=1e-3)

```

```

109
110 def train(model, dataloader, optimizer, num_epochs=10):
111     model.train()
112     for epoch in range(num_epochs):
113         total_loss = 0
114         for batch in dataloader:
115             batch = batch.to(device)
116             optimizer.zero_grad()
117             recon_batch, mu, logvar = model(batch)
118             loss = loss_function(recon_batch, batch, mu, logvar)
119             loss.backward()
120             total_loss += loss.item()
121             optimizer.step()
122     avg_loss = total_loss / len(dataloader.dataset)
123     print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}")
124
125
126 train(vae, train_loader, optimizer, num_epochs=10)
127
128
129 def generate_samples(model, num_samples=16):
130     model.eval()
131     with torch.no_grad():
132         z = torch.randn(num_samples, 128).to(device)
133         samples = model.decode(z).cpu()
134     grid_size = int(num_samples**0.5)
135     fig, axes = plt.subplots(grid_size, grid_size, figsize=(grid_size*2, grid_size*2))
136     for i in range(grid_size):
137         for j in range(grid_size):
138             idx = i * grid_size + j
139             axes[i, j].imshow(samples[idx].permute(1, 2, 0))
140             axes[i, j].axis('off')
141     plt.suptitle("Randomly Generated CelebA Faces")
142     plt.show()
143
144
145 generate_samples(vae, num_samples=16)
146
147
148 def visualize_latent_grid(model, grid_size=8):
149     model.eval()
150     with torch.no_grad():
151         z = torch.zeros(grid_size * grid_size, 128).to(device)
152         x_values = torch.linspace(-3, 3, grid_size)
153         y_values = torch.linspace(-3, 3, grid_size)
154         for i, x_val in enumerate(x_values):
155             for j, y_val in enumerate(y_values):
156                 index = i * grid_size + j
157                 z[index, 0] = x_val
158                 z[index, 1] = y_val
159             generated = model.decode(z).cpu()
160     fig, axes = plt.subplots(grid_size, grid_size, figsize=(grid_size*2, grid_size*2))
161     for i in range(grid_size):
162         for j in range(grid_size):
163             idx = i * grid_size + j
164             axes[i, j].imshow(generated[idx].permute(1, 2, 0))
165             axes[i, j].axis('off')
166     plt.suptitle("Latent Space Grid Traversal (Varying dims 0 and 1)")
167     plt.show()
168
169
170 visualize_latent_grid(vae, grid_size=8)
171
172
173 def sample_shifted_latents(model, offset=3, num_samples=16):
174     model.eval()
175     with torch.no_grad():
176         z = torch.randn(num_samples, 128).to(device) + offset
177         generated = model.decode(z).cpu()
178     grid_size = int(num_samples**0.5)
179     fig, axes = plt.subplots(grid_size, grid_size, figsize=(grid_size*2, grid_size*2))
180     for i in range(grid_size):
181         for j in range(grid_size):
182             idx = i * grid_size + j
183             axes[i, j].imshow(generated[idx].permute(1, 2, 0))
184             axes[i, j].axis('off')
185     plt.suptitle(f"Generated Samples from Latent Vectors Shifted by {offset}")

```

```
186     plt.show()  
187  
188 sample_shifted_latents(vae, offset=3, num_samples=16)
```




Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to

enable.

Saving kaggle.json to kaggle (2).json

Dataset URL: <https://www.kaggle.com/datasets/therealcyberlord/50k-celeba-dataset-64x64>

License(s): unknown

50k-celeba-dataset-64x64.zip: Skipping, found more recently modified local copy (use --force to force download)

replace celeba_64/50k/000001.jpg? [yles, [n]o, [A]ll, [N]one, [r]ename: A

Found 50000 images in ./celeba_64

Epoch [1/10], Loss: 401.2150

Epoch [2/10], Loss: 236.3383

Epoch [3/10], Loss: 212.6846

Epoch [4/10], Loss: 203.5287

Epoch [5/10], Loss: 198.2734

Epoch [6/10], Loss: 194.7579

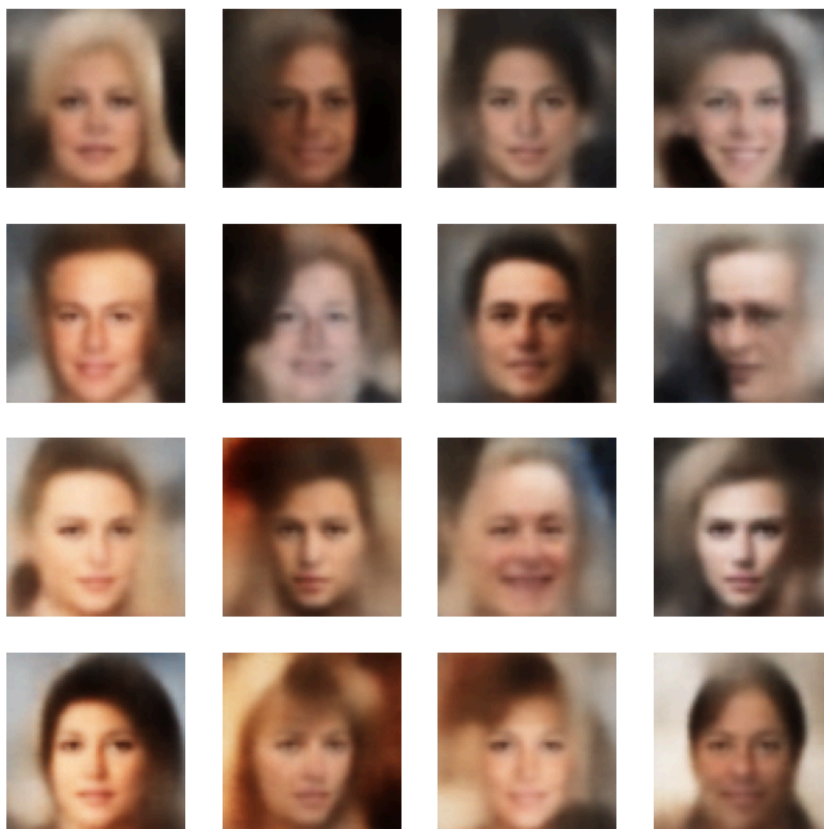
Epoch [7/10], Loss: 192.3561

Epoch [8/10], Loss: 190.3011

Epoch [9/10], Loss: 188.5780

Epoch [10/10], Loss: 187.1841

Randomly Generated CelebA Faces



Latent Space Grid Traversal (Varying dims 0 and 1)

