EX 8   IMAGE GENERATION USING VARIATIONAL AUTOENCODER (VAE)

DATE: 16/10/25

Problem Statement:

Implement a Variational Autoencoder (VAE) to generate new images from a given dataset. Train the model to learn the latent representation of images and generate new samples from the learned distribution.

Suggested Dataset: CelebA Dataset

Objectives:

1. Understand the concept of generative models using latent space representations.
2. Implement encoder-decoder architecture with reparameterization.
3. Train a VAE on CelebA and generate new human face images.
4. Visualize generated samples from the latent space.

Scope:

VAEs are probabilistic generative models capable of learning latent representations and generating realistic samples. This experiment explores the VAE pipeline—encoding, sampling via reparameterization, and decoding—to generate new samples that resemble the training distribution.

Tools and Libraries Used:

1. Python 3.x
2. PyTorch
3. torchvision
4. matplotlib
5. CelebA Dataset

Implementation Steps:

Step 1: Import Required Libraries

import torch import
torch.nn as nn
import
torch.nn.functional
as F import
torchvision

```python
from torchvision import datasets,
transforms from torch.utils.data import
DataLoader, Subset import
matplotlib.pyplot as plt
```

Step 2: Configure Parameters and Load Dataset

```python
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

image_size = 64
batch_size = 128
latent_dim = 100
num_epochs = 5
learning_rate = 1e-3

transform = transforms.Compose([
transforms.CenterCrop(178),
transforms.Resize(image_size),
transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

dataset    =    datasets.CelebA(root='data',    split='train',    download=True,
transform=transform) dataset = Subset(dataset, range(500))  # Limit to 500
samples for speed dataloader = DataLoader(dataset, batch_size=batch_size,
shuffle=True)
```

Step 3: Define Encoder Network

```python
class
Encoder(nn.Module):
def __init__(self,
latent_dim):
super(Encoder,
self).__init__()
self.conv = nn.Sequential(
      nn.Conv2d(3, 64, 4, 2, 1), nn.ReLU(),
nn.Conv2d(64, 128, 4, 2, 1), nn.BatchNorm2d(128),
nn.ReLU(),           nn.Conv2d(128, 256, 4, 2, 1),
nn.BatchNorm2d(256), nn.ReLU(),          nn.Conv2d(256,
512, 4, 2, 1), nn.BatchNorm2d(512), nn.ReLU()
    )
    self.fc_mu = nn.Linear(512*4*4, latent_dim)
self.fc_logvar = nn.Linear(512*4*4, latent_dim)
```

```python
    def
forward(self, x):
x = self.conv(x)
x =
x.view(x.size(0),
-1)
        return self.fc_mu(x), self.fc_logvar(x)
```

Step 4: Define Decoder Network

```python
class Decoder(nn.Module):    def __init__(self, latent_dim):
super(Decoder, self).__init__()       self.fc = nn.Linear(latent_dim,
512*4*4)       self.deconv = nn.Sequential(
nn.ConvTranspose2d(512, 256, 4, 2, 1), nn.BatchNorm2d(256),
nn.ReLU(),         nn.ConvTranspose2d(256, 128, 4, 2, 1),
nn.BatchNorm2d(128), nn.ReLU(),        nn.ConvTranspose2d(128,
64, 4, 2, 1), nn.BatchNorm2d(64), nn.ReLU(),
nn.ConvTranspose2d(64, 3, 4, 2, 1), nn.Tanh()
    )

  def forward(self, z):
    x = self.fc(z)
    x = x.view(x.size(0), 512, 4, 4)
    return self.deconv(x)
```

Step 5: Define the VAE Model

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

  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)
    return self.decoder(z), mu, logvar
```

Step 6: Define Loss Function

```python
def vae_loss(recon_x, x, mu, logvar):    recon_loss =
F.mse_loss(recon_x, x, reduction='sum')    kl_div = -
0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
return recon_loss + kl_div
```

Step 7: Train the Model and Generate Images

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

for epoch in range(num_epochs):
  model.train()    total_loss = 0
for images, _ in dataloader:
images = images.to(device)
recon, mu, logvar = model(images)
loss = vae_loss(recon, images, mu,
logvar)

    optimizer.zero_grad()
loss.backward()
    optimizer.step()

    total_loss += loss.item()

  print(f"Epoch                    [{epoch+1}/{num_epochs}],                    Loss:
{total_loss/len(dataloader.dataset):.4f}")

  # Generate and visualize samples
  model.eval()    with
torch.no_grad():       z =
torch.randn(64,
latent_dim).to(device)
    sample_images = model.decoder(z).cpu() * 0.5 + 0.5  # De-
normalize      grid =
torchvision.utils.make_grid(sample_images, nrow=8)
plt.imshow(grid.permute(1, 2, 0))       plt.axis('off')
    plt.title(f"Generated Faces - Epoch {epoch+1}")
plt.show()
```
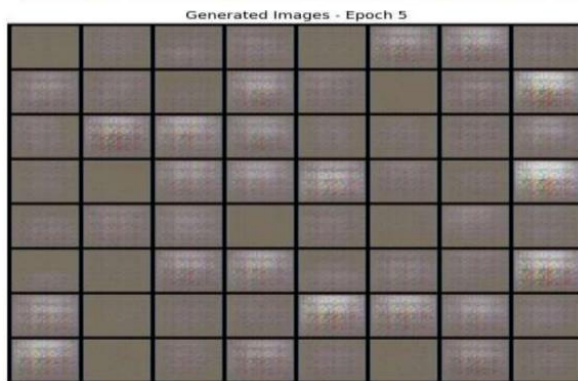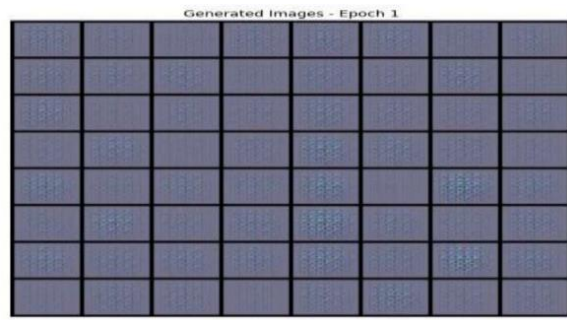
Generated Images - Epoch 1

Generated Images - Epoch 5

Conclusion:

In this experiment, a Variational Autoencoder was implemented and trained on a subset of the CelebA dataset. The model effectively learned to encode facial features into a latent space and generate realistic synthetic faces from random samples in that space. This showcases the generative power of deep neural architectures.