39_Sanskruti Nijai

EXPERIMENT NO 05

Design the architecture and implement the autoencoder model for Image Compression.

Code and output:

```
import torch
import torchvision
from torch import nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
rng = np.random.default_rng(123456)
```

```
[2] data = torchvision.datasets.MNIST(root='~/data', download=True)
    data = data.data
    data = data.float() / 255.
    data = data.view(-1, 1, 28, 28)
    print(data.shape)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to /root/data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 214073853.54it/s]Extracting /root/data/MNIST/raw/train-images-idx3-ubyte.gz to /root/data/MNIST/raw


Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to /root/data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 11598591.90it/s]
Extracting /root/data/MNIST/raw/train-labels-idx1-ubyte.gz to /root/data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to /root/data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 79304314.98it/s]Extracting /root/data/MNIST/raw/t10k-images-idx3-ubyte.gz to /root/data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to /root/data/MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|██████████| 4542/4542 [00:00<00:00, 13482327.51it/s]
Extracting /root/data/MNIST/raw/t10k-labels-idx1-ubyte.gz to /root/data/MNIST/raw

torch.Size([60000, 1, 28, 28])
```

```
[3] class AutoEncoder(nn.Module):
        def __init__(self):
            super().__init__()
            self.encoder = nn.Sequential(
                nn.Flatten(),
                nn.Linear(28*28, 100),
                nn.ReLU(),
                nn.Linear(100, 10),
                nn.ReLU(),
            )
            self.decoder = nn.Sequential(
                nn.Linear(10, 100),
                nn.ReLU(),
                nn.Linear(100, 28*28),
                nn.Sigmoid()
            )

        def encode(self, x):
            return self.encoder(x)

        def decode(self, x):
            x = self.decoder(x)
            return x.view(-1,1,28,28)

        def forward(self, x):
            return self.decode(self.encode(x))
```

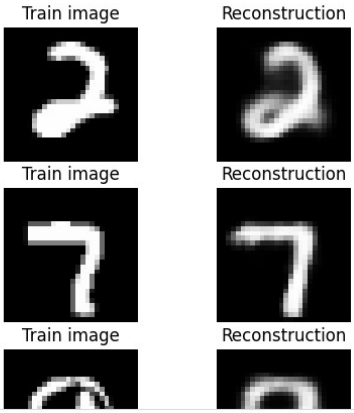
```
[4] model = AutoEncoder().cuda()
    opt = torch.optim.Adam(model.parameters())
```

```
[5] for epoch in range(25):
        print(f'Epoch {epoch+1}/25')
```

```
[5] for epoch in range(25):
        print(f'Epoch {epoch+1}/25')
        for i in range(0, data.shape[0], 32):
            x = data[i:i+32].cuda()
            x_rec = model(x)
            loss = F.binary_cross_entropy(x_rec, x)

            opt.zero_grad()
            loss.backward()
            opt.step()

        data = data[rng.permutation(len(data))]
        print(f'\tloss: {loss.item():.4f}')
```

```
Epoch 1/25
        loss: 0.1784
Epoch 2/25
        loss: 0.1492
Epoch 3/25
        loss: 0.1418
Epoch 4/25
        loss: 0.1496
Epoch 5/25
        loss: 0.1422
Epoch 6/25
        loss: 0.1400
Epoch 7/25
        loss: 0.1547
Epoch 8/25
        loss: 0.1331
Epoch 9/25
        loss: 0.1508
Epoch 10/25
        loss: 0.1287
Epoch 11/25
        loss: 0.1538
```

```
[6] plt.figure(figsize=(5,10))

    for i in range(5):
        plt.subplot(5, 2, i*2+1, title=f'Train image')
        plt.imshow(np.squeeze(x[i].cpu()), cmap='gray')
        plt.axis('off')

        plt.subplot(5, 2, i*2+2, title='Reconstruction')
        with torch.no_grad(): plt.imshow(np.squeeze(x_rec[i].cpu()), cmap='gray')
        plt.axis('off')
```




```
[7] # Sample two random images and encode
    f = model.encode(x[0:2])
    f1,f2 = f[0].unsqueeze(0),f[1].unsqueeze(0)

    # Show reconstructions of interpolated codes
    plt.figure(figsize=(20,5))
    reconstructions = []
    for i in range(20):
        v = i/19.
        f_interp = f1*(1-v) + f2*v
        with torch.no_grad():
            x_rec_interp = np.squeeze(model.decode(f_interp).cpu())
            reconstructions.append(x_rec_interp)

        plt.subplot(2,10,i+1)
        plt.imshow(x_rec_interp, cmap='gray')
        plt.axis('off')
```