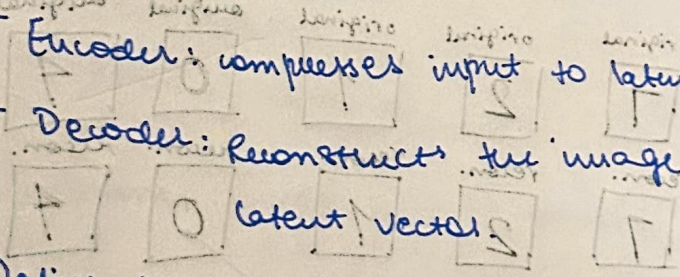
Exp 10:

Perform compression on mnist dataset using auto encoder.

Aim: To implement an Autoencoder that compresses and reconstruct MNIST handwritten digit images using PyTorch.

Pseudo code:

1. Import libraries & load MNIST dataset
2. Normalize & flatten the images
3. Define Autoencoder:
   - Encoder: compresses input to latent vector
   - Decoder: Reconstructs the image from latent vector



4. Define loss (MSE) & optimizer (Adam)
5. Train model:
   → forward pass
   ~~Backward pass~~
   → compute loss
   → Backpropagate
   → Update weights.
6. Evaluate on test data
7. Visualize:
   → Original v/s Reconstructed images
   → Latent space (compressed features)

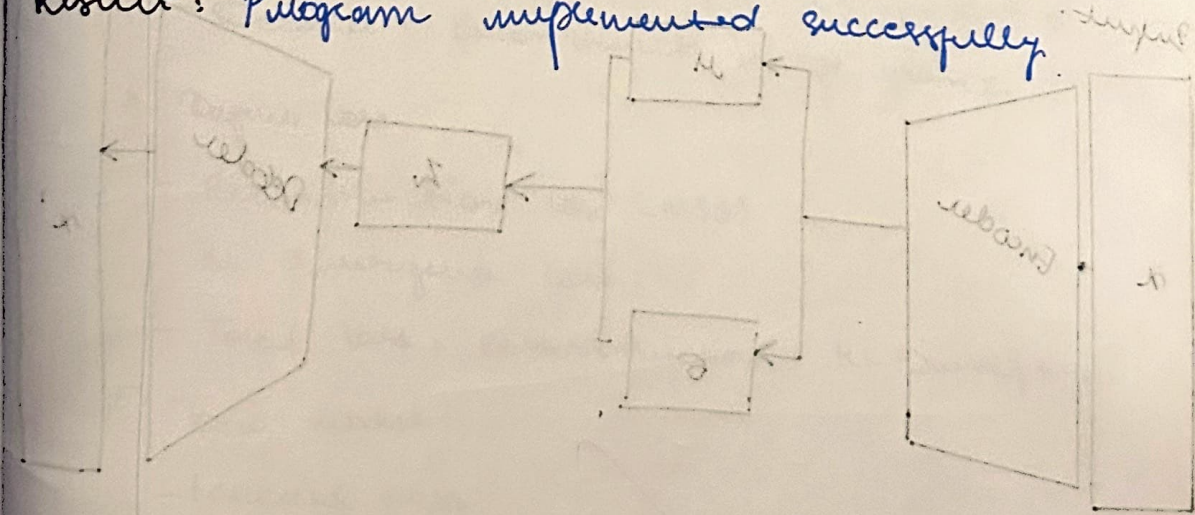
Justification:

An autoencoder is an unsupervised neural network that learns to compress (encode) input data into a smaller representative (latent space) and then reconstruct (decode) it back to the original form.

* It helps in:

→ Data compression.
→ Noise reduction
→ feature extraction
→ Dimensionality reduction.

Result: Program implemented successfully.



Output :

Epoch 1, loss: 0.0257

Epoch 2, loss: 0.0163

Epoch 3, loss: 0.0141

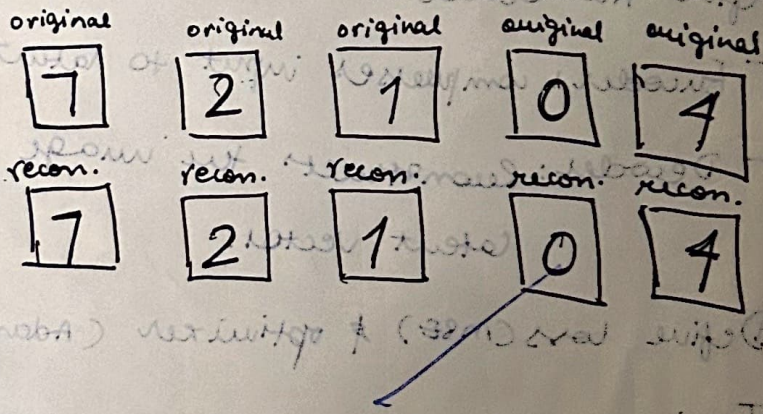
Epoch 4, loss: 0.0121

Epoch 5, loss: 0.0098.

Accuracy: 0.9689

Recall: 0.8528

f1 : 0.8676



Commands | + Code | + Text | Run all

[1]

```
import torch, torch.nn as nn, torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, recall_score, f1_score
transform = transforms.ToTensor()
train = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test = datasets.MNIST(root='./data', train=False, download=True, transform=transform)
train_loader = DataLoader(train, batch_size=128, shuffle=True)
test_loader = DataLoader(test, batch_size=128, shuffle=False)
class AE(nn.Module):
    def __init__(self):
        super().__init__()
        self.enc = nn.Sequential(nn.Linear(784,128), nn.ReLU(), nn.Linear(128,32))
        self.dec = nn.Sequential(nn.Linear(32,128), nn.ReLU(), nn.Linear(128,784), nn.Sigmoid())
    def forward(self,x):
        x = x.view(-1,784)
        return self.dec(self.enc(x))

device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = AE().to(device)
opt = optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.MSELoss()
for epoch in range(5):
    for imgs,_ in train_loader:
```

Variables | Terminal

8:37 AM | Python 3

Untitled8.ipynb ☆ Saving failed since 8:35 AM

File Edit View Insert Runtime Tools Help

Share A

Commands + Code + Text ▷ Run all

RAM Disk

[1]

```
for epoch in range(5):
    for imgs,_ in train_loader:
        imgs = imgs.to(device)
        out = model(imgs)
        loss = loss_fn(out, imgs.view(-1,784))
        opt.zero_grad(); loss.backward(); opt.step()
    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
model.eval()
imgs, _ = next(iter(test_loader))
imgs = imgs.to(device)
recon = model(imgs).cpu().detach().view(-1,1,28,28)
orig = imgs.cpu().view(-1,784).numpy() > 0.5
recon_bin = recon.view(-1,784).numpy() > 0.5
acc = accuracy_score(orig.flatten(), recon_bin.flatten())
rec = recall_score(orig.flatten(), recon_bin.flatten())
f1 = f1_score(orig.flatten(), recon_bin.flatten())
print(f"Accuracy: {acc:.4f}, Recall: {rec:.4f}, F1: {f1:.4f}")
fig, ax = plt.subplots(2,5, figsize=(10,4))
for i in range(5):
    ax[0,i].imshow(imgs[i].cpu().view(28,28), cmap='gray'); ax[0,i].set_title("Original")
    ax[1,i].imshow(recon[i].view(28,28), cmap='gray'); ax[1,i].set_title("Reconstructed")
    ax[0,i].axis('off'); ax[1,i].axis('off')
plt.show()
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 55.4MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.82MB/s]
```

Variables Terminal

8:37 AM

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 55.4MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.82MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 14.6MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 9.43MB/s]
Epoch 1, Loss: 0.0257
Epoch 2, Loss: 0.0163
Epoch 3, Loss: 0.0141
Epoch 4, Loss: 0.0121
Epoch 5, Loss: 0.0098
Accuracy: 0.9689, Recall: 0.8528, F1: 0.8676
```