D. Khaja Nawaz

RA2311047010037

Deep Learning Techniques (LaB)

Date	Title	Sign
24/07/2025	1. Exploring the deep learning platform	
31/07/25	2. Implement a classifier using open-source dataset	7/8/25
31/07/25	3. Study of the classifiers with respect to statistical parameters	14/8/25
14/08/25	4. Build a simple feed forword neural network to recognize handwritten character	14/8/25
22/08/25	5. Study of Activation functions and their role	9/9
09/09/25	6. ~~Implement~~ Implement gradient descent and backpropagation in deep neural network	
16/09/25	7. Build a CNN model to classify cat and Dog image	23/9/25
30/09/2025	8. Experiment of LSTM	10/10/25
30/09/2025	9. Build a Recurrent Neural Network	
9/10/2025	10. perform compression on MNIST dataset using autoencoder	
9/10/2025	11. Experiments using variational autoen	
	12. Implement a Deep convolutional GAN to generate complex color image	
	13. Understanding the architecture of pre-trained model	
	14. Implement a pre-trained CNN model as feature Extractor using	
	15. Implement a Yolo model to detect objects	

02/10/25 Lab:10: perform compression on MNIST dataset using autoencoder.

*AIM =
Design, train and evaluate a simple auto encoder to compress and reconstruct MINIST images measures reconstruction quality.

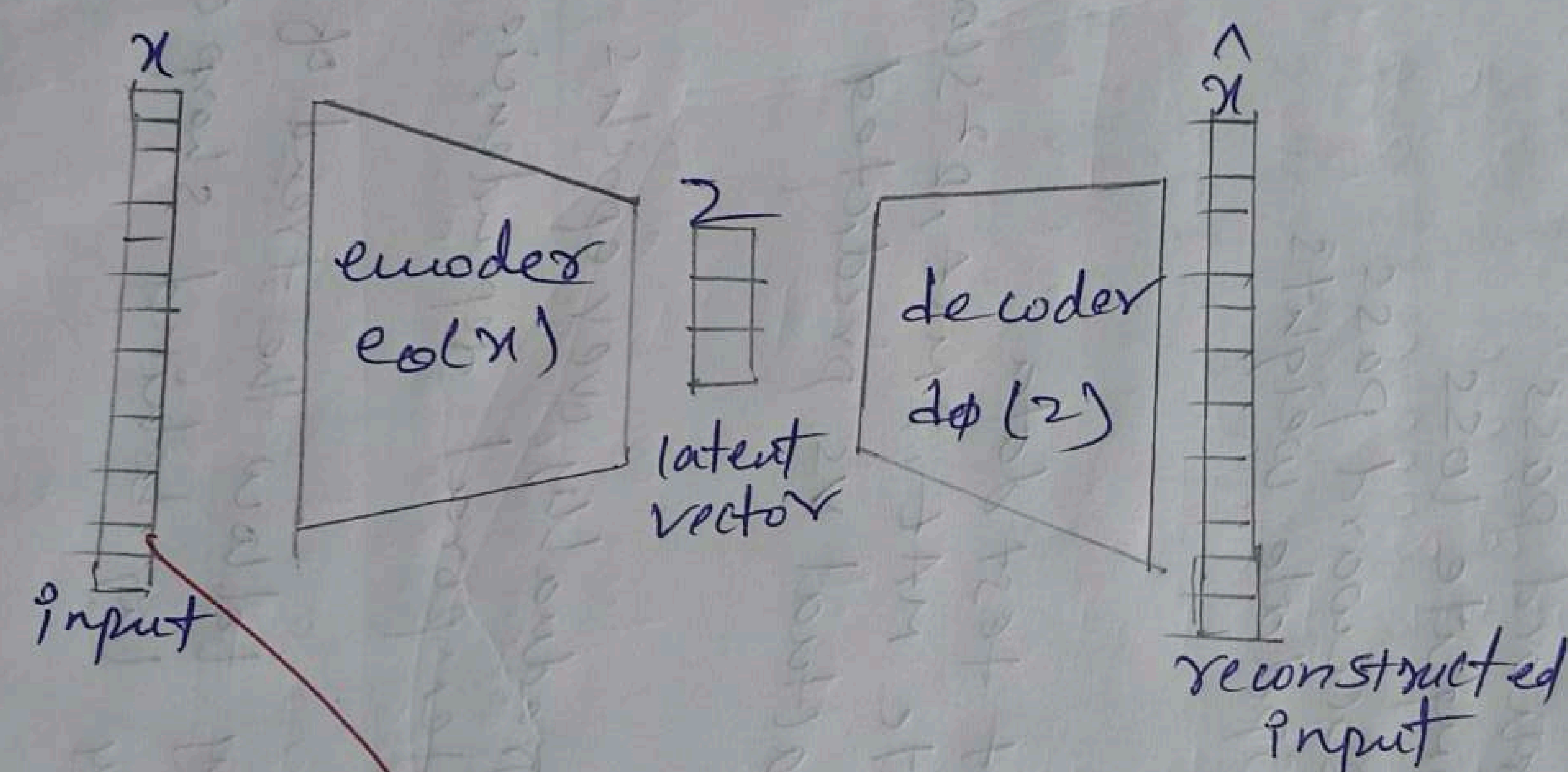
*objective =
1. To understand the concept of autoencoders for unsupervised learning
2. To compress high-dimensional image data (28X28 pixels) into a low-dimensional latent space
3. To reconstruct the original image from the compressed (encoded) representation.
4. To evaluate how well the autoencoder preserves image features after compression.
5. To visualize the origin and reconstructed images for comparison.

*pseudocode =
```
Begin
    Load MNIST dataset
    Normalize and reshape image(28X28 -> 784)

    Define encoder:
        Input: 784 neurons
        Hidden: 64 neurons (compressed layer)

    Define decoder:
        Input: 64 neurons
        Output: 784 neurons (reconstructed in
```

Architecture of autoencoder



Output :-

Epoch 1/10, loss : 0.0495

Epoch 2/10 loss = 0.0212

Epoch 3/10, loss = 0.0152

Epoch 4/10, loss = 0.0120

Epoch 5/10, loss = 0.0104

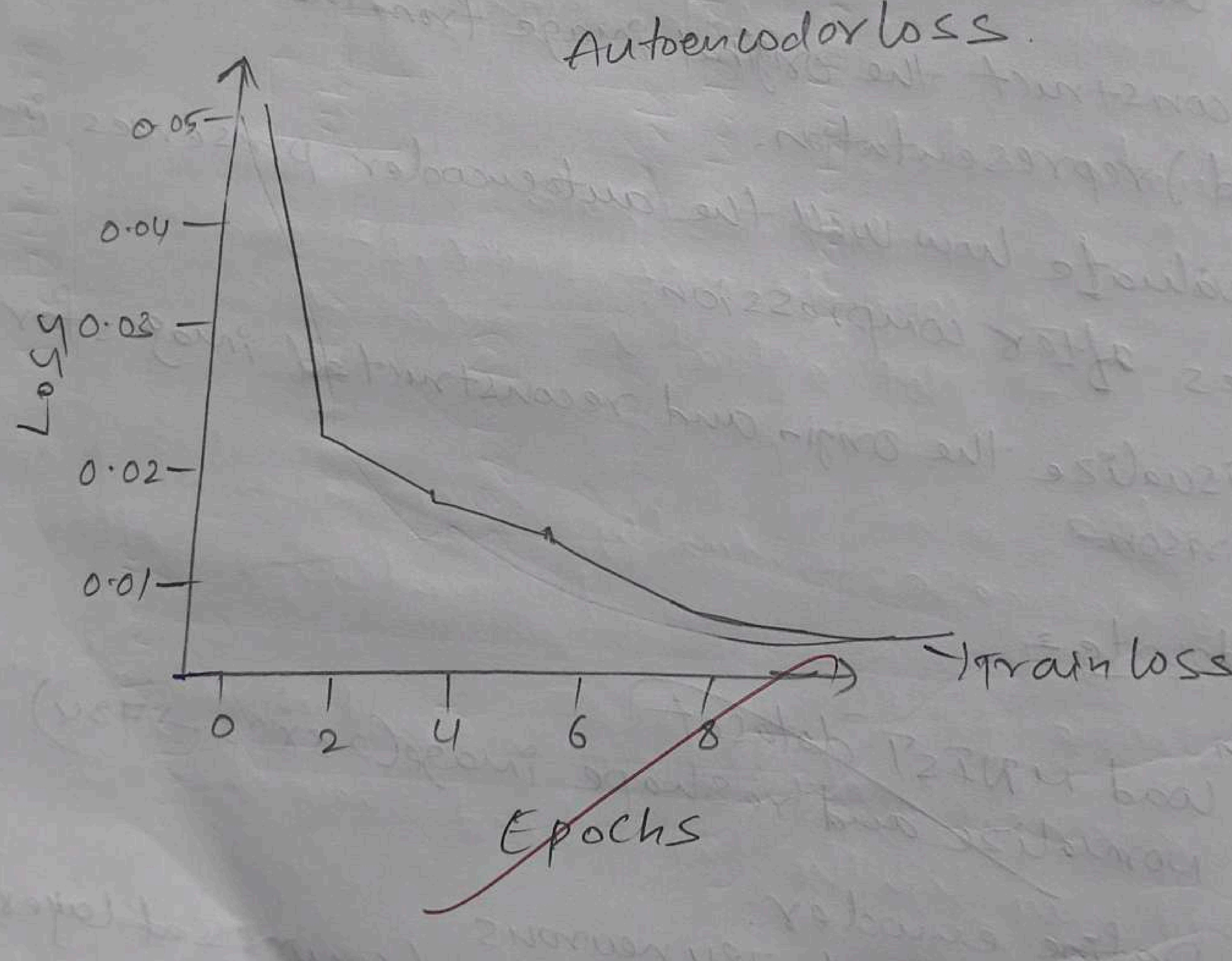
Epoch 6/10, loss = 0.0093

Epoch 7/10, loss = 0.0085

Epoch 8/10, loss = 0.0078

Epoch 9/10, loss = 0.0073

Epoch 10, loss = 0.0068



compile model using Adam optimizer and binary cross entropy loss

Train model with input=output for several epochs

Encode test data → get compressed features

Decode compressed data → reconstruct images

Decode compressed data → reconstructed images.

Display original vs reconstructed images.

*Observation:

→ The training accuracy increases with epoch while the loss decreases

→ The reconstructed images visually resemble the original MINST.

→ The decreasing trend in the overall loss suggest the latent space in becoming more structured and aligned.

→ The AE successfully reconstructed the original input data from its compressed latent representation.

*Result:

Successfully compressed on MNIST dataset using autoencoders

Lab10.ipynb

File Edit View Insert Runtime Tools Help

Commands | + Code | + Text | Run all

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

# Hyperparameters
batch_size = 128
epochs = 10
learning_rate = 1e-3
latent_dim = 64

# Dataset
transform = transforms.ToTensor()
train_dataset = datasets.MNIST(root='data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='data', train=False, download=True, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Autoencoder
class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(nn.Flatten(), nn.Linear(28*28, 128), nn.ReLU(), nn.Linear(128, latent_dim))
        self.decoder = nn.Sequential(nn.Linear(latent_dim, 128), nn.ReLU(), nn.Linear(128, 28*28), nn.Sigmoid())
    def forward(self, x):
        z = self.encoder(x)
        return self.decoder(z)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training with loss tracking
loss_list = []
for epoch in range(epochs):
    running_loss = 0
    for x, _ in train_loader:
        x = x.to(device)
        optimizer.zero_grad()
        out = model(x)
        loss = criterion(out, x.view(x.size(0), -1))
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_loss = running_loss / len(train_loader)
```

Lab10.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
    print(f"Epoch {epoch+1}, Loss: {avg_loss:.4f}")

# Plot loss
plt.plot(loss_list, label='Train Loss')
plt.xlabel('Epochs'); plt.ylabel('Loss'); plt.title('Autoencoder Loss'); plt.legend(); plt.sho
```

```
Epoch 1, Loss: 0.0495
Epoch 2, Loss: 0.0212
Epoch 3, Loss: 0.0152
Epoch 4, Loss: 0.0120
Epoch 5, Loss: 0.0104
Epoch 6, Loss: 0.0093
Epoch 7, Loss: 0.0085
Epoch 8, Loss: 0.0078
Epoch 9, Loss: 0.0073
Epoch 10, Loss: 0.0068
```