[1] ✓ 16s

```
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
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
```

[2] ✓ 3s

```
transform = transforms.Compose([transforms.ToTensor()])
train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_data, batch_size=256, shuffle=True)
test_loader = DataLoader(test_data, batch_size=256, shuffle=False)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 17.9MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 481kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.49MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 11.7MB/s]
```

[3] ✓ 0s

```
class Autoencoder(nn.Module):
    def __init__(self, encoding_dim=32):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, encoding_dim)
```

Untitled8.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands | + Code | + Text | Run all

RAM Disk

[3] ✓ 0s

```
        def forward(self, x):
            x = x.view(x.size(0), -1)
            encoded = self.encoder(x)
            decoded = self.decoder(encoded)
            return decoded
```

[4] ✓ 0s

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Autoencoder(encoding_dim=32).to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

[5] ✓ 3m

```
num_epochs = 20
train_losses = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for imgs, _ in train_loader:
        imgs = imgs.to(device)
        outputs = model(imgs)
        loss = criterion(outputs, imgs.view(imgs.size(0), -1))

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

        running_loss += loss.item()

    avg_loss = running_loss / len(train_loader)
    train_losses.append(avg_loss)
    print(f"Epoch [{epoch+1}/{num_epochs}] Loss: {avg_loss:.4f}")
```

Variables | Terminal

7:05 PM | Python 3

```
Epoch [1/20] Loss: 0.0743
Epoch [2/20] Loss: 0.0390
Epoch [3/20] Loss: 0.0303
Epoch [4/20] Loss: 0.0266
Epoch [5/20] Loss: 0.0243
Epoch [6/20] Loss: 0.0224
Epoch [7/20] Loss: 0.0204
Epoch [8/20] Loss: 0.0192
Epoch [9/20] Loss: 0.0181
Epoch [10/20] Loss: 0.0173
Epoch [11/20] Loss: 0.0165
Epoch [12/20] Loss: 0.0159
Epoch [13/20] Loss: 0.0154
Epoch [14/20] Loss: 0.0148
Epoch [15/20] Loss: 0.0144
Epoch [16/20] Loss: 0.0140
Epoch [17/20] Loss: 0.0136
Epoch [18/20] Loss: 0.0132
Epoch [19/20] Loss: 0.0128
Epoch [20/20] Loss: 0.0125
```

[6]

```
model.eval()
```

Untitled8.ipynb

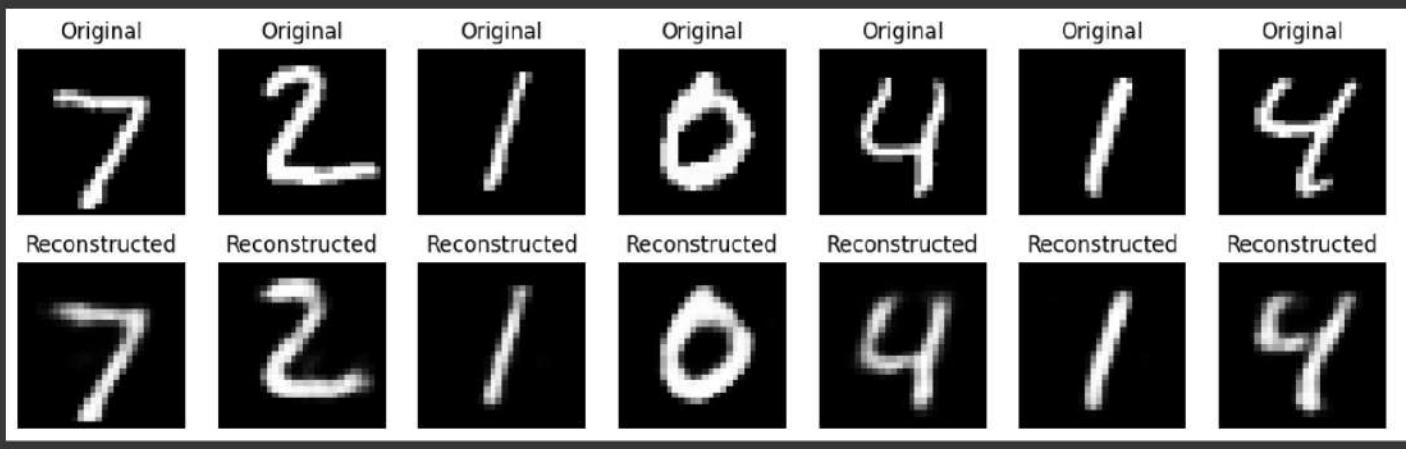
File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[7] ✓ 0s

```
    plt.figure(figsize=(20, 4))
    for i in range(n):
        # Original
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(original[i].reshape(28, 28), cmap='gray')
        plt.title("Original")
        plt.axis("off")
        # Reconstructed
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(reconstructed[i].reshape(28, 28), cmap='gray')
        plt.title("Reconstructed")
        plt.axis("off")
    plt.show()

show_images(originals, reconstructions)
```



Variables Terminal

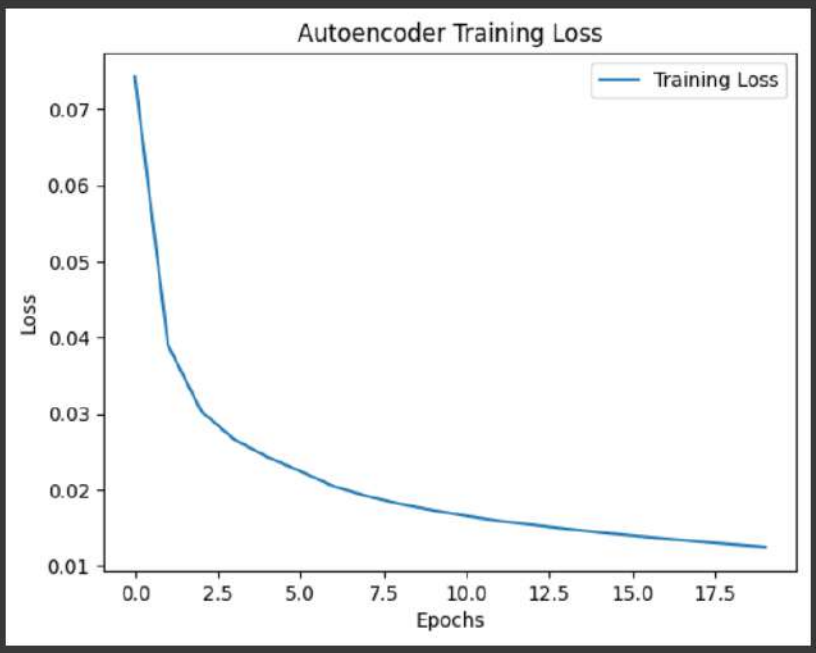
Untitled8.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

[8] ✓ 0s

```
plt.plot(train_losses, label='Training Loss')
plt.title("Autoencoder Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Variables Terminal

Untitled9.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands + Code + Text Run all

RAM Disk

+ Code + Text

[1]
```
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
```

[2]
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

[3]
```
batch_size = 128
learning_rate = 1e-3
latent_dim = 20
epochs = 15
```

[4] 3s
```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 18.8MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 503kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.67MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 7.55MB/s]
```

Variables Terminal

Python 3

Untitled9.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[5]

```
class VAE(nn.Module):
    def __init__(self, latent_dim):
        super(VAE, self).__init__()

        # Encoder
        self.fc1 = nn.Linear(784, 400)
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)

        # Decoder
        self.fc3 = nn.Linear(latent_dim, 400)
        self.fc4 = nn.Linear(400, 784)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def encode(self, x):
        h1 = self.relu(self.fc1(x))
        return self.fc_mu(h1), self.fc_logvar(h1)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        h3 = self.relu(self.fc3(z))
        return self.sigmoid(self.fc4(h3))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
```

Variables Terminal

Untitled9.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands | + Code | + Text | Run all

RAM Disk

[11]

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

# ------------------------------
# 6. Initialize Model and Optimizer
# ------------------------------
model = VAE(latent_dim).to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# ------------------------------
# 7. Training Loop
# ------------------------------
train_losses = []

model.train()
for epoch in range(epochs):
    train_loss = 0
    for data, _ in train_loader:
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)
        loss = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

    avg_loss = train_loss / len(train_loader.dataset)
    train_losses.append(avg_loss)
```

Variables | Terminal

Python 3

Untitled9.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[13]

```
# ------------------------------
# 10. Latent Space Visualization (for latent_dim = 2)
# ------------------------------
if latent_dim == 2:
    zs, labels = [], []
    with torch.no_grad():
        for data, target in test_loader:
            data = data.to(device)
            mu, logvar = model.encode(data.view(-1, 784))
            z = model.reparameterize(mu, logvar)
            zs.append(z.cpu())
            labels.append(target)
    zs = torch.cat(zs)
    labels = torch.cat(labels)

    plt.figure(figsize=(7,6))
    plt.scatter(zs[:, 0], zs[:, 1], c=labels, cmap='tab10', s=10)
    plt.colorbar()
    plt.title("Latent Space Representation (2D)")
    plt.xlabel("z₁")
    plt.ylabel("z₂")
    plt.show()

# ------------------------------
# 11. Generate New Images
# ------------------------------
with torch.no_grad():
    z = torch.randn(16, latent_dim).to(device)
    samples = model.decode(z).cpu()

    plt.figure(figsize=(6, 6))
    for i in range(16):
        plt.subplot(4, 4, i+1)
```

Variables Terminal

Untitled9.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[13]

```
# ------------------------------
# 11. Generate New Images
# ------------------------------
with torch.no_grad():
    z = torch.randn(16, latent_dim).to(device)
    samples = model.decode(z).cpu()

    plt.figure(figsize=(6, 6))
    for i in range(16):
        plt.subplot(4, 4, i+1)
        plt.imshow(samples[i].view(28, 28), cmap='gray')
        plt.axis('off')
    plt.suptitle("Generated Digits from Random Latent Vectors", fontsize=14)
    plt.show()

# ------------------------------
# 12. Compute Simple Reconstruction Accuracy
# ------------------------------
def reconstruction_accuracy(original, reconstructed, threshold=0.5):
    original = original.view(-1, 784)
    reconstructed = (reconstructed > threshold).float()
    correct = (original == reconstructed).float().mean()
    return correct.item() * 100

acc = reconstruction_accuracy(test_batch.cpu(), recon_batch.cpu())
print(f"Reconstruction Accuracy: {acc:.2f}%")
```

```
Epoch [1/15] - Average Loss: 189.9661
Epoch [2/15] - Average Loss: 167.3070
Epoch [3/15] - Average Loss: 162.9294
Epoch [4/15] - Average Loss: 160.4904
Epoch [5/15] - Average Loss: 158.7898
Epoch [6/15] - Average Loss: 157.5080
```

Variables Terminal

```
Epoch [1/15] - Average Loss: 189.9661
Epoch [2/15] - Average Loss: 167.3070
Epoch [3/15] - Average Loss: 162.9294
Epoch [4/15] - Average Loss: 160.4904
Epoch [5/15] - Average Loss: 158.7898
Epoch [6/15] - Average Loss: 157.5080
Epoch [7/15] - Average Loss: 156.4309
Epoch [8/15] - Average Loss: 155.5703
Epoch [9/15] - Average Loss: 154.8629
Epoch [10/15] - Average Loss: 154.2743
Epoch [11/15] - Average Loss: 153.6860
Epoch [12/15] - Average Loss: 153.2136
Epoch [13/15] - Average Loss: 152.7763
Epoch [14/15] - Average Loss: 152.3487
Epoch [15/15] - Average Loss: 151.9465
```

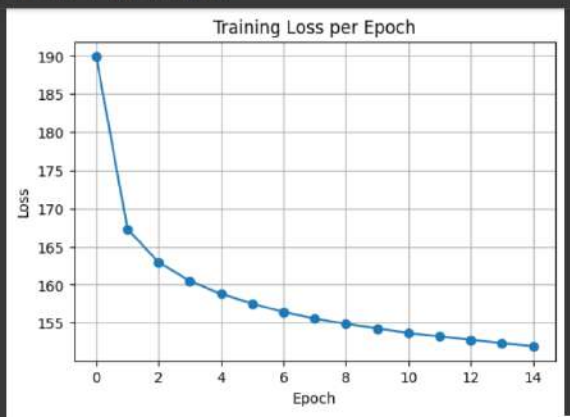
Untitled9.ipynb

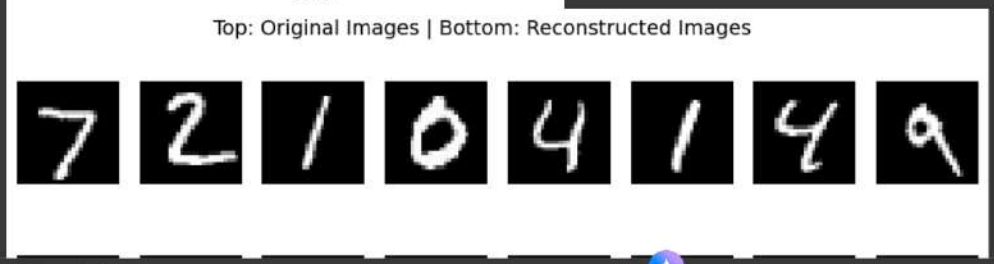
File Edit View Insert Runtime Tools Help

Share

Commands | + Code + Text | Run all

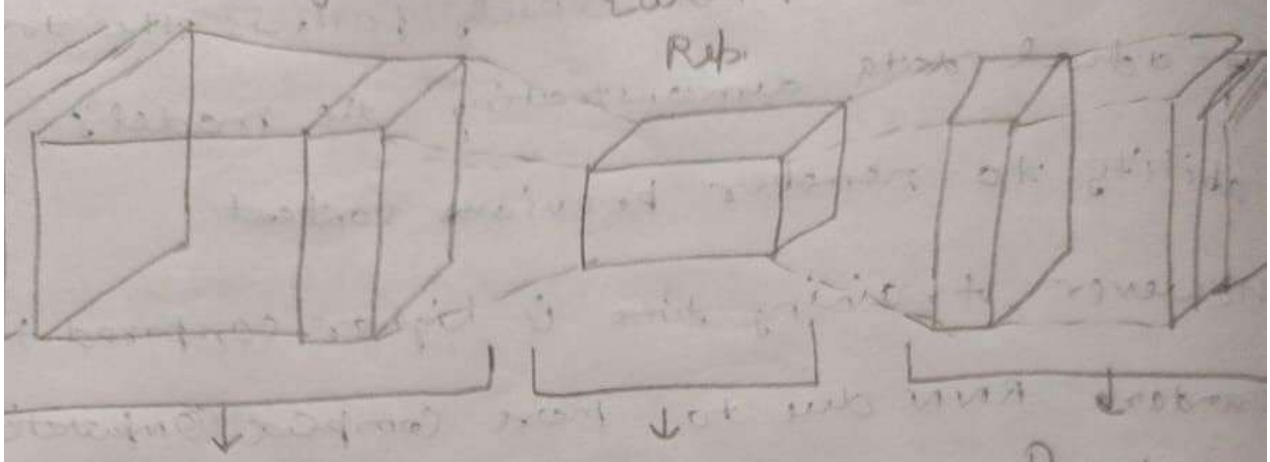
RAM Disk





Variables Terminal

Python 3



(Auto encoder)

8-10-25

Lab 10:- (Perform Compression on MNIST dataset using auto encoder)

Aim:- To implement an Autoencoder neural network for compressing and reconstructing images from MNIST dataset.

Pseudo code:-

- Import Required libraries.
- Load the MNIST dataset.
- Normalization of pixel values.
- Flattem the images into 1D vectors.
- Define the Auto encoder model:

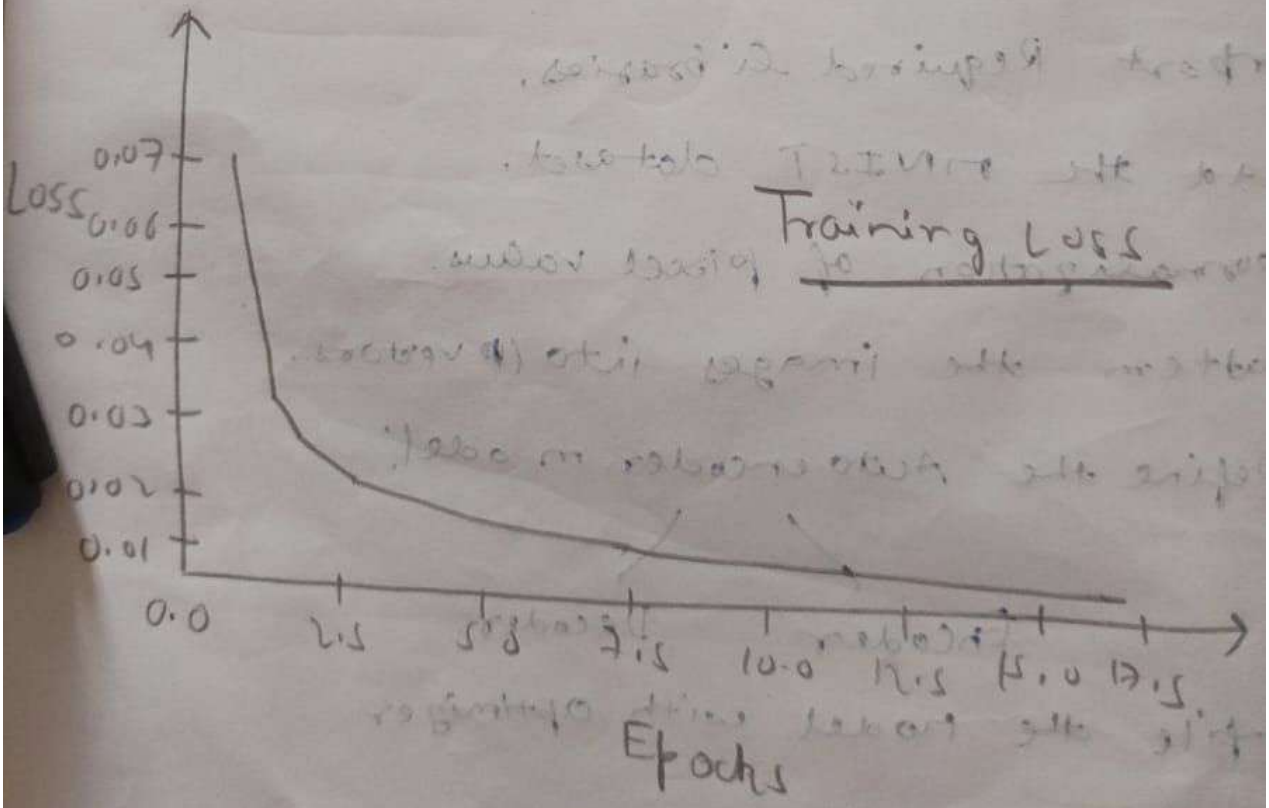
  Encoder Decoder.
- Compile the model with Optimizer
- Train the Model
- Use encoder part to compress images.
- Use decoder part to reconstruct images.
- Visualize:

(observation)

- The autoencoder successfully learns to reconstruct MNIST digits after several epoc

Output

Epoch 1: Loss = 0.0743
Epoch 2: Loss = 0.0390
Epoch 3: Loss = 0.0303
Epoch 4: Loss = 0.0266
Epoch 5: Loss = 0.0243
Epoch 6: Loss = 0.0204
Epoch 7: Loss = 0.0192
Epoch 8: Loss = 0.0181
Epoch 9: Loss = 0.0173
Epoch 10: Loss = 0.0165



Training Loss

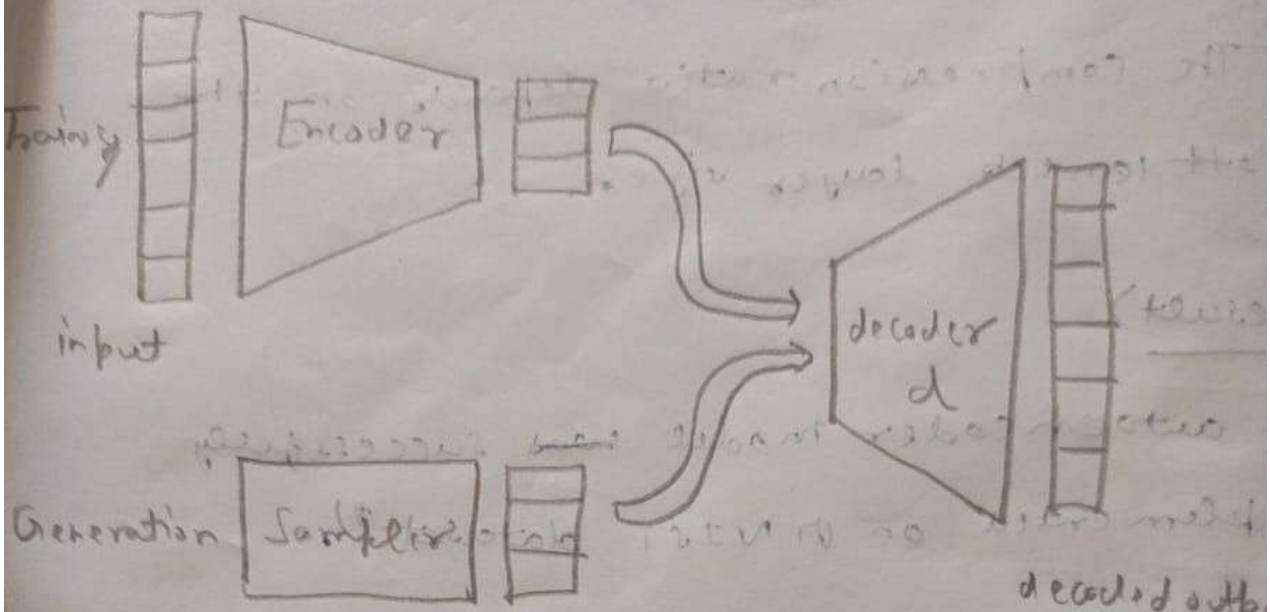
. The training loss decreases gradually,

. The compressed representation is much smaller in size compared to original.

. The compression ratio depends on the bottleneck layer size.

(Result)

An autoencoder model was successfully implemented on MNIST dataset [1].


Training
Encoder
input
decoder
Generation
Sampler
decoded o/p

(V.A.E)

17-10-25

Lab 11:- (Using Variational Auto encoders)

Aim:- To implement and analyse a Variational Auto encoder (VAE) for learning latent representation of MNIST dataset.

Pseudocode:-

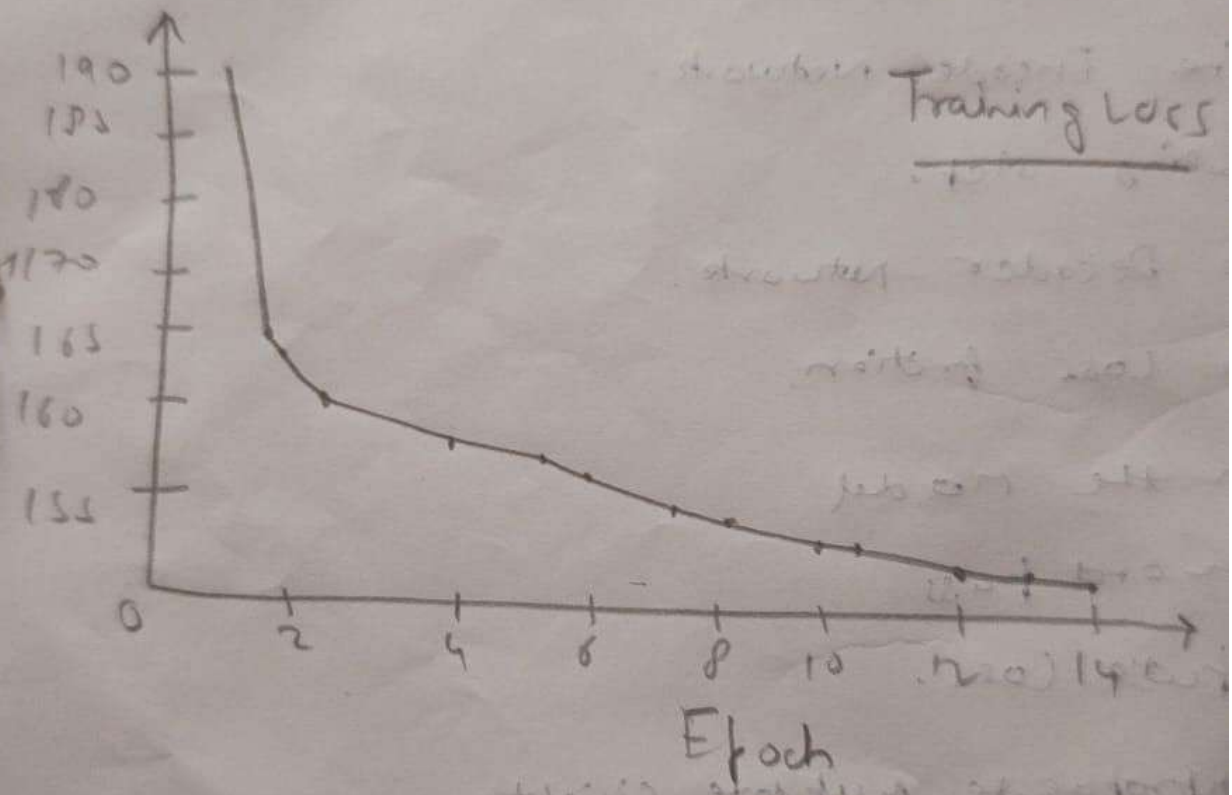
- Import libraries.
- Load MNIST dataset.
- Define Encoder Network.
- Sampling step.
- Define Decoder Network.
- Define Loss function.
- Train the Model
  - → Forward pass
  - → Compute Loss.
  - → Backpropogate & update weights.
- After Training
  - → Reconstruct test image
  - → Generate new images by sampling.
- Visualize.

(Observation)

- During trains, the reconstruction loss grad decreased, indicating better learning of input

Output:-

Epoch 1 : Loss = 189.966
Epoch 2 : Loss = 167.107
Epoch 3 : Loss = 162.929
Epoch 4 : Loss = 160.490
Epoch 5 : Loss = 158.789
Epoch 6 : Loss = 157.508
Epoch 7 : Loss = 156.430
Epoch 8 : Loss = 155.570
Epoch 9 : Loss = 154.862
Epoch 10 : Loss = 154.274



- The KL Divergence stabilized over time, showing that the latent space was learning a valid distribution.

- The model succesfully learned a smooth latent space, where interpolation b/w two points produced meaningful transitions between digits.

(Result)

"An variational Auto encoder model succesfully implemented on MNIST dataset."

Accuracy :- 80.67%.