

WEEK-10

Aim:- To build and train an Autoencoder to compress and reconstruct handwritten digit images.

Objective:-

- 1) Understand the concept of encoding and decoding.
- 2) Reduce input data dimensions using Autoencoder.
- 3) Visualize how the network learns compressed image features.

Algorithm:-

- 1) Import libraries and MNIST dataset.
- 2) Normalize image pixel values between 0-1.
- 3) Define encoder (Input \rightarrow compressed representation).
- 4) Define decoder (Compressed \rightarrow reconstructed image).
- 5) Combine encoder and decoder into one Autoencoder model.
- 6) Compile and train the model using MSE loss.
- 7) Evaluate performance and visualize original vs reconstructed image.

pseudo code:-

Begin

Load MNIST dataset

Normalize and reshape image ($28 \times 28 \rightarrow 784$)

Define ^{Encoder} ~~decoder~~:

Input: 784 Neurons

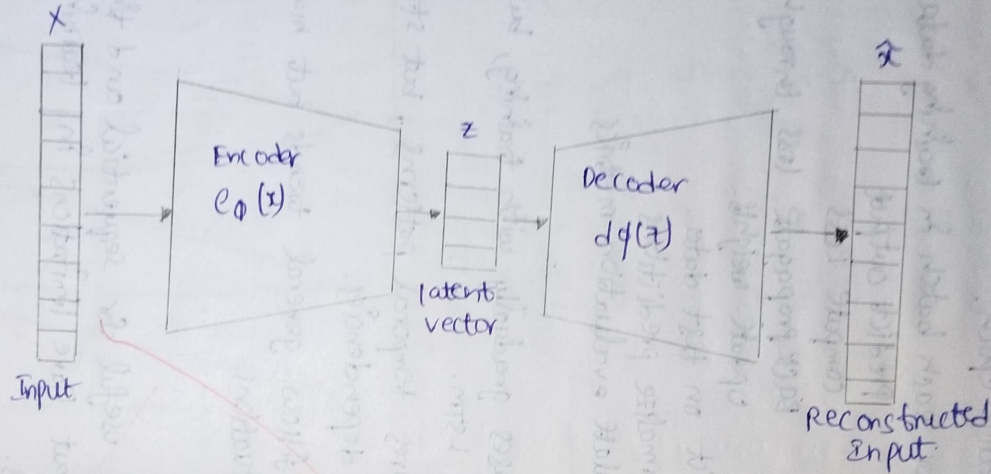
Hidden: 64 Neurons (compressed layer)

Define decoder:

Input: 64 Neurons

Output: 78 Neurons (reconstructed image)

Architecture of Autoencoder



compile model using Adam optimizer and binary cross entropy loss.

Train model with input-output for several epochs.

Encode test data \rightarrow get compressed features.

Decode compressed data \rightarrow reconstruct images.

Display original vs reconstructed images.

observation:-

- \rightarrow The training accuracy increases with epoch while the loss decreases.
- \rightarrow The reconstructed images visually resemble the original MNIST.
- \rightarrow The decreasing trend in the overall loss suggest the latent space is becoming more structured and designed.
- \rightarrow The Auto encoder successfully reconstructed the original input data from its compressed latent representation.

conclusion:-

Auto encoders can efficiently perform dimensionality reduction and data compression, learning important features automatically without labels.

Result:-

The Autoencoder successfully compressed MNIST images into low dimensional representations and reconstructed them with minimal distortion.

2/11/21

Output:-

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

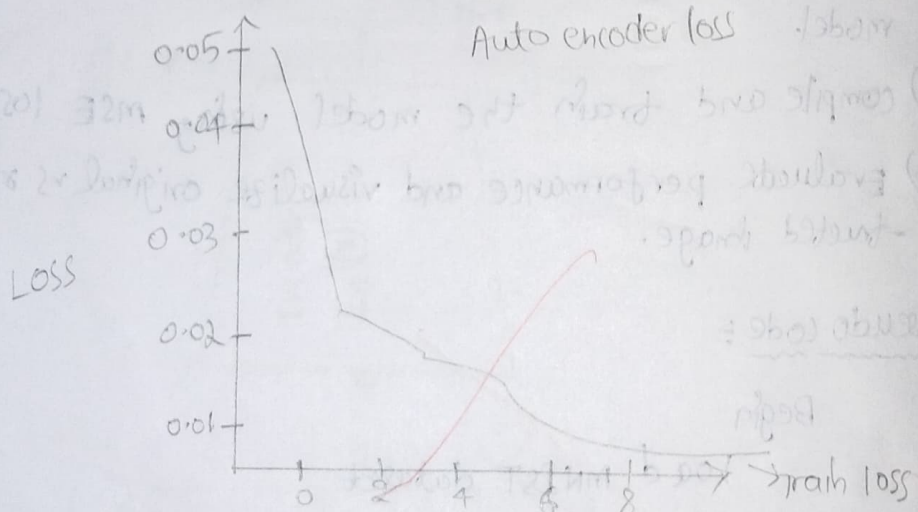
Epoch 6, loss = 0.0043

Epoch 7, loss = 0.0085

Epoch 8, loss = 0.0078

Epoch 9, loss = 0.0073

Epoch 10, loss = 0.0068





```
from tensorflow.keras.layers import Input, Dense
import matplotlib.pyplot as plt
import numpy as np

(x_train, _), (x_test, _) = mnist.load_data() # labels not needed
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

x_train = x_train.reshape(len(x_train), 784)
x_test = x_test.reshape(len(x_test), 784)

input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded) # compressed layer

decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded) # reconstruct pixels

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse', metrics=['accuracy'])

history = autoencoder.fit(
    x_train, x_train,
    epochs=10,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test, x_test)
)

plt.figure(figsize=(6,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Autoencoder Training Loss")
plt.xlabel("Epochs")
plt.ylabel("MSE Loss")
plt.legend()
plt.show()
decoded_imgs = autoencoder.predict(x_test)
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis("off")

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Reconstructed")
    plt.axis("off")
plt.show()
```

Epoch 1/10
235/235 — 9s 31ms/step - accuracy: 0.0100 - loss: 0.0978 - val_accuracy: 0.0004 - val_loss: 0.0363

Epoch 2/10
235/235 — 5s 22ms/step - accuracy: 0.0007 - loss: 0.0329 - val_accuracy: 0.0000 - val_loss: 0.0258

Epoch 3/10
235/235 — 4s 17ms/step - accuracy: 0.0007 - loss: 0.0249 - val_accuracy: 0.0100 - val_loss: 0.0216

Epoch 4/10
235/235 — 6s 19ms/step - accuracy: 0.0092 - loss: 0.0213 - val_accuracy: 0.0118 - val_loss: 0.0188

Epoch 5/10
235/235 — 4s 16ms/step - accuracy: 0.0098 - loss: 0.0186 - val_accuracy: 0.0106 - val_loss: 0.0170

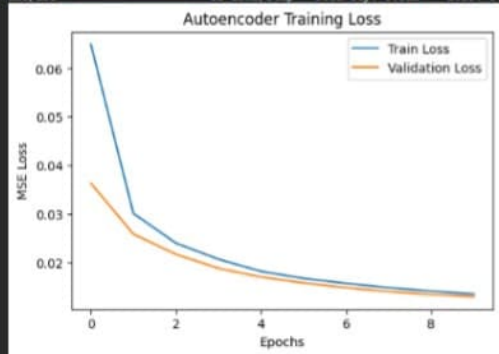
Epoch 6/10
235/235 — 5s 19ms/step - accuracy: 0.0090 - loss: 0.0170 - val_accuracy: 0.0106 - val_loss: 0.0157

Epoch 7/10
235/235 — 4s 18ms/step - accuracy: 0.0105 - loss: 0.0159 - val_accuracy: 0.0113 - val_loss: 0.0148

Epoch 8/10
235/235 — 4s 16ms/step - accuracy: 0.0105 - loss: 0.0149 - val_accuracy: 0.0113 - val_loss: 0.0140

Epoch 9/10
235/235 — 4s 18ms/step - accuracy: 0.0121 - loss: 0.0141 - val_accuracy: 0.0127 - val_loss: 0.0134

Epoch 10/10
235/235 — 5s 19ms/step - accuracy: 0.0124 - loss: 0.0136 - val_accuracy: 0.0131 - val_loss: 0.0129



313/313 — 1s 2ms/step

