**PRACTICAL - 6**

**Aim:**

**Denoising  
Images Using Autoencoders**  
  
In this task, you will implement a denoising autoencoder to  
clean noisy images, using the MNIST Handwritten Digits dataset. You will  
explore the encoding, decoding, and reconstruction processes in neural networks  
and understand how autoencoders relate to PCA. This experiment will also  
highlight the role of regularization and noise handling in autoencoders. Steps  
to Follow  
  
1.      Set Up the  
Dataset:  
  
a.       Load and  
preprocess the MNIST dataset.  
  
b.      Introduce  
noise to the images by adding random Gaussian or salt-and-pepper noise. Retain  
the original images as clean versions for comparison.  
  
2.      Design the  
Autoencoder:  
  
a.       Build a  
neural network architecture with an encoder that compresses the input images  
into a lower-dimensional latent space and a decoder that reconstructs the  
images from the latent representation.  
  
b.      Use  
non-linear activation functions like ReLU or sigmoid for the layers.  
  
Train the Autoencoder:  
a.       Train the  
autoencoder with noisy images as input and clean images as output.  
  
b.      Use a  
suitable loss function, such as Mean Squared Error (MSE), to measure the  
difference between the reconstructed and original images.  
  
c.       Monitor  
training loss to ensure the model is learning to remove noise effectively.  
  
4.      Test the  
Autoencoder:  
  
a.       Evaluate the  
model on a separate test set of noisy images and compare the reconstructed  
images with their clean counterparts.  
  
5.      Analyze  
Regularization:  
  
a.       Experiment  
with adding regularization techniques, such as L1/L2 penalties or Dropout, to  
the autoencoder  
  
b.      Observe their  
impact on the reconstruction quality and model generalization.  
  
6.      Visualize  
Results:  
  
a.       Display  
side-by-side comparisons of noisy, clean, and reconstructed images.  
  
b.      Plot the  
latent space representation to understand how the autoencoder encodes the input  
data.  
  
7.      Summarize  
Insights: Reflect on how denoising autoencoders handle noise effectively and  
improve image quality. Discuss the differences between autoencoders and PCA,  
emphasizing the power of neural networks in capturing non-linear features.

**Code:**

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import matplotlib.pyplot as plt

# 1. Load and preprocess the MNIST dataset

(x\_train, \_), (x\_test, \_) = keras.datasets.mnist.load\_data()

# Normalize the images to range [0,1] and add a channel dimension

x\_train = x\_train.astype("float32") / 255.0

x\_test = x\_test.astype("float32") / 255.0

x\_train = np.expand\_dims(x\_train, axis=-1)  # Shape: (60000, 28, 28, 1)

x\_test = np.expand\_dims(x\_test, axis=-1)    # Shape: (10000, 28, 28, 1)

# 2. Introduce noise (Gaussian noise)

noise\_factor = 0.4

x\_train\_noisy = x\_train + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_train.shape)

x\_test\_noisy = x\_test + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_test.shape)

# Clip values to stay within [0,1]

x\_train\_noisy = np.clip(x\_train\_noisy, 0.0, 1.0)

x\_test\_noisy = np.clip(x\_test\_noisy, 0.0, 1.0)

# 3. Build the Autoencoder Model

## Encoder

input\_img = keras.Input(shape=(28, 28, 1))

x = layers.Conv2D(32, (3, 3), activation="relu", padding="same")(input\_img)

x = layers.MaxPooling2D((2, 2), padding="same")(x)

x = layers.Conv2D(64, (3, 3), activation="relu", padding="same")(x)

x = layers.MaxPooling2D((2, 2), padding="same")(x)

encoded = layers.Conv2D(128, (3, 3), activation="relu", padding="same")(x)  # Latent Space

## Decoder

x = layers.Conv2DTranspose(64, (3, 3), activation="relu", padding="same")(encoded)

x = layers.UpSampling2D((2, 2))(x)

x = layers.Conv2DTranspose(32, (3, 3), activation="relu", padding="same")(x)

x = layers.UpSampling2D((2, 2))(x)

decoded = layers.Conv2DTranspose(1, (3, 3), activation="sigmoid", padding="same")(x)  # Output layer

# Define the Autoencoder

autoencoder = keras.Model(input\_img, decoded)

autoencoder.compile(optimizer="adam", loss="mse")  # Mean Squared Error Loss

# 4. Train the Autoencoder

autoencoder.fit(

    x\_train\_noisy, x\_train,

    epochs=3,

    batch\_size=128,

    validation\_data=(x\_test\_noisy, x\_test)

)

# 5. Test the Autoencoder (Reconstruct Images)

decoded\_images = autoencoder.predict(x\_test\_noisy)

# 6. Visualization of Results

n = 10  # Number of images to display

plt.figure(figsize=(20, 6))

for i in range(n):

    # Noisy Input

    plt.subplot(3, n, i + 1)

    plt.imshow(x\_test\_noisy[i].reshape(28, 28), cmap="gray")

    plt.axis("off")

    if i == 0:

        plt.title("Noisy Input")

    # Denoised Output

    plt.subplot(3, n, i + 1 + n)

    plt.imshow(decoded\_images[i].reshape(28, 28), cmap="gray")

    plt.axis("off")

    if i == 0:

        plt.title("Denoised Output")

    # Original Image

    plt.subplot(3, n, i + 1 + 2 \* n)

    plt.imshow(x\_test[i].reshape(28, 28), cmap="gray")

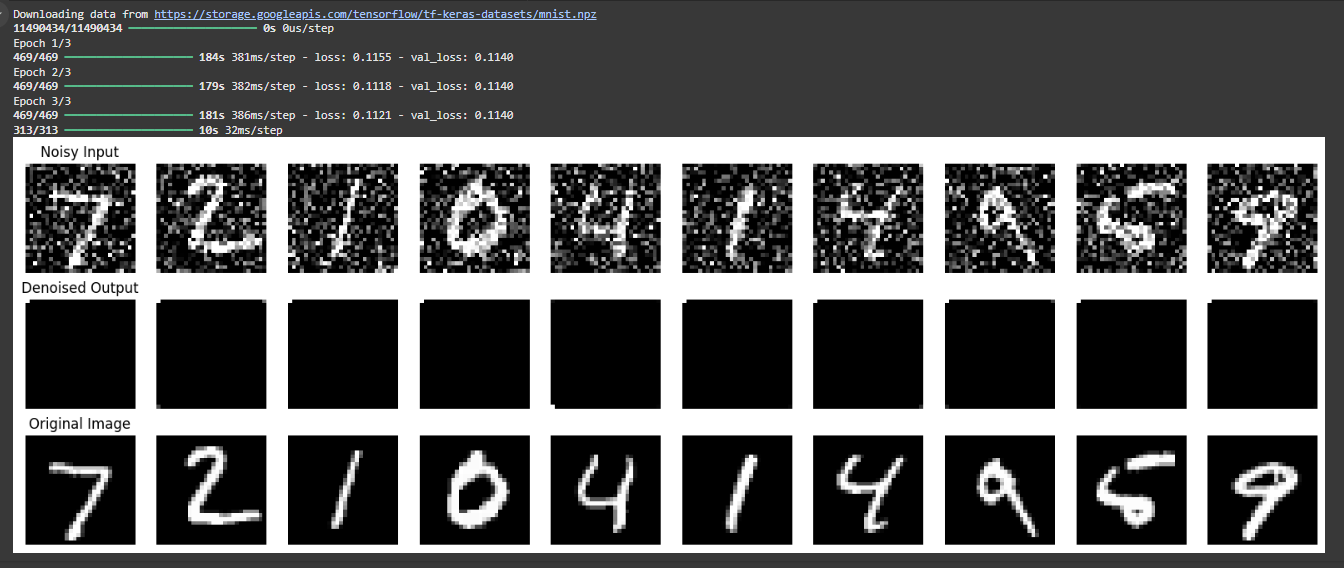
    plt.axis("off")

    if i == 0:

        plt.title("Original Image")

plt.show()

**Output:**

****

**Learning Outcome:**

Through this task, I have gained hands-on experience in applying Principal Component Analysis (PCA) to reduce the dimensionality of high-dimensional data while preserving its essential features. By working with the Wine dataset, I learned how to preprocess data by standardizing it to ensure consistency in scale. I applied PCA to extract the principal components and reduced the data to two dimensions, which I visualized in a 2D scatter plot. I was able to examine the explained variance ratio to understand how much variance each principal component captures and observed the effectiveness of dimensionality reduction in maintaining key patterns in the data. Additionally, I reconstructed the data from the reduced dimensions, quantified the reconstruction error, and visualized the original versus reduced data, highlighting the trade-off between reducing complexity and preserving important information. This task reinforced my understanding of PCA’s ability to simplify complex datasets while retaining critical insights, and it demonstrated how PCA can be useful in exploratory data analysis and classification.