**Lab Exercise 11- Autoencoder Decoder with Custom Image Dataset in Generative AI**

**Objective**

This exercise demonstrates the use of autoencoders for learning compressed representations of images and reconstructing them using a custom image dataset. You will focus on how the **decoder** part of the autoencoder reconstructs images from latent space.

**Exercise Overview**

1. Use a different image dataset (e.g., CIFAR-10 or any custom dataset).
2. Implement the encoder and decoder parts of the autoencoder.
3. Train the autoencoder on the dataset.
4. Visualize the reconstructed images.

**Prerequisites**

1. Install Python (3.7 or later).
2. Install required libraries:

pip install tensorflow matplotlib

1. Download or prepare a custom image dataset if not using CIFAR-10.

**Tasks**

**Task 1: Import Libraries**

Start by importing the required libraries.

**Task 2: Load and Preprocess the Dataset**

Load the CIFAR-10 dataset or any custom image dataset. Normalize the images and prepare them for training.

**Task 3: Define Autoencoder Architecture**

Implement an encoder for compressing the images and a decoder for reconstructing them.

**Task 4: Train and Test the Autoencoder**

Train the model and evaluate its performance by reconstructing the images.

**Task 5: Visualize Original and Reconstructed Images**

Compare the input images and the output from the decoder.

**Code Implementation**

Here’s an example implementation using the CIFAR-10 dataset:

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, Flatten, Reshape, Conv2D, Conv2DTranspose

from tensorflow.keras.datasets import cifar10

# Step 1: Load and preprocess the dataset

(x\_train, \_), (x\_test, \_) = cifar10.load\_data()

# Normalize pixel values to [0, 1]

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

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

# Shape information

input\_shape = x\_train.shape[1:] # (32, 32, 3)

# Step 2: Define the Autoencoder

# Encoder

input\_img = Input(shape=input\_shape)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input\_img)

x = Conv2D(64, (3, 3), activation='relu', padding='same', strides=(2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same', strides=(2, 2))(x)

latent = Flatten()(x)

# Decoder

x = Reshape((8, 8, 128))(latent)

x = Conv2DTranspose(128, (3, 3), activation='relu', padding='same')(x)

x = Conv2DTranspose(64, (3, 3), activation='relu', padding='same', strides=(2, 2))(x)

decoded = Conv2DTranspose(3, (3, 3), activation='sigmoid', padding='same', strides=(2, 2))(x)

# Autoencoder

autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Step 3: Train the Autoencoder

autoencoder.fit(

x\_train, x\_train,

epochs=20,

batch\_size=256,

shuffle=True,

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

)

# Step 4: Test and Visualize the Results

# Reconstruct the images

decoded\_imgs = autoencoder.predict(x\_test)

# Display original and reconstructed images

n = 10 # Number of images to display

plt.figure(figsize=(20, 4))

for i in range(n):

# Original

ax = plt.subplot(2, n, i + 1)

plt.imshow(x\_test[i])

plt.title("Original")

plt.axis('off')

# Reconstructed

ax = plt.subplot(2, n, i + n + 1)

plt.imshow(decoded\_imgs[i])

plt.title("Reconstructed")

plt.axis('off')

plt.show()

**Expected Output**

1. **Training Process**: The loss should decrease as the autoencoder learns to reconstruct images.
2. **Visualization**:
   * **Top row**: Original images from the test dataset.
   * **Bottom row**: Reconstructed images generated by the decoder.