**Practical 8**

**Encoding and decoding using deep autoencoder.**

**8**

**Aim:** **Performing encoding and decoding of images using deep autoencoder.**

**Description:**

**Image Compression and Reconstruction with Deep Autoencoders**

Deep autoencoders offer a powerful technique for encoding and decoding images. Here's a breakdown of the process in 5 points:

1. Network Architecture: A deep autoencoder consists of two main parts: an encoder and a decoder. The encoder progressively reduces the image's dimensionality, capturing its essential features in a compressed "latent space."
2. Learning Compressive Representation: During training, the encoder learns to represent the image in this lower-dimensional space. This compressed representation discards redundant information while retaining key details.
3. Decoding for Reconstruction: The decoder then takes this compressed representation and attempts to reconstruct the original image.
4. Training and Loss Function: The autoencoder is trained to minimize the difference between the original image and the reconstructed image. This ensures the encoder captures the most important information for faithful reconstruction.
5. Applications: Deep autoencoders have various applications in image processing. They can be used for image compression, denoising (removing noise from images), and anomaly detection (identifying unusual patterns in images).

**Code:**

# 8. Performing encoding and decoding of images using deep autoencoder.

import keras

from keras import layers

from keras.datasets import mnist

import numpy as np

encoding\_dim = 32

# this is our input image

input\_img = keras.Input(shape=(784,))

# "encoded" is the encoded representation of the input

encoded = layers.Dense(encoding\_dim, activation='relu')(input\_img)

# "decoded" is the lossy reconstruction of the input

decoded = layers.Dense(784, activation='sigmoid')(encoded)

# creating autoencoder model

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

# create the encoder model

encoder = keras.Model(input\_img, encoded)

encoded\_input = keras.Input(shape=(encoding\_dim,))

# Retrieve the last layer of the autoencoder model

decoder\_layer = autoencoder.layers[-1]

# create the decoder model

decoder = keras.Model(encoded\_input, decoder\_layer(encoded\_input))

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

# scale and make train and test dataset

(X\_train, \_), (X\_test, \_) = mnist.load\_data()

X\_train = X\_train.astype('float32') / 255.

X\_test = X\_test.astype('float32') / 255.

X\_train = X\_train.reshape((len(X\_train), np.prod(X\_train.shape[1:])))

X\_test = X\_test.reshape((len(X\_test), np.prod(X\_test.shape[1:])))

print(X\_train.shape)

print(X\_test.shape)

# train autoencoder with training dataset

autoencoder.fit(X\_train, X\_train,

epochs=50,

batch\_size=256,

shuffle=True,

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

encoded\_imgs = encoder.predict(X\_test)

decoded\_imgs = decoder.predict(encoded\_imgs)

import matplotlib.pyplot as plt

n = 10 # How many digits we will display

plt.figure(figsize=(40, 4))

for i in range(10):

# display original

ax = plt.subplot(3, 20, i + 1)

plt.imshow(X\_test[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# display encoded image

ax = plt.subplot(3, 20, i + 1 + 20)

plt.imshow(encoded\_imgs[i].reshape(8, 4))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# display reconstruction

ax = plt.subplot(3, 20, 2 \* 20 + i + 1)

plt.imshow(decoded\_imgs[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()

**Output:**

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A group of black squares with white letters

Description automatically generated

**Learning:**

1. **Deep Autoencoder Architecture**: The code defines a deep autoencoder architecture using the Keras library, which consists of an encoder and a decoder. The encoder compresses the input images into a lower-dimensional latent space representation, while the decoder reconstructs the original images from this representation.
2. **Model Compilation and Training**: The autoencoder model is compiled with the Adam optimizer and binary cross-entropy loss function. It is then trained on the MNIST dataset, which consists of grayscale images of handwritten digits. Training is performed for 50 epochs with a batch size of 256, and the training and validation data are shuffled during training.
3. **Data Preprocessing**: The MNIST dataset is loaded and preprocessed. The pixel values of the images are scaled to the range [0, 1] by dividing by 255. Additionally, the shape of the input images is reshaped to a vector of length 784 before training.
4. **Encoder and Decoder Models**: Separate models for the encoder and decoder are created using the trained autoencoder model. These models allow for the encoding and decoding of images independently. The encoder model takes input images and outputs their encoded representations, while the decoder model takes encoded representations and reconstructs the original images.
5. **Visualization**: The code includes visualization of the original images, encoded representations, and reconstructed images for a sample of the test data. Matplotlib is used to display these images in a grid format, with each row showing the original image, its encoded representation, and the reconstructed image.