Ex No: 7

BUILD AUTOENCODERS WITH KERAS/TENSORFLOW

Aim:

To build autoencoders with Keras/TensorFlow.

Procedure:

1. Import the necessary libraries for numerical operations, deep learning, and visualization.
2. Load the MNIST dataset and split it into training and testing sets.
3. Normalize the pixel values of the images to the range [0, 1] to enhance model performance.
4. Reshape the dataset to include a channel dimension for compatibility with convolutional layers.
5. Define the autoencoder architecture, starting with an encoder that compresses the input images.
6. Add a decoder to reconstruct the compressed representations back to the original image dimensions.
7. Combine the encoder and decoder into a single model, compile it with Adam optimizer and binary cross-entropy loss.
8. Train the autoencoder using the training dataset, validating it with the test dataset.
9. Use the trained autoencoder to reconstruct the images from the test dataset.
10. Visualize the original and reconstructed images side by side for comparison.

Code:

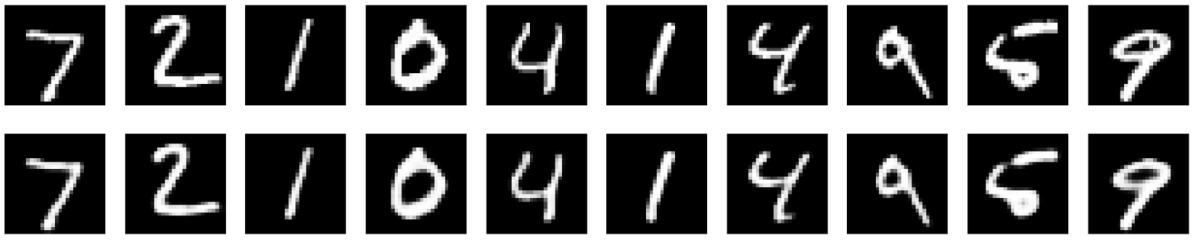
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
# Load the MNIST dataset
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
# Normalize the data
x_train = x_train.astype("float32") / 255.0
```

```
x_test = x_test.astype("float32") / 255.0
# Reshape the data to (num_samples, 28, 28, 1)
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
# Build the autoencoder
input_img = layers.Input(shape=(28, 28, 1))
# Encoder
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)
# Decoder
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
# Compile the autoencoder
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
# Train the autoencoder
autoencoder.fit(x_train, x_train, epochs=10, batch_size=256, shuffle=True,
validation_data=(x_test, x_test))
# Reconstruct the images
decoded_imgs = autoencoder.predict(x_test)
# Plot original and reconstructed images
n = 10  # Number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
```

```
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()
```

Output:

```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 16s 54ms/step - loss: 0.3011 - val_loss: 0.0915
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 12s 53ms/step - loss: 0.0896 - val_loss: 0.0817
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 16s 68ms/step - loss: 0.0815 - val_loss: 0.0774
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 15s 65ms/step - loss: 0.0778 - val_loss: 0.0754
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 17s 73ms/step - loss: 0.0759 - val_loss: 0.0741
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 16s 70ms/step - loss: 0.0746 - val_loss: 0.0733
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 18s 74ms/step - loss: 0.0739 - val_loss: 0.0727
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 17s 71ms/step - loss: 0.0732 - val_loss: 0.0721
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 17s 73ms/step - loss: 0.0725 - val_loss: 0.0720
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 17s 74ms/step - loss: 0.0723 - val_loss: 0.0713
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step
```



Result:

Thus the program to build autoencoders with Keras/TensorFlow has been executed successfully.