ABHINAV SAXENA 0176CD221007

Implementation of Autoencoders

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score,
recall_score
from sklearn.model_selection import train_test_split
from keras import layers, losses
from keras.datasets import mnist
from keras.models import Model
```

Load the MNIST dataset

```
# Loading the MNIST dataset and extracting training and testing data
(x_train, _), (x_test, _) = mnist.load_data()

# Normalizing pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Displaying the shapes of the training and testing datasets
print("Shape of the training data:", x_train.shape)
print("Shape of the testing data:", x_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Shape of the training data: (60000, 28, 28)
Shape of the testing data: (10000, 28, 28)
```

Definition of the Autoencoder model as a subclass of the TensorFlow Model class

```
# Definition of the Autoencoder model as a subclass of the TensorFlow
Model class

class SimpleAutoencoder(Model):
      def __init__(self,latent_dimensions , data_shape):
            super(SimpleAutoencoder, self).__init__()
            self.latent_dimensions = latent_dimensions
            self.data_shape = data_shape

            # Encoder architecture using a Sequential model
            self.encoder = tf.keras.Sequential([
                  layers.Flatten(),
                  layers.Dense(latent_dimensions, activation='relu'),
            ])
```

```
            # Decoder architecture using another Sequential model
            self.decoder = tf.keras.Sequential([
                  layers.Dense(tf.math.reduce_prod(data_shape),
activation='sigmoid'),
                  layers.Reshape(data_shape)
            ])

      # Forward pass method defining the encoding and decoding steps
      def call(self, input_data):
            encoded_data = self.encoder(input_data)
            decoded_data = self.decoder(encoded_data)
            return decoded_data

# Extracting shape information from the testing dataset
input_data_shape = x_test.shape[1:]

# Specifying the dimensionality of the latent space
latent_dimensions = 64

# Creating an instance of the SimpleAutoencoder model
simple_autoencoder = SimpleAutoencoder(latent_dimensions,
input_data_shape)
```

Compile and Fit Autoencoder

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist

# Load MNIST Dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize to
[0,1]

# Reshape for Fully Connected Layers
x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)

# Autoencoder Model
latent_dim = 64  # Size of compressed representation

# Encoder
input_img = Input(shape=(28 * 28,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(latent_dim, activation='relu')(encoded)
```

```
# Decoder
decoded = Dense(128, activation='relu')(encoded)
decoded = Dense(28 * 28, activation='sigmoid')(decoded)  # Output in
[0,1]

# Autoencoder Model
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse')

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

# Generate Reconstructions
reconstructed_imgs = autoencoder.predict(x_test[:10])

# Plot Original vs. Reconstructed Images
fig, axes = plt.subplots(2, 10, figsize=(10, 2))
for i in range(10):
    axes[0, i].imshow(x_test[i].reshape(28, 28), cmap='gray')
    axes[0, i].axis('off')
    axes[1, i].imshow(reconstructed_imgs[i].reshape(28, 28),
cmap='gray')
    axes[1, i].axis('off')
plt.show()
```

```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.0880 - val_loss:
0.0310
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0275 - val_loss:
0.0188
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0180 - val_loss:
0.0143
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0141 - val_loss:
0.0119
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.0120 - val_loss:
0.0107
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0109 - val_loss:
0.0099
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0101 - val_loss:
0.0092
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.0094 - val_loss:
0.0086
```

```
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.0088 - val_loss:
0.0081
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.0083 - val_loss:
0.0079
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
```