**1.Create a neural network that can analyze movie reviews and classify the sentiment expressed in each review as positive, negative, or neutral.**

**PROGRAM**

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import to\_categorical

# Load the dataset

df = pd.read\_csv('movie.csv')

# Preprocessing the dataset

reviews = df['review'].values

sentiments = df['sentiment'].values

# Tokenize and Pad sequences

tokenizer = Tokenizer(num\_words=5000, oov\_token="<OOV>") # Reduce vocabulary size

tokenizer.fit\_on\_texts(reviews)

sequences = tokenizer.texts\_to\_sequences(reviews)

padded\_sequences = pad\_sequences(sequences, maxlen=10) # Reduce max sequence length

# Convert labels to numerical values

label\_map = {'positive': 0, 'neutral': 1, 'negative': 2}

numerical\_labels = [label\_map[label] for label in sentiments]

categorical\_labels = to\_categorical(numerical\_labels, num\_classes=3)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(padded\_sequences, categorical\_labels, test\_size=0.2, random\_state=42)

# Build a simpler Neural Network Model

model = tf.keras.Sequential([

tf.keras.layers.Embedding(input\_dim=5000, output\_dim=64, input\_length=10), # Reduced embedding size and input length

tf.keras.layers.GlobalAveragePooling1D(), # Simplified layer instead of LSTM

tf.keras.layers.Dense(16, activation='relu'), # Reduced number of neurons

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(3, activation='softmax') # 3 classes: positive, neutral, negative

])

# Compile the Model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model for fewer epochs

print("Training started...")

model.fit(X\_train, y\_train, epochs=3, batch\_size=32, validation\_data=(X\_test, y\_test)) # Reduced epochs

print("Training completed!")

# Save the model for future use

model.save('simplified\_sentiment\_model.h5')

print("Model saved as 'simplified\_sentiment\_model.h5'")

# Function to predict sentiment for a new sentence

def predict\_sentiment(sentence):

# Tokenize and pad the new sentence

sequence = tokenizer.texts\_to\_sequences([sentence])

padded\_sequence = pad\_sequences(sequence, maxlen=10) # Keep maxlen consistent

prediction = model.predict(padded\_sequence)

sentiment\_labels = ['positive', 'neutral', 'negative']

sentiment = sentiment\_labels[np.argmax(prediction)]

return sentiment

# Test the function with a new sentence

new\_sentence = "One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked"

print(f"Sentiment: {predict\_sentiment(new\_sentence)}")

**OUTPUT**

Training started...

Epoch 1/3

warnings.warn(

**1250/1250** ━━━━━━━━━━━━━━━━━━━━ **8s** 5ms/step - accuracy: 0.6168 - loss: 0.7231 - val\_accuracy: 0.7276 - val\_loss: 0.5275

Epoch 2/3

**1250/1250** ━━━━━━━━━━━━━━━━━━━━ **7s** 5ms/step - accuracy: 0.7536 - loss: 0.5040 - val\_accuracy: 0.7320 - val\_loss: 0.5206

Epoch 3/3

**1250/1250** ━━━━━━━━━━━━━━━━━━━━ **7s** 6ms/step - accuracy: 0.7731 - loss: 0.4721 - val\_accuracy: 0.7308 - val\_loss: 0.5229

Training completed!

Model saved as 'simplified\_sentiment\_model.h5'

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 117ms/step

Sentiment: positive

**2. Develop an autoencoder to identify anomalies in time-series data. The model should be able to reconstruct normal patterns accurately but fail to reconstruct anomalies, highlighting them for further investigation.**

**PROGRAM**

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D

from tensorflow.keras.models import Model

from tensorflow.keras.datasets import mnist

# Load the MNIST dataset

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

# Normalize and reshape the data

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

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

x\_train = np.reshape(x\_train, (len(x\_train), 28, 28, 1)) # (60000, 28, 28, 1)

x\_test = np.reshape(x\_test, (len(x\_test), 28, 28, 1)) # (10000, 28, 28, 1)

# Create noisy images by adding random noise to the original images

noise\_factor = 0.5

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 be in the range [0, 1]

x\_train\_noisy = np.clip(x\_train\_noisy, 0., 1.)

x\_test\_noisy = np.clip(x\_test\_noisy, 0., 1.)

# Visualize some of the noisy images

n = 10

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

for i in range(n):

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

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

plt.title("Noisy Image")

plt.axis('off')

plt.show()

# Build the Autoencoder Model

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

# Encoder

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

x = MaxPooling2D((2, 2), padding='same')(x)

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

encoded = MaxPooling2D((2, 2), padding='same')(x)

# Decoder

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)

x = UpSampling2D((2, 2))(x)

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

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

# Autoencoder model

autoencoder = Model(input\_img, decoded)

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

# Summary of the model

autoencoder.summary()

# Train the autoencoder

autoencoder.fit(x\_train\_noisy, x\_train,

epochs=10,

batch\_size=128,

shuffle=True,

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

# Predict on the test data

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

# Visualize the results: original, noisy, and denoised images

n = 10

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

for i in range(n):

# Display original

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

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

plt.title("Original Image")

plt.axis('off')

# Display noisy

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

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

plt.title("Noisy Image")

plt.axis('off')

# Display reconstructed (denoised)

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

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

plt.title("Denoised Image")

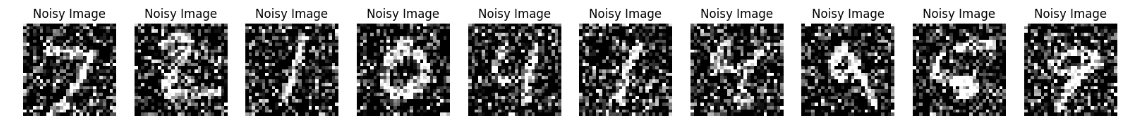
plt.axis('off')

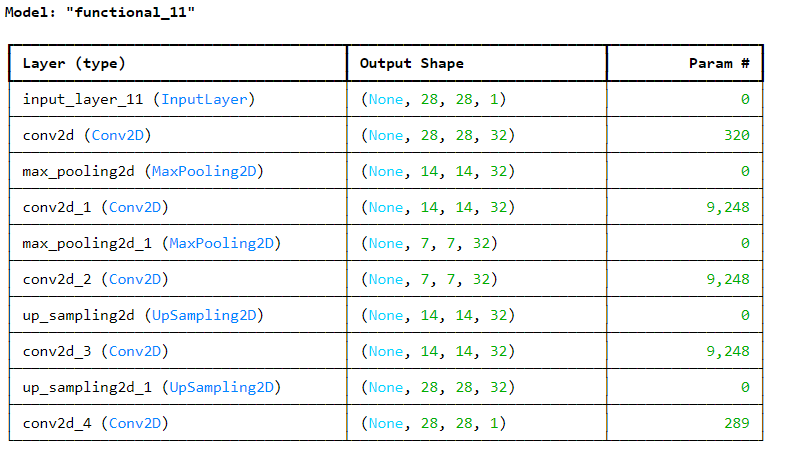
plt.show()

**OUTPUT**

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step

****

****

Total params: 28,353 (110.75 KB)

Trainable params: 28,353 (110.75 KB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 35s 68ms/step - loss: 0.2514 - val\_loss: 0.1168

Epoch 2/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 32s 68ms/step - loss: 0.1157 - val\_loss: 0.1092

Epoch 3/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 31s 67ms/step - loss: 0.1094 - val\_loss: 0.1056

Epoch 4/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 30s 64ms/step - loss: 0.1061 - val\_loss: 0.1032

Epoch 5/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 31s 65ms/step - loss: 0.1037 - val\_loss: 0.1017

Epoch 6/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 32s 69ms/step - loss: 0.1021 - val\_loss: 0.1005

Epoch 7/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 42s 70ms/step - loss: 0.1010 - val\_loss: 0.0997

Epoch 8/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 40s 68ms/step - loss: 0.1002 - val\_loss: 0.0991

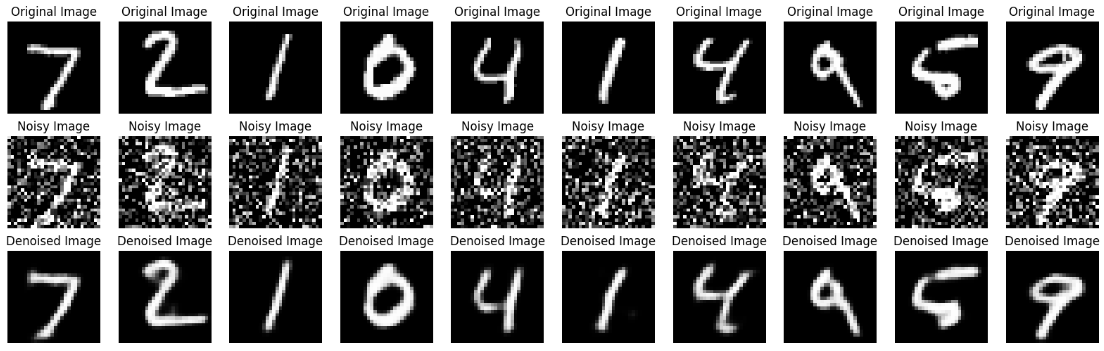
Epoch 9/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 32s 68ms/step - loss: 0.0996 - val\_loss: 0.0982

Epoch 10/10

469/469 ━━━━━━━━━━━━━━━━━━━━ 31s 66ms/step - loss: 0.0988 - val\_loss: 0.0979

313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step

****