**Exercise:9**

**Accuracy Based:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import imdb

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Embedding, SimpleRNN

# Setting parameters

max\_features = 10000  # Number of words to consider as features

maxlen = 500  # Cuts off texts after this number of words (among the max\_features most common words)

batch\_size = 32

# Loading data

print('Loading data...')

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

print(len(x\_train), 'train sequences')

print(len(x\_test), 'test sequences')

# Padding sequences to ensure uniform length

print('Pad sequences (samples x time)')

x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen)

print('x\_train shape:', x\_train.shape)

print('x\_test shape:', x\_test.shape)

# Building the model

model = Sequential()

model.add(Embedding(max\_features, 32))

model.add(SimpleRNN(32))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['acc'])

print(model.summary())

# Training the model

print('Training...')

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=batch\_size, validation\_split=0.2)

# Evaluating the model

print('Evaluating...')

loss, accuracy = model.evaluate(x\_test, y\_test)

print('Test Loss:', loss)

print('Test Accuracy:', accuracy)

OR

**Input Based:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import imdb

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Embedding, SimpleRNN

# Setting parameters

max\_features = 10000  # Number of words to consider as features

maxlen = 500  # Cuts off texts after this number of words (among the max\_features most common words)

batch\_size = 32

# Loading data

print('Loading data...')

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

print(len(x\_train), 'train sequences')

print(len(x\_test), 'test sequences')

# Padding sequences to ensure uniform length

print('Pad sequences (samples x time)')

x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen)

print('x\_train shape:', x\_train.shape)

print('x\_test shape:', x\_test.shape)

# Building the model

model = Sequential()

model.add(Embedding(max\_features, 32))

model.add(SimpleRNN(32))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['acc'])

print(model.summary())

# Training the model

print('Training...')

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=batch\_size, validation\_split=0.2)

# Function to preprocess user input

def preprocess\_input(text):

    word\_to\_index = imdb.get\_word\_index()

    words = text.lower().split()

    filtered\_words = [word\_to\_index[word] if word in word\_to\_index and word\_to\_index[word] < max\_features else 0 for word in words]

    padded\_sequence = pad\_sequences([filtered\_words], maxlen=maxlen)

    return padded\_sequence

# Function to predict sentiment for user input

def predict\_sentiment(text):

    preprocessed\_text = preprocess\_input(text)

    prediction = model.predict(preprocessed\_text)

    return prediction[0][0]

# Allow user input for sentiment analysis

while True:

    user\_input = input("Enter a movie review (type 'exit' to quit): ")

    if user\_input.lower() == 'exit':

        break

    else:

        sentiment = predict\_sentiment(user\_input)

        if sentiment > 0.5:

            print("Positive Sentiment")

        else:

            print("Negative Sentiment")

**Exercise:10**

**Simple:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, LSTM, RepeatVector

from tensorflow.keras.callbacks import ModelCheckpoint

# Generate random data for demonstration

data = np.random.rand(1000, 10, 1)  # Example data: 1000 sequences of length 10 with 1 feature

# Define model architecture

latent\_dim = 2  # Dimensionality of the latent space

inputs = Input(shape=(10, 1))

encoded = LSTM(4)(inputs)

encoded = RepeatVector(10)(encoded)  # Repeat the encoded representation 10 times

decoded = LSTM(4, return\_sequences=True)(encoded)

decoded = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(1))(decoded)

# Build the autoencoder model

autoencoder = Model(inputs, decoded)

# Compile the model

autoencoder.compile(optimizer='adam', loss='mse')

# Print model summary

autoencoder.summary()

# Train the model

autoencoder.fit(data, data, epochs=50, batch\_size=32, validation\_split=0.2)

# After training, you can use the encoder and decoder separately if needed

encoder = Model(inputs, encoded)

encoded\_input = Input(shape=(latent\_dim, 4))

decoder\_layer = autoencoder.layers[-2](encoded\_input)

decoder\_layer = autoencoder.layers[-1](decoder\_layer)

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

OR

**Input Based:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, LSTM, RepeatVector

from tensorflow.keras.callbacks import ModelCheckpoint

# Define model architecture

latent\_dim = 2  # Dimensionality of the latent space

inputs = Input(shape=(10, 1))

encoded = LSTM(4)(inputs)

encoded = RepeatVector(10)(encoded)  # Repeat the encoded representation 10 times

decoded = LSTM(4, return\_sequences=True)(encoded)

decoded = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(1))(decoded)

# Build the autoencoder model

autoencoder = Model(inputs, decoded)

# Compile the model

autoencoder.compile(optimizer='adam', loss='mse')

# Print model summary

autoencoder.summary()

# Allow user input for sequences

while True:

    user\_input = input("Enter a sequence of 10 numbers separated by spaces (type 'exit' to quit): ")

    if user\_input.lower() == 'exit':

        break

    else:

        # Convert user input to a sequence of floats

        sequence = [float(x) for x in user\_input.split()]

        if len(sequence) != 10:

            print("Please enter exactly 10 numbers.")

            continue

        sequence = np.array(sequence).reshape(1, 10, 1)  # Reshape to match model input shape

        # Encode the sequence

        encoded\_sequence = autoencoder.predict(sequence)

        # Decode the encoded sequence

        decoded\_sequence = autoencoder.predict(encoded\_sequence)

        print("Original Sequence:", sequence)

        print("Encoded Sequence:", encoded\_sequence)

        print("Decoded Sequence:", decoded\_sequence)

**Exercise:11**

#!pip install tensorflow==2.8

import tensorflow as tf

from tensorflow import keras

import numpy as np

# Load MNIST dataset

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

X\_train = X\_train / 127.5 - 1.0  # Rescale images to [-1, 1]

X\_train = np.expand\_dims(X\_train, axis=-1)

# Generator model

generator = keras.Sequential([

    keras.layers.Dense(7 \* 7 \* 128, input\_shape=(100,)),

    keras.layers.Reshape((7, 7, 128)),

    keras.layers.Conv2DTranspose(64, kernel\_size=3, strides=2, padding='same'),

    keras.layers.LeakyReLU(alpha=0.2),

    keras.layers.Conv2DTranspose(1, kernel\_size=3, strides=2, padding='same', activation='tanh')

])

# Discriminator model

discriminator = keras.Sequential([

    keras.layers.Conv2D(64, kernel\_size=3, strides=2, padding='same', input\_shape=(28, 28, 1)),

    keras.layers.LeakyReLU(alpha=0.2),

    keras.layers.Conv2D(128, kernel\_size=3, strides=2, padding='same'),

    keras.layers.LeakyReLU(alpha=0.2),

    keras.layers.Flatten(),

    keras.layers.Dense(1, activation='sigmoid')

])

# Compile discriminator

discriminator.compile(loss='binary\_crossentropy', optimizer=keras.optimizers.Adam(learning\_rate=0.0002), metrics=['accuracy'])

# Freeze discriminator's weights during GAN training

discriminator.trainable = False

# GAN model

gan\_input = keras.Input(shape=(100,))

generated\_image = generator(gan\_input)

gan\_output = discriminator(generated\_image)

gan = keras.Model(gan\_input, gan\_output)

# Compile GAN

gan.compile(loss='binary\_crossentropy', optimizer=keras.optimizers.Adam(learning\_rate=0.0002))

# Training parameters

batch\_size = 64

epochs = 10

sample\_interval = 1000

# Training loop

for epoch in range(epochs):

    # Train discriminator

    idx = np.random.randint(0, X\_train.shape[0], batch\_size)

    real\_images = X\_train[idx]

    noise = np.random.normal(0, 1, (batch\_size, 100))

    fake\_images = generator.predict(noise)

    # Label real and fake images

    real\_labels = np.ones((batch\_size, 1))

    fake\_labels = np.zeros((batch\_size, 1))

    # Train discriminator

    d\_loss\_real = discriminator.train\_on\_batch(real\_images, real\_labels)

    d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, fake\_labels)

    d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

    # Train generator

    noise = np.random.normal(0, 1, (batch\_size, 100))

    g\_loss = gan.train\_on\_batch(noise, real\_labels)

    # Print progress

    if epoch % sample\_interval == 0:

        print(f'Epoch {epoch}, D Loss: {d\_loss[0]}, G Loss: {g\_loss}')

        # Print discriminator accuracy

        metrics\_names = discriminator.metrics\_names

        accuracy\_index = metrics\_names.index('accuracy')

        \_, accuracy = discriminator.evaluate(np.concatenate([real\_images, fake\_images]), np.concatenate([real\_labels, fake\_labels]), verbose=0)

        print(f"Discriminator Accuracy: {accuracy:.4f}")

**Exercise:12**

import tensorflow as tf #!pip install tensorflow==2.8

from tensorflow.keras.applications import VGG16

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os  # Import the os module !pip install os

from google.colab import drive

drive.mount('/content/drive')

data\_dir = '/content/drive/MyDrive/Collab'

# Load the pre-trained VGG16 model (without the fully connected layers)

vgg\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Freeze the weights of the pre-trained layers so they are not updated during training

for layer in vgg\_model.layers:

    layer.trainable = False

# Create a new model

model = Sequential()

# Add the pre-trained VGG16 model

model.add(vgg\_model)

# Flatten the output of VGG16

model.add(Flatten())

# Add fully connected layers for classification

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

# Number of classes in your dataset

num\_classes = 2

# Output layer for multi-class classification

model.add(Dense(num\_classes, activation='softmax'))

# Compile the model

model.compile(optimizer=Adam(lr=1e-4), loss='categorical\_crossentropy', metrics=['accuracy'])

# Load and preprocess the data using ImageDataGenerator

train\_data\_dir = os.path.join(data\_dir, 'train')

validation\_data\_dir = os.path.join(data\_dir, 'validation')

train\_datagen = ImageDataGenerator(rescale=1./255)

test\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

    train\_data\_dir,

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical',  # Use 'categorical' for multi-class classification

    shuffle=True

)

validation\_generator = test\_datagen.flow\_from\_directory(

    validation\_data\_dir,

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical',  # Use 'categorical' for multi-class classification

    shuffle=False

)

# Class labels

class\_labels = train\_generator.class\_indices

print("Class labels:", class\_labels)

# Train the model

model.fit(

    train\_generator,

    steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

    epochs=10,  # Adjust the number of epochs as needed

    validation\_data=validation\_generator,

    validation\_steps=validation\_generator.samples // validation\_generator.batch\_size

)

# Evaluate the model on the validation data

validation\_loss, validation\_accuracy = model.evaluate(validation\_generator)

print("Validation Accuracy:", validation\_accuracy)

#https://www.kaggle.com/code/samarthsoni106/cat-and-dog-classification-tensorflow/input