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
```