Ex.No:1 **VECTOR ADDITION**

Aim

To implement Simple Vector Addition using Tensorflow.

Algorithm

Step1:Import the necessary package

Step2:Declare variables to get vector size and elements.

Step3:Check if both the size of vector are same

Step4:Get values from user for two vectors using loop

Step5:Convert the list to tensorflow vector

Step6:Add the vector elements

Step7:Display the result

Program

```
import tensorflow as tf
# Get the number of elements in the vectors from the user
num_elements1 = int(input("Enter the number of elements in the first vector: ))
num_elements2 = int(input("Enter the number of elements in the second vector: "))
# Check if the number of elements in the vectors is the same
if num_elements1 != num_elements2:
    print("Error: Vectors must have the same number of elements.")
else:
    # Initialize empty lists to store the vectors
    vector1 = []
    vector2 = []
    # Use a loop to get values for the first vector
    print("Enter values for the first vector:")
    for i in range(num_elements1):
        value = float(input(f"Element {i+1}: "))
        vector1.append(value)
    # Use another loop to get values for the second vector
```

```
print("Enter values for the second vector:")
for i in range(num_elements2):
    value = float(input(f"Element {i+1}: "))
    vector2.append(value)
# Convert the lists to TensorFlow tensors
tf_vector1 = tf.constant(vector1, dtype=tf.float32)
tf_vector2 = tf.constant(vector2, dtype=tf.float32)
# Perform vector addition using TensorFlow
result_vector = tf_vector1 + tf_vector2
print("Vector 1:", vector1)
print("Vector 2:", vector2)
print("Resulting Vector:", result_vector)
```

```
Enter the number of elements in the first vector: 3
Enter the number of elements in the second vector: 5
Error: Vectors must have the same number of elements.
Enter the number of elements in the first vector: 2
Enter the number of elements in the second vector: 2
Enter values for the first vector:
Element 1: 5
Element 2: 9
Enter values for the second vector:
Element 1: 34
Element 2: 78
Vector 1: [5.0, 9.0]
Vector 2: [34.0, 78.0]
Resulting Vector: tf.Tensor([39. 87.], shape=(2,), type=float32)
```

Result

Thus, the program has been executed successfully.

Ex.No:2 **SIMPLE REGRESSION**

Aim

To implement simple regression model using Keras

Algorithm

Step1:Import the necessary packages

Step2:Load the dataset

Step3:Preprocess the dataset for missing values ,split as train and test

Step4:Build the model with the hyperparameters

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Predict the values given by user .

Program

```
#import the necessary packages
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
#load the dataset
dataset = pd.read_csv('D:/SJIT/DL/LAB/er.csv')
print(dataset.head())
#feature selection for input and target
X=dataset['Age']
y=dataset['Price']
#split the x,y into training and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
#Building the model
```

model = keras.Sequential([keras.layers.Dense(units=1, input_shape=(1,), activation='linear')])

#compiling the Model

model.compile(optimizer='sgd', loss='mean_squared_error' , metrics='accuracy')

#Train the model with epochs and batch size

model.fit(X_train, y_train, epochs=10, verbose=1)

#Predict the test value on trained model

pre= model.predict(X_test)

#visualize the dataset along with regression line

plt.scatter(X_train, y_train, label='Training Data')

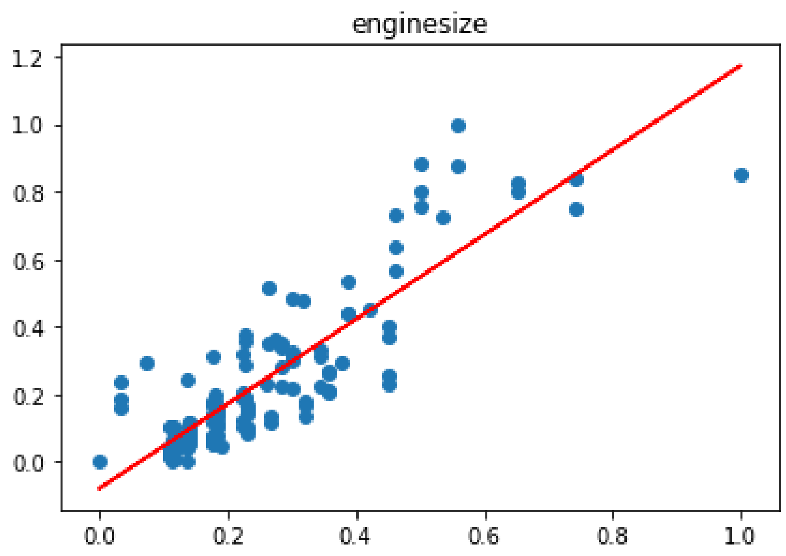
plt.plot(X_test, pre, 'r-', label='Regression Line', linewidth=3)

plt.xlabel('Input Feature')

plt.ylabel('Target Variable')

plt.legend()

plt.show()



Result

Thus, the program has been executed successfully.

Ex.No:3 **SIMPLE PERCEPTRON**

Aim

To Implement a simple perceptron in TensorFlow/Keras Environment.

Algorithm

Step1:Import the necessary packages

Step2:Load the randomized value

Step3:Add the model specification like activation function,input shape.

Step4:Build the model with the hyperparameters

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Predict the values given by user.

Program

```
#import the necessary packages
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 1])
model = Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))
model.compile(loss='mean_squared_error',optimizer=SGD(learning_rate=0.1),metrics='accuracy')
model.fit(x, y, epochs=1000, verbose=0)
```

<keras.src.callbacks.History at 0x1fa06d22150>

```
loss = model.evaluate(x, y)
print(f'Loss: {loss}')
```

<keras.src.callbacks.History at 0x1fa06d22150>

```
predictions = model.predict(x)
```

1/1 [==============================] - 0s 95ms/step

```
pre = np.array(predictions)
for i in range(len(X)):
    print(f'Input: {X[i]}, Predicted Output: {predictions[i][0]:.2f}')
```

Predictions:

Input: [0 0], Predicted Output: 0.22

Input: [0 1], Predicted Output: 0.87

Input: [1 0], Predicted Output: 0.87

Input: [1 1], Predicted Output: 0.99

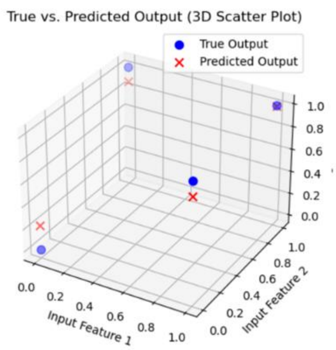
```
weights, biases = model.layers[0].get_weights()
print("Weights:",weights)
print("\nBiases:",biases)
```

Weights: [[3.1921592]

[3.1871004]]

Biases: [-1.2799131]

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], y, c='b', marker='o', label='True Output', s=50)
ax.scatter(X[:, 0], X[:, 1], pre, c='r', marker='x', label='Predicted Output', s=50)
ax.set_xlabel('Input Feature 1')
ax.set_ylabel('Input Feature 2')
ax.set_zlabel('Output')
ax.set_title('True vs. Predicted Output (3D Scatter Plot)')
plt.legend()
plt.show()
```



Result

Thus, the program has been executed successfully.

Ex.No:4 **FEEDFORWARD NEURAL NETWORK**

Aim

To Implement a multi layer perceptron network model in TensorFlow/Keras.

Algorithm

Step1:Import the necessary packages

Step2:Load the dataset

Step3:Dense layer to be more than one and activation function,input shape.

Step4:Build the model with the hyperparameters

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Predict the values given by user.

Program

#import the necessary library

import pandas as pd

import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt

#load the dataset

dataset = pd.read_csv('D:/SJIT/DL/LAB/loan.csv')

print(dataset.head())

	exp	salary	loan
0	6	25620	1
1	10	22262	1
2	0	76763	0
3	1	69384	0
4	0	50882	0

#Assigning the input features to X and class label to Y

```
X=dataset[['exp','salary']]
y=dataset['loan']
# initializing an empty neural network model
model = keras.Sequential()
# Adding layers the model
        # Input layer with 2 features
model.add(layers.Dense(64, input_dim=2, activation='relu'))
#32 hidden layers with rectified linear unit as activation
model.add(layers.Dense(32, activation='relu'))
 # Output layer for binary classification
model.add(layers.Dense(1, activation='sigmoid'))
#model compilation
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#train the model with specified epochs and batchsize
model.fit(X, y, epochs=100, batch_size=32,)
```

```
<keras.src.callbacks.History at 0x1f1f8c3bc10>
```

```
#print the value of loss and accuracy
loss, accuracy = model.evaluate(X, y)
print(f"Loss: {loss}, Accuracy: {accuracy}")
```

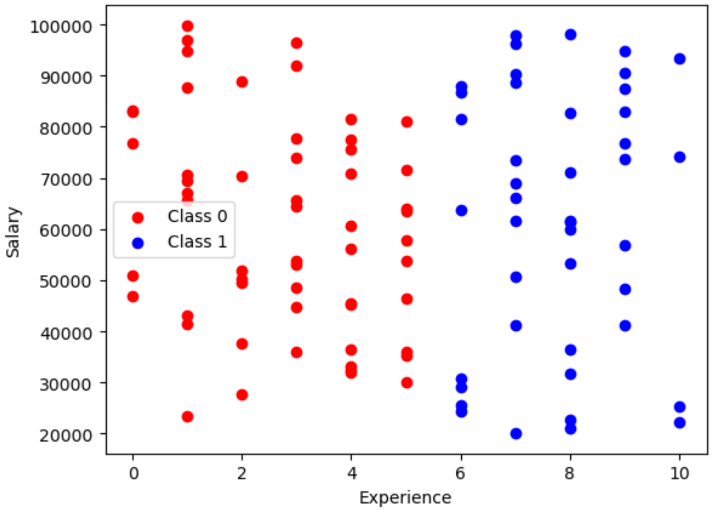
```
4/4 [==============================] - 0s 6ms/step - loss: 47.6960
- accuracy: 0.4343
Loss: 47.696041107177734, Accuracy: 0.4343434274196625
```

```
#predict the model on test data
X_test = [[1,10000]]  # Example test data
predictions = model.predict(X_test)
#display the predicted label for input data
binary_predictions = (predictions > 0.5).astype(int)
binary_predictions
```

```
array([[1]])
```

#Visualize in graph

```
plt.scatter(X[y == 0]['exp'], X[y == 0]['salary'], c='r', label='Class 0')
plt.scatter(X[y == 1]['exp'], X[y == 1]['salary'], c='b', label='Class 1')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.legend()
plt.show()
```



Result

Thus, the program has been executed successfully.

Ex.No:5 **IMAGE CLASSIFICATION USING CNN**

Aim

To implement an Image classifier model using CNN in Keras/Tensorflow

Algorithm

Step1:Import the necessary packages

Step2:Load the dataset of apple and tomato

Step3:Add the convolution layers with filter size and pooling

Step4:Build the model with the hyperparameters and train the model

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Classify the image given by user.

Program

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import os
from tensorflow.keras.preprocessing import image
import numpy as np
train_dir = "D:/SJIT/DL/LAB/at/train"
test_dir = "D:/SJIT/DL/LAB/at/test"
img_height, img_width = 224, 224
num_classes = len(os.listdir(train_dir))
datagen = ImageDataGenerator( rescale=1./255, validation_split=0.2)
train_generator = datagen.flow_from_directory(train_dir,
   target_size=(224,224),    batch_size=20,
class_mode='categorical',subset='training',shuffle=True)
```

Found 236 images belonging to 2 classes.

validation_generator = datagen.flow_from_directory(train_dir, target_size=(224,224), batch_size=20, class_mode='categorical',subset='validation', shuffle=False)

Found 58 images belonging to 2 classes.

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')])
    model.compile(optimizer='adam',loss='categorical_crossentropy',
    metrics=['accuracy'])
    model.fit(train_generator, epochs=10, validation_data=validation_generator)
```

```
Epoch 1/10
12/12 [==============================] - 7s 507ms/step - loss: 0.6828 - accuracy: 0.5890 - val_loss: 0.6767 - val_accuracy: 0.5
690
Epoch 2/10
12/12 [==============================] - 6s 509ms/step - loss: 0.6235 - accuracy: 0.6525 - val_loss: 0.6692 - val_accuracy: 0.5
517
Epoch 3/10
12/12 [==============================] - 7s 627ms/step - loss: 0.6466 - accuracy: 0.5678 - val_loss: 0.6561 - val_accuracy: 0.6
034
Epoch 4/10
12/12 [==============================] - 8s 635ms/step - loss: 0.5550 - accuracy: 0.7458 - val_loss: 0.7181 - val_accuracy: 0.6
207
Epoch 5/10
12/12 [==============================] - 8s 661ms/step - loss: 0.4958 - accuracy: 0.7797 - val_loss: 0.7376 - val_accuracy: 0.6
207
Epoch 6/10
12/12 [==============================] - 8s 676ms/step - loss: 0.5235 - accuracy: 0.7669 - val_loss: 0.6958 - val_accuracy: 0.6
207
Epoch 7/10
12/12 [==============================] - 8s 654ms/step - loss: 0.5402 - accuracy: 0.7627 - val_loss: 0.6964 - val_accuracy: 0.6
379
Epoch 8/10
12/12 [==============================] - 7s 616ms/step - loss: 0.4487 - accuracy: 0.8051 - val_loss: 0.8291 - val_accuracy: 0.6
034
Epoch 9/10
12/12 [==============================] - 8s 657ms/step - loss: 0.4468 - accuracy: 0.8051 - val_loss: 0.8409 - val_accuracy: 0.5
690
Epoch 10/10
12/12 [==============================] - 8s 629ms/step - loss: 0.5805 - accuracy: 0.6737 - val_loss: 0.7621 - val_accuracy: 0.6
207

Out[12]: <keras.src.callbacks.History at 0x27c472c25d0>
```

```
img_path = "D:\\SJIT\\DL\\LAB\\lp.jpg"  # Replace with the path to your image

img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size if needed

img = image.img_to_array(img)

img = np.expand_dims(img, axis=0)

img = img / 255.0

predictions = model.predict(img)
```

1/1 [==============================] - 0s 140ms/step

```
predicted_class = np.argmax(predictions)

class_labels = {0: 'apples', 1: 'tomatoes'}

predicted_label = class_labels[predicted_class]

print(f"Predicted class: {predicted_class} (Label: {predicted_label})")
```

Predicted Class:apple

Result

Thus,the Program has been executed successfully.

Ex:No:6 **FINE TUNE HYPERPARAMETERS**

Aim

To Improve the Deep learning classification model by fine tuning hyper parameters.

Algorithm

Step1:Import the necessary packages

Step2:Load the dataset of apple and tomato

Step3:Add the convolution layers with filter size and pooling

Step4:Build the model with the hyperparameters and train the model

Step5:Fine tune the hyperparameters of the model for better accuracy

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Classify the image given by user.

Program

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import os
from tensorflow.keras.preprocessing import image
import numpy as np
train_dir = "D:/SJIT/DL/LAB/at/train"
test_dir = "D:/SJIT/DL/LAB/at/test"
img_height, img_width = 224, 224
num_classes = len(os.listdir(train_dir))
datagen = ImageDataGenerator( rescale=1./255, validation_split=0.2)
train_generator = datagen.flow_from_directory(train_dir,
```

```
    target_size=(224,224),    batch_size=20,
class_mode='categorical',subset='training',shuffle=True)
```

Found 236 images belonging to 2 classes.

```
validation_generator = datagen.flow_from_directory(train_dir,     target_size=(224,224),
batch_size=20, class_mode='categorical',subset='validation', shuffle=False)
```

Found 58 images belonging to 2 classes.

```
model = Sequential([Conv2D(32, (3, 3), activation='relu', input_shape=(img_height,
img_width, 3)),MaxPooling2D((2, 2)), Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),Conv2D(64, (3, 3), activation='relu'),MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),Flatten(),Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')])
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])
img_path = "D:\\SJIT\\DL\\LAB\\lp.jpg"  # Replace with the path to your image
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size if needed
img = image.img_to_array(img)
img = np.expand_dims(img, axis=0)
img = img / 255.0
predictions = model.predict(img)
1/1 [==============================] - 0s 140ms/step
predicted_class = np.argmax(predictions)
class_labels = {0: 'apples', 1: 'tomatoes'}
predicted_label = class_labels[predicted_class]
print(f"Predicted class: {predicted_class} (Label: {predicted_label})")
```

Predicted Class:apple

Result

Thus,the Program has been executed successfully.

EX:No:7 TRANSFER LEARNING

Aim

To Implement a Transfer Learning concept in Image Classification.

Algorithm

Step1:Import the necessary packages

Step2:Load the required dataset

Step3:Select the pretrained model and their weights

Step4:Freeze the last layer of the model

Step5:Fine tune the hyperparameters of the model for better accuracy

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Classify the image given by user.

Program

```
import tensorflow as tf
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
# Set your custom dataset path
train_dir = "D:/SJIT/DL/LAB/at/train"
test_dir = "D:/SJIT/DL/LAB/at/test"
# Define hyperparameters
img_width, img_height = 224, 224
batch_size = 32
num_classes = 2  # The number of classes in your dataset
epochs = 10
# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width,
img_height, 3))
# Create a custom classification model on top of VGG16
model = Sequential()
model.add(base_model)  # Add the pre-trained VGG16 model
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax')
```

 # Freeze the pre-trained layers

```
for layer in base_model.layers:
    layer.trainable = False
```

 # Compile the model

```
model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])
```

 # Train the model

```
model.fit(train_generator, epochs=epochs, validation_data=validation_generator)
```

 # Optionally, you can unfreeze and fine-tune some layers

```
for layer in base_model.layers[-4:]:
    layer.trainable = True
model.compile(optimizer=Adam(lr=0.00001), loss='categorical_crossentropy',
metrics=['accuracy'])
```

 # Continue training for additional epochs

```
model.fit(train_generator, epochs=epochs, validation_data=validation_generator)
img_path = "D:\\SJIT\\DL\\LAB\\lp.jpg"  # Replace with the path to your image
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size if needed
img = image.img_to_array(img)
img = np.expand_dims(img, axis=0)
img = img / 255.0
predictions = model.predict(img)
1/1 [==============================] - 0s 140ms/step
predicted_class = np.argmax(predictions)
class_labels = {0: 'apples', 1: 'tomatoes'}
predicted_label = class_labels[predicted_class]
print(f"Predicted class: {predicted_class} (Label: {predicted_label})")
```

Predicted Class:apple

Result

Thus,the program has been executed Successfully.

EX:No:8 **PRE-TRAINED MODEL ON KERAS**

Aim

To use a pre trained model on Keras for Transfer Learning

Algorithm

Step1:Import the necessary packages

Step2:Load the required dataset

Step3:Select the pretrained model and their weights

Step4:Freeze the last layer of the model

Step5:Fine tune the hyperparameters of the model for better accuracy

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Classify the image given by user.

Program

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# Preprocess the data
x_train = preprocess_input(x_train)
x_test = preprocess_input(x_test)
# Convert labels to one-hot encoding
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

```
# Load the pre-trained VGG16 model (without top layers)
base_model = VGG16(weights='imagenet', include_top=False)
# Add your custom classification layers on top
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
# Assuming 10 classes for CIFAR-10
predictions = Dense(10, activation='softmax')(x)
model = tf.keras.models.Model(inputs=base_model.input, outputs=predictions)
# Freeze the layers of the pre-trained base model
for layer in base_model.layers:
    layer.trainable = False
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
datagen.fit(x_train)
batch_size = 32
epochs = 10
history = model.fit(datagen.flow(x_train, y_train, batch_size=batch_size),
    steps_per_epoch=len(x_train) / batch_size,epochs=epochs, validation_data=(x_test,
y_test))
# Load a test image
test_image_path = 'D:\\SJIT\\DL\\LAB\\horse.jpg'  # Replace with the path to your test
image
test_image = image.load_img(test_image_path, target_size=(224, 224))
test_image = image.img_to_array(test_image)
```

```
test_image = np.expand_dims(test_image, axis=0)

test_image = preprocess_input(test_image)

# Make predictions

predictions = model.predict(test_image)

# Decode and print the top-3 predicted classes

decoded_predictions = tf.keras.applications.vgg16.decode_predictions(predictions,
top=3)[0]

print("Predictions:")

for i, (imagenet_id, label, score) in enumerate(decoded_predictions):

    print(f"{i + 1}: {label} ({score:.2f})")

# Display the test image

img = plt.imread('gh.jpg')

plt.imshow(img)

plt.show()
```

horse

Result

Thus the program has been executed successfully.

EX:No:9a **SENTIMENT ANALYSIS USING RNN**

Aim

To Perform Sentiment Analysis on text reviews using RNN.

Algorithm

Step1:Import the necessary packages

Step2:Load the imdb review dataset from online

Step3:Convert the words as key and values

Step4:Split the train and test and fix the key words to be of same size

Step5:Set hyperparameters of the model for better accuracy

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Provide the sentiment of the text given by user.

Program

```
from tensorflow.keras.layers import SimpleRNN,Dense, Embedding
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
import numpy as np
vocab_size = 5000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
print(x_train[0])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 4s 0us/step
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4,
173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5,
150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38,
13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4,
22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17,
12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 2, 16,
480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48,
25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4,
107, 117, 2, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26,
400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071,
56, 26, 141, 6, 194, 2, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144,
30, 2, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88,
12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 2, 19, 178, 32]
```

 # Getting all the words from word_index dictionary

word_idx = imdb.get_word_index()

 # Originally the index number of a value and not a key,

 # hence converting the index as key and the words as values

word_idx = {i: word for word, i in word_idx.items()}

 # again printing the review

print([word_idx[i] for i in x_train[0]])

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [==============================] - 0s 0us/step
['the', 'as', 'you', 'with', 'out', 'themselves', 'powerful', 'lets',
'loves', 'their', 'becomes', 'reaching', 'had', 'journalist', 'of', 'lot',
'from', 'anyone', 'to', 'have', 'after', 'out', 'atmosphere', 'never',
'more', 'room', 'and', 'it', 'so', 'heart', 'shows', 'to', 'years', 'of',
'every', 'never', 'going', 'and', 'help', 'moments', 'or', 'of', 'every',
'chest', 'visual', 'movie', 'except', 'her', 'was', 'several', 'of',
'enough', 'more', 'with', 'is', 'now', 'current', 'film', 'as', 'you', 'of',
'mine', 'potentially', 'unfortunately', 'of', 'you', 'than', 'him', 'that',
'with', 'out', 'themselves', 'her', 'get', 'for', 'was', 'camp', 'of',
'you', 'movie', 'sometimes', 'movie', 'that', 'with', 'scary', 'but', 'and',
'to', 'story', 'wonderful', 'that', 'in', 'seeing', 'in', 'character', 'to',
'of', '70s', 'and', 'with', 'heart', 'had', 'shadows', 'they', 'of', 'here',
'that', 'with', 'her', 'serious', 'to', 'have', 'does', 'when', 'from',
'why', 'what', 'have', 'critics', 'they', 'is', 'you', 'that', "isn't",
'one', 'will', 'very', 'to', 'as', 'itself', 'with', 'other', 'and', 'in',
'of', 'seen', 'over', 'and', 'for', 'anyone', 'of', 'and', 'br', "show's",
'to', 'whether', 'from', 'than', 'out', 'themselves', 'history', 'he',
'name', 'half', 'some', 'br', 'of', 'and', 'odd', 'was', 'two', 'most',
'of', 'mean', 'for', '1', 'any', 'an', 'boat', 'she', 'he', 'should', 'is',
'thought', 'and', 'but', 'of', 'script', 'you', 'not', 'while', 'history',
'he', 'heart', 'to', 'real', 'at', 'and', 'but', 'when', 'from', 'one',
'bit', 'then', 'have', 'two', 'of', 'script', 'their', 'with', 'her',
'nobody', 'most', 'that', 'with', "wasn't", 'to', 'with', 'armed', 'acting',
'watch', 'an', 'for', 'with', 'and', 'film', 'want', 'an']
```

```
# Get the minimum and the maximum length of reviews
print("Max length of a review:: ", len(max((x_train+x_test), key=len)))
print("Min length of a review:: ", len(min((x_train+x_test), key=len)))
```

```
Max length of a review::  2697
Min length of a review::  70
```

```
from tensorflow.keras.preprocessing import sequence
# Keeping a fixed length of all reviews to max 400 words
max_words = 400
x_train = sequence.pad_sequences(x_train, maxlen=max_words)
x_test = sequence.pad_sequences(x_test, maxlen=max_words)
x_valid, y_valid = x_train[:64], y_train[:64]
x_train_, y_train_ = x_train[64:], y_train[64:]
# fixing every word's embedding size to be 32
embd_len = 32
# Creating a RNN model
RNN_model = Sequential(name="Simple_RNN")
RNN_model.add(Embedding(vocab_size,embd_len,input_length=max_words))
# In case of a stacked(more than one layer of RNN)
# use return_sequences=True
RNN_model.add(SimpleRNN(128,activation='tanh',return_sequences=False))
RNN_model.add(Dense(1, activation='sigmoid'))
# printing model summary
print(RNN_model.summary())
# Compiling model
RNN_model.compile(loss="binary_crossentropy",optimizer='adam',metrics= ['accurcy'])
# Training the model
history = RNN_model.fit(x_train_,
y_train_,batch_size=64,epochs=5,verbose=1,validation_data=(x_valid, y_valid))
# Printing model score on test data
print()
print("Simple_RNN Score---> ", RNN_model.evaluate(x_test, y_test, verbose=0))
```

```
Model: "Simple_RNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 400, 32)           160000

 simple_rnn (SimpleRNN)      (None, 128)               20608

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 180737 (706.00 KB)
Trainable params: 180737 (706.00 KB)
Non-trainable params: 0 (0.00 Byte)

_________________________________________________________________
None
Epoch 1/5
390/390 [==============================] - 74s 178ms/step - loss:
0.6710 - accuracy: 0.5570 - val_loss: 0.6137 - val_accuracy: 0.6406
Epoch 2/5
390/390 [==============================] - 67s 173ms/step - loss:
0.5784 - accuracy: 0.6828 - val_loss: 0.6273 - val_accuracy: 0.6875
Epoch 3/5
390/390 [==============================] - 68s 175ms/step - loss:
0.5125 - accuracy: 0.7515 - val_loss: 0.6292 - val_accuracy: 0.6719
Epoch 4/5
390/390 [==============================] - 67s 171ms/step - loss:
0.4596 - accuracy: 0.7880 - val_loss: 0.6726 - val_accuracy: 0.6094
Epoch 5/5
390/390 [==============================] - 69s 177ms/step - loss:
0.4190 - accuracy: 0.8129 - val_loss: 0.4257 - val_accuracy: 0.8438

Simple_RNN Score--->  [0.4607064425945282, 0.8044000267982483]
```

Result

Thus,the program has been executed successfully.

EX:No:9b **SENTIMENT ANALYSIS USING RNN**

Aim

To Perform Sentiment Analysis on text reviews using RNN.

Algorithm

Step1:Import the necessary packages

Step2:Load the dataset from local computer

Step3:Convert the words as key and values

Step4:Split the train and test and fix the key words to be of same size

Step5:Set hyperparameters of the model for better accuracy

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Provide the sentiment of the text given by user.

Program

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
import pandas as pd
from sklearn.preprocessing import LabelEncoder
dataset = pd.read_csv('twitter_training.csv')
texts = dataset['text'].astype(str).tolist()  # Convert all values to strings
labels = dataset['label'].tolist()
dataset['text'].fillna('', inplace=True)
print(dataset.head())
```

```
    sno        place     label  \
0  2401  Borderlands  Positive
1  2401  Borderlands  Positive
2  2401  Borderlands  Positive
3  2401  Borderlands  Positive
4  2401  Borderlands  Positive

                                                text
0  im getting on borderlands and i will murder yo...
1  I am coming to the borders and I will kill you...
2  im getting on borderlands and i will kill you ...
3  im coming on borderlands and i will murder you...
4  im getting on borderlands 2 and i will murder ...
```

 # Tokenize the texts

tokenizer = Tokenizer()

tokenizer.fit_on_texts(texts)

vocab_size = len(tokenizer.word_index) + 1

 # Convert texts to sequences

sequences = tokenizer.texts_to_sequences(texts)

 # Pad sequences to have consistent length

max_length = max(len(seq) for seq in sequences)

padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post')

label_encoder = LabelEncoder()

labels = label_encoder.fit_transform(dataset['label'])

 # Now, labels should be numerical

labels = tf.keras.utils.to_categorical(labels)

 # Build the RNN model

model = Sequential()

model.add(Embedding(vocab_size, 64, input_length=max_length))

model.add(SimpleRNN(128))

model.add(Dense(4, activation='softmax')) # Assuming binary classification

 # Compile the model

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

 # Train the model

model.fit(padded_sequences, labels, epochs=10, validation_split=0.2)

 # Input a new sentence

new_sentence = "This is a good place to visit."

 # Tokenize and pad the new sentence

new_sequence = tokenizer.texts_to_sequences([new_sentence])

new_padded_sequence = pad_sequences(new_sequence, maxlen=max_length, padding='post')

```
# Train the model
model.fit(padded_sequences, labels, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
1868/1868 [==============================] - 147s 77ms/step - loss: 1.3840 - accuracy: 0.2852 - val_loss: 1.3547 - val_accurac
y: 0.2515
Epoch 2/10
1868/1868 [==============================] - 142s 76ms/step - loss: 1.3795 - accuracy: 0.2863 - val_loss: 1.3710 - val_accurac
y: 0.3515
Epoch 3/10
1868/1868 [==============================] - 138s 74ms/step - loss: 1.3769 - accuracy: 0.2869 - val_loss: 1.3444 - val_accurac
y: 0.3515
Epoch 4/10
1868/1868 [==============================] - 139s 74ms/step - loss: 1.3766 - accuracy: 0.2864 - val_loss: 1.3425 - val_accurac
y: 0.3515
Epoch 5/10
1868/1868 [==============================] - 143s 77ms/step - loss: 1.3803 - accuracy: 0.2821 - val_loss: 1.3541 - val_accurac
y: 0.2831
Epoch 6/10
1868/1868 [==============================] - 141s 76ms/step - loss: 1.3807 - accuracy: 0.2813 - val_loss: 1.3642 - val_accurac
y: 0.3508
Epoch 7/10
1868/1868 [==============================] - 144s 77ms/step - loss: 1.3795 - accuracy: 0.2829 - val_loss: 1.3536 - val_accurac
y: 0.3515
Epoch 8/10
1868/1868 [==============================] - 146s 78ms/step - loss: 1.3794 - accuracy: 0.2853 - val_loss: 1.3447 - val_accurac
```

 # Make predictions

predictions = model.predict(new_padded_sequence)

 # Convert predictions to class labels

predicted_label = label_encoder.inverse_transform([tf.argmax(predictions, axis=1).numpy()[0]])[0]

 # Print the result

print(f"The model predicts: {predicted_label}")

The model predicts:positive

Result

Thus,the program has been executed successfully.

EX:No:10 **LSTM BASED AUTOENCODER**

Aim

To Implement an LSTM based Autoencoder in TensorFlow/Keras.

Algorithm

Step1:Import the necessary packages

Step2:Load the dataset from online

Step3:Compress the input dimension of the images

Step4:Split the train and test

Step5:Set hyperparameters of the encoder

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:model provides the reduces dimension of the original image.

Program

import numpy as np

import tensorflow as tf

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

from tensorflow.keras.models import Model

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import plot_model

import matplotlib.pyplot as plt

 # Load MNIST dataset

(x_train, _), (x_test, _) = mnist.load_data()

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 10s 1us/step
```

 # Normalize and reshape the data

x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0

x_train = np.reshape(x_train, (len(x_train), 28, 28))

x_test = np.reshape(x_test, (len(x_test), 28, 28))

 # Define the model

latent_dim = 32

inputs = Input(shape=(28, 28))

encoded = LSTM(latent_dim)(inputs)

decoded = RepeatVector(28)(encoded)

decoded = LSTM(28, return_sequences=True)(decoded)

sequence_autoencoder = Model(inputs, decoded)

 # Compile the model

sequence_autoencoder.compile(optimizer='adam', loss='mean_squared_error')

 # Print the model summary

sequence_autoencoder.summary()

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28)]          0

 lstm (LSTM)                 (None, 32)                7808

 repeat_vector (RepeatVecto  (None, 28, 32)            0
 r)

 lstm_1 (LSTM)               (None, 28, 28)            6832

=================================================================
Total params: 14640 (57.19 KB)
Trainable params: 14640 (57.19 KB)
Non-trainable params: 0 (0.00 Byte)
```

 # Train the model

sequence_autoencoder.fit(x_train, x_train, epochs=10, batch_size=128, shuffle=True, validation_data=(x_test, x_test))

```
Epoch 1/10
469/469 [==============================] - 24s 41ms/step - loss: 0.0641
- val_loss: 0.0562
Epoch 2/10
469/469 [==============================] - 18s 38ms/step - loss: 0.0530
- val_loss: 0.0498
Epoch 3/10
469/469 [==============================] - 16s 33ms/step - loss: 0.0476
- val_loss: 0.0450
Epoch 4/10
469/469 [==============================] - 16s 33ms/step - loss: 0.0440
- val_loss: 0.0421
Epoch 5/10
469/469 [==============================] - 16s 34ms/step - loss: 0.0415
- val_loss: 0.0399
Epoch 6/10
469/469 [==============================] - 15s 32ms/step - loss: 0.0394
- val_loss: 0.0383
Epoch 7/10
469/469 [==============================] - 16s 35ms/step - loss: 0.0378
- val_loss: 0.0364
Epoch 8/10
469/469 [==============================] - 18s 37ms/step - loss: 0.0364
- val_loss: 0.0351
Epoch 9/10
469/469 [==============================] - 17s 36ms/step - loss: 0.0351
- val_loss: 0.0341
Epoch 10/10
469/469 [==============================] - 15s 32ms/step - loss: 0.0341
- val_loss: 0.0331
Out[11]:
<keras.src.callbacks.History at 0x1a9e16a6350>
```

 # Generate reconstructed images

decoded_images = sequence_autoencoder.predict(x_test)

```
313/313 [==============================] - 5s 11ms/step
```

 # Plot original and reconstructed images

n = 10 **# Number of images to display**

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

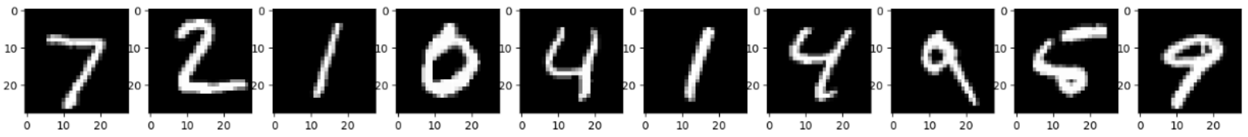
for i in range(n):

  # Original images

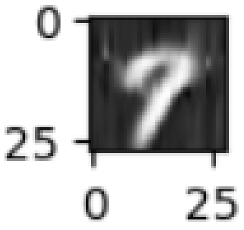
 ax = plt.subplot(2, n, i + 1)

 plt.imshow(x_test[i].reshape(28, 28))

```
plt.gray()
ax.get_xaxis().set_visible(True)
ax.get_yaxis().set_visible(True)
```



```
    # Reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_images[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Result

Thus,the program has been executed successfully.

EX:No:11 **IMAGE GENERATION USING GAN**

Aim

To Generate a new image using GAN

Algorithm

Step1:Import the necessary packages

Step2:Load the required dataset

Step3:Define the generator and discriminator model

Step4:Combine the both models to form a GAN

Step5:Train the GAN by specifying the hyperparameters.

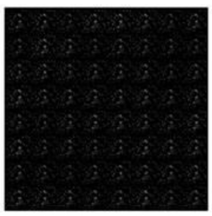
Step6:Generate the new images and plot along with dataset images

Program

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Reshape, Flatten
from tensorflow.keras.layers import BatchNormalization, LeakyReLU
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist
# Load MNIST data
(x_train, _), (_, _) = mnist.load_data()
# Normalize and reshape data
x_train = x_train / 127.5 - 1.0
x_train = np.expand_dims(x_train, axis=3)
# Define the generator model
generator = Sequential()
generator.add(Dense(128 * 7 * 7, input_dim=100))
generator.add(LeakyReLU(0.2))
generator.add(Reshape((7, 7, 128)))
generator.add(BatchNormalization())
```

```
generator.add(Flatten())

generator.add(Dense(28 * 28 * 1, activation='tanh'))

generator.add(Reshape((28, 28, 1)))

# Define the discriminator model

discriminator = Sequential()

discriminator.add(Flatten(input_shape=(28, 28, 1)))

discriminator.add(Dense(128))

discriminator.add(LeakyReLU(0.2))

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

# Compile the discriminator

discriminator.compile(loss='binary_crossentropy',
optimizer=Adam(learning_rate=0.0002, beta_1=0.5), metrics=['accuracy'])

# Freeze the discriminator during GAN training

discriminator.trainable = False

# Combine generator and discriminator into a GAN model

gan = Sequential()

gan.add(generator)

gan.add(discriminator)

# Compile the GAN

gan.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0002,
beta_1=0.5))

# Function to train the GAN

def train_gan(epochs=1, batch_size=128):

    batch_count = x_train.shape[0] // batch_size

    for e in range(epochs):

        for _ in range(batch_count):

            noise = np.random.normal(0, 1, size=[batch_size, 100])

            generated_images = generator.predict(noise)

            image_batch = x_train[np.random.randint(0, x_train.shape[0], size=batch_size)]

            X = np.concatenate([image_batch, generated_images])
```

```
            y_dis = np.zeros(2 * batch_size)
            y_dis[:batch_size] = 0.9  # Label smoothing
            discriminator.trainable = True
            d_loss = discriminator.train_on_batch(X, y_dis)
            noise = np.random.normal(0, 1, size=[batch_size, 100])
            y_gen = np.ones(batch_size)
            discriminator.trainable = False
            g_loss = gan.train_on_batch(noise, y_gen)
        print(f"Epoch {e+1}/{epochs}, Discriminator Loss: {d_loss[0]},
         Generator Loss: {g_loss}")
# Train the GAN
train_gan(epochs=200, batch_size=128)
# Generate and plot some images
def plot_generated_images(epoch, examples=10, dim=(1, 10), figsize=(10, 1)):
    noise = np.random.normal(0, 1, size=[examples, 100])
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(examples, 28, 28)
    plt.figure(figsize=figsize)
    for i in range(generated_images.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generated_images[i], interpolation='nearest', cmap='gray_r')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig(f'gan_generated_image_epoch_{epoch}.png')
# Plot generated images for a few epochs
for epoch in range(1, 10):
    plot_generated_images(epoch)
```



Epoch 1

Epoch 200

Result

Thus,the program has been executed successfully.

EX:No:12 CLASSIFY AN IMAGE USING PRETRAINED MODEL

Aim

To Train a Deep learning model to classify a given image using pre trained model

Algorithm

Step1:Import the necessary packages

Step2:Load the required dataset

Step3:Select the pretrained model and their weights

Step4:Freeze the last layer of the model

Step5:Fine tune the hyperparameters of the model for better accuracy

Step5:Compile the model with accuracy ,loss and optimizer

Step6:Train the model with specified epochs and batch size

Step7:Classify the image given by user.

Program

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input,
decode_predictions
```

 # Load the pre-trained VGG16 model

```
model = VGG16(weights='imagenet')
```

 # Load and preprocess an image for prediction

```
def preprocess_image(image_path):

    img = image.load_img(image_path, target_size=(224, 224))

    img_array = image.img_to_array(img)

    img_array = np.expand_dims(img_array, axis=0)

    img_array = preprocess_input(img_array)

    return img_array
```

 # Function to predict the class of an image

```
def predict_image_class(image_path):
```

img_array = preprocess_image(image_path)

predictions = model.predict(img_array)

 # Get the top 3 predictions

decoded_predictions = decode_predictions(predictions, top=3)[0]

for i, (imagenet_id, label, score) in enumerate(decoded_predictions):

print(f"{i + 1}: {label} ({score:.2f})")

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 456s 1us/step
```

 # Test for Prediciton

image_path = 'car.jpg'

predict_image_class(image_path)

```
1/1 [==============================] - 0s 253ms/step
1: sports_car (0.61)
2: racer (0.13)
3: car_wheel (0.09)
```

Result

Thus,the program has been executed successfully.

EX:No:13 RECOMMENDATION SYSTEM

Aim

To create a Recommendation system from sales data using Deep Learning

Algorithm

Program

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Embedding, Flatten, Dense, Concatenate

# Load sales data (user_id, product_id, purchase)

sales_data = pd.read_csv('sales_data.csv')  # Replace with the path to your sales data file

# Use label encoding to convert user and product IDs to numerical values

user_encoder = LabelEncoder()

product_encoder = LabelEncoder()

sales_data['user_id'] = user_encoder.fit_transform(sales_data['user_id'])

sales_data['product_id'] = product_encoder.fit_transform(sales_data['product_id'])

# Split the data into training and testing sets

train_data, test_data = train_test_split(sales_data, test_size=0.2, random_state=42)

# Define the neural network model for collaborative filtering

def build_model(user_dim, product_dim):

    user_input = Input(shape=(1,), name='user_input')

    product_input = Input(shape=(1,), name='product_input')

    user_embedding = Embedding(input_dim=user_dim, output_dim=50,
input_length=1)(user_input)

    product_embedding = Embedding(input_dim=product_dim, output_dim=50,
input_length=1)(product_input)

    user_flatten = Flatten()(user_embedding)
```

```
    product_flatten = Flatten()(product_embedding)
    concatenated = Concatenate()([user_flatten, product_flatten])
    dense1 = Dense(128, activation='relu')(concatenated)
    output = Dense(1, activation='sigmoid')(dense1)
    model = Model(inputs=[user_input, product_input], outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
# Get the number of unique users and products
user_dim = sales_data['user_id'].nunique()
product_dim = sales_data['product_id'].nunique()
# Build and train the model
model = build_model(user_dim, product_dim)
train_user_ids, train_product_ids, train_labels = train_data['user_id'], train_data['product_id'], train_data['purchase']
test_user_ids, test_product_ids, test_labels = test_data['user_id'], test_data['product_id'], test_data['purchase']
model.fit([train_user_ids, train_product_ids], train_labels, epochs=5, batch_size=64, validation_data=([test_user_ids, test_product_ids], test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate([test_user_ids, test_product_ids], test_labels)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
# Make recommendations for a user
# Replace with the user ID for which you want to make recommendations
user_id_to_predict = 100
user_products = sales_data[sales_data['user_id'] == user_id_to_predict]['product_id'].unique()
all_products = np.arange(product_dim)
products_to_recommend = np.setdiff1d(all_products, user_products)
# Predict purchase probability for each product
predictions = model.predict([np.full_like(products_to_recommend, user_id_to_predict), products_to_recommend])
```

```
# Sort products by predicted probability in descending order
sorted_indices = np.argsort(predictions[:, 0])[::-1]
recommended_products = products_to_recommend[sorted_indices]
print("Top 5 recommended products:")
for i in range(5):
    product_id = product_encoder.inverse_transform(recommended_products[i])
    print(f"Product ID: {product_id}")
```

S.No	Product Id	Products
1.	As3565	Acer Laptop
2.	D345f6	Wireless charger
3.	G46765	Sony ipad
4.	T2345	Laptop bag
5.	X56f56	Dell wireless Mouse

Result

Thus,the program has been executed successfully.

Ex:No:14 OBJECT DETECTION USING CNN

Aim

To Implement object detection using CNN

Algorithm

Step1:Import the necessary package

Step2:Load the required dataset

Step3:Define the CNN model along with the convolution,filter,pooling

Step4:Initialize the metrics,optimizer and loss

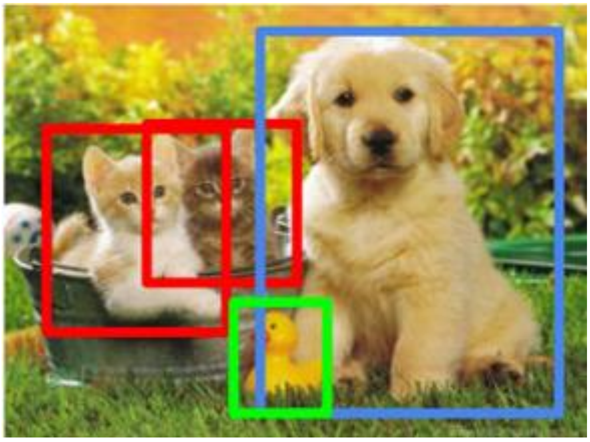
Step5:Train the model on the dataset.

Step6:The model provides bounding boxes on the objects in image

Program

```
import tensorflow as tf
from tensorflow.keras import layers
# Define the custom object detection model
def custom_object_detection_model(input_shape=(224, 224, 3), num_classes=1):
    model = tf.keras.Sequential()
    # Feature extraction layers
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(256, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    # Flatten the output and add dense layers for bounding box regression
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu'))
```

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

    model.add(layers.Dense(4, activation='linear'))  # 4 values for bounding box (x, y, width, height)

    # Add dense layer for class prediction

    model.add(layers.Dense(num_classes, activation='sigmoid'))

    return model

    # Instantiate the model

model = custom_object_detection_model()

    # Compile the model

model.compile(optimizer='adam', loss='mse')  # Use mean squared error for bounding box regression

   # Display the model summary

model.summary()
```



Result

Thus,the program has been executed successfully.

EX:NO:15 REINFORCEMENT ALGORITHM-NLP

Aim

To Implement any simple Reinforcement Algorithm for an NLP problem

Algorithm

Step1:Import the necessary pacakage

Step2:Define the action,policy and reward for the task

Step3:Train the loop with DDPG algorithm

Step4:Evaluate the trained model to test the performance

Step5:Deploy the Reinforcement algorithm to learn the pattern

Program

```
import random
import numpy as np
class SimpleNLPEnvironment:
    def __init__(self):
        # Define possible actions and states
        self.actions = ['generate', 'skip']
        self.states = ['start', 'mid', 'end']
                # Initialize state and dialogue history
        self.state = 'start'
        self.dialogue_history = []
    def reset(self):
        # Reset environment to the initial state
        self.state = 'start'
        self.dialogue_history = []
    def step(self, action):
        # Simulate the environment's response to the agent's action
        # Randomly determine reward
        reward = random.choice([0, 1])  # 1 for positive reward, 0 for no reward
        # Update dialogue history based on the action
```

```
        self.dialogue_history.append(action)
        # Update state based on dialogue history
        if len(self.dialogue_history) == 1:
            self.state = 'mid'
        elif len(self.dialogue_history) == 2:
            self.state = 'end'
        # Check if the agent has reached the end state
        done = self.state == 'end'
        # Provide feedback to the agent based on the reward
        if reward == 1:
            feedback = "Good job!"
        else:
            feedback = "Try again."
        # Return the next state, reward, and additional information
        return self.state, reward, done, feedback
class QLearningAgent:
    def __init__(self, actions, states, learning_rate=0.1, discount_factor=0.9,
exploration_prob=0.1):
        self.actions = actions
        self.states = states
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_prob = exploration_prob
        # Initialize Q-values
        self.q_values = {(state, action): 0.0 for state in states for action in actions}

    def choose_action(self, state):
        # Epsilon-greedy exploration strategy
        if random.uniform(0, 1) < self.exploration_prob:
            return random.choice(self.actions)
```

```
        else:
            # Choose action with the highest Q-value
            return max(self.actions, key=lambda action: self.q_values[(state, action)])
    def update_q_values(self, state, action, reward, next_state):
        # Q-value update using the Q-learning formula
        max_q_next = max(self.q_values[(next_state, a)] for a in self.actions)
        self.q_values[(state, action)] += self.learning_rate * (
            reward + self.discount_factor * max_q_next - self.q_values[(state, action)]
        )
# Instantiate the environment and agent
env = SimpleNLPEnvironment()
agent = QLearningAgent(actions=env.actions, states=env.states)
# Training loop
num_episodes = 1000
for episode in range(num_episodes):
    env.reset()
    state = env.state
    while True:
        # Choose an action using the Q-learning agent
        action = agent.choose_action(state)
        # Take the chosen action and observe the environment
        next_state, reward, done, feedback = env.step(action)
        # Update Q-values based on the observed reward and next state
        agent.update_q_values(state, action, reward, next_state)
        # Move to the next state
        state = next_state

        # Check if the episode is complete
        if done:
            break
```

```
# After training, test the agent's performance
env.reset()
state = env.state
print("Test the agent:")
while True:
    # Choose the best action according to the learned Q-values
    action = agent.choose_action(state)
    print(f"Agent chooses action: {action}")
    next_state, _, done, feedback = env.step(action)
    # Move to the next state
    state = next_state
    # Check if the episode is complete
    if done:
        break
```

Training Episode 1:

Agent chooses action: generate

Agent chooses action: skip

Training Episode 2:

Agent chooses action: generate

Agent chooses action: generate

Test the agent:

Agent chooses action: generate

Agent chooses action: skip

Agent chooses action: generate

Result

Thus,the program has been executed successfully.