INDEX

Ex. No:1 Date:6-2-2024	IMPLEMENT SIMPLE VECTOR ADDITION IN TENSORFLOW

AIM:

To write a python program to implement simple vector addition in TensorFlow

ALGORITHM:

Step 1: Start

Step 2: Import the TensorFlow library.

Step 3: Define the input vectors that you want to add together. These vectors can be represented as lists, NumPy arrays, or TensorFlow tensors.

Step 4: Convert the input vectors into TensorFlow constant tensors using the tf.constant function. This step ensures that the input data is compatible with TensorFlow operations.

Step 5: Perform addition operation using the tf.add function to add the two input tensors together. This function performs element-wise addition, adding corresponding elements from each tensor.

Step 6: Run the TensorFlow Session

Step 7: Retrieve the result using the numpy() method to convert the TensorFlow tensor to a NumPy array. Step 8: Output the result of the addition operation, which represents the sum of the two input vectors.

Step 9: End.

PROGRAM:

```
import tensorflow as tf # creating a scalar scalar = tf.constant(7) scalar

scalar.ndim

# create a vector

vector = tf.constant([10, 10])

# checking the dimensions of vector vector.ndim

# creating a matrix

matrix = tf.constant([[1, 2], [3, 4]]) print(matrix)

print("the number of dimensions of a matrix is :"+str(matrix.ndim))



# creating two tensors

matrix = tf.constant([[1, 2], [3, 4]])

matrix1 = tf.constant([[2, 4], [6, 8]])

# addition of two matrices

print("Addition of two matrices:")

print(matrix+matrix1)
```

OUTPUT:

```
 tf.Tensor ( [[1 2] [3 4]], shape=(2, 2), dtype=int32)

the number of dimensions of a matrix is :2 Addition of two matrices:

tf.Tensor ( [[ 3 6] [ 9 12]], shape=(2, 2), dtype=int32)
```

RESULT:

Thus the program for simple vector addition in TensorFlow was executed successfully

Ex. No:2 Date:6-2-2024	IMPLEMENT A REGRESSION MODEL IN KERAS

AIM:

To write a python program to implement a regression model in Keras.

ALGORITHM:

Step 1: Start

Step 2: Import libraries NumPy and TensorFlow libraries are imported. Specifically, TensorFlow's Keras API is imported to define and train the neural network model.

Step 3: Generate Random data for regression is generated using NumPy. X represents the features, and y represents the labels. The labels (y) are generated based on a linear relationship with some added noise.

Step 4: Define a sequential model is defined using Keras. It consists of two dense layers. The first layer has 10 neurons with ReLU activation function, and it expects input of shape (1,). The second layer has 1 neuron, which is the output neuron for regression.

Step 5: The model is compiled using the Adam optimizer and mean squared error loss function, which are commonly used for regression tasks.

Step 6: The model is trained on the generated data for 100 epochs with a batch size of 32. The training process aims to minimize the mean squared error loss.

Step 7: Once the model is trained, predictions are made on the same data X used for training.

Step 8: A loop is used to print the predictions made by the model along with the corresponding true labels(y). This allows for a visual comparison of the model's performance.

Step 9: Stop.

PROGRAM:

```
import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

# Generate some random data for regression

np.random.seed(0)

X = np.random.rand(100, 1) # Features

y = 2 * X.squeeze() + 1 + np.random.randn(100) * 0.1 # Labels

# Define the model

model = keras.Sequential([layers.Dense(10, activation='relu', input_shape=(1,)),
layers.Dense(1) ])  # Output layer with one neuron for regression

# Compile the model

model.compile(optimizer='adam', loss='mse') # Using mean squared error loss for regression
# Train the model

model.fit(X, y, epochs=100, batch_size=32, verbose=0) # Training for 100 epochs

# Make predictions

predictions = model.predict(X)

# Print some predictions and true labels for comparison for i in range(5):

print("Predicted:", predictions[i][0], "\tTrue:", y[i])
```

OUTPUT:

```
4/4 [==============================] - 0s 3ms/step

Predicted: 2.0447748 True: 1.9811120237763138

Predicted: 2.3311834 True: 2.520461381440258

Predicted: 2.1376472 True: 2.252092996116334

Predicted: 2.038009    True: 1.9361419973660712

Predicted: 1.8153862 True: 1.9961348180573695
```

RESULT:

Thus the program for a regression model in Keras was executed successfully.

Ex. No:3 Date:6-2-2024	**IMPLEMENT A PERCEPTRON IN TENSORFLOW/KERAS**

AIM:

To write a python program to implement a perceptron in TensorFlow/Keras Environment.

ALGORITHM:

Step 1: Start

Step 2: NumPy and TensorFlow libraries are imported. Specifically, TensorFlow's Keras API is imported to define and train the neural network model.

Step 3: Example data for a logical OR operation is generated. X contains input binary vectors, and y contains corresponding output labels.

Step 4: A sequential model is defined using Keras. It consists of a single dense layer with one neuron. The input shape is (2,), matching the shape of the input vectors. The activation function used is sigmoid, suitable for binary classification tasks like logical OR.

Step 5: The model is compiled using the Adam optimizer and binary cross-entropy loss function, which are common choices for binary classification tasks. Accuracy is also set as a metric to monitor during training.

Step 6: The model is trained on the example data (X and y) for 1000 epochs. The training process aims to minimize the binary cross-entropy loss.

Step 7: Once training is complete, the model is evaluated on the same dataset it was trained on. The loss and accuracy metrics are printed.

Step 8: Finally, the trained model is used to make predictions on the input data X, and the predictions are printed.

Step 9: Stop

PROGRAM:

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Generate some example data for a logical OR operation

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([0, 1, 1, 1])

# Define the perceptron model

model = Sequential([Dense(1, input_shape=(2,), activation='sigmoid', use_bias=True)])

# Compile the model

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

# Train the model

model.fit(X, y, epochs=1000, verbose=0)

# Evaluate the model

loss, accuracy = model.evaluate(X, y)

print("Loss:", loss)

print("Accuracy:", accuracy)

# Make predictions

predictions = model.predict(X)

print("Predictions:", predictions.flatten())
```

OUTPUT:

```
1/1 [==============================] - 0s 156ms/step - loss: 0.5838 - accuracy:
0.7500 Loss: 0.5837528109550476

Accuracy: 0.75

1/1 [==============================] - 0s 46ms/step Predictions: [0.66924465
0.78853077 0.5416373 0.68530434]
```

RESULT:

Thus the program for perceptron in TensorFlow/Keras Environment was done successfully

Ex. No:4 Date:13-2-2024	IMPLEMENT A FEED FORWARD NETWORK IN TENORFLOW/KERAS

AIM:

To write a python program to implement a Feed Forward Neural Network using TensorFlow/keras.

ALGORITHM:

Step 1: Import necessary libraries

Step 2: Set seed for reproducibility.

Step 3: Load and split MNIST dataset into training, validation, and test sets.

Step 4: Print the shapes of the training, validation, and test sets to check the data dimensions.

Step 5: Plot a few samples from the training set using matplotlib.pyplot.

Step 6: Reshape the input images to a 1D array (flatten) for feeding into the neural network.

Step 7: Normalize the pixel values to be between 0 and 1.

Step 8: Load the Fashion MNIST dataset and print the labels of the first few samples to check.

Step 9: Convert the integer labels to one-hot encoded format using to_categorical.

PROGRAM:

```
import random

import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import mnist, fashion_mnist

from tensorflow.keras.utils

import to_categorical SEED_VALUE = 42

random.seed(SEED_VALUE)

np.random.seed(SEED_VALUE)

Tf .random.set_seed(SEED_VALUE)

(X_train_all, y_train_all), (X_test, y_test) = mnist.load_data()

X_valid = X_train_all[:10000]

X_train = X_train_all[10000:]

y_valid = y_train_all[:10000]

y_train = y_train_all[10000:]

print(X_train.shape)

print(X_valid.shape)

print(X_test.shape)

plt.figure(figsize=(18, 5))

for i in range(3):

plt.subplot(1, 3, i + 1)

plt.axis(True)

plt.imshow(X_train[i], cmap='gray')

plt.subplots_adjust(wspace=0.2, hspace=0.2)

X_train = X_train.reshape((X_train.shape[0], 28 * 28))

X_train = X_train.astype("float32") / 255

X_test = X_test.reshape((X_test.shape[0], 28 * 28))

X_test = X_test.astype("float32") / 255
```

X_valid = X_valid.reshape((X_valid.shape[0], 28 * 28))

X_valid = X_valid.astype("float32") / 255

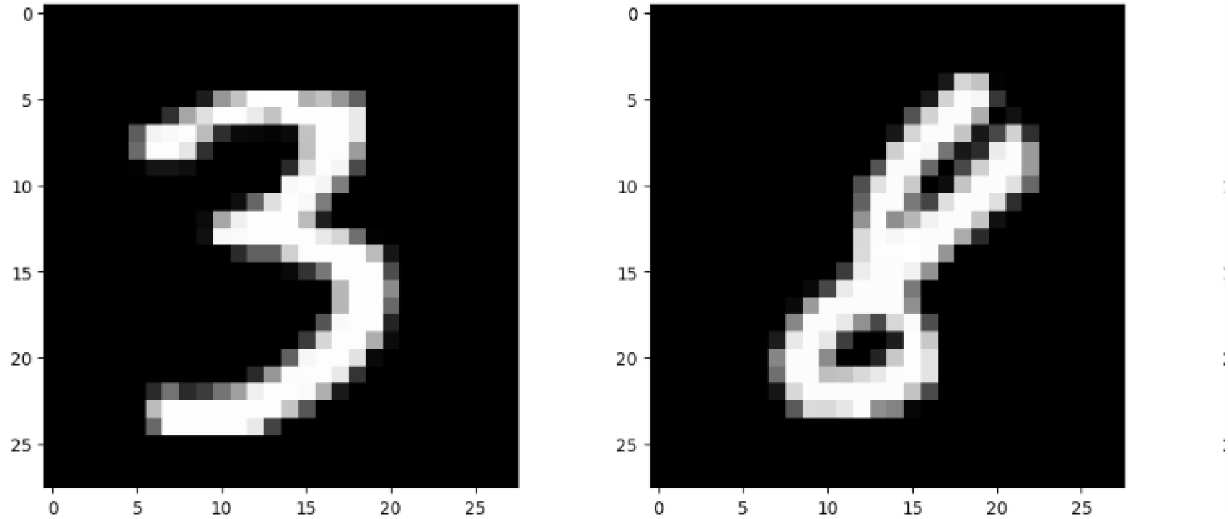
((X_train_fashion, y_train_fashion), (_, _)) = fashion_mnist.load_data()
print(y_train_fashion[0:9])

y_train_onehot = to_categorical(y_train_fashion[0:9])

print(y_train_onehot)

OUTPUT:

```
4422102/4422102 [==============================] - 0s 0us/step
[9 0 0 3 0 2 7 2 5]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
```



RESULT:

Thus the program for a Feed Forward neural network using TensorFlow/Keras was executed successfully.

Ex. No:5 **Date:13-2-2024**	**IMPLEMENT A IMAGE CLASSIFIER USING CNN IN TENSORFLOW/KERAS**

AIM:

To write the python program to implement an image classifier using CNN in tensorflow/ keras.

ALGORITHM:

Step 1: Import necessary libraries.

Step 2: Load and prepare the CIFAR-10 dataset.

Step 3: Define class names for the CIFAR-10 dataset.

Step 4: Visualize the first 25 images from the training set.

Step 5: Define the convolutional neural network (CNN) model.

Step 6: Compile the model and train the model on the training data.

Step 7: Plot the training history (accuracy and epochs) and evoluate the model on the test data.

PROGRAM:

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

train_images, test_images = train_images / 255.0, test_images / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer','dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(train_images[i])
```

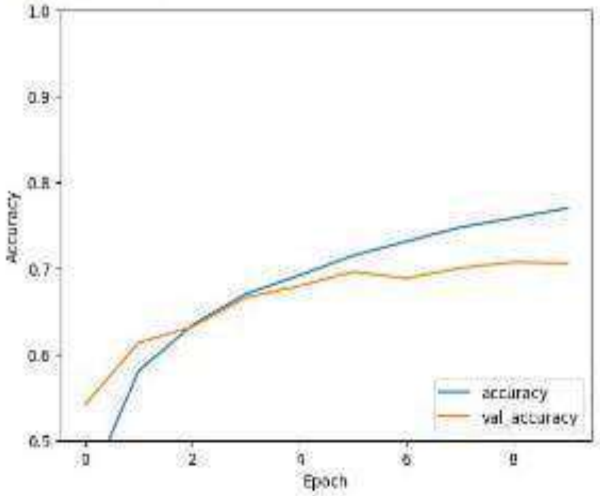
```
# The CIFAR labels happen to be arrays,

# which is why you need the extra index

plt.xlabel(class_names[train_labels[i][0]])

plt.show()

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))

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(64, (3, 3), activation='relu'))

model.summary()

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10))

model.summary()

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentrop
(from_logits=True), metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images,
test_labels)) plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val_accuracy'], label='val_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0.5, 1])

plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)

print(test_acc)
```

OUTPUT:


frog
truck
truck
deer
automobile
automobile
bird
horse
ship
cat
deer
horse
horse
bird
truck
truck
truck
cat
bird
frog
deer
cat
frog
frog
bird

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56320 (220.00 KB)
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 64)          36928

 flatten_1 (Flatten)         (None, 1024)              0

 dense_2 (Dense)             (None, 64)                65600

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
1563/1563 [==============================] - 88s 55ms/step - loss: 1.5345 - accuracy: 0.4369 - val_loss: 1.2593 - val_accuracy: 0.5416
Epoch 2/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.1778 - accuracy: 0.5807 - val_loss: 1.0868 - val_accuracy: 0.6135
Epoch 3/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.0410 - accuracy: 0.6339 - val_loss: 1.0502 - val_accuracy: 0.6318
Epoch 4/10
1563/1563 [==============================] - 71s 46ms/step - loss: 0.9434 - accuracy: 0.6702 - val_loss: 0.9687 - val_accuracy: 0.6661
Epoch 5/10
```

```
dense_2 (Dense)             (None, 64)              65600

dense_3 (Dense)             (None, 10)              650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
1563/1563 [==============================] - 88s 55ms/step - loss: 1.5345 - accuracy: 0.4369 - val_loss: 1.2593 - val_accuracy: 0.5416
Epoch 2/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.1778 - accuracy: 0.5807 - val_loss: 1.0868 - val_accuracy: 0.6135
Epoch 3/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.0410 - accuracy: 0.6339 - val_loss: 1.0502 - val_accuracy: 0.6318
Epoch 4/10
1563/1563 [==============================] - 71s 46ms/step - loss: 0.9434 - accuracy: 0.6702 - val_loss: 0.9687 - val_accuracy: 0.6661
Epoch 5/10
1563/1563 [==============================] - 63s 40ms/step - loss: 0.8714 - accuracy: 0.6918 - val_loss: 0.9300 - val_accuracy: 0.6795
Epoch 6/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.8136 - accuracy: 0.7148 - val_loss: 0.8874 - val_accuracy: 0.6962
Epoch 7/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.7654 - accuracy: 0.7312 - val_loss: 0.8970 - val_accuracy: 0.6883
Epoch 8/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.7229 - accuracy: 0.7475 - val_loss: 0.8896 - val_accuracy: 0.7003
Epoch 9/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6876 - accuracy: 0.7584 - val_loss: 0.8575 - val_accuracy: 0.7072
Epoch 10/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6519 - accuracy: 0.7698 - val_loss: 0.8691 - val_accuracy: 0.7058
313/313 - 3s - loss: 0.8691 - accuracy: 0.7058 - 3s/epoch - 11ms/step
0.7057999968528748
```



RESULT:

Thus the python program for image classifier using CNN in tensorflow/keras was executed successfully.

Ex. No:6 Date:13-2-2024	**IMPROVE THE DEEP LEARNING MODE BY TUNING HYPER PARAMETERS**

AIM:

To write a python program to improve the deep learning model by fine tuning hyper parameters.

ALGORITHM:

Step 1: Import necessary libraries.

Step 2: Generate a synthetic dataset.

Step 3: This creates a dataset with 1000 samples, 20 features, 10 of which are informative and 2 classes. Then define the parameter distribution for hyper parameter tuning.

Step 4: Initialize the decision tree classifier and perform hyper parameter tuning using Randomized Search CV.

Step 5: Print the best parameters and best score found during the hyper parameter tuning process.

PROGRAM:

```
import numpy as np

from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, n_classes=2,
random_state=42)

from scipy.stats import randint

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import RandomizedSearchCV

param_dist = {"max_depth": [3, None], "max_features": randint(1, 9), "min_samples_leaf":
randint(1, 9), "criterion": ["gini", "entropy"]}

tree = DecisionTreeClassifier()

tree_cv = RandomizedSearchCV(tree, param_dist, cv=5)

tree_cv.fit(X, y)

print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))

print("Best score is {}".format(tree_cv.best_score_))
```

OUTPUT:

Tuned Decision Tree Parameters: {'criterion': 'entropy', 'max_depth': None, 'max_features': 7, 'min_samples_leaf': 8}

Best score is 0.827

RESULT:

Thus the program for deep learning model by fine tuning hyper parameters was executed successfully

Ex. No:7 Date:26-2-2024	**IMPLEMENT A TRANSFER LEARNING CONCEPT IN IMAGE CLASSIFICATION**

AIM:

To write a python program to implement a transfer learning concept in image classification.

ALGORITHM:

Step 1: Import tensorflow as tf.

Step 2: Define the class names and directory containing training images.

Step 3: Set up data augmentation parameters for training data.

Step 4: Load and augment training data using flow_from_directory.

Step 5: Load the pre-trained VGG16 model (excluding the top layer) and freeze some layers.

Step 6: Add custom classification layers on top of the VGG16 base model.

Step 7: Compile and train the model and Save the trained model. Step 8: Use the model for predictions on a sample image.

PROGRAM:

```
import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import ResNet50

import numpy as np

import matplotlib.pyplot as plt

class_names = ['Cats', 'Dogs'] # Update with your actual class names

train_dir = r'C:\Users\LENOVO\PycharmProjects\nn\train'

train_datagen = ImageDataGenerator(

rescale=1./255,

rotation_range=40,

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_dir,
target_size=(224, 224), # ResNet50 input size
batch_size=32,
class_mode='categorical'
)
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
layer.trainable = False
x = layers.GlobalAveragePooling2D()(base_model.output)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
predictions = layers.Dense(len(class_names), activation='softmax')(x)
transfer_model = models.Model(inputs=base_model.input, outputs=predictions)
transfer_model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
transfer_model.summary()
print("Training started...")
history = transfer_model.fit(train_generator, epochs=10)
print("Training completed.")
print("Saving the model...")
transfer_model.save(r'C:\Users\LENOVO\PycharmProjects\nn\transfer_learning_resnet50_model.h5')
print("Model saved successfully.")
print("Making predictions...")
```

```
img_path = r'C:\Users\LENOVO\PycharmProjects\nn\pet.jpg' # Update with the path to the image you want to classify

img = tf.keras.preprocessing.image.load_img(img_path, target_size=(224, 224)) # Resize images to match the input size expected by ResNet50

img_array = tf.keras.preprocessing.image.img_to_array(img)

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

img_array /= 255.0 # Normalize pixel values to [0, 1]

predictions = transfer_model.predict(img_array)

predicted_class = np.argmax(predictions[0])

 predicted_class_name = class_names[predicted_class]

plt.imshow(img)

plt.axis('off')

plt.title('Predicted Class: {}'.format(predicted_class_name))

plt.show()

print("Prediction completed.")
```

OUTPUT:

```
C:\Users\LENOVO\PycharmProjects\nn\venv\Scripts\python.exe C:\Users\LENOVO\PycharmProjects\nn\nnex7.py
2024-03-22 22:56:09.763832: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off
2024-03-22 22:56:10.342297: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off
Found 10 images belonging to 2 classes.
2024-03-22 22:56:11.891333: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical op
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1,792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36,928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73,856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147,584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295,168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590,080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590,080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2,097,408
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 2)	514

```
 Total params: 16,812,610 (64.14 MB)
 Trainable params: 2,097,922 (8.00 MB)
 Non-trainable params: 14,714,688 (56.13 MB)
Training started...
Epoch 1/10
C:\Users\LENOVO\PycharmProjects\nn\venv\lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:120: UserWarning: Your `PyDataset` class should call `super().__i
  self._warn_if_super_not_called()
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9000 - loss: 0.4111
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 401ms/step - accuracy: 0.5000 - loss: 2.5024
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step - accuracy: 0.6000 - loss: 1.6607
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step - accuracy: 0.6000 - loss: 0.8565
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 0.8000 - loss: 0.3384
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step - accuracy: 0.8000 - loss: 0.3828
Epoch 7/10
```

```
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 401ms/step - accuracy: 0.5000 - loss: 2.5024
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step - accuracy: 0.6000 - loss: 1.6607
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step - accuracy: 0.6000 - loss: 0.8565
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 0.8000 - loss: 0.3384
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step - accuracy: 0.8000 - loss: 0.3828
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step - accuracy: 0.7000 - loss: 0.6102
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 1.0000 - loss: 0.0953
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 476ms/step - accuracy: 0.8000 - loss: 0.4273
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 421ms/step - accuracy: 0.9000 - loss: 0.2762
Training completed.
Saving the model...
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead
Model saved successfully.
Making predictions...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step
```

RESULT:

Thus the python program for a transfer learning concept in image classification was executed successfully.

Ex. No:8 Date:26-2-2024	USING A PRE TRAINED MODEL ON KERAS FOR TRANSFER LEARNING

AIM:

To write a python program to use a pre trained model on keras for transfer learning.

ALGORITHM:

Step 1: Import the necessary libraries.

Step 2: Define the class names (in this case, 'Cats' and 'Dogs') and specify the directory containing the Training images.

Step 3: Define data augmentation parameters using ImageDataGenerator to augment the training data.

Step 4: Use flow_from_directory to load and augment the training images from the specified directory.

Step 5: Load the pre-trained VGG16 model from Keras applications, excluding its top layer (fully connected Layers.

Step 6: Optionally, freeze some layers of the base VGG16 model to prevent their weights from being updated during training.

Step 7: Add custom layers on top of the VGG16 base model to adapt it to the binary classification task.

Step 8: Create a new model using models. Model with the VGG16 base model's input and the custom classification layers as output.

Step 9: Compile the transfer learning model using Adam optimizer, binary cross-entropy loss function for binary classification, and accuracy as the metric.

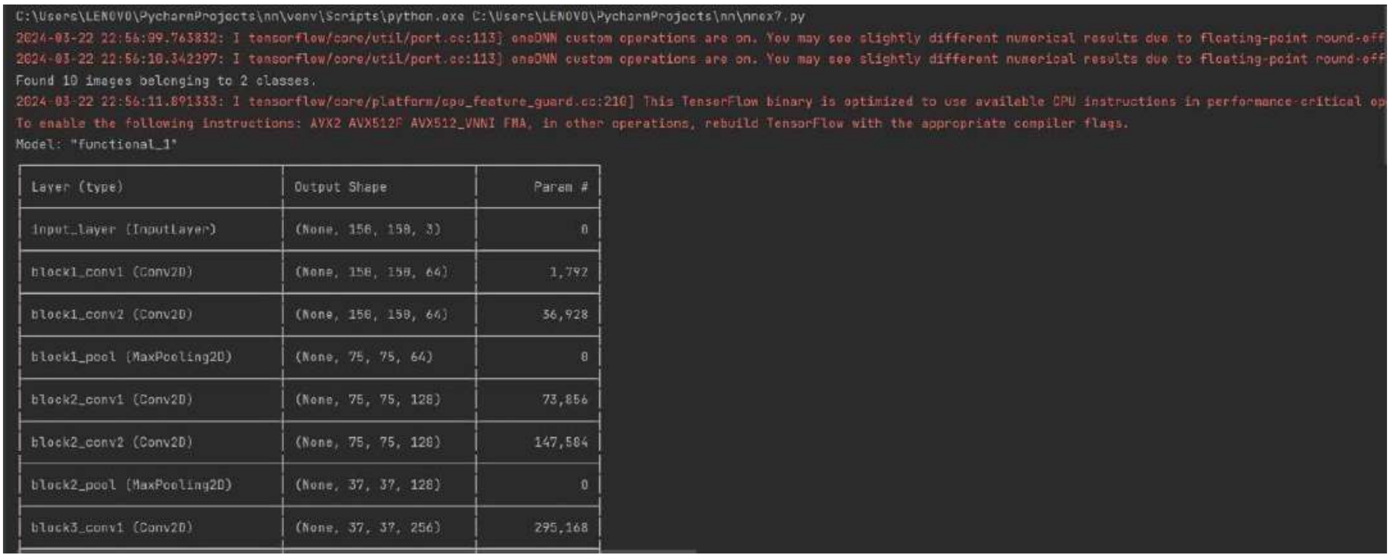
Step 10: Save and Display the image along with the predicted class name to visualize the classification result.

PROGRAM:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
class_names = ['Cats', 'Dogs']
train_dir = r'C:\Users\LENOVO\PycharmProjects\nn\train'
train_datagen = ImageDataGenerator(
rescale=1./255,
rotation_range=40,
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_dir,
target_size=(150, 150),
batch_size=32,
class_mode='binary' # Use 'binary' for binary classification
)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
for layer in base_model.layers:
layer.trainable = False
```

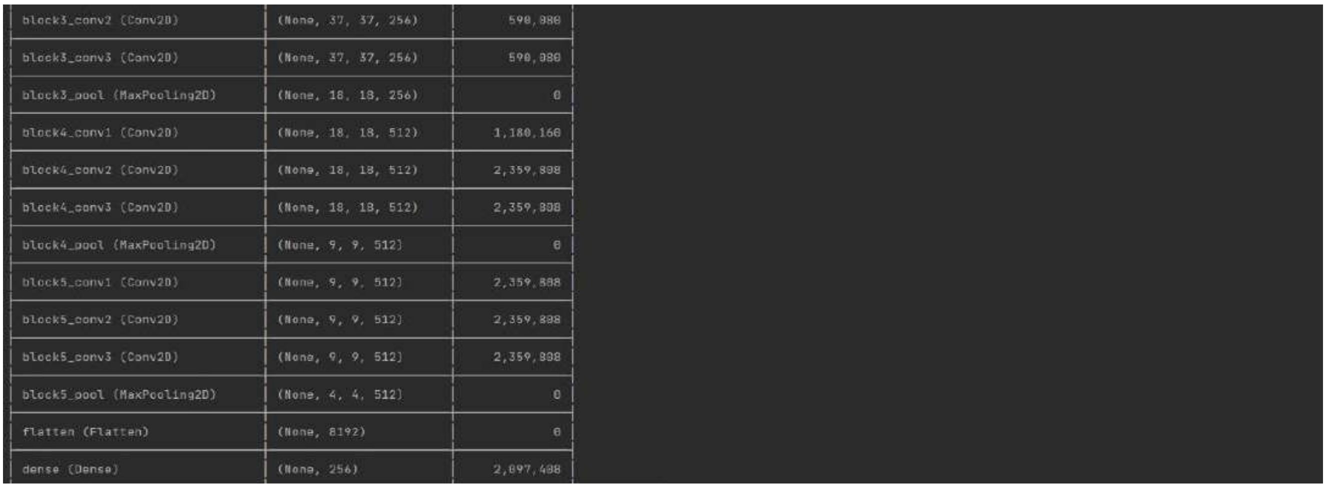
```
x = layers.Flatten()(base_model.output)

x = layers.Dense(256, activation='relu')(x)

x = layers.Dropout(0.5)(x)

predictions = layers.Dense(1, activation='sigmoid')(x) # Binary classification, so 1 output neuron with sigmoid activation

transfer_model = models.Model(inputs=base_model.input, outputs=predictions)

transfer_model.compile(optimizer='adam',

loss='binary_crossentropy',

metrics=['accuracy'])

transfer_model.summary()

print("Training started...")

history = transfer_model.fit(train_generator, epochs=10)

print("Training completed.")

print("Saving the model...")

transfer_model.save(r'C:\Users\LENOVO\PycharmProjects\nn\transfer_learning_model1.keras')

print("Model saved successfully.")

img_path = r'C:\Users\LENOVO\PycharmProjects\nn\pet.jpg'

img = image.load_img(img_path, target_size=(150, 150))

img_array = image.img_to_array(img)

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

img_array /= 255.0 # Normalize pixel values to [0, 1]

print("Making predictions...")

predictions = transfer_model.predict(img_array)

predicted_class = predictions[0][0] # Since it's binary, you can directly take the first element of the prediction array

predicted_class_name = class_names[int(predicted_class)] # Convert the predicted class to its name plt.imshow(img)

plt.axis('off')

plt.title('Predicted Class: {}'.format(predicted_class_name))

plt.show()
```

OUTPUT:



```
C:\Users\LENOVO\PycharmProjects\nn\venv\Scripts\python.exe C:\Users\LENOVO\PycharmProjects\nn\nnex7.py
2024-03-22 22:56:09.763832: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off
2024-03-22 22:56:10.342297: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off
Found 10 images belonging to 2 classes.
2024-03-22 22:56:11.891333: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical op
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1,792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36,928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73,856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147,584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295,168



Layer (type)	Output Shape	Param #
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590,080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590,080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2,097,408

```
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 401ms/step - accuracy: 0.5000 - loss: 2.5024
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step - accuracy: 0.6000 - loss: 1.6607
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step - accuracy: 0.6000 - loss: 0.8565
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 0.8000 - loss: 0.3384
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step - accuracy: 0.8000 - loss: 0.3828
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step - accuracy: 0.7000 - loss: 0.6102
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 1.0000 - loss: 0.0953
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 476ms/step - accuracy: 0.8000 - loss: 0.4273
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 421ms/step - accuracy: 0.9000 - loss: 0.2762
Training completed.
Saving the model...
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead
Model saved successfully.
Making predictions...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step
```

RESULT:

Thus the python program for pre-trained model on keras for transfer learning was executed successfully.

Ex. No:9 **Date:27-2-2024**	**PERFORM SENTIMENT ANALYSIS USING RNN**

AIM:

To write a python program to perform sentiment analysis using RNN.

ALGORITHM:

Step 1: Import necessary libraries.

Step 2: Define the class names (in this case, 'Cats' and 'Dogs') and specify the directory containing the training images.

Step 3: Define data augmentation parameters using ImageDataGenerator to augment the training data.

Step 4: Use flow_from_directory to load and augment the training images from the specified directory.

Step 5: Load the pre-trained VGG16 model from Keras applications, excluding its top layer (fully connected layers).

Step 6: Optionally, freeze some layers of the base VGG16 model to prevent their weights from being updated during training.

Step 7: Add custom layers on top of the VGG16 base model to adapt it to the binary classification task.

Step 8: Compile the transfer learning model using Adam optimizer, binary cross-entropy loss function for binary classification, and accuracy as the metric.

Step 9: train the model using fit with the augmented training data generator and a specified number of epochs.

Step10: Display the image along with the predicted class name to visualize the classification result.

PROGRAM:

ACCURACY:

```
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
max_features = 10000
maxlen = 500
batch_size = 32
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')
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)
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())
print('Training...')
history = model.fit(x_train, y_train, epochs=10, batch_size=batch_size, validation_split=0.2)
print('Evaluating...')
```

```
loss, accuracy = model.evaluate(x_test, y_test)

print('Test Loss:', loss)

print('Test Accuracy:', accuracy)
```

OUTPUT:

```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
x_train shape: (25000, 500)
x_test shape: (25000, 500)
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, None, 32)          320000

 simple_rnn_3 (SimpleRNN)    (None, 32)                2080

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 322113 (1.23 MB)
Trainable params: 322113 (1.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Training...
Epoch 1/10
625/625 [==============================] - 69s 108ms/step - loss: 0.6262 - acc: 0.6191 - val_loss: 0.4342 - val_acc: 0.8048
Epoch 2/10
625/625 [==============================] - 66s 105ms/step - loss: 0.3723 - acc: 0.8397 - val_loss: 0.3613 - val_acc: 0.8434
Epoch 3/10
625/625 [==============================] - 68s 109ms/step - loss: 0.2969 - acc: 0.8824 - val_loss: 0.3644 - val_acc: 0.8528
Epoch 4/10
```

```
Epoch 4/10
625/625 [==============================] - 66s 106ms/step - loss: 0.2458 - acc: 0.9046 - val_loss: 0.3540 - val_acc: 0.8556
Epoch 5/10
625/625 [==============================] - 68s 109ms/step - loss: 0.2181 - acc: 0.9173 - val_loss: 0.3886 - val_acc: 0.8466
Epoch 6/10
625/625 [==============================] - 68s 109ms/step - loss: 0.1803 - acc: 0.9321 - val_loss: 0.4272 - val_acc: 0.8414
Epoch 7/10
625/625 [==============================] - 66s 105ms/step - loss: 0.1498 - acc: 0.9449 - val_loss: 0.4374 - val_acc: 0.8298
Epoch 8/10
625/625 [==============================] - 68s 108ms/step - loss: 0.1109 - acc: 0.9592 - val_loss: 0.5212 - val_acc: 0.8188
Epoch 9/10
625/625 [==============================] - 66s 105ms/step - loss: 0.0948 - acc: 0.9676 - val_loss: 0.6003 - val_acc: 0.8120
Epoch 10/10
625/625 [==============================] - 68s 108ms/step - loss: 0.0812 - acc: 0.9722 - val_loss: 0.5837 - val_acc: 0.8220
Enter a movie review (type 'exit' to quit): I hate this movie
1/1 [==============================] - 0s 183ms/step
Negative Sentiment
Enter a movie review (type 'exit' to quit): I like this movie
1/1 [==============================] - 0s 41ms/step
Positive Sentiment
Enter a movie review (type 'exit' to quit):
```

RESULT:

Thus the program for sentiment analysis using RNN was executed successfully.

Ex. No:10 Date:27-2-2024	**IMPLEMENT AN LSTM BASED AUTOENCODER IN TENSORFLOW/KERAS**

AIM:

To write a python program to implement an LSTM based auto-encoder tensorflow/keras.

ALGORITHM:

Step 1: Import necessary libraries.

Step 2: Create random data for demonstration purposes. The data consists of 1000 sequences, each of length 10, with 1 feature.

Step 3: Set the dimensionality of the latent space (latent_dim) to 2.

Step 4: Use an LSTM layer with 4 units for encoding the input sequences (encoded).

Step 5: Decode the repeated representation using another LSTM layer with 4 units and a time-distributed dense layer to reconstruct the original input shape.

Step 6: Create the auto-encoder model using the defined input and output layers and Compile the auto-encoder model with the Adam optimizer and mean squared error (MSE) loss function.

Step 7: Print a summary of the auto-encoder model to review its architecture.

PROGRAM:

```
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
data = np.random.rand(1000, 10, 1) feature
latent_dim = 2 inputs = Input(shape=(10, 1))
encoded = LSTM(4)(inputs)
encoded = RepeatVector(10)(encoded)
decoded = LSTM(4, return_sequences=True)(encoded)
decoded = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(1))(decoded)
 autoencoder = Model(inputs, decoded)
 autoencoder.compile(optimizer='adam',  loss='mse')
autoencoder.summary()
autoencoder.fit(data, data, epochs=50, batch_size=32, validation_split=0.2)
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)
```

OUTPUT:

```
Model: "model_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 10, 1)]           0

 lstm_6 (LSTM)               (None, 4)                 96

 repeat_vector_3 (RepeatVec  (None, 10, 4)             0
 tor)

 lstm_7 (LSTM)               (None, 10, 4)             144

 time_distributed_3 (TimeDi  (None, 10, 1)             5
 stributed)

=================================================================
Total params: 245 (980.00 Byte)
Trainable params: 245 (980.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Enter a sequence of 10 numbers separated by spaces (type 'exit' to quit): 3 5 7 4 77 4 5 9 1 22
1/1 [==============================] - 1s 653ms/step
1/1 [==============================] - 0s 18ms/step
Original Sequence: [[[ 3.]
  [ 5.]
  [ 7.]
  [ 4.]
  [77.]
  [ 4.]
```

```
Original Sequence: [[[ 3.]
  [ 5.]
  [ 7.]
  [ 4.]
  [77.]
  [ 4.]
  [ 5.]
  [ 9.]
  [ 1.]
  [22.]]]
Encoded Sequence: [[[ 0.11506256]
  [ 0.13293919]
  [ 0.10644364]
  [ 0.06383099]
  [ 0.01806891]
  [-0.02500868]
  [-0.06302401]
  [-0.09534584]
  [-0.12220283]
  [-0.1441995 ]]]
Decoded Sequence: [[[-0.00644221]
  [-0.00818746]
  [-0.00692648]
  [-0.00382397]
  [ 0.00032244]
  [ 0.00497115]
  [ 0.00975966]
  [ 0.01445099]
  [ 0.01889618]
  [ 0.0230079 ]]]
Enter a sequence of 10 numbers separated by spaces (type 'exit' to quit): exit
```

RESULT:

Thus the program for an LSTM (Long short-Term Memory) based auto-encoder was executed successfully.

Ex. No:11 **Date:27-2-2024**	**IMAGE GENERATION USING GAN**

AIM:

To write a python program to implement image generation using GAN.

ALGORITHM:

Step 1: Import TensorFlow, Keras, and NumPy.

Step 2: Load the MNIST dataset and preprocess it by normalizing the pixel values to the range [-1, 1] and adding a channel dimension.

Step 3: Create the generator model using a Sequential model with layers for dense, reshape, and transpose convolution operations.

Step 4: Create the discriminator model using another Sequential model with convolution layers followed by a dense layer for binary classification.

Step 5: Compile the discriminator model with binary cross-entropy loss and the Adam optimizer.

Step 6: Set discriminator. trainable = False to freeze the discriminator's weights during GAN training.

Step 7: Create the GAN model by connecting the generator and discriminator in a sequential manner.

Step 8: Compile the GAN model with binary cross-entropy loss and the Adam optimizer.

Step 9: END.

PROGRAM:

```
import tensorflow as tf

from tensorflow import keras

import numpy as np

(X_train, _), (_, _) = keras.datasets.mnist.load_data()

X_train = (X_train.astype(np.float32) - 127.5) / 127.5 # Normalize to [-1, 1]

X_train = np.expand_dims(X_train, axis=-1)

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 = 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')

])

discriminator.compile(loss='binary_crossentropy',

optimizer=keras.optimizers.Adam(learning_rate=0.0002),

metrics=['accuracy'])

discriminator.trainable = False

gan_input = keras.Input(shape=(100,))

generated_image = generator(gan_input)
```

```
gan_output = discriminator(generated_image)
gan = keras.Model(gan_input, gan_output)
gan.compile(loss='binary_crossentropy',
optimizer=keras.optimizers.Adam(learning_rate=0.0002))
batch_size = 64
epochs = 10
sample_interval = 1000
for epoch in range(epochs):
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)
real_labels = np.ones((batch_size, 1))
fake_labels = np.zeros((batch_size, 1))
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)
noise = np.random.normal(0, 1, (batch_size, 100))
g_loss = gan.train_on_batch(noise, real_labels)
if epoch % sample_interval == 0:
print(f'Epoch {epoch}, D Loss: {d_loss[0]}, G Loss: {g_loss}')
_, accuracy = discriminator.evaluate(np.concatenate([real_images, fake_images]),
 np.concatenate([real_labels, fake_labels]), verbose=0)
print(f"Discriminator Accuracy: {accuracy:.4f}")
```

OUTPUT:

```
Epoch 0, D Loss: 0.7081464231014252, G Loss: 0.6910318732261658
Discriminator Accuracy: 0.5625
```

RESULT:

Thus the program for image generation using GAN (Generative Adversarial Network) was executed successfully.

Ex. No:12 Date:27-2-2024	**TRAIN A DEEP LEARNING MODEL TO CLASSIFY A GIVEN IMAGE USING PRE TRAINED MODEL**

AIM:

To write a python program to train a deep learning model to classify a given image using pre trained model.

ALGORITHM:

Step 1: Import the necessary libraries.

Step 2: Mount your Google Drive to access the data.

Step 3: Set the directory where your data is located.

Step 4: Load the VGG16 model pre-trained on ImageNet, excluding the fully connected layers.

Step 5: Freeze the weights of the pre-trained layers so they are not updated during training.

Step 6: Create a new Sequential model and add the pre-trained VGG16 model as a layer.

Step 7: Flatten the output of VGG16 and add fully connected layers for classification, including dropout layers for regularization.

Step 8: Flatten the output of VGG16 and add fully connected layers for classification, including dropout layers for regularization.

Step 9: Set the number of classes in your dataset and add an output layer with softmax activation for multi- class classification.

Step 10: Compile the model with the Adam optimizer, categorical cross-entropy loss for multi-class classification, and accuracy as a metric.

Step 11: Use ImageDataGenerator to load and preprocess the data, rescaling pixel values to the range [0, 1].

Step 12: Create data generators for training and validation data, specifying target size, batch size, class model, and shuffle parameters.

Step 13: Train the model using model and Evaluate the model on the validation data using model.

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

from google.colab import drive

drive.mount('/content/drive')

data_dir = '/content/drive/MyDrive/Collab'

vgg_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in vgg_model.layers:

layer.trainable = False

model = Sequential()

model.add(vgg_model)

model.add(Flatten())

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

model.add(Dropout(0.5))

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

model.add(Dropout(0.5))

num_classes = 2

model.add(Dense(num_classes, activation='softmax'))

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

train_data_dir = data_dir + '/train'

validation_data_dir = 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 = train_generator.class_indices
print("Class labels:", class_labels)
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
)
validation_loss, validation_accuracy = model.evaluate(validation_generator)
print("Validation Accuracy:", validation_accuracy)
```

OUTPUT:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 72 images belonging to 2 classes.
Found 22 images belonging to 2 classes.
Class labels: {'cats': 0, 'dogs': 1}
Epoch 1/10
2/2 [==============================] - 27s 5s/step - loss: 0.8236 - accuracy: 0.5000
Epoch 2/10
2/2 [==============================] - 40s 21s/step - loss: 1.0687 - accuracy: 0.5938
Epoch 3/10
2/2 [==============================] - 24s 5s/step - loss: 0.7461 - accuracy: 0.6000
Epoch 4/10
2/2 [==============================] - 25s 19s/step - loss: 0.7508 - accuracy: 0.5500
Epoch 5/10
2/2 [==============================] - 26s 21s/step - loss: 0.7140 - accuracy: 0.7000
Epoch 6/10
2/2 [==============================] - 40s 20s/step - loss: 0.7749 - accuracy: 0.6250
Epoch 7/10
2/2 [==============================] - 25s 20s/step - loss: 0.6645 - accuracy: 0.6750
Epoch 8/10
2/2 [==============================] - 24s 5s/step - loss: 0.4762 - accuracy: 0.7500
Epoch 9/10
2/2 [==============================] - 24s 19s/step - loss: 0.4926 - accuracy: 0.7500
Epoch 10/10
2/2 [==============================] - 40s 21s/step - loss: 0.3696 - accuracy: 0.8438
1/1 [==============================] - 14s 14s/step - loss: 0.2537 - accuracy: 0.8182
Validation Accuracy: 0.8181818127632141
```

RESULT:

Thus the program for deep learning model to classify a given image using pre trained model was executed successfully

Ex. No:13 Date:19-3-2024	RECOMMENDATION SYSTEM FROM SALES DATA USING DEEP LEARNING

AIM:

To write a python program to build the recommendation system from sales data using deep learning.

ALGORITHM:

Step 1: Import TensorFlow and NumPy.

Step 2: Define the number of users, items, and samples.

Step 3: Generate random user IDs, item IDs, and ratings for training, validation, and testing sets.

Step 4: Create a class Collaborative Filtering Model that inherits from tf.keras.Model.

Step 5: Implement the call method in Collaborative Filtering Model to compute the dot product of user and item embeddings.

Step 6: Create an instance of Collaborative Filtering Model with the specified number of users, items, and embedding size.

Step 7: Use the fit method to train the model on the training data and Use the evaluate method to evaluate the model on the test data.

Step 8: Print the test loss.

PROGRAM:

```
import tensorflow as tf

import numpy as np

num_users = 1000

num_items = 500

num_samples = 10000

user_ids_train = np.random.randint(0, num_users, num_samples)

item_ids_train = np.random.randint(0, num_items, num_samples)

ratings_train = np.random.randint(1, 6, num_samples) # Assume ratings are integers between
1 and 5 user_ids_val = np.random.randint(0, num_users, num_samples)

item_ids_val = np.random.randint(0, num_items, num_samples)

ratings_val = np.random.randint(1, 6, num_samples)

user_ids_test = np.random.randint(0, num_users, num_samples)

item_ids_test = np.random.randint(0, num_items, num_samples)

ratings_test = np.random.randint(1, 6, num_samples)

class CollaborativeFilteringModel(tf.keras.Model):

def  init (self, num_users, num_items, embedding_size):

super(CollaborativeFilteringModel, self). init ()

self.user_embedding = tf.keras.layers.Embedding(num_users, embedding_size)

self.item_embedding = tf.keras.layers.Embedding(num_items, embedding_size)

self.dot = tf.keras.layers.Dot(axes=1)

def call(self, inputs):

user_id, item_id = inputs

user_embedding = self.user_embedding(user_id)

item_embedding = self.item_embedding(item_id)

return self.dot([user_embedding, item_embedding])

embedding_size = 50

model = CollaborativeFilteringModel(num_users, num_items, embedding_size)

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

history = model.fit([user_ids_train, item_ids_train], ratings_train,
```

```
validation_data=([user_ids_val, item_ids_val], ratings_val),
epochs=10, batch_size=64)
loss = model.evaluate([user_ids_test, item_ids_test], ratings_test)
print("Test Loss:", loss)
```

OUTPUT:

```
Epoch 1/10
157/157 [==============================] - 1s 5ms/step - loss: 11.0114 - val_loss: 11.0653
Epoch 2/10
157/157 [==============================] - 1s 4ms/step - loss: 10.9318 - val_loss: 11.0413
Epoch 3/10
157/157 [==============================] - 1s 4ms/step - loss: 10.7215 - val_loss: 10.8739
Epoch 4/10
157/157 [==============================] - 1s 4ms/step - loss: 10.1070 - val_loss: 10.2295
Epoch 5/10
157/157 [==============================] - 1s 4ms/step - loss: 8.7497 - val_loss: 8.8200
Epoch 6/10
157/157 [==============================] - 1s 5ms/step - loss: 6.7104 - val_loss: 6.8815
Epoch 7/10
157/157 [==============================] - 1s 5ms/step - loss: 4.5827 - val_loss: 5.0574
Epoch 8/10
157/157 [==============================] - 1s 3ms/step - loss: 2.9915 - val_loss: 3.7981
Epoch 9/10
157/157 [==============================] - 1s 5ms/step - loss: 2.0817 - val_loss: 3.1121
Epoch 10/10
157/157 [==============================] - 1s 3ms/step - loss: 1.6313 - val_loss: 2.7936
313/313 [==============================] - 1s 2ms/step - loss: 2.6849
Test Loss: 2.6848864555358887
```

RESULT:

Thus the program for recommendation system from sales data using deep learning was executed successfully.

Ex. No:14 Date:19-3-2024	IMPLEMENT OBJECT DETECTION USING CNN

AIM:

To write a python program to implement object detection using CNN.

ALGORITHM:

Step 1: import necessary libraries.

Step 2: Load MNIST data and preprocess.

Step 3: Set the grid size for the mask and define functions for creating colored digits and data.

Step 4: Pick a random digit from the MNIST dataset then Make the digit colorful by multiplying it with random values.

Step 5: Create empty arrays for images (X) and labels (y).Call the make_numbers function to populate X and y with colorful digits and their respective labels.

Step 6: Define a function to assign colors based on probabilities. Step 7: Define a function show_predict to visualize predictions.

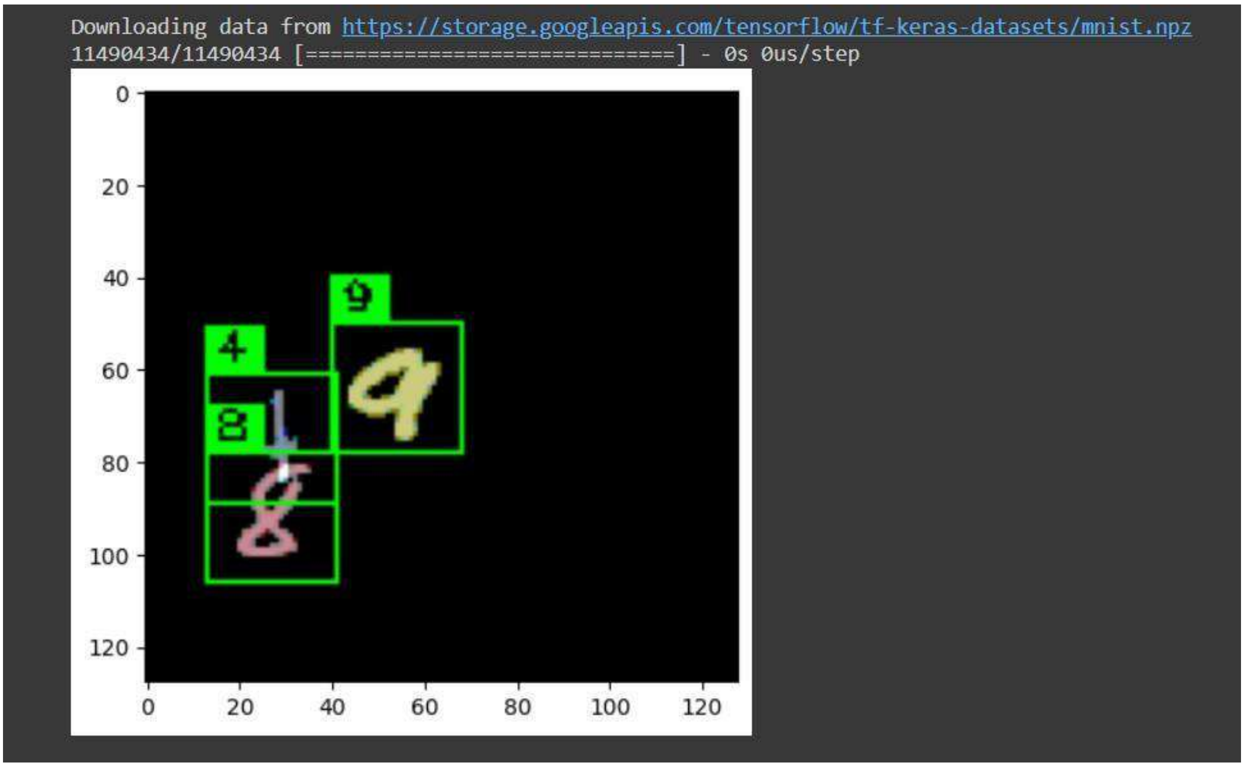
Step 8: Show predictions for the generated sample using the show_predict function

PROGRAM:

```
import numpy as np

import tensorflow as tf

import cv2

import matplotlib.pyplot as plt

(X_num, y_num), _ = tf.keras.datasets.mnist.load_data()

X_num = np.expand_dims(X_num, axis=-1).astype(np.float32) / 255.0

grid_size = 16 # image_size / mask_size

def make_numbers(X, y):

for _ in range(3):

idx = np.random.randint(len(X_num))

number = X_num[idx] @ (np.random.rand(1, 3) + 0.1) # Make digit colorful

kls = y_num[idx]

px, py = np.random.randint(0, 100), np.random.randint(0, 100)

mx, my = (px+14) // grid_size, (py+14) // grid_size

channels = y[my][mx]

if channels[0] > 0:

channels[0] = 1.0

channels[1] = px - (mx * grid_size) # x1

channels[2] = py - (my * grid_size) # y1

channels[3] = 28.0 # x2, in this demo image only 28 px as width

channels[4] = 28.0 # y2, in this demo image only 28 px as height

channels[5 + kls] = 1.0

X[py:py+28, px:px+28] += number

def make_data(size=64):

X = np.zeros((size, 128, 128, 3), dtype=np.float32)

y = np.zeros((size, 8, 8, 15), dtype=np.float32)

for i in range(size):

make_numbers(X[i], y[i])
```

```
X = np.clip(X, 0.0, 1.0)

return X, y

def get_color_by_probability(p):

if p < 0.3:

return (1., 0., 0.)

if p < 0.7:

return (1., 1., 0.)

return (0., 1., 0.)

def show_predict(X, y, threshold=0.1):

X = X.copy()

for mx in range(8):

for my in range(8):

channels = y[my][mx]

prob, x1, y1, x2, y2 = channels[:5]

if prob < threshold:

continue

color = get_color_by_probability(prob)

px, py = (mx * grid_size) + x1, (my * grid_size) + y1

cv2.rectangle(X, (int(px), int(py)), (int(px + x2), int(py + y2)), color, 1)

cv2.rectangle(X, (int(px), int(py - 10)), (int(px + 12), int(py)), color, -1)

kls = np.argmax(channels[5:])

cv2.putText(X, f'{kls}', (int(px + 2), int(py-2)), cv2.FONT_HERSHEY_PLAIN, 0.7, (0.0,
0.0, 0.0))

plt.imshow(X)

X, y = make_data(size=1)

show_predict(X[0], y[0])

plt.show()
```

OUTPUT:



RESULT:

Thus the python program for object detection using CNN (Convolutional Neural Network) was executed successfully

Ex. No:15 Date:19-3-2024	IMPLEMENT ANY SIMPLE REINFORCEMENT ALGORITHM FOR AN NLP PROBLEM

AIM:

To write a python program to implement any simple reinforcement algorithm for an NLP problem.

ALGORITHM:

Step 1: Initialize Q-learning parameters: num_states, num_actions, Q-table, alpha, gamma, epsilon.

Step 2: Define environment simulation: simulate_environment(state, action).

Step 3: Implement Q-learning algorithm:

a.Define train_q_learning(num_episodes) function.

b.Loop for each .

Step 4: Interactive dialogue interface, Define interactive_dialogue() function.

Step 5: Train the Q-learning model;Define num_episodes for training episodes.

Step 6: Start interactive dialogue,Call interactive_dialogue() function to begin interactive dialogue system.

PROGRAM:

```
import numpy as np

num_states = 10

num_actions = 10

Q = np.zeros((num_states, num_actions))

alpha = 0.1

gamma = 0.9

epsilon = 0.1

def simulate_environment(state, action):

reward = 0

next_state = (state + action) % num_states

return next_state, reward

def train_q_learning(num_episodes):

for episode in range(num_episodes):

state = np.random.randint(0, num_states)

for _ in range(num_states):

if np.random.uniform(0, 1) &lt; epsilon:

action = np.random.randint(0, num_actions) # Exploration

else:

action = np.argmax(Q[state, :])

next_state, reward = simulate_environment(state, action)

Q[state, action] = (1 - alpha) * Q[state, action] + alpha * (reward + gamma *
np.max(Q[next_state, :]))

state = next_state

def generate_response(state):

action = np.argmax(Q[state, :])

return action

def interactive_dialogue():

print(&quot;Welcome to the dialogue system!&quot;)

print(&quot;Enter your dialogue context (an integer between 0 and 9):&quot;)
```

```
while True:
try:
context = int(input())
if 0 &lt;= context &lt; num_states:
response_action = generate_response(context)
print(&quot;Generated response action:&quot;, response_action)
else:
print(&quot;Context should be an integer between 0 and 9.&quot;)
except ValueError:
print(&quot;Invalid input. Please enter an integer.&quot;)
num_episodes = 1000
train_q_learning(num_episodes)
interactive_dialogue()
```

OUTPUT:

```
Welcome to the dialogue system!
Enter your dialogue context (an integer between 0 and 9):
hi
Invalid input. Please enter an integer.
7
Generated response action: 0
45
Context should be an integer between 0 and 9.
0
Generated response action: 0
&*
Invalid input. Please enter an integer.
```

RESULT:

Thus the python program for reinforcement algorithm for an NLP problem was executed successfully.