Exp No: 3 **BUILD A RECURRENT NEURAL NETWORK**

Aim:

To build a recurrent neural network with Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Program:

```
#Importing Libraries
import pandas as pd
import glob
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import glob

CATS = ['Abyssinian', 'Bengal', 'Birman', 'Bombay', 'British_Shorthair', 'Egyptian_Mau',
'Maine_Coon', 'Persian', 'Ragdoll', 'Russian_Blue', 'Siamese', 'Sphynx']

cats_images = []
dogs_images = []

for img in glob.glob(path):
    if any(cat in img for cat in CATS):
        cats_images.append(img)
    else:
        dogs_images.append(img)
```

```
print('There are {} images of cats'.format(len(cats_images)))
print('There are {} images of dogs'.format(len(dogs_images)))

#Splitting Data into Sets
#shuffle the lists
np.random.shuffle(cats_images)
np.random.shuffle(dogs_images)

#split the data into train, validation and test sets
train_d, val_d, test_d = np.split(dogs_images, [int(len(dogs_images)*0.7),
int(len(dogs_images)*0.8)])
train_c, val_c, test_c = np.split(cats_images, [int(len(cats_images)*0.7),
int(len(cats_images)*0.8)])

train_dog_df = pd.DataFrame({'image':train_d, 'label':'dog'})
val_dog_df = pd.DataFrame({'image':val_d, 'label':'dog'})
test_dog_df = pd.DataFrame({'image':test_d, 'label':'dog'})

train_cat_df = pd.DataFrame({'image':train_c, 'label':'cat'})
val_cat_df = pd.DataFrame({'image':val_c, 'label':'cat'})
test_cat_df = pd.DataFrame({'image':test_c, 'label':'cat'})

train_df = pd.concat([train_dog_df, train_cat_df])
val_df = pd.concat([val_dog_df, val_cat_df])
test_df = pd.concat([test_dog_df, test_cat_df])

print('There are {} images for training'.format(len(train_df)))
print('There are {} images for validation'.format(len(val_df)))
print('There are {} images for testing'.format(len(test_df)))

# Creating ImageDataGenerators
BATCH_SIZE = 32
IMG_HEIGHT = 224
IMG_WIDTH = 224

#create the ImageDataGenerator object and rescale the images
trainGenerator = ImageDataGenerator(rescale=1./255.)
valGenerator = ImageDataGenerator(rescale=1./255.)
testGenerator = ImageDataGenerator(rescale=1./255.)
```

```
#convert them into a dataset
trainDataset = trainGenerator.flow_from_dataframe(
  dataframe=train_df,
  class_mode="binary",
  x_col="image",
  y_col="label",
  batch_size=BATCH_SIZE,
  seed=42,
  shuffle=True,
  target_size=(IMG_HEIGHT,IMG_WIDTH) #set the height and width of the images
)

valDataset = valGenerator.flow_from_dataframe(
  dataframe=val_df,
  class_mode='binary',
  x_col="image",
  y_col="label",
  batch_size=BATCH_SIZE,
  seed=42,
  shuffle=True,
  target_size=(IMG_HEIGHT,IMG_WIDTH)
)

testDataset = testGenerator.flow_from_dataframe(
  dataframe=test_df,
  class_mode='binary',
  x_col="image",
  y_col="label",
  batch_size=BATCH_SIZE,
  seed=42,
  shuffle=True,
  target_size=(IMG_HEIGHT,IMG_WIDTH)
)

#Building and Training the Model
model = keras.Sequential([
    keras.layers.InputLayer(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
```

```
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(128, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(256, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(512, (3, 3), activation='relu'),
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dense(1, activation='sigmoid')
])

epochs=2

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(trainDataset, epochs=epochs, validation_data=(valDataset))

loss, acc = model.evaluate(testDataset)

print('Loss:', loss)
print('Accuracy:', acc)

#Preprocessing and Predicting on New Image
def preprocess(image):
    img_resize = tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH])
    img_norm = img_resize / 255.0
    return img_norm

# Load and preprocess the image
img_path = r'C:\Personal\Kavin\DLC\archive\images\Abyssinian_1.jpg'
img = plt.imread(img_path)

# Ensure the image has 3 channels (RGB). Convert grayscale to RGB if necessary
if img.ndim == 2:  # grayscale image
    img = np.expand_dims(img, axis=-1)  # add channel dimension
    img = np.concatenate([img]*3, axis=-1)  # convert to RGB by duplicating channels

# Convert to a Tensor and add batch dimension
img = tf.convert_to_tensor(img, dtype=tf.float32)

# Check the image shape before preprocessing
```

```
print("Image shape before preprocessing:", img.shape)

# Add batch dimension and preprocess
img = tf.expand_dims(img, axis=0)  # Add batch dimension
img = preprocess(img)

# Check the shape after preprocessing
print("Image shape after preprocessing:", img.shape)

# Perform prediction
predictions = model.predict(img)
print(predictions)

# Example class labels for a binary classification problem (adjust as needed)
class_names = ['cat', 'dog']

# Perform prediction
predictions = model.predict(img)

# Get the class index with the highest probability
predicted_class_index = np.argmax(predictions, axis=1)[0]

# Print the predicted class label
print("Predicted class:", class_names[predicted_class_index])
```

Output:

```
Epoch 1/2
C:\Users\kavin\AppData\Roaming\Python\Python311\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
162/162 ━━━━━━━━━━━━━━━━━━━━ 511s 3s/step - accuracy: 0.6781 - loss: 0.6162 - val_accuracy: 0.6752 - val_loss: 0.6163
Epoch 2/2
162/162 ━━━━━━━━━━━━━━━━━━━━ 467s 3s/step - accuracy: 0.6787 - loss: 0.6052 - val_accuracy: 0.6712 - val_loss: 0.6214

47/47 ━━━━━━━━━━━━━━━━━━━━ 42s 892ms/step - accuracy: 0.7011 - loss: 0.6127
Loss: 0.6158978939056396
Accuracy: 0.6928281188011169

Image shape before preprocessing: (400, 600, 3)
Image shape after preprocessing: (1, 224, 224, 3)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
[[0.49661055]]

Image shape before preprocessing: (400, 600, 3)
Image shape after preprocessing: (1, 224, 224, 3)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
[[0.49661055]]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
Predicted class: cat
```

Result:

CNN has been successfully built using the provided resources.