VELS

INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)

(Deemed to be University Estd. u/s 3 of the UGC Act, 1956)

PALLAVARAM - CHENNAI

ACCREDITED BY NAAC WITH 'A' GRADE

Marching Beyond 30 Years Successfully

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

21PBAI62-PRACTICAL – COGNITIVE LEARNING LABORATORY

YEAR 2023 -2024

NAME OF THE STUDENT :

REGISTER NUMBER :

COURSE :

YEAR :

SEMESTER :

VELS

VELS INSTITUTE OF SCIENCE, TECHNOLOGY AND ADVANCED STUDIES (VISTAS)
Deemed to be University Estd. U/S 3 of the UGC ACT, 1956
NAAC ACCREDITED WITH 'A' GRADE
PALLAVARAM, CHENNAI

BONAFIDE CERTIFICATE

Reg. No.

This is to certify that the Bonafide Record of this Practical Work was completed by Mr./Ms……………………………………… of **B.TECH COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)** in the Cognitive Learning Laboratory during the academic year of 2023 -2024.

HEAD OF THE DEPARTMENT **STAFF-IN-CHARGE**

Submitted for the Practical Examination held on …………………………..

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

S.NO	DATE	EXPERIMENT NAME	PAGE NO	MARKS	SIGNATURE
1		FACIAL RECOGNITION			
2		IMAGE RECOGNITION			
3		SPEECH RECOGNITION			
4		GESTURE RECOGNITION			
5		PATTERN RECOGNITION			
6		FEATURE COMPARISON MODELS			
7		LAB RECOGNITION			
8		NETWORKS MODELS			

Ex. No:1	**FACIAL RECOGNITION**

AIM:

PROCEDURE:

PROGRAM:

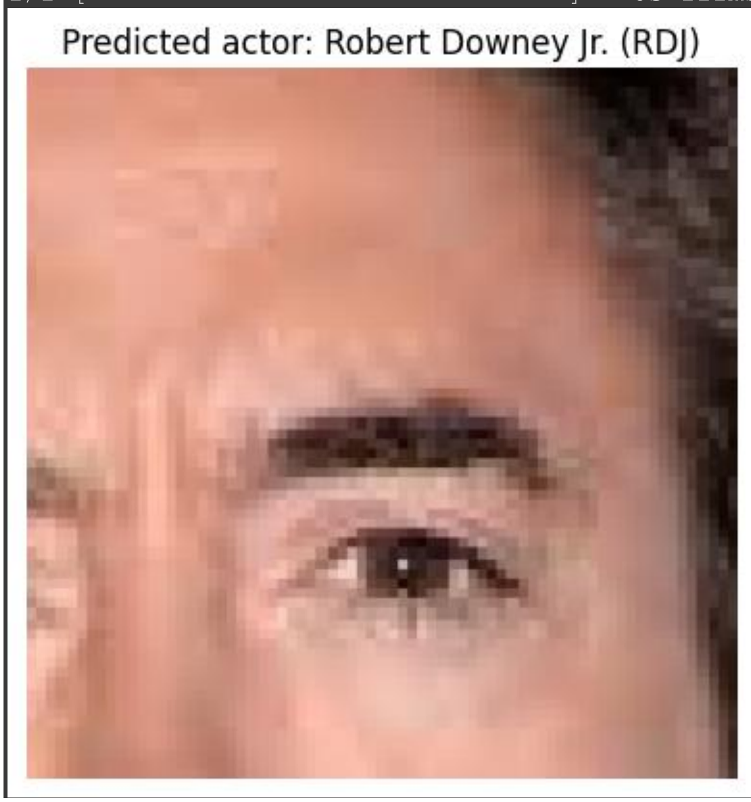
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
import os
# Path to your training dataset
train_data_dir = '/content/drive/MyDrive/Colab Notebooks/archive/cropped_images'
# Count the number of subdirectories (classes)
num_classes = len(os.listdir(train_data_dir))
# Define your model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 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(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))
# Compile the model
model.compile(optimizer=Adam(lr=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])
# Create an ImageDataGenerator for data augmentation and loading images
batch_size = 32
train_datagen = ImageDataGenerator(
rescale=1./255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(
train_data_dir,
target_size=(224, 224),
batch_size=batch_size,
class_mode='categorical')
# Train the model
epochs = 10 # You can adjust the number of epochs based on your needs
model.fit_generator(
train_generator,
steps_per_epoch=train_generator.samples // batch_size,
```

```
epochs=epochs)
# Save the model
model.save('facial_recognition_model.h5')
WARNING:absl:lr is deprecated in Keras optimizer, please use learning_rate or use the legacy
optimizer,
e.g.,tf.keras.optimizers.legacy.Adam.
Found 274 images belonging to 5 classes.
<ipython-input-3-99a3c70880cf>:49: UserWarning: Model.fit_generator is deprecated and will be
removed in a future version.
Please use Model.fit, which supports generators.
 model.fit_generator(
Epoch 1/10
8/8 [==============================] - 73s 8s/step - loss: 2.8260 - accuracy: 0.1942
Epoch 2/10
8/8 [==============================] - 32s 4s/step - loss: 1.5891 - accuracy: 0.2603
Epoch 3/10
8/8 [==============================] - 32s 4s/step - loss: 1.5098 - accuracy: 0.3223
Epoch 4/10
8/8 [==============================] - 31s 4s/step - loss: 1.3152 - accuracy: 0.4339
Epoch 5/10
8/8 [==============================] - 32s 4s/step - loss: 1.2362 - accuracy: 0.5165
Epoch 6/10
8/8 [==============================] - 32s 4s/step - loss: 1.2587 - accuracy: 0.5620
Epoch 7/10
8/8 [==============================] - 32s 4s/step - loss: 0.9988 - accuracy: 0.6446
Epoch 8/10
8/8 [==============================] - 31s 4s/step - loss: 0.8946 - accuracy: 0.6860
Epoch 9/10
8/8 [==============================] - 34s 4s/step - loss: 0.7615 - accuracy: 0.7190
Epoch 10/10
8/8 [==============================] - 33s 4s/step - loss: 0.7325 - accuracy: 0.7107
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are
saving your model as an
HDF5 file via model.save(). This file format is considered legacy. We recommend using instead
the native Keras format, e.g.
model.save('my_model.keras').
 saving_api.save_model(
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
# Load the trained model
model = load_model('/content/facial_recognition_model.h5')
```

```
# Map class indices to actor names
actor_mapping = {
0: 'Chris Evans',
1: 'Chris Hemsworth',
2: 'Mark Ruffalo',
3: 'Robert Downey Jr. (RDJ)',
4: 'Scarlett Johansson'
}
# Path to the image you want to predict
image_path = '/content/drive/MyDrive/Colab
Notebooks/archive/cropped_images/robert_downey_jr/robert_downey_jr1.png'
# Load and preprocess the image
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 /= 255.0 # Rescale to [0, 1]
# Make predictions
predictions = model.predict(img_array)
# Get the class with the highest probability
predicted_class = np.argmax(predictions)
# Get the predicted actor name
predicted_actor = actor_mapping.get(predicted_class, 'Unknown')
# Display the image
plt.imshow(img)
plt.axis('off')
plt.title(f"Predicted actor: {predicted_actor}")
plt.show()
```

OUTPUT:

```
1/1 [==============================] - 0s 111ms/step
```



RESULT:

Ex. No:2	**IMAGE RECOGNITION**

AIM:

PROCEDURE:

PROGRAM:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Define data directories
train_dir = '/content/drive/MyDrive/fruits'
validation_dir = '/content/drive/MyDrive/validation/validation 1'
# Image size and batch size
img_size = (64, 64)
batch_size = 32
# Data augmentation for training
train_datagen = ImageDataGenerator(
rescale=1./255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True
)
# Validation data should not be augmented
validation_datagen = ImageDataGenerator(rescale=1./255)
# Create data generators
train_generator = train_datagen.flow_from_directory(
train_dir,
target_size=img_size,
batch_size=batch_size,
class_mode='categorical'
)
validation_generator = validation_datagen.flow_from_directory(
validation_dir,
target_size=img_size,
batch_size=batch_size,
class_mode='categorical'
)
# Define the model architecture
model = tf.keras.Sequential([
tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(6, activation='softmax') # 6 classes, adjust as needed])
```

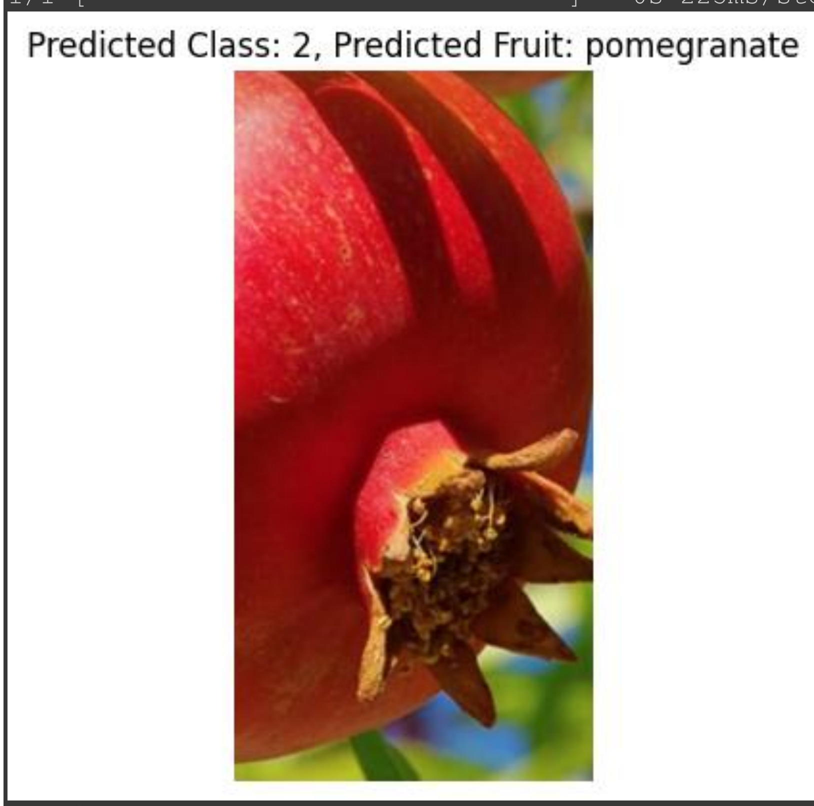
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(
train_generator,
steps_per_epoch=train_generator.samples // batch_size,
epochs=10, # Adjust as needed
validation_data=validation_generator,
validation_steps=validation_generator.samples // batch_size
)
# Save the trained model
model.save('/content/fruit_recognition_model_trained.h5')
Found 654 images belonging to 6 classes.
Found 60 images belonging to 6 classes.
Epoch 1/10
20/20 [==============================] - 49s 2s/step - loss: 1.5748 - accuracy: 0.3746 -
val_loss: 1.1222 - val_accuracy: 0.6250
Epoch 2/10
20/20 [==============================] - 33s 2s/step - loss: 1.0867 - accuracy: 0.5643 -
val_loss: 0.6546 - val_accuracy: 0.7188
Epoch 3/10
20/20 [==============================] - 32s 2s/step - loss: 0.9392 - accuracy: 0.6543 -
val_loss: 0.7804 - val_accuracy: 0.6875
Epoch 4/10
20/20 [==============================] - 32s 2s/step - loss: 0.8150 - accuracy: 0.6913 -
val_loss: 0.6536 - val_accuracy: 0.7500
Epoch 5/10
20/20 [==============================] - 32s 2s/step - loss: 0.6805 - accuracy: 0.7540 -
val_loss: 0.7574 - val_accuracy: 0.5938
Epoch 6/10
20/20 [==============================] - 33s 2s/step - loss: 0.6325 - accuracy: 0.7540 -
val_loss: 0.3115 - val_accuracy: 0.9062
Epoch 7/10
20/20 [==============================] - 34s 2s/step - loss: 0.5708 - accuracy: 0.7814 -
val_loss: 0.4432 - val_accuracy: 0.8125
Epoch 8/10
20/20 [==============================] - 35s 2s/step - loss: 0.5071 - accuracy: 0.8103 -
val_loss: 0.3246 - val_accuracy: 0.8750
Epoch 9/10
20/20 [==============================] - 34s 2s/step - loss: 0.5100 - accuracy: 0.7990 -
val_loss: 0.3943 - val_accuracy: 0.8125
Epoch 10/10
20/20 [==============================] - 34s 2s/step - loss: 0.4599 - accuracy: 0.8505 -
```

```
val_loss: 0.2202 - val_accuracy: 0.9062evaluation = model.evaluate(validation_generator,
steps=validation_generator.samples // batch_size)
print("Validation Accuracy: {:.2f}%".format(evaluation[1] * 100))
1/1 [==============================] - 3s 3s/step - loss: 0.1884 - accuracy: 0.9062
Validation Accuracy: 90.62%from tensorflow.keras.models import load_model
# Load the trained model
loaded_model = load_model('/content/fruit_recognition_model_trained.h5')
# Example usage for making predictions on a new image
new_image_path = '/content/drive/MyDrive/fruits/pomegranate/Image_3@.jpg'
processed_new_image = preprocess_image(new_image_path)
prediction = loaded_model.predict(processed_new_image)
predicted_class = np.argmax(prediction)
# Mapping class index to fruit name
class_names = {0: 'peas', 1: 'pineapple', 2: 'pomegranate', 3: 'potato', 4: 'tomato', 5:
'watermelon'}
predicted_fruit_name = class_names[predicted_class]
print("Predicted Class Index:", predicted_class)
print("Predicted Fruit Name:", predicted_fruit_name)
1/1 [==============================] - 0s 77ms/step
Predicted Class Index: 2
Predicted Fruit Name: pomegranate
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
import numpy as np
import matplotlib.pyplot as plt
# Load the trained model
loaded_model = load_model('/content/fruit_recognition_model_trained.h5')
# Mapping class index to fruit name
class_names = {0: 'peas', 1: 'pineapple', 2: 'pomegranate', 3: 'potato', 4: 'tomato', 5:
'watermelon'}
# Function to preprocess the input image
def preprocess_image(image_path):
img = image.load_img(image_path, target_size=(64, 64))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
return img_array / 255.0
# Function to make predictions
def predict_and_display_image(image_path):
processed_image = preprocess_image(image_path)
prediction = loaded_model.predict(processed_image)
predicted_class = np.argmax(prediction)
predicted_fruit_name = class_names[predicted_class]
```

```
# Display the image
img = image.load_img(image_path)
plt.imshow(img)

plt.title(f"Predicted Class: {predicted_class}, Predicted Fruit:
{predicted_fruit_name}")
plt.axis('off')
plt.show()
print("Predicted Class Index:", predicted_class)
print("Predicted Fruit Name:", predicted_fruit_name)
# Example usage
new_image_path = '/content/drive/MyDrive/fruits/pomegranate/Image_3@.jpg'
predict_and_display_image(new_image_path)
```

OUTPUT:

```
1/1 [==============================] - 0s 225ms/step
```




```
Predicted Class Index: 2
Predicted Fruit Name: pomegranate
```

RESULT:

Ex. No:3	**SPEECH RECOGNITION**

AIM:

PROCEDURE:

PROGRAM:

```
import speech_recognition as sr
# Initialize the recognizer
recognizer = sr.Recognizer()
# Specify the path to the audio file
audio_file = "/content/03-01-08-02-02-02-01.wav"
# Load the audio file
with sr.AudioFile(audio_file) as source:
audio = recognizer.record(source) # Read the entire audio file
# Recognize speech using Google Speech Recognition
try:
print("Transcription: " + recognizer.recognize_google(audio))
except sr.UnknownValueError:
print("Sorry, could not understand audio")
except sr.RequestError as e:
print("Could not request results from Google Speech Recognition service;
{0}".format(e)
```

OUTPUT:

```
Transcription: dogs are sitting by the door
```

RESULT:

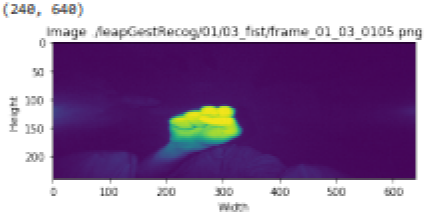
Ex. No:4	GESTURE RECOGNITION

AIM:

PROCEDURE:

PROGRAM:

```
%matplotlib inline
from google.colab import files
import os

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import cv2
import pandas as pd

# Sklearn
from sklearn.model_selection import train_test_split # Helps with organizing data for training
from sklearn.metrics import confusion_matrix # Helps present results as a confusion-matrix

print(tf.__version__)

    1.13.1


# Unzip images, ignore this cell if files are already in the workspace
!unzip leapGestRecog.zip


# We need to get all the paths for the images to later load them
imagepaths = []

# Go through all the files and subdirectories inside a folder and save path to images inside list
for root, dirs, files in os.walk(".", topdown=False):
  for name in files:
    path = os.path.join(root, name)
    if path.endswith("png"): # We want only the images
      imagepaths.append(path)

print(len(imagepaths)) # If > 0, then a PNG image was loaded

    20000


# This function is used more for debugging and showing results later. It plots the image into the notebook

def plot_image(path):
  img = cv2.imread(path) # Reads the image into a numpy.array
  img_cvt = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Converts into the corret colorspace (RGB)
  print(img_cvt.shape) # Prints the shape of the image just to check
  plt.grid(False) # Without grid so we can see better
  plt.imshow(img_cvt) # Shows the image
  plt.xlabel("Width")
  plt.ylabel("Height")
  plt.title("Image " + path)


plot_image(imagepaths[0]) #We plot the first image from our imagepaths array

    (240, 640)
```



```
X = [] # Image data
y = [] # Labels

# Loops through imagepaths to load images and labels into arrays
for path in imagepaths:
  img = cv2.imread(path) # Reads image and returns np.array
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Converts into the corret colorspace (GRAY)
  img = cv2.resize(img, (320, 120)) # Reduce image size so training can be faster
  X.append(img)

  # Processing label in image path
  category = path.split("/")[3]
  label = int(category.split("_")[0][1]) # We need to convert 10_down to 00_down, or else it crashes
  y.append(label)

# Turn X and y into np.array to speed up train_test_split
X = np.array(X, dtype="uint8")
X = X.reshape(len(imagepaths), 120, 320, 1) # Needed to reshape so CNN knows it's different images
y = np.array(y)

print("Images loaded: ", len(X))
print("Labels loaded: ", len(y))

print(y[0], imagepaths[0]) # Debugging
```

```
Images loaded:  20000
Labels loaded:  20000
3 ./leapGestRecog/01/03_fist/frame_01_03_0105.png
```

```
ts = 0.3 # Percentage of images that we want to use for testing. The rest is used for training.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=ts, random_state=42)
```

```
# Recreate the exact same model, including weights and optimizer.
# model = keras.models.load_model('handrecognition_model.h5')
# model.summary()

# To use the pre-trained model, just load it and skip to the next session.
```

```
# Import of keras model and hidden layers for our convolutional network
from keras.models import Sequential
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers import Dense, Flatten
```

```
Using TensorFlow backend.
```

```
# Construction of model
model = Sequential()
model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(120, 320, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from te
Instructions for updating:
Colocations handled automatically by placer.
```

```
# Configures the model for training
model.compile(optimizer='adam', # Optimization routine, which tells the computer how to adjust the parameter values to minimize the loss fun
              loss='sparse_categorical_crossentropy', # Loss function, which tells us how bad our predictions are.
              metrics=['accuracy']) # List of metrics to be evaluated by the model during training and testing.
```

```
# Trains the model for a given number of epochs (iterations on a dataset) and validates it.
model.fit(X_train, y_train, epochs=5, batch_size=64, verbose=2, validation_data=(X_test, y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.
Instructions for updating:
Use tf.cast instead.
Train on 14000 samples, validate on 6000 samples
```

```
Epoch 1/5
 - 305s - loss: 0.5644 - acc: 0.9133 - val loss: 0.0096 - val acc: 0.9968
Epoch 2/5
 - 302s - loss: 0.0169 - acc: 0.9961 - val_loss: 0.0145 - val_acc: 0.9958
Epoch 3/5
 - 301s - loss: 0.0045 - acc: 0.9987 - val_loss: 0.0015 - val_acc: 0.9992
Epoch 4/5
 - 301s - loss: 6.5339e-05 - acc: 1.0000 - val_loss: 4.1817e-04 - val_acc: 0.9998
Epoch 5/5
 - 305s - loss: 1.6688e-05 - acc: 1.0000 - val_loss: 3.7710e-04 - val_acc: 0.9998
<keras.callbacks.History at 0x7f8c5fb09c18>
```

```
# Save entire model to a HDF5 file
model.save('handrecognition_model.h5')
```

```
test_loss, test_acc = model.evaluate(X_test, y_test)

print('Test accuracy: {:2.2f}%'.format(test_acc*100))
```

```
6000/6000 [==============================] - 39s 6ms/step
Test accuracy: 99.98%
```

```
predictions = model.predict(X_test) # Make predictions towards the test set
```

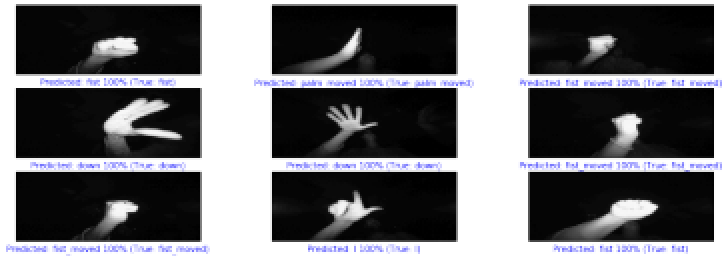
```
np.argmax(predictions[0]), y_test[0] # If same, got it right
```

```
(8, 8)
```

```
# Function to plot images and labels for validation purposes
def validate_9_images(predictions_array, true_label_array, img_array):
  # Array for pretty printing and then figure size
  class_names = ["down", "palm", "l", "fist", "fist_moved", "thumb", "index", "ok", "palm_moved", "c"]
  plt.figure(figsize=(15,5))

  for i in range(1, 10):
    # Just assigning variables
    prediction = predictions_array[i]
    true_label = true_label_array[i]
    img = img_array[i]
    img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)

    # Plot in a good way
    plt.subplot(3,3,i)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(prediction) # Get index of the predicted label from prediction

    # Change color of title based on good prediction or not
    if predicted_label == true_label:
      color = 'blue'
    else:
      color = 'red'

    plt.xlabel("Predicted: {} {:2.0f}% (True: {})".format(class_names[predicted_label],
                                  100*np.max(prediction),
                                  class_names[true_label]),
                                  color=color)
  plt.show()
```

```
validate_9_images(predictions, y_test, X_test)
```

OUTPUT:




```
y_pred = np.argmax(predictions, axis=1) # Transform predictions into 1-D array with label number

# H = Horizontal
# V = Vertical

pd.DataFrame(confusion_matrix(y_test, y_pred),
             columns=["Predicted Thumb Down", "Predicted Palm (H)", "Predicted L", "Predicted Fist (H)", "Predicted Fist (V)", "Predicted Th
             index=["Actual Thumb Down", "Actual Palm (H)", "Actual L", "Actual Fist (H)", "Actual Fist (V)", "Actual Thumbs up", "Actual In
```

	Predicted Thumb Down	Predicted Palm (H)	Predicted L	Predicted Fist (H)	Predicted Fist (V)	Predicted Thumbs up	Predicted Index	Pr
Actual Thumb Down	604	0	0	0	0	0	0	
Actual Palm (H)	0	617	0	1	0	0	0	
Actual L	0	0	621	0	0	0	0	
Actual Fist (H)	0	0	0	605	0	0	0	
Actual Fist (V)	0	0	0	0	596	0	0	

RESULT:

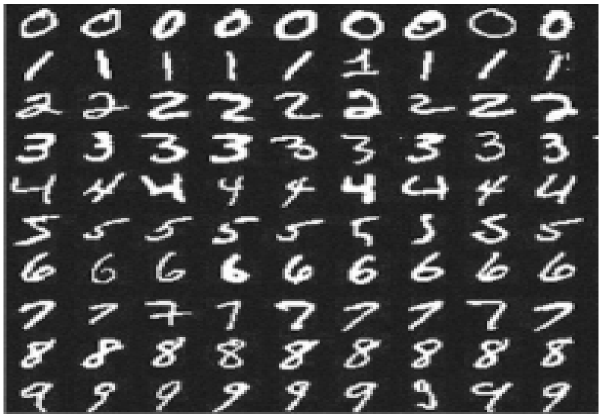
Ex. No: 5	PATTERN RECOGNITION

AIM:

PROCEDURE:

PROGRAM:

Pattern recognition (using Convolutional Neural Network)



Import dependencies

```
# Selecting Tensorflow version v2 (the command is relevant for Colab only).
%tensorflow_version 2.x
```

```
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sn
import numpy as np
import pandas as pd
import math
import datetime
import platform

print('Python version:', platform.python_version())
print('Tensorflow version:', tf.__version__)
print('Keras version:', tf.keras.__version__)
```

```
Python version: 3.7.6
Tensorflow version: 2.1.0
Keras version: 2.2.4-tf
```

```
# Load the TensorBoard notebook extension.
# %reload_ext tensorboard
%load_ext tensorboard
```

Start coding or generate with AI.

```
# Clear any logs from previous runs.
!rm -rf ./.logs/
```

Load the data

```
mnist_dataset = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist_dataset.load_data()
```

```
print('x_train:', x_train.shape)
print('y_train:', y_train.shape)
print('x_test:', x_test.shape)
print('y_test:', y_test.shape)
```

```
x_train: (60000, 28, 28)
y_train: (60000,)
x_test: (10000, 28, 28)
y_test: (10000,)
```

```
# Save image parameters to the constants that we will use later for data re-shaping and for model traning.
(_, IMAGE_WIDTH, IMAGE_HEIGHT) = x_train.shape
IMAGE_CHANNELS = 1

print('IMAGE_WIDTH:', IMAGE_WIDTH);
print('IMAGE_HEIGHT:', IMAGE_HEIGHT);
print('IMAGE_CHANNELS:', IMAGE_CHANNELS);
```

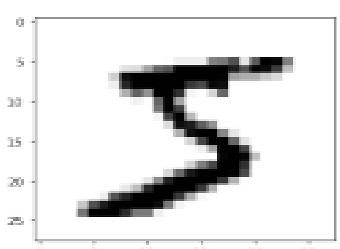
```
IMAGE_WIDTH: 28
IMAGE_HEIGHT: 28
IMAGE_CHANNELS: 1
```

```
pd.DataFrame(x_train[0])
```

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25
0	0	0	0	0	0	0	0	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	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	175	26	166	255	247	127	0	0
6	0	0	0	0	0	0	0	0	30	36	...	225	172	253	242	195	64	0	0
7	0	0	0	0	0	0	0	49	238	253	...	93	82	82	56	39	0	0	0
8	0	0	0	0	0	0	0	18	219	253	...	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	80	156	...	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	14	...	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	...	25	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	...	150	27	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	...	253	187	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	...	253	249	64	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	...	253	207	2	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	...	250	182	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	...	78	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	23	66	...	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	18	171	219	253	...	0	0	0	0	0	0	0	0
23	0	0	0	0	55	172	226	253	253	253	...	0	0	0	0	0	0	0	0
24	0	0	0	0	136	253	253	253	212	135	...	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

28 rows × 28 columns

```
plt.imshow(x_train[0], cmap=plt.cm.binary)
plt.show()
```



```
numbers_to_display = 25
num_cells = math.ceil(math.sqrt(numbers_to_display))
plt.figure(figsize=(10,10))
for i in range(numbers_to_display):
    plt.subplot(num_cells, num_cells, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(y_train[i])
plt.show()
```

```
x_train_with_chanels = x_train.reshape(
    x_train.shape[0],
    IMAGE_WIDTH,
    IMAGE_HEIGHT,
    IMAGE_CHANNELS
)

x_test_with_chanels = x_test.reshape(
    x_test.shape[0],
    IMAGE_WIDTH,
    IMAGE_HEIGHT,
    IMAGE_CHANNELS
)
```

```
print('x_train_with_chanels:', x_train_with_chanels.shape)
print('x_test_with_chanels:', x_test_with_chanels.shape)
```

```
x_train_with_chanels: (60000, 28, 28, 1)
x_test_with_chanels: (10000, 28, 28, 1)
```

```
x_train_normalized = x_train_with_chanels / 255
x_test_normalized = x_test_with_chanels / 255
```

```
# Let's check just one row from the 0th image to see color chanel values after normalization.
x_train_normalized[0][18]
```

```
array([[0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
```

RESULT:

Ex. No:6	**FEATURE COMPARISON MODEL**

AIM:

PROCEDURE:

PROGRAM:

```
# -*- coding: utf-8 -*-
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
train_data = pd.read_csv(r'C:\Users\Database\Downloads\creditcardfrauddetection\fraudTrain.csv')
test_data = pd.read_csv(r'C:\Users\Database\Downloads\creditcardfrauddetection\fraudTest.csv')
train_data.drop(columns=['unix_time'], inplace=True)
test_data.drop(columns=['unix_time'], inplace=True)
X_train = train_data.drop(columns=['category'])
y_train = train_data['category']
X_test = test_data.drop(columns=['category'])
y_test = test_data['category']
numerical_columns = X_train.columns
numerical_columns = X_train.select_dtypes(include=['int64', 'float64']).columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_columns)])
lr_classifier = Pipeline(steps=[('preprocessor', preprocessor),
                                ('classifier', LogisticRegression())])
lr_classifier.fit(X_train, y_train)
lr_predictions = lr_classifier.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)

print("Logistic Regression:")
print("Accuracy:", lr_accuracy)
print("Confusion Matrix:")
print(confusion_matrix(y_test, lr_predictions))
print("Classification Report:")
print(classification_report(y_test, lr_predictions))
dt_classifier = Pipeline(steps=[('preprocessor', preprocessor),
                                ('classifier', DecisionTreeClassifier())])
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("\nDecision Trees:")
print("Accuracy:", dt_accuracy)
print("Confusion Matrix:")
print(confusion_matrix(y_test, dt_predictions))
print("Classification Report:")
print(classification_report(y_test, dt_predictions))
```

OUTPUT:

```
Logistic Regression:
Accuracy: 0.14218876806443545
Confusion Matrix:
[[    0     0 27133     0  6256     0  5507     0     0     0     0    54
   1154     0]
 [    0     0 27466     0  4368     0  6003     0     0     0     0    54
   1377     0]
 [    0     0 48812     0  1790     0  2975     0     0     0     0   154
   2639     0]
 [    0     0 14727     0   827     0  2680     0     0     0     0    41
   1151     0]
 [    0     0 25849     0 22317     0  1603     0     0     0     0   485
   2299     0]
 [    0     0 26435     0  3971     0  5111     0     0     0     0    52
   1105     0]
 [    0     0 37111     0  6359     0  7077     0     0     0     0    67
   1731     0]
 [    0     0 34063     0  6568     0  6152     0     0     0     0    65
   1844     0]
 [    0     0 16319     0  5240     0  5308     0     0     0     0     7
    493     0]
 [    0     0 21282     0  5268     0  7229     0     0     0     0    55
    740     0]
 [    0     0 27555     0  3984     0  6659     0     0     0     0    70
   1059     0]
 [    0     0 25892     0  6284     0  8879     0     0     0     0     0
    724     0]
 [    0     0 27064     0  9850     0 12064     0     0     0     0     2
    811     0]
 [    0     0 10137     0  2342     0  4752     0     0     0     0    40
    178     0]]
```

```
Decision Trees:
Accuracy: 0.1944435946944409
Confusion Matrix:
[[ 4194  2999  3847  1271  2618  3529  4759  3590  1897  2133  3470  2378
   2640   779]
 [ 2977  5270  3362  1846  2193  2745  3985  4676  1577  1928  3127  2338
   2375   869]
 [ 4168  3400 19105  3075  5026  3767  5521  5219  1011  1023  3508   797
    710    40]
 [ 1259  1868  2862  2354  1443  1396  2027  2806   391   730  1389   487
    370    44]
 [ 2816  2297  4865  1550 22357  2341  4180  3661  1411  1018  2129  2630
   1259    39]
 [ 3375  2723  3568  1424  2204  4362  4461  3515  1418  1632  3451  1874
   2033   634]
 [ 4675  3964  5208  2047  4032  4536  7247  5341  1973  2266  4727  3033
   2587   709]
 [ 3495  4665  5016  2962  3506  3569  5145  7252  1742  2322  3924  2175
   2288   631]
 [ 1904  1658   958   372  1264  1405  1961  1808  2500  2430  1784  3502
   4325  1496]
 [ 2151  2010  1005   779   984  1657  2110  2327  2404  6632  2136  3718
   4936  1725]
 [ 3446  3200  3475  1331  2083  3377  4548  3959  1779  2100  3738  2490
   2819   982]
 [ 2359  2392   824   470  2483  1917  3007  2156  3538  3645  2532  6928
   6965  2563]
 [ 2715  2568   668   390  1217  2137  2618  2342  4329  4938  2871  6850
  12741  3407]
 [  825   872    64    39    31   672   717   611  1480  1762   981  2570
   3449  3376]]
Classification Report:
```

RESULT:

Ex. No:7	**LAB RECOGNITION**

AIM:

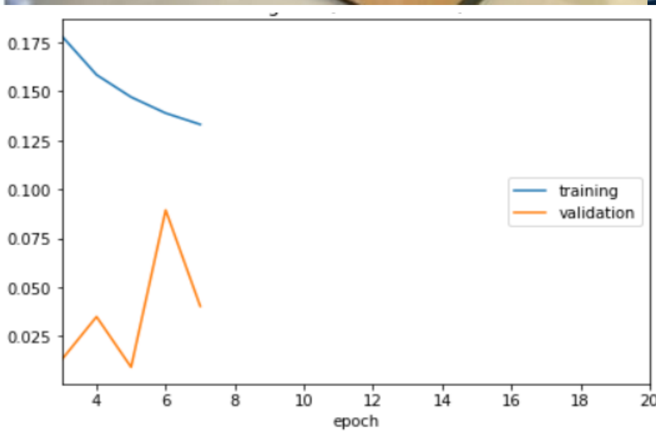
PROCEDURE:

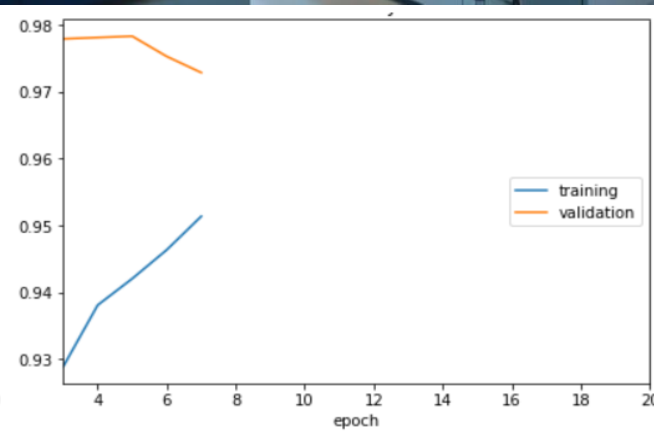
PROGRAM:

```
# google colab does not come with torch installed. And also, in course we are using torch 0.4.
# so following snippet of code installs the relevant version

from os.path import exists
from wheel.pep425tags import get_abbr_impl, get_impl_ver, get_abi_tag
platform = '{}{}-{}'.format(get_abbr_impl(), get_impl_ver(), get_abi_tag())
cuda_output = !ldconfig -p|grep cudart.so|sed -e 's/.*\.\([0-9]*\)\.\([0-9]*\)$/cu\1\2/'
accelerator = cuda_output[0] if exists('/dev/nvidia0') else 'cpu'

!pip install -q http://download.pytorch.org/whl/{accelerator}/torch-0.4.1-{platform}-linux_x86_64.whl torchvision
import torch


# we will verify that GPU is enabled for this notebook
# following should print: CUDA is available!  Training on GPU ...
#
# if it prints otherwise, then you need to enable GPU:
# from Menu > Runtime > Change Runtime Type > Hardware Accelerator > GPU

import torch
import numpy as np

# check if CUDA is available
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')


# we need pillow version of 5.3.0
# we will uninstall the older version first
!pip uninstall -y Pillow
# install the new one
!pip install Pillow==5.3.0
# import the new one
import PIL
print(PIL.PILLOW_VERSION)
# this should print 5.3.0. If it doesn't, then restart your runtime:
# Menu > Runtime > Restart Runtime


# Imports here


# we will download the required data files
!wget -cq https://github.com/udacity/pytorch_challenge/raw/master/cat_to_name.json
!wget -cq https://s3.amazonaws.com/content.udacity-data.com/courses/nd188/flower_data.zip
!rm -r flower_data || true
!unzip -qq flower_data.zip


data_dir = '/flower_data'
train_dir = data_dir + '/train'
valid_dir = data_dir + '/valid'


# TODO: Define your transforms for the training and validation sets
data_transforms =

# TODO: Load the datasets with ImageFolder
image_datasets =

# TODO: Using the image datasets and the trainforms, define the dataloaders
dataloaders =


import json

with open('cat_to_name.json', 'r') as f:
    cat_to_name = json.load(f)


# TODO: Build and train your network


# TODO: Save the checkpoint
```

```python
# TODO: Write a function that loads a checkpoint and rebuilds the model


def process_image(image):
    ''' Scales, crops, and normalizes a PIL image for a PyTorch model,
        returns an Numpy array
    '''

    # TODO: Process a PIL image for use in a PyTorch model


def imshow(image, ax=None, title=None):
    """Imshow for Tensor."""
    if ax is None:
        fig, ax = plt.subplots()

    # PyTorch tensors assume the color channel is the first dimension
    # but matplotlib assumes is the third dimension
    image = image.numpy().transpose((1, 2, 0))

    # Undo preprocessing
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    image = std * image + mean

    # Image needs to be clipped between 0 and 1 or it looks like noise when displayed
    image = np.clip(image, 0, 1)

    ax.imshow(image)

    return ax


def predict(image_path, model, topk=5):
    ''' Predict the class (or classes) of an image using a trained deep learning model.
    '''

    # TODO: Implement the code to predict the class from an image file
```

OUTPUT:






```
Log-loss (cost function):
training   (min:    0.133, max:    0.258, cur:    0.133)
validation (min:    0.009, max:    0.134, cur:    0.040)

accuracy:
training   (min:    0.891, max:    0.951, cur:    0.951)
validation (min:    0.966, max:    0.978, cur:    0.973)
Epoch 8/20
624/625 [============================>.] - ETA: 0s - loss: 0.1323 - accuracy: 0.9514
```

RESULT:

Ex. No:8	NETWORK MODELS

AIM:

PROCEDURE:

PROGRAM:

7]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img

dataset_path = r'C:\Users\User\Downloads\flowers'

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',
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 3457 images belonging to 5 classes.
Found 860 images belonging to 5 classes.
```

8]:
```
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

def build_model(base_model, num_classes):
    for layer in base_model.layers:
        layer.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    return Model(inputs=base_model.input, outputs=predictions)

base_model_vgg = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224,
model_vgg = build_model(base_model_vgg, num_classes=5)

base_model_resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(224,
model_resnet = build_model(base_model_resnet, num_classes=5)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/r
esnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 59s 1us/step
```

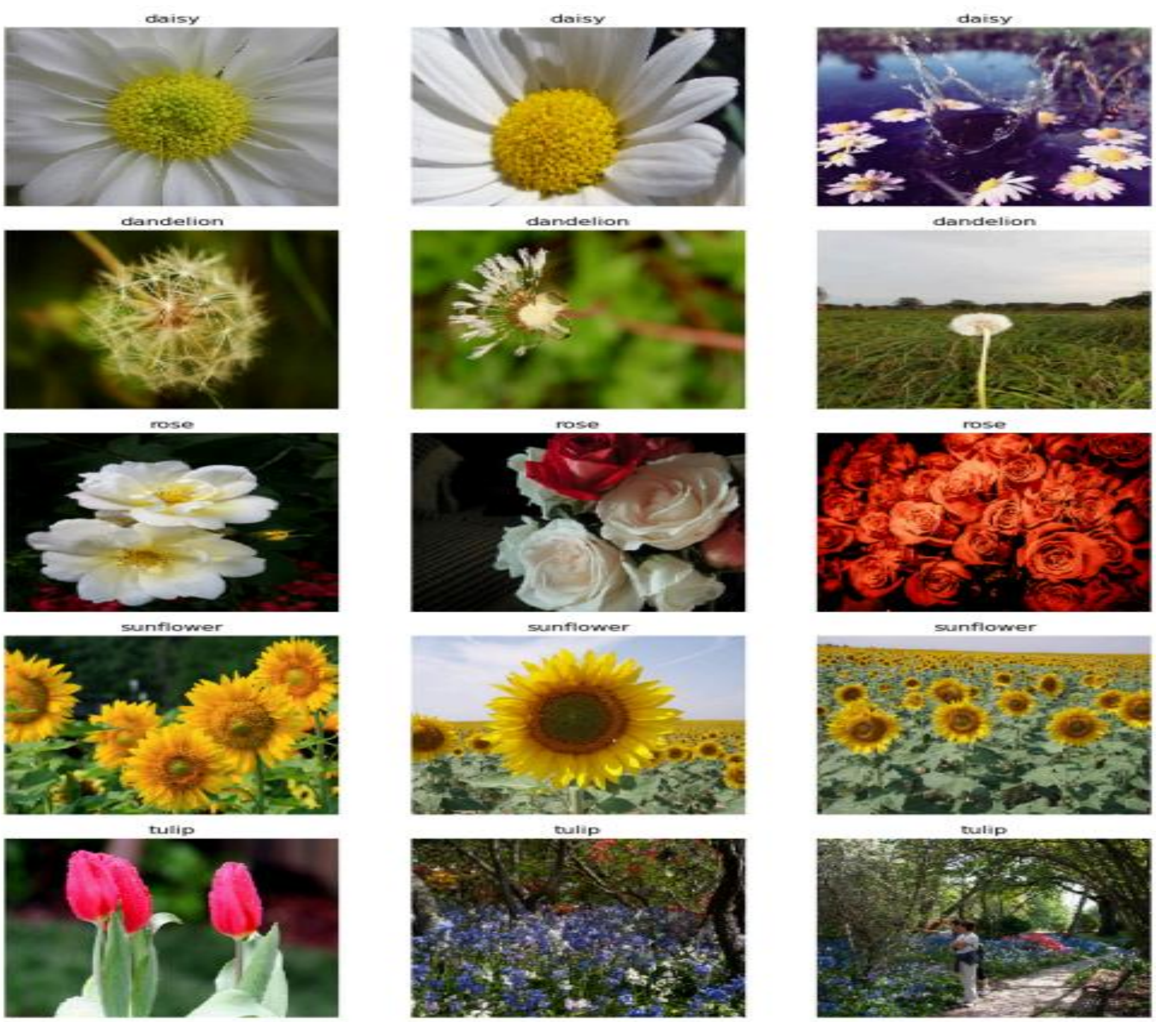
9]:
```
model_vgg.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
model_vgg.fit(train_generator, epochs=10, validation_data=validation_generator)
```

```
model_resnet.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc
model_resnet.fit(train_generator, epochs=10, validation_data=validation_generator)
```

```
Epoch 1/10
109/109 [==============================] - 1418s 13s/step - loss: 1.0499 - accuracy:
0.5846 - val_loss: 0.8108 - val_accuracy: 0.6860
Epoch 2/10
109/109 [==============================] - 1335s 12s/step - loss: 0.7848 - accuracy:
0.7185 - val_loss: 0.7025 - val_accuracy: 0.7488
Epoch 3/10
109/109 [==============================] - 1336s 12s/step - loss: 0.6896 - accuracy:
0.7550 - val_loss: 0.6240 - val_accuracy: 0.7802
Epoch 4/10
109/109 [==============================] - 1334s 12s/step - loss: 0.6151 - accuracy:
0.7738 - val_loss: 0.5811 - val_accuracy: 0.7837
Epoch 5/10
109/109 [==============================] - 1334s 12s/step - loss: 0.5893 - accuracy:
0.7790 - val_loss: 0.6428 - val_accuracy: 0.7709
Epoch 6/10
109/109 [==============================] - 1336s 12s/step - loss: 0.5414 - accuracy:
0.7984 - val_loss: 0.5314 - val_accuracy: 0.8070
Epoch 7/10
109/109 [==============================] - 1334s 12s/step - loss: 0.5788 - accuracy:
0.7917 - val_loss: 0.5950 - val_accuracy: 0.7884
Epoch 8/10
109/109 [==============================] - 1351s 12s/step - loss: 0.5231 - accuracy:
0.8097 - val_loss: 0.5492 - val_accuracy: 0.7977
Epoch 9/10
109/109 [==============================] - 1552s 14s/step - loss: 0.5177 - accuracy:
0.8068 - val_loss: 0.5469 - val_accuracy: 0.8047
Epoch 10/10
109/109 [==============================] - 1385s 13s/step - loss: 0.4804 - accuracy:
0.8224 - val_loss: 0.5526 - val_accuracy: 0.7907
Epoch 1/10
109/109 [==============================] - 547s 5s/step - loss: 1.6966 - accuracy:
0.2942 - val_loss: 1.7318 - val_accuracy: 0.2558
Epoch 2/10
109/109 [==============================] - 547s 5s/step - loss: 1.5608 - accuracy:
0.3248 - val_loss: 1.5242 - val_accuracy: 0.3523
Epoch 3/10
109/109 [==============================] - 716s 7s/step - loss: 1.5032 - accuracy:
0.3538 - val_loss: 1.5270 - val_accuracy: 0.2651
Epoch 4/10
109/109 [==============================] - 960s 9s/step - loss: 1.4876 - accuracy:
0.3749 - val_loss: 1.4600 - val_accuracy: 0.4081
Epoch 5/10
109/109 [==============================] - 566s 5s/step - loss: 1.4710 - accuracy:
0.3685 - val_loss: 1.4787 - val_accuracy: 0.3744
Epoch 6/10
109/109 [==============================] - 553s 5s/step - loss: 1.4840 - accuracy:
0.3627 - val_loss: 1.4587 - val_accuracy: 0.3965
Epoch 7/10
109/109 [==============================] - 630s 6s/step - loss: 1.4582 - accuracy:
0.3792 - val_loss: 1.4410 - val_accuracy: 0.3907
Epoch 8/10
109/109 [==============================] - 660s 6s/step - loss: 1.4471 - accuracy:
0.3763 - val_loss: 1.6047 - val_accuracy: 0.2686
Epoch 9/10
109/109 [==============================] - 599s 5s/step - loss: 1.4582 - accuracy:
0.3827 - val_loss: 1.4147 - val_accuracy: 0.4326
Epoch 10/10
109/109 [==============================] - 546s 5s/step - loss: 1.4357 - accuracy:
0.3934 - val_loss: 1.3888 - val_accuracy: 0.4244
```

Out[19]: <keras.src.callbacks.History at 0x1dbcaa5eb50>

In [25]:
```
def display_dataset_samples(dataset_path, num_images=3):
    flower_classes = os.listdir(dataset_path)
    plt.figure(figsize=(10, num_images * len(flower_classes)))
    for i, flower_class in enumerate(flower_classes):
        flower_class_path = os.path.join(dataset_path, flower_class)
        images = os.listdir(flower_class_path)[:num_images]
        for j, image in enumerate(images):
            img_path = os.path.join(flower_class_path, image)
            img = load_img(img_path, target_size=(224, 224))
            plt.subplot(len(flower_classes), num_images, i * num_images + j + 1)
            plt.imshow(img)
            plt.title(flower_class)
            plt.axis('off')
    plt.tight_layout()
    plt.show()

display_dataset_samples(dataset_path)
```



n [30]:
```
def visualize_model_prediction(model, img_path, class_indices):
    img = load_img(img_path, target_size=(224, 224))
    img_array = np.expand_dims(np.array(img) / 255.0, axis=0)
```

```
    predictions = model.predict(img_array)
    predicted_class = np.argmax(predictions, axis=1)
    plt.imshow(img)
    plt.title(f'Predicted: {list(class_indices.keys())[predicted_class[0]]}, Probabil
    plt.axis('off')
    plt.show()

# Example usage (replace 'path/to/your/test/image.jpg' with a real image path)
visualize_model_prediction(model_vgg, r'C:\Users\User\Downloads\flowers\rose\2952573
```

```
1/1 [==============================] - 1s 1s/step
```

Predicted: rose, Probability: 0.9993624091148376

```
1]: visualize_model_prediction(model_resnet, r'C:\Users\User\Downloads\flowers\rose\2952
```

```
1/1 [==============================] - 2s 2s/step
```

Predicted: tulip, Probability: 0.36886510252952576

```
import numpy as np
from sklearn.metrics import classification_report
from math import ceil

def evaluate_model(model, generator):
    steps = ceil(generator.samples / generator.batch_size)
    predictions = model.predict(generator, steps=steps)
    predicted_classes = np.argmax(predictions, axis=1)

    # Handling cases where the number of predictions exceeds the number of true label
    true_classes = generator.classes[:len(predicted_classes)]

    class_labels = list(generator.class_indices.keys())
    report = classification_report(true_classes, predicted_classes, target_names=cla
    print(report)

# Evaluate both models
evaluate_model(model_vgg, validation_generator)
evaluate_model(model_resnet, validation_generator)
```

OUTPUT:

```
27/27 [==============================] - 266s 10s/step
              precision    recall  f1-score   support

       daisy       0.18      0.12      0.15       152
   dandelion       0.20      0.20      0.20       210
        rose       0.20      0.21      0.21       156
   sunflower       0.21      0.25      0.23       146
       tulip       0.26      0.27      0.26       196

    accuracy                           0.21       860
   macro avg       0.21      0.21      0.21       860
weighted avg       0.21      0.21      0.21       860

27/27 [==============================] - 108s 4s/step
              precision    recall  f1-score   support

       daisy       0.20      0.20      0.20       152
   dandelion       0.27      0.39      0.32       210
        rose       0.00      0.00      0.00       156
   sunflower       0.18      0.12      0.14       146
       tulip       0.23      0.37      0.29       196

    accuracy                           0.23       860
   macro avg       0.18      0.21      0.19       860
weighted avg       0.18      0.23      0.20       860
```

RESULT: