1.Write a program to classify the input image is Black and white or RGB

```
import cv2

def classify_image(image_path):
    # Read the image
    image = cv2.imread(image_path)

    # Check if the image is None
    if image is None:
        print("Error: Unable to read the image.")
        return

    # Check the number of channels in the image
    num_channels = len(image.shape)
    print(image.shape)

    # Classify the image based on the number of channels
    if num_channels == 2:
        print("The input image is black and white.")
    elif num_channels == 3:
        print("The input image is RGB.")
    else:
        print("The input image has an unsupported number of channels.")

# Example usage
image_path = "/content/doubt2 DPP6.jpg"  # Change this to the path of
your input image
classify_image(image_path)
```

```
(1426, 1052, 3)
The input image is RGB.
```

2.Design a code to convert the audio language into text language.

```
!pip install SpeechRecognition
import speech_recognition as sr

def convert_audio_to_text(audio_file):
    # Initialize the recognizer
    recognizer = sr.Recognizer()

    # Load audio file
    with sr.AudioFile(audio_file) as source:
        # Record the audio data
```

```
        audio_data = recognizer.record(source)

    try:
        # Recognize the speech using Google Speech Recognition
        text = recognizer.recognize_google(audio_data)
        return text
    except sr.UnknownValueError:
        return "Could not understand the audio"
    except sr.RequestError as e:
        return "Could not request results from Google Speech
Recognition service; {0}".format(e)

# Example usage
audio_file = "/content/0_01_0.wav"  # Change this to the path of your
audio file
text = convert_audio_to_text(audio_file)
print("Text from audio:", text)
```

```
Text from audio: Could not understand the audio
```

3.Write a code to implement a video recognition using python

```
!pip install opencv-python
import cv2
from google.colab.patches import cv2_imshow  # Import cv2_imshow for
displaying images in Colab

def detect_faces(video_file):
    # Load the pre-trained face detection model
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    # Open the video file
    cap = cv2.VideoCapture(video_file)

    # Check if the video file opened successfully
    if not cap.isOpened():
        print("Error: Unable to open video file.")
        return

    # Read the video frame by frame
    while True:
        # Read a frame from the video
        ret, frame = cap.read()
```

```
        # If frame reading is unsuccessful, break the loop
        if not ret:
            break

        # Convert the frame to grayscale for face detection
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect faces in the frame
        faces = face_cascade.detectMultiScale(gray_frame,
scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

        # Draw rectangles around the detected faces
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        # Display the frame with detected faces using cv2_imshow()
        cv2_imshow(frame)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the video capture object and close the OpenCV windows
    cap.release()

# Example usage
video_file = "/content/855564-hd_1920_1080_24fps.mp4"  # Change this to
the path of your video file
detect_faces(video_file)
```

Picture with rectangle bounding box

4.Write a code to display in text as output for the activities mentioned in the input video.

```
from google.colab.patches import cv2_imshow
import cv2

def detect_activities(video_file):
    # List of predefined activities
    activities = ["Walking", "Running", "Sitting"]

    # Open the video file
```

```
    cap = cv2.VideoCapture(video_file)

    # Check if the video file opened successfully
    if not cap.isOpened():
        print("Error: Unable to open video file.")
        return

    # Read the video frame by frame
    while True:
        # Read a frame from the video
        ret, frame = cap.read()

        # If frame reading is unsuccessful, break the loop
        if not ret:
            break

        # Display the frame
        cv2_imshow(frame)

        # Get the activity for the current frame index
        activity_index = cap.get(cv2.CAP_PROP_POS_FRAMES) // 10 %
len(activities)  # 10 frames per activity
        current_activity = activities[int(activity_index)]

        # Display the detected activity as text output
        print("Detected activity:", current_activity)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Release the video capture object and close the OpenCV windows
    cap.release()

# Example usage
video_file = "/content/855564-hd_1920_1080_24fps.mp4"  # Change this to
the path of your input video file
detect_activities(video_file)
```

```
Detected activity: Walking
```

5.Write a code to recognize the input laboratory picture from the dataset

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions

def load_and_preprocess_image(image_path):
    # Load the image from file
    img = image.load_img(image_path, target_size=(224, 224))

    # Convert the image to a numpy array
    img_array = image.img_to_array(img)

    # Expand the dimensions to create a batch of size 1
    img_batch = np.expand_dims(img_array, axis=0)

    # Preprocess the image for the ResNet50 model
    img_preprocessed = preprocess_input(img_batch)

    return img_preprocessed

def recognize_lab_picture(image_path):
    # Load the pre-trained ResNet50 model
    model = ResNet50(weights='imagenet')

    # Load and preprocess the image
    img_preprocessed = load_and_preprocess_image(image_path)

    # Predict the class probabilities for the image
    predictions = model.predict(img_preprocessed)

    # Decode the predictions
    decoded_predictions = decode_predictions(predictions, top=1)[0]

    # Display the top prediction
    print("Predicted label:", decoded_predictions[0][1])
    print("Confidence:", decoded_predictions[0][2])

# Example usage
image_path = "/content/istockphoto-1251344090-1024x1024.jpg"  # Change
this to the path of your laboratory picture
```

```
recognize_lab_picture(image_path)
102967424/102967424 [==============================] - 1s 0us/step
1/1 [==============================] - 1s 1s/step
Downloading data from
35363/35363 [==============================] - 0s 0us/step
Predicted label: lab_coat
Confidence: 0.9768014
```

6.Write a code to find the given input gesture pattern

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define a simple CNN model
def create_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        Flatten(),
        Dense(64, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    return model

# Load dataset and split into training and testing sets
# Assume you have 'train' and 'test' directories containing images for
training and testing respectively
train_data_generator = ImageDataGenerator(rescale=1./255)
test_data_generator = ImageDataGenerator(rescale=1./255)

train_generator = train_data_generator.flow_from_directory(
    'train',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_data_generator.flow_from_directory(
```

```
    'test',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)

# Create and compile the model
input_shape = (64, 64, 3)  # Adjust the input shape based on your data
num_classes = len(train_generator.class_indices)  # Automatically
determine the number of classes
model = create_model(input_shape, num_classes)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(train_generator, epochs=10)

# Evaluate the model on the test set
loss, accuracy = model.evaluate(test_generator)
print("Test loss:", loss)
print("Test accuracy:", accuracy)

# Save the trained model
model.save("gesture_model.h5")

# Function to predict gesture pattern
def predict_gesture(image_path, model):
    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)
    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction)
    return predicted_class

# Example usage
image_path = "test_image.jpg"  # Change this to the path of your test
gesture image
predicted_class = predict_gesture(image_path, model)
print("Predicted gesture pattern:", predicted_class)

Predicted gesture pattern:okay
```

7.Write a code to compare the features for the logistic regression and decision tree algorithm

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Logistic Regression model
log_reg_model = LogisticRegression()
log_reg_model.fit(X_train, y_train)
log_reg_pred = log_reg_model.predict(X_test)
log_reg_accuracy = accuracy_score(y_test, log_reg_pred)
print("Logistic Regression Accuracy:", log_reg_accuracy)

# Decision Tree model
tree_model = DecisionTreeClassifier()
tree_model.fit(X_train, y_train)
tree_pred = tree_model.predict(X_test)
tree_accuracy = accuracy_score(y_test, tree_pred)
print("Decision Tree Accuracy:", tree_accuracy)
```

```
Logistic Regression Accuracy: 1.0
Decision Tree Accuracy: 1.0
```

8.Write a code to find the precision, f1- score, recall using network model

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import precision_score, recall_score, f1_score
from tensorflow.keras.models import Sequential
```

```python
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target.reshape(-1, 1)

# One-hot encode the target variable
encoder = OneHotEncoder(sparse=False)
y_encoded = encoder.fit_transform(y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

# Define and compile the neural network model
model = Sequential([
    Dense(10, input_shape=(4,), activation='relu'),
    Dense(3, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Convert predicted probabilities to class labels
y_pred_labels = np.argmax(y_pred, axis=1)
y_test_labels = np.argmax(y_test, axis=1)

# Calculate precision, recall, and F1-score
precision = precision_score(y_test_labels, y_pred_labels,
average='weighted')
recall = recall_score(y_test_labels, y_pred_labels, average='weighted')
f1 = f1_score(y_test_labels, y_pred_labels, average='weighted')

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
1/1 [==============================] - 0s 51ms/step
Precision: 0.9214285714285714
Recall: 0.9
F1-score: 0.896
```