COMPUTER VISION - LAB

(COURSE CODE: 23UPCSC4L04)

A Laboratory record Submitted to Periyar University, Salem.
In partial fulfillment of the Requirements for the
Degree of

MASTER OF SCIENCE IN DATA SCIENCE

BY

GOWTHAM S
REG NO: U23PG507DTS010

DEPARTMENT OF COMPUTER SCIENCE

PERIYAR UNIVERSITY

PERIYAR PALKALAI NAGAR,
SALEM – 636011

CERTIFICATE

This is to certify that the Programming Laboratory entitled **"COMPUTER VISION- LAB"** (Course code: 23UPCSC4L04) is a bonafide record work done by **Mr. GOWTHAM.S** Register No. **U23PG507DTS010** as partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN DATA SCIENCE** in the Department of Computer Science, Periyar University, Salem, during the year **2023 – 2025.**

Faculty In-Charge　　　　　　　　　　　　　　　　　　Head of the Department

Submitted for the practical examination held on ……. /……. /………………

Marks Obtained

Internal Examiner　　　　　　　　　　　　　　　　　　External Examiner

INDEX

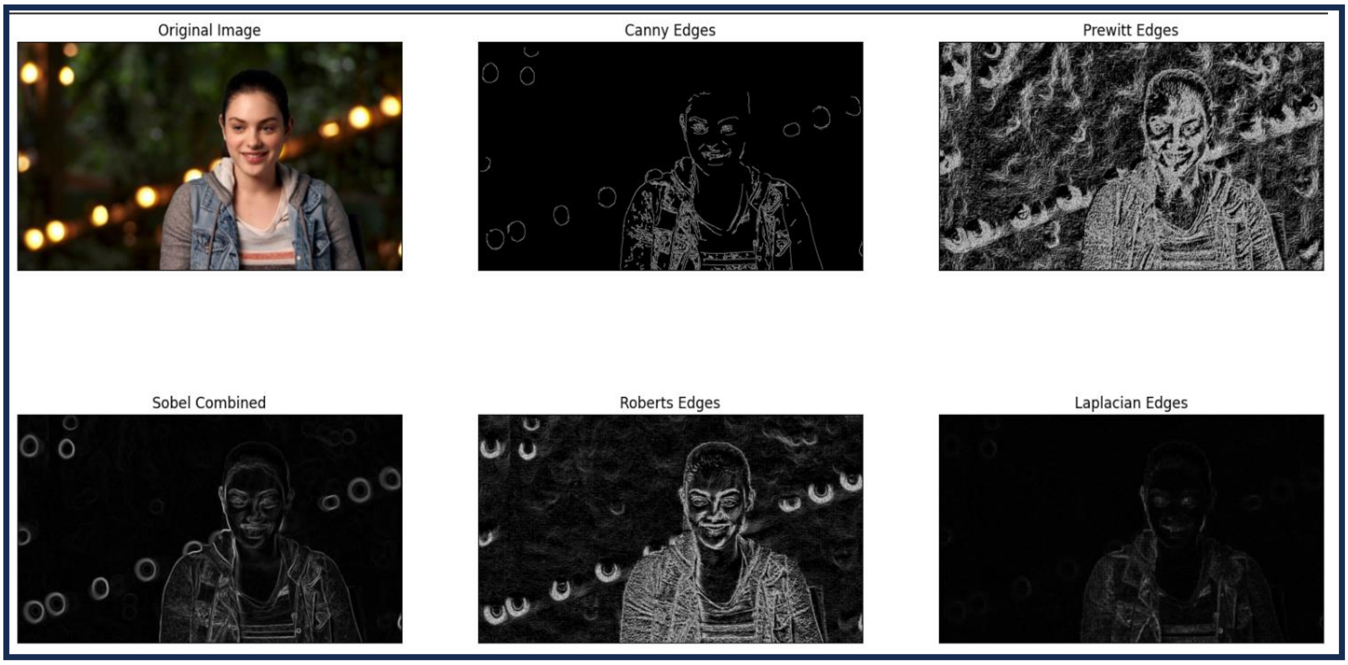
MORPHOLOGICAL OPERATIONS

INPUT:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
# Load the image in grayscale
image = cv2.imread('/content/drive/MyDrive/Application photo.jpg',cv2.IMREAD_GRAYSCALE)
# Check if the image was loaded successfully
if image is None:
    print("Error: Could not load image.")
    exit()
# Apply a binary threshold to the image
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
# Define a kernel for the morphological operations
kernel = np.ones((5, 5), np.uint8)
# Apply morphological operations
dilated = cv2.dilate(binary_image, kernel, iterations=1)
eroded = cv2.erode(binary_image, kernel, iterations=1)
opened = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)
closed = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE, kernel)
# Plot the results
plt.figure(figsize=(12, 10))
plt.subplot(2, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.subplot(2, 3, 2)
plt.title('Dilated Image')
plt.imshow(dilated, cmap='gray')
plt.axis('off')
plt.subplot(2, 3, 3)
plt.title('Eroded Image')
```

```
plt.imshow(eroded, cmap='gray')

plt.axis('off')
plt.subplot(2, 3, 4)

plt.title('Opened Image')

plt.imshow(opened, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title('Closed Image')
plt.imshow(closed, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

OUTPUT:



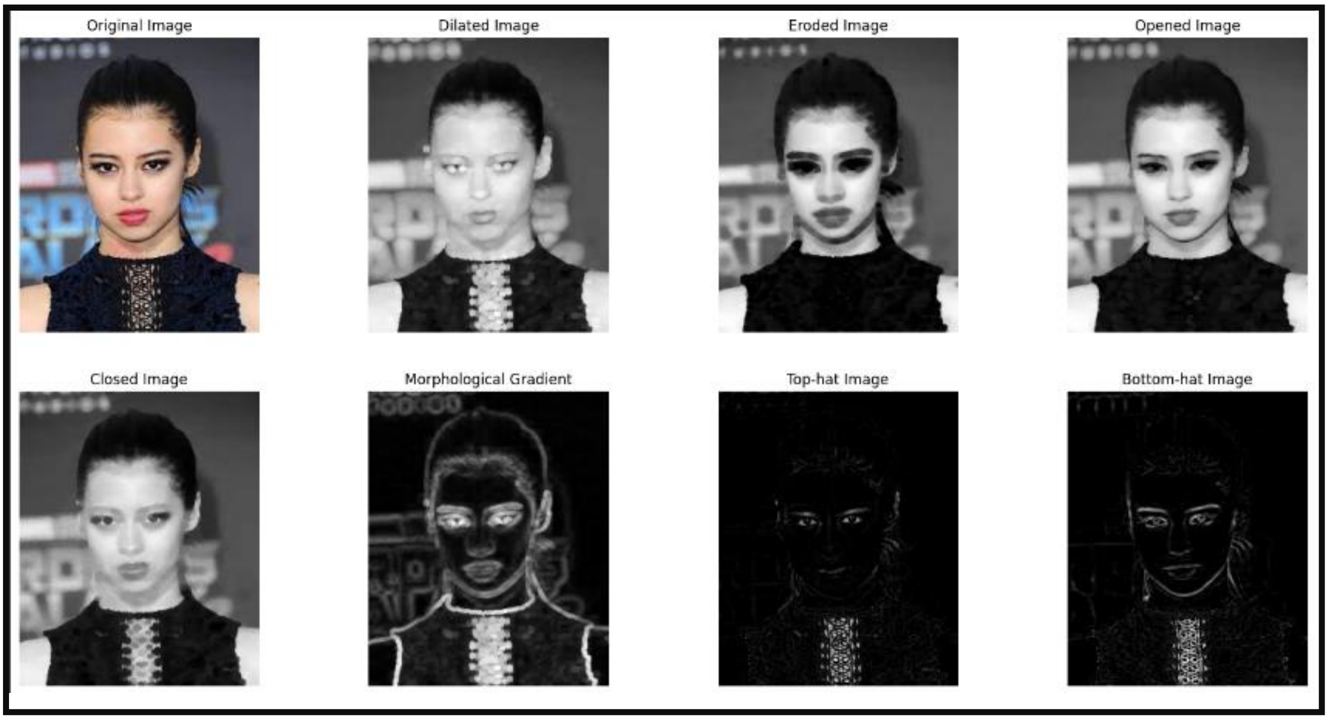
EDGE DETECTION

INPUT:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color, filters
# Load the image in grayscale
image = io.imread('   /content/mine.jpg')
gray_image = color.rgb2gray(image)
# Sobel Edge Detection
sobel_edges = filters.sobel(gray_image)
# Prewitt Edge Detection
prewitt_edges = filters.prewitt(gray_image)
# Roberts Edge Detection
roberts_edges = filters.roberts(gray_image)
# Laplacian of Gaussian (LoG) Edge Detection
log_edges = filters.laplace(gray_image)
# Zero-cross Edge Detection (using Laplacian of Gaussian approximation)
zero_cross_edges = filters.sobel(gray_image)
# Canny Edge Detection
canny_edges = cv2.Canny((gray_image * 255).astype(np.uint8), 100, 200)
# Display the results
plt.figure(figsize=(12, 6))
plt.subplot(2, 4, 1)
plt.imshow(gray_image, cmap='gray')
plt.title("Gray Scale Image")
plt.axis('off')
plt.subplot(2, 4, 2)
plt.imshow(sobel_edges, cmap='gray')
plt.title("Sobel")
plt.axis('off')
plt.subplot(2, 4, 3)
plt.imshow(prewitt_edges, cmap='gray')
```

```
plt.title("Prewitt")
plt.axis('off')
plt.subplot(2, 4, 4)
plt.imshow(roberts_edges, cmap='gray')
plt.title("Roberts")
plt.axis('off')
plt.subplot(2, 4, 5)
plt.imshow(log_edges, cmap='gray')
plt.title("LoG")
plt.axis('off')
plt.subplot(2, 4, 6)
plt.imshow(zero_cross_edges, cmap='gray')
plt.title("Zerocross")
plt.axis('off')
plt.subplot(2, 4, 7)
plt.imshow(canny_edges, cmap='gray')
plt.title("Canny")
plt.axis('off')
plt.tight_layout()
plt.show()
```

OUTPUT:



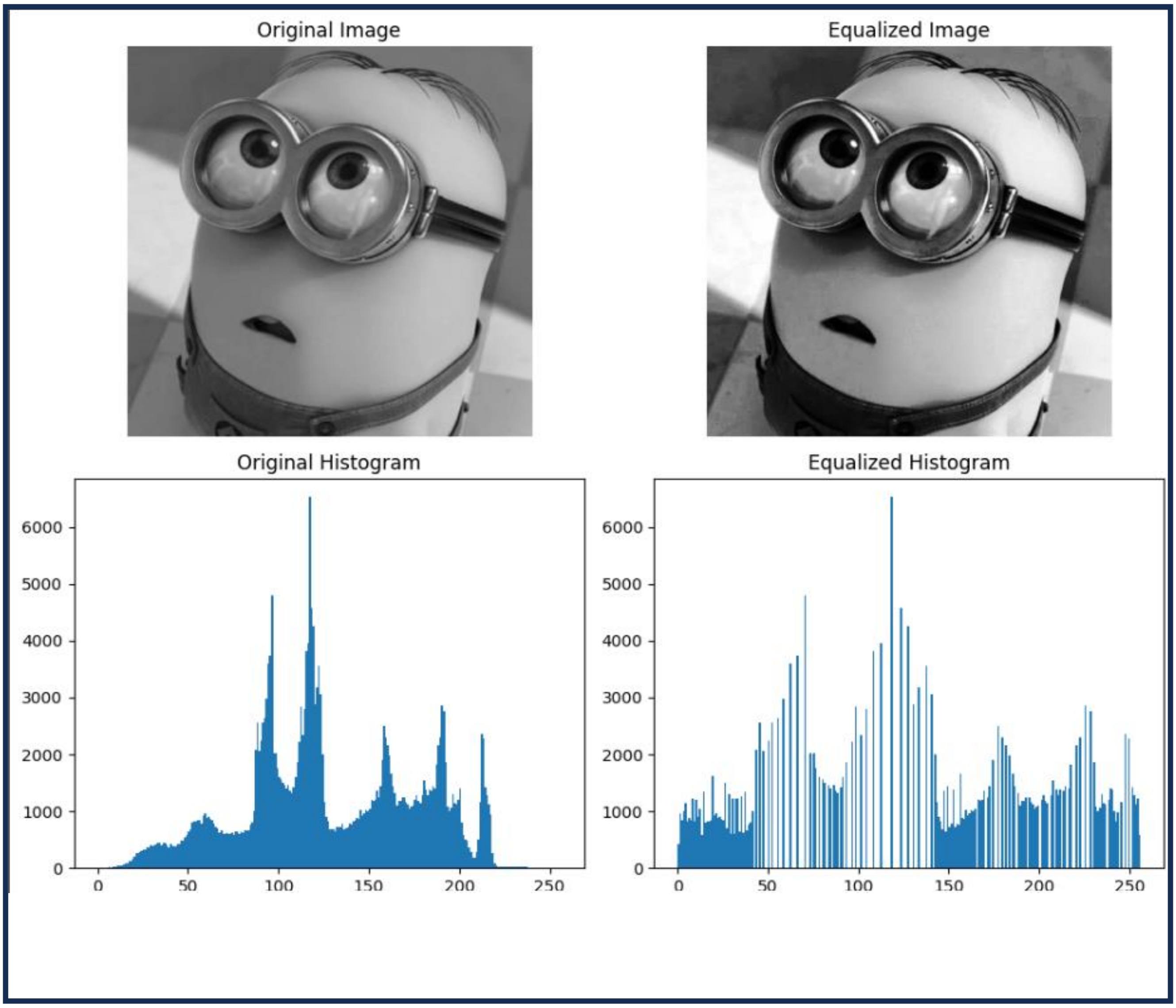
HISTOGRAM EQUALIZATION

INPUT:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img_path = r"   /content/mini.jpg"
img = cv.imread(img_path, cv.IMREAD_GRAYSCALE)

# Check if the image is loaded correctly
if img is None:
   print(f"Error: Unable to load image at {img_path}")
else:
   # Apply Histogram Equalization
   equalized_img = cv.equalizeHist(img)

   # Save the resulting image
   cv.imwrite('equalized_image.png', equalized_img)

   # Plotting the original and equalized images and their histograms
   plt.figure(figsize=(10, 8))

   plt.subplot(2, 2, 1)
   plt.imshow(img, cmap='gray')
   plt.title('Original Image')
   plt.axis('off')

   plt.subplot(2, 2, 2)
   plt.imshow(equalized_img, cmap='gray')
   plt.title('Equalized Image')
   plt.axis('off')
```

```
plt.subplot(2, 2, 3)
plt.hist(img.ravel(), 256, [0, 256])
plt.title(‘Original Histogram’)

plt.subplot(2, 2, 4)
plt.hist(equalized_img.ravel(), 256, [0, 256])
plt.title(‘Equalized Histogram’)

plt.tight_layout()
plt.show()
```

OUTPUT:



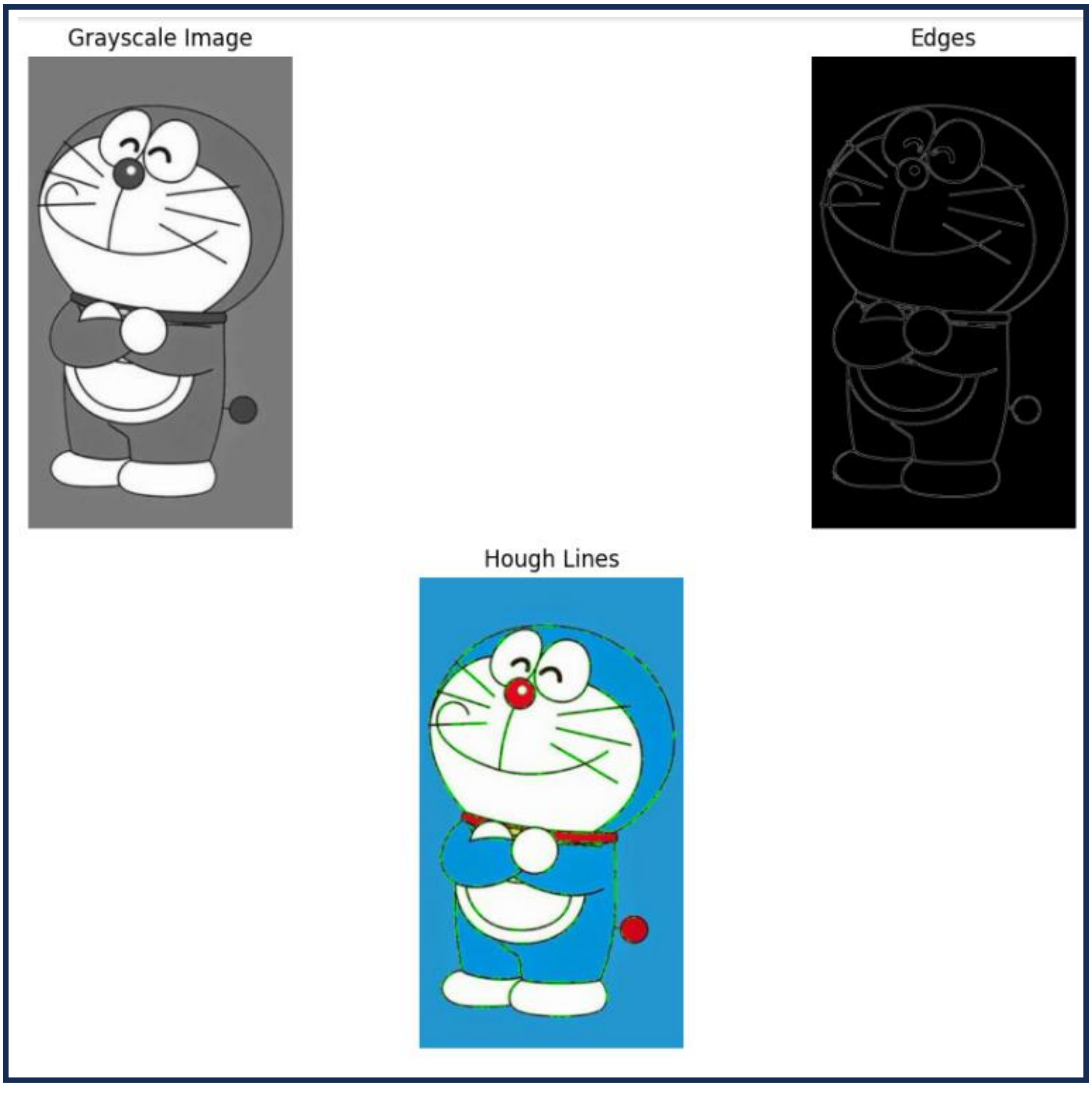
HOUGH LINE

INPUT:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

from google.colab import files


img = cv2.imread("/content/dore.webp")


# Convert the image to grayscale

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


# Perform Canny edge detection

edges = cv2.Canny(gray, 50, 150)  # Adjusted threshold values


# Parameters for Hough transform

minLineLength = 3

maxLineGap = 1


# Perform Hough line detection

lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 50, minLineLength, maxLineGap)  # Adjusted parameters

# Create a new figure

plt.figure(figsize=(12, 8))

# Display grayscale image

plt.subplot(2, 2, 1)

plt.imshow(gray, cmap='gray')

plt.title('Grayscale Image')

plt.axis('off')

# Display edge-detected image

plt.subplot(2, 2, 2)

plt.imshow(edges, cmap='gray')

plt.title('Edges')

plt.axis('off')
```

```
# Display the original image with Hough lines overlayed
plt.subplot(2, 2, (3, 4))
img_with_lines = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
if lines is not None:
   for line in lines:
      x1, y1, x2, y2 = line[0]
      cv2.line(img_with_lines, (x1, y1), (x2, y2), (0, 255, 0), 2)  # Draw lines on the image
plt.imshow(img_with_lines)
plt.title('Hough Lines')
plt.axis('off')
plt.tight_layout()
plt.show()
```

OUTPUT:



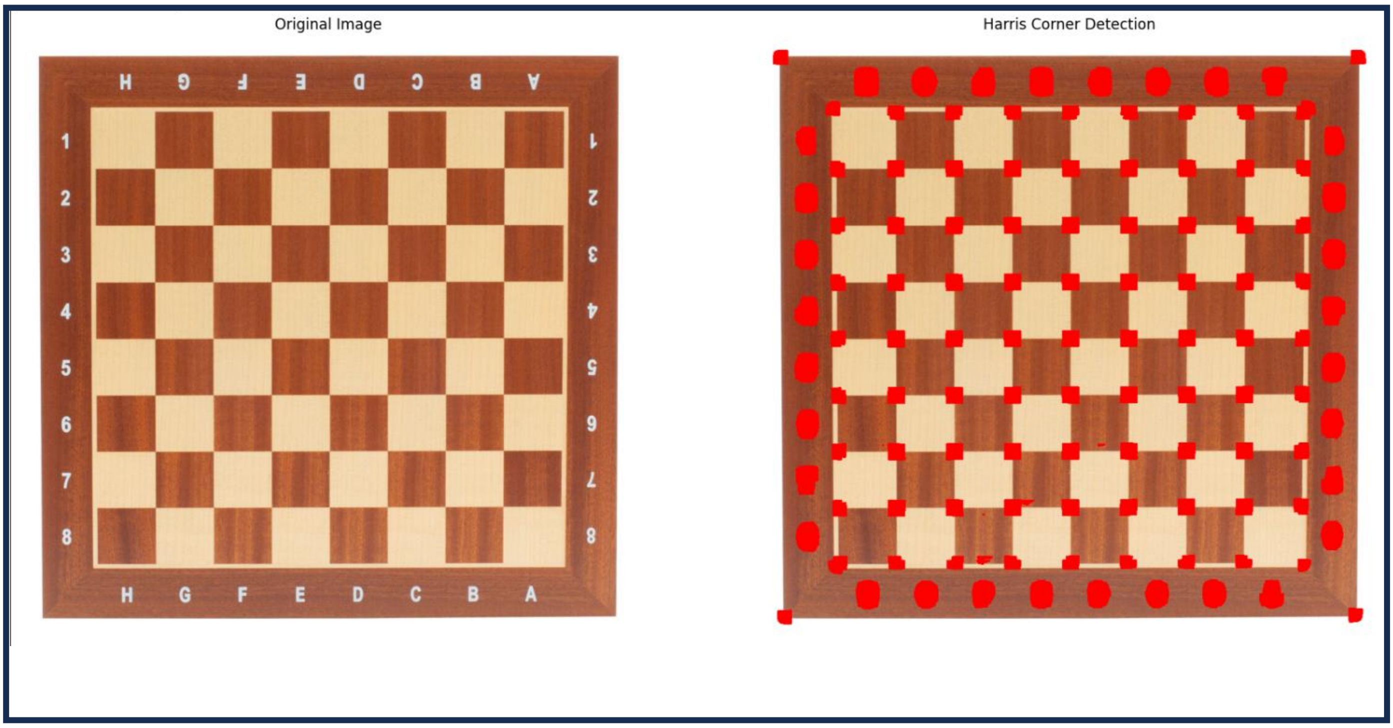
HARRIS CORNER DETECTION

INPUT:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab import files

filename = 'chess.jpg'
img = cv2.imread(filename)
if img is None:
   raise FileNotFoundError(f"Image not found: {filename}")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray, 25, 3, 0.04)
dst = cv2.dilate(dst, None)
img[dst > 0.01 * dst.max()] = [0, 0, 255]

original_img_rgb = cv2.cvtColor(cv2.imread(filename), cv2.COLOR_BGR2RGB)
detected_img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(20, 10))
# Original image
plt.subplot(1, 2, 1)
plt.imshow(original_img_rgb)
plt.title('Original Image')
plt.axis('off')

# Image with detected corners
plt.subplot(1, 2, 2)
plt.imshow(detected_img_rgb)
plt.title('Harris Corner Detection')
```

```
plt.axis('off')
```

OUTPUT:



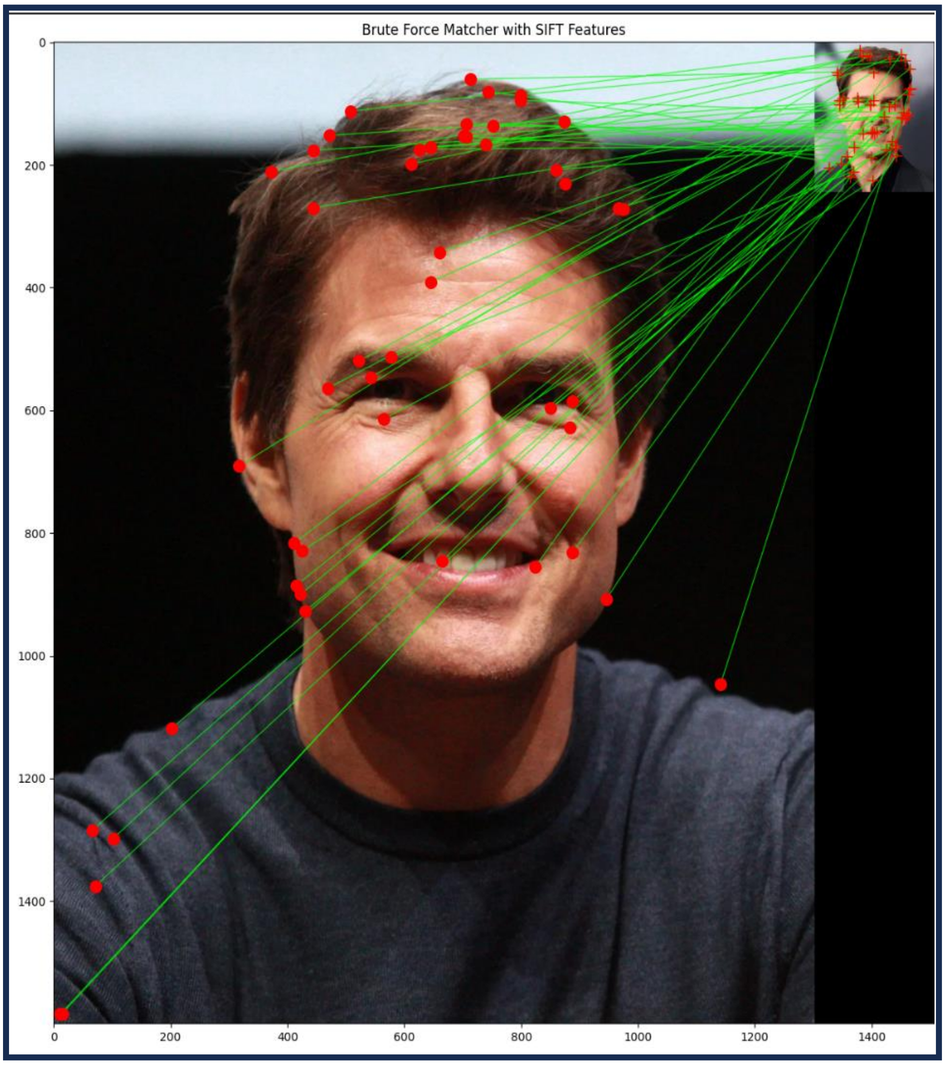
BRUTE FORCE MATCHERS METHOD

INPUT:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

image1_path = ("/content/indian-boys-group-crowds-park-EC1DWE.jpg")
image2_path = ("/content/indian-boys-group-crowds-park-EC1DWE.jpg")

img1 = cv2.imread(image1_path)
img2 = cv2.imread(image2_path)

gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray1, None)
kp2, des2 = sift.detectAndCompute(gray2, None)

bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
matches = bf.match(des1, des2)
matches = sorted(matches, key=lambda x: x.distance)

img_matches = np.empty((max(gray1.shape[0], gray2.shape[0]), gray1.shape[1] + gray2.shape[1], 3), dtype=np.uint8)
cv2.drawMatches(img1, kp1, img2, kp2, matches[:50], img_matches,
                matchColor=(0, 255, 0), singlePointColor=(255, 0, 0), flags=2)

plt.figure(figsize=(15, 15))
plt.imshow(cv2.cvtColor(img_matches, cv2.COLOR_BGR2RGB))
plt.title('Brute Force Matcher with SIFT Features')
```

```
for match in matches[:50]:
    img1_idx = match.queryIdx
    img2_idx = match.trainIdx
    (x1, y1) = kp1[img1_idx].pt
    (x2, y2) = kp2[img2_idx].pt

    plt.plot(x2 + gray1.shape[1], y2, 'r+', markersize=10)
    plt.plot(x1, y1, 'ro', markersize=10)

plt.axis('off')
plt.show()
```

OUTPUT:



WATERSHED ALGORITHM

INPUT:

```
import cv2

import numpy as np

from scipy.ndimage import distance_transform_edt

from skimage import color, segmentation, filters, morphology

import matplotlib.pyplot as plt

from google.colab import files


# Step 1: Upload the image to Google Colab

uploaded = files.upload()


# Step 2: Load the uploaded image

image_path = list(uploaded.keys())[0]

I = cv2.imread("/content/Application photo.jpg")

I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)


# Step 3: Convert to Grayscale (if necessary)

if len(I.shape) == 3:

   I_gray = cv2.cvtColor(I, cv2.COLOR_RGB2GRAY)


# Step 4: Noise Reduction

I_filtered = cv2.GaussianBlur(I_gray, (5, 5), 2)


# Step 5: Compute the Gradient Magnitude

Ix = filters.sobel_h(I_filtered)

Iy = filters.sobel_v(I_filtered)

gradmag = np.sqrt(Ix*2 + Iy*2)


# Step 6: Marker-Based Segmentation

# Compute the distance transform

ret, binary_image = cv2.threshold(I_filtered, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
```

```
D = distance_transform_edt(binary_image)

# Identify regional maxima and create markers
local_maxi = morphology.local_maxima(D)
markers = np.zeros_like(I_filtered, dtype=int)
markers[local_maxi] = np.arange(1, np.sum(local_maxi) + 1)

# Step 7: Apply the Watershed Transform
markers = segmentation.watershed(-D, markers, mask=binary_image)

# Define 7 distinct colors for visualization
colors = [
    [255, 0, 0],    # Red
    [0, 255, 0],    # Green
    [0, 0, 255],    # Blue
    [255, 255, 0],  # Yellow
    [0, 255, 255],  # Cyan
    [255, 0, 255],  # Magenta
    [192, 192, 192] # Gray
]
# Map each label to one of the 7 colors
Lrgb = np.zeros((*markers.shape, 3), dtype=np.uint8)
unique_labels = np.unique(markers)

for k, label in enumerate(unique_labels):
    if label == 0:
        continue  # Skip the background
    mask = markers == label
    color_idx = k % 7
    for c in range(3):
        Lrgb[:, :, c] += (mask * colors[color_idx][c]).astype(np.uint8)
```

Step 8: Visualize the Results

fig, axes = plt.subplots(1, 3, figsize=(18, 6))

axes[0].imshow(I)

axes[0].set_title('Original Image')

axes[0].axis('off')

axes[1].imshow(gradmag, cmap='gray')

axes[1].set_title('Gradient Magnitude')

axes[1].axis('off')

axes[2].imshow(Lrgb)

axes[2].set_title('Watershed Transform')

axes[2].axis('off')

plt.show()

OUTPUT:

K MEANS CLUSTERING

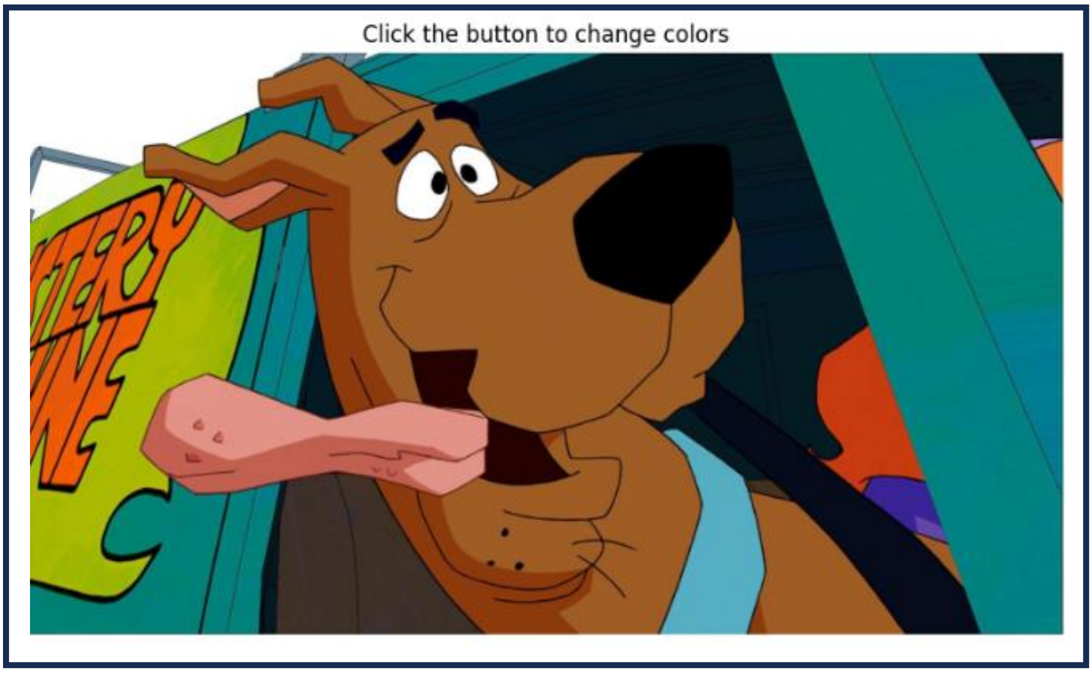
INPUT:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from google.colab import files
import cv2
import IPython
from IPython.display import display, clear_output
import ipywidgets as widgets

# Step 1: Upload the image
uploaded = files.upload()

# Step 2: Load the uploaded image
# Assuming only one image is uploaded and we access the file name
image_path = next(iter(uploaded))
image = cv2.imread(image_path)

# Convert image from BGR (OpenCV default) to RGB for proper display in matplotlib
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 3: Reshape the image into a 2D array of pixel values
rows, cols, channels = image.shape
pixels = image.reshape(rows * cols, channels).astype(float)

# Step 4: Define the number of clusters (K)
num_clusters = 5  # You can change this to any desired number of clusters
```

```python
# Step 5: Apply K-Means clustering
kmeans = KMeans(n_clusters=num_clusters, max_iter=200, random_state=42)
cluster_idx = kmeans.fit_predict(pixels)
cluster_centers = kmeans.cluster_centers_

# Function to update the image with new random colors
def update_image():
    random_colors = np.random.randint(0, 255, (num_clusters, 3))

    recolored_pixels = np.zeros_like(pixels)
    for i in range(num_clusters):
        recolored_pixels[cluster_idx == i, :] = random_colors[i, :]

    # Reshape back into image dimensions
    recolored_image = recolored_pixels.reshape(rows, cols, channels).astype(np.uint8)

    # Display the updated image
    plt.figure(figsize=(10, 8))
    plt.imshow(recolored_image)
    plt.title('Colors changed! Click button again for new colors')
    plt.axis('off')
    plt.show()

# Create a button widget
button = widgets.Button(description="Change Colors")

# Define the button click event handler
def on_button_click(b):
    update_image()

# Assign the click event handler to the button
button.on_click(on_button_click)
# Display the button
display(button)
```

```
# Initially display the original image
plt.figure(figsize=(10, 8))
plt.imshow(image)
plt.title('Click the button to change colors')
plt.axis('off')
plt.show()
```

OUTPUT:



PCA DIMENSIONAL REDUCTION

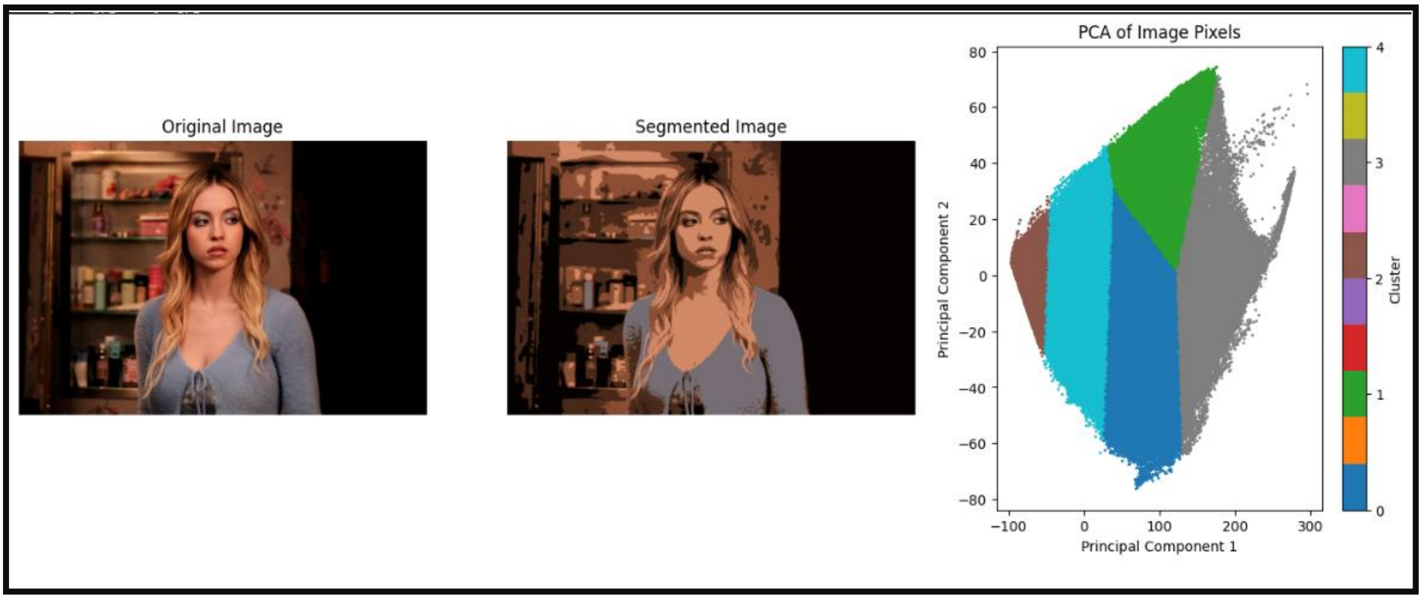
INPUT:

```
# Import necessary libraries
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from google.colab import files

# Upload the image
uploaded = files.upload()

# Extract the filename from the uploaded files
image_path = list(uploaded.keys())[0]

# Function to perform K-means clustering on image pixels and plot PCA
def kmeans_on_image(image_path, cluster_n):
  # Load the image
  img = cv.imread(image_path)
  if img is None:
    raise ValueError("Image not found or unable to load.")

  # Convert the image to RGB
  img_rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)

  # Reshape the image to a 2D array of pixels
  pixel_values = img_rgb.reshape((-1, 3))
  pixel_values = np.float32(pixel_values)
```

```
# Perform K-means clustering
  term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
  _, labels, centers = cv.kmeans(pixel_values, cluster_n, None, term_crit, 10, cv.KMEANS_RANDOM_CENTERS)

  # Convert centers to uint8
  centers = np.uint8(centers)

  # Map the labels to center colors
  segmented_img = centers[labels.flatten()]

  # Reshape segmented image to original dimensions
  segmented_img = segmented_img.reshape(img_rgb.shape)

  # Apply PCA for 2D visualization
  pca = PCA(n_components=2)
  pixel_values_pca = pca.fit_transform(pixel_values)

  # Create a scatter plot for PCA
  plt.figure(figsize=(18, 6))

  # Plot the original and segmented images
  plt.subplot(1, 3, 1)
  plt.imshow(img_rgb)
  plt.title('Original Image')
  plt.axis('off')

  plt.subplot(1, 3, 2)
  plt.imshow(segmented_img)
  plt.title('Segmented Image')
  plt.axis('off')

  # Plot PCA results
  plt.subplot(1, 3, 3)
  scatter = plt.scatter(pixel_values_pca[:, 0], pixel_values_pca[:, 1], c=labels.flatten(), cmap='tab10', s=1)
```

```
plt.colorbar(scatter, ticks=range(cluster_n), label='Cluster')
    plt.title('PCA of Image Pixels')
    plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

    plt.show()

    silhouette_avg = silhouette_score(pixel_values, labels.flatten())
    print(f"Silhouette Score: {silhouette_avg}")

# Number of clusters
cluster_n = 5
kmeans_on_image(image_path, cluster_n)
```

OUTPUT:



OBJECT DETECTION

INPUT:

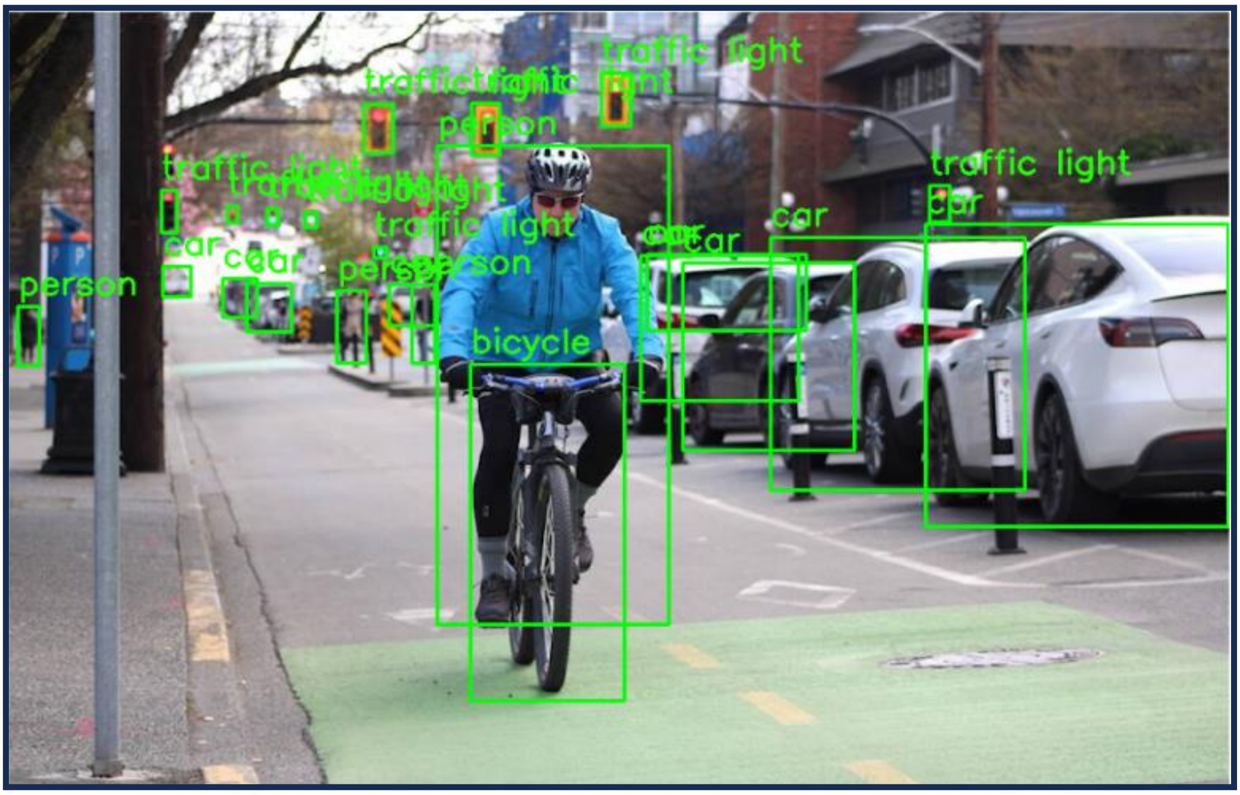
#step 1 : pip install libraries

```
# Step 2: Import Libraries
import cv2
import numpy as np
import torch
from torchvision import models
import torchvision.transforms as T
import matplotlib.pyplot as plt
from google.colab import files

# Step 3: Upload Image File
uploaded = files.upload()

# Step 4: Get the Uploaded File Name
image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file

# Verify if the image was uploaded successfully
if image_path:
    print(f"Image '{image_path}' uploaded successfully.")
else:
    print("Error: No image was uploaded. Please try again.")

# Step 5: Load the Pre-trained Mask R-CNN Model
model = models.detection.maskrcnn_resnet50_fpn(pretrained=True)
model.eval()

# Step 6: Define Function to Perform Object Detection
def detect_objects(image_path):
    # Read the input image
    image = cv2.imread(image_path)
```

```
# Check if the image is loaded properly
    if image is None:
        print("Error: Image not loaded. Check the file path or ensure the file exists.")
        return

    # Convert BGR image to RGB
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Transform image to match the model's input requirements
    transform = T.Compose([T.ToTensor()])
    image_tensor = transform(image_rgb)

    # Perform object detection
    with torch.no_grad():
        output = model([image_tensor])

    # Get the detected boxes, labels, and scores
    boxes = output[0]['boxes'].numpy()
    labels = output[0]['labels'].numpy()
    scores = output[0]['scores'].numpy()

    # Define labels for COCO dataset
    coco_labels = [
        '_background_', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat',
        'traffic light', 'fire hydrant', 'N/A', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse',
        'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N/A',
        'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat',
        'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'N/A', 'wine glass', 'cup', 'fork',
        'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
        'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table', 'N/A', 'N/A', 'toilet',
        'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink',
        'refrigerator', 'N/A', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
    ]
```

```
# Set detection threshold
    detection_threshold = 0.7

    # Plot the detections on the image
    for i in range(len(boxes)):
        if scores[i] > detection_threshold:
            # Draw the bounding box
            (x1, y1, x2, y2) = boxes[i].astype(int)
            cv2.rectangle(image_rgb, (x1, y1), (x2, y2), (0, 255, 0), 2)

            # Add label
            label = coco_labels[labels[i]]
            cv2.putText(image_rgb, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36, 255, 12), 2)

    # Display the image
    plt.figure(figsize=(12, 8))
    plt.imshow(image_rgb)
    plt.axis('off')
    plt.show()

# Step 7: Run Detection on Your Image
if image_path:
    detect_objects(image_path)
```

OUTPUT:



traffic light
traffic light
person
traffic light
traffic light
car
person
car
car
car
car
person
traffic light
person
bicycle
car
car
traffic light
car