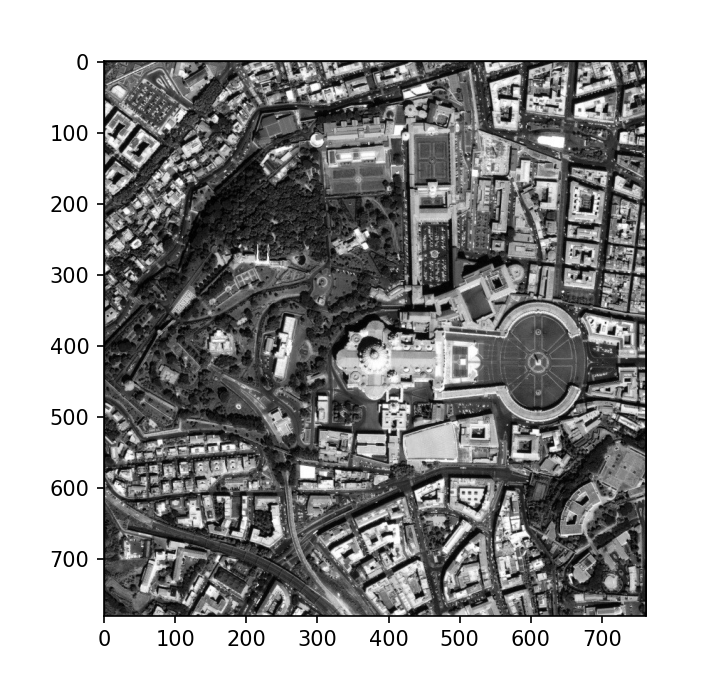
CODE IMPLEMENTATION

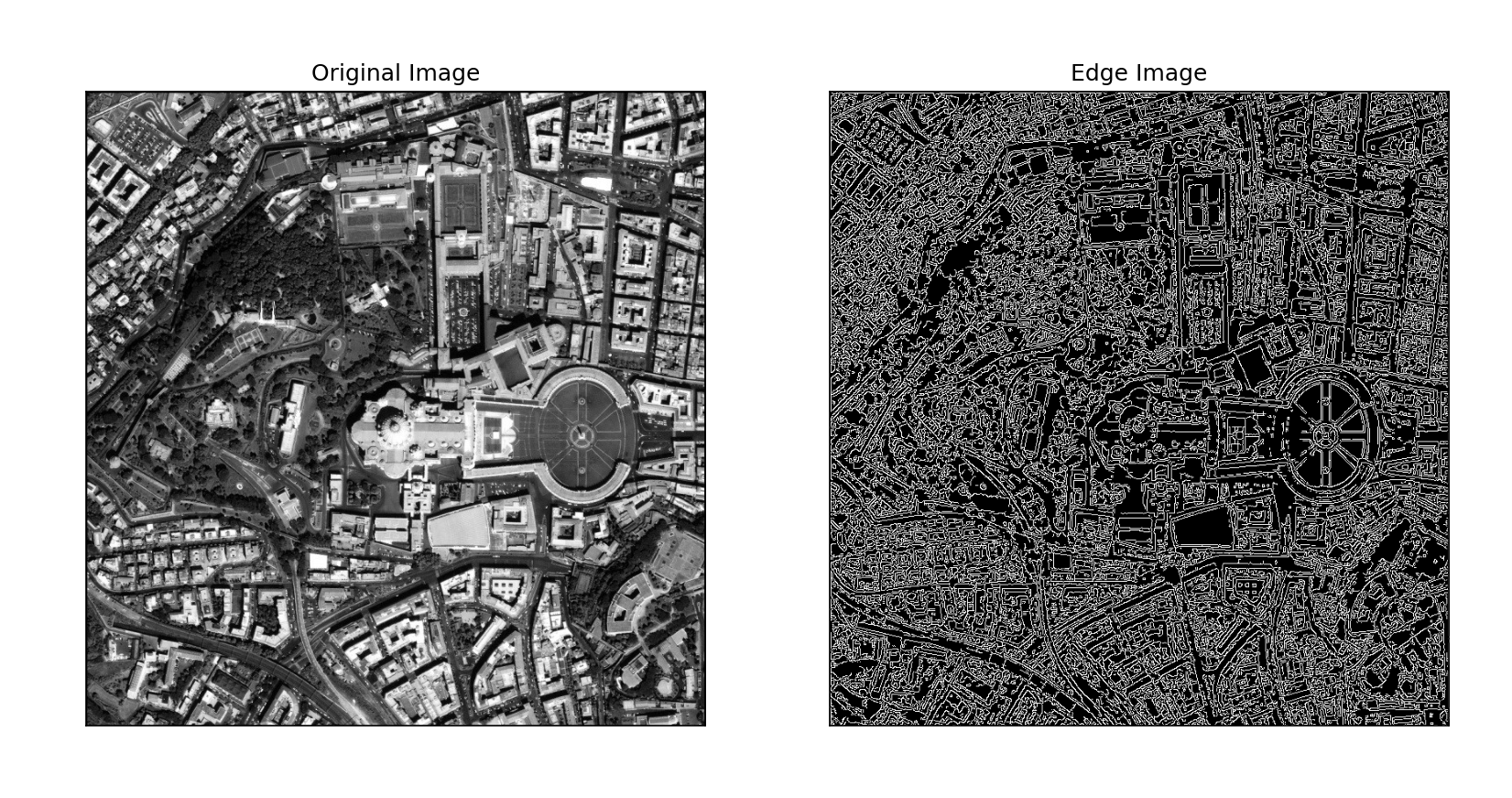
GRAYSCALE CONVERSION:

import cv2  
import matplotlib.pyplot as plot  
  
img = cv2.imread('data.jpg', cv2.IMREAD\_GRAYSCALE)  
  
#diplaying the gray image  
plot.imshow(img,cmap="gray")  
plot.show()



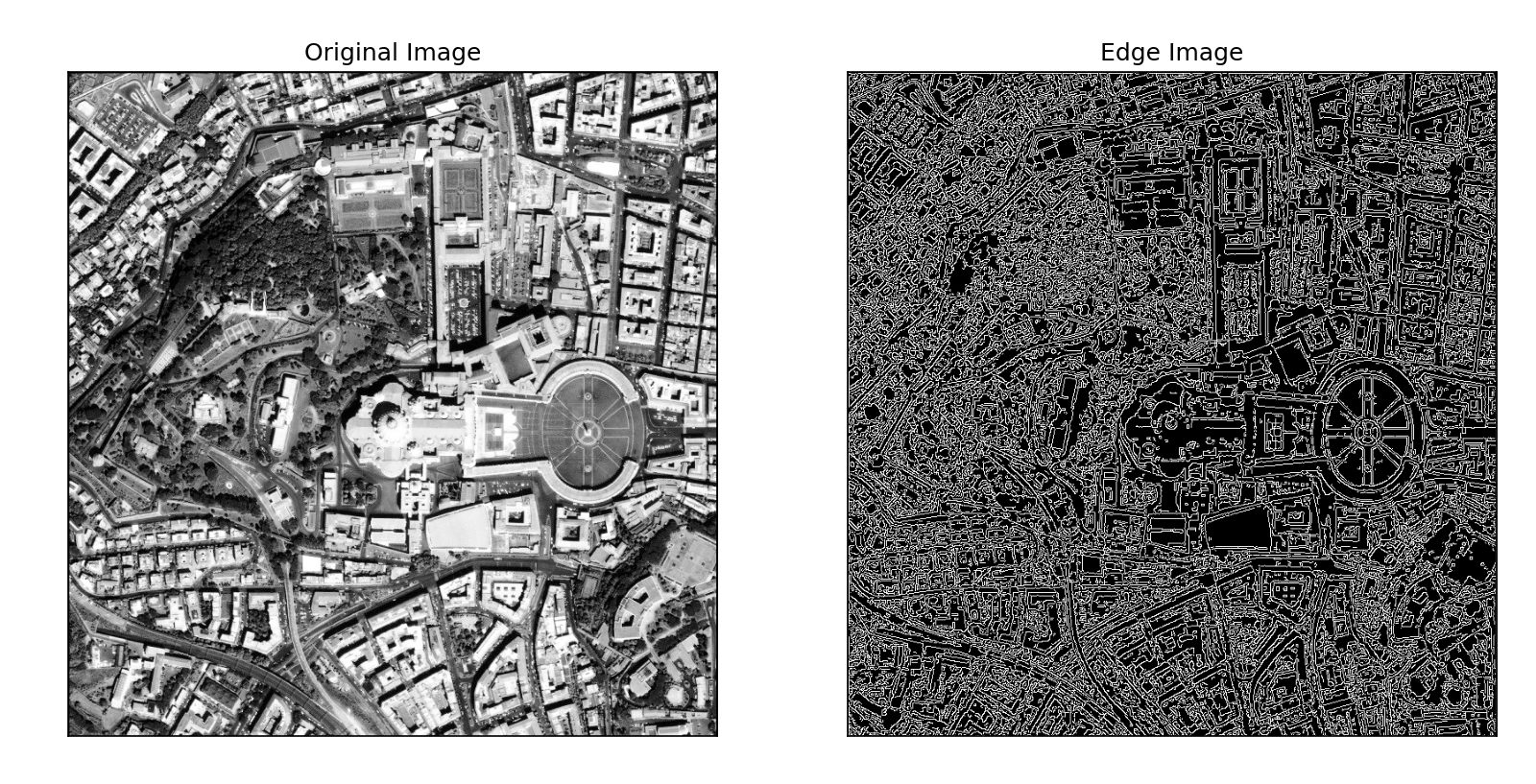
Canny edge deduction:

import cv2 as cv  
from matplotlib import pyplot as plt  
  
#gray scale conversion  
img = cv.imread('data.jpg', cv.IMREAD\_GRAYSCALE)  
  
assert img is not None, "file could not be read, check with os.path.exists()"  
  
edges = cv.Canny(img,100,200)  
plt.subplot(121),plt.imshow(img,cmap = 'gray')  
plt.title('Original Image'), plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(edges,cmap = 'gray')  
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])  
plt.show()



Canny edge deduction of histogram equalized image:

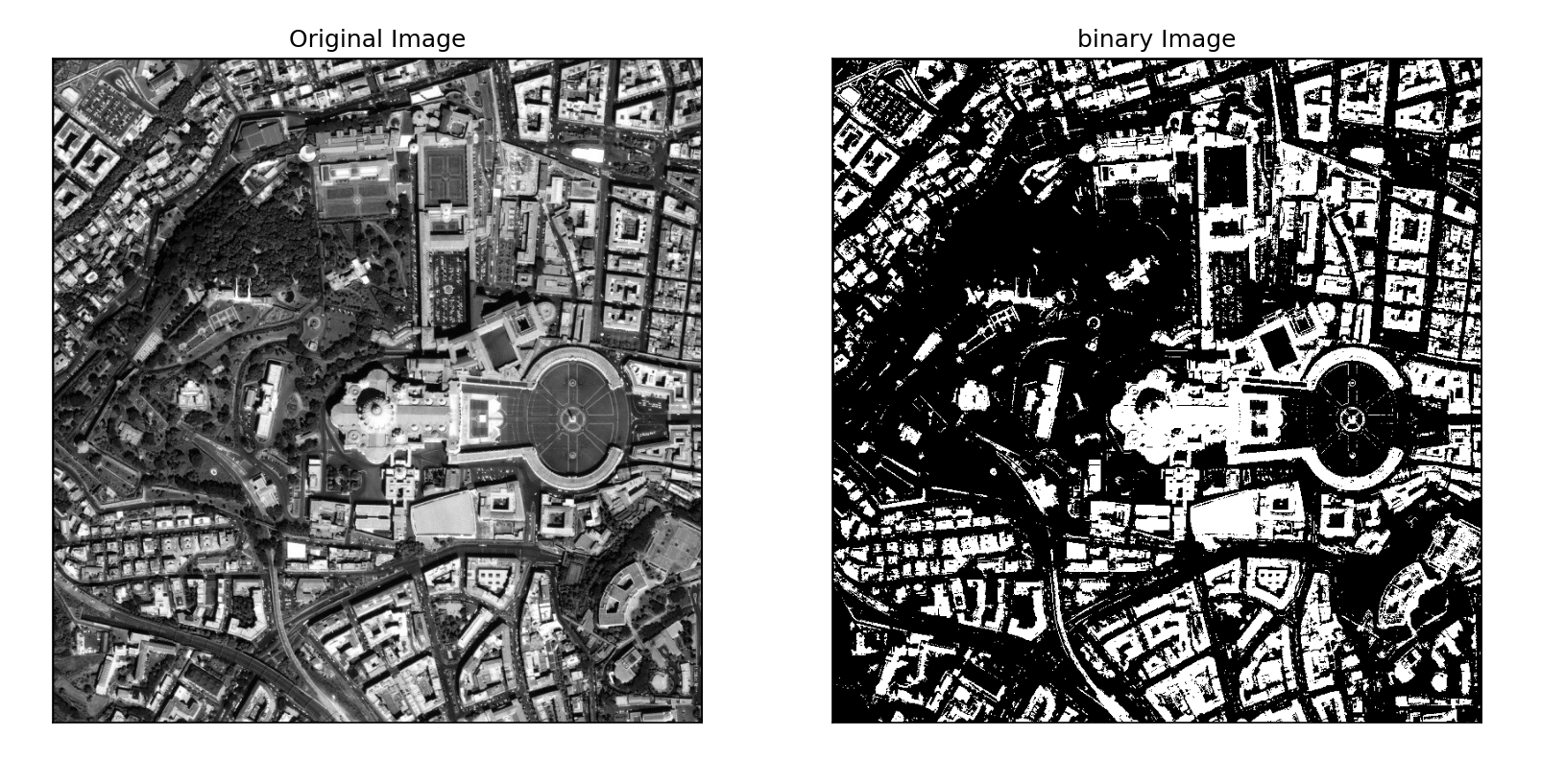
import cv2 as cv  
import matplotlib.pyplot as plt  
  
img = cv.imread('data.jpg',0)  
img = cv.equalizeHist(img)  
  
edges = cv.Canny(img,100,200)  
plt.subplot(121),plt.imshow(img,cmap = 'gray')  
plt.title('Original Image'), plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(edges,cmap = 'gray')  
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])  
plt.show()



Binary image conversion:

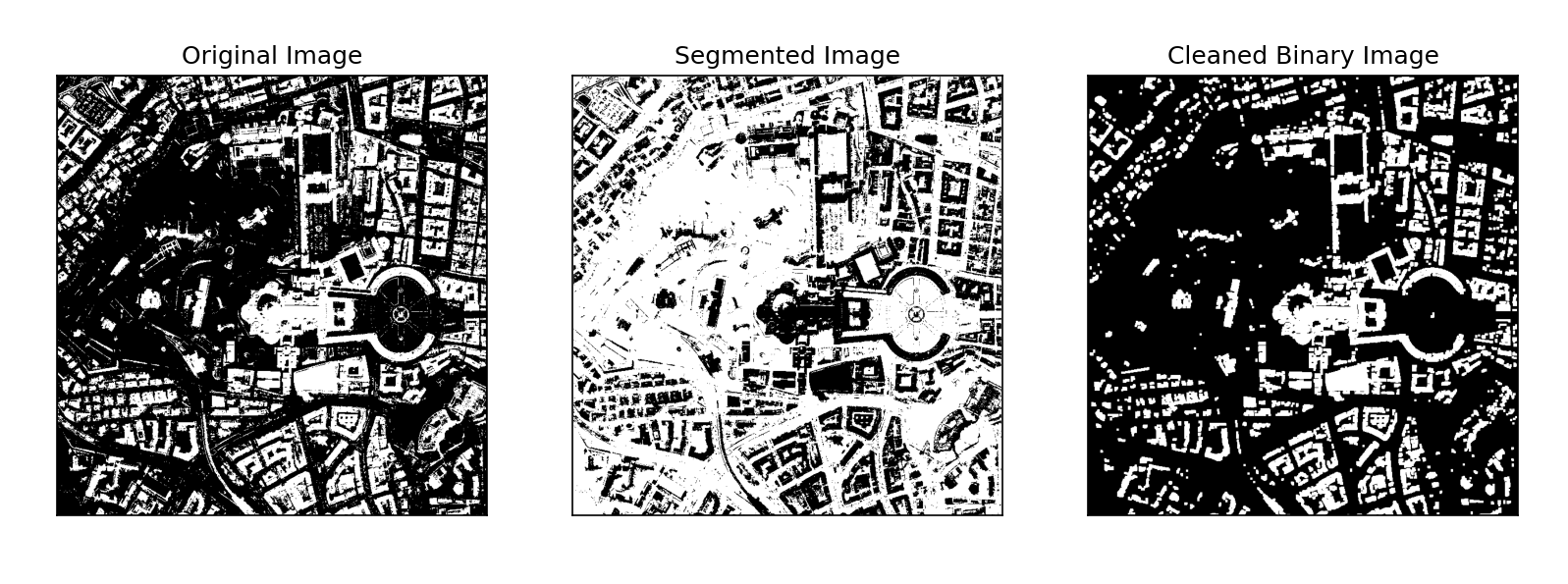
import cv2  
import matplotlib.pyplot as plt  
  
def convert\_to\_binary(image\_path, threshold\_value=127):  
 # Read the image  
 image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
  
 # Apply binary thresholding  
 \_, binary\_image = cv2.threshold(image, threshold\_value, 255, cv2.THRESH\_BINARY)  
  
 plt.subplot(121), plt.imshow(image, cmap='gray')  
 plt.title('Original Image'), plt.xticks([]), plt.yticks([])  
 plt.subplot(122), plt.imshow(binary\_image, cmap="gray")  
 plt.title('binary Image'), plt.xticks([]), plt.yticks([])  
 plt.show()  
  
# Example usage

image\_path = 'data.jpg'  
convert\_to\_binary(image\_path)



Noise removed from segmented image :

import cv2  
import numpy as np  
from sklearn.cluster import KMeans  
import matplotlib.pyplot as plt  
  
  
def convert\_to\_binary(image\_path, threshold\_value=127):  
 # Read the image  
 image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
  
 # Apply binary thresholding  
 \_, binary\_image = cv2.threshold(image, threshold\_value, 255, cv2.THRESH\_BINARY)  
  
 return binary\_image  
  
  
def remove\_noise(image, kernel\_size=4):  
 # Define the kernel for morphological operations  
 kernel = np.ones((kernel\_size, kernel\_size), np.uint8)  
  
 # Apply erosion followed by dilation to remove noise  
 cleaned\_image = cv2.morphologyEx(image, cv2.MORPH\_OPEN, kernel)  
  
 return cleaned\_image  
  
  
def segment\_image(image, num\_clusters=3):  
 # Reshape the image to be a list of pixels  
 pixels = image.reshape((-1, 1))  
  
 # Apply k-means clustering  
 kmeans = KMeans(n\_clusters=num\_clusters)  
 kmeans.fit(pixels)  
  
 # Get the labels for each pixel  
 labels = kmeans.labels\_  
  
 # Reshape labels to the shape of the original image  
 segmented\_image = labels.reshape(image.shape)  
  
 # Convert labels to 8-bit for display  
 segmented\_image = (segmented\_image \* (255 / (num\_clusters - 1))).astype(np.uint8)  
  
 # Display the original, noisy, and segmented images  
 plt.subplot(131), plt.imshow(image, cmap="gray")  
 plt.title('Original Image'), plt.xticks([]), plt.yticks([])  
  
 # Remove noise from the binary image  
 cleaned\_binary\_image = remove\_noise(image)  
  
 plt.subplot(133), plt.imshow(cleaned\_binary\_image, cmap="gray")  
 plt.title('Cleaned Binary Image'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(132), plt.imshow(segmented\_image, cmap="gray")  
 plt.title('Segmented Image'), plt.xticks([]), plt.yticks([])  
  
 plt.show()  
  
  
# Example usage  
image\_path = 'data.jpg'  
binary\_image = convert\_to\_binary(image\_path)  
segment\_image(binary\_image)



Detecting Building Boundaries:

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
def preprocess\_image(image\_path, threshold\_value=127, noise\_removal\_kernel=3):  
 # Read the color image  
 color\_image = cv2.imread(image\_path)  
  
 # Convert to grayscale  
 gray\_image = cv2.cvtColor(color\_image, cv2.COLOR\_BGR2GRAY)  
  
 # Apply binary thresholding  
 \_, binary\_image = cv2.threshold(gray\_image, threshold\_value, 255, cv2.THRESH\_BINARY)  
  
 # Define the kernel for morphological operations  
 kernel = np.ones((noise\_removal\_kernel, noise\_removal\_kernel), np.uint8)  
  
 # Apply erosion followed by dilation to remove noise  
 cleaned\_binary\_image = cv2.morphologyEx(binary\_image, cv2.MORPH\_OPEN, kernel)  
  
 return color\_image, binary\_image, cleaned\_binary\_image  
  
def detect\_buildings(image, cleaned\_image):  
 # Find contours in the cleaned binary image  
 contours, \_ = cv2.findContours(cleaned\_image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 # Draw contours on the original image  
 building\_boundaries\_image = image.copy()  
 cv2.drawContours(building\_boundaries\_image, contours, -1, (0, 255, 0), thickness=2)  
  
 return building\_boundaries\_image  
  
def segment\_and\_detect(image\_path, threshold\_value=127, noise\_removal\_kernel=3):  
 # Preprocess the image  
 color\_image, binary\_image, cleaned\_binary\_image = preprocess\_image(image\_path, threshold\_value, noise\_removal\_kernel)  
  
 # Detect buildings and draw boundaries on the color image  
 building\_boundaries\_image = detect\_buildings(color\_image, cleaned\_binary\_image)  
  
 # Display the original, binary, and building-boundaries images  
 plt.subplot(131), plt.imshow(cv2.cvtColor(color\_image, cv2.COLOR\_BGR2RGB))  
 plt.title('Original Image'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(132), plt.imshow(binary\_image, cmap="gray")  
 plt.title('Binary Image'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(133), plt.imshow(cv2.cvtColor(building\_boundaries\_image, cv2.COLOR\_BGR2RGB))  
 plt.title('Building Boundaries'), plt.xticks([]), plt.yticks([])  
  
 plt.show()  
  
# Example usage  
image\_path = 'data.jpg'  
segment\_and\_detect(image\_path)

