lab1-numpy

November 19, 2023

[1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

[2]:
```
list = [1, 2, 1, 1, 2, 1, 2, 3, 4, 1, 3, 2]
```

[3]:
```
ar = np.array(list)
```

[4]:
```
ar
```

[4]:
```
array([1, 2, 1, 1, 2, 1, 2, 3, 4, 1, 3, 2])
```

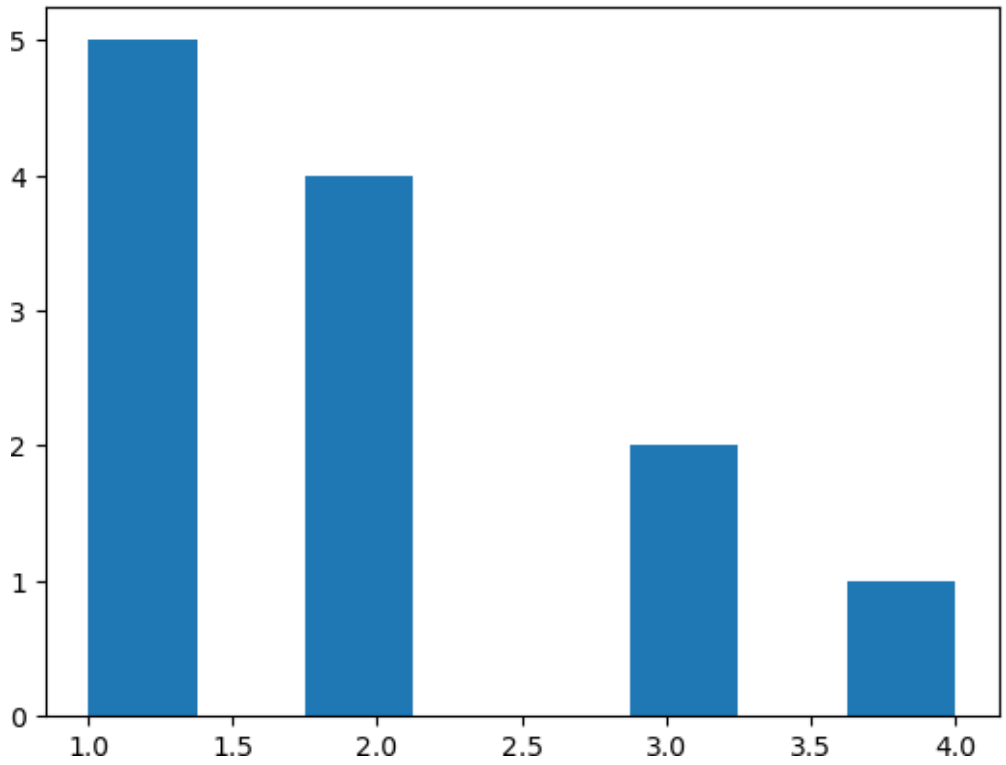
[5]:
```
#1 HISTOGRAM
np.histogram(ar)
```

[5]:
```
(array([5, 0, 0, 4, 0, 0, 2, 0, 0, 1]),
 array([1. , 1.3, 1.6, 1.9, 2.2, 2.5, 2.8, 3.1, 3.4, 3.7, 4. ]))
```

[6]:
```
fig = plt.figure(figsize =(10, 7))
```

```
<Figure size 1000x700 with 0 Axes>
```

[7]:
```
plt.hist(ar, bins = 8)
plt.show()
```



[8]:
```
#2 EUCLIDEAN AND MANHATTAN
l1 = [1, 23, 3,4,5]
l1 = np.array(l1)
l2 = [5, 6, 7, 7, 4]
l2 = np.array(l2)
```

[9]:
```
sub = l1-l2
sub = sub**2
tot = sub.sum()
tot=tot**(1/2)
print("EUCLIDEAN: ", tot)
```

EUCLIDEAN: 18.193405398660254

[10]:
```
sub = abs(l1-l2)
tot = sub.sum()
print("MANHATTAN: ", tot)
```

MANHATTAN: 29

[11]:
```
#3 DOT PRODUCT
nl = l1*l2
```

```
tot = nl.sum()
print("DOT PRODUCT: ", tot)
```

```
DOT PRODUCT:  212
```

[12]:
```
#4FROBENIUS NORM
sq = l1**2
tot = sq.sum()
print("FROBENIUS NORM: ", tot)
```

```
FROBENIUS NORM:  580
```

[13]:
```
#5 ADDITION AND MULTIPLICATION
l1 = [[1,2], [3, 4]]
l1 = np.array(l1)
l2 = [[5, 6], [7, 8]]
l2 = np.array(l2)
```

[14]:
```
sum = l1+l2
mul = np.matmul(l1,l2)
print("ADDITION: ", sum)
print("MULTIPLICATION: ", mul)
```

```
ADDITION:  [[ 6  8]
 [10 12]]
MULTIPLICATION:  [[19 22]
 [43 50]]
```

[15]:
```
ar = l1
```

[16]:
```
#7 NORMALISATION
mi = ar.min()
ma = ar.max()
nl = (ar-mi)/(ma-mi)
print("NORMALIZED: ", nl)
```

```
NORMALIZED:  [[0.         0.33333333]
 [0.66666667 1.        ]]
```

[17]:
```
#8 STANDARDISATION
nl=ar**2
sum=nl.sum()
sum=sum**(1/2)
nl=ar/sum
print("NORMALIZED: ", nl)
```

```
NORMALIZED:  [[0.18257419 0.36514837]
 [0.54772256 0.73029674]]
```

[29]:
```
#9 SUBMATRIX TO ZERO
l1 = [[1,2,3],[4,5,6],[7,8,9]]
l1 = np.array(l1)
l1
```

[29]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[30]:
```
l1[1:, 2:] = 0
```

[31]:
```
print("SUBMATRIX TO ZERO: \n", l1)
```

```
SUBMATRIX TO ZERO:
 [[1 2 3]
 [4 5 0]
 [7 8 0]]
```

[37]:
```
#10 PADDING
l1 = [[1,2,3],[4,5,6],[7,8,9]]
l1 = np.array(l1)
l1=np.pad(l1, 1, mode='constant')
print("PADDING")
l1
```

```
PADDING
```

[37]:
```
array([[0, 0, 0, 0, 0],
       [0, 1, 2, 3, 0],
       [0, 4, 5, 6, 0],
       [0, 7, 8, 9, 0],
       [0, 0, 0, 0, 0]])
```

[ ]:

lab2-basic-image-processing

November 19, 2023

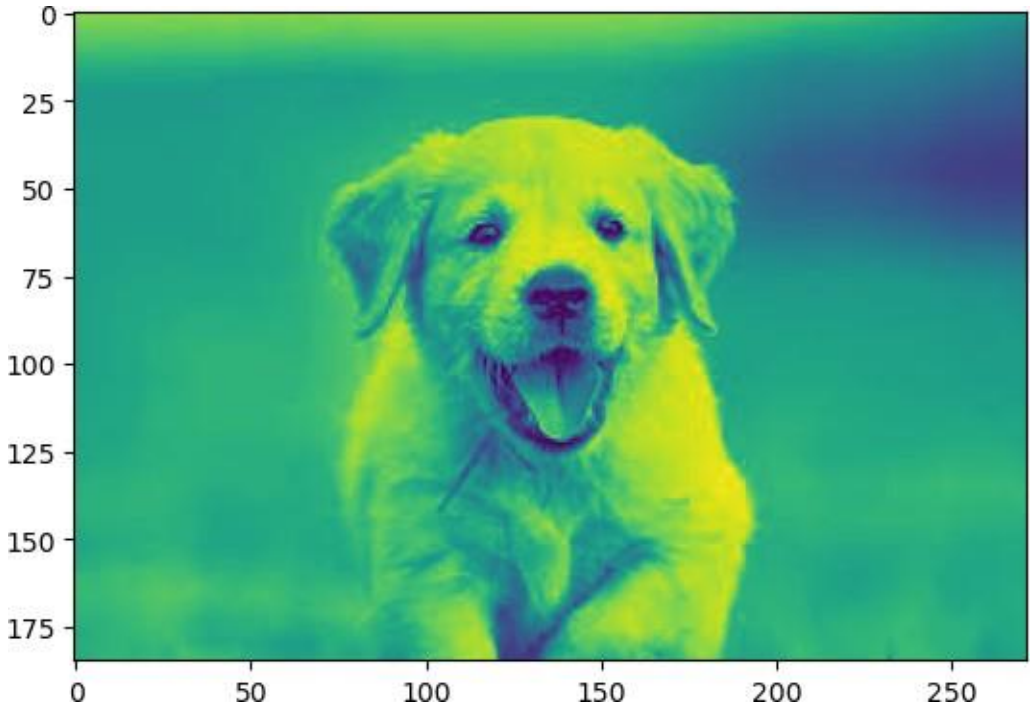
1 Basic Image Processing

[11]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
```

1.1 Reading an Image

[2]:
```
img = cv2.imread('Puppy.jpeg',0)
plt.imshow(img)
```

[2]: <matplotlib.image.AxesImage at 0x1502a7b20>



The image has been correctly loaded by openCV as a numpy array, but the color of each pixel has been sorted as BGR. Matplotlib's plot expects an RGB image so, for a correct display of the image, it is necessary to swap those channels. This operation can be done either by using openCV conversion functions cv2.cvtColor() or by working directly with the numpy array.

1.1.1 Find the shape of image.

[3]:
```
img.shape
#  img.size
```

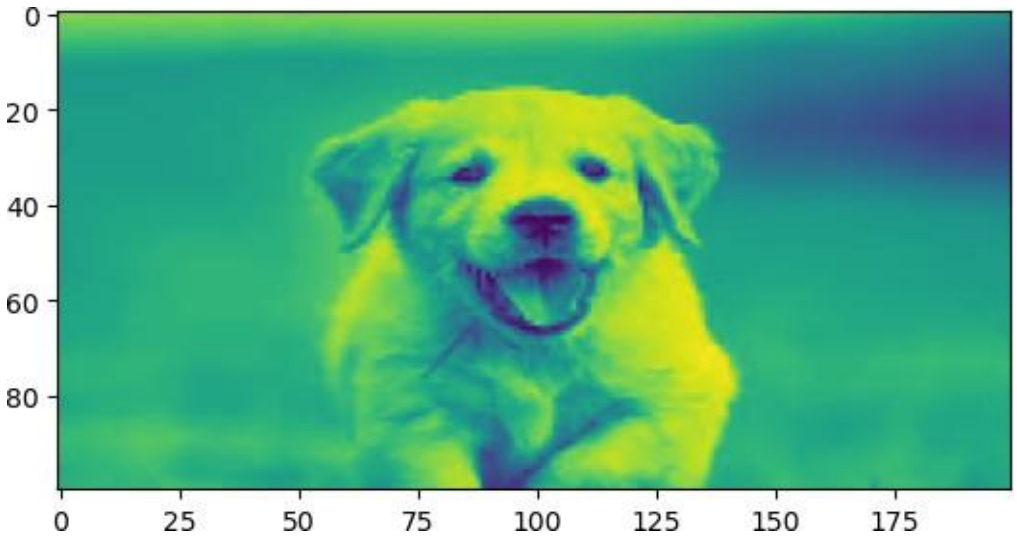
[3]: (185, 272)

1.1.2 Resize the image by 1) providing the specified size 2) providing the scales.

1.1.3 cv2.resize

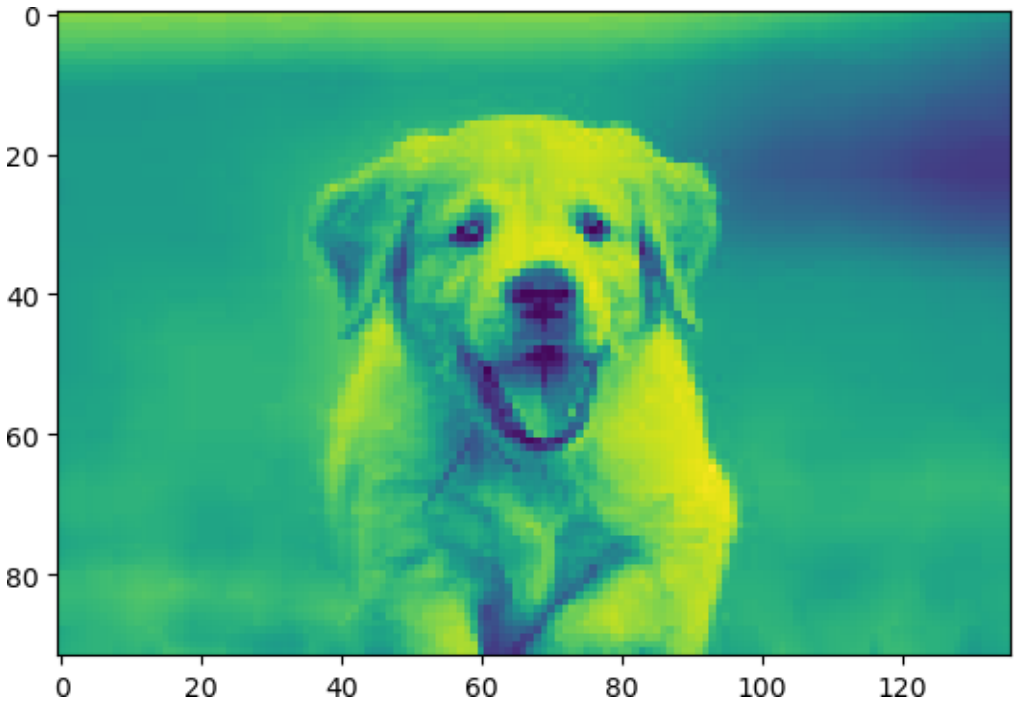
[4]:
```
size = cv2.resize(img, (200, 100))
plt.imshow(size)
```

[4]: <matplotlib.image.AxesImage at 0x11b9480d0>



[9]:
```
p = 0.5
scale = cv2.resize(img, None, fx=p, fy=p)
plt.imshow(scale)
```

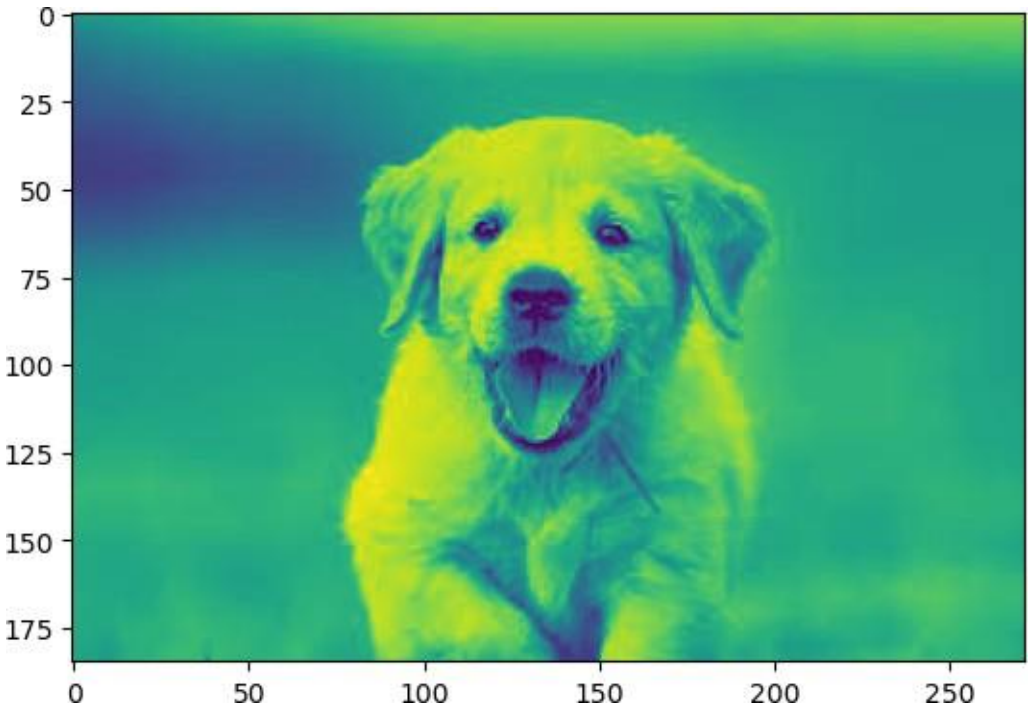
[9]: <matplotlib.image.AxesImage at 0x151b9ff70>



1.1.4 Flip your image 1) along x axis 2) along y axis 3) along both axis

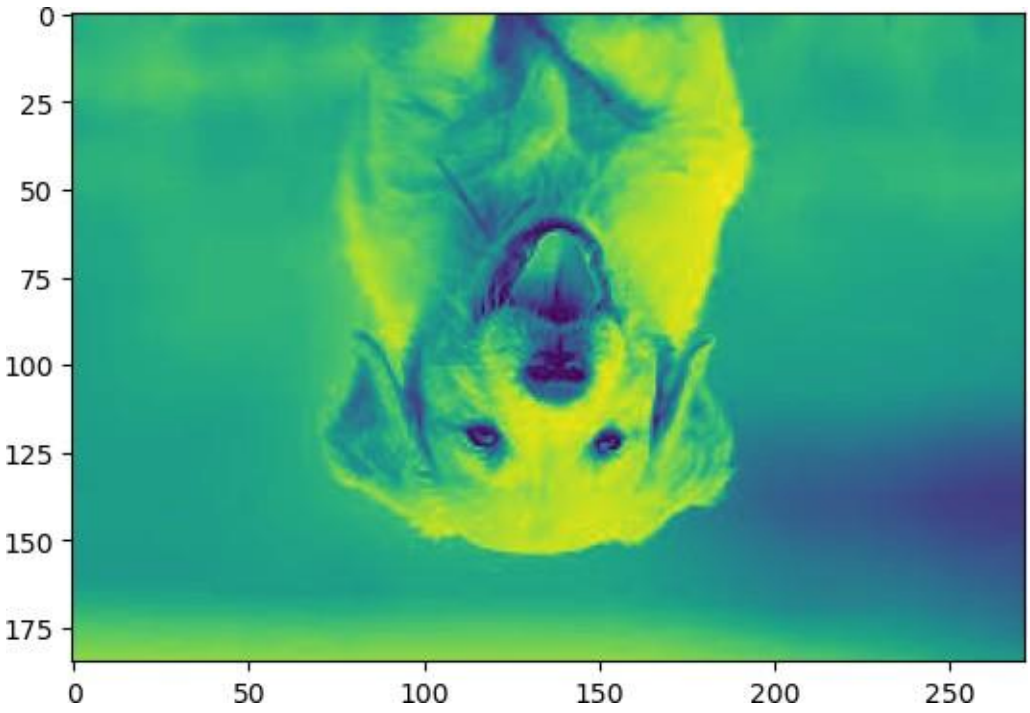
[9]:
```
img_h = cv2.flip(img, 1)
plt.imshow(img_h)
```

[9] : <matplotlib.image.AxesImage at 0x11bacff40>



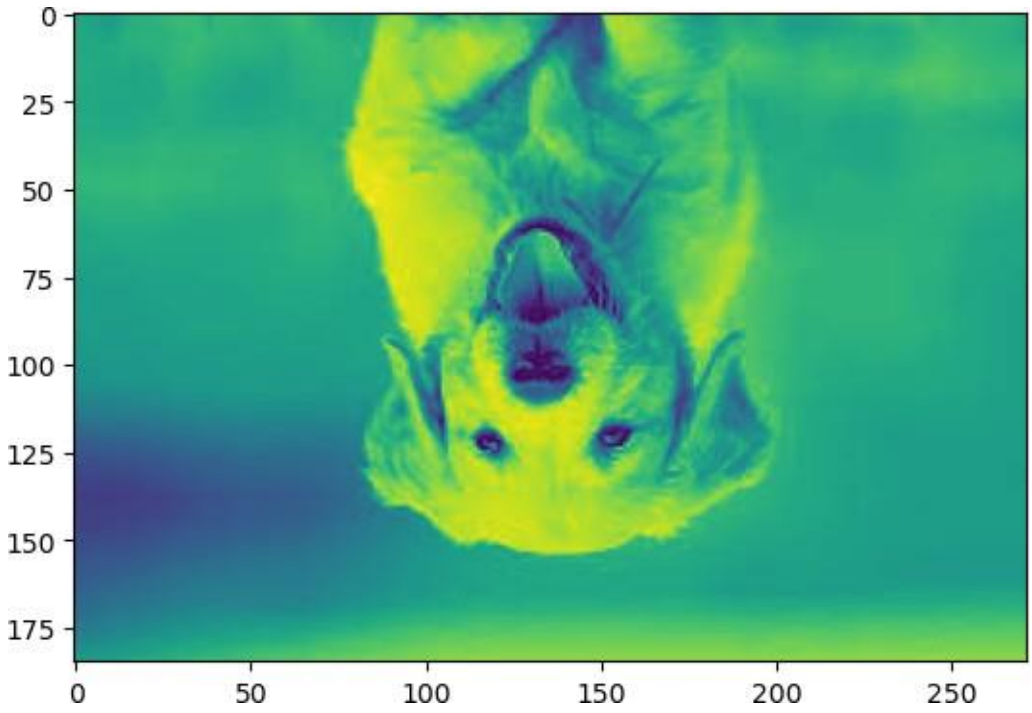
[10]:
```
img_v = cv2.flip(img, 0)
plt.imshow(img_v)
```

[10] : <matplotlib.image.AxesImage at 0x11bb26e60>



[11]:
```
img_vh = cv2.flip(img, -1)
plt.imshow(img_vh)
```

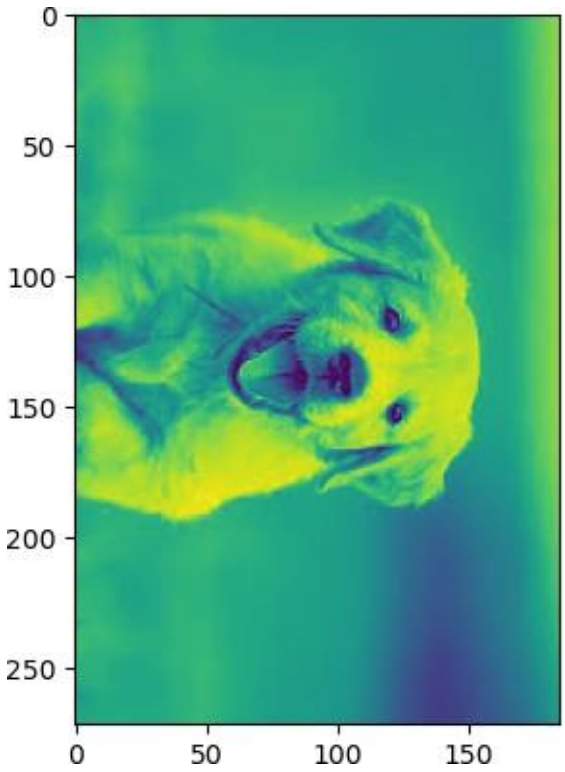
[11] : <matplotlib.image.AxesImage at 0x11bb8dd50>



1.1.5 Rotate the image

[12]:
```
ri = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
plt.imshow(ri)
```

[12] : <matplotlib.image.AxesImage at 0x11bbecaf0>



1.1.6 Saving image files.

[13]:
```
cv2.imwrite("my_new_picture.jpg",img)
#Keep in mind, the above stored the BGR version of the image.
```

[13] : True

1.1.7 Opening Image Files with OpenCV

[54]:
```
img = cv2.imread("Images/Puppy.jpg",0)
# Show the image with OpenCV
cv2.imshow("window_name",img)
# Wait for something on keyboard to be pressed to close window.
cv2.waitKey(10000)
```

[54]: -1

1.1.8 Drawing shapes on Images

[1]:
```
import cv2
import numpy as np

# Create a blank image with white background
width, height = 640, 480  # specify your desired width and height
blank_image = 255 * np.ones(shape=[height, width, 3], dtype=np.uint8)

# 1. On top left corner
top_left_corner = (0, 0)
bottom_right_corner = (100, 100)  # specify the width and height of the␣
↪rectangle
cv2.rectangle(blank_image, top_left_corner, bottom_right_corner, (0, 0, 255),␣
↪-1) # -1 fills the rectangle

# 2. On top right corner
top_right_corner = (width - 100, 0)
bottom_left_corner = (width, 100)
cv2.rectangle(blank_image, top_right_corner, bottom_left_corner, (0, 255, 0),␣
↪-1)

# 3. On bottom left corner
bottom_left_corner = (0, height - 100)
top_right_corner = (100, height)
cv2.rectangle(blank_image, bottom_left_corner, top_right_corner, (255, 0, 0),␣
↪-1)

# 4. On bottom right corner
bottom_right_corner = (width, height)
top_left_corner = (width - 100, height - 100)
cv2.rectangle(blank_image, bottom_right_corner, top_left_corner, (255, 255, 0),␣
↪-1)

# 5. On center
rect_width, rect_height = 150, 150
center_x, center_y = width // 2, height // 2
top_left_corner = (center_x - rect_width // 2, center_y - rect_height // 2)
bottom_right_corner = (center_x + rect_width // 2, center_y + rect_height // 2)
cv2.rectangle(blank_image, top_left_corner, bottom_right_corner, (0, 255, 255),␣
↪-1)

# 6. On arbitrary location
arbitrary_location = (300, 200)  # specify the top-left corner coordinates
rect_width, rect_height = 120, 200  # specify the width and height of the␣
↪rectangle
top_left_corner = arbitrary_location
```

```
bottom_right_corner = (arbitrary_location[0] + rect_width,
↪arbitrary_location[1] + rect_height)
cv2.rectangle(blank_image, top_left_corner, bottom_right_corner, (255, 0, 255),
↪-1)

# Display the image with rectangles
cv2.imshow("Image with Rectangles", blank_image)
cv2.waitKey(10000)
cv2.destroyAllWindows()
```

[ ]:
```
### Draw a circle of arbitrary size on this blank image and fill this circle

import cv2
import numpy as np

# Create a blank image with white background
width, height = 640, 480  # specify your desired width and height
blank_image = 255 * np.ones(shape=[height, width, 3], dtype=np.uint8)

# Center and radius of the circle
center_coordinates = (width // 2, height // 2)  # center of the image
radius = 100  # specify the radius of the circle

# Draw a filled circle
color = (0, 255, 0)  # specify the color of the circle (here, green in BGR
↪format)
thickness = -1  # specify thickness as -1 to fill the circle
cv2.circle(blank_image, center_coordinates, radius, color, thickness)

# Display the image with the filled circle
cv2.imshow("Image with Filled Circle", blank_image)
cv2.waitKey(10000)
cv2.destroyAllWindows()
```

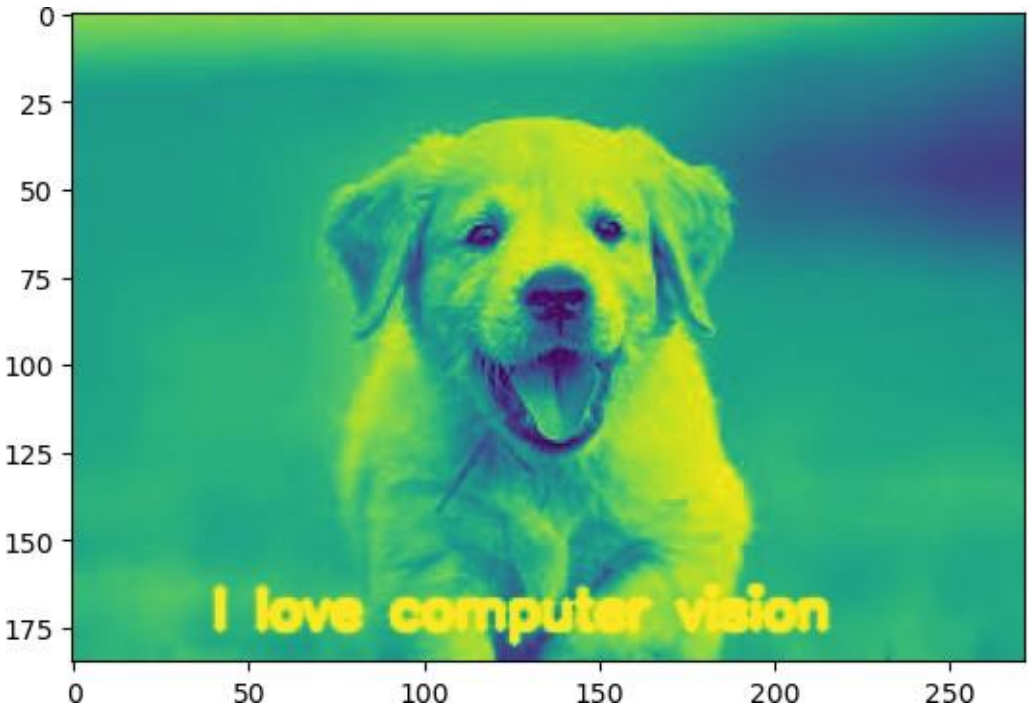
[2]:
```
### Draw a diagonal blue line with thickness of 5 px

import cv2
import numpy as np

# Create a blank image with white background
width, height = 640, 480  # specify your desired width and height
blank_image = 255 * np.ones(shape=[height, width, 3], dtype=np.uint8)

# Start and end points of the diagonal line
start_point = (50, 50)  # specify the start point (x, y)
end_point = (width - 50, height - 50)  # specify the end point (x, y)
```

```
# Color of the diagonal line (here, blue in BGR format)
color = (255, 0, 0)

# Thickness of the diagonal line (here, 5 pixels)
thickness = 5

# Draw a diagonal line on the blank image
cv2.line(blank_image, start_point, end_point, color, thickness)

# Display the image with the diagonal line
cv2.imshow('Image with Diagonal Line', blank_image)
cv2.waitKey(5000)
cv2.destroyAllWindows()
```

1.1.9 Drawing text on images

[13]:
```
## Write "I love computer vision" at the bottom of your image.

#  cv2.putText
image = cv2.imread('Puppy.jpeg',0)
font = cv2.FONT_HERSHEY_SIMPLEX
org = (40, 175)
fontScale = 0.5
color = (255, 0, 0)
thickness = 2

# Using cv2.putText() method
image = cv2.putText(image, 'I love computer vision', org, font,
                    fontScale, color, thickness, cv2.LINE_AA)
plt.imshow(image)
```

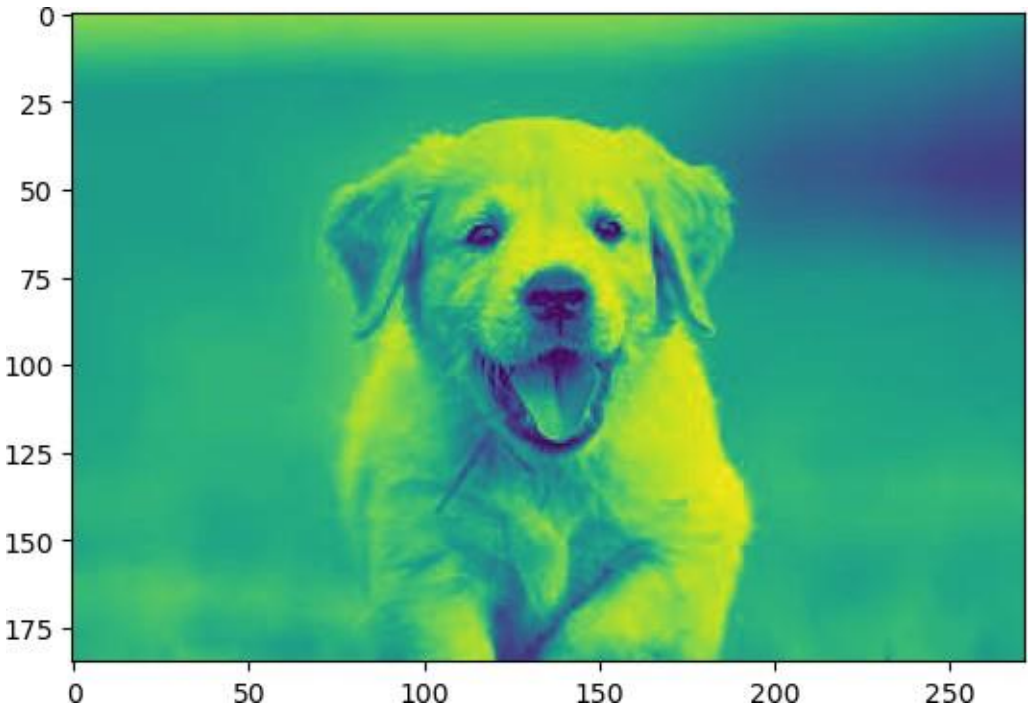
[13]: <matplotlib.image.AxesImage at 0x15559cac0>



1.1.10 Color Spaces

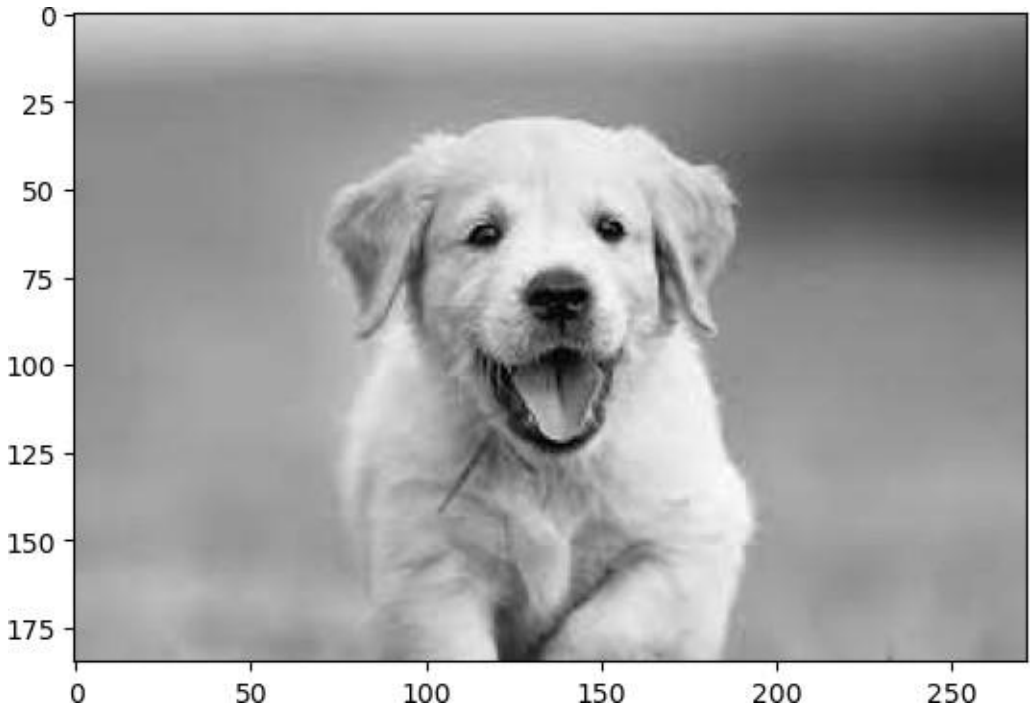
[11]:
```
img = cv2.imread('Puppy.jpeg',0)
plt.imshow(img) # It will display in opencv default colorspace that is BGR
```

[11]: <matplotlib.image.AxesImage at 0x16c9a5f00>



[14]:
```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

[14] : <matplotlib.image.AxesImage at 0x1555cb460>



2 Blending and Pasting Images

For some computer vision systems, we'll want to be able to paste our own image on top of an already existing image or video. We may also want to blend images, maybe we want to have a "highlight" effect instead of just a solid box or empty rectangle.

Let's explore what is commonly known as **Arithmetic Image Operations** with OpenCV. These are referred to as Arithmetic Operations because OpenCV is simply performing some common math with the pixels for the final effect.

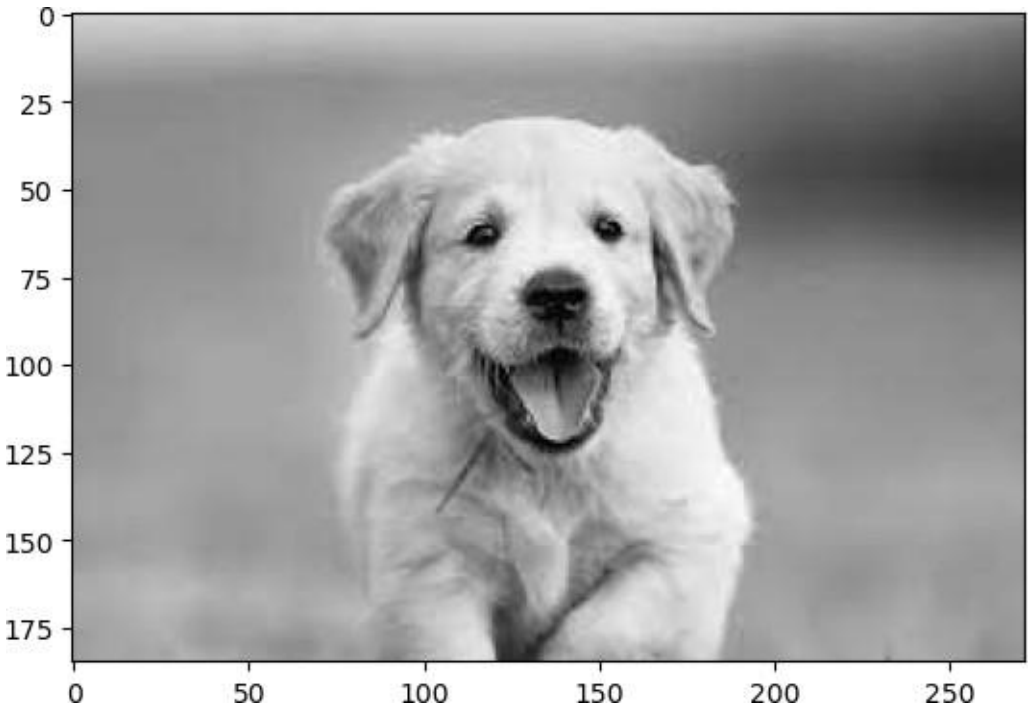
2.1 Blending Images

Blending Images is actually quite simple, let's look at a simple example.

[14]:
```
# Two images
img1 = cv2.imread('Puppy.jpeg',0)
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
# img2 = cv2.imread('watermark_no_copy.png')
img2 = cv2.cvtColor(blank_image, cv2.COLOR_BGR2RGB)
```

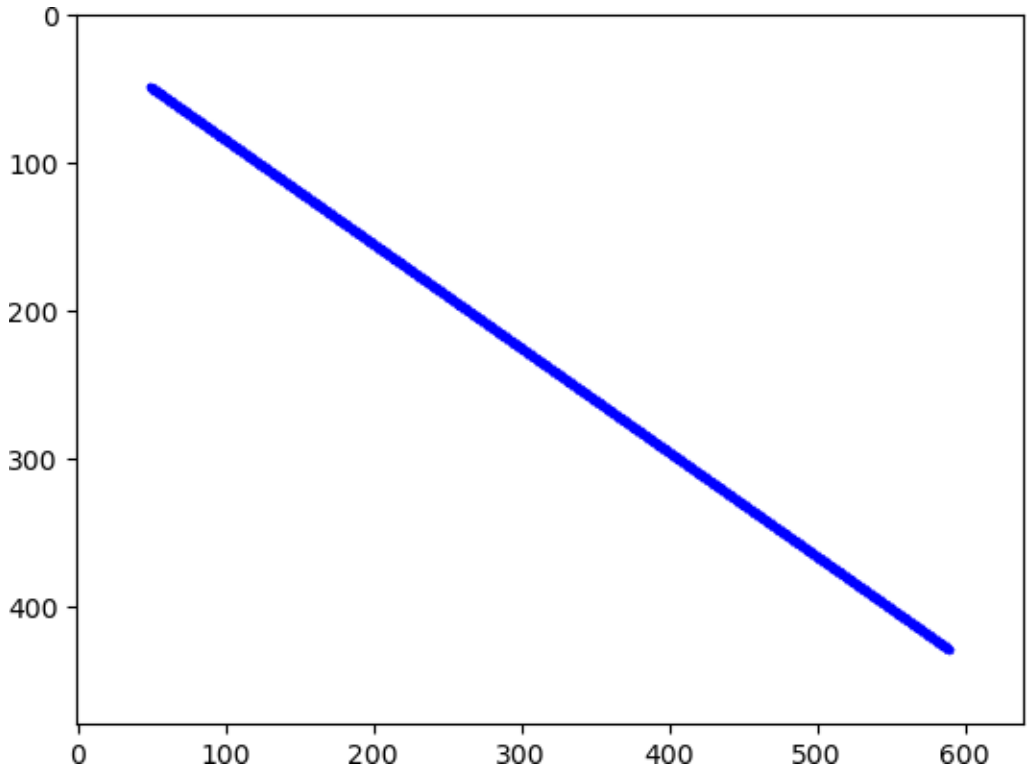
[15]:
```
plt.imshow(img1)
```

[15] : <matplotlib.image.AxesImage at 0x16ca13eb0>



[16]:
```
plt.imshow(img2)
```

[16] : <matplotlib.image.AxesImage at 0x16ca82b60>



[17]:
```
img1.shape
```

[17]: (185, 272, 3)

[18]:
```
img2.shape
```

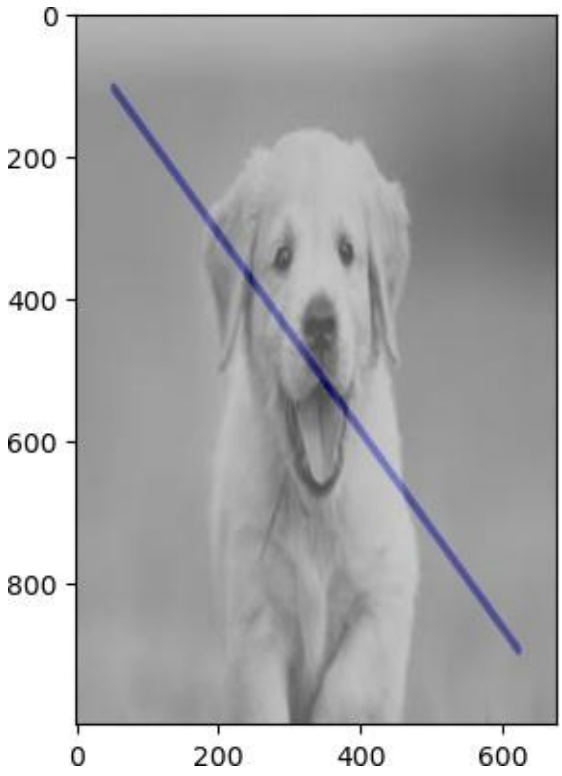
[18]: (480, 640, 3)

Resizing the Images so that both images are of same size.

[19]:
```
# BLENDING IMAGES OF SAME SIZE

img1 =cv2.resize(img1, (675,1000))
img2 =cv2.resize(img2, (675,1000))
```

[20]:
```
blended = cv2.addWeighted(src1=img1,alpha=0.5,src2=img2,beta=0.3,gamma=0)
plt.imshow(blended)
# this function only works when the images are of same size.
```

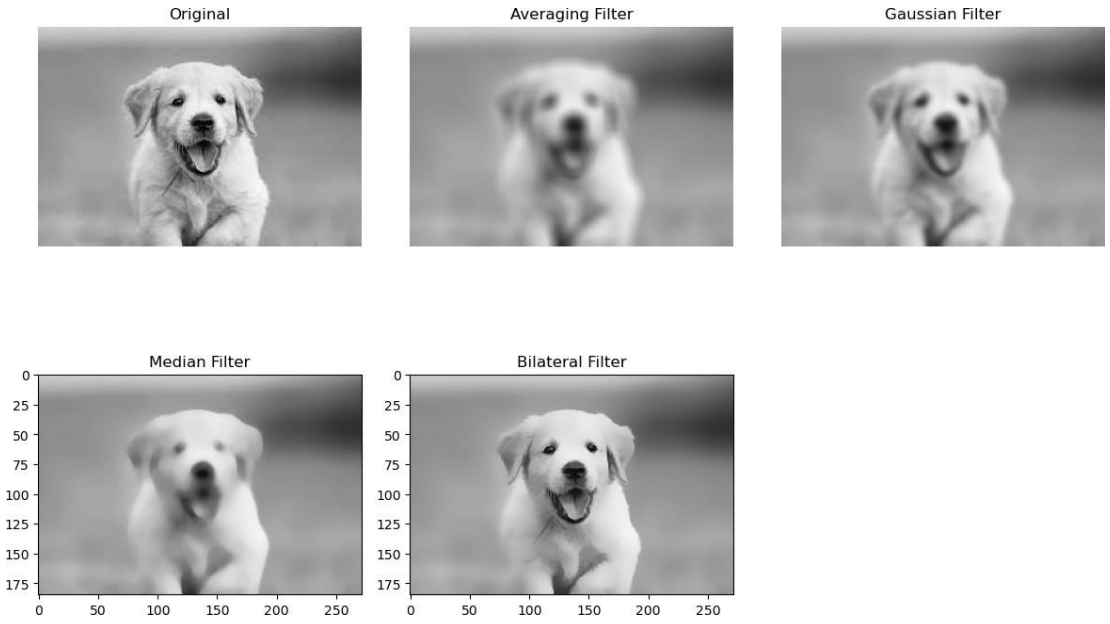
[20]: <matplotlib.image.AxesImage at 0x16cacce80>



2.1.1 Blurring the images with a low pass filters

[21]:
```
image = cv2.imread('Puppy.jpeg',0) # Replace with your image path
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Apply Averaging filter

averaged = cv2.blur(image, (11, 11))

# Apply Gaussian Filter

gaussian = cv2.GaussianBlur(image, (11, 11), 0)

# Apply Median Filter

median = cv2.medianBlur(image, 11)

# Apply Bilateral FIlter
bilateral = cv2.bilateralFilter(image, 15, 75, 75)

# Display the original and filtered images
plt.figure(figsize=(12, 8))
```

```
plt.subplot(2, 3, 1)
plt.title("Original")
plt.imshow(image)
plt.axis('off')

plt.subplot(2, 3, 2)
plt.title("Averaging Filter")
plt.imshow(averaged)
plt.axis('off')

plt.subplot(2, 3, 3)
plt.title("Gaussian Filter")
plt.imshow(gaussian)
plt.axis('off')

plt.subplot(2, 3, 4)
plt.title("Median Filter")
plt.imshow(median)


plt.subplot(2, 3, 5)
plt.title('Bilateral Filter')
plt.imshow(bilateral)

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



[ ]:

lab3-video-processing

November 19, 2023

1 Video Processing

OpenCV can automatically connect to your laptop's built in camera or your USB camera if you've installed that specific USB camera drivers.

1.0.1 Reading and Displaying a Video

[6]:
```
import cv2

# Connects to your computer's default camera
cap = cv2.VideoCapture(0)

# Automatically grab width and height from video feed
# (returns float which we need to convert to integer for later on!)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while True:

    # Capture frame-by-frame
    ret, frame = cap.read()
    # print(ret)

    # Our operations on the frame come here
    # gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',frame)

    # This command let's us quit with the "q" button on a keyboard.
    # Simply pressing X on the window won't work!
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

# When everything done, release the capture and destroy the windows
cap.release()
cv2.destroyAllWindows()
```

1.0.2 Writing a Video Stream to File

FourCC is a 4-byte code used to specify the video codec. The list of available codes can be found in fourcc.org. It is platform dependent.

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html#saving- a-video

[7]:
```
import cv2

cap = cv2.VideoCapture(0)

# Automatically grab width and height from video feed
# (returns float which we need to convert to integer for later on!)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))


# MACOS AND LINUX: *'XVID' (MacOS users may want to try VIDX as well just in␣
 ↪case)
# WINDOWS *'DIVX'
# the next item is the frame rate i.e., no. of frames per seconds. FOr USB␣
 ↪cameras it is mostly between 20 to 30.

writer = cv2.VideoWriter("classroom.mp4", cv2.VideoWriter_fourcc(*"DIVX"), 25,␣
 ↪(width, height))

## This loop keeps recording until you hit Q or escape the window
## You may want to instead use some sort of timer, like from time import sleep␣
 ↪and then just record for 5 seconds.

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Write the video
    writer.write(frame)

    # Display the resulting frame
    cv2.imshow("frame",frame)

    # This command let's us quit with the "q" button on a keyboard.
    # Simply pressing X on the window won't work!
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break


cap.release()
```

```
writer.release()
cv2.destroyAllWindows()
```

1.0.3 With Video Files

[8]:
```
import cv2
import time
# Same command function as streaming, its just now we pass in the file path, nice!
cap = cv2.VideoCapture("myfirstvideo.mp4")

# FRAMES PER SECOND FOR VIDEO
fps = 25

# Always a good idea to check if the video was acutally there
# If you get an error at thsi step, triple check your file path!!
if cap.isOpened()== False:
    print("Error opening the video file. Please double check your file path for typos. Or move the movie file to the same location as this script/notebook")


# While the video is opened
while cap.isOpened():

    # Read the video file.
    ret, frame = cap.read()

    # If we got frames, show them.
    if ret == True:

        cv2.imshow("frame",frame)

        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord("q"):
            break

    # Or automatically break this whole loop if the video is over.
    else:
        break

cap.release()
# Closes all the frames
cv2.destroyAllWindows()
```

2 Assignment 1:

1. Read a live stream from the camera on your computer or if you do not have camera, read a video file.
2. Draw a Blue circle in the center and a green rectangle at the top left of video.
3. Display the frame number on every frame in a video.

[9]:
```
#1
import cv2
import time
# Same command function as streaming, its just now we pass in the file path, nice!
cap = cv2.VideoCapture('myfirstvideo.mp4')

# FRAMES PER SECOND FOR VIDEO
fps = 25

# Always a good idea to check if the video was acutally there
# If you get an error at thsi step, triple check your file path!!
if cap.isOpened()== False:
    print("Error opening the video file. Please double check your file path for typos. Or move the movie file to the same location as this script/notebook")


# While the video is opened
while cap.isOpened():

    # Read the video file.
    ret, frame = cap.read()

    # If we got frames, show them.
    if ret == True:

        cv2.imshow('frame',frame)

        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Or automatically break this whole loop if the video is over.
    else:
        break

cap.release()
# Closes all the frames
cv2.destroyAllWindows()
```

[20]:
```
#2
import cv2

cap = cv2.VideoCapture(0)

# Automatically grab width and height from video feed
# (returns float which we need to convert to integer for later on!)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
# While the video is opened
while True:

    # Read the video file.
    ret, frame = cap.read()

    # If we got frames, show them.
    if ret == True:

        center_coordinates = (320, 240)
        radius = 100
        color = (255, 0, 0)
        thickness = 2
        image = cv2.circle(frame, center_coordinates, radius, color, thickness)

        x1, y1, x2, y2 = 50, 50, 250, 150
        cv2.rectangle(image, (x1, y1), (x2, y2), (0,255, 0), 2)

        cv2.imshow("frame",image)

        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Or automatically break this whole loop if the video is over.
    else:
        break

cap.release()
# Closes all the frames
cv2.destroyAllWindows()
```

[27]:
```
#3
cap = cv2.VideoCapture(0)

# Automatically grab width and height from video feed
# (returns float which we need to convert to integer for later on!)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
```

```
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
# While the video is opened
counter=1
while True:

    # Read the video file.
    ret, frame = cap.read()

    # If we got frames, show them.
    if ret == True:

        font = cv2.FONT_HERSHEY_SIMPLEX
        org = (50, 50)
        fontScale = 1
        color = (255, 0, 0)
        thickness = 2
        image = cv2.putText(frame, str(counter), org, font, fontScale, color,
↪thickness, cv2.LINE_AA)
        counter+=1

        cv2.imshow('frame',image)

        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Or automatically break this whole loop if the video is over.
    else:
        break

cap.release()
# Closes all the frames
cv2.destroyAllWindows()
```

3 Assignment 2:

1. Read a live stream from the camera on your computer or if you do not have camera, read a video file.
2. Perform the background subtraction using simple frame differencing.
3. Convert every frame to binary using cv2.threshold()
4. Apply morphological operators to remove noise and fill holes.
5. Save and display the final background subtracted video.

[ ]:
```
#1
import cv2
import time
```

```
# Same command function as streaming, its just now we pass in the file path, nice!
cap = cv2.VideoCapture('myfirstvideo.mp4')

# FRAMES PER SECOND FOR VIDEO
fps = 25

# Always a good idea to check if the video was acutally there
# If you get an error at thsi step, triple check your file path!!
if cap.isOpened()== False:
    print("Error opening the video file. Please double check your file path for typos. Or move the movie file to the same location as this script/notebook")


# While the video is opened
while cap.isOpened():

    # Read the video file.
    ret, frame = cap.read()

    # If we got frames, show them.
    if ret == True:

        cv2.imshow('frame',frame)

        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Or automatically break this whole loop if the video is over.
    else:
        break

cap.release()
# Closes all the frames
cv2.destroyAllWindows()
```

[41]:
```
#2
import numpy as np
cap = cv2.VideoCapture(0)

# Automatically grab width and height from video feed
# (returns float which we need to convert to integer for later on!)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
# While the video is opened
```

```
prevFrame=np.zeros([480,640],dtype=float)
while True:

    # Read the video file.
    ret, frame = cap.read()
    # print(frame.type)

    # If we got frames, show them.
    if ret == True:

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cv2.imshow('frame',frame-prevFrame)
        prevFrame=frame

        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Or automatically break this whole loop if the video is over.
    else:
        break

cap.release()
# Closes all the frames
cv2.destroyAllWindows()
```

[46]:
```
#3
import numpy as np
cap = cv2.VideoCapture(0)

# Automatically grab width and height from video feed
# (returns float which we need to convert to integer for later on!)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
# While the video is opened

prevFrame=np.zeros([480,640],dtype=float)
while True:

    # Read the video file.
    ret, frame = cap.read()
    # print(frame.type)

    # If we got frames, show them.
    if ret == True:

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
        ret,thres=cv2.threshold(frame-prevFrame,0,255,cv2.THRESH_BINARY_INV)

        cv2.imshow('frame',thres)
        prevFrame=frame

        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Or automatically break this whole loop if the video is over.
    else:
        break

cap.release()
# Closes all the frames
cv2.destroyAllWindows()
```

[47]:
```
#4
import numpy as np
cap = cv2.VideoCapture(0)

# Automatically grab width and height from video feed
# (returns float which we need to convert to integer for later on!)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
# While the video is opened

kernel=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))

prevFrame=np.zeros([480,640],dtype=float)
while True:

    # Read the video file.
    ret, frame = cap.read()
    # print(frame.type)

    # If we got frames, show them.
    if ret == True:

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        ret,thres=cv2.threshold(frame-prevFrame,0,255,cv2.THRESH_BINARY_INV)

        #Appling morphological operators to remove noise and fill holes.
        binary_mask=cv2.morphologyEx(thres,cv2.MORPH_OPEN,kernel)
        binary_mask=cv2.morphologyEx(thres,cv2.MORPH_CLOSE,kernel)
```

```
        cv2.imshow('frame',binary_mask)
        prevFrame=frame

        # Press q to quit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Or automatically break this whole loop if the video is over.
    else:
        break

cap.release()
# Closes all the frames
cv2.destroyAllWindows()
```

[ ]:

4 Assignment 3:

Perform Background Subtraction using the following in openCV: 1. Mixture of Gaussian (MOG) 2. BackgroundSUbtractorGMG

[4]:
```
import cv2

cap = cv2.VideoCapture(0)

# Create MOG2 object
fgbg = cv2.createBackgroundSubtractorMOG2()

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Apply background subtraction
    frame=cv2.resize(frame,(640,480))
    fgmask = fgbg.apply(frame)

    cv2.imshow('Original', frame)
    cv2.imshow('Foreground', fgmask)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
```

```
cv2.destroyAllWindows()
```

[4]:
```
-1
```

[3]:
```
#2
import cv2

cap = cv2.VideoCapture(0)

# Create GMG object
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
fgbg = cv2.bgsegm.createBackgroundSubtractorGMG(initializationFrames=12)

while True:
    ret, frame = cap.read()
    if not ret:
        break
    frame=cv2.resize(frame,(640,480))
    # Apply background subtraction
    fgmask = fgbg.apply(frame)
    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel)

    cv2.imshow('Original', frame)
    cv2.imshow('Foreground', fgmask)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[3], line 8
      6 # Create GMG object
      7 kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
----> 8 fgbg = cv2.bgsegm.createBackgroundSubtractorGMG(initializationFrames=12)
     10 while True:
     11     ret, frame = cap.read()

AttributeError: module 'cv2' has no attribute 'bgsegm'
```

[ ]:

ocnxbthie

November 19, 2023

1 Object Detection

[2]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

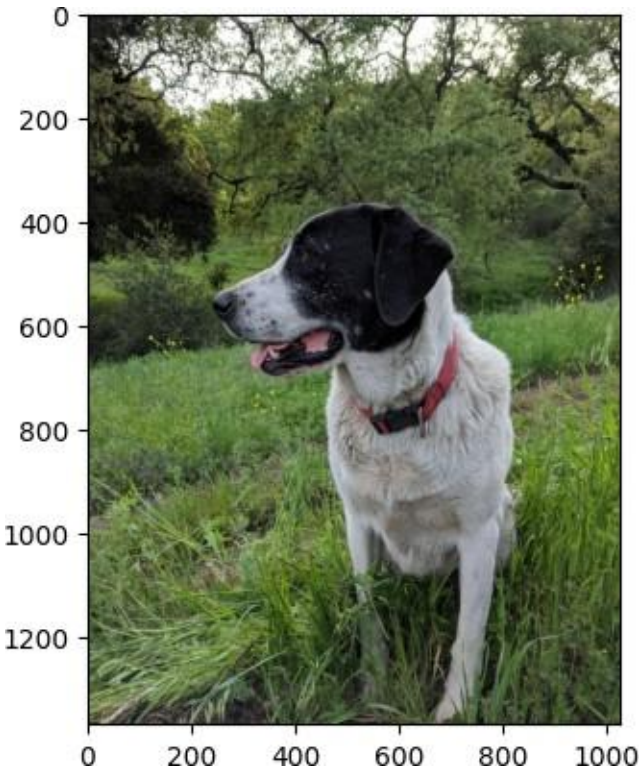
1.1 Method-1: Template Matching

1. Template matching is the simplest form of object detection.
2. It simply scans a larger image for a provided template by sliding the template target image across the larger image.
3. Comparision are based on correlation based metric.

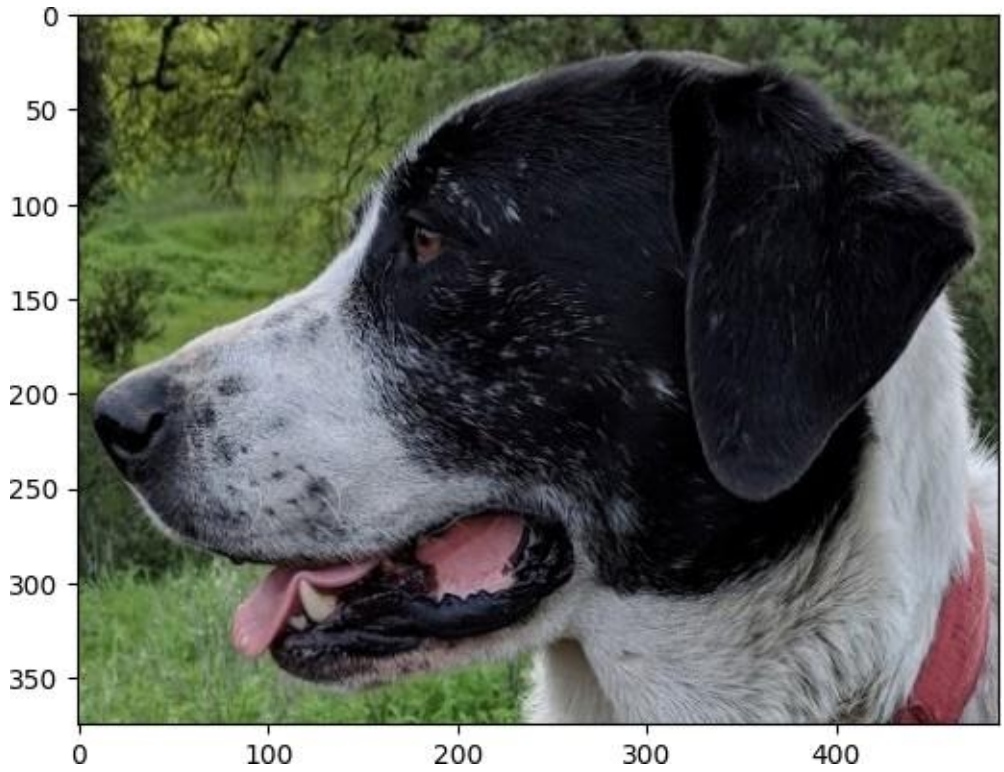
https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

[2]:
```
# Read a full image.
full = cv2.imread('sammy.jpg')
full = cv2.cvtColor(full, cv2.COLOR_BGR2RGB)
plt.imshow(full)
full.shape
```

[2]: (1367, 1025, 3)



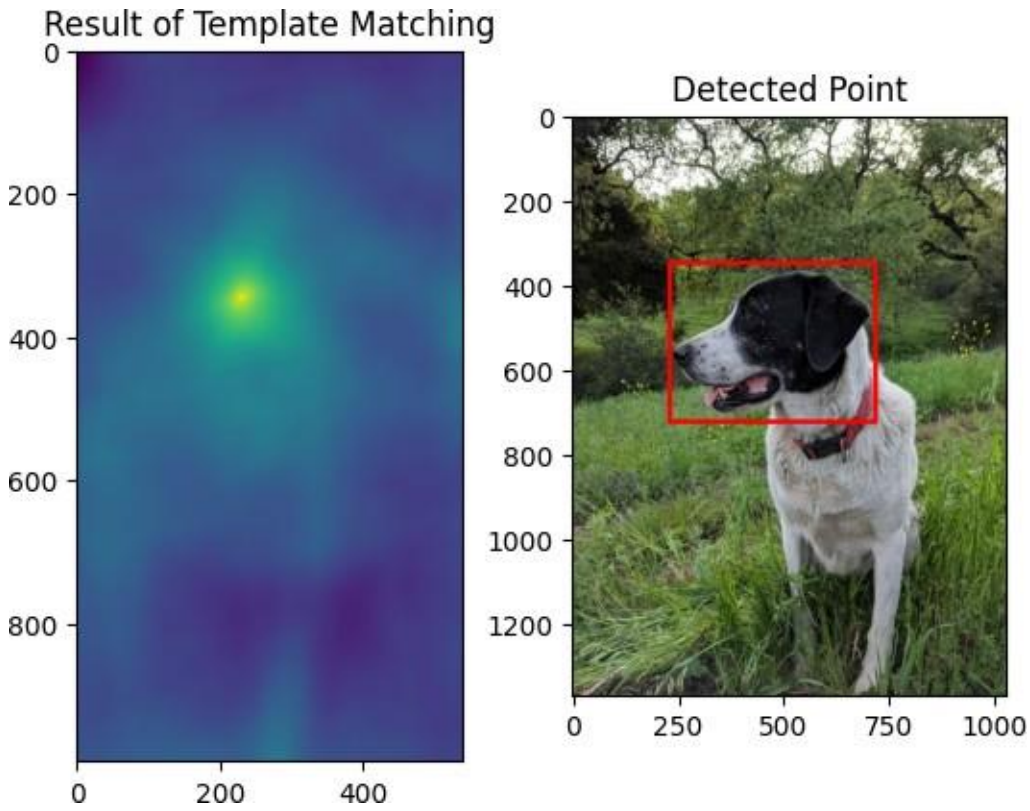
[3]:
```
# Read the template image
face= cv2.imread('sammy_face.jpg')
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
plt.imshow(face)

height, width,channels = face.shape
```



[4]:
```
# Create a copy of the image
full_copy = full.copy()

# Apply template Matching with the method
res = cv2.matchTemplate(full_copy, face, cv2.TM_CCOEFF)

# Grab the Max and Min values, plus their locations
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

# Set up drawing of Rectangle
top_left = max_loc

# Assign the Bottom Right of the rectangle
bottom_right = (top_left[0] + width, top_left[1] + height)

# Draw the Red Rectangle
cv2.rectangle(full_copy, top_left, bottom_right, 255, 10)

# Plot the Images
plt.subplot(121)
plt.imshow(res)
plt.title("Result of Template Matching")
```

```
plt.subplot(122)
plt.imshow(full_copy)
plt.title('Detected Point')
```

[4]: Text(0.5, 1.0, 'Detected Point')



1.2 Task -1

Perform the template matching with the following metrics and show the results.

cv2.TM_CCOEFF_NORMED

cv2.TM_CCORR

cv2.TM_CCORR_NORMED

cv2.TM_SQDIFF

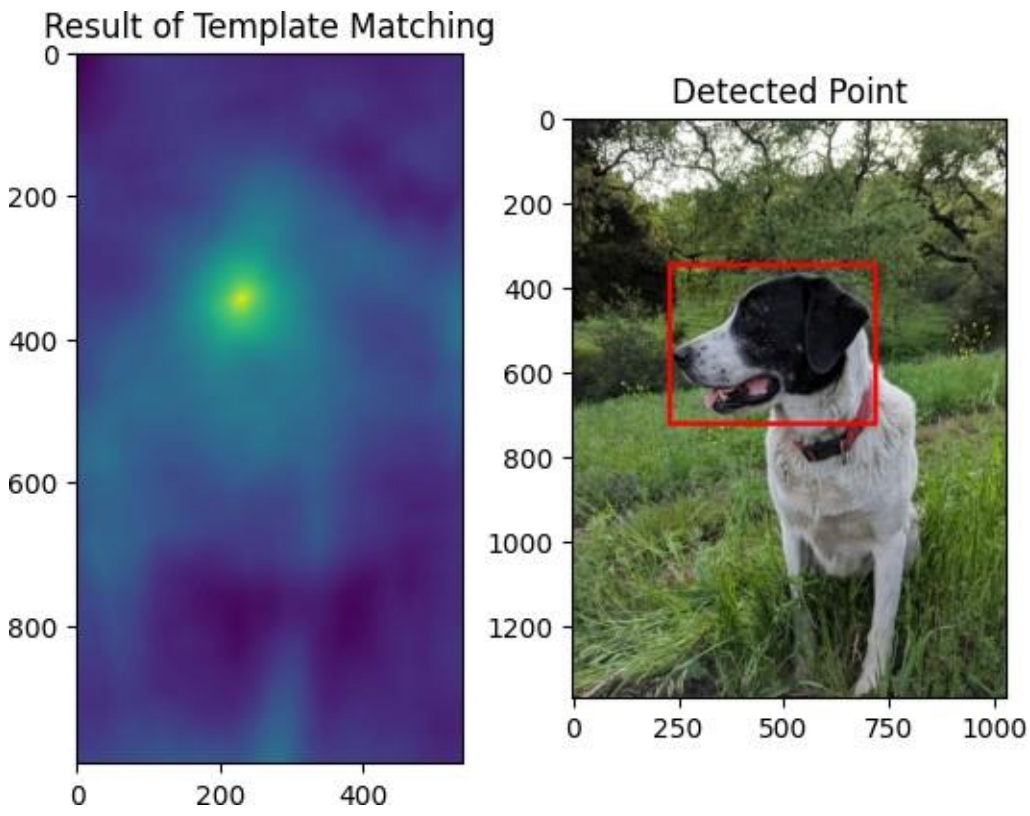
cv2.TM_SQDIFF_NORMED

[5]:
```
full_copy = full.copy()

res2 = cv2.matchTemplate(full_copy, face, cv2.TM_CCOEFF_NORMED)
```

```
res3 = cv2.matchTemplate(full_copy, face, cv2.TM_CCORR)
res4 = cv2.matchTemplate(full_copy, face, cv2.TM_CCORR_NORMED)
res5 = cv2.matchTemplate(full_copy, face, cv2.TM_SQDIFF)
res6 = cv2.matchTemplate(full_copy, face, cv2.TM_SQDIFF_NORMED)

# Grab the Max and Min values, plus their locations
min_val2, max_val2, min_loc2, max_loc2 = cv2.minMaxLoc(res2)
min_val3, max_val3, min_loc3, max_loc3 = cv2.minMaxLoc(res3)
min_val4, max_val4, min_loc4, max_loc4 = cv2.minMaxLoc(res4)
min_val5, max_val5, min_loc5, max_loc5 = cv2.minMaxLoc(res5)
min_val6, max_val6, min_loc6, max_loc6 = cv2.minMaxLoc(res6)

# Set up drawing of Rectangle
top_left2 = max_loc2
top_left3 = max_loc3
top_left4 = max_loc4
top_left5 = max_loc5
top_left6 = max_loc6

# Assign the Bottom Right of the rectangle
bottom_right2 = (top_left2[0] + width, top_left2[1] + height)
bottom_right3 = (top_left3[0] + width, top_left3[1] + height)
bottom_right4 = (top_left4[0] + width, top_left4[1] + height)
bottom_right5 = (top_left5[0] + width, top_left5[1] + height)
bottom_right6 = (top_left6[0] + width, top_left6[1] + height)

# Draw the Red Rectangle
```

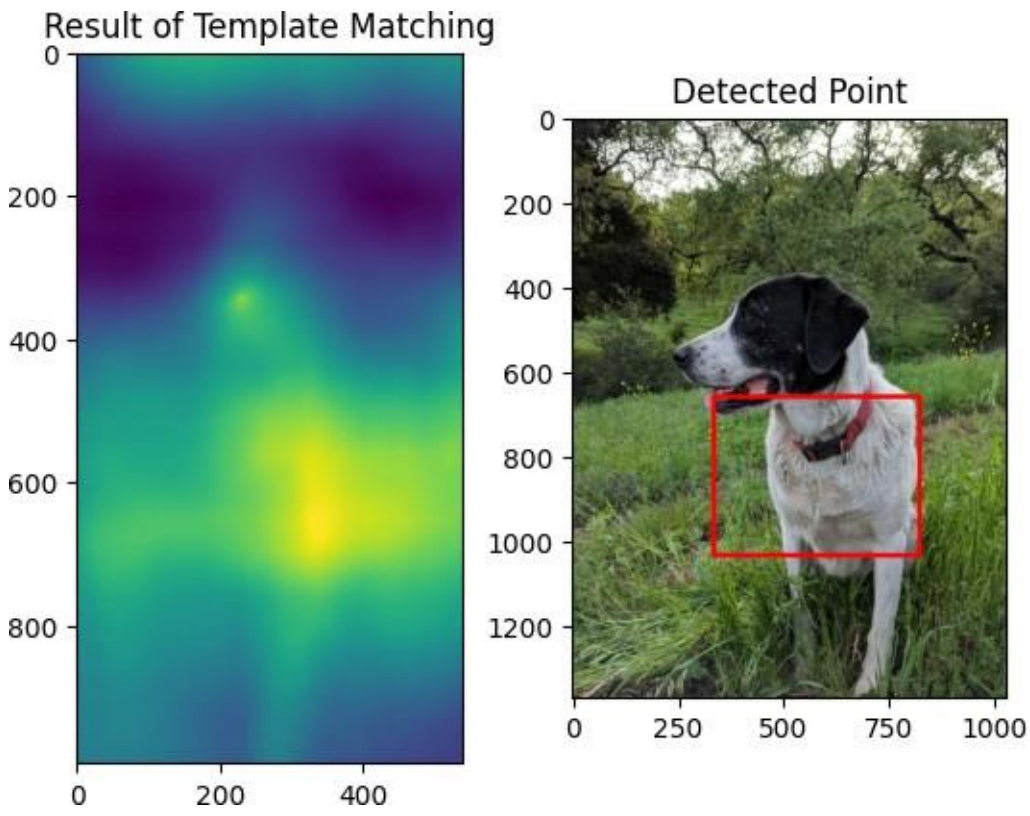
[45]:
```
full_copy = full.copy()
cv2.rectangle(full_copy, top_left2, bottom_right2, 255, 10)
# Plot the Images
plt.subplot(121)
plt.imshow(res2)
plt.title("Result of Template Matching")
plt.subplot(122)
plt.imshow(full_copy)
plt.title("Detected Point")
```

[45] : Text(0.5, 1.0, 'Detected Point')



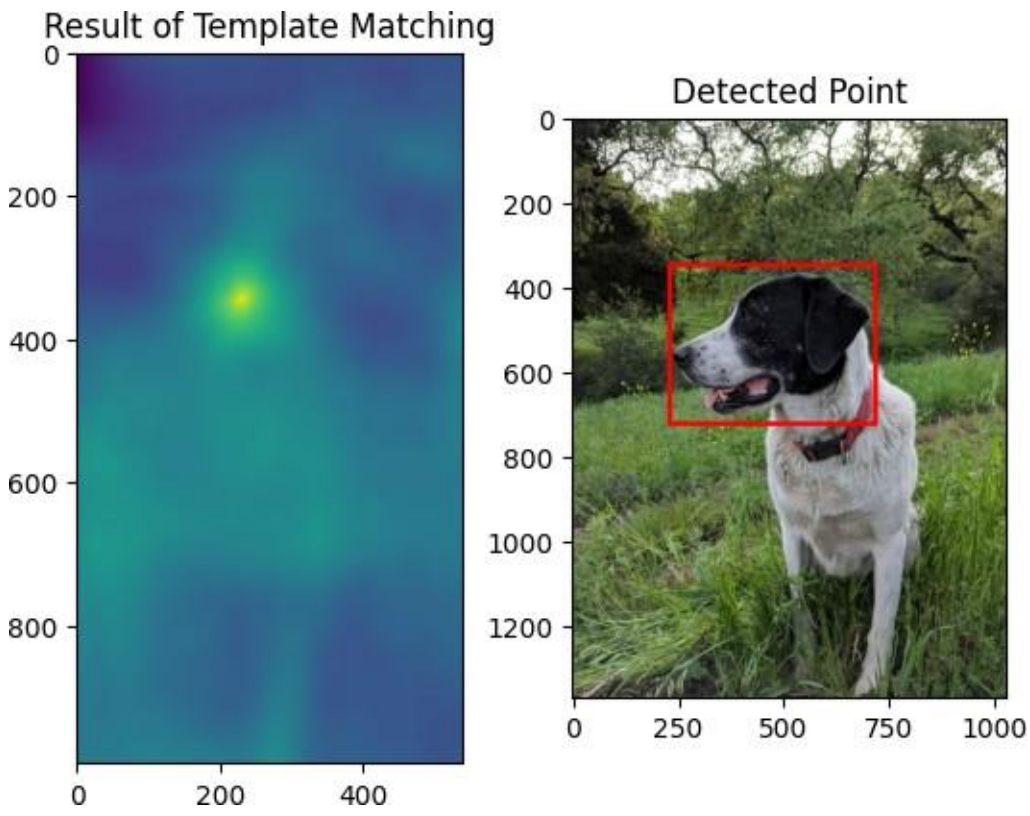
[46]:
```
full_copy = full.copy()
cv2.rectangle(full_copy, top_left3, bottom_right3, 255, 10)
# Plot the Images
plt.subplot(121)
plt.imshow(res3)
plt.title("Result of Template Matching")
plt.subplot(122)
plt.imshow(full_copy)
plt.title("Detected Point")
```

[46] : Text(0.5, 1.0, 'Detected Point')



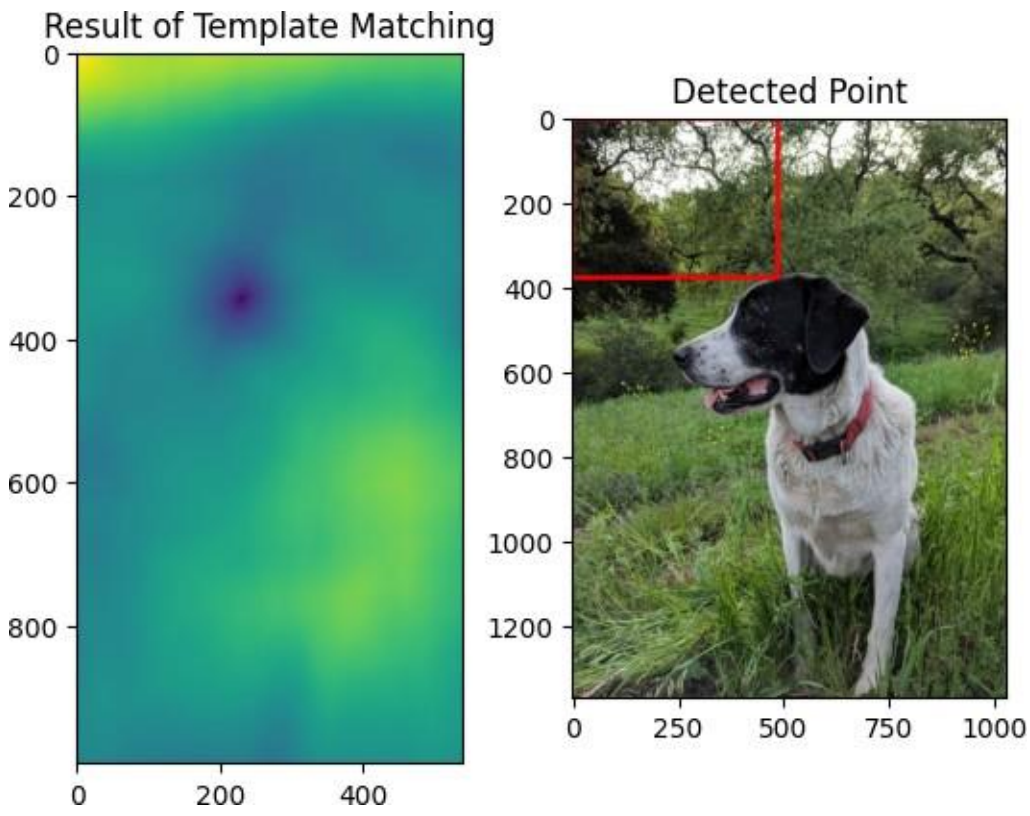
[47]:
```
full_copy = full.copy()
cv2.rectangle(full_copy, top_left4, bottom_right4, 255, 10)
# Plot the Images
plt.subplot(121)
plt.imshow(res4)
plt.title("Result of Template Matching")
plt.subplot(122)
plt.imshow(full_copy)
plt.title("Detected Point")
```

[47] : Text(0.5, 1.0, 'Detected Point')



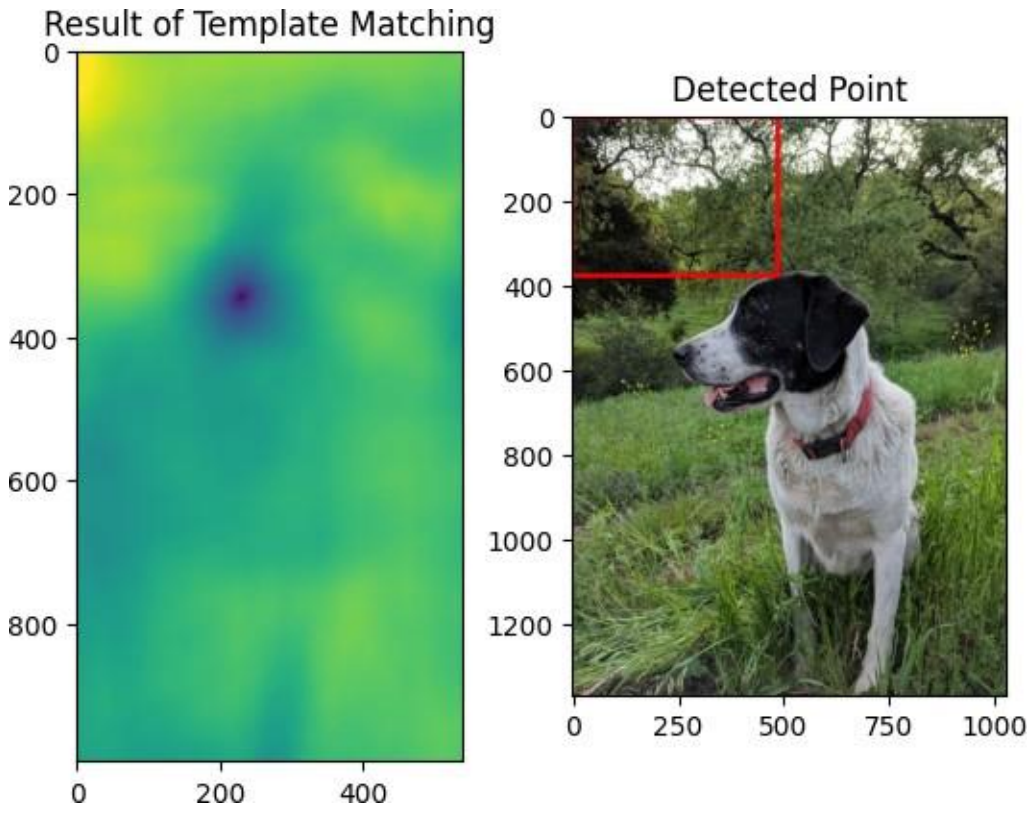
[48]:
```
full_copy = full.copy()
cv2.rectangle(full_copy, top_left5, bottom_right5, 255, 10)
# Plot the Images
plt.subplot(121)
plt.imshow(res5)
plt.title("Result of Template Matching")
plt.subplot(122)
plt.imshow(full_copy)
plt.title("Detected Point")
```

[48] : Text(0.5, 1.0, 'Detected Point')



[49]:
```
full_copy = full.copy()
cv2.rectangle(full_copy, top_left6, bottom_right6, 255, 10)
# Plot the Images
plt.subplot(121)
plt.imshow(res6)
plt.title("Result of Template Matching")
plt.subplot(122)
plt.imshow(full_copy)
plt.title("Detected Point")
```

[49] : Text(0.5, 1.0, 'Detected Point')



2 Method 2: Corner Detection

2.0.1 A corner can be interpreted as the junction of two edges, where an edge is a sudden change in image brightness.

2.0.2 Task-2:

Apply the following corner detection algorithm on your chess image and display the corners in red color on the given image.

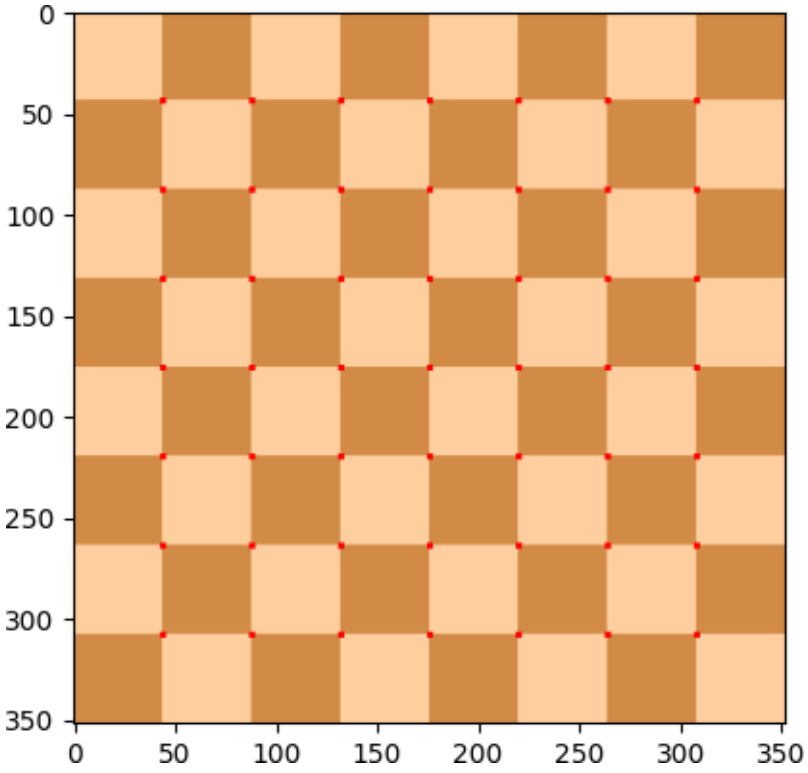
1. Harris corner detection: cv2.cornerHarris()

2. Shi-Tomasi: cv2.goodFeaturesToTrack()

[7]:
```
import cv2
import numpy as np

# Load the image
image = cv2.imread('flat_chessboard.png')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 1. Harris Corner Detection
```

```
harris_dst = cv2.cornerHarris(gray, 2, 3, 0.04)
harris_image = image.copy()
harris_image[harris_dst > 0.01 * harris_dst.max()] = [0, 0, 255]
rgb_img = cv2.cvtColor(harris_image, cv2.COLOR_BGR2RGB)

# Display results
plt.imshow(rgb_img)
```

[7]: <matplotlib.image.AxesImage at 0x1f25fc19d20>



3 Method 3: Edge Detection

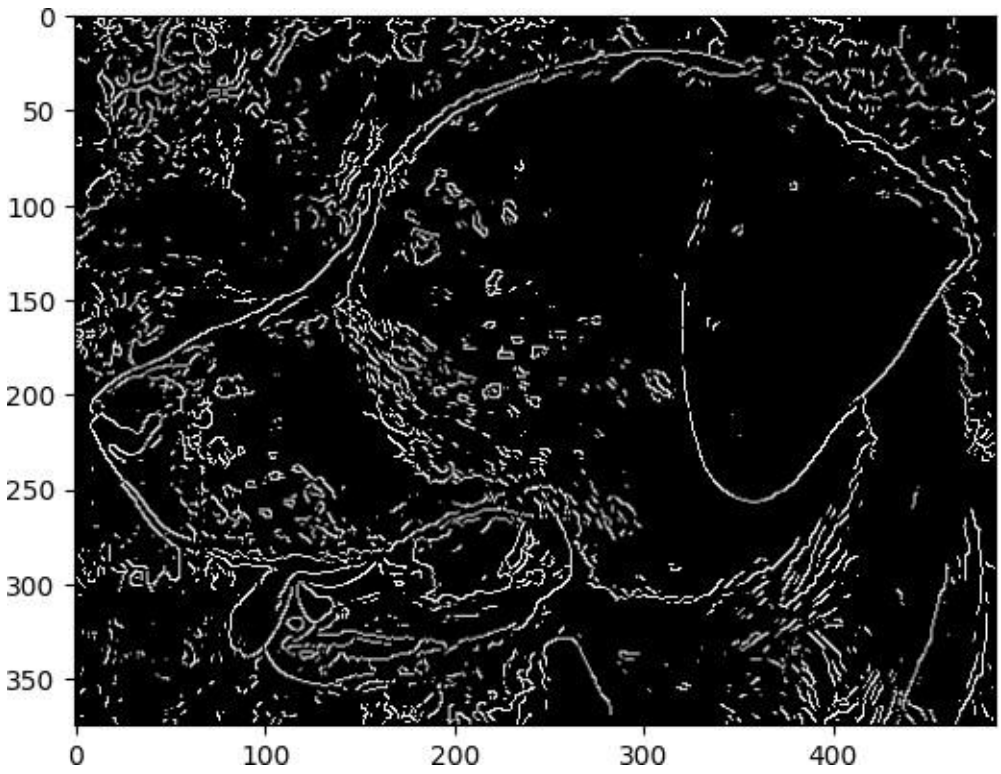
3.0.1 Task-3:

3.0.2 Apply Canny Edge Detection algorithm on the following image. Before applying the canny edge function, it is a good idea to blur/smooth the image to remove the noise.

1. cv2.blur()
2. cv2.Canny()

[10]:
```
face= cv2.imread('sammy_face.jpg')
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
blurred = cv2.GaussianBlur(face, (5, 5), 0)
edges = cv2.Canny(blurred, 100, 100)
rgb_img = cv2.cvtColor(edges, cv2.COLOR_BGR2RGB)

# Display results
plt.imshow(rgb_img)
```

[10]: <matplotlib.image.AxesImage at 0x1f25fada500>



4 Method 4: Contour Detection

4.1 External and Internal Contours

4.1.1 Task-4:

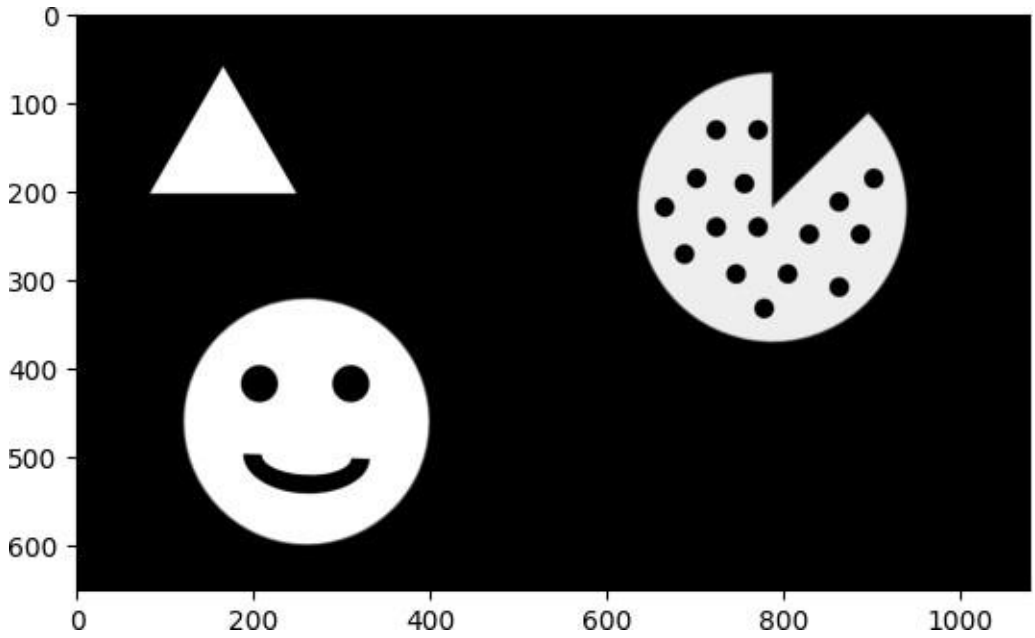
Detect the external and internal contours on the real images. findContours function will return back contours in an image, and based on the RETR method called, you can get back external, internal, or both:

- cv2.RETR_EXTERNAL:Only extracts external contours

- cv2.RETR_CCOMP: Extracts both internal and external contours organized in a two-level hierarchy
- cv2.RETR_TREE: Extracts both internal and external contours organized in a tree graph
- cv2.RETR_LIST: Extracts all contours without any internal/external relationship

[40]:
```
img = cv2.imread('internal_external.png',0)
plt.imshow(img,cmap='gray')
```

[40]: <matplotlib.image.AxesImage at 0x1da0c1ff6d0>



[41]:
```
contours, hierarchy = cv2.findContours(img, cv2.RETR_CCOMP, cv2.
↪CHAIN_APPROX_SIMPLE)
```

[52]:
```
# FInd the number of contours
# print(contours)
len(contours)
```

[52]: 22

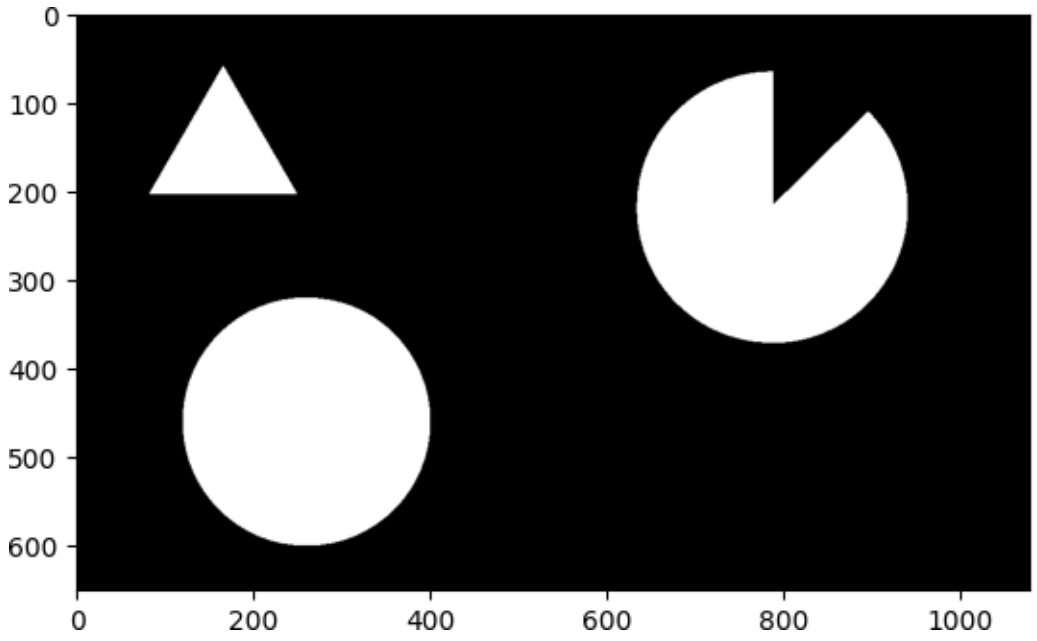
[53]:
```
hierarchy.shape
```

[53]: (1, 22, 4)

[60]:
```
# Draw External Contours

# Set up empty array
```

```
external_contours = np.zeros(img.shape)

# For every entry in contours
for i in range(len(contours)):

    # last column in the array is -1 if an external contour (no contours inside
    of it)
    if hierarchy[0][i][3] == -1:

        # We can now draw the external contours from the list of contours
        cv2.drawContours(external_contours, contours, i, 255, -1)

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

[60]: <matplotlib.image.AxesImage at 0x1da2156ab90>



Method 5: Image Segmentation

1. Image Thresholding (Manual and OTSU thresolding)

```
image = cv2.imread('K:\sem7\sammy.jpg')threshold_value = 128

 # Apply thresholding
_, binary_image = cv2.threshold(image, threshold_value, 255,cv2.THRESH_BINARY)

 # Display the thresholded images cv2.imshow('Thresholded Image
 (Manual)', binary_image)cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
image = cv2.imread('K:\sem7\sammy.jpg')

 # Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

 # Apply Otsu's thresholding
_, binary_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY
+ cv2.THRESH_OTSU)

 # Display the thresholded images cv2.imshow('Thresholded Image
 (Otsu)', binary_image)cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. Region Growing and Splitting

```
def region_growing(image, seed):
    # Create a binary mask for the segmented region
    h, w = image.shape[:2]
    segmented = np.zeros_like(image, dtype=np.uint8)

    # Parameters for region growing
    threshold = 30  # Adjust this threshold based on your image
    neighbors = [(0, 1), (1, 0), (0, -1), (-1, 0)]  # 4-connectivity

    # Seed point and initial region value
    seed_value = image[seed]

    # Create a queue for pixel traversal
    queue = [seed]

    while queue:
        current_pixel = queue.pop(0)

        for neighbor in neighbors:
            x, y = current_pixel[0] + neighbor[0], current_pixel[1] +
neighbor[1]

            # Check if the neighbor is within the image bounds
            if 0 <= x < h and 0 <= y < w:
                neighbor_value = image[x, y]

                # Check the similarity criteria (you can customize
 this)
                if abs(int(neighbor_value) - int(seed_value)) <threshold and
```

```
 # Choose a seed point (you may need to adjust this based on yourimage)
seed_point = (100, 100)

 # Apply region growing
segmented_image = region_growing(image, seed_point)

 # Display the segmented images
 cv2.imshow('Original Image', image)
cv2.imshow('Segmented Image (Region Growing)', segmented_image)cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. K- Means Clustering

```
image = cv2.imread('K:\sem7\sammy.jpg')

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

# Convert to floating-point type
pixels = np.float32(pixels)

# Define criteria and apply kmeans()
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,0.2)
k = 3   # Number of clusters (adjust as needed)
_, labels, centers = cv2.kmeans(pixels, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)
```

```
# Convert back to 8-bit values
centers = np.uint8(centers)

# Map the labels to the centers
segmented_image = centers[labels.flatten()]

# Reshape back to the original image shape
segmented_image = segmented_image.reshape(image.shape)

# Display the segmented images
cv2.imshow('Segmented Image (K-Means Clustering)', segmented_image)cv2.waitKey(0)
cv2.destroyAllWindows()
```

4. Hough Transform

```
image = cv2.imread('K:\sem7\sammy.jpg')
```

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

 # Apply edge detection (optional but often used before HoughTransform)
edges = cv2.Canny(gray_image, 50, 150, apertureSize=3)

 # Apply Hough Line Transform
lines = cv2.HoughLines(edges, 1, np.pi / 180, threshold=100)

 # Draw the detected lines on the original image
for line in lines:
        rho, theta = line[0]a =
        np.cos(theta)
        b = np.sin(theta)x0
        = a * rho
        y0 = b * rho
        x1 = int(x0 + 1000 * (-b))y1 =
        int(y0 + 1000 * (a)) x2 = int(x0
        - 1000 * (-b))y2 = int(y0 -
        1000 * (a))
        cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

 # Display Hough-transformed images
 cv2.imshow('Hough Lines', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```