```
import numpy as np
```

```
my_list=[1,2,3,4,5]
my_list
```

```
[1, 2, 3, 4, 5]
```

```
arr=np.array([2,3,4,5,6,7,8])
arr
```

```
array([2, 3, 4, 5, 6, 7, 8])
```

```
type(arr)
```

```
numpy.ndarray
```

```
one_darray=np.arange(0,10)
one_darray.reshape(2,2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-e6af96c1e779> in <cell line: 2>()
      1 one_darray=np.arange(0,10)
----> 2 one_darray.reshape(2,2)

ValueError: cannot reshape array of size 10 into shape (2,2)
```

SEARCH STACK OVERFLOW

```
one_darray.reshape(5,2)
```

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

```
np.arange(0,15,2)
# 2 is step-size ( 2-1 ) is the skipping
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14])
```

```
np.zeros((2,2))
```

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

```
np.ones((2,2))
```

```
array([[1., 1.],
       [1., 1.]])
```

```
np.random.seed(101)
arr=np.random.randint(0,100,10)
arr
```

```
array([95, 11, 81, 70, 63, 87, 75,  9, 77, 40])
```

```
# np.random.seed(101)
arr=np.random.randint(0,100,10)
arr
```

```
array([ 4, 63, 40, 60, 92, 64,  5, 12, 93, 40])
```

```
np.random.seed(35)
arr=np.random.randint(0,100,10)
arr
```

```
array([73, 15, 55, 33, 63, 64, 11, 11, 56, 72])
```

```
arr.min()
```

```
11
```

```
arr.max()
```

```
73
```

```
arr.mean()
```

```
45.3
```

```
arr.argmin()
```

```
6
```

```
arr.argmax()
```

```
0
```

```
arr
```

```
array([73, 15, 55, 33, 63, 64, 11, 11, 56, 72])
```

INDEXING AND SLICING

```
a=np.arange(0,90).reshape(10,9)
a
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8],
       [ 9, 10, 11, 12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23, 24, 25, 26],
       [27, 28, 29, 30, 31, 32, 33, 34, 35],
       [36, 37, 38, 39, 40, 41, 42, 43, 44],
       [45, 46, 47, 48, 49, 50, 51, 52, 53],
       [54, 55, 56, 57, 58, 59, 60, 61, 62],
       [63, 64, 65, 66, 67, 68, 69, 70, 71],
       [72, 73, 74, 75, 76, 77, 78, 79, 80],
       [81, 82, 83, 84, 85, 86, 87, 88, 89]])
```

```
a[2,3]
```

```
21
```

```
row=2
column=3
a[row,column]
```

```
21
```

```
# SLICING
# from 2nd row till 5th row - 1 ( n-1)th row
a[2:5]
```

```
array([[18, 19, 20, 21, 22, 23, 24, 25, 26],
       [27, 28, 29, 30, 31, 32, 33, 34, 35],
       [36, 37, 38, 39, 40, 41, 42, 43, 44]])
```

```
a[5:,8]
```

```
array([53, 62, 71, 80, 89])
```

```
a[5,:]
```

```
array([45, 46, 47, 48, 49, 50, 51, 52, 53])
```

```
a[0:3,0:3]
```

```
array([[ 0,  1,  2],
       [ 9, 10, 11],
       [18, 19, 20]])
```

```
import matplotlib.pyplot as plt
from PIL import Image
pic=Image.open('/content/randomimage.jpeg')
pic
```

```
pic_arr=np.asarray(pic)
pic_arr.shape
#height (177)
#width (284)
#channel (3)
```

```
(177, 284, 3)
```

```
pic1=Image.open('/content/grayscaleimg.jpg')
pic1
```

```
pic_arr=np.asarray(pic1)
pic_arr.shape
#height (530)
#width (350)
#channel (3)
```

```
(530, 350, 3)
```

```
pic_arr.max()
```

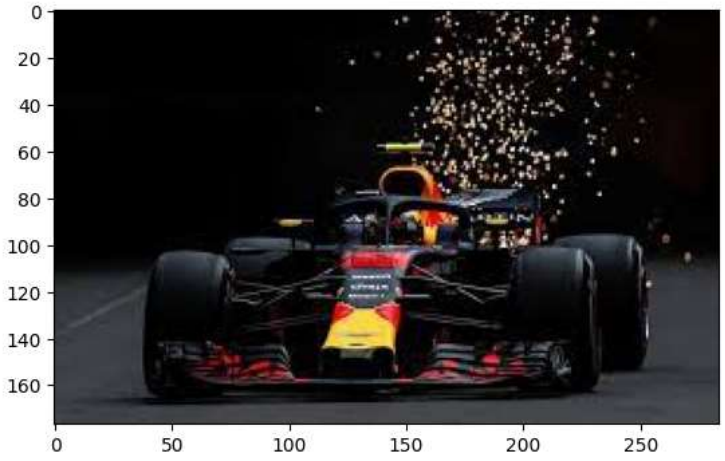
```
224
```

```
plt.imshow(pic1)
```

```
<matplotlib.image.AxesImage at 0x7882bc7adf90>
```


```
plt.imshow(pic)
```

```
<matplotlib.image.AxesImage at 0x7882bc399990>
```



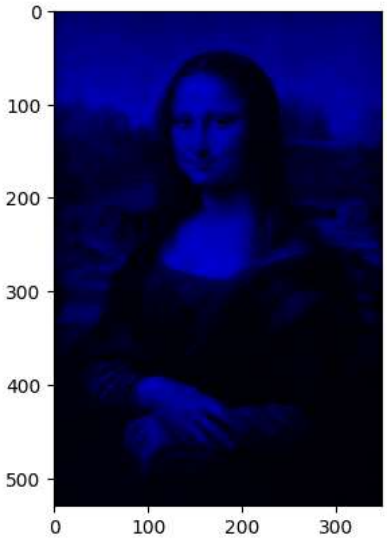
DAY 2

```
pic_red=pic_arr.copy()
```

```
pic_red[:, :, 0]= 0
 # zero out contribution form grenn
pic_red[:, :, 1]= 0
 # zero out contribution from blue
```

```
plt.imshow(pic_red)
```

```
<matplotlib.image.AxesImage at 0x788287cc7640>
```




```
# day 2 new notebook
```

```
# day 2 new notebook
# open images files in a notebook


import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline


img=cv2.imread('/content/randomimage1.jpeg')
print(img)
```

```
None
```

```
img=cv2.imread('/content/randomimage.jpeg')
print(img)
```

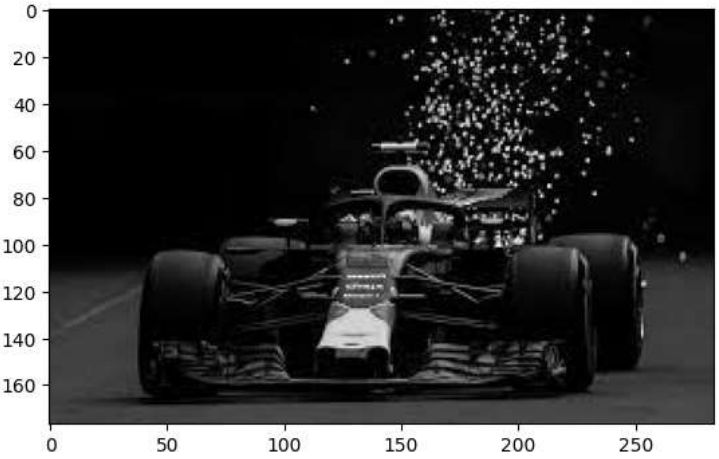
```
[[[ 0  0  0]
  [ 0  0  0]
  [ 0  0  0]
  ...
  [ 0  1  2]
  [ 0  1  2]
  [ 0  1  2]]

 [[ 0  0  0]
  [ 0  0  0]
  [ 0  0  0]
  ...
  [ 0  1  2]
  [ 0  1  2]
  [ 0  1  2]]

 [[ 0  0  0]
  [ 0  0  0]
  [ 0  0  0]
  ...
  [ 0  1  2]
  [ 0  1  2]
  [ 0  1  2]]

 ...

 [[77 72 71]
  [77 72 71]
  [76 71 70]
  ...
  [76 71 70]
  [76 71 70]
  [77 72 71]]

 [[72 67 66]
  [71 66 65]
  [71 66 65]
  ...
  [76 71 70]
  [76 71 70]
  [77 72 71]]

 [[62 57 56]
  [72 67 66]
  [75 70 69]
  ...
  [82 77 76]
  [82 77 76]
  [82 77 76]]]
```

```
img_rgb=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
img_rgb.shape
```

```
(177, 284, 3)
```

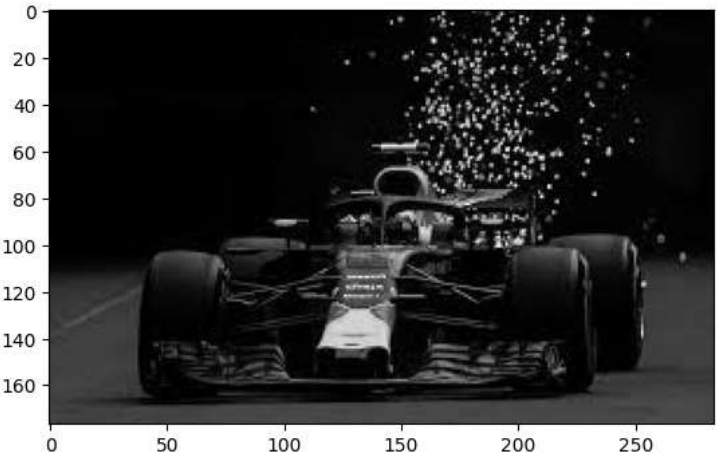
```
# cv2.IMREAD_GRAYSCALE
img_gray=cv2.imread('/content/randomimage.jpeg',cv2.IMREAD_GRAYSCALE)
plt.imshow(img_gray,cmap='gray')
# img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
# plt.imshow(img_rgb)
img_gray.shape
```

```
(177, 284)
```



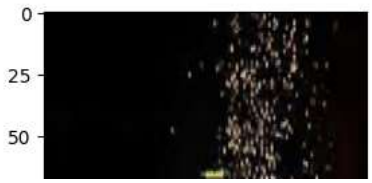
```
# cv2.COLOR_BGR2GRAY
img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plt.imshow(img_gray,cmap="gray")
img_gray.shape
```

```
(177, 284)
```



```
img=cv2.resize(img_rgb,(130,200))
plt.imshow(img)
# Interpolation - increasing / decreasing the size of the image
```

```
<matplotlib.image.AxesImage at 0x7cabe2abb430>
```

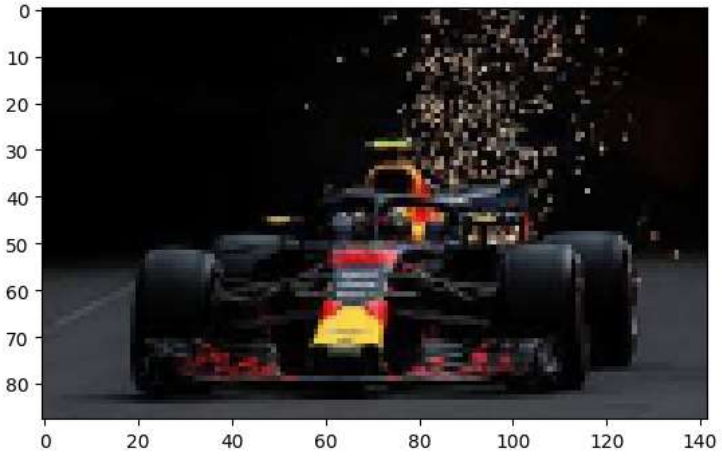



```
# RESIZE USING RATIO
w_ratio=0.5
h_ratio=0.5
new_img=cv2.resize(img_rgb,(0,0),img_rgb,0.8,0.3)
plt.imshow(new_img)

# we are passing height and width as 0,0
```

```
<matplotlib.image.AxesImage at 0x7cabe6b010f0>
```


```
new_img=cv2.resize(img_rgb,(0,0),img_rgb,w_ratio,h_ratio)
plt.imshow(new_img)

# new_width = original_width * w_ratio
# new_height = original_height * h_ratio
```

```
<matplotlib.image.AxesImage at 0x7cabe2752350>
```



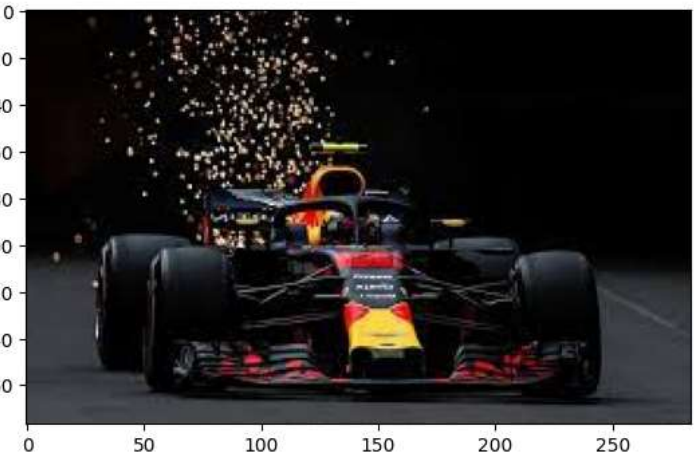

```
# Flipping Images
# 0 - along y axis
# 1- mirror along y axis
# -1 - along x axis


img_flip=cv2.flip(img_rgb,-1)
plt.imshow(img_flip)
```

```
<matplotlib.image.AxesImage at 0x7cabe26e9660>
```



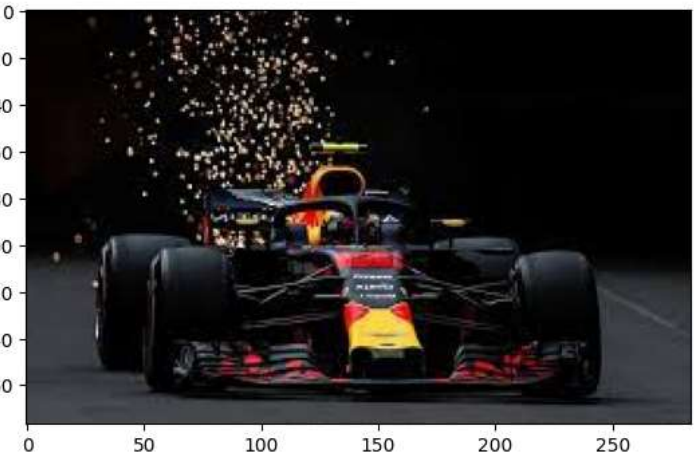
```
img_flip0=cv2.flip(img_rgb,0)
plt.imshow(img_flip)
```

```
<matplotlib.image.AxesImage at 0x7cabe28429e0>
```




```
img_flip=cv2.flip(img_rgb,1)
plt.imshow(img_flip)
```

```
<matplotlib.image.AxesImage at 0x7cabe2822e30>
```




```
# SAVING IMAGES
type(img_flip0)
```

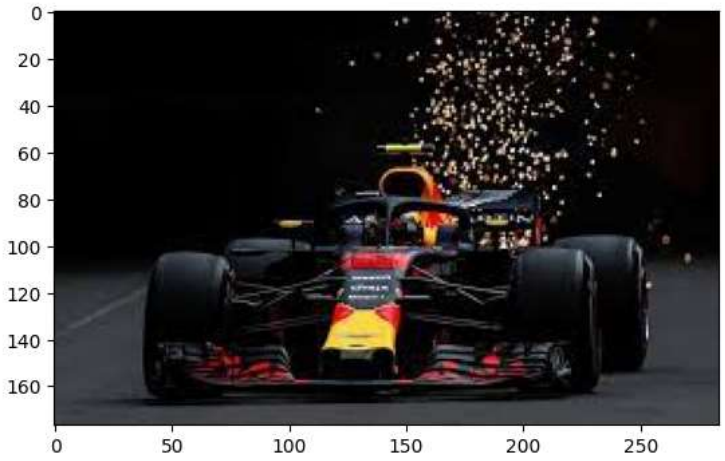
```
numpy.ndarray
```

```
cv2.imwrite('flippedimaged.jpeg',img_flip0)
```

```
True
```

```
 # Interpolation is the way yhe extra pixels in the new image is calculated.
# If the original image is small, then the largest image is rescaled image has extra pixels which is not exactly the same as a nearby pix

# inter_nearest - a nearest-neigbor interpolation
# inter_linear - a bilinear interpolation ( used by default )
# inter_cubic - a bicubic interpolation over 4x4 pixel neigbourhood
# inter_lanczos4 - a lanczos interpolation over 8x8 pixel


import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import cv2

img_path='/content/randomimage.jpeg'
img=cv2.imread(img_path,cv2.COLOR_BGR2RGB)
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x79f007d67af0>
```

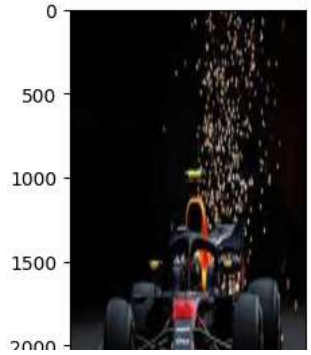



```
img_nearest=cv2.resize(img,(1400,2950),interpolation=cv2.INTER_NEAREST)
img_nearest
plt.imshow(img_nearest)
```

```
<matplotlib.image.AxesImage at 0x79f0041c6fe0>
```


```
img_bilinear=cv2.resize(img,(1400,2950),interpolation=cv2.INTER_LINEAR)
img_bilinear
plt.imshow(img_bilinear)
```

```
<matplotlib.image.AxesImage at 0x79efffe62200>
```




```
img_bicubic=cv2.resize(img,(1400,2950),interpolation=cv2.INTER_CUBIC)
img_bicubic
plt.imshow(img_bicubic)
```

```
<matplotlib.image.AxesImage at 0x79efffedccd0>
```


```
img_lanczos=cv2.resize(img,(1400,2950),interpolation=cv2.INTER_LANCZOS4)
img_lanczos
plt.imshow(img_lanczos)
```

```
<matplotlib.image.AxesImage at 0x79efffd44250>
```


```
# DRAWING ON IMAGE
BLANK_IMG=np.zeros(shape=(512,512,3),dtype=np.int16)
BLANK_IMG
```

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

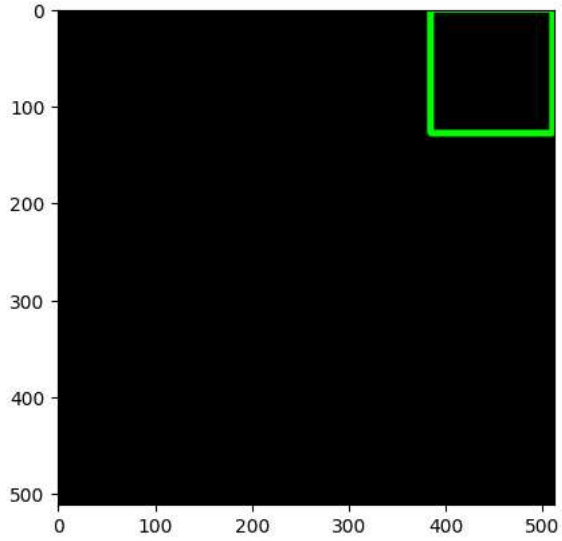
```
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        ...,
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        ...,
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       ...,

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        ...,
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        ...,
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        ...,
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]]], dtype=int16)
```

```
BLANK_IMG.shape
```

```
(512, 512, 3)
```

```
# pt1 = top left
# pt2 = botton right
cv2.rectangle(BLANK_IMG,pt1=(384,0),pt2=(510,128),color=(0,255,0),thickness=5)
plt.imshow(BLANK_IMG)
```

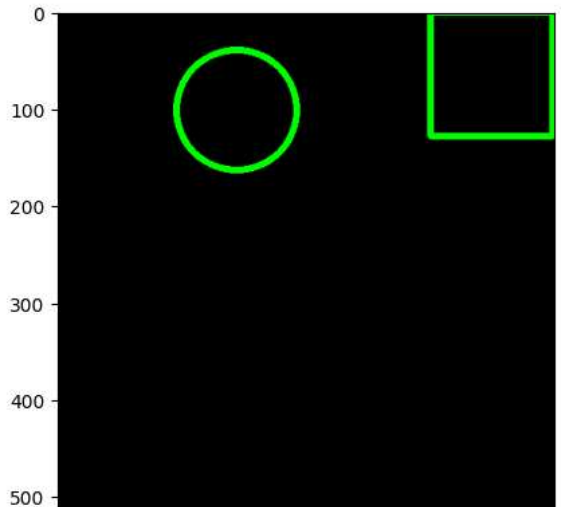
```
<matplotlib.image.AxesImage at 0x79efffaa7760>
```




```
cv2.circle(BLANK_IMG,center=(184,101),radius=62,color=(0,255,0),thickness=5)
plt.imshow(BLANK_IMG)
```

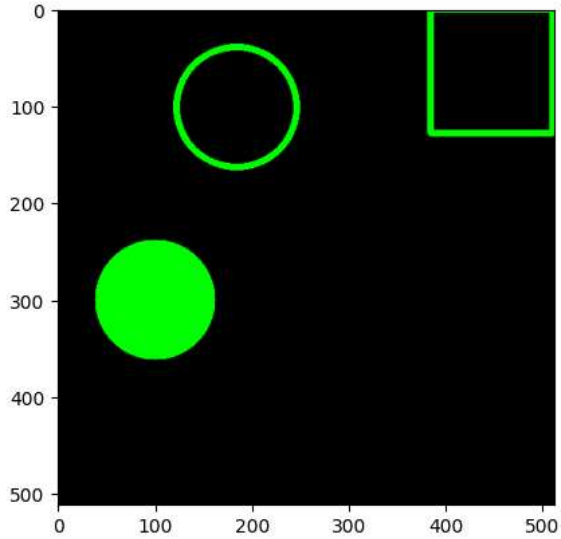
```
# for circle give radius
```

```
<matplotlib.image.AxesImage at 0x79efff412410>
```



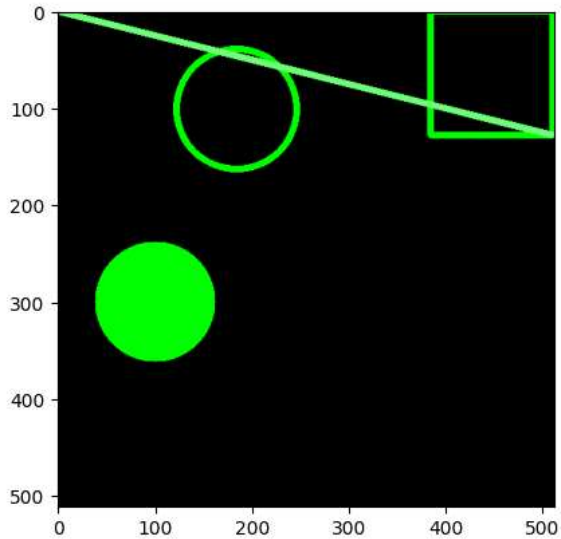

```
cv2.circle(BLANK_IMG,center=(100,300),radius=62,color=(0,255,0),thickness=-1)
plt.imshow(BLANK_IMG)

# give negative thickness ( constant = -1 )
```

```
<matplotlib.image.AxesImage at 0x79efff4836d0>
```



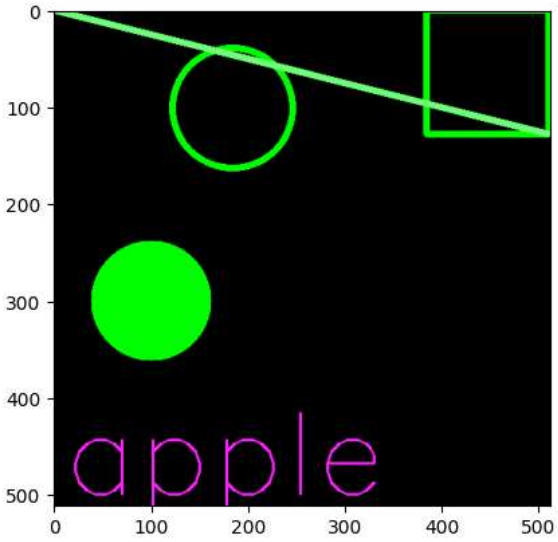

```
cv2.line(BLANK_IMG,pt1=(0,0),pt2=(510,128),color=(110,255,120),thickness=5)
plt.imshow(BLANK_IMG)

# same parameters just like the rectangle function
```

```
<matplotlib.image.AxesImage at 0x79efff4efbe0>
```



```
font=cv2.FONT_HERSHEY_SIMPLEX
a=input("enter the text duh??")
cv2.putText(BLANK_IMG,text=a,org=(10,500),fontFace=font,fontScale=4,color=(255,32,255),thickness=2,)
plt.imshow(BLANK_IMG)

# org - origin
```

```
enter the text duh??apple
<matplotlib.image.AxesImage at 0x79efff37c8b0>
```




```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import cv2

img='/content/unnamed.jpg'
img=cv2.imread(img,cv2.COLOR_BGR2RGB)
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x79effe9362c0>
```


```
cv2.circle(img,center=(350,241),radius=110,color=(0,255,0),thickness=5)
plt.imshow(img)

# for circle give radius
```

```
<matplotlib.image.AxesImage at 0x79effe99b730>
```


```
font=cv2.FONT_HERSHEY_SIMPLEX
a=input("enter the text duh??")
cv2.putText(img,text=a,org=(200,400),fontFace=font,fontScale=0.5,color=(255,32,255),thickness=2,)
plt.imshow(img)

# org - origin
```

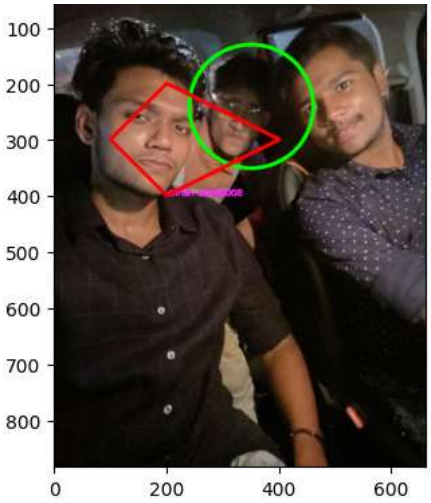
```
enter the text duh??dhrish e0320008
<matplotlib.image.AxesImage at 0x79effe829120>
```


```
vertices=np.array([[100,300],[200,200],[400,300],[200,400]],np.int32)
vertices
```

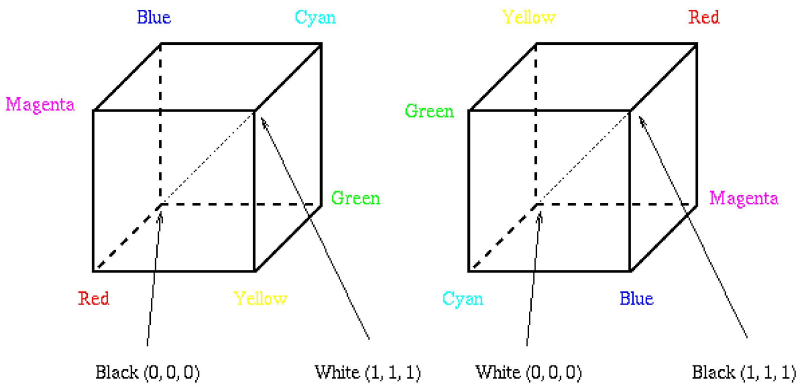
```
array([[100, 300],
       [200, 200],
       [400, 300],
       [200, 400]], dtype=int32)
```

```
pts=vertices.reshape((-1,1,2))
cv2.polylines(img,[pts],isClosed=True,color=(255,0,0),thickness=5)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x79effe89a740>
```

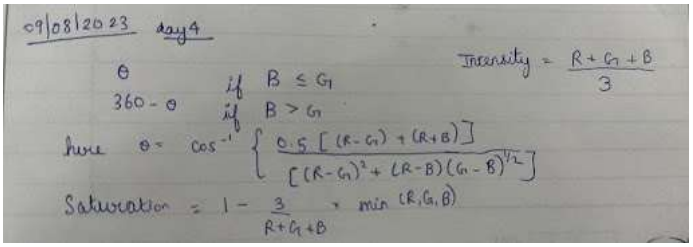


```
# Introduction to Image processing

# Module 2 -  Image processing - digital Image processing
# Feature of the image or information about the image
# Take image as input but in image processing the output is also an image whereas in computer vision the output can be some features/inf
# Image processing o/p - image
# computer vision - feature / information about the image

# Image transformation  - Flipping , rotation and crop

# Color Mapping - Additive Color Model and Subtractive Color Model

# Additive Color Model - Red Green Blue ( Digital Displays) ( combing two lights - using the lights)
# Printer - SUbtractive color Model ( Ceyon Magenta Yellow Black ) ( uses the ink to display the color)

# Computer generally displays rgb using 8-red,green,blue ( 8 for each of them) ->
# Since each bit can be 0 or 1 ->
# thats why 2 power 8 = 256 channels for each of them
```



The RGB Cube The CMY Cube

```
# HUE or HSL

# 1) Hue : It is a color attribute that describes a pure color
# 2) Saturation : It measures the extent to which a pure color within is diluted by white light ( darkness or whiteness)
# 3) Intensity : Key factor in describing the color sensation p;.
```

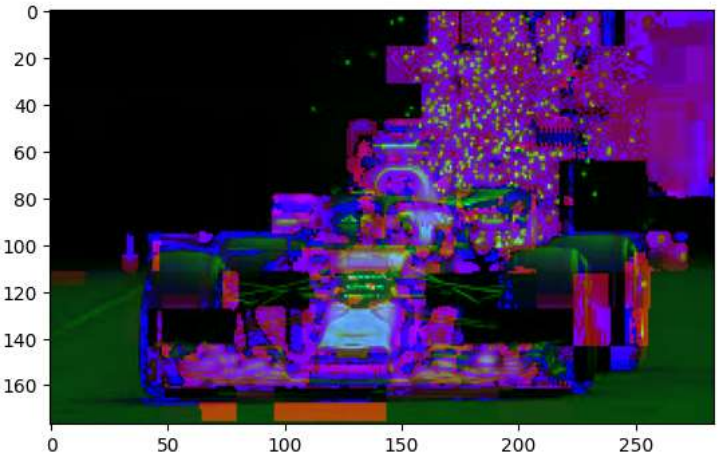



```
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline


img=cv2.imread('/content/randomimage.jpeg')


# converting to different color spaces
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f040f6a7190>
```

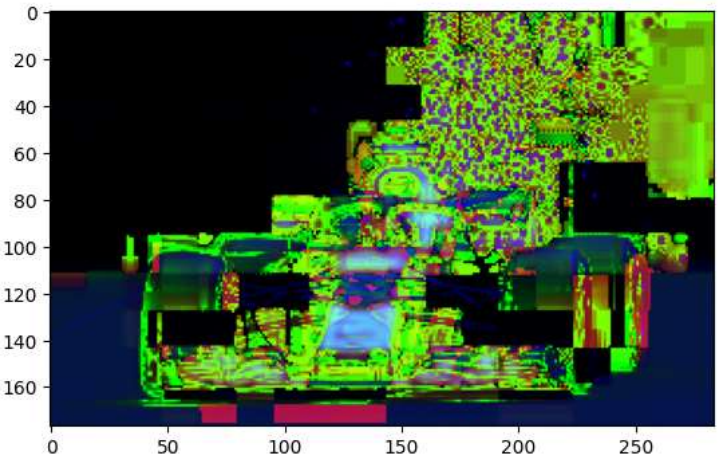
```
# converting to different color spaces BGR to HLS

img1=cv2.cvtColor(img,cv2.COLOR_BGR2HLS)
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7f040f6ff130>
```



```
# converting to different color spaces BGR to HSV

img2=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7f040f8886d0>
```

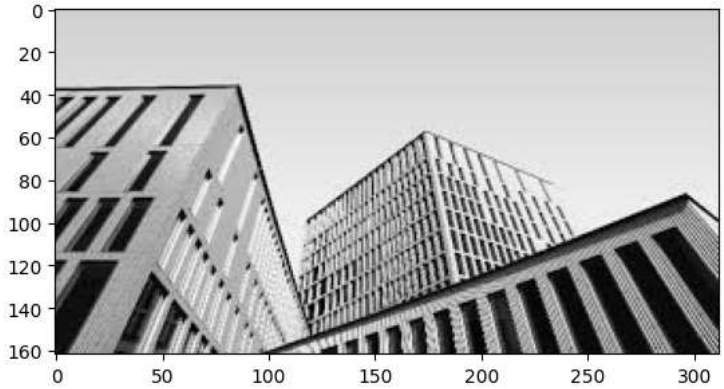


```
# blending - mixing of 2 images -> size should be same for blending
# pasting - one image on the top of another image


img=cv2.imread('/content/randomimage.jpeg')
img1=cv2.imread('/content/b&wimages.jpeg')


plt.imshow(img)
img.shape
```

```
(177, 284, 3)
```

```
plt.imshow(img1)
img1.shape
```

```
(162, 312, 3)
```



```
img=cv2.resize(img,(177,284))
img1=cv2.resize(img1,(177,284))
```

```
img.shape
```

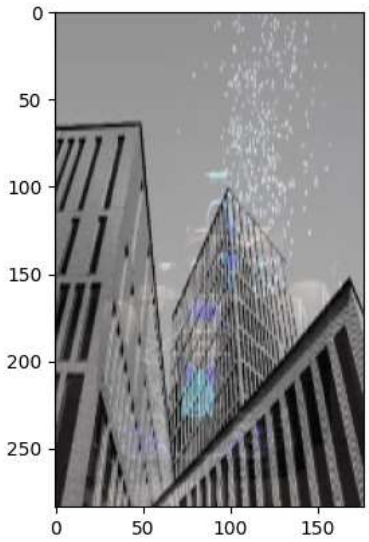
```
(284, 177, 3)
```

```
img1.shape
```

```
(284, 177, 3)
```

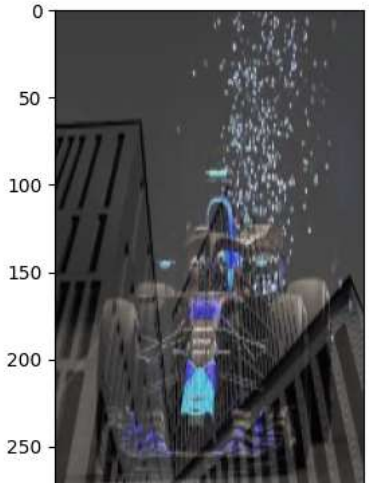
```
blended=cv2.addWeighted(src1=img1,alpha=0.7,src2=img,beta=0.3,gamma=0)
plt.imshow(blended)
```

```
<matplotlib.image.AxesImage at 0x7f040f6fc0d0>
```



```
blended=cv2.addWeighted(src1=img,alpha=0.7,src2=img1,beta=0.3,gamma=0)
plt.imshow(blended)
```

```
<matplotlib.image.AxesImage at 0x7f040f479450>
```



Colab paid products - Cancel contracts here

✓ 2s completed at 12:43 PM

Blending and Pasting Images

For some computer vision systems, we'll want to be able to post 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. We'll see a bit of this in our code.

Blending Images

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

```
import cv2


# Two images
img1 = cv2.imread('/content/dog_backpack.jpg')
img2 = cv2.imread('/content/watermark_no_copy.png')


img1.shape
```

```
(1401, 934, 3)
```

```
img2.shape
```

```
(1280, 1277, 3)
```

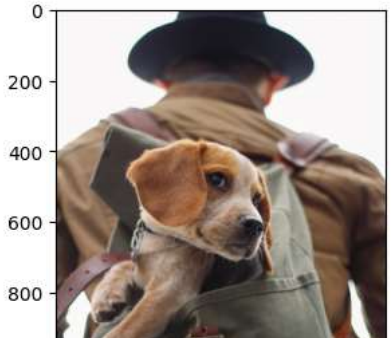
```
import matplotlib.pyplot as plt
%matplotlib inline


plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7dd4762250f0>
```

Whoops! Let's remember to fix the RGB!

```
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)


plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7dd47614b250>
```



```
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7dd4761fcf70>
```

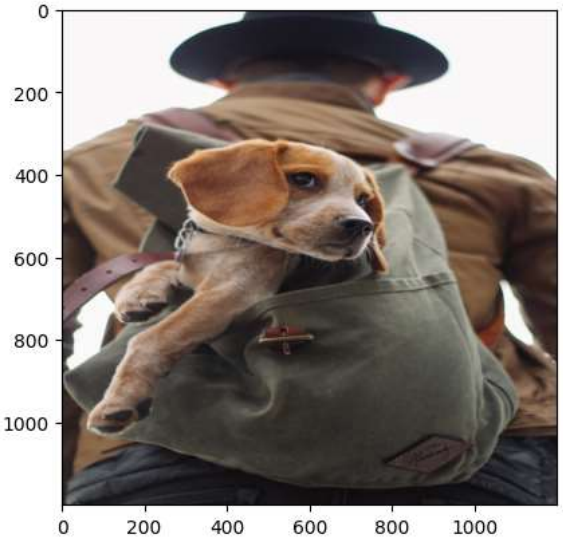
Resizing the Images

```
img1 =cv2.resize(img1,(1200,1200))
img2 =cv2.resize(img2,(1200,1200))
```

Let's practice resizing the image, since the DO NOT COPY image is actually quite large 1200 by 1200, and our puppy in backpack image is 1400 by 1000

```
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7dd4760b6fe0>
```



```
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7dd475f403a0>
```

Blending the Image

We will blend the values together with the formula:

$$img1 * \alpha + img2 * \beta + \gamma$$

```
img1.shape
```

```
(1200, 1200, 3)
```

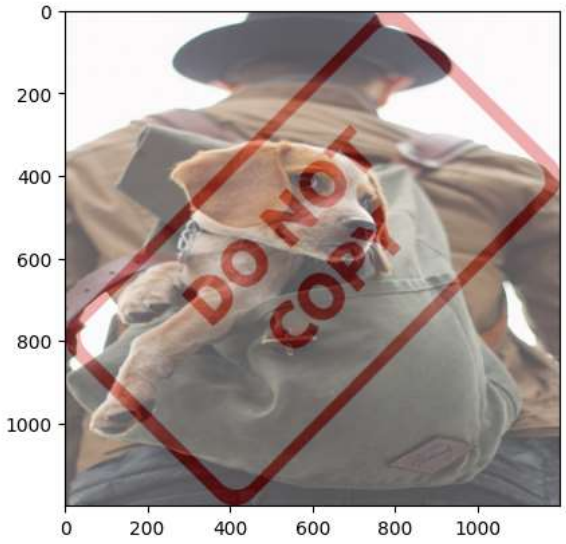
```
img2.shape
```

```
(1200, 1200, 3)
```

```
blended = cv2.addWeighted(src1=img1,alpha=0.7,src2=img2,beta=0.3,gamma=0)
```

```
plt.imshow(blended)
```

```
<matplotlib.image.AxesImage at 0x7dd475fb02e0>
```



Overlaying Images of Different Sizes

We can use this quick trick to quickly overlap different sized images, by simply reassigning the larger image's values to match the smaller image.

```
# Load two images
img1 = cv2.imread('/content/dog_backpack.jpg')
img2 = cv2.imread('/content/watermark_no_copy.png')
img2 =cv2.resize(img2,(600,600))
```

```
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

large_img = img1
small_img = img2


x_offset=0
y_offset=0


large_img[y_offset:y_offset+small_img.shape[0], x_offset:x_offset+small_img.shape[1]] = small_img


plt.imshow(large_img)
```

```
<matplotlib.image.AxesImage at 0x7dd475e20730>
```

Blending Images of Different Sizes

Imports

```
import numpy as np
import cv2
```

Importing the images again and resizing

```
# Load two images
img1 = cv2.imread('/content/dog_backpack.jpg')
img2 = cv2.imread('/content/watermark_no_copy.png')
img2 =cv2.resize(img2,(600,600))

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)


plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7dd475f0e0b0>
```


```
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7dd475ddbc70>
```

Create a Region of Interest (ROI)

```
img1.shape
```

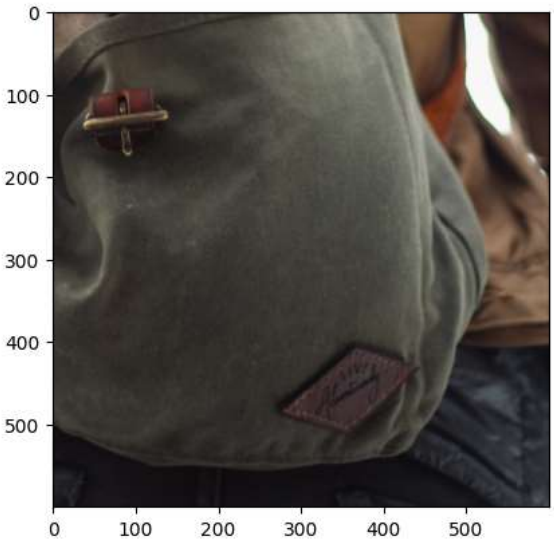
```
(1401, 934, 3)
```

```
x_offset=934-600
y_offset=1401-600
```

```
# Creating an ROI of the same size of the foreground image (smaller image that will go on top)
rows,cols,channels = img2.shape
# roi = img1[0:rows, 0:cols ] # TOP LEFT CORNER
roi = img1[y_offset:1401,x_offset:943] # BOTTOM RIGHT CORNER
```

```
plt.imshow(roi)
```

```
<matplotlib.image.AxesImage at 0x7dd475cd5cc0>
```




```
roi.shape
```

```
(600, 600, 3)
```

Creating a Mask

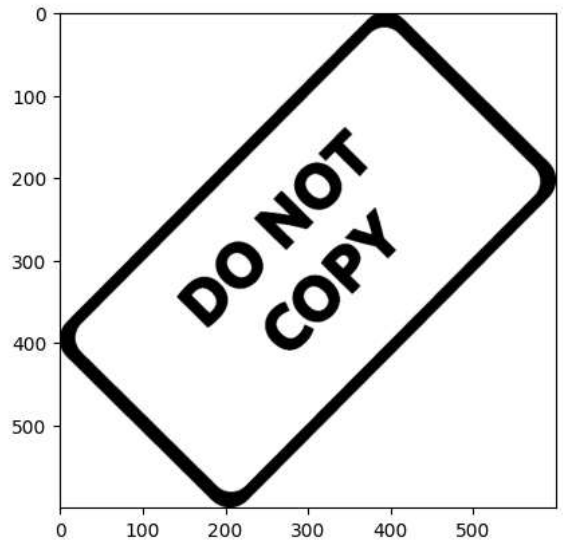
```
# Now create a mask of logo and create its inverse mask also
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
```

```
img2gray.shape
```

```
(600, 600)
```

```
plt.imshow(img2gray,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7dd475b47340>
```



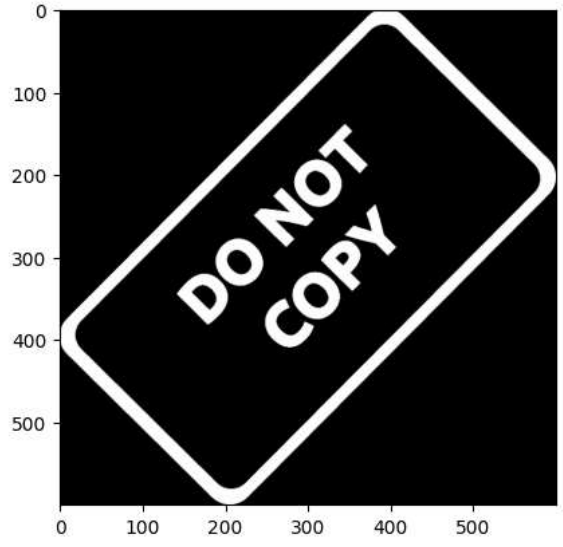

```
mask_inv = cv2.bitwise_not(img2gray)
```

```
mask_inv.shape
```

```
(600, 600)
```

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

```
<matplotlib.image.AxesImage at 0x7dd475bbf820>
```



Convert Mask to have 3 channels

```
white_background = np.full(img2.shape, 255, dtype=np.uint8)
```

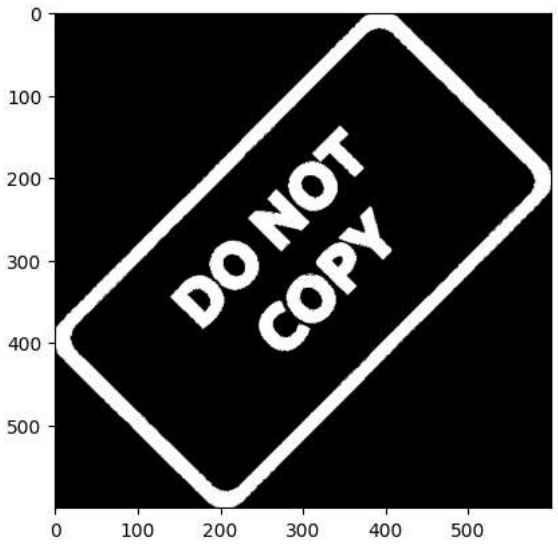
```
bk = cv2.bitwise_or(white_background, white_background, mask=mask_inv)
```

```
bk.shape
```

```
(600, 600, 3)
```

```
plt.imshow(bk)
```

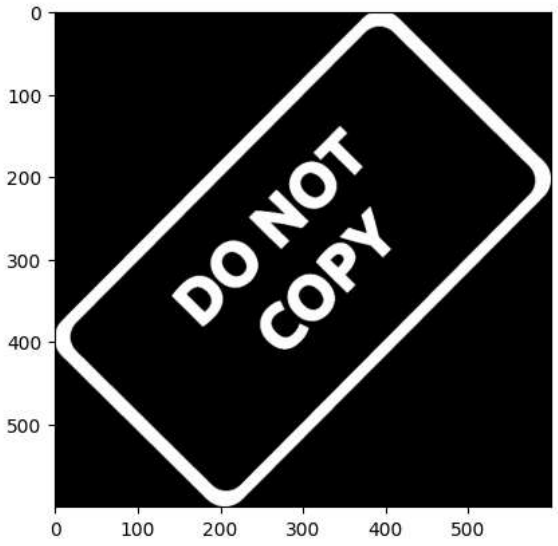
```
<matplotlib.image.AxesImage at 0x7dd475a48b80>
```



Grab Original FG image and place on top of Mask

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

```
<matplotlib.image.AxesImage at 0x7dd475ac10f0>
```




```
fg = cv2.bitwise_or(img2, img2, mask=mask_inv)
```

```
plt.imshow(fg)
```

```
<matplotlib.image.AxesImage at 0x7dd47592a740>
```


```
fg.shape
```

```
(600, 600, 3)
```

Get ROI and blend in the mask with the ROI

```
final_roi = cv2.bitwise_or(roi,fg)
```

```
plt.imshow(final_roi)
```

```
<matplotlib.image.AxesImage at 0x7dd475999330>
```

Now add in the rest of the image

```
large_img = img1
small_img = final_roi


large_img[y_offset:y_offset+small_img.shape[0], x_offset:x_offset+small_img.shape[1]] = small_img

plt.imshow(large_img)
```

<matplotlib.image.AxesImage at 0x7dd475824790>

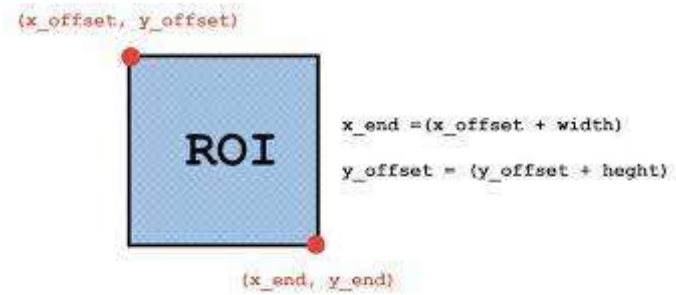
Great Work!

Check out these documentation examples and links for more help for these kinds of tasks (which can be really tricky!)

1. https://stackoverflow.com/questions/10469235/opencv-apply-mask-to-a-color-image/38493075
2. https://stackoverflow.com/questions/14063070/overlay-a-smaller-image-on-a-larger-image-python-opencv
3. https://docs.opencv.org/3.4/d0/d86/tutorial_py_image_arithmetics.html

✓ 0s completed at 10:47 AM

```
# Region of Intrest

# (x_offset, y_offset) - starting point
# x_end = (x_offset + width ) # width of the small image
# y_offset = (y_offset + height ) # height of the small image
# X_end, y_end -> ending point ( calculate this )
```

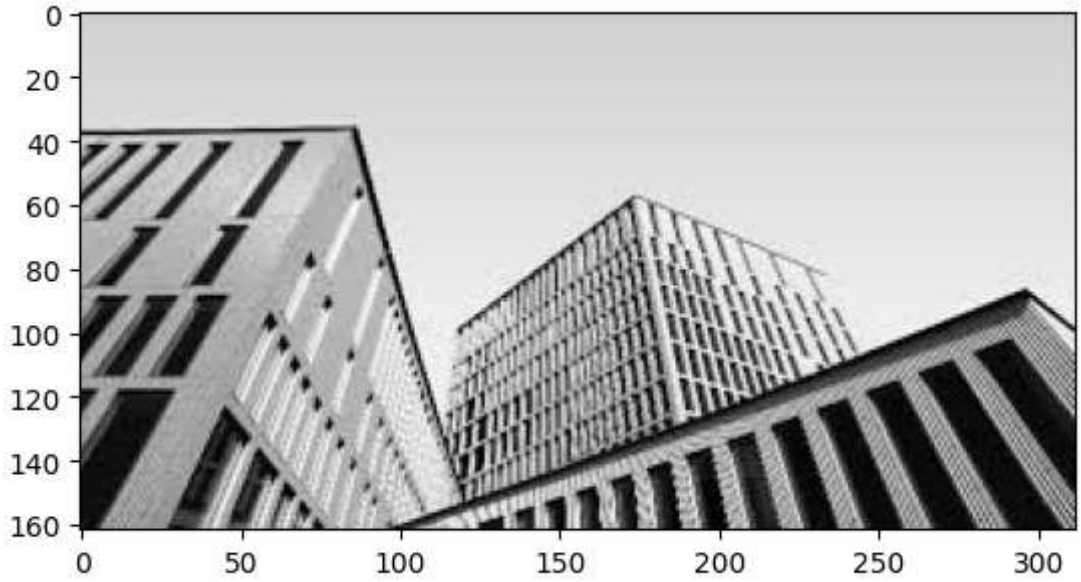



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


img=cv2.imread('/content/randomimage.jpeg') # small image
img1=cv2.imread('/content/b&wimages.jpeg') # larger image



# converting to different color spaces

# THIS IS A SMALLER IMAGE

img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
img=cv2.resize(img,(57,54))
img.shape
```

```
(54, 57, 3)
```


```
# THIS IS GOING TO BE THE LARGER IMAGE

# converting to different color spaces
img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
plt.imshow(img1)
img1=cv2.resize(img1,(157,154))

img1.shape
```

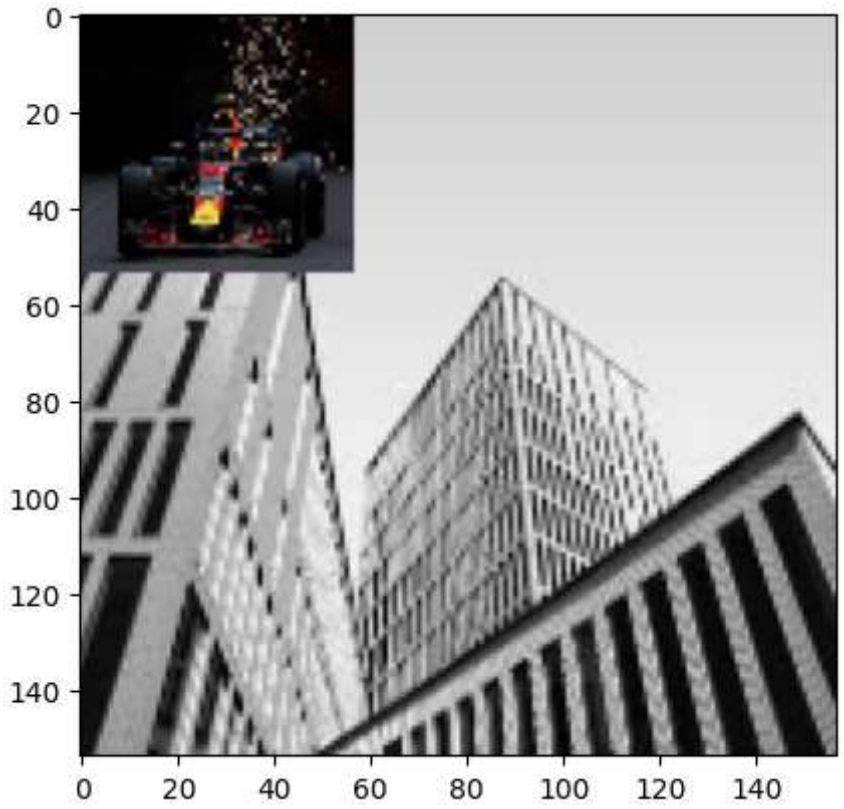
```
(154, 157, 3)
```




```
# Set the initial x and y offsets to 0
x_offset=0
y_offset=0

# Calculate the ending y-coordinate of the image by adding the image's height to the y off
y_end=y_offset+img.shape[0]

# Calculate the ending x-coordinate of the image by adding the image's width to the x offs
x_end=x_offset+img.shape[1]

# x_offset and y_offset are starting points for positioning an image on a larger canvas.
# y_end is calculated by adding the height of the image to the y offset. This gives the y-
# x_end is calculated by adding the width of the image to the x offset. This gives the x-c


img1[y_offset:y_end,x_offset:x_end]=img

# This line of code is copying the contents of the img (presumably an image) and pasting i
# The positioning of this copy-paste action is determined by the values of y_offset, y_enc

# y_offset:y_end specifies a range of rows in the img1 image where the copied img will be
```

```
# It starts from the y_offset row and goes up to (but doesn't include) the y_end row.

# x_offset:x_end specifies a range of columns in the img1 image where the copied img will
# It starts from the x_offset column and goes up to (but doesn't include) the x_end columr

# So, effectively, this line of code is placing the contents of img onto img1 at a specifi
# The result is that img will appear within the defined region of img1.


plt.imshow(img1)
```

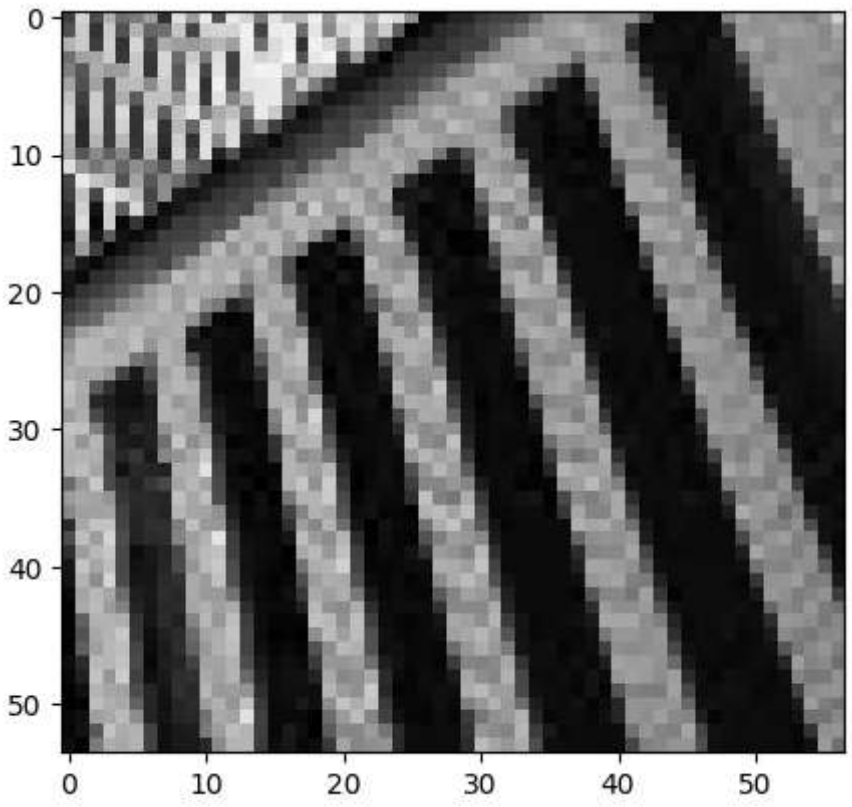
```
<matplotlib.image.AxesImage at 0x7affa4bb4a90>
```




```
# REGION OF INTREST

# x_offset and y_offset are being calculated based on the difference between certain value
# These values are used to determine where the smaller image (foreground image) will be pl

x_offset = 154 - 54
y_offset = 157-57
```

```
# Creating an ROI of the same size of the
# foreground image ( smaller image that will go on top )
rows,cols,channels = img.shape

# roi = img1[0:rows,0:cols] # TOP LEFT CORNER
# The Region of Interest (ROI) is defined in the larger image (img1) using the calculated
# The ROI is essentially a subsection of img1 that matches the size of the foreground imag

roi = img1[y_offset:154,x_offset:157]
plt.imshow(roi)
```

```
<matplotlib.image.AxesImage at 0x7affa507dae0>
```

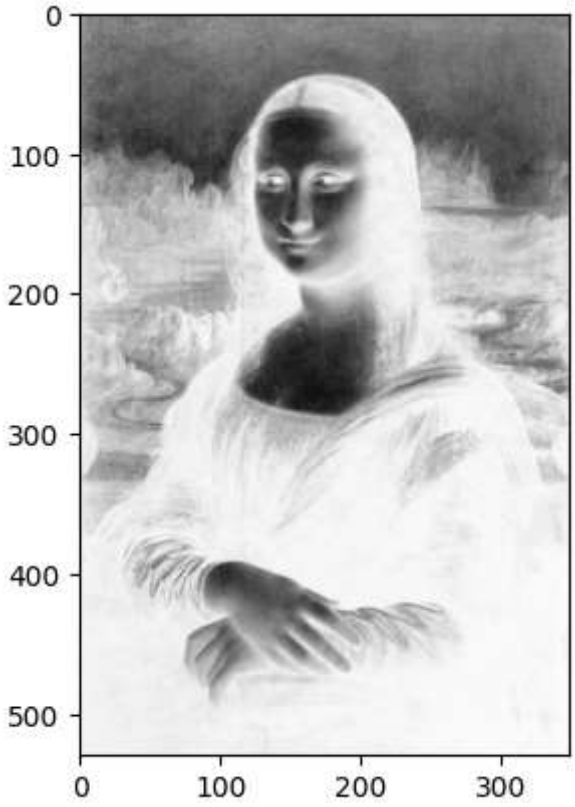



```
img=cv2.imread('/content/image.jpg')
img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plt.imshow(img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7affa484bd60>
```


```
# creating a mask -> its an inverse layer ( black and white to white and black)

# cv2.bitwise_not(img) is a function that takes an image (img) and creates an inverse vers
# This is commonly used to create masks where the background becomes the foreground and vi
mask_inv=cv2.bitwise_not(img)

# The code then displays the resulting inverted mask using plt.imshow(mask_inv, cmap='gray
# The cmap='gray' argument specifies that the colormap used for display should be grayscal
plt.imshow(mask_inv,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7affa479b280>
```




```
# Creating a 3-channel white background image with the same dimensions as img
white_bg = np.full(img.shape, 255, dtype=np.uint8)

# Using bitwise OR operation to combine the white background image with itself,
# using the mask_inv as a mask
bg = cv2.bitwise_or(white_bg, white_bg, mask=mask_inv)

# np.full(img.shape, 255, dtype=np.uint8) creates an all-white image with the same dimensi

# The cv2.bitwise_or() function performs a bitwise OR operation between two images. In thi
```

```
# This means that where the mask is black (corresponding to the original image), no change
# the white background gets combined.
```

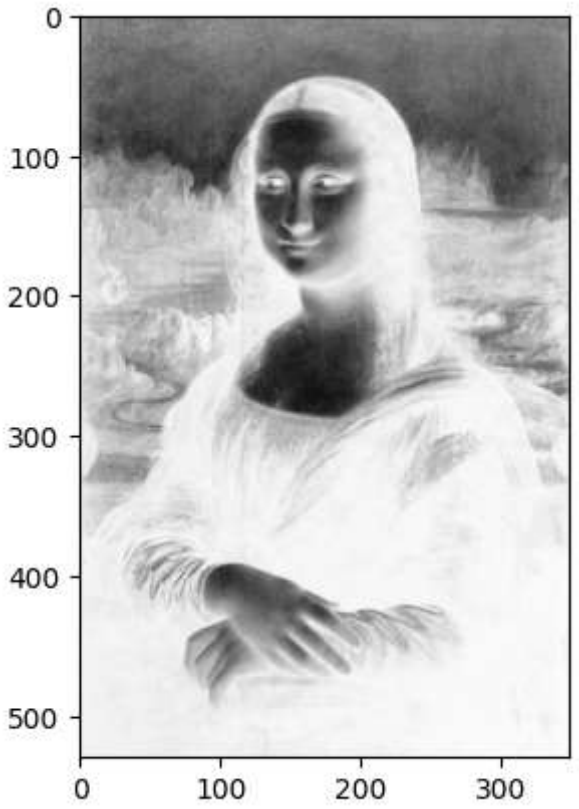
```
bg.shape
```

```
(530, 350)
```

```
# Grab Original FG image and place on top of Mask
```

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

```
<matplotlib.image.AxesImage at 0x7affa3b464a0>
```




```
fg = cv2.bitwise_or(img, img, mask=mask_inv)
```

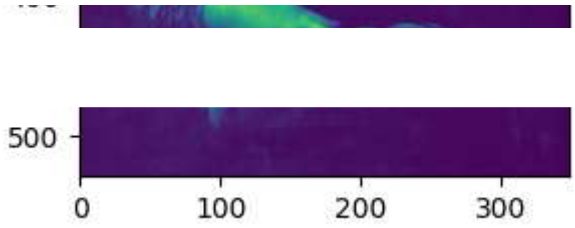
```
plt.imshow(fg)
```

```
<matplotlib.image.AxesImage at 0x7affa3a25ba0>
```


```
fg.shape
```

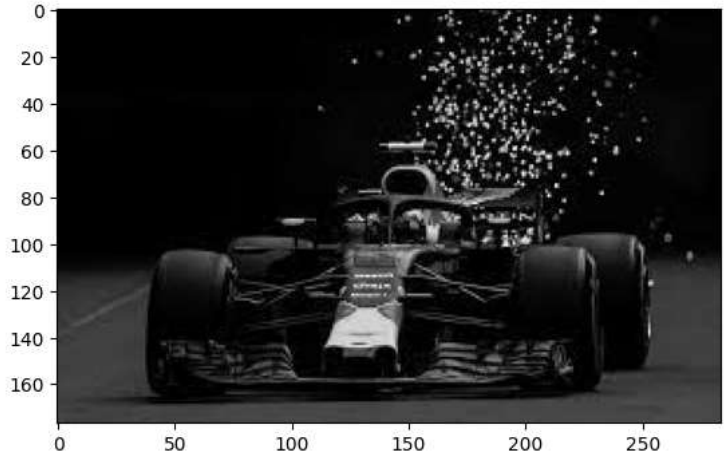
```
(530, 350)
```

```
# rest continued in 01-Blending-and-Pasting-Images.ipynb
```



```
# Thresholding -> Converting an image to consist of only two values ( white or black)
# -> Used in Segmentation specifically
# -> Used when we are looking for shapes or edges

# -> 3 types of  Thresholding Techniques :
# -> Simple - uses cv.threshold to apply thresholding
# -> Adaptive
# -> Ostu's

# i) SIMPLE THRESHOLDING -> Threshold_binary - if the pixel intensity is > then set threshold to 255
#                         - if the pixel value is < then set threshold to 0

#                         -> thresh_binary_inv - opposite of threshold binary

#                         -> thresh_trunc - pixel intensity value is greater than truncated to the threshold. the pixel value is set to be
#                         - all other values are set to be the same

#                         -> thresh_tozero - pixel intensity is set to 0, for all the pixel intensity less than the threshold value

#                         -> thresh_tozero_inv - inverted/opposite case of thresh_tozero

# SYNTAX : cv2.threshold(source,thresholdValue,maxVal,thresholding Technique)
# source : the input image
# threshold value : example 100
# maxVal : example :255
# thresholding technique : the type of thresholding


# ADAPTIVE THRESHOLDING  - block_size should be an odd number ( greater value change it to 255 and lesser value to 0)
#                         - any simple thresholding technique can be applied here

# SYNTAX : cv2.adaptive threshold(image,255,cv2.adaptive_thresh_mean_c,cv2.thresh_binary,block_size,constant)

# constant - even number ( most of the cases it is zero [0] )
# threshold value = calculated mean - constant
# example : 114 - 4 = 110 ( 110 is the threshold value )

# block_size = size of the neighborhood used for calculating the local mean

# thresh_binary - pixels above threshold white and below threshold black

# 0 - black color
# 255 - white color

# adaptive_thresh_mean_c - calculates the threshold based on the mean of the pizel values in the neigbourhood


# Day 7
# adding the 0 flag to read it in black and white
import cv2
import matplotlib.pyplot as plt
img=cv2.imread('/content/randomimage.jpeg',0)


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

```
<matplotlib.image.AxesImage at 0x7a2c94835a50>
```



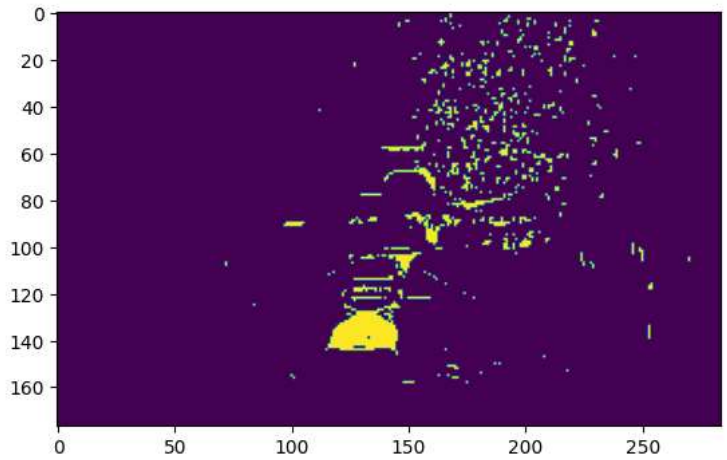

```
# Binary - Note : cv2 will return 2 vales one is threshold and then it is converted_image
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

```
threshold
```

```
127.0
```

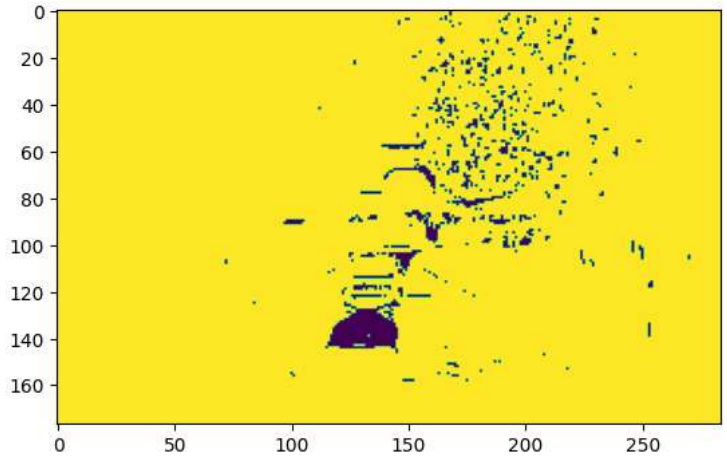
```
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c945927d0>
```



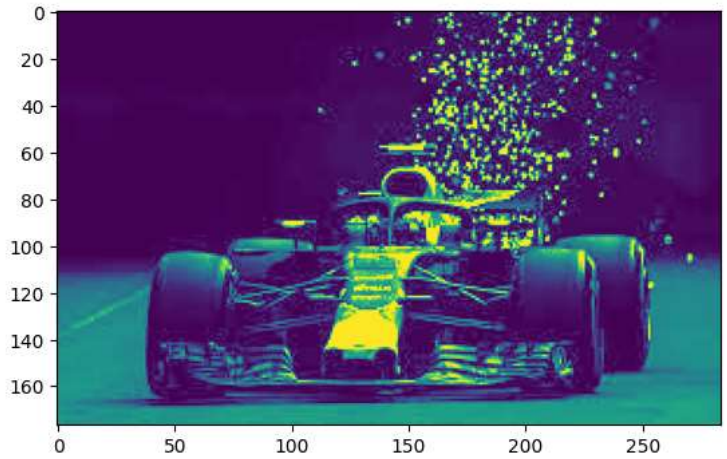

```
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c945ee4d0>
```



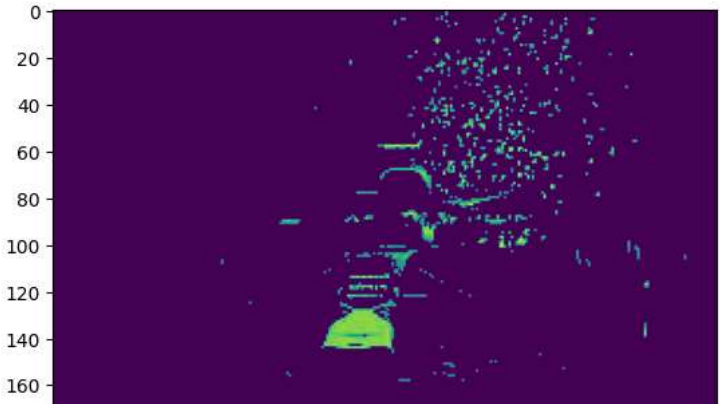

```
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c865114b0>
```



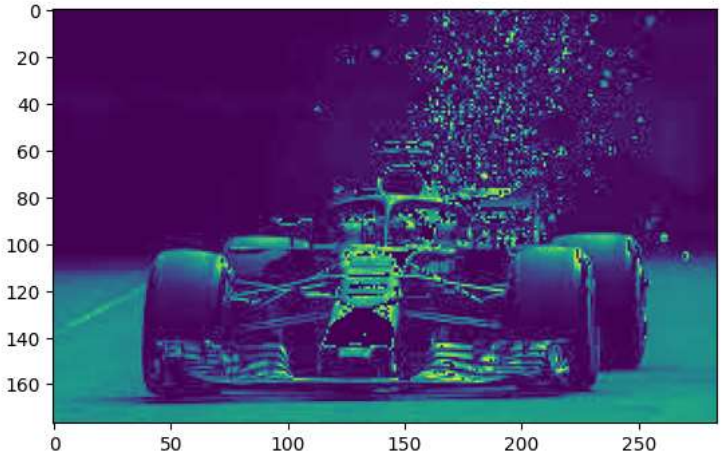

```
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c9422fd30>
```



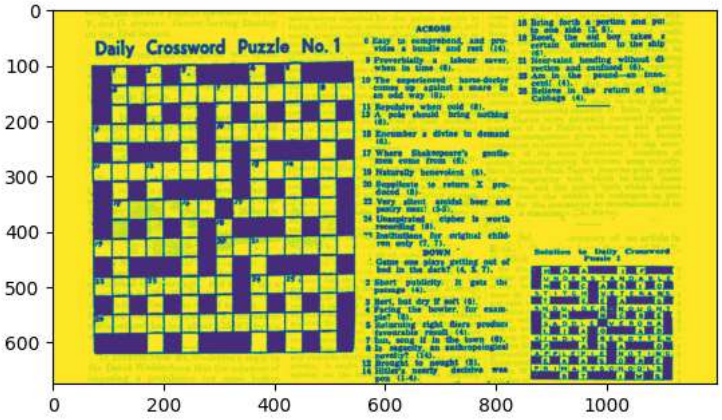
```
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c9411c9a0>
```



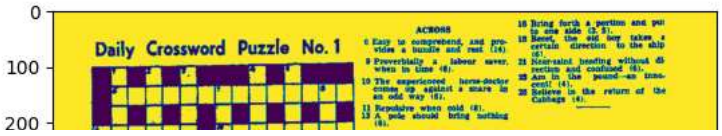
```
img2=cv2.imread('/content/crosswordimage.jpg',0)

plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7a2c854ad960>
```

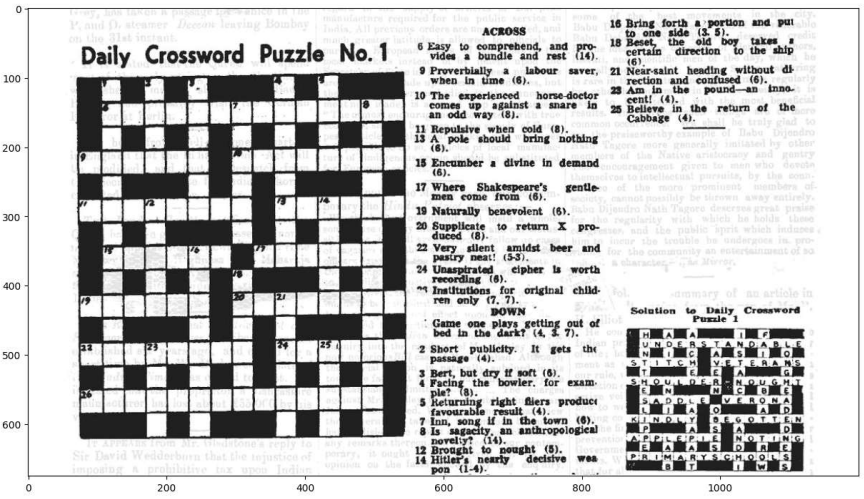



```
# binary threshold
threshold,converted_image = cv2.threshold(img2,127,255,cv2.THRESH_BINARY)
plt.imshow(converted_image)
```

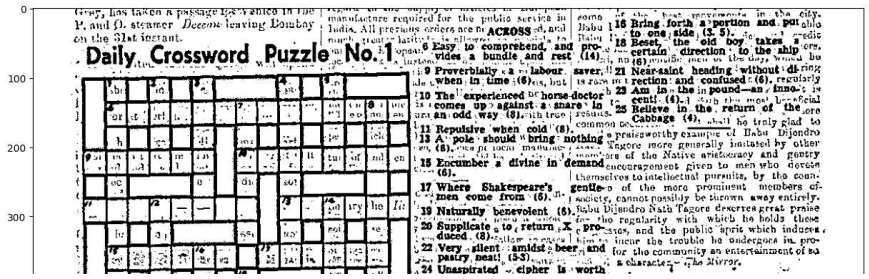
```
<matplotlib.image.AxesImage at 0x7a2c857170d0>
```



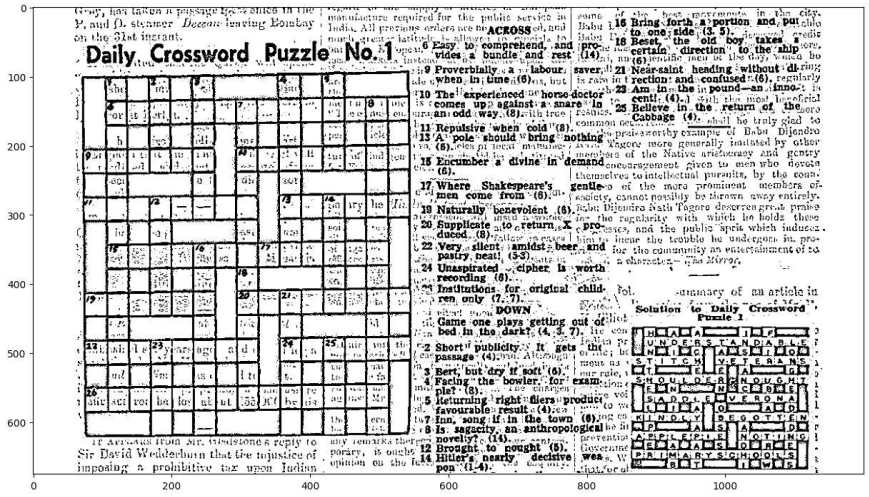

```
def show_pic(img2):
  fig=plt.figure(figsize=(15,15))
  ax=fig.add_subplot(111)
  ax.imshow(img2,cmap='gray')
show_pic(img2)
```




```
# adaptive threshold / cv2.threshold -> uses only grayscale
img3=cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,8)
show_pic(img3)
```



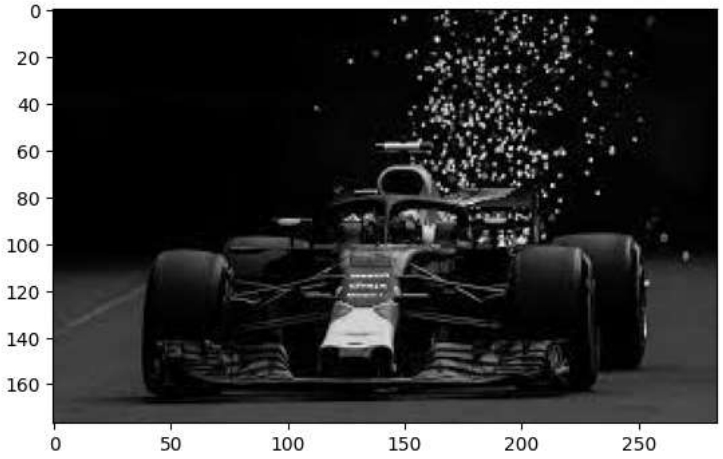

```
img4=cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,8)
show_pic(img4)
```




```
# BLURRING AND SMOOTHING
# Image Filtering -> Filter is a small matrix that is used to sharpen or blur our original image.
# - Convolution operation ( mathematical operation )
# - Used in image processing to combine a filter (kernel) with an image in order to create a modified image that highlights the specific
```

```
# Thresholding -> Converting an image to consist of only two values ( white or black)
# -> Used in Segmentation specifically
# -> Used when we are looking for shapes or edges

# -> 3 types of  Thresholding Techniques :
# -> Simple - uses cv.threshold to apply thresholding
# -> Adaptive
# -> Ostu's

# i) SIMPLE THRESHOLDING -> Threshold_binary - if the pixel intensity is > then set threshold to 255
#                         - if the pixel value is < then set threshold to 0

#                         -> thresh_binary_inv - opposite of threshold binary

#                         -> thresh_trunc - pixel intensity value is greater than truncated to the threshold. the pixel value is set to be
#                         - all other values are set to be the same

#                         -> thresh_tozero - pixel intensity is set to 0, for all the pixel intensity less than the threshold value

#                         -> thresh_tozero_inv - inverted/opposite case of thresh_tozero

# SYNTAX : cv2.threshold(source,thresholdValue,maxVal,thresholding Technique)
# source : the input image
# threshold value : example 100
# maxVal : example :255
# thresholding technique : the type of thresholding


# ADAPTIVE THRESHOLDING  - block_size should be an odd number ( greater value change it to 255 and lesser value to 0)
#                         - any simple thresholding technique can be applied here

# SYNTAX : cv2.adaptive threshold(image,255,cv2.adaptive_thresh_mean_c,cv2.thresh_binary,block_size,constant)

# constant - even number ( most of the cases it is zero [0] )
# threshold value = calculated mean - constant
# example : 114 - 4 = 110 ( 110 is the threshold value )

# block_size = size of the neighborhood used for calculating the local mean

# thresh_binary - pixels above threshold white and below threshold black

# 0 - black color
# 255 - white color

# adaptive_thresh_mean_c - calculates the threshold based on the mean of the pizel values in the neigbourhood


# Day 7
# adding the 0 flag to read it in black and white
import cv2
import matplotlib.pyplot as plt
img=cv2.imread('/content/randomimage.jpeg',0)


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

```
<matplotlib.image.AxesImage at 0x7a2c94835a50>
```



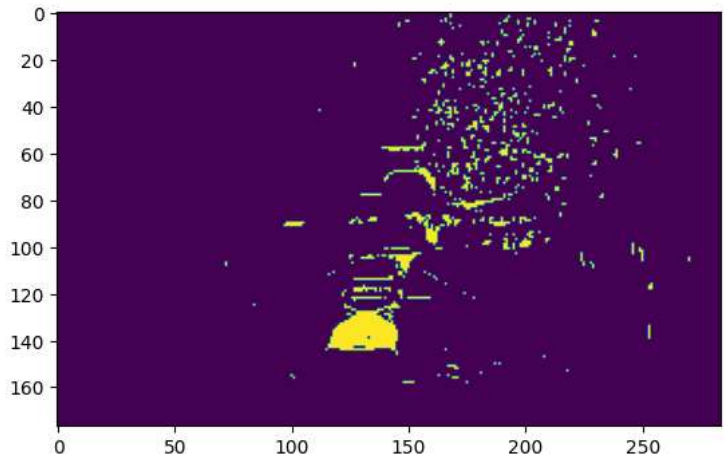

```
# Binary - Note : cv2 will return 2 vales one is threshold and then it is converted_image
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

```
threshold
```

```
127.0
```

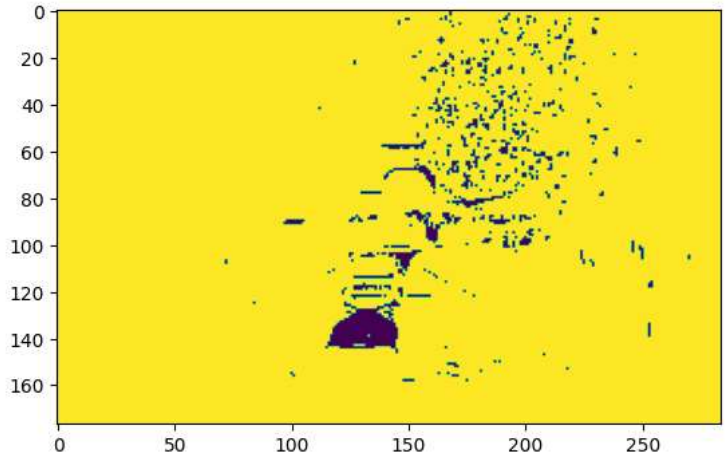
```
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c945927d0>
```



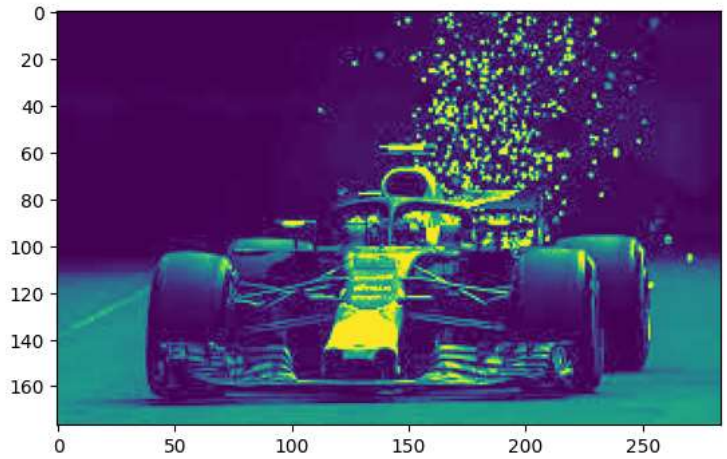
```
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c945ee4d0>
```



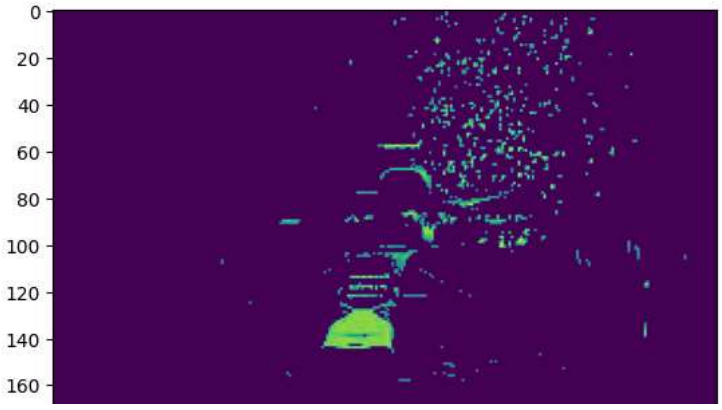
```
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c865114b0>
```



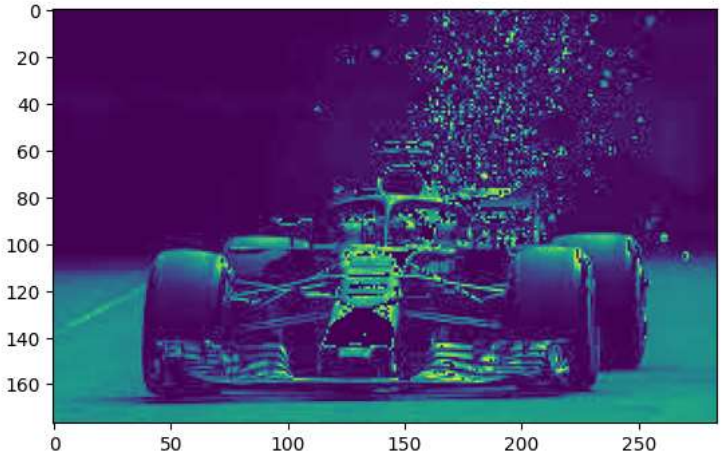
```
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c9422fd30>
```



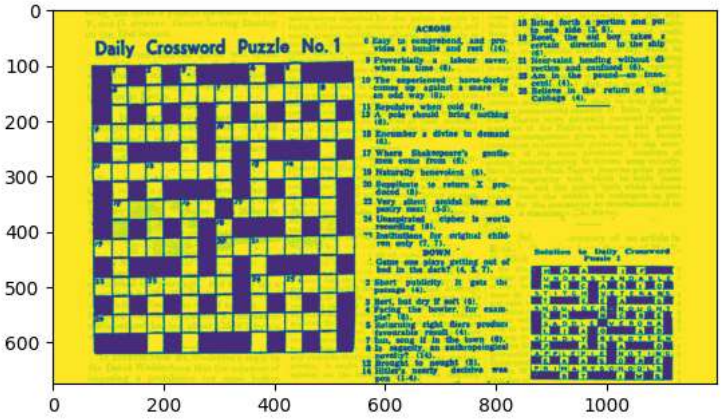
```
threshold,converted_image = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c9411c9a0>
```



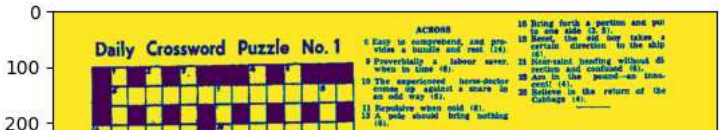
```
img2=cv2.imread('/content/crosswordimage.jpg',0)

plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7a2c854ad960>
```

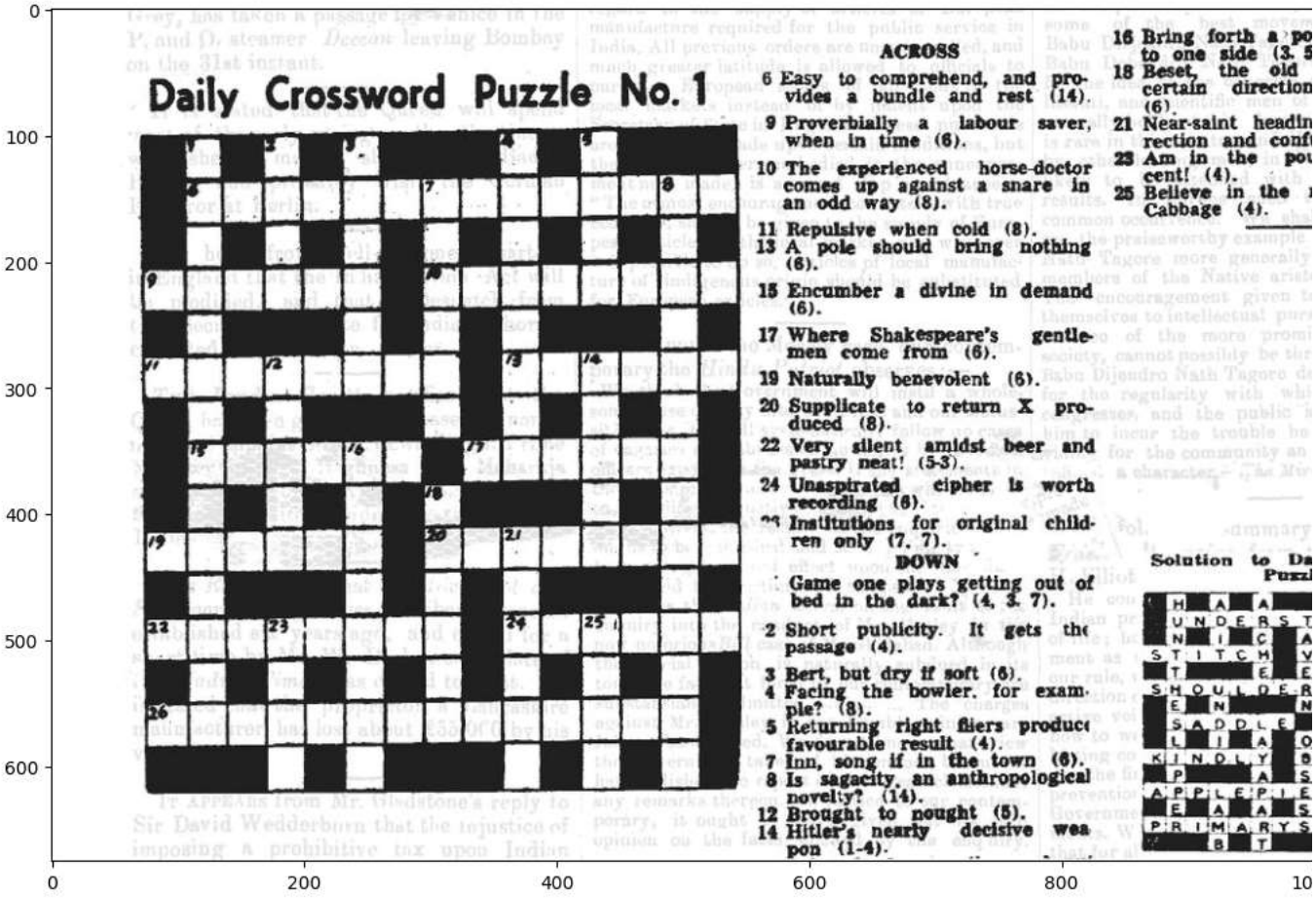



```
# binary threshold
threshold,converted_image = cv2.threshold(img2,127,255,cv2.THRESH_BINARY)
plt.imshow(converted_image)
```

```
<matplotlib.image.AxesImage at 0x7a2c857170d0>
```

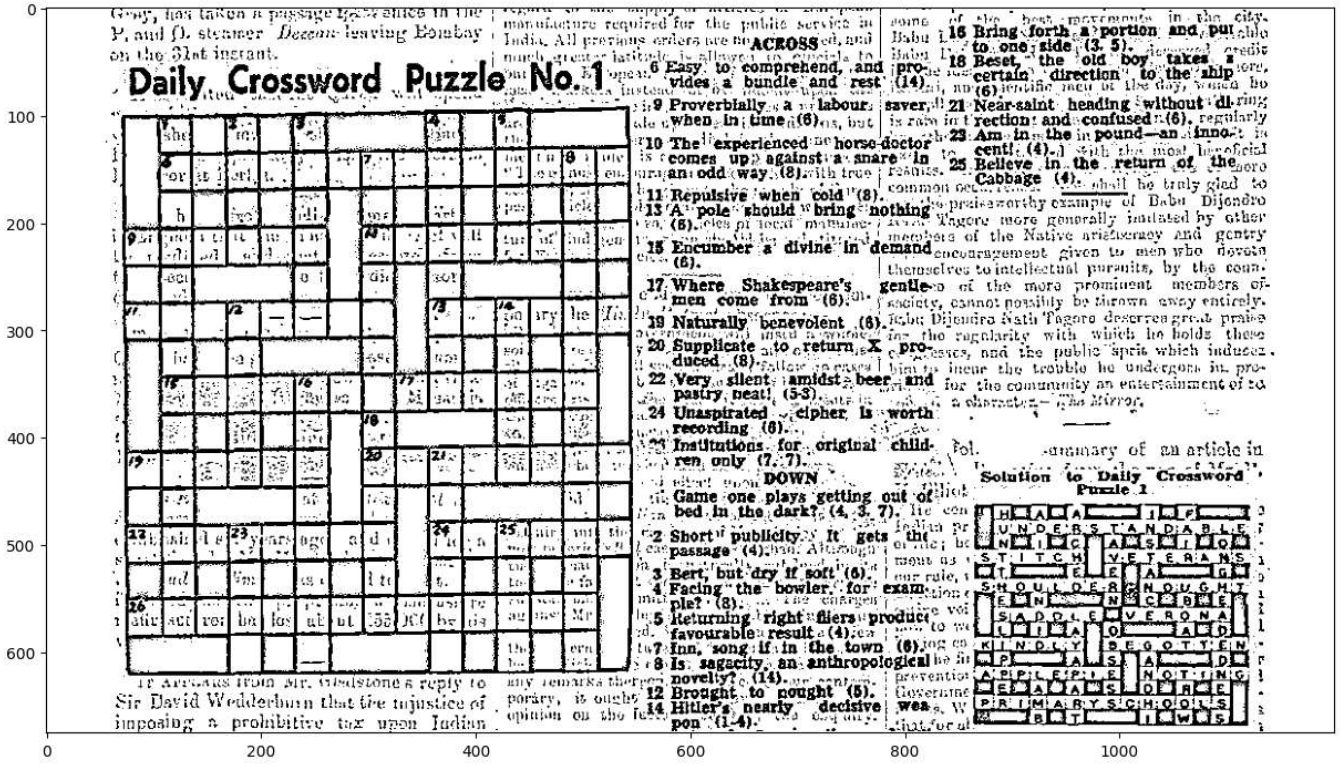



```
def show_pic(img2):
  fig=plt.figure(figsize=(15,15))
  ax=fig.add_subplot(111)
  ax.imshow(img2,cmap='gray')
show_pic(img2)
```




```
# adaptive threshold / cv2.threshold -> uses only grayscale
img3=cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,8)
show_pic(img3)
```


```
img4=cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,8)
show_pic(img4)
```

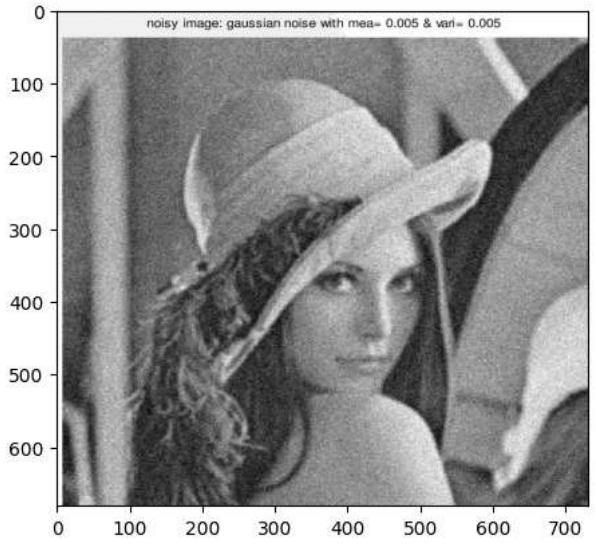



```
# BLURRING AND SMOOTHING
# Image Filtering -> Filter is a small matrix that is used to sharpen or blur our original image.
# - Convolution operation ( mathematical operation )
# - Used in image processing to combine a filter (kernel) with an image in order to create a modified image that highlights the specific
```

```
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

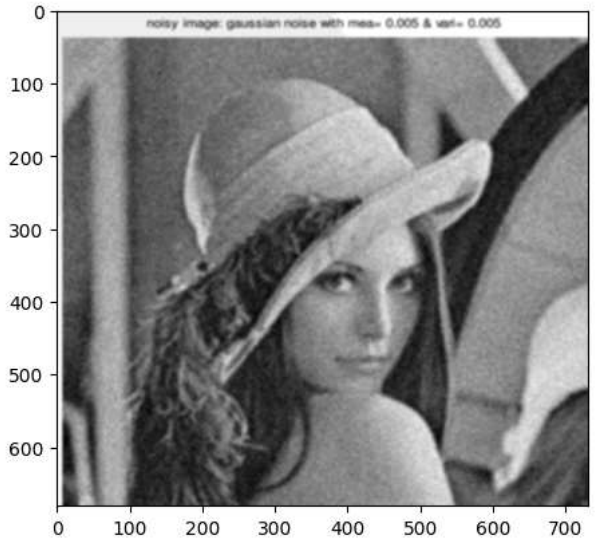
```
img=cv2.imread('/content/noisyimage.png',0)
plt.imshow(img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7da4678419c0>
```



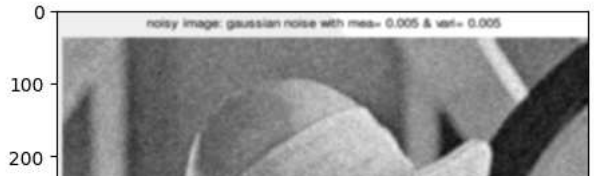

```
blurred_img=cv2.blur(img,ksize=(5,5))
plt.imshow(blurred_img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7da4672bfe80>
```



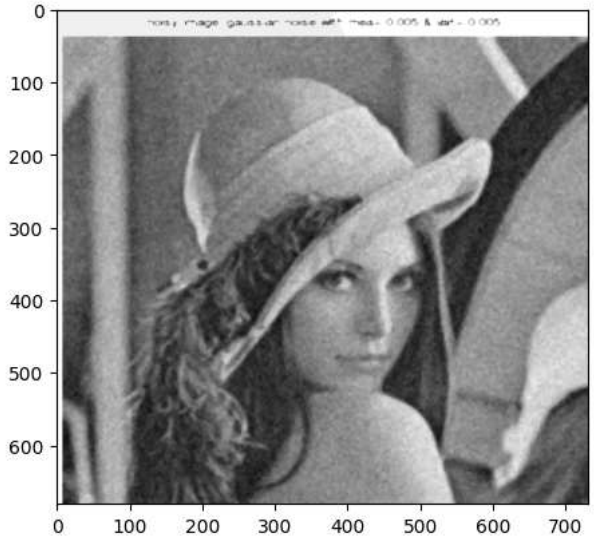

```
blurred_img=cv2.GaussianBlur(img,(5,5),10)
plt.imshow(blurred_img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7da46730e9b0>
```



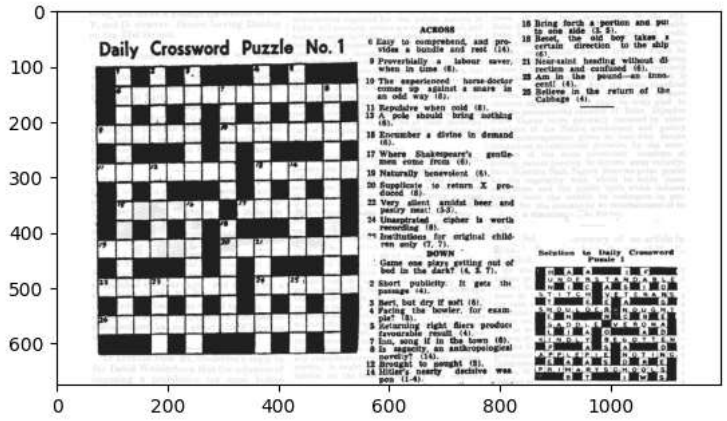

```
blurred_img=cv2.medianBlur(img,5)
plt.imshow(blurred_img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7da4669d1db0>
```



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img=cv2.imread('/content/crosswordimage.jpg',cv2.IMREAD_GRAYSCALE)
kernel=np.ones((5,5),np.uint8) # structure element
erosion=cv2.erode(img,kernel,iterations=1)
plt.imshow(img,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f3afc1748e0>
```

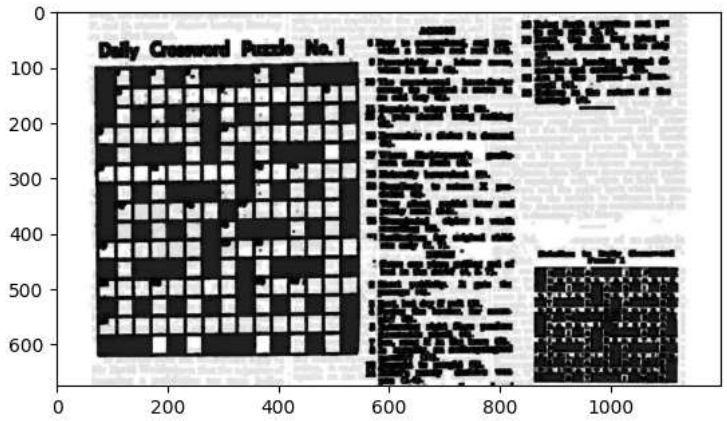



```
# a=np.array([0,1,0,1,1,1,0,1,0])
# kernel=a.reshape(3,3)
```

```
array([[0, 1, 0],
       [1, 1, 1],
       [0, 1, 0]])
```

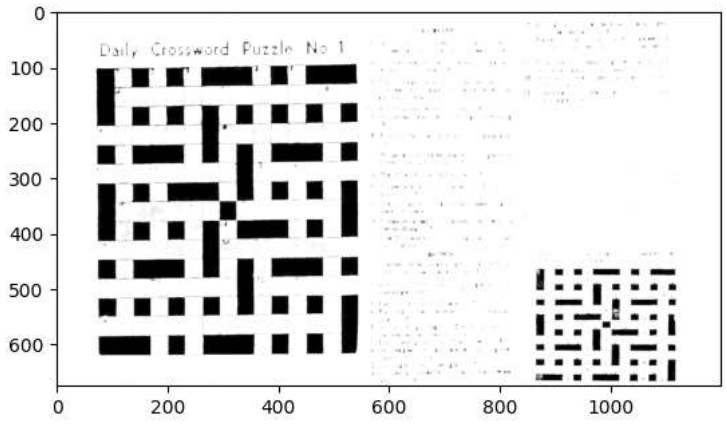
```
plt.imshow(erosion,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f3afc1dd300>
```



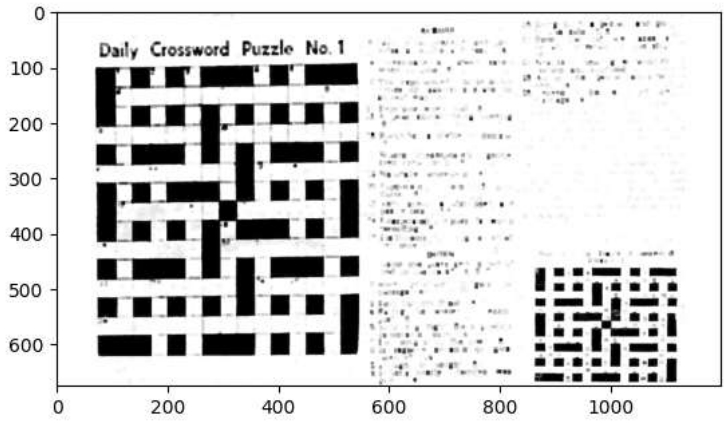

```
dilation=cv2.dilate(img,kernel,iterations=1)
plt.imshow(dilation,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f3afc24f160>
```



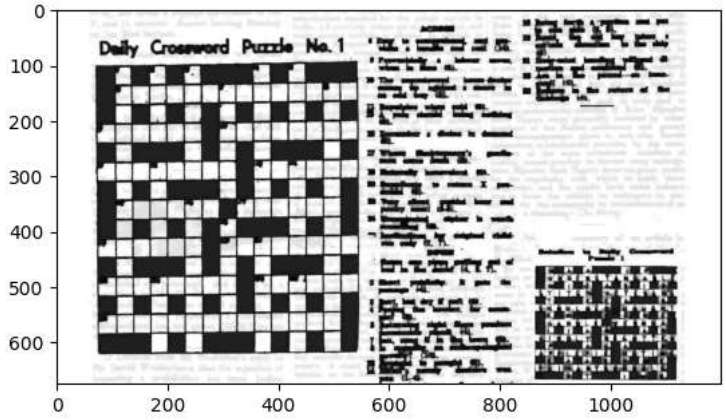
```
closing=cv2.morphologyEx(img,cv2.MORPH_CLOSE,kernel)
plt.imshow(closing,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f3afc0e0520>
```



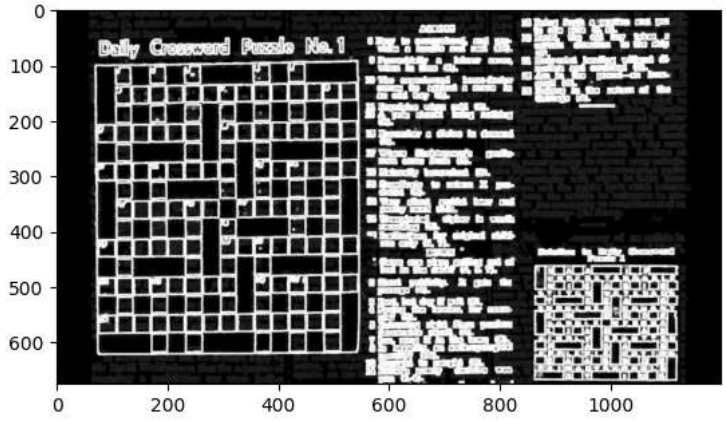

```
opening=cv2.morphologyEx(img,cv2.MORPH_OPEN,kernel)
plt.imshow(opening,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f3afc14dc60>
```




```
gradient=cv2.morphologyEx(img,cv2.MORPH_GRADIENT,kernel)
plt.imshow(gradient,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f3afbfc3a30>
```

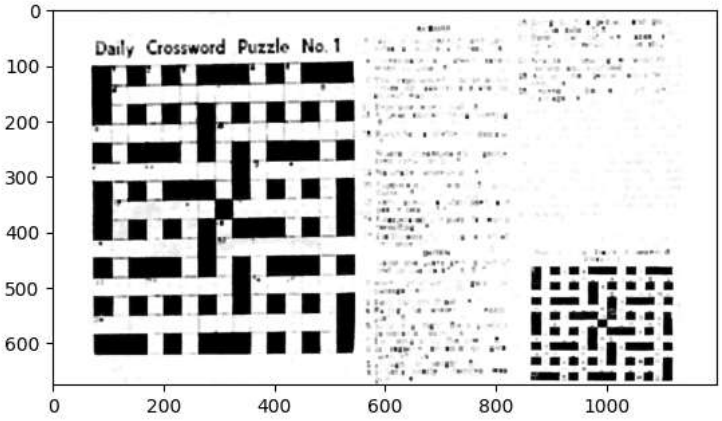



```
tophat=cv2.morphologyEx(img,cv2.MORPH_TOPHAT,kernel)
plt.imshow(tophat,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f3afc0509d0>
```

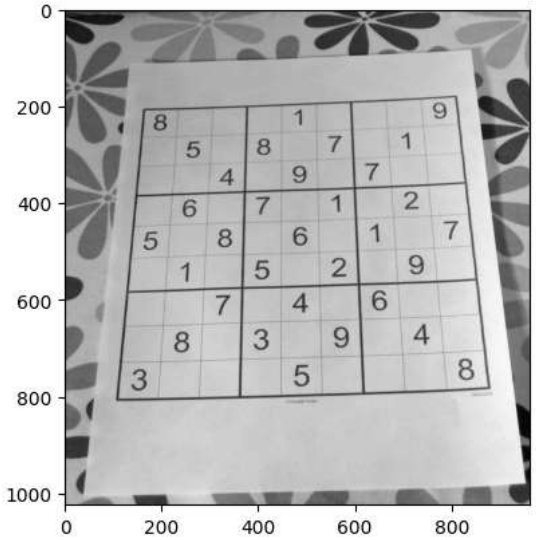
```
blackhat=cv2.morphologyEx(img,cv2.MORPH_BLACKHAT,kernel)
plt.imshow(closing,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f3afbec0f40>
```



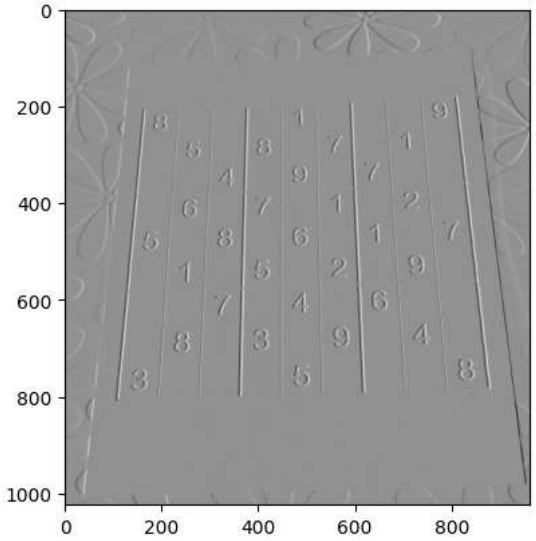
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load an image
image_path = '/content/sudoku.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Apply the Sobel filter to compute gradients
gradient_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
gradient_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)

# Compute gradient magnitude
gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)

plt.imshow(image,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7969d5e33040>
```



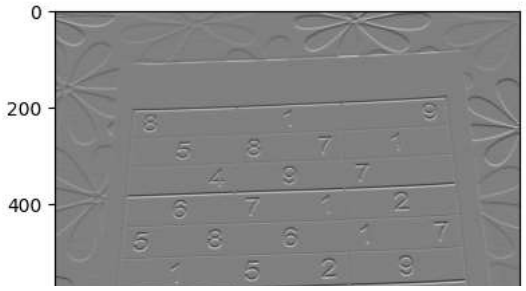

```
plt.imshow(gradient_x,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7969d5f9c070>
```




```
plt.imshow(gradient_y,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7969d5ef8040>
```

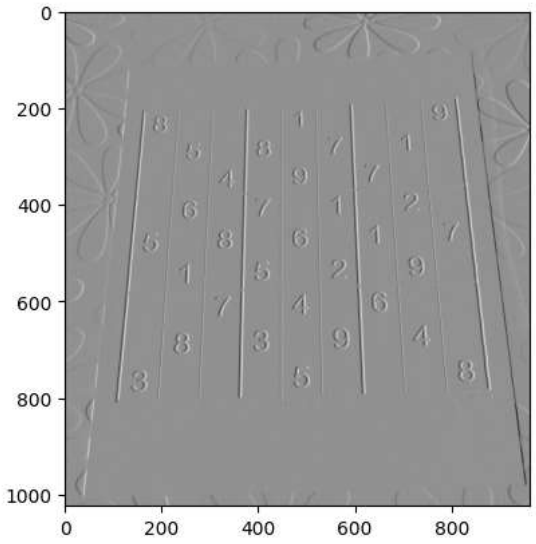


```
gradient_orientation = np.arctan2(gradient_y, gradient_x) * (180 / np.pi)
```

```
array([[ 8.06225775,  8.06225775,  9.89949494, ...,  9.        ,
         5.38516481,  6.70820393],
       [ 8.06225775,  8.06225775,  9.89949494, ...,  9.        ,
         5.38516481,  6.70820393],
       [ 5.        ,  5.        ,  5.83095189, ..., 18.        ,
         7.61577311,  5.09901951],
       ...,
       [ 1.        ,  1.        ,  5.83095189, ..., 13.45362405,
         7.        , 16.15549442],
       [ 5.83095189,  5.83095189,  7.61577311, ..., 13.89244399,
         4.47213595, 13.        ],
       [ 5.38516481,  5.38516481, 13.        , ...,  6.40312424,
         5.83095189,  7.28010989]])
```

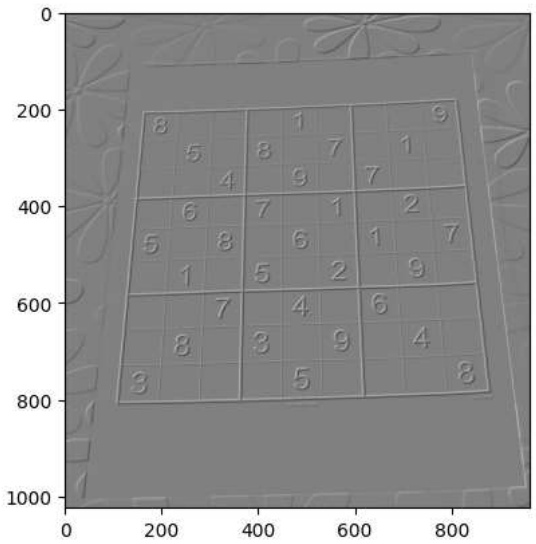
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load an image
image_path = '/content/sudoku.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Define the Prewitt kernels for gradient calculation
kernel_x = np.array([[-1, 0, 1],
                     [-1, 0, 1],
                     [-1, 0, 1]])

kernel_y = np.array([[-1, -1, -1],
                     [0, 0, 0],
                     [1, 1, 1]])

# Apply the Prewitt kernels to compute gradients
gradient_x = cv2.filter2D(image, cv2.CV_64F, kernel_x)
gradient_y = cv2.filter2D(image, cv2.CV_64F, kernel_y)

# Compute gradient magnitude
gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)

# Display the original image and gradient magnitude
plt.imshow(gradient_x,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7969d2e0e7a0>
```



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

# Load an image
image_path = '/content/sudoku.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Define the Roberts Cross kernels for gradient calculation
kernel_x = np.array([[1, 0],
                     [0, -1]])

kernel_y = np.array([[0, 1],
                     [-1, 0]])

# Apply the Roberts Cross kernels to compute gradients
gradient_x = cv2.filter2D(image, cv2.CV_64F, kernel_x)
gradient_y = cv2.filter2D(image, cv2.CV_64F, kernel_y)

# Compute gradient magnitude
gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)

# Display the original image and gradient magnitude
plt.imshow(gradient_x,cmap="gray")
```

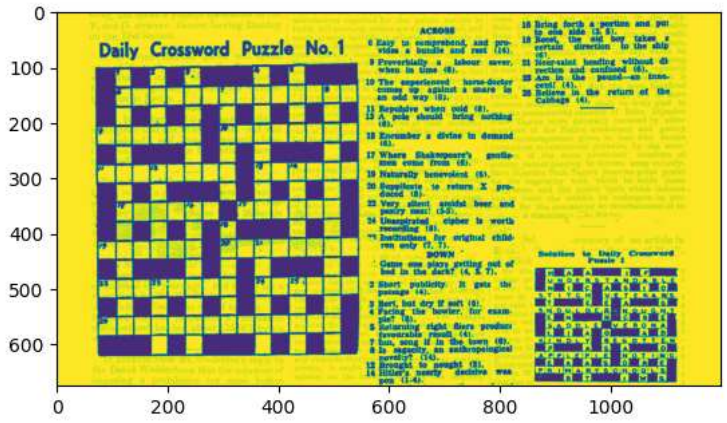
```
<matplotlib.image.AxesImage at 0x7969d2ef3130>
```



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

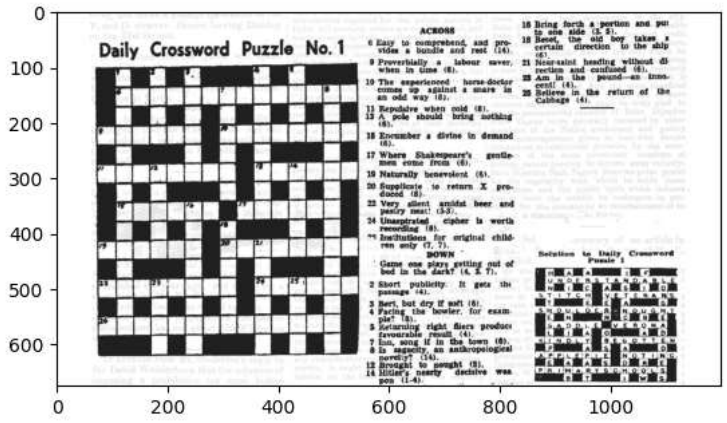
```
img=cv2.imread('/content/crosswordimage.jpg',cv2.IMREAD_GRAYSCALE)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7d790b7b9240>
```



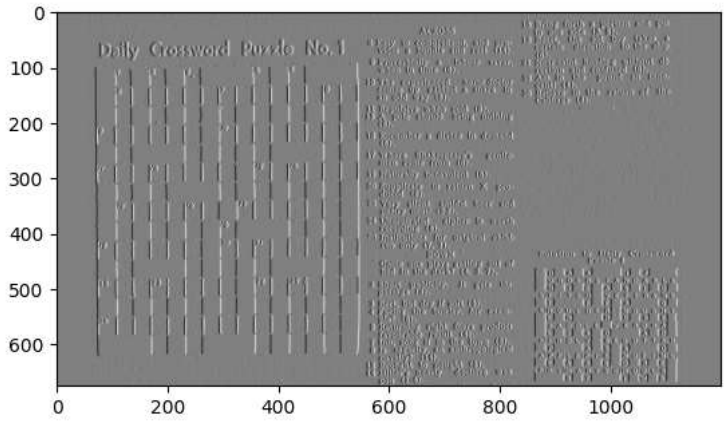

```
gradient_x=cv2.Sobel(img,cv2.CV_64F,1,0,ksize=3)  # this is for x_gradeint that is 1,0
gradient_y=cv2.Sobel(img,cv2.CV_64F,0,1,ksize=3)  # this is for y_gradeint that is 0,1

gradient_magnitude=np.sqrt(gradient_x**2+gradient_y**2)
plt.imshow(img,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7d790b69e050>
```



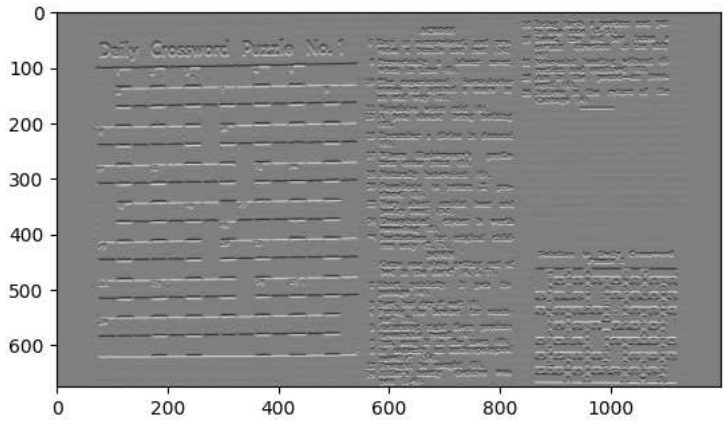

```
plt.imshow(gradient_x,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7d790b8afd30>
```



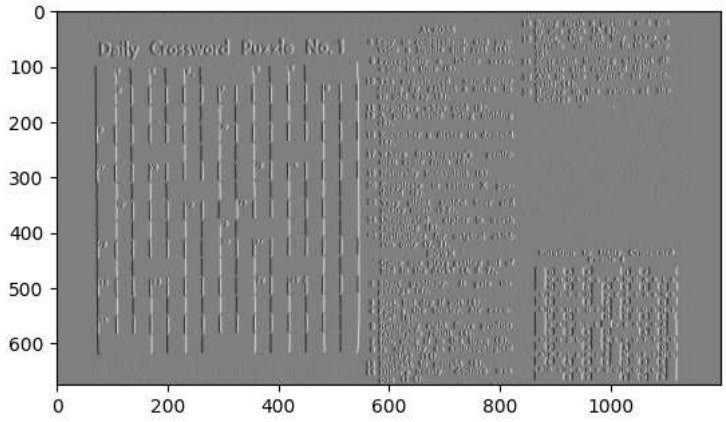

```
plt.imshow(gradient_y,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7d790b724fd0>
```



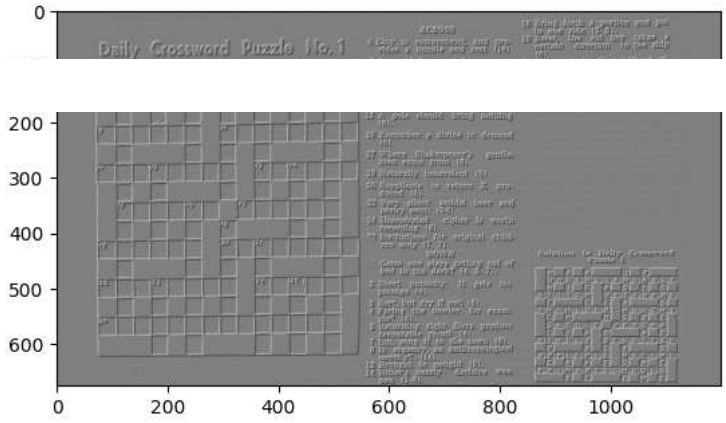

```
gradient_orientation=np.arctan2(gradient_y,gradient_x)*(180/np.pi)


# Define the Prewitt kernels for gradient calculation
kernel_x = np.array([[-1, 0, 1],
                     [-1, 0, 1],
                     [-1, 0, 1]])

kernel_y = np.array([[-1, -1, -1],
                     [0, 0, 0],
                     [1, 1, 1]])

gradient_x=cv2.filter2D(img,cv2.CV_64F,kernel_x)
gradient_y=cv2.filter2D(img,cv2.CV_64F,kernel_y)
gradient_magnitude=np.sqrt(gradient_x**2+gradient_y**2)
plt.imshow(gradient_x,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7d790b763a60>
```




```
# Define the Roberts Cross kernels for gradient calculation
kernel_x = np.array([[1, 0],
                     [0, -1]])

kernel_y = np.array([[0, 1],
                     [-1, 0]])

gradient_x=cv2.filter2D(img,cv2.CV_64F,kernel_x)
gradient_y=cv2.filter2D(img,cv2.CV_64F,kernel_y)
gradient_magnitude=np.sqrt(gradient_x**2+gradient_y**2)
plt.imshow(gradient_x,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7d7909064c40>
```

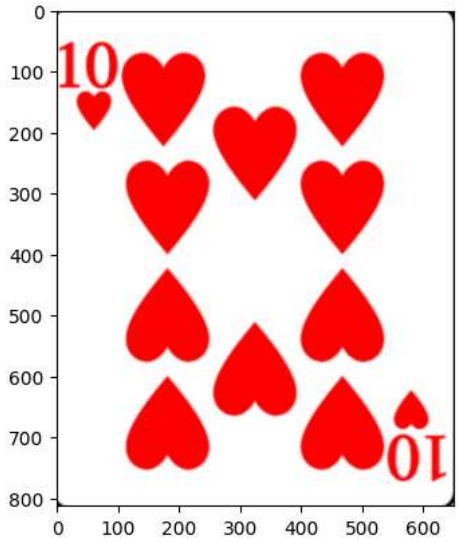


MULTIPLE LOCATION MATCH

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
%matplotlib inline
```

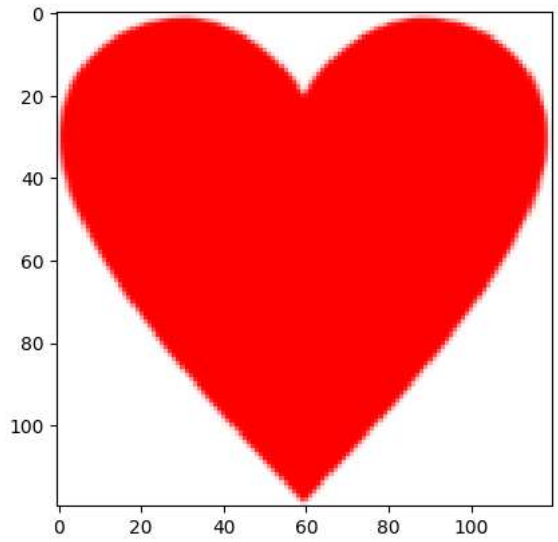
```
img=cv2.imread('/content/image.png')
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7962e3fbfb50>
```




```
img1=cv2.imread('/content/template.png')
img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
img1=cv2.resize(img1,(120,120))
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7962e3e3ff10>
```




```
result=cv2.matchTemplate(img,img1,cv2.TM_CCOEFF_NORMED)
```

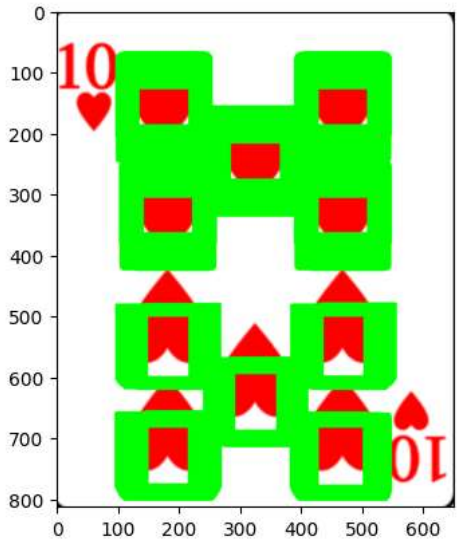
```
threshold=0.5
# define a threshold
# find locations where the result is above the threshold
locations=np.where(result>=threshold)
locations
```

```
(array([ 67,  67,  67, ..., 682, 682, 682]),
 array([109, 110, 111, ..., 419, 420, 421]))
```

```
for loc in zip(*locations[::-1]):
  cv2.rectangle(img,loc,(loc[0]+img1.shape[1],loc[1]+img1.shape[0]),(0,255,0),2)
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7962e3eaf460>
```



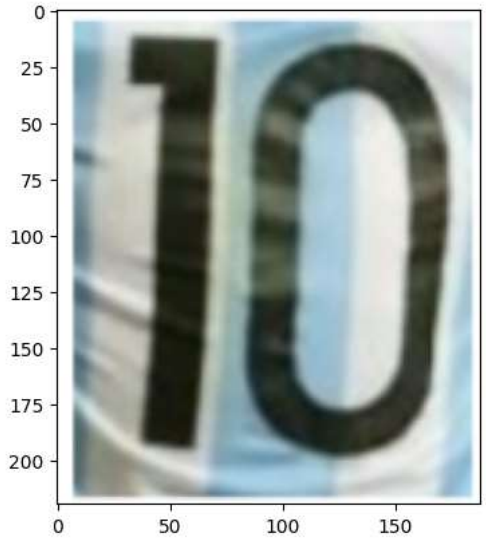
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
%matplotlib inline
```

```
img=cv2.imread('/content/image.png')
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x78adfa636cb0>
```


```
img1=cv2.imread('/content/template.png')
img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x78adf9c84340>
```




```
height,width,channels=img1.shape # template image
height
```

```
220
```

```
width
```

```
188
```

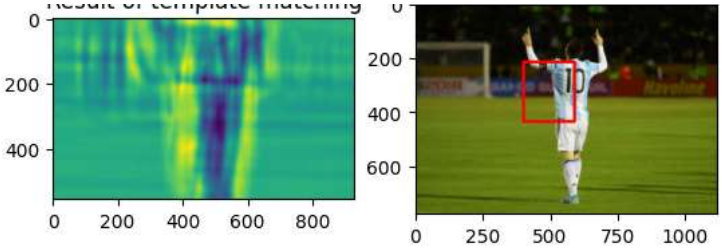
```
channels
```

```
3
```

```
#Eval function ()  --->>> these brackets are called Eval function
#It will convert keyword to function
#Eval takes the already available functions like max ,min and other sort of in build python functions
```

```
# all the 6 methods for comparing in the list
methods = ['cv2.TM_CCOEFF','cv2.TM_CCOEFF_NORMED','cv2.TM_CCORR','cv2.TM_CCORR_NORMED','cv2.TM_SQDIFF','cv2.TM_SQDIFF_NORMED']


for m in methods:
  # Create a copy of the image
  img_copy=img.copy()
  # get actual function instead of string
  method=eval(m)
  res=cv2.matchTemplate(img_copy,img1,method)
  # grab their min and max values, plus their locations
  min_val,max_val,min_loc,max_loc=cv2.minMaxLoc(res)
   # Set up drawing of the rectangle
   # If the method is TM_SQDIFF or TM_SQFIFF_NORMED, take minimum
   # Notice the coloring on the last 2 left hand side images
  if method in [cv2.TM_SQDIFF,cv2.TM_SQDIFF_NORMED]:
    top_left=min_loc
  else:
    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(img_copy,top_left,bottom_right,255,10) # 255 is the color of the rectangle and 10 is the thickness of the rectangle

  plt.subplot(121)
  plt.imshow(res)
  plt.title('Result of template matching')
  plt.subplot(122)
  plt.imshow(img_copy)
  plt.title('detected Point')
  plt.suptitle(m)
  plt.show()
  print('\n')
  print('\n')
```

cv2.TM_CCOEFF

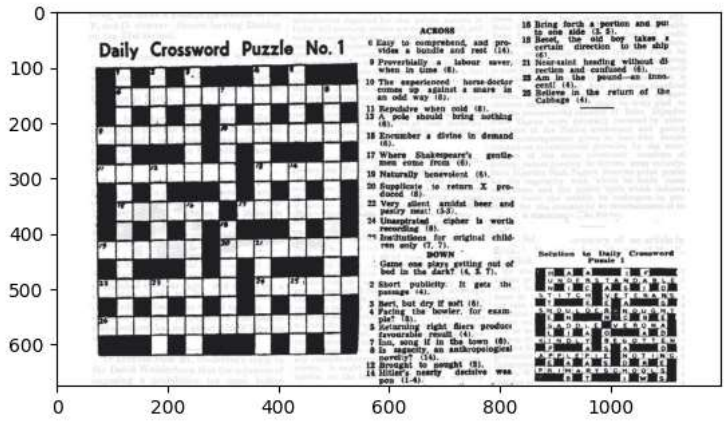


cv2.TM_CCOEFF_NORMED

detected Point

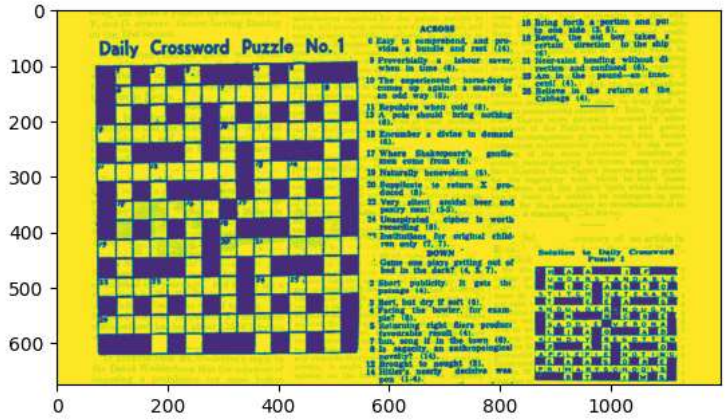
```
import cv2
import numpy as np
import matplotlib.pyplot as plt


image=cv2.imread('/content/crosswordimage.jpg')
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x78fbca6a8bb0>
```



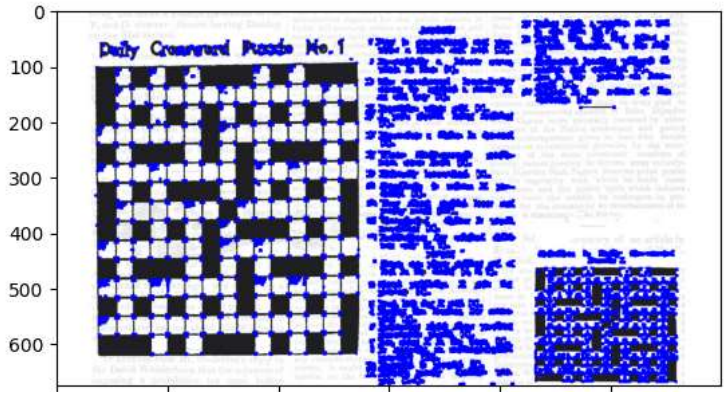

```
# grayscale color space
operatedImage=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
plt.imshow(operatedImage)
```

```
<matplotlib.image.AxesImage at 0x78fbca683880>
```



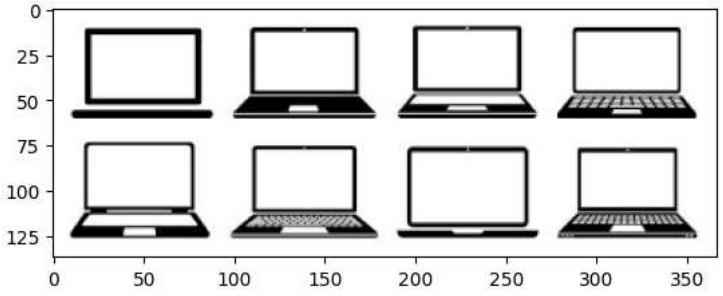

```
# modify the data type
# setting to 32 bit floating point
operatedImage = np.float32(operatedImage)
# 32 bit floats can provide sufficient precision
# without consuming excessive memory



# apply the cv2.cornerHarris method
# to detect the corners with appropriate
# Values as input parameters
# (Gray scale img,neighbourhood,kernel sire and k value)
# k=0.07 make the algorithm more sensitive to corners,which#you can adjust this parameter based on the characteristics
# and the desired corner detection results#Results are marked through the dilated corners
dest=cv2.cornerHarris(operatedImage,2,5,0.07)


# resuls are marked through the dilated corners
# increase the bright area portions
dest = cv2.dilate(dest,None)


# reverting back to the original image,
# with optimal threshold value
image[dest > 0.01 * dest.max()]=[0,0,255]


plt.imshow(image,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x78fbb8b31ff0>
```



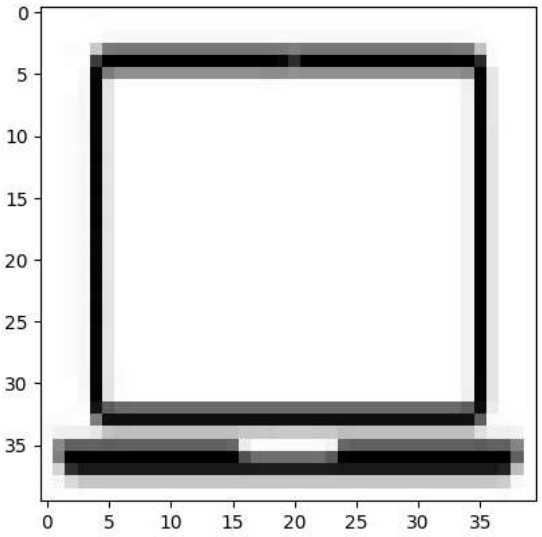
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
%matplotlib inline


img=cv2.imread('/content/laptop.jpg')
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7a2fcb90ddb0>
```




```
img1=cv2.imread('/content/temp.png')
img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
img1=cv2.resize(img1,(40,40))
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7a2fcb97fc70>
```

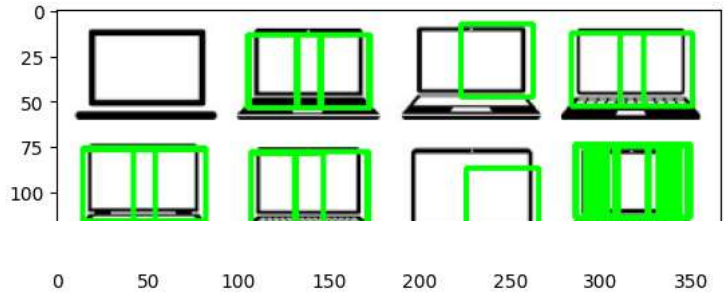



```
 result=cv2.matchTemplate(img,img1,cv2.TM_CCOEFF_NORMED)


threshold=0.5
# define a threshold
# find locations where the result is above the threshold
locations=np.where(result>=threshold)
locations
```

```
(array([ 8, 13, 13, 14, 14, 14, 14, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74,
        74, 74, 74, 74, 74, 74, 74, 74, 75, 75, 76, 76, 77, 77, 78, 78, 78,
        79, 87]),
 array([223, 284, 311, 105, 106, 132, 133, 286, 287, 292, 293, 294, 295,
        296, 297, 298, 299, 300, 301, 302, 303, 304, 308, 309, 310, 287,
        309,  14,  42,  14,  42, 107, 131, 132, 107, 226]))
```

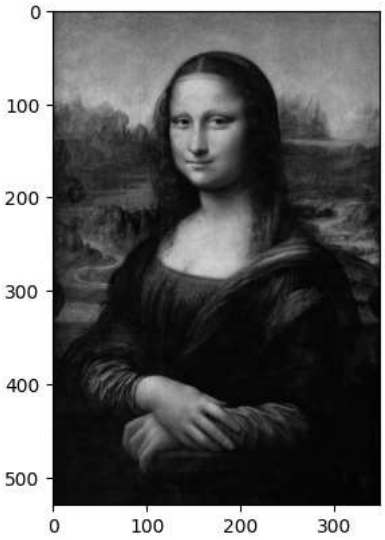
```
for loc in zip(*locations[::-1]):
  cv2.rectangle(img,loc,(loc[0]+img1.shape[1],loc[1]+img1.shape[0]),(0,255,0),2)



plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7a2fcb824190>
```



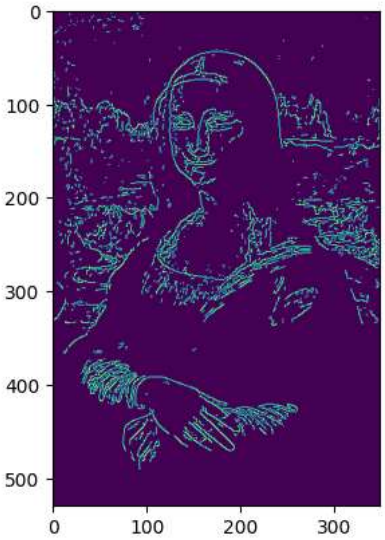
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


image=cv2.imread('/content/grayscaleimg.jpg')
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7ebce84ebee0>
```




```
# threshold 1 below the threshold value do not consider it as edges
# threshold 2 above the threshold value do consider it as edges

# threshold 1 : this is the lower threshold for edge detection.
# this is used to identify weak edges in the image.
# Pixels with the gradient values below this threshold are not considered edges and are suppressed

# threshold2 : this is the higher threshold for edge detection.
# this is used to identify strongs edges in the image.
# Pixels with the gradient values above this threshold are stromg edges.

edges = cv2.Canny(image=image,threshold1=100,threshold2=120)
plt.imshow(edges)
```

```
<matplotlib.image.AxesImage at 0x7ebce82e9210>
```




```
# calculate the median pixel value
med_val=np.median(image)
med_val
```

```
36.0
```

```
lower=0.30*med_val
lower
```

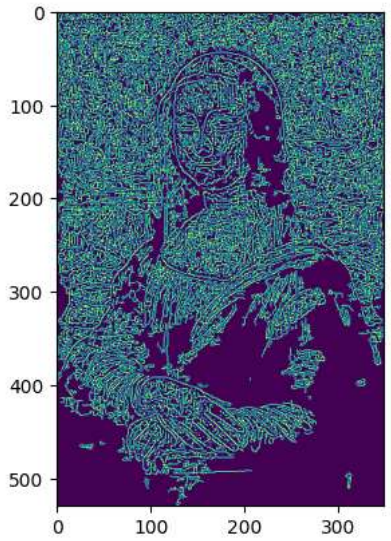
```
10.799999999999999
```

```
upper=1.20*med_val
upper
```

```
43.199999999999996
```

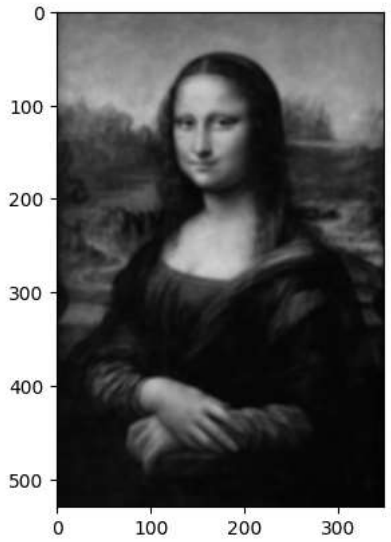
```
edges = cv2.Canny(image=image,threshold1=lower,threshold2=upper)
plt.imshow(edges)

# edges = cv2.Canny(image=image,lower,upper)
```

```
<matplotlib.image.AxesImage at 0x7ebce3f19ff0>
```



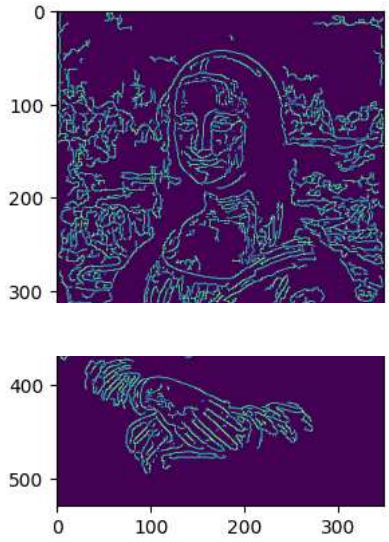

```
blurred_img=cv2.blur(image,ksize=(5,5))
plt.imshow(blurred_img)
```

```
<matplotlib.image.AxesImage at 0x7ebce8745d50>
```



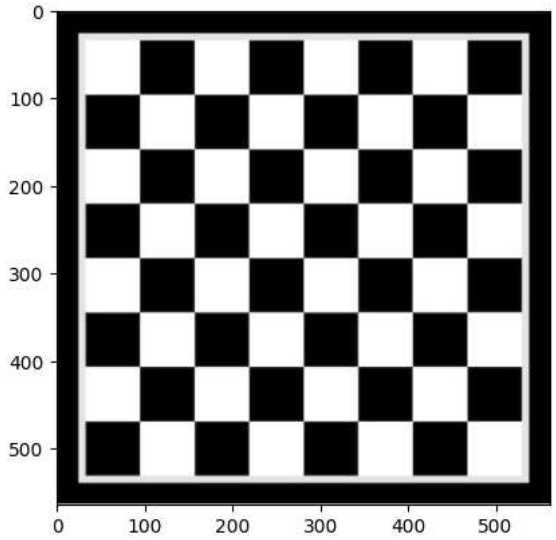

```
edges = cv2.Canny(image=blurred_img,threshold1=lower,threshold2=upper)
plt.imshow(edges)
```

```
<matplotlib.image.AxesImage at 0x7ebce93b7cd0>
```



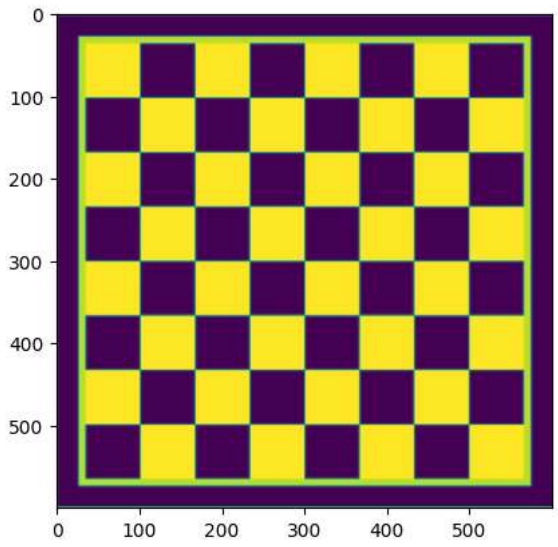
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


image1=cv2.imread('/content/chessboard.png')
plt.imshow(image1)
image1.shape
```

```
(565, 562, 3)
```



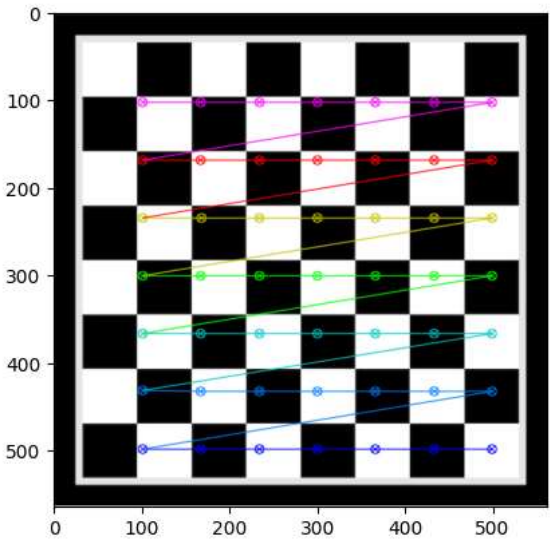

```
image=cv2.resize(image1,(600,600))
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
plt.imshow(gray)

image.shape
# for grid detection the height and width should be the same
```

```
(600, 600, 3)
```



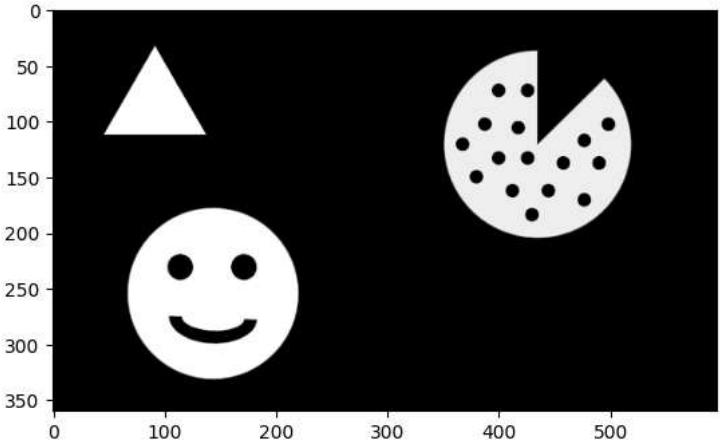

```
# define the size of the grid ( number of inner croners to be found )
chess_size = (7,7)

# find the corners of the chessboard
# ret-> Return boolean ; corner avaliable or not(true or false)
#Corners ->Returns values

#If corners are found
# Draw circles at the corners and display
ret,corners = cv2.findChessboardCorners(gray,chess_size,None)


if ret:
  cv2.drawChessboardCorners(image1,chess_size,corners,ret)
  plt.imshow(image1)
```

```
else:
  print("achichoo")
```



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


image1=cv2.imread('/content/internal_external.png')
plt.imshow(image1)
image1.shape

# we can detect the border of an object
```

```
(360, 596, 3)
```



Keras Basics

```
# Variance of wavelet Transformed image ( continuous )
# Skewness of wavelet Transformed image ( continuous )
# Curtosis of wavelet Transformed image ( continuous )
# Entropy of image ( continuous )
# Class ( integer )
```

```
from numpy import genfromtxt
df=genfromtxt('/content/bank_note_data.txt',delimiter=',')
df
```

```
array([[  3.6216 ,   8.6661 ,  -2.8073 ,  -0.44699,   0.     ],
       [  4.5459 ,   8.1674 ,  -2.4586 ,  -1.4621 ,   0.     ],
       [  3.866  ,  -2.6383 ,   1.9242 ,   0.10645,   0.     ],
       ...,
       [ -3.7503 , -13.4586 ,  17.5932 ,  -2.7771 ,   1.     ],
       [ -3.5637 ,  -8.3827 ,  12.393  ,  -1.2823 ,   1.     ],
       [ -2.5419 ,  -0.65804,   2.6842 ,   1.1952 ,   1.     ]])
```

```
y=df[:,4]
y
```

```
array([0., 0., 0., ..., 1., 1., 1.])
```

```
x=df[:,0:4]
x
```

```
array([[  3.6216 ,   8.6661 ,  -2.8073 ,  -0.44699],
       [  4.5459 ,   8.1674 ,  -2.4586 ,  -1.4621 ],
       [  3.866  ,  -2.6383 ,   1.9242 ,   0.10645],
       ...,
       [ -3.7503 , -13.4586 ,  17.5932 ,  -2.7771 ],
       [ -3.5637 ,  -8.3827 ,  12.393  ,  -1.2823 ],
       [ -2.5419 ,  -0.65804,   2.6842 ,   1.1952 ]])
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=42)
```

```
x_train
```

```
array([[-0.8734  , -0.033118, -0.20165 ,  0.55774 ],
       [ 2.0177  ,  1.7982  , -2.9581  ,  0.2099  ],
       [-0.36038 ,  4.1158  ,  3.1143  , -0.37199 ],
       ...,
       [-7.0364  ,  9.2931  ,  0.16594 , -4.5396  ],
       [-3.4605  ,  2.6901  ,  0.16165 , -1.0224  ],
       [-3.3582  , -7.2404  , 11.4419  , -0.57113 ]])
```

```
x_test
```

```
array([[ 1.5691  ,  6.3465  , -0.1828  , -2.4099  ],
       [-0.27802 ,  8.1881  , -3.1338  , -2.5276  ],
       [ 0.051979,  7.0521  , -2.0541  , -3.1508  ],
       ...,
       [ 3.5127  ,  2.9073  ,  1.0579  ,  0.40774 ],
       [ 5.504   , 10.3671  , -4.413   , -4.0211  ],
       [-0.2062  ,  9.2207  , -3.7044  , -6.8103  ]])
```

```
y_train
```

```
array([1., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 1., 1., 0.,
       1., 1., 1., 0., 1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0.,
       1., 0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 0., 0.,
       0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1.,
       0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 0., 1., 0., 0., 0., 0., 1., 1., 1., 0., 0.,
       0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 1.,
       1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 1., 0.,
       0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 1., 1.,
       0., 1., 1., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0.,
       0., 1., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0.,
       1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
       0., 1., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 1., 1., 1., 0.,
       1., 0., 0., 0., 1., 0., 0., 1., 1., 1., 0., 1., 0., 1., 1., 0., 0.,
       1., 1., 1., 1., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 1., 0., 1.,
```

```
       1., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 1.,
       1., 1., 1., 1., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0., 0.,
       0., 0., 0., 1., 0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 1., 1., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0., 1., 0., 1., 0.,
       0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 1., 1.,
       0., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1.,
       0., 0., 0., 1., 0., 0., 1., 1., 0., 1., 0., 0., 1., 0., 1., 1., 1.,
       0., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 0., 0., 0., 0., 1., 0.,
       0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0.,
       0., 1., 0., 1., 1., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0.,
       1., 0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 1., 1., 1., 1.,
       0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1.,
       0., 1., 0., 1., 0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 1.,
       0., 1., 0., 1., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 1., 0.,
       0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0.,
       0., 1., 1., 0., 0., 0., 1., 1., 1., 1., 0., 1., 1., 0., 0., 0., 1.,
       1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 1., 1.,
       0., 1., 1., 1., 0., 1., 1., 0., 0., 0., 0., 1., 1., 1., 0., 1., 0.,
       0., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 1., 0., 0., 0., 1., 0.,
       0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0.,
       1., 0., 0., 1., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 1.,
       1., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 0., 0.,
       1., 0., 0., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 0.,
       0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0.,
       1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 0., 1., 0., 0.,
       0., 0., 0., 1., 0., 1., 1., 1., 0., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 1., 1., 0., 1., 0., 1.,
       0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 1., 0., 1., 1., 0.,
       1., 0., 1., 1., 0., 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 1.,
       1.])
```

```
y_test
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 1., 1.,
       1., 1., 1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1.,
       1., 0., 1., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 1., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0.,
       0., 1., 1., 0., 1., 0., 1., 0., 0., 1., 1., 1., 1., 0., 1., 0., 0.,
       0., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0., 0., 1., 0.,
       0., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 0., 0., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1.,
       0., 1., 0., 1., 0., 0., 0., 1., 1., 1., 1., 1., 0., 1., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 1.,
       1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 0., 0.,
       1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 0., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1.,
       0., 0., 0., 1., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1., 1.,
       0., 0., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0.,
       0., 0., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1.,
       1., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 1.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 1.,
       1., 0., 1., 1., 0., 1., 1., 0., 0., 1., 0., 0., 1., 0., 1., 0., 1.,
       1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0.,
       0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 0., 1.,
       0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1., 1., 1., 1.,
       0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0.,
       0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0.,
       0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0.])
```

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaled_train=scaler.fit_transform(x_train)
scaled_test=scaler.fit_transform(x_test)
```

```
scaled_test
```

```
array([[0.65576832, 0.76019778, 0.22784929, 0.52919178],
       [0.51021119, 0.83150383, 0.09609469, 0.51778828],
       [0.53621584, 0.78751834, 0.14430053, 0.45740888],
       ...,
       [0.80892829, 0.62703326, 0.28324337, 0.80218187],
       [0.96584712, 0.91587388, 0.03898168, 0.37308892],
       [0.51587076, 0.87148571, 0.07061886, 0.10285426]])
```

Building the network with Keras

```
from keras.models import Sequential
from keras.layers import Dense


model=Sequential()
model.add(Dense(4,input_dim=4,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
# sigmoid function to output 0 or 1


model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])


model.fit(scaled_train,y_train,epochs=100,verbose=1)
```

```
Epoch 1/100
29/29 [==============================] - 1s 3ms/step - loss: 0.7047 - accuracy: 0.4570
Epoch 2/100
29/29 [==============================] - 0s 2ms/step - loss: 0.6972 - accuracy: 0.4538
Epoch 3/100
29/29 [==============================] - 0s 3ms/step - loss: 0.6899 - accuracy: 0.4864
Epoch 4/100
29/29 [==============================] - 0s 2ms/step - loss: 0.6828 - accuracy: 0.5484
Epoch 5/100
29/29 [==============================] - 0s 2ms/step - loss: 0.6714 - accuracy: 0.5495
Epoch 6/100
29/29 [==============================] - 0s 2ms/step - loss: 0.6549 - accuracy: 0.5495
Epoch 7/100
29/29 [==============================] - 0s 2ms/step - loss: 0.6438 - accuracy: 0.5789
Epoch 8/100
29/29 [==============================] - 0s 2ms/step - loss: 0.6344 - accuracy: 0.6072
Epoch 9/100
29/29 [==============================] - 0s 2ms/step - loss: 0.6251 - accuracy: 0.6627
Epoch 10/100
29/29 [==============================] - 0s 3ms/step - loss: 0.6154 - accuracy: 0.6910
Epoch 11/100
29/29 [==============================] - 0s 2ms/step - loss: 0.6054 - accuracy: 0.7247
Epoch 12/100
29/29 [==============================] - 0s 2ms/step - loss: 0.5953 - accuracy: 0.7312
Epoch 13/100
29/29 [==============================] - 0s 2ms/step - loss: 0.5851 - accuracy: 0.7650
Epoch 14/100
29/29 [==============================] - 0s 3ms/step - loss: 0.5729 - accuracy: 0.7639
Epoch 15/100
29/29 [==============================] - 0s 2ms/step - loss: 0.5609 - accuracy: 0.7845
Epoch 16/100
29/29 [==============================] - 0s 2ms/step - loss: 0.5486 - accuracy: 0.7943
Epoch 17/100
29/29 [==============================] - 0s 2ms/step - loss: 0.5362 - accuracy: 0.7965
Epoch 18/100
29/29 [==============================] - 0s 3ms/step - loss: 0.5231 - accuracy: 0.8041
Epoch 19/100
29/29 [==============================] - 0s 3ms/step - loss: 0.5095 - accuracy: 0.8052
Epoch 20/100
29/29 [==============================] - 0s 3ms/step - loss: 0.4963 - accuracy: 0.8139
Epoch 21/100
29/29 [==============================] - 0s 3ms/step - loss: 0.4819 - accuracy: 0.8172
Epoch 22/100
29/29 [==============================] - 0s 2ms/step - loss: 0.4680 - accuracy: 0.8237
Epoch 23/100
29/29 [==============================] - 0s 4ms/step - loss: 0.4526 - accuracy: 0.8259
Epoch 24/100
29/29 [==============================] - 0s 3ms/step - loss: 0.4385 - accuracy: 0.8422
Epoch 25/100
29/29 [==============================] - 0s 3ms/step - loss: 0.4239 - accuracy: 0.8487
Epoch 26/100
29/29 [==============================] - 0s 3ms/step - loss: 0.4098 - accuracy: 0.8575
Epoch 27/100
29/29 [==============================] - 0s 3ms/step - loss: 0.3948 - accuracy: 0.8672
Epoch 28/100
29/29 [==============================] - 0s 4ms/step - loss: 0.3812 - accuracy: 0.8803
Epoch 29/100
29/29 [==============================] - 0s 4ms/step - loss: 0.3673 - accuracy: 0.8825
```

```
y_pred=model.predict(x_test)
y_pred
```

```
15/15 [==============================] - 0s 2ms/step
array([[9.20854514e-28],
       [6.93063454e-11],
       [8.19733668e-15],
       [0.00000000e+00],
```

```
       [3.75721536e-28],
       [2.85020317e-22],
       [3.00352596e-26],
       [2.52650863e-29],
       [2.81372959e-24],
       [4.02340680e-28],
       [1.00000000e+00],
       [1.00000000e+00],
       [1.30226319e-21],
       [9.99287248e-01],
       [1.40388018e-17],
       [1.00000000e+00],
       [9.98186886e-01],
       [9.98186886e-01],
       [9.99721706e-01],
       [9.98186886e-01],
       [4.44235067e-31],
       [1.75702188e-25],
       [9.98186886e-01],
       [2.64845137e-32],
       [9.98186886e-01],
       [1.69667070e-29],
       [7.18662487e-37],
       [9.99108672e-01],
       [0.00000000e+00],
       [0.00000000e+00],
       [1.00000000e+00],
       [9.66346010e-24],
       [2.75836817e-18],
       [9.98186886e-01],
       [9.98186886e-01],
       [5.08299943e-35],
       [1.00000000e+00],
       [1.00000000e+00],
       [1.00000000e+00],
       [5.92959009e-26],
       [8.35832565e-27],
       [1.00000000e+00],
       [1.00000000e+00],
       [4.44491278e-30],
       [9.98186886e-01],
       [9.99458730e-01],
       [9.98186886e-01],
       [8.12956167e-18],
       [3.53270683e-18],
       [1.00000000e+00],
       [5.57486315e-18],
       [4.58032502e-31],
       [6.48378941e-29],
       [2.75666532e-32],
       [2.60135519e-21],
       [1.00000000e+00],
```

```
import numpy as np
y_pred=np.round(y_pred).astype(int)
y_pred
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [0],
       [1],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [0],
       [1],
       [0],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [1],
       [0],
```

```
       [0],
       [1],
       [1],
       [0],
       [1],
       [1],
       [1],
       [0],
       [0],
       [1],
       [1],
       [0],
       [1],
       [1],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0]
```

```
# model.fit(scaled_train,y_train,epochs=100,verbose=0)
```

```
# model.fit(scaled_train,y_train,epochs=100,verbose=2)
```

```
# model.fit(scaled_train,y_train,epochs=100,verbose=3)
```

```
model.evaluate(x=scaled_test,y=y_test)
```

```
15/15 [==============================] - 0s 2ms/step - loss: 0.0831 - accuracy: 0.9757
[0.08308165520429611, 0.9757174253463745]
```

```
from sklearn.metrics import confusion_matrix,classification_report
```

```
confusion_matrix(y_test,y_pred)
```

```
array([[254,   3],
       [  1, 195]])
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.99      0.99       257
         1.0       0.98      0.99      0.99       196

    accuracy                           0.99       453
   macro avg       0.99      0.99      0.99       453
weighted avg       0.99      0.99      0.99       453
```

```
model.save('mymodel.h5')
from keras.models import load_model
newmodel=load_model('mymodel.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file v
  saving_api.save_model(
```

```
newmodel.predict([[0.65576832, 0.76019778, 0.22784929, 0.52919178],
  [0.51021119, 0.83150383, 0.09609469, 0.51778828]])
```

```
1/1 [==============================] - 0s 83ms/step
array([[0.00290658],
       [0.28803214]], dtype=float32)
```