NOTES FOR OPENCV

ARRAYS

In [1]:
```
# importing numpy
import numpy as np
```

In [3]:
```
#creating the list
list=[1,2,3,45,5]
```

In [4]:
```
#creating an array with list
arr=np.array(list)
```

In [6]:
```
#displaying array
arr
```

Out[6]:
```
array([ 1,  2,  3, 45,  5])
```

In [7]:
```
#creating the array with range
arr1=np.arange(1,10)
arr1
```

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

In [8]:
```
#creating the array with range of intervals
arr2=np.arange(1,10,2)
arr2
```

Out[8]:
```
array([1, 3, 5, 7, 9])
```

In [12]:
```
#creating zeros with numpy arrays
np.zeros((5,4)) #rows anda columns #remember two brackets
```

Out[12]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [13]:
```
#creating ones with numpy arrays
np.ones((5,4)) #rows anda columns #remember two brackets
```

Out[13]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [14]:
```
np.random.seed(101) # getting random numbers till 101
arr = np.random.randint(0,100,10)#getting till 10 numbers
```

In [15]:
```
#displaying array
arr
```

Out[15]:
```
array([95, 11, 81, 70, 63, 87, 75,  9, 77, 40])
```

In [16]:
```
arr2 = np.random.randint(0,100,10) #creating an array with random numbers of 1
0 indexex
```

In [18]:
```
arr2 #displaying arr2
```

Out[18]:
```
array([ 4, 63, 40, 60, 92, 64,  5, 12, 93, 40])
```

In [20]:
```
arr2.max() #getting maximum value in the array
```

Out[20]:
```
93
```

In [21]:
```
arr2.min() #getting minimum value in the array
```

Out[21]:
```
4
```

In [23]:
```
arr2.mean() #finding mean value
```

Out[23]:
```
47.3
```

In [24]:
```
arr2.argmin() #min element index
```

Out[24]:
```
0
```

In [25]:
```
arr2.argmax() #max element index
```

Out[25]:
```
8
```

In [26]:
```
arr2.reshape(2,5) #shapping the array with rows and columns   (rows*columns=nu
mber of elements in the array) is important
```

Out[26]:
```
array([[ 4, 63, 40, 60, 92],
       [64,  5, 12, 93, 40]])
```

In [27]:
```
mat = np.arange(0,100).reshape(10,10) #both aranging and shapping in the code
```

In [28]:
```
# displaying the array
mat
```

Out[28]:
```
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],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

In [29]:
```
mat[2,3]  #getting the value of the given row and column
```

Out[29]:
```
23
```

In [31]:
```
#array slycing
mat[2,3:]
```

Out[31]:
```
array([23, 24, 25, 26, 27, 28, 29])
```

In [32]:
```
#example 2
mat[2,4:]
```

Out[32]:
```
array([24, 25, 26, 27, 28, 29])
```

In [34]:
```
#slying according to the given row,column indexes
mat[0:3,0:3]
```

Out[34]:
```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22]])
```

END OF ARRAY CONCEPTS

IMAGES AND NUMPY

Numpy can read in certain files types, this includes images stored as arrays. In this quick lecture, we'll quickly cover how you can work with images in NumPy. *Keep in mind, we will mainly be using OpenCV to open and view images,

In [7]:
```
#importing the numpy module
import numpy as np
```

In [13]:
```
#importing matplotlib as plt          it is used to view the image in array fo
rmat
import matplotlib.pyplot as plt
```

In [9]:
```
#importing image package from PIL
from PIL import Image
```

In [10]:
```
#opening Image
pic = Image.open('dog_backpack.png')
```

In [4]:
```
#Displaying the pic
pic
```

Out[4]:

In [5]:
```
#displaying the type of the iamge
type(pic)
```

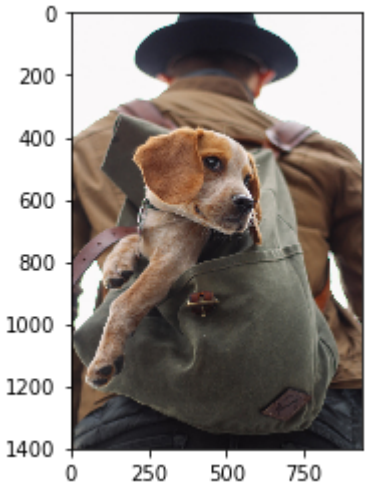
Out[5]:
```
PIL.JpegImagePlugin.JpegImageFile
```

In [11]:
```
#converting the normal image to array image with asarray() method to view in p
lt.imshow()
pic_arr = np.asarray(pic)
pic_arr.shape
```

Out[11]:
```
(1401, 934, 3)
```

In [12]:
```
#displaying the image in array format
plt.imshow(pic_arr)
```

Out[12]:
```
<matplotlib.image.AxesImage at 0x1f308095f60>
```



In [14]:
```
#to copy the pic_arr to pic_red variable
pic_red=pic_arr.copy()
```

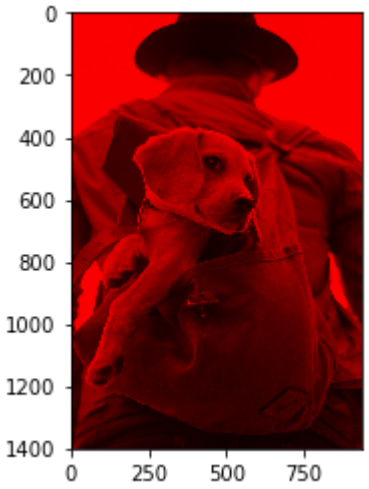
In [15]:
```
pic_red[:, :, 1] = 0    # Zero out contribution from green # to destroy green
 colour
pic_red[:, :, 2] = 0    # Zero out contribution from blue  # to destroy blue c
olour
```

In [16]:
```
#displaying the red image in array format
pic_red
```

Out[16]:
```
array([[[250,   0,   0],
        [250,   0,   0],
        [250,   0,   0],
        ...,
        [250,   0,   0],
        [250,   0,   0],
        [250,   0,   0]],

       [[250,   0,   0],
        [250,   0,   0],
        [250,   0,   0],
        ...,
        [250,   0,   0],
        [250,   0,   0],
        [250,   0,   0]],

       [[250,   0,   0],
        [250,   0,   0],
        [250,   0,   0],
        ...,
        [250,   0,   0],
        [250,   0,   0],
        [250,   0,   0]],

       ...,

       [[ 49,   0,   0],
        [ 51,   0,   0],
        [ 53,   0,   0],
        ...,
        [ 75,   0,   0],
        [ 72,   0,   0],
        [ 70,   0,   0]],

       [[ 50,   0,   0],
        [ 52,   0,   0],
        [ 53,   0,   0],
        ...,
        [ 75,   0,   0],
        [ 73,   0,   0],
        [ 70,   0,   0]],

       [[ 51,   0,   0],
        [ 51,   0,   0],
        [ 54,   0,   0],
        ...,
        [ 76,   0,   0],
        [ 71,   0,   0],
        [ 67,   0,   0]]], dtype=uint8)
```

In [17]:
```
#displaying the image with imshow
plt.imshow(pic_red)
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x1f30a143978>
```



START WITH OPEN CV

In [4]:
```
#importing numpy module
import numpy as np
#importing matplotlib
import matplotlib.pyplot as plt
#imporitng cv2 module
import cv2
```

In [5]:
```
#we can also use cv2 module to read the image
image=cv2.imread("00-puppy.jpg")
```

In [6]:
```
#before viewing the image please check the type that it has converted to nump
y.ndarray
type(image)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
# displaying the output image
plt.imshow(image)
```

Out[7]: <matplotlib.image.AxesImage at 0x1edb2ca7208>

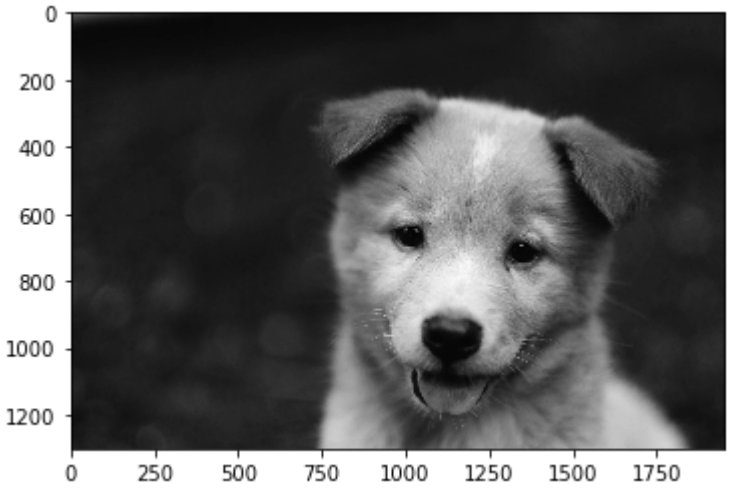
In [8]:
```
#the default colour of an image is bgr we need to convert into rgb, we need to convert it by cvtcolour function in cv2 module
image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
```

In [9]:
```
# displaying the output image
plt.imshow(image)
```

Out[9]: <matplotlib.image.AxesImage at 0x1edb3bdf4a8>

In [10]:
```
#converting the image in black and white
img_gray = cv2.imread('00-puppy.jpg',cv2.IMREAD_GRAYSCALE)
plt.imshow(img_gray,cmap='gray')
```

Out[10]: <matplotlib.image.AxesImage at 0x1edb4391c88>



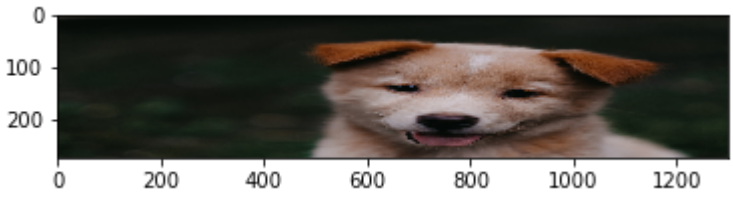
In [11]:
```
image.shape
# width, height, color channels
```

Out[11]: (1300, 1950, 3)

In [12]:
```
#resizing the image into required shape with resize function
img =cv2.resize(image,(1300,275))
```

In [13]:
```
# displaying the output image
plt.imshow(img)
```

Out[13]: <matplotlib.image.AxesImage at 0x1edb44259e8>



In [15]:
```
#resizing with ratios
w_ratio = 0.5
h_ratio = 0.5
```

In [18]:
```
#resizing the image with parameters given by the user
new_img =cv2.resize(image,(0,0),img,w_ratio,h_ratio)
```

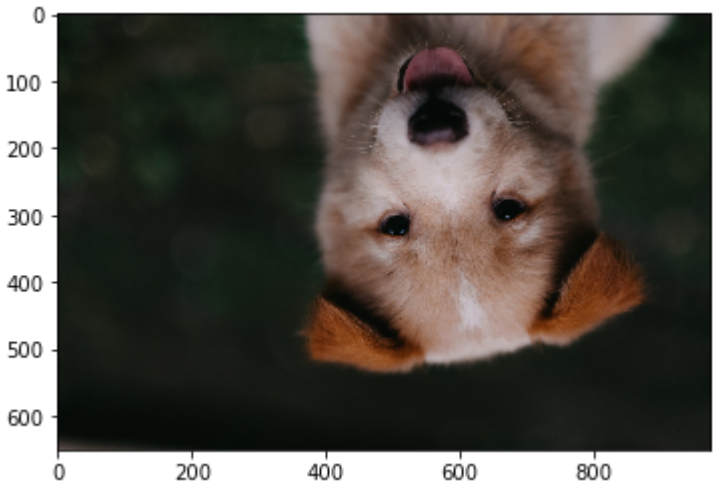
In [19]:
```
# displaying the output image
plt.imshow(new_img)
```

Out[19]: <matplotlib.image.AxesImage at 0x1edb45ae400>

FLIPPING IMAGES

In [20]:
```
# Along central x axis
new_img = cv2.flip(new_img,0)
plt.imshow(new_img)
```

Out[20]: <matplotlib.image.AxesImage at 0x1edb4604438>



In [21]:
```
# Along central y axis
new_img = cv2.flip(new_img,1)
plt.imshow(new_img)
```

Out[21]: <matplotlib.image.AxesImage at 0x1edb4ffd4e0>

In [22]:
```
# Along both axis
new_img = cv2.flip(new_img,-1)
plt.imshow(new_img)
```

Out[22]: <matplotlib.image.AxesImage at 0x1edb50544e0>

WRITING THE EDITTED IMAGE INTO PC

In [23]:
```
#storing the image to the pc
cv2.imwrite('my_new_picture.jpg',new_img)
```

Out[23]: True

DISPLAYING THE IMAGE IN LARGER FORMAT

In [24]:
```
#step 1: using the figure function and figsize parameter
fig = plt.figure(figsize=(10,8))
#step 2:adding rhe image an subplot
ax = fig.add_subplot(111)
#step3:viewing
ax.imshow(new_img)
```

Out[24]: <matplotlib.image.AxesImage at 0x1edb50a86d8>

Opening Image Files with OpenCV

In [27]:
```
# MUST BE RUN AS .py SCRIPT IN ORDER TO WORK.
# PLEASE MAKE SURE TO WATCH THE FULL VIDEO FOR THE EXPLANATION TO THIS NOTEBOO
K
# TO BE CLEAR: RUNNING THIS CELL WILL KILL THE KERNEL IF YOU USE JUPYTER DIREC
TLY

import cv2

img = cv2.imread('00-puppy.jpg',cv2.IMREAD_GRAYSCALE)
# Show the image with OpenCV
cv2.imshow('window_name',img)
# Wait for something on keyboard to be pressed to close window.
cv2.waitKey()
```

Out[27]: -1

DRAWING ON THE IMAGES

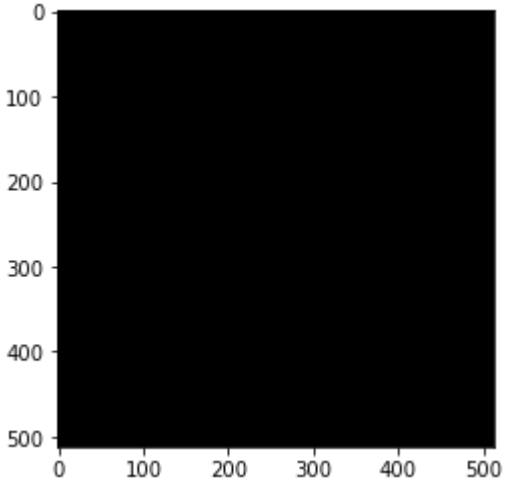
In [28]:
```
#creating an blank image with zeros array
blank_img = np.zeros(shape=(512,512,3),dtype=np.int16)
```

In [30]:
```
#viewing the shape
blank_img.shape
```

Out[30]: (512, 512, 3)

In [31]:
```
#displaying the image
plt.imshow(blank_img)
```

Out[31]: <matplotlib.image.AxesImage at 0x1edb448e4e0>



SHAPES IN THE IMAGE

Rectangles

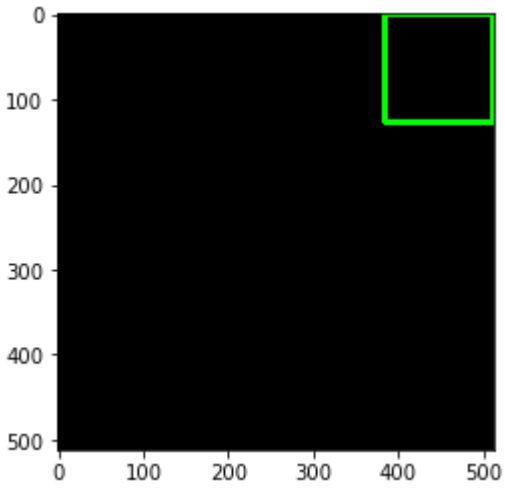
- img Image.
- pt1 Vertex of the rectangle.
- pt2 Vertex of the rectangle opposite to pt1 .
- color Rectangle color or brightness (grayscale image).
- thickness Thickness of lines that make up the rectangle. Negative values, like #FILLED,mean that the function has to draw a filled rectangle.
- lineType Type of the line. See #LineTypes
- shift Number of fractional bits in the point coordinates.

In [32]:
```
# pt1 = top left
# pt2 = bottom right
#parameters=(source,left coordinates,right coordinates,color,thickness)
cv2.rectangle(blank_img,pt1=(384,0),pt2=(510,128),color=(0,255,0),thickness=5)
```

Out[32]:
```
array([[[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   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)
```

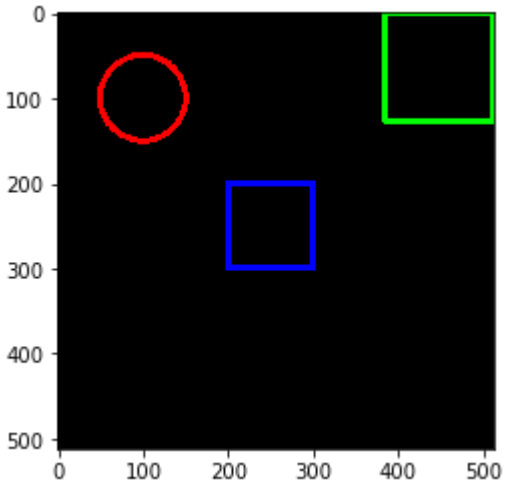
In [33]:
```
# displaying the output image
plt.imshow(blank_img)
```

Out[33]: <matplotlib.image.AxesImage at 0x1edb44e75c0>



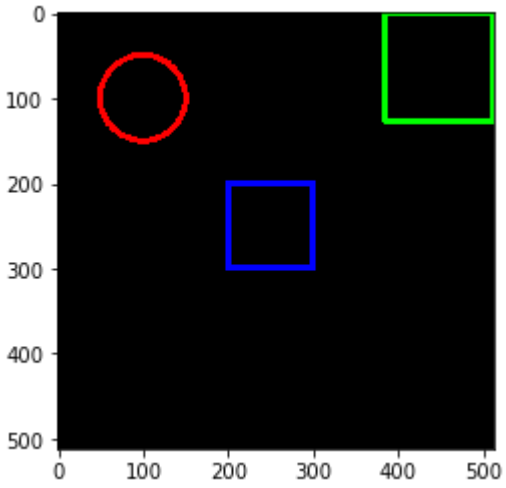
In [36]:
```
# pt1 = top left
# pt2 = bottom right
#parameters=(source,left coordinates,right coordinates,color,thickness)
cv2.rectangle(blank_img,pt1=(200,200),pt2=(300,300),color=(0,0,255),thickness=
5)
plt.imshow(blank_img)
```

Out[36]: <matplotlib.image.AxesImage at 0x1edb5bafb38>



In [37]:
```
#creating an circle in the image
#paramenters=(source,center,radius,color,thickness)
cv2.circle(img=blank_img, center=(100,100), radius=50, color=(255,0,0), thickn
ess=5)
plt.imshow(blank_img)
```

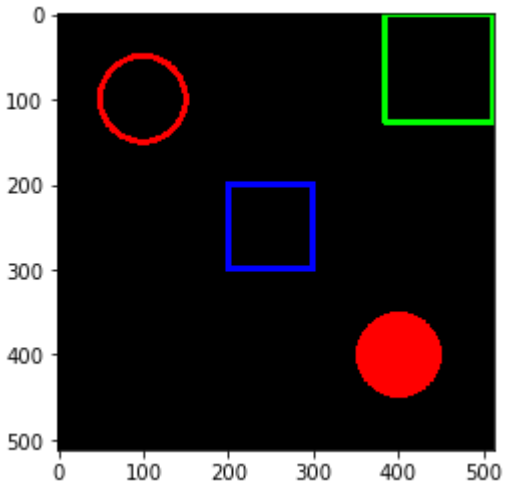
Out[37]: <matplotlib.image.AxesImage at 0x1edb6029ac8>



FILLING IN THE SHAPES

In [39]:
```
#in thickness we need to give negative value to fill the image
cv2.circle(img=blank_img, center=(400,400), radius=50, color=(255,0,0), thickn
ess=-1)
plt.imshow(blank_img)
```

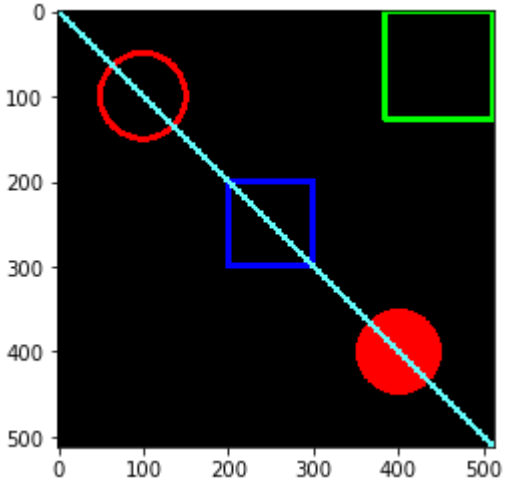
Out[39]: <matplotlib.image.AxesImage at 0x1edb60dab70>



In [40]:
```
#drawing the line in the image
```

In [41]:
```
# Draw a diagonal blue line with thickness of 5 px
cv2.line(blank_img,pt1=(0,0),pt2=(511,511),color=(102, 255, 255),thickness=5)
plt.imshow(blank_img)
```

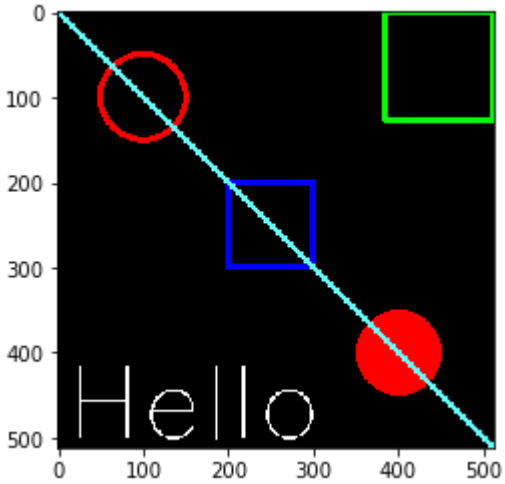
Out[41]: <matplotlib.image.AxesImage at 0x1edb6131c50>



TEXT IN THE IMAGE

In [42]:
```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(blank_img,text='Hello',org=(10,500), fontFace=font,fontScale= 4,co
lor=(255,255,255),thickness=2,lineType=cv2.LINE_AA)
plt.imshow(blank_img)
```

Out[42]: <matplotlib.image.AxesImage at 0x1edb6189eb8>



DRAWING CICLES THROUGH MOUSE WHERE EVER WE WANT

In [1]:
```
import cv2
import numpy as np
# Create a function based on a CV2 Event (Left button click)
def draw_circle(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(img,(x,y),100,(0,255,0),-1)

# Create a black image
img = np.zeros((512,512,3), np.uint8)
# This names the window so we can reference it
cv2.namedWindow(winname='my_drawing')
# Connects the mouse button to our callback function
cv2.setMouseCallback('my_drawing',draw_circle)

while True: #Runs forever until we break with Esc key on keyboard
    # Shows the image window
    cv2.imshow('my_drawing',img)
    # EXPLANATION FOR THIS LINE OF CODE:
    # https://stackoverflow.com/questions/35372700/whats-0xff-for-in-cv2-waitk
ey1/39201163
    if cv2.waitKey(20) & 0xFF == 27:
        break
# Once script is done, its usually good practice to call this line
# It closes all windows (just in case you have multiple windows called)
cv2.destroyAllWindows()
```

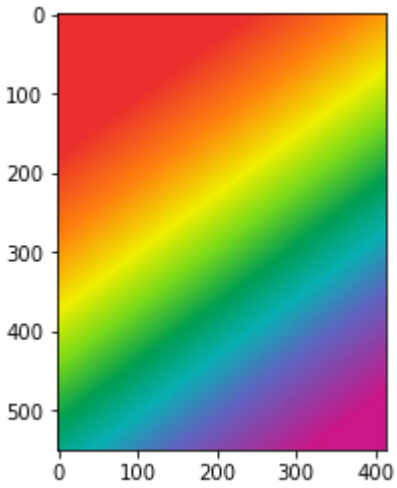
CREATING AN RECTANGE WITH THE MOUSE BY DRAGGING

In [2]:
```
import cv2
import numpy as np


# Create a function based on a CV2 Event (Left button click)
drawing = False # True if mouse is pressed
ix,iy = -1,-1

# mouse callback function
def draw_rectangle(event,x,y,flags,param):
    global ix,iy,drawing,mode

    if event == cv2.EVENT_LBUTTONDOWN:
        # When you click DOWN with left mouse button drawing is set to True
        drawing = True
        # Then we take note of where that mouse was located
        ix,iy = x,y

    elif event == cv2.EVENT_MOUSEMOVE:
        # Now the mouse is moving
        if drawing == True:
            # If drawing is True, it means you've already clicked on the left
 mouse button
            # We draw a rectangle from the previous position to the x,y where
 the mouse is
            cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),-1)


    elif event == cv2.EVENT_LBUTTONUP:
        # Once you lift the mouse button, drawing is False
        drawing = False
        # we complete the rectangle.
        cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),-1)



# Create a black image
img = np.zeros((512,512,3), np.uint8)
# This names the window so we can reference it
cv2.namedWindow(winname='my_drawing')
# Connects the mouse button to our callback function
cv2.setMouseCallback('my_drawing',draw_rectangle)

while True: #Runs forever until we break with Esc key on keyboard
    # Shows the image window
    cv2.imshow('my_drawing',img)


    # CHECK TO SEE IF ESC WAS PRESSED ON KEYBOARD
    if cv2.waitKey(1) & 0xFF == 27:
        break
# Once script is done, its usually good practice to call this line
# It closes all windows (just in case you have multiple windows called)
cv2.destroyAllWindows()
```

ALL THE TYPES OF THRESHOLD IMAGES

In [2]:
```
#to convert the colour into perfet black and white image
import numpy as np
import matplotlib.pyplot as plt
import cv2
from PIL import Image
#opening the image
image1=Image.open("rainbow.jpg")
# displaying the output image
plt.imshow(image1)
```

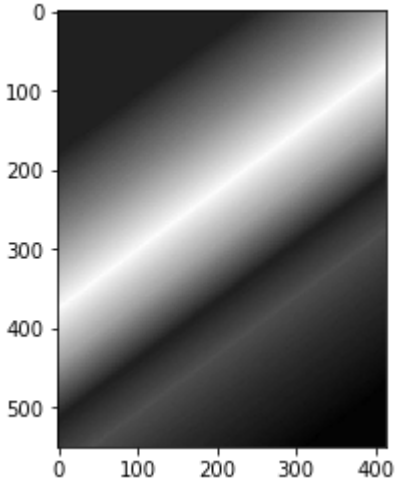
Out[2]: <matplotlib.image.AxesImage at 0x1a97e49abe0>



In [3]:
```
# reading the image
image1=cv2.imread("rainbow.jpg",0)
```

In [4]:
```
# displaying the output image
plt.imshow(image1,cmap="gray")
```

Out[4]: <matplotlib.image.AxesImage at 0x1a97e529f60>



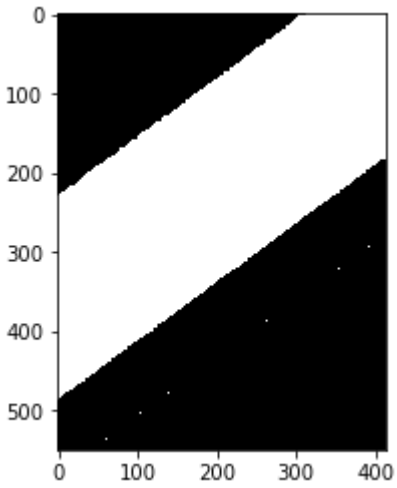
In [5]:
```
#COPYING THRESHLOD TO RET AND THRESH SO,
#in this process the value below 127 will become 0 and the value above 127 wil
l become 255
ret,thres=cv2.threshold(image1,127,255,cv2.THRESH_BINARY)
```

In [6]:
```
ret #returning the value
```

Out[6]:
```
127.0
```

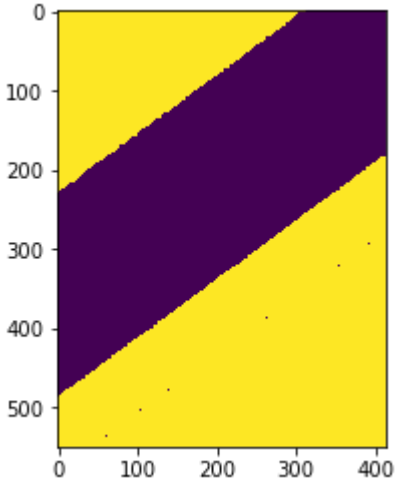
In [7]:
```
# displaying the output image
plt.imshow(thres,cmap="gray")
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x1a97e590588>
```



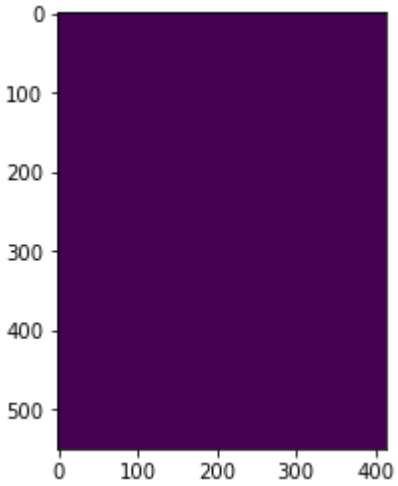
In [8]:
```
ret,thres=cv2.threshold(image1,127,255,cv2.THRESH_BINARY_INV)
# displaying the output image
plt.imshow(thres)
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x1a97e5e8518>
```



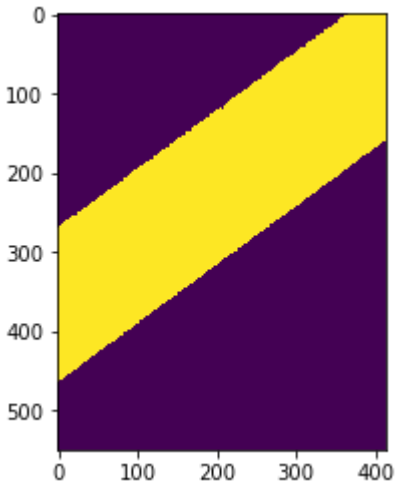
In [9]:
```
ret,thres=cv2.threshold(image1,127,255,cv2.THRESH_MASK)
# displaying the output image
plt.imshow(thres)
```

Out[9]: <matplotlib.image.AxesImage at 0x1a97e637fd0>



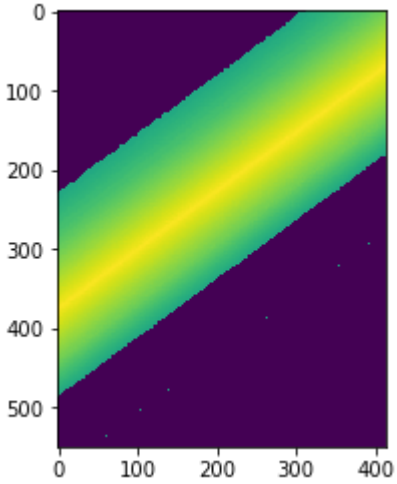
In [10]:
```
ret,thres=cv2.threshold(image1,127,255,cv2.THRESH_OTSU)
# displaying the output image
plt.imshow(thres)
```

Out[10]: <matplotlib.image.AxesImage at 0x1a97e69fb38>



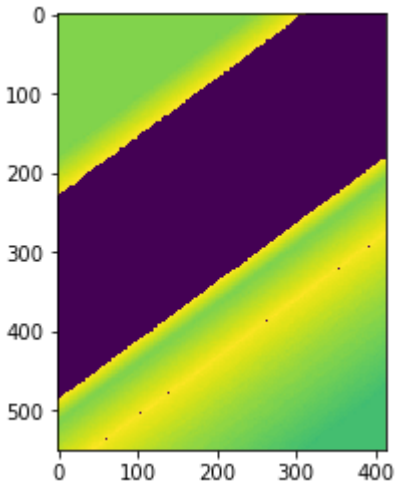
In [11]:
```
ret,thres=cv2.threshold(image1,127,255,cv2.THRESH_TOZERO)
# displaying the output image
plt.imshow(thres)
```

Out[11]: <matplotlib.image.AxesImage at 0x1a97e6f56a0>



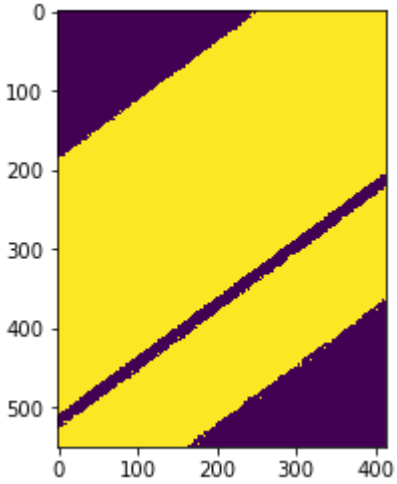
In [12]:
```
ret,thres=cv2.threshold(image1,127,255,cv2.THRESH_TOZERO_INV)
# displaying the output image
plt.imshow(thres)
```

Out[12]: <matplotlib.image.AxesImage at 0x1a97e74f278>



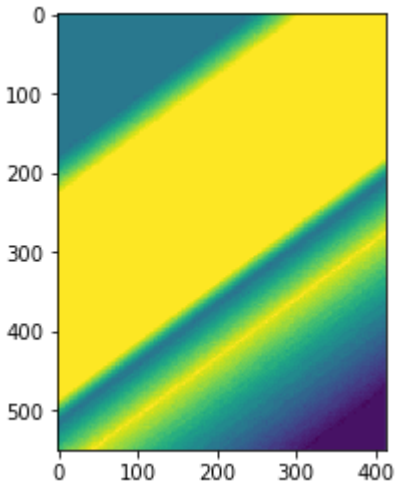
In [13]:
```
ret,thres=cv2.threshold(image1,127,255,cv2.THRESH_TRIANGLE)
# displaying the output image
plt.imshow(thres)
```

Out[13]: <matplotlib.image.AxesImage at 0x1a97e79ccf8>



In [14]:
```
ret,thres=cv2.threshold(image1,127,255,cv2.THRESH_TRUNC)
# displaying the output image
plt.imshow(thres)
```

Out[14]: <matplotlib.image.AxesImage at 0x1a97e7f2908>



BLENDING TWO IMAGES IN SUITABLE WAY

In [15]:
```
#importing all requirements
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [16]:
```
#while reading the image the default colour in imread will be bgr
image1=cv2.imread("dog_backpack.png")
```

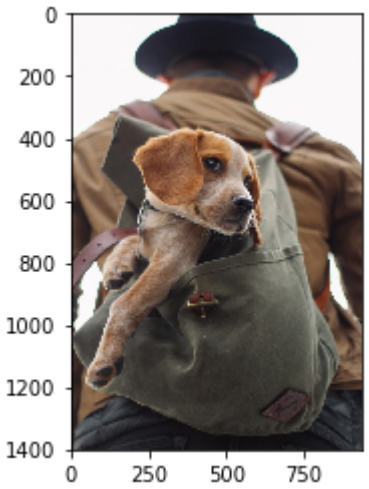
In [24]:
```
#so, we use cvtColor to convert to rgb
image1=cv2.cvtColor(image1,cv2.COLOR_BGR2RGB)
```

In [32]:
```
#while reading the image the default colour in imread will be bgr
image2=cv2.imread("watermark_no_copy.png")
```

In [33]:
```
#so, we use cvtColor to convert to rgb
image2=cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
```

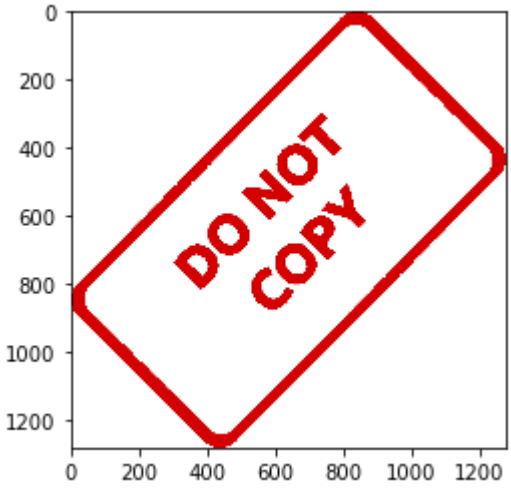
In [34]:
```
# displaying the output image
plt.imshow(image1)
```

Out[34]: <matplotlib.image.AxesImage at 0x1a97f635400>



In [35]:
```
# displaying the output image
plt.imshow(image2)
```

Out[35]: <matplotlib.image.AxesImage at 0x1a90001d438>



In [36]:
```
# to make the image 2 in 600 x 600 we use resize in cv2
image2=cv2.resize(image2,(600,600))
```

In [37]:
```
# displaying the output image
plt.imshow(image2)
```

Out[37]: <matplotlib.image.AxesImage at 0x1a900532cf8>

In [38]:
```
#in shape function we get height,width,no of channels so
image1.shape
```

Out[38]: (1401, 934, 3)

In [39]:
```
#copying the respected variables form the image shape
height,width,no_of_channels=image1.shape
```

In [40]:
```
# diplaying the height
height
```

Out[40]: 1401

In [41]:
```
#displaying the width
width
```

Out[41]: 934

In [43]:
```
#displaying the no_of_channels(3 channels are = red green blue)
no_of_channels
```

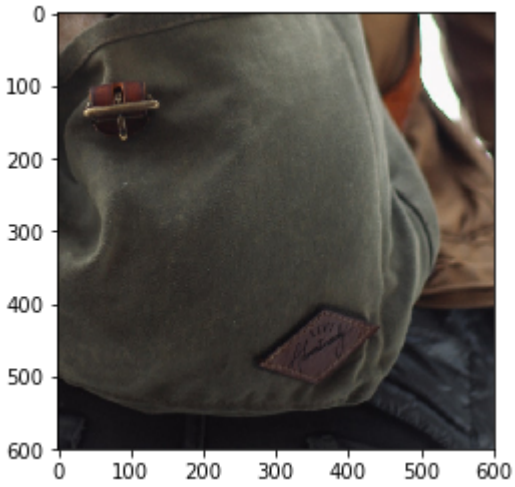
Out[43]: 3

In [44]:
```
#x_offset and y_offset are locator of image like points in graph
x_offset=934-600
y_offset=1401-600
```

In [45]:
```
#for the roi(region of intrest) we need x_offset,y_offset so,
roi=image1[y_offset:1401,x_offset:943]
```

In [46]:
```
# displaying the output image
plt.imshow(roi)
```

Out[46]: <matplotlib.image.AxesImage at 0x1a902a334e0>

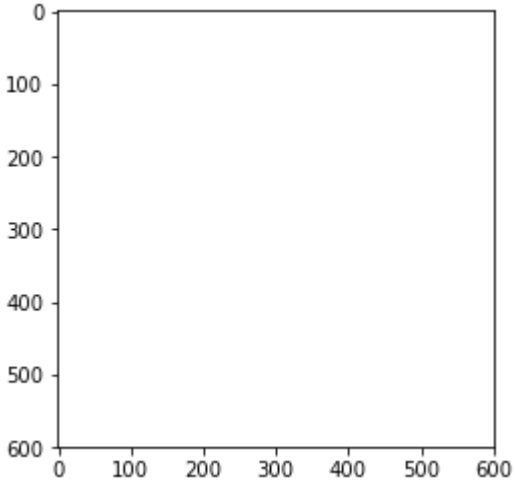


In [47]:
```
#converting suitable image
```

In [48]:
```
#creating white background
white_background=np.full(image2.shape,255,dtype=np.uint8)
```

In [49]:
```
# displaying the output image
plt.imshow(white_background)
```

Out[49]: <matplotlib.image.AxesImage at 0x1a9031fc2e8>



In [50]:
```
#converting the image2 to gray(black and white color)
img2gray=cv2.cvtColor(image2,cv2.COLOR_RGB2GRAY)
#bitwise_not hepls to inverse the colours(eg:it converts from black to white a
nd white to black)
mask_inv=cv2.bitwise_not(img2gray)
fg=cv2.bitwise_or(image2,image2,mask=mask_inv)
```

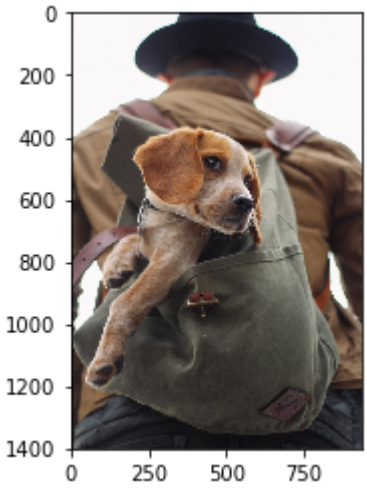
In [51]:
```
#masking means overlaping one and another image
#final_roi is masking image 2 to cut shoterned image1
final_roi=cv2.bitwise_or(roi,fg)
```

In [52]:
```
# displaying the output image
plt.imshow(final_roi)
```

Out[52]: <matplotlib.image.AxesImage at 0x1a902748160>

In [53]:
```
# displaying the output image
plt.imshow(image1)
```

Out[53]: <matplotlib.image.AxesImage at 0x1a9027ddc50>

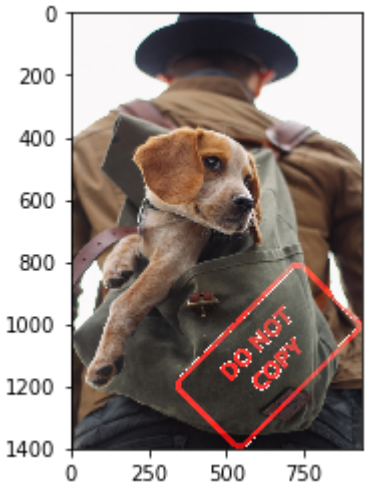


In [54]:
```
larger_image=image1
small_image=final_roi
```

In [55]:
```
#combing two images of different size to smaller image
larger_image[y_offset:y_offset+small_image.shape[0],x_offset:x_offset+small_im
age.shape[1]]= small_image
```

In [56]:
```
# displaying the output image
plt.imshow(larger_image)
```

Out[56]: <matplotlib.image.AxesImage at 0x1a90288b6a0>



BLENDING TWO IMAGES OF SAME SIZE

In [75]:
```
#BLENDING(COMBINIG TWO IMAGES IN SAME SIZE)
```

In [76]:
```
import matplotlib.pyplot as plt
import cv2
```

In [77]:
```
#reading the image with cv2 module
image1=cv2.imread("dog_backpack.png")
```

In [78]:
```
# imread will not show any error to find it we use type (it should display num
py.ndarray) if not it is an error(check image
# location)
type(image1)
```

Out[78]: numpy.ndarray

In [79]:
```
#reading the image with cv2 module
image=cv2.imread("watermark_no_copy.png")
```

In [80]:
```
# imread will not show any error to find it we use type (it should display num
py.ndarray) if not it is an error(check image
# location)
type(image)
```

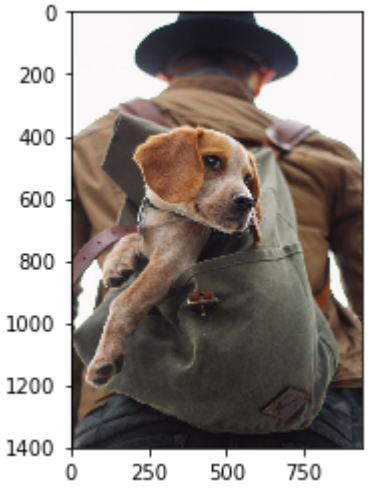
Out[80]: numpy.ndarray

In [81]:
```
#converting BLUE GREEN RED TO RED GREEN BLUE(ORIGINAL COLOR) WITH CVT
image1=cv2.cvtColor(image1,cv2.COLOR_BGR2RGB)
```

In [82]:
```
#converting BLUE GREEN RED TO RED GREEN BLUE(ORIGINAL COLOR) WITH CVT
image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
```

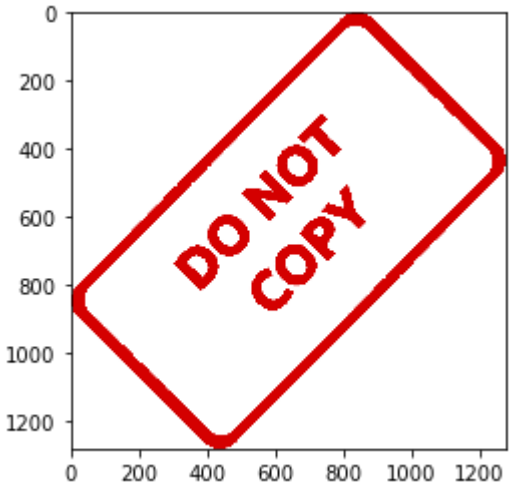
In [83]:
```
#DISPLAYING IMAGE WITH IMSHOW
plt.imshow(image1)
```

Out[83]: <matplotlib.image.AxesImage at 0x1a90471fdd8>



In [84]:
```
#DISPLAYING IMAGE WITH IMSHOW
plt.imshow(image)
```

Out[84]: <matplotlib.image.AxesImage at 0x1a9048fa748>



In [85]:
```
#to blend the image with same size we use resize to change the size
```

In [86]:
```
#resizing the first image as 1000 of width and 1000 height
image1=cv2.resize(image1,(1000,1000))
```

In [87]:
```
#resizing the second image as 1000 of width and 1000 height
image=cv2.resize(image,(1000,1000))
```

In [90]:
```
#blending two images
#we use addWeighted() function
#Formula=src1*alpha+src2*betha+gamma
```

BLURRING AND SMOTHING

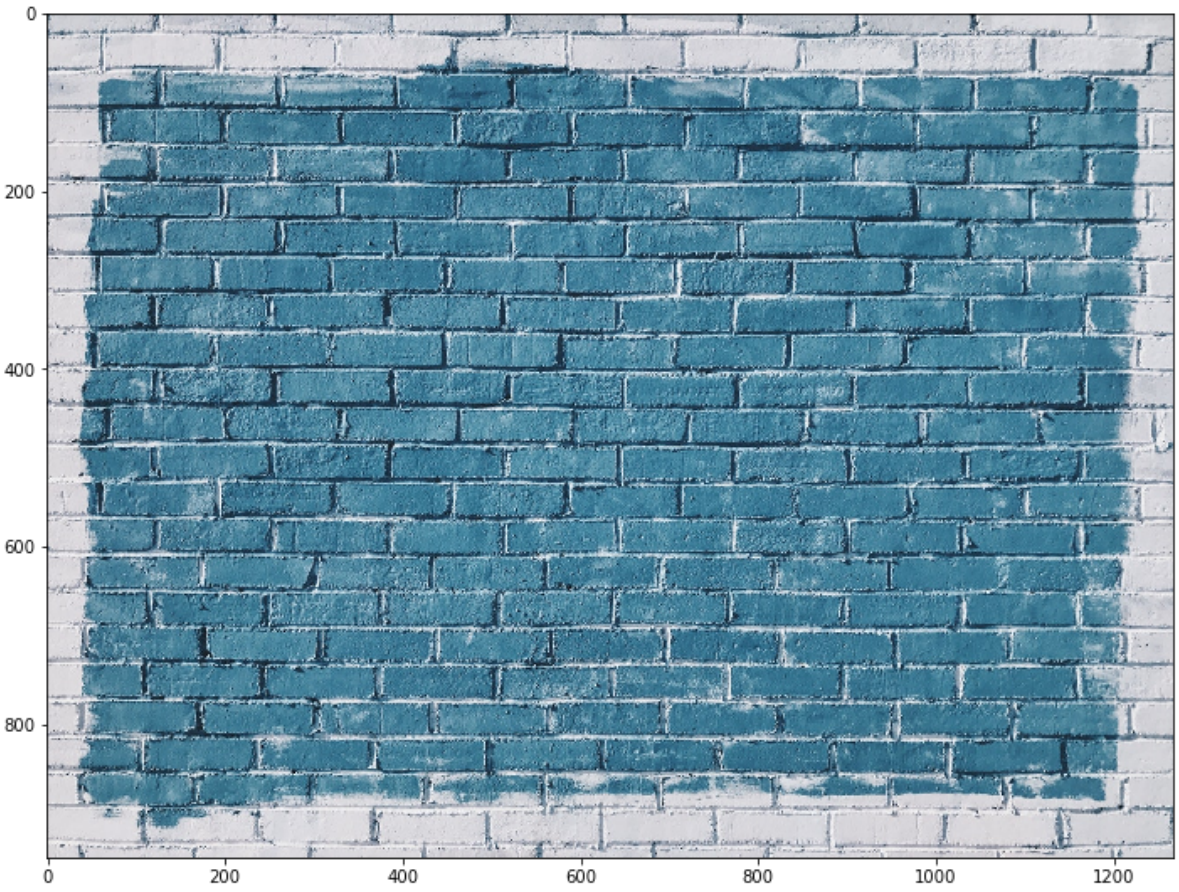
In [99]:
```
# importing warnings module
import warnings
warnings.filterwarnings('ignore')
```

In [104]:
```
#importing the required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [105]:
```
#function to read image
def load_img():
    #reading the image
    img = cv2.imread('bricks.jpg').astype(np.float32) / 255
    #converting the  BLUE GREEN RED TO RED GREEN BLUE
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img
```

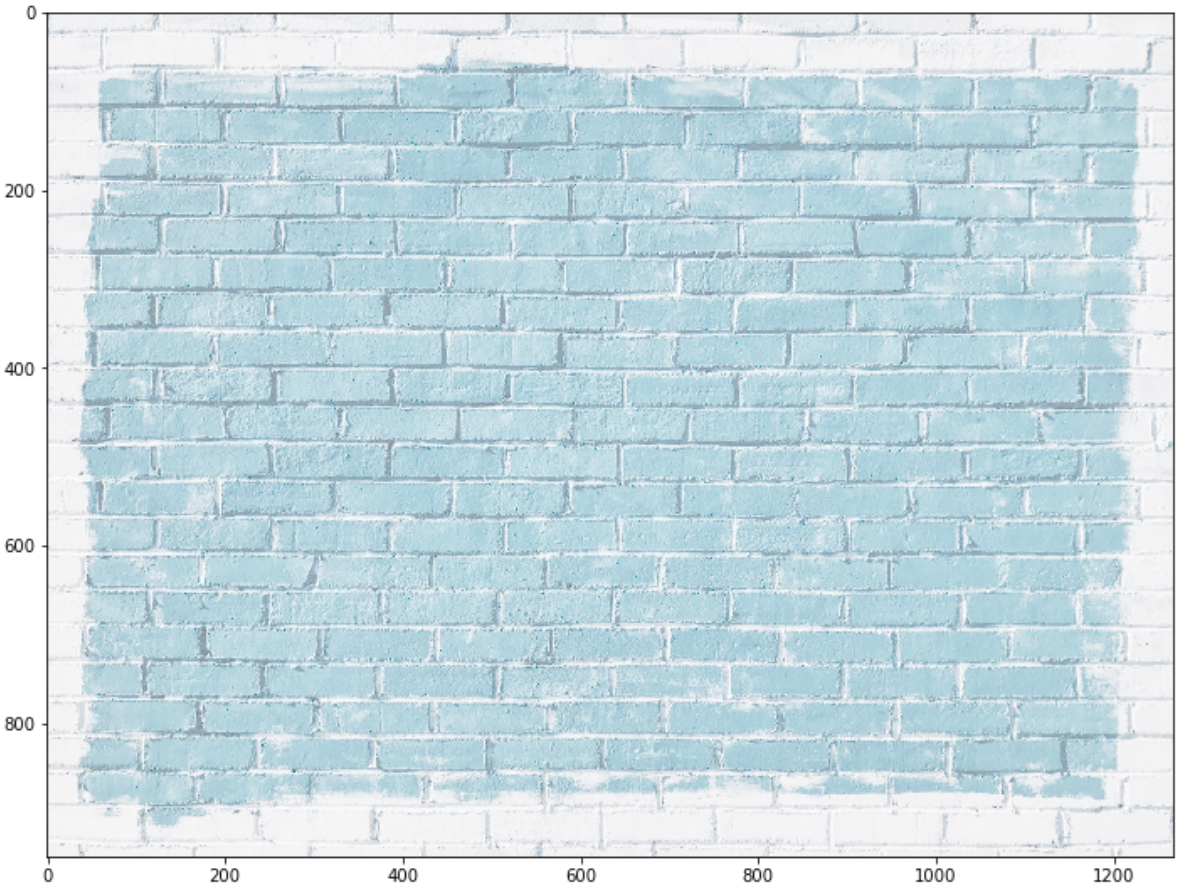
In [106]:
```
#display the image in Bigger size function
def display_img(img):
    fig = plt.figure(figsize=(12,10))
    ax = fig.add_subplot(111)
    ax.imshow(img)
```

In [107]:
```
#running both the images
i = load_img()
display_img(i)
```

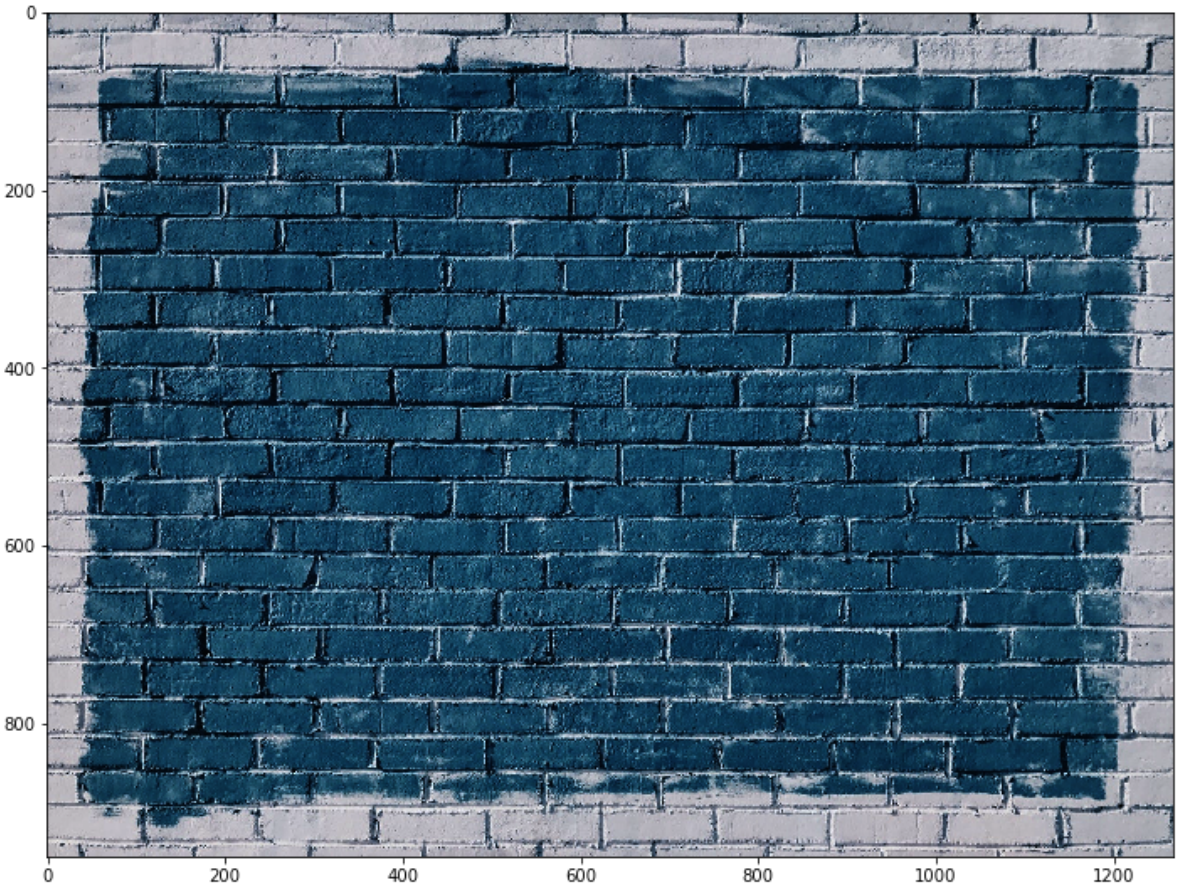


Gamma Correction : Practical Effect of Increasing Brightness

In [108]:
```
#reloading the image
img = load_img()
gamma = 1/4
#power function is used to reduce the brightness
#if gamma value is less than 1 then the brightness will be less
#if the gamma value is more than 1 then the brightness will be high
effected_image = np.power(img, gamma)
display_img(effected_image)
```



In [109]:
```
#displaying the image
img = load_img()
gamma = 2
#increasing the brightness with gamma
effected_image = np.power(img, gamma)
#displaying the image
display_img(effected_image)
```



In [117]:
```
img = load_img()
#assigning the required font to font variable
font = cv2.FONT_HERSHEY_COMPLEX
#function to print text in image with text ,place to print,font style,size,col
or,thickness
cv2.putText(img,text='bricks',org=(10,600), fontFace=font,fontScale= 10,color=
(255,0,0),thickness=4)
#displaying the output image
display_img(img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

Create the Kernel

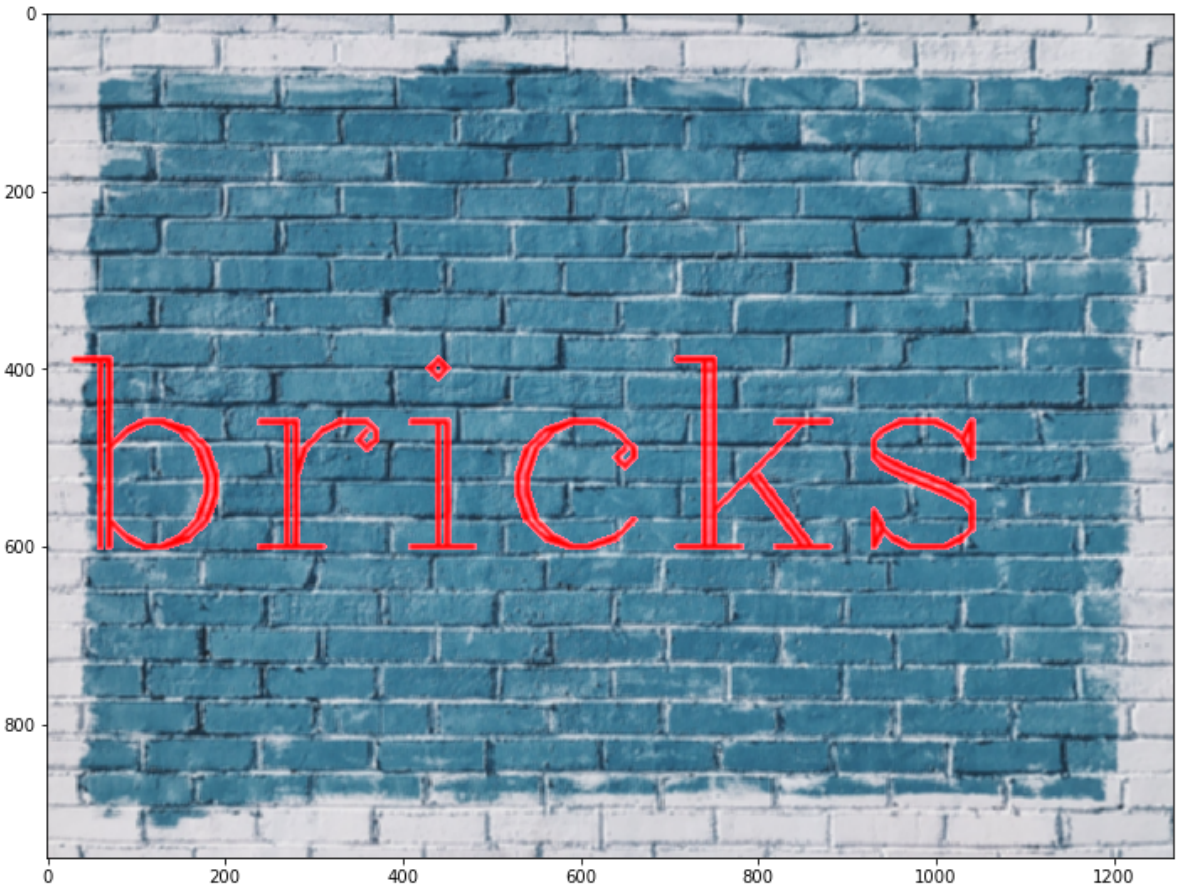
In [118]:
```
#CREATING THE KERNEL WITH ONES FUNCTION
kernel = np.ones(shape=(5,5),dtype=np.float32)/25
```

In [119]:
```
#displaying the kerne
kernel
```

Out[119]:
```
array([[0.04, 0.04, 0.04, 0.04, 0.04],
       [0.04, 0.04, 0.04, 0.04, 0.04],
       [0.04, 0.04, 0.04, 0.04, 0.04],
       [0.04, 0.04, 0.04, 0.04, 0.04],
       [0.04, 0.04, 0.04, 0.04, 0.04]], dtype=float32)
```

In [120]:
```
#adding an filter with kernel
#this filter2D helps to make the little smoothing the image
dst = cv2.filter2D(img,-1,kernel)
#displaying the image
display_img(dst)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



AVERAGING

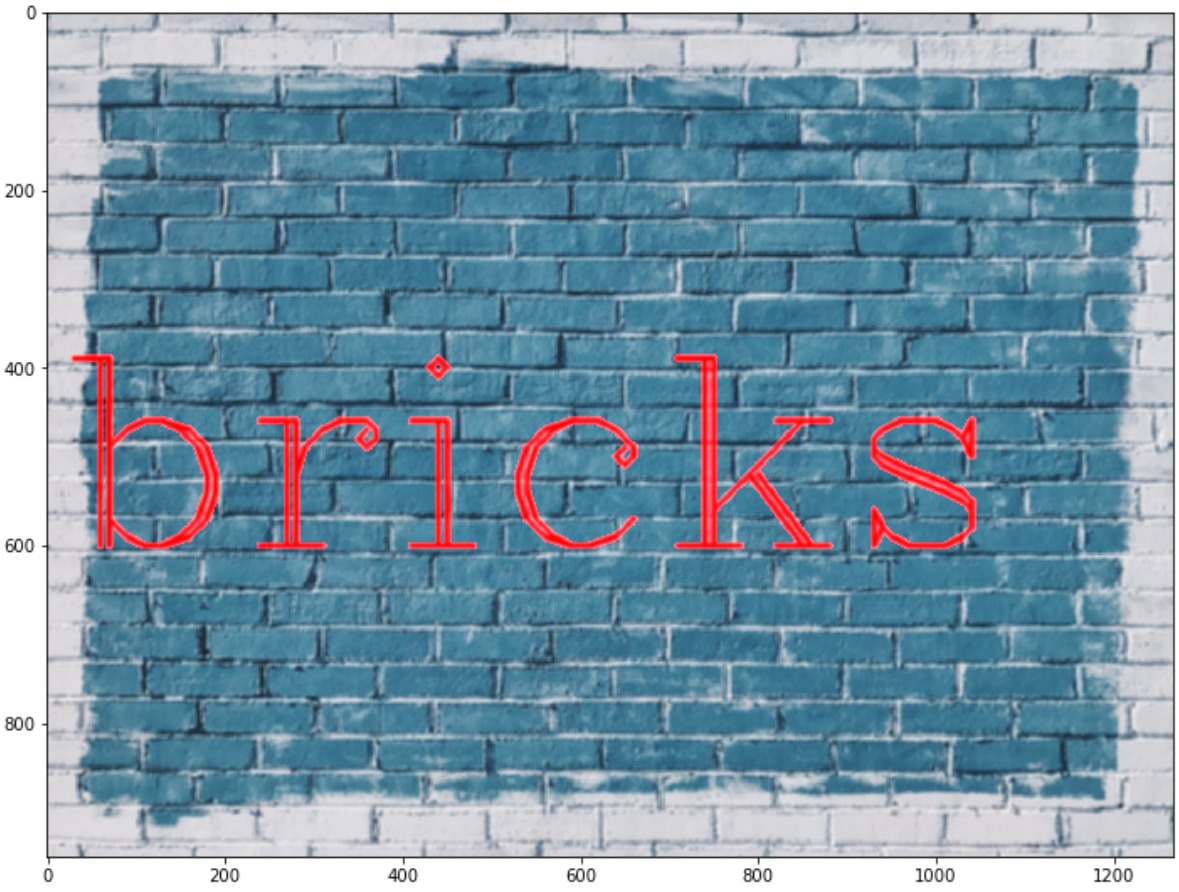
In [121]:
```
#loading the image
img = load_img()
#adding the font
font = cv2.FONT_HERSHEY_COMPLEX
#adding the text into the image
cv2.putText(img,text='bricks',org=(10,600), fontFace=font,fontScale= 10,color=
(255,0,0),thickness=4)
#displaying the image
display_img(img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

In [115]:
```
#BLURING THE IMAGE
blurred_img = cv2.blur(img,ksize=(5,5))
```

In [116]:
```
#displaying the blurred image
display_img(blurred_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



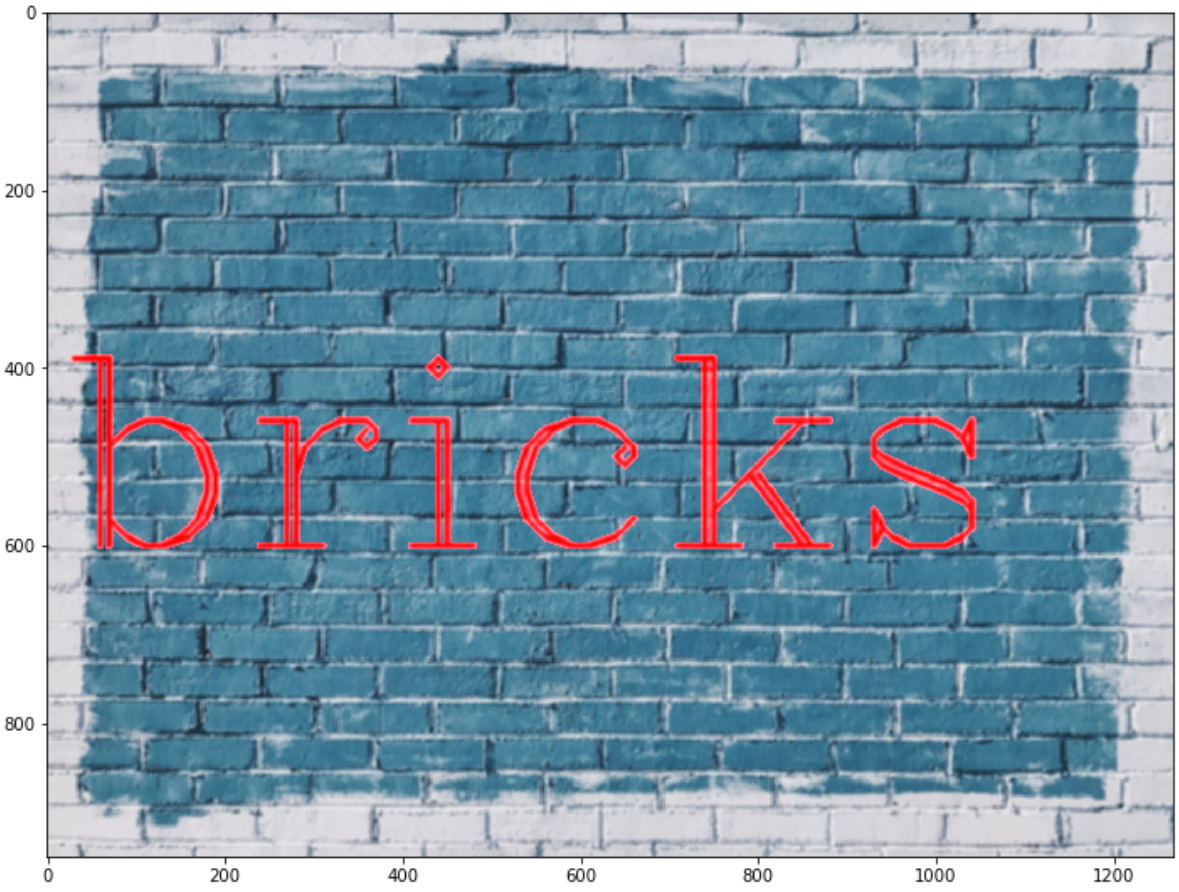
Gaussian Blurring

In [125]:
```
img = load_img()
#ADDING FONT
font = cv2.FONT_HERSHEY_COMPLEX
#ADDING TEXT TO IMAGE
cv2.putText(img,text='bricks',org=(10,600), fontFace=font,fontScale= 10,color=
(255,0,0),thickness=4)
#displaying the image
display_img(img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

In [126]:
```
#ANOTHER WAY OF BLURING
blurred_img = cv2.GaussianBlur(img,(5,5),10)
display_img(blurred_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



Median Blurring

In [127]:
```
img = load_img()
#ADDING FONT IN THE IMAGE
font = cv2.FONT_HERSHEY_COMPLEX
#ADDING TEXT INTO IMAGE
cv2.putText(img,text='bricks',org=(10,600), fontFace=font,fontScale= 10,color=
(255,0,0),thickness=4)
#displaying the image
display_img(img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

In [129]:
```
#BLURING THE IMAGE
median = cv2.medianBlur(img,5)
#DISPLAYING THE IMAGE
display_img(median)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

Adding Noise

In [131]:
```
#READING THE IMAGE
img = cv2.imread('sammy.jpg')
#converting bgr2rgb colour with cvtColor
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

In [132]:
```
#dispalying the max value
img.max()
```

Out[132]: 255

In [133]:
```
#dispalying the min value
img.min()
```

Out[133]: 0

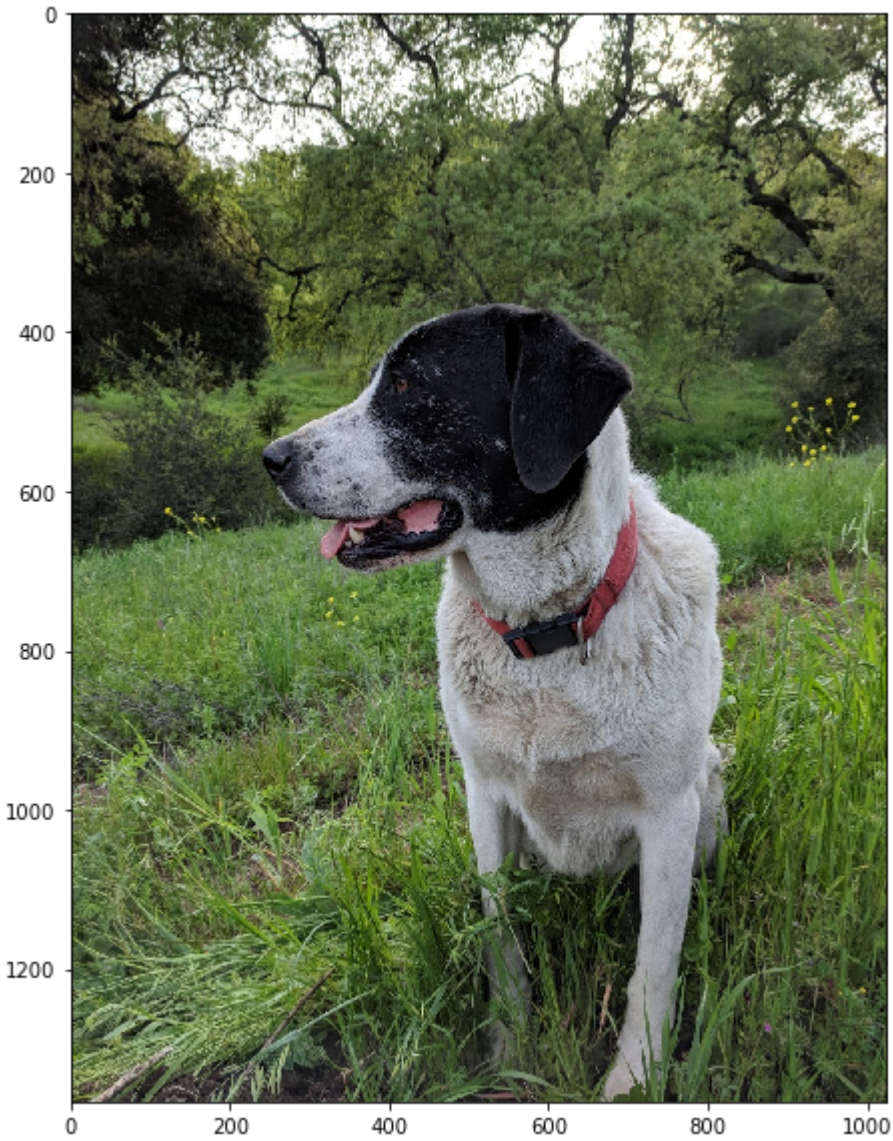
In [134]:
```
#dispalying the mean
img.mean()
```

Out[134]: 89.33240744375256

In [135]:
```
#dispalying the shape
img.shape
```

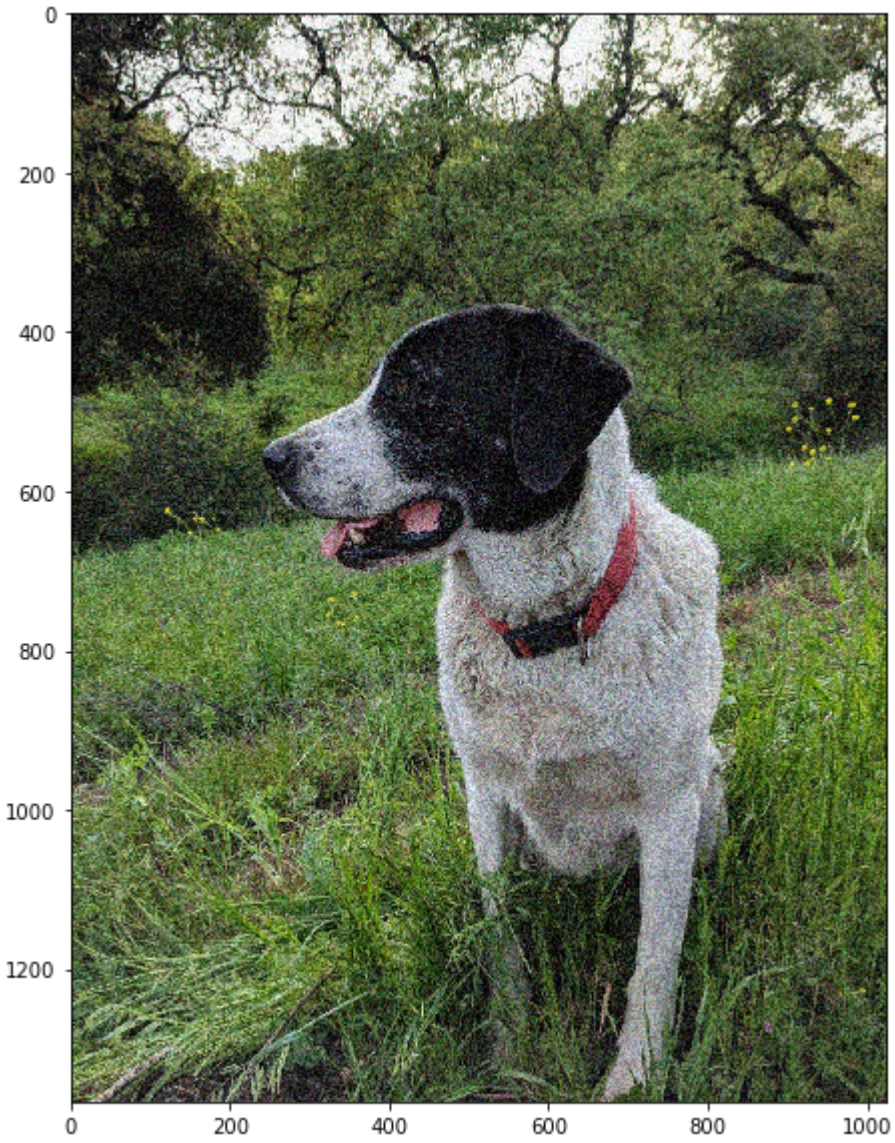
Out[135]: (1367, 1025, 3)

In [136]:
```
#dispalying the image
display_img(img)
```

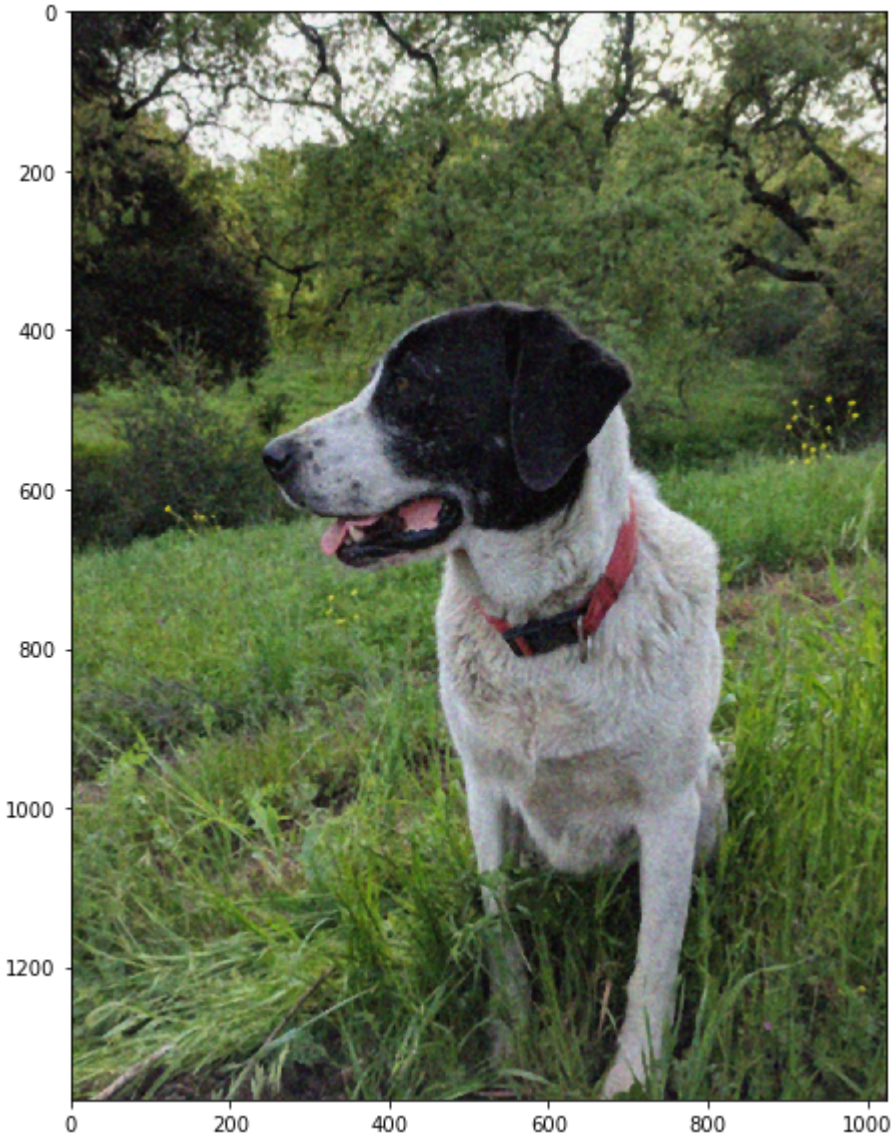


In [137]:
```
#reading the imagae
noise_img = cv2.imread('sammy_noise.jpg')
```

In [138]:
```
#displaying the iamge
display_img(noise_img)
```



In [139]:
```
#bluring the image with the noise image
median = cv2.medianBlur(noise_img,5)
#displaying the image
display_img(median)
```



Bilateral Filtering

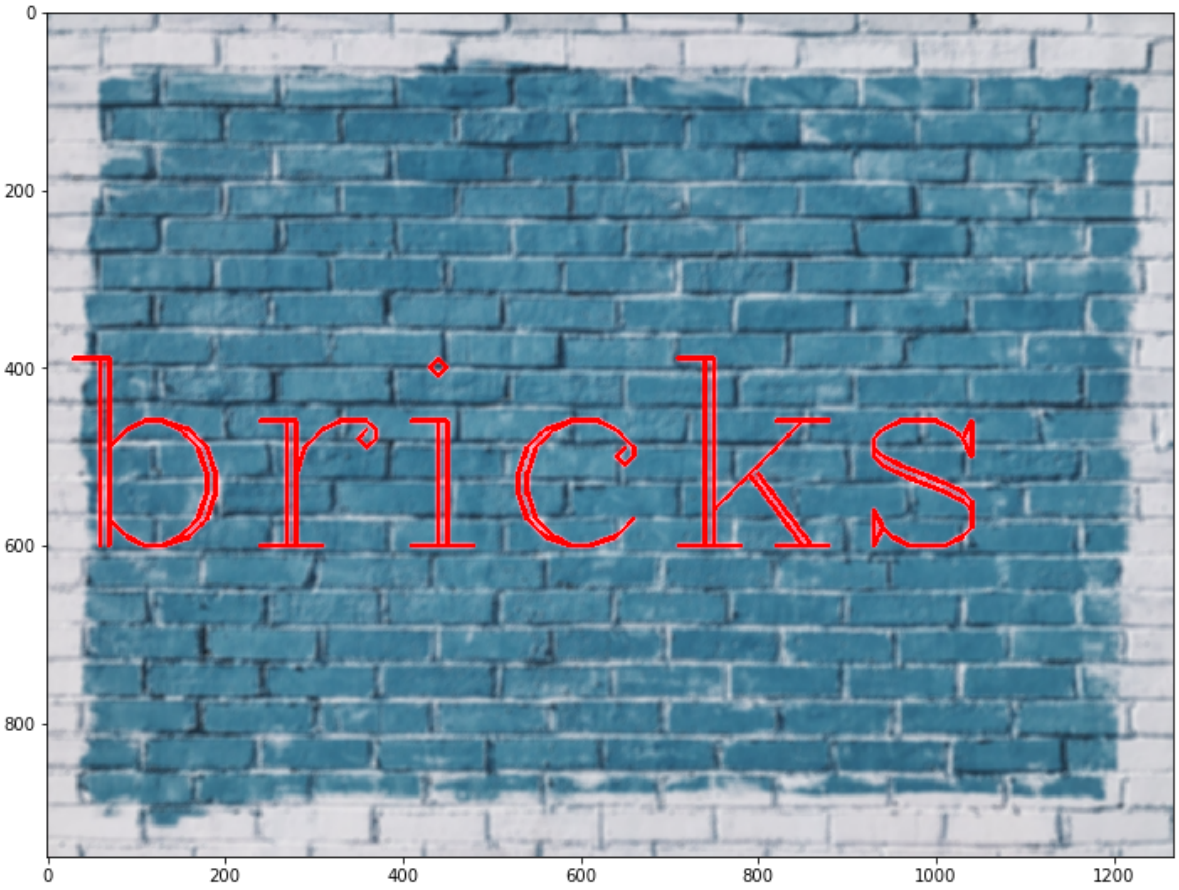
In [140]:
```
img = load_img()
font = cv2.FONT_HERSHEY_COMPLEX
cv2.putText(img,text='bricks',org=(10,600), fontFace=font,fontScale= 10,color=
(255,0,0),thickness=4)
display_img(img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

In [141]:
```
#another type of filter for smothness and blurring
blur = cv2.bilateralFilter(img,9,75,75)
```

In [142]:
```
#dispalying the image
display_img(blur)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



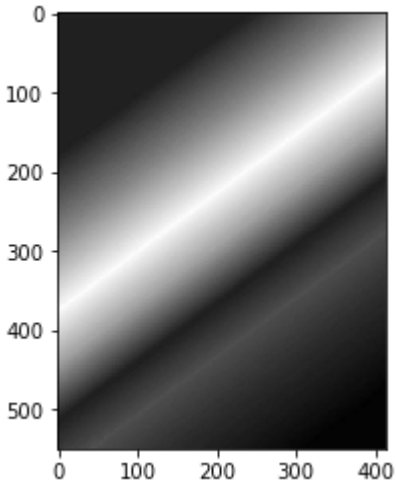
CONVERTING COLOUR IMAGE TO BLACK AND WHITE WITH CV2

In [144]:
```
#to convert the colour into perfet black and white image
#IMPORTING THE REQUIRED MODULES
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [146]:
```
#READING THE IMAGE
image1=cv2.imread("rainbow.jpg",0)
```

In [147]:
```
plt.imshow(image1,cmap="gray")
```

Out[147]: <matplotlib.image.AxesImage at 0x1a90b1aa3c8>



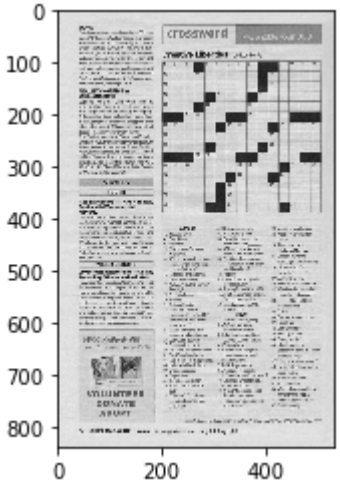
FUNCTION TO INCREASE THE SIZE OF IMAGE WITH CV2 ANF THRESHOLD WITH NEWSPAPER

In [151]:
```
#IMPORTING THE MODULES
import matplotlib.pyplot as plt
import cv2
```

In [152]:
```
#in the reading function we use 0 for black image
image1=cv2.imread("crossword.jpg",0)
```

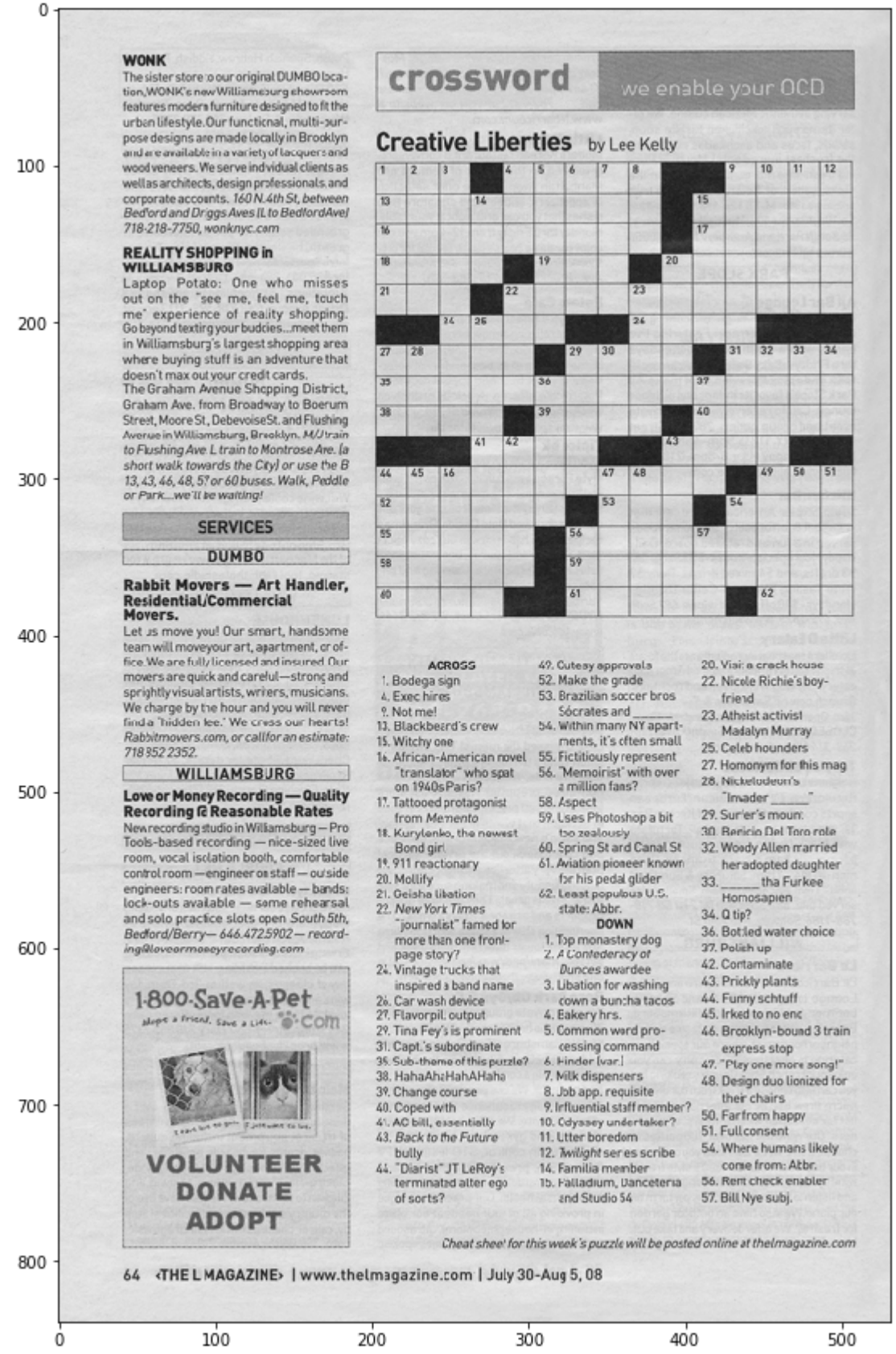
In [153]:
```
#cmap is used for black like image
plt.imshow(image1,cmap="gray")
```

Out[153]: <matplotlib.image.AxesImage at 0x1a904cd14a8>



In [154]:
```
#function to make the image bigger
def show_pic(image1):
    figure1=plt.figure(figsize=(15,15))
    ax=figure1.add_subplot(111)
    ax.imshow(image1,cmap="gray")
```

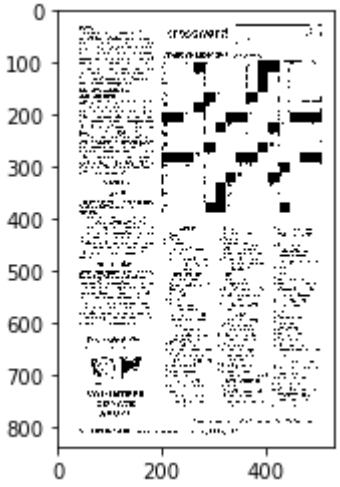
In [155]:
```
#displaying the image
show_pic(image1)
```



In [156]:
```
#threshoding the image
ret,threshold1=cv2.threshold(image1,127,255,cv2.THRESH_BINARY)
```

In [157]:
```
#displaying the image
plt.imshow(threshold1,cmap="gray")
```

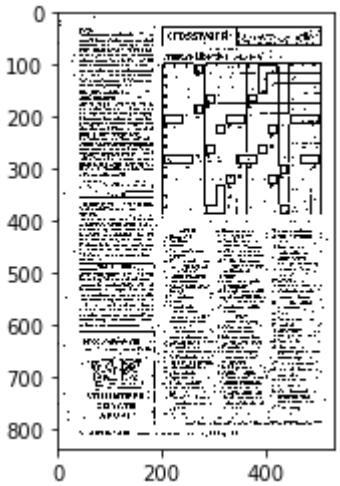
Out[157]: <matplotlib.image.AxesImage at 0x1a904690cf8>



In [158]:
```
#another type of threshold which is adaptiveThreshold
threshold2=cv2.adaptiveThreshold(image1,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THR
ESH_BINARY,11,8)
```

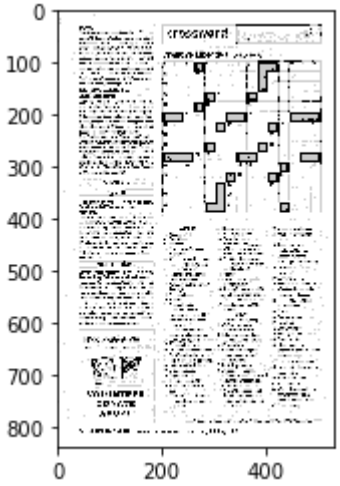
In [159]:
```
#displaying the image
plt.imshow(threshold2,cmap="gray")
```

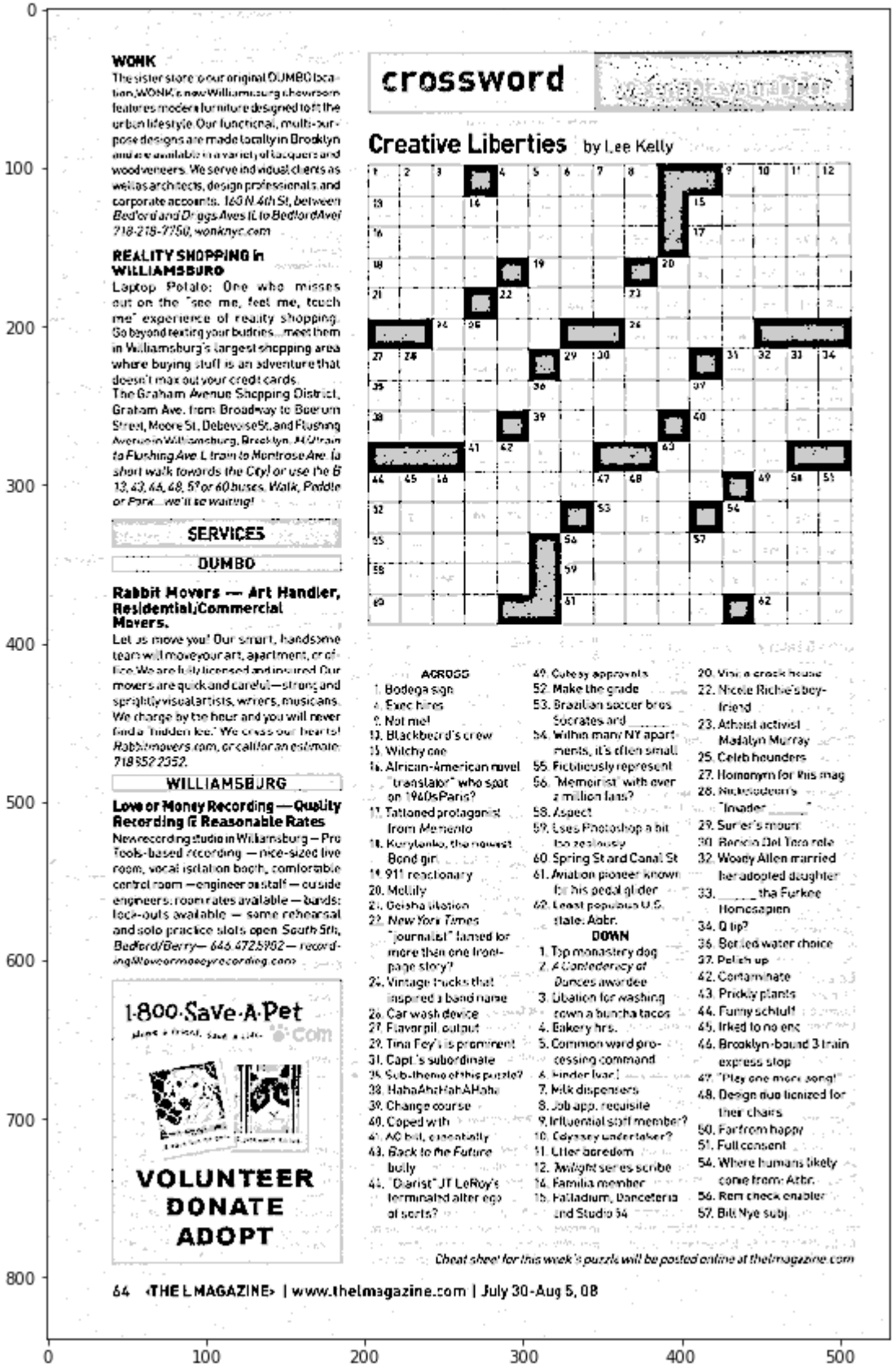
Out[159]: <matplotlib.image.AxesImage at 0x1a9030920f0>



In [160]:
```
#blending the image with both threshold1 and threshold2
blended=cv2.addWeighted(threshold1,0.8,threshold2,0.8,0)
```

In [161]:
```
#displaying the image
plt.imshow(blended,cmap="gray")
show_pic(blended)
```


0
100
200
300
400
500
600
700
800
0
200
400



WONK

The sister store of our original DUMBO location, WONK's new Williamsburg showroom features modern furniture designed to fit the urban lifestyle. Our functional, multi-purpose designs are made locally in Brooklyn and are available in a variety of lacquers and wood veneers. We serve individual clients as well as architects, design professionals and corporate accounts. *160 N 4th St, between Bedford and Driggs Aves (L to Bedford Ave) 718-218-7750, wonknyc.com*

REALITY SHOPPING in WILLIAMSBURG

Laptop Potato: One who misses out on the "see me, feel me, touch me" experience of reality shopping. Go beyond texting your buddies...meet them in Williamsburg's largest shopping area where buying stuff is an adventure that doesn't max out your credit cards.
The Graham Avenue Shopping District, Graham Ave. from Broadway to Boerum Street, Moore St. Debevoise St. and Flushing Avenue in Williamsburg, Brooklyn. *M (Train to Flushing Ave. L train to Montrose Ave. (a short walk towards the City) or use the B 13, 43, 44, 48, 57 or 60 buses. Walk, Peddle or Park...we'll be waiting!*

SERVICES

DUMBO

Rabbit Movers — Art Handler, Residential/Commercial Movers.

Let us move you! Our smart, handsome team will move your art, apartment, or office. We are fully licensed and insured. Our movers are quick and careful—strong and sprightly visual artists, writers, musicians. We charge by the hour and you will never find a "hidden fee." We cross our hearts! *Rabbitmovers.com, or call for an estimate 718 852 2352.*

WILLIAMSBURG

Love or Money Recording — Quality Recording @ Reasonable Rates

New recording studio in Williamsburg – Pro Tools-based recording — nice-sized live room, vocal isolation booth, comfortable control room — engineer as staff — outside engineers, room rates available — bands: lock-outs available — some rehearsal and solo practice slots open. *South 5th, Bedford/Berry— 646.472.5952 — recordingMoneormoneyrecording.com*

crossword

Creative Liberties by Lee Kelly

ACROSS

1. Bodega sign
4. Exec hires
9. Not me!
13. Blackbeard's crew
15. Witchy one
16. African-American novel "translator" who spat on 1960s Paris?
17. Tattooed protagonist from *Memento*
18. Konylenko, the newest Bond girl
19. 911 reactionary
20. Mellify
21. Geisha station
22. *New York Times* "journalist" famed for more than one front-page story?
24. Vintage trucks that inspired a band name
26. Car wash device
27. Flavorpill output
29. Tina Fey's is prominent
31. Capt.'s subordinate
35. Sub-theme of this puzzle?
38. HahaAhaHahAHaha
39. Change course
40. Coped with
41. AC bill, essentially
43. *Back to the Future* bully
44. "Diarist" JT LeRoy's terminated alter ego of sorts?
49. Cutesy approvals
52. Make the grade
53. Brazilian soccer bros Socrates and _____
54. Within many NY apartments, it's often small
55. Fictitiously represent
56. "Memorist" with over a million fans?
58. Aspect
59. Uses Photoshop a bit too zealously
60. Spring St and Canal St
61. Aviation pioneer known for his pedal glider
62. Least populous U.S. state: Abbr.

DOWN

1. Top monastery dog
2. *A Confederacy of Dunces* awardee
3. Libation for washing down a buncha tacos
4. Bakery hrs.
5. Common word processing command
6. Kinder bud
7. Milk dispensers
8. Job app. requisite
9. Influential staff member?
10. Odyssey undertaker?
11. Utter boredom
12. *Twilight* series scribe
14. Familia member
15. Palladium, Danceteria and Studio 54
20. Visit a crack house
22. Nicole Richie's boyfriend
23. Atheist activist Madalyn Murray
25. Celeb hounders
27. Homonym for this mag
28. Nickelodeon's "Invader _____"
29. Sumer's mount
30. Benicio Del Toro role
32. Woody Allen married her adopted daughter
33. _____ tha Furkee Homosapien
34. Q tip?
36. Bottled water choice
37. Polish up
42. Contaminate
44. Prickly plants
45. Inked to no one
46. Brooklyn-bound 3 train express stop
47. "Play one more song!"
48. Design duo lionized for their chairs
50. Forlorn happy
51. Full consent
54. Where humans likely come from: Abbr.
56. Rent check enabler
57. Bill Nye subj.

Cheat sheet for this week's puzzle will be posted online at thelmagazine.com

HSV AND HSL CONVERSION

In [163]:
```
#importing the required module
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [164]:
```
#reading the image
img1=cv2.imread("uspassport.jpg")
```

In [165]:
```
#viewing the type
type(img1)
```

Out[165]: numpy.ndarray

In [166]:
```
#viewing the image
plt.imshow(img1)
```

Out[166]: <matplotlib.image.AxesImage at 0x1a902ac82b0>

In [167]:
```
#viewing the original image by changing the colour with cvtColor
original_image=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
```

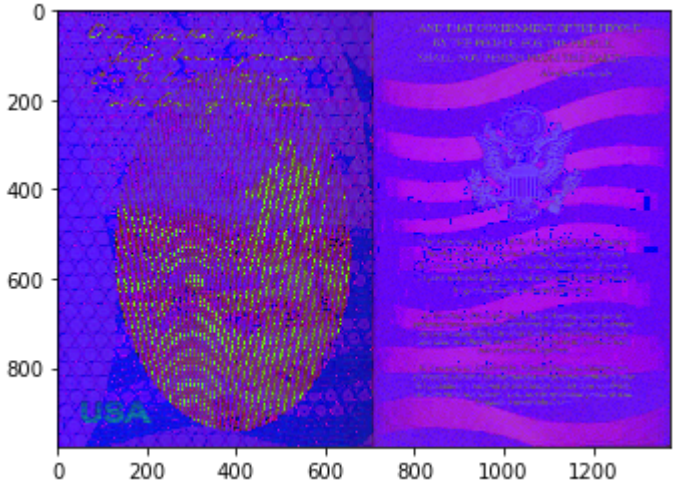
In [168]:
```
#viewing the image
plt.imshow(original_image)
```

Out[168]: <matplotlib.image.AxesImage at 0x1a904cb2da0>

In [169]:
```
#Converting the image from BGR2HSV with cvtColor
img2=cv2.cvtColor(img1,cv2.COLOR_BGR2HSV)
```

In [170]:
```
plt.imshow(img2)
```

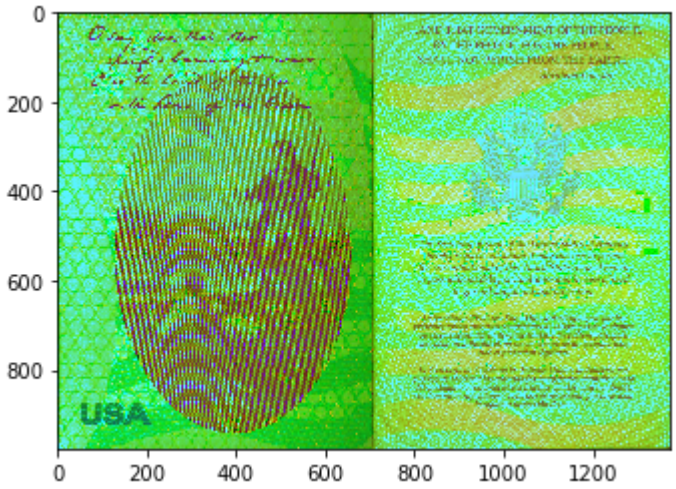
Out[170]: <matplotlib.image.AxesImage at 0x1a904119e10>



In [171]:
```
#Converting the image from BGR2HLS with cvtColor
img3=cv2.cvtColor(img1,cv2.COLOR_BGR2HLS)
```

In [172]:
```
#displaying the image
plt.imshow(img3)
```

Out[172]: <matplotlib.image.AxesImage at 0x1a904793eb8>



OVERLAPPING SMALLER IMAGER TO LAGER IMAGES

In [173]:
```
# overlapping one smaller image to another larger image of different size
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [174]:
```
#reading the image
image1=cv2.imread("dog_backpack.png")
```

In [175]:
```
type(image1)
```

Out[175]: numpy.ndarray

In [179]:
```
#reading the image
image2=cv2.imread("watermark_no_copy.png")
```

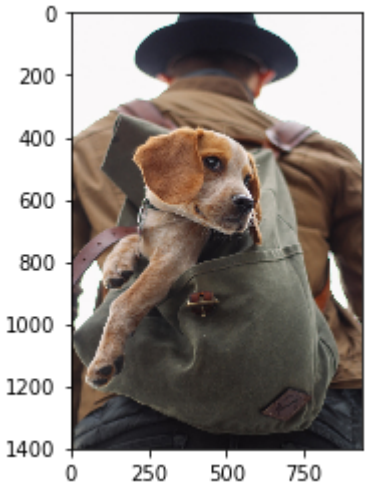
In [180]:
```
type(image2)
```

Out[180]: numpy.ndarray

In [181]:
```
#converting BLUE GREEN RED TO RED GREEN BLUE (ORIGINAL IMAGE)
image1=cv2.cvtColor(image1,cv2.COLOR_BGR2RGB)
image2=cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
```

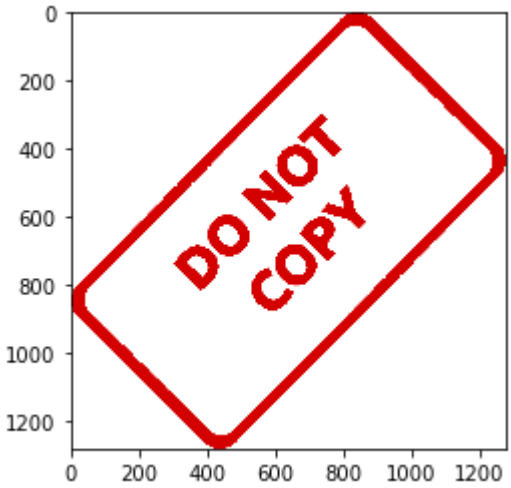
In [182]:
```
#displaying the image with matplotlib
plt.imshow(image1)
```

Out[182]: <matplotlib.image.AxesImage at 0x1a9048822e8>



In [183]:
```
plt.imshow(image2)
```

Out[183]: <matplotlib.image.AxesImage at 0x1a9049807b8>



In [184]:
```
#resizing the image2 to 400 (to make it small)
image2=cv2.resize(image2,(400,400))
```

In [185]:
```
#Process of combining  two images of different size
larger_image=image1
smaller_image=image2
```

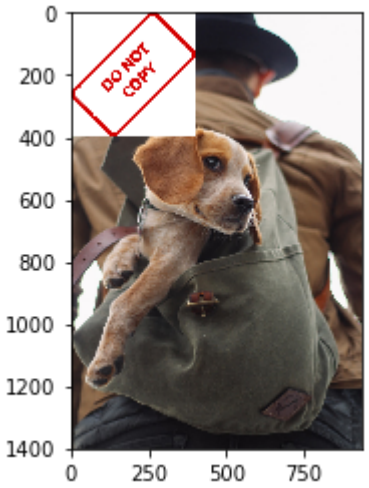
In [186]:
```
x_offset=0
y_offset=0
```

In [187]:
```
x_end=x_offset+smaller_image.shape[1]
y_end=y_offset+smaller_image.shape[0]
```

In [188]:
```
#method to implant smaller image to larger image image
larger_image[y_offset:y_end,x_offset:x_end]=smaller_image
```

In [189]:
```
plt.imshow(larger_image)
```

Out[189]: <matplotlib.image.AxesImage at 0x1a904174eb8>



OPENING THE CAMERA

In [190]:
```
import cv2
#to get default camera of the pc
capture=cv2.VideoCapture(0)

#to retrive the width of the screen #make sure it contains int
width=int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
#to retrive the height of the screen #make sure it contains int
height=int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))

while True:
    #reading the frame
    ret,frame=capture.read()
    #showing the frame
    cv2.imshow("frame",frame)
    #exiting with q key
    if cv2.waitKey(1) & 0xFF== ord('q'):
        break
# When everything done, release the capture and destroy the windows
capture.release()
cv2.destroyAllWindows
```

Out[190]: <function destroyAllWindows>

OPENING VIDEOFILES

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

# FRAMES PER SECOND FOR VIDEO
fps = 25

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


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



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

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




         # Display the frame at same frame rate of recording
        # Watch lecture video for full explanation
        time.sleep(1/fps)
        cv2.imshow('frame',frame)

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

            break

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

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

RECORDING THE LIVE CAMERA

In [2]:
```
import cv2

cap = cv2.VideoCapture(0)

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


# MACOS AND LINUX: *'XVID' (MacOS users may want to try VIDX as well just in c
ase)
# WINDOWS *'VIDX'
writer = cv2.VideoWriter('student_capture.mp4', cv2.VideoWriter_fourcc(*'VIDX'
),25, (width, height))


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

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


    # Write the video
    writer.write(frame)

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

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


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

DRAWING CIRCLE IN LIVE CAMERA

In [3]:
```
import cv2

cap = cv2.VideoCapture(0)


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

# We're using // here because in Python // allows for int classical division,
# because we can't pass a float to the cv2.circle function



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

    # Draw a rectangle on stream
    cv2.circle(frame, center=(100,100),radius=100, color=(0,0,255),thickness=
4)


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

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

# When everything is done, release the capture
cap.release()
cv2.destroyAllWindows()
```

RECTANGLE IN LIVE CAMERA

In [4]:
```
import cv2

cap = cv2.VideoCapture(0)


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

# We're using // here because in Python // allows for int classical division,
# because we can't pass a float to the cv2.rectangle function

# Coordinates for Rectangle
x = width//2
y = height//2

# Width and height
w = width//4
h = height//4

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

    # Draw a rectangle on stream

    cv2.rectangle(frame, (x, y), (x+w, y+h), color=(0,0,255),thickness= 4)

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

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

# When everything is done, release the capture
cap.release()
cv2.destroyAllWindows()
```

DRAWING ON CAMERA

In [6]:
```
import cv2
# Create a function based on a CV2 Event (Left button click)

# mouse callback function
def draw_rectangle(event,x,y,flags,param):

    global pt1,pt2,topLeft_clicked,botRight_clicked

    # get mouse click
    if event == cv2.EVENT_LBUTTONDOWN:

        if topLeft_clicked == True and botRight_clicked == True:
            topLeft_clicked = False
            botRight_clicked = False
            pt1 = (0,0)
            pt2 = (0,0)

        if topLeft_clicked == False:
            pt1 = (x,y)
            topLeft_clicked = True

        elif botRight_clicked == False:
            pt2 = (x,y)
            botRight_clicked = True


# Haven't drawn anything yet!

pt1 = (0,0)
pt2 = (0,0)
topLeft_clicked = False
botRight_clicked = False

cap = cv2.VideoCapture(0)

# Create a named window for connections
cv2.namedWindow('Test')

# Bind draw_rectangle function to mouse cliks
cv2.setMouseCallback('Test', draw_rectangle)


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

    if topLeft_clicked:
        cv2.circle(frame, center=pt1, radius=5, color=(0,0,255), thickness=-1)

    #drawing rectangle
    if topLeft_clicked and botRight_clicked:
        cv2.rectangle(frame, pt1, pt2, (0, 0, 255), 2)


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

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

# When everything is done, release the capture
cap.release()
cv2.destroyAllWindows()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-6-f648f8e5b4cb> in <module>
     32 botRight_clicked = False
     33
---> 34 cap = cv2.VideoCapture(0)
     35
     36 # Create a named window for connections

KeyboardInterrupt:
```

Template-Matching

In [1]:
```
#importing the modules
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
#reading the image and converting to original color
full_image=cv2.imread("sammy.jpg")
full_image=cv2.cvtColor(full_image,cv2.COLOR_BGR2RGB)
```

In [3]:
```
type(full_image)
```

Out[3]:
```
numpy.ndarray
```

In [4]:
```
face_image=cv2.imread("sammy_face.jpg")
```

In [5]:
```
type(face_image)
```

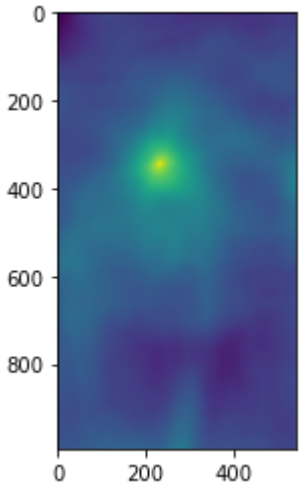
Out[5]:
```
numpy.ndarray
```

In [6]:
```
#all the list of methods in templete matching
methods= ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR','cv2.TM_CCOR
R_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']
```

In [7]:
```
#using matchTemplate function in cv2 module
res=cv2.matchTemplate(full_image,face_image,cv2.TM_CCOEFF)
```

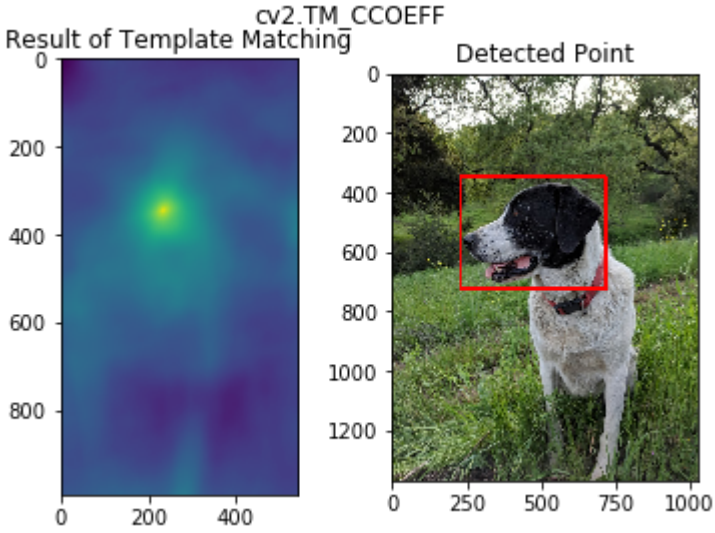
In [8]:
```
plt.imshow(res)
```

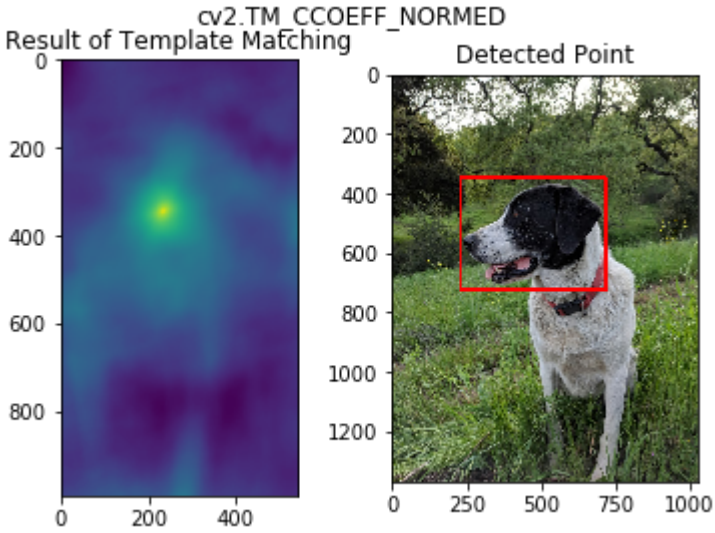
Out[8]: <matplotlib.image.AxesImage at 0x230e7a84c50>

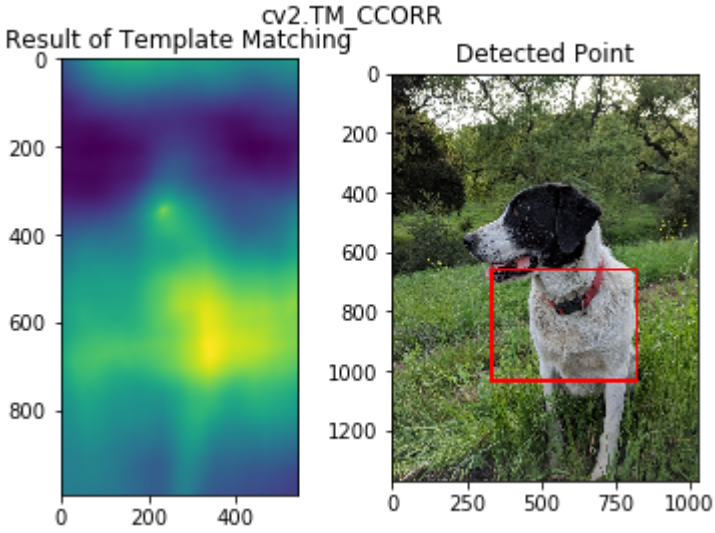


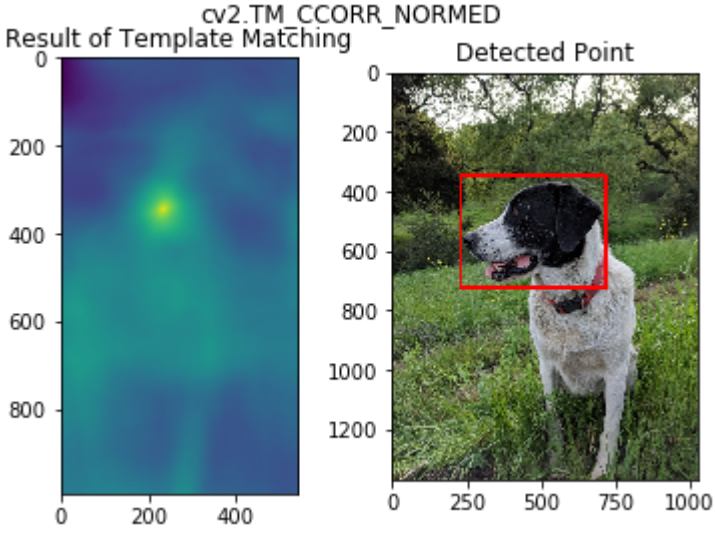
In [9]:
```
height, width,channels = face_image.shape
```

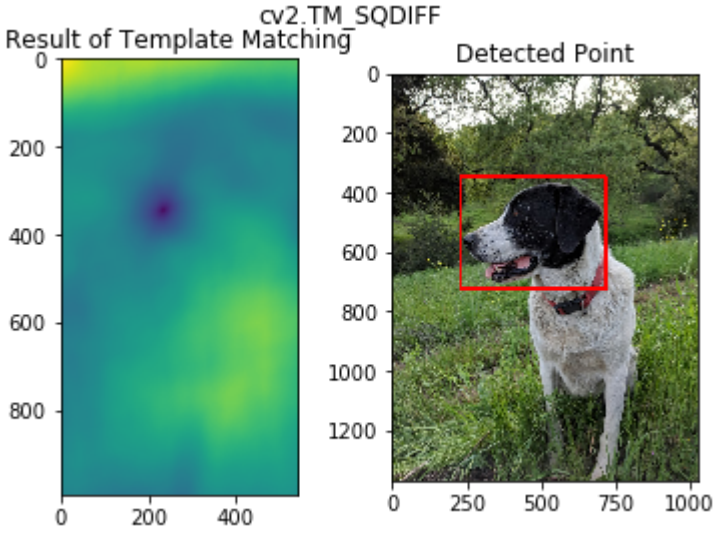
In [10]:
```
for m in methods:

    # Create a copy of the image
    full_copy = full_image.copy()

    # Get the actual function instead of the string
    method = eval(m)

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

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

    # Set up drawing of Rectangle

    # If the method is TM_SQDIFF or TM_SQDIFF_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(full_copy,top_left, bottom_right, 255, 10)

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

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


    plt.show()
    print('\n')
    print('\n')
```

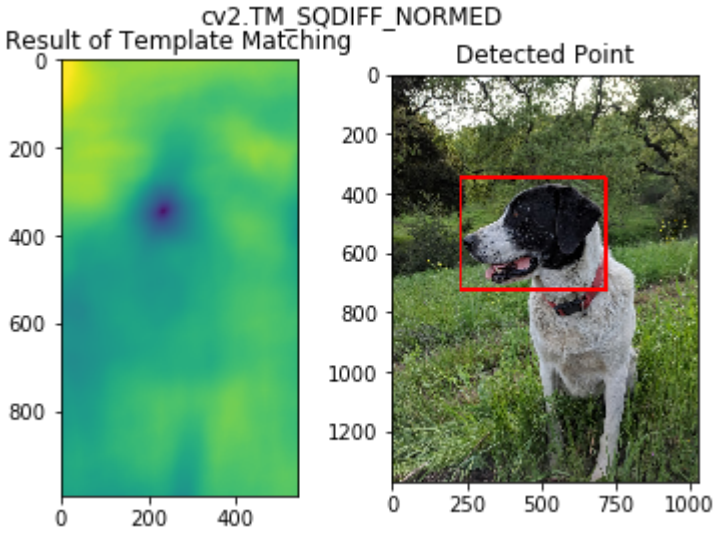

cv2.TM_CCOEFF
Result of Template Matching
Detected Point


cv2.TM_CCOEFF_NORMED
Result of Template Matching
Detected Point


cv2.TM_CCORR
Result of Template Matching
Detected Point
0
200
400
600
800
1000
1200
0
250
500
750
1000


cv2.TM_CCORR_NORMED
Result of Template Matching
Detected Point
0
200
400
600
800
1000
1200
0
250
500
750
1000





HARRIS CORNER DETECTION

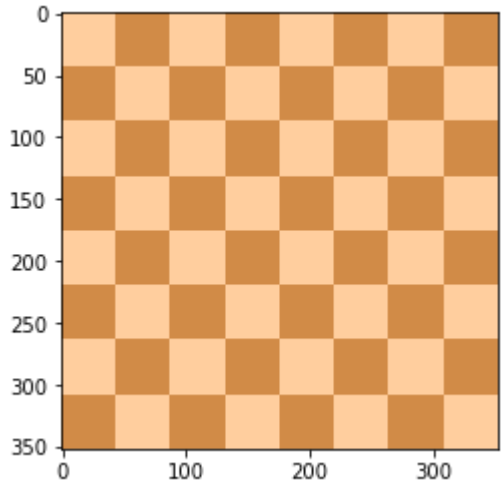
In [2]:
```
#importing the module
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [21]:
```
#reading the image
flat_chess=cv2.imread("flat_chessboard.png")
```

In [28]:
```
#converting the color
flat_chess=cv2.cvtColor(flat_chess,cv2.COLOR_BGR2RGB)
```

In [29]:
```
#displaying the image
plt.imshow(flat_chess)
```

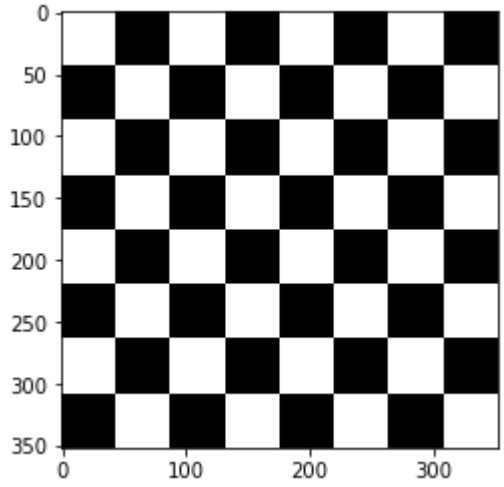
Out[29]: <matplotlib.image.AxesImage at 0x2f89f891048>



In [30]:
```
#converting the original image to black and white image
gray_flat_chess=cv2.cvtColor(gray_flat_chess,cv2.COLOR_RGB2GRAY)
```

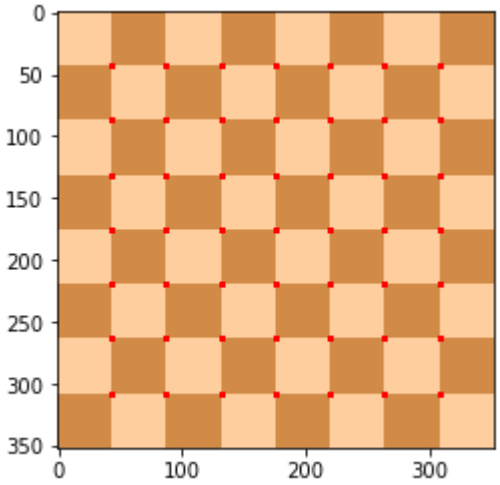
In [31]:
```
#displaying the image remember that to make it black we need to use cmap="gra
y" in plt.imshow
plt.imshow(gray_flat_chess,cmap="gray")
```

Out[31]: <matplotlib.image.AxesImage at 0x2f89f3fa7f0>



In [32]:
```
# Convert Gray Scale Image to Float Values
gray = np.float32(gray_flat_chess)

# Corner Harris Detection
dst = cv2.cornerHarris(src=gray,blockSize=2,ksize=3,k=0.04)

# result is dilated for marking the corners, not important to actual corner de
tection
# this is just so we can plot out the points on the image shown
dst = cv2.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on the image.
flat_chess[dst>0.01*dst.max()]=[255,0,0]

plt.imshow(flat_chess)
```

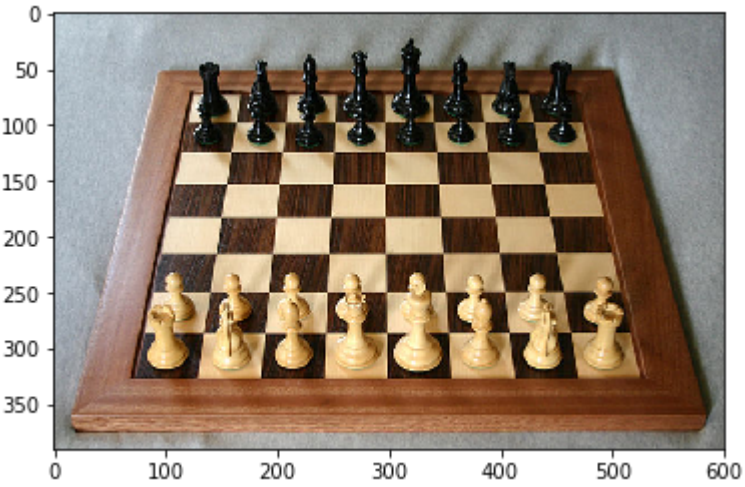
Out[32]: <matplotlib.image.AxesImage at 0x2f89f519940>



In [33]:
```
#opening the image and convertig to original image
real_chess_board=cv2.imread("real_chessboard.jpg")
real_chess_board=cv2.cvtColor(real_chess_board,cv2.COLOR_BGR2RGB)
```

In [34]:
```
#displaying the image
plt.imshow(real_chess_board)
```

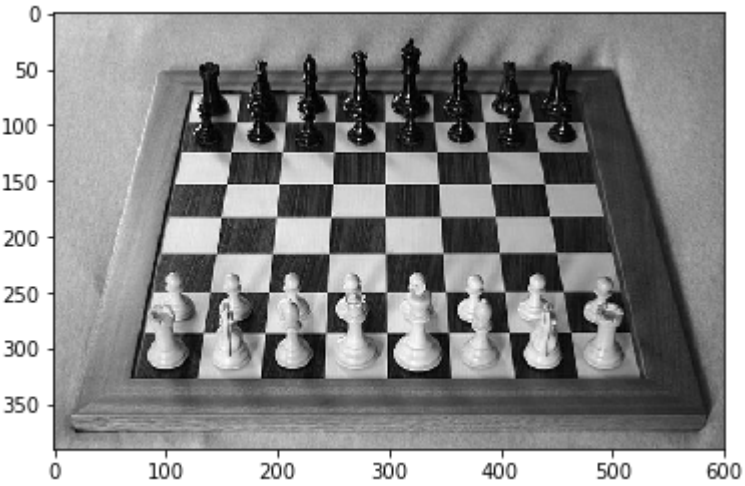
Out[34]: <matplotlib.image.AxesImage at 0x2f89f970978>



In [36]:
```
#converting the image to black and white
gray_real_chess_board=cv2.cvtColor(real_chess_board,cv2.COLOR_RGB2GRAY)
```

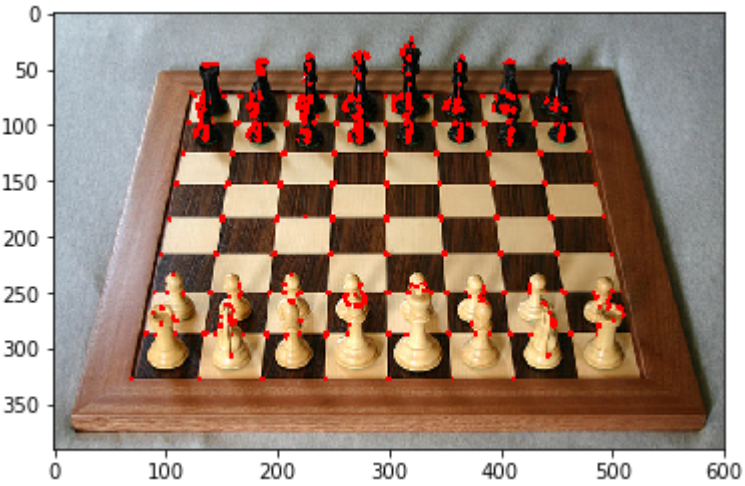
In [37]:
```
plt.imshow(gray_real_chess_board,cmap="gray")
```

Out[37]: <matplotlib.image.AxesImage at 0x2f89fe98630>



In [44]:
```
gray = np.float32(gray_real_chess_board)

# Corner Harris Detection
dst = cv2.cornerHarris(src=gray,blockSize=2,ksize=3,k=0.04)

# result is dilated for marking the corners, not important to actual corner de
tection
# this is just so we can plot out the points on the image shown
dst = cv2.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on the image.
real_chess_board[dst>0.01*dst.max()]=[255,0,0]

plt.imshow(real_chess_board)
```

Out[44]: <matplotlib.image.AxesImage at 0x2f89fe212e8>



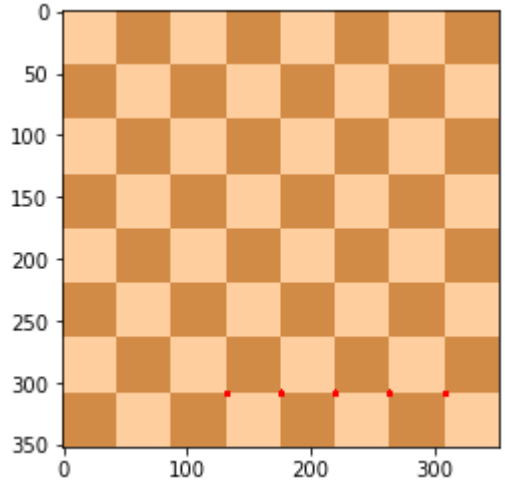
Shi-Tomasi Detection

In [2]:
```
#importing the modules
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```
#opening the images
flat_chess = cv2.imread('flat_chessboard.png')
flat_chess = cv2.cvtColor(flat_chess,cv2.COLOR_BGR2RGB)
gray_flat_chess = cv2.cvtColor(flat_chess,cv2.COLOR_BGR2GRAY)
```

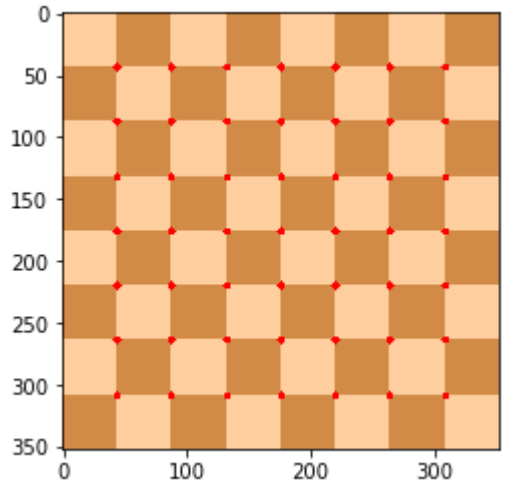
In [4]:
```
corners = cv2.goodFeaturesToTrack(gray_flat_chess,5,0.01,10)
corners = np.int0(corners)

for i in corners:
    x,y = i.ravel()
    cv2.circle(flat_chess,(x,y),3,255,-1)

plt.imshow(flat_chess)
```

Out[4]: <matplotlib.image.AxesImage at 0x19e18e8b470>



In [5]:
```
corners = cv2.goodFeaturesToTrack(gray_flat_chess,64,0.01,10)
corners = np.int0(corners)

for i in corners:
    x,y = i.ravel()
    cv2.circle(flat_chess,(x,y),3,255,-1)

plt.imshow(flat_chess)
```

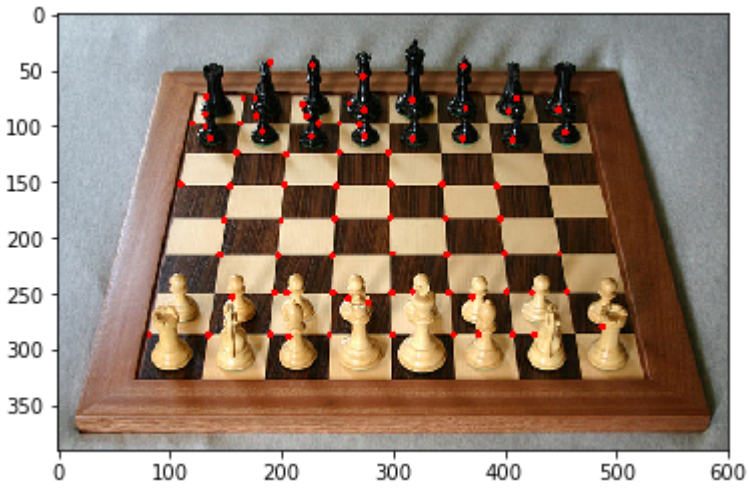
Out[5]: <matplotlib.image.AxesImage at 0x19e18f1af28>



In [6]:
```
real_chess = cv2.imread('real_chessboard.jpg')
real_chess = cv2.cvtColor(real_chess,cv2.COLOR_BGR2RGB)
gray_real_chess = cv2.cvtColor(real_chess,cv2.COLOR_BGR2GRAY)
```

In [7]:
```
corners = cv2.goodFeaturesToTrack(gray_real_chess,80,0.01,10)
corners = np.int0(corners)

for i in corners:
    x,y = i.ravel()
    cv2.circle(real_chess,(x,y),3,255,-1)

plt.imshow(real_chess)
```

Out[7]: <matplotlib.image.AxesImage at 0x19e18f819e8>



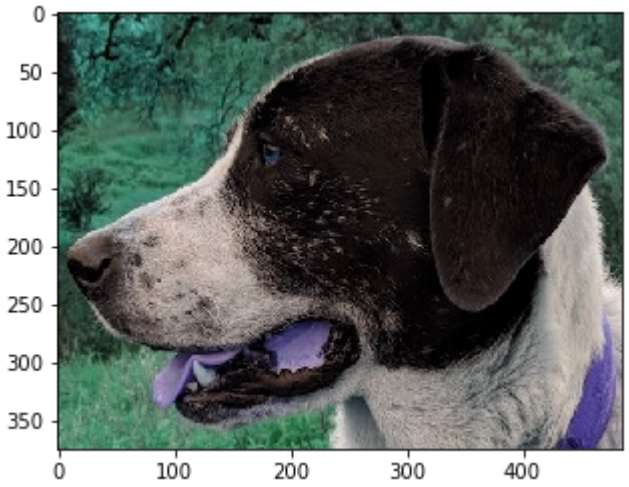
Canny edge detection

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
img = cv2.imread('sammy_face.jpg')
```

In [3]:
```
plt.imshow(img)
```

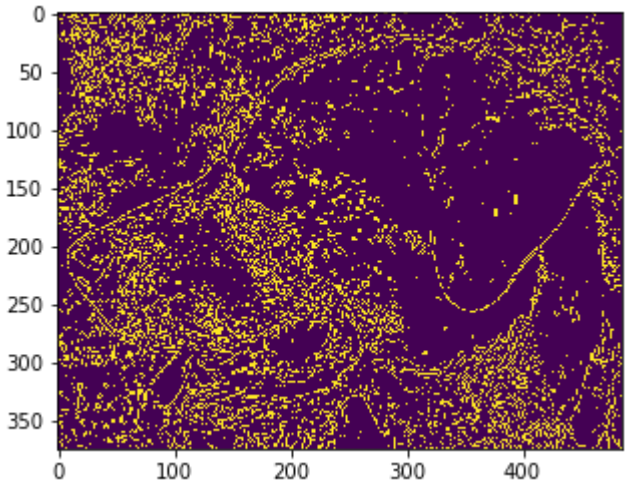
Out[3]: <matplotlib.image.AxesImage at 0x1fed8a970b8>



In [4]:
```
edges = cv2.Canny(image=img, threshold1=127, threshold2=127)
```

In [5]:
```
plt.imshow(edges)
```

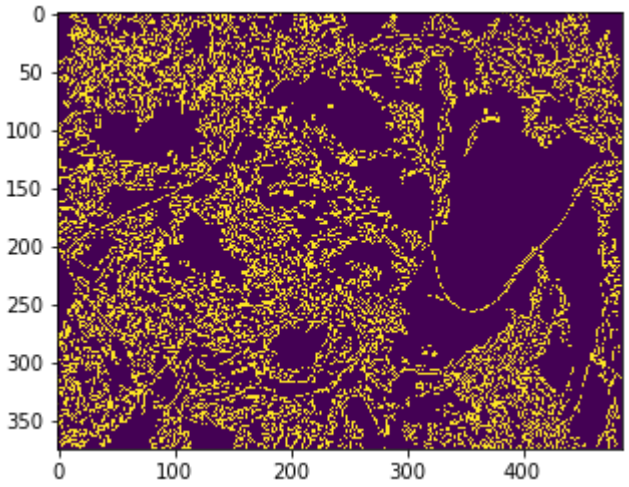
Out[5]: <matplotlib.image.AxesImage at 0x1fed8b2fcf8>



In [6]:
```
edges = cv2.Canny(image=img, threshold1=0, threshold2=255)
```

In [7]:
```
plt.imshow(edges)
```

Out[7]: <matplotlib.image.AxesImage at 0x1fed8b94c50>



choosing thresholds

In [8]:
```
# Calculate the median pixel value
med_val = np.median(img)
```

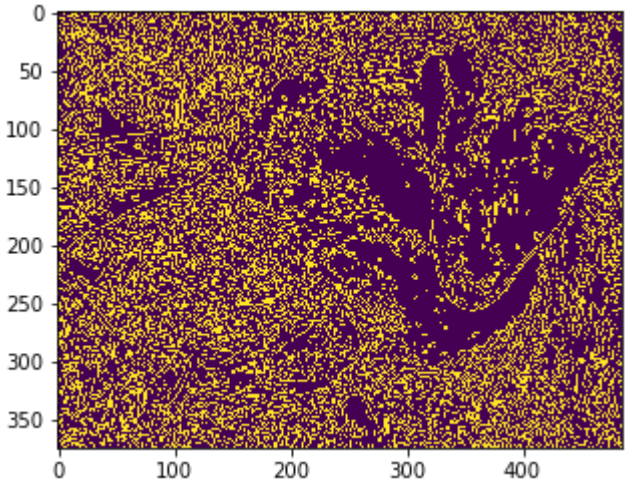
In [9]:
```
# Lower bound is either 0 or 70% of the median value, whicever is higher
lower = int(max(0, 0.7* med_val))
```

In [10]:
```
# Upper bound is either 255 or 30% above the median value, whichever is lower
upper = int(min(255,1.3 * med_val))
```

In [11]:
```
edges = cv2.Canny(image=img, threshold1=lower , threshold2=upper)
```

In [13]:
```
plt.imshow(edges)
```

Out[13]: <matplotlib.image.AxesImage at 0x1fed8c4d358>



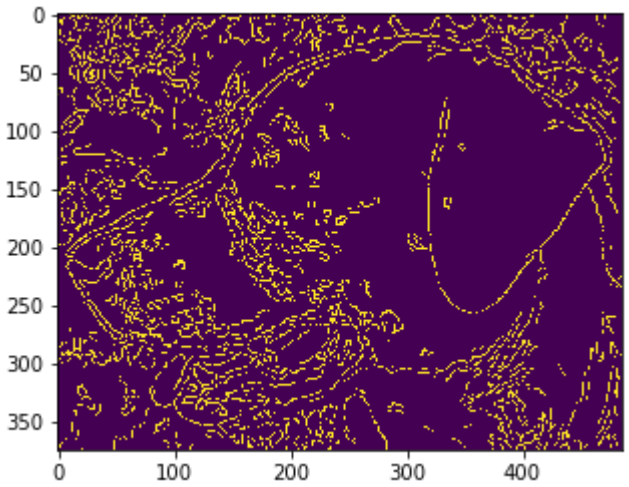
Sometimes it helps to blur the images first, so we don't pick up minor edges.

In [15]:
```
blurred_img = cv2.blur(img,ksize=(5,5))
```

In [16]:
```
edges = cv2.Canny(image=blurred_img, threshold1=lower , threshold2=upper)
```

In [17]:
```
plt.imshow(edges)
```

Out[17]: <matplotlib.image.AxesImage at 0x1fed8ca39e8>



In [18]:
```
upper
```

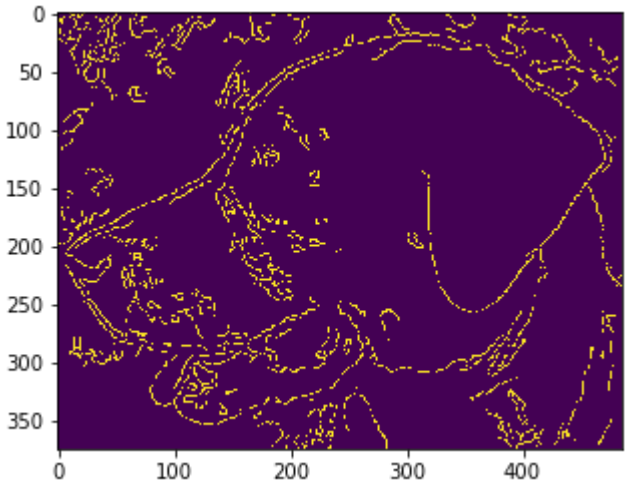
Out[18]: 83

In [19]:
```
lower
```

Out[19]: 44

In [20]:
```
edges = cv2.Canny(image=blurred_img, threshold1=lower , threshold2=upper+50)
plt.imshow(edges)
```

Out[20]: <matplotlib.image.AxesImage at 0x1fed8d424e0>



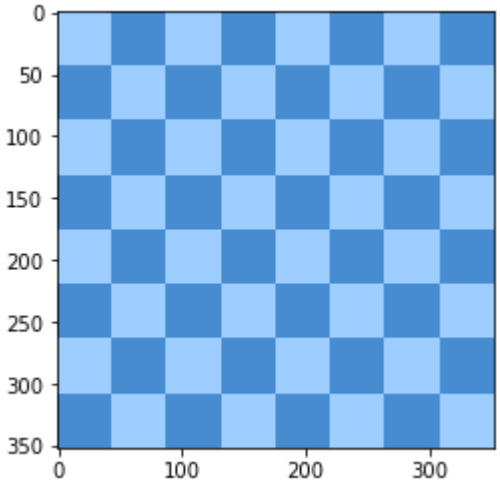
Grid Detection

In [21]:
```
import cv2
import matplotlib.pyplot as plt
```

In [24]:
```
flat_chess=cv2.imread("flat_chessboard-Copy1.png")
```

In [25]:
```
plt.imshow(flat_chess,cmap="gray")
```

Out[25]:
```
<matplotlib.image.AxesImage at 0x1fed8d9b9b0>
```



In [26]:
```
found, corners = cv2.findChessboardCorners(flat_chess,(7,7))
```

In [27]:
```
if found:
    print('OpenCV was able to find the corners')
else:
    print("OpenCV did not find corners. Double check your patternSize.")
```

```
OpenCV was able to find the corners
```

In [28]:
```
corners.shape
```

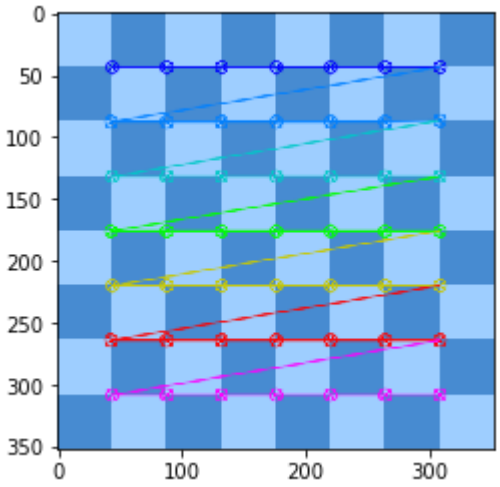
Out[28]:
```
(49, 1, 2)
```

In [29]:
```
flat_chess_copy = flat_chess.copy()
cv2.drawChessboardCorners(flat_chess_copy, (7, 7), corners, found)
```

Out[29]:
```
array([[[158, 206, 255],
        [158, 206, 255],
        [158, 206, 255],
        ...,
        [ 71, 139, 209],
        [ 71, 139, 209],
        [ 71, 139, 209]],

       [[158, 206, 255],
        [158, 206, 255],
        [158, 206, 255],
        ...,
        [ 71, 139, 209],
        [ 71, 139, 209],
        [ 71, 139, 209]],

       [[158, 206, 255],
        [158, 206, 255],
        [158, 206, 255],
        ...,
        [ 71, 139, 209],
        [ 71, 139, 209],
        [ 71, 139, 209]],

       ...,

       [[ 71, 139, 209],
        [ 71, 139, 209],
        [ 71, 139, 209],
        ...,
        [158, 206, 255],
        [158, 206, 255],
        [158, 206, 255]],

       [[ 71, 139, 209],
        [ 71, 139, 209],
        [ 71, 139, 209],
        ...,
        [158, 206, 255],
        [158, 206, 255],
        [158, 206, 255]],

       [[ 71, 139, 209],
        [ 71, 139, 209],
        [ 71, 139, 209],
        ...,
        [158, 206, 255],
        [158, 206, 255],
        [158, 206, 255]]], dtype=uint8)
```

In [31]:
```
plt.imshow(flat_chess_copy)
```

Out[31]: <matplotlib.image.AxesImage at 0x1fed8e40a20>

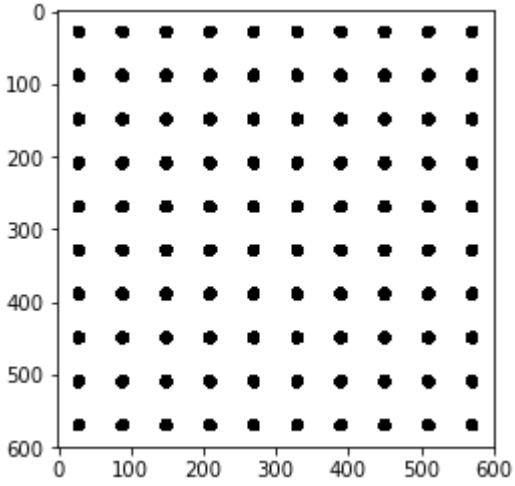


Circle Based Grids

In [33]:
```
dots=cv2.imread("dot_grid.png")
```

In [34]:
```
plt.imshow(dots)
```

Out[34]: <matplotlib.image.AxesImage at 0x1fed9e68b70>



In [35]:
```
found, corners = cv2.findCirclesGrid(dots, (10,10), cv2.CALIB_CB_SYMMETRIC_GRID)
```

In [36]:
```
found
```

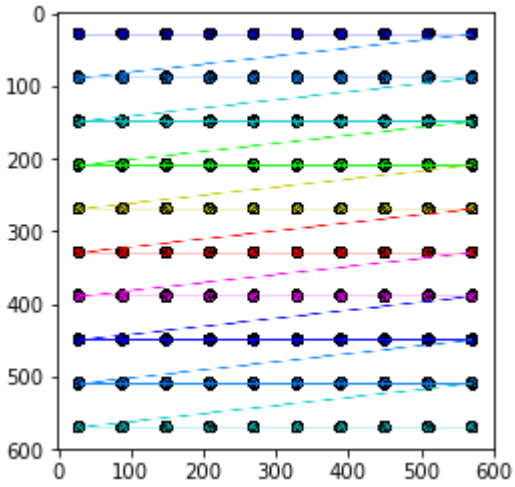
Out[36]: True

In [37]:
```
dbg_image_circles = dots.copy()
cv2.drawChessboardCorners(dbg_image_circles, (10, 10), corners, found)
```

Out[37]:
```
array([[[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       ...,

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]]], dtype=uint8)
```

In [38]:
```
plt.imshow(dbg_image_circles)
```

Out[38]: <matplotlib.image.AxesImage at 0x1fedbc61908>



FEATURE MATCHING

In [4]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

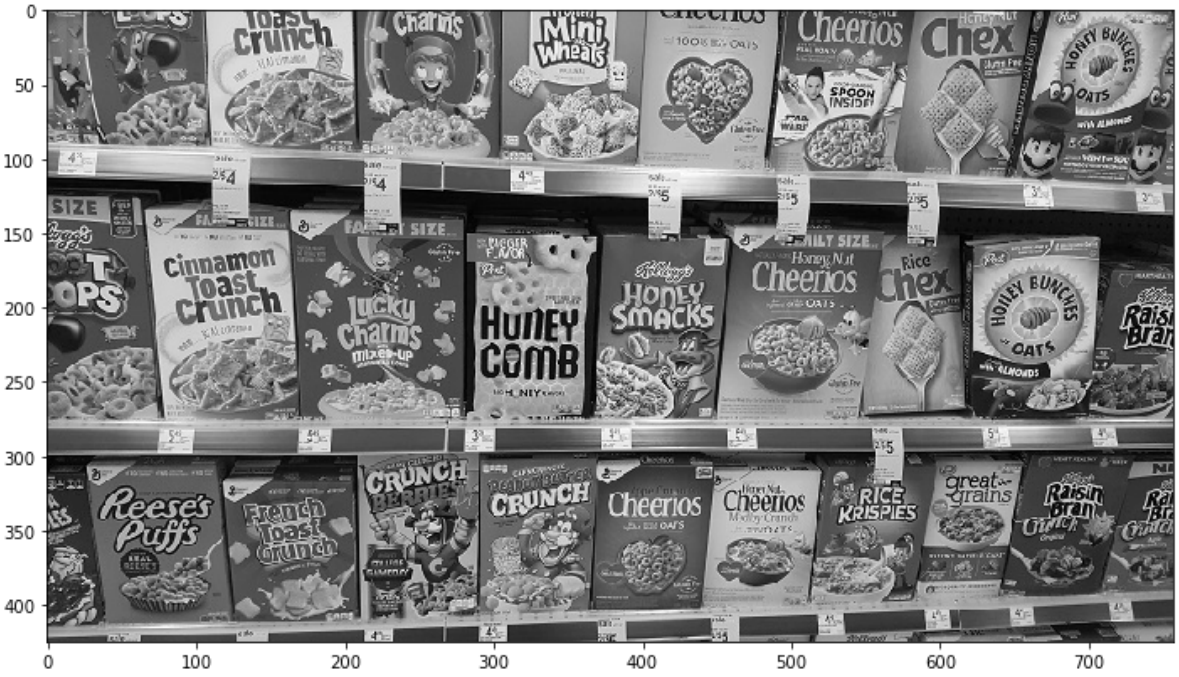
In [5]:
```
def display(img,cmap='gray'):
    fig = plt.figure(figsize=(12,10))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap='gray')
```

In [6]:
```
reeses=cv2.imread("reeses_puffs.png",0)
```

In [7]:
```
display(reeses)
```

In [8]:
```
cereals=cv2.imread("many_cereals.jpg",0)
```

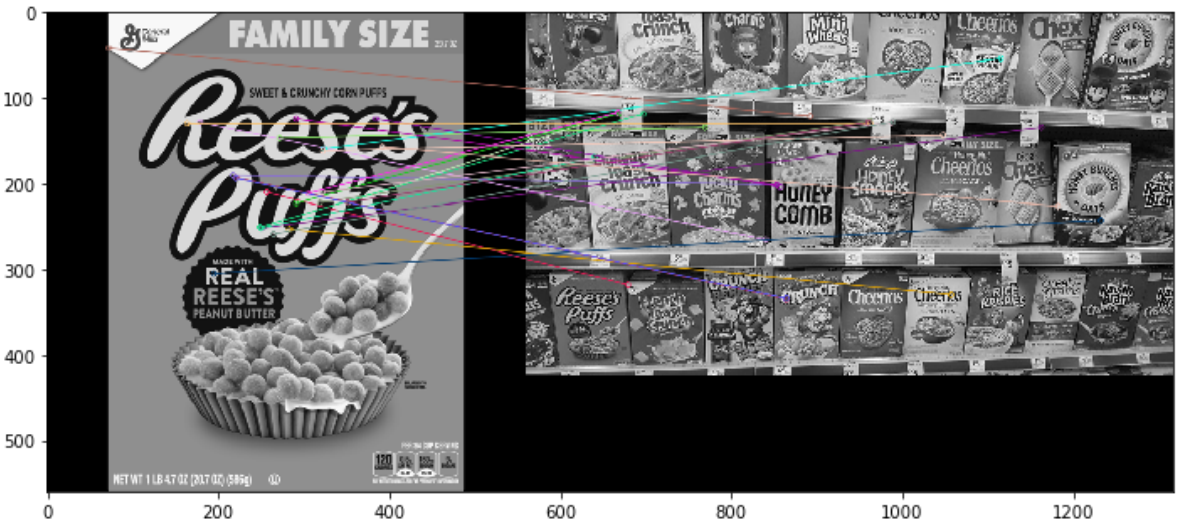
In [9]:
```
display(cereals)
```



Brute Force Detection with ORB Descriptors

In [10]:
```
# Initiate ORB detector
orb = cv2.ORB_create()

# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(reeses,None)
kp2, des2 = orb.detectAndCompute(cereals,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 25 matches.
reeses_matches = cv2.drawMatches(reeses,kp1,cereals,kp2,matches[:25],None,flag
s=2)
```

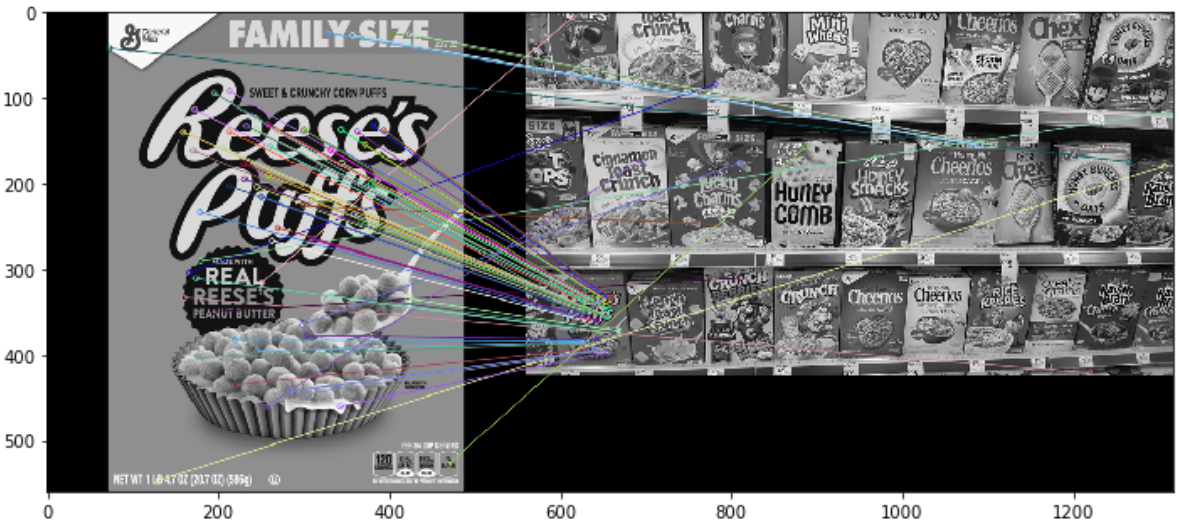
In [11]:
```
display(reeses_matches)
```



Brute-Force Matching with SIFT Descriptors and Ratio Test

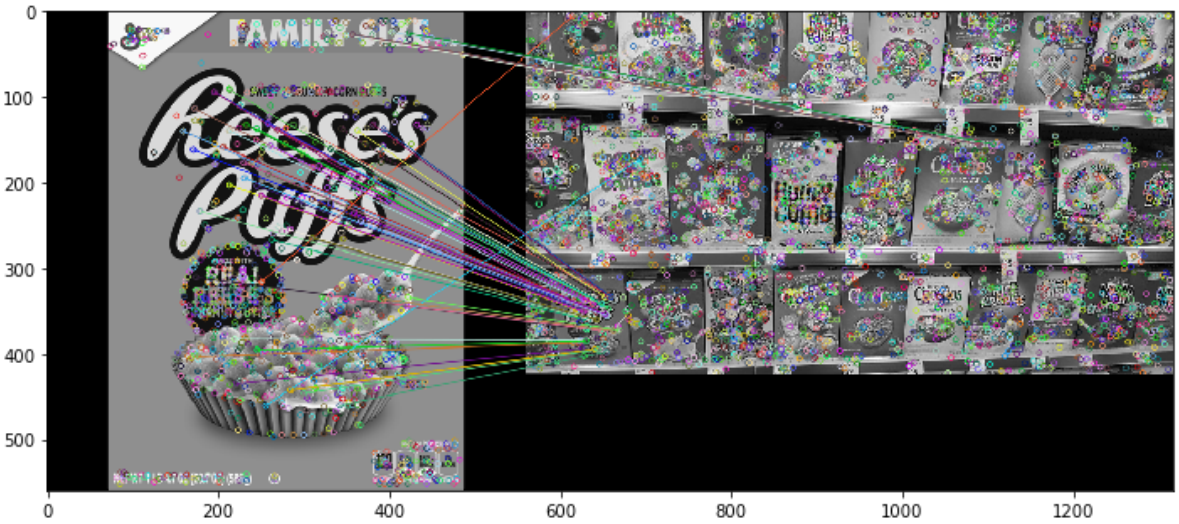
In [12]:
```
# Create SIFT Object
sift = cv2.xfeatures2d.SIFT_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(reeses,None)
kp2, des2 = sift.detectAndCompute(cereals,None)

# BFMatcher with default params
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

# Apply ratio test
good = []
for match1,match2 in matches:
    if match1.distance < 0.75*match2.distance:
        good.append([match1])

# cv2.drawMatchesKnn expects list of lists as matches.
sift_matches = cv2.drawMatchesKnn(reeses,kp1,cereals,kp2,good,None,flags=2)
```

In [13]:
```
display(sift_matches)
```



FLANN based Matcher

In [14]:
```
# Initiate SIFT detector
sift = cv2.xfeatures2d.SIFT_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(reeses,None)
kp2, des2 = sift.detectAndCompute(cereals,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

good = []

# ratio test
for i,(match1,match2) in enumerate(matches):
    if match1.distance < 0.7*match2.distance:

        good.append([match1])


flann_matches = cv2.drawMatchesKnn(reeses,kp1,cereals,kp2,good,None,flags=0)

display(flann_matches)
```



In [15]:
```
# Initiate SIFT detector
sift = cv2.xfeatures2d.SIFT_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(reeses,None)
kp2, des2 = sift.detectAndCompute(cereals,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in range(len(matches))]

# ratio test
for i,(match1,match2) in enumerate(matches):
    if match1.distance < 0.7*match2.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0),
                   matchesMask = matchesMask,
                   flags = 0)

flann_matches = cv2.drawMatchesKnn(reeses,kp1,cereals,kp2,matches,None,**draw_
params)
```

In [16]:
```
display(flann_matches)
```

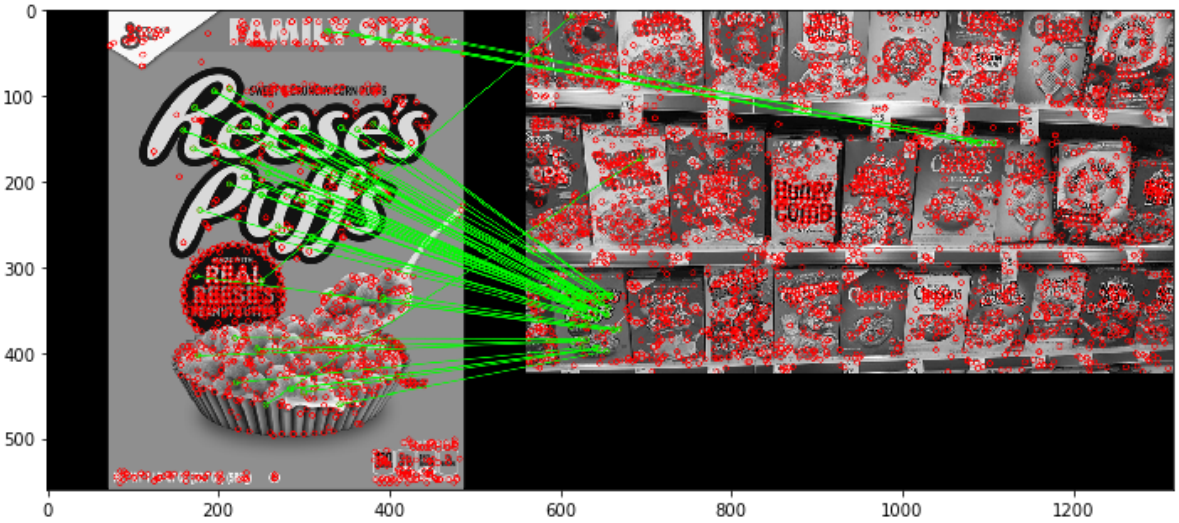


Image Segementation and the Watershed Algorithm

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

In [3]:
```
def display(img,cmap=None):
    fig = plt.figure(figsize=(10,8))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap=cmap)
```

Common Coin Example

In [4]:
```
sep_coins=cv2.imread("pennies.jpg")
```

In [5]:
```
display(sep_coins)
```

Median Blur

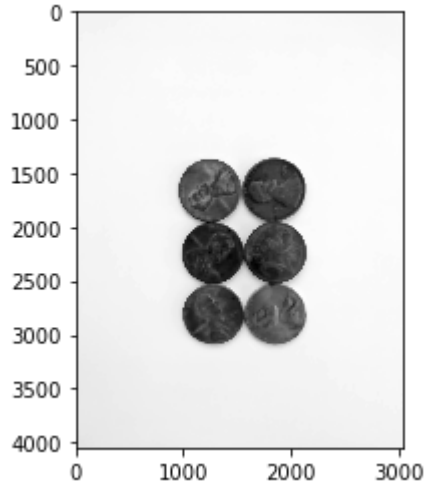
In [6]:
```
sep_blur = cv2.medianBlur(sep_coins,25)
```

In [7]:
```
display(sep_blur)
```

In [8]:
```
gray_sep_coins = cv2.cvtColor(sep_blur,cv2.COLOR_BGR2GRAY)
```

In [9]:
```
plt.imshow(gray_sep_coins,cmap="gray")
```

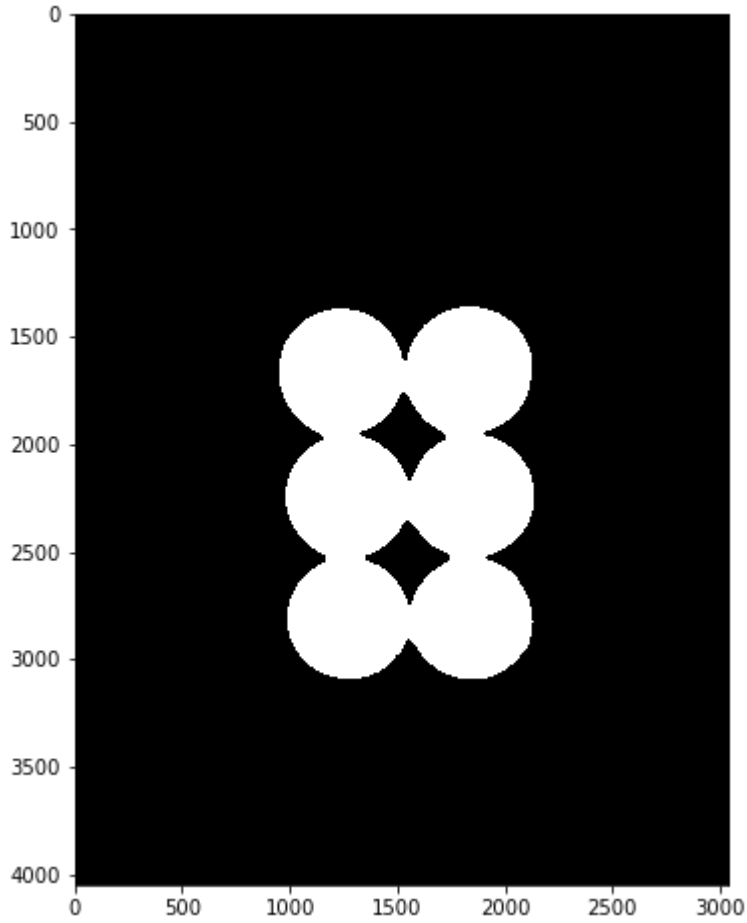
Out[9]:
```
<matplotlib.image.AxesImage at 0x27221578b00>
```



Binary Threshold

In [10]:
```
ret, sep_thresh = cv2.threshold(gray_sep_coins,160,255,cv2.THRESH_BINARY_INV)
```

In [12]:
```
display(sep_thresh,cmap="gray")
```



Find Counters

In [16]:
```
contours, hierarchy = cv2.findContours(sep_thresh.copy(), cv2.RETR_CCOMP, cv2.
CHAIN_APPROX_SIMPLE)
```

In [18]:
```
# For every entry in contours
for i in range(len(contours)):

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

        # We can now draw the external contours from the list of contours
        cv2.drawContours(sep_coins, contours, i, (255, 0, 0), 10)
```

In [19]:
```
display(sep_coins)
```

Watershed-Algorithm

Step 1: Read Image

In [28]:
```
img=cv2.imread("pennies.jpg")
```

Step 2: Apply Blur

In [29]:
```
img = cv2.medianBlur(img,35)
```

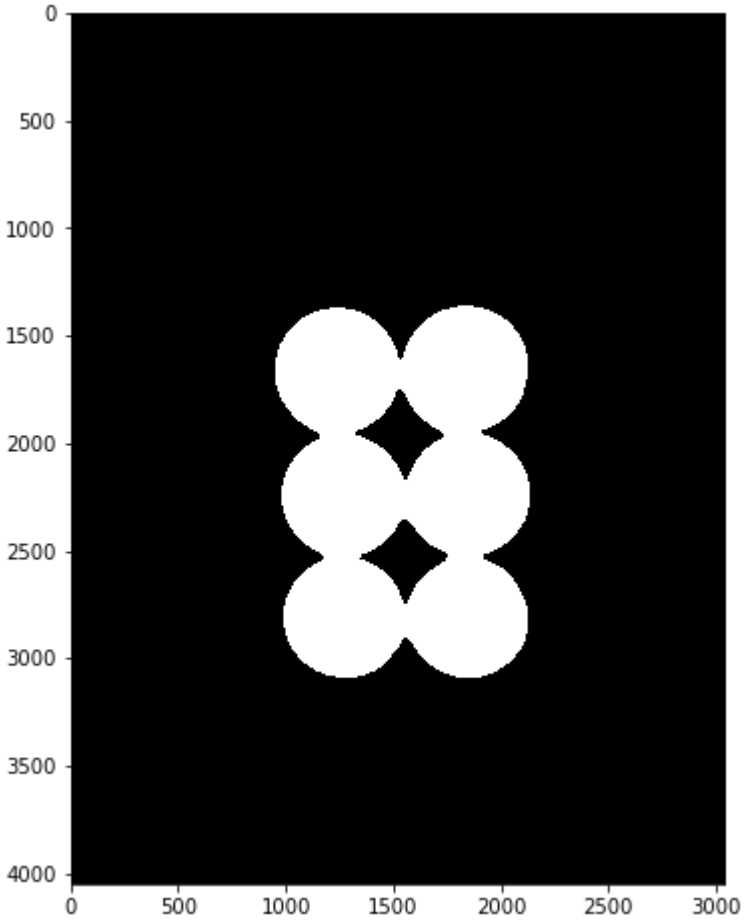
Step 3:Convert to grayscale

In [30]:
```
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

Step 4: Apply Threshold (Inverse Binary with OTSU as well)

In [32]:
```
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

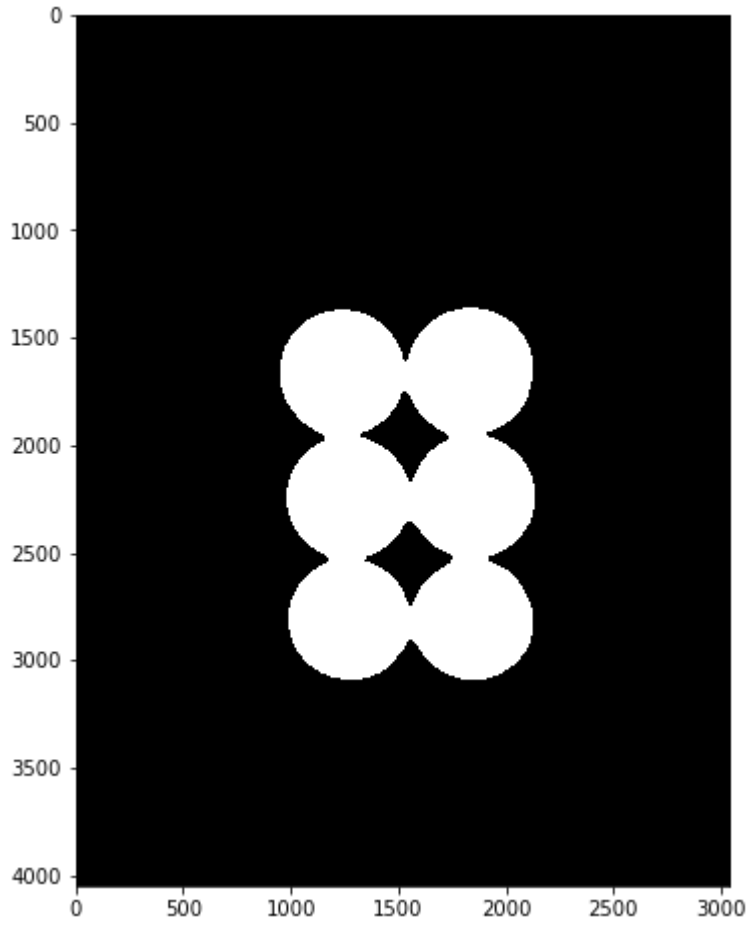
In [33]:
```
display(thresh,cmap="gray")
```



Optional Step 5: Noise Removal

In [34]:
```
# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
```

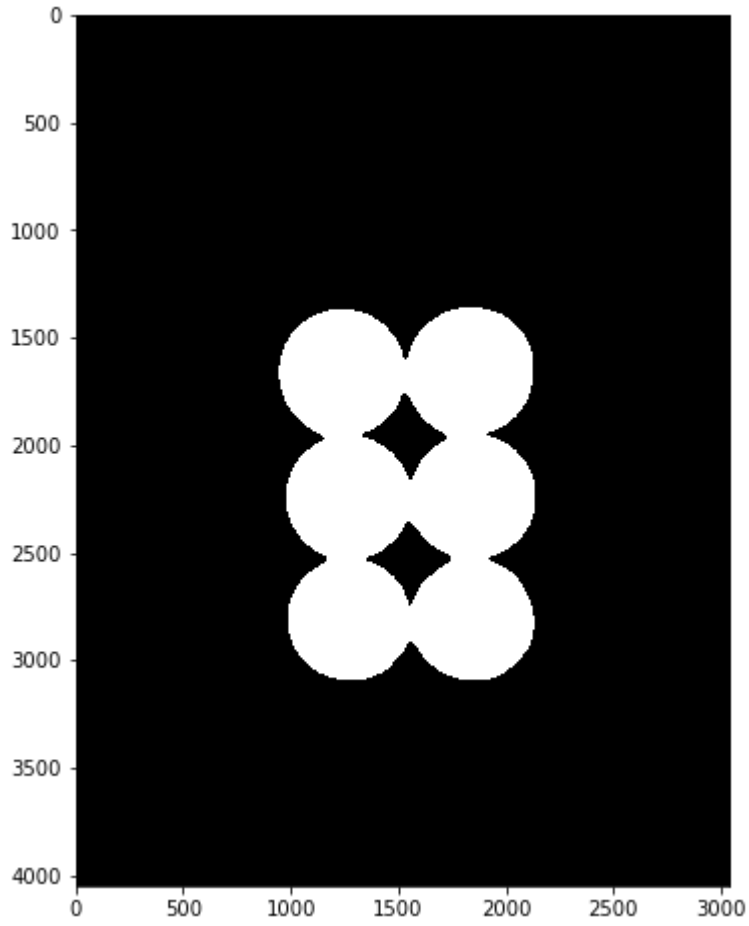
In [36]:
```
display(opening,cmap="gray")
```



Step 6: Grab Background that you are sure of

In [40]:
```
# sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=3)
```

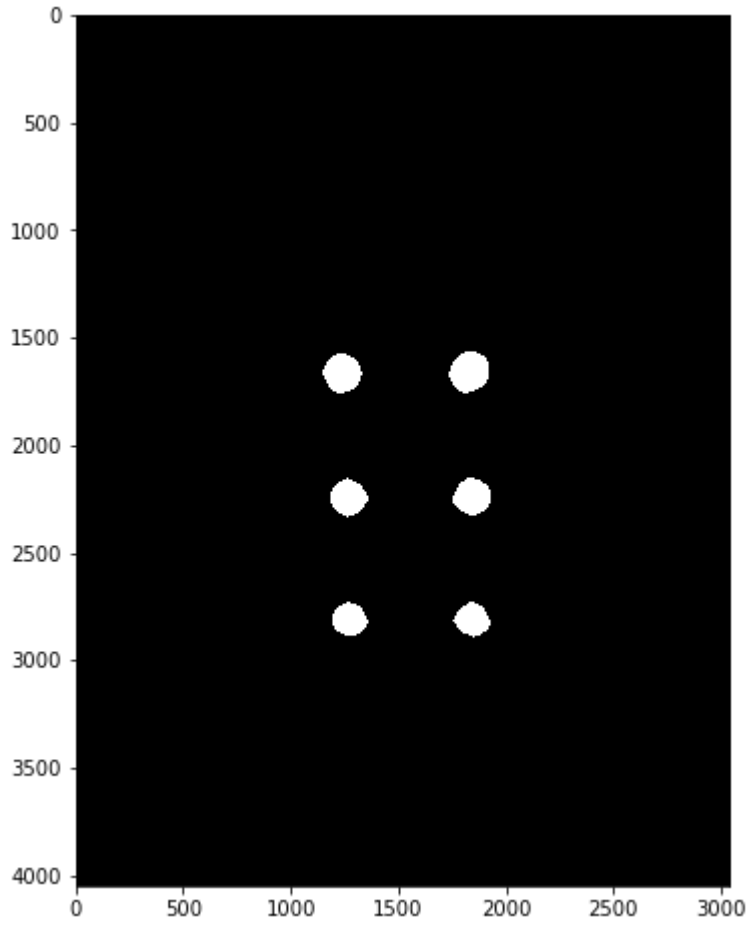
In [42]:
```
display(sure_bg,cmap="gray")
```



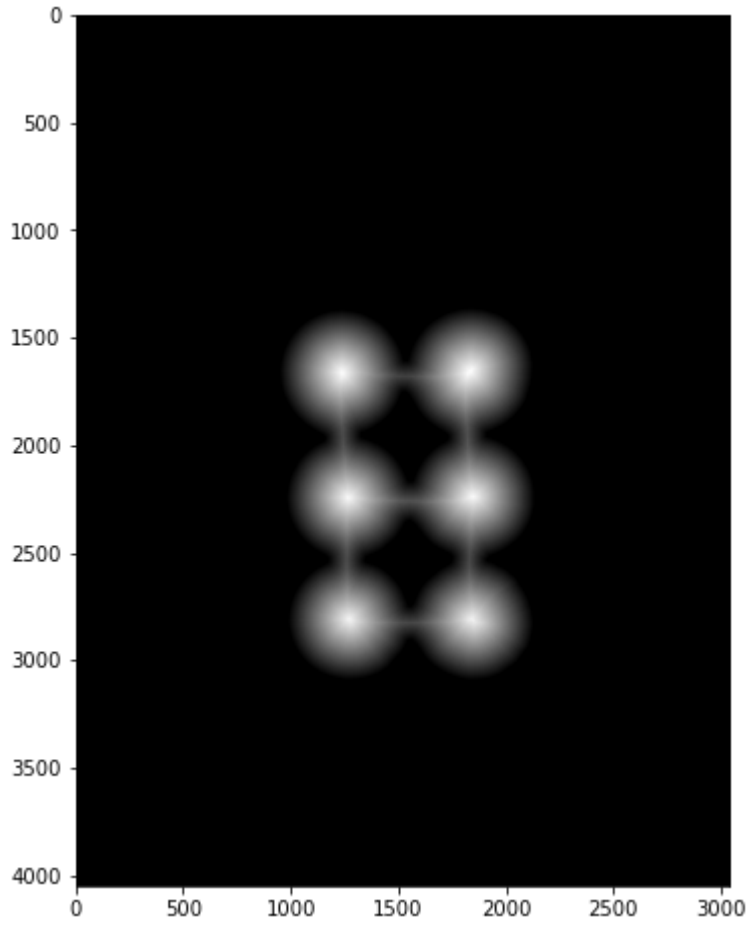
Step 7: Find Sure Foreground

In [43]:
```
# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
```

In [46]:
```
display(sure_fg,cmap="gray")
```



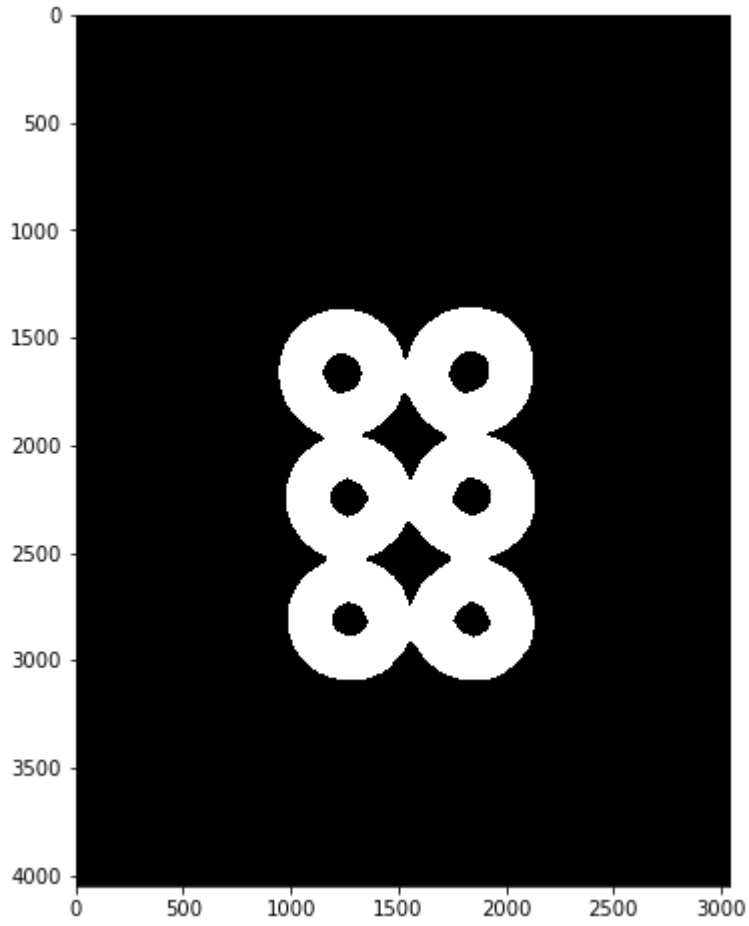
In [47]:
```
display(dist_transform,cmap='gray')
```



Step 8: Find Unknown Region

In [48]:
```
# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
```

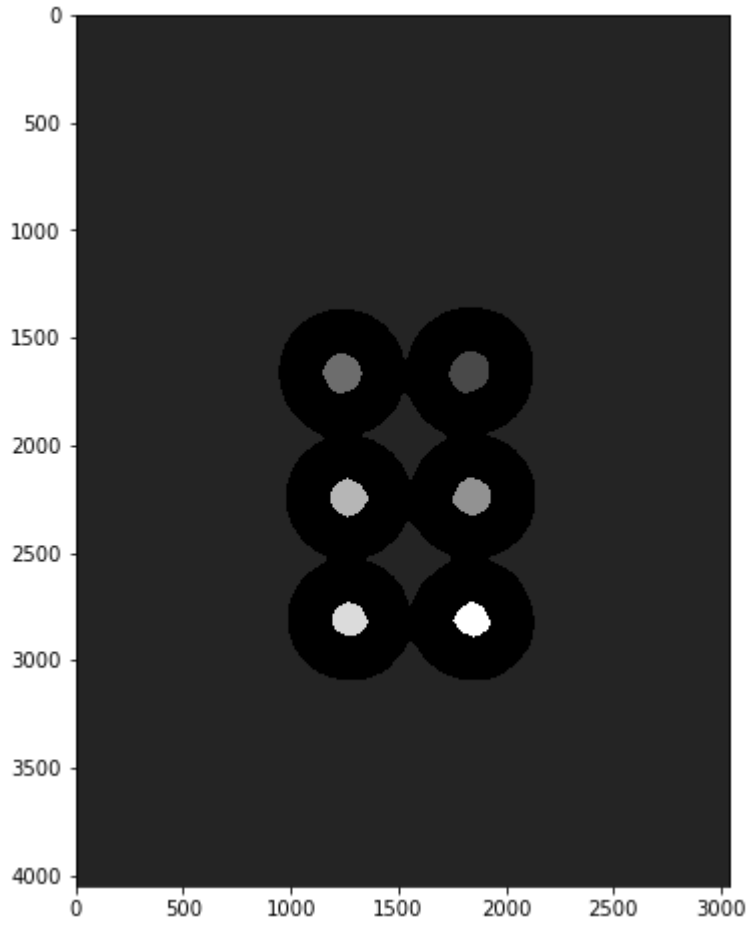
In [49]:
```
display(unknown,cmap="gray")
```



Step 9: Label Markers of Sure Foreground

In [50]:
```
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)
# Add one to all labels so that sure background is not 0, but 1
markers = markers+1
# Now, mark the region of unknown with zero
markers[unknown==255] = 0
```

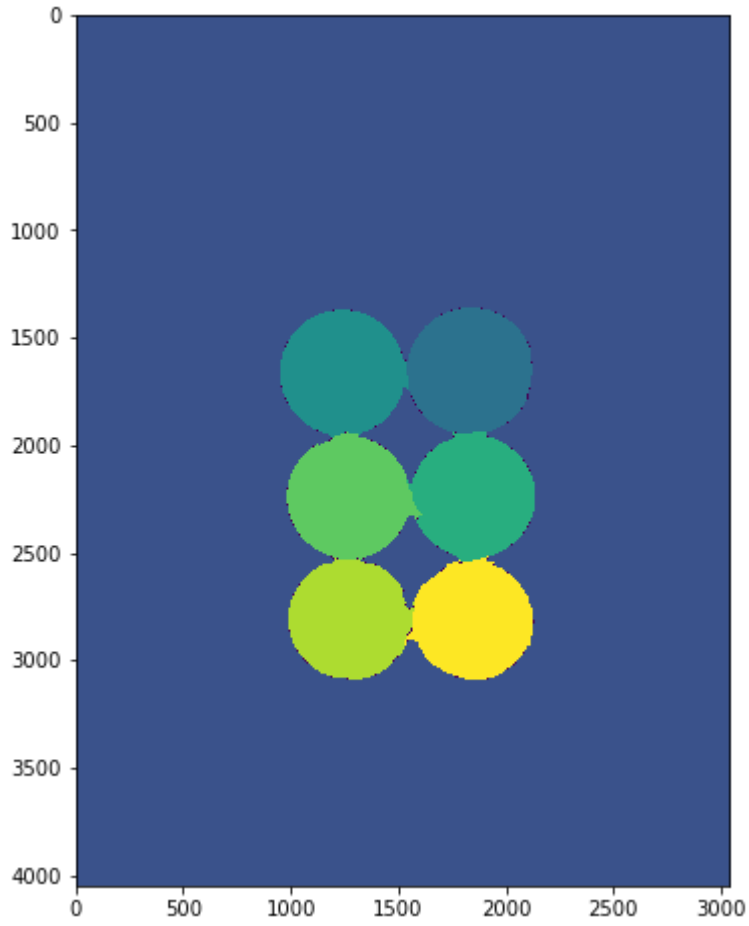
In [51]:
```
display(markers,cmap="gray")
```



Step 10: Apply Watershed Algorithm to find Markers

In [52]:
```
markers = cv2.watershed(img,markers)
```

In [53]:
```
display(markers)
```



Step 11: Find Contours on Markers

In [56]:
```
contours, hierarchy = cv2.findContours(markers.copy(), cv2.RETR_CCOMP, cv2.CHA
IN_APPROX_SIMPLE)

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

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

        # We can now draw the external contours from the list of contours
        cv2.drawContours(sep_coins, contours, i, (255, 0, 0), 10)
```

In [57]:
```
display(sep_coins)
```