Shree Ganeshya Namaha

Assignment 1 of Computer Vision

author : Anurag Gupta

Email : anuragsgupta9211@gmail.com

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
import PIL.Image as images
from skimage import io
import cv2 as cv
```

In [ ]:
```
import os

def getImage(url ) -> tuple:
    if os.path.exists(url):
        # image = io.imread(url)
        image_cv = cv.imread(url)
        image_np = np.array(image_cv)
        image_np = image_np.astype('int')
        print(f"Image Name {url}, Image Shape {image_np.shape}, dtype {image_np.dtype} ")

        gray_image = cv.cvtColor(image_cv, cv.COLOR_BGR2GRAY)

        return image_np,image_cv,gray_image
    return None

def getGrayImage(img):
    return cv.cvtColor(img,cv.COLOR_BGR2GRAY)
```

In [ ]:
```
url = "./images/rose1.jpg"
og_rose_img_np,grey_rose_img_cv,grey_rose_img = getImage(url)
```

```
Image Name ./images/rose1.jpg, Image Shape (481, 640, 3), dtype int32
```

In [ ]:
```
url = "./images/school.jpg"
school_img_np,school_img_cv,gray_school_img = getImage(url)



url = "./images/low.jpeg"
og_low_img_np,og_low_img_cv,gray_low_img = getImage(url)

url = "./images/lenna.png"
og_lenna_img_np ,og_lenna_img_cv,gray_lenna_img = getImage(url)

url = "./images/flowers.jpeg"
og_flower_img_np,og_flower_img_cv,gray_flower_img = getImage(url)
url = "./images/sunflower.jpeg"
og_sunfower_img_np,og_sunfower_img_cv,gray_sunfower_img = getImage(url)
```

```
Image Name ./images/school.jpg, Image Shape (552, 800, 3), dtype int32
Image Name ./images/low.jpeg, Image Shape (256, 256, 3), dtype int32
Image Name ./images/lenna.png, Image Shape (512, 512, 3), dtype int32
Image Name ./images/flowers.jpeg, Image Shape (1800, 2880, 3), dtype int32
Image Name ./images/sunflower.jpeg, Image Shape (296, 474, 3), dtype int32
```

3. Check image type

```
#checking type of image
print("The loaded image is of type:",type(image))
print("The loaded image is of size:",image.shape)
print("intensity at index", image[50][50])
```

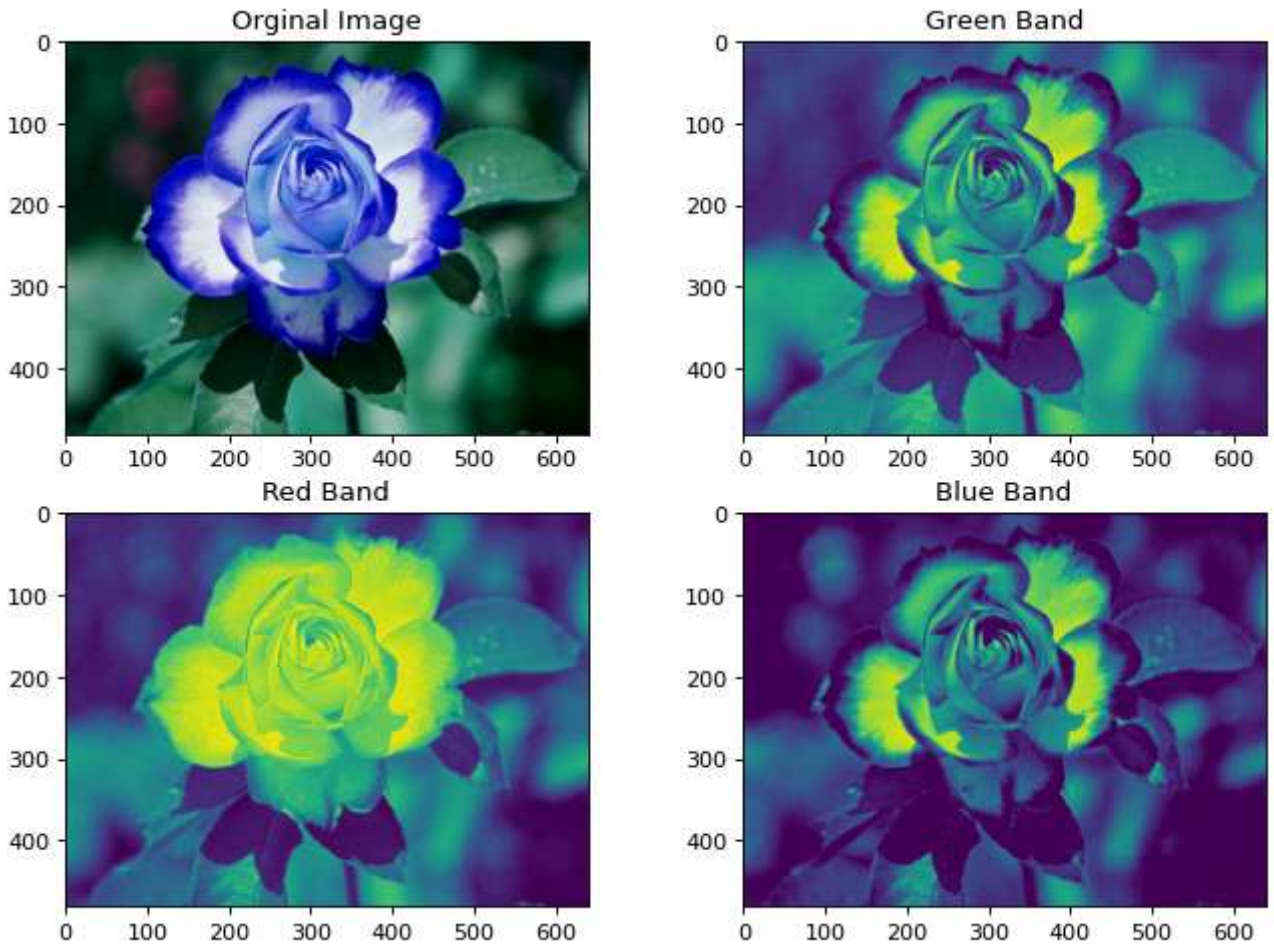
In [ ]:
```
print("Type of loaded Image", type(school_img_np))
print("Shape of loaded Image", school_img_np.shape)
print("Intensity at index ", school_img_np[50][50])
```

```
Type of loaded Image <class 'numpy.ndarray'>
Shape of loaded Image (552, 800, 3)
Intensity at index  [219 170 114]
```

Color Space

In [ ]:
```
og_image = og_rose_img_np
B,G,R = cv.split(og_image)

# fig = plt.figure(10,7)
fig, plots = plt.subplots(2, 2, figsize=(10, 7))

plots[0][0].set_title("Orginal Image")
plots[0][0].imshow(og_image)

plots[0][1].set_title("Green Band")
plots[0][1].imshow(G)

plots[1][0].set_title("Red Band")
plots[1][0].imshow(R)
plots[1][1].set_title("Blue Band")
plots[1][1].imshow(B)
```

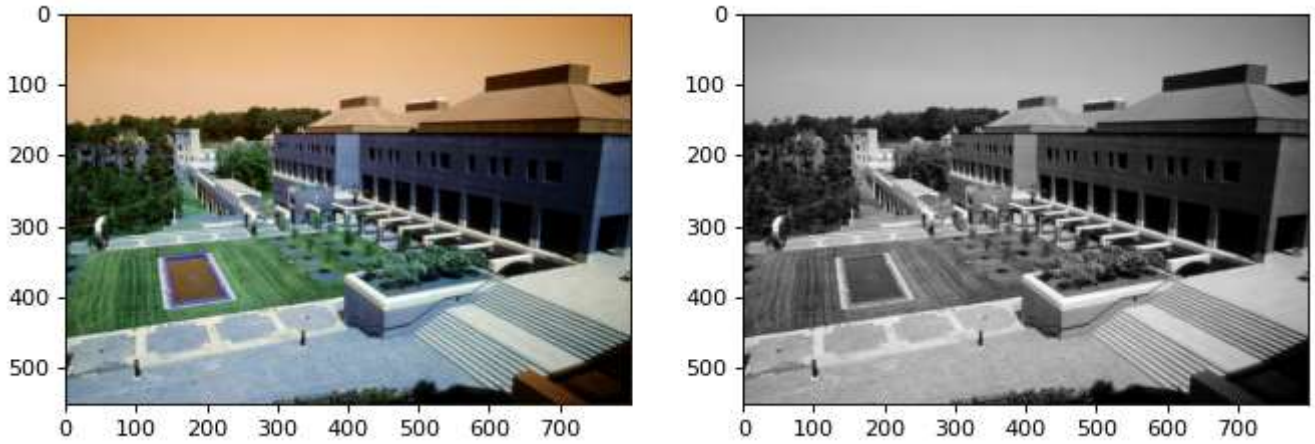
Out[ ]:
```
<matplotlib.image.AxesImage at 0x24da7af5650>
```



Converting Grayscale

In [ ]:
```
sc_img= cv.imread('./images/school.jpg')
gray_image = cv.cvtColor(sc_img,cv.COLOR_BGR2GRAY)
fig = plt.figure(figsize=(10,7))
fig.add_subplot(1,2,1)
plt.imshow(school_img_cv,cmap=plt.get_cmap('gray'))
fig.add_subplot(1,2,2)
plt.imshow(gray_image,cmap=plt.get_cmap('gray'))
```

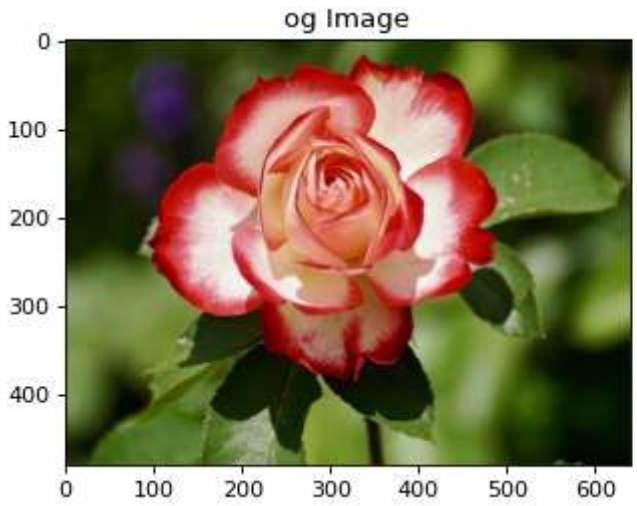
Out[ ]:
```
<matplotlib.image.AxesImage at 0x24d9a0b9d50>
```

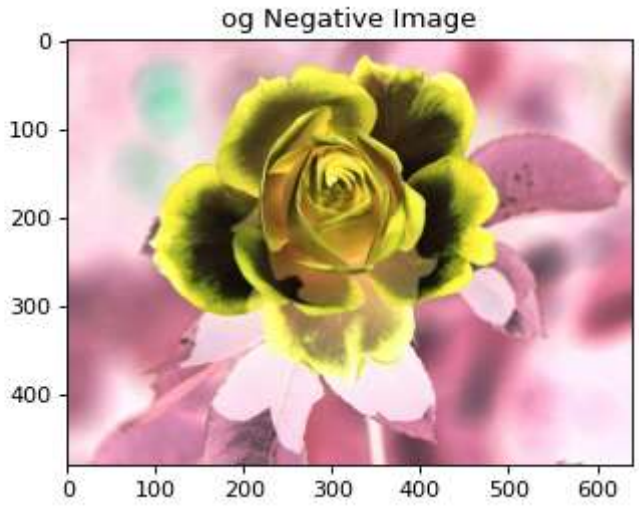


Creating a Negative Image

In [ ]:
```
# gray_image = cv.cvtColor(rose_img,cv.COLOR_BGR2GRAY)
rose_img = io.imread('./images/rose1.jpg')
rose_neg_img = np.abs(og_rose_img_np-255)
fig = plt.figure(figsize=(10,7))
fig.add_subplot(1,2,1)
plt.title("og Image")
plt.imshow(rose_img,cmap=plt.get_cmap('gray'))
fig.add_subplot(1,2,2)
plt.title("og Negative Image")
plt.imshow(rose_neg_img,cmap=plt.get_cmap('gray'))
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24d9a0d3ad0>
```





Histogram Equalisation

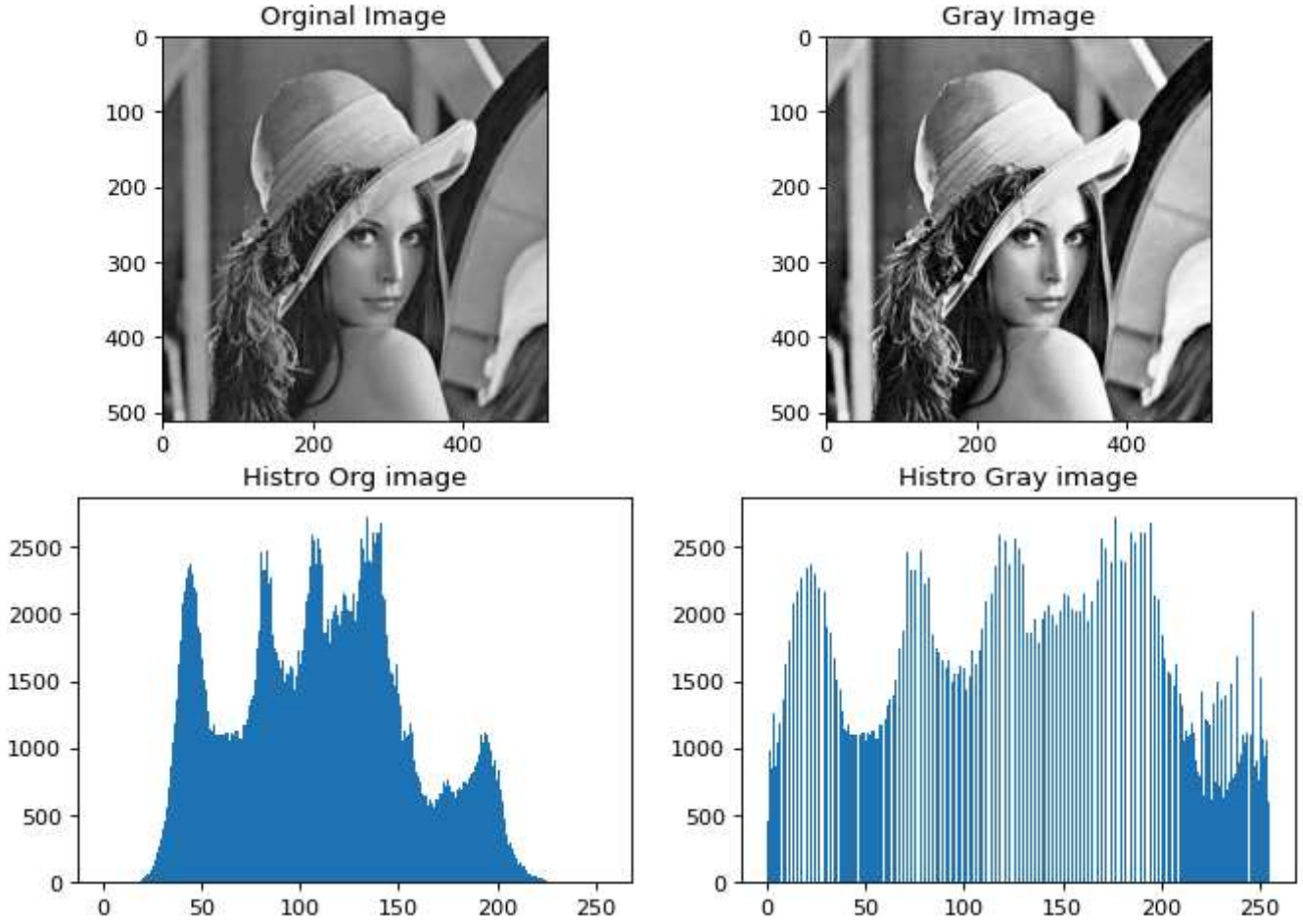
In [ ]:
```
image_path = './images/lenna.png'
gaussian_image = io.imread(image_path)

# Convert the image to grayscale using OpenCV
gray_lena_image = cv.cvtColor(gaussian_image, cv.COLOR_BGR2GRAY)

# Ensure the grayscale image is of data type np.uint8
gray_lena_image = gray_lena_image.astype(np.uint8)

hist_equalis_img = cv.equalizeHist(gray_lena_image)

fig = plt.figure(figsize=(10,7))

fig.add_subplot(2,2,1)
plt.imshow(gray_lena_image,cmap=plt.get_cmap('gray'))
plt.title("Orginal Image")

fig.add_subplot(2,2,2)
plt.title("Gray Image")
plt.imshow(hist_equalis_img,cmap='gray')

fig.add_subplot(2,2,3)
```

```
plt.title("Histro Org image")
plt.hist(gray_lena_image.ravel(),bins=256,range=[0,255])


fig.add_subplot(2,2,4)
plt.title("Histro Gray image")
plt.hist(hist_equalis_img.ravel(),bins=256,range=[0,255])

# plt.imshow(np.hstack([gray_image,hist_equalis]))
```

Out[ ]:
```
(array([ 457.,  978.,  852., 1263.,  872., 1038., 1186.,    0., 1367.,
        1629.,    0., 1794.,    0., 2079.,    0., 2170.,    0., 2264.,
           0.,    0., 2334.,    0., 2364.,    0., 2301.,    0., 2197.,
           0.,    0., 2162., 1902.,    0., 1863.,    0., 1668., 1509.,
           0., 1438., 1267., 1136., 1122., 1168., 1098., 1095., 1094.,
        1093.,    0., 1097., 1117., 1109., 1061., 1114., 1092., 1126.,
        1125., 1075., 1063., 1174., 1177.,    0., 1214., 1319., 1363.,
           0., 1398., 1505.,    0., 1737.,    0., 1868.,    0., 2451.,
           0., 2320.,    0., 2327.,    0.,    0., 2466.,    0., 2229.,
           0., 2272.,    0., 1843.,    0., 1736., 1707.,    0., 1655.,
           0., 1590., 1647.,    0., 1488., 1555.,    0., 1559., 1615.,
           0., 1601., 1432.,    0., 1539., 1728.,    0., 1630.,    0.,
        1724., 1889.,    0., 2094.,    0.,    0., 2144.,    0., 2360.,
           0., 2586.,    0.,    0., 2539.,    0., 2369.,    0.,    0.,
        2561.,    0., 2488.,    0., 2368.,    0., 1854.,    0., 1855.,
           0., 1965.,    0., 1783.,    0., 1966., 2014.,    0., 2061.,
           0., 1986.,    0., 1922.,    0., 2026.,    0., 2149.,    0.,
        2142.,    0., 2031.,    0., 2018.,    0., 2025.,    0., 2150.,
           0., 1950.,    0., 2088.,    0.,    0., 2254.,    0., 2567.,
           0., 2490.,    0.,    0., 2378.,    0., 2727.,    0.,    0.,
        2393.,    0., 2379.,    0.,    0., 2600.,    0., 2527.,    0.,
           0., 2598.,    0., 2600.,    0.,    0., 2679.,    0., 2135.,
           0., 2100.,    0., 1840., 1666.,    0., 1571., 1559.,    0.,
        1461., 1629.,    0., 1400., 1322., 1052., 1134., 1083., 1096.,
        1183., 1119.,  904.,  823.,  786., 1419.,  651., 1209., 1203.,
        1179.,  619., 1331.,  742., 1495.,  716., 1360.,  638., 1389.,
         695.,  752., 1483.,  782.,  802., 1680.,  898.,  951., 1097.,
        1046., 1118.,    0., 1102., 2023.,  870.,  903.,  756., 1523.,
        1074.,  943., 1050.,  599.]),
 array([  0.        ,   0.99609375,   1.9921875 ,   2.98828125,
          3.984375  ,   4.98046875,   5.9765625 ,   6.97265625,
          7.96875   ,   8.96484375,   9.9609375 ,  10.95703125,
         11.953125  ,  12.94921875,  13.9453125 ,  14.94140625,
         15.9375    ,  16.93359375,  17.9296875 ,  18.92578125,
         19.921875  ,  20.91796875,  21.9140625 ,  22.91015625,
         23.90625   ,  24.90234375,  25.8984375 ,  26.89453125,
         27.890625  ,  28.88671875,  29.8828125 ,  30.87890625,
         31.875     ,  32.87109375,  33.8671875 ,  34.86328125,
         35.859375  ,  36.85546875,  37.8515625 ,  38.84765625,
         39.84375   ,  40.83984375,  41.8359375 ,  42.83203125,
         43.828125  ,  44.82421875,  45.8203125 ,  46.81640625,
         47.8125    ,  48.80859375,  49.8046875 ,  50.80078125,
```

```
 51.796875  ,  52.79296875,  53.7890625 ,  54.78515625,
 55.78125   ,  56.77734375,  57.7734375 ,  58.76953125,
 59.765625  ,  60.76171875,  61.7578125 ,  62.75390625,
 63.75      ,  64.74609375,  65.7421875 ,  66.73828125,
 67.734375  ,  68.73046875,  69.7265625 ,  70.72265625,
 71.71875   ,  72.71484375,  73.7109375 ,  74.70703125,
 75.703125  ,  76.69921875,  77.6953125 ,  78.69140625,
 79.6875    ,  80.68359375,  81.6796875 ,  82.67578125,
 83.671875  ,  84.66796875,  85.6640625 ,  86.66015625,
 87.65625   ,  88.65234375,  89.6484375 ,  90.64453125,
 91.640625  ,  92.63671875,  93.6328125 ,  94.62890625,
 95.625     ,  96.62109375,  97.6171875 ,  98.61328125,
 99.609375  , 100.60546875, 101.6015625 , 102.59765625,
103.59375   , 104.58984375, 105.5859375 , 106.58203125,
107.578125  , 108.57421875, 109.5703125 , 110.56640625,
111.5625    , 112.55859375, 113.5546875 , 114.55078125,
115.546875  , 116.54296875, 117.5390625 , 118.53515625,
119.53125   , 120.52734375, 121.5234375 , 122.51953125,
123.515625  , 124.51171875, 125.5078125 , 126.50390625,
127.5       , 128.49609375, 129.4921875 , 130.48828125,
131.484375  , 132.48046875, 133.4765625 , 134.47265625,
135.46875   , 136.46484375, 137.4609375 , 138.45703125,
139.453125  , 140.44921875, 141.4453125 , 142.44140625,
143.4375    , 144.43359375, 145.4296875 , 146.42578125,
147.421875  , 148.41796875, 149.4140625 , 150.41015625,
151.40625   , 152.40234375, 153.3984375 , 154.39453125,
155.390625  , 156.38671875, 157.3828125 , 158.37890625,
159.375     , 160.37109375, 161.3671875 , 162.36328125,
163.359375  , 164.35546875, 165.3515625 , 166.34765625,
167.34375   , 168.33984375, 169.3359375 , 170.33203125,
171.328125  , 172.32421875, 173.3203125 , 174.31640625,
175.3125    , 176.30859375, 177.3046875 , 178.30078125,
179.296875  , 180.29296875, 181.2890625 , 182.28515625,
183.28125   , 184.27734375, 185.2734375 , 186.26953125,
187.265625  , 188.26171875, 189.2578125 , 190.25390625,
191.25      , 192.24609375, 193.2421875 , 194.23828125,
195.234375  , 196.23046875, 197.2265625 , 198.22265625,
199.21875   , 200.21484375, 201.2109375 , 202.20703125,
203.203125  , 204.19921875, 205.1953125 , 206.19140625,
207.1875    , 208.18359375, 209.1796875 , 210.17578125,
211.171875  , 212.16796875, 213.1640625 , 214.16015625,
215.15625   , 216.15234375, 217.1484375 , 218.14453125,
```

```
        219.140625  , 220.13671875, 221.1328125 , 222.12890625,
        223.125     , 224.12109375, 225.1171875 , 226.11328125,
        227.109375  , 228.10546875, 229.1015625 , 230.09765625,
        231.09375   , 232.08984375, 233.0859375 , 234.08203125,
        235.078125  , 236.07421875, 237.0703125 , 238.06640625,
        239.0625    , 240.05859375, 241.0546875 , 242.05078125,
        243.046875  , 244.04296875, 245.0390625 , 246.03515625,
        247.03125   , 248.02734375, 249.0234375 , 250.01953125,
        251.015625  , 252.01171875, 253.0078125 , 254.00390625,
        255.        ]),
<BarContainer object of 256 artists>)
```



Thresholding

Practice question

1. Use differnt threshold and try thresholding
2. Try various thresholding techniques like OTSU, Kapur
3. Try Multithresholding(one or more thresholds)

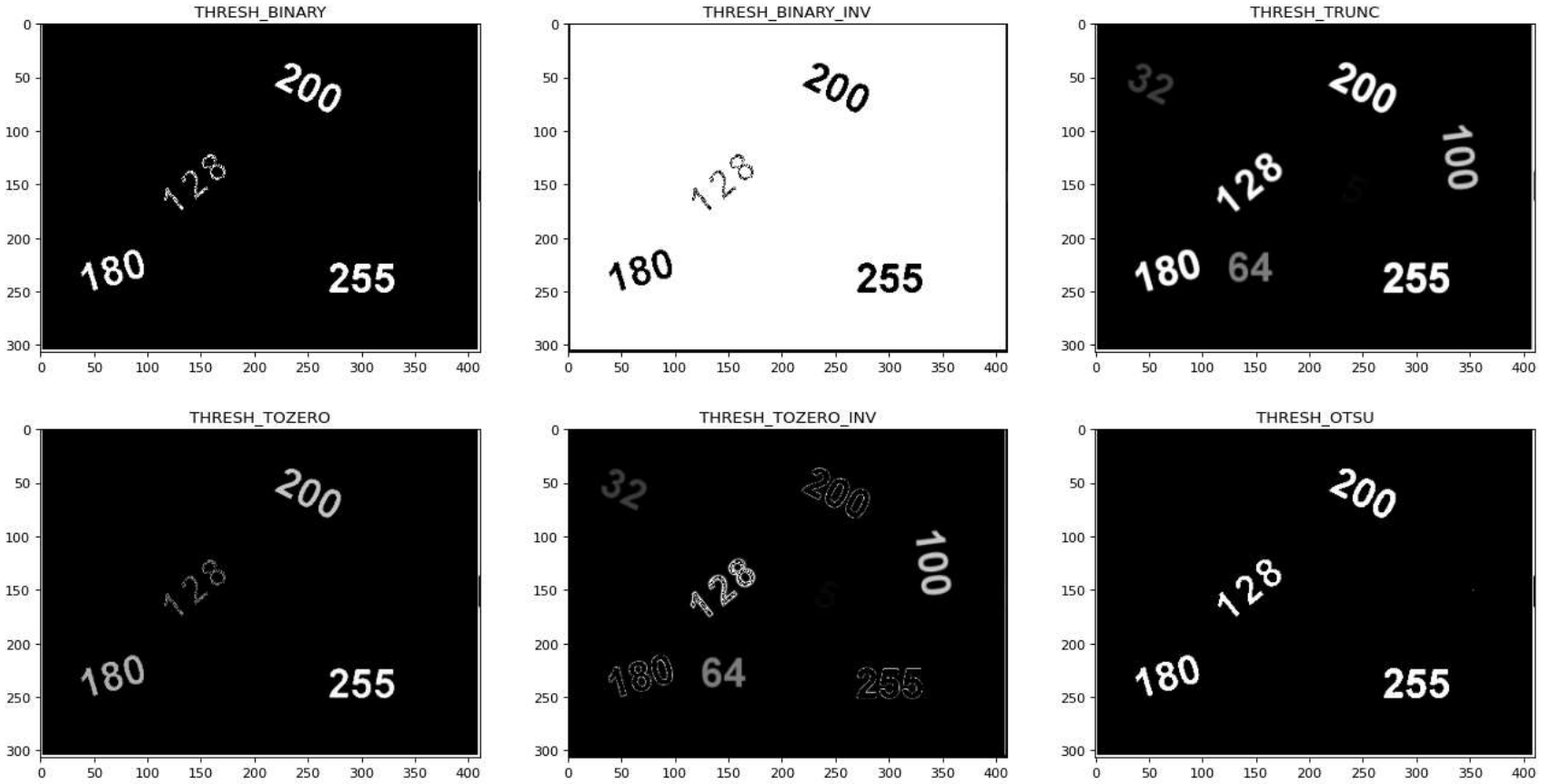
1. Use differnt threshold and try thresholding

In [ ]:
```
image_path = './images/ostuExample.png'
gaussian_image = cv.imread(image_path)

# Convert the image to grayscale using OpenCV
gray_image = cv.cvtColor(gaussian_image, cv.COLOR_BGR2GRAY)

# Ensure the grayscale image is of data type np.uint8
# gray_image = gray_image.astype(np.uint8)
ret, thresh1 = cv.threshold(gray_image, 127,255,cv.THRESH_BINARY)
ret, thresh2 = cv.threshold(gray_image, 127,255,cv.THRESH_BINARY_INV)
ret, thresh3 = cv.threshold(gray_image, 127,255,cv.THRESH_TRUNC)
ret, thresh4 = cv.threshold(gray_image, 127,255,cv.THRESH_TOZERO)
ret, thresh5 = cv.threshold(gray_image, 127,255,cv.THRESH_TOZERO_INV)
ret, threshOstu = cv.threshold(gray_image, 127,255,cv.THRESH_OTSU)
# ret, thresh5 = cv.threshold(gray_image, 127,255,cv.)
```

In [ ]:
```
# plt.imshow(np.hstack([gray_image,thresh1]))
fig = plt.figure(figsize=(20,10))
col, row = 2,3
fig.add_subplot(col,row,1)

plt.imshow(thresh1,cmap='gray')
plt.title("THRESH_BINARY")

fig.add_subplot(col,row,2)
```

```
plt.title("THRESH_BINARY_INV")
plt.imshow(thresh2,cmap='gray')

fig.add_subplot(col,row,3)
plt.title("THRESH_TRUNC")
plt.imshow(thresh3,cmap='gray')
fig.add_subplot(col,row,4)
plt.title("THRESH_TOZERO")
plt.imshow(thresh4,cmap='gray')
fig.add_subplot(col,row,5)
plt.title("THRESH_TOZERO_INV")
plt.imshow(thresh5,cmap='gray')
fig.add_subplot(col,row,6)
plt.title("THRESH_OTSU")
plt.imshow(threshOstu,cmap='gray')
```

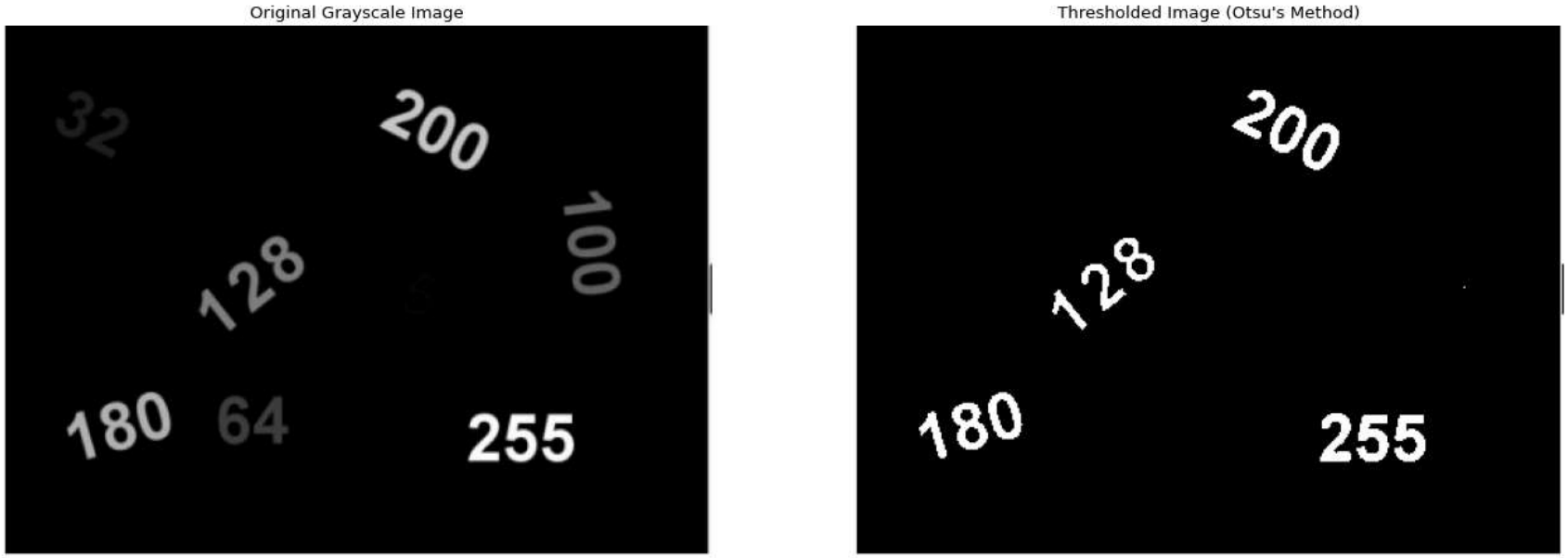
Out[ ]: <matplotlib.image.AxesImage at 0x24d9c393e50>



Apply Otsu's thresholding

In [ ]:
```
image_path = './images/ostuExample.png'
ostu_image = cv.imread(image_path)

# Convert the image to grayscale using OpenCV
ostu_image = cv.cvtColor(ostu_image, cv.COLOR_BGR2GRAY)

rest, thresholded_image = cv.threshold(ostu_image, 127, 255,cv.THRESH_OTSU)

# Create a figure with the specified size
fig, ax = plt.subplots(1, 2, figsize=(20, 10))

# Display the original grayscale image
```

```
ax[0].imshow(ostu_image, cmap='gray')
ax[0].set_title("Original Grayscale Image")
ax[0].axis('off')  # Hide axis

# Display the thresholded image using Otsu's method
ax[1].imshow(thresholded_image, cmap='gray')
ax[1].set_title("Thresholded Image (Otsu's Method)")
ax[1].axis('off')  # Hide axis

# Show the plot
plt.show()
```

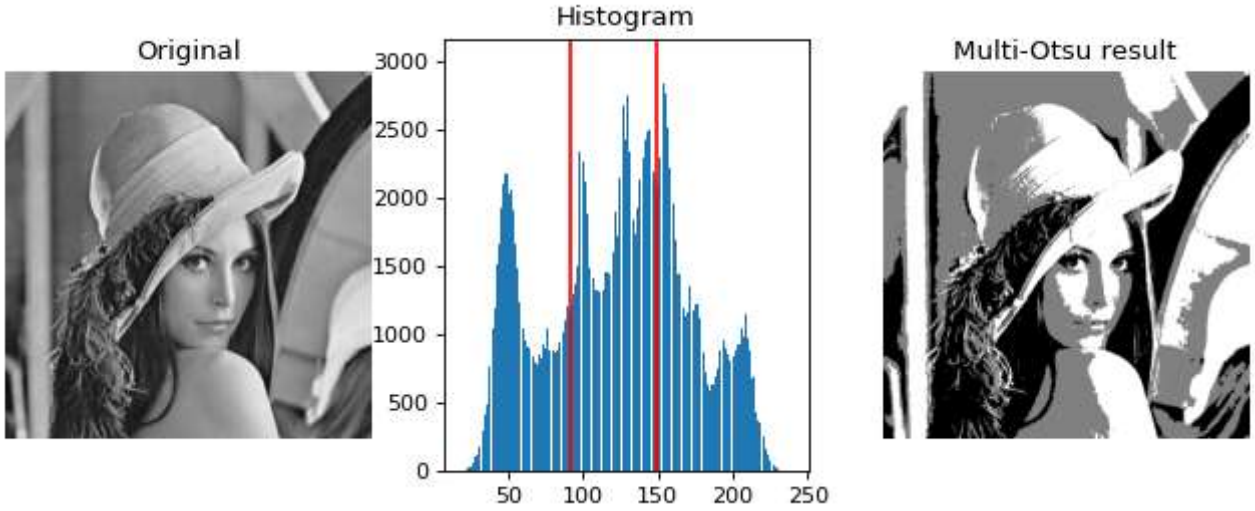


Try MultiThresholding (one or more thresholds)

The multi-Otsu threshold is a thresholding algorithm that is used to separate the pixels of an input image into several different classes, each one obtained according to the intensity of the gray levels within the image

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

```
from skimage import data
from skimage.filters import threshold_multiotsu

# Setting the font size for all plots.
# matplotlib.rcParams['font.size'] = 9

# The input image.
image = cv.imread('./images/lenna.png',0)
# Applying multi-Otsu threshold for the default value, generating
# three classes.
thresholds = threshold_multiotsu(image)

# Using the threshold values, we generate the three regions.
regions = np.digitize(image, bins=thresholds)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(10, 3.5))

# Plotting the original image.
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original')
ax[0].axis('off')

# Plotting the histogram and the two thresholds obtained from
# multi-Otsu.
ax[1].hist(image.ravel(), bins=255)
ax[1].set_title('Histogram')
for thresh in thresholds:
    ax[1].axvline(thresh, color='r')

# Plotting the Multi Otsu result.
ax[2].imshow(regions, cmap='gray')
ax[2].set_title('Multi-Otsu result')
ax[2].axis('off')

plt.subplots_adjust()

plt.show()
```



Task 3: Explore different color spaces i.e. yCbCr and Lab?

8. Adaptive Thresholding

```
#Adaptive threshold
adaptive_thresh1 = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 11, 2)
adaptive_thresh2 = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 11, 2)

#displaying the threshold images
cv2_imshow(np.hstack([adaptive_thresh1,adaptive_thresh2]))
```

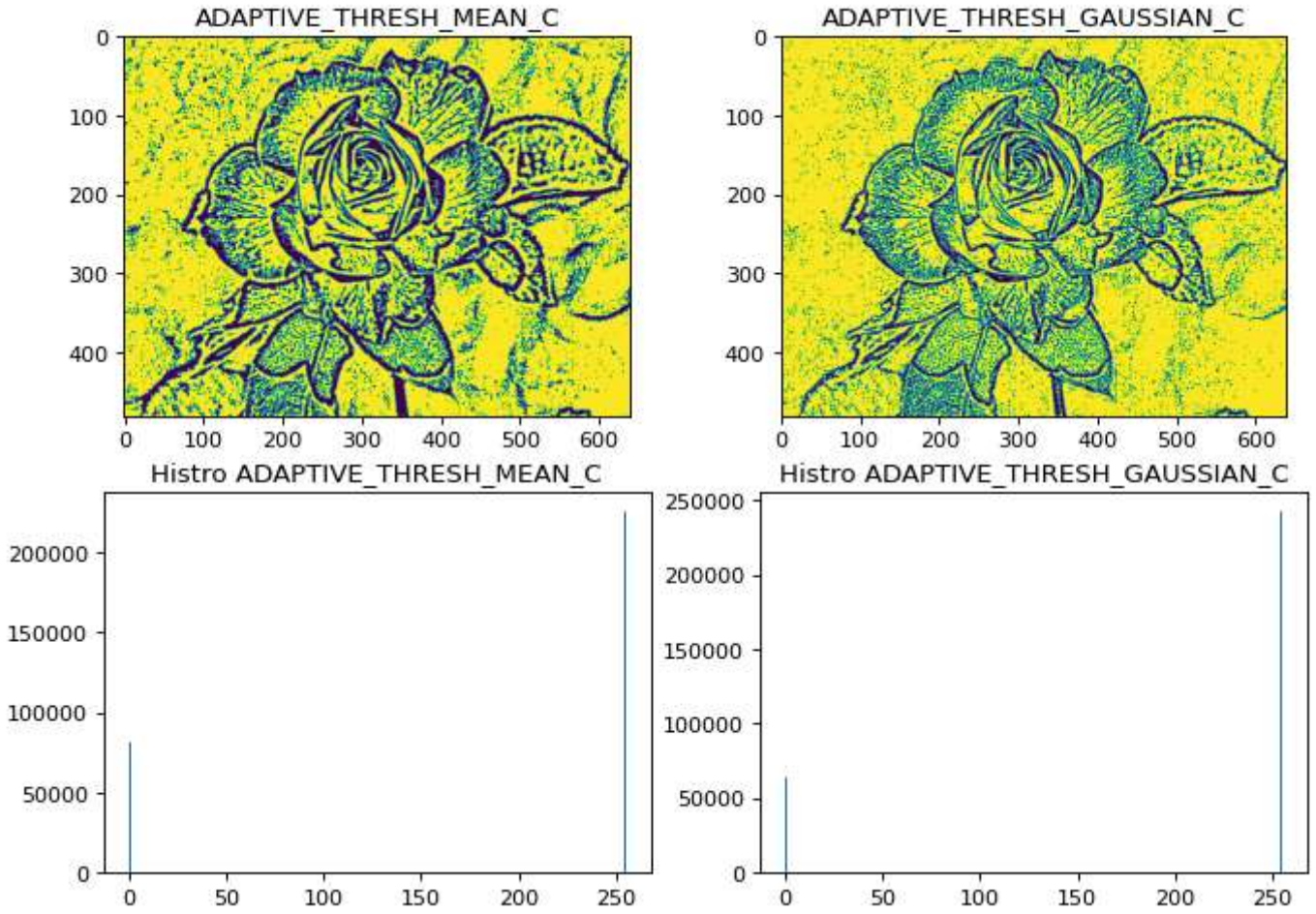
In [ ]:
```
adaptive_thresh1 = cv.adaptiveThreshold(grey_rose_img,255,cv.ADAPTIVE_THRESH_MEAN_C,
                                        cv.THRESH_BINARY,11,2)
adaptive_thresh2 = cv.adaptiveThreshold(grey_rose_img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,
                                        cv.THRESH_BINARY,11,2)
```

```
fig =plt.figure(figsize=(10,7))
fig.add_subplot(2,2,1)
plt.title("ADAPTIVE_THRESH_MEAN_C")
plt.imshow(adaptive_thresh1)

fig.add_subplot(2,2,2)
plt.title("ADAPTIVE_THRESH_GAUSSIAN_C")
plt.imshow(adaptive_thresh2)


fig.add_subplot(2,2,3)
plt.title("Histro ADAPTIVE_THRESH_MEAN_C")
plt.hist(adaptive_thresh1.ravel(), bins=256)

fig.add_subplot(2,2,4)
plt.title("Histro ADAPTIVE_THRESH_GAUSSIAN_C")
plt.hist(adaptive_thresh2.ravel(), bins=256)
```

```
Out[ ]: (array([ 64700.,      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.,      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.,
                     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., 243140.]),
         array([  0.        ,   0.99609375,   1.9921875 ,   2.98828125,
                  3.984375  ,   4.98046875,   5.9765625 ,   6.97265625,
                  7.96875   ,   8.96484375,   9.9609375 ,  10.95703125,
                 11.953125  ,  12.94921875,  13.9453125 ,  14.94140625,
                 15.9375    ,  16.93359375,  17.9296875 ,  18.92578125,
```

```
 19.921875  ,  20.91796875,  21.9140625 ,  22.91015625,
 23.90625   ,  24.90234375,  25.8984375 ,  26.89453125,
 27.890625  ,  28.88671875,  29.8828125 ,  30.87890625,
 31.875     ,  32.87109375,  33.8671875 ,  34.86328125,
 35.859375  ,  36.85546875,  37.8515625 ,  38.84765625,
 39.84375   ,  40.83984375,  41.8359375 ,  42.83203125,
 43.828125  ,  44.82421875,  45.8203125 ,  46.81640625,
 47.8125    ,  48.80859375,  49.8046875 ,  50.80078125,
 51.796875  ,  52.79296875,  53.7890625 ,  54.78515625,
 55.78125   ,  56.77734375,  57.7734375 ,  58.76953125,
 59.765625  ,  60.76171875,  61.7578125 ,  62.75390625,
 63.75      ,  64.74609375,  65.7421875 ,  66.73828125,
 67.734375  ,  68.73046875,  69.7265625 ,  70.72265625,
 71.71875   ,  72.71484375,  73.7109375 ,  74.70703125,
 75.703125  ,  76.69921875,  77.6953125 ,  78.69140625,
 79.6875    ,  80.68359375,  81.6796875 ,  82.67578125,
 83.671875  ,  84.66796875,  85.6640625 ,  86.66015625,
 87.65625   ,  88.65234375,  89.6484375 ,  90.64453125,
 91.640625  ,  92.63671875,  93.6328125 ,  94.62890625,
 95.625     ,  96.62109375,  97.6171875 ,  98.61328125,
 99.609375  , 100.60546875, 101.6015625 , 102.59765625,
103.59375   , 104.58984375, 105.5859375 , 106.58203125,
107.578125  , 108.57421875, 109.5703125 , 110.56640625,
111.5625    , 112.55859375, 113.5546875 , 114.55078125,
115.546875  , 116.54296875, 117.5390625 , 118.53515625,
119.53125   , 120.52734375, 121.5234375 , 122.51953125,
123.515625  , 124.51171875, 125.5078125 , 126.50390625,
127.5       , 128.49609375, 129.4921875 , 130.48828125,
131.484375  , 132.48046875, 133.4765625 , 134.47265625,
135.46875   , 136.46484375, 137.4609375 , 138.45703125,
139.453125  , 140.44921875, 141.4453125 , 142.44140625,
143.4375    , 144.43359375, 145.4296875 , 146.42578125,
147.421875  , 148.41796875, 149.4140625 , 150.41015625,
151.40625   , 152.40234375, 153.3984375 , 154.39453125,
155.390625  , 156.38671875, 157.3828125 , 158.37890625,
159.375     , 160.37109375, 161.3671875 , 162.36328125,
163.359375  , 164.35546875, 165.3515625 , 166.34765625,
167.34375   , 168.33984375, 169.3359375 , 170.33203125,
171.328125  , 172.32421875, 173.3203125 , 174.31640625,
175.3125    , 176.30859375, 177.3046875 , 178.30078125,
179.296875  , 180.29296875, 181.2890625 , 182.28515625,
183.28125   , 184.27734375, 185.2734375 , 186.26953125,
```

```
       187.265625  , 188.26171875, 189.2578125 , 190.25390625,
       191.25      , 192.24609375, 193.2421875 , 194.23828125,
       195.234375  , 196.23046875, 197.2265625 , 198.22265625,
       199.21875   , 200.21484375, 201.2109375 , 202.20703125,
       203.203125  , 204.19921875, 205.1953125 , 206.19140625,
       207.1875    , 208.18359375, 209.1796875 , 210.17578125,
       211.171875  , 212.16796875, 213.1640625 , 214.16015625,
       215.15625   , 216.15234375, 217.1484375 , 218.14453125,
       219.140625  , 220.13671875, 221.1328125 , 222.12890625,
       223.125     , 224.12109375, 225.1171875 , 226.11328125,
       227.109375  , 228.10546875, 229.1015625 , 230.09765625,
       231.09375   , 232.08984375, 233.0859375 , 234.08203125,
       235.078125  , 236.07421875, 237.0703125 , 238.06640625,
       239.0625    , 240.05859375, 241.0546875 , 242.05078125,
       243.046875  , 244.04296875, 245.0390625 , 246.03515625,
       247.03125   , 248.02734375, 249.0234375 , 250.01953125,
       251.015625  , 252.01171875, 253.0078125 , 254.00390625,
       255.        ]),
<BarContainer object of 256 artists>)
```



In [ ]:

Filtering

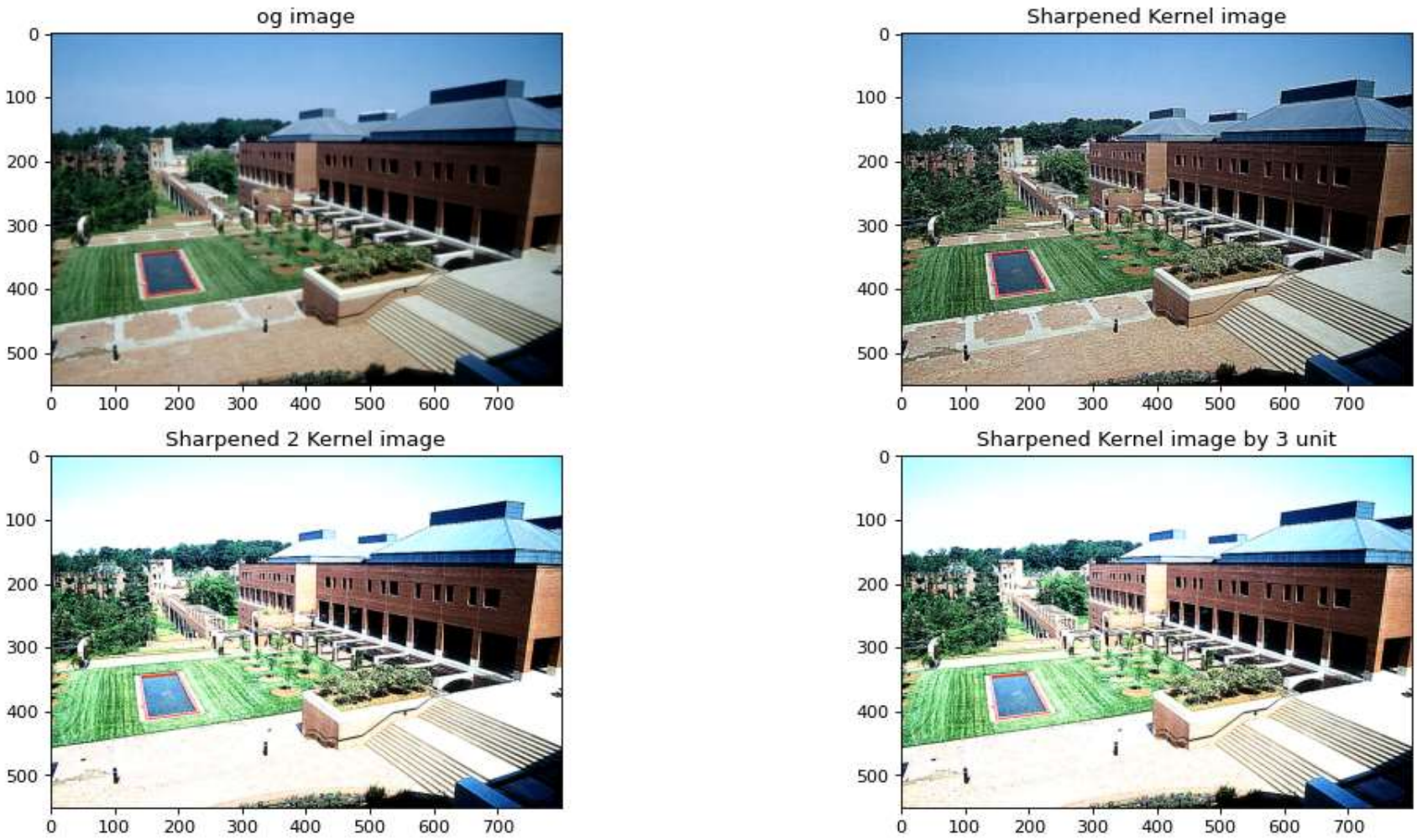
Practice question:

1. Apply Guassian smoothing on the lena image by varying sigma (1,2,0.5,0.25 and 1.5) or by varying radius [use GaussianBlur() function]
2. suggest and apply a kernel to move the image to left by 1 pixel
3. suggest and apply a kernel to move the image to down right by 1 pixel
4. suggest and apply a kernel to sharpen the image by 3 units
5. Apply a median filter and get the kernel for the same
6. apply a averaging filter

In [ ]:
```
kernel_sharpening = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])
kernel_sharpening_2 = np.array([[-1,-1,-1],[-1,10,-1],[-1,-1,-1]])
# kernel_sharpening_3 = np.array([[-3,-3,-3],[-3,10,-3],[-3,-3,-3]])
kernel_sharpening_3 = np.array([[3,3,3],[3,-10,3],[3,3,3]])
```

In [ ]:
```
fig, ax = plt.subplots(2,2,figsize=(16,8))
# fig.tight_layout()
school_img = cv.imread('./images/school.jpg')
# school_img,_,m= getImage('./images/school.jpg')
ax[0][0].imshow(cv.cvtColor(school_img,cv.COLOR_BGR2RGB))
ax[0][0].set_title('og image')

sharpened = cv.filter2D(school_img,-1,kernel_sharpening)
sharpened2 = cv.filter2D(school_img,-1,kernel_sharpening_2)

ax[0][1].set_title('Sharpened Kernel image')
ax[0][1].imshow(cv.cvtColor(sharpened,cv.COLOR_BGR2RGB))

ax[1][0].set_title('Sharpened 2 Kernel image')
ax[1][0].imshow(cv.cvtColor(sharpened2,cv.COLOR_BGR2RGB))
ax[1][1].set_title('Sharpened Kernel image by 3 unit')
ax[1][1].imshow(cv.cvtColor(sharpened2,cv.COLOR_BGR2RGB))
# plt.show()
```

Out[ ]: <matplotlib.image.AxesImage at 0x24d9d7f3e50>



Edge Detection

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

# Load the image in grayscale
lena_gray = cv.imread("./images/rose1.jpg", cv.IMREAD_GRAYSCALE)
```

```
# Apply Gaussian blur to the grayscale image with a 3x3 kernel
gaussian_blurred_img = cv.GaussianBlur(lena_gray, (3, 3), 0)

# Apply Sobel edge detection in the X direction (dx=1, dy=0)
sobel_x = cv.Sobel(src=gaussian_blurred_img, ddepth=cv.CV_64F, dx=1, dy=0, ksize=5)

# Apply Sobel edge detection in the Y direction (dx=0, dy=1)
sobel_y = cv.Sobel(src=gaussian_blurred_img, ddepth=cv.CV_64F, dx=0, dy=1, ksize=5)

# Apply Sobel edge detection in both X and Y directions (dx=1, dy=1)
sobel_xy = cv.Sobel(src=gaussian_blurred_img, ddepth=cv.CV_64F, dx=1, dy=1, ksize=5)
```

In [ ]:

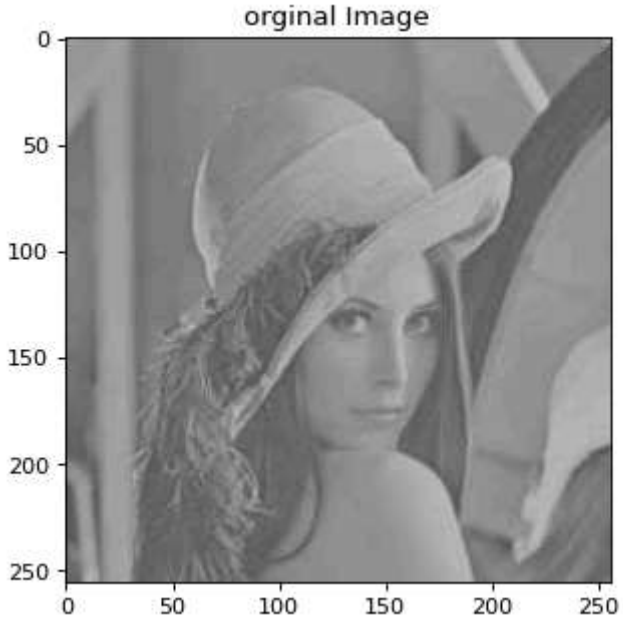
```
# Display the results
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Show the original grayscale image
axs[0, 0].imshow(lena_gray, cmap='gray')
axs[0, 0].set_title("Original Grayscale Image")
axs[0, 0].axis('off')

# Show the Gaussian-blurred image
axs[0, 1].imshow(gaussian_blurred_img, cmap='gray')
axs[0, 1].set_title("Gaussian Blurred Image")
axs[0, 1].axis('off')

# Show the Sobel X image
axs[1, 0].imshow(sobel_x, cmap='gray')
axs[1, 0].set_title("Sobel X Direction")
axs[1, 0].axis('off')

# Show the Sobel Y image
axs[1, 1].imshow(sobel_y, cmap='gray')
axs[1, 1].set_title("Sobel Y Direction")
axs[1, 1].axis('off')

# Show the plot
plt.show()
```

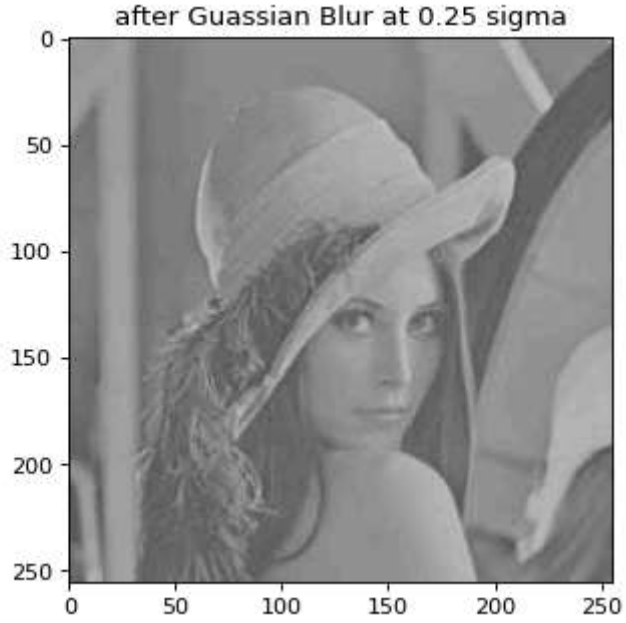
Pratice Question 1:

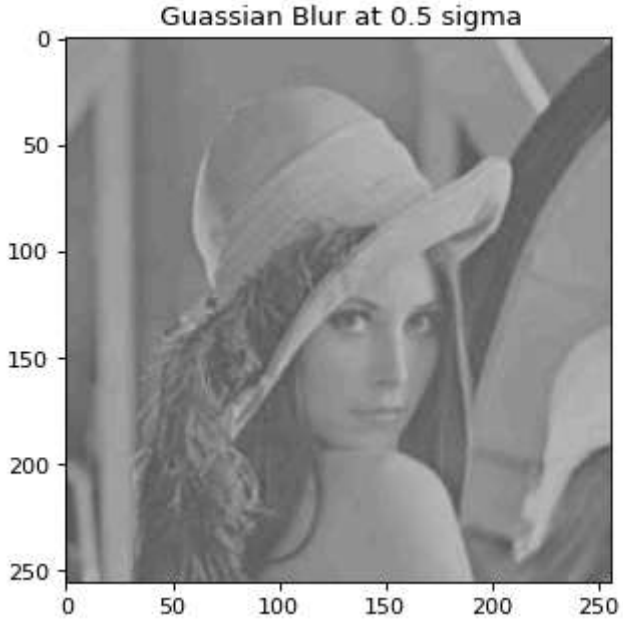
Apply Guassian smoothing on the lena image by varying sigma (1,2,0.5,0.25 and 1.5) or by varying radius [use GaussianBlur() function]

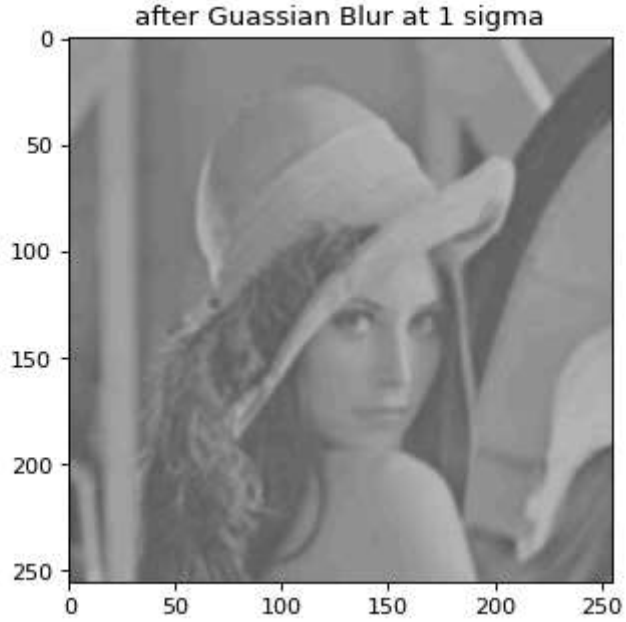
In [ ]:
```
low_img_np= cv.imread('./images/low.jpeg')
gaussian_image = cv.GaussianBlur(low_img_np, (5, 5), 1)
# 1,2,0.5,0.25
fig = plt.figure(figsize=(12,15))
# fig.tight_layout()
col = 2
row = 3

fig.add_subplot(row,col,1)
plt.imshow(low_img_np)
plt.title("orginal Image")


fig.add_subplot(row,col,2)
plt.title("after Guassian Blur at 0.25 sigma")
plt.imshow(cv.GaussianBlur(low_img_np, (5, 5), 0.25))

fig.add_subplot(row,col,3)
plt.title(" Guassian Blur at 0.5 sigma")
plt.imshow(cv.GaussianBlur(low_img_np, (5, 5), 0.5))

fig.add_subplot(row,col,4)
plt.title("after Guassian Blur at 1 sigma")
plt.imshow(cv.GaussianBlur(low_img_np, (5, 5), 1))

fig.add_subplot(row,col,5)
plt.title("Guassian Blur at 2 sigma")
plt.imshow(cv.GaussianBlur(low_img_np, (5, 5), 2))
```

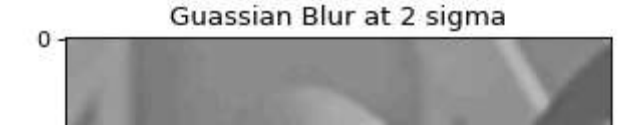
Out[ ]:
```
<matplotlib.image.AxesImage at 0x24d9c9f18d0>
```

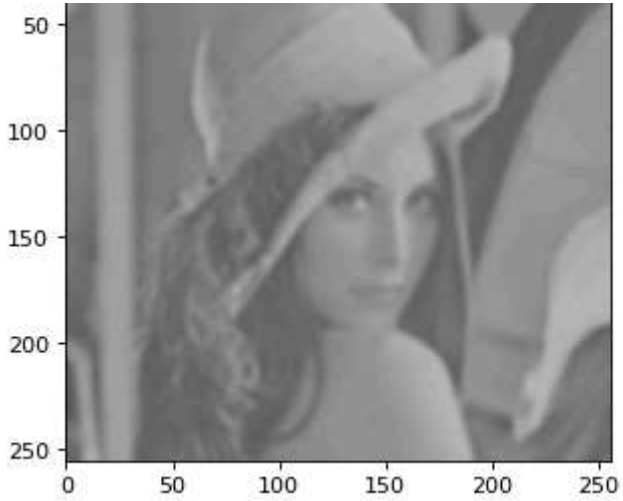

orginal Image
0
50
100
150
200
250
0
50
100
150
200
250


after Guassian Blur at 0.25 sigma
0
50
100
150
200
250
0
50
100
150
200
250


Guassian Blur at 0.5 sigma
0
50
100
150
200
250
0
50
100
150
200
250


after Guassian Blur at 1 sigma
0
50
100
150
200
250
0
50
100
150
200
250


Guassian Blur at 2 sigma
0



Pratice Question 2,3,4,5,6:

2. suggest and apply a kernel to move the image to left by 1 pixel
3. suggest and apply a kernel to move the image to down right by 1 pixel
4. suggest and apply a kernel to sharpen the image by 3 units
5. Apply a median filter and get the kernel for the same
6. apply a averaging filter

a)left shift kernel
0 0 0
0 0 1
0 0 0
b) average filter
0.111 0.111 0.111
0.111 0.111 0.111
0.111 0.111 0.111
c) move up by a pixel
d) move down and left pixel
e) horizontal edges
f) vertical edges
g)blur image

In [ ]:
```
left, right, up, down = 1,1,1,1
# left, right, up, down = 2,2,2,2
# left, right, up, down = 3,3,3,3
# left, right, up, down = 5,5,5,5
depth = -1
school_img = rose_img
left_shift_kernel = np.array([[0,0,0],
                      [0,0,left],
                      [0,0,0]]
)
right_shift_kernel = np.array([[0,0,0],
                       [right,0,0],
                       [0,0,0]])
```

```
up_shift_kernel = np.array([[0,up,0],
                    [0,0,0],
                    [0,0,0]])

down_shift_kernel = np.array([[0,0,0],
                      [0,0,0],
                      [0,down,0]])

down_left_shift_kernel = np.array([[0,0,0],
                           [0,0,left],
                           [0,down,0]])

blur_kernel = np.array(  [[0.111,0.111,0.111],
                           [0.111,0.111,0.111],
                           [0.111,0.111,0.111]])
sigma = 8
gaussian_blur_kernel = np.array(  [[sigma+1/16,sigma+2/16,sigma+1/16],
                           [sigma+2/16,sigma+4/16,sigma+2/16],
                           [sigma+1/16,sigma+2/16,sigma+1/16]])


left_shift_img = cv.filter2D(school_img,depth,left_shift_kernel)
right_shift_img = cv.filter2D(school_img,depth,right_shift_kernel)
up_shift_img = cv.filter2D(school_img,depth,up_shift_kernel)
down_shift_img = cv.filter2D(school_img,depth,down_shift_kernel)
down_left_shift_img = cv.filter2D(school_img,depth,down_left_shift_kernel)

blur_kernel_img = cv.filter2D(school_img,depth,blur_kernel)
gaussian_blur_kernel_img = cv.filter2D(school_img,depth,blur_kernel)
```

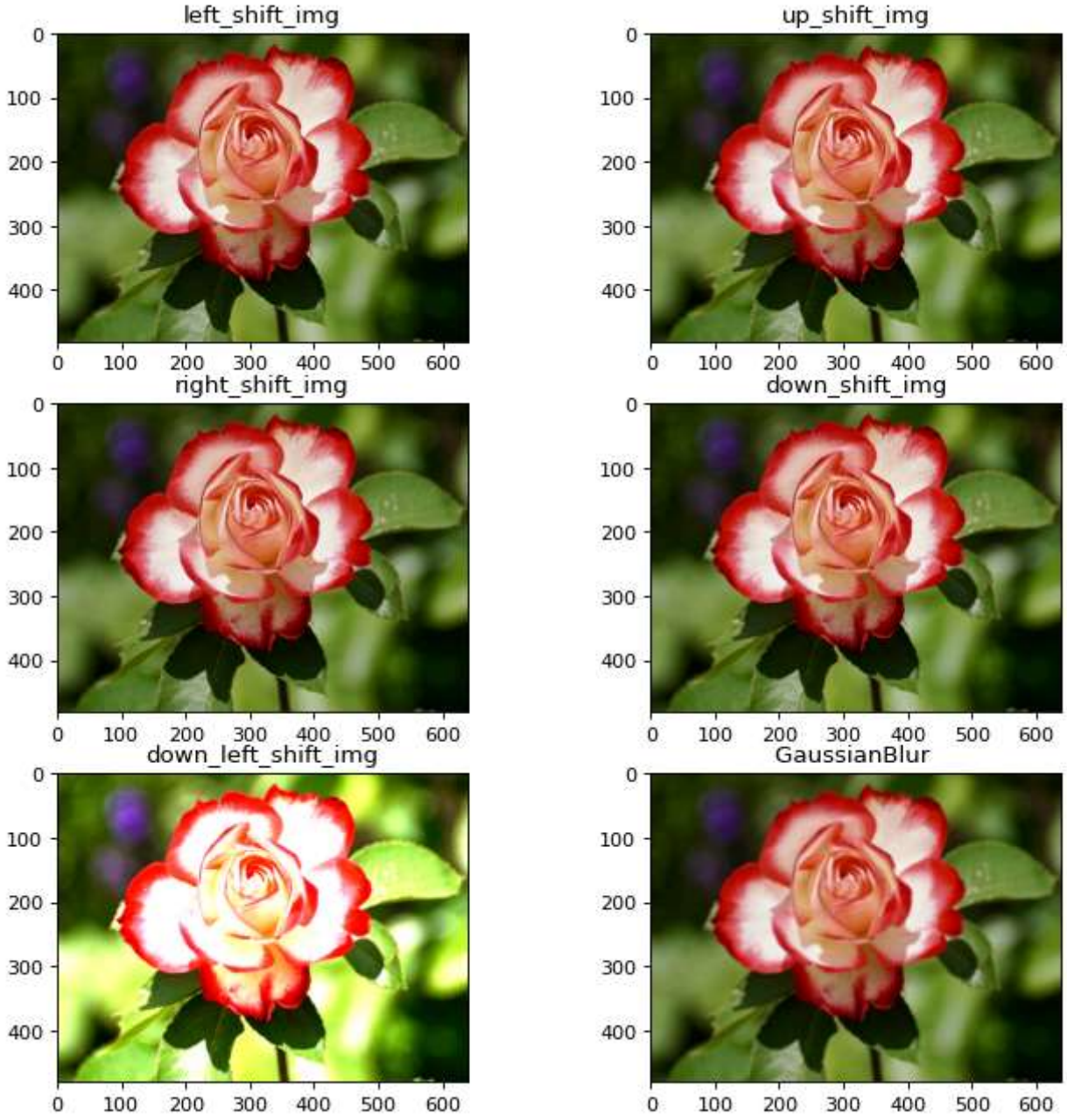
```
to plot multiple images
fig=plt.figure(figsize=(10,7))
fig.add_subplot(1,2,1)
plt.imshow(image1)
plt.title("First")
fig.add_subplot(1,2,2)
plt.imshow(image2)
plt.title("Second")
```

In [ ]:

```
fig = plt.figure(figsize=(10,20))
fig.tight_layout()
row = 6
col = 2

fig.add_subplot(row,col,1)
plt.imshow(left_shift_img)
plt.title("left_shift_img")

fig.add_subplot(row,col,2)
plt.imshow(up_shift_img)
plt.title("up_shift_img")

fig.add_subplot(row,col,3)
plt.imshow(right_shift_img)
plt.title("right_shift_img")


fig.add_subplot(row,col,4)
plt.imshow(down_shift_img)
```

```
plt.title("down_shift_img")


fig.add_subplot(row,col,5)
plt.imshow(down_left_shift_img)
plt.title("down_left_shift_img")
gaussian_image = cv.GaussianBlur(school_img, (5, 5), 8)
fig.add_subplot(row,col,6)
plt.imshow(gaussian_image,cmap="gray")
plt.title("GaussianBlur ")
```

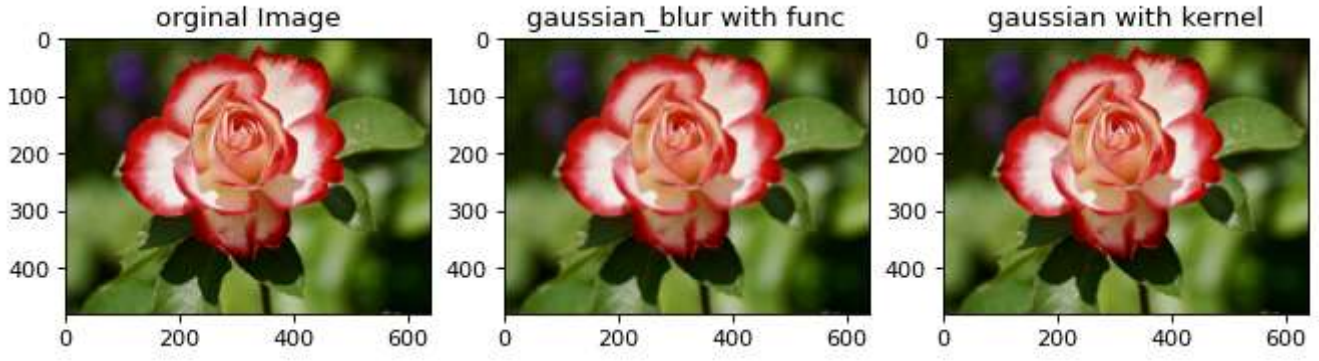
Out[ ]:
```
Text(0.5, 1.0, 'GaussianBlur ')
```



In [ ]:
```
fig = plt.figure(figsize=(10,7))
fig.tight_layout()
row = 1
```

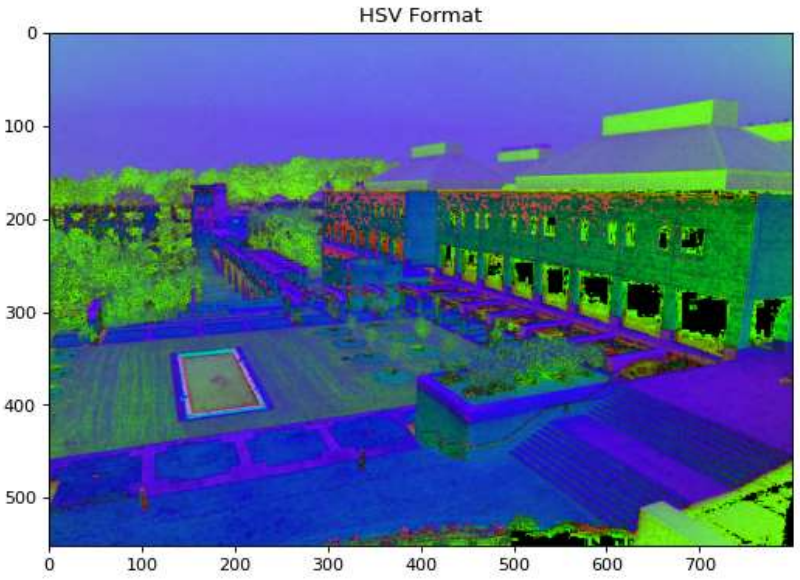
```
col = 3
og_gaussian_image = rose_img
gaussian_image = cv.GaussianBlur(og_gaussian_image, (5, 5), 8)
fig.add_subplot(row,col,1)
plt.imshow(og_gaussian_image)
plt.title("orginal Image")
fig.add_subplot(row,col,2)
plt.imshow(gaussian_image)
plt.title("gaussian_blur with func")

fig.add_subplot(row,col,3)
plt.imshow(gaussian_blur_kernel_img)
plt.title("gaussian with kernel")
```

Out[ ]: Text(0.5, 1.0, 'gaussian with kernel')



In [ ]:
```
image_path = './images/school.jpg'
school_img = cv.imread(image_path)
hsv = cv.cvtColor(school_img,cv.COLOR_BGR2HSV)

fig = plt.figure(figsize=(17,10))
col, row = 1,2
fig.add_subplot(col,row,1)

plt.imshow(school_img,cmap='gray')
plt.title("Orginal Image")

fig.add_subplot(col,row,2)
plt.title("HSV Format")
plt.imshow(hsv,cmap='gray')
```

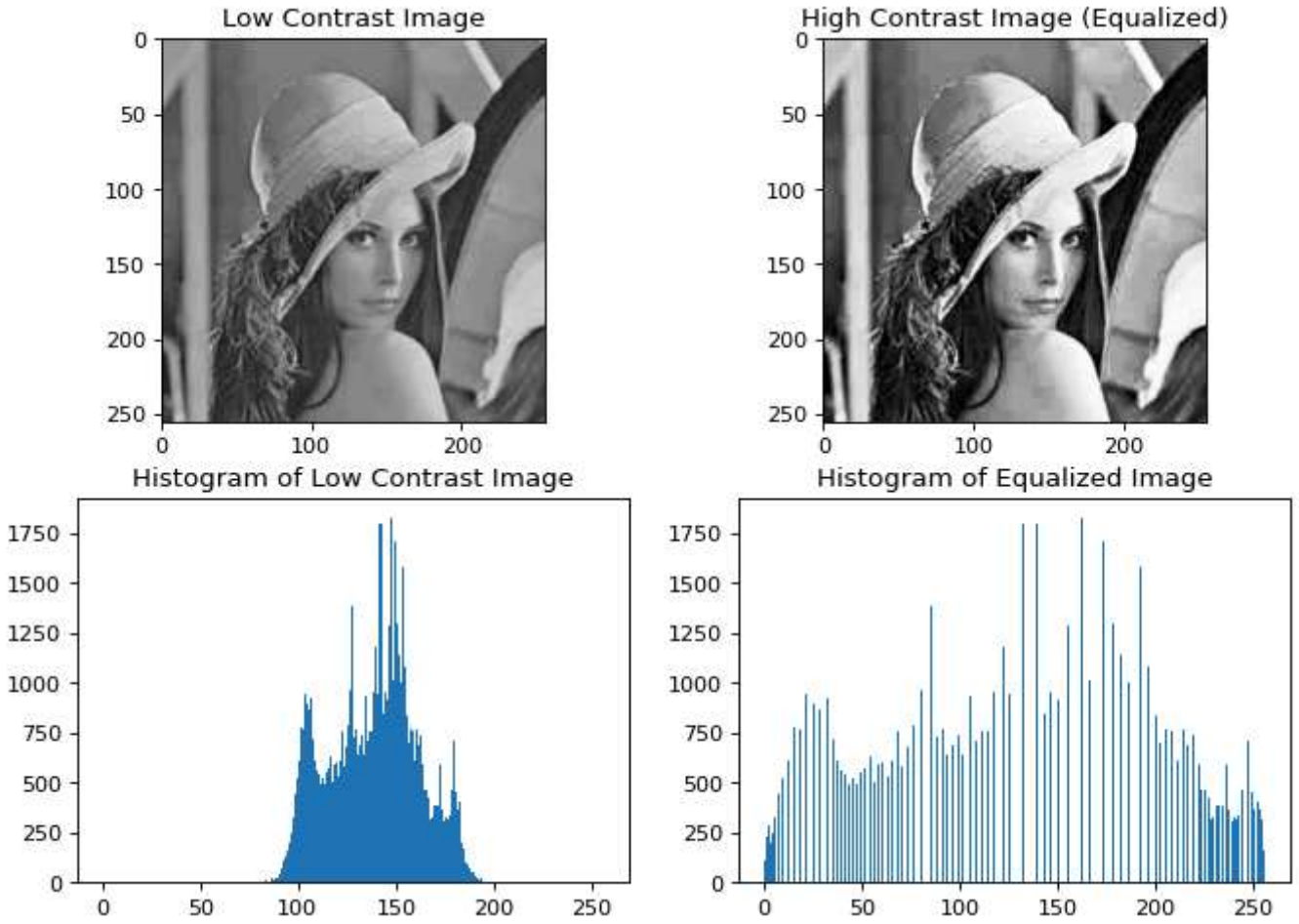
Out[ ]: <matplotlib.image.AxesImage at 0x24da4c6d650>



search low contrast images
download one of image
read the image
show the image
next cell
plot histogram of it
next cell

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

# Load the low-contrast image
low_contrast_img_path = './images/low.jpeg'
low_contrast_img = plt.imread(low_contrast_img_path, 0)

# Convert the image to NumPy array and data type int
low_contrast_img = cv.cvtColor(low_contrast_img, cv.COLOR_RGB2GRAY)
low_contrast_img = np.array(low_contrast_img)
```

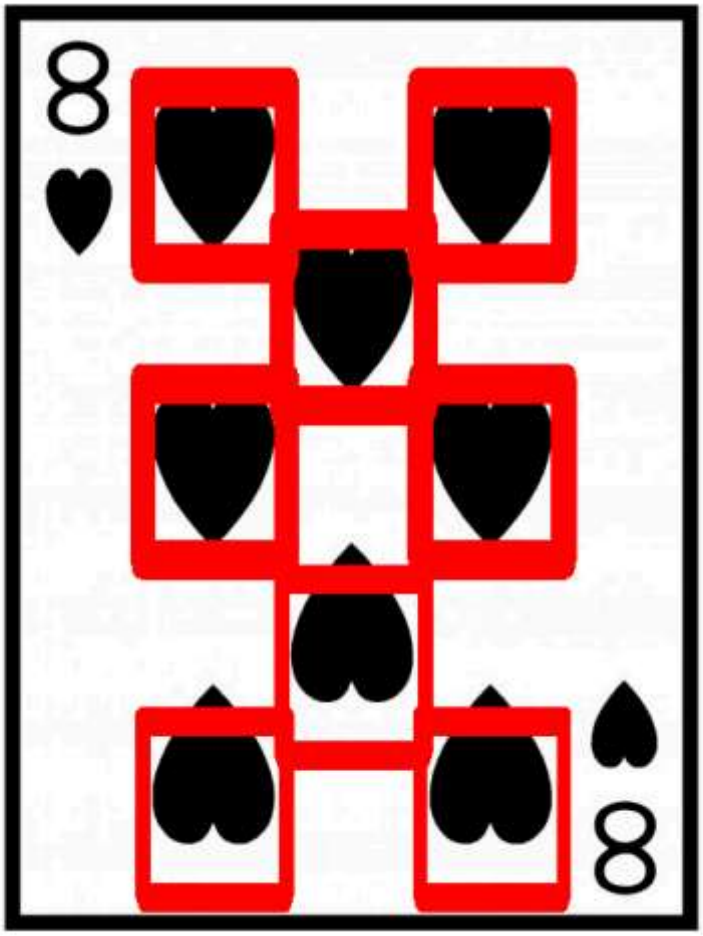
```python
# low_contrast_img = low_contrast_img.astype(np.uint8)

# Perform histogram equalization on the low-contrast image
equalized_img = cv.equalizeHist(low_contrast_img)

# Create a figure with the specified size
fig = plt.figure(figsize=(10, 7))

# Plot the original low-contrast image
fig.add_subplot(2, 2, 1)
plt.imshow(low_contrast_img, cmap='gray')
plt.title("Low Contrast Image")

# Plot the equalized high-contrast image
fig.add_subplot(2, 2, 2)
plt.imshow(equalized_img, cmap='gray')
plt.title("High Contrast Image (Equalized)")

# Plot the histogram of the low-contrast image
fig.add_subplot(2, 2, 3)
plt.hist(low_contrast_img.ravel(), bins=256, range=[0, 256])
plt.title("Histogram of Low Contrast Image")

# Plot the histogram of the equalized image
fig.add_subplot(2, 2, 4)
plt.hist(equalized_img.ravel(), bins=256, range=[0, 256])
plt.title("Histogram of Equalized Image")

# Display the plots
plt.show()
```



Template Matching

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

# File paths for the source image and the template image
source_image_path = './images/card.jpeg'
template_image_path = './images/temp.png'

# Load the source image and template image in grayscale
```

```python
source_image = cv.imread(source_image_path)
template_image = cv.imread(template_image_path, cv.IMREAD_GRAYSCALE)

# Convert the source image to grayscale
source_image_gray = cv.cvtColor(source_image, cv.COLOR_BGR2GRAY)

# Get the width and height of the template image
template_width, template_height = template_image.shape[::-1]

# Perform template matching using the TM_CCORR_NORMED method
matching_result = cv.matchTemplate(source_image_gray, template_image, cv.TM_CCORR_NORMED)

# Define the matching threshold
matching_threshold = 0.83

# Find the locations where the matching score meets or exceeds the threshold
matching_locations = np.where(matching_result >= matching_threshold)

# Iterate over the found locations and draw rectangles on the source image
for top_left_corner in zip(*matching_locations[::-1]):
    bottom_right_corner = (
        top_left_corner[0] + template_width,
        top_left_corner[1] + template_height
    )
    cv.rectangle(
        source_image, top_left_corner, bottom_right_corner,
        (0, 0, 255), 1  # Red color, 1 pixel border thickness
    )

# File path for saving the result image
result_image_path = './res.png'

# Save the annotated source image as an image file
cv.imwrite(result_image_path, source_image)

# Display the resulting image
result_image = plt.imread(result_image_path)
plt.imshow(result_image)
plt.axis('off')  # Hide the axes for better visualization
plt.show()
```



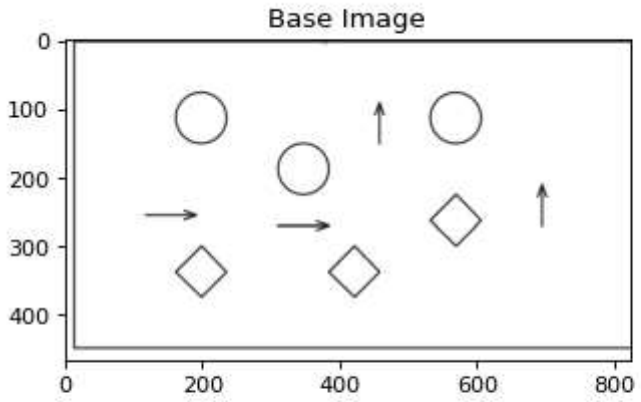
Practice question

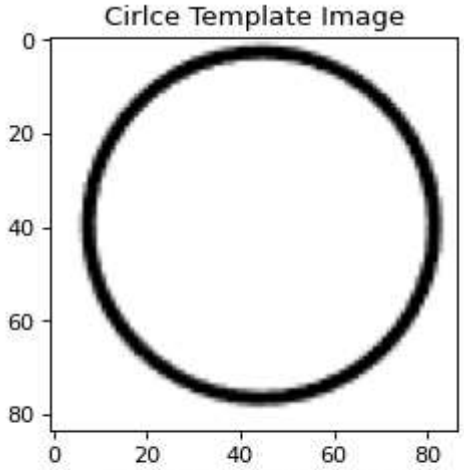
1. try to match a group of similar objects (2 stars as above)
2. try to match a group of diffrent objects (a stars and a cup etc.)

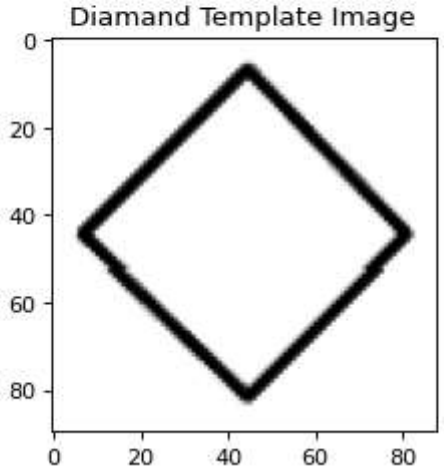
In [ ]:
```
fig = plt.figure(figsize=(10,7))

baseImg = cv.imread('./images/baseImage.png')
baseImg = cv.cvtColor(baseImg,cv.COLOR_BGR2GRAY)
circleTemplate = cv.imread('./images/circleTemplate.png')
```

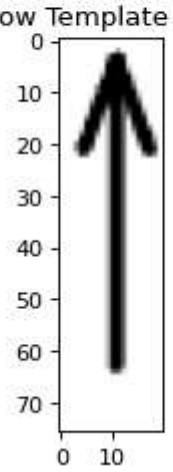
```
cirlceTemplate = cv.cvtColor(circleTemplate, cv.COLOR_BGR2GRAY)

diamandTemplate = cv.imread('./images/diamandTemplate.png')

diamandTemplate = cv.cvtColor(diamandTemplate, cv.COLOR_BGR2GRAY)

arrowTemplate = cv.imread('./images/arrowTemplate.png')
arrowTemplate = cv.cvtColor(arrowTemplate, cv.COLOR_BGR2GRAY)

fig.add_subplot(2,2,1)
plt.imshow(baseImg,cmap='gray')
plt.title("Base Image")

fig.add_subplot(2,2,2)
plt.imshow(cirlceTemplate,cmap='gray')
plt.title("Cirlce Template Image")

fig.add_subplot(2,2,3)
plt.imshow(diamandTemplate,cmap='gray')
plt.title("Diamand Template Image")

fig.add_subplot(2,2,4)
plt.imshow(arrowTemplate,cmap='gray')
plt.title("arrow Template Image")
```

Out[ ]: Text(0.5, 1.0, 'arrow Template Image')









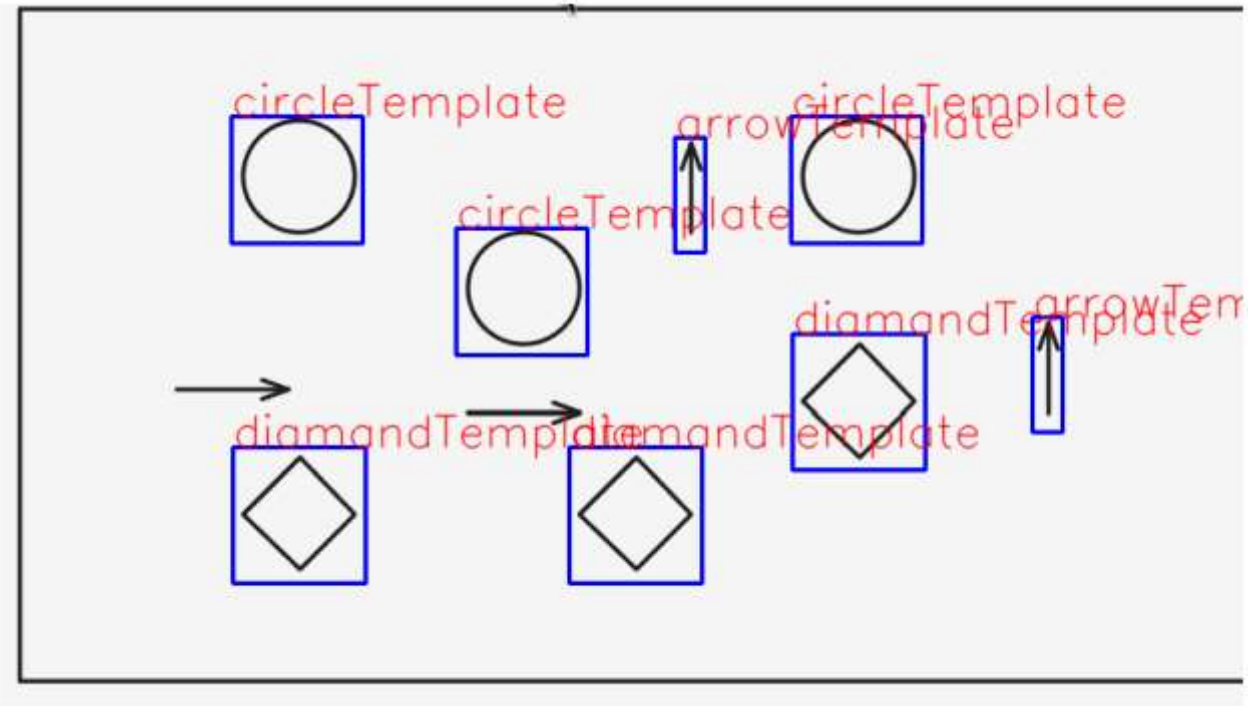
In [ ]:
```
listTemplate = []

image_folder = "./templates/"
for filename in os.listdir(image_folder):
    template_img = cv.imread(os.path.join(image_folder, filename))
    template_img = cv.cvtColor(template_img, cv.COLOR_BGR2GRAY)
    listTemplate.append((filename.split('.')[0], template_img))
```

In [ ]:
```
import MTM
from MTM import matchTemplates
hits = matchTemplates(listTemplate,
                      baseImg,
```

```
                        score_threshold=0.9,
                        searchBox=(0, 0, 3000, 750),
                        method=cv.TM_CCOEFF_NORMED,
                        maxOverlap=0.1)
```

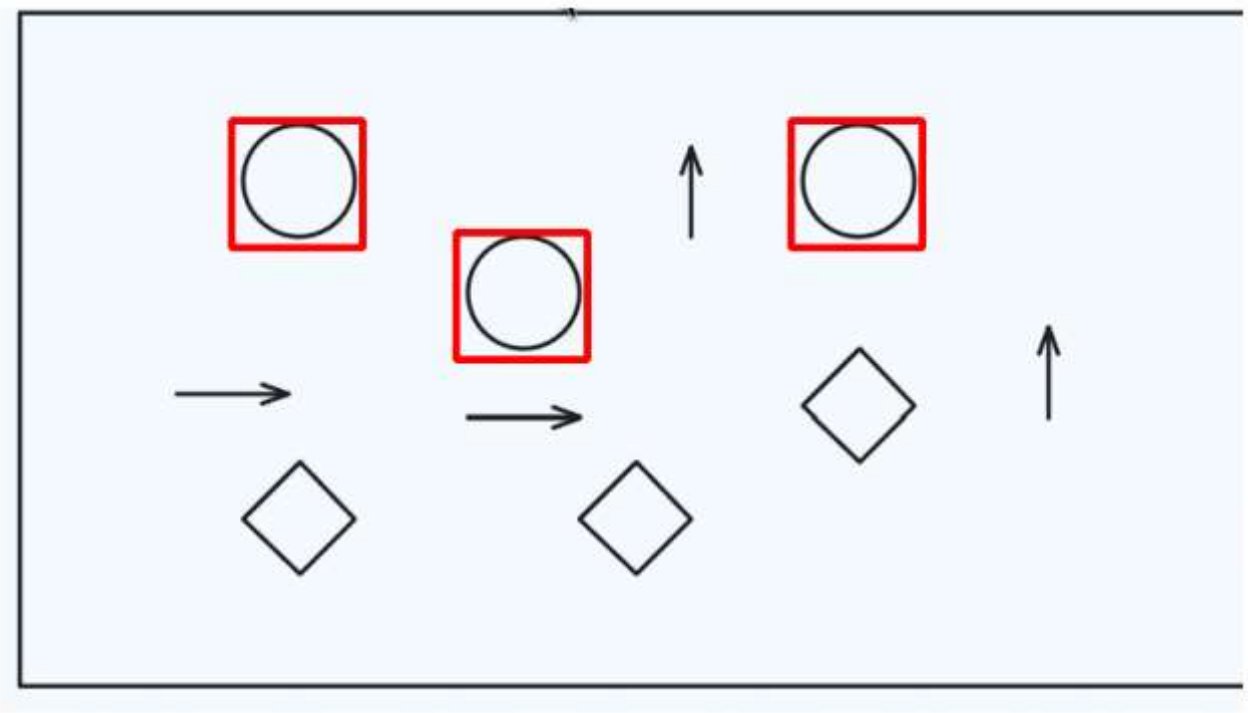
In [ ]:

```
import matplotlib.pyplot as plt
from MTM import drawBoxesOnRGB
overlay = drawBoxesOnRGB(baseImg,
                         hits,
                         showLabel = True,
                         labelColor=(255, 0, 0),
                         boxColor = (0, 0, 255),
                         labelScale=1,
                         boxThickness = 2)

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



In [ ]:

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

img_rgb = cv.imread('./images/baseImage.png')

img_gray = cv.cvtColor(img_rgb, cv.COLOR_BGR2GRAY)
template = cv.imread('./images/circleTemplate.png', cv.IMREAD_GRAYSCALE)
template2 = cv.imread('./images//diamandTemplate.png', cv.IMREAD_GRAYSCALE)

w, h = template.shape[::-1]

res = cv.matchTemplate(img_gray,template,cv.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
 cv.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,0,255), 2)



result_image_path = './res.png'

# Save the annotated source image as an image file
cv.imwrite(result_image_path, img_rgb)

# Display the resulting image
result_image = plt.imread(result_image_path)
plt.imshow(result_image)
plt.axis('off')  # Hide the axes for better visualization
plt.show()
```



Refrence:

1. httpsU.sbme:tu.tQnals.g
2. https:dZ.learmpen.cynmLedge:.detectiQ.n:us.ing-ApencyL
3.

refrpr-eyious-acticle 4. https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781 bl

Practice Question

1. Check for the optimal number of clusters in an image either using silloutte or elbow method
2. Quantify the quality of the clustered image using DBI or silloutte methold

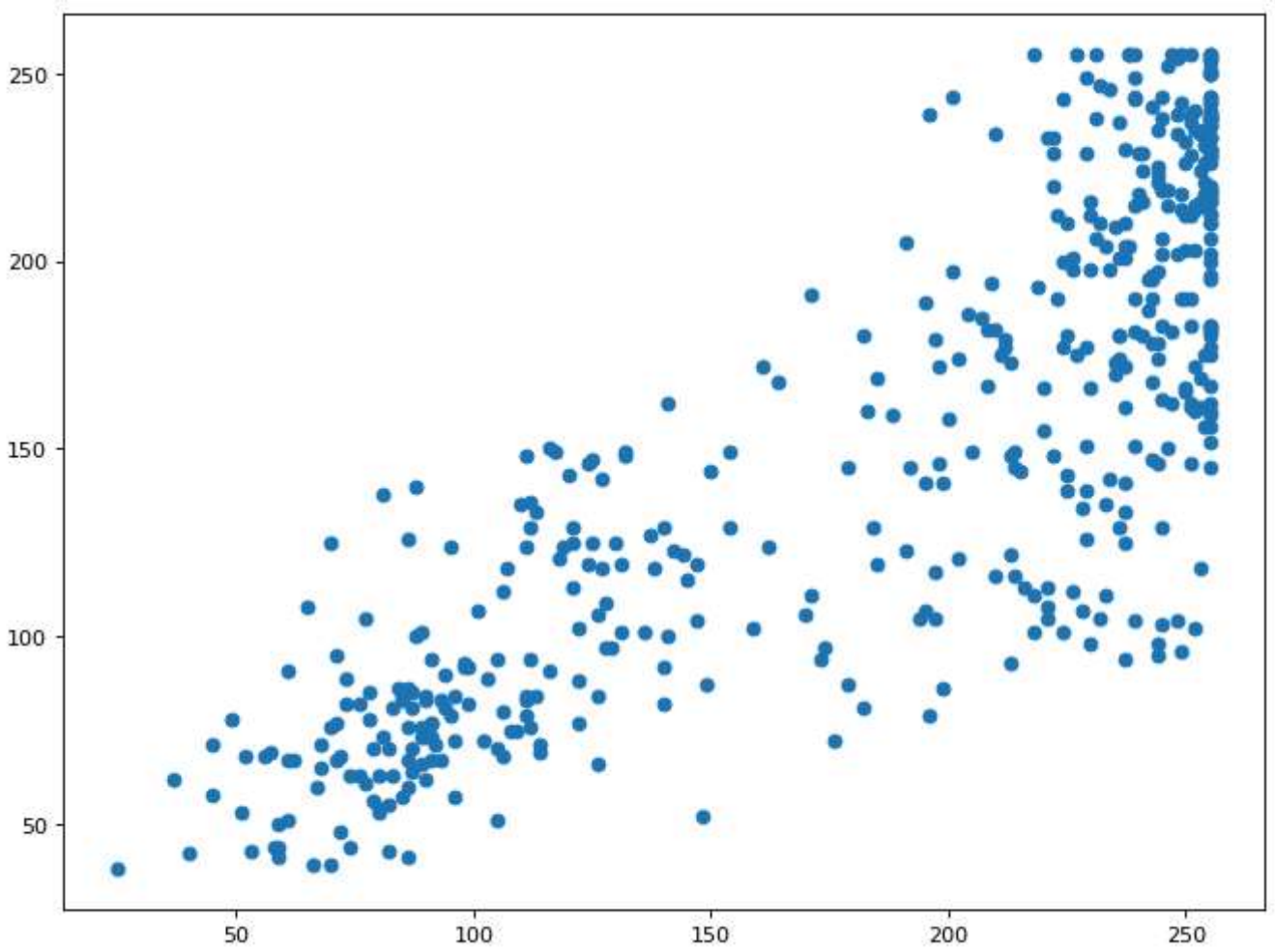
check for the optimal number of clusters in an images either using silloutte or eblow method()

In [ ]:
```
img = plt.imread('./images/sunflower.jpeg')
img_np = np.array(img)
img_np = img_np.astype('int')
plt.imshow(img_np,cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x24dab389550>
```

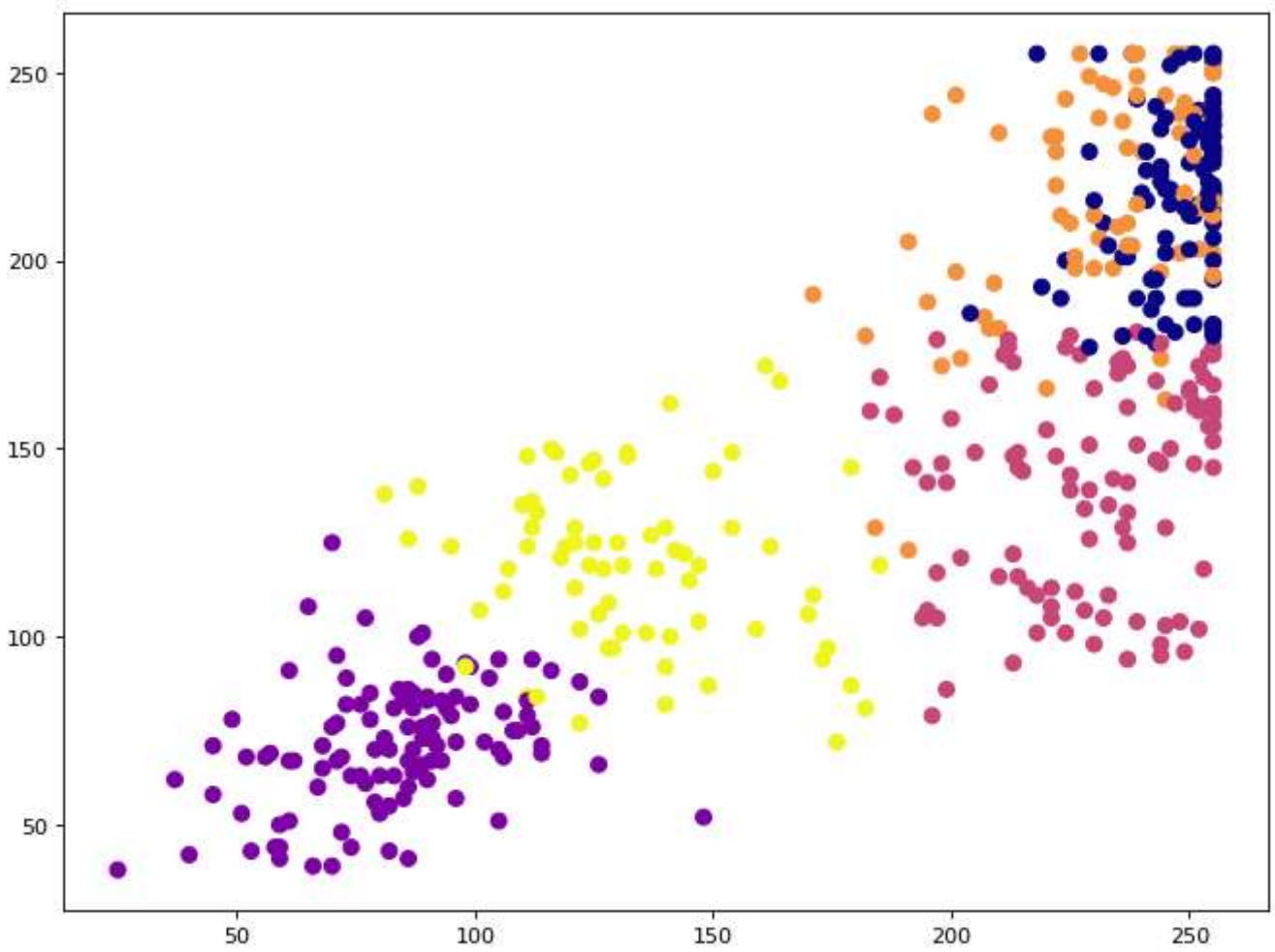
In [ ]:
```
x,y = img_np[:2]

plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':80})
plt.scatter(x[:, 0], x[:, 1])
```

Out[ ]: <matplotlib.collections.PathCollection at 0x24dab602450>



In [ ]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score
Cluster = KMeans(n_clusters=5)
Cluster.fit(x)
y_pred = Cluster.predict(x)
# print(y_pred)
```

In [ ]:
```
plt.scatter(x[:, 0], x[:, 1], c=y_pred, s=50, cmap='plasma')
plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':80})
```



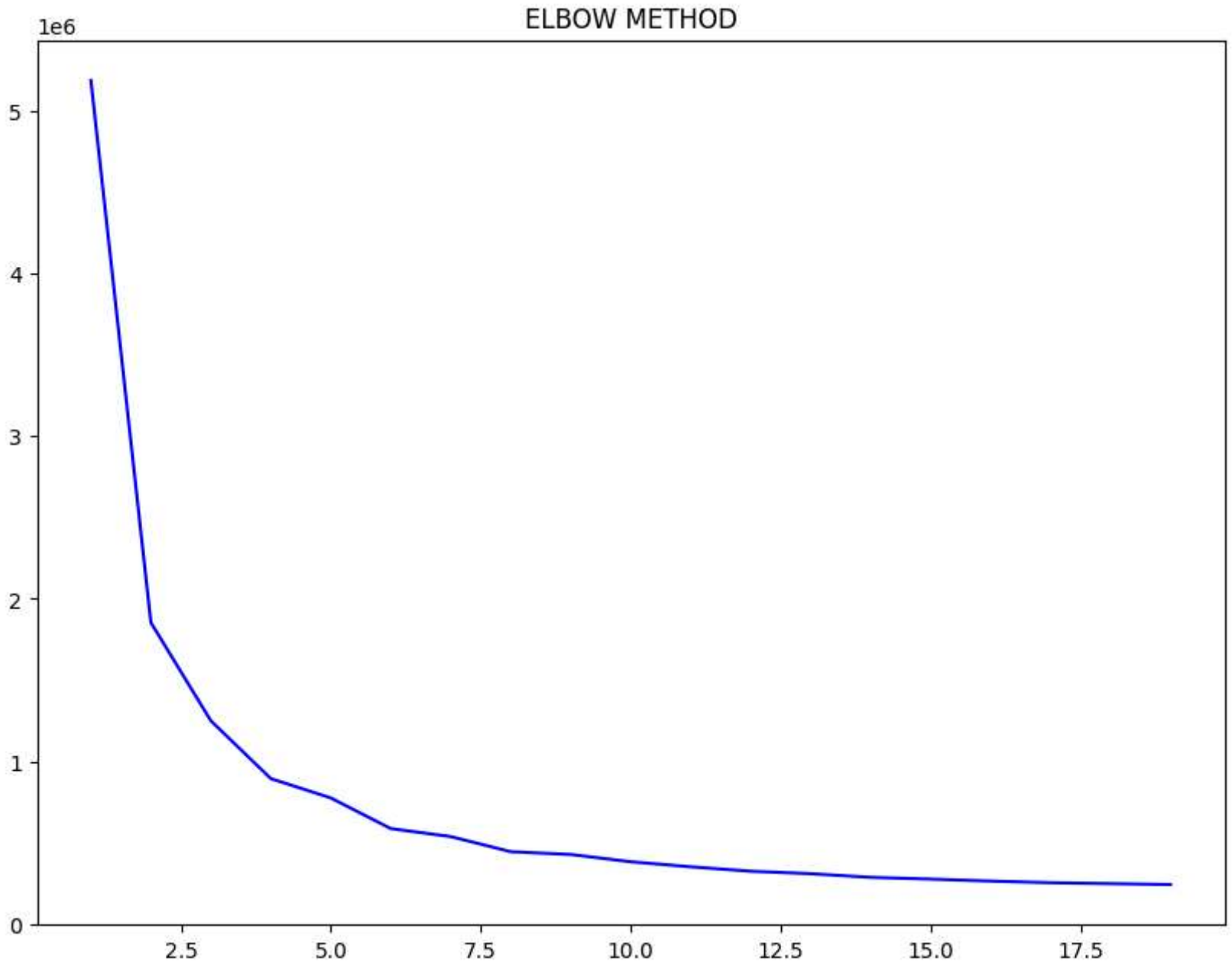
Elbow method

In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Array to store inertia values (sum of squared distances)
elbow = []

# Calculate inertia for different numbers of clusters
for i in range(1, 20):
```

```
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=101)
    kmeans.fit(x)
    elbow.append(kmeans.inertia_)

# Plot the elbow method using Seaborn's lineplot
plt.figure(figsize=(10, 7.5), dpi=100)
plt.title('ELBOW METHOD')
sns.lineplot(x=range(1, 20), y=elbow, color='blue')
# plt.show()
```

Out[ ]: <Axes: title={'center': 'ELBOW METHOD'}>



In [ ]:
```
import cv2
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
og_img = cv2.imread("./images/rose1.jpg")
img = cv2.cvtColor(og_img, cv2.COLOR_BGR2RGB)

img=cv2.cvtColor(og_img,cv2.COLOR_BGR2RGB)
vectorized = img.reshape((-1,3))
vectorized = np.float32(vectorized)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 3
attempts=10
ret,label,center=cv2.kmeans(vectorized,K,None,criteria,attempts,cv2.KMEANS_PP_CENTERS)
center = np.uint8(center)
res = center[label.flatten()]
result_image = res.reshape((img.shape))
figure_size = 10
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(img)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(result_image)
plt.title('Segmented Image when K = %i' % K), plt.xticks([]), plt.yticks([])
plt.show()
```

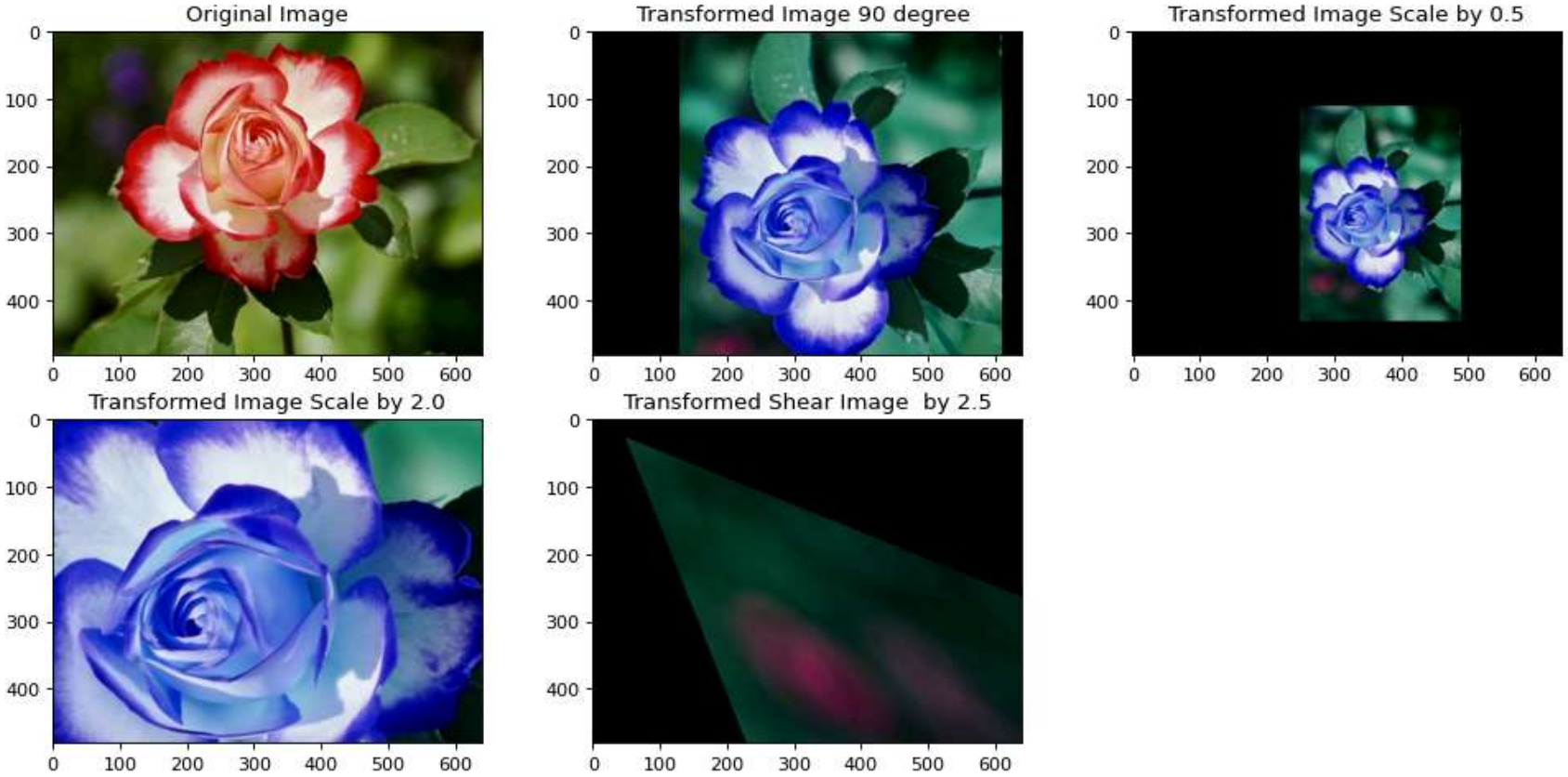
Practice question

1. Try Affine transformation
2. try scaling the object in the image by sx=2 and sy=2
3. try scaling a object in the image by sx=0.5,sy=0.5
4. try shearing with sx=2.5 and sy=1.5
5. rotate the object by 90degree
6. tranform the object to new coordinates

In [ ]:

```
img = cv.imread('./images/rose1.jpg')


rows, cols = img.shape[:2]

img_rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)
center = (cols // 2, rows // 2)
angle = 90  # Rotate by 45 degrees
tx, ty = 50, 30  # Translate by 50 units in x and 30 units in y

scale = 2

rotation_matrix = cv.getRotationMatrix2D(center, angle, 0.5)
rotation_matrix_o = cv.getRotationMatrix2D(center, angle, 1)
rotation_matrix_s = cv.getRotationMatrix2D(center, angle, scale)

rotation_matrix[0, 2] += tx
rotation_matrix[1, 2] += ty

rotation_matrix_o[0, 2] += tx
rotation_matrix_o[1, 2] += ty

rotation_matrix_s[0, 2] += tx
rotation_matrix_s[1, 2] += ty

transformed_img = cv.warpAffine(img, rotation_matrix_o, (cols, rows))
transformed_img1 = cv.warpAffine(img, rotation_matrix, (cols, rows))
transformed_img2 = cv.warpAffine(img, rotation_matrix_s, (cols, rows))
```

```
sx = 2.5
sy = 2.5
shear_matrix = np.array([
[1, sx, tx],
[sy, 1, ty]
], dtype=float)
# Apply the affine transformation using the shearing matrix
transformed_img_shear = cv.warpAffine(img, shear_matrix, (cols, rows))
# Apply the affine transformation

# Display the original and transformed images using matplotlib
plt.figure(figsize=(15, 7))

plt.subplot(2, 3, 1)
plt.imshow(img_rgb)
plt.title('Original Image')

plt.subplot(2, 3, 2)
plt.imshow(transformed_img)
plt.title('Transformed Image 90 degree ')
plt.subplot(2, 3, 3)
plt.imshow(transformed_img1)
plt.title('Transformed Image Scale by 0.5')
plt.subplot(2, 3, 4)
plt.imshow(transformed_img2)
plt.title('Transformed Image Scale by 2.0')
plt.subplot(2, 3, 5)
plt.imshow(transformed_img_shear)
plt.title('Transformed Shear Image  by 2.5')

plt.show()
```


Original Image
Transformed Image 90 degree
Transformed Image Scale by 0.5
Transformed Image Scale by 2.0
Transformed Shear Image by 2.5
0
100
200
300
400
500
600