```
## shree ganeshya namaha
```

```
import numpy as np
import pandas as pd
import cv2
from skimage import io
import matplotlib.pyplot as plt
from PIL import Image
```

uploading a image from drive and finding dimensions of the image

```
img=io.imread('nature.jpg')
img=np.array(img)
img=img.astype('int')
print(img.shape)
print(img.dtype)
```

```
(183, 275, 3)
int64
```

```
plt.imshow(img)
plt.title("Nature Image")
```

```
Text(0.5, 1.0, 'Nature Image')
```

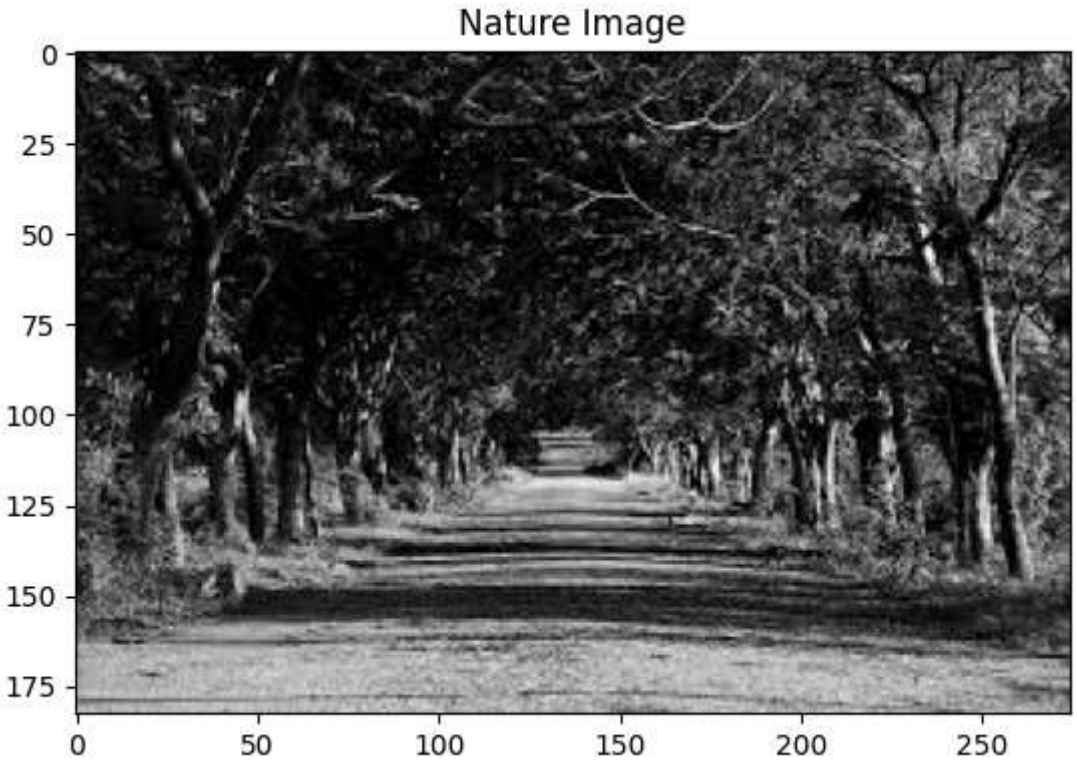
converting a image to gray scale

1. reading the image from channel number 0

2. use the cv.color function along with cv2.COLOR_BGR2GRAY as constant for image conversion
3. Split function also can be used to split the image into vaarious channels

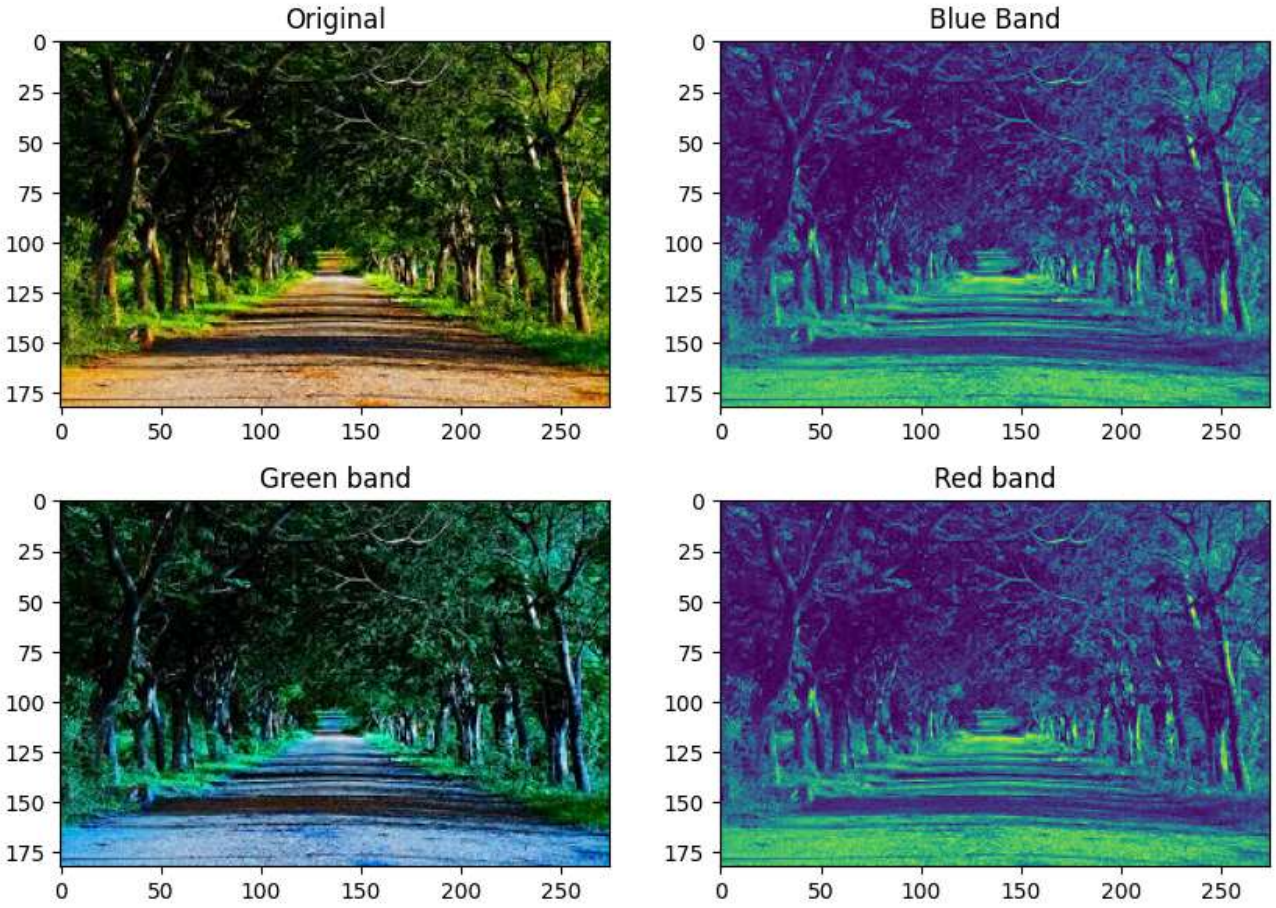
```
img_gray = cv2.imread('nature.jpg', 0)
plt.imshow(img_gray,cmap='gray')
plt.title("Nature Image")
```

```
Text(0.5, 1.0, 'Nature Image')
```



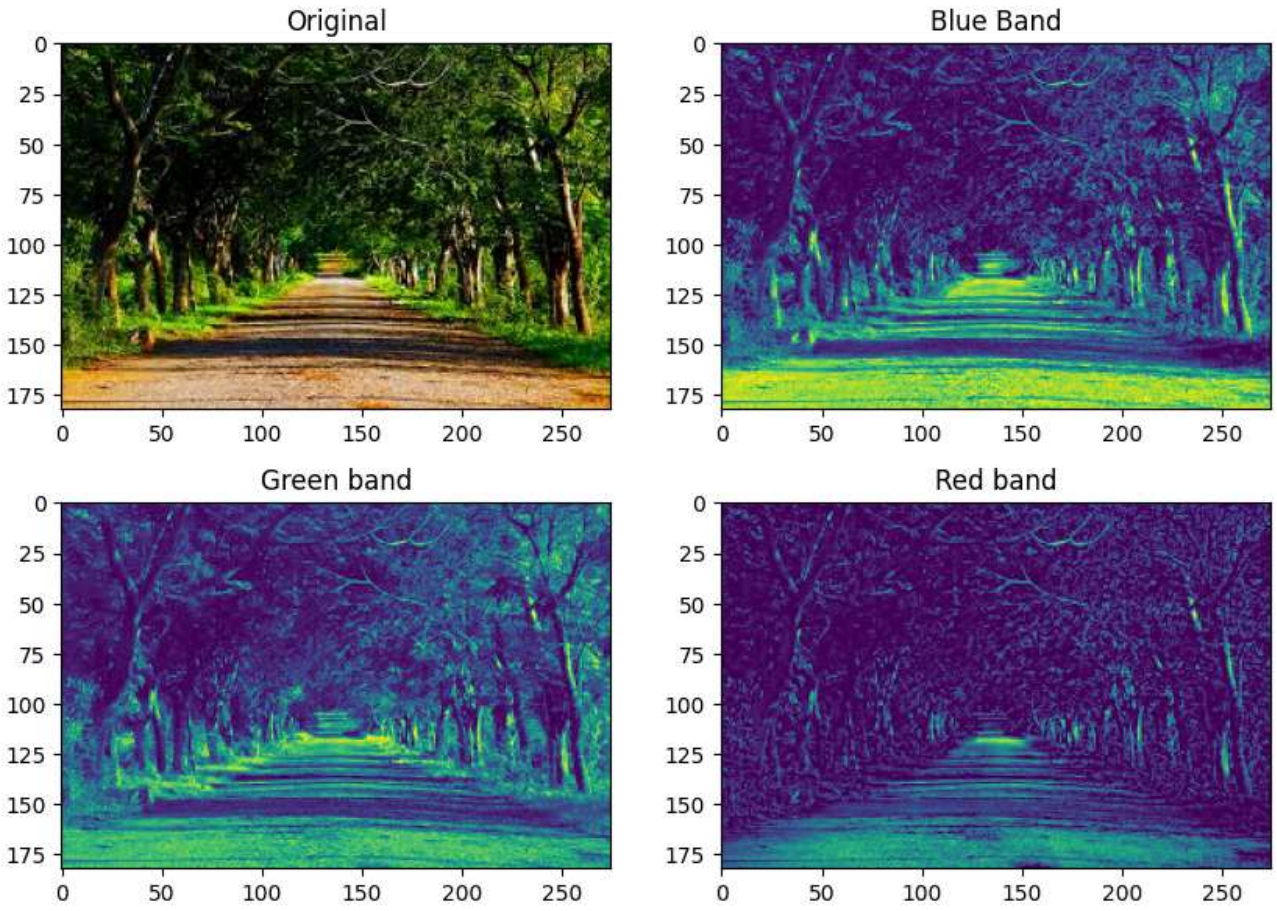
```
blue = cv2.imread('nature.jpg', 0)
red=cv2.imread('nature.jpg', 2)
green=cv2.imread('nature.jpg', 1)


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

fig.add_subplot(2,2,1)
plt.imshow(img)
plt.title("Original")

fig.add_subplot(2,2,2)
plt.imshow(blue)
plt.title("Blue Band")

fig.add_subplot(2,2,3)
plt.imshow(green)
plt.title("Green band")

fig.add_subplot(2,2,4)
plt.imshow(red)
plt.title("Red band")
```

```
Text(0.5, 1.0, 'Red band')
```



```
(b_channel, g_channel, r_channel) = cv2.split(img)

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

fig.add_subplot(2,2,1)
plt.imshow(img)
plt.title("Original")

fig.add_subplot(2,2,2)
plt.imshow(b_channel)
plt.title("Blue Band")

fig.add_subplot(2,2,3)
plt.imshow(g_channel)
plt.title("Green band")

fig.add_subplot(2,2,4)
plt.imshow(r_channel)
plt.title("Red band")
```

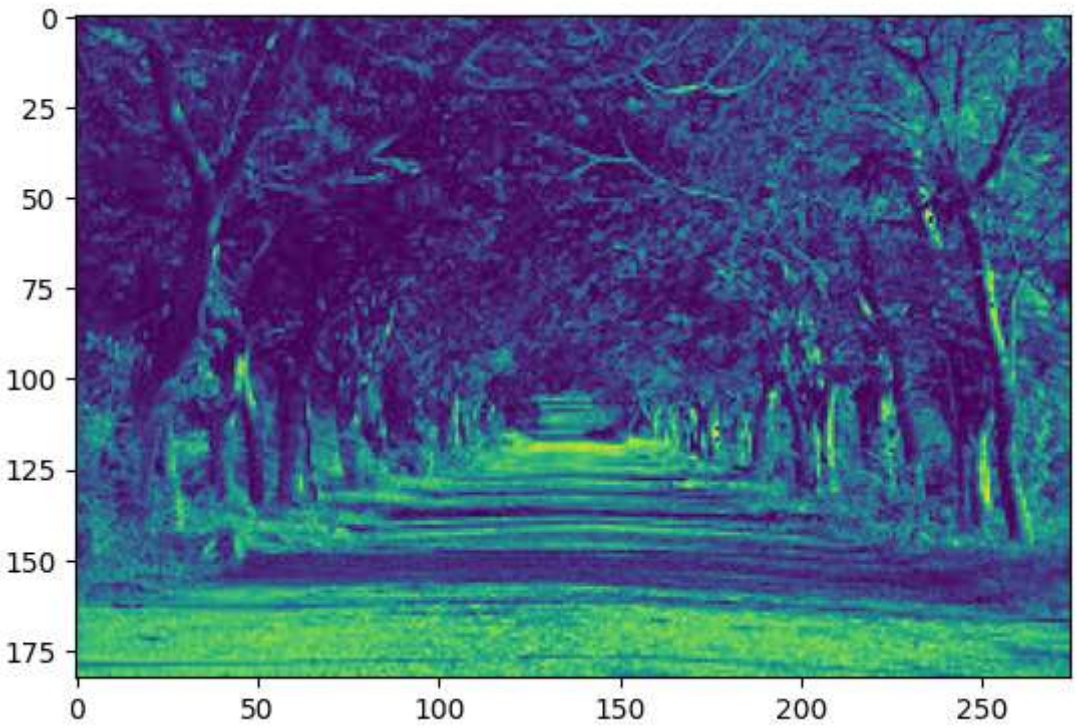
```
Text(0.5, 1.0, 'Red band')
```



creating a negative of the image

```
neg_img=np.abs(img_gray-255)
plt.imshow(neg_img)
```

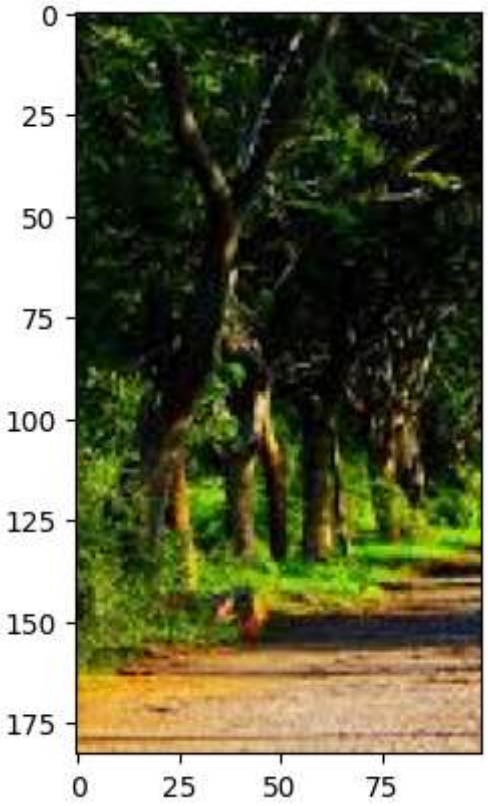
```
<matplotlib.image.AxesImage at 0x7824179be890>
```



Slicing and Dicing a image, Cropping an image

```
plt.imshow(img[:,0:100,:])
```

```
<matplotlib.image.AxesImage at 0x782415d9d5d0>
```



Generating the histogram for an image:

bins-> gives the number of bins expected and range -> gives the range of the bins

```
hist=np.histogram(img_gray,bins=256)
print(hist)
```

```
(array([2171,  353,  415,  497,  505,  502,  559,  561,  565,  589,  584,
        575,  580,  613,  590,  592,  556,  568,  557,  525,  546,  532,
        517,  462,  485,  496,  480,  517,  486,  451,  451,  433,  421,
        411,  416,  445,  402,  390,  397,  405,  362,  404,  365,  363,
        346,  351,  362,  340,  318,  296,  338,  330,  323,  302,  282,
        238,  282,  270,  251,  261,  263,  261,  256,  271,  258,  271,
        251,  249,  245,  228,  258,  207,  235,  228,  262,  188,  194,
        197,  222,  225,  216,  193,  206,  182,  189,  196,  186,  203,
        190,  210,  201,  207,  201,  190,  155,  194,  156,  180,  175,
        187,  178,  163,  151,  186,  150,  175,  167,  146,  175,  154,
        170,  136,  164,  151,  143,  162,  132,  140,  128,  146,  126,
        146,  141,  121,  134,  131,  132,  122,  134,  124,  117,  114,
        131,   98,  119,  108,  120,  109,  120,  102,  118,  110,   91,
        123,   92,  113,  111,   97,  116,  126,  121,  111,  136,  116,
        129,  127,  104,  104,  121,  125,  126,  131,  148,  140,  136,
        121,  109,  122,  132,  129,  132,  126,  159,  145,  139,  123,
        169,  138,  134,  159,  130,  139,  138,  124,  149,  136,  143,
        134,  143,  137,  128,  107,  114,   95,  107,   86,  102,  104,
         85,   79,   68,   70,   73,   48,   46,   48,   40,   46,   26,
         17,   32,   31,   21,   22,   22,   13,   13,   17,   11,    9,
          8,   10,    8,    7,    8,    7,    7,    6,    3,    4,    7,
          9,    3,    3,    3,    0,    3,    3,    2,    1,    1,    4,
          2,    1,    2,    1,    1,    0,    2,    0,    0,    2,    1,
          0,    1,    7]), array([  0.        ,   0.99609375,   1.9921875 ,   2.98
        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,
```

```
plt.hist(img.ravel(),bins=256,range=[0,256])# note the diff when range is given and range
# when range is given it take that as bins
# else it takes 0 to 0.999 as bins
```

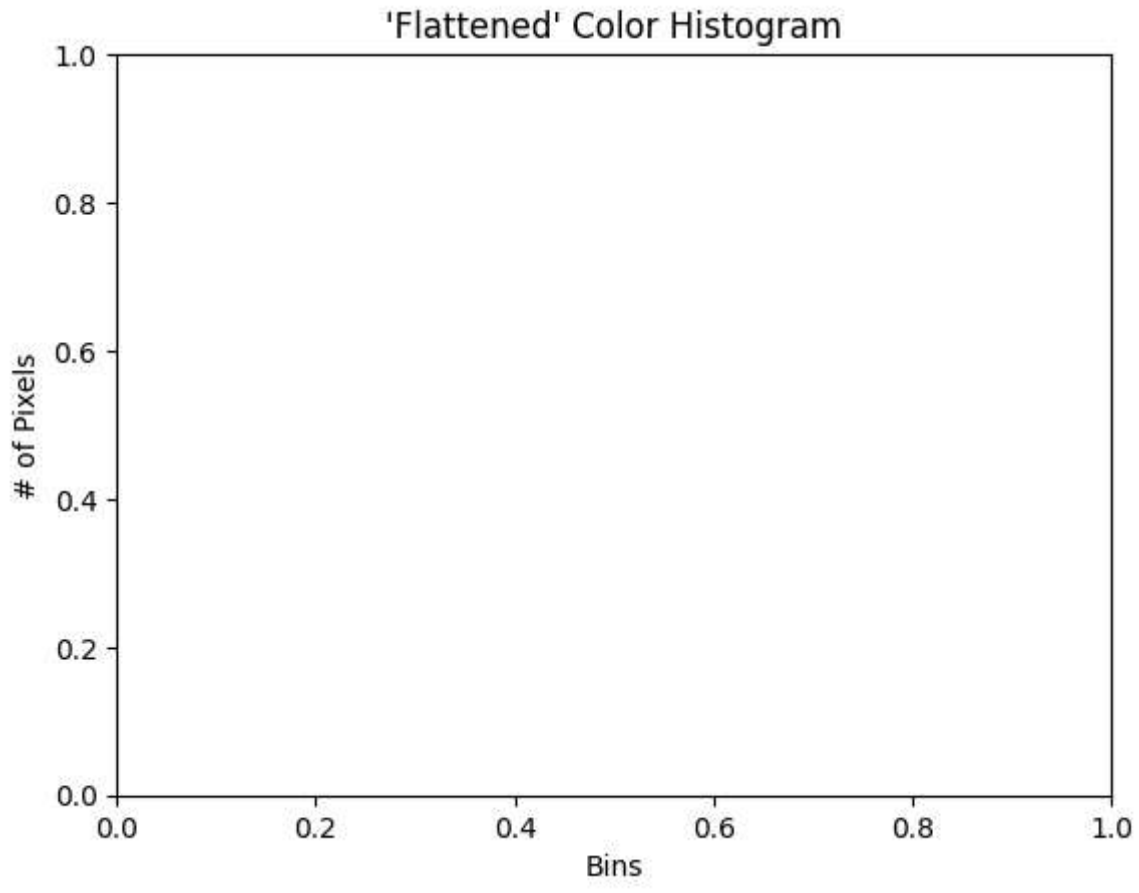
```
(array([1.7326e+04, 1.7390e+03, 1.7510e+03, 1.8440e+03, 1.8980e+03,
        1.9090e+03, 1.9730e+03, 1.9680e+03, 1.9410e+03, 1.8940e+03,
        1.9200e+03, 1.8710e+03, 1.8170e+03, 1.7580e+03, 1.7070e+03,
        1.7360e+03, 1.6770e+03, 1.6780e+03, 1.6150e+03, 1.5850e+03,
        1.5860e+03, 1.4850e+03, 1.4120e+03, 1.3970e+03, 1.4200e+03,
        1.3610e+03, 1.2940e+03, 1.3230e+03, 1.3710e+03, 1.2630e+03,
        1.2610e+03, 1.1670e+03, 1.2370e+03, 1.2300e+03, 1.1700e+03,
        1.2010e+03, 1.1390e+03, 1.0940e+03, 1.0980e+03, 1.0430e+03,
        9.9700e+02, 1.0180e+03, 1.0860e+03, 9.8400e+02, 9.7000e+02,
        9.1200e+02, 9.1200e+02, 8.5200e+02, 8.6800e+02, 8.6100e+02,
        8.4000e+02, 8.1000e+02, 8.2100e+02, 8.5100e+02, 8.0400e+02,
        7.8700e+02, 7.4700e+02, 7.7600e+02, 7.8200e+02, 7.3500e+02,
        6.7400e+02, 7.0000e+02, 7.2200e+02, 6.7900e+02, 6.9000e+02,
        6.4100e+02, 6.8700e+02, 6.4000e+02, 6.4600e+02, 6.2400e+02,
        5.9500e+02, 6.0000e+02, 5.8300e+02, 5.7200e+02, 5.7200e+02,
        5.2000e+02, 5.5200e+02, 5.6500e+02, 5.6200e+02, 5.1000e+02,
        5.2900e+02, 5.6100e+02, 5.2800e+02, 5.2400e+02, 5.4700e+02,
        4.7900e+02, 4.9300e+02, 4.7600e+02, 4.6700e+02, 5.0200e+02,
        4.7900e+02, 4.5900e+02, 4.7900e+02, 4.4000e+02, 4.3900e+02,
        4.6700e+02, 4.2700e+02, 4.3600e+02, 4.3500e+02, 3.7900e+02,
        4.6500e+02, 4.1700e+02, 4.2900e+02, 4.0000e+02, 4.2900e+02,
        4.0500e+02, 3.7900e+02, 4.3000e+02, 4.1500e+02, 4.1900e+02,
        4.0900e+02, 3.6500e+02, 3.7700e+02, 3.9100e+02, 3.4500e+02,
        3.8300e+02, 3.6500e+02, 3.7900e+02, 3.4300e+02, 3.9400e+02,
        3.3700e+02, 3.3600e+02, 3.7000e+02, 3.0200e+02, 3.4800e+02,
        3.5500e+02, 3.0600e+02, 3.2700e+02, 3.4400e+02, 3.1600e+02,
        2.8600e+02, 3.6600e+02, 3.1600e+02, 3.4700e+02, 3.2600e+02,
        2.8500e+02, 3.3800e+02, 2.7500e+02, 3.0300e+02, 3.0600e+02,
        2.7600e+02, 3.2900e+02, 3.1100e+02, 3.0600e+02, 3.3000e+02,
        2.9100e+02, 3.3900e+02, 2.6800e+02, 2.8300e+02, 2.9500e+02,
        2.7000e+02, 2.6900e+02, 2.8500e+02, 2.9300e+02, 2.7500e+02,
        2.4700e+02, 2.6100e+02, 2.1400e+02, 2.4700e+02, 2.4600e+02,
        2.4900e+02, 2.7200e+02, 2.5200e+02, 2.5100e+02, 2.5600e+02,
        2.5000e+02, 2.4000e+02, 2.3200e+02, 2.2300e+02, 2.4100e+02,
        2.6000e+02, 2.3700e+02, 2.2200e+02, 2.5200e+02, 2.4500e+02,
        2.1400e+02, 2.4700e+02, 2.1400e+02, 1.9800e+02, 2.6900e+02,
        2.4100e+02, 2.4300e+02, 2.3300e+02, 2.3200e+02, 1.7400e+02,
        2.2200e+02, 2.3000e+02, 2.1200e+02, 2.1400e+02, 2.1200e+02,
        1.8700e+02, 1.6400e+02, 1.8100e+02, 1.8700e+02, 1.9400e+02,
        1.7800e+02, 1.6000e+02, 1.7800e+02, 1.5600e+02, 1.5100e+02,
        1.4500e+02, 1.4100e+02, 1.4300e+02, 1.5000e+02, 1.4200e+02,
        1.5500e+02, 1.6600e+02, 1.6000e+02, 1.6000e+02, 1.3800e+02,
        1.2500e+02, 1.3200e+02, 1.4400e+02, 1.5400e+02, 1.4200e+02,
        1.4900e+02, 1.4600e+02, 1.4400e+02, 1.5500e+02, 1.3800e+02,
        1.5600e+02, 1.5200e+02, 1.6200e+02, 1.3300e+02, 1.1600e+02,
        1.3700e+02, 1.2800e+02, 1.1900e+02, 1.1400e+02, 1.1700e+02,
        1.1600e+02, 1.2600e+02, 1.1900e+02, 9.2000e+01, 8.9000e+01,
        8.1000e+01, 7.0000e+01, 7.8000e+01, 6.9000e+01, 6.7000e+01,
        4.8000e+01, 5.5000e+01, 4.2000e+01, 3.5000e+01, 3.1000e+01,
        3.7000e+01, 3.6000e+01, 2.2000e+01, 2.5000e+01, 2.9000e+01,
        1.8000e+01, 2.1000e+01, 1.5000e+01, 1.3000e+01, 1.2000e+01,
        7.8000e+01]),
 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.,
```

Color Histogram

cv2.calcHist(images, channels, mask, histSize, ranges)

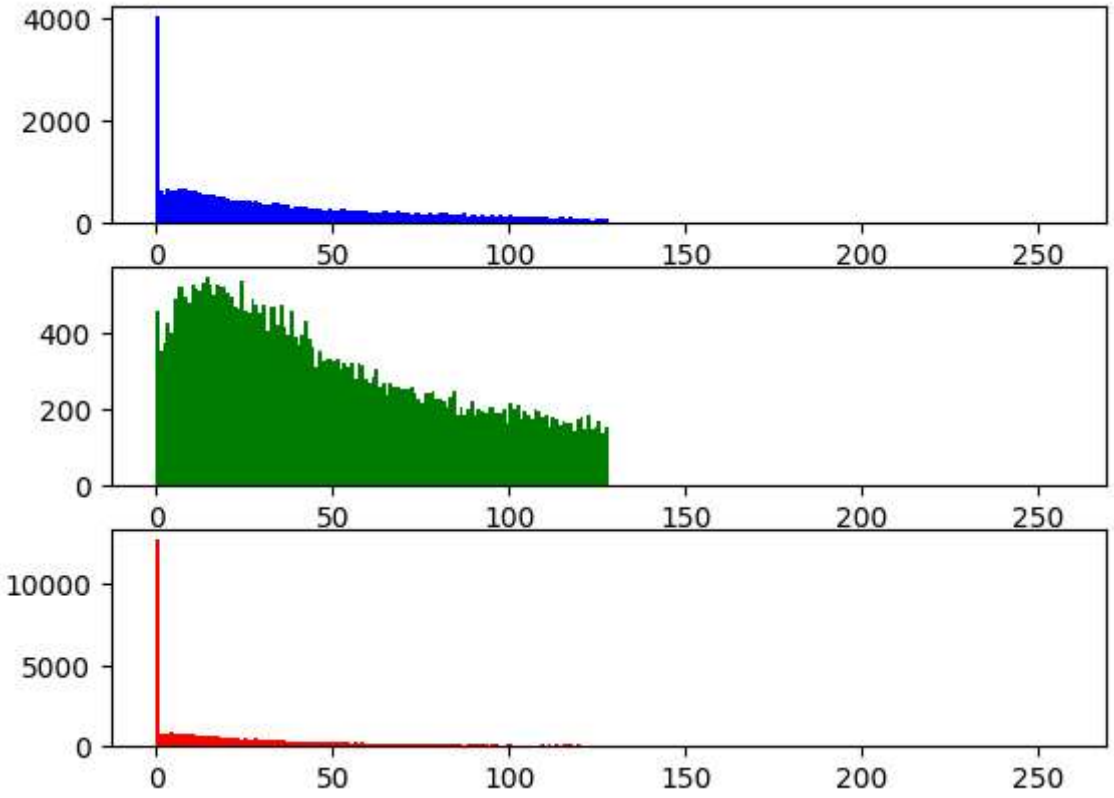
```
import cv2 as cv
chans = cv.split(img)
colors = ("b", "g", "r")
plt.figure()
plt.title("'Flattened' Color Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
# loop over the image channels
for (chan, color) in zip(chans, colors):
    # create a histogram for the current channel and plot it
    #hist = cv.calcHist([chan], [0], None, [256], [0, 256])
    plt.hist(chans[:,:,0], color=color)
    plt.xlim([0, 256])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-9761ce9e745c> in <cell line: 9>()
     10         # create a histogram for the current channel and plot it
     11         #hist = cv.calcHist([chan], [0], None, [256], [0, 256])
---> 12         plt.hist(chans[:,:,0], color=color)
     13         plt.xlim([0, 256])

TypeError: tuple indices must be integers or slices, not tuple
```



```
print(img.shape)
print(img.dtype)
img=img.astype('int8')
# plot the histogram
plt.subplot(3,1,1), plt.hist(img[:,:,0].ravel(), bins=256,range=[0,256], color='b')
plt.subplot(3,1,2), plt.hist(img[:,:,1].ravel(), bins=256, range=[0,256],color='g')
plt.subplot(3,1,3), plt.hist(img[:,:,2].ravel(), bins=256, range=[0,256], color='r')
plt.show()
```

```
(183, 275, 3)
int64
```



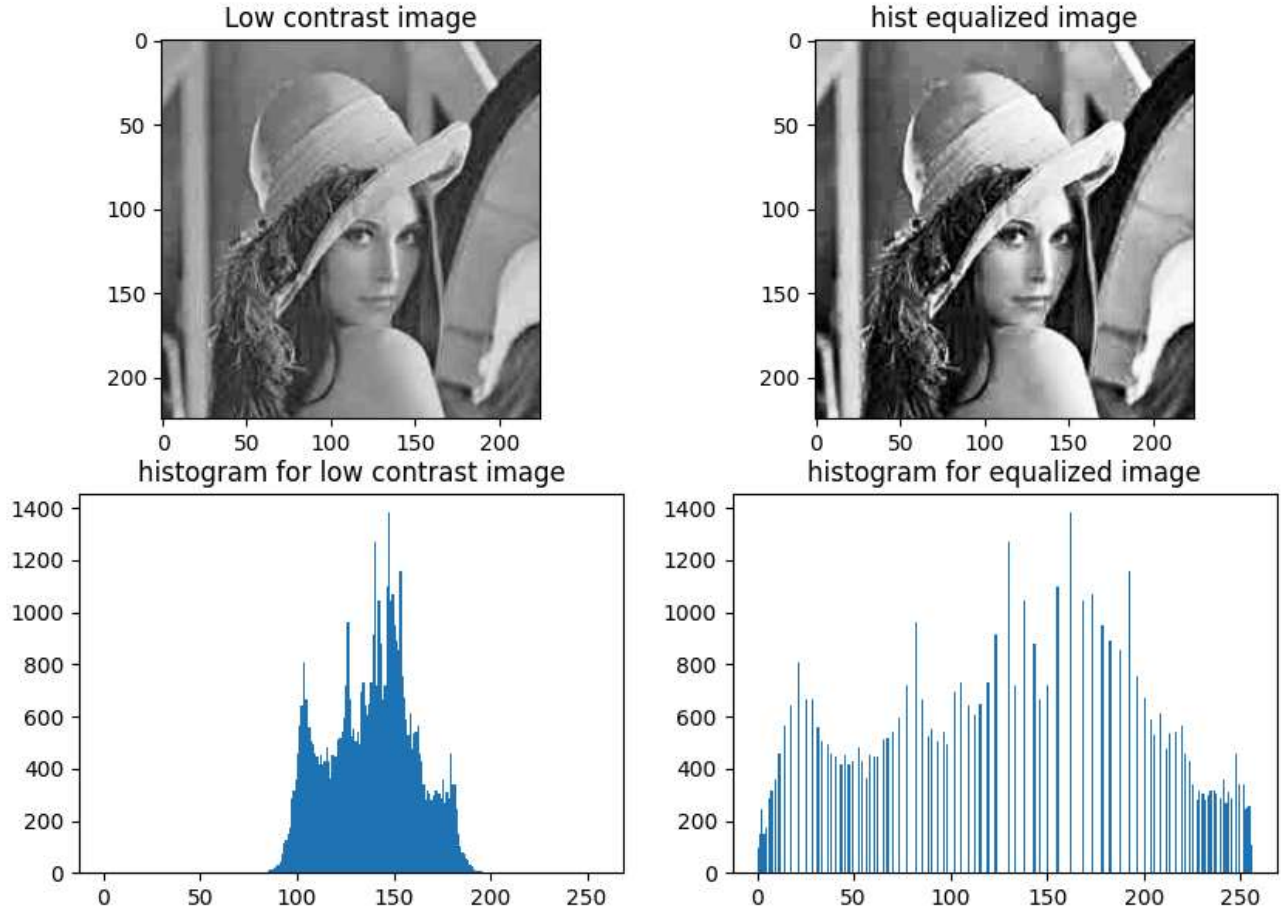
Histogram Equalization

Reference:

1. https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html
2. https://medium.com/jungletronics/histogram-equalization-34149fc299a6
3. https://pythongeeks.org/histogram-equalization-in-opencv/

```
lena_low=cv.imread('lena_low.jpg',0)
print(lena_low.shape)
lena_hist_eq=cv.equalizeHist(lena_low)
#lena_hist_eq = np.hstack((lena_low, hist_eq)) # give this to combine the low and high c
fig=plt.figure(figsize=(10,7))

fig.add_subplot(2,2,1)
plt.imshow(lena_low,cmap='gray')
plt.title("Low contrast image")

fig.add_subplot(2,2,2)
plt.imshow(lena_hist_eq,cmap='gray')
plt.title("hist equalized image")

fig.add_subplot(2,2,3)
plt.hist(lena_low.ravel(),bins=256,range=[0,256])
plt.title("histogram for low contrast image")

fig.add_subplot(2,2,4)
plt.hist(lena_hist_eq.ravel(),bins=256,range=[0,256])
plt.title("histogram for equalized image")
```

```
(225, 225)
Text(0.5, 1.0, 'histogram for equalized image')
```



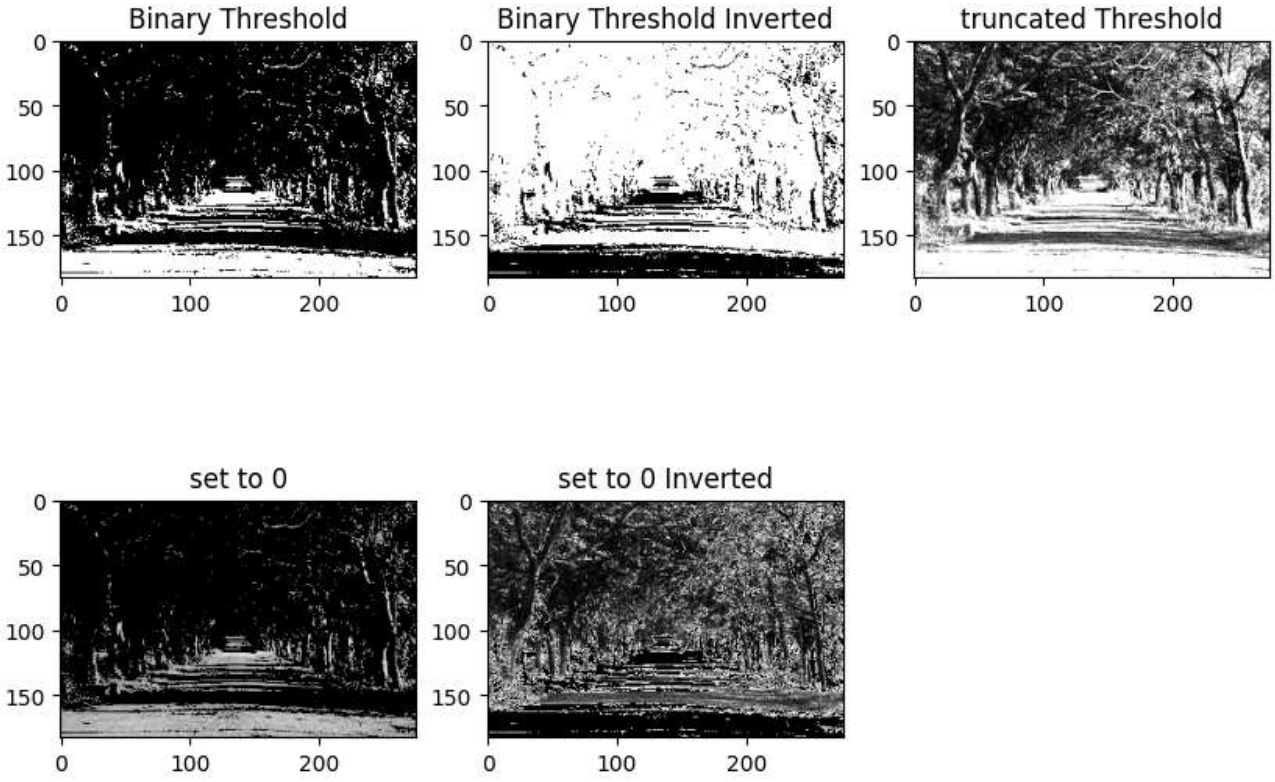
Thresholding

refernce:

1. https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/
2. https://learnopencv.com/opencv-threshold-python-cpp/

```
ret, thresh1 = cv2.threshold(img_gray, 120, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img_gray, 120, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img_gray, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img_gray, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img_gray, 120, 255, cv2.THRESH_TOZERO_INV)

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

fig.add_subplot(2,3,1)
plt.imshow(thresh1,cmap='gray')
plt.title('Binary Threshold')

fig.add_subplot(2,3,2)
plt.imshow(thresh2,cmap='gray')
plt.title("Binary Threshold Inverted")

fig.add_subplot(2,3,3)
plt.imshow(thresh3,cmap='gray')
plt.title("truncated Threshold")

fig.add_subplot(2,3,4)
plt.imshow(thresh4,cmap='gray')
plt.title("set to 0")

fig.add_subplot(2,3,5)
plt.imshow(thresh5,cmap='gray')
plt.title("set to 0 Inverted")
```

```
Text(0.5, 1.0, 'set to 0 Inverted')
```



Practice question

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

Edge detection

These are the kernels used for Sobel Edge Detection:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{1}$$

X-Direction Kernel

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2}$$

Y-Direction Kernel

When these kernels are convolved with the original image, you get a 'Sobel edge image'.

- If we use only the Vertical Kernel, the convolution yields a Sobel image, with edges enhanced in the X-direction
- Using the Horizontal Kernel yields a Sobel image, with edges enhanced in the Y-direction.

Let G_x and G_y represent the intensity gradient in the x and y directions respectively. If A and B denote the X and Y kernels defined above:

$$G_x = A * I \tag{3}$$

$$G_y = B * I \tag{4}$$

where $*$ denotes the convolution operator, and I represents the input image.

The final approximation of the gradient magnitude, G can be computed as:

$$G = \sqrt{G_x^2 * G_y^2} \tag{5}$$

And the orientation of the gradient can then be approximated as:

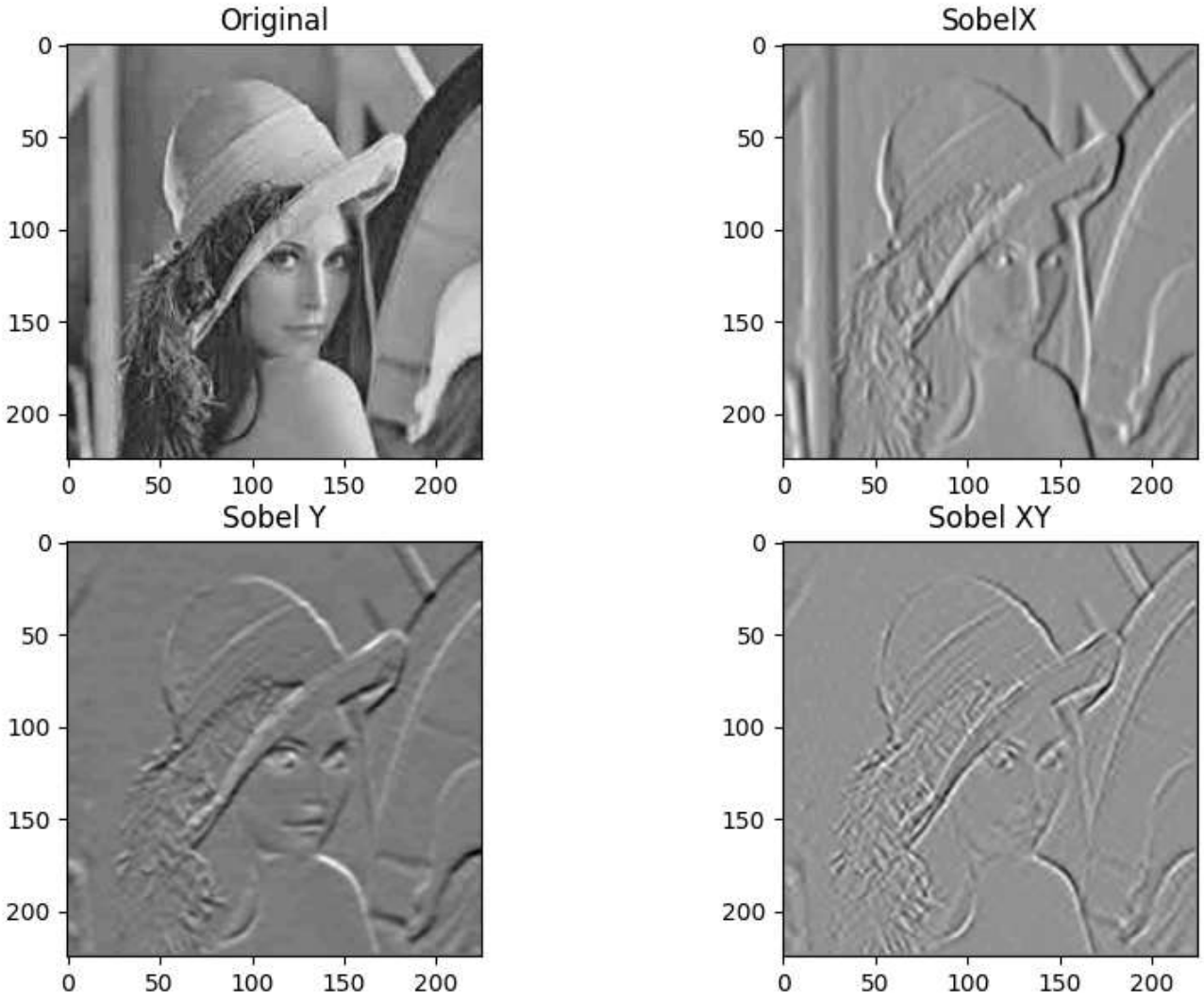
$$\Theta = arctan(G_y/G_x) \tag{6}$$

Refrence:

1. https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html#demo
2. https://learnopencv.com/edge-detection-using-opencv/
3. https://www.geeksforgeeks.org/python-opencv-smoothing-and-blurring/?ref=previous_article
4. https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1

```
lena_gray=cv.imread('lena_low.jpg',0)
print(lena_low.shape)

img_blur = cv.GaussianBlur(lena_gray, (3,3), 0)

# Sobel Edge Detection
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5) # Sobel Edge De
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5) # Sobel Edge De
sobelxy = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5) # Combined X a

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

fig.add_subplot(2,2,1)
plt.imshow(lena_gray,cmap='gray')
plt.title('Original')

fig.add_subplot(2,2,2)
plt.imshow(sobelx,cmap='gray')
plt.title("SobelX")

fig.add_subplot(2,2,3)
plt.imshow(sobely,cmap='gray')
plt.title("Sobel Y")

fig.add_subplot(2,2,4)
plt.imshow(sobelxy,cmap='gray')
plt.title("Sobel XY")
```

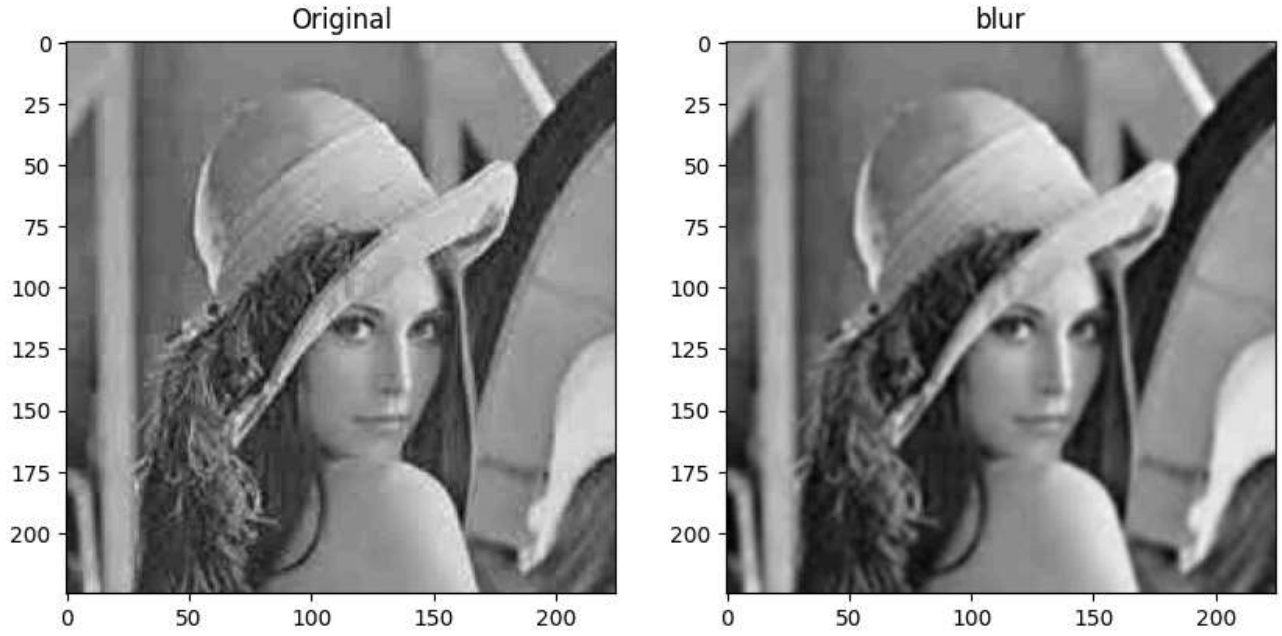
```
(225, 225)
Text(0.5, 1.0, 'Sobel XY')
```



Using custom kernel

```
blur_filter = np.array([[ 0.111, 0.111, 0.111],
                        [ 0.111, 0.111, 0.111],
                        [ 0.111, 0.111, 0.111]])

# Filter the image using filter2D, which has inputs: (grayscale image, bit-depth, kernel
filtered_image_blur = cv2.filter2D(lena_gray, -1, blur_filter)

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

fig.add_subplot(1,2,1)
plt.imshow(lena_gray,cmap='gray')
plt.title('Original')

fig.add_subplot(1,2,2)
plt.imshow(filtered_image_blur,cmap='gray')
plt.title("blur")
```

```
Text(0.5, 1.0, 'blur')
```



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

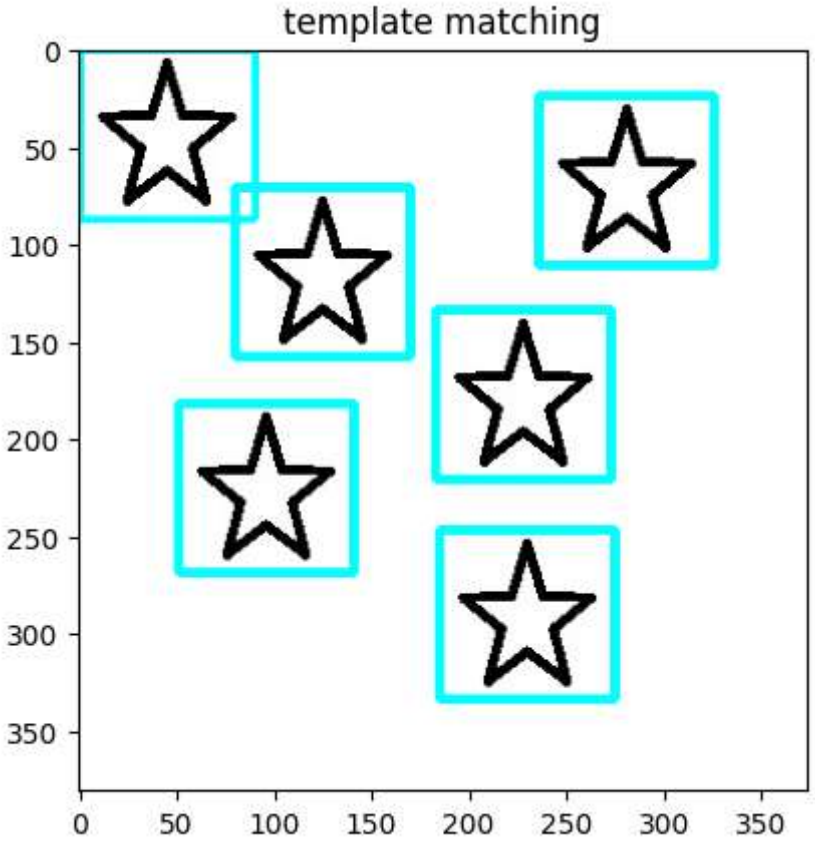
Template Matching

Reference:

1. https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/
2. https://pyimagesearch.com/2021/03/22/opencv-template-matching-cv2-matchtemplate/

```
img_star = cv.imread('star.jpg')
print(img_star.shape)
#plt.imshow(img_star)

# Convert it to grayscale
img_star_gray = cv2.cvtColor(img_star, cv2.COLOR_BGR2GRAY)
print(img_star_gray.shape)
#plt.imshow(img_star_gray)

# Read the template
template = cv2.imread('star_template.jpg')
template=cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
print(template.shape)
# plt.imshow(template)

# Store width and height of template in w and h
w, h = template.shape #(::-1)
print(w,h)

# Perform match operations.
res = cv2.matchTemplate(img_star_gray, template, cv2.TM_CCOEFF_NORMED)

# Specify a threshold
threshold = 0.8

# Store the coordinates of matched area in a numpy array
loc = np.where(res >= threshold)

# Draw a rectangle around the matched region.
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_star, pt, (pt[0] + w, pt[1] + h), (0, 255, 255), 2)

#print(loc)
# Show the final image with the matched area.
plt.imshow(img_star)
plt.title("template matching")
```

```
(381, 374, 3)
(381, 374)
(90, 87)
90 87
Text(0.5, 1.0, 'template matching')
```



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.)

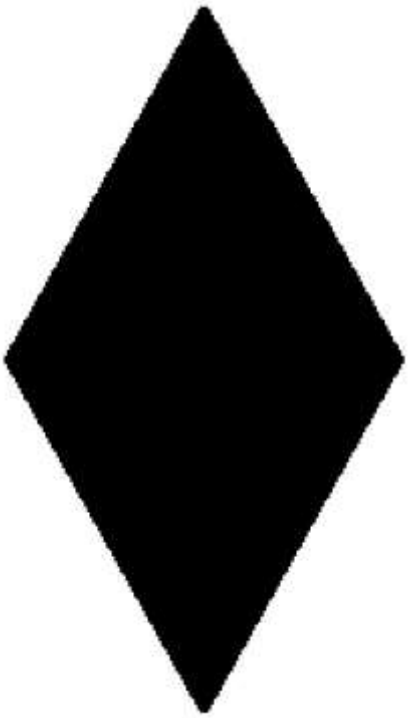
HOG Descriptor

Reference:

1. https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/
2. https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f
3. https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa

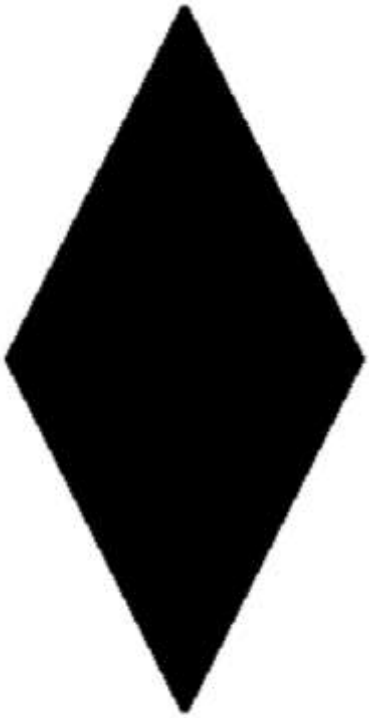
```
from skimage.io import imread
from skimage.transform import resize
from skimage.feature import hog
from skimage import exposure
import matplotlib.pyplot as plt

img_dia = imread('diamond.png')
img_dia=img_dia[100:370,50:200,:]
plt.axis("off")
plt.imshow(img_dia)
print(img_dia.shape)
```

```
(270, 150, 3)
```



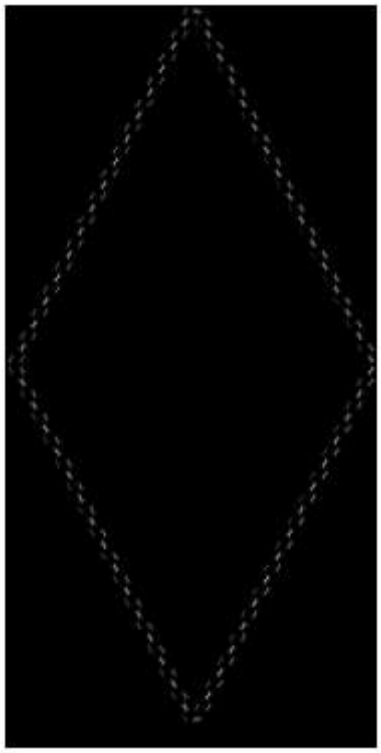
```
resized_img = resize(img_dia, (128*4, 64*4))
plt.axis("off")
plt.imshow(resized_img)
print(resized_img.shape)
```

```
(512, 256, 3)
```



```
fd, hog_image = hog(resized_img, orientations=9, pixels_per_cell=(8,
8),
cells_per_block=(2, 2), visualize=True,
multichannel=True)
plt.axis("off")
plt.imshow(hog_image, cmap="gray")
plt.show()
```

```
<ipython-input-117-06acf1ec875b>:1: FutureWarning: `multichannel` is a deprecated ar
  fd, hog_image = hog(resized_img, orientations=9, pixels_per_cell=(8,
```



```
print(fd.shape)
```

```
(70308,)
```

Calculation of hog features for an image of size 128, 64

cell =8 * 8
along horizontal =128/8=16
no of movements along horizontal direction=16-1 =15
along vertical =64/8=8
no of movements along vertical direction=8-1=7
no of blocks = 7 * 15 = 105
no of bins for 8 * 8 =9
no of features for 8 * 8 =9
block = 4 * cell
no features per block= 4 * no of features per cell
=4 * 9=36
total number of features= no of blocks * number of features per block
= 105 * 36= 3780

Calculation of hog features for an image of size 512, 256

along horizontal =512/8=64

no of movements along horizontal direction=64-1 =63

along vertical =256/8= 32

no of movements along vertical direction=32-1=31

no of blocks = 63 * 31 = 1953

no of bins for 8 * 8 =9

no of features for 8 * 8 =9

block = 4*cell*

no features per block= 4 \ no of features per cell

=4 * 9=36

total number of features= no of blocks* number of features per block

= 1953*36= 70308

Practise Question:

1. generate HOG descriptor for camera man Image and astronout image
2. what will be the size of feature vector for 512 * 1024 imagefeature size

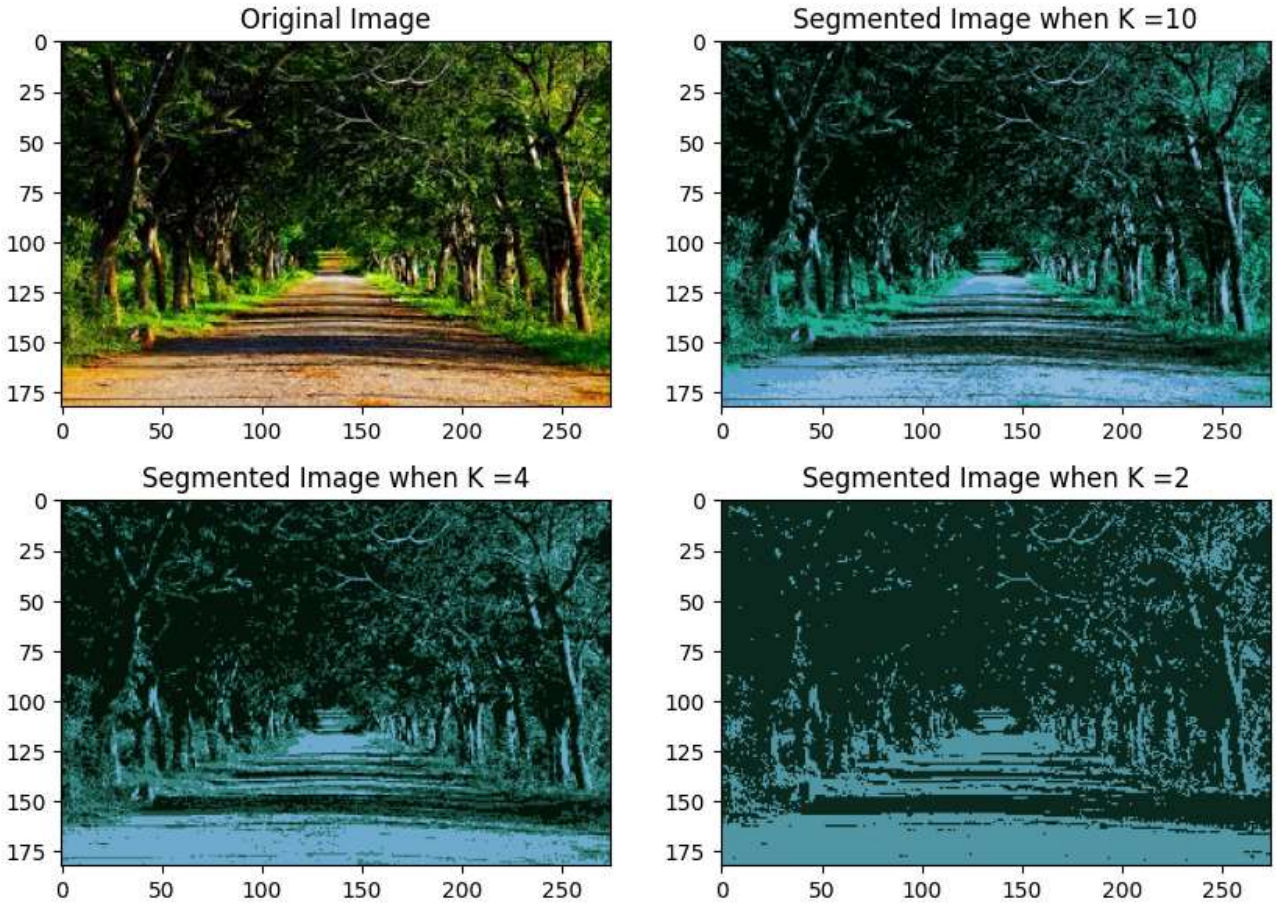
Kmeans Clustering

Reference:

1. https://www.simplilearn.com/tutorials/machine-learning-tutorial/k-means-clustering-algorithm
2. https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/
3. https://www.geeksforgeeks.org/k-means-clustering-introduction/
4. https://www.kdnuggets.com/2019/08/introduction-image-segmentation-k-means-clustering.html

```
img = cv.imread("nature.jpg")
img_rgb = cv.cvtColor(img, cv2.COLOR_BGR2RGB) #defining image to experiment with number c
print(img.shape)
vectorized = img.reshape((-1,3))
vectorized = np.float32(vectorized)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 10
attempts=10
res,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))
#image 3
K = 4
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_image1 = res.reshape((img.shape))
#image 4
K = 2
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_image2 = res.reshape((img.shape))


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

fig.add_subplot(2,2,1)
plt.imshow(img_rgb)
plt.title('Original Image')

fig.add_subplot(2,2,2)
plt.imshow(result_image)
plt.title('Segmented Image when K =10')

fig.add_subplot(2,2,3)
plt.imshow(result_image1,cmap='gray')
plt.title('Segmented Image when K =4')

fig.add_subplot(2,2,4)
plt.imshow(result_image2)
plt.title('Segmented Image when K =2')
```

```
(183, 275, 3)
Text(0.5, 1.0, 'Segmented Image when K =2')
```



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

Image Opeartion

1. Down sampling an image
2. Upsampling an image
3. Scaling an image
4. Rotating an image

Reference:

1. https://medium.com/analytics-vidhya/rotating-images-with-opencv-and-imutils-99801cb4e03e
2. https://learnopencv.com/image-resizing-with-opencv/

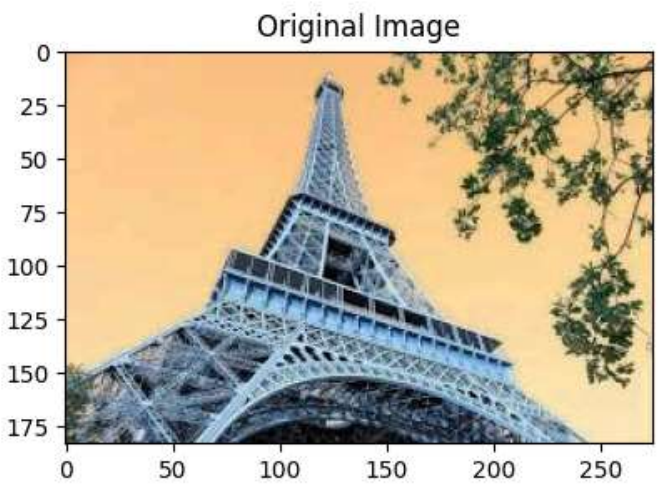
down sampling an image

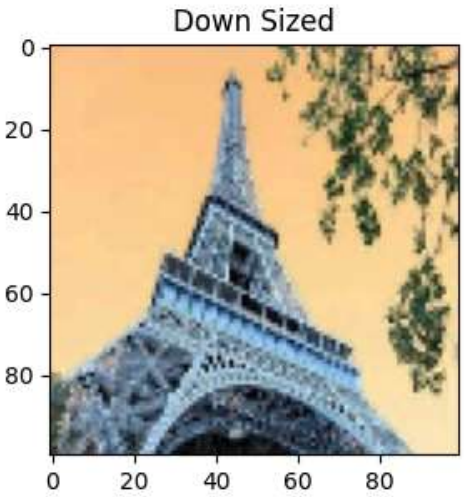
```
img = cv2.imread('eiffel2.jpg')
print(img.shape)
img1=cv2.pyrDown(img)
print(img1.shape)
img2=cv2.pyrDown(img1)
print(img2.shape)
```

```
(183, 275, 3)
(92, 138, 3)
(46, 69, 3)
```

```
down_width = 100
down_height = 100
down_points = (down_width, down_height)
resize_down = cv2.resize(img, down_points, interpolation= cv2.INTER_LINEAR)

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

fig.add_subplot(2,2,1)
plt.imshow(img)
plt.title('Original Image')

fig.add_subplot(2,2,2)
plt.imshow(resize_down)
plt.title('Down Sized')
```

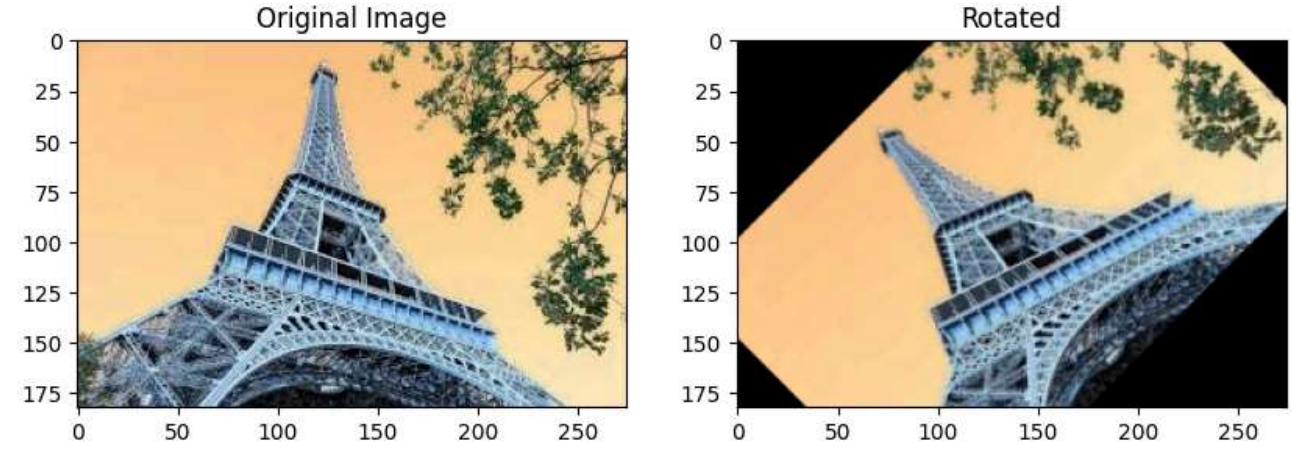
```
Text(0.5, 1.0, 'Down Sized')
```





```
img = cv2.imread('eiffel2.jpg')
height, width = img.shape[:2]
centerX, centerY = (width // 2, height // 2)
M = cv2.getRotationMatrix2D((centerX, centerY), 45, 1.0)
rotated = cv2.warpAffine(img, M, (width, height))

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

fig.add_subplot(2,2,1)
plt.imshow(img)
plt.title('Original Image')

fig.add_subplot(2,2,2)
plt.imshow(rotated)
plt.title('Rotated')
```

```
Text(0.5, 1.0, 'Rotated')
```



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

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
from skimage.data import astronaut
from skimage.color import rgb2gray
from skimage.filters import sobel
```

```
#call SLIC, with segment compactness = 10
segments_slic = slic(img, n_segments=250, compactness=10, sigma=1)
gradient = sobel(rgb2gray(img))
#call watershed with segment compactness = 0.1
segments_watershed = watershed(gradient, markers=250, compactness=0.01)
#count the number of segments
print('SLIC number of segments:
{}'.format(len(np.unique(segments_slic))))
print('watershed number of segments:
{}'.format(len(np.unique(segments_watershed))))
fig, ax = plt.subplots(2, 2, figsize=(10, 10), sharex=True,
sharey=True)
ax[0, 0].imshow(mark_boundaries(img, segments_slic))
ax[0, 0].set_title('SLIC')
ax[0, 1].imshow(mark_boundaries(img, segments_watershed))
ax[0, 1].set_title('Compact watershed')
for a in ax.ravel():
a.set_axis_off()
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
SyntaxError                               Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/IPython/core/compilerop.py in
ast_parse(self, source, filename, symbol)
     99         Arguments are exactly the same as ast.parse (in the standard
library),
```