1hist

```
#@title 1hist
#matplotlib
plt.hist(img.ravel(), 256, [0, 256])
#256 - max number of pixel
#0-255 - range of the pixel
plt.show()

# 2) How to add white pixel to the image
"""import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = np.zeros((200, 200), np.uint8)
cv.rectangle(img, (0,100), (200,200), (255), -1)
#(0,100), (200,200) are the demension of image
# 255 pixel value for the created rectangle
#-1 thickness to fill the rectangle
#cv.imshow("img", img)
plt.hist(img.ravel(), 256, [0,256])
plt.show()"""
# let us add some more pixels to the image
"""import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = np.zeros((200, 200), np.uint8)
cv.rectangle(img, (0,100), (200,200), (255), -1)
cv.rectangle(img, (0,50), (100,100), (127), -1)
#15000 pixels are black, 20000 pixels are white and 5000 pixels are gray
plt.hist(img.ravel(), 256, [0,256])
plt.show()"""


#let us try to use image for hist
"""import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('grey.tif', 1)
plt.hist(img.ravel(), 256, [0,256])
#plt.imshow(img)
plt.show()"""

#How to find the pixel intensity of different colors.
"""import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('Colorimage.jpg', 1)
b,g,r=cv.split(img)
cv.imshow("img", img)
cv.imshow("img", b)
cv.imshow("img", g)
cv.imshow("img", r)
plt.hist(b.ravel(), 256, [0,256])
plt.hist(g.ravel(), 256, [0,256])
plt.hist(r.ravel(), 256, [0,256])
plt.show()"""
```

```
#Histogram plot using cv2\
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('grey.tif')
hist = cv.calcHist([img], [0], None, [10], [0,256])
plt.plot(hist)
```

Affine transform

```
#@title Affine transform
img =cv2.imread('Colorimage.jpg')
rows, columns, ch = img.shape  #function shape returns the shape of the i/p image
pt1 = np.float32([[50,50], [200,50], [50, 200]])
#pt(point) is defined in the original image, this point should not be co-linear
pt2 = np.float32([[10,100], [200,50], [100, 250]])
matrix = cv2.getAffineTransform(pt1, pt2) #Transform function to create a matrix
new_img = cv2.warpAffine(img, matrix, (columns, rows))
```

distance

```
#@title distance

image = np.zeros((36,36,3), np.uint8)

#selection of red pixel location
image[10,15] = [255,0,0] # 0 - B, 0 -G, 255 - Red

#Choose green pixel location

image[21,23] = [0, 255, 0]

cv2.imwrite('created_image.jpeg', image)
plt.imshow(image)

#To find the distance between the pixels
x1, y1, x2, y2 = 10, 15, 21, 23
distance = math.sqrt((x2-x1)**2 -(y2-y1)**2)
print(distance)
```

filter all

```
#@title filter all

img = cv2.imread('salt_pepper.png')
#img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # converting image from BGR to RGB

#opencv read the image in BGR format

#define the kernel with 5 by 5
kernel=np.ones((5,5), np.float32)/25

out_img=cv2.filter2D(img, -1, kernel)
blur=cv2.blur(img, (5,5))
```

```
gblur=cv2.GaussianBlur(img, (5,5), 0)

median =cv2.medianBlur(img, 5)

bilateralfilter = cv2.bilateralFilter(img, 9, 75, 75)
#img - input image
#-1 desired depth of output image
#Apply kernel on the image using filter 2D
"""titles = ['image', '2Dconvolution']
images=[img, out_img]"""


titles = ['image', '2Dconvolution', 'blur', 'gblur', 'median', 'bilateralfilter']
images=[img, out_img, blur, gblur, median, bilateralfilter]

for i in range(6):
    plt.subplot(2, 3, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])

plt.show()
```

gamma transform

```
#@title gamma transform
img = cv2.imread('grey.tif', 1) # 0 reads as a grey scale image
gamma = 2   # gamma transform
img_gamma1 = np.power(img, gamma)  #raising the image by gamma(2)

gamma = 10
img_gamma2 = np.power(img, gamma)


gamma = 20
img_gamma3 = np.power(img, gamma)

plt.subplot(221), plt.imshow(img), plt.title('input image')
plt.subplot(222), plt.imshow(img_gamma1), plt.title('gamma=2')
plt.subplot(223), plt.imshow(img_gamma2), plt.title('gamma=10')
plt.subplot(224), plt.imshow(img_gamma3), plt.title('gamma=20')
```

histogram equ

```
#@title histogram equ

import cv2
from matplotlib import pyplot as plt
img =cv2.imread('Colorimage.jpg')

#converting image to LAB color
lab_image=cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
l, a, b =cv2.split(lab_image)

#histogram of l component
plt.hist(l.flat, bins=100, range=(0,255))
plt.show()
```

```
#Apply histogram equalization on L channel
equ = cv2.equalizeHist(l)
plt.hist(equ.flat, bins=100, range=(0,255))
plt.show()

#combine the hist equalized l channel back to A and B channel
updated_lab_img1 = cv2.merge((equ, a, b))

#Convert the LAB image back to color(RGB)
hist_eq_image =cv2.cvtColor(updated_lab_img1, cv2.COLOR_LAB2BGR)


#Apply CLAHE to L channel
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
clahe_img = clahe.apply(l)

#PLot the histogram
plt.hist(clahe_img.flat, bins=100, range=(10, 255))

#combine the CLAHE enhanced L-channel back to A and B
updated_lab_img2 = cv2.merge((clahe_img, a, b))

#Convert the LAB image back to color(RGB)

CLAHE_img = cv2.cvtColor(updated_lab_img2, cv2.COLOR_LAB2RGB)
"""plt.show('original image', img)
plt.show('equalized image', hist_eq_image)
plt.show('CLAHE IMAGE', CLAHE_img)"""


cv2.imshow('original image', img)
cv2.imshow('equalized image', hist_eq_image)
cv2.imshow('CLAHE IMAGE', CLAHE_img)
cv2.waitKey(0)
```

inverse log

```
#@title inverse log

"""from matplotlib import pyplot as plt
import numpy as np
from matplotlib.colors import NoNorm
r = np.arange(0, 255)
c=255/(np.log(1+255))
y=np.exp(r)**(1/c)-1
plt.plot(r,y)"""

#Apply Inverse log transform on image

from skimage import io
from matplotlib import pyplot as plt
import numpy as np
from matplotlib.colors import NoNorm
img = io.imread('lighterimage.tif')
c=255/np.log(1+255)
inverse_log_image = np.exp(img**1/c)-1
inverse_log_image = np.array(inverse_log_image, dtype=np.uint8) #float is converted to int8
```

```
plt.figure(figsize=(8,4))
plt.subplot(1,3,1)
plt.imshow(img, cmap='gray', norm=NoNorm())
plt.subplot(1, 3, 2)
plt.imshow(inverse_log_image, cmap='gray', norm=NoNorm())
#plt.imshow(log_image, cmap='gray')
plt.show()
```

open close dilate

```
#@title open close dilate
img = cv2.imread('morpho.tif', cv2.IMREAD_GRAYSCALE)

_, mask =cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)

# 127 - VALUE OF THRESHOLD

kernel = np.ones((5,5), np.uint8)
#dilation =cv2.dilate(mask, kernel)
dilation =cv2.dilate(mask, kernel, iterations=1)
#In the dilation process black pixel were reduced, or white pixel are incresed.

# To remove the black pixel completely, i apply iteration, apply kernel for any number of iteration.

#bigger the kernel size, black pixel are turned to white, by dilation but size of the white object i

#Dilation -  if atleast one pixel matches with kernel, than change the pixel to 1

#Therefore the shape of white object is increasing.

erosion = cv2.erode(mask, kernel, iterations=1)

# the shape of the white object is reduced.  erosion - all pixels of image need to be match with ker


opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

#erosion followed by dilation
closing = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

#dilation followed by erosion

#change the kernel size and iteration number and see the difference.

gradient = cv2.morphologyEx(mask, cv2.MORPH_GRADIENT, kernel)
#difference between the dilation and erosion

tophat=cv2.morphologyEx(mask, cv2.MORPH_TOPHAT, kernel)

#diff b/w the input image and opening image.

titles = ['input image', 'mask', 'dilation', 'erosion', 'opening', 'gradient', 'tophat']
images = [img, mask, dilation, erosion, opening, gradient, tophat]
```

tranform

```
#@title tranform
```

```
img_vh=cv2.flip(img, -1) #refleection

#rotation
img=cv2.imread('grey.tif', 0)
plt.imshow(img)
rows, cols = img.shape

matrix = cv2.getRotationMatrix2D((rows/2, cols/2), 60, 1)
#matrix =cv2.getRotationMatrix2D((rows/2, cols/2), 60, 2.5)
new_img = cv2.warpAffine(img, matrix, (500, 320))
new_img = cv2.warpAffine(img, matrix, (rows, cols))

#translation
img = cv2.imread('color.jpg')
plt.imshow(img)

height, width = img.shape[:2]
print(height, width)

height_fourth, width_fourth = height/4, width/4

print(height_fourth, width_fourth)

#Translate matrix
t = np.float32([[1,0,height_fourth], [0,1, width_fourth]])

print(t)
translation=cv2.warpAffine(img, t, (width, height))
plt.imshow(translation)
```