```
# Numpy and Images with pil WO OCV
import numpy as np
import matplotlib.pyplot as plt
get_ipython().magic('matplotlib inline')
from PIL import Image
import os
print(os.getcwd())
os.chdir('E:\\Locker\\Sai\\SaiHCourseNait\\DecBtch\\R_Datasets\\')
print(os.getcwd())
import warnings
warnings.filterwarnings('ignore')

pic = Image.open("00-puppy.jpg")
pic

print(type(pic))

pic_arr = np.asarray(pic)
print(pic_arr)

print(type(pic_arr))

print(pic_arr.shape)

plt.imshow(pic_arr)
```

```
#ntc the coords of X-axis & Y-axis

# There r 3 channels inside this


pic_red = pic_arr.copy()


plt.imshow(pic_red)

plt.show()


pic_red.shape

# This has 3 chnnls

# R G B


pic_red[:,:,0]

# This will access the Red channel

# ntc the vals r not btw 0 & 255

# everything in 1300, everything in 1950 & 0 whi maps to R in  R G B


plt.imshow(pic_red[:,:,0])

# This is internally virdis color scale by matplot


plt.imshow(pic_red[:,:,0],cmap='gray')

# BW, cmap is colormap, gray scale

# 0 mean no Red, pure black

# ntc the color of the ears of the dog, no Red, they r light

# Red channel values r btw 0-255
```

```
plt.imshow(pic_red)

plt.show()

# ntc the color of the ears of the dog, slightly brownish red


plt.imshow(pic_red[:,:,1])

plt.show()

# This is for Green

# ntc the color of the ears, green is gone, they r slightly dark


plt.imshow(pic_red[:,:,2])

plt.show()

# This is for Blue

# ntc the color of the ears, , they r darker


# This is G channel

pic_red[:,:,1]


# Zeroing Green

# Tk out the G channel

pic_red[:,:,1] = 0


pic_red[:,:,1]


# make all the green vals 0
```

```
plt.imshow(pic_red)

plt.show()

# ntc it looks purplsh, why, only b & r r left


# tk out the blue

pic_red[:,:,2] = 0

plt.imshow(pic_red) # ntc it looks redsh

plt.show()


pic_red.shape


pic_red[:,:,0].shape


pic_red[:,:,1].shape
```

 # *

 # With OCV

 # open cmd prompt as admin

 # #conda install opencv #-python

 # #pip install opencv-python

#

'''

https://www.lfd.uci.edu/~gohlke/pythonlibs/

find opencv

download

opencv_python-3.4.6-cp37-cp37m-win_amd64.whl

 # open cmd prompt as admin

python -m pip install --upgrade pip

go to the downloaded dir

!pip3 install opencv_python-3.4.6-cp37-cp37m-win_amd64.whl

'''

pip3 install opencv-contrib-python

 # Open Ana Navigator

 # Select Environments on LHS

 # Sel Ana

 # Sel Play

 # Sel jupyter notebook

 # v can import cv2 (It appears like it did not install in base env)

import numpy as np

import matplotlib.pyplot as plt

```
get_ipython().magic('matplotlib inline')



import cv2


img = cv2.imread('00-puppy.jpg')


img


plt.imshow(img) # ntc slightly bluish, why
# ocv & mtplt hv dfrnt orders of rgb channels
# mtplt fills in as r g b
# ocv fills in b g r
# The image has been correctly loaded by openCV as a numpy array,
# but the color of each pixel has been sorted as BGR.
# Matplotlib's plot expects an RGB image so, for a correct display of the image,
# swap those channels.


# This can be done by using openCV conversion functions cv2.cvtColor()
# or by working directly with the numpy array.
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # this swaps the channels
# this is converting it to bgr to rgb
plt.imshow(img_rgb)
plt.show()
```

```
img_gray = cv2.imread('00-puppy.jpg',cv2.IMREAD_GRAYSCALE)
plt.imshow(img_gray)
plt.show()
# why it is not in BW, coz of the order of channels
```

```
img_gray = cv2.imread('00-puppy.jpg',cv2.IMREAD_GRAYSCALE)
plt.imshow(img_gray,cmap='gray')
plt.show()
```

```
img_gray = cv2.imread('00-puppy.jpg',cv2.IMREAD_GRAYSCALE)
plt.imshow(img_gray,cmap='magma')
plt.show()
```

```
print(img_gray.shape)
# ntc it has only 1 channel

print(img_gray.min()) # min pxl val
print(img_gray.max()) # max pxl val
```

```
# ## Resize Images, Img manipulation

plt.imshow(img_rgb) # this is the orgnl img

plt.show()


img_rgb.shape

# width, height, color channels


img =cv2.resize(img_rgb,(1000,400)) # resize it, this will squeeze it

plt.imshow(img) # ntc the X-axis & Y-axis

plt.show()


img =cv2.resize(img_rgb,(1300,275)) # resize it

plt.imshow(img)

plt.show()



# ### Resize By ratio


w_ratio = 0.8

h_ratio = 0.2


new_img =cv2.resize(img_rgb,(0,0),img,w_ratio,h_ratio) # 80% smaller

plt.imshow(new_img)
```

```
plt.show()



w_ratio = 0.5 # 50% of orgnl width

h_ratio = 0.5 # 50% of orgnl ht


new_img =cv2.resize(img_rgb,(0,0),img,w_ratio,h_ratio) # 50% smaller

plt.imshow(new_img)

plt.show()


plt.imshow(img_rgb)

plt.show()


print(new_img.shape)

print(img_rgb.shape)


#get_ipython().magic('matplotlib qt')

plt.imshow(new_img)

plt.show()




# ### Flipping Images

#get_ipython().magic('matplotlib inline')
```

```
# Along central x axis
new_img = cv2.flip(new_img,0) # 0 is hor axis
plt.imshow(new_img)
plt.show()
```

```
# Along central y axis
new_img = cv2.flip(new_img,1) # 1 is vert axis
plt.imshow(new_img)
plt.show()
```

```
# Along both axis
new_img = cv2.flip(new_img,-1)
plt.imshow(new_img)
plt.show()
```

```
# # Saving Image Files


type(new_img)


cv2.imwrite('my_new_picture.jpg',new_img)

# the above stored the BGR version of the image.

# ## Larger Displays in the Notebook, as the NB displays smaller one by default


fig = plt.figure(figsize=(10,8)) # defining the canvas size

ax = fig.add_subplot(111)

ax.imshow(img_rgb)


fig = plt.figure(figsize=(2,2))

ax = fig.add_subplot(111)

ax.imshow(img_rgb)

# 001




# Opening imgs with opencv in a script

# RUN AS .py SCRIPT not in NB.

# RUNNING THIS CELL WILL KILL THE KERNEL IF YOU USE JUPYTER DIRECTLY
```

```
#open cmd to this folder
#cv_py_1.py

import os
print(os.getcwd())
os.chdir('E:\\Locker\\Sai\\SaiHCourseNait\\DecBtch\\R_Datasets\\')
print(os.getcwd())
import cv2
img = cv2.imread('00-puppy.jpg',cv2.IMREAD_GRAYSCALE)

while True:
        # Show the image with OpenCV
        cv2.imshow('MyPuppy',img)
        # Wait for something on keyboard to be pressed to close window.
        # Wait for 1 ms & if Esc key is pressed
        if (cv2.waitKey() & 0xFF==27):
                break

cv2.destroyAllWindows()
```

```
# 002

# # Drawing on Images

blank_img = np.zeros(shape=(512,512,3),dtype=np.int16)

# gv a shape of 512 x 512

blank_img #ntc all 0s, 0 is black, 1 is white


plt.imshow(blank_img)



print(blank_img.shape)
```

 # -----

 # # Shapes

 # ----

 # ### 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.

 # pt1 = top left corner, x1,y1

 # pt2 = bottom right conrner, x2,y2

cv2.rectangle(blank_img,pt1=(384,0),pt2=(510,128),color=(0,255,0),thickness=5)

 # v r giving 2 pts

 # cv2.rectangle(blank_img,pt1=(384,0),pt2=(510,128),color=(0,255,0))

plt.imshow(blank_img)

 # ntc the thickness moves from middle to rt so it is reducing it

```
blank_img = np.zeros(shape=(512,512,3),dtype=np.int16)

cv2.rectangle(blank_img,pt1=(384,10),pt2=(510,150),color=(0,255,0),thickness=5)

plt.imshow(blank_img) # ntc the thickness




# draw a blue rectangle in the middle of the image.

# pt1 = top left

# pt2 = bottom right

cv2.rectangle(blank_img,pt1=(200,200),pt2=(300,300),color=(0,0,255),thickness=5)

plt.imshow(blank_img)



# ## Circles

cv2.circle(img=blank_img, center=(100,100), radius=50, color=(255,0,0), thickness=8)

plt.imshow(blank_img)



# ### Filled In

cv2.circle(img=blank_img, center=(400,400), radius=50, color=(255,0,0), thickness=-1)

plt.imshow(blank_img)



# ### Lines

# 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)



# ### Text

font = cv2.FONT_HERSHEY_SIMPLEX

cv2.putText(blank_img,text='Hello',org=(10,500), fontFace=font,fontScale=
4,color=(255,255,255),thickness=2,lineType=cv2.LINE_AA)

#org is the bottom left corner of the text string in the img

#color is white

plt.imshow(blank_img)



# ### Polygons

# To draw a polygon, first we need coordinates of vertices.

# Make those points into an array of shape ROWSx1x2 where ROWS are number of vertices

# and it should be of type int32.

blank_img = np.zeros(shape=(512,512,3),dtype=np.int32)

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

vertices


vertices.shape


pts = vertices.reshape((-1,1,2)) # add a 3rd dim ie 1

pts
```

```
cv2.polylines(blank_img,[pts],isClosed=True,color=(255,0,0),thickness=5)

plt.imshow(blank_img)

#color is red
```

```
#

# # Direct Drawing with Mouse dynamically not statically

# ## THESE SHOULD ALL BE RUN AS A .py SCRIPT. or in a single cell

# ## SCRIPT 1: Connecting a Function for Drawing

#v1

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):

    pass
```

```
# 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)

# Create a black image
img = np.zeros((512,512,3), np.uint8)

while True: #Runs forever until we break with Esc key on keyboard
    # Shows the image window
    cv2.imshow('my_drawing',img)
    if cv2.waitKey(20) & 0xFF == 27:
        break
# Once script is done, its usually good  to call this line
# It closes all windows (just in case you have multiple windows called)
cv2.destroyAllWindows()
# Esc to close



# V2
def draw_circle(event,x,y,flags,param): # x,y r pssd from setMouseCallback
    if event == cv2.EVENT_LBUTTONDOWN: # when the left of mouse is clicked down
        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)
    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()
```

```
# V3

# ## Adding Functionality with Event Choices

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)

    elif event == cv2.EVENT_RBUTTONDOWN:

        cv2.circle(img,(x,y),100,(0,0,255),-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)
    if cv2.waitKey(20) & 0xFF == 27:
        break
cv2.destroyAllWindows()
```

```
# V4
# # Dragging with Mouse
import cv2
import numpy as np
```

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

# mouse callback function
def draw_rectangle(event,x,y,flags,param): # x, y r pssed by setMousecallback
    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)
                                # ix & iy r the position where v hv alrd clckd, like x1, y1
                                # x & y where the mouse currently is, like x2, y2
```

```
                    # color is green
                    # -1 will fill the rect

    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
cv2.destroyAllWindows()
```

 # 004

 # Assignment

 # #### 1. Flip the image back pack dog upside down and display it in the notebook.

 # #### 2. Draw an empty RED rectangle around the dogs face and display the image in the notebook.

 # #### 3. Draw a BLUE TRIANGLE in the middle of the image. The size and angle is up to you, but it should be a triangle (three sides) in any orientation.

 # ### 4. figure our how to fill in this triangle?

 # #### 5. (NOTE: YOU WILL NEED TO RUN THIS AS A SCRIPT).

 # Create a script that opens the picture and allows you to draw empty red circles

 # whever you click the RIGHT MOUSE BUTTON DOWN.

```
# *** Image Processing Techniques:


img = cv2.imread('00-puppy.jpg')

plt.imshow(img) # this is default colorspace


# ### Converting to Different Colorspaces


img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img)


# **Converting to HSV**

# https://en.wikipedia.org/wiki/HSL_and_HSV

img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # just another colorspace model

plt.imshow(img)
```

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)# just another colorspace model

plt.imshow(img)
```

```
# *** Blending & Pasting Images *** #


# Two images

img1 = cv2.imread('dog_backpack.png')

plt.imshow(img1)

plt.show()


img2 = cv2.imread('watermark_no_copy.png')

plt.imshow(img2)

plt.show()
```

```
print(img1.shape)
print(img2.shape)


img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
plt.imshow(img1)
plt.show()

img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
plt.imshow(img2)
plt.show()
```

```
# ### Resizing the Images


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

# resizing the image, since the DO NOT COPY image is actually quite large 1200 by 1200,

# and puppy in backpack image is 1400 by 1000

plt.imshow(img1)

plt.show()


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

plt.imshow(img2)

plt.show()



# ### Blending the Image

# blend the values together with the formula:

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

print(img1.shape)


print(img2.shape)


# addWeighted only works for the imgs of same size

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

plt.imshow(blended)

plt.show()
```

```
# ntc back pack img has lighter background



blended = cv2.addWeighted(src1=img1,alpha=0.8,src2=img2,beta=0.1,gamma=0.5)

plt.imshow(blended)

plt.show()



blended = cv2.addWeighted(src1=img1,alpha=0.8,src2=img2,beta=0.1,gamma=10)

plt.imshow(blended)

plt.show()









# ## Overlaying Images of Different Sizes,

# overlay large img on a small img

# this is (No Blending)

# trick to overlap different sized images,

# by simply reassigning the larger image's values to match the smaller image.
```

```
# Load two images
img1 = cv2.imread('dog_backpack.png')
img2 = cv2.imread('watermark_no_copy.png')
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

img2 =cv2.resize(img2,(600,600))
plt.imshow(img2)
plt.show()

plt.imshow(img1) #ntc the X-axis & Y-axis
plt.show()

large_img = img1
print(large_img.shape)

small_img = img2
print(small_img.shape)

# Numpy slicing to the rescue
x_offset=0
y_offset=0

x_end = x_offset + small_img.shape[1] # w & h
print(x_end)
```

```
y_end = y_offset + small_img.shape[0] # w & h

print(y_end)


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

large_img[y_offset:y_end, x_offset:x_end] = small_img

plt.imshow(large_img)

plt.show()

# this is replacing the large img on smll img
```

```
# ## Blending Images of Different Sizes & Masking & roi

# Blend opencv with messy image not replacing opencvs blk bg on messy

# ### Importing the images again and resizing

# Load two images
```

```
img1 = cv2.imread('dog_backpack.png')

img2 = cv2.imread('watermark_no_copy.png')

img2 =cv2.resize(img2,(600,600))

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

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



plt.imshow(img1)

plt.show()


plt.imshow(img2)

plt.show()


# ### Create a Region of Interest (ROI)

print(img1.shape)


# v can tk any coords

x_offset=934-600 # 934 is the wdth of the larger image

y_offset=1401-600 # 1401 is the ht of the larger img



print(img2.shape)


# Creating an ROI of the same size of the foreground image

# (smaller image that will go on top)
```

```
rows,cols,channels = img2.shape

# tuple unpacking


print(rows)

print(cols)


# create roi

# roi = img1[0:rows, 0:cols ] # TOP LEFT CORNER

roi = img1[y_offset:1401,x_offset:943] # BOTTOM RIGHT CORNER

# this is the btm rt hnd corner of the img

plt.imshow(roi)

plt.show()

print(roi.shape)



# ### Next step v create a Mask

# create a mask of logo and create its inverse mask also

img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

plt.imshow(img2gray)

plt.show()


print(img2gray.shape)


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

plt.show()
```

```
# inverse the colors
mask_inv = cv2.bitwise_not(img2gray)
plt.imshow(mask_inv)
plt.show()

# this is b&w
plt.imshow(mask_inv,cmap='gray')
plt.show()

print(mask_inv.shape)
#ntc the color channels is gone

# ## Convert Mask to have 3 channels
white_background = np.full(img2.shape, 255, dtype=np.uint8)
white_background

plt.imshow(white_background)

bk = cv2.bitwise_or(white_background, white_background, mask=mask_inv)
bk
plt.imshow(bk)

print(bk.shape)
```

```
# ### Grab Original FG image and place on top of Mask
plt.imshow(mask_inv,cmap='gray')
plt.show()

fg = cv2.bitwise_or(img2, img2, mask=mask_inv)
plt.imshow(fg)
plt.show()

plt.imshow(img2)

print(fg.shape)

# ### Get ROI and blend in the mask with the ROI
print(roi)

plt.imshow(roi)
plt.show()

final_roi = cv2.bitwise_or(roi,fg)
plt.imshow(final_roi)
plt.show()

# ### Now add in the rest of the image
large_img = img1
small_img = final_roi
```

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

plt.imshow(large_img)

plt.show()
```

```
# *** Image thresholding *** #


img = cv2.imread('rainbow.jpg')

plt.imshow(img)

plt.show()


# Adding the 0 flag to read it in black and white

img = cv2.imread('rainbow.jpg',0)

plt.imshow(img,cmap='gray')# this is gray scale image

plt.show()


# ## Different Threshold Types

ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

# thresholding is where certain vals r cutoff ie lt r assigned below 0

# and others abv r 1

ret

# 127 is min

# 255 is max pssbl thrshld


img.max() # this max val in the img
```

```
plt.imshow(thresh1,cmap='gray')
plt.show()
# this is binary img, all r either 0 or 255
# 0 is blk



# ### Binary Inverse
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
plt.imshow(thresh2,cmap='gray')
plt.show()



# ###  Threshold Truncation
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
plt.imshow(thresh3,cmap='gray')
plt.show()
# sm of them keep the orignl val and others get down back to the threshld
# if it is abv cutoff it is threshlding it
# otherwise it keeps it



# ### Threshold to Zero
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
plt.imshow(thresh4,cmap='gray')
plt.show()
```

```
# ### Threshold to Zero (Inverse of abv)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)
plt.imshow(thresh5,cmap='gray')
plt.show()



# # Real World Applications
# ## Adaptive Thresholding
# ### Sudoku Image
img = cv2.imread("crossword.jpg",0)
plt.imshow(img)
plt.show()

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

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

```
# ## Simple Binary

ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

show_pic(th1)

# v loose sm of the quality, sm of the gray spaces get thrlded to white

# but v want them to keep as blk


# incr the thrshld

ret,th1 = cv2.threshold(img,200,255,cv2.THRESH_BINARY)

show_pic(th1)


# ### Adaptive Threshold is a soln to losing quality

#    @param src Source 8-bit single-channel image.

#    . @param dst Destination image of the same size and the same type as src.

#    . @param maxValue Non-zero value assigned to the pixels for which the condition is satisfied

#    . @param adaptiveMethod Adaptive thresholding algorithm to use, see
#AdaptiveThresholdTypes.

#    . The #BORDER_REPLICATE | #BORDER_ISOLATED is used to process boundaries.

#    . @param thresholdType Thresholding type that must be either #THRESH_BINARY or
#THRESH_BINARY_INV,

#    . see #ThresholdTypes.

#    . @param blockSize Size of a pixel neighborhood that is used to calculate a threshold value for
the

#    . pixel: 3, 5, 7, and so on.

#    . @param C Constant subtracted from the mean or weighted mean (see the details below).
Normally, it

#    . is positive but may be zero or negative as well.
```

```
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,8)

# it will look around the neighborhood of pixl vals

# the blk squares r not filled in due to thrshld/cutoff unlike binary thrsld

# Play with last 2 numbers

show_pic(th2)



th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY,15,8)

show_pic(th3)



blended = cv2.addWeighted(src1=th1,alpha=0.7,src2=th2,beta=0.3,gamma=0)

show_pic(blended)

# this is a blend of dfrnt thrshlods
```

```
# *** Blurring & Smoothing


# ## Convenience Functions

# function for loading the puppy image.

import warnings

warnings.filterwarnings('ignore')


def load_img():

    img = cv2.imread('bricks.jpg').astype(np.float32) / 255

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    return img


def display_img(img):

    fig = plt.figure(figsize=(12,10))

    ax = fig.add_subplot(111)

    ax.imshow(img)


i = load_img()
```

```
display_img(i)
```

```
# ### Gamma Correction : Practical Effect of Increasing Brightness
img = load_img()
gamma = 1/4
effected_image = np.power(img, gamma)
display_img(effected_image)
# Gamma Corr is abt increasing/decreasing the brightness effect
# the img is brighter if gamma < 1
```

```
gamma = 1/10
effected_image = np.power(img, gamma)
display_img(effected_image)
# img fades more
```

```
img = load_img()
gamma = 2
effected_image = np.power(img, gamma)
```

```
display_img(effected_image)

# since G is > 1 img gets darker




gamma = 8

effected_image = np.power(img, gamma)

display_img(effected_image)

# almost black




# Blurring

# ### Low Pass Filter with a 2D Convolution

# A fitlering operation known as 2D convolution can be used to create

# a low-pass filter.

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)

#org is the position


# setup the K for LPF
```

```
# ### Create the Kernel

kernel = np.ones(shape=(5,5),dtype=np.float32)/25

kernel
```

```
1/25 # 0.04
```

```
dst = cv2.filter2D(img,-1,kernel)

# -1 is neg depth

display_img(dst)

# ntc the spacing in letters is pink & filled
```

```
# ## Averaging

# bck to orgnl img

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)

# this is the orgnl img
```

```
blurred_img = cv2.blur(img,ksize=(5,5)) # this is default cv2s built in kernel

display_img(blurred_img)



blurred_img = cv2.blur(img,ksize=(10,10)) # this is default cv2s built in kernel

display_img(blurred_img)









# ## Gaussian Blurring

# tks a group of pxls calc avg or median then mk them the outputs

# this the fresh img

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)



blurred_img = cv2.GaussianBlur(img,(5,5),10)
```

```
display_img(blurred_img)
```

```
# ## Median Blurring
```

```
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)
```

```
median = cv2.medianBlur(img,5)
display_img(median)
# ntc k in bricks it is not tht much blurrd as in gaussian blur
# it is more clear this is removing noise from text
```

```
# ### Adding Noise
# a more useful case of Median Blurring by adding some random noise to an image.
img = cv2.imread('sammy.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
display_img(img)

print(img.max())
```

```
print(img.min())

print(img.mean())

print(img.shape)



noise_img = cv2.imread('sammy_noise.jpg') # this is noisy img

display_img(noise_img)



median = cv2.medianBlur(noise_img,5)

display_img(median)

# this looks btr than the noisy img although not as clear as the org img







# ## Bilateral Filtering

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)



blur = cv2.bilateralFilter(img,9,75,75)
```

```
display_img(blur)

# the img has blurrd but the txt is slightly clear

# it is in between median blur & mean

# the obj is to reduce the noise or the detail in the img
```

```
# *** Morphological operators

# They r specialzed kernels to achieve special effects


def load_img():

    blank_img =np.zeros((600,600))

    font = cv2.FONT_HERSHEY_SIMPLEX

    cv2.putText(blank_img,text='ABCDE',org=(50,300), fontFace=font,fontScale=
5,color=(255,255,255),thickness=25,lineType=cv2.LINE_AA)

    return blank_img
```

```
def display_img(img):
    fig = plt.figure(figsize=(6,5))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap='gray')


img = load_img()
display_img(img)


# ## Erosion
# Erodes away boundaries of foreground objects.
# Works when foreground is light color (preferrably white) and background is dark.
kernel = np.ones((5,5),np.uint8)
# create the kernel
erosion1 = cv2.erode(img,kernel,iterations = 1)
display_img(erosion1)
# FG is white, BG is black
# ntc the cnnection btw A & B is slightly weak compared to the orgnl


img = load_img()
kernel = np.ones((5,5),np.uint8)
erosion5 = cv2.erode(img,kernel,iterations = 4)
```

```
display_img(erosion5)

# cnnctns r getting eroded away




# ## Opening

# Opening is erosion followed by dilation. Used in removing background noise!

img = load_img()

white_noise = np.random.randint(low=0,high=2,size=(600,600))

white_noise

# arr of pts 0 & 1 of size 600 x 600



display_img(white_noise)


img.max()

# ntc 0s and 1s



# now add the white noise into our txt img

white_noise = white_noise*255

white_noise

# this is the same scale of orgl img
```

```
display_img(white_noise)



# now add the white noise into our txt img
noise_img = white_noise+img
display_img(noise_img)
# img of dark bg & light fg w a lot of noise



# morphological operator
opening = cv2.morphologyEx(noise_img, cv2.MORPH_OPEN, kernel)
display_img(opening)
# ntc v r able to remove the noise wo distorting the orgnl img
# this is for remving the bg noise



#if v now see the orgl img
display_img(img)
# dilation is expanding on img



# ### Closing
#  Useful in removing noise from foreground objects, as black dots on top of the
# white text.
```

```
img = load_img() # reload/reset the img

display_img(img)



black_noise = np.random.randint(low=0,high=2,size=(600,600))

black_noise



display_img(black_noise) # looks same as white noise



black_noise= black_noise * -255

black_noise # ntc the neg vals it will not affect the black but white fg



display_img(black_noise) # ntc the white fg is filled with noise



# add blck nose to the img

black_noise_img = img + black_noise

display_img(black_noise_img)

display_img(noise_img) # this is white noise
```

```
#black_noise = np.random.randint(low=0,high=2,size=(600,600))

#black_noise


#display_img(black_noise_img)



# v will try to mk the neg vals to zeros, as the pxls r from 0 to 255

black_noise_img[black_noise_img==-255] = 0

black_noise_img

display_img(black_noise_img)



# closing is a process to clean up the FG

# opening is a process to clean up he BG

closing = cv2.morphologyEx(black_noise_img, cv2.MORPH_CLOSE, kernel)

display_img(closing)



# ## Morphological Gradient

# Difference between dilation and erosion of an image.

img = load_img()
```

```
display_img(img)



# erosion tries to remove the edges of alphabet and blck bg

# dilation tries to add around the edges and mkae it more bubbly

# gradient will tk the dfrnt btw the 2

gradient = cv2.morphologyEx(img,cv2.MORPH_GRADIENT,kernel)

display_img(gradient)

# this is a simple process of edge detection

# this is the dfrnt btw erosion & dilation
```

```
# *** Gradients
# is an extension of morphological operators
# it helps in sohpistciated opers such as edge detection
# which r used in object detection, tracking & image classification
# In gradient the color changes from black to white and viceversa
# can b tracked using algs


img = cv2.imread('sudoku.jpg',0)

def display_img(img):
    fig = plt.figure(figsize=(6,5))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap='gray')

display_img(img)


# this is x gradeint Sobel
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
# cv2.CV_64F is pxl precision
# 1 is derivative x (dx)
# 0 is derivative y (dy)
```

```
# ksize is an odd num v can chang this
display_img(sobelx)
# vert lins r clear
# it ds not erase any hor lines


sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
display_img(sobely)
# ntc hor lines r clear


# this is the laplacean of the img
# 2nd der of x & 2nd der of y
laplacian = cv2.Laplacian(img,cv2.CV_64F)
display_img(laplacian)
# ntc it tries to do edge detection wrt to x & y
# it gets rid of sm noise
# ntc the edges around 8 went frm white to black to white
# nd the nums r clear
# ex of an appli whi will look at a sudoku pic
# nd auto fill in sm nums
```

```
# ### Blending Images

# for combining multiple things

# combining sobel & x grad & sobel & y grad

blended = cv2.addWeighted(src1=sobelx,alpha=0.5,src2=sobely,beta=0.5,gamma=0)

display_img(blended)

# this is the blended result of x grad & y grad

# edges r more clear

# v can also combine thresholding & morphological operators


# using threshold oper

ret,th1 = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)

display_img(th1)

# this also detects the edges

# but it dsnt see the grid lines inside
```

```
blended.shape


# using morp oper

# ### Morphological Operators

kernel = np.ones((4,4),np.uint8)

gradient = cv2.morphologyEx(blended,cv2.MORPH_GRADIENT,kernel)

display_img(gradient)

# this gvs b & w result

# this is used for edge detection

# In edge detection all the techniqs r combined and called




# more combinations

# Try it on laplacian result

kernel = np.ones((3,3),np.uint8)

gradient = cv2.morphologyEx(blended,cv2.MORPH_GRADIENT,kernel)

display_img(gradient)



# ### Thresholds

ret,th1 = cv2.threshold(img,100,255,cv2.THRESH_BINARY)

display_img(th1)
```

```
ret,th1 = cv2.threshold(gradient,200,255,cv2.THRESH_BINARY_INV)

display_img(th1)



ret,th1 = cv2.threshold(blended,100,255,cv2.THRESH_BINARY_INV)

display_img(th1)
```

```
# *** Histograms *** #
```

```
dark_horse = cv2.imread('horse.jpg') # original BGR

plt.imshow(dark_horse)

plt.show()


show_horse = cv2.cvtColor(dark_horse, cv2.COLOR_BGR2RGB)

plt.imshow(show_horse)

plt.show()

# ntc a lot of black around

# v dont see bgr


rainbow = cv2.imread('rainbow.jpg')

plt.imshow(rainbow)

plt.show()


show_rainbow =cv2.cvtColor(rainbow, cv2.COLOR_BGR2RGB)

plt.imshow(show_rainbow)

plt.show()

# a lot of red dist followed by green and so on



blue_bricks = cv2.imread('bricks.jpg')

plt.imshow(blue_bricks)

plt.show()
```

```
show_bricks = cv2.cvtColor(blue_bricks, cv2.COLOR_BGR2RGB)

plt.imshow(show_bricks)

plt.show()

# this is blue on white
```

```
# ### OpenCV Histogram

# **cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])**

# * images : it is the source image of type uint8 or float32.

# it should be given in square brackets, ie, “[img]”.

# * channels : it is also given in square brackets.

# It is the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.

# * mask : mask image.

# To find histogram of full image, it is given as “None”. But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)

# * histSize : this represents our BIN count.

# Need to be given in square brackets. For full scale, we pass [256].

# * ranges : this is our RANGE. Normally, it is [0,256].
```

```
hist_values = cv2.calcHist([blue_bricks],channels=[0],mask=None,histSize=[256],ranges=[0,256])
```

```
hist_values.shape
```

```
plt.plot(hist_values)

plt.show()

# ntc the peak at 150 & 225



plt.imshow(show_horse)

plt.show()


plt.imshow(dark_horse)

plt.show()


hist_values = cv2.calcHist([dark_horse],channels=[0],mask=None,histSize=[256],ranges=[0,256])

plt.plot(hist_values)

plt.show()

# most of the vals for blue chnnl r 0

# coz most of the img is black

# nd the horse is brown

# remembr blk is 0 pxl and white is 1




# ## Plotting 3 Color Histograms

img = blue_bricks

color = ('b','g','r')
```

```
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.title('Blue Bricks Image')
plt.show()



plt.imshow(show_bricks)
plt.show()
# the contrib of blue is more



img = dark_horse
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.title('Dark Horse')
plt.show()



# zoom x and y axis
for i,col in enumerate(color):
```

```
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,50])
            plt.ylim([0,500000])
plt.title('Dark Horse')
plt.show()
```

```
img = rainbow
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.title('Rainbow Image')
plt.show()
```

```
# ### Masking
# We can mask only certain parts of the image.
img = rainbow
img.shape
```

```
# create a mask


img.shape[:2]

# same as above


mask = np.zeros(img.shape[:2], np.uint8)

mask


plt.imshow(mask)

plt.show()


plt.imshow(mask,cmap='gray')

plt.show()



mask[300:400, 100:400] = 255

# slice a rect from the img

# then set it to 255 whi is wht


plt.imshow(mask,cmap='gray')


plt.imshow(show_rainbow)

# this is color corrected
```

```
masked_img = cv2.bitwise_and(img,img,mask = mask)

# this is used for hist calc


# create a mask in the rainbow

# this is used for rgb color calc

show_masked_img = cv2.bitwise_and(show_rainbow,show_rainbow,mask = mask)




plt.imshow(show_masked_img)

# ntc not much red is here cmprd to orgl img



hist_mask_values_red =
cv2.calcHist([rainbow],channels=[2],mask=mask,histSize=[256],ranges=[0,256])

# rainbow is orgnl img

# 2 is red channl in b g r

# mask is the obj v created on top


plt.plot(hist_mask_values_red)

plt.title('Histogram for RED values for the Masked Area')

# v low vals for red & most of the vals r 0


plt.imshow(show_rainbow)

plt.show()

# ntc a lot of red
```

```
hist_full_values_red = cv2.calcHist([rainbow],channels=[2],mask=None,histSize=[256],ranges=[0,256])

plt.plot(hist_full_values_red)

plt.title('Histogram for RED values of the full image')

plt.show()

# ntc a lot of vals for red
```

```
# # Histogram Equalization
```

```
gorilla = cv2.imread('gorilla.jpg',0)
```

```
def display(img,cmap=None):
    fig = plt.figure(figsize=(5,4))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap)
```

```
display(gorilla)


# ## Single Channel (Grayscale)

display(gorilla,cmap='gray')



# Calc histogram, then equalize it, then visualize the dfrnc, then run on color version

hist_values = cv2.calcHist([gorilla],channels=[0],mask=None,histSize=[256],ranges=[0,256])

# channels[0] is 0 coz it is gray there is only one channel

plt.plot(hist_values)

plt.show()

# looks like v hv lot of lighter color and few darker but not much black

# if v look at the img the lighter colors r coming frm the rock

# bhnd gorilla and gor is also a little lighter nd not compleely dark

# the peak is the lighter gray and as v mv to rhs is the darker colr



# this is a ver of gor

eq_gorilla = cv2.equalizeHist(gorilla)

display(eq_gorilla,cmap='gray')

# ntc the contrast has increased

# the lighter vals now r min vals of 0

# nd the darker vals r black whi r the near the hair of gor
```

```
# nd the edges of rock




hist_values = cv2.calcHist([eq_gorilla],channels=[0],mask=None,histSize=[256],ranges=[0,256])

plt.plot(hist_values)

# ntc a lot of spikes going down coz of 0 vals

# a lot of lighter vals hv cm 0

# whereas the orgnl one is more flat



# ## redo the abv for Color Image

color_gorilla = cv2.imread('gorilla.jpg')

display(color_gorilla)



show_gorilla = cv2.cvtColor(color_gorilla,cv2.COLOR_BGR2RGB)

display(show_gorilla)



# Convert to HSV colorspace to equlaize a color img

# equzli increass the color contrast
```

```
# hue sat val
hsv = cv2.cvtColor(color_gorilla, cv2.COLOR_BGR2HSV)
display(show_gorilla)



# Grab V channel
hsv[:,:,2]
# vals chnnl is wht v r interested in
# 0 chnnl is the hue
# 1 chnnl s the sat

hsv[:,:,2].max() # max val

hsv[:,:,2].min()



# equalize val chnnl, then replace the orgnl vals
hsv[:,:,2] = cv2.equalizeHist(hsv[:,:,2])



# Convert back hsv to RGB to visualize
eq_color_gorilla = cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)
display(eq_color_gorilla)
# compare this to orgnl v hv higher contrast
```

 # ntc the rock bhnd is much lighter compared to the orgnl