

**Assignment # 1**

**Digital Image Processing**

**Submitted To: Dr. Noman Islam**

**Submitted (Group)By:**

**Waleed Shaikh (11496)**

**Ayaz Qureshi (9910)**

**Mustansir Billah Asad Abbasi (9924)**

**Rashid Rahim (9715)**

Date: 2th May 2018

**import numpy as np**

**import cv2**

**from PIL import Image**

**from PIL import ImageFilter**

**from matplotlib import pyplot as plt**

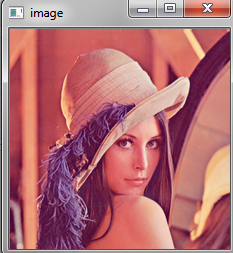
**#Load the image**

img = cv2.imread('lena.png')

cv2.imshow('image',img)

cv2.waitKey(0)

cv2.destroyAllWindows()



**#Average Filter**

img = cv2.imread('lena.png')

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

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

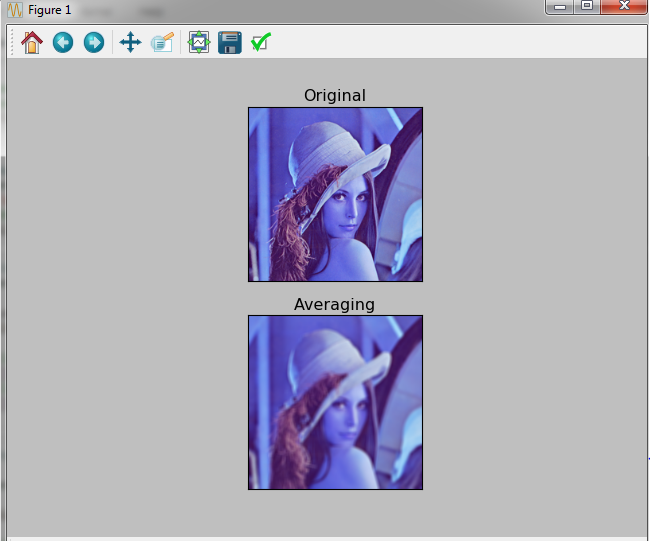
plt.subplot(2,1,1),plt.imshow(img),plt.title('Original')

plt.xticks([]), plt.yticks([])

plt.subplot(2,1,2),plt.imshow(dst),plt.title('Averaging')

plt.xticks([]), plt.yticks([])

plt.show()



**#Image Bluring (Image Smoothing)**

img = cv2.imread('lena.png')

blur = cv2.blur(img,(5,5))

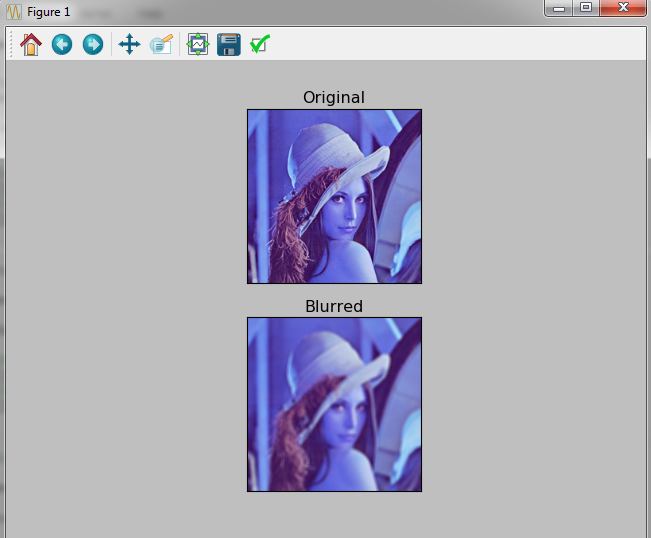
plt.subplot(211),plt.imshow(img),plt.title('Original')

plt.xticks([]), plt.yticks([])

plt.subplot(212),plt.imshow(blur),plt.title('Blurred')

plt.xticks([]), plt.yticks([])

plt.show()



**#Gaussian Filter**

img = cv2.imread('lena.png')

gaussian = cv2.GaussianBlur(img,(5,5),0)

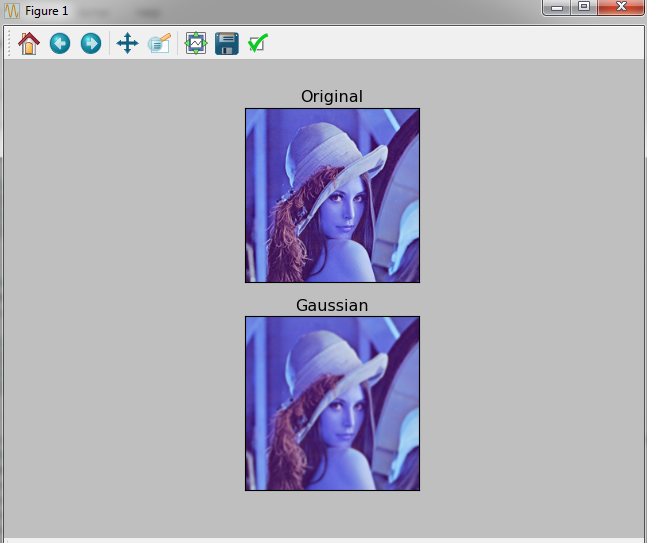
plt.subplot(2,1,1),plt.imshow(img),plt.title('Original')

plt.xticks([]), plt.yticks([])

plt.subplot(2,1,2),plt.imshow(gaussian),plt.title('Gaussian')

plt.xticks([]), plt.yticks([])

plt.show()



**#Median Filtering**

img = cv2.imread('lena.png')

median = cv2.medianBlur(img,5)

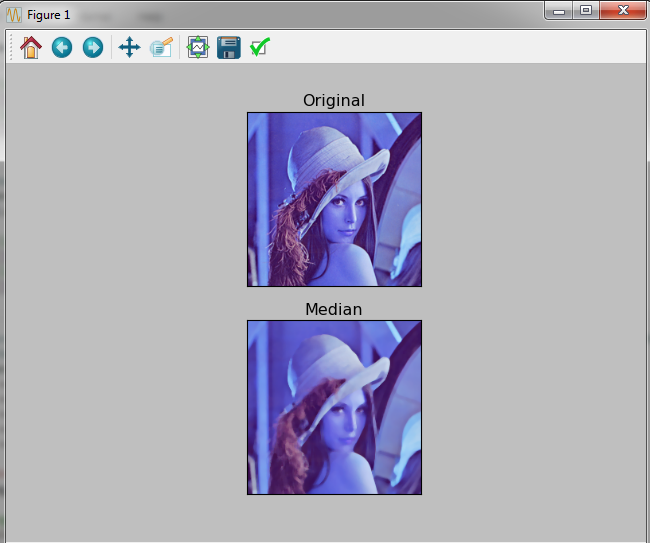
plt.subplot(2,1,1),plt.imshow(img),plt.title('Original')

plt.xticks([]), plt.yticks([])

plt.subplot(2,1,2),plt.imshow(median),plt.title('Median')

plt.xticks([]), plt.yticks([])

plt.show()



**#Apply Different Types Of Thresholding (cv2.THRESH\_BINARY #cv2.THRESH\_BINARY\_INV #cv2.THRESH\_TRUNC #cv2.THRESH\_TOZERO #cv2.THRESH\_TOZERO\_INV)**

**#These are the Simple thresholding Types**

img = cv2.imread('lena.png',0)

ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY\_INV)

ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH\_TRUNC)

ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO)

ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO\_INV)

titles = ['Original Image','BINARY','BINARY\_INV','TRUNC','TOZERO','TOZERO\_INV']

images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

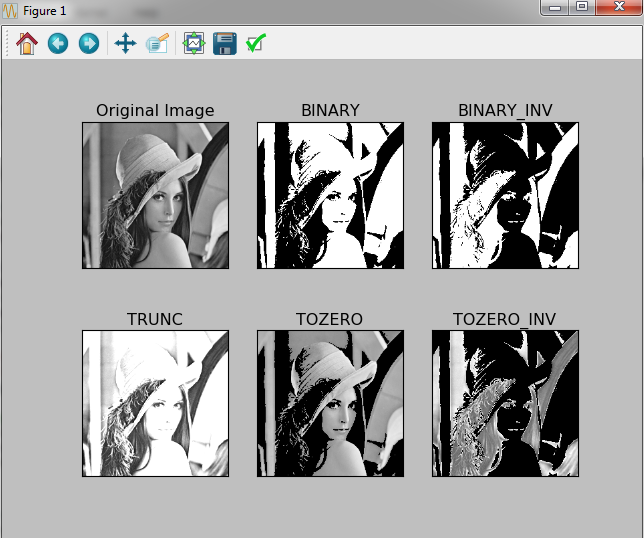
for i in range(6):

plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()



**#Adaptive Thresholding**

img = cv2.imread('lena.png',0)

img = cv2.medianBlur(img,5)

ret,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C,\

cv2.THRESH\_BINARY,11,2)

th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,\

cv2.THRESH\_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',

'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']

images = [img, th1, th2, th3]

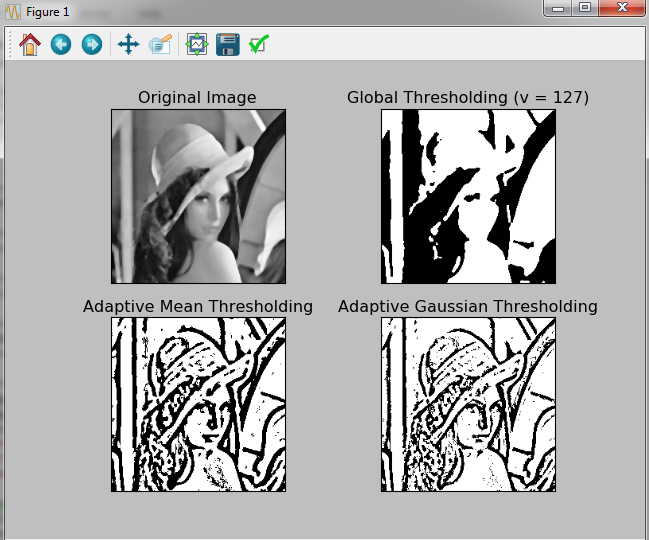
for i in range(4):

plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')

plt.title(titles[i])

plt.xticks([]),plt.yticks([])

plt.show()



**#Otsu's Binary Thresholding**

img = cv2.imread('lena.png',0)

**# global thresholding**

ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

**# Otsu's thresholding**

ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

**# Otsu's thresholding after Gaussian filtering**

blur = cv2.GaussianBlur(img,(5,5),0)

ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

**# plot all the images and their histograms**

images = [img, 0, th1,

img, 0, th2,

blur, 0, th3]

titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',

'Original Noisy Image','Histogram',"Otsu's Thresholding",

'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

for i in range(3):

plt.subplot(3,3,i\*3+1),plt.imshow(images[i\*3],'gray')

plt.title(titles[i\*3]), plt.xticks([]), plt.yticks([])

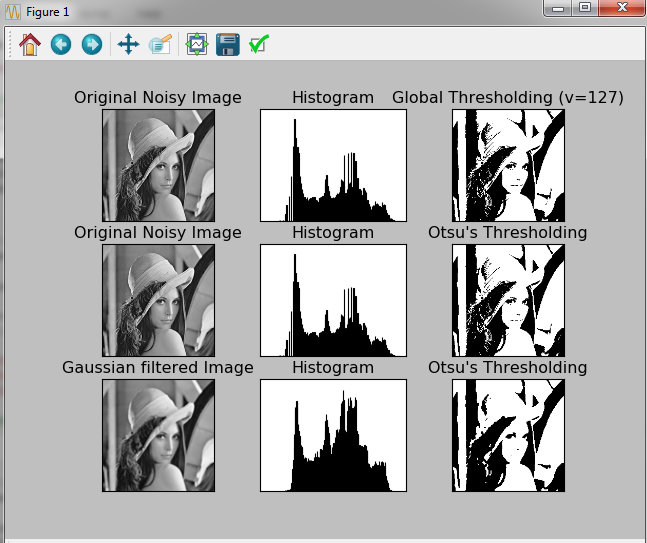
plt.subplot(3,3,i\*3+2),plt.hist(images[i\*3].ravel(),256)

plt.title(titles[i\*3+1]), plt.xticks([]), plt.yticks([])

plt.subplot(3,3,i\*3+3),plt.imshow(images[i\*3+2],'gray')

plt.title(titles[i\*3+2]), plt.xticks([]), plt.yticks([])

plt.show()



**#Erosion and Dilation Apply on Image**

img = cv2.imread('lena.png')

kernel = np.ones((5,5), np.uint8)

img\_erosion = cv2.erode(img, kernel, iterations=1)

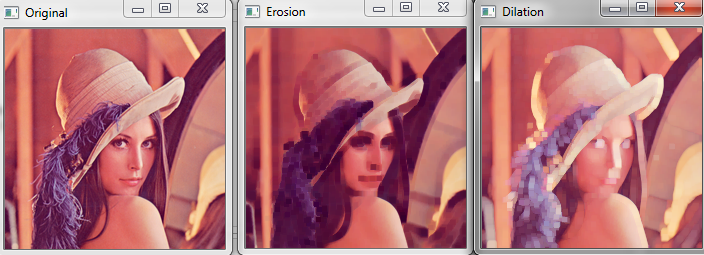
img\_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Original', img)

cv2.imshow('Erosion', img\_erosion)

cv2.imshow('Dilation', img\_dilation)

cv2.waitKey(0)



**#Opening And Closing on Image**

img = cv2.imread('lena.png',0)

kernel = np.ones((5,5), np.uint8)

opening = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel)

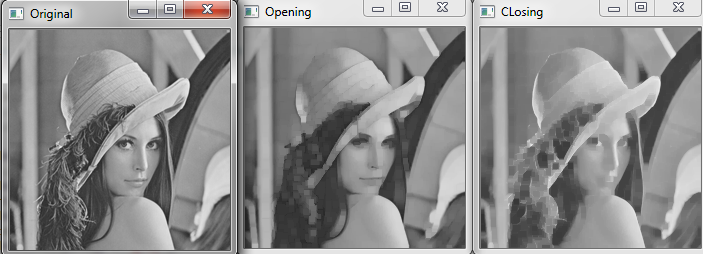
closing = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)

cv2.imshow('Original', img)

cv2.imshow('Opening', opening)

cv2.imshow('CLosing', closing)

cv2.waitKey(0)



**#Laplacian, Sobel on X-axis , Sobel of Y-axis Filter Apply On Image**

img = cv2.imread('lena.png',0)

laplacian = cv2.Laplacian(img,cv2.CV\_64F)

sobelx = cv2.Sobel(img,cv2.CV\_64F,1,0,ksize=5)

sobely = cv2.Sobel(img,cv2.CV\_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')

plt.title('Laplacian'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')

plt.title('Sobel X'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')

plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()img = cv2.imread('lena.png',0)

laplacian = cv2.Laplacian(img,cv2.CV\_64F)

sobelx = cv2.Sobel(img,cv2.CV\_64F,1,0,ksize=5)

sobely = cv2.Sobel(img,cv2.CV\_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')

plt.title('Laplacian'), plt.xticks([]), plt.yticks([])

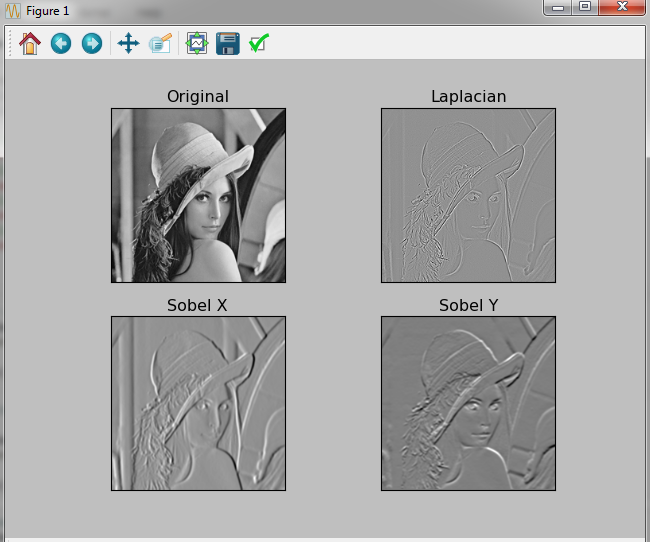
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')

plt.title('Sobel X'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')

plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()

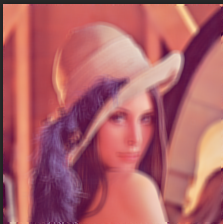


**#Blur Filter Apply**

original = Image.open("lena.png")

blurred = original.filter(ImageFilter.BLUR)

blurred.show()



**#Contour Filter**

original = Image.open("lena.png")

Contour = original.filter(ImageFilter.CONTOUR)

Contour.show()



**#Edge Enhancement Filter**

original = Image.open("lena.png")

edgeEnhance = original.filter(ImageFilter.EDGE\_ENHANCE)

edgeEnhance.show()



**#Edge Enhancement More Filter(mean more Sharpen the Edges)**

original = Image.open("lena.png")

edgeEnhanceMore = original.filter(ImageFilter.EDGE\_ENHANCE\_MORE)

edgeEnhanceMore.show()



**#Embross FIlter Apply on Image Which is help full to detect Edges of Image**

original = Image.open("lena.png")

emboses = original.filter(ImageFilter.EMBOSS)

emboses.show()



**#FIND\_EDGES filter apply on image**

original = Image.open("lena.png")

findEdges = original.filter(ImageFilter.FIND\_EDGES)

findEdges.show()



**#SMOOTH FIlter Apply On Image**

original = Image.open("lena.png")

smooth = original.filter(ImageFilter.SMOOTH)

smooth.show()



**#Sharpen The Image Filter**

original = Image.open("lena.png")

sharpen = original.filter(ImageFilter.SHARPEN)

sharpen.show()



**#Low Pass Filter on Image**

import cv2

import matplotlib.pyplot as plt

def main():

imgpath = "lena.png"

img = cv2.imread(imgpath, 1)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

average = cv2.boxFilter(img, -1, (53, 53))

blur = cv2.blur(img, (13, 13))

gaussian = cv2.GaussianBlur(img, (37, 37), 0)

titles = ['Original Image', 'Average Filter',

'Blur', 'Gaussian Blur']

outputs = [img, average, blur, gaussian]

for i in range(4):

plt.subplot(2, 2, i+1)

plt.imshow(outputs[i])

plt.title(titles[i])

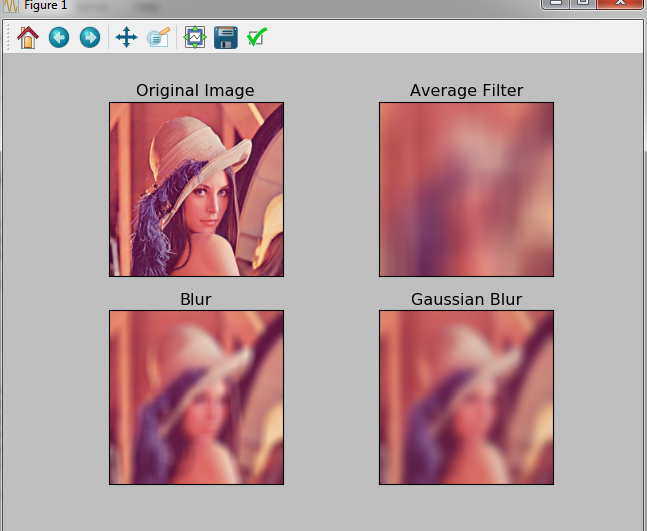
plt.xticks([])

plt.yticks([])

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()



**# High Filter Laplacian**

import cv2

import matplotlib.pyplot as plt

def main():

imgpath = "lena.png"

img = cv2.imread(imgpath, 1)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

edges = cv2.Laplacian(img, -1, ksize=29, scale=1, delta=0,

borderType=cv2.BORDER\_DEFAULT)

output = [img, edges]

titles = ['Original', 'Edges']

for i in range(2):

plt.subplot(1, 2, i+1)

plt.imshow(output[i], cmap = 'gray')

plt.title(titles[i])

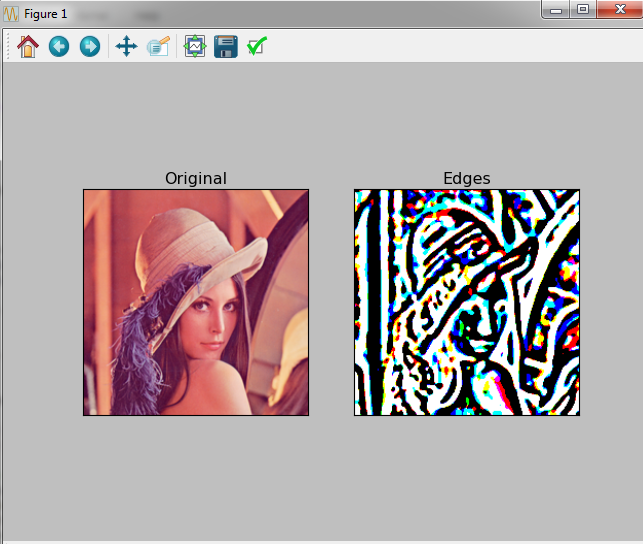
plt.xticks([])

plt.yticks([])

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()



**#Edges detection on x and y axis**

import cv2

import matplotlib.pyplot as plt

def main():

imgpath = "lena.png"

img = cv2.imread(imgpath, 1)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

edgesx = cv2.Sobel(img, -1, dx=3, dy=0, ksize=11, scale=1,

delta=0, borderType=cv2.BORDER\_DEFAULT)

edgesy = cv2.Sobel(img, -1, dx=0, dy=3, ksize=11, scale=1,

delta=0, borderType=cv2.BORDER\_DEFAULT)

edges = edgesx + edgesy

output = [img, edgesx, edgesy, edges]

titles = ['Original', 'dx=1 dy=0', 'dx=0 dy=1', 'Edges']

for i in range(4):

plt.subplot(2, 2, i+1)

plt.imshow(output[i], cmap = 'gray')

plt.title(titles[i])

plt.xticks([])

plt.yticks([])

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()

