|  |  |
| --- | --- |
| Index | 190521G |
| Name | Rathnayake R.M.L.W |

**EN2550 Assignment 1 on Intensity Transformations and Neighborhood Filtering**

1. 

#1

import numpy as np

import matplotlib.pyplot as plt

import cv2 as cv

x\_1 = np.array([i for i in range(0,50)]).astype(np.uint8)

x\_2 = np.array([(100+155/100\*(i-50)) for i in range(50,150)]).astype(np.uint8)

x\_3 = np.array([i for i in range(150,256)]).astype(np.uint8)

#xa = np.array([i for i in range(0,256)]).astype(np.uint8)

x= np.concatenate((x\_1,x\_2,x\_3),axis=0).astype(np.uint8)

img = cv.imread('emma\_gray.jpg')

new\_img = cv.LUT(img,x)

fig , ax = plt.subplots(1,2, figsize = (10,5))

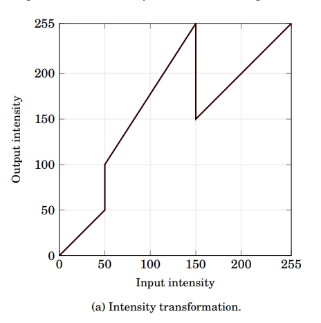
ax[0].imshow(img)

ax[0].set\_title('Original')

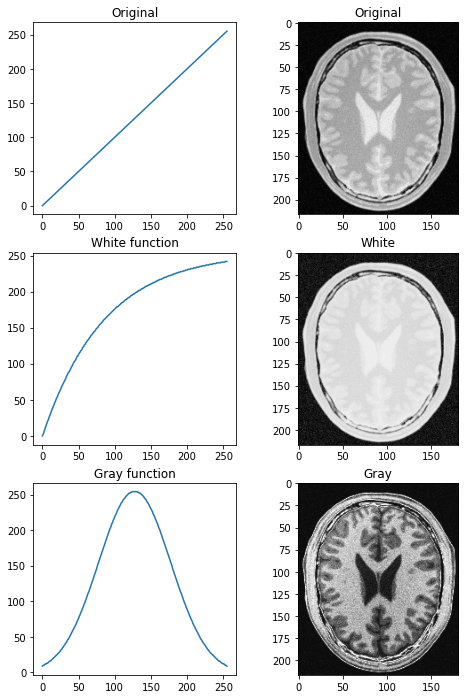
ax[1].imshow(new\_img)

ax[1].set\_title('Transformed')





The intensity has only changed in the region of (50 to 150), therefore the values closer to white and black, hasn’t changed. Value closer to gray, has increased closer to white.

1. 

import numpy as np

import matplotlib.pyplot as plt

import cv2 as cv

x = np.array([i for i in range(0,256)]).astype(np.uint8)

img = cv.imread('brain\_proton\_density\_slice.png')

#function for white

f\_w = np.array([(1-1/np.exp(3\*i/255))\*255 for i in range(0,256)]).astype(np.uint8)

#print(f\_w)

img\_w = cv.LUT(img,f\_w)

#function for grey

f\_g = np.array([(np.exp(-13\*(i/255-0.5)\*\*2))\*255 for i in range(0,256)]).astype(np.uint8)

img\_g = cv.LUT(img,f\_g)

fig, ax = plt.subplots(3,2, figsize= (8,12))

ax[0,0].plot(x,x)

ax[0,0].set\_title('Original')

ax[1,0].plot(x,f\_w)

ax[1,0].set\_title('White function')

ax[2,0].plot(x,f\_g)

ax[2,0].set\_title('Gray function')

ax[0,1].imshow(img)

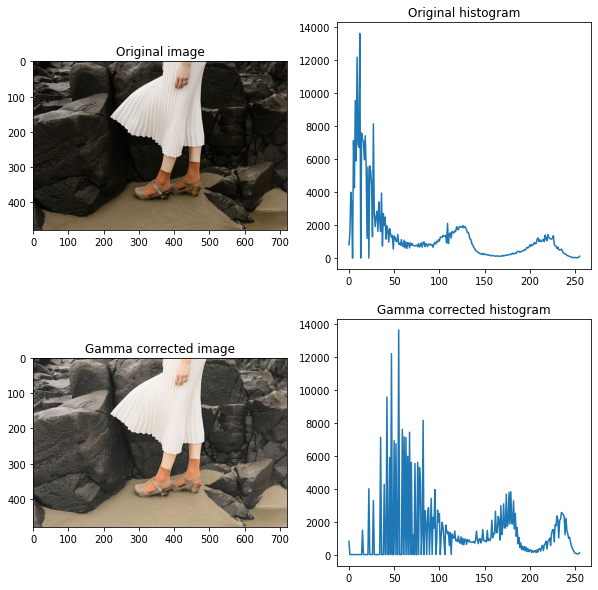
ax[0,1].set\_title('Original')

ax[1,1].imshow(img\_w)

ax[1,1].set\_title('White')

ax[2,1].imshow(img\_g)

ax[2,1].set\_title('Gray')

1. 

import numpy as np

import cv2 as cv

import matplotlib.pyplot as plt

img = cv.imread('highlights\_and\_shadows.jpg')

assert img is not None

lab\_img = cv.cvtColor(img, cv.COLOR\_BGR2LAB)

(L,A,B) = cv.split(lab\_img)

gamma = 0.5

t = np.array([(((i/255)\*\*gamma)\*255) for i in range (0,256)]).astype(np.uint8)

new\_L = cv.LUT(L,t)

corrected\_img = cv.merge([new\_L,A,B])

fig , ax = plt.subplots(2,2, figsize=(10,10))

ax[0][0].imshow(cv.cvtColor(img, cv.COLOR\_BGR2RGB))

ax[0][0].set\_title("Original image")

ax[1][0].imshow(cv.cvtColor(corrected\_img, cv.COLOR\_LAB2RGB))

ax[1][0].set\_title("Gamma corrected image")

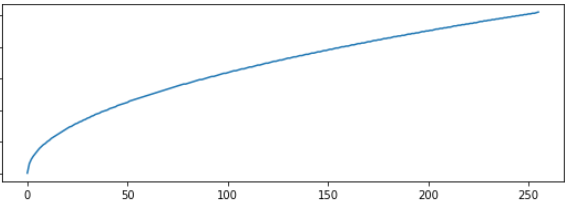
ax[0][1].plot(cv.calcHist([L],[0],None,[256],[0,256]))

ax[0][1].set\_title("Original histogram")

ax[1][1].plot(cv.calcHist([new\_L],[0],None,[256],[0,256]))

ax[1][1].set\_title("Gamma corrected histogram")

The intensity mapping graph is not linear.



Due to this gamma curve, the intensity has slightly changed.

img\_eq= equalize(img)

hist\_1 = cv.calcHist([img], [0], None, [256], [0,256])

hist\_2 = cv.calcHist([img\_eq], [0], None, [256], [0,256])

fig, ax = plt.subplots(2,2,figsize = (20,6))

ax[0][0].imshow(cv.cvtColor(img, cv.COLOR\_GRAY2RGB))

ax[0][0].axis('off')

ax[0][1].plot(hist\_1)

ax[1][0].imshow(cv.cvtColor(img\_eq.astype(np.uint8), cv.COLOR\_GRAY2RGB))

ax[1][1].plot(hist\_2)

ax[1][0].axis('off')

def equalize(img):

    hist = cv.calcHist([img],[0],None, [256], [0,256])  #original histogram

    max\_id =0

    max\_num=0

    for i in range (0,256):

        if hist[i]> max\_num:

            max\_num= hist[i]

            max\_id = i

    look\_up= np.array([0 for i in range (0,256)])

    for p in range (10,246):

        if p< max\_id:

            new\_val = p-((max\_id - p)\*\*2)/20

            if new\_val>=0:

                look\_up[p] = round(new\_val)

        elif p> max\_id:

            new\_val = p+((p- max\_id)\*\*2)/20

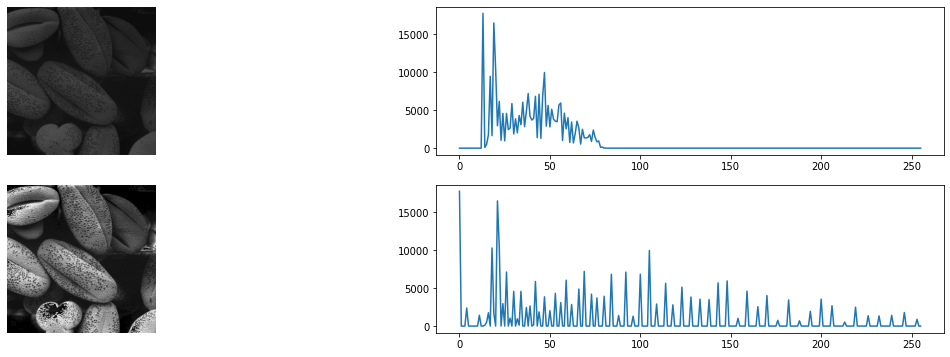
            if new\_val>=0:

                look\_up[p] = round(new\_val)

    a = np.array([i for i in range (0,256)])

    img\_2 =cv.LUT(img,look\_up).astype(np.uint8)

    return(img\_2)

The equalizing function is used to equalize, instead of “equalizeHist()” function.**6.**

img = cv.imread('einstein.png', cv.IMREAD\_GRAYSCALE).astype('float32')

assert img is not None

sobel\_v = np.array([[1,2,1],[0,0,0],[-1,-2,-1]], dtype = np.float32)

img\_v = cv.filter2D(img,-1,sobel\_v)

sobel\_h = np.array([[1,0,-1],[2,0,-2],[1,0,-1]], dtype = np.float32)

img\_h = cv.filter2D(img,-1,sobel\_h)

gradient\_img = np.sqrt(img\_v\*\*2 + img\_h\*\*2)

def sobel(img, kernel):

    #getting the dimentions of image and kernal

    (i\_y , i\_x) = img.shape[:2]

    (k\_y , k\_x) = kernel.shape[:2]

    # make empty array for output image

    # "border" :- borders of input image

    border = (k\_x-1)//2

    img = cv.copyMakeBorder(img, border, border, border, border,cv.BORDER\_REPLICATE)

    output = np.zeros((i\_y,i\_x), dtype = "float32")

    #loop over the image (sliding kernal)

    for y in np.arange(border, i\_y + border):

        for x in np.arange(border, i\_x + border):

            #extract the rrgion of interest of the image

            ROI = img[y - border:y + border + 1, x - border:x + border + 1]

            #convolution

            con = (ROI\* kernel).sum()

            #store the new value in output

            output[y-border, x-border] = con

    # rescale the output image to be in the range [0,255]

    #output = rescale\_intensity(output, in\_range=(0, 255))

    #output = (output \* 255).astype("uint8")

    return output

‘Sobel() ‘ is the function that is manually convoluting an image with a given kernel.

img\_v\_2 = sobel(img, sobel\_v)

img\_h\_2 = sobel(img, sobel\_h)

gradient\_img2= np.sqrt((img\_v\_2)\*\*2 + (img\_h\_2)\*\*2)

The manually calculated gradient is given by “gradient\_img2”.

**Part c**

Using the associative nature of convolution, the gradient is calculated by using the “sobel()” function twice.

# c

k\_1=  np.array([[1],[2],[1]])

k\_2 = np.array([[1,0,-1]])

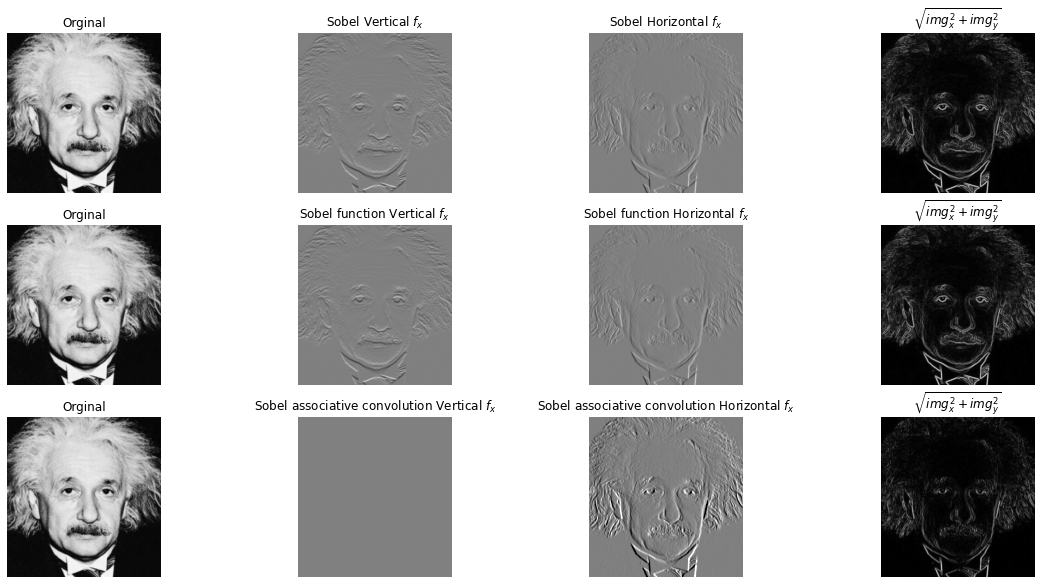
trans\_k\_1=k\_1.T

trans\_k\_2=k\_2.T

img\_v\_3 = sobel(sobel(img,trans\_k\_2), trans\_k\_1)

img\_h\_3 = sobel(sobel(img,k\_1), k\_2)

gradient\_img3= np.sqrt((img\_v\_3)\*\*2 + (img\_h\_3)\*\*2)



**7. a)**

import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

img = cv.imread('daisy.jpg')

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

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

bgd\_Model = np.zeros((1,65),np.float64)

fgd\_Model = np.zeros((1,65),np.float64)

rect = (25,125,530,450)

cv.grabCut(img,mask,rect,bgd\_Model,fgd\_Model,5,cv.GC\_BGD ) #cv.GC\_INIT\_WITH\_RECT)

mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')

#cutted image

img\_cut = img\*mask2[:,:,np.newaxis]

#background

back\_gnd = img - img\_cut

#plot

fig, ax = plt.subplots(2,2, figsize=(10,10))

ax[0,0].imshow(img)

ax[0,1].imshow(mask2)

ax[1,0].imshow(img\_cut)

ax[1,1].imshow(back\_gnd)

ax[0,0].set\_title('Original')

ax[0,1].set\_title('Mask')

ax[1,0].set\_title('cutted image (foregroung)')

ax[1,1].set\_title('back ground')

plt.show()

**b) f**

# enhancing

kernel\_size = 11

sigma = 2

blur\_background = cv.GaussianBlur(back\_gnd, (kernel\_size,kernel\_size),sigma)

img\_enhanced = blur\_background + img\_cut

fig, ax\_1 = plt.subplots(1,2, figsize= (10,5))

ax\_1[0].imshow(img)

ax\_1[0].set\_title('Original')

ax\_1[0].axis('off')

ax\_1[1].imshow(img\_enhanced)

ax\_1[1].set\_title('Enhanced')

ax\_1[1].axis('off')

plt.plot()

**c)**  In the enhanced image, the outline is not much dark, but it is clear that the outline is not very smooth. It is because the “grabCut()” function cannot exactly detect the edges 100% accurately.