Department of Electronics and Telecommunication Engineering
University of Moratuwa

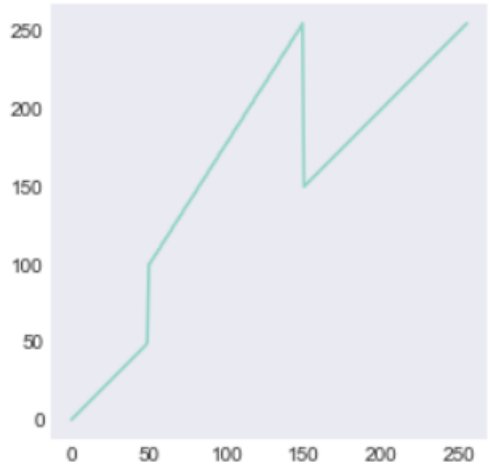
EN2550 - Fundamentals of Image Processing and Machine Vision

Assignment 1

Pathirana R.P.U.A. 190432J

https://github.com/Uvin99/EN2550_Assignment1

Question 1 – Intensity Transformation




```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np


img_orig = cv.imread('emma_gray.jpg')

t1=np.linspace(0,50,50)
t2=np.linspace(100,255,100)
t3=np.linspace(150,255,106)

t = np.concatenate((t1,t2,t3),axis=0).astype(np.uint8)
fig,ax =plt.subplots()
ax.plot(t)
ax.set_aspect('equal')


transformed = cv.LUT(img_orig,t)
f, axarr = plt.subplots()
axarr.imshow(transformed)
axarr.set_xticks([]), axarr.set_yticks([])
```

Original image

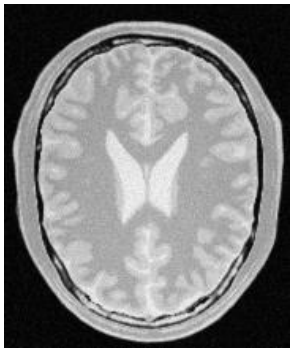
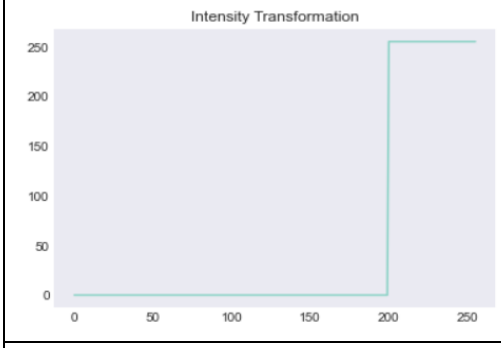
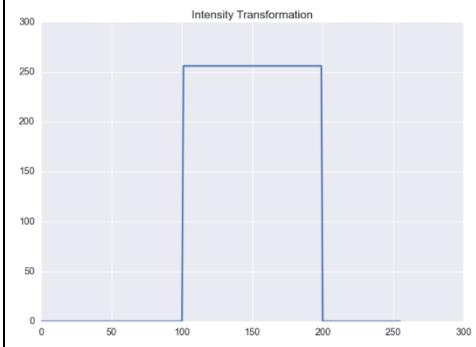
Transformed image

It can be observed that pixels in between 50 to 150 are replaced with values in between 100 to 255.

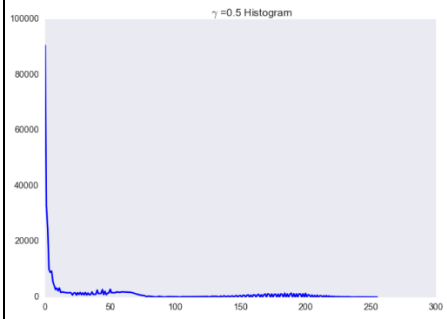
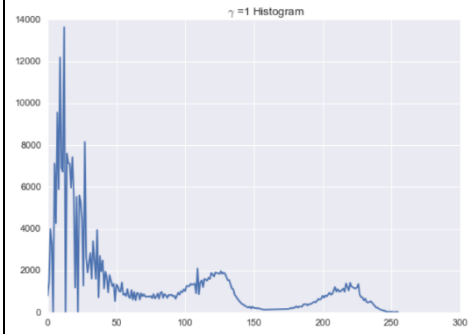
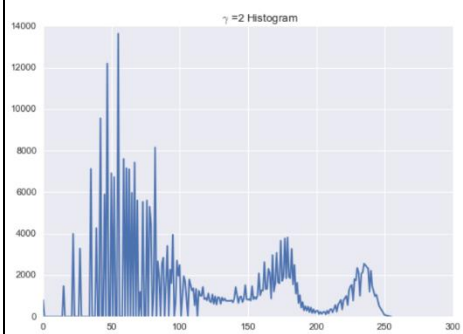
Question 2

When accentuating white matter, pixels with values 200 to 255 are considered as white matter. All the white atter pixels are replaced with the value 255 and all other pixels are replaced with the value 0 so that white matter pixels can be identified.
Pixels with values between 100 to 200 are considered gray matter. They are replaced with 255 (while) pixels and all other pixels are replaced with 0 so that gray matter can be clearly identified.
The implementation approach in python is the same as in question 1.

Intensity transformation plots and result comparisons are displayed in the following table.

Original image	Accentuate white matter	Accentuate gray matter
	 	 

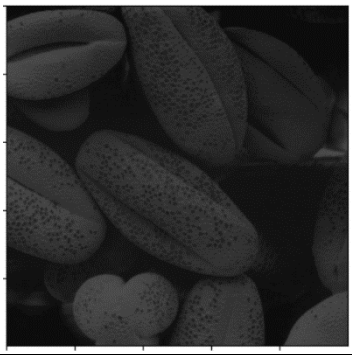
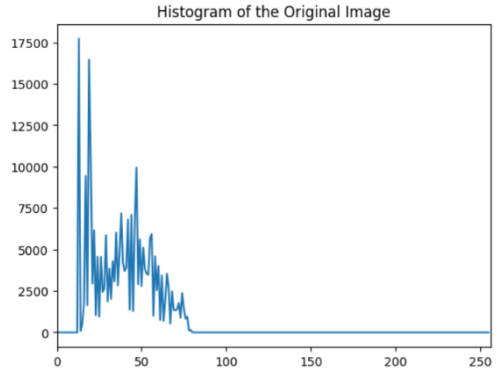
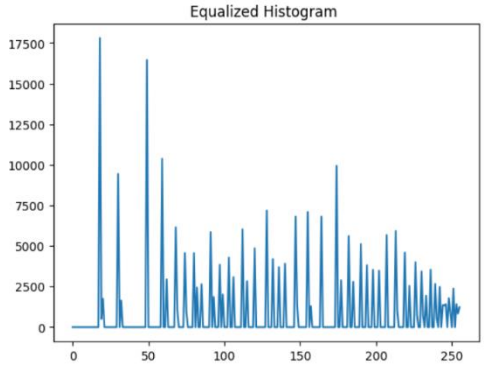
Question 3 – Gamma Correction

$\gamma = 0.5$	$\gamma = 1$ $(Orinal\ Image)$	$\gamma = 2$
		
		

```python
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np

img = cv.imread('highlights_and_shadows.jpg', cv.IMREAD_COLOR)
assert img is not None


gamma_list = [0.5,1,2]
for gamma in gamma_list:
    img_LAB = cv.cvtColor(img, cv.COLOR_BGR2LAB)

    table = np.array([(i/255.0)**(1/gamma)*255.0 for i in np.arange(0,256)]).astype('uint8')

    L , A, B = cv.split(img_LAB)
    L_new = cv.LUT(L,table)
    img_transformed = cv.merge((L_new,A,B))
    fig , ax =  plt.subplots()
    ax.set_xticks([]), ax.set_yticks([])
    ax.imshow(cv.cvtColor(img_transformed,cv.COLOR_LAB2RGB))
    ax.set_title("      $\gamma$ ={} Corrected Image".format(gamma))
    plt.style.use('seaborn-dark')
```

When applying gamma correction to the L plane, it can be observed that the resultant image gets darker when the gamma value is less than 1 and the resultant image gets brighter when the gamma value is greater than 1. This behavior can be observed in the histograms as well. It is observed that the image gets darker and brighter with decreasing and increasing gamma values.

Question 4 – Histogram equalization

Before Histogram equalization	After Histogram equalization
	
	

The main purpose of carrying out histogram equalization is to get a uniform distribution of intensities across the image. From the above result comparison, we can observe that this is achieved. After equalizing the darkness of the image is reduced.

Code for Histogram equalization

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('shells.png',cv.IMREAD_GRAYSCALE)
hist,bins = np.histogram(img.ravel(),256,[0,256])
plt.style.use('default')

fig , ax =  plt.subplots()


ax.imshow(img, cmap = 'gray' , vmin=0 , vmax =255)

fig , ax =  plt.subplots()

ax.set_xlim([0,256])
ax.plot(hist)
ax.set_title('Histogram of the Original Image')
```

```
def HistEqualize(img):

    hist = np.zeros((1,256))
    for i in range(0,256):
        hist[0][i] = np.count_nonzero(img==i)

    rows , cols = img.shape

    equalized_histogram = np.zeros((1,256))

    for i in range(0,256):
        S = (255/(rows*cols))*np.sum(hist[:,:i+1])
        equalized_histogram[0][i] = round(S)

    hist_equalized_img = cv.LUT(img, equalized_histogram)

    hist_eqalized = np.zeros((1,256))

    for i in range(0,256):
        hist_eqalized[0][i] = np.count_nonzero(hist_equalized_img==i)


    return hist_eqalized[0] , hist_equalized_img

fig , ax =  plt.subplots()

eq , eq_img = HistEqualize(img)
ax.plot(eq)
ax.set_title('Equalized Histogram ')

fig , ax =  plt.subplots()
ax.imshow(eq_img, cmap = 'gray', vmin =0, vmax =255)

plt.show()
```

Question 5 (Images are zoomed and cropped for ease of comparison. All the relevant full images can be found in the GitHub)

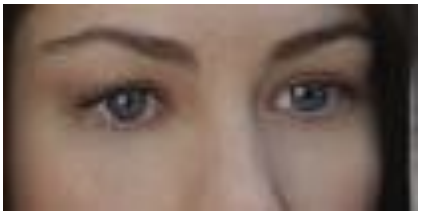
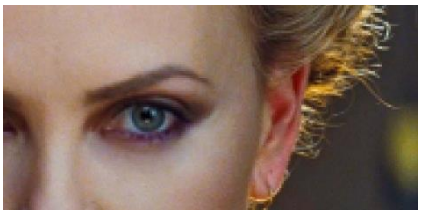
Original image (cropped for comparison)	Zoomed nearest neighbor (cropped for comparison)	Zoomed bilinear interpolation (cropped for comparison)
		
		
		

Image	Nearest neighbor			Bilinear Interpolation		
	B_SSD	G_SSD	R_SSD	B_SSD	G_SSD	R_SSD
Im01	13830386433	21236254210	26235176489	13924673211	21400390920	26505871629
Im02	7078852028	7286550162	7507441830	6940779584	7117796790	7385588556
Im03	16030818348	14403656397	12083672322	15914525918	14190816245	11872433638

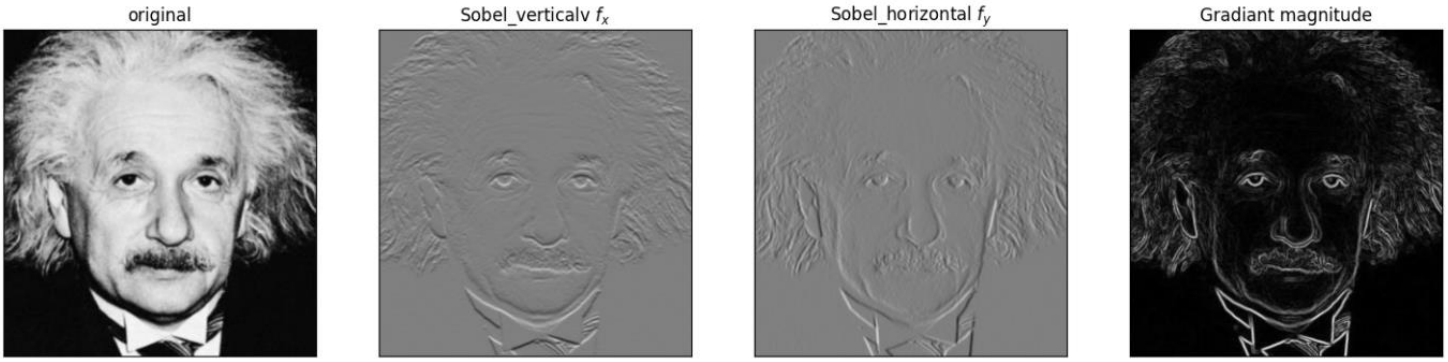
Nearest Neighbor – One-pixel value is repeated according to zoom size given image and zoom factor. When implementing in python first the image is upsampled. USandNN function is defined for this and it returns the zoomed image.

Bilinear Interpolation – First the image is upsampled using a separate up sample function. Four nearest neighbors are used to determine the intensity of a given location. The remaining pixels are filled by the Interpolate function. Since the image is padded in upsampling, we need to remove image padding. removePadding function is used for this.

Comparison – Bilinear interpolation should be better as its pixels' value depends on 4 pixels from the original image. But observing the SSD values from the above table, we can see the error value in nearest neighbor method is less. It shows the nearest neighbor is more similar. Even though SSD shows that the nearest neighbor is better, by observing and comparing the image results, we could still see that bilinear interpolation is better.

Question 6 – Sobel filtering

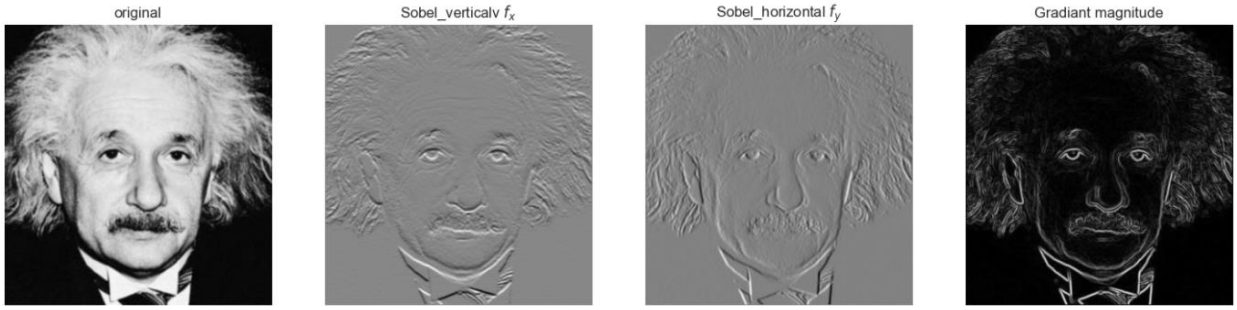
a)




```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
im = cv.imread('einstein.png', cv.IMREAD_GRAYSCALE).astype(np.float32)
assert im is not None
sobel_v = np.array([(-1, -2, -1), (0, 0, 0), (1, 2, 1)] , dtype=np.float32)
im_x = cv.filter2D(im,-1, sobel_v )
sobel_h = np.array([(-1, 0, 1), (-2, 0, 2), (-1, 0, 1)], dtype=np.float32)
im_y = cv.filter2D(im,-1, sobel_h )
grad_mag = np.sqrt ((im_x**2)+(im_y**2))
fig, ax = plt.subplots(1,4, figsize=(18,16))
ax[0].imshow(im,cmap='gray', vmin = 0, vmax=255)
ax[0].set_title("original")
ax[1].imshow(im_x ,cmap='gray', vmin = -1020, vmax=1020)
ax[1].set_title("Sobel_verticalv $f_x$")
ax[2].imshow(im_y,cmap='gray', vmin = -1020, vmax=1020)
ax[2].set_title("Sobel_horizontal $f_y$")
ax[3].imshow(grad_mag,cmap='gray', )
ax[3].set_title("Gradiant magnitude")
for i in range (4):
    ax[i].set_xticks([]), ax[i].set_yticks([])
```

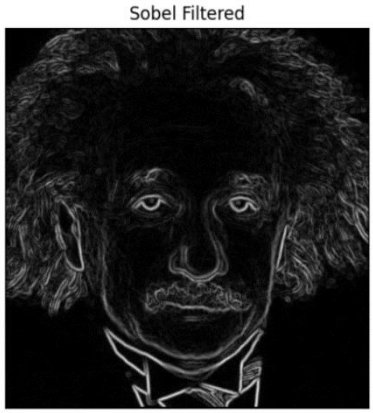
b)

```
def filter(image, kernel):                          # function defined insted of filter2D function
    k_rows, k_cols  = kernel.shape
    k_r, k_c = math.floor(k_rows/2), math.floor(k_cols/2)
    h, w = image.shape
    float_img = cv.normalize(image.astype('float'), None, 0.0, 1.0, cv.NORM_MINMAX)
    result_img = np.zeros(image.shape, 'float')

    for i in range(k_r, h - k_c):
        for j in range(k_c, w - k_c):
            result_img[i,j] = np.matmul(float_img[i-k_r:i + k_r + 1, j - k_c : j + k_c + 1].flatten(), kernel.flatten())
    return result_img
```



c)

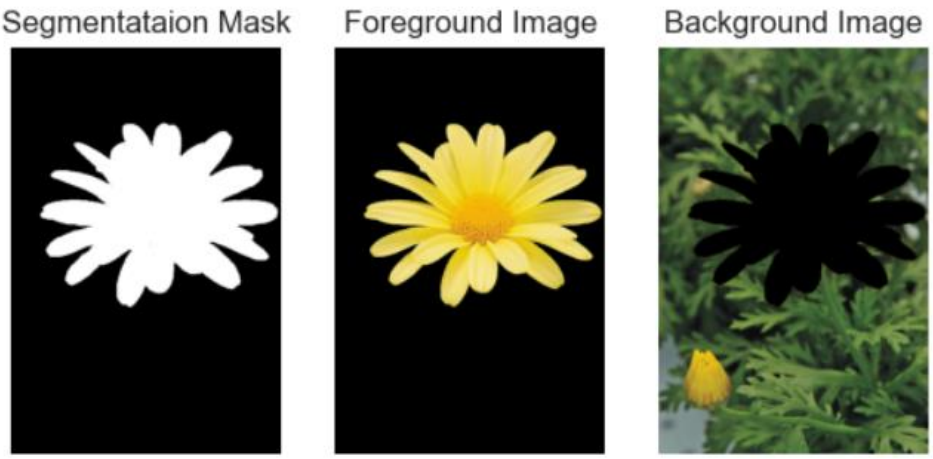
```
A = np.array([[1],[2], [1]], dtype=np.float32)
B = np.array([[1,0,-1]], dtype=np.float32)

C = np.matmul(A,B)  # using the given property

new_sobel_h = -1*C
new_sobel_v = new_sobel_h.transpose()

im_x = filter(im, new_sobel_v )
im_y = filter(im, new_sobel_h )
grad_mag = np.sqrt ((im_x**2)+(im_y**2))
fig, ax = plt.subplots()

ax.imshow(grad_mag,cmap='gray')
ax.set_title('Sobel Filtered')
```



Filter2D function is used in part a. A separate filter function is defined for part b. and it is used is part c as well.

Question 7

a)




```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
im = cv.imread('daisy.jpg', cv.IMREAD_COLOR)
mask = np.zeros(im.shape[:2],np.uint8)
bg_model = np.zeros((1,65),np.float64)
fg_model = np.zeros((1,65),np.float64)

rect = (25,125,530,450) # reactangle
mask , bg_model, fg_model = cv.grabCut(im, mask, rect,bg_model, fg_model ,3,cv.GC_INIT_WITH_RECT)
output_mask = np.where((mask==cv.GC_BGD) | (mask==cv.GC_PR_BGD),0,1)
output_mask = (output_mask*255).astype("uint8")

output_img = cv.bitwise_and(im,im, mask=output_mask)
back_img = im - output_img
fig , ax = plt.subplots(1,3)
ax[0].imshow(cv.cvtColor(output_mask, cv.COLOR_BGR2RGB))
ax[0].set_xticks([]),ax[0].set_yticks([])
ax[0].set_title('Segmentataion Mask')

ax[1].imshow(cv.cvtColor(output_img, cv.COLOR_BGR2RGB))
ax[1].set_xticks([]),ax[1].set_yticks([])
ax[1].set_title('Foreground Image')

ax[2].imshow(cv.cvtColor(back_img, cv.COLOR_BGR2RGB))
ax[2].set_xticks([]),ax[2].set_yticks([])
ax[2].set_title('Background Image')
```

b)


```
back_img_blured = cv.GaussianBlur(back_img,(9,9),10)
enhanced_img = output_img + back_img_blured
fig , ax = plt.subplots(1,2,figsize= (10,5))

ax[0].imshow(cv.cvtColor(im, cv.COLOR_BGR2RGB))
ax[0].set_xticks([]),ax[0].set_yticks([])
ax[0].set_title('Original Image')

ax[1].imshow(cv.cvtColor(enhanced_img, cv.COLOR_BGR2RGB))
ax[1].set_xticks([]),ax[1].set_yticks([])
ax[1].set_title('Enhanced Image')

plt.show()
```

Background is gaussian blurred and added to the output image to produce the enhanced image.

c)

Original image Enhanced image

It can be seen that the background just beyond the edge of the flower is dark.
The weights of pixels in edges are determined by the probability of a pixel being foreground or background. Since there is a large difference in pixel color the edge between them gets a low weight. Therefore the background just beyond the edge of the flower is dark.

References

1)https://www.researchgate.net/publication/301443589_Template_Matching_using_Sum_of_Squared_Difference_and_Normalized_Cross_Correlation
2) https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html
3) https://answers.opencv.org/question/105994/blurred-mask-on-image/