EN2550 Assignment 1 on Intensity Transformations and Neighbourhood Filtering

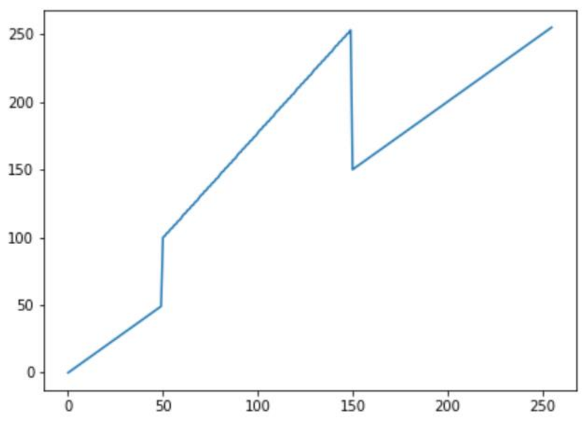
Index: 190018V

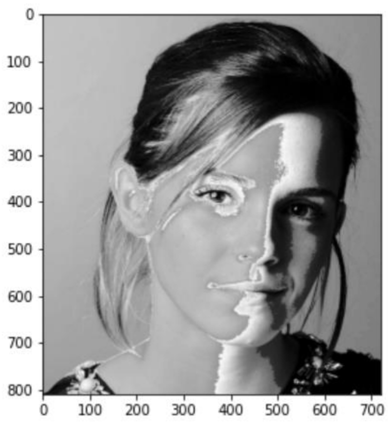
Name: Abeywickrama K.C.S.

GitHub Link: https://github.com/KCSAbeywickrama/EN2550-Excercises/tree/master/Assignment_01

1)

```
f=cv.imread('emma_gray.jpg',cv.IMREAD_GRAYSCALE)
t=np.zeros(255-0+1).astype(np.uint8)
r1=np.linspace(0,50,(50+1-0)).astype(np.uint8)
r2=np.linspace(100,255,(150+1-50)).astype(np.uint8)
r3=np.linspace(150,255,(255+1-150)).astype(np.uint8)
t[0:50+1]=r1
t[50:150+1]=r2
t[150:255+1]=r3
g=cv.LUT(f,t)
```

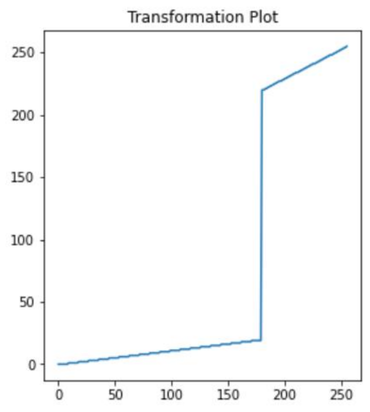


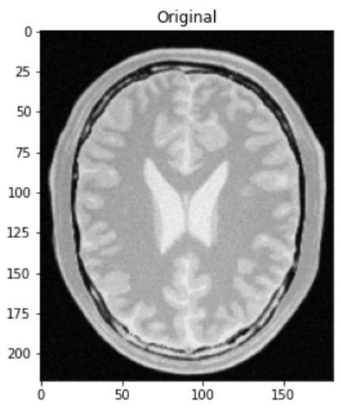


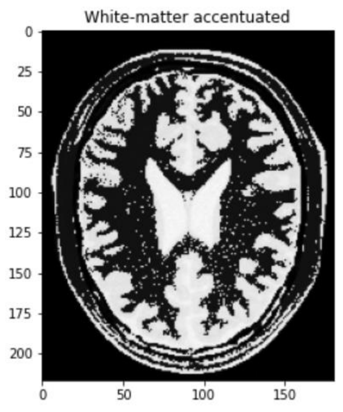
By this transformation, pixels with 50-150 intensities have increased while other keep as it is. As a result, dark side of the face has turn into white

2) (a)

```
f=cv.imread('brain_proton_density_slice.png',cv.IMREAD_GRAYSCALE)
a=180
t=np.zeros(256).astype(np.uint8)
r1=np.linspace(0,20,(a+1)).astype(np.uint8)
r2=np.linspace(220,255,(255+1-a)).astype(np.uint8)
t[0:a+1]=r1
t[a:255+1]=r2
g=cv.LUT(f,t)
```



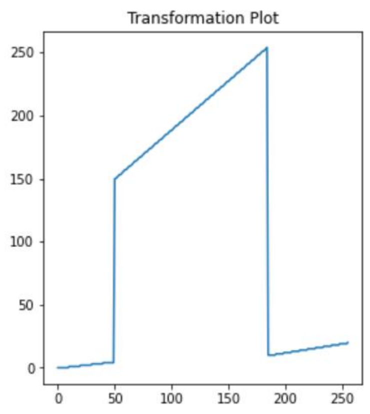


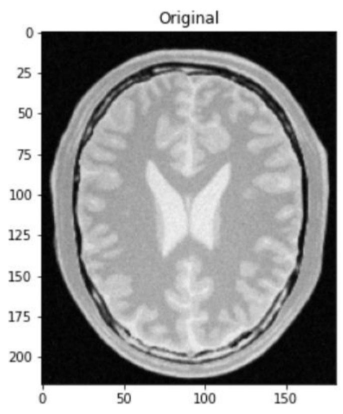


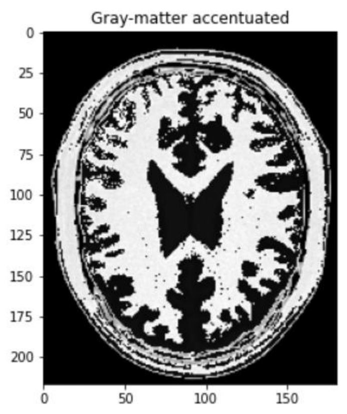
To accentuate white matter, increase the intensity of bright pixels more while decreasing the intensity of other dark pixels. The turning point of intensity transformation curve has found by trial & error.

2 (b)

```
f=cv.imread('brain_proton_density_slice.png',cv.IMREAD_GRAYSCALE)
a1,a2=50,185
t=np.zeros(256).astype(np.uint8)
r1=np.linspace(0,5,(a1+1-0)).astype(np.uint8)
r2=np.linspace(150,255,(a2+1-a1)).astype(np.uint8)
r3=np.linspace(10,20,(255+1-a2)).astype(np.uint8)
t[0:a1+1]=r1
t[a1:a2+1]=r2
t[a2:255+1]=r3
```



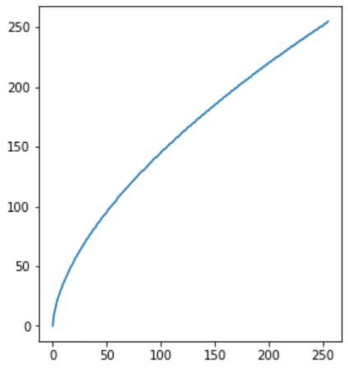


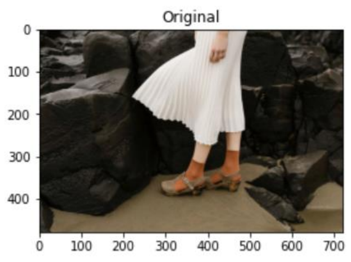


Pixels in 50-185 intensity range has considered as Gray matter colour. The range was found by trial & error. Intensity of Gray pixels are increased while intensity of bright pixels and dark pixels are decreased using the above transformation curve.

3 (a)

```python
f=cv.imread('highlights_and_shadows.jpg')
lab_org=cv.cvtColor(f,cv.COLOR_BGR2Lab)
gamma=0.6
t=np.array([(p/255)**gamma*255 for p in range(0,256)]).astype(np.uint8)

lab_t=np.copy(lab_org)
lab_t[:,:,0]=t[lab_t[:,:,0]]
```





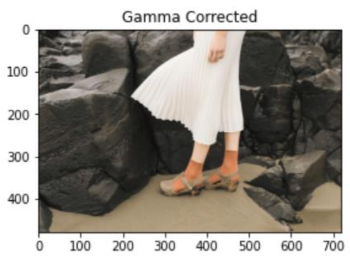
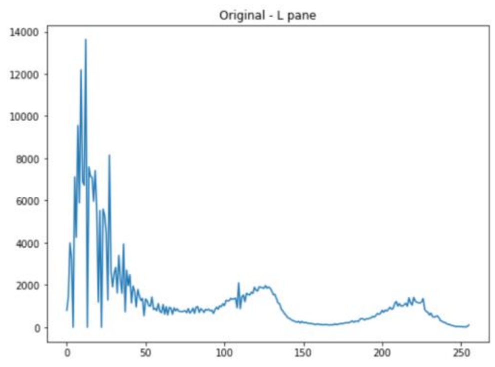
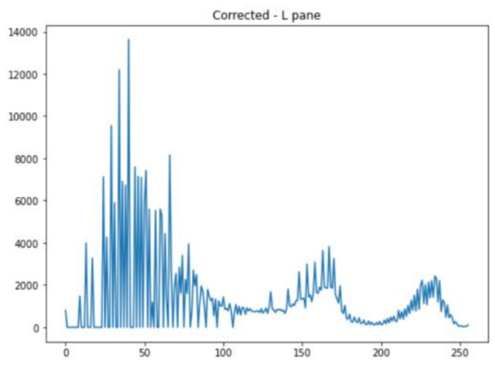


Image ‘f’ which is in BGR space has converted to Lab space 1st. Then apply a gamma correction like in the plot only to the L plane. As a result, lightness has increased in dark pixels so dark area has lightened in the gamma corrected image.

3 (b)

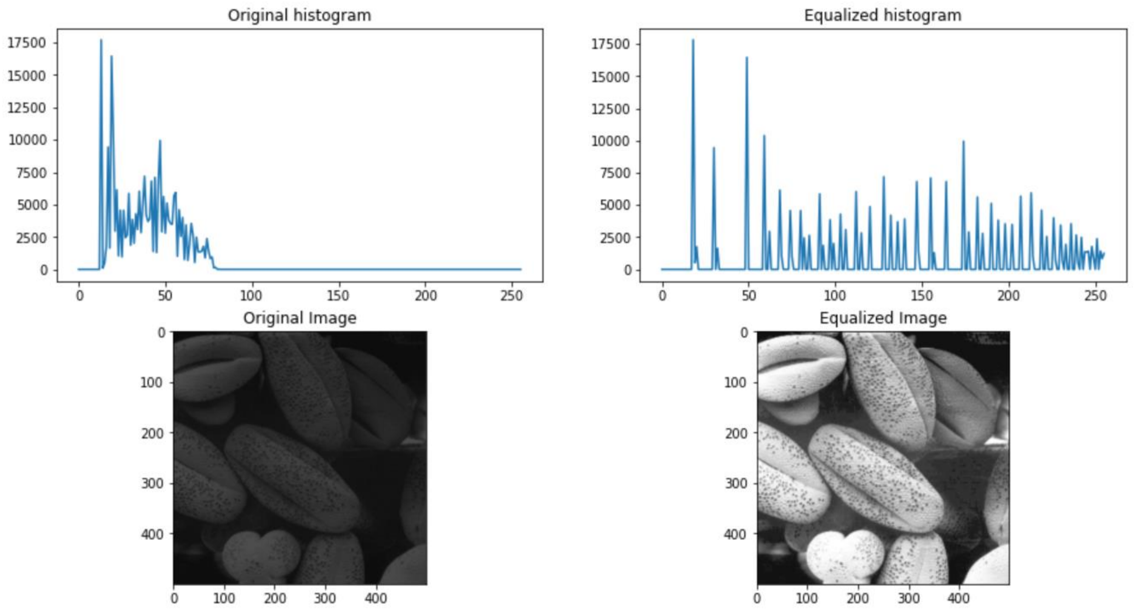




Increasement of lightness of dark pixels is represented by right shift of the histogram

4)

```python
f=cv.imread('shells.png',cv.IMREAD_GRAYSCALE)
hist_org=cv.calcHist([f],[0],None,[256],[0,256])
cumsum=hist_org.cumsum()
t=np.round(cumsum*255/f.size).astype(np.uint8)
g=t[f]
hist_eq=cv.calcHist([g],[0],None,[256],[0,256])
```



After equalizing dark areas has brightened & image is clearer than before.

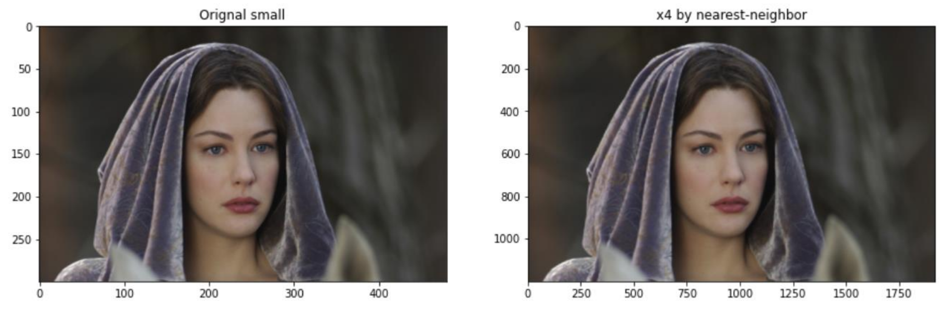
5) (a) nearest-neighbour method

```
def zoomNearestNeighbor(im,scale):

    rows=int(scale*im.shape[0])
    cols=int(scale*im.shape[1])
    chnls=im.shape[2]
    zoomed=np.zeros((rows,cols,chnls),dtype=im.dtype)

    for i in range(rows):
        for j in range(cols):
            im_i=round(i/scale)
            im_j=round(j/scale)
            if(im_i>=im.shape[0]): im_i=im.shape[0]-1
            if(im_j>=im.shape[1]): im_j=im.shape[1]-1
            zoomed[i,j]=im[im_i,im_j]

    return zoomed
```

```
im01.png: noramalized sum of squared difference = 0.022723643109202385
```

Orignal small
x4 by nearest-neighbor

```
im02.png: noramalized sum of squared difference = 0.010222389362752438
```

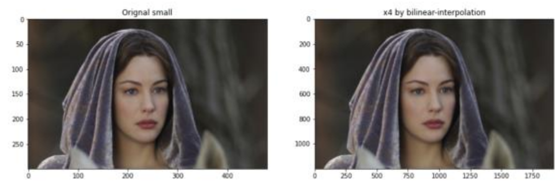


5) (b) bilinear-interpolation

```
def zoomBilinearInterpolation(im,scale):

    rows=int(scale*im.shape[0])
    cols=int(scale*im.shape[1])
    chnls=im.shape[2]
    zoomed=np.zeros((rows,cols,chnls),dtype=im.dtype)
    for m in range(rows):
        for n in range(cols):
                i,j=m/scale,n/scale
                i0,j0=int(i),int(j)
                i1,j1=i0+1,j0+1
                di0,dj0=i-i0,j-j0
                di1,dj1=1-di0,1-dj0

                if(i1>=im.shape[0]): i1=im.shape[0]-1
                if(j1>=im.shape[1]): j1=im.shape[1]-1

                k0=im[i0,j0]*di1+im[i1,j0]*di0
                k1=im[i0,j1]*di1+im[i1,j1]*di0
                k=k0*dj1+k1*dj0
                zoomed[m,n]=k
    return zoomed
```

```
im01.png: noramalized sum of squared difference = 0.017942212522029877
```

```
im02.png: noramalized sum of squared difference = 0.007791651412844658
```



Since matplotlib autofit the image displayed in same size. But from axis coordinates we can see the size of image has increased.

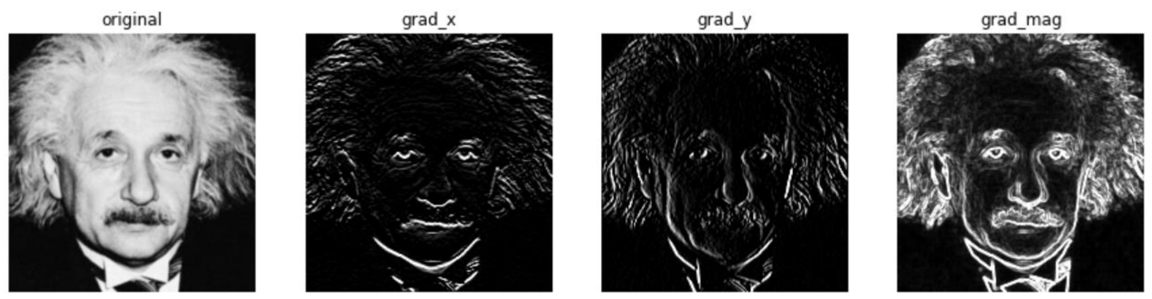
normalized sum of squared difference (NSSD) comparison

	nearest-neighbour	bilinear interpolation
im01	0.0227	0.0179
im02	0.0102	0.0078
im03	0.0147	0.0112

NSSD values is lower in bilinear interpolation method it means its more accurate than the nearest-neighbour method. But execution time is high

6 (a)

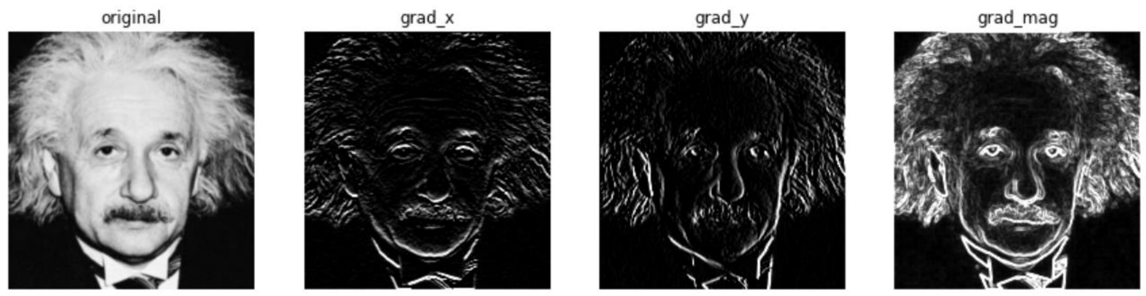
```
f=cv.imread('einstein.png',cv.IMREAD_GRAYSCALE).astype(np.float32)
sobel_v=np.array([[-1,-2,-1],[0,0,0],[1,2,1]],dtype=np.float32)
f_x=cv.filter2D(f,-1,sobel_v)
sobel_h=np.array([[-1,0,1],[-2,0,2],[-1,0,1]],
dtype=np.float32)
f_y=cv.filter2D(f,-1,sobel_h)
grad_mag=np.sqrt(f_x**2+f_y**2)
```



By convolving Sobel horizontal & vertical kernels gradient of image is obtained. By getting the gradient magnitude using gradient x & gradient y we can see all edges are emphasized

6(b)

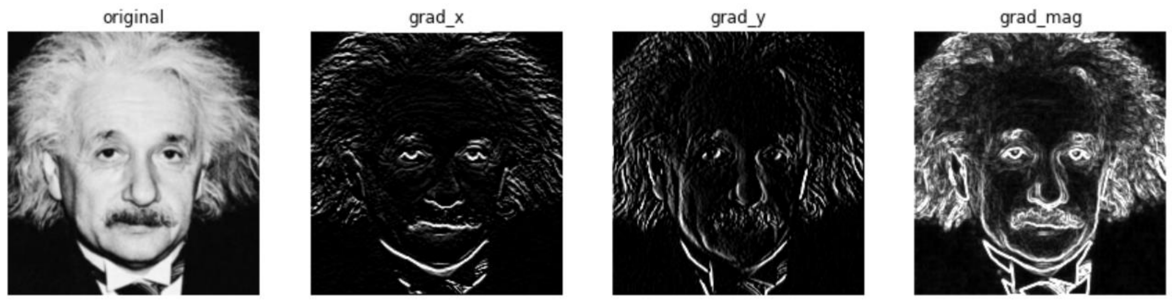
```
def convolve(img,kernal):
    kw=kernal.shape[0]
    kh=kernal.shape[1]
    kernal180=kernal[::-1,::-1]
    padded=np.pad(img,((kw//2,kw//2),(kh//2,kh//2)))
    res=np.zeros(img.shape).astype(np.float32)
    for m in range(img.shape[0]):
        for n in range(img.shape[1]):
            res[m,n]=np.sum(padded[m:m+kw,n:n+kh]*kernal180)

    return res
```



similar result can be obtained using this custom convolve code.

6(c)

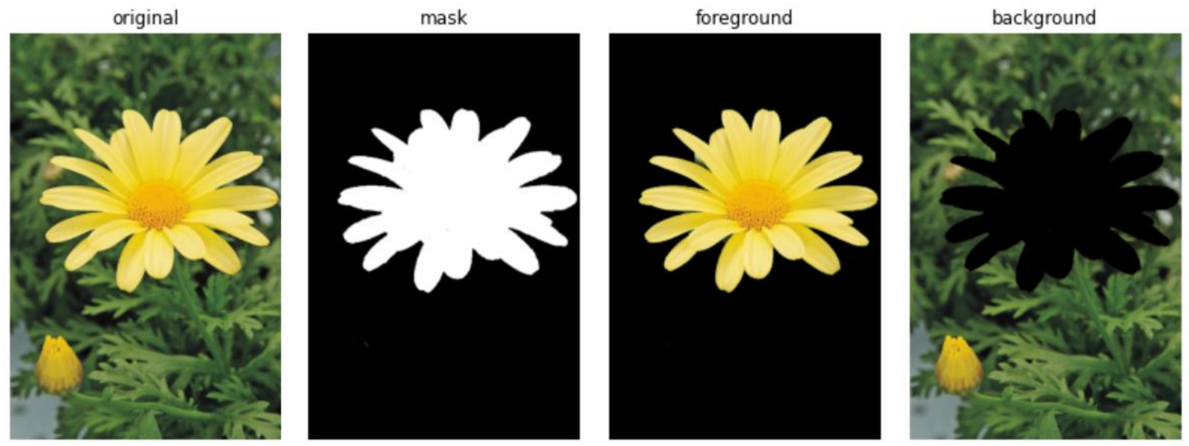
```
sobel_h1=np.array([[1],[2],[1]])
sobel_h2=np.array([[1,0,-1]])

sobel_v1=np.array([[1],[0],[-1]])
sobel_v2=np.array([[1,2,1]])


f_y=convolve(f,sobel_h1)
f_y=convolve(f_y,sobel_h2)

f_x=convolve(f,sobel_v1)
f_x=convolve(f_x,sobel_v2)
```



Similar result can be obtained by convolving image in 2 times by using the property which has given in the question.

7) (a)

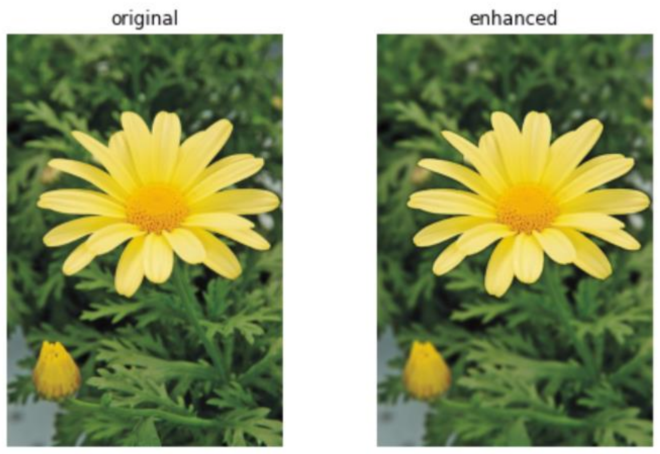
```
rect = (56,150,560,500)
cv.grabCut(img,mask,rect,bgdModel,fgdModel,2,cv.GC_INIT_WITH_RECT)
mask = np.where((mask==2)|(mask==0),0,1).astype('uint8')

imgFg = img*mask[:,:,np.newaxis]
imgBg=img-imgFg
```



We can obtain foreground mask from grabCut() method & then applying it to the original image we can filter out the foreground. By subtracting foreground from the original we can obtain background

7) (b)

```
imgBgBlur=cv.GaussianBlur(imgBg,(9,9),12)
imgEnhnsd=np.bitwise_or(imgFg,imgBgBlur)
```



Background has blurred using GaussianBlur() method. Then foreground area of the blurred background replaced by the extracted foreground image using bitwise_or() method

7 (c)

When we blur the background image, foreground area of background image is existed as black. So, in the blurred image pixels near edge become darker due to mixing with that black area when blurring. So, edge of the flower quite dark in the enhanced image