EN3160 Intensity Transformations and Neighborhood Filtering- K.N.A.L. – 200164H

Question 1 – Implementing the intensity transformation on an image **GitHub link: Link to GitHub**

```
img=cv.imread("C:\\Users\\leonf\Music\\image_processing\\emma.jpg",0)
img=cv.cvtColor(img,cv.COLOR_BGR2RGB)
fig,im=plt.subplots(1,2,figsize=(7,7))
arr1=np.linspace(0,50,51).astype(np.uint8)
arr2=np.linspace(101,255,100).astype(np.uint8)
arr3=np.linspace(150,255,105).astype(np.uint8)
arr=np.concatenate((arr1,arr2))
arr=np.concatenate((arr,arr3))
```

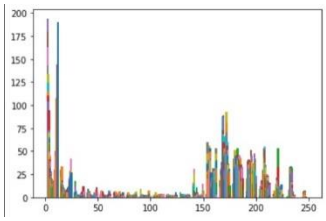
The LUT function in openCV is used to get the transformation applied to the image given

```
img_new=cv.LUT(img,arr)
```

The darkest and brightest values of the provided intensity transformation remain unaltered. (0-50 and 150-255). The transition affects the medium intensity range of 50-150. As a result, the darkest parts, such as hair and pupils, and the lightest areas, such as skin, stay unchanged, while sections with medium intensities have grown lighter.(For example, the shaded part of the face)

Question 2 – Implementing the intensity transformation to enhance white matter and grey matter.

The color distribution of the image was better understood through a histogram, and the transformation was then applied.



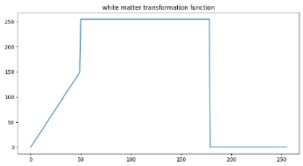
For white matter

```
arr1=np.linspace(0,10,50).astype(np.uint8)
arr2=np.linspace(0,10,129).astype(np.uint8)
arr3=np.linspace(255,255,77).astype(np.uint8)
arr=np.concatenate((arr1,arr2))
fin_arr=np.concatenate((arr,arr3))
img_new=cv.LUT(img,fin_arr)
```

For gray matter

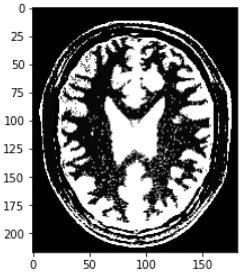
```
arr1=np.linspace(0,150,50).astype(np.uint8)
arr2=np.linspace(255,255,129).astype(np.uint8)
arr3=np.linspace(0,0,77).astype(np.uint8)
arr=np.concatenate((arr1,arr2))
fin_arr=np.concatenate((arr,arr3))
img_new=cv.LUT(img,fin_arr)
```

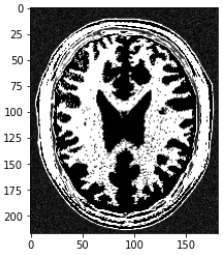
The relevant transformations increasing the intensities of grey and white matter were thereby generated, and those transformations were applied to the image next.



white matter enhanced

gray matter enhanced.





Question 3 – Gemma Correction to L Plane

(a). We should apply Gemma correction to the L plane of L*a*b color space.

```
img=cv.imread("C:\\Users\\leonf\Music\\image_processing\\highlights_and_shadows.jpg",1)
img=cv.cvtColor(img,cv.COLOR_BGR2LAB)
```

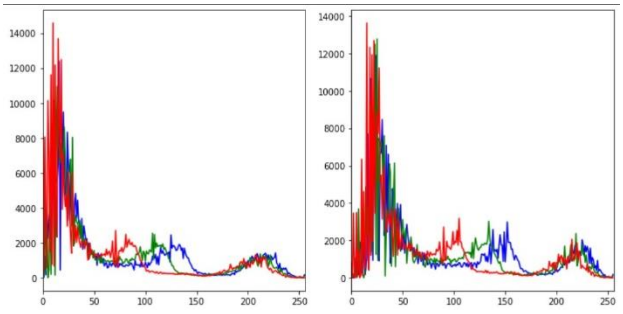
Here, I open the image in OpenCV as a color image and convert it to L*A*B color space.

Next, The gamma correction was applied to the L plane by splitting it and setting gamma to 0.8. To make the more natural to our eye and to spread out the Histogram of the transformed image.

```
img_new=np.copy(img)
gamma=0.8
img_new[:,:,0]=(((img[:,:,0]/255)**gamma)*255).astype(np.uint8)
fig,ax=plt.subplots(1,2,figsize=(15,18))
img=cv.cvtColor(img,cv.COLOR_LAB2RGB)
img_new=cv.cvtColor(img_new,cv.COLOR_LAB2RGB)
ax[0].imshow(img)
ax[1].imshow(img_new)
```

(b). Next the histograms of the original image and the gamma-corrected image are generated using the OpenCV *calcHist* function

```
color=['b','g','r']
fig,ax=plt.subplots(1,2,figsize=(10,5))

for i,c in enumerate(color):
    hist=cv.calcHist([img],[i],None,[256],[0,256])
    ax[0].plot(hist,color=c)
    ax[0].set_xlim([0,256])

color=['b','g','r']
for i,c in enumerate(color):
    hist=cv.calcHist([img_new],[i],None,[256],[0,256])
    ax[1].plot(hist,color=c)
    ax[1].set_xlim([0,256])

plt.tight_layout()
plt.show()
```



Question 4– Increasing the vibrance by intensity transformation on an image.

(a).

```
spider_img=cv.imread("C:\\Users\\leonf\Music\\image_processing\\spider.png",1)
spider=cv.cvtColor(spider_img,cv.COLOR_BGR2RGB)
fig,ax=plt.subplots(1,2,figsize=(15,15))
ax[0].imshow(spider)
spider_img=cv.cvtColor(spider_img,cv.COLOR_BGR2HSV)
hue,saturation,value=cv.split(spider_img)
```

The image is opened in pyhon OpenCV in color and then converted to hsv color space, splitted each plane.

(b)

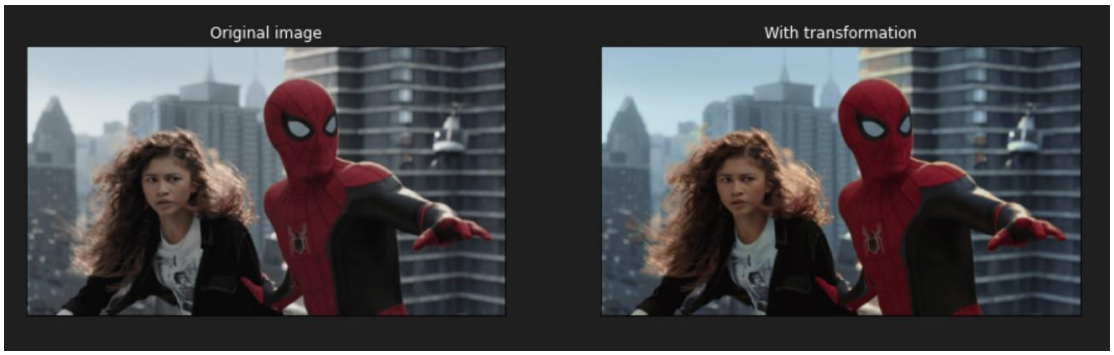
```
a=0.2
stan=70
saturation=(np.minimum(saturation+(a)*128*np.exp((-(saturation-128)**2)/(2*stan**2)),255)).astype(n
```

(c). In the original image the colors are little bit dull and so to make the image little bit saturated I apliead the a to 0.2 to make the image more visually pleasing.

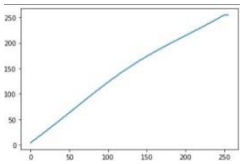
(d) Then the three planes are recombined using merge function

```
spider_img=cv.merge([hue,saturation,value])
```

(e) The image in hsv format was converted into RGB and displayed next using matplotlib.



The Intensity transformation



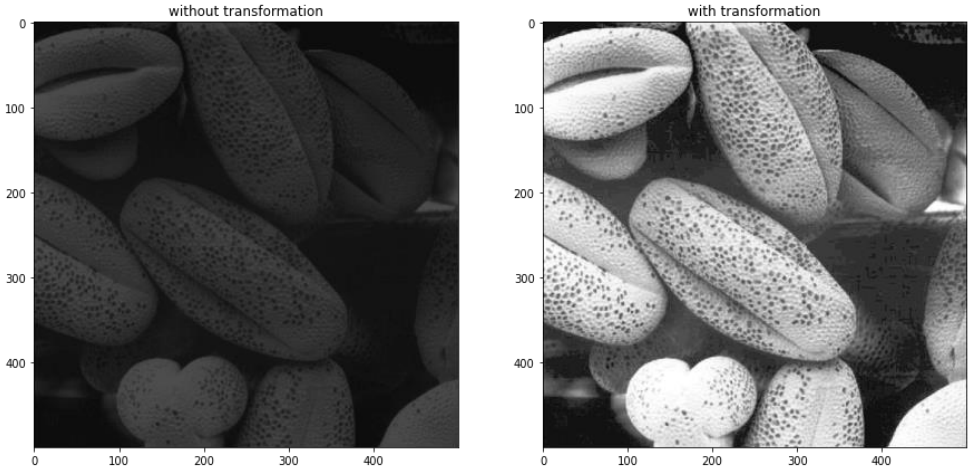
Question 5–Histogram Equalization.

The primary purpose of histogram equalization is to efficiently distribute the concentrated portions of the histogram across the range 0-255.

I followed the following steps to equalize the histogram of the image given.

1. Created a NumPy array of size 256, initialized with all zeros. This will be used to store of the number of pixel values in the input image.
2. Another NumPy array of size 256, with zeros initialized, it will be used to hold pixel count cumulative sums.
3. The 2D image was converted into a 1D array, for easier processing using. reshape function.
4. Using NumPy's unique function, I found the unique pixel values and counts in the 1D array from 3.
5. Iterated over the distinct pixel values and their corresponding counts. Updated the corresponding index in the arr array with the count for each unique pixel value. The frequency of each pixel value in the image is effectively filled into the arr array.
6. The count of the first pixel value was used to initialize the first element of the sum_arr array, effectively setting the initial cumulative sum.
7. The cumulative sum was then calculated.
8. The cumulative sums were scaled and rounded to fit the range of pixel values (0-255).
9. Using OpenCV's LUT (lookup table) function, apply the calculated transformation to the original image img
10. Last the images were displayed.

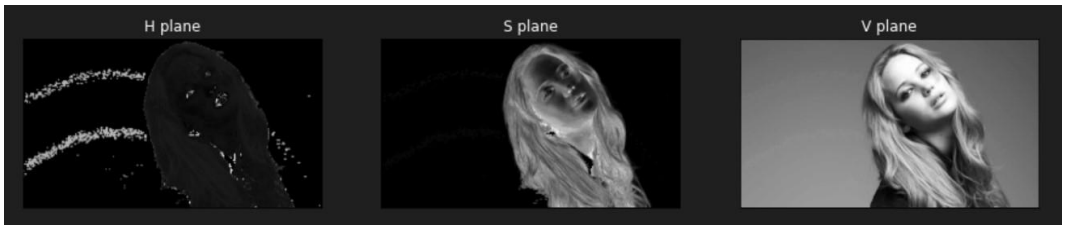
```
arr=np.zeros((256))
sum_arr=np.zeros((256))
img_new=np.reshape(img,[250000,])
ele,count=np.unique(img_new,return_counts=True)
for i,j in enumerate(ele):
    arr[j]=count[i]
sum_arr[0]=arr[0]
for k in range(1,256):
    sum_arr[k]=arr[k]+sum_arr[k-1]

sum_arr=np.round((sum_arr*255)/250000)
img_new=cv.LUT(img,sum_arr)
```



Question 6–Histogram Equalization Foreground.

(a) First the image was opened and then split into hue, saturation, and values and then displayed using matplotlib.

```
jen_img=cv.imread("C:\\Users\\leonf\Music\\image_processing\\jeniffer.jpg",1)
jen_img=cv.cvtColor(img,cv.COLOR_BGR2HSV)
#jen_img=cv.cvtColor(jen_img,cv.COLOR_RGB2HSV)
hue,saturation,value=cv.split(jen_img)
```

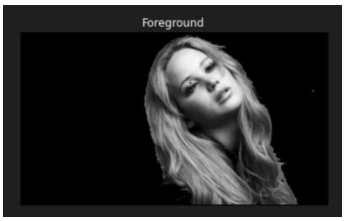


(b) Here saturation plane is selected to threshold and exctract the foreground . Then applying openCV threshold function to the saturation plane a binary mask is created .

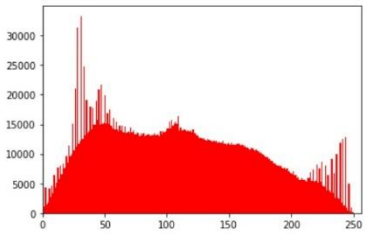
```
ret, mask = cv.threshold(saturation, 11, 1, cv.THRESH_BINARY)
```

(c) then using OpenCV bitwise_and fuction the foreground is extracted and then the Histogram is drawn.

```
img2=cv.bitwise_and(jen_img,jen_img,mask=mask)
```




```
hist,bins=np.histogram(img2.ravel(),256,[0,256])
plt.hist(img2.flatten(),256,[1,256],color='r')
plt.xlim([0,256])
hist[0] = 0
```

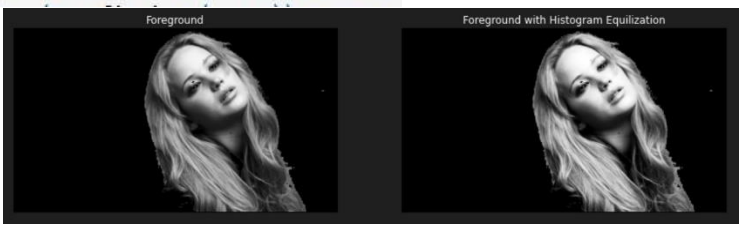


(d) Cumulative sum is calculated using np.cumsum

```
hist1=hist.cumsum()
```

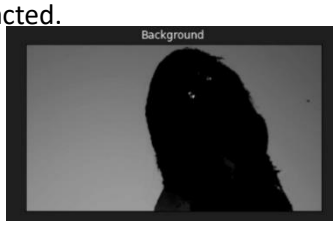
(e)

```
norm_hist1=hist1*(255/(hist.sum()))
img_new=cv.LUT(img2,norm_hist1).astype(np.uint8)
```

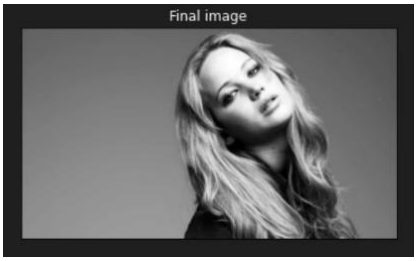


(f) First background bit mask was obtained by inversing the bit mask of the Foreground. Then using the bitwise_and operation with the original image the Background was extracted.

```
mask=1-mask
img_back=cv.bitwise_and(jen_img,jen_img,mask=mask)
```



(g) Finally the Equalized Foreground and the Background was added together to obtain the final image



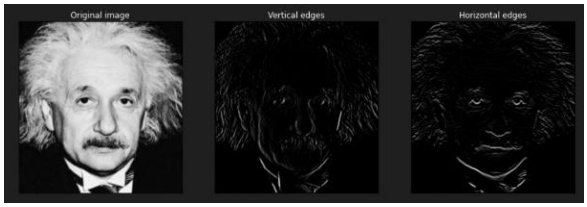
Question 7– Filtering with the Sobel operator can compute the gradient.

(a) filter2D to Sobel filter the image

```
img=cv.imread("C:\\Users\\leonf\Music\\image_processing\\einstein.png",0)
ker1=np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
ker2=np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
img1=cv.filter2D(img,-1,ker1)
img2=cv.filter2D(img,-1,ker2)
```

(b) own code to Sobel filter the image.

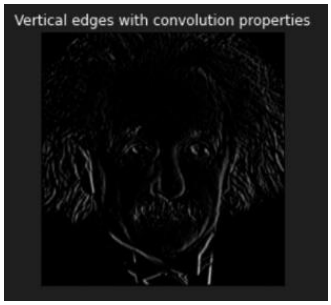
```
def conv(img,kernel):
    assert kernel.shape[0]%2==1 and kernel.shape[1]%2==1
    height=img.shape[0]
    width=img.shape[1]
    ker_hei,ker_wid=math.floor(kernel.shape[0]/2),math.floor(kernel.shape[1]/2)
    image_float=cv.normalize(img.astype(float),None,0.0,1.0,cv. NORM_MINMAX)
    result=np.zeros(img.shape, float)
    for i in range(ker_hei,height-ker_hei):
        for j in range(ker_wid,width-ker_wid):
            result[i,j]=np.maximum(np.dot(image_float[i-ker_hei:i+ker_hei+1,j-ker_wid:j +ker_wid+1].flatten(),kernel.flatten()),0)

    return result
```



(c) Using the property

```
ker1=np.array([[1],[2],[1]])
ker2=np.array([[1,0,-1]])
sup1_img=conv(img,ker1)
sup2_img=conv(sup1_img,ker2)
```

$$\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} = \begin{bmatrix}1\\ 2\\ 1\end{bmatrix} * [1 \quad 0 \quad -1] \quad and \quad \begin{bmatrix}1 & 2 & -1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} = \begin{bmatrix}1\\ -0\\ 1\end{bmatrix} * [1 \quad 2 \quad 1]$$



Question 8–Zooming.

Zooming refers to increasing the number of pixels in an image so that you can see more detail when you zoom in. The small images provided were used to zoom in. They were compared to the provided zoomed images. The OpenCV function resize () was used as directed to zoom in on the images using two different algorithms.

1. Nearest Neighbor

```
height,width=org_small_img.shape[:2]

scaled_height=int(height*4)
scaled_width= int(width*4)

zoom_img = cv.resize(org_small_img, (scaled_width, scaled_height), interpolation = cv.INTER_NEAREST)
```

2. Bilinear Interpolation

```
height,width=org_small_img.shape[:2]

scaled_height=int(height*4)
scaled_width= int(width*4)

zoom_img = cv.resize(org_small_img, (scaled_width, scaled_height), interpolation = cv.INTER_LINEAR)
```

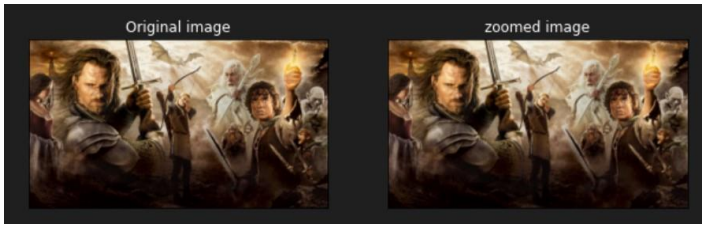
Computing mean square difference.

```
sq_dif=np.square(org_large_img-zoom_img)

mse=np.mean(sq_dif)

print(f"Normalize SSD :{mse}")
```

Carried out these for all images provided

Results: Nearest Neighbor

Image01



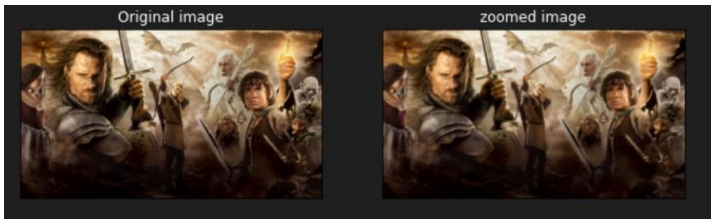

```
Normalize SSD :31.284316486625514
```

Image 02


```
Normalize SSD :11.902013310185184
```

Results: Bilinear Interpolation

Image 01




```
Normalize SSD :31.053094618055557
```

Image 02


```
Normalize SSD :10.682991753472223
```

When zoomed in using the nearest neighbor method, the image appears pixelated due to duplicate pixels. The image zoomed with bilinear interpolation, on the other hand, appears much smoother. The sum of squared differences is less in bilinear interpolation.

Question 9–Segmentation.

GrabCut uses image segmentation based on graph cuts to separate the required image. When we specify the range of pixels in the object to be extracted, it fairly accurately separates the image from the background.

a)

```
flower_img=cv.imread("C:\\Users\\leonf\Music\\image_processing\\flower.jpg",1)
flower_img=cv.cvtColor(flower_img,cv.COLOR_BGR2RGB)
mask=np.zeros(flower_img.shape[:2],np.uint8)

fground=np.zeros((1,65),np.float64)
bground=np.zeros((1,65),np.float64)

rect=(40,150,560,500)

(mask,fground,bground)=cv.grabCut(flower_img,mask,rect,fground,bground,5,mode=cv.GC_INIT_WITH_RECT)
```

```
mask1=np.where((mask==cv.GC_BGD)|(mask==cv.GC_PR_BGD),0,1)
mask1=(mask1*255).astype(np.uint8)

foreground=cv.bitwise_and(flower_img,flower_img,mask=mask1)
```

```
mask2=np.where(mask==cv.GC_PR_FGD,0,1)
mask2=(mask2*255).astype(np.uint8)

back=cv.bitwise_and(flower_img,flower_img,mask=mask2)
```

```
back=cv.GaussianBlur(back,(11,11),4)
fin_img=np.clip(cv.add(back,foreground),0,255)
plt.imshow(fin_img)
plt.xticks([])
plt.yticks([])
plt.title("Final Image",c="w")
```

b) When the Gaussian blur is applied to the background image, it blurs the margins of the flower's black section. As a convolutional filter is applied, the values of those pixels in those edges may rise beyond zero. When this blurred backdrop is combined with the foreground image, the pixel values around the edge will be higher than the foreground image's yellow color. They appear darker as a result.