Assignment 01 - Intensity Transformations and Neighborhood Filtering

Herath H.M.S.I. – Github Profile – 210218M

October 1, 2024

---

1 Question 01

Intensity transformation given is implemented by the following code

```
arr1 = np.linspace(0,50,50,dtype='uint8')
arr2 = np.linspace(100,255,100,dtype='uint8')
arr3 = np.linspace(150,255,106,dtype='uint8')
t = np.concatenate((arr1, arr2, arr3),dtype='uint8')  # Concatenate arrays
```

And it is applied on the grayscale image by,

```
g = cv2.LUT(f, t)                                     # Apply intensity transformation
```

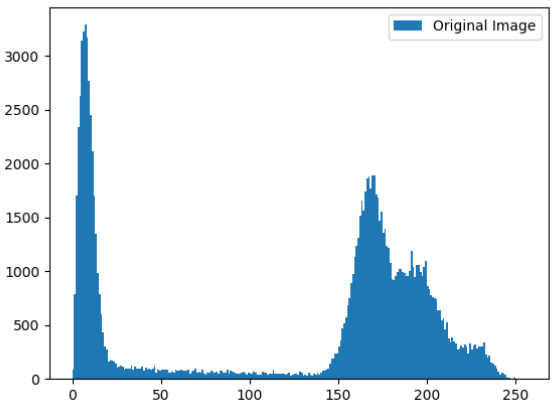
.

Image before and after applying transformation as follows,

Original Image

Image after applying intensity transformation

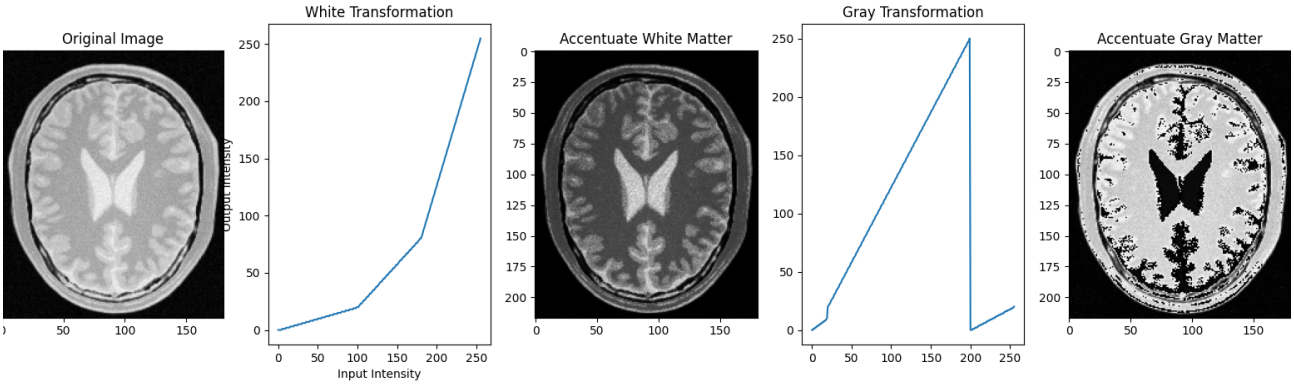
2 Question 02



Historgram of the Original Image

Historgram of the original image can be observed as in the left. Leftmost peak indicates the black pixels surrounded the brain. Next highest peak indicates gray and next higher peak indicates the white matter. So gradient of the intensity transformation adjusted as follows to clearly distinguish variations around those peaks.

```
garr1 = np.linspace(0,10,20,dtype='uint8')
garr2 = np.linspace(20,250,180,dtype='uint8')
garr3 = np.linspace(0,20,56,dtype='uint8')

# Lookup table for the transformation
t_wh = np.concatenate((warr1, warr2, warr3),dtype='uint8')  # Concatenate arrays
t_gr = np.concatenate((garr1, garr2, garr3),dtype='uint8')

# Apply intensity transformation
g = cv2.LUT(image, t_gr)          #Accrenuate gray matter
w = cv2.LUT(image, t_wh)          #Accrenuate white matter
```



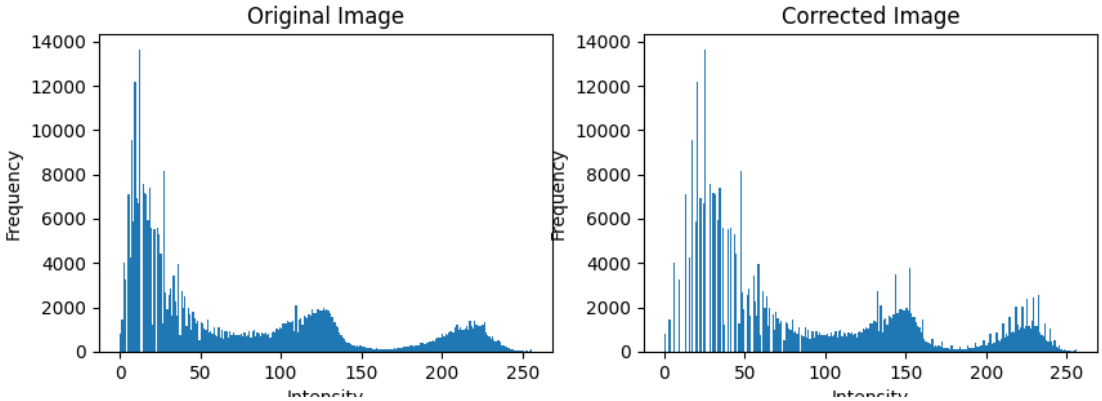
3 Question 03

```
image = cv2.imread("../images/highlights_and_shadows.jpg")  # As applying gamma correction
    on L plane
L, a, b = cv2.split(cv2.cvtColor(image, cv2.COLOR_BGR2LAB)) # Split the L*, a*, b* channels

gamma = 0.75
t = (np.arange(0,256)/255)**gamma * 255
```

```
L_corrected = cv2.LUT(L, t).astype(np.uint8)

corrected_imglab = cv2.merge([L_corrected, a, b])
```

Original Image

Image after Gamma Correction

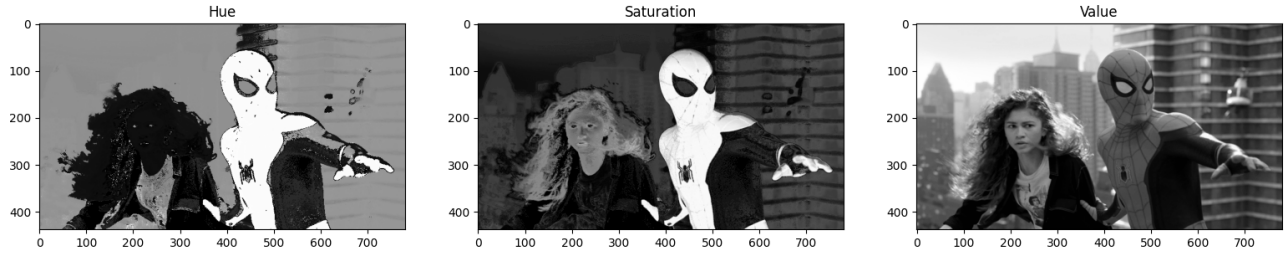


Historgrams of the Original and Gamma Corrected Image

4 Question 04

a)

```
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
h, s, v = cv2.split(hsv)
```



b)

```
a, sigma = 0.75, 70
x = np.arange(0, 256)
t = x + a * 128 * np.exp(-((x - 128) ** 2) / (2 * sigma ** 2))
t = np.clip(t, 0, 255)
Snew = cv2.LUT(s, t).astype(np.uint8)
t_img = cv2.merge([h, Snew, v])
```

c)

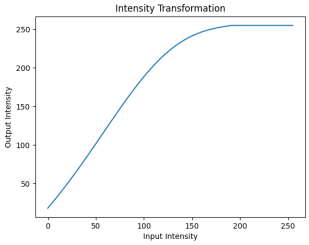
Value of a related to visually pleasing output is 0.75

d)

```
new_img = cv2.cvtColor(t_img, cv2.COLOR_HSV2BGR)
```

e)

Original Image

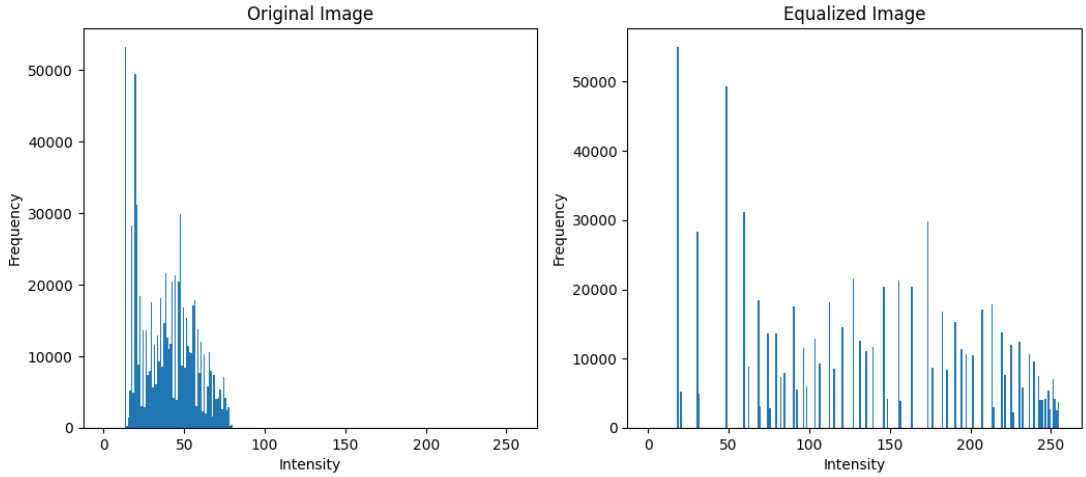


Intensity Transformation

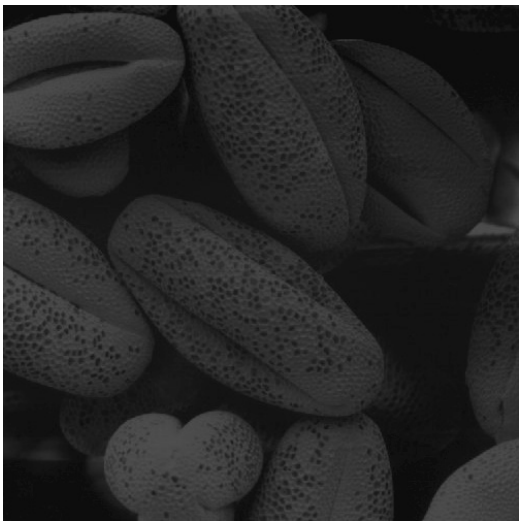
Vibrance Enhanced Image

5 Question 05

```
def historgram(image):
    '''  Takes image as the input and returns the historgram '''
    unique, counts = np.unique(image, return_counts=True)
    frequency = dict(zip(unique, counts))
    hist = np.array([frequency[i] if i in frequency.keys() else 0 for i in range(256)])
    return hist

def histEqualization(image):
    ''' Takes image to be equalized and returns the historgram equalized image '''
    cdf = np.cumsum(historgram(image))
    t = np.floor(cdf * 255/np.prod(image.shape)).astype(np.uint8)
    newimg = t[image]
    return newimg
```



Historgrams before and after normalization



Original Image

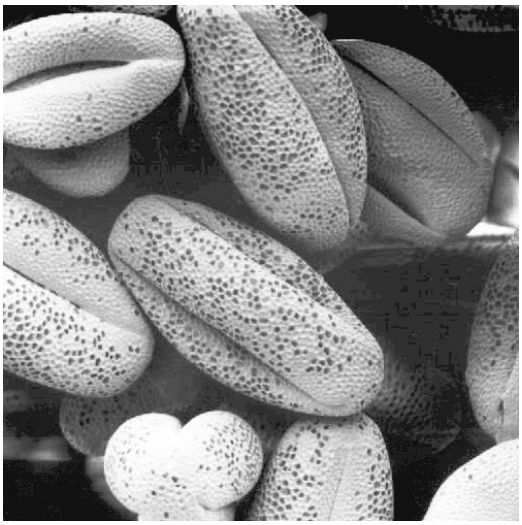


Image After Historgram Equalization

6 Question 06

a)

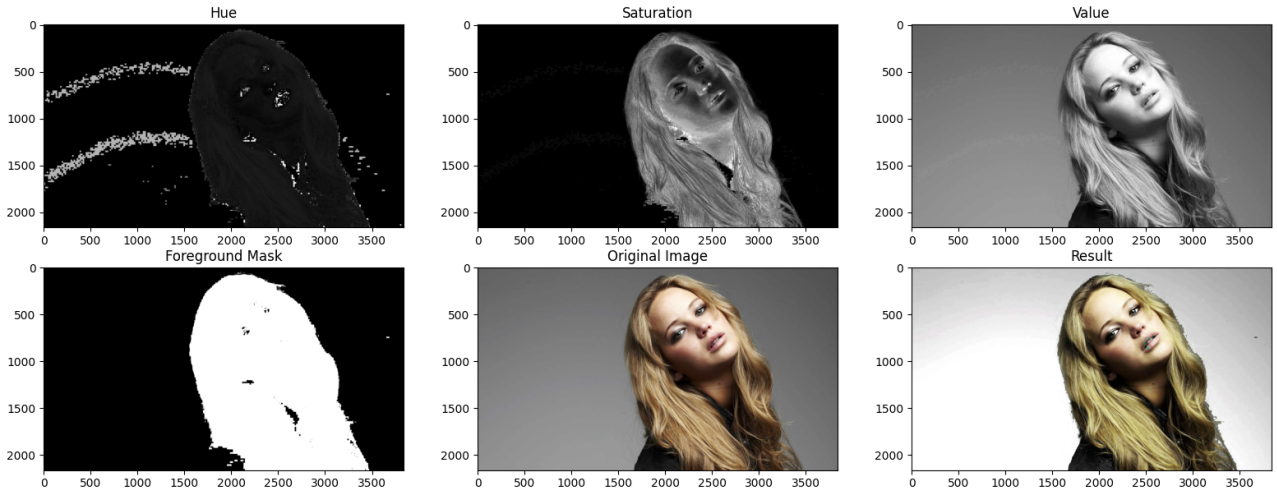
b)

Saturation plane is better to extract the foreground mask.

c,d,e)

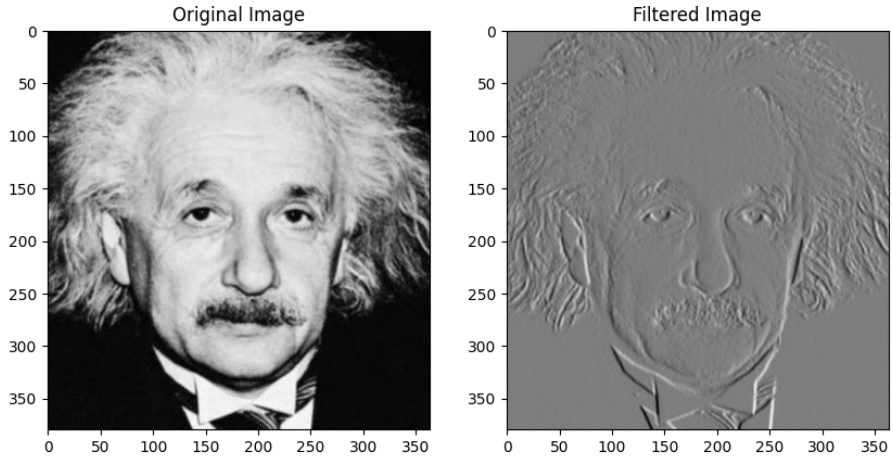
```
foreground = cv2.bitwise_and(v, mask)                          #Obtain foreground using
hist = cv2.calcHist([v], [0], mask, [256], [0, 256])           #Histogram of the value channel
cdf = hist.cumsum()
t = np.array([(256-1)/(cdf[-1])*cdf[k] for k in range(256)]).astype("uint8")
vn = t[v]
```

f)



Hue, Saturation and Value Planes on Grayscale

7 Question 07



Original and Filtered Image

Existing filter2D Function

```
sobelh = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
filtered_func = cv2.filter2D(image, -1, sobelh)
```

Using own code to sobel filter the image

```
for i in range(1, image.shape[0]-1):
    for j in range(1, image.shape[1]-1):
        filtered_own[i, j] = np.sum(pad_image[i-1:i+2, j-1:j+2] * filter)
```

Using Seperable property of Sobel filter

```
for i in range(1, image.shape[0]-1):                #convolve with sobelh1 column vector
    filtered_own1[i] = np.sum(pad_image[i-1:i+2,1:-1] * sobelh1, axis=0)

for j in range(1, image.shape[1]-1):                #convolve with sobelh2 row vector
    filtered_own1[:,j] = np.sum(pad_image[1:-1,j-1:j+2] * sobelh2, axis=1)
```

Following results indicate that there is a significant reduction of cost with the use of seperability property of the kernel.

```
ppData/Local/Programs/Python/Python312/python.exe "d:/projects/github/Image-Processing-and-Computer-Vision/Intensity Transformations and
Neighborhood Filtering/codes/q7.py"
Time taken by filter2D function:  0.0010251998901367188
Time taken by own code:  0.627361536026001
Time taken by seperable property:  0.009042739868164062
```

8 Question 08

We can use opencv's resize function for interpolate on both bilinear and nearest neighbour method.

```
#Nearest Neighbour Method
cv2.resize(img, None, fx=scale, fy=scale, interpolation=cv2.INTER_NEAREST)
```

```
#Bilinear Interpolation Method
cv2.resize(img, None, fx=scale, fy=scale, interpolation=cv2.INTER_LINEAR)
```

Nearest Neighbour (SSD: 31.28432)

Bilinar Interpolation (SSD: 31.05309)

Zoomed Images by Both Methods

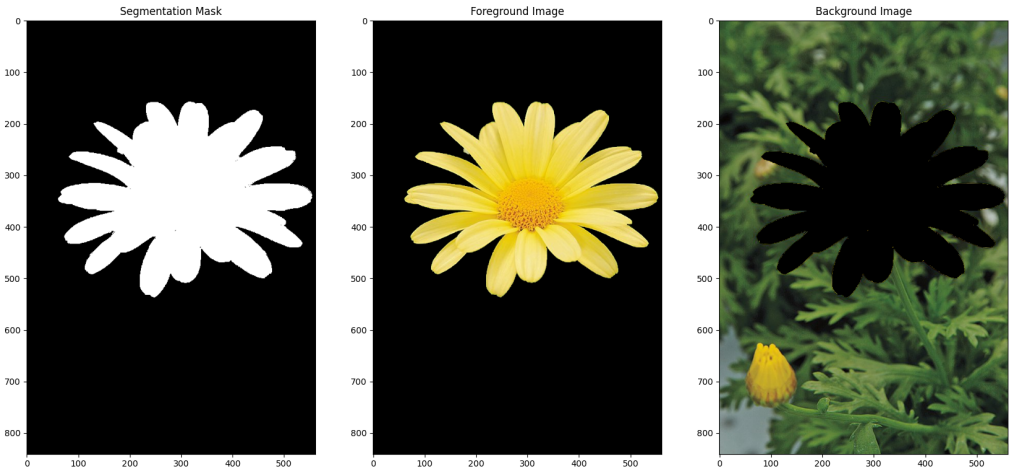
Image Comparison	Nearest Neighbour (SSD)	Bilinear (SSD)
Image1 vs im1_small	31.2843	31.0531
Image2 vs im2_small	11.9020	10.6830
taylor vs taylor_small	36.9263	36.3784
taylor_small vs taylor_very_small	39.3728	39.5428

Table 1: SSD between original images and zoomed images using both methods

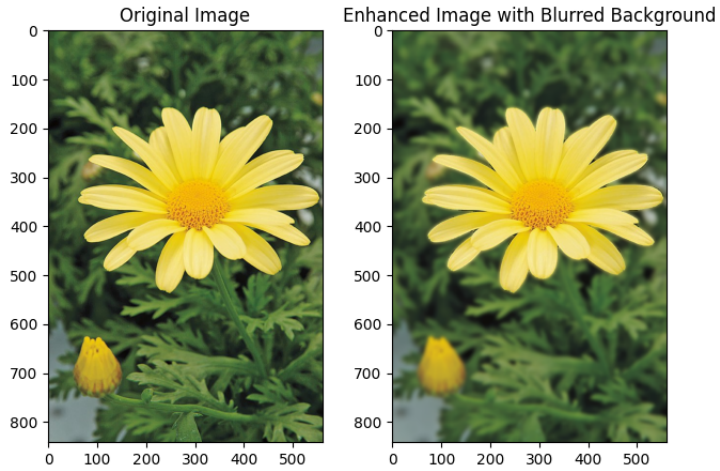
9 Question 09

```
cv2.grabCut(image, mask, rect, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)

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

foreground = image_rgb * mask2[:, :, np.newaxis] # Extract the foreground
background = image_rgb * (1 - mask2[:, :, np.newaxis]) # Extract the background
blurred_background = cv2.GaussianBlur(image_rgb, (21, 21), 0)

# Combine blurred background with the foreground
enhanced_image = blurred_background.copy()
enhanced_image[mask2 == 1] = foreground[mask2 == 1] # Keep the foreground unchanged
```

a)



b)



c)

When applying Gaussian blur on background image it contains black pixels in the place of foreground image. So spatial filtering takes the effect of those pixels introducing some darkness.