University of Moratuwa
Department of Electronic and Telecommunication Engineering

REPORT

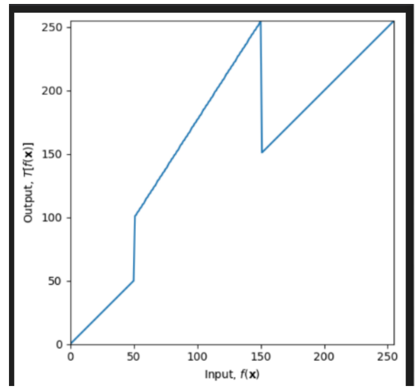
Image Processing & Machine Vision

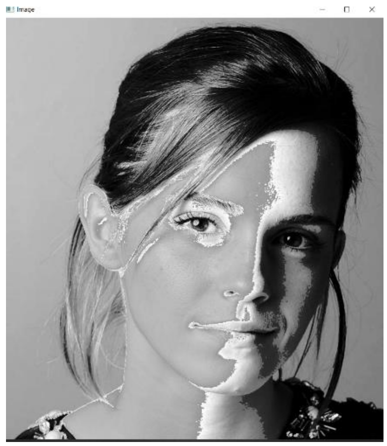
Supervisors – Dr. Ranga Rodrigo

SAIRISAN.R - 200552V

Question 1

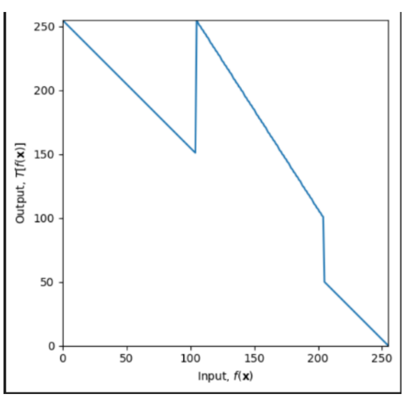
```python
t1 = np.linspace(0,50,51).astype('uint8')   # 0,50,101
t2 = np.linspace(51,100,0).astype('uint8')   #51,200,50
t3 = np.linspace(101,255,100).astype('uint8')   #201,255,105
t4 = np.linspace(255,150,0).astype('uint8')   # 0,50,101
t5 = np.linspace(151,255,105).astype('uint8')   #51,200,50
transform = np.concatenate((t1, t2), axis=0).astype('uint8')
transform = np.concatenate((transform, t3), axis=0).astype('uint8')
transform = np.concatenate((transform, t4), axis=0).astype('uint8')
transform = np.concatenate((transform, t5), axis=0).astype('uint8')
```

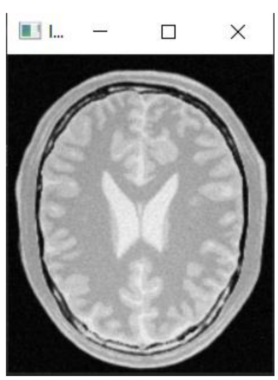


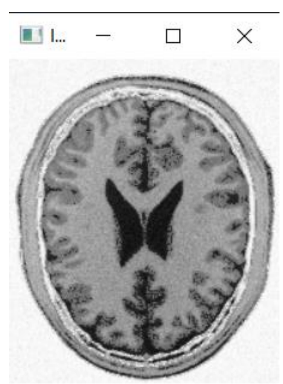


Question 2

```python
t1 = np.linspace(0,50,51).astype('uint8')   # 0,50,101
t2 = np.linspace(51,100,0).astype('uint8')   #51,200,50
t3 = np.linspace(101,255,100).astype('uint8')   #201,255,105
t4 = np.linspace(255,150,0).astype('uint8')   # 0,50,101
t5 = np.linspace(151,255,105).astype('uint8')   #51,200,50
transform = np.concatenate((t1, t2), axis=0).astype('uint8')
transform = np.concatenate((transform, t3), axis=0).astype('uint8')
transform = np.concatenate((transform, t4), axis=0).astype('uint8')
transform = np.concatenate((transform, t5), axis=0).astype('uint8')

transform=np.flip(transform)        #flipping the intensity map array of ealier to get negative.
```



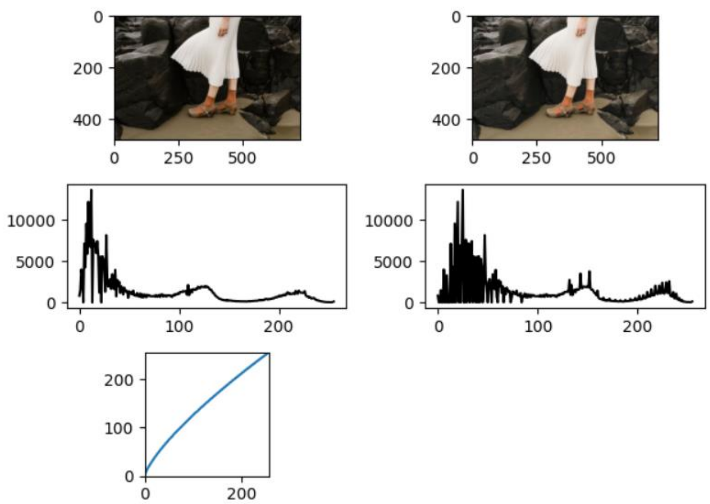




Flip the previous Question transform function to get negative image. So White matter and grey matter are now in their original colour easy to check by medical professionals.

Question 3

```
img_lab = cv.cvtColor(img_orig, cv.COLOR_BGR2LAB)   # Convert the image to LAB color space
L_channel = img_lab[:, :, 0]  # Extract the L channel
# gamma correction to the L channel
gamma = 0.75 #possible value make image looking good.
table = np.array([(i/255.0)**gamma*255.0 for i in np.arange(0, 256)]).astype('uint8')
L_channel_gamma = cv.LUT(L_channel, table)
img_lab_gamma = img_lab.copy()          # Replace the gamma corrected L channel in the LAB image
img_lab_gamma[:, :, 0] = L_channel_gamma
# Convert the LAB image with gamma corrected L channel back to RGB
img_gamma = cv.cvtColor(img_lab_gamma, cv.COLOR_LAB2BGR)
img_gamma = cv.cvtColor(img_gamma, cv.COLOR_BGR2RGB)
```



Gamma Correction Intensity Transformation

Gamma value of 0.5 is reasonable value that make good visual to the image. Here gamma should be less than 1 because in the histogram we can visualize that dark pixels (intensity) are higher. And this makes dark areas to light areas slightly.

Question 4

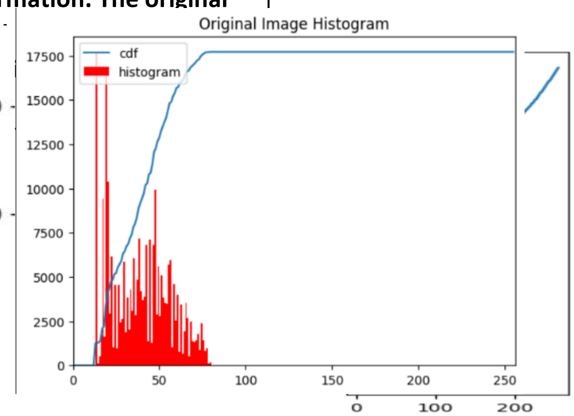
```
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV) #BGR to HSV
hue, saturation, value = cv.split(hsv_image) # Split the HSV image into separate channels
a = 0.5
sigma=40
func=saturation+a*128*np.exp(-((saturation-128)**2)/(2*((sigma)**2)))
transform=np.minimum(func,255).astype('uint8')   # intensity transformation
enhanced_hsv_image = cv.merge((hue, transform, value))        # Combine the transformed satu
enhanced_image = cv.cvtColor(enhanced_hsv_image, cv.COLOR_HSV2BGR)
```

This code enhances image saturation using a Gaussian-based transformation. The original imag... ile ... vibra... y t...

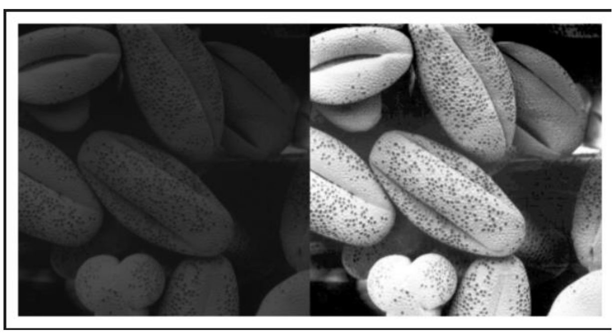
Original Image

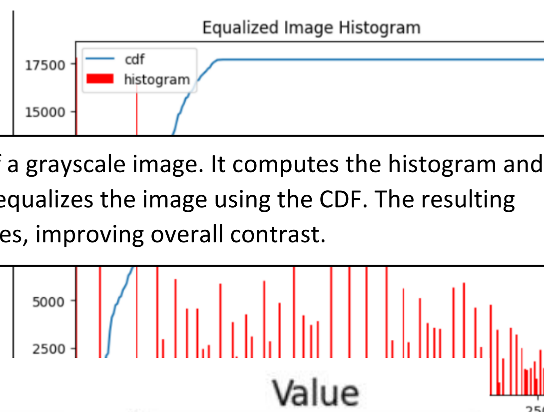
Vibrance-Enhanced Image

```
# defining a function for histogram equalization
def histo_normalized(image):
    hist, bins = np.histogram(image.ravel(), 256, [0, 256])
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()
    equ = cv.equalizeHist(image)
    return cdf_normalized,equ
#apply hist equlize
cdf_normalized,equalize=histo_normalized(img)
```



Question 5





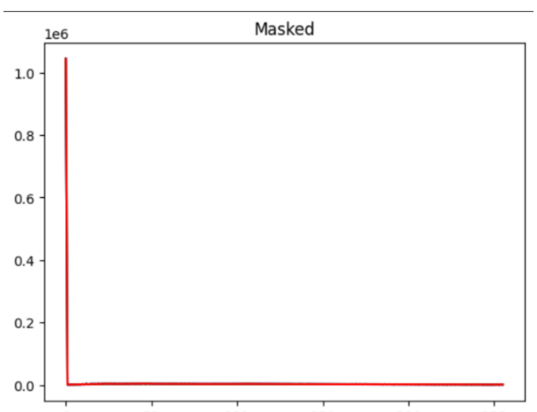
his code applies histogram equalization to enhance the contrast of a grayscale image. It computes the histogram and cumulative distribution function (CDF) of the original image, then equalizes the image using the CDF. The resulting equalized image has a more balanced distribution of pixel intensities, improving overall contrast.

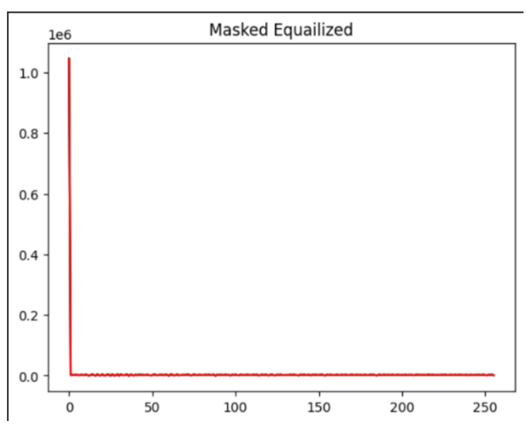
Question 6

We can clearly see that saturation plane is best to choose for separating foreground helps to create masking image. After that masking image is used to extract foreground of original image and each colour channel histogram are equalized and combined with background of original image. Here background can be generated by inverse of masking image is used as mask to extract background image from original.

```python
def resize_im(img,scale):
    new_width = int(img.shape[1] * scale)
    new_height = int(img.shape[0] * scale)
    img=cv.resize(img, (new_width, new_height))
    return img
```

```python
threashold,t=cv.threshold(saturation,11,255,cv.THRESH_BINARY)
masked=cv.bitwise_and(image,image,mask=t)
cv.imshow('masked', masked)
cv.imshow('masking image', t)  #t=mask image

color = ('b', 'g', 'r')

b, g, r = cv.split(masked)
array=[b,g,r]
array_new=[]

for i, c in enumerate(color):             #enumerate is usedhere, so i =0,1,2   (it
    hist, bins = np.histogram(masked.ravel(), 256, [0, 256])
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()
    plt.plot(hist, color=c)                       #plot the graph for need color
    equi = cv.equalizeHist(array[i])
    array_new.append(equi)

plt.title("Masked")
plt.show()


masked_new = cv.merge(array_new)

for i, c in enumerate(color):             #enumerate is usedhere, so i =0,1,2   (it
    hist, bins = np.histogram(masked_new.ravel(), 256, [0, 256])
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()
    plt.plot(hist, color=c)                       #plot the graph for need color

plt.title("Masked Equailized")
```





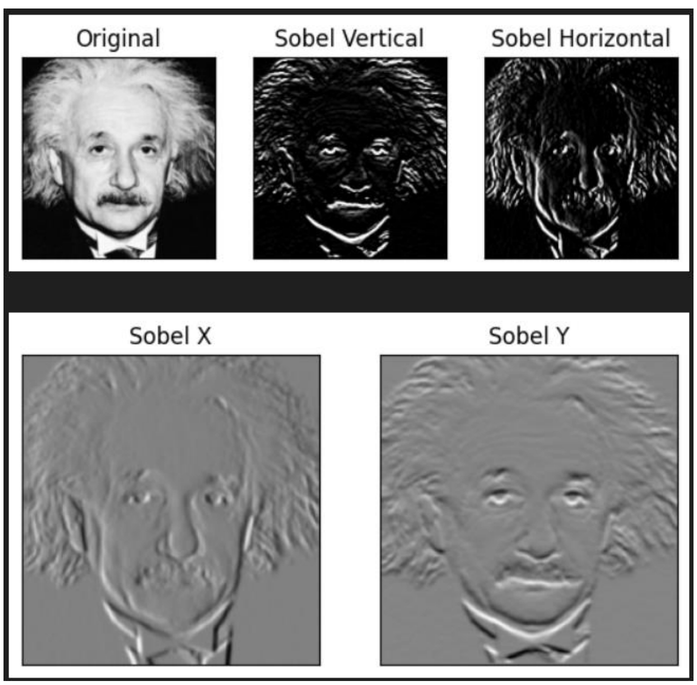
Since there are only few changes in output final image because in the masked image the greatest number of pixels are in 0 intensity so histogram equalization would have a minimal impact. Most of the pixel would still remain dark, and the transformation would primarily redistribute intensities within the dark region, making subtle adjustments but not significantly altering the bright pixel.

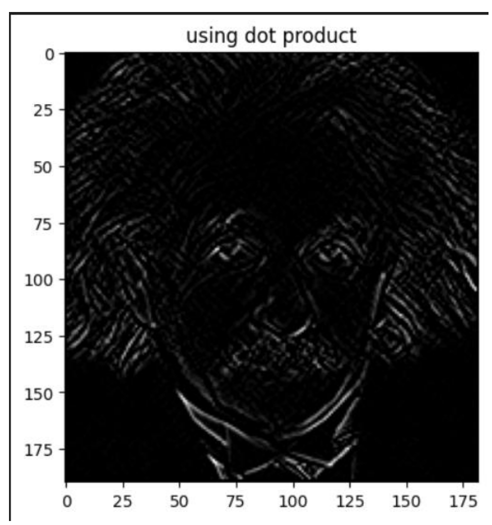
Question 7

```
kernel1 = np.array([[-1, 0, 1],
                    [ -2,  0,  2],
                    [ -1,  0,  1]], dtype=np.float32)


img_filtered1 = cv.filter2D(img, -1, kernel1)

# Display the filtered image
axes[2].imshow(img_filtered1, cmap='gray')
axes[2].set_title('Sobel Horizontal')
axes[2].set_xticks([]), axes[2].set_yticks([])
plt.show()
```

```
# Apply Sobel filter to the image for edge detectio
sobel_x = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=5)
sobel_y = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=5)
```

```
# Define the 1D row and column matrices for So
sobel_x = np.array([[-1, 0, 1]])  # Sobel fil
sobel_y = np.array([[-1], [0], [1]])  # Sobel

# Create the 2D Sobel filter by multiplying t
sobel_filterx = np.dot(sobel_y, sobel_x)
```





Question 8

```python
def zoom_nearest_neighbor(image, scale):
    height, width, channels = image.shape
    new_height = int(height * scale)
    new_width = int(width * scale)

    zoomed_image = np.zeros((new_height, new_width, channels), dtype=np.uint8)

    for i in range(new_height):
        for j in range(new_width):
            original_i = int(i / scale)
            original_j = int(j / scale)
            zoomed_image[i, j] = image[original_i, original_j]

    return zoomed_image
```

```
SSD Value: 847702528.0
```

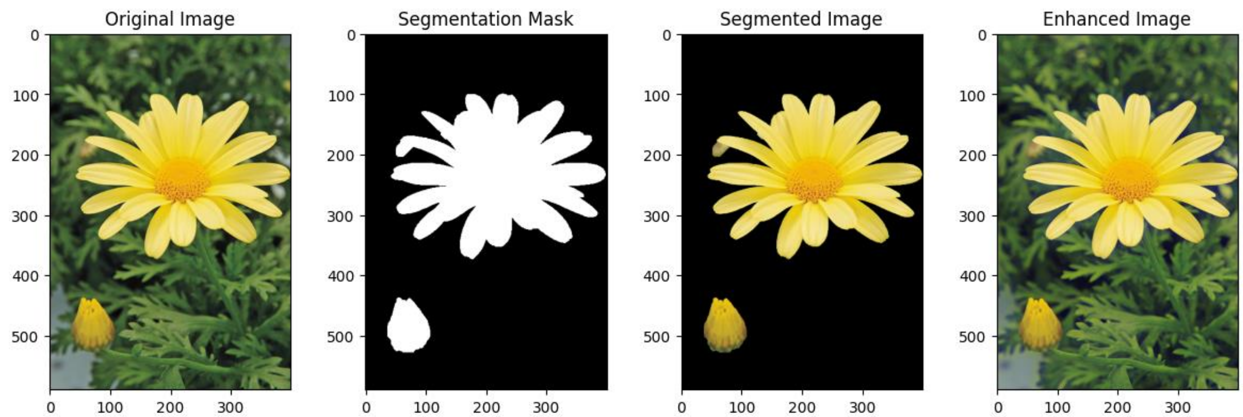
```python
def zoom_bilinear(image, scale):
    height, width, channels = image.shape
    new_height = int(height * scale)
    new_width = int(width * scale)

    zoomed_image = np.zeros((new_height, new_width, channels), dtype=np.uint8)

    for i in range(new_height):
        for j in range(new_width):
            # Calculate the corresponding coordinates in the original image
            original_i = i / scale
            original_j = j / scale


            i1, j1 = int(original_i), int(original_j)
            i2, j2 = min(i1 + 1, height - 1), min(j1 + 1, width - 1) #dimenstion shld not exceed image

            # Calculate the interpolation weights
            difx, dify = original_i - i1, original_j - j1 # 0.67,0.34
            w1 = (1 - difx) * (1 - dify)    #0.67 * 0.34
            w2 = difx * (1 - dify)
            w3 = (1 - difx) * dify
            w4 = difx * dify

            # Perform bilinear interpolation
            pixel_value = (image[i1, j1] * w1 + image[i1, j2] * w2 +
                           image[i2, j1] * w3 + image[i2, j2] * w4).astype(np.uint8)

            zoomed_image[i, j] = pixel_value

    return zoomed_image
```

```
SSD Value: 1239645952.0
```

```python
def calculate_ssd(image1, image2):
    diff = image1.astype(np.float32) - image2.astype(np.float32)
    squared_diff = np.square(diff)
    ssd = np.sum(squared_diff)

    return ssd

# zoomed_large = zoom_nearest_neighbor(large_im, 4)             #uncomment to see neares
# zoomed_small=zoom_nearest_neighbor(small_im, 4)
zoomed_large = zoom_bilinear(large_im, 4)
zoomed_small=zoom_bilinear(small_im, 4)
zoomed_large = cv.resize(zoomed_large, (zoomed_small.shape[1], zoomed_small.shape[0]))
```

Question 9

```
mask = np.zeros(image.shape[:2], np.uint8)
bg = np.zeros((1, 65), np.float64)
fg = np.zeros((1, 65), np.float64)
rect = (10, 10, image.shape[1] - 10, image.shape[0] - 10)  # Define a rough rectangle around the flower
cv2.grabCut(image, mask, rect, bg, fg, 5, cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
segmented_image = image * mask2[:, :, np.newaxis]
blurred_background = cv2.GaussianBlur(image, (0, 0), 10)
enhanced_image = image.copy()  # Apply the mask for focus
enhanced_image[np.where(mask2[:, :, np.newaxis] == 0)] = blurred_background[np.where(mask2[:, :, np.newaxis] == 0)]
```



The reason the background just beyond the edge of the flower is quite dark in the enhanced image is because, during the blurring step, the pixels in the background are blurred significantly, causing them to become darker and less distinct, while the flower remains sharp and in focus.

Github - [SAIRISAN123/Intensity-Transformations-and-Neighborhood-Filtering (github.com)]