Assignment 01: Intensity Transformations and Neighborhood Filtering

EN3160 - Image Processing and Machine Vision

Department of Electronic and Telecommunication Engineering
University of Moratuwa

Samaranayake M.A.S
Index Number: 210559H

1 Implementing Intensity Transformation

Here we have a peace-wise curve for the intensity transformation. To implement the equation following code snippet used and the following result obtained.

```
    x1 = np.linspace(0, 50, 51)
    y1 = x1
    x2 = np.array([50, 50])
    y2 = np.array([50, 100])
    x3 = np.linspace(50, 150, 101)
    y3 = np.linspace(100, 250, 101)
    x4 = np.array([150, 150])
    y4 = np.array([250, 150])
    x5 = np.linspace(150, 255, 106)
    y5 = np.linspace(150, 255, 106)
    # Combine the segments
    x = np.concatenate((x1, x2[1:], x3[1:], x4[1:], x5[1:]))
    y = np.concatenate((y1, y2[1:], y3[1:], y4[1:], y5[1:]))
```

Listing 1: Piece-wise intensity curve

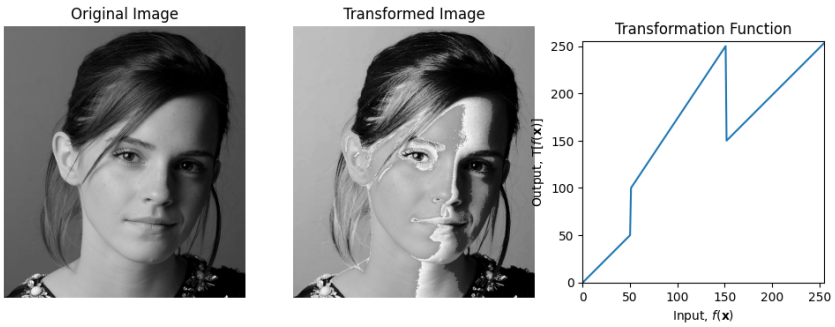
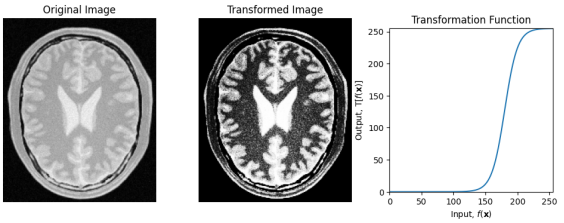


Figure 1: Results after the intensity transformation

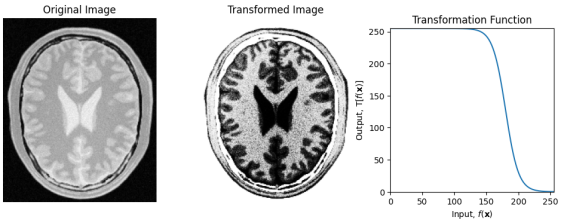
Here we can see that, the darker areas of the image gets more brighter and some brightest parts of the image get more dark.

2 Intensity Transformations of White Matter and Gray Matter

The above image shows the white matter and the gray matter of the Brain. White matter is the middle part of the brain and the outer cells are the gray matter. The following method can be used to extracts those parts. Here I have used the sigmoid and inverse sigmoid functions to extract.



(a) Gray Matter



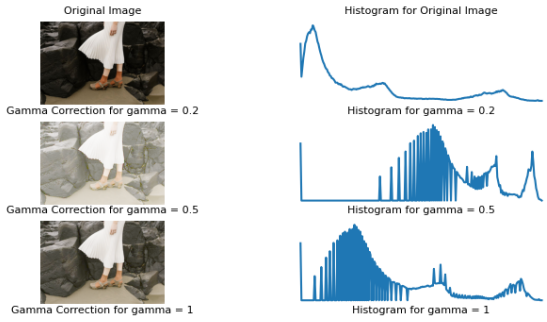
(b) White Matter

```python
def gray_matter_curve(x):
    #inverse sigmoid function to map the intensity values for
       gray matter
    aplpha = 0.1
    beta = 180
    curve = 255/(1+np.exp(aplpha*(x-beta)))
    return curve

def white_matter_curve(x):
    #sigmoid function to map the intensity values for white
       matter
    aplpha = 0.1
    beta = 180
    curve = 255/(1+np.exp(-aplpha*(x-beta)))
    return curve
```

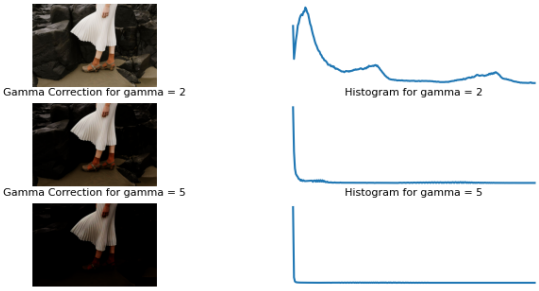
Listing 2: Intensity Curves

3 Gamma Correction

Here we do the linear plane Gamma Correction, with the Gamma values, 0.2, 0.5, 2, 5. Here gamma value 1 refers to the "Original Image".



(a) Gamma Correction



(b) Gamma Correction

4 Vibration Enhancement

Here we split the image into Hue, Saturation, Value planes and apply an intensity transformation for the Saturation for the enhancement of the vibrance of the image. Following code snippet will apply the intensity transformation with $alpha = 0.5$ and $sigma = 70$

```
    def IntensityTransformation(x):
        alpha=0.5
        sigma=70
        curve=x+alpha*128*(np.exp(-((x-128)**2)/(2*sigma**2)))
        return np.minimum(curve, 255)
```

Listing 3: Intensity Curve

Figure 4: Gamma Correction

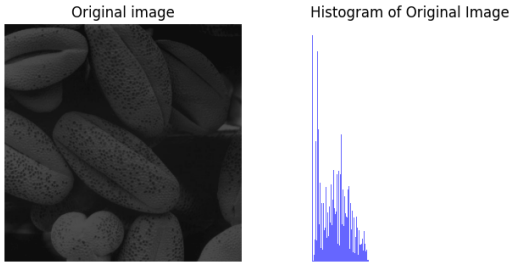
5 Histogram Equalization

Histogram is the pixel intensity distribution of a Image. If we apply the histogram equalization, then the pixel intensity will be distributed throughout 0-255.

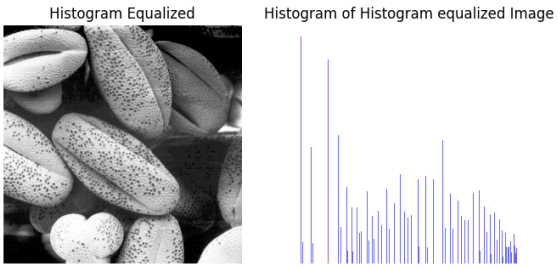
The following code snippet shows the function for *Histogram Equalization*. Here when the histogram equalization is applied, the darker areas get more brightness and image will be more cleared. Here the histograms will be spread through out the 0-255 area.

```
#Function to equalize the histogram of an image
def histogramEqualized(image):

    hist, bins = np.histogram(image.flatten(), bins=256, range
       =[0, 256])
    cdf = hist.cumsum()  # Cumulative sum of the histogram
    cdf_normalized = cdf * hist.max() / cdf.max()  # Normalizing
       the CDF for visualization
    cdf_min = cdf[cdf > 0].min()  # Get the minimum non-zero
       value of the CDF
    cdf_normalized = (cdf - cdf_min) * 255 / (cdf.max() - cdf_min
       )
    cdf_normalized = cdf_normalized.astype(np.uint8)  # Map the
       CDF values to 0-255 range
    equalized_image = cdf_normalized[image]  # Replace pixel
       values using the CDF

    return equalized_image, cdf_normalized
```

Listing 4: Intensity Curve



(a) Before Histogram Equalization



(b) After Histogram Equalization

6 Apply histogram equalization only to the foreground of an image

Here what we do is after importing the image we split it into the Hue, Saturation, and Value planes and selected the saturation plane to mask extraction. After that, we create a mask with the threshold value as 11, to extract the foreground by doing *bitwise_and* operation. To extract the background we can use inverse mask. After extracting the foreground. Then, by applying histogram equalization to the foreground, brightness of the foreground will be increased. By merging foreground and the background, finally we can get the brightness increased image of the character.

Figure 6: Histogram equalization for the foreground

7 Sobel Filtering

7.1 Filter using existing filter2D method

When apply the sobel filtering with filter2D method for the following sobal kernals we can see the following output.

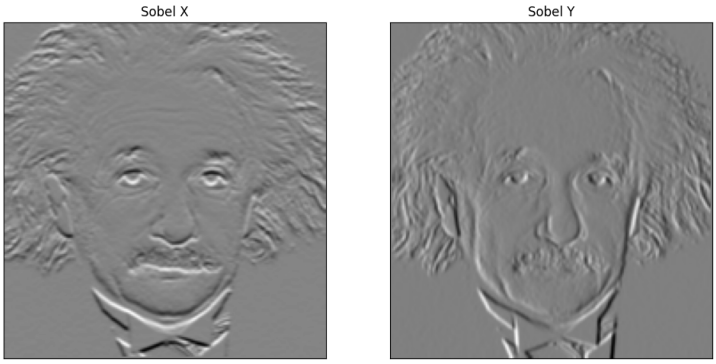


Figure 7: After applying sobel filter

7.2 Filtering using a numerical method

Here we multiply the matrices and obtain the output of the Sobel filtering. The following code snippet derives how the filtering happens.

```
def sobel_filter(image, kernel):
    M, N = image.shape
    (k,k) = kernel.shape
    #sobel filtering without padding
    filtered_image = np.zeros((M,N), dtype=np.float64)
    for i in range(1, M-k+2):
        for j in range(1, N-k+2):
            filtered_image[i,j] = np.sum(image[i-1:i+2, j-1:j+2]*
               kernel)

    return filtered_image
```

Listing 5: Intensity Curve

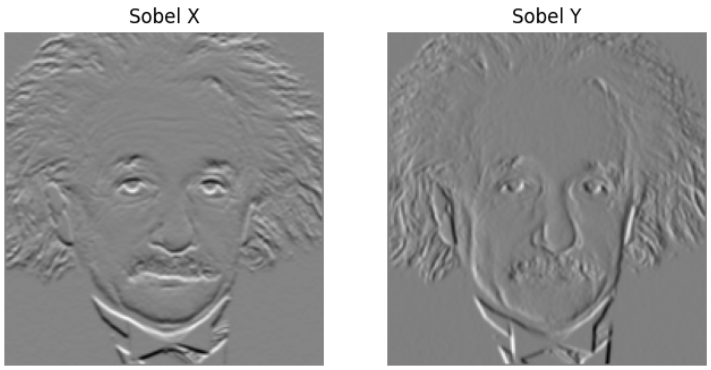


Figure 8: After applying Sobel filter

7.3 Use the given property

Here we apply horizontal and verticle filters to the image to identify the intensity changes, then apply the filters again to detect the edges. The following code snippet shows how to apply sobel filter.

```
def sobel_filter(image):
    # Define the vertical and horizontal 1D Sobel filters
    sobel_vertical=np.array([[1], [2], [1]])
    sobel_horizontal=np.array([[1, 0, -1]])
    # Apply the vertical filter
    vertical_pass=cv.filter2D(image, -1, sobel_vertical)
    # Apply the horizontal filter on the result of the vertical
       pass
    sobel_x=cv.filter2D(vertical_pass, -1, sobel_horizontal)
    # Apply the filters in reverse for the Y-direction Sobel
       filter:
    sobel_horizontal_y=np.array([[1], [0], [-1]])  # Horizontal
       filter for y-direction (1D column)
```

```
    sobel_y=cv.filter2D(image, -1, sobel_horizontal_y)
    return sobel_x, sobel_y
```

Listing 6: Intensity Curve

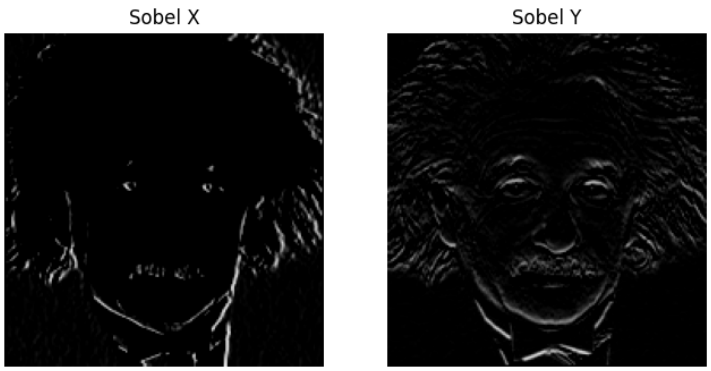


Figure 9: After applying Sobel filter

8 Image Zooming

The following function will zoom the image by faster 4. Compared to the given large images, the zoomed images have little bit of noise. The following results were also obtained.

```
def zoom_image(image, zoom_factor, interpolation):
    M, N, c = image.shape
    new_M = int(M * zoom_factor)
    new_N = int(N * zoom_factor)
    if interpolation == cv.INTER_NEAREST:
        zoomed_image = cv.resize(image, (new_N, new_M),
           interpolation=cv.INTER_NEAREST)
    elif interpolation == cv.INTER_LINEAR:
        zoomed_image = cv.resize(image, (new_N, new_M),
           interpolation=cv.INTER_LINEAR)
    else:
        try:
            zoomed_image = cv.resize(image, (new_N, new_M),
               interpolation=interpolation)
        except:
            raise ValueError('Invalid interpolation method')
    return zoomed_image
```

Listing 7: Intensity Curve

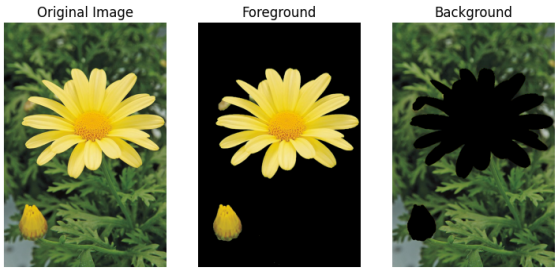
Figure 10: After zooming the images

9 Image Segmentation

Here we use *grabCut segmentation* to mask the foreground and the background of the image. After masking we add a gaussian blur to the background and then add the foreground and the background to produce a blurred-background enhanced image.Masking the image cause to cut some of the pixels in the background as well. That results some darker areas around the margins.

```
M,N,c = img.shape
mask = np.zeros(img_2.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
rect = (50,50,img.shape[1]-50,img.shape[0]-50)
cv.grabCut(img_2,mask,rect,bgdModel,fgdModel,5,cv.
    GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
foreground = img* mask2[:, :, np.newaxis]  # Foreground
background = img* (1 - mask2[:, :, np.newaxis])  # Background
```

Listing 8: Intensity Curve



(a) Foreground Background Mask

(b) Background-blurred enhanced Image

10 GitHub Repository

Click here to view the GitHub repository