University of Moratuwa

FUNDAMENTALS OF IMAGE PROCESSING

EN2550

INTENSITY TRANSFORMATIONS & NEIGHBORHOOD FILTERING

S.Sanjith : 190562G

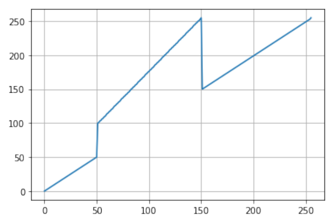
March 6, 2022

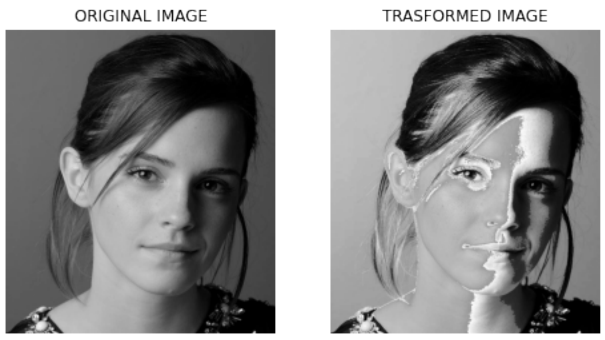
Contents

Git-hub link : `https://github.com/sanjith1999/EN2550-Assignments.git`

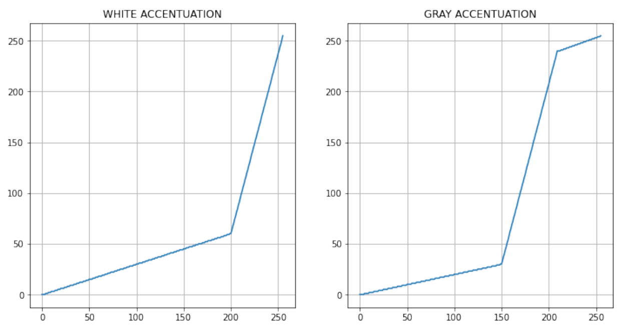
1 Intensity Transformation

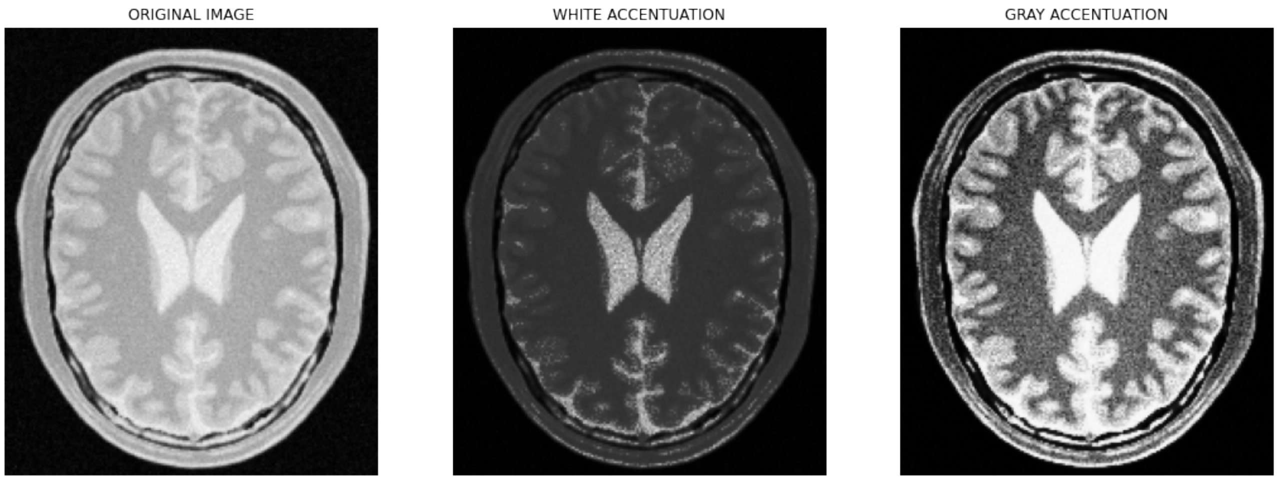
The first image depicts the situation where a photo of a girl is taken without a clear lighting setup. Most of the details on the right side of the face are hidden. Through stretching and increasing the brightness of such range with an intensity transformation a viewer can extract the details hidden in such range. Furthermore, through the results depicted in the image, it is obvious that with fine-tuning of brightness and contrast parameters it is possible to generate perfect photos from photos without proper lighting to an extent.





2 Accenuation of a White & Gray Matter

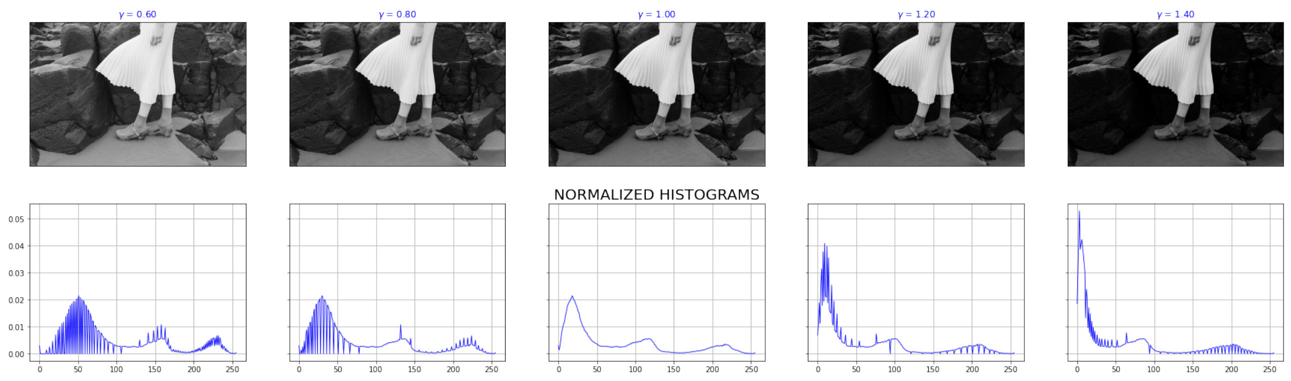




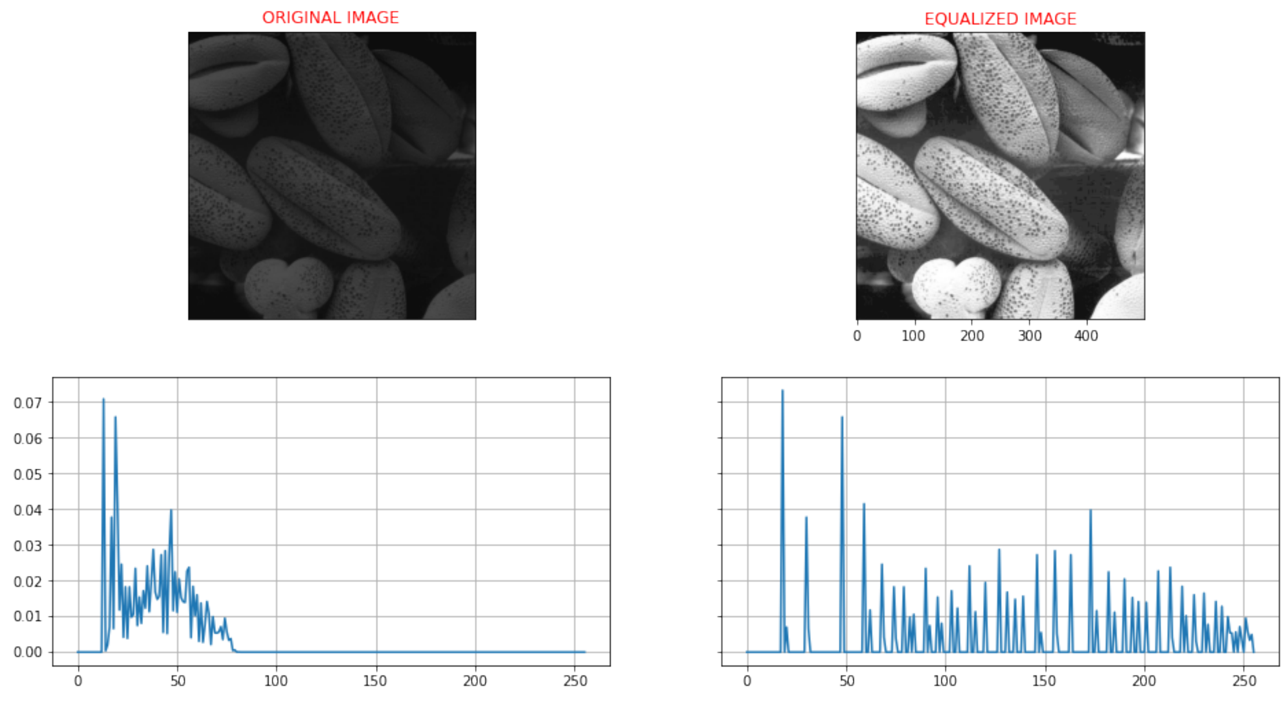
To accentuate the details in a particular intensity region we have to provide more intensity range to such a small range through intensity transformations as shown in the above plots. The above picture describes a situation where white and gray regions are being provided with a wider range of intensities to extract more information from them. The second image is the best suit for analyzing white matter and the third image for gray matter for medical diagnosing purposes.

3 Gamma Correction

Gamma correction can be used to change the distribution of intensities throughout the image. In the image illustrated below, it is obvious that $\gamma > 1$ will generate a darker image than the original. And $\gamma < 1$ produces an image that is lighter than the original which is the best suit for this situation. So it is important to choose the best γ according to the range of intensity distribution.



4 Histogram Equalization



Histogram equalization is a technique used to equalize the distribution of intensities over the full available range. The idea is to use the normalized cumulative distribution of the original image as the intensity transformation. The above histograms depict the distribution of intensities in the original and transformed image. Although the image is darker after the transformation, intensities have been distributed all over the range. This technique is suitable for images that contain intensities contracted over a small range. The following code illustrates the implementation of this technique.

```
    hist_f4=cv.calcHist([f4],[0],None,[256],[0,256])
    pdf=hist_f4/np.sum(hist_f4)
    cdf=np.cumsum(pdf)

    t_equalization=255*cdf
    f4_equalized=cv.LUT(f4,t_equalization)
    hist_equalized=cv.calcHist([f4_equalized],[0],None,[256],[0,256])
    pdf_equalized=hist_equalized/np.sum(hist_equalized)
```

5 Image Zoom

Stretching the dimensions of the original image generates a larger version of the current image.

5.1 Nearest-Neighbor Method

Intensity approximation of a zoomed member to the nearest pixel in the original.

```
    for row in range(zoomedImg.shape[0]):
        source_row = min(round(row/sf), img.shape[0]-1)
        for column in range(zoomedImg.shape[1]):
            source_column = min(round(column/sf), img.shape[1]-1)

            # FOR GRAY IMAGE
            if len(img.shape) == 2:
                zoomedImg[row][column] = img[source_row][source_column]

            # FOR COLOR IMAGE
            else:
                for channel in range(3):
                    zoomedImg[row][column][channel] = \
                        img[source_row][source_column][channel]
```

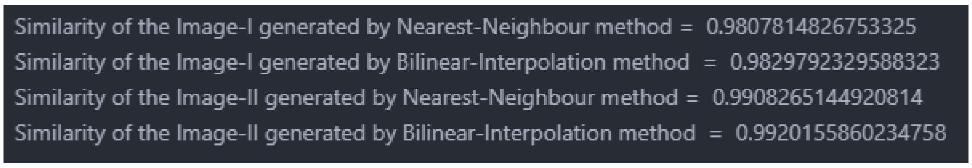
5.2 Bi-Linear Interpolation Method

Intensity calculation for each member of the zoomed image via linearly interpolating the four nearest neighbors in the original image.

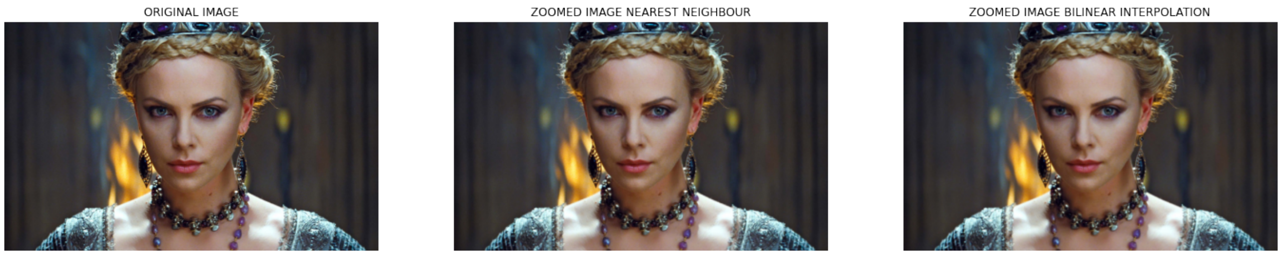
```
for row in range(zoomedImg.shape[0]):
    row_position = row/sf
    row_below = math.floor(row_position)
    row_up = min(math.ceil(row_position),img.shape[0]-1)
    for column in range(zoomedImg.shape[1]):
        column_position = column/sf
        column_previous = math.floor(column_position)
        column_next = min(math.ceil(column_position),img.shape[1]-1)
        delta_row = row_position - row_below
        delta_column = column_position - column_previous

        # FOR GRAY IMAGE
        if len(img.shape) == 2:
            interVal1 = img[row_below][column_previous]*(1-delta_row)\
                + img[row_up][column_previous]*(delta_row)
            interVal2 = img[row_below][column_next]*(1-delta_row)\
                + img[row_up][column_next]*(delta_row)
            zoomedImg[row][column] = (interVal1*(1-delta_column)
                                      + interVal2*(delta_column))
```

Analyzation of Results

Similarity Measure with Normalized SSD



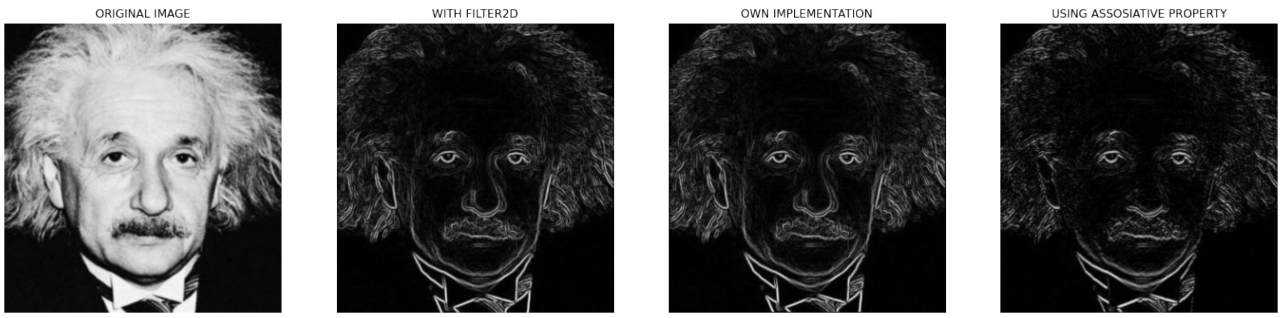
```
Similarity of the Image-I generated by Nearest-Neighbour method =  0.9807814826753325
Similarity of the Image-I generated by Bilinear-Interpolation method  =  0.9829792329588323
Similarity of the Image-II generated by Nearest-Neighbour method =  0.9908265144920814
Similarity of the Image-II generated by Bilinear-Interpolation method  =  0.9920155860234758
```


ORIGINAL IMAGE
ZOOMED IMAGE NEAREST NEIGHBOUR
ZOOMED IMAGE BILINEAR INTERPOLATION

By the results obtained for similarity measure, it is obvious that bilinear interpolation yields more accurate results for zoom images. Anyhow both the methods yield the result with more than 99% percent similarity which leads to same-looking images for human eyes. The pictures of original and zoomed images from both the methods look indistinguishable although they have similarity variation in thrid decimal point. The code used to calculate the normalized error and similarity percent is as follows.

```
    e1_nn = cv.norm(f5_1,zoom_im1_nn, cv.NORM_L2)
    e1_bi = cv.norm(f5_1,zoom_im1_bi, cv.NORM_L2)
    similarity_nn_1 = 1 - e1_nn/(f5_1.shape[0]*f5_1.shape[1])
    similarity_bi_1 = 1 - e1_bi/(f5_1.shape[0]*f5_1.shape[1])
```

6 Sobel Filter



Sobel filter gives a good idea about the edges in an image. The images shown above are generated using a Sobel filter with different approaches. From the images, it is obvious that the filter traces out a clear edge outline of Einstein.

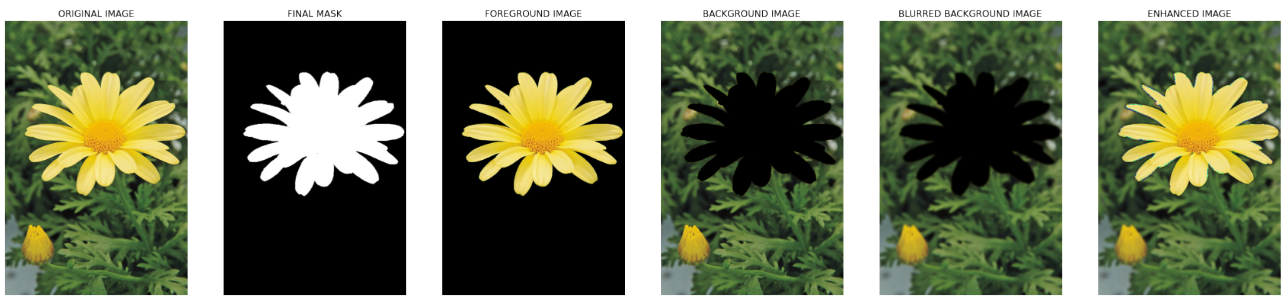
The similarity of the results ensures the idea that convolution is associative and commutative. Although loop implementation of result yields the same result as filter2D function, it increases the computational complexity of the calculation. All the codes used to generate such images are illustrated below.

```
    kernel_v=np.array([(-1,-2,-1),(0,0,0),(1,2,1)],np.float32)
    kernel_h=np.array([(-1,0,1),(-2,0,2),(-1,0,1)],np.float32)

    f6_x=cv.filter2D(f6,-1,kernel_v)
    f6_y=cv.filter2D(f6,-1,kernel_h)
    f6_sobel=np.sqrt(f6_x**2+f6_y**2)

    # WITH OWN IMPLEMENTATION
    kernel_vo=np.array([(1,2,1),(0,0,0),(-1,-2,-1)],np.float32)
    kernel_ho=np.array([(1,0,-1),(2,0,-2),(1,0,-1)],np.float32)
    f6_xo=filter(f6,kernel_vo)
    f6_yo=filter(f6,kernel_ho)
    f6_sobel_o=np.sqrt(f6_xo**2+f6_yo**2)

    # USING THE ASSOSIATIVE PROPERTY
    kernel_v1 = np.array([-1, 0, 1], dtype = np.float32)
    kernel_v2 = np.array([[1], [2], [1]], dtype = np.float32)
    kernel_h1 = np.array([1, 2, 1], dtype = np.float32)
    kernel_h2 = np.array([[-1], [0], [1]], dtype = np.float32)

    f6_x_=cv.filter2D(f6,-1,kernel_v1)
    f6_x_=cv.filter2D(f6_x_,-1,kernel_v2)

    f6_y_=cv.filter2D(f6,-1,kernel_h1)
    f6_y_=cv.filter2D(f6_y_,-1,kernel_h2)


    f6_sobel_asso=(np.sqrt(f6_x_**2+f6_y_**2))
```

7 Image Enhancement

Separating foreground and background features of an image and performing filter operations appropriately, can be effectively used to enhance the images.

The following sequence of images illustrates the steps of the image enhancement process.



The background just beyond the edge of the flower is quite dark in the enhanced image. This is because when blurring the background using a gaussian kernel, the background just beyond the edge of the flower is affected by the neighboring darker pixels contained in the flower. In the background, the flower is replaced by dark pixels.
The techniques used in the image enhancement process are explained below...

7.1 Segmentation of Image

By creating a loop enclosing the foreground features using the grabCut function and zeroing out all the other pixels, we can create a mask to separate the foreground image from the original image.

```
    mask = np.zeros(f7.shape[:2],np.uint8)
    bgdModel = np.zeros((1,65),np.float64)
    fgdModel = np.zeros((1,65),np.float64)
    rect = (30,70,650,550)

    cv.grabCut(f7,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)
    mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
    fore = f7*mask2[:,:,np.newaxis]
    back = f7 - fore
```

7.2 Blurring Background

After separating the background pixels of an image and applying a simple Gaussian filter we can blur the background portion of the image.

```
    blurred_bg = cv.GaussianBlur(back, (9,9), 4)
    enhanced = fore + blurred_bg
```

References

1. Similar Image Processing Project
2. Opencv Home
3. An Article on Usage of Sobel Operator
4. An Article on Image zooming
5. Tutorial on Foreground Segmentation and Extraction