190026T

AHMAED M.I.I

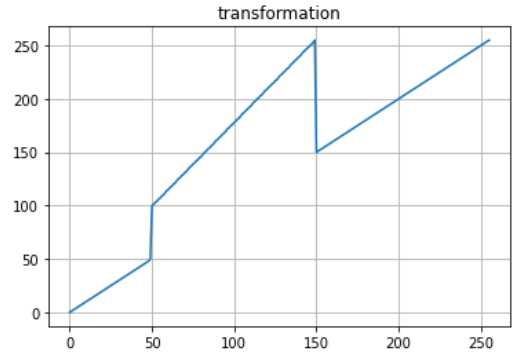
GitHub: https://github.com/ishrath99/S4_EN_2550/tree/main/Assignments/Assignment_1

EN 2550

Assignment 1 – Report

Question 1

After applying the following transformation, the result appears as follows.



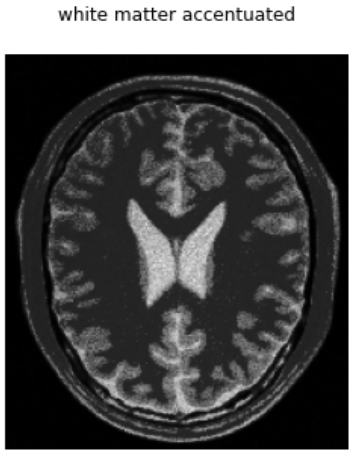
original image

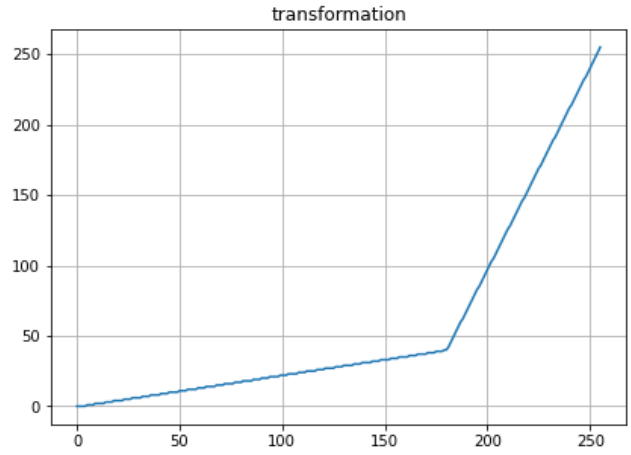
transformed

The white (150-255) and the black (0-50) pixels of the image is stays the same. Gray pixels of the image have been transformed to white pixels. Pixels ranging from 51 to 99 does not appear in the transformed image.

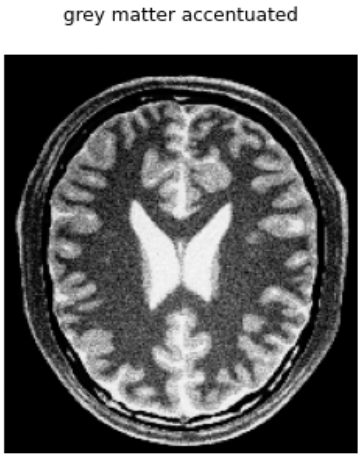
Question 2

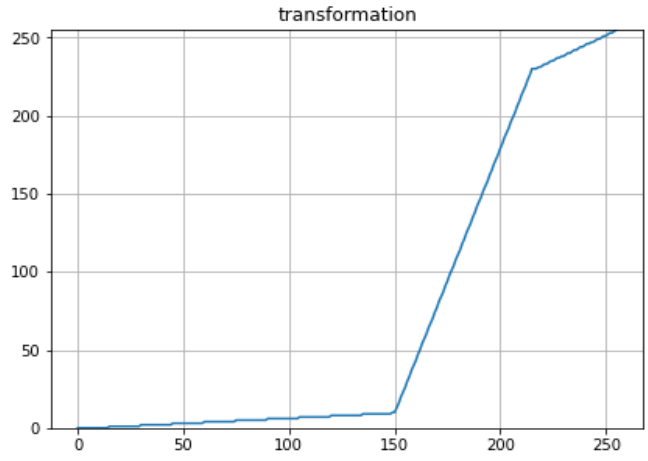
White and gray matter accentuation is done to the following image.





As we can observe, details of the white pixels are accentuated. They are mapped to a larger range.

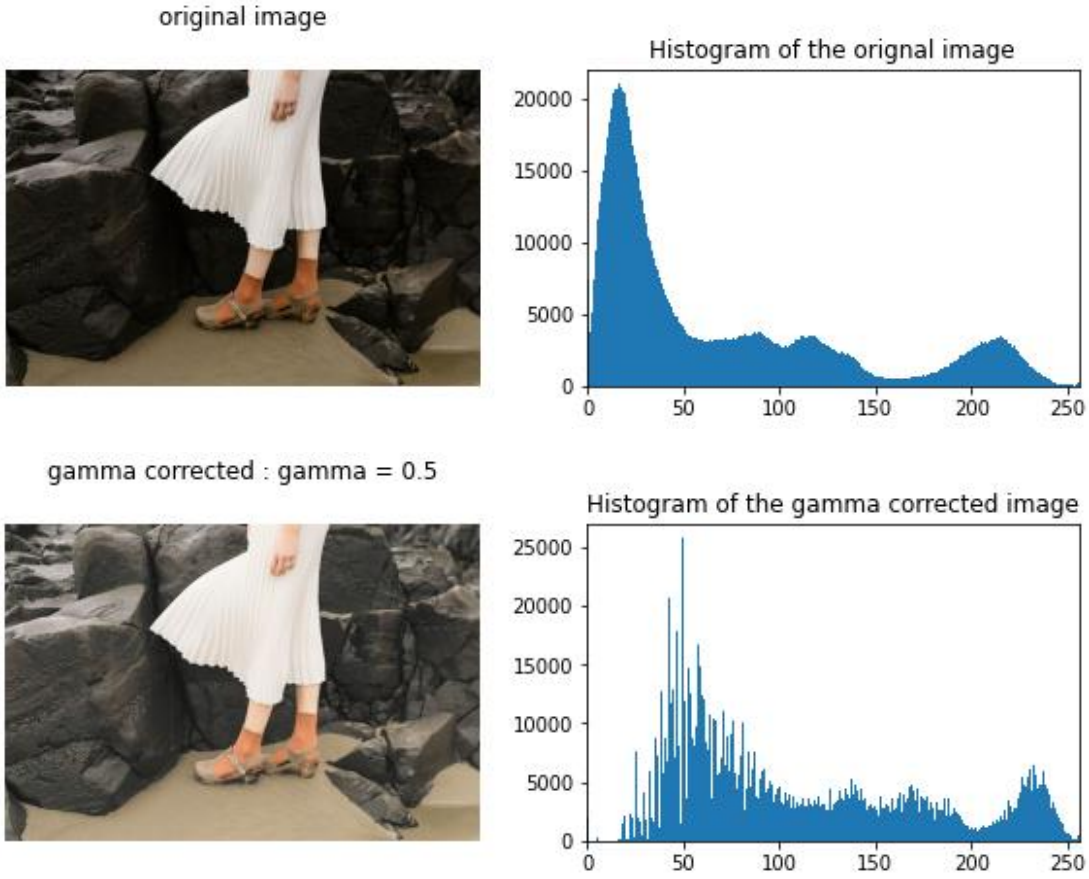




Here, grey pixels (150-230) are mapped to a larger range to accentuate the gray pixels of the image.

Question 3

Gamma correction has been done on the L plane in the L*a*b plane of the image. Gamma value is chosen to 0.5. As the L refers to the lightness of the image, we can expect to gamma corrected image to be brighter. Because, for a gamma value like 0.5, darker pixels get mapped to a larger range of lighter pixels.



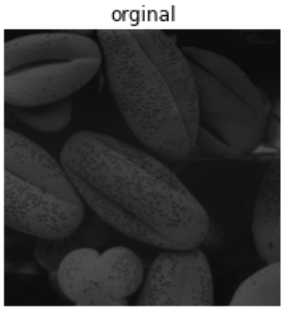
It can be observed from both histogram and the image that, darker pixels are reduced and they are shifted to the right side (towards lighter pixels).

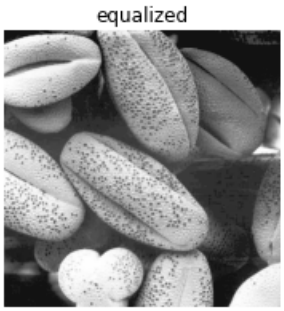
Question 4

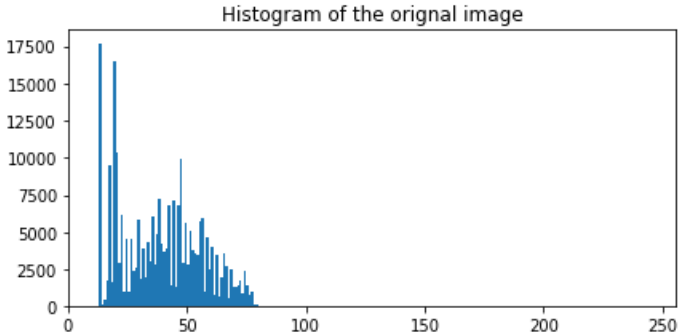
Following is the function to carry out histogram equalization.

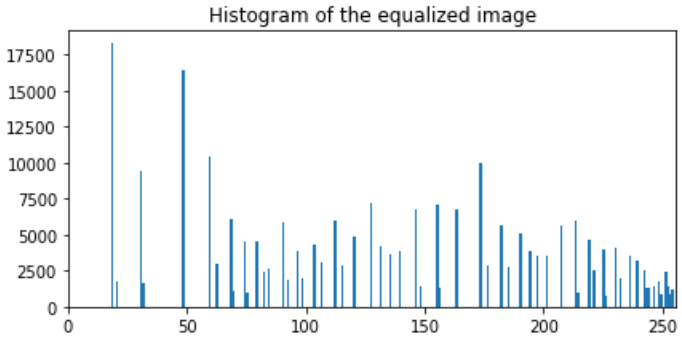
```
def histogramEqualization(image):
    hist ,bins = np.histogram(image.ravel(), 256, [0, 256])
    cdf = hist.cumsum()
    cdf_normalized = cdf*(len(hist)-1)/cdf.max()
    transformation = cdf_normalized.astype(int)
    equalized = cv.LUT(image, transformation)
    return equalized
```

The following figure shows the results after applying the above function to an image.









Question 5

Function for zooming using bilinear interpolation technique,

```
def zoom_bilinear_interpolation(image, scale):
    rows = int(image.shape[0]*scale)
    cols = int(image.shape[1]*scale)
    zoomed = np.zeros((rows, cols, 3), dtype=image.dtype)
    for r in range(rows):
        for c in range(cols):
            if math.ceil(r/scale) == image.shape[0]:
                r -= scale
            if math.ceil(c/scale) == image.shape[1]:
                c -= scale

            r_im, c_im = r/scale, c/scale
            row_floor_weight = math.ceil(r_im) - r_im
            row_ceil_weight = r_im - math.floor(r_im)
            col_floor_weight = math.ceil(c_im) - c_im
            col_ceil_weight = c_im - math.floor(c_im)
            l_pixel = image[math.floor(r_im), math.floor(c_im)]*row_floor_weight +
image[math.ceil(r_im), math.floor(c_im)]*row_ceil_weight
            r_pixel = image[math.floor(r_im), math.ceil(c_im)]*row_floor_weight +
image[math.ceil(r_im), math.ceil(c_im)]*row_ceil_weight
            zoomed[r,c,0] = round(l_pixel[0]*col_floor_weight + r_pixel[0]*col_ceil_weight)
            zoomed[r,c,1] = round(l_pixel[1]*col_floor_weight + r_pixel[1]*col_ceil_weight)
            zoomed[r,c,2] = round(l_pixel[2]*col_floor_weight + r_pixel[2]*col_ceil_weight)

    return zoomed.astype(np.uint8)
```

Function for zooming using nearest neighbor technique,

```
def zoom_nearest_neighbour(image, scale):
    rows = int(image.shape[0]*scale)
    cols = int(image.shape[1]*scale)
    zoomed = np.zeros((rows, cols, 3), dtype=image.dtype)
    for r in range(rows):
        for c in range(cols):
            if round(r/scale) == image.shape[0]:
                r -= scale
            if round(c/scale) == image.shape[1]:
                c -= scale
            zoomed[r,c] = image[round(r/scale), round(c/scale)]

    return zoomed.astype(np.uint8)
```

Scaling factors are chosen in the range (0, 10]. The following figures show the results.

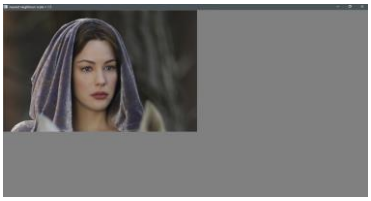
- NN: Nearest neighbor
- BI: Bilinear interpolation

Original

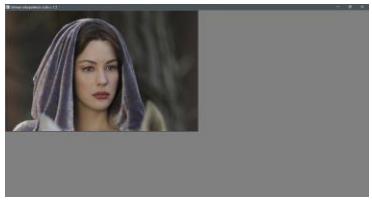
NN: scale = 2.5

BI: scale = 2.5

Original



NN: scale = 1.5



BI: scale = 1.5

Original

NN: scale = 0.8

BI: scale = 0.8

Results can be viewed clearly in the following GitHub repository,

https://github.com/ishrath99/S4_EN_2550/tree/main/Assignments/Assignment_1/q5results

Similarity of 2 images after zooming has been checked using the ***cv.norm*** function. Original and the zoomed image are compared, and the results shows that, they are almost the same.

```
Similarity for image 1 =  0.9690700202931698
Similarity for image 2 =  0.9833335779097806
```

Question 6

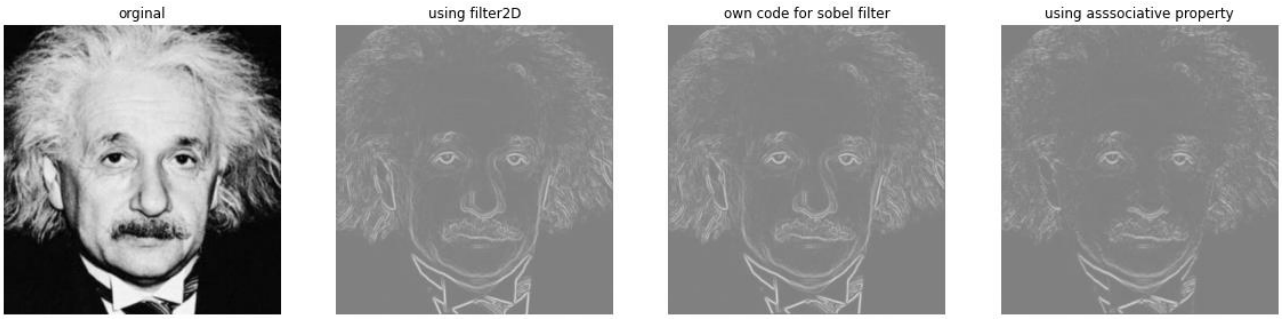
Function to Sobel filter an image,

```
def filter(image, kernel):
    assert kernel.shape[0]%2 == 1 and kernel.shape[1] % 2 == 1
    kernel_h, kernel_w = math.floor(kernel.shape[0]/2), math.floor(kernel.shape[1]/2)
    image_h, image_w = image.shape
    image_float = image.astype('float')
    filtered = np.zeros(image.shape, 'float')

    for i in range(kernel_h, image_h - kernel_h):
        for j in range(kernel_w , image_w - kernel_w):
            filtered[i, j] = np.dot(image_float[i - kernel_h : i + kernel_h + 1 , j - kernel_w
: j+ kernel_w + 1].flatten(), kernel.flatten())

    return filtered
```

After filtering in both directions, final output is obtained by taking the magnitude of the gradients. Following figure shows the original image and the outputs obtained by ***filter2D*** function, custom code for filtering and by using the associative property of matrices.



Code for using associative property,

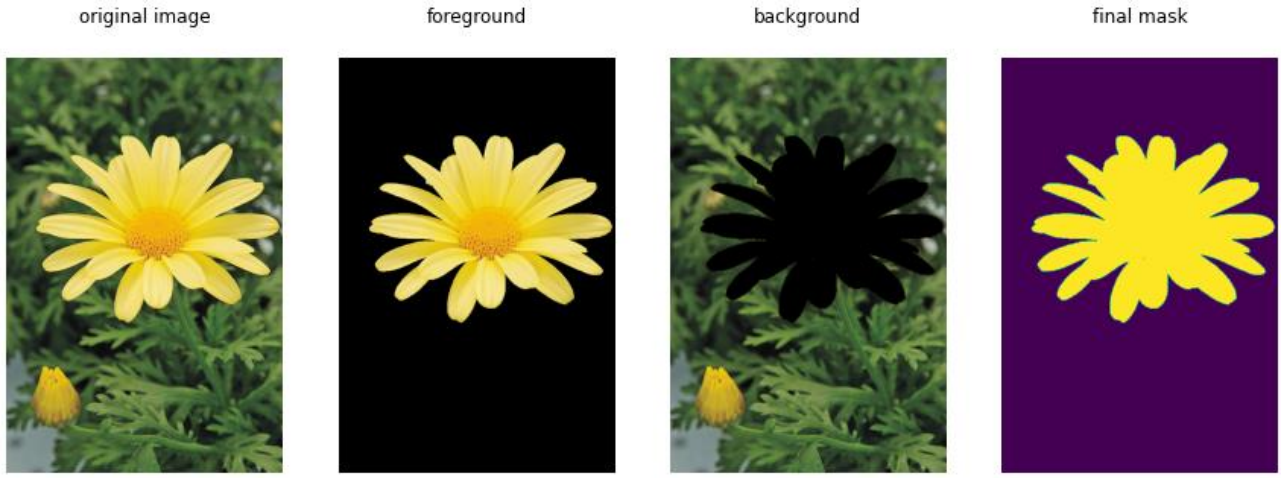
```
kernel_y1 = np.array([-1, 0, 1], dtype = np.float32)
kernel_y2 = np.array([[-1], [2], [1]], dtype = np.float32)
image_y2 = cv.filter2D(einstein, -1, kernel_y1)
image_y2 = cv.filter2D(image_y2, -1, kernel_y2)

kernel_x1 = np.array([-1, -2, -1], dtype = np.float32)
kernel_x2 = np.array([[1], [0], [-1]], dtype = np.float32)
image_x2 = cv.filter2D(einstein, -1, kernel_x1)
image_x2 = cv.filter2D(image_x2, -1, kernel_x2)

grad_mag2 = np.sqrt(image_y2**2 + image_x2**2)
```

Question 7

The following figure shows the original image, foreground image, background image and the final segmentation mask.



For the purpose of producing an enhanced image, background of the image is blurred. Gaussian blur with kernel size 9 and sigma 4 is used to blur the background. The following figure shows the original image, blurred background and the enhanced image.

c) Because, when blurring the background using gaussian kernel, background just beyond the edge of the flower is affected by the neighboring darker pixels contained in the flower. In the background, flower is replaced by the dark pixels.