EN2550

Fundamentals of Image Processing and Machine vision

Assignment 01

Index Number: 190328V

Name: KUMARA B.W.J.C.

GitHub Link: https://github.com/WikumJCK/EN2550_Image_Processessing.git

Question 01

In intensity transformation, we enhance selected range of pixel vales, So, brightness of those pixels will increase. New features of the image can be obtained by this method. In the below, image of Emma Watson is enhanced according to the intensity transformation graph showed in figure 01, the result is pixels that are gray have enhanced so dark side of the face is lighted.

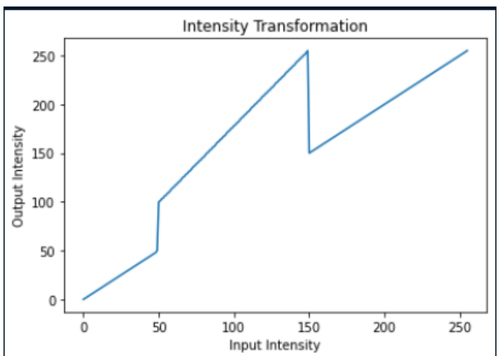

Fig 01

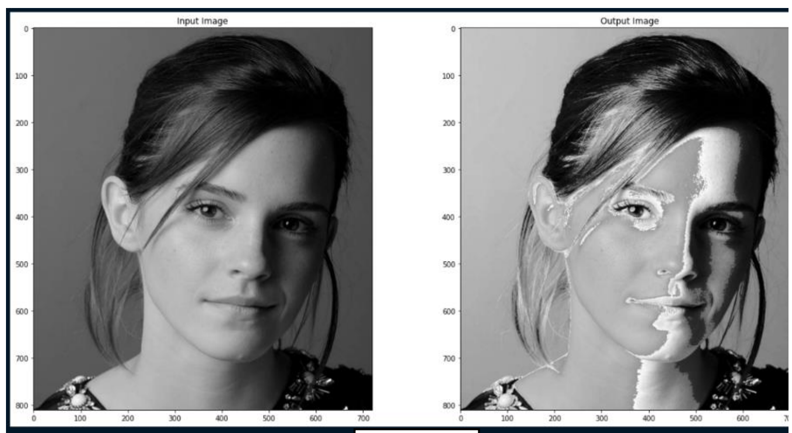

Fig 02

```python
# Creating the Intensity transformation array
t1 = np.linspace(0,50,50)
t2 = np.linspace(50,100,0)
t3 = np.linspace(100,255,100)
t4 = np.linspace(255,150,0)
t5 = np.linspace(150,255,106)
t = np.concatenate((t1,t2,t3,t4,t5),axis =0).astype(np.uint8)

assert len(t) == 256
t_img = cv.LUT(img,t) # Intensity Transformation
```

Using Look up tables to map the corresponding pixel values reduce the computational complexity of doing intensity transformation.

Question 02

In this Question we have enhance white matter and gray matter using intensity transformation. First, we must identify the range of pixel values we need to enhance to separate gray matter and white matter. For that I manually changed intensity transformation graph and selected suitable range.

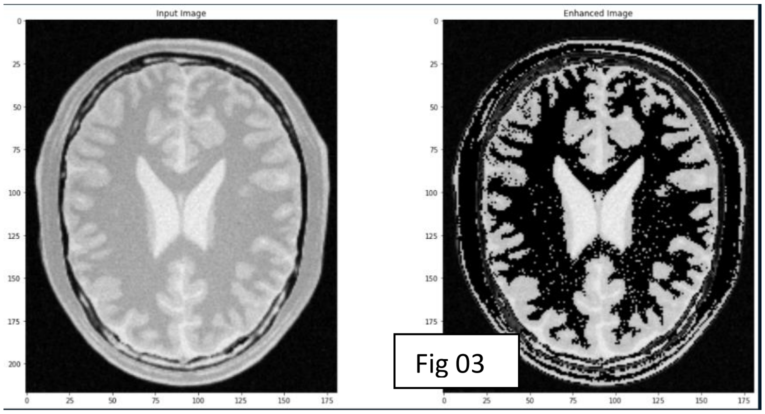

Fig 03

Fig 03 represents original image and Gray and White matter enhanced image, Black color section represent white matter section while white color section represent gray matter section. Intensity transformation of the above image is shown in Fig 04.

Code used in this Question is same as Q01,

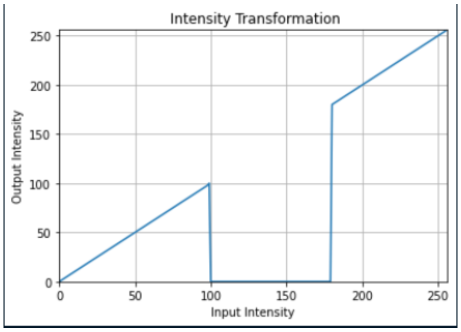

Fig 04

Question 03

Gamma correction is also a Intensity transformation, in this method intensity transformation is nonlinear. By changing gamma value, we can change the darkness of the image.

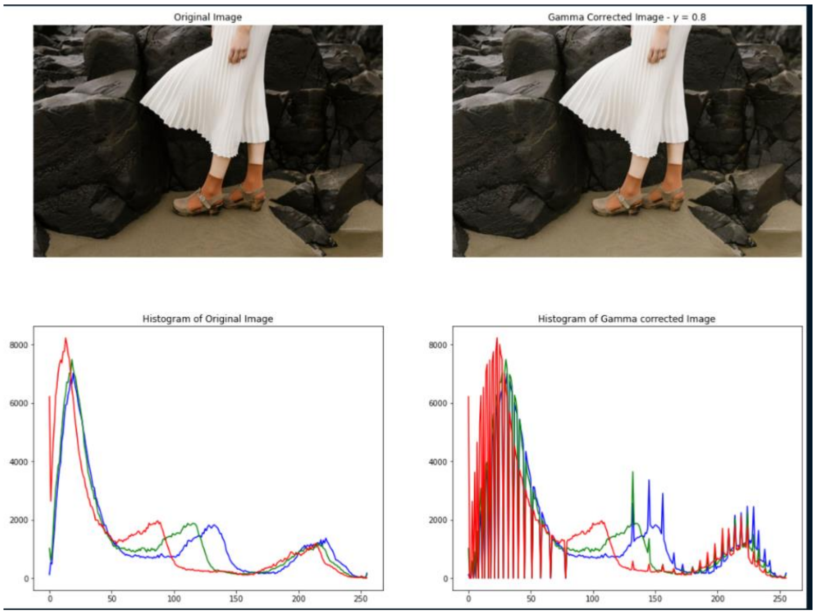

Fig 05

```python
gamma =0.8 # Selece Gamma Value

img_orig = cv.imread(r'./Images/highlights_and_shadows.jpg')

table = np.array([(i/255.0)**(gamma)*255.0 for i in np.arange(0,256)]).astype('uint8')
img_gamma = cv.LUT(img_orig,table)

img_orig = cv.cvtColor(img_orig,cv.COLOR_BGR2RGB)
img_gamma = cv.cvtColor(img_gamma,cv.COLOR_BGR2RGB)
```

Question 04

In a picture, number of pixels with same intensity is not equal. Therefore for a normal image this histogram is not flat, the process of making the histogram flat is called histogram equalization. This results image being more vibrant. Here are the results(Fig 06),

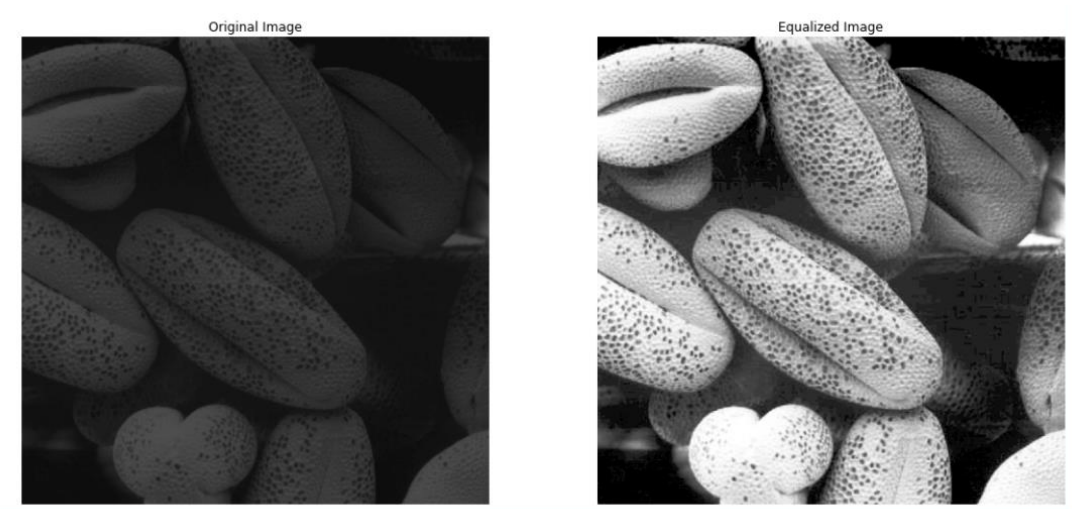

Fig 06

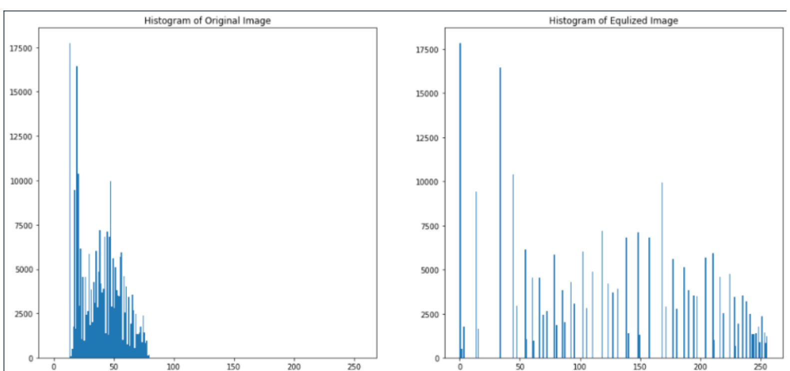

Fig 07

This figure (Fig 07)shows histogram of two images before and after equalization.

```python
img = cv.imread(r'./Images/shells.png',cv.IMREAD_GRAYSCALE)
assert img is not None
fig , ax = plt.subplots(2,2,figsize = (18,18),facecolor ='white')
equ = cv.equalizeHist(img)
```

Question 05

There are two methods to zoom an image, Nearest neighbor zooming and Bilinear interpolation. The more accurate one is using bilinear interpolation also it gives smoother image even if it is zoomed. But when we use Nearest neighbor zooming image get pixelated when size increased.

- Nearest neighbor zooming

```
S = 5

Rows = int(S*img.shape[0])
Coloumns = int(S*img.shape[1])
print(Rows,Coloumns)
img_zoom = np.zeros((Rows,Coloumns),dtype=img.dtype)

for r in range(0,Rows):
    for c in range(0,Coloumns):
        img_zoom[r,c] = img[int(r/S),int(c/S)]
```

Fig 08

Fig 09

Fig 08 shows original image and Fig 09 represent part of zoomed image, we can see that it is pixelated.

- Bilinear Interpolation

```
S = 2

org_R =img.shape[0]
org_C = img.shape[1]
Rows = int(S*org_R)
Coloumns = int(S*org_C)
print(Rows,Coloumns)
img_zoom = np.zeros((Rows,Coloumns),dtype=img.dtype)

for r in range(0,Rows):
    for c in range(0,Coloumns):
        R = r/S
        C = c/S
        i_R = int(R)
        i_C = int(C)
        d_R = R-i_R
        d_C = C-i_C
        if i_R==org_R-1 or i_C==org_C-1:
            img_zoom[r,c] = img[i_R,i_C]
            continue
        P1 = img[i_R,i_C]*(1-d_R)+img[i_R+1,i_C]*(d_R)
        P2 = img[i_R,i_C+1]*(1-d_R)+img[i_R+1,i_C+1]*(d_R)
        Pix_val = P1*(1-d_C)+P2*d_C

        img_zoom[r,c] = Pix_val
```

Question 06

In normal intensity transformation value of a pixel depended only on the same pixel. But, in Spatial filtering we consider neighbor pixels to calculate value to a pixel using a kernel. By using suitable kernel we can obtain blurred, Sobel horizontal and Soble vertical images. The data obtained from Soble filtering can be combined with original image to increase sharpness of the image. This figure(Fig 10) shows original image and Sobel filtered images

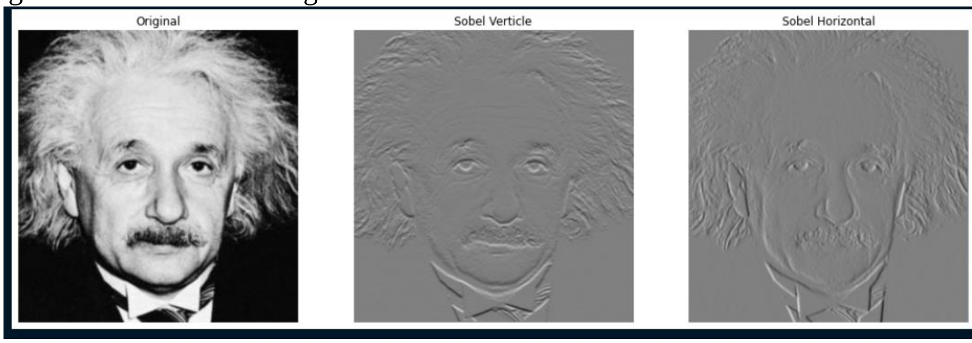

Fig 10

- Using inbuilt OpenCV function

```
img = cv.imread(r'./Images/einstein.png',cv.IMREAD_GRAYSCALE).astype(np.float32)
assert img is not None

sobel_v = np.array([[-1,-2,-1],[0,0,0],[1,2,1]],dtype= np.float32)
im_x = cv.filter2D(img,-1,sobel_v)

sobel_h = np.array([[-1,0,1],[-2,0,2],[-1,0,1]],dtype= np.float32)
im_y = cv.filter2D(img,-1,sobel_h)
```

- By defining own function

```
def filter2D(image, kernel):
    assert kernel.shape[0] % 2 and kernel.shape[1] % 2
    h_offset = kernel.shape[0]//2
    w_offset = kernel.shape[1]//2
    h, w = image.shape
    result = np.zeros(image.shape, dtype = np.float32)

    for r in range(h_offset, h-h_offset):
        for c in range(w_offset, w-w_offset):
            result[r][c] = np.dot(image[r-h_offset:r+h_offset+1, c-w_offset:c+w_offset+1].flatten(), kernel.flatten())

    return result


img = cv.imread(r'./Images/einstein.png',cv.IMREAD_GRAYSCALE).astype(np.float32)
assert img is not None

sobel_v = np.array([[-1,-2,-1],[0,0,0],[1,2,1]],dtype= np.float32)
im_x = filter2D(img,sobel_v)

sobel_h = np.array([[-1,0,1],[-2,0,2],[-1,0,1]],dtype= np.float32)
im_y = filter2D(img,sobel_h)
```

- By convolution

```
im = cv.imread('Images/einstein.png', cv.IMREAD_GRAYSCALE).astype(np.float32)
assert im is not None

k1_v = np.array([[-1],[0],[1]], dtype = np.float32)
im_intermediate_v = filter2D(im, k1_v)
k2_v = np.array([[1, 2, 1]] , dtype = np.float32)
im_sobelv = filter2D(im_intermediate_v, k2_v)

k1_h = np.array([[1], [2], [1]], dtype = np.float32)
im_intermediate_h = filter2D(im, k1_h)
k2_h = np.array([[-1, 0, 1]] , dtype = np.float32)
im_sobelh = filter2D(im_intermediate_h, k2_h)
```

Question 07

Using inbuilt function grabCut in OpenCV library we can mask out specific object in defined area in our image. Also, we can remove the background of that image. In part (a) in this question we separate flower from the image. Masked foreground in background is showed in following figure.(Fig 11)

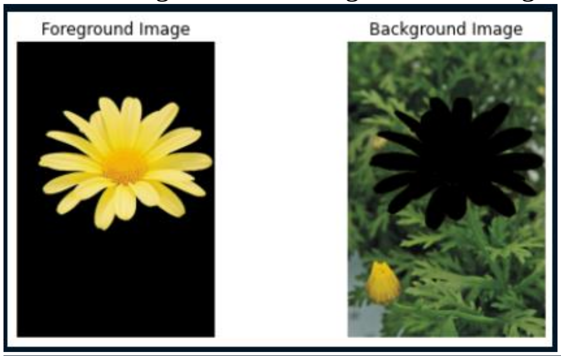

Fig 11

```python
img = cv.imread('images/daisy.jpg', cv.IMREAD_COLOR)
assert img is not None

mask = np.zeros(img.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
rect = (60, 150, 500, 400)
cv.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv.GC_INIT_WITH_RECT)

#foreground
mask_foreground = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img_foreground = img * mask2[:, :, np.newaxis]

#background
mask_background = np.where((mask==1)|(mask==3),0,1).astype('uint8')
img_background = img * mask3[:, :, np.newaxis]
```

Masks of those two images are showed in below,

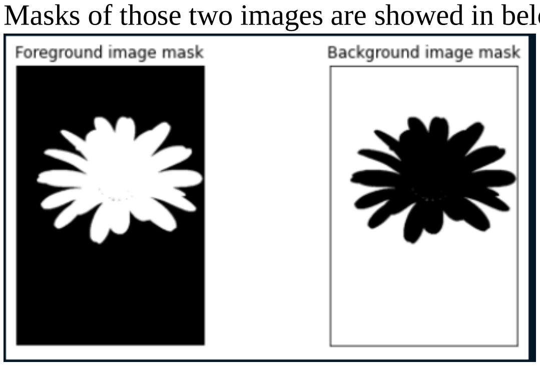

Fig 12

In part (b) of this question, first we blur the masked background of the image and combine it with masked foreground, then we can we that flower is focused.

```python
img = cv.imread('images/daisy.jpg', cv.IMREAD_COLOR)
assert img is not None

k = 9
sigma = 3
blured_img = cv.GaussianBlur(img_background, (k, k), sigma)
enhanced_img = img_foreground + blured_img
```