EN3160 - Assignment 1 on Intensity Transformations and Neighborhood Filtering

200310E – Tharaka Kodithuwakku

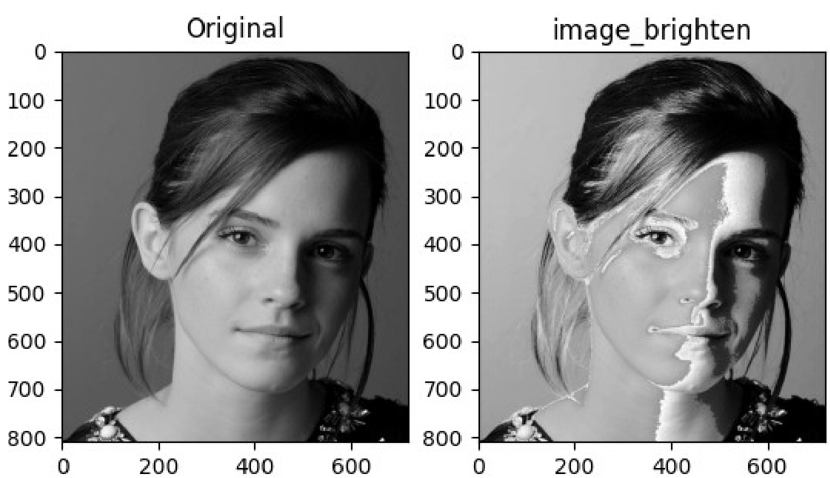
01) Implement the intensity transformation

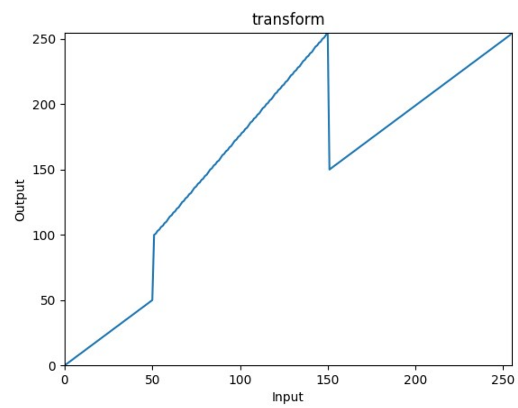
```
c = np.array([(50,50),(50,100),(150,255),(150,150)] ) # take cordinates (x,y) into an array

#make whole function as parts and then combined everything
t1=np.linspace(0,c[0,1],c[0,0]+1-0)
t2=np.linspace(c[0,1],c[1,1],c[1,0]-c[0,0])
t3=np.linspace(c[1,1],c[2,1],c[2,0]-c[1,0])
t4=np.linspace(c[2,1],c[3,1],c[3,0]-c[2,0])
t5=np.linspace(c[3,1],255,255-c[3,0])

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')
```

List out cordinates

Build the whole transformation as part by part

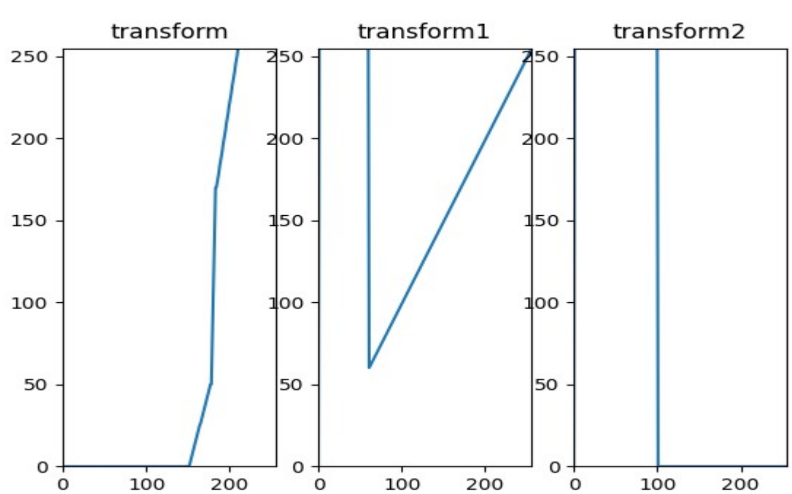
Merge them to make whole transform

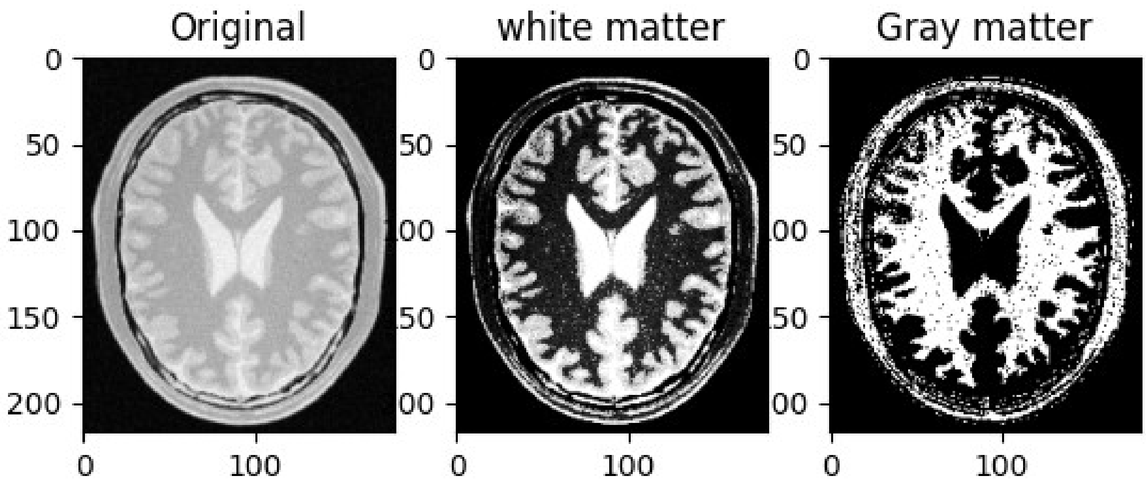




02) Accentuate white matter gray matter in the brain proton density image

```
image_transformed = cv.LUT(img_original,transform)
image_transformed21 = cv.LUT(img_original,transform1)
image_transformed22 = cv.LUT(image_transformed21,transform)
image_transformed2 = cv.LUT(image_transformed22,transform2)
```

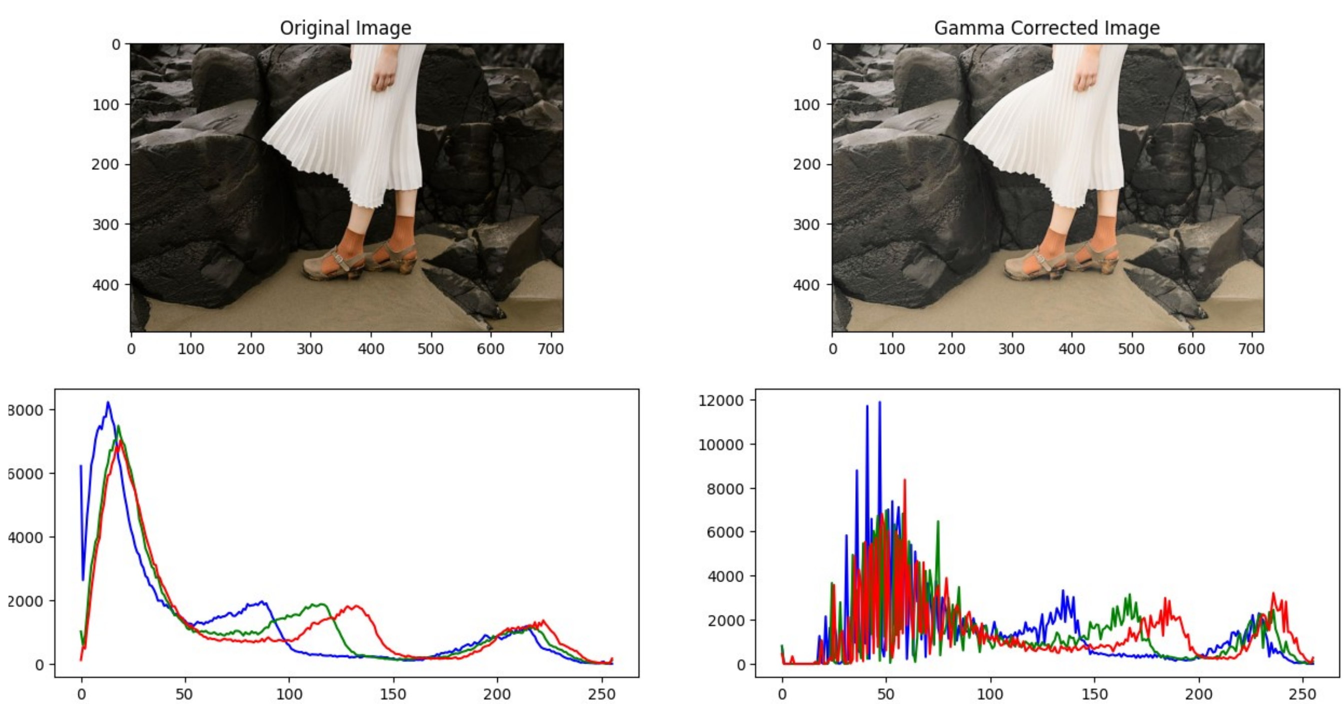




03) Apply gamma correction to the L plane in the L∗a∗b∗ color space

```
# Gamma correction factor
gamma = 0.51

# Convert BGR image to Lab color space
img_lab = cv.cvtColor(img_org, cv.COLOR_BGR2Lab)

# Split the Lab image into L, a, and b channels
L, a, b = cv.split(img_lab)

# Calculate the lookup table for gamma correction
table = np.array([(i / 255.0) ** (gamma) * 255.0 for i in np.arange(0, 256)]).astype('uint8')

# Apply gamma correction to the L channel using the lookup table
L_corrected = cv.LUT(L, table)
```

```
color = ('b','g','r')
for i, c in enumerate(color):
    hist_orig = cv.calcHist([img_org], [i], None, [256], [0, 256])
    axarr[1, 0].plot(hist_orig, color=c)
    hist_gamma = cv.calcHist([img_gamma_corrected], [i], None, [256], [0, 256])
    axarr[1, 1].plot(hist_gamma, color=c)
```



04) Increasing the vibrance of a photograph

a)

```
img_n = cv.cvtColor(img,cv.COLOR_BGR2RGB)
img_hsv = cv.cvtColor(img_n, cv.COLOR_BGR2HSV)
img_hsv_n = cv.cvtColor(img_hsv,cv.COLOR_BGR2RGB)

# Split the HSV image into H, S, and V channels
H, S, V = cv.split(img_hsv)
```

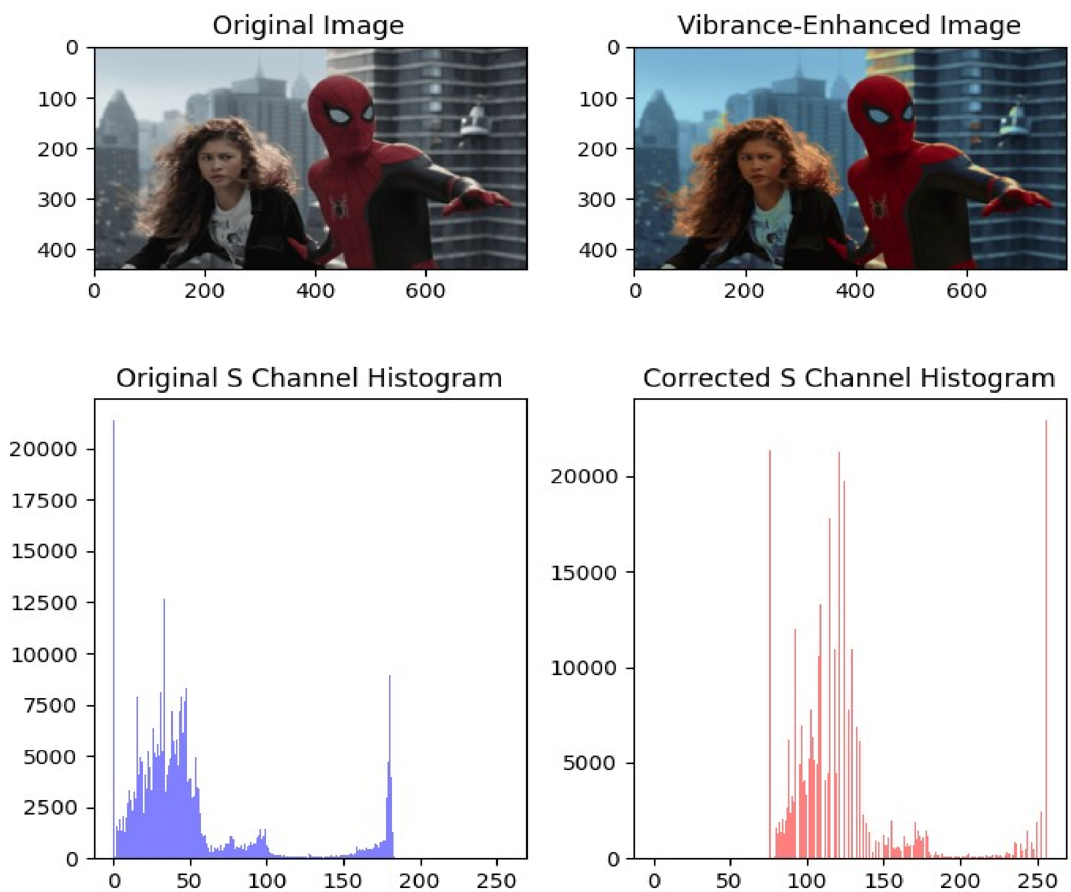
b)

```
# Apply the saturation adjustment formula
a = 0.6  # Adjust this parameter as needed
sigma = 70  # Adjust this parameter as needed
S_adjustment = a * 128 * np.exp(-(S - 128) ** 2 / (2 * sigma ** 2)) ## s + s_adj
S_corrected = np.minimum(np.maximum(S + S_adjustment, 0), 255).astype('uint8')
```

d)

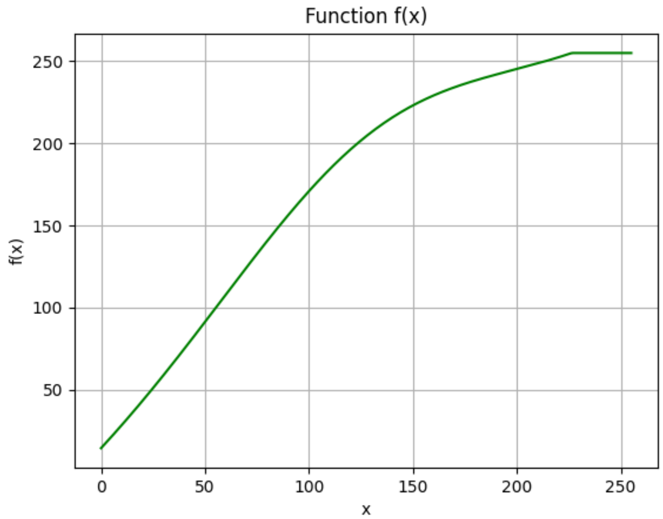
```
# Merge the corrected S channel with the original H and V channels
img_hsv_corrected = cv.merge((H, S_corrected, V))
```

e)

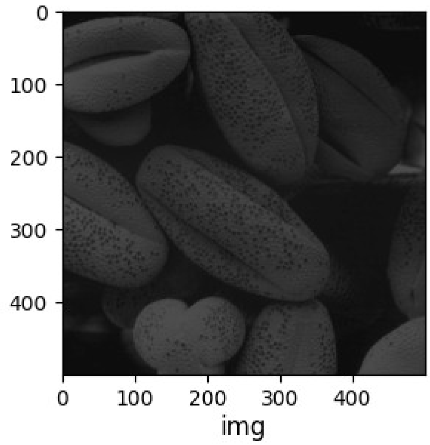


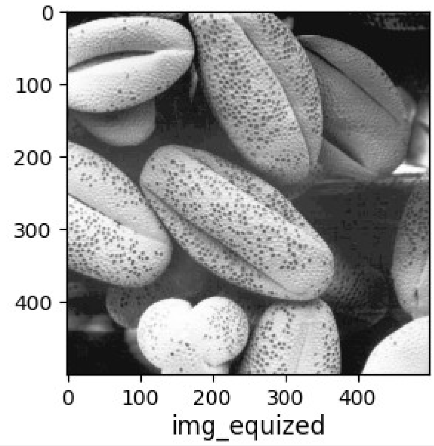
As the parameter "a" gets closer to 1, it enhances the vibrancy or saturation of the image. When "a" is set to 0.6, it strikes a balance that maintains a more natural appearance without excessive vibrancy. Adjusting the "a" value allows you to control the level of saturation enhancement in the image, with lower values producing a more subdued effect and higher values making the colors appear more vibrant.

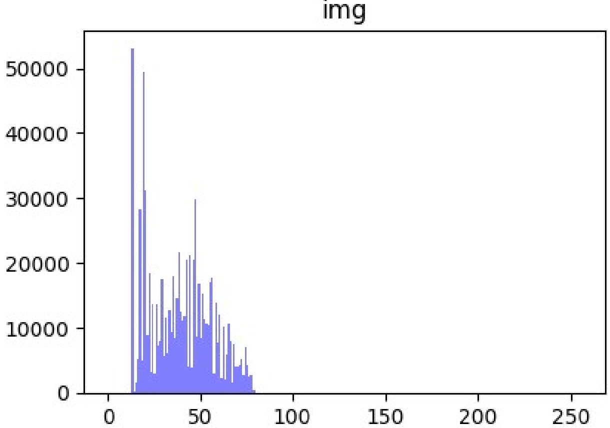
e)

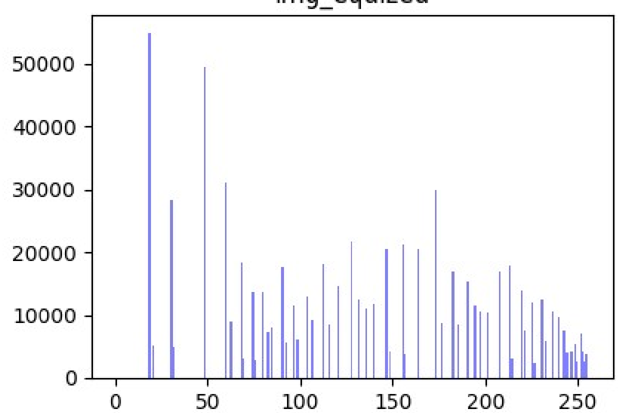


05) A function of histogram equalization










```python
def histogram_equalization(image, num_bins=256):
    # Calculate histogram of the input image
    histogram, bins = np.histogram(image.flatten(), bins=num_bins, range=[0, 256])
    # Calculate cumulative distribution function (CDF)
    cdf = histogram.cumsum()
    # Normalize CDF to the dynamic range of the image
    cdf_normalized = cdf * (num_bins - 1) / cdf[-1]
    # Map original intensities to new equalized intensities
    equalized_image = np.interp(image.flatten(), bins[:-1], cdf_normalized)
    # Reshape to the original image shape
    equalized_image = equalized_image.reshape(image.shape)
    return equalized_image.astype(np.uint8)
```

06) Apply histogram equalization only to the foreground of an image to produce a image with a histogram equalized foreground

a)

```
# Split the HSV image into H, S, and V component
H, S, V = cv.split(hsv_img)
```

b)

```
threshold_value = 13  # determines the point at which the pixels will be classif
foreground_mask = cv.threshold(S, threshold_value, 255, cv.THRESH_BINARY)[1]
```

c)

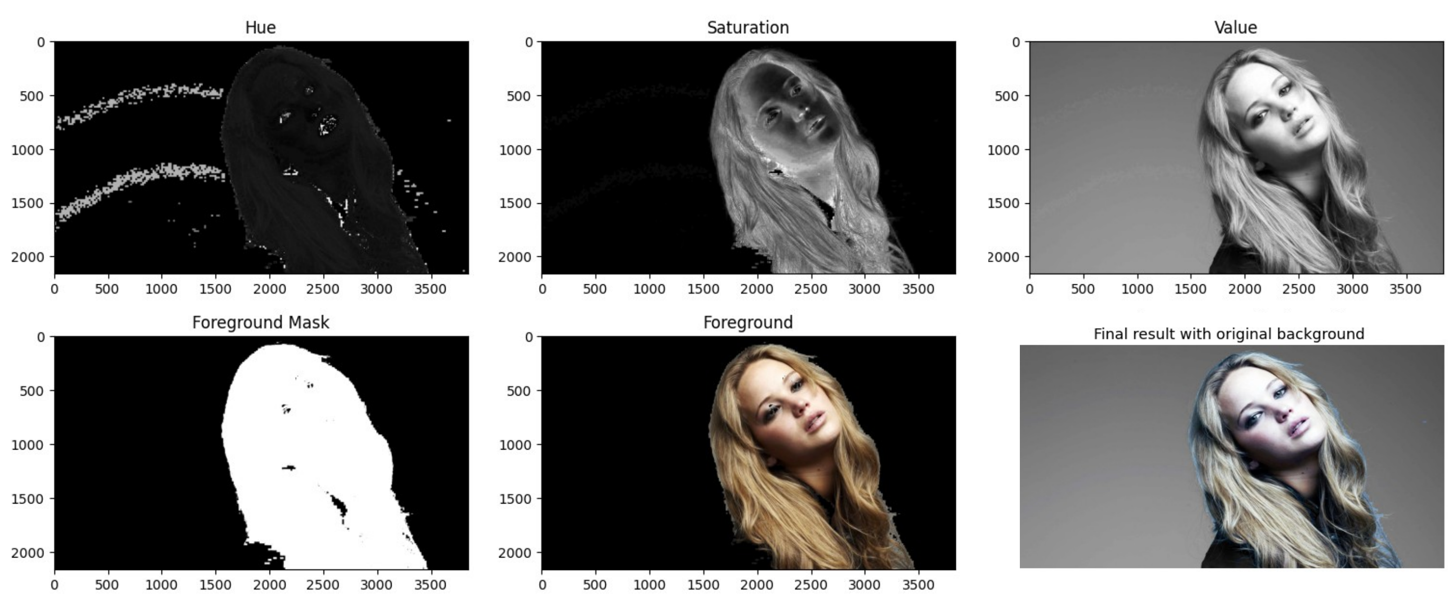
```
B, G, R = cv.split(img)

# Extract the foreground for each color channel
foreground_B = cv.bitwise_and(B, B, mask=foreground_mask)
foreground_G = cv.bitwise_and(G, G, mask=foreground_mask)
foreground_R = cv.bitwise_and(R, R, mask=foreground_mask)

# Merge the foreground channels back into a single image
foreground = cv.merge([foreground_R, foreground_G, foreground_B])
```

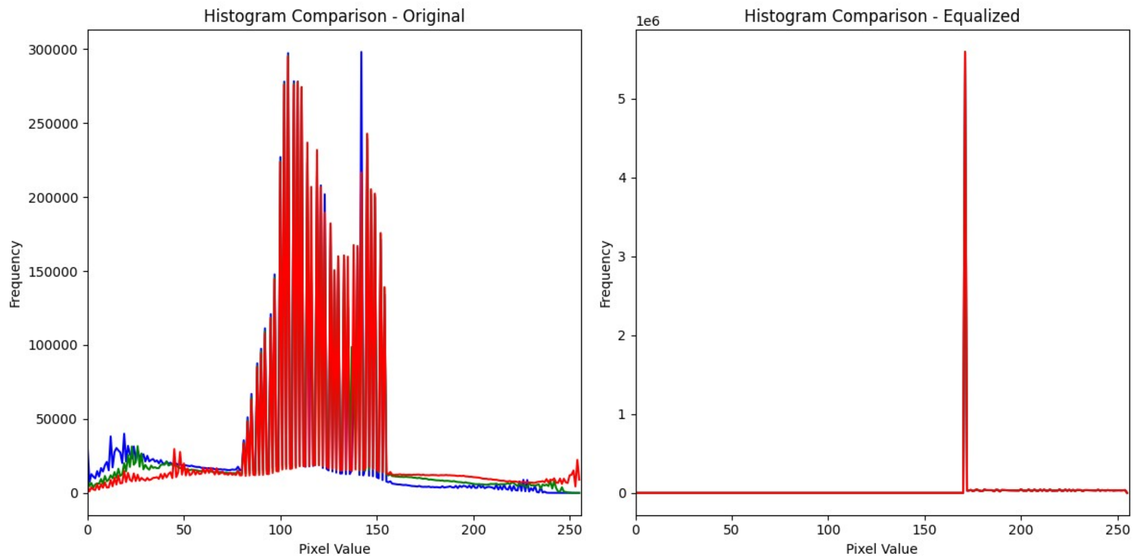
d)

```
cumulative_sum = np.cumsum(hist_foreground)
print(cumulative_sum)
```

e)

```
background_mask_3d = 255 - foreground_mask
background_hsv = np.bitwise_and(hsv_img, background_mask_3d)   # Extrac
background_rgb = cv.cvtColor(background_hsv, cv.COLOR_HSV2RGB)
final_image = background_rgb + equalized_result      # Add with foregrou
```



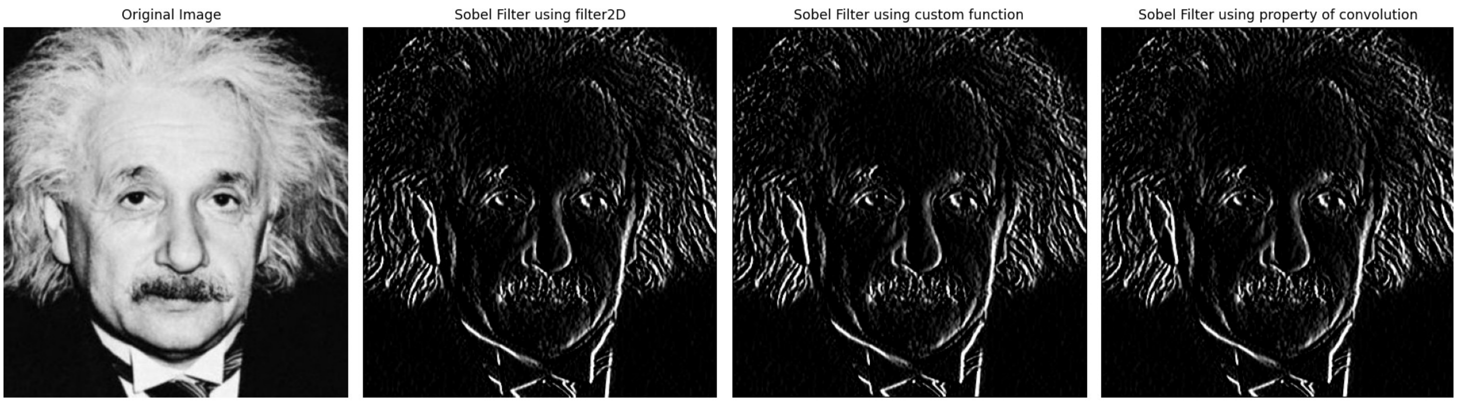


07) Filtering with the Sobel operator

```
def filter(image , kernel):
    assert kernel.shape[0]%2 == 1 and kernel.shape[1]%2 == 1
    k_hh, k_hw = kernel.shape[0] // 2, kernel.shape[1] // 2
    h, w = image.shape
    image_float = cv.normalize(image.astype('float'), None, 0, 1, cv.NORM_MINMAX)
    result = np.zeros(image.shape, 'float')

    for m in range(k_hh, h - k_hh):
        for n in range(k_hw, w - k_hw):
            result[m, n] = np.dot(image_float[m-k_hh: m+k_hh+1, n-k_hw: n+k_hw+1].flatten(), kernel.flatten())

    result = result * 255   # Undo normalization
    result = np.minimum(255, np.maximum(0, result)).astype(np.uint8) # Limit between 0 and 255
    return result
```

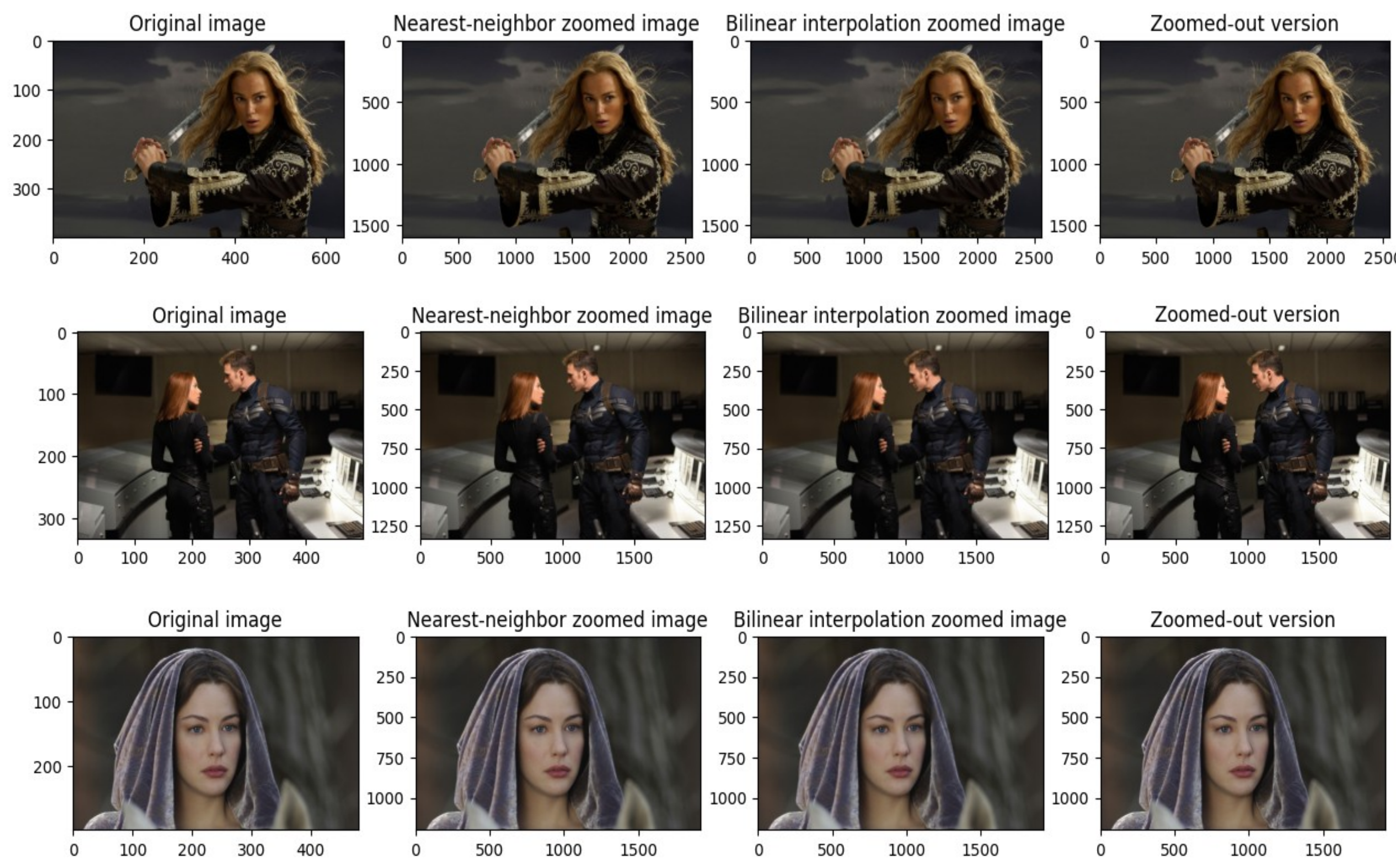
```
def filter_in_steps(image, kernel1, kernel2):

    image_float = cv.normalize(image.astype('float'), None, 0, 1, cv.NORM_MINMAX)
    result = filter_step(filter_step(image_float, kernel1), kernel2)
    result = result * 255
    result = np.minimum(255, np.maximum(0, result)).astype(np.uint8) # Limit betwee
    return result
```

```
kernel = np.array([(-1,-2,-1),(0,0,0),(1,2,1)],dtype='float')
imgc = cv.filter2D(img,-1,kernel)
```



08) A program to zoom images by a given factor s ∈ (0,10)

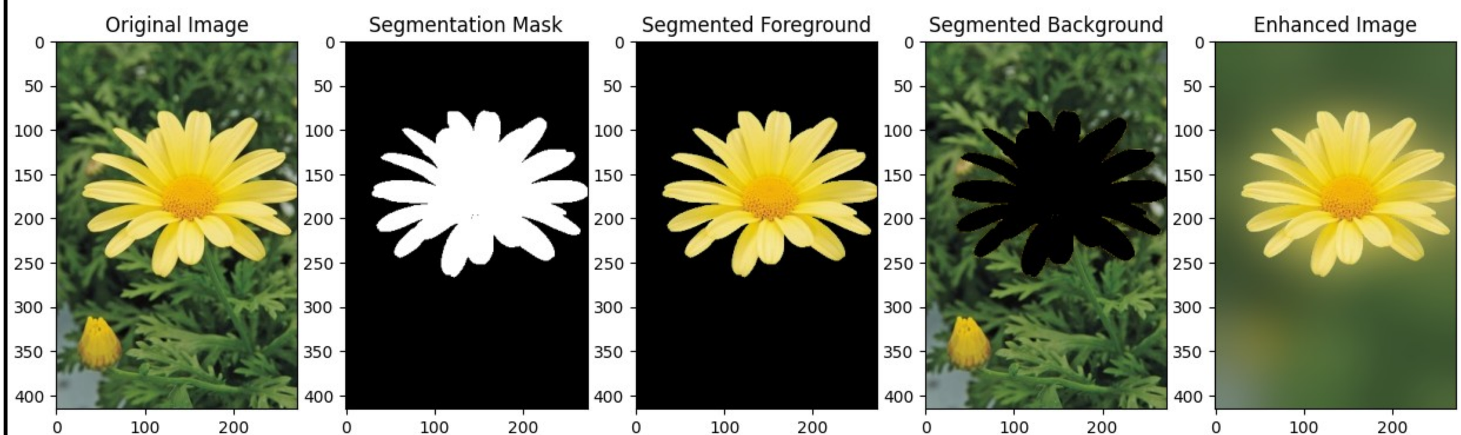
```
def images_set():
 for j in range(4):
    image = cv.imread(original_images[j])
    image_zoom_out = cv.imread(zoom_outs[j])

    image_bilinear = cv.resize(image, None, fx=4, fy=4, interpolation=cv.INTER_LINEAR)
    image_near = cv.resize(image, None, fx=4, fy=4, interpolation=cv.INTER_NEAREST)
```



09) A flower image problem

```python
# (a) Perform image segmentation
segmentation_mask = np.zeros(original_image.shape[:2], dtype=np.uint8)

# Define a region of interest (ROI) within the image
roi_rect = (30, 30, original_image.shape[1] - 30, original_image.shape[0] - 150)
background_model = np.zeros((1, 65), dtype=np.float64)
foreground_model = np.zeros((1, 65), dtype=np.float64)
cv.grabCut(original_image, segmentation_mask, roi_rect, background_model, foreground_model, 5, cv.GC_INIT_WITH_RECT)

# Create a binary mask where 1 represents the foreground and 0 represents the background
binary_mask = np.where((segmentation_mask == 2) | (segmentation_mask == 0), 0, 1).astype('uint8')

# Apply the mask to the original image to extract the segmented foreground
segmented_foreground = original_image * binary_mask[:, :, np.newaxis]
```

```python
# (b) Enhance the image
blurred_background = cv.GaussianBlur(original_image, (0, 0), 30)
enhanced_image = np.where(binary_mask[:, :, np.newaxis] == 1, original_image, blurred_background)
```



The observed dark edges in the enhanced image are a result of the blending process during Gaussian blur, where the black pixels within the mask boundary are combined with the background pixels along the mask's border. This blending can lead to an undesirable visual effect, causing the edges to appear darker than intended.

Github Link - https://github.com/baymax06in19/EN-3160

* * *