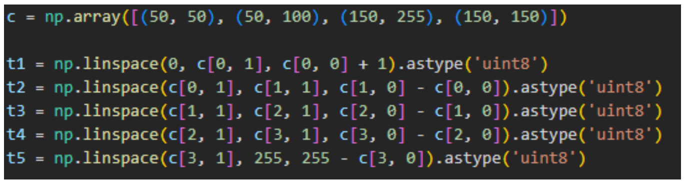
EN3160 Assignment 1- I.P.D.D.Rajapaksha 210503H

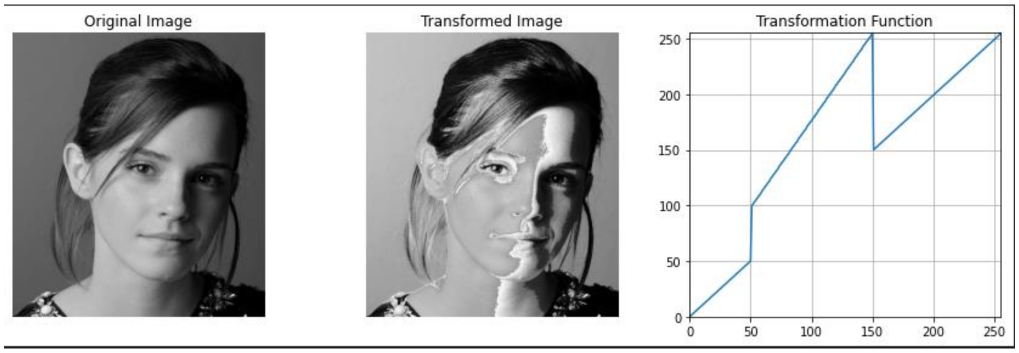
GitHub : https://github.com/DinethraDivanjana2001/EN3166-Image-processing-and-machine-vision/tree/main

1.0. Intensity transformation

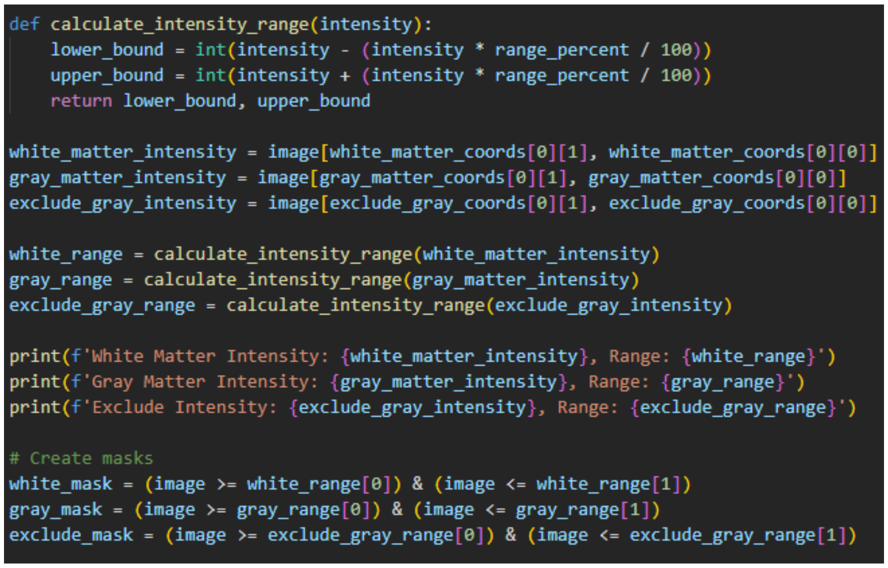
Transformation function



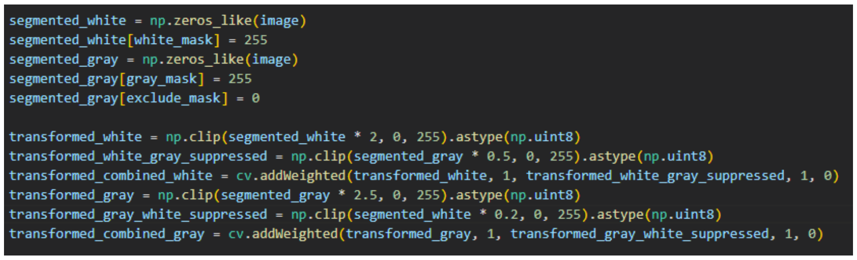
```
c = np.array([(50, 50), (50, 100), (150, 255), (150, 150)])

t1 = np.linspace(0, c[0, 1], c[0, 0] + 1).astype('uint8')
t2 = np.linspace(c[0, 1], c[1, 1], c[1, 0] - c[0, 0]).astype('uint8')
t3 = np.linspace(c[1, 1], c[2, 1], c[2, 0] - c[1, 0]).astype('uint8')
t4 = np.linspace(c[2, 1], c[3, 1], c[3, 0] - c[2, 0]).astype('uint8')
t5 = np.linspace(c[3, 1], 255, 255 - c[3, 0]).astype('uint8')
```



2.0. Brain image white and grey matter separation and intensity transformation

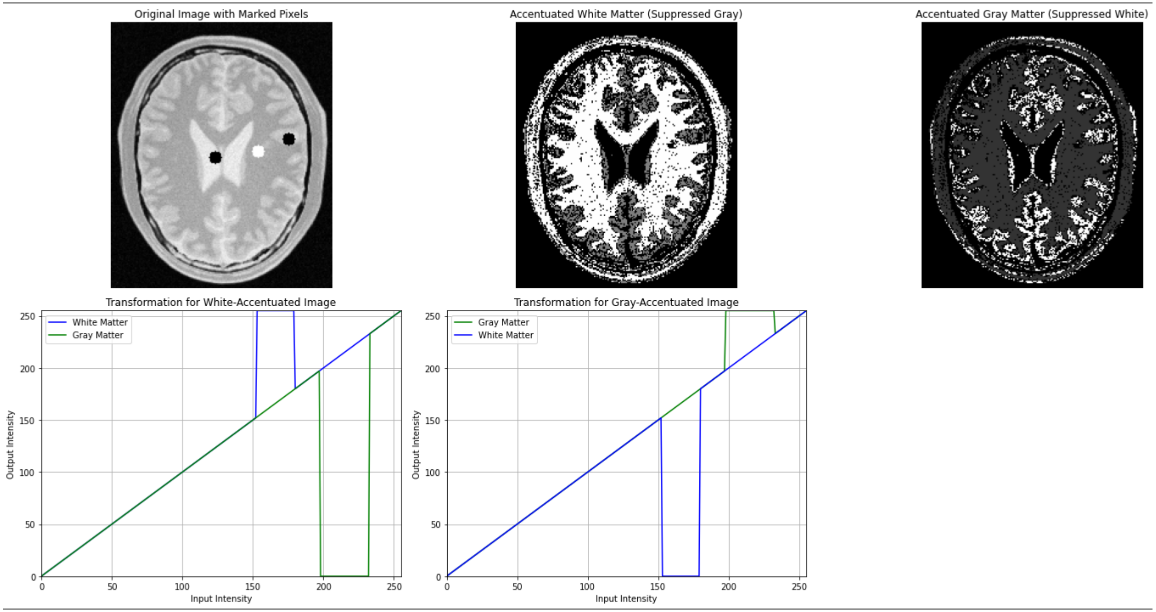


```
def calculate_intensity_range(intensity):
    lower_bound = int(intensity - (intensity * range_percent / 100))
    upper_bound = int(intensity + (intensity * range_percent / 100))
    return lower_bound, upper_bound

white_matter_intensity = image[white_matter_coords[0][1], white_matter_coords[0][0]]
gray_matter_intensity = image[gray_matter_coords[0][1], gray_matter_coords[0][0]]
exclude_gray_intensity = image[exclude_gray_coords[0][1], exclude_gray_coords[0][0]]

white_range = calculate_intensity_range(white_matter_intensity)
gray_range = calculate_intensity_range(gray_matter_intensity)
exclude_gray_range = calculate_intensity_range(exclude_gray_intensity)

print(f'White Matter Intensity: {white_matter_intensity}, Range: {white_range}')
print(f'Gray Matter Intensity: {gray_matter_intensity}, Range: {gray_range}')
print(f'Exclude Intensity: {exclude_gray_intensity}, Range: {exclude_gray_range}')

# Create masks
white_mask = (image >= white_range[0]) & (image <= white_range[1])
gray_mask = (image >= gray_range[0]) & (image <= gray_range[1])
exclude_mask = (image >= exclude_gray_range[0]) & (image <= exclude_gray_range[1])
```



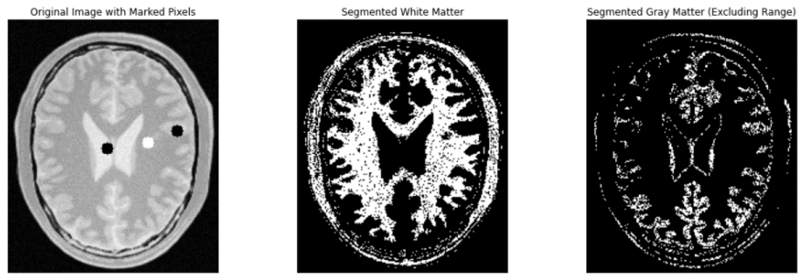
```python
segmented_white = np.zeros_like(image)
segmented_white[white_mask] = 255
segmented_gray = np.zeros_like(image)
segmented_gray[gray_mask] = 255
segmented_gray[exclude_mask] = 0

transformed_white = np.clip(segmented_white * 2, 0, 255).astype(np.uint8)
transformed_white_gray_suppressed = np.clip(segmented_gray * 0.5, 0, 255).astype(np.uint8)
transformed_combined_white = cv.addWeighted(transformed_white, 1, transformed_white_gray_suppressed, 1, 0)
transformed_gray = np.clip(segmented_gray * 2.5, 0, 255).astype(np.uint8)
transformed_gray_white_suppressed = np.clip(segmented_white * 0.2, 0, 255).astype(np.uint8)
transformed_combined_gray = cv.addWeighted(transformed_gray, 1, transformed_gray_white_suppressed, 1, 0)
```

Here I got the two points in the brain image as reference and located one in the white matter area and the second one in the grey matter area then took an intensity range for both of them as 7 % and using that range separated the two areas. Then, I did the intensity transformation.

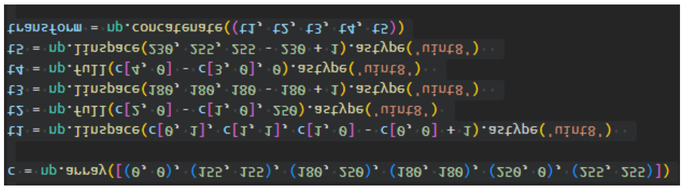

Original Image with Marked Pixels
Accentuated White Matter (Suppressed Gray)
Accentuated Gray Matter (Suppressed White)
Transformation for White-Accentuated Image
White Matter
Gray Matter
Output Intensity
Input Intensity
Transformation for Gray-Accentuated Image
Gray Matter
White Matter
Output Intensity
Input Intensity

But middle part should not include to any of those area so remove that part by getting another intensity value from middle point but it reduce the accuracy of the grey matter area.

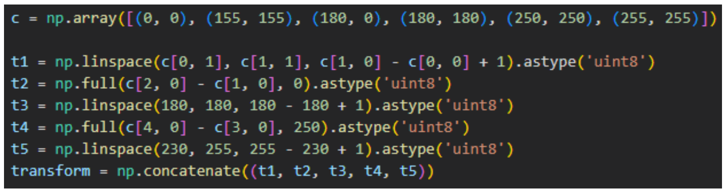
Then


Original Image with Marked Pixels
Segmented White Matter
Segmented Gray Matter (Excluding Range)

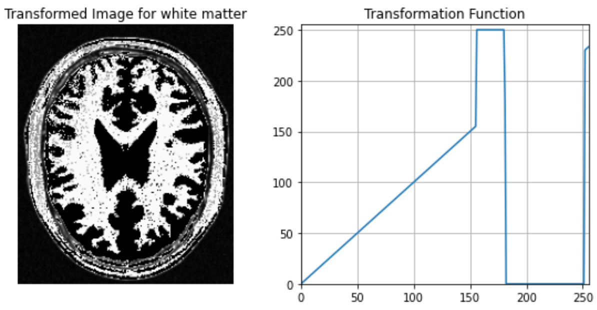
define the intensity transformation function as below and get the output sing previous method.

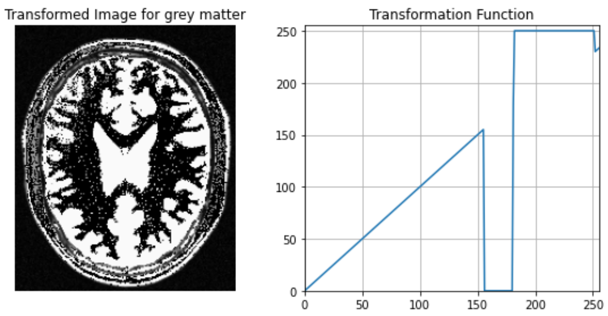


```python
c = np.array([(0, 0), (155, 155), (180, 250), (180, 180), (250, 0), (255, 255)])

t1 = np.linspace(c[0, 1], c[1, 1], c[1, 0] - c[0, 0] + 1).astype('uint8')
t2 = np.full(c[2, 0] - c[1, 0], 250).astype('uint8')
t3 = np.linspace(180, 180, 180 - 180 + 1).astype('uint8')
t4 = np.full(c[4, 0] - c[3, 0], 0).astype('uint8')
t5 = np.linspace(230, 255, 255 - 230 + 1).astype('uint8')
transform = np.concatenate((t1, t2, t3, t4, t5))
```



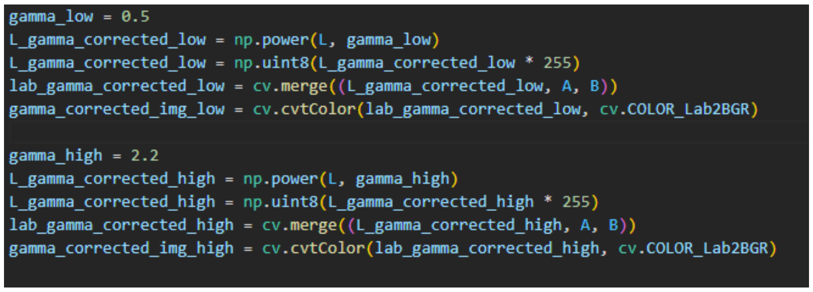
```python
c = np.array([(0, 0), (155, 155), (180, 0), (180, 180), (250, 250), (255, 255)])

t1 = np.linspace(c[0, 1], c[1, 1], c[1, 0] - c[0, 0] + 1).astype('uint8')
t2 = np.full(c[2, 0] - c[1, 0], 0).astype('uint8')
t3 = np.linspace(180, 180, 180 - 180 + 1).astype('uint8')
t4 = np.full(c[4, 0] - c[3, 0], 250).astype('uint8')
t5 = np.linspace(230, 255, 255 - 230 + 1).astype('uint8')
transform = np.concatenate((t1, t2, t3, t4, t5))
```



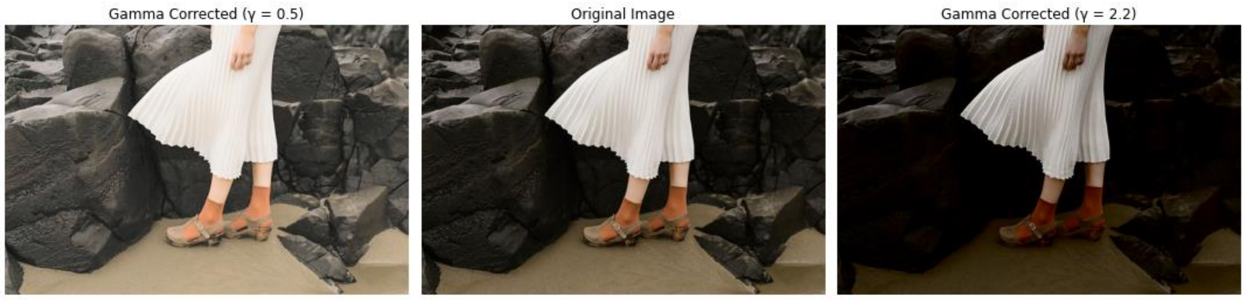


3.0. Gamma Correction and Histogram Analysis

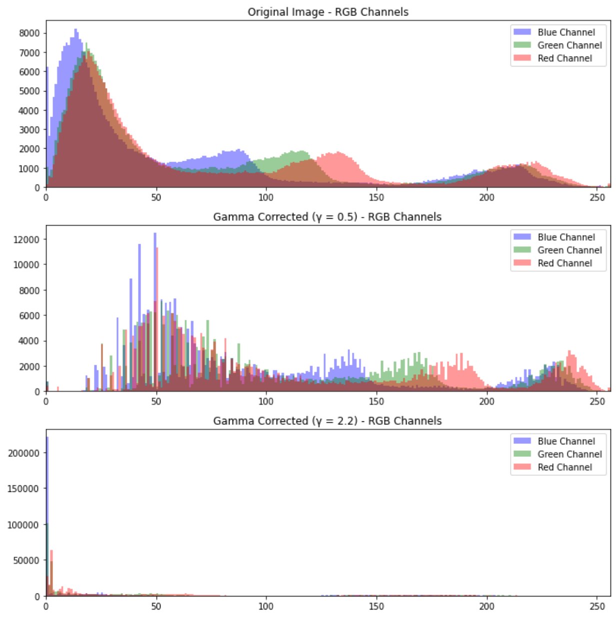
a).



```
gamma_low = 0.5
L_gamma_corrected_low = np.power(L, gamma_low)
L_gamma_corrected_low = np.uint8(L_gamma_corrected_low * 255)
lab_gamma_corrected_low = cv.merge((L_gamma_corrected_low, A, B))
gamma_corrected_img_low = cv.cvtColor(lab_gamma_corrected_low, cv.COLOR_Lab2BGR)

gamma_high = 2.2
L_gamma_corrected_high = np.power(L, gamma_high)
L_gamma_corrected_high = np.uint8(L_gamma_corrected_high * 255)
lab_gamma_corrected_high = cv.merge((L_gamma_corrected_high, A, B))
gamma_corrected_img_high = cv.cvtColor(lab_gamma_corrected_high, cv.COLOR_Lab2BGR)
```



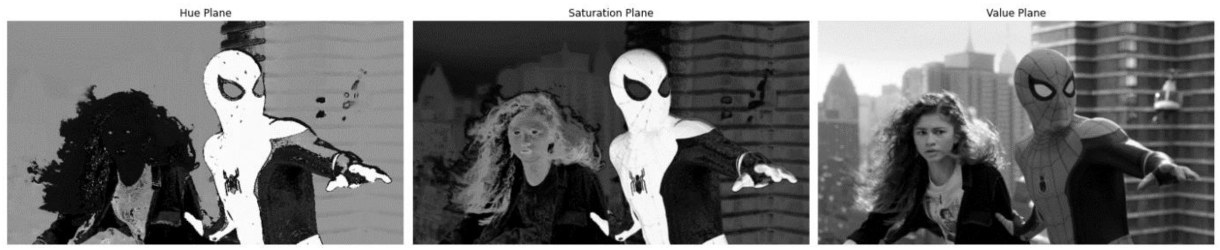
b).



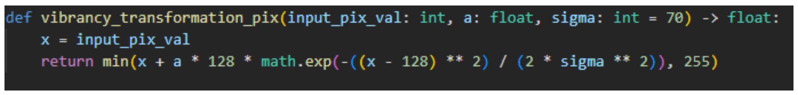
First applied gamma correction to the luminance (L) channel of an image converted to the Lab* color space. Then used two gamma values, γ=0.5 and γ=2.2, to enhance brightness and contrast. The process began by converting the image from BGR to Lab* color space. After separating the L, a, and b channels, applied gamma correction to the L channel using both values. Once the correction was complete, merged the adjusted L channel with the original a and b channels. Finally, converted the combined result back to BGR format for display.

4.0. Increasing the vibrance of a photograph

a). convert it to HSV color space, extract and display the Hue, Saturation, and Value channels

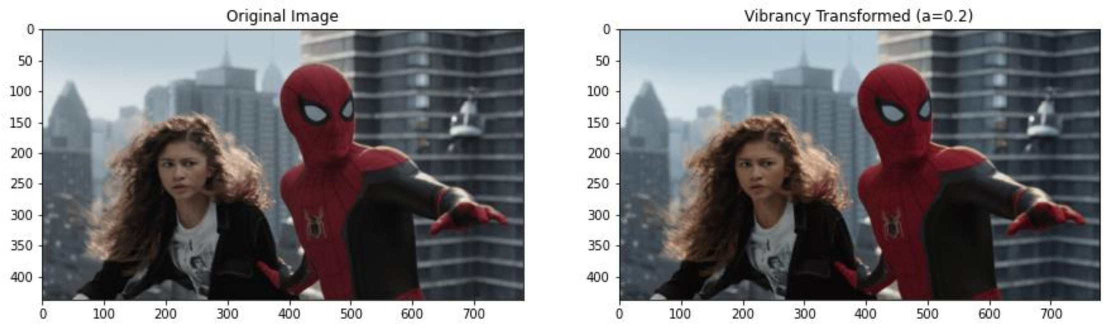


b). apply intensity transformation to channel



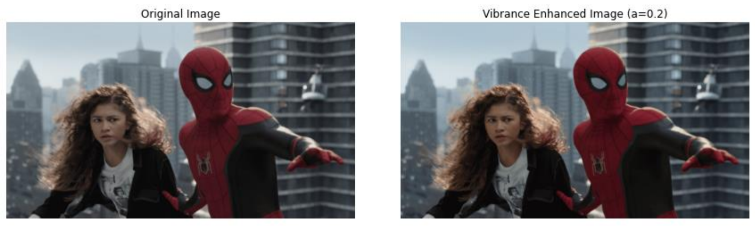
```python
def vibrancy_transformation_pix(input_pix_val: int, a: float, sigma: int = 70) -> float:
    x = input_pix_val
    return min(x + a * 128 * math.exp(-((x - 128) ** 2) / (2 * sigma ** 2)), 255)
```

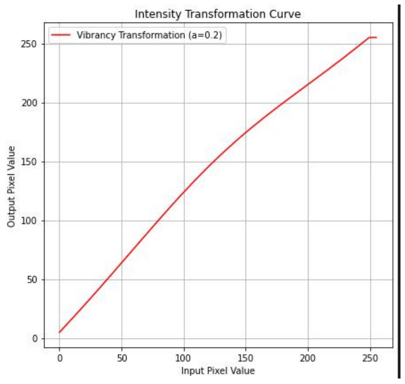
c).adjust a value and check what is the best by comparing the images with the original one.



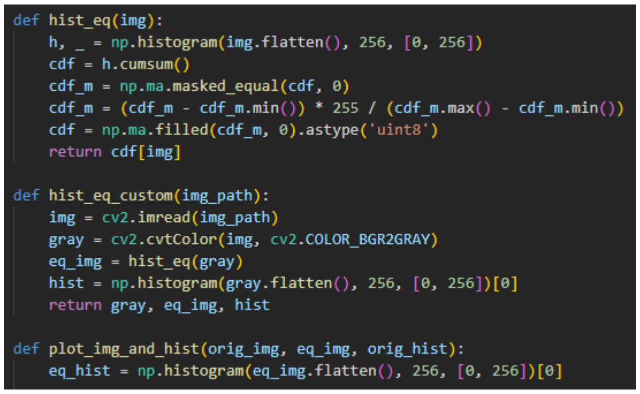
Clearly see when a=0.2 is the best option

Then recombined frame image used to get the vibrance-enhanced image, and the intensity transformation

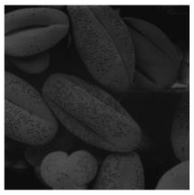


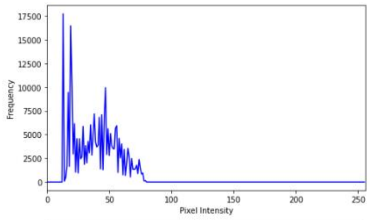


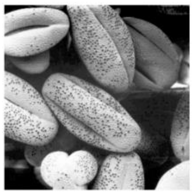
5.0. histograms before and after equalization.

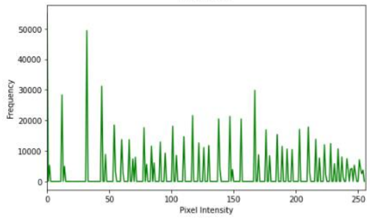


```
def hist_eq(img):
    h, _ = np.histogram(img.flatten(), 256, [0, 256])
    cdf = h.cumsum()
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
    cdf = np.ma.filled(cdf_m, 0).astype('uint8')
    return cdf[img]

def hist_eq_custom(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    eq_img = hist_eq(gray)
    hist = np.histogram(gray.flatten(), 256, [0, 256])[0]
    return gray, eq_img, hist

def plot_img_and_hist(orig_img, eq_img, orig_hist):
    eq_hist = np.histogram(eq_img.flatten(), 256, [0, 256])[0]
```





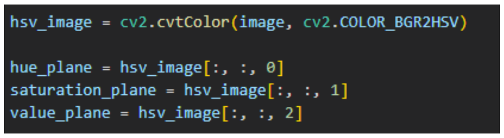




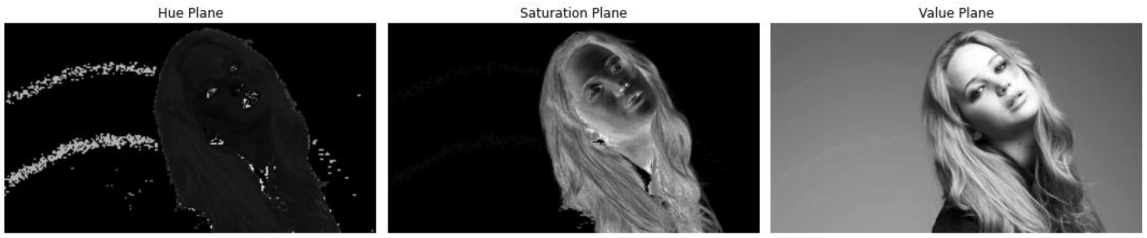
After equalization image becomes brighter and can clearly see in the graph pixel intensity increase as expected.

6.0.

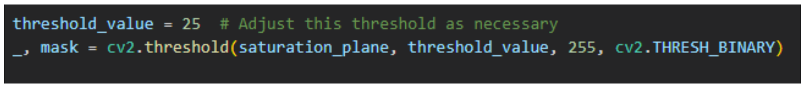
a).Load Image and Split into hue, saturate and value Planes



```
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

hue_plane = hsv_image[:, :, 0]
saturation_plane = hsv_image[:, :, 1]
value_plane = hsv_image[:, :, 2]
```

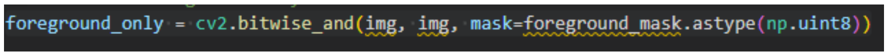


b).Next Create Foreground Mask

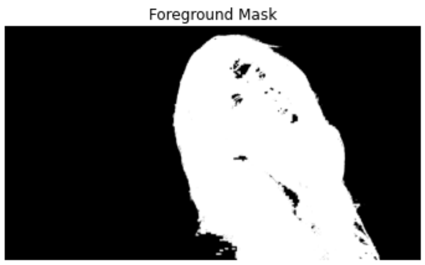


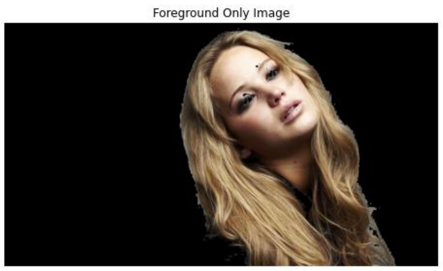
```
threshold_value = 25  # Adjust this threshold as necessary
_, mask = cv2.threshold(saturation_plane, threshold_value, 255, cv2.THRESH_BINARY)
```

c).Then extract foreground



```
foreground_only = cv2.bitwise_and(img, img, mask=foreground_mask.astype(np.uint8))
```

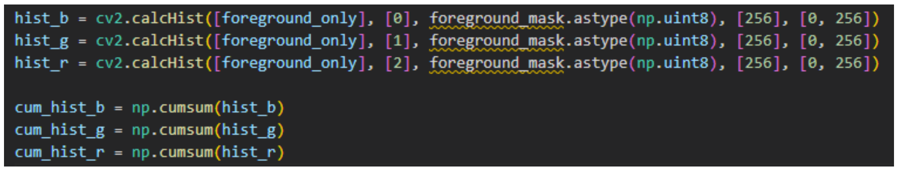




b)

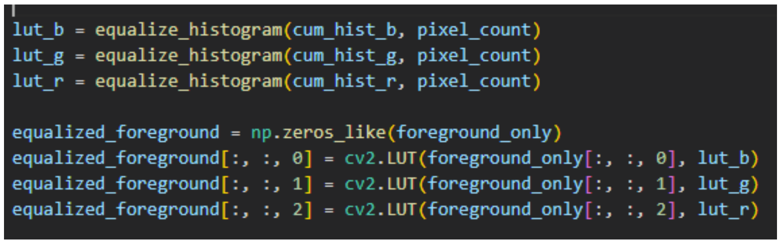
c)

d).compute the histogram the botain cumulative sum



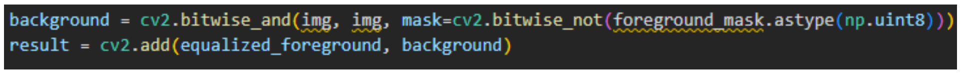
```
hist_b = cv2.calcHist([foreground_only], [0], foreground_mask.astype(np.uint8), [256], [0, 256])
hist_g = cv2.calcHist([foreground_only], [1], foreground_mask.astype(np.uint8), [256], [0, 256])
hist_r = cv2.calcHist([foreground_only], [2], foreground_mask.astype(np.uint8), [256], [0, 256])

cum_hist_b = np.cumsum(hist_b)
cum_hist_g = np.cumsum(hist_g)
cum_hist_r = np.cumsum(hist_r)
```

e). histogram-equalize the foreground.

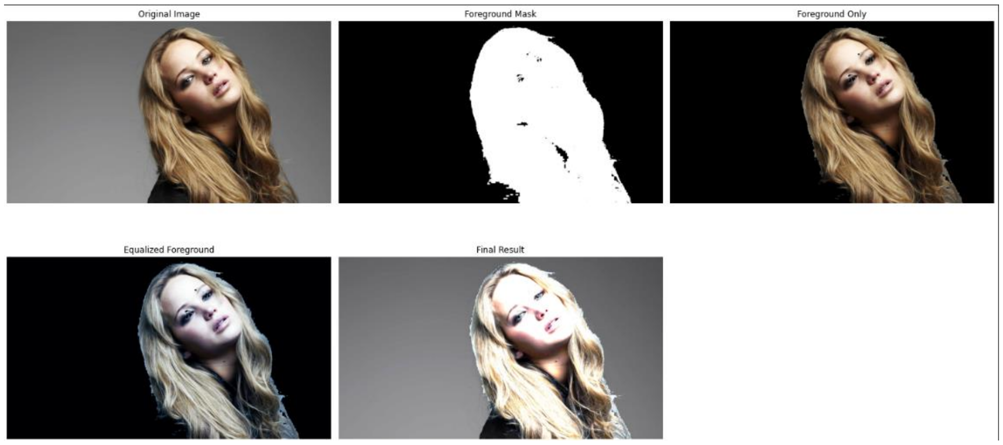


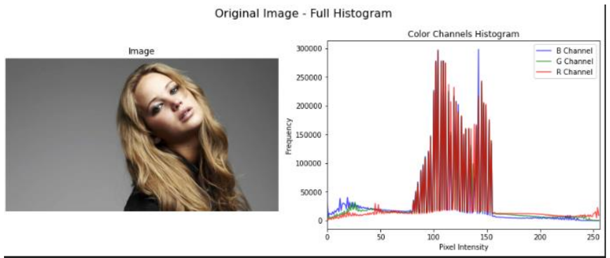
```
lut_b = equalize_histogram(cum_hist_b, pixel_count)
lut_g = equalize_histogram(cum_hist_g, pixel_count)
lut_r = equalize_histogram(cum_hist_r, pixel_count)

equalized_foreground = np.zeros_like(foreground_only)
equalized_foreground[:, :, 0] = cv2.LUT(foreground_only[:, :, 0], lut_b)
equalized_foreground[:, :, 1] = cv2.LUT(foreground_only[:, :, 1], lut_g)
equalized_foreground[:, :, 2] = cv2.LUT(foreground_only[:, :, 2], lut_r)
```

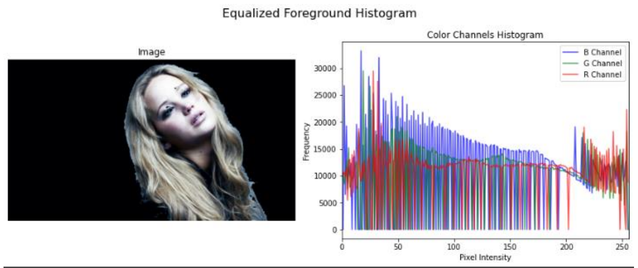
f). part this code line for combining foreground with background

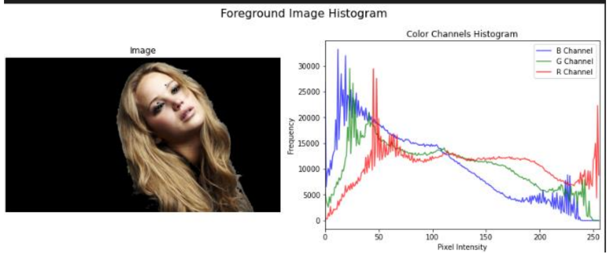


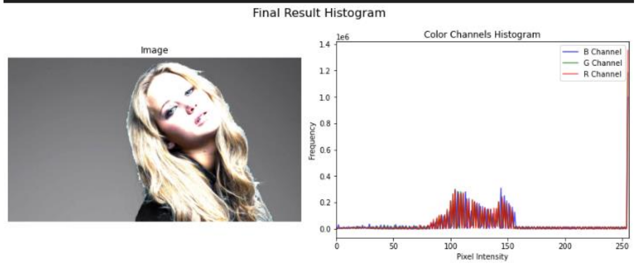
```
background = cv2.bitwise_and(img, img, mask=cv2.bitwise_not(foreground_mask.astype(np.uint8)))
result = cv2.add(equalized_foreground, background)
```











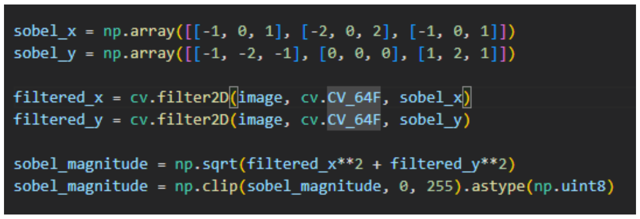
histogram equalization to the RGB channels independently. This approach led to color distortions, as it did not take into account the interrelationships between the channels. The resulting image appeared grayish, indicating that the histogram equalization process faced issues.

Next, I included the original black background in the histogram calculation, which skewed the results and negatively affected the overall outcome. As a result, the image became bright and washed out, suggesting that many pixel values were pushed toward the higher end of the intensity range.

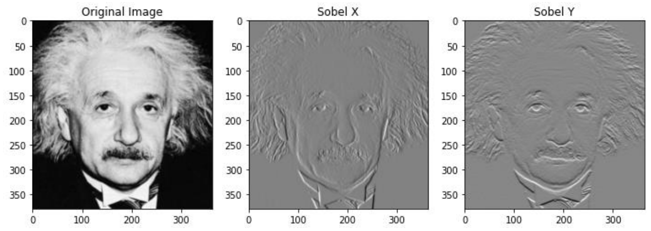
Ultimately, this method caused a loss of color saturation, further compromising the quality of the final image.

7.0. Spatial filtering

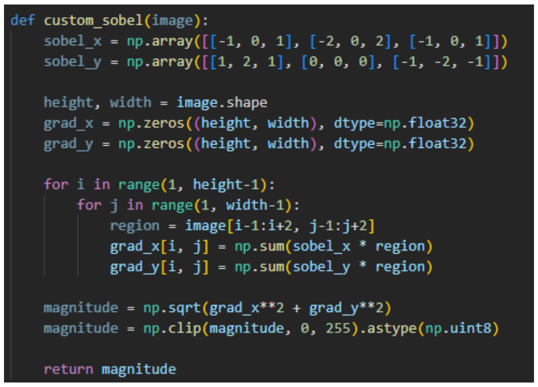
a).uses the built-in `filter2D` function to apply the Sobel filter.



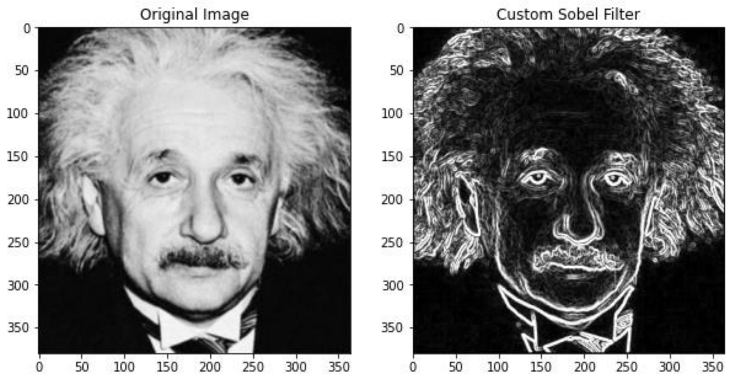
```
sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

filtered_x = cv.filter2D(image, cv.CV_64F, sobel_x)
filtered_y = cv.filter2D(image, cv.CV_64F, sobel_y)

sobel_magnitude = np.sqrt(filtered_x**2 + filtered_y**2)
sobel_magnitude = np.clip(sobel_magnitude, 0, 255).astype(np.uint8)
```



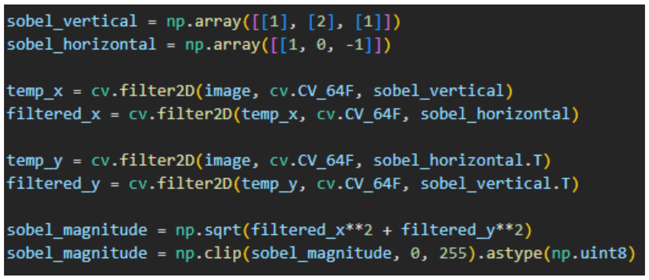
b). shows a custom Sobel filter implementation.



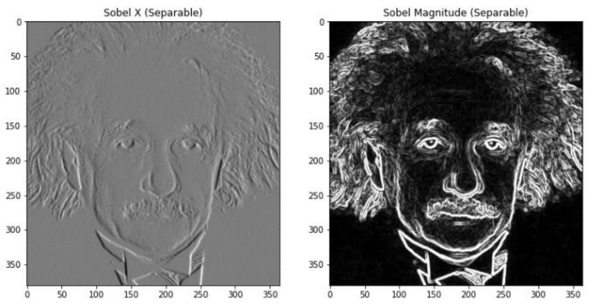
```
def custom_sobel(image):
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

    height, width = image.shape
    grad_x = np.zeros((height, width), dtype=np.float32)
    grad_y = np.zeros((height, width), dtype=np.float32)

    for i in range(1, height-1):
        for j in range(1, width-1):
            region = image[i-1:i+2, j-1:j+2]
            grad_x[i, j] = np.sum(sobel_x * region)
            grad_y[i, j] = np.sum(sobel_y * region)

    magnitude = np.sqrt(grad_x**2 + grad_y**2)
    magnitude = np.clip(magnitude, 0, 255).astype(np.uint8)

    return magnitude
```



c).applies separable Sobel filters to break the filtering into vertical and horizontal components.

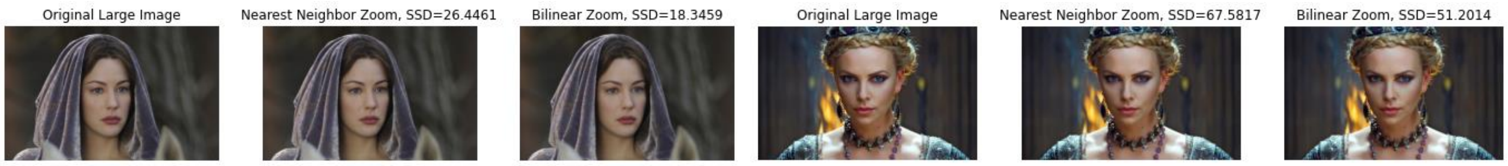


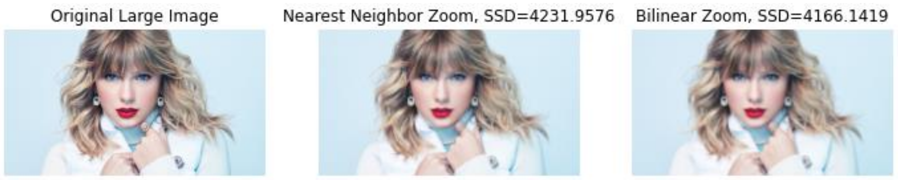
```
sobel_vertical = np.array([[1], [2], [1]])
sobel_horizontal = np.array([[1, 0, -1]])

temp_x = cv.filter2D(image, cv.CV_64F, sobel_vertical)
filtered_x = cv.filter2D(temp_x, cv.CV_64F, sobel_horizontal)

temp_y = cv.filter2D(image, cv.CV_64F, sobel_horizontal.T)
filtered_y = cv.filter2D(temp_y, cv.CV_64F, sobel_vertical.T)

sobel_magnitude = np.sqrt(filtered_x**2 + filtered_y**2)
sobel_magnitude = np.clip(sobel_magnitude, 0, 255).astype(np.uint8)
```

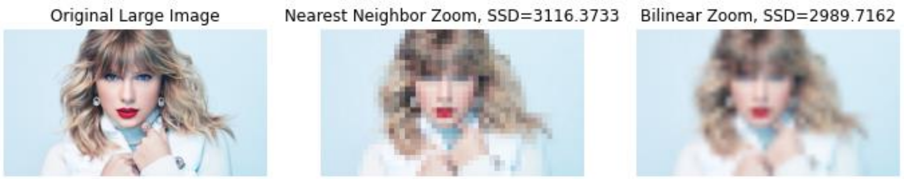


8.0. Zoom function

Original Large Image
Nearest Neighbor Zoom, SSD=136.2691
Bilinear Zoom, SSD=115.0919

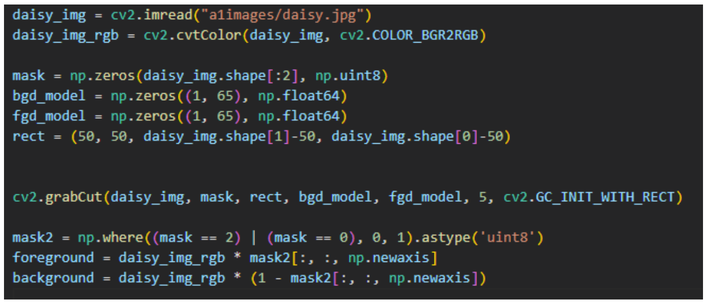

Original Large Image
Nearest Neighbor Zoom, SSD=26.4461
Bilinear Zoom, SSD=18.3459
Original Large Image
Nearest Neighbor Zoom, SSD=67.5817
Bilinear Zoom, SSD=51.2014


Original Large Image
Nearest Neighbor Zoom, SSD=4231.9576
Bilinear Zoom, SSD=4166.1419

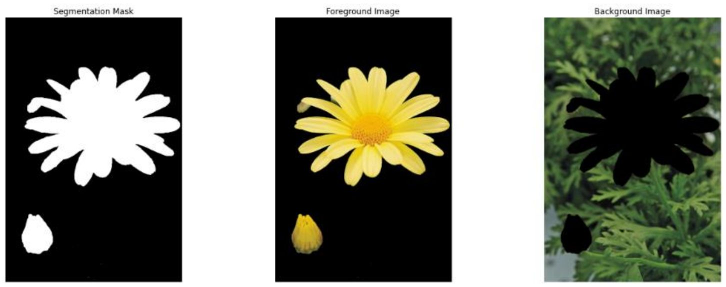

Original Large Image
Nearest Neighbor Zoom, SSD=3116.3733
Bilinear Zoom, SSD=2989.7162

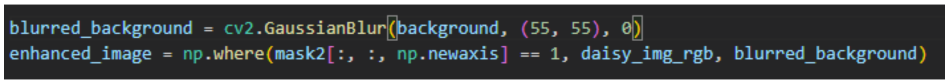
In the first case, when zooming a small image, the SSD values are higher (4231.96 for nearest-neighbor and 4166.14 for bilinear), indicating significant differences from the original large image. The nearest-neighbor method produces more pixelation, while bilinear interpolation smooths the zoomed image. In the second case, with a very small image, the SSD values are lower (3116.37 for nearest-neighbor and 2989.72 for bilinear), reflecting less variation due to the lower detail in the very small image. Overall, bilinear interpolation provides smoother results, but both methods struggle to restore lost detail.

9.0. Image Segmentation and Enhancement Using grabCut Algorithm

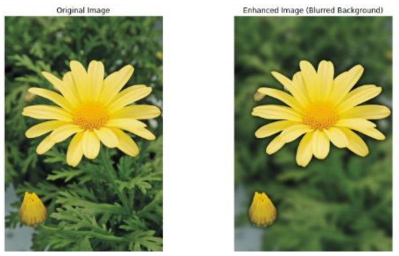


```python
daisy_img = cv2.imread("a1images/daisy.jpg")
daisy_img_rgb = cv2.cvtColor(daisy_img, cv2.COLOR_BGR2RGB)

mask = np.zeros(daisy_img.shape[:2], np.uint8)
bgd_model = np.zeros((1, 65), np.float64)
fgd_model = np.zeros((1, 65), np.float64)
rect = (50, 50, daisy_img.shape[1]-50, daisy_img.shape[0]-50)


cv2.grabCut(daisy_img, mask, rect, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)

mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
foreground = daisy_img_rgb * mask2[:, :, np.newaxis]
background = daisy_img_rgb * (1 - mask2[:, :, np.newaxis])
```





```python
blurred_background = cv2.GaussianBlur(background, (55, 55), 0)
enhanced_image = np.where(mask2[:, :, np.newaxis] == 1, daisy_img_rgb, blurred_background)
```



When the background is blurred using a Gaussian blur, the kernel (filter) considers nearby pixels to compute the new value for each pixel. If there are dark pixels in close proximity to the edge of the flower, the blur effect will incorporate these dark pixels into the neighboring areas, causing the area just beyond the flower to appear darker. This is a side effect of how Gaussian blur works—it smooths the transition by averaging pixel values, so if some dark pixels are near the bright edge of the flower, the blur causes a darker halo-like effect near the boundary.