Assignment 1 - Intensity Transformations and Neighborhood Filtering

EN3160: Image Processing And Machine Vision

Index No. - 210549D

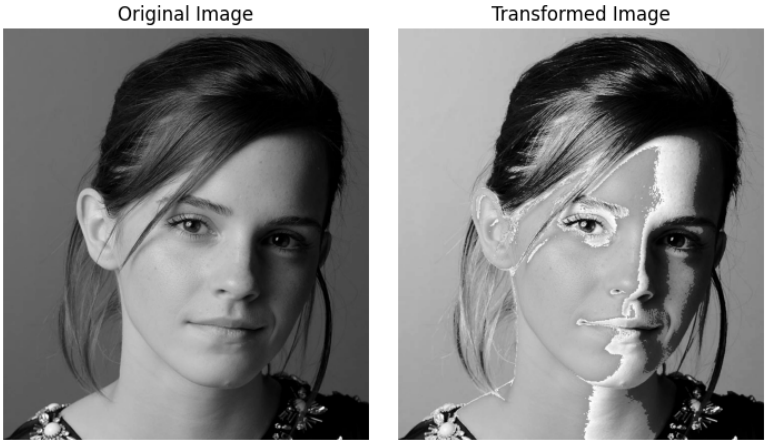
Name - N.P.S.S. Rupasinghe

GitHub Link for Code

Important parts of the code are included in this document, for more details visit GitHub

Question 1: Intensity Transformation

```python
def intensity_transformation(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    lookup_table = np.zeros(256, dtype=np.float64)
    for i in range(0, 51):
        lookup_table[i] = i  # Maps 0-50
    m = (255-100)/100
    c = 255 - (m * 150)
    for i in range(51, 151):
        lookup_table[i] = m * i + c  # Maps 50-150
    for i in range(151, 256):
        lookup_table[i] = i  # Maps 150-255
    # Apply the transformation using the lookup table
    transformed_image = cv2.LUT(image, lookup_table)
```



Question 2: Accentuation of White and Gray Matter in Brain Image

```python
def transform_gray_matter(input_intensity):#Function to transform intensity for gray matter
    if 186 <= input_intensity <= 255:
        value = 1.75 * input_intensity + 30
        return min(value, 255)  # Cap at 255
    else: return input_intensity
def transform_white_matter(input_intensity):#Function to transform intensity for white matter
    if 150 <= input_intensity <= 185:
        value = 1.55 * input_intensity + 22.5
        return min(value, 255)  # Cap at 255
    else: return input_intensity
```

The chosen intensity ranges for gray matter (186 to 255) and white matter (150 to 185) are based on typical pixel values observed in medical imaging. These ranges effectively capture the unique intensity characteristics of each type, ensuring optimal enhancement for improved visibility and differentiation.

Question 3: Gamma Correction

```python
def gamma_correction(image_path, gamma_value):
    img = cv2.imread(image_path, cv2.IMREAD_COLOR)
    assert img is not None, "Image could not be read, check the file path."
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_lab = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2Lab)
    L, a, b = cv2.split(img_lab) # Split the Lab image into L*, a*, and b* channels
    t = np.array([(p / 255) ** gamma_value * 255 for p in range(256)]).astype(np.uint8)
    L_gamma = cv2.LUT(L, t) # Apply the gamma correction using the lookup table
    img_gamma_lab = cv2.merge([L_gamma, a, b])
    img_gamma_rgb = cv2.cvtColor(img_gamma_lab, cv2.COLOR_Lab2RGB)
    # Display the original and gamma-corrected images using the helper function
    display_image('Original Image', img_rgb, cmap=None)
    display_image(f'Gamma Corrected Image (Gamma = {gamma_value})', img_gamma_rgb, cmap=None)
```

Figure 1: Original, gamma=0.8, gamma=1.4

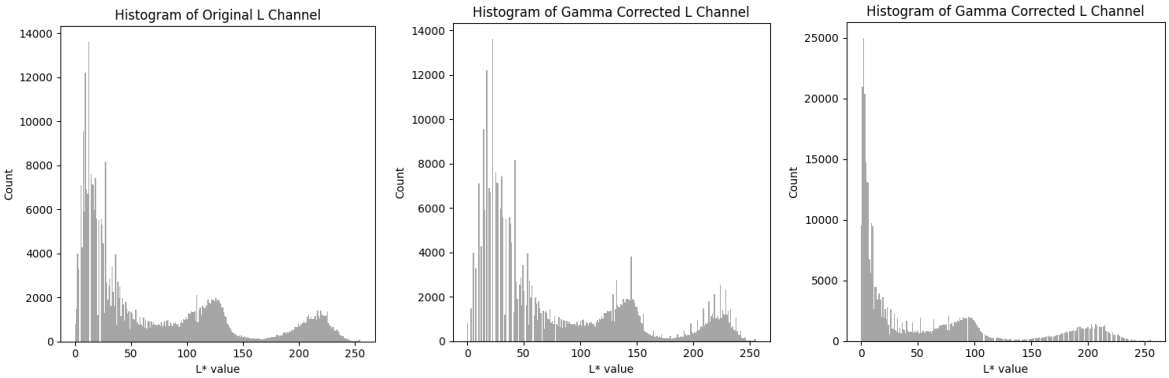
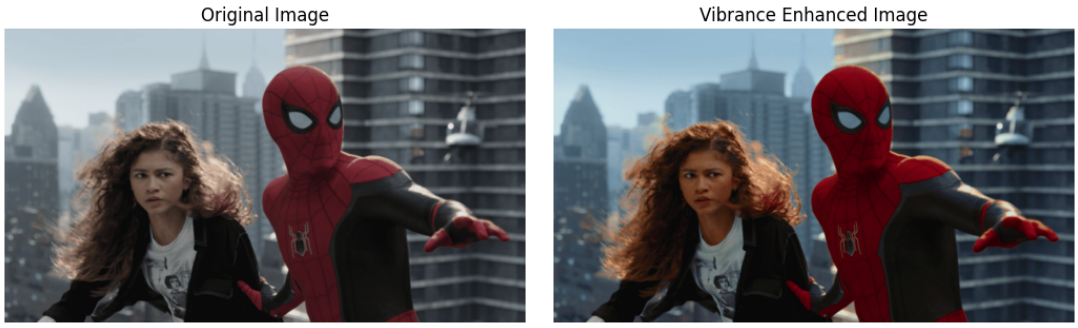


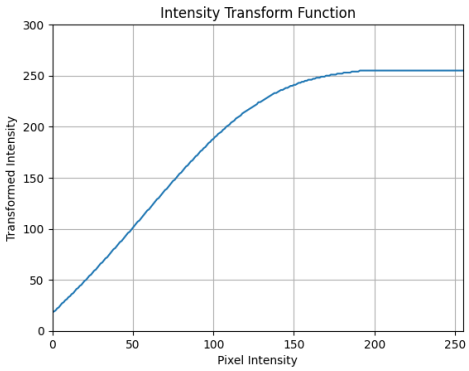
Figure 2: L channel histogram of original and gamma corrected 0.8 and 1.4 images

Question 4: Vibrance Enhancement

```python
def vibrance_enhancement(image_path, a=0.7, sigma=70):
    img = cv2.imread(image_path)
    splited_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # Convert the image to HSV
    H, S, V = cv2.split(splited_img)
    # Intensity transform function
    t = np.array([p + a * 128 * np.exp(-(p - 128) ** 2 / (2 * sigma ** 2)) for p in
    ↪  range(256)])
    # Clip values to make sure they are in the range [0, 255]
    t_clipped = np.clip(t, 0, 255).astype(np.uint8)
    # Apply the intensity transform function to the saturation channel
    s_transformed = cv2.LUT(S, t_clipped)
    result = cv2.merge([H, s_transformed, V]) #Merge the H, transformed S, and V
    result = cv2.cvtColor(result, cv2.COLOR_HSV2RGB)
```

Chosen a value a = 0.75 and sigma = 70





Question 5: Histogram Equalization

```
def calculate_histogram(image):
    histogram = np.zeros(256, dtype=int)
    for pixel in image.flatten():
        histogram[pixel] += 1
    return histogram
def histogram_equalization(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    equalized_image = cv2.equalizeHist(image)
    original_hist = calculate_histogram(image)
    equalized_hist = calculate_histogram(equalized_image)
    display_image('Original Image', image)
    display_image('Equalized Image', equalized_image)
```

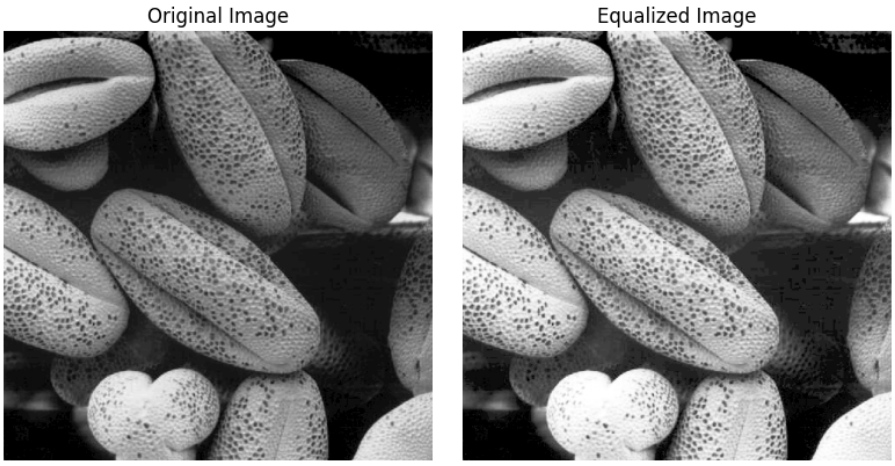


Figure 3: Original (left) and Mean Filtered (right) images.

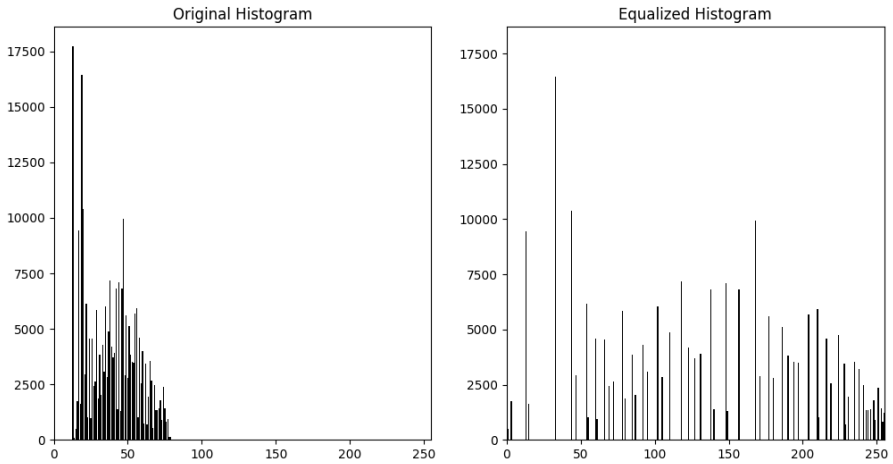


Figure 4: Original and Equalized histograms

Question 6: Histogram Equalization Foreground

```
def histogram_equalization_foreground(image_path):
    image = cv2.imread(image_path)
    image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

```
# Split the HSV image into hue, saturation, and value planes
hue, saturation, value = cv2.split(image_hsv)
# Thresholding to create a mask based on the value component
_, mask = cv2.threshold(value, 75, 255, cv2.THRESH_BINARY)
# Apply bitwise_and to isolate the foreground in the value plane
foreground_value = cv2.bitwise_and(value, value, mask=mask)
# Calculate and display the histogram of the foreground value plane
foreground_hist = calculate_histogram(foreground_value)
# Calculate the cumulative sum of the histogram (CDF)
cdf = np.cumsum(foreground_hist)
# Normalize the CDF to map the value plane
cdf_normalized = (cdf * 255 / cdf.max()).astype(np.uint8)
# Map the value plane using the normalized CDF
equalized_value = cdf_normalized[foreground_value]
# Combine the equalized foreground with the original background
background_value = cv2.bitwise_and(value, value, mask=cv2.bitwise_not(mask))
combined_value = cv2.bitwise_or(equalized_value, background_value)
# Combine with original hue and saturation
final_image_hsv = cv2.merge([hue, saturation, combined_value])
# Convert the final HSV image back to BGR for display
final_image = cv2.cvtColor(final_image_hsv, cv2.COLOR_HSV2BGR)
```

Figure 5: Hue Saturation and Value plots

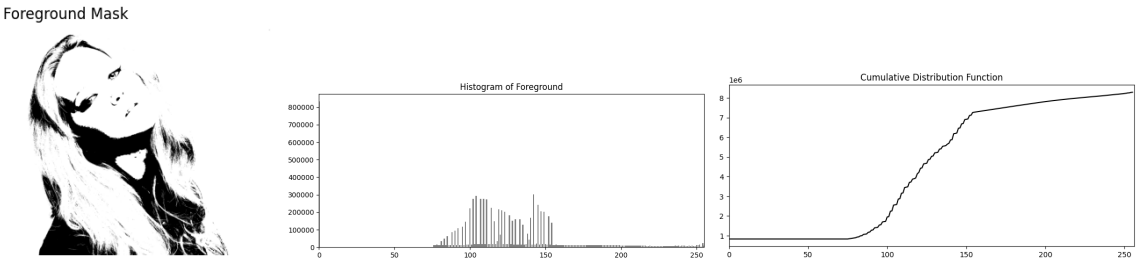


Figure 6: Foreground Mask, Histogram of Foreground, Cumulative Distribution Function

Question 7: Sobel Filtereing

```python
def sobel_filtering_cv2(image_path): # (a) Using cv2.filter2D to Sobel filter the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    sobel_x = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]], dtype="int")
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], dtype="int")
    sobel_filtered_x = cv2.filter2D(image, -1, sobel_x)
    sobel_filtered_y = cv2.filter2D(image, -1, sobel_y)
    sobel_filtered = np.sqrt(sobel_filtered_x.astype(np.float32)**2 + sobel_filtered_y.astype(np.float32)**2)
    sobel_filtered = np.uint8(sobel_filtered)
    display_sobel_results(image, sobel_filtered_x, sobel_filtered_y, sobel_filtered)

# (b) Write your own code to Sobel filter the image
def sobel_filtering_manual(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    height, width = image.shape
    sobel_x = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]], dtype="int")
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], dtype="int")
    sobel_filtered_x = np.zeros_like(image, dtype=np.float32)
    sobel_filtered_y = np.zeros_like(image, dtype=np.float32)
    for i in range(1, height - 1):
        for j in range(1, width - 1):
            # Apply Sobel X kernel
            sobel_filtered_x[i, j] = (sobel_x[0, 0] * image[i - 1, j - 1] + sobel_x[0, 1] * image[i - 1, j] +
                                      sobel_x[0, 2] * image[i - 1, j + 1] + sobel_x[1, 0] * image[i, j - 1] +
                                      sobel_x[1, 1] * image[i, j] +
                                      sobel_x[1, 2] * image[i, j + 1] + sobel_x[2, 0] * image[i + 1, j - 1] +
                                      sobel_x[2, 1] * image[i + 1, j] +
                                      sobel_x[2, 2] * image[i + 1, j + 1])

            # Apply Sobel Y kernel
            sobel_filtered_y[i, j] = (sobel_y[0, 0] * image[i - 1, j - 1] + sobel_y[0, 1] * image[i - 1, j] +
                                      sobel_y[0, 2] * image[i - 1, j + 1] + sobel_y[1, 0] * image[i, j - 1] +
                                      sobel_y[1, 1] * image[i, j] +
                                      sobel_y[1, 2] * image[i, j + 1] + sobel_y[2, 0] * image[i + 1, j - 1] +
                                      sobel_y[2, 1] * image[i + 1, j] +
                                      sobel_y[2, 2] * image[i + 1, j + 1])

    sobel_filtered = np.sqrt(sobel_filtered_x**2 + sobel_filtered_y**2)
    sobel_filtered = np.uint8(sobel_filtered)
    display_sobel_results(image, sobel_filtered_x, sobel_filtered_y, sobel_filtered)

# (c) Using the separable property of the Sobel filter (two 1D convolutions)
def sobel_filtering_separable(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    sobel_x = np.array([[1], [2], [1]], dtype="int") @ np.array([[1, 0, -1]], dtype="int")
    sobel_y = np.array([[1], [2], [1]], dtype="int") @ np.array([[1, 0, -1]], dtype="int")
    sobel_x_filtered = cv2.filter2D(image, -1, sobel_x)
    sobel_y_filtered = cv2.filter2D(image, -1, sobel_y)
    sobel_filtered = np.sqrt(sobel_x_filtered.astype(np.float32)**2 + sobel_y_filtered.astype(np.float32)**2)
    sobel_filtered = np.uint8(sobel_filtered)
    display_sobel_results(image, sobel_x_filtered, sobel_y_filtered, sobel_filtered)
```

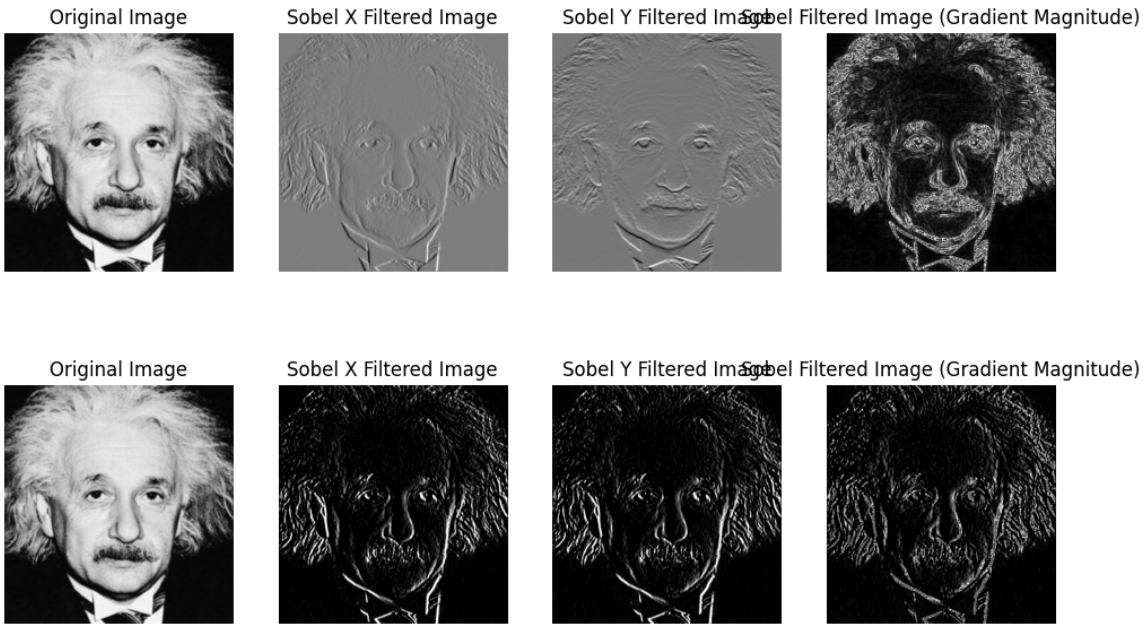


Figure 7: original, Sobel x filtered, Sobel y filtered, Sobel filtered

Question 8: Image Zooming

```python
def zoom_image(image_path, scale, reference_image_path, method='bilinear', bypass_size_error=True):
    image = cv2.imread(image_path)
    reference_image = cv2.imread(reference_image_path)
    # Resize the image based on the chosen method
    if method == 'bilinear':
        zoomed_image = cv2.resize(image, None, fx=scale, fy=scale, interpolation=cv2.INTER_LINEAR)
    elif method == 'nearest':
        zoomed_image = cv2.resize(image, None, fx=scale, fy=scale, interpolation=cv2.INTER_NEAREST)
    else:
        print("Invalid interpolation method. Choose 'bilinear' or 'nearest'.")
        return
    # Resize the reference image to match the zoomed image size
    resized_reference_image = cv2.resize(reference_image, (zoomed_image.shape[1], zoomed_image.shape[0]))
    # Calculate normalized SSD with the resized reference image
    ssd_value = compute_normalized_ssd(zoomed_image, resized_reference_image, bypass_size_error=bypass_size_error)
    # Display the original and zoomed images
    display_image('Original Image', image)
    display_image(f'Zoomed Image (Method: {method}, SSD: {ssd_value:.2f})', zoomed_image)
```

Figure 8: Original - im02, bilinear(SSD: 14.93), nearest(SSD: 19.06)

Figure 9: Original - taylor_very_small, bilinear(SSD: 371.94), nearest(SSD: 450.57)

Question 9: Image Segmentation with GrabCut

```
def grabcut_segmentation(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    mask = np.zeros(image.shape[:2], np.uint8)
    rect = (50, 50, image.shape[1] - 50, image.shape[0] - 50)# Define initial bounding box for segmentation
    bgd_model = np.zeros((1, 65), np.float64)
    fgd_model = np.zeros((1, 65), np.float64)
    cv2.grabCut(image, mask, rect, bgd_model, fgd_model, 5,  cv2.GC_INIT_WITH_RECT) # Apply GrabCut
    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8') # Extract foreground and background
    foreground = image * mask2[:, :, np.newaxis]
    background = image * (1 - mask2[:, :, np.newaxis])
    blurred_background = cv2.GaussianBlur(background, (55, 55), 0) # Produce a blurred background
    enhanced_image = np.where(mask2[:, :, np.newaxis] == 1, foreground, blurred_background) # Combine the sharp fo
```

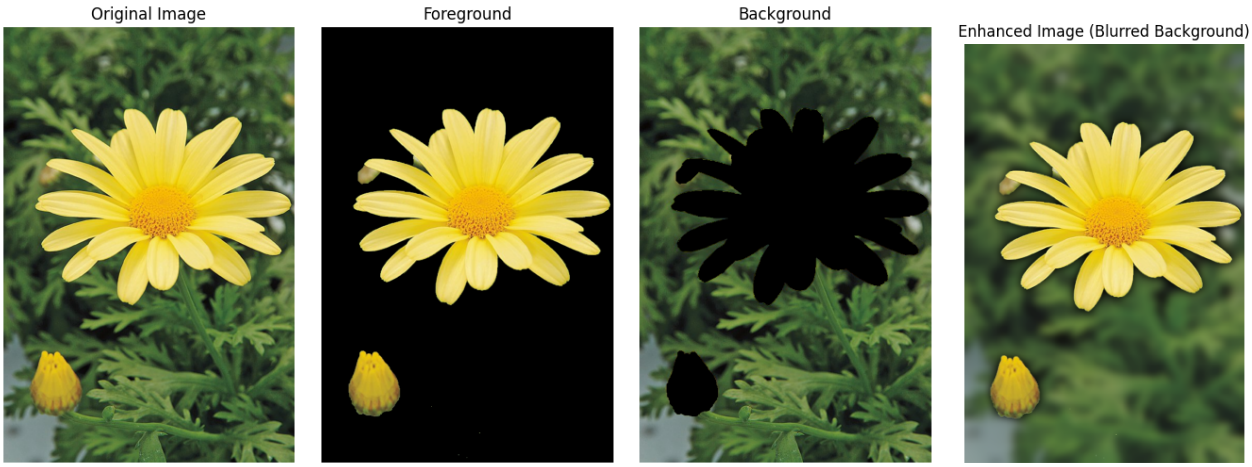


Figure 10: Original, Foreground, Background, Enchanced

The darkness around the flower's edges in the enlarged image is mainly due to the blurring technique used during enhancement. This effect causes darker shadows at the margins to blend into the lighter background, softening the transition and amplifying contrast, making the boundary appear darker than in the original image.