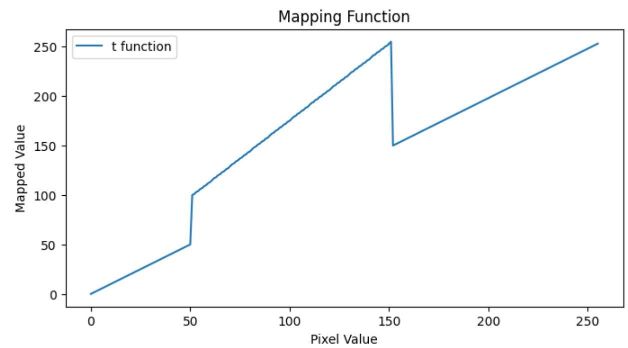
Index no: 200022X

EN3160 Assignment 1

Intensity Transformations and Neighborhood Filtering

1. Implement the intensity transformation

```
image = cv.imread('emma.jpg', cv.IMREAD_GRAYSCALE)

t1 = np.linspace(0, 50, 51).astype('uint8')
t2 = np.linspace(100, 255, 101).astype('uint8')
t3 = np.linspace(150, 255, 106).astype('uint8')

transform = np.concatenate((t1, t2), axis=0)
transform = np.concatenate((transform, t3), axis=0)

transform = transform[:256].astype(np.uint8)

g = cv.LUT(image, transform)
```

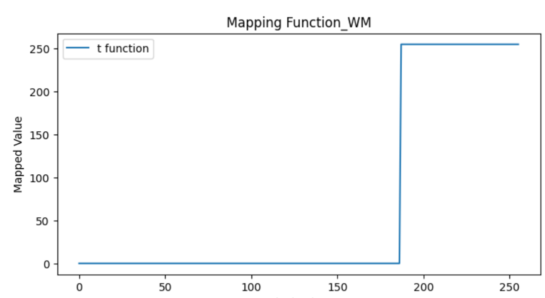


The pixel values between 50 and 150 grayscales in the original image are mapped to higher pixel values. Therefore, those pixel values have become lighter in the processed image compared to the original image. Other pixel values remain the same in both images.

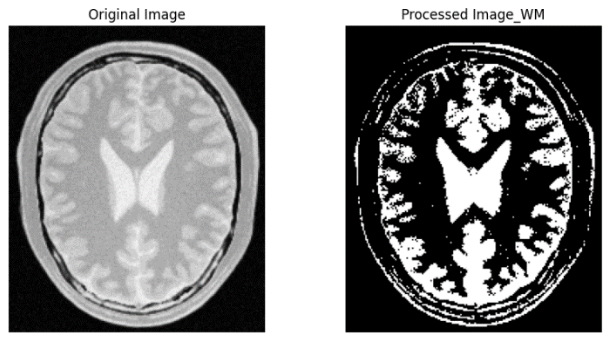
2. Accentuate white matter & gray matter in the brain proton density image

White matter

```
image = cv.imread('BrainProtonDensitySlice9.png', cv.IMREAD_GRAYSCALE)

t1_WM = np.linspace(0, 0, 187).astype('uint8')
t2_WM = np.linspace(255,255, 70).astype('uint8')

transform_WM = np.concatenate((t1_WM, t2_WM), axis=0)
transform_WM = transform_WM[:256].astype(np.uint8)

g_WM = cv.LUT(image, transform_WM)
```

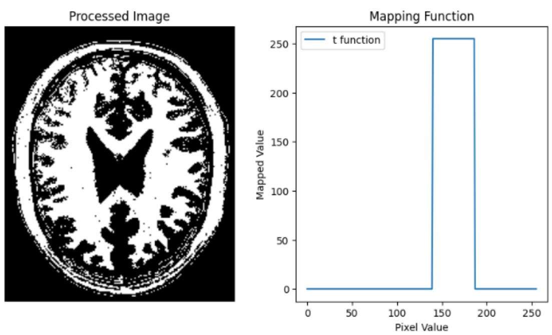


white matter is attenuated by mapping near white pixel values to 255(white). Here 187 to 255 pixel values are identified as near white and mapped to 255 while other pixel values are mapped to black(0).



Gray matter

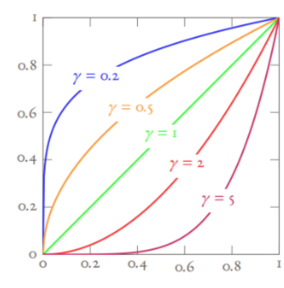
Similarly, gray matter is attenuated by mapping 140 to 187 pixels to white(255).

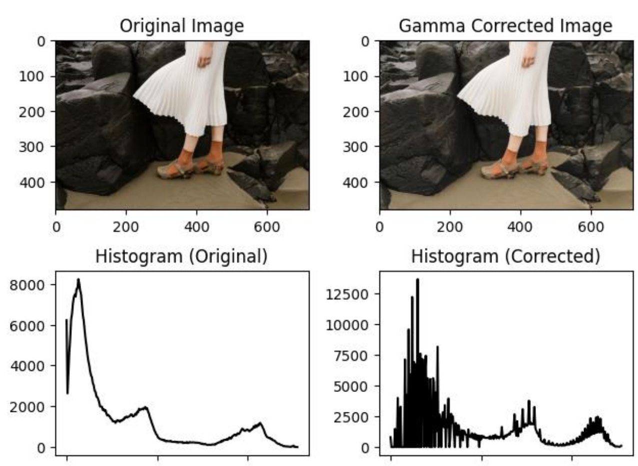



```
t1_GM = np.linspace(0, 0, 140).astype('uint8')
t2_GM = np.linspace(255, 255, 47).astype('uint8')
t3_GM = np.linspace(0, 0, 70).astype('uint8')

transform_GM = np.concatenate((t1_GM, t2_GM), axis=0)
transform_GM = np.concatenate((transform_GM, t3_GM), axis=0)
transform_GM = transform_GM[:256].astype(np.uint8)

g_GM = cv.LUT(image, transform_GM)
```

3. Apply Gamma correction to L plane

```
# Convert the image to the Lab color space
img_lab = cv.cvtColor(img_orig, cv.COLOR_BGR2Lab)

# Extract the L channel
L_channel = img_lab[:, :, 0]

# Apply gamma correction to the L channel
gamma = 0.7
table = np.array([(i / 255.0) ** gamma * 255.0 for i in range(256)]).astype('uint8')
L_corrected = cv.LUT(L_channel, table)

# Update the L channel in the Lab image
img_lab[:, :, 0] = L_corrected
```





Since the gamma value(0.7) is less than 1, map a narrow range of dark pixels to a wider range of dark pixels. Therefore, the dark regions of the image have become slightly brighter and increased the visibility of details in the dark region of the image.

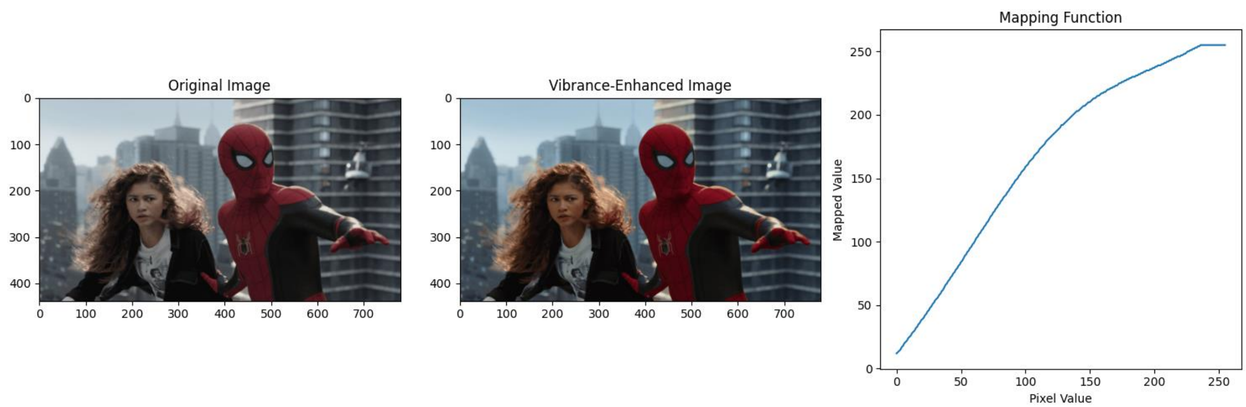
```
# Calculate histograms for original and corrected images
hist_orig = cv.calcHist([img_orig], [0], None, [256], [0, 256])
hist_corrected = cv.calcHist([img_lab], [0], None, [256], [0, 256])
```

According to histogram, we can see low-intensity pixel values have moved to the right.

Gamma value = 0.

4. Increasing the vibrance of a photograph

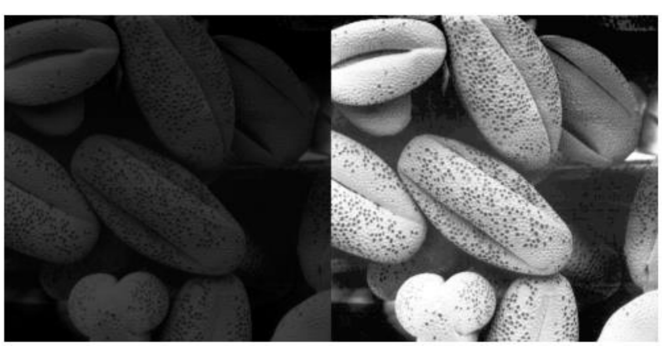
```python
# Load the original image
img_orig = cv.imread('spider.png', cv.IMREAD_COLOR)

# Convert the image to the HSV color space
img_hsv = cv.cvtColor(img_orig, cv.COLOR_BGR2HSV)

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

# Apply the intensity transformation
a = 0.5
sigma = 70
table = np.array([min(i + (a * 128) * math.exp(-(i - 128) ** 2 / (2 * sigma ** 2)), 255) for i in range(256)]).astype('uint8')
S_transformed = cv.LUT(S, table)

# Update the S plane in the HSV image
img_hsv[:, :, 1] = S_transformed
```

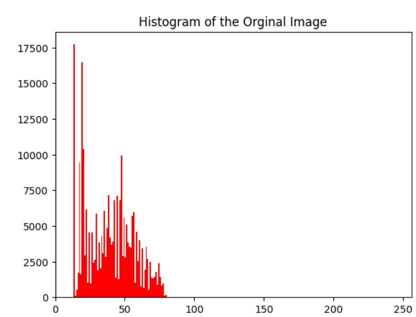


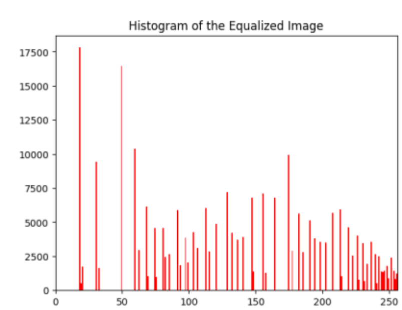
The original image looks dull and, after applying vibrance enhancement, the image has become more lively. According to the mapping function, it amplifies the intensities of colour in the mid-range.

a is selected as 0.5

5. histogram equalization







steps

- Calculating CDF of histogram
- Mapping CDF value to (0,255) range
- Apply CDF as the transformation function.

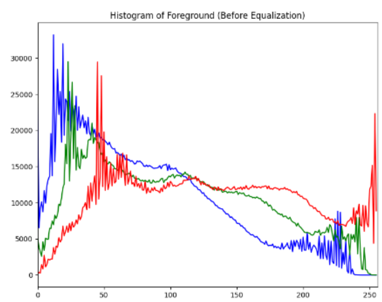
According to the histogram, In the equalized image, pixel values are spread out over (0-255) range. Therefore equalized image is brighter and more details of the image are visible.

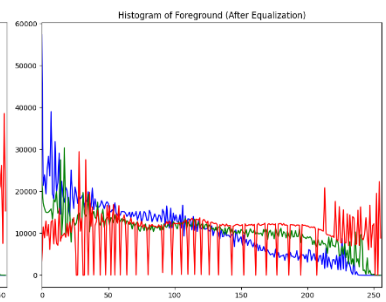
```
def equalizeHistogram(img):

    img_height = img.shape[0]
    img_width = img.shape[1]
    histogram = np.zeros([256], np.int32)

    # calculate histogram
    for i in range(0, img_height):
        for j in range(0, img_width):
            histogram[img[i, j]] +=1

    pdf_img = histogram / histogram.sum()   # calculate pdf of the image

    cdf = np.zeros([256], float)

    # For loop for cdf
    for i in range(0, 256):
        for j in range(0, i+1):
            cdf[i] += pdf_img[j]

    cdf = np.zeros(256, float)
    cdf[0] = pdf_img[0]
    for i in range(1, 256):
        cdf[i] = cdf[i-1] + pdf_img[i]

    cdf_eq = np.round(cdf * 255, 0) # mapping, transformation function T(x)

    imgEqualized = np.zeros((img_height, img_width))

    # for mapping input image to s.
    for i in range(0, img_height):
        for j in range(0, img_width):
            r = img[i, j]
            s = cdf_eq[r]
            imgEqualized[i, j] = s
```

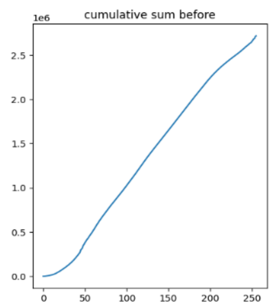
6. **apply histogram equalization only to the foreground**

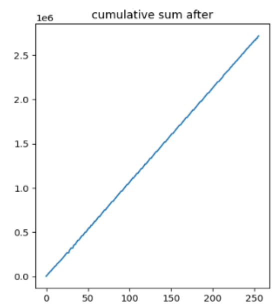
```
# Convert the image to HSV color space
myimage_hsv = cv2.cvtColor(img_orig, cv2.COLOR_BGR2HSV)

# Split the HSV image into hue, saturation, and value planes
hue_plane, saturation_plane, value_plane = cv2.split(myimage_hsv)

# Threshold the saturation plane to obtain the foreground mask
foreground_mask = cv2.threshold(saturation_plane, 12, 255, cv2.THRESH_BINARY)[1]

# Extract the foreground using the mask
foreground = cv2.bitwise_and(img_orig, img_orig, mask=foreground_mask)


# Define color channels
color = ('b', 'g', 'r')

# Plot histograms for each color channel
plt.figure(figsize=(15, 6))
for i, c in enumerate(color):
    hist_foreground = cv2.calcHist([foreground], [i], foreground_mask, [256], [0, 256])
    plt.subplot(1, 2, 1)
    plt.plot(hist_foreground, color=c)

plt.xlim([0, 256])
plt.title('Histogram of Foreground (Before Equalization)')


hist_cumsum = np.cumsum(hist_foreground)
```

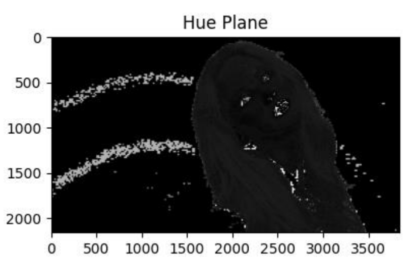
```
# Split the equalized foreground into hue, saturation, and value planes
h_plane, s_plane, v_plane = cv2.split(cv2.cvtColor(foreground, cv2.COLOR_BGR2HSV))

# Apply histogram equalization to the value plane
equ_v_plane = cv2.equalizeHist(v_plane)

# Combine the equalized value plane with the original hue and saturation planes
equ_hsv_image = cv2.merge((h_plane, s_plane, equ_v_plane))
equalized_image = cv2.cvtColor(equ_hsv_image, cv2.COLOR_HSV2BGR)

# Plot histograms for each color channel after equalization
for i, c in enumerate(color):
    hist_foreground_eq = cv2.calcHist([equalized_image], [i], foreground_mask, [256], [0, 256])
    plt.subplot(1, 2, 2)
    plt.plot(hist_foreground_eq, color=c)

plt.xlim([0, 256])
plt.title('Histogram of Foreground (After Equalization)')

plt.tight_layout()
plt.show()

# Placeholder for background_mask
background_mask = np.ones_like(foreground_mask)  # Replace this with the actual background mask

# Combine the equalized foreground and original background
result = cv2.add(cv2.bitwise_and(img_orig, img_orig, mask=background_mask), equalized_image)

# Convert the equalized HSV image back to RGB color space
result_rgb = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)

hist_cumsum_eq = np.cumsum(hist_foreground_eq)
```

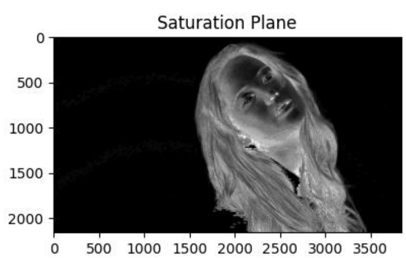


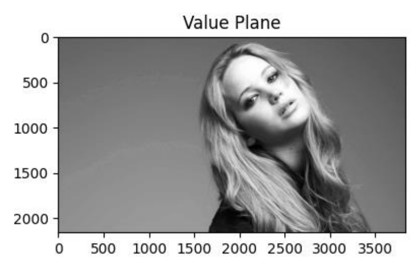


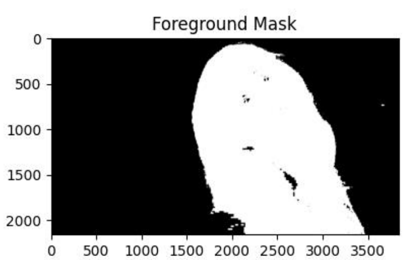


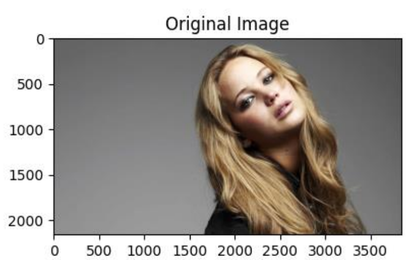


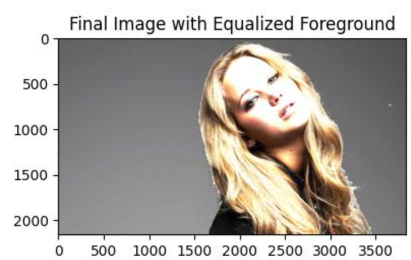








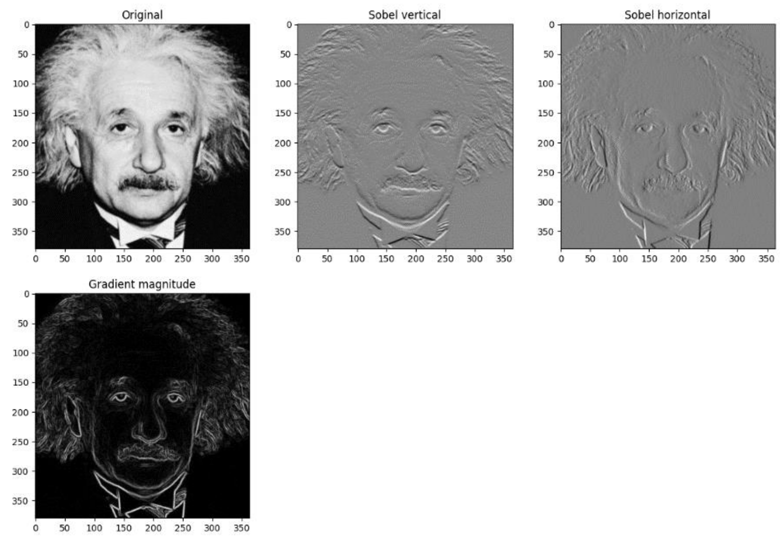




7. Filtering with the Sobel operator can compute the gradient.

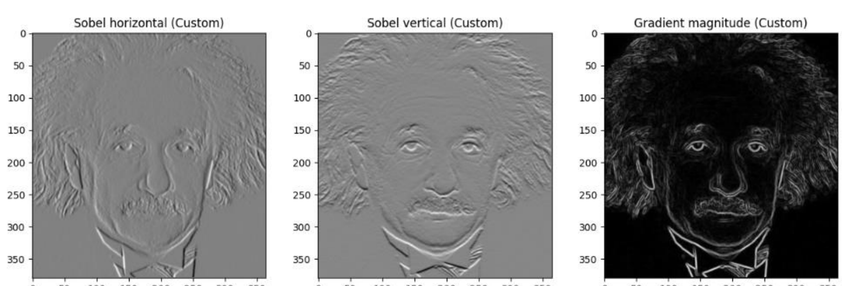
```python
kernel_v = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype='float32')
img_v = cv.filter2D(img, -1, kernel_v)

kernel_h = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype='float32')
img_h = cv.filter2D(img, -1, kernel_h)

img_grd = np.sqrt(img_h ** 2 + img_v ** 2)
```

Using filter2D



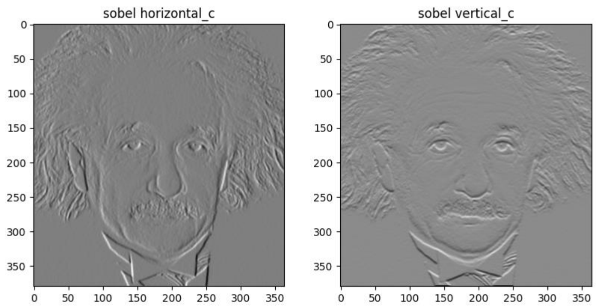

```python
k_h = kernel_h.shape[0] // 2
k_w = kernel_h.shape[1] // 2

img_h2 = np.zeros((img_height, img_width))
img_v2 = np.zeros((img_height, img_width))

for i in range(k_h, img_height - k_h):
    for j in range(k_w, img_width - k_w):
        img_patch = img[i - k_h:i + k_h + 1, j - k_w:j + k_w + 1]
        img_h2[i, j] = np.sum(np.multiply(kernel_h, img_patch))
        img_v2[i, j] = np.sum(np.multiply(kernel_v, img_patch))

img_grd2 = np.sqrt(img_h2 ** 2 + img_v2 ** 2)
```

Own function



Using Convolution

```python
arr_v1 = np.array([[1], [0], [-1]])
arr_v2 = np.array([[1, 2, 1]])

img_v_s = signal.convolve2d(img, arr_v1, mode="same")
img_v3 = signal.convolve2d(img_v_s, arr_v2, mode="same")

arr_h1 = np.array([[1], [2], [1]])
arr_h2 = np.array([[1, 0, -1]])

img_h_s = signal.convolve2d(img, arr_h1, mode="same")
img_h3 = signal.convolve2d(img_h_s, arr_h2, mode="same")
```



8. zooming image

Neareset neighbour

```python
def nearest_neighbor_zoom(img, scale):
    rows = int(scale * img.shape[0])
    columns = int(scale * img.shape[1])
    zoomed_img = np.zeros((rows, columns, 3), dtype=img.dtype)

    for i in range(0, rows - 1):
            for j in range(0, columns - 1):
                x = i / scale
                y = j / scale
                img_x_pos = round(x) if (x <= img.shape[0] - 1) else int(x)
                img_y_pos = round(y) if (y <= img.shape[1] - 1) else int(y)
                zoomed_img[i, j] = img[round(img_x_pos), round(img_y_pos)]
    return zoomed_img
```

Bilinear interpolation

```python
def bilinear_interpolation_zoom(img,scale):
    rows = int(scale * img.shape[0])
    columns = int(scale * img.shape[1])
    zoomed_img = np.zeros((rows, columns, 3), dtype=img.dtype)
    for i in range(0, rows - 1):
            for j in range(0, columns - 1):
                x, y = i / scale, j / scale
                x_left = math.floor(x)
                x_right = math.ceil(x) if (x <= img.shape[0] - 1) else math.floor(x)
                y_top = math.floor(y)
                y_bottom = math.ceil(y) if (y <= img.shape[1] - 1) else math.floor(y)
                r_y = (y - y_top)
                r_x = (x - x_left)
                for channel in range(3):
                    zoomed_img[i, j, channel] = math.floor(
                        (img[x_left, y_top, channel] * (1 - r_y) + img[x_left, y_bottom, channel] * (r_y)) * (1 - r_x)
                        + (img[x_right, y_top, channel] * (1 - r_y) + img[x_right, y_bottom, channel] * (r_y)) * (r_x))

    return zoomed_img
```

Normalized ssd

```python
def SSD(original_img, zoomed_img):
    return np.square(original_img - zoomed_img).sum() / (original_img.shape[0] * original_img.shape[1] * original_img.shape[2])
```

```
orginal image1:  (270, 480, 3)
zoomed nearest2:  (1080, 1920, 3)
zoomed bilinear2:  (1080, 1920, 3)
Normalized SSD for Nearest-Neighbor Interpolation: 40.18347736625515
Normalized SSD for Bilinear Interpolation: 39.32977446630658

orginal image2:  (300, 480, 3)
zoomed nearest2:  (1200, 1920, 3)
zoomed bilinear2:  (1200, 1920, 3)
Normalized SSD for Nearest-Neighbor Interpolation: 16.926062210648148
Normalized SSD for Bilinear Interpolation: 16.345512297453702
```

If closely observed we can see bilinear interpolation is smoother than nearest neighbor. the nearest neighbor method is pixelated because of duplicate pixels. The sum of squared difference value of bilinear interpolation is lesser than nearest neighbor zooming.

9.Apply blurring only to background

```
# Initialize a mask
mask = np.zeros(image.shape[:2], np.uint8)

# Create rectangles to define foreground and background areas
rect = (0, 200, 800, 600)   #x,y,w,h

# Apply GrabCut algorithm
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
cv2.grabCut(image, mask, rect, bgdModel, fgdModel, 8, cv2.GC_INIT_WITH_RECT)

# Modify the mask to create a binary mask
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Apply the mask to the original image
segmented_image = image * mask2[:, :, np.newaxis]

background_image= image - segmented_image

#Apply average blurring to the background
blurred_background = cv.blur(background_image, (9,9))

final_image=segmented_image + blurred_background
```

When blurring(averaging) the background, the intensity of the pixel locations corresponding to the edge of the foreground will be darker, because in the background image foreground region is a dark region. Therefore when we combine foreground and background edges will be darkened.

Github Repository : [Nusrath-Amana/EN3160-Image-Processing-and-Machine-Vision (github.com)]