DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION
UNIVERSITY OF MORATUWA

EN 3160: Image Processing and Machine Vision

This is offered as a "EN 3160: Image Processing and Machine Vision" module's partial completion.

Assignment 01

200686J : Vishagar A.

29^{th} of august, 2023

Abstract

This report deals with the procedure of solving each set of problems provided for our assignment. This includes the theoretical approach for each question and implementation of those approaches for real world images. These questions deal with many important basic image processing techniques like intensity transformation, spatial filtering, gamma correction, histogram equalization and much more.

Contents

1 Question 01 . . . . . **3**

2 Question 02 . . . . . **3**

3 Question 03 . . . . . **4**

4 Question 04 . . . . . **4**

5 Question 05 . . . . . **5**

6 Question 06 . . . . . **5**

7 Question 07 . . . . . **6**

7.1 Using Buit-in Filter2D function 6

7.2 Using Iterations 7

7.3 Using multiplication of two kernels 7

8 Question 08 . . . . . **7**

9 Question 09 . . . . . **8**

10 References **8**

11 Github Repository **8**

1 Question 01

Question one deals with intensity transformation for a image. Our task is to do a intensity transformation for a given image. The intensity transformation function was also given. An intensity transformation is a technique that maps each intensity value of an input image to a corresponding output intensity value through a mathematical expression.

```
# Create Intensity Transformer with the following characteristics
c= np.array([(50,50),(50,100),(150,255),(150,150)])
t1 = np.linspace(0,c[0,1],c[0,0]+1).astype("uint8")
# print(len(t1))
t2 = np.linspace(c[1,1]+1,c[2,1],c[2,0]-c[1,0]).astype("uint8")
# print(len(t2))
t3 = np.linspace(c[3,1]+1,255,255-c[3,0]).astype("uint8")
# print(len(t3))

# Plot the intensity transformer
transform = np.concatenate((t1,t2),axis=0).astype("uint8")
transform = np.concatenate((transform,t3),axis=0).astype("uint8")
# Perform Transformation and visualize usinf opencv
transformed_image = cv.LUT(img,transform)
```

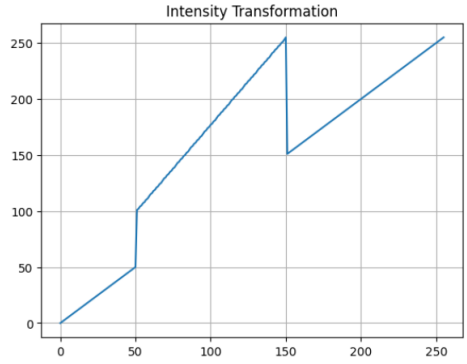

Figure 1: Intensity Transformer

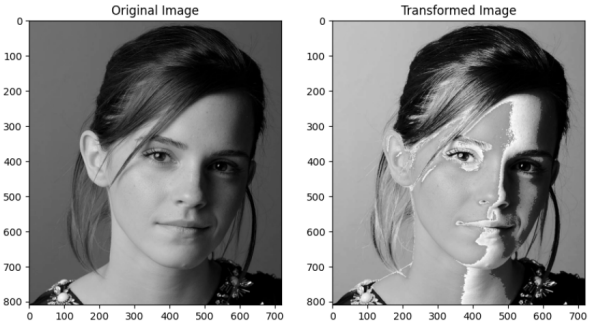

Figure 2: Result after Intensity Transformation

2 Question 02

Question2 deals with intensity transformers as in question one, but here particularly we were asked to accentuate gray and white matters in a brain proton density image. In range of intensities from 50 to 180 gray matter got accentuated and from 180 to 255 white matter got accentuated.

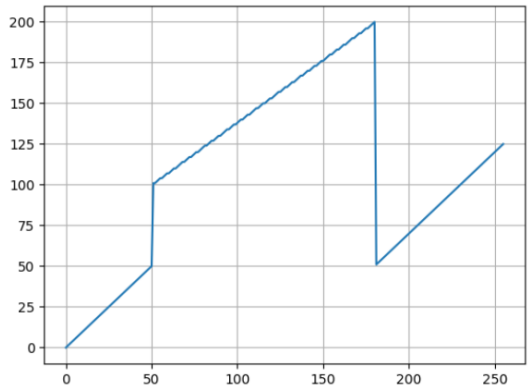

Figure 3: Intensity Transformer

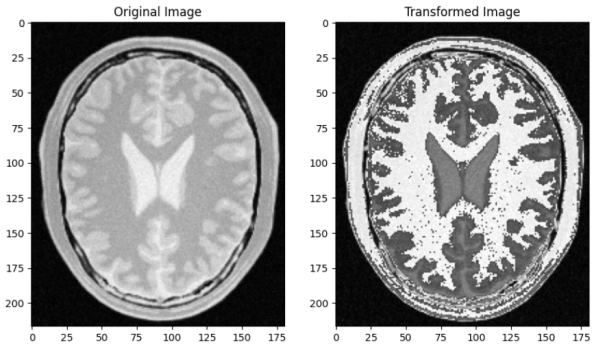

Figure 4: Accentuated gray matter

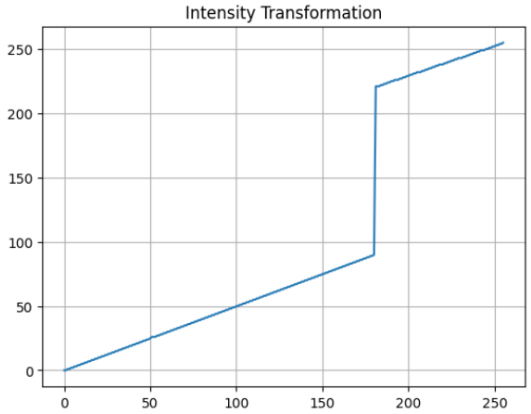

Figure 5: Intensity Transformer

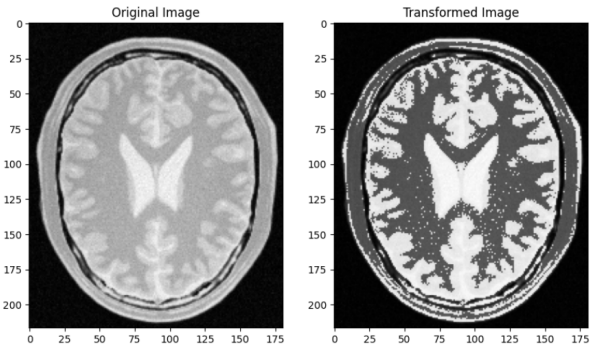

Figure 6: White Matter

3 Question 03

In Question 3 we were asked to perform gamma correction particularly for L plan from (L,a,b) categories. Here we are applying a gamma correction for lightness (L) plane to increase and enhance the brightness of the image.

```
def apply_gamma_correction(image_lab, gamma):

    global L,L_corrected
    # Split the LAB image into L*, a*, and b* channels
    L, a, b = cv.split(image_lab)

    # Apply gamma correction to the L* channel
    L_corrected = np.power(L / 255.0, gamma) * 255.0

    # Ensure the corrected L* channel is in the appropriate data range
    L_corrected = np.clip(L_corrected, 0, 255).astype('uint8')

    # Merge the corrected L* channel with the original a* and b* channels
    corrected_lab = cv.merge([L_corrected, a, b])

    return corrected_lab

# Convert the image to LAB color space
image_lab = cv.cvtColor(img, cv.COLOR_BGR2LAB)
# Set the desired gamma value
gamma = 0.7
# Apply gamma correction to the LAB image
corrected_image_lab = apply_gamma_correction(image_lab, gamma)
# Convert the corrected LAB image back to BGR color space
corrected_image_bgr = cv.cvtColor(corrected_image_lab, cv.COLOR_LAB2BGR)
```

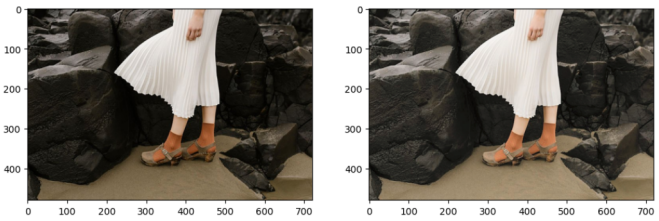

Figure 7: Gamma Corrected Image

4 Question 04

In this question we were asked to do a intensity transformation in a particular image plane (Saturation) extracted from hue,saturation and value planes to enhance tha vibrancy of the images.

```
# Convert the image to HSV color space
image_hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
# Spliting image to HSV planes
hue, saturation, value = cv.split(image_hsv)
# Define the Saturation Transformer
def satIntensityTransformer(x,alpha,zigma):

    transformer = x +(alpha*128)*np.exp((-(x-128)**2)/(2*(zigma**2)))

    return min(transformer,255)

# Define the new array after transformation
transformed_saturation= np.zeros(saturation.shape)
# perform transformation
for i in range(len(saturation)):
    for j in range(len(saturation[i])):
        transformed_saturation[i][j] = satIntensityTransformer(saturation[i][j],0.7,70)
# make sure the values are in the range of 0 to 255
transformed_saturation= transformed_saturation.astype('uint8')
```

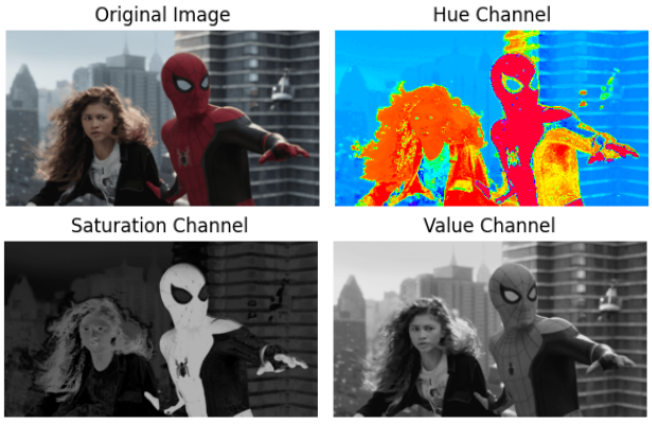

Figure 8: Hue,Saturation and Value Planes

5 Question 05

In this question we were asked to carry out histogram equalization but without using **cv2.equihist()** option. Alternate method in the slides was implemented.

```
#Calculate the histogram of the image
hist,bins = np.histogram(img.ravel()
    ,256,[0,256])
# find probabilities for each intensity
    level
probabilities = hist / np.sum(hist)
# find cumulative sum of pixels
cumsum=np.zeros(256)

for i in range(len(cumsum)):
    cumsum[i] = np.sum(hist[:i])
equalized_cumsum = np.zeros(256)
for x in range(len(equalized_cumsum)):
    equalized_cumsum[x] = (cumsum[x] * 255)
        / img.size

equalized_cumsum = equalized_cumsum.astype(
    'uint8')
equalized_image = np.zeros(img.shape)
for i in range(len(img)):
    for j in range(len(img[i])):
        equalized_image[i][j] =
            equalized_cumsum[img[i][j]]
equalized_image = equalized_image.astype('
    uint8')

array1 = np.arange(256)
array2 = hist
array3 = probabilities
array4 = cumsum
array5 = equalized_cumsum

# Combine arrays into a list of tuples
combined_data = list(zip(array1, array2,
    array3,array4,array5))
# Define headers for the columns
headers = ['r_k', 'n_k', 'Pr(r_k)','
    Cumulative n_k','equalized rounded']
# Print the combined data using tabulate
table = tabulate(combined_data, headers=
    headers, tablefmt='grid')
```

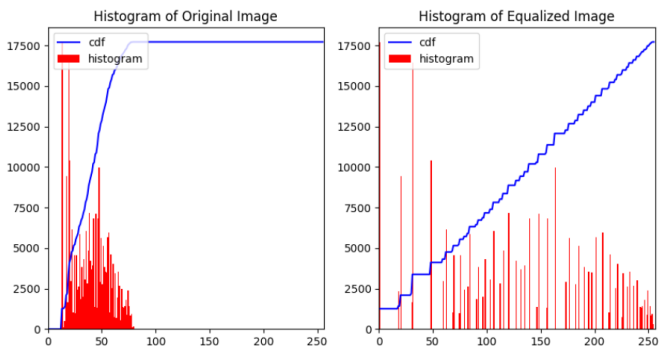

Figure 9: Equalized Histogram

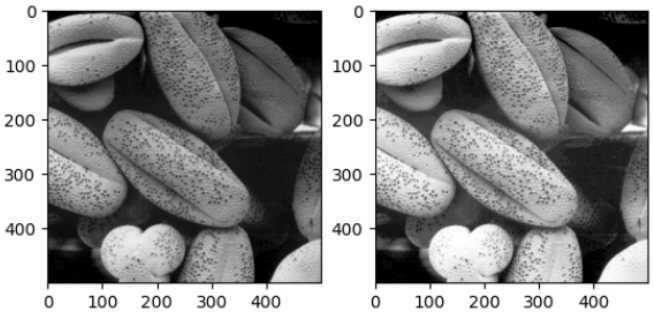

Figure 10: Histogram Corrected Image

6 Question 06

In this question we were asked to split the image in to hue,saturation and value planes , visulaize it and find a mask to extract the foreground and background from the image and do histogram equalization for the foreground and enhance the image.

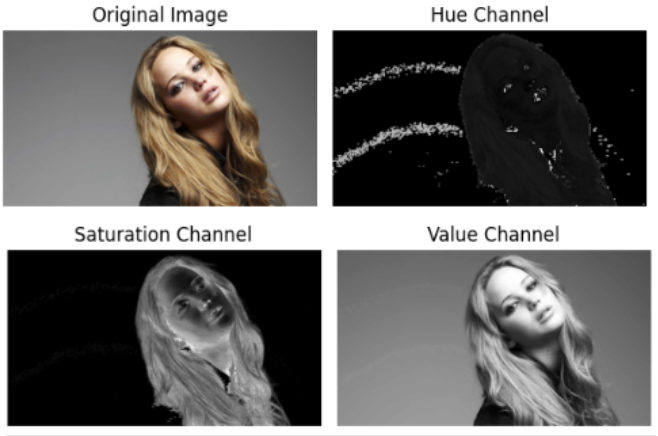

Figure 11: Hue Saturation and Value Planes

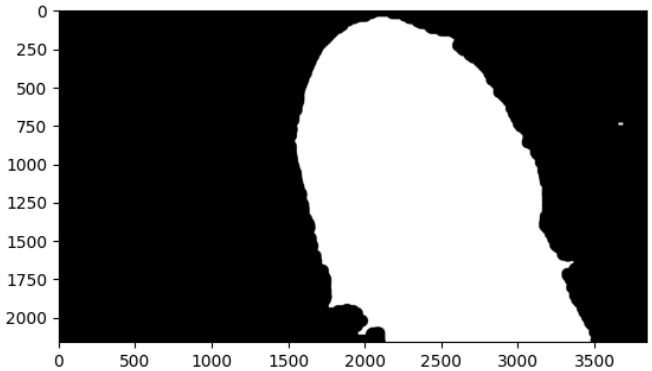

Figure 12: Mask

```
threshold_value = 11
_,foreground_mask=cv.threshold(saturation,
    threshold_value, 255, cv.THRESH_BINARY)
foreground_mask = cv.morphologyEx(
    foreground_mask, cv.MORPH_CLOSE, cv.
    getStructuringElement(cv.MORPH_ELLIPSE,
    (80, 80)))
#masking operation using bitwise_and
foreground = cv.bitwise_and(img, img,mask=
    foreground_mask)
# Convert the foreground to grayscale for
    histogram calculation
foreground_gray = cv.cvtColor(foreground,
    cv.COLOR_BGR2GRAY)
# Extract the background
gray_image = cv.cvtColor(img,cv.
    COLOR_BGR2GRAY)
background_image = gray_image -
    foreground_gray
```

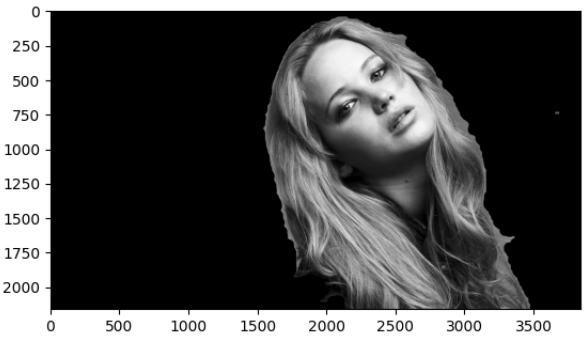

Figure 13: Extracted foreground

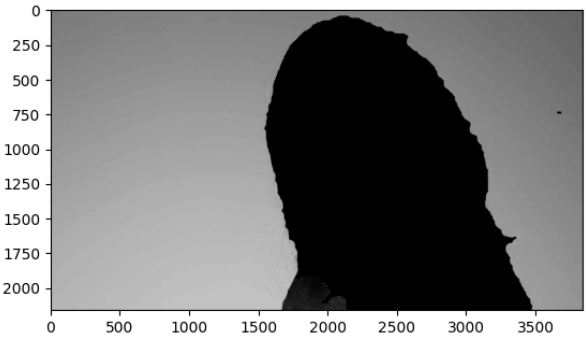

Figure 14: Extracted background

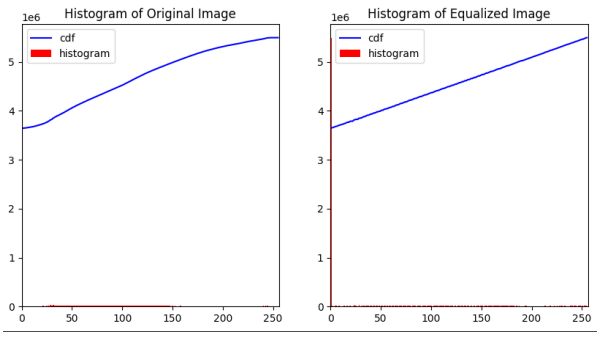

Figure 15: Histogram Correction

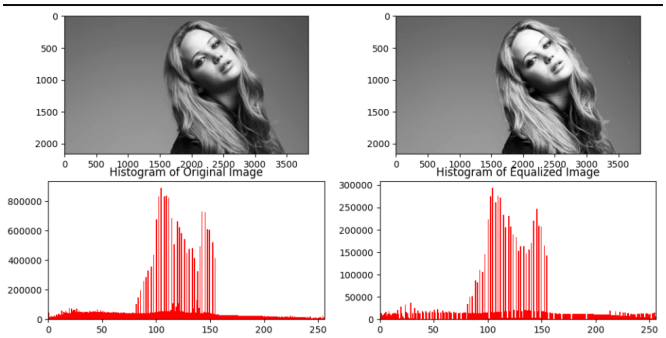

Figure 16: Histogram Corrected Image

7 Question 07

In this section we were asked to perform sobel filtering for a given image and find the gradient magnitude using three different approaches. Normally sobel filter is used to detect the vertical and horizontal edges or gradients of an image. Therefore sobel filters are generally called as “Sobel Horizontal” and “Sobel Vertical”. with their respective kernels.How ever in this question we were asked to use the given kernel therefore I have computed the respective gradient and sobel filtered image for the given kernel.

7.1 Using Buit-in Filter2D function

```
# Creating the kernel(2d convolution matrix
    )
kernel1 = np.array
    ([[1,0,-1],[2,0,-2],[1,0,-1]])
# Applying the filter2D() function
img_filtered = cv.filter2D(img,cv.CV_64F,
    kernel1)
```

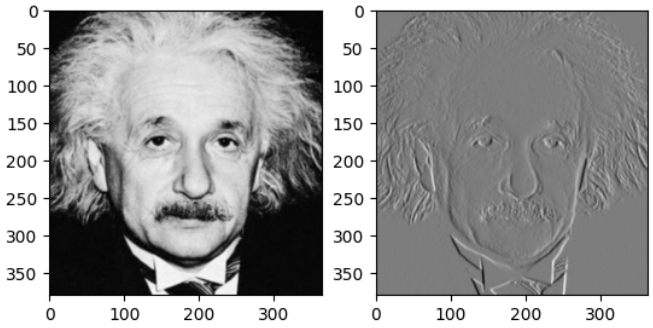

Figure 17: Gradient Magnitude using Filter2D

7.2 Using Iterations

```
# Sobel kernels
sobel_kernel_x = np.array([[-1, 0, 1],
                           [-2, 0, 2],
                           [-1, 0, 1]])
# Convolution operation
gradient_x = np.zeros_like(img, dtype=float)
for y in range(1, img.shape[0] - 1):
    for x in range(1, img.shape[1] - 1):
        window = img[y-1:y+2, x-1:x+2]
        gradient_x[y, x] = np.sum(window * sobel_kernel_x)

# Compute gradient magnitude
gradient_magnitude = np.abs(gradient_x)
# gradient_magnitude = (gradient_magnitude / np.max(gradient_magnitude)) * 255
gradient_magnitude = gradient_magnitude.astype(np.uint8)
```

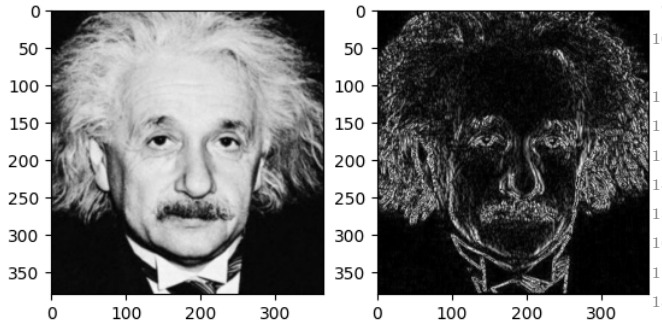

Figure 18: Gradient Magnitude using Iterations

7.3 Using multiplication of two kernels

```
# Defining the kernels
kernel1 = np.array([[1], [2], [1]])
kernel2 = np.array([[1, 0, -1]])

# Applying the convolutions
# kernel = kernel1 * kernel2
conv = cv.filter2D(img, cv.CV_64F, kernel1)
conv = cv.filter2D(conv, cv.CV_64F , kernel2)
```

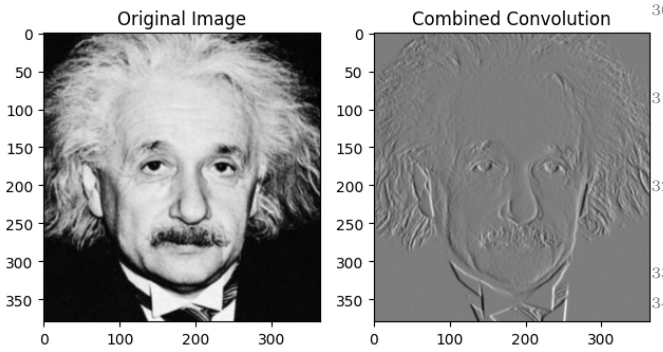

Figure 19: Gradient Magnitude using Multiplication

8 Question 08

In this question we were asked to zoom an image using nearest neighbout and bilinear interpolation methods. Nearest neighbour interpolation is a simple method in which the value of the new pixel is determined by the value of the nearest pixel in the original image. Bilinear interpolation is a more complex method in which the value of the new pixel is determined by the weighted average of the 4 nearest pixels in the original image.

```
def zoom_nearest_neighbor(image, factor):
    h, w, _ = image.shape
    new_h = int(h * factor)
    new_w = int(w * factor)
    zoomed_image = np.zeros((new_h, new_w, 3), dtype=np.uint8)
    for i in range(new_h):
        for j in range(new_w):
            orig_i = int(i / factor)
            orig_j = int(j / factor)
            zoomed_image[i, j] = image[orig_i, orig_j]

    return zoomed_image

def zoom_bilinear(image, factor):
    h, w, _ = image.shape
    new_h = int(h * factor)
    new_w = int(w * factor)
    zoomed_image = np.zeros((new_h, new_w, 3), dtype=np.uint8)
    for i in range(new_h):
        for j in range(new_w):
            orig_i = i / factor
            orig_j = j / factor
            i1, i2 = int(np.floor(orig_i)), int(np.ceil(orig_i))
            j1, j2 = int(np.floor(orig_j)), int(np.ceil(orig_j))
            i1 = max(0, min(i1, h - 1))  # Ensure indices stay within image boundaries
            i2 = max(0, min(i2, h - 1))
            j1 = max(0, min(j1, w - 1))
            j2 = max(0, min(j2, w - 1))
            value = (1 - (orig_i - i1)) * (1 - (orig_j - j1)) * image[i1, j1] + \
                    (1 - (orig_i - i1)) * (orig_j - j1) * image[i1, j2] + \
                    (orig_i - i1) * (1 - (orig_j - j1)) * image[i2, j1] + \
                    (orig_i - i1) * (orig_j - j1) * image[i2, j2]

            zoomed_image[i, j] = value.astype(np.uint8)
    return zoomed_image
```

```
zoom_factor = 2  # Change this to the desired zoom factor
# Zoom using nearest-neighbor
zoomed_nn = zoom_nearest_neighbor(img, zoom_factor)
# Zoom using bilinear interpolation
zoomed_bilinear = zoom_bilinear(img, zoom_factor)
```

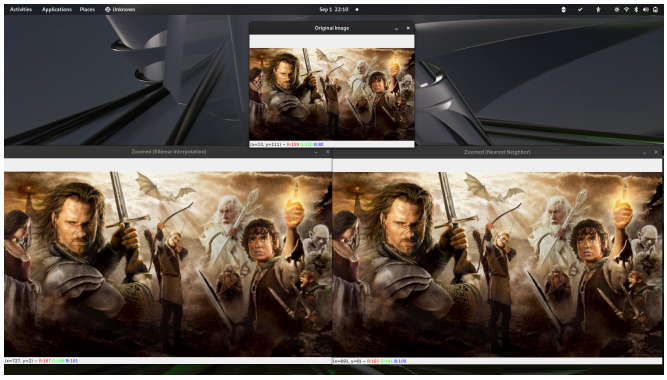

Figure 20: Zooming of an image

9 Question 09

In this question we were asked to come over a mask to extract the foreground from a given image and apply a gaussian filter to the extracted background to make it blur.

```
# Initialize the mask (1s for sure foreground, 0s for sure background)
mask = np.zeros(image.shape[:2], np.uint8)
# Define the rectangular region of interest for initial segmentation
rect = (35, 35, image.shape[1] - 25, image.shape[0] - 25)

bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
cv.grabCut(image, mask, rect, bgdModel, fgdModel, 5, cv.GC_INIT_WITH_RECT)

# Modify the mask to create a binary foreground mask
foreground_mask = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
foreground_image = image * foreground_mask[:, :, np.newaxis]
background_image = image * (1 - foreground_mask[:, :, np.newaxis])
```

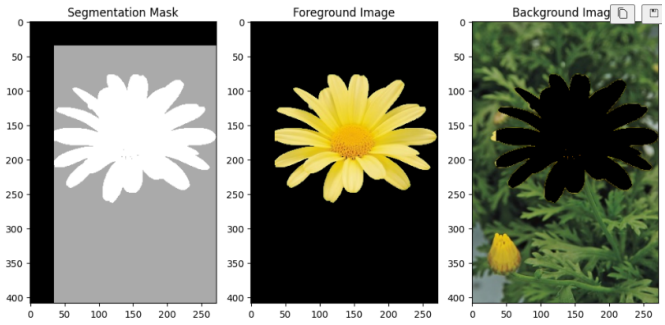

Figure 21: Mask and extraction of foreground and background

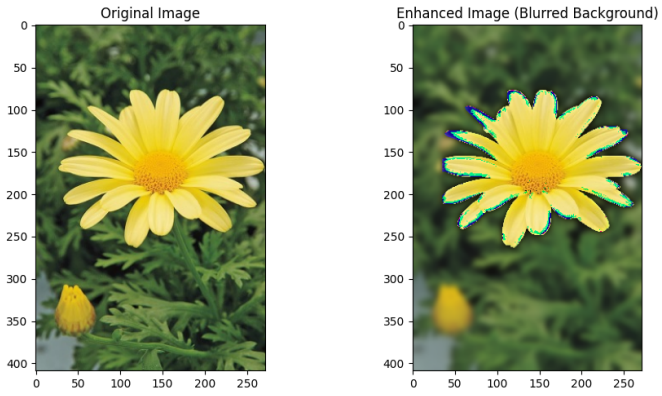

Figure 22: Blurred background and result

10 References

- OpenCV Documentation
- Youtube Resources

11 Github Repository

Following is the link to my Github repository for this assignment.

Github/EN3160_Assignment_01