EN3160 Assignment 1 on Intensity Transformations and Neighborhood Filtering

Index No: 200285E

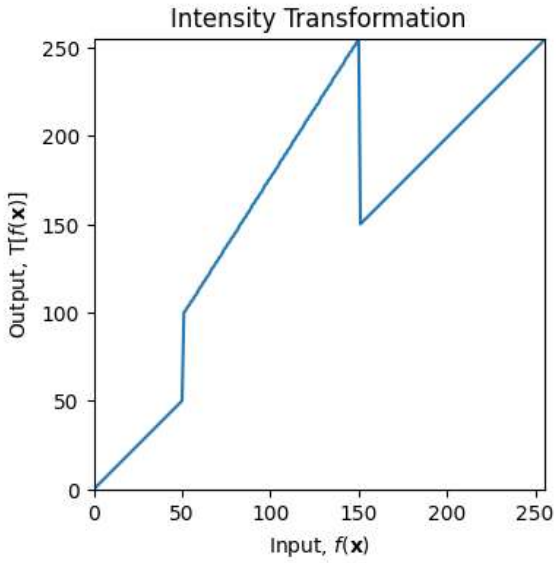
GitHub link:https://github.com/kannangaranv/EN3160_Image_Processing_and_Machine_Vision

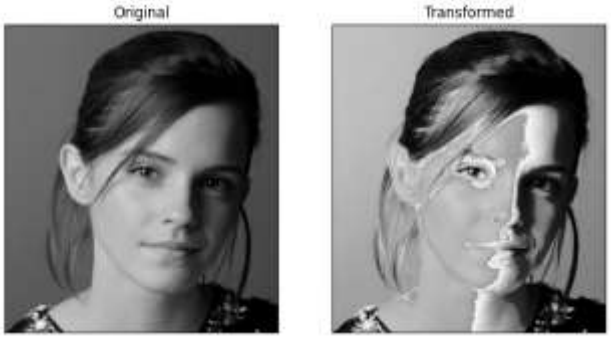
Question 1

Code:

```
#Identify points that change
transformations and put them to array
c =
np.array([(50,50),(50,100),(150,255),(1
50,150)])
#Apply corresponding transformation to
each region
t1 = np.linspace(0, c[0,1], c[0,0]+1 -
0).astype('uint8')
t2 = np.linspace(c[1,1], c[2,1], c[2,0]
- c[1,0]).astype('uint8')
t3 = np.linspace(c[3,1], 255, 255 -
c[3,0]).astype('uint8')
#Combine the transformations
transform = np.concatenate((t1,t2),
axis=0).astype('uint8')
transform =
np.concatenate((transform,t3),
axis=0).astype('uint8')
#Apply the transformation to image.
image_transformed = cv.LUT(img_orig,
transform)
```

Results:





Interpretation & Discussion:

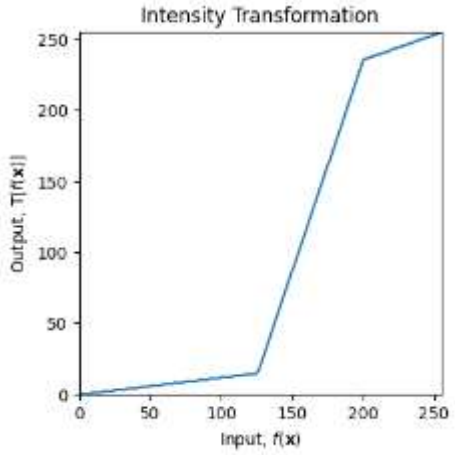
This is a question related to intensity transformation. So, I first identified the points where change the transformation and divided the color range (0-255) into three regions. Then applied corresponding transformation to each region according to graph given. Finally, combined the three transformations and applied the transformation to the given image.

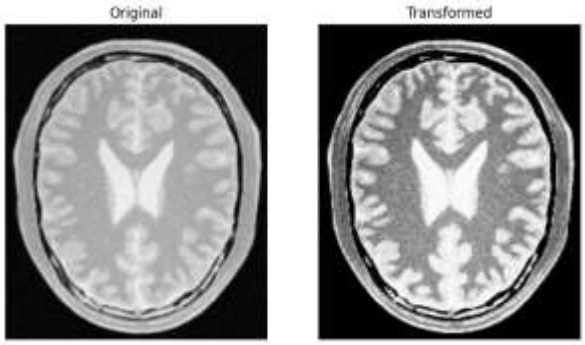
Question 2

Code:

```
#Identify and points that change
transformations and put them to array
c = np.array([(125,15),(200,235)])
#Apply corresponding transformation to
each region
t1 = np.linspace(0, 15, c[0,0]+1 -
0).astype('uint8')
t2 = np.linspace(c[0,1] + 1, c[1,1],
c[1,0] - c[0,0]).astype('uint8')
t3 = np.linspace(c[1,1] + 1, 255, 255 -
c[1,0]).astype('uint8')
#Combine the transformations
transform = np.concatenate((t1,t2),
axis=0).astype('uint8')
transform =
np.concatenate((transform,t3),
axis=0).astype('uint8')
#Apply the transformation to image.
image_transformed = cv.LUT(img_orig,
transform)
```

Results:





Interpretation and Discussion:

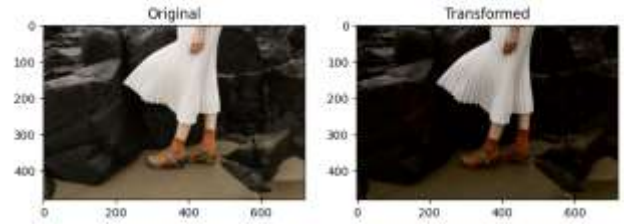
This is a question which is similar to the above question. Here we had to accentuate white matter and gray matter. To accentuate white matters, a wide region of light region was mapped into a narrow region of light region. As well, to accentuate gray matter, a wide region of dark region was mapped into a narrow region of dark region. So, using the above results we can clearly see how white matter and gray matter have accentuated.

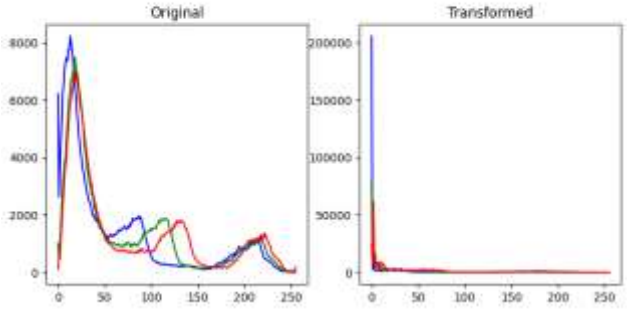
Question 3

Code:

```
#Convert the BGR to LAB format
lab_img = cv.cvtColor(img_orig,
cv.COLOR_BGR2LAB)
#Split the image into L, a, b planes.
L,a,b = cv.split(lab_img)
#Apply Gamma correction to  L plane.
gamma = 2
table =
np.array([(i/255.0)**gamma*255.0 for i
in np.arange(0,256)]).astype('uint8')
gamma_L = cv.LUT(L, table)
new_lab_img = cv.merge((gamma_L,a,b))
#Convert LAB to BGR format
new_img = cv.cvtColor(new_lab_img,
cv.COLOR_LAB2BGR)
```

Results:





Interpretation & Discussion:

This is a question which is related to Gamma correction. It was mentioned to apply gamma correction to L plane in L*a*b color space. So, first I split the color space into L,a,b three planes. Then applied gamma correction only to L plane. Here I used the gamma value as **2**. Then merged the three planes again. Using the histograms we can verify that how the gamma correction has affected to r,g,b planes. By comparing the original and transformed image, we can see how the gamma correction has controlled the brightness of the original image.

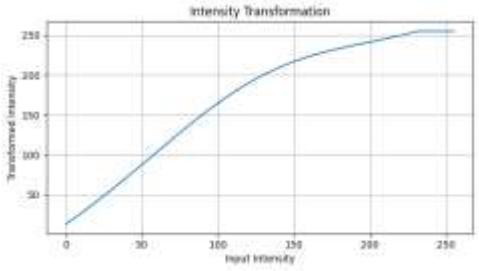
Question 4

Code:

```
# Function for intensity transformation
def transformation(x, a, sig = 70):
    return np.minimum(x + ((a*128) *
np.exp(-(x - 128)**2 / (2 * sig**2))),
255)
```

```
#Convert RGB to HSV format.
hsv_img = cv.cvtColor(img_orig,
cv.COLOR_BGR2HSV)
# Split the image into hue, saturation,
and value.
hue, saturation, value =
cv.split(hsv_img)
# Apply Intensity transformation to the
saturation plane.
a = 0.55
transformed_vec =
np.vectorize(transformation)
transformed =
transformed_vec(saturation,
a).astype(np.uint8)
new_hsv_image = cv.merge((hue,
transformed, value))
new_image = cv.cvtColor(new_hsv_image,
cv.COLOR_HSV2RGB)
#Transformation curve
x = np.linspace(0, 255, 256)
transformation_curve =
transformation(x, a)
```

Results:



Interpretation & Discussion:

This is a question related to increasing vibrance of a image. For that first I split the given image into hue, saturation and value planes. Then applied the given intensity transformation to the saturation plane. Then combined the three planes. Here I used 0.55 value for **a**. So, using the above results we can see how the vibrance has increased in order to get a visually pleasing output.

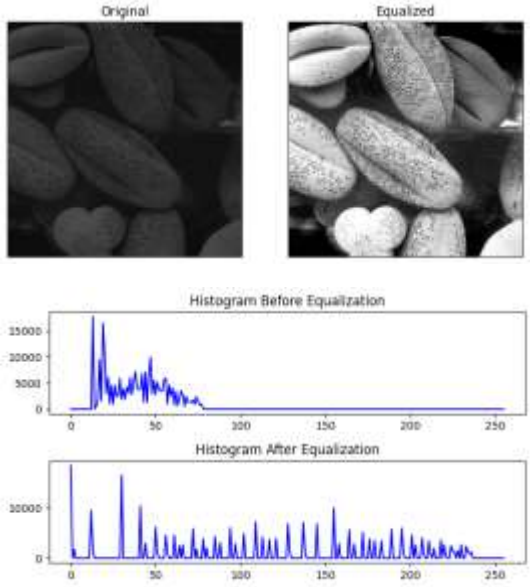
Question 5

Code:

```
#Function to historgram equalization
def histogram_equalization(image):
    def calculate_histogram(image):
        histogram =
Counter(image.flatten())
        return histogram

    def calculate_cdf(histogram):
        cdf = dict()
        cum_sum = 0
        for intensity, freq in
sorted(histogram.items()):
            cum_sum += freq
            cdf[intensity] = cum_sum
        return cdf

    def normalize_cdf(cdf, image):
        cdf_min = min(cdf.values())
        normalized_cdf = {k: ((v-
cdf_min)/(image.size-1))*255 for k, v
in cdf.items()}
        return normalized_cdf

    histogram =
calculate_histogram(image)
    cdf = calculate_cdf(histogram)
    normalized_cdf = normalize_cdf(cdf,
image)
```

```
    #Apply normalized cdf to each
pixel.
    equalized_image =
np.array([[normalized_cdf[pixel] for
pixel in row] for row in image])
    return equalized_image
```

Results:



Interpretation & Discussion:

This is a question to create a function to histogram equalization. Here I first calculated the histogram of the input image which gives the frequency distribution of intensity values in the image. Then, based on the histogram calculated the cdf. The normalized_cdf function used to normalize the cdf values to span the entire range of intensity values (0,255). Then applied normalized cdf to each pixel in the input image.

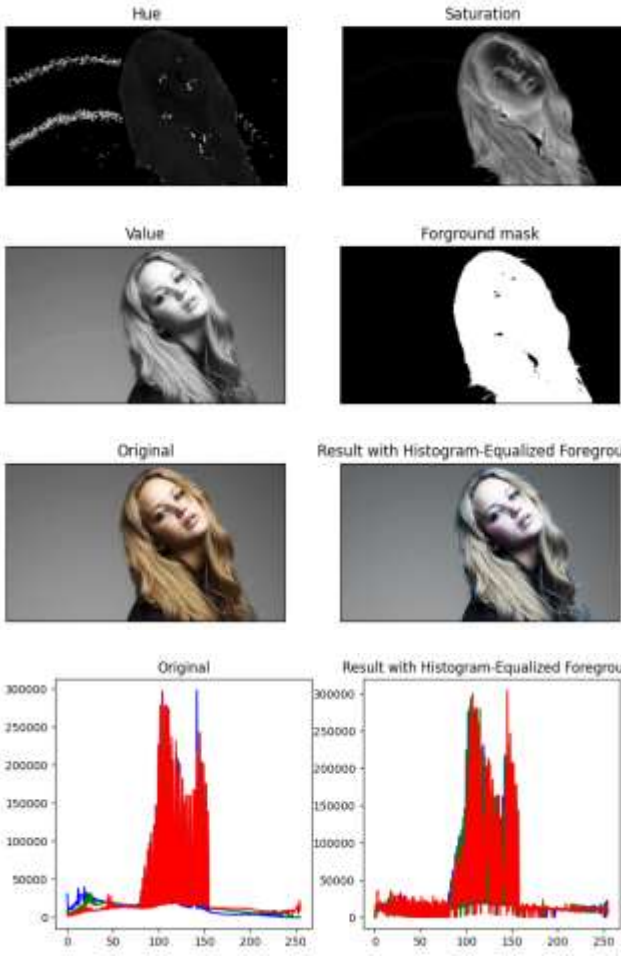
Question 6

Code:

```
#convert BGR to HSV format.
image_hsv = cv.cvtColor(image,
cv.COLOR_BGR2HSV)
# Split the image into hue, saturation,
and value planes
hue, saturation, value =
cv.split(image_hsv)
# Threshold to extract the foreground
mask
_, foreground_mask =
cv.threshold(saturation, 15, 255,
cv.THRESH_BINARY)
# Extract the foreground only using
cv2.bitwise_and
foreground_saturation =
cv.bitwise_and(saturation,saturation,
mask=foreground_mask)
foreground_hue =
cv.bitwise_and(hue,hue,
mask=foreground_mask)
foreground_value =
cv.bitwise_and(value,value,
mask=foreground_mask)
foreground_hsv =
cv.merge((foreground_hue,foreground_sat
uration,foreground_value))
#split the foreground to r,g,b planes
foreground_rgb =
cv.cvtColor(foreground_hsv,
cv.COLOR_HSV2RGB)
r, g, b = cv.split(foreground_rgb)
color = ('r', 'g', 'b')
palate = np.array([])
#Apply histogram equalization to each
r,g,b plane of the foreground.
for i, c in enumerate(color):
    foreground_hist =
cv.calcHist([foreground_rgb], [i],
foreground_mask, [256], [0, 256])
    np.set_printoptions(precision=8,
suppress=True)
    cdf = np.cumsum(foreground_hist)
    cdf_normalized = cdf * (255 /
np.sum(foreground_hist))
    rounded =
np.round(cdf_normalized).astype(int)
    palate = np.append(palate, rounded)

r_equalized = cv.LUT(r, palate[0:256])
g_equalized = cv.LUT(g,
palate[256:512])
b_equalized = cv.LUT(b,
palate[512:768])
foreground_equalized =
cv.merge((r_equalized, g_equalized,
b_equalized))
foreground_equalized =
foreground_equalized.astype(np.uint8)
#Extract the background only.
background_hue= cv.bitwise_and(hue,
hue,
mask=cv.bitwise_not(foreground_mask))
```

```
background_saturation =
cv.bitwise_and(saturation, saturation,
mask=cv.bitwise_not(foreground_mask))
background_value =
cv.bitwise_and(value, value,
mask=cv.bitwise_not(foreground_mask))
background_hsv =
cv.merge((background_hue,background_sat
uration,background_value))
background_rgb =
cv.cvtColor(background_hsv,
cv.COLOR_HSV2RGB)
# Add the histogram equalized
foreground with the background
result = cv.add(foreground_equalized,
background_rgb)
```

Results:



Interpretation & Discussion:

This is a question to obtain histogram equalized foreground. For this first I split the image into hue, saturation and value planes. And then displayed each plane in gray scale. Using these images, we can clearly identify that the best plane which can be used to extract foreground mask is the saturation plane. So, I selected the saturation plane as the threshold plane and its threshold value as 15. And then extracted the foreground using cv.bitwise_and function. Then applied histogram equalization to each r,g,b plane of foreground. Then I extracted the background. Then added the histogram equalized foreground with the background. By using the results, we can see that, intensity has only changed in foreground image.

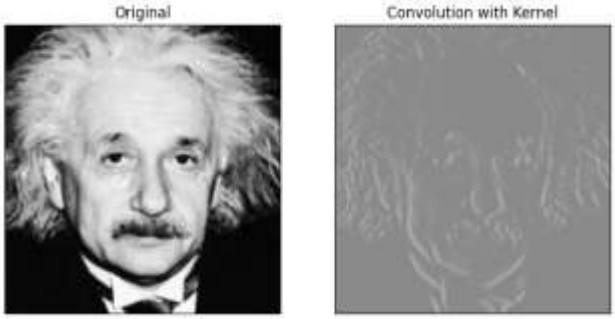
Question 7

Sobel horizontal filter using 2Dfilter function.

Code:

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

Results:



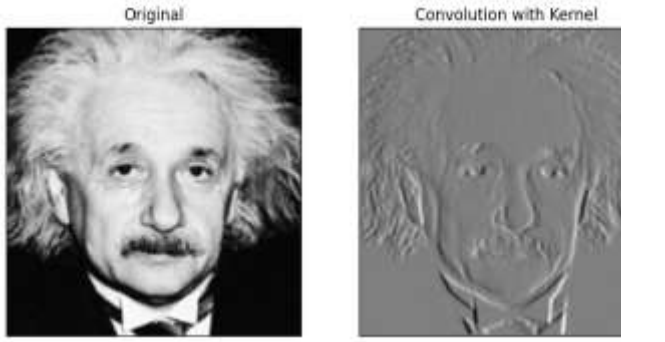
Sobel horizontal filter using own function.

Code:

```
# Define the sobel 3x3 kernel
kernel = np.array([
    [1, 0, -1],
    [2, 0, -2],
    [1, 0, -1]
])
```

```
# Get dimensions of the input matrix
and the kernel
img_height, img_width = img.shape
kernel_height, kernel_width =
kernel.shape
# Initialize an empty  matrix
img_matrix = np.zeros((img_height,
img_width))
# Perform matrix convolution
for i in range(img_height-
kernel_height+1):
    for j in range(img_width-
kernel_width+1):
        window = img[i:i+kernel_height,
j:j+kernel_width]
        img_matrix[i+1, j+1] =
np.sum(window * kernel)
#normalize the image.
new_image = ((img_matrix-
np.min(img_matrix))/(np.max(img_matrix)
-np.min(img_matrix)))*255
```

Results:



Interpretation & Discussion:

This is a question related to filtering with sobel operation. Here I first got the kernel as a 2D array. Images already existed as 2D arrays. Then I created an empty array equal to the original image's dimensions. Then performed convolution among kernel and the original image using nested loops. The resulted convolution value is stored in corresponding position of the empty array. This process is done for all the positions in the image. After the convolution the result is normalized to have values between 0 and 255. By the above results we can see that how the edges in the input image have highlighted.

Question 8

Code:

```
#function to zoom using nearest
neighbour
def zoom_nearest(img, zoom_factor):
    old_h, old_w = img.shape[:2]
    new_h = int(old_h * zoom_factor)
    new_w = int(old_w * zoom_factor)
    scaled_img = np.zeros((new_h,
new_w, img.shape[2]), dtype=np.uint8)

    for i in range(new_h):
        for j in range(new_w):
            initial_i = int(i /
zoom_factor)
            initial_j = int(j /
zoom_factor)
            scaled_img[i, j] =
img[initial_i, initial_j]

    return scaled_img

# Function to Compute the normalized
sum of squared difference (SSD)
def normalized_ssd(img1, img2):
    return np.sum((np.array(img1,
dtype=np.float32) - np.array(img2,
dtype=np.float32))**2)

# Set the zoom factor (s ∈ (0, 10])
zoom_factor = 4.0
# Zoom using nearest-neighbor
near_img = zoom_nearest(original_img,
zoom_factor)
# Zoom using bilinear interpolation
bilinear_img =
cv.resize(original_img,None, fx =
zoom_factor, fy = zoom_factor,
interpolation = cv.INTER_LINEAR)
#Calculate the normalized ssd to
compare sacled image and originally
scaled image
near_normalized_ssd =
normalized_ssd(scaled_img,near_img)
bilinear_normalized_ssd =
normalized_ssd(scaled_img,bilinear_img)
```

Results:


```
Normalized SSD between nearest
neighbour scaled image and original
scaled image: 847702528.0000
Normalized SSD between bilinear scaled
image and original scaled image:
715963392.0000
```

Interpretation & Discussion:

This is a question which is related to the zooming process. Here I created a function to zoom using nearest neighbor based on the definition of nearest neighbor zooming. I used a existing function to zoom using bilinear interpolation. When comparing the normalized ssd values of nearest neighbor and bilinear interpolation, small value is given by the bilinear interpolation. So, zooming using bilinear interpolation is better than zooming using nearest neighbor.

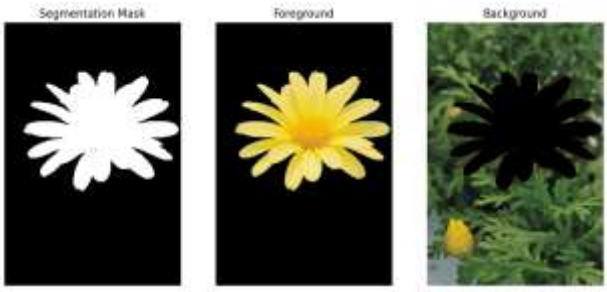
Question 9

Code:

```
# Define a rectangle around the around
the flower
rec = (0, 50, 350, 200)
# Initialize mask and
background/foreground models
mask = np.zeros(original_img.shape[:2],
np.uint8)
back_model = np.zeros((1, 65),
np.float64)
fore_model = np.zeros((1, 65),
np.float64)
# Apply GrabCut algorithm
cv.grabCut(original_img, mask, rec,
back_model, fore_model, 5,
cv.GC_INIT_WITH_RECT)
Modified_mask = np.where((mask == 2) |
(mask == 0), 0, 1).astype('uint8')
# Apply the mask to the original image
foreground_img = original_img *
Modified_mask[:, :, np.newaxis]
background_img = original_img -
foreground_img

# Apply gamma correction to the
foreground image
gamma1 = 1.5
table1 = np.array([(i / 255.0) **
gamma1 * 255.0 for i in np.arange(0,
256)]).astype('uint8')
H, S, V = cv.split(foreground_img)
gamma1_S = cv.LUT(S, table1)
foreground_img_gamma1 = cv.merge((H,
gamma1_S, V))

gamma2 = 5
table2 = np.array([(i / 255.0) **
gamma2 * 255.0 for i in np.arange(0,
256)]).astype('uint8')
L, a, b =
cv.split(foreground_img_gamma1)
gamma2_L = cv.LUT(L, table2)
foreground_img_gamma2 =
cv.merge((gamma2_L, a, b))
```

```
# Apply Gaussian blur to the background
blurred_background =
cv.GaussianBlur(background_img, (15,
15), sigmaX=0.8)
#Dark the background
brightness_factor = 0.7
dark_burred_background =
np.clip(blurred_background *
brightness_factor, 0,
255).astype(np.uint8)

# Apply gamma correction to the
background image
gamma3 = 1.5
gamma4 = 1.2
table3 = np.array([(i / 255.0) **
gamma3 * 255.0 for i in np.arange(0,
256)]).astype('uint8')
table4 = np.array([(i / 255.0) **
gamma4 * 255.0 for i in np.arange(0,
256)]).astype('uint8')
H, S, V =
cv.split(dark_burred_background)
gamma3_V = cv.LUT(V, table3)
gamma4_S = cv.LUT(S, table4)
gamma3_H = cv.LUT(H, table3)
background_img_gamma =
cv.merge((gamma3_H, gamma4_S,
gamma3_V))

# Combine the dark blurred background
and gamma-corrected foreground
enhanced_image = foreground_img_gamma2
+ background_img_gamma
```

Results:



Interpretation & Discussion:

This is a question which is related to producing an enhanced image with a substantially blurred background. For this first I selected a rectangle around the flower and applied GrabCut algorithm to produce a mask which can be used to extract foreground and background. Then apply gamma corrections to different planes of the foreground to enhance the foreground. Then I blurred the background (using Gaussian blur) and reduced the brightness of the background. As well I added gamma correction to background also. Finally, I combined the modified foreground and modified background to produce an enhanced image with a substantially blurred background.

The reason for the background just beyond the edge of the flower quite dark in the enhanced image is a result of the Gaussian blur applied to the background. The blur causes the background colors, including the dark ones, to spread out and leak into the areas around the flower's edge. This effect creates a transition from the flower to the blurred background, making the edge of the flower appear darker.