GIVING CARTOON EFFECT TO COLORFUL IMAGES

Elif Gürkan

About The Assignment

The purpose of this assignment was to use picture filters to construct a more straightforward form of real-time video abstraction. To create a cartoon-like look, the stages involved image smoothing, edge detection, quantization, and mixing the quantized and edge images.

1. Image Smoothing: A Gaussian filter with different sigma values was applied to smooth out the image. The best smoothing level was determined by observing the impacts of various sigma values. For comparison, a median filter and colvolve filter were also taken into account.

2. Edge Detection: For edge detection, the Thresholded Difference of Gaussian (DoG) filter was employed. The original image was convolved with the DoG kernel, which was defined as the difference of two Gaussian functions. To acquire edges, the filtered image underwent thresholding.

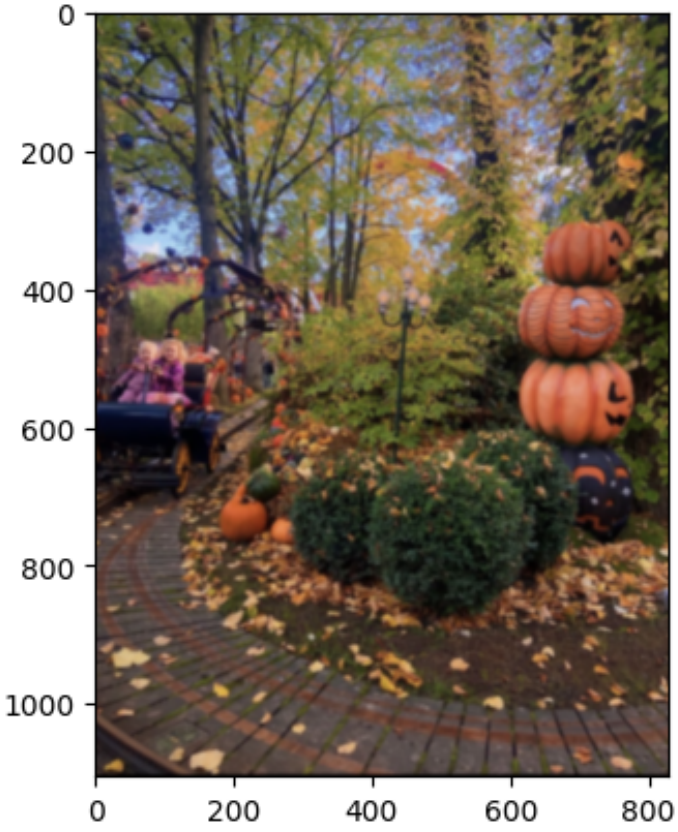
3. Image Quantization: To improve quantization outcomes, the RGB data were transformed to Lab color space. After quantizing the luminance (L) channel, the image was returned to RGB.

4. Combining the Quantized picture and Edges: For each channel, the quantized picture was multiplied by the edges that were retrieved from the smoothed image and then flipped. It was confirmed that the obtained values fell into the allowed range [0, 255].

1. Image Smoothing

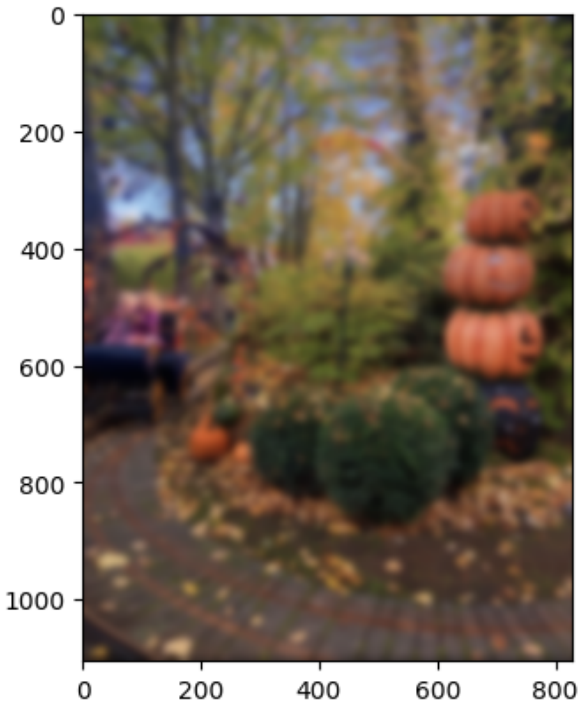
In image smoothing, I tried the functions such as: scipy.ndimage.gaussian filter, scipy.ndimage.convolve and scipy.ndimage.median filters. And after I experienced the outcomes of these filters, I have decided to go with the gaussian filter. I think it's simpler than the others and for the final outcome, which I observed for each filter, there are no big differences between the changes of using the filters.

- Using scipy.ndimage.gaussian : I basically focused on the sigma value and observed the outputs from the sigma changes. In all my inputs, as the sigma value increases, the image becomes blurrier and smoother, but in some inputs, as I increase the sigma value, the colors of the image turn into black and white.



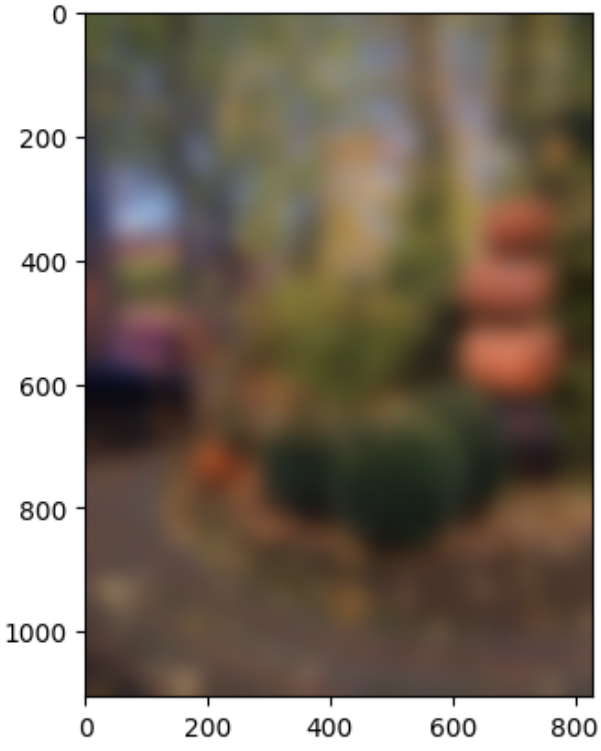
sigma = 2

It's blurry just as we wanted it to be. And it was my the best choice than the other values.



sigma = 5

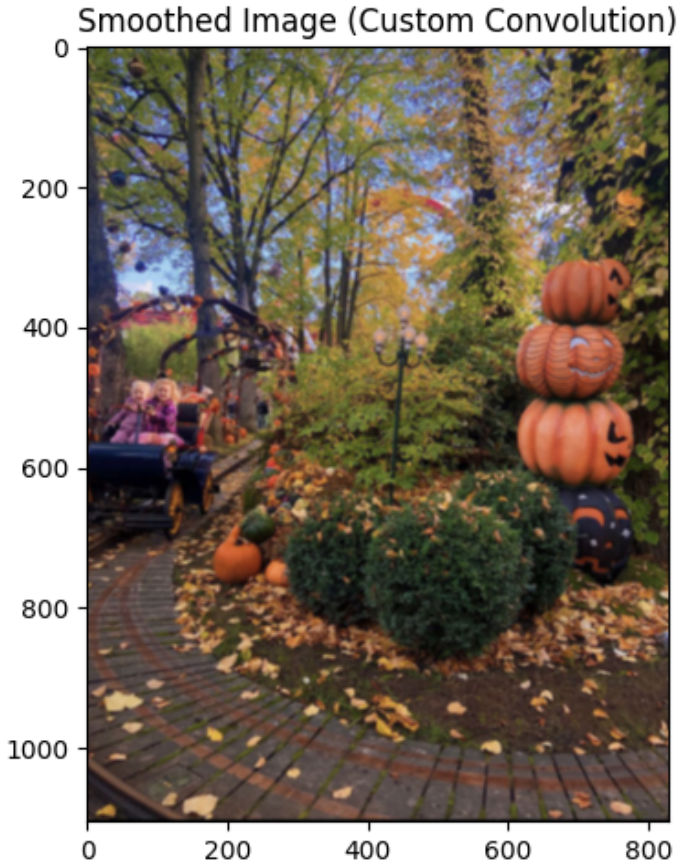
It gets more smoother when we rise the sigma value



sigma = 20

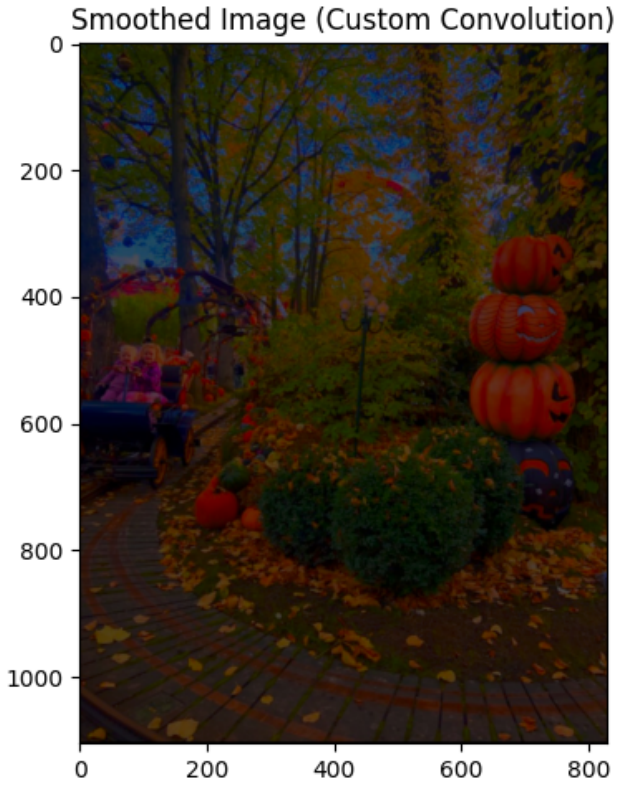
The edges are all gone.

- Using scipy.ndimage.convolve : While using this function, I focused on the kernel sizes. I've used 3x3, 5x5 and 7x7 box filters as a custom smoothing kernel. The best kernel size was a 5x5 box filter.And in the 7x7 box filter, The parts that should have been illuminated were dark and the color values of the picture seemed to be reversed. The outputs according to my observations :



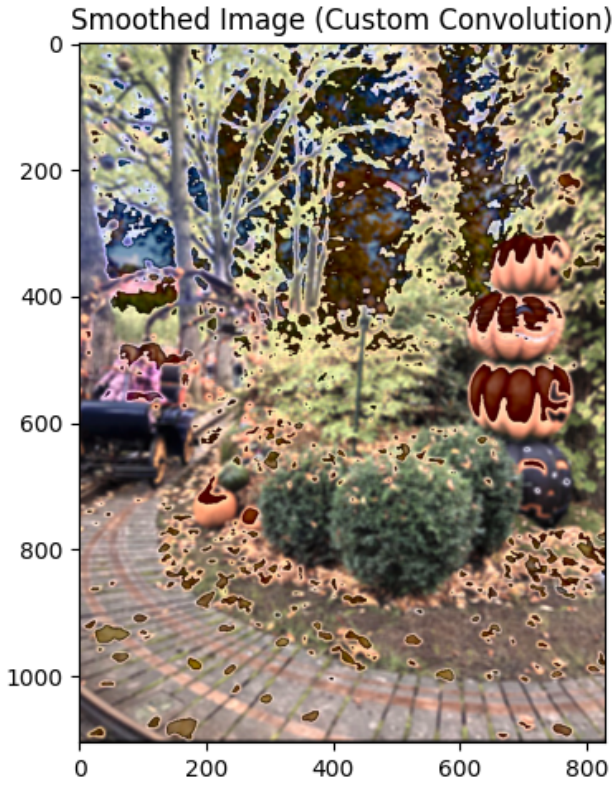
kernel size = 5x5 box filter

The edges get smoother and its blurry just as we wanted. So it is the best kernel size



kernel size = 3x3 box filter

The output is very dark and the light field values are very low with this filter.



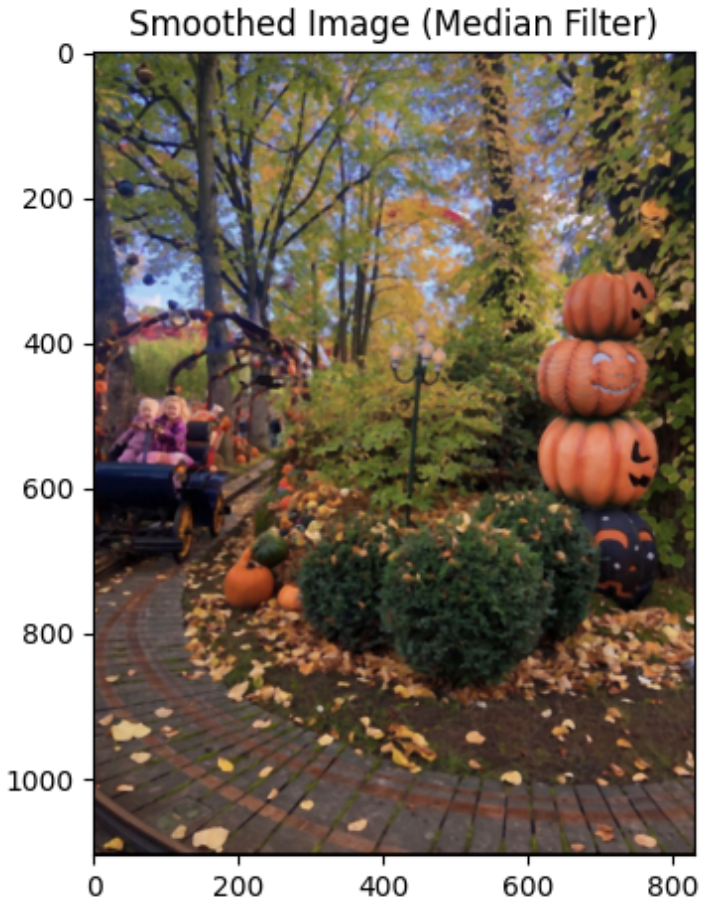
kernel size = 7x7 box filter

The parts that should have been illuminated were dark and the color values of the picture seemed to be reversed.

- Using scipy.ndimage.median_filter: The median filter function takes the input image and a specified kernel size as arguments. The median filter is effective for removing salt-and-pepper noise and preserving edges. In the application of this function,

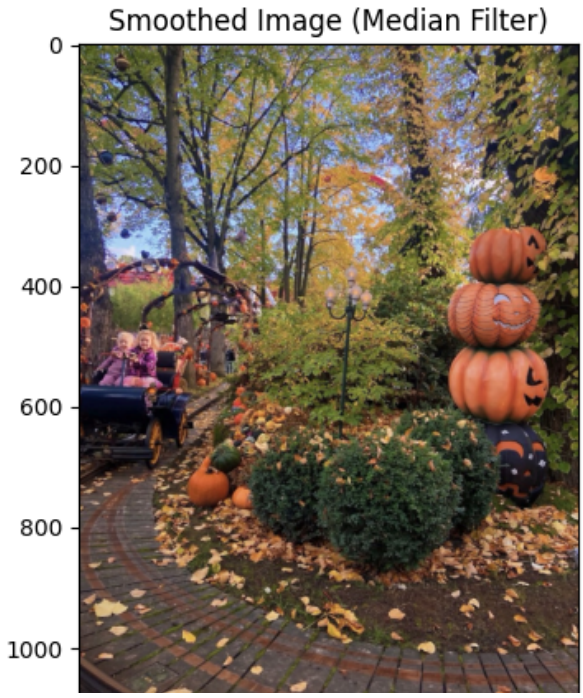
```
# Apply median filter for smoothing

smoothed_L = median_filter(L, size=5)
```

I changed the size here. The best value was size = 5 for this function. I also tried size = 2 and size = 20 values, too.



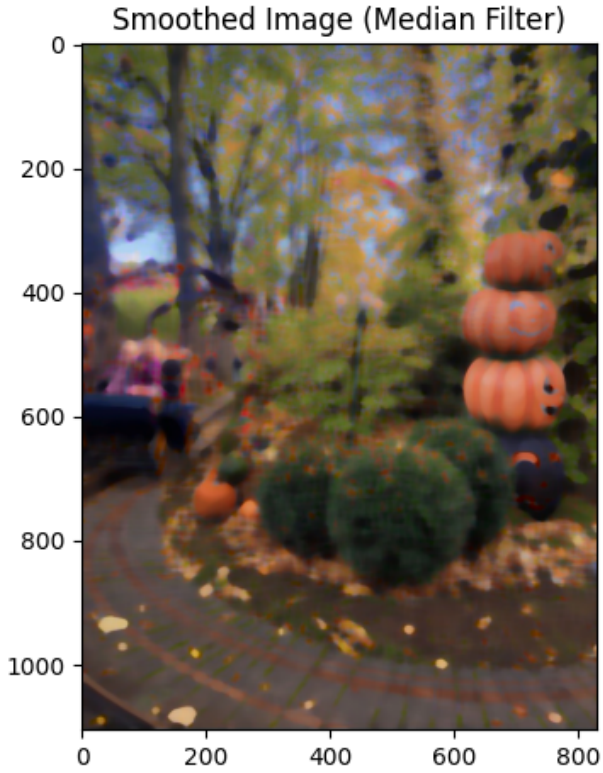
size = 5

When I tried it with this value, I got a result similar to the best values in Gaussian and convolve filters.



size =2

In size = 2, The image is not as smooth as we would like because the value is not high enough. That's why I chose not to use this value.



size = 20

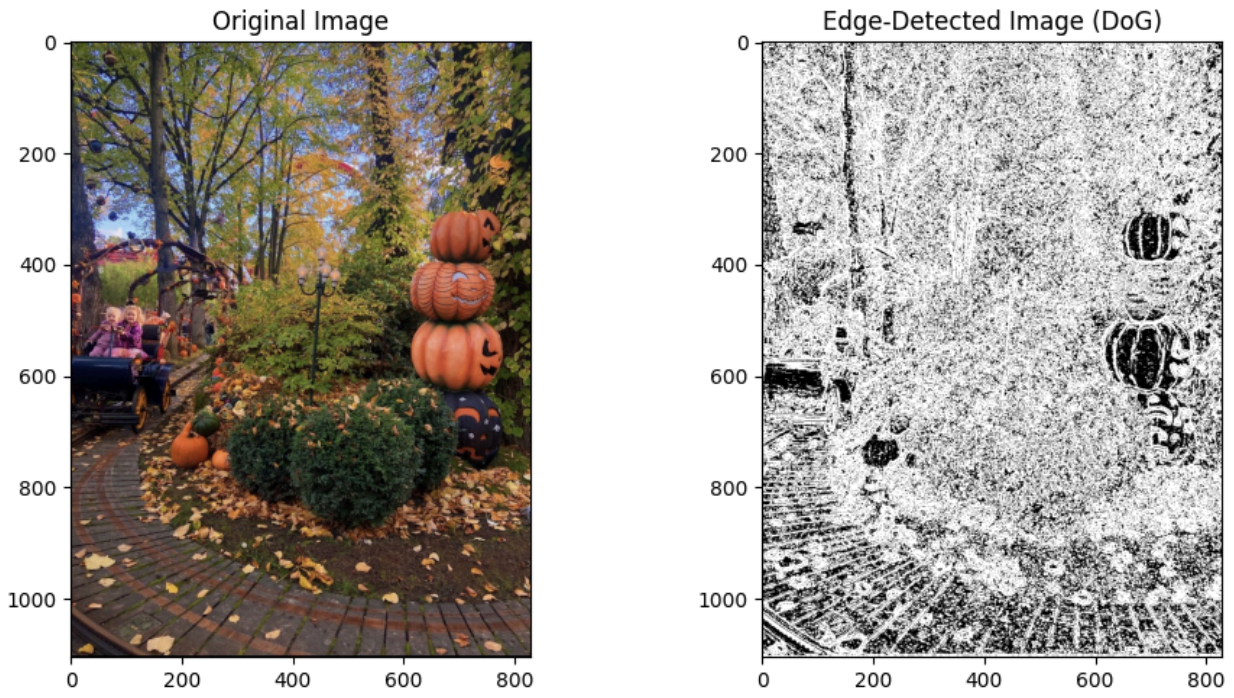
Considering that the median filter works by changing the value of each pixel with the median value of neighboring pixels, when size = 20, I encountered an image that was smoothed but whose pixel appearance increased.

2. Edge Detection

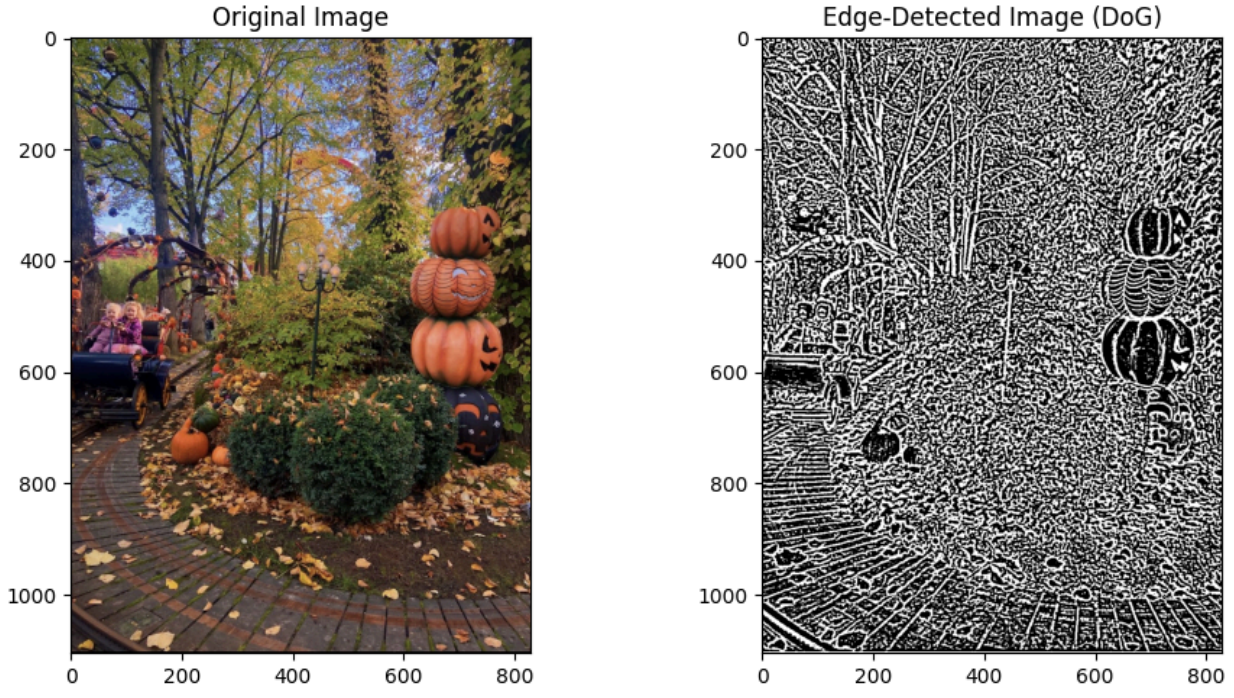
In edge detection, it's more efficient to work with grayscale images because they contain only intensity information (no color channels). This simplifies the computation.
Before performing edge detection, it's good to apply a Gaussian blur to the image. This helps in reducing noise and removing small details, which can be mistaken for edges.
I used edge detection using the Difference of Gaussians (DoG) method, a technique where two Gaussian filters with different standard deviations are applied to images to highlight edges or boundaries.

```
# standard deviations for the Gaussian filters
sigma1 = 1.5
sigma2 = 2
gaussian1 = gaussian_filter(gray_image, sigma=sigma1)
gaussian2 = gaussian_filter(gray_image, sigma=sigma2)
# DoG
edges = gaussian1 - gaussian2
threshold_value = 7
edges[edges < threshold_value] = 0
edges[edges >= threshold_value] = 255
```

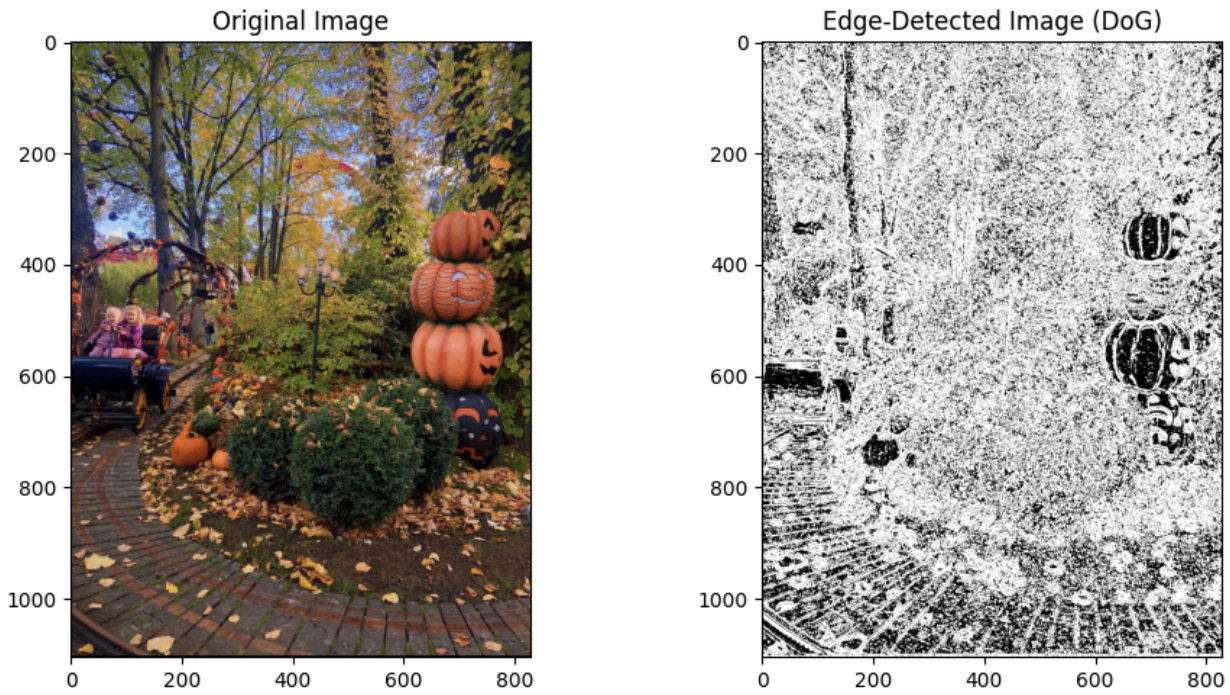
DoG is calculated by subtracting the two Gaussian-filtered images (gaussian1 and gaussian2).This operation emphasizes the edges by accentuating high-frequency components of the image (regions with rapid intensity changes).The resulting DoG image (edges) is thresholded to create a binary image that emphasizes stronger edges while suppressing weaker ones.Any pixel value below the threshold_value is set to 0 (considered as non-edge), and values above the threshold are set to 255 (considered as edges).The threshold_value is essential in controlling the sensitivity of edge detection, influencing which edges are retained based on their strength in the DoG output.



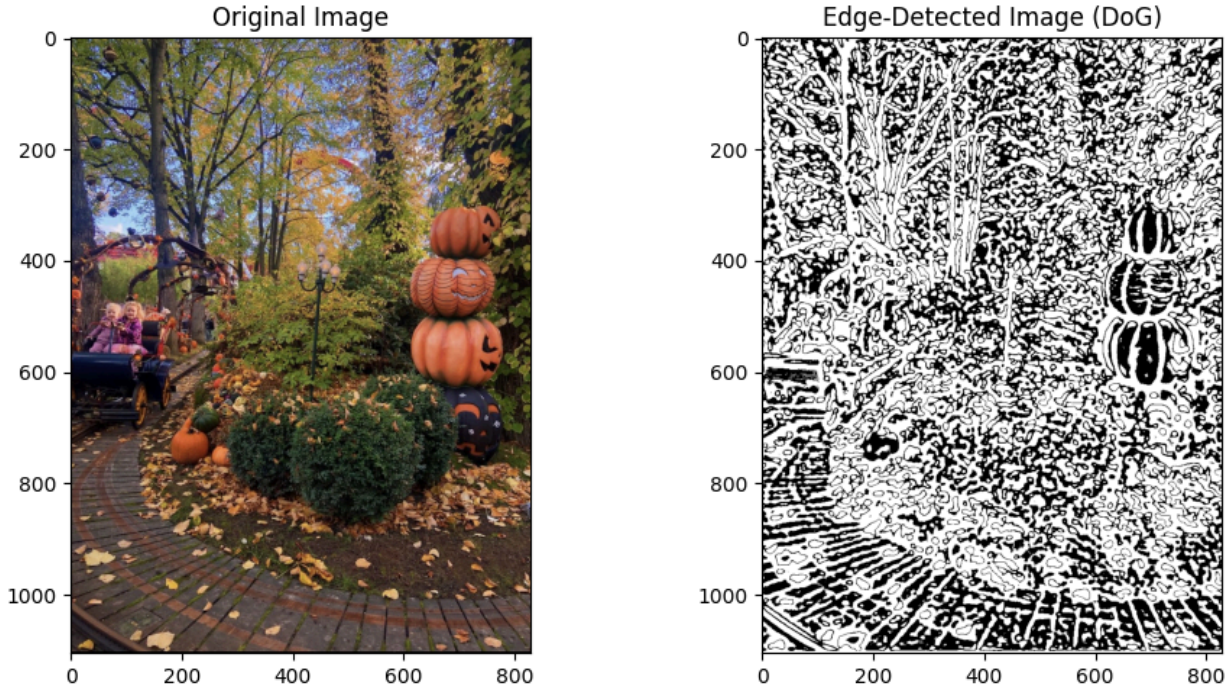
Threshold value = 2



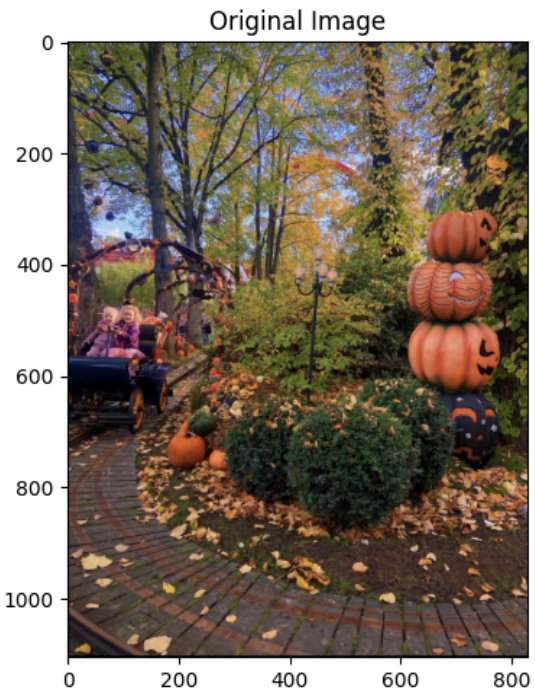
Threshold value = 7

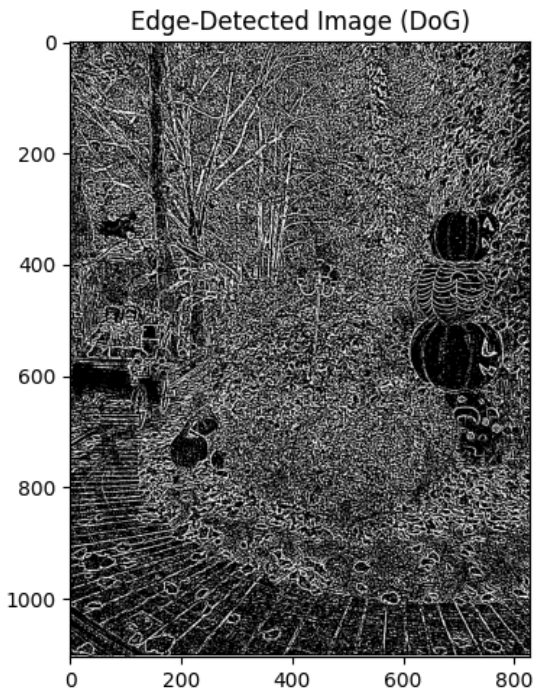


Threshold value = 254



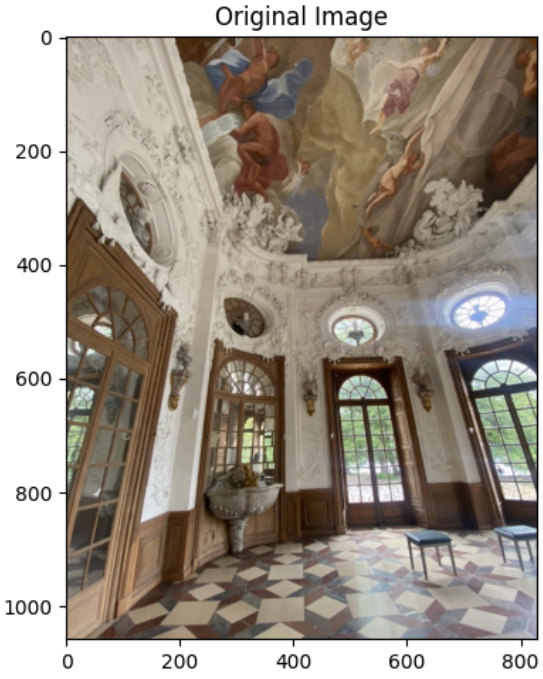
Sigma1 =1.1 Sigma2 = 1.5

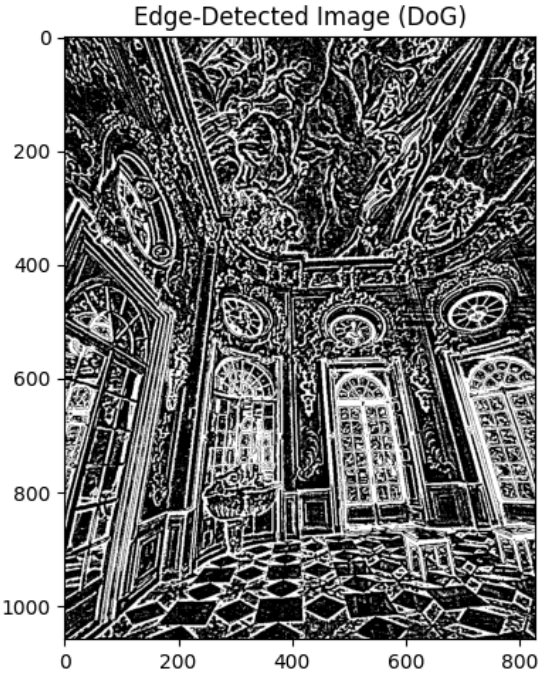


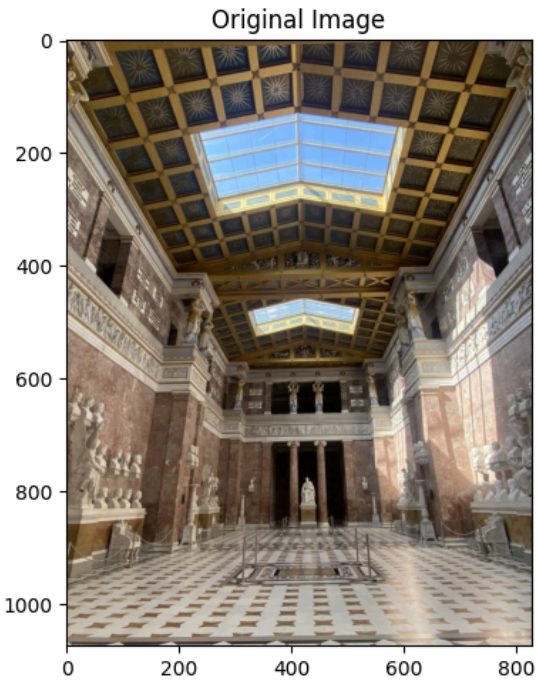


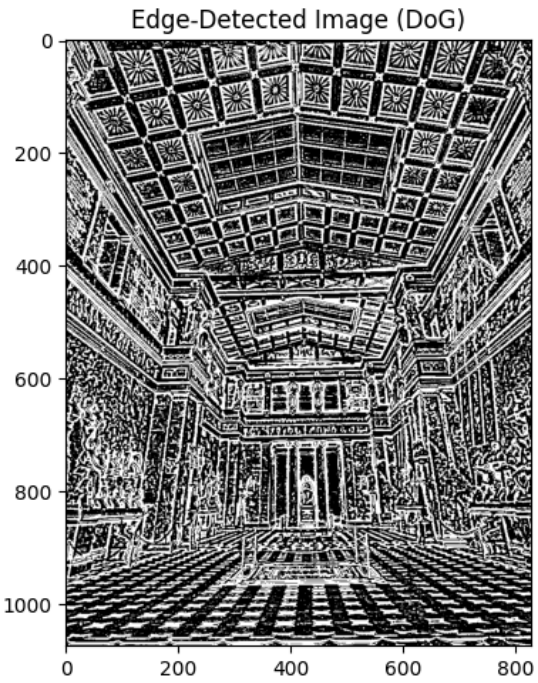
sigma1= 3 sigma2 =5

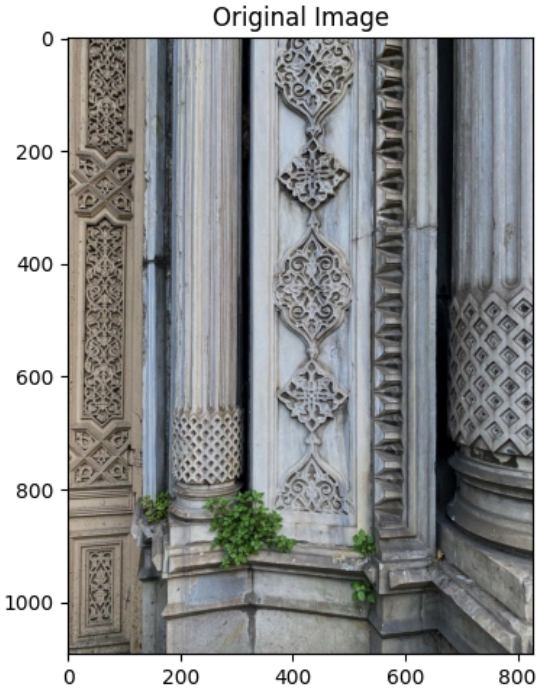
With the sigma1 = 1.5 , sigma2 = 2 and treshold_value 7;

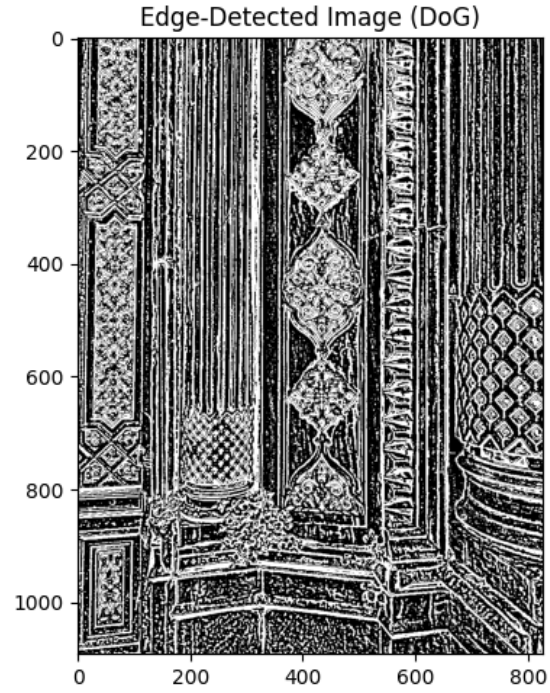


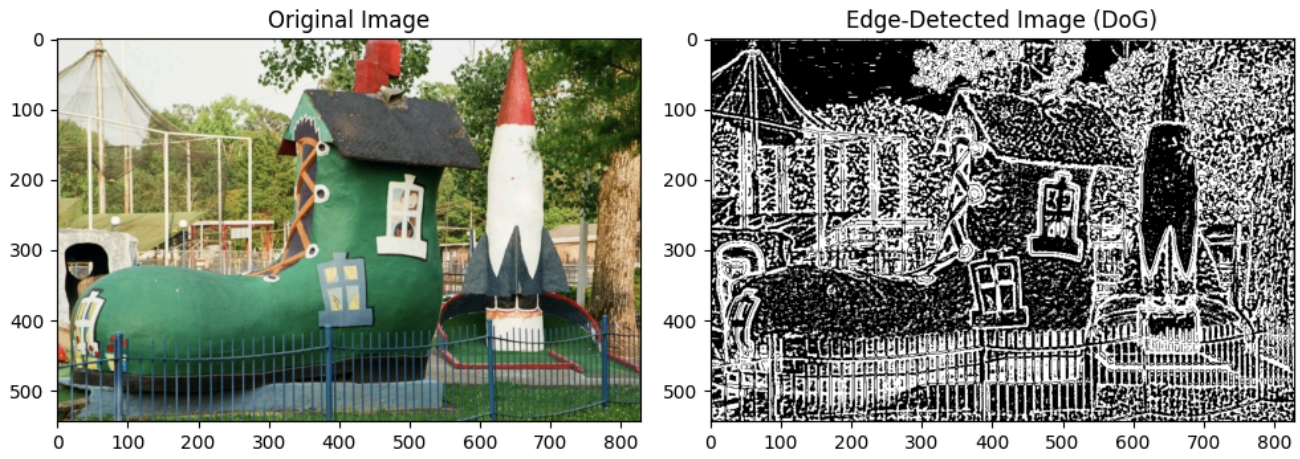



Original Image
0
200
400
600
800
1000
0
200
400
600
800

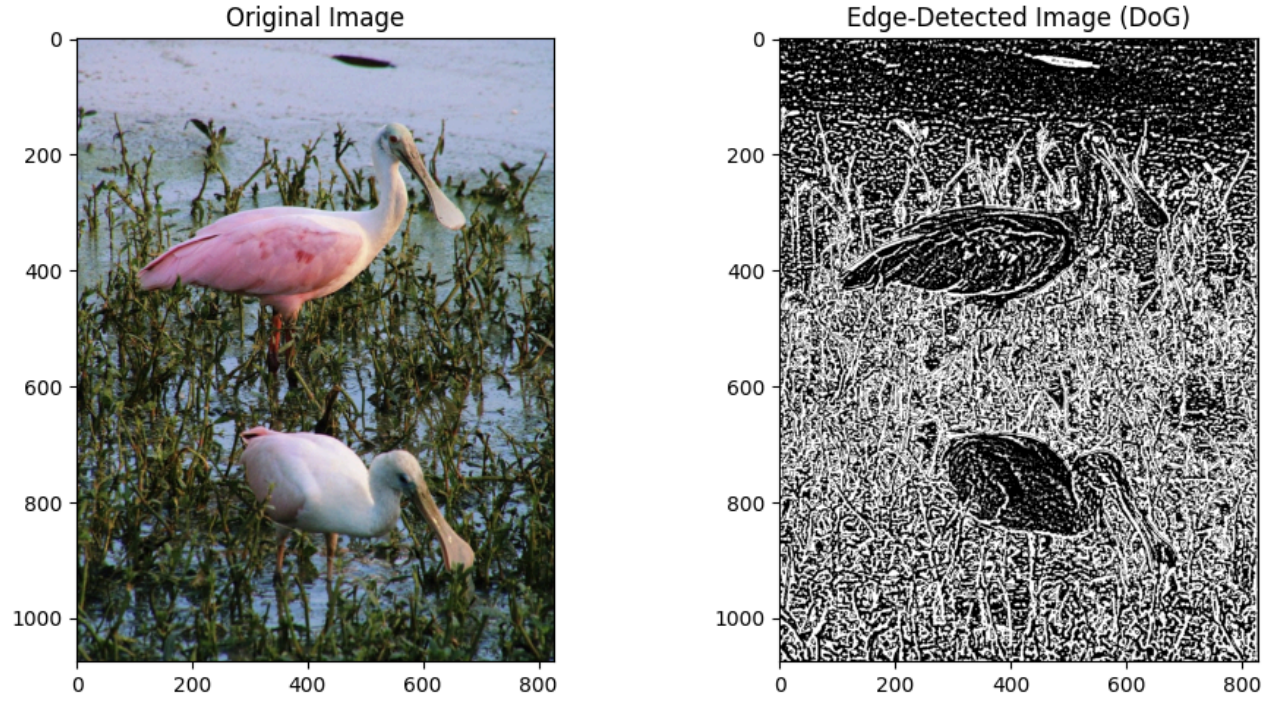

Edge-Detected Image (DoG)
0
200
400
600
800
1000
0
200
400
600
800


Original Image
0
200
400
600
800
1000
0
200
400
600
800

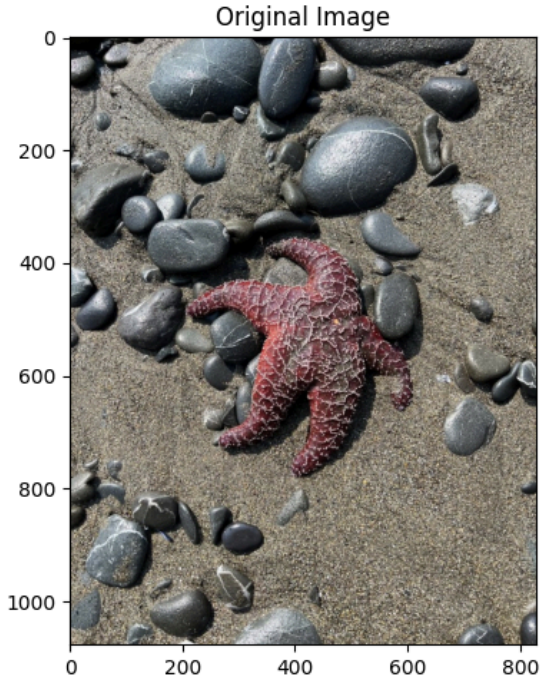

Edge-Detected Image (DoG)
0
200
400
600
800
1000
0
200
400
600
800

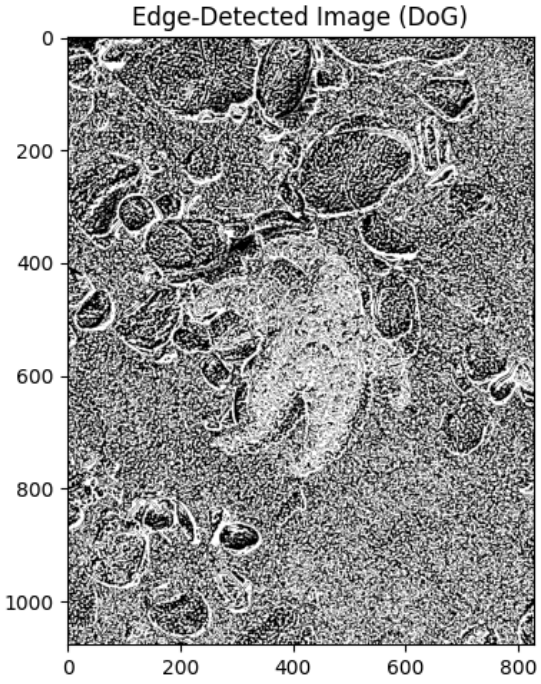


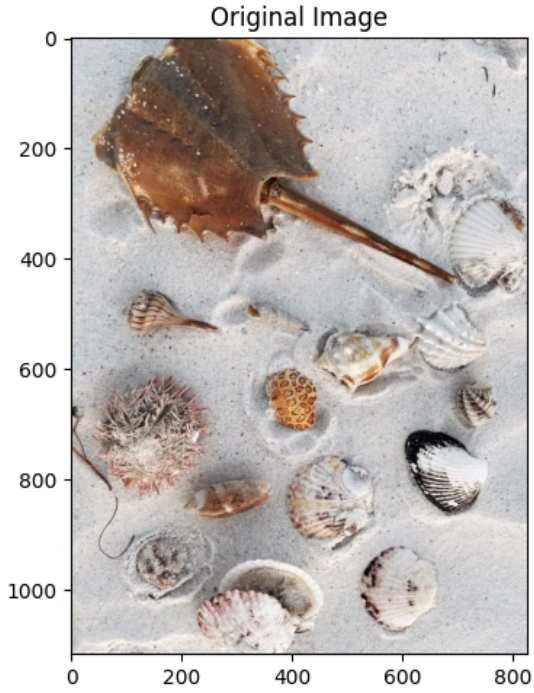
The both values were good for most of the images I used as input, but there were some images that are brightly colored and have beach sand/homogeneity.
Brightly colored images might cause a lack of contrast in edge detection algorithms. Edges are typically areas where one pixel significantly differs from another. In a very bright or homogenous image, these differences might be minimal, making it challenging for edge detection algorithms to accurately identify edges.

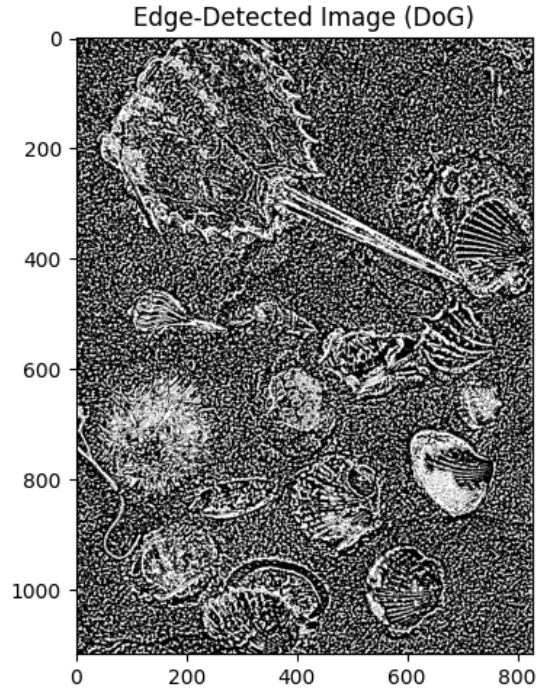


Areas like beach sand or homogenous regions might often be perceived as weak or misleading edges by edge detection algorithms. Since pixel values are similar in these areas, algorithms might identify these regions as edges. This can lead to non-edge areas being detected as edges, causing unwanted illusions.







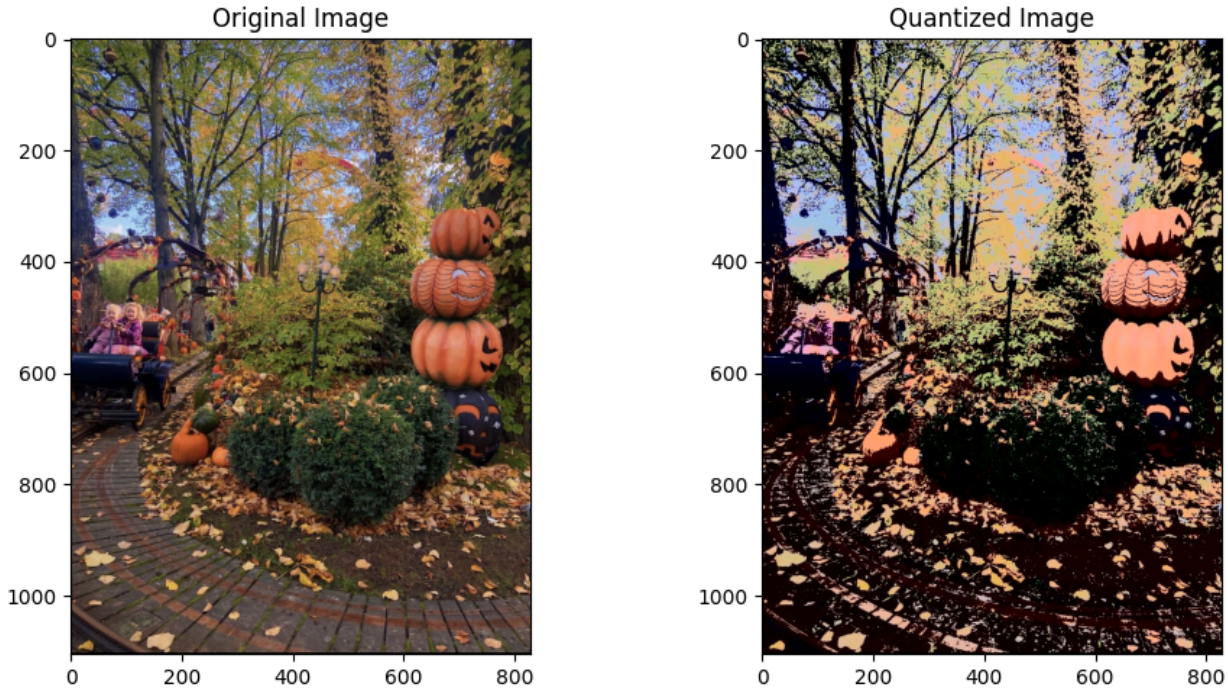


3. Image Quantization

The Lab color space separates an image into three channels: L (luminance), a (green to red), and b (blue to yellow). The L channel represents the brightness information. Using Lab color space for quantization can be advantageous because it separates color information from luminance, making it easier to manipulate.

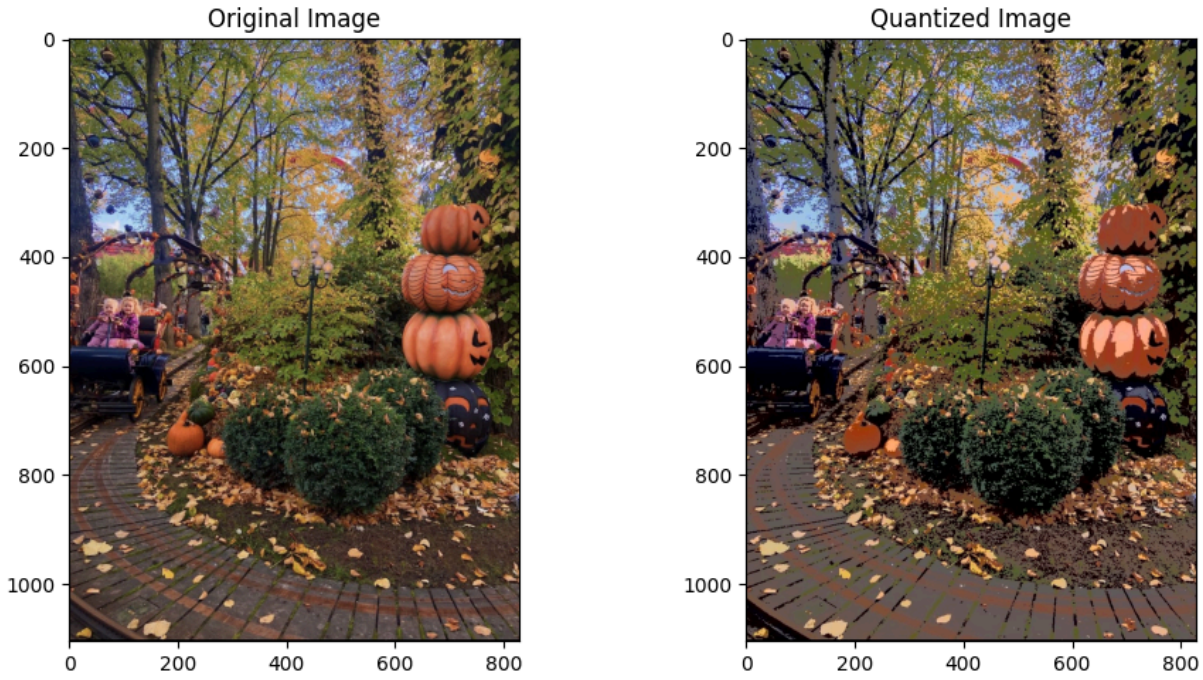
```
# the number of quantization levels
quantization_level = 2
```

This parameter determines the number of levels the luminance channel will be quantized into. So, I made changes about the value of the parameter to find the more accurate value for my image inputs.



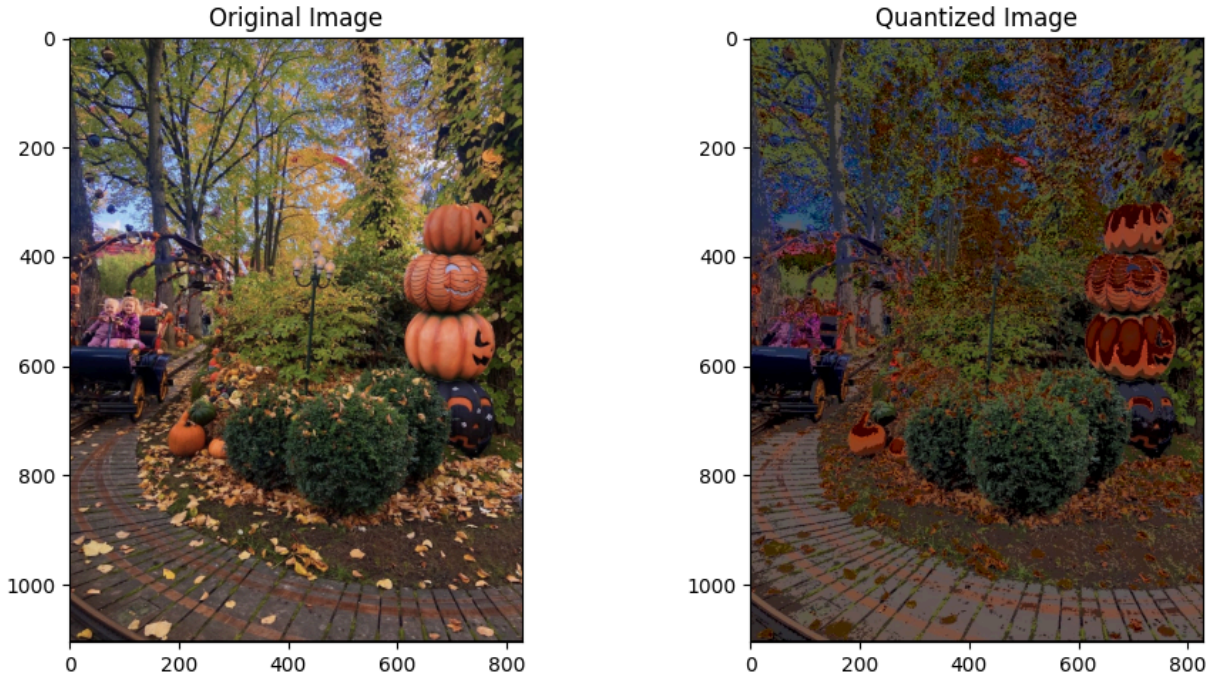
quantization_level = 1.5

When we set the level to 1.5, the bright places get brighter, the dark places get darker. So this value is not proper.



quantization_level = 2

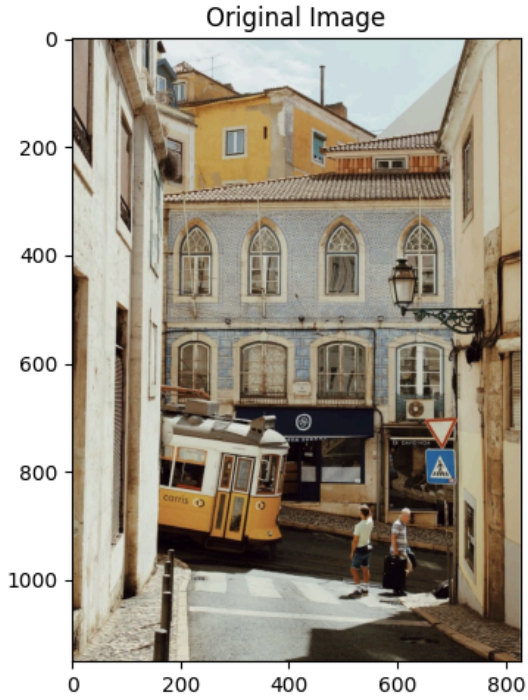
This value is the best value for this input and most of the images. The colors are fitted really well and balanced , and the details are very appropriate.

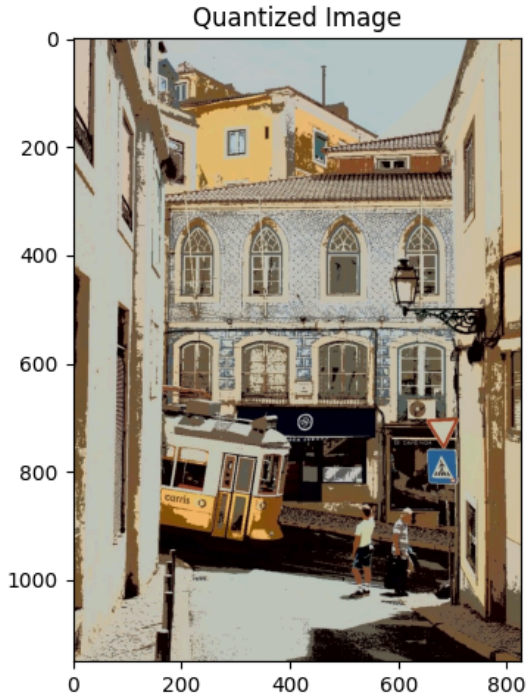


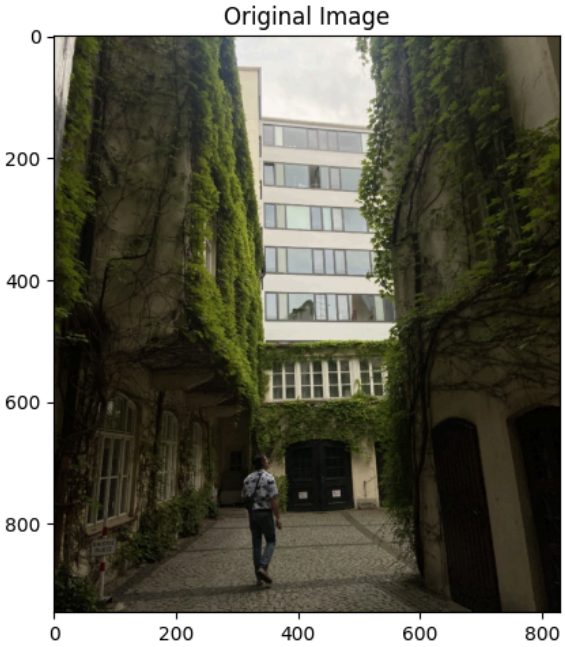
quantization_level = 3

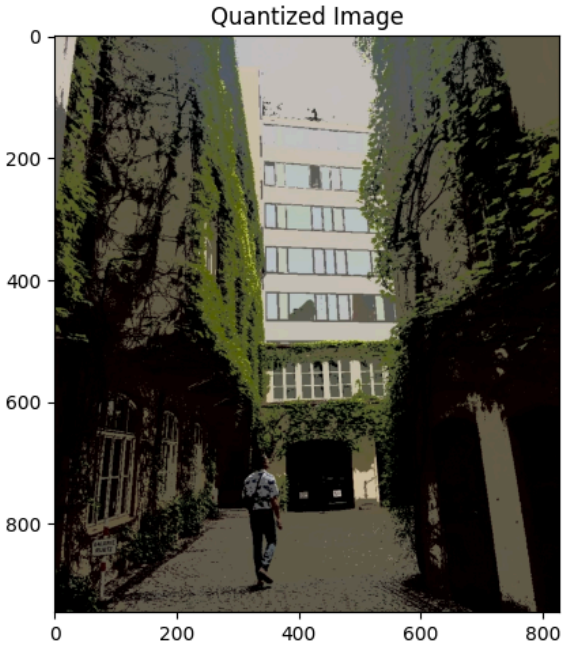
As I raise the value, the illuminated areas are decreasing. Light areas decrease and turn into negative colors.The edges and details are too much.

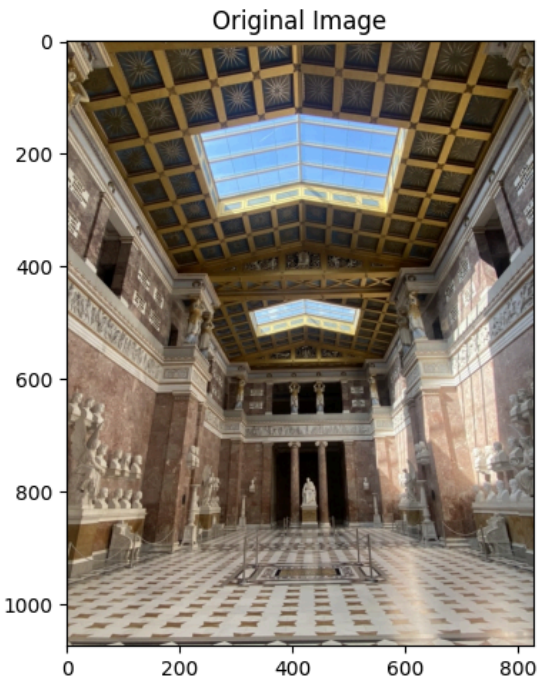
Some examples with quantization_level = 2

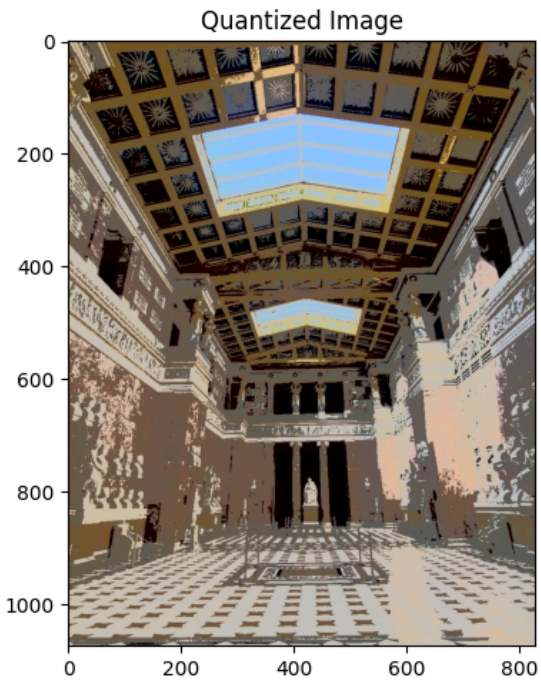




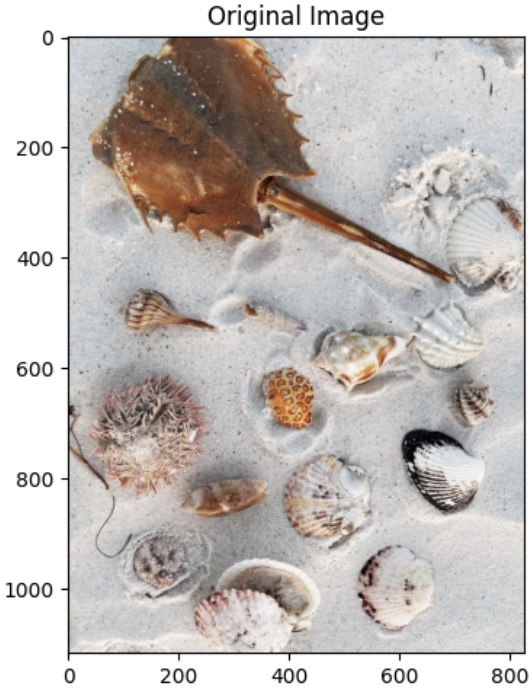


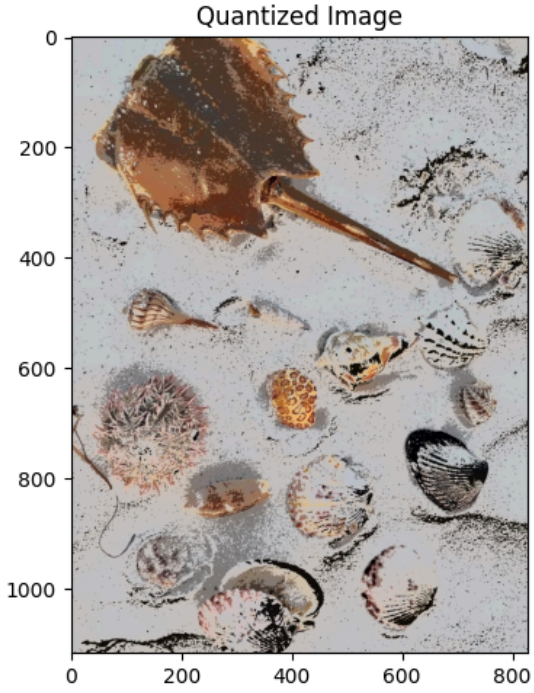




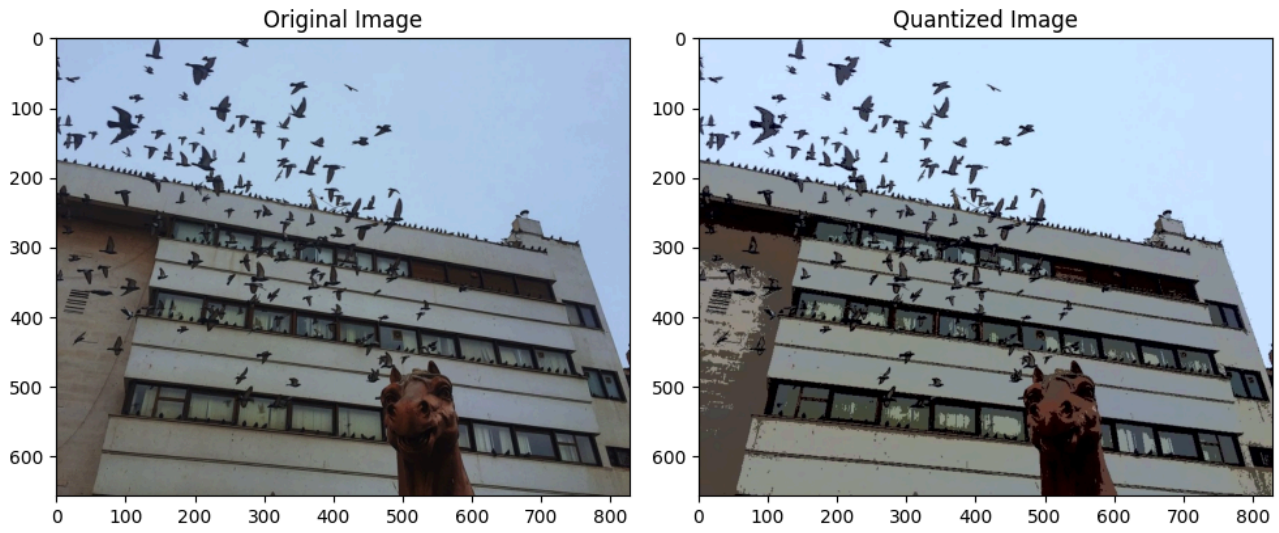


But in some bright pictures, when I increased the quantization_level in bright images, I got better results because it balanced the image.

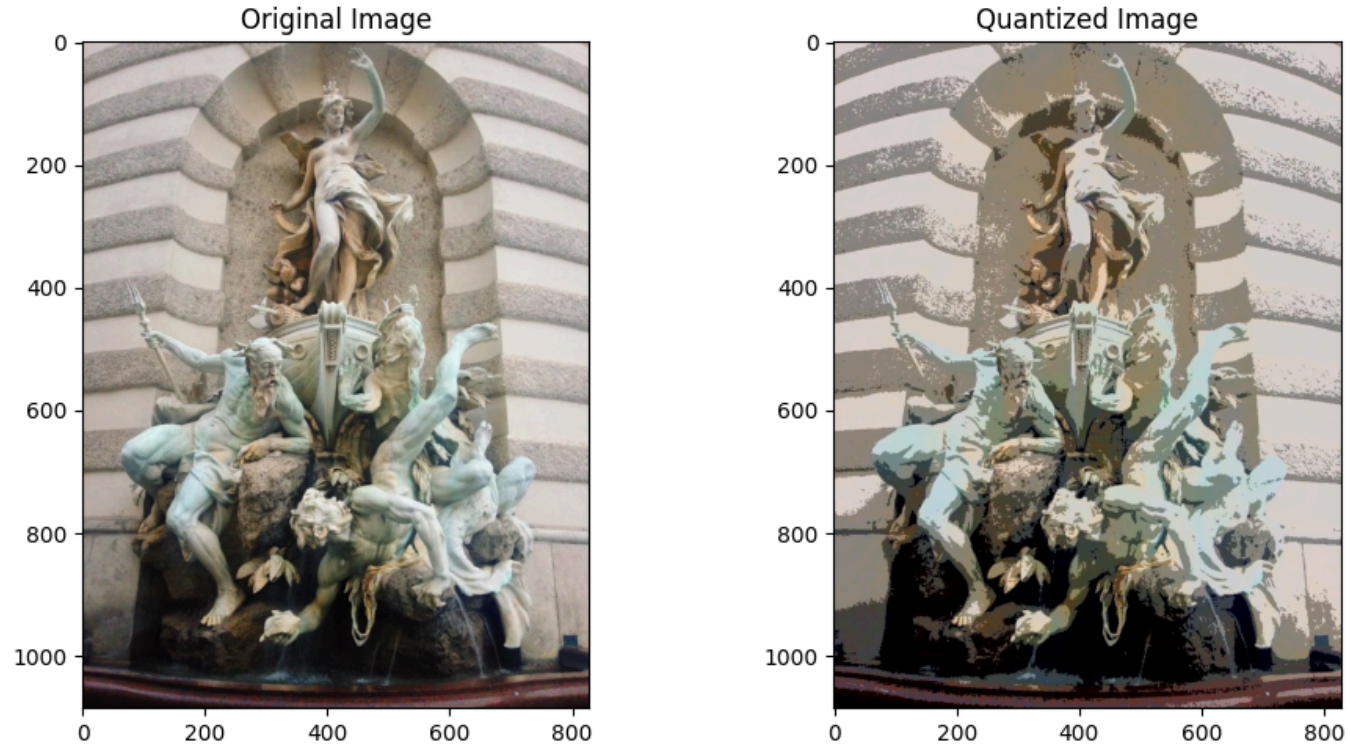




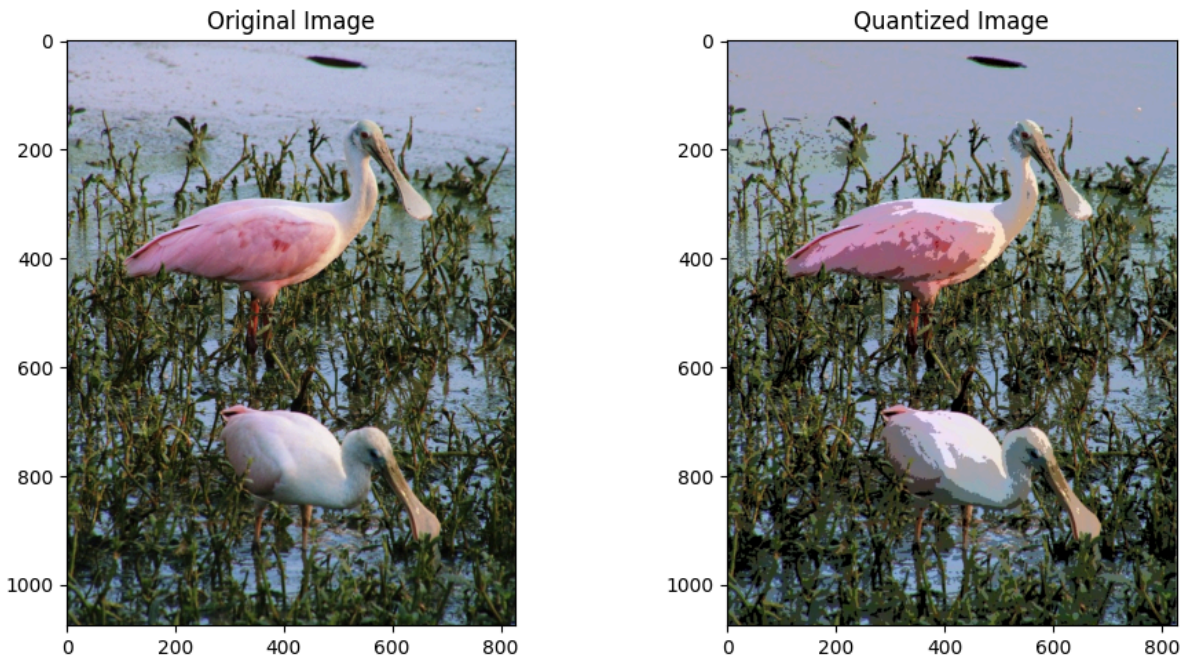
quantization_level = 2.5



quantization_level = 2.4



quantization_level = 2.5



quantization_level = 2.7

4. Combining Edge and Quantized Image

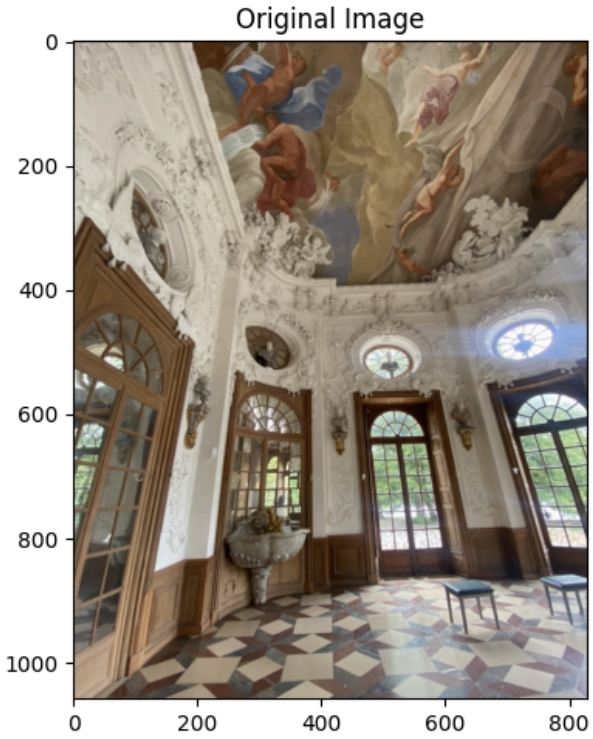
After I did edge detection and quantization, I made the combination of them.

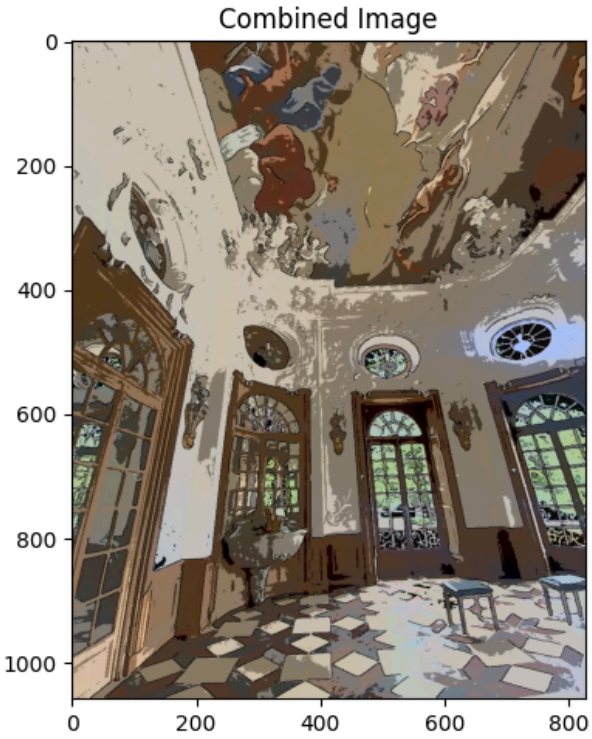
```
# Normalize the edges to a range of [0, 1]
normalized_edges = edges / 255.0

weight_quantized_image = 0.9 # adjust to balance the contribution
weight_edges = 1 - weight_quantized_image

combined_image = (weight_quantized_image * quantized_rgb_image) +
(weight_edges * normalized_edges[..., np.newaxis])

# valid range [0, 255]
combined_image = np.clip(combined_image, 0, 255).astype(np.uint8)
```

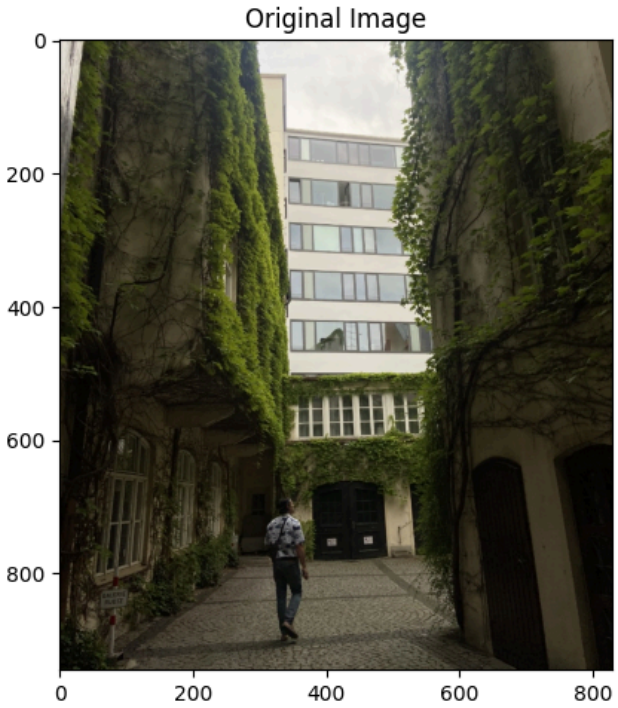
Firstly,I ensured that the pixel values in the image matrix'edges' are between 0 and 1. Because pixel values in most image processing applications typically vary from 0 to 255, dividing by 255.0 normalizes them.Then,in the weight part, the weight allocated to the quantized (stylized) image is represented by this variable. It is set to 0.9 in this case, indicating that the quantized image provides 90% of the total image.The edge weight is calculated as the complement of the weight assigned to the quantized picture.

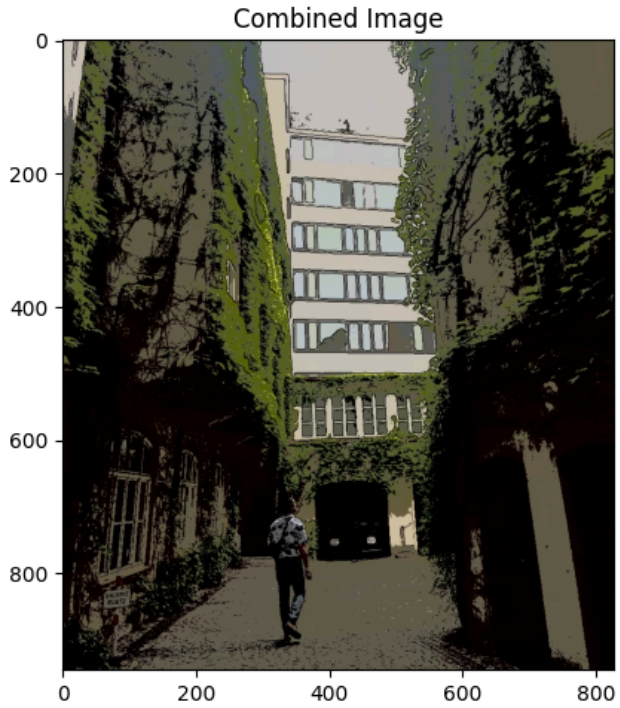
After that,I generated the combined image. The weighted sum of the quantized image and the normalized edges is used in this section. Then, each pixel of the quantized image is multiplied by the weight assigned to it, emphasizing the stylistic analysis features in the original image, and each pixel of the normalized edges is multiplied by the weight assigned to them. To match the dimensions of the quantized image for the weighted addition, np.newaxis is used to add an extra dimension to the normalized edges array.

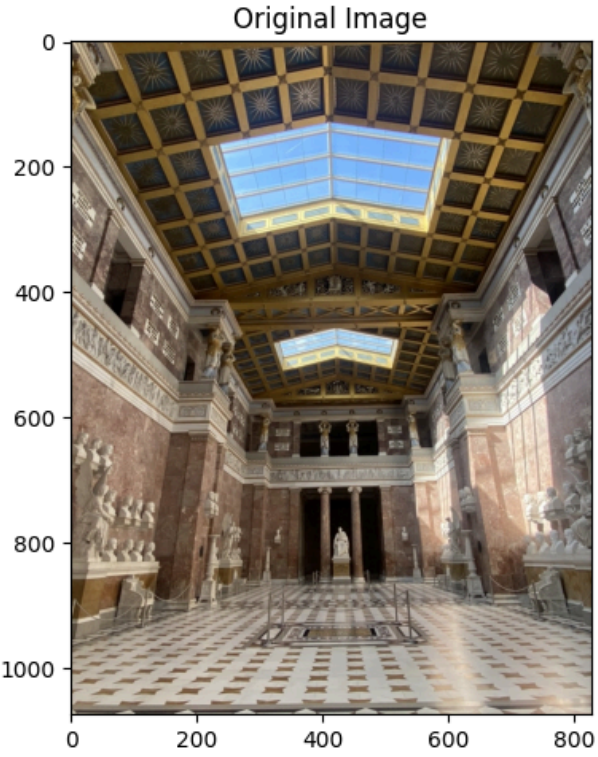
Finally, I checked that the combined image's pixel values are within the valid range of [0, 255]. Outside of this range, the np.clip function restricts values, and astype(np.uint8) converts the pixel values to an 8-bit unsigned integer format suitable for image representation.

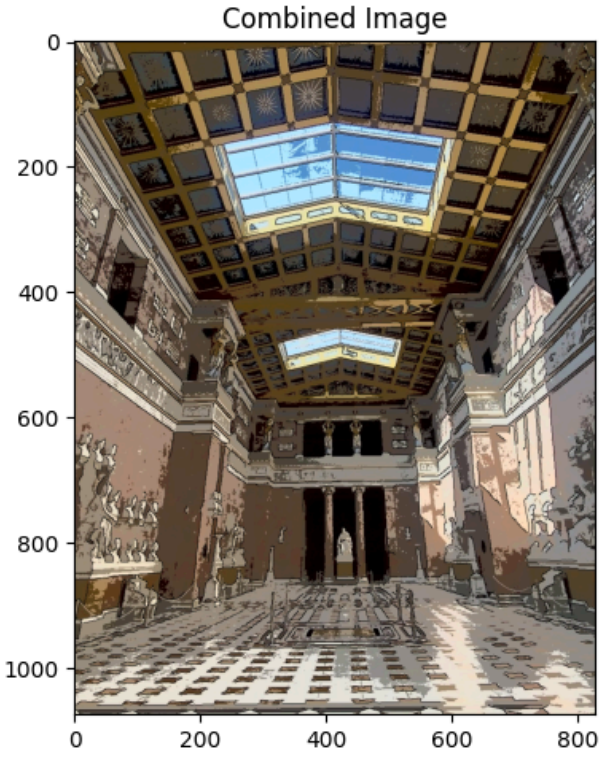


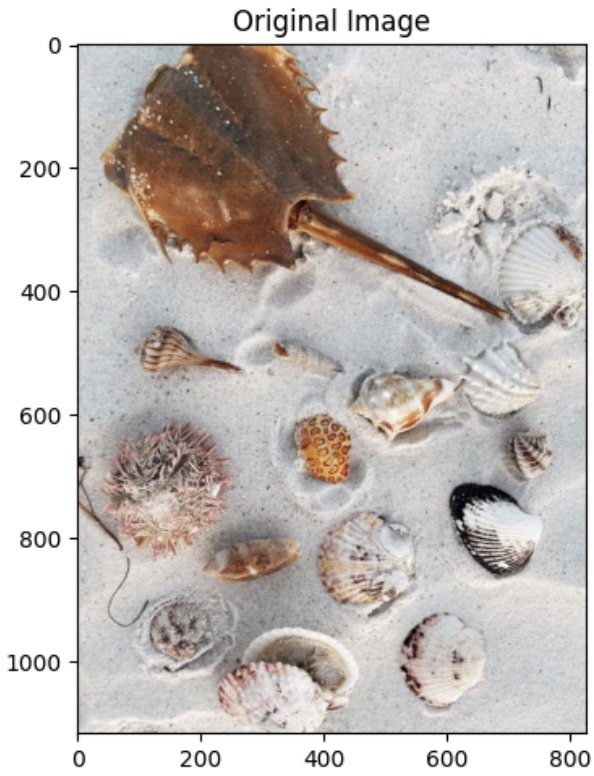


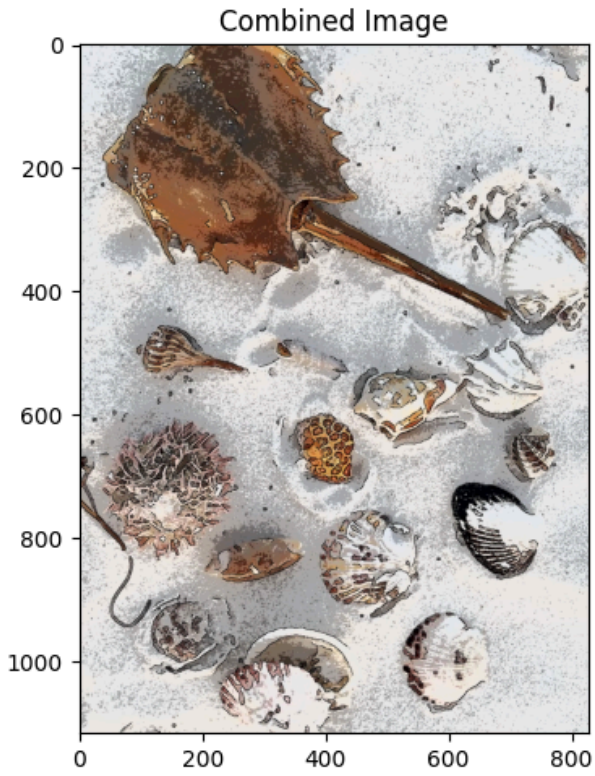

Original Image
0
200
400
600
800
0
200
400
600
800

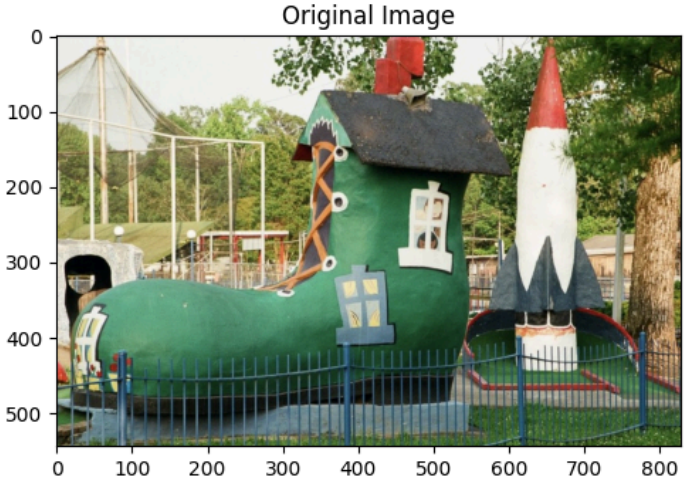

Combined Image
0
200
400
600
800
0
200
400
600
800

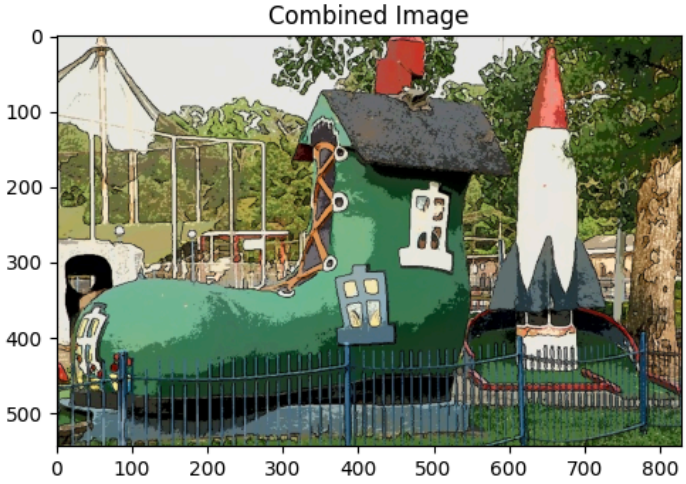

Original Image
0
200
400
600
800
1000
0
200
400
600
800

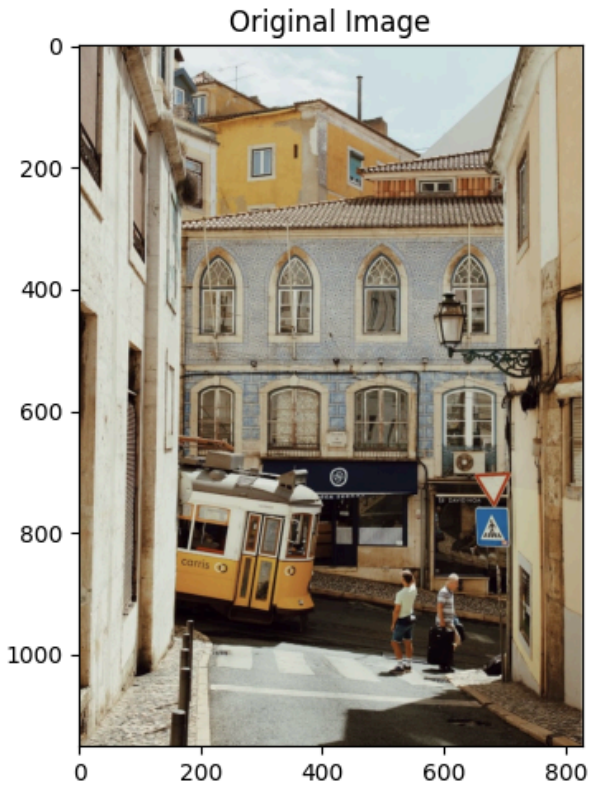

Combined Image
0
200
400
600
800
1000
0
200
400
600
800

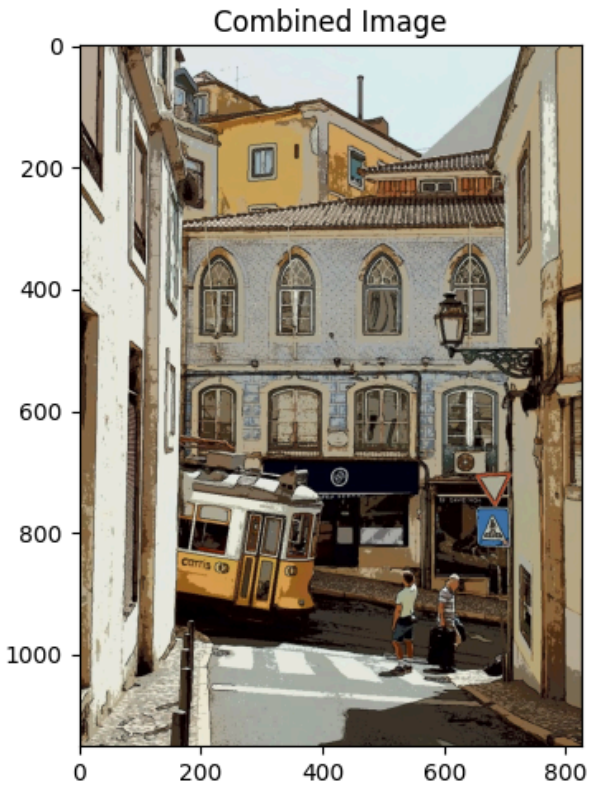

Original Image
0
200
400
600
800
1000
0
200
400
600
800

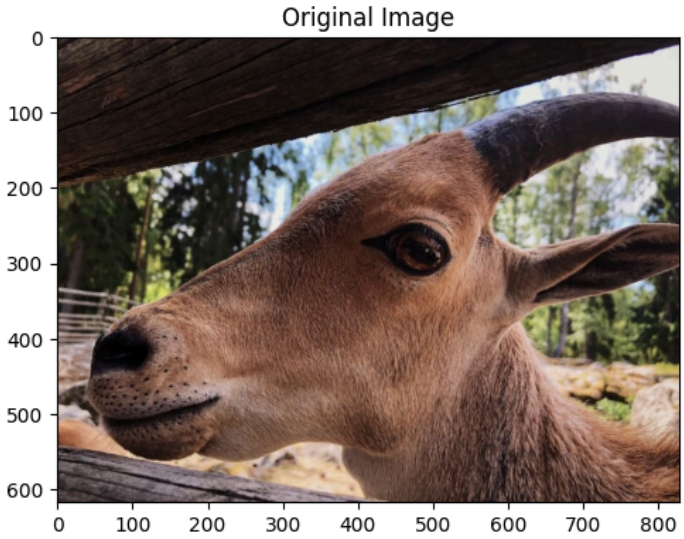

Combined Image
0
200
400
600
800
1000
0
200
400
600
800

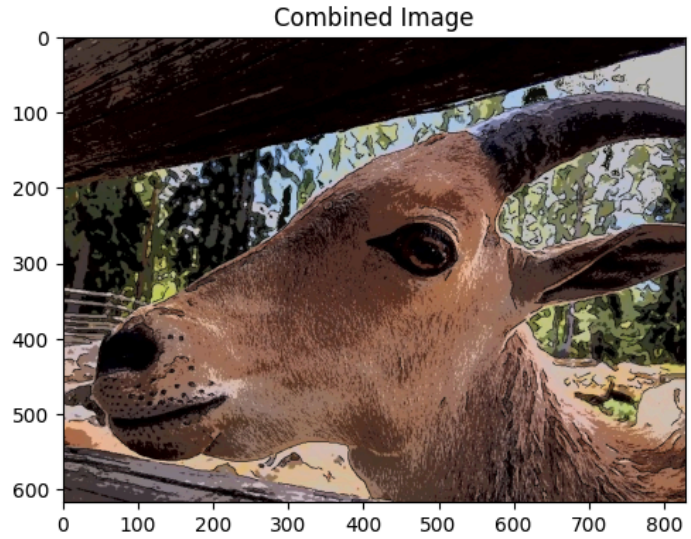

Original Image
0
100
200
300
400
500
0
100
200
300
400
500
600
700
800

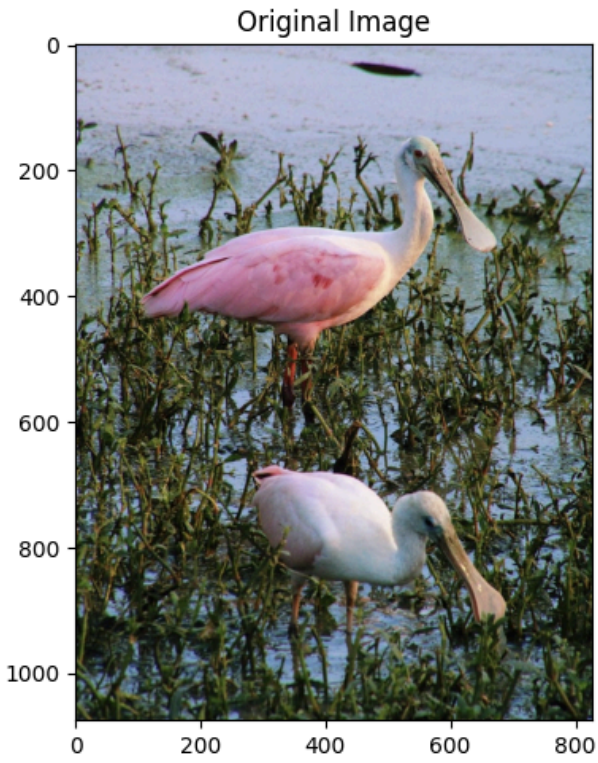

Combined Image
0
100
200
300
400
500
0
100
200
300
400
500
600
700
800

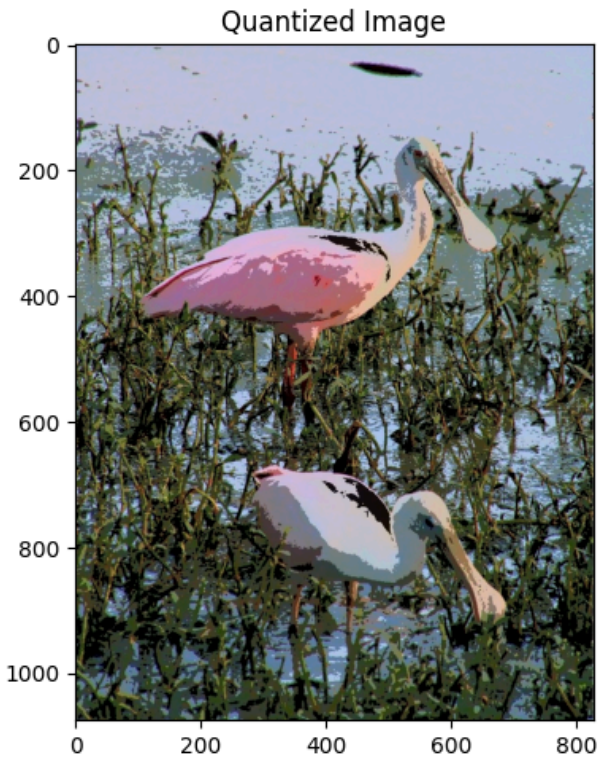

Original Image
0
200
400
600
800
1000
0
200
400
600
800

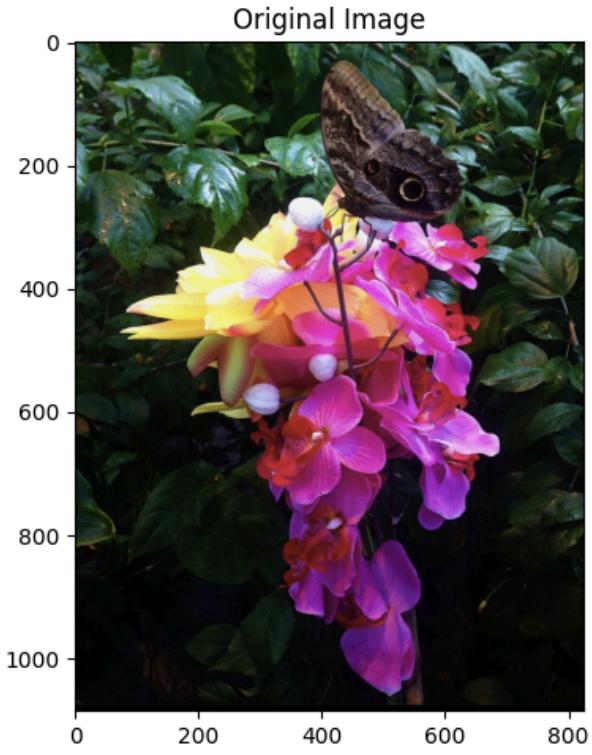

Combined Image
0
200
400
600
800
1000
0
200
400
600
800

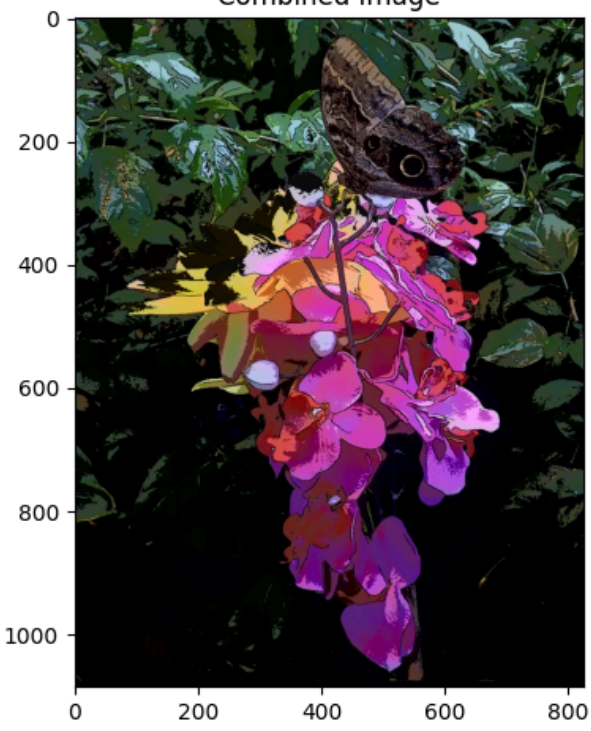

Original Image
0
100
200
300
400
500
600
0
100
200
300
400
500
600
700
800

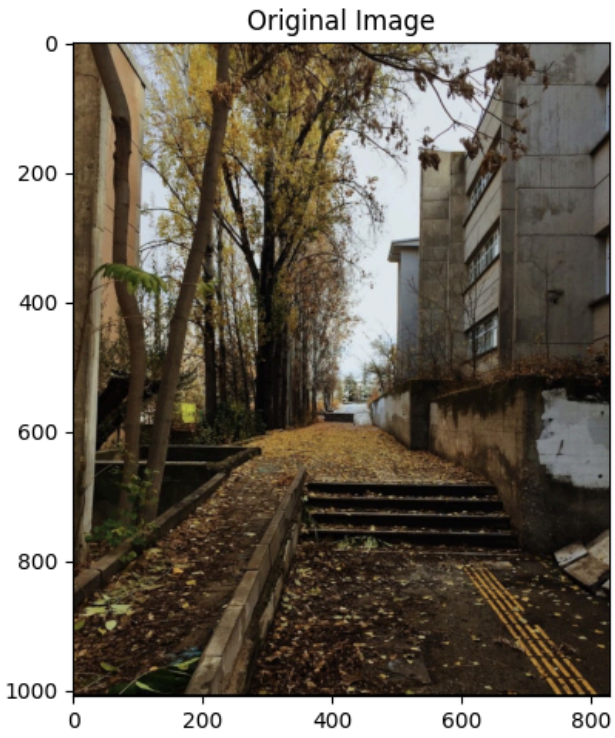

Combined Image
0
100
200
300
400
500
600
0
100
200
300
400
500
600
700
800

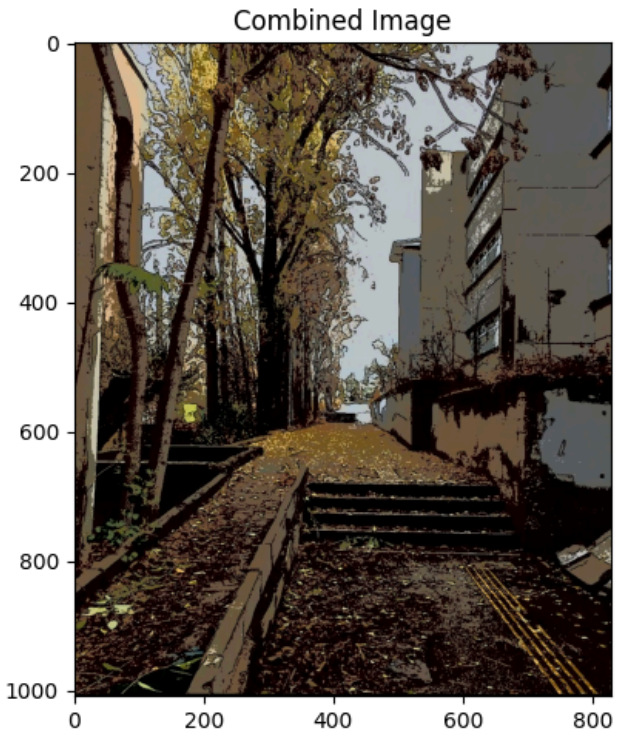

Original Image
0
200
400
600
800
1000
0
200
400
600
800

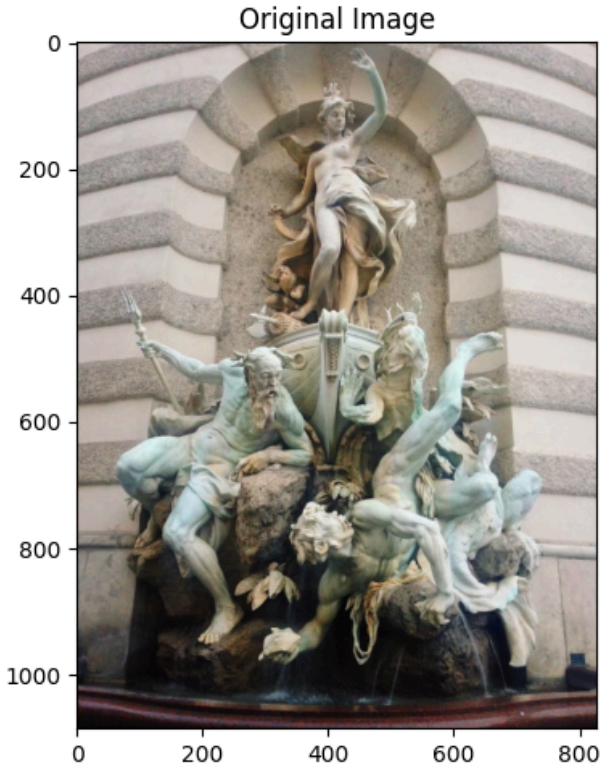

Quantized Image
0
200
400
600
800
1000
0
200
400
600
800

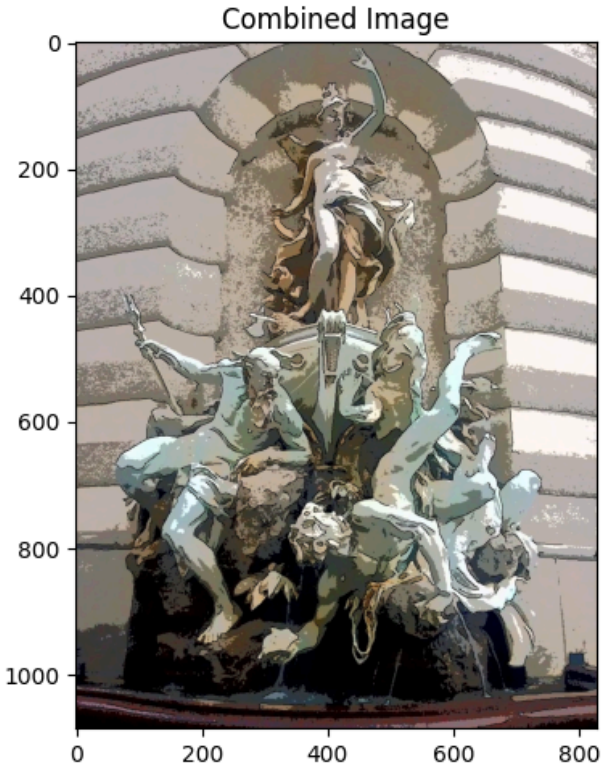

Original Image
0
200
400
600
800
1000
0
200
400
600
800

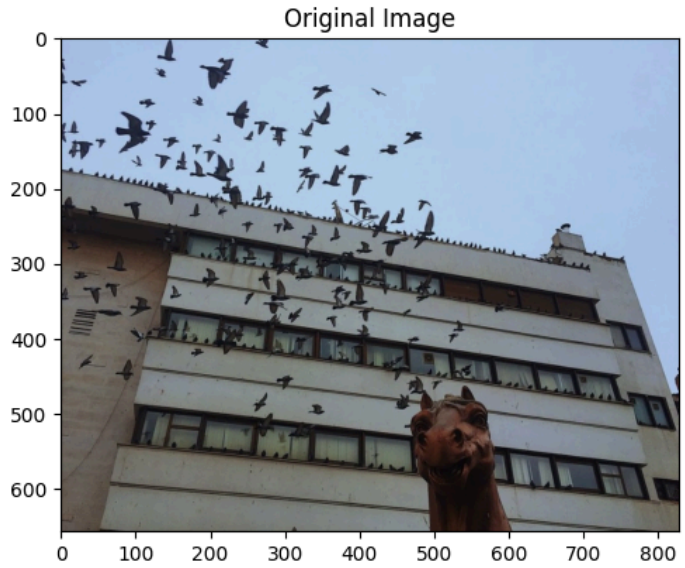

Combined Image
0
200
400
600
800
1000
0
200
400
600
800

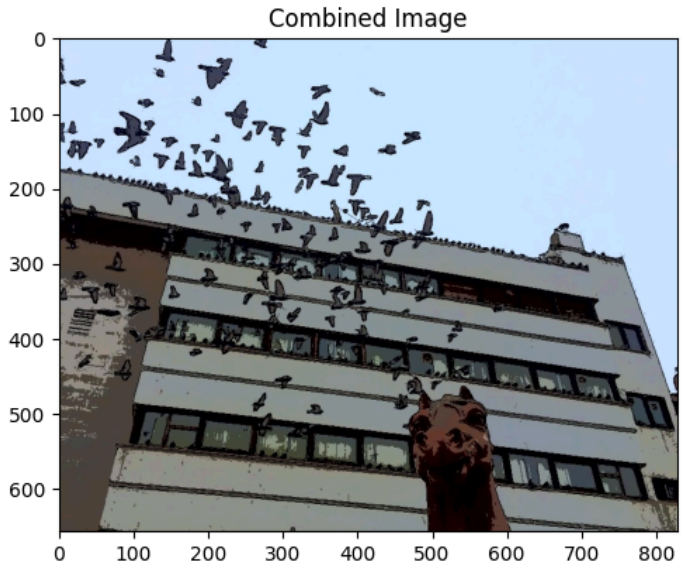

Original Image
0
200
400
600
800
1000
0
200
400
600
800

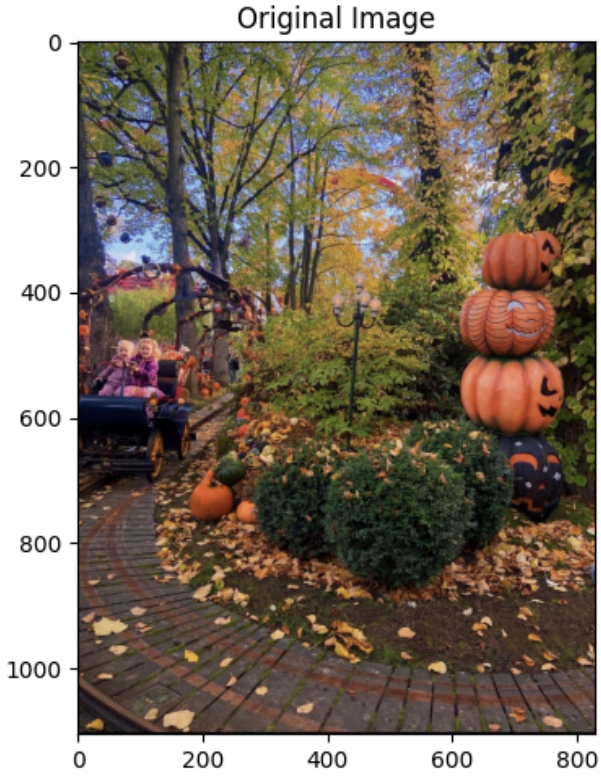

Combined Image
0
200
400
600
800
1000
0
200
400
600
800

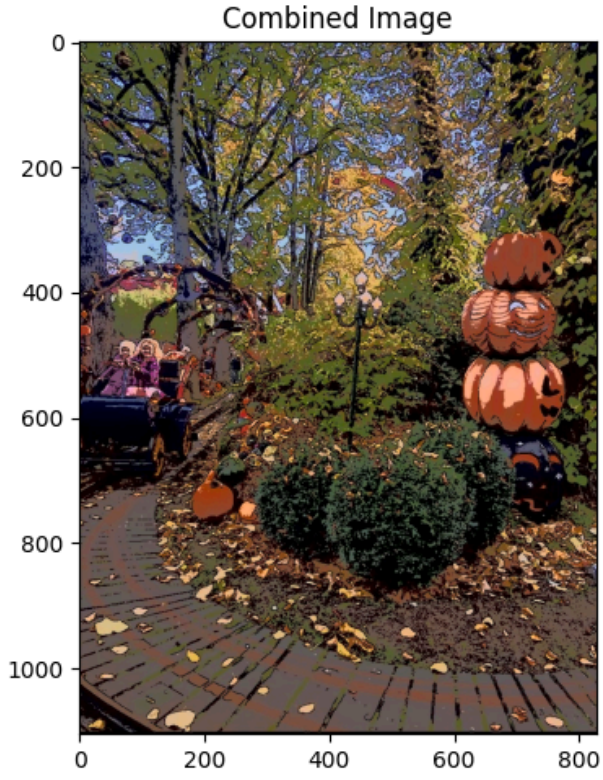

Original Image
0
200
400
600
800
1000
0
200
400
600
800


Combined Image
0
200
400
600
800
1000
0
200
400
600
800


Original Image
0
100
200
300
400
500
600
0
100
200
300
400
500
600
700
800


Combined Image
0
100
200
300
400
500
600
0
100
200
300
400
500
600
700
800


Original Image
0
200
400
600
800
1000
0
200
400
600
800


Combined Image
0
200
400
600
800
1000
0
200
400
600
800