Clab-1 Report

ENGN4528

Yixi Rao

U6826541

20/03/2022

Task-1: Matlab Warm-up.

Describe (in words where appropriate) the result/function of each of the following commands of your preferred language in report. Please utilize the inbuilt help () command if you are unfamiliar with these functions.

(1) a = np.array([[1, 2, 3],[5, 2, 20]])

This generate a (2 x 3) matrix a, the first row is [1, 2, 3] and second row is [5, 2, 20]

(2) b = a[1 , :]

The result b is the second row of matrix a, which is gained from the slice of a. b is [5, 2, 20]

(3) f = np.random.randn(200,1)

F is a (200 x 1) matrix and its value is sampled from the "standard normal" distribution.

(4) g = f[f > 0]

g is a matrix, which is obtained from matrix f by selecting the element greater than 0 in f. So, all the elements in g are greater than 0.

(5) x = np.zeros (50) + 0.5

First, x is a (1 x 50) matrix or array, which elements are all 0.5 because the np.zeros (50) will generate a matrix given a shape which element are all zero and “+” will add 0.5 to all the elements in that zero matrix

(6) y = 0.5 * np.ones([1, len(x)])

np.ones will return a new matrix with given shape (1 x 50) and type, filled with ones. And the matrix will be multiplied by 0.5. So, y is a (1 x 50) matrix, filled with 0.5.

(7) z = x + y

Z is obtained by adding x and y element-wise, x shape is (1 x 50) y is (1 x 50), so z will be (1 x 50), all the elements are 1.

(8) a = np.linspace(1,200)

np.linspace will return evenly spaced numbers over a specified interval. So, ‘a’

is a 50 evenly spaced samples, calculated over the interval [1, 200].

(9) b = a[: :-1]

Using a slicing operation on ‘a’ to get a reverse array of ‘a’ and assign it to the variable ‘b’.

(10) b[b <35]=0

First, finding all the elements in b that are less than 35, and reassign its value to 0, now b is a matrix which elements less than 35 will be 0.

Task-2: Basic Image I/O (2 marks)

2.a Read this image from its JPG file, and resize the image to 384 x 256 in columns x rows (0.2 marks).

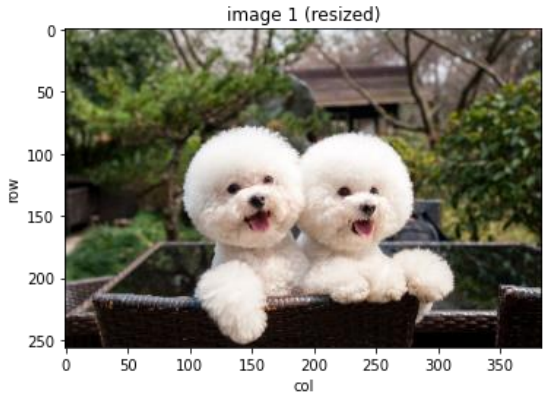

Figure 2.a.1

Using the cv2.resize() to resize the image. The figure 2.a.1 shows the result of the resized image

2.b Convert the colour image into three grayscale channels, i.e., R,G,B images, and display each of the three grayscale images separately

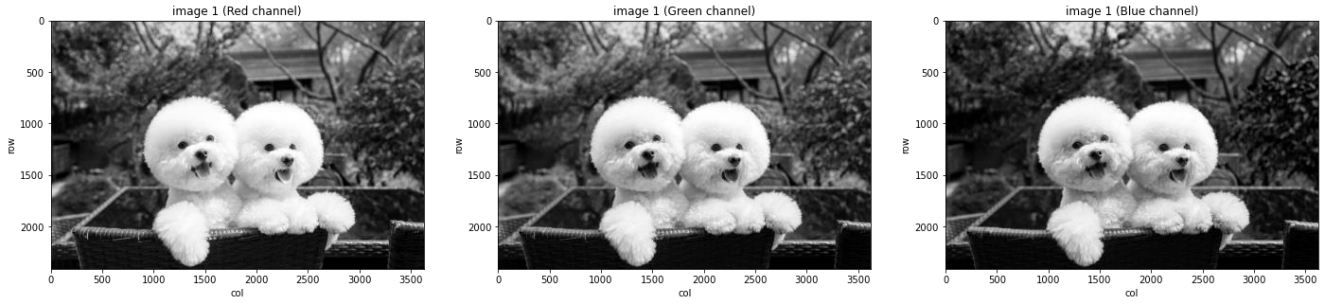

figure 2.b.1

The figure 2.b.1 shows the three grayscale channel images. The images are red channel grayscale image, green channel grayscale image, and blue channel grayscale image from the left to right. These three images are gained by using the cv2.split function.

2.c Compute the histograms for each of the grayscale images, and display the 3 histograms

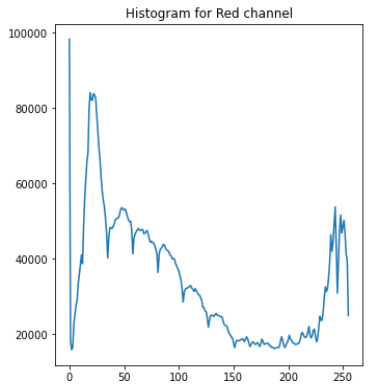

figure 2.a.1

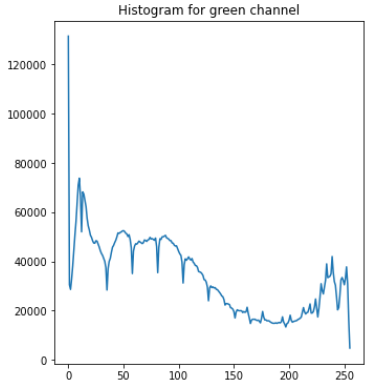

figure 2.c.2

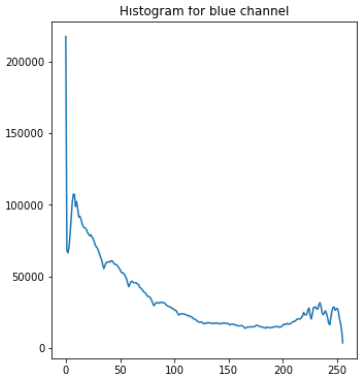

figure 2.c.3

The figure 2.c.1, figure 2.c.2 and figure 2.c.3 show the histogram of the three grayscale channel images. The order is red, green, blue.

2.d Apply histogram equalisation to the resized image and its three grayscale channels, and then display the 4 histogram equalization image

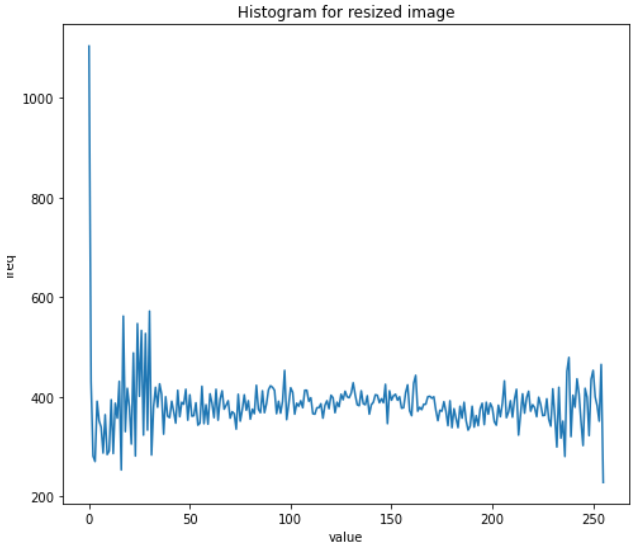

figure 2.d.1

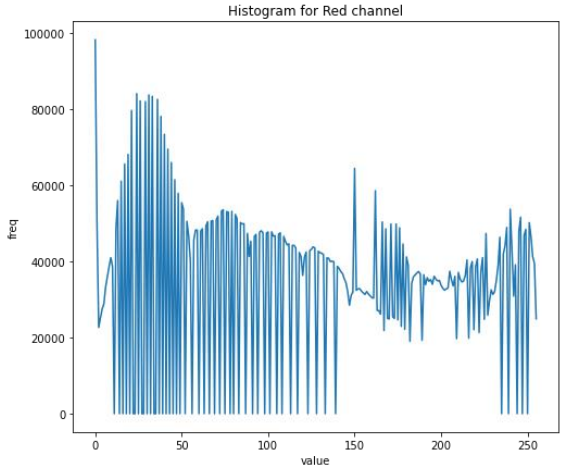

figure 2.d.2

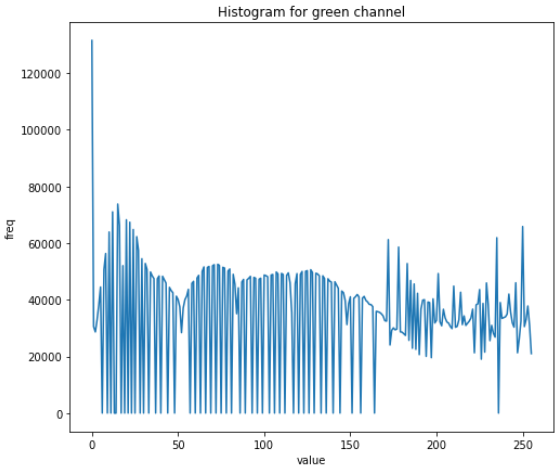

figure 2.d.3

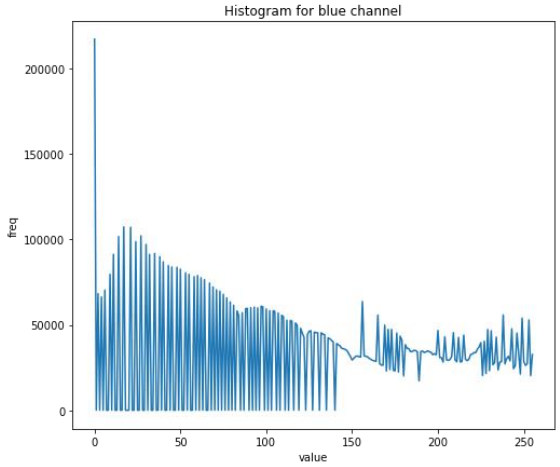

figure 2.d.4

The figure 2.d.1 shows the histogram of the resized image after applying the histogram equalisation. The figure 2.d.2, figure 2.d.3 and figure 2.d.4 show the histogram of three grayscale channels after applying the histogram equalisation. Comparing with the histograms in 2.c, we can observe that the intensities are now distributed better on the histogram utilizing the full range of intensities evenly.

Task-3: Colour space conversion

1. Based on the formulation of RGB-to-HSV conversion, write your own function cvRGB2HSV() that converts the RGB image to HSV colour space (1.4 marks). Read in Fig. 2(a) and convert it with your function, and then display the H, S, V channels in your report

I write two functions to do this task, first function is cv_PixelRGB2HSV, which input is a pixel contained the R, G, B value of it. Then, I follow the conversion function shown in lecture slides (displayed in figure 3.1.1) to implement the RGB to HSV conversion, note that all the R, G, B value are normalized by dividing 255. The second function is the cv_RGB2HSV which takes an image as input, it will return a

HSV image by applying cv_PixelRGB2HSV function to all the pixels in the input's image. This process is done by using the double 'for loop' to iterate through all rows and columns' pixels. Then, I use the cv2.split to separate the H, S, V value. The H, S, V channel's histograms are displayed in figure 3.1.2, figure 3.1.3 and figure 3.1.4.

$$X_{max} := \max(R, G, B) =: V$$

$$X_{min} := \min(R, G, B)$$

$$C := X_{max} - X_{min}$$

$$H := \begin{cases} 0, & \text{if } C = 0 \\ 60^\circ \cdot \left(0 + \frac{G-B}{C}\right), & \text{if } V = R \\ 60^\circ \cdot \left(2 + \frac{B-R}{C}\right), & \text{if } V = G \\ 60^\circ \cdot \left(4 + \frac{R-G}{C}\right), & \text{if } V = B \end{cases}$$

$$S_V := \begin{cases} 0, & \text{if } V = 0 \\ \frac{C}{V}, & \text{otherwise} \end{cases}$$

figure 3.1.1

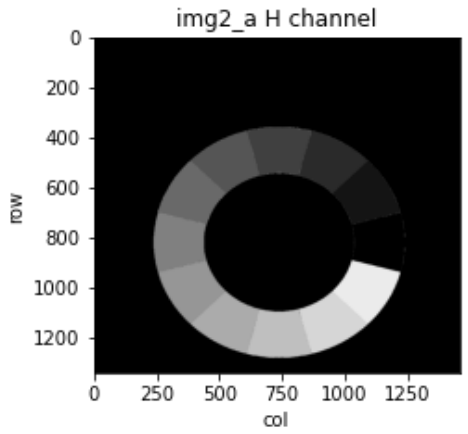

figure 3.1.2

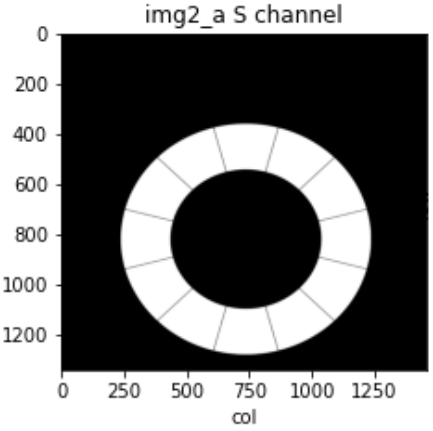

figure 3.1.3

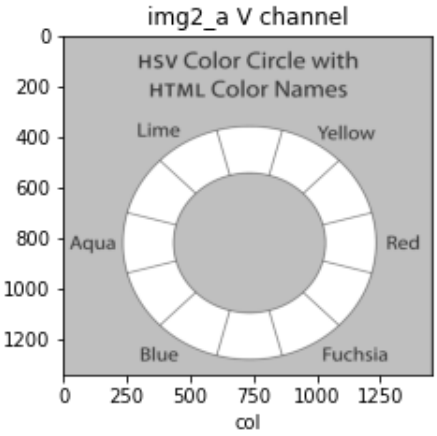

figure 3.1.4

2. Compute the average H values of five colour regions in Fig. 2(b) with your function and Python's inbuilt function rgb2hsv() . Print both of them under the corresponding regions. You also need to explain how to distinguish and divide the five regions, and how to calculate the average Hue value

We know that the fig (2b) contains five color regions but we do not know the exact edge of each color region. By further observing the image, I notice that all the regions are vertically distributed in the image, so I know that the edges will be some vertical bars. Now, the task is to find the edges, we know that different regions will have some different Hues, saturations, and values, so there must have some HSV value changes between two regions. We utilize this fact to calculate each pixel's HSV value difference diff(x, y) between the pixel (x, y) and (x – 1, y), and if this value is greater

than 0, then we know an edge is detected. Besides, we only need to apply the diff(x, y) in the first row of the image since regions are separated vertically. After applying the diff(x, y) to the first row of the image, we receive a batch of points which value is greater than 0, then, we need to divide the region. This is done by clustering all the points that are very close to each other and then compute the average point of it, the result is the edge we need. To compute the average Hue value of different regions, we use the four indices (edges) to slice the image to get different regions of the image, and then compute the average Hue value of each region by using np.mean().

The figure 3.2.1 shows the result of the average H values of five color regions using my function, and the figure 3.2.2 shows the result of the average H values of five color regions using python's inbuilt function.

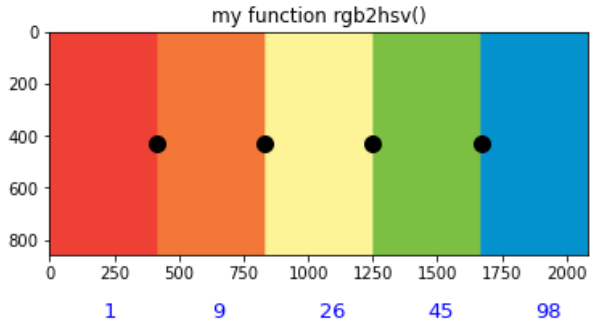

figure 3.2.1

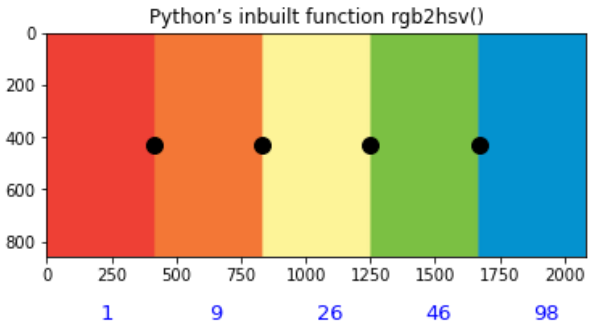

figure 3.2.2

The black dots in the figure 3.2.1 and figure 3.2.2 represent the edge (index).

Task-4: Image Denoising via a Gaussian Filter

1.Read in image2.jpg. Crop a square image region corresponding to the central facial part of the image, resize it to 512×512, and save this square region to a new grayscale image. Please display the two images. Make sure the pixel value range of this new image is within [0, 255]

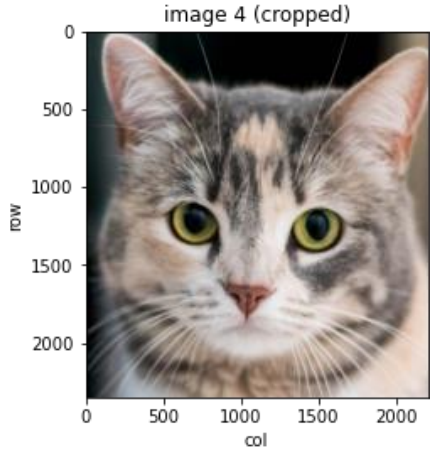

figure 4.1.1

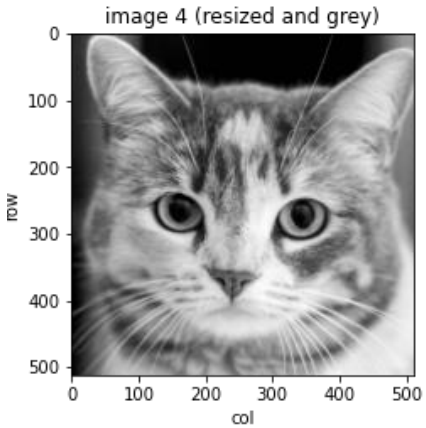

figure 4.1.2

figure 4.1.1 shows the cropped image, and figure 4.1.2 shows the greyscale resized image.

2. Add Gaussian noise to this new 512x512 image (Review how you generate random number in Task-1). Use Gaussian noise with zero mean, and standard deviation of 15

I use the np.random.normal to generate the gaussian noise, and use the clip(0, 255) function to ensure the range is in (0, 255). The figure 4.2.1 shows the result image with noise and its original image.

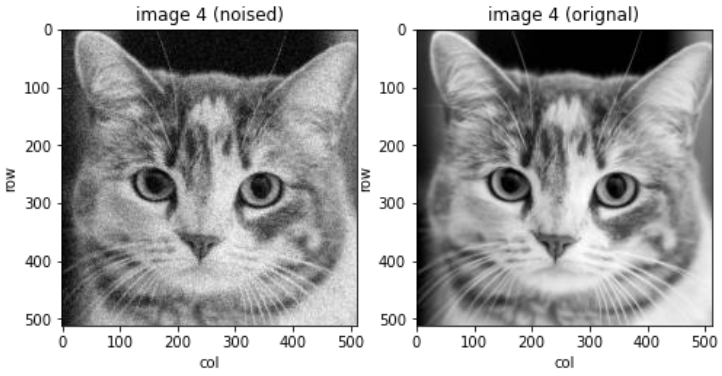

figure 4.2.1

3. Display the two histograms side by side, one before adding the noise and one after adding the noise

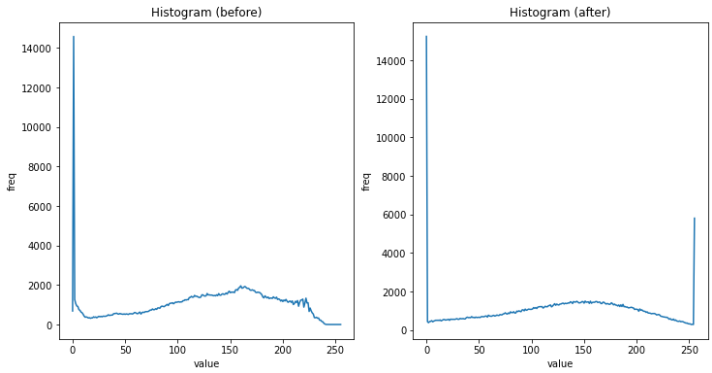

figure 4.3.1

The figure 4.3.1 shows the histogram before adding the noise (left) and the histogram after adding the noise.

4. Implement your own Matlab function that performs a 7x7 Gaussian filtering

In order to implement the my_Gauss_filter() function, we need to first define the kernel. The kernel is created by using the create_kernel(σ, k_size) function, this function will create the Gaussian filter's kernel using the input σ and k_size, it will return a (k_size x k_size) matrix, which values are created by using Gaussian

function. Now using the kernel and the image, we can do the convolution to get the output image.

5. Apply your Gaussian filter to the above noisy image, and display the smoothed images and visually check their noise-removal effects. One of the key parameters to choose for the task of image filtering is the standard deviation of your Gaussian filter. You may need to test and compare different Gaussian kernels with different standard deviations

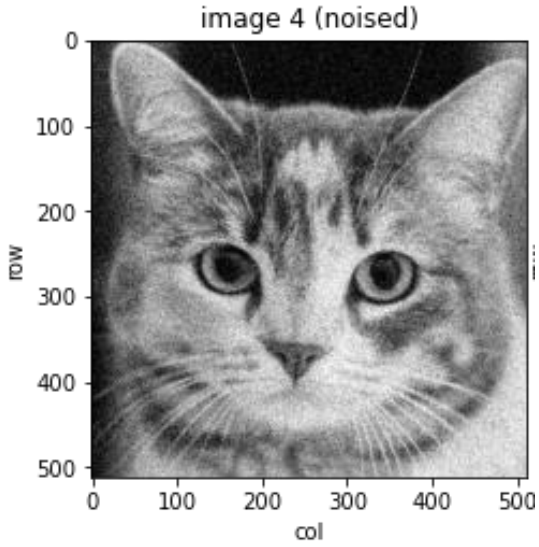

figure 4.5.1

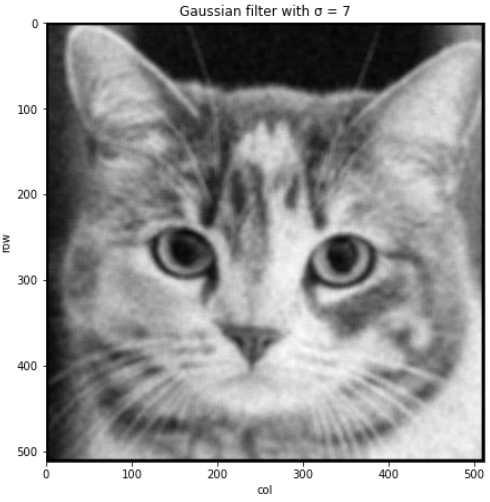

figure 4.5.2

Figure 4.5.1 shows the original image with some Gaussian noise, figure 4.5.2 shows the image after applying the Gaussian filtering, we can clearly see that many noises are removed, and figure 4.5.2 obviously smoother and more blurry than the figure

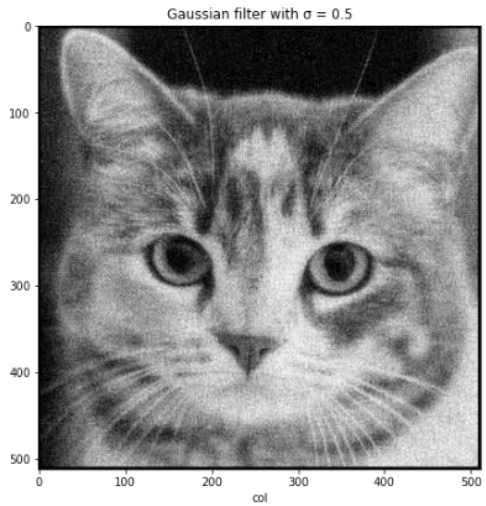

figure 4.5.3

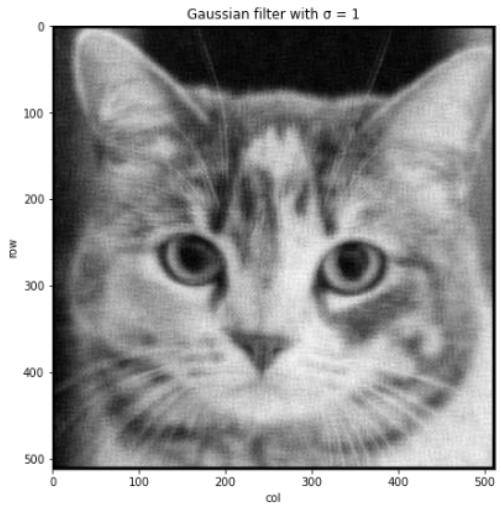

figure 4.5.4

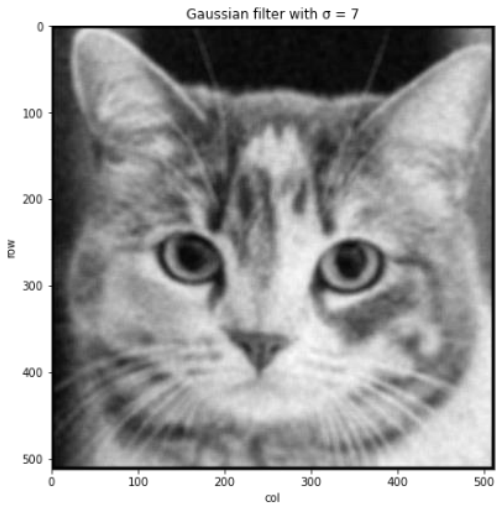

figure 4.5.5

4.5.1.

Figure 4.5.3, Figure 4.5.4 and Figure 4.5.5 show the investigation of how the different

Gaussian kernels with different standard deviations influence the image. The figure 4.5.3 uses sigma = 0.5, the figure 4.5.4 uses sigma = 1 and figure 4.5.5 uses sigma = 7. From the left to right of the figures, we can observe that the images are gradually blurrier and the noise are gradually removed. So, we can conclude that adjusting sigma is really adjusting how much the surrounding pixels affect the current pixel and increasing sigma increases the influence of the distant pixels on the center pixel, and the filtering result is smoother and blurrier.

6. Compare your result with that by Matlab's inbuilt 7x7 Gaussian filter

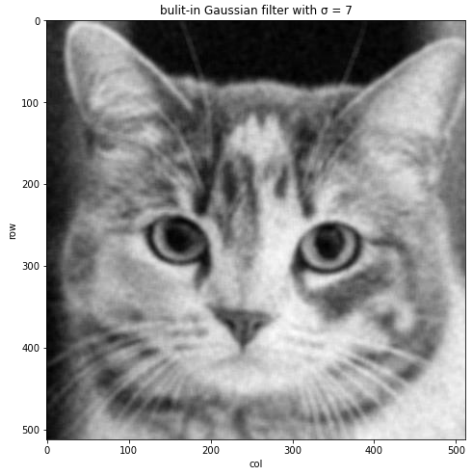

figure 4.6.1

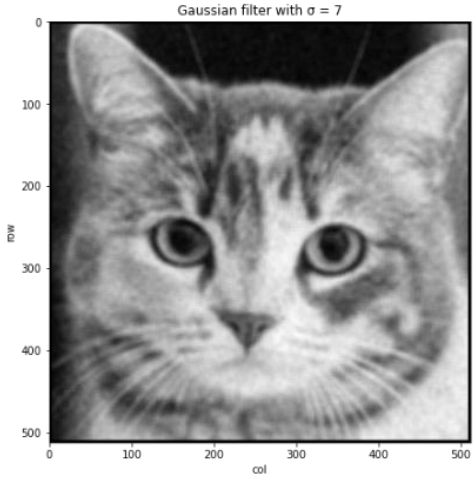

figure 4.6.2

The figure 4.6.1 is the image applying the cv2 inbulid Gaussian filter with sigma = 7 and the figure 4.6.2 is the image applying my Gaussian filter with sigma = 7. And we can observe that there is nearly no difference and the two results are nearly identical except for I do not deal with the blank value on the border of the image.

Task -5: Implement your own 3x3 Sobel filter in Matlab/Python

1.<u>Test it on sample images</u> (1.5 marks), and <u>compare your result with the inbuilt Sobel edge detection function</u> (0.5 marks). Briefly explain the function of the Sobel filter and how it can achieve this function (1.0 mark).

In order to implement the Sobel edge detector, I first define sobel_filter(img) function which takes the original image as input and output the image derivative. In this function, it first defines the Sobel operator or kernel, for example Mx = $\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2, \\ -1 & 0 & 1 \end{matrix}$ and My = $\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$, we now using these two kernels for convolution on the image, which means we take the derivative in both directions. For example, convolving image I with the kernel Mx to gain Gx and convolving image I with the

kernel My to gain Gy. And for each pixel of the image, the approximate gradient is calculated by combining the above two results using the following formular $G = \sqrt{(Gx^2 + Gy^2)}$. The result for image 2 is displayed in figure 5.1.1 and result for

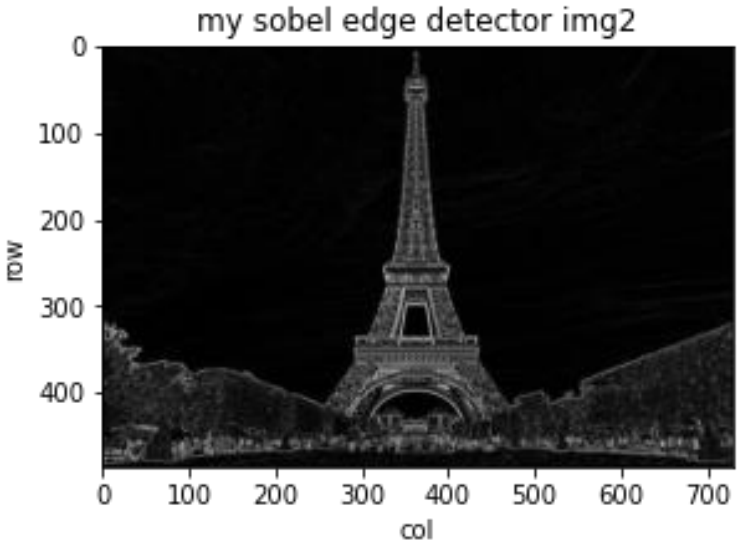

figure 5.1.1

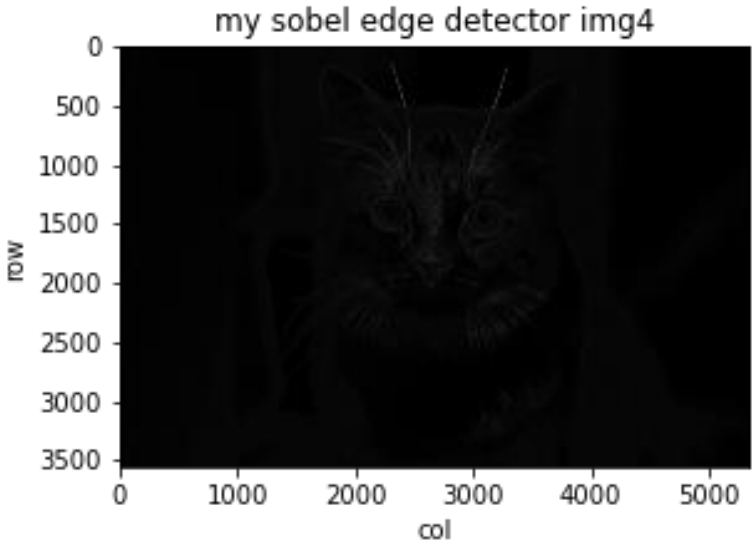

figure 5.1.2

image 4 is displayed in figure 5.1.2.

The figure (figure 5.1.3 and figure 5.1.4) below shows that the image after using the inbuilt Sobel edge detection

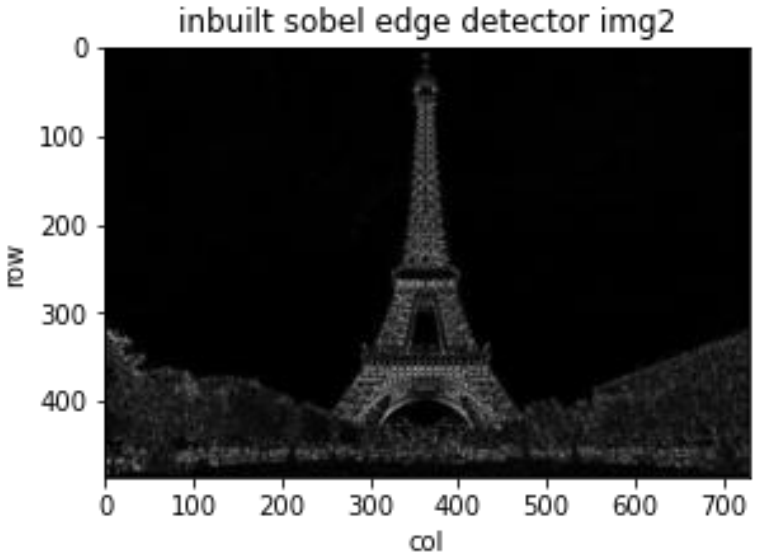

figure 5.1.3

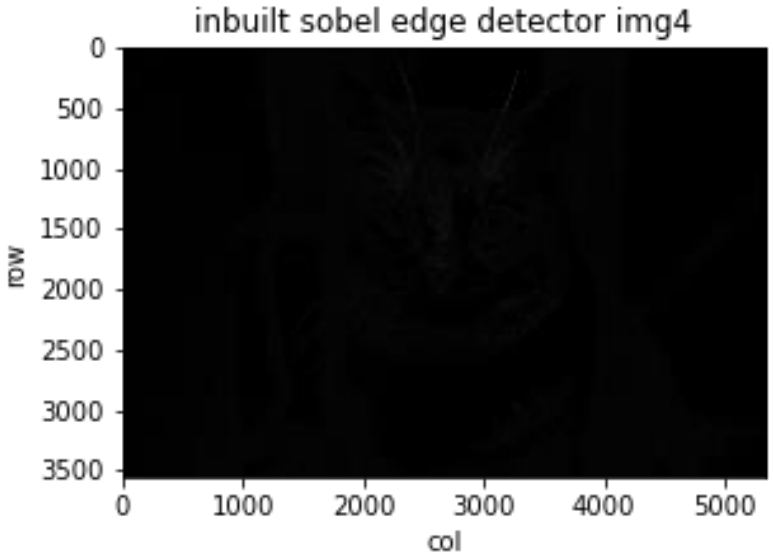

figure 5.1.4

We can see that both my function and the inbuilt function can identify the edges of the two images.

The function of the Sobel filter is to detect the edge and calculates the first image derivatives, so the function of Sobel operator combines Gaussian smoothing and differential derivation. This is achieved by using the two kernels that have already

mentioned above. Essentially, the function of the kernel is trying to find out the amount of difference between left (upper) region of the image and right (bottom) region of the image, if the value is very big, we know that an edge is detected.