Clab-1 Report
ENGN6528

Wangshu Cai
U7546753

13/03/2023

Task-1: Matlab (Python) Warm-up. (2 marks):

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.

Note: Different from Matlab, Python users need to import external libraries by themselves. And we assume you already know some common package abbreviations (e.g. numpy = np). (0.2 marks each)

Python

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

Create a variable a and initialize it as a 2-D array with 2 rows and 3 columns. The values of the first row are 1, 2, and 3. The values of the second row are 5, 2, 20.

(2) b = a[0, :]

Assign new variable b to be the first row of a, which is now an array of [1, 2, 3].

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

Initialize a random array with 200 rows and 1 column. The values are all generated (pseudo) randomly in normal distribution with mean 0 and variance 1.

(4) g = f[f > 0]

Create a new variable g with a new 1-D array with all elements of f that are greater than 0.

(5) x = np.zeros (20) + 0.5

Create a new variable x, then create an 1-D array (20 elements) with all values to be 0. After that, adding 0.5 to the array. According to the broadcast property, all of the entries of the array should be 0 + 0.5 = 0.5. Finally assign the result array to x.

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

Create a new variable y, then create an 1-D array (20 elements) with all values to be 1. After that, multiply the array by 0.5. According to the broadcast property, all of the entries of the array should be 1 * 0.5 = 0.5. Finally assign the result to y.

(7) z = x + y

Create a new variable z, then add x and y and assign the result to z. Since x and y are of the same shape (20,), we can perform vector add. The result should be a 1-D array with 20 columns and all values to be 1.

(8) a = np.linspace(1,100)

Create a new 1-D array (50 elements) starting with 1 and ending with 50. Other numbers are evenly spaced samples between the first and the last elements. The array is assigned to a.

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

The reversed version of array in a is assigned to b.

(10) b[b <25]=0
assign all of the elements less than 25 to be 0. It can also affect the same values in a.

Hint:

Do the necessary typecasting (uint8 and double) when processing and displaying the image data in the following tasks. For Python, please be aware of the default datatype of different libraries (e.g. Image, matplotlib, cv2). An improper datatype of image will cause many troubles when you want to display the image.

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

(a) (b) (c)

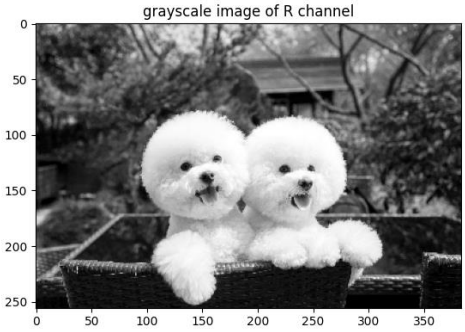
Figure 1

In this task, you are asked to:
Given the three images shown in Figure 1(you can find them in the CLAB folder on Wattle):
1. Read those images, and save them to JPG image files named ‘image1.jpg’, ‘image2.jpg’ and ‘image3.jpg’. (0 marks).
2. Using image1.jpg, develop short computer code that does the following tasks:

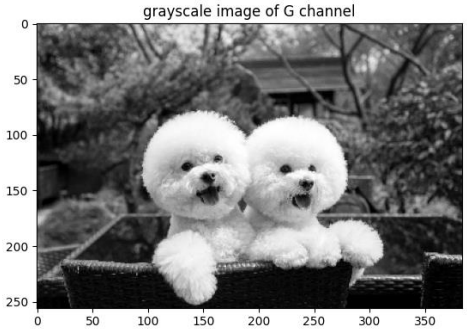
a. Read this image from its JPG file, and resize the image to 384 x 256 in columns x rows (0.2 marks).

```
# %% a
# resize the image into 384x256
img1_resized = cv2.resize(img1, (384, 256))
# Show image from bgr to rgb
plt.imshow(img1_resized[:,:,::-1])
plt.show()
```

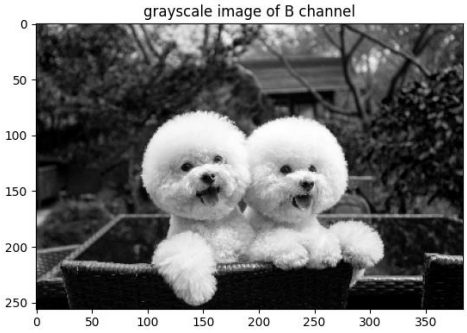
b. Convert the colour image into three grayscale channels, i.e., R-channel, G-channel, B-channel images, and display each of the three grayscale images separately (0.2 marks for each channel, 0.6 marks in total).



(a)



(b)

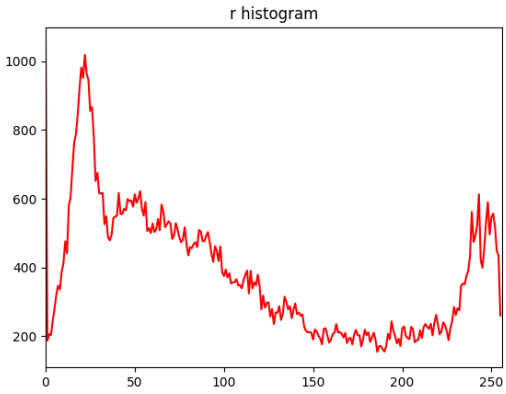


(c)

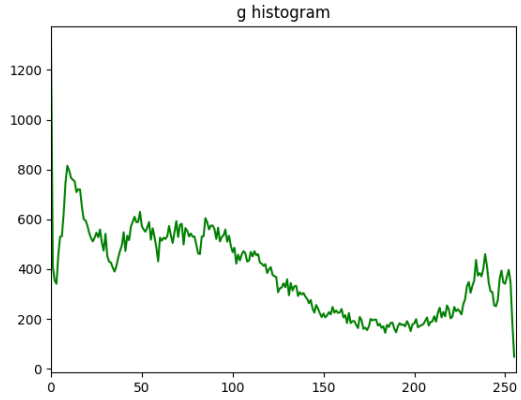
Fig 2 Illustrates the image of grayscale images for all of the 3 channels of image 1. (a): grayscale image of R channel. (b): grayscale image of G channel. (c): grayscale image of B channel

Reading from each channel is achieved by accessing the image array with specific 3rd dimension index. 0 for blue channel, 1 for green channel, and 2 for red channel.

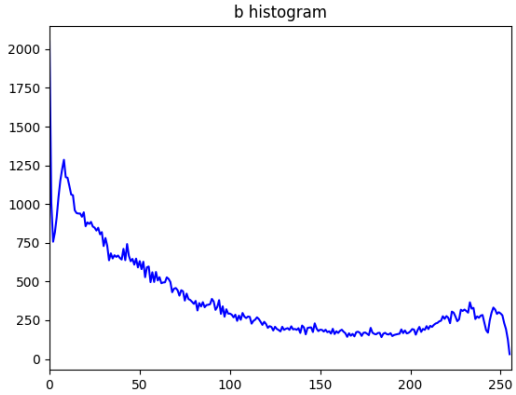
c. Compute the histograms for each of the grayscale images, and display the 3 histograms (0.2 marks for each histogram, 0.6 marks in total).



(a)



(b)



(c)

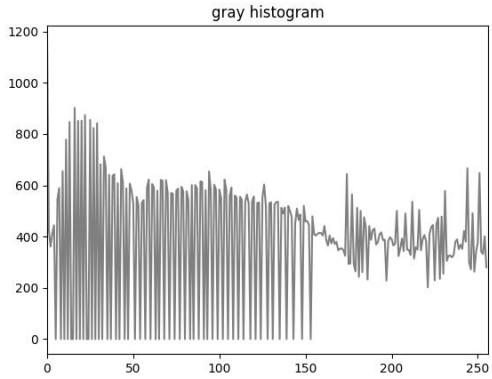
Fig 3 illustrates the histogram of 3 channels of image 1. (a): histogram of R channel. (b): histogram of G channel. (c): histogram of B channel

The method to create histogram is provided in Python HelperSheet.pdf. We can observe that the distributions of the histograms of different channels are different.

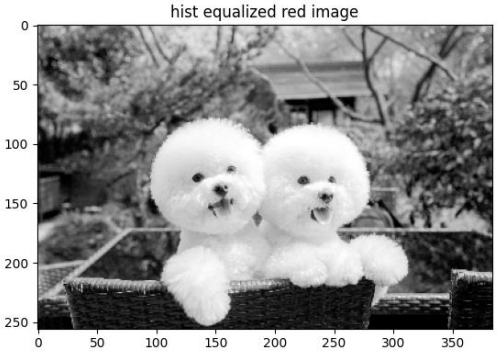
d. Convert the colour image to a grayscale image and then apply the histogram equalisation to this grayscale image, and the R-channel, G-Channel, and B-Channel images (mentioned in b.), respectively. Then display the 4 images after applying the histogram equalisation process, and their corresponding histograms. (0.15 marks for each (image, histogram pair), 0.6 marks in total). (Hint: you can use inbuilt functions for implementing histogram equalisation. e.g. histeq() in Matlab or cv2.equalizeHist() in Python).


hist equalized grayscale image

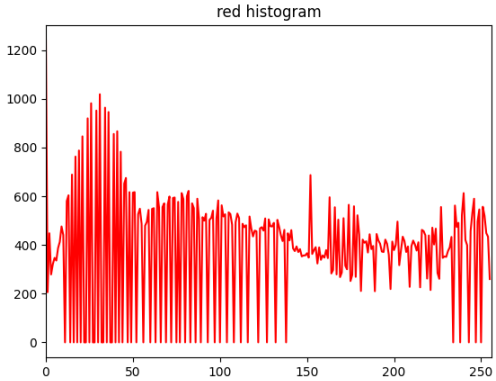
(a)


gray histogram

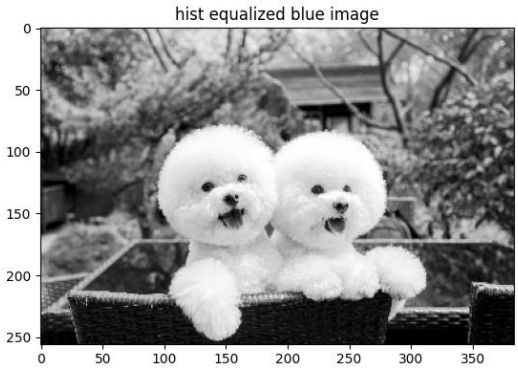
(b)


hist equalized red image

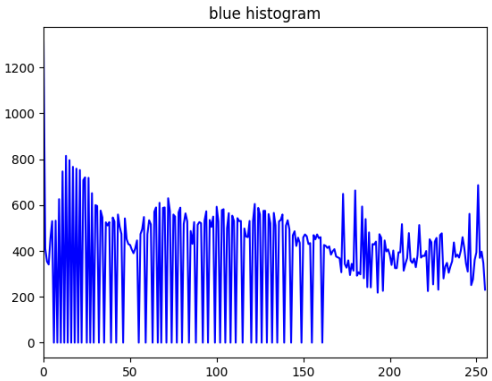
(c)


red histogram

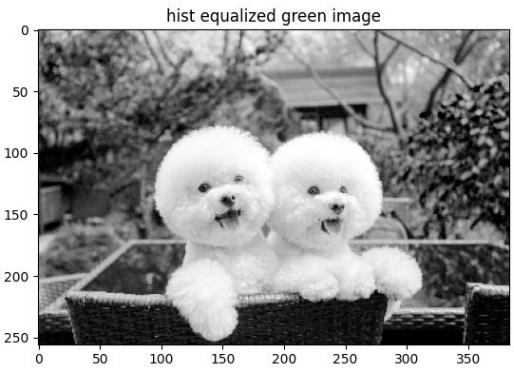
(d)

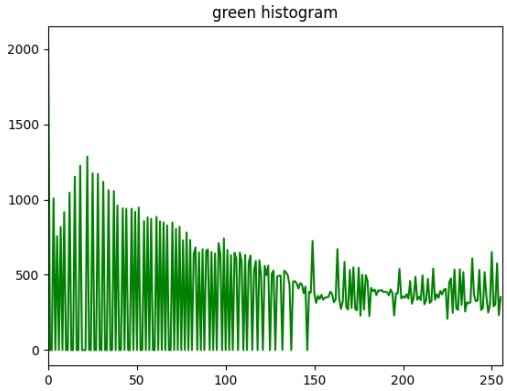

hist equalized blue image

(e)


blue histogram

(f)





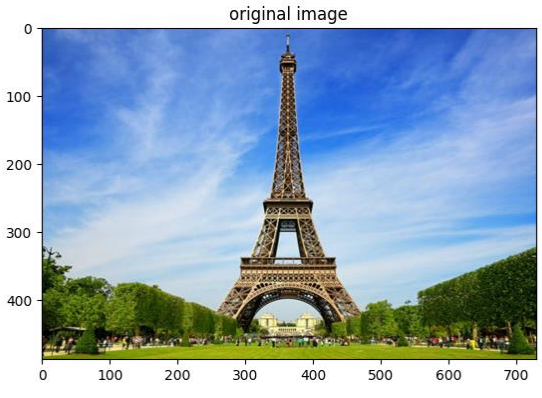
(g) (h)

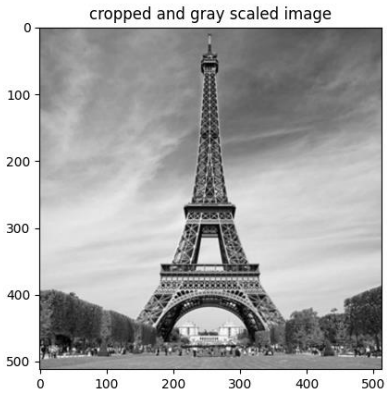
Fig 4 illustrates 4 pairs of equalized images and histograms. (a): histogram equalized image of gray scale image of image 1. (b): histogram of (a). (c): histogram equalized image of gray scale image of R channel of image 1. (d): histogram of (c). (e): histogram equalized image of gray scale image of B channel of image 1. (f): histogram of (e). (g): histogram equalized image of gray scale image of G channel of image 1. (h): histogram of (g).

We can observe that since the images for different channel have histograms with about a U shape, the equalized histograms are very similar to the original histograms.

Task-3: Image Denoising via a Gaussian Filter (5 marks)

1. Read in image2.jpg. Crop a square image region corresponding to the central 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] (0.5 marks).





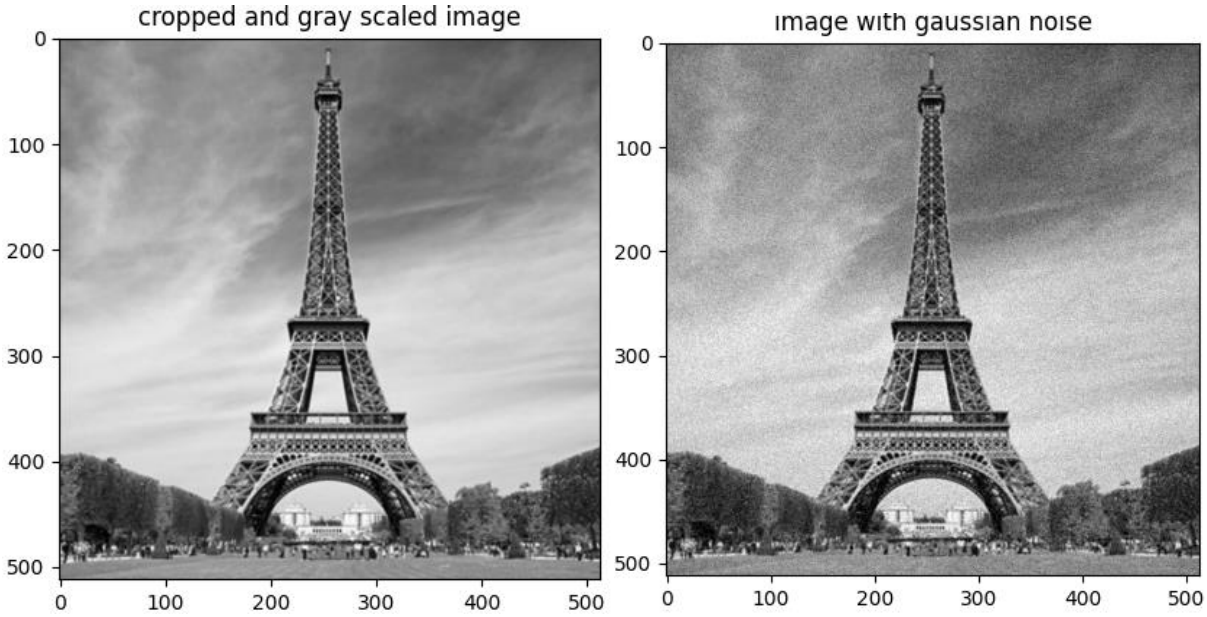
(a) (b)

Fig 5 illustrates the original and cropped gray scaled image of image 2. (a): the original image of image 2. (b): cropped gray scaled image of image 2.

The image is cropped from left and right, each side with length (img2.shape[1] - img2.shape[0]) // 2. After that, the image is resized and turned into grayscale.

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 (0.5 marks). *The intensity values of the Generated image should be within [0, 255].*

Hint: Make sure your input image range is within [0, 255]. Kindly, you may need *np.random.randn()* in Python. While Matlab provides a convenient function *imnoise()*. Please check the default setting of these inbuilt function.

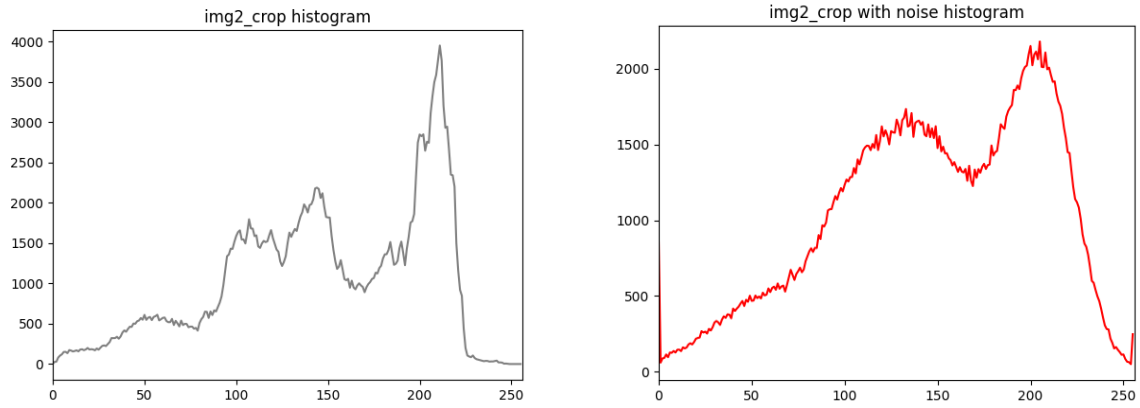


(a) (b)

Fig 6 illustrates the cropped gray scaled images of image 2with and without Gaussian noise. (a): the cropped gray scaled image of image 2. (b): the image with Gaussian noise of image 2.

The noise (a np array with size equal to that of the original image) was generated with np.random.randn, multiplying it with standard deviation, and adding it with mean. After that, the values outside of [0,255] is clipped by assigning all index with values less than 0 to be 0 and values more than 255 to be 255. Finally, a type conversion is performed from double into uint8 to unify the datatype.

3. Display the two histograms side by side, one before adding the noise and one after adding the noise (0.25 marks for each histogram, 0.5 marks in total).



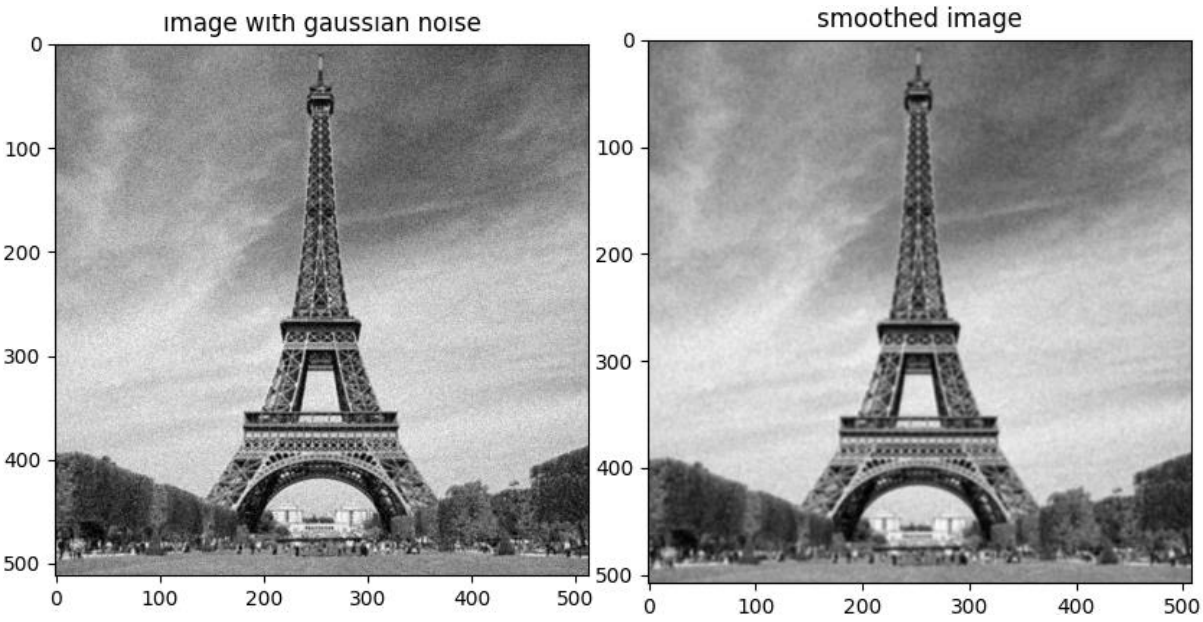
(a) (b)

Fig 7 illustrates the histograms of cropped gray scaled image and the image with Gaussian noise of image 2. (a): the histogram of the cropped gray scaled image of image 2. (b): the histogram of the image with Gaussian noise of image 2.

4. Implement your own Matlab/Python function that performs a 5x5 Gaussian filtering (1.5 marks). Your function interface is:

```python
# %% 4
# Function performing 5x5 Gaussian filtering
def my_Gauss_filter(noisy_img, my_gauss_kernel):
    # Detect if image is with enough size
    if noisy_img.shape[0] < 5 or noisy_img.shape[1] < 5:
        raise Exception('Need image greater than 5x5')
    # Initialize a array of zeros with the size after filtering: (x-4,y-4)
    ans = np.zeros((noisy_img.shape[0] - 4, noisy_img.shape[1] - 4))
    # Iterate through the original image
    for i in range(noisy_img.shape[0] - 4):
        for j in range(noisy_img.shape[1] - 4):
            # Apply the kernel to the selected part of image with convolution
            # operation and update the answer image
            ans[i, j] = np.sum(noisy_img[i:i + 5, j:j + 5] * my_gauss_kernel)
    return ans
```

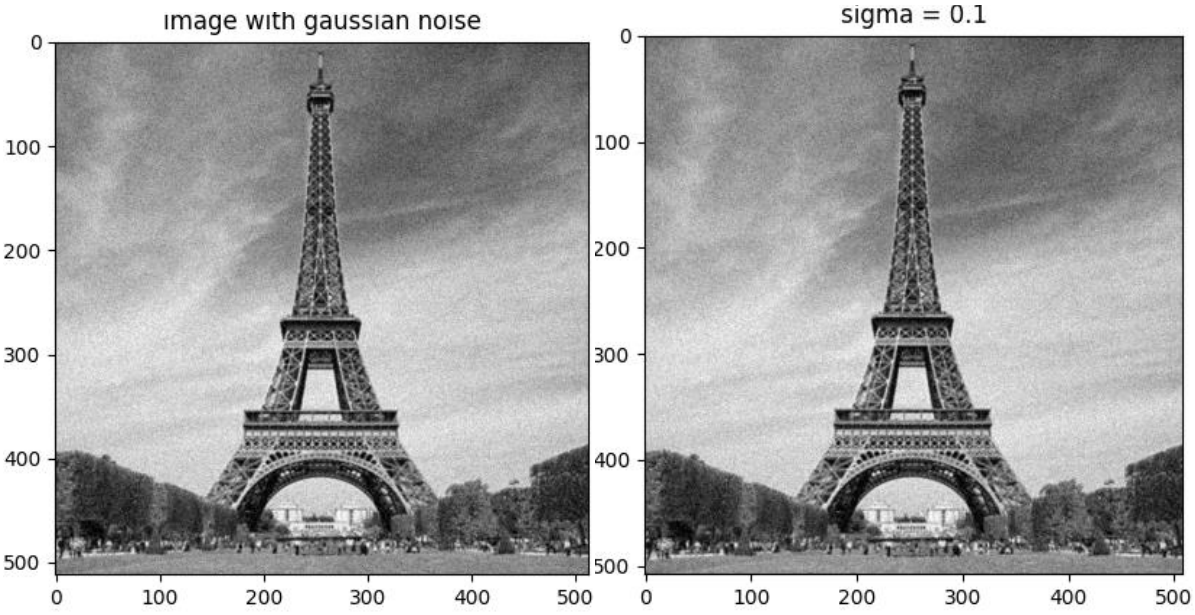
5. Apply your Gaussian filter to the above noisy image, and display the smoothed images and visually check their noise-removal effects (0.5 marks in total).



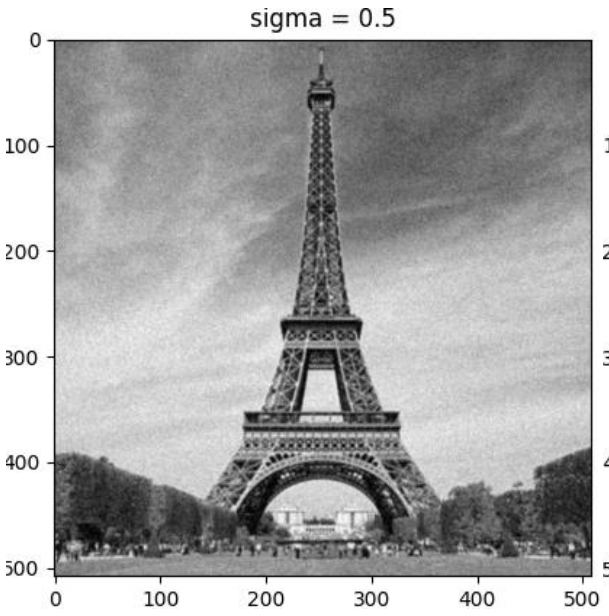
(a) (b)

Fig 8 illustrates the cropped gray scaled image with Gaussian noise of image 2 and its smoothed version (a): the image with Gaussian noise of image 22. (b): smoothed version of (a).

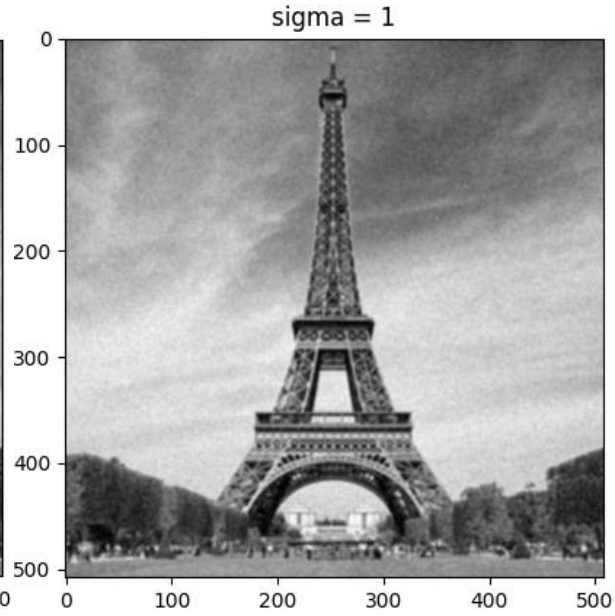
One of the key parameters to choose for the task of image filtering is the standard deviation of your Gaussian filter. You may need to test and compare different Gaussian kernels with different standard deviations (1.0 marks).



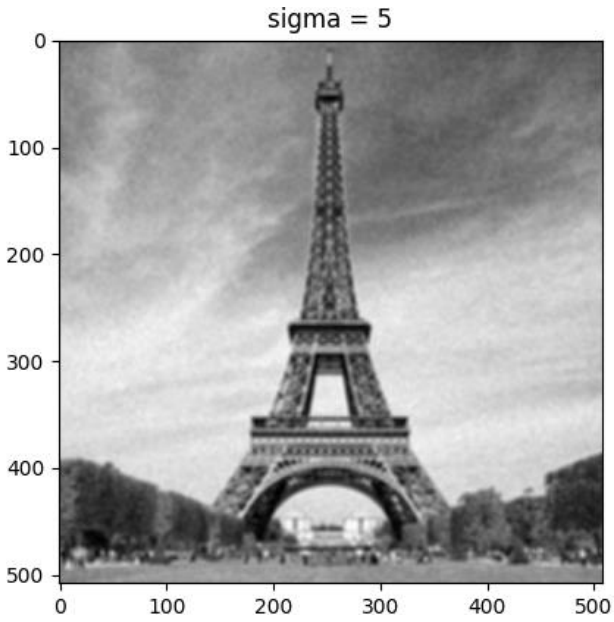
(a) (b)


sigma = 0.5
0
100
200
300
400
500
0
100
200
300
400
500

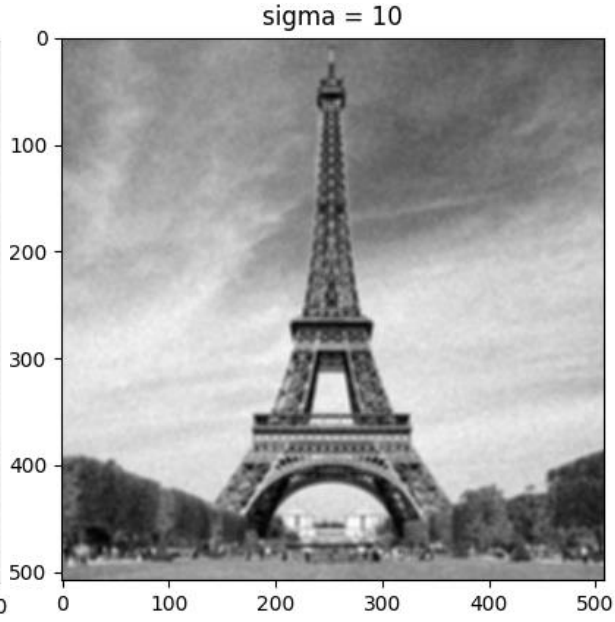
(c)


sigma = 1
0
100
200
300
400
500
0
100
200
300
400
500

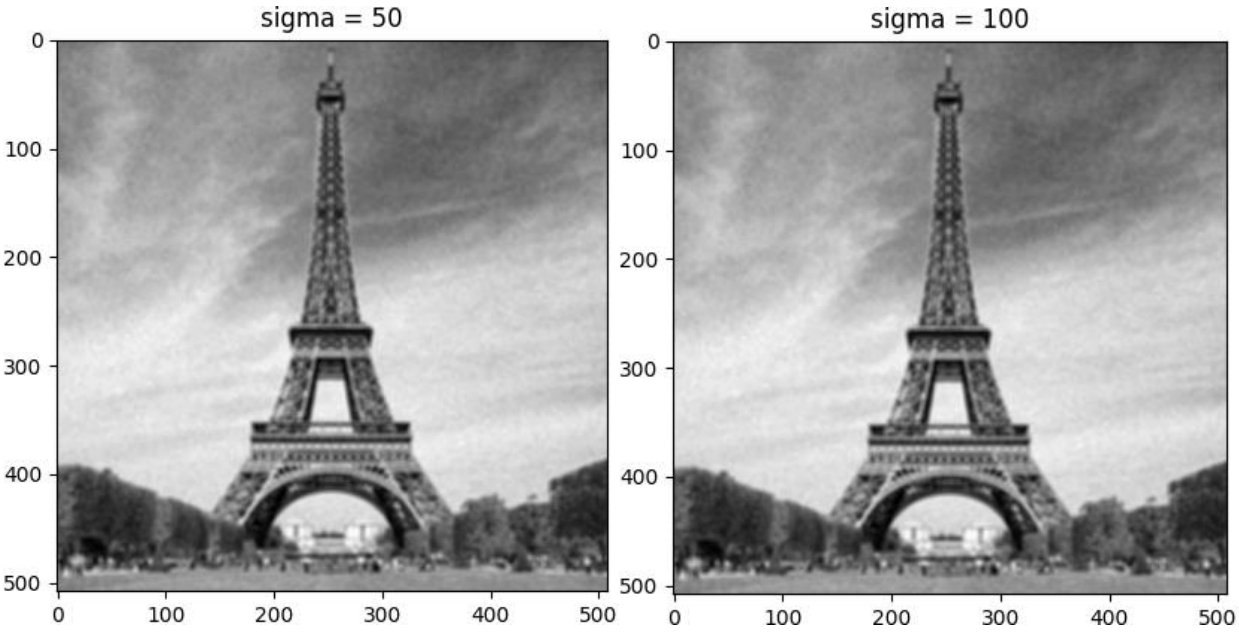
(d)


sigma = 5
0
100
200
300
400
500
0
100
200
300
400
500

(e)


sigma = 10
0
100
200
300
400
500
0
100
200
300
400
500

(f)



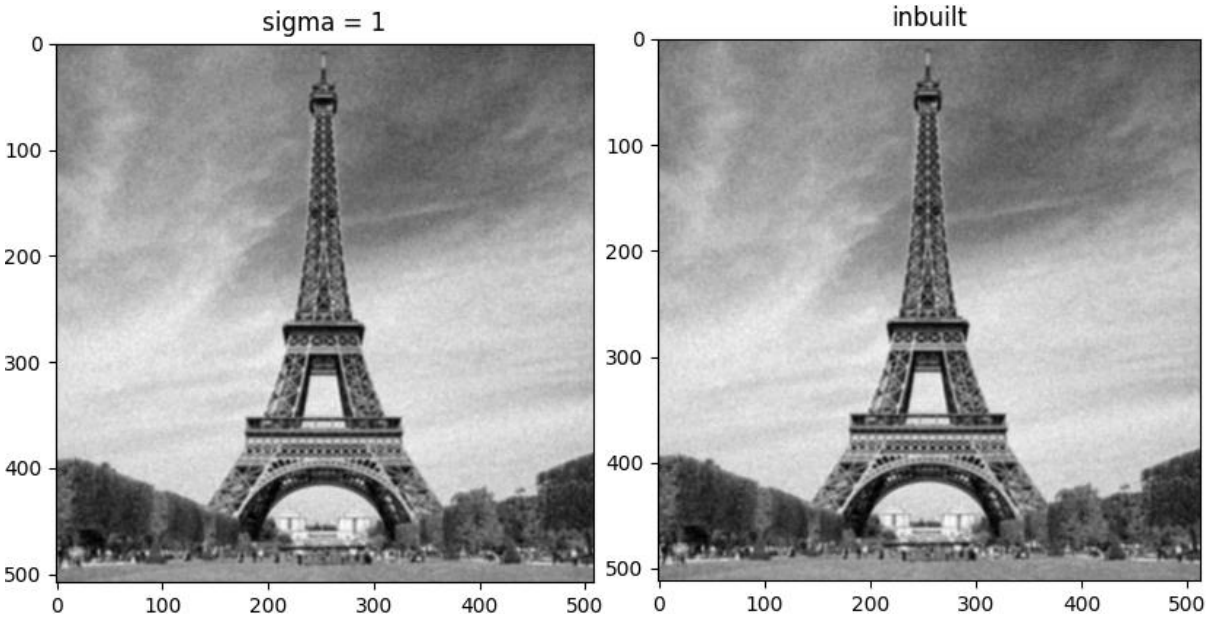
(g) (h)

Figure 9 The contrast between filtered images after applying Gaussian filter with different sigma. (a) original image (b) sigma = 0.1 (c) sigma = 0.5 (d) sigma = 1 (e) sigma = 5 (f) sigma = 10 (g) sigma = 50 (h) sigma = 100

Note: In doing this task you MUST NOT use any Matlab's (or Python's) inbuilt image filtering functions (e.g. *imfilter()*, *filter2()* in Matlab, or *cv2.filter2D()* in Python). In other words, you are required to code your own 2D filtering code, based on the original mathematical definition for 2D convolution. However, you are allowed to generate a 5x5 sized Gaussian kernel with inbuilt functions.

6. Compare your result with that by Python's or Matlab's inbuilt 5x5 Gaussian filter (e.g. conveniently cv2.GaussianBlur() in Python or filter2(), imfilter() in Matlab). Please show whether the two results are nearly identical (0.5 marks).

Further reading material: http://setosa.io/ev/image-kernels/



(g) (h)

Figure 10 The contrast between filtered images after applying Gaussian filter with different sigma. (a) original image (b) sigma = 1

The left image is generated with my own function using Gaussian kernel with standard deviation of 1 and the right one is generated with inbuilt function in cv2. They are nearly identical and actually have 50% of identical pixel values.

Task -4: Implement your own 3x3 Sobel filter in Matlab/Python (3 marks)

Sobel edge detector.

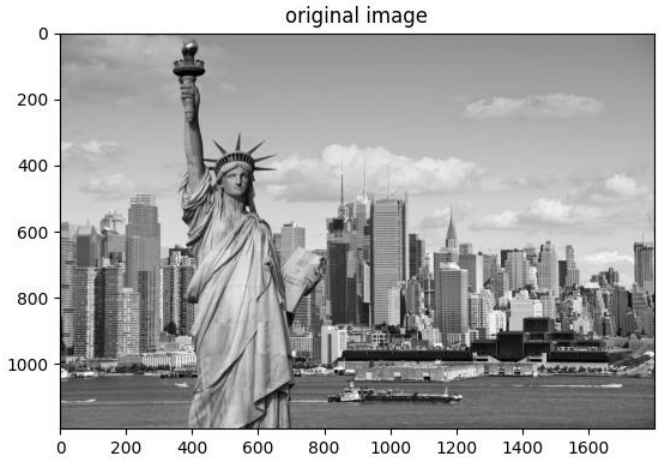
You need to implement your own 3x3 Sobel filter. Again, you must not use inbuilt functions or any inbuilt edge detection filter. The implementation of the filter should be clearly presented with your code.

```python
# sobel filter
def sobel_filter(img):
    # create the sobel kernel with size 3x3
    sobel_kernel = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
    # padding the image
    img_padding = np.zeros((img.shape[0] + 2, img.shape[1] + 2))
    img_padding[1:-1, 1:-1] += img
    # apply the sobel filter in y direction
    ans = np.zeros((img.shape[0], img.shape[1]))
    for i in range(img_padding.shape[0] - 2):
        for j in range(img_padding.shape[1] - 2):
            ans[i, j] = np.sum(img_padding[i:i + 3, j:j + 3] * sobel_kernel)
    # apply sobel filter in x direction
    ans1 = np.zeros((img.shape[0], img.shape[1]))
    sobel_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    for i in range(img_padding.shape[0] - 2):
        for j in range(img_padding.shape[1] - 2):
            ans[i, j] = np.sum(img_padding[i:i + 3, j:j + 3] * sobel_kernel)
    # merging the two results
    ans = np.sqrt(ans ** 2 + ans1 ** 2)
    # normalize the result
    ans = (ans * 255 / ans.max()).astype(np.uint8)
    return ans
```

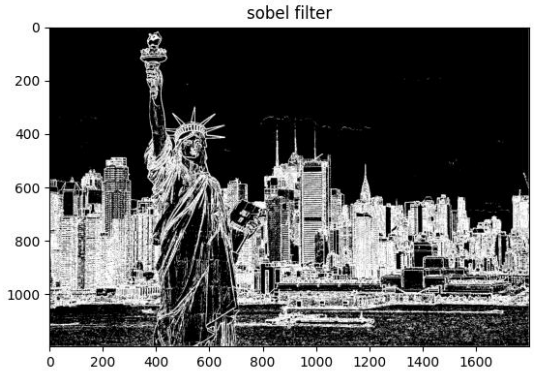
Test it on the image, image3.jpg, and compare your result with inbuilt Sobel edge detection function (0.5 marks),

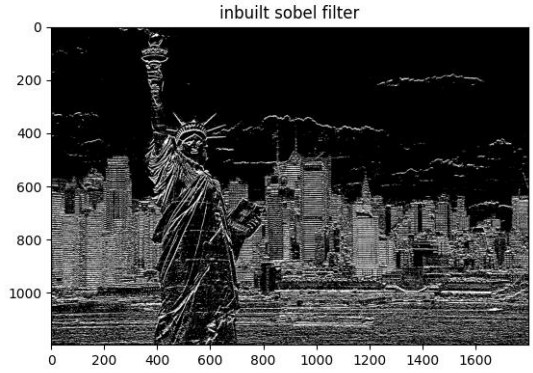
Show 3 images here and provide appropriate figure captions.

image3.jpg **(grayscale),** image that after applying your Sobel edge detection function, image that after applying inbuilt Sobel edge detection function



(a)





(b) (c)

Figure 11 The contrast between filtered images after applying Sobel filter with my function and inbuilt function. (a) original image (b) image applied my function (c) image applied inbuilt function

Briefly explain the function of Sobel filter and how it can achieve this function (0.5 mark).

Firstly, I create two Sobel filters (horizontal and vertical), then, I padded the input image with 1 pixel with value 0 to keep the size of the resulting image the same as the input. After that, I created a matrix with the same size of the unpadded input image. In the main computation part, convolution operation is performed with a 3x3 sub-part of the padded input image with center (i, j) and the Sobel filters. The result is updated into the output matrix with coordinate (i, j). Finally, I merged them together, clipped, end returned the result. As the edges of an image can be recognized by us if the dividing parts of the edge

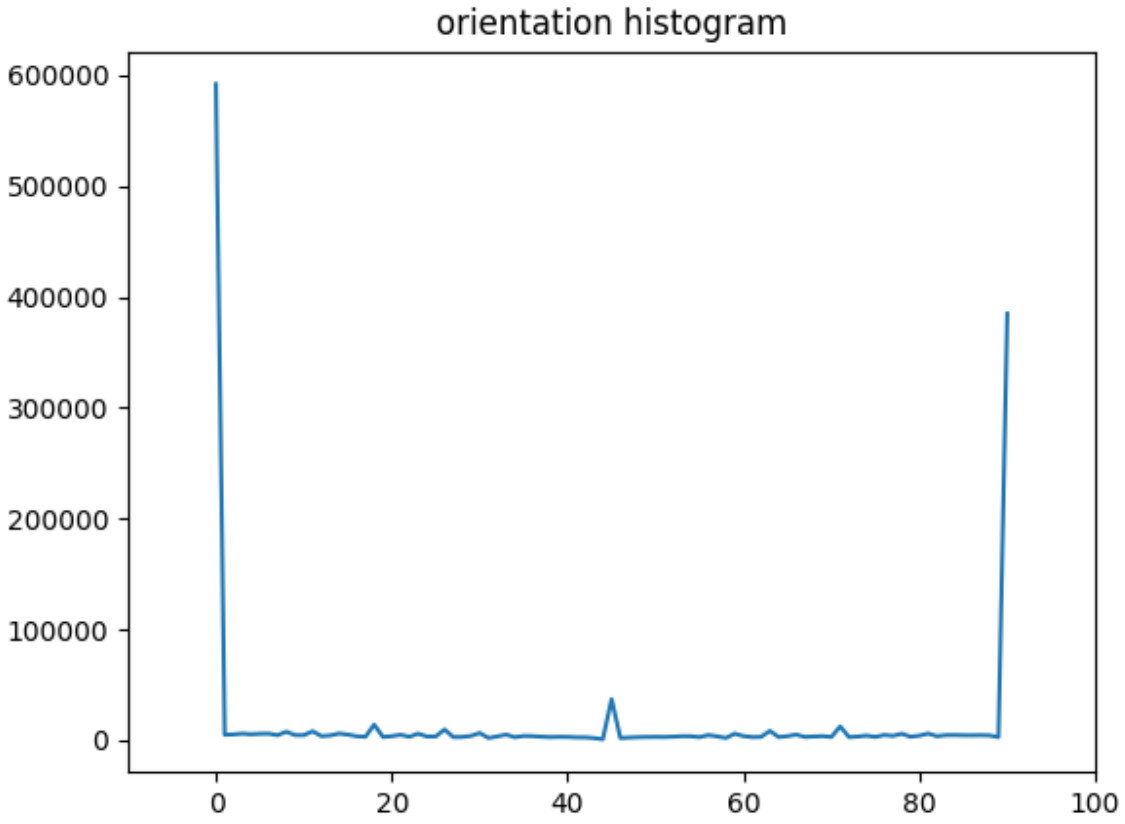
have very large gradient, it could achieve this function because with the convolution operation, left and right (for the vertical Sobel filter) has very different values, the result should be larger, which is consistent with our expectation: larger gradient (difference) on the two sides of the edge.

Investigate the accuracy of the Sobel filter in terms of predicting the orientation of the edge. If it is used for edge detection, what might cause any inaccuracy of the edge orientation estimation that you see? Look at the histogram of the gradient orientation and find out the major edge orientations in the image. Please discuss whether the gradient orientation histogram align with respect to the real expected edges for the image or not.

You can use the inbuilt Sobel function to discuss this question. (2.0 mark)

If Sobel filter is used for edge detection, noises could cause the inaccuracy because any extreme values at the edge of the filter will cause the resulting value to be huge enough to be seen as an edge. We can use Gaussian blur to minimize the inaccuracy. Moreover, the size of the filter could also affect the accuracy. We need to measure the tradeoff to choose a proper filter size because small filters could detect thin edge but vulnerable to noises. Large filters are less vulnerable to noises but cannot accurately detect thin edges.

It can be observed that the filter could predict the edge with high accuracy. The main cause of the inaccuracy would be the noise pixels in the image.



(a)

Figure 12 The histogram of orientation of edges after applying Sobel filter with horizontal and vertical directions.

We can observe from the histogram of orientation plotted with arctan function to horizontal and vertical edge that the major edge orientation is 0 degree (horizontal). Since the buildings in the image are tall, I expected that the number of vertical edges would be larger. However, the sum of length of horizontal edges are longer because there are a lot more vertical lines representing the floors in the buildings.

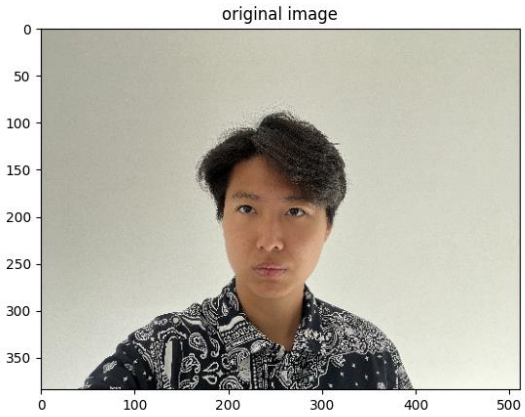
Task -5: Image Rotation (3 marks)

Take one frontal face photo of yourself, under normal lighting condition, against a white wall. (i.e., as if a passport photo). The image should be in landscape shape (i.e., the longer side is in the horizontal direction), and resized to 384x 512. (Height x Width)

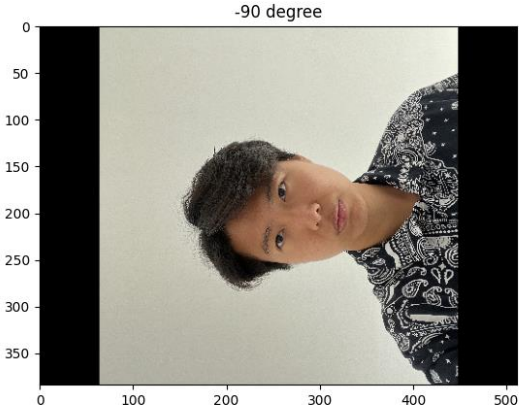
1. Implement your own function my_rotation() for image rotation by any given angle between [-90º, 90º]. Display images rotated by -90º, -45º, -15º, 45º, and 90º (0.20 for each image, 1.0 mark in total).

Note: positive for clockwise. Negative for anti-clockwise. Eg. -45^{o} means rotate 45^{o} in anti-clockwise direction.

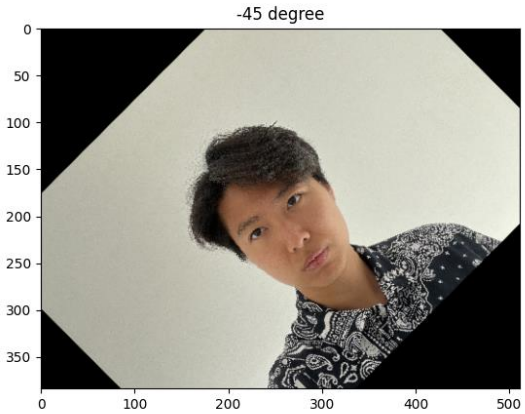
```python
# %% 1
def my_rotation(img, angle):
    if angle > 90 or angle < -90:
        raise Exception('angle should be in [-90,90]')
    # get the rotation matrix
    rotation_matrix = cv2.getRotationMatrix2D((img.shape[1] // 2, img.shape[0] // 2), -angle, 1)
    # get the rotated image
    rotated_img = cv2.warpAffine(img, rotation_matrix, (img.shape[1], img.shape[0]))
    return rotated_img
```



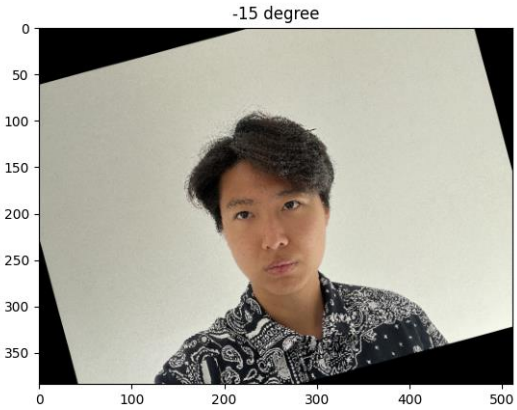
(a)



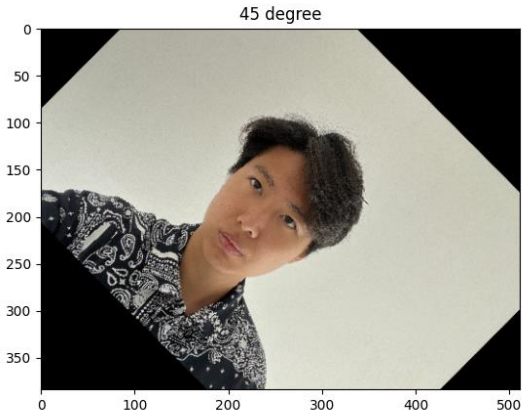
(b)



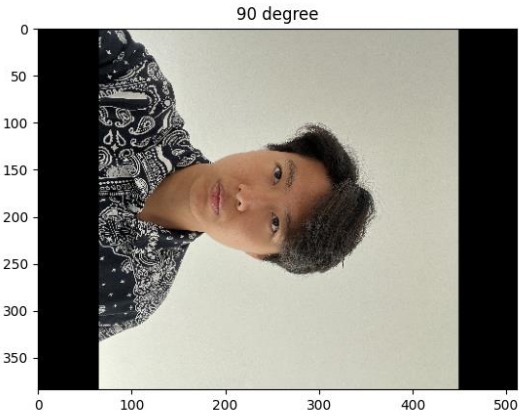
(c)



(d)



(e)

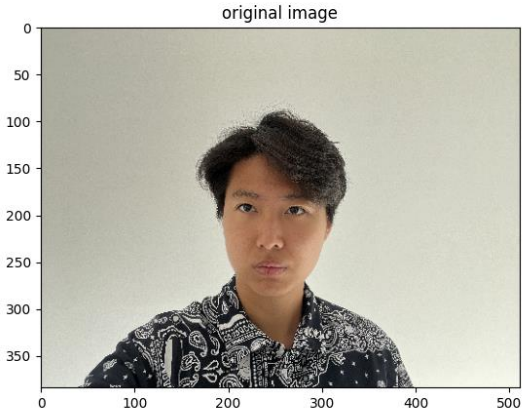


(f)

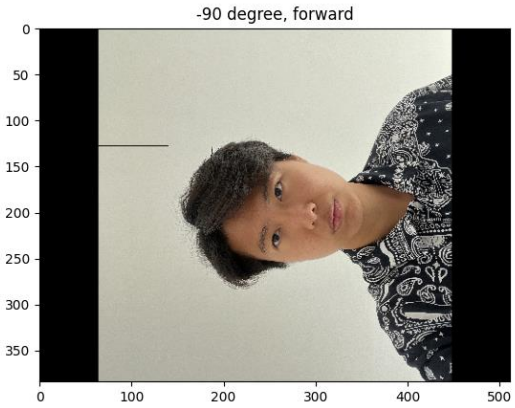
Figure 13 The contrast between filtered images after applying rotation. (a) original image (b) image rotated with -90 degrees clockwise (c) image rotated with -445 degrees clockwise (d) image rotated with -15 degrees clockwise (e) image rotated with 45 degrees clockwise (f) image rotated with \90 degrees clockwise

2. Compare forward and backward mapping and analyze their difference (1.0 mark).
[Hints: you are required to write my_rotation(), using, forward mapping and backward mapping, respectively. Obtain the images after rotating using two different methods.]

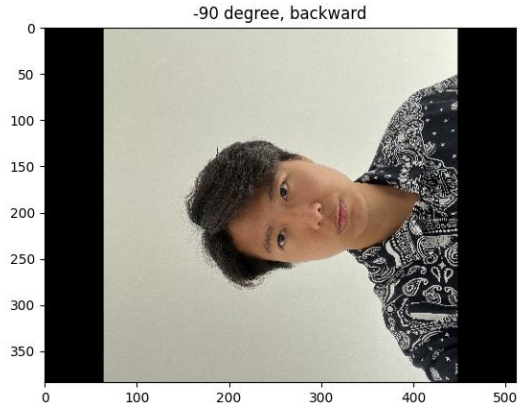
```python
def my_rotation1(img, angle, forward=True):
    # create the result image
    img1 = np.zeros((img.shape[0], img.shape[1], img.shape[2]))
    # detect if the angle is in [-90,90]
    if angle > 90 or angle < -90:
        raise Exception('angle should be in [-90,90]')
    # get the center of the image
    center = np.array([img.shape[1] // 2, img.shape[0] // 2])
    # angle to radian
    pi_degree = math.radians(angle)
    # create the rotation matrix
    if forward:
        warping_matrix = np.array([[math.cos(pi_degree), -math.sin(pi_degree)], [math.sin(pi_degree),
                                                                                 math.cos(pi_degree)]])
    else:
        warping_matrix = np.linalg.inv(np.array([[math.cos(pi_degree), -math.sin(pi_degree)], [math.sin(pi_degree),
                                                                                               math.cos(pi_degree)]]))
    # iterate through index of the source or the result image
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            # center the coordinate
            coordinate = np.array([j, i]) - center
            # perform the rotation
            coordinate_new = np.dot(warping_matrix, coordinate)
            # generate the new coordinate
            x = int(coordinate_new[0] + center[0])
            y = int(coordinate_new[1] + center[1])
            # check if the new coordinate is in the image
            if img.shape[1] > x >= 0 and img.shape[0] > y >= 0:
                # update the result image
                if forward:
                    img1[y, x] = img[i, j]
                else:
                    img1[i, j] = img[y, x]
    return img1.astype(np.uint8)
```



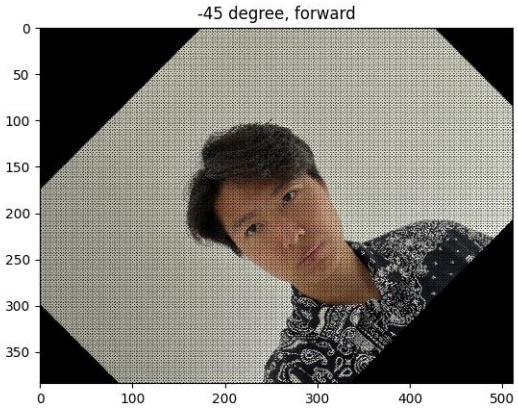
(a)


-90 degree, forward

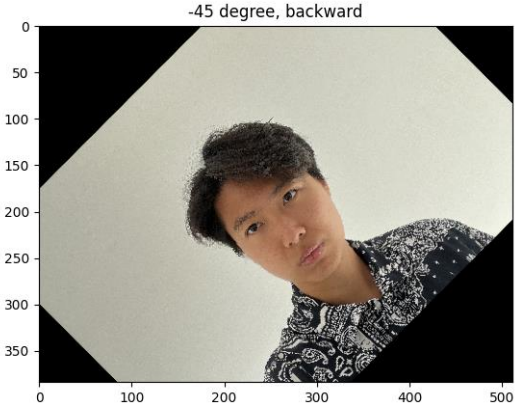
(b)


-90 degree, backward

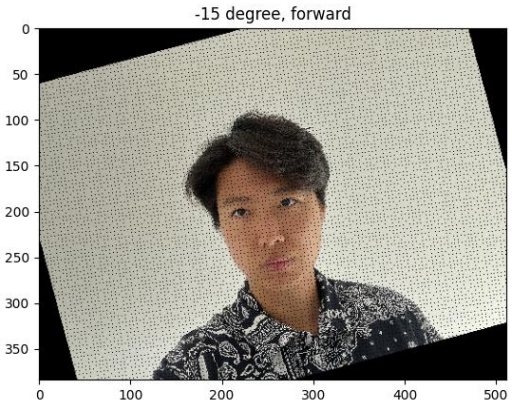
(c)


-45 degree, forward

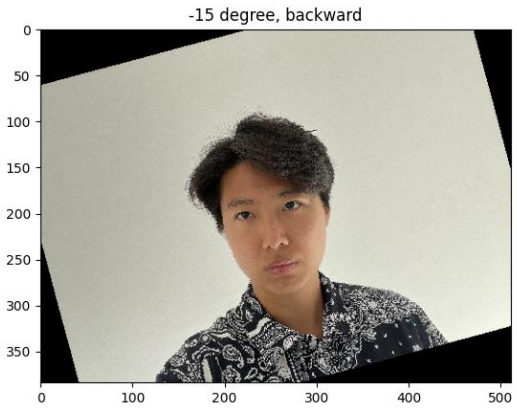
(d)


-45 degree, backward

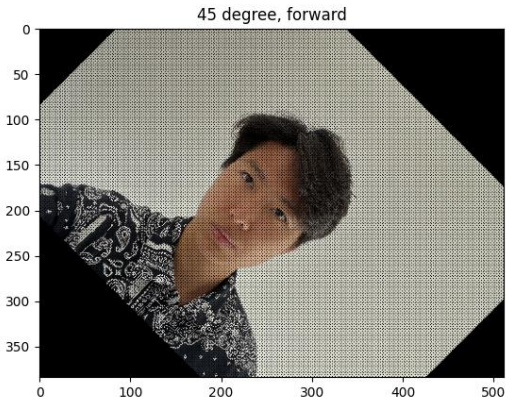
(e)


-15 degree, forward

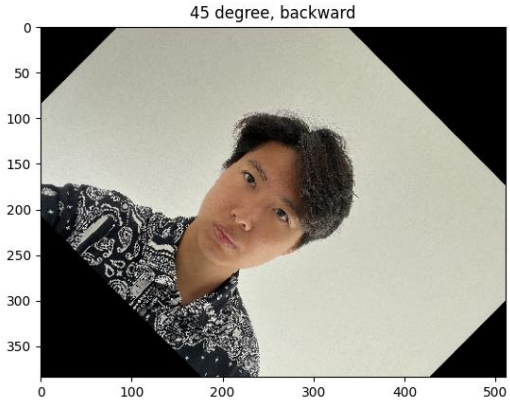
(f)


-15 degree, backward

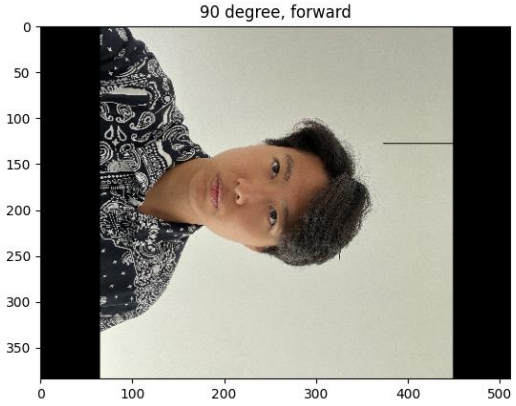
(g)



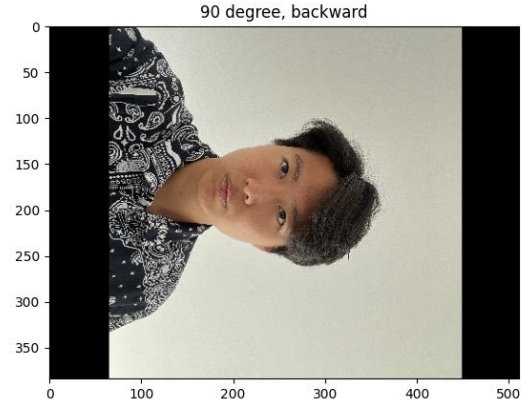
(h)



(i)



(j)

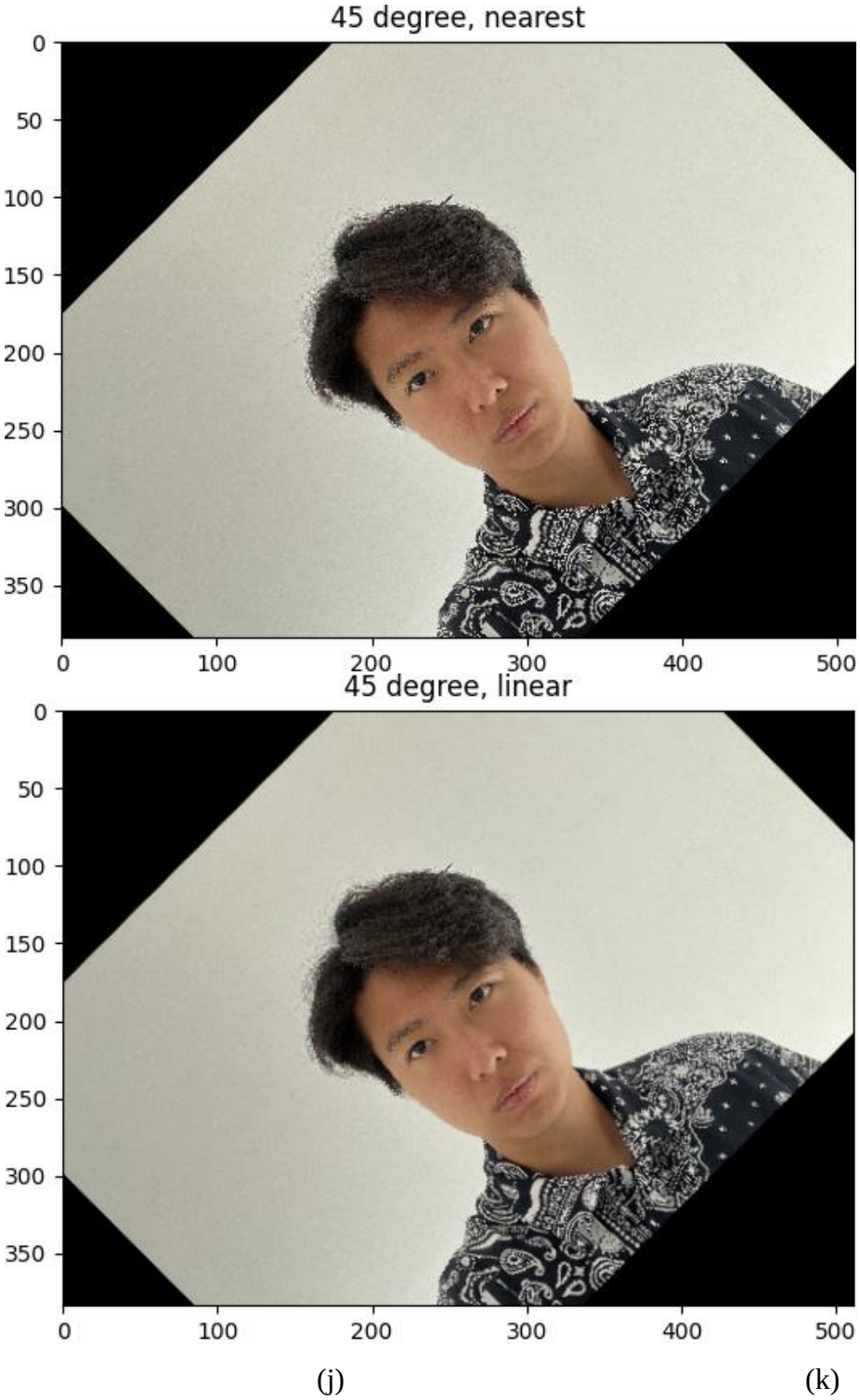


(k)

Figure 14 The contrast between filtered images after applying rotation with forward mapping and backward mapping. (a) original image (b) image rotated with -90 degrees clockwise with forward mapping (c) image rotated with -90 degrees clockwise with backward mapping (d) image rotated with -45 degrees clockwise with forward mapping (e) image rotated with -45 degrees clockwise with backward mapping (f) image rotated with -15 degrees clockwise with forward mapping (g) image rotated with -15 degrees clockwise with backward mapping (h) image rotated with 45 degrees clockwise with forward mapping (i) image rotated with 45 degrees clockwise with backward mapping (j) image rotated with 90 degrees clockwise with forward mapping (k) image rotated with 90 degrees clockwise with backward mapping

The forward method generated images with holes, but the ones generated by backward method do not contain them. In conclusion, the backward mapping could generate better results.

3. Compare the methods using bilinear interpolation or using nearest neighbor method in the inverse mapping/warping method and analyze their differences (1.0 mark).



(j) (k)

Figure 15 The contrast between filtered images after applying rotation with bilinear interpolation and nearest neighbor method (a) images after applying rotation with nearest neighbor method (b) images after applying rotation with bilinear interpolation method.

We can see that the one with nearest neighbor method is sharper because it only copy the color value into the new image at the calculated position, causing the loss of some pixels between their neighbors. However, the bilinear method calculates the color with weighted average with nearby colors, which will make the result images more continuous.