DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UCLA
EC ENGR 188: THE FOUNDATIONS OF COMPUTER VISION

INSTRUCTOR: Prof. Achuta Kadambi
TA: Howard Zhang

NAME: Haoxuan Li
UID: 705738656

HOMEWORK 1

PROBLEM	TYPE	TOPIC	MAX. POINTS
1	Analytical	LSI Systems	10
2	Coding	2D-Convolution	5
3	Coding	Image Blurring and Denoising	15
4	Coding	Image Gradients	5
5	Analytical + Coding	Image Filtering	15
6	Analytical	Interview Question (Bonus)	10

Motivation

The long-term goal of our field is to teach robots how to see. The pedagogy of this class (and others at peer schools) is to take a *bottom-up* approach to the vision problem. To teach machines how to see, we must first learn how to represent images (lecture 2), clean images (lecture 3), and mine images for features of interest (edges are introduced in lecture 4 and will thereafter be a recurring theme). This is an evolution in turning the matrix of pixels (an unstructured form of "big data") into something with structure that we can manipulate.

The problem set consists of:

- analytical questions to solidify the concepts covered in the class, and
- coding questions to provide a basic exposure to image processing using Python.

You will explore various applications of convolution such as image blurring, denoising, filtering and edge detection. These are fundamental concepts with applications in many computer vision and machine learning applications. You will also be given a computer vision job interview question based on the lecture.

Homework Layout

The homework consists of 6 problems in total, with subparts for each problem. There are 2 types of problems in this homework - analytical and coding. All the problems need to be answered in the Overleaf document. Make a copy of the Overleaf project, and fill in your answers for the questions in the solution boxes provided.

For the analytical questions you will be directly writing their answers in the space provided below the questions. For the coding problems you need to use the Jupyter notebook provided Jupyter notebook (see the Jupyter notebook for each sub-part which involves coding). After writing your code in the Jupyter notebook you need to copy paste the same code in the space provided below that question on Overleaf. For instance, for Question 2 you have to write a function 'conv2D' in the Jupyter notebook (and also execute it) and then copy that function in the box provided for Question 2 here in Overleaf. In some questions you are also required to copy the saved images (from Jupyter) into the solution boxes in Overleaf. For instance, in Question 3.2 you will be visualizing the gaussian filter. So you will run the corresponding cells in Jupyter (which will save an image for you in PDF form) and then copy that image in Overleaf.

Submission

You will need to make two submissions: (1) Gradescope: You will submit the Overleaf PDF with all the answers on Gradescope. (2) We will send out announcement later on how to submit your Jupyter Notebook (.ipynb file) with all the cells executed.

Software Installation

You will need Jupyter to solve the homework. You may find these links helpful:

- Jupyter (https://jupyter.org/install)
- Anaconda (https://docs.anaconda.com/anaconda/install/)

1 Image Processing

1.1 Periodic Signals (1.0 points)

Is the 2D complex exponential $x(n_1,n_2) = \exp(j(\omega_1 n_1 + \omega_2 n_2))$ periodic in space? Justify.

$x(n_1+N_1, n_2+N_2) = x(n_1,n_2)$

Substituting the definition of $x(n_1,n_2)$

$\exp(j(\omega_1(n_1+N_1)+\omega_2(n_2+N_2))) = \exp(j(\omega_1 n_1+\omega_2 n_2))$

$\exp(j(\omega_1 n_1+\omega_2 n_2))\exp(j(\omega_1 N_1+\omega_2 N_2)) = \exp(j(\omega_1 n_1+\omega_2 n_2))$

Cancelling

$\exp(j(\omega_1 N_1+\omega_2 N_2)) = 1$

This equation holds if and only if $\omega_1 N_1 + \omega_2 N_2$ is an integer multiple of 2π where k is an integer.

$\omega_1 N_1 + \omega_2 N_2 = 2\pi k$

Periodic in space if and only if the ratio ω_1/ω_2 is rational.

1.2 Working with LSI systems (3.0 points)

Consider an LSI system $T[x] = y$ where x is a 3 dimensional vector, and y is a scalar quantity. We define 3 basis vectors for this 3 dimensional space: $x_1 = [1,0,0]$, $x_2 = [0,1,0]$ and $x_3 = [0,0,1]$.
(i) Given $T[x_1] = a$, $T[x_2] = b$ and $T[x_3] = c$, find the value of $T[x_4]$ where $x_4 = [5,4,3]$. Justify your approach briefly (in less than 3 lines).
(ii) Assume that $T[x_3]$ is unknown. Would you still be able to solve part (i)?
(iii) $T[x_3]$ is still unknown. Instead you are given $T[x_5] = d$ where $x_5 = [1,-1,-1]$. Is it possible to now find the value of $T[x_4]$, given the values of $T[x_1], T[x_2]$ (as given in part (i)) and $T[x_5]$? If yes, find $T[x_4]$ as a function of a,b,d; otherwise, justify your answer.

(i) Assuming LSI, express x_4 as a linear combination of x_1, x_2, and x_3. That is by linearity since $x_4 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3$ then $T[x_4] = \alpha_1 T[x_1] + \alpha_2 T[x_2] + \alpha_3 T[x_3]$, therefore if $x_4 = [5,4,3]$ then $T[x_4] = 5a+4b+3c$.

(ii) No, we would not be able to solve part (i) if $T[x_3]$ is unknown. This is because x_4 is expressed as a linear combination of x_1, x_2, and x_3, but we do not have enough information to determine the coefficients of this linear combination without knowing $T[x_3]$.

(iii) Yes, assuming LSI, express x_4 as a linear combination of x_1, x_2, and x_5. That is by linearity since $x_4 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_5 x_5$ then $T[x_4] = \alpha_1 T[x_1] + \alpha_2 T[x_2] + \alpha_5 T[x_5]$.

$T[x_4] = 5a+4b+3(a-b-e) \rightarrow T[x_4] = 8a+b-3d.$

1.3 Space invariance (2.0 points)

Evaluate whether these 2 linear systems are space invariant or not. (The answers should fit in the box.)
(i) $T_1[x(n_1)] = 2x(n_1)$
(ii) $T_2[x(n_1)] = x(2n_1)$.

(i) $T_1[x(n_1)] = 2x(n_1)$ is space invariant.

$T_1[x(n_1 - k)] = 2x(n_1 - k) \rightarrow 2x(n_1 - k)$

(ii) $T_2[x(n_1)] = x(2n_1)$ is space variant

$T_2[x(n_1 - k)] = x(2(n_1 - k)) = x(2n_1 - 2k) \nrightarrow x(2n_1 - k)$

1.4 Convolutions (4.0 points)

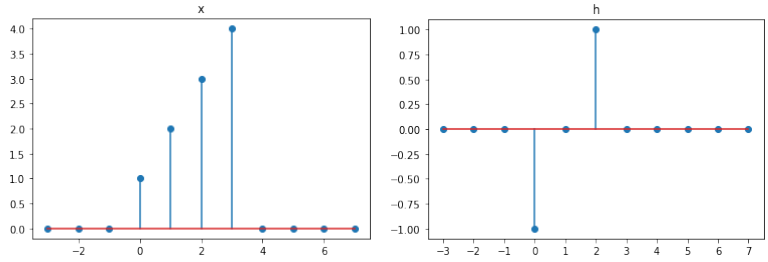


Figure 1: (a) Graphical representation of x (b) Graphical representation of h

Consider 2 discrete 1-D signals $x(n)$ and $h(n)$ defined as follows:

$$\begin{aligned} x(i) &= i+1 \quad \forall i \in \{0,1,2,3\} \\ x(i) &= 0 \quad \forall i \notin \{0,1,2,3\} \\ h(i) &= i-1 \quad \forall i \in \{0,1,2\} \\ h(i) &= 0 \quad \forall i \notin \{0,1,2\} \end{aligned} \tag{1}$$

(i) Evaluate the discrete convolution $h * x$.
(ii) Show how you can evaluate the non-zero part of $h * x$ as a product of 2 matrices H and X. Use the commented latex code in the solution box for typing out the matrices.

(i)

	0	1	2	3	4	
*	-1	0	1			
0	-1	-2	-3	-4		
	0	0	0	0	0	
		0	1	2	3	4
0	-1	-2	-2	-2	3	4

$h * x$ = [0, -1, -2, -2, -2, 3, 4] where '-1' denotes the zeroth index and anything outside this signal is presumed zero.

(ii)

$$H = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} \qquad X = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 0 \\ 0 \end{bmatrix}$$

$$H \cdot X = \begin{bmatrix} -1 \\ -2 \\ -2 \\ -2 \\ 3 \\ 4 \end{bmatrix}^T = \begin{bmatrix} -1 & -2 & -2 & -2 & 3 & 4 \end{bmatrix}$$

2 2D-Convolution (5.0 points)

In this question you will be performing 2D convolution in Python. Your function should be such that the convolved image will have the same size as the input image i.e. you need to perform zero padding on all the sides. (See the Jupyter notebook.)

This question is often asked in interviews for computer vision/machine learning jobs.

Make sure that your code is within the bounding box below.

```
def conv2D(image: np.array, kernel: np.array = None):
    padding_size = kernel.shape[0] // 2
    padded_image = np.pad(image, padding_size)

    convolved_image = np.zeros(image.shape)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            for m in range(kernel.shape[0]):
                for n in range(kernel.shape[0]):
                    convolved_image[i,j] += kernel[m,n]
                    * padded_image[i+m, j+n]

    return convolved_image
```

3 Image Blurring and Denoising (15.0 points)

In this question you will be using your convolution function for image blurring and denoising. Blurring and denoising are often used by the filters in the social media applications like Instagram and Snapchat.

3.1 Gaussian Filter (3.0 points)

In this sub-part you will be writing a Python function which given a filter size and standard deviation, returns a 2D Gaussian filter. (See the Jupyter notebook.)

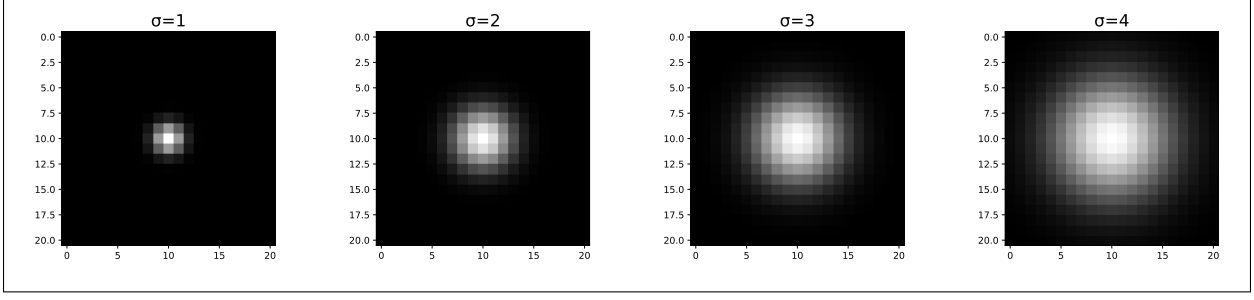
Make sure that your code is within the bounding box.

```
def gaussian_filter(size: int, sigma: float):
    grid = np.arange(-size//2+1, size//2+1)
    x, y = np.meshgrid(grid, grid)

    gauss = 1/(2*np.pi*sigma**2) * np.exp(-(x**2 + y**2)/(2*sigma**2))

    #normalize
    gauss = gauss / np.sum(gauss)

    return gauss
```

3.2 Visualizing the Gaussian filter (1.0 points)

(See the Jupyter notebook.) You should observe that increasing the standard deviation (σ) increases the radius of the Gaussian inside the filter.

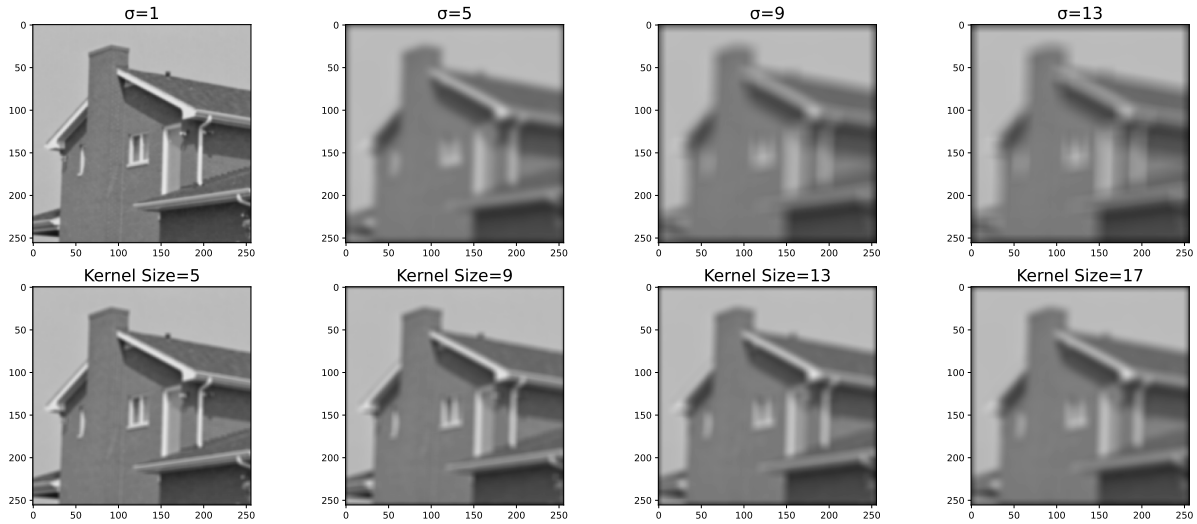
Copy the saved image from the Jupyter notebook here.



3.3 Image Blurring: Effect of increasing the filter size and σ (1.0 points)

(See the Jupyter notebook.) You should observe that the blurring should increase with the kernel size and the standard deviation.

Copy the saved image from the Jupyter notebook here.



3.4 Blurring Justification (2.0 points)

Provide justification as to why the blurring effect increases with the kernel size and σ?

Kernel size refers to the size of the matrix used to convolve with the image. Therefore, the larger the Kernel, the larger the matrix used to average out each pixel, leading to a more significant blurring effect.

The sigma value is related to the standard deviation of the Gaussian distribution. A larger sigma value results in a wider Gaussian distribution which means more pixels are included in the averaging process and having a more significant blurring effect.

3.5 Median Filtering (3.0 points)

In this question you will be writing a Python function which performs median filtering given an input image and the kernel size. (See the Jupyter notebook.)

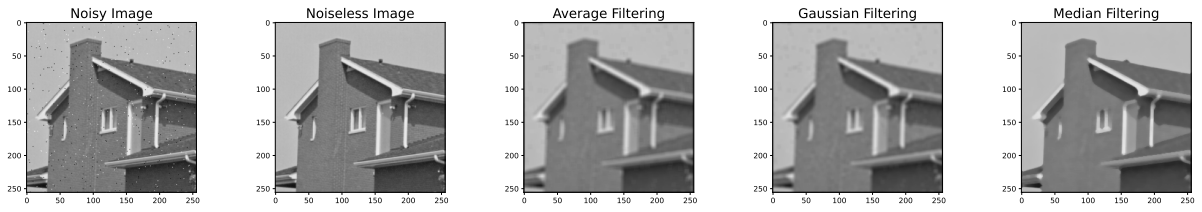
Make sure that your code is within the bounding box.

```python
def median_filtering(image: np.array, kernel_size: int = None):
    padding_size = kernel_size // 2
    padded_image = np.pad(image, padding_size)

    filtered_image = np.zeros(image.shape)

    for i in range(filtered_image.shape[0]):
        for j in range(filtered_image.shape[1]):
            filtered_image[i,j] = np.median(padded_image[i:i+kernel_size,
            j:j+kernel_size])

    return filtered_image
```

3.6 Denoising (1.0 points)

(See the Jupyter notebook.)

Copy the saved image from the Jupyter notebook here.



3.7 Best Filter (2.0 points)

In the previous part which filtering scheme performed the best? And why?

The average and Gaussian filter preformed similarly with overall blurring methods in order to filter out noise. The median is therefore the best because it removed the noise why limiting the blurring effect.

3.8 Preserving Edges (2.0 points)

Which of the 3 filtering methods preserves edges better? And why? Does this align with the previous part?

The median filter because unlike the average and Gaussian filter, the median filter is much better at preserving sharp edges because it is based off of the neighboring pixels.

4 Image Gradients (5.0 points)

In this question you will be visualizing the edges in an image by using gradient filters. Gradients filters, as the name suggests, are used for obtaining the gradients (of the pixel intensity values with respect to the spatial location) of an image, which are useful for edge detection.

4.1 Horizontal Gradient (1.0 points)

In this sub-part you will be designing a filter to compute the gradient along the horizontal direction. (See the Jupyter notebook.)

Make sure that your code is within the bounding box.

```
gradient_x = np.array([[1/6,0,-1/6],[1/6,0,-1/6],[1/6,0,-1/6]])
```

4.2 Vertical Gradient (1.0 points)

In this sub-part you will be designing a filter to compute the gradient along the vertical direction. (See the Jupyter notebook.)

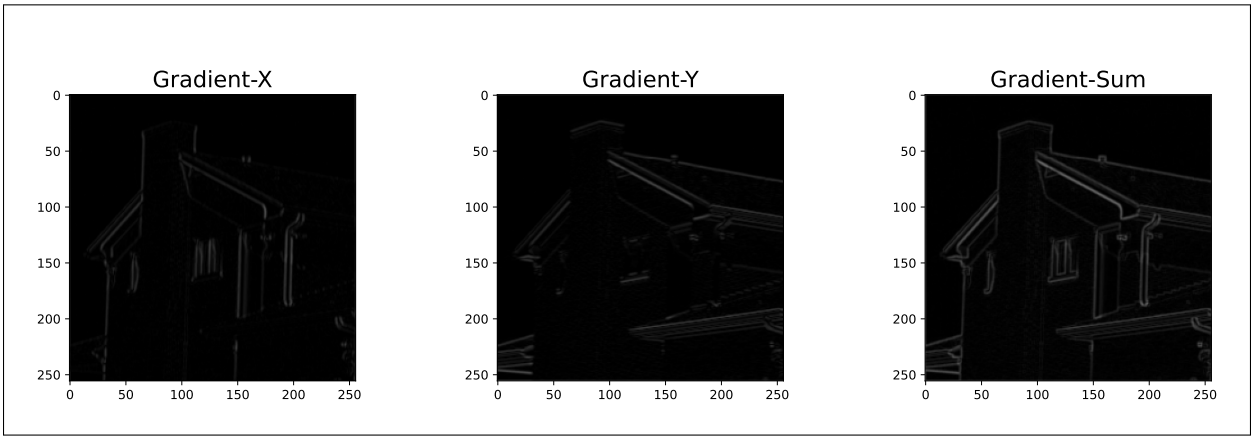
Make sure that your code is within the bounding box.

```
gradient_y = np.array([[1/6,1/6,1/6],[0,0,0],[-1/6,-1/6,-1/6]])
```

4.3 Visualizing the gradients (1.0 points)

(See the Jupyter notebook.)

Copy the saved image from the Jupyter notebook here.



4.4 Gradient direction (1.0 points)

Using the results from the previous part how can you compute the gradient direction at each pixel in an image?

After applying the Prewitt filter. Compute the direction by finding the arctan of the vertical derivative divided by the horizontal derivative. Then visualize the gradient direction of each pixel as an image where the pixel is colored based on its gradient direction.

4.5 Separable filter (1.0 points)

Is the gradient filter separable? If so write it as a product of 1D filters.

Yes the gradient filter is separable $G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ is the horizontal Prewit filter and $G_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$ is the vertical Prewitt filter. Therefore G = $G_x * G_y^T$ where the resulting 2D filter G can be used to compute the gradient of an image in the same way as the original 2D Prewitt filter.

5 Beyond Gaussian Filtering (15.0 points)

5.1 Living life at the edge (3.0 points)

The goal is to understand the weakness of Gaussian denoising/filtering, and come up with a better solution. In the lecture and the coding part of this assignment, you would have observed that Gaussian filtering does not preserve edges. Provide a brief justification.

[Hint: Think about the frequency domain interpretation of a Gaussian filter and edges.]

The Gaussian filter acts as a low-pass filter, which removes high-frequency components from the image, including edges. Filtering involves essentially spreading the intensity values of adjacent pixels, however because edges are sudden changes in intensity, they become averaged out with surrounding pixels which gives an overall blurring effect as corner begin to melt into the picture.

5.2 How to preserve edges (2.0 points)

Can you think of 2 factors which should be taken into account while designing filter weights, such that edges in the image are preserved? More precisely, consider a filter applied around pixel p in the image. What 2 factors should determine the filter weight for a pixel at position q in the filter window?

1. Proximity of pixel q to pixel p. Pixel that are more close to pixel p are more likely to be the same edge or structure and therefore should be given more weight in the filtering process. The Gaussian filter for example is a distance based filter that puts more emphasises on surrounding pixels.

2. Intensity of pixel q to pixel p. Pixel q with similar intensity to pixel p are more likely to be the same edge or structure. Therefore, following the idea of proximity, these intensities should be given higher weights.

5.3 Deriving a new filter (2.0 points)

For an image I, we can denote the output of Gaussian filter around pixel p as

$$GF[I_p] = \sum_{q \in S} G_\sigma(\|p-q\|) I_q.$$

I_p denotes the intensity value at pixel location p, S is the set of pixels in the neighbourhood of pixel p. G_{σ_p} is a 2D-Gaussian distribution function, which depends on $\|p-q\|$, i.e. the spatial distance between pixels p and q. Now based on your intuition in the previous question, how would you modify the Gaussian filter to preserve edges?

[Hint: Try writing the new filter as

$$BF[I_p] = \sum_{q \in S} G_\sigma\big(\|p-q\|\big) f\big(I_p, I_q\big) I_q.$$

What is the structure of the function $f(I_p, I_q)$? An example of structure is $f(I_p, I_q) = h(I_p \times I_q)$ where $h(x)$ is a monotonically increasing function in x?]

$f(I_p, I_q) = G_\sigma\big(\|I_p - I_q\|\big)$

5.4 Complete Formula (3.0 points)

Check if a 1D-Gaussian function satisfies the required properties for $f(.)$ in the previous part. Based on this, write the complete formula for the new filter BF.

$BF[I_p] = \frac{1}{W_p} \sum_{q \in S} G_\sigma\big(\|p-q\|\big) G_\sigma\big(\|I_p - I_q\|\big) I_q.$

Where $\frac{1}{W_p}$ is the normalization term in case the filtered image appears darker or lighter to the original image.

5.5 Filtering (3.0 points)

In this question you will be writing a Python function for this new filtering method (See the Jupyter notebook.)

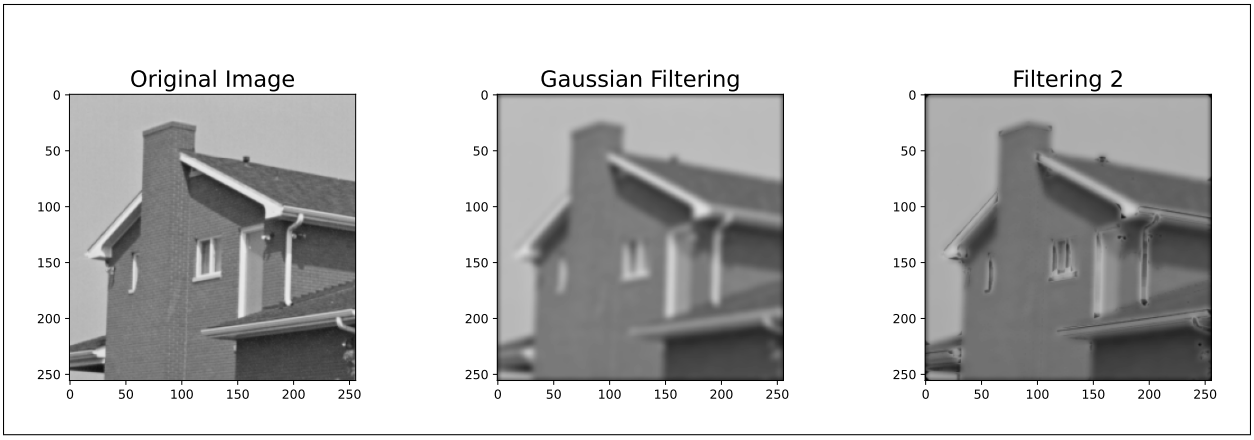
Make sure that your code is within the bounding box.

```
def filtering_2(image: np.array, kernel: np.array = None, sigma_int:
float = None, norm_fac: float = None):
    # Spatial gaussian filter
    spatial_filt_img = conv2D(image, kernel)

    # Intensity gaussian filter
    intensity_filt_img = median_filtering(image, kernel.shape[0])
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            #1/(2*np.pi*sigma**2) * np.exp(-(x**2 + y**2)/(2*sigma**2))
            intensity_filt_img[i,j] = 1/(2*np.pi*sigma_int**2) *
            np.exp(-(intensity_filt_img[i,j]-image[i,j])**2
            / (2*sigma_int**2))

    bilateral_image = spatial_filt_img * intensity_filt_img
```

```
    bilateral_image = bilateral_image / norm_fac

    return bilateral_image
```

5.6 Blurring while preserving edges (1.0 points)

Copy the saved image from the Jupyter notebook here.



5.7 Cartoon Images (1 points)

Natural images can be converted to their cartoonized versions using image processing techniques. A cartoonized image can generated from a real image by enhancing the edges, and flattening the intensity variations in the original image. What operations can be used to create such images? [Hint: Try using the solutions to some of the problems covered in this homework.]

Figure 2: (a) Cartoonized version (b) Natural Image

In terms of enhancing the edges, the Prewitt filter can detect edges, which we can then intensify by increasing their contrast and thickness to make them more prominent. Instead of using a general Gaussian filter, we can look into using a bilateral or median filter that preserves the edges to some extend and smooths the rest of the colors.

6 Interview Question (Bonus) (10 points)

Consider an 256×256 image which contains a square (9×9) in its center. The pixels inside the square have intensity 255, while the remaining pixels are 0. What happens if you run a 9×9 median filter infinitely many times on the image? Justify your answer.

Assume that there is appropriate zero padding while performing median filtering so that the size of the filtered image is the same as the original image.

The final filtered image will be a the same image without the 9x9 box, in other words, an image that is purely intensity 0. This is because the median filter tends to round off corners. After infinite runs of the median filter, the center pixels will be smoothed out completely into none of the intensity 255 remains.