DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UCLA
ECE211A: DIGITAL IMAGE PROCESSING

INSTRUCTOR: Prof. Achuta Kadambi
TAS: Pradyumna Chari

NAME: Peter Racioppo
UID: 103953689

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 (lectures 1,2) and clean images (lecture 3,4), and mine images for features of interest (edges are introduced in lecture 5 and will remain 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 comprising of PDF and code: (1) CCLE: You will save this Overleaf as PDF with all the answers (and necessary work) in boxes. We would like you to use LaTeX, but will accept handwritten or tablet notes. (2) You will also submit your Jupyter notebook (.ipynb file) with all the cells executed or a Matlab script with equivalent code. The course staff will provide only limited Matlab support.

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.

Yes, by Euler's Theorem, $e^{j(\omega_1 n_1+\omega_2 n_2)} = cos(\omega_1 n_1 + \omega_2 n_2) + isin(\omega_1 n_1 + \omega_2 n_2)$.
Letting $n_2 = 0$, $e^{j(\omega_1 n_1)} = cos(\omega_1 n_1) + isin(\omega_1 n_1)$, which is clearly periodic in n_1, and likewise for n_2.

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) Let $T = [x,y,z]$. $T = [a,b,c]$, so $T[x_4] = [a,b,c]^T[5,4,3] = 5a+4b+3c$.

(ii) No, because T is a linear operator. There are two equations in three unknowns.

(iii) Yes. We still have that $x = a$ and $y = b$. Finally, we have that $x-y-z = d \Rightarrow z = a-b-d$.
Thus $T[x_4] = [a,b,a-b-d]^T[5,4,3] = 5a+4b+3(a-b-d) = 8a+b-3d$.

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) $T_1[x(n_1+\delta)] = 2x(n_1+\delta)$
Yes, the system is space invariant, because shifting the input by δ shifts the output by δ.
(ii) $T_2[x(n_1+\delta)] = x(2(n_1+\delta)) = x(2n_1+2\delta)$.
No, the system is not space invariant, because shifting the input by δ shifts the output by 2δ.

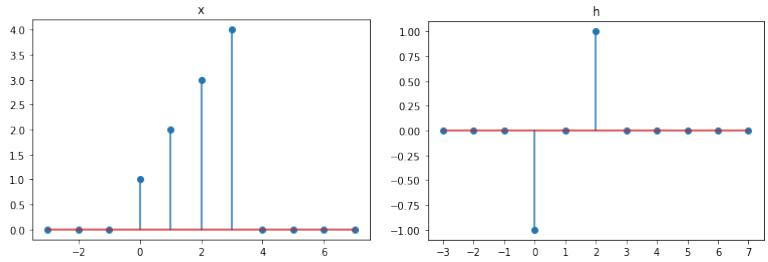


Figure 1: (a) Graphical representation of x (b) Graphical representation of h

1.4 Convolutions (4.0 points)

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) We'll use h as the kernel.
Flipping h, we have the kernel:

$$\begin{aligned} h^*(i) &= 1 \quad i = -2 \\ h^*(i) &= -1 \quad i = 0 \\ h^*(i) &= 0 \quad \forall i \notin \{0,-2\} \end{aligned} \tag{2}$$

Convolving, we have:

$$\begin{aligned} x(i) * h^*(i) &= -1 \quad i = 0 \\ x(i) * h^*(i) &= -2 \quad \forall i \in \{1,2,3\} \\ x(i) * h^*(i) &= 3 \quad i = 4 \\ x(i) * h^*(i) &= 4 \quad i = 5 \\ x(i) * h^*(i) &= 0 \quad \forall i \notin \{0,1,2,3,4,5\} \end{aligned} \tag{3}$$

(ii)

$$H = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$X^T = \begin{bmatrix} 0 & 0 & 0 & 1 & 2 & 3 & 4 & 0 & 0 & 0 \end{bmatrix}$$

$$HX = \begin{bmatrix} 0 & 0 & -1 & -2 & -2 & -2 & 3 & 4 & 0 & 0 \end{bmatrix}$$

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):
    conv = np.zeros_like(image)
    nx, ny = np.shape(image)
    kx, ky = np.shape(kernel)
    pad_x = int((kx-1)/2)
    pad_y = int((ky-1)/2)
    img_pad = np.pad(image,pad_width=((pad_x, pad_x),(pad_y,pad_y)))

    for i in np.arange(nx):
        for j in np.arange(ny):
            conv[i,j] = np.sum(kernel*img_pad[i:i+kx,j:j+ky])

    return conv
```

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.)

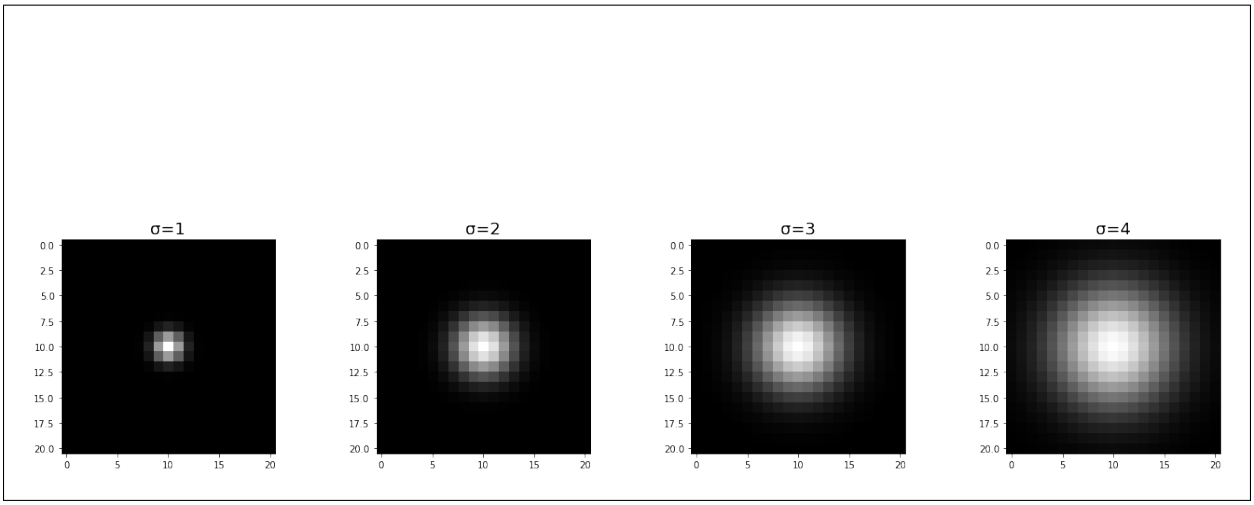
Make sure that your code is within the bounding box.

```
def gaussian_filter(size: int, sigma: float):
    mu = int((size-1)/2)
    gauss_filt = np.zeros((size,size))

    for x in np.arange(size):
        for y in np.arange(size):
            num = np.exp(-((x-mu)**2 + (y-mu)**2)/(2*sigma**2))
            denom = 2*np.pi*sigma**2
            gauss_filt[x,y] = num/denom
    gauss_filt /= np.sum(gauss_filt)

    return gauss_filt
```

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.

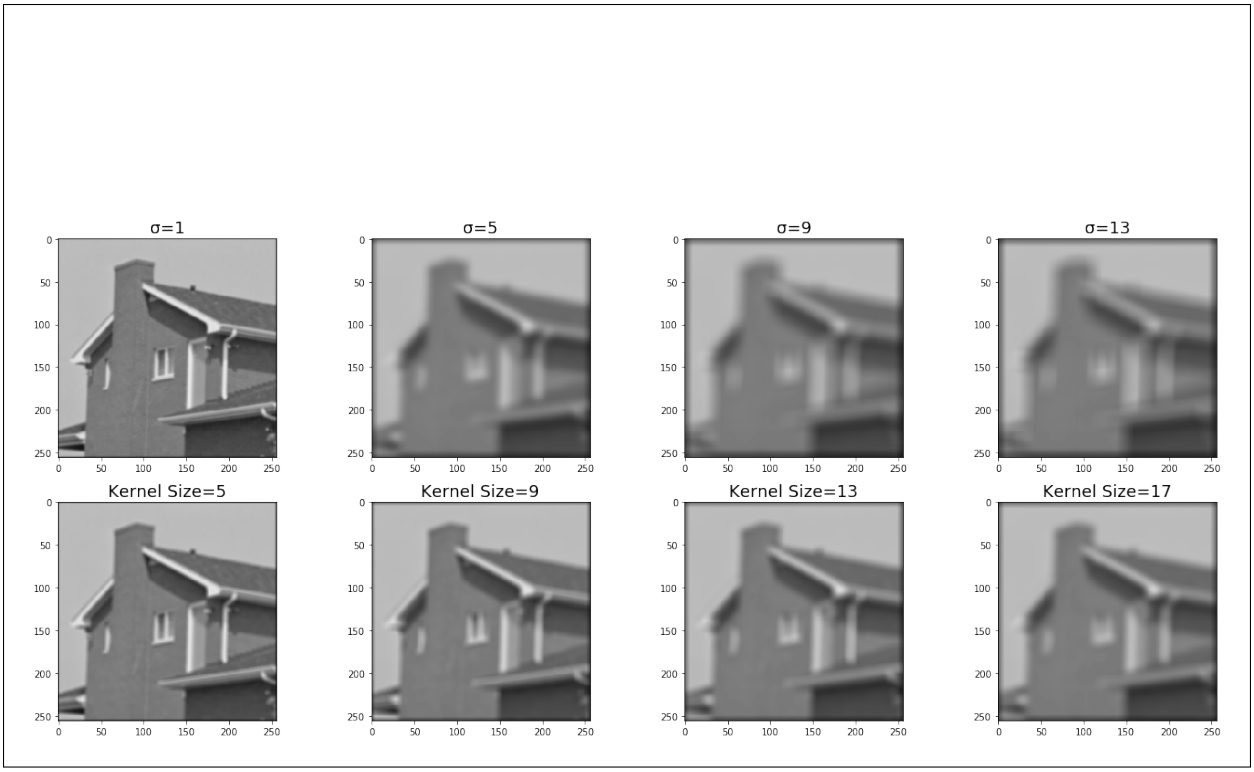
Copy the saved image from the Jupyter notebook here.



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?

By increasing the kernel size, we are doing a weighted average over a larger area of the image, which means that we decrease the sharpness of local features. By increasing σ in the Gaussian filter, we are increasing the contributions in the weighted average of pixels which are further from the center pixel. This again has the effect of increasing the contribution of more distant features as opposed to more local ones.

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.)

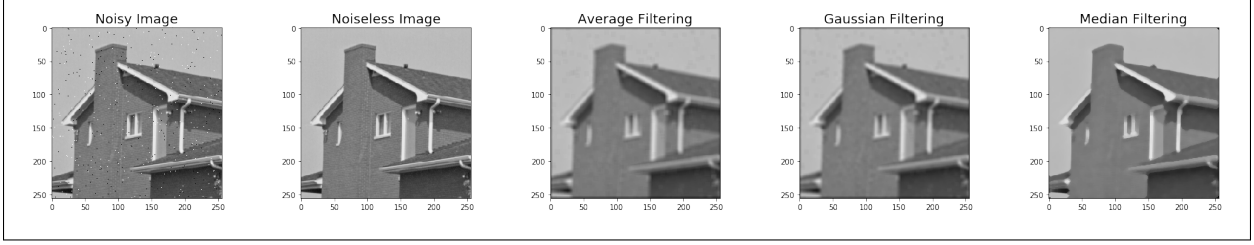
Make sure that your code is within the bounding box.

```
def median_filtering(image: np.array, kernel_size: int = None):
    med = np.zeros_like(image)
    nx, ny = np.shape(image)
    pad_x = int((kernel_size-1)/2)
    pad_y = int((kernel_size-1)/2)
    img_pad = np.pad(image,pad_width=((pad_x, pad_x),(pad_y,pad_y)))

    for i in np.arange(nx):
        for j in np.arange(ny):
            med[i,j] = np.median(img_pad[i:i+kernel_size,j:j+kernel_size])

    return med
```

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 median filter returns the sharpest image. The median operation tends to preserve sharp sharp boundaries, whereas the average and Gaussian filters blur them. The median operation also deals better with outliers, which means it is able to more efficient deal with the noise.

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 preserves edges better. Consider two regions of equal intensity separated by a straight line (e.g. a region of 0s and a region of 1s). The median kernel returns 1 when it is centered on a pixel inside the region of 1s and 0 when it is centered on a pixel in the region of 0s. In the same situation, the average and Gaussian filters would return a value between 0 and 1 for the pixels at the edge. The median filter tends to return a representative value for a pixel by something akin to majority vote. Yes, this is basically the reason the median produced a sharper image in the previous part.

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.zeros((3,3))
gradient_x[:,0] = np.ones(3)
gradient_x[:,2] = -np.ones(3)
gradient_x
```

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.)

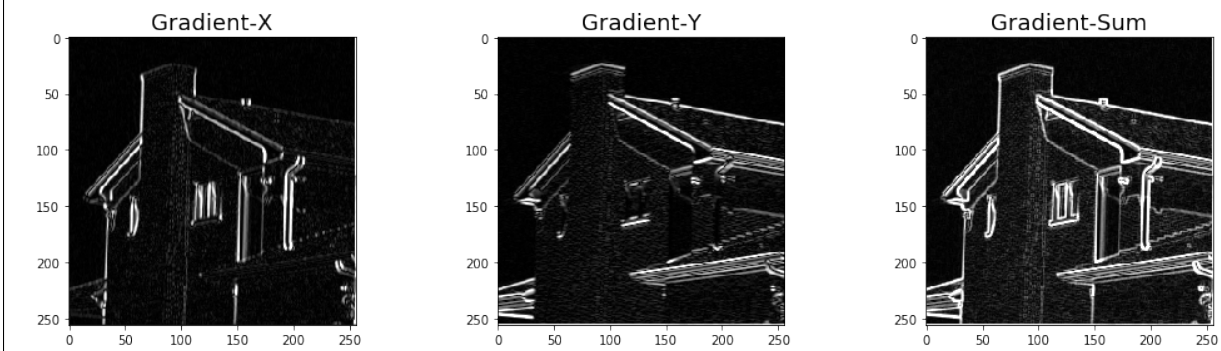
Make sure that your code is within the bounding box.

```
gradient_y = np.zeros((3,3))
gradient_y[0,:] = np.ones(3)
gradient_y[2,:] = -np.ones(3)
gradient_y
```

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?

Suppose the magnitude of the gradient at a certain pixel is g_x in the x-direction and g_y in the y-direction. Then the direction of the gradient vector is $g_\theta = arctan(g_y/g_x)$.

4.5 Separable filter (1.0 points)

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

Let $g = [1, 0, -1]$ and $a = [1, 1, 1]^T$. Then $G_x = ag$ is the horizontal gradient filter and $G_y = g^T a^T$ is the vertical gradient filter.

The sum of the horizontal and vertical filters is $G = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$, which has rank two. Thus, (since it's not rank one) it cannot be represented as an outer product and hence is not separable.

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.]

Convolution is equivalent to multiplication in the frequency domain. The Fourier Transform of a Gaussian is a Gaussian. Thus, a Gaussian filter will tend to cut off high-frequency features of an image. Edges are high-frequency features.

Another way of thinking about is simply that the Gaussian filter is a weighted average of pixel intensities in a neighborhood centered on the pixel. The weighting in the average decreases exponentially as a function of distance from the center pixel, so this is an improvement over a plain average. However, it's still a weight average and will tend to smear out sharp features.

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?

In order to avoid blurring together pixels in different features, we should take into higher consideration those pixels which have similar intensities (small values of the norm of the difference). Likewise, we might take into higher consideration pixels which have similar gradients, so as to preserve edges and high-frequency features.

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

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

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?]

We should have that $f(I_p, I_q) = 1$ when $p = q$ and f should be a non-increasing (and probably a monotonically decreasing) function of $||I_p - I_q||$. For example, we could let $f(I_p, I_q) = G(||I_p - I_q||)$, where G is a Gaussian function, or $f(I_p, I_q) = \frac{1}{1+||I_p - I_q||}$ or $f(I_p, I_q) = 1 - \sigma(||I_p - I_q||)$, where σ is a sigmoid activation, among many other possibilities.

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.

The 1D-Gaussian is monotonically decreasing in $||I_p - I_q||$ and 0 when $p = q$. The new formula is:

$$BF[I_p] = \sum_{q \in S} G_\sigma(\|p - q\|) \frac{1}{2\pi\sigma^2} \frac{exp(-||I_p - I_q||^2)}{2\sigma^2}.$$

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.)

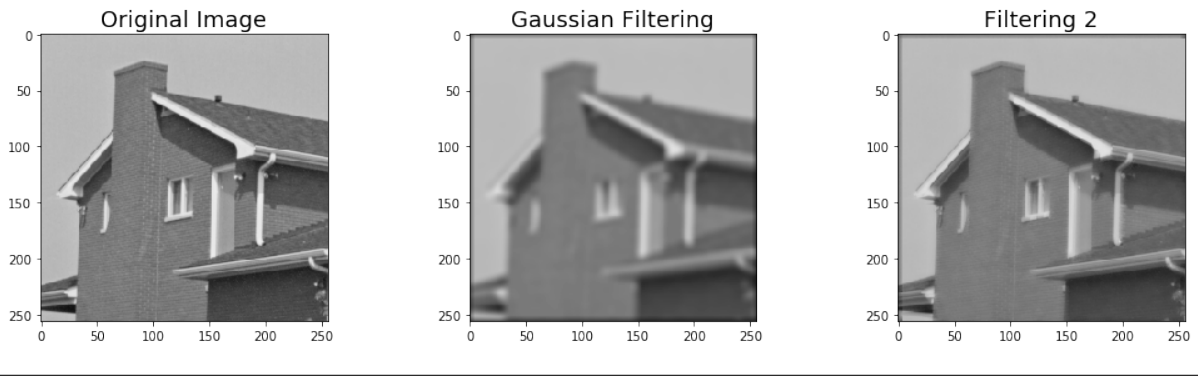
Make sure that your code is within the bounding box.

```
def filtering_2(image: np.array, kernel: np.array = None,
sigma_int: float = None, norm_fac: float = None):

    nx, ny = np.shape(image)
    kx, ky = np.shape(kernel)
    pad_x = int((kx-1)/2)
    pad_y = int((ky-1)/2)
    img_pad = np.pad(image,pad_width=((pad_x, pad_x),(pad_y,pad_y)))
    filt = np.zeros_like(image)

    for i in np.arange(nx):
```

```
        for j in np.arange(ny):
            num = np.exp(-((img_pad[i,j] - img_pad[i:i+kx,j:j+ky])**2)/
            (2*sigma_int**2)
            denom = 2*np.pi*sigma_int**2)
            gauss_I = num/denom*norm_fac
            gauss_I[pad_x,pad_y] = 1
            kernel_I = np.multiply(kernel,gauss_I)
            kernel_I /= np.sum(kernel_I)
            filt[i,j] = np.sum(kernel_I*img_pad[i:i+kx,j:j+ky])

    return filt
```

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

A filter similar to filtering 2 could also used, but with thresholding on intensities rather than a continuous Gaussian function. The median filter appears to be good at flattening out images while preserving edges. A similar idea is to use a k-nearest-neighbors type method to reduce the number of colors used. We might use the gradient filter to distinguish edges and then use the most prominent edges to divide the image into disjoint subsets, but this would probably miss a lot of detail.

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.

Consider what happens when our kernel is centered on one of the 9 pixels in the central square. If the kernel is centered on the middle square, the median filter returns the median of 9 values of 255, which is clearly 255. Likewise, if the kernel is centered on one of the 4 pixels on the exterior of the central square which are not a corner, the filter returns the median of 6 255s and 3 0s, which is again 255. However, if the kernel is centered on the corner of the central square, the filter returns the median of 5 0s and 4 255s, which is zero. On the other hand, all of the pixels which originally had value 0 remain at this value.

On the second iteration, we have a central cross. If the kernel is centered on one of the pixels on the exterior of the cross, the filter returns the median of 5 0s and 4 255s, which is 0. The only remaining 255 value is the central pixel, which the median filter changes to a 0 on its third iteration.