DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UCLA
EC ENGR 188: INTRODUCTION TO COMPUTER VISION

INSTRUCTOR: Prof. Achuta Kadambi, Prof. Stefano Soatto
TA: Zhen Wang

NAME: Anuj Agrawal
UID: 605627630

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) CCLE: You will submit your Jupyter notebook (.ipynb file) with all the cells executed on CCLE.

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.

In continuous space, the given signal is always periodic in space with a periodicity of $2\pi/\omega_o$.
In discrete space, the given signal is periodic in space only when
$\omega_o N = 2\pi m$, where n, m $\in \mathbb{Z}$,
$\omega_o/2\pi = m/n$
i.e. $\omega_o/2\pi$ is a rational number

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) Given that the system is LSI it follows,
$T[\alpha_1 x_1(n_1,n_2) + \alpha_2 x_2(n_1,n_2)] = \alpha_1 T[x_1(n_1,n_2)] + \alpha_2 T[x_2(n_1,n_2)]$
$x_4 = [5,4,3]$, can be written as a linear combination of the three basis vectors s.t.
$x_4 = [5,4,3] = 5[1,0,0] + 4[0,1,0] + 3[0,0,1] = 5x_1 + 4x_2 + 3x_3$
Therefore $T[x_4] = 5T[x_1] + 4T[x_2] + 3T[x_3] = 5a + 4b + 3c$

(ii) We won't be able to solve part (i) if $T[x_3]$ is unknown as no linear combination of basis vectors x_1 and x_2 will give x_4 since the z component of both these basis vectors are zero. Hence no linear combination of their respective system outputs (i.e. $T[x_1] and T[x_2]$) can solve for $T[x_4]$.

(iii) $x_4 = [5,4,3] = \alpha_1[1,0,0] + \alpha_2[0,1,0] + \alpha_3[1,-1,-1]$
Solving the above equation gives, $\alpha_1 = 4, \alpha_2 = 1, \alpha_3 = -3$
Therefore $T[x_4] = 4T[x_1] + 1T[x_2] - 3T[x_5] = 4a + 1b - 3d$

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) $T_2[x(n_1)] = x(2n_1)$.

For a LSI system, if $g(n_1) = T[x(n_1)]$ and $x_{n_o}(n_1) \triangleq x(n_1 - n_0)$ then
$g_{n_o}(n_1) = T[x_{n_o}(n_1)] = g(n_1 - n_o)$

(i) $T_1[x(n_1)] = 2x(n_1)$
For, $x_{n_o}(n_1)$, $T_1[x_{n_o}(n_1)] = 2x_{n_o}(n_1) = 2x(n_1 - n_o)$
Also, $g(n_1) = 2f(n_1)$, where $f(n_1) = x(n_1 - n_o)$ implies $g(n_1) = 2x(n_1 - n_o)$
as $T_1[x_{n_o}(n_1)] = g(n_1)$ the system is space invariant.

(ii) $T_2[x(n_1)] = x(2n_1)$.
For, $x_{n_o}(n_1)$, $T_1[x_{n_o}(n_1)] = x_{n_o}(2n_1 - 2n_o)$
Also, $g(n_1) = f(2n_1)$, where $f(n_1) = x(n_1 - n_o)$ implies $g(n_1) = x(2n_1 - n_o)$
as $T_1[x_{n_o}(n_1)] \neq g(n_1)$ the system is not space invariant.

1.4 Convolutions (4.0 points)

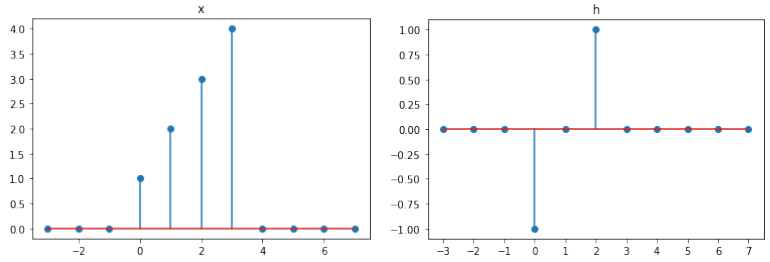


Figure 1: (a) Graphical representation of x (b) Graphical representation of h

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) y[n] = $h[n] * x[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i]$
$y[n] = \{1(-1)], [(2(-1)+1(0)], [3(-1)+2(0)+1(1)], [4(-1)+3(0)+2(1)], [4(0)+3(1)], [4(1)]\}$
$y[n] = \{-1,-2,-2,-2,3,4\}$

(ii) The given convolution of signals can bw written as a multiplication of two matrices by the Toeplitz method. Let,

$h[n] = [-1, 0, 1]$
$x[n] = [1, 2, 3, 4]$

then Toeplitz matrix of $x[n]$, $D_n(x[n]) = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \\ 4 & 3 & 2 \\ 0 & 4 & 3 \\ 0 & 0 & 4 \end{bmatrix}$

y[n] = $h[n] * x[n]$ = $D_n(x[n]).h[n]$ =$x[n].D_n(h[n])$

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

$y[n] = \{-1, -2, -2, -2, 3, 4\}$

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):
  k_shape = kernel.shape #shape of the kernel
  pad_factor = int(k_shape[0]/2) #padding each side
  i_shape = image.shape #shape of the input image
  image = np.pad(image, ((pad_factor,pad_factor),(pad_factor,pad_factor)),
                 'constant') #padded image
  i_new_shape = image.shape #shape of the new image
  conv_matrix = np.zeros(i_shape) #New convoluted image (all zeros)

  for i in range(i_new_shape[0] - k_shape[0] + 1):
    for j in range(i_new_shape[0] - k_shape[0] + 1):
      conv_matrix[i][j] = \
      np.sum(np.multiply(image[i:i+k_shape[0],j:j+k_shape[0]],kernel))

  return conv_matrix
```

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.)

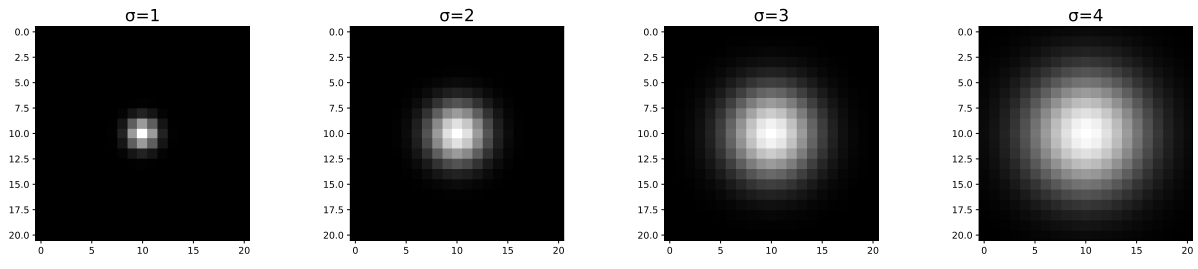
Make sure that your code is within the bounding box.

```
def gaussian_filter(size: int, sigma: float):
    kernel = np.zeros((size,size)) #empty kernel
    mx = size//2 #mean x
    my = size//2 #mean y
    c = 1 / (2*(sigma**2)*np.pi)
    for i in range(size):
        for j in range(size):
            kernel[i,j] = c*np.exp((-(i-mx)**2-(j-my)**2)/(2*(sigma**2)))
    return kernel/(np.sum(kernel))
```

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.

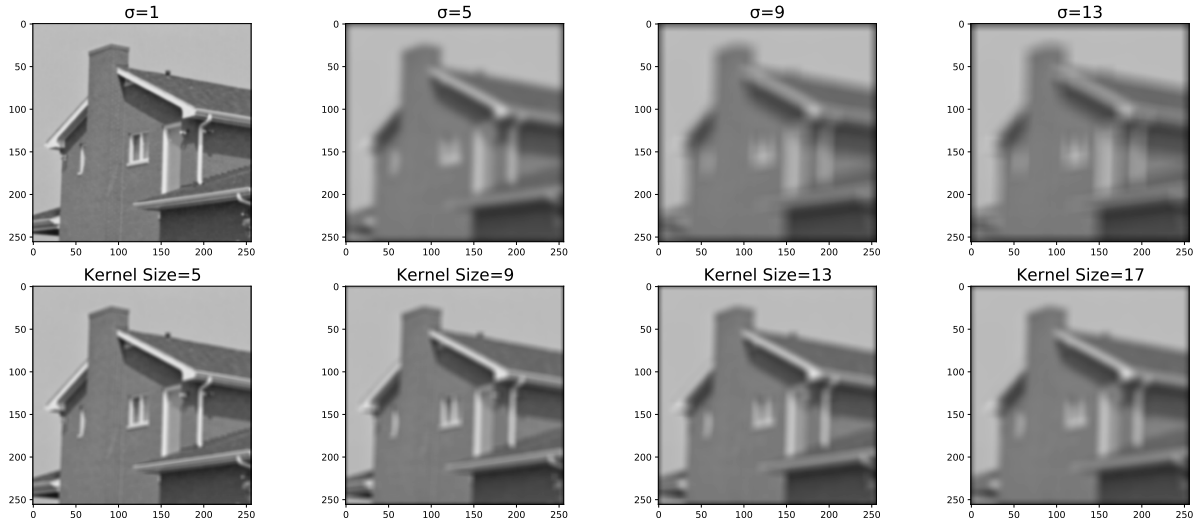
Copy the saved image from the Jupyter notebook here.



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?

(i) Change of blurring effect of kernel size
Kernel size of a Gaussian kernel controls the number of pixel around the mean values that are taken into computation while applying the filter. A larger value of the kernel size implies a bigger filter, i.e. more pixel around the mean pixel contribute to the value of the current pixel leading to an blurrier image.

(ii) Blurring effect of σ
Variance σ^2 of a Gaussian kernel controls the spread of the output of the function around the mean value. A large value of σ signifies a wider spread or more variance around the mean. Since a values father away from the mean position has a increased weightage the overall image is blurrier.

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.

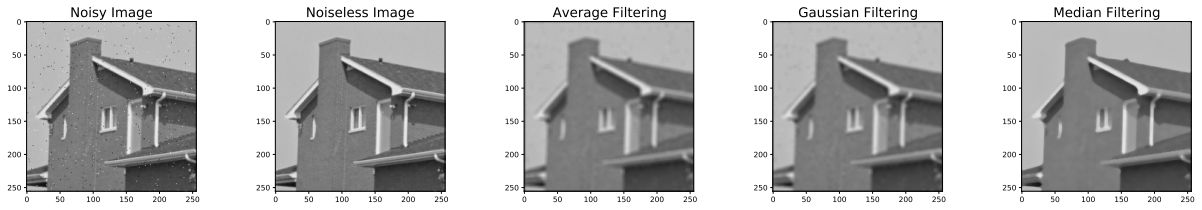
```python
def median_filtering(image: np.array, kernel_size: int = None):
  pad_factor = int(kernel_size/2) #padding each side
  i_shape = image.shape #shape of the input image
  image = np.pad(image, ((pad_factor,pad_factor),(pad_factor,pad_factor)),
                 'constant') #padded image
  i_new_shape = image.shape #shape of the new image
  output = np.zeros(i_shape) #New convoluted image (all zeros)
  for i in range(i_new_shape[0] - kernel_size + 1):
    for j in range(i_new_shape[0] - kernel_size + 1):
      output[i][j] = np.median(image[i:i+kernel_size,j:j+kernel_size])

  return output
```

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 filtering scheme gives the best result amongst all other filters because it has clearest edge and minimal blurring outputting a sharper image.
This is also validated from the relative absolute distance. Median filter has the smallest value indicating maximum closeness to original image.

Average Filtering: 0.06525493187234264
Gaussian Filtering: 0.05995105367092967
Median Filtering: 0.03332913485508449

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 the edge.

This is because unlike other filters median filter does not act like a low pass filter. Noise is generally assumed to be an outlier and median filter finds the mid most element (in terms of pixel intensity) for a given kernel. Preserving the edges of the image and weeding out the noise.

This aligns with the previous answer.

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.matmul(np.array([[1, 1, 1]]).transpose(),
                               np.array([[1, 0, -1]]))
```

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.)

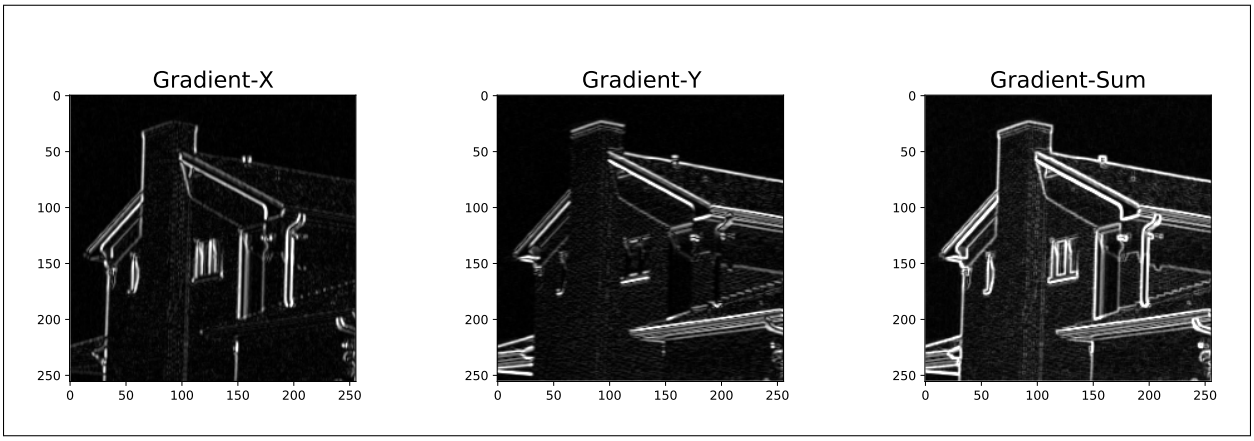
Make sure that your code is within the bounding box.

```
gradient_y = np.matmul(np.array([[1, 0, -1]]).transpose(),
                               np.array([[1, 1, 1]]))
```

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?

The gradient direction at each pixel in a image in calculated from the following formula:
$g[x,y] = \tan^{-1}(S_y/S_x)$
Where S_y is the gradient in y direction and S_x is the gradient in x direction at point (x,y).

4.5 Separable filter (1.0 points)

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

yes,

$$H_x^P = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

$$H_y^P = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

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.]

Gaussian filtering/denoising on a image has a blurring or smoothing effect due to its frequency response. It acts at a low pass filter that attenuates high spatial frequency components from a image. This helps in image denoising as noise are random outlier point in a image which causes high frequency change locally. On the other hand, edge in a images are regions of high frequency as there is rapid change in spatial properties just across the edge. While applying a Gaussian filter (low pass filter) the localization properties of the edge is not preserved.

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?

A filter should be such that the localization information of an image should be preserved. Two factors to be taken in account for designing such a filter is:
(i) Distance between the pixel p and q , i.e. $(\|p-q\|)$
(ii) Intensity difference between the pixel at p and q, i.e. $(\|I_p - I_q\|)$

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

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

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?]

$f(I_p, I_q)$ can be a function that weigh outs /removes image pixels whose values differ drastically from the central pixel while denoising the images. This is a data dependent operation that measures the similarity in intensity values to the middle pixels. Such a function can be the be Gaussian filter with mean around the central pixel.

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

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.

Yes, a 1D Gaussian filter will satisfy the following criteria as it will softly reject pixels with higher intensity difference.

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

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):

  k_shape = kernel.shape #shape of the kernel
  pad_factor = int(k_shape[0]/2) #padding each side
  i_shape = image.shape #shape of the input image
  image = np.pad(image, ((pad_factor,pad_factor),(pad_factor,pad_factor)),
                 'constant') #padded image
  i_new_shape = image.shape #shape of the new image
  conv_matrix = np.zeros(i_shape) #New convoluted image (all zeros)

  for i in range(i_new_shape[0] - k_shape[0] + 1):
    for j in range(i_new_shape[0] - k_shape[0] + 1):
```

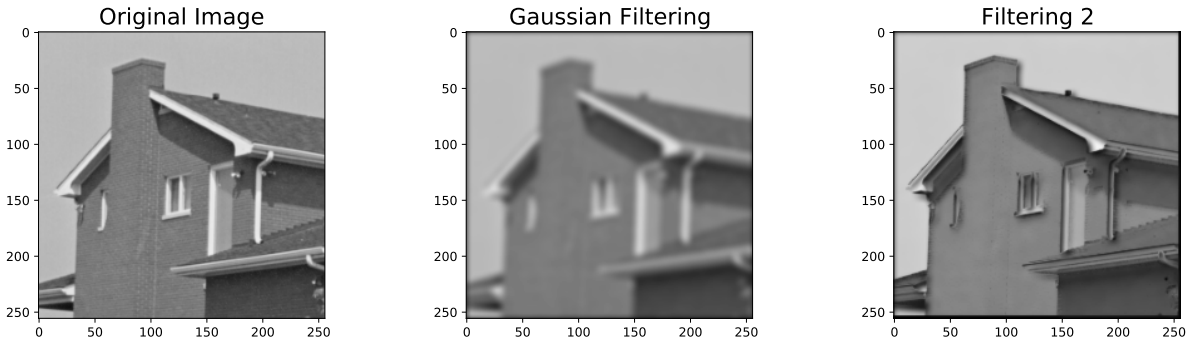
```
      image_sec = image[i:i+k_shape[0],j:j+k_shape[0]]
      d_guass = np.zeros(image_sec.shape)
      mxy = image_sec[size//2, size//2]
      c = 1 / (2*(sigma_int**2)*np.pi)
      # c = 1
      for idx, ixy in np.ndenumerate(image_sec):
        d_guass[idx] = c * np.exp(-((ixy - mxy)**2)/(2*(sigma_int**2)))

      conv_matrix[i][j] = \
      np.sum(np.multiply(np.multiply(d_guass,kernel),image_sec)) * norm_fac

  return conv_matrix
```

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

Cartooniztion can be achieved through the following sequence of operation:
(i) Bilateral Filtering : It will smoothen out the noise in the image while maintaining sharpness of the edges.
(ii) Median Filtering : It will remove outlier noise and further make the image intensity variation uniform.
(iii) Canny Edge Detection: It will be able to detect prominent edges in the images which can be used to enhance the edge pixels to give sharper boundaries.

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 median filter works by returning the median element (in terms of pixel intensity) of all the pixels of the image overlapping with the kernel. While performing the median filtering over a binary image of the square, the filter will only return a value of 255 if at least 50% of intensity values overlapping with the kernel have a value of 255. In case of a 9x9 kernel this value is at least 41. **When you apply the filter over the square image, the square will start diminishing from the outermost pixel (i.e. vertices) to the innermost pixel (i.e. center) after each iteration in a symmetric fashion till the square vanishes.** The inward trend is due to the fact the the vertex pixels have the least surrounding pixels with value 255 (i.e. 24) while the center pixel has the most (i.e 80).