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

INSTRUCTOR: Prof. Achuta Kadambi
TA: Howard Zhang

NAME: Yaman Yucel
UID: 605704529

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.

$x(n_1,n_2) = \exp(j(\omega_1 n_1 + \omega_2 n_2))$
$x(n_1+\Delta n_1, n_2+\Delta n_2) = \exp(j(\omega_1(n_1+\Delta n_1) + \omega_2(n_2+\Delta n_2)))$
$x(n_1+\Delta n_1, n_2+\Delta n_2) = \exp(j\omega_1 n_1)\exp(j\omega_2 n_2)\exp(j\omega_1\Delta n_1)\exp(j\omega_2\Delta n_2)$
$x(n_1,n_2) = x(n_1+\Delta n_1, n_2+\Delta n_2)$ if $\omega_1\Delta n_1 = 2\pi k_1$ and $\omega_2\Delta n_2 = 2\pi k_2$
since $\Delta n_1, \Delta n_2, k_1, k_2$ are integers, ω_1 and ω_2 should be integer multiples of 2π
Therefore, 2D complex exponential is not periodic in space.

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) $x_4 = 5x_1 + 4x_2 + 3x_3 \Rightarrow T[x_4] = T[5x_1 + 4x_2 + 3x_3]$
$T[x_4] = 5T[x_1] + 4T[x_2] + 3T[x_3] \Rightarrow T[x_4] = 5a + 4b + 3c$
(ii) $T[x_4] = 5T[x_1] + 4T[x_2] + 3T[x_3] \Rightarrow T[x_4] = 5a + 4b + 3T[x_3]$
There will be 2 unknown but one equation $\Rightarrow T[x_4]$ is not uniquely solvable.
(iii) $x_5 = x_1 - x_2 - x_3 \Rightarrow x_3 = x_1 - x_2 - x_5$
$T[x_4] = T[5x_1 + 4x_2 + 3x_3] \Rightarrow T[x_4] = T[5x_1 + 4x_2 + 3(x_1 - x_2 - x_5)]$
$T[x_4] = T[8x_1 + x_2 - 3x_5)] \Rightarrow T[x_4] = 8a + b - 3d$

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)
$T_1[x_1(n_1)] = 2x_1(n_1)$
$x_1(n_1 + k) = x_2(n_1)$

$T_1[x_2(n_1)] = 2x_2(n_1) = 2x_1(n_1+k)$
i is space invariant.
(ii)
$T_2[x_1(n_1)] = x_1(2n_1)$
$x_1(n_1+k) = x_2(n_1)$
$T_2[x_2(n_1)] = x_2(2n_1) = x_1(2n_1+2k) \neq x_1(2n_1+k)$
ii is space variant.

1.4 Convolutions (4.0 points)

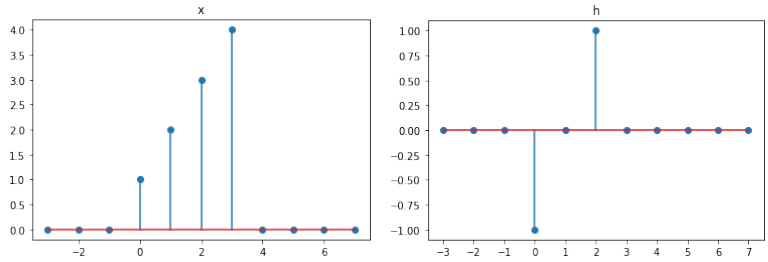


Figure 1: (a) Graphical representation of x (b) Graphical representation of h

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.

Although regular convolution can be done, a simple method can be used to compute y.
$y[n] = x[n] * h[n] = x[n] * (-\delta[n] + \delta[n-2])$
$y[n] = x[n] * (-\delta[n]) + x[n] * \delta[n-2] = -x[n] + x[n-2]$
(i)
$y[0] = x[-2]h[2] + x[0]h[0] = -1$
$y[1] = x[-1]h[2] + x[1]h[0] = -2$
$y[2] = x[0]h[2] + x[2]h[0] = -2$
$y[3] = x[1]h[2] + x[3]h[0] = -2$
$y[4] = x[2]h[2] + x[4]h[0] = 3$
$y[5] = x[3]h[2] + x[5]h[0] = 4$

(ii) $X = \begin{bmatrix} x[0] & x[-2] \\ x[1] & x[-1] \\ x[2] & x[0] \\ x[3] & x[1] \\ x[4] & x[2] \\ x[5] & x[3] \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 3 & 1 \\ 4 & 2 \\ 0 & 3 \\ 0 & 4 \end{bmatrix}, H = \begin{bmatrix} h[0] \\ h[2] \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

$$x * h \Rightarrow XH = Y = \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ -2 \\ -2 \\ 3 \\ 4 \end{bmatrix}$$

or one can also write them as $HX = Y$

$$H = \begin{bmatrix} h[0] & h[-1] & h[-2] & h[-3] \\ h[1] & h[0] & h[-1] & h[-2] \\ h[2] & h[1] & h[0] & h[-1] \\ h[3] & h[2] & h[1] & h[0] \\ h[4] & h[3] & h[2] & h[1] \\ h[5] & h[4] & h[3] & h[2] \end{bmatrix}, X = \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

$$x * h \Rightarrow HX = Y = \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ -2 \\ -2 \\ 3 \\ 4 \end{bmatrix}$$

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):
    m = kernel.shape[0]
    n, k = image.shape
    padded_image = np.pad(image, pad_width = [(m//2, m//2)])
    new_image = np.zeros((n,k))
    for i in range(n):
        for j in range(k):
            window = padded_image[i:i+ m, j:j+m]
            new_image[i][j] = np.sum(window * kernel)
    return new_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.)

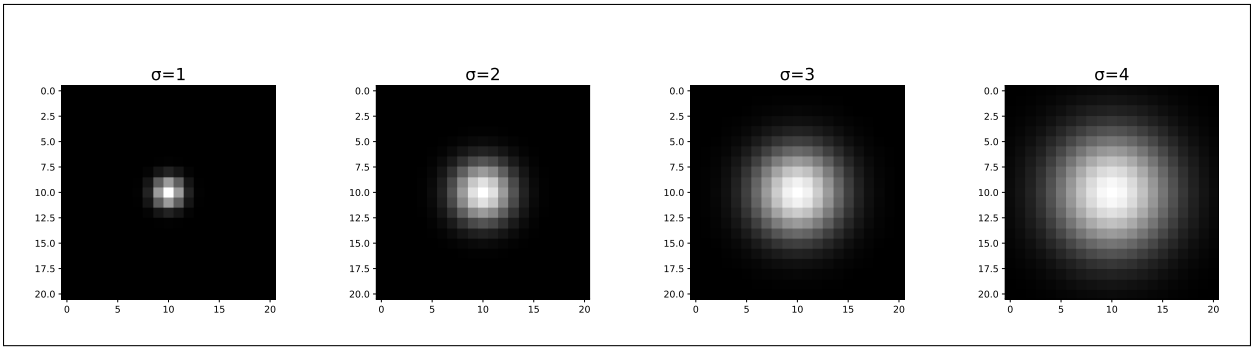
Make sure that your code is within the bounding box.

```
def gaussian_filter(size: int, sigma: float):
    assert size%2 == 1
    ax1 = np.linspace(-(size - 1) / 2., (size - 1) / 2., size)
    ax2 = np.linspace(-(size - 1) / 2., (size - 1) / 2., size)
    gauss2d = np.zeros((size,size))
    for i in range(size):
        for j in range(size):
            gauss2d[i,j] = np.exp(-(ax1[i]*ax1[i]+ax2[j]*ax2[j])/(2*sigma**2))
    return gauss2d / np.sum(gauss2d)
```

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.

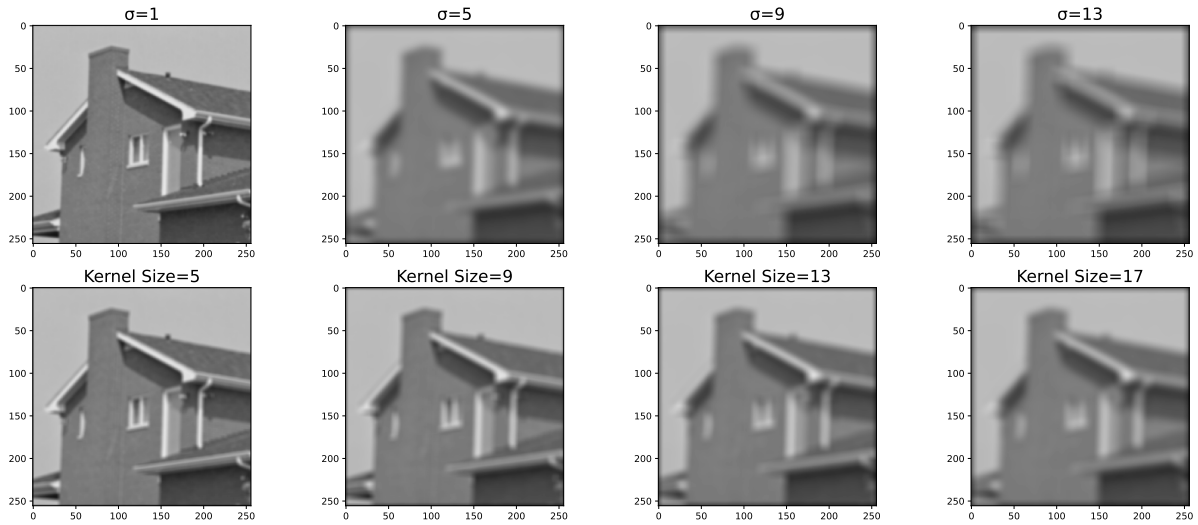
Copy the saved image from the Jupyter notebook here.



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: Larger kernel size means that more neighboring pixels are used in the each step of convolution, which yields to a more blurring effect.

σ: Higher σ means that the weights are more spread out, therefore neighboring pixels contribute more to the convolution. In result, higher σ yields to a more 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.

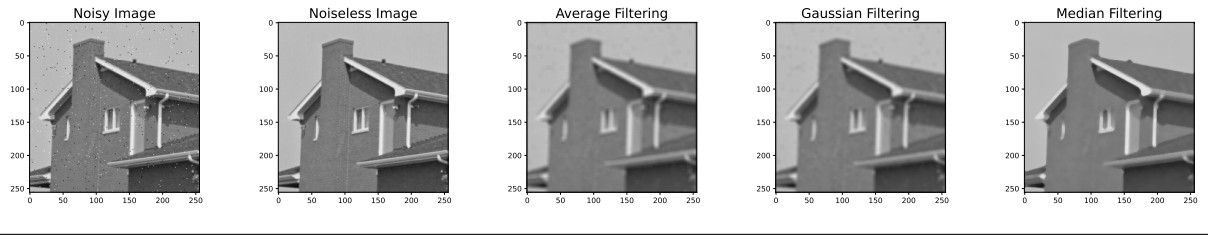
```
def median_filtering(image: np.array, kernel_size: int = None):
    m = kernel_size
    n, k = image.shape
```

```
    padded_image = np.pad(image, pad_width = [m//2, m//2])
    new_image = np.zeros((n,k))
    for i in range(n):
        for j in range(k):
            window = padded_image[i:i+m, j:j+m]
            new_image[i][j] = np.median(window)
    return new_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?

Best Filter: Median Filtering
Why: Median Filtering works best for the salt and pepper noise, since it preserves sharpness and edges, which were distorted by average and Gaussian filtering. Salt and pepper noise introduces outliers to the image which can be easily eliminated by taking the median of values. Since image values are nearly continuous, median filtering does not disrupt the image as much as average and Gaussian filtering. However, note that median filtering is nonlinear, therefore it is a computationally heavy algorithm and also blurs the image. Also, relative absolute difference of median filtering is much lower than other filtering methods.
Average Filtering: 0.06525493187234264
Gaussian Filtering: 0.05995105367092965
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?

Median filter preserves edges better, because unlike Gaussian filters it does not blur the edges. Median value is insensitive to the extreme values, therefore median filtering only eliminates the outliers. Gaussian and average filters compute a weighted average of the neighboring pixels, which yields to a blur and loss of edges. In median filtering, corners are rounded, but line edges are preserved. Yes, it aligns with 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.array([[1,0,-1],[1,0,-1],[1,0,-1]])
#Prewitt used , you can also use sobel (10-1 20-2 10-1)
#I have used the wikipedia notation, not the slides notation,
#therefore filter is flipped.
```

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.)

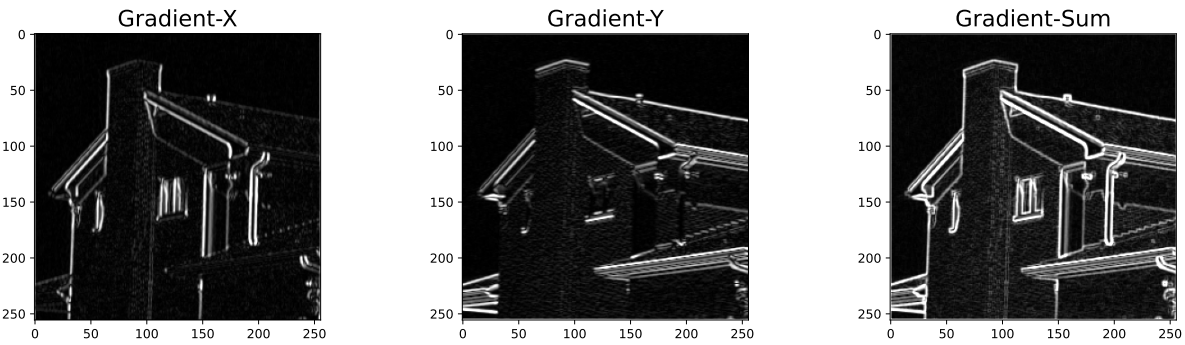
Make sure that your code is within the bounding box.

```
gradient_y = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
#Prewitt used , you can also use sobel (-1-2-1 000 121)
#I have used the wikipedia notation, not the slides notation,
#therefore filter is flipped.
```

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 convolving with Sobel or Prewitt operator, one can use vertical and horizontal gradients to approximate gradient direction and magnitude.
Gradient magnitude = $\sqrt{dx^2 + dy^2}$
Gradient direction $(\theta) = \arctan(\frac{dy}{dx})$
where dy,dx are the pixel's gradient found with edge detector filter.

4.5 Separable filter (1.0 points)

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

Yes, gradient filter is separable. I have used the wikipedia notation, not the slides notation, therefore filter is flipped.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = yx^T$$

$$G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = xy^T$$

Also, they can be written as convolution of 1D filters.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = y * x$$

$$G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = y * x$$

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.]

Fourier transform of a centered Gaussian filter is a centered Gaussian signal with inverted amplitude and standard deviation. Convolution in spatial domain is multiplication in frequency domain, therefore frequencies of an image are multiplied with a gaussian signal centered at the origin. In result of the multiplication, as Gaussian signal diminishes when input is far away from the mean, high frequencies are multiplied with small values, whereas low frequencies are multiplied with high values. Edges contain the high frequency, therefore edges can not be preserved with direct Gaussian filtering.

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?

Factor 1) Similarity in intensity between pixel p and q
Explanation: The weight for the pixel q that has similar intensity of pixel p should be higher to preserve edges in the image because pixels that make the same edge have similar intensity. If a neighbor point has significantly different intensity, that point is not part of the edge. Therefore, that neighbor should get a small weight, whereas points that have similar intensity with p should get a higher weight.
Factor 2) Spatial distance between pixel p and pixel q.
Explanation: The weight for pixel q that is nearer to pixel p should be higher to preserved edges in the image because edges are typically close to each other in the image. Closer pixels should get high weight, whereas far away pixels should get low weight.

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?]

$f(I_p, I_q)$ should give high values only when I_p and I_q are similar to each other. When I_p and I_q are very different from each other, function should give small values. Therefore following candidate functions can be used. Note that, these functions do not have a constant term in front of them since we normalize the multiplied kernel.
1) Gaussian function
$f(x,y) = \exp\left(\frac{-(x-y)^2}{2\sigma^2}\right)$
You can also control the decay with σ
2) Exponential function
$f(x,y) = \exp\left(\frac{-|x-y|}{\sigma}\right)$
This function is also similar to Gaussian function as difference between intensities are used.
3) Truncated linear function:
$f(x,y) = 1 - \frac{|x-y|}{T}$
This function is also similar to Gaussian function as difference between intensities are used, but difference is not penalized with an exponential function. T determines the the maximum distance between the intensity values for which the kernel will have non-zero weight.
4) Huber function:
$f(x,y) = \exp\left(\frac{-(x-y)^2}{2\sigma^2}\right)\left(1 - \frac{|x-y|}{T}\right)$
This function is a combination of 1 and 3. T determines the the maximum distance between the intensity values for which the kernel will have non-zero weight.

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.

$G_r\big(I_p - I_q\big) = \exp\left(\frac{-(I_p-I_q)^2}{2\sigma^2}\right)$ satisfies the property of giving high values when I_p is similar to I_q, therefore gaussian function is suitable.
$BF[I_p] = \sum_{q \in S} G_\sigma\big(\|p-q\|\big) G_r\big(I_p - I_q\big) I_q$ where $G_r\big(I_p - I_q\big) = \exp\left(\frac{-(I_p-I_q)^2}{2\sigma^2}\right)$
One should also add the normalization factor.
$BF[I_p] = \frac{1}{W_p} \sum_{q \in S} G_\sigma\big(\|p-q\|\big) G_r\big(I_p - I_q\big) I_q.$

$W_p = \sum_{q \in S} G_\sigma(\|p-q\|) G_r(I_p - I_q)$
where $G_r(I_p - I_q) = \exp\left(\frac{-(I_p - I_q)^2}{2\sigma^2}\right)$

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.

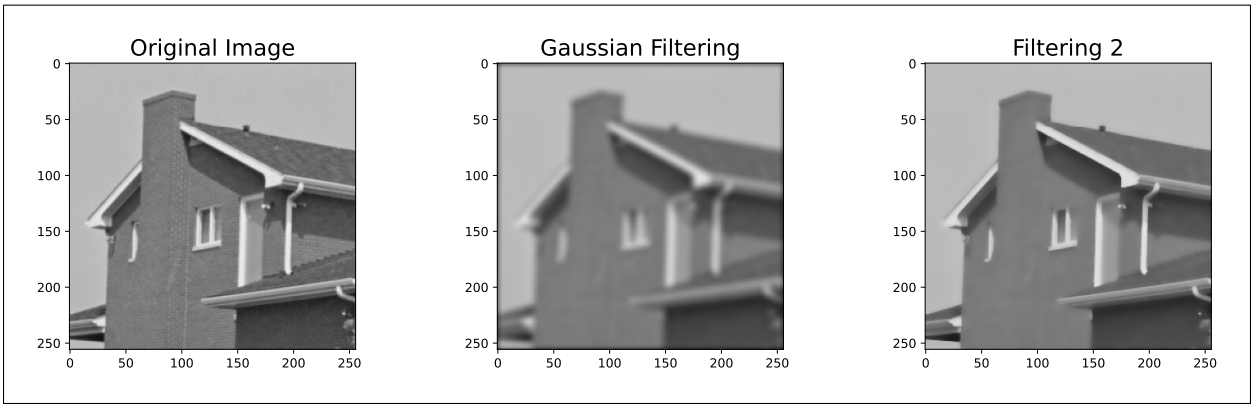
```
def filtering_2(image: np.array, kernel: np.array = None, sigma_int: float = None, norm
    m = kernel.shape[0]
    padded_image = np.pad(image, pad_width = [m//2, m//2])
    filtered_image = np.zeros_like(image)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            window = padded_image[i:i+m, j:j+m]
            #const = (1 / (sigma_int * np.sqrt(2 * np.pi)))
            i_kernel =  np.exp(-(window - image[i, j])**2/(2*sigma_int**2))
            kernel_bi = i_kernel * kernel
            filtered_image[i, j] = np.sum(kernel_bi*window)/np.sum(kernel_bi)

    return filtered_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

Below techniques can be combined to generate a cartoonized version of an image
1) Edge Detection:
With edge detection one can identify the borders of cartoonification. It can be easily see that the edges define the change in colors in a cartoonized image
2) Color quantization:
It can be easily seen that the number of colors are significantly reduced in the cartoonized image. Therefore, using k-means clustering or median cut algorithm one can group similar colors together.
3) Bilateral filtering:
As we have observed above, cartoonized image requires preserved edges, however other fea-

tures are completely blurred. Therefore, bilateral filtering is a perfect fit for cartoonification. A larger window might be preferred since we require a highly blurred image, this technique is also called as non-local means filtering.
4) Threshold:
Using threshold foreground can be separated from background therefore, one might easily focus on foreground after elimination of foreground.
5) Median Filtering:
To eliminate outliers or different intensities in the image after several processing, one can use median filtering to obtain perfectly cartoonized image.

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.

It becomes a zero matrix after step 3. At the first step, edges of the square become 0, resulting a diamond shaped 255 matrix. Then, 5x5 square with 0 values at the corners is obtained at the second step. In third step, all values are set to 0. After, zeroing all intensities, it is redundant to perform median filtering as median filtering will produce the same output in each step. Note that we can not center a square on a even-sized image, but answer does not change even square is not perfectly centered.