Department of Electrical and Computer Engineering, UCLA
EC ENGR 188: The Foundations of Computer Vision

Instructor: Prof. Achuta Kadambi
TA: Rishi Upadhyay

Name: Benjamin Cruz
UID: XXX-XXX-XXX

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 here: `https://www.overleaf.com/read/wtqbbvbgqrzn#e962bb`. 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 here: `https://colab.research.google.com/drive/1Te89OU1Q6yf4cRBAYeB-YFaIPY6Wv9js?usp=sharing` (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

Submission will be done via Gradescope. You will need to submit three things: (1) this PDF with answers filled out (from Overleaf), (2) Your .ipynb file, (3) a PDF printout of your .ipynb file with all cells executed. You do not need to create a folder, you will be able to upload all three files directly to gradescope.

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.

To be periodic, we need our exponential to follow the format of

$$x(n_1+\Delta_1, n_2+\Delta_2) = x(n_1,n_2)$$

Given that $x(n_1,n_2) = \exp(j(\omega_1 n_1 + \omega_2 n_2))$. lets substitute in our Δ_1 and Δ_2

$$x(n_1+\Delta_1, n_2+\Delta_2) = \exp(j(\omega_1(n_1+\Delta_1) + \omega_2(n_2+\Delta_2)))$$

$$x(n_1+\Delta_1, n_2+\Delta_2) = \exp(j(\omega_1 n_1 + \omega_1\Delta_1 + \omega_2 n_2 + \omega_2\Delta_2))$$
$$x(n_1+\Delta_1, n_2+\Delta_2) = \exp(j\omega_1 n_1)\cdot\exp(j\omega_1\Delta_1)\cdot\exp(j\omega_2 n_2)\cdot\exp(j\omega_2\Delta_2)$$

If $\Delta_1 = \frac{2\pi}{\omega_1}$ and $\Delta_2 = \frac{2\pi}{\omega_2}$ then

$\exp(j\omega_1\Delta_1) = \exp(j2\pi) = 1$ and $\exp(j\omega_2\Delta_2) = \exp(j2\pi) = 1$ and substituting that

$$x(n_1+\Delta_1, n_2+\Delta_2) = \exp(j\omega_1 n_1)\cdot 1\cdot\exp(j\omega_2 n_2)\cdot 1$$
$$x(n_1+\Delta_1, n_2+\Delta_2) = \exp(j(\omega_1 n_1 + \omega_2 n_2))$$

So our complex exponential is 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) $T[x_4]$ = 5a + 4b + 3c. Given that T is an LSI system, we know that we can use the properties of an LSI system here. We can break down x_4 into sums of x_1,x_2,x_3 and specifically $5x_1 + 4x_2 + 3x_3$. Applying our properties of a LSI system, we can get $5T[x_1] + 4T[x_2] + 3T[x_1]$.
ii) We will not be able to solve part (i). We need T[x_3] to solve for x_4.
iii) Yes! We can break down x_4 into a sum of x_1, x_2, and x_5. This will change from part (i). We need our sum to be x_4 and that will be $x_4 = 8x_1 + x_2 + (-3)x_5$ and using our properties of an LSI system, we get 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) $T_1[x(n_1)] = 2x(n_1) \Rightarrow T_1[x(n_1-k)] = 2x(n_1-k)$ This is space invariant
ii) $T_1[x(n_1)] = x(2n_1) \Rightarrow T_1[x(n_1-k)] = x(2(n_1-k)) = x(2n_1-2k)$ This is not space invariant

1.4 Convolutions (4.0 points)

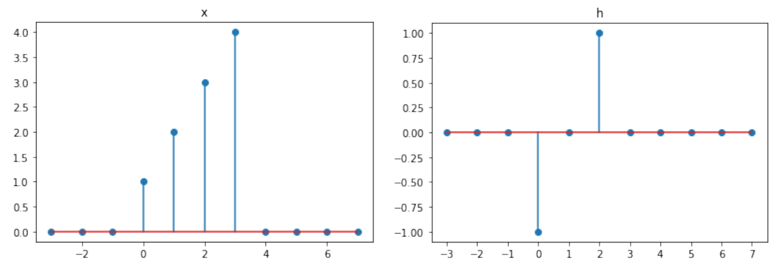


Figure 1: (a) Graphical representation of x (b) Graphical representation of h

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) $h*x(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k)$ Following this formula , we can get values for the convolution for $0 \leq n < 6$ which can represented as an array of $[-1,-2,-2,-2,3,4]$
ii) We can evaluate the non-zero part of the convolution through a product of 2 matrices by using the Toeplitz matrix. We can create a Toeplitz matrix H using the values from h(n) and padding with 0's when necessary to achieve the desired size of convolution which is a $4+3-1 = 6x4$ matrix where 4 is the size of x(n). We then take x(n) and vectorize into a size of x(n) by 1 matrix. Multiplying these two matrices will give us the non-zero parts of convolution. h(n) = [1 0 -1] and x(n) = [1 2 3 4]

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

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):
  y = np.zeros(image.shape)
  if type(kernel) == None: kernel = np.ones((3,3))
  ki = kernel.shape[0]
  kj = kernel.shape[1]
  image_pad = np.pad(image, (((ki//2), ki), ((kj//2), kj)), mode='constant')
  kernel_flip = np.flip(kernel)
  for  i in range(image.shape[0]):
    for j in range(image.shape[1]):
      y[i,j] = (image_pad[i:i+ki, j:j+kj] * kernel_flip).sum()
  return y
```

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.)

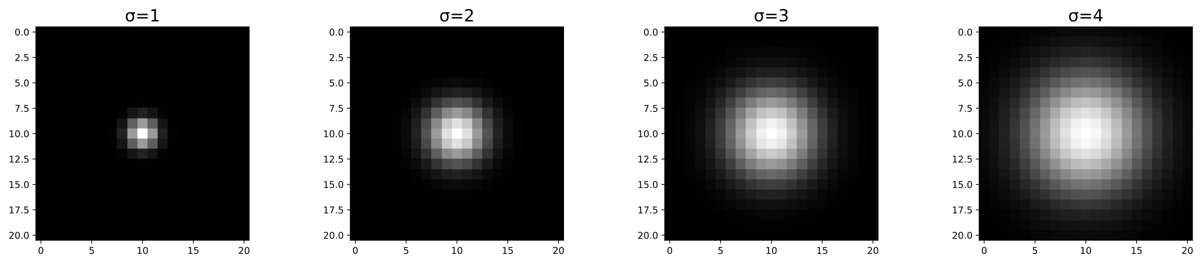
Make sure that your code is within the bounding box.

```
def gaussian_filter(size: int, sigma: float):
  gf = np.zeros((size,size))
  for x in range(size):
    for y in range(size):
      gf[x, y] = (1/(2 * np.pi * (sigma**2))) *
      np.exp(-((x-size//2)**2 + (y-size//2)**2)/(2*sigma**2))
  return gf / gf.sum()
```

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.

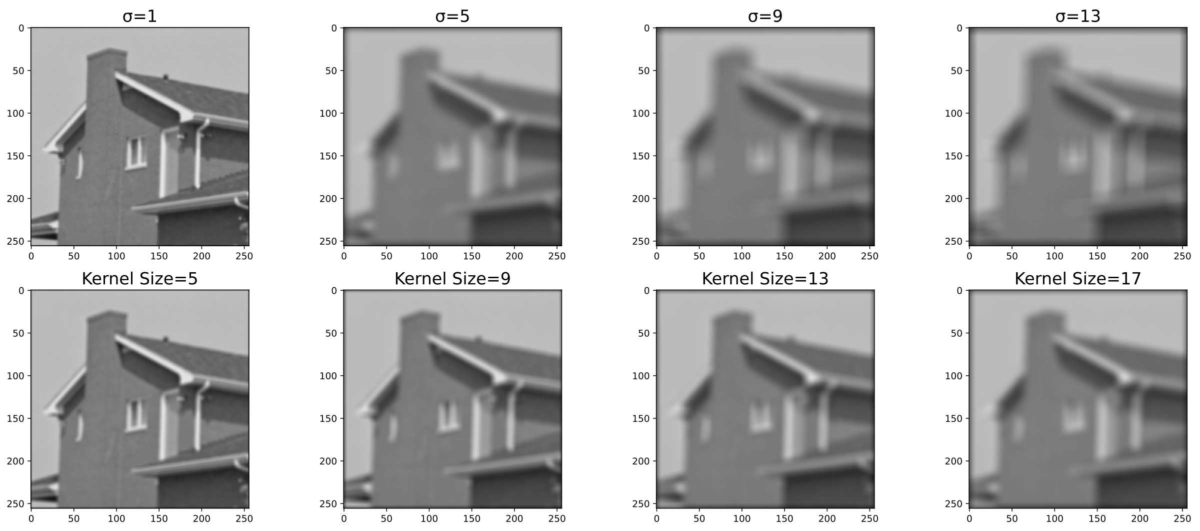
Copy the saved image from the Jupyter notebook here.



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?

The blurring effect increases with kernel size because we are averaging over a larger area which takes in more values/pixels. Increasing σ will broaden the filter and take in more values around the center which means nearby pixels have less of an impact compared to the new pixels that are now accounted for with a larger σ

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.)

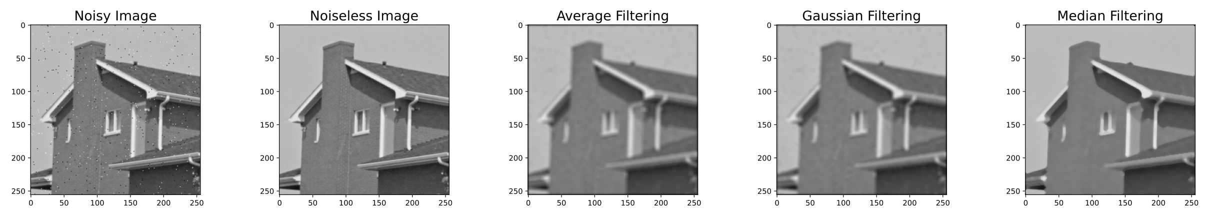
Make sure that your code is within the bounding box.

```python
def median_filtering(image: np.array, kernel_size: int = None):
 y = np.zeros(image.shape)
 if type(kernel_size) == None: kernel_size = 3
 image_pad = np.pad(image, (((kernel_size//2), kernel_size),
 ((kernel_size//2), kernel_size)), mode='constant')
 for  i in range(image.shape[0]):
   for j in range(image.shape[1]):
     y[i,j] = np.median(image_pad[i:i+kernel_size, j:j+kernel_size])
 return y
```

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?

Median filtering. It had the lowest relative absolute distance between the images compared to average and Gaussian filtering and is less affected by the noise compared to the other two filters.

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 filtering. It does this because it won't be affected when it gets near an edge. In a linear filter, this change in value would affect the filter but to the Median filter (a non linear filter), it wont be heavily affected since outliers or bigger/smaller values are sent towards the beginning/end which means they will most likely will not be the median. It does align 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.

```
x1 = np.tile([-1, 0, 1], (3,1))
x2 = np.zeros((3,3))
x2[1,1] = 1
gradient_x = conv2D(x1, x2)
```

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.)

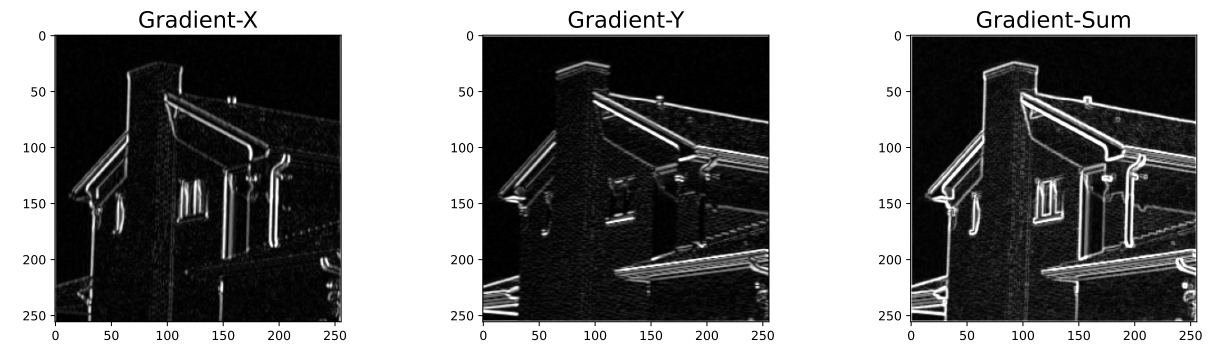
Make sure that your code is within the bounding box.

```
gradient_y = gradient_x.T
```

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?

We can compute the gradient direction this by using an atan2 function or some version of an inverse tangent function and computing the inverse tangent of the gradient-y divided by the gradient-x such as $grad_{direction} = tan^{-1} \frac{grad_y}{grad_x}$

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.

$$grad_x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}, \quad grad_y \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{3}$$

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 is a low pass filter so it captures a lot of the overall picture but the very intricate details such as edges since they are high frequency so they are 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?

Spacial distance is one factor. How far pixel q is from pixel p should decide the weight such that closer pixels should have more weight. Another factor is intensity. The higher the intensity of a pixel is should mean that higher weight is given to that pixel.

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?]

The new filter and its $f(I_p, I_q)$ should depend on the range weight. $f(I_p, I_q) = h(|I_p - I_q|)$

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.

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

5.5 Filtering (3.0 points)

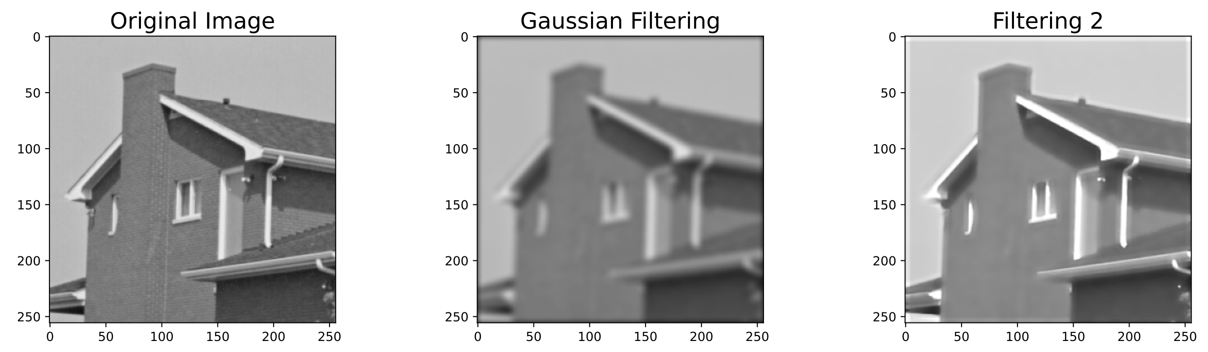
In this question you will be writing a Python function for this new filtering method (See the Jupyter notebook.)

Make sure that your code is within the bounding box.

```
def filtering_2(image: np.array, kernel: np.array = None, sigma_int: float = None, norr
  bgf = np.zeros(image.shape)
  if type(kernel) == None: kernel = np.ones((3,3))
  ki = kernel.shape[0]
  kj = kernel.shape[1]
  image_pad = np.pad(image, (((ki//2), ki), ((kj//2), kj)), mode='constant')
  kernel_flip = np.flip(kernel)
  for x in range(image.shape[0]):
    for y in range(image.shape[1]):
      L2 = image_pad[x:x+ki, y:y+kj]
      gf = (1/(2 * np.pi * (sigma_int**2)) * np.exp(
      -((L2 - L2[ki//2, kj//2])**2)/(2*sigma_int**2)))
      bgf[x, y] = (1/norm_fac) * ((gf/gf.sum()) * kernel_flip * L2).sum()
  return bgf
```

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.]

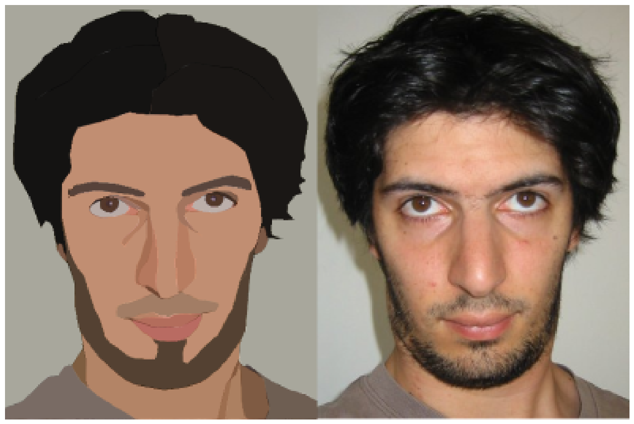


Figure 2: (a) Cartoonized version (b) Natural Image

We can use the filter we made above to achieve this effect. Comparing the houses, the bilateral filter preserves and enhances edges and uses more solid colors across the image as opposed to numerous colors (flattening the intensity).

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.

If we were to filter with a median filter infinitely many times, we will slowly lose the edges each time we filter it. Since the center is very small (9x9 compared to 256x256), the entire image will just be 0 after some number of iterations,