Introduction to Computer Vision (ECSE 415)
Assignment 1: Image Filtering

Due date: 11:59PM, October 2nd, 2020

Please submit your assignment solutions electronically via the myCourses assignment dropbox. The submission should include a single jupyter notebook. More details on the format of the submission can be found below. Submissions that do not follow the format will be penalized 10%. Attempt all parts of this assignment. The assignment will be graded out of total of **70 points**. You can use OpenCV and Numpy library functions for all parts of the assignment except stated otherwise. Students are expected to write their own code. (Academic integrity guidelines can be found at https://www.mcgill.ca/students/srr/academicrights/integrity). Assignments received up to 48 hours late will be penalized by 30%. Assignments received more than 48 hours late will not be graded.

Submission Instructions

1. Submit a single jupyter notebook consisting of the solution of the entire assignment.
2. Comment your code appropriately.
3. Do not forget to run Markdown cells.
4. Do not submit input/output images. Output images should be displayed in the jupyter notebook itself. Assume input images are kept in the same directory as the codes.
5. Make sure that the submitted code is running without error. Add a README file if required.
6. If external libraries were used in your code please specify their name and version in the README file.
7. Answers to reasoning questions should be comprehensive but concise.
8. Submissions that do not follow the format will be penalized 10%.

Note: For this assignment, we will work with grayscale images. DO NOT forget to convert your images to GrayScale from RGB images.

1 Thresholding (12 Points)

Thresholding is the simplest method of image segmentation. From a grayscale image, thresholding can be used to create binary images. Here, each pixel in an image is replaced with a foreground label (i.e. a white pixel with 255 value) if the image intensity $I_{i,j}$ is satisfies some pre-defined condition (Ex. if $I_{i,j} > T$), or with a background label (i.e. a black pixel with 0 value) otherwise.

Simple Binary Thresholding:

$$S_{i,j} = \begin{cases} 255 & \text{if } I_{i,j} > T; \quad (1) \\ 0 & \text{otherwise.} \quad (2) \end{cases}$$

Inverse Binary Thresholding:

$$S_{i,j} = \begin{cases} 255 & \text{if } I_{i,j} < T; \quad (3) \\ 0 & \text{otherwise.} \quad (4) \end{cases}$$

Window Binary Thresholding:

$$S_{i,j} = \begin{cases} 255 & \text{if } T1 > I_{i,j} > T2; \quad (5) \\ 0 & \text{otherwise.} \quad (6) \end{cases}$$

You are given an image named "numbers.jpg" (Figure 1(a)) which contains multiple different multi-digit numbers. Your task is to threshold the image using the pre-defined conditions for thresholding defined above.

Note that you are **not** allowed to use the openCV **cv2.threshold** function for this task. Instead implement thresholding using basic python or numpy functions.

1. Threshold the image at three different thresholds 1) 55 2) 90 and 3) 150 using simple binary thresholding and inverse binary thresholding as defined above. **(3 points)**
2. Write your observations about thresholded images at different thresholds. How many and which numbers are segmented at each threshold? (A number is considered as segmented if all digits of that number are considered as foreground in the thresholded image) What else do you observe at each threshold? **(3 points)**
3. Threshold the image using Window binary thresholding using three different range of thresholds. 1) T1=55 and T2=90, 2) T1=90 and T2=150, 3) T1=55 and T2=150. Write your observations. How many and which numbers are segmented at each threshold? **(3 points)**

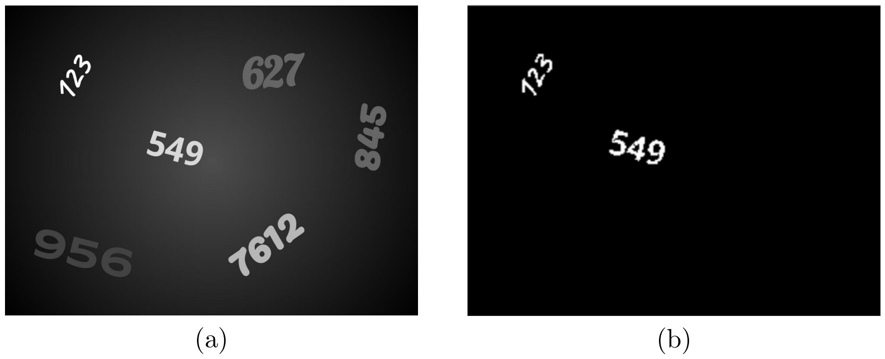


Figure 1: (a) Input image for thresholding, (b) Example of output image of thresholding. Note that only numbers ”123” and ”549” are segmented (foreground pixels).

4. In a practical application, we vary the value of the hyper-parameters (here, the threshold values) for any of the above mentioned thresholding methods, such that we get the desired output. Find a threshold value such that only numbers ”123” and ”549” are segmented (i.e. considered as foreground - white pixel - 255 value). See Figure 1(b). Report your finding for at least three different threshold values, and write how it helped you in narrowing down the desired hyper-parameter value. **(3 points)**

2 Denoising (18 Points)

You are given a clean image named ‘lighthouse’ (Figure 2(a)) and an image corrupted by additive white Gaussian noise (Figure 2(b)). You are allowed to use OpenCV/Scikit-learn functions for this section.

Apply the following filtering operations:

~~1. Filter the noisy image using a 5×5 Gaussian filter with variance equal to 2.~~ **(3 points)**

~~2. Filter the noisy image using a box filter of the same size. **(3 points)**~~

3. ~~Compare the Peak-Signal-to-Noise-Ratio (PSNR) of both of the denoised images to that of the clean image and state which method gives the superior result. (Use the PSNR function provided by opencv~~) **(3 points)**

You are also given an image corrupted by salt and pepper noise (Figure 2(c)). Apply the following filtering operations:

~~4. Filter the noisy image using the same Gaussian filter as used in the previous question. **(3 points)**~~

~~5. Filter the noisy image using a median filter of the same size. **(3 points)**~~

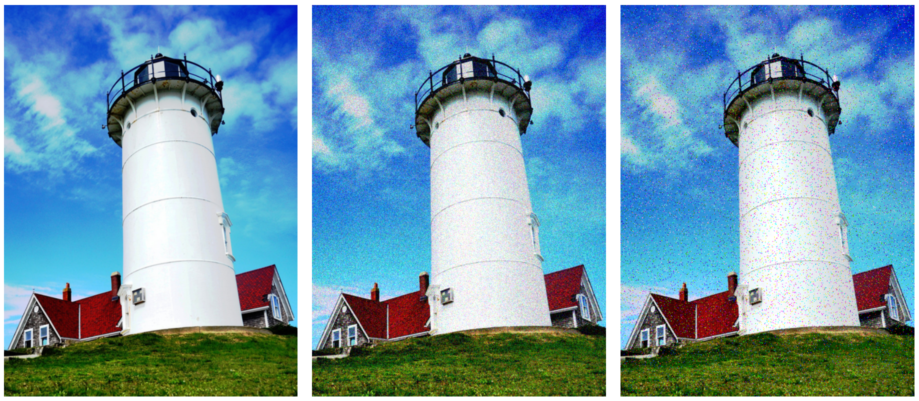

(a) (b) (c)

Figure 2: Input images for denoising. (a) clean image (b) image corrupted with Gaussian noise (c) image corrupted with salt and pepper noise.

6. ~~Compare the PSNR of both of the denoised images to that of the clean image and state which method gives a better result.~~ **(3 points)**

3 Sobel edge detector (16 Points)

In this question, you will assess the effect of varying the kernel size on the results of an edge detection algorithm. You will detect edges in a clean image named, 'cameraman' (Figure 3(a)). You are allowed to use OpenCV/Scikit-learn

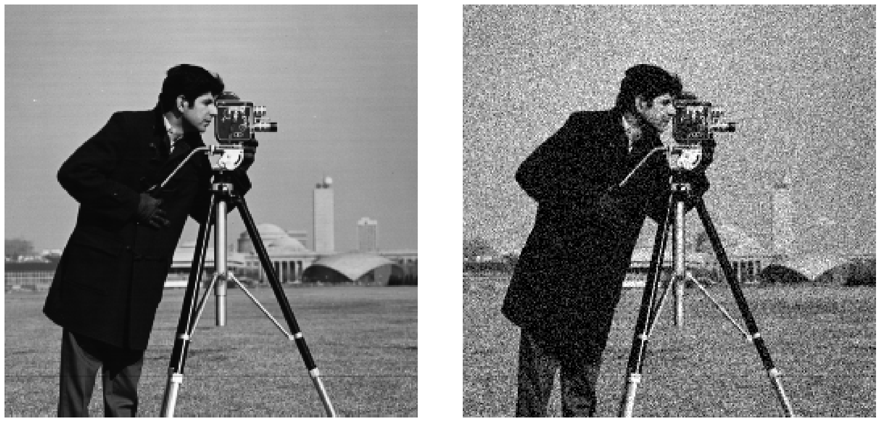

(a) (b)

Figure 3: Input image for edge detection. (a) clean image (b) image corrupted with Gaussian noise.

functions for this section.

- ~~Apply a Sobel edge detector with the kernel size of 3×3, 5×5 and 7×7 to the image. Threshold the filtered image to detect edges. Use two values of thresholds: 10% and 20% of the maximum pixel value in the filtered image.~~ ~~**(4 points)**~~
- ~~Comment on the effect of filter size on the output.~~ ~~**(2 points)**~~

Next, you will evaluate the effect of denoising prior to edge detection. For the following questions, you will use noisy image as shown in Figure 3(b).

- ~~Apply a Sobel edge detector with the kernel size of 3×3. Threshold the filtered image to detect edges. Use two values of thresholds: 10% and 20% of the maximum pixel value in the filtered image.~~ ~~**(4 points)**~~
- ~~Denoise the image with a 3×3 box filter and then apply the same Sobel edge detector, with the same values of the thresholds, from the previous question.~~ ~~**(4 points)**~~
- ~~Comment on the effectiveness of using denoising prior to edge detection.~~ **(2 points)**

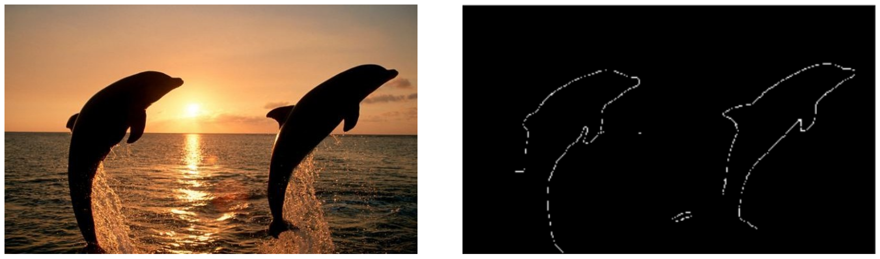

(a) (b)

Figure 4: (a) Input image for canny edge detection (b) expected output.

4 Canny Edge Detection (12 Points)

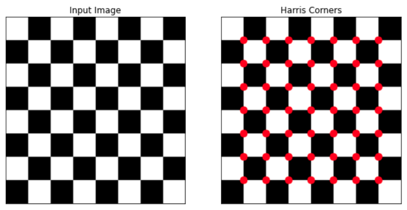
For this section, experiments will be performed on the 'dolphin.jpg' (Figure 4(a)) image.

1. Briefly describe the 4 main steps of Canny edge detection. **(2 points)**
2. As you saw in Tutorial-2, the 3 main hyperparameters of Canny Edge detection are the Gaussian Smoothing Kernel size (K), and the Lower (L) and Higher (H) Thresholds used for Hysteresis. In this section, we will

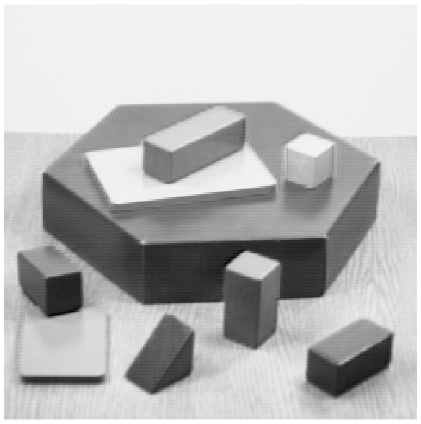
observe the effect of changing these hyperparameters. You will experiment on 3 different values for all 3 parameters (K = 5,9,13, L = 10,30,50, H = 100, 150,200). Vary the values of each hyper-parameter and keep other hyper-parameters constant. Do this procedure for all combination of hyper-parameters mentioned above. This should results in total 27 triplets of hyper-parameters. E.g. (K,L,U) = (5,10,100), (5,10,150), (9,10,200), ... Use canny edge detection (cv2.GaussianBlur and cv2.Canny) for each of these triplets. **(4 points)**

3. Comment on how changing values of each hyper-parameters (K,L,U) effects the overall edge detection. Is there is any relationship between any hyper-parameters? **(3 points)**

4. Find a value of each hyper-parameter such that only dolphin edges are detected. (Figure 4(b)) **(3 points)**



(a)



(b)

Figure 5: (a) checkerboard input image and expected harris corner output (red dots represents detecteed harris corners) (b) shapes image.

5 Harris Corner Detection (12 points)

Implement the Harris corner detector as described in class (lecture-5 slide-48 and tutorial-2) using numpy **(5 points)**. This has the following steps:

1. Compute Image derivatives (optionally, blur first)

2. Compute Square of derivatives

3. Apply Gaussian Filtering on the output of step-2

4. Get Cornerness function response (Determinant(H)-kTrace(H)2), where k=0.05. (You can vary value of k for your application)

5. Perform non-maxima suppression (as in the Canny edge detector)

You will apply Harris Corner Detector for three different images:

1. Checkerboard image Figure 5(a) (Input image). Change value of threshold to get detected corners similar to Figure (a) (Harris Corner). Observe and report affect of changing threshold values **(3 points)**

2. Shape image Figure 5(b). Try different value of thresholds and report your observations. **(2 points)**

3. Take any one face image from Google Face thumbnail collection dataset. Apply harris corner detector on this image. Try different value of thresholds and report your observations. **(2 points)**