TED UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

**Fall 2019 - CMPE 362 Digital Image Processing**

**Assignment 2**

# Spatial Filtering

by

Kaan Üstündağ

16171099958

## Introduction

In this report, I will be mentioning about the results, outcomes and explanation of the process of part 2, 3, 4, 5, and 6(bonus) and, I will be discussing the results of these parts. All the m-file implementation of these parts are included digitally.

## Part 2: Correlation of an image and a template

In this part, I used the *my_imfilter* function that I created in part 1 for 2D correlation operation. As input images I have used *figure 1* and *figure 2* images. In *figure 2,* you can see only one emoji face which we use as kernel in this part.

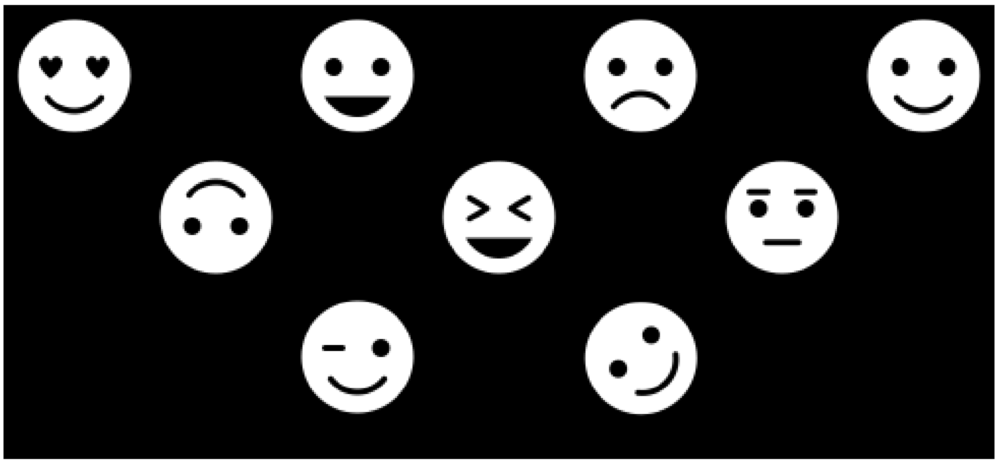

*Figure 1: emoji.png main input image*

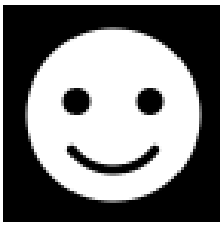

*Figure 2: smile.png kernel image source*

So, when I applied *my_imfilter* function by using these two input images, I observed blurred pixels around all the emoji's, you can see the result in *figure 3* below. We could use the results of correlation to find the exact location for the template image with the help of the cross-correlation operation. Thus, when we find the maximum peak result of the correlation result (*figure 3),* we can locate the center of the matching template. By using the center of it I draw a rectangle around the matching emoji and the result is just like in the *figure 4.*

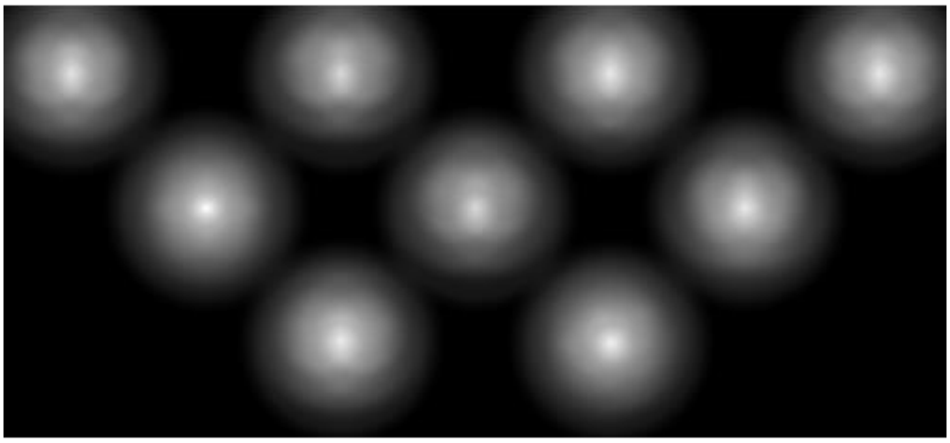

*Figure 3: Result of the correlation operation by using an image and a template*

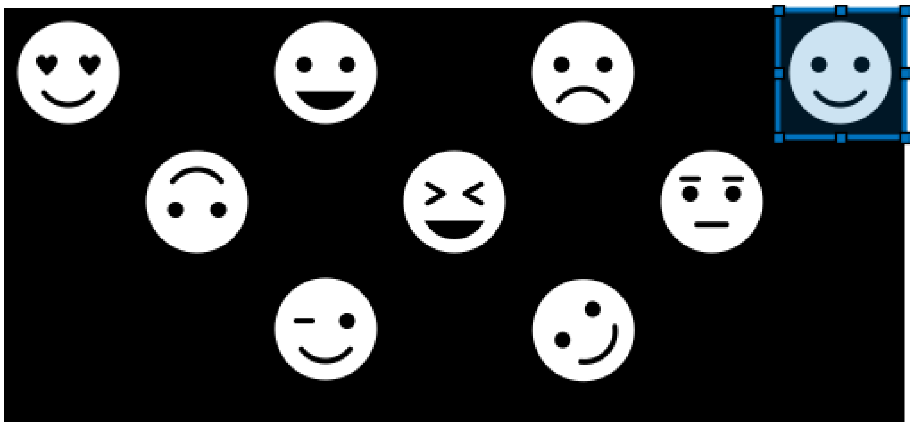

*Figure 4: I used the maximum peak as the center of the matching emoji and draw a rectangle around the matched emoji.*

Therefore, the maximum peak result lead us to the center of the location of the template image that we used to search it in the emoji.png. All in all, we can use the results of correlation to find the location of the template, by simply finding the maximum peak we also find the template location that we were looking for.

## Part 3: Comparison of Averaging filtering and Gaussian filtering

In this part, I have used *fspecial* function both with *"average"* and *"gaussian"* calls and, I have filtered all the results with *imfilter* function from MATLAB. Also, for using gaussian smoothing process I have used both 7x7 hsize (sigma value= 1.5) and 15x15 hsize (sigma value=3.5) metrics to get my result, and for the averaging process I have used 7x7 and 15x15 as hsize values. At the end I have compared my results for the same hsize values both for averaging and gaussian smoothing. The original input image that is used for this part is shown in *figure 5.*

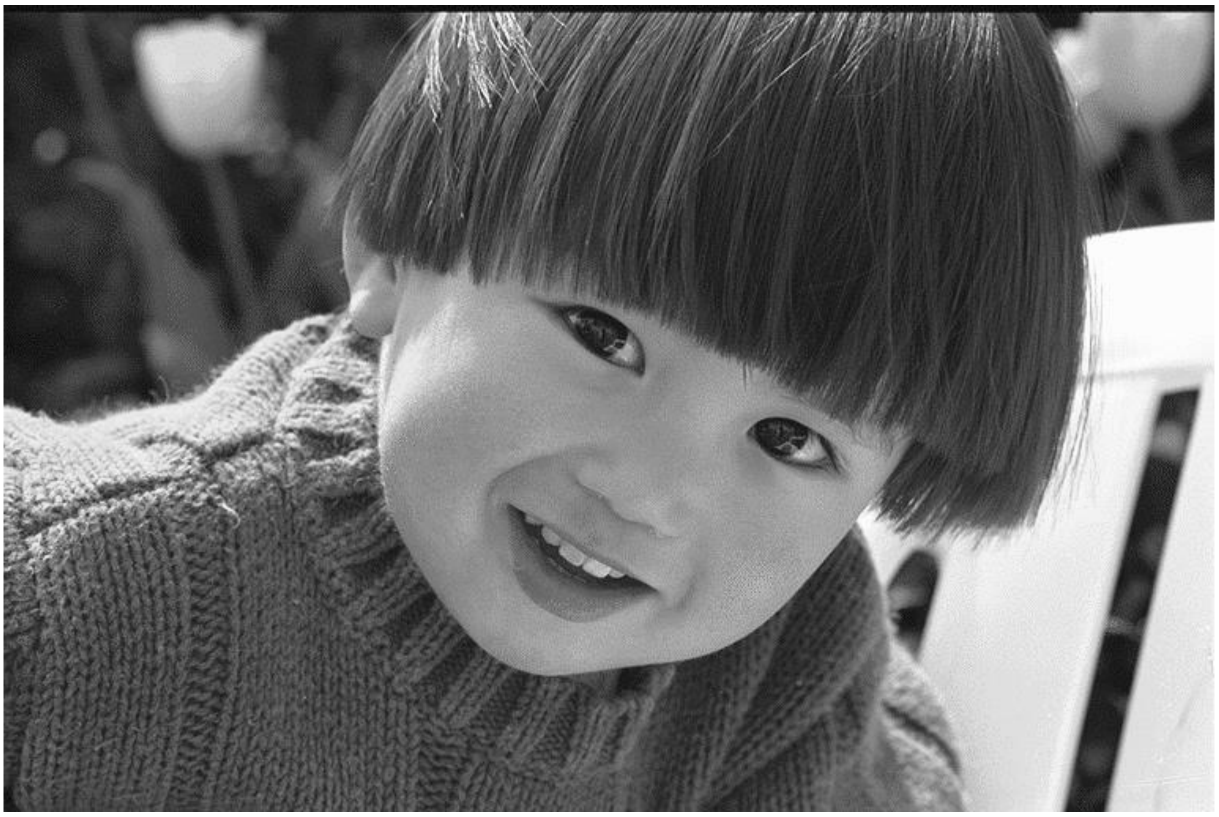

*Figure 5: boy.png*

Firstly, I used 7x7 hsize for both averaging and gaussian smoothing and observed the result that is shown in *figure 6.*

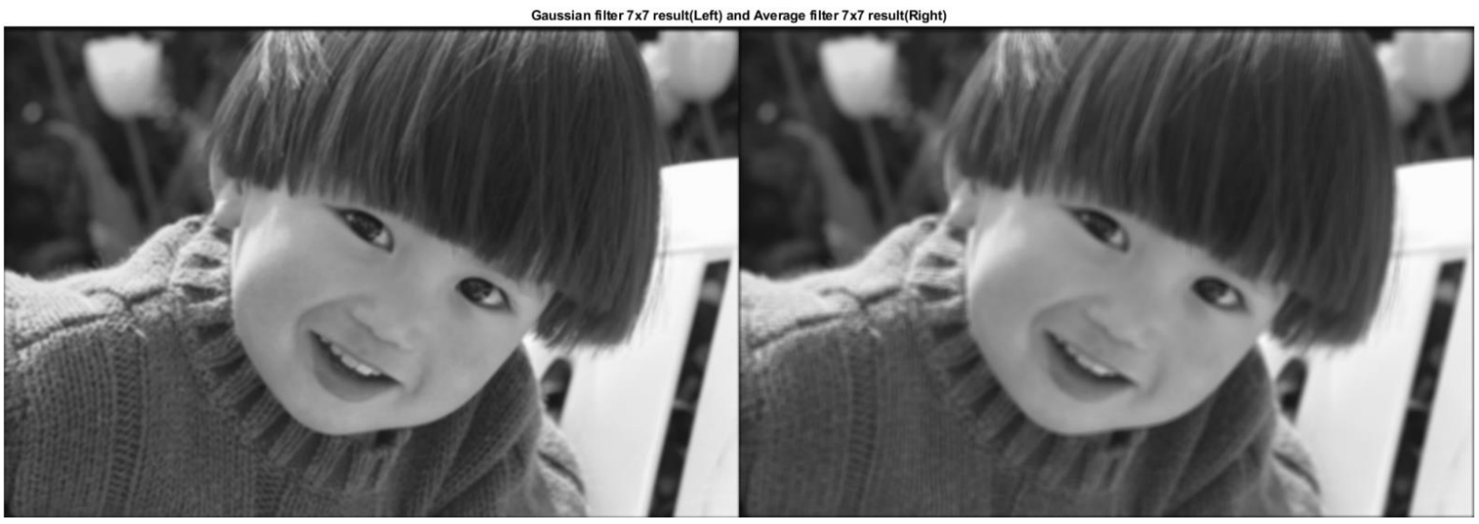
Gaussian filter 7x7 result(Left) and Average filter 7x7 result(Right)

*Figure 6: 7x7 hsize results for both filtering methods*

In the result, I have observed that averaging filter with 7x7 hsize is a bit blurred version than 7x7 hsized (sigma=1.5) gaussian filter result. Next, I have used 15x15 hsize for both filtering methods and gained a result like in the *figure 7.* In this part of the experiment again the average filtering is way more blurred than the gaussiang filtering. As a result, I have compared both averaging and gaussian smoothing methods with same hsize values and observed that due to additional parameter of gaussian filtering which is standart deviation value as sigma, gaussian filtering is more reliable than average filtering and also it is more useful for precision works.

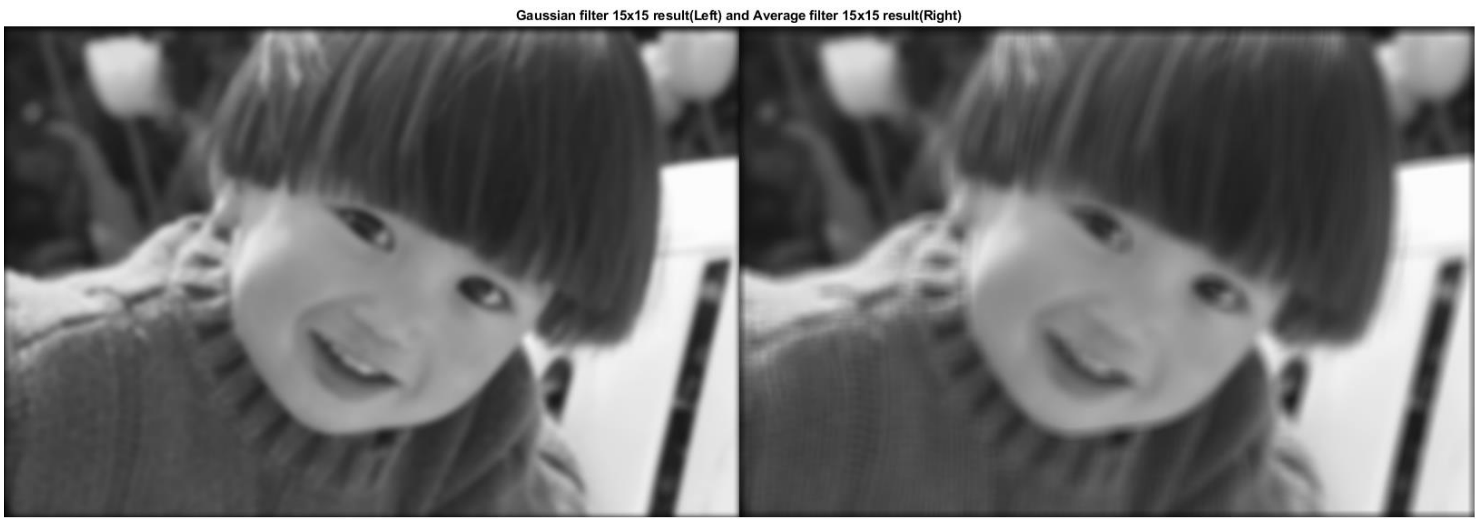
Gaussian filter 15x15 result(Left) and Average filter 15x15 result(Right)

*Figure 7: 15x15 hsize results for both filtering methods*

## Part 4: Median Filtering

In this part, I used boy.png as input image and first thing that I have applied to the input image was adding some noise with the help of *imnoise* method. So, I have used salt & pepper noise model and also, for two different corrupted image I have used %30 and %50 pixel percentage for adding noise to the image. After this step I have gained two noised result which are shown in the *figure 8.*

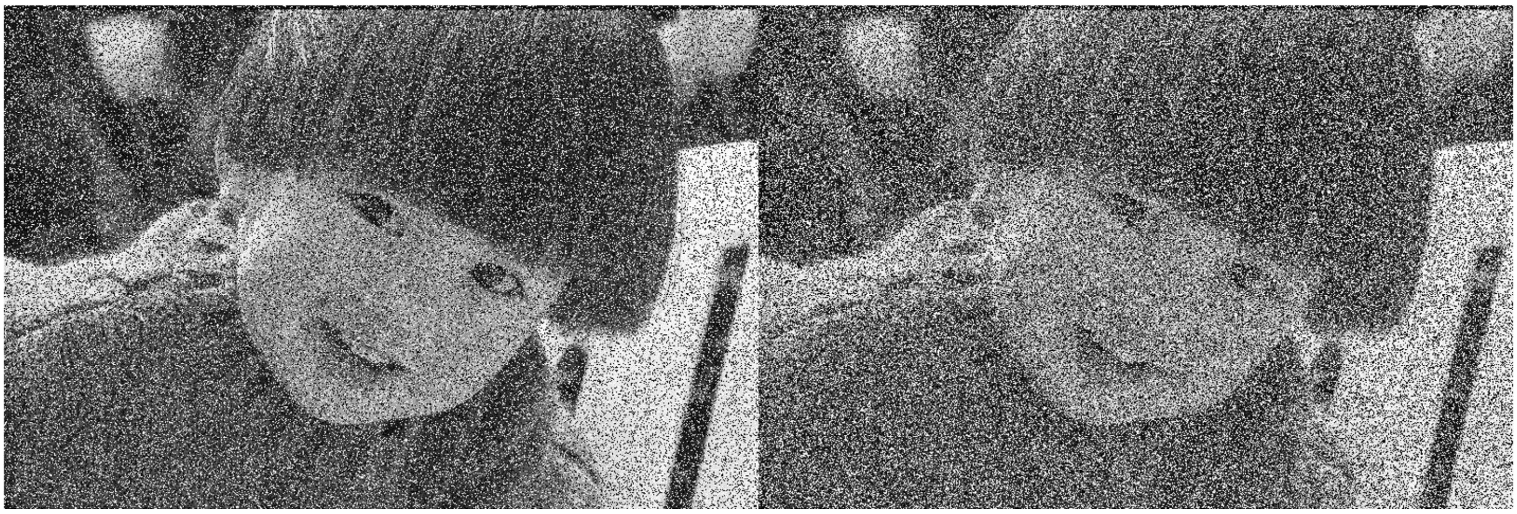
Figure 8: %30 noised version (Left) vs %50 noised version (Right)

Next, I have applied both 7x7 and 15x15 neighborhood around the corresponding pixels for the input image with applying median filtering to the %30 noised version of the image by using *medfilt2* function of MATLAB. Also, I have applied the average filtering method for the %30 noised version of the input image with 7x7 and 15x15 hsize values. Combined results for the four output is shown in *figure 9.*

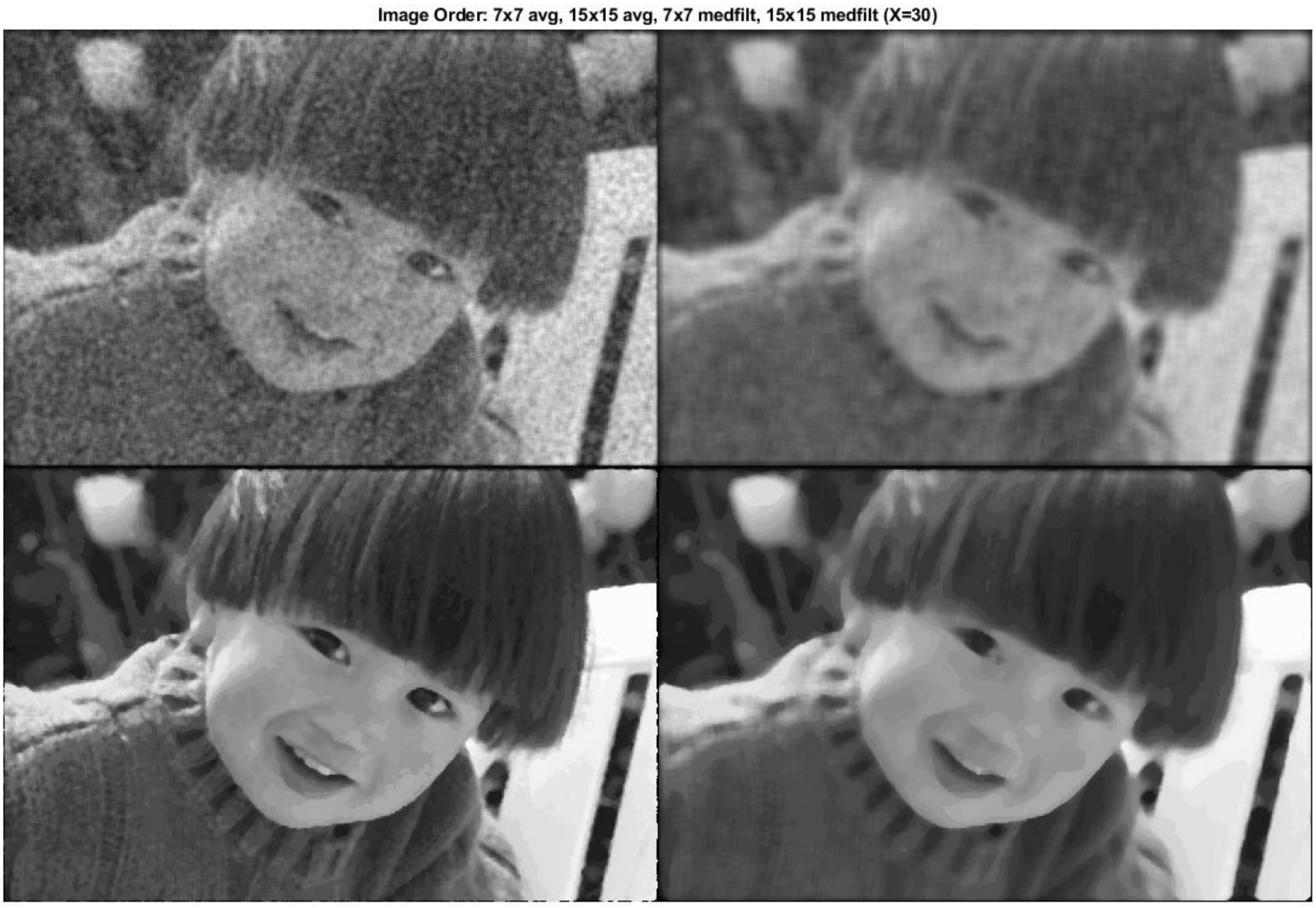
Image Order: 7x7 avg, 15x15 avg, 7x7 medfilt, 15x15 medfilt (X=30)
Figure 9: 7x7 Average filter result (Left top), 15x15 Average filter result (Right top), 7x7 Median filter result (Left bottom), 15x15 Median filter result (Right bottom)

So, for the results of %30 noised image, average filtering couldn't clearly handle the noise of the image. However, the median filter results show us a much clearer version of the noised image. It is close to the original version of the image, in 15x15 neighborhood version of the median filtering gained some blur effect due to its corresponding pixels of 15x15 size.

Lastly, I have used %50 noised version of the image for both average and median filtering methods and again I used 7x7 and 15x15 mask sizes for both methods. The combined results of this step are shown in *figure 10.*

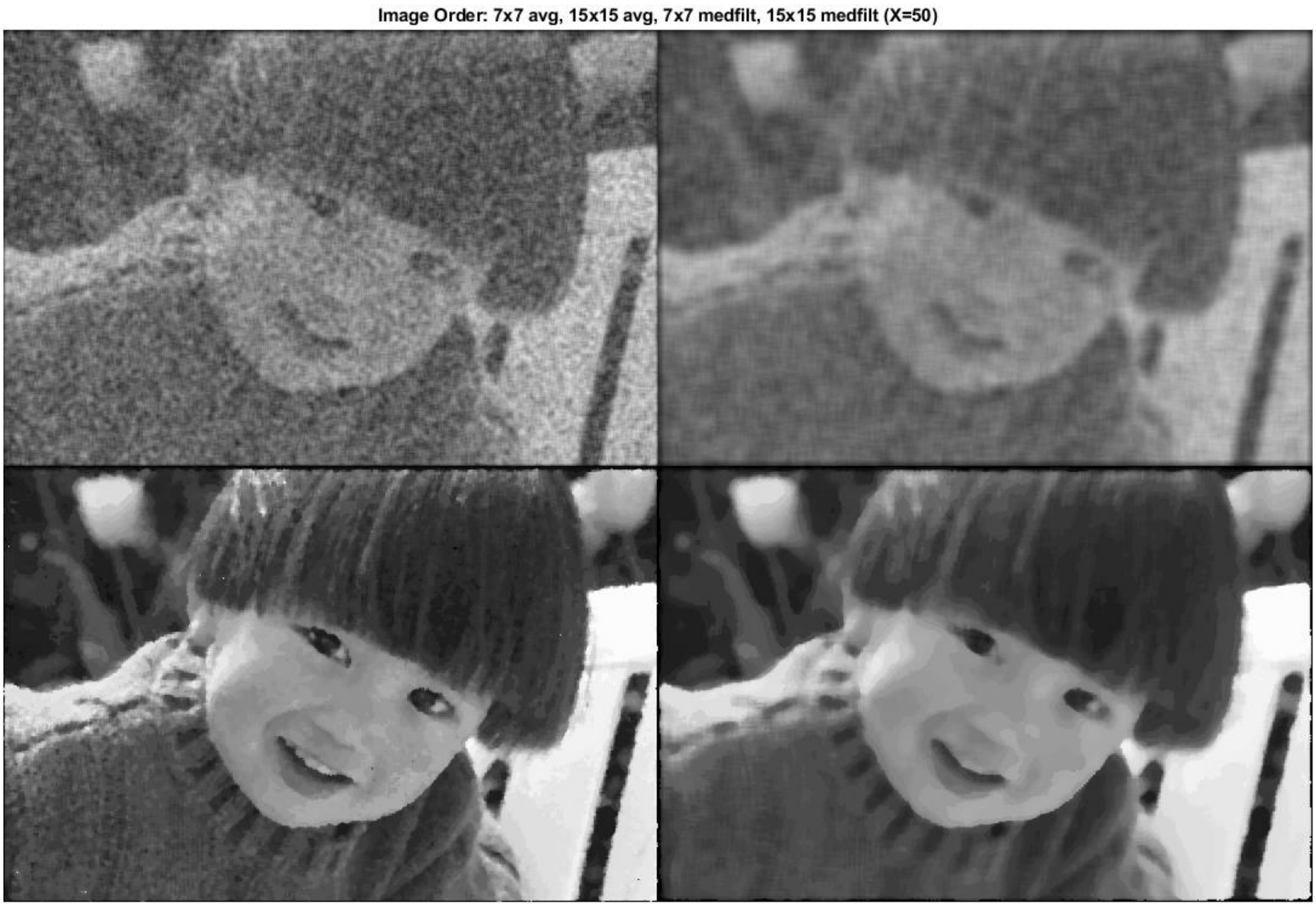
Image Order: 7x7 avg, 15x15 avg, 7x7 medfilt, 15x15 medfilt (X=50)

*Figure 10: 7x7 Average filter result (Left top), 15x15 Average filter result (Right top), 7x7 Median filter result (Left bottom), 15x15 Median filter result (Right bottom)*

Again, in the results, we can easily see that median filtering method handled most of the noised pixels when we compare it to the average filtering. However, in 7x7 mask sized version of the median filtering result we can see some minor noised pixels left from the %50 noised image, but in the 15x15 mask sized version there are no noise pixel.

## Part 5: Unsharp Masking

In this part, I have created my own unsharp masking function which takes an input image and sigma value for producing a sharpened version of the input image. So, in this part I have used an input image of moon which is a dark contrasted image (*figure 11)*. First, I have applied *imgaussfilt* method for the smoothing process with 7x7 mask size and 1.5 sigma value and produced a smoothed version of the input image, which is shown in the *figure 12*. Secondly, I have subtracted blurred version from the original version of the image to produce edge image of the input image (*figure 13)*.

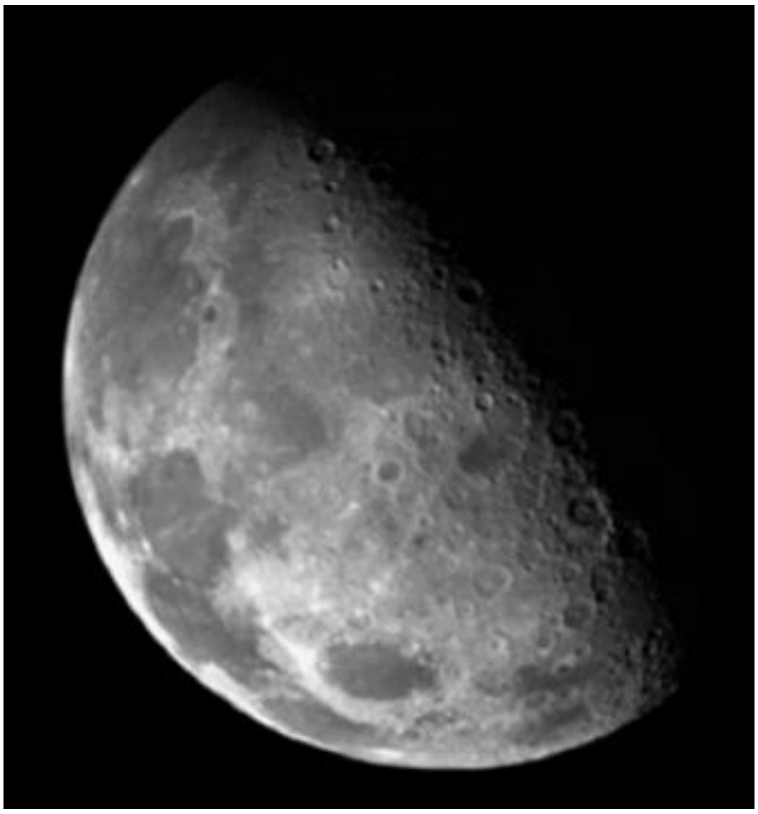

*Figure 11: moon.png as input image for this part*

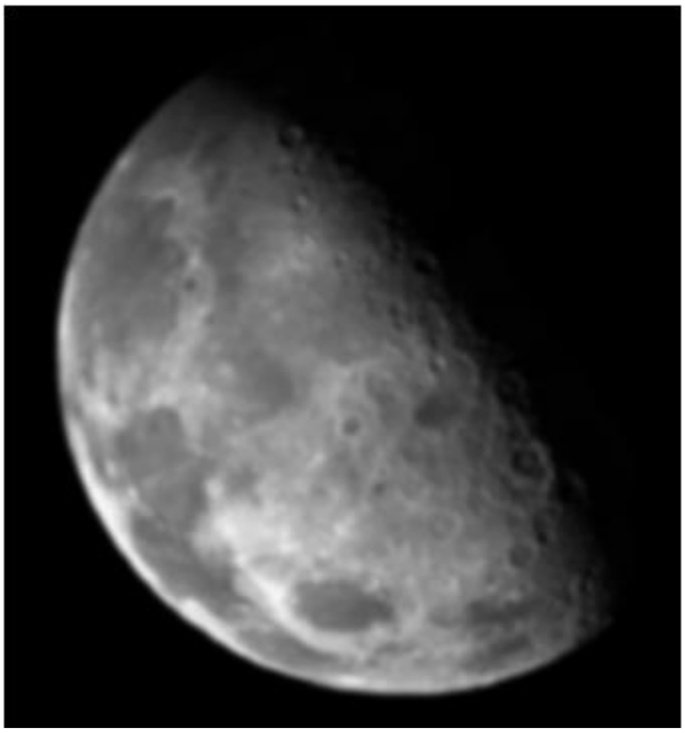

*Figure 12: Blurred version of the input image*

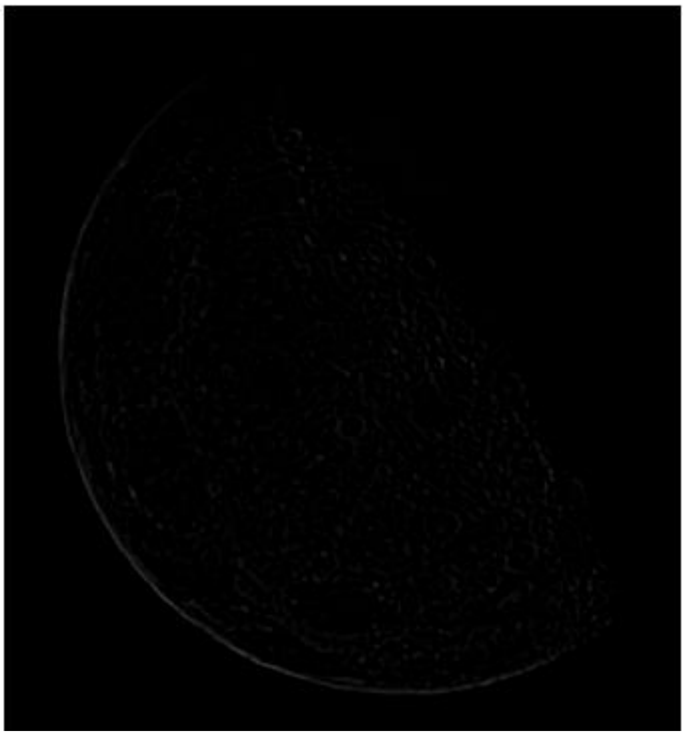

*Figure 13: Subtraction of blurred version from input image result (edge image)*

Lastly, to produce the sharpened version of the input image I have added the edge image (*figüre 13)* to the original image (*figure 11)* and observed a satisfying sharpened result just like in the *figure 14.*

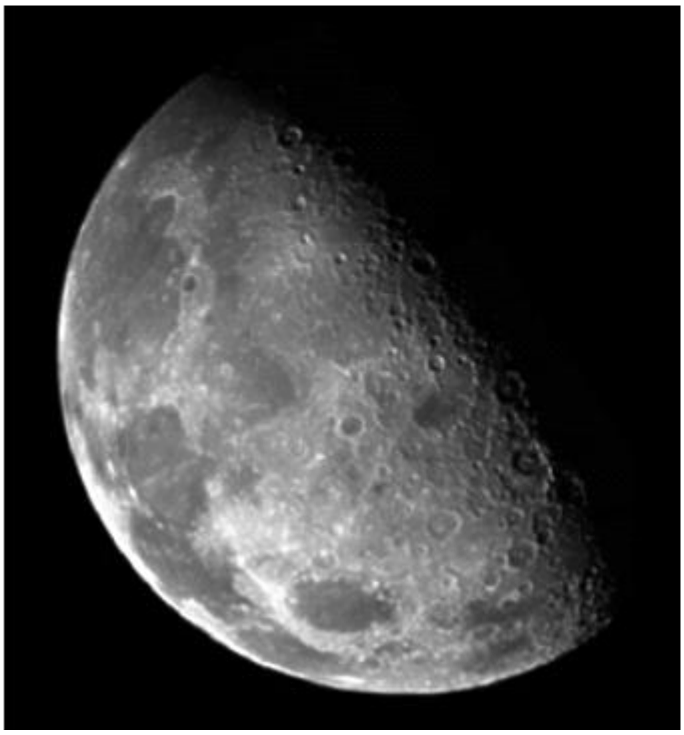

*Figure 14: Sharpened version of the moon.png*

To see the result and discuss the outcome, I have compared the original version with the sharpened version of the input image that I have worked on, in the *figure 15.* So, the result of my unsharp masking operation produced a sharpened version of the input image and the result is much more satisfying, clearer and more contrasted when we compare it to the first version.

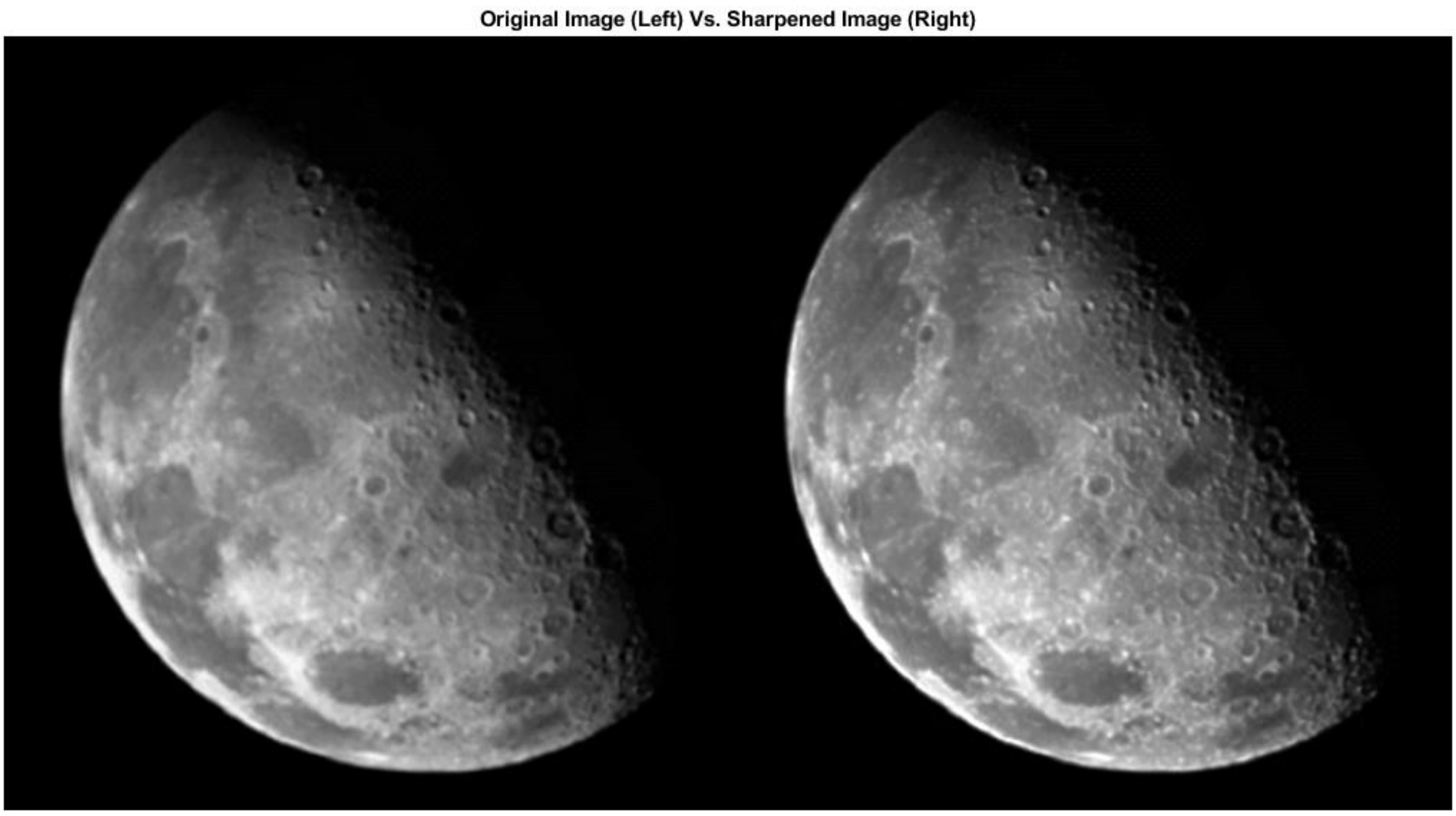

*Figure 15: Comparison of the original version and sharpened version of the input image*

## Part 6: Applying Bilateral filter (BONUS)

In this part, I have used *imbilatfilt* function of MATLAB, and used baboon.png image as input. So, I have used different values of degreeOfSmoothing and spatialSigma parameters of the function and observed that when I increased the spatialSigma parameter too much the process is getting really slow, as a result of that terrible time complexity I have majorly changed the degreeOfSmoothing continuously of course I also have changed the spatialSigma but didn’t increase it too much, in *figure 16* you may see combined outputs of the different parameterized bilateral filtered versions of the input image.

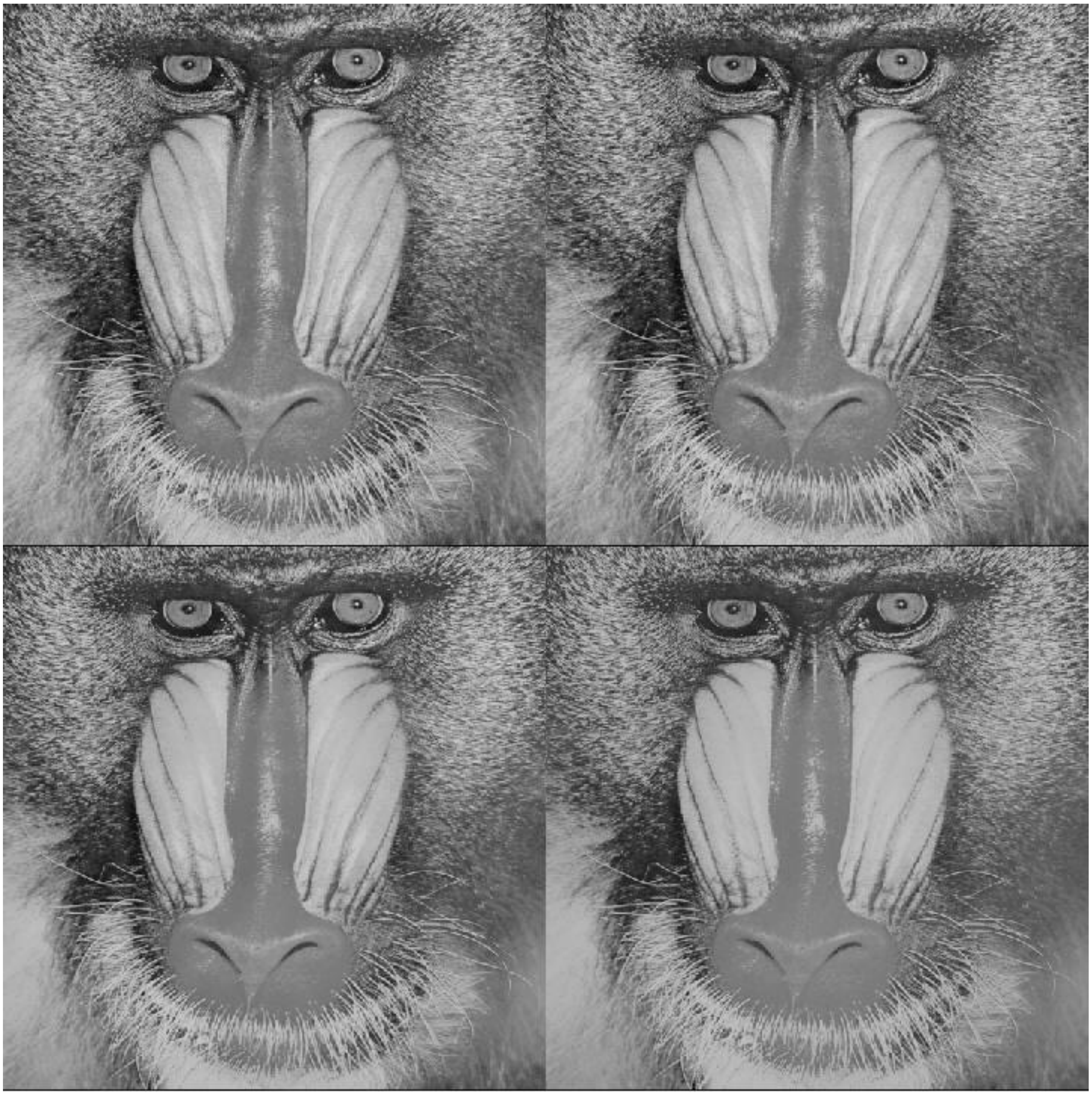
*Figure 16:Four output results combined for the bilateral filtering. Separated details of these images are mentioned below.*

So, I have used the following parameters for the experiment; (degreeOfSmoothing shortened by “d” and spatialSigma is shortened by “s”)

- Left Top Image (figure 16): Original version
- Right Top Image (figure 16): d=10, s=1
- Left Bottom Image (figure 16): d=100, s=10
- Right Bottom Image (figure 16): d=300, s=30

In the *figure 16,* I have observed that when I increase degreeOfsmoothing with much higher values the output image is clearly smoothed version, just like in the right bottom image of *figure 16.*

Additionally, I have waited some time to observe a high spatialSigma value version of the image and gained the result that is shown in *figure 17.* In this result, I have used 100 spatialSigma value and 10 degreeOfSmoothing. However, the result doesn’t seem so different than the original version thus, I have applied 100 spatial Sigma and 100 degreeOfSmoothing to observe if it is gonna have a salient result (*figure 18).* Again, the result wasn’t so sensible.

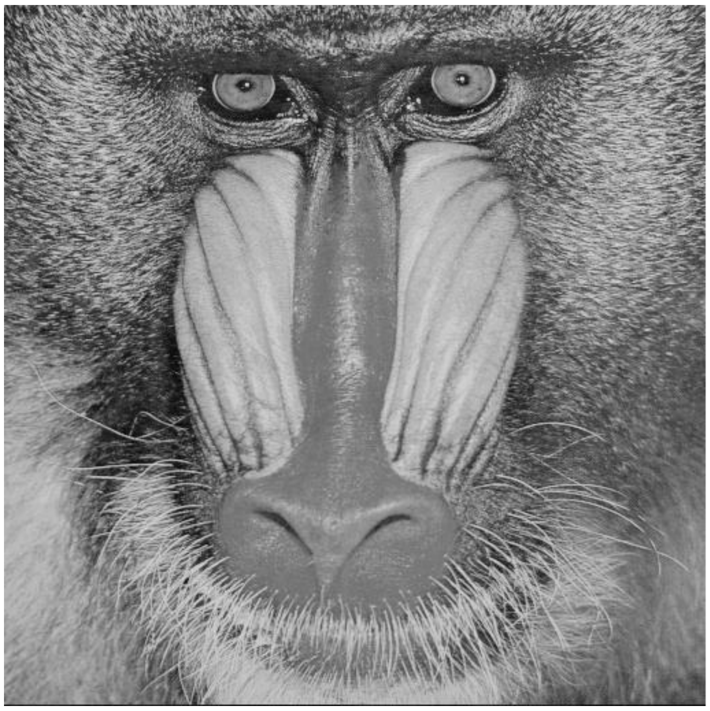

*Figure 18: spatialSigma=100, degreeOfSmoothing=10*

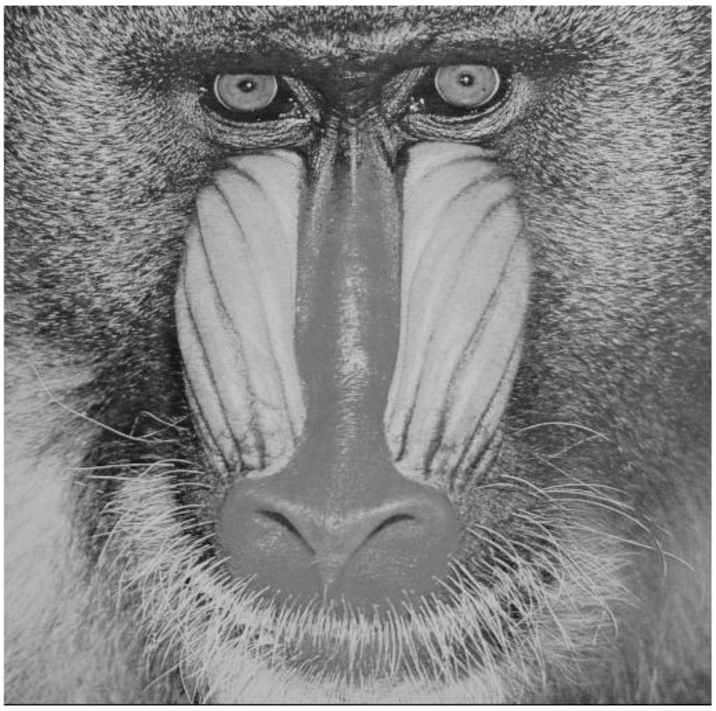

*Figure 17: spatialSigma=100, degreeOfSmoothing=100*