BARTIN UNIVERSITY

Computer Engineering Department

MAHMUD MARDİNİ
18670310077

BSM423

Basics of Computer Vision

Homwork 2 Report

One dimensional Gaussian is defined as

[1]
$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

x is input, mean is μ and variance is σ^2.

1) You will implement 1D - Gaussian function.
   - X values are range of [− 100, 100]
   - Stepsize = 1
   - Variance σ^2 is in different values which are 0.2, 1.0 and 5.0
   - Mean μ = 0

   You can use built-in exp function of Python.

   Use plot function to see the results. Put them into your assignment report.

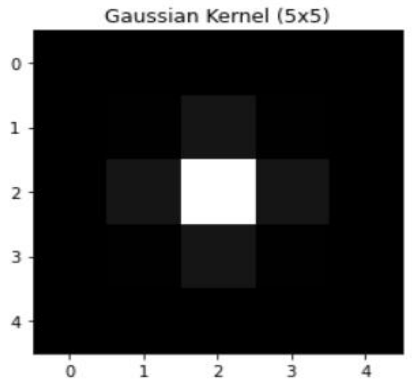
Answer 1:

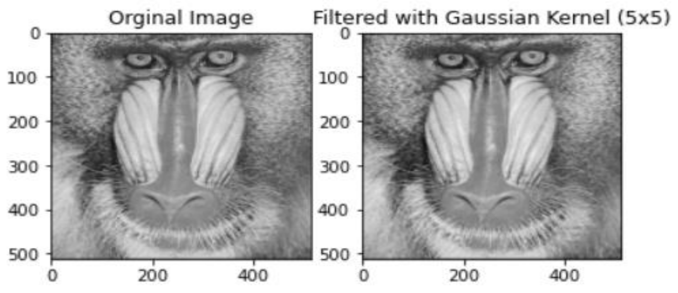
In this question, I used 3 functions. These functions are:

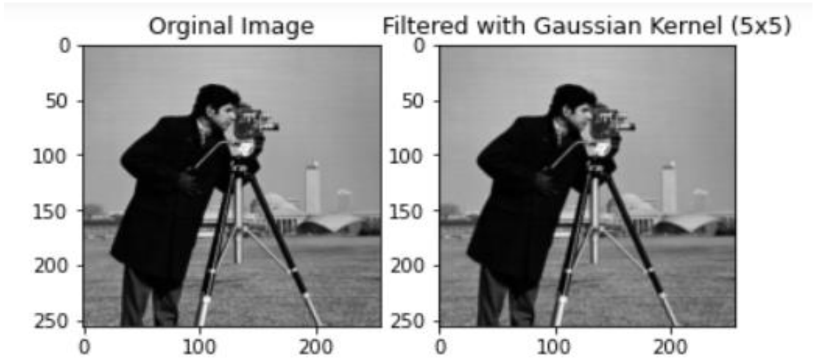
1. gaussian_filter_formula(x, mean, sigma) that helps to apply gaussian kernel formula
2. gaussian_kernel(kernel_size, sigma) that returns the gaussian kernel using previous function
3. showFilteredImages(images, kernel, filter_name, average=True) that compares input and output images

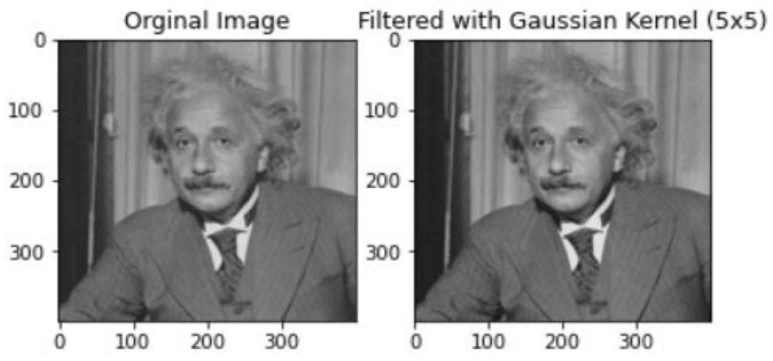
Outputs:

(kernel with size of 5x5 and varyans = 0.2)

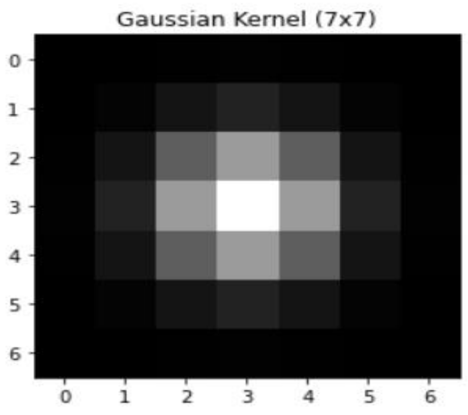


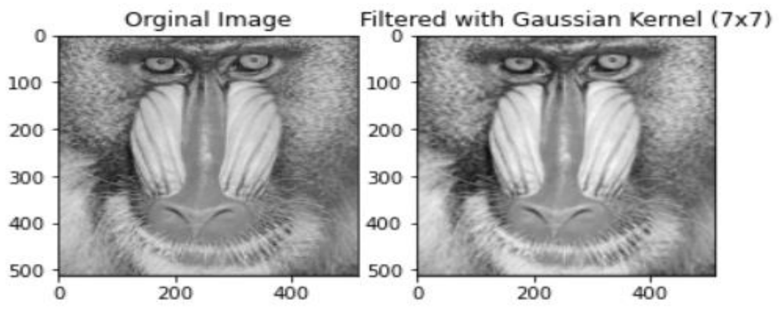


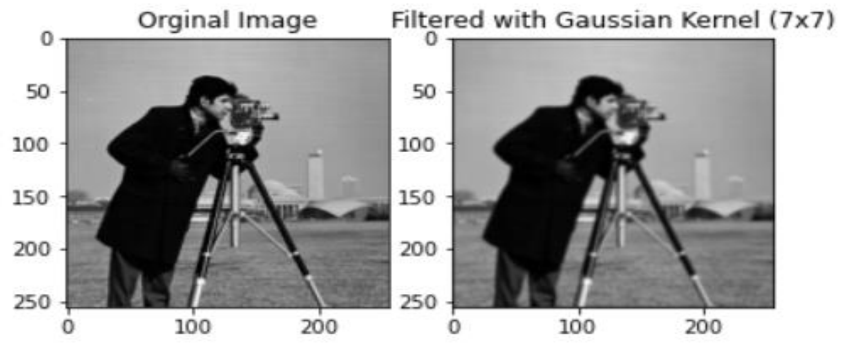


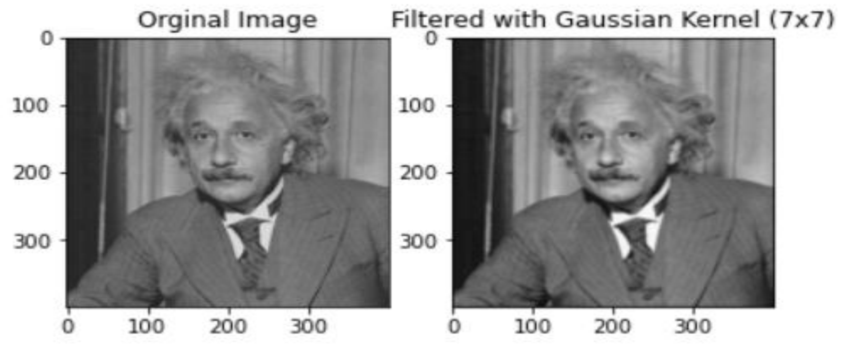


(kernel with size of 7x7 and varyans = 1.0)

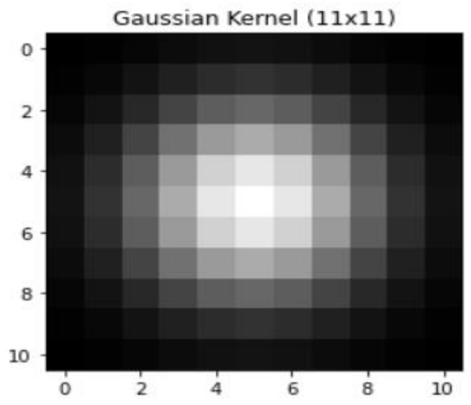


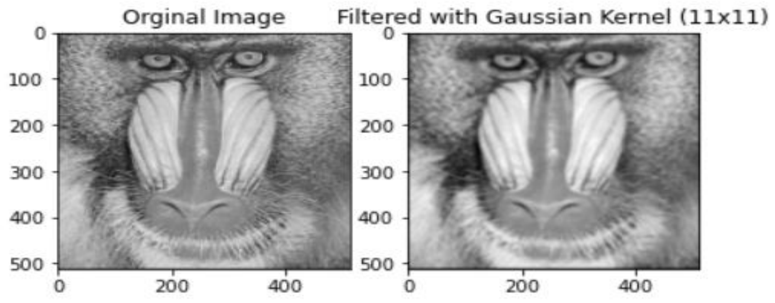


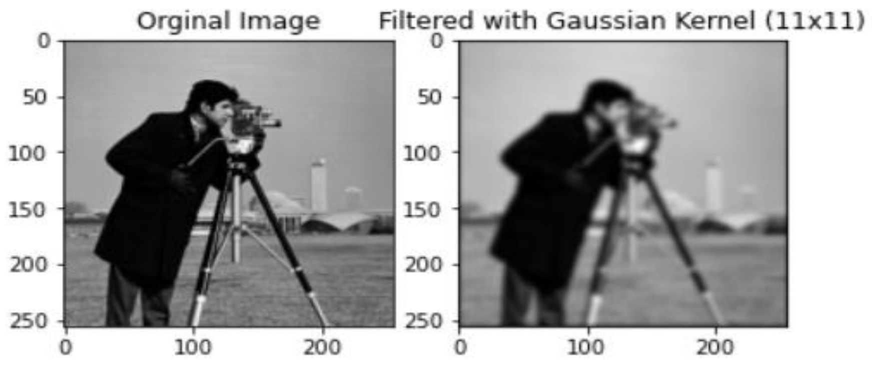


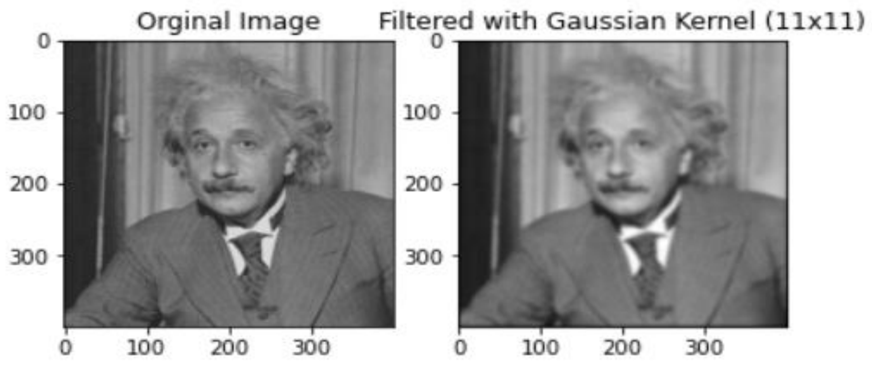


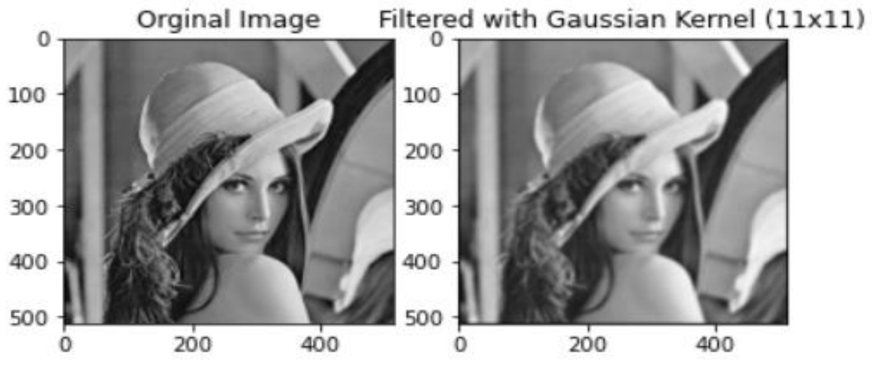
(kernel with size of (11x11) and varyans = 5.0)





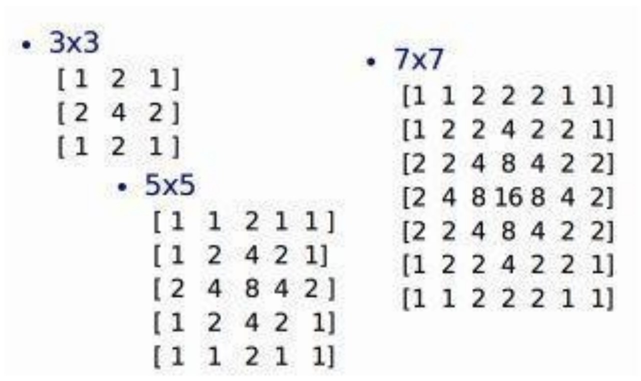






2) Use 3x3, 5x5, 7x7 Gaussian kernels given below. You will do normalization on filters dividing by the sum of coefficient. [2] Implement each kernel to the each given image in
‘sample images1’ directory in the zip file. Put the results into your assignment report.

[2]



- 3x3

```
[1 2 1]
[2 4 2]
[1 2 1]
```

- 5x5

```
[1 1 2 1 1]
[1 2 4 2 1]
[2 4 8 4 2]
[1 2 4 2 1]
[1 1 2 1 1]
```

- 7x7

```
[1 1 2 2 2 1 1]
[1 2 2 4 2 2 1]
[2 2 4 8 4 2 2]
[2 4 8 16 8 4 2]
[2 2 4 8 4 2 2]
[1 2 2 4 2 2 1]
[1 1 2 2 2 1 1]
```

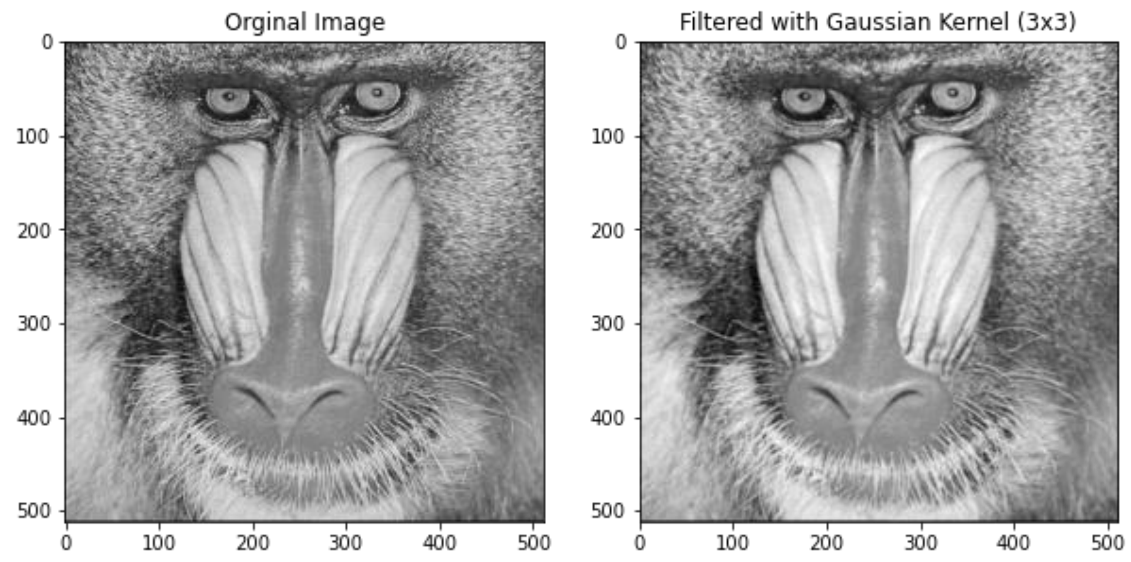
Answer 2:

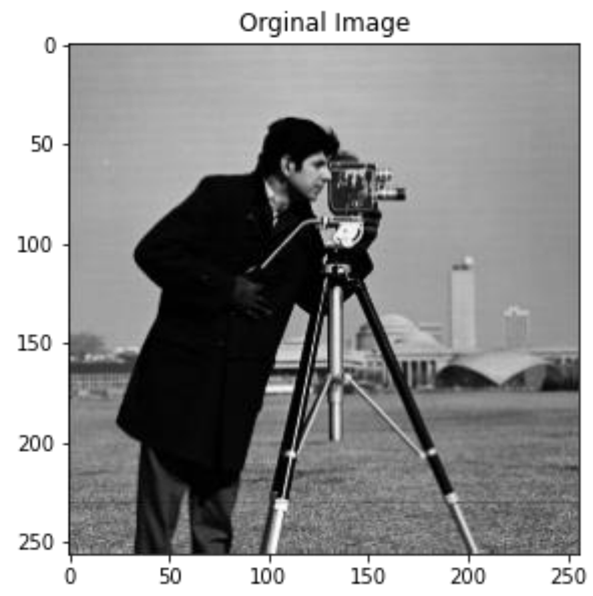
In this question, I built the given kernels using numpy library, then I did the normalization by divide each element of kernel by the sum of all elements.

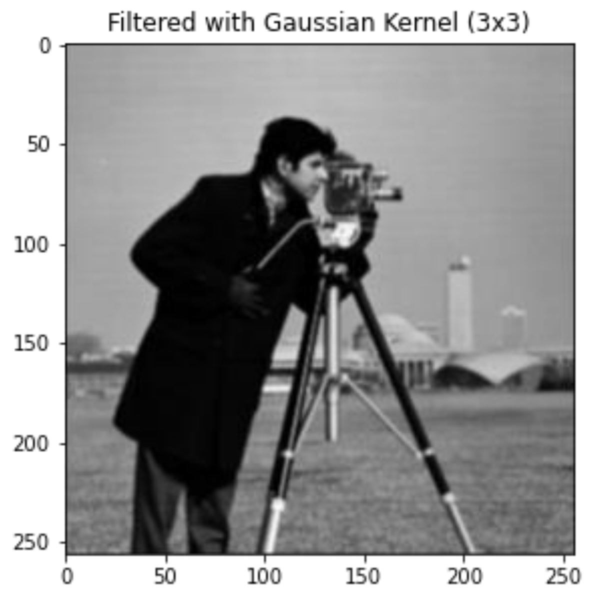
Then I implemented the kernels to each image in sample_images1 folder using my convolution function.

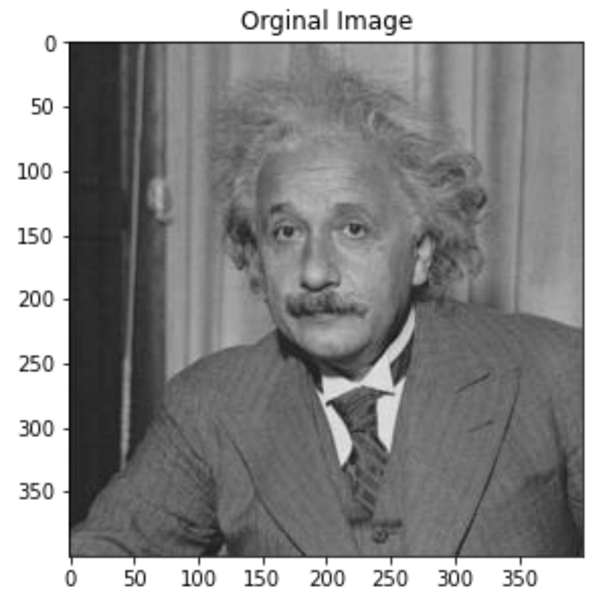
Then I get the result using showFilteredImages function.

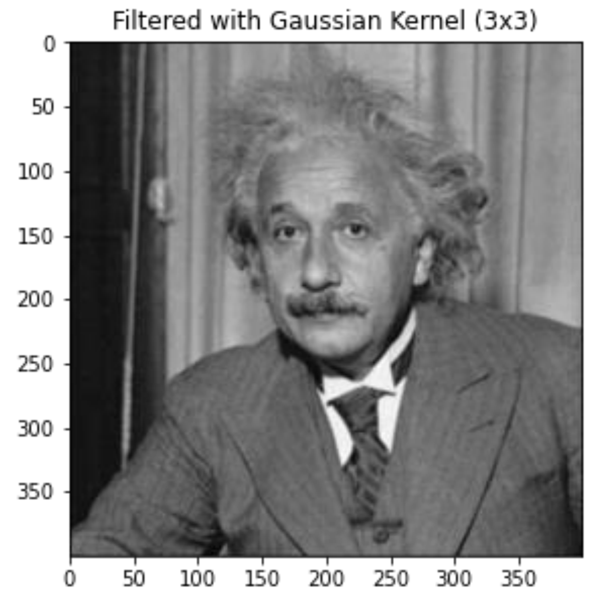
Outputs:

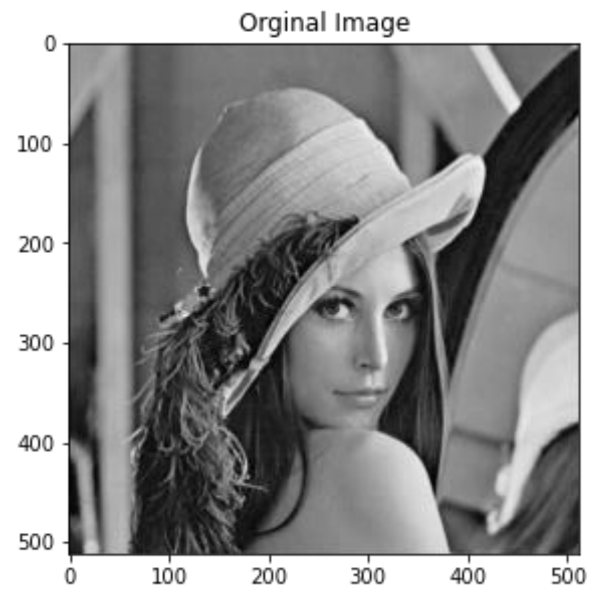


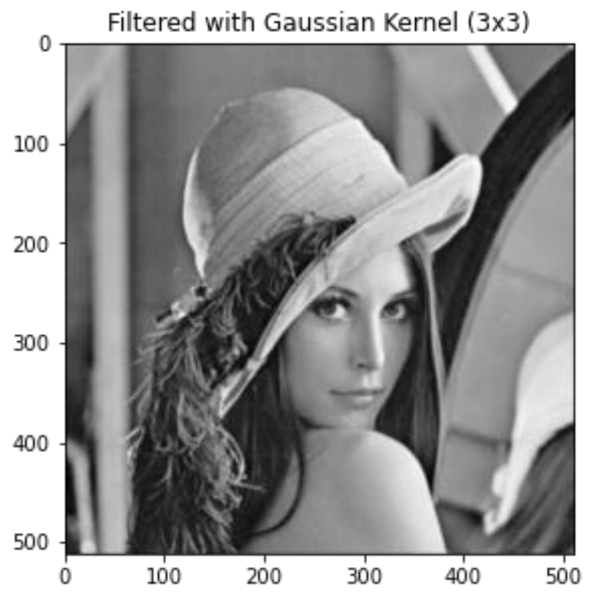

Orginal Image
0 50 100 150 200 250
0 50 100 150 200 250

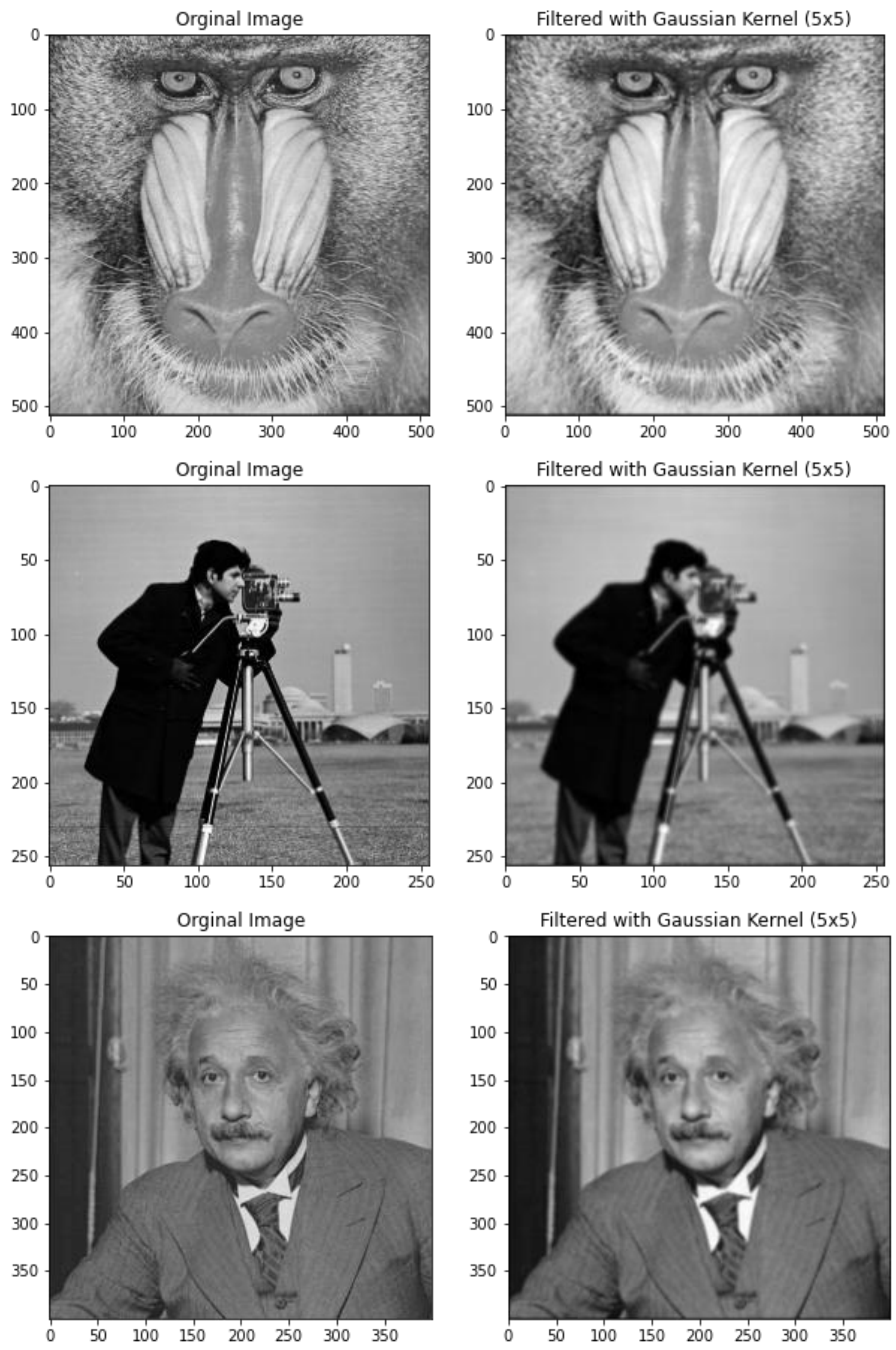

Filtered with Gaussian Kernel (3x3)
0 50 100 150 200 250
0 50 100 150 200 250

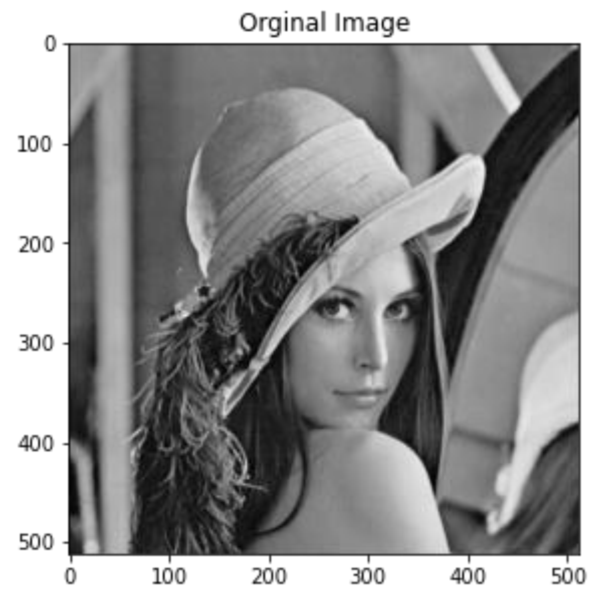

Orginal Image
0 50 100 150 200 250 300 350
0 50 100 150 200 250 300 350

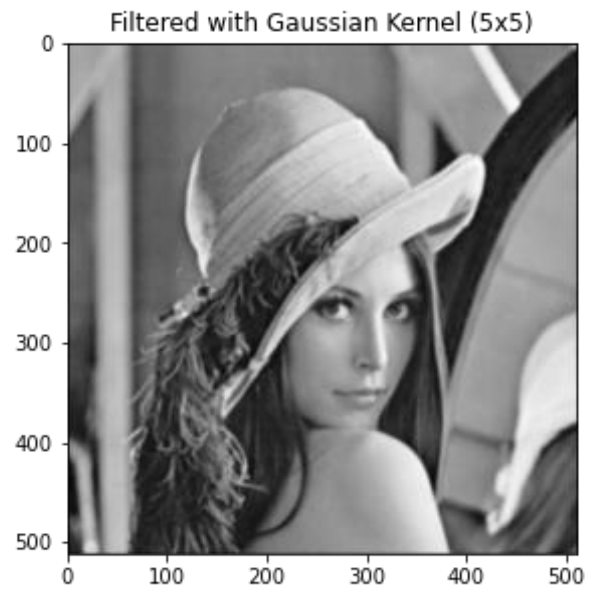

Filtered with Gaussian Kernel (3x3)
0 50 100 150 200 250 300 350
0 50 100 150 200 250 300 350

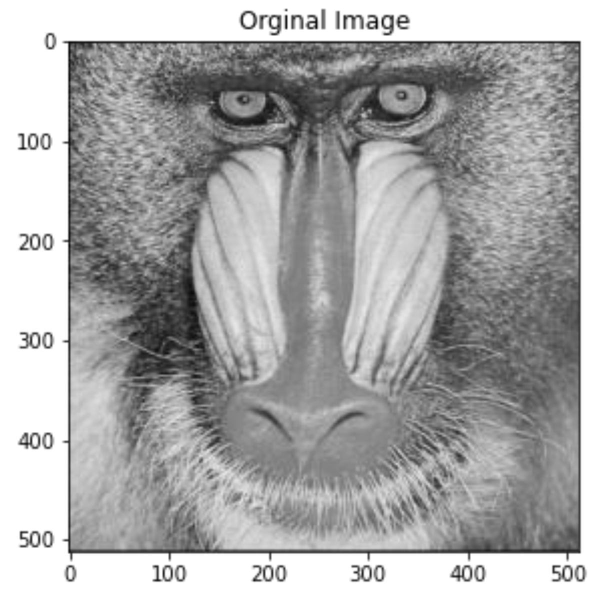

Orginal Image
0 100 200 300 400 500
0 100 200 300 400 500

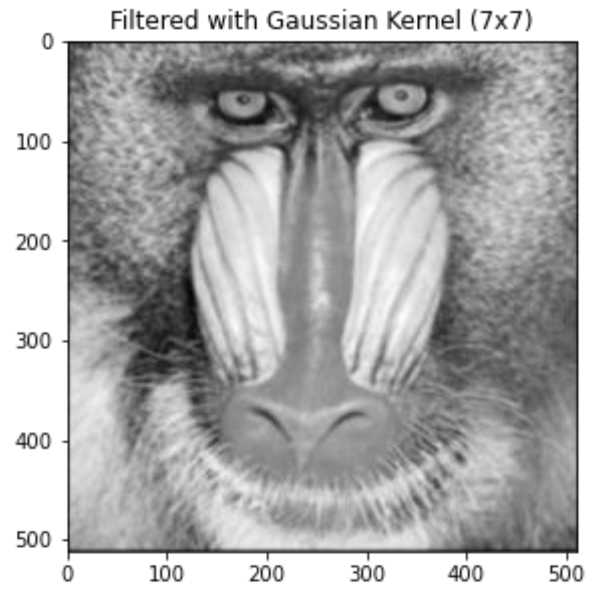

Filtered with Gaussian Kernel (3x3)
0 100 200 300 400 500
0 100 200 300 400 500

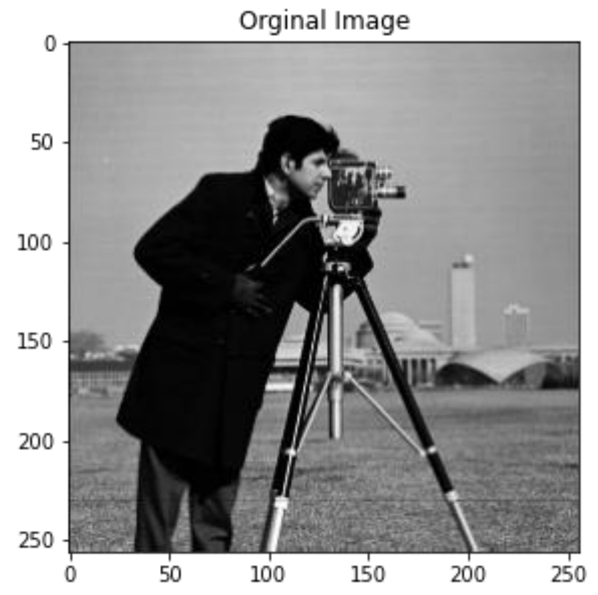

Orginal Image
Filtered with Gaussian Kernel (5x5)
Orginal Image
Filtered with Gaussian Kernel (5x5)
Orginal Image
Filtered with Gaussian Kernel (5x5)

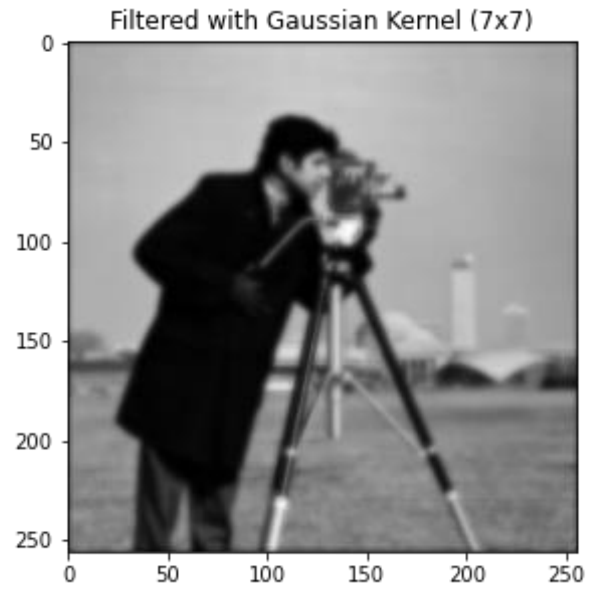

Orginal Image


Filtered with Gaussian Kernel (5x5)


Orginal Image


Filtered with Gaussian Kernel (7x7)


Orginal Image


Filtered with Gaussian Kernel (7x7)

3) Use Gaussian kernels given in question#2 as sharpening filter. According to lecture slides,

$$\underset{\substack{\uparrow \\ \text{image}}}{F} + \alpha\,(F - \underbrace{F * H}_{\text{blurred image}})$$

H will be Gaussian Kernel. Use different α values. Implement each filter to your own face image. Put the results into your assignment report.

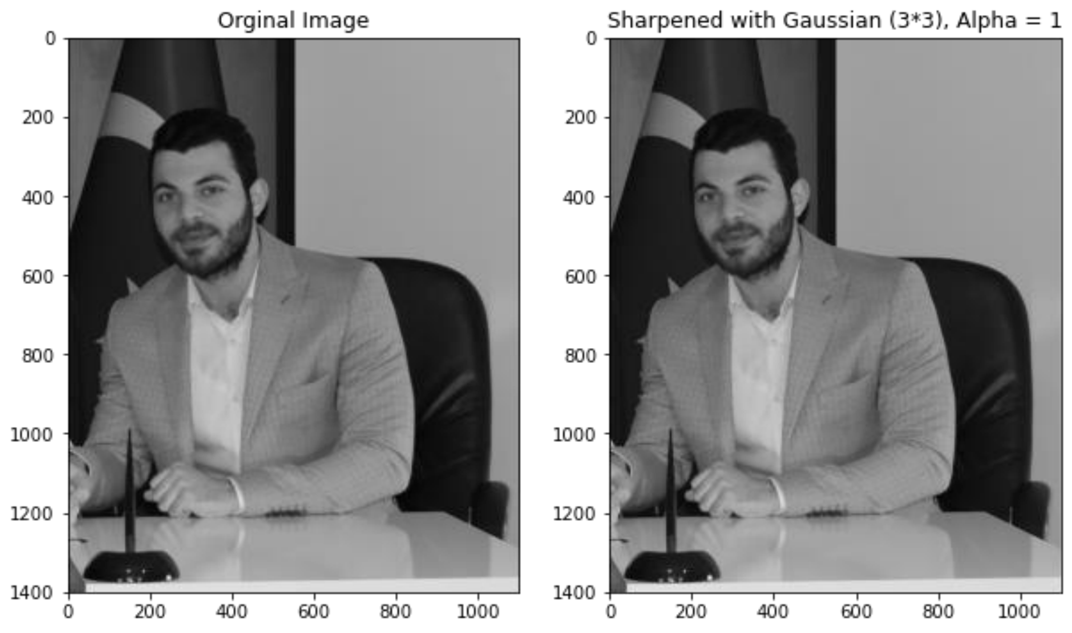
Answer 3:

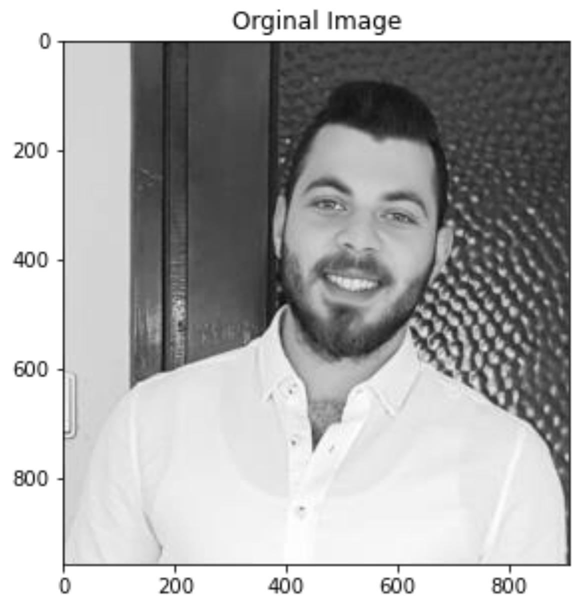
In this question I stored my images in my_images[] array then I built two functions:

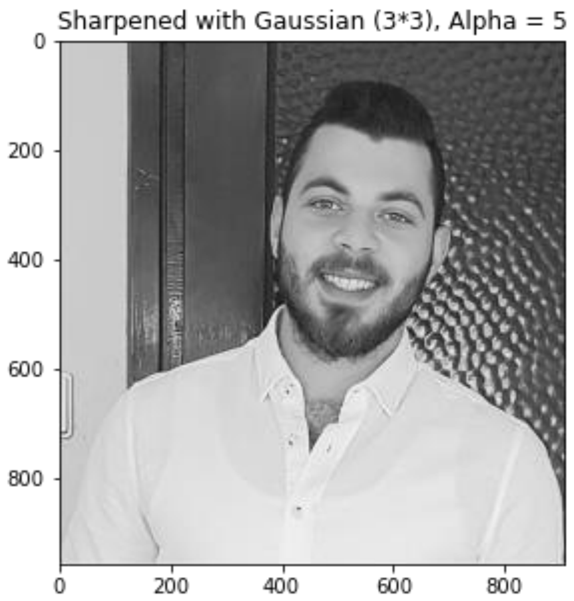
1. sharpening_with_gaussian(img, kernel, alpha) that returns sharpened images by applying the given rule (F = F + alpha(F - (F * H))) using convolution function to get the blurred image and multiply it with alpha and then add it to the original image. These operations done by iterate each pixel of images (original, blurred images and image details) with limit pixels values between 0 and 255 (like I did in convolution function).

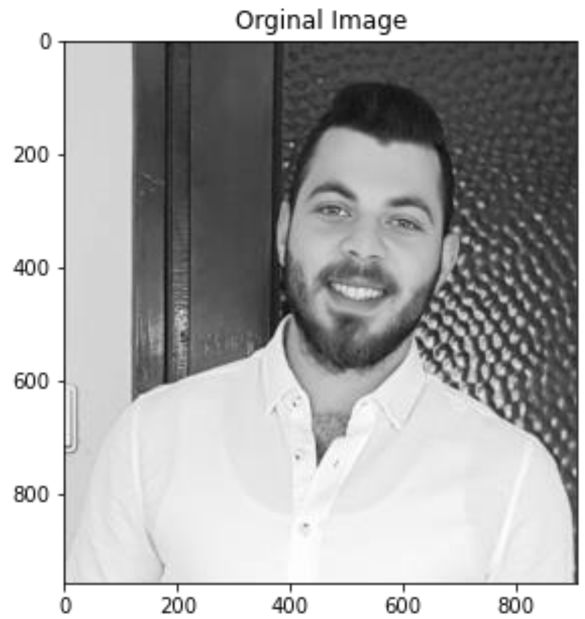
2. showSharpenedImages(input_images, kernel, alpha, filter_details) that shows sharpened images and compare them with the original ones.

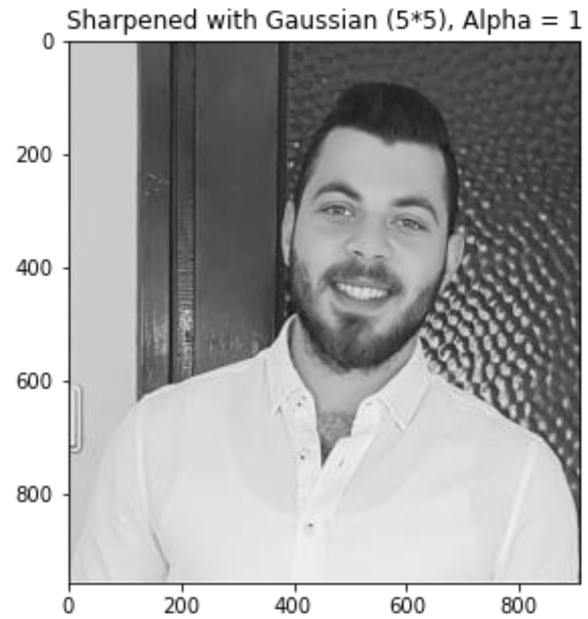
Outputs:



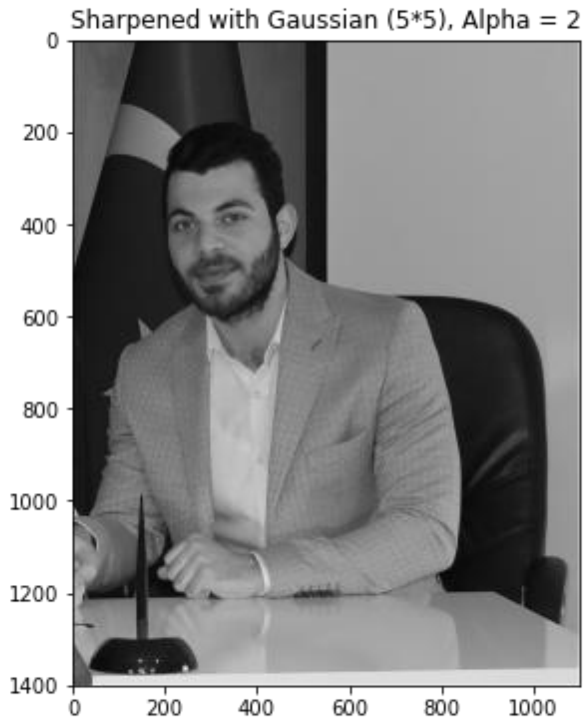

Orginal Image

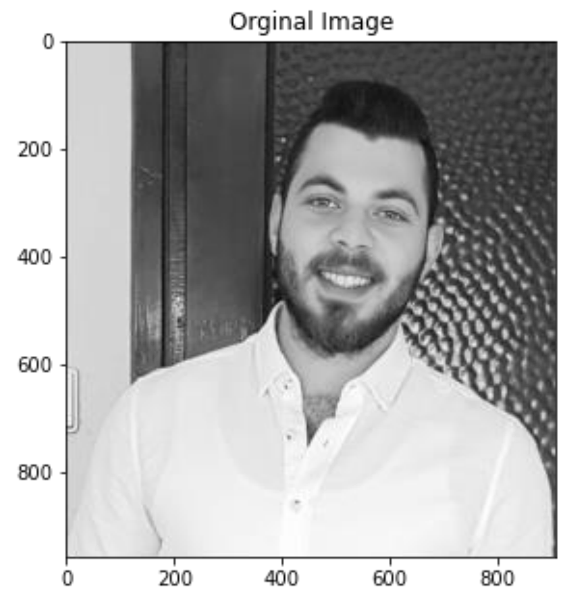

Sharpened with Gaussian (3*3), Alpha = 5

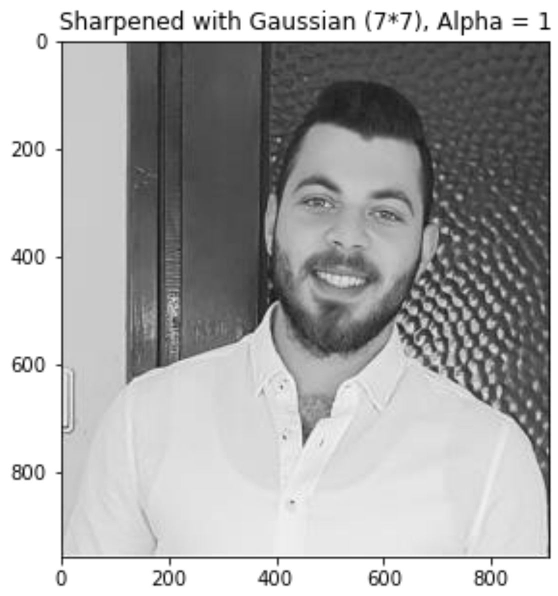

Orginal Image

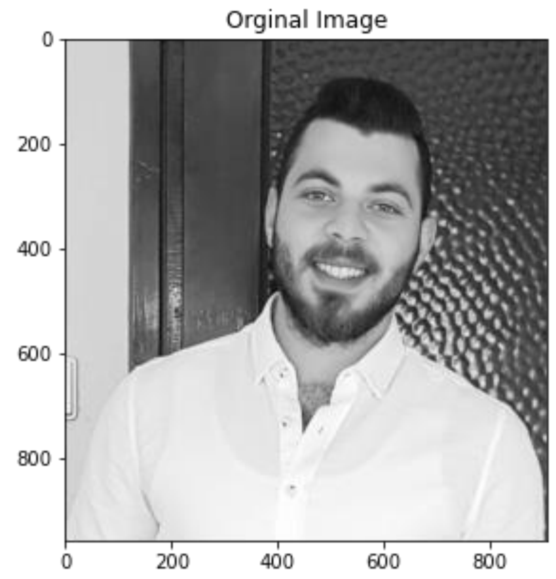

Sharpened with Gaussian (5*5), Alpha = 1

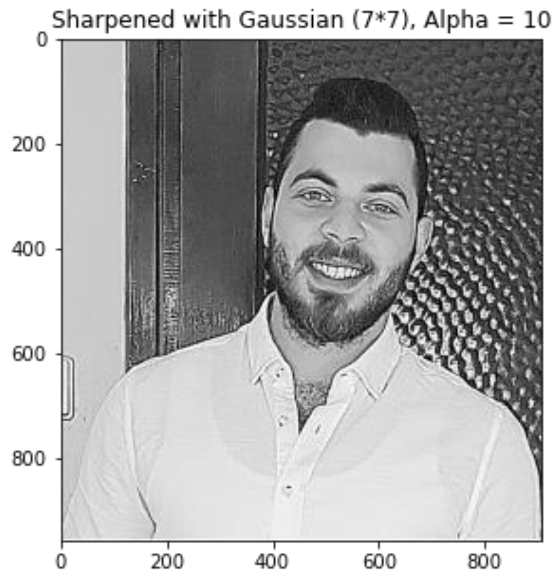
Orginal Image


Sharpened with Gaussian (5*5), Alpha = 2









4) In zip file, there are salt and pepper noise added images in ‘sample images2’ directory.

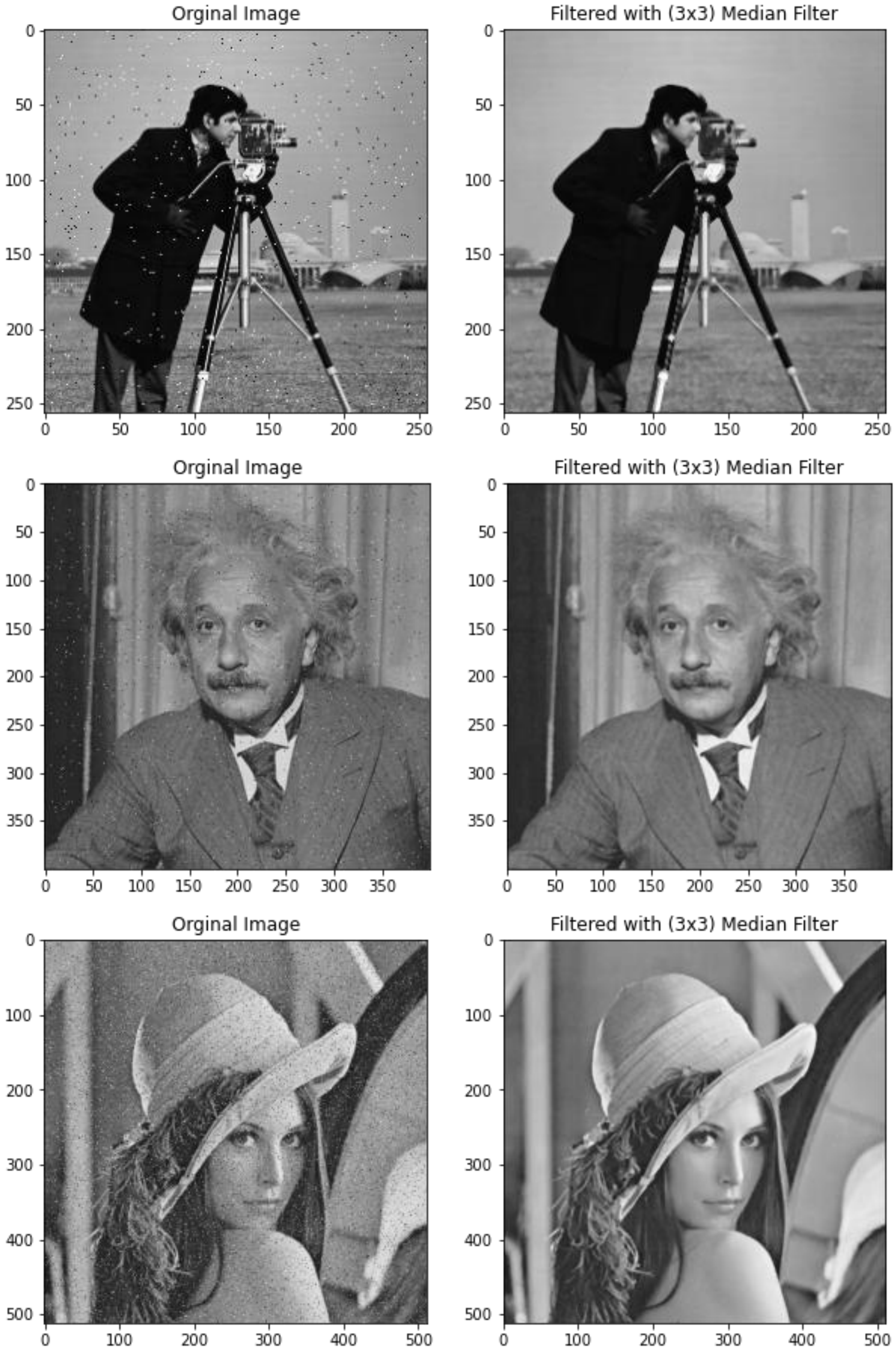
   Apply 3x3, 5x5 and 7x7 median filters to these images. Put the results you’re your assignment report.

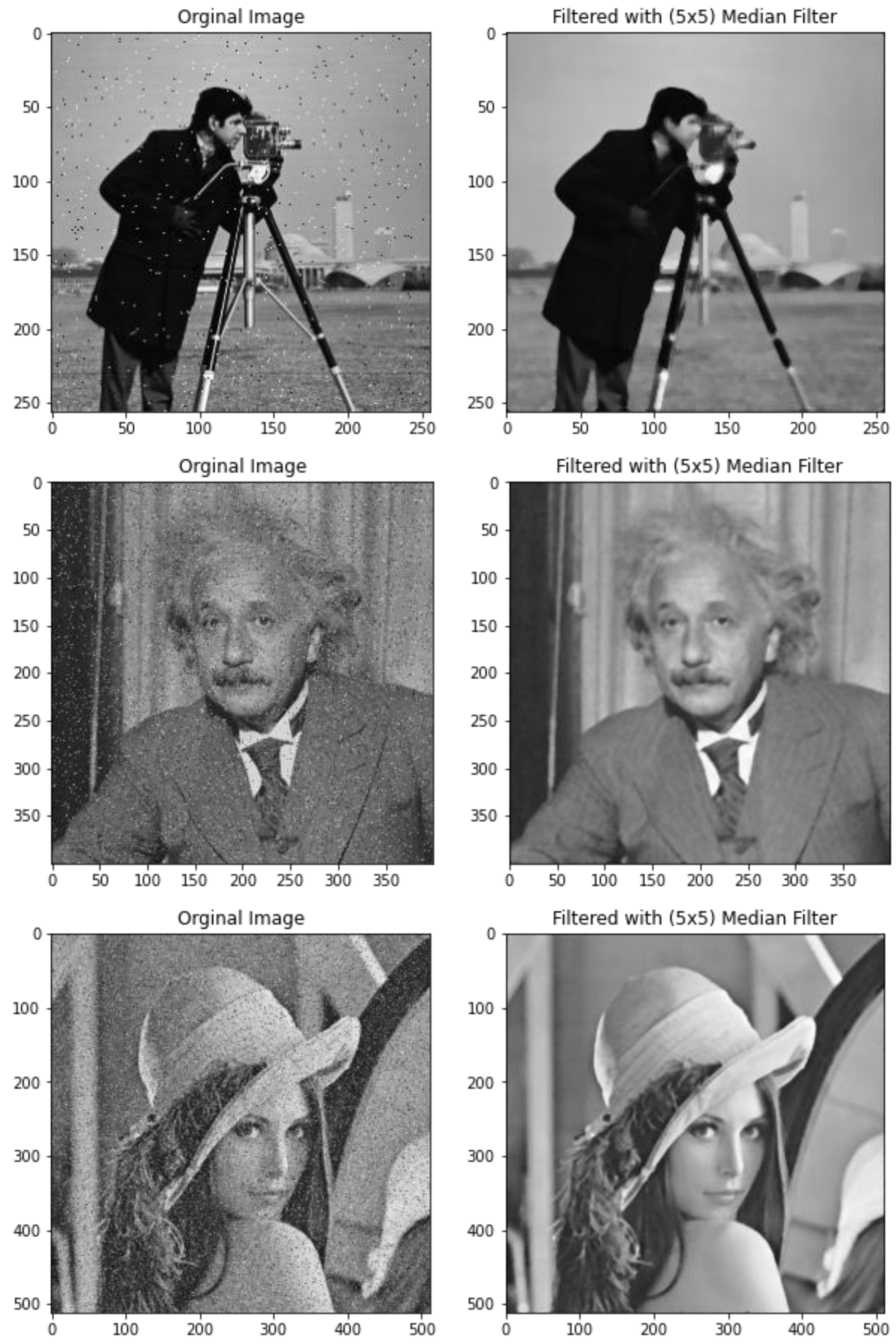
Answer 4:

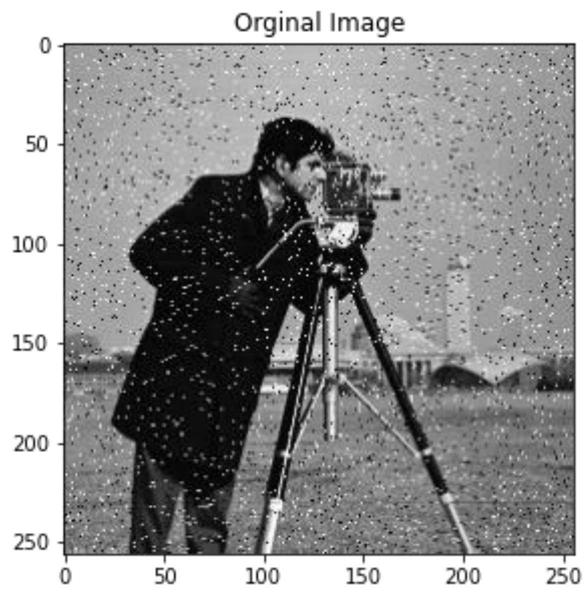
In this question I stored my images in imgs2 [] array then I used 3 functions:

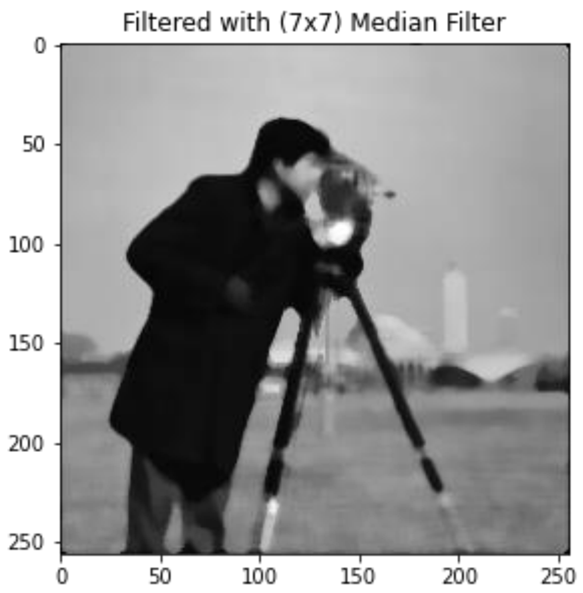
1. medianFilter(img, kernel_size) that take an image as input and implement median filter to it after adding zero padding according to kernel size.
   To do that, I iterate each pixel of input image and took iterated pixels in the number of kernel size. Then I put these pixels in one vector using flatten() function to sort them and take the median pixel. I used insertion sort algorithm to sort vector elements then I set the median pixel and so salt and pepper noise has removed.
2. insertionSort(vector) that sort vector elements using insertion sort algorithm. I used this algorithm because in generally the vector size won’t be too big so insertion sort algorithm works fast in such this problem.
3. showFilteredImages(input_images, kernel_size) that shows filtered images and compare them with the original ones.

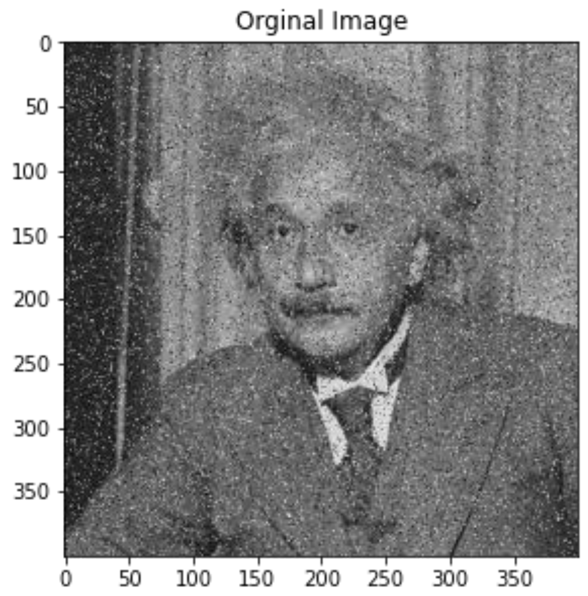
Outputs:

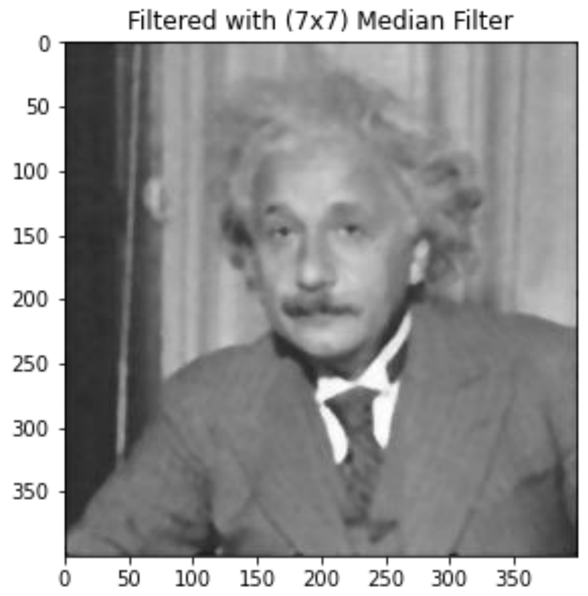

Orginal Image
Filtered with (3x3) Median Filter
Orginal Image
Filtered with (3x3) Median Filter
Orginal Image
Filtered with (3x3) Median Filter

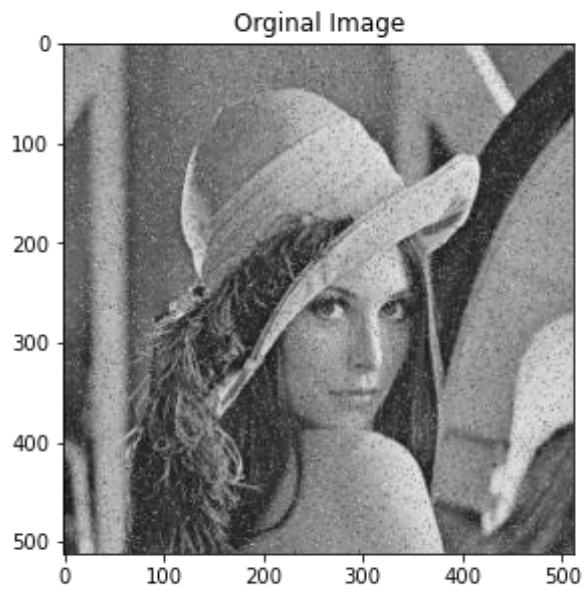

Orginal Image
Filtered with (5x5) Median Filter
Orginal Image
Filtered with (5x5) Median Filter
Orginal Image
Filtered with (5x5) Median Filter

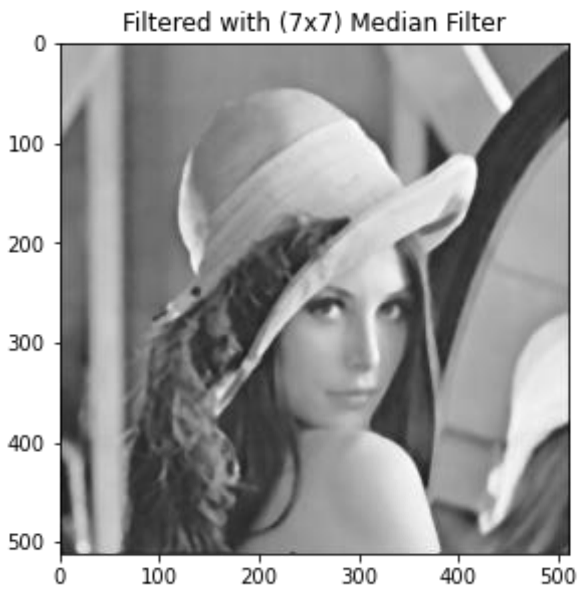

Orginal Image


Filtered with (7x7) Median Filter


Orginal Image


Filtered with (7x7) Median Filter


Orginal Image

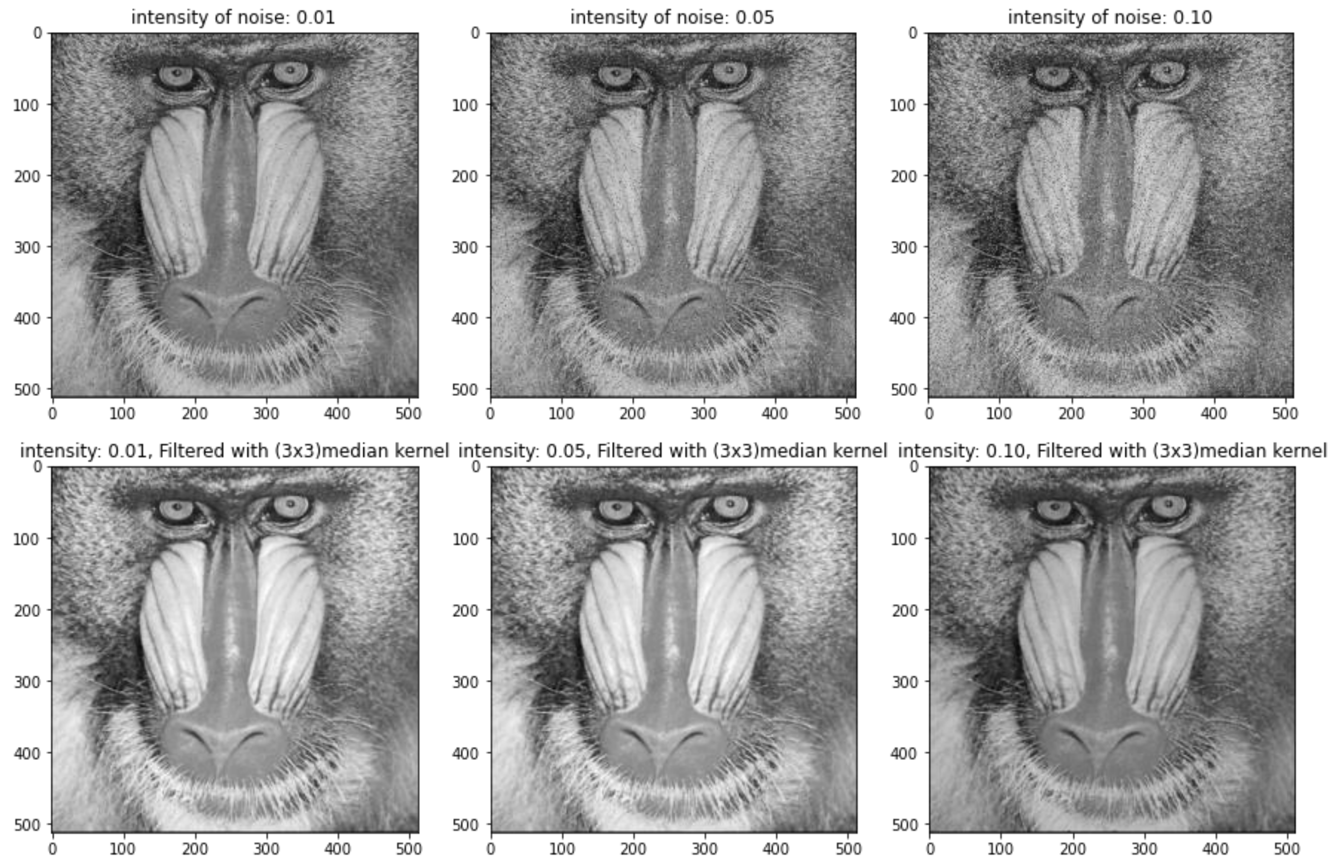

Filtered with (7x7) Median Filter

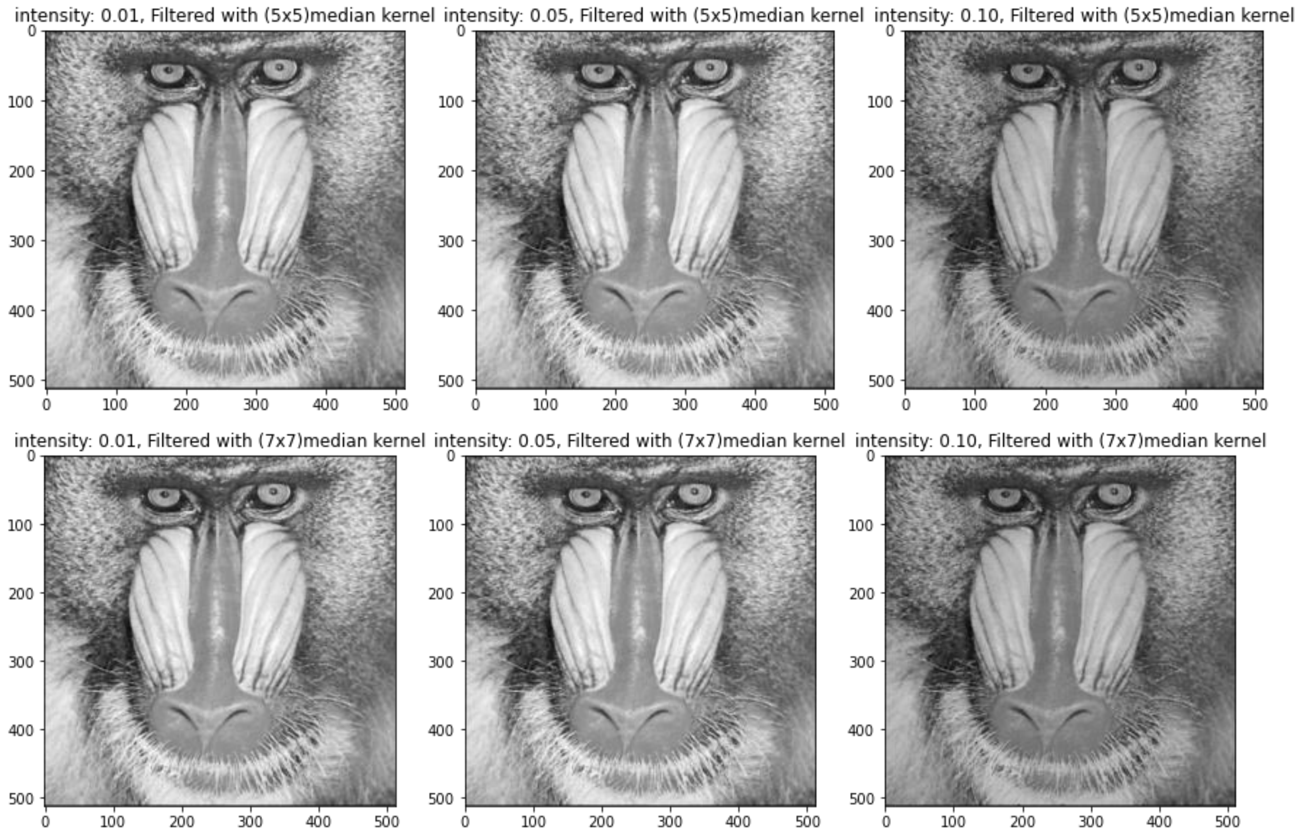
5) Take the Baboon image from 'sample images1' directory, and add salt and pepper noise. Create three different outputs where intensity of the salt and pepper noise differ for each output. Your outputs should be similar to the images in 'sample images2' directory. Then apply 3x3, 5x5 and 7x7 median filters to the images that you have created. Put the results into your assignment report.

Answer 5:

In this question, I read Baboon image and added salt and pepper noisy in different intensity using salt_and_pepper(img, intensity) function.
This function iterate each pixel in an image and creates random values between 0 and 1.
Then it check if given intensity bigger than created random number or not. If bigger, changes the iterated pixel to white or black (adds salt or pepper). For example: If Intensity is 0.7 then the probability of being bigger than random number is 70% and the noise will be much.
If intensity is 0 then it won't be bigger than created number and the output will be the image itself.
The noise color specified by a counter. If counter variable is odd the added noise will be white, else will be black. So, salt and pepper noisy will be added randomly.
Then I printed the outputs using subplots function.
After that I applied to the images median filters I have created and so the noise has removed.

Outputs:


intensity of noise: 0.01
intensity of noise: 0.05
intensity of noise: 0.10
intensity: 0.01, Filtered with (3x3)median kernel
intensity: 0.05, Filtered with (3x3)median kernel
intensity: 0.10, Filtered with (3x3)median kernel
0
100
200
300
400
500


intensity: 0.01, Filtered with (5x5)median kernel
intensity: 0.05, Filtered with (5x5)median kernel
intensity: 0.10, Filtered with (5x5)median kernel
intensity: 0.01, Filtered with (7x7)median kernel
intensity: 0.05, Filtered with (7x7)median kernel
intensity: 0.10, Filtered with (7x7)median kernel
0
100
200
300
400
500