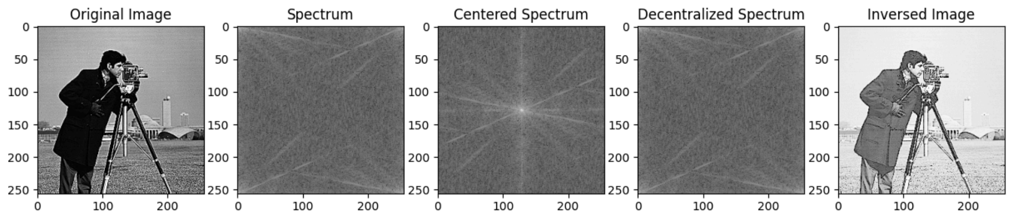
Ashkan Zarkhah
610399196

You can find all images and plots in the “plots” folder, and all the programming codes, in the “code” folder.

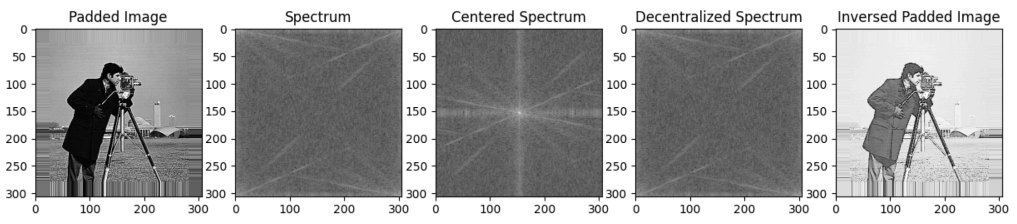
Report:

- **Filtering in Frequency Domain:**
    a. First, we develop a function to load an image and convert it to a grayscale image. Then, we develop a function to first calculate the frequency spectrum with using the "fft" function of the numpy library, then it calculates the centered spectrum using the "fft shift" function of the numpy library and then it inverses the "fft shift" to calculate the decentralized spectrum and at the end, it inverses the "fft" function to invert the image and it returns all steps of the computation. With developing these two functions, now we can apply our first question in the image and plots are:

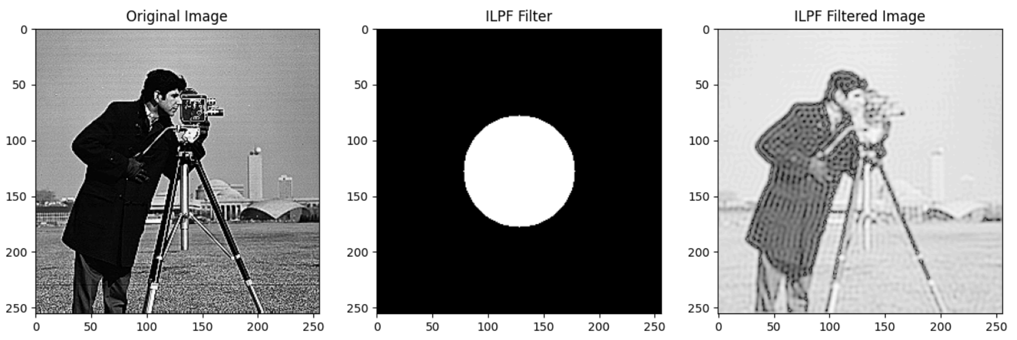


    b. First, we develop a function to apply nearest row and column intensities padding using the "copyMakeBorder" function of the cv2 library, and we apply the padding on all 4 sides of the image. With having this function and the previous functions, now we can first pad our image and then apply all calculations requested in question one.

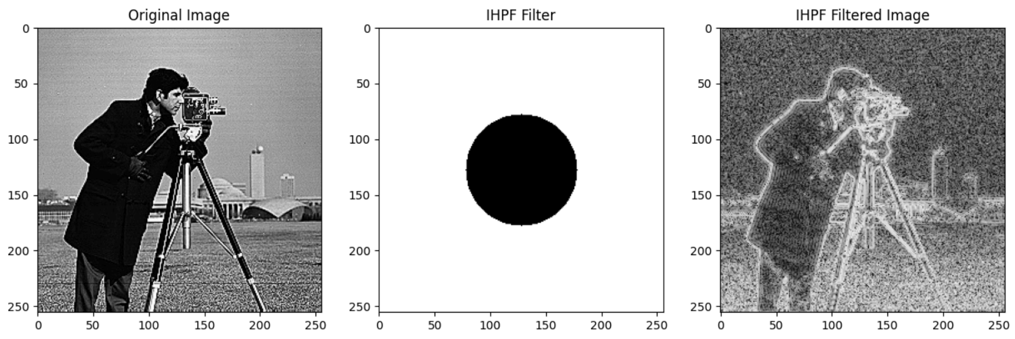
For the padding we use the padding size of 25 and the results are:



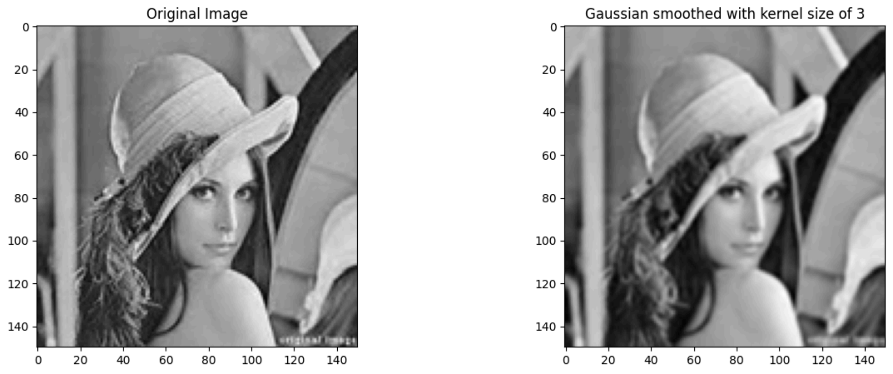
c. To apply ILPF filtering on an image, we first need to calculate the corresponding filter matrix. So, we first develop a function to calculate our ILPF matrix. Then we need a function to move our image into the frequency domain and apply the center spectrum on it and then apply the ILPF filtering matrix on it. So, we develop a function to apply "fft" and "fft shift" on our image, and then it applies the inputted filter on it, and at last it inverses the "fft shift" and the "fft" transmission and returns the filtered image. The results are:



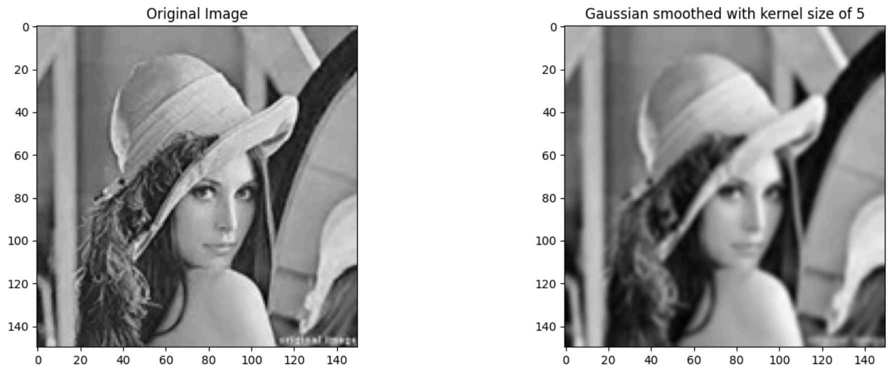
Also for the IHPF matrix, we develop another function, and it creates a filter exactly complement of the ILPF matrix. And with using the function we had for applying a filter on an image, we apply the IHPF filtering on our image too. the results are:



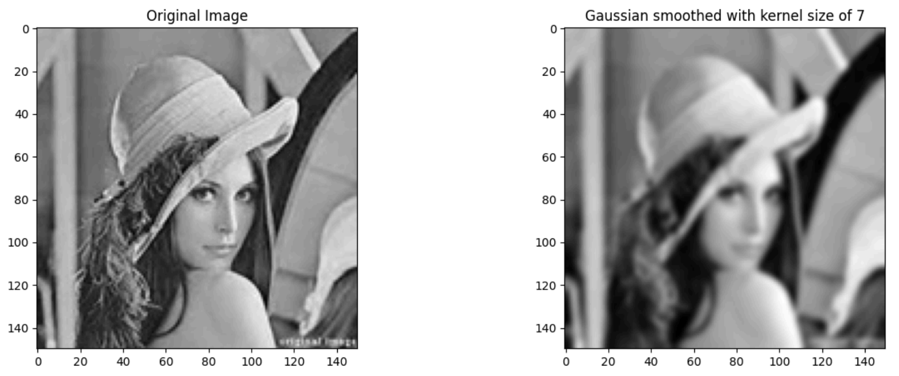
d. At last, for applying Gaussian smoothing, we develop a function to take the image and the kernel size and apply a Gaussian smoothing with the variance equal to zero with the function "GaussianBlur" of the cv2 library. As it is predicted, with increasing the kernel size, our pictures smooths more and gets more blur. the results for kernel size of 3 is:



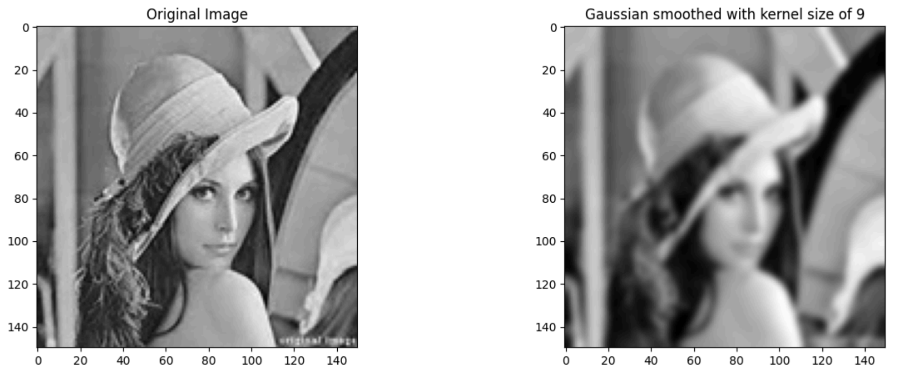
and for the kernel size 5 we have:



and for the kernel size of 7 we have:

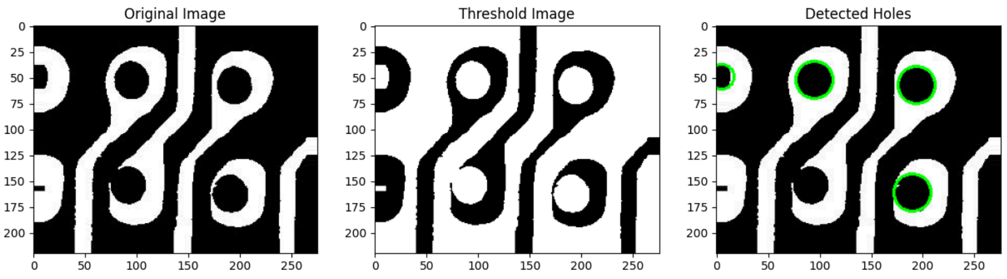


and at last for the kernel size of 9 we have:

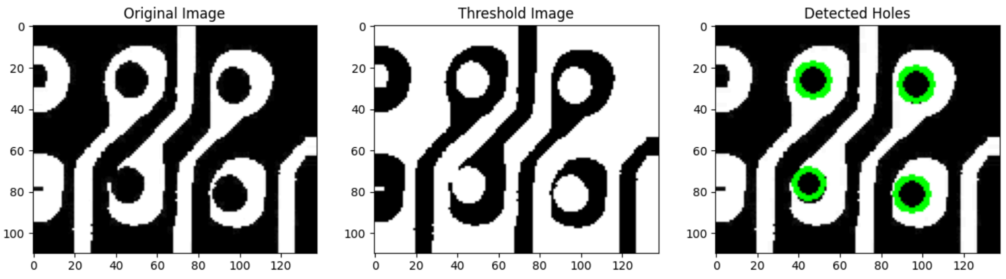


- **Morphology Operators:**
  a. To find the holes, we use two different methods, first one calculates more obvious holes, and the second one can find all holes. With using both of these methods, we can first find our holes' average diameters and by looking for holes with about that size, we can find all holes.
     In the first method, we first pad our image so that the neighboring holes gets more clarified. Then we apply a thresholding on our image to make it binary. Then we use the "findContours" function of the cv2 library to find all contours on the image. Then to visualize the holes we have found, we convert it to a BGR image and we loop over all found contours, and select the ones that are not too big or too small. Then, for the right contours with a good amount of area, we find the smallest enclosing circle over them and we draw and save their diameters. We can see that the results are:

```
Number of holes: 4
Diameters: [36.83182144165039, 36.730995178222656, 24.166292190551758,
36.935245513916016]
```



As we can see, we have missed only one of the holes, and also we have that its diameter is about 36. So know we use the second method to find this hole too.
In the second method, we first apply maximum pooling on the image, and also we use the "morphologyEx" function of the cv2 library to find small collisions that are left and separate them from each other. Also, when we are looking for our holes, we look for a smaller set of areas(because maximum pooling enlarges the background). We can see that the results are:



Now we have detected the only left hole too.

Key findings:

- **With image transformation, there is a risk of losing data**
- **Gaussian smoothing blurs the image**
- **Performing filtering multiple times has different outputs**
- **Finding holes that collide with the background is very hard and non-accurate**
- **Padding can help to find information lost because of the image size restrictions**

References:

- **OpenCV (cv2) library (for most of the calculations)**
- **Matplotlib library (for visualizing images and histograms)**
- **Numpy library (for mathematic operations)**