Report on Laboratory 2
Max, Min, Median, Gaussian filters and Histograms

Computer Vision UniPD

Alessandro Viespoli 2120824

March 2024

Setup and instructions to run the code

OS: Linux (Pop!_OS)
Enviroment: CLion
CMake: 3.27.8
OpenCV: 4.5.4
The main code that elaborate all the tasks is named `main.cpp`. The header file `ImageFilters.h` can be found in the `/include` directory, while its source code in `/src`. The images to be load are found in the `\Images` folder.
To run the code simply extract the content of the .zip file and run in the terminal:

1. cmake .
2. cmake --build .

This will create a Lab2 executable; run it and pass as `argv[1]` the ABSOLUTE PATH where the `\Images` folder is located.
All generated images are saved in the `\Images` directory; however not all generated images are shown as windows. Each time an image appears the execution is paused until a key is pressed.

Task 1

Figure 1: Garden grayscale image

Task 2

The `maxFilter()` and `minFilter()` functions receives as input the source image and the kernel size. The functions throw `std::invalid_argument()` if the kernel size is even. In order to apply the filter pixel by pixel the cycle start at the top-leftmost pixel (index (0,0)), then an internal for-loop look look for the nearby elements according to the kernel size. The functions do not apply any padding to the border, hence when the indexes are out of the image's border the iteration in the for loop is skipped.

For the **Lena image** the personal best is obtained with a min filter with kernel size 3. With larger kernel size the image becomes unrecognizable, whereas the min filter is chosen because personally it is easier to see the characteristics of the image; possibly caused by the fact that the image is darker to its counterpart (max filtered image kernel size 3).

Figure 2: Lena min filter, kernel size 3

For the **Astronaut image** the personal best uses both min and max filter with kernel size 3. This is the case because since the image is affected by a salt and pepper noise, hence neither the max or min filter can remove completely this kind of noise; therefore one filter can not remove both the salt or the pepper noise at the same time. Larger kernel sizes make the noise even worse.

Figure 3: Astronaut max filter, k = 3

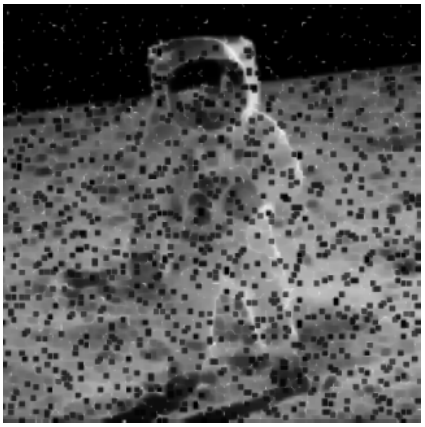

Figure 4: Astronaut min filter, k = 3

For the **Garden image** the best result in my opinion to eliminate the electric cables without corrupting the image too much is the one with max filter and kernel size 5. Higher kernel size values would over degrade the image.

Figure 5: Garden max filter, kernel size 5

Task 3

The `medianFilter()` function receives as input the source image and the kernel size. The functions throw `std::invalid_argument()` if the kernel size is even. In order to apply the filter pixel by pixel the cycle start at the top-leftmost pixel, then an internal for-loop saves in a `std::vector` the nearby elements according to the kernel size; then it is sorted and the median value is taken. The function do not apply any padding to the border, hence when the indexes are out of the image's border the iteration in the for loop is skipped.
For the **Lena image** the best one is obtained kernel size 7. Even though the image is blurred, the corruption of the initial image is not prevalent anymore. For

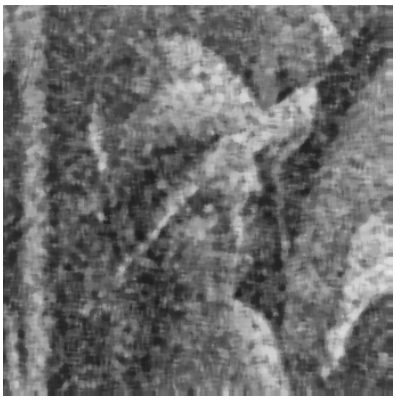

Figure 6: Lena median filter, kernel size 7

the **Astronaut image** the median filter completely removes the salt and pepper noise at kernel size 9. Clearly, the image is not very sharp and detailed.

Figure 7: Astronaut median filter, kernel size 9

For the **Garden image** a kernel size of 9 almost completely removes the electric cables, although higher kernel size values would remove them even more in spite of image quality.

Figure 8: Garden median filter, kernel size 9

Task 4

Figure 9: Lena gaussian smoothing filter, kernel size 7

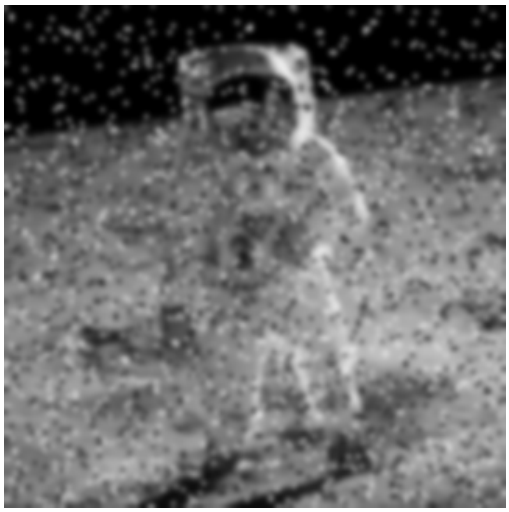

Figure 10: Astronaut gaussian smoothing filter, kernel size 9

Figure 11: Garden gaussian smoothing filter, kernel size 11

Task 5

The histogram of the Garden_grayscale image has been calculated using the `cv::calcHist()` function with with 256 beans and [0, 255] range.

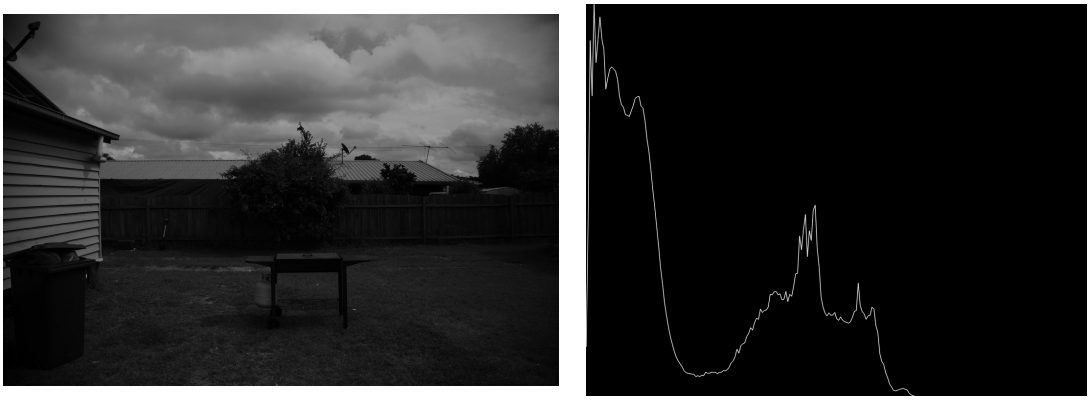

Figure 12: Histogram of the Garden_grayscale image

Task 6

In order to visualize an equalized histogram, first the equilized grayscale image is obtained with the function `cv::equalizeHist()`, then its histogram is computed with the same functions and options as task 5.

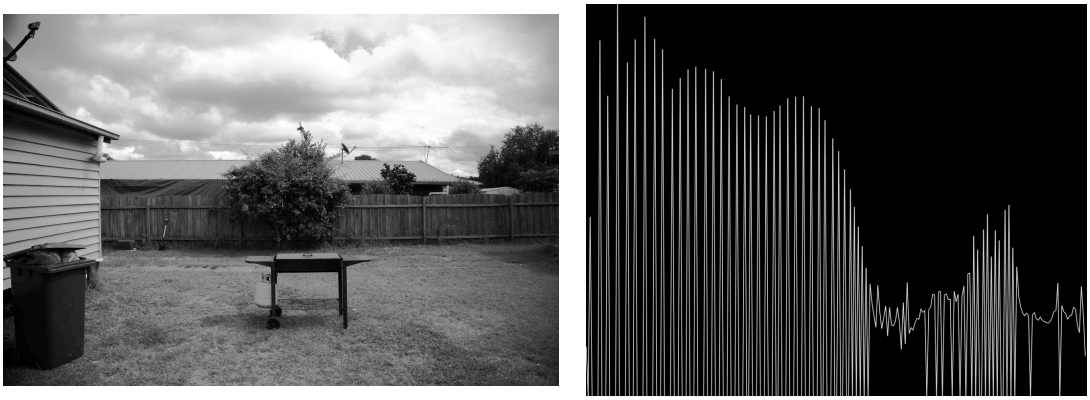

Figure 13: Histogram of the Garden_grayscale equalized image