An Implementation and Analysis of Median Image Filtering

Léo PERNET-MUGNIER
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai
leo.pernet-mugnier@insa-lyon.fr

***Abstract*—Since digital images are everywhere in our daily life, Digital Image Enhancement is useful to automatically improve the perceived quality of images for human beings. In this paper, we describe and implement an algorithm that performs Median Filtering. Our algorithm has good visual results but is very slow compared to existing one. Also, we perform some tests to see if Median Filtering could enhance other situations than Salt and Pepper Noise, but we can't find any other good situation.**

I. Introduction

Median Filtering is known to be very efficient against salt-and-pepper noise [1]. Its principle is, for each pixel, to find and sort the values of it and its neighbors as an array. Then we replace the pixel value by the median value of this array. The parameters are all about choosing which neighbors we want to consider. In this paper, we call *Window* this neighboring area around the pixel. The Window has a size and a shape, which we describe with a matrix. For example:

$$w = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{1}$$

Here, *w* describes a 3x3 cross shaped window.

Salt and Pepper noise is a uniformly distributed noise where some random pixels on dark background turn white and some random pixels on a light background turn black.

In this paper, we propose an algorithm that could perform median filtering for any size and shape of window.

II. The Algorithm

The principle of our algorithm is in several layers.

First, we browse the image pixel by pixel using imbricated loops. Then for every pixel, we browse the window around, using once again imbricated loops. We select every value as described by the window parameter. Figure 1 Shows an example with a 3x3 cross window.

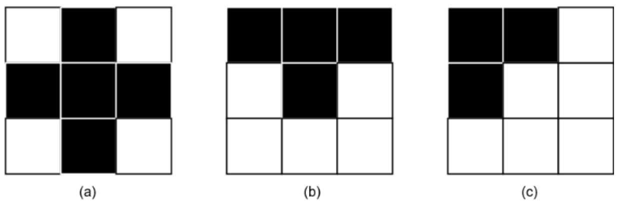

Fig. 2. Example of how a cross shaped window (a) fits the boundaries in our algorithm.

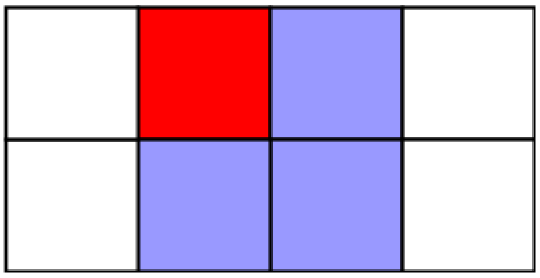

Fig. 3. An example of even window matrix, with its center in blue, and the location of the current pixel, in red.

The major problem is boundaries, and they are different ways to deal with it. First, we could just ignore it. After all, it only concerns the borders of the image, which we could sacrifice to gain some calculation time and energy. In this paper, we don't use this method because we want to preserve the image as much as possible. Another solution is to adapt the window to the boundaries. For this, we implemented the simplest way, i.e. truncate the window to fit the border, as shown in figure 2.

Another problem is where do we locate the current pixel in the window matrix. This question is trivial for a square odd matrix, because the matrix as a single center. But for even window matrices, the center is a 2x2 square. We arbitrary decided to locate the current pixel in the left-top corner of this center, as shown in figure 3.

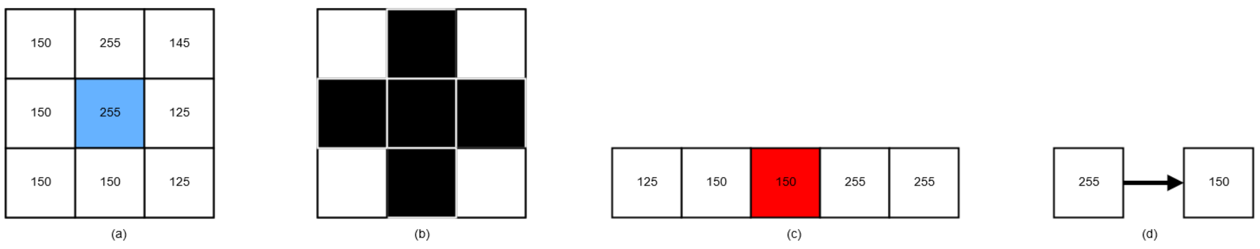

Fig. 1. Let's consider (a), a pixel and its neighbors. (b) shows our window matrix: the black cells are the ones we want the values. The algorithms grab the wanted values and sort them (c). Finally, we replace the pixel value (d) with the median value of this array.

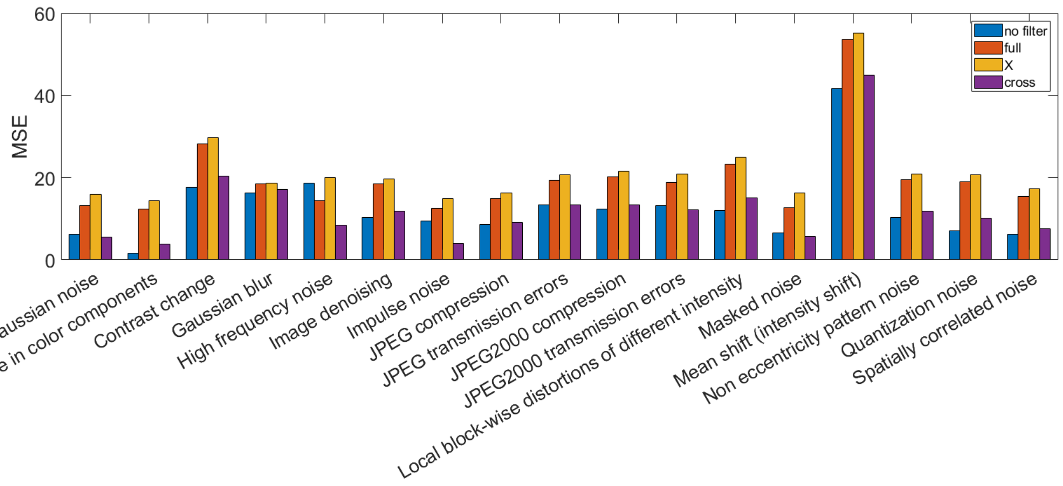

Fig. 5. MSE depending on image alteration, for three shapes of 3x3 window.

III. Implementation and Test

A. Pre-implemented median filtering in MATLAB

The function *medfilt2* performs a median filtering on a 2D matrix. We can define the size of the window, but not its shape: it's always a full window. To use different shape, we can use the function *ordfilt2*, which basically replaces a pixel value with new value that can be computed using its neighbors.

We ensured that those two pre-implemented solutions are giving the same results, for full windows only.

B. Setup

We compare the two algorithms with 3x3 windows, using three window shapes: full, X, cross (also called +) and also with a 4x2 full window, like in figure 3. We use MSE as metric to compare the two solutions. MSE is adapted here, because we only want to know if there are some differences between the two implementations. The image used is a salt-and-pepper noised image. Finaly, we did the time measurements using the MATLAB *Run and time* tool.

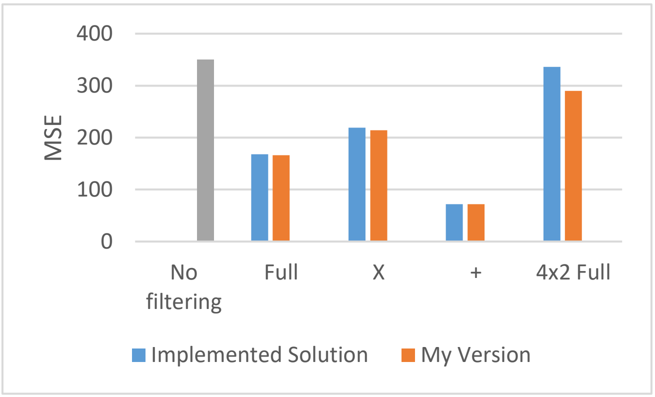

Fig. 4. MSE depending on Median Filtering type

C. Results

The MSEs are shown in figure 4. With the 3x3 windows, the MSE are almost the same for the two implementations. Visually, all the Salt and Pepper noise disappears in all the cases. However, for the 4x2 window, we notice much larger difference. This is certainly caused by different policies of current pixel localization in the window, c.f. *II*. On figure 4, MSE is lower for our implementation, which could mean our solution is better. Actually, this occurs because the pre implemented solution generates some one-pixel black strips at the right and bottom of the image, which has a great impact on the MSE, but not really on the image quality. In the same time, our solution distorts and blurs the image much more than the pre-implemented one. So, none of the two solutions are perfect.

In the other hand, there is a huge different of computation time between the two solutions. The pre-implemented solution takes between 0,006 and 0,027 seconds per image when my implementation takes between 1,7 and 2 seconds per image.

So, my solution is about a hundred times slower than the pre-implemented one for similar results.

IV. Using Median Filtering for other image defaults

As seen in *I* and *III*, Median Filtering is very efficient against salt and pepper noise. In this part, we wanted to see how could Median Filtering be useful against other alterations.

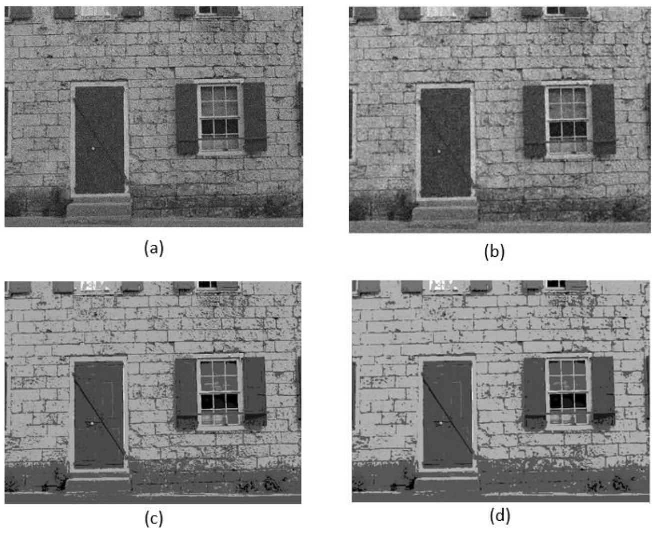

Fig. 6. Median Cross 3x3 filtering on High Frequency Noise (b) and on Impulse Noise (d). (a) and (b) are respectively raw High Frequency Noised and Impulse Noised.

A. Setup

We use the *Tempere Image Database (TID)* [2]. It’s a database of 25 different images, with each 17 kinds of alterations and 4 alteration levels, for a total of 1700 images. The purpose is to compare each altered image with the reference. Then, we Median Filter all those altered images, and we compare them again with the reference image. The purpose is to see if median filtering could be efficient against image default other than Salt and Pepper noise.

By simplicity and lack of time, we only use MSE as image assessment method. So, we keep in mind that the following results are not too be fully trust as human perceived quality. Also, for simplicity, we decide to turn colored images in grayscale.

B. Results

Figure 5 shows that Median Filtering bring some good results only for two alteration type: High Frequency Noise and Impulse Noise. But when comparing the images, the enhancement is still not impressive at all, as shown in figure 6.

Another interesting result is that for every kind of alteration, Cross Shaped Window is much better than Full Window and X Shaped Window. A simple visual observation confirms this result.

So, we weren’t able to find any other case than Salt and Pepper noise where Median Filtering is useful.

Conclusion

The MATLAB scripts used in this paper are all given with it.

In this paper, we designed and implemented a fully operational Median Filtering algorithm. Despites its slowness, our solution is more flexible than pre-implemented ones and may be useful when using exotic filtering window. But in most of the case, the pre implemented solution is much faster for similar results.

Also, we tested Median Filtering against 18 types of images defaults, and we figured out that the filtering was only useful in the case of Salt and Pepper noise.

Finally, we noticed that cross shaped window is better than full window and X-shaped window in every tested case.

Acknowledgment

This paper was deeply supported by Aretha Franklin’s and Ray Charles awesome music. Thanks to my flat mates for bearing it.

References

[1] Hsieh, Mu-Hsien, et al. "Fast and efficient median filter for removing 1–99% levels of salt-and-pepper noise in images." *Engineering Applications of Artificial Intelligence* 26.4 (2013): 1333-1338.

[2] N. Ponomarenko, V. Lukin, A. Zelensky, K. Egiazarian, M. Carli, F. Battisti, "TID2008 - A Database for Evaluation of Full-Reference Visual Quality Assessment Metrics", Advances of Modern Radioelectronics, Vol. 10, pp. 30-45, 2009.