2D filters methods for image processing in bioinformatics

Formation Master 2 Bioinformatique **Cours 4TBI901U**
Auteurs Gary BOUCHENTOUF * Tristan FRANCES Thomas MAUCOURT
Responsable UE Bioinformatique Structurale Jean-Christophe TAVEAU

Introduction

The development of technologies allowing images captures has permited scientists to use new devices in many domains such as astronomy, geology, biology, etc... (*Add reasons why they use this tools*). The main problem to deal with images is their processing since a raw format image is rarely sufficient to make analysis. Actually there are additional steps required which are called image processing. After treatment the images are ready for analysis and interpretations. There are many possibilities to process an image according to the final result wanted. In this report we will focus on 2D Linear Filters. Filters are used to convert an input image into an output, based on the convolution product principle (Figure 1).

$$f[x,y] * g[x,y] \quad = \quad \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

Figure 1. Basic convolution product formula. This mathematical formula can be seen as the combination of two matrix in image processing.

Basically, a convolution product is the result of two matrix combinations. Convolution operation needs an image and a convolution mask also called kernel. An image is actually a bidimensional matrix in which, each square corresponds to a specific pixel value. Its dimension has a value of *width x height*. This matrix will be combined with the convolution mask which is a matrix *j x k* with *j* and *k* odds values. It is important to keep in mind that if the kernel is not symetrical, it needs a rotation of 180 degrees. The result of the convolution product gives a new value to the central pixel of the kernel. This value is based on the weighted average of the surrounding pixels. It leads to a new image with modified pixels values.

Through this report we will discuss three filters using this mathematical principle in ImageJ[^IMAGEJ] software. The first operation studied will be the convolution, then the Gaussian blur and finally the mean filter.

1. Material and Methods

1.2. Convolve

We previously define that convolution needed an image and a kernel to be performed. That raises a problem of coverage between the mask and the targeted sample of the picture. Indeed at the borders of the image the kernel can have some of its parts outside the picture (Figure 1). To solve this issue several possibilities can be applied :

- adding a stripe of pixels with a value equals to 0 (black) around the image
- extending the borders by mirroring the edges
- starting the processing at position $(X_{(kw-1)/2}, Y_{(kh-1)/2})$ with *kw* and *kh* respectively the kernel's width and height

As convolution is a well defined mathematical principle very used in many domains and especially in image processing, several methods have been developed to improve the computation. We can find four main implementations of convolution :

- basic approach
- convolution with separable kernels
- recursive filtering
- fast convolution

First the "basic approach". It is simply based on the definition of convolution. It computes the inner product of the current sample of the image and the kernel. The value obtained is then stored into the central mask's pixel. For information, the algorithmic complexity of such method is *$O(N^2)$*. This approach is still widely used due to its simplicity and the possibility to parallelize the processing[^SKA2008].

An other way to perform convolution is allowed in the case of specific kernels called separable[^ROB1986]. A kernel is separable in 2-Dimensions when it can be decomposed into two vectors (a column vector and a row vector) (Figure). These kinds of kernels are found in Gaussian masks for example. Such kernels decrease the complexity because they allow to separate the basic convolution process into two simpler convolutions.

The third method available to apply convolution is called recursive filtering[^BJA2002]. This approach is based on the dependence of a sample (rank *n*) with the previous one (rank *n-1*, *n-2*, ...). It needs a step to define the recursive formula needed for the convolution product wanted. This can be very hard task. Thus it is not a very used method among the others one.

The last approach we will discuss is the most popular and is actually the one implemented into ImageJ software. The fast convolution[^JAN2000] is based on the frequency domain[^RNB2006] unlike the previous we discussed that were used into the spatial (or time) domain. This method go through an extra step by using the Fast Fourier Transform (FFT)[^RNB2006] on the input signal before performing convolution. According to a mathematical principle a convolution into the frequency domain is equivalent to a product of

Fourier Transforms (Reference). This means that a convolution into the spatial domain is equivalent to a multiplication into the frequency domain. Thus :

$$\textit{kernel} * \textit{sample} = FFT^{-1}(FFT(\textit{kernel}) \times FFT(\textit{sample}))$$

('*' stands for convolution and 'x' for multiplication)

By looking for plugins implementing the different methods described above, we have been able to locate just one plugin different from the one used by default in ImageJ. This plugin is called **Real_Convolver.java** (credit to Wayne Rasband) and implements the simple convolution into the spatial domain. It allows the use of convolution only for 32 bits images, so the totality of the benchmarking has been realized on this type of image.

1.3. Gaussian Blur

The Gaussian Blur also known as Gaussian Smoothing operator is a convolution operator used to blur images and remove details and noise. In this sense it is similar to the mean filter, but differs by using a different kernel. Differently sized kernels containing different patterns of numbers give rise to different results under convolution. For instance, Figure X shows a 5x5 kernel that implements a Gaussian Blur filter.

$$\frac{1}{273}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 7 & 4 & 1 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 7 & 26 & 41 & 26 & 7 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 1 & 4 & 7 & 4 & 1 \\ \hline \end{array}$$

Gaussian Blur Kernel

The Gaussian kernel is named after Carl Friedrich Gauß (1777-1855), a German mathematician. They are part of separable kernels described above and are composed of Gaussian values. Using such kernel allows a strong decrease of the algorithmic complexity and thus a quicker processing. Gaussian Blur filtering consists in realising a convolution on a picture with a Gaussian function. The values in each row of the kernel come from the Gaussian function.

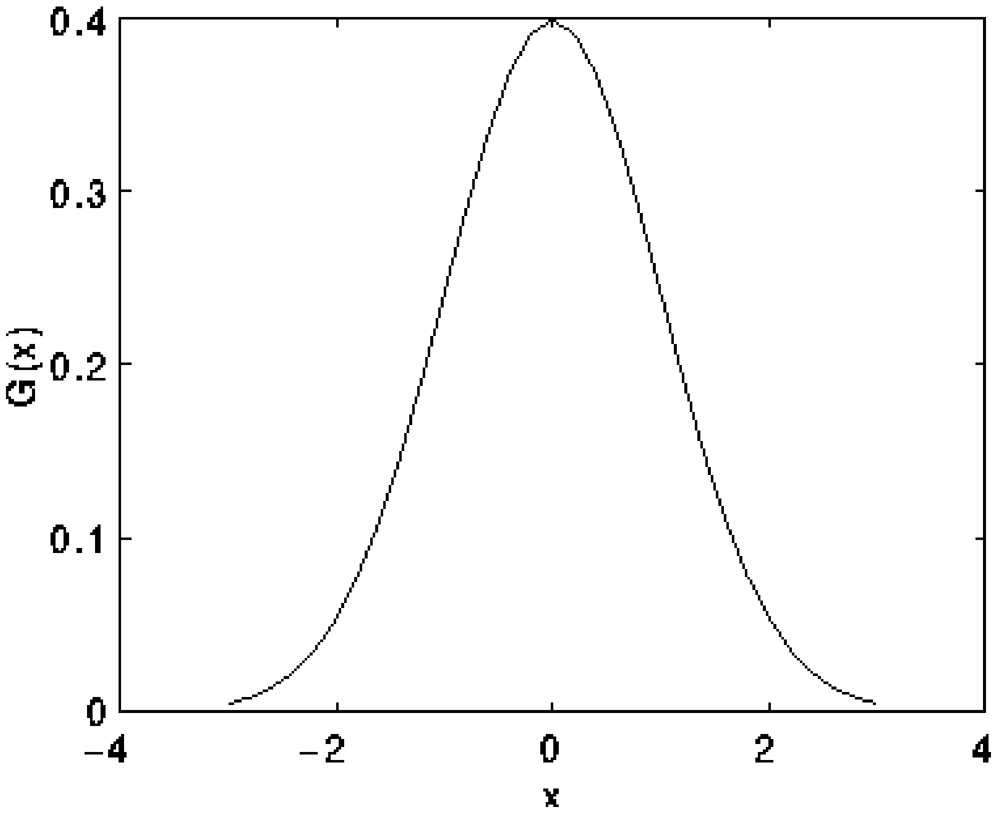
$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

Gaussian Blur 1-D Equation

In the equation, x is the distance from the origin for the absciss, y is the distance from the origin for the ordinate, and σ is the standard deviation of the Gaussian distribution.

Again, to speed up image processing, algorithm have been developed. Here we will compare the ImageJ default Gaussian Blur filter to a JAVA plugin, Accurate Gaussian Blur. This last has been implemented for high accuracy treatement especially for 32-bits images. Also, these methods encounter the same trouble as the convolution in term of image edges.

Gaussian Blur 1-D Distribution shape



Gaussian 1-D function

1.4. Mean filter

Among the linear filters, the most common is the mean filter beacause it is easy to implement and also reliable. As it has been said before (see section *1.3 Gaussian blur*), it is a smoothing filter which purpose is to smooth an image by blurring and remove details and noise. Doing so, a convolution mask is used, it can be with various shapes (square, rectangular or circular) and in vast majority the weights in the kernel are uniforms (meaning the values in the kernel are the same), but those can also be triangular (i.e. inversely proportionnal to distance from the input sample). [^MAT2007]

For example if S_{xy} represents the set of coordinates in a rectangular sub image window of size $m \times n$ centered at point *(x,y)*, the arithmetic mean filtering process computes the average value of the initial image *g(x,y)* in the area defined by S_{xy}. The value of the final image *f* at any point *(x,y)* is simply the arithmetic mean computed using the pixels in the region defined by S_{xy}. In other words :

Equation of the arithmetic mean filter

This operation can be implemented using a convolution mask in which all coefficients have a value of *1/(kw x kh)*. A mean filter simply smoothes local variations in an image. Noise is reduced as a result of blurring. The main problem of this filter is that noisy pixels (including anomalous spikes) are weighted the same as all the other pixels in the kernel. [^GAJ2011] Because it uses a convolution kernel, we find the same issue as describe in section *1.2 Convolve*, thus the same solutions can be applied here for the edgesof the picture.

During our research, we did not find any plugins performing mean filter in a different way than the one implemented in ImageJ software. That is why the benchmarking were realised only with the mean filter already implemented in ImageJ.

1.5. Benchmark

Benchmarks were realized based on time of processing ont the same computer. A JavaScript script was implemented to automatize the process (Supplementary Data 1). First a warmup phase consisting in running 20 times the chosen operation is performed. After that the script switch to the testing phase consisting in 20 runs of the selected. This process is used on the same RGB image with a size of *1920*1200*. This image was converted into 32 bits, 16 bits and 8 bits greyscale images and the benchmark was performed on each of these pictures for each filter when it was possible. To get more data on other sizes of image, the initial picture was also resized two times. This lead to three images sizes : *1920*1200*, *960*600* and *512*320*. The results are displayed in the ImageJ log console and saved into a csv file. In this case, time processing of Convolve, Mean and Gaussian blur were calculated. An R script allows to get a boxplot of the results showing the different quartils and the mean of data for the warmup and the testing phase. The process has been repeated for the alternatives plugins found for the different filters.

Benchmarking in JavaScript is really complicated since the results are very variable and that they depend a lot on the state of the computer used for the test (Référence Article Taveau). Moreover, implementing a good benchmark implies a deep understanding of the execution processes and the interpreter. Thus the results obtained by our method have to be taken with care and as informative.

2. Results

2.1. Convolve

We compared two plugins, one realizing the "original" way to make a convolution and another one getting an extra step of FFT to get the result. The first thing important to note is that the output images from one method and the other are the same (Figure).

Figure X. Convolution performed with ```Real_Convolver.java``` (on the left) and the ImageJ default convolution filter (on the right)

First, the time of processing for the different images size were computed using R (Figure). This shows a diminution of processing time the smaller the image gets.

Figure X. Comparison of convolution processing time for 3 types of images. The reduction of pixels leads to a decrese in time processing for the ImageJ implementation of convolution and for the plugin ```Real_Convolver.java```.

In a second time a comparison between **Real_Convolver.java** and the convolution plugin implemented into ImageJ was performed (Figure). We can see that the plugin using the step of FFT, the ImageJ's one, takes less time to process an image no matter the considered size.

Figure X. Processing time for different images sizes and comparison between the ImageJ convolution and the `Real_Convolver.java`. From left to right we can see the diminution of time processing for both plugins but with a better performance for the FFT extra step processing.

2.2. Gaussian Blur

In regards to Gaussian blur, after having executed the two plugins on a same image, we have compared their execution time and results are introduced in figure x. Here, just four plots are represented but the trend in the twelves graph generated were the same.

FigureX : Representation of the benchmark of ImageJ gaussian blur filter and accurate gaussian blur plugin

The general trend for each box plot if we compare both processes is the same. The Accurate plugin has an execution time always longer than the imageJ default filter.

If we take look at the values on ordinate, we notice a time diminution when we apply the filter and plugin on smaller images. This fact is also true for the image type, but this is the image size which has the more influence on time execution.

2.3. Mean filter

The results of this benchmark are represented in *FigureX*.

FigureX : Representation of the benchmark for the mean filter of ImageJ

We obtain 4 graphics (one per type of image) which contains 3 boxplot (one per size of image) comparing the difference of time processing. We can see there's no real differences for the time processing between the type of image. However it is not the case for the size that shows the bigger the image is, the longer the processing last.

3. Discussion

The several benchmarks performed allowed us to observe certain behaviors for the different plugins tested.

For the convolution part, we have seen that the algorithm using an FFT before the convolution (ImageJ) seems to have better performances than the Real_Convolver.java. As expected we saw a decrease of processing time the less the image is big. That can easily be explained by the fact that it needs less computing for tiny image because there are less pixels to process.

The time difference between the Gaussian Blur filter and the Accurate Gaussian blur filter can be explain by the fact that this last has high accuracy. We can also say that images size have an influence on process duration. Same thing on image type.

Concerning the mean filter, the results of the benchmark do not allow us to conclude because of the lack of alternative plugins and the weak sturdiness of the benchmarking using JavaScript. This can be explained by the fact that the mean filter isn't a good noise removal, due to its simple algorithm. Several other methods, like Gaussian blur filter or median filter (no linear), have been developed to produce more efficients filters for noise removal.

Conclusion

Through this work, we were able to study the different process used for 2D filtering. The final goal is to implement the best algorithm into a "web ImageJ like" for each filter studied.

For the further steps, we decided to use the convolution algorithm found in ImageJ source code. We will also implement the recognition of separable kernels to increase a bit more the processing by creating two vectors that will be used to make the convolution.

We also consider to implement a Convolve Class which has the purpose to make easier the implementation of the Gaussian Blur and mean filter implementations. Thus, those would be implemented by simply calling this class and specifying the paramaters adapted in agreement with the operation to apply.

References

[^MAT2007]: MATT HALL. Smooth operator: Smoothing seismic interpretations and attributes. The Leading Edge. 2007 Jan;26(1):16–

20.

[^GAJ2011]: Gajanand Gupta. Algorithm for Image Processing Using Improved Median Filter and Comparison of Mean, Median and Improved Median Filter. International Journal of Soft Computing and Engineering (IJSCE) ISSN. 2011 Nov;Volume-1(Issue-5):2231–307.

[^BJA2002]: B. Jähne. Digital Image Processing. Springer, 5th edition, 2002.

[^JAN2000]: Jan J, Engineers I of E. Digital Signal Filtering, Analysis and Restoration. IET; 2000. 430 p.

[^RNB2006]: R. N. Bracewell. Fourier Analysis and Imaging. Springer, 2006

[^ROB1986]: Robert Hummel and David Loew. Computing Large-Kernel Convolutions of Images. Technical report, New York University, Courant Institute of Mathematical Sciences, 1986.

[^SKA2008]: S. Kadam. Parallelization of Low-Level Computer Vision Algorithms on Clusters. In AMS ’08: Proceedings of the 2008 Second Asia International Conference on Modelling & Simulation (AMS), pages 113–118, Washington, DC, USA, 2008. IEEE Computer Society. ISBN: 978-0-7695-3136-6.

[^IMAGEJ]: Rasband, W.S., ImageJ, U. S. National Institutes of Health, Bethesda, Maryland, USA, https://imagej.nih.gov/ij/, 1997-2016.