Lucas Versini

Image Denoising

Homework n°1 - Experimental report

1 Ponomarenko algorithm

1.1 Explanation of the algorithm

The goal of the Ponomarenko algorithm is to estimate the noise in an image.

- The idea is first to consider all square blocks of a given size in the image, and compute their Discrete Cosine Transform (DCT).
- For each block, the empirical variance of the DCT is computed based only on the low-frequencies (as the medium and high frequencies may contain image-dependent information, near edges or for some textures).
- The blocks are sorted in ascending order of the previously computed variance. The first K of them are being kept for the next step, where K can be found either iteratively, or set to 0.5% of the total number. The idea is that these blocks do not contain much low frequencies, and are therefore likely to contain mostly noise.
- Then, for each high frequency, a variance is computed by using the K blocks from previous step.
- The variance σ^2 of the noise is estimated as the median of the variances computed in the previous step.

1.2 Experiments on the algorithm

The authors of [1] present a way to estimate the noise as a function of the intensity for each color channel.
We run experiments for several images, containing different types of patterns.
For each image, we consider the original image, a uniformly noisy image (with parameters $A = 4900, B = 0$ in the online demo), and a signal-dependent noisy image (with parameters $A = 4900, B = 10$ in the online demo).

1.2.1 Flat image and chessboard

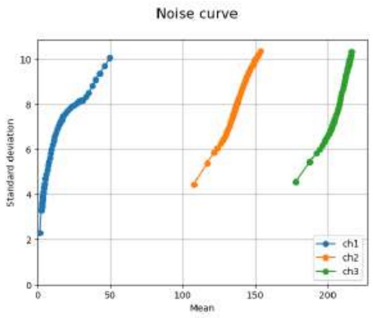
We first analyze the results for an almost uniform image, which can be seen in Figure 1, and a chessboard, which can be seen in Figure 2.

- For the original images, the algorithm detects some noise for both images. However, the estimated amplitude remains rather small, so the results are still rather good.
- When adding a uniform noise, the result is really precise: the retrieved standard deviation is very close to the real one ($\sqrt{4900} = 70$). The lines are almost horizontal, so the algorithm considers that the noise is essentially the same for all channels and all intensities, which is indeed the case.

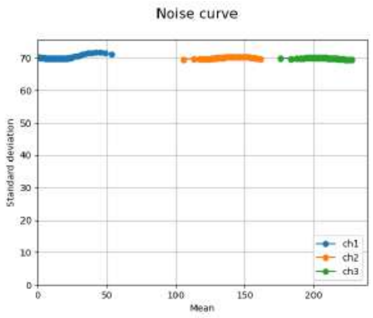
(a) Original image

(b) Uniformly noisy image

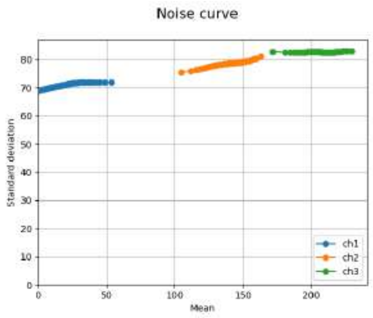
(c) Signal-dependent noisy image



(d) Original image

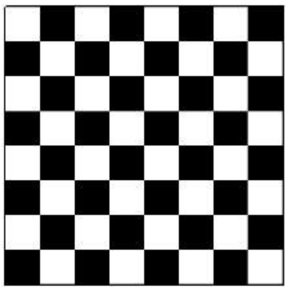


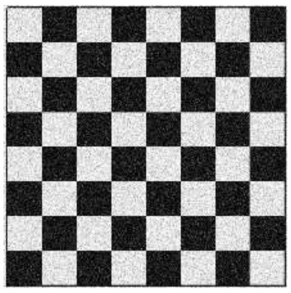
(e) Uniformly noisy image curve

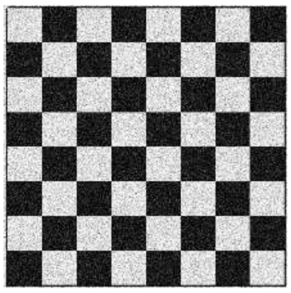


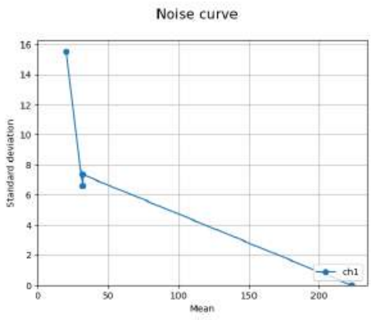
(f) Signal-dependent noisy image curve

Figure 1: Experiments on a first image

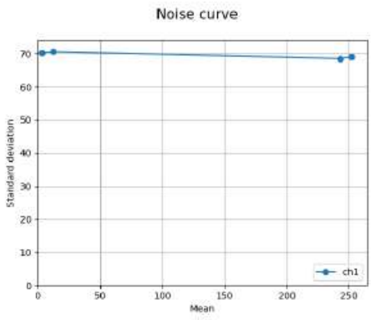

(a) Original chessboard


(b) Uniformly noisy chessboard

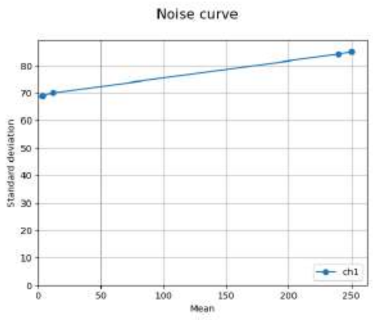

(c) Signal-dependent noisy chessboard



(d) Original noisy chessboard curve



(e) Uniformly noisy chessboard curve



(f) Signal-dependent noisy chessboard curve

Figure 2: Experiments on a chessboard

- When adding a signal-dependent noise, the standard deviation for the low intensity pixels is once again estimated to 70 approximately, and for the high intensity pixels it is between 80 and 85, which is approximately correct as well (since the real value is $\sqrt{4900 + 255 \times 10} \approx 86$).

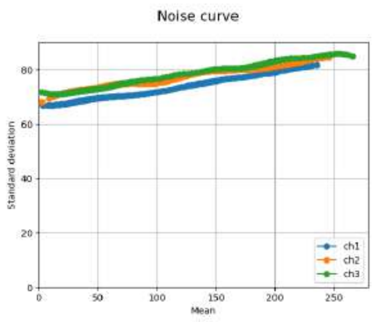
So overall, the algorithm performs quite well on these images.

For the chessboard, the high frequencies in the image are transitions between black and white, which are not present for blocks entirely black or entirely white. Thus the algorithm manages quite well to find these uniform blocks, where it can then accurately estimate the noise.

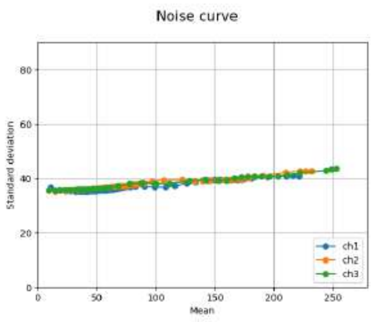
Also note that the algorithm is performed at **different scales**. When using a zoom factor of 2 for instance (see Figure 3), we find that the estimated noise is approximately divided by 2 as well, which is what is expected (since the noice really behaves this way).

(a) Original image

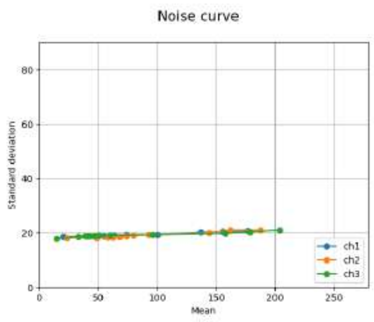
(b) Signal-dependent noisy image



(c) Scale 0



(d) Scale 1



(e) Scale 2

Figure 3: Experiments on a "real" image

1.2.2 Textured image

We now use an image which contains more texture than the previous ones, in Figure 4.
Here again, the results are quite good: the estimated noise in the clean image is low, it is about 70 in the uniformly noisy image, and a bit below 80 for the signal-dependent noise ($\sqrt{4900 + 120 \times 10} \approx 78$).
In all three cases, the estimated noise is approximately the same for all colors (i.e., all channels).

1.2.3 Impact of the parameters

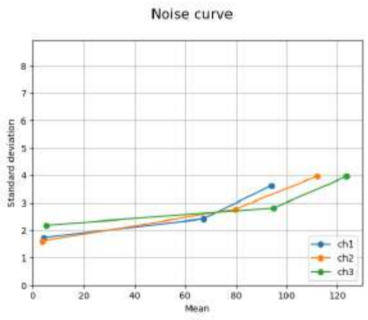
In the online demo, several parameters can be adjusted. Let us quickly discuss them.

- *The percentile.* If we use a percentile of 50%, the estimated noise in the uniform noise case tends to be overestimated (≈ 75), which makes sense since more blocks with high variance are then used.

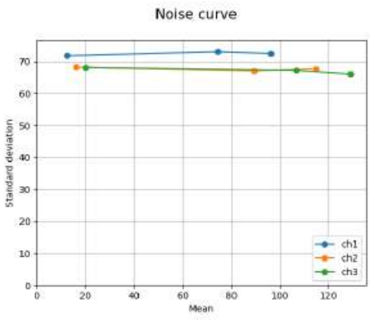
(a) Original image

(b) Uniformly noisy image

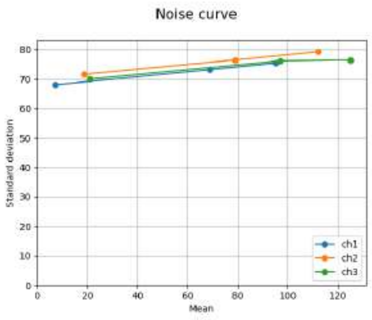
(c) Signal-dependent noisy image



(d) Original image



(e) Uniformly noisy image curve



(f) Signal-dependent noisy image curve

Figure 4: Experiments on a textured image

If we use the iterated version of the algorithm to find K, the results are globally identical.

- *The block size.* When using a larger block size, the number of blocks is reduced, and therefore the number of points in the curves is reduced as well. On the above images, no real difference was observed for a size between 3 and 21. However, if we used for instance a block size larger than the size of a chessboard square, the results would likely become less good.
- Median / mean. When using the mean instead of the median, the results are more or less the same. But of course, in general they will not be, and the mean may be too sensitive to extreme values (in the K variances used to estimate the noise).

2 Experiments on DCT denoising

We now analyze the DCT denoising strategy, presented in [2].

The main idea of this strategy is to separate the different channels, split the image into patches, apply DCT to these patches, filter out the frequencies with low amplitude (which are assumed to be noise), and apply IDCT before recombining the channels.

Figure 5 shows the an image of a chessboard after it was denoised. Note that the image was bigger than what is shown here, we have just zoomed in on the figure.
What can be seen (if you zoom in) is that there are artifacts everywhere: near the edges of course, but also in the middle of uniform areas.

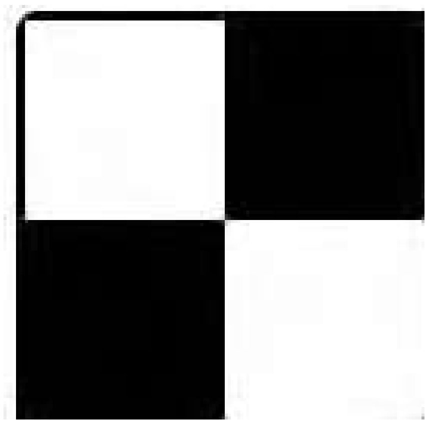
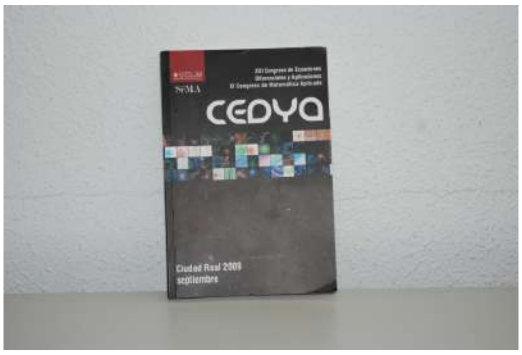
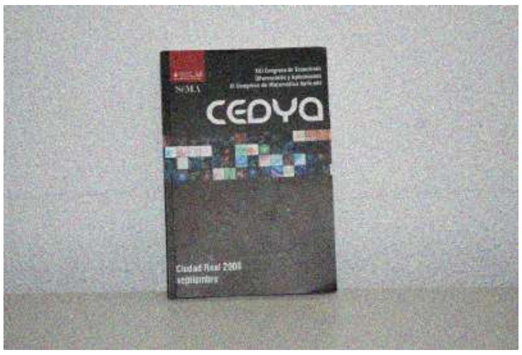


Figure 5: Denoised image (with DCT)

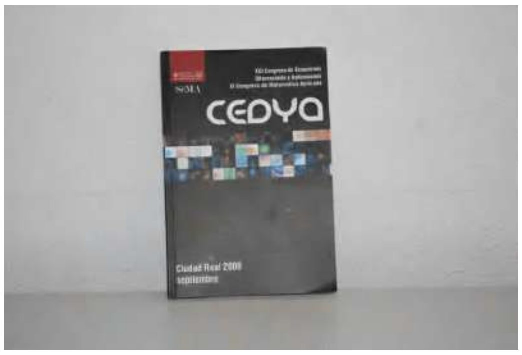
Figure 6 shows the original image of a book, as well as the noisy image, while **??** shows the denoised image, and the difference between the original image and the denoised image.
We see that the algorithm has mostly smoothed the wall in the background: though some artifacts are present, we see that the original asperities of the wall have disappeared, because they were mistaken with noise.
In fact, in most of the picture, we see that many things have been smoothed, leading to an image that is sort of blurry, which is explained by the fact that the details, with small intensity in terms of frequency, were suppressed.



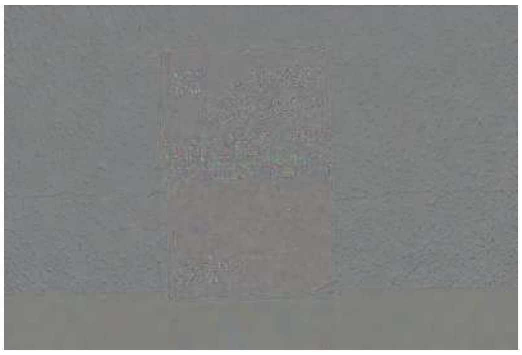
(a) Original image



(b) Noisy image

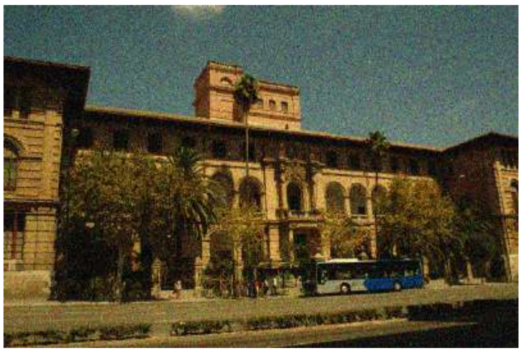


(c) Denoised image

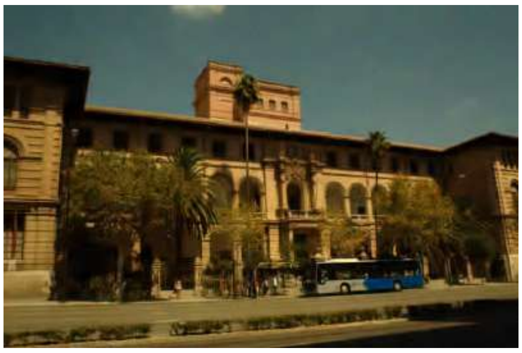


(d) Difference

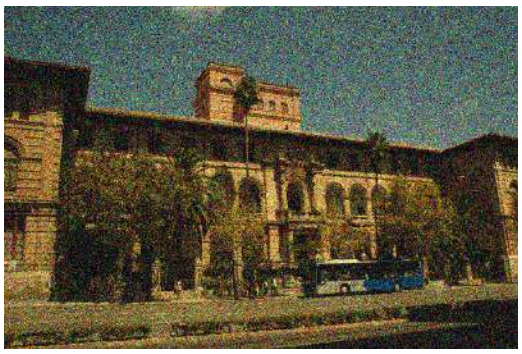
Figure 6: Original, noisy and denoised images of a book



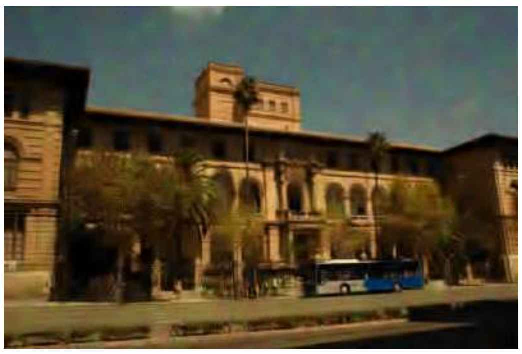
(a) Noisy ($\sigma = 30$)



(b) Denoised ($\sigma = 30$)



(c) Noisy ($\sigma = 60$)



(d) Denoised ($\sigma = 60$)

Figure 7: Experiments on the image of a building

This behavior can also be seen in Figure 7, where this time the original background (the sky)

was already uniform.
There again, most of the noise has disappeared from the sky, but some artifacts are still present when zooming in, especially (and logically) when $\sigma = 60$.

Therefore the DCT method tends to eliminate the noise, but:

- In uniform regions, some artifacts remain. This is because when using a small block size, the noise with low frequency cannot be handled correctly;
- In other regions, the result is not satisfying, since the image looks blurry, for instance with artifacts near the edges.

References

[1] Miguel Colom and Antoni Buades. Analysis and Extension of the Ponomarenko et al. Method, Estimating a Noise Curve from a Single Image. *Image Processing On Line*, 3:173–197, 2013. https://doi.org/10.5201/ipol.2013.45.

[2] Guoshen Yu and Guillermo Sapiro. DCT Image Denoising: a Simple and Effective Image Denoising Algorithm. *Image Processing On Line*, 1:292–296, 2011. https://doi.org/10.5201/ipol.2011.ys-dct.