Image Processing On Line

An implementation and detailed analysis of the K-SVD image denoising algorithm

Arthur Leclaire and Marc Lebrun

published • 2012-05-19
reference • Arthur Leclaire and Marc Lebrun, *An implementation and detailed analysis of the K-SVD image denoising algorithm*, Image Processing On Line, 2012.
DOI : http://dx.doi.org/10.5201/ipol.2012.llm-ksvd

French version/Version française

- Arthur Leclaire, Université Paris Descartes arthur.leclaire@parisdescartes.fr
- Marc Lebrun, ENS Cachan marc.lebrun@cmla.ens-cachan.fr

Edited by Jean-Michel Morel

Abstract

K-SVD is a signal representation method which, from a set of signals, can derive a dictionary able to approximate each signal with a sparse combination of the atoms. This paper focuses on the K-SVD-based image denoising algorithm. The implementation is described in detail and its parameters are analyzed and varied to come up with a reliable implementation.

Overview

Denoising is a major task of image processing. In the last decades, several denoising algorithms have been proposed.

One class of such algorithms contains those which take profit of the analysis of the image in a (redundant) frame. For example, in this subset, we can mention the threshold of the image coefficients in an orthonormal basis, like the cosine basis, [20] and [19], a wavelet basis [8], or a curvelet basis [18]. In this category can also be included the methods which try to recover the main structures of the signal by using a dictionary (which basically consists of a possibly redundant set of generators). The matching pursuit algorithm [15] and the orthogonal matching pursuit [7] are of this type. The efficiency of these methods comes from the fact that natural images can be sparsely approximated in these dictionaries.

The variational methods form a second class of denoising algorithms. Among them let us mention the total variation (TV) denoising [17], [4] where the chosen regularity model is the set of functions of bounded variations.

In another class, one could include methods that take advantage of the non-local similarity of patches in the image. Among the most famous, we can name NL-means [3], BM3D [6], and NL-Bayes [10].

The K-SVD-based denoising algorithm merges some concepts coming from these three classes, paving the way of dictionary learning. Indeed, the efficiency of the dictionary is encoded

through a functional which is optimized taking profit of the non-local similarities of the image. It is divided into three steps : a) sparse coding step, where, using the initial dictionary, one computes sparse approximations of all patches (with a fixed size) of the image; b) dictionary update, where the dictionary is updated in such a way that the quality of the sparse approximations is increased; and next : c) reconstruction step which recovers the denoised image from the set of denoised patches. Actually, before getting to c), the algorithm carries out K iterations of steps a) and b).

There is by now a thriving literature about dictionary learning. Here we will only quote the main articles that led to the design of the K-SVD algorithm for color images. The K-SVD method was introduced in [1] where the whole objective was to optimize the quality of sparse approximations of vectors in a learned dictionary. Even if this article noticed the interest of the technique in image processing tasks, it is in [9] that a detailed study has been led on gray-level images denoising. The adaptation to color images was treated in [14]. This last article proved that the K-SVD method can also be useful in other image processing tasks, such as non-uniform denoising, demoisaicing and inpainting.

Following these articles, dictionary learning has become a very active research topic. To go beyond the scope of this article, see [13] or [11].

Theoretical Description

To get a maximal coherence between the different documents about K-SVD, we use the same notations as in the article [14].

Algorithm for Grayscale Images

This paragraph explains the algorithm described in [9] . We work with images written in column vectors. In practice, in our C++ code, images are scanned one row at a time, these rows being next concatenated to make a single column vector. The same is done for patches. Hence, let us denote by $\mathbf{x}_0$ a size N column vector containing the unknown clean grayscale image. Starting from $\mathbf{x}_0$, we assume that the noisy image is obtained as

$$\mathbf{y} = \mathbf{x}_0 + \mathbf{w}$$

where $\mathbf{w}$ is a white Gaussian noise vector of zero mean and known standard deviation σ. Consequently, we look for an image $\hat{x}$ that is close to the initial image, such that each of its patches admits a sparse representation in terms of a learned dictionary.

For every possible position *(i,j)* of a pixel in the image $\mathbf{x}$, we denote by $\mathbf{R}_{ij}\mathbf{x}$ the size n column vector formed by the grayscale levels of the squared $\sqrt{n} \times \sqrt{n}$ patch of the image $\mathbf{x}$ and whose top-left corner has coordinates *(i,j)*. One can notice that, with the column notation, $\mathbf{R}_{ij}\mathbf{x}$ is precisely the multiplication of $\mathbf{x}$ (column vector of size N) by a matrix $\mathbf{R}_{ij}x$ of size $n \times N$ whose columns are indexed by the image pixels. Each of the rows of $\mathbf{R}_{ij}\mathbf{x}$ allows to extract the value of one pixel of the image $\mathbf{x}$ and thus is zero except for the coefficient of index p, which is equal to 1.

In the following, the notation **D** refers to a dictionary. It is a matrix of size $n \times k$, with $k \geq n$ whose columns are normalized (in Euclidean norm). We take $k \geq n$ because otherwise, there is no chance that the columns of **D** can span $\Re^n$. The algorithm will require an initialization of the dictionary : to this end, we may choose an usual orthogonal basis (discrete cosine transform, wavelets...), or we may collect patches from clean images or even from the noisy image itself (without forgetting the normalization). We give two examples of dictionaries in Figure [1].

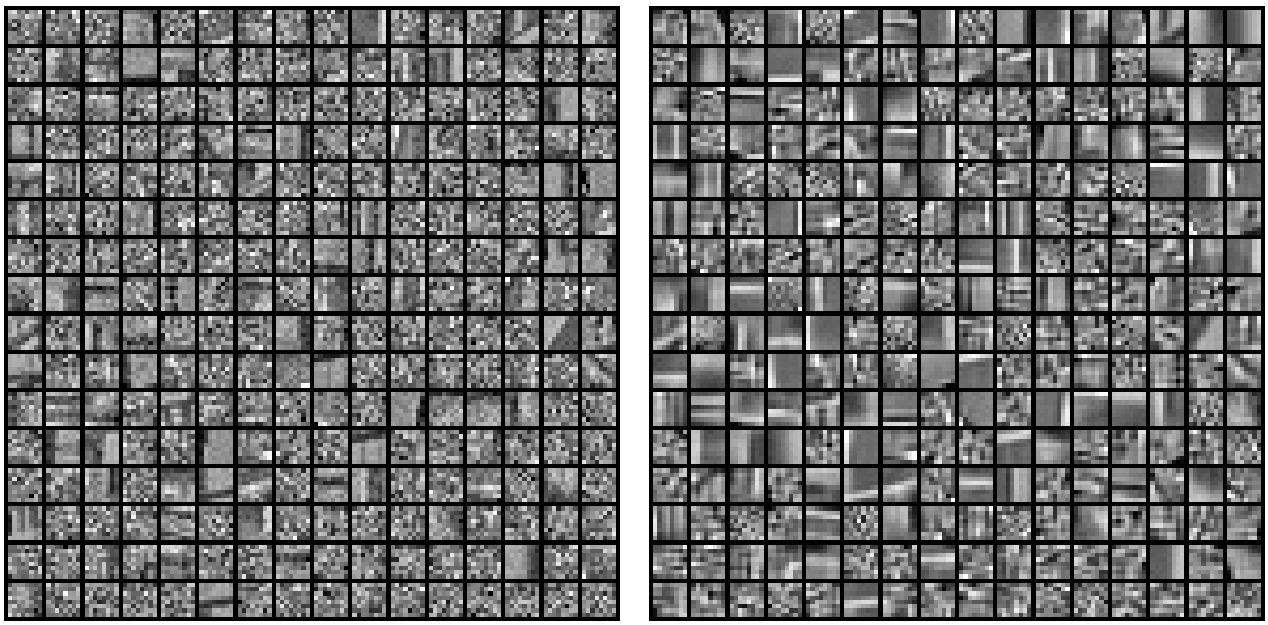


Figure 1 - Left, a dictionary formed with random patches from the image "Castle" (converted in grayscale levels) after addition of a white Gaussian noise. Right, the dictionary obtained at the end of the K-SVD algorithm. For each atom, the contrast is enhanced differently.

The dictionary allows one to compute a sparse representation $\boldsymbol{\alpha}_{ij}$ of each patch $\mathbf{R}_{ij}x$. The representations $\boldsymbol{\alpha}_{ij}$ will thus be column vectors of size k satisfying $\mathbf{R}_{ij}x \approx \mathbf{D}\boldsymbol{\alpha}_{ij}$. We put them together in a matrix $\boldsymbol{\alpha}$ with k rows and N_p columns where N_p is the number of patches of size $\sqrt{n} \times \sqrt{n}$ of the image. With the above notation it is easy to detail each part of the algorithm. At first, $\hat{\mathbf{D}}$ is initialized with an initial dictionary denoted by $\mathbf{D}_{init}$. The initialization alternatives will be discussed later on.

The first step looks for sparse representations of the patches $\mathbf{R}_{ij}\mathbf{y}$ of $\mathbf{y}$ in the dictionary $\hat{\mathbf{D}}$. In other words, for each patch $\mathbf{R}_{ij}\mathbf{y}$, a column vector $\hat{\alpha_{ij}}$ (of size k) is built such that it has only a few non-zero coefficients and such that the distance between $\mathbf{R}_{ij}\mathbf{y}$ and its sparse approximation $\hat{\mathbf{D}}\hat{\alpha_{ij}}$ is small.

The second step updates, one by one, the columns of the dictionary $\hat{\mathbf{D}}$ and the representations $\hat{\alpha_{ij}}$ in such a way that all patches in the image $\mathbf{y}$ become more efficiently represented. Therefore, the goal is to decrease the quantity

$$\sum_{i,j} \|\hat{\mathbf{D}}\hat{\alpha_{ij}} - \mathbf{R}_{ij}\mathbf{y}\|_2^2$$

while keeping the sparsity of the vectors $\hat{\alpha_{ij}}$.

K iterations of these two first steps are performed. Once finished, to each patch $\mathbf{R}_{ij}\mathbf{y}$ of the image $\mathbf{y}$ corresponds the denoised version $\hat{\mathbf{D}}\hat{\alpha_{ij}}$. The third and last step consists in merging the denoised versions of all patches of the image in order to obtain the final denoised image. A new parameter λ is introduced in this part, which blends a portion of the initial noisy image into the final result. To obtain a pixel p of the denoised image, a simple average is done on the values of p in the denoised patches to which it belongs (weighted by 1), and the value of p in the noisy image $\mathbf{y}$ (weighted by λ).

We will now take a closer look at each one of the three parts of the method.

Sparse Coding

This step allows, with a fixed dictionary $\hat{\mathbf{D}}$, to compute sparse representations $\hat{\alpha}$ of the patches $\mathbf{R}_{ij}\mathbf{y}$ of the image in $\hat{\mathbf{D}}$. More precisely, an ORMP (Orthogonal Recursive Matching Pursuit) gives an approximate solution of the (NP-complete) problem

$$\underset{\alpha_{ij}}{\operatorname{argmin}} ||\alpha_{ij}||_0 \quad \text{such that} \quad ||\mathbf{R}_{ij}\mathbf{y} - \hat{\mathbf{D}}\alpha_{ij}||_2^2 \leq n(C\sigma)^2 \qquad (1)$$

where $\|\alpha_{ij}\|_0$ refers to the l^0 norm of $\boldsymbol{\alpha}_{ij}$, i.e. the number of non-zero coefficients of $\boldsymbol{\alpha}_{ij}$. We remind the reader that $\hat{\mathbf{D}}$ is a matrix whose size is $n \times k$, that $\boldsymbol{\alpha}_{ij}$ is a size k column vector and that $\mathbf{R}_{ij}\mathbf{y}$ is a size n column vector. If it were perfect, this ORMP would find a patch with the sparsest representation in $\hat{\mathbf{D}}$ and which distance to $\mathbf{R}_{ij}\mathbf{y}$ is less than $n(C\sigma)^2$. This last constraint brings in a new parameter C. This coefficient multiplying the standard deviation σ guarantees that, with high probability, a white Gaussian noise of standard deviation σ on n pixels has an l^2 norm lower than $\sqrt{n}C\sigma$. We give details on the choice of C in Section 4 . In fact, the ORMP is not perfect : indeed, it only allows one to find a patch having one sparse (not necessarily the sparsest) representation in $\hat{\mathbf{D}}$ and which distance to $\mathbf{R}_{ij}\mathbf{y}$ is lower than $n(C\sigma)^2$.

Let us give more details about how the ORMP can compute a sparse representation of a patch. A good reference to learn about ORMP is [5]. Nevertheless, we shall give here a complete explanation using the notation of our C++ code. In order to use lighter notations, we will rather explain how the ORMP finds a sparse representation $a \in \Re^k$ of a vector $\mathbf{x} \in \Re^n$ in a dictionary formed by the normalized vectors d_1, ..., d_k which span $\Re^n$. This explanation can be found here.

> Let $\mathbf{x}$ be a vector of $\Re^n$. We wand to find a sparse representation $\boldsymbol{\alpha}$ of $\mathbf{x}$ in the dictionary $\mathbf{D}$ formed by the normalized vectors d_1, ..., d_{k-1}. Precisely, we are going to give an approximate solution of the following optimization problem
>
> $$\underset{\alpha \in \mathbb{R}^k}{\operatorname{argmin}} ||\alpha||_0 \quad \text{such that} \quad ||x - \mathbf{D}\alpha||_2^2 \leq \varepsilon \qquad (2)$$
>
> We will detail the choice of the atoms in order to stick to our C++ code. We denote by l_j the index of the element of the dictionary that we choose at the step $j \geq 0$. We also let $L_j = \{l_0, ..., l_j\}$. Let us assume that we are at the beginning of the j^{th} loop ($j \geq 0$) (and thus l_0, ..., l_{j-1} are already chosen). We start by introducing the residue

$$r = x - \mathrm{Proj}_{\mathrm{Vect}(d_{l_0},\ldots,d_{l_{j-1}})}(x)$$

where Proj_F refers to the orthogonal projection on the subspace F, and where $\mathrm{Vect}_{d_{l_0}, \ldots, d_{l_{j-1}}}$ refers to the space spanned by the vectors $d_1, \ldots, d_{j-1}$. If $||r||_2 < \varepsilon$ then we stop and $\boldsymbol{\alpha}$ is the representation of $\mathrm{Proj}_{\mathrm{Vect}(d_{l_0},\ldots,d_{l_{j-1}})}(x)$ in $(d_{l_0}, \ldots, d_{l_{j-1}})$ already obtained at the previous step, cf. its computation at the end of the loop (if we break when $j = 0$, then $\boldsymbol{\alpha} = 0$). We choose l_j in order to minimize the norm of the new potential residue

$$l_j = \underset{i \notin L_{j-1}}{\mathrm{argmin}} \|x - \mathrm{Proj}_{\mathrm{Vect}(d_{l_0},\ldots,d_{l_{j-1}},d_i)}(x)\|^2$$

Thanks to the Pythagorean theorem, this amounts to

$$l_j = \underset{i \notin L_{j-1}}{\mathrm{argmax}} \|\mathrm{Proj}_{\mathrm{Vect}(d_{l_0},\ldots,d_{l_{j-1}},d_i)}(x)\|^2$$

Then we set $L_j = L_{j-1} \cup \{l_j\}$.
In order to compute the orthogonal projections

$$\mathrm{Proj}_{\mathrm{Vect}(d_{l_0},\ldots,d_{l_{j-1}},d_i)}(x) \ , \quad (i \notin L_{j-1})$$

we use the Gram-Schmidt process. We denote by $(t_{l_0}, \ldots, t_{l_{j-1}})$ the orthogonal family obtained after Gram-Schmidt orthogonalization of $(d_{l_0}, \ldots, d_{l_{j-1}})$, and by $(e_{l_0}, \ldots, e_{l_{j-1}})$ the orthonormal family obtained after Gram-Schmidt orthonormalization of $(t_{l_0}, \ldots, t_{l_{j-1}})$. For $i \notin L_{j-1}$, we denote by $(t_{l_0}, \ldots, t_{l_{j-1}}, t_i^{(j)})$ the family obtained after Gram-Schmidt orthogonalization of $(d_{l_0}, \ldots, d_{l_{j-1}}, d_i)$, and $(e_{l_0}, \ldots, e_{l_{j-1}}, e_i^{(j)})$ the (orthonormal) family obtained by normalizing of $(t_{l_0}, \ldots, t_{l_{j-1}}, t_i^{(j)})$. The reader have to be aware that this orthonormalization can be progressively computed : at the j^{th} step, the vectors $(t_{l_0}, \ldots, t_{l_{j-1}})$ and $(e_{l_0}, \ldots, e_{l_{j-1}})$ are already computed. It is thus sufficient to detail, at the j^{th}, the computation of d_i and $t_i^{(j)}$ for $i \notin L_{j-1}$

$$t_i^{(j)} = d_i - \sum_{p=0}^{j-1} < d_i, e_{l_p} > e_{l_p} \ ,$$

$$\|t_i^{(j)}\|^2 = 1 - \sum_{p=0}^{j-1} < d_i, e_{l_p} >^2 \ ,$$

$$e_i^{(j)} = \frac{t_i^{(j)}}{\|t_i^{(j)}\|} \ .$$

We notice that

$$\text{Proj}_{\text{Vect}(d_{l_0},\ldots,d_{l_{j-1}},d_i^{(j)})}(x) = < x, e_{l_0} > e_{l_0} + \ldots + < x, e_{l_{j-1}} > e_{l_{j-1}} + < x, e_i^{(j)} > e_i^{(j)}$$

(where Proj_F refers to the orthogonal projection onto the subspace F) and, consequently,

$$\|\text{Proj}_{\text{Vect}(d_{l_0},\ldots,d_{l_{j-1}},d_i)}(x)\|^2 = < x, e_{l_0} >^2 + \ldots + < x, e_{l_{j-1}} >^2 + < x, e_i^{(j)} >^2 \ .$$

Therefore, maximizing the norm of the projection is equivalent to maximizing <**x**, $e_i^{(j)}$>. This is why we choose

$$l_j = \underset{i \notin L_{j-1}}{\text{argmax}} \ < x, e_i^{(j)} >^2$$

and with this index comes the vector $t_{l_j} = t_{l_j}^{(j)}$ and the normalized vector $e_{l_j} = e_{l_j}^{(j)}$. The computation of <**x**, $e_i^{(j)}$> is done by replacing $e_{l_j}^{(j)}$ by its above given definition

$$< x, e_i^{(j)} > = \frac{< x, d_i > - \sum_{p=0}^{j-1} < d_i, e_{l_p} >< x, e_{l_p} >}{\sqrt{1 - \sum_{p=0}^{j-1} < d_i, e_{l_p} >^2}} \ . \qquad (3)$$

To implement this computation efficiently, we notice that the denominator and the square of the numerator are nothing but the subtraction of those used at the previous step by respectively <d_i, $e_{l_{j-1}}$><**x**, $e_{l_{j-1}}$> and <d_i, $e_{l_{j-1}}$>. Hence, at each step, we need <d_i, $e_{l_{j-1}}$> and <**x**, $e_{l_{j-1}}$> which correspond in the code to the variables **D_ELj[i][j]** and **x_elj**, which are updated at each loop. The computation of <**x**, $e_{l_{j-1}}$> is not a problem (it is only the formula (3) of the previous step !). However, we have to explain the update of <d_i, $e_{l_{j-1}}$>. We will see thereafter that the computation of **α** requires the coordinates of (e_{l_0}, ..., $e_{l_{j-1}}$) on the basis (d_{l_0}, ..., $d_{l_{j-1}}$) and we will explain how we can obtain them progressively. Once these coordinates are computed, the scalar product <d_i, $e_{l_{j-1}}$> can be obtained by a linear combination of the scalar products <d_i, d_{l_s}>, ($0 \leq s < j$). The numerator (<**x**, $t_i^{(j)}$>) is then saved in the variable **x_T[i]**, and the square of the denominator in the variable **scores[i]**. Once we have chosen l_j, we can go back to the beginning of the loop to stop or choose the next atom. Clearly, the algorithm terminates because the atoms d_1, ..., d_k span $\Re^n$.

At this point let us assume that we are at the end of the j^{th} loop (and thus, we have chosen l_1, ..., l_j). We still have to explain how the sparse reprentation **α** of **x** in **D** is computed. As (e_{l_0}, ..., e_{l_j}) is orthonormal and span Vect(d_{l_0}, ..., d_{l_j}), we have

$$x \approx \mathrm{Proj}_{\mathrm{Vect}(d_{l_0},\ldots,d_{l_j})} = \sum_{p=0}^{j} < x, e_{l_p} > e_{l_p} .$$

The coefficients <**x**, e_p>, ($p < j$) have already been computed in the preceding step. The last coefficient is given by the equality (3) for $i = l_j$. Finally, we have to go back to the representation in terms of d_{l_1}, ..., d_{l_j}. To this aim, we introduce the coordinates of the (e_{l_0}, ..., $e_{l_{j-1}}$) on the basis (d_{l_0}, ..., $d_{l_{j-1}}$). Let us denote them by a_{pq}, ($p \leq q$)

$$\forall p < j, \quad e_{l_p} = \sum_{q=0}^{p} a_{pq} d_{l_q} .$$

At the $j\text{-}1^{th}$ step, the a_{pq} are computed for $p < j$ (and again $p \leq q$). It suffices to explain how we compute a_{jq} for $q \geq j$. For the definition e_{l_j}, replacing the e_{l_p}, ($p < j$), we obtain

$$e_{l_j} = \frac{1}{\|t_{l_j}\|} \left(d_{l_j} - \sum_{p=0}^{j-1} \sum_{q=0}^{p} < d_{l_j}, e_{l_p} > a_{pq} d_{l_q} \right) ,$$

from which we get (after inverting the sums)

$$a_{jj} = \frac{1}{\|t_{l_j}\|} , \qquad (4)$$

$$\forall q < j, \quad a_{jq} = -\frac{1}{\|t_{l_j}\|} \left(\sum_{p=q}^{j-1} < d_{l_j}, e_{l_p} > a_{pq} \right) . \qquad (5)$$

Finally, we have

$$x \approx \sum_{p=0}^{j} < x, e_{l_p} > e_{l_p} = \sum_{q=0}^{j} \left(\sum_{p=q}^{j} < x, e_{l_p} > a_{pq} \right) d_{l_q} ,$$

and thus we set

$$\alpha_s = 0 , \quad \text{if } s \notin L_j, \quad \text{and}$$

$$\alpha_s = \sum_{p=q}^{j} < x, e_{l_p} > a_{pq} , \quad \text{if } s = l_q .$$

We insist on the fact that the coordinates of (e_{l_0}, ..., $e_{l_{j-1}}$) on the basis (d_{l_0}, ..., $d_{l_{j-1}}$) are also required for the choice of the index l_j, as explained above. Subsequently, it is natural to compute these coordinates at each loop.

Correspondence with the notations used in the code

Now we link the notations used in the explanation above with the notations used in the code. First, in the code, let us warn the reader that we have used indexation in column order, that is, **D[i]** refers to the **i**-th column of the matrix **D**. We have also used a convention : whenever a variable contains the matrix multiplication of the transpose of **B** by **A**, then the result is saved in the variable **A_B**. Therefore, **A_B** = ^{T}**BA**, and **A_B[p][q]** is the scalar product between **A[p]** and **B[q]**. Let us add that **elj** (even if it is not a proper variable) will of course refer to e_{l_j}. Similarly, **DLj** (resp. **ELj**) will refer to the matrix whose columns are (in order) d_{l_0}, ..., d_{l_j} (resp. d_{e_0}, ..., e_{l_j}). Last, **T** will refer to the matrix whose columns are $t_0^{(j)}$, ..., $t_{k-1}^{(j)}$.

- **Np** = N_p
- **n** = n
- **k** = k
- **epsilon** = ε
- **L** : maximal sparsity allowed for the representations (here we do not use this constraint, i.e. in our code, **L** = min(n, k))
- **norm[i]** = $\|t_i^{(j)}\|^2 = 1 - \sum_{p=0}^{j-1} < d_i, e_{l_p} >^2$
- **x_T[i]** = $< x, t_i^{(j)} > = < x, d_i > - \sum_{p=0}^{j-1} < d_i, e_{l_p} >< x, e_{l_p} >$
- **scores[i]** = $< x, e_i^{(j)} >^2 = \frac{\mathrm{Rdn}[i]^2}{\mathrm{norm}[i]}$
- **lj** = l_j
- **invNorm** = **1/sqrt(norm[lj])** = $\frac{1}{\|t_{l_j}\|}$
- **x_elj** = **x_T[lj] * invNorm** = <**x**, e_{l_j}>
- **x_el[p]** = <**x**, e_{l_p}>
- **delta** = **x_elj * x_elj** = <**x**, e_{l_j}>^{2}
- **normr** = $\|x\|^2 - \sum_{p=0}^{j} < x, e_{l_p} >^2$
- **D_DLj[i][s]** = <d_i, d_{l_s}>
- **A[p][q]** = a_{pq}, ($p \geq q$)
- **D_ELj[i][j]** is equal to <d_i, e_{l_j}> at the end of the j-th loop
- **val** temporarily saves the variable <d_i, e_{l_j}>
- **coord[q]** = $\boldsymbol{\alpha}_{l_q} = \sum_{p=q}^{j} < x, e_{l_p} > a_{pq}$: "``coordinate" of **x** on d_{l_q}
- **s** : summing index.

Some remarks on the implementation

> **Update of A** equations (4) and (5) suggest the update :

$$\texttt{A}[j][j] = \texttt{invNorm}$$

$$\forall i < j, \quad \texttt{A}[j][i] = \left(-\sum_{k=i}^{j-1} \texttt{D\_ELj}[\texttt{lj}][\texttt{k}] * \texttt{A}[\texttt{k}][\texttt{i}] \right) \cdot \texttt{invNorm}\,.$$

Numerical stability an artificial break is added in the code. It happens if $||t_j|| < 10^{-6}$. Thus the ORMP is stopped in order to avoid the division by $||t_j||$

Dictionary Update

In this step, we update the columns of the dictionary one by one, to make the quantity

$$\sum_{i,j} \|\hat{\mathbf{D}}\hat{\alpha_{ij}} - \mathbf{R}_{ij}\mathbf{y}\|_2^2 \qquad (6)$$

decrease, without increasing the sparsity penalty $||\boldsymbol{\alpha}_{ij}||_0$. We will denote by $\hat{d}_l$ $(1 \leq l \leq k)$ the columns of the dictionary $\hat{\mathbf{D}}$.

First, let us try to minimize the quantity (6) without taking care of the sparsity. As explained above, we go through the columns of the dictionary, and the index of the current column will be denoted by l, $(1 \leq l \leq k)$. We are going to modify the atom $\hat{d}_l$ and the coefficients $\hat{\alpha_{ij}}(l)$ in order to improve the approximations in an L^2 distortion sense. In order to translate this objective into an optimization problem, for each (i, j), we introduce the residue

$$e_{ij}^l = \mathbf{R}_{ij}\mathbf{y} - \hat{\mathbf{D}}\hat{\alpha_{ij}} + \hat{d}_l\hat{\alpha_{ij}}(l) \qquad (7)$$

which is the error committed by deciding not to use $\hat{d}_l$ any more in the representation of the patch $\mathbf{R}_{ij}\mathbf{y}$: e_{ij}^l is thus a size n vector.

These residues are grouped together in a matrix $\mathbf{E}_l$ (whose columns are indexed by (i, j). The values of the coefficients $\hat{\alpha_{ij}}(l)$ are also grouped in a row vector denoted by $\hat{\alpha}^l$. Therefore, $\mathbf{E}_l$ is a matrix of size $n \times N_p$ and $\hat{\alpha}^l$ is a row vector of size N_p. We need to find a new $\hat{d}_l$ and a new row vector $\hat{\alpha}^l$ which minimize

$$\sum_{i,j} \|\hat{\mathbf{D}}\hat{\alpha_{ij}} - \hat{d}_l\hat{\alpha_{ij}}(l) + d_l\alpha^l - \mathbf{R}_{ij}\mathbf{y}\|_2^2 = ||\mathbf{E}_l - d_l\alpha^l||_F^2 \qquad (8)$$

where the squared Frobenius norm $||M||_F^2$ refers to the sum of the squared elements of M. This Frobenius norm is also equal to the sum of the squared (Euclidean) norm of the columns, and it is easy to check that minimizing (8) amounts to reduce the approximation error caused by $\hat{d}_l$. It is well-known that the minimization of such a Frobenius norm consists in a rank-one approximation, which always admits a solution, practically given by the singular value

decomposition (SVD). Using the SVD of $\mathbf{E}_l$:

$$\mathbf{E}_l = U\Delta V^T \qquad (9)$$

(where U and V are orthogonal matrices and where Δ is the null matrix except from its first diagonal, where it is non-negative and decreasing), the updated values of $\hat{d}_l$ and $\hat{\alpha}^l$ are respectively the first column of U and the first column of V multiplied by $\Delta(1,1)$. By the way, we will notice that the rank-one approximation does not require the computation of the whole matrices U, V, and Δ. In our implementation, it is sufficient to use a truncated SVD, which is much faster (especially if $\mathbf{E}_l$ is large). The method used to compute the truncated SVD can be found here.

> To use lighter notations, we use, as in the code, the notation $X = \mathbf{E}_l$. Starting from the SVD (9), one can write
>
> $$XX^T = U\Delta\Delta^T U^T ;$$
>
> $$X^T X = V\Delta^T\Delta V^T .$$
>
> As a result, $\Delta(1, 1)$ is the square of the greatest eigenvalue of the symetrical positive-definite matrix $X\ X^T$, and the first column of U is the corresponding eigenvector. The same observation is valid for V. Therefore, we can find these eigenvectors and $\Delta(1, 1)$ thanks to the power method applied to the matrices $X\ X^T$ and $X^T\ X$. Concerning the convergence of the power method, one could refer to [2]. One could notice that in the pseudo-code that we present below, the power method can be applied to the two matrices simultaneously.
>
> The SVD function takes as arguments a matrix X of which we want the SVD, a maximal number of iterations **max_iter** (set to 100 in the code) and a tolerance threshold ε (set to 10^{-6} in the code). It gives back an approximation s of the greatest singular value of X, an approximation u of the first column of U, and an approximation v of the first column of V.
>
> Here is the pseudo-code.
> Initialization : we arbitrarily initialize v (in the code, we set $v = \hat{d}_l$); we also set $i = 0$, $s = 1$ and $s_{old} = 0$.
> While ($i <$ **max_iter** and $\left|\frac{s - s_{old}}{s}\right| > \varepsilon$), we proceed to the following affectations :
>
> - $u \leftarrow Xv$;
> - $u \leftarrow u / ||u||$;
> - $v \leftarrow X^T u$;
> - $s_{old} \leftarrow s$;

- $s \leftarrow ||v||$;
- $v \leftarrow v / s$.

The values of s, u, and v obtained at the end of this loop are the return values of the truncated SVD.

Remark : At the end of this algorithm, we thus have

$$XX^T u \approx \lambda u$$

where λ is the greatest eigenvalue of $X\,X^T$. Taking the scalar product with u, and since u is normalized, we have

$$\|X^T u\|^2 = \langle XX^T u, u \rangle \approx \lambda\,,$$

which yields

$$s \approx \sqrt{\lambda}\,.$$

This explains why s is an approximation of the largest singular value of X.

This way, for each $l = 1, \ldots, k$, the energy (6) never increases. But for now, the sparsity of the coefficients is not under control. In order to do that, a slight modification is brought into the preceding process : for each l, the operations involved in the update of $\hat{d}_l$ and $\hat{\alpha}^l$ are restricted to the patches which already used the atom $\hat{d}_l$ before the update. Setting

$$\omega_l = \{\,(i,j)\,|\,\hat{\alpha_{ij}}(l) \neq 0\,\}\,,$$

the values that we will group together in $\mathbf{E}_l$ and $\hat{\alpha}^l$ will be only the values of e_{ij}^l and $\hat{\alpha_{ij}}(l)$ for indices $(i, j) \in \omega_l$. Hence, the indices (i, j) of the sum of the LHS of (8) will be restricted to $(i, j) \in \omega_l$; the matrix $\mathbf{E}_l$ is now of size $n \times \mathrm{Card}(\omega_l)$ and $\hat{\alpha}^l$ is now a row vector of size $\mathrm{Card}(\omega_l)$. Also, in (6), note that the terms of indices $(i, j) \notin \omega_l$ are not affected by this update. This proves that this modification decreases (6) without increasing $||\boldsymbol{\alpha}_{ij}||_0$. This modification also implies a reduction of the matrix $\mathbf{E}_l$ which SVD is being computed.

Recall that the sparse coding computes sparse representations $\hat{\alpha}$ and that the dictionary updates make $\hat{\mathbf{D}}$ change but also modify $\hat{\alpha}$. After K iterations of these steps, we are in possession of a learned dictionary $\hat{\mathbf{D}}$ and of sparse representations $\hat{\alpha_{ij}}$ of the patches of the image.

Reconstruction

Now that the first two parts of the algorithm built a dictionary $\hat{\mathbf{D}}$ and sparse representations $\hat{\alpha_{ij}}$ which are well-adapted to our image, we can build the globally denoised image by solving the minimization problem

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\mathbb{R}^N}{\operatorname{argmin}} \lambda||\mathbf{x}-\mathbf{y}||_2^2 + \sum_{i,j} ||\hat{\mathbf{D}}\hat{\alpha_{ij}} - \mathbf{R}_{ij}\mathbf{x}||_2^2 . \qquad (10)$$

The first term controls the global proximity to our reconstruction $\hat{\mathbf{x}}$ with the noisy image **y**. It is thus a fidelity term that is weighted by the parameter λ. The second term controls the proximity of the patch $\mathbf{R}_{ij}\hat{\mathbf{x}}$ to our reconstruction to the denoised patch $\mathbf{D}\boldsymbol{\alpha}_{ij}$. This functional is quadratic, coercive, and differentiable. Subsequently, this problem admits a unique solution that we can compute explicitly:

$$\hat{\mathbf{x}} = \left(\lambda\mathbf{I} + \sum_{i,j}\mathbf{R}_{ij}^T\mathbf{R}_{ij}\right)^{-1}\left(\lambda\mathbf{y} + \sum_{i,j}\mathbf{R}_{ij}^T\hat{\mathbf{D}}\hat{\alpha_{ij}}\right) . \qquad (11)$$

This formula may seem complicated, but it is in fact very simple. The only thing to notice is that the matrix to be inverted is diagonal. In consequence, this formula only means that the value of a pixel in the denoised image is computed by averaging the value of this pixel in the noisy image (weighted by λ) and the values of this pixel on the patches to which it belongs (weighted by 1). We obtain the values of the pixels of $\hat{\mathbf{x}}$, one by one, without requiring any matrix inversion that (11) would perhaps suggest.

Comments

In the articles (9) and (14) the following minimization problem is mentioned :

$$(\hat{\mathbf{x}}, \hat{\mathbf{D}}, \hat{\boldsymbol{\alpha}}) = \underset{\mathbf{D},\mathbf{x},\boldsymbol{\alpha}}{\operatorname{Argmin}} \lambda||\mathbf{x}-\mathbf{y}||_2^2 + \sum_{i,j}\mu_{ij}||\alpha_{ij}||_0 + \sum_{i,j}||\mathbf{D}\alpha_{ij} - \mathbf{R}_{ij}\mathbf{x}||_2^2 \qquad (12)$$

which groups all the quantities that we have tried to minimize in the preceding paragraphs.

Let us briefly analyze this formula, even though the forthcoming comments are slightly redundant with the previous explanation :

- the first term controls the global proximity of $\hat{\mathbf{x}}$ to the noisy image **y** (fidelity term);
- the second term controls the sparsity of the representations of the patches;
- last, the third term controls for each (*i*, *j*), the proximity of the patch $\mathbf{R}_{ij}\hat{\mathbf{x}}$ of our reconstruction to the denoised patch $\mathbf{D}\boldsymbol{\alpha}_{ij}$.

The coefficients λ and μ_{ij} set the balance between the importance given to the fidelity term and to the sparsity constraints of the representations of the patches.

This non-convex problem is too difficult to be addressed in this form. This explains why the article [9] suggests to break it down into parts, and to try to minimize separately the different terms of (12). This way, we are led to the K-SVD algorithm. Notice also a serious difference: the values of μ_{ij} are not required in the above implementation.

Without specifying values for μ_{ij}, we cannot really address the problems of linking the minimization of (12) and the suggested iterative method. Moreover, we do not understand why the authors did not set only one weight μ rather than weights μ_{ij} depending on the patches. We would have to explain why the sparsity of certain patches are more important than others. If the

μ_{ij} are not equal, then their determination is still a crucial point of the method that remains to be analyzed.

The alternation of sparse coding step and dictionary update step renders the analysis of the afore-mentioned energies difficult. On the one hand, the ORMP is only an approximate solution. On the other hand, in the sparse coding step, the constraints are formed by parts of the Frobenius norm that is minimized in the dictionary update. For this reason, we want to insist on the fact that the minimization of (12) is nothing but a possible interpretation of the K-SVD method. Of course, solving directly the problem (12) is appealing but seems for now out of reach.

The reader could notice that, each time we update an atom of the dictionary, the algorithm uses a SVD. Since there are k atoms in the dictionary, this explains the name K-SVD. As stated in [1], the reference to K-means is not just formal : in K-means, we do not allow sparse combinations of the atoms, but we try to optimize the dictionary in such a way that the error committed by representing each observation with a single atom in the dictionary is minimal.

Extending K-SVD to Color Images

It is now time to present the method proposed in [14] to adapt the grayscale algorithm to color images. To address this problem, a first suggestion would be to apply the K-SVD algorithm to each channel R, G and B separately. This naïve solution gives color artifacts that are shown on the left image in Figure 2.

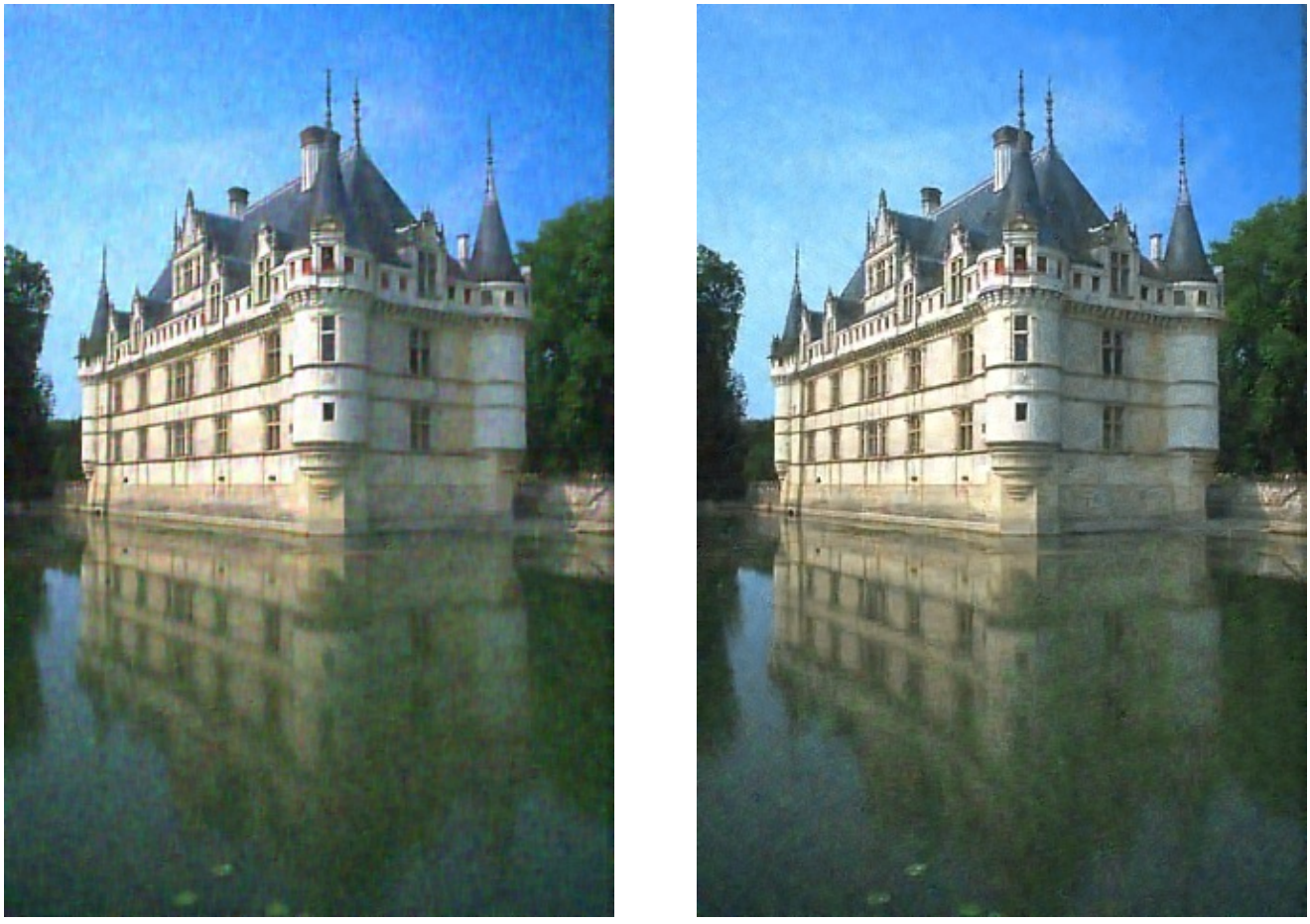


Figure 2 - Denoised images with separated channels (left), and then concatenated channels (right). ($\sigma = 25$). The reader will notice that the denoising is better on the sky and the water surfaces.

They are due to the fact that in natural images there is an important correlation between channels. Another suggestion would be to apply a principal component analysis on channels RGB, which would uncorrelate them, and then to apply the first suggestion in this more appropriate environment. This solution has not been tried because the new proposition of [14] seems even more promising.

In order to obtain the colors correctly, the algorithm previously described will be applied on column vectors which are the concatenation of the R,G,B values. In this way, the algorithm will better update the dictionary, because it is able to learn correlations which exist between color channels. An example of color dictionary is shown in Figure 3.

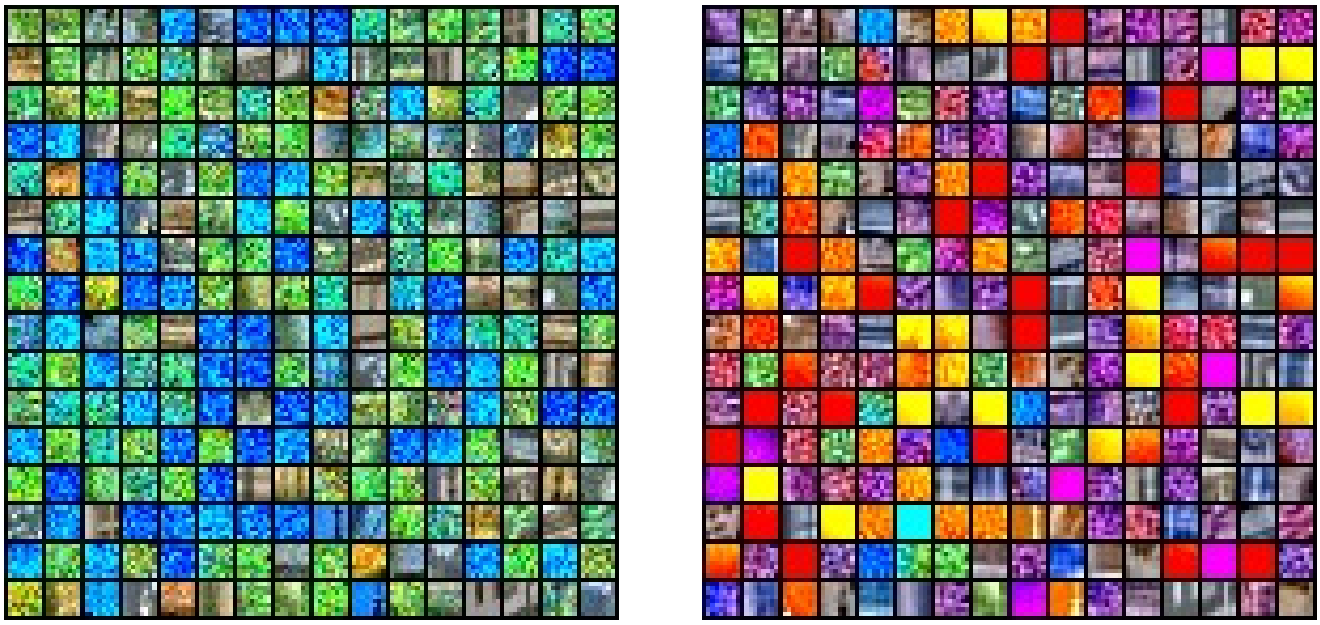


Figure 3 - Left : dictionary composed by patches extracted randomly from the "Castle" image, on which a white gaussien noise has been added. Right : dictionary obtained at the end of the color version of K-SVD. The contrast is enhanced independently for each atom.

One can see the difference in Figure 2. We remind the reader that from now on the size of columns which represent images is $3N$, and the size of columns which represent patches is $3n$.

Unfortunately, even with this adaptation, non-negligible color artifacts are still present.

The authors of [14] justify these artifacts with the following statement : the previously described algorithm tries to adapt the dictionary to all patches contained in the image. This need of universality implies that the atoms of the dictionary tend to look like grayscale atoms. To correct these color artifacts, [14] suggests to modify the metric used in the stopping condition of the ORMP. From now on we use the metric inferred by the scalar product

$$\langle y, x \rangle_\gamma = y^t x + \frac{\gamma}{n^2} y^t J^t J x \qquad (13)$$

instead of the Euclidean metric, where we denote by J the matrix whose size is $3n \times 3n$ built from three diagonal blocks of size $n \times n$, full of 1, and where $\gamma \geq 0$ is parameter which needs to be fixed. In other words, the new norm can be written as

$$||x||_\gamma^2 = ||x||^2 + \gamma(m_R(x)^2 + m_G(x)^2 + m_B(x)^2) \qquad (14)$$

where we denote by $m_C(\mathbf{x})$ the average of $\mathbf{x}$ on the channel C (and where the Euclidean norm is denoted by ||.||).

Thus, the new metric, under the parameter γ, put more importance of the proximity of the mean value of the patches. This color correction can be easily integrated in the ORMP thanks to the following equality :

$$I + \frac{\gamma}{n}J = \left(I + \frac{a}{n}J\right)^t \left(I + \frac{a}{n}J\right) \qquad (15)$$

where $a > 0$ is chosen so that $\gamma = 2a + a^2$. Thus we can write for all vectors $\mathbf{x}$,

$$||x||_\gamma = \left\|(I + \frac{a}{n}J)x\right\| \ . \qquad (16)$$

Consequently, to work with the new metric, all columns have to be multiplied by $(I + (a/n)\ J)$ and we can work again with the Euclidean norm. Nevertheless we remind the reader that in the ORMP all columns of the dictionary are normalized, which is why a diagonal matrix D is introduced. Its elements are the inverses of the norm of the colums of $(I + (a/n)\ J)\mathbf{D}$. Its size is $k \times k$. Then $(I + (a/n)\ J)\mathbf{D}D$ has normalized columns. Now the ORMP can be applied to obtain the $\hat{\beta_{ij}}$ such that

$$\left(I + \frac{a}{n}J\right)\mathbf{R}_{ij}\mathbf{y} \approx \left(I + \frac{a}{n}J\right)\mathbf{D}\mathcal{D}\hat{\beta_{ij}}$$

for the Euclidean norm. In the next session, if we denote

$$\hat{\alpha_{ij}} = \mathcal{D}\hat{\beta_{ij}} \ ,$$

we get

$$\mathbf{R}_{ij}\mathbf{y} \approx \mathbf{D}\hat{\alpha_{ij}}$$

for the norm $||.||_\gamma$.

One can notice the contribution of this color version in Figure 4.

Original

Noisy image

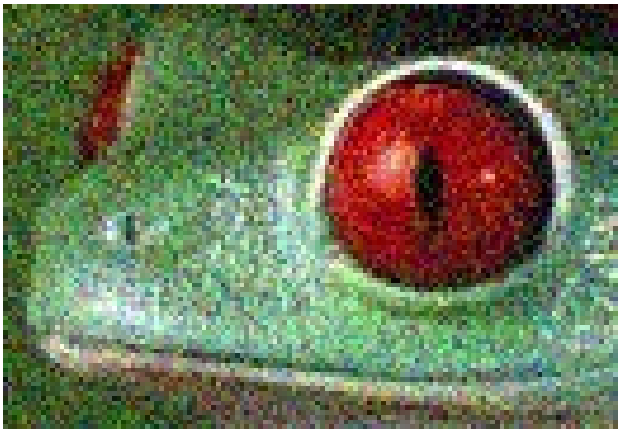
$\gamma = 0$

$\gamma = 5.25$

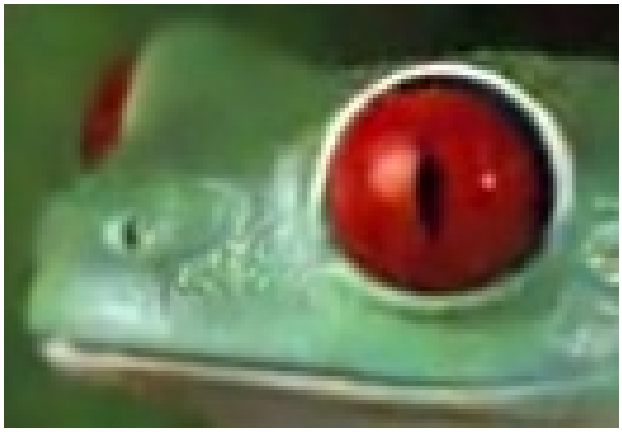
Figure 4 - Denoising for $\sigma = 30$ with $\gamma = 0$ and $\gamma = 5.25$. Some color artifacts still remain, but the denoising is slightly better in some areas when $\gamma = 5.25$, cf Figure 5.

Here again has appeared a new parameter γ which will be briefly discussed in the following part. figure 5.

Original



Noisy image



$\gamma = 0$

$\gamma = 5.25$

Figure 5 - Denoising for $\sigma = 30$ with $\gamma = 0$ and $\gamma = 5.25$. Zooms.

Summary of the Algorithm

In this section all steps of the algorithm are summarized in their right order.

Input : the noisy image **y**, an initial dictionary $\mathbf{D}_{init}$ and the parameters listed in the next part.

All patches of the noisy image are collected in column vectors $\mathbf{R}_{ij}\mathbf{y}$ (channels R, G and B are concatenated). We set initially $\hat{\mathrm{D}} = \mathbf{D}_{init}$. Do K times the two following steps :

1. Sparse coding
   The inverses of the norms of the columns of $(I + (a/n)\,J)\hat{\mathrm{D}}$ are put in a diagonal matrix $\mathcal{D}$.
   An ORMP is applied to the vectors $(I + (a/n)\,J)\,\mathbf{R}_{ij}\mathbf{y}$ with the dictionary $(I + (a/n)\,J)\mathbf{D}\mathcal{D}$ in a such way that sparse coefficients $\hat{\beta}_{ij}$ for the Euclidean norm are obtained, such that :

$$\left(I + \frac{a}{n}J\right)\mathbf{R}_{ij}\mathbf{y} \approx \left(I + \frac{a}{n}J\right)\mathbf{D}\mathcal{D}\hat{\beta}_{ij};$$

   Deduce $\hat{\alpha_{ij}} = \mathcal{D}\hat{\beta}_{ij}$(sparse too) which then verify

$$\mathbf{R}_{ij}\mathbf{y} \approx \mathbf{D}\hat{\alpha_{ij}}$$

   for the norm $\|.\|_\gamma$.

2. Dictionary update
   For each $l = 1, \ldots, k$, proceed to the four following steps :

   1. Introduce $\omega_l = \{ (i, j) \mid \hat{\alpha_{ij}}(l) \neq 0 \}$.
   2. For each $(i, j) \in \omega_l$, obtain the residue

   $$e_{ij}^l = \mathbf{R}_{ij}\mathbf{y} - \hat{\mathbf{D}}\hat{\alpha_{ij}} + \hat{d}_l\hat{\alpha_{ij}}(l) \; ;$$

   3. Put these column vectors together in a matrix $\mathbf{E}_l$. Values $\hat{\alpha_{ij}}(l)$ are also assembled in a row vector denoted by $\hat{\alpha}^l$ for $(i, j) \in \omega_l$;
   4. Update $\hat{d}_l$ and $\hat{\alpha}^l$ as solutions of the minimization problem :

   $$(d_l, \hat{\alpha}^l) = \underset{d_l, \alpha^l}{\text{Argmin}} \|\mathbf{E}_l - d_l\alpha^l\|_F^2 \; .$$

   In practice a truncated SVD is applied to the matrix $\mathbf{E}_l$. It provides partially U, V (orthogonal matrices) and Δ (filled in with zeroes except on its first diagonal), such that $\mathbf{E}_l = U\Delta V^T$ Then $\hat{d}_l$ is defined again as the first column of U and $\hat{\alpha}^l$ as the first column of V multiplied by $\Delta(1,1)$.

Then the final result $\hat{\mathbf{x}}$ is obtained thanks to a weighting aggregation (the formula has already been explained) :

$$\hat{\mathbf{x}} = \left(\lambda\mathbf{I} + \sum_{i,j} \mathbf{R}_{ij}^t\mathbf{R}_{ij} \right)^{-1} \left(\lambda\mathbf{y} + \sum_{i,j} \mathbf{R}_{ij}^t\hat{\mathbf{D}}\hat{\alpha_{ij}} \right) \; .$$

Influence of the Parameters on the Performance

One can notice that the algorithm as described previously has plenty of parameters that can be tuned. Here is the exhaustive list :

- C : multiplier coefficient;
- λ : weight of the noisy image;
- K : number of iterations;
- k : size of the dictionary;
- γ : color correction parameter;
- $\sqrt{n}$: size of patches.

The question is to pick the right values for the various parameters listed above, and to evaluate their influence on the final result.

Influence of *C*

This parameter is used in the stopping condition of the ORMP. In order to understand the chosen value, let us get started with a clean patch x_0 (where the length of the column is denoted by $\tilde{n} = n$ (resp. $\tilde{n} = 3n$) for grayscale (resp. color) images), on which a white Gaussian noise w is added to obtain a noisy patch **x**. Then the ORMP tries to find a vector **α** as sparse as possible such that

$$||x - D\alpha||_2 \leq \sqrt{\tilde{n}}C\sigma \, .$$

If the noise has norm lower than $\sqrt{\tilde{n}}C\sigma$, then **x** will be in the x_0-centered sphere, which radius is $\sqrt{\tilde{n}}C\sigma$. If we assume that x_0 is the only element of this sphere to have a sparse representation in the dictionary Dx_0. Then we will ensure that the noise has a large probability to belong to this sphere. Thus the idea of [14] is to force

$$\mathbb{P}(||w||_2 \leq \sqrt{\tilde{n}}C\sigma) = 0.93 \, . \qquad (17)$$

Practically, the corresponding value is obtained by using the inverse of the distribution function of $\chi^2(\tilde{n})$.

Influence of the Weighting Parameter λ

If for all $\lambda \geq 0$ we denote by $\hat{\mathbf{x}}_\lambda$ the final result of the algorithm as described previously using the parameter λ, then according to the definition of $\hat{\mathbf{x}}_\lambda$ (cf. formula (11)), one can notice that

$$\hat{\mathbf{x}}_\lambda = \frac{1}{\lambda + 1}(\lambda \mathbf{y} + \hat{\mathbf{x}}_0) \, .$$

If we want to remove from the noisy image the same quantity of energy than the one that was added by the noise, then it is natural to choose the parameter λ so that

$$\|\hat{\mathbf{x}}_\lambda - \mathbf{y}\| = \sqrt{\tilde{N}}\sigma \qquad (18)$$

where $\tilde{N}$ is equal to N (resp. $3N$) for grayscale (resp. color) images. In other words the distance between $\hat{\mathbf{x}}_\lambda$ and **y** is forced to be exactly equal to $\sqrt{\tilde{N}}\sigma$. As $\hat{\mathbf{x}}_\lambda$ belongs to the segment [$\hat{\mathbf{x}}_\lambda$, **y**], a such λ exists if and only if

$$d := \|\hat{\mathbf{x}}_0 - \mathbf{y}\| \geq \sqrt{\tilde{N}}\sigma \, .$$

In this case one can easily see that the only λ leading to the equality (18) is

$$\lambda = \frac{d}{\sqrt{\tilde{N}}\sigma} - 1 \, .$$

Despite this theoretical value, the algorithm has been tested with plenty of choices for λ. If λ is taken too large, then the contribution of the noisy image is too important and adds too much noise, which consequently reduces the PSNR, as one can see in the following table :

	$\lambda = 0$		$\lambda = 0.05$		$\lambda = 0.1$		$\lambda = 0.15$		$\lambda = 0.2$		$\lambda = 0.25$		$\lambda = 0.3$	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
2	44.43	1.53	**44.77**	**1.47**	44.70	1.48	44.52	1.52	44.32	1.55	44.13	1.58	43.97	1.61
5	39.03	2.85	**39.08**	**2.83**	38.97	2.87	38.78	2.93	38.57	3.01	38.34	3.08	38.12	3.16
10	34.55	4.77	**34.58**	**4.76**	34.53	4.79	34.42	4.84	34.28	4.92	34.13	5.01	33.96	5.11
20	30.47	7.64	**30.47**	**7.63**	30.44	7.66	30.38	7.72	30.30	7.79	30.20	7.88	30.08	7.98
30	**28.18**	**9.94**	**28.18**	**9.94**	28.15	9.98	28.10	10.03	28.03	10.11	27.96	10.20	27.86	10.30
40	**26.60**	**11.92**	26.58	11.95	26.55	11.99	26.50	12.06	26.45	12.14	26.38	12.24	26.30	12.35
60	**24.36**	**15.44**	24.33	15.49	24.29	15.55	24.25	15.63	24.20	15.73	24.14	15.83	24.07	15.95
80	**22.76**	**18.55**	22.73	18.62	22.69	18.71	22.64	18.81	22.59	18.92	22.54	19.03	22.48	19.17
100	**21.47**	**21.53**	21.43	21.63	21.38	21.74	21.34	21.86	21.28	21.99	21.23	22.13	21.17	22.28

In **bold** the best result for a given σ. Other parameters are fixed to : $K = 15$; $n = 25$; $\gamma = 5.25$; $k = 256$.

Here are some visual results :
σ = 10

λ = 0

λ = 0.05

λ = 0.15

λ = 0.25

σ = 30

$\lambda = 0$

$\lambda = 0.05$

$\lambda = 0.15$

$\lambda = 0.25$

$\sigma = 80$

$\lambda = 0$

$\lambda = 0.05$

$\lambda = 0.15$

$\lambda = 0.25$

The following table shows the comparison between the empirically obtained parameter (λ_e) and the theoretically obtained parameter (λ_t) :

	λ_t			λ_e		
σ	PSNR	RMSE	Value	PSNR	RMSE	Value
5	**38.83**	**2.92**	0.0050	38.64	2.98	0.05
10	**34.24**	**4.95**	0.0078	34.10	5.03	0.05
20	**29.85**	**8.20**	0.012	29.84	8.21	0.05
30	27.64	10.58	0.013	**27.68**	**10.54**	0.05
40	26.08	12.66	0.014	**26.10**	**12.63**	0.0
60	**24.05**	**16.00**	0.018	24.00	16.08	0.0
80	22.78	18.52	0.017	**22.80**	**18.46**	0.0
100	**21.88**	**20.54**	0.019	21.84	20.63	0.0

In **bold** the best result for a given σ. Other parameters are fixed to : $K = 15$; $n = 25$; $\gamma = 5.25$; $k = 256$.

In the end the final kept value for λ is the one given by (18).

Influence of the Number of Iterations *K*

The iterative aspect of the method is important, because it allows the dictionary to be updated and then to obtain a better sparse representation of the patches of the image. Moreover it allows to show empirically the convergence of the method. Indeed when K is large enough further iterations should improve the dictionary only marginally. Depending on the convergence of the method (which can change according to σ), a huge number of iterations is assumed to be needed in order to assure the best possible estimate. On the other side, each iteration is really expensive in terms of processing time. Thus avoiding spurious iterations allows one to obtain a faster algorithm. In consequence the main goal is to obtain a good compromise between having enough iterations to obtain a good result close to the optimum and having a correct processing time.

Here is a table showing the PSNR and RMSE evolutions depending on the number of iterations :

	$\sigma = 2$		$\sigma = 5$		$\sigma = 10$		$\sigma = 20$		$\sigma = 30$	
K	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
1	44.26	1.56	38.13	3.16	33.78	5.22	29.57	8.47	27.19	11.15
2	44.38	1.54	38.41	3.06	34.18	4.98	30.09	7.97	27.80	10.39
3	44.47	1.52	38.56	3.01	34.34	4.89	30.24	7.84	27.96	10.20
4	44.49	1.52	38.63	2.98	34.39	4.86	30.29	7.80	28.00	10.14
5	44.53	1.51	38.65	2.98	34.41	4.85	30.30	7.79	28.02	10.12
6	44.54	1.51	38.67	2.97	34.42	4.84	30.32	7.77	28.04	10.11
7	44.54	1.51	38.69	2.96	34.42	4.84	30.33	7.77	28.04	10.10
8	44.53	1.51	38.70	2.96	34.44	4.84	30.33	7.76	28.06	10.08
9	44.57	1.51	38.71	2.96	34.45	4.83	30.35	7.75	28.07	10.07
10	44.59	1.50	38.72	2.95	34.45	4.83	30.36	7.74	28.08	10.06
11	44.39	1.54	38.73	2.95	34.46	4.82	30.37	7.73	28.08	10.05
12	44.41	1.53	38.73	2.95	34.47	4.82	30.38	7.72	28.09	10.05
13	44.65	1.49	38.73	2.95	34.48	4.81	30.39	7.71	28.10	10.04
14	44.57	1.51	38.75	2.94	34.48	4.81	30.40	7.70	28.10	10.03
15	44.40	1.53	38.75	2.94	34.49	4.80	30.40	7.70	28.11	10.02
16	44.45	1.53	38.75	2.94	34.50	4.80	30.41	7.69	28.12	10.01
17	44.35	1.54	38.75	2.95	34.51	4.80	30.42	7.68	28.12	10.00
18	44.36	1.54	38.76	2.94	34.51	4.80	30.42	7.68	28.13	10.00
19	44.55	1.51	38.77	2.94	34.52	4.79	30.42	7.68	28.13	9.99
20	44.59	1.50	38.77	2.94	34.53	4.79	30.43	7.67	28.13	9.99

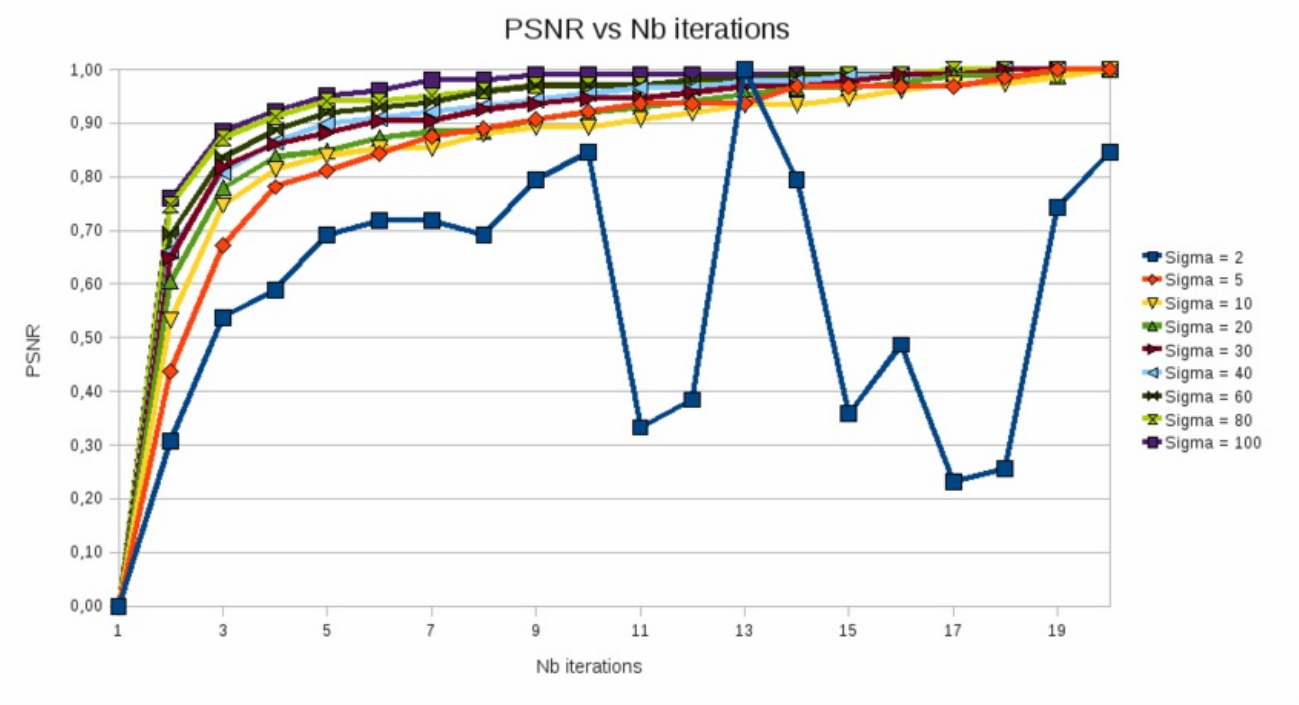
	$\sigma = 40$		$\sigma = 60$		$\sigma = 80$		$\sigma = 100$	
K	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
1	25.67	13.27	23.24	17.56	21.61	21.17	20.32	24.56
2	26.26	12.40	23.92	16.23	22.38	19.38	21.11	22.44
3	26.39	12.22	24.06	15.98	22.51	19.10	21.24	22.11
4	26.44	12.14	24.11	15.88	22.55	19.02	21.28	22.00
5	26.47	12.11	24.14	15.83	22.58	18.95	21.31	21.93
6	26.48	12.09	24.15	15.81	22.58	18.94	21.32	21.89
7	26.49	12.08	24.16	15.79	22.59	18.91	21.34	21.86
8	26.50	12.07	24.18	15.76	22.60	18.90	21.34	21.84
9	26.51	12.05	24.19	15.75	22.61	18.89	21.35	21.83
10	26.52	12.04	24.19	15.74	22.61	18.87	21.35	21.82
11	26.53	12.03	24.19	15.73	22.61	18.87	21.35	21.82
12	26.53	12.02	24.20	15.72	22.61	18.87	21.35	21.82
13	26.54	12.00	24.20	15.71	22.62	18.85	21.35	21.81
14	26.54	12.00	24.21	15.71	22.62	18.85	21.35	21.81
15	26.55	11.99	24.21	15.71	22.63	18.84	21.35	21.82
16	26.55	11.99	24.21	15.70	22.63	18.83	21.35	21.82
17	26.55	11.99	24.21	15.69	22.64	18.82	21.35	21.82
18	26.56	11.98	24.22	15.69	22.64	18.82	21.36	21.81
19	26.56	11.98	24.22	15.68	22.64	18.81	21.36	21.81
20	26.56	11.98	24.22	15.68	22.64	18.81	21.36	21.81

Other parameters are fixed to : $n = 5$; $\gamma = 5.25$; $\lambda = 0.15$; $k = 256$.

One can notice that for $\sigma \geq 5$ the PSNR converges, and the higher σ, the faster the convergence of the PSNR. Thus it is possible to keep few iterations for high values of noise. In order to better illustrate the speed of the PSNR convergence in function of K and σ, here is a Figure showing f(PSNR(i)) according to the number of iterations i, where f is defined by

$$f(x_i) = \frac{x_i - x_0}{x_m}$$

with $x_m = \max(x_i - x_0)$.



In the following, the number of iterations will therefore be fixed to $K = 15$, no matter what σ.

Influence of the Size of the Dictionary k

The only constraint on the size of the dictionary is to generate $\Re^n$. As we want some redundancy, we set $k \geq n$. Here is a study about this parameter :

	$k = 128$		$k = 196$		$k = 256$		$k = 320$	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
2	44.13	1.58	44.40	1.54	44.66	1.49	**44.70**	**1.48**
5	38.44	3.05	38.69	2.96	38.73	2.95	**38.76**	**2.94**
10	34.32	4.90	34.41	4.85	34.45	4.83	**34.50**	**4.80**
20	30.32	7.77	30.37	7.72	30.42	7.68	**30.44**	**7.67**
30	28.07	10.07	28.08	10.05	28.10	10.03	**28.10**	**10.04**
40	26.54	12.01	**26.55**	**11.99**	26.53	12.01	26.54	12.00
60	**24.37**	**15.42**	24.32	15.50	24.30	15.54	24.26	15.61
80	**22.73**	**18.63**	22.66	18.76	22.61	18.88	22.57	18.97
100	**21.48**	**21.60**	21.48	21.59	21.32	21.91	21.27	22.02

In **bold** the best result for a given σ. Other parameters are fixed to : $K = 15$; $n = 5$; $\gamma = 5.25$; $\lambda = 0.15$.

According to this table one can see that it might be interesting to choose larger sizes for the dictionary for relatively small noise ($\sigma \leq 30$), and smaller sizes for high noise ($\sigma \geq 60$). Although this parameter has an influence on the processing time, it remains relatively flexible according to PSNR results. In the following, this parameter will therefore be fixed to $k = 256$.

Influence of the Correction Parameter γ

The parameter γ is only used in the case of color images. We will see that this empirical parameter is quite flexible, because some low variations on its value have almost no consequences on the final result.

	$\gamma = 3.5$		$\gamma = 4.5$		$\gamma = 4.75$		$\gamma = 5$		$\gamma = 5.25$	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
2	44.24	1.56	**44.67**	**1.49**	44.66	1.49	44.64	1.49	44.66	1.49
5	38.75	2.94	**38.77**	**2.94**	38.75	2.94	38.75	2.94	**38.77**	**2.94**
10	34.48	4.81	34.50	4.80	34.47	4.82	34.49	4.81	34.50	4.80
20	**30.39**	**7.71**	**30.39**	**7.71**	**30.39**	**7.71**	30.37	7.72	30.35	7.74
30	**28.10**	**10.04**	28.08	10.05	28.09	10.04	**28.10**	**10.04**	28.09	10.05
40	26.53	12.02	26.52	12.04	26.51	12.06	26.52	12.03	26.50	12.06
60	24.29	15.56	24.27	15.59	24.28	15.57	24.27	15.60	24.26	15.62
80	22.68	18.72	**22.70**	**18.69**	22.68	18.72	22.64	18.81	22.69	18.71
100	**21.40**	**21.70**	21.36	21.79	21.37	21.78	21.38	21.74	21.37	21.78

	$\gamma = 5.5$		$\gamma = 5.75$		$\gamma = 6$		$\gamma = 7$	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
2	44.58	1.50	**44.67**	**1.49**	44.64	1.49	44.66	1.49
5	38.76	2.94	**38.77**	**2.94**	**38.77**	**2.94**	38.74	2.95
10	34.49	4.81	34.49	4.81	34.49	4.81	**34.51**	**4.80**
20	30.37	7.73	30.36	7.73	30.37	7.72	30.36	7.73
30	28.07	10.06	28.08	10.06	28.09	10.05	28.07	10.07
40	**26.54**	**12.01**	26.50	12.06	26.51	12.04	26.51	12.04
60	24.24	15.64	24.27	15.60	**24.29**	**15.55**	24.26	15.61
80	22.67	18.75	22.69	18.71	22.66	18.77	22.68	18.73
100	21.38	21.75	21.37	21.79	21.37	21.79	21.36	21.79

In **bold** the best result for a given σ. Other parameters are fixed to : $K = 15$; $n = 5$; $\lambda = 0.15$; $k = 256$.

In the following and according to the original article the correction parameter will therefore be fixed to $\gamma = 5.25$.

Influence of the Size of the Patches *n*

The size of the patches has a huge influence on the final result, and we can win several decibels in PSNR by choosing an appropriate *n*. As for most of the patch-based denoising method, best results are obtained by working with relatively big patches :

	$\sqrt{n}=3$		$\sqrt{n}=5$		$\sqrt{n}=7$		$\sqrt{n}=9$	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
5	38.52	3.02	**39.07**	**2.84**	38.87	2.90	38.55	3.01
10	33.87	5.16	**34.65**	**4.72**	34.45	4.83	34.18	4.98
20	29.27	8.77	**30.52**	**7.59**	30.32	7.77	30.04	8.02
30	26.46	12.12	**28.19**	**9.93**	28.11	10.03	27.78	10.41
40	24.40	15.36	**26.56**	**11.99**	26.55	12.00	26.24	12.42
60	21.51	21.42	24.42	15.33	**24.66**	**14.91**	24.37	15.41
80	19.30	27.63	22.72	18.64	**23.33**	**17.39**	23.15	17.74
100	17.56	33.78	21.47	21.53	**22.39**	**19.37**	22.38	19.38

	$\sqrt{n}=11$		$\sqrt{n}=13$		$\sqrt{n}=15$	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
5	38.42	3.06	38.12	3.17	36.80	3.68
10	33.89	5.15	33.62	5.31	33.37	5.47
20	29.74	8.31	29.48	8.56	29.26	8.77
30	27.48	10.78	27.20	11.12	26.99	11.41
40	25.90	12.92	25.60	13.38	25.35	13.77
60	24.06	15.98	23.69	16.67	23.41	17.21
80	22.87	18.32	22.59	18.92	22.28	19.60
100	22.13	19.98	21.84	20.63	21.56	21.31

In **bold** the best result for a given σ. Other parameters are fixed to : $K = 15$; $k = 256$; $\lambda = 0.05$; $\gamma = 5.25$.

Similarly to other patch-based denoising method (for example BM3D), it is necessary to increase the size of the patches when the noise increases. Despite the fact that according to PSNR/RMSE results it seems better to take relatively small patches ($\sqrt{n} = 5$ or 7) for small values of noise, we have to take into consideration the visual result. Here are visual results for several values of the noise and for all studied patch sizes :

σ = 10

Noisy image

$\sqrt{n} = 3$

$\sqrt{n} = 5$

$\sqrt{n} = 7$

$\sqrt{n} = 9$

$\sqrt{n} = 11$

$\sqrt{n} = 13$

$\sqrt{n} = 15$

$\sigma = 30$

Noisy image

$\sqrt{n} = 3$

$\sqrt{n} = 5$

$\sqrt{n} = 7$

$\sqrt{n} = 9$

$\sqrt{n} = 11$

$\sqrt{n} = 13$

$\sqrt{n} = 15$

$\sigma = 60$

Noisy image

$\sqrt{n} = 3$

$\sqrt{n} = 5$

$\sqrt{n} = 7$

$\sqrt{n} = 9$

$\sqrt{n} = 11$

$\sqrt{n} = 13$

$\sqrt{n} = 15$

$\sigma = 100$

Noisy image

$\sqrt{n} = 3$

$\sqrt{n} = 5$

$\sqrt{n} = 7$

$\sqrt{n} = 9$

$\sqrt{n} = 11$

$\sqrt{n} = 13$

$\sqrt{n} = 15$

One can notice that visually the choice is not so easy. Too small patches give huge artifacts, and lead to many low frequency fluctuations, but with big patches almost all details are lost. We get a visually nicer image, but completely blurred. In conclusion a compromise has to be found, which cannot only be chosen according to the PSNR/RMSE results, but also taken into account the visual aspect. The values of n which will therefore be kept are

σ	$0 < \sigma \leq 20$	$20 < \sigma \leq 60$	$60 < \sigma$
$\sqrt{n}$	5	7	9

A Detailed Study of Possible Variants

Origin of the Initial Dictionary

In the original article, and in the previously described algorithm, the dictionary is initialized by taking randomly k patches in the noisy image. Despite the fact that the method works well in this way, one can wonder whether there would be a better way to initialize the dictionary. For example by taking random patches from a noise-free image. Let us denote :

- **Init**$_0$: the dictionary is initialized on the original noiseless image;
- **Init**$_1$: the dictionary is initialized on the original noisy image;
- **Init**$_2$: the dictionary is initialized on a noise-free reference image.

	$\mathbf{Init}_0$		$\mathbf{Init}_1$		$\mathbf{Init}_2$	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
5	**38.80**	**2.93**	38.76	2.94	38.65	2.98
10	**34.47**	**4.82**	**34.42**	**4.82**	34.29	4.92
20	**30.40**	**7.70**	30.35	7.74	30.28	7.81
30	28.05	10.10	**28.09**	**10.04**	27.87	10.30
40	26.44	12.15	**26.51**	**12.05**	26.27	12.38
60	24.21	15.70	**24.25**	**15.63**	24.21	15.71
80	22.64	18.82	**22.65**	**18.79**	22.61	18.88
100	21.29	21.99	**21.31**	**21.93**	21.28	22.00

In **bold** the best result for a given σ. Parameters are fixed to : $K = 15$; $n = 25$; $\gamma = 5.25$; $k = 256$; $\lambda = 0.15$.

One can think that the initialization of the dictionary is quite important (because we run the algorithm with few number iterations, so the maximum is not reached), because depending on the initialization we have variations of more than 0.1dB. But when σ increases, one observes less variation in the results. An explanation might be that the number of iterations K is then more appropriate, so we are close to optimality, and the initialization is not really crucial.

In conclusion the initialization of the dictionary is not crucial, and the initialization by taking random patches from the noisy image is quite good.

Training of the Dictionary

Because the algorithm can hardly take into account parallel instructions (at least the update of the dictionary), the algorithm as previously described in this article is extremely slow. Its processing time is directly proportional to the size of the dictionary (although the study shows that the size of the dictionary can be reduced without affecting the result too much) as well as to the number of patches contained in the image (then to the size of the image itself) and to the size of the patches.

If obviously we cannot reduce the size of the image and if we cannot modify the size of the patches without highly damaging the final result, it is still possible to reduce the number of patches used during the training part of the dictionary, by applying the following principle :

1. The set of patches is built on the whole image;
2. Keep one patch out of T to build a T times smaller patch set;
3. Apply the loop on the ORMP and the update of the dictionary by SVD K times on this subset, in order to obtain a final dictionary D_f;
4. Then apply with only one iteration the whole algorithm on the initial full set of patches, but with D_f as previously obtained.

With this simple trick it is then possible to divide the processing time by slightly less than T (we have to apply at the end a single iteration on the whole set of patches, which can be slower than the previous 15 ones on the subset). Before applying this trick, we have to determinate its impact on the final result, in order to find the more appropriate value of T for each σ. We have seen during the study of the parameters that the result in PSNR for $\sigma = 2$ is highly chaotic depending on the number of iterations K. For that reason we do not present results for this

particular value of noise.

Nevertheless here is a summary table for some values of T ($T = 1$ represents the initial algorithm as described in this article without any modifications) :

	$T=1$		$T=2$		$T=4$		$T=8$		$T=12$		$T=16$		$T=20$	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
5	38.84	2.91	38.86	2.91	38.89	2.90	38.90	2.89	38.89	2.90	38.91	2.89	**38.92**	**2.88**
10	34.47	4.82	34.49	4.81	34.51	4.80	34.49	4.81	34.52	4.79	**34.55**	**4.78**	34.50	4.80
20	30.28	7.80	**30.31**	**7.78**	30.28	7.80	**30.31**	**7.78**	30.29	7.80	30.29	7.80	**30.31**	**7.78**
30	**28.05**	**10.09**	**28.05**	**10.09**	28.04	10.10	28.04	10.10	28.04	10.10	28.01	10.13	28.01	10.14
40	26.61	11.91	**26.63**	**11.88**	**26.63**	**11.88**	26.59	11.94	26.59	11.94	26.56	11.98	26.58	11.96
60	**24.69**	**14.87**	24.65	14.92	24.65	14.92	24.60	15.02	24.56	15.08	24.56	15.08	24.52	15.15
80	**23.28**	**17.47**	23.23	17.59	23.22	17.61	23.14	17.75	23.07	17.90	23.00	18.06	23.03	17.98
100	**22.28**	**19.61**	22.25	19.67	22.18	19.84	22.12	19.98	22.07	20.09	22.04	20.15	22.00	20.25

In **bold** the best result for a given σ. Parameters are fixed to : $K = 15$; $n = 49$; $\gamma = 5.25$; $k = 256$; $\lambda = 0.05$.

This study shows that it is possible to highly reduce the processing time of this method whilst keeping a result close to the original method. According to the obtained results, it seems reasonable to take $T = 16$ for $\sigma \leq 40$ and $T = 8$ for $\sigma > 40$.

In order to help the readers to make up their own idea concerning the gain in term of processing time with this trick, here is a table showing the processing time in seconds for a 512×512×3 image on a i5 processor with 8Go of Ram. Moreover the process of the ORMP is fully parallelized :

σ	5	10	20	30	40	60	80	100
$T = 1$	1306	446	213	165	152	141	138	137
T tabulated	140	53	28	23	22	29	29	28

Thanks to this trick, we obtain reasonable processing time for $\sigma \geq 10$. Moreover we can decrease this time to 112 seconds (resp. 42s.) for $\sigma = 5$ (resp. $\sigma = 10$) by taking $T = 32$, without decreasing the PSNR. But we cannot decrease the processing time more, because we have to process a single iteration of the full set of patches, which is mainly responsible for the processing time. One can be surprised by the fact that the processing time is decreasing with respect to σ. But it can be easily explained :

- For very small values of noise, it is quite complex to get a sparse representation of the patches since they are very different from one another. Then at the end of the ORMP we have to process a large matrix;
- On the contrary for very high noise the signal is covered by the noise, then patches are very similar. Thus it is easier to get a sparse representation of them, and then at the end of the ORMP the matrix is even smaller.

Comparison with several Classic and Recent Methods

In order to evaluate the real capacity of this new denoising method, a fair and precise comparison with other state-of-the-art methods needs to be done. The other considered methods are :

- BM3D;
- DCT denoising;
- NL-means;
- TV denoising;
- NL-Bayes.

Moreover, results for K-SVD will be shown for the algorithm which gives best PSNR results (named K-SVD 1 in the following) and the one which gives better visual results (K-SVD 2) (to know the difference, please see the study on the influence of the size of the patches n).

The following study has been led on the following noise-free color image ($\sigma_{reel} << 1$). All algorithms have been processed on the same noisy images obtained from noiseless images (saved in real values and not sampled on [0, 255])) : [show/hide]

Comparative Table

	TV denoising		NL-means		DCT denoising		K-SVD 1	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
5	35.57	4.25	37.37	3.45	38.92	2.89	39.04	2.85
10	31.61	6.70	33.44	5.43	34.25	4.94	34.60	4.75
20	28.09	10.05	29.70	8.35	29.92	8.14	30.46	7.65
30	26.19	12.50	27.07	11.30	27.58	10.65	28.17	9.95
40	24.94	14.44	25.57	13.43	26.14	12.58	26.72	11.76
60	23.31	17.42	23.36	17.32	24.19	15.74	24.66	14.91
80	22.26	19.66	21.83	20.66	22.96	18.14	23.40	17.24
100	21.52	21.41	20.85	23.12	22.08	20.07	22.45	19.23

	K-SVD 2		BM3D		NL-Bayes	
σ	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
5	39.06	2.84	39.53	2.69	39.92	2.57
10	34.59	4.75	35.13	4.47	35.43	4.32
20	30.48	7.63	31.00	7.19	31.27	6.97
30	28.14	9.99	28.74	9.32	28.98	9.07
40	26.70	11.79	27.09	11.27	27.48	10.78
60	24.57	15.07	25.39	13.71	25.61	13.37
80	23.31	17.42	24.21	15.71	24.25	15.63
100	22.05	20.14	23.21	17.62	23.26	17.52

According to the results, one can observe that the order of the methods is quite independent of noise value. In order to help the reader to make up its own idea concerning the method performances comparatively to the others, here is a summary table which shows a mean of the scores :

TV denoising		NL-means		DCT denoising		K-SVD 1	
PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE
26.69	13.30	27.40	12.88	28.26	11.64	28.69	11.04
K-SVD 2		BM3D		NL-Bayes			
PSNR	RMSE	PSNR	RMSE	PSNR	RMSE		
28.61	11.2	29.29	10.25	29.53	10.03		

Images

In addition to the PNSR/RMSE results, it is really interesting to compare visually those methods. Here are the results for $\sigma = 20$:[Show/Hide].

Noisy image

TV denoising

NL-means

DCT denoising

K-SVD 1

K-SVD 2

BM3D

NL-Bayes

Conclusion

In this article, we have proposed a detailed analysis of the K-SVD algorithm, already introduced in the articles [9] and [14]. Through this explanation, we showed why we could expect remarkable denoising results from this algorithm. But we also noticed immediately the difficulty of the related optimization problems.

On the numerical side, we have observed the stability of this method, but we also brought up its heavy computational cost. In spite of these drawbacks, our experiments have clarified the impact of the different parameters on the result, and thus we have proposed reliable values to tune some of them. Moreover, we showed some denoising experiments which prove that the K-SVD method leads to good results, both in terms of PSNR values and of visual quality. The skeptical reader can pursue our experiments by applying the proposed demonstration to the images of her choice. Finally, the suggested modification (taking into account only a subset of the patches of the image) seems to get similar results with an interesting reduction of the execution time.

In conclusion, the K-SVD method can be considered to be part of the state of the art. But, above all it has to be seen as a first successful use of dictionary learning to address an image processing task. The more recent algorithms of this field, in particular those which replace the l^0-sparsity constraint by a l^1 constraint (cf. [12]), seem very promising. They lead to a great gain in computational time, and therefore allow one to handle bigger images.

Acknowledgement

The authors are grateful to Julien Mairal for his help and advice.

Glossary

Global Notations

- **x** : generic notation for an image;

- x_0 : clean image;
- N : number of pixels of x_0;
- $\tilde{N}$: is equal to N (resp. $3N$) for a grayscale (resp. color) image;
- w : white Gaussian noise which is added to x_0;
- σ : noise standard deviation;
- $\mathbf{y}$: noisy image : $\mathbf{y} = x_0 + w$;
- $\hat{\mathbf{x}}$: denoised image obtained after applying the algorithm;
- $\hat{\mathbf{x}}_\lambda$: (in the paragraph 4.2) final denoised image obtained after applying the algorithm with the parameter λ;
- (i, j) : position of a generic pixel in the image $\mathbf{x}$;
- n : total number of pixels in a patch. As we are working with square patches, n is a perfect square;
- N_p : number of patches whose size is $\sqrt{n} \times \sqrt{n}$ contained in the image $\mathbf{x}$;
- $\mathbf{R}_{ij}$: matrix whose size is $n \times N$ making the extraction of a square patch whose size is $\sqrt{n} \times \sqrt{n}$ and whose up left pixel has coordinate (i, j). Columns of $\mathbf{R}_{ij}$ are indexed by the pixels of $\mathbf{x}$;
- $\mathbf{D}$: generic notation for a dictionary;
- d_l : column of index l ($1 \leq l \leq k$) of the dictionary $\mathbf{D}$;
- k : number of atoms in the dictionary;
- $\boldsymbol{\alpha}_{ij}$: generic notation for the representation of the patch $\mathbf{R}_{ij}x$ in the dictionary : $\mathbf{R}_{ij}x \approx \mathbf{D}\boldsymbol{\alpha}_{ij}$;
- $\boldsymbol{\alpha}$: matrix whose columns are formed by $\boldsymbol{\alpha}_{ij}$. Then columns of $\boldsymbol{\alpha}$ are indexed by (i, j) and the matrix has as many columns as there is patches of size $\sqrt{n} \times \sqrt{n}$ in the image $\mathbf{x}$;
- $\hat{\mathbf{D}}$: current dictionary (updated at each iteration of the algorithm);
- K : number of iterations of the algorithm;
- $\mathbf{D}_{init}$: initial dictionary;
- $\hat{\alpha}_{ij}$: current representation of the patch $\mathbf{R}_{ij}x$ in $\hat{\mathbf{D}}$ (updated for each iteration of the algorithm);
- $\hat{\boldsymbol{\alpha}}$: matrix whose columns are $\hat{\alpha}_{ij}$;

- λ : weighting of $||\mathbf{x} - \mathbf{y}||_2^2$ in the minimization problem (12). This coefficient is used during the reconstruction step;
- μ_{ij} : weighting of $||\boldsymbol{\alpha}_{ij}||_0$ in the minimization problem (12). This coefficient is not explicitely used in the algorithm;
- C : Thanks to this coefficient, the norm l^2 on n pixels of a white Gaussian noise whose standard deviation is σ is lower than $\sqrt{nC}\sigma$ with probability 0.93. This coefficient is used during the break condition of the ORMP;
- $\hat{d}_l$: column of the dictionary whose index is l, $(1 \leq l \leq k)$;
- $\hat{\alpha_{ij}}(l)$: coefficient of $\hat{\alpha_{ij}}$ of index l. It matches to the weighting of the atom $\hat{\mathbf{d}}_l$ in the representation of the patch $\mathbf{R}_{ij}x$;
- $e_{ij}^{\,l} = \mathbf{R}_{ij}\hat{\mathbf{x}} - \hat{\mathbf{D}}\hat{\alpha_{ij}} + \hat{\mathbf{d}}_l\hat{\alpha_{ij}}(l)$: residue corresponding to the atom l and the patch $\mathbf{R}_{ij}x$ (it is a column vector whose size is n);
- $\mathbf{E}_l$: matrix grouping the residues $e_{ij}^{\,l}$ together;
- $(U, \Delta\ V)$: singular value decomposition of $\mathbf{E}_l$;
- ω_l : set of indices all (i, j) such that $\hat{\alpha_{ij}}(l) \neq 0$;
- $\mathbf{I}$: identity square matrix whose size is $N \times N$;
- γ : parameter of the new metric of the ORMP for the color processing;
- $\mathbf{x}$, $\mathbf{y}$: generic notations for column vectors whose size is $\tilde{n}$;
- $\boldsymbol{\alpha}$: generic notation for the representation of a vector $\mathbf{x}$ in the dictionary $\mathbf{D}$: $\mathbf{x} \approx \mathbf{D}\boldsymbol{\alpha}$;
- J : square matrix whose size is $3n \times 3n$, built with three blocks of size $n \times n$ full of 1;
- $m_C(\mathbf{x})$: average of $\mathbf{x}$ in the channel C;
- I : square identity matrix, whose size is $3n \times 3n$;
- a : positive solution of $\gamma = 2a + a^2$. Then we get $a = \sqrt{(1 + \gamma)} - 1$;
- $\tilde{n}$: $\tilde{n} = n$ (resp. $3n$) if we are working on grayscale (resp. color) images;
- $\hat{\beta_{ij}}$: result of the ORMP for the current representation of the color patch $\mathbf{R}_{ij}x$ in $\hat{\mathbf{D}}$ with the metric $||.||$;
- D : diagonal matrix containing the inverse of the norm of the columns of $(I + (a/n)J)\mathbf{D}$.

Specific Notations for the Explanation of the ORMP

- x : generic vector of $\Re^n$;
- **D** : dictionary used to compute a sparse representation of **x**;
- d_0, ..., d_{k-1} : atoms of the dictionary (columns of **D**);
- $\boldsymbol{\alpha} \in \Re^k$: sparse representation of **x** in **D**;
- Proj_F : orthogonal projection onto the vectorial sub-space F;
- j : loop index of the ORMP, from 0 to k;
- l_j : index of the j^{th} chosen vector;
- $L_j = \{l_0, ..., l_j\}$;
- $r = \mathbf{x} - \text{Proj}_{\text{Vect}(d_{l_0}, ..., d_{l_{j-1}})}(\mathbf{x})$: current residue;
- $(t_{l_0}, ..., t_{l_{j-1}})$: orthogonal set of $(d_{l_0}, ..., d_{l_{j-1}})$ obtained by Gram-Schmidt process;
- $(e_{l_0}, ..., e_{l_{j-1}})$: orthonormal set of $(d_{l_0}, ..., d_{l_{j-1}})$ obtained by Gram-Schmidt process;
- $(t_{l_0}, ..., t_{l_{j-1}}, t_i^{(j)})$: orthogonal set of $(d_{l_0}, ..., d_{l_{j-1}}, d_i)$ obtained by Gram-Schmidt process;
- $(e_{l_0}, ..., e_{l_{j-1}}, e_i^{(j)})$: orthonormal set of $(d_{l_0}, ..., d_{l_{j-1}}, d_i)$ obtained by Gram-Schmidt process;
- a_{pq} : coordinate of e_{l_p} on the vector d_{l_q} (equal to 0 except if $p \leq q$).

Specific Notations for the Explanation of the Truncated SVD

- X : matrix on which the truncated SVD will be applied. Let us denote $X = U\Delta V^T$ with U and V orthogonal matrices, and coefficients of Δ are equal to 0 except on its first diagonal where they are non-negative and decreasing;
- U, Δ, V : exact SVD of X : $X = U\Delta V^T$;
- s : estimate of the biggest singular value of $X X^T$;
- s_{old} : value of s at the end of the previous loop;
- u : estimate of the first column of U;
- V : estimate of the first column of V;
- **max_iter** : maximal number of authorized iterations (fixed to 100 in the C++ code);
- ε : tolerance threshold controlling the break condition of the SVD (fixed to 10^{-6} in the C++

code).

Source code

A C/C++ implementation is provided.

- source Code : zip

It should compile on any system since it is only strict ANSI C/C++. It is distributed under the GPL licence. Basic compilation and usage instructions are included in the README.txt file. This code requires `libpng` to read/write PNG images and is limited to 8bit RGB or grayscale PNG image files. The same code is used for the online demo.

On Line Demo

An on-line demo of this algorithm is available. User can choose to run the normal algorithm, the fast algorithm, the algorithm tuned for best PSNR results and the algorithm tuned to give better visual results.

References

1. M. Aharon, M. Elad, and A. Bruckstein. "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation". *IEEE Transactions on image processing*, pages 9-12, 2005. DOI:10.1109/TSP.2006.881199
2. G. Allaire and S. M. Kaber. "Algèbre Linéaire Numérique". Ellipses, Paris, 2002. ISBN:2729810013
3. A. Buades, B. Coll, and J.M. Morel. "A non local algorithm for image denoising". *IEEE Computer Vision and Pattern Recognition", 2 :60-65, 2005. DOI:10.1109/CVPR.2005.38*
4. A. Chambolle. "An algorithm for total variation minimization and applications". *Journal of Mathematical Imaging and Vision*, 20 :89-97, 2004. DOI:10.1023/B:JMIV.0000011325.36760.1e
5. S. F. Cotter, R. Adler, R.D. Rao, and K. Kreutz-Delgado. "Forward sequential algorithms for best basis selection". In *Vision, Image and Signal Processing, IEE Proceedings*, 146: 235–244, 1999. DOI:10.1049/ip-vis:19990445
6. K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. "Image denoising by sparse 3D transform-domain collaborative filtering". *IEEE Transactions on image processing*, 16, 2007. DOI:10.1109/TIP.2007.901238
7. G. Davis, S. Mallat, and M. Avellaneda. "Adaptive greedy approximations". *Journal of constructive Approximation*, 13 :57–98, 1997. DOI:10.1007/BF02678430
8. D. Donoho and I. Johnstone. "Ideal spatial adaptation by wavelet shrinkage". *Biometrika*, 81 :425–455, 1993. DOI:10.1093/biomet/81.3.425
9. M. Elad and M. Aharon. "Image denoising via sparse and redundant representations over learned dictionaries". *IEEE Transactions on image processing*, 15(12) :3736–3745, 2006. DOI:10.1109/TIP.2006.881969
10. M. Lebrun, A. Buades, and J.M. Morel. "Implementation of the Non-local Bayes image denoising". *Image Processing on Line*. ipol.im . Workshop, 2011.
11. J. Mairal. "Représentations parcimonieuses en apprentissage statistique, traitement d'image et vision par ordinateur". PhD thesis, 2010.

12. J. Mairal, F. Bach, J. Ponce, and G. Sapiro. "Online learning for matrix factorization and sparse coding". *Journal of Machine Learning Research*, 11 :19–60, 2010.
13. J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. "Non-local sparse models for image restoration". In *ICCV'09*, 2272–2279, 2009. DOI:10.1109/ICCV.2009.5459452
14. J. Mairal, M. Elad, and G. Sapiro. "Sparse representation for color image restoration". *IEEE Transactions on image processing*, 17(1) :53–69, 2008. DOI:10.1109/TIP.2007.911828
15. S. Mallat and Z. Zhang. "Matching pursuits with time-frequency dictionaries". *IEEE Transactions on signal processing*, 41(12), December 1992. DOI:10.1109/78.258082
16. Y.C. Pati, R. Rezaiifar, Y.C. Pati R. Rezaiifar, and P. S. Krishnaprasad. "Orthogonal matching pursuit : Recursive function approximation with applications to wavelet decomposition". In *Proceedings of the 27th Annual Asilomar Conference on Signals, Systems, and Computers*, pages 40–44, 1993. DOI:10.1109/ACSSC.1993.342465
17. L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear total variation based noise removal algorithms". *Phys. D*, 60 :259–268, 1992. DOI:10.1016/0167-2789(92)90242-F
18. J.L. Starck, E.J. Candès, and D.L. Donoho. "The curvelet transform for image denoising". *IEEE Transactions on image processing*, 11 :670–684, 2002. DOI:10.1109/TIP.2002.1014998
19. L.P. Yaroslavsky. "Local adaptive image restoration and enhancement with the use of DFT and DCT in a running window". In *Proceedings of SPIE*, 2825 :2–13, 1996. DOI:10.1007/3-540-76076-8_114
20. L.P. Yaroslavsky, K.O. Egiazarian, and J.T. Astola. "Transform domain image restoration methods : review, comparison, and interpretation". In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 4304 :155–169, 2001. DOI:10.1117/12.424970

Image credits.

Castle : D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In Int'l Conf. Computer Vision, volume 2, pages 416–423, July 2001.

Frog : http://www.grenouille.info

Valdemossa : CC-BY A. Buades