IPOL IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2019–06–13.
Submitted on 2018–04–12, accepted on 2019–04–10.
ISSN 2105–1232 © 2019 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2019.226

Implementation of a Denoising Algorithm Based on High-Order Singular Value Decomposition of Tensors

Fabien Feschet

Université Clermont Auvergne, CNRS, SIGMA Clermont, Institut Pascal, F-63000 Clermont-Ferrand, France
(fabien.feschet@dg-medical.fr)

Communicated by Pablo Arias *Demo edited by* Pablo Arias

Abstract

This article presents an implementation of a denoising algorithm based on High-Order Singular Value Decomposition (HOSVD) of tensors. It belongs to the class of patch-based methods such as BM3D and NL-Bayes. It exploits the grouping of similar patches in a local neighborhood into a 3D matrix also called a third order tensor. Instead of performing a different processing in each dimension, as in BM3D for instance, it is based on the decomposition of a tensor simultaneously in all dimensions reducing it to a core tensor in a similar way as SVD does for matrices in computing the diagonal matrix of singular values. The core tensor is filtered and a tensor is reconstructed by inverting the HOSVD. As it is common in patch-based algorithms, all tensors containing a pixel are then merged to produce an output image.

Source Code

The C++ source code, the code documentation, and the online demo are accessible at the IPOL web page of this article.[1] Compilation and usage instructions are included in the `README.txt` file of the archive.

Keywords: denoising; sparsity; adaptive grouping; tensor; high-order singular value decomposition

1 Introduction

Denoising an image is one of the most studied problems in image processing due to its need in a wide variety of situations [15]. The simplest model for this problem considers the observed image being given as

$$\text{observation } [I] \;=\; \text{image } [I_{\text{clean}}] \;+\; \text{noise } [\eta]\,. \tag{1}$$

The model assumes additive noise and also assumes independence between clean image and noise. Usually the noise is supposed to be Gaussian with mean 0 and variance σ^2 which is considered as an

[1] https://doi.org/10.5201/ipol.2019.226

unknown of the problem. However, to study the denoising performance of an algorithm, we manage to fix the variance and the clean image I_{clean} is known. So, in the paper, σ is supposed to be known while it has to be estimated in practice. Many approaches have been proposed to solve the denoising problem either using Fourier transform [20], wavelets [5] or in the spatial domain [1, 18]. Two state-of-the-art methods are BM3D [4] and NL-Bayes [2], both called patch-based methods. Such methods use the principle of grouping similar small size patches in a local neighborhood, resulting in a three dimensional array of patches called a tensor. Then, either space-domain or transform-domain approaches are used on the group to denoise every patch. The grouping of patches allows collaborative-filtering and requires averaging, since any pixel of the image belongs to several patches. Having obtained a first estimation of the clean image I_{clean}, a final estimation is computed using either Wiener filtering for BM3D or collaborative filtering for NL-Bayes. This idea of iterative denoising is not new and also leads to methods like SOS Boosting [17] or DDID [7] and DA3D [14]. One of the drawbacks of iterative denoising is the tendency to oversmooth the denoised image due to the regularity principle used in the iterations.

In the present paper, we propose and discuss an implementation of another patch-based denoising algorithm presented by A. Rajwade et al. [16] and based on the theory of tensor decompositions [3]. The idea of this algorithm is that tensors, intuitively multi-dimensional matrices, offer the possibility to deal with a patch-grouping as a whole and thus to avoid performing different processings on different dimensions. The chosen decomposition is called HOSVD for High-Order Singular Value Decomposition. As the SVD does for matrices, the HOSVD computes a core tensor from a tensor and this core tensor can be manipulated for instance through a threshold transform. Since the HOSVD is easily invertible, the modified core tensor permits to reconstruct an approximate tensor of a patch-group which is the denoised tensor of the group. Rajwade et al. [16] also performed a second step, a Wiener-like filtering, to increase the PSNR and compared their algorithm with BM3D. In this paper, we compare with both BM3D and NL-Bayes using the implementation given in IPOL papers [9, 10]. We used the PSNR measure as it provides fast and reproducible comparisons of algorithms on the same images. We also focus on some artifacts arising in color image denoising to evaluate the use-case of the HOSVD algorithm. It must be noted that other tensor decompositions are available [3] but they are not taken into account in this paper.

Theoretical aspects of the denoising performance can also be tackled in the context of Oracle Denoising [11]. The work [16] and of Milanfar et al. [11] have been applied to two different contexts in Oracle denoising. Recall that Oracle denoising is based on knowledge of the clean image and thus cannot lead to practical algorithms. However, it permits to recover the best performances of denoising procedures and as such can be used as a gold standard. In [16] the Oracle denoising was based on the singular value decomposition of (clean) patches such that the singular vectors were perfectly known. Then, noisy patches were projected on the basis given by the singular vectors and classical manipulation, like hard thresholding, was performed on the noisy singular values. This led to far better denoising performances than any state-of-the-art methods. In [11] a different approach was taken. Given a denoising operator, written as a linear operator, the MSE performances between denoised image and true image was minimized. This led to the determination of the best shrinking operation on the eigenvalues of the operator so as to minimize MSE.

The paper is organized as follows. The main algorithm is presented in Section 2. The performance of this algorithm is compared with BM3D and NL-Bayes in Section 3 and the parameters of the HOSVD algorithm are studied in Section 4. Extensions are given using DA3D, SOS Boosting and we provide an Oracle-based denoising scenario in Section 5. Finally, we conclude the paper in Section 6.

2 Description of the Algorithm

The description of the HOSVD-based denoising algorithm follows the presentation given in [16]. We refer to the work of Cichocki et al [3] for tensor decompositions. The acronym HOSVD is called Multilinear SVD (or MLSVD) in [3] since it is now the acceptable name for the HOSVD first presented in [8]. We first present the tensors, then the HOSVD is introduced. After that, we provide the presentation of the algorithm given in [16] including the Wiener-like second step denoising. We then provide some remarks on the implementation that permit to avoid useless transforms in the code.

2.1 A Short Introduction to Tensors

Usual matrices are referred to as two-dimensional arrays since their indices are fully described by subsets of $\mathbb{N}^2$. This construction can obviously be extended providing higher dimensional indexing of the elements of arrays. Tensors are simply the generic term for multi-dimensional arrays. The **order** of a tensor is the dimension of its indexing set. A real tensor $\mathcal{T}$ of order N hence belongs to $\mathbb{R}^{I_1 \times \cdots \times I_N}$ and its elements or entries are $t(i_1, \cdots, i_N)$. By definition, order one tensors are vectors and order two tensors are matrices. A **subtensor** is a tensor having as indexes set a subset of the tensor indexes set. The order of a subtensor is the number of non constant dimensions.

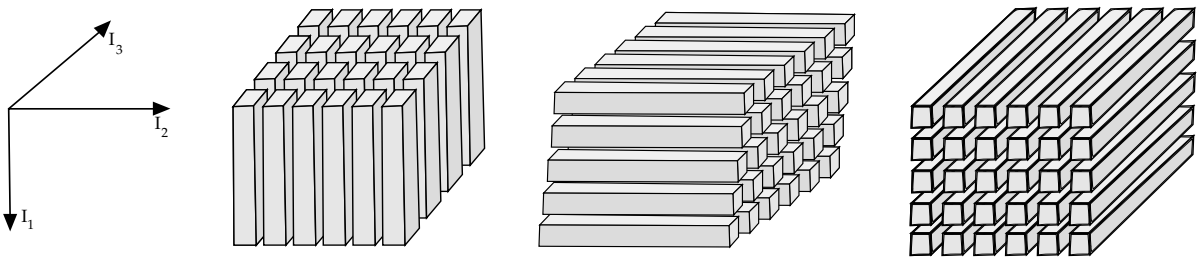


Figure 1: Fibers in all possible dimensions for a 3 dimensional tensor.

Fibers are order one subtensors using the whole set of indices for the solely varying dimension (see Figure 1). Hence for $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, a fiber is a tensor whose elements are $t(i_1, \cdots, :, \cdots, i_N)$ such that all dimensions except one is fixed.

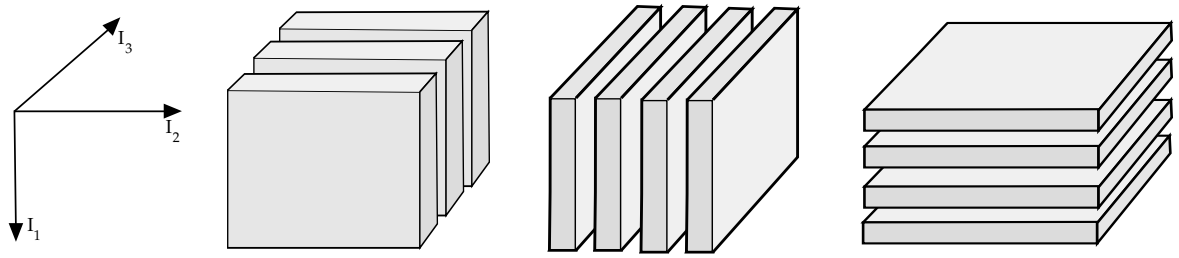
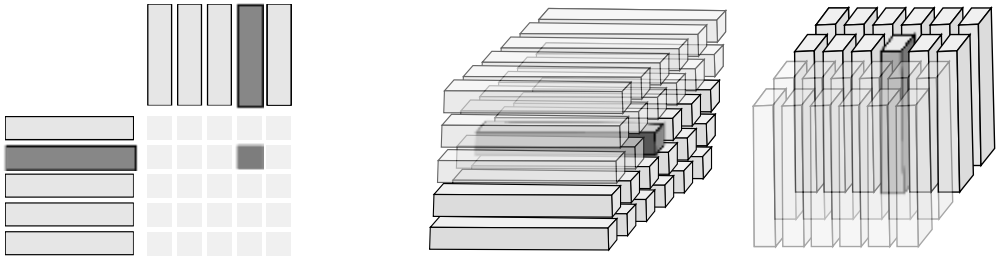


Figure 2: Slices for all possible pairs of dimensions for a 3 dimensional tensor.

A **slice** or **matrix slice** is an order two subtensor also having all indices in the non constant dimensions (see Figure 2). So, from a tensor, we can extract matrices by fixing all dimensions except two. A **tensor slice** is an order k subtensor where $k > 2$ dimensions are not fixed, still using all indices in the varying dimensions.

When defining operators like sum and product of tensors, care must be taken over the dimensions of the involved tensors and the definitions must be coherent with classical sum of vectors and matrices and classical vector product, matrix-vector product and matrix-matrix product. This is done by

reshaping tensors, that is rewriting all elements of a tensor into a tensor of different order. The used terms are *folding* and *unfolding*, where the latter flattens a tensor into a matrix. **Mode-n** unfolding (or matricization) of $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ yields a matrix $\boldsymbol{T}_{(n)} \in \mathbb{R}^{I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)}$ with entries $t(i_n, j_1 \cdots j_{n-1} j_{n+1} \cdots j_N)$ where the second index is the lexicographic or reverse lexicographic ordering of all dimensions except n. To see why those transforms help in defining operators which coincide with classical matrix operators, we might recall that when performing the product of two matrices (see Figure 3 left), each entry of the resulting matrix is a dot product uv^t where u is a (line) vector and v a column vector. Using fibers (see Figure 3 right), this dot product might be extended to the product of any pair of fibers of two tensors. Hence, the folding and unfolding operators permit to look only at the case of the operator between a tensor and a matrix, possibly obtained by folding, and where lengths coincide.



(left) (right)

Figure 3: (left) matrix product is a series of dot products. (right) Dot product is perfectly valid between fibers having the same length.

The **mode-n** product of $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\boldsymbol{M} \in \mathbb{R}^{J_n \times I_n}$ gives $\mathcal{P} \in \mathbb{R}^{I_1 \times \cdots \times J_n \times \cdots \times I_N}$ with $p(i_1, \cdots, i_{n-1}, j, i_{n+1}, \cdots, i_N) = \sum_{i_n=1}^{I_n} t(i_1, \cdots, i_{n-1}, i_n, i_{n+1}, \cdots, i_N) b(j, i_n)$ and is noted $\mathcal{P} = \mathcal{T} \times_n \boldsymbol{M}$. Using unfolding, we remark that this implies that $\boldsymbol{P}_{(n)} = \boldsymbol{M}\boldsymbol{T}_{(n)}$. For instance, when A and U are matrices, $A \times_1 U = UA$ and $A \times_2 U = AU^T$. Moreover, for $n \neq m$ and tensor $\mathcal{A}$ and matrices U and V, we have $\mathcal{A} \times_m U \times_n V = \mathcal{A} \times_n V \times_m U$. When $n = m$, care must be taken and we have $\mathcal{A} \times_n U \times_n V = \mathcal{A} \times_n (UV)$.

The **full multi-linear product** of a tensor $\mathcal{T}$ and a family of matrices $B^{(i)}$ is the generalization of the mode-n product to one matrix per dimension (see Figure 3 in [3]) as

$$[\![\mathcal{T}; B^{(1)}, \ldots, B^{(N)}]\!] = \mathcal{T} \times_1 B^{(1)} \times_2 B^{(2)} \ldots \times_N B^{(N)}.$$

The classical Kronecker product is also used and is defined as follows. $\boldsymbol{K} = \boldsymbol{M} \otimes \boldsymbol{N}$ with $\boldsymbol{M} \in \mathbb{R}^{I_1 \times I_2}$ and $\boldsymbol{N} \in \mathbb{R}^{J_1 \times J_2}$ giving $\boldsymbol{K} \in \mathbb{R}^{I_1 J_1 \times I_2 J_2}$ with $k((i_1-1)J_1+j_1, (i_2-1)J_2+j_2) = m(i_1, i_2)n(j_1, j_2)$.

2.2 HOSVD Decomposition

To introduce the HOSVD decomposition, it is necessary to introduce first the Tucker decomposition [19]. Indeed, HOSVD is only a special case of the Tucker decomposition. Let $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ be a given tensor. The Tucker decomposition of $\mathcal{T}$ is

$$\mathcal{T} = [\![\mathcal{S}; B^{(1)}, \ldots, B^{(N)}]\!] = \mathcal{S} \times_1 B^{(1)} \times_2 B^{(2)} \ldots \times_N B^{(N)},$$

where $\mathcal{S}$ is called the core-tensor and $B^{(n)}$ are called the mode-n component matrices. This decomposition can always be computed. Using mode-n unfolding, it can be written as

$$\boldsymbol{T}_{(n)} = B^{(n)} \boldsymbol{S}_{(n)} \left(B^{(1)} \otimes \ldots B^{(n-1)} \otimes B^{(n+1)} \otimes \ldots B^{(N)}\right)^T.$$

The HOSVD is then a Tucker decomposition where all mode-n component matrices $B^{(n)}$ are orthogonal and the core tensor $\mathcal{S}$ is all orthogonal. A tensor is said to be all orthogonal if the following two conditions are satisfied:

1. Slices in each mode are mutually orthogonal,
2. The Frobenius norm of slices in each mode are decreasing with the increase of the running index.

In practice, HOSVD is computed using classical SVD. Indeed, let us denote by

$$\mathcal{T} = [\![\mathcal{S}; U^{(1)}, \ldots, U^{(N)}]\!]_{\text{HOSVD}} ,$$

the HOSVD decomposition of tensor $\mathcal{T}$. Then for each mode-n unfolding of $\mathcal{T}$, we can compute its SVD leading to $\boldsymbol{T}_{(n)} = U^{(n)} \boldsymbol{\Sigma_n} {V^{(n)}}^T$. Then, the core tensor is computed using

$$\mathcal{S} = [\![\mathcal{T}; {U^{(1)}}^T, \ldots, {U^{(N)}}^T]\!] .$$

2.3 HOSVD-Based Denoising Algorithm

Building a denoising algorithm based on the HOSVD relies on the construction of a tensor. Patch-based methods are well suited for this construction, which is why the method proposed by A. Rajwade et al [16] is based upon tensors built from local patches. Indeed, as in other patch-based methods (see Figure 3 in [9] or Section 3.3 Grouping in [10]), the idea is the following. For a given $p \times p$ reference patch P_{ref}, a window around it is considered as well as all $p \times p$ patches P inside the window. This is done in order to keep the search for local similarity in a neighborhood of the reference patch. We define the similarity or distance between two patches by their normalized L_2 distance. A threshold is fixed on the distances as $\tau_d = 3\sigma^2 p^2$. Every patch P such that $|| P_{\text{ref}} - P ||> \tau_d$ is not considered in the sequel. So, after computing the distances for all patches in the window around P_{ref}, we have an ordered list, in increasing distance ordering, of all patches in the window. This list is cut if it contains more than $K_{\max}$ patches where $K_{\max}$ is a global constant. Hence, due to the threshold, the number of patches kept is at most $K_{\max}$, including the reference patch. This number is denoted by K and depends on the reference patch.

From this collection of K patches, it is easy to build a tensor, simply by collecting the patches along the third dimension of the tensor and keeping the patches as slices with respect to the first two dimensions of the tensor. So, we have a tensor $\mathcal{T} \in \mathbb{R}^{p \times p \times K}$. This corresponds obviously to the fact that each pixel value is coded in one dimension, thus corresponding to gray level images. For color images, the computed tensor has four dimensions, the third one being the color coding and the last one the K patches such that $\mathcal{T}_{\text{color}} \in \mathbb{R}^{p \times p \times 3 \times K}$. This is an advantage of tensors: for multi-components images nothing must be modified in the algorithm. For the explanation of the algorithm, the gray level case is described but readily the algorithm extends to the color case without any change.

The next step computes the HOSVD decomposition of $\mathcal{T}$ leading to

$$\mathcal{T} = [\![\mathcal{S}; U^{(1)}, \ldots, U^{(N)}]\!]_{\text{HOSVD}} .$$

The matrices $U^{(n)}$ are unaffected by the algorithm to ensure the inversion of the transform. Only the core tensor $\mathcal{S}$ is modified. Following [5], a hard threshold is applied on the entries of $\mathcal{S}$, the threshold being chosen as $\sigma\sqrt{2 \log p^2 K}$. This leads to the core tensor $\mathcal{S}_{\text{th}}$. Then a tensor is built by inverting the HOSVD

$$[\![\mathcal{S}_{\text{th}}; U^{(1)}, \ldots, U^{(N)}]\!] = \mathcal{T}_{\text{th}} .$$

The algorithm is repeated for all reference patches and then an averaging of the overlapping $\mathcal{T}_{\text{th}}$ is performed as in other patch-based denoising algorithms. A second pass is also performed and corresponds to applying a Wiener-like filtering. The idea is to use the result of the first pass of HOSVD to decompose the patches so as to correct the estimation of the core of noisy patches with respect to this first pass. This procedure is just a Wiener-like filtering but is not an optimal filter [12]. Hence, we must be aware that the gain obtained with this filtering might be lower than with the truly optimal filter. We decided to keep this method because it is the one used in the original paper [16]. The second pass used the output of the first pass to perform the block matching with the same threshold for similarity. The Wiener-like transform is given in the following equation (2) where $\hat{c}_n$ is the filtered core, $\hat{c}$ is the core obtained with the image of the first pass and c_n is the core of the noisy image using the block matching on the output of the first pass.

$$\hat{c}_n = c_n \frac{\hat{c}}{\hat{c} + \sigma^2} \,. \tag{2}$$

2.4 A Remark on the Implementation

An efficient implementation of the algorithm must deal with the tensor representation of the collection of similar patches. It should be noted that the HOSVD is computed from the SVD of the mode-n unfolding of the tensor $\mathcal{T}$. There are three unfoldings: $\boldsymbol{T}_{(1)} = U^{(1)}\boldsymbol{\Sigma_1}V^{(1)^T}$, $\boldsymbol{T}_{(2)} = U^{(2)}\boldsymbol{\Sigma_2}V^{(2)^T}$ and $\boldsymbol{T}_{(3)} = U^{(3)}\boldsymbol{\Sigma_3}V^{(3)^T}$. The main question is on the computation of the core tensor $\mathcal{S}$. We can choose any mode-n unfolding to perform the computation. For instance, using the mode-1 unfolding, we have

$$\boldsymbol{T}_{(1)} = U^{(1)}\boldsymbol{S}_{(1)}\left(U^{(2)} \otimes U^{(3)}\right)^T \,,$$

or in other words, since the Kronecker product of orthogonal matrices is itself an orthogonal matrix,

$$\boldsymbol{S}_{(1)} = U^{(1)^T}\boldsymbol{T}_{(1)}\left(U^{(2)} \otimes U^{(3)}\right) \,,$$

recalling that $(A \otimes B)^T = A^T \otimes B^T$ and $A^{T^T} = A$. Let us remark then that the threshold can be performed either on the entries of the core tensor or on its mode-1 unfolding. Indeed, the mode-1 unfolding of the threshold of $\mathcal{S}$ is the threshold of the mode-1 unfolding of $\mathcal{S}$ since unfolding is just reshaping a tensor. Hence, we can avoid representing the tensor and simply use its mode-1 unfolding matricization. This permits to use libraries dedicated to matrices such as `Eigen`[2] in the code. More generally, in the whole code, we represent tensors by their mode-n unfolding. This is by far the most efficient way to compute the HOSVD in a loop over all reference patches in the image.

2.5 Complexity Analysis

The detailed Algorithm 1 for HOSVD denoising contains two parts. In the first part, all denoised tensors are computed using hard thresholding of the core tensor of similar patches. Then in a second step, for each pixel of the output image, the tensors values at that pixel are simply averaged.

Let us first give the notations. The patch size is denoted by kHard in the code in a similar way as in the IPOL BM3D implementation [9]. The similarity is searched in a window of size nHard around the reference patch. The number of similar patches in this window is denoted by nSx_r. Finally, chnls is the number of channels in the image since grayscale images and color images must be processed differently.

The complexity of Algorithm 1 is guided by the size of the patches and the number of channels in the input image. Neglecting the pre-computation of the block matching, the algorithm needs to

[2]Eigen v3, G. Guennebaud and B. Jacob and others, 2010, http://eigen.tuxfamily.org

fill the four unfoldings in time proportional to the size of the tensor, that is : $\text{kHard}^2 \times \text{nSx}_r \times \text{chnls}$. Remember that nSx_r is bounded by a fixed constant K set to 30 in the original paper [16]. This bound is only justified by the computation time in the original implementation. Here, we also bound nSx_r in our implementation, even if it is a lot more faster than the original Matlab implementation of [16] and set the default bound to be the original bound of 30.

Algorithm 1: HOSVD denoising

input : I_{noisy}: original image, σ
output: I_{denoised}: Denoised image
Set kHard = 8 *//patch size*
Precomputes Block matching with threshold: $3 \times \text{chnls} \times \sigma^2 \times \text{kHard}^2$
//chnls is the number of channels (1 for grayscale image, 3 otherwise)
foreach *pixel position* (i, j) **do**
  //nSx_r is the number of similar patches to the patch at (i, j)
  Define matrix **unfold1**(kHard,kHard $\times$ nSx_r $\times$ chnls)
  Define matrix **unfold2**(kHard,kHard $\times$ nSx_r $\times$ chnls)
  Define matrix **unfold3**(chnls,$\text{kHard}^2 \times \text{nSx}_r$) *// unused if chnls==1*
  Define matrix **unfold4**(nSx_r,$\text{kHard}^2\times$ chnls)
  Fill unfold1, unfold2, unfold3 and unfold4 **from** I_{noisy}
  Compute the SVD svd_i **of** unfold_i
  Define matrix U42
  if *chnls == 1* **then**
    U42 = svd_4.matrixU() $\otimes$ svd_2.matrixU()
  if *chnls == 3* **then**
    U42 = svd_4.matrixU() $\otimes$ svd_3.matrixU() $\otimes$ svd_2.matrixU()
  Define matrix **core** = svd_1.matrixU()^T $\times$ unfold_1 $\times$ U42
  Threshold core using $\sqrt{2 \times \log(\text{kHard}^2 \times \text{nSx}_r \times \text{chnls})}$
  Store denoised tensor **fromCore** = svd_1.matrixU() $\times$ core $\times$ U42^T
Perform averaging of all overlapping patches to produce I_{denoised}

For the computations performed in the loop, the complexity of the SVD of a matrix of size $m \times n$ is $O(\min\{mn^2, m^2n\})$ [6]. Hence, using the definitions and sizes of the four unfoldings of the local tensor, we obtain a complexity $O(\max\{\text{chnls} \times \text{kHard}^4 \times \text{nSx}_r^2, \text{chnls}^2 \times \text{kHard}^3 \times \text{nSx}_r^2\})$. Considering that the dependency over the number of channels in the image is not a primary concern, we obtain a $O(\text{kHard}^4 \times \text{nSx}_r^2)$ complexity. This could be obviously changed for hyper spectral images when the number of channels exceeds the size of kHard. The cost of Kronecker products is linked to the sizes of U matrices in the SVD decompositions of the unfoldings. This gives $O(\text{kHard}^2 \times \text{nSx}_r^2)$ in both cases. Since all tensors are represented by matrices, the cost of the $\times$ operator is simply the cost of matrix-matrix multiplications. This leads to $O(\text{kHard}^3 \times \text{nSx}_r^2)$ for the computations of the 1-unfolding of the core tensor. The threshold operation and the inversion of the HOSVD obviously have this complexity, neglecting the dependency over chnls. So the whole complexity is bounded by $O(\text{nSx}_r^2 \times \text{kHard}^4)$ for each pixel position (i, j).

3 Evaluation of the Algorithm

In this section, we provide a comparison between three algorithms on the same set of images given in Table 1. Most of them are classical images. We added an image, the `pupuce` image of a cat because

it has some fast background color variations.

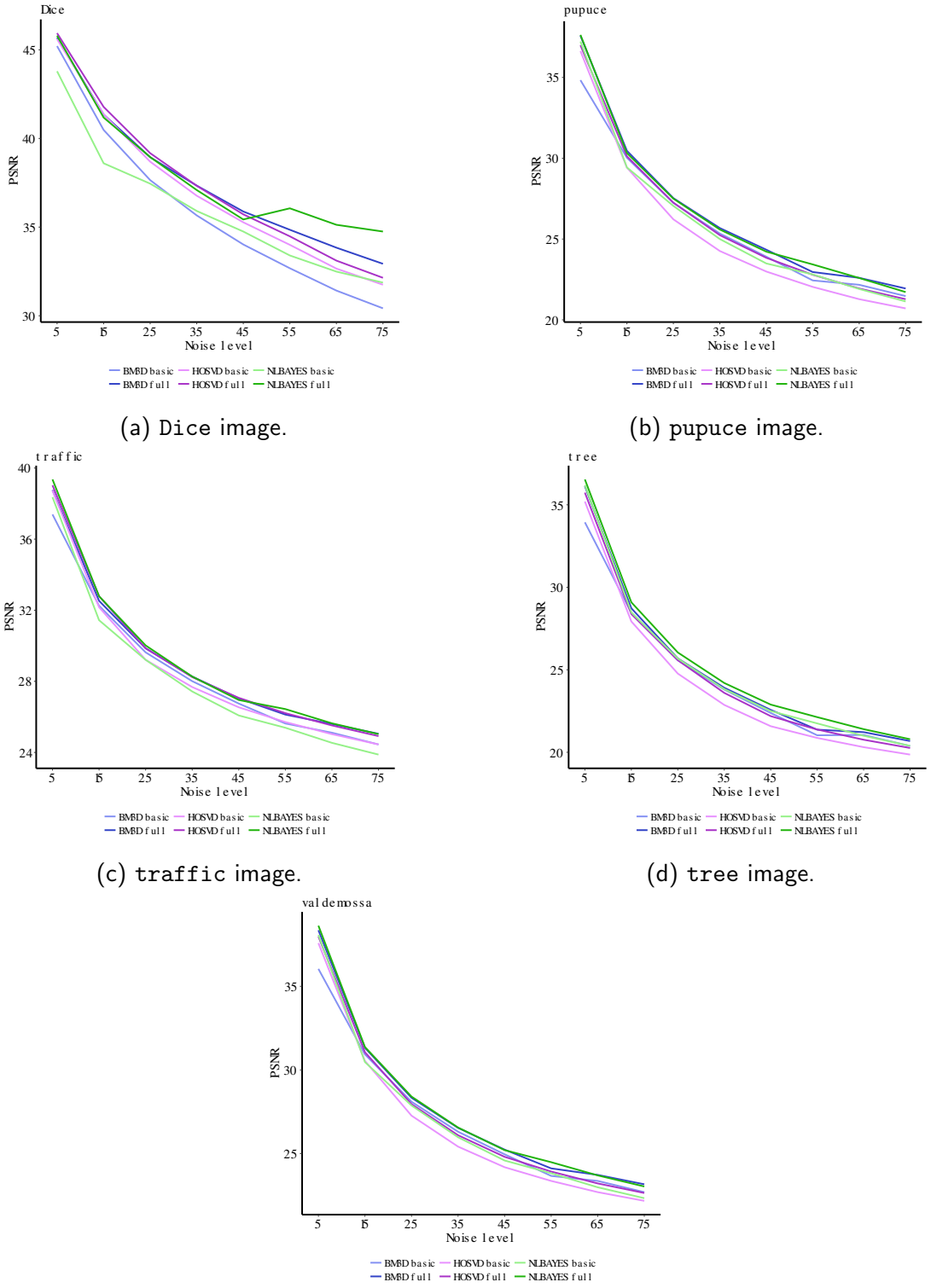
Table 1: Images used for the experiments.

The experimental process was as follows. First, for each image, the noise was added depending on the σ value. Then, the noisy image was saved. All algorithms were tested with the same input noisy image and with the same σ value which means that the results were comparable. PSNR and RMSE were computed using an external program to ensure identical behavior for all three algorithms. In the sequel since color image denoising is not comparable to gray level images denoising with respect to σ, we decided to separate color experiments from gray level experiments. It should be also noted that HOSVD has a high computation power demand and so is inherently slower than BM3D and NL-Bayes. The latter was the fastest method in average.

3.1 Comparison with BM3D and NL-Bayes

We tested the three algorithms HOSVD, BM3D and NL-Bayes using their IPOL implementations for the last two using the standard parameters given in the README file. To be fair we performed all experiments in the OPP color space since both BM3D and NL-Bayes used it. Moreover, default parameters for HOSVD were settled as in [16] and we postponed the tunning of HOSVD to the next sections. All algorithms used the normalized L2 distances between patches. For HOSVD, the block-matching computations were performed in all channels. Block-matching computations were performed only in the O channel for BM3D. For NL-Bayes, only the O channel was used in basic denoising while all channels were used in the full denoising for block-matching. In HOSVD, the threshold bound, originally only given for the RGB space [16], was adapted to OPP by summing $\sigma^2_{\text{channel}}$ over all channels to replace σ^2 in the original formulation. Color images comparisons are given in Figure 4 and gray level images comparisons are given in Figure 5.

Some comments can be done after those measurements. Globally, all algorithms performed well in the low and medium noise scenarios. For low noise, HOSVD was sometimes better than BM3D



(a) `Dice` image.

(b) `pupuce` image.

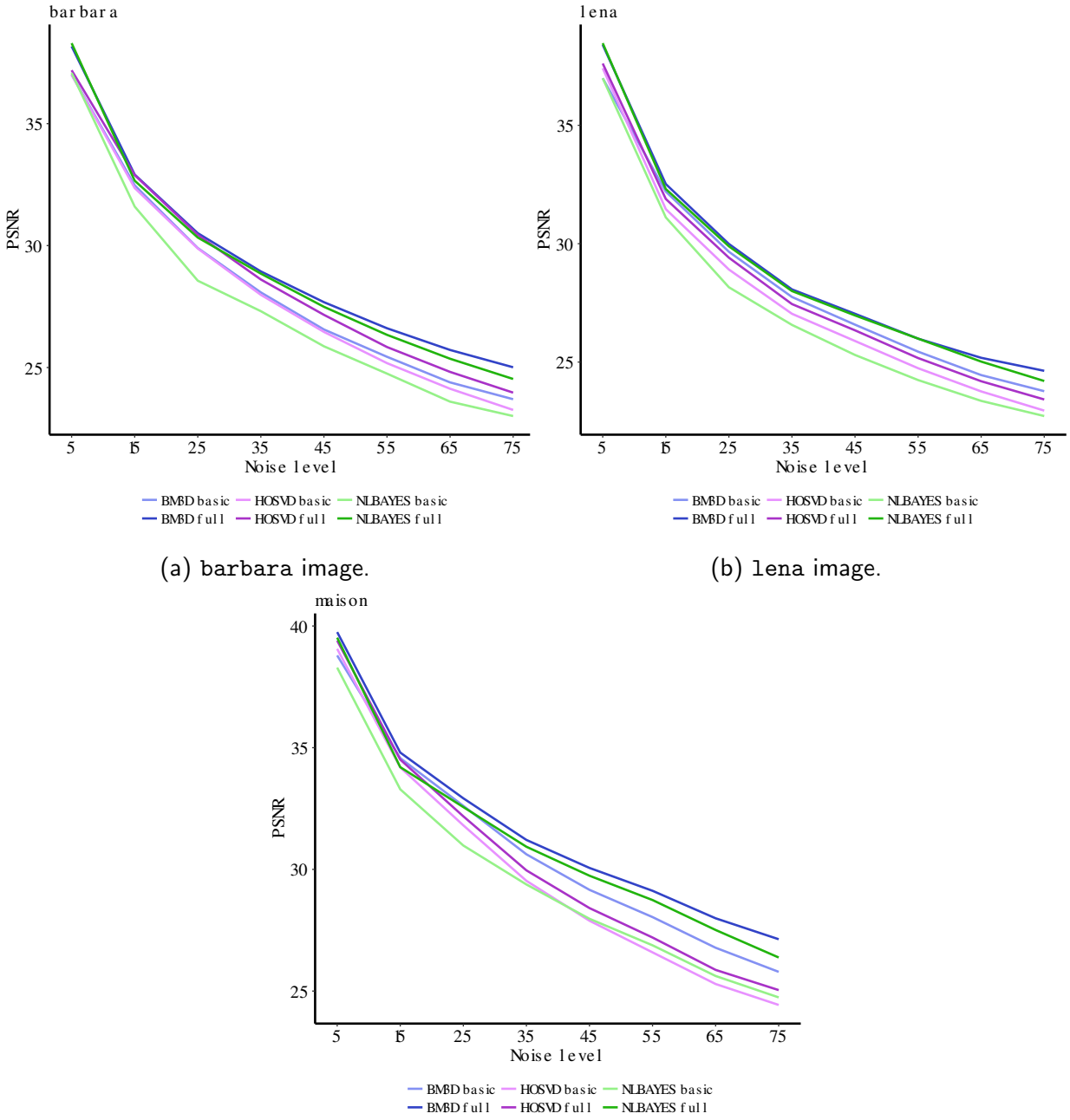
(c) `traffic` image.

(d) `tree` image.

(e) `valdemossa` image.

Figure 4: Comparison of color images with respect to the noise level using *basic* and *full* steps.

and NL-Bayes. For medium noise, NL-Bayes was an interesting competitor. For high noise, BM3D and NL-Bayes had nearly identical performances but NL-Bayes has less variations in its performance. HOSVD was not competitive with high noise. It must be remarked that the gains in PSNR obtained via the second step were rather different between all algorithms. As it can be seen in the Figures, the Wiener step of HOSVD did not significantly improved performances. This is in conformance with



(a) `barbara` image.

(b) `lena` image.

(c) `maison` image.

Figure 5: Comparison of gray level images with respect to the noise level using *basic* and *full* steps.

Rajade et al. [16]. We must recall that the Wiener step of HOSVD is only an approximate Wiener and thus has rather low performance. For the `traffic` image, HOSVD has similar performance than BM3D and NL-Bayes in all noise levels. Those comments are coherent with the results found in [16] and we can conclude that when correctly parametrized, HOSVD might be competitive but with default values it was not the case in average in the OPP color space.

With gray level images, the situation was easier to understand. Using the default parameters of [16], HOSVD was not competitive in terms of PSNR. Globally also, BM3D was superior to NL-Bayes in nearly all tests, especially in high noise.

To conclude, we must also add that the experiments were performed in the ideal scenario were the noise did not suffer from quantization as the noisy images were saved without quantization. It should be noted that the performances of all algorithms were rather different when quantization was

applied to the noisy images. In practice, this should be taken into account.

3.2 Color Effects

In [16], it was also mentioned that the performance of BM3D suffered from color artifacts in high noise conditions. So we study in this section, the behavior of the three algorithms on the `tree` image. With this image, we want to test the robustness of all algorithms on a complex background. We only provide experiments for two different values of the noise σ for which the behaviors of all algorithms differed. It should be noticed that the color artifacts shown here are not specific of the `tree` image but are also obtained for instance with the `Dice` image in the background or to a lesser extent in the `pupuce` image on the cat (see Figure 12).

Figure 6: HOSVD basic and full ($\sigma = 35$)

Figure 7: BM3D basic and full ($\sigma = 35$)

Figure 8: NL-Bayes basic and full ($\sigma = 35$)

Images obtained with $\sigma = 35$ are given in Figure 6 for HOSVD, in Figure 7 for BM3D and in Figure 8 for NL-Bayes. All computations are done in the OPP space.

With a higher level of noise ($\sigma = 75$ in Figures 9, 10 and 11), we can clearly observe that BM3D looses its robustness with respect to the original colors. This artifact is neither present for HOSVD nor for NL-Bayes. Moreover, we can observe why NL-Bayes is a clear winner. It is far better for restoring details of the images. HOSVD is good at preserving the original contrast. NL-Bayes has a tendency to produce flatten images, which explains partially its robustness.

Figure 9: HOSVD basic and full ($\sigma = 75$)

Figure 10: BM3D basic and full ($\sigma = 75$)

Figure 11: NL-Bayes basic and full ($\sigma = 75$)

Figure 12: BM3D full ($\sigma = 75$) resulting in erroneous colors.

4 Study of the Parameters

Let us recall the list of parameters that might influence the performance of the algorithm:

- the color space, which can be RGB or OPP
- kHard, the size of any patch
- $K_{\max}$, the maximum number of similar patches in the block-matching computation
- nHard, the size of the search window for the similarity of patches
- the threshold for the similarity between patches,
  initially fixed to $3 \times \text{chnls} \times \sigma^2 \times \text{kHard}^2$
- the threshold for hard-thresholding of the core of the tensor,
  initially fixed to $\sqrt{2 \times \log(\text{kHard}^2 \times \text{nSx}_r \times \text{chnls})}$
- the acceleration factor pHard, as in the BM3D implementation [9]

As usual, since it is useless to test the surface answer of the algorithm to a combination of all the parameters, in the sequel, only one or two parameters were modified at a time and the others were usually fixed using its original values [16]. Except for Section 4.1, all experiments were done in the OPP color space.

4.1 Influence of Color Space

In [16], the RGB color space was used. However, both BM3D and NL-Bayes work in the OPP color space. Hence, we studied the influence of the color space in the denoising performance. Results are given in Figure 13.

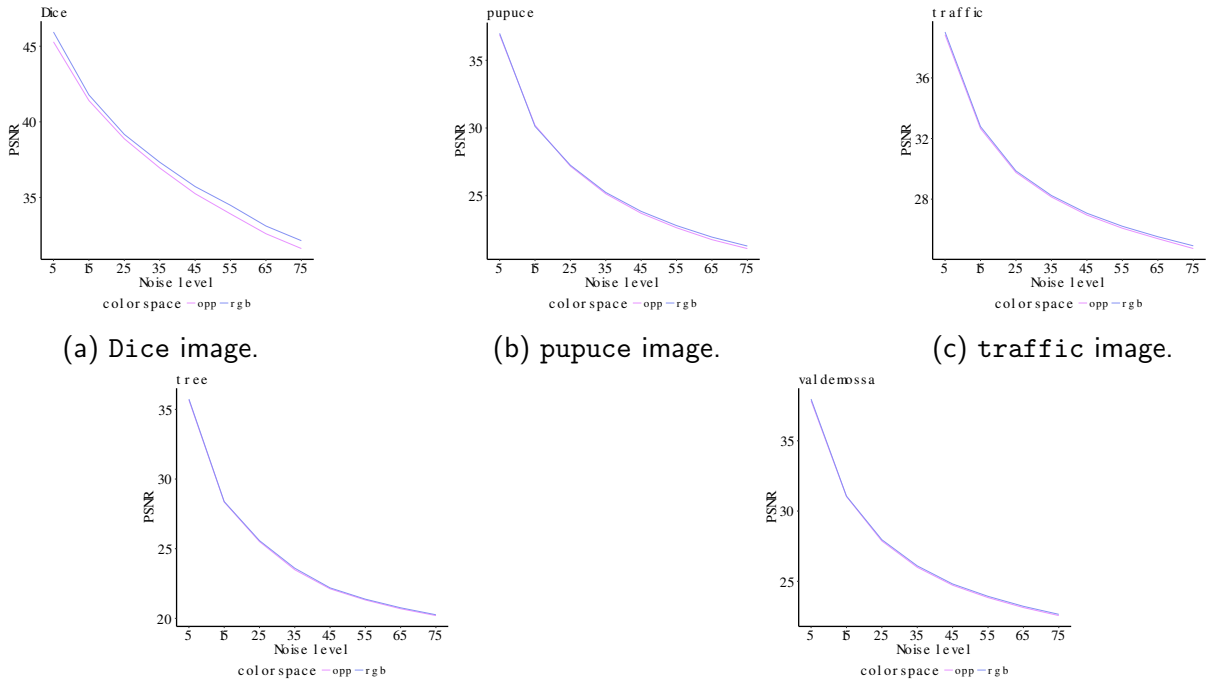
Contrarily to what was found for BM3D, we found that HOSVD had better performances in the RGB color space. Sometimes, the increase in PSNR was low, sometimes it had a mix impact on PSNR. As we did not found any case were using the RGB color space led to a decrease in PSNR, we must recommend its use for color image denoising.

4.2 Influence of kHard and $K_{\max}$

We decided to conduct an experiment where both kHard and $K_{\max}$ varied. For each image, kHard was varied from 5 to 11 and $K_{\max}$ from 20 to 80. We selected four noise levels: low/moderate ($\sigma = 15$), moderate ($\sigma = 25$) moderate/high ($\sigma = 45$) and high ($\sigma = 75$). Color images denoising results are given in Figure 14 and results for gray level images are given in Figure 15.

For the `Dice` image, changing kHard had a significant impact for any $K_{\max}$. But this was the only image for which it was the case (see Figure 14). As noted by an anonymous reviewer in a preliminary version of this paper, it is however counter-intuitive to see that for high noise, lowering kHard still had a positive influence on PSNR whatever $K_{\max}$ was. The other images had a more interpretable behavior with a low influence of kHard on the PSNR at the end of the full denoising. It seems that always choosing a low kHard is not a bad advice despite the non reliability of the normalized L_2 distances between patches for high noise.

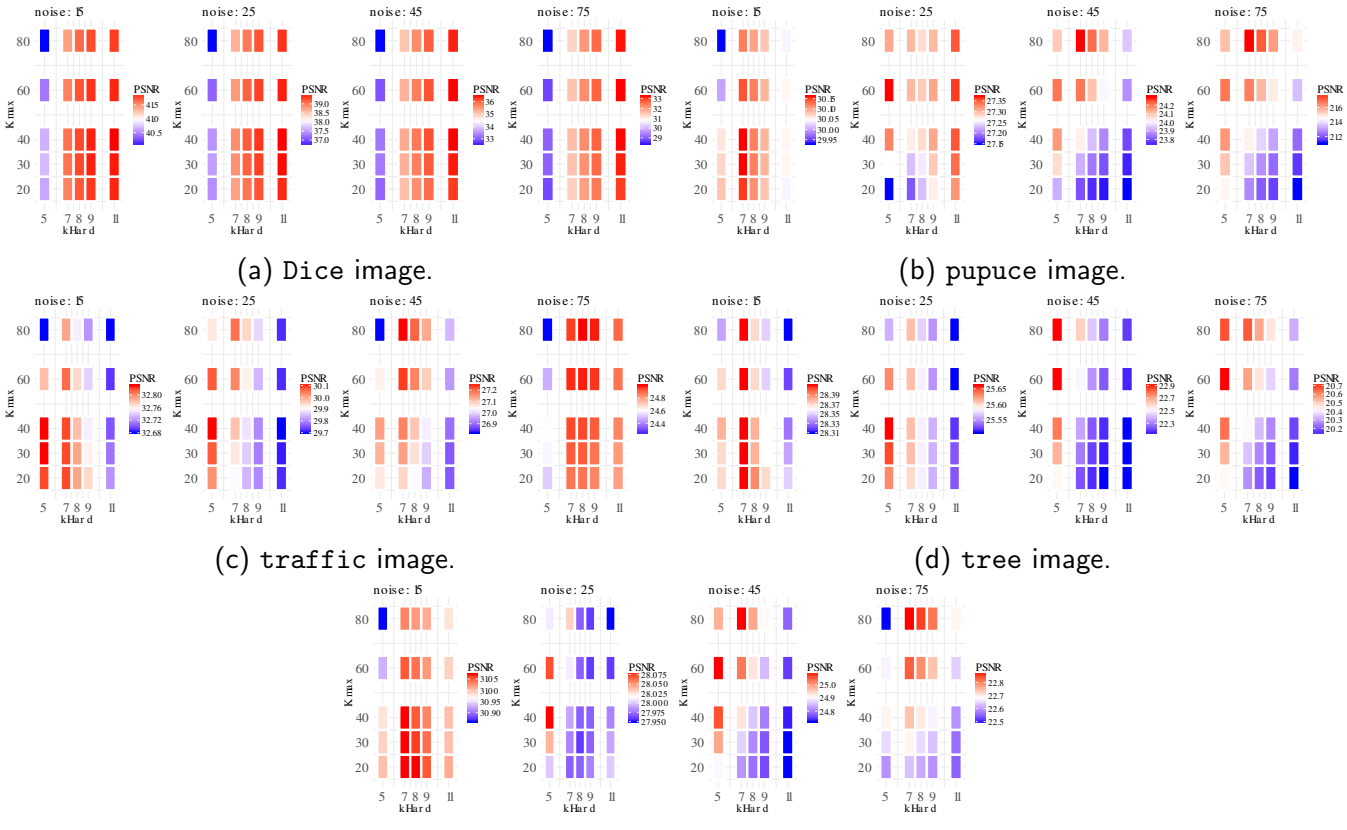
The influence of $K_{\max}$ can be assessed by looking at the PSNR with fixed kHard and varying $K_{\max}$. It seems that $K_{\max}$ should be chosen to be under 40 in all images. Increasing $K_{\max}$ over 40 did not lead to an increase in the PSNR.



(a) Dice image. (b) pupuce image. (c) traffic image.

(d) tree image. (e) valdemossa image.

Figure 13: color space influence for color images using full step.



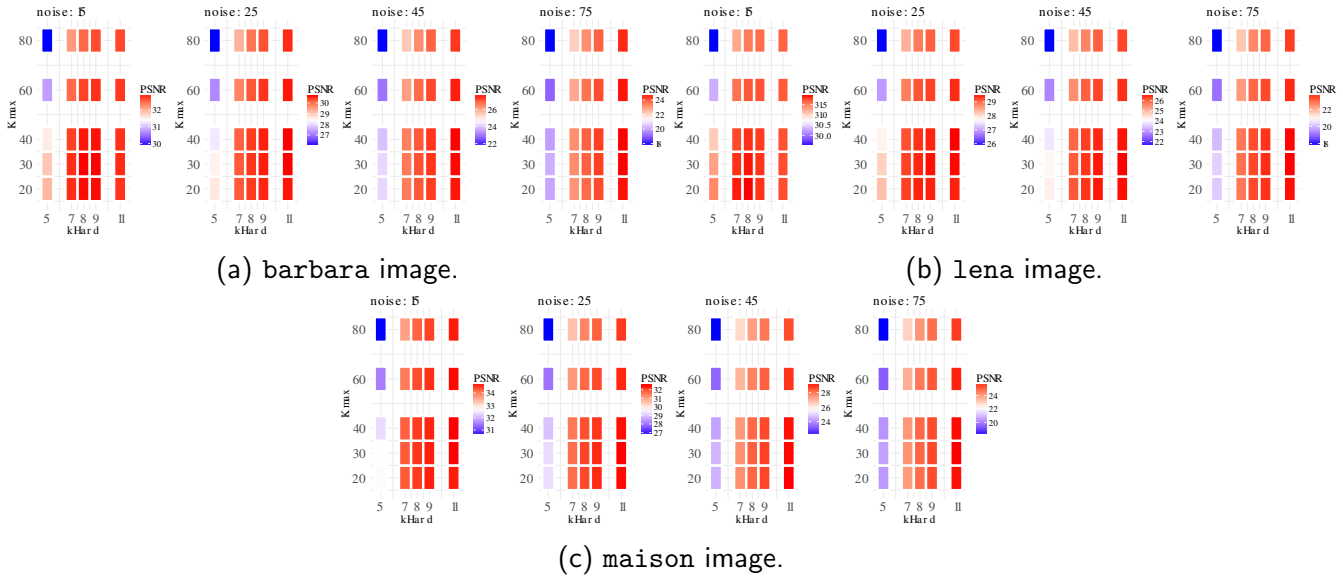
(a) Dice image. (b) pupuce image.

(c) traffic image. (d) tree image.

(e) valdemossa image.

Figure 14: Influence of kHard and $K_{\max}$ for color images using *full* step.

As seen in Figure 15, the behavior of HOSVD on gray level images was much more classical since increasing kHard had a positive influence on the PSNR. Again choosing a high value $K_{\max}$ did not



(a) `barbara` image.

(b) `lena` image.

(c) `maison` image.

Figure 15: Influence of kHard and $K_{\max}$ for gray level images using *full* step.

change the PSNR so a low value is acceptable for gray level images.

4.3 Influence of the Search Window Size

nHard is the half-size of the searching window for similar patches and is initially set to 16 as default. Obviously, fixing this parameter is linked with the spatial variability of both noise and image structures. Indeed, if the noise varies spatially, low values should be chosen. This is the same for restoring different structures in a noisy image. Color images denoising results are shown in Figure 16 and results for gray level images are displayed in Figure 17. The value of nHard was chosen to vary from 10 to 28 while $K_{\max}$ varied from 20 to 80.

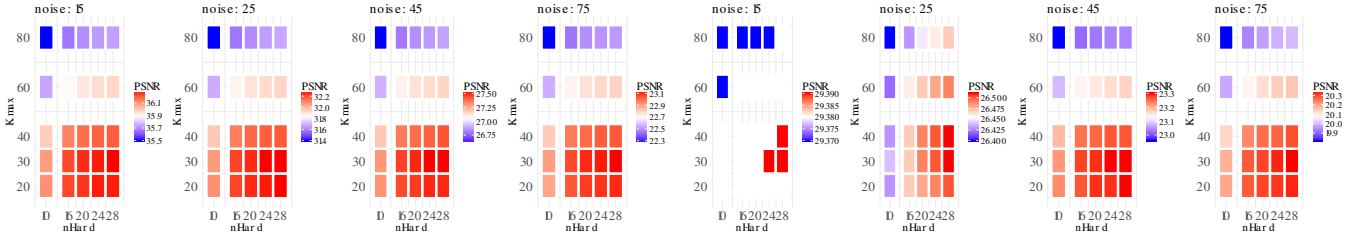
As depicted in Figure 16, the influence of nHard on the PSNR was logical for color images. Increasing nHard led to better results but they cannot be considered as significantly better. For gray level images (see Figure 17), the result was opposite. Indeed, an increase in nHard led to a decrease in PSNR. So the value of nHard should be lower for gray level images than for color images. A value of nHard equal to 16 is sufficient for gray level images while nHard=20 is a good choice for color images.

4.4 Influence of the Block-Matching Threshold

The block-matching threshold is the threshold on the similarity of patches. Its influence was tested by varying its value from 0.4 to 2.0 as a multiplicative constant. Except for the `tree` image, the block-matching threshold did not led to PSNR variations except for the lowest value (see Figure 18 for color images and Figure 19 for gray level images). We cannot consider that the block-matching threshold had an influence on the PSNR so its default value of 1.0 is an acceptable value.

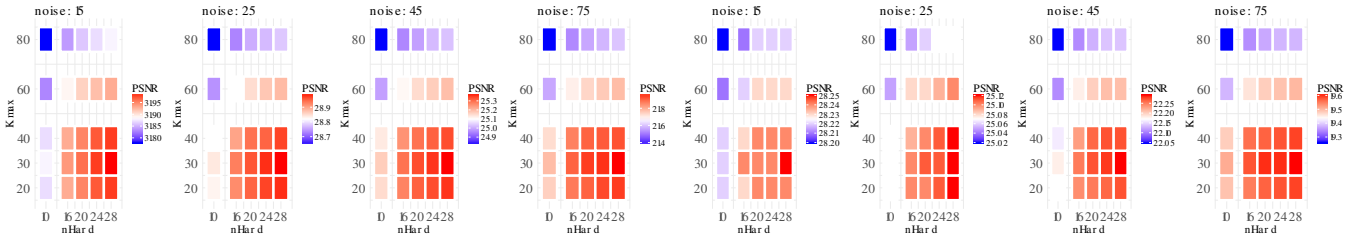
4.5 Influence of the Hard Threshold

The core of the HOSVD of the tensor of patches is hard thresholded [16]. This parameter was expected to be crucial in the performance of the HOSVD denoising as it is the case with classical SVD denoising. The threshold is based on the same approach than the one for classical SVD denoising using the result [5] on the core tensor. Its influence was tested by setting its value from 0.4 to 2.0 as



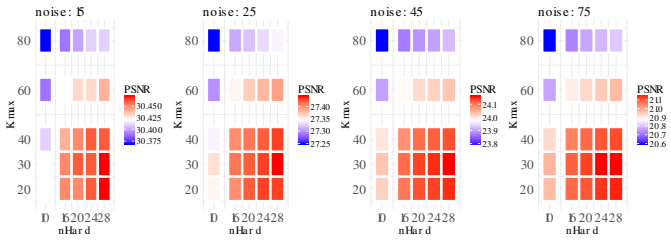
(a) `Dice` image.

(b) `pupuce` image.



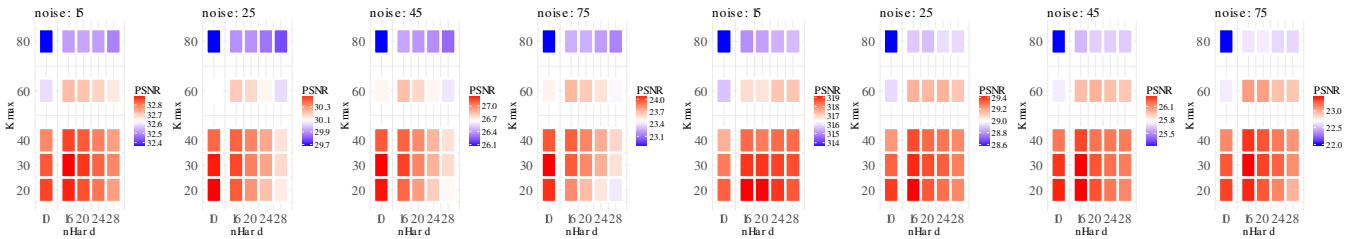
(c) `traffic` image.

(d) `tree` image.



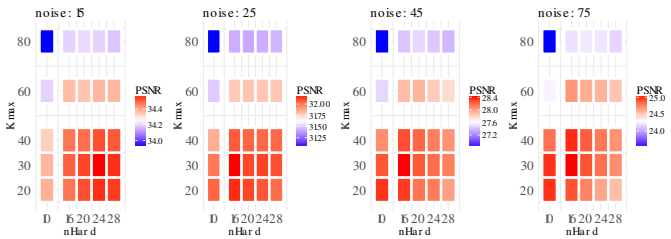
(e) `valdemossa` image.

Figure 16: Influence of nHard and $K_{\max}$ for color images using *full* step.



(a) `barbara` image.

(b) `lena` image.

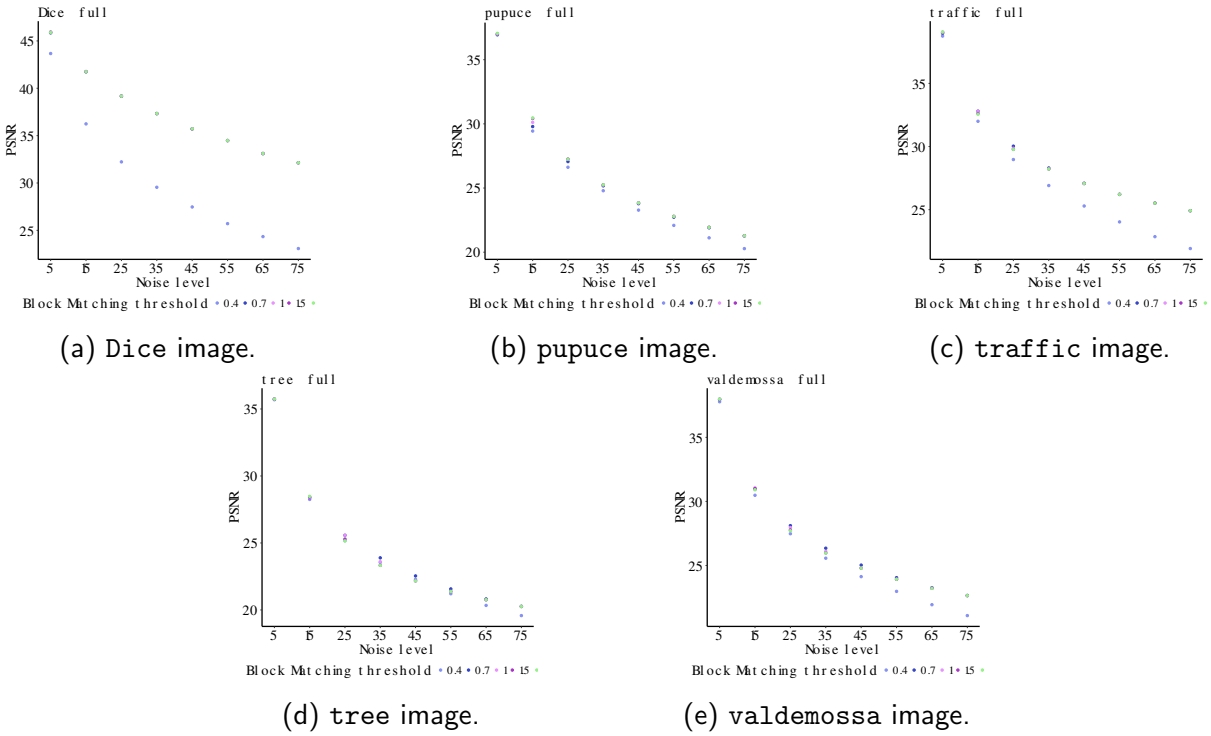


(c) `maison` image.

Figure 17: Influence of nHard and $K_{\max}$ for gray level images using *full* step.

a multiplicative constant. Results are given in Figure 20 for color images and in Figure 21 for gray level images.

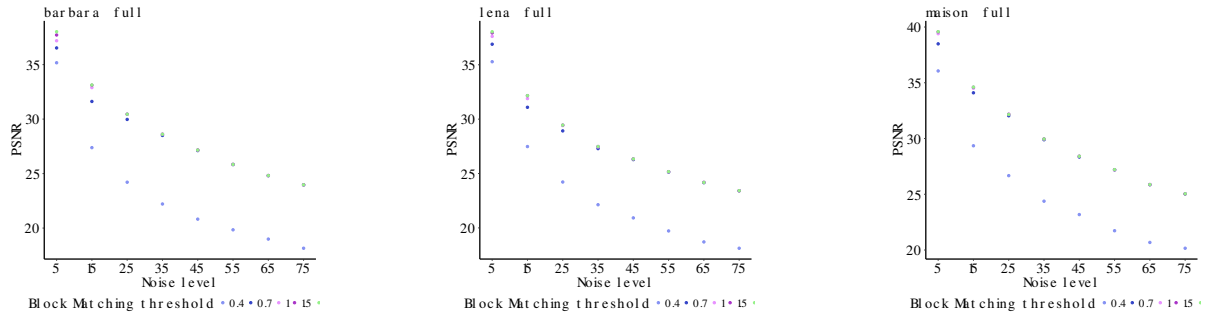
The obtained results were unexpexted. Indeed, the value of 0.7 was better for color images except for the `Dice` image. However for gray level images, values from 1.0 to 2.0 resulted in nearly the same PSNR values. Values below 1.0 were significantly worse so cannot be considered as good choices for gray level images. We hence recommend the value of 0.7 for color images and the value of 1.0 for



(a) `Dice` image. (b) `pupuce` image. (c) `traffic` image.

(d) `tree` image. (e) `valdemossa` image.

Figure 18: Influence of the block-matching threshold for color images using *full* step.



(a) `barbara` image. (b) `lena` image. (c) `maison` image.

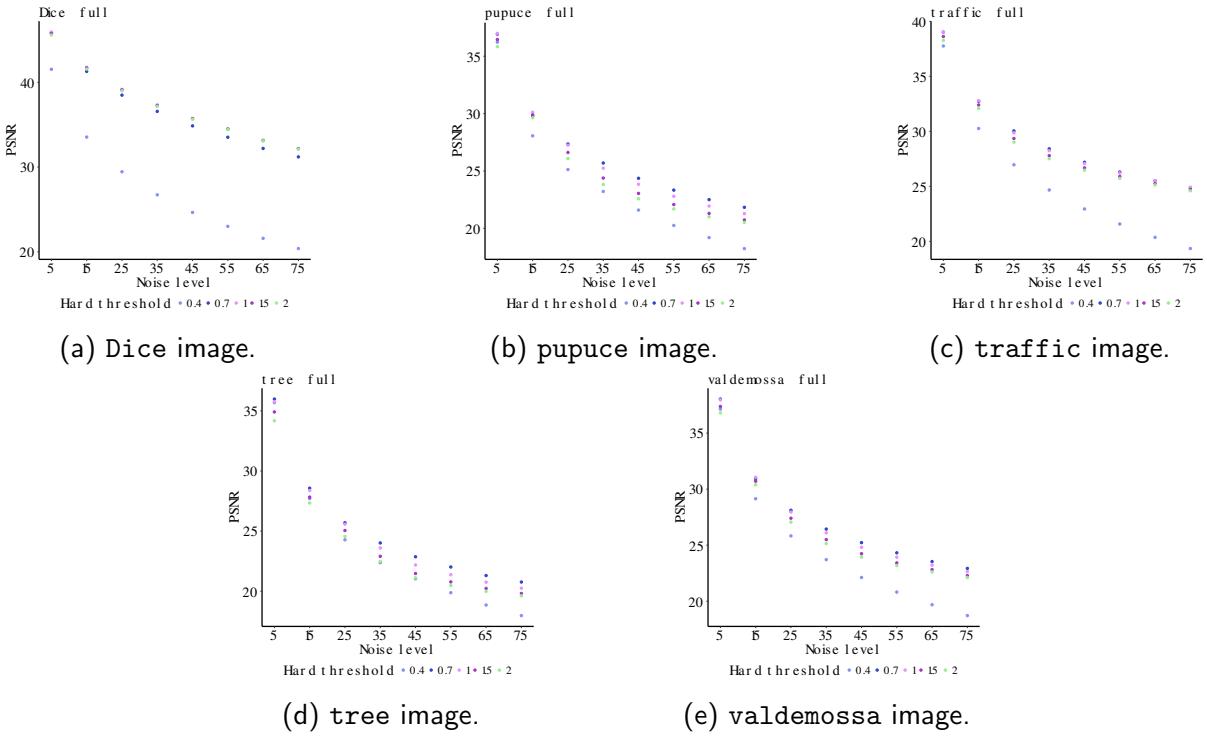
Figure 19: Influence of the block-matching threshold for gray level images using *full* step.

gray level images.

4.6 Influence of the Acceleration Factor

The acceleration factor pHard is set to 3 as default and has the same meaning than in the BM3D IPOL implementation [9]. The idea is that potentially the performance of the algorithm can be kept high while reducing the computation time using partial computations. Results are given in Figure 22 for color images and in Figure 23 for gray level images.

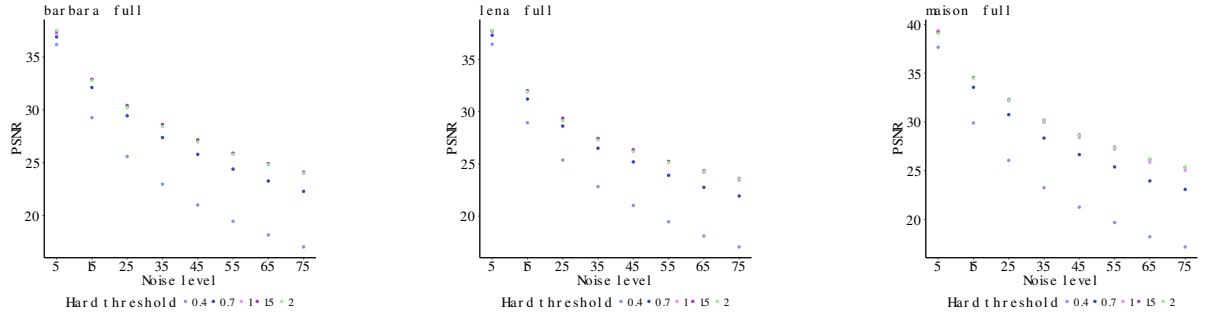
For color images (see Figure 22), the acceleration factor had a small positive influence on PSNR under low noise whereas it had no impact from medium to high noise. For gray level images (see Figure 23), even in low noise, the acceleration factor had almost no influence on the PSNR. Hence, with the default value of 8×8 patches, the acceleration factor can be chosen as high as 5. Since HOSVD has a high computational complexity, we hence recommend to use an acceleration factor of at least 3. This was the default value in all our experiments.



(a) `Dice` image. (b) `pupuce` image. (c) `traffic` image.

(d) `tree` image. (e) `valdemossa` image.

Figure 20: Influence of the hard threshold for color images using *full* step.



(a) `barbara` image. (b) `lena` image. (c) `maison` image.

Figure 21: Influence of the hard threshold for gray level images using *full* step.

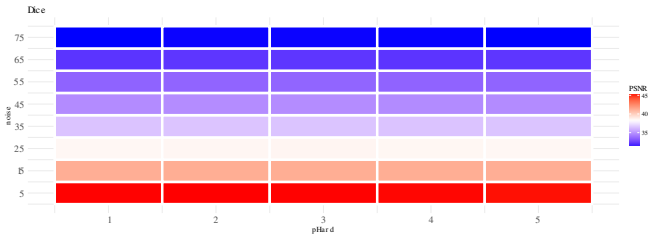
5 Extensions

In this section, we introduce several extensions of the implementation of the presented algorithm [16]. We propose to perform iterative denoising with the one step enhancement introduced in DA3D [14] and the iterative SOS Boosting procedure [17]. These extensions are followed by an Oracle scenario given in [16] for the SVD denoising and applied here to the HOSVD decomposition.

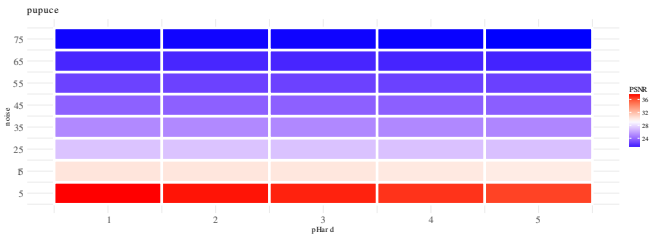
5.1 A Step Further

We tested the use of the DA3D method [14] as a black-box to compute the influence of one step of iteration on the result of a denoising algorithm. We used Wiener filtering of HOSVD as the input of the DA3D algorithm. Results are given in Figure 24 for color images and in Figure 25 for gray level images.

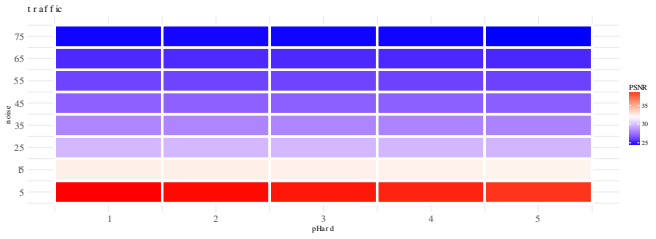
In average, no significant increase in PSNR was obtained for HOSVD on color images, except for the `Dice` image. Indeed, if the clean image contains important smooth parts then the DA3D method



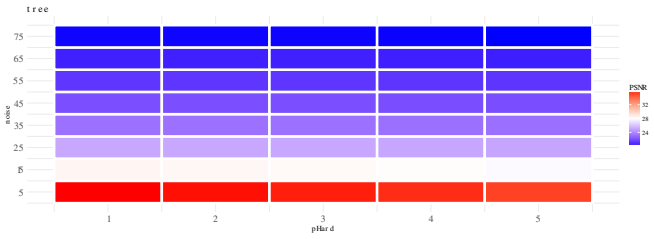
(a) `Dice` image.



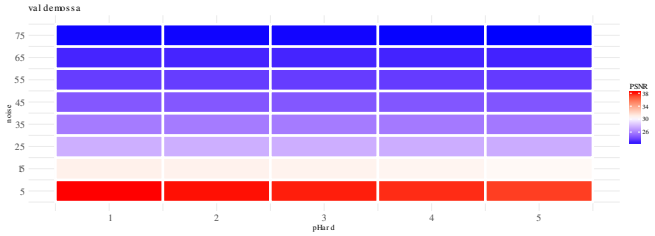
(b) `pupuce` image.



(c) `traffic` image.

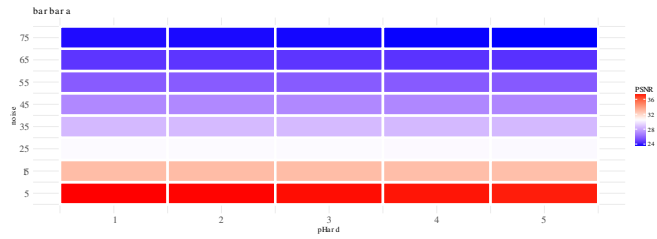


(d) `tree` image.

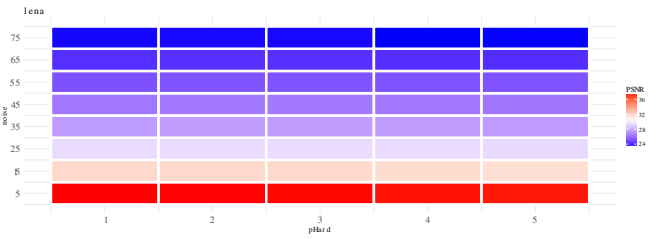


(e) `valdemossa` image.

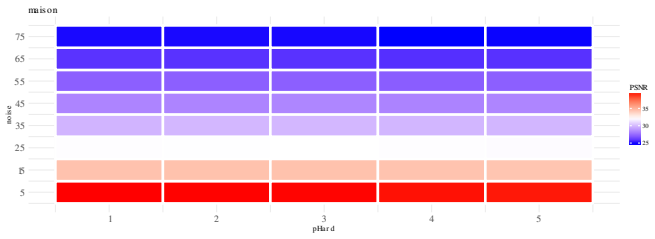
Figure 22: Influence of the acceleration factor for color images using *full* step.



(a) `barbara` image.



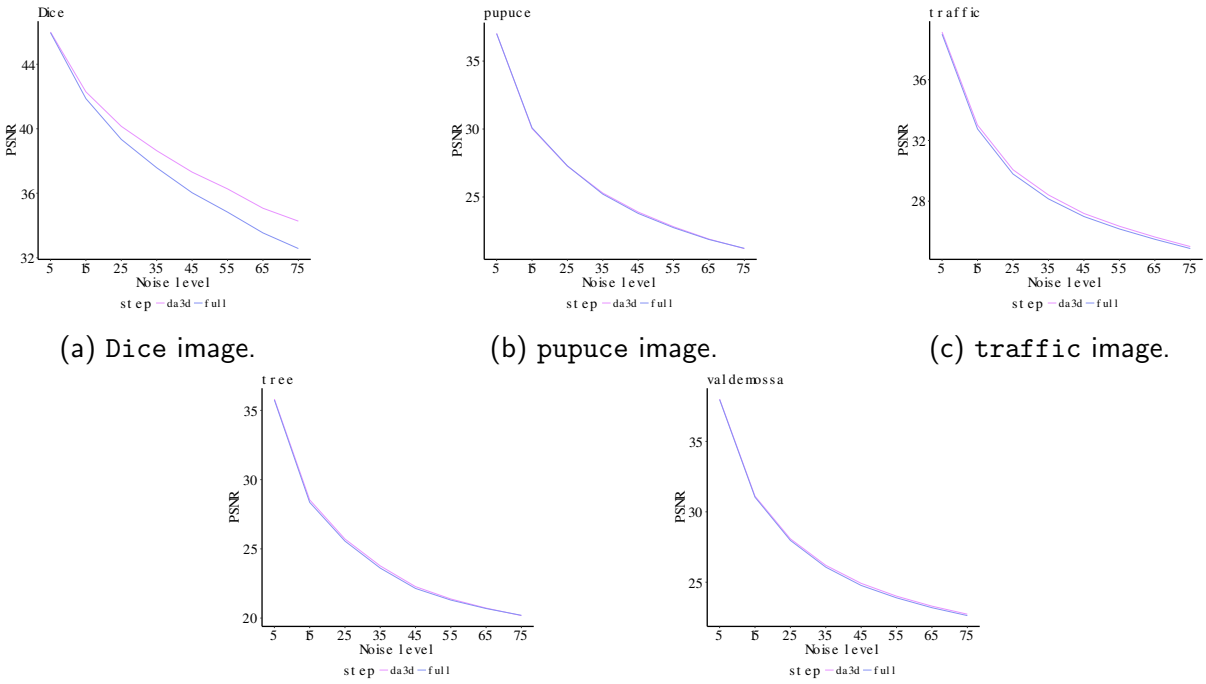
(b) `lena` image.



(c) `maison` image.

Figure 23: Influence of the acceleration factor for gray level images using *full* step.

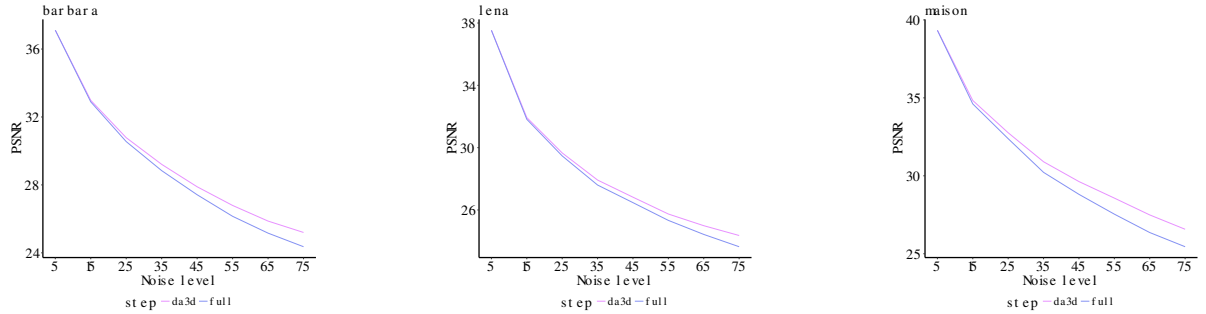
gave significant increases in PSNR. For HOSVD and for gray level images, a gain was obtained for strong noise (see Figure 25). The impact of DA3D was more important as the noise level increased, which is a good point. Hence, except for some particular cases, the one step DA3D procedure is only always interesting for gray level images. We did not see any pejorative effects of using DA3D on our test images set.



(a) Dice image. (b) pupuce image. (c) traffic image.

(d) tree image. (e) valdemossa image.

Figure 24: DA3D denoising for color images.



(a) barbara image. (b) lena image. (c) maison image.

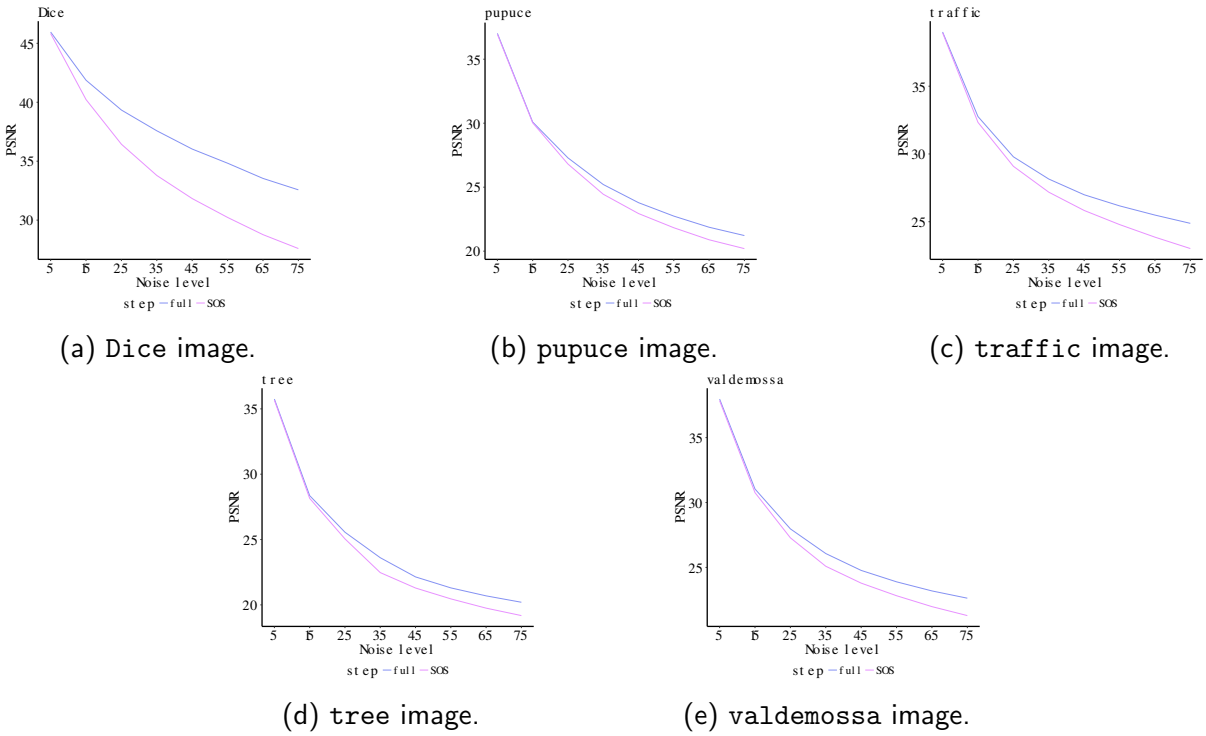
Figure 25: DA3D denoising for gray level images.

5.2 Iterative Denoising

In [17], the increase in PSNR for BM3D was limited. The main reason given by the authors was that the performance of BM3D was considered really high with respect to theoretical performance bounds. We implemented SOS Boosting for HOSVD to see if the same behavior was encountered with HOSVD. The implementation was done as an external script avoiding any quantization effects all along the process. We used three iterations for SOS denoising as in [17] for BM3D. Results are given in Figure 26 for color images and in Figure 27 for gray level images.

The situation was a little bit worse in fact. We tested a lot of parameters for SOS Boosting. During each test, we only noted that the PSNR was decreasing regularly with the formulas used in [17] that are a weighted mean between the noisy image and the SOS estimation. We were unable to find a situation were SOS Boosting permitted to increase the PSNR. We might recall the SOS iterations for HOSVD copied from the ones of BM3D,

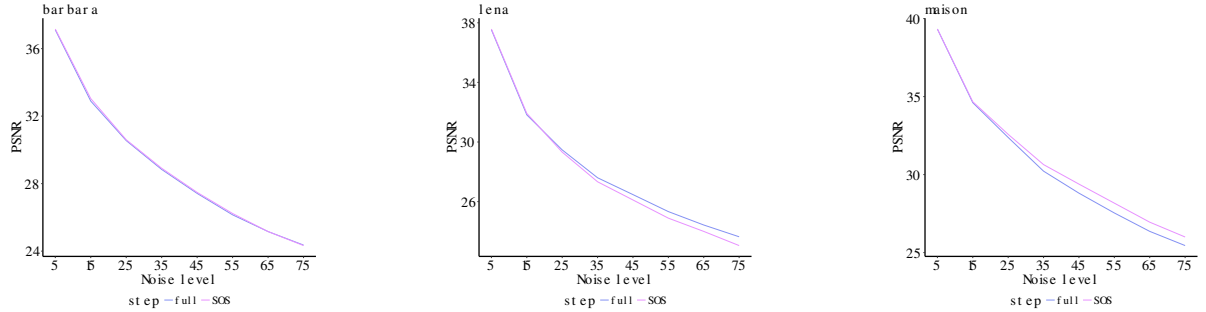
$$\hat{x}^{(k+1)} = \frac{1}{1-\tilde{\rho}} f\left((1-\tilde{\rho})y + \tilde{\rho}\hat{x}^{(k)}\right) - \frac{\tilde{\rho}}{1-\tilde{\rho}}\hat{x}^{(k)}\,.$$



(a) `Dice` image. (b) `pupuce` image. (c) `traffic` image.

(d) `tree` image. (e) `valdemossa` image.

Figure 26: SOS denoising for color images.



(a) `barbara` image. (b) `lena` image. (c) `maison` image.

Figure 27: SOS denoising for gray level images.

Hence starting with $\hat{x}^{(0)} = 0$, we notice that

$$\hat{x}^{(1)} = \frac{1}{1-\tilde{\rho}} f\left((1-\tilde{\rho})y\right) ,$$

and obviously if the denoising algorithm $f()$ is invariant to scale transforms then

$$\hat{x}^{(1)} = \frac{1}{1-\tilde{\rho}} f\left((1-\tilde{\rho})y\right) \simeq f(y) .$$

This is indeed the case for HOSVD. Thus, assuming equality between $\hat{x}^{(1)}$ and $f(y)$, we get

$$\hat{x}^{(2)} = \frac{1}{1-\tilde{\rho}} f\left((1-\tilde{\rho})y + f(y)\right) - \frac{\tilde{\rho}}{1-\tilde{\rho}} f(y) .$$

Hence the denoising algorithm $f()$ is applied on a regularized noisy image with the smooth image $f(y)$. Taking into account the fact that any denoising algorithm has a tendency to smooth the noisy

image, we get an image which is more smooth than $f(y)$. Then subtracting the image $f(y)$ leads to an excesively smooth image. As such any sub-image which was sufficiently denoised at the first step is smoothed again by the iteration of the denoising algorithm which inherently results in a decrease in the PSNR. This means that in our context, SOS Boosting is not able to increase the denoising performance because it is applied to the whole image and not to sub-images. This usage of SOS Boosting is beyond the content of the present study but might be required to produce an increase in PSNR.

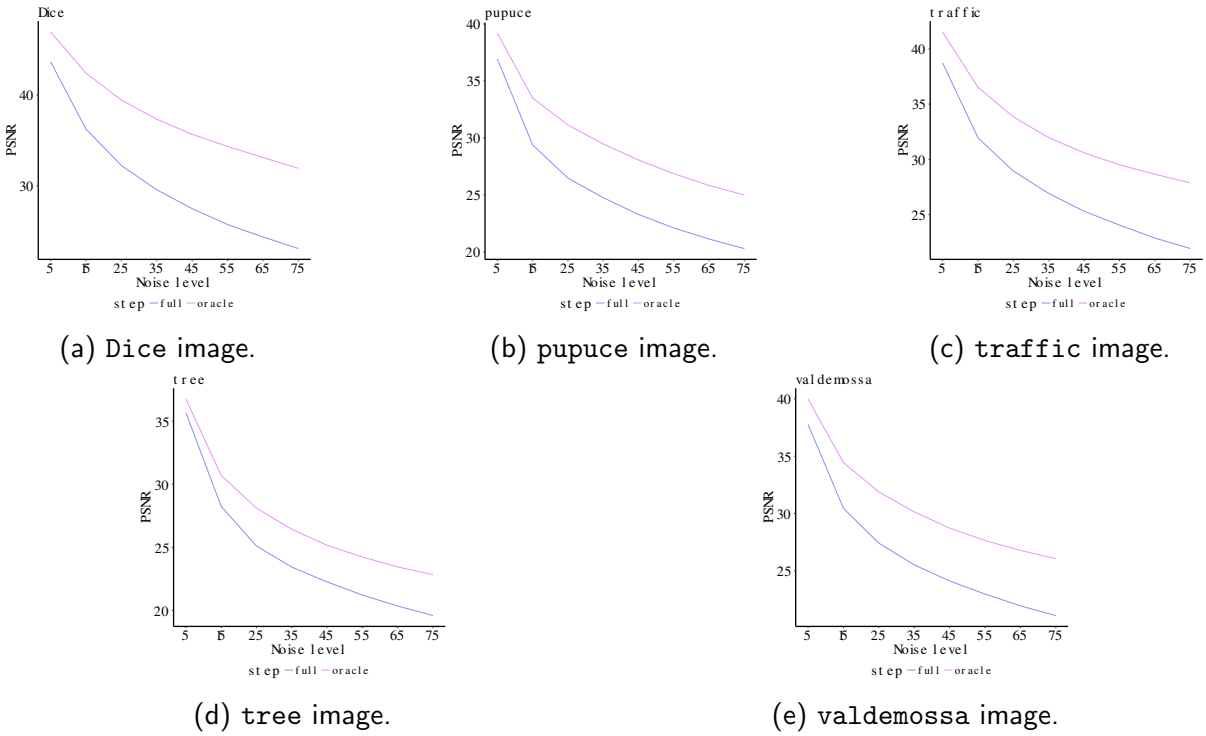
5.3 Oracle Denoising in HOSVD

Aside the Oracle denoising scenario based on minimization of MSE knowing the original clean image as in [11], the authors in [16] also propose an Oracle scenario based on SVD decomposition. The process is as follows. For each patch of size $p \times p$, the Singular Value Decomposition of the clean corresponding patch is provided by an Oracle. That is, for a patch P we know its decomposition $P = USV^T$. For the patch Q in the noisy image at the same position, we can compute its projection using the true singular vectors $S_Q = U^T QV$. Then, S_Q is processed with a hard threshold of $\sigma\sqrt{2\log p^2}$ and, after reconstruction, averaged over all overlapping patches. The authors in [16] showed that this results in a significant increase in the PSNR. They concluded that, whereas the singular values have been the object of most researches, singular vectors could be at least as important than singular values, perhaps even more important. However, this experimentation is coherent with what is known for the SVD for which the noise component corresponds to the small singular values, the ones that are put to zero with hard thresholding. So it is coherent that knowing the true singular vectors is by far most beneficial than knowing the true singular values. In the Oracle scenario, the true clean image might be used at several steps. We chose to restrict its use in a similar way as in a SVD decomposition. Indeed, the noisy image is still used for block-matching and thus this part is inherently imperfect. Then the tensors U_i of the HOSVD are computed with respect to the clean image. Then the core of the noisy tensor is computed with the clean U_i and the core is hard thresholded. A reconstruction is performed still using the clean U_i. The averaging is not modified at all and thus has an influence of the denoised image. Results are presented in Figure 28 for color images and in Figure 29 for gray level images.

As expected, the benefit of the Oracle scenario increased with an increase in the noise level. The effect of the Oracle scenario did not rely upon the nature of the images since both gray level images and color images led to large increases in PSNR. We can thus conclude that the singular vectors are indeed extremely important and that works should be done using them to increase the practical performance of patch-based denoising algorithms.

6 Conclusion

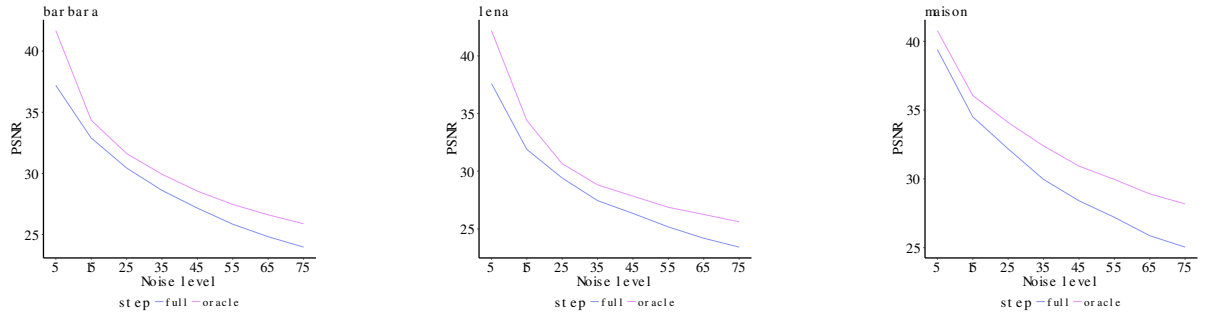
We provide an implementation of the HOSVD tensor decomposition for denoising [16] and a complete analysis of the parameters of the implementation. All experiments lead us to the conclusion that HOSVD can be useful in denoising but its actual performance is globally limited by a sub-optimal second pass of the Wiener-like filtering. We also notice its good performance on images containing regular textures. However, the method is hard to tune due to the sensitivity of the denoising performance to several of its parameters, which should be considered as an alert to the fact that those parameters should probably be fixed by a local analysis of the non-uniformity of the patches. Our Oracle scenario showed that HOSVD might be a good opponent to state-of-the-art algorithms when correctly tuned and with an efficient second pass. We were not able to provide really good increases in PSNR with the use of the DA3D algorithm [14] and with SOS Boosting [17], contrarily



(a) `Dice` image. (b) `pupuce` image. (c) `traffic` image.

(d) `tree` image. (e) `valdemossa` image.

Figure 28: Oracle denoising for color images.



(a) `barbara` image. (b) `lena` image. (c) `maison` image.

Figure 29: Oracle denoising for gray level images.

to what was expected from the experiments provided in the reference paper. For the latter, we did not observe any positive influence for HOSVD probably due to an excess of smoothing during the first passes of SOS Boosting. For DA3D, we did observe some interesting PSNR improvements.

To conclude, we must recommend to add a true Wiener filtering [13] to HOSVD as a second pass as its main advantage is its independence of the dimension of the images, that is, it readily extends to any dimension without significant modifications.

Acknowledgement

We thank both reviewers for constructive comments on a preliminary version of the manuscript which greatly improved the content and the presentation of this study. We thank Pablo Arias and Jean-Michel Morel for providing us specific code for avoiding quantization when saving noisy images.

Image Credits

 F. Feschet, CC-BY

 ,  ,  M. Colom, CC-BY

 A Buades, CC-BY-SA

 ,  ,  The USC-SIPI Image Database[3]

References

[1] A. Buades, B. Coll, and J.M. Morel, *A review of image denoising algorithms, with a new one*, Multiscale Modeling and Simulation, 4 (2006), pp. 490–530. http://dx.doi.org/10.1137/040616024.

[2] A. Buades, B. Coll, and J-M. Morel, *A non-local algorithm for image denoising*, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Washington, DC, USA, 2005, IEEE Computer Society, pp. 60–65. http://dx.doi.org/10.1109/CVPR.2005.38.

[3] A Cichocki, C Mandic, A. H Phan, C. Caiafa, G. Zhou, Q. Zhao, and L De Lathauwer, *Tensor decompositions for signal processing applications. from two-way to multiway component analysis*, IEEE Signal Processing Magazine, 32 (2015), pp. 145–163. https://doi.org/10.1109/MSP.2013.2297439.

[4] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, *Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering*, IEEE Transactions on Image Processing, 16 (2007), pp. 2080–2095. http://dx.doi.org/10.1109/TIP.2007.901238.

[5] D.L. Donoho and J.M. Johnstone, *Ideal spatial adaptation by wavelet shrinkage*, Biometrika, 81 (1994), pp. 425–455. https://doi.org/10.1093/biomet/81.3.425.

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*, Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

[7] C. Knaus and M. Zwicker, *Dual-domain image denoising*, in Proceedings of the IEEE International Conference on Image Processing (ICIP), IEEE Computer Society, 2013, pp. 440–444. https://doi.org/10.1109/ICIP.2013.6738091.

[8] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1253–1278. http://dx.doi.org/10.1137/S0895479896305696.

[9] M. Lebrun, *An Analysis and Implementation of the BM3D Image Denoising Method*, Image Processing On Line, 2 (2012), pp. 175–213. https://doi.org/10.5201/ipol.2012.l-bm3d.

[10] M. Lebrun, A. Buades, and J-M. Morel, *Implementation of the "Non-Local Bayes" (NL-Bayes) Image Denoising Algorithm*, Image Processing On Line, 3 (2013), pp. 1–42. https://doi.org/10.5201/ipol.2013.16.

[3] Allan G Weber, The USC-SIPI image database version 5, USC-SIPI Report, 315 (1997), pp. 124.

[11] P. Milanfar, *Symmetrizing smoothing filters*, SIAM Journal on Imaging Sciences, 6 (2013), pp. 263–284. https://doi.org/10.1137/120875843.

[12] D. Muti and S. Bourennane, *Multidimensional filtering based on a tensor approach*, Signal Processing, 30 (2005), pp. 1172 – 1204. https://doi.org/10.1016/j.sigpro.2004.11.029.

[13] D. Muti, S. Bourennane, and J. Marot, *Lower-rank tensor approximation and multiway filtering*, SIAM Journal on Matrix Analysis and Applications, 85 (2008), pp. 2338 – 2353. https://doi.org/10.1137/060653263.

[14] N. Pierazzo and G. Facciolo, *Data Adaptive Dual Domain Denoising: a Method to Boost State of the Art Denoising Algorithms*, Image Processing On Line, 7 (2017), pp. 93–114. https://doi.org/10.5201/ipol.2017.203.

[15] S. Preethi and D. Narmadha, *A survey on image denoising techniques*, International Journal of Computer Applications, 58 (2012), pp. 27–30. https://doi.org/10.5120/9288-3488.

[16] A. Rajwade, A. Rangarajan, and A. Manduchiee, *Image denoising using the higher order singular value decomposition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 35 (2013), pp. 849–862. https://doi.org/10.1109/TPAMI.2012.140.

[17] Y. Romano and M. Elad, *Boosting of image denoising algorithms*, SIAM Journal on Imaging Sciences, 8 (2015), pp. 1187–1219. https://doi.org/10.1137/140990978.

[18] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, in Proceedings of the Sixth International Conference on Computer Vision (ICCV), Washington, DC, USA, 1998, IEEE Computer Society, pp. 839–. https://doi.org/10.1109/ICCV.1998.710815.

[19] L.R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311. https://doi.org/10.1007/BF02289464.

[20] G. Yu and G. Sapiro, *DCT image denoising: a simple and effective image denoising algorithm*, Image Processing On Line, (2011). https://doi.org/10.5201/ipol.2011.ys-dct.