Published in Image Processing On Line on 2018–12–23.
Submitted on 2018–12–10, accepted on 2018–12–17.
ISSN 2105–1232 © 2018 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.2018.242

EPLL: An Image Denoising Method using a Gaussian Mixture Model Learned on a Large Set of Patches

Samuel Hurault, Thibaud Ehret, Pablo Arias

CMLA, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France
thibaud.ehret@cmla.ens-cachan.fr

Communicated by Pablo Arias *Demo edited by* Thibaud Ehret

Abstract

The *Expected Patch Log-Likelihood* method, introduced by Zoran and Weiss, allows for whole image restoration using a patch-based prior (in the likelihood sense) for which a maximum a-posteriori (MAP) estimate can be calculated. The prior used is a Gaussian mixture model whose parameters are learned from a dataset of natural images. This article presents a detailed implementation of the algorithm in the context of denoising of images contaminated with white additive Gaussian noise. In addition, two possible extensions of the algorithm to handle color images are compared.

Source Code

The reviewed source code and documentation for this algorithm are available at the web page of this article[1]. Compilation and usage instructions are included in the README.txt file of the archive.

Keywords: denoising; multiscale; patch based

1 Introduction

Image denoising aims to restore the latent clean image x from its noise-corrupted version $y = x + v$, where v is commonly assumed to be additive white Gaussian noise of standard deviation σ. In this work we will reproduce and optimize the EPLL denoising algorithm introduced by [19] which is a Bayesian patch-based method [11]. Image priors are a popular tool for image restoration tasks. However, the high dimension of images makes learning or optimization with such priors very hard. This is why state-of-the art methods learn priors over small image patches ([10, 9, 18]).

A part of these denoising methods have attempted to take advantage of the internal self-similar structures present in most images. Early approaches like non-local means (NL-means) [1] and

[1]https://doi.org/10.5201/ipol.2018.242

UINTA [2] proposed to look for similar patches of an image and average them. Patch-based denoising methods [10, 9, 18] have since then developed into attempting to model the patch space of an image.

Instead of using similar patches of the same image, EPLL uses a prior (namely a Gaussian mixture model) learned from a very large set of patches taken from several images. It can thus be considered an external denoising method (the target image is denoised using other images), or what [11] calls a “shotgun method”. EPLL can be seen as an external version of the piecewise linear estimator (PLE) method [18]: PLE learns a GMM specific to each noisy image, whereas EPLL uses a fixed GMM learned once from a collection of clean patches.

Thus, the first step in EPLL will be to extract patches from a dataset of clean images and to learn a GMM prior from them by maximizing the likelihood. Once a prior is set, given a noisy image y, a first approach to perform denoising could be to decompose it into overlapping patches, to denoise every patch separately and finally to aggregate the results by simple averaging. This aggregation of overlapping patch estimates is common in patch-based algorithms [10, 9]; it improves the estimation as it averages for any pixel a set of different estimates.

However, applying the prior only on patches without any control on the whole image is not optimal. Indeed, averaging the values obtained for each pixel from the patches that contain it creates new patches in the constructed image, which might be unlikely under our prior. The EPLL (Expected Patch Log Likelihood) method by Zoran and Weiss [19] addresses this very problem. The aim of the method is simple: suppose we take a random patch from our reconstructed image, we wish this patch to be likely under our prior. In other words, we wish to find a reconstructed image in which every patch is likely under our prior while keeping the reconstructed image still close to the corrupted one. This variational approach is still popular as shown by recent denoising papers [5, 8]. In [17] a general approach to aggregate local patch models into a global image model is proposed.

A flaw in patch-based modeling is the enforced locality of the model. Even if EPLL endeavors to work globally on the image, when working with small sized patches we neglect the long-range interactions present in the image. Based on this observation, we shall invoke here the general multi-scale framework of [14], which can be applied to any denoising algorithm. A different multiscale version of EPLL was proposed by Papyan and Elad [16] by applying the EPLL prior of different image scales simultaneously. This results in a unique energy, and has the advantage that it can be applied as a prior in other restoration problems.

The contributions of this article are to provide an implementation of the EPLL method, to optimize the choice of its parameters, to adapt it to color image denoising and to improve it with a general multi-scale approach.

2 Description of the EPLL Method

2.1 Learning: Gaussian Mixture Prior and Expectation Maximization

The patch prior assumed by EPLL is a Gaussian mixture model (GMM). The probability density of a patch $\mathbf{x}$, represented by a vector of $\mathbb{R}^d$, is assumed to satisfy

$$p(\mathbf{x}) = \sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k). \tag{1}$$

Here, $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$ is a d-dimensional Gaussian density (a patch in the image is $\sqrt{d} \times \sqrt{d}$) of mean $\boldsymbol{\mu}_k$ ($d \times 1$) and covariance matrix Σ_k ($d \times d$). The coefficients w_k represent the mixing weights for each mixture component. They add to one i.e. $\sum_k w_k = 1$.

Learning is performed using the Expectation Maximization (EM) algorithm. It is computed over N natural patches $\mathbf{x}_i$ with their DC component removed (the DC component corresponds to the mean intensity value of the patch). The DC component is the direction of dominant variation in the patch space, and it can be estimated from the noisy patch with a residual noise of standard deviation $\sigma/\sqrt{d}$. By removing it, the GMM focuses on other patch structures.

We review here the classic EM algorithm as explained in more detail in [3]. The goal is to maximize the likelihood function $\prod_{i=1}^{N} p(\mathbf{x}_i)$ with respect to the parameters (comprising the means, covariances of the components and the mixing coefficients). We introduce the posterior probability γ_{ik}, which represents the responsibility that takes the component k for generating the patch $\mathbf{x}_i$,

$$\gamma_{ik} = \frac{w_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} w_j \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_j, \Sigma_j)}. \tag{2}$$

The EM algorithm alternates between an expectation step and a maximization step:

1. Initialize the responsibilities γ_{ik} randomly.

2. **M step:** Estimate the parameters using the current responsibilities

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k}\sum_{i=1}^{N}\gamma_{ik}\mathbf{x}_i, \quad \Sigma_k^{new} = \frac{1}{N_k}\sum_{i=1}^{N}\gamma_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k^{new})(\mathbf{x}_i - \boldsymbol{\mu}_k^{new})^T, \quad w_k^{new} = \frac{N_k}{N}$$

   where $N_k = \sum_{i=1}^{N} \gamma_{ik}$.

3. **E step:** Re-evaluate the responsibilities using formula (2) with the current parameter values.

4. Evaluate the log-likelihood of the image x

$$\mathcal{L}(x|\boldsymbol{\mu}, \Sigma, w) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} w_k N(\mathbf{x}_i | \boldsymbol{\mu}_k, \Sigma_k)\right),$$

   and check for convergence of the log-likelihood. If the convergence criterion is not satisfied return to step 2.

It was observed in [19] that the means of the components $\boldsymbol{\mu_k}$ for the GMM learned from datasets of patches were almost $\mathbf{0}$. In the following we will assume that $\boldsymbol{\mu_k} = \mathbf{0}$.

2.2 Denoising of Grayscale Images

In all this section we consider a grayscale image x defined over a rectangular grid Ω of size $n \times m$. The image x is considered in a vectorized form so that x_k corresponds to the k^{th} pixel, and P_i is a matrix which extracts the i^{th} patch from the image so that $\mathbf{x}_i = P_i x$.

2.2.1 Energy to Minimize

Computing the maximum a posteriori (MAP) to estimate the clean patch $\mathbf{x}_i$ given a noisy patch $\mathbf{y}_i$ and a local prior p leads to the following energy to minimize for each image patch

$$E_i(\mathbf{x}_i; \mathbf{y}_i) = \frac{\|\mathbf{x}_i - \mathbf{y}_i\|^2}{2\sigma^2} - \log p(\mathbf{x}_i).$$

However, as explained in the introduction, we want to work globally on the image in order to remain close to the noisy image. The basic idea behind the method is to maximize the *expected patch log-likelihood* (EPLL) computed over the whole image while still being close to the corrupted image in a way which is dependent on the corruption model. Given an image x, the EPLL under the patch prior p is defined as

$$\mathrm{EPLL}_p(x) = \sum_{i\in\Omega_P} \log p(P_i x) = \sum_{i\in\Omega_P} \log p(\mathbf{x}_i), \tag{3}$$

where $\Omega_P \subset \Omega$ is a set of patches which covers the image domain Ω.

Assuming a patch location in the image is chosen uniformly at random, EPLL can be interpreted as the expected log likelihood of a patch in the image (up to a multiplication by $1/N$). It serves as a global prior for an image (although it doesn't define a probability in the space of images). Given a noisy image y, the energy to minimize in order to find the denoised image using the patch prior p is

$$E(x;y) = \sum_{k\in\Omega} N_k \frac{\|x_k - y_k\|^2}{2\sigma^2} - \mathrm{EPLL}_p(x), \tag{4}$$

where Ω is the image domain and N_k represents the number of overlapping patches in which the pixel k appears (pixels close to the border of the image have smaller N_k).

2.2.2 Optimization

Direct optimization of the energy in (4) may be very hard, depending on the prior used. The authors of EPLL propose to use an optimization method called Half-Quadratic Splitting [6]. We introduce as auxiliary variables a set of patches $\{\mathbf{z}_i\}_1^N$, one for each patch $P_i x$ in the image, yielding the following energy to minimize

$$E_\beta(x, \{\mathbf{z}^i\}; y) = \sum_{k\in\Omega:w} N_k \frac{\|x_k - y_k\|^2}{2\sigma^2} + \sum_{i\in\Omega_P} \beta \frac{\|P_i x - \mathbf{z}_i\|^2}{2\sigma^2} - \log p(\mathbf{z}_i). \tag{5}$$

Note that as $\beta \to +\infty$ we restrict the patches $P_i x$ to be equal to the auxiliary variables $\{\mathbf{z}_i\}$ and the solutions of (5) can be considered as approximations of solutions for (4). For a fixed value of β, optimizing (5) can be done iteratively, first solving for x while keeping $\{\mathbf{z}_i\}$ constant, then solving for $\{\mathbf{z}_i\}$ with the newly found x kept constant. The algorithm initializes x with the noisy image y and iterates the process about six times. After each iteration the value of β is increased according to a predefined schedule.

Step 1 : Solving for $\{\mathbf{z}_i\}$ given x : Patch MAP Estimation

The problem of minimizing (5) with respect to $\{\mathbf{z}_i\}$ is separable, meaning that it is equivalent to minimizing for each patch $\mathbf{z}_j$ independently of the rest. Each of these problems can be seen as a MAP for an individual patch, with prior p and noise level $\frac{\sigma}{\sqrt{\beta}}$. Given a noisy patch $P_i x$, the MAP problem with a GMM prior is non-convex. Computing a local maximum can be too costly, considering that it is done for each image patch. This is addressed in [19] with the following approximate MAP estimation procedure:

1. Given a noisy patch $P_i x$, remove its DC component (as the GMM was learned over zero-mean patches) and compute the conditional mixing weights $\gamma_{ik} = P(k|P_i x)$.

2. Choose the component with the highest conditional mixing weight $k_i^* = \mathrm{argmax}_k \gamma_{ik}$.

3. Estimate the MAP by a Wiener filter solution for the k_i^*-th component

$$\mathbf{z}_i = \left(\Sigma_{k_i^*} + \frac{\sigma^2}{\beta} I\right)^{-1} \Sigma_{k_i^*} P_i x. \tag{6}$$

   This follows from differentiating the reduced quadratic function $E_\beta(x, \{\mathbf{z}_i\}; y)$ (5) with respect to $\mathbf{z}_i$ and equating to zero.

4. Add the previously removed DC component to each estimated patch.

Step 2 : Solving for x given $\{\mathbf{z}_i\}$

If we fix $\{\mathbf{z}_i\}$, the energy (5) is quadratic in x thus allows a closed form solution. Taking the derivative of (5) w.r.t to the vector x, setting to 0 and solving the resulting equation yields

$$x = \left((1+\beta) \sum_{i \in \Omega_P} P_i^T P_i\right)^{-1} \left(\left(\sum_{i \in \Omega_P} P_i^T P_i\right) y + \beta \sum_{i \in \Omega_P} P_i^T \mathbf{z}_i\right) = \frac{y + (\sum_{i \in \Omega_P} P_i^T P_i)^{-1} \sum_i P_i^T \mathbf{z}_i}{1+\beta}. \tag{7}$$

$P_i^T \mathbf{z}_i$ represents the adjoint transformation of P_i. It creates an image where the values in the patch $\mathbf{z}_i$ are put in their appropriate position and the rest of the image is set to zero. The matrix $\sum_{i \in \Omega_P} P_i^T P_i$ is diagonal of size $nm \times nm$. Each element in the diagonal is associated to a pixel and equal to the number of patches that contain it. Then, we set z to be the image after averaging the overlapping patches $\mathbf{z}_i$. It is given by

$$z = \left(\sum_{i \in \Omega_P} P_i^T P_i\right)^{-1} \sum_{i \in \Omega_P} P_i^T \mathbf{z}_i. \tag{8}$$

Finally we have

$$x = \frac{y + \beta z}{1+\beta}. \tag{9}$$

This means that the solution for x at each optimization step is just a weighted average between the noisy image y and the image z resulting from the aggregation of the auxiliary patches.

3 Implementation

3.1 Algorithms

We present the pseudo-code for the EPLL denoising framework in Algorithm 1, the MAP approximation in Algorithm 2 and the computation of the log probability in Algorithm 3.

In order to be faster, we pre-compute and store the eigen-decomposition of each covariance matrix of the GMM. These are then used in Algorithm 3 and the Wiener filtering in Algorithm 2. We denote by $v_1^k, \dots, v_d^k$ the eigenvalues of Σ_k and by Q_k the $d \times d$ eigenvector matrix,

For the Wiener filtering (6) we need to invert a $d \times d$ matrix, an operation which is $\mathcal{O}(d^3)$. This inversion can be done efficiently if we store the eigen-decompositions of the covariance matrices of the GMM. Indeed, we have that

$$\mathbf{z}_i = \left(\Sigma_{k_i^*} + \frac{\sigma^2}{\beta} I\right)^{-1} \Sigma_{k_i^*} P_i x = Q_{k_i^*} S_{k_i^*} Q_{k_i^*}^T P_i x,$$

Algorithm 1: EPLLhalfQuadraticSplit

```
input  : $y$                                                        Noisy image
         $\sigma$                                                   Noise standard deviation
         betas                                                      List of beta values
         $T$                                                        Number of times the optimization is done
         step                                                       Patches sampling step
         GMM                                                        Gaussian mixture model used for denoising
output: $\hat{x}$                                                   Estimate of clean image

$\hat{x} \leftarrow y$                                              Initialization

foreach $\beta$ in betas do
    for $t = 1$ to $T$ do
        $z \leftarrow$ aprxMAPGMM $(\hat{x}, \sigma/\sqrt{\beta}$, step, GMM)   Calculate aggregate image of MAP estimates
        $\hat{x} \leftarrow \frac{y + \beta z}{1 + \beta}$          Calculate the current estimate for the clean image
    end
end
```

Algorithm 2: aprxMAPGMM

```
input  : $x$                                                        The noisy image
         $s$                                                        Noise standard deviation for the current estimate $= \frac{\sigma}{\sqrt{\beta}}$
         step                                                       Patch-sampling step
         GMM                                                        Gaussian mixture model used for denoising
output: $z$                                                         Output image, aggregated of patch MAPs

Extract all patches $\mathbf{x}_i$ in $x$ with a step step
for each patch $\mathbf{x}_i$ do
    $\mathbf{x}'_i \leftarrow \mathbf{x}_i - \text{mean}(\mathbf{x}_i)$        Remove DC component from each patch
for each $(\Sigma_j, w_j)$ in GMM do                                $\Sigma_j$ is the cov. and $w_j$ the weight of the $j^{th}$ component
    $W[j, \cdot] \leftarrow \text{logGaussPDF}(X', \Sigma_j + s^2 I, w_j)$   $X'$ is a matrix containing the patches $\mathbf{x}'_i$ as rows
for each patch $\mathbf{x}'_i$ do
    $k_i^* \leftarrow \text{argmax}(W[\cdot, i])$
    $\mathbf{z}_i = (\Sigma_{k_i^*} + s^2 I)^{-1} \Sigma_{k_i^*} \mathbf{x}'_i + \text{mean}(\mathbf{x}_i)$

Aggregate patches $\hat{\mathbf{z}}_i$ into $z$
```

Algorithm 3: logGaussPDF

```
input  : $X$                                                        Matrix with patches as rows
         $\Sigma$                                                   The covariance matrix of the Gaussian model
         $w$                                                        The weight of the Gaussian model in the GMM
output: $W$                                                         The log probability associated to each patch in Y

$W \leftarrow \log(w) - \frac{d \log(2\pi) + \log\det(\Sigma) + X\Sigma^{-1}X^T}{2}$   d is the dimension of a patch
```

where $S_{k_i^*}$ is a diagonal matrix with the Wiener shrinkage coefficients

$$S_{k_i^*} = \operatorname{diag}\left(\frac{v_j^{k_i^*}}{v_j^{k_i^*} + \frac{\sigma^2}{\beta}} \right)_{1 \leq j \leq d}.$$

For Algorithm 3, we have to calculate for a patch $\mathbf{x}_i = P_i x$ the value of terms $\mathbf{x}_i^T \left(\Sigma_k + \sigma^2/\beta I\right)^{-1} \mathbf{x}_i$, where Σ_k is a covariance matrix of the mixture. Using the eigen-decomposition of Σ_k and the diagonal matrix

$$R_k = \operatorname{diag}\left(v_j^k + \frac{\sigma^2}{\beta} \right)_{1 \leq j \leq d},$$

we have that

$$\mathbf{x}_i^T \left(\Sigma_k + \sigma^2/\beta I\right)^{-1} \mathbf{x}_i = \|R_k^{-1/2} Q_k^T \mathbf{x}_i\|^2. \tag{10}$$

3.2 Learning Results

We used the database of images [15] composed of TIF images of size 768×576. In order to consider them noiseless, we zoomed out these images by a factor of 2. To do so, the images were filtered by a Gaussian kernel of standard deviation 1.4 and every pixel out of 4 was selected. Among the 419 images of size 384×288 , we selected uniformly about 70% images to perform learning, the remaining 30% becoming a test dataset. We transformed them to grayscale images by extracting the Y channel from YUV color space

$$Y = 0.30R + 0.59G + 0.11B,$$

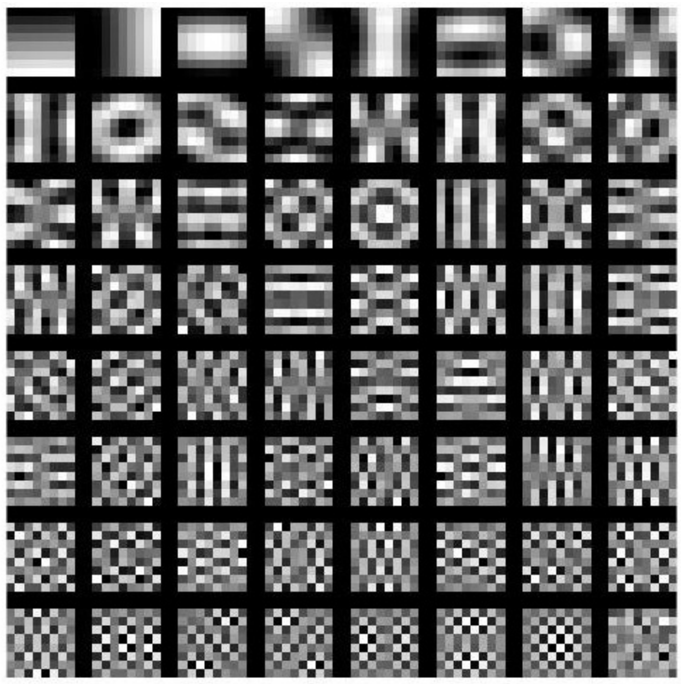
where R,G,B are the channels of the color image. The learning step was realized independently from the denoising step. The GMM was learned once, and then used for the different denoising tasks.

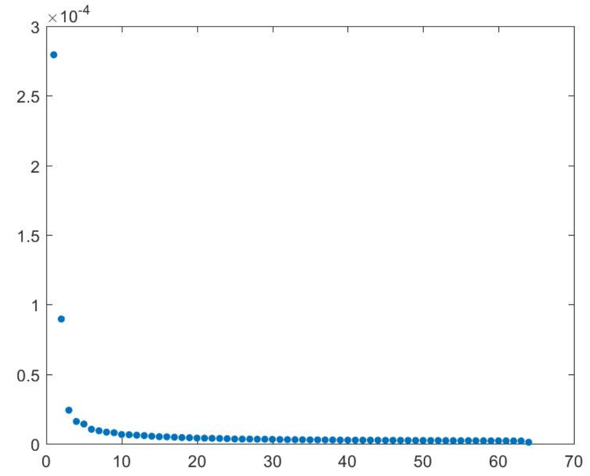
A patch x is represented by a vector $x \in \mathbb{R}^d$. From this database, we extracted around 30×10^6 patches and removed their DC component, then we randomly selected 2×10^6 patches to perform EM. Training with the above training set took around 30h with a MATLAB code[2] on an Intel Xeon E5-2650 v2 at 2.60GHz CPU.

In Figures 1, 2, 3 and 4 we represent 4 of the 200 Gaussian components learned. They are ordered decreasingly with respect to the mixing weight coefficients π_k. The means of the learned components are very close to zero and therefore modified to be zero. The eigenvectors (represented as 8×8 patches) are sorted by eigenvalue from largest to smallest. Given a mixture component, to understand what kind of patch it represents, we compute and display 8 patches sampled from the Gaussian model. They are obtained with a weighted average of the eigenvectors, the weights following a normal distribution of mean 0 and variance equal to the corresponding eigenvalue. We note that the components have very rich structures, while some resemble the DCT bases (1), some model edges (2,3) or textures (4).

The first Gaussian has its mixing weight π_k bigger than the others ($\pi_1 = 0.10$, $\pi_2 = 0.06$, $\pi_2 = 0.02$). It is represented in Figure 1, this Gaussian has a covariance matrix with very small eigenvalues. This means that it has a very small variance in every direction, and it can be seen as a peak at the zero patch. Thus, it models the flat patches of the dataset. Moreover, we can notice that among its very low eigenvalues, two of them, and particularly the first one, are significant compared to the others. They give importance to the two first eigenvectors represented, which represent horizontal and vertical smooth gradings. That may be explained by the fact that a flat patch in the nature is never completely flat but always shows some shading.

[2]http://www.mathworks.com/matlabcentral/fileexchange/26184-em-algorithm-for-gaussian-mixture-model


(a) eigenvectors



(b) corresponding eigenvalues

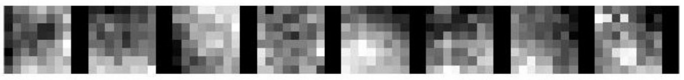
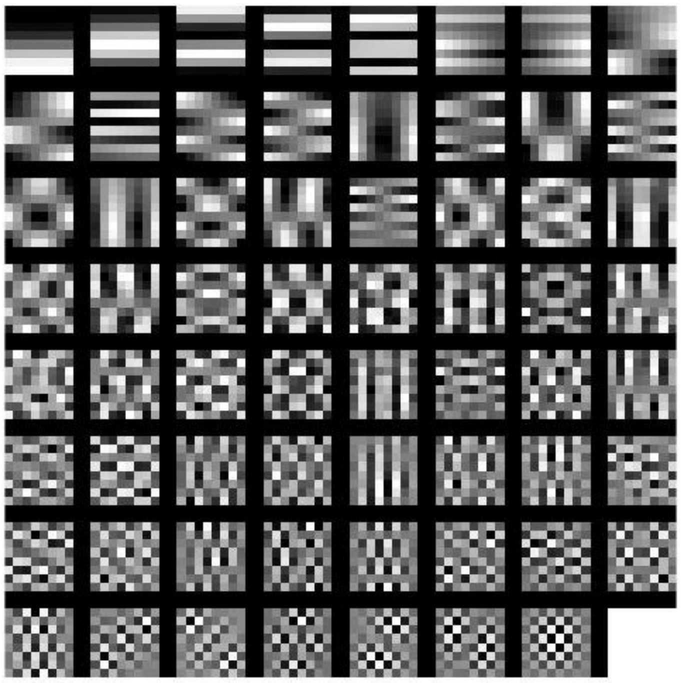
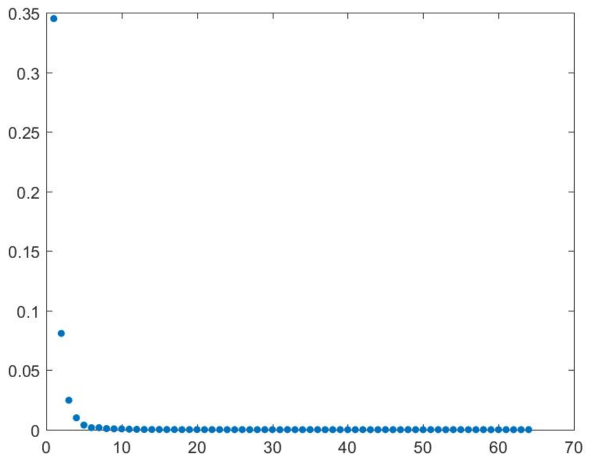

(c) 8 simulated patches

Figure 1: Component 1 of the Gaussian Mixture Model. The very low eigenvalues indicate that it represents flat patches. The eight simulated patches are scaled from black to white for a better understanding of their structure, but would look uniformly black otherwise.


(a) eigenvectors



(b) corresponding eigenvalues

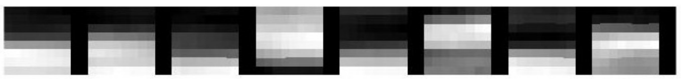
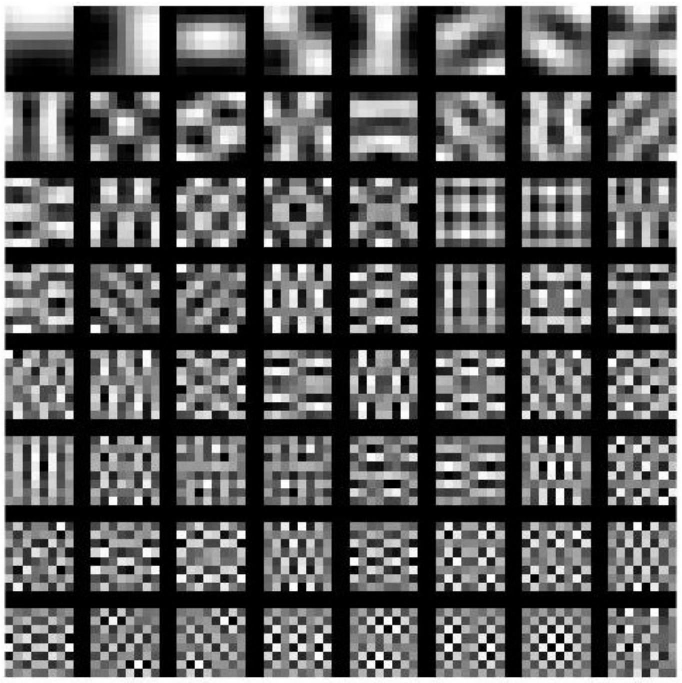
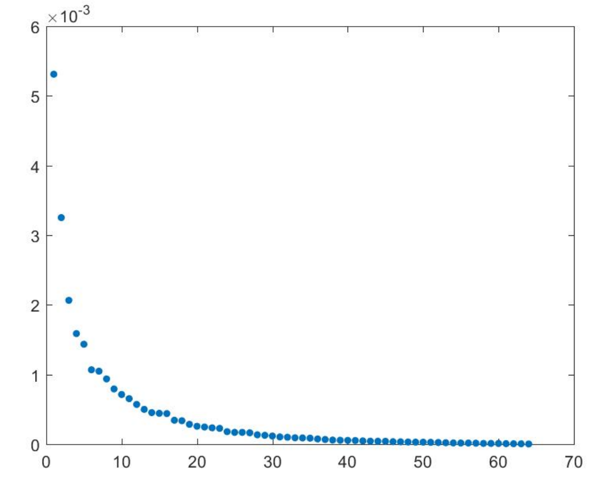

(c) 8 simulated patches

Figure 2: Component 11 of the Gaussian Mixture Model. The sharp decay of the eigenvalues gives importance only to the top eigenvectors in the simulated patches: it simulates horizontal edges.


(a) eigenvectors


(b) corresponding eigenvalues

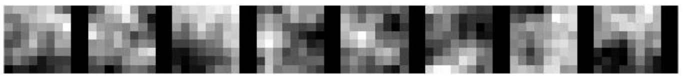
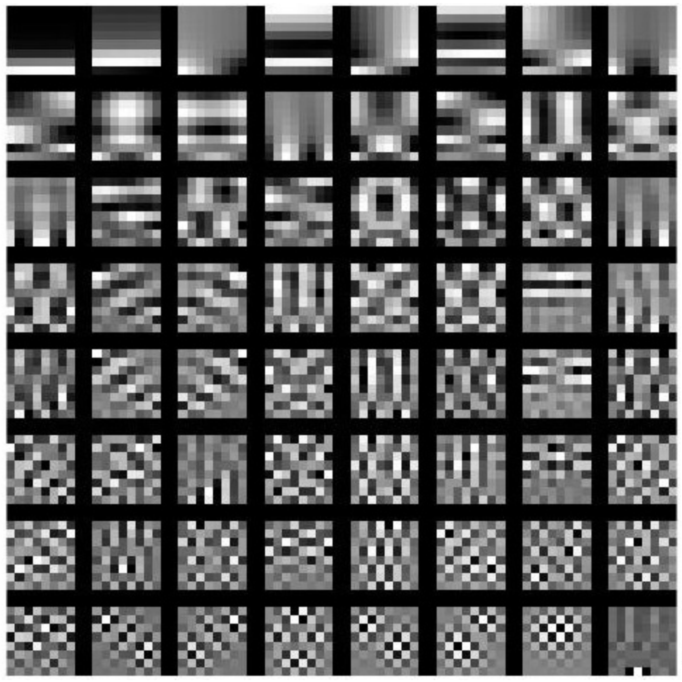
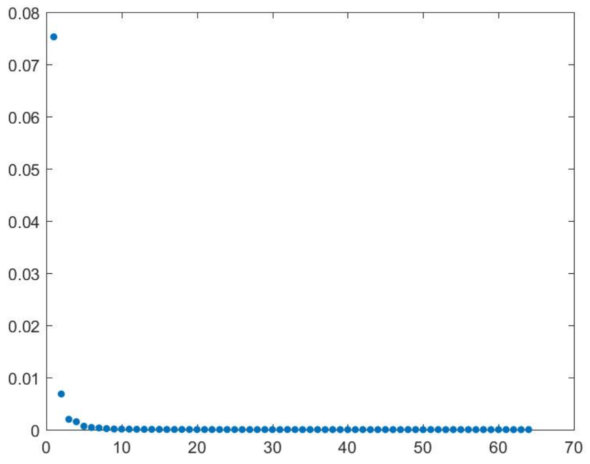

(c) 8 simulated patches

Figure 3: Component 13 of the Gaussian Mixture Model. The decay of the eigenvalues is slower and gives importance to many eigenvectors in the simulated patches: it therefore simulates a texture patch.


(a) eigenvectors


(b) corresponding eigenvalues

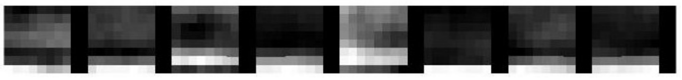

(c) 8 simulated patches

Figure 4: Component 14 of the Gaussian Mixture Model. The sharp decay of the eigenvalues gives importance only to the top eigenvectors in the simulated patches: it simulates patches with an edge at the bottom.

4 Influence of the Parameters on the Performance of the Algorithm

We applied the previously described algorithm to noiseless images to which a simulated white Gaussian noise had been added. For every study and for a given combination of parameters, 10 images will be chosen uniformly in the dataset of test images. To evaluate quantitatively the denoising results we use the PSNR.

4.1 Learning

In the following we evaluate the influence and optimize the various parameters which take part in the learning of the GMM. The parameters are N, the size of the dataset of patches used to perform EM and K the number of components in the GMM.

4.1.1 Number of Gaussian Components K

A visual inspection of the Gaussian components shows that many of them are similar to another one. To verify if these were redundant, we tested reducing the number of components in the Gaussian Mixture. Like in [19], we fixed $N = 2 \cdot 10^6$ and modified K. We chose 10 images of the test dataset of images and found the average PSNRs presented in Table 1.

σ	K=200	K=100	K=50
5	40.443	40.379	40.298
10	36.575	36.523	36.447
20	32.915	32.893	32.840
30	30.814	30.765	30.743
40	29.385	29.185	29.178
50	28.307	27.854	27.861

Table 1: Influence of K

We observe that the denoising performance decreases when K is reduced. Yet, this decay is negligible for small σ. For large noise values, the drop in performance is larger. This makes sense: the higher the noise, the more the estimation shrinks the patches toward the prior model.

On the other hand, the number of components has a strong influence on the running time. We would need to minimize this number without influencing the denoising performances. According to these experiments, we will choose the following values for K:

$5 \leq \sigma < 30$	$30 \leq \sigma$
$K = 100$	$K = 200$

4.1.2 Size of the Dataset of Patches N

We now keep $K = 200$ and modify N to obtain Table 2.

$N = 2 \cdot 10^6$ yields a better performance, especially for high values of the noise. As the choice of N does not influence the execution time of the algorithm, we decided to keep $N = 2 \cdot 10^6$.

4.2 Denoising

We now want to evaluate and optimize the various parameters which take part in the denoising process. The parameters are: s, the step used to extract and aggregate the overlapping patches; and the schedule of β values in use to minimize the energy.

σ	$N = 5 \cdot 10^5$	$N = 1 \cdot 10^6$	$N = 2 \cdot 10^6$
5	40.277	40.281	40.286
10	36.303	36.307	36.306
20	32.480	32.482	32.476
30	30.237	30.238	30.321
40	28.570	28.567	28.843
50	27.073	27.066	27.692

Table 2: Influence of N

4.2.1 Extracting Step s

Originally set to be 1, we try in Table 3 to change the step s used to extract and aggregate patches.

σ	s=1	s=2	s=3	s=4
5	40.505	40.440	40.360	40.230
10	36.662	36.585	36.469	36.311
20	32.974	32.889	32.747	32.553
30	30.890	30.755	30.603	30.427
40	29.435	29.173	29.044	28.879
50	28.395	27.842	27.736	27.601

Table 3: Influence of the step of patch extraction

For high values of the noise, details are easily lost, a pixel of the noisy image requires thus more estimations to reduce the variance from the right estimation. Between a step of 1 and a step of 2, the number of estimations for each pixel (except on edges) goes from 64 to 16. This explains the increasing difference in PSNR with the noise standard deviation. We therefore keep a step of 1.

4.2.2 Choice of β Values

Zoran and Weiss [19] proposed the values of β to be $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5) = (1, 4, 8, 16, 32, 64)$. In this section we verify that these betas are indeed the best. For that we tried a range of different βs centered around the value proposed by [19] and we kept the value which gave the best final PSNR. The βs were changed one at a time. We keep the growth which gives the best final PSNR. In Table 4, we give for different values of σ, the optimized list $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5)$ and the gain in PSNR with respect to the schedule proposed by [19]. This optimization hardly changes the denoising efficiency except for high values of the noise. We keep the value for β presented in Table 5.

5 Low-Rank Covariance Matrices for Speed-Up

Most Gaussian components learned have most of the variance concentrated on very few eigenvalues, as showed in Section 3.2. Following this observation, we tested speeding-up the computation by reducing the ranks of the covariance matrices. For that we selected the $x\%$ first eigenvalues and eigenvectors for the computations. In practice, as can be seen in Table 6, reducing the rank greatly reduces the PSNR for small noise ($\sigma < 20$). It can be used for a gain in computation time with large noise ($\sigma \geqslant 20$), with a small drop in PSNR though.

σ	β_1	β_2	β_3	β_4	β_5	β_6	PSNR gained
5	1	2	5	10	18	30	+0.03
10	1	3	6	12	25	60	+0.02
20	1	4	8	10	30	70	+0.04
25	1	5	8	16	32	64	+0.02
30	1	2	8	16	32	64	+0.03
40	1	2	8	16	32	64	+0.21
50	1	2	8	16	32	64	+0.53

Table 4: Optimization of the β values

$5 \leq \sigma < 30$	$30 \leq \sigma$
$\beta = (1, 4, 8, 16, 32, 64)$	$\beta = (1, 2, 8, 16, 32, 64)$

Table 5: Optimized betas that should be used for the best results

6 Multi-Scale Denoising

In [16] the authors propose a specific multi-scale version for EPLL based on the definition of a complex energy combining the energy introduced in Section 2.2.1 at different scales. It produced improvements in performance both visually and quantitatively. We choose here to add the general multiscale denoising framework [14] which can be applied to any existing single-scale denoising algorithm.

The DCT multiscaler [14] builds a DCT image pyramid. The down-sampling of the image is simply done by extracting the low frequencies from the DCT transform of the image, and by computing the inverse DCT (IDCT) on just those frequencies. Each layer of the pyramid has half the width and half the height of the previous one. In addition, the DCT of an image transforms additive Gaussian white noise into additive Gaussian white noise with its standard deviation getting halved at each successive scale. Thus, no particular adaptation of the initial single scale denoising algorithm is needed to denoise the coarse layers. Recomposing the pyramid is trivial, since it can be reduced to substituting the low frequencies of a layer with the frequencies of the coarser layer.

The drawback of this substitution is that, since each layer is essentially the result of the convolution of the previous one with a sinc-like function, ringing artifacts due to the Gibbs effect unavoidably appear in the result. Thus, if the denoising algorithm alters the high frequencies of the coarse layer, the recomposed image inevitably presents unpleasant ringing artifacts. This ringing is reduced by the "soft fusion" method described in [7].

Facciolo et al. [14] already showed that applying this multiscale fusion to the original EPLL

σ	5%	10%	25%	50%	75%	100%
5	23.07	23.46	27.33	31.50	35.52	37.73
10	23.04	23.42	27.08	30.42	32.75	33.41
20	22.97	23.31	26.46	28.63	29.65	29.87
30	22.90	23.18	25.89	27.32	27.86	27.99
40	22.79	23.03	25.37	26.40	26.74	26.82
Speed-up	×3.55	×3.23	×2.46	×1.82	×1.43	×1

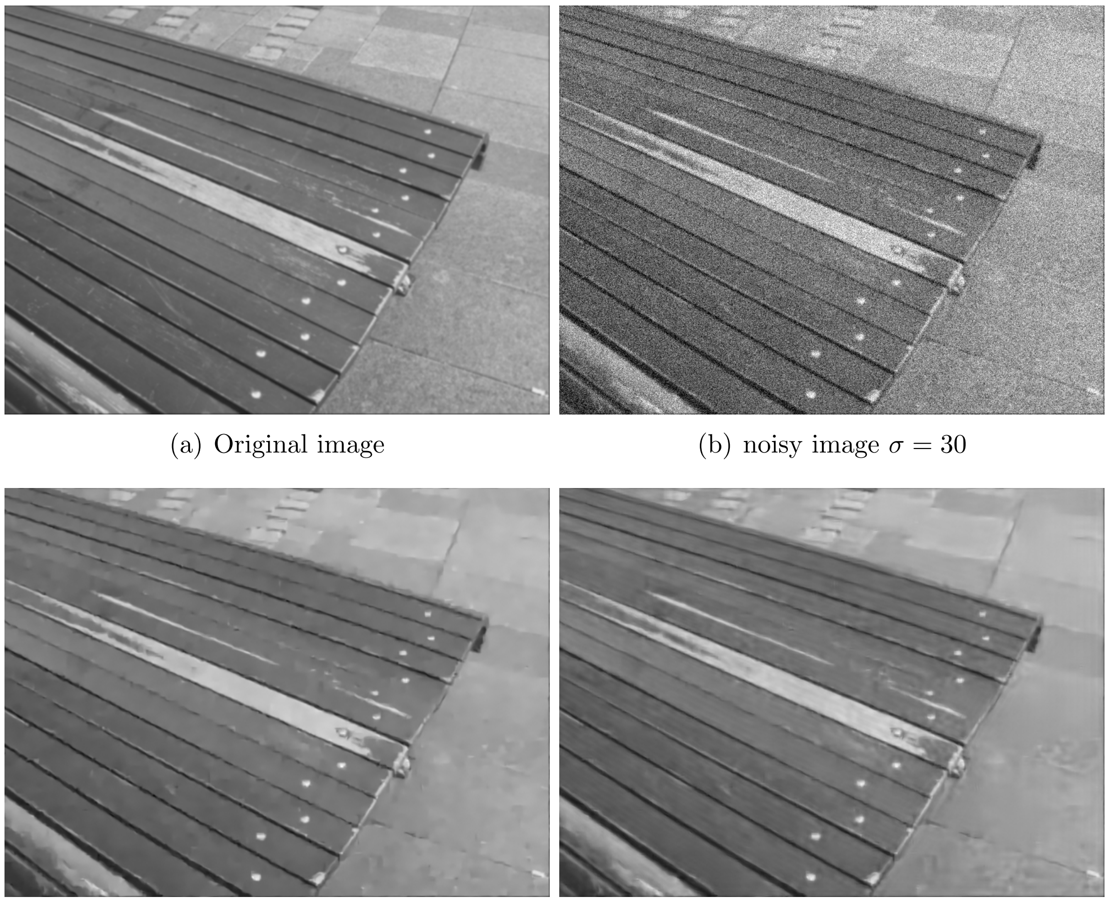
Table 6: Denoising using only a given percentage (in number) of the best eigenvectors, 100% corresponds to full rank. The extra line compares the computation time with the 100% reference time.

yields improvements comparable to those obtained by [16]. Hence, we limit ourselves to examine the multiscale improvement obtained by the "soft fusion" method. We tested the algorithm with 10 images randomly chosen from the test set. We show the results in Table 7. The size of these images is 768 × 576 and the multiscale approach uses 3 scales.

σ	EPLL	MS EPLL
5	**43.389**	43.325
10	**39.513**	39.476
20	35.592	**35.652**
30	33.308	**33.471**
40	31.662	**31.909**

Table 7: Influence of multi-scale denoising.

For a small amount of noise, denoising is easy and it is not necessary to add a complicated multi-scale method which ringing artifacts would worsen the estimation. Furthermore, multi-scale denoising tends to damage thin textures still present for slight noise. However, as soon as $\sigma = 30$, these artifacts are negligible beyond the ones due to the noise and the contribution brought by the multi-scale method begins to be serious. Figure 5 shows an example of multi-scale denoising.


(a) Original image (b) noisy image $\sigma = 30$

(c) EPLL PSNR=33.256 (d) **MS EPLL PSNR=33.451**

Figure 5: Visual comparison between single and multi-scale denoising on an image from the test set.

7 Oracular Denoising

To have an idea of the performance of our algorithm, we can try to run the algorithm with the clean image as oracle. At each iteration (each value of β), we keep the conditional mixing weight from the clean image instead of from the current estimate. Therefore, each patch has always the same Gaussian assigned to calculate the MAP estimate. Table 8 gives the results on the image *Lena*.

σ	Normal	Oracle
5	37.96	38.41
10	33.96	34.63
20	30.58	31.44
30	28.59	29.41
40	27.13	27.80

Table 8: Denoising taking the clean image as oracle

This experiment shows that the learned GMM has a very good denoising potential and that one of the main difficulties is to determine the right Gaussian to calculate the MAP estimate.

A second experiment shows the impact of the DC component estimation. While it is not realistic to have an algorithm that uses the oracle to choose the Gaussian assigned to a patch, it is actually reasonable to see how well the algorithm performs with a better estimate of the DC component. Table 9 shows the results of such an experiment. Overall using a better estimation of the DC components can really improve the quality of denoising.

σ	Normal	mean from cleanI
5	37.88	38.28
10	34.12	34.70
20	30.51	31.44
30	28.60	29.68
40	27.27	27.41

Table 9: Denoising using the mean from the original clean image

8 Denoising of Color Images

To perform color denoising, we use three different methods. The first one is to denoise each channel R, G and B of the noisy image independently using the previous algorithm and to recombine them together. The second one follows the same principle but on a different color basis. These two methods use the GMM learned previously for grayscale patches. The third one is to learn a GMM for color, using a database of color patches.

8.1 Denoising Channel by Channel

Using the R, G and B channels independently creates noticeable color artifacts. The noisy image y in the usual RGB color space can be converted to a different color space where an independent denoising of each channel will reduce these artifacts.

We tested 2 transformations: the YUV transform denoted by the matrix A_{YUV} and the opponent transform used in [4], A_{OPP}.

$$A_{\text{YUV}} = \begin{pmatrix} 0.30 & 0.59 & 0.11 \\ -0.15 & -0.29 & 0.44 \\ 0.61 & -0.51 & -0.10 \end{pmatrix},$$

$$A_{\text{OPP}} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{pmatrix}.$$

We wrote the matrices above without normalization for readability, but we worked with normalized versions of the matrices where each row has unit norm (the normalized A_{OPP} is an orthogonal matrix, and keeps the noise in the channels uncorrelated). In that way, σ still is the value of the standard deviation of the noise on each channel. The compared denoising performances with these different color transforms are shown in Table 10.

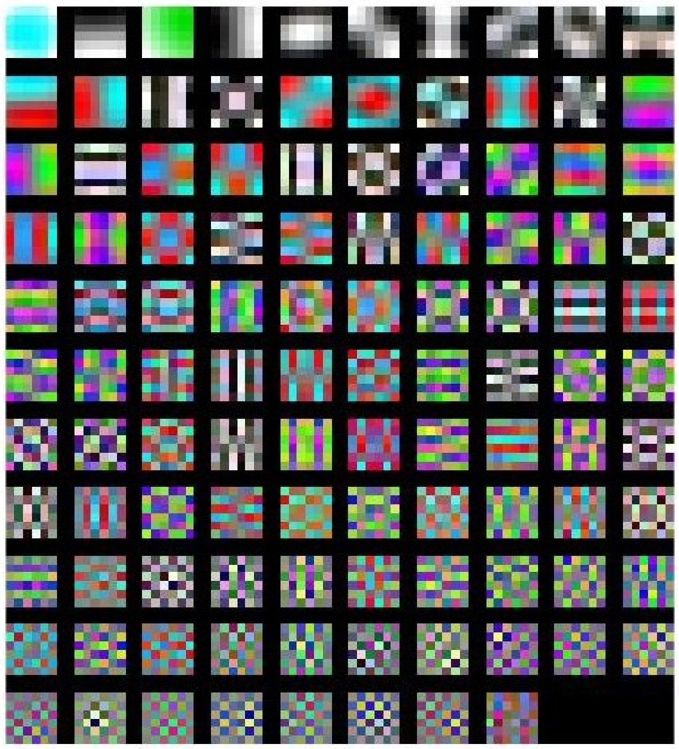
The YUV and OPP color basis lead to much better results in PSNR than the RGB basis. As was observed in 10, the first channel of the OPP transform is an average of the three colors, which captures the geometry of the patch. The average improves the signal to noise ratio, allowing for a better performance of the algorithm in this component. The remaining components express the chromatic content of the patch. They are differences of channels, which cancel or attenuate the signal. Although the SNR is worse for these components, it can be expected that the remaining signal has a higher regularity and is easier to denoise. A similar reasoning can be applied to the YUV color space. However, A_{OPP} is an orthogonal matrix and keeps the noise decorrelated, while A_{YUV} does not. This might be the reason for the better performance of the OPP color space.

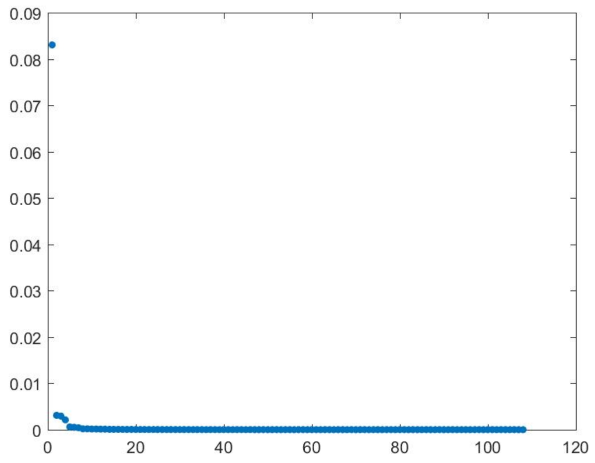
8.2 Learning in Color

To learn in color, we used the original database of color images [15] with size reduced by 2 as explained in 3.2.

We extracted color patches setting the d-dimensional patches from each channel R, G and B together in a $3d$ vector. We used the exact same learning method as before (EM) to learn a GMM, in this case with a dimensionality of $3d$. We kept $N = 2 \cdot 10^6$ and $K = 200$ but we now use a smaller patch size $\sqrt{d} = 6$ because of memory limitations. Four components of the GMM are represented in Figures 6, 7, 8, 9.

In Table 10, we compare all the denoising methods previously introduced: channel by channel with the 3 different color bases and with the prior learned in color. The study was conducted on 10 test images from the test set of images. The RMSE between the clean image and the denoised is computed on all channels.


(a) eigenvectors



(b) corresponding eigenvalues

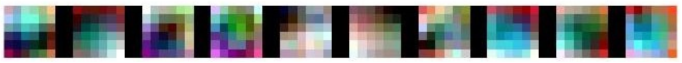
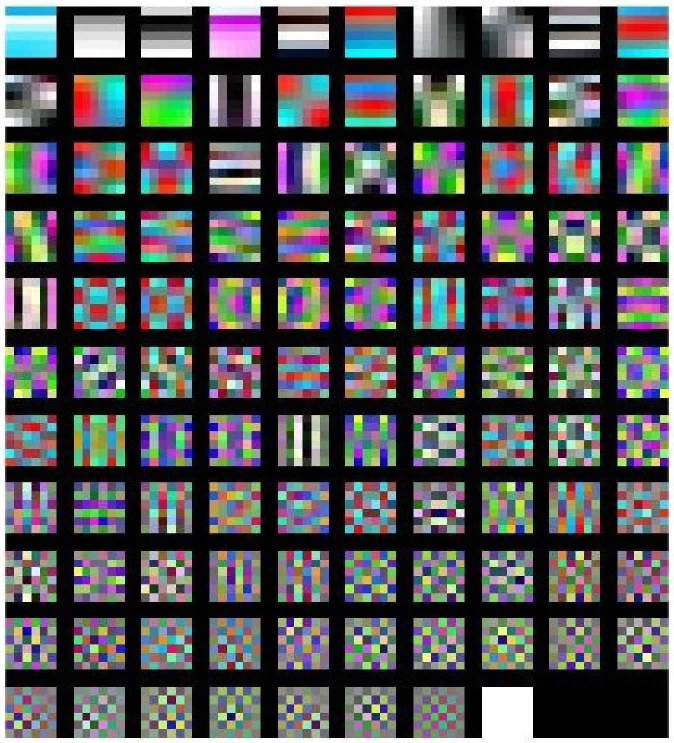
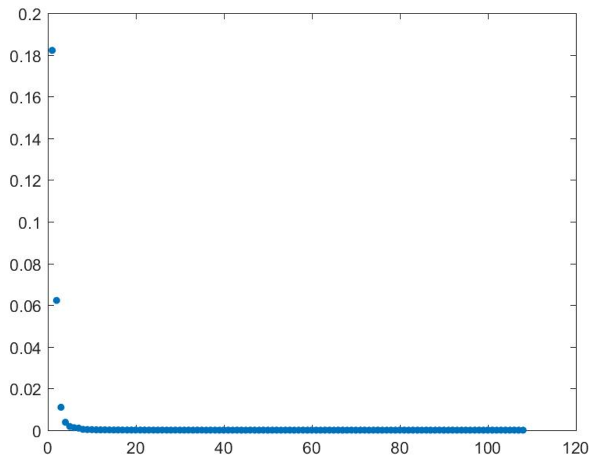

(c) 10 simulated patches

Figure 6: Component 2 of the Gaussian Mixture Model. The very low eigenvalues indicates that it represents flat patches with blue as a dominant color.


(a) eigenvectors



(b) corresponding eigenvalues

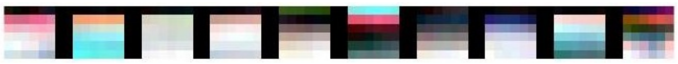
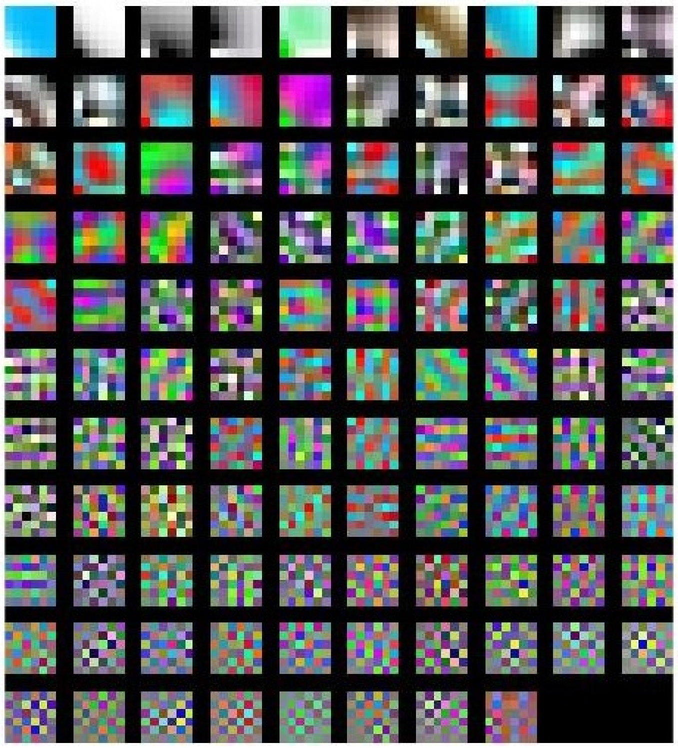
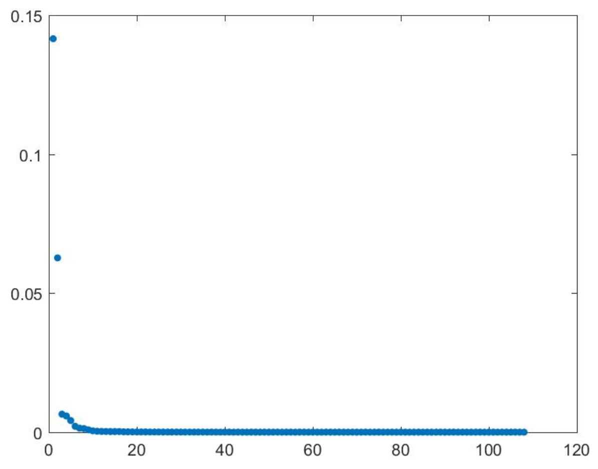

(c) 10 simulated patches

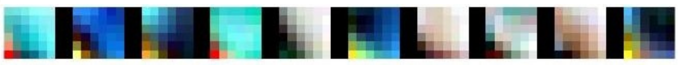
Figure 7: Component 6 of the Gaussian Mixture Model. The sharp decrease of the eigenvalues gives importance only to the top eigenvectors in the simulated patches: it simulates horizontal edges.



(a) eigenvectors

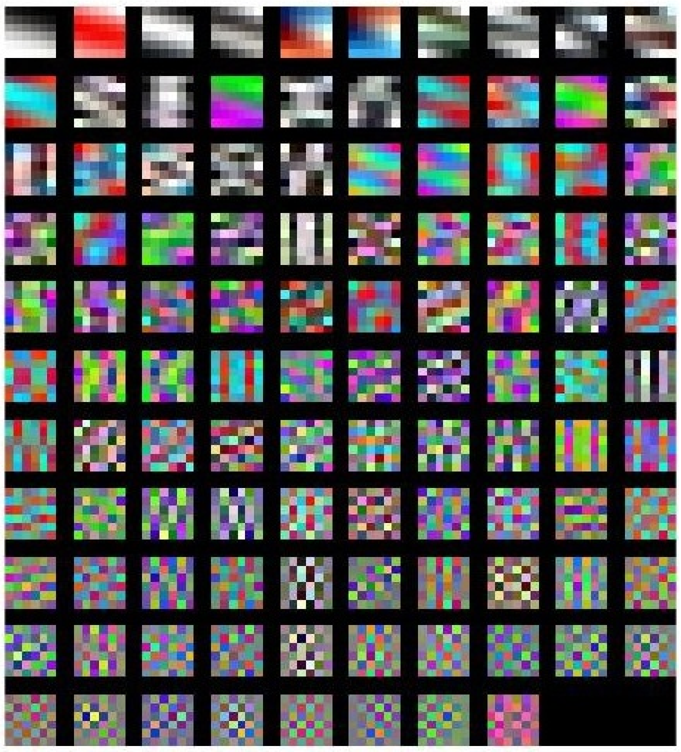


(b) corresponding eigenvalues

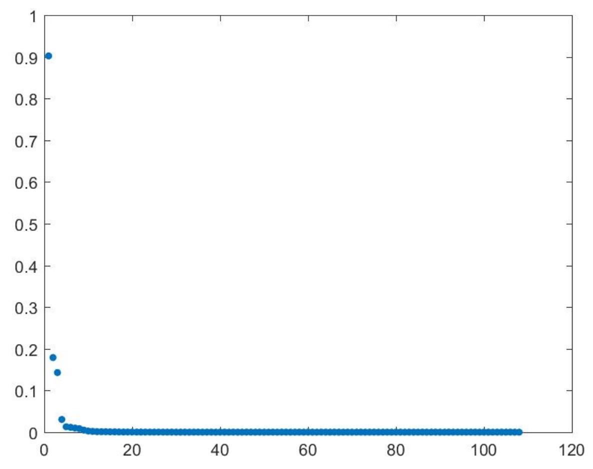


(c) 10 simulated patches

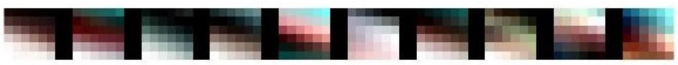
Figure 8: Component 34 of the Gaussian Mixture Model. The sharp decrease of the eigenvalues gives importance only to the top eigenvectors in the simulated patches: it simulates a corner.



(a) eigenvectors



(b) corresponding eigenvalues



(c) 10 simulated patches

Figure 9: Component 45 of the Gaussian Mixture Model. The sharp decrease of the eigenvalues gives importance only to the top eigenvectors in the simulated patches: it simulates diagonal edges.

σ	RGB	YUV	OPP	GMM color
5	41.300	41.954	42.239	**42.508**
10	37.367	38.066	38.457	**38.716**
20	33.544	34.316	34.745	**34.944**
30	31.321	32.150	32.609	**32.690**
40	29.757	29.909	**31.106**	31.069
50	28.560	29.509	**29.910**	29.766
60	28.560	29.509	**29.910**	29.766

Table 10: Comparison between the different color denoising methods

For $\sigma \leq 30$, the color Gaussian mixture on the color image leads to better results than the grayscale one on each channel. However, for $\sigma > 30$, the opposite is true. We show in Figures 12,13, 14 the visual difference between the “OPP” and the “GMM-color” methods. It shows that for high noise level, the Gaussian mixture sometimes fails to reconstruct the right color. Indeed, with the second option, we realize three different estimations for each patch. One mistake on one channel is less likely to affect the visual result. For example, in Figure 14, the color on flat regions (sky and road) has not been reconstructed correctly. Even if the PSNR is sometimes close (12,13, 14), the “OPP-method” gives a better visual quality, especially for flat regions or textures in color. Nevertheless, denoising with a color GMM is likely to be improved using more components. Due to memory limits, this experience has not been realized.

Although we didn’t perform a detailed study of the multi-scale version of the algorithm in color, we show in Figure 10 one example of color single and multi-scale denoising. It has been computed with the denoising channel by channel. The contribution of the multi-scale version on flat regions is clear.

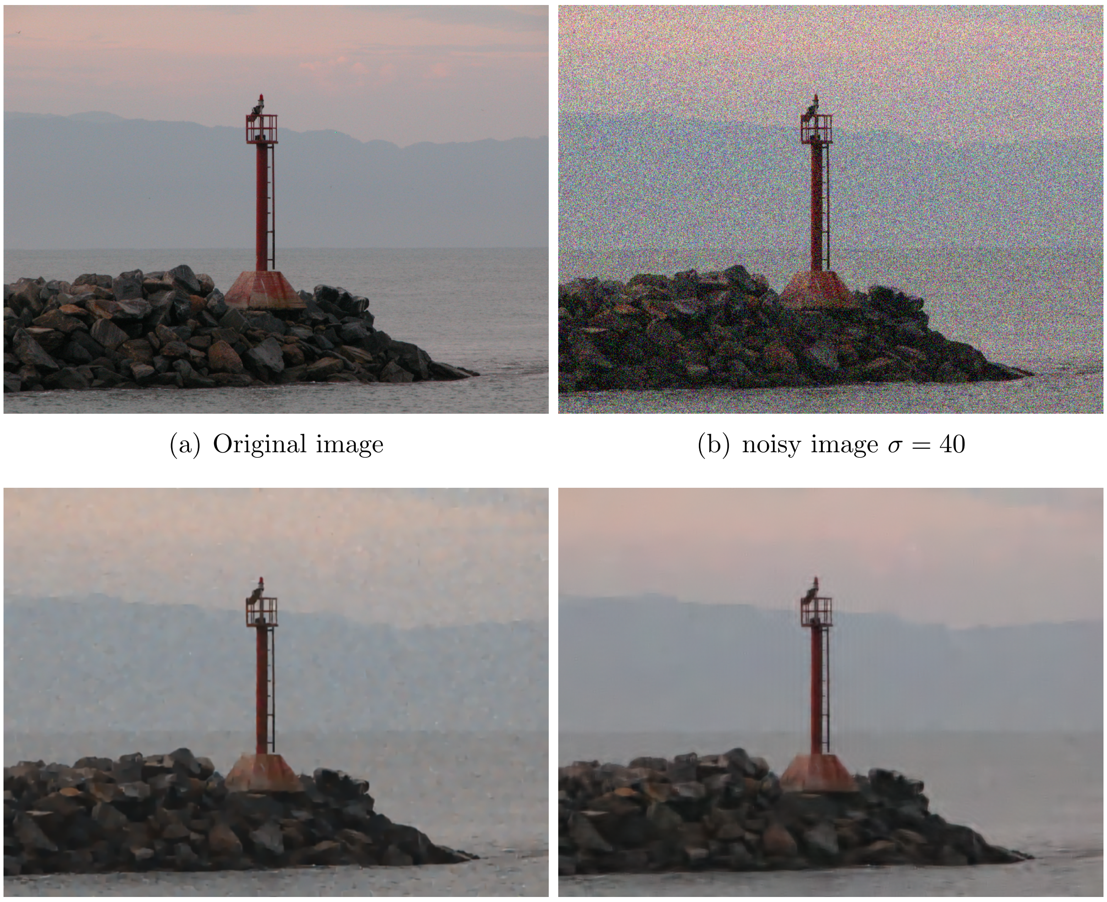
9 Comparison with Other Algorithms

In order to evaluate the real capacity of the algorithm, we compared it to state-of-the-art methods implemented in IPOL: BM3D [9], Non-local Bayes [10], DCT [7] and K-SVT [13]. It is also interesting to compare visually all methods. We present in Figures 11, 12, 13, 14, for each algorithm, the denoising of different images and the corresponding PSNR.

10 Conclusions

In this work we studied in detail the EPLL algorithm, which performs denoising based on the assumption that the distribution of image patches can be modeled by a GMM. We proposed two extensions of it: a multi-scale version and a color version for which we tested several approaches. Our work has led us to the following conclusions:

- The algorithm stands among the best denoising algorithms.
- The prior learned over a large database of images models efficiently, only with 200 Gaussians, the diversity of patches. Indeed, the denoising performance is close to that of NL-Bayes, a method that learns an appropriate Gaussian for each group of some 100 similar patches in the image. Using 200 Gaussian components seems to be enough to reach a similar performance.
- The parameters introduced by [19] are close to be optimal. The parameter optimization did not lead to significant gains in denoising quality.

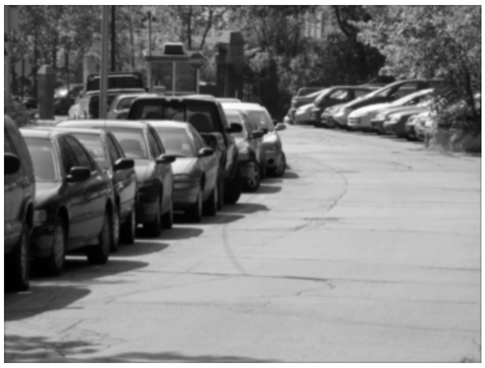


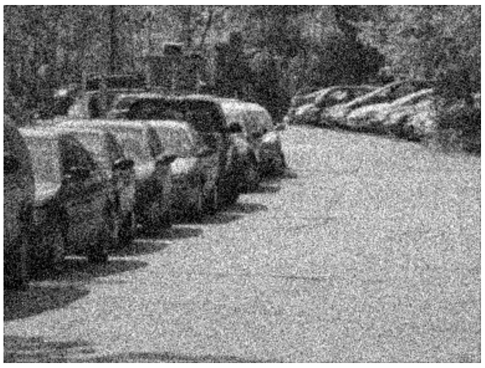
(a) Original image (b) noisy image $\sigma = 40$

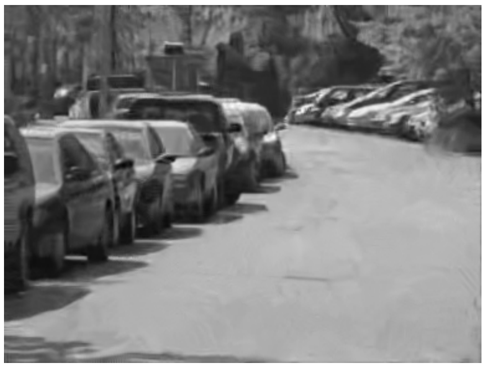
(c) EPLL PSNR=31,124 (d) **MS EPLL PSNR=31.255**

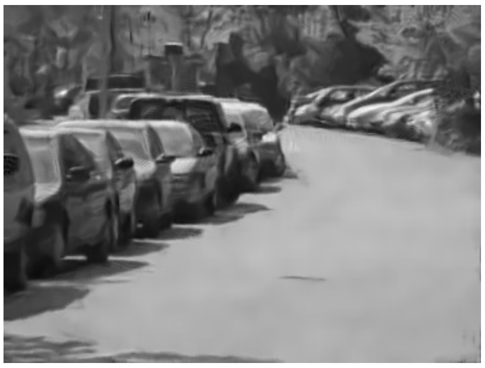
Figure 10: Single and multi-scale color denoising

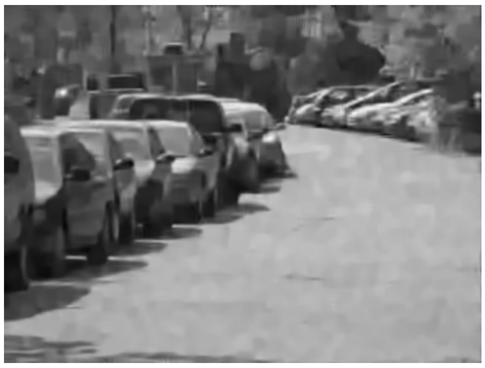
- We have shown that the multi-scale framework [14] improves significantly the treatment of low-frequency noise.
- There is still room for improvements on color denoising. The parameters of the learning have not been optimized and gain in PSNR is likely to be possible by improving the number of components in the Gaussian Mixture.
- The influence of the size of the patch is also to be studied in further experiments.
- The data plays a fundamental role in the quality of the denoising result. In the article we presented results on a base of TIF images. The learning had also been realized on the JPEG database [12] used by [19]. Each prior gave better denoising results on test images extracted from its original database. This JPEG influence could be studied in further experiments.

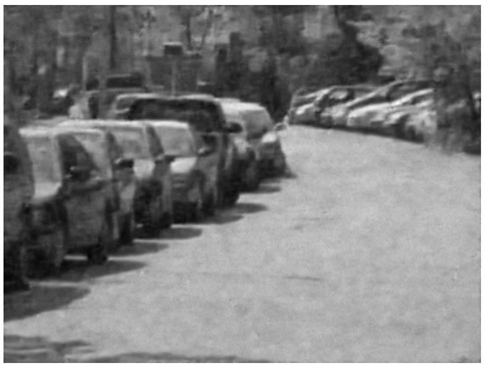

(a) Original image


(b) noisy image $\sigma = 30$


(c) BM3D PSNR=29.89


(d) NL-Bayes PSNR=29.719


(e) DCT PSNR=29.175


(f) K-SVD PSNR=28.871

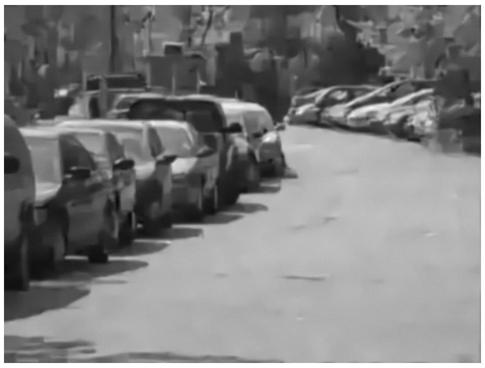
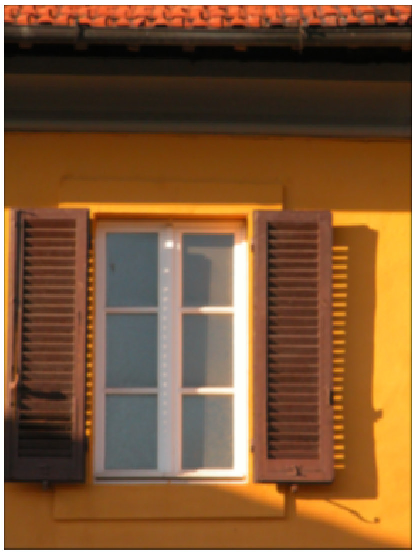
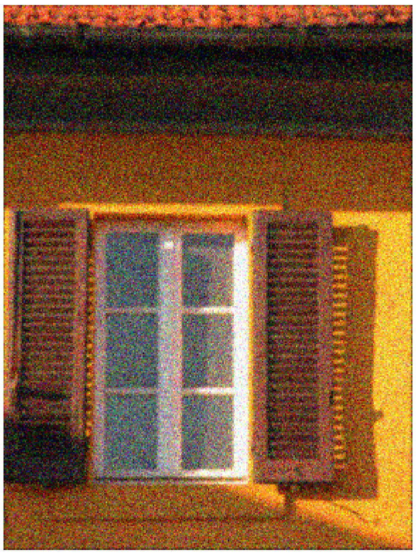
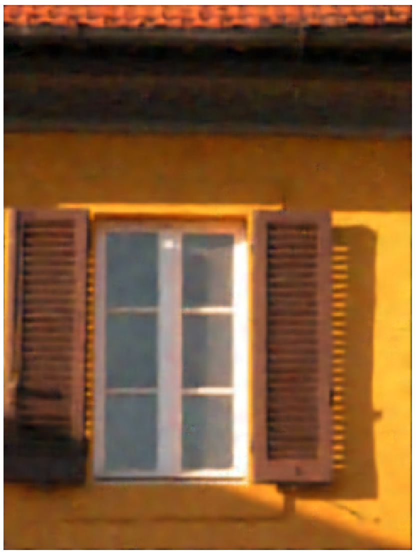
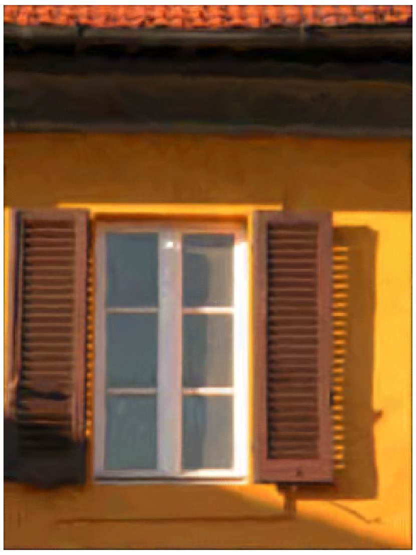

(g) EPLL PSNR=29.886

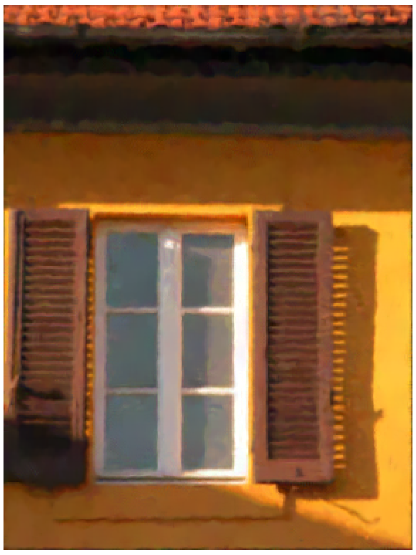
Figure 11: Comparison with a black and white image from the test dataset

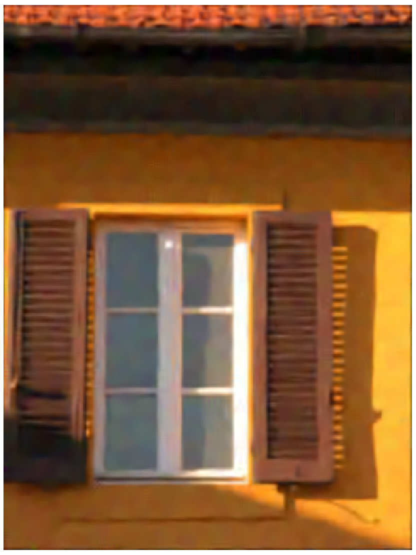

(a) Original image

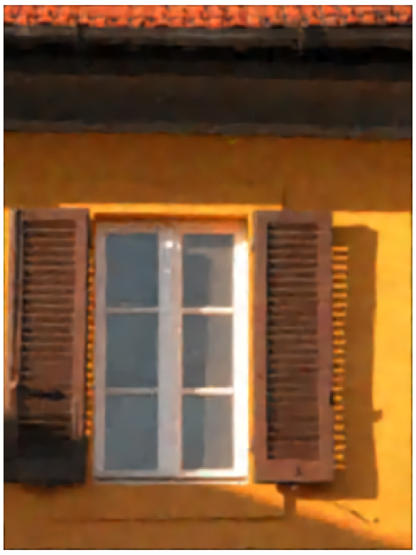

(b) noisy image $\sigma = 30$


(c) K-SVD PSNR=32.847


(d) BM3D PSNR=33.950


(e) NLBayes PSNR=32.833


(f) DCT PSNR=32.941


(g) GMM-color PSNR=33.466

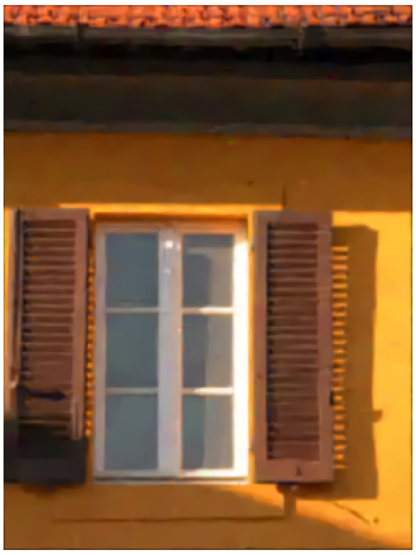
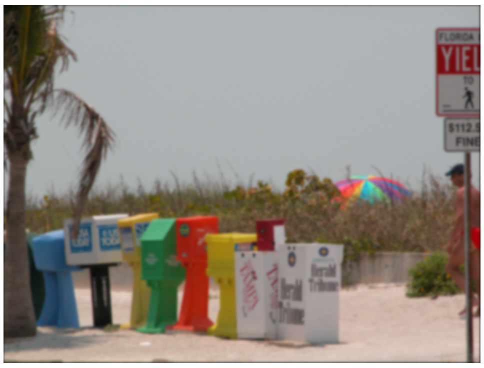
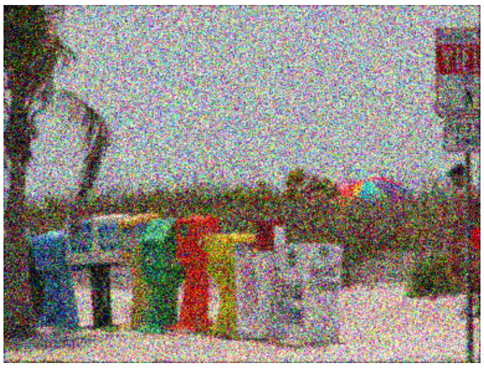
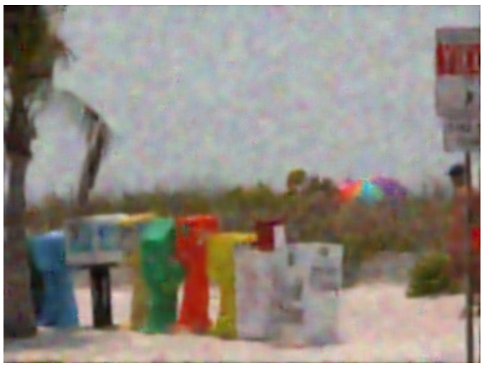
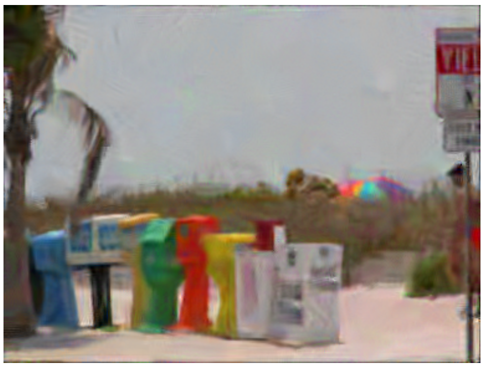

(h) OPP PSNR=33.477

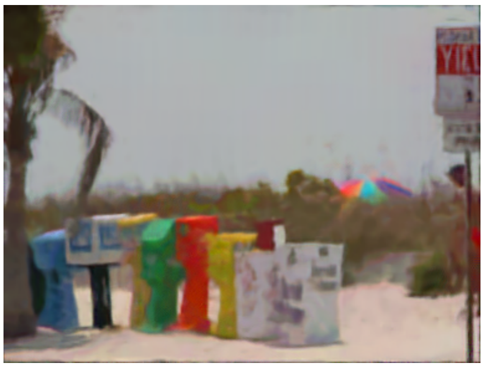
Figure 12: Comparison with a color image from the test dataset

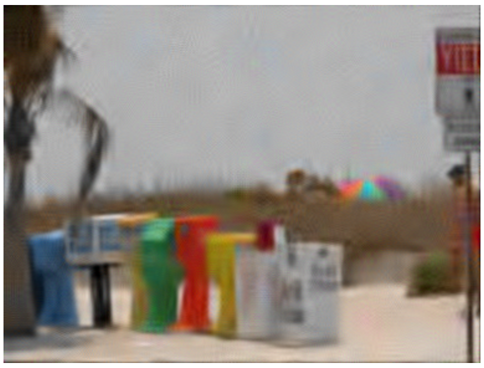

(a) Original image

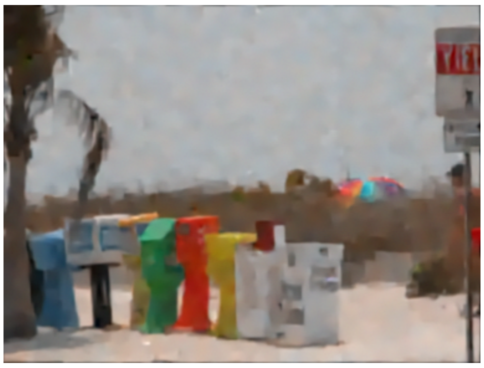

(b) noisy image $\sigma = 60$


(c) K-SVD PSNR=28.63


(d) BM3D PSNR=29.52


(e) NLBayes PSNR=29.59


(f) DCT PSNR= 28.42


(g) GMM-color PSNR=29.516

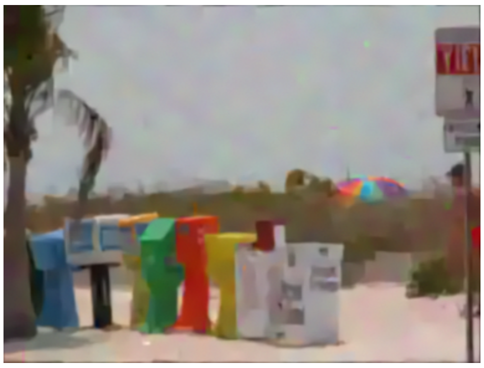
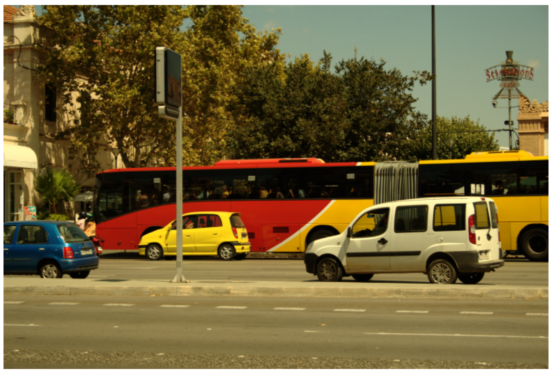
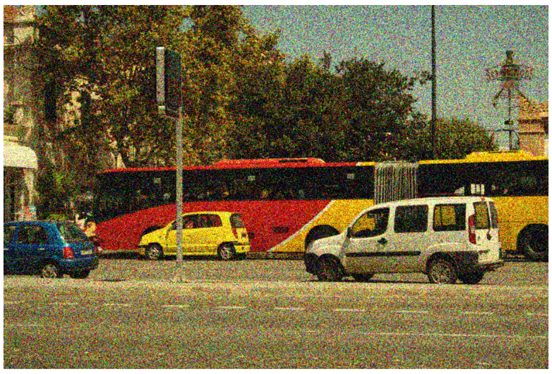
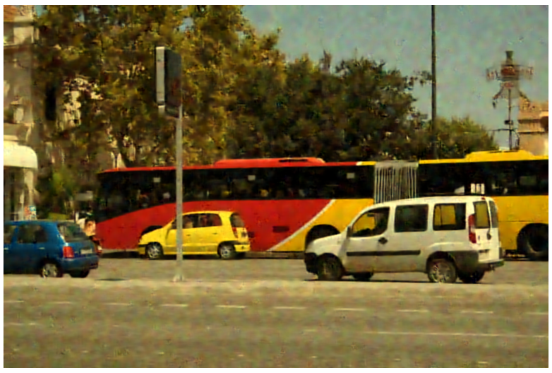
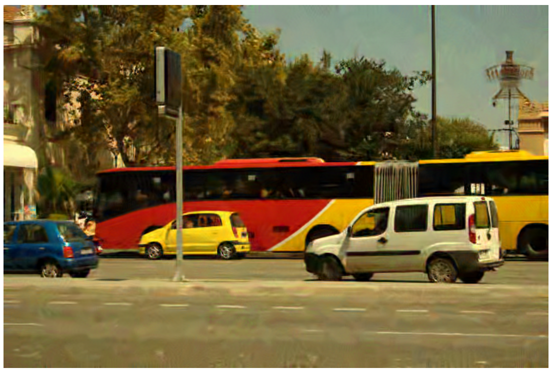

(h) OPP PSNR=29.610

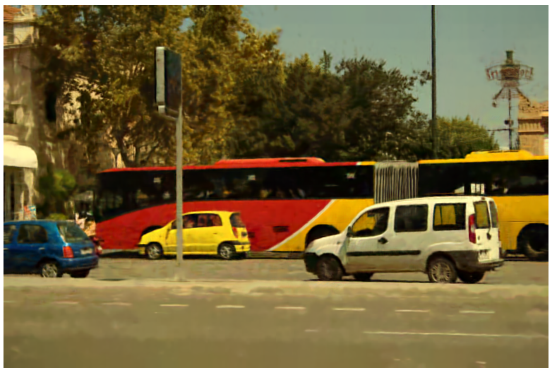
Figure 13: Comparison with a color image from the test dataset

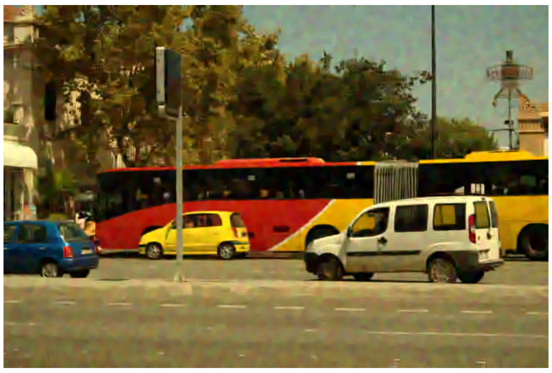

(a) Original image

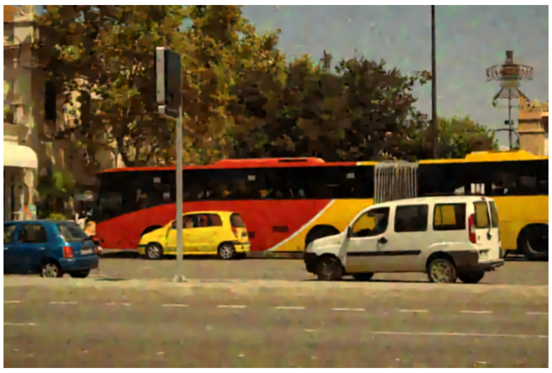

(b) noisy image $\sigma = 50$


(c) K-SVD PSNR=26.14


(d) BM3D PSNR=26.44


(e) NLBayes PSNR=26.83


(f) DCT PSNR=26.00


(g) GMM-color PSNR=25.78

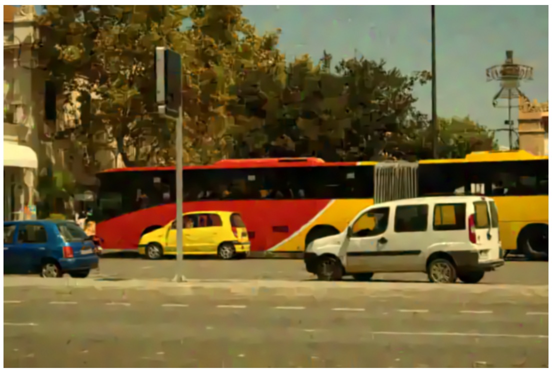

(h) OPP PSNR= 26.14

Figure 14: Comparison using the image traffic

Acknowledgment

Work partly financed by IDEX Paris-Saclay IDI 2016, ANR-11-IDEX-0003-02, Office of Naval research grant N00014-17-1-2552, DGA Astrid project "filmer la Terre" n^o ANR-17-ASTR-0013-01, DGA Defals challenge n^o ANR-16-DEFA-0004-01, MENRT and a scholarship of the French Ministry for Higher Studies, Research and Innovation.

Image Credits

The McGill Calibrated Colour Image Database for the images used in Figures 5, 10, 11, 12, 13 and Miguel Colom for the traffic image used in Figure 14.

References

[1] B.Coll A.Buades and JM.Morel, *Non-Local Means Denoising*, Image Processing On Line, 1 (2011), pp. 208–212. http://dx.doi.org/10.5201/ipol.2011.bcm_nlm.

[2] S. P. Awate and R. T. Whitaker, *Unsupervised, information-theoretic, adaptive image filtering for image restoration*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 28 (2006), pp. 364–376. http://dx.doi.org/10.1109/TPAMI.2006.64.

[3] C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006. ISBN 0387310738.

[4] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, *Color image denoising via sparse 3d collaborative filtering with grouping constraint in luminance-chrominance space*, in Proceedings of IEEE International Conference on Image Processing (ICIP), vol. 1, IEEE, 2007, pp. I–313. https://doi.org/10.1109/ICIP.2007.4378954.

[5] C-A. Deledalle, S. Parameswaran, and T.Q. Nguyen, *Image denoising with generalized Gaussian mixture model patch priors*, SIAM Journal on Imaging Sciences, 11 (2018), pp. 2568–2609. https://doi.org/10.1137/18M116890X.

[6] D. Geman and C. Yang, *Nonlinear image recovery with half-quadratic regularization*, IEEE Transactions on Image Processing, 4 (1995), pp. 932–946.

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

[8] A. Houdard, C. Bouveyron, and J. Delon, *High-dimensional mixture models for unsupervised image denoising (HDMI)*, (2018). https://doi.org/10.1137/17M1135694.

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

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

[11] A. Buades J.M. Morel M. Lebrun, M. Colom, *Secrets of image denoising cuisine*, Acta Numerica, 21 (2012), pp. 475–576. https://doi.org/10.1017/S0962492912000062.

[12] 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 Proceedings of International Conference on Computer Vision, vol. 2, July 2001, pp. 416–423. https://doi.org/10.1109/ICCV.2001.937655.

[13] M.Lebrun and A.Leclaire, *An Implementation and Detailed Analysis of the K-SVD Image Denoising Algorithm*, Image Processing On Line, 2 (2012), pp. 96–133. https://doi.org/10.5201/ipol.2012.llm-ksvd.

[14] G. Facciolo N. Pierazzo, J.M. Morel, *Multi-scale DCT denoising*, Image Processing On Line, (2017), pp. 288–308. https://doi.org/10.5201/ipol.2017.201.

[15] A. Olmos and A.A Kingdom, *A biologically inspired algorithm for the recovery of shading and reflectance images*, in Perception, vol. 33, 2004, pp. 1463–1473. https://doi.org/10.1068/p5321.

[16] V. Papyan and M. Elad, *Multi-scale patch-based image restoration*, IEEE Transactions on image processing, 25 (2016), pp. 249–261. https://doi.org/10.1109/TIP.2015.2499698.

[17] A. Saint-Dizier, J. Delon, and C. Bouveyron, *A unified view on patch aggregation*. working paper or preprint, Aug. 2018.

[18] G. Yu, G. Sapiro, and S. Mallat, *Solving inverse problems with piecewise linear estimators: From Gaussian mixture models to structured sparsity*, IEEE Transactions on Image Processing, 21 (2012), pp. 2481–2499. https://doi.org/10.1109/TIP.2011.2176743.

[19] D. Zoran and Y. Weiss, *From learning models of natural image patches to whole image restoration*, in Proceedings of the 13th International Conference on Computer Vision (ICCV), 2011, pp. 479–486. http://dx.doi.org/10.1109/ICCV.2011.6126278.