

Accelerating GMM-based patch priors for image restoration: Three ingredients for a 100x speed-up

Shibin Parameswaran, Charles-Alban Deledalle, Loïc Denis, Truong Q. Nguyen, Truong Trong Nguyen

▶ **To cite this version:**

Shibin Parameswaran, Charles-Alban Deledalle, Loïc Denis, Truong Q. Nguyen, Truong Trong Nguyen. Accelerating GMM-based patch priors for image restoration: Three ingredients for a 100x speed-up. 2017. hal-01617722v2

HAL Id: hal-01617722
https://hal.science/hal-01617722v2

Preprint submitted on 27 Aug 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Accelerating GMM-based patch priors for image restoration: Three ingredients for a 100× speed-up

Shibin Parameswaran, Charles-Alban Deledalle, Loïc Denis and Truong Q. Nguyen

Because of ArXiv file size limitation, images have been compressed in this document. Assessing restoration quality by visiual inspection is then impossible due to compression artifacts. Please refer to `https://www.math.u-bordeaux.fr/~cdeledal/fepll` to access a pdf file with optimal quality.

***Abstract*—Image restoration methods aim to recover the underlying clean image from corrupted observations. The Expected Patch Log-likelihood (EPLL) algorithm is a powerful image restoration method that uses a Gaussian mixture model (GMM) prior on the patches of natural images. Although it is very effective for restoring images, its high runtime complexity makes EPLL ill-suited for most practical applications. In this paper, we propose three approximations to the original EPLL algorithm. The resulting algorithm, which we call the fast-EPLL (FEPLL), attains a dramatic speed-up of two orders of magnitude over EPLL while incurring a negligible drop in the restored image quality (less than 0.5 dB). We demonstrate the efficacy and versatility of our algorithm on a number of inverse problems such as denoising, deblurring, super-resolution, inpainting and devignetting. To the best of our knowledge, FEPLL is the first algorithm that can competitively restore a 512×512 pixel image in under 0.5s for *all* the degradations mentioned above *without* specialized code optimizations such as CPU parallelization or GPU implementation.**

***Index Terms*—Image restoration, image patch, Gaussian mixture model, efficient algorithms**

I. Introduction

Patch-based methods form a very popular and successful class of image restoration techniques. These methods process an image on a patch-by-patch basis where a patch is a small sub-image (*e.g.*, of 8×8 pixels) that captures both geometric and textural information. Patch-based algorithms have been at the core of many state-of-the-art results obtained on various image restoration problems such as denoising, deblurring, super-resolution, defogging, or compression artifact removal to name a few. In image denoising, patch-based processing became popular after the success of the Non-Local Means algorithm [2]. Subsequently, continued research efforts have led to significant algorithmic advancements in this area [1], [8], [43], [11], [29], [40], [26]. Other inverse problems such as image super-resolution and image deblurring have also benefited from patch-based models [9], [16], [32], [37], [14], [27].

Among these various patch-based methods, the Expected Patch Log-Likelihood algorithm (EPLL) [43] deserves a special mention due to its restoration performance and versatility. The EPLL introduced an innovative application of Gaussian Mixture Models (GMMs) to capture the prior distribution of patches in natural images. Note that a similar idea was introduced concurrently in [40]. The success of this method is evident from the large number of recent works that extend the original EPLL formulation [19], [38], [28], [4], [31], [33], [20]. However, a persistent problem of EPLL-based algorithms is their high runtime complexity. For instance, it is orders of magnitude slower than the well-engineered BM3D image denoising algorithm [8]. However, extensions of BM3D that perform super-resolution [10] and other inverse problems [22] require fundamental algorithmic changes, making BM3D far less adaptable than EPLL. Other approaches that are as versatile as EPLL [36], [5], [21] either lack the algorithmic efficiency of BM3D or the restoration efficacy of EPLL.

Another class of techniques that arguably offers better runtime performance than EPLL-based methods (but not BM3D) are those based on deep learning. With the advancements in computational resources, researchers have attempted to solve some classical inverse problems using multi-layer perceptrons [3] and deep networks [6], [13], [23]. These methods achieve very good restoration performance, but are heavily dependent on the amount of training data available for each degradation scenario. Most of these methods learn filters that are suited to restore a specific noise level (denoising), blur (deblurring) or upsampling factor (super-resolution), which makes them less attractive to serve as generic image restoration solutions. More recently, Zhang *et al.* [42] demonstrated the use of deep residual networks for general denoising problems, single-image super-resolution and compression artifact removal. Unlike earlier deep learning efforts, their approach can restore images with different noise levels using a single model which is learned by training on image patches containing a range of degradations. Even in this case, the underlying deep learning model requires retraining whenever a new degradation scenario different from those considered during the learning stage is encountered.

More recently, [34] proposed training a single deep network to solve many inverse problems. This work uses an iterative scheme that alternatively enforces a good fit to the learned prior model of natural images (via a projection operator performed by a deep neural network) and a satisfying fidelity to the data (via the direct model). The projection operator takes the form of an auto-encoder, trained by an adversarial strategy, that processes 64×64 patches with 1024-dimensional latent space (*i.e.*, 4 times smaller). This framework shows promise but, in its current form, requires a very large image database for training. In addition, almost all of the training and testing in the original publication are conducted on small 64×64 images. Another limitation, as noted by the authors, is that the

S. Parameswaran, C.-A. Deledalle and T. Nguyen are with the Department of Electrical and Computer Engineering, University of California, San Diego, 9500 Gilman Dr, La Jolla, CA 92093, e-mail: sparames@ucsd.edu, cdeledalle@eng.ucsd.edu and tqn001@eng.ucsd.edu.

C. Deledalle is also with IMB, CNRS, Univ. Bordeaux, Bordeaux INP, F-33405 Talence, France, e-mail: charles-alban.deledalle@math.u-bordeaux.fr

L. Denis is with Univ Lyon, UJM-Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, Saint-Etienne, France, e-mail: loic.denis@univ-st-etienne.fr.

regularization parameter that controls the contribution of prior is fixed during training stage. In other words, regularization parameter cannot be changed during test time which can be an issue in certain situations. In contrast, our approach needs much less data during training stage and, during test time, the regularization parameter can be tuned according to the signal-to-noise ratio of the image being addressed.

Moreover, it is much harder to gain insight into the actual model learned by a deep architecture compared to a GMM. For this reason, even with the advent of deep learning methods, flexible algorithms like EPLL that have a transparent formulation remain relevant for image restoration.

Recently, researchers have tried to improve the speed of EPLL by replacing the most time-consuming operation in the EPLL algorithm with a machine learning-based technique of their choice [39], [35]. These methods were successful in accelerating EPLL to an extent but did not consider tackling all of its bottlenecks. In contrast, this paper focuses on accelerating EPLL by proposing algorithmic approximations to all the prospective bottlenecks present in the original algorithm proposed by Zoran *et al.* [43]. To this end, we first provide a complete computational and runtime analysis of EPLL, present a new and efficient implementation of original EPLL algorithm and then finally propose innovative approximations that lead to a novel algorithm that is more than $100\times$ faster compared to the efficiently implemented EPLL (and $350\times$ faster than the runtime obtained by using the original implementation [43]).

Contributions: The main contributions of this work are the following. We introduce three strategies to accelerate patch-based image restoration algorithms that use a GMM prior. We show that, when used jointly, they lead to a speed-up of the EPLL algorithm by two orders of magnitude. Compared to the popular BM3D algorithm, which represents the current state-of-the-art in terms of speed among CPU-based implementations, the proposed algorithm is almost an order of magnitude faster. The three strategies introduced in this work are general enough to be applied individually or in any combination to accelerate other related algorithms. For example, the random subsampling strategy is a general technique that could be reused in any algorithm that considers overlapping patches to process images; the flat tail spectrum approximation can accelerate any method that needs Gaussian log-likelihood or multiple Mahalanobis metric calculations; finally, the binary search tree for Gaussian matching can be included in any algorithm based on a GMM prior model and can be easily adapted for vector quantization techniques that use a dictionary.

For reproducibility purposes, we release our software on GitHub along with a few usage demonstrations (available at https://goo.gl/xjqKUA).

II. Expected Patch Log-Likelihood (EPLL)

We consider the problem of estimating an image $\boldsymbol{x} \in \mathbb{R}^N$ (N is the number of pixels) from noisy linear observations $\boldsymbol{y} = \mathcal{A}\boldsymbol{x} + \boldsymbol{w}$, where $\mathcal{A} : \mathbb{R}^N \to \mathbb{R}^M$ is a linear operator and $\boldsymbol{w} \in \mathbb{R}^M$ is a noise component assumed to be white and Gaussian with variance σ^2. In a standard denoising problem $\mathcal{A}$ is the identity matrix, but in more general settings, it can account for loss of information or blurring. Typical examples for operator $\mathcal{A}$ are: a low pass filter (for *deconvolution*), a masking operator (for *inpainting*), or a projection on a random subspace (for *compressive sensing*). To reduce noise and stabilize the inversion of $\mathcal{A}$, some *prior* information is used for the estimation of $\boldsymbol{x}$. The EPLL introduced by Zoran and Weiss [43] includes this *prior* information as a model for the distribution of patches found in natural images. Specifically, the EPLL defines the restored image as the maximum *a posteriori* estimate, corresponding to the following minimization problem:

$$\underset{\boldsymbol{x}}{\operatorname{argmin}} \frac{P}{2\sigma^2}\|\mathcal{A}\boldsymbol{x} - \boldsymbol{y}\|^2 - \sum_{i\in\mathcal{I}} \log p\left(\mathcal{P}_i\boldsymbol{x}\right) \tag{1}$$

where $\mathcal{I} = \{1, \ldots, N\}$ is the set of pixel indices, $\mathcal{P}_i : \mathbb{R}^N \to \mathbb{R}^P$ is the linear operator extracting a patch with P pixels centered at the pixel with location i (typically, $P = 8\times 8$), and $p(.)$ is the *a priori* probability density function (*i.e.*, the statistical model of noiseless patches in natural images). While the first term in eq. (1) ensures that $\mathcal{A}\boldsymbol{x}$ is close to the observations $\boldsymbol{y}$ (this term is the negative log-likelihood under the white Gaussian noise assumption), the second term regularizes the solution $\boldsymbol{x}$ by favoring an image such that all its patches fit the *prior* model of patches in natural images. The authors of [43] showed that this *prior* can be well approximated (upon removal of the DC component of each patch) using a zero-mean Gaussian Mixture Model (GMM) with $K = 200$ components, that reads for any patch $\boldsymbol{z} \in \mathbb{R}^P$, as

$$p\left(\boldsymbol{z}\right) = \sum_{k=1}^{K} w_k \frac{1}{(2\pi)^{P/2}|\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2}\boldsymbol{z}^t\boldsymbol{\Sigma}_k^{-1}\boldsymbol{z}\right), \tag{2}$$

where the weights w_k (such that $w_k > 0$ and $\sum_k w_k = 1$) and the covariance matrices $\boldsymbol{\Sigma}_k \in \mathbb{R}^{P\times P}$ are estimated using the Expectation-Maximization algorithm [12] on a dataset consisting of 2 million "clean" patches extracted from the training set of the Berkeley Segmentation (BSDS) dataset [30].

Half-quadratic splitting: Problem (1) is a large non-convex problem where $\mathcal{A}$ couples all unknown pixel values $\boldsymbol{x}$ and the patch prior is highly non-convex. A classical workaround, known as half-quadratic splitting [15], [25], is to introduce N auxiliary unknown vectors $\boldsymbol{z}_i \in \mathbb{R}^P$, and consider instead the penalized optimization problem that reads, for $\beta > 0$, as

$$\underset{\boldsymbol{x},\boldsymbol{z}_1,\ldots,\boldsymbol{z}_N}{\operatorname{argmin}} \frac{P}{2\sigma^2}\|\mathcal{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{\beta}{2}\sum_{i\in\mathcal{I}}\|\mathcal{P}_i\boldsymbol{x} - \boldsymbol{z}_i\|^2 - \sum_{i\in\mathcal{I}}\log p\left(\boldsymbol{z}_i\right). \tag{3}$$

When $\beta \to \infty$, the problem (3) becomes equivalent to the original problem (1). In practice, an increasing sequence of β is considered, and an alternating optimization scheme is used:

$$\left\{\hat{\boldsymbol{z}}_i \leftarrow \underset{\boldsymbol{z}_i}{\operatorname{argmin}} \frac{\beta}{2}\|\mathcal{P}_i\hat{\boldsymbol{x}} - \boldsymbol{z}_i\|^2 - \log p\left(\boldsymbol{z}_i\right)\right\}_{i=1..N} \tag{4}$$

$$\hat{\boldsymbol{x}} \leftarrow \underset{\boldsymbol{x}}{\operatorname{argmin}} \frac{P}{2\sigma^2}\|\mathcal{A}\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{\beta}{2}\sum_{i\in\mathcal{I}}\|\mathcal{P}_i\boldsymbol{x} - \hat{\boldsymbol{z}}_i\|^2. \tag{5}$$

TABLE I
COMPARISON OF THE EXECUTION TIME OF OUR IMPLEMENTATION OF EPLL WITH AND WITHOUT PROPOSED ACCELERATIONS. EXPERIMENT CONDUCTED ON 40 IMAGES EACH OF SIZE 481×321 IN DENOISING SETTING. PROFILING WAS CARRIED OUT USING MATLAB (R2014B) ON A PC WITH INTEL(R) CORE(TM) I7-4790K CPU @4.00GHZ AND 16 GB RAM. EXECUTION TIMES ARE REPORTED AS AVERAGE NUMBER OF SECONDS PER IMAGE (S) AND PERCENTAGE OF THE TOTAL TIME (%).

Step	Without accelerations		With the proposed accelerations	
(Gaussian selection)	38.27s	95 %	0.21s	67 %
(Patch estimation)	0.95s	2 %	0.04s	13 %
(Patch extraction)	0.43s	1 %	0.02s	6 %
(Patch reprojection)	0.15s	0 %	0.01s	4 %
Others	0.52s	1 %	0.03s	10 %
Total	40.32s		0.31s	

Algorithm 1 The five steps of an EPLL iteration [43]

for all $i \in \mathcal{I}$

$\tilde{\boldsymbol{z}}_i \leftarrow \mathcal{P}_i \hat{\boldsymbol{x}}$ (Patch extraction)

$k_i^\star \leftarrow \underset{1 \leqslant k_i \leqslant K}{\operatorname{argmin}} \; \log w_{k_i}^{-2} + \log \left| \boldsymbol{\Sigma}_{k_i} + \frac{1}{\beta} \mathrm{Id}_P \right| + \tilde{\boldsymbol{z}}_i^t \left(\boldsymbol{\Sigma}_{k_i} + \frac{1}{\beta} \mathrm{Id}_P \right)^{-1} \tilde{\boldsymbol{z}}_i$ (Gaussian selection)

$\hat{\boldsymbol{z}}_i \leftarrow \left(\boldsymbol{\Sigma}_{k_i^\star} + \frac{1}{\beta} \mathrm{Id}_P \right)^{-1} \boldsymbol{\Sigma}_{k_i^\star} \tilde{\boldsymbol{z}}_i$ (Patch estimation)

$\tilde{\boldsymbol{x}} \leftarrow \left(\sum_{i \in \mathcal{I}} \mathcal{P}_i^t \mathcal{P}_i \right)^{-1} \sum_{i \in \mathcal{I}} \mathcal{P}_i^t \hat{\boldsymbol{z}}_i$ (Patch reprojection)

$\hat{\boldsymbol{x}} \leftarrow \left(\mathcal{A}^t \mathcal{A} + \beta \sigma^2 \mathrm{Id}_N \right)^{-1} \left(\mathcal{A}^t \boldsymbol{y} + \beta \sigma^2 \tilde{\boldsymbol{x}} \right)$ (Image estimation)

Algorithm: Subproblem (5) corresponds to solving a linear inverse problem with a Tikhonov regularization, and has an explicit solution often referred to as Wiener filtering:

$$\hat{\boldsymbol{x}} = \left(\mathcal{A}^t \mathcal{A} + \frac{\beta \sigma^2}{P} \sum_{i \in \mathcal{I}} \mathcal{P}_i^t \mathcal{P}_i \right)^{-1} \left(\mathcal{A}^t \boldsymbol{y} + \frac{\beta \sigma^2}{P} \sum_{i \in \mathcal{I}} \mathcal{P}_i^t \hat{\boldsymbol{z}}_i \right), \tag{6}$$

where $\mathcal{P}_i^t \mathcal{P}_i$ is a diagonal matrix whose i-th diagonal element corresponds to the number of patches overlapping the pixel of index i. This number is a constant equal to P (assuming proper boundary conditions are used), which allows to split the computation into two steps *Patch reprojection* and *Image estimation* as shown in Alg. 1. Note that the step *Patch reprojection* is simply the average of all overlapping patches. In contrast, subproblem (4) cannot be obtained in closed form as it involves a term with the logarithm of a sum of exponentials. A practical solution proposed in [43] is to keep only the component $k_i^\star$ maximizing the likelihood for the given patch assuming it is a zero-mean Gaussian random vector with covariance matrix $\boldsymbol{\Sigma}_{k_i} + \frac{1}{\beta} \mathrm{Id}_P$. With this approximation, the solution of (4) is also given by Wiener filtering, and the resulting algorithm iterates the steps described in Alg. 1. The authors of [43] found that using $T = 5$ iterations, with the sequence $\beta = \frac{1}{\sigma^2}\{1, 4, 8, 16, 32\}$, for the initialization $\hat{\boldsymbol{x}} = \boldsymbol{y}$, provides relevant solutions in denoising contexts for a wide range of noise level values σ^2.

A. Complexity via eigenspace implementation

The algorithm summarized in the previous section may reveal cumbersome computations as it requires performing numerous matrix multiplications and inversions. Nevertheless, as the matrices $\boldsymbol{\Sigma}_k$ are known *prior* to any calculation, their eigendecomposition can be computed offline to improve the runtime. If we denote the eigendecomposition (obtained offline) of $\boldsymbol{\Sigma}_k = \boldsymbol{U}_k \boldsymbol{S}_k \boldsymbol{U}_k^t$, such that $\boldsymbol{U}_k \in \mathbb{R}^{P \times P}$ is unitary and $\boldsymbol{S}_k$ is diagonal with positive diagonal elements ordered in decreasing order, steps *Gaussian selection* and *Patch estimation* can be expressed in the space of coefficients $\boldsymbol{c}$ as

$$\left\{ \tilde{\boldsymbol{c}}_i^k \leftarrow \boldsymbol{U}_k^t \tilde{\boldsymbol{z}}_i \right\}_{\substack{k=1..K \\ i=1..N}} \qquad \mathcal{O}(NKP^2) \tag{7}$$

$$\left\{ k_i^\star \leftarrow \underset{1 \leqslant k \leqslant K}{\operatorname{argmin}} \; \iota^k + \sum_{j=1}^{P} \left(\log \nu_j^k + \frac{[\tilde{\boldsymbol{c}}_i^k]_j^2}{\nu_j^k} \right) \right\}_{i=1..N} \qquad \mathcal{O}(NKP) \tag{8}$$

$$\left\{ [\hat{\boldsymbol{c}}_i]_j \leftarrow \gamma_j^{k_i^\star} [\tilde{\boldsymbol{c}}_i^{k_i^\star}]_j \right\}_{\substack{j=1..P \\ i=1..N}} \qquad \mathcal{O}(NP) \tag{9}$$

$$\left\{ \hat{\boldsymbol{z}}_i \leftarrow \boldsymbol{U}_{k_i^\star} \hat{\boldsymbol{c}}_i \right\}_{i=1..N} \qquad \mathcal{O}(NP^2) \tag{10}$$

where $[\tilde{\boldsymbol{c}}]_j$ denotes the j-th entry of vector $\tilde{\boldsymbol{c}}$, $\iota^k = -2 \log w_k$, $\nu_j^k = [\boldsymbol{S}_k]_{jj} + \frac{1}{\beta}$, and $\gamma_j^k = [\boldsymbol{S}_k]_{jj} / \nu_j^k$ with $[\boldsymbol{S}_k]_{jj}$ the j-th entry on the diagonal of matrix $\boldsymbol{S}_k$. The complexity of each operation is indicated on its right and corresponds to the number of operations per iteration of the alternate optimization scheme. The steps *Patch extraction* and *Patch reprojection* share a complexity of $\mathcal{O}(NP)$. Finally, the complexity of step *Image estimation* depends on the transform $\mathcal{A}$. In many scenarios of interest, $\mathcal{A}^t \mathcal{A}$ can be diagonalized using a fast transform and the inversion of $\mathcal{A}^t \mathcal{A} + \beta \sigma^2 \mathrm{Id}_N$ can be performed efficiently in the transformed domain (since Id_N is diagonal in any orthonormal basis). For instance, it leads to $\mathcal{O}(N)$ operations for denoising or inpainting, and $\mathcal{O}(N \log N)$ for periodical deconvolutions or super-resolution problems, thanks to the fast Fourier transform (these are the settings we have adopted in this paper). If $\mathcal{A}$ cannot be easily diagonalized, this step can be performed using conjugate gradient (CG) method, as done in [43], at a computational cost that depends on the number of CG iterations (*i.e.*, on the conditioning of $\mathcal{A}^t \mathcal{A} + \beta \sigma^2 \mathrm{Id}_N$). In any case, as shown in the next section, this step has a complexity independent of P and K and is one of the faster operations in the image restoration problems considered in this paper.

B. Computation time analysis

In order to uncover the practical computational bottlenecks of EPLL, we have performed the following computational analysis. To identify clearly which part is time consuming, it is

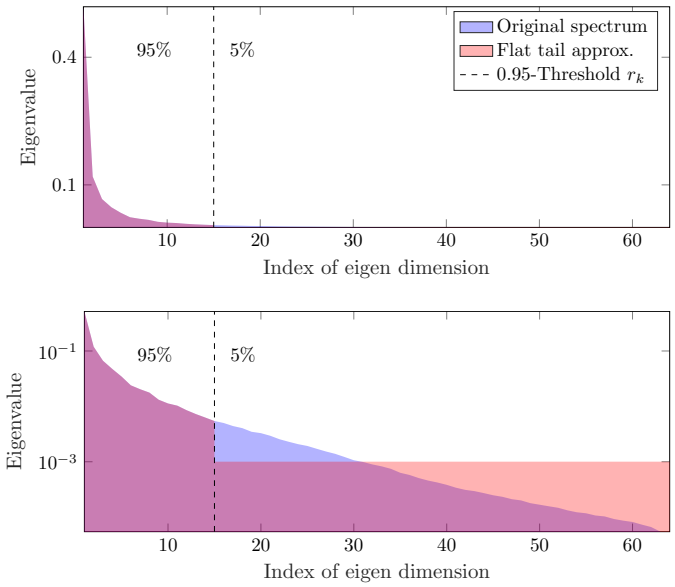


Fig. 1. Flat tail approximation: (a) with eigenvalues display on linear and (b) logarithmic scale. The $r_k = 15$ first eigen components explains $\rho = 95\%$ of the variability.

important to make the algorithm implementation as optimal as possible. Therefore, we refrain from using the MATLAB code provided by the original authors [43] for speed comparisons. Instead, we use a MATLAB/C version of EPLL based on the eigenspace implementation described above, where some steps are written in C language and interfaced using mex functions. This version, which we refer to as EPLLc, provides results identical to the original implementation while being 2-3 times faster. The execution time of each step for a single run of EPLLc is reported in the second column of Table I. Reported times fit our complexity analysis and clearly indicate that the Step *Gaussian selection* causes significant bottleneck due to $\mathcal{O}(NP^2K)$ complexity.

In the next section, we propose three independent modifications leading to an algorithm with a complexity of $\mathcal{O}(NP\bar{r}\log K/s^2)$ with two constants $1 \leqslant s^2 \leqslant P$ and $1 \leqslant \bar{r} \leqslant P$ that control the accuracy of the approximations introduced. The algorithm, in practice, is more than 100 times faster as shown by its runtimes reported in the third column.

III. Fast EPLL: the three key ingredients

We propose three accelerations based on (i) scanning only a (random) subset of the N patches, (ii) reducing the number of mixture components matched, and (iii) projecting on a smaller subspace of the covariance eigenspace. We begin by describing this latter acceleration strategy in the following paragraph.

A. Speed-up via flat tail spectrum approximation

To avoid computing the P coefficients of the vector $\tilde{c}_i^k$ in eq. (7), we rely on a flat-tail approximation. The k-th Gaussian model is said to have a flat tail if there exists a rank r_k such that for any $j > r_k$, the eigenvalues are constant: $[\boldsymbol{S}_k]_{j,j} = \lambda_k$.

Denoting by $\bar{\boldsymbol{U}}_k \in \mathbb{R}^{P\times r_k}$ (resp. $\bar{\boldsymbol{U}}_k^c \in \mathbb{R}^{P\times r_k^c}$) the matrix formed by the r_k first (resp. $r_k^c = P - r_k$ last) columns of $\boldsymbol{U}_k$, we have $\bar{\boldsymbol{U}}_k^c(\bar{\boldsymbol{U}}_k^c)^t = \mathrm{Id}_P - \bar{\boldsymbol{U}}_k\bar{\boldsymbol{U}}_k^t$. It follows

$$(\boldsymbol{\Sigma}_k + \tfrac{1}{\beta}\mathrm{Id}_P)^{-1} = \bar{\boldsymbol{U}}_k(\bar{\boldsymbol{S}}_k + \tfrac{1}{\beta}\mathrm{Id}_{r_k})^{-1}\bar{\boldsymbol{U}}_k^t + (\lambda_k + \tfrac{1}{\beta})^{-1}(\mathrm{Id}_P - \bar{\boldsymbol{U}}_k\bar{\boldsymbol{U}}_k^t), \quad (11)$$

$$(\boldsymbol{\Sigma}_k + \tfrac{1}{\beta}\mathrm{Id}_P)^{-1}\boldsymbol{\Sigma}_k = \bar{\boldsymbol{U}}_k(\bar{\boldsymbol{S}}_k + \tfrac{1}{\beta}\mathrm{Id}_{r_k})^{-1}\bar{\boldsymbol{S}}_k\bar{\boldsymbol{U}}_k^t + \lambda_k(\lambda_k + \tfrac{1}{\beta})^{-1}(\mathrm{Id}_P - \bar{\boldsymbol{U}}_k\bar{\boldsymbol{U}}_k^t), \quad (12)$$

where $\bar{\boldsymbol{S}}_k \in \mathbb{R}^{r_k\times r_k}$ is the diagonal matrix formed by the r_k first rows and columns of $\boldsymbol{S}_k$. Steps *Gaussian selection* and *Patch estimation* can thus be rewritten as

$$\left\{\tilde{\boldsymbol{c}}_i^k \leftarrow \bar{\boldsymbol{U}}_k^t\tilde{\boldsymbol{z}}_i\right\}_{\substack{k=1..K\\ i=1..N}} \qquad \mathcal{O}(NKP\bar{r}) \quad (13)$$

$$\left\{k_i^\star \leftarrow \underset{1\leqslant k\leqslant K}{\operatorname{argmin}}\ \iota_k + r_k^c\log\nu_P^k + \frac{\|\tilde{\boldsymbol{z}}_i\|^2}{\nu_P^k} + \sum_{j=1}^{r_k}\left(\log\nu_j^k + \frac{[\tilde{\boldsymbol{c}}_i^k]_j^2}{\nu_j^k} - \frac{[\tilde{\boldsymbol{c}}_i^k]_j^2}{\nu_P^k}\right)\right\}_{i=1..N} \qquad \mathcal{O}(NK\bar{r}) \quad (14)$$

$$\left\{[\hat{\boldsymbol{c}}_i]_j \leftarrow (\gamma_j^{k_i^\star} - \gamma_P^{k_i^\star})\left[\tilde{\boldsymbol{c}}_i^{k_i^\star}\right]_j\right\}_{\substack{j=1..r_{k_i^\star}\\ i=1..N}} \qquad \mathcal{O}(NP\bar{r}) \quad (15)$$

$$\left\{\hat{\boldsymbol{z}}_i \leftarrow \bar{\boldsymbol{U}}_{k_i^\star}\hat{\boldsymbol{c}}_i + \gamma_P^{k_i^\star}\tilde{\boldsymbol{z}}_i\right\}_{i=1..N} \qquad \mathcal{O}(NP\bar{r}) \quad (16)$$

where $\nu_P^k = \lambda_k + \frac{1}{\beta}$, $\gamma_P^k = \lambda_k/\nu_P^k$. As $\|\tilde{\boldsymbol{z}}_i\|^2$ can be computed once for all k, the complexity of each step is divided by $P/\bar{r}$, where $\bar{r} = \frac{1}{K}\sum_{k=1}^K r_k$ is the average rank after which eigenvalues are considered constant.

In practice, covariance matrices $\boldsymbol{\Sigma}_k$ are not flat-tail but can be approximated by a flat-tail matrix by replacing the lowest eigenvalues by a constant λ_k. To obtain a small value of $\bar{r}$ (hence a large speed-up), we preserve a fixed proportion $\rho \in (0, 1]$ of the total variability and replace the smallest eigenvalues accounting for the remaining $1-\rho$ fraction of the variability by their average (see Fig. 1): r_k is the smallest integer such that $\mathrm{Tr}(\bar{\boldsymbol{S}}_k) \geqslant \rho\,\mathrm{Tr}(\boldsymbol{S}_k)$. Choosing $\rho = 0.95$ means that 5% of the variability, in the eigendirections associated to the smallest eigenvalues, is assumed to be evenly spread in these directions. In practice, the choice of $\rho = 0.95$ leads to an average rank of $\bar{r} = 19.6$ (for $P = 8\times 8$) for a small drop of PSNR as shown in Fig. 4. Among several other covariance approximations that we tested, for instance, the one consisting in keeping only the r_k first directions, the flat tail approximation provided the best trade-off in terms of acceleration and restoration quality. The analyses showing the superiority of the proposed approximation over the more common approach of keeping only the first r_k directions, and the effect of ρ on image quality are included in the supplementary document.

B. Speed-up via a balanced search tree

As shown in Table I, the step *Gaussian selection* has a complexity of $\mathcal{O}(NP^2K)$, reduced to $\mathcal{O}(NPK\bar{r})$ using the flat tail spectrum approximation. This step remains the biggest bottleneck since each query patch has to be compared to all the K components of the GMM. To make this step even more efficient, we reduce its complexity using a balanced search

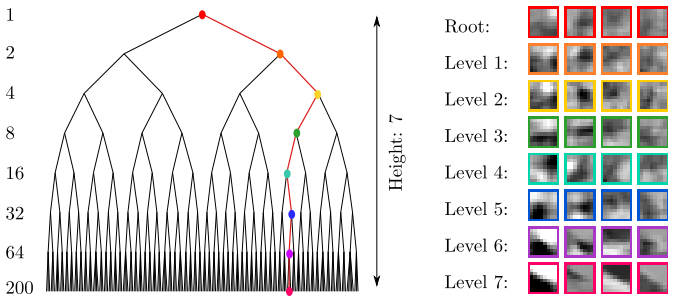
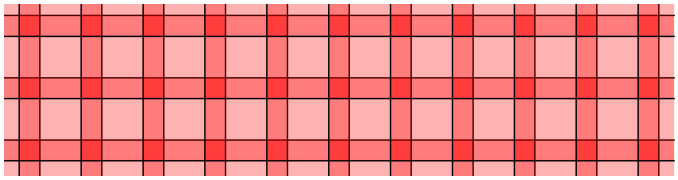
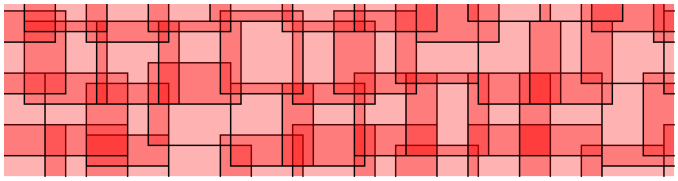


Fig. 2. (left) Our obtained search tree (with numbers of nodes for each level). (right) Four patches well represented by each of the eight nodes along the branch highlighted in red in the tree. These patches are randomly sampled from the generative model encoded at each node of the branch (patches on the same column are generated from the same random seed).



(a) Regular patch subsampling at (i_0, j_0)



(b) Stochastic patch subsampling at (i, j)

Fig. 3. Illustration of patch subsampling. Instead of extracting all patches, only a subset of patches is extracted either (a) regularly or (b) with some randomizations. Patches are represented by 8×8 squares and the red intensity represents the number of patches overlapping the corresponding pixel.

tree. As described below, such a tree can be built offline by adopting a bottom-up strategy that repeatedly collapses the original GMM to models with fewer components, until the entire model is reduced to a single Gaussian model.

We progressively combine the GMM components from one level to form the level above, by clustering the K components into $L < K$ clusters of similar ones, until the entire model is reduced to a single component. The similarity between two zero-mean Gaussian models with covariance $\mathbf{\Sigma}_1$ and $\mathbf{\Sigma}_2$ is measured by the symmetric Kullback-Leibler (KL) divergence

$$\mathrm{KL}(\mathbf{\Sigma}_1, \mathbf{\Sigma}_2) = \tfrac{1}{2}\,\mathrm{Tr}(\mathbf{\Sigma}_2^{-1}\mathbf{\Sigma}_1 + \mathbf{\Sigma}_1^{-1}\mathbf{\Sigma}_2 - 2\mathrm{Id}_P). \qquad (17)$$

Based on this divergence, at each level n, we look for a partition Ω^n of the K Gaussian components into L clusters (with about equal sizes) minimizing the following optimization problem

$$\underset{\Omega^n}{\mathrm{argmin}} \sum_{l=1}^{L} \sum_{k_1,k_2\in\Omega_l^n} \mathrm{KL}(\mathbf{\Sigma}_{k_1}, \mathbf{\Sigma}_{k_1}), \qquad (18)$$

such that $\bigcup_{l=1}^{L} \Omega_l^n = [K]$ and $\Omega_{l_1}^n \bigcap_{l_1\neq l_2} \Omega_{l_2}^n = \emptyset$, where Ω_l^n is the l-th set of Gaussian components for the GMM at level n. This clustering problem can be approximately solved using the genetic algorithm of [24] for the Multiple Traveling Salesmen Problem (MTSP). MTSP is a variation of the classical Traveling Salesman Problem where several salesmen visit a unique set of cities and return to their origins, and each city is visited by exactly one salesman. This attempts to minimize the total distance traveled by all salesmen. Hence, it is similar to our original problem given in eq. (18) where the Gaussian components and the clusters correspond to K cities and L salesmen, respectively. Given the clustering at level n, the new GMM at level $n-1$ is obtained by combining the zero-mean Gaussian components such that, for all $1 \leqslant l \leqslant L$:

$$w_l^{n-1} = \sum_{k\in\Omega_l^n} w_k^n \quad \text{and} \quad \mathbf{\Sigma}_l^{n-1} = \frac{1}{w_l^{n-1}} \sum_{k\in\Omega_l^n} w_k^n \mathbf{\Sigma}_k^n, \qquad (19)$$

where $\mathbf{\Sigma}_k^n$ and w_k^n are the corresponding covariance matrix and weight of the k-th Gaussian component at level n. Following this scheme, the original GMM of $K = 200$ components is collapsed into increasingly more compact GMMs with $K = 64$, 32, 16, 8, 4, 2 and 1 components. The main advantage of using MTSP compared to classical clustering approaches, is that this procedure can be adapted easily to enforce approximately equal sized clusters, simply by enforcing that each salesman visits at least 3 cities for the last level and 2 for the other ones.

We also experimented with other clustering strategies such as the hierarchical kmeans-like clustering in [17] and hierarchical agglomerative clustering. With no principled way to enforce even-sized clusters, these approaches, in general, lead to unbalanced trees (with comb structured branches) which result in large variations in computation times from one image to another. Although they all lead to similar denoising performances, we opted for MTSP based clustering to build our Gaussian tree in favor of obtaining a stable speed-up profile for our resulting algorithm. Please refer to the supplementary document for timing comparisons of MTSP vs. other tree building strategies.

In Fig. 2 we show that the tree obtained using MTSP-based clustering is almost a binary tree (left) and also display the types of patches it encodes along a given path (right). Such a balanced tree structure lets one avoid testing each patch against all K components. Instead, a patch is first compared to the two first nodes in level 1 of the tree, then the branch providing the smallest cost is followed and the operation is repeated at higher levels until a leaf has been reached. Using this balanced search tree reduces the complexity of step *Gaussian selection* to $\mathcal{O}(NP\bar{r}\log K)$.

C. Speed-up via the restriction to a random subset of patches

The simplest and most effective proposed acceleration consists of subsampling the set $\mathcal{I}$ of N patches to improve the complexity of the four most time-consuming steps, see Table I. One approach, followed by BM3D [8], consists of restricting the set $\mathcal{I}$ to locations on a regular grid with spacing $s \in [1, \sqrt{P}]$ pixels in both directions, leading to a reduction of complexity by a factor s^2. We refer to this approach as the regular patch subsampling. A direct consequence is that $|\mathcal{I}| = N/s^2$ and the complexity is divided by s^2. However, we observed that this strategy consistently creates blocky artifacts revealing the regularity of the extraction pattern. A random

sampling approach, called "jittering", used in the computer graphics community [7] is preferable to limit this effect. This procedure ensures that each pixel is covered by at least one patch. The location (i_0, j_0) of a point of the grid undergoes a random perturbation, giving a new location (i, j) such that

$$i_0 - \left\lfloor \frac{\sqrt{P}-s}{2} \right\rfloor \leqslant i \leqslant i_0 + \left\lfloor \frac{\sqrt{P}-s}{2} \right\rfloor$$
$$\text{and} \quad j_0 - \left\lfloor \frac{\sqrt{P}-s}{2} \right\rfloor \leqslant j \leqslant j_0 + \left\lfloor \frac{\sqrt{P}-s}{2} \right\rfloor, \qquad (20)$$

where $\lfloor \cdot \rfloor$ denotes the flooring operation. We found experimentally that independent and uniform perturbations offered the best performance in terms of PSNR and visual quality against all other tested strategies. In addition, we also resample these positions at each of the T iterations and add a (random) global shift to ensure that all pixels have the same expected number of patches covering them.

Figure 3 illustrates the difference between a regular grid and a jittered grid of period $s=6$ for patches of size $P=8\times 8$. In both cases, all pixels are covered by at least one patch, but the stochastic version reveals an irregular pattern.

Nevertheless, when using random subsampling, a major bottleneck occurs when $\mathcal{A}^t\mathcal{A}$ is not diagonal because the inversion involved in eq. (6) cannot be simplified as in Alg. 1. Using a conjugate gradient is a practical solution but will negate the reduction of complexity gained by using subsampling. To the best of our knowledge, this is the main reason why patch subsampling has not been utilized to speed up EPLL. Here, we follow a different path. We opt for approximating the solution of the original problem (involving all patches) rather than evaluating the solution of an approximate problem (involving random subsample of patches). More precisely, we speed up Alg. 1 by replacing the complete set of indices by the random subset of patches. In this case, step *Patch reprojection* consists of averaging only this subset of overlapping restored patches. This novel and nuanced idea avoids additional overhead and attains dramatic complexity improvements compared to the standard approach. Note that even in the case of some inverse problems, such as deblurring, super-resolution, inpainting and devignetting, this strategy can still be used in order to avoid conjugate gradient and maintain a large speed-up.

Experiments conducted on our validation dataset show that this strategy used with $s=6$ leads to an acceleration of about $36\times$ with less than a 0.2dB drop in PSNR. In comparison, for a similar drop of PSNR, the regular patch subsampling can only achieve a $9\times$ acceleration with $s=3$ (plots included in the supplementary).

D. Performance analysis

Figure 4 shows the image restoration performance and speed-up obtained when the three ingredients are applied separately or in combination. The results are averaged over 40 images from the test set of BSDS dataset [30] that is set aside for validation purposes. The speed-up is calculated with respect to the EPLLc implementation which is labeled "original" in Fig. 4. Among the three ingredients, random subsampling or jittering (labeled "subsampling") leads to the largest speed-up ($32\times$), while the usage of the search tree provides more than $7\times$ faster

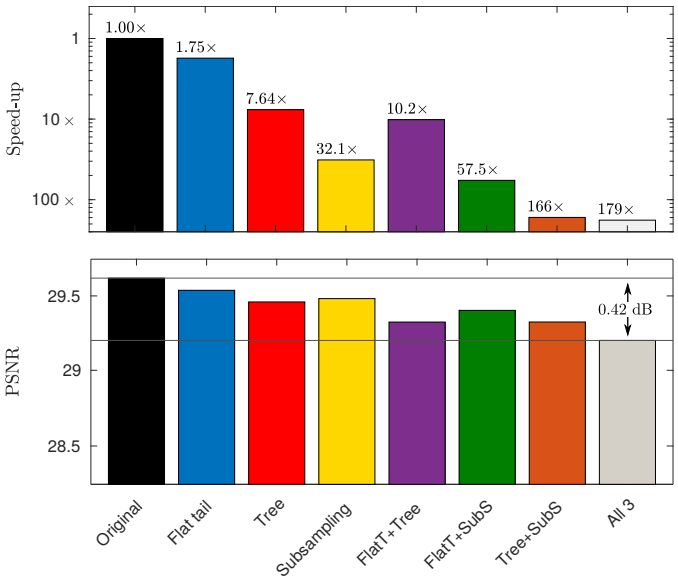


Fig. 4. (top) Average speed-up and (bottom) average PSNR for our three accelerations, and all possible combinations of them, on the 40 images of the BSDS validation set.

processing. The average speed-up obtained when combining all three ingredients is around $179\times$ on our validation set consisting of images of size 481×321, for an average drop of PSNR less than 0.5dB. Note that most of the runtime gain is achieved by combining binary search tree and random subsampling ($166\times$), while the flat tail acceleration leads to a modest acceleration of less than 10%. Nevertheless, we include it in our approach because it introduces another parameter (ρ) that addresses the bottleneck presented by the dimensionality (P^2) in a unique way. Adjusting ρ lets a practitioner choose an appropriate operating point suitable for their need in the speed-vs-quality trade-off space without losing too much in terms of quality.

IV. Related methods

To the best of our knowledge, there are only two other approaches [39], [35] that have attempted to accelerate EPLL. Unlike our approach, these methods focus on accelerating only one of the steps of EPLL namely the *Gaussian selection* step. Both use machine learning techniques to reduce its runtime.

In [39], the authors use a binary decision tree to approximate the mapping $\tilde{\boldsymbol{z}}_i \mapsto k_i^*$ performed in step *Gaussian selection*. At each node k and level n of the tree, the patch $\tilde{\boldsymbol{z}}_i$ is confronted with a linear separator in order to decide if the recursion should continue on the left or right child given by

$$\langle \boldsymbol{a}_k^n, \tilde{\boldsymbol{z}}_i \rangle + b_k^n \gtrless 0 \qquad (21)$$

where $(\boldsymbol{a}_k^n, b_k^n)$ are the parameters of the hyperplane for the k-th node at level n. These separators are trained offline on all pairs of $(\tilde{\boldsymbol{z}}_i, k_i^*)$ obtained after the first iteration of EPLL for a *given* β and noise level σ. Once a leaf has been reached, its index provides a first estimate for the index k_i^*. To reduce errors due to large variations among the neighboring pixels, this method further employs a Markov random fields on the resulting map of Gaussian components which runs in $\mathcal{O}(NK)$ complexity.

Hence, their overall approach reduces the complexity of step *Gaussian selection* from $\mathcal{O}(NKP^2)$ to $\mathcal{O}(N(PD+K))$, where $D = 12$ is the depth of the learned decision tree.

In [35], the authors approximate the *Gaussian selection* step, by using a gating (feed-forward) network with one hidden layer

$$\tilde{\boldsymbol{z}}_i \mapsto \iota^k + \sum_{j=1}^{Q} \left(\log \nu_j^k + \omega_j^k (\tilde{\boldsymbol{d}}_i)_j^2 \right) \quad \text{with} \quad \tilde{\boldsymbol{d}}_i = \boldsymbol{V}^t \tilde{\boldsymbol{z}}_i \quad (22)$$

where Q is the size of the hidden layer. The matrix $\boldsymbol{V} \in \mathbb{R}^{P \times Q}$ encodes the weights of the first layer, ω^k corresponds to the weights of the hidden layer and they are learned discriminatively to approximate the exact *posterior* probability:

$$\tilde{\boldsymbol{z}}_i \mapsto \iota^k + \sum_{j=1}^{P} \left(\log \nu_j^k + \frac{(\tilde{\boldsymbol{c}}_i^k)_j^2}{\nu_j^k} \right) \quad \text{with} \quad \tilde{\boldsymbol{c}}_i^k = \boldsymbol{U}_k^t \tilde{\boldsymbol{z}}_i \quad (23)$$

that we encounter in eq. (7) and (8). Theoretically, a new network will need to be trained for each type of degradations, noise levels and choices of β (recall that $\nu_j^k = (\boldsymbol{S}_k)_{jj} + \frac{1}{\beta}$). However, the findings of [35] indicate that applying a network learned on clean patches and with $\frac{1}{\beta} = 0$ is effective regardless of the type of degradation or the value of β. Their main advantage can be highlighted by comparing eq. (22) and (23) where complexity is reduced from $\mathcal{O}(NKP^2)$ to $\mathcal{O}(NQ(K + P))$. The authors utilize this benefit by choosing $Q = 100$.

Unlike these two approaches, our method does not try to learn the Gaussian selection rule directly (which depends on both the noise level through $1/\beta$ and the *prior* model through the GMM). Instead, we simply define a hierarchical organization of the covariance matrices $\boldsymbol{\Sigma}_k$. In other words, while the two other approaches try to infer the *posterior* probabilities (or directly the maximum *a posteriori*), our approach provides an approximation to the *prior* model. During runtime, this approximation of the *prior* is used in the *posterior* for the Gaussian selection task. Please note that the value of β does not play a role in determining the *prior*. This allows us to use the same search tree independently of the noise level, degradations, *etc.* Given that the main advantage of EPLL is that the same model can be used for any type of degradations, it is important that this property remains true for the accelerated version. Last but not least, the training of our search tree takes a few minutes while the training steps for the above mentioned approach take from several hours to a few days [39].

Apart from methods that accelerate EPLL, [20] and [19] are two works that use strategies sharing similarities to our proposed flat-tail and binary search tree approximations, respectively. In [20], the authors use a flat-tail GMM to model distribution of noisy patches. Since the GMM is learned directly on noisy patches, the constant value of the tail corresponds to the noise variance. This allows to improve inference when learning on noisy datasets, estimating the noise level, and retrieving the intrinsic dimension of each cluster. This is different from our approach which uses GMM priors learned on clean patch and provides flat-tail approximation using mean of least significant coefficients. In [19], the authors introduce a general data structure called *covariance tree* (CovTree). A CovTree is constructed by first building a binary space partition tree on patches (data points) and patches in each node are then modeled using a Gaussian distribution. While the resulting tree of Gaussians do share some similarities with our binary search tree, the learning process is very different. The covariance tree is built directly on data points, our approach is applied on an already learned GMM. In addition, unlike CovTree, our proposed strategy can handle any number of components (not just powers of two) and also encodes all parameters of mixture components including the mixing weights.

In the next section, we show that our proposed accelerations produce restoration results with comparable quality to competing methods while requiring a smaller amount of time.

V. Numerical experiments

In this section, we present the results obtained on various image restoration tasks. Our experiments were conducted on standard images of size 512×512 such as Barbara, Boat, Couple, Fingerprint, Lena, Mandrill and on 60 test images of size 481×321 from the Berkeley Segmentation Dataset (BSDS) [30] (the original BSDS test set contains 100 images, the other 40 was used for validation purposes while setting parameters ρ and s). For denoising, we compare the performance of our fast EPLL (FEPLL) to the original EPLL algorithm [43] and BM3D [8]. For the original EPLL, we have included timing results given by our own `MATLAB/C` implementation (EPLLc) and the `MATLAB` implementation provided by the authors (EPLLm). We also compare our restoration performance and runtime against other fast restoration methods introduced to achieve competitive trade-off between runtime efficiency and image quality. These methods include RoG [35] (a method accelerating EPLL based on feedforward networks described in Sec. IV), and CSF [36] (a fast restoration technique using random field-based architecture).

For deblurring experiments, we additionally compare with field-of-experts (FoE)-based non-blind deconvolution [5] denoted as iPiano. We contacted the corresponding author of [39] and got confirmation that the implementation of their algorithm (briefly described in IV) is not publicly available. Due to certain missing technical details, we were unable to reimplement it faithfully. However, the results reported in [39] indicate that their algorithm performs in par with BM3D in terms of both PSNR and time. Hence, BM3D results can be used as a faithful proxy for the expected performance of Wang et al.'s algorithm [39].

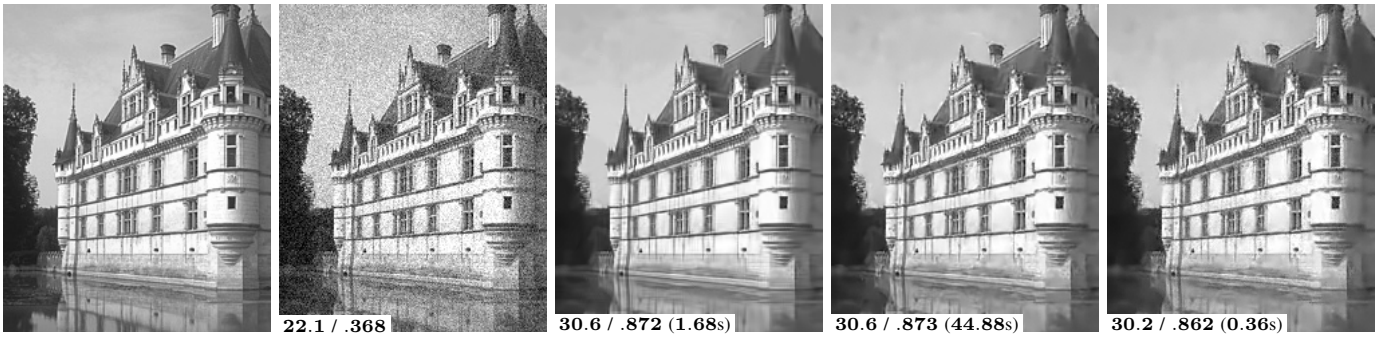
To explicitly illustrate the quality vs. runtime tradeoff of FEPLL, we include results obtained using a slightly slower version of FEPLL referred to as FEPLL$'$, that does not use the balanced search tree and uses a flat tail spectrum approximation with $\rho = 0.98$. Please note that FEPLL$'$ is not meant to be better or worse than FEPLL, it is just another version running at a different PSNR/time tradeoff which allows us to compare our algorithm to others operating in different playing fields.

Finally, to illustrate the versatility of FEPLL, we also include results for other inverse problems such as devignetting, super-resolution, and inpainting. Additional results can be found in the supplementary.

Parameter settings: In our experiments, we use patches of size $P = 8 \times 8$, and the GMM provided by Zoran *et*

TABLE II
PSNR, SSIM AND EXECUTION TIME ON THE BSDS TEST SET (AVERAGE ON 60 IMAGES OF SIZE 481×321), AND ON SIX STANDARD IMAGES (EACH OF SIZE 512×512 AND RESULTS AVERAGED OVER 10 INDEPENDENT NOISE REALIZATIONS) FOR THE PROPOSED FEPLL AND FEPLL′, EPLL (WITH TIMING GIVEN FOR BOTH EPLLM [43] AND OUR EPLLC), BM3D [8], CSF [36], ROG [35] AND DNCNN [41] WITH 3 DIFFERENT LEVELS OF NOISE.

σ	Algo.	Berkeley	Barbara	Boat	Couple	Fingerprint	Lena	Mandrill
		PSNR/SSIM						
5	FEPLL	36.8 /.959	36.9 /.958	36.6 /.930	36.6 /.944	35.4 /.984	38.2 /.941	35.1 /.959
	FEPLL′	37.1 /.962	37.1 /.959	36.7 /.933	36.8 /.946	35.5 /.984	38.3 /.943	35.2 /.960
	EPLL	37.3 /.963	37.6 /.962	36.8 /.933	37.3 /.950	36.4 /.987	38.6 /.944	35.2 /.960
	RoG	37.1 /.959	36.7 /.954	36.5 /.920	37.2 /.946	36.4 /.987	38.3 /.939	35.1 /.956
	BM3D	37.3 /.962	38.3 /.964	37.3 /.939	37.4 /.949	36.5 /.987	38.7 /.944	35.3 /.959
	$\text{CSF}_{3\times3}$	36.8 /.952	37.0 /.955	36.7 /.929	37.1 /.945	36.2 /.986	38.2 /.938	34.8 /.953
	DnCNN	34.5 /.915	35.5 /.937	34.4 /.878	34.5 /.893	32.9 /.968	36.3 /.907	32.3 /.899
20	FEPLL	29.1 /.812	29.0 /.852	30.2 /.802	29.9 /.813	27.6 /.908	32.3 /.863	26.4 /.784
	FEPLL′	29.3 /.831	29.5 /.866	30.3 /.814	30.1 /.825	27.8 /.916	32.4 /.864	26.5 /.802
	EPLL	29.5 /.836	29.8 /.872	30.6 /.821	30.4 /.834	28.3 /.924	32.6 /.869	26.7 /.807
	RoG	29.4 /.828	28.4 /.838	30.5 /.815	30.3 /.827	28.3 /.922	32.5 /.865	26.5 /.794
	BM3D	29.4 /.824	31.7 /.904	30.8 /.824	30.7 /.842	28.8 /.928	33.0 /.876	26.6 /.794
	$\text{CSF}_{3\times3}$	29.0 /.805	28.3 /.821	30.2 /.802	29.9 /.812	28.0 /.916	31.8 /.837	26.1 /.778
	DnCNN	30.0 /.847	31.0 /.896	31.1 /.831	31.0 /.848	28.8 /.930	33.4 /.883	27.0 /.815
60	FEPLL	24.5 /.614	23.6 /.636	25.5 /.644	25.1 /.629	22.2 /.722	27.4 /.742	21.4 /.468
	FEPLL′	24.5 /.620	23.8 /.648	25.5 /.646	25.1 /.636	22.4 /.745	27.3 /.733	21.6 /.499
	EPLL	24.8 /.631	24.0 /.660	25.8 /.659	25.4 /.649	22.6 /.755	27.6 /.747	21.7 /.506
	RoG	24.6 /.623	23.3 /.626	25.6 /.654	25.2 /.640	22.4 /.739	27.4 /.747	21.5 /.482
	BM3D	24.8 /.637	26.3 /.757	25.9 /.671	25.6 /.666	23.8 /.801	28.2 /.778	21.7 /.500
	$\text{CSF}_{3\times3}$	22.0 /.489	21.4 /.490	22.7 /.502	22.5 /.505	21.2 /.727	23.4 /.511	20.2 /.466
	DnCNN	25.3 /.665	24.9 /.717	26.4 /.686	26.0 /.683	23.1 /.781	28.5 /.790	22.1 /.555
		Time (in seconds)						
	FEPLL	0.27	0.38	0.38	0.37	0.36	0.38	0.38
	FEPLL′	0.96	1.28	1.28	1.28	1.26	1.27	1.28
	EPLLc	43.82	71.14	71.24	71.31	71.28	71.23	71.31
	EPLLm	82.68	145.15	143.68	144.30	144.13	144.15	143.86
	RoG	1.16	1.92	1.92	1.93	1.91	1.93	1.92
	BM3D	1.60	2.52	2.68	2.59	2.17	2.61	2.64
	$\text{CSF}_{3\times3}$	0.87	1.09	1.09	1.13	1.09	1.09	1.08
	$\text{CSF}^{\text{gpu}}_{3\times3}$	0.38	0.39	0.39	0.39	0.39	0.39	0.39
	DnCNN	2.09	3.57	3.55	3.60	3.60	3.60	3.61
	$\text{DnCNN}^{\text{gpu}}$	0.28	0.63	0.63	0.64	0.63	0.64	0.64



(a) Reference $\boldsymbol{x}$ (b) Noisy image $\boldsymbol{y}$ (c) BM3D result $\hat{\boldsymbol{x}}$ (d) EPLLc result $\hat{\boldsymbol{x}}$ (e) FEPLL result $\hat{\boldsymbol{x}}$

Fig. 5. Illustration of a denoising problem with noise standard deviation $\sigma = 20$. Part of: (a) the original image (b) its noisy version (c-e) denoised results of competitive methods with PSNR/SSIM and time (inset).

al. [43] with $K = 200$ components. The 200-components GMM is progressively collapsed into smaller GMMs with $K=64, 32, 16, 8, 4, 2$ and 1, and then all Gaussians of the tree are modified offline by flat-tail approximations with $\rho=0.95$. The final estimate for the restored image is obtained after 5 iterations of our algorithm with β set to $\lambda\sigma^{-2}\{1, 4, 8, 16, 32\}$ where $\lambda = \min\{N^{-1}\|\mathcal{A}^t\mathcal{A}\|_F^2\|\mathcal{A}\|_2^{-2}, 250\sigma^2\}$. For denoising, where $\mathcal{A}$ is identity, $\lambda = 1$ which boils down to the setting used by Zoran *et al.* [43]. For inverse problems, we found that the initialization $\hat{\boldsymbol{x}} = (\mathcal{A}^t\mathcal{A} + 0.2\sigma^2/\lambda\nabla)^{-1}\mathcal{A}^t\boldsymbol{y}$, with ∇ the image Laplacian, provides relevant solutions whatever the linear operator $\mathcal{A}$ and the noise level σ^2. While the authors of [43] do not provide any further direction for setting β and the initialization in general inverse problems, our proposed setting leads to competitive solutions irrespective of $\mathcal{A}$ and σ^2. For BM3D [8], EPLLm [43], RoG [35], CSF [36] and iPiano [5] we use the implementations provided by the original authors and use the default parameters prescribed by them.

Denoising: Table II shows the quantitative performances of FEPLL on the denoising task compared to EPLLm [43],

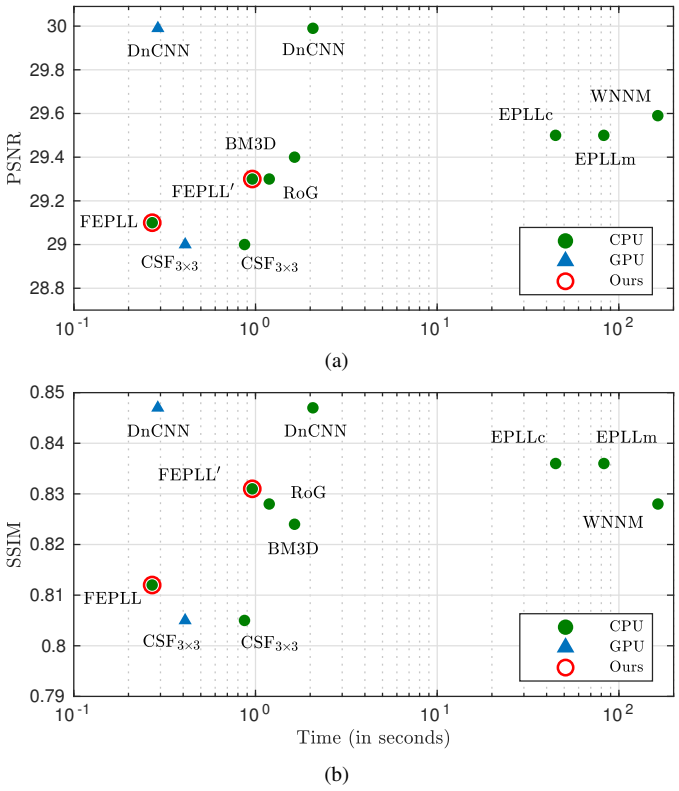


Fig. 6. (a) PSNR and (b) SSIM versus time for different restoration methods in a denoising problem with noise standard deviation $\sigma = 20$. PSNR, SSIM and time are averaged on the 60 BSDS test images, each of size 481×321. Optimal methods tend to be in the top-left corner.

EPLLc (our own `MATLAB/C` implementation), RoG [35] BM3D [8], CSF [36] and DnCNN [42]. We evaluate the algorithms under low-, mid- and high-noise settings by using Gaussian noise of variance 5^2, 20^2 and 60^2, respectively. The result labeled "Berkeley" is an average over 60 images from the BSDS testing set [30]. All numbers are averages obtained on 10 independent noise realizations. Figures 6 provide graphical representations of these performances in terms of PSNR/SSIM versus computation time for the BSDS images for the noise variance setting $\sigma^2 = 20^2$. In this figure, we have also included a recent approach based on weighted nuclear norm minimization (WNNM) [18]. On average, FEPLL results are 0.5dB below regular EPLL and BM3D; however, FEPLL is approximately 7 times faster than BM3D, 170-200 times faster than EPLLc and over 350 times faster than EPLLm. FEPLL outperforms the faster CSF algorithm in terms of both PSNR and time. In this case, FEPLL is even faster than the GPU accelerated version of CSF (CSF_{gpu}). Our approach is 4 times faster than RoG with a PSNR drop of 0.1-0.3dB. Nevertheless, if we slow down FEPLL to FEPLL′, we can easily neutralize this quality deficit while still being faster than RoG. While DnCNN offers better results, DnCNN (CPU) is about $10\times$ slower than FEPLL (also CPU). Our FEPLL (CPU) is also slightly faster than DnCNN (GPU). WNNM offers comparable results to FEPLL' but is about $500\times$ slower. Note that these accelerations are obtained purely based on the approximations and *no* parallel processing is used. Also, in most cases, a loss of 0.5dB may not affect the visual quality of the image. To illustrate this, we show a sample image denoised by BM3D, EPLL and FEPLL in Fig. 5.

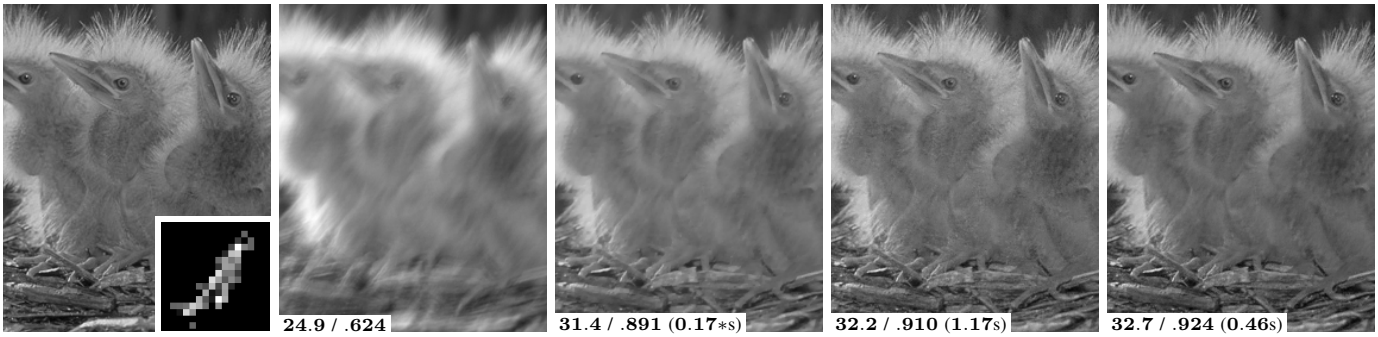
TABLE III
PSNR, SSIM AND EXECUTION TIME ON THE BSDS TEST SET (AVERAGE OF 60 IMAGES OF SIZE 481×321), AND ON STANDARD IMAGES (AVERAGE OF 6 IMAGES OF SIZE 512×512) FOR THE PROPOSED FEPLL AND FEPLL′, CSF [36], ROG [35] AND CHEN ET AL.'S [5] METHOD THAT IS CALLED *iPiano* IN THEIR IMPLEMENTATION. THE BLUR KERNEL USED IS THE ONE PROVIDED IN ALONG WITH IPIANO IMPLEMENTATION AND NOISE IS SET TO $\sigma = 0.5$.

Algo.	Berkeley		Classic	
	PSNR/SSIM	Time (s)	PSNR/SSIM	Time (s)
iPiano	29.5 / .824	29.53	29.9 / .848	59.10
CSF_{pw}	30.2 / .875	0.50 (0.14*)	30.5 / 0.870	0.47 (0.14*)
RoG	31.3 / .897	1.19	31.8 / .915	2.07
FEPLL	33.1 / .928	0.40	32.8 / .931	0.46
FEPLL'	33.2 / .930	1.01	33.0 / .933	1.82

Deblurring: Table III shows the performance of FEPLL when used for deblurring as compared to RoG [35], iPiano [5] and CSF [36]. Once again, FEPLL uses Alg. 1 with operator $\mathcal{A}$ in the *Image estimation* step defined by $\mathcal{A}\boldsymbol{x} = \mathcal{F}^{-1}[\mathcal{F}[\boldsymbol{h}] \odot \mathcal{F}[\boldsymbol{x}]]$, where $\mathcal{F}$ is the fast Fourier transform and $\mathcal{F}^{-1}$ its inverse, $\odot$ indicates element-wise product and $\boldsymbol{h}$ is the blur kernel. For these experiments, we use the blur kernel provided by Chen et al. [5] along with their algorithm implementation. The results under the label "Berkeley" are averaged over 60 images from the BSDS test dataset [30]. The results labeled "Classic" is averaged over the six standard images (Barbara, Boat, Couple, Fingerprint, Lena and Mandrill). FEPLL consistently outperforms its efficient competitors both in terms of quality and runtime. Although the GPU version of CSF is faster, the restoration quality obtained by CSF is 2-3dB lower than FEPLL. The proposed algorithm outperforms RoG by 1-1.8dB while running 3 and 5 times faster on "Berkeley" and "Classic" datasets, respectively.

The superior qualitative performance of FEPLL is demonstrated in Fig. 7. For brevity, we only include the deblurring results obtained from the top competitors of FEPLL algorithm in terms of both quality and runtime. As observed, FEPLL provides the best quality vs. runtime efficiency trade-off. In contrast, a deblurring procedure using the regular EPLL is around 350 times slower than FEPLL with the original implementation [43]. Specifically, on the sample image shown in Fig. 7, EPLL provides a qualitatively similar result (not shown in the figure) with a PSNR of 32.7 dB and SSIM of 0.922 in 142 seconds.

Other inverse problems: Unlike BM3D, EPLL and FEPLL are more versatile and handle a wide range of inverse problems without any change in formulation. In Fig. 8, we show the results obtained by FEPLL on problems such as (a) devignetting, which involves a progressive loss of intensity, (b) super-resolution and (c) inpainting. To show the robustness of our method, the input images of size 481×321 were degraded with zero-mean Gaussian noise with $\sigma = 2$. All of the restoration results were obtained within/under 0.4 seconds and with the same set of parameters explained above (cf. *Parameter settings*).



(a) Reference $\boldsymbol{x}$ / Blur kernel (b) Blurry image $\boldsymbol{y}$ (c) CSF_{pw} result $\hat{\boldsymbol{x}}$ (d) RoG result $\hat{\boldsymbol{x}}$ (e) FEPLL result $\hat{\boldsymbol{x}}$

Fig. 7. Illustration of a deblurring problem with noise standard deviation $\sigma = 0.5$. Part of: (a) the original image and the blur kernel (inset), (b) blurry version (c-e) deblurred results of competitive methods with PSNR/SSIM and time (inset). The '*' indicates runtime on GPU while others are CPU times.

VI. Conclusion

In this paper, we accelerate EPLL by a factor greater than 100 with negligible loss of image quality (less than 0.5dB). This is achieved by combining three independent strategies: a flat tail approximation, matching via a balanced search tree, and stochastic patch sampling. We show that the proposed accelerations are effective in denoising and deblurring problems, as well as in other inverse problems such as super-resolution and devignetting. An important distinction of the proposed accelerations is their genericity: the accelerated EPLL prior can be applied to many restoration tasks and various signal-to-noise ratios, in contrast to existing accelerations based on learning techniques applied to specific conditions (such as image size, noise level, blur kernel, etc.) and that require an expensive re-training to address a different problem.

Since the speed-up is achieved solely by reducing the algorithmic complexity, we believe that further inclusion of accelerations based on parallelization and/or GPU implementations will allow for real-time video processing. Moreover, the acceleration techniques introduced in this work are general strategies that can be used to speed up other image restoration and/or related machine learning algorithms. For reproducibility purposes, the code of FEPLL is made available on GitHub[1].

References

[1] M. Aharon, M. Elad, and A. Bruckstein. k -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, Nov 2006.

[2] A. Buades, B. Coll, and J.-M. Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation*, 4(2):490–530, 2005.

[3] H. Burger, C. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[4] N. Cai, Y. Zhou, S. Wang, B. W.-K. Ling, and S. Weng. Image denoising via patch-based adaptive gaussian mixture prior method. *Signal, Image and Video Processing*, 10(6):993–999, 2016.

[5] Y. Chen, R. Ranftl, and T. Pock. Insights into analysis operator learning: From patch-based sparse models to higher order mrfs. *IEEE Transactions on Image Processing*, 23(3):1060–1072, March 2014.

[6] Y. Chen, W. Yu, and T. Pock. On learning optimized reaction diffusion processes for effective image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5261–5269, 2015.

[1]https://goo.gl/xjqKUA

[7] R. L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)*, 5(1):51–72, 1986.

[8] 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(8):2080–2095, August 2007.

[9] A. Danielyan, A. Foi, V. Katkovnik, and K. Egiazarian. Image upsampling via spatially adaptive block-matching filtering. In *Signal Processing Conference, 2008 16th European*, pages 1–5. IEEE, 2008.

[10] A. Danielyan, A. Foi, V. Katkovnik, and K. Egiazarian. Spatially adaptive filtering as regularization in inverse imaging: Compressive sensing super-resolution and upsampling. *Super-Resolution Imaging*, pages 123–154, 2010.

[11] C.-A. Deledalle, J. Salmon, and A. S. Dalalyan. Image denoising with patch based PCA: local versus global. In *BMVC 2011-22nd British Machine Vision Conference*, pages 25–1. BMVA Press, 2011.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[13] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016.

[14] K. Egiazarian and V. Katkovnik. Single image super-resolution via BM3D sparse coding. In *Signal Processing Conference (EUSIPCO), 2015 23rd European*, pages 2849–2853. IEEE, 2015.

[15] D. Geman and C. Yang. Nonlinear image recovery with half-quadratic regularization. *IEEE Transactions on Image Processing*, 4(7):932–946, 1995.

[16] D. Glasner, S. Bagon, and M. Irani. Super-resolution from a single image. In *IEEE 12th International Conference on Computer Vision*, 2009.

[17] J. Goldberger and S. T. Roweis. Hierarchical clustering of a mixture model. In *NIPS*, volume 17, pages 505–512, 2004.

[18] S. Gu, L. Zhang, W. Zuo, and X. Feng. Weighted nuclear norm minimization with application to image denoising. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2862–2869, 2014.

[19] T. Guillemot, A. Almansa, and T. Boubekeur. Covariance Trees for 2D and 3D Processing. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2014.

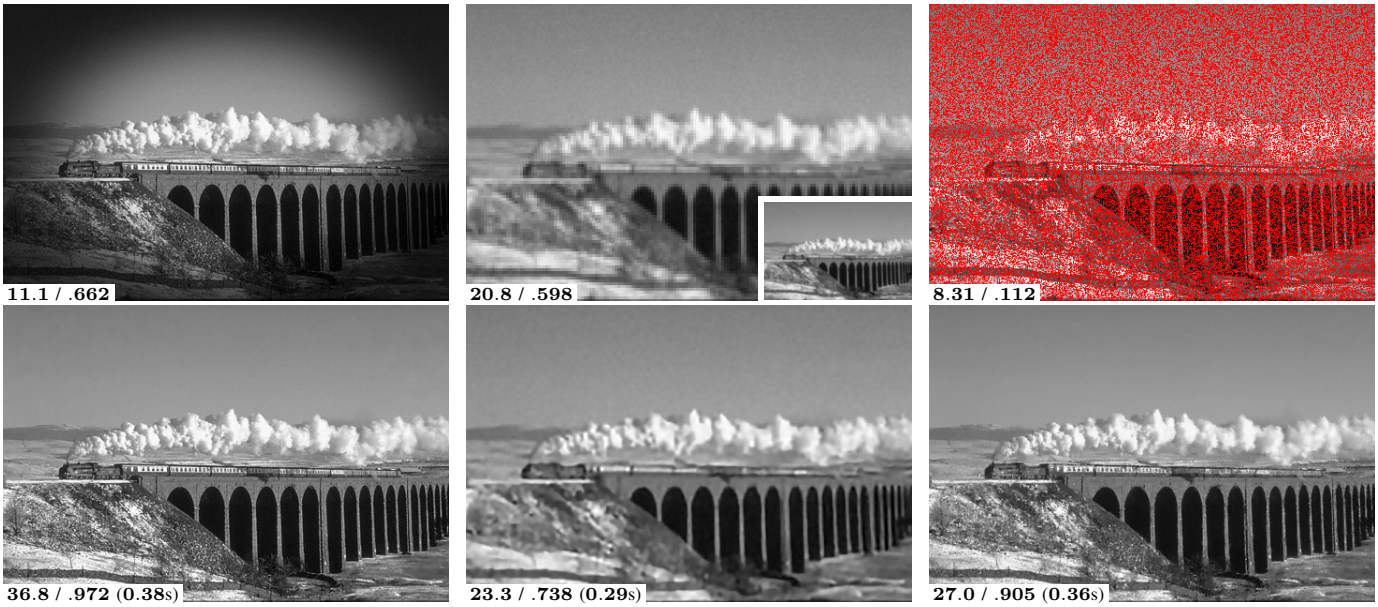
[20] A. Houdard, C. Bouveyron, and J. Delon. High-Dimensional Mixture Models For Unsupervised Image Denoising (HDMI). Preprint hal-01544249, Aug. 2017.

[21] J. Jancsary, S. Nowozin, and C. Rother. Regression tree fields – an efficient, non-parametric approach to image labeling problems. In *25th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[22] V. Katkovnik and K. Egiazarian. Nonlocal image deblurring: Variational formulation with nonlocal collaborative L0-norm prior. In *Local and Non-Local Approximation in Image Processing, 2009. LNLA 2009. International Workshop on*, pages 46–53. IEEE, 2009.

[23] J. Kim, J. Kwon Lee, and K. Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1646–1654, 2016.

[24] J. Kirk. Source code for "multiple traveling salesmen problem - genetic algorithm", https://fr.mathworks.com/matlabcentral/fileexchange/



(a) devignetting (b) $\times 3$ super-resolution (c) 50% inpainting

Fig. 8. FEPLL on various inverse problems. All inputs contain Gaussian noise with $\sigma = 2$. Top row: (a) the observation in a devignetting problem, (b) the bi-cubic interpolation and the actual low-resolution size image (inset) in a $\times 3$ super-resolution problem and (c) the observation in an inpainting problem with 50% of missing pixels shown in red. Bottom row: respective FEPLL results all obtained in less than 0.4s.

19049-multiple-traveling-salesmen-problem-genetic-algorithm. MATLAB Central File Exchange. Retrieved May 6, 2014, 2008.

[25] D. Krishnan and R. Fergus. Fast image deconvolution using hyper-laplacian priors. In *Advances in Neural Information Processing Systems*, pages 1033–1041, 2009.

[26] M. Lebrun, A. Buades, and J.-M. Morel. A nonlocal bayesian image denoising algorithm. *SIAM Journal on Imaging Sciences*, 6(3):1665–1688, 2013.

[27] E. López-Rubio. Superresolution from a single noisy image by the median filter transform. *SIAM Journal on Imaging Sciences*, 9(1):82–115, 2016.

[28] E. Luo, S. H. Chan, and T. Q. Nguyen. Adaptive image denoising by mixture adaptation. *IEEE Transactions on Image Processing*, 25(10):4489–4503, 2016.

[29] J. Mairal, F. Bach, and J. Ponce. Task-driven dictionary learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):791–804, 2012.

[30] 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 *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 416–423. IEEE, 2001.

[31] V. Papyan and M. Elad. Multi-scale patch-based image restoration. *IEEE Transactions on image processing*, 25(1):249–261, 2016.

[32] M. Protter, M. Elad, H. Takeda, and P. Milanfar. Generalizing the nonlocal-means to super-resolution reconstruction. *IEEE Transactions on image processing*, 18(1):36–51, 2009.

[33] Y. Ren, Y. Romano, and M. Elad. Example-based image synthesis via randomized patch-matching. *arXiv preprint arXiv:1609.07370*, 2016.

[34] J. Rick Chang, C.-L. Li, B. Poczos, B. Vijaya Kumar, and A. C. Sankaranarayanan. One network to solve them all–solving linear inverse problems using deep projection models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5888–5897, 2017.

[35] D. Rosenbaum and Y. Weiss. The return of the gating network: combining generative models and discriminative training in natural image priors. In *Advances in Neural Information Processing Systems*, pages 2683–2691, 2015.

[36] U. Schmidt and S. Roth. Shrinkage fields for effective image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2774–2781, 2014.

[37] A. Singh, F. Porikli, and N. Ahuja. Super-resolving noisy images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2846–2853, 2014.

[38] J. Sulam and M. Elad. Expected patch log likelihood with a sparse prior. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 99–111. Springer, 2015.

[39] Y. Wang, S. Cho, J. Wang, and S.-F. Chang. Discriminative indexing for probabilistic image patch priors. In *European Conference on Computer Vision*, pages 200–214. Springer, 2014.

[40] 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(5):2481–2499, 2012.

[41] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *arXiv preprint arXiv:1608.03981*, 2016.

[42] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[43] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *International Conference on Computer Vision*, pages 479–486. IEEE, November 2011.

Supplementary for "Accelerating GMM-based patch priors for image restoration: Three ingredients for a 100× speed-up"

Shibin Parameswaran*, Charles-Alban Deledalle*†,
Loïc Denis‡ and Truong Q. Nguyen*.

August 8, 2018

This document gives additional results and analysis of FEPLL algorithm described on "Accelerating GMM-based patch priors for image restoration: Three ingredients for a 100× speed-up".

1 Additional analysis

Flat tail approximation based acceleration First, we demonstrate the superiority of the proposed flat tail approximation over the commonly used approach of ignoring the least significant eigendirections. In Supplemental Figure 1, we show that the naïve approach of zeroing out (or ignoring) coefficients of the least significant eigendirections is inferior to the proposed approach of replacing these coefficients by the mean. Our approach can provide reasonably good performance even for very low values of ρ. For example, when $\rho = 0.20$ our approach leads to a PSNR of 27.5 compared to 25.1 obtained by the zeroing out strategy.

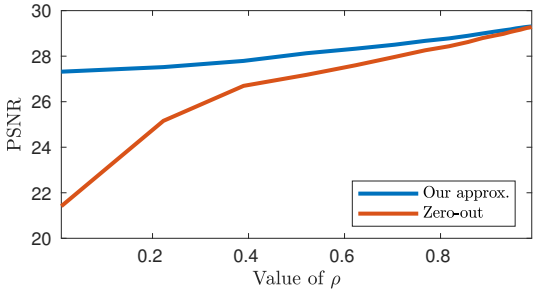
In a second experiment, we analyze the influence of the parameter $\rho \in (0, 1]$ used in the flat tail approximation. Supplemental Figure 2 shows the curve of PSNR as a function of speed-up for varying values of ρ. This experiment is repeated twice either with or without enabling the other two accelerations. Visual results highlight that as ρ decreases to zero, residual noise starts appearing around salient structures. From these experiments, the choice of $\rho = 0.95$ leads to a good trade-off in terms of speed and visual quality for a drop of PSNR lower than 0.2dB.

Search tree acceleration In the second supplementary experiment, we analyze the influence of the clustering method used to successively collapse the GMM model at a given level into a smaller model at a lower level. We compare the proposed approach based on Multiple Traveling Salesmen Problem [2] (MTSP) using the symmetric KL divergence as a semi-metric to a hierarchical GMM clustering algorithm similar to K-means based on KL divergence as proposed in [1] and the standard hierarchical agglomerated clustering (HAC) using symmetric KL divergence. Supplemental Figure 3 shows that MTSP leads to a well-balanced tree, with a height of 7 (almost a binary tree except for the last level due to 200 mixture components). In comparison, using [1]'s K-Means inspired algorithm provides a tree of height 7 but is not balanced and HAC leads to a tree of height 59 with comb structured branches. Please note that a shorter tree is preferred for a faster Gaussian selection step. The tree built using the proposed MTSP strategy not only leads to better PSNR/SSIM, but also provides a more stable computation time for all images of a fixed size (irrespective of content) due to its balanced structure. This is confirmed in Supplemental Figure 4 that displays box-plots

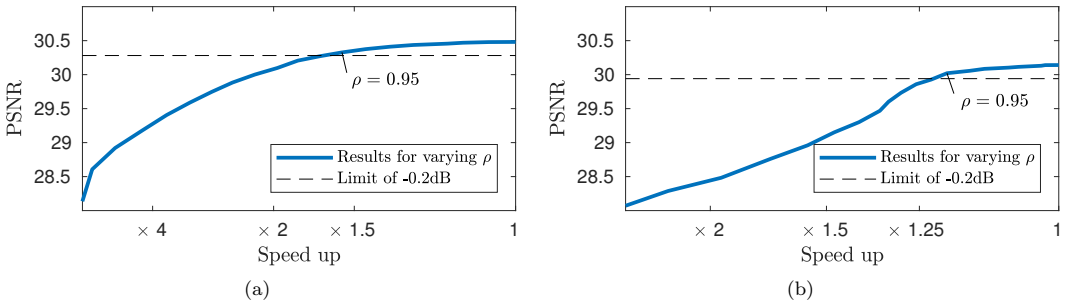
*Department of Electrical and Computer Engineering, University of California, San Diego, 9500 Gilman Dr, La Jolla, CA 92093, e-mail: sparames@ucsd.edu, cdeledalle@eng.ucsd.edu and tqn001@eng.ucsd.edu.

†IMB, CNRS, Univ. Bordeaux, Bordeaux INP, F-33405 Talence, France, e-mail: charles-alban.deledalle@math.u-bordeaux.fr

‡Univ Lyon, UJM-Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, Saint-Etienne, France, e-mail: loic.denis@univ-st-etienne.fr.



Supplemental Figure 1: Comparison between the naïve approach of zeroing out (ignoring) coefficients of the least significant eigendirections versus the proposed flat-tail approximation approach of replacing the coefficients with mean. Performance is compared in terms of PSNR obtained while denoising with different values of ρ (percent energy captured) used for thresholding singular values.





(c) Noisy (d) $\rho = 0.5$ (e) $\rho = 0.8$ (f) $\rho = 0.95$ (g) $\rho = 1$

Supplemental Figure 2: Top: PSNR as a function of speed-up for varying values of the proportion ρ used in the flat tail approximation when (a) no other accelerations are used and (b) the other two accelerations are also used. The parameter offering the best speed-up for a drop of at most -0.2dB is indicated. Bottom: (c) a noisy image with $\sigma = 20$ and (d-g) FEPLL results obtained for increasing value of ρ (includes the other two accelerations as well).

obtained from computation time statistics based on five independent runs of the algorithm on 40 different images of the BSDS dataset (all images have the same size). In contrast to our MTSP based strategy, the trees built using [1]'s method and HAC strategy lead to computation times that vary drastically depending on the image content.

Stochastic patch sub-sampling The last analysis focuses on the influence of the period $1 \leqslant s \leqslant \sqrt{P}$ used in the stochastic patch sub-sampling. In addition, we also compare against regular sub-sampling. Supplemental Figure 5 shows the curve of PSNR as a function of speed-up for varying values of s where $P = 8$. The experiment has been performed in both cases where either the other two accelerations are disabled or enabled. Visual results highlight that as s increases to $\sqrt{P}$, blocky

artifacts start appearing especially when using a regular patch sub-sampling. In comparison, the stochastic sub-sampling leads to competing results even for $s = \sqrt{P}$, which corresponds to a regular grid that is globally shifted by a random shift at each iteration of FEPLL. These experiments show that the choice of $s = 6$ leads to good trade-off in terms of speed and visual quality for a PSNR drop of less than 0.2dB.

2 Additional results

Denoising results As the first set of additional results, we provide the visual results obtained on five of the standard images: Barbara, Boat, Couple, Fingerprint, Lena and Mandrill, in the setting $\sigma = 20$. Results are displayed in Supplemental Figure 6.

Deconvolution results As the second set additional results, we show deconvolution results for five of testing images from the BSDS dataset [3] with a Gaussian blur setting of 3 pixels (standard deviation 3) and noise level $\sigma = 2$. Results are displayed in Supplemental Figure 7.

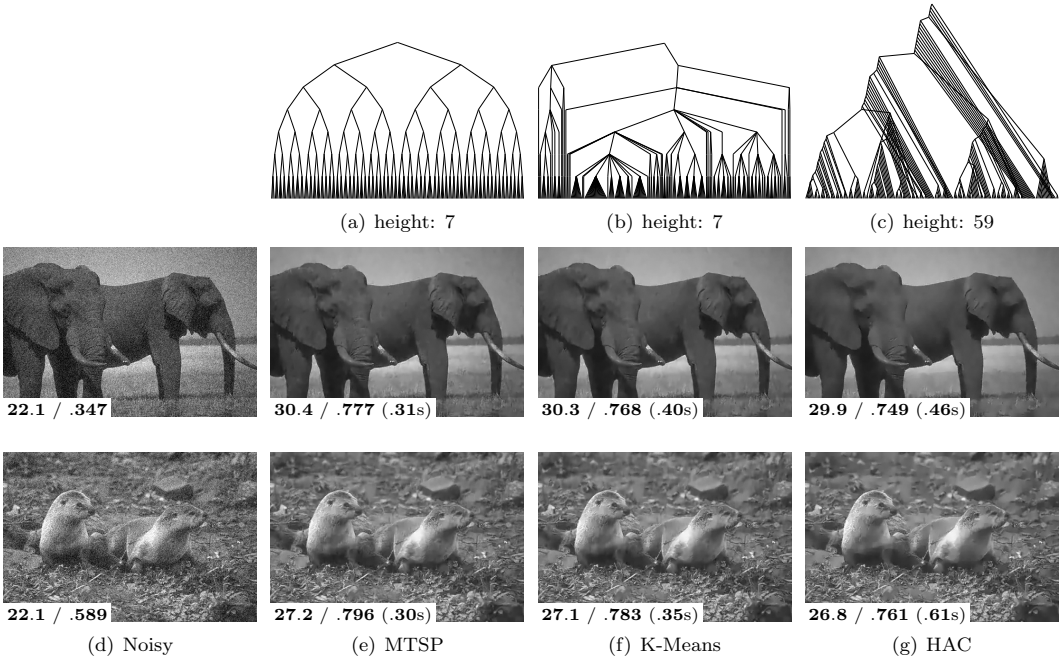
Super-resolution results As the last set of additional results, super-resolution results are given for five of the testing images from the BSDS dataset [3] for super-resolution by a factor of 3 with a noise level of $\sigma = 2$. Note that the sub-sampling operator also includes a small Gaussian blur of 0.5 pixels as well as a Kaiser window apodization for anti-aliasing. See Supplemental Figure 8.

References

[1] J. Goldberger and S. T. Roweis. Hierarchical clustering of a mixture model. In *NIPS*, volume 17, pages 505–512, 2004.

[2] J. Kirk. Source code for "multiple traveling salesmen problem - genetic algorithm", `https://fr.mathworks.com/matlabcentral/fileexchange/19049-multiple-traveling-salesmen-problem-genetic-algorithm`. MATLAB Central File Exchange. Retrieved May 6, 2014, 2008.

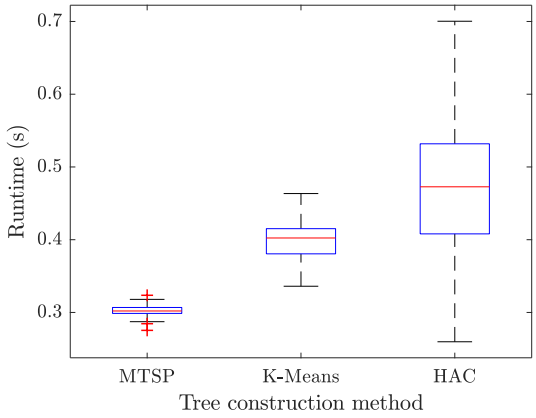
[3] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 416–423. IEEE, 2001.



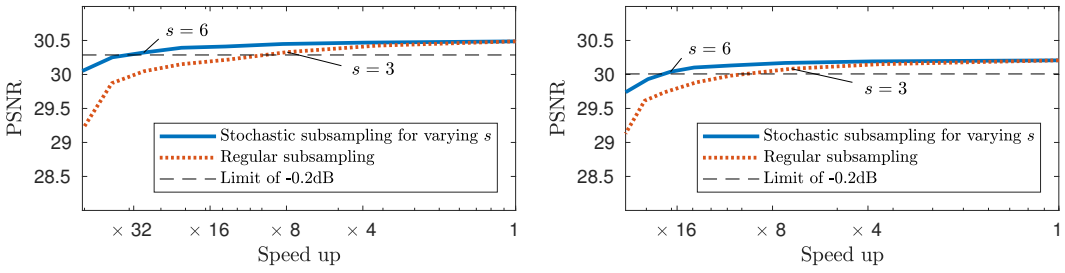
(a) height: 7 (b) height: 7 (c) height: 59

(d) Noisy (e) MTSP (f) K-Means (g) HAC

Supplemental Figure 3: Influence of clustering techniques on the search tree and the results. (left) Multiple Traveling Salesmen Problem [2] (MTSP), (center) Hierarchical K-means like clustering proposed in [1] and (right) Hierarchical agglomerated clustering (HAC). (a-c) The corresponding trees. (d-i) The corresponding results on two denoising problems with $\sigma = 20$ involving two different images of size 481×321. PSNR/SSIM and time are indicated for each result.



Supplemental Figure 4: Timing scatter obtained over 5 runs for 40 images using Gaussian trees built using different clustering methods we tested. "MTSP" refers to the Multiple Traveling Salesman (proposed), "K-Means" is the method introduced by [1] and "HAC" is a simple Hierarchical agglomerative clustering approach.



(a) without the other accelerations

(b) with the other accelerations



(c) Reference (d) Stochastic $s = 2$ (e) Stochastic $s = 4$ (f) Stochastic $s = 6$ (g) Stochastic $s = 8$



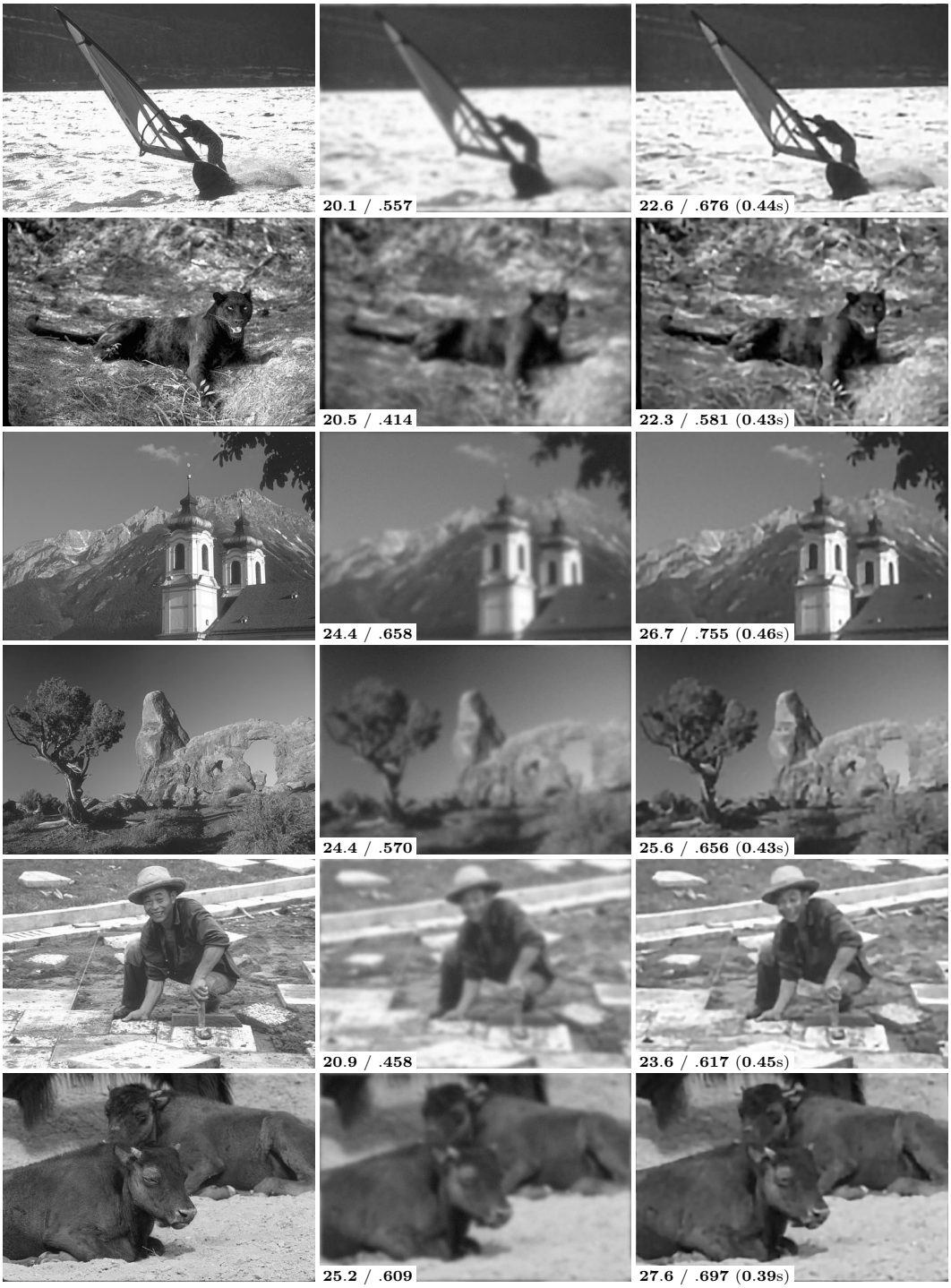
(h) Noisy (i) Regular $s = 2$ (j) Regular $s = 4$ (k) Regular $s = 6$ (l) Regular $s = 8$

Supplemental Figure 5: PSNR as a function of speed-up for varying values of (top) the patch extraction expected period s when (a) no other accelerations are used and (b) the other two accelerations are also used. Stochastic and regular sub-sampling are compared. The parameter offering the best speed-up within a drop of at most -0.2dB is indicated. (c) A reference image, (h) its noisy version with $\sigma = 20$ and results obtained for increasing value of s (including the other two accelerations as well) for (d-g) the stochastic version and (i-l) the regular version.



(a) Reference (b) Noisy (c) FEPLL (d) EPLLc (e) BM3D

Supplemental Figure 6: Denoising results where $\sigma = 20$ obtained on five standard images, from top to bottom: Barbara, Boat, Couple, Fingerprint, Lena and Mandrill.



(a) Reference image $\boldsymbol{x}$ (b) Blurry image $\boldsymbol{y}$ (c) Deblurred result $\hat{\boldsymbol{x}}$

Supplemental Figure 7: Illustration of a deblurring problem of a Gaussian convolution of width 3 pixels with a noise of standard deviation $\sigma = 2$. (a) The original high-resolution (HR) image $\boldsymbol{x}$. (b) The low-resolution (LR) image $\boldsymbol{y} = \mathcal{A}\boldsymbol{x} + \boldsymbol{w}$ and its bicubic interpolation. (c) The super-resolution result $\hat{\boldsymbol{x}}$ obtained by our Fast EPLL.



(a) Reference HR image $\boldsymbol{x}$ (b) Bicubic / LR image $\boldsymbol{y}$ (c) Super-resolution result $\hat{\boldsymbol{x}}$

Supplemental Figure 8: Illustration of a super-resolution by a factor $\times 3$ with a noise of standard deviation $\sigma = 2$. (a) The original high-resolution (HR) image $\boldsymbol{x}$. (b) The low-resolution (LR) image $\boldsymbol{y} = \mathcal{A}\boldsymbol{x} + \boldsymbol{w}$ and its bicubic interpolation. (c) The super-resolution result $\hat{\boldsymbol{x}}$ obtained by our Fast EPLL.