How to SAIF-ly Boost Denoising Performance

Hossein Talebi, *Student Member, IEEE*, Xiang Zhu, *Student Member, IEEE*, and Peyman Milanfar, *Fellow, IEEE*

***Abstract*— Spatial domain image filters (e.g., bilateral filter, non-local means, locally adaptive regression kernel) have achieved great success in denoising. Their overall performance, however, has not generally surpassed the leading transform domain-based filters (such as BM3-D). One important reason is that spatial domain filters lack efficiency to adaptively fine tune their denoising strength; something that is relatively easy to do in transform domain method with shrinkage operators. In the pixel domain, the smoothing strength is usually controlled globally by, for example, tuning a regularization parameter. In this paper, we propose spatially adaptive iterative filtering (SAIF)[1] a new strategy to control the denoising strength locally for any spatial domain method. This approach is capable of filtering local image content iteratively using the given base filter, and the type of iteration and the iteration number are automatically optimized with respect to estimated risk (i.e., mean-squared error). In exploiting the estimated local signal-to-noise-ratio, we also present a new risk estimator that is different from the often-employed SURE method, and exceeds its performance in many cases. Experiments illustrate that our strategy can significantly relax the base algorithm's sensitivity to its tuning (smoothing) parameters, and effectively boost the performance of several existing denoising filters to generate state-of-the-art results under both simulated and practical conditions.**

***Index Terms*— Image denoising, pixel aggregation, risk estimator, spatial domain filter, SURE.**

I. Introduction

SINCE noise exists in basically all modern imaging systems, denoising is among the most fundamental image restoration problems studied in the last decade. There have been numerous denoising algorithms, and in general they can be divided into two main categories: transform domain methods and spatial domain methods.

Transform domain methods are developed under the assumption that the clean image can be well represented as a combination of few transform basis vectors, so the signal-to-noise-ratio (SNR) can be estimated and used to appropriately shrink the corresponding transform coefficients. Specifically, if a basis element is detected as belonging to the true signal, its coefficient should be mostly preserved. On the other hand, if an element is detected as a noise component, its coefficient should be shrunk more, or removed. By doing this, noise can be effectively suppressed while most structures and finer details of the latent image are preserved.

Different algorithms in this category vary in either the transform selection or the shrinkage strategy. Fixed transforms (e.g. wavelet, DCT) are often employed as in [1], [2], and are easy to calculate. However, they may not be effective in representing natural image content with sparse coefficient distributions, and that would inevitably increase the requirement on the shrinkage performance. Non-fixed transforms are also applied. For example, Muresan [3] and Zhang [4] use principle component analysis (PCA). Compared with fixed transformations, PCA is more adaptive to local image content and thus can lead to a more sparse coefficient distribution. However, such decompositions can be quite sensitive to noise. K-SVD [5] and K-LLD [6] use over-complete dictionaries generated from training, which is more robust to noise but computationally expensive. The shrinkage strategy is another important factor that needs to be fully considered. Though there are many competing strategies, it has been shown that the Wiener criterion, which determines the shrinking strength according to (estimated) SNR in each basis element, is perhaps the best strategy that gets close to the optimal performance with respect to mean-squared-error (MSE) [7]. In fact, in practice it has achieved state-of-the-art denoising performance with even simple fixed transforms (such as DCT in BM3D) [2].

Spatial domain methods concentrate on a different noise suppression approach, which estimates each pixel value as a weighted average of other pixels, where higher weights are assigned to "similar" pixels [8]–[12]. Pixel similarities can be calculated in various ways. For the bilateral filter, similarity is determined by both geometric and photometric distances between pixels [8]. Takeda et al. proposed a locally adaptive regression kernel (LARK) denoising method, robustly measuring the pixel similarity based on geodesic distance [9]. Another successful method called non-local means (NLM) extends the bilateral filter by replacing point-wise photometric distance with patch distances, which is more robust to noise [10].[2]

In practice, determining the denoising strength for spatial domain methods is a general difficulty. For example, these methods always contain some tuning (smoothing) parameters that may strongly affect the denoising performance. A larger smoothing parameter would suppress more noise and meanwhile erase more useful image information, ending up with an over-smoothed (biased) output. Less smoothing would

Manuscript received May 31, 2012; revised September 20, 2012; accepted November 1, 2012. Date of publication December 4, 2012; date of current version February 6, 2013. This work was supported in part by the AFOSR Grant FA9550-07-1-0365 and NSF Grant CCF-1016018. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Richard J. Radke.

The authors are with the Department of Electrical Engineering, University of California, Santa Cruz, Santa Cruz, CA 95064 USA (e-mail: htalebi@soe.ucsc.edu; xzhu@soe.ucsc.edu; milanfar@soe.ucsc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2012.2231691

[1]SAIF is the Middle Eastern/Arabic name for sword. This acronym somehow seems appropriate for what the algorithm does by precisely tuning the value of the iteration number.

[2]Spatial domain methods include any method that is based upon the computation of a kernel that is applied locally to the pixel data directly. It is possible to approximately implement many regularization based methods in this framework, but we do not believe there is a one to one correspondence between kernel spatial domain methods and regularization based Bayesian methods [13].

preserve high-frequency signal but also do little denoising (estimation variance). An interesting alternative to tedious parameter tuning is iterative filtering. With this approach, which we promote here, even with a filter estimated from a badly misplaced smoothing parameter, we can still get a well estimated output by applying the same denoising filter several times. But it would seem that the iteration number then becomes another tuning parameter that needs to be carefully treated. Some approaches were developed to handle such parameters. Ramani's Monte–Carlo SURE is capable of optimizing any denoising algorithm with respect to MSE [14], but it requires Gaussian assumption on noise. In [15] we developed a no-reference image content measure named Metric Q to select optimal parameters. However, both Monte–Carlo SURE and Metric Q can only adjust the filtering degree *globally*. Much more efficient estimates could be obtained by smartly changing the denoising strength *locally* as we propose in this paper. More specifically, we present an approach capable of *automatically* adjusting the denoising strength of spatial domain methods according to local SNR. A second contribution is a novel method for estimating the local SNR.

In [13] Milanfar illustrates that a spatial domain denoising process can always be approximated as a transform domain filter, where the orthogonal basis elements are the eigenvectors of a symmetric and positive definite matrix determined by the filter; and the shrinkage coefficients are the corresponding eigenvalues ranging in [0, 1]. For filters such as NLM and LARK the eigenvectors corresponding to the dominant eigenvalues could well represent latent image contents. Based on this idea, we propose a spatially adapted iterative filtering (SAIF) strategy capable of controlling the denoising strength *locally* for any given spatial domain method. The proposed method iteratively filters local image patches, and the iteration method and iteration number are automatically optimized with respect to local MSE, which is estimated from the given image. To estimate the MSE for each patch, we propose a new method called *plug-in* risk estimator. This method is biased and works based on a "pilot" estimate of the latent image. For the sake of comparison, we also derive the often used Stein's unbiased risk estimator (SURE) [16] for the data dependent filtering scheme. While [17] also uses SURE to optimize the NLM kernel parameters, we illustrate that (1) the plug-in estimator can be superior for the same task, and (2) the adaptation approach can be extended to be spatially varying. The paper is organized as follows. In Section II we briefly provide some background, especially [13]'s analysis on spatial domain filters. Section III reviews two iterative methods to control the smoothing strength for the filters. Section IV describes the proposed SAIF strategy in detail. Experimental results are given in Section V to show the performance of the SAIF strategy using several leading filters. Finally we conclude this paper in Section VI.

II. Background

Let us consider the measurement model for the denoising problem:

$$y_i = z_i + e_i, \qquad \text{for } i = 1, \ldots, n \tag{1}$$

where $z_i = z(\mathbf{x}_i)$ is the underlying image at position $\mathbf{x}_i = [x_{i,1}, x_{i,2}]^T$, y_i is the noisy pixel value, and e_i denotes zero-mean white noise[3] with variance σ^2. The problem of denoising is to recover the set of underlying samples $\mathbf{z} = [z_1, \ldots, z_n]^T$. The complete measurement model for the denoising problem in vector notation is:

$$\mathbf{y} = \mathbf{z} + \mathbf{e}. \tag{2}$$

As explained in [9], [13] most spatial domain filters can be represented through the following non-parametric restoration framework:

$$\widehat{z}_i = \arg\min_{z_i} \sum_{j=1}^{n} [z_i - y_j]^2 K(\mathbf{x}_i, \mathbf{x}_j, y_i, y_j) \tag{3}$$

where $\widehat{z}_i$ denotes the estimated pixel at position x_i, and the weight (or kernel) function $K(\cdot)$ measures the similarity between the samples y_i and y_j at positions $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively.

Perhaps the most well known kernel function is the Bilateral (BL) filter [8], which smooths images by means of a nonlinear combination of nearby image values. The method combines pixel values based on both their geometric closeness and their photometric similarity. This kernel can be expressed in a separable fashion as follows:

$$K_{ij} = \exp\left\{ \frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{h_x^2} + \frac{-(y_i - y_j)^2}{h_y^2} \right\} \tag{4}$$

in which h_x and h_y are smoothing (control) parameters.

The NLM [10] is another very popular data-dependent filter which closely resembles the bilateral filter except that the photometric similarity is captured in a patch-wise manner:

$$K_{ij} = \exp\left\{ \frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{h_x^2} + \frac{-\|\mathbf{y}_i - \mathbf{y}_j\|^2}{h_y^2} \right\} \tag{5}$$

where $\mathbf{y}_i$ and $\mathbf{y}_j$ are patches centered at y_i and y_j, respectively. In theory (though not in actual practice,) the NLM kernel has just the patch-wise photometric distance ($h_x \to \infty$).

More recently, the LARK (also called *Steering Kernel* in some publications) [9] was introduced which exploits the geodesic distance based on estimated gradients:

$$K_{ij} = \exp\left\{ -(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{C}_{ij} (\mathbf{x}_i - \mathbf{x}_j) \right\} \tag{6}$$

in which $\mathbf{C}_{ij}$ is a local covariance matrix of the pixel gradients computed from the given data.[4] The gradient is computed from the noisy measurements y_j in a patch around $\mathbf{x}_i$. Robustness to noise and perturbations of the data is an important advantage of LARK.

In general, all of these restoration algorithms are based on the same framework (3) in which some data-adaptive kernels are assigned to each pixel contributing to the filtering. Minimizing equation (3) gives a normalized weighted averaging process:

$$\widehat{z}_i = \mathbf{w}_i^T \mathbf{y} \tag{7}$$

[3]We make no other distributional assumptions on the noise.

[4]Refer to [9] for more details on $\mathbf{C}_{ij}$.

where the weight vector $\mathbf{w}_i$ is

$$\mathbf{w}_i = \frac{1}{\sum_{j=1}^{n} K_{ij}}[K_{i1}, K_{i2}, \ldots, K_{in}]^T. \quad (8)$$

By stacking the weight vectors together, the filtering process for all the sample pixels can be represented simultaneously through a matrix-vector multiplication form

$$\widehat{\mathbf{z}} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_n^T \end{bmatrix} \mathbf{y} = \mathbf{W}\mathbf{y} \quad (9)$$

where $\widehat{\mathbf{z}}$ and $\mathbf{W}$ denote the estimated signal and the filter matrix, respectively. While the data-dependent filter $\mathbf{W}(\mathbf{y})$ is not generally linear, in Appendix VI we show that our employed filter $\mathbf{W}(\widetilde{\mathbf{z}})$, which is computed based on the *pre-filtered* patch $\widetilde{\mathbf{z}}$, can be closely treated as a filter that is not *stochastically* dependent on the input data.

$\mathbf{W}$ is a positive row-stochastic matrix (every row sums up to one). This matrix is not generally symmetric, though it has real, positive eigenvalues [13]. The Perron-Frobenius theory describes the spectral characteristics of this matrix [18], [19]. In particular, the eigenvalues of $\mathbf{W}$ satisfy $0 \leq \lambda_i \leq 1$; the largest one is uniquely equal to one ($\lambda_1 = 1$) while the corresponding eigenvector is $\mathbf{v}_1 = \frac{1}{\sqrt{n}}[1, 1, \ldots, , 1]^T$. The last property implies that a flat image stays unchanged after filtering by $\mathbf{W}$.

Although $\mathbf{W}$ is not a symmetric matrix in general, it can be closely approximated with a symmetric positive definite matrix[5] [20]. The symmetrized $\mathbf{W}$ must also stay row-stochastic, which means we get a symmetric positive definite matrix which is doubly (i.e., row- and column-) stochastic. The symmetric $\mathbf{W}$ enables us to to compute its eigen-decomposition as follows:

$$\mathbf{W} = \mathbf{V}\mathbf{S}\mathbf{V}^T \quad (10)$$

where $\mathbf{S} = \text{diag}[\lambda_1, \ldots, \lambda_n]$ contains the eigenvalues in decreasing order $0 \leq \lambda_n \leq, \ldots, < \lambda_1 = 1$, and $\mathbf{V}$ is an orthogonal matrix $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ containing the respective eigenvectors of $\mathbf{W}$ in its columns. Since $\mathbf{V}$ is orthogonal, its columns specify a set of basis functions. So the filtering process can be explained as:

$$\widehat{\mathbf{z}} = \mathbf{W}\mathbf{y} = \mathbf{V}\mathbf{S}\mathbf{V}^T\mathbf{y} \quad (11)$$

where the input data $\mathbf{y}$ is first transformed into the domain spanned by the eigenvectors of $\mathbf{W}$; then, each coefficient is scaled (shrunk) by the factor λ_i; and finally an inverse transform is applied, yielding the output.

From the above analysis we see that the denoising strength for each basis of a given filter is thus controlled by the shrinkage factor $\{\lambda_i\}$. In the following sections we will discuss this more explicitly.

[5]Indeed, it can be shown that $\frac{1}{n}\|\mathbf{W} - \mathbf{W}_{\text{sym}}\|_F = O(n^{-\frac{1}{2}})$. That is, the RMS error gets smaller with increasing dimension.

III. Iterative Filtering Methods

Optimal shrinkage strategies based on various spatial domain filters have been discussed in [13], where statistical analysis shows that the optimal filter with respect to MSE is the local Wiener filter with $\lambda_i = \frac{1}{1+\text{snr}_i^{-1}}$, where snr_i denotes signal-to-noise ratio of the i-th channel. However, the local Wiener filter requires exact knowledge of the local signal-to-noise (SNR) in each basis channel, which is not directly accessible in practice. In denoising schemes such as [2] and [4] the Wiener shrinkage criterion works based on a pilot estimate of the latent image. Still, the Wiener filter's performance strictly relies on accuracy of this estimate. Iterative filtering can be a reliable alternative for reducing sensitivity of the basis shrinkage to the estimated local SNR. Then, the iteration number will be the only parameter to be locally optimized.

To approach the locally optimal filter performance in a stable way, we propose the use of two iterative local operators; namely *diffusion* and *boosting*. In [21] we have shown that performance of any type of kernel could be enhanced by iterative diffusion which gradually removes the noise in each iteration. Yet, diffusion filtering also takes away latent details from the underlying signal. On the other hand, iterative boosting is a mechanism to preserve these lost details of the signal. By using the two iterative filtering methods, we can avoid either over-smoothing or under-smoothing due to incorrect parameter settings. In other words, these two methods provide a way to start with any filter, and properly control the values of shrinkage factors $\{\lambda_i\}$ to achieve a good and stable approximation of the Wiener filter. In the following we discuss the two approaches, separately.

A. Diffusion

The idea of diffusion in image filtering was originally motivated by the physical principles of heat propagation and described using a partial differential equation. In our context, we consider the discrete version of it, which is conveniently represented by repeated applications of the same filter as described in [13]:

$$\widehat{\mathbf{z}}_k = \mathbf{W}\widehat{\mathbf{z}}_{k-1} = \mathbf{W}^k\mathbf{y}. \quad (12)$$

Each application of $\mathbf{W}$ can be interpreted as one step of anisotropic diffusion with the filter $\mathbf{W}$. Choosing a small iteration number k preserves the underlying structure, but also does little denoising. Conversely, a large k tends to over-smooth and remove noise and high frequency details at the same time. Minimization of MSE (or more accurately an estimate of it) determines when is the best time to stop filtering, which will help avoid under- or over- smoothing.

As long as $\mathbf{W}$ is symmetric, the filter in the iterative model (12) can be decomposed as:

$$\mathbf{W}^k = \mathbf{V}\mathbf{S}^k\mathbf{V}^T \quad (13)$$

in which $\mathbf{S}^k = \text{diag}[\lambda_1^k, \ldots, \lambda_n^k]$. It is worth noting that despite the common interpretation of k as a discrete step, the spectral decomposition of $\mathbf{W}^k$ makes it possible to replace k with any positive real number.

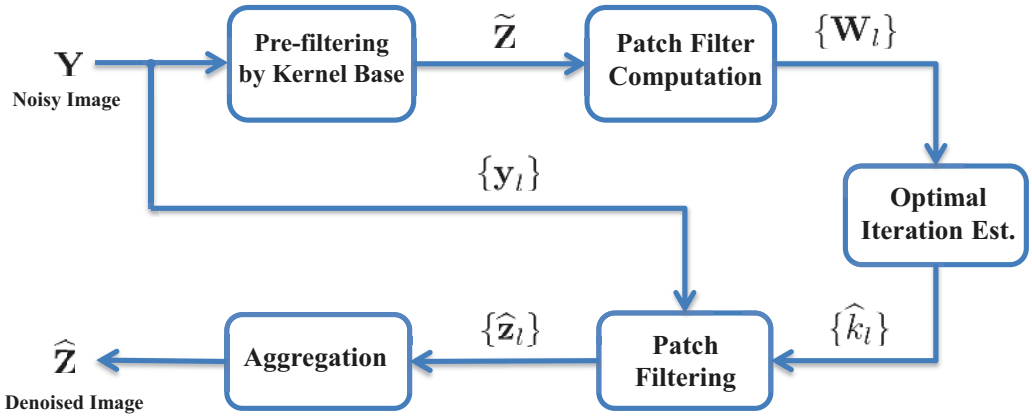


Fig. 1. Diagram of SAIF method.

The latent image $\mathbf{z}$ can be written in the column space of $\mathbf{V}$ as $\mathbf{b} = \mathbf{V}^T\mathbf{z}$, where $\mathbf{b} = [b_1, b_2, \ldots, b_n]^T$, and $\{b_i^2\}$ denote the projected signal energy over all the eigenvectors. As shown in [13] the iterative estimator $\widehat{\mathbf{z}}_k = \mathbf{W}^k\mathbf{y}$ has the following squared bias:

$$\|\text{bias}_k\|^2 = \|(\mathbf{I} - \mathbf{W}^k)\mathbf{z}\|^2 = \sum_{i=1}^{n}(1 - \lambda_i^k)^2 b_i^2. \quad (14)$$

Correspondingly, the estimator's variance is:

$$\text{var}(\widehat{\mathbf{z}}_k) = \text{tr}(\text{cov}(\widehat{\mathbf{z}}_k)) = \sigma^2 \sum_{i=1}^{n} \lambda_i^{2k}. \quad (15)$$

Overall, the MSE is given by

$$\text{MSE}_k = \|\text{bias}_k\|^2 + \text{var}(\widehat{\mathbf{z}}_k) = \sum_{i=1}^{n}(1 - \lambda_i^k)^2 b_i^2 + \sigma^2 \lambda_i^{2k}. \quad (16)$$

As the iteration number k grows, the bias term increases, but the variance decays to the constant value of σ^2. Of course, this expression for the MSE is not practically useful yet, since the coefficients $\{b_i^2\}$ are not known. Later we describe a way to estimate the MSE in practice. But first, let us introduce the second iterative mechanism which we will employ. Boosting is discussed in the following and as we will see, its behavior is quite different from the diffusion filtering.

B. Boosting

Although the classic diffusion filtering has been used widely, this method often fails in denoising image regions with low SNR. This is due to the fact that each diffusion iteration is essentially one step of low-pass filtering. In other words, diffusion always removes some components of the noise and signal, concurrently. This shortcoming is tackled effectively by means of boosting which *recycles* the removed components of signal from the residuals, in each iteration. Defining the residuals as the difference between the estimated signal and the noisy signal: $\mathbf{r}_k = \mathbf{y} - \widehat{\mathbf{z}}_{k-1}$, the iterated estimate can be expressed as

$$\begin{aligned}\widehat{\mathbf{z}}_k &= \widehat{\mathbf{z}}_{k-1} + \mathbf{W}\mathbf{r}_k \\ &= \sum_{j=0}^{k} \mathbf{W}(\mathbf{I} - \mathbf{W})^j\mathbf{y} = \left(\mathbf{I} - (\mathbf{I} - \mathbf{W})^{k+1}\right)\mathbf{y} \end{aligned} \quad (17)$$

where $\widehat{\mathbf{z}}_0 = \mathbf{W}\mathbf{y}$. As can be seen, as k increases, the estimate returns to the noisy signal $\mathbf{y}$. In other words, the boosting filter has fundamentally different behavior than the diffusion iteration where the estimated signal gets closer to a constant after each iteration. The squared magnitude of the bias after k iterations is

$$\|\text{bias}_k\|^2 = \|(\mathbf{I} - \mathbf{W})^{k+1}\mathbf{z}\|^2 = \sum_{i=1}^{n}(1 - \lambda_i)^{2k+2} b_i^2. \quad (18)$$

And the variance of the estimator also is

$$\text{var}(\widehat{\mathbf{z}}_k) = \text{tr}(\text{cov}(\widehat{\mathbf{z}}_k)) = \sigma^2 \sum_{i=1}^{n}\left(1 - (1 - \lambda_i)^{k+1}\right)^2. \quad (19)$$

Then the overall MSE is

$$\text{MSE}_k = \sum_{i=1}^{n}(1 - \lambda_i)^{2k+2} b_i^2 + \sigma^2\left(1 - (1 - \lambda_i)^{k+1}\right)^2. \quad (20)$$

As k grows, the bias term decreases and the variance increases. Contrasting the behavior of the diffusion iteration, we observe that *when diffusion fails to improve the filtering performance, it can be replaced by boosting*. This is the fundamental observation that motivates our approach. More specifically, the contribution of this work is that we simultaneously and automatically optimize the type and number of iterations locally to boost the performance of a given base filtering procedure.

IV. Proposed Method

Based on the analysis from Section III we propose an image denoising strategy which, given any filter using the framework (3), can boost its performance by utilizing its spatially adapted transform and by employing an optimized iteration method. This iterative filtering is implemented patch-wise, so that it is capable of automatically adjusting the local smoothing strength according to local SNR. Fig. 1 depicts a

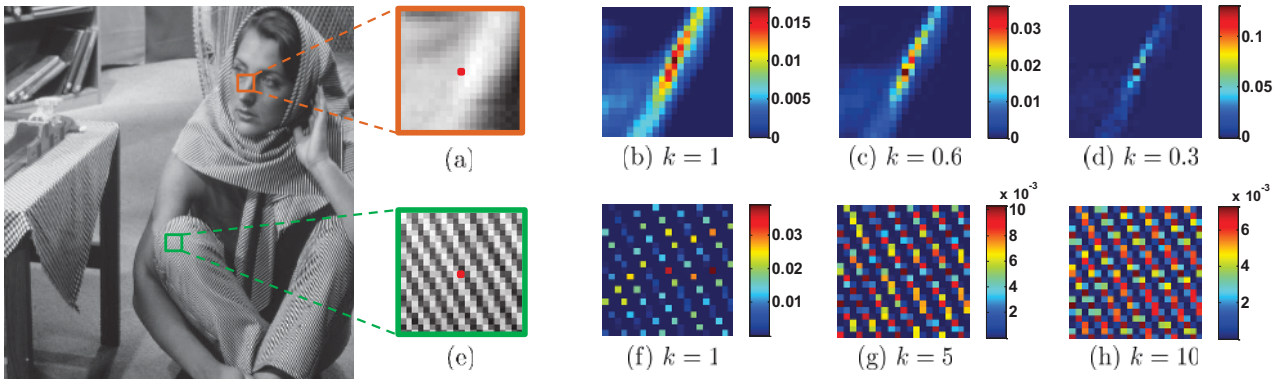


Fig. 2. Filters based on the NLM kernel with different iteration number k. (a) Smooth patch and the jth pixel. (b) jth row of the patch filter $\mathbf{W}$. (c) and (d) jth row of the iterated patch filter $\mathbf{W}^k$ for different iteration numbers. (e) Texture patch and the j-th pixel. (f) jth row of the patch filter $\mathbf{W}$. (g) and (h) jth row of the iterated patch filter $\mathbf{W}^k$ for different iteration numbers.

block diagram of the proposed approach. Starting from the noisy image $\mathbf{Y}$ and splitting it into N overlapping patches $\{\mathbf{y}_l\}_{l=1}^N$, we aim to denoise each noisy patch $\mathbf{y}_l$, separately. To calculate the local filter $\mathbf{W}_l$, we use an estimated image $\widetilde{\mathbf{Z}}$ which is filtered by the standard kernel baseline. After that, MSEs for the two iteration approaches (diffusion and boosting) are estimated for each patch and by comparing their values, the optimal iteration method and consequently the iteration number $\widehat{k}_l$ is selected, generating the filtered patch $\widehat{\mathbf{z}}_l$. Since these filtered patches are overlapped, an aggregation method is finally carried out to compute the denoised image $\widehat{\mathbf{Z}}$. The key steps of this approach are the optimal iteration estimation and the aggregation, which will be described in the rest of this section.

A. Optimal Iteration Estimation

Given a patch $\mathbf{y}$ and its filter matrix $\mathbf{W}$, the task of this step is to select the best iteration method (either diffusion or boosting) and its iteration number that minimizes the MSE. More explicitly, the optimal stopping time $\widehat{k}$ for each iteration method can be expressed as:

$$\widehat{k} = \arg\min_k \ \mathrm{MSE}_k. \tag{21}$$

One way to compute an unbiased estimate of MSE is the often-used SURE [16]. An alternative we propose here is the plug-in risk estimator, which is biased and works based on an estimate of the local SNR. First, note that in practice, eigenvalues and eigenvectors of the filter are always estimated from a *pre-filtered* patch $\widetilde{\mathbf{z}}$, obtained using the base filter with some arbitrary parameter settings. More explicitly we have:

$$\mathbf{W}(\widetilde{\mathbf{z}}) = \mathbf{V}\mathbf{S}\mathbf{V}^T. \tag{22}$$

It is worth repeating that despite the earlier interpretation of k as a discrete step, the spectral decomposition of $\mathbf{W}^k$ makes it clear that in practice, k can be any positive real number. To be more specific, $\mathbf{W}^k = \mathbf{V}\mathbf{S}^k\mathbf{V}^T$, with $\mathbf{S}^k = \mathrm{diag}[\lambda_1^k, \ldots, , \lambda_n^k]$ where k is any non-negative real number. In actual implementation, the filter can be applied with modified eigenvalues for any $k > 0$. This may seem like a minor point, but in practice can significantly improve the performance as compared to when k is restricted to only positive integers. In effect, a real-valued k automatically and smoothly adjusts the local bandwidth of the filter. Fig. 2 illustrates the iterated filters for two different patches. As can be seen, while decreasing the iteration number k can be interpreted as smaller tuning parameter h_y for NLM kernel, larger k is equivalent to a wider kernel.

Next, We discuss the two risk estimators and show that the plug-in can exploit the estimated local SNR to have better performance as compared to the SURE estimator.

1) Plug-In Risk Estimator: The plug-in estimator is described in Algorithm 1. In this method, risk estimators for diffusion and boosting are computed based on the pre-filtered patch $\widetilde{\mathbf{z}}$, computed using the base filter with arbitrary parameters. More explicitly, the signal coefficients can be estimated as:

$$\widetilde{\mathbf{b}} = \mathbf{V}^T\widetilde{\mathbf{z}}. \tag{23}$$

This estimate's contribution can be interpreted as equipping the risk estimator with some prior knowledge of the local SNR of the image. The estimated signal coefficients allow us to use (16) and (20) to estimate MSE_k in each patch:

Diffusion Plug-in Risk Estimator:

$$\text{Plug-in}_k^{df} = \sum_{i=1}^{n}(1-\lambda_i^k)^2\widetilde{b}_i^2 + \sigma^2\lambda_i^{2k}. \tag{24}$$

Boosting Plug-in Risk Estimator:

$$\text{Plug-in}_k^{bs} = \sum_{i=1}^{n}(1-\lambda_i)^{2k+2}\widetilde{b}_i^2 + \sigma^2\left(1-(1-\lambda_i)^{k+1}\right)^2. \tag{25}$$

In each patch, minimum values of Plug-in_k^{df} and Plug-in_k^{bs} as a function of k are computed and compared, and the iteration type with the least risk is chosen. It is worth mentioning that since the optimal iteration number $\widehat{k}$ can be any real positive value, in the implementation of the diffusion filter, $\mathbf{W}^{\widehat{k}}$ is replaced by $\mathbf{V}\mathbf{S}^{\widehat{k}}\mathbf{V}^T\mathbf{y}$ in which $\mathbf{S}^{\widehat{k}} = \mathrm{diag}[\lambda_1^{\widehat{k}}, \ldots, , \lambda_n^{\widehat{k}}]$. This has been similarly shown for the boosting filter in Algorithm 1. Next, for the sake of comparison, the SURE estimator is discussed.

Algorithm 1: Plug-in Risk Estimator

Input: Noisy Patch: $\mathbf{y}$, Pre-filtered Patch: $\widetilde{\mathbf{z}}$, Patch Filter: $\mathbf{W}$

Output: Denoised Patch: $\widehat{\mathbf{z}}$

Eigen-decomposition of the filter $\mathbf{W}(\widetilde{\mathbf{z}}) = \mathbf{V}\mathbf{S}\mathbf{V}^T$;

$\widetilde{\mathbf{b}} = \mathbf{V}^T\widetilde{\mathbf{z}} \Leftarrow$ Compute the signal coefficients;

Plug-in$_k^{df}$, Plug-in$_k^{bs} \Leftarrow$ Compute the estimated risks;

if $\min\{\text{Plug-in}_k^{df}\} < \min\{\text{Plug-in}_k^{bs}\}$

  $\widehat{k} = \underset{k}{\text{argmin}}\ \text{Plug-in}_k^{df} \Leftarrow$ Diffusion optimal iteration number;

  $\widehat{\mathbf{z}} = \mathbf{V}\mathbf{S}^{\widehat{k}}\mathbf{V}^T\mathbf{y} \Leftarrow$ Diffusion patch denoising;

else

  $\widehat{k} = \underset{k}{\text{argmin}}\ \text{Plug-in}_k^{bs} \Leftarrow$ Boosting optimal iteration number;

  $\widehat{\mathbf{z}} = \mathbf{V}\left(\mathbf{I} - (\mathbf{I}-\mathbf{S})^{\widehat{k}+1}\right)\mathbf{V}^T\mathbf{y} \Leftarrow$ Boosing patch denoising;

end

2) SURE Estimator: Denoting $F(\mathbf{y})$ as an estimate of the latent signal $\mathbf{z}$, the SURE estimator or MSE is defined as:

$$\text{SURE}(\mathbf{y}) = \|\mathbf{y} - F(\mathbf{y})\|^2 + 2\sigma^2\text{div}(F(\mathbf{y})) - n\sigma^2 \qquad (26)$$

where $\text{div}(F(\mathbf{y})) \equiv \sum_i \frac{\partial F_i(\mathbf{y})}{\partial y_i}$. Under the additive Gaussian noise assumption, this random variable is an unbiased estimate of the MSE. In our context, $F(\mathbf{y})$ is replaced by $\mathbf{W}^k\mathbf{y}$ which can be approximately treated as a linear filtering framework (see Appendix A). With this linear approximation we have: $\text{div}(F(\mathbf{y})) \approx \text{tr}(\mathbf{W}^k)$, and then the SURE estimator for the diffusion process can be expressed as:

$$\text{SURE}_k^{df} = \|(\mathbf{I} - \mathbf{W}^k)\mathbf{y}\|^2 + 2\sigma^2\text{tr}(\mathbf{W}^k) - n\sigma^2. \qquad (27)$$

Considering the eigen-decomposition of the filter, replacing $\mathbf{y}$ with $\mathbf{V}\check{\mathbf{b}}$ (where $\check{\mathbf{b}}$ is the energy distribution of the *noisy* signal over the eigenvectors; $\check{\mathbf{b}} = \mathbf{V}^T\mathbf{y}$) after some simplifications, we have

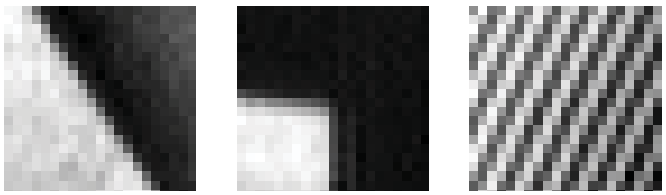
$$\begin{aligned}\text{SURE}_k^{df} &= (\mathbf{V}\check{\mathbf{b}})^T\mathbf{V}(\mathbf{I}-\mathbf{S}^k)^2\mathbf{V}^T(\mathbf{V}\check{\mathbf{b}}) + 2\sigma^2\text{tr}(\mathbf{W}^k) - n\sigma^2 \\ &= \sum_{i=1}^{n}(1-\lambda_i^k)^2\check{b}_i^2 + 2\sigma^2\lambda_i^k - \sigma^2. \qquad (28)\end{aligned}$$

It is easy to show that the expected value of SURE_k^{df} will replace $\check{b}_i^2$ with $b_i^2 + \sigma^2$, which after simplification indeed yields (16). Replacing $F(\mathbf{y})$ with $\widehat{\mathbf{z}}_k = \left(\mathbf{I} - (\mathbf{I}-\mathbf{W})^{k+1}\right)\mathbf{y}$ for boosting filtering, we can get the corresponding SURE as:

$$\begin{aligned}\text{SURE}_k^{bs} &= \|(\mathbf{I}-\mathbf{W})^{k+1}\mathbf{y}\|^2 \\ &\quad + 2\sigma^2\text{tr}(\mathbf{I} - (\mathbf{I}-\mathbf{W})^{k+1}) - n\sigma^2. \qquad (29)\end{aligned}$$

Doing the same simplifications as before and replacing $\mathbf{y}$ with $\mathbf{V}\check{\mathbf{b}}$, we get

$$\begin{aligned}\text{SURE}_k^{bs} &= (\mathbf{V}\check{\mathbf{b}})^T\mathbf{V}(\mathbf{I}-\mathbf{S})^{2k+2}\mathbf{V}^T(\mathbf{V}\check{\mathbf{b}}) \\ &\quad + 2\sigma^2\text{tr}(\mathbf{I} - (\mathbf{I}-\mathbf{W})^{k+1}) - n\sigma^2 = \sum_{i=1}^{n}(1-\lambda_i)^{2k+2}\check{b}_i^2 \\ &\quad + 2\sigma^2(1-(1-\lambda_i)^{2k+2}) - \sigma^2. \qquad (30)\end{aligned}$$



(a) (b) (c)

Fig. 3. Clean patches from *Barbara*. (a) Edge. (b) Corner. (c) Texture.

Again, we can show that the expected value of SURE_k^{bs} yields (20). This SURE estimator also has a similar algorithmic description to the plug-in represented in Algorithm. 1.

3) Plug-In Versus SURE: While SURE estimator is widely used for the task of risk estimation, our introduced plug-in estimator is superior in many cases. Figs. 4, 5 and 6 illustrate MSE denoising curves of the three patches shown in Fig. 3, perturbed[6] by AWGN with $\sigma = 25$. Three denoiser kernels (Bilateral [8], NLM [10] and LARK [9]) are used in our experiments, true MSE values of diffusion and twicing methods and corresponding estimated risks are compared for each patch. As can be seen, the plug-in outperforms the SURE estimator in most instances, certainly insofar as the shape of the curves are concerned.[7] While this experiment anecdotally shows that the plug-in is superior to the SURE estimator, a theoretical analysis is much more convincing. We detail this analysis in Appendix B, and summarize it below. Our study shows that two important factors affect performance of the plug-in estimator; namely the pre-filter, and the baseline kernel type. Intuitively, the major advantage of the plug-in is use of the local SNR, which is estimated by the pre-filter. However, the SURE estimator is only a function of the kernel type (eigenvalues of the kernel $\{\lambda_i\}$).

The accuracy of these two risk estimators is analyzed in Appendix VI. With Gaussian assumption for the noise in the pre-filtered patch $\widetilde{\mathbf{z}} = \mathbf{z} + \boldsymbol{\eta}$ where $\boldsymbol{\eta} = \mathbf{N}(\mathbf{0}, \nu^2\mathbf{I})$, and defining $\beta = \frac{\nu^2}{\sigma^2}$ as relative variance of the noise in the pre-filtered and noisy signal, we prove that the plug-in estimator outperforms SURE when in each channel i:

$$\text{snr}_i \geq \frac{\beta^2(n+2) - 2}{4(1-\beta)} \qquad (31)$$

where $\text{snr}_i = \frac{b_i^2}{\sigma^2}$ is the signal-to-noise ratio of channel i. For the plug-in estimator to be superior to SURE, regardless of snr_i, the relative variance must be $\beta \leq \sqrt{\frac{2}{n+2}}$. This implies that for a typical patch size (say $n = 11 \times 11$) and a moderately effective pre-filter ($\beta \leq 0.13$), the plug-in estimator is consistently better than SURE.

[6]Averaging over 50 noise realizations in a Monte–Carlo simulation.

[7]We also compared the SURE estimator with the Monte–Carlo SURE in [14], but this did not yield better results. As Ramani et al discuss [14], their Monte–Carlo SURE has a sufficiently low variance estimate when $F(\mathbf{y})$ mostly performs "local" operations (or equivalently, $\mathbf{W}(\mathbf{y})$ is quite sparse in $F(\mathbf{y}) = \mathbf{W}(\mathbf{y})\mathbf{y}$). Yet, when $\mathbf{y}$ is an image patch, $F(\mathbf{y})$ is not a local operator (or equivalently, $\mathbf{W}(\mathbf{y})$ is not "nearly" diagonal). As a result, variance of the Monte–Carlo SURE estimator will be large for the purpose of patch-based risk estimation.

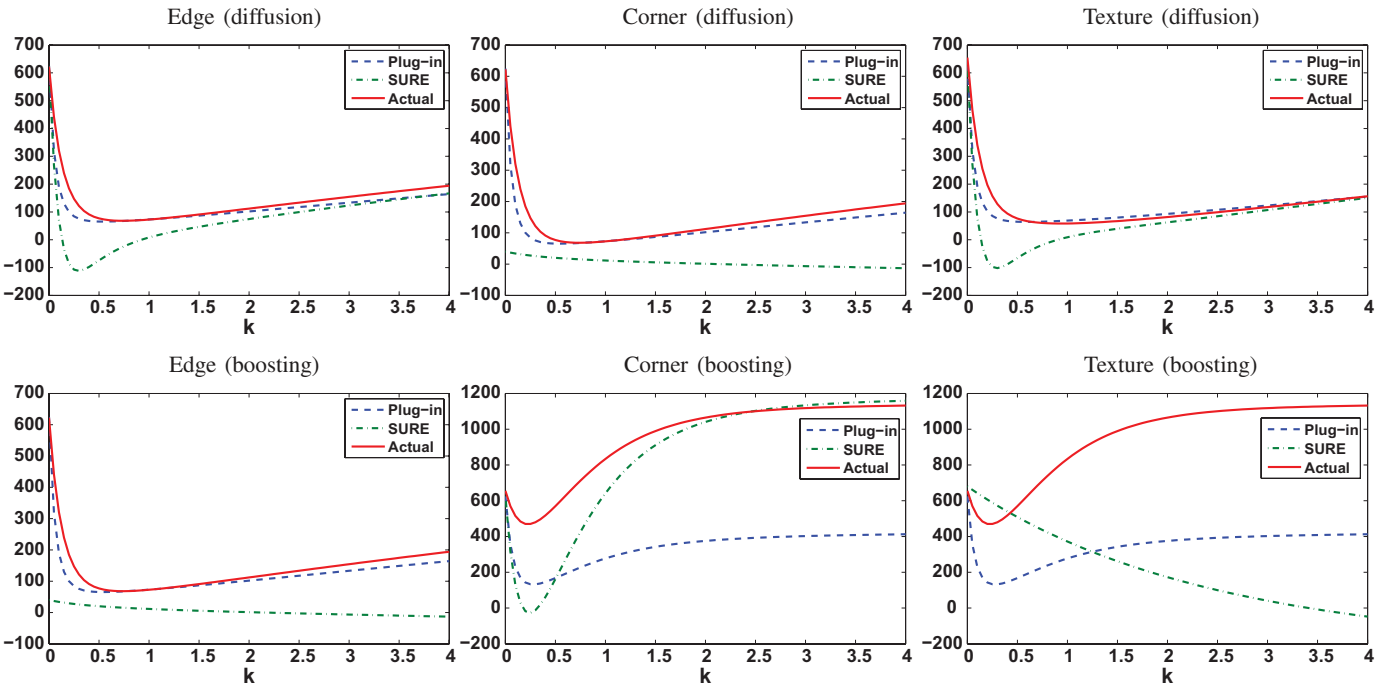


Fig. 4. MSE of the three patches using Bilateral kernel [8] with diffusion/boosting iterations and plug-in/SURE estimators.

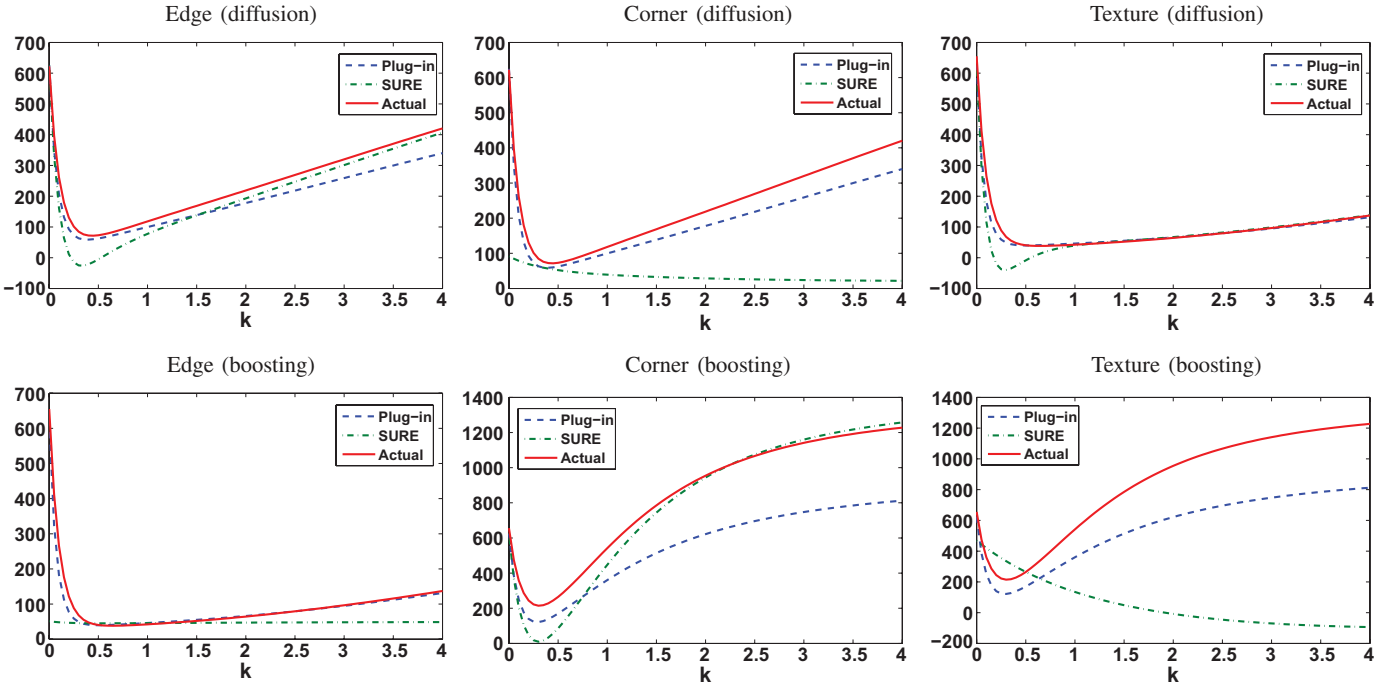


Fig. 5. MSE of the three patches using NLM kernel [10] with diffusion/boosting iterations and plug-in/SURE estimators.

Another key factor in the comparison of the two estimators is their relative sensitivity to the type of kernel. In Appendix VI we study the effect of kernel type, specifically the filter eigenvalues. This analysis shows that when $\lambda_i \to 0$, as long as (31) holds, the plug-in is less sensitive than SURE. NLM and Bilateral are examples of these "aggressive" kernels with many eigenvalues close to 0. On the other hand, our sensitivity analysis also implies that for filters with typically larger eigenvalues $\{\lambda_i\}$, the plug-in estimator is more sensitive.

These results motivate replacing SURE with the plug-in risk estimator for many estimation instances. In our experiments (Section V) on the whole image we compare performance of the two estimators for each base kernel and confirm our claims. But before moving on to the experiments, we explain our aggregation strategy in the next section.

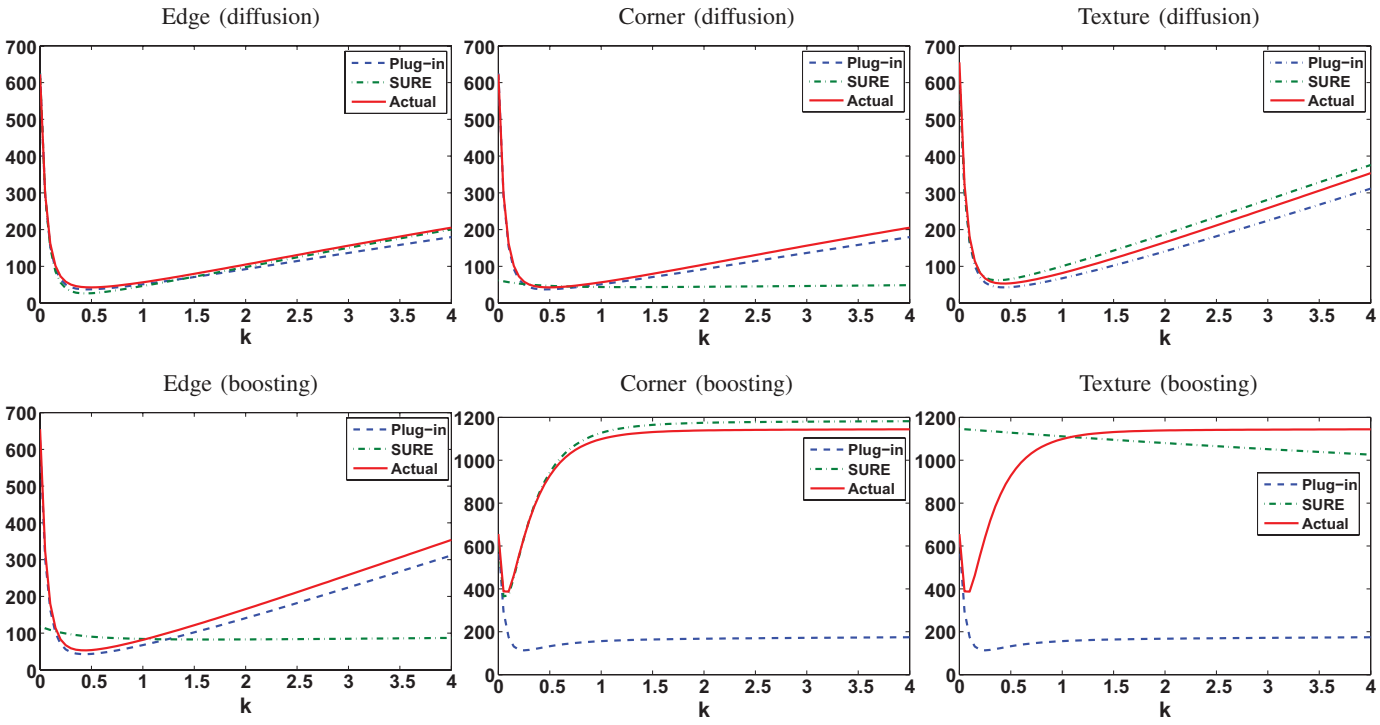


Fig. 6. MSE of the three patches using LARK kernel [9] with diffusion/boosting iterations and plug-in/SURE estimators.

B. Aggregation

So far, we have found the best estimate for the iteration number in each patch. Our optimized per-patch filtering can be expressed as choosing between one of these two iterations:

$$\text{Diffusion}: \quad \widehat{\mathbf{z}}_j = \mathbf{W}^{\widehat{k}_j}\mathbf{y}_j \tag{32}$$

$$\text{Boosting}: \quad \widehat{\mathbf{z}}_j = \left(\mathbf{I} - (\mathbf{I}-\mathbf{W})^{\widehat{k}_j+1}\right)\mathbf{y}_j \tag{33}$$

in which $\mathbf{y}_j$ and $\widehat{\mathbf{z}}_j$ are the j-th noisy and denoised patch, respectively, and $\widehat{k}_j$ denotes the estimated ideal stopping time for this patch. To simplify the notation, let us denote the two filters in $\mathbf{W}^{\widehat{k}_j}$ and $\left(\mathbf{I}-(\mathbf{I}-\mathbf{W})^{\widehat{k}_j+1}\right)$ as $\widehat{\mathbf{W}}_j$. In other words, each patch is estimated as $\widehat{\mathbf{z}}_j = \widehat{\mathbf{W}}_j\mathbf{y}_j$, where $\widehat{\mathbf{W}}_j$ can be computed from either diffusion or boosting process. As a result of the overlapped patches, multiple estimates are obtained for each pixel. We need to aggregate all of these estimates to compute the final estimate for each pixel. What we have is a vector of estimates of the pixel z_l which need to be aggregated to form the final denoised pixel $\widehat{z}_l$. Fig. 7 illustrates an example of three overlapping patches and the computed estimates in each of them. The weighted averaging can improve the aggregation especially when the weights are estimated based on the risk associated with each estimate. Considering that the plug-in and SURE assign, respectively, biased and unbiased risk estimates to each pixel, we discuss two aggregation strategies which best fit each estimator. In our framework, a variance-based aggregation is employed for SURE and an exponentially weighted averaging is used for the plug-in estimator.

1) Variance-Based Aggregation: A possible weighted average is the LMMSE (linear minimum mean-squared-error)

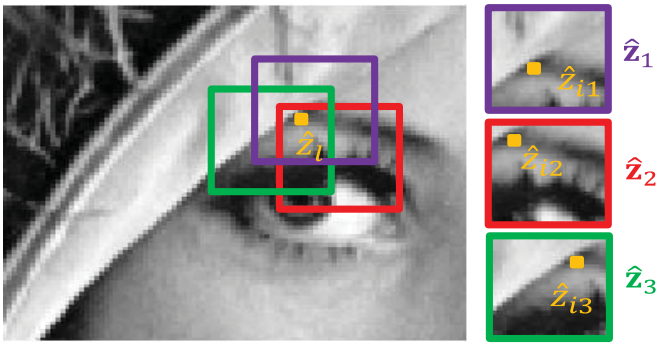


Fig. 7. Overlapping patches give multiple estimates for each pixel. Example of three overlapping patches $\widehat{\mathbf{z}}_1$, $\widehat{\mathbf{z}}_2$, and $\widehat{\mathbf{z}}_3$ give three estimates $\widehat{z}_{i1}$, $\widehat{z}_{i2}$, and $\widehat{z}_{i3}$ for computing the final denoised pixel $\widehat{z}_l$.

scheme that takes into account the relative confidence in each estimate as measured by the inverse of the estimator error variance. More explicitly, the error covariance of our proposed estimator is approximated as:

$$\mathbf{C}_e = \text{cov}(\widehat{\mathbf{z}}_j - \mathbf{z}_j) = \text{cov}(\widehat{\mathbf{W}}_j\mathbf{e}_j) = \sigma^2\widehat{\mathbf{W}}_j^2. \tag{34}$$

We denote $\widehat{z}_{ij}$ as the denoised estimate for the i-th pixel in the j-th patch $\mathbf{z}_j$. Then, the variance of the error associated with the i-th pixel estimate in the j-th patch, v_{ij} , is given by the i-th diagonal element of $\mathbf{C}_e$. Inverse of the estimator error variances v_{ij}, are the weights we use for the aggregation:

$$\widehat{z}_l = \sum_{j=1}^{M} \frac{\frac{\widehat{z}_{ij}}{v_{ij}}}{\sum_{j=1}^{M}\frac{1}{v_{ij}}} \tag{35}$$

Fig. 8. Some benchmark images that we use to evaluate performance of denoising methods. From left to right: *Peppers*, *Lena*, *Cameraman*, *Man*, *Boat*, and *Mandrill*.

TABLE I

PSNR VALUES FOR THE APPLICATION OF BILATERAL KERNEL [8] WITH FIXED PARAMETERS FOR EACH NOISE REALIZATION (1ST COLUMN); SAIF WITH SURE ESTIMATOR (2ND COLUMN), AND SAIF WITH THE PLUG-IN RISK ESTIMATOR (3RD COLUMN)

σ	*Peppers* (512 × 512)			*Lena* (512 × 512)			*Cameraman* (256 × 256)			*Man* (512 × 512)		
	Standard	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**
5	37.12	37.19	**37.24**	37.31	37.48	**37.63**	37.30	37.35	**37.63**	36.01	36.33	**36.36**
15	31.64	31.18	**32.57**	31.34	31.10	**32.48**	30.18	30.35	**30.87**	29.61	29.82	**30.29**
25	28.69	27.87	**30.44**	28.48	27.78	**30.29**	27.03	26.66	**28.14**	26.97	26.81	**27.94**
σ	*Boat* (512 × 512)			*Stream* (512 × 512)			*Parrot* (256 × 256)			*Mandrill* (512 × 512)		
	Standard	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**
5	35.95	**36.24**	36.22	34.83	35.05	**35.06**	36.83	36.93	**37.1**	34.48	34.50	**34.57**
15	29.88	29.81	**30.60**	27.79	28.16	**28.21**	29.85	29.64	**30.52**	26.99	27.34	**27.33**
25	27.16	26.62	**28.31**	25.32	25.26	**25.83**	26.86	26.30	**27.84**	24.20	24.30	**25.57**

TABLE II

PSNR VALUES FOR THE APPLICATION OF NLM KERNEL [10] WITH FIXED PARAMETERS FOR EACH NOISE REALIZATION (1ST COLUMN); SAIF WITH SURE ESTIMATOR (2ND COLUMN), AND SAIF WITH THE PLUG-IN RISK ESTIMATOR (3RD COLUMN)

σ	*Peppers* (512 × 512)			*Lena* (512 × 512)			*Cameraman* (256 × 256)			*Man* (512 × 512)		
	Standard	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**
5	37.34	**37.62**	37.48	38.02	38.42	**38.45**	37.75	37.98	**38.06**	36.73	37.01	**37.07**
15	31.94	33.14	**33.34**	31.93	33.12	**33.39**	30.86	31.62	**31.73**	29.96	30.66	**30.82**
25	29.10	30.96	**31.29**	28.91	30.55	**30.94**	27.84	28.94	**28.98**	27.05	28.02	**28.19**
σ	*Boat* (512 × 512)			*Stream* (512 × 512)			*Parrot* (256 × 256)			*Mandrill* (512 × 512)		
	Standard	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**
5	36.62	**36.90**	36.89	35.36	35.42	**35.55**	37.18	37.58	**37.66**	34.92	34.95	**35.05**
15	30.38	30.83	**31.17**	28.06	28.53	**28.58**	30.51	31.24	**31.32**	27.64	27.96	**28.03**
25	27.43	28.10	**28.58**	25.24	25.83	**25.88**	27.71	28.67	**28.87**	24.55	24.96	**25.13**

TABLE III

PSNR VALUES FOR THE APPLICATION OF THE LARK KERNEL [9] WITH FIXED PARAMETERS FOR EACH NOISE REALIZATION (1ST COLUMN); SAIF WITH SURE ESTIMATOR (2ND COLUMN), AND SAIF WITH THE PLUG-IN RISK ESTIMATOR (3RD COLUMN)

σ	*Peppers* (512 × 512)			*Lena* (512 × 512)			*Cameraman* (256 × 256)			*Man* (512 × 512)		
	Standard	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**
5	36.18	**37.20**	36.89	36.49	**38.39**	37.75	36.31	**37.92**	37.37	35.66	**37.16**	36.78
15	32.08	**33.33**	33.13	32.41	**33.75**	33.52	30.34	**31.41**	30.98	30.60	**31.17**	30.95
25	30.04	**31.38**	31.32	30.12	**31.40**	31.34	27.95	**28.69**	28.19	27.95	**28.54**	28.28
σ	*Boat* (512 × 512)			*Stream* (512 × 512)			*Parrot* (256 × 256)			*Mandrill* (512 × 512)		
	Standard	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**
5	35.72	**36.57**	36.34	34.90	**35.66**	35.51	36.02	**37.73**	37.22	34.69	**35.16**	35.10
15	31.03	**31.63**	31.54	28.29	**28.76**	28.20	30.35	**31.23**	30.92	27.20	**28.05**	27.38
25	28.43	**29.12**	28.98	25.69	**26.18**	25.59	27.58	**28.62**	28.26	24.01	**25.12**	24.02

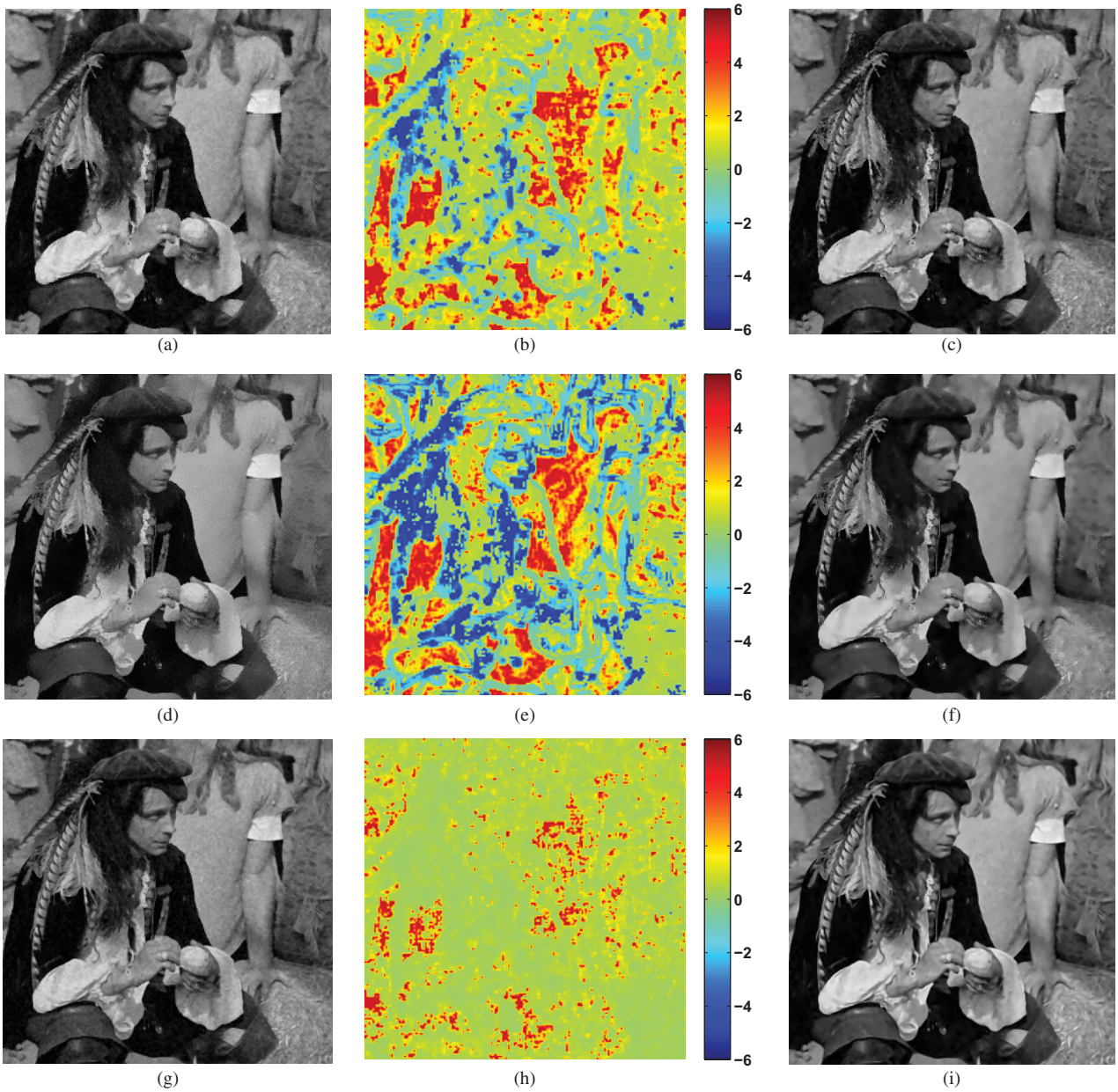


Fig. 9. Denoising example for the plug-in and SURE estimators with different kernels. AWGN with $\sigma = 25$ is added to *Man* image. (a) Bilateral kernel. (b) Iteration map for the Bilateral kernel. Colorbar: positive iteration numbers for diffusion and negative ones for boosting. (c) Plug-in estimator for Bilateral kernel. (d) NLM kernel. (e) Iteration map for the NLM kernel. (f) Plug-in estimator for NLM kernel. (g) LARK kernel. (h) Iteration map for the NLM kernel. (i) SURE estimator for LARK kernel.

where M is the number of computed estimates for the l-th pixel. This approach is adequate for the case when the estimated risk is unbiased (as in SURE). When using the plug-in estimator of risk, we must take bias into account with an exponential aggregator, described next.

2) Exponentially Weighted Aggregation: A way to take the bias into account is to consider the overall MSE rather than the variance. This has been studied in [22] and [23] where the exponentially weighted aggregation is introduced. The estimated risk associated with the $i\text{-}th$ estimate in the $j\text{-}th$ patch, r_{ij}, is computed by the plug-in estimator in (24) and (26). Thus, the l-th pixel has the following estimate:

$$\widehat{z}_l = \sum_{j=1}^{M} \frac{\widehat{z}_{ij} \exp(-r_{ij})}{\sum_{j=1}^{M} \exp(-r_{ij})} \tag{36}$$

where the confidence coefficients $\{\exp(-r_{ij})\}$, are the weights we use for the aggregation.

V. Experiments

In this section we evaluate performance of the proposed method for denoising various images. Since our method is motivated to improve performance of any kernel-based

TABLE IV
PERFORMANCE OF THE PLUG-IN ESTIMATOR FOR THE NLM KERNEL WITH DIFFERENT SMOOTHING PARAMETERS UNDER WGN CORRUPTION WITH $\sigma = 15$

h_y	*Peppers* (512 × 512)			*Lena* (512 × 512)			*Cameraman* (256 × 256)			*Man* (512 × 512)		
	Standard	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**
3	29.56	31.84	33.16	29.64	31.99	33.34	28.88	30.71	31.52	28.44	30.29	30.85
6	31.89	32.77	33.30	31.73	32.80	33.40	30.53	31.42	31.63	29.31	30.49	30.79
9	30.63	32.93	33.22	30.20	32.84	33.21	28.57	31.45	31.52	27.38	30.30	30.57
h_y	*Boat* (512 × 512)			*Stream* (512 × 512)			*Parrot* (256 × 256)			*Mandrill* (512 × 512)		
	Standard	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**	**Standard**	**SURE**	**Plug-in**
3	28.80	30.60	31.31	27.12	28.41	28.61	28.49	30.37	31.09	26.66	27.69	27.90
6	29.80	30.84	31.20	27.23	28.46	28.58	30.25	31.03	31.32	26.72	27.95	28.04
9	27.87	30.68	30.97	25.15	28.25	28.42	28.72	31.06	31.26	24.35	27.79	27.93

TABLE V
DENOISING PERFORMANCE OF SOME POPULAR METHODS (LPG-PCA [4], BM3-D [2]) UNDER WGN CORRUPTION, COMPARED WITH SAIF FOR THE LARK [9] AND NLM [10] KERNELS. RESULTS NOTED ARE AVERAGE PSNR (TOP) AND SSIM [24] (BOTTOM) OVER 10 INDEPENDENT NOISE REALIZATIONS FOR EACH σ

σ	*Man* (512 × 512)				*Parrot* (256 × 256)			
	LPG-PCA	**BM3D**	**SAIF** (LARK)	**SAIF** (NLM)	**LPG-PCA**	**BM3D**	**SAIF** (LARK)	**SAIF** (NLM)
5	37.13	**37.33**	37.16	37.07	37.76	**37.92**	37.73	37.66
	0.951	**0.956**	0.953	0.952	0.966	**0.967**	**0.967**	0.964
15	30.84	**31.24**	31.17	30.82	31.14	**31.43**	31.23	31.32
	0.850	**0.863**	0.860	0.847	0.892	**0.896**	0.890	0.890
25	28.25	**28.81**	28.54	28.19	28.59	**28.94**	28.62	28.87
	0.772	**0.794**	0.782	0.759	0.843	**0.850**	0.839	0.847
σ	*Stream* (512 × 512)				*Mandrill* (512 × 512)			
	LPG-PCA	**BM3D**	**SAIF** (LARK)	**SAIF** (NLM)	**LPG-PCA**	**BM3D**	**SAIF** (LARK)	**SAIF** (NLM)
5	35.62	**35.75**	35.66	35.55	**35.31**	35.25	35.16	35.05
	0.963	**0.965**	**0.965**	0.963	0.957	0.958	**0.959**	0.956
15	28.53	28.74	**28.76**	28.58	28.09	**28.17**	28.05	28.03
	0.835	0.846	**0.849**	0.834	0.832	0.843	**0.844**	0.837
25	25.86	**26.21**	26.18	25.88	25.16	**25.45**	25.12	25.13
	0.720	0.739	**0.743**	0.710	0.726	**0.746**	0.737	0.724

denoising, we first compare our results with ones from the three standard kernels: LARK, NLM and Bilateral. We also test stability of SAIF when an arbitrary tuning parameter is used. In all cases the proposed estimators show a promising improvement over the standard kernels. We will show that the resulting SAIF-ly improved filters are comparable, in terms of MSE (PSNR) and SSIM [24], to state-of-the-art denoising methods, and in many cases visually superior.

In our first simulations the patch size is set as 11×11 and in a Monte–Carlo simulation, 10 independent noise realizations were used. We varied k from 0 to 6 with 0.05 as the step size. As an initial guess for the smoothing parameters in the kernels, we set $h_x = 2\sqrt{2}$ and $h_y = 20\sqrt{2}\sigma$ in the Bilateral kernel, $h_y = 0.43\sigma$ in the NLM filter and the smoothing parameter in LARK [9] is fixed as 0.25σ.

Fig. 8 shows some benchmark images we used. Tables I, II and III show PSNR results of the standard kernel (fixed parameters in Bilateral, NLM, or LARK), SURE and the plug-in estimators. It can be seen that for Bilateral and Non-local means kernels, the plug-in estimator shows consistent improvement over both the standard estimate using the kernel, and the optimally iterated kernel from SURE. For the LARK kernel, the SURE method outperforms the plug-in estimator. Apparently, this performance difference occurs most noticeably for highly textured images such as Mandrill.

In Fig. 9 the two types of iterations, diffusion and boosting, are compared for the three kernels. The iteration map identifies the type and number of applied iterations on the patches of the image. The map colorbar represents positive and negative values for diffusion and boosting iterations, respectively. As can be seen, while diffusion recovers most of the flat and smooth patches, boosting takes care of the texture and more complex ones. It is worth noting that applying overlapped patches also has the advantage of computing multiple estimates for each pixel from the *both* iteration types. In other words, in the

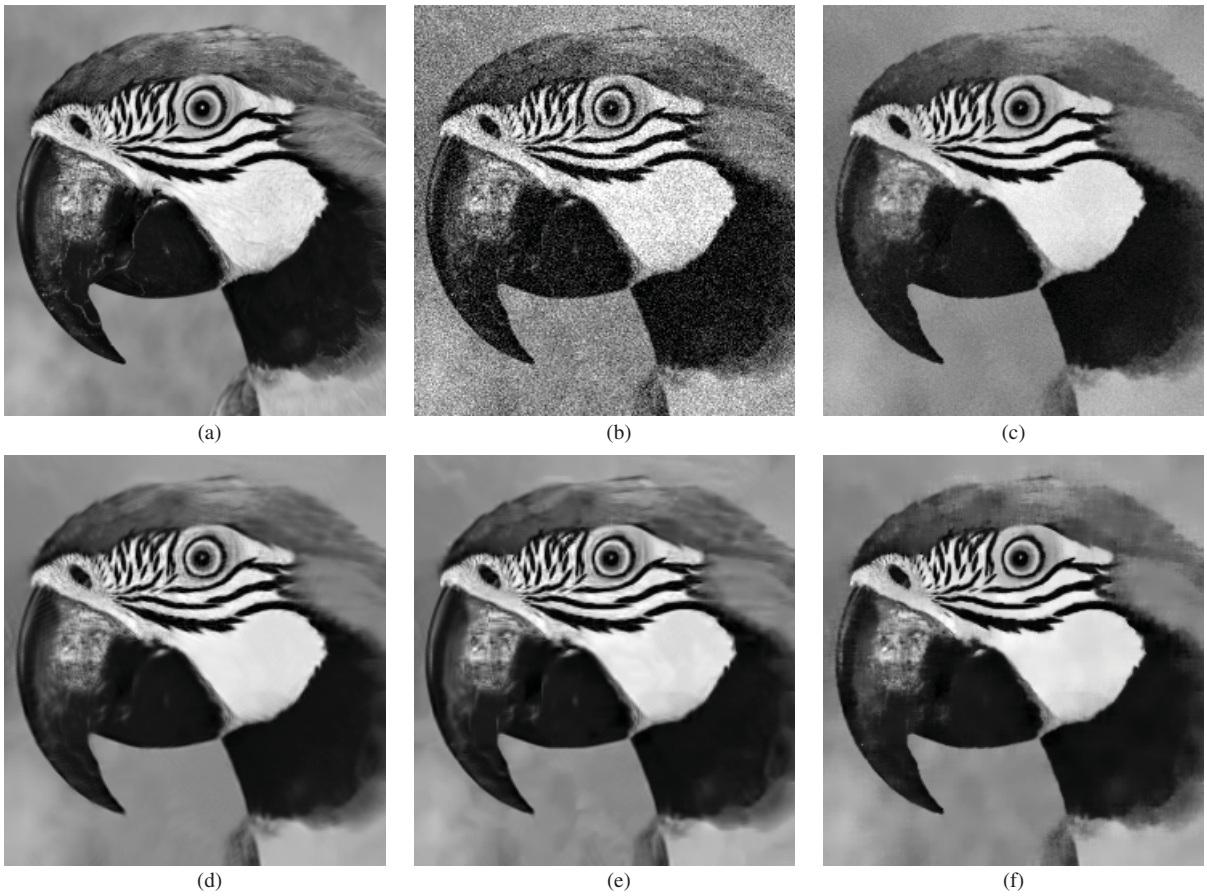


Fig. 10. Comparison of denoising performance on noisy *Parrot* image corrupted by AWGN of $\sigma = 25$. (a) Original image. (b) Noisy input. (c) NLM [10]. (d) LPG-PCA [4]. (e) BM3D [2]. (f) Proposed SAIF (NLM).

aggregation process, some pixels may be from diffusion in one patch, whereas others may be from boosting in another overlapping patch. This is especially useful for pixels at the border of smooth and texture regions.

The effect of the parameter tuning is studied in Table IV. In this set of simulations, NLM kernel weights with different control parameter, h_y, were computed for each image and then fed to the plug-in estimator. As can be seen, across a large range of h_y, performance of the proposed estimators is quite stable and shows consistent improvement over the baseline kernel. From these results, we can see that it is possible to improve the performance of the standard kernel with an arbitrary starting parameter by employing the proper number and type of iteration with the proposed MSE estimator.

Performances of the proposed SAIF algorithm and other methods are quantified across different noise levels in Table V, which shows that the proposed method is quantitatively quite comparable to LPG-PCA [4] and BM3D [2].

Fig. 10 demonstrates the denoising results of Parrot image obtained by different methods compared to the proposed method using the plug-in estimator. In addition to about 1 dB improvement over the baseline NLM in terms of PSNR, visual quality of the proposed method also is comparable and even superior to the state of the art denoising. In this case SAIF appears to recover some texture regions which were over-smoothed by BM3D and LPG-PCA. Fig. 11 shows performance of the SURE estimator for the Stream image. As can be seen, result by SAIF is visually close to BM3D and LPG-PCA. We note that our Matlab code and additional results are available at the project website[8].

In terms of computational complexity, denoising a 256×256 grayscale image with an unoptimized implementation of our method in Matlab take, on average, about 180 seconds. For such images, LPG-PCA (implemented by its authors for Matlab) take, on average, 280 seconds. BM3D, with its optimized implementation (implemented by the authors mostly in C and compiled as Mex for Matlab), takes significantly less time (about 1 second on average) for these images. However, our method is sped up significantly by reducing the amount of overlap between patches. For example, when estimating every fifth patch, our method requires only 120 seconds on average (including pre-filtering) with a minor drop in performance (about 0.1 dB).

VI. Conclusion

We have presented a framework for improved denoising by data-dependent kernels. Given any spatial domain filter, we can boost its performance to near state-of-the-art by employing

[8]Available at: http://www.soe.ucsc.edu/~htalebi/SAIF.php.

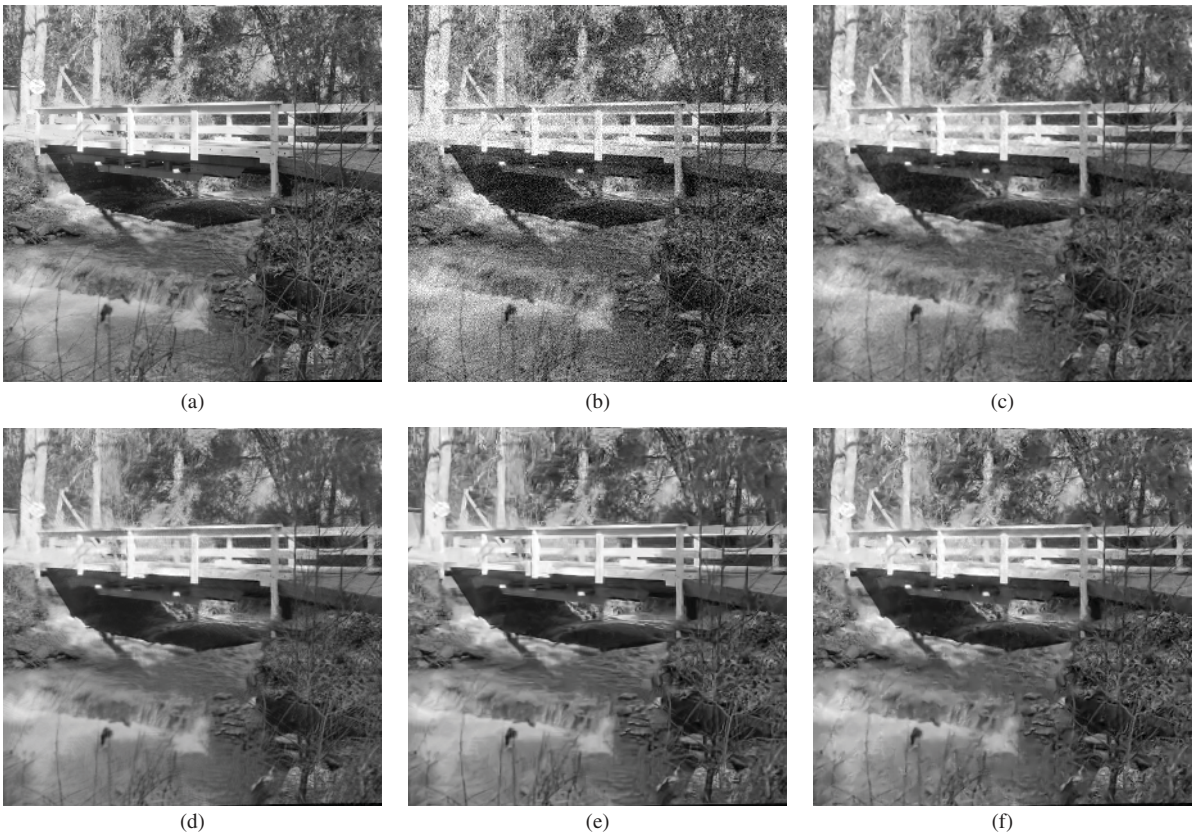


Fig. 11. Comparison of denoising performance on noisy parrot image corrupted by AWGN of $\sigma = 25$. (a) Original image. (b) Noisy input. (c) NLM [10]. (d) LPG-PCA [4]. (e) BM3D [2]. (f) Proposed SAIF (NLM).

optimized iteration methods. This iterative filtering is implemented patch-wise. Armed with diffusion and boosting as two *complementary* iteration techniques, each patch is filtered by the optimum local filter. More specifically, by exploiting the best iteration number and method which minimizes MSE in each patch, SAIF is capable of automatically adjusting the local smoothing strength according to local SNR. The experimental results demonstrate that the proposed approach improves the performance of kernel based filtering methods in terms of both PSNR (MSE) and subjective visual quality.

Using the estimated local SNR as empirical prior knowledge of the latent signal, we proposed the plug-in estimator which can outperform SURE estimator in many cases. Comparison of the plug-in and SURE motivates us to integrate them under a unique algorithm to automatically select the local estimator with minimum risk.

We assumed that the noise variance does not vary over the whole image. However, a local estimate of the noise variance leads to a better estimate of the local SNR. In other words, adding this feature to our proposed method can improve performance of both plug-in and SURE estimators.

APPENDIX A
APPROXIMATION OF THE DATA-DEPENDENT FILTER

By applying an effective pre-filtering, the filter weight matrix $\mathbf{W}$ is largely dependent on the *latent* image rather than the noisy input image [20]. To be more specific, consider the practical implementation of the filter as follows:

$$\widehat{\mathbf{z}} = \mathbf{W}(\widetilde{\mathbf{z}})\mathbf{y}. \tag{37}$$

Denote the estimate of the j-th pixel as:

$$\widehat{z}_j = \mathbf{w}_j^T(\widetilde{\mathbf{z}})\mathbf{y} \tag{38}$$

where $\mathbf{w}_j^T$ denotes the j-th row the filter $\mathbf{W}(\widetilde{\mathbf{z}})$. Assuming that the pre-filtered image is now only corrupted by a small additive noise: $\widetilde{\mathbf{z}} = \mathbf{z} + \boldsymbol{\eta}$, we can make the following first order Taylor approximation:

$$\widehat{z}_j = \mathbf{w}_j^T(\widetilde{\mathbf{z}})\mathbf{y} \approx \mathbf{w}_j^T(\mathbf{z})\mathbf{y} + \boldsymbol{\eta}^T\mathbf{G}_j\mathbf{y} \tag{39}$$

where the $n \times n$ diagonal matrix $\mathbf{G}_j$ contains the vector $\nabla\mathbf{w}_j$ along its diagonal entries, in which $\nabla\mathbf{w}_j(\mathbf{z})$ denotes the gradient of the vector $\mathbf{w}_j$ with respect to its argument. The first term in the above ($\mathbf{w}_j^T(\mathbf{z})\mathbf{y}$) is the *oracle* filter. The second term is the error between the oracle and the practical filter:

$$\triangle_j = \mathbf{w}_j^T(\widetilde{\mathbf{z}})\mathbf{y} - \mathbf{w}_j^T(\mathbf{z})\mathbf{y} \approx \boldsymbol{\eta}^T\mathbf{G}_j\mathbf{y}. \tag{40}$$

As can be seen, this error is dependent on the quality of the pre-filter (how small is $\boldsymbol{\eta}$) and also smoothness of the kernel ($\nabla\mathbf{w}_j$). We note that this gradient refers to the shape of the baseline kernel, and the way it depends on its argument (Gaussian being typical) and not on the actual underlying image. While a good choice of pre-filter can sufficiently suppress the noise, tuning the smoothness parameter of the

baseline kernel guarantees the gradient of the filter to be small. As a result, we can be assured that our approximation is reliable for the performance analysis in the paper.

APPENDIX B
MEAN-SQUARED ERROR OF THE PLUG-IN AND SURE ESTIMATORS

Here we derive an expression for expected error of each risk estimator and then use this to compare the plug-in and SURE estimators. We start from the expression for diffusion plug-in risk estimator:

$$\text{Plug-in}_k^{df} = \sum_{i=1}^{n}(1-\lambda_i^k)^2\widetilde{b_i^2} + \sigma^2\lambda_i^{2k}. \tag{41}$$

We assume that $\widetilde{b_i} = b_i + \eta_i$, where η_i is the projected noise of the pre-filtered image on the eigenvectors of the filter, where η_i are AWGN with mean zero and covariance $\nu^2\mathbf{I}$. The expected value of eq. 41 can be expressed as:

$$\text{E}[\text{Plug-in}_k^{df}] = \sum_{i=1}^{n}(1-\lambda_i^k)^2(b_i^2+\nu^2) + \sigma^2\lambda_i^{2k}. \tag{42}$$

Consequently the bias of the plug-in estimator can be written as:

$$\text{bias}(\text{Plug-in}_k^{df}) = \nu^2\sum_{i=1}^{n}(1-\lambda_i^k)^2. \tag{43}$$

The variance term also is:

$$\text{var}(\text{Plug-in}_k^{df}) = 2\nu^2\sum_{i=1}^{n}(1-\lambda_i^k)^4(\nu^2+2b_i^2). \tag{44}$$

Then the MSE of the Plug-in_k^{df} estimator is as follows:

$$\begin{aligned}\text{MSE}_{\text{Plug-in}_k^{df}} &= \nu^4\Big(\sum_{i=1}^{n}(1-\lambda_i^k)^2\Big)^2 \\ &\quad + 2\nu^2\sum_{i=1}^{n}(1-\lambda_i^k)^4(\nu^2+2b_i^2).\end{aligned} \tag{45}$$

Unsurprisingly, as ν decreases, MSE of the estimator tends to zero. We can also derive MSE of the SURE estimator in (28) by replacing $\breve{b}_i$ with $b_i + e_i$ where $\mathbf{e} = \mathbf{N}(\mathbf{0}, \sigma^2\mathbf{I})$. Then the variance term and the MSE are:

$$\text{MSE}_{\text{SURE}_k^{df}} = 2\sigma^2\sum_{i=1}^{n}(1-\lambda_i^k)^4(\sigma^2+2b_i^2). \tag{46}$$

Comparison of (45) and (46) can determine the better risk estimator. To accomplish this comparison, we aim to define an upper bound for the error incurred by the plug-in estimator risk. Applying the Cauchy-Schwartz inequality to the squared bias term in (45):

$$\nu^4\Big(\sum_{i=1}^{n}(1-\lambda_i^k)^2\Big)^2 \le n\nu^4\sum_{i=1}^{n}(1-\lambda_i^k)^4. \tag{47}$$

Then we have:

$$\text{MSE}_{\text{Plug-in}_k^{df}} \le \sum_{i=1}^{n}(1-\lambda_i^k)^4\left((n+2)\nu^4 + 4\nu^2b_i^2\right). \tag{48}$$

Defining $\beta = \frac{\nu^2}{\sigma^2}$ as relative variance of the noise in the pre-filtered and noisy signal, comparison of (46) and (48) shows that the plug-in estimator outperforms SURE when in each channel i:

$$\text{snr}_i \ge \frac{\beta^2(n+2)-2}{4(1-\beta)} \tag{49}$$

where $\text{snr}_i = \frac{b_i^2}{\sigma^2}$ is the signal-to-noise ratio of channel i. Similarly for the boosting iteration the MSE of the plug-in estimator is:

$$\begin{aligned}\text{MSE}_{\text{Plug-in}_k^{bs}} &= \nu^4\left(\sum_{i=1}^{n}(1-\lambda_i)^{2k+2}\right)^2 \\ &\quad + 2\nu^2\sum_{i=1}^{n}(1-\lambda_i)^{4k+4}(\nu^2+2b_i^2)\end{aligned} \tag{50}$$

and also for the boosting SURE estimator in (20) we have:

$$\text{MSE}_{\text{SURE}_k^{bs}} = 2\sigma^2\sum_{i=1}^{n}(1-\lambda_i)^{4k+4}(\sigma^2+2b_i^2). \tag{51}$$

Doing the same analysis as we did for the diffusion iteration, the given constraint in (49) is obtained again for the boosting iteration.

APPENDIX C
SENSITIVITY OF THE PLUGIN AND SURE ESTIMATORS

What we study here is the sensitivity of each estimator to the baseline kernel type. Assuming the MSE expressions as functions of the filter eigenvalues $\{\lambda_i\}$, their derivatives can explain the sensitivity of the risk to the filter (kernel) type. Defining $\varepsilon_i = (1-\lambda_i^k)^2$, we have

$$\frac{\partial\text{MSE}_{\text{Plug-in}_k^{df}}}{\partial\varepsilon_j} = 2\nu^4\sum_{i=1}^{n}\varepsilon_i + (4\nu^4 + 8\nu^2b_j^2)\varepsilon_j \tag{52}$$

and also for the SURE risk estimator given by (46) the derivative is

$$\frac{\partial\text{MSE}_{\text{SURE}_k^{df}}}{\partial\varepsilon_j} = (4\sigma^4 + 8\sigma^2b_j^2)\varepsilon_j. \tag{53}$$

As $\varepsilon_j \to 1$, we can see that the plug-in sensitivity is bounded as:

$$\lim_{\varepsilon_j\to 1}\frac{\partial\text{MSE}_{\text{Plug-in}_k^{df}}}{\partial\varepsilon_j} \le 2(n+2)\nu^4 + 8\nu^2b_j^2 \tag{54}$$

and also for the SURE estimator we have:

$$\lim_{\varepsilon_j\to 1}\frac{\partial\text{MSE}_{\text{SURE}_k^{df}}}{\partial\varepsilon_j} = (4\sigma^4 + 8\sigma^2b_j^2). \tag{55}$$

Comparison of (54) and (55) shows that when $\lambda_i \to 0$, as long as (49) holds, the plug-in estimator is less sensitive than SURE.

On the other hand as $\varepsilon_j \to 0$ the SURE estimator's derivative tends to 0 and yet the sensitivity of the plug-in

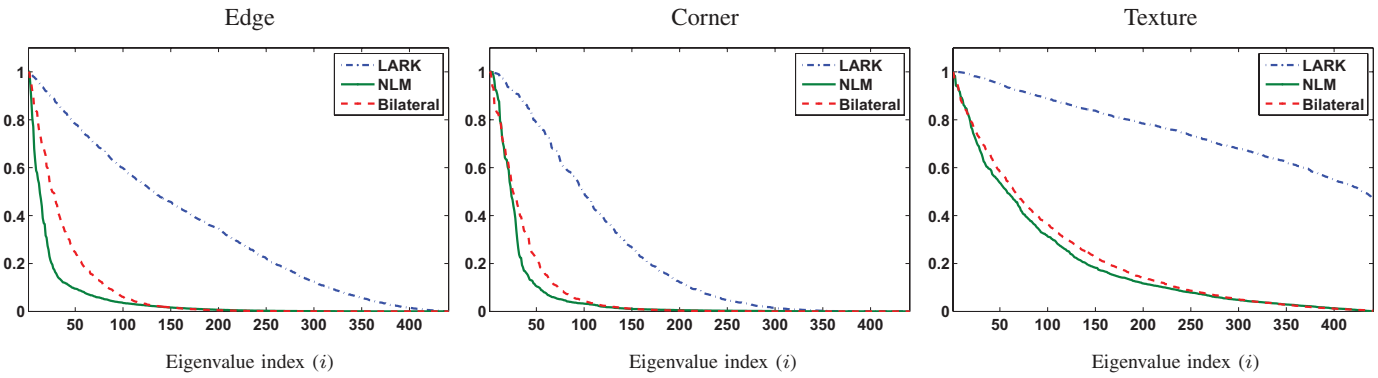


Fig. 12. Spectrum of filters computed from the patches in Fig. 3. Among the three kernels, LARK eigenvalues are larger than NLM and Bilateral.

method remains dependent on the other eigenvalues of the filter:

$$\lim_{\varepsilon_j \to 0} \frac{\partial \text{MSE}_{\text{Plug-in}_k^{df}}}{\partial \varepsilon_j} = 2\nu^4 \sum_{i=1}^{n} \varepsilon_i. \tag{56}$$

This, in particular shows that the SURE estimator is more reliable when the base kernel has eigenvalues $\{\lambda_i\}$ closer to one. The LARK filter [9] is an example of this type of kernels with less aggressive spectrum than NLM [10] and Bilateral [8]. Fig. 12 compares spectrum of the three kernels for the selected patches in Fig. 3. As can be seen, the LARK spectrum has eigenvalues that are larger than the ones from NLM and Bilateral kernels for all the tested patches. Overall, we can conclude that the plug-in estimator better fits aggressive kernel bases like NLM and Bilateral.

Sensitivity analysis and results of the boosting iteration are similar to the presented diffusion process. The only difference is that the derivation variable ε_i should be defined as $(1-\lambda_i)^{2k+2}$.

REFERENCES

[1] J. Portilla, V. Strela, M. Wainwright, and E. P. Simoncelli, "Image denoising using scale mixtures of Gaussians in the wavelet domain," *IEEE Trans. Image Process.*, vol. 12, no. 11, pp. 1338–1351, Nov. 2003.

[2] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-D transform-domain collaborative filtering," *IEEE Trans. Image Process.*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007.

[3] D. D. Muresan and T. W. Parks, "Adaptive principal components and image denoising," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2003, pp. 101–104.

[4] L. Zhang, W. Dong, D. Zhang, and G. Shi, "Two-stage image denoising by principal component analysis with local pixel grouping," *Pattern Recognit.*, vol. 43, pp. 1531–1549, Apr. 2010.

[5] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3736–3745, Dec. 2006.

[6] P. Chatterjee and P. Milanfar, "Clustering-based denoising with locally learned dictionaries," *IEEE Trans. Image Process.*, vol. 18, no. 7, pp. 1438–1451, Jul. 2009.

[7] P. Chatterjee and P. Milanfar, "Is denoising dead?" *IEEE Trans. Image Process.*, vol. 19, no. 4, pp. 895–911, Apr. 2010.

[8] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jan. 1998, pp. 836–846.

[9] H. Takeda, S. Farsiu, and P. Milanfar, "Kernel regression for image processing and reconstruction," *IEEE Trans. Image Process.*, vol. 16, no. 2, pp. 349–366, Feb. 2007.

[10] A. Buades, B. Coll, and J. M. Morel, "A review of image denoising algorithms, with a new one," *Multiscale Model. Simulat., Int. J.*, vol. 4, no. 2, pp. 490–530, 2005.

[11] C. Kervrann and J. Boulanger, "Optimal spatial adaptation for patch-based image denoising," *IEEE Trans. Image Process.*, vol. 15, no. 10, pp. 2866–2878, Oct. 2006.

[12] J. Boulanger, C. Kervrann, and P. Bouthemy, "Space-time adaptation for patch-based image sequence restoration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1096–1102, Jun. 2007.

[13] P. Milanfar, "A tour of modern image filtering," *IEEE Signal Process. Mag.*, to be published.

[14] S. Ramani, T. Blu, and M. Unser, "Monte-Carlo SURE: A black-box optimization of regularization parameters for general denoising algorithms," *IEEE Trans. Imag. Process.*, vol. 17, no. 9, pp. 1540–1554, Sep. 2008.

[15] X. Zhu and P. Milanfar, "Automatic parameter selection for denoising algorithms using a no-reference measure of image content," *IEEE Trans. Imag. Process.*, vol. 19, no. 12, pp. 3116–3132, Dec. 2010.

[16] C. Stein, "Estimation of the mean of a multivariate normal distribution," *Ann. Statist.*, vol. 9, no. 6, pp. 1135–1151, Nov. 1981.

[17] M. Van De Ville and D. Kocher, "Nonlocal means with dimensionality reduction and sure-based parameter selection," *IEEE Trans. Imag. Process.*, vol. 20, no. 9, pp. 2683–2690, Sep. 2011.

[18] E. Seneta, *Non-Negative Matrices and Markov Chains*. Berlin, Germany: Springer-Verlag, 1981.

[19] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge, MA: Cambridge Univ. Press, 1991.

[20] P. Milanfar, "Symmetrizing smoothing filters," *SIAM J. Imag. Sci.*, to be published.

[21] H. Talebi and P. Milanfar, "Improving denoising filters by optimal diffusion," in *Proc. ICIP, Appear Conf.*, 2012, pp. 1–4.

[22] J. Salmon and E. Le Pennec, "NL-Means and aggregation procedures," in *Proc. ICIP Conf.*, Nov. 2009, pp. 2977–2980.

[23] A. S. Dalalyan and A. B. Tsybakov, "Aggregation by exponential weighting, sharp pac-bayesian bounds and sparsity," *Mach. Learn.*, vol. 72, nos. 1–2, pp. 39–61, 2008.

[24] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Imag. Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

Hossein Talebi (S'11) received the B.S. and M.S. degrees from the Isfahan University of Technology, Isfahan, Iran, in 2007 and 2010, respectively. He is currently pursuing the Ph.D. degree with the University of California, Santa Cruz, all in electrical engineering.

He is currently a Researcher with the Multi-Dimensional Signal Processing Laboratory (MDSP Lab), University of California. His current research interests include image and video processing (i.e., denoising, interpolation, and superresolution).

Xiang Zhu (S'08)received the B.S. and M.S. degrees from Nanjing University, Nanjing, China, in 2005 and 2008, respectively. He is currently pursuing the Ph.D. degree with the University of California, Santa Cruz, all in electrical engineering.

His current research interests include the domain of image processing (denoising, deblurring, super-resolution, and image quality assessment) and computer vision.

Peyman Milanfar (F'10) received the B.S. degree in electrical engineering and mathematics from Berkeley, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge. He was with SRI, and was a Consulting Professor of computer science at Stanford. He is a Professor of electrical engineering with University of California, Santa Cruz, and was an Associate Dean of research from 2010 to 2012. He is currently on leave at Google-[x]. He founded MotionDSP, which has brought state-of-the-art video enhancement to the market. His technical expertises are in statistical signal, image and video processing, computational photography, and vision.

Dr. Milanfar is a member of the IEEE Signal Processing Society's IVMSP Technical Committee, and its Awards Board.