Manuscript submitted to
AIMS' Journals
Volume **X**, Number **0X**, XX **200X**

doi:10.3934/xx.xx.xx.xx

pp. **X–XX**

A FAST PATCH-DICTIONARY METHOD FOR WHOLE IMAGE RECOVERY

Yangyang Xu

Department of Computational and Applied Mathematics, Rice University
Houston, TX 77005, USA

Wotao Yin

Department of Mathematics, University of California, Los Angeles, CA 90095, USA

Abstract. Various algorithms have been proposed for dictionary learning. Among those for image processing, many use *image patches* to form dictionaries. This paper focuses on the image recovery from corrupted linear measurements, each of which may encode the whole image. We address the open issue of representing an image by *overlapping* patches: the overlapping leads to an excessive number of dictionary coefficients to determine, which are often much more than the number of pixels. With very few exceptions, this issue has limited the applications of image-patch methods to the "local" kind of tasks such as denoising, inpainting, cartoon-texture decomposition, super-resolution, and image deblurring, for which one can process a few patches at a time. Our focus is global imaging tasks such as compressive sensing and medical image recovery, where the whole image is encoded together, making it either impossible or very ineffective to update a few patches at a time.

Our strategy is to divide the sparse recovery into multiple subproblems, each of which handles a subset of non-overlapping patches, and then the results of the subproblems are averaged to yield the final recovery. This simple strategy is surprisingly effective in terms of both quality and speed.

In addition, we accelerate computation of the learned dictionary by applying a recent block proximal-gradient method, which not only has a lower per-iteration complexity but also takes fewer iterations to converge, compared to the current state-of-the-art. We also establish that our algorithm globally converges to a stationary point. Numerical results on synthetic data demonstrate that our algorithm can recover a more faithful dictionary than two state-of-the-art methods.

Combining our whole-image recovery and dictionary-learning methods, we numerically simulate image inpainting, compressive sensing recovery, and deblurring. Our recovery is more faithful than those of a total variation method and a method based on overlapping patches. Our matlab code is competitive in terms of both speed and quality.

1. Introduction.

Our general problem is to restore an image $\mathbf{M}$ from its corrupted linear measurements in the form of $\mathbf{b} = \mathcal{A}(\mathbf{M}) + \boldsymbol{\xi}$, where $\mathcal{A}$ is a linear operator and $\boldsymbol{\xi}$ is some noise. Examples of such recovery include image denoising

2010 *Mathematics Subject Classification.* Primary: 94A08, 94A12; Secondary: 90C90.

Key words and phrases. dictionary learning, sparse representation, non-overlapping patches, whole-image recovery, non-convex optimization.

Y. Xu is supported by NSF grant ECCS-1028790. W. Yin is partially supported by NSF grants DMS-0748839 and DMS-1317602, and ARO/ARL MURI grant FA9550-10-1-0567.

($\mathcal{A}$ equals the identity operator $\mathcal{I}$), super-resolution ($\mathcal{A}$ is a downsampling operator), image deblurring ($\mathcal{A}$ is a blurring operator), compressive imaging recovery ($\mathcal{A}$ is a compressed sensing operator), as well as medical imaging recovery ($\mathcal{A}$ can be a downsampled Fourier or a Radon operator, for example).

This paper restores the image $\mathbf{M}$ by computing its sparse representation under a learned dictionary. Following the approach pioneered in [7], we numerically form a dictionary that sparsely represents each and all the overlapping *patches* of $\mathbf{M}$. Given such a dictionary $\mathbf{D}$, we reconstruct the image patches by finding their sparse coefficients and then recover the image from the patches.

We address an open issue regarding *whole–image* recovery: the large number of overlapping patches lead to a large number of free coefficients in the recovery, which can cause overfitting and slow computation. This issue has limited most of the patch-based methods (with a few exceptions we shall review below) to the “local” or “nearly local” kinds of image processing tasks such as denoising, inpainting, deblurring, and super-resolution. For these tasks, one or a few patches can be processed at a time, independently of the majority of the remaining patches, thus avoiding the overfitting issue. We, however, consider the more difficult “global” kind of task such as compressive sensing recovery, where each piece of the measurements encodes the whole image and thus it is either impossible or very ineffective to process one or a few patches at a time.

Bearing this issue in mind, we *do not* process either one patch at a time or all the overlapping patches at once, but instead we process one subset of *non-overlapping, covering* patches at a time. (Covering means that the subset of patches covers all the pixels of the image.) Each time, we process this subset of patches and obtain a recovery of the whole image. After we process multiple different subsets of *non-overlapping, covering* patches, we obtain multiple whole-image recoveries, whose average is taken to eliminate the grid artifact that might exist in the individual ones. This simple strategy is surprisingly effective. Computationally, the different subsets of patches can be processed in parallel, and we found using merely five different subsets is enough to remove the grid artifact. For each subset, the corresponding ℓ_1 minimization problem is rather small: if 8×8 patches are used, it only has roughly $1/64$ of the free variables that one would have if all the overlapping patches are processed at once. Qualitatively, the averaged recovery has a higher PSNR than other state-of-the-art approaches that address the overfitting issue by incorporating additional image structures.

We also introduce a fast algorithm for learning the dictionary $\mathbf{D}$, which plays a vital role in both our proposed recovery method and others. Here, $\mathbf{D}$ can be pre-learned from a set of similar images, and then either fixed during the recovery or iteratively updated in adaptive to the image under recovery. Following [7], after recovering an image, we update the dictionary to fit the recovered image by solving an ℓ_1-regularized model. We introduce an algorithm to update dictionary $\mathbf{D}$ and sparse coefficient $\mathbf{Y}$ alternatively. Unlike existing algorithms (e.g. [8, 1]), it does not exactly minimize over either $\mathbf{D}$ or $\mathbf{Y}$, yet it decreases the energy very fast and provably converges to a stationary solution. Our code and several demos can be downloaded from our websites. Before giving more details of our approach and its numerical results, we first review the related literature.

1.1. Image recovery by dictionary.

Various methods have been developed to restore an image from its corrupted and/or incomplete measurements. One popular class of recovery methods are based on sparse coding and dictionary such as those in [7, 2, 18]. We say a signal $\mathbf{x} \in \mathbb{R}^n$ is sparse (or approximately sparse) under a dictionary $\mathbf{D} \in \mathbb{R}^{n\times K}$ if $\mathbf{x} = \mathbf{D}\mathbf{y}$ (or $\mathbf{x} \approx \mathbf{D}\mathbf{y}$) and $\mathbf{y} \in \mathbb{R}^K$ has only a few nonzeros. Many types of signals can be sparsely represented by some dictionary. For example, natural images are approximately sparse under dictionaries based on various wavelet, curvelet, shearlet, and other transforms. Suppose $\mathbf{x}$ has a sparse representation under a dictionary $\mathbf{D}$. Then given $\mathbf{D}$ and linear measurements $\mathbf{b} = \mathcal{A}(\mathbf{x}) + \boldsymbol{\xi}$, one can recover $\mathbf{x}$ through sparsely coding $\mathbf{x}$ via solving

$$\min_{\mathbf{y}} \|\mathbf{y}\|_0, \text{ subject to } \|\mathcal{A}(\mathbf{D}\mathbf{y}) - \mathbf{b}\|_2^2 \le \epsilon, \tag{1}$$

where $\|\cdot\|_0$ counts the nonzero number of its argument and is often approximated by $\|\cdot\|_1$ for tractable computation, and $\epsilon \ge 0$ is a parameter corresponding to $\boldsymbol{\xi}$. Once a solution $\mathbf{y}$ of (1) is obtained, the original signal $\mathbf{x}$ can be estimated by $\mathbf{D}\mathbf{y}$. The dictionary $\mathbf{D}$ can be either predetermined or learned from a set of training data. Predetermined dictionaries, such as orthogonal or overcomplete wavelets, curvelets, and discrete cosine transforms (DCT), can have advantages of fast implementation over a learned one. Assuming easy availability of training datasets, however, it has been demonstrated (e.g., in [10, 7, 17]) that a learned dictionary can better adapt to natural signals and improve the recovery quality.

For natural images, existing methods such as MOD [8] and KSVD [1] learn a dictionary $\mathbf{D}$ to sparsely represent the patches of an image, rather than the whole image itself. In other words, the size of dictionary atoms is the same as that of the image patches, for example, 6×6 or 8×8. To denoise an image $\mathbf{M}$ with a patch-size dictionary, the pioneering work [7] denoises each of the overlapping patches of $\mathbf{M}$ via sparse coding and then estimates $\mathbf{M}$ as the average of all the denoised patches together with the observed noisy image. This patch-based method was then extended to compressed sensing MRI – a whole-image recovery problem – in [18], which starts from a rough estimate of $\mathbf{M}$, then simultaneously updates dictionary $\mathbf{D}$ and sparse coefficients of all overlapping patches, and finally averages all the recovered patches to estimate $\mathbf{M}$. Dong et al. in [6] use local dictionaries to sparsely represent local patches and incorporate additional local auto-regression (AR) and non-local similarity (NLS) terms to reduce overfitting and improve recovery results. Their model was demonstrated effective on image debluring and super-resolution. These and their follow-up works (e.g., [9, 25]) use overlapping patches since tiling non-overlapping patches can cause visible grid artifact along the patch boundaries, which is avoided by using overlapping patches.

1.2. **Learn a dictionary.** Due to a lack of analytic structures, it can be computationally demanding to learn a dictionary. One of the most popular algorithms for dictionary learning is KSVD [1]:

$$\min_{\mathbf{D},\mathbf{Y}} \|\mathbf{D}\mathbf{Y} - \mathbf{X}\|_F^2, \text{ subject to } \|\mathbf{d}_i\|_2 = 1, i = 1, \ldots, K;\ \|\mathbf{y}_j\|_0 \le s, j = 1, \ldots, p, \tag{2}$$

where $\mathbf{X} \in \mathbb{R}^{n\times p}$ is the training dataset, $\|\cdot\|_2$ denotes the Euclidean norm, s is a parameter to control sparsity, and $\mathbf{d}_i$ is the ith column of $\mathbf{D}$. KSVD attempts to solve (2) by alternatively updating $\mathbf{Y}$ and $\mathbf{D}$ in a certain way. The objective is monotonically non-increasing and the denoising and inpainting performances are very good, but the convergence to a stationary point is not guaranteed. Furthermore, it is slow as it performs SVD to update $\mathbf{D}$ and exact minimization to update every $\mathbf{y}_j$ in each iteration.

Another popular method is the online dictionary learning (OLM) [14], which, via an online update approach, attempts to solve

$$\min_{\mathbf{D},\mathbf{Y}} \frac{1}{2}\|\mathbf{DY}-\mathbf{X}\|_F^2+\lambda\|\mathbf{Y}\|_1, \text{ subject to } \|\mathbf{d}_i\|_2\le 1, i=1,\ldots,K, \tag{3}$$

where $\|\mathbf{Y}\|_1=\sum_{i,j}|y_{ij}|$ is a convex relaxation of $\|\cdot\|_0$, and λ is a tuning parameter to balance data fitting and sparsity level. OLM alternatively updates $\mathbf{Y}$ and $\mathbf{D}$ as follows. When $\mathbf{D}$ is fixed, it randomly picks a batch of columns of $\mathbf{X}$ and applies sparse coding to each selected column. Letting S be the index set of all previously selected samples and $\mathbf{Y}_S$ contain their sparse coefficients, the method then updates $\mathbf{D}$ to the solution of $\min_{\mathbf{D}}\{\|\mathbf{DY}_S-\mathbf{X}_S\|_F^2, \|\mathbf{d}_i\|_2\le 1, \forall i\}$, where $\mathbf{X}_S$ denotes the submatrix consisting of all columns of $\mathbf{X}$ indexed by S. The above two steps are then repeated until convergence. The algorithm often runs faster than KSVD, and its efficiency relies on the assumption that all training samples have the same distribution. Assuming that the training data admits bounded probability with a compact support and $\mathbf{Y}_S\mathbf{Y}_S^\top$ is uniformly positive definite, it is shown that the iterate sequence asymptotically satisfies the first-order optimality condition of (3). The global convergence of the iterate sequence is still open.

We refer the interested readers to the review paper [20] for other dictionary learning methods. In addition, more complicated models have been proposed to learn dictionaries for specific tasks; see [15, 13] for example. We do not intend to consider those models and will keep our focus on (3) in this paper.

1.3. Contributions.

This paper makes the following contributions:

- We propose a simple, novel method that recovers a whole image by applying sparse coding to its patches. In addition to the traditional denoising, inpainting, and deblurring tasks, the method can be applied to recovering an image from its whole-image linear measurements, which arise in the applications of compressive sensing and medical imaging. The method is simple and can include additional energy terms and constraints, as well as to be embedded in more complicated imaging applications. We want to emphasize that our method recovers the whole image at a time and is different from local recovery methods such as those in [7, 14] which process image patches one by one.
- Along with the method, we introduce a numerical algorithm for dictionary learning that is fast and has provable convergence to a stationary point. The algorithm is based on our recent work on block proximal gradient update in [22]. Compared to the existing algorithms, the proposed algorithm has a low per-iteration cost and converges fast.
- We provide Matlab codes for three different imaging tasks that are (i) inpainting: fill in image missing pixels; (ii) compressive sensing recovery: recover an image from its undersampled linear measurements; (iii) image deblurring: restore a clean image from its blurs. On these tasks, our codes compare favorably to total variation (TV) methods, as well as those from [14, 6] using overlapping patches and learned dictionaries.

1.4. Organization.

The rest of the paper is organized as follows. In section 2, we give a new model for recovering an image from its linear measurements, and also discuss how to improve recovery results. Section 3 applies a block proximal gradient method to (3) and makes a new dictionary learning algorithm. Numerical results are reported in section 4, and finally section 5 concludes the paper.

2. Problem formulation.

Given a patch-size dictionary $\mathbf{D}$, we aim at recovering an image $\mathbf{M}$ from its corrupted linear measurements $\mathbf{b} = \mathcal{A}(\mathbf{M}) + \boldsymbol{\xi}$, where $\mathcal{A}$ is a linear operator and $\boldsymbol{\xi}$ is some noise. The case of $\mathcal{A} = \mathcal{I}$ has been considered in the pioneering work [7], which alternatively performs sparse coding to denoise every patch and takes average over all overlapping denoised patches together with the observed noisy image.

Throughout the discussion in the remaining part of the paper, we assume that a generic image has size $N_1 \times N_2$ and training patches to be $n_1 \times n_2$. The dictionary $\mathbf{D}$ has K atoms, and all of them are vectors in $n_1 n_2$ dimensional space. Keep in mind that an $m \times n$ matrix is equivalent to an $m \cdot n$ vector under Matlab's `reshape` operation. Hence, we will use a matrix and its reshaped vector interchangeably. For example, a dictionary atom can be regarded as either a vector of length $n_1 n_2$ or an $n_1 \times n_2$ patch.

2.1. Our model.

Motivated by [7], we exactly represent an image by

$$\mathbf{M} = \left(\mathcal{T}_P\right)^{-1}\Big(\sum_{(i,j)\in P} \mathcal{R}_{ij}^\top(\mathbf{D}\mathbf{y}_{ij})\Big), \quad \mathcal{T}_P := \sum_{(i,j)\in P} \mathcal{R}_{ij}^\top \mathcal{R}_{ij}, \tag{4}$$

where $\mathcal{R}_{ij}$ is the operator extracting the (i,j)-th patch (*not* the (i,j)-th pixel), $\mathcal{R}_{ij}^\top$ is the adjoint of $\mathcal{R}_{ij}$, and P contains either all the patches, or a subset of the patches, that cover all the pixels of $\mathbf{M}$, ensuring that $\mathcal{T}_P$ is invertible. Note that $\mathcal{T}_P$ is diagonal, and thus its inverse can be implemented in a pixel-by-pixel manner. A pixel is overlappingly included in multiple patches. The value of a pixel is the average of its values in those overlapping patches. Each $\mathbf{D}\mathbf{y}_{ij}$ contributes to the representation of the (i,j)-th patch of the image, and it may or may not equal the patch.

Using this representation, we make the following weighted ℓ_1 model:

$$\min_{\mathbf{y}} \sum_{(i,j)\in P} \|\mathbf{w}_{ij} \odot \mathbf{y}_{ij}\|_1, \text{ subject to } \Big\|\mathcal{A}\mathcal{T}_P^{-1}\Big(\sum_{(i,j)\in P} \mathcal{R}_{ij}^\top(\mathbf{D}\mathbf{y}_{ij})\Big) - \mathbf{b}\Big\|_2 \le \sigma, \tag{5}$$

where $\mathbf{w}_{ij} \ge 0$ is a weight vector for $(i,j) \in P$, σ is the noise level determined by $\boldsymbol{\xi}$, and "$\odot$" denotes component-wise product. Equivalently, one can consider the unconstrained model:

$$\min_{\mathbf{y}} \sum_{(i,j)\in P} \|\mathbf{w}_{ij} \odot \mathbf{y}_{ij}\|_1 + \frac{1}{2\nu}\Big\|\mathcal{A}\mathcal{T}_P^{-1}\Big(\sum_{(i,j)\in P} \mathcal{R}_{ij}^\top(\mathbf{D}\mathbf{y}_{ij})\Big) - \mathbf{b}\Big\|_2^2, \tag{6}$$

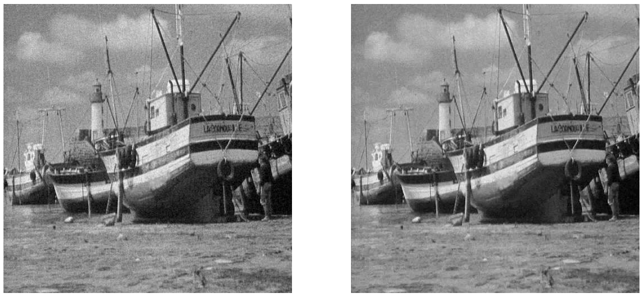
where ν is a parameter corresponding to σ. Upon solving (5) or (6), one can use $\mathcal{T}_P^{-1} \sum_{(i,j)\in P} \mathcal{R}_{ij}^\top(\mathbf{D}\mathbf{y}_{ij})$ to estimate $\mathbf{M}$.

Remark 1. Our models are similar to that in [6]:

$$\begin{aligned}\min_{\mathbf{y}} \sum_{(i,j)\in S} \|\mathbf{y}_{ij}\|_1 &+ \tfrac{1}{2\nu}\left\|\mathcal{A}\left(\Big(\sum_{(i,j)\in S} \mathcal{R}_{ij}^\top \mathcal{R}_{ij}\Big)^{-1}\Big(\sum_{(i,j)\in S} \mathcal{R}_{ij}^\top(\mathbf{D}_{k_{ij}}\mathbf{y}_{ij})\Big)\right) - \mathbf{b}\right\|_2^2 \\ &+ \mathrm{AR}(\mathbf{y}) + \mathrm{NLS}(\mathbf{y}),\end{aligned} \tag{7}$$

where S denotes the set of all overlapping patches, ν is a parameter balancing sparsity and data fitting, $\mathbf{D}_{k_{ij}}$ is a given local dictionary used to represent the (i,j)-th patch, and $\mathrm{AR}(\cdot)$ and $\mathrm{NLS}(\cdot)$ are two regularization terms corresponding to local auto-regression and non-local similarity. The local dictionaries are often incomplete (i.e., fewer columns than rows). Similar to non-overlapping patches (see next paragraph), non-completeness of local dictionaries and AR and NLS terms can

Figure 1. Image denosing comparison of two methods: (left image) solving (6) with all patches used at once, (right image) solving (6) with one subset of non-overlapping, covering patches. In (6), $\nu = 0.05$, and $\mathbf{D}$ was learned according to section 4.2.



PSNR = 26.98
All patches used at once

PSNR = 30.57
One subset of non-overlapping covering patches

reduce variable freedom and increase recoverability of (7). However, the use of more dictionaries and complicated regularization terms makes (7) more difficult to solve than our models.

Choice of P. One question is how to choose P, the subset of covering patches, such that (5) or (6) work well for recovering $\mathbf{M}$. We let P be a subset of non-overlapping, covering patches and focus on the unconstrained model (6). Figure 1 compares the two approaches. In this test, we set $\mathcal{A} = \mathcal{I}$ and $\mathbf{b} = \mathbf{M} + 0.05\boldsymbol{\xi}$ with $\boldsymbol{\xi} \sim \mathcal{N}(0, \mathbf{I})$, and we compared (6) with two different P's. In Figure 1, the left image uses all overlapping patches, and the right image uses one subset of non-overlapping, covering patches. We see that (6) with all patches produces much worse result than that with non-overlapping P.

We want to emphasize here that our results do not counter the intuition that using more patches should give better recovery. The results in Table 2 of section 4 demonstrate that using more different subsets of non-overlapping, covering patches can consistently improve the recovered image quality. The phenomenon in Figure 1 can be explained as follows. Using all the overlapping patches in (5) or (6) introduces too many unknowns to decide, a few times more than the number of the pixels. The ℓ_1 minimization typically needs $O(s\log(n/s))$ or more measurements to recover an s-sparse signal of length n. Suppose that the $\mathbf{y}_{ij}$ corresponding to each patch has at least r nonzeros and all the $(N_1 - n_1 + 1)(N_2 - n_2 + 1)$ overlapping patches are used. Then vector $\mathbf{y}$ has $n = K(N_1 - n_1 + 1)(N_2 - n_2 + 1)$ entries out of which at least $s = r(N_1 - n_1 + 1)(N_2 - n_2 + 1)$ are nonzeros. On the other hand, we have at most N_1N_2 measurements, not sufficiently many to reach $O(s\log(n/s)) = O(rN_1N_2\log(K/r))$.

An alternative explanation of the bad performance of (5) or (6) using all patches can be made through the model proposed in [18], which uses a variable $\mathbf{X}$ to represent the whole image and enforces[1] $\mathcal{R}_{ij}(\mathbf{X}) = \mathbf{D}\mathbf{y}_{ij}$ for every (i,j)-th patch. In (5) and (6), we can also use such a variable and enforce $\mathbf{X} = \mathcal{T}_P^{-1}\big(\sum_{(i,j)\in P}\mathcal{R}_{ij}^\top(\mathbf{D}\mathbf{y}_{ij})\big)$.

[1] Strictly speaking, $\mathcal{R}_{ij}(\mathbf{X}) \approx \mathbf{D}\mathbf{y}_{ij}$, $\forall(i,j)$ is achieved in [18].

However, note that

$$\big\{(\mathbf{X},\mathbf{y}):\, \mathcal{R}_{ij}(\mathbf{X}) = \mathbf{D}\mathbf{y}_{ij},\, \forall (i,j)\in P\big\} \subset \big\{(\mathbf{X},\mathbf{y}):\, \mathbf{X} = \mathcal{T}_P^{-1}\big(\sum_{(i,j)\in P} \mathcal{R}_{ij}^\top(\mathbf{D}\mathbf{y}_{ij})\big)\big\},$$

and the inclusion is strict if P is a set of overlapping covering patches. Hence, the overfitting problem of (5) or (6) using all patches is essentially because of insufficiently many constraints. *Therefore, unless more constraints or regularizations on* $\mathbf{y}$ *are introduced to help (see* (7) *or* [18, (4)] *for instance), we cannot use all the patches in* (5) *or* (6).

The method in [18] alternatingly updates $\mathbf{X}$ and $\mathbf{y}_{ij}$'s, and the $\mathbf{X}$-update owns an efficient closed form solution when $\mathcal{A}\mathcal{A}^\top = \mathcal{I}$ such as partial Fourier operator assumed in [18]. However, for general $\mathcal{A}$, it becomes difficult to perform its $\mathbf{X}$-update. In addition, updating the sparsity coefficients of all patches is much more expensive than that for just one set of nonoverlapping covering patches.

Since the image may not be evenly divided and the selected patches need to cover all the pixels of the image, we allow them to have different sizes. Slightly abusing the notation, we still use P to denote the set of selected patches, but P can also contain some smaller patches near the boundary. Although we can partition the image arbitrarily with blocks no greater than $n_1 \times n_2$, for simplicity we make the following assumptions.

Assumption 1 (Image partition by patches)**.**

- Interior patches (e.g., patch “A” in Figure 2) have size $n_1 \times n_2$;
- Left and right boundary patches have n_1 rows, and lower and upper boundary patches have n_2 columns; patch “B” in Figure 2 is an example;
- Corner patches (e.g., patch “C” in Figure 2) can have fewer than n_1 rows and n_2 columns;
- All patches are vertically and horizontally aligned.

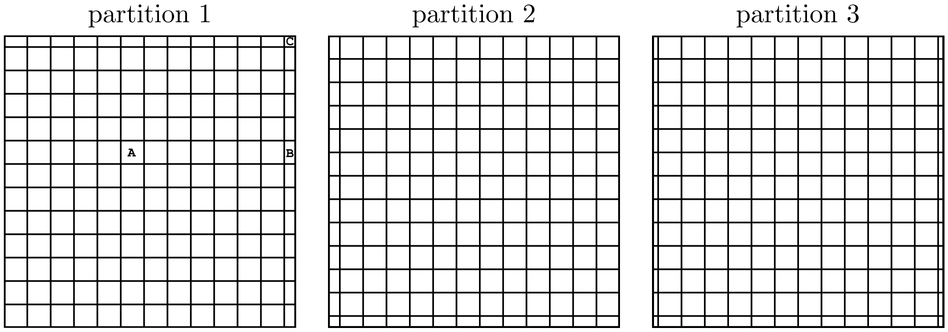
Remark 2. Under Assumption 1, the way an image is partitioned into patches is uniquely determined by the size of the upper-left corner patch.

Figure 2 illustrates how we partition a 100 × 100 image into non-overlapping patches in three different ways. Every patch is no greater than 8 × 8, and all interior patches are 8 × 8. However, since the image cannot be evenly partitioned, the patches near the boundary of the image may be smaller than 8 × 8. For example, in partition 1, all the right boundary patches are 8 × 4, and the upper-right corner patch is 4 × 4; in partition 3, all the left and right boundary patches are 8 × 2, and the lower-left and lower-right corner patches are 4 × 2.

Definition of operators. As P consists of non-overlapping covering patches, then every pixel must be contained by exactly one patch, and it is not difficult to verify $\mathcal{T}_P = \mathcal{I}$. If $(i,j)\in P$ is one interior patch, then $\mathcal{R}_{ij}(\mathbf{M})$ means to take the (i,j)-th patch of $\mathbf{M}$, and $\mathcal{R}_{ij}^\top(\mathbf{x})$ is to first generate an $N_1 \times N_2$ zero matrix, and then add $\mathbf{x}$ to its (i,j)-th patch. However, as $(i,j)\in P$ is a boundary or corner patch and its size is smaller than $n_1 \times n_2$, the corresponding operators need to act accordingly. For example, let (i,j) be patch “C” in Figure 2. Then we define

- $\mathcal{R}_{ij}(\mathbf{M})$: first generate an 8 × 8 zero matrix, and then replace its *upper-right* 4 × 4 corner submatrix with the *upper-right* 4 × 4 corner patch of $\mathbf{M}$;

FIGURE 2. Three different ways of partitioning a 100 × 100 image into non-overlapping patches, where each patch is no greater than 8 × 8 and all interior patches have size 8 × 8.



- $\mathcal{R}_{ij}^\top(\mathbf{x})$: first generate a 100 × 100 zero matrix, and then replace the *upper-right* 4×4 corner patch corresponding to "C" with the *upper-right* 4×4 corner submatrix of $\mathbf{x}$.

Averaging scheme. As shown in [7], tiling non-overlapping patches to perform image denoising would yield visible artifacts on block boundaries, and it was also observed when we solved (6) once with non-overlapping patches. Although using all patches in (6) at once does not give good recovery, we still want to use them in some way. Note that we have the freedom to choose P in (6), so we can solve it for different P's. For example, if $\mathbf{M} \in \mathbb{R}^{100\times 100}$ and dictionary atoms are 8×8, we can partition the image into non-overlapping patches in the three different ways in Figure 2, and solve (6) for each partition. It turns out that averaging the recovered images from different P's can remove the artifacts occuring on block boundaries and improve PSNR value; see the numerical results in section 4. Algorithm 1 summarizes our method. Note that (6) can be solved for different P's in parallel.

Note that in Algorithm 1, the dictionary $\mathbf{D}$ is fixed, and with a "good" $\mathbf{D}$, the algorithm can often produce a nice estimated image $\tilde{\mathbf{M}}$ averaging over a small number of recovered images (e.g., $t = 3$). Hence, running Algorithm 1 just once can already achieve the goal of reconstructing a whole image from its linear measurements. However, as discussed in the next section, one can adaptively update $\mathbf{D}$ and run Algorithm 1 repeatedly to yield better reconstruction.

Algorithm 1:

```
Data: Dictionary $\mathbf{D}$, patch size $(n_1, n_2)$, image size $(N_1, N_2)$, measurements $\mathbf{b}$,
      linear operator $\mathcal{A}$, and parameter $\nu$.
Choose $t$ different ways to partition the image into non-overlapping patches; denote
them as $P_1, \ldots, P_t$.
Solve (6) for $P_k$ and let the recovered image be $\mathbf{M}_k$, for $k = 1, \ldots, t$.
Average all the recovered images by $\tilde{\mathbf{M}} = \frac{1}{t}\sum_{k=1}^t \mathbf{M}_k$ and output $\tilde{\mathbf{M}}$.
```

2.2. **Adaptive dictionary update.** After obtaining an estimated image $\tilde{\mathbf{M}}$ by Algorithm 1, we can update the dictionary $\mathbf{D}$ using patches extracted from $\tilde{\mathbf{M}}$. Since $\tilde{\mathbf{M}}$ is close to the original image $\mathbf{M}$, the updated dictionary $\mathbf{D}$ from $\tilde{\mathbf{M}}$ should better represent the patches of $\mathbf{M}$. Hence, it is possible to further improve the result using the adaptively updated dictionary, and this process can be repeated several times. Algorithm 2 summarizes our adaptive method.

We observe that only the first adaptive update gives significant improvement, and subsequent ones make only minor changes to the dictionary and thus little improvement to the recovered image. For this reason, in the numerical experiments, we will update the dictionary only once.

Algorithm 2:

```
Data: Dictionary $\mathbf{D}$, patch size $(n_1, n_2)$, image size $(N_1, N_2)$, measurements $\mathbf{b}$,
      linear operator $\mathcal{A}$, and parameter $\nu$.
repeat
    Run Algorithm 1 and let the recovered image be $\tilde{\mathbf{M}}$.
    Update dictionary $\mathbf{D}$ from patches extracted from $\tilde{\mathbf{M}}$.
until convergence
```

3. **Block proximal gradient method for dictionary learning.** Both Algorithms 1 and 2 require an initial dictionary $\mathbf{D}$, which can be an analytic dictionary such as orthogonal or overcomplete wavelets, curvelets or DCT, or a learned one. For our purpose, a learned dictionary is preferable since it can be more adaptive to natural images [10, 7, 17]. To learn a dictionary, one can apply any available solver such as MOD, KSVD and OLM. We choose to use a new dictionary learning method, which applies the BPG method proposed in [22] to (3). Compared to some state-of-the-art methods, the new algorithm is often faster and produces more faithful dictionaries. Although (3) is non-convex jointly with respect to $\mathbf{D}$ and $\mathbf{Y}$, it is convex with respect to each of them while the other one is fixed. With this bi-convexity property, the BPG method is shown to generate a sequence globally converging to a stationary point of (3).

3.1. **Block proximal gradient method.** Recently, [22] characterized a class of *multi-convex* problems and proposed a BPG method for solving these problems. For simplicity and our purpose, we review the method only for *bi-convex* problems like (3). Consider

$$\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x},\mathbf{y}) + r_x(\mathbf{x}) + r_y(\mathbf{y}), \tag{8}$$

where f is differentiable and convex with respect to either $\mathbf{x}$ or $\mathbf{y}$ by fixing the other one, and r_x, r_y are extended-valued convex functions. At the k-th iteration of BPG, $\mathbf{x}$ and $\mathbf{y}$ are updated alternatively by

$$\mathbf{x}^k = \operatorname*{argmin}_{\mathbf{x}} \langle \nabla_{\mathbf{x}} f(\hat{\mathbf{x}}^k, \mathbf{y}^{k-1}), \mathbf{x} - \hat{\mathbf{x}}^k \rangle + \frac{L_x^k}{2} \|\mathbf{x} - \hat{\mathbf{x}}^k\|_2^2 + r_x(\mathbf{x}), \tag{9a}$$

$$\mathbf{y}^k = \operatorname*{argmin}_{\mathbf{y}} \langle \nabla_{\mathbf{y}} f(\mathbf{x}^k, \hat{\mathbf{y}}^k), \mathbf{y} - \hat{\mathbf{y}}^k \rangle + \frac{L_y^k}{2} \|\mathbf{y} - \hat{\mathbf{y}}^k\|_2^2 + r_y(\mathbf{y}), \tag{9b}$$

where L_x^k is a Lipschitz constant of $\nabla_x f(\mathbf{x}, \mathbf{y}^{k-1})$ with respect to $\mathbf{x}$, $\hat{\mathbf{x}}^k = \mathbf{x}^{k-1} + \omega_x^k(\mathbf{x}^{k-1} - \mathbf{x}^{k-2})$ denotes an extrapolated point with weight $\omega_x^k \geq 0$, and L_y^k and $\hat{\mathbf{y}}^k$ have the same meanings for $\mathbf{y}$.

BPG is a variant of the block coordinate minimization (BCM) method (see [21] and the references therein), which updates $\mathbf{x}, \mathbf{y}$ cyclically by minimizing the objective with respect to one block of variables at a time while the other is fixed at its most recent value. Though BCM decreases the objective faster, subproblems for BCM are usually much more difficult than those in (9). For simple r_x and r_y, the updates in (9) have closed form solutions.

Under some boundedness assumptions, [22] establishes subsequence convergence of the BPG method. Further assuming the so-called Kurdyka-Łojasiewicz (KL) property (see [12, 5] for example), it shows that the sequence $\{(\mathbf{x}^k, \mathbf{y}^k)\}$ generated by (9) globally converges to a stationary point of (8) .

3.2. **Dictionary learning.** We learn a dictionary from training dataset $\mathbf{X}$ via solving (3). Let

$$\ell(\mathbf{D}, \mathbf{Y}) = \frac{1}{2}\|\mathbf{D}\mathbf{Y} - \mathbf{X}\|_F^2$$

be the fidelity term in (3). Applying (9) to (3), we alternatively update $\mathbf{D}$ and $\mathbf{Y}$ by

$$\mathbf{D}^k = \operatorname*{argmin}_{\mathbf{D}\in\mathcal{D}} \langle \nabla_{\mathbf{D}}\ell(\hat{\mathbf{D}}^k, \mathbf{Y}^{k-1}), \mathbf{D} - \hat{\mathbf{D}}^k\rangle + \frac{L_d^k}{2}\|\mathbf{D} - \hat{\mathbf{D}}^k\|_F^2, \tag{10a}$$

$$\mathbf{Y}^k = \operatorname*{argmin}_{\mathbf{Y}} \langle \nabla_{\mathbf{Y}}\ell(\mathbf{D}^k, \hat{\mathbf{Y}}^k), \mathbf{Y} - \hat{\mathbf{Y}}^k\rangle + \frac{L_y^k}{2}\|\mathbf{Y} - \hat{\mathbf{Y}}^k\|_F^2 + \lambda\|\mathbf{Y}\|_1, \tag{10b}$$

where

$$\mathcal{D} = \{\mathbf{D} : \|\mathbf{d}_i\|_2 \le 1, i = 1, \dots, K\}$$

is the constraint set of $\mathbf{D}$, $\hat{\mathbf{D}}^k = \mathbf{D}^{k-1} + \omega_d^k(\mathbf{D}^{k-1} - \mathbf{D}^{k-2})$ and $\hat{\mathbf{Y}}^k = \mathbf{Y}^{k-1} + \omega_y^k(\mathbf{Y}^{k-1} - \mathbf{Y}^{k-2})$ denote extrapolated points with $\omega_d^k, \omega_y^k \le 1$, and L_d^k and L_y^k are taken as Lipschitz constants of $\nabla_{\mathbf{D}}\ell(\mathbf{D}, \mathbf{Y}^{k-1})$ and $\nabla_{\mathbf{Y}}\ell(\mathbf{D}^k, \mathbf{Y})$ about $\mathbf{D}$ and $\mathbf{Y}$ respectively. Note that the extrpolated point $\hat{\mathbf{D}}^k$ may be not in the feasible set $\mathcal{D}$, but $\mathbf{D}^k$ is always kept in $\mathcal{D}$.

The updates in (10) can be explicitly written as

$$\mathbf{D}^k = \mathcal{P}_{\mathcal{D}}\left(\hat{\mathbf{D}}^k - \frac{1}{L_d^k}\nabla_{\mathbf{D}}\ell(\hat{\mathbf{D}}^k, \mathbf{Y}^{k-1})\right), \tag{11a}$$

$$\mathbf{Y}^k = \mathcal{S}_{\lambda/L_y^k}\left(\hat{\mathbf{Y}}^k - \frac{1}{L_y^k}\nabla_{\mathbf{Y}}\ell(\mathbf{D}^k, \hat{\mathbf{Y}}^k)\right), \tag{11b}$$

where in (11a), $\mathcal{P}_{\mathcal{D}}(\cdot)$ denotes the Euclidean projection to $\mathcal{D}$ defined for any $\mathbf{D}$ as

$$\big(\mathcal{P}_{\mathcal{D}}(\mathbf{D})\big)_i = \frac{\mathbf{d}_i}{\max(1, \|\mathbf{d}_i\|_2)},\ i = 1, \dots, K,$$

and in (11b), $\mathcal{S}_\tau(\cdot)$ denotes soft-thresholding operator defined for any $\mathbf{Y}$ by

$$\big(\mathcal{S}_\tau(\mathbf{Y})\big)_{ij} = \operatorname{sign}(y_{ij}) \cdot \max(|y_{ij}| - \tau, 0),\ \forall\ i, j.$$

Note that $\nabla_{\mathbf{D}}\ell(\mathbf{D}, \mathbf{Y}) = (\mathbf{D}\mathbf{Y} - \mathbf{X})\mathbf{Y}^\top$ and

$$\|\nabla_{\mathbf{D}}\ell(\mathbf{D}, \mathbf{Y}) - \nabla_{\mathbf{D}}\ell(\tilde{\mathbf{D}}, \mathbf{Y})\|_F = \|(\mathbf{D} - \tilde{\mathbf{D}})\mathbf{Y}\mathbf{Y}^\top\|_F \le \|\mathbf{Y}\mathbf{Y}^\top\|\|\mathbf{D} - \tilde{\mathbf{D}}\|_F,\ \forall\ \mathbf{D}, \tilde{\mathbf{D}},$$

where $\|\mathbf{A}\|$ denotes matrix operator norm of $\mathbf{A}$. Hence, $\|\mathbf{Y}\mathbf{Y}^\top\|$ is a Lipschitz constant of $\nabla_{\mathbf{D}}\ell(\mathbf{D}, \mathbf{Y})$ about $\mathbf{D}$. Similarly, $\|\mathbf{D}^\top\mathbf{D}\|$ is a Lipschitz constant of $\nabla_{\mathbf{Y}}\ell(\mathbf{D}, \mathbf{Y})$ about $\mathbf{Y}$. Throughout our numerical tests, we take

$$L_d^k = \|\mathbf{Y}^{k-1}(\mathbf{Y}^{k-1})^\top\|, \qquad L_y^k = \|(\mathbf{D}^k)^\top\mathbf{D}^k\|. \tag{12}$$

Although $\|\mathbf{Y}\mathbf{Y}^\top\|_F$ and $\|\mathbf{D}^\top\mathbf{D}\|_F$ are also gradient Lipschitz constants and cheaper than the spectral norms to calculate, we notice that the Frobenius norms are often too large and make the algorithm converge slow. In addition, the cost of computing L_d^k and L_y^k as in (12) is much lower than that of getting $\nabla_{\mathbf{D}}\ell(\hat{\mathbf{D}}^k, \mathbf{Y}^{k-1})$ and $\nabla_{\mathbf{Y}}\ell(\mathbf{D}^k, \hat{\mathbf{Y}}^k)$, and thus it will not decrease the per-iteration complexity much even if we use cheaper Lipschitz constants like $\|\mathbf{Y}\mathbf{Y}^\top\|_F$ and $\|\mathbf{D}^\top\mathbf{D}\|_F$.

The extrapolation weights are taken as[2]

$$\omega_d^k = 0.9999 \min\left(\omega^k, \sqrt{\frac{L_d^{k-1}}{L_d^k}}\right), \qquad \omega_y^k = 0.9999 \min\left(\omega^k, \sqrt{\frac{L_y^{k-1}}{L_y^k}}\right), \tag{13}$$

where $\omega^k = \frac{t_{k-1}-1}{t_k}$ with $t_0 = 1$ and $t_k = \frac{1}{2}\left(1 + \sqrt{1 + 4t_{k-1}^2}\right)$. The weight ω^k has been used in FISTA [3], showing that this kind of extrapolation significantly accelerates the proximal gradient method for convex composite problems. We observe that the extrapolation with weights in (13) can also greatly speed up the BPG method for solving (3).

The setting of ω_d^k and ω_y^k in (13) follows that of [22]. Intuitively, we want them to be close to ω^k to have fast convergence. However, since (3) is non-convex, simply setting them to ω^k does not guarantee convergence of the iterates. Hence, we perform "min" operation in (13) as a safeguard to control the extrapolation and avoid iterate divergence. Letting $\omega_d^k \le \tau\sqrt{\frac{L_d^{k-1}}{L_d^k}}$ and $\omega_y^k \le \tau\sqrt{\frac{L_y^{k-1}}{L_y^k}}$ for some $\tau < 1$ (e.g., $\tau = 0.9999$) yields square summable results of the iterates, which is a key step to have global convergence in Theorem 3.1 below.

To make the whole objective non-increasing, we redo the k-th iteration by setting $\omega_d^k = \omega_y^k = 0$ (i.e., no extrapolation) if $F(\mathbf{D}^k, \mathbf{Y}^k) > F(\mathbf{D}^{k-1}, \mathbf{Y}^{k-1})$, where

$$F(\mathbf{D}, \mathbf{Y}) = \frac{1}{2}\|\mathbf{D}\mathbf{Y} - \mathbf{X}\|_F^2 + \lambda\|\mathbf{Y}\|_1$$

is the objective of (3). As shown in [22], the setting of $\omega_d^k = \omega_y^k = 0$ guarantees $F(\mathbf{D}^k, \mathbf{Y}^k)$ no greater than $F(\mathbf{D}^{k-1}, \mathbf{Y}^{k-1})$. The non-increasing property is not only required by global convergence, but also important to make the algorithm perform stably and converge rapidly. The pseudocode of our method is shown in Algorithm 3.

Remark 3. Our algorithm uses proximal update for both $\mathbf{D}$ and $\mathbf{Y}$. It differs from other methods such as KSVD and OLM which perform exact minimization to update $\mathbf{D}$ and/or $\mathbf{Y}$. Maintaining closed form solutions for both $\mathbf{D}$ and $\mathbf{Y}$-subproblems ensures the algorithm to have a lower per-iteration complexity, and the extrapolation technique lets it take a small number of iterations to achieve a faithful solution.

3.3. **Convergence results.** Note that (3) is equivalent to

$$\min_{\mathbf{D},\mathbf{Y}} \frac{1}{2}\|\mathbf{D}\mathbf{Y} - \mathbf{X}\|_F^2 + \lambda\|\mathbf{Y}\|_1 + \delta_{\mathcal{D}}(\mathbf{D}), \tag{14}$$

[2] In (13), the number "0.9999" can be replaced by any positive number less than *one* to guarantee the global convergence of the algorithm. Numerically, the number that is closer to *one* makes the algorithm converge faster. We observed that even if it was equal *one*, the algorithm still converged.

Algorithm 3: Block proximal gradient for dictionary learning

```
Data: training samples $\mathbf{X}$, parameter $\lambda > 0$, and initial points
      $(\mathbf{D}^{-1}, \mathbf{Y}^{-1}) = (\mathbf{D}^0, \mathbf{Y}^0)$
for $k = 1, 2, \cdots$ do
    Set $L_d^k$ and $\omega_d^k$ by (12) and (13), respectively.
    Let $\hat{\mathbf{D}}^k = \mathbf{D}^{k-1} + \omega_d^k(\mathbf{D}^{k-1} - \mathbf{D}^{k-2})$ and get $\mathbf{D}^k$ by (11a).
    Set $L_y^k$ and $\omega_y^k$ by (12) and (13), respectively.
    Let $\hat{\mathbf{Y}}^k = \mathbf{Y}^{k-1} + \omega_y^k(\mathbf{Y}^{k-1} - \mathbf{Y}^{k-2})$ and get $\mathbf{Y}^k$ by (11b).
    if $F(\mathbf{D}^k, \mathbf{Y}^k) > F(\mathbf{D}^{k-1}, \mathbf{Y}^{k-1})$ then
ReDo    Re-update $\mathbf{D}^k$ and $\mathbf{Y}^k$ by (11a) and (11b) with $\hat{\mathbf{D}}^k = \mathbf{D}^{k-1}$ and
        $\hat{\mathbf{Y}}^k = \mathbf{Y}^{k-1}$, respectively.
    if Some stopping conditions are satisfied then
        Output $(\mathbf{D}^k, \mathbf{Y}^k)$ and stop.
```

where $\delta_{\mathcal{D}}(\cdot)$ is the indicator function on $\mathcal{D}$. According to [22], the objective of (14) is semi-algebraic [4] and has the KL property. In addition, the sequence $\{\mathbf{D}^k\}$ is in the bounded set $\mathcal{D}$, and positive λ makes $\{\mathbf{Y}^k\}$ bounded because otherwise the objective of (14) will blow up. Hence, $\{(\mathbf{D}^k, \mathbf{Y}^k)\}$ has a finite limit point, and the Lipschitz constants specified in (12) must be upper bounded. On the other hand, as long as $\{\mathbf{D}^k\}$ and $\{\mathbf{Y}^k\}$ are uniformly away from origin, L_d^k and L_y^k are uniformly above *zero*. Therefore, according to Theorem 2.8 of [22], we immediately have the following theorem.

Theorem 3.1. *Let $\{(\mathbf{D}^k, \mathbf{Y}^k)\}$ be the sequence generated by Algorithm 3. If both $\{\mathbf{D}^k\}$ and $\{\mathbf{Y}^k\}$ are uniformly away from origin, then $(\mathbf{D}^k, \mathbf{Y}^k)$ converges to a stationary point of (14) or equivalently (3).*

4. Numerical results.

In this section, we first test Algorithm 3 for dictionary learning and compare it with KSVD [1] and OLM [14] on synthetic data. Then we do a set of image recovery tests to show the effectiveness of model (6) and the adaptive method discussed in section 2.2.

4.1. Synthetic test for dictionary recovery.

This test compares Algorithm 3 with methods KSVD and OLM for dictionary learning. We chose KSVD and OLM because they appear to be most popular in the literature and their codes are both available online. In addition, they have been demonstrated efficient for many image processing tasks. There are other dictionary learning algorithms such as MOD [8] and recursive least squares [19]. However, we do not intend to exhaust all of them.

Following [1], we generated the test data as follows. We first generated a dictionary $\mathbf{D} \in \mathbb{R}^{n \times K}$ with Matlab command `randn(n,K)` and normalized each column of $\mathbf{D}$ to have unit ℓ_2-norm. Then we generated p training samples in the n-dimensional space. Each sample is a linear combination of uniformly randomly selected r columns of $\mathbf{D}$, and the coefficients were Gaussian randomly generated. On the same data, we ran KSVD for (2), and both Algorithm 3 and OLM for (3). In (2) we set $s = r$, i.e., the true sparsity level was assumed, and in (3) we set $\lambda = 0.5/\sqrt{n}$. Algorithm 3 was terminated as long as

$$\frac{|F(\mathbf{D}^k, \mathbf{Y}^k) - F(\mathbf{D}^{k+1}, \mathbf{Y}^{k+1})|}{1 + F(\mathbf{D}^k, \mathbf{Y}^k)} \le 10^{-5}$$

Table 1. Average running time (sec), iteration numbers and recovery rates (%) of 50 independent runs by Algorithm 3, KSVD, and online learning method (OLM).

	Algorithm 3			OLM	KSVD		Algorithm 3			OLM	KSVD		Algorithm 3			OLM	KSVD	
r	time	iter	rate	rate	time	rate	time	iter	rate	rate	time	rate	time	iter	rate	rate	time	rate
	$(K,p)=(2n,20n)$						$(K,p)=(2n,100n)$						$(K,p)=(4n,100n)$					
4	1.42	169	98.97	98.83	14.1	97.11	3.57	165	99.56	99.81	54.4	99.44	9.63	240	99.21	99.46	62.7	98.96
6	1.84	218	99.08	98.28	18.9	97.11	4.44	195	99.39	99.94	78.4	99.28	12.4	305	99.25	99.64	88.9	98.19
8	2.44	286	98.61	92.08	23.9	6.25	5.78	251	99.56	98.28	103	99.50	17.3	430	99.21	99.58	116	98.21
10	3.18	404	96.97	63.50	29.1	0.00	6.94	314	99.39	94.11	129	97.25	24.1	582	98.63	98.47	145	0.00
12	5.44	645	74.25	33.94	34.4	0.00	8.03	389	99.47	56.81	157	0.17	34.4	851	95.82	80.57	174	0.00

was satisfied in three consecutive iterations or it ran over 1000 iterations. KSVD was run to 200 iterations, and OLM ran to the same time as that of Algorithm 3. All other parameters for KSVD and OML were set to their default values.

We fixed $n = 36$ and tested three different pairs of (K, p). For each pair of (K, p), sparsity level r varied among $\{4, 6, 8, 10, 12\}$. The recovery of each atom $\mathbf{d}$ of the original dictionary $\mathbf{D}$ was regarded successful if

$$\max_{1\le i\le K} \frac{|\mathbf{d}^\top \tilde{\mathbf{d}}_i|}{\|\mathbf{d}\|_2 \|\tilde{\mathbf{d}}_i\|_2} \ge 0.99,$$

where $\tilde{\mathbf{d}}_i$ is the i-th column of an estimated dictionary $\tilde{\mathbf{D}}$. The average running time, iteration numbers and recovery rates of 50 independent runs are shown in Table 1. From the table, we see that our method used much less time than KSVD with comparable recovery rates. When sparsity level r is big (e.g., $r = 12$) or the training samples are not so many (e.g., $p = 20n$), our method got much higher recovery rates than those by KSVD. For big r's (e.g., $r = 10, 12$) or small p's (e.g., $p = 20n$), OLM tends to give lower rates than our method. It could be because our method converges fast but OLM does not. For other cases, our method and OLM gave almost the same results. We want to mention that OLM solves the same model as that by our method, and in all cases OLM should be able to give results similar to ours if it is allowed to run a sufficiently long time.

4.2. **Whole image recovery.** This section tests the performance of Algorithms 1 and 2 on image recovery. Two different dictionaries were compared for Algorithm 1. One was an overcomplete DCT, generated in the same way as in [1]. Another one was learned from 20,000 8×8 grayscale patches, that were 100 randomly extracted patches from each of the 200 images in the training set of the Berkeley segmentation dataset [16]. For the learned dictionary, we first subtracted each training patch by its mean, and then trained a dictionary $\hat{\mathbf{D}}$ using these zero-mean patches via solving (3) with $K = 256$ by Algorithm 3, where we chose $\lambda = 0.8/\sqrt{n}$ to make the average nonzero number per column of $\mathbf{Y}$ about 8. Finally, we let $\mathbf{D} = [\mathbf{e}, \hat{\mathbf{D}}] \in \mathbb{R}^{64\times 257}$ and used $\mathbf{D}$ in our tests, where $\mathbf{e}$ is a vector with all *one*'s. Such an atom with constant components is called a DC in [1], which shows that the processed dictionary $\mathbf{D}$ performs better than $\hat{\mathbf{D}}$ for real-world image processing tasks. Here, we want to mention that for an image patch $\mathbf{x}$, if $\mathbf{x} - \text{mean}(\mathbf{x})$ has a sparse representation under $\hat{\mathbf{D}}$, i.e., $\mathbf{x} - \text{mean}(\mathbf{x}) = \hat{\mathbf{D}}\mathbf{y}$ with sparse $\mathbf{y}$, then $\mathbf{x} = \text{mean}(\mathbf{x})\mathbf{e} + \hat{\mathbf{D}}\mathbf{y}$, which means $\mathbf{x}$ is sparse under $\mathbf{D}$. Therefore, the above processing is reasonable. The used overcomplete DCT is also 64×257, and its first column is a DC. For Algorithm 2, we used the above $\mathbf{D}$ as its initial dictionary, and updated the dictionary only once by learning a new one via Algorithm 3 using patches[3] of the first-step estimated image, which is exactly the output of Algorithm 1 using $\mathbf{D}$. Then we used the updated dictionary to perform image recovery once more to get the final result.

Implementation. In (6), we took $\mathbf{b} = \mathcal{A}(\mathbf{M}) + \sigma\boldsymbol{\xi}$, where $\boldsymbol{\xi} \sim \mathcal{N}(0, \mathbf{I})$ is Gaussian noise, and $\sigma = \hat{\sigma}\|\mathcal{A}(\mathbf{M})\|_2/\|\boldsymbol{\xi}\|_2$ throughout our tests, where $\hat{\sigma}$ varied among $\{0.01, 0.05, 0.10\}$. We took $\nu = \sigma$ for the first two kinds of $\mathcal{A}$ and $\nu = 0.1\sigma$ for the third kind of $\mathcal{A}$. The definitions of different $\mathcal{A}$'s are given in the next paragraph. In addition, we set all elements of $\mathbf{w}_{ij}$ to *one* except its first component, which was set to *zero*. Under this setting, using any DC as the first atom of $\mathbf{D}$ would make no difference for the solution of (6). Then, (6) was solved via YALL1 (version 1.4) [24], for which we used Gaussian random starting point and 10^{-4} as its stopping tolerance. All other parameters of YALL1 were set to their default values. We chose YALL1 due to its high efficiency for solving (6) and easy call by providing operations of $\mathcal{A}$ and $\mathcal{A}^\top$.

Three different kinds of $\mathcal{A}$ were tested. The first one did image inpainting and used the sampling operator $\mathcal{P}_\Omega$, which takes all pixels of its argument in Ω and zeros out all others. The adjoint of $\mathcal{P}_\Omega$ is to fill in the locations in Ω by its argument and other locations by zero. The second one did compressed image recovery and took $\mathcal{A}$ as the composition of $\mathcal{P}_\Omega$ and two-dimensional complex-valued circulant operator $\mathcal{C}_2$, i.e., $\mathcal{A} = \mathcal{P}_\Omega \circ \mathcal{C}_2$. We did the same normalization to $\mathcal{A}$ as in [23], which uses such $\mathcal{A}$ for testing learned circulant sensing operators. Performing $\mathcal{C}_2$ on a matrix $\mathbf{M}$ can be realized by one fast Fourier transform (FFT), one inverse FFT and some component-wise multiplications, and the adjoint of $\mathcal{C}_2$ is to do one fast Fourier transform (FFT), one inverse FFT and some component-wise divisions. The third kind of $\mathcal{A}$ was a blurring operator with a 9 × 9 kernel. We used two different kernels, which were generated by Matlab's commands `fspecial('average',[9,9])` and `fspecial('motion',10,45)` respectively. The implementation of a blurring operator can also be realized by one FFT, one inverse FFT, and some component-wise products. Hence, all the three kinds of $\mathcal{A}$ can be easily realized in algorithms and in hardware.

Our method processes a subset of nonoverlapping, covering patches together, and thus it recovers the whole image at a time. For $\mathcal{A} = \mathcal{P}_\Omega$, one can denoise all overlapping patches independently since every measurement only involves one single pixel. However, this way would require noise information on each patch while our method only on the whole image. For the other two $\mathcal{A}$'s, every measurement mixes more than one or even all image pixels, and one would not be able to process different patches independently.

Results. First, let us see how the averaging scheme in Algorithm 1 improves the recovery performance. We tested it on the grayscale versions of Castle and Lena images shown in Figure 3, and both of the two images are unrelated to the training samples. We chose five different partitions, whose upper-left corner patches were 8 × 8, 8 × 4, 4 × 8, 8 × 2, and 2 × 8, respectively. (Recall that each partition is uniquely determined by its upper-left corner patch under Assumption 1.) For each partition, we solved (6) to obtain a recovered image. Let the recovered images be denoted by $\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \mathbf{M}_4, \mathbf{M}_5$. We compared PSNR values of the running average $\mathbf{M}_j^{av} = \frac{1}{j}\sum_{i=1}^{j}\mathbf{M}_i$ and the $\mathbf{M}_i$ that had the greatest PSNR among the

[3] Similarly, we subtracted every patch by its mean, and we augmented the learned dictionary by adding $\mathbf{e}$ as one more atom.

FIGURE 3. Four tested images. From left to right: Castle, Lena, Plane, Boat

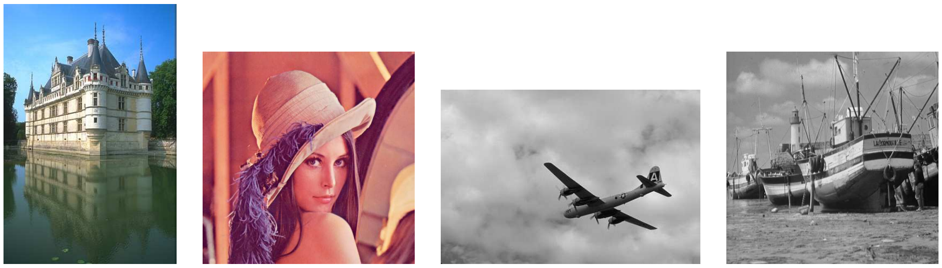


TABLE 2. PSNR values of averaged images for $j = 1, \dots, 5$. Every measurement vector contains 1% Gaussian noise. For both image inpainting ($\mathcal{A} = \mathcal{P}_\Omega$) and compressed imaging ($\mathcal{A} = \mathcal{P}_\Omega \circ \mathcal{C}_2$), 30% pixels were chosen uniformly at random.

Image	$\mathbf{M}_{\text{best}}$	$\mathbf{M}_1^{av}$	$\mathbf{M}_2^{av}$	$\mathbf{M}_3^{av}$	$\mathbf{M}_4^{av}$	$\mathbf{M}_5^{av}$	$\mathbf{M}_{\text{best}}$	$\mathbf{M}_1^{av}$	$\mathbf{M}_2^{av}$	$\mathbf{M}_3^{av}$	$\mathbf{M}_4^{av}$	$\mathbf{M}_5^{av}$
	image inpainting						compressed imaging					
Castle	25.23	25.05	25.80	26.21	26.36	26.48	34.13	34.04	34.86	35.27	35.44	35.57
Lena	29.96	29.91	31.01	31.49	31.71	31.81	38.84	38.84	39.53	39.81	39.95	40.03
	"average" blurring						"motion" blurring					
Castle	28.82	28.77	29.26	29.56	29.65	29.71	32.60	32.49	33.09	33.36	33.48	33.57
Lena	32.26	32.22	32.79	33.09	33.20	33.29	36.55	36.55	37.17	37.44	37.56	37.63

five, denoted $\mathbf{M}_{\text{best}}$. Table 2 lists the average results of five independent runs for four different $\mathcal{A}$'s and noise level $\hat{\sigma} = 1\%$. Figures 4 and 5 in the appendix show the recovered images and their residuals of the first run corresponding to the smallest and greatest PSNRs and also the averaged ones. For the first two $\mathcal{A}$'s, we took 30% uniformly random pixels, i.e., SR $:= \frac{|\Omega|}{N_1 N_2} = 30\%$. From the results, we see that the averaging scheme consistently improves the recovery performance. Note that there are at most $n_1 n_2$ different partitions under Assumption 1. We observed that the more different partitions we used, the better result could we get by the averaging scheme. However, the rate of improvement drops as the number of partitions increases, as shown in Table 2. For this reason, we only use three different partitions in the remaining experiments.

Next, we compare Algorithm 1 with two different dictionaries and Algorithm 2 on the four images shown in Figure 3. All of these images were unrelated to the learned dictionary $\mathbf{D}$. To show the effectiveness of (6), we also included a TV-based method for the first two $\mathcal{A}$'s and an overlapping patch-based method for the third kind of $\mathcal{A}$ in the comparison. The TV-based method solves

$$\min_{\mathbf{M}} \|\mathbf{M}\|_{\text{TV}} + \frac{\gamma}{2}\|\mathcal{A}(\mathbf{M}) - \mathbf{b}\|_2^2, \tag{15}$$

where $\|\cdot\|_{\text{TV}}$ denotes TV semi-norm, and the overlapping patch-based method solves (7). We employed TVAL3 (version beta2.4) [11] to solve (15), and its default settings were used. The model (7) was solved by the algorithm in [6], and its code was available online from the authors' webpage. We set its maximum number of

Table 3. PSNR values of recovered images for image inpainting ($\mathcal{A} = \mathcal{P}_\Omega$). From left to right, the results correspond to Algorithm 1 with learned dictionary, Algorithm 1 with DCT, Algorithm 2, and TV method, respectively. Bold is best.

Image	SR=30%				SR=50%			
noise level $\hat{\sigma} = 1\%$								
Castle	26.16	24.58	**26.37**	25.05	29.30	27.41	**29.51**	27.88
Lena	31.40	28.57	**31.70**	29.07	35.33	31.98	**35.44**	32.43
Plane	32.66	29.17	**33.46**	30.31	37.43	32.62	**38.56**	33.53
Boat	28.49	25.79	**29.14**	26.70	31.86	29.05	**32.48**	30.00
noise level $\hat{\sigma} = 5\%$								
Castle	26.09	24.57	**26.23**	24.99	29.22	27.30	**29.47**	27.68
Lena	31.02	28.50	**31.22**	28.91	34.49	31.64	**34.81**	31.96
Plane	32.29	29.18	**32.54**	30.11	36.84	32.58	**37.54**	33.02
Boat	28.31	25.82	**28.63**	26.62	31.67	28.88	**32.28**	29.70
noise level $\hat{\sigma} = 10\%$								
Castle	25.85	24.47	**25.98**	24.81	28.76	27.02	**29.00**	27.15
Lena	30.34	28.25	**30.53**	28.47	33.21	30.96	**33.41**	30.77
Plane	31.81	29.06	**32.03**	29.50	35.67	32.18	**36.27**	31.57
Boat	27.78	25.62	**28.07**	26.30	30.75	28.36	**31.30**	28.90

iterations to 10^4, which was sufficiently large to make the algorithm to solve (7) to a high accuracy. In their code, the second group of local dictionaries were used, and we tuned its sparsity parameter `par.tau` and `par.c1` used for adaptively updating the sparsity parameter while all the other parameters were set to their default values. For color images, each of RGB channels was recovered independently.

For $\mathcal{A} = \mathcal{P}_\Omega$, we tested SR $= 30\%,\ 50\%$, and for $\mathcal{A} = \mathcal{P}_\Omega \circ \mathcal{C}_2$, we tested SR $= 10\%,\ 20\%,\ 30\%$. For each tested image, we chose three different partitions, whose upper-left corner patches were 8×8, 8×4, and 4×8, respectively. The same three partitions were used in both Algorithms 1 and 2. Table 3 lists the average results of five independent trials by the compared methods for $\mathcal{A} = \mathcal{P}_\Omega$, Table 4 for $\mathcal{A} = \mathcal{P}_\Omega \circ \mathcal{C}_2$ and Table 5 for image deblurring. From the results, we see that Algorithm 1 works better with learned $\mathbf{D}$ than DCT except for the Castle image when $\mathcal{A}$ is `average` blurring operator and $\hat{\sigma} = 10\%$. Our method with learned $\mathbf{D}$ is consistently better for $\mathcal{A} = \mathcal{P}_\Omega$ and much better for $\mathcal{A} = \mathcal{P}_\Omega \circ \mathcal{C}_2$ than TV-based model (15). For both blurring operators, our method is better than that in [6] for solving (7) except when noise level $\hat{\sigma} = 10\%$, the latter performs better on the Boat image for `average` and the Castle image for both `average` and `motion`. In addition, Algorithm 2 with adaptively updated dictionary makes improvement over Algorithm 1 in all cases except for the Castle and Boat images when $\mathcal{A}$ is `average` blurring operator and $\hat{\sigma} = 10\%$. The improvement usually increases as SR increases. It is reasonable since higher SRs give cleaner images, which further generate better dictionaries.

We provide open source codes on our websites and welcome the interested reader to try it on more datasets.

5. Conclusions.

Dictionary learning has been popularly applied to image denoising, super-resolution, classification and feature extraction. Various algorithms have been proposed for learning dictionaries to achieve different goals. In this paper,

Table 4. PSNR values of recovered images for compressed imaging ($\mathcal{A} = \mathcal{P}_\Omega \circ \mathcal{C}_2$). From left to right, the results correspond to Algorithm 1 with learned dictionary, Algorithm 1 with DCT, Algorithm 2, and TV method, respectively. Bold is best.

Image	SR=10%				SR=20%				SR=30%			
noise level $\hat{\sigma} = 1\%$												
Castle	27.91	26.00	**28.27**	23.35	32.00	30.73	**33.35**	25.10	35.22	35.46	**37.76**	27.27
Lena	32.19	29.53	**32.36**	25.75	36.41	33.86	**36.76**	27.93	39.48	37.64	**40.06**	28.79
Plane	39.56	36.11	**40.72**	30.45	42.76	40.88	**44.15**	31.52	45.60	43.51	**46.37**	32.70
Boat	28.80	26.08	**28.93**	24.67	32.48	30.00	**32.98**	27.21	34.64	33.58	**35.66**	27.24
noise level $\hat{\sigma} = 5\%$												
Castle	27.59	25.78	**27.87**	23.20	31.19	29.77	**31.97**	24.74	33.38	32.77	**34.58**	26.71
Lena	31.22	29.07	**31.38**	25.70	34.49	32.49	**34.73**	27.87	36.53	34.94	**36.83**	28.78
Plane	36.92	32.59	**37.52**	29.92	39.28	37.08	**40.01**	31.76	40.45	38.87	**41.17**	31.83
Boat	28.21	25.86	**28.32**	24.65	31.50	29.12	**31.79**	25.95	33.41	31.65	**33.85**	28.08
noise level $\hat{\sigma} = 10\%$												
Castle	26.79	25.23	**26.97**	22.82	30.00	28.66	**30.50**	25.03	31.72	30.86	**32.44**	25.31
Lena	29.92	28.03	**30.03**	25.88	32.75	31.04	**32.92**	27.51	34.24	32.76	**34.42**	28.65
Plane	34.33	30.14	**34.73**	28.37	36.69	33.84	**37.15**	30.24	37.41	35.70	**37.84**	31.19
Boat	27.26	25.35	**27.33**	24.76	30.13	28.06	**30.30**	27.01	31.90	30 12	**32.16**	26.89

Table 5. PSNR values of recovered images for image deblurring. From left to right, the results correspond to blurred image, Algorithm 1 with learned dictionary, Algorithm 1 with DCT, Algorithm 2, and the overlapping patch-based method in [6] for solving (7), respectively.

Image	"average" blurring					"motion" blurring				
noise level $\hat{\sigma} = 1\%$										
Castle	22.37	29.50	28.62	**29.56**	28.69	23.24	33.28	33.06	**34.26**	31.71
Lena	26.00	33.01	32.05	**33.04**	32.38	27.88	37.34	36.43	**37.58**	35.14
Plane	27.88	37.27	34.60	**37.62**	33.74	28.66	40.98	39.63	**41.55**	35.35
Boat	23.36	31.45	30.14	**31.54**	30.70	24.64	34.79	34.11	**35.31**	33.92
noise level $\hat{\sigma} = 5\%$										
Castle	22.03	26.39	25.99	**26.47**	25.42	22.91	27.39	26.43	**27.62**	26.85
Lena	25.85	29.41	29.02	**29.60**	28.86	27.63	30.37	29.10	**31.16**	30.89
Plane	27.64	31.84	30.89	**32.00**	29.91	28.36	34.18	31.78	**34.49**	31.74
Boat	23.27	27.86	27.17	**27.95**	27.18	24.51	28.93	27.62	**29.17**	28.67
noise level $\hat{\sigma} = 10\%$										
Castle	21.77	24.10	24.14	23.92	**24.93**	22.60	24.95	24.29	25.02	**25.30**
Lena	25.44	27.49	27.02	**27.55**	27.25	26.95	29.61	28.64	**29.65**	28.33
Plane	26.95	30.24	29.03	**30.39**	28.13	27.54	32.50	30.57	**32.83**	29.01
Boat	22.99	25.34	25.20	25.13	**25.81**	24.13	26.66	25.62	**26.75**	26.70

we focus on whole-image recovery and develop novel methods for learning dictionaries and then recovering images quickly and faithfully. Our algorithm not only has low per-iteration complexity and also converges fast. In the algorithm, using non-overlapping patches and averaging across different subsets of patches greatly reduce the variable freedom and are critical for fast and successful recovery.

FIGURE 4. Recovered Castle images and their residuals corresponding to the smallest (the first two columns) and greatest (the 3rd and 4th columns) PSNRs and also the averaged images (the last two columns). First row: image inpainting ($\mathcal{A} = \mathcal{P}_\Omega$) from 30% pixels chosen uniformly at random and with 1% Gaussian noise; Second row: compressive image recovery ($\mathcal{A} = \mathcal{P}_\Omega \circ \mathcal{C}_2$) from 30% measurements chosen uniformly at random and with 1% Gaussian noise; Third row: "average" debluring with 1% Gaussian noise; Fourth row: "motion" debluring with 1% Gaussian noise.

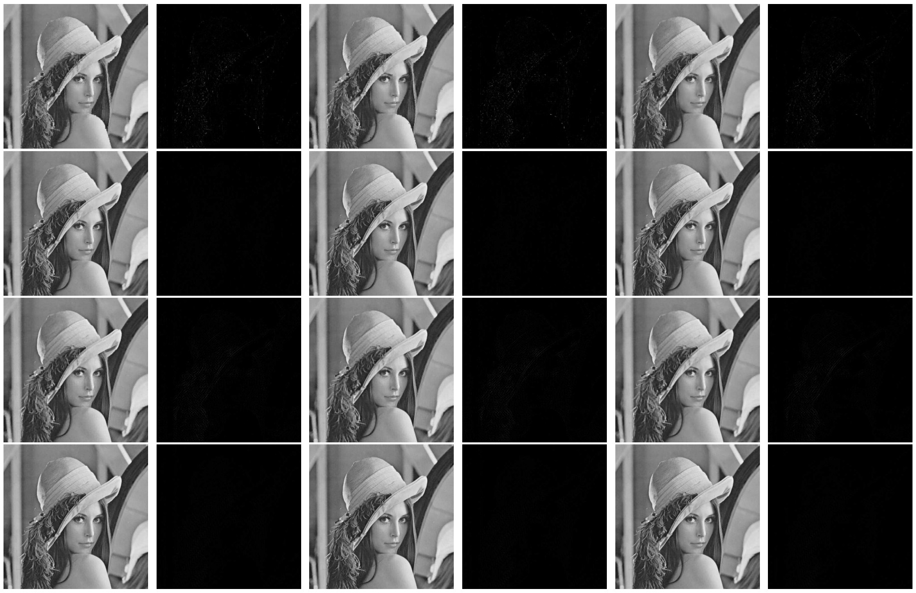
Appendix A. **Recovered images and residuals.** Figures 4 and 5 depict the recovered images and their residuals corresponding to the smallest and greatest PSNRs and also the averaged ones of the first run for Table 2.

REFERENCES

[1] M. AHARON, M. ELAD, AND A. BRUCKSTEIN, *K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation*, Signal Processing, IEEE Transactions on, 54 (2006), pp. 4311–4322.

[2] GHOLAMREZA ANBARJAFARI AND HASAN DEMIREL, *Image super resolution based on interpolation of wavelet domain high frequency subbands and the spatial domain input image*, ETRI J, 32 (2010), pp. 390–394.

FIGURE 5. Recovered Lena images and their residuals corresponding to the smallest (the first two columns) and greatest (the 3rd and 4th columns) PSNRs and also the averaged images (the last two columns). First row: image inpainting ($\mathcal{A} = \mathcal{P}_\Omega$) from 30% pixels chosen uniformly at random and with 1% Gaussian noise; Second row: compressive image recovery ($\mathcal{A} = \mathcal{P}_\Omega \circ \mathcal{C}_2$) from 30% measurements chosen uniformly at random and with 1% Gaussian noise; Third row: "average" debluring with 1% Gaussian noise; Fourth row: "motion" debluring with 1% Gaussian noise.



[3] A. BECK AND M. TEBOULLE, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, SIAM Journal on Imaging Sciences, 2 (2009), pp. 183–202.

[4] E. BIERSTONE AND P.D. MILMAN, *Semianalytic and subanalytic sets*, Publications Mathématiques de l'IHÉS, 67 (1988), pp. 5–42.

[5] J. BOLTE, A. DANIILIDIS, AND A. LEWIS, *The Lojasiewicz inequality for nonsmooth subanalytic functions with applications to subgradient dynamical systems*, SIAM Journal on Optimization, 17 (2007), pp. 1205–1223.

[6] WEISHENG DONG, LEI ZHANG, GUANGMING SHI, AND XIAOLIN WU, *Image deblurring and super-resolution by adaptive sparse domain selection and adaptive regularization*, Image Processing, IEEE Transactions on, 20 (2011), pp. 1838–1857.

[7] MICHAEL ELAD AND MICHAL AHARON, *Image denoising via sparse and redundant representations over learned dictionaries*, Image Processing, IEEE Transactions on, 15 (2006), pp. 3736–3745.

[8] KJERSTI ENGAN, SVEN OLE AASE, AND JOHN HÅKON HUSØY, *Multi-frame compression: Theory and design*, Signal Processing, 80 (2000), pp. 2121–2140.

[9] LEYUAN FANG, SHUTAO LI, QING NIE, JOSEPH A IZATT, CYNTHIA A TOTH, AND SINA FARSIU, *Sparsity based denoising of spectral domain optical coherence tomography images*, Biomedical optics express, 3 (2012), pp. 927–942.

[10] KENNETH KREUTZ-DELGADO, JOSEPH F MURRAY, BHASKAR D RAO, KJERSTI ENGAN, TE-WON LEE, AND TERRENCE J SEJNOWSKI, *Dictionary learning algorithms for sparse representation*, Neural computation, 15 (2003), pp. 349–396.

[11] C LI, W YIN, AND Y ZHANG, *TVAL3: TV minimization by augmented lagrangian and alternating direction algorithms*, 2009.

[12] S. Łojasiewicz, *Sur la géométrie semi-et sous-analytique*, Ann. Inst. Fourier (Grenoble), 43 (1993), pp. 1575–1595.

[13] Julien Mairal, Francis Bach, and Jean Ponce, *Task-driven dictionary learning*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 34 (2012), pp. 791–804.

[14] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, *Online dictionary learning for sparse coding*, in Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 689–696.

[15] Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman, *Supervised dictionary learning*, arXiv preprint arXiv:0809.3083, (2008).

[16] 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, vol. 2, IEEE, 2001, pp. 416–423.

[17] Ignacio Ramirez, Pablo Sprechmann, and Guillermo Sapiro, *Classification and clustering via dictionary learning with structured incoherence and shared features*, in Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, 2010, pp. 3501–3508.

[18] Saiprasad Ravishankar and Yoram Bresler, *Mr image reconstruction from highly undersampled k-space data by dictionary learning*, Medical Imaging, IEEE Transactions on, 30 (2011), pp. 1028–1041.

[19] Karl Skretting and Kjersti Engan, *Recursive least squares dictionary learning algorithm*, Signal Processing, IEEE Transactions on, 58 (2010), pp. 2121–2130.

[20] Ivana Tosic and Pascal Frossard, *Dictionary learning*, Signal Processing Magazine, IEEE, 28 (2011), pp. 27–38.

[21] P. Tseng, *Convergence of a block coordinate descent method for nondifferentiable minimization*, Journal of Optimization Theory and Applications, 109 (2001), pp. 475–494.

[22] Y. Xu and W. Yin, *A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion*, SIAM Journal on Imaging Sciences, 6 (2013), pp. 1758–1789.

[23] Y. Xu, W. Yin, and S. Osher, *Learning circulant sensing kernels*, To appear in Inverse Problems and Imaging, (2014).

[24] Y Zhang, J Yang, and Wotao Yin, *YALL1: Your algorithms for l1*, MATLAB software, http://yall1.blogs.rice.edu/, (2010).

[25] Yongqiang Zhao, Jinxiang Yang, Qingyong Zhang, Lin Song, Yongmei Cheng, and Quan Pan, *Hyperspectral imagery super-resolution by sparse representation and spectral regularization*, EURASIP Journal on Advances in Signal Processing, 2011 (2011), pp. 1–10.

E-mail address: yangyang.xu@rice.edu
E-mail address: wotaoyin@math.ucla.edu