TRANSFORM LEARNING BASED IMAGE AND VIDEO PROCESSING

BY

BIHAN WEN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Yoram Bresler

ABSTRACT

In recent years, sparse signal modeling, especially using the synthesis dictionary model, has received much attention. Sparse coding in the synthesis model is, however, NP-hard. Various methods have been proposed to learn such synthesis dictionaries from data. Numerous applications such as image denoising, magnetic resonance image (MRI), and computed tomography (CT) reconstruction have been shown to benefit from a good adaptive sparse model. Recently, the sparsifying transform model has received interest, for which sparse coding is cheap and exact, and learning, or data-driven adaptation admits computationally efficient solutions. In this thesis, we present two extensions to the transform learning framework, and some applications.

In the first part of this thesis, we propose a union of sparsifying transforms model. Sparse coding in this model reduces to a form of clustering. The proposed model is also equivalent to a structured overcomplete sparsifying transform model with block cosparsity, dubbed OCTOBOS. The alternating algorithm introduced for learning such transforms involves simple closed-form solutions. Theoretical analysis provides a convergence guarantee for this algorithm. It is shown to be globally convergent to the set of partial minimizers of the non-convex learning problem. When applied to images, the algorithm learns a collection of well-conditioned square transforms, and a good clustering of patches or textures. The resulting sparse representations for the images are better than those obtained with a single learned transform, or with analytical transforms. We show the promising performance of the proposed approach in image denoising, which compares quite favorably with approaches involving a single learned square transform or an overcomplete synthesis dictionary, or Gaussian mixture models. The proposed denoising method is also faster than the synthesis dictionary based approach.

Next, we develop a methodology for online learning of square sparsifying transforms. Such online learning can be particularly useful when dealing with

big data, and for signal processing applications such as real-time sparse representation and denoising. The proposed transform learning algorithms are shown to have a significantly lower computational cost than online synthesis dictionary learning. In practice, the sequential learning of a sparsifying transform typically converges faster than batch mode transform learning. Preliminary experiments show the usefulness of the proposed schemes for sparse representation, and denoising.

In the third part, we present a video denoising framework based on online 3D sparsifying transform learning. The proposed scheme has low computational and memory costs, and can handle streaming video. Our numerical experiments show promising performance for the proposed video denoising method compared to popular prior or state-of-the-art methods.

To my parents and fiancé, for their love and support.

ACKNOWLEDGMENTS

My thanks go to my great adviser, Prof. Yoram Bresler, for his inspiring guidance, engaging suggestions, rewarding discussions, and endless encouragement throughout this work. One may think that a good adviser fills a young mind with knowledge. Yes, but moreover, Prof. Bresler gave me a compass so that the knowledge leads to exciting new ideas and fruitful outcomes.

I would like to thank my colleague Dr. Saiprasad Ravishankar for various collaborations with me, as well as other members of our research group: Yanjun Li, Luke Pfister, Dr. Kiryung Lee, and Elad Yarkoni for the many useful discussions that we had. I have been extremely fortunate being a part of this group.

I owe special thanks to my parents for their support through my entire life, and my fiancé Jing Yang, without whose love, understanding, and encouragement I would not have finished this thesis.

Last but not least, I would like to acknowledge the support for this work from the National Science Foundation (NSF) under grants CCF-1018660, and CCF-1320953.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The sparsity of signals and images in a certain transform domain or dictionary has been heavily exploited in signal and image processing. It is well known that natural signals and images have an essentially sparse representation (few significant non-zero coefficients) in analytical transform domains such as discrete cosine transform (DCT) and wavelets [8]. This property has been used in designing various compression algorithms and in compression standards such as JPEG2000 [9].

Well-known models for sparsity include the synthesis, the analysis [10], and the transform models [11, 12]. Each of these models can be used to generate sparse representations, or sparse codes for a signal. However, sparse coding in the transform model is computationally much cheaper than in the other models. Recent research has focused on adaptation of sparse models to data [13, 14, 15, 12], which turns out to be advantageous in applications. In particular, the learning of transform models has been shown to be much cheaper than synthesis, or analysis learning. Adaptive transforms also provide better signal/image reconstruction quality in applications [16, 17, 18, 19].

In this chapter, we introduce and discuss the various sparse models and their learning in more detail. The discussion will highlight the various drawbacks of prior models. Our own contributions are then discussed.

1.1 Synthesis and Analysis Model

The *synthesis model* suggests that a real-world signal $y \in \mathbb{R}^n$ satisfies $y = Dx + e$ with $D \in \mathbb{R}^{n \times L}$ a synthesis dictionary, $x \in \mathbb{R}^L$ sparse, and e an approximation term in the signal domain [20]. We say that $x \in \mathbb{R}^L$ is sparse if $\|x\|_0 \ll L$, where the l_0 quasi norm counts the number of non-zero entries in x. When $L = n$ and D is full rank, it forms a basis. When $L > n$, the fat

matrix D is said to be an overcomplete dictionary.

Given a signal y and dictionary D, the well-known synthesis sparse coding problem finds the sparse code x that minimizes $\|y - Dx\|_2^2$ subject to $\|x\|_0 \leq s$, where s is the required sparsity level. This synthesis sparse coding problem is, however, NP-hard (non-deterministic polynomial-time hard) [21, 22]. Various algorithms [23, 24, 25, 26, 27, 28, 29] have been proposed to solve this problem. While some of these algorithms are guaranteed to provide the correct solution under certain conditions, these conditions are often violated in applications. Furthermore, these algorithms are typically computationally expensive when used for large-scale problems in image processing and computer vision.

Adapting the synthesis model turns out to be advantageous in applications. Learned synthesis dictionaries have been demonstrated to be useful in applications such as denoising, inpainting, deblurring, and demosaicing [3, 30, 31, 32]. Additionally, inverse problems such as those in computed tomography (CT) [33], and magnetic resonance imaging (MRI) [34, 35] benefit from an adaptive synthesis model.

The synthesis dictionary learning problem has been studied in many recent papers [36, 37, 13, 38]. Given a matrix $Y \in \mathbb{R}^{n \times N}$ whose columns represent training signals, the problem of learning an adaptive dictionary D that gives a sparse representation for the signals in Y can be formulated as follows [13]:

$$\text{(P0s)} \quad \min_{D,X} \|Y - DX\|_F^2 \;\; s.t. \;\; \|X_i\|_0 \leq s \;\; \forall \; i$$

Here, the subscript i indexes the i^{th} column of a matrix. The columns of the matrix $X \in \mathbb{R}^{n \times N}$ denote the sparse codes of the columns (or, signals) of Y, and s denotes the sparsity level allowed for each training signal. Various algorithms have been proposed for synthesis dictionary learning [37, 13, 38, 39, 40, 41, 42, 43], that typically alternate between a *sparse coding step* (solving for, or updating X), and a *dictionary update step* (solving for D [1]). Among these various algorithms, the K-SVD method [13] has been particularly popular and demonstrated to be useful in numerous applications such as denoising, inpainting, deblurring, and MRI. However, since (P0s) is

[1] Some algorithms (e.g., K-SVD) also update the non-zero coefficients of the sparse code X in the dictionary update step.

non-convex and NP-hard, methods such as K-SVD [2] can get easily caught in local minima, or saddle points [44].

While the synthesis model has received enormous attention, the *analysis* approach [10] has also been gaining traction recently. The analysis model suggests that a signal $y \in \mathbb{R}^n$ satisfies $\|\Omega y\|_0 \ll m$, where $\Omega \in \mathbb{R}^{m \times n}$ is known as the analysis dictionary. A more general *noisy signal analysis model* [45, 14] has also been studied, where the signal $y \in \mathbb{R}^n$ is modeled as $y = z + e$ with $\Omega z \in \mathbb{R}^m$ sparse, i.e., $\|\Omega z\|_0 \ll m$. Here, e is a noise term that is assumed to be small in the signal domain. Given the noisy signal y and analysis dictionary Ω, the *analysis sparse coding* problem [14] finds z by minimizing $\|y - z\|_2^2$ subject to $\|\Omega z\|_0 \leq m - l$, where l is referred to as the cosparsity level (number of zeros) [14]. This problem too is NP-hard and similarly to sparse coding in the synthesis model, approximate algorithms exist for analysis sparse coding [46, 45, 47, 48, 14, 49, 50, 51]. The learning of analysis dictionaries has also been studied in several recent papers [52, 53, 54, 45, 55, 14, 56, 57]. The analysis learning problems are typically non-convex and NP-hard, and the various learning algorithms tend to be computationally expensive.

1.2 Transform Model for Sparse Representation

Very recently [12], our group investigated a generalized analysis model called the *transform model*, which suggests that a signal $y \in \mathbb{R}^n$ is approximately sparsifiable using a transform $W \in \mathbb{R}^{m \times n}$, that is, $Wy = x + e$, where $x \in \mathbb{R}^m$ is sparse, i.e., $\|x\|_0 \ll m$. Here, e is the approximation error, which is assumed to be small. The distinguishing feature from the synthesis and from the noisy analysis models is that this approximation error is in the transform rather than in the signal domain.

When $m = n$, the transform $W \in \mathbb{R}^{n \times n}$ is called a square transform. On the other hand, for $m > n$, the transform is called a tall or overcomplete transform. Various analytical transforms are known to approximately sparsify natural signals, such as the discrete cosine transform (DCT), wavelets [8], ridgelets [58], contourlets [59], and curvelets [60]. The transform model

[2] In fact, the K-SVD method, although popular, does not have any convergence guarantees.

has been shown to be more general than both the analysis and the noisy signal analysis models [12].

When a sparsifying transform W is known for the signal y, *transform sparse coding* finds a sparse code x of sparsity s by minimizing the sparsification error $\|Wy - x\|_2^2$ subject to $\|x\|_0 \leq s$. This problem is easy and its solution is obtained exactly by zeroing out all but the s coefficients of largest magnitude in the vector Wy. In contrast, sparse coding with synthesis or analysis dictionaries involves solving NP-hard problems approximately. Given W and sparse code x, one can also recover a least squares estimate of the signal y by minimizing the residual $\|Wy - x\|_2^2$ over y. The recovered signal is $W^\dagger x$, with $W^\dagger$ denoting the pseudo-inverse of W.

Adapting the transform to data provides advantages in many applications [12, 61, 17, 16, 18, 19]. Learned transforms provide better signal/image representations than analytical transforms such as the DCT or wavelets. Moreover, compared to the synthesis and analysis models, the transform model allows for exact and extremely fast computations. Transform learning formulations also do not involve highly non-convex functions involving the product of multiple unknown matrices (such as in Problem (P0s)). Thus, in spite of its apparent similarity to the analysis and the noisy analysis model, the transform model enjoys important advantages over the noisy analysis or synthesis models, whether as a fixed or data-driven, adaptive model.

1.3 Summary of Contributions

One drawback of the current transform learning problems and algorithms is that they are restricted to the case of square transforms [12, 17, 16, 18]. For natural images with highly complicated structures, a single learned square transform may not provide sufficient sparsification. In the thesis, we propose a novel problem formulation and algorithm for learning structured overcomplete sparsifying transforms with block cosparsity constraint (OCTOBOS), which can capture such data diversity and structure. Our algorithm is an alternating minimization algorithm, and each step of the algorithm involves simple (computationally cheap) closed-form solutions. Sparse coding in the proposed OCTOBOS model reduces to a form of clustering and is computationally inexpensive. Novel convergence guarantee for the proposed alter-

nating OCTOBOS learning algorithm is provided. Our adapted OCTOBOS model provides a better sparse representation of images than adaptive single square transforms and analytical transforms such as the DCT. We present an adaptive image denoising formulation and algorithm exploiting the OCTOBOS model in this work. The denoising performance of the proposed approach is better than that obtained using adaptive square transforms, or adaptive overcomplete synthesis dictionaries (K-SVD). Our denoising scheme also performs better than the well-known Gaussian mixture model (GMM) approach [4], and is comparable to the state-of-the-art BM3D denoising [2] in some cases.

Though transform learning schemes demonstrate advantages in computational efficiency, prior works about transform learning focused on batch learning [12, 1], where the sparsifying transform is adapted using all the training data simultaneously. In the thesis, our proposed learning framework adapts sequentially the sparsifying transform and sparse codes (and/or signal estimates) for signals (or, measurements) that arrive, or are processed, sequentially. Such online/sequential transform learning is amenable to big data, and to applications such as real-time sparse representation (compression), denoising, and compressed sensing. Additionally, we also extend the online transform learning to higher dimensional data, and propose the video denoising scheme by using learned 3D sparsifying transforms. Our numerical results demonstrate the promising performance of the proposed method as compared to popular or state-of-art methods.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 focuses on learning structured overcomplete sparsifying transforms with efficient implementations. We propose a union of sparsifying transform model, which is also equivalent to overcomplete sparsifying transform model with block cosparsity, dubbed OCTOBOS. Efficient learning and image denoising algorithms using OCTOBOS are presented with convergence guarantee. The applications including image segmentation (data classification), sparse representation, and image denoising are demonstrated. In Chapter 3, a novel methodology for online learning of square sparsifying transform is discussed. We

present highly efficient online/mini-batch learning algorithms, and demonstrate their usefulness in big data applications, including denoising large-scale images. In Chapter 4, we extend to learning 3D square sparsifying transform for video denoising. The proposed framework adapts transform to extracted 3D patches from corrupted video and denoises them sequentially. We present numerical results which demonstrate promising performance that is better than that of well-known methods. Finally in Chapter 5, we conclude with remarks on possible future directions.

CHAPTER 2

STRUCTURED OVERCOMPLETE SPARSIFYING TRANSFORM LEARNING WITH BLOCK COSPARSITY

In this chapter, we investigate a union of square sparsifying transforms model in this work [1]. In this model, we consider a collection (union) of square transforms $\{W_i\}_{i=1}^K$. A candidate signal is said to match (or, belong to) a particular transform in the collection if that transform provides the *best sparsification* for the signal among all the transforms in the collection.

A motivation for the proposed model is that natural signals and images (even if they belong to a single class such as MRI images, music signals, etc.) need not be sufficiently sparsifiable by a single transform. For example, image patches from different regions of a natural image usually contain different features, or textures. Thus, having a union of transforms would allow groups of patches with common features (or, textures) to be better sparsfied by their own texture-specific transform.

We will show that this union of square transforms model can be interpreted as an overcomplete sparsifying transform model with an additional constraint of block cosparsity for the transform sparse code. Here, the overcomplete transform is formed by stacking the transforms in $\{W_i\}_{i=1}^K$ on top of each other. For the sake of brevity, we will also refer to our OverComplete TransfOrm model with BlOck coSparsity constraint as the OCTOBOS model. In the remainder of this chapter, we will use the terms 'union of transforms', or 'OCTOBOS' interchangeably, depending on the context.

In this chapter, we only briefly discuss the possibility of classification (or, segmentation) using our proposed transform learning scheme. Indeed, the classification application is not the focus of this work, and a detailed study of this application will be considered for future work. Instead, to illustrate the usefulness of the learned union of transforms/OCTOBOS model, we focus in this chapter on applications such as image representation and denoising. We also provide convergence guarantee results for OCTOBOS learning.

[1] The material of this chapter has previously appeared in [62, 74]

The proposed union of square transforms model and its various alternative interpretations are described in Section 2.1. In Section 2.2, we describe our algorithm for learning the proposed structured overcomplete sparsifying transform, and discuss the algorithm's computational properties. In Section 2.3, we provide the summary of the convergence analysis for our transform learning algorithm. The application of our transform learning framework to image denoising is discussed in Section 2.4. We then present experimental results demonstrating the convergence behavior, and the promising performance of our proposed approach in Section 2.5.

2.1 OCTOBOS Model and Learning Problem Formulation

2.1.1 The Union of Transforms Model

The square sparsifying transform model has been investigated [12] recently. Here, we extend the single square transform model to a union of transforms model, which suggests that a signal $y \in \mathbb{R}^n$ is approximately sparsifiable by a particular transform in the collection $\{W_k\}_{k=1}^K$, where $W_k \in \mathbb{R}^{n \times n} \, \forall \, k$ are themselves square transforms. Thus, there exists a particular W_k such that $W_k y = x + e$, with $x \in \mathbb{R}^n$ sparse, and a transform residual e that is sufficiently small.

Given a signal $y \in \mathbb{R}^n$, and a union (or, collection) of square transforms $\{W_k\}_{k=1}^K$, we need to find the best matching transform (or, model) for the signal that gives the smallest sparsification error. This can be formulated as the following sparse coding problem:

$$\text{(P1)} \quad \min_{1 \leq k \leq K} \min_{z^k} \left\| W_k y - z^k \right\|_2^2 \quad s.t. \;\; \left\| z^k \right\|_0 \leq s \;\; \forall \; k$$

Here, z^k denotes the sparse representation of y in the transform W_k, with the maximum allowed sparsity level being s. We assume that the W_k's are all identically scaled in (P1). Otherwise, they can be rescaled (for example, to unit spectral or Frobenius norm) prior to solving (P1).

In order to solve (P1), we first find the optimal sparse code $\hat{z}^k$ for each

[2] k as $\hat{z}^k = H_s(W_k y)$, where the operator $H_s(\cdot)$ is the projector onto the s-ℓ_0 ball, i.e., $H_s(b)$ zeros out all but the s elements of largest magnitude in $b \in \mathbb{R}^n$. If there is more than one choice for the s coefficients of largest magnitude in a vector b, which can occur when multiple entries in b have identical magnitude, then we choose $H_s(b)$ as the projection of b for which the indices of the s largest magnitude elements in b are the lowest possible. Now, Problem (P1) reduces to

$$\min_{1 \leq k \leq K} \|W_k y - H_s(W_k y)\|_2^2 \tag{2.1}$$

To solve the above problem, we compute the sparsification error (using the optimal sparse code above) for each k and choose the best transform $W_{\hat{k}}$ as the one that provides the smallest sparsfication error (among all the W_k's). This is an exact solution technique for Problem (P1). Problem (P1) then also provides us with an optimal sparse code $\hat{z}^{\hat{k}} = H_s(W_{\hat{k}} y)$ for y. Given such a sparse code, one can also recover a signal estimate by minimizing $\left\| W_{\hat{k}} y - \hat{z}^{\hat{k}} \right\|_2^2$ over all $y \in \mathbb{R}^n$. The recovered signal is then given by $\hat{y} = W_{\hat{k}}^{-1} \hat{z}^{\hat{k}}$.

Since Problem (P1) matches a given signal y to a particular transform, it can be potentially used to cluster a collection of signals according to their transform models. The sparsification error term in (2.1) can be viewed as a clustering measure in this setting. This interpretation of (P1) indicates the possible usefulness of the union of transforms model in applications such as classification.

2.1.2 The Overcomplete Transform Model Interpretation

We now propose an interpretation of the union of transforms model as a structured overcomplete transform model (or, the OCTOBOS model). The 'equivalent' overcomplete transform is obtained from the union of transforms by stacking the square sub-transforms as $W = \begin{bmatrix} W_1^T \mid W_2^T \mid \ \ldots \ \mid W_K^T \end{bmatrix}^T$. The tall matrix $W \in \mathbb{R}^{m \times n}$, with $m = Kn$, and thus, $m > n$ (overcomplete transform) for $K > 1$.

The signal y is assumed to obey the model $Wy = x + e$, where the $x \in \mathbb{R}^m$

[2] For each k, this is identical to the single transform sparse coding problem.

is assumed to be "block cosparse", and e is a small residual. The block cosparsity of x is defined here using the following ℓ_0-type norm:

$$\|x\|_{0,s} = \sum_{k=1}^{K} I(\left\|x^k\right\|_0 \leq s) \tag{2.2}$$

Here, $x^k \in \mathbb{R}^n$ is the block of x corresponding to the transform W_k in the tall W, and s is a given sparsity level for block x^k. The operator $I(\cdot)$ above is an indicator function with $I(Q) = 1$ when statement Q is true, and $I(Q) = 0$ otherwise. We say that x is 1-block cosparse if there is exactly one block of x with at least $n - s$ zeros, i.e., $\|x\|_{0,s} = 1$ in this case.

In the proposed overcomplete transform model for signal y, we formulate the following sparse coding problem, which we refer to as the OCTOBOS sparse coding problem.

$$\text{(P2)} \quad \min_{x} \|Wy - x\|_2^2 \quad s.t. \quad \|x\|_{0,s} \geq 1$$

Problem (P2) finds an x with at least one block that has $\geq n - s$ zeros. In particular, we now introduce the following proposition, that the Problems (P1) and (P2) are equivalent. The detailed proof is included in [62, 63]. This equivalence is the basis for the interpretation of the union of transforms model as an overcomplete transform model.

Proposition 1. *The minimum values of the sparsification errors in Problems (P1) and (P2) are identical. The optimal sparse code(s) in (P1) is equal to the block(s) of the optimal $\hat{x}$ in (P2) satisfying $\left\|\hat{x}^k\right\|_0 \leq s$.*

The optimal $\hat{x}$ in (P2) by itself cannot be called a sparse code, since it typically has many more non-zeros than zeros.[3] However, the particular s-sparse block(s) of $\hat{x}$ can be considered as a sparse code, and one could also recover a signal estimate from this code similar to the union of transforms case.[4] Note that the many non-zeros in $\hat{x}$ help keep the overcomplete transform residual small.

The OCTOBOS model enforces a block cosparsity constraint. Alternatively, one could consider the model $Wy = x + e$ with a tall transform

[3]For example, when vector Wy has no zeros, then the optimal $\hat{x}$ in (P2) has exactly $n - s \ll Kn$ (for large K) zeros – all the zeros are concentrated in a single block of $\hat{x}$.

[4]More precisely, the index of the sparse block is also part of the sparse code. This adds just $\log_2 K$ bits per index to the sparse code.

$W \in \mathbb{R}^{Kn \times n}$, but without any block cosparsity constraint on x, and assuming that x has at least $n - s$ zeros, i.e., $\|x\|_0 \leq (K-1)n + s$. The sparse coding in this model would be identical to thresholding (zeroing out the $n-s$ elements of smallest magnitude of) Wy. However, it is unclear how to easily combine the non-zeros and zeros to form a length n sparse code.[5] Therefore, we do not pursue this case (non-block cosparse model) in this work.

2.1.3 An OCTOBOS Optimality Property

Here, we consider two data matrices $Y_1 \in \mathbb{R}^{n \times N}$ and $Y_2 \in \mathbb{R}^{n \times M}$ (columns of the matrices represent signals), each of which is sparsified by a different square transform. We provide a condition under which using just one of the two transforms for both Y_1 and Y_2 will increase the total sparsification error (computed over all signals in Y_1 and Y_2). Thus, when the proposed condition holds, the union of transforms provides a better model for the collection of data compared to any one transform.

The proposed condition is based on the spark property [64]. The proof is included in [62, 63]. For a matrix $A \in \mathbb{R}^{n \times r}$, the spark is defined to be the minimum number of columns of A that are linearly dependent.

Proposition 2. *Given two sets of data $Y_1 \in \mathbb{R}^{n \times N}$ and $Y_2 \in \mathbb{R}^{n \times M}$, suppose there exist non-identical and non-singular square transforms $W_1, W_2 \in \mathbb{R}^{n \times n}$, that exactly sparsify the datasets as $W_1 Y = X_1$ and $W_2 Y_2 = X_2$, where the columns of both X_1 and X_2 have sparsity $\leq s$. If* $\text{spark}\left[W_1^{-1} \mid W_2^{-1}\right] > 2s$, *then the columns of $W_2 Y_1$ have sparsity $> s$.*

If the spark condition above holds, then the sparsification errors of the columns of Y in W_2 (using sparsity level s) are strictly positive [62, 63]. We can also derive an alternative condition that involves the mutual coherence of $B = \left[W_1^{-1} \mid W_2^{-1}\right]$. The mutual coherence of the matrix B [20] is defined as follows:

$$\mu(B) = \max_{1 \leq k,j \leq m, k \neq j} \frac{\left|B_k^T B_j\right|}{\|B_k\|_2 \cdot \|B_j\|_2} \tag{2.3}$$

Unlike the spark, the mutual coherence is easy to compute, and characterizes the dependence between the columns (indexed by the subscripts j and k in

[5] We need a length n code in a square and invertible sub-transform of W in order to perform signal recovery uniquely.

(2.3)) of matrix B. It is known that the spark and mutual coherence of a matrix B are related as follows [20]:

$$\text{spark}(B) \geq 1 + \frac{1}{\mu(B)} \tag{2.4}$$

Therefore, in Proposition 2, the spark condition can be replaced by the following (more stringent) sufficient condition involving the mutual coherence of B.

$$\mu(B) < \frac{1}{2s-1} \tag{2.5}$$

If the above condition holds, then by equation (2.4), the spark condition of Proposition 2 automatically holds, and thus we will have that the columns of $W_2 Y_1$ have sparsity $> s$.

The spark-based sufficient condition in Proposition 2 can be interpreted as a similarity measure between the models W_1 and W_2. In the extreme case, when $W_1 = W_2$, the aforementioned matrix B has minimum possible spark ($= 2$). In broad terms, if W_1 and W_2 are sufficiently different, as measured by the spark condition in Proposition 2, or the coherence condition in (2.5), then the union of transforms model, or OCTOBOS, provides a better model than either one of the transforms alone.

The difference between W_1 and W_2 as measured by the spark, or coherence conditions, is invariant to certain transformations. In particular, if W_1 is an exact full rank sparsifier of matrix Y_1, then one can also obtain equivalent transforms by permuting the rows of W_1, or by pre-multiplying W_1 with a diagonal matrix with non-zero diagonal entries. All these equivalent transforms sparsify Y_1 equally well (i.e., they provide the same sparsity level of s). It is easy to see that if the condition spark $\left[W_1^{-1} \mid W_2^{-1}\right] > 2s$ (or, alternatively, the mutual coherence-based condition) holds with respect to a particular W_1 and W_2 in Proposition 2, then it also automatically holds with respect to any other equivalent W_1 and equivalent W_2.

2.1.4 Square Transform Learning

Consider a training data matrix $Y \in \mathbb{R}^{n \times N}$ whose columns are the given training signals, and a sparse code matrix $X \in \mathbb{R}^{n \times N}$ whose columns are the sparse representations (or, sparse codes) of the corresponding columns of Y

in a sparsifying transform $W \in \mathbb{R}^{n \times n}$. Previous work [12] proposed to learn a (single) square sparsifying transform W and sparse code X that minimize the sparsification error given by $\|WY - X\|_F^2$. The sparsification error is the modeling error in the transform model, and hence we minimize it in order to learn the best possible transform model. Analytical transforms such as the Wavelets and DCT are known to provide low sparsification errors for natural images. The single square transform learning problem was proposed as follows:

$$\text{(P3)} \quad \min_{W,X} \ \|WY - X\|_F^2 + \lambda Q(W)$$
$$s.t. \ \ \|X_i\|_0 \leq s \ \ \forall \ i$$

where $Q(W) = -\log |\det W| + \|W\|_F^2$. Here, the subscript i denotes the i^{th} column of the sparse code matrix X. Problem (P3) has $Q(W)$ [6] as an additional regularizer in the objective to prevent trivial solutions. The log determinant penalty enforces full rank on W and eliminates degenerate solutions such as those with zero, or repeated rows. The $\|W\|_F^2$ penalty helps remove a scale ambiguity in the solution (the scale ambiguity occurs when the data admits an exactly sparse representation). Together, the log determinant and Frobenius norm penalty terms help control the condition number of the learned transform. Badly conditioned transforms typically convey little information and may degrade performance in applications.

It was shown [12] that the condition number $\kappa(W)$ can be upper bounded by a monotonically increasing function of $Q(W)$. Hence, minimizing $Q(W)$ encourages reduction of condition number. Given a normalized transform W and a scalar $a \in \mathbb{R}$, $Q(aW) \to \infty$ as the scaling $a \to 0$ or $a \to \infty$. Thus, $Q(W)$ also penalizes bad scalings. Furthermore, when $\lambda \to \infty$ in (P3), the condition number of the optimal transform(s) tends to 1, and the spectral norm (or, scaling) tends to $1/\sqrt{2}$.

We set $\lambda = \lambda_0 \|Y\|_F^2$ in (P3), where λ_0 is a constant. This setting makes Problem (P3) invariant to the scaling of the data Y as follows. When the data Y is replaced with aY ($a \in \mathbb{R}$, $a \neq 0$) in (P3), we can set $X = aX'$. Then, the objective function for this case becomes $a^2 \left(\|WY - X'\|_F^2 + \lambda_0 \|Y\|_F^2 \ Q(W) \right)$. Since this is just a scaled version of the objective in (P3), the minimization of

[6] The weights on the log-determinant and Frobenius norm terms are set to the same value in this thesis.

it over (W, X') (with sparse X') yields the same solution as (P3). Thus, the learned transform for data aY is the same as for Y, while the learned sparse code for aY is a times that for Y. This makes sense since the sparsifying transform for a dataset is not expected to change, when the data is trivially scaled. Thus, the setting $\lambda = \lambda_0 \|Y\|_F^2$ achieves scale invariance for the solution of (P3).[7]

2.1.5 OCTOBOS Learning Formulations and Properties

Problem Formulations

Similar to the square sparsifying transform learning problem, we propose the following OCTOBOS learning formulation that learns a tall sparsifying transform $W \in \mathbb{R}^{Kn \times n}$ and sparse code matrix $X \in \mathbb{R}^{Kn \times N}$ from training data $Y \in \mathbb{R}^{n \times N}$.

$$\begin{aligned} \text{(P4)} \quad & \min_{W,X} \ \|WY - X\|_F^2 + Q'(W) \\ & s.t. \ \ \|X_i\|_{0,s} \geq 1 \ \ \forall \ i \end{aligned}$$

Here, $\|X_i\|_{0,s}$ is defined as in equation (2.2). The function $Q'(W)$ is defined as follows:

$$Q'(W) = \sum_{k=1}^{K} \lambda_k Q(W_k) \tag{2.6}$$

The regularizer $Q'(W)$ controls the condition number of the sub-blocks of W, and λ_k are positive weights.

One can also formulate the transform learning problem in the union of transforms model as follows:

$$\begin{aligned} \text{(P5)} \quad & \min_{\{W_k, X_i, C_k\}} \ \sum_{k=1}^{K} \left\{ \sum_{i \in C_k} \|W_k Y_i - X_i\|_2^2 + \lambda_k Q(W_k) \right\} \\ & s.t. \ \ \|X_i\|_0 \leq s \ \ \forall \ i, \ \ \{C_k\} \in G \end{aligned}$$

[7] On the other hand, if λ is a fixed constant, there is no guarantee that the optimal transforms for scaled and un-scaled Y in (P3) are related.

Here, the set $\{C_k\}_{k=1}^K$ indicates a clustering of the training signals $\{Y_i\}_{i=1}^N$. The cluster C_k contains the indices i corresponding to the signals Y_i in the k$^{\text{th}}$ cluster. The signals in the k$^{\text{th}}$ cluster are matched to transform W_k. The set G is the set of all possible partitions of the set of integers $[1:N] \triangleq \{1, 2, ..., N\}$, or in other words, G is the set of all possible $\{C_k\}$, and is defined as follows:

$$G = \left\{ \{C_k\} : \bigcup_{k=1}^{K} C_k = [1:N],\, C_j \bigcap C_k = \emptyset,\, \forall\, j \neq k \right\}$$

The constraint involving G thus enforces the various C_k in $\{C_k\}_{k=1}^K$ to be disjoint, and their union to contain the indices for all training signals. Note that the term $\sum_{k=1}^K \sum_{i \in C_k} \|W_k Y_i - X_i\|_2^2$ in (P5) is the sparsification error for the data Y in the union of transforms model.

The weights λ_k in (P4) and (P5) are chosen as $\lambda_k = \lambda_0 \|Y_{C_k}\|_F^2$, where Y_{C_k} is a matrix whose columns are the signals of Y in the k$^{\text{th}}$ cluster. The rationale for this choice of λ_k is similar to that presented earlier for the λ weight in (P3). Specifically, when the clusters $\{C_k\}_{k=1}^K$ are fixed to their optimal values in (P5), the optimization problem (P5) reduces to K square transform learning problems of the form of (P3), each involving a particular data matrix Y_{C_k}. Thus, the setting $\lambda_k = \lambda_0 \|Y_{C_k}\|_F^2$ achieves scale invariance for the solutions of these K problems. The setting also implies that λ_k itself is a function of the unknown C_k (function of the signal energy in cluster C_k) in the optimization problem (P5). When the $\{W_k\}$ are fixed, the $Q'(W)$ penalty in (P5) encourages a larger concentration of data energy ($\|Y_{C_k}\|_F^2$) in the cluster corresponding to a smaller $Q(W_k)$ (i.e., corresponding to smaller condition number and reasonable scaling).

Properties of Formulations (P4) and (P5)

The following results [63, 62] imply that the learning Problems (P4) and (P5) are equivalent. The details of the proof are presented in [63, 62].

Lemma 1. *The minimum values of the objectives in Problems (P4) and (P5) are identical. Moreover, any optimal union of transforms in (P5) can be vertically concatenated to form an optimal overcomplete W for (P4). Similarly, any optimal W in (P4) can be used to generate an optimal union of*

transforms for (P5).

Although (P4) and (P5) are equivalent, Problem (P5) is more intuitive and amenable to alternating optimization schemes (see Section 2.2 for such a scheme). If we were to alternate between updating X and W in (P4), we would not be able to directly maintain (without additional constraints) the property that the transform domain residual for each Y_i is zero in all but (at most) one block of W, during the update of W. This is not a problem for (P5), since its objective only considers the residual of each Y_i in one (best) block.

The following result [63, 62] indicates that the minimum objective in the union of transforms Problem (P5) is always lower than the minimum objective value in the single transform learning problem (P3). This means that either the optimal sparsification error in (P5) is lower than the corresponding value in (P3), or the optimal regularizer (that controls the condition number(s) of the transform block(s)) in (P5) is smaller than the corresponding value in (P3), or both of these conditions hold.

Proposition 3. *The minimum value of the objective in (P5) can only be lower than the minimum objective value in (P3)[63, 62].*

We will empirically illustrate in Section 2.5 that our algorithm for (P5) (in Section 2.2) provides a lower value of both the objective and sparsification error compared to the algorithm for (P3).

It was shown by [12] that the objective of Problem (P3) is lower bounded. The following lemma [63, 62] confirms that the objectives of the proposed learning formulations are lower bounded too.

Lemma 2. *The objectives in Problems (P4) and (P5) are both lower bounded by* $\lambda Q_0 = \lambda_0 \, Q_0 \, \|Y\|_F^2$*, where* $Q_0 = \frac{n}{2} + \frac{n}{2} \log(2)$.

We use the preceding lemma to prove the following proposition, which pertains to the identifiability of good models (models that sparsify well and are well-conditioned) by our Problem (P5). Proposition 4 [63, 62] also pertains to the case when the lower bounds in Lemma 2 are achievable.

Proposition 4. *Given a training data matrix* $Y \in \mathbb{R}^{n \times N}$*, let* $\{Y_{C_k}\}$ *be a collection of data matrices formed according to a clustering rule* $\{C_k\}$*. Suppose that* $\{W_k\}$ *is a collection of unit conditioned square transform models, each*

with spectral norm $1/\sqrt{2}$,[8] *that exactly sparsifies the clustered data* $\{Y_{C_k}\}$ *as* $W_k Y_{C_k} = X_{C_k}\ \forall\, k$, *with each* X_{C_k} *having s-sparse columns. Then, the set* $\{W_k, C_k, X_{C_k}\}$ *is a global minimizer of (P5), i.e., the underlying model is identifiable by solving (P5). [63, 62]*

Thus, when an "error-free" union of transforms model exists for the data, and the transforms are all unit conditioned, Proposition 4 guarantees that such a union of transforms model is a global minimizer of the proposed Problem (P5). Therefore, it makes sense to solve (P5) in order to find such good OCTOBOS models.

We now show that the role of the λ_0 weight in (P5) is to control the condition number and scaling of the transform blocks W_k ($1 \leq k \leq K$). If we were to minimize only the $\hat{Q}(W) = Q'(W)/\lambda_0 = \sum_{k=1}^{K} \|Y_{C_k}\|_F^2 \, Q(W_k)$ regularizer in Problem (P5) with respect to the unknowns, then the minimum value would be $Q_0 \|Y\|_F^2$ according to Lemma 2. This minimum is achieved with W_k's that are unit conditioned, and with spectral norm of $1/\sqrt{2}$ (i.e., transforms with identical scaling). Thus, similar to Corollary 2 in [12], we have that as $\lambda_0 \to \infty$ in (P5), the condition number of the optimal transforms in (P5) tends to 1, and their spectral norm (scaling) tends to $1/\sqrt{2}$. Therefore, as $\lambda_0 \to \infty$, our formulation (P5) approaches a union of unit-conditioned transforms learning problem. We also empirically show in Section 2.5 that when λ_0 is properly chosen (but finite), the condition numbers and norms of the learned W_k's in (P5) are very similar. Note that we need the W_k's to be similarly scaled for the sparsification error in (P5) to be fully meaningful (since otherwise, a certain W_k with a very small scaling can trivially give the best sparsification error for a signal).

Another interesting fact about OCTOBOS learning is that both (P4) and (P5) admit an equivalence class of solutions similar to (P3). For example, one can permute the rows within an optimal block W_k (along with a permutation of the corresponding sparse codes), or pre-multiply W_k by a diagonal ± 1 sign matrix (and multiply the sparse codes accordingly), without affecting its optimality. In (P4), one can also permute the blocks W_k within an optimal W (and correspondingly permute the sparse codes) to produce equivalent optimal solutions.

[8] If the transforms have a different spectral norm, they can be trivially scaled to have spectral norm $1/\sqrt{2}$.

We note that in spite of sharing the common theme of a mixture of models, our OCTOBOS model and learning formulation are quite different from the Gaussian mixture model (GMM) approach of [4], and [65]. In the GMM-based models, the signal can be thought of (cf. [4]) as approximated by a linear combination of a few (orthonormal) eigenvectors of the covariance matrix of the mixture component to which it belongs. In contrast, in the OCTOBOS approach, the transform blocks W_k (equivalently, the class-conditional square sparsifying transforms) are not eigenvectors of some covariance matrices. Instead they are directly optimized (via (P5)) for transform-domain sparsity of the training data. Our OCTOBOS learning also enforces well-conditioning rather than exact orthonormality of the transform blocks. These features distinguish our OCTOBOS framework from the GMM-based approach.

2.2 Transform Learning Algorithm and Properties

2.2.1 Algorithm

We propose an alternating algorithm to solve the joint minimization Problem (P5). In one step of our proposed algorithm called the *sparse coding and clustering step*, we solve for $\{C_k\}$, $\{X_i\}$ with fixed $\{W_k\}$ in (P5). In the other step of the algorithm called the *transform update step*, we solve for the transforms $\{W_k\}$ in (P5) with fixed sparse codes.

Sparse Coding and Clustering

Given the training matrix Y, and fixed transforms $\{W_k\}$ (or, the equivalent overcomplete W), we solve the following Problem (P6) (which is just (P5) with fixed transforms) to determine the sparse codes and clusters. As before, the clusters are disjoint and every training signal belongs to exactly one cluster.

$$\text{(P6)} \quad \min_{\{C_k\},\{X_i\}} \sum_{k=1}^{K} \sum_{i \in C_k} \left\{ \|W_k Y_i - X_i\|_2^2 + \eta_k \|Y_i\|_2^2 \right\}$$
$$s.t. \;\; \|X_i\|_0 \leq s \;\; \forall \; i, \;\; \{C_k\} \in G$$

The weight $\eta_k = \lambda_0 \, Q(W_k)$ above. This is a fixed weight, since W_k is fixed in this step. We refer to the term $\|W_k Y_i - X_i\|_2^2 + \eta_k \|Y_i\|_2^2$, with $X_i = H_s(W_k Y_i)$ (i.e., the optimal sparse code of Y_i in transform W_k) as a clustering measure corresponding to the signal Y_i. This is a modified version of the measure in (P1), and includes the additional penalty $\eta_k \|Y_i\|_2^2$ determined by the regularizer (i.e., determined by the conditioning of W_k [9]). It is easy to observe that the objective in (P6) involves the summation of only N such 'clustering measure' terms (one for each signal). Since every training signal is counted exactly once (in one cluster) in the double summation in Problem (P6), we can construct the equivalent optimization problem as follows.

$$\sum_{i=1}^{N} \min_{1 \leq k \leq K} \left\{ \|W_k Y_i - H_s(W_k Y_i)\|_2^2 + \eta_k \|Y_i\|_2^2 \right\} \tag{2.7}$$

The minimization over k for each Y_i above determines the cluster C_k (in (P6)) to which Y_i belongs. For each Y_i, the optimal cluster index $\hat{k}$ [10] is such that $\|W_{\hat{k}} Y_i - H_s(W_{\hat{k}} Y_i)\|_2^2 + \eta_{\hat{k}} \|Y_i\|_2^2 \leq \|W_j Y_i - H_s(W_j Y_i)\|_2^2 + \eta_j \|Y_i\|_2^2$, $\forall j \neq \hat{k}$. The optimal $\hat{X}_i$ in (P6) is then $H_s(W_{\hat{k}} Y_i)$. There is no coupling between the sparse coding/clustering problems in (2.7) for the different training signals $\{Y_i\}_{i=1}^N$. Thus, the training signals can be sparse coded and clustered in parallel.

Transform Update Step

Here, we solve for $\{W_k\}$ in (P5) with fixed $\{C_k\}$, $\{X_i\}$. Although this is an unconstrained joint minimization problem over the set of transforms, the optimization problem is actually separable (due to the objective being in summation form) into K unconstrained problems, each involving only a particular square transform W_k. Thus, the transform update problem becomes

$$\text{(P7)} \quad \min_{W_k} \sum_{i \in C_k} \|W_k Y_i - X_i\|_2^2 + \lambda_k Q(W_k)$$

[9] This clustering measure will encourage the shrinking of clusters corresponding to any badly conditioned, or badly scaled transforms.

[10] When two or more clusters are equally optimal, then we pick the one corresponding to the lowest cluster index k.

Here, $\lambda_k = \lambda_0 \|Y_{C_k}\|_F^2$ is a fixed weight. Problem (P7) is solved separately for each k, which can be done in parallel. Problem (P7) is similar to the transform update problem encountered for (P3) [1], and can thus be solved similarly.

Let U be the matrix whose columns are the training signals Y_i belonging to the k$^{\text{th}}$ cluster (i.e., $i \in C_k$). Let V be the corresponding (fixed) sparse code matrix. Problem (P7) can then be solved exactly and efficiently by using the simple closed-form solution proposed by [1]. First, we decompose the positive-definite matrix $UU^T + \lambda_k I$ as $UU^T + \lambda_k I = LL^T$ (e.g., by Cholesky decomposition, or taking the eigen value decomposition (EVD) square root), where I is the $n \times n$ identity. Next, we obtain the full singular value decomposiiton (SVD) of the matrix $L^{-1}UV^T$ as $B\Sigma R^T$, where B, Σ, and R are all $n \times n$ matrices. Then, the optimal transform $\hat{W}_k$ in (P7) is given as

$$\hat{W}_k = \frac{R}{2}\left(\Sigma + (\Sigma^2 + 2\lambda_k I)^{\frac{1}{2}}\right) B^T L^{-1} \tag{2.8}$$

where the $(\cdot)^{\frac{1}{2}}$ operation above denotes the positive definite square root. The closed-form solution (2.8) is guaranteed to be a global optimum of Problem (P7). Compared to iterative optimization methods such as conjugate gradients (CG), (2.8) allows for both cheap and exact computation of the solution of the transform update problem. The OCTOBOS learning Algorithm A1 is summarized in Fig. 2.1.

2.2.2 Computational Cost

The algorithm for (P5) consists of the sparse coding and clustering step, and the transform update step. We derive the computational cost of each of these steps.

Sparse coding and clustering. First, the sparse code of every training signal with respect to every transform W_k is computed. The computation of $W_k Y$ requires $n^2 N$ multiply-add operations. The projection of $W_k Y$ onto the s-ℓ_0 ball, if done by full sorting, would require $O(nN \log n)$ operations. Thus, the cost of finding the sparse representation of the training matrix in a particular W_k is dominated by $O(n^2 N)$. Since this needs to be done for each k, the total number of operations scales as $O(Kn^2 N)$. Denoting $m = Kn$ this cost is $O(mnN)$.

OCTOBOS Learning Algorithm A1

Input : Y - training matrix with N signals, s - sparsity level, λ_0 - constant, K - number of clusters, J - number of iterations.

Output : $\left\{\hat{W}_k\right\}$ - learned transforms, $\left\{\hat{X}_i\right\}$ - learned sparse codes, $\left\{C_k\right\}$ - learned clusters.

Initialize: $\left\{\hat{W}_k^0\right\}$.

For $t = 1$: J Repeat

1) **Clustering and Sparse Coding: For $1 \leq i \leq N$,**

   (a) Compute $\eta_k = \lambda_0 \, Q(\hat{W}_k^{t-1})$, $1 \leq k \leq K$.

   (b) Include the index i in the cluster $C_{\hat{k}}^{t-1}$ if $\left\|\hat{W}_{\hat{k}}^{t-1} Y_i - H_s(\hat{W}_{\hat{k}}^{t-1} Y_i)\right\|_2^2 + \eta_{\hat{k}} \left\|Y_i\right\|_2^2 \leq \eta_j \left\|Y_i\right\|_2^2 + \left\|\hat{W}_j^{t-1} Y_i - H_s(\hat{W}_j^{t-1} Y_i)\right\|_2^2$ holds for all $j \neq \hat{k}$.

   (c) $\hat{X}_i^{t-1} = H_s(\hat{W}_{\hat{k}}^{t-1} Y_i)$.

2) **Transform Update: For each $1 \leq k \leq K$, do**

   (a) Let $\Psi \triangleq C_k^{t-1}$. Compute $\lambda_k = \lambda_0 \left\|Y_\Psi\right\|_F^2$.

   (b) Compute $L^{-1} = \left(Y_\Psi Y_\Psi^T + \lambda_k I\right)^{-1/2}$.

   (c) Compute full SVD of $L^{-1} Y_\Psi (\hat{X}_\Psi^{t-1})^T$ as $B \Sigma R^T$.

   (d) $\hat{W}_k^t = \frac{R}{2}\left(\Sigma + (\Sigma^2 + 2\lambda_k I)^{\frac{1}{2}}\right) B^T L^{-1}$.

End

Figure 2.1: Algorithm A1 for OCTOBOS learning via (P5). A superscript of t is used to denote the iterates in the algorithm.

Table 2.1: Computational cost per-iteration for square sparsifying transform, OCTOBOS, and KSVD learning.

	Square Transform	OCTOBOS	KSVD
Cost	$O(n^2N)$	$O(mnN)$	$O(mn^2N)$

Next, for each training signal, in order to perform clustering, the clustering measure, which is the sum of the sparsification error and weighted signal energy, is computed with respect to all the transforms. The total cost of computing the sparsification error (taking into account that the sparsification error is only computed using the transform domain residual on the complement of the support of the sparse code) for all training signals in all clusters is $O((n-s)KN)$. Since the weighted signal energy term $\lambda_0 \|Y_i\|_2^2 Q(W_k)$ needs to be computed for all i, k, we first compute $\lambda_0 Q(W_k)$ for all k at a cost of $O(Kn^3)$ (computing determinants dominates this cost). Next, the term $\|Y_i\|_2^2$ is computed for all i at a cost of $O(nN)$. Then, computing $\lambda_0 \|Y_i\|_2^2 Q(W_k)$ and adding it to the corresponding sparsification error term for all i, k, requires $O(KN)$ operations. Finally, to compute the best cluster for each signal requires $O(K-1)$ comparisons (between the clustering measure values for different transforms), and thus $O((K-1)N)$ operations for the entire Y matrix. Assuming, $N \gg n$, it is clear based on the preceding arguments that the computation of $\{W_kY\}$ dominates the computations in the sparse coding and clustering step, with a cost of $O(mnN)$ (or, $O(Kn^2N)$).

Transform update. In this step, we compute the closed-form solution for the transform in each cluster using (2.8). First, the matrix $UU^T + \lambda_k I_n$ (notations defined in Section 2.2.1) needs to be computed for each (disjoint) cluster. This requires $O(n^2N)$ multiply-add operations totally (over all clusters). (Note that the computation of all the λ_k's requires only $O(nN)$ operations.) The computation of L and L^{-1} requires $O(n^3)$ operations for each cluster, and thus about $O(Kn^3)$ operations for K clusters. Next, the matrix UV^T is computed for each cluster. Since V has s-sparse columns, this matrix multiplication gives rise to a total (over all clusters) of $\alpha n^2 N$ multiply-add operations (assuming $s = \alpha n$, with $\alpha < 1$). Finally, the computation of $L^{-1}UV^T$, its SVD, and the closed-form update (2.8) require $O(n^3)$ operations per cluster, or about $O(Kn^3)$ operations for K clusters. Since

$N \gg m = Kn$ typically, we have that the cost of the transform update step scales as $O(n^2N)$. Thus, for $K \gg 1$, the transform update step is cheaper than the sparse coding step for the proposed algorithm.

Based on the preceding arguments, it is clear that the computational cost per iteration (of sparse coding and transform update) of our algorithm scales as $O(mnN)$.[11] This is much lower (in order) than the per-iteration cost of learning an $n \times m$ overcomplete synthesis dictionary D using K-SVD [13], which (assuming, as in the transform model, that the synthesis sparsity level $s = \beta n$ with $\beta < 1$) [12] scales as $O(mn^2N)$. Our transform learning also holds a similar (per-iteration) computational advantage over analysis dictionary learning schemes such as analysis K-SVD. The computational costs per-iteration of square transform, OCTOBOS, and KSVD learning are summarized in Table 2.1.

As illustrated in our experiments in Section 2.5, both the OCTOBOS and square transform learning algorithms converge in few iterations in practice. Therefore, the per-iteration computational advantages for OCTOBOS over K-SVD typically translate to a net computational advantage in practice.

OCTOBOS learning could be used for a variety of purposes including clustering (classification), denoising, and sparsity-based signal compression. In the latter case, we also need to compute $\hat{Y}_i = W_k^{-1} X_i$, for all $i \in C_k$, and $\forall\ k$, in order to recover estimates of the signals from their (compressed) transform codes. Computing W_k^{-1}, $1 \leq k \leq K$, has $O(Kn^3)$ computational cost. However, this cost does not depend on the number of training signals $N \gg K$ (typically), and is therefore negligible compared to the total cost $O(snN)$ $(= O(n^2N)$ for $s \propto n)$ of multiplying the once computed W_k^{-1} for all k, with the corresponding sparse codes. The latter cost is the same as for multiplying a synthesis dictionary D with its sparse codes.

[11] Setting $m = n$ for the case $K = 1$, this agrees with previous cost analysis for square transform learning using (P3), which has per-iteration cost of $O(n^2N)$ [12].

[12] The notion that sparsity s scales with the signal dimension n is rather standard. For example, while $s = 1$ may work for representing the 4×4 patches of an image in a DCT dictionary with $n = 16$, the same sparsity level of $s = 1$ for an $n = 256^2$ DCT dictionary for a 256×256 (vectorized) image would lead to very poor image representation. Therefore, the sparsity s must increase with the size n. A typical assumption is that the sparsity s scales as a fraction (e.g., 5% or 10%) of the image or, patch size n. Otherwise, if s were to increase only sub-linearly with n, it would imply that larger (more complex) images are somehow better sparsifiable, which is not true in general.

2.3 Convergence Analysis

While some recent works [66, 67, 68, 69, 70, 71] study the convergence of (specific) synthesis dictionary learning algorithms, [13] none of them consider the union of dictionaries case. The prior works also typically require many restrictive assumptions (e.g., noiseless data) for their results to hold. In contrast, we present a novel convergence theory here for learning a union of (transform) sparse models [63, 62]. Importantly, although our proposed Problem formulation (P5) is highly non-convex (due to the ℓ_0 quasi norm on the sparse codes and the log-determinant penalty), the results hold with few or no assumptions.

Very recently, a convergence result [72] has been derived for the algorithm for the (single) square transform learning Problem (P3). In this section, we analyze the convergence behavior of the proposed OCTOBOS learning algorithm that solves (P5), by summarizing the main theoretical results. The details of the proofs are presented elsewhere [63, 62].

The convergence results [63, 62] are more conveniently stated in terms of an unconstrained form of (P5). Problem (P5) has the constraint $\|X_i\|_0 \leq s \, \forall \, i$, which can equivalently be added as a penalty in the objective by using a barrier function $\phi(X)$ (where $X \in \mathbb{R}^{n \times N}$ is the matrix of *all* sparse codes), which takes the value $+\infty$ when the constraint is violated, and is zero otherwise. Then, we denote the objective of (P5) as

$$g\left(W, X, \Gamma\right) = \sum_{k=1}^{K} \sum_{i \in C_k} \|W_k Y_i - X_i\|_2^2 + \phi(X) + \sum_{k=1}^{K} \lambda_k Q(W_k) \tag{2.9}$$

where $W \in \mathbb{R}^{Kn \times n}$ is obtained by stacking the W_k's on top of one another, the matrix $X \in \mathbb{R}^{n \times N}$ contains the sparse codes X_i as its columns, and the row vector $\Gamma \in \mathbb{R}^{1 \times N}$ is such that its i$^{\text{th}}$ element $\Gamma_i \in \{1, .., K\}$ denotes the cluster index (label) corresponding to the signal Y_i. As discussed in Section 2.1.5, the clusters $\left\{C_k\right\}$ form a disjoint partitioning of $[1 : N]$.

Problem (P5) is to find the best possible union of sparsifying transforms

[13] Most of these (synthesis dictionary learning) algorithms have not been demonstrated to be practically useful in applications such as denoising. [70] show that their method denoises worse than the K-SVD method [3].

model for a given set of data Y, by minimizing the sparsification error, and controlling the condition numbers (avoiding trivial solutions) of the cluster-specific transforms. Proposition 4 in Section 2.1.5 established the identifiability of good models by solving Problem (P5). Here, we discuss the convergence behavior of our algorithm A1 that solves (P5).

For our convergence results, we make only the following mild assumption [63, 62].

Assumption 1. Our proposed solution involves computing SVDs, EVDs (of small $n \times n$ matrices), and scalar square roots. Although in practice these quantities are computed using iterative methods, we assume, for the theoretical analysis, that they are computed exactly. In practice, standard numerical methods are guaranteed to quickly provide machine precision accuracy for these computations.

Since the training matrix $Y \in \mathbb{R}^{n \times N}$, the training signals are also bounded, i.e., $\max_i \|Y_i\|_2 < \infty$. For a vector z, let $\beta_j(z)$ denote the magnitude of the j^{th} largest element (magnitude-wise) of z. For a matrix B, $\|B\|_\infty \triangleq \max_{i,j} |B_{ij}|$. We say that a sequence $\{b^t\}$ has an accumulation point b, if there is a subsequence that converges to b. For our iterative Algorithm A1 (in Fig. 2.1), we denote the iterates (or, outputs) at each iteration t by the set (W^t, X^t, Γ^t), where W^t denotes the matrix obtained by stacking the cluster-specific transforms W_k^t $(1 \leq k \leq K)$, X^t is the sparse code matrix with the sparse codes X_i^t $(1 \leq i \leq N)$ as its columns, and Γ^t is a row vector containing the cluster indices Γ_i^t $(1 \leq i \leq N)$ as its elements. Each Γ_i^t contains the cluster index corresponding to signal Y_i. The following theorem [63, 62] provides a convergence result for the OCTOBOS learning Algorithm A1.

Theorem 1. *Let $\{W^t, X^t, \Gamma^t\}$ denote the iterate sequence generated by Algorithm A1 with training data Y and initial (W^0, X^0, Γ^0). Then, the objective sequence $\{g^t\}$ with $g^t \triangleq g(W^t, X^t, \Gamma^t)$ is monotone decreasing, and converges to a finite value, say $g^* = g^*(W^0, X^0, \Gamma^0)$. The iterate sequence is bounded, and all its accumulation points are equivalent in the sense that they achieve the same value g^* of the objective. Finally, every accumulation point (W, X, Γ) is a fixed point of Algorithm A1 satisfying the following partial*

global optimality conditions:

$$(X, \Gamma) \in \underset{\tilde{X}, \tilde{\Gamma}}{\arg\min}\ g\left(W, \tilde{X}, \tilde{\Gamma}\right) \tag{2.10}$$

$$W \in \underset{\tilde{W}}{\arg\min}\ g\left(\tilde{W}, X, \Gamma\right) \tag{2.11}$$

as well as the following partial local optimality property:

$$g\left(W + dW, X + \Delta X, \Gamma\right) \geq g\left(W, X, \Gamma\right) \tag{2.12}$$

Property (2.12) *holds for all* dW *with sufficiently small perturbations to the individual cluster-specific transforms* $dW_k \in \mathbb{R}^{n \times n}$ *satisfying* $\|dW_k\|_F \leq \epsilon_k$ *for some* $\epsilon_k > 0$ *that depends on the specific* W_k*, and the following condition on* ΔX*. For every* $1 \leq k \leq K$*, let* $\Delta X_{C_k} \in \mathbb{R}^{n \times |C_k|}$ *be the matrix with columns* $\Delta X_i \in \mathbb{R}^n$ *for* $i \in C_k$*. Then,* ΔX *is such that* $\Delta X_{C_k} \in \mathrm{R1_k} \cup \mathrm{R2_k}$ *for all* k*, where*

R1$_k$*: The half-space* $tr\left\{(W_k Y_{C_k} - X_{C_k})\Delta X_{C_k}^T\right\} \leq 0$.

R2$_k$*: The* local region *defined by*
$\|\Delta X_{C_k}\|_\infty < \min_{i \in C_k} \{\beta_s(W_k Y_i) : \|W_k Y_i\|_0 > s\}$.

Furthermore, if we have $\|W_k Y_i\|_0 \leq s \,\forall\, i \in C_k$*, then the corresponding* ΔX_{C_k} *can be arbitrary.*

The local region $\mathrm{R2}_k$ in Theorem 1 that determines the size of the local perturbation ΔX_{C_k} in class k is determined by the scalar $\gamma_k = \min_{i \in C_k} \{\beta_s(W_k Y_i) : \|W_k Y_i\|_0 > s\}$. This scalar is computed by (i) taking the columns of $W_k Y$ corresponding to the k^{th} cluster; (ii) choosing only the vectors with sparsity $> s$; (iii) finding the s^{th} largest magnitude element of those vectors; and (iv) picking the smallest of those values.

Theorem 1 indicates that for a particular starting point (W^0, X^0, Γ^0), the iterate sequence in our algorithm converges to an equivalence class of accumulation points. Every accumulation point has the same cost $g^* = g^*(W^0, X^0, \Gamma^0)$, [14] and is a fixed point of the algorithm, as well as a partial local minimizer (with respect to the cluster transforms and sparse code

[14] The exact value of g^* may vary with initialization. We will empirically illustrate in Section 2.5.2 that our algorithm is also insensitive to initialization.

variables) of the objective $g(W, X, \Gamma)$. Since Algorithm A1 minimizes the objective $g(W, X, \Gamma)$ by alternating between the minimization over (X, Γ) and W, and obtains the global optimum in each of these minimizations, it follows that the algorithm's fixed points satisfy the partial global optimality conditions (2.10) and (2.11).

Thus, we can also say that, for each initial (W^0, X^0, Γ^0), the iterate sequence in OCTOBOS converges to an equivalence class of fixed points, or an equivalence class of partial local/global minimizers satisfying (2.10), (2.11), and (2.12). This is summarized by the following Corollary [63, 62].

Corollary 1. *For a particular initial (W^0, X^0, Γ^0), the iterate sequence in Algorithm A1 converges to an equivalence class of fixed points that are also partial minimizers satisfying* (2.10)*,* (2.11)*, and* (2.12)*.*

The following corollary [63, 62] summarizes the convergence of Algorithm A1 for any starting point (the phrase "globally convergent" refers to convergence from any initialization).

Corollary 2. *The iterate sequence in Algorithm A1 is globally convergent to the set of partial minimizers of the non-convex objective $g(W, X, \Gamma)$.*

We would like to emphasize that unlike results in previous work (for synthesis learning), Theorem 1 that shows the convergence of the proposed non-convex OCTOBOS learning algorithm is free of any restrictive conditions or requirements. Theorem 1 also holds for any choice of the parameter λ_0 in (P5), which controls the condition number of the cluster transforms. The condition (2.12) in Theorem 1 holds true not only for local (or small) perturbations in the sparse codes, but also for arbitrarily large perturbations of the sparse codes in a half space, as defined by region $\mathrm{R1}_k$.

While Theorem 1 shows partial local/global optimality for Algorithm A1, the following Theorem 2 establishes that, under certain conditions, every accumulation point of the iterate sequence in Algorithm A1 is a stationary point of the overall objective. Algorithm A1 then converges to the set of stationary points of the overall problem.

Equation (2.7) indicates that the objective that is minimized in Problem

(P5) can be equivalently written as

$$f(W) = \sum_{i=1}^{N} \min_{k} \left\{ \|W_k Y_i - H_s(W_k Y_i)\|_2^2 + \lambda_0 \, Q(W_k) \, \|Y_i\|_2^2 \right\} \tag{2.13}$$

This equivalent objective is now only a function of the transforms $\{W_k\}$ (with the cluster assignment being implicit). Our OCTOBOS learning algorithm can be thought of as an alternating minimization algorithm to minimize the function $f(W)$, that also involves the optimization with respect to the additionally introduced sparse code variables.

We now state Theorem 2 [63, 62] in terms of the equivalent objective $f(W)$.

Theorem 2. *Let $\{W^t, X^t, \Gamma^t\}$ denote the iterate sequence generated by Algorithm A1 with training data Y and initial (W^0, X^0, Γ^0). Let each accumulation point (W, X, Γ) of the iterate sequence be such that (X, Γ) is the* unique *minimizer of $g\left(W, \tilde{X}, \tilde{\Gamma}\right)$ for fixed W. Then, every accumulation point of the iterate sequence is a stationary point of the objective $f(W)$.*

Theorem 2 establishes that the iterates converge to the set of stationary points of $f(W)$. It assumes that for every accumulation point (W, X, Γ), the pair (X, Γ) is the *unique* minimizer of $g\left(W, \tilde{X}, \tilde{\Gamma}\right)$ for fixed W. Note that the condition $(X, \Gamma) \in \underset{\tilde{X}, \tilde{\Gamma}}{\arg\min} \; g\left(W, \tilde{X}, \tilde{\Gamma}\right)$ is already guaranteed by Theorem 1. Only the uniqueness of the sparse coding and clustering solution is additionally assumed in Theorem 2, i.e., we assume that there are no ties in assigning the clusters or sparse codes.

Although the result in Theorem 2 depends on the uniqueness condition, the following conjecture [63, 62] postulates that provided the following Assumption 2 [63, 62] (that uses a probabilistic model for the data) holds, the uniqueness condition holds with probability 1, i.e., the probability of a tie in assigning the cluster or sparse code is zero.

Assumption 2. The training signals $Y_i \in \mathbb{R}^n$ for $1 \leq i \leq N$, are drawn independently from an absolutely continuous probability measure over the n-dimensional ball $\hat{S} \triangleq \{y \in \mathbb{R}^n : \|y\|_2 \leq c_0\}$ for some $c_0 > 0$.

Conjecture 1. *Let Assumption 2 hold. Then, with probability 1, every accumulation point (W, X, Γ) of Algorithm A1 is such that (X, Γ) is the* unique *minimizer of $g\left(W, \tilde{X}, \tilde{\Gamma}\right)$ for fixed W.*

If Conjecture 1 holds, then every accumulation point of the iterate sequence in Algorithm A1 is immediately a stationary point of $f(W)$ with probability 1.

To summarize, the convergence analysis demonstrates the following properties of Algorithm A1.

(i) The objective sequence in Algorithm A1 converges.

(ii) The sequence of iterates is bounded.

(iii) The iterate sequence has an accumulation point.

(iv) All the accumulation points of the iterate sequence have a common objective value.

(v) Every accumulation point of the iterate sequence is a fixed point of the algorithm satisfying the partial global optimality conditions (2.10) and (2.11).

(vi) Every fixed point of the algorithm is a local minimizer of $g(W, X, \Gamma)$ with respect to the transforms $\{W_k\}$ and sparse codes $\{X_i\}$.

(vii) Under the uniqueness condition stated in Theorem 2, every accumulation point is a stationary point of the equivalent objective $f(W)$.

Items (i)-(vi) above pertain to Theorem 1 and establish the various results in Theorem 1, while item (vii) pertains to Theorem 2.

2.4 Image Denoising

There are numerous applications that benefit from a good sparse model. Image denoising is an important and classical application that has been widely studied. The goal of denoising is to recover an estimate of an image $x \in \mathbb{R}^P$ (2D image represented as a vector) from its corrupted measurement $y = x+h$, where $h \in \mathbb{R}^P$ is a noise vector. Here, we consider h whose entries are i.i.d. Gaussian with zero mean and variance σ^2. We propose an adaptive image denoising framework in this section that exploits the proposed union of transforms model, or OCTOBOS model.

2.4.1 Problem Formulation

Similar to previous work on dictionary-based image denoising [3], we work with image patches. We model them as sparse in a transform domain. We allow overlapping patches, which provide an additional averaging effect that reduces noise. The patches considered can be vectorized to form the columns of a training matrix, allowing us to utilize the proposed schemes such as (P5) to learn an adaptive transform for patches.

Similar to the previous formulation [17] for adaptive square transform-based denoising, we propose the following image denoising problem formulation that exploits the union of transforms model.

$$\begin{aligned} \min_{\{W_k, x_i, \alpha_i, C_k\}} & \sum_{k=1}^{K} \sum_{i \in C_k} \left\{ \|W_k x_i - \alpha_i\|_2^2 + \lambda_i' Q(W_k) \right\} \\ & + \tau \sum_{i=1}^{N} \|R_i\, y - x_i\|_2^2 \\ s.t. \;\; & \|\alpha_i\|_0 \leq s_i \;\; \forall \; i, \;\; \{C_k\} \in G \end{aligned} \tag{P8}$$

Here, $R_i \in \mathbb{R}^{n \times P}$ is defined to be a patch extraction operator, i.e., $R_i y \in \mathbb{R}^n$ denotes the i$^{\text{th}}$ patch of the image y as a vector. We assume a total of N overlapping patches. Compared with Problem (P5), the denoising problem includes the additional, yet important data fidelity term $\tau \sum_{i=1}^{N} \|R_i\, y - x_i\|_2^2$. The assumption in (P8) is that there exist noiseless $x_i \in \mathbb{R}^n$ that approximate $R_i y$, and are approximately sparsifiable by the learned model. The weight τ for the fidelity term is typically inversely proportional to the noise level σ, that is assumed known a priori. Vector $\alpha_i \in \mathbb{R}^n$ in (P8) denotes the sparse representation of x_i in a specific cluster transform W_k, with an a priori unknown sparsity level s_i. The weight λ_i' is set based on the given noisy data $R_i y$ as $\lambda_0 \|R_i y\|_2^2$. The net weight on the $Q(W_k)$ regularizer in (P8) is then $\lambda_k = \sum_{i \in C_k} \lambda_i'$. Thus, similar to Problem (P5), the weight λ_k here varies depending on C_k.

Since $\tau \propto 1/\sigma$, we have the result that when $\sigma = 0$, the optimal $x_i = R_i\, y$ in (P8). In that case, (P8) reduces to the transform learning problem (P5). Since the patch-based framework is used in formulation (P8), the denoised image x is obtained by averaging the learned x_i's at their respective locations in the image [17].

Problem (P8) has the disadvantage that it involves a priori unknown sparsity levels s_i. These sparsity levels have to be estimated in practice. An alternative version of (P8) would replace the penalty $\tau \sum_{i=1}^{N} \|R_i\, y - x_i\|_2^2$ by constraints $\|R_i\, y - x_i\|_2^2 \leq nC^2\sigma^2\ \forall\ i$, where C is a constant. Furthermore, the sparsity constraints in (P8) can be converted to a penalty of the form $\sum_{i=1}^{N} \gamma_i \|\alpha_i\|_0$. Although this modification eliminates the issue of unknown sparsity levels s_i, it introduces another new set of unknown parameters $\gamma_i > 0$. [15] In this chapter, we will work with the formulation (P8) that uses the (simple) data fidelity penalty and sparsity constraints. Our algorithm for (P8) will additionally estimate the (minimum) sparsity levels for which the condition $\|R_i\, y - x_i\|_2^2 \leq nC^2\sigma^2\ \forall\ i$ is satisfied (similar to [3]).

2.4.2 Algorithm for (P8)

The proposed iterative algorithm is aimed at solving the non-convex Problem (P8). While one could solve (P8) with fixed s_i (e.g., s_i set to 10% of the patch size), we observed that the denoising performance is better when the s_i's are tuned adaptively as discussed above. Each iteration of our algorithm involves intra-cluster transform learning, variable sparsity level update, and clustering steps. The denoised patches x_i are updated in the final iteration. The denoised image is reconstructed when the iterations complete.

Intra-Cluster Transform Learning

Given $\{x_i\}$, $\{s_i\}$, and the clusters C_k, we solve for the cluster transforms $\{W_k\}$ and the corresponding sparse codes $\{\alpha_i\}$ in (P8). This problem separates out into K different single transform learning problems (similar to (P3)). The k$^{\text{th}}$ problem is as follows:

$$\min_{\{W_k, \alpha_i\}} \sum_{i \in C_k} \left\{ \|W_k x_i - \alpha_i\|_2^2 + \lambda_i' Q(W_k) \right\}$$
$$s.t.\ \ \|\alpha_i\|_0 \leq s_i\ \ \forall\ i \in C_k \tag{2.14}$$

This problem is solved by alternating between sparse coding and transform update steps. For each of these steps, we use closed-form solutions [1].

[15] The γ_i's need to be set accurately for the modified formulation to work well in practice.

Algorithm for (P8)

Input : y - noisy image, s - initial fixed sparsity, K - number of clusters, L - number of iterations of algorithm for (P8), σ^2 - an estimate of noise variance, J - number of transform learning iterations, λ_0 - a constant.

Output : x - Denoised image estimate

Initialization : Patches $x_i = R_i y$ and $s_i = s$ for $i = 1, 2, ..., N$. Initial $W_k = W_0 \, \forall \, k$, C_k - random cluster selection for each $i \in \{1, ..., N\}$.

For l = 1:L Repeat

1. For $k = 1...K$, update W_k and the corresponding α_i alternatingly (to solve problem (2.14)), with fixed clusters C_k and $x_i = R_i y$. The number of alternations for each k is J.

2. Update s_i for all $i = 1, 2, ..., N$: Increase s_i in $\alpha_i = H_{s_i}(W_k R_i y)$ in (2.15) where $i \in C_k$, until the error condition $\|R_i \, y - x_i\|_2^2 \leq nC^2\sigma^2$ is reached.

3. For each $i \in \{1, ..., N\}$, perform clustering and sparse coding with $x_i = R_i y$: calculate $\tilde{\alpha}_i^k = H_{s_i}(W_k x_i) \; \forall \, k$ and include i in the cluster $C_{\hat{k}}$ if $\left\| W_{\hat{k}} x_i - \tilde{\alpha}_i^{\hat{k}} \right\|_2^2 + \eta_{\hat{k}} \left\| x_i \right\|_2^2 \leq \left\| W_j x_i - \tilde{\alpha}_i^j \right\|_2^2 + \eta_j \left\| x_i \right\|_2^2 \; \forall \, j \neq \hat{k}$, with the $\eta_j = \lambda_0 \, Q(W_j)$. The optimal code $\alpha_i = \tilde{\alpha}_i^{\hat{k}}$.

End

Update x : Obtain the denoised patches x_i satisfying the error condition in step 2 above, and average them at their respective image locations.

Figure 2.2: Algorithm to solve (P8), and obtain a denoised image estimate x. A particular initialization is mentioned above for $\{W_k, C_k, s_i\}$, for simplicity. The W_0 above can be chosen to be the DCT, wavelets, etc.

Intra-Cluster Sparsity Level Update

Now, we update the sparsity levels s_i for all i. We adopt a similar method for updating the sparsity levels as introduced by [17].

With a fixed cluster transform W_k and α_i ($i \in C_k$), one can solve for x_i in (P8) in the least squares sense as follows:

$$x_i = \begin{bmatrix} \sqrt{\tau}\, I \\ W_k \end{bmatrix}^{\dagger} \begin{bmatrix} \sqrt{\tau} R_i y \\ \alpha_i \end{bmatrix} = G_1 R_i y + G_2 \alpha_i \tag{2.15}$$

where I is the $n \times n$ identity, and the matrices G_1 and G_2 are given as $G_1 = \tau \left(\tau I + W_k^T W_k\right)^{-1}$ and $G_2 = \left(\tau I + W_k^T W_k\right)^{-1} W_k^T$. Both G_1 and G_2 are computed once for each cluster.

With α_i held at $H_{s_i}(W_k R_i y)$, we choose s_i to be the smallest integer such that the x_i in (2.15) satisfies the error condition $\|R_i\, y - x_i\|_2^2 \leq nC^2\sigma^2$. This can be done efficiently by pre-computing $m_i = G_1 R_i y$ and adding to it one scaled column of G_2 at a time (corresponding to incrementing s_i by 1 in $\alpha_i = H_{s_i}(W_k R_i y)$), until the error condition is met.

We only update the s_i's in this step, except in the final algorithm iteration, when the x_i's computed above satisfying the $\|R_i\, y - x_i\|_2^2 \leq nC^2\sigma^2$ condition represent the final denoised patches. Note that the sparse code is also further updated here for each $i \in C_k$ as $\alpha_i = H_{s_i}(W_k R_i y)$, using the optimal s_i.

Clustering

With fixed $\{s_i\}$, $\{W_k\}$, and $\{x_i\}$, we solve (P8) with respect to the clusters $\{C_k\}$ and sparse codes $\{\alpha_i\}$. This problem is similar to (P6). For each i, we solve a sparse coding and clustering problem in the union of transforms model. We calculate $\tilde{\alpha}_i^k = H_{s_i}(W_k R_i y)\ \forall\, k$, and choose the cluster $C_{\hat{k}}$ if we have that $\left\|W_{\hat{k}} x_i - \tilde{\alpha}_i^{\hat{k}}\right\|_2^2 + \eta_{\hat{k}} \left\|x_i\right\|_2^2 \leq \left\|W_j x_i - \tilde{\alpha}_i^j\right\|_2^2 + \eta_j \left\|x_i\right\|_2^2\ \forall\, j \neq \hat{k}$, where $\eta_j = \lambda_0\, Q(W_j)$. Then, the optimal $\alpha_i = \tilde{\alpha}_i^{\hat{k}}$. Note that the clustering step is not performed in the final iteration of our algorithm for (P8).

Computing Denoised Image Estimate

The denoised image patches $\{x_i\}$ obtained from the iterative scheme for (P8) are restricted to their range (e.g., 0-255 for unsigned 8-bit integer class). We output the denoised image by averaging the denoised patches at their respective image locations. The summary of the method for (P8) is presented in Fig. 2.2.

In order to enhance the method's efficiency, we typically perform the intra-cluster transform learning in our algorithm using only a subset of patches that are selected uniformly at random in each cluster. The patches are all mean-subtracted in our algorithm, and the means are added back to the final denoised patch estimates.

Our algorithm learns a union of transforms using noisy patches, and updates the sparsity levels s_i adaptively during the iterations. One could use the final s_i's output from the algorithm and re-solve (P8) with fixed s_i's by alternating between the intra-cluster transform learning, x_i update (by least squares), and clustering steps. However, we observed in our experiments that such additional iterations produce at most a minor additional improvement in denoising performance. Hence, to save run time, we do not include them.

Note that similar to previous work [17], we do not enforce the constraint $R_i x = x_i$ explicitly in (P8), but rather treat extracted patches as individual data points. Although the final denoised image estimate is computed by averaging all patches (at respective locations), the approach may be sub-optimal [4], but results in an effective algorithm with low computational cost. Numerical results presented in Section 2.5 demonstrate this.

2.4.3 Improved Denoising by Iteratively Resolving (P8)

Our aforementioned algorithm obtains a denoised image estimate by solving (P8) once. We propose an improved iterative denoising scheme that makes multiple passes through (P8), each time replacing y by its latest denoised version, setting the noise level to an estimate of the remaining noise in the denoised image produced in the previous pass. Each iteration of this denoising scheme uses the same algorithm (for (P8)) as in Section 2.4.2.

2.5 Numerical Experiments

2.5.1 Framework

Overview

In this section, we present results demonstrating the promise of the proposed adaptive union of transforms, or OCTABOS framework, in image representation and image denoising. First of all, we illustrate the convergence behavior of our alternating transform learning algorithm. We consider the behavior of our algorithm with various initializations to empirically check whether the algorithm is sensitive to initializations. We will also provide an example showing the clustering/classification ability of our approach. Then, we illustrate the promise of the proposed transform learning approach for representing various images. We compare the image representation quality provided by our learned union of transforms (or, equivalently our learned OCTABOS transforms) to that provided by learned single square transforms and fixed analytical transforms such as the DCT. Finally, we demonstrate the potential of the proposed adaptive overcomplete transform-based image denoising scheme. We will show that the proposed approach performs better than methods involving learned single square transforms and learned overcomplete synthesis dictionaries (K-SVD). The computational advantages of the proposed approach over the synthesis dictionary-based approach will also be demonstrated.

The data in our experiments are generated as the patches of natural images. We employ our proposed transform learning Problem (P5) for learning adaptive sparse representations of such patches. The means (DC values) of the patches are removed and we only sparsify the mean-subtracted patches which are stacked as columns of the training matrix Y (patches reshaped as vectors). The means are added back for image display. Mean removal is typically adopted in image processing applications such as image denoising [17].

All our implementations were coded in Matlab version R2013b. The Matlab implementation of K-SVD denoising [3] available from Michael Elad's website [73] was used in our comparisons. For K-SVD denoising, we used the built-in parameter settings of the author's implementation. All com-

putations were performed with an Intel Core i7 CPU at 2.9GHz and 4GB memory, employing a 64-bit Windows 7 operating system.

Quality/Performance Metrics

We have previously introduced several metrics to evaluate the quality of learned transforms [12, 17]. The *normalized sparsification error* (NSE) for a single transform W is defined as $\|WY - X\|_F^2 / \|WY\|_F^2$, where Y is the data matrix, and the columns $X_i = H_s(WY_i)$ of the matrix X denote the sparse codes [12]. The NSE measures the fraction of energy lost in sparse fitting in the transform domain, and is an interesting property to observe for the adaptive transforms. For our proposed approach, since we have a union of square transforms and clustered patches, we compute the normalized sparsification error as follows:

$$\text{NSE} = \frac{\sum_{k=1}^{K} \sum_{i \in C_k} \|W_k Y_i - X_i\|_2^2}{\sum_{k=1}^{K} \sum_{i \in C_k} \|W_k Y_i\|_2^2} \tag{2.16}$$

Here, the numerator is the net sparsification error (i.e., the total transform domain residual), and the denominator denotes the total transform domain energy. The numerator is always $\leq$ the denominator. The sparse codes in the k^{th} cluster above are $X_i = H_s(W_k Y_i)$. For the proposed NSE definition to be meaningful, we assume that the W_k's are all normalized (e.g., they have unit spectral norm). When $K = 1$, the above definition is identical to the previously proposed NSE [12] for a single transform.

For image representation, a useful performance metric is the recovery peak signal to noise ratio (or *recovery PSNR* (RP)), which for the case of a single transform W was previously defined as $255\sqrt{P} / \|Y - W^{-1}X\|_F$ in decibels (dB), where P is the number of image pixels and X is again the transform sparse code of data Y [12]. The recovery PSNR measures the error in recovering the patches Y (or equivalently the image, in the case of non-overlapping patches) as $W^{-1}X$ from their sparse codes X obtained by projecting WY onto the ℓ_0 ball. The recovery PSNR serves as a simple surrogate for the performance of a learned transform in the compression application. For our proposed union of transforms approach, the recovery PSNR is redefined in

terms of the clusters as follows:

$$\mathrm{RP} = \frac{255\sqrt{P}}{\sqrt{\sum_{k=1}^{K} \sum_{i \in C_k} \left\| Y_i - W_k^{-1} X_i \right\|_2^2}} \tag{2.17}$$

Note that each patch Y_i belongs to exactly one cluster C_k above.

For image denoising, similar to previous work [3], we measure the image reconstruction PSNR computed between the true noiseless reference and the noisy or denoised images.

2.5.2 Convergence and Learning Behavior

Here, we illustrate the convergence behavior of our alternating OCTOBOS learning algorithm for image data. We extract the $\sqrt{n} \times \sqrt{n} = 8 \times 8$ ($n = 64$) non-overlapping mean-subtracted patches from the 512×512 Barbara image (shown later in Fig. 2.8). The data matrix $Y \in \mathbb{R}^{n \times N}$ in this case has 4096 vectorized training patches. We learn a union of transforms (or, equivalently an OCTOBOS transform) for this data by solving (P5). The parameters are set as $\lambda_0 = 3.1 \times 10^{-3}$, $s = 11$ (which is roughly $1/6^{\text{th}}$ of the data dimension), and the number of clusters $K = 2$. The choice of λ_0 here ensures well-conditioning of the blocks of the learned overcomplete transform. Badly conditioned transforms degrade performance in applications [12]. Hence, we focus our investigations here only on the well-conditioned scenario.

In the experiments, we initialize the learning algorithm for (P5) with the clusters $\{C_k\}$ and cluster transforms $\{W_k\}$. The sparse codes X_i in (P5) are then computed for the initialization, and the alternating algorithm is executed beginning with the transform update step.

Here, we study the convergence behavior of the algorithm for various initializations. We consider two different scenarios. In Scenario A, we fix the initial clusters $\{C_k\}$ (each patch is assigned uniformly at random to one of $K = 2$ clusters), and vary the initialization for the cluster transforms $\{W_k\}$. Four different initializations for the $\{W_k\}$ are considered: (i) the 64×64 2D DCT matrix (obtained as the Kronecker product of two 8×8 1D DCT matrices); (ii) the Karhunen-Loève Transform (KLT) initialization, obtained by inverting (transposing) the left singular matrices of the data in each cluster; (iii) the identity matrix; and (iv) a random matrix with i.i.d. Gaussian

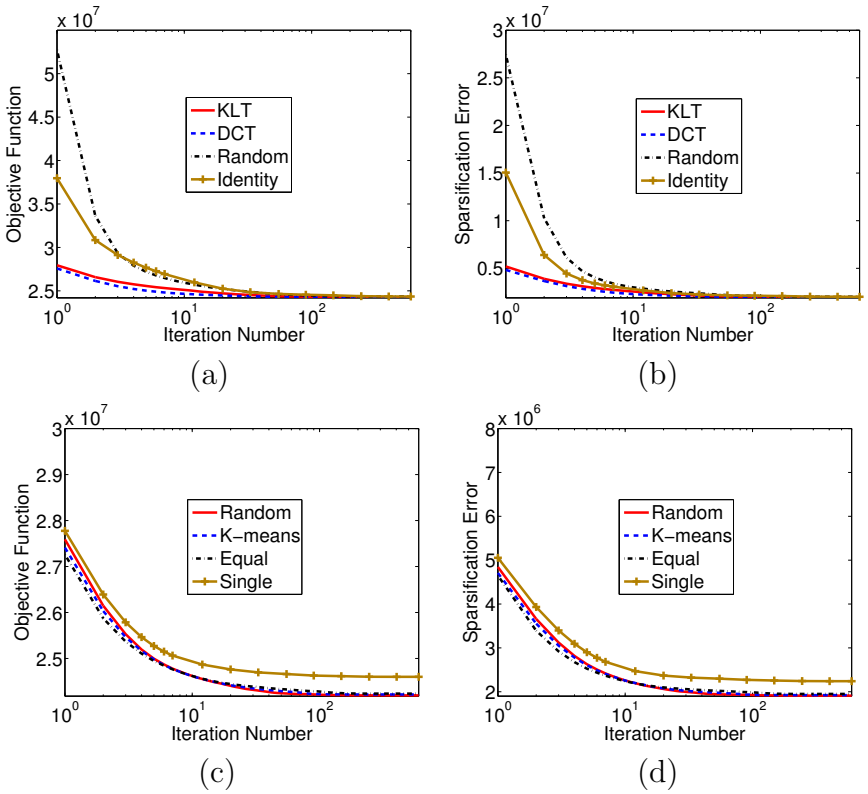


Figure 2.3: Behavior of the OCTOBOS Learning Algorithm for (P5): (a) Objective function with different transform initializations; (b) Sparsification error with different transform initializations; (c) Objective function with different cluster initializations for $K = 2$, along with the objective for single square transform ($K = 1$) learning; (d) Sparsification error with different cluster initializations for $K = 2$, along with the sparsification error for single square transform ($K = 1$) learning.

entries (zero mean and standard deviation 0.2), respectively. [16] In Scenario B, we fix the initial cluster transforms $\{W_k\}$ to be the 2D DCT, and vary the initialization for the $\{C_k\}$ in (P5). We consider three initializations for the clusters here: (i) the initialization obtained by using the well-known k-means algorithm; (ii) random clustering, where each patch is assigned uniformly at random (a different random clustering is used here, than the one in the aforementioned Scenario A) to one of the clusters; and (iii) the patches on the left half of the image in one cluster, and the remaining patches in the second cluster. We will refer to the initialization (iii) as 'equal' initialization, for

[16] Note that for the case of the DCT, identity, and random initializations, the same matrix is used to initialize all the W_k's.

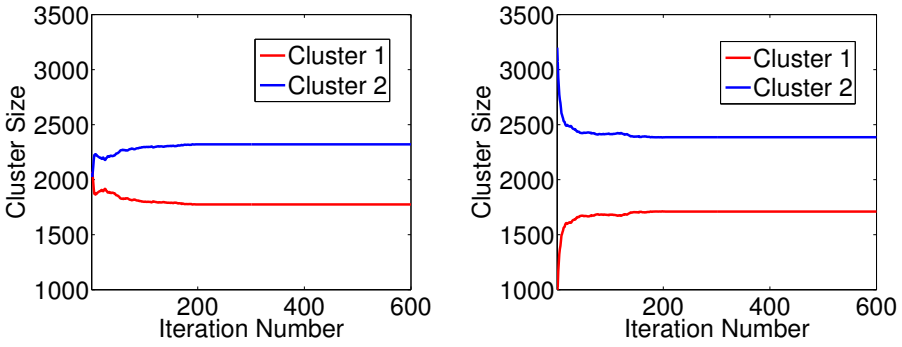


Figure 2.4: Cluster size convergence for (P5) corresponding to Fig. 2.3(c): (a) Case of random cluster initialization; (b) Case of 'equal' cluster initialization. Note that the cluster sizes change dramatically in the first iteration in (b).

simplicity.

Figure 2.3 shows the progress of the algorithm over iterations for the various initializations of $\{W_k\}$ (Scenario A in Figs. 2.3(a) and 2.3(b)), and $\{C_k\}$ (Scenario B in Figs. 2.3(c) and 2.3(d)). Both the objective function (Figs. 2.3(a) and 2.3(c)) and sparsification error (Figs. 2.3(b) and 2.3(d)) converge quickly for our algorithm. Importantly, the final values of the objective (similarly, the sparsification error) are nearly identical for all the initializations. This indicates that our learning algorithm is reasonably robust, or insensitive to initialization. Good initializations for the $\{W_k\}$ such as the DCT and KLT lead to faster convergence of learning (Figs. 2.3(a) and 2.3(b)).

For comparison, we also plot in Figs. 2.3(c) and 2.3(d), the behavior of the algorithm for $K = 1$. In this case it reduces to the single square transform learning algorithm via (P3) [12, 1]. The parameters such as s and λ_0 are set to the same values as for $K = 2$. Fig. 2.3(c) shows the objective for single square transform learning converging to a larger value compared to OCTOBOS learning. Likewise, the sparsification error for OCTOBOS for $K = 2$ (Fig. 2.3(d)) is 0.67 dB better than that provided by the learned single square transform. This confirms our expectation based on Proposition 3 in Section 2.1.

The learned square blocks of the overcomplete transform here have similar condition numbers (≈ 1.4) and Frobenius norms (≈ 5) for all initializations. This confirms that an appropriate choice of λ_0 allows the learned W_k's to

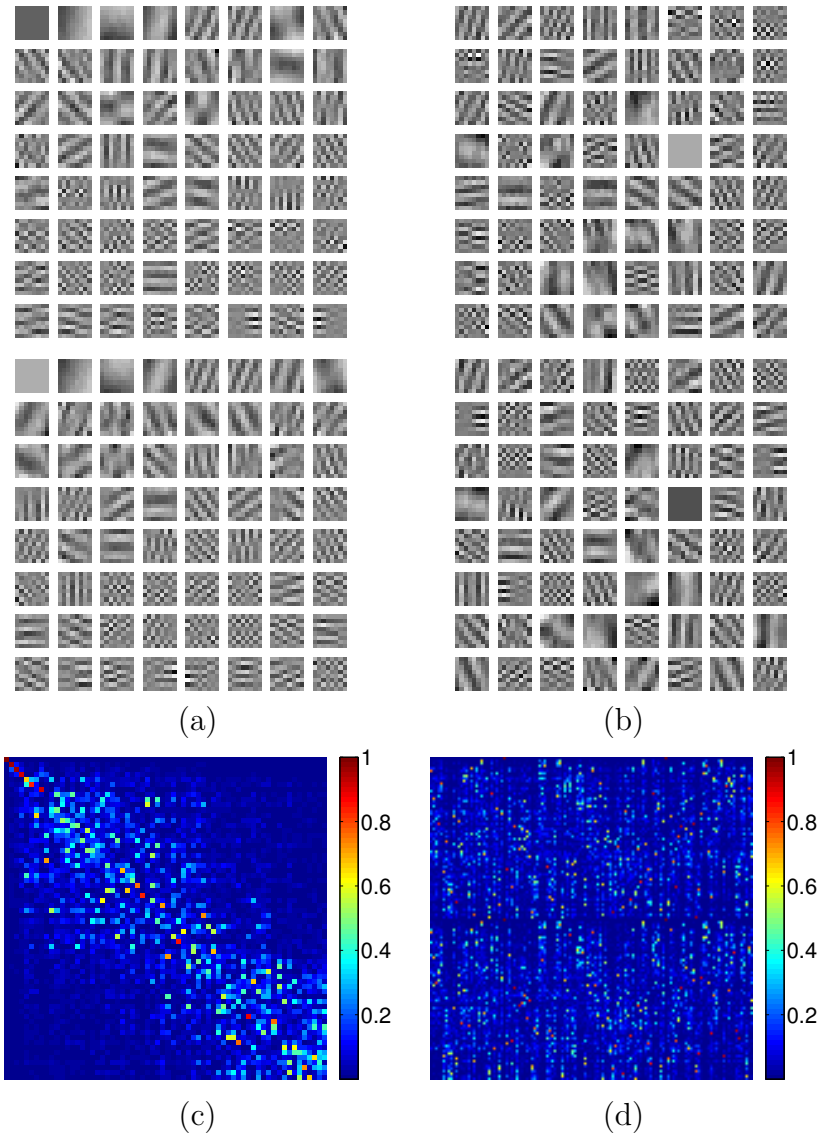


Figure 2.5: Learned OCTOBOS transforms corresponding to Fig. 2.3(a): Rows of the learned overcomplete transform W shown as patches (the two square blocks of W are separated by a white space) for the case of (a) KLT initialization, and (b) random matrix initialization; Magnitude of the cross-gram matrix computed: (c) between the two learned (row-normalized) square blocks in (a); and (d) between the two (row-normalized) overcomplete transforms in (a) and (b).

be similarly scaled, and ensures that the sparsification error term in (P5) is fully meaningful.

Next, we plot in Fig. 2.4 the cluster sizes over iterations for two different initializations of $\{C_k\}$ – random (Fig. 2.4(a)) and 'equal' (Fig. 2.4(b)).

The final values of $|C_1|$ and $|C_2|$ for the two initializations are similar. We observed that the (learned) clusters themselves can be similar (although, not necessarily identical) for various initializations.

Figure 2.5 visualizes the transforms learned by our alternating algorithm with the KLT and with random initializations for $\{W_k\}$ (the aforementioned Scenario A). The rows, or atoms of the learned overcomplete transforms are shown as patches. The learned transforms exhibit geometric and texture-like features, achieved by adaptation to the patches of the Barbara image. In order to gauge the similarity, or difference between the learned OCTOBOS transforms with different initializations, we show in Figure 2.5(d), the magnitude of the cross-gram matrix [17] computed between the transforms in Figs. 2.5(a) and 2.5(b). We normalize the rows of the transforms prior to computing their cross-gram matrix. The cross-gram matrix then indicates the coherence between every pair of rows from the two overcomplete transforms. For the 128×128 cross-gram matrix in this case, there are only 15 entries with amplitude above 0.9. This indicates that the two learned OCTOBOS transforms are not similar, i.e., they are not related by just row permutations and sign changes. However, interestingly, both still sparsify the data Y equally well. Therefore, as far as sparsification is concerned, the two different overcomplete transforms can be considered essentially equivalent [12].

How similar are the square blocks of the same overcomplete transform? In Fig. 2.5(c), we show the magnitude of the 64×64 cross-gram matrix computed between the (row normalized) blocks in Fig. 2.5(a). In this case, there are only 5 entries with amplitude above 0.9, indicating that the two learned square blocks are quite different. This is not surprising, since the two blocks here correspond to disjoint clusters.

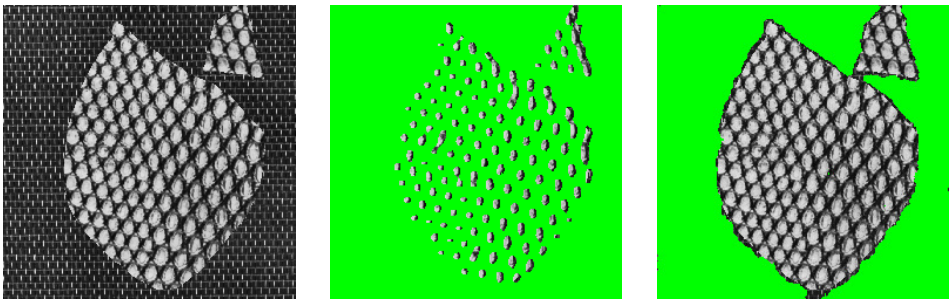
Although we considered the image Barbara in our convergence study here, we observed similar behavior for our algorithm for other images as well.

2.5.3 Clustering Behavior

In this subsection, we briefly illustrate the clustering behavior of our OCTOBOS learning scheme [62, 74]. First, we consider the 251×249 input image shown in Fig. 2.6(a). The image was formed by combining two textures [18]

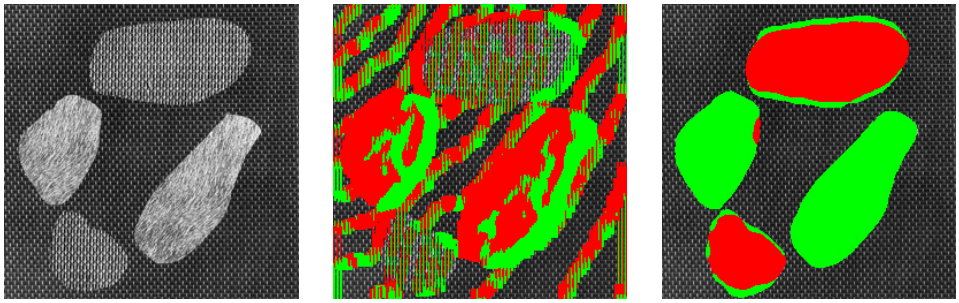
[17] For two matrices A and B of same size, the cross-gram matrix is computed as AB^T.

[18] Textures downloaded from `http://www.ux.uis.no/~tranden/brodatz.html`



(a) (b) (c)

Figure 2.6: $K = 2$ clustering example: (a) Input image. (b) Input image with pixels clustered into Class 1 shown in Green for the K-means initialization, and (c) OCTOBOS.



(a) (b) (c)

Figure 2.7: $K = 3$ clustering example: (a) Input image, (b) Input image with pixels clustered into Classes 1 and 2 shown in Green and Red, respectively, for the K-means initialization, (c) Input image with pixels clustered into Classes 1 and 2 shown in Green and Red, respectively, for OCTOBOS.

from the Brodatz database [75]. The goal is to cluster the pixels of the image into one of two classes. In order to do so, we adopt the following strategy. We consider all overlapping mean-subtracted patches from the input image, and employ formulation (P5) to learn an adaptive clustering of the patches. Since overlapping patches are used, each pixel in the image typically belongs to many overlapping patches. We cluster a pixel into a particular class C_k if the majority of the patches to which it belongs are clustered into that class by (P5).

We use 9×9 (overlapping mean-subtracted) patches ($n = 81$), and set $s = 10$, $K = 2$, and λ_0 is set as in Section 2.5.2. We initialize OCTOBOS learning using the clustering result of the k-means algorithm. The two cluster transforms are initialized with the DCT. We now use the aforementioned

strategy to cluster the pixels of the input two-texture image into one of two classes using OCTOBOS. Fig. 2.6(c) shows the clustering result obtained using OCTOBOS. As a comparison, Fig. 2.6(b) shows the image pixels clustered into each class for the k-means initialization. The proposed scheme is seen to improve over the k-means result. Alternative initializations for the OCTOBOS clusters such as random initialization also provide similar final clustering results, but typically require more iterations to converge.

Figure 2.7 shows clustering results for a 256×256 three texture image (Fig. 2.7(a)). The parameters for OCTOBOS are $K = 3$, and n, s, λ_0 are set just as for the case of Fig. 2.6. OCTOBOS (Fig. 2.7(c)) is again seen to improve over the k-means initialization (Fig. 2.7(b)).

The clustering examples here illustrate some *preliminary* potential for the OCTOBOS scheme in classification. We also observed reasonable clustering results with other texture images. Note that unlike prior work in synthesis dictionary-based classification (e.g., [76]), we do not have additional penalties in (P5) that discriminate (e.g., by enforcing incoherence) between the learned transform blocks. An extension of our OCTOBOS scheme by incorporating such classification-specific penalties (and other classification-specific heuristics) may be useful for the classification application. We leave the detailed investigation of the classification application (for example, the study of potential discriminative OCTOBOS learning methods) for future work.

2.5.4 Sparse Representation of Images

In this section, we study the potential of the proposed OCTOBOS learning scheme for sparsely representing various images. We consider the six images shown in Fig. 2.8. The images Cameraman and Peppers have 256×256 pixels. The image Man has 768×768 pixels, and Couple, Barbara, and Lena have 512×512 pixels. We learn overcomplete transforms for each of the images by solving (P5). We consider 8×8 non-overlapping mean-subtracted patches, and set λ_0, s to the same values as in Section 2.5.2. Different levels of overcompleteness (K) are considered. We initialize the OCTOBOS learning with random clustering (each patch assigned uniformly at random to one of K clusters) and the 2D DCT transform for the W_k's. For comparison, we also learn a square transform (i.e., the $K = 1$ case) for the images, which is

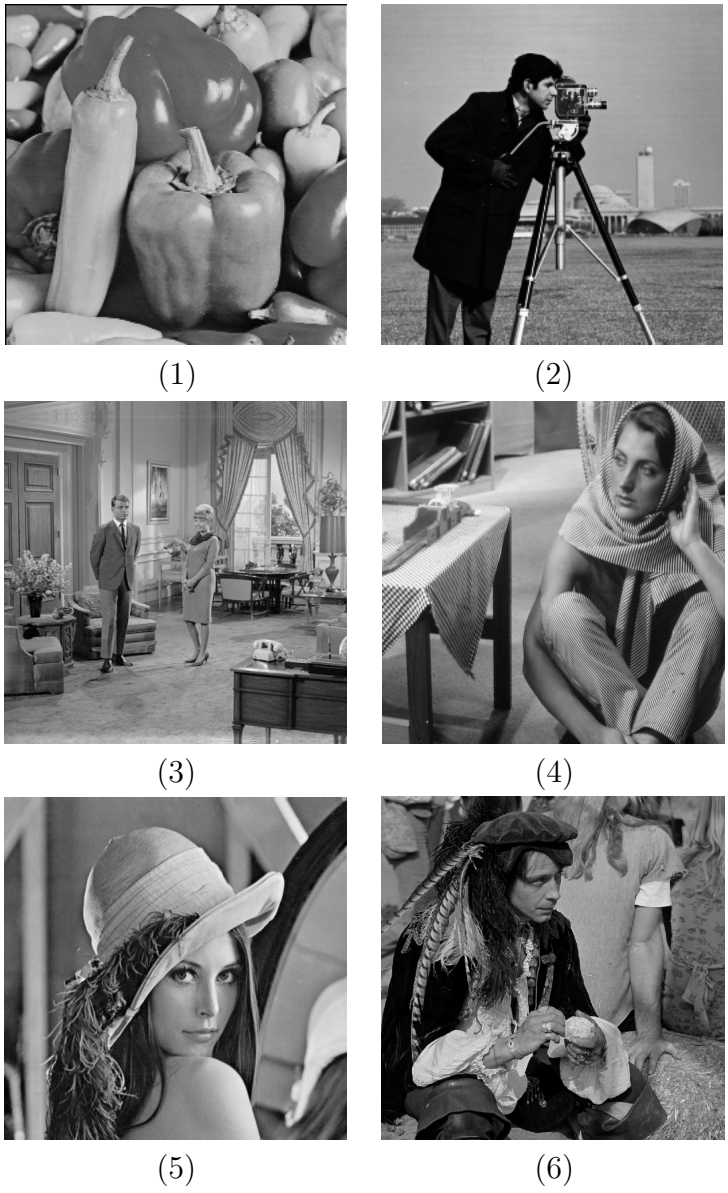


Figure 2.8: Images used for sparse representation and denoising experiments: (1) Peppers, (2) Cameraman, (3) Couple, (4) Barbara, (5) Lena, (6) Man. These images are numbered 1 through 6 in our results.

equivalent to solving (P3) [1]. All the transform learning schemes are run for 100 iterations.

Tables 2.2 and 2.3 list the NSE and recovery PSNR metrics for the various learned OCTOBOS transforms, along with the corresponding values for the learned (single) square transforms, the analytical transform of 2D DCT, and KLT. The learned transforms (both OCTOBOS and square) provide

Table 2.2: NSE metric (in percentage) for the learned OCTOBOS transforms with different K, and for the learned single square ($K = 1$) transforms (SQ) [1], and for DCT, and KLT. The best NSE values are marked in bold. The last row of the table provides average values computed over the six images.

Image	DCT	KLT	SQ	OCTOBOS			
				$K = 2$	$K = 4$	$K = 8$	$K = 16$
1	4.5	4.9	2.4	2.0	1.3	0.6	**0.2**
2	9.0	9.7	4.7	3.5	1.9	0.7	**0.1**
3	6.6	7.3	5.0	4.3	3.8	3.0	**2.3**
4	6.8	8.9	4.3	3.5	3.0	2.6	**1.8**
5	4.7	4.8	3.2	2.8	2.4	2.0	**1.4**
6	9.1	9.5	6.4	5.7	5.2	4.6	**3.8**
Av.	6.8	7.5	4.3	3.6	2.9	2.2	**1.6**

significantly better sparsification and recovery compared to the DCT and KLT. Importantly, as K increases, the learned OCTOBOS transforms provide increasingly better image representation compared to the learned square transform. The recovery PSNR increases monotonically, and NSE decreases likewise, as K increases. This clearly indicates that different groups/clusters of patches in the images are better sparsified by different (rather than identical) adaptive transforms. Another interesting observation is regarding the amount of improvement in NSE and recovery PSNR that OCTOBOS provides compared to the learned square transform for different images. Images that have more diverse features, or more patches (e.g., Barbara, Man) require a larger value of K to achieve a certain amount of improvement (for OCTOBOS vs. square) than images with less diverse features, or fewer patches (e.g., peppers, cameraman).

In order to further explore the relevance of the OCTOBOS model for the various images, we consider the following experiment with $K = 2$. For each image, we swap the two learned OCTOBOS blocks between the two learned clusters. We then recompute the NSE and recovery PSNR metrics for the images using the learned clusters, but swapped transforms. The results are shown in Table 2.4. We observe that the metrics have significantly worsened (compared to the result without swapping) with the swapping of the transforms. This now clearly indicates that the two learned clusters (or, equivalently, the two learned transforms) are quite different from each other

Table 2.3: Recovery PSNR for the learned OCTOBOS transforms with different K, and for the learned single square ($K = 1$) transforms (SQ) [1], and for DCT, and KLT. The best PSNR values are marked in bold. The last row of the table provides average PSNR values computed over the six images.

Image	DCT	KLT	SQ	OCTOBOS			
				$K = 2$	$K = 4$	$K = 8$	$K = 16$
1	33.2	32.8	35.2	36.0	38.0	41.1	**45.9**
2	30.0	29.7	32.0	33.3	35.9	40.9	**47.8**
3	34.0	33.5	34.5	35.0	35.6	36.5	**37.9**
4	32.9	31.7	34.5	35.4	36.0	36.6	**38.1**
5	36.9	36.8	37.6	38.2	38.8	39.8	**41.4**
6	32.5	32.3	33.1	33.5	33.9	34.5	**35.3**
Average	33.2	32.8	34.5	35.2	36.4	38.2	**41.1**

for the images. In fact, with swapping, the NSE and recovery PSNR results are worse than those obtained with a single learned square transform, since in the latter case, the transform is at least learned over all the image patches. Note that the learned OCTOBOS blocks all have similar and good condition numbers in Table 2.4, as expected.

2.5.5 Image Denoising

We present preliminary results for our adaptive OCTOBOS-based image denoising framework (based on (P8)). We work with the six images shown in Fig. 2.8, and simulate i.i.d. Gaussian noise at 5 different noise levels ($\sigma = 5$, 10, 15, 20, 100) for each of the images. We compare the denoising results obtained by our proposed algorithm in Section 4.3, with those obtained by the adaptive overcomplete K-SVD denoising scheme [3], the GMM-based denoising method [4], and the BM3D method [2], which is a state-of-the-art image denoising method. Note that as opposed to the K-SVD scheme, our OCTOBOS method is quite constrained due to the block cosparsity of the sparse code.

We work with 8×8 ($n = 64$) overlapping image patches in our experiments. For OCTOBOS-based denoising, we consider a 256×64 transform, i.e., $K = 4$. A corresponding 64×256 synthesis dictionary is used in the synthesis K-SVD denoising method. We fixed the initial sparsity levels s_i to 6 for all

Table 2.4: Swapping Experiment: The learned OCTOBOS blocks for the $K = 2$ case are swapped between the two learned clusters, and the NSE (in percentage) and recovery PSNR metrics are recomputed for various images. For comparison, we also include the corresponding metrics for the $K = 2$ (no swapping) and learned square transform ($K = 1$) cases. The subscripts a, b, and c, are used to denote the following scenarios:
a : Swapping Experiment result, b : OCTOBOS (K = 2) result, and
c : Square (K = 1) result. The condition numbers of the two learned OCTOBOS blocks ($\kappa(W_1)$ and $\kappa(W_2)$) are also provided for all images.

Image	1	2	3	4	5	6
RP_a	**32.3**	**29.2**	**33.3**	**32.3**	**36.4**	**32.2**
RP_b	36.0	33.3	35.0	35.4	38.2	33.5
RP_c	35.2	32.0	34.5	34.5	37.6	33.1
NSE_a	**4.8**	**9.9**	**6.6**	**7.3**	**4.3**	**8.1**
NSE_b	2.0	3.5	4.3	3.5	2.8	5.7
NSE_c	2.4	4.7	5.0	4.3	3.2	6.4
$\kappa(W_1)$	1.44	1.59	1.58	1.48	1.51	1.67
$\kappa(W_2)$	1.32	1.57	1.29	1.35	1.33	1.38

patches in our algorithm for (P8). We chose $C = 1.08$, and $\lambda_0 = 3.1 \times 10^{-2}$. We perform multiple passes through (P8), as discussed in Section 2.4.3. For each noise level (σ) of the original noisy image, the number of times that (P8) is solved, and the corresponding noise levels (for each pass through (P8)) were determined empirically. [19] These same parameters were used for all the images in our experiments. Other parameters in our algorithm such as the number of iterations (L, J in Fig. 2.2) for (P8) were set empirically. An example of OCTOBOS denoising is shown in Fig. 2.9.

Table 2.5 lists the PSNRs obtained by denoising with OCTOBOS, overcomplete K-SVD, GMM, and BM3D. First, the OCTOBOS scheme clearly provides better PSNRs than K-SVD for all images and noise levels. Comparing the PSNR values obtained by the 256×64 OCTOBOS to those of the 64×256 synthesis K-SVD dictionary for each image and noise level, we obtain an average PSNR improvement (average computed over all images and noise levels) of 0.30 dB for OCTOBOS over K-SVD. The improvement

[19] The noise level estimates decrease over the iterations (passes through (P8)). We also found empirically that underestimating the noise standard deviation (during each pass through (P8)) led to better performance.

over K-SVD for individual examples is up to 0.66 dB in Table 2.5. Thus, the OCTOBOS method outperforms K-SVD despite using a constrained (block cosparse) transform. We also obtain an average speedup of 2.8x for OCTOBOS denoising over K-SVD denoising. [20] This is because the various steps of OCTOBOS-based denoising such as the sparse coding and clustering step are computationally very cheap.

Our OCTOBOS denoising scheme is also 0.05 dB better on an average (over all images and noise levels) compared to GMM-based denoising in Table 2.5. Although the state-of-the-art BM3D method is quite better than OCTOBOS at $\sigma = 100$, OCTOBOS denoising is only 0.22 dB worse than BM3D on the average at other noise levels ($\sigma \leq 20$ in Table 2.5). OCTOBOS denoising also performs comparably to BM3D (at lower noise levels) for certain images such as Cameraman and Peppers.

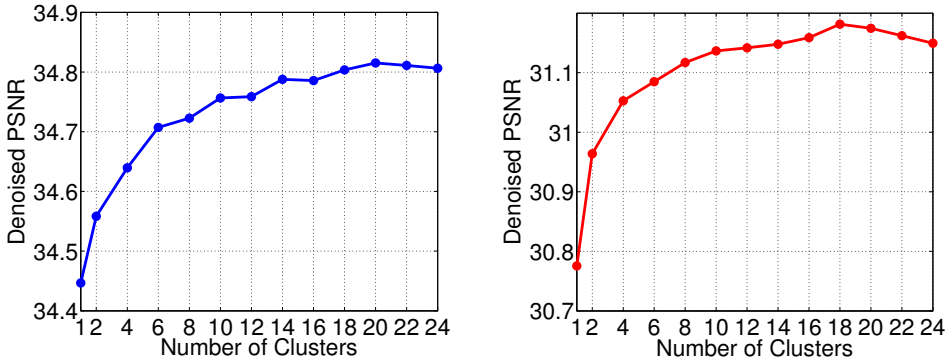
Next, using the same parameters as in the preceding experiments, we study the behavior of OCTOBOS denoising as a function of the overcompleteness K of the transform. Figures 2.10(a) and 2.10(b) plot the denoising PSNRs for Barbara as a function of the number of clusters K for $\sigma = 10$ and $\sigma = 20$, respectively. In both cases, the denoising PSNR increases with K up to an optimal value of K, beyond which the PSNR begins to slowly drop. Initially, as K increases, the OCTOBOS model becomes richer, and thus provides increasingly better denoising. However, when K becomes too large, [21], one cannot reliably learn all the OCTOBOS square blocks from the limited number of noisy training data associated with each block, without overfitting the noise. Thus, the PSNR begins to drop for very large K. This effect is more pronounced the higher the noise level, as seen in Fig. 2.10, where the optimal K where the plot peaks is lower for $\sigma = 20$, than for $\sigma = 10$. The same trend continues at $\sigma = 100$ (not shown in Fig. 2.10). The plots in Fig. 2.10 also illustrate the advantage (up to 0.4 dB improvement for this example) of OCTOBOS-based denoising over the single square transform-based ($K = 1$) denoising. This gap increases when the OCTOBOS parameters are better tuned for larger K.

[20] Our MATLAB implementation of OCTOBOS denoising is not currently optimized for efficiency. Therefore, the speedup here is computed by comparing our unoptimized MATLAB implementation to the corresponding MATLAB implementation [73] of K-SVD denoising.

[21] Compare this behavior to the monotone increase with K of the recovery PSNR for image representation (see Section 2.5.4).

(a) (b)

Figure 2.9: Denoising result: (a) Noisy Cameraman (PSNR = 22.10 dB), (b) Denoised Cameraman (PSNR = 30.24 dB) obtained using the OCTOBOS scheme.



(a) (b)

Figure 2.10: Denoising PSNR for Barbara as a function of the number of clusters K: (a) $\sigma = 10$, (b) $\sigma = 20$.

Thus, our results for OCTOBOS-based denoising are quite comparable to, or better than, the results obtained by previous image denoising schemes such as GMM denoising, BM3D, K-SVD denoising, and adaptive square transform denoising. The learned OCTOBOS (square) blocks for all images and noise levels in our experiments are well-conditioned (condition numbers of $1 - 2$). We expect the denoising PSNRs for OCTOBOS to improve further with optimal parameter tuning. Our method is limited at very high noise (such as $\sigma = 100$) due to the fact that the learning is done using corrupted data. Therefore, in the high noise setting, using a fixed OCTOBOS transform (learned over a database of images that share similar properties to the image

being denoised) may provide better denoising. This topic is worthy of further future investigation. Moreover, since the state-of-the-art BM3D is a non-local method, we believe that a non-local extension to the OCTOBOS scheme could lead to even better OCTOBOS denoising performance. We plan to investigate such an extension in the near future.

Table 2.5: PSNR values for denoising with 256×64 OCTOBOS transform, along with the corresponding values for denoising using BM3D [2], the 64×256 overcomplete K-SVD [3], and the GMM method [4].

Image	σ	Noisy PSNR	BM3D	K-SVD	GMM	OCTOBOS
Peppers	5	34.14	38.09	37.78	37.95	38.09
	10	28.10	34.66	34.24	34.51	34.57
	15	24.58	32.69	32.18	32.54	32.43
	20	22.12	31.33	30.80	31.18	30.97
	100	8.11	23.17	21.79	22.97	22.23
Cameraman	5	34.12	38.21	37.81	38.06	38.19
	10	28.14	34.15	33.72	34.00	34.15
	15	24.61	31.91	31.50	31.85	31.94
	20	22.10	30.37	29.82	30.21	30.24
	100	8.14	23.15	21.76	22.89	22.24
Couple	5	34.16	37.48	37.28	37.35	37.40
	10	28.11	34.01	33.51	33.79	33.73
	15	24.59	32.08	31.46	31.84	31.71
	20	22.11	30.78	30.02	30.51	30.34
	100	8.13	23.46	22.57	23.30	22.88
Barbara	5	34.15	38.30	38.08	37.59	38.31
	10	28.14	34.97	34.41	33.61	34.64
	15	24.59	33.05	32.33	31.28	32.53
	20	22.13	31.74	30.83	29.74	31.05
	100	8.11	23.61	21.87	22.13	22.41
Lena	5	34.16	38.70	38.61	38.55	38.71
	10	28.12	35.88	35.49	35.56	35.64
	15	24.63	34.26	33.74	33.87	33.92
	20	22.11	33.01	32.41	32.60	32.59
	100	8.14	25.75	24.51	25.24	25.17
Man	5	34.15	36.76	36.47	36.75	36.73
	10	28.13	33.18	32.71	33.14	32.98
	15	24.63	31.32	30.78	31.32	31.07
	20	22.11	30.03	29.40	30.02	29.74
	100	8.14	23.83	22.76	23.65	22.92

CHAPTER 3

ONLINE SPARSIFYING TRANSFORM LEARNING

In this chapter, we investigate the problem of online sparsifying transform learning [1]. Prior work on transform learning focused on batch learning [12, 1], where the sparsifying transform is adapted using all the training data simultaneously. However, for big data, the dimensions of the training data set are typically very large. Hence, batch learning of a sparsifying transform using existing alternating algorithms [1] is computationally expensive in both time and memory, and maybe even infeasible. Moreover, in real-time applications, the data arrives sequentially, and must also be processed sequentially to limit latency. Thus, this setting renders batch learning infeasible, since in real-time applications, one does not have access to all the data at once. To address these problems, we introduce in this work a scheme for online, or sequential, learning of a square sparsifying transform.

The proposed learning framework adapts sequentially the sparsifying transform and sparse codes (and/or signal estimates) for signals (or, measurements) that arrive, or are processed sequentially. Such online/sequential transform learning is amenable to big data, and to applications such as real-time sparse representation (compression), denoising, and compressed sensing. As we show in this work, online transform learning involves cheap computations and modest memory requirements. Our numerical experiments illustrate the usefulness of our schemes for big data processing (online sparse representation, and denoising). As we show, the sequential transform learning scheme can also converge faster than the batch transform learning scheme [12, 1, 72] in practice.

While the online learning of synthesis dictionaries has been studied previously [40, 39, 77, 78, 79], the online adaptation of the transform model allows for much cheaper computations. Furthermore, the proof by Mairal et al. [40] of the convergence of online synthesis dictionary learning requires various

[1]The material of this chapter has previously appeared in [85, 86]

restrictive assumptions. In contrast, recent works provide convergence guarantees for online transform learning [80, 63], which relies on only a few simple assumptions. Another feature distinguishing our formulation is that in the previous work [40], the objective is biconvex, so that the non-convexity in the problem vanishes when a particular variable is kept fixed. This is not the case in our formulation, in which the non-convexity is due to the ℓ_0 "norm" and the log determinant terms. Our formulation remains non-convex even when one of the variables is fixed.

Other very recent works consider synthesis dictionary learning for big data. Wang et al. [81] propose a scheme to incrementally add new columns to the learned dictionary for every new block of signals (sequentially) processed. However, the dictionary size in this method grows continuously (as more blocks of signals are processed), which is undesirable. Another recent work on synthesis dictionary learning for big data is a split and merge learning algorithm [82], which, however, is not an online algorithm. The big dataset is split into subsets, and dictionaries are learned in parallel for each subset, before being merged to a single smaller dictionary. However, as the size of the big dataset increases, either the size of each subset increases monotonically, or the final merging step becomes more complex, requiring increasing time and memory. Although faster than conventional dictionary learning, the dictionary learned by this method [82] is worse.

We organize the rest of this chapter as follows. Section 3.1 describes the prior work on batch transform learning, and then presents our proposed problem formulations for online and mini-batch (that handles blocks of signals sequentially) transform learning and denoising. In Section 3.2, we present efficient algorithms to solve our proposed problem formulations, and discuss our algorithms' computational, latency, and memory advantages. Section 3.3 provides experimental results demonstrating the convergence and computational properties of the proposed schemes. We also show results for sparse representation and denoising.

3.1 Online Transform Learning Formulations

3.1.1 Batch Learning

In batch learning, the sparsifying transform is adapted to all the training data simultaneously. Given a matrix $Y \in \mathbb{R}^{n \times N}$, whose columns y_i $(1 \leq i \leq N)$ represent all the training signals, the problem of learning an adaptive square sparsifying transform W (in batch mode) is formulated as follows [12, 1]:

$$\text{(P9)} \quad \min_{W,X} \|WY - X\|_F^2 + \lambda v(W) \quad s.t. \quad \|x_i\|_0 \leq s \ \ \forall \ i$$

where x_i denotes the i^{th} column of the sparse code matrix X, s is a given sparsity level, and $v(W) = -\log |\det W| + \|W\|_F^2$. The term $\|WY - X\|_F^2$ in (P9) is the *sparsification error* for the data Y in the transform W. The sparsification error is the modeling error in the transform model, and hence we minimize it in order to learn the best possible transform model.

Problem (P9) also has $v(W)$ as a regularizer in the objective to prevent trivial solutions [12]. Specifically, the log determinant penalty enforces full rank on the transform W, and eliminates degenerate solutions such as those with zero, or repeated, rows. The $\|W\|_F^2$ penalty helps remove a 'scale ambiguity' [12] in the solution (the scale ambiguity occurs when the data admits an exactly sparse representation). Together, the log determinant and Frobenius norm penalty terms fully control the condition number of the learned transform [12]. This eliminates badly conditioned transforms, which typically convey little information and may degrade performance in applications. As shown in [12], the condition number of the transform $\kappa(W)$ can be upper bounded by a monotonically increasing function of $v(W)$. Hence, minimizing $v(W)$ encourages reduction of the condition number. In the limit $\lambda \to \infty$, the condition number of the optimal transform(s) in (P9) tends to 1. In practice, the transforms learned via (P9) have condition numbers close to 1 even for finite λ [1, 72]. The specific choice of λ depends on the application and desired condition number. The regularizer $v(W)$ also penalizes bad scalings. Given a transform W and a scalar $\alpha \in \mathbb{R}$, $v(\alpha W) \to \infty$ as the scaling $\alpha \to 0$ or $\alpha \to \infty$.

To make the two terms in the cost of (P9) scale similarly, we set $\lambda = \lambda_0 \|Y\|_F^2$ with constant $\lambda_0 > 0$. We have shown that the cost function in

(P9) is lower bounded [12] by $\frac{n\lambda}{2} + \frac{n\lambda}{2}\log(2) > 0$. The minimum objective value for Problem (P9) equals this lower bound if and only if there exists a pair $(\hat{W}, \hat{X})$ with $\hat{X}$ whose columns have sparsity $\leq s$ and $\hat{W}$ whose (non-zero) singular values are all equal, such that $\hat{W}Y = \hat{X}$.

Problem (P9) admits an equivalence class of solutions/minimizers. Given a particular minimizer $(\tilde{W}, \tilde{X})$, we can form equivalent minimizers by simultaneously permuting the rows of $\tilde{W}$ and $\tilde{X}$, or by pre-multiplying them by a diagonal ± 1 sign matrix [12]. More generally, because the objective in (P9) is unitarily invariant, then given a minimizer $(\tilde{W}, \tilde{X})$, the pair $(\Xi\tilde{W}, \Xi\tilde{X})$ is another equivalent minimizer for all sparsity-preserving *orthonormal* matrices Ξ, i.e., orthonormal Ξ satisfying $\|\Xi\tilde{x}_i\|_0 \leq s \ \forall \ i$.

3.1.2 Online Learning

We now introduce our problem formulation for online sparsifying transform learning. The goal here is to adapt the transform and sparse code to data that arrive, or are processed sequentially. For time $t = 1, 2, 3, ...$, the optimization problem to update the sparsifying transform and sparse code based on new data $y_t \in \mathbb{R}^n$ is as follows:

$$\text{(P10)} \ \left\{\hat{W}_t, \hat{x}_t\right\} = \underset{W,\, x_t}{\arg\min} \frac{1}{t}\sum_{j=1}^{t}\left\{\|Wy_j - x_j\|_2^2 + \lambda_j v(W)\right\}$$
$$s.t. \ \ \|x_t\|_0 \leq s, \ x_j = \hat{x}_j, \ 1 \leq j \leq t-1$$

where $\lambda_j = \lambda_0 \|y_j\|_2^2 \ \forall \, j$, $\hat{W}_t$ is the optimal transform at time t, and $\hat{x}_t$ is the optimal sparse code for y_t. Note that only the latest sparse code is updated at time t. The condition $x_j = \hat{x}_j, \ 1 \leq j \leq t-1$, is therefore assumed. For brevity, we will not explicitly restate this condition (or, its appropriate variant) in the formulations in the rest of this chapter. On the other hand, at each time t the transform $\hat{W}_t$ is optimized using all the data $\{y_j\}_{j=1}^t$ and sparse codes $\{x_j\}_{j=1}^t$ up to time t. Problem (P10) is simply an online version of the batch problem (P9), and hence it shares some properties with (P9). Specifically, the constant λ_0 controls the condition number of the learned transform.

Although Problem (P10) outputs an optimal $\hat{W}_t$ for each t, it is typically impractical to store (in memory) $\hat{W}_t$ for all t. In our experiments, we store

only the latest $\hat{W}_t$, and use it as an initialization for the algorithm that solves for $\hat{W}_{t+1}$. At any time instant t, one can obtain a least squares estimate of the signals $\{y_j\}_{j=1}^{t}$ from their sparse codes as $\left\{\hat{W}_t^{-1}\hat{x}_j\right\}_{j=1}^{t}$ (i.e., 'decompressing' the signals from stored sparse codes).

For small values of t, Problem (P10) may highly overfit the transform to the data. This is typically undesirable. In order to overcome this problem, for small values of t, we only perform an update of the sparse codes (with a fixed W – set to a reasonable initialization).

Problem (P10) can be further modified, or improved in certain scenarios. For example, for non-stationary data, it may not be possible to fit a single transform W to y_t for all t. In this case, one can introduce a forgetting factor ρ^{t-j} (with a constant $0 < \rho < 1$), that scales the terms in (P10). Such a forgetting factor would diminish the influence of "old" data. The objective function (within the minimization) in (P10) is then modified as

$$\frac{1}{t}\sum_{j=1}^{t}\rho^{t-j}\left\{\|Wy_j - x_j\|_2^2 + \lambda_j v(W)\right\} \tag{3.1}$$

Note that this is only one form of the forgetting factor (cf. [40] for another form).

For fixed size data sets, Problem (P10) can be used as an effective sequential learning and sparse coding (compression) strategy. In this case, it is typically useful to make multiple passes through the data set to overcome the causality restriction on the update of the sparse codes. In this case, the same training signals are used, or examined multiple times by (P10), which crucially allows to better update the sparse code using a transform determined by the entire data. Similar strategies have been proposed for online synthesis dictionary learning [40].

3.1.3 Mini-batch Learning

A useful variation of online learning is mini-batch learning [40], where we process more than one signal at a time. Mini-batch learning may provide potential reduction in operation count over online learning. However, the processing of blocks of signals leads to increased latency, and memory requirements.

Assuming a fixed block size (or, mini-batch size) of M, the J^{th} ($J \geq 1$) block of signals (in terms of time sequence $\{y_t\}$) is $Y_J = \Big[y_{JM-M+1} \mid ... \mid y_{JM} \Big]$. For $J = 1, 2, 3, ...$, the mini-batch sparsifying transform learning problem is formulated as follows:

$$\Big\{ \hat{W}_J, \hat{X}_J \Big\} = \underset{W,\, X_J}{\arg\min} \frac{1}{JM} \sum_{j=1}^{J} \left\{ \|WY_j - X_j\|_F^2 + \Lambda_j v(W) \right\}$$
$$s.t. \;\; \|x_{JM-M+i}\|_0 \leq s \;\; \forall\, i \in \{1, .., M\} \;\; \text{(P11)}$$

where the weight $\Lambda_j = \lambda_0 \|Y_j\|_F^2$, and the matrix $X_J = \Big[x_{JM-M+1} \mid ... \mid x_{JM} \Big]$ contains the block of sparse codes corresponding to the block Y_J.

Note that both Problems (P10) and (P11) handle signals sequentially, or involve sequential learning. However, (P10) handles one signal at a time, whereas (P11) uses blocks of signals at a time. In order to clearly distinguish between these two cases in the rest of this chapter, we will use the terminology 'online learning' to refer to only the case where one signal is processed at a time instant, and we use 'mini-batch learning' to explicitly refer to the case $M > 1$.

3.1.4 Online Denoising Formulation

Online (and mini-batch) transform learning could be used for various applications such as sparse representation (compression), denoising, compressed sensing, etc. Here, we consider an extension of (P10) and (P11) (which by themselves, can be used for sparse representation of signals) to denoising. Denoising aims to recover an estimate of the signal $z \in \mathbb{R}^n$ from its measurement $y = z + h$, corrupted by noise h. Here, we consider a time sequence of measurements $\{y_t\}$, with $y_t = z_t + h_t$, and $h_t \in \mathbb{R}^n$ being the noise. We assume h_t whose entries are independent identically distributed (i.i.d.) Gaussian with zero mean and variance σ^2. The goal of online denoising is to recover estimates of $z_t \; \forall \; t$. We model the underlying noiseless signals z_t as approximately sparse in a (unknown) transform domain.

Previous work [17, 12] presented a formulation for adaptive sparsifying transform-based batch denoising. Here, we instead present a simple denoising formulation that is a modification of the online learning Problem (P10). For

$t = 1, 2, 3, ...,$ we solve

$$\text{(P12)} \quad \min_{W, x_t} \frac{1}{t} \sum_{j=1}^{t} \left\{ \|Wy_j - x_j\|_2^2 + \lambda_j v(W) + \tau_j^2 \|x_j\|_0 \right\}$$

where the weights $\tau_j \propto \sigma$. Problem (P12) is to estimate $\hat{W}_t$, and $\hat{x}_t$, and the denoised signal is computed simply as $\hat{z}_t = \hat{W}_t^{-1} \hat{x}_t$. Similar to the extension of (P10) to (P11), we can also extend (P12) to its mini-batch version. The different variations of (P10) suggested in Section 3.1.2 (such as forgetting factor, and multiple passes) can also be applied here.

Problem (P12) can also be used for patch-based denoising of large images [3, 17], or image sequences. The overlapping patches of the noisy images are processed sequentially, and the denoised image is obtained by averaging the denoised patches at their respective image locations.

3.2 Algorithms and Properties

3.2.1 Batch Transform Learning

Previous work [12, 1, 72] proposed an alternating algorithm for solving Problem (P9) that alternates between solving for X (*sparse coding step*) and W (*transform update step*), with the other variable kept fixed. The sparse coding step is as follows:

$$\min_{X} \|WY - X\|_F^2 \quad s.t. \quad \|x_i\|_0 \leq s \ \forall \ i \tag{3.2}$$

The above problem is to project each column of WY onto the (non-convex) set of vectors with sparsity $\leq s$. An optimal solution [12, 17] to (3.2) is computed exactly as $\hat{x}_i = H_s(Wy_i) \ \forall \, i$, where the operator $H_s(\cdot)$ zeros out all but the s coefficients of largest magnitude in a vector. If there is more than one choice for the s coefficients of largest magnitude in a vector z, which can occur when multiple entries in z have identical magnitude, then we choose $H_s(z)$ as the projection of z for which the indices of the s largest magnitude elements in z are the lowest possible.

The transform update step of (P9) involves the following unconstrained

non-convex minimization:

$$\min_{W} \ \|WY - X\|_F^2 + \lambda v(W) \tag{3.3}$$

The solution to (3.3) is also computed in closed-form [1, 72]. Let us factorize $YY^T + \lambda I$ (where I is the identity matrix) as LL^T (e.g., the Cholesky factor, or the positive definite square root [2] L), with $L \in \mathbb{R}^{n \times n}$. Further, let $L^{-1}YX^T = Q\Sigma R^T$ be a full singular value decomposition (SVD), where Q, Σ, and R are $n \times n$ matrices. Then, a global minimizer of (3.3) is given as

$$\hat{W} = 0.5R \left(\Sigma + \left(\Sigma^2 + 2\lambda I \right)^{\frac{1}{2}} \right) Q^T L^{-1} \tag{3.4}$$

where the $(\cdot)^{\frac{1}{2}}$ in (3.4) denotes the positive definite square root. The solution above is unique if and only if YX^T is non-singular [72]. Furthermore, the solution is invariant to the choice of factor L.

We now discuss the computational cost and memory requirements of this batch transform learning algorithm. The total cost per iteration (of sparse coding and transform update) of the batch transform learning algorithm scales (assuming $n \ll N$) as $O(Nn^2)$. This is much lower than the per-iteration cost of learning an $n \times K$ overcomplete ($K > n$) synthesis dictionary D using K-SVD [13], which scales (assuming that the synthesis sparsity level $s \propto n$, and $K \propto n$) as $O(Nn^3)$. Previous work [12, 1, 72] has demonstrated that batch transform learning also converges quickly (in a small number of iterations) in practice.[3] The memory requirement of batch transform, or dictionary learning scales as $O(Nn)$. This cost becomes prohibitive for large N.

3.2.2 Online Transform Learning

Here, we solve Problem (P10) at each time instant t by alternating minimization (similar to (P9)).

[2] This is nothing but the well-known eigenvalue decomposition square root.

[3] Moreover, it is guaranteed to converge to (at least) a local minimum of the objective [72].

Sparse Coding

In the sparse coding step, we solve (P10) for x_t with fixed $W = \hat{W}_{t-1}$ (warm start) as follows:

$$\min_{x_t} \|Wy_t - x_t\|_2^2 \;\; s.t. \;\; \|x_t\|_0 \leq s \tag{3.5}$$

The sparse coding solution is given as $\hat{x}_t = H_s(Wy_t)$, with $H_s(\cdot)$ defined as in Section 3.2.1.

Exact Transform Update

In the transform update step, we solve (P10) with fixed $x_j = \hat{x}_j,\ 1 \leq j \leq t$, as follows:

$$\min_{W} \frac{1}{t} \sum_{j=1}^{t} \left\{ \|Wy_j - x_j\|_2^2 + \lambda_j v(W) \right\} \tag{3.6}$$

This problem has a closed-form solution (similar to (3.4)). Let $t^{-1} \sum_{j=1}^{t} \left(y_j y_j^T + \lambda_j I \right) = L_t L_t^T$ (e.g., the positive definite or eigenvalue decomposition (EVD) square root L_t). We compute the full SVD of $L_t^{-1} \Theta_t = Q_t \Sigma_t R_t^T$, where

$$\Theta_t = t^{-1} \sum_{j=1}^{t} y_j \hat{x}_j^T \tag{3.7}$$

Then, a closed-form solution [4] to (3.6) is given as

$$\hat{W}_t = 0.5 R_t \left(\Sigma_t + \left(\Sigma_t^2 + 2\beta_t I \right)^{\frac{1}{2}} \right) Q_t^T L_t^{-1} \tag{3.8}$$

We can compute $\Gamma_t \triangleq t^{-1} \sum_{j=1}^{t} y_j y_j^T$, Θ_t, and $\beta_t \triangleq \sum_{j=1}^{t} t^{-1} \lambda_j$ sequentially over time. However, computing the inverse square root L_t^{-1}, the matrix-matrix product $L_t^{-1} \Theta_t$, and its full SVD would all cost $O(n^3)$ computations. Instead, we propose a computationally cheaper transform update algorithm as follows.

Efficient Approximate Transform Update

The following algorithm involves efficient SVD computations, and eliminates matrix-matrix multiplications. To compute the transform update solu-

[4] The solution (3.8) is unique if and only if Θ_t has full rank.

tion, we first efficiently factorize $t^{-1}\sum_{j=1}^{t}\left(y_j y_j^T + \lambda_j I\right)$ as $L_t L_t^T$ (i.e., take square root), with $L_t \in \mathbb{R}^{n\times n}$. Here, we work with the eigenvalue decomposition (EVD) square root. Denoting the full EVD of $\Gamma_{t-1} = (t-1)^{-1}\sum_{j=1}^{t-1} y_j y_j^T$ as $U_{t-1}\Delta_{t-1}U_{t-1}^T$, the full EVD of $\Gamma_t = (1-t^{-1})\Gamma_{t-1} + t^{-1}y_t y_t^T$ can be computed as $(U_t, \Delta_t, U_t) = \mathcal{U}\left[U_{t-1}, (1-t^{-1})\Delta_{t-1}, U_{t-1}, A_t\right]$, where $A_t = t^{-1}y_t y_t^T$. Here, $\mathcal{U}(U, \Sigma, V, A)$ is the SVD rank-1 update operation that produces the full SVD (U', Σ', V') of matrix $M + A$, where M has full SVD (U, Σ, V), and A is the rank-1 term [83, 84]. Furthermore, let $\beta_{t-1} = \sum_{j=1}^{t-1}(t-1)^{-1}\lambda_j$. Then, $\beta_t = (1-t^{-1})\beta_{t-1} + t^{-1}\lambda_t$. The EVD square root of $t^{-1}\sum_{j=1}^{t}\left(y_j y_j^T + \lambda_j I\right)$ is then computed as $L_t = U_t\left(\Delta_t + \beta_t I\right)^{\frac{1}{2}}U_t^T$ and its inverse is $L_t^{-1} = U_t\left(\Delta_t + \beta_t I\right)^{-\frac{1}{2}}U_t^T$.

The matrix-matrix products in the formula for L_t^{-1} are not explicitly computed. Instead, we will only need the application of L_t^{-1} to a vector, which can be performed efficiently with $O(n^2)$ computation by applying U_t^T, the diagonal matrix $\left(\Delta_t + \beta_t I\right)^{-\frac{1}{2}}$, and U_t in succession.

In order to compute the closed-form solution to (4.4), we need to also compute the full SVD of $t^{-1}\sum_{j=1}^{t} L_t^{-1}y_j\hat{x}_j^T$. In order to simplify this computation, we perform the following approximation:

$$L_t^{-1}\Theta_t = L_t^{-1}\left\{(1-t^{-1})\Theta_{t-1} + t^{-1}y_t\hat{x}_t^T\right\} \tag{3.9}$$

$$\approx (1-t^{-1})L_{t-1}^{-1}\Theta_{t-1} + t^{-1}L_t^{-1}y_t\hat{x}_t^T \tag{3.10}$$

With the above approximation, and the fact that $t^{-1}L_t^{-1}y_t\hat{x}_t^T$ is a rank-1 matrix, the estimate of the full SVD of $L_t^{-1}\Theta_t$ can be obtained by performing a rank-1 update as $(Q_t, \Sigma_t, R_t) = \mathcal{U}\left[Q_{t-1}, (1-t^{-1})\Sigma_{t-1}, R_{t-1}, \tilde{A}_t\right]$, where $\tilde{A}_t = t^{-1}L_t^{-1}y_t\hat{x}_t^T$.

Now, once the full SVD estimate of $L_t^{-1}\Theta_t$ is computed as $Q_t\Sigma_t R_t^T$ (compute only the matrices in this decomposition, not the products), the closed-form solution for Problem (3.6) is simply

$$\hat{W}_t = 0.5R_t\left(\Sigma_t + \left(\Sigma_t^2 + 2\beta_t I\right)^{\frac{1}{2}}\right)Q_t^T L_t^{-1} \tag{3.11}$$

Again, we do not perform any of the matrix-matrix multiplications in (3.11). Instead, we store the individual matrices, and apply them one by one on vectors, at a computational cost of $O(n^2)$.

Note that the only approximation in the above algorithm arises in (3.10).

Otherwise all rank-1 updates above can be performed up to machine precision accuracy. The net error in the approximation in (3.10) at time t (i.e., the difference between $L_t^{-1}\Theta_t$ and its SVD estimate at time t)[5] is given as $E_t = \sum_{j=2}^{t} \frac{j-1}{t}\Upsilon_j$, where $\Upsilon_j = \left(L_j^{-1} - L_{j-1}^{-1}\right)\Theta_{j-1}$. The proof of this result is in Appendix A. In the formula of E_t, the Υ_j for smaller j values gets scaled by smaller numbers (i.e., $(j-1)/t$). Furthermore, recent works [80, 85, 63] show that Υ_j decays (in norm) as C/j, for some constant C. Based on that result, it is easy to show that the approximation error E_t is bounded by C for all t.

In order to prevent undesirable error accumulations over time, one may monitor the relative error $\|E_t\|_F \,/\, \left\|L_t^{-1}\Theta_t\right\|_F$, and reset the computation as shown below. The relative error can be shown to be upper bounded (up to a scale factor [6]) by $\sum_{j=2}^{t}(\left\|L_j^{-1} - L_{j-1}^{-1}\right\|_F / \left\|L_t^{-1}\right\|_F)$. Since we only store the eigenvectors and eigenvalues of L_j^{-1} rather than L_j^{-1} itself, we further easily bound the term $\left\|L_j^{-1} - L_{j-1}^{-1}\right\|_F$ in the preceding expression by $\|U_jS_j - U_{j-1}S_{j-1}\|_F + \min\left\{\|(U_j - U_{j-1})S_j\|_F, \|(U_j - U_{j-1})S_{j-1}\|_F\right\}$, where the matrix U_j was defined previously [7] and $S_j \triangleq (\Delta_j + \beta_j I)^{-1/2}$. We thus have a simple upper bound for the relative error that is cheap to compute at $O(n^2)$ cost, and can be monitored. If it rises above a threshold ϵ, we compute the SVD of $L_t^{-1}\Theta_t$ directly, in which case any possible accumulated error is wiped out. In our experiments, we observed that L_t^{-1} converges quickly [8] (over t) for data consisting of natural signals. In such cases, the exact SVD of $L_t^{-1}\Theta_t$ can be obtained for some initial time instances, after which the approximation (3.10) is observed to work very well. In fact, we observed reasonable performance, even with keeping L_t^{-1} fixed beyond a small t.

An alternative way to perform the transform update (3.6) would be to use the stochastic gradient descent method. However, the gradient computation requires computing the inverse of a matrix (a term like W^{-T} [12]). This computation scales worse ($O(n^3)$) than the computation (see Section 3.2.2)

[5] SVD estimates computed at time j are further used in the rank-1 update at time $j+1$.

[6] The factor is $\frac{\max_{2\le j\le t}\|\Theta_j\|_2}{\sigma_n(\Theta_t)}$, where σ_n is the smallest singular value of a matrix, and the $\|\cdot\|_2$ norm denotes the spectral norm. The factor is finite for Θ_t that is full rank.

[7] Since the eigenvectors of a matrix can always be multiplied by -1 to yield equally valid alternative eigenvectors, we may flip the sign of any row of U_j so that the row is a closer match to the corresponding row of U_{j-1} in the bound.

[8] For example, when y_t are independent and identically distributed, $t^{-1}\sum_{j=1}^{t} y_j y_j^T$ converges (as $t \to \infty$) with probability 1 to a covariance matrix, and L_t^{-1} would also converge.

for the proposed method.

While one could alternate multiple times (for each t) between the sparse coding and transform update steps of (P10), we perform only a single alternation to save computations, and to prevent overfitting to the current data. Our overall algorithm for (P10) is shown in Fig. 3.1.

Handling Variations to (P10)

Our algorithm can be easily modified to accommodate the various modifications to Problem (P10) suggested in Section 3.1.2 for non-stationary data, and for fixed data sets. For example, when making multiple passes over a fixed data set of size N, the update formula (3.10) would not involve any approximation since $L_t^{-1} = L_{t-1}^{-1} = \hat{L}^{-1}$ after the first pass, where $\hat{L}$ is the square root of $N^{-1} \sum_{j=1}^{N} \left(y_j y_j^T + \lambda_j I \right)$ (the set of training data remains the same when making multiple passes over the data set), and the term $t^{-1} L_t^{-1} y_t \hat{x}_t^T$ in (3.10) is replaced by $t^{-1} L_t^{-1} y_t (\hat{x}_t - \hat{x}_t')^T$, where $\hat{x}_t'$ is the 'older' version of the sparse code of y_t, which is removed from the formula (and the objective). When the sparse codes are not themselves stored, one can adopt a similar technique as in [40], or use a forgetting factor (3.1) when making multiple passes over the data set, in order to forget the 'older' bad sparse codes.

When using the forgetting factor ρ (as in (4.1)) in online transform learning, the various operations for the transform update step of (P10) are modified as follows. We find the SVD square root L_t of $t^{-1} \sum_{j=1}^{t} \rho^{t-j} \left(y_j y_j^T + \lambda_j I \right)$, and compute the full SVD of $t^{-1} \sum_{j=1}^{t} \rho^{t-j} L_t^{-1} y_j \hat{x}_j^T$. The methodology of transform update remains the same as before, except that we work with the (modified) matrices/scalars $\Gamma_t = \rho(1 - t^{-1}) \Gamma_{t-1} + t^{-1} y_t y_t^T$, $\beta_t = \rho(1 - t^{-1}) \beta_{t-1} + t^{-1} \lambda_t$, and $\Theta_t = \rho(1 - t^{-1}) \Theta_{t-1} + t^{-1} y_t \hat{x}_t^T$, in the aforementioned steps, with $\Gamma_0 = \Theta_0 = 0$ and $\beta_0 = 0$.

[9] If transform update isn't performed for some initial t (Section 3.1.2), then all SVDs are computed exactly for the first transform update. For simplicity, the monitoring of the relative error for (3.10) is not shown in Fig. 3.1.

Online Transform Learning Algorithm A1

Input: The sequence $\{y_t\}$.

Initialize: $\hat{W}_0 = W_0$, $\Delta_0 = \Sigma_0 = 0$, $U_0 = Q_0 = R_0 = I$, $\beta_0 = 0$.

For $t = 1, 2, 3, ...$ **Repeat**

1. **Sparse Coding:** $\hat{x}_t = H_s(\hat{W}_{t-1} y_t)$.
2. Update $\beta_t = (1 - t^{-1})\beta_{t-1} + t^{-1}\lambda_0 \left\| y_t \right\|_2^2$.
3. **Transform Update:**
   (a) $(U_t, \Delta_t, U_t) \leftarrow \mathcal{U}\left[U_{t-1}, (1 - t^{-1})\Delta_{t-1}, U_{t-1}, A_t\right]$ by rank-1 update where $A_t = t^{-1} y_t y_t^T$.
   (b) $\mathrm{SVD}(L_t^{-1}) \leftarrow (U_t, (\Delta_t + \beta_t I)^{-\frac{1}{2}}, U_t^T)$.
   (c) $(Q_t, \Sigma_t, R_t) \leftarrow \mathcal{U}\left[Q_{t-1}, (1 - t^{-1})\Sigma_{t-1}, R_{t-1}, \tilde{A}_t\right]$ by rank-1 update where $\tilde{A}_t = t^{-1} L_t^{-1} y_t \hat{x}_t^T$.
   (d) Store the matrix factors for $\hat{W}_t$ in (3.11), where L_t^{-1} is defined in (b).

End

Figure 3.1: Algorithm A1 to solve (P10) by alternating minimization.[9]

Computational and Memory Costs

We now discuss the computational cost and memory requirements of the online transform learning algorithm. The computational cost of the sparse coding step is dominated [12] by the computation of the product Wy_t, and therefore scales as $O(n^2)$. In contrast, the projection operation in (3.5) requires only $O(n \log n)$ operations [12], when employing sorting. The computational cost of the transform update step is dominated by $O(n^2 \log^2 n)$ for the rank-1 SVD updates [83, 84]. Thus, the total cost per signal (or, per time instant) of our algorithm (sparse coding and transform update) scales as $O(n^2 \log^2 n)$. This is better (especially for large n) than the computational cost per signal for online learning of an $n \times K$ overcomplete synthesis dictionary D, which scales (assuming synthesis sparsity $s \propto n$, and $K \propto n$) as at least $O(n^3)$ [40]. The (local) memory requirement of our algorithm scales modestly as $O(n^2)$, since we need to store $n \times n$ matrices.

3.2.3 Mini-batch Transform Learning

Here, we solve Problem (P11), that processes blocks of signals, by (a single iteration of) alternating minimization. In the sparse coding step, we solve for X_J in (P11), with fixed W ($= \hat{W}_{J-1}$, i.e., warm start) as follows:

$$\min_{X_J} \|WY_J - X_J\|_F^2 \quad s.t. \quad \|x_{JM-M+i}\|_0 \leq s \; \forall \, i \tag{3.12}$$

The optimal solution to (3.12) is obtained as $\hat{x}_{JM-M+i} = H_s(Wy_{JM-M+i}) \; \forall \; i \in \{1, .., M\}$.

The transform update step solves (P11) with fixed $X_j = \hat{X}_j$, $1 \leq j \leq J$, as

$$\min_{W} \; \frac{1}{JM} \sum_{j=1}^{J} \left\{ \|WY_j - X_j\|_F^2 + \Lambda_j \, Q(W) \right\} \tag{3.13}$$

When the block size M is small ($M \ll n$), we can use the same approximate transform update procedure as in Section 3.2.2, but with the rank-1 updates replaced by rank-M updates. The rank-M updates can be performed as M rank-1 updates for small M. For larger M ($M \sim O(n)$, or larger), (3.13) is solved using an exact transform update procedure (similar to the one for Problem (4.4)). Figure 3.2 shows the overall algorithm for the case of large M.

We now discuss the computational cost and memory requirements of the mini-batch version of online transform learning. The computational cost of the sparse coding step scales as $O(Mn^2)$. For small M, the cost of the transform update step scales as $O(Mn^2 \log^2 n)$. For large M, since the transform update is performed as in Fig. 3.2 (i.e., matrix inverses, matrix-matrix multiplications, and SVDs are computed directly, but scalars/matrices are accumulated over time wherever possible), the cost of transform update (Steps 2 and 3 in Fig. 3.2) scales as $C_1 Mn^2 + C_2 n^3$, where C_1 and C_2 are constants. Assuming that $C_2 n < C_1 M$ (large M), the transform update cost scales as $O(Mn^2)$. Thus, the total computation per iteration (or, per block) of our mini-batch algorithm (sparse coding and transform update) scales as $O(Mn^2)$ for large M, and $O(Mn^2 \log^2 n)$ for small M. In each of these cases, the cost is better than the cost per block (of size M) for mini-batch learning of an $n \times K$ synthesis dictionary D, which scales (assuming synthesis sparsity $s \propto n$, and $K \propto n$) as $O(Mn^3)$ [40]. The memory requirement of mini-batch

Mini-batch Transform Learning Algorithm A2

Input: Sequence $\{y_t\}$ is processed in blocks of size M.

Initialize: $\hat{W}_0 = W_0$, $\tilde{\Theta}_0 = \tilde{\Gamma}_0 = 0$, $\tilde{\beta}_0 = 0$.

For $J = 1, 2, 3, ...$ **Repeat**

1. **Sparse Coding:** $\hat{x}_l = H_s(\hat{W}_{J-1}\ y_l)\ \forall\ l$ such that $JM - M + 1 \leq l \leq JM$.

2. **Prepare for Transform Update:**

$$Y_J = \left[y_{JM-M+1} \mid y_{JM-M+2} \mid \ \ ... \ \ \mid y_{JM}\right].$$

$$\hat{X}_J = \left[\hat{x}_{JM-M+1} \mid \hat{x}_{JM-M+2} \mid \ \ ... \ \ \mid \hat{x}_{JM}\right].$$

$$\tilde{\Theta}_J = (1 - J^{-1})\,\tilde{\Theta}_{J-1} + J^{-1}M^{-1}Y_J\hat{X}_J^T.$$

$$\tilde{\Gamma}_J = (1 - J^{-1})\,\tilde{\Gamma}_{J-1} + J^{-1}M^{-1}Y_JY_J^T.$$

$$\tilde{\beta}_J = (1 - J^{-1})\,\tilde{\beta}_{J-1} + J^{-1}M^{-1}\lambda_0\left\|Y_J\right\|_F^2.$$

3. **Transform Update:**

   (a) Compute the full SVD of $\tilde{\Gamma}_J + \tilde{\beta}_J I$ as $\tilde{U}_J\tilde{\Delta}_J\tilde{U}_J^T$.

   (b) $\tilde{L}_J^{-1} = \tilde{U}_J\tilde{\Delta}_J^{-\frac{1}{2}}\tilde{U}_J^T$ (inverse square root).

   (c) Compute full SVD of $\tilde{L}_J^{-1}\tilde{\Theta}_J$ as $\tilde{Q}_J\tilde{\Sigma}_J\tilde{R}_J^T$.

   (d) $\hat{W}_J = 0.5\tilde{R}_J\left(\tilde{\Sigma}_J + \left(\tilde{\Sigma}_J^2 + 2\tilde{\beta}_J I\right)^{\frac{1}{2}}\right)\tilde{Q}_J^T\tilde{L}_J^{-1}$.

End

Figure 3.2: Algorithm A2 to solve (P11) for large block size M.

transform learning scales as $O(Mn)$ for large M, and $O(n^2)$ for small M.

3.2.4 Comparison of Transform Learning Schemes

We now compare and contrast the online, mini-batch, and batch transform learning schemes in terms of their computational costs, memory requirements, and latency. We measure latency as the time duration (the inter-arrival time between two signals is taken as 1 time unit) between the arrival of the first signal, and the generation of its corresponding output (e.g., sparse code).[10]

Table 3.1 summarizes the various costs for the transform-based schemes. We show the computational cost per sample, i.e., the cost normalized by the number of samples processed. For a given number of N samples, the batch scheme typically requires several iterations, their number denoted by P, to converge to a good transform. Thus, the batch scheme has total computational cost of $O(PNn^2)$. In practice, P depends on n, N, and algorithm initialization, and is typically larger for bigger, or more complex problems. On the other hand, as shown in related works [80, 85, 63, 86], and in the experiments of Section 3.3, the online and mini-batch schemes produce good transforms for large N (total number of signals processed sequentially). Therefore, the net computational cost for processing N signals (and converging) for the online scheme is $O(Nn^2 \log^2 n)$. Thus, assuming $\log^2 n < P$ (which is typically observed in practice), the online scheme is computationally more effective (in order) than the batch scheme for big data (large N). The computational cost of processing N signals (and thus converging, in the case of large N) for the mini-batch scheme for large M (and N/M blocks) is $O(Nn^2)$, which is even lower in order (by factor $\log^2 n$) than the cost for the (one signal at a time) online scheme.

Importantly, assuming $n, M \ll N$, the online and mini-batch transform learning schemes have far lower memory requirements and latency compared to the batch scheme. The mini-batch scheme itself has higher memory and latency costs than the online scheme.

As discussed in the preceding subsections, the dictionary learning schemes [40, 13] have computational cost per sample (not shown in Table 3.1) propor-

[10] Here, for simplicity, we assume that computations can be performed instantaneously.

Table 3.1: Comparison of online learning, mini-batch learning, and batch learning in terms of their computational cost per sample, memory cost, and latency.

Properties	Online	Mini-batch		Batch
		Small M	Large M	
Computations	$O(n^2 \log^2 n)$	$O(n^2 \log^2 n)$	$O(n^2)$	$O(Pn^2)$
Memory	$O(n^2)$	$O(n^2)$	$O(nM)$	$O(nN)$
Latency	0	$M-1$	$M-1$	$N-1$

tional to n^3. For large signals (i.e., large n), the cost of dictionary learning is much larger than the cost of the transform-based methods.

3.2.5 Denoising

Problem (P12) is identical to (P10), except for the fact that it uses a sparsity penalty function, rather than constraints. Therefore, when solving (P12) at each t by alternating minimization, the sparse coding step (with fixed $W = \hat{W}_{t-1}$) is

$$\min_{x_t} \|Wy_t - x_t\|_2^2 + \tau_t^2 \|x_t\|_0 \tag{3.14}$$

A solution [72] $\hat{x}_t$ of (3.14) is $\hat{x}_t = \hat{H}_{\tau_t}(Wy_t)$, where the hard thresholding operator $\hat{H}_\tau(\cdot)$ is defined as

$$\left(\hat{H}_\tau(b)\right)_k = \begin{cases} 0 & , \ |b_k| < \tau \\ b_k & , \ |b_k| \geq \tau \end{cases} \tag{3.15}$$

where $b \in \mathbb{R}^n$, and the subscript k indexes vector entries. Therefore, the sparse coding solution is simply obtained by hard thresholding, with a threshold proportional to the noise level σ (similar to traditional techniques involving analytical transforms [87]). The transform update step of (P12) is identical to (P10). The denoised signal is computed as $\hat{W}_t^{-1}\hat{x}_t$. By (3.11) we have

$$\hat{W}_t^{-1} = \beta_t^{-1} L_t Q_t \left((\Sigma_t^2 + 2\beta_t I)^{\frac{1}{2}} - \Sigma_t \right) R_t^T \tag{3.16}$$

Assuming that the various matrices in the above decomposition are stored in memory, $\hat{W}_t^{-1}\hat{x}_t$ can be computed using matrix-vector multiplications at a cost of $O(n^2)$. The net computational cost of denoising per signal (i.e.

for sparse coding, transform update, and denoised signal computation) then scales as $O(n^2 \log^2 n)$ (same cost as for the online learning algorithm for (P10)). For mini-batch transform learning based denoising, the net computational cost of denoising a block is similar to the costs in Section 3.2.3.

3.3 Numerical Experiments

3.3.1 Framework

Online transform learning is shown to converge asymptotically, and produces a good transform [80, 63]. Here, we present numerical results illustrating the practical convergence behavior of online (and mini-batch) transform learning, as well as the usefulness of the proposed schemes for image representation and denoising. First, we consider synthetic data generated sequentially using a particular transform model, and study the ability of our online schemes to converge to a good model. Second, we study the usefulness of online/sequential transform learning for sparse representation of images. Finally, we present results for online denoising using Problem (P12). We consider the patch-based denoising of some standard (regular sized) images as well as some very large images (where batch learning was observed to be infeasible on the particular computing platform used for our experiment). The latter case is a candidate big data problem, since it involves a large number of patches, which can be potentially denoised efficiently and sequentially using online transform learning.

All transform learning implementations were coded in Matlab version R2013b. Similarly, the Matlab implementation of K-SVD denoising [3] (a popular batch synthesis dictionary-based denoising scheme) available from Michael Elad's website [73] was used in our comparisons. For K-SVD denoising, we used the built-in parameter settings of the author's implementation. All computations were performed with an Intel Core i7 CPU at 2.9 GHz and 4 GB memory, employing a 64-bit Windows 7 operating system.

We define the normalized reconstruction error as $\|Y - W^{-1}X\|_F^2 \,/\, \|Y\|_F^2$, where Y is a matrix whose columns are the data vectors, W is a transform, and X is the corresponding sparse code matrix. The normalized reconstruction error metric is used to measure the performance of learned transforms

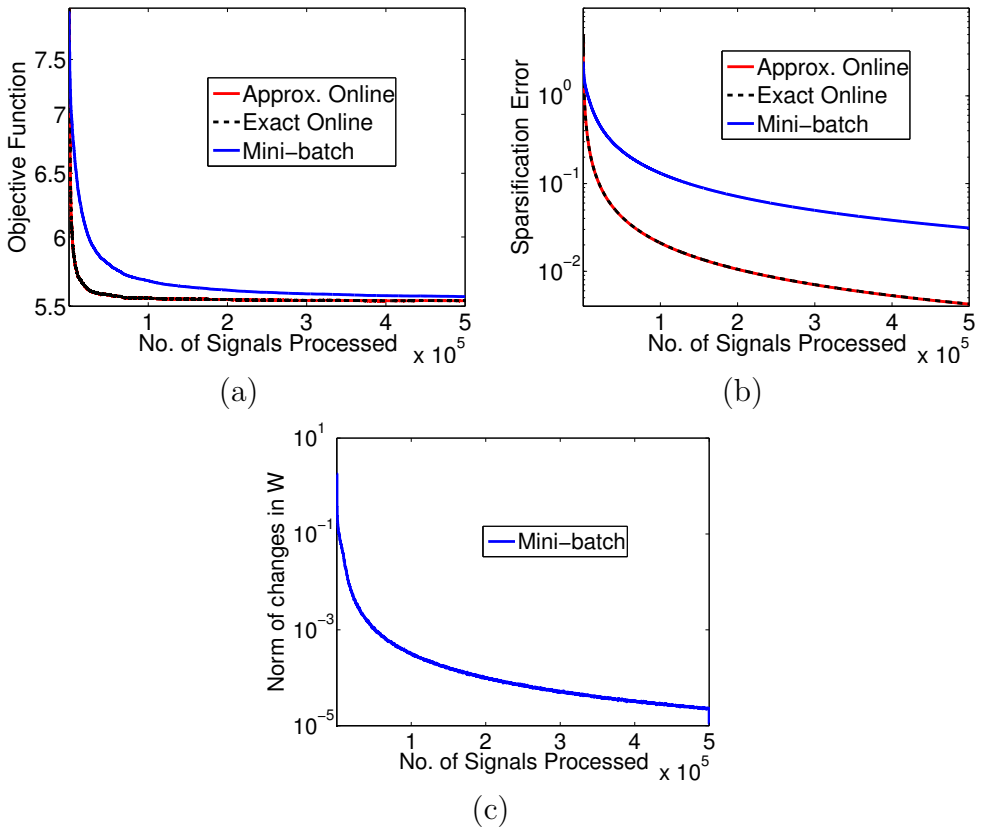


Figure 3.3: Convergence behavior of the online (both the exact and approximate versions) and mini-batch learning schemes as a function of amount of data processed: (a) Objective function, (b) Sparsification error, (c) $||\hat{W}_{t+1} - \hat{W}_t||_F$ for mini-batch scheme.

for signal/image representation. It can be thought of as a simple surrogate for the compression performance of a transform. For image denoising, similar to prior work, we measure the peak-signal-to-noise ratio (PSNR) computed between the true noiseless reference, and the noisy or denoised images.

3.3.2 Convergence and Sparse Representation

Convergence

First, we illustrate the convergence behavior of the proposed algorithms. We generate the input data y_t sequentially as $W^{-1}x_t$ using a random orthonormal 20×20 matrix W, and sparse codes x_t obtained by thresholding i.i.d. Gaus-

sian vectors at sparsity level $s = 3$. We then use our online and mini-batch transform learning algorithms for (P10) and (P11), to sequentially learn the transform and sparse codes for the data y_t. The parameter $\lambda_0 = 3.1 \times 10^{-2}$, $s = 3$, and the size of the mini-batch $M = 320$. We test (and compare) both the exact (see Section 3.2.2) and approximate (see Section 3.2.2) versions of the online transform learning algorithm. As discussed in Section 3.2.2, we monitor the upper bound on the relative approximation error. If it rises above a threshold $\epsilon = 10$, we compute the SVD of $L_t^{-1}\Theta_t$ directly in the transform update step of the algorithm. Our algorithms are initialized with the 20×20 DCT matrix.

Figures 3.3(a) and 3.3(b) show the objective function (of Problems (P10)/(P11)) and sparsification error (i.e., the objective without the regularizer) as a function of the number of signals processed, for our online and mini-batch schemes. Both the objective and sparsification error converge quickly for our schemes. The exact and approximate online schemes behave identically. Moreover, for the approximate version, we observed that the error threshold ϵ is violated (i.e., an exact SVD is performed) only 0.2% of the time. This indicates that the faster approximate online scheme works equally well as the exact version. The online schemes converge slightly faster than the mini-batch scheme as a function of the number of signals processed. This is because the transform update step is performed more frequently for the (one signal at a time) online schemes. However, the proposed approximate online transform learning scheme typically has a higher run time (due to the $\log^2 n$ factor in computations – see Section 3.2.4) than the mini-batch scheme.

As shown in Fig. 3.3(c), the difference between successive iterates, i.e., $\left\|\hat{W}_{t+1} - \hat{W}_t\right\|_F$ converges close to zero for the mini-batch schemes. A similar behavior is observed for the online schemes. The learned transforms using our exact online, approximate online, and mini-batch algorithms have condition numbers of 1.02, 1.02, and 1.04 respectively. By Fig. 3.3(b), they provide a sparsification error close to zero. The normalized reconstruction error computed using the learned $\hat{W}_t$ and the sparse codes generated sequentially is < 0.01 for our schemes, indicating that they have learned a good model for the data $\{y_t\}$.

Table 3.2: Reconstruction error improvements (dB) over patch-based 2D DCT for the batch, mini-batch, and online transform learning schemes for image data. The results for the mini-batch and online schemes are also shown with multiple passes through the dataset.

	Batch	Mini-batch		Online	
		1 pass	30 passes	1 pass	30 passes
Reconstruction Error Improv.	1.3	0.8	1.3	0.8	1.3

Sparse Representation of Images

Here, we learn a sparsifying transform for natural image patches. We extract all non-overlapping patches of size 8×8 (about 1.6×10^5 patches) from the images in the USC-SIPI database [88] (the color images are converted to gray-scale images). We use our (approximate) online and mini-batch transform learning algorithms to learn a transform and sparse codes sequentially on the (mean-subtracted) patches. The parameters are set as $\lambda_0 = 6.2 \times 10^{-2}$, $s = 11$, and $M = 256$.

Table 3.2 shows the improvements in the normalized reconstruction error achieved by the online, mini-batch and batch [12] transform learning schemes over the fixed 2D DCT for the image patches. For the online and mini-batch schemes, we also show results with multiple passes through the same data (each time a particular signal is repeated, its old sparse code is replaced with the latest one – see Section 3.2.2). Both the online and mini-batch schemes (either with or without multiple passes) provide better reconstruction quality compared to the DCT. The proposed schemes also perform similar to the batch scheme (and about 1.3 dB better than the DCT) at the end of 30 passes. Importantly, even with $30\times$ passes, the mini-batch scheme runs $3\times$ faster than the batch algorithm in achieving similar reconstruction quality. The learned transforms were all well-conditioned in this experiment.

(a) (b)

(c) (d)

Figure 3.4: The large images used in our denoising experiments: (a) Airport (1024×1024), (b) Man (1024×1024), (c) Railway (2048×2048), (d) Campus ($3264 \times 3264 \times 3$).

3.3.3 Online Image Denoising

Regular-size Image Denoising

Here, we present some results for our simple denoising framework (P12). We consider image denoising, where the overlapping image patches are processed and denoised sequentially. We work with the images Couple (512×512) and Man (768×768),[11] and simulate i.i.d. additive Gaussian noise at three different noise levels (standard deviation $\sigma = 5$, 10, 20) for the images.

We denoise the 8×8 overlapping image patches (the mean is subtracted during learning, and added back in the reconstruction step) sequentially (no cycling) using adaptive mini-batch denoising. Once the denoised patches in each mini-batch are computed, we immediately put them back at their cor-

[11] These are standard images that have been used in prior work (e.g., [3, 17]).

Table 3.3: PSNR values (dB) and run times (seconds) for denoising regular-size images at different noise levels ($\sigma = 5$, 10, 20), using batch K-SVD [3], batch transform learning [1], and mini-batch transform learning-based denoising schemes.

Images	σ	Noisy PSNR		Batch K-SVD	Batch TL	Mini-batch TL
Couple	5	34.16	PSNR	37.28	37.33	37.33
			time	1250	92	20
	10	28.11	PSNR	33.51	33.62	33.62
			time	671	68	19
	20	22.11	PSNR	30.02	30.02	30.03
			time	190	61	20
Man	5	34.15	PSNR	36.47	36.66	36.75
			time	1279	205	45
	10	28.13	PSNR	32.71	32.96	33.00
			time	701	130	44
	20	22.11	PSNR	29.40	29.57	29.52
			time	189	80	41

responding locations in the denoised image. Note that the denoised image is computed by averaging the denoised patches at their respective 2D locations. Our scheme requires minimal memory (we only store data required for computations at a particular time instant t) and mimics a real-time denoising setup.

The parameters are set to $\lambda_0 = 3.1 \times 10^{-2}$, $M = 64$, and $\tau_j = 1.73\sigma \ \forall j$. We work with a forgetting factor $\rho < 1$ (corresponding to an extension of (3.1) for the mini-batch case), which was found to provide a slight improvement. The best choice of ρ depends on the size of the mini-batch, patch size (signal size), and noise level, and was set empirically to $\rho = 0.87$, 0.95, and 0.99, for $\sigma = 5$, 10, and 20, respectively.

Our denoising results are compared to K-SVD denoising [13, 3, 73], and to batch square transform learning-based denoising [1] (with parameters set as in [17]). The images in this experiment have sizes compatible (i.e., small enough to avoid memory overflows) with the batch denoising schemes. The goal of the comparison to the batch dictionary/transform schemes is not to show the state-of-the-art performance of our method in a general denois-

ing application. Rather, we focus on the sequential aspect of our method, and aim to demonstrate that the proposed adaptive mini-batch transform-based denoising algorithm with smaller latency, memory and computational requirements, can be used as an efficient and effective alternative to adaptive batch mode dictionary/transform denoising.

Table 3.3 lists the denoising PSNRs and run times (including the time for generating the denoised image) for the various methods. The mini-batch transform denoising method provides comparable, or better denoising performance compared to the batch-based methods, while being much faster. We compute the average speedup provided by our mini-batch denoising scheme over the adaptive batch-based methods. For each image and noise level, the ratio of the run times of batch denoising and mini-batch denoising is computed, and this speedup is averaged over the images and noise levels in Table 3.3. The mini-batch transform learning-based denoising scheme provides an average speedup of 26.0× and 3.4× respectively, over the batch K-SVD and batch transform denoising schemes.

Large-Scale Image Denoising

In the context of big data, batch learning is typically infeasible due to the strict practical limits on computational and memory resources. However, we can potentially use the proposed online, or mini-batch transform learning schemes to sparse code, or denoise large images, and image sequences. Here, we present preliminary results for mini-batch denoising of large-scale images (both gray-scale and color images). We work with the gray-scale images in Fig. 3.4(a)-(c), and the color image in Fig. 3.4(d). The largest of these images has about 11 megapixels (when it is processed by a patch-based scheme, there are about 11 million overlapping patches in total). We simulate i.i.d. additive Gaussian noise at 3 different noise levels ($\sigma = 20$, 50, 100) for these images. In the case of the color image, noise is added to each of the red (R), green (G), and blue (B) color channels.

The size of the mini-batch is set to $M = 256$. The forgetting factor is set to $\rho = 0.88$, 0.92, and 0.95, for $\sigma = 20$, 50, and 100 respectively, for gray-scale images, and $\rho = 0.95$ for the color image. Other parameters in our algorithm are set as in the regular-size image denoising experiment described above. To apply the transform, we vectorize all image patches. For color

image denoising, we form a patch vector by stacking the patch's vectorized red, green, and blue components [30]. We therefore learn a larger 192×192 transform for the patches of the color image.

Table 3.4 lists the denoising PSNRs and run times (the latter are averaged over the noise levels σ for each image) obtained using the proposed simple mini-batch transform learning-based denoising algorithm, as well as those obtained using a similar scheme but involving the fixed patch-based 2D DCT (no learning). For the DCT-based algorithm, we use $\tau_j = 2.45\sigma \ \forall j$, which was found to be empirically optimal in our experiments. As evidenced from Table 3.4, the mini-batch denoising algorithm provides better PSNRs than the DCT for all images and noise levels.[12] To denoise the large-scale images, the mini-batch algorithm takes up to 23% longer for 1024×1024 images (with roughly 1 million patches) or for the 2048×2048 image (roughly 4 million patches), and about 40% longer for the 3264×3264 color image (roughly 11 million patches) respectively, than the 2D DCT method. Therefore, although there is no learning involved in the latter case, our adaptive scheme typically denoises about as fast as the fixed transform.

Figure 3.5 shows a zoom-in of the denoised Man image, obtained using the proposed mini-batch transform learning-based denoising algorithm at $\sigma = 20$. The zoom-in shows reasonable reconstruction of the image features from the noisy measurements. Thus, the results here demonstrate the potential of our scheme for limited latency (or, real-time) denoising of large-scale data.

A feasible alternative method for large-scale image denoising is to break the large image into smaller "mini-images", and perform batch denoising on each of the mini-images. To test the effectiveness of this alternative, we divided each of our large noisy images into 256×256 overlapping (an overlap of 7 pixels) mini-images, and performed batch transform denoising on the mini-images [1]. The mini-images were denoised one-by-one. This allows us to use the transform learned by the batch denoising algorithm in the current mini-image as an initialization (for the first mini-image, the initialization used is the 2D DCT) for the batch denoising scheme [1] in the next nearest mini-image. Such a warm start leads to better denoising performance (within each mini-image). Once all the mini-images are denoised by the batch method, they are averaged together at their corresponding 2D locations in the large

[12] We did not observe any marked improvement in the PSNRs, when replacing DCT with other fixed transforms, such as wavelets.

Table 3.4: PSNR values (dB) and run times (seconds) for denoising the large gray-scale and color images with mini-batch transform learning (TL) based method, and the 2D DCT at different noise levels ($\sigma = 20$, 50, 100). The noisy image PSNRs are given under noise level σ in parentheses. The best denoising PSNR for each noise level and image is marked in bold.

Images	Methods	$\sigma = 20$ (22.11)	$\sigma = 50$ (14.15)	$\sigma = 100$ (8.13)	Run Times
Airport	DCT	28.79	24.65	21.00	23
	TL	**28.83**	**25.07**	**22.53**	28
Man	DCT	30.44	25.80	21.87	23
	TL	**30.64**	**26.62**	**23.88**	27
Railway	DCT	31.90	26.44	22.04	90
	TL	**32.42**	**27.58**	**24.35**	111
Campus	DCT	30.89	25.88	21.99	323
	TL	**33.10**	**27.47**	**23.24**	451

(a) (b)

Figure 3.5: Zoom-in of large-scale image (Man 1024×1024) denoising results: (a) Noisy image (PSNR = $22.11\,dB$), (b) Denoised image using the proposed adaptive mini-batch scheme (PSNR = $30.64\,dB$).

image. We observed that this alternative denoising scheme performs (on the noisy images in Table 3.4) comparably to, or worse than (by $0.1 - 0.3$ dB), our proposed mini-batch transform learning-based denoising scheme. Importantly, the alternative mini-image-based batch method is $7\times$ slower on the average.

As we have emphasized, the proposed adaptive online and mini-batch schemes are capable of being applied to realistic tasks such as real-time sparse coding (compression) and denoising. The idea of image inpainting using mini-batch synthesis dictionary learning has been discussed in [40]. However, the scheme therein does not solve a real-time adaptive inpainting problem. Rather, a dictionary is first learned over all the data, and then later used to reconstruct (with fixed dictionary) the patches. Therefore, we do not directly compare to that work here.

CHAPTER 4

VIDEO DENOISING BY ONLINE 3D SPARSIFYING TRANSFORM LEARNING

This chapter demonstrates video denoising application by learning 3D sparsifying transform [1]. Denoising is one of the most fundamental problems in signal processing. The goal in denoising is to take corrupted signals, images or video and process them to obtain clean or high-quality estimates. This is especially useful for applications that require high-quality signals and images such as medical imaging applications, surveillance video, etc. The ubiquitous use of relatively low-quality smart phone cameras has also led to the increasing importance of video denoising.

Several methods have been proposed in the past for the denoising of video data. Some of these methods are based on motion estimation and compensation [90, 91]. In these methods, on top of spatial similarity, temporal redundancy is exploited by filtering along the estimated motion trajectories. Other video denoising methods exploit the sparsity of video data in some known transform domain or dictionary such as the discrete cosine transform (DCT), or wavelets, to enable better noise attenuation [92, 93]. Non-local methods have also become very popular in video denoising in recent years. Methods such as VBM3D [6] and VBM4D [7] have been shown to provide excellent performance in video denoising. These methods also exploit sparsifying transforms such as the DCT as part of their framework.

Recently, the adaptation of sparse models (such as the synthesis dictionary model [10, 20], analysis dictionary model [14], or transform model [12, 11]) based on training signals has received increasing attention [13, 3, 39, 94, 54, 14, 57, 12], and has been shown to be beneficial in various applications including image or video denoising. While the data-driven adaptation of synthesis dictionaries for the purpose of denoising video or 3D data [95, 5] has been studied in some recent papers, the usefulness of learned sparsifying transforms has not been explored.

[1] The material of this chapter has previously appeared in [89]

In this work, we focus on video denoising by using learned 3D sparsifying transforms. As opposed to general synthesis dictionary model, where sparse coding is NP-hard (non-deterministic polynomial-time hard) [21, 22], the transform model has the advantage that sparse coding in the model can be performed exactly and cheaply by zeroing out all but a certain number of non-zero transform coefficients of largest magnitude. The learning of sparsifying transforms is typically much cheaper than synthesis or analysis dictionary learning [12, 17]. Very recently, we introduced the idea of online learning of sparsifying transforms for signals or image patches [85, 80]. Online learning is particularly useful for big data, and for applications such as real-time denoising, i.e., denoising of streaming data. As opposed to batch transform learning [12], where the transform is learned using all the training data, online transform learning has the advantage that it handles (training) data sequentially, and involves much cheaper computations, and memory requirements. It has also been shown to be cheaper than online overcomplete synthesis dictionary learning [85].

While we have shown the usefulness of online transform learning for large-scale image denoising [85], the usefulness of transform learning (either online or batch) for video denoising has not been explored. Moreover, video data typically have redundancy along the time axis, which will not be captured by learning sparsifying transforms for the 2D patches of the video frames. Therefore, in this chapter, we propose a novel online video denoising scheme based on 3D sparsifying transform learning. Our framework iteratively adapts the sparsifying transform and sparse codes for (overlapping) 3D (spatio-temporal) patches that are extracted sequentially from groups of frames. Denoised versions of the 3D patches are estimated in each iteration of our algorithm, and denoised versions of the video frames are estimated by averaging the denoised 3D patches at their respective spatio-temporal locations. Our numerical results demonstrate the promising performance of the proposed method as compared to well-known alternatives such as adaptive overcomplete dictionary-based denoising, VBM3D, VBM4D, or 3D DCT-based denoising.

4.1 Denoising Problem Formulations

We briefly discuss the recently proposed formulations for denoising based on online and mini-batch transform learning.

4.1.1 Signal or Image Denoising by Online Transform Learning

The goal in denoising is to recover an estimate of a signal u from the measurement $y = u + e$, corrupted by additive noise e. Here, we consider a time sequence of measurements $\{y_t\}$, with $y_t = u_t + e_t$, and $e_t \in \mathbb{R}^n$ being the noise. We assume e_t whose entries are independent and identically distributed (i.i.d.) Gaussian with zero mean and variance σ_t^2. The goal of online denoising is to recover estimates of $y_t \ \forall \ t$. We model the underlying signals as approximately sparse in an (unknown) transform domain.

In prior work [85], we proposed a denoising methodology based on online sparsifying transform learning, where the transform is adapted based on sequentially processed data. For time $t = 1, 2, 3, ...$, the problem of updating the adaptive transform and sparse code (i.e., the sparse representation in the adaptive transform domain) to account for the new noisy signal $y_t \in \mathbb{R}^n$ is

$$\text{(P1)} \left\{\hat{W}_t, \hat{x}_t\right\} = \underset{W,\, x_t}{\arg\min} \frac{1}{t} \sum_{\tau=1}^{t} \left\{\|W y_\tau - x_\tau\|_2^2 + \lambda_\tau \nu(W)\right\} + \frac{1}{t} \sum_{\tau=1}^{t} \alpha_\tau^2 \|x_\tau\|_0 \qquad s.t. \ x_\tau = \hat{x}_\tau, \ 1 \leq \tau \leq t-1$$

where $\nu(W) = -\log |\det W| + \|W\|_F^2$ is a transform learning regularizer [12], $\lambda_\tau = \lambda_0 \|y_\tau\|_2^2$ with $\lambda_0 > 0$, and the weights $\alpha_\tau \propto \sigma_\tau$. The $\|\cdot\|_0$ operation counts the number of non-zeros in a vector or matrix. Matrix $\hat{W}_t$ in (P1) is the optimal transform at time t, and $\hat{x}_t$ is the optimal sparse code for y_t. Note that at time t, only the latest optimal sparse code $\hat{x}_t$ is updated in (P1),[2] along with the transform $\hat{W}_t$. The condition $x_j = \hat{x}_j, \ 1 \leq \tau \leq t-1$, is therefore assumed. For brevity, we will not explicitly restate this condition (or, its appropriate variant) in the formulations in the rest of this chapter.

[2]This is because only the signal y_t is assumed to be stored in memory at time t for the online scheme.

The regularizer $\nu(W)$ in (P1) prevents trivial solutions and controls the condition number and scaling of the learned transform [12]. In the limit $\lambda_0 \to \infty$ (and assuming the y_τ, $1 \leq \tau \leq t$, are not all zero), the condition number of the optimal transform in (P1) tends to 1. In practice, the transforms learned via (P1) are well conditioned for finite λ_0 [85]. The specific choice of λ_0 (and condition number) depends on the application.

A simple least-squares denoised signal estimate is obtained using (P1) at each time t as $\hat{u}_t = \hat{W}_t^{-1} \hat{x}_t$. Problem (P1) can also be used for patch-based denoising of large images [85]. The overlapping patches of the noisy images are processed sequentially, and the denoised image is obtained by averaging the denoised patches at their respective image locations.

For non-stationary data, it may not be desired to fit a single transform W to y_t for all t. We previously proposed [85] to address this case by introducing a forgetting factor $\rho^{t-\tau}$ (with a constant $0 < \rho < 1$), that scales the terms in (P1). Such a forgetting factor diminishes the influence of "old" data. The objective function in (P1) is then modified as

$$\frac{1}{t} \sum_{\tau=1}^{t} \rho^{t-\tau} \left\{ \|W y_\tau - x_\tau\|_2^2 + \lambda_\tau \nu(W) + \alpha_\tau^2 \|x_\tau\|_0 \right\} \tag{4.1}$$

Another useful variation of Problem (P13) involves *mini-batch* learning, where a block, or group, or mini-batch of signals is processed at a time [85]. Assuming a fixed block size M, the L^{th} ($L \geq 1$) block of signals is $Y_L = \Big[y_{LM-M+1} \mid y_{LM-M+2} \mid \ \ldots \ \mid y_{LM} \Big]$. For $L = 1, 2, 3, ...$, the mini-batch sparsifying transform learning problem is

$$\begin{aligned} \left\{ \hat{W}_L, \hat{X}_L \right\} = \underset{W,\, X_L}{\arg\min} \frac{1}{LM} \sum_{j=1}^{L} \left\{ \|W Y_j - X_j\|_F^2 + \Lambda_j \, v(W) \right\} \\ + \frac{1}{LM} \sum_{j=1}^{L} \sum_{i=1}^{M} \alpha_{jM-M+i}^2 \left\| x_{jM-M+i} \right\|_0 \end{aligned} \tag{P14}$$

where the regularizer weight is $\Lambda_j = \lambda_0 \|Y_j\|_F^2$, and the matrix $X_L = \Big[x_{LM-M+1} \mid x_{LM-M+2} \mid \ \ldots \ \mid x_{LM} \Big]$ contains the block of sparse codes corresponding to Y_L. A simple denoised estimate of the noisy block of signals in Y_L is obtained for each L as $\hat{U}_L = \hat{W}_L^{-1} \hat{X}_L$. The mini-batch transform learning Problem (P2) is a generalized version of (P1), with (P2) being equivalent

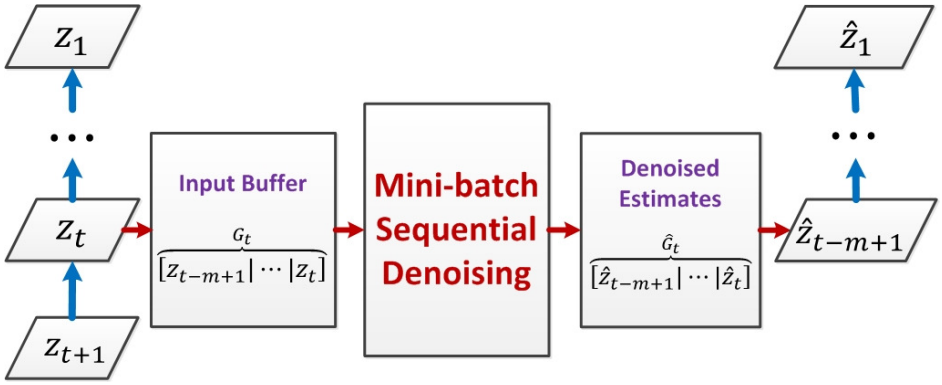


Figure 4.1: A simple illustration of the proposed online video denoising scheme by 3D sparsifying transform learning.

to (P1), for $M = 1$. Mini-batch learning can provide potential speedups over the $M = 1$ case in applications, but this comes at the cost of higher memory requirements and latency [85].

4.1.2 Online Video Denoising Framework

Prior work on adaptive sparsifying transform-based image denoising [17, 62, 85] learned the transform matrix from 2D image patches. However, in video denoising, exploiting the sparsity and redundancy in both the spatial and temporal dimensions typically performs better than denoising each frame separately [95, 5]. We therefore propose online video denoising by sparsifying transform learning on 3D spatio-temporal patches.

Figure 4.1 illustrates the framework of our proposed online video denoising scheme. The frames of the noisy video (assumed to be corrupted by additive i.i.d. Gaussian noise) denoted as $z_\tau \in \mathbb{R}^{a \times b}$ arrive at $\tau = 1, 2, 3,$ etc. At time $\tau = t$, the newly arrived frame z_t is added to a fixed-size FIFO (first in first out) buffer that stores a block of m consecutive frames $\{z_i\}_{i=t-m+1}^{t}$, and the oldest frame z_{t-m} is dropped. We denote this spatio-temporal tensor data of frames stacked along the temporal dimension as $G_t = \left[z_{t-m+1} \mid z_{t-m+2} \mid \ \ldots \ \mid z_t \right]$, with $G_t \in \mathbb{R}^{a \times b \times m}$. (For $t < m$, the unavailable frames are replaced by all-zero frames.) The partially overlapping $n_1 \times n_2 \times n_3$ 3D patches of G_t are extracted sequentially in a spatially and temporally contiguous order, and a spatio-temporal sparsifying transform is

adapted in an online manner, and used to denoise the patches. The denoised estimates of the set of noisy frames within each G_t are estimated by averaging the overlapping parts of the denoised 3D patches at their respective locations in 3D. Each noisy frame z_t arises once in each of the tensors in the set $\{G_j\}_{j=t}^{t+m-1}$. Therefore, the denoised estimates of z_t computed based on each of these m tensors are further averaged together to generate the final denoised version (output) of z_t. So, there is (at least) an $m-1$ frame delay between the arrival of z_t and the generation of its final denoised estimate. In Fig. 4.1, $\hat{G}_t$ stores the most up-to-date denoised estimates of the noisy frames in G_t. Only the leftmost frame $\hat{z}_{t-m+1}$ in $\hat{G}_t$ is output, since all other frame estimates will be updated further based on future G_τ's ($\tau > t$).

We now discuss the formulation for sequentially denoising the 3D spatio-temporal patches in each G_t (for $t = 1, 2, 3,$ etc.). Let $R_i G_t \in \mathbb{R}^n$ (with $n = n_1 n_2 n_3$, $n_3 \leq m$) denote the vectorized form of the i$^{\text{th}}$ 3D patch extracted from G_t (a total of P partially overlapping patches are assumed), with R_i being a patch-extraction operator. We process a group, or mini-batch, of M 3D patches at a time from G_t. Let N be the total number of mini-batches in each G_t ($N = P/M$). Then, for a particular time t, we solve the following transform learning problem for each $k = 1, 2, 3, ..., N$, to adapt the transform and sparse codes based on the k$^{\text{th}}$ mini-batch in G_t:

$$\left\{\hat{W}_{L_k}, \hat{X}_{L_k}\right\} = \underset{W, X_{L_k}}{\arg\min} \frac{1}{L_k M} \sum_{j=1}^{L_k} \rho^{L_k - j} \left\{ \|WY_j - X_j\|_F^2 \right\}$$
$$+ \frac{1}{L_k M} \sum_{j=1}^{L_k} \rho^{L_k - j} \left\{ \Lambda_j \, \nu(W) + \sum_{i=1}^{M} \alpha_{j,i}^2 \, \|X_{j,i}\|_0 \right\} \quad \text{(P15)}$$

where $L_k \triangleq N \times (t-1) + k$. In (P3), $Y_j = \left[R_{lM-M+1} G_{q+1} \mid ... \mid R_{lM} G_{q+1} \right] \in \mathbb{R}^{n \times M}$, with $q \triangleq \lfloor j/N \rfloor$ and $l \triangleq j - qN$, indexes the mini-batches processed from the various G_τ, $1 \leq \tau \leq t$. The matrix $X_j \in \mathbb{R}^{n \times M}$ denotes the sparse codes corresponding to the mini-batch Y_j, and $X_{j,i}$ denotes the i$^{\text{th}}$ column of X_j, whose sparsity weight in (P3) is $\alpha_{j,i}^2$. Note that the factor $L_k M$ in (P3) denotes the total number of 3D patches processed from all G_τ, $1 \leq \tau \leq t$. We use a forgetting factor $\rho^{L_k - j}$ in (P3) to diminish the influence of old frames or old mini-batches. Once (P3) is solved, the denoised version of the current mini-batch of noisy signals is computed simply as $\hat{U}_{L_k} = \hat{W}_{L_k}^{-1} \hat{X}_{L_k}$.

4.2 Algorithms and Properties

We refer to our video denoising methodology by solving (P3) as VIDOLSAT (VIdeo Denoising by Online Learning of SpArsifying Transforms). Our proposed method for (P3) involves a sparse coding step and a transform update step [85]. We perform another extra sparse coding step here for improved accuracy.

4.2.1 Sparse Coding

In the sparse coding step, we solve for $\hat{X}_{L_k}$ in (P3) with fixed $W = \hat{W}_{L_k-1}$, as follows:

$$\hat{X}_{L_k} = \underset{X_{L_k}}{\arg\min} \|WY_{L_k} - X_{L_k}\|_F^2 + \sum_{i=1}^{M} \alpha_{L_k,i}^2 \|X_{L_k,i}\|_0 \tag{4.2}$$

A solution to $\hat{X}_{L_k}$ of (4.2) is given as $\hat{X}_{L_k,i} = \hat{H}_{\alpha_{L_k,i}}(WY_{L_k,i})\ \forall\ i$ [85]. Here, the hard thresholding operator $\hat{H}_\alpha(\cdot)$ is defined as

$$\left(\hat{H}_\alpha(b)\right)_p = \begin{cases} 0 & , \ |b_p| < \alpha \\ b_p & , \ |b_p| \geq \alpha \end{cases} \tag{4.3}$$

where $b \in \mathbb{R}^n$, and the subscript p indexes vector entries. This simple hard thresholding operation for sparse coding is similar to traditional techniques involving analytical sparsifying transforms [87].

4.2.2 Transform Update

In the transform update step, we solve Problem (P3) for W with fixed $X_j = \hat{X}_j$, $1 \leq j \leq L_k$, as follows:

$$\min_W \frac{1}{L_k M} \sum_{j=1}^{L_k} \rho^{L_k-j} \left\{ \|WY_j - X_j\|_F^2 + \Lambda_j \nu(W) \right\} \tag{4.4}$$

This problem has a closed-form solution (similar to Section III-B2 in [85]).

Define $b_k = L_k M$. Let $P_{L_k} \in \mathbb{R}^{n \times n}$ be the square root of $b_k^{-1} \sum_{j=1}^{L_k} \rho^{L_k-j}(Y_j Y_j^T + \Lambda_j I)$. Denoting the full singular value decomposition (SVD) of $P_{L_k}^{-1}\Theta_{L_k}$ as

$Q_{L_k}\Sigma_{L_k}U_{L_k}^T$, with $\Theta_{L_k} = b_k^{-1}\sum_{j=1}^{L_k}\rho^{L_k-j}Y_jX_j^T$, we then have that the closed-form to (4.4) is

$$\hat{W}_{L_k} = 0.5U_{L_k}\left(\Sigma_{L_k} + \left(\Sigma_{L_k}^2 + 2\beta_{L_k}I\right)^{\frac{1}{2}}\right)Q_{L_k}^T P_{L_k}^{-1} \tag{4.5}$$

where I denotes the identity matrix, and $(\cdot)^{\frac{1}{2}}$ denotes the positive definite square of a positive definite matrix. The quantities $\Gamma_{L_k} \triangleq b_k^{-1}\sum_{j=1}^{L_k}\rho^{L_k-j}Y_{L_k}Y_{L_k}^T$, Θ_{L_k}, and $\beta_{L_k} \triangleq \sum_{j=1}^{L_k} b_k^{-1}\Lambda_j$ are all computed sequentially over time [85].

4.2.3 Multi-pass Denoising

In order to further enhance the denoising performance, we perform multiple passes of denoising for each G_t in our framework [62]. In each pass, we construct the Y_j's in (P3) using the 3D patches extracted from the latest denoised estimates of the G_t's from the previous pass. As the sparsity penalty weights $\alpha_{j,i} \propto \sigma$, the noise level σ in each such pass is set to an estimate of the remaining noise in the denoised G_t's from the previous pass.

4.2.4 VIDOLSAT Properties

The per-frame computational cost of the proposed VIDOLSAT algorithms is $O(n^2PK)$, where $W \in \mathbb{R}^{n\times n}$, P is the number of partially overlapping patches in G_t, and K is the number of passes in the multi-pass scheme. Assuming $J \gg nM/m$ (large videos), the proposed algorithms have memory cost scaling as $O(Jm)$, where m is the number of frames in G_t, and J is the number of pixels in each frame.

4.3 Numerical Experiments

In this section, we present preliminary results for our VIDOLSAT algorithm. We work with the standard gray-scale videos *Salesman* ($288 \times 352 \times 50$), *Miss America* ($288 \times 360 \times 150$), and *Coastguard* ($144 \times 176 \times 300$) (available at [96]), and simulate i.i.d. Gaussian noise at 5 different noise levels ($\sigma = 5, 10, 15, 20, 50$) for each video. We compare the denoising results obtained by our VIDOLSAT algorithm to those obtained by popular methods

Table 4.1: Comparison of video denoising PSNR values for several methods. **Top Left**: Patch-based 3D DCT denoising; **Top Right**: sparse K-SVD [5]; **Middle Left**: VBM3D [6]; **Middle Right**: VBM4D [7]; **Bottom Left**: VIDOLSAT with $n = 512$; **Bottom Right**: VIDOLSAT with $n = 768$. For each video and noise level, the best denoising PSNR is marked in bold.

σ	*Salesman*		*Miss America*		*Coastguard*	
5	40.87	41.03	42.03	41.99	38.47	38.55
	40.43	40.82	41.51	41.88	38.32	39.12
	41.55	**41.69**	42.30	**42.33**	**39.60**	39.53
10	36.92	37.02	39.46	39.72	34.61	34.75
	37.29	37.12	39.64	39.85	34.82	35.35
	37.84	**38.02**	40.31	**40.34**	**35.73**	35.67
15	34.66	34.73	37.66	38.35	32.52	32.70
	35.53	34.95	38.70	38.65	33.03	33.24
	35.59	**35.82**	39.19	**39.22**	**33.67**	33.65
20	33.07	33.21	36.21	37.25	31.07	31.33
	34.14	33.33	37.97	37.79	31.73	31.72
	34.02	**34.26**	38.32	**38.40**	32.23	**33.26**
50	27.84	28.37	30.60	33.41	26.56	27.06
	28.33	28.32	34.55	34.28	26.90	27.05
	29.34	**29.72**	35.15	**35.28**	27.99	**28.12**

such as VBM3D [6], VBM4D [7], sparse K-SVD denoising [5], and patch-based 3D DCT denoising (this is the same as the VIDOLSAT method, but using 3D DCT instead of the learned transform). We used the publicly available implementations of the sparse K-SVD [97], VBM3D and VBM4D [96] algorithms.

For our VIDOLSAT algorithms, we work with $8 \times 8 \times 8$ ($n = 512$) and $8 \times 8 \times 12$ ($n = 768$) overlapping 3D patches, with $m = 8$ ($m = n_3$) and 12 respectively. We set spatial overlap stride $v = 1$ for the 3D patches, $\lambda_0 = 1.0 \times 10^{-2}$, $M = 15 \times n$, and $\alpha_{j,i} = 1.9\sigma$. Other parameters such as ρ, K (number of passes) and the estimated noise levels in each pass of the multi-pass scheme were tuned empirically [85, 62]. For (fixed) 3D DCT denoising, the setting $\alpha_{j,i} = 2.45\sigma$ is used which was found to work optimally in our experiments.

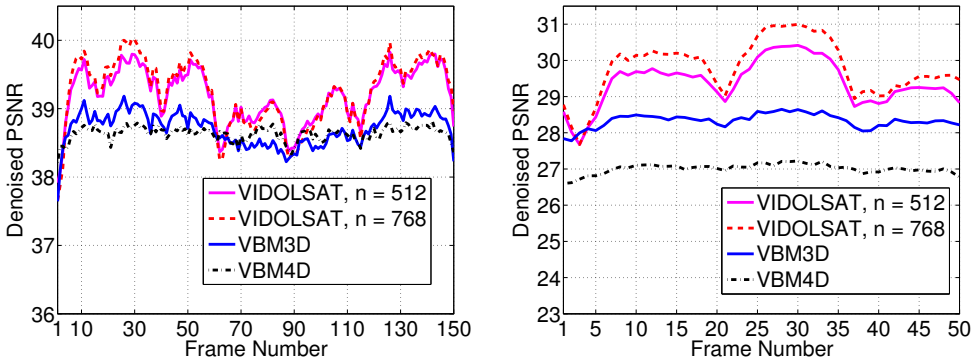


Figure 4.2: Frame-by-frame PSNR(dB) of the video (a) *Miss America* with $\sigma = 15$, and (b) *Salesman* with $\sigma = 50$ denoised by the proposed scheme VIDOLSAT ($n = 512$ and $n = 768$), VBM3D and VBM4D.

To evaluate the performance of the various schemes, we measure the denoised peak signal-to-noise ratio (PSNR) computed between the noiseless reference and the denoised video. Table 4.1 lists the denoised PSNRs obtained by 3D DCT denoising, sparse K-SVD denoising, VBM3D, VBM4D, and VIDOLSAT with two different temporal patch sizes. The VIDOLSAT algorithm with $n = 512$ provides average PSNR improvements in Table 4.1 of 1.35 dB, 0.89 dB, 0.66 dB, and 0.63 dB respectively over the 3D DCT, sparse KSVD, VBM3D, and VBM4D denoising methods. The corresponding improvements provided by VIDOLSAT with $n = 768$ are 1.45 dB, 0.99 dB, 0.76 dB, and 0.72 dB respectively. With either patch size, VIDOLSAT clearly provides better PSNRs than any of the competing methods for all videos and noise levels. Thus our proposed method demonstrates promising performance in video denoising compared to popular competing methods.

Figure 4.2 illustrates the frame-by-frame denoised PSNRs obtained using the VIDOLSAT algorithm for the videos *Miss America* and *Salesman* at $\sigma = 15$ and $\sigma = 50$, respectively, along with the corresponding PSNR values for VBM3D and VBM4D. It is clear that VIDOLSAT outperforms the competing methods for most of the frames. Figure 4.3 shows one frame of the denoised video *Salesman* at $\sigma = 50$. Comparing 4.3(c) and (d), the denoising result using VIDOLSAT clearly shows better reconstruction than the result using VBM4D from the highly noisy measurements.

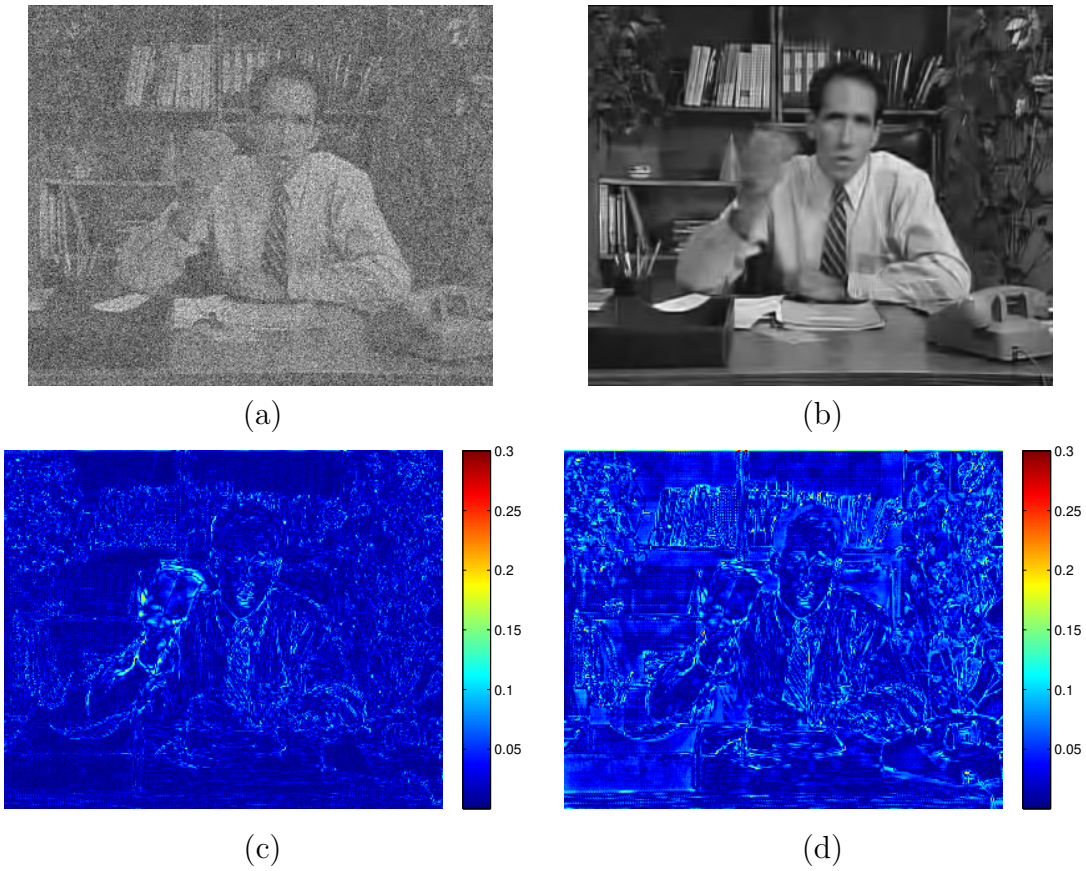


Figure 4.3: One frame of *Salesman* denoising result: (a) Noisy frame (PSNR = 14.13 dB). (b) Denoised frame using the proposed VIDOLSAT scheme with $n = 768$ (PSNR = 30.97 dB). (c) Magnitude of error in (b). (d) Magnitude of error in the denoised frame using VBM4D (PSNR = 27.20 dB).

CHAPTER 5

CONCLUSION

In this thesis, we focused on the transform model learning for sparse representations and its applications. We first presented a novel union of sparsifying transforms model. We showed that this model can also be interpreted as an overcomplete sparsifying transform model with an extra block cosparsity constraint (OCTOBOS) on the sparse code. The sparse coding in the proposed model can be interpreted as a form of clustering. We presented a novel problem formulation and algorithm for learning the proposed OCTOBOS transforms. Our algorithm involves simple closed-form solutions, and the theoretical analysis established global convergence of the algorithm to the set of partial minimizers of the non-convex learning problem. For natural images, our learning scheme gives rise to a union of well-conditioned transforms, and clustered patches or textures. It is also usually insensitive to initialization. The adapted model provides better sparsification errors and recovery PSNRs for images compared to learned single square transforms, and analytical transforms. In the application of image denoising, the proposed scheme typically provides better image reconstruction quality compared to adaptive (single) square transforms, and adaptive overcomplete synthesis dictionaries. These results suggest that the proposed OCTOBOS learning produces effective models adapted to the data. The usefulness of our OCTOBOS learning scheme in these applications merits detailed study and extensive evaluation. Likewise, other applications, e.g., inverse problems such as MRI [16] and CT [18, 19], and classification, merit further study.

We then presented the problem formulation for online learning of square sparsifying transforms. The formulation is to sequentially update the sparsifying transform and sparse code for signals that arrive, or are processed, sequentially. The proposed algorithm involves a sparse coding step and a transform update step per signal. Each of these steps is implemented efficiently. We also presented a mini-batch version of our online algorithm that

can handle blocks of signals at a time. The proposed schemes were shown to be computationally much cheaper (in terms of cost per signal) than online synthesis dictionary learning. In practice, the online/mini-batch sparsifying transform learning converges quickly. We presented experiments demonstrating the usefulness of online transform learning in sparse signal representation and denoising.

To further demonstrate the usefulness of the proposed online transform learning framework, we extended to learning 3D sparsifying transform learning for online video denoising. The proposed method uses a temporally sliding window strategy to extract a small set of noisy video frames at each time instant, and then generates an instantaneous denoised estimate of these frames using an efficient 3D (overlapping) patch-based denoising scheme. Specifically, the 3D patches of the set of noisy frames are extracted sequentially, and a sparsifying transform is adapted in an online manner, and used to denoise the patches. The denoised set of frames within each temporal window is estimated by averaging the denoised 3D patches at their respective locations in 3D. Each video frame arises in multiple overlapping temporal windows, and its denoised estimates computed in each of those windows are further averaged together to generate the final denoised version of that frame. Our numerical results demonstrate the promising performance of the proposed method as compared to well-known alternatives such as adaptive overcomplete dictionary-based denoising, VBM3D, VBM4D, or 3D DCT-based denoising. In future work, we plan to extend to online learning of an overcomplete transform, or other applications such as the denoising of 4D medical imaging data.

APPENDIX A

APPROXIMATION ERROR IN ONLINE ALGORITHM

Here, we calculate the error introduced by the approximation (3.10) in the transform update step of our online learning algorithm. Let us denote $L_t^{-1}\Theta_t$, $t^{-1}L_t^{-1}y_t x_t^T$, and $\left(L_t^{-1} - L_{t-1}^{-1}\right)\Theta_{t-1}$ by M_t, z_t, and Υ_t, respectively. Then, by (3.9), we have

$$M_t = (1 - t^{-1})M_{t-1} + (1 - t^{-1})\Upsilon_t + z_t \tag{A.1}$$

Equation (3.10) introduces an approximation to M_t, and then computes the SVD of the approximate matrix. Let us denote the approximate matrix as $\hat{M}_t$. The SVD of $\hat{M}_t$ is computed via a rank-1 update to the SVD of $(1 - t^{-1})\hat{M}_{t-1}$. Thus, we have by (3.10) that

$$\hat{M}_t = (1 - t^{-1})\hat{M}_{t-1} + z_t \tag{A.2}$$

Subtracting (A.2) from (A.1) and denoting $M_t - \hat{M}_t$ by E_t yields

$$E_t = (1 - t^{-1})(E_{t-1} + \Upsilon_t) \tag{A.3}$$

Assuming $E_1 = 0$, equation (A.3) implies that $E_t = \sum_{j=2}^{t} \frac{j-1}{t}\Upsilon_j$. ■

REFERENCES

[1] S. Ravishankar and Y. Bresler, "Closed-form solutions within sparsifying transform learning," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 5378–5382.

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

[3] 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, 2006.

[4] D. Zoran and Y. Weiss, "From learning models of natural image patches to whole image restoration," in *IEEE International Conference on Computer Vision*, Nov 2011, pp. 479–486.

[5] R. Rubinstein, M. Zibulevsky, and M. Elad, "Double sparsity: Learning sparse dictionaries for sparse signal approximation," *IEEE Transactions on Signal Processing*, vol. 58, no. 3, pp. 1553–1564, 2010.

[6] K. Dabov, A. Foi, and K. Egiazarian, "Video denoising by sparse 3d transform-domain collaborative filtering," in *Proc. 15th European Signal Processing Conference*, 2007.

[7] M. Maggioni, G. Boracchi, A. Foi, and K. Egiazarian, "Video denoising, deblocking, and enhancement through separable 4-d nonlocal spatiotemporal transforms," *IEEE Trans. Image Process.*, vol. 21, no. 9, pp. 3952–3966, 2012.

[8] S. Mallat, *A Wavelet Tour of Signal Processing*. Academic Press, 1999.

[9] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, "An overview of JPEG-2000," in *Proc. Data Compression Conf.*, 2000, pp. 523–541.

[10] M. Elad, P. Milanfar, and R. Rubinstein, "Analysis versus synthesis in signal priors," *Inverse Problems*, vol. 23, no. 3, pp. 947–968, 2007.

[11] W. K. Pratt, J. Kane, and H. C. Andrews, "Hadamard transform image coding," *Proc. IEEE*, vol. 57, no. 1, pp. 58–68, 1969.

[12] S. Ravishankar and Y. Bresler, "Learning sparsifying transforms," *IEEE Trans. Signal Process.*, vol. 61, no. 5, pp. 1072–1086, 2013.

[13] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on signal processing*, vol. 54, no. 11, pp. 4311–4322, 2006.

[14] R. Rubinstein, T. Faktor, and M. Elad, "K-SVD dictionary-learning for the analysis sparse model," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5405–5408.

[15] S. Ravishankar and Y. Bresler, "Learning sparsifying transforms for signal and image processing," in *SIAM Conference on Imaging Science*, 2012, p. 51.

[16] S. Ravishankar and Y. Bresler, "Sparsifying transform learning for compressed sensing MRI," in *Proc. IEEE Int. Symp. Biomed. Imag.*, 2013, pp. 17–20.

[17] S. Ravishankar and Y. Bresler, "Learning doubly sparse transforms for images," *IEEE Trans. Image Process.*, vol. 22, no. 12, pp. 4598–4612, 2013.

[18] L. Pfister, "Tomographic reconstruction with adaptive sparsifying transforms," M.S. thesis, University of Illinois at Urbana-Champaign, Aug. 2013.

[19] L. Pfister and Y. Bresler, "Model-based iterative tomographic reconstruction with adaptive sparsifying transforms," in *SPIE International Symposium on Electronic Imaging: Computational Imaging XII*, 2014.

[20] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, 2009.

[21] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comput.*, vol. 24, no. 2, pp. 227–234, Apr. 1995.

[22] G. Davis, S. Mallat, and M. Avellaneda, "Adaptive greedy approximations," *Journal of Constructive Approximation*, vol. 13, no. 1, pp. 57–98, 1997.

[23] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit : recursive function approximation with applications to wavelet decomposition," in *Asilomar Conf. on Signals, Systems and Comput.*, 1993, pp. 40–44 vol.1.

[24] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[25] I. F. Gorodnitsky, J. George, and B. D. Rao, "Neuromagnetic source imaging with FOCUSS: A recursive weighted minimum norm algorithm," *Electrocephalography and Clinical Neurophysiology*, vol. 95, pp. 231–251, 1995.

[26] G. Harikumar and Y. Bresler, "A new algorithm for computing sparse solutions to linear inverse problems," in *ICASSP*, May 1996, pp. 1331–1334.

[27] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, 1998.

[28] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *Annals of Statistics*, vol. 32, pp. 407–499, 2004.

[29] W. Dai and O. Milenkovic, "Subspace pursuit for compressive sensing signal reconstruction," *IEEE Trans. Information Theory*, vol. 55, no. 5, pp. 2230–2249, 2009.

[30] J. Mairal, M. Elad, and G. Sapiro, "Sparse representation for color image restoration," *IEEE Trans. on Image Processing*, vol. 17, no. 1, pp. 53–69, 2008.

[31] M. Aharon and M. Elad, "Sparse and redundant modeling of image content using an image-signature-dictionary," *SIAM Journal on Imaging Sciences*, vol. 1, no. 3, pp. 228–247, 2008.

[32] J. Mairal, G. Sapiro, and M. Elad, "Learning multiscale sparse representations for image and video restoration," *SIAM Multiscale Modeling and Simulation*, vol. 7, no. 1, pp. 214–241, 2008.

[33] H. Y. Liao and G. Sapiro, "Sparse representations for limited data tomography," in *Proc. IEEE International Symposium on Biomedical Imaging (ISBI)*, 2008, pp. 1375–1378.

[34] S. Ravishankar and Y. Bresler, "MR image reconstruction from highly undersampled k-space data by dictionary learning," *IEEE Trans. Med. Imag.*, vol. 30, no. 5, pp. 1028–1041, 2011.

[35] S. Ravishankar and Y. Bresler, "Multiscale dictionary learning for MRI," in *Proc. ISMRM*, 2011, p. 2830.

[36] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[37] K. Engan, S. Aase, and J. Hakon-Husoy, "Method of optimal directions for frame design," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1999, pp. 2443–2446.

[38] M. Yaghoobi, T. Blumensath, and M. Davies, "Dictionary learning for sparse approximations with the majorization method," *IEEE Transaction on Signal Processing*, vol. 57, no. 6, pp. 2178–2191, 2009.

[39] K. Skretting and K. Engan, "Recursive least squares dictionary learning algorithm," *IEEE Transactions on Signal Processing*, vol. 58, no. 4, pp. 2121–2130, 2010.

[40] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *J. Mach. Learn. Res.*, vol. 11, pp. 19–60, 2010.

[41] S. K. Sahoo and A. Makur, "Dictionary training for sparse representation as generalization of k-means clustering," *IEEE Signal Processing Letters*, vol. 20, no. 6, pp. 587–590, June 2013.

[42] L. N. Smith and M. Elad, "Improving dictionary learning: Multiple dictionary updates and coefficient reuse," *IEEE Signal Processing Letters*, vol. 20, no. 1, pp. 79–82, Jan 2013.

[43] M. Sadeghi, M. Babaie-Zadeh, and C. Jutten, "Dictionary learning for sparse representation: A novel approach," *IEEE Signal Processing Letters*, vol. 20, no. 12, pp. 1195–1198, Dec 2013.

[44] R. Rubinstein, A. M. Bruckstein, and M. Elad, "Dictionaries for sparse representation modeling," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1045–1057, 2010.

[45] R. Rubinstein and M. Elad, "K-SVD dictionary-learning for analysis sparse models," in *Proc. SPARS11*, June 2011, p. 73.

[46] A. Chambolle, "An algorithm for total variation minimization and applications," *J. Math. Imaging Vis.*, vol. 20, no. 1-2, pp. 89–97, 2004.

[47] S. Nam, M. E. Davies, M. Elad, and R. Gribonval, "Cosparse analysis modeling - uniqueness and algorithms," in *ICASSP*, 2011, pp. 5804–5807.

[48] E. J. Candès, Y. C. Eldar, D. Needell, and P. Randall, "Compressed sensing with coherent and redundant dictionaries," *Applied and Computational Harmonic Analysis*, vol. 31, no. 1, pp. 59–73, 2011.

[49] Y. Liu, M. Tiebin, and S. Li, "Compressed sensing with general frames via optimal-dual-based ℓ_1-analysis," *IEEE Transactions on Information Theory*, vol. 58, no. 7, pp. 4201–4214, July 2012.

[50] M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, "Noise aware analysis operator learning for approximately cosparse signals," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5409–5412.

[51] R. Giryes, S. Nam, M. Elad, R. Gribonval, and M. Davies, "Greedy-like algorithms for the cosparse analysis model," *Linear Algebra and its Applications*, vol. 441, pp. 22–60, 2014, special Issue on Sparse Approximate Solution of Linear Systems.

[52] G. Peyré and J. Fadili, "Learning analysis sparsity priors," in *Proc. Int. Conf. Sampling Theory Appl. (SampTA)*, Singapore, May 2-6, 2011.

[53] M. Yaghoobi, S. Nam, R. Gribonval, and M. Davies, "Analysis operator learning for overcomplete cosparse representations," in *European Signal Processing Conference (EUSIPCO)*, 2011, pp. 1470–1474.

[54] B. Ophir, M. Elad, N. Bertin, and M. Plumbley, "Sequential minimal eigenvalues - an approach to analysis dictionary learning," in *Proc. European Signal Processing Conference (EUSIPCO)*, 2011, pp. 1465–1469.

[55] Y. Chen, T. Pock, and H. Bischof, "Learning ℓ_1-based analysis and synthesis sparsity priors using bi-level optimization," in *Proc. Workshop on Analysis Operator Learning vs. Dictionary Learning, NIPS*, 2012.

[56] S. Hawe, M. Kleinsteuber, and K. Diepold, "Analysis operator learning and its application to image reconstruction," *IEEE Transactions on Image Processing*, vol. 22, no. 6, pp. 2138–2150, June 2013.

[57] M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, "Constrained overcomplete analysis operator learning for cosparse signal modelling," *IEEE Trans. Signal Process.*, vol. 61, no. 9, pp. 2341–2355, 2013.

[58] E. J. Candès and D. L. Donoho, "Ridgelets: A key to higher-dimensional intermittency?" *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 357, no. 1760, pp. 2495–2509, 1999.

[59] M. N. Do and M. Vetterli, "The contourlet transform: an efficient directional multiresolution image representation," *IEEE Transactions on Image Processing*, vol. 14, no. 12, pp. 2091–2106, 2005.

[60] E. J. Candès and D. L. Donoho, "Curvelets - a surprisingly effective nonadaptive representation for objects with edges," in *Curves and Surfaces*. Vanderbilt University Press, 1999, pp. 105–120.

[61] S. Ravishankar and Y. Bresler, "Learning doubly sparse transforms for image representation," in *IEEE Int. Conf. Image Process.*, 2012, pp. 685–688.

[62] B. Wen, S. Ravishankar, and Y. Bresler, "Structured overcomplete sparsifying transform learning with convergence guarantees and applications," *Int. J. Computer Vision*, vol. 114, no. 2, pp. 137–167, 2015.

[63] S. Ravishankar, "Adaptive sparse representations and their applications," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Dec. 2014.

[64] D. L. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via 1 minimization," *Proceedings of the National Academy of Sciences*, vol. 100, no. 5, pp. 2197–2202, 2003.

[65] G. Yu, G. Sapiro, and S. Mallat, "Solving inverse problems with piecewise linear estimators: From Gaussian mixture models to structured sparsity," *IEEE Transactions on Image Processing*, vol. 21, no. 5, pp. 2481–2499, 2012.

[66] D. A. Spielman, H. Wang, and J. Wright, "Exact recovery of sparsely-used dictionaries," in *Proceedings of the 25th Annual Conference on Learning Theory*, 2012, pp. 37.1–37.18.

[67] A. Agarwal, A. Anandkumar, P. Jain, P. Netrapalli, and R. Tandon, "Learning sparsely used overcomplete dictionaries via alternating minimization," 2013, preprint: http://arxiv.org/abs/1310.7991.

[68] S. Arora, R. Ge, and A. Moitra, "New algorithms for learning incoherent and overcomplete dictionaries," 2013, preprint: http://arxiv.org/pdf/1308.6273v5.pdf.

[69] Y. Xu and W. Yin, "A fast patch-dictionary method for whole-image recovery," 2013, UCLA CAM report 13-38.

[70] C. Bao, H. Ji, Y. Quan, and Z. Shen, "ℓ_0 norm based dictionary learning by proximal methods with global convergence," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3858–3865.

[71] A. Agarwal, A. Anandkumar, P. Jain, P. Netrapalli, and R. Tandon, "Learning sparsely used overcomplete dictionaries," *Journal of Machine Learning Research*, vol. 35, pp. 1–15, 2014.

[72] S. Ravishankar and Y. Bresler, "ℓ_0 sparsifying transform learning with efficient optimal updates and convergence guarantees," *IEEE Trans. Signal Process.*, vol. 63, no. 9, pp. 2389–2404, 2014.

[73] M. Elad, "Michael Elad personal page," 2009, online: http://www.cs.technion.ac.il/ elad/.

[74] B. Wen, S. Ravishankar, and Y. Bresler, "Learning overcomplete sparsifying transforms with block cosparsity," in *IEEE International Conference on Image Processing (ICIP)*, 2014.

[75] P. Brodatz, *Textures: A Photographic Album for Artists and Designers*. New York: Dover, 1965.

[76] I. Ramirez, P. Sprechmann, and G. 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.

[77] C. Lu, J. Shi, and J. Jia, "Online robust dictionary learning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 415–422.

[78] J. Xing, J. Gao, B. Li, W. Hu, and S. Yan, "Robust object tracking with online multi-lifespan dictionary learning," in *IEEE International Conference on Computer Vision (ICCV)*, Dec 2013, pp. 665–672.

[79] G. Zhang, Z. Jiang, and L. S. Davis, "Online semi-supervised discriminative dictionary learning for sparse representation," in *Computer Vision ACCV 2012*, 2013, pp. 259–273.

[80] S. Ravishankar and Y. Bresler, "Online sparsifying transform learning - part ii: Convergence analysis," *IEEE Journal of Selected Topics in Signal Process.*, vol. 9, no. 4, pp. 637–646, 2015.

[81] L. Wang, K. Lu, P. Liu, R. Ranjan, and L. Chen, "IK-SVD: Dictionary learning for spatial big data via incremental atom update," *Computing in Science & Engineering*, vol. 16, no. 4, pp. 41–52, 2014.

[82] S. Mukherjee and C. S. Seelamantula, "A split-and-merge dictionary learning algorithm for sparse representation," 2014, preprint: .

[83] P. Stange, "On the efficient update of the singular value decomposition," *PAMM*, vol. 8, no. 1, pp. 10 827–10 828, 2008.

[84] P. Stange, "On the efficient update of the singular value decomposition subject to rank-one modifications," 2011, online: http://www.digibib.tu-bs.de/?docid=00040371.

[85] S. Ravishankar, B. Wen, and Y. Bresler, "Online sparsifying transform learning - part i: Algorithms," *IEEE Journal of Selected Topics in Signal Process.*, vol. 9, no. 4, pp. 625–636, 2015.

[86] S. Ravishankar, B. Wen, and Y. Bresler, "Online sparsifying transform learning for signal processing," *IEEE Global Conference on Signal and Information Processing*, 2014.

[87] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 1994.

[88] "The USC-SIPI Image Database," [Online: http://sipi.usc.edu/database/database.php?volume=misc; accessed July-2014].

[89] B. Wen, S. Ravishankar, and Y. Bresler, "Video denoising by online 3d sparsifying transform learning," in *IEEE International Conference on Image Processing (ICIP)*, 2015.

[90] V. Zlokolica, A. Pizurica, and W. Philips, "Recursive temporal denoising and motion estimation of video," in *Proc. IEEE Int. Conf. Image Proc., ICIP*, vol. 3, 2004, pp. 1465–1468.

[91] F. Jin, P. Fieguth, and L. Winger, "Wavelet video denoising with regularized multiresolution motion estimation," *EURASIP Journal on Advances in Signal Processing*, vol. 2006, pp. 1–11, 2006.

[92] D. Rusanovskyy and K. Egiazarian, "Video denoising algorithm in sliding 3d dct domain," in *Advanced Concepts for Intelligent Vision Systems*, 2005, pp. 618–625.

[93] N. Rajpoot, Z. Yao, and R. Wilson, "Adaptive wavelet restoration of noisy video sequences," in *Proc. IEEE Int. Conf. Image Proc., ICIP*, vol. 2, 2004, pp. 957–960.

[94] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Non-local sparse models for image restoration," in *IEEE 12th International Conference on Computer Vision*, 2009, pp. 2272–2279.

[95] M. Protter and M. Elad, "Image sequence denoising via sparse and redundant representations," *IEEE Trans. on Image Processing*, vol. 18, no. 1, pp. 27–36, 2009.

[96] "BM3D website," online: http://www.cs.tut.fi/ foi/GCF-BM3D/; accessed Dec 2014.

[97] R. Rubinstein, "Ron Rubinstein web page," online: http://www.cs.technion.ac.il/ ronrubin; accessed Dec 2014.