A Tree-based Dictionary Learning Framework

Renato Budinich & Gerlind Plonka

July 22, 2019

We propose a new outline for dictionary learning methods based on a hierarchical clustering of the training data. Through recursive application of a clustering method the data is organized into a binary partition tree representing a multiscale structure. The dictionary atoms are defined adaptively based on the data clusters in the partition tree. This approach can be interpreted as a generalized wavelet transform. The computational bottleneck of the procedure is then the chosen clustering method: when using K-means the method runs much faster than K-SVD. Thanks to the multiscale properties of the partition tree, our dictionary is structured: when using Orthogonal Matching Pursuit to reconstruct patches from a natural image, dictionary atoms corresponding to nodes being closer to the root node in the tree have a tendency to be used with greater coefficients.

1 Introduction

In many applications one is interested in sparsely approximating a set of N n-dimensional data points Y_j, columns of an $n \times N$ real matrix $\mathbf{Y} = (Y_1, \dots, Y_N)$. Assuming that the data can be efficiently represented in a transformed domain, given by applying a linear transform $\mathbf{D} \in \mathbb{R}^{n \times K}$, one is interested in solving the sparse coding problem

$$\min_{\mathbf{X} \in \mathbb{R}^{K \times N}} ||\mathbf{Y} - \mathbf{D}\mathbf{X}|| \,, \qquad \text{where} \quad ||X_j||_0 \leq S \quad \forall\, j = 1, \dots, N \,, \tag{1.1}$$

where $S \in \mathbb{R}$ is a parameter called *sparsity*, X_j is the j-th column of the encoding matrix $\mathbf{X} = (X_1, \dots, X_N) \in \mathbb{R}^{K \times N}$ and $||\cdot||_0$ is the so-called 0-norm which is defined as the number of non-zero components of a vector (and is not really a norm). The jth column of the *encoding matrix* $\mathbf{X}$ gives the coefficients used in the linear combination of columns of $\mathbf{D}$ (which are termed *atoms* of the *dictionary*) to approximate the j-th column Y_j of $\mathbf{Y}$. How well the data Y_j can indeed by approximated by $\mathbf{D}X_j$ with an S-sparse vector X_j is of course dependent on $\mathbf{Y}$ and on the choice of $\mathbf{D}$.

The sparse coding problem in (1.1) is NP-hard (see Natarajan (1995)) and thus one can only hope to find an approximate minimizer $\mathbf{X}$. Within the last years a multitude of methods has been proposed to find approximated solutions to problem (1.1). Most of these are greedy algorithms that sequentially select the S dictionary atoms to approximate the columns Y_j of $\mathbf{Y}$, as e.g. Orthogonal Matching Pursuit (OMP) or the Iterative Thresholding method by Blumensath and Davies (2008). Many approaches replace the 0-norm in (1.1) by the 1-norm to obtain a convex minimization problem that can in turn be solved efficiently, see e.g. Beck and Teboulle (2009); Chambolle and Pock (2011) and Basis Pursuit methods, see e.g. Pati et al. (1993); Davies et al. (1997); Tropp (2004). For specific dictionary matrices exact solvers exists, see e.g. Dragotti and Lu (2014) for $\mathbf{D} = [\mathbf{I}, \mathbf{F}]$ with $\mathbf{I}$ the identity and $\mathbf{F}$ the Fourier matrix.

Finding a dictionary matrix $\mathbf{D}$ that admits the most efficient representation of the given data set $\mathbf{Y}$ is even more delicate. The often considered *dictionary learning* problem consists in finding both the optimal transformation $\mathbf{D}$ and the sparse coding matrix $\mathbf{X}$,

$$\min_{\mathbf{D}\in\mathbb{R}^{n\times K}, \mathbf{X}\in\mathbb{R}^{K\times N}} ||\mathbf{Y} - \mathbf{D}\mathbf{X}|| \qquad \text{where} \quad ||X_j||_0 \leq S \quad \forall j = 1, \ldots, N\,. \tag{1.2}$$

In this problem (which is also also NP-hard, see Tillmann (2015)) one is supposing that there exists an approximate factorization $\mathbf{DX}$ of the matrix $\mathbf{Y}$ where $\mathbf{X}$ is (column-wise) sparse. The most well-known method to tackle (1.2) is the K-SVD algorithm by Aharon et al. (2006). K-SVD decouples the problem (1.2) into a nested minimization problem and proposes an iteration scheme which at every step updates independently first the encoding matrix $\mathbf{X}$ and then the dictionary $\mathbf{D}$. At every iteration step of the K-SVD method, K SVDs of a matrix with n rows and at most N columns must be computed: in practice this is rather expensive.

The models (1.1) and (1.2) both implicitly assume that the given training data points Y_j are vectors. However, in many applications the data already possesses a multidimensional spatial structure, which is not leveraged when the data points are vectorized into the columns of the matrix $\mathbf{Y}$. In the last years there have been attempts to propose other dictionary learning methods, which on the one hand try to take the structure of the data into account and on the other hand impose further structure of the dictionary matrix in order to come up with more efficient dictionary learning algorithms for special applications, see e.g. Yankelevsky and Elad (2016); Cai et al. (2014); Liu et al. (2017, 2018).

In this paper, we want to propose a general dictionary learning approach, which is based on organizing the training data into a binary tree corresponding to a hierarchical clustering, thereby providing a multiscale structure that we employ to construct the dictionary atoms of $\mathbf{D}$. In particular, we completely separate the sparse coding problem (1.1) and the problem of fixing the dictionary $\mathbf{D}$.

The dictionary learning process consists of two steps; the computation of the binary partition tree which provides a hierarchical adaptive clustering of the training data, and the determination of the dictionary elements from the partition tree.

There is a variety of possibilities to construct the binary partition tree. In particular, we can choose different similarity measures to cluster the data sets into two clusters; the similarity measure should particularly account for the a priori structure of the data Y_j. Furthermore, we can employ certain pre-defined structure of the dictionary elements, as e.g. tensor product structure as in Zeng et al. (2015) or rank conditions as proposed in Liu et al. (2018). The choice of the similarity measure strongly influences the efficiency of the partition tree computation and thus the complete dictionary learning method.

In order to determine the dictionary elements from the partition tree we propose a procedure that can be interpreted as an adaptive generalized Haar wavelet transform. To illustrate this analogy, we will show that the classical Haar wavelet transform can be transferred to a binary tree construction from bottom to top, the usual "local to global" approach. Due to its linearity and invertibility, this transform it is however equivalent to the top to bottom construction which is the one we use in our method, thus making it "global to local". However, our proposed method is adaptive which means the tree is data dependent, while in the classical Haar wavelet case the tree is completely determined by a linear transform.

Having found the dictionary matrix $\mathbf{D}$ from the clusters in the binary tree, we still need to solve the sparse coding problem (1.1). For our application we will use OMP to sparsely code the data. We compare our method with K-SVD in various natural image reconstruction tasks: it usually performs slightly worse in terms of quality of the reconstruction but is faster especially for growing number of data points. This is to be expected since, when using Lloyd's algorithm for K-means, our algorithm has linear complexity.

The structure of this paper is as follows. In Section 2 we extensively describe the proposed procedure for dictionary learning. We start with the construction of the binary partition tree in Section 2.1 and show in Section 2.2 how to extract the atoms from the partition tree. In Section 2.3, we illustrate the connection of our dictionary construction with an adaptive Haar wavelet transform. Section 2.4 is concerned with some algorithmic aspects of the dictionary learning procedure. In Section 3 we present some application results for various reconstruction tasks comparing our method to K-SVD.

2 Tree-based dictionary learning framework

Differently from other dictionary learning methods, where the dictionary matrix $\mathbf{D}$ and the sparse coding matrix $\mathbf{X}$ are optimized simultaneously, our proposed method concerns itself only with learning $\mathbf{D}$; a sparse coding method such as OMP must be employed in a second, separate step.

Assume that we are given set of data $\mathbf{Y} = \{Y_1, \ldots, Y_N\}$, where all Y_j have the same known data structure. The Y_j can be any type of data, as long as:

- we have a meaningful two-way clustering method for it, which should ideally separate the data according to salient features;
- we can take linear combinations of the samples.

We will thus ask that the samples live in a vector space V. The Y_j can for example be vectors, $Y_j \in \mathbb{R}^n$, image patches $Y_j \in \mathbb{R}^{m_1 \times m_2}$, tensors, or more generally have a finite graph structure. The dictionary learning process itself consists of two parts:

1. computation of a binary partition tree which gives a hierarchical clustering of the training data,
2. determination of dictionary elements from the partition tree.

Within the next two subsections we will introduce notations and describe these two steps in detail.

2.1 Construction of the partition tree

We assume that each data sample Y_j can be uniquely identified by its index $j \in \{1, \ldots, N\}$. We want to construct a binary partition tree T whose nodes are associated to subsets of the index set $\{1, \ldots, N\}$, i.e. each node must correspond to a unique subset of the training data. We will interchangeably identify the nodes with the subset of indexes or of data points - this should be clear from the context and won't be source of ambiguity. Let the root node be

$$\mathcal{N}_{0,0} := \{1, \ldots, N\},$$

in general, $\mathcal{N}_{\ell,k}$ is the node at level ℓ that has $\mathcal{N}_{\ell+1,2k}$ and $\mathcal{N}_{\ell+1,2k+1}$ as children nodes. For a binary tree with a *complete* level ℓ we have 2^ℓ nodes in this level, i.e., there will be nodes $\mathcal{N}_{\ell,k}$ for all $k \in \{0, \ldots, 2^\ell - 1\}$. If the level is not complete, there will be nodes $\mathcal{N}_{\ell,k}$ only for certain values of k. We call this tree the *partition* tree because for each (non-leaf) node $\mathcal{N}_{\ell,k}$ of the tree, the two children nodes satisfy the properties

$$\begin{aligned} \mathcal{N}_{\ell+1,2k} \cup \mathcal{N}_{\ell+1,2k+1} &= \mathcal{N}_{\ell,k}\ , \\ \mathcal{N}_{\ell+1,2k}\ , \mathcal{N}_{\ell+1,2k+1} &\neq \emptyset \text{ and} \\ \mathcal{N}_{\ell+1,2k} \cap \mathcal{N}_{\ell+1,2k+1} &= \emptyset. \end{aligned} \tag{2.1}$$

The tree T is generated by recursive application of the clustering method to partition a given subset of the data into two subsets. The tree T obtained in this way need not to be complete (i.e. not all leaf nodes will in general be at the same level), and the number of elements in the subsets $\mathcal{N}_{\ell+1,2k}$ and $\mathcal{N}_{\ell+1,2k+1}$ need not to be the same. Thus, we will need some rule to decide whether a node set $\mathcal{N}_{\ell,k}$ will be partitioned into two further subsets or not; we will discuss this in Section 2.4.

In order to obtain a meaningful partition tree that leads to a good dictionary, we need to choose an appropriate similarity measure that governs the clustering procedure. To reduce the numerical effort, we may first employ a dimensionality reduction method to the given data and then apply the clustering method according to a suitable distance measure to the reduced data. If we know that the data (and thus also the dictionary elements that we want to construct) should have a certain special structure, as e.g., block circulant or block Toeplitz matrices, then this structure could be already employed in the dimensionality reduction step.

In our numerical experiments in Section 3 we compare K-means, K-maxoids (see Bauckhage and Sifa (2015)), both with $K = 2$, and Spectral Clustering. K-maxoids and especially K-means are faster due to the lower computational complexity of the algorithms; from a practical point of view, the main difference between the two is that while K-means offers as representatives of the clusters the sample average of data points therein contained, K-maxoids gives as representative a particular data point. Spectral Clustering has the theoretical advantage that it can be applied on a data-graph built in any way from the data: one isn't then restricted to the Euclidean distance but can cluster the data based on any type of similarity measure between the data points.

In the remainder of the section, we present a toy example and some further remarks on possible strategies for construction of the partition tree.

Example 1. We are given the set of training patches

$$Y_1 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 1 & 3 \end{pmatrix}, Y_2 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 1 & 5 \end{pmatrix}, Y_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, Y_4 = \begin{pmatrix} 2 & 0 & 0 \\ 5 & 5 & 0 \\ 2 & 7 & 5 \end{pmatrix},$$
$$Y_5 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{pmatrix}, Y_6 = \begin{pmatrix} 2 & 2 & 0 \\ 3 & 5 & 1 \\ 2 & 5 & 7 \end{pmatrix}, Y_7 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 2 \end{pmatrix}, Y_8 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

To construct the partition tree, we fix first the root node $\mathcal{N}_{0,0} := \{1, 2, 3, 4, 5, 6, 7, 8\}$. We create the tree using 2-means clustering and the `FIFO` procedure with parameters `mincard`$= 2$ and $\epsilon = 1$; see Section 2.4. This means, further branching will only be performed if the cardinality of a node is above `mincard` and the clustering minimization function is above the threshold ϵ. The clustering minimization function used here is the so-called within-cluster sum of squares (WCSS) or distortion measure, see Budinich (2018). Initially 2-means is applied to the full set of training patches corresponding to the node $\mathcal{N}_{0,0}$, separating patches Y_4 and Y_6 from the rest; then it is run on node $\mathcal{N}_{1,0} = \{1, 2, 3, 5, 7, 8\}$, splitting it into $\mathcal{N}_{2,0} = \{1, 2, 5\}$ and $\mathcal{N}_{2,1} = \{3, 7, 8\}$. The tree obtained is displayed in Figure 1.

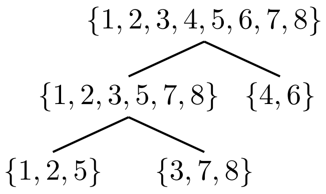


Figure 1: Partition tree obtained by applying the 2-means-clustering to the data in Example 1.

For comparison, we also employ another approach to determine the partition tree to the same training data set, where we first use a dimensionality reduction procedure before employing the 2-means algorithm. For the first partition, we compute the centroid $A_{0,0} := \frac{1}{8}\sum_{j=1}^{8} Y_j$ and evaluate the spectral norms of the difference matrices $s_j := \|A_{0,0} - Y_j\|_2$ for $j = 1, \ldots, 8$, thereby reducing the Y_j to a one-dimensional feature. Let

$$s_{r_1} \leq s_{r_2} \leq \ldots \leq s_{r_8}$$

be the obtained ordered feature numbers of training data. In this special case for the set of cardinality 8, the 2-means algorithm reduces to the minimization problem

$$\hat{\mu} := \operatorname*{arg\,min}_{1\leq\mu\leq 7} \left[\sum_{n=1}^{\mu} \left(s_{r_n} - \frac{1}{\mu}\sum_{\nu=1}^{\mu} s_{r_\nu} \right)^2 + \sum_{n=\mu+1}^{8} \left(s_{r_n} - \frac{1}{8-\mu}\sum_{\nu=\mu+1}^{8} s_{r_\nu} \right)^2 \right] \tag{2.2}$$

which can be solved exactly.

We obtain the partition into $\mathcal{N}_{1,0} = \{1,2,4,5,6\}$, $\mathcal{N}_{1,0} = \{3,7,8\}$. We proceed further in the same way for partitioning these subsets and obtain the tree in Figure 2. Here we have applied a partition of a subsets as long as we have more than two entries in this set or the value $\hat{\mu}$ in (2.2) is larger than 1.

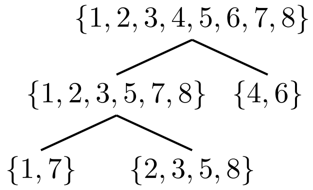


Figure 2: Partition tree obtained by applying the 2-means-clustering to the reduced data in Example 1.

If the norm of the different training data strongly varies, we recommend a normalization of the data before starting the process of tree construction.

Remark 1. There is a large variety of further methods that may be applied to achieve the partition tree.
1. For example, in Liu et al. (2018), a method has been employed to image patches, where in the first step, all elements Y_j of the training set are approximated by their rank-1 approximations $\sigma_j \mathbf{u}_j\, \mathbf{v}_j^T$ of Y_j using e.g. a partial SVD. Here σ_j denotes the maximal singular value of Y_j, and $\mathbf{u}_j$, $\mathbf{v}_j$ the corresponding left and right singular vectors. As a similarity measure for the 2-means algorithm a one-dimensional feature is defined that compares the rank-1 approximation of the centroids of the node clusters with the rank-1 approximations of Y_j. Such a procedure is particularly meaningful if the training patches themselves are noisy.
2. Differently from the example above, one may employ rank-r approximations $Y_j^{(r)}$ of the training patches $Y_j \in \mathbb{R}^{m_1 \times m_1}$ with $r \leq m_1$. The partition into subsets can be performed using the Frobenius norm as a similarity measure $\| \, Y_j^{(r)} - Y_k^{(r)}\|_F$, which is

the Euclidean distance of the vectorized patches in $\mathbb{R}^{m_1^2}$. For clustering, one may apply the 2-means algorithm, the 2-maxoid method by Bauckhage and Sifa (2015) or a spectral clustering method, see Shi and Malik (2000); Yan et al. (2009).
3. For image training patches we may apply a Fourier, DCT or wavelet transform to all Y_j in a first step to obtain M-term approximation of Y_j, where only the M largest coefficients in the wavelet expansion of Y_j are kept. Then the clustering procedure is applied to these approximations of the image patches to determine the partition tree.

2.2 Dictionary construction from the partition tree

In this subsection we will describe how to extract the dictionary elements from the partition tree. We will apply a multiscale procedure which is borrowed from the classical Haar wavelet construction but is here applied to our adaptive setting. To extract the dictionary from the partition tree we have to make a choice for the representative of each node: we will in general use the sample averages, and call these the *centroids* of the subset. We are however not limited to this choice and depending on the application, clustering method or structure of the data we may choose different representatives; see below for some examples.
Once we have a set way to choose representatives for each node, we take as dictionary atoms

1. a first “low-pass” element, which is given by the representative of the root node;

2. for each node $\mathcal{N}_{\ell,k}$ in the partition tree which possesses two children nodes $\mathcal{N}_{\ell+1,2k}$ and $\mathcal{N}_{\ell+1,2k+1}$, the difference between the representatives of the children nodes.

We define the centroid of node (ℓ, k) as

$$A_{\ell,k} := \frac{1}{|\mathcal{N}_{\ell,k}|} \sum_{j \in \mathcal{N}_{\ell,k}} Y_j \tag{2.3}$$

and then define the dictionary atoms as

$$\begin{aligned} A_{0,0} &= \frac{1}{N} \sum_{j=1}^{N} Y_j, \\ D_{\ell,k} &= A_{\ell+1,2k} - A_{\ell+1,2k+1} \ . \end{aligned} \tag{2.4}$$

Note that these centroids or differences of centroids have then to be normalized according to some fixed norm to get the dictionary elements - we will always use the Frobenius norm. We call the so obtained dictionary the **Haar dictionary** and we denote it with $\mathbf{D}^H$, i.e.

$$\mathbf{D}^H := \{A_{0,0}, D_{0,0}, D_{1,0}, D_{1,1,}, \ldots\} \ . \tag{2.5}$$

The choice of this name will become clear in Section 2.3, where we show the connection to the Haar wavelet transform.

Remark 2. By contrast, one can also take the normalized centroids corresponding to the leaves of the partition tree as dictionary atoms. We will call this the **centroids dictionary** and denote it with $\mathbf{D}^C$, i.e.

$$\mathbf{D}^C := \{A_{\lambda_1}, A_{\lambda_1}, \ldots\} \ , \tag{2.6}$$

where $\lambda_1, \lambda_2, \ldots$ are the leaves of the partition tree. An approach similar to the centroid dictionary construction has been also taken in Zeng et al. (2015), where however the centroids are replaced by their (normalized) rank-d approximations. More precisely, Zeng et al. (2015) considered a two-dimensional dictionary transform with a left and a right dictionary matrix, where the matrices built from the d right singular vectors and the d left singular vectors, respectively, form elements of the left resp. right dictionary. This means that the dictionary elements are (approximations of) centroids of small sets of image patches building a cluster of low variance according to the used similarity measure.

The atoms in the centroids dictionary $\mathbf{D}^C$ corresponding to the lower nodes in the tree may potentially suffer from excessively high correlation, given that they represent clusters in close proximity of one another. It is known that high correlation between dictionary atoms is not ideal for sparse representation (see for example Elad (2010)). Therefore we would advise to use the Haar dictionary $\mathbf{D}^H$ instead, especially for very deep trees.

Example 2. We reconsider the toy example 1 in Subsection 2.1 with the partition tree in Figure 1. By computing the sample averages of all the patches in each node we obtain the following centroids (rounded to two digits):

$$A_{0,0} = \begin{pmatrix} 1.13 & 0.25 & 0 \\ 1.5 & 2.38 & 0.13 \\ 0.63 & 1.88 & 3.38 \end{pmatrix}, \ A_{1,0} = \begin{pmatrix} 0.83 & 0 & 0 \\ 0.67 & 1.5 & 0 \\ 0.17 & 0.5 & 2.5 \end{pmatrix}, \ A_{1,1} = \begin{pmatrix} 2 & 1 & 0 \\ 4 & 5 & 0.5 \\ 2 & 6 & 6 \end{pmatrix},$$

$$A_{2,0} = \begin{pmatrix} 1 & 0 & 0 \\ 0.67 & 2 & 0 \\ 0 & 0.67 & 4.3 \end{pmatrix}, \ A_{2,1} = \begin{pmatrix} 0.67 & 0 & 0 \\ 0.67 & 1 & 0 \\ 0.33 & 0.33 & 0.67 \end{pmatrix}.$$

We thus obtain the two dictionaries as

$$\mathbf{D}^C := \left\{ \frac{A_{0,0}}{||A_{0,0}||_F}, \frac{A_{2,0}}{||A_{2,0}||_F}, \frac{A_{2,1}}{||A_{2,1}||_F}, \frac{A_{1,1}}{||A_{1,1}||_F} \right\}$$

$$\mathbf{D}^H := \left\{ \frac{A_{0,0}}{||A_{0,0}||_F}, \frac{A_{1,0} - A_{1,1}}{||A_{1,0} - A_{1,1}||_F}, \frac{A_{2,0} - A_{2,1}}{||A_{2,0} - A_{2,1}||_F} \right\} \ .$$

Note that the centroids dictionary $\mathbf{D}^C$ has one element more than the Haar dictionary $\mathbf{D}^H$ - this is always the case.

As already mentioned, we are not limited to choosing the centroids as *representatives* of a subset of training data: in this regard there is a large variety of possibilities, where in particular special dictionary structure can be incorporated. This choice can also depend on the chosen clustering procedure: 2-maxoids for example outputs not only the partition but also two maxoids, which are particular data points belonging to each of these two subsets respectively. Thus in our numerical tests, when using 2-maxoids we define $A_{\ell,k}$ as the maxoid of node $\mathcal{N}_{\ell,k}$ and leave (2.4) unchanged.

Example 3. We describe here the procedure that has been employed in Liu et al. (2018). The construction of a partition tree is similar to ours, see also Remark 1. To determine the dictionary, $A_{0,0}$ and all the other centroids $A_{\ell,k}$ as in (2.3) are employed. Liu et al. then compute the optimal rank-1 approximations of these centroids of the form

$$\sigma_k \mathbf{u}_k \mathbf{v}_k^T$$

where σ_k is the maximal singular value of $A_{\ell,k}$ and $\mathbf{u}_k$ and $\mathbf{v}_k$ are the corresponding left and right singular vectors. These rank-1 approximations are used as representatives for the construction of the dictionary: the low-pass dictionary element

$$A_0 := \frac{\mathbf{u}_0 \mathbf{v}_0^T}{\left|\left|\mathbf{u}_0 \mathbf{v}_0^T\right|\right|_F}$$

and the further dictionary elements

$$\tilde{D}_{\ell,k} := \sigma_{2k} \mathbf{u}_{2k} \mathbf{v}_{2k}^T - \sigma_{2k+1} \mathbf{u}_{2k+1} \mathbf{v}_{2k+1}^T, \qquad D_{\ell,k} := \frac{\tilde{D}_{\ell,k}}{\left|\left|\tilde{D}_{\ell,k}\right|\right|_F}.$$

This construction provides dictionary elements of rank at least 2 and is particularly suitable for noisy training patches.

Example 4. Another possibility could be for example to use M-term wavelet expansions of the centroids as representatives of the subsets. The atom $D_{\ell,k}$ of $\mathbf{D}^H$ would then be the normalized difference of the two obtained M-term approximations $\hat{A}_{\ell+1,2k}$ and $\hat{A}_{\ell+1,2k+1}$. The obtained dictionary atoms would thus possess at most $2M$ terms in the wavelet expansion.

2.3 The Haar-dependency Tree

In this subsection we want to show the connection between the Haar wavelet dictionary and our tree-based dictionary construction. We start by recalling some basic facts about the one-dimensional Haar transform which is the most simple case of wavelet transform, see for example Damelin and Miller Jr (2012) or Mallat (2008). Suppose we are given a digital signal $\mathbf{a} \in \mathbb{R}^N$, for simplicity let $N = 2^L$ for some $L \in \mathbb{N}$ and denote the N components of $\mathbf{a}_L := \mathbf{a}$ by $a_{L,0}, \ldots, a_{L,N-1}$. For $j = L-1, \ldots, 0$, we define the recursive formulas for the so-called *approximation* and *detail* coefficients of the transform as

$$\begin{aligned} a_{j,k} &= \frac{1}{\sqrt{2}}\big(a_{j+1,2k} + a_{j+1,2k+1}\big), \\ d_{j,k} &= \frac{1}{\sqrt{2}}\big(a_{j+1,2k} - a_{j+1,2k+1}\big), \end{aligned} \qquad k = 0, 1, \ldots, 2^j - 1\,. \tag{2.7}$$

These formulas are known as *synthesis formulas* and j is known as the *level* of the transform: the lower the level the coarser approximation the coefficients $a_{j,k}$ provide since they are an average of more samples. By applying the synthesis formulas recursively L times, the vector $\mathbf{a}_L = (a_{L,0}, \ldots, a_{L,N-1})^T$ is linearly transformed into the vector $(a_{0,0}, d_{0,0}, \mathbf{d}_1^T, \mathbf{d}_2^T, \ldots, \mathbf{d}_{L-1}^T)^T$, where $\mathbf{d}_j := (d_{j,0}, \ldots,\ d_{j,2^j-1})^T$ contains the detail or wavelet coefficients of level j.

The synthesis formulas are easily inverted to obtain the *reconstruction formulas*

$$\begin{aligned} a_{j+1,2k} &= \frac{1}{\sqrt{2}}(a_{j,k} + d_{j,k}) \\ a_{j+1,2k+1} &= \frac{1}{\sqrt{2}}(a_{j,k} - d_{j,k}) \end{aligned} \tag{2.8}$$

The linear transform is hence invertible, and in fact even orthogonal.

It is possible to represent the dependency between approximation coefficients in the synthesis formulas by means of a binary tree (similarly to what is done in Murtagh (2007)), by associating a node to each $a_{j,k}$ which has two sons, $a_{j+1,2k}$ and $a_{j+1,2k+1}$. We start by identifying each of the original samples $a_{L,k}$ with a leaf node, and subsequently for each level $j = L-1, \ldots, 0$ and for each $k = 0, \ldots, 2^j - 1$ we add a node (corresponding to $a_{j,k}$) which has as sons the two coefficients at the previous levels from which it is computed, i.e. $a_{j+1,2k}$ and $a_{j+1,2k+1}$. If we apply the full L levels of the Haar wavelet transform we obtain a binary tree with root node $a_{0,0}$; note that the concept of level of the tree and of the Haar wavelet transform here coincide, with the root node being at level 0 and the original samples at level L. In Figure 3 we show this tree for $N = 8$: here we're labeling each node with the respective approximation and detail coefficients. The synthesis formulas (2.7) tell us that we can compute the labels of a node from the approximation coefficient of its son nodes, while the reconstruction formulas (2.8) tell us the reverse process is possible. Since the Haar wavelet transform is invertible, we can equivalently represent the leaves of the tree (the original samples) using all the detail coefficients $d_{j,k}$ in the non-leaf nodes and the approximation coefficient $a_{0,0}$ related to the root node.

This tree representation of the coefficients allows to clearly determine the dependency among them: a coefficient is determined by all and only the samples that are leaf nodes in the sub-tree rooted in itself. This idea has been used for example in Budinich (2017) (in an adaptive setting) to reconstruct only a region of interest in an image while retaining some global information.

It is possible to express this dependency explicitly: with a simple induction proof it

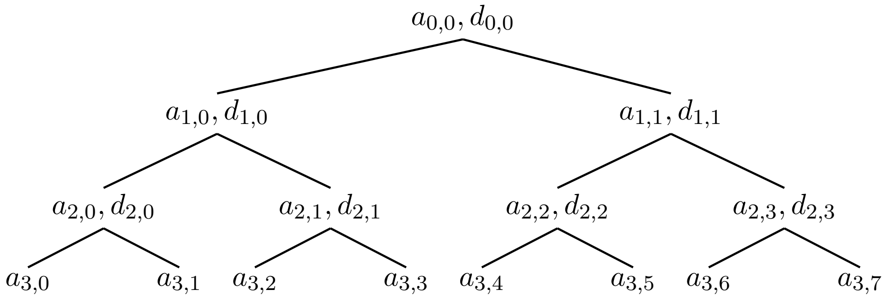


Figure 3: The Haar dependency tree for $N = 8$.

can be seen that for $k = 0, 1, \ldots, L$, we have

$$\begin{aligned} a_{L-\ell,k} &= 2^{-\frac{\ell}{2}} \sum_{h=k2^\ell}^{(k+1)2^\ell-1} a_{L,h} \\ \text{for } \ell > 0 \quad d_{L-\ell,k} &= 2^{-\frac{\ell}{2}} \left(\sum_{h=k2^\ell}^{(2k+1)2^{\ell-1}-1} a_{L,h} - \sum_{h=(2k+1)2^{\ell-1}}^{(k+1)2^\ell-1} a_{L,h} \right) , \end{aligned} \tag{2.9}$$

i.e., we can write each approximation and detail wavelet coefficient as a linear combination of the samples a_L themselves. Note that here ℓ is indicating the co-level, or equivalently the depth of the subtree rooted in the node. If we define index sets

$$\mathcal{N}_{L-\ell,k} = \{k2^\ell, k2^\ell + 1, \ldots, (k+1)2^\ell - 1\} ,$$

we can rewrite (2.9) as

$$\begin{aligned} a_{L-\ell,k} &= 2^{-\frac{\ell}{2}} \sum_{j \in \mathcal{N}_{L-\ell,k}} a_{L,j} \\ \text{for } \ell > 0 \quad d_{L-\ell,k} &= 2^{-\frac{\ell}{2}} \left(\sum_{j \in \mathcal{N}_{L-(\ell-1),2k}} a_{L,j} - \sum_{j \in \mathcal{N}_{L-(\ell-1),2k+1}} a_{L,j} \right) , \end{aligned} \tag{2.10}$$

which resemble formulas (2.4), with the exception of the normalization coefficient. This is an important distinction, arising from the fact that the Haar wavelet transform is deterministic and thus we know explicitly how the index sets $\mathcal{N}_{L-\ell,k}$ are made; in particular their cardinality depends only on the co-level ℓ. In the definition of $D_{\ell,k}$ in (2.4) instead we are weighing the sums over $\mathcal{N}_{\ell,2k}$ and $\mathcal{N}_{\ell,2k+1}$ with the reciprocal of their cardinalities, which will in general be different.

2.4 Algorithmic aspects for hierarchical clustering based dictionaries

As represented in the preceding subsections, our method for dictionary learning consists of two conceptual steps, computing a hierarchical clustering of the training data and its associated partition tree and computing the dictionary atoms from this tree. To achieve the hierarchical clustering we will use 2-means, 2-maxoids and spectral clustering with various data-graphs, see Section 3 for more details.

To determine the dictionary elements, we employ the procedure described in Section 2.2. In the following we summarize some algorithmic aspects of the dictionary construction procedure, where we assume that N training samples are given, and we want to learn $K \leq N$ dictionary elements. In Algorithm (1) an outline of our procedure for the centroid dictionary $\mathbf{D}^C$ and the Haar dictionary $\mathbf{D}^H$ is given as pseudo-code. There are two important variables to specify for each instance of this procedure: the data structure used for storing the nodes to visit in the tree (line (2)) and the branching criteria to evaluate on each node (line (7)). There are two main choices for the setting of these variables that determine a different behavior of the algorithm: in the first case the `tovisit` data structure is set to a FIFO[1] queue and in the second to a priority queue.

In the first case, when `tovisit` is a FIFO queue, the tree will be visited breadth-first and the branching criteria will be set to check for two conditions:

1. whether the cardinality of node ν is above a threshold `mincard`,
2. whether the value of the clustering minimization function is above a threshold ϵ [2].

In this case we do not have direct control over the cardinality K of the produced dictionary, we simply know that it will be a decreasing function of ϵ. On the other hand we have the certainty that the final clusters will be very small: they either must have fewer than `mincard` elements or, when partitioned further, give a value of the clustering minimization function below ϵ. This means that the clustering procedure gives some sort of adaptive resolution of the space: the tree branches go deeper where the data is more spread out, and in any case they go deep enough so that in all regions of the data space the final clusters have approximately the same size.

In the second case, when `tovisit` is a priority queue, we use the variance of the node being put in `tovisit` as the key and we always extract the value from `tovisit` with the highest key value. This means that we give priority in the tree visit to those nodes corresponding to higher variance, or equivalently we explore first those regions of the data space where the data is more spread out. In this case the branching criteria is set to check for the two following conditions:

1. whether the cardinality of ν is above a threshold `mincard`,
2. whether the number of branchings already occurred is smaller than $K - 1$

[1] first in first out

[2] we could alternatively check if some measure of "spreadness" of the data corresponding to the node ν (for example its variance) is above the set threshold

This means that, if the sample set is large enough, exactly $K-1$ branchings will occur, and thus the Haar-dictionary will consist of K dictionary elements. In our tests we will always use this priority queue variant because of the convenience of setting the dictionary cardinality K. However, depending on the application the FIFO variant might be more appropriate.

Algorithm 1 Haar-like tree based dictionary learning procedure

```
Input: Training data $\mathcal{S} = \{Y_1, \dots, Y_N\}$, clustering procedure, mincard, dictionary cardinality $K$ or parameter $\epsilon$
Output: Centroids dictionary $\mathbf{D}^C$ and Haar dictionary $\mathbf{D}^H$
Initialize $\mathbf{D}^C = \mathbf{D}^H := \{A_r\}$
Initialize tovisit = DataStructure()
tovisit.put($r$)
while tovisit is not empty do
    $\nu$ = tovisit.get()
    Partition $\mathcal{N}_\nu$ into $\mathcal{N}_{\nu_0}$ and $\mathcal{N}_{\nu_1}$ (with representatives $A_{\nu_0}$ and $A_{\nu_1}$)
    if BRANCH_CRITERIA($\nu$) is TRUE then
        if $\nu \neq r$ then
            Remove $A_\nu$ from $\mathbf{D}^C$
        end if
        Add edges $(\nu, \nu_0)$ and $(\nu, \nu_1)$ to $E$
        Add $A_\nu$ to $\mathbf{D}^C$ and $D_\nu$ to $\mathbf{D}^H$
        tovisit.put($\nu_0$)
        tovisit.put($\nu_1$)
    end if
end while
```

The computational complexity of our method essentially depends on the clustering procedure used and on the number of branchings done (i.e., the number of nodes in the tree). Denoting with $\tilde{\mathcal{N}}$ the leaf nodes of the tree and supposing that we use the 2-means clustering by computing $\mathcal{I}$ iterations of Lloyd's algorithm, for each non-leaf node $\nu \in \mathcal{N} \setminus \tilde{\mathcal{N}}$ we require $\mathcal{O}(|\mathcal{S}_\nu| n\mathcal{I})$ elementary operations for the clustering and $\mathcal{O}(|\mathcal{S}_\nu| n)$ for computing the associated dictionary element. Thus in this case the total computational cost is

$$\sum_{\nu \in N \setminus \tilde{N}} \mathcal{O}(|\mathcal{S}_\nu| n\mathcal{I}) \leq \mathcal{O}(KNn\mathcal{I}) \ . \tag{2.11}$$

Remark 3. Our method can be adapted to online dictionary learning (see for example Mairal et al. (2009) and Lu et al. (2013)). In this scenario one wishes to update the dictionary based on new incoming learning data. If we suppose that the structure of the hierarchical clustering remains unchanged even with the addition of the new training data, it is sufficient to assign each new data point to the cluster corresponding to one of the leaves and then travel on the tree from these leaf nodes up to the root node. Only

the dictionary atoms associated to nodes so encountered (i.e. the ancestors of the leaf nodes containing the new data points) will be affected.
If the new incoming data presents very different features than the original training data, then it would produce substantially different clusters and the hypotheses of the tree not changing would become unrealistic. One could identify the regions of the data space that are changing and recompute only the corresponding subtrees.
Finally, our method can be used to produce, with no further computational costs, sub-dictionaries adapted to only a portion of the data. This can be done by simply selecting an appropriate subtree and the dictionary atoms associated to it. This could be used for example to accelerate the sparse coding procedure, by first assigning a sample to one of the leaf clusters and then selecting a subtree containing this leaf, whose associated dictionary would be used for sparse coding. One could thus regulate with a parameter the trade-off between speed and accuracy of the sparse coding method: a more shallow tree would correspond to a smaller dictionary and thus faster computation times.

3 Numerical Experiments

In this section we will carry out natural image reconstruction tasks using K-SVD and particular variants of our method. We will compare computation times and the quality of the reconstructions using the HaarPSI index (Reisenhofer et al. (2018)). The HaarPSI of two images is a real number in $(0, 1]$ indicating the visual similarity of two images, where 1 means the two images are the same and a lower number indicates higher distortion. We choose this index because we are testing reconstruction of natural images and the HaarPSI has the best correlation with human subjective quality assessment. The implementation of our method was done in python[3] while for K-SVD we used the KSVD-box Matlab software[4]. All the numerical tests were run on a MacBook Pro Mid 2012 with an Intel Ivy Bridge i5 2.5Ghz CPU. The exact code used to produce the results in this section can be found in the `batch_tests.py` file in the git repository.

Using patches extracted from the `flowers_pool` image (Figure 4) as training data, we computed the dictionaries with various methods and used OMP to reconstruct patches from the same image. While we randomly extracted patches from the set of overlapping patches in the image, we reconstructed non-overlapping patches due to time inefficiency of OMP when dealing with a large number of data points. We ran the test with different values of patch number, patch size, clustering method and reconstruction sparsity. For clustering we used the classical K-means method with $K = 2$, the K-maxoids method (Bauckhage and Sifa (2015)) with $K = 2$ and the Spectral Clustering method (Shi and Malik (2000)) with different data graphs. The K-maxoids is slightly slower than K-means but offers as class representatives some particular patches in the data-set as opposed to the cluster centroids. In our case we hypothesized this would be an advantage, since it would give us dictionary patches that are more sharp and less blurry, which will be summed in linear combinations anyways by OMP. Spectral Clustering relies on what

[3] available at `https://github.com/nareto/haardict`

[4] avaiable at http://www.cs.technion.ac.il/~ronrubin/software.html

we call the data similarity graph, a complete graph with vertices given by patches and edge weights given by their similarity under some measure. For the latter we used the Frobenius norm, the aforementioned HaarPSI and the Earth Movers' Distance (see for example Rubner et al. (2000)). Because of the $O(N^2)$ computations of such similarity measure required spectral clustering is much slower and unusable for larger patch sizes and quantity; we thus restricted the computation of this clustering to simpler cases. In all cases we set the dictionary cardinality to be 50% bigger than the dimension of the vectorized patches, i.e. for 8×8 patches we computed dictionaries with 96 atoms.

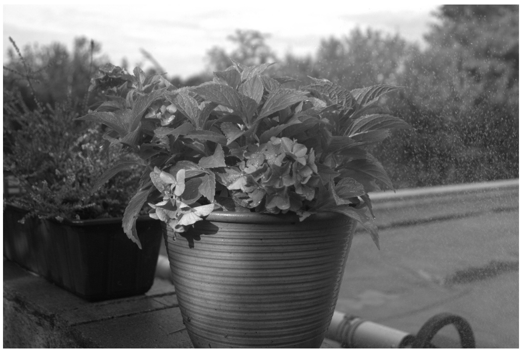


Figure 4: `flowers_pool` image

In Figure 5 (left) the computation times to learn various dictionaries are shown, in logarithmic scale. It can be clearly seen that spectral clustering performs much worse than 2-means or 2-maxoids, especially when using HaarPSI or Earth Mover's Distance as similarity measure. The reconstruction HaarPSI values (shown in Figure 5 (right)) are only in certain cases better than other methods. Overall we consider spectral clustering's computation times prohibitive for anything but very small number of data points, and we thus excluded it from further tests.
In Figure 6 (left) we plot the computation times required for learning dictionaries trained on different number of 8×8 patches. In Figure 6 (right) instead we plot the HaarPSI values of the reconstructed images (with sparsity 4) from these dictionaries. The learning times clearly show the better performance of our method, especially when using 2-means clustering. K-SVD still gives better quality reconstructions though, followed by the Haar-dictionary with 2-means clustering and the Centroids dictionary with 2-maxoids clustering.

We observed that our Haar-dictionary captures some structure that is not present in the K-SVD atoms: dictionary atoms associated to nodes at smaller levels in the tree (i.e. closer to the root node) are used with larger coefficients by OMP. To see this we consider the solution $\mathbf{X}$ (in the notation of (1.1)) proposed by OMP, we sum its rows in absolute values and associate these numbers to the corresponding dictionary atoms; we define

$$\eta_k := \sum_{j=1}^{N} |X_{kj}| \,, \qquad k = 1, \dots, K \,. \tag{3.1}$$

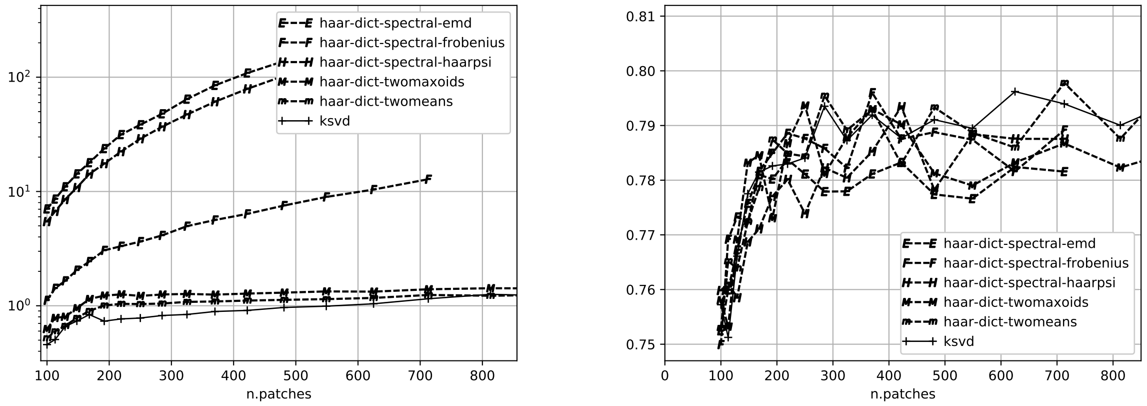


Figure 5: Left: Times in seconds (in logarithmic scale) required to learn the dictionaries as a function of the number of 8 × 8 patches used for training. Right: HaarPSI values of the reconstructed images with sparsity 5 as a function of the number of patches used for training.

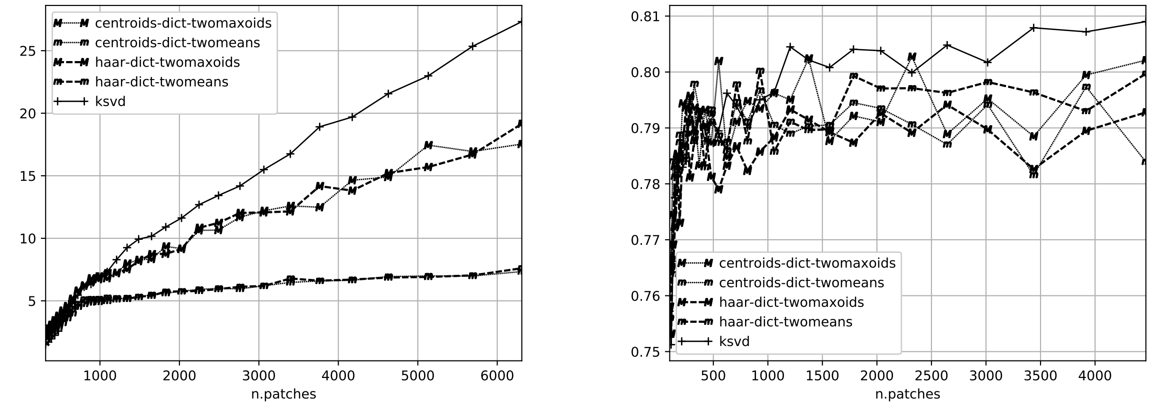
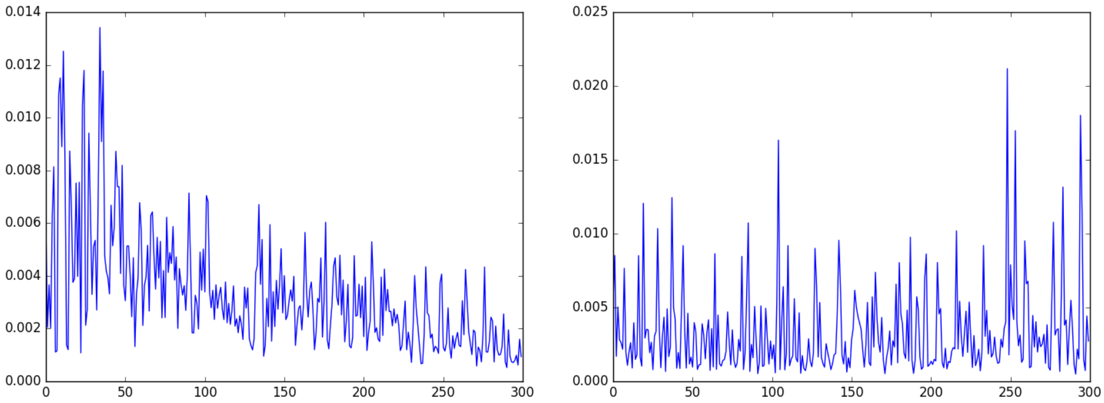


Figure 6: Left: Times in seconds required to learn the dictionaries as a function of the number of 16 × 16 patches used for training. Right: HaarPSI values of the reconstructed images with sparsity 4 as a function of the number of patches used for training.

The number η_k gives us a measure of how important the dictionary atom k is, in the sense that it is more used in the sparse linear combinations of the reconstructed patches. In Figure 7 we represent the vectors η for the Haar-dictionary (with 2-means clustering) and the K-SVD dictionary: it can be seen in both plots (and this is mostly the case in all the tests we've conducted) that there are few atoms that are used very frequently in the reconstruction and other atoms that are used with far less frequency. The difference however is that the plot for the Haar-dictionary presents a decreasing trend: atoms that are computed earlier are more used by OMP. Since in this case the FIFO tree visit strategy was used, these atoms correspond to the first levels of the tree: this means that the atoms that OMP uses the most in the sparse coding procedure are given by the differences between representatives of large clusters, i.e., they distinguish between features of the data at a very coarse level.



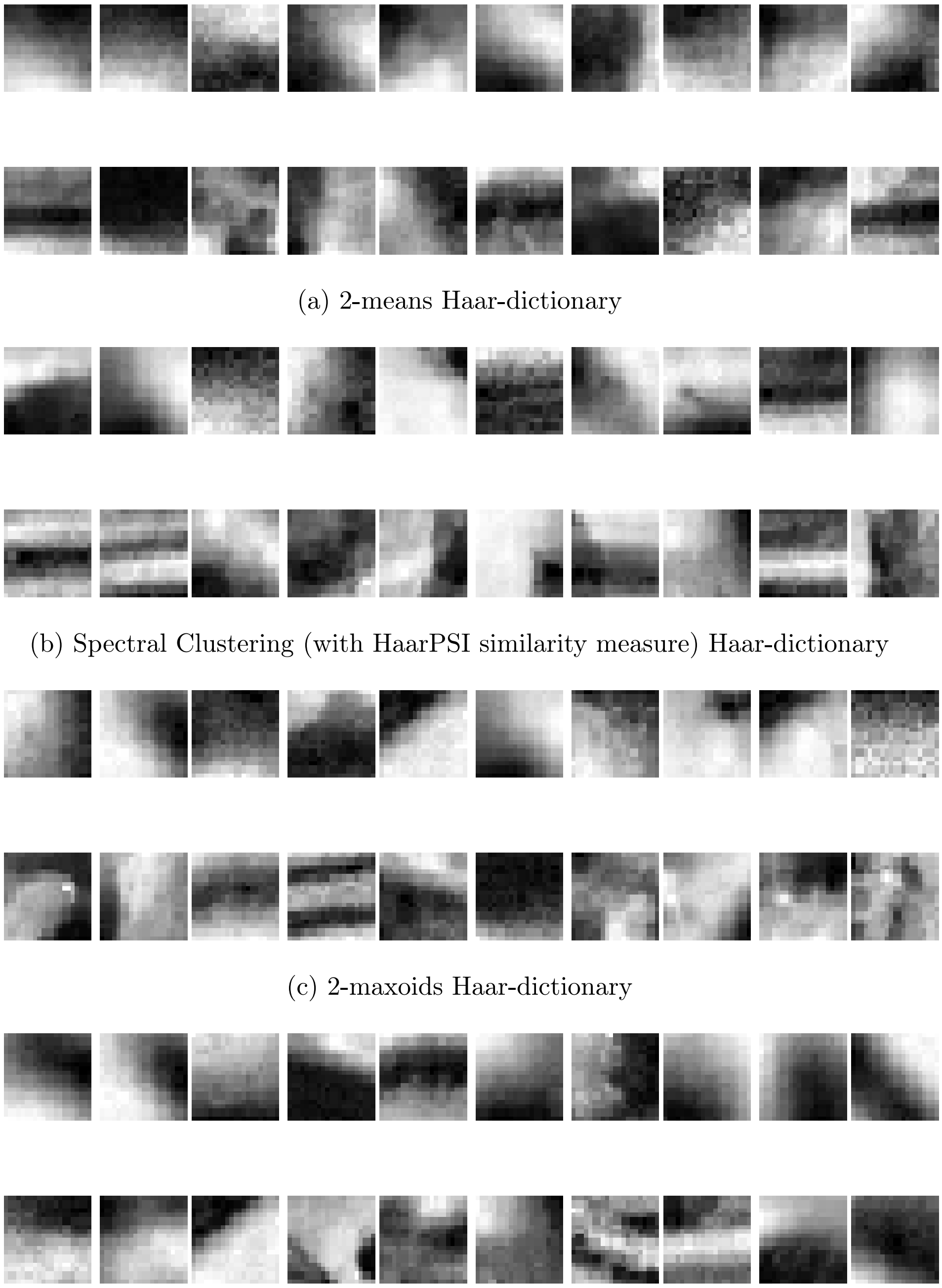
(a) Haar-dictionary

(b) K-SVD dictionary

Figure 7: Values of η_k defined in (3.1) for the Haar-dictionary (with 2-means clustering and FIFO tree visit) and K-SVD dictionary with 300 elements computed on the 32×32 patches of the `flowers-pool` image when used for the reconstruction of this same image.

This property could be used to obtain a sub-dictionary with similar reconstruction power by limiting the tree-depth; this would accelerate OMP. We remark that the atoms in this sub-dictionary associated to nodes closer to the root node would be stable to variations in the data-set given for example by noise, since they represent coarse-level features in the data.

Finally in Figure 8 we show the most used (i.e. ordered by decreasing values of η_k) dictionary patches for various dictionaries. It can be seen that when using 2-means our dictionary produces very smoothed out patches; this is due to the Haar-dictionary elements being difference of centroids of sibling clusters. The patches obtained instead using the 2-maxoids clustering have, as expected, sharper edges.



(a) 2-means Haar-dictionary

(b) Spectral Clustering (with HaarPSI similarity measure) Haar-dictionary

(c) 2-maxoids Haar-dictionary

(d) `K-SVD` dictionary

Figure 8: 20 dictionary atoms with highest η_k values (when reconstructing with sparsity 5) for various dictionaries. All the dictionaries were trained on 500 16×16 patches extracted from the `flowers_pool` image.

Acknowledgement

The authors gratefully acknowledge support by the German Research Foundation in the framework of the RTG 2088.

References

Aharon, M., Elad, M., and Bruckstein, A. (2006). K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Trans. Signal Process.*, 54(11):4311–4322.

Bauckhage, C. and Sifa, R. (2015). k-maxoids clustering. In *LWA*, pages 133–144.

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.*, 2(1):183–202.

Blumensath, T. and Davies, M. E. (2008). Iterative thresholding for sparse approximations. *J. Fourier Anal. Appl.*, 14(5-6):629–654.

Budinich, R. (2017). A region-based easy-path wavelet transform for sparse image representation. *Int. J. Wavelets Multiresolut. Inf. Process.*, 15(05):1750045.

Budinich, R. (2018). *Adaptive Multiscale Methods for Sparse Image Representation and Dictionary Learning*. PhD thesis, University of Göttingen.

Cai, J., Ji, H., Shen, Z., and Ye, G. (2014). Data-driven tight frame construction and image denoising. *Appl. Comput. Harmon. Anal.*, 37(1):89–105.

Chambolle, A. and Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *J. Math. Imaging Vis.*, 40(1):120–145.

Damelin, S. B. and Miller Jr, W. (2012). *The Mathematics of Signal Processing*, volume 48. Cambridge University Press.

Davies, G., Mallat, S., and Avellaneda, M. (1997). Adaptive greedy approximations. *Constr. Approx.*, 13(1):57–98.

Dragotti, P. L. and Lu, Y. M. (2014). On sparse representation in Fourier and local bases. *IEEE Trans. Inf. Theory*, 60(12):7888–7899.

Elad, M. (2010). *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer Publishing Company, Incorporated, 1st edition.

Liu, L., Ma, J., and Plonka, G. (2018). Sparse graph-regularized dictionary learning for suppressing random seismic noise. *Geophysics*, 83(3):V215–V231.

Liu, L., Plonka, G., and Ma, J. (2017). Seismic data interpolation and denoising by learning a tensor tight frame. *Inverse Problems*, 33(10):105011.

Lu, C., Shi, J., and Jia, J. (2013). Online robust dictionary learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 415–422.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696. ACM.

Mallat, S. (2008). *A Wavelet Tour of Signal Processing: The Sparse Way.* Academic Press.

Murtagh, F. (2007). The Haar wavelet transform of a dendrogram. *J. Classification*, 24(1):3–32.

Natarajan, B. K. (1995). Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–234.

Pati, Y. C., Rezaiifar, R., and Krishnaprasad, P. S. (1993). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE.

Reisenhofer, R., Bosse, S., Kutyniok, G., and Wiegand, T. (2018). A Haar wavelet-based perceptual similarity index for image quality assessment. *Signal Process., Image Commun.*, 61:33–43.

Rubner, Y., Tomasi, C., and Guibas, L. J. (2000). The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vis.*, 40(2):99–121.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905.

Tillmann, A. M. (2015). On the computational intractability of exact and approximate dictionary learning. *IEEE Signal Process. Lett.*, 22(1):45–49.

Tropp, J. A. (2004). Greed is good: Algorithmic results for sparse approximation. *IEEE Trans. Inf. Theory*, 50(10):2231–2242.

Yan, D., Huang, L., and Jordan, M. I. (2009). Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 907–916. ACM.

Yankelevsky, Y. and Elad, M. (2016). Dual graph regularized dictionary learning. *IEEE Trans. Signal Inf. Process. Netw.*, 2(4):611–624.

Zeng, X., Bian, W., Liu, W., Shen, J., and Tao, D. (2015). Dictionary pair learning on Grassmann manifolds for image denoising. *IEEE Trans. Image Process.*, 24(11):4556–4569.