Adaptive Multiscale Methods for Sparse Image Representation and Dictionary Learning

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Dissertation zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades

Doctor rerum naturalium

der Georg-August-Universität Göttingen

im Promotionsstudiengang *Mathematical Sciences*

der Georg-August University School of Science (GAUSS)

vorgelegt von

Renato Budinich

aus Trieste, Italien

Göttingen, 2018

Betreuungsausschuss

Gerlind Plonka-Hoch, Institute for Applied and Numerical Mathematics, University of Göttingen

Armin Iske, Department of Mathematics, University of Hamburg

Anita Schöbel, Institute for Applied and Numerical Mathematics, University of Göttingen

Russel Luke, Institute for Applied and Numerical Mathematics, University of Göttingen

Stephan Huckemann, Institute for Mathematical Stochastic, University of Göttingen

Dorothea Bahns, Department of Mathematics and Computer Science, University of Göttingen

Mitglieder der Prüfungskommission

Gerlind Plonka-Hoch, Institute for Applied and Numerical Mathematics, University of Göttingen

Armin Iske, Department of Mathematics, University of Hamburg

Tag der mündlichen Prüfung

23 November 2018

Foreword

Taking the PhD position at the Institut für Numerische und Angewandte Mathematik in Göttingen was a decision that involved many unknowns: I would have to separate from my family, friends and girlfriend, I didn't know German and I hadn't yet met my supervisor. Still, I had a good feeling about it and I can now say it turned out to be the most beneficial event yet to occur in my life. I met amazing people from all over the world that helped me discover my true vocations and the way I should pursue, giving me the most precious gift of friendship when it was most needed. They have made my stay abroad not only tolerable but exciting and I now look forward to the many more adventures that await us together in the future. This thesis is, I hope, the proof that I also couldn't have hoped for a better supervisor than Gerlind Plonka, who by example taught me the value of perseverance and hard work in research. Above all, Gerlind fostered my confidence in my own abilities by always giving me great freedom and being very encouraging and appreciative of my work, even when I thought it didn't hold so much value.

I want to dedicate this thesis to my whole "Vitalba" family, with whom I grew up and whom I sorely miss. In particular, I want to express my gratitude to my parents for having always been there like a trusted beacon, even (especially) when the sea was rough or the horizon covered in fog. I also want to dedicate this work to my friend Giulio Regeni, who was killed like a beast and taught me to live like a Man.

Contents

0 Introduction **7**

1 Preliminaries: a Review of some Data Analysis Tools **11**
1.1 Notations 11
1.2 Wavelet Transform 12
1.3 SVD 22
1.4 PCA 24
1.5 2DPCA 29
1.5.1 Properties of 2DPCA 32
1.5.2 Bilateral 2DPCA 33
1.6 Image Quality Assessment 34
1.7 Clustering data 36
1.7.1 K-Means 36
1.7.2 Graph Based Methods 39
1.7.3 Spectral Clustering 41
1.7.4 Felzenszwalb-Huttenlocher Clustering 43

2 Tree-based Haar-wavelet Generalizations **47**
2.1 Classical one-dimensional wavelet Haar transform 54
2.2 Adaptive dictionary 56
2.3 RBEPWT 56

3 The Region Based Easy Path Wavelet Transform **59**
3.1 General Framework 60
3.2 EPWT 62
3.3 RBEPWT 63
3.3.1 Path finding 64
3.3.2 Remarks on the Segmentation Method 68
3.3.3 Encoding Cost and Quality Measurement 69
3.4 Thresholding for a Region of Interest 70
3.5 Numerical Experiments 73

4 Sparse Coding and Dictionary Learning **83**
4.1 Sparse Coding 84
4.1.1 Matching Pursuit Algorithms 86
4.2 Dictionary Learning 90
4.2.1 The K-SVD method 91

4.2.2 Storage cost of a dictionary 93

5 Tree-based Dictionaries 95

5.1 Previous literature 95
5.2 A patch based model 101
5.3 Tree-based dictionary 103
5.3.1 The Algorithm 106
5.4 Numerical experiments 109

6 Conclusion 129

0 Introduction

A great class of problems in Signal Processing consists in finding a clever re-encoding of data such that in the new domain some features or properties therein contained are made apparent and thus easy to select for: for example a Fourier Transform will clearly show the components of a superposition of periodic signals. In the last 20-30 years there has been an increasing push to find domains in which natural signals exhibit *sparsity*, i.e. they can be approximated by a linear combination of only a small number of basis elements: this is an efficient representation because the original signal (or an approximation thereof) can be reconstructed from very few coefficients. This equates to searching for a domain where truncating the linear combination sum that adds up to the signal gives a good approximation, and is in this sense one of the oldest methodologies in applied Mathematics. The novelty here is in asking for sparsity of the coefficients instead of a small norm and explicitly looking for a basis that makes such representations possible and effective. In this context the concept of basis has been replaced by that of *dictionary*, a set of vectors (the dictionary *atoms*) that generate the space but are not necessarily linearly independent. The redundancy inherent in a dictionary allows one to obtain a good approximation of a given signal by projecting it on the low-dimensional space generated by a small number of dictionary atoms; this procedure is called *sparse coding* and, once a dictionary is chosen, can be done quite effectively through Pursuit algorithms.

However the more fundamental question is that of dictionary design: given a class of signals, which is a good choice of dictionary atoms that will enable an optimal sparse coding procedure? There are two possible ways to follow in the pursuit of an answer to this question: analytical and adaptive dictionaries. The former developed into the theory of *frames* which are usually generated by transformations of a kernel function and include for example curvelets, shearlets and Gabor frames. Given that in this case a closed mathematical expression is known for the dictionary atoms, research in this area usually focuses on proving convergence and stability results. On the other hand for adaptive dictionaries, which is what we'll be concerned with in this thesis, the atoms are computed numerically through some method that processes a large data-set of the signals one is interested in analyzing and reconstructing: this process is called *dictionary learning*. Specifically this is a task in unsupervised learning, conceptually much harder than problems in supervised learning for which incredibly high-performance machine learning methods have been developed in the last few years. An unsupervised learning method knows nothing about the input data and wishes to find some structure in it.

In this thesis we are particularly interested in sparse representation of natural images for lossy compression, and for this we propose two original adaptive methods. The first, termed Region Based Easy Path Wavelet Transform (RBEPWT) (Budinich (2017b), Budinich (2017a)) builds on previous work on the Easy Path Wavelet Transform (Plonka

(2009), Plonka et al. (2012), Plonka et al. (2013) and Plonka et al. (2011)). Both EPWT and RBEPWT work by vectorizing the image along some path with low resistance, i.e. such that the image gray-values along the path have a slow variation which allows for a better compressibility. Once the image is vectorized, one level of a standard one-dimensional wavelet transform is applied, the points decimated and the procedure iterated; in this regard RBEPWT and EPWT are identical, what they differ in is in how the path is found. The EPWT always greedily chooses the pixel closest in space and gray-value as next point in the path: this is a good heuristic with the disadvantage that, in order to re-encode the image from its coefficients, one needs to know the paths for all levels of the transform which thus have to be stored. The new RBEPWT addresses this issue by taking a different approach: the image is first segmented into regions of low variation in gray-values and subsequently a path-finding procedure is employed to vectorize each region independently. The path-finding is a heuristic rule for assigning to any set of points in space an ordering that depends only on their geometric configuration in relation to one another. In this way one has to store only the segmentation information instead of all the paths and thus the adaptivity cost is decreased in exchange for a non-optimal path: the RBEPWT still provides a good sparse representation of natural images, in the sense that if we enforce sparsity by keeping only the largest coefficients in this domain and apply the inverse transform, we obtain a good approximation of the original image. A particularity of the RBEPWT is that it easily allows to encode a Region of Interest with different quality than the rest of the image, using close to the minimal number of coefficients needed to do so. This application of the transform is obtained by observing that there is a tree structure among the coefficients that resembles that of the one-dimensional wavelet transform: this structure enables to determine the area of influence of each coefficient in the image.

Our second original method is the Haar-dict: this is not image-specific like the RBEPWT and fits more precisely in the general framework provided by the dictionary learning literature. The main idea of the method stems from another generalization of the one-dimensional wavelet transform, in particular the Haar wavelet: we use a clustering method such as K-means to recursively partition the samples into finer and finer clusters, obtaining a binary tree that, just as in the Haar transform, encodes some concept of resolution. Nodes that are higher represent coarser features common to large clusters of data, while the tree goes deeper in those regions of the data space where the data is organized into smaller clusters. The structure of the binary tree is completely data-dependent and this is where the adaptiveness of the method comes from. The dictionary atoms are then taken analogously to the wavelet coefficients as differences of averages of sibling nodes in the tree. Our algorithm is not iterative and usually performs slightly worse than previous methods in terms of quality of image reconstruction but is much faster since its computational cost depends only linearly on the number of patches.

The structure of this thesis is as follows: the first Chapter will be dedicated to reviewing various basic tools that we will need throughout the rest of the thesis, namely basic wavelet theory, singular value decomposition, principal component analysis (along with its less known two-dimensional variant) and data clustering methods. Furthermore we will prove one new result on the one-dimensional 2-means. In Chapter 2 we will propose a theoretical framework to capture the similarities between RBEPWT, Haar-dict and the one-dimensional Haar wavelet transform. In fact we argue that both our methods leverage the binary tree structure inherent in the Haar wavelet transforms, with the Haar-dict explicitly generalizing it and the RBEPWT modifying it to allow for a permutation step between levels of the tree (which corresponds to the re-ordering given by the path-finding procedure). Though the notations may not always be fluent, we think this attempt is important to grasp the fundamental similarities between these apparently very distant transforms. In Chapter 3 we will introduce the EPWT and RBEPWT with two different path-finding procedures and go through some numerical experiments in image compression. We will show how with RBEPWT a Region of Interest can easily be encoded with different quality than the rest of the image. In Chapter 4 we will review the sparse coding and dictionary learning problems, specifically the Pursuit algorithms and the state-of-the-art K-SVD method which has been applied successfully to many dictionary learning tasks. This works by iteratively updating each atom with rank-1 approximations provided by singular value decompositions; it is effective but computationally costly, with a quadratic dependency on the number of patches in the data-set. Finally in Chapter 5 we will describe in detail the Haar-dict method with all its variants, alongside a review of the literature that brought us to formulate it in these terms. We will further present various numerical results that test the method under different conditions comparing it to K-SVD and to RBEPWT.

1 Preliminaries: a Review of some Data Analysis Tools

1.1 Notations

We introduce here various notations that we will use throughout the text.

- Given a set S we will indicate with $|S|$ its cardinality and with 2^S its power set.
- Given sets $S \subset U$ we will indicate with $\mathbb{1}_S : U \to \{0, 1\}$ the characteristic function of S, i.e. $S = \mathbb{1}_S^{-1}(1)$.
- Given disjoint sets A and B we will indicate with $C = A \sqcup B$ their disjoint union: by this we mean that $C = A \cup B$, $A \cap B = \emptyset$ and $A, B \neq \emptyset$.
- Given a natural number $k \in \mathbb{N}$ we will indicate with $[k]$ the set of all natural numbers smaller than k, i.e. $[k] := \{0, 1, \ldots, k-1\}$.
- Given a real number α we will indicate with $\lfloor \alpha \rfloor$ the maximum integer smaller or equal to α.
- We will indicate with I_n the identity matrix of order n.
- Given a matrix X, we will indicate its i, j-th entry with X_{ij}, its i-th row with $X_{i\cdot}$ and its j-th column with $X_{\cdot j}$.
- Given a complex matrix A, we will indicate with A^H its conjugate transpose.
- Given vectors $v_1, \ldots, v_n$ in a vector space V, we will indicate with $< v_1, \ldots, v_n >$ the subspace generated by these vectors.
- Given a pre-Hilbert space V and two vectors $v, w \in V$ we will indicate their scalar product with $\langle v, w \rangle$.
- Given two vectors $v, w \in \ell^2(\mathbb{C})$ we will indicate with $\star$ their discrete convolution:

$$(v \star w)_k = \sum_{n \in \mathbb{Z}} v_n w_{k-n} \ .$$

- Given a function $f \in L^2(\mathbb{R})$ we will indicate with $\hat{f}$ its Fourier transform.

1.2 Wavelet Transform

The most fundamental mathematical tool of signal processing is the Fourier transform which isometrically transforms a signal (typically an L^2 function of time or space) into a function of frequency by a scalar product with complex sinusoidal functions. The choice of this basis, while natural from a mathematical point of view and enabling the proof of many important properties of the transform, is not optimal for the many naturally occurring signals which are only piece-wise regular: because of sinusoids being smooth and having non-compact support, many coefficients will be needed to account for the discontinuities of these signals. It is for this reason (alongside the cheap computation cost) that wavelet transform theory was a huge success in the 1990s: there are wavelet bases that are localized in both time/space and frequency domains and need only few coefficients to give good approximations of many natural signals like images and audio recordings. This allows for very efficient compression schemes based on simple thresholding: given an expansion of a signal in a wavelet basis, only the largest (i.e. presumably most important) coefficients are transmitted over a channel while the others are ignored. We will give a very brief review of wavelet theory concerned only with the results needed in the main Chapters of this thesis. We will be mostly following the exposition given in Damelin and Miller Jr (2012); for a much more in-depth look on the subject see for example Daubechies (1992) or the more modern Mallat (2008).

We start by giving the definition of a multi-resolution analysis, which is one of the basic concepts in wavelet theory. The scaled and translated versions of a function are used to generate a sequence of subspaces of $L^2(\mathbb{R})$. This will eventually allow for the design of bases of $L^2(\mathbb{R})$ which are localized both in time and frequency, ideal for efficient sparse representation of many types of naturally occurring signals.

Definition 1.2.1. Given a sequence of subspaces $\{V_j\}_{j\in\mathbb{Z}} \subset L^2(\mathbb{R})$ called *approximation spaces* and a function $\phi \in V_0$ normalized such that $\int_{\mathbb{R}} \phi = 1$, the couple $(\{V_j\}_{j\in\mathbb{Z}}, \phi)$ is a *multi-resolution analysis* if:

1. $\forall j \in \mathbb{Z} \quad V_j \subset V_{j+1}$
2. $L^2(\mathbb{R}) = \overline{\cup_{j\in\mathbb{Z}} V_j}$
3. $\cap_{j\in\mathbb{Z}} V_j = \{0\}$
4. $\forall k \in \mathbb{Z} \quad f \in V_0 \iff f(\cdot - k) \in V_0$
5. $f \in V_j \iff f(2\cdot) \in V_{j+1}$
6. $\{\phi(\cdot - k)\}_{k\in\mathbb{Z}}$ is an orthonormal basis for V_0

The function ϕ is called the *scaling function* or *father wavelet*, and the index j is referred to as the *level* or *scale*.

Remark 1. In some texts the level j is used in the opposite direction, i.e. $V_{j+1} \subset V_j$; we choose the convention above in order to be coherent with the concept of level in the

Haar dependency tree introduced in Chapter 2. With our definition a larger j means a finer approximation of $L^2(\mathbb{R})$ or equivalently a finer scale.

It is possible to weaken the definition by requiring that $\{\phi(\cdot - k)\}_{k\in\mathbb{Z}}$ be only a Riesz basis of V_0, i.e. that there exist constants A and B such that

$$\forall \omega \in [-\pi, \pi] \quad A \leq \sum_{k\in\mathbb{Z}} |\hat{\phi}(\omega + 2k\pi)|^2 \leq B \ .$$

It is in fact possible to orthogonalize a Riesz basis and obtain an orthonormal one. Define now $\phi_{j,k}(t) := 2^{j/2}\phi(2^j t - k)$; as a direct consequence of properties 4, 5 and 6 of Definition 1.2.1 one can see that $\{\phi_{j,k}\}_{k\in\mathbb{Z}}$ is an orthonormal basis for V_j for all $j \in \mathbb{Z}$. Since for example $V_0 \subset V_1$, we can write $\phi(\cdot)$ as a linear combination of the translates of $\sqrt{2}\phi(2\cdot)$, i.e. there exists $\{h_n\}_{n\in\mathbb{Z}}$ such that

$$\begin{aligned} \phi(t) &= \sqrt{2}\sum_{k\in\mathbb{Z}} h_k \phi(2t - k) && (1.2.2) \\ &= \sum_{k\in\mathbb{Z}} h_k \phi_{1,k}(t) \ . && (1.2.3) \end{aligned}$$

If we plug (1.2.2) into the definition of $\phi_{j,k}$ we obtain

$$\begin{aligned} \forall k \in \mathbb{Z} \quad \phi_{j,k}(t) &= 2^{j/2}\sqrt{2}\sum_{l\in\mathbb{Z}} h_l \phi(2(2^j t - k) - l) \\ &= \sum_{l\in\mathbb{Z}} h_l 2^{\frac{j+1}{2}} \phi(2^{j+1}t - (2k + l)) \\ &= \sum_{n\in\mathbb{Z}} h_{n-2k} 2^{\frac{j+1}{2}} \phi(2^{j+1}t - n) \\ &= \sum_{n\in\mathbb{Z}} h_{n-2k} \phi_{j+1,n}(t) \ , \end{aligned}$$

where $n = 2k + l$. Thus $\phi_{j,k}(t)$ can be obtained as the $2k$-th component of the discrete convolution of vectors $\{\phi_{j+1,n}(t)\}_{n\in\mathbb{Z}}$ and $\{h_{-n}\}_{n\in\mathbb{Z}}$,

$$\forall k \in \mathbb{Z} \quad \phi_{j,k}(t) = (\phi_{j+1,\cdot}(t) \star h_{-\cdot})_{2k} \ , \qquad (1.2.4)$$

which is known as the **dilation equation**. This operation can be seen as passing the vector $\{\phi_{j+1,n}(t)\}_{n\in\mathbb{Z}}$ into the filter that has $\{h_{-n}\}_{n\in\mathbb{Z}}$ as impulse response and subsequently downsampling by a factor of 2. Note that we could have obtained a similar equation directly from the fact that $V_j \subset V_{j+1}$, but this alone can't tell us that the coefficients h_k are always the same, independently of level j. This fact allows us to obtain the basis for all the levels from the father wavelet by recursively convoluting it (always with the same vector) and then downsampling this convolution by a factor of 2. We will later see that this same scheme can be applied to the wavelet coefficients themselves, allowing for a very efficient computation of the wavelet transform of a signal. The vector of coefficients $\{h_n\}_{n\in\mathbb{Z}}$ has several properties, which we state and prove in the following Lemma.

Lemma 1.2.5. *Let* $\{h_n\}_{n\in\mathbb{Z}}$ *be the vector such that* (1.2.4) *holds. Then we have:*

1. $\sum_{k\in\mathbb{Z}} h_k = \sqrt{2}$,
2. $||h||_{\ell^2} = 1$,
3. $\forall m \in \mathbb{Z} \quad \sum_{k\in\mathbb{Z}} h_k \overline{h_{k-2m}} = \delta_{0m}$.

Proof. The first property follows by integrating both sides of equation (1.2.2) over the whole real line and applying a simple linear substitution on the right hand side. The third property follows from (1.2.3), the dilation equation and the orthonormality of $\phi_{j,\cdot}$:

$$\begin{aligned}\delta_{0m} &= \langle \phi_{0,0}, \phi_{0,m} \rangle \\ &= \sum_{k\in\mathbb{Z}} h_k \sum_{j\in\mathbb{Z}} \overline{h_{j-2m}} \langle \phi_{1,k}, \overline{\phi_{1,j}} \rangle \\ &= \sum_{k\in\mathbb{Z}} h_k \overline{h_{k-2m}} \ .\end{aligned}$$

The second property is a particular case of the third, obtained when $m = 0$. ■

Example 1 (Haar Wavelets)**.** Let $\phi(t) = \mathbb{1}_{[0,1)}(t)$ and

$$V_j = \{ \text{ square integrable functions that are constant on } [k2^{-j}, (k+1)2^{-j}] \text{ for all } k \in \mathbb{Z}\} \ .$$

Then $(\{V_j\}_{j\in\mathbb{Z}}, \phi)$ is a multi-resolution analysis. It's straightforward to see that the dilation equation in this case is

$$\phi_{j,k}(t) = \frac{1}{\sqrt{2}} \left(\phi_{j+1,2k}(t) + \phi_{j+1,2k+1}(t)\right) \ ,$$

i.e.

$$h_n = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } n = 0, 1 \\ 0 & \text{otherwise} \ . \end{cases} \tag{1.2.6}$$

◇

We now wish to introduce the orthogonal space to V_j in V_{j+1}: we thus define

$$W_j := \{f \in V_{j+1} \mid \forall g \in V_j \ \langle f, g\rangle = 0\}$$

and have $V_{j+1} = V_j \oplus W_j$; we will call W_j the *wavelet space* of level j. We now would like to find $\psi \in W_0$ such that:

1. it has norm 1,

2. its translates $\{\psi(\cdot - k)\}_{k\in\mathbb{Z}}$ form an orthonormal basis of W_0,
3. it is orthogonal to the family $\{\phi_{0k}\}_{k\in\mathbb{Z}}$.

Since $W_0 \subset V_1$ it will surely be possible to write

$$\psi(t) = \sqrt{2}\sum_{k\in\mathbb{Z}} g_k \phi(2t-k) \ . \tag{1.2.7}$$

It's straightforward to see that the conditions we require on ψ translate to the following conditions on $\{g_n\}_{n\in\mathbb{Z}}$ (analogously to Lemma (1.2.5)):

Lemma 1.2.8. *If ψ satisfies the conditions above then the following must hold*

1. $||g_k||_{\ell^2} = 1$
2. $\forall m \in \mathbb{Z} \quad \sum_{k\in\mathbb{Z}} g_k \overline{g_{k-2m}} = \delta_{0m}$
3. $\forall m \in \mathbb{Z} \quad \sum_{k\in\mathbb{Z}} h_k \overline{g_{k-2m}} = 0$

One possible solution to these conditions is choosing $g_n := (-1)^n \overline{h_{-1-n}}$; we will make this choice and call ψ the **mother wavelet**.

Example 2 (Haar Wavelets)**.** For the Haar wavelets we have

$$g_n = \begin{cases} \frac{(-1)^n}{\sqrt{2}} & \text{if } n = -2, -1 \\ 0 & \text{otherwise} \ ; \end{cases} \tag{1.2.9}$$

then

$$\begin{aligned} \psi(t) &= \phi(2t+2) - \phi(2t+1) \\ &= \mathbb{1}_{[-1,-\frac{1}{2})}(t) - \mathbb{1}_{[-\frac{1}{2},0)}(t) \ . \end{aligned}$$

◇

Define now $\psi_{j,k}(t) := 2^{j/2}\psi(2^j t - k)$; similarly as done for $\phi_{j,k}$, we can plug (1.2.7) into the definition of $\psi_{j,k}$ and obtain the **wavelet equation**:

$$\forall k \in \mathbb{Z} \quad \psi_{j,k} = (\phi_{j+1,\cdot} \star g_{-\cdot})_{2k} \ . \tag{1.2.10}$$

We can write the dilation equation (1.2.4) and the wavelet equation (1.2.10) as a system in matrix form:

$$\left[\begin{array}{c} \phi_{j,\cdot} \\ \hline \psi_{j,\cdot} \end{array}\right] = \Omega\left[\phi_{j+1,\cdot}\right] \tag{1.2.11}$$

where

$$\left[\begin{array}{c}\phi_{j,\cdot}\\ \hline \psi_{j,\cdot}\end{array}\right] =: \left[\begin{array}{c}\vdots\\ \phi_{j,k-1}\\ \phi_{j,k}\\ \phi_{j,k+1}\\ \vdots\\ \hline \vdots\\ \psi_{j,k-1}\\ \psi_{j,k}\\ \psi_{j,k+1}\\ \vdots\end{array}\right] , \quad [\phi_{j+1,\cdot}] =: \begin{bmatrix}\vdots\\ \phi_{j+1,k-1}\\ \phi_{j+1,k}\\ \phi_{j+1,k+1}\\ \vdots\end{bmatrix} , \quad \Omega =: \left[\begin{array}{c}\Omega_h\\ \hline \Omega_g\end{array}\right] ,$$

and the infinite matrices Ω_h and Ω_g have $\{h_{n-2k}\}_{n\in\mathbb{Z}}$ and $\{g_{n-2k}\}_{n\in\mathbb{Z}}$ as k-th row,

$$\Omega_h := \begin{bmatrix}\vdots\\ \{h_{n-2(k-1)}\}_{n\in\mathbb{Z}}\\ \{h_{n-2k}\}_{n\in\mathbb{Z}}\\ \{h_{n-2(k+1)}\}_{n\in\mathbb{Z}}\\ \vdots\end{bmatrix} , \quad \Omega_g := \begin{bmatrix}\vdots\\ \{g_{n-2(k-1)}\}_{n\in\mathbb{Z}}\\ \{g_{n-2k}\}_{n\in\mathbb{Z}}\\ \{g_{n-2(k+1)}\}_{n\in\mathbb{Z}}\\ \vdots\end{bmatrix} .$$

For a proof of the following see Damelin and Miller Jr (2012).

Proposition 1.2.12. *The infinite matrix Ω is unitary.*

We can thus easily invert (1.2.11) and obtain

$$[\Omega_h^H \mid \Omega_g^H] \left[\begin{array}{c}\phi_{j,\cdot}\\ \hline \psi_{j,\cdot}\end{array}\right] = [\phi_{j+1,\cdot}] \ , \tag{1.2.13}$$

where Ω_h^H has $\{\overline{h_{n-2k}}\}_{n\in\mathbb{Z}}$ as k-th column

$$\Omega_h^H = [\ldots \mid \{\overline{h_{n-2(k-1)}}\}_{n\in\mathbb{Z}} \mid \{\overline{h_{n-2k}}\}_{n\in\mathbb{Z}} \mid \{\overline{h_{n-2(k+1)}}\}_{n\in\mathbb{Z}} \mid \ldots]$$

or equivalently $\{\overline{h_{l-2n}}\}_{n\in\mathbb{Z}}$ as l-th row:

$$\Omega_h^H = \begin{bmatrix}\vdots\\ \{\overline{h_{(l-1)-2n}}\}_{n\in\mathbb{Z}}\\ \{\overline{h_{l-2n}}\}_{n\in\mathbb{Z}}\\ \{\overline{h_{(l+1)-2n}}\}_{n\in\mathbb{Z}}\\ \vdots\end{bmatrix} ,$$

and similarly

$$\Omega_g^H = \begin{bmatrix} \vdots \\ \{\overline{g_{(l-1)-2n}}\}_{n\in\mathbb{Z}} \\ \{\overline{g_{l-2n}}\}_{n\in\mathbb{Z}} \\ \{\overline{g_{(l+1)-2n}}\}_{n\in\mathbb{Z}} \\ \vdots \end{bmatrix} .$$

We can thus expand (1.2.13) and obtain the **wavelet inverse equation**:

$$\forall k \in \mathbb{Z} \quad \phi_{j+1,k} = \sum_{n\in\mathbb{Z}} \left[\overline{h_{k-2n}}\phi_{j,n} + \overline{g_{k-2n}}\psi_{j,n} \right] . \tag{1.2.14}$$

Thus we proved the following:

Proposition 1.2.15. *An orthonormal basis for W_j is given by $\{\psi_{j,k}\}_{k\in\mathbb{Z}}$; an alternative orthonormal basis for V_{j+1} is then given by $\{\phi_{j,k}, \psi_{j,k'}\}_{k,k'\in\mathbb{Z}}$.*

We now obtain the main result for a multi-resolution analysis:

Theorem 1.2.16. *For any $j \in \mathbb{N}$ we have:*

$$L^2(\mathbb{R}) = V_j \oplus \sum_{k=-\infty}^{j} W_k = V_j \oplus W_j \oplus W_{j-1} \oplus \dots .$$

This means that any $f \in L^2(\mathbb{R})$ may be written as

$$f = f_j + \sum_{k=-\infty}^{j} w_k$$

for some $f_j \in V_j$ and for $w_k \in W_k$.

The theorem implies that given a function $f \in L^2(\mathbb{R})$ we can write it in the basis given by the translations and dilations of the mother and father wavelet. Consider now the projection of f onto V_j, $\mathcal{P}_j(f)$. We can decompose V_j into a sum of wavelet spaces of decreasing levels and an approximation space for the lowest level:

$$\forall n \in \mathbb{N} \quad V_j = W_{j-1} \oplus V_{j-1} = W_{j-1} \oplus W_{j-2} \oplus V_{j-2} = \dots = W_j \oplus \dots \oplus W_{j-n} \oplus V_{j-n} . \tag{1.2.17}$$

Associated to each of these decompositions of V_j we have a different expansion of $\mathcal{P}_j(f)$, for example from $V_j = W_{j-1} \oplus V_{j-1}$ we can write:

$$\mathcal{P}_j(f) = \sum_{k\in\mathbb{Z}} \phi_{j,k} \langle \mathcal{P}_j(f), \phi_{j,k} \rangle$$

$$= \sum_{k\in\mathbb{Z}} \phi_{j-1,k}\langle \mathcal{P}_j(f), \phi_{j-1,k}\rangle + \sum_{k\in\mathbb{Z}} \psi_{j-1,k}\langle \mathcal{P}_j(f), \psi_{j-1,k}\rangle \ . \tag{1.2.18}$$

A fundamental result of wavelet theory is that we can use the dilation and wavelet equation to establish a relation between any two such subsequent expansions, where the coefficients at one level are determined by convolution and downsampling of the coefficients at the previous level with the low and high pass filters $\{\overline{h_{-n}}\}_{n\in\mathbb{Z}}$ and $\{\overline{g_{-n}}\}_{n\in\mathbb{Z}}$. In fact if we define for any level $j \in \mathbb{Z}$ the so called **approximation** and **detail** coefficients

$$\begin{aligned} \forall k \in \mathbb{Z} \quad a_{j,k} &:= \langle \mathcal{P}_j(f), \phi_{j,k}\rangle \\ \forall k \in \mathbb{Z} \quad d_{j,k} &:= \langle \mathcal{P}_j(f), \psi_{j,k}\rangle \ , \end{aligned} \tag{1.2.19}$$

by applying (1.2.4) we obtain

$$\begin{aligned} \forall k \in \mathbb{Z} \quad a_{j,k} &= \int_{\mathbb{R}} \mathcal{P}_j(f)(t)\overline{\phi_{j,k}}(t)\ dt \\ &= \int_{\mathbb{R}} \mathcal{P}_j(f)(t) \sum_{l\in\mathbb{Z}} \overline{h_{l-2k}\phi_{j+1,l}(t)}\ dt \\ &= \sum_{l\in\mathbb{Z}} \overline{h_{l-2k}} \int_{\mathbb{R}} \mathcal{P}_j(f)(t)\overline{\phi_{j+1,l}(t)}\ dt \\ &= \sum_{l\in\mathbb{Z}} \overline{h_{l-2k}} a_{j+1,l} \\ &= (a_{j+1,\cdot} \star \overline{h_{-\cdot}})_{2k} \end{aligned} \tag{1.2.20}$$

and similarly by applying (1.2.10)

$$\forall k \in \mathbb{Z} \quad d_{j,k} = (a_{j+1,\cdot} \star \overline{g_{-\cdot}})_{2k} \ . \tag{1.2.21}$$

These are known as the **analysis formulas** and can be put in matrix form using the matrix Ω

$$\begin{bmatrix} a_{j,\cdot} \\ \hline d_{j,\cdot} \end{bmatrix} = \begin{bmatrix} \overline{\Omega_h} \\ \hline \overline{\Omega_g} \end{bmatrix} [a_{j+1,\cdot}] \ , \tag{1.2.22}$$

which we can invert to obtain

$$\left[\Omega_h^T \mid \Omega_g^T\right] = \begin{bmatrix} a_{j,\cdot} \\ \hline d_{j,\cdot} \end{bmatrix} = [a_{j+1,\cdot}] \ , \tag{1.2.23}$$

or equivalently

$$\forall k \in \mathbb{Z} \quad a_{j+1,k} = \sum_{n\in\mathbb{Z}} [\, h_{k-2n}a_{j,n} + g_{k-2n}d_{j,n}] \tag{1.2.24}$$

which are known as the **synthesis formulas**.

Example 3 (Haar analysis and synthesis formulas)**.** In the case of the Haar wavelets the analysis formulas are

$$\forall k \in \mathbb{Z} \quad \begin{aligned} a_{j,k} &= \frac{1}{\sqrt{2}}\big(a_{j+1,2k} + a_{j+1,2k+1}\big) \\ d_{j,k} &= \frac{1}{\sqrt{2}}\big(a_{j+1,2k} - a_{j+1,2k+1}\big) \ , \end{aligned} \tag{1.2.25}$$

and the synthesis ones

$$\forall k \in \mathbb{Z} \quad a_{j+1,k} = \begin{cases} \frac{1}{\sqrt{2}}(a_{j,n} + d_{j,n}) & \text{if } k = 2n \text{ for some } n \in \mathbb{N} \\ \frac{1}{\sqrt{2}}(a_{j,n} - d_{j,n}) & \text{if } k = 2n+1 \text{ for some } n \in \mathbb{N} \ . \end{cases} \tag{1.2.26}$$

◇

We now wish to illustrate the typical real world use of wavelet theory: suppose we have only a finite set of discrete samples $\{f_n\}_{n=0}^{N}$ of the function $f \in L^2(\mathbb{R})$ and suppose that ϕ and ψ have compact support $[0, \Phi]$ for some $\Phi \in \mathbb{R}$. This means that the vectors $\{h_n\}_{n\in\mathbb{Z}}$ and $\{g_n\}_{n\in\mathbb{Z}}$ will have only a finite number of non-zero elements; furthermore the matrices Ω_h and Ω_g will be finite. Suppose for simplicity that $N = 2^J$ for some $J \in \mathbb{N}$. In committing what is sometimes termed the *wavelet crime* we will identify the approximation coefficients at the highest level with the samples themselves, i.e. $a_{J,n} = f_n$. This amounts to using f_n as approximation for the scalar product $\langle \phi_{J,n}, f\rangle$ which is reasonable if J is large enough since $\phi_{J,n}$ has support $[n, n + \frac{\Phi}{2^J}]$ and has integral 1. We can then compute an alternative representation of the samples by using the analysis formulas according to the following scheme

$$\begin{tikzcd}
a_{J,\cdot} \arrow[r, "\downarrow 2 \circ \star\overline{h_{-\cdot}}"] \arrow[dr, "\downarrow 2 \circ \star\overline{g_{-\cdot}}"] & a_{J-1,\cdot} \arrow[r, "\downarrow 2 \circ \star\overline{h_{-\cdot}}"] \arrow[dr, "\downarrow 2 \circ \star\overline{g_{-\cdot}}"] & a_{J-2,\cdot} & \cdots \arrow[r, "\downarrow 2 \circ \star\overline{h_{-\cdot}}"] \arrow[dr, "\downarrow 2 \circ \star\overline{g_{-\cdot}}"] & a_{J-L,\cdot} \\
 & d_{J-1,\cdot} & d_{J-2,\cdot} & & d_{J-L,\cdot}
\end{tikzcd} \tag{1.2.27}$$

where $L \in \mathbb{N}, L \leq J$ is the **level** of the transform and by $\downarrow 2 \circ \star\overline{h_{-\cdot}}$ (respectively $\downarrow 2 \circ \star\overline{g_{-\cdot}}$) we intend the application of analysis equation (1.2.20) (respectively (1.2.21)) which is a convolution with the low-pass (high-pass) filter followed by a down-sampling of scale 2. These operations amount to a linear invertible operator that transforms the original representation

$$[a_{J,0}, \ldots, a_{J,N}]$$

into

$$[a_{J-L,\cdot} \mid d_{J-L,\cdot} \mid d_{J-L+1,\cdot} \mid \; \ldots \; \mid d_{J-1,\cdot}] \; . \tag{1.2.28}$$

These representations are equivalent because using formula (1.2.24) one can obtain (reconstruct) $a_{J-L+1,\cdot}$ from $a_{J-L,\cdot}$ and $d_{J-L,\cdot}$, then $a_{J-L+2,\cdot}$ from $a_{J-L+1,\cdot}$ and $d_{J-L+1,\cdot}$ and so on; this amounts to reading the scheme (1.2.27) from right to left.

Example 4 (Haar Transform of a signal)**.** Suppose we are given the discrete vector of signal values

$$z = a_3 = \sqrt{2}\; [2, 6, 0, 6, 3, 6, 1, 0] \in \mathbb{R}^8$$

and we wish to compute its coefficients under a 3-level Haar wavelet transform. By applying (1.2.25) recursively we obtain

$$\begin{aligned} a_2 &= [8, 6, 9, 1] \\ d_2 &= [-4, -6, -3, 1] \\ \\ a_1 &= \left[\frac{14}{\sqrt{2}}, \frac{10}{\sqrt{2}}\right] \\ d_1 &= \left[-\frac{2}{\sqrt{2}}, \frac{8}{\sqrt{2}}\right] \\ \\ a_0 &= [12] \\ d_0 &= [2] \; . \end{aligned}$$

Thus the representation of z under the transform is

$$[\; a_0 \mid d_0 \mid d_1 \mid d_2 \;] = \left[12 \middle| 2 \middle| \frac{2}{\sqrt{2}}, \frac{8}{\sqrt{2}} \middle| -4, -6, -3, 1\right] \; .$$

◇

Computing the wavelet transform (1.2.28) using the convolutions with $\{\overline{h_{-n}}\}_{n\in\mathbb{N}}$ and $\{\overline{g_{-n}}\}_{n\in\mathbb{N}}$ as depicted in scheme (1.2.27) is very cheap computationally (compared to a direct computation of all the inner products with the $\phi_{j,n}$ and $\psi_{j,n}$): suppose $\{\overline{h_{-n}}\}_{n\in\mathbb{N}}$ and $\{\overline{g_{-n}}\}_{n\in\mathbb{N}}$ have respectively Q and R non-zero elements and the signal $a_{J,\cdot}$ consists of N real samples. Then, ignoring border effects due to the finiteness of the signal support, to compute each entry of $a_{J-1,\cdot}$ Q multiplications and $Q-1$ sums are needed, and similarly for each entry of $d_{J-1,\cdot}$ R multiplications and $R-1$ sums; the total complexity of computing $a_{J-1,\cdot}$ and $d_{J-1,\cdot}$ is thus $\mathcal{O}(N(Q+R))$. The total complexity of computing

L levels of the transform is then

$$\begin{aligned}\sum_{l=J}^{J-L} \mathcal{O}(\frac{N}{2^{J-l}}(Q+R)) &= \mathcal{O}(N(Q+R)\sum_{l=J}^{J-L} 2^{-(J-l)}) \\ &= \mathcal{O}(N(Q+R)\sum_{k=0}^{L} 2^{-k}) \\ &= \mathcal{O}((2-2^{-L})N(Q+R)) \ ,\end{aligned}$$

where $k = J - l$.

We started from a multiresolution analysis $(\{V_j\}_{j\in\mathbb{Z}}, \phi)$ and from this derived h, g and ψ. However we saw how all properties of ϕ and ψ can be formulated as properties of h and g; typically one specifies the filters and from this the multiresolution analysis, ϕ and ψ are determined. This is possible thanks to the following Theorem (see Mallat (2008) Theorem 7.2 at page 271 for a proof):

Theorem 1.2.29. *If $\hat{h}(\omega)$ is 2π-periodic, continuously differentiable in 0, such that $\hat{h}(0) = \sqrt{2}$ and*

$$\forall \omega \in \mathbb{R} \quad |\hat{h}(\omega)|^2 + |\hat{h}(\omega+\pi)|^2 = 2 \ ,$$

then

$$\hat{\phi}(\omega) := \prod_{p\in\mathbb{N}} \frac{\hat{h}(2^{-p}\omega)}{\sqrt{2}}$$

is the Fourier transform of a father wavelet $\phi \in L^2(\mathbb{R})$. Conversely, the above conditions on $\hat{h}$ are always satisfied when ϕ is a father wavelet and in this case h is called a conjugate mirror filter.

Finally we wish to mention the trade-off one has to deal with when designing wavelets to use for sparse representation (i.e. few large coefficients) of natural signals: this is due to searching for a mother wavelet that has both a high number of vanishing moments and a small support. A function ψ has p vanishing moments if

$$\forall k \in [p] \quad \langle \cdot^k, \psi(\cdot)\rangle = 0 \ ,$$

i.e. ψ is orthogonal to all polynomials of degree at most p. This is a desirable property because it ensures that, in areas where the signal f is in C^n with $n \geq p$, the wavelet coefficients will have small amplitude since $\langle \psi, f\rangle = \langle \psi, f - \tilde{f}_{p-1}\rangle$ where $\tilde{f}_{p-1}$ is the Taylor polynomial of degree $p-1$ for f. On the other hand, at a point where the signal f has a singularity one would wish for wavelets with small support, so that the singularity will be included in the domain of only few wavelet basis functions. So, depending on the density of the singularities, one would wish for wavelets with many vanishing moments or small support.

These two requirements, at first sight independent, turn out to both be dependent on properties of h: for more precise statements and proofs of the following see Section 7.2 of Mallat (2008):

Theorem 1.2.30. *If ψ is a mother wavelet and h a conjugate mirror filter, then*

1. *ψ has p vanishing moments $\Leftrightarrow$ $\hat{h}(\omega)$ and its first $p-1$ derivatives are zero at $\omega = \pi$.*

2. *ψ has compact support $\Leftrightarrow$ h has compact support (i.e. it represents a Finite Impulse Response filter).*

3. *If ψ has p vanishing moments then h has at least $2p$ non-zero coefficients.*

These remarks are at the base of the construction due to Daubechies (see Daubechies (1988) and Daubechies (1992)) of wavelets that are optimal with respect to the last condition in the Theorem: they have p vanishing moments and the corresponding conjugate filters have exactly $2p$ zeroes. These wavelets in fact perform very well for sparse representation of natural images, and we will use them for the Region Based Easy Path Wavelet Transform in Chapter 3.

1.3 SVD

Given an $m \times n$ complex matrix A it always holds that the $n \times n$ matrix $A^H A$ is positive semi-definite, since $x^H A^H A x = ||Ax||_2^2 \geq 0$ for any $x \in \mathbb{C}^n$. Therefore the eigenvalues λ_i of $A^H A$ are non-negative and we can suppose $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq 0$. The values $\sigma_k := \sqrt{\lambda_k}$ are called the *singular values* of A. For a proof of the following see for example Stoer and Bulirsch (2013) (Theorem 6.4.10 p. 383).

Theorem 1.3.1. *Let A be a complex $m \times n$ matrix; then there exists a unitary $m \times m$ matrix U, a unitary $n \times n$ matrix V and an $m \times n$ diagonal matrix $\Sigma = diag(\sigma_1, \ldots, \sigma_n)$ such that*

$$A = U \Sigma V^H \tag{1.3.2}$$

and this is called the singular value decomposition *of A. Furthermore, if A has rank r then $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0 = \sigma_{r+1} = \ldots = \sigma_n$.*

From the theorem it follows that A^H has the same singular values as A. The matrices U and V being unitary also implies that $U^H A A^H U = U^H A V V^H A^H U = \Sigma \Sigma^H$ and analogously $V^H A^H A V = \Sigma^H \Sigma$. These are eigendecompositions and we can thus deduce that the columns of U are the m orthonormal eigenvectors of the $m \times m$ matrix AA^H, while the columns of V are the n orthonormal eigenvectors of the $n \times n$ matrix $A^H A$. The columns of U and V are called *left-singular* and *right-singular vectors* of A, respectively. The SVD of a matrix has several interesting properties, some of which we prove here:

Theorem 1.3.3. *Given an $m \times n$ matrix A with SVD decomposition $A = U \Sigma V^H$, the following hold:*

1. *range*$(A) =< U_{\cdot 1}, \ldots, U_{\cdot r} >$,
2. *ker*$(A) =< V_{\cdot r+1}, \ldots, V_{\cdot n} >$,
3. *if* $A = A^H$ *the singular values of* A *are the absolute values of its eigenvalues,*
4. $||A||_2 = \sigma_1$,
5. $||A||_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_r^2}$,
6. *if* $m = n$ *then* $|det(A)| = \prod_{i=1}^{m} \sigma_i$.

Proof.

(1) This is a consequence of V being non-singular and

$$\{\Sigma x \mid x \in \mathbb{R}^n\} = \{x \in \mathbb{R}^m \mid x_{r+1} = \ldots = x_m = 0\} \ .$$

(2) First of all note that since U is a non-singular matrix $\ker(A) = \ker(\Sigma V^H)$. Furthermore given the orthonormality of the columns of V for any $w \in \mathbb{C}^n$ we can write $w = a_1 V_{\cdot 1} + \ldots + a_n V_{\cdot n}$ and

$$\begin{aligned} \Sigma V^H w &= \Sigma \begin{bmatrix} (V_{\cdot 1})^H w \\ \vdots \\ (V_{\cdot n})^H w \end{bmatrix} \\ &= \Sigma \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \\ &= \begin{bmatrix} \sigma_1 a_1 \\ \vdots \\ \sigma_r a_r \\ 0 \\ \vdots \\ 0 \end{bmatrix} \ . \end{aligned}$$

Thus $w \in \ker(\Sigma V^H) \Leftrightarrow a_1 = \ldots = a_r = 0$ i.e. if and only if $w \in < V_{\cdot r+1}, \ldots, V_{\cdot n} >$.

(3) Since A is hermitian we can write the eigendecomposition $A = Q\Lambda Q^H = Q|\Lambda|\text{sgn}(\Lambda)Q^H$ where Q is a unitary matrix and by $|\Lambda|$ and $\text{sgn}(\Lambda)$ we mean the matrix that has as entries the absolute values and the signs of the entries of Λ respectively. Since $\text{sgn}(\Lambda)Q^H$ is also unitary, this is actually an SVD of A.

(4)-(6) These follow from the invariance under unitary transformations of the determinant, the 2-norm and the Frobenius norm. ■

The most common use of the SVD is for low rank approximation of a matrix. Note that, by writing Σ as a sum of rank one matrices $\mathrm{diag}(0,\ldots,0,\sigma_j,0,\ldots,0)$, we can rewrite (1.3.2) as

$$A = \sum_{j=1}^{r} \sigma_j U_{\cdot j} (V^H)_{j\cdot} \tag{1.3.4}$$

$$= \sum_{j=1}^{r} \sigma_j U_{\cdot j} (V_{\cdot j})^H \ , \tag{1.3.5}$$

which is a sum of rank one matrices. A fundamental result is that if we define, for $k \leq r$, A_k as the truncated sum

$$A_k = \sum_{j=1}^{k} \sigma_j U_{\cdot j} (V_{\cdot j})^H \ , \tag{1.3.6}$$

this gives the optimal low rank approximation of A under both the 2-norm and Frobenius norm; note that we're always supposing $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n > 0$. For a proof of the following (for the spectral norm) see for example Lecture 5 of Trefethen and Bau III (1997).

Eckart-Young Theorem 1.3.7. *For any* $0 \leq k \leq r$ *we have*

$$||A - A_k||_2 = \min_{\substack{B \in \mathbb{C}^{m \times n} \\ rk(B) \leq k}} ||A - B||_2 = \sigma_{k+1} \tag{1.3.8}$$

$$||A - A_k||_F = \min_{\substack{B \in \mathbb{C}^{m \times n} \\ rk(B) \leq k}} ||A - B||_F = \sqrt{\sigma_{k+1}^2 + \ldots + \sigma_r^2} \ . \tag{1.3.9}$$

Finally we wish to remark that the classic method to compute the SVD of an $m \times n$ matrix is due to Golub and Kahan and has complexity $O(mn^2)$ (see for example Trefethen and Bau III (1997)).

1.4 PCA

Principal component analysis (PCA) is the statiscal procedure of finding an orthonormal basis of the space the data resides in such that the data written in this new basis has the largest possible variance along the first and subsequent basis elements. It has a deep connection with SVD which we will explore towards the end of this section. Formally, suppose we have a matrix $X \in \mathbb{R}^{N \times n}$ where each row represents the realization of a real random vector of dimension n: each column thus represents a different feature and each row a different realization or experiment. We can always suppose that the columns of X are centered, i.e. have sample mean 0. We now look for an orthonormal basis

$v^1, \dots, v^n \in \mathbb{R}^n$ such that

$$\begin{aligned}
v^1 &= \operatorname*{arg\,max}_{||w||=1} \mathrm{Var}[Xw] \\
v^2 &= \operatorname*{arg\,max}_{\substack{||w||=1 \\ w^T v^1=0}} \mathrm{Var}[Xw] \\
&\vdots \\
v^n &= \operatorname*{arg\,max}_{\substack{||w||=1 \\ w^T v^1 = w^T v^2 = \ldots = w^T v^{n-1}=0}} \mathrm{Var}[Xw] \ ,
\end{aligned} \tag{1.4.1}$$

i.e., we want for v^1 to be such that the rows of X projected onto it have maximum variance; we then ask the same for v^2 with the added constraint for it to be orthogonal to v^1 and so on. Such vectors are called *principal components* of the data in X. We can then define the matrix $V = [v^1|\ldots|v^n]$ with the new basis elements as columns and define $Z = XV$; the rows of Z are the data written in the new basis and are also called *feature vectors* or *scores*. See Figure (1.1). Note that since we are supposing the columns of X to be centered, so are the columns of Z:

$$\begin{aligned}
\overline{Z_{\cdot j}} &= \frac{1}{N}\sum_{i=1}^{N} Z_{ij} \\
&= \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{n} X_{ik} V_{kj} \\
&= \frac{1}{N}\sum_{k=1}^{n}\left(\sum_{i=1}^{N} X_{ik}\right) V_{kj} \\
&= 0 \ .
\end{aligned}$$

The optimal basis is actually given by the orthogonal set of eigenvectors of the *covariance matrix* $X^T X$. To prove this we need a result on the so-called Rayleigh quotient: given a hermitian matrix M and a nonzero vector x, the *Rayleigh quotient* of M over x is

$$R(M,x) = \frac{x^H M x}{x^H x} \ . \tag{1.4.2}$$

Note that:

- for any nonzero constant c, $R(M, cx) = R(M, x)$
- if x is an eigenvector of M with eigenvalue λ then $R(M, x) = \lambda$.

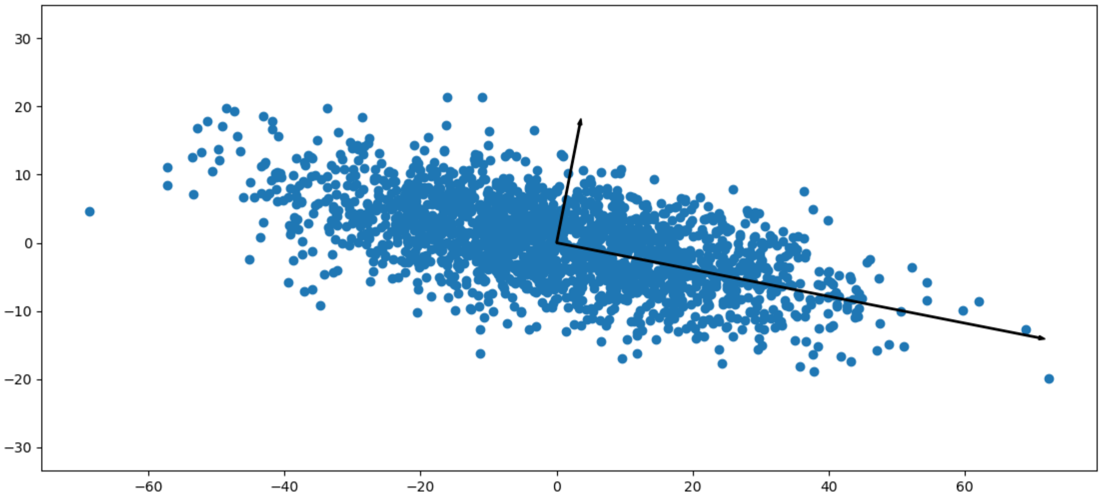


Figure 1.1: Scatter plot of multivariate Gaussian distribution centered at $(0,0)$. The vectors shown are given by the PCA of the data.

Min-Max Theorem (Courant,Weyl) 1.4.3. *Let* $M \in \mathbb{R}^{n \times n}$ *be a Hermitian matrix and* $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n$ *its eigenvalues. Then, for* $i = 0, \ldots, n-1$ *it holds:*

$$\lambda_{i+1} = \min_{u^1,\ldots,u^i \in \mathbb{C}^n} \max_{\substack{w \in \mathbb{C}^n \setminus \{0\} \\ w^H u^1 = \ldots = w^H u^i = 0}} R(M, w)$$

and in particular the minimum is achieved when $u^1, \ldots, u^i$ *are the first* i *orthonormal eigenvectors of* M*. This means that if we call the eigenvectors of* M $x_1, \ldots, x_n$ *we have*

$$\lambda_{i+1} = \max_{\substack{w \in \mathbb{C}^n \setminus \{0\} \\ w^H x_1 = \ldots = w^H x_i = 0}} R(M, w) \ . \tag{1.4.4}$$

Proof. We will sketch the proof of (1.4.4); for the complete proof see Stoer and Bulirsch (2013) (Theorem 6.9.14, p. 453). Since for any $i = 0, \ldots, n-1$, x_{i+1} is orthogonal to all the previous eigenvectors $x_1, \ldots, x_i$ and $R(M, x_{i+1}) = \lambda_{i+1}$, we have

$$\lambda_{i+1} \leq \max_{\substack{w \in \mathbb{C}^n \setminus \{0\} \\ w^H x_1 = \ldots = w^H x_i = 0}} R(M, w) \ .$$

For the other inequality let w be any nonzero vector orthogonal to $x_1, \ldots, x_i$: this means we can write w as a linear combination of the other eigenvectors, i.e.

$$w = \rho_{i+1} x_{i+1} + \ldots + \rho_n x_n \ .$$

Since $R(M, cw) = R(M, w)$ for any nonzero constant c we can suppose $||w|| = 1$ which implies $\sum_{k>i} |\rho_k|^2 = 1$ due to the orthonormality of the eigenvectors. We thus can write

$$\begin{aligned} R(M, w) &= (\rho_{i+1}x_{i+1} + \ldots + \rho_n x_n)^H M(\rho_{i+1}x_{i+1} + \ldots + \rho_n x_n) \\ &= |\rho_{i+1}|^2\lambda_{i+1} + \ldots + |\rho_n|^2\lambda_n \\ &\leq \lambda_{i+1}(|\rho_{i+1}|^2 + \ldots + |\rho_n|^2) \\ &= \lambda_{i+1} \ . \end{aligned}$$

■

Proposition 1.4.5. *The optimal basis solution to* (1.4.1) *is given by the eigenvectors of the covariance matrix* X^TX.

Proof. If w is a unitary vector we can write the variance of the vector Xw as the Rayleigh quotient of X^TX:

$$\begin{aligned} \mathrm{Var}[Xw] &= \frac{1}{N}\sum_{i=1}^{N}(Xw)_i^2 \\ &= \frac{1}{N}\,||Xw||_{\ell^2}^2 \\ &= \frac{1}{N}w^TX^TXw \\ &= \frac{1}{N}R(X^TX, w) \ . \end{aligned}$$

Thus the set of orthonormal vectors maximizing the data variance of the feature vectors is the same as that maximizing the Rayleigh quotient of X^TX, which by Theorem 1.4.3 is given by the eigenvectors of X^TX. ■

Remark 2. The data written in the new basis given by PCA is uncorrelated. In fact, because of the orthonormality of the matrix U, the covariance matrix Z^TZ is diagonal:

$$\begin{aligned} Z^TZ &= V^TX^TXV \\ &= V^TV\Lambda V^TV \\ &= \Lambda \end{aligned}$$

where $X^TX = V\Lambda V^T$ is the eigendecomposition of X^TX.

Remark 3. The eigenvectors of X^TX are the right-singular vectors of X; thus usually the PCA is obtained through the SVD of X which is computationally more advantageous.

In practice the most common use of PCA is for dimension reduction: by considering only the first $k \leq n$ eigenvectors of the covariance matrix, we can obtain a k-dimensional approximation of the data points. To do this simply consider $V_{|k} = [v^1| \dots |v^k|0| \dots |0]$[1] and define $Z_{|k} := XV_{|k}$; the k-dimensional feature vectors (rows of Z) are the projection of the original data onto the subspace generated by $v^1, \dots, v^k$. This is the best k-dimensional subspace to project onto, a statement that can be justified in three ways:

1. since $v^1, \dots, v^k$ satisfy (1.4.1) we are sure that each direction explains most of the variance in the data while remaining uncorrelated (orthogonal) to the others.

2. If we suppose the data to be perturbed by isotropic noise (i.e., with same variance along any direction) then since along $v^1, \dots, v^k$ the variance is maximal, the signal to noise ratio (defined as the ratio between signal and noise variances) is maximal along the first principal components.

3. Consider an SVD of $X = U\Sigma V^T$ and the low-rank approximation X_k as defined in (1.3.6). We have

   $$\begin{aligned} X_k V &= \sum_{j=1}^{k} \sigma_j U_{\cdot j} (V_{\cdot j})^T V \\ &= \sum_{j=1}^{k} \sigma_j U_{\cdot j} \delta_j^T \ , \end{aligned}$$

   where δ_j is the n-dimensional vector having as k-th component δ_{jk}. It is trivial to check that $Z_{|k} = XV_{|k} = X_k V$; this means that we can write $X_k = Z_{|k} V^T$, which by Theorem 1.3.7 is the best k-rank approximation of X in the 2-norm and Frobenius norm. In other words, the feature vectors of dimension k, when rewritten in the original coordinates, give the best k-dimensional approximation of the original data samples (in these two special norms). Incidentally note that $Z = XV = U\Sigma I$ is an SVD of Z, and Z_k (the best k-rank approximation of Z) is the same as $Z_{|k}$ (the original data projected on the first k eigenvectors of $X^T X$).

PCA can also be used for *nearest neighbor classification*: suppose we are given a data set $y_1, \dots, y_N$ and let V be the matrix with the principal components as columns, let d be a distance between the feature vectors (for example $d(z^1, z^2) := \left|\left|z^1 - z^2\right|\right|_2$) and suppose that we clustered the feature vectors $z_1, \dots, z_N$ into l classes (or, more typically, suppose this clustering is already given in the form of labels on the data set). We can then classify a new input vector $y \in \mathbb{R}^m$ by computing its feature vector $z^T = y^T V$ and solve

$$\min_{j=1,\dots,l} d(z, \bar{z}_j) \ ,$$

---

[1] in practice one usually defines $V_{|k} := [v^1| \dots |v^k]$, such that $Z_{|k}$ is an $N \times k$ matrix. We are using this definition where the last $n - k$ columns of these matrices are zero to show the connection with the SVD below, where we state that $XV_{|k} = X_k V$.

where the $\bar{z}_j$ is some representative of the class j, usually the average of the samples therein contained. In applications this is usually done with k-dimensional feature vectors. The advantages of classifying the feature vectors instead of the samples themselves are speed (it's faster to compute the distance of k-dimensional vectors instead of n-dimensional ones) and robustness: as remarked above the feature vectors have, in the first components, optimal signal to noise ratio.

A classical example of this use of PCA is that of face recognition through *eigenfaces*, as originally proposed in Turk and Pentland (1991): each data point is the vectorized version of a gray-scale valued image, and thus the first few PCA components represent some abstract faces, called eigenfaces. The dimension reduction process here means that we write each of the original faces as a linear combination of only a few eigenfaces. If we suppose to have multiple images of each person, we can use this information to partition the data into the sets of faces belonging to the same person. If then we receive a new image it is possible to use nearest neighbor classification to recognize it, i.e. assign it to the person corresponding to the closest class. This procedure gave very promising results and Turk and Pentland (1991) was a seminal paper.

1.5 2DPCA

One of the shortcomings of using PCA for face recognition as proposed in Turk and Pentland (1991) is the need to vectorize the face images. This means that throughout the process each data point is treated as a one-dimensional entity, and computing all the needed scalar products among these objects amounts to keeping track only of the pixel by pixel correlations between images.

In order to improve on this aspect, in Yang et al. (2004) the two-dimensional PCA (2DPCA) method was proposed. While originally developed for the specific application of nearest neighbor classification of face images, this method is well suited for any type of two-dimensional data where we wish to apply dimensionality reduction without loosing the intrinsic two-dimensional correlations. We will thus describe how the method works without referring specifically to face images.

Like PCA this method uses a set of sample data to compute a covariance matrix, and then rewrites the samples in the basis given by the eigenvectors of this matrix. Restricting to the eigenvectors corresponding to largest eigenvalues corresponds to a dimension reduction that explains most of the data variance. The key difference between applying PCA to vectorized matrices and using 2DPCA is that the latter works directly on the matrices: the resulting covariance matrix is smaller and takes into account all the two-dimensional correlations intrinsic in the data. We will start by describing uni-lateral 2DPCA, which works only on rows or columns of the sample data matrices and in subsection 1.5.2 introduce bi-lateral 2DPCA, which is what we are really interested in.

Let $Y \in \mathbb{R}^{m \times n}$ be a random matrix which we sample from to gather the two-dimensional input data; we can think of Y and its samples in the terms of a certain class of images (e.g. natural images or two-dimensional seismic data) or small patches extracted from these. The basic idea of the uni-lateral horizontal 2DPCA is to search for a vector $x \in \mathbb{R}^n$ such that the rows of Y projected onto x, that is $z = Yx \in \mathbb{R}^m$, have maximum variance. The rationale is that this x will be the direction that best approximates the two-dimensional data samples of Y or, as is usually said of PCA, explains most of the variance in it. However this is actually inaccurate, since the variance of a set of samples of z would be the covariance matrix

$$\begin{aligned} S_x &:= \mathbb{E}\Big[(z - \mathbb{E}[z])(z - \mathbb{E}[z])^T\Big] \\ &= \mathbb{E}\Big[(Y - \mathbb{E}[Y])x\big[(Y - \mathbb{E}[Y])x\big]^T\Big] \ , \end{aligned}$$

and not a number that one can maximize. Thus we will ask to maximize the *total scatter* $\sum_{i=1}^m \mathrm{Var}[z_i]$, which is nothing else but the trace of S_x.

Proposition 1.5.1. *The total scatter of the projection of the random matrix Y onto direction x can be expressed as the Rayleigh quotient of x with respect to*

$$G := \mathbb{E}\Big[(Y - \mathbb{E}[Y])^T(Y - \mathbb{E}[Y])\Big] \ ,$$

i.e.

$$tr(S_x) = x^T G x \ . \tag{1.5.2}$$

Furthermore G is positive semidefinite.

Proof. We have

$$\begin{aligned} tr(S_x) =& \sum_{i=1}^m \mathbb{E}\Big[(Y_{i\cdot} - \mathbb{E}[Y]_{i\cdot})x \ (Y_{i\cdot} - \mathbb{E}[Y]_{i\cdot})x\Big] \\ =& \sum_{i=1}^m \mathbb{E}\Big[\sum_{j=1}^n (Y_{ij} - \mathbb{E}[Y]_{ij})x_j \sum_{k=1}^n (Y_{ik} - \mathbb{E}[Y]_{ik})x_k\Big] \\ =& \sum_{j,k} x_j x_k \mathbb{E}\Big[\sum_i (Y - \mathbb{E}[Y])^T_{ji}(Y - \mathbb{E}[Y])_{ik}\Big] \\ =& \sum_{j,k} x_j x_k \mathbb{E}\Big[(Y - \mathbb{E}[Y])^T_{j\cdot}(Y - \mathbb{E}[Y])_{\cdot k}\Big] \\ =& x^T \mathbb{E}\Big[(Y - \mathbb{E}[Y])^T(Y - \mathbb{E}[Y])\Big] x \ . \end{aligned}$$

For the second part of the proof simply consider that

$$\begin{aligned} \forall x \in \mathbb{R}^n \quad x^T G x &= tr(S_x) \\ &= \sum_{i=1}^m \mathrm{Var}[z_i] \geq 0 \ . \end{aligned}$$

■

Call the n eigenvalues of G $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq 0$ and the corresponding eigenvectors (which we will always suppose orthonormal) $u_1, \ldots, u_n$, then u_1 is the unitary vector maximizing $x^T G x$. If then we wish to find the next optimal direction, i.e. the unitary vector maximizing $x^T G x$ with the added constraint of being orthogonal to u_1, this will be given by the second eigenvector u_2 (see Theorem 1.4.3). We can thus take the first d eigenvectors as columns of an $n \times d$ matrix $U = [u_1 | \ldots | u_d]$, and define the $m \times d$ matrix $Z = YU$; Z is a lower dimensional (fewer columns) approximation of Y.

In practice, from N sample $m \times n$ matrices $Y_1, \ldots, Y_N$ we will use the estimator for G

$$G_R := \frac{1}{N} \sum_{j=1}^{N} (Y_j - \bar{Y})^T (Y_j - \bar{Y}) \ , \tag{1.5.3}$$

where $\bar{Y} = \frac{1}{N} \sum_{j=1}^{N} Y_j$ is the sample mean. This $n \times n$ matrix is called the *image right covariance* or *image row covariance* matrix of the matrix samples $Y_1, \ldots, Y_N$. The 2DPCA method consists of finding the first d orthonormal eigenvectors of G_R, taking them as columns of a matrix $U = [u_1 | \ldots | u_d]$, and for every sample Y_j computing its *horizontal feature matrix* $Z_j = Y_j U$. Note that if $d = n$ then U is unitary, and from the feature matrices one can recover the original matrices by multiplying them on the right with U^T. However in the usual case $d < n$, Z_j is to be regarded as a lower dimensional approximation of Y_j, for one can recover only $Y_j U U^T = Z_j U^T$.

The following theorem shows that 2DPCA is equivalent to the usual PCA if one takes as samples the rows of the images. The proof consists of straightforward computations and can be found for example in Kong et al. (2005).

Proposition 1.5.4. *The matrix G_R is the covariance matrix of the rows of the centered samples $Y_1, \ldots, Y_N$, i.e.*

$$G_R = \frac{1}{N} \Psi \Psi^T$$

where

$$\Psi^T = \left[\begin{array}{c} Y_1 - \bar{Y} \\ \hline Y_2 - \bar{Y} \\ \hline \vdots \\ \hline Y_N - \bar{Y} \end{array} \right]$$

Remark 4. This equivalence allows for a direct comparison and illustrates two fundamental advantages of 2DPCA over the usual PCA applied on the vectorized samples: we have more samples in a lower dimensional space (mN samples in $\mathbb{R}^n$ instead of N samples in $\mathbb{R}^{mn}$) and we have to compute eigenvectors of an $n \times n$ matrix instead of an $m^2 n^2$ matrix.

Until now we have illustrated *horizontal* 2DPCA; one can use the transpose of the samples Y_j^T instead of Y_j and thus obtain the *vertical* 2DPCA, which operates on columns of the samples instead of rows. This will amount to finding the largest eigenvectors of the $m \times m$ *image left covariance* or *image column covariance* matrix

$$G_L := \frac{1}{N} \sum_{j=1}^{N} (Y_j - \bar{Y})(Y_j - \bar{Y})^T \ , \tag{1.5.5}$$

which, because of Proposition 1.5.4, is equivalent to finding the d largest eigenvectors $v_1, \ldots, v_d$ of the covariance matrix of the columns of the sample matrices (i.e. performing PCA on columns). If we use these eigenvectors as columns of a matrix $V = [v_1| \ldots |v_d]$ then we can obtain the *vertical feature matrices* as $W_j^T = Y_j^T V$, or equivalently $W_j = V^T Y_j$.

1.5.1 Properties of 2DPCA

In Yang and Liu (2007) the authors illustrate three important properties of the 2DPCA, which we summarize here.

Translation Invariance. Horizontal (vertical) 2DPCA is invariant to row (columns) permutations in the samples. In fact it's easy to verify that each term in the sum in (1.5.3) is a sum of rank 1 matrices of the form $v^T v$ where v is a row of the sample, i.e.

$$(Y_j - \bar{Y})^T (Y_j - \bar{Y}) = \sum_{i=1}^{m} \left[(Y_j - \bar{Y})_{i\cdot} \right]^T (Y_j - \bar{Y})_{i\cdot} \ ,$$

and thus a permutation of the rows is just a rearrangement of the terms of the double sum defining G_R.

Uncorrelation. The columns (rows) of the feature matrix obtained from horizontal (vertical) 2DPCA are uncorrelated. In fact (for horizontal 2DPCA and $Z = YU$):

$$\begin{aligned} \mathrm{Cov}(Z_{\cdot j}, Z_{\cdot k}) &= \mathbb{E}\left[\left[(Y - \bar{Y}) u_j \right]^T (Y - \bar{Y}) u_k \right] \\ &= u_j^T G_R u_k \\ &= 0 \end{aligned}$$

because of the orthornomality of the eigenvectors of G_R.

Minimal Mean Square Error Representation. Given any orthonormal basis $x_1, \ldots, x_n$ of $\mathbb{R}^n$, one can project the rows of a matrix $Y \in \mathbb{R}^{m \times n}$ onto these vectors and write

$$Y = \sum_{j=1}^{n} (Y x_j) x_j^T \ .$$

By restricting the sum to the first d vectors one can then obtain an approximation $\hat{Y}_{x_1, \ldots, x_d} := \sum_{j=1}^{d} (Y x_j) x_j^T$. The following proposition can then be proven (see Zhang et al. (2006)):

Proposition 1.5.6. *Given the orthonormal eigenvectors* $u_1, \ldots, u_n$ *of* G_R *corresponding to eigenvalues* $\lambda_1 \geq \ldots \geq \lambda_n \geq 0$ *and some integer* $d \leq n$*, then* $\hat{Y}_{u_1,\ldots,u_d}$ *minimizes* $\epsilon^2 := \mathbb{E}\left[\left|\left|Y - \hat{Y}_{x_1,\ldots,x_d}\right|\right|_F\right]$ *which is then equal to* $\epsilon^2 = \sum_{j=q+1}^{n} \lambda_j$*.*

1.5.2 Bilateral 2DPCA

In order to preserve the spatial correlation in the samples $Y_1, \ldots, Y_N$ both row and column-wise, it is possible to find an $n \times r$ matrix U and an $m \times l$ matrix V to right and left multiply the samples and obtain $l \times r$ *bilateral feature matrices* of the form $Z_i = V^T Y_i U$. A first straightforward approach to do this is presented in Zhang and Zhou (2005) and consists in applying horizontal 2DPCA to find U and vertical 2DPCA to find V; we will call this *simple bilateral* 2DPCA. However, as noted in Kong et al. (2005), what we really want is to optimize simultaneously for both projections, i.e. find U and V solution to

$$\min_{U \in \mathbb{R}^{n \times r}, V \in \mathbb{R}^{m \times l}} \sum_{i=1}^{N} \left|\left|Y_i - V Z_i U^T\right|\right|_F \ . \tag{1.5.7}$$

Computing independently horizontal and vertical PCA is a compromise which we will settle for in our numerical trials, since problem (1.5.7) is not easily solvable. In fact in Kong et al. (2005) it is shown that (1.5.7) is equivalent to maximizing

$$\sum_{i=1}^{N} \left|\left|V^T Y_i U\right|\right|_F^2 = tr(\bar{G}) \ ,$$

where $\bar{G}$ is the covariance matrix of the projected samples, i.e.

$$\begin{aligned} \bar{G} =& \frac{1}{N} \sum_{i=1}^{N} Z_i^T Z_i \\ =& \frac{1}{N} \sum_{i=1}^{N} U^T Y_i^T V V^T Y_i U \ . \end{aligned}$$

The trace of $\bar{G}$ cannot be expressed as a Rayleigh quotient, so unlike for the horizontal or vertical 2DPCA there is no straightforward way to solve the optimization problem. However in Kong et al. (2005) the authors propose an iterative algorithm which converges to a local minimum of (1.5.7), based on the following Lemma:

Lemma 1.5.8. *1. Given* V_{opt} *optimal,* U_{opt} *columns' are the first r eigenvectors (i.e. corresponding to the r largest eigenvalues) of the matrix*

$$\bar{G}' := \frac{1}{N} \sum_{i=1}^{N} Y_i^T V_{opt} V_{opt}^T Y_i$$

2. *Given U_{opt} optimal, V_{opt} columns' are the first l eigenvectors (i.e. corresponding to the l largest eigenvalues) of the matrix*

$$\bar{G}'' := \frac{1}{N}\sum_{i=1}^{N} Y_i U_{opt} U_{opt}^T Y_i^T$$

The algorithm then simply consists of initializing U_0 (for example to the identity matrix or the matrix given by horizontal 2DPCA) and for each iteration k compute the new optimal V_k and U_k, until the relative error decrease

$$\theta := \frac{\mathcal{E}(k-1) - \mathcal{E}(k)}{\mathcal{E}(k-1)} \text{ , where } \mathcal{E}(k) = \frac{1}{N}\sum_{i=1}^{N} \left|\left|Y_i - V_k Z_i U_i^T\right|\right|_F^2 \ ,$$

goes below a certain threshold ϵ.

1.6 Image Quality Assessment

A typical task in signal processing is that of reconstructing or approximating a signal $y \in \mathbb{R}^n$ with a distorted version $\tilde{y} \in \mathbb{R}^n$. In order to compare different reconstruction methods it is thus essential to have a good measure of the distortion. One of the most basic such measures is the Mean Square Error (MSE) which is nothing but the normalized squared 2-norm:

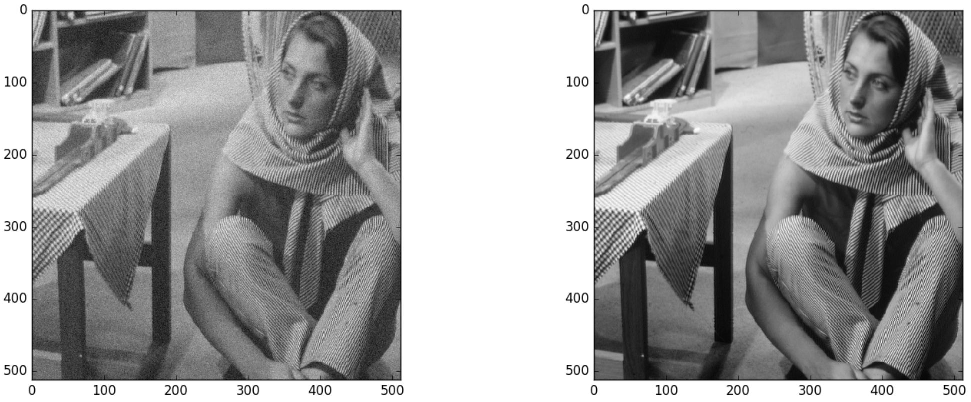
$$\text{MSE}(y, \tilde{y}) := \frac{1}{n} \left|\left|y - \tilde{y}\right|\right|_2^2 \ .$$

In order to account for the very large dynamic range of many natural signals the Peak Signal to Noise Ratio (PSNR) has historically been used as a reference distortion measure, which transforms the MSE into a logarithmic decibel scale:

$$\begin{aligned} \text{PSNR}(y, \tilde{y}) &:= 10 \log_{10} \frac{y_\infty^2}{\text{MSE}(y, \tilde{y})} \\ &= 20 \log_{10} \frac{y_\infty}{\sqrt{MSE(y, \tilde{y})}} \end{aligned}$$

where $y_\infty = \max_{i=1,\dots,n} |y_i|$. In Chapters 3 and 5 we'll be interested in the case of the signal being an image, and particularly a natural image that has to be viewed by humans. The human visual system is incredibly complex,very selective in the information it registers and sophisticated in processing it. It suffices to think that the estimated bandwidth of the receptors on the retina is estimated around 20 Gb/s while that of the optic nerve connecting the eyes to the visual cortex V1 (where most of the selective processing just starts) is around 4 Gb/s (see Echeverri (2006)). The definitive qualitative distortion measure for images will be for us "how distorted image $\tilde{y}$ appears when shown to a number of human subjects", though this is of course very hard to quantify, especially

Figure 1.2: `barbara` image (see Figure (5.9c) at page 127) with added Gaussian noise (a) and rescaled by a $1 + \epsilon$ factor (b)



(a) $\mathrm{PSNR} = 25.7, \mathrm{HaarPSI} = 0.487$ (b) $\mathrm{PSNR} = 25.58, \mathrm{HaarPSI} = 0.993$

when compared to the ease of computing the PSNR or a similar index for digital images stored on a computer. It is on the other hand necessary to find something better than the PSNR, since as can be seen from Figure (1.2) this index is not good for example at discriminating between white noise and a simple rescaling of the pixel values (i.e. a change in luminosity of the image), while the second image is clearly for our eyes more similar to the original than the first. We thus say that a visual quality index performs better than another if it correlates better with distortion scores given by humans on degraded versions of an image such as those used in the tests reported on in Ponomarenko et al. (2015) and Larson and Chandler (2010).

Many indexes that perform better than the PSNR have been proposed in the last three decades, notably the Structural Similarity (SSIM) (see Wang et al. (2004)) and the Visual Saliency-Induced Index (VSI) (see Zhang et al. (2014)). Very recently the Haar Wavelet-Based Perceptual Similarity Index (HaarPSI) has been proposed in Reisenhofer et al. (2018): from the tests conducted by the authors it appears to behave better than all other indexes. We will thus use this index in our numerical tests on image reconstruction, alongside the PSNR for historical and comparative reasons. The HaarPSI uses the function

$$\forall a, b \in \mathbb{R} \quad S(a, b) := \frac{2ab + C}{a^2 + b^2 + C}$$

for some $C \in \mathbb{R}$ as a fundamental measure of dissimilarity between absolute values of a 4-level Haar bi-dimensional wavelet transform. It compares the local dissimilarity of these coefficients between two images and finally sums them up with appropriate weights to obtain a global dissimilarity index, which is 1 if the two images are perfectly identical and gradually decreases to 0 as distortion increases.

1.7 Clustering data

A problem we'll be dealing with in Chapters 3 and 5 is that of partitioning a data set into clusters that somehow capture the salient features of the data. This is a very difficult unsupervised learning task, mainly because it is not clear what the objective is: though most methods used in practice aim to minimize some measure of inner difference in the clusters (usually equivalent to maximizing the difference between different clusters) how this intra- and inter-differences are defined and computed varies with each proposed model and of course also with the chosen topology for the space. However given a specific application of these methods, it is usually possible to give an indirect measure of the quality of the clustering: for example we could use the image reconstruction methods proposed in Chapters 3 and 5 to give an indirect assessment of the performance of different clustering methods in the form of the quality of the reconstructed image.

1.7.1 K-Means

We will give a brief overview of some popular clustering methods, starting from the classic and still widely used K-Means. Given a data set $\mathcal{S} = \{s_1, \dots, s_N\} \subset \mathbb{R}^n$ this method aims at finding a partition $S_1, \dots, S_K$ of $\mathcal{S}$ that minimizes the so-called within-cluster sum of squares (WCSS):

$$\begin{aligned} \min_{\substack{S_1,\dots,S_K \\ S_1\sqcup\dots\sqcup S_K=\mathcal{S}}} \mathrm{WCSS}(S_1,\dots,S_K) &= \min_{\substack{S_1,\dots,S_K \\ S_1\sqcup\dots\sqcup S_K=\mathcal{S}}} \sum_{i=1}^{K} \sum_{s\in S_i} ||s-\mu_i||_2^2 \\ \text{where} \quad \mu_i &:= \frac{1}{|S_i|} \sum_{s\in S_i} s \ . \end{aligned} \tag{1.7.1}$$

In other words the WCSS associated to $\mathcal{S}$ is the partition minimizing the sum of terms $|S_i|\mathrm{Var}S_i$. The problem is NP-Hard (see Aloise et al. (2009)) so some approximation technique must be used: the typical procedure is the Loyd's Algorithm described in Algorithm (1). This consists in iterating a two-step process where first we update the clusters based on the distance between the samples and the centroids and then we update the centroids in function of the newly found clusters. Since both these steps decrease the value of the WCSS function the algorithm is guaranteed to converge to a local minimum. Instead of fixing the number of iterations $\mathcal{I}$ one could fix a threshold for the value for the WCSS and stop when the chosen partition gives a WCSS value below this. Furthermore there are various possible strategies for the choice of the initial centroids, which is very important for the convergence of the algorithm. In Ding and He (2004) it is shown that the PCA of the data could supply a good initial approximation of the centroids, while most software implementations use random guesses. For example in the implementation in the `scikit-learn` python library (which we used for our numerical tests in Subsection 5.4) the algorithm is run by default 10 times with different random seeds dictating the choice of initial centroids and finally the best solution is chosen.

If we set to $\mathcal{I}$ the number of iterations, the complexity of Lloyd's Algorithm is easily

Algorithm 1 Lloyd's Algorithm

```
Input: Data set $\mathcal{S} = \{s_1, \dots, s_N\} \subset \mathbb{R}^n$, number of clusters $K \in \mathbb{N}, K < N$ and number of iterations $\mathcal{I}$
Output: Partition $S_1, \dots, S_K$ of $\mathcal{S}$ approximating optimal solution to (1.7.1)
Randomly choose $\mu_1^{(0)}, \dots, \mu_K^{(0)} \in \mathbb{R}^n$
for $i = 1, \dots, \mathcal{I}$ do
    Assignment Step: Assign each point to the nearest $\mu_k^{(i)}$, i.e. define
        $\forall k = 1, \dots, K \; S_k^{(i)} := \left\{ s \in \mathcal{S} \;\middle|\; \left\| s - \mu_k^{(i-1)} \right\| \leq \left\| s - \mu_j^{(i-1)} \right\| \; \forall j \neq k \right\}$
    Update Step: Recompute the centroids of the clusters:
        $\mu_k^{(i)} := \frac{1}{|S_k^{(i)}|} \sum_{s \in S_k^{(i)}} s$
end for
$\forall k = 1, \dots, K \; S_k := S_k^{(\mathcal{I})}, \; \mu_k := \mu_k^{(\mathcal{I})}$
```

seen to be $O(nNK\mathcal{I})$. In the special case of the samples being one-dimensional and $K = 2$, we can find the exact minimizing solution at the cost of ordering the samples (so in $\mathcal{O}(N \log N)$. In fact, supposing the samples are ordered $s_1 \leq \ldots \leq s_N$, it is sufficient to look at solutions of the type

$$(\{s_1, \dots, s_k\}, \{s_{k+1}, \dots, s_N\})$$

for all $k = 1, \dots, N-1$, as we prove in the following original Theorem.

Theorem 1.7.2. *For $n = 1$ and $K = 2$, suppose the samples are ordered, i.e. $\mathcal{S} = \{s_1 \leq s_2 \leq, \dots, \leq s_N\} \subset \mathbb{R}$; then $\exists k \in \{0, 1, \dots, N\}$ s.t. the optimal solution to* (1.7.1) *is of the form (S_1^k, S_2^k) with $S_1^k = \{s_1, \dots, s_k\}$ and $S_2^k = \{s_{k+1}, \dots, s_N\}$.*

Proof. Observe that for any candidate solution (S_1, S_2) we can write

$$\begin{aligned}
\mathrm{WCSS}(S_1, S_2) &= \sum_{s \in S_1} |s - \mu_1|^2 + \sum_{s \in S_2} |s - \mu_2|^2 \\
&= \sum_{s \in S_1} \left(s^2 + \mu_1^2 - 2s\mu_1 \right) + \sum_{s \in S_2} \left(s^2 + \mu_2^2 - 2s\mu_2 \right) \\
&= \sum_{s \in \mathcal{S}} s^2 + |S_1|\mu_1^2 - 2\mu_1 \sum_{s \in S_1} s + |S_2|\mu_2^2 - 2\mu_2 \sum_{s \in S_2} s \\
&= \sum_{s \in \mathcal{S}} s^2 + |S_1|\mu_1^2 - 2|S_1|\mu_1^2 + |S_2|\mu_2^2 - 2|S_2|\mu_2^2
\end{aligned}$$

where we used the fact that $\sum_{s\in\mathcal{S}_i} s = |\mathcal{S}_i|\mu_i$ for $i = 1, 2$ by definition of the centroids. We can then write

$$\mathrm{WCSS}(S_1, S_2) = \sum_{s\in\mathcal{S}} s^2 - \mathcal{F}(S_1, S_2)$$

where the first term is a constant depending only on $\mathcal{S}$ and

$$\mathcal{F}(S_1, S_2) := |S_1|\mu_1^2 + |S_2|\mu_2^2$$

is always positive; we are thus looking to maximize $\mathcal{F}(S_1, S_2)$. Now, suppose we have a solution $S = (S_1, S_2)$: what happens to the value of $\mathcal{F}$ if we switch two elements, i.e. we reassign one element of S_1 to S_2 and one element of S_2 to S_1? Define this new candidate solution $S^* = (S_1^*, S_2^*)$ as $S_1^* = (S_1 \setminus \{s_h\}) \cup \{s_k\}$ and $S_2^* = \mathcal{S} \setminus S_1^*$, for some $s_h \in S_1$ and $s_k \in S_2$. We can then write

$$\begin{aligned}
\mathcal{F}(\mathcal{S}^*) &= \frac{|S_1|}{|S_1|^2}\left(\sum_{s\in S_1^*} s_i\right)^2 + \frac{|S_2|}{|S_2|^2}\left(\sum_{s\in S_2^*} s_i\right)^2 \\
&= \frac{1}{|S_1|}\left(\sum_{s\in S_1} s_i - s_h + s_k\right)^2 + \frac{1}{|S_2|}\left(\sum_{s\in S_2} s_i - s_k + s_h\right)^2 \\
&= \frac{1}{|S_1|}\left[\left(\sum_{s\in S_1} s_i\right)^2 + s_h^2 + s_k^2 - 2s_h s_k + 2s_k\left(\sum_{s\in S_1} s_i\right) - 2s_h\left(\sum_{s\in S_1} s_i\right)\right] + \\
&\quad + \frac{1}{|S_2|}\left[\left(\sum_{s\in S_2} s_i\right)^2 + s_h^2 + s_k^2 - 2s_h s_k + 2s_h\left(\sum_{s\in S_2} s_i\right) - 2s_k\left(\sum_{s\in S_2} s_i\right)\right] \\
&= |S_1|\mu_1^2 + |S_2|\mu_2^2 + \left(\frac{1}{|S_1|} + \frac{1}{|S_2|}\right)(s_h - s_k)^2 + 2\mu_1(s_k - s_h) + 2\mu_2(s_h - s_k) \\
&= \mathcal{F}(\mathcal{S}) + f(s_k - s_h)\ ,
\end{aligned}$$

where

$$f(x) = \frac{N+1}{|S_1||S_2|}x^2 + 2(\mu_1 - \mu_2)x\ .$$

The function f is quadratic with two zeros at 0 and $2\frac{|S_1||S_2|}{N+1}(\mu_2 - \mu_1)$. Now let $\Sigma = (\Sigma_1, \Sigma_2)$ be the optimal solution to problem (1.7.1), and suppose without lack of generality that $s_1 \in \Sigma_1$: we want to show that $\nexists i, j,\ i < j$ s.t. $s_j \in \Sigma_1$ and $s_i \in \Sigma_2$. If this were not the case, we show that it would always possible to find indices h and k such that $s_h \in \Sigma_1$, $s_k \in \Sigma_2$ such that $f(s_k - s_h) \geq 0$, thus giving Σ^* a better value than Σ. In fact, there are 3 possible cases:

- $\mu_2 - \mu_1 < 0$: it suffices to choose h and k such that $s_k - s_h > 0$, always possible because $s_1 \in \Sigma_1$;
- $\mu_2 - \mu_1 = 0$: it suffices to choose $s_h \neq s_k$;
- $\mu_2 - \mu_1 > 0$: it suffices to choose h and k such that $s_k - s_h < 0$, which can be done since we're supposing the claim to be false and thus $\exists i < j$ s.t. $s_i \in \Sigma_2$ and $s_j \in \Sigma_1$.

■

It is interesting to note that, if we define the data matrix $Y \in \mathbb{R}^{n \times N}$ with the samples s_i as columns, the K-means optimization problem can be stated in matrix form as

$$\min_{\substack{C \in \mathbb{R}^{n \times K} \\ X \in \mathbb{R}^{K \times N}}} ||Y - CX||_F^2 \ , \ \forall j = 1, \ldots, N \ X_{\cdot j} = e_k \text{ for some } k = 1, \ldots, K \ , \tag{1.7.3}$$

where the columns of the matrix C represent the centroids of the clusters and the columns of X the assignment of each sample to one of the clusters. The problem so stated is a particular case of the dictionary learning problem (4.0.2) which we will treat in Chapter 4; in this interpretation, we are trying to approximate each of the N data points with one of the K atoms (centroids in the K-means usual formulation). The Loyd algorithm is then simply an instance of the general scheme 7 at page 91.

It is possible to generalize the K-means problem to any metric space (X, d) by substituting the centroids with the *Frechet means*:

$$\forall S \subseteq \mathcal{S} \ \mu_S := \arg\min_{\mu \in X} \sum_{s \in S} d^2(s, \mu) \ .$$

Problem (1.7.1) then becomes

$$\min_{\substack{S_1, \ldots, S_K \\ S_1 \sqcup \ldots \sqcup S_K = \mathcal{S}}} \text{WCSS}(S_1, \ldots, S_K) = \min_{\substack{S_1, \ldots, S_K \\ S_1 \sqcup \ldots \sqcup S_K = \mathcal{S}}} \sum_{i=1}^{K} \sum_{s \in S_i} d^2(s - \mu_i, 0) \ . \tag{1.7.4}$$

Lloyd's algorithm still works, the problem is that computing the Frechet means is usually not trivial at all and a research subject in itself (see for example Kalcsics et al. (2002) and Nickel and Puerto (1999)). If one simply uses the arithmetic averages in the update step, regardless of the topology, then the algorithm is not guaranteed to converge since the value of WCSS may not decrease if the μ_i are not the real Frechet means.

1.7.2 Graph Based Methods

Another class of methods for clustering a set of N data points are the graph-based methods which work exclusively on a weighted graph structure obtained from an arbitrary similiarity measure defined on the samples. The appeal of such methods is the straightforward possibility to consider particular and non-standard topologies for the space the samples live in: the graph simply captures the structure given by application of the similarity measure on the samples and can be easily experimented with. The construction of the graph does require $O(N^2)$ applications of the similiarity measure in order to compute the weights, which makes this class of methods much less suitable for large data sets than K-means.

Given a data set $\mathcal{S} = \{s_1, \dots, s_N\}$ we define the complete *data similarity graph* $G = (V, E)^2$ with $V = \{1, \dots, N\}$ and $E = \mathcal{S} \times \mathcal{S}$. We furthermore suppose to have a *similarity measure*, i.e. a function $\sigma : \mathcal{S} \times \mathcal{S} \to [0, 1]$ such that $\sigma(s_i, s_j)$ gives a measure of how similar samples $s_i, s_j \in \mathcal{S}$ are, 1 meaning they are the same and smaller values meaning they are increasingly different. A typical choice is applying a Gaussian kernel to a distance d defined on $\mathcal{S}$, i.e.

$$\sigma(x, y) = e^{-\beta \frac{d^2(x,y)}{\sigma^2_{\mathcal{S}}}} , \tag{1.7.5}$$

where $\sigma^2_{\mathcal{S}}$ is the data variance of the samples and β is some positive real number. The weights for the graph are then defined as

$$w_{ij} = \mathbb{1}_{(\rho,1]}(\sigma(s_i, s_j)) \tag{1.7.6}$$

for some parameter $\rho \in [0, 1)$ that gives a lower limit to the similarities we wish to take into consideration: we are ignoring all the relations betwen samples that are too different from each other because all the corresponding edge weights are set to 0. This parameter effectively allows us to not consider the complete graph of the whole data-set (i.e. in practice edges with weight 0 won't be added to E at all), which is a necessary compromise when dealing with anything but trivially small data-sets. Having defined the $N \times N$ *affinity matrix* $W = \{w_{ij}\}_{i,j=1,\dots,N}$ and the diagonal $N \times N$ matrix D whose i-th element is the degree of node s_i i.e. $d_i = \sum_{j \in V} w_{ij}$, we define the *normalized Laplacian* matrix

$$L = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}} .$$

Note that the larger ρ is the sparser W and L will be.

We will describe in detail two graph based clustering methods, namely spectral (Shi and Malik (2000)) and Felzenszwalb-Huttenlocher (Felzenszwalb and Huttenlocher (2004)) clustering. Both these methods were originally developed for the task of image segmentation. In this case the graph $G = (V, E)$ is built from a gray-valued image in the following way: each pixel is a vertex and edges connect only neighboring pixels i.e. apart from the borders of the image each pixel/vertex has 8 neighbors. In the original papers however the weights are given a different meaning: in Shi and Malik (2000) the more standard convention of using a Gaussian kernel like in equation (1.7.5) is used, where the base distance d is simply the absolute value difference between the pixels. Thus in this

---

[2] with a slight abuse of notation we will sometimes consider instead $G = (\mathcal{S}, E)$ i.e. we will interchange the sample s_i with its associated vertex v_i: this will be clear from the context and won't cause loss of generality

case the edge weights represent the degree of similarity between vertices. In Felzenszwalb and Huttenlocher (2004) instead no Gaussian kernel is used: the weight of an edge connecting pixels s_i and s_j is given by $|s_i - s_j|$, thus the edge weights in this case represent the degree of dissimilarity between vertices. In order to keep consistent with the original papers we maintained this difference which is purely formal: one can always switch between a similarity and dissimilarity graph by a simple manipulation of the matrix W.

1.7.3 Spectral Clustering

The first graph-based clustering method we are interested in is the *spectral clustering* method, which partitions the vertices of the data similarity graph based on the spectral properties of the graph Laplacian matrix. A computation of the second eigenvector of L is required which in general takes $O(N^3)$ time. The method is thus infeasible for anything but small data sets, unless one uses some approximated version of the method like the one proposed in Yan et al. (2009), where k-means is first applied with relatively high k and then spectral clustering is used on the centroids of the k classes. However here we will refer to the spectral clustering method as detailed in Shi and Malik (2000), where it was originally developed for the task of image segmentation.

Spectral clustering is a top-down procedure, this means it works by recursively splitting subgraphs into two: we'll describe the procedure as applied in the first step to the whole graph, then this exact same procedure is recursively applied to the two obtained subgraphs. The main idea is to consider a partitioning of the nodes $\mathcal{S} = A \sqcup B = A \sqcup (\mathcal{S} \setminus A)$ as the *cut* that it induces, i.e. the set of edges connecting vertices in A to $\mathcal{S} \setminus A$. For any two subsets of nodes $A, B \subset \mathcal{S}$ we can define

$$\mathrm{assoc}(A, B) := \sum_{u \in A, v \in B} w_{uv} \ ;$$

the weight associated to the cut induced by $A \subset \mathcal{S}$ is

$$\mathrm{cut}_A := \mathrm{assoc}(A, \mathcal{S} \setminus A) \ .$$

This is some measure of how separated the points in A are from those outside, according to the chosen similarity measure: it is a measure of inter-class variance. The first strategy that comes to mind is to simply look for the node subset A that minimizes cut_A and partition the set of nodes into $\mathcal{S} = A \sqcup (\mathcal{S} \setminus A)$, but of course this favours noticeably solutions where the cardinality of A is very small (i.e. with very few outgoing edges). In order to balance for the cardinality the authors of Shi and Malik (2000) define

$$\mathrm{Ncut}_A = \mathrm{Ncut}(A, \mathcal{S} \setminus A) = \frac{\mathrm{cut}_A}{\mathrm{assoc}(A, \mathcal{S})} + \frac{\mathrm{cut}_{\mathcal{S} \setminus A}}{\mathrm{assoc}(\mathcal{S} \setminus A, \mathcal{S})} \ , \tag{1.7.7}$$

where $\mathrm{assoc}(A, \mathcal{S}) = \sum_{u \in A, v \in \mathcal{S}} w_{uv}$ is the total weight of edges with at least one vertex in A, i.e. it counts both the edges that remain internal to A and those that have the other edge in $\mathcal{S} \setminus A$; note that $\mathrm{Ncut}_A = \mathrm{Ncut}_{\mathcal{S} \setminus A}$. This means that Ncut_A is the sum of the proportions of total weights of edges with one vertex in A (respectively $\mathcal{S} \setminus A$) that are outgoing (i.e. have the other vertex in the complementary set). The authors show that minimizing (1.7.7) is equivalent to maximizing

$$\text{Nassoc}_A = \text{Nassoc}(A, \mathcal{S} \setminus A) = \frac{\text{assoc}(A, A)}{\text{assoc}(A, \mathcal{S})} + \frac{\text{assoc}(\mathcal{S} \setminus A, \mathcal{S} \setminus A)}{\text{assoc}(\mathcal{S} \setminus A, \mathcal{S})} \ , \tag{1.7.8}$$

i.e. minimizing the similarity between A and $\mathcal{S} \setminus A$ or maximizing the similarity within nodes in A and in $\mathcal{S} \setminus A$ lead to the same optimal solution A^*. Solving exactly this problem is NP-Hard, so the authors propose to reformulate the problem introducing indicator variables $x_1, \dots, x_N \in \{-1, 1\}$ s.t. $x_i = 1 \Leftrightarrow s_i \in A$ and later relax the integrality constraint obtaining a polynomial time approximated solution. The authors start with writing

$$\text{Ncut}_x := \text{Ncut}_A = \frac{\sum_{x_i>0, x_j<0} -w_{ij} x_i x_j}{\sum_{x_i>0} d_i} + \frac{\sum_{x_i<0, x_j>0} -w_{ij} x_i x_j}{\sum_{x_i<0} d_i}$$

and, after various manipulations, obtain

$$\text{Ncut}_x = \text{Ncut}_y = \frac{y^T (D - W) y}{y^T D y} \ , \tag{1.7.9}$$

where $y = (\mathbf{1} + x) - b(\mathbf{1} - x)$, $b = \frac{\sum_{x_i>0} d_i}{\sum_{x_i<0} d_i}$ and $\mathbf{1}$ is the vector of all ones. By substituting $z = D^{\frac{1}{2}} y$ into (1.7.9), the authors finally obtain that minimizing (1.7.7) is equivalent to solving

$$\min_{\substack{z \ s.t. \ z_i \in \{2\sqrt{d_i}, -2b\sqrt{d_i}\} \\ z^T z_0 = 0}} \frac{z^T L z}{z^T z} \ , \tag{1.7.10}$$

where $z_0 = D^{\frac{1}{2}} \mathbf{1}$ is eigenvector of L with eigenvalue 0, corresponding to the trivial solution where all vertices belong to the same cluster, i.e. $A = \mathcal{S}$. If we relax the condition forcing the discreteness of the components of z, we can use theorem 1.4.3 to state that the optimal value of (1.7.10) is given by the second eigenvector of L (i.e. the one with smallest non-zero eigenvalue). This is computed with the Lanczos method (see for example Stoer and Bulirsch (2013)) and then discretized by selecting a threshold τ and defining $A := \{s_i | y_i > \tau\}$. One could theoretically use the successive eigenvectors to further partition the sets, but the authors claim to have had better results by recomputing the second eigenvector of the Laplacian matrices of the subgraphs $(A, E_{|A \times A})$ and $(\mathcal{S} \setminus A, E_{|\mathcal{S} \setminus A \times \mathcal{S} \setminus A})$ (which are submatrices of L).

For the threshold level τ used to discretize the second eigenvector of L Shi and Malik propose to use either 0, the median value or the value such that the resulting partition has the minimum value of $Ncut_A$. The implementation of spectral clustering in the `scikit-learn` python library (which we will use in our numerical tests) uses by default k-means to threshold these values. All possible choices generally give very similar values of τ, all very close to 0.

Finally we wish to remark how the eigenvectors of L can be used as basis functions to expand real-valued functions defined on the vertices of the graph. This is done analogously to the Fourier transform for functions defined on $\mathbb{R}^n$, which consists of writing the original function in terms of the eigenvectors of the differential Laplace operator (complex exponentials). If one defines the *zero crossings* of a real-valued function f defined on the vertices of a graph $G = (V, E)$ as the set of edges connecting vertices that f maps to values with different sign, then it can be seen that eigenvectors corresponding to larger eigenvalues have larger numbers of zero-crossings. This gives a further insight into the spectral clustering procedure: the second eigenvector of L gives the optimal[3] cut in the graph and it also represents the slowest varying signal (if we exclude the all-positive first eigenvector) with respect to the topology induced by the similarity measure, i.e. the edge weights. For a general introduction to the field of signal processing on graphs see Shuman et al. (2013).

1.7.4 Felzenszwalb-Huttenlocher Clustering

Another graph-based clustering method is the Felzenszwalb-Huttenlocher method (we'll use FH for brevity) which in Felzenszwalb and Huttenlocher (2004) was also originally developed for segmenting images. While spectral clustering is a top-down method (recursively partitioning the set of vertices into two subsets) the FH method is bottom-up: each point in the graph starts as its own cluster and at each iteration step two clusters are examined and a decision as to whether merge them is taken. This decision depends on two quantities (defined below): the internal difference of each cluster and the difference between the two, which are two further measures of intra and inter class variance respectively. For ease of notation and consistency with the original paper, as explained in subsection 1.7.2 we will suppose without loss of generality that here the weights in the graph represent a dissimilarity: two data points that are very similar will be connected by an edge with a low weight and vice versa. Given any subset of vertices $A \subset V$ its *internal difference* is defined as

$$\mathrm{Int}(A) := \max_{e \in \mathrm{MST}(A,E)} w_e \ , \tag{1.7.11}$$

where $\mathrm{MST}(A, E)$ is the minimum spanning tree[4] (MST) of the subgraph with vertices A and all edges in E that are between vertices in A, i.e. the MST of $(A, E \cap A \times A)$. With a slight abuse in notation we indicate with $e \in \mathrm{MST}(A, E)$ an edge of the MST and with w_e the associated weight. The quantity $\mathrm{Int}(A)$ measures the intra variance in A because it tells us that there is a path between any two vertices in A using only edges of weight smaller than $\mathrm{Int}(A)$. Furthermore, given two regions A_1 and A_2, the authors

---

[3] apart from the approximation introduced by relaxing the discreteness constraint

[4] the covering tree (i.e. including all vertices) with minimum cumulative edge weight

define the *difference* between the two regions as

$$\mathrm{Diff}(A_1, A_2) := \min_{(i,j)\in A_1\times A_2\cap E} w_{ij} \ . \tag{1.7.12}$$

Finally given two regions A_1 and A_2 they define a predicate which evaluates to `True` if there is evidence of a boundary between the two regions and to `False` otherwise:

$$D(A_1, A_2) := \begin{cases} \texttt{True} & \text{if } \mathrm{Diff}(A_1, A_2) > \min\{\mathrm{I}_k(A_1), \mathrm{I}_k(A_2)\} \\ \texttt{False} & \text{otherwise} \end{cases} , \tag{1.7.13}$$

where

$$\mathrm{I}_k(A) := \mathrm{Int}(A) + \frac{k}{|A|} \tag{1.7.14}$$

is the *scaled internal difference* of a region. The scaling parameter k is introduced to give a preference to regions of cardinality not too small with respect to k. In other words comparing `Diff`(A_1, A_2) with the scaled internal differences instead of the internal differences makes it harder to find evidence for a boundary if one (or both) of the regions considered have very small cardinality; it is a way to discourage clusters with too few elements. Note that instead of the term $\frac{k}{|A|}$ any positive function $\tau(A)$ could be used: the segmentation will then give a preference to regions A for which $\tau(A)$ is big.
With all these quantities set in place, the authors propose a procedure similar to the classic Kruskal algorithm for computing the MST: starting with each vertex defined as its own cluster they iterate through the edges by increasing weight and consider the predicate $\mathrm{D}(A_1, A_2)$, where A_1 and A_2 are the current clusters containing the vertices on the two sides of the edge. If the predicate evaluates to `False`, then the two regions are merged. Because the edges are being considered by increasing weight, there is no need to evaluate the minimum in the definition of $\mathrm{Diff}(A_1, A_2)$: the predicate can be evaluated simply by comparing the current edge weight with $\min\{\mathrm{I}_k(A_1), \mathrm{I}_k(A_2)\}$. See Algorithm (2). The method is very fast and the dominating computational cost is given by ordering E by edge weight, which can be done in $O(|E| \log |E|)$ time.

Algorithm 2 Felzenszwalb-Huttenlocher clustering algorithm

```
Input: Graph $G = (V, E) = (\mathcal{S}, E)$
Output: $C = (A_1, \dots, A_r)$ clustering of $V$, i.e. $V = A_1 \sqcup \dots \sqcup A_r$
Sort $E$ into $\pi = (o_1, \dots, o_m)$ by non-decreasing edge weight
Set $C^0 = (\{s_0\}, \dots, \{s_N\})$, i.e. each vertex is its own cluster
for $i = 1, \dots, m$ do
    Consider the vertices $s_h$ and $s_l$ of the $i$-th edge, i.e. $o_i = (s_h, s_l)$
    Let $A_h^{i-1}$ and $A_l^{i-1}$ be the components of $C^{i-1}$ containing $s_h$ and $s_l$ respectively
    if $A_h^{i-1} \neq A_l^{i-1}$ and $w_{hl} \leq \min\{\mathrm{I}_k(A_h^{i-1}), \mathrm{I}_k(A_l^{i-1})\}$ then
        $C^i$ is obtained from $C^{i-1}$ by merging $A_h^{i-1}$ and $A_l^{i-1}$
    else
        $C^i = C^{i-1}$
    end if
end for
Set $C = C^m$
```

2 Tree-based Haar-wavelet Generalizations

In this Chapter we want to present a scheme for adaptive generalizations of the Haar wavelet transform; we will then see in chapters 3 and 5 two actual implementations of such a scheme, used for sparse image representation and dictionary learning respectively. The first insight is to consider what we'll term the *Haar dependency tree* associated to a wavelet transform, similarly as to what proposed in Ram et al. (2011) and Murtagh (2007) (see also Gavish et al. (2010)). In multi-level wavelet transforms, the coefficients at one level are computed as the convolution between the coefficients at the previous level and a pair of filters of finite support. Each coefficient will then be a linear combination of only s coefficients at the previous level, where s is the length of the filter support. These dependencies between coefficients at different levels are represented as edges between nodes in the tree, where each node corresponds to a coefficient.

We will be mainly interested in the simplest case, that is ℓ levels of a Haar wavelet transform. If applied to a one-dimensional signal of length 2^ℓ the associated tree is a complete binary tree, where the leafs correspond to the samples and all other nodes to an approximation coefficient, the root node being the global average. We will see how, due to linearity, it is equivalent building the tree starting from the leafs (as is usually done) or from the root node.

With this scheme in mind, we are interested in two approaches to make the transform adaptive, which can be seen as particular cases of a single more general framework. The first consists of using a clustering procedure to successively partition or agglomerate the samples, depending on whether we're building the tree starting from the root node or the leafs; we will be mainly interested in the former. In Chapter 5 we will use this procedure to develop a dictionary learning method. The second consists of inserting a permutation step before the convolution and downsampling operations of the wavelet transform, which in the tree representation corresponds to inserting bipartite subgraphs in between nodes corresponding to different levels of the transform; in this case we are building the tree from the bottom up. In Chapter 3 we will be using this procedure to design an adaptive wavelet transform for images.

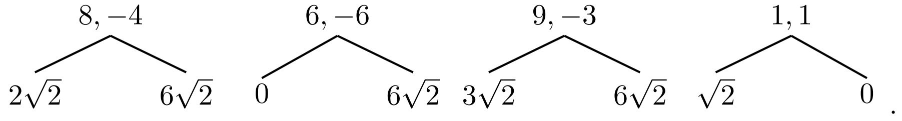
We will start by going through a simple example, namely the 3-level one-dimensional Haar wavelet transform of the signal

$$z = \sqrt{2}\ [2, 6, 0, 6, 3, 6, 1, 0] \in \mathbb{R}^8$$

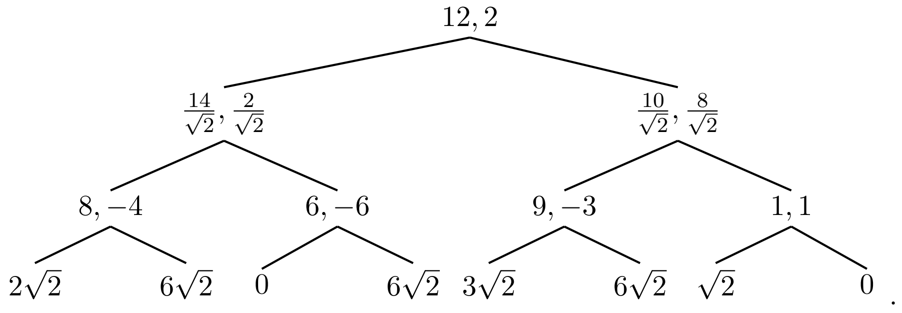
which we already computed in Example 4 at p. 20. The main concepts we are interested in were inspired by this simple case, and we think this approach will aid the more abstract theory in the rest of the Chapter. We associate each level of the wavelet transform to a level of a binary tree, where the nodes correspond to the approximation coefficients. The tree is constructed from the leaf nodes upwards: at the third level of the tree there is a leaf node for each sample in $z = \{a_{3,k}\}_{k=0,\dots,7}$. The convolution of the values in $\{a_{3,k}\}_{k=0,\dots,7}$ with the low and high pass wavelet filters (which are both of length 2 in the Haar case) and subsequent downsampling give origin to $\{a_{2,k}\}_{k=0,\dots,3}$ and $\{d_{2,k}\}_{k=0,\dots,3}$:

$$\text{for } k = 0, \dots, 3 \qquad \begin{aligned} a_{2,k} &= \frac{1}{\sqrt{2}}\big(a_{3,2k} + a_{3,2k+1}\big) \\ d_{2,k} &= \frac{1}{\sqrt{2}}\big(a_{3,2k} - a_{3,2k+1}\big)\ , \end{aligned}$$

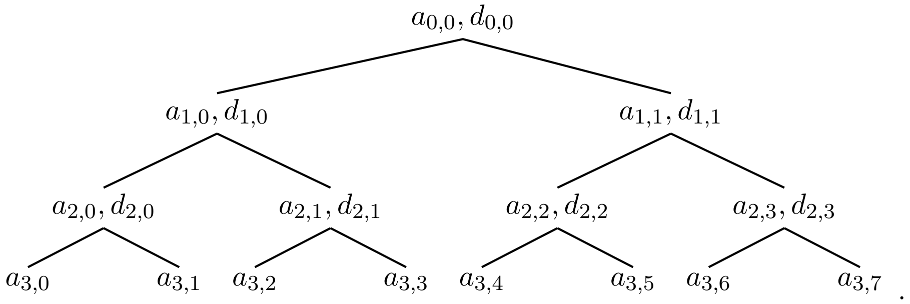
which are half the length of $\{a_{3,k}\}_{k=0,\dots,7}$ (see also equation (1.2.25) at page 19). For each value v in $\{a_{2,k}\}_{k=0,\dots,3}$ a new node in the tree is created, and edges are added from it to the nodes corresponding to the values in $\{a_{3,k}\}_{k=0,\dots,7}$ of which v is a weighted average. So after the first level of the transform we have this forest of 4 trees, where for each non-leaf node we are labeling the node with the couple (a, d) consisting of the respective approximation and detail wavelet coefficients:



We can then extend this process to further levels, obtaining a forest of 2 trees (each now with 4 leafs) associated to the computation of a_1, and finally a single tree (with 8 leafs) for the computation of all the 3 levels of the transform:



If we label the nodes in the tree with $(a_{j,k}, d_{j,k})$ where j represents the level and k the component of the vector, we obtain the following tree describing the dependency structure between coefficients at different levels of a Haar wavelet transform for any vector in $\mathbb{R}^8$:



There are three reasons why we are interested in this construction:

1. it gives some intuition as to why the normal bottom-up procedure of computing $\{a_{2,k}\}_{k=0,\dots,3}$ and $\{d_{2,k}\}_{k=0,\dots,3}$ from z, $\{a_{1,k}\}_{k=0,\dots,1}$ and $\{d_{1,k}\}_{k=0,\dots,1}$ from $\{a_{2,k}\}_{k=0,\dots,3}$ and so on is equivalent to a top-down procedure: since to go up one level in the tree we do a weighted average of the level below, due to linearity it must be possible to express the values of each node as a weighted average of the values in the leafs that are sons to it. In fact we will see in Proposition 2.0.7 (and more specifically in its Corollary) that if we know z, we can for example directly compute $a_{0,0}$ and $d_{0,0}$ as

$$\begin{aligned} a_0 &= \frac{\sqrt{2}^3}{8} \sum_{i=1}^{8} z_i \\ d_0 &= \frac{\sqrt{2}^3}{8} (\sum_{i=1}^{4} z_i - \sum_{i=5}^{8} z_i) \ . \end{aligned}$$

2. The concept of the tree allows for at least two generalizations which insert data-adaptivity into the transform: these are obtained by allowing non-uniform partitionings of the data (which lead to non-complete binary trees) and adding a permutation layer in each layer of the tree. We will detail these in Sections 2.2 and 2.3.

3. The tree clearly illustrates the relation of dependency between coefficients: the second value of a_1 (i.e. $\frac{10}{\sqrt{2}}$) is totally independent from the first half of the vector z, i.e. it is not affected by changes in the values of $z_1, \dots, z_4$. We will use this concept in the case of the Region Based Easy Path Wavelet Transform (RBEPWT) for restricting the transform to a region of interest in an image.

We now wish to give a slightly more abstract description of the procedure above in order to build a framework that will encompass the classical one-dimensional Haar wavelet transform as well as the RBEPWT and tree-based dictionary constructions of Chapters 3 and 5, respectively. Let $\mathcal{S} = \{s_1, \dots, s_n\} \subset V$ be the finite collection of samples we wish to analyze, where V is some vector space. In the simplest case of a one-dimensional signal (e.g. audio or stock price), V is the real line $\mathbb{R}$; in the case of $m \times n$ patches extracted from a gray-valued image, V is $\mathbb{R}^{m \times n}$. We will be organizing the data in $\mathcal{S}$ in a binary tree $T = (N, E)$ structure, where each node $\nu \in N$ corresponds to some subset $\mathcal{S}_\nu \subset \mathcal{S}$ and the edges in E describe a hierarchy of these sets with respect to the $\subset$ partial order. We will suppose that a root node r has been fixed, and given a node ν we will call ν_0 and ν_1 his two sons. More generally we will use binary strings to indicate the path in the tree starting from a certain node: ν_{010} and ν_{011} are the left and right sons of node ν_{01} which can be reached from ν by picking the right son of its left son. We furthermore define a *level* function $\ell : N \to \mathbb{N}$ which associates to any node its level in the tree:

$$\ell(\nu) = \begin{cases} 0 & \text{if } \nu = r \\ \ell(\mu) + 1 & \text{if } \nu = \mu_0 \text{ or } \nu = \mu_1 \ . \end{cases}$$

The *depth* of the tree can then be defined as $\Delta = \max_{\nu \in N} \ell(\nu)$. We can use the level sets of ℓ to define the *slices*

$$N_\lambda = \{\nu \in N | \ell(\nu) = \lambda\} \ . \tag{2.0.1}$$

We will call N_ℓ the set of leaf nodes (i.e. the only nodes in N that have no sons); note that $N_\Delta \subseteq N_\ell$ is always true, but the opposite inclusion holds only in the special case of the tree being complete, i.e., when all leafs are at the same (maximum) level. It is clear that we can write N as the disjoint union of the slices N_λ:

$$N = \sqcup_{\lambda=0}^{\Delta} N_\lambda \ .$$

Finally we stress that if $\nu \in N_\lambda \setminus N_\ell$ then its sons ν_0, ν_1 are in $N_{\lambda+1}$. We can now give the following definition, which we will soon clarify with a detailed example:

Definition 2.0.2. Given a set of samples $\mathcal{S} = \{s_1, \ldots, s_n\} \subset V$ in a vector space V, we will call a *Haar dependency tree* any binary tree $T = (N, E)$ with root r in which every node ν corresponds to a subset of the samples $\mathcal{S}_\nu \subseteq \mathcal{S}$ and where two functions $\mathcal{A}, \mathcal{D} : N \to V$ are defined s.t.:

1. the edges of the tree describe a hierarchy between the sets $\mathcal{S}_\nu$ for the $\subset$ partial order. Namely each $\mathcal{S}_\nu$ is partitioned into the two sets corresponding to its sons:

$$\forall \nu \in N \setminus N_\ell,\ \mathcal{S}_\nu = \mathcal{S}_{\nu_0} \sqcup \mathcal{S}_{\nu_1} \ .$$

2. The following holds

$$\forall \nu \in N,\ \mathcal{A}(\nu) := \begin{cases} \frac{K_A}{|\mathcal{S}_\nu|} \Big(|\mathcal{S}_{\nu_0}| \mathcal{A}(\nu_0) + |\mathcal{S}_{\nu_1}| \mathcal{A}(\nu_1) \Big) & \text{if } \nu \in N \setminus N_\ell \\ \alpha(\mathcal{S}_\nu) & \text{if } \nu \in N_\ell \ , \end{cases} \tag{2.0.3}$$

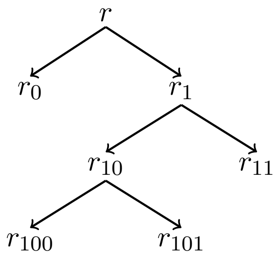
$$\forall \nu \in N \setminus N_\ell,\ \mathcal{D}(\nu) := \frac{K_D}{|\mathcal{S}_\nu|} \Big(|\mathcal{S}_{\nu_0}| \mathcal{A}(\nu_0) - |\mathcal{S}_{\nu_1}| \mathcal{A}(\nu_1) \Big) \ , \tag{2.0.4}$$

where K_A and K_D are some real costants and $\alpha : 2^{\mathcal{S}} \to V$ some function which associates a representative vector to any set of vectors (for all practical purposes we will use the sample mean, see below).

We will denote such a Haar dependency tree with $(T, r, \{\mathcal{S}_\nu\}_{\nu \in N}, \mathcal{A}, \mathcal{D})$.

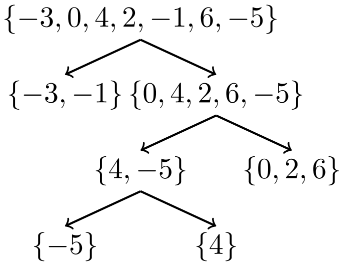
Equations (2.0.3) and (2.0.4) resemble those for computing approximation and detail coefficients in the classical one-dimensional Haar wavelet transform, and in fact in that case they will reduce to these (see Subsection 2.1). Note however that here for a given node ν, $\mathcal{A}(\nu)$ and $\mathcal{D}(\nu)$ are elements of V, which is not in general a field; thus it doesn't make sense to think of these values as coefficients that multiply basis functions, as is the case for the wavelet transform. We will therefore refer to functions $\mathcal{A}$ and $\mathcal{D}$ as *approximation* and *detail vectors* associated to the nodes of the tree (but still use the term *coefficients* when $V = \mathbb{R}$).

Example 5. Suppose we are given the following tree



,

which partitions the sample set $\mathcal{S} = \{-3, 0, 4, 2, -1, 6, -5\}$ in the following way:



.

Suppose $K_A = K_D = 1$ and $\alpha(\mathcal{S}_\nu) = \frac{1}{|\mathcal{S}_\nu|} \sum_{w \in \mathcal{S}_\nu} w$; we can then compute (starting from the leafs) the values of $\mathcal{A}$ and $\mathcal{D}$ according to (2.0.3) and (2.0.4):

$$\begin{aligned}
\mathcal{A}(r_{100}) &= -5 \\
\mathcal{A}(r_{101}) &= 4 \\
\mathcal{A}(r_{11}) &= \frac{1}{3}(0 + 2 + 6) \\
&= \frac{8}{3}
\end{aligned}$$

$$
\begin{aligned}
\mathcal{A}(r_0) &= \frac{1}{2}(-3-1)\\
&= -2\\
\mathcal{A}(r_{10}) &= \frac{1}{2}(|\{-5\}|\mathcal{A}(r_{100}) + |\{4\}|\mathcal{A}(r_{101}))\\
&= \frac{1}{2}(-5+4)\\
&= -\frac{1}{2}\\
\mathcal{D}(r_{10}) &= \frac{1}{2}(|\{-5\}|\mathcal{A}(r_{100}) - |\{4\}|\mathcal{A}(r_{101}))\\
&= \frac{1}{2}(-5-4)\\
&= -\frac{9}{2}\\
\mathcal{A}(r_1) &= \frac{1}{5}(|\{4,-5\}|\mathcal{A}(r_{10}) + |\{0,2,6\}|\mathcal{A}(r_{11}))\\
&= \frac{1}{5}\left(2(-\frac{1}{2}) + 3(\frac{8}{3})\right)\\
&= \frac{7}{5}\\
\mathcal{D}(r_1) &= \frac{1}{5}(|\{4,-5\}|\mathcal{A}(r_{10}) - |\{0,2,6\}|\mathcal{A}(r_{11}))\\
&= \frac{1}{5}\left(2(-\frac{1}{2}) - 3(\frac{8}{3})\right)\\
&= -\frac{9}{5}\\
\mathcal{A}(r) &= \frac{1}{7}(|\{-3,-1\}|\mathcal{A}(r_0) + |\{0,4,2,6,-5\}|\mathcal{A}(r_1))\\
&= \frac{1}{7}\left(2(-2) + 5(\frac{7}{5})\right)\\
&= \frac{3}{7}\\
\mathcal{D}(r) &= \frac{1}{7}(|\{-3,-1\}|\mathcal{A}(r_0) + |\{0,4,2,6,-5\}|\mathcal{A}(r_1))\\
&= \frac{1}{7}\left(2(-2) - 5(\frac{7}{5})\right)\\
&= -\frac{11}{7}\ .
\end{aligned}
$$

Note that $\mathcal{A}(r) = \frac{3}{7} = \frac{1}{7}\sum_{s\in\mathcal{S}} s$. ◇

Like in the classical wavelet transform we have synthesis formulas: with formulas (2.0.3) and (2.0.4) we can compute the values of $\mathcal{A}(\nu)$ and $\mathcal{D}(\nu)$ from the values of $\mathcal{A}(\nu_0)$ and $\mathcal{A}(\nu_1)$ (i.e. we're moving up the tree), but we can also compute $\mathcal{A}(\nu_0)$ and $\mathcal{A}(\nu_1)$

from $\mathcal{A}(\nu)$ and $\mathcal{D}(\nu)$ (i.e. going down the tree):

Proposition 2.0.5. *Given a Haar dependency tree* $(T, r, \{\mathcal{S}_\nu\}_{\nu \in N}, \mathcal{A}, \mathcal{D})$*, the following holds for any node* $\nu \notin N_\ell$*:*

$$\begin{aligned}
\mathcal{A}(\nu_0) &= \frac{|\mathcal{S}_\nu|}{2|\mathcal{S}_{\nu_0}|}\left(\frac{1}{K_A}\mathcal{A}(\nu) + \frac{1}{K_D}\mathcal{D}(\nu)\right) \\
\mathcal{A}(\nu_1) &= \frac{|\mathcal{S}_\nu|}{2|\mathcal{S}_{\nu_1}|}\left(\frac{1}{K_A}\mathcal{A}(\nu) - \frac{1}{K_D}\mathcal{D}(\nu)\right) \ .
\end{aligned} \tag{2.0.6}$$

Proof. Simply substitute (2.0.3) and (2.0.4) into the right hand sides of equations (2.0.6). ■

Thus if we know the value of $\mathcal{A}$ and $\mathcal{D}$ for a certain node ν, we can use these equations to determine the values for all the nodes in the subtree rooted in ν, including the values of α on the leaf nodes. This is possible in the important special case, which we will suppose to be in from now on, of α being the sample mean, i.e. $\alpha(\mathcal{S}_\nu) = \frac{1}{|\mathcal{S}_\nu|}\sum_{w \in \mathcal{S}_\nu} w$. We have the following:

Proposition 2.0.7. *If* α *is the sample mean, then for any node* $\nu \in N$ *such that the binary tree rooted in* ν $T_\nu = (N_\nu, E \cap N_\nu \times N_\nu)$ *is complete (where* N_ν *is the subset of nodes that are successors to* ν*) we have*

$$\begin{aligned}
\mathcal{A}(\nu) &= \frac{K_A^\delta}{|\mathcal{S}_\nu|}\sum_{w \in \mathcal{S}_\nu} w = K_A^\delta \alpha(S_\nu) \\
\mathcal{D}(\nu) &= \frac{K_D K_A^{\delta-1}}{|\mathcal{S}_\nu|}\Big(\sum_{w \in \mathcal{S}_{\nu_0}} w - \sum_{w \in \mathcal{S}_{\nu_1}} w\Big) \ ,
\end{aligned} \tag{2.0.8}$$

where δ *is the depth of* T_ν*.*

Proof. We prove (2.0.8) by induction on δ. The base case $\delta = 0$ (i.e. $\nu \in N_\ell$) follows from definition (2.0.3). If the statement is true for δ, take $\nu \in N$ such that T_ν has depth $\delta + 1$. Since T_ν is complete we know that T_{ν_0} and T_{ν_1} have both depth δ, thus we have:

$$\begin{aligned}
A(\mathcal{S}_\nu) &= \frac{K_A}{|\mathcal{S}_\nu|}\big(|\mathcal{S}_{\nu_0}|A(S_{\nu_0}) + |\mathcal{S}_{\nu_1}|A(S_{\nu_1})\big) \\
&= \frac{K_A}{|\mathcal{S}_\nu|}\big(|\mathcal{S}_{\nu_0}|K_A^\delta \alpha(S_{\nu_0}) + |\mathcal{S}_{\nu_1}|K_A^\delta \alpha(S_{\nu_1})\big) \\
&= \frac{K_A^{\delta+1}}{|\mathcal{S}_\nu|}\Big(\sum_{w \in \mathcal{S}_{\nu_0}} w + \sum_{w \in \mathcal{S}_{\nu_1}} w\Big) \\
&= K_A^{\delta+1}\alpha(S_\nu) \ .
\end{aligned}$$

The formula for $\mathcal{D}(\nu)$ follows simply by substituting (2.0.8) into definition (2.0.4). ■

Corollary 2.0.9. *If the Haar dependency tree is complete (e.g. in the classical one-dimensional Haar transform) then equations* (2.0.8) *hold for all the nodes in the tree.*

If we want to drop the hypotheses on the subtree being complete, we have to require $K_A = K_D = 1$. In fact, given a node ν in the tree, we know that $\mathcal{A}(\nu)$ will be some weighted average of the samples in $\mathcal{S}_{\nu_0}$ and $\mathcal{S}_{\nu_1}$, but we don't know these weights exactly because they are of the form K_A^γ where γ depends on the depth and structure of the subtrees centered in ν_0 and ν_1, which we know nothing about. We have:

Proposition 2.0.10. *If α is the sample mean and $K_A = K_D = 1$, then for any node $\nu \in N$ we have:*

$$\begin{aligned} \mathcal{A}(\nu) &= \alpha(S_\nu) \\ \mathcal{D}(\nu) &= \frac{1}{|\mathcal{S}_\nu|}\Big(\sum_{w \in \mathcal{S}_{\nu_0}} w - \sum_{w \in \mathcal{S}_{\nu_1}} w \Big) \ . \end{aligned} \tag{2.0.11}$$

Proof. The proof goes exactly as the one for Proposition (2.0.7), with the exception that in the induction step we don't need for T_{ν_0} and T_{ν_1} to have the same depth since $K_A^\delta = K_D^\delta = 1$ for any δ. ■

2.1 Classical one-dimensional wavelet Haar transform

Let $V = \mathbb{R}$, $K_A = K_D = \sqrt{2}$ and α be the sample mean. Let's suppose that $n = 2^L$ with $L \in \mathbb{N}$ and that we want to apply exactly L levels of the wavelet transform. Like shown in the example in the beginning of the Section, the tree is built from the bottom up: for each s_i in the sample set $\mathcal{S} = \{s_1, \ldots, s_N\}$ we define a leaf node λ_i. The sets associated to the leafs nodes are the singletons with the corresponding samples, i.e. $\mathcal{S}_{\lambda_i} = \{s_i\}$. We then iterate through the leafs and for each pair of nodes λ_{2j} and λ_{2j+1} we add a node ν_j that has them as sons. From formulas (2.0.3) and (2.0.4) we obtain

$$\begin{aligned} \mathcal{A}(\nu_j) &= \frac{\sqrt{2}}{2}\big(\alpha(\mathcal{S}_{\lambda_{2j}}) + \alpha(\mathcal{S}_{\lambda_{2j+1}})\big) \\ &= \frac{1}{\sqrt{2}}\big(s_{2j} + s_{2j+1}\big) \text{ and} \\ \mathcal{D}(\nu_j) &= \frac{\sqrt{2}}{2}\big(\alpha(\mathcal{S}_{\lambda_{2j}}) - \alpha(\mathcal{S}_{\lambda_{2j+1}})\big) \\ &= \frac{1}{\sqrt{2}}\big(s_{2j} - s_{2j+1}\big) \ , \end{aligned} \tag{2.1.1}$$

which are the classical analysis formulas for the Haar wavelet transform (see (1.2.25)). Thus, as expected, in this case $\mathcal{A}$ and $\mathcal{D}$ represent exactly the approximation and detail coefficients. We can of course iterate this procedure and build the whole tree like we did in the example at the beginning of the Section. In this case if $\nu \in N_{\Delta-k}$ then the set $\mathcal{S}_\nu$ is given by a window on the signal of length 2^k: $\{s_h, \ldots, s_{h+2^k-1}\}$. Since the samples lie on the real line the nodes at all levels are naturally ordered, a fact that allows us to specify exactly this window. We have:

Lemma 2.1.2. *If ν_j is the j-th node at level $\Delta - k$ then* $\mathcal{S}_{\nu_j} = \{s_{j2^k}, \ldots, s_{j2^k+2^k-1}\}$.

Proof. By induction on k: if $k = 0$ then ν_j is a leaf node and thus by definition $\mathcal{S}_{\nu_j} = \{s_j\}$. Suppose now the formula is true for $k-1$ and take ν_j to be the j-th node in $N_{\Delta-k}$. From (2.1.1) we have $\mathcal{S}_{\nu_j} = \mathcal{S}_{\mu_{2j}} \cup \mathcal{S}_{\mu_{2j+1}}$ where μ_{2j} and μ_{2j+1} are respectively the $2j$-th and $2j+1$-th nodes in $N_{\Delta-k+1}$. Thus by using the induction hypothesis, noticing the last element of $\mathcal{S}_{\mu_{2j}}$ is the predecessor to the first element of $\mathcal{S}_{\mu_{2j+1}}$ we obtain

$$\begin{aligned}\mathcal{S}_{\nu_j} &= \{s_{2j2^{k-1}}, \ldots, s_{2j2^{k-1}+2^{k-1}-1}\} \cup \{s_{(2j+1)2^{k-1}}, \ldots, s_{(2j+1)2^{k-1}+2^{k-1}-1}\} \\ &= \{s_{j2^k}, \ldots, s_{j2^k+2^k-1}\} .\end{aligned}$$

■

Using the Lemma we can write Proposition 2.0.7 in the notation of Section 1.2 for the approximation and detail coefficients of the classical Haar wavelet transform:

Proposition 2.1.3. *Given the L-levels Haar wavelet transform of the signal* $[s_1, \ldots, s_{2^L}]$, *for any $k = 1, \ldots, L$ we can write the j-th component of a_{L-k} (d_{L-k}) as a uniformly weighted sum of 2^k components of the samples:*

$$a_{L-k}(j) = 2^{-\frac{k}{2}} \sum_{h=j2^k}^{j2^k+2^k-1} s_h \tag{2.1.4}$$

$$d_{L-k}(j) = 2^{-\frac{k}{2}} \left(\sum_{h=j2^k}^{j2^k+2^{k-1}-1} s_h - \sum_{h=j2^k+2^{k-1}}^{j2^k+2^k-1} s_h \right) . \tag{2.1.5}$$

The underlying reason that permits us to proove this Proposition is of course the linearity and invertibility of the analysis and synthesis formulas: basically we have simply found a closed form for the recurrent equations (1.2.25).

Finally we wish to discuss what happens when we compute fewer than L levels of the transform. Starting with the leaf nodes λ_i, for each level of the transform a new level of nodes is added which contains half the nodes of the preceding level. If we are computing L levels we will go all the way to the root node r. If we stop before, we will have a multi-rooted tree, or equivalently a set of disjoint complete binary trees: if we compute only l levels of the transform we will have $n/2^l$ trees, each with 2^l leaf nodes. The trees being disjoint means that the values of approximation and detail coefficients at upper levels will only be influenced by the value of the leaf nodes in that very same tree. Thus the computation of the tree is completely decoupled into $n/2^l$ independent computations of trees corresponding to only a fraction of the samples.

2.2 Adaptive dictionary

Let α be the sample mean, $K_A = K_D = 1$ and the tree is built top-down. There are no restrictions on the vector space V as long as we have some clustering method $\mathcal{C} : 2^{\mathcal{S}} \to 2^{\mathcal{S}} \times 2^{\mathcal{S}}$ that partitions any subset of data in two, i.e.

$$\forall S \subset \mathcal{S} \quad S = \mathcal{C}(S)_0 \sqcup \mathcal{C}(S)_1 \ .$$

We start from the root node, i.e. we apply $\mathcal{C}$ to the whole set $\mathcal{S}$ obtaining nodes r_0 and r_1. We define $\mathcal{S}_{r_0} := \mathcal{C}(\mathcal{S})_0$ and $\mathcal{S}_{r_1} := \mathcal{C}(\mathcal{S})_1$. In general given node ν, if $\mathcal{S}_\nu$ does not satisfy a stopping condition[1], we will add to the tree the son nodes ν_0 and ν_1 with associated sets $\mathcal{S}_{\nu_0} := \mathcal{C}(\mathcal{S}_\nu)_1$ and $\mathcal{S}_{\nu_1} := \mathcal{C}(\mathcal{S}_\nu)_2$.

Differently than in the other two cases (classical Haar transform and RBEPWT), the tree here isn't generally complete: leaf nodes may have different levels. If only a certain number of tree nodes is required for, the tree structure depends also on details of the tree visiting method and branching criteria: see Subsection 5.3.1. As a general guideline, the deeper a subtree is the smaller cardinalities will have its leafs, with the extreme being that each leaf corresponds to a singleton set containing just one sample.

In Chapter 5 we will be interested specifically in the case $V = \mathbb{R}^m \times \mathbb{R}^n$: the data points will be small $m \times n$ patches extracted from one or multiple gray-valued images. We will use all the detail vectors from the tree and $\mathcal{A}(r)$ as an overcomplete dictionary and will be able to control the cardinality of the dictionary both directly (limiting the number of visited tree nodes) or indirectly (branching on a node ν only if $\mathcal{S}_\nu$ is big enough): see Subsection 5.3.1.

2.3 RBEPWT

For the RBEPWT (Region Based Easy Path Wavelet Transform) in Chapter 3 we will be building the tree bottom-up: let $V = \mathbb{R}$, $K_A = K_D = \sqrt{2}$ and α be the sample mean. The structure of the transform will be slightly more complex and the Haar dependency tree definition has to be modified. In fact, as we will explain in full detail in Chapter 3, the RBEPWT consists of vectorizing a two-dimensional gray-value image along a certain path and then applying one level of a wavelet transform (here we're always restricting ourselves to the Haar wavelet). The reordering of the data due to the path finding procedure corresponds to applying a permutation to the vectorized samples prior to convolving them with the wavelet analysis filters. We can model this procedure with the Haar dependency tree by redefining the sons of a node to incorporate the permutation, obtaining a construction that is basically what is described in Ram et al. (2011). We thus need to modify definition (2.0.2) as follows:

Definition 2.3.1. A *Haar dependency tree with permutation* is a complete Haar dependency tree $(T, r, \{\mathcal{S}_\nu\}_{\nu \in N}, \mathcal{A}, \mathcal{D})$ together with a bijection $\sigma_\lambda : N_\lambda \to N_\lambda$ for every level

---

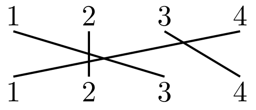
[1] for example if the sample variance of the data points in $\mathcal{S}_\nu$ is above a fixed threshold. See Chapter 5.

λ. The nodes at every level are indexed $N_\lambda = \{\nu_\lambda^0, \dots, \nu_\lambda^{K_\lambda}\}$ and the sons of a node $\nu := \nu_\lambda^j$ with $\lambda < \Delta$ are so defined:

$$\begin{aligned} \nu_0 &:= \nu_{\lambda+1}^{\sigma_{\lambda+1}^{-1}(2j)} , \\ \nu_1 &:= \nu_{\lambda+1}^{\sigma_{\lambda+1}^{-1}(2j+1)} . \end{aligned} \tag{2.3.2}$$

With this definition of sons that incorporates the permutation, points (1)-(2) of definition (2.0.2) must hold, where again $\nu := \nu_\lambda^j$. We will denote such a Haar dependency tree with permutation with $(T, r, \{\mathcal{S}_\nu\}_{\nu \in N}, \{\sigma_\lambda\}_{\lambda=0,\dots,\Delta}, \mathcal{A}, \mathcal{D})$.

To understand why the definition has to be altered in this way, it may be of help to visualize the permutation as a bipartite graph: create a set of nodes corresponding to the elements being permuted, then create a second identical set of nodes, adding an edge between a node from the first set and one from the second if the permutation brings the former element into the latter. For example we would represent the cyclical permutation $\sigma = (1\ 4\ 3)$ with:



.

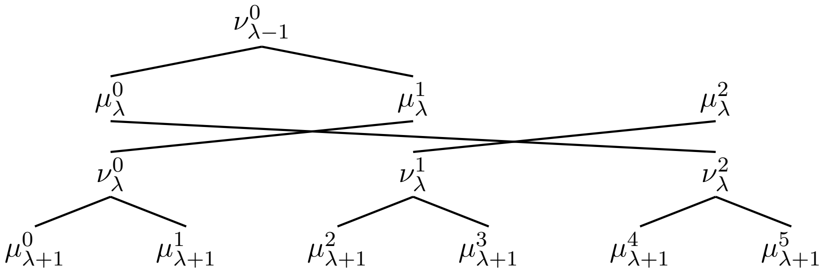
We can then think of inserting such a bipartite graph into every level of the Haar dependency tree: for every node ν_λ^j at level λ we create an additional twin node $\mu_\lambda^{\sigma_\lambda(j)}$, and a corresponding edge between these two nodes. These edges simply represent a reordering of the samples - we want the values of $\mathcal{A}$ and $\mathcal{D}$, as well as the sets $\mathcal{S}_{\nu_\lambda^j}$ and $\mathcal{S}_{\mu_\lambda^{\sigma_\lambda(j)}}$ to be the same on any node ν_λ^j and his twin node $\mu_\lambda^{\sigma_\lambda(j)}$. A node $\nu_{\lambda-1}^j$ will have as sons μ_λ^{2j} and μ_λ^{2j+1} on which the values of $\mathcal{A}(\nu_{\lambda-1}^j)$ and $\mathcal{D}(\nu_{\lambda-1}^j)$ will depend. These sons are in turn connected through the bipartite graph to their twin nodes, so we can actually write the values of $\mathcal{A}$ and $\mathcal{D}$ on $\nu_{\lambda-1}^j$ without introducing nodes of type μ_λ. For example we can write

$$\begin{aligned} \mathcal{D}(\nu_{\lambda-1}^j) &= \frac{K_D}{|\mathcal{S}_\nu|}\Big(|\mathcal{S}_{\mu_\lambda^{2j}}|\mathcal{A}(\mu_\lambda^{2j}) - |\mathcal{S}_{\mu_\lambda^{2j+1}}|\mathcal{A}(\mu_\lambda^{2j+1})\Big) \\ &= \frac{K_D}{|\mathcal{S}_\nu|}\Big(|\mathcal{S}_{\nu_\lambda^{\sigma_\lambda^{-1}(2j)}}|\mathcal{A}(\nu_\lambda^{\sigma_\lambda^{-1}(2j)}) - |\mathcal{S}_{\nu_\lambda^{\sigma_\lambda^{-1}(2j+1)}}|\mathcal{A}(\nu_\lambda^{\sigma_\lambda^{-1}(2j+1)})\Big) , \end{aligned}$$

and analogously for $\mathcal{A}$. This is exactly what we described more succinctly in definition (2.3.1). See Figure (2.1).

We will use this tree construction in Section 3.4 to separate the RBEPWT coefficients into two classes depending on whether they are responsible or not for the reconstruction of a certain region in the image; this will allow us to encode with different quality levels a Region of Interest and the rest of the image.

Figure 2.1: Part of a Haar dependency tree with permutation where $\sigma_\lambda = (0\ 1\ 2)$



3 The Region Based Easy Path Wavelet Transform

Instead of choosing an a priori frame to approximate the image, one can instead choose to adapt the frame elements to the particular image. Many different approaches to this concept have been studied in the recent years, such as bandlets Le Pennec and Mallat (2005), grouplets Mallat (2009) and dictionary learning methods (see Chapters 4 and 5). For a review of adaptive image representations see Peyré (2011). In Plonka (2009) a new adaptive transform for images termed *Easy Path Wavelet Transform* (EPWT) was introduced. In this method, a path is found among the points of the image that leverages local correlation in the gray values so as to produce a one-dimensional signal with low entropy. At every level of the transform such a path is found and then a one-dimensional wavelet transform is applied. The image quality obtained with this method is very good when compared to other methods; the main drawback comes from the need to store the paths for each level, which are needed during decoding. In Plonka et al. (2012) and Plonka et al. (2013) it was shown that, with a suitable choice of the paths, the N-term approximation given by the EPWT is optimal for piecewise-Hölder functions. In Plonka et al. (2011) the EPWT was used as part of a hybrid method for Image Approximation while in Heinen and Plonka (2012) for denoising of scattered data.

In recent years there has been a big interest in using wavelet-like transforms for natural image compression and denoising. One wishes to find an appropriate transform such that the most important information of the image is concentrated in few coefficients; by then thresholding coefficients and keeping only the largest ones, one hopes to obtain a good quality approximation of the original image. However simply using a two-dimensional tensor wavelet transform doesn't yield good results, mainly because edges are poorly preserved. This is due to the support of the basis elements not being adapted to the directional geometric properties present in natural images. Therefore transforms such as curvelets (Candes et al. (2006),Candès and Donoho (2004)) and shearlets (Guo et al. (2004),Guo and Labate (2007)) have been developed, which are highly redundant frames with strong anisotropic selectivity. However these are non-adaptive frames which loose their near optimal approximation properties if strong hypotheses are dropped on the edges in the image (namely piecewise C^2).

In Budinich (2017b) and Budinich (2017a) a variation on the original EPWT method was proposed, called the *Region Based Easy Path Wavelet Transform* (RBEPWT) method. The idea of this method is to reduce the adaptivity storage cost by not requiring to store the paths like in the EPWT. In order to achieve this, a segmentation method is applied to the image in a first step, in order to partition the image into areas of low variation of the gray values. Then, for each region, a path is found in some canonical manner:

the path depends only on the geometrical shape of the region's border and not on the gray-values of the inner pixels. In this way in the final encoding one needs to store only the regions obtained from the initial segmentation step and the wavelet coefficients; the paths, being a deterministic function of the region's shape, can be recomputed on-the-fly during decoding, and thus need not be stored. In this Chapter we review the aforementioned papers with the exception of subsection 3.3.3 which is original to this manuscript. The quality of the lossy compression obtained from decoding a hard-thresholded set of coefficients heavily depends on the initial segmentation. In this regard one would like a segmentation that finds regions where the local variance in the gray-values of the pixels is low - we do not care if the identified regions correspond to semantic areas in the image (which is the objective of many segmentation methods used in computer vision), but we wish instead to not have big jumps in the gray-values in the regions. We thus need a segmentation algorithm that essentially does a clustering: if we think of the image as a set of points in a three-dimensional space, where the first two coordinates are the indexes of the pixel and the third its gray-value, we wish for the segmentation to identify the main clusters of such points. We experimented with the Felzenszwalb-Huttenlocher algorithm (see Section 1.7.4) and the Texture and Boundary Encoding Segmentation (TBES) proposed in Rao et al. (2010).

3.1 General Framework

Both the original EPWT method and the RBEPWT conform to a common general framework which we will describe here. Let $f : I \to \{0, \dots, 255\}$ be the input image, where $I = \{(i, j) \mid 0 \leq i, j \leq n-1\}$ is the set of indexes and $n = 2^k$ for some $k \in \mathbb{N}$; assume I is ordered in some canonical manner, for example using the lexicographical ordering. Let $L \in \mathbb{N}$ be the number of levels of the transform; $2^L \leq n^2$ must hold. Choose a set of low-pass and high-pass 1-dimensional wavelet analysis and synthesis filters $h, g, \bar{h}, \bar{g}$ (see Section 1.2).
Define $I^L := I$ and $f^L := f$. The encoding for the first level consists of:

1. Finding a *path* in I^L; this can be thought equivalently as a function
$$p^L : \{0, 1, \dots, |I^L| - 1\} \to I^L \tag{3.1.1}$$
or as a permutation of the elements of I^L. How this path is found is the central point of such methods, and what differentiates the EPWT from the RBEPWT; we will comment more on this later. We can then define $\tilde{f}^L : \{0, 1, \dots, |I^L| - 1\} \to \{0, \dots, 255\}$ by $\tilde{f}^L := f^L \circ p^L$ as the 1-dimensional signal obtained from sampling the image along the path.

2. Apply one level of a periodic 1-dimensional discrete wavelet transform to $\tilde{f}^L$, obtaining approximation and detail coefficients a^L and d^L respectively (see equations

(1.2.20) and (1.2.21) at page 18):

$$\begin{aligned} a^L(k) &:= \tilde{f}^L \star \bar{h}(2k) \\ d^L(k) &:= \tilde{f}^L \star \bar{g}(2k), \quad k = 0, \ldots, \frac{|I^L|}{2} - 1 \ . \end{aligned} \tag{3.1.2}$$

3. Define $I^{L-1} := \{p^L(2k) \text{ s.t. } k = 0, \ldots, \frac{|I^L|}{2} - 1\}$; it is the subset of I^L obtained by taking only the coordinates corresponding to even indexes in the path p^L, i.e. it is I^L decimated by a factor of 2, following the order induced by p^L. Define $f^{L-1} : I^{L-1} \to \{0, \ldots, 255\}$ by $f^{L-1}(p^L(2k)) = a^L(k)$.

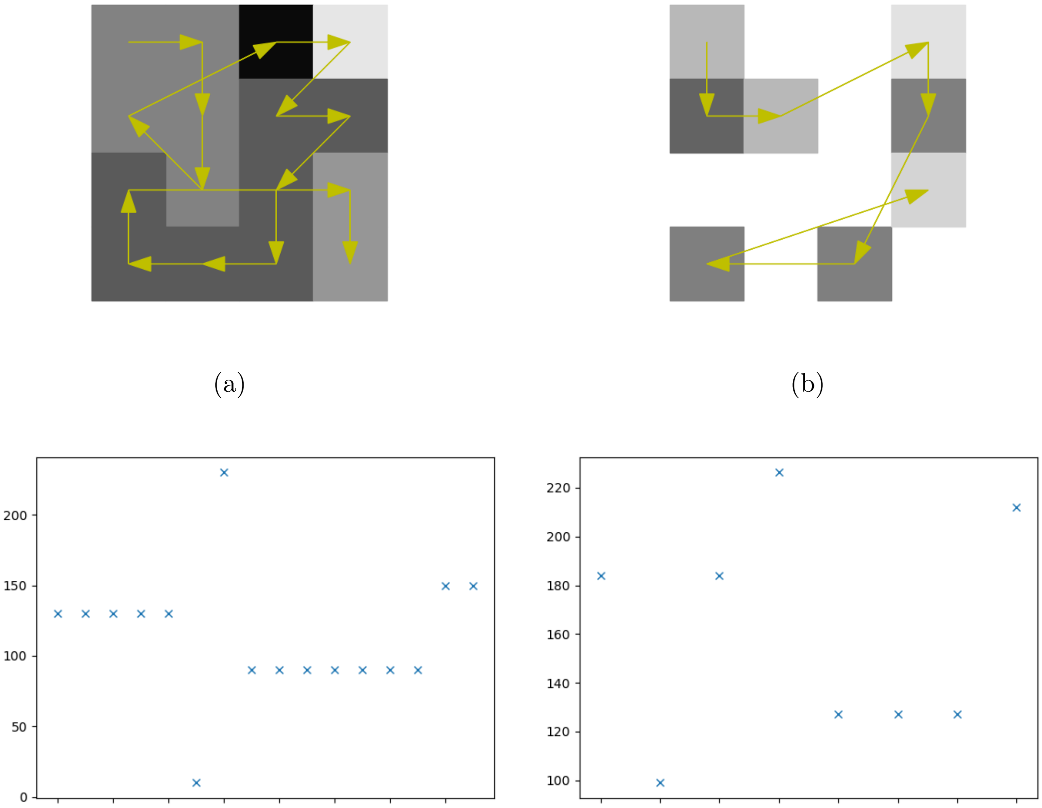
Now, I^{L-1} and f^{L-1} are a new set of indexes and a vector of values respectively, with half the points of I^L. We remark that I^{L-1} will not in general be of the form $\{(i,j) \mid 0 \leq i,j \leq k\}$, i.e. it won't be a set of uniformly spread points; the path procedure must thus not make such an assumption. One can then iterate the 3 steps of encoding, at each level halving the number of points. See Figure 3.1.
In summary the encoding steps of the transform consist of:

1. permuting the order of the points (i.e. find a path),
2. applying a discrete wavelet transform,
3. storing the detail vector and use the approximation vector as values for the new level, which is obtained by down-sampling the points along the found path.

Since we are interested in lossy compression, typically after the encoding a thresholding procedure will be used on the coefficients enforcing sparsity. Following the usual assumption that the coefficients with smallest absolute value in the encoded image are the least significant ones, we simply keep the N largest coefficients (in absolute value) and set the others to 0.
For decoding one needs the approximation vector for the lowest level, all the wavelet detail vectors and the permutations for each level. Then the decoding procedure consists simply of

1. applying the inverse wavelet transform for that level,
2. applying the inverse permutation for that level.

Note that, since we are only interested in the final result of the decoding (where the spatial disposition of the points is given by the canonical ordering chosen in advance), we do not need to know, for the intermediate levels, which value in the vector corresponds to which point in space. In other words, while during encoding we down-sample both the points in space and the vectors of values associated to them, in the decoding phase we can operate on the vectors only, upsampling and permuting them. The spatial information is needed only during encoding in order to find the path, i.e. the order according to which we vectorize the 2-dimensional data; during decoding this order (given by the inverse permutations) is already known.



(a) (b)

(c) (d)

Figure 3.1: Toy example of the general procedure for EPWT and RBEPWT: a 4×4 grey-value image is vectorized along a path and its next level points are obtained by downsampling by a factor of 2; here $L = 4$ and $n = 4$. (a), (b): the points in I^4 and I^3 respectively with the chosen paths shown as yellow arrows; the values of a^4 and a^3 are the greyvalues of the pixels. (c), (d): plots of the vectorized images along the paths for level 4 and 3 respectively, i.e. $\tilde{f}^4$ and $\tilde{f}^3$.

3.2 EPWT

In the EPWT method the path at level l starts from a canonical point (for example $(0,0)$) and at each step, among the closest points that are still avaiable (i.e. aren't already part of the path), it greedily chooses the point that gives the least difference in absolute value:

$$\begin{cases} p^l(0) & := (0,0) \\ p^l(k+1) & \in \arg\min_{(i,j) \in A_{\tilde{h}}(k)} |f^l(i,j) - f^l(p^l(k))| \end{cases} \tag{3.2.1}$$

$$\begin{aligned}&\text{where } \bar{h} := \min\{h \in \mathbb{N},\ A_h(k) \neq \emptyset\} \\ &\text{and } A_h(k) = B_h^\circ(p^l(k)) \bigcap \left(I^l \setminus \cup_{m=1}^{k-1} p^l(m)\right) .\end{aligned}$$

Here $B_h^\circ((i,j))$ denotes the punctured ball[1] centered in point (i,j) of radius h and $\tilde{h}$ is the minimum required radius to obtain a neighborhood of point (i,j) that contains points in I^l that have not yet been taken as previous points of the path. If the $\arg\min$ set contains more than one element, the choice can be done so as to minimize the direction change in the path.
In order to have access to the inverse permutations during encoding, the paths for each level have to be stored, alongside the wavelet detail and approximation coefficients. In Plonka (2009) it was shown that, with the same number of coefficients, the EPWT greatly outperforms the classical 2-dimensional wavelet transform. However the storage cost is strongly affected by the need to store all the permutations; this is the point the RBEPWT wishes to address.

3.3 RBEPWT

In the region based EPWT, we first apply a segmentation method to the image, in order to obtain a partition of the original image into *regions* (subsets of points). The rationale is that if we have regions where the variation in gray-values is small, then it is not so important which path is taken inside that region. We can then have a canonical path-finding procedure which does not depend on the gray-values. In this way the final encoding consists of the wavelet detail, the approximation coefficients and the segmentation. We need not to store all the paths, since these depend only on the segmentation and not on the pixel grayvalues and thus they can be recomputed on-the-fly during decoding.
We will comment in subsection 3.3.2 on the properties the segmentation method should have. For now suppose that the segmentation step identifies regions $R_0, R_1, \ldots, R_{r-1} \in \mathcal{P}(I)$ where I is the set of indexes in the image and $\mathcal{P}(I)$ denotes the set of sets of I. The regions form a partition of the index set I (i.e. $\cup_{i=0}^{r-1} R_i = I$ and $R_i \cap R_k = \emptyset$ for all $i \neq k$) and are given in the form of a *label image* $\Lambda \in \{0, 1, \ldots, r-1\}^{N \times N}$, obtained by filling region R_h with the value h, such that $(i,j) \in R_h \Leftrightarrow \Lambda_{i,j} = h$.
Suppose now we have a function Π that associates to any set of points a Hamiltonian path in the complete graph generated by these points (or equivalently, supposing the points in R are already ordered, a permutation of these points). In other words, for any region R we wish for $\Pi(R)$ to be a bijection from $\{0, 1, \ldots, |R|-1\}$ to R. Later we will present two examples of such functions Π, the region-easy-path and the region-grad-path.
Call $R_k^L := R_k$ for all $k = 0, \ldots, r-1$ and define the region collection at the highest level L of the transform as $\mathcal{R}^L := \{R_0^L, R_1^L, \ldots, R_{r-1}^L\}$. For each region k, call ρ_k^L the path in it given by Π, i.e. $\rho_k^L = \Pi(R_k^L) : \{0, 1, \ldots, |R_k^L|\} \to R_k^L$. By gluing all these paths

---

[1] we use either the max distance, defined by $d((i,j),(k,l)) = \max\{|i-k|, |j-l|\}$, or the usual Euclidean distance

together we obtain

$$p^L := \rho_0^L \cup \rho_1^L \cup \ldots \cup \rho_{r-1}^L,$$

which is a bijection from $\{0, 1, \ldots, |I^L|\}$ to I^L, i.e. a permutation of the whole index set. At this point a global path is defined, and we can proceed just as described in points 1-3 of the general framework described in subsection 3.1: define $\tilde{f}^L := f^L \circ p^L$, compute a^L and d^L through discrete convolution with two wavelet analysis filters as in (3.1.2), define I^{L-1} and f^{L-1}. Additionally now we have to define the new region collection $\mathcal{R}^{L-1} := \{R_0^{L-1}, R_1^{L-1}, \ldots, R_{r-1}^{L-1}\}$, where $R_k^{L-1} = R_k^L \cap I^{L-1}$. An equivalent definition would be:

$$\begin{aligned} R_0^{L-1} &= \rho_0^L(\text{even numbers}) \\ R_k^{L-1} &= \begin{cases} \rho_k^L(\text{even numbers}) & \text{if } |R_{k-1}^{L-1}| \text{ is even} \veebar\ R_{k-1}^{L-1} = \rho_{k-1}^L(\text{odd numbers}) \\ \rho_k^L(\text{odd numbers}) & \text{otherwise}, \end{cases} \end{aligned} \tag{3.3.1}$$

where here $\veebar$ denotes the exclusive `OR`, i.e., it evaluates to true when one and only one of the two conditions is true. This procedure can be repeated analogously for each level. As already mentioned, the final encoding of the image will consist of the segmentation information, the wavelet detail coefficients $d^L, \ldots, d^1$ and the approximation coefficient for the lowest level a^1.

For decoding, since the permutations here aren't stored, we first need to recompute them; to do so we simply need to apply the whole encoding procedure ignoring the pixel values but not the segmentation information, which has been stored and is thus available.

By what has been said it is clear that for our method, the path finding procedure Π must have the following characteristics:

1. It must not depend on the points in the region laying on a regular uniform grid: from level $L-1$ onward in fact the points in each region will usually be unevenly distributed in space.
2. It must be completely deterministic: this is needed in order to obtain the same paths during encoding and decoding.
3. It must depend solely on the geometric disposition of the points in the region, i.e. on the segmentation. It cannot depend on the gray-values of the image since these aren't available during decoding. An exception can be made, as in the region-grad-path, if one is willing to store additional information in the final encoding.

3.3.1 Path finding

Algorithm 3 shows the region-easy-path procedure to find a path π in a region R: starting from some point (which for example can be chosen using the lexicographical ordering), the algorithm tries always to select the closest available neighbour. If there are more points equally close, it selects the one that would make for the straightest path, the

Algorithm 3 Region-easy-path algorithm

```
Input: region $R$
Output: path $\pi$
     choose starting point $p \in R$                    ▷ $p$ is the current point
     $Q = R \setminus p$                                ▷ $Q$ is the set of avaiable points
     $v = (1, 0)$                                       ▷ $v$ is the preferred direction
     while $Q \neq \emptyset$ do
 5:      $\bar{h} = \min\{h \in \mathbb{N},\ B_h(p) \cap Q \neq \emptyset\}$
         $C = \arg\min_{\kappa \in B_{\bar{h}}(p) \cap Q} ||p - \kappa||$
         if $|C| \geq 2$ then
             $C = \arg\max_{\phi \in C} < \phi - p, v >$
             if $|C| \geq 2$ then
10:              rotate $v$ by $\pi/2$
                 goto 8
             end if
         end if
         pick $\psi \in C$                  ▷ there is no choice to be done here: $C = \{\psi\}$
15:      append $\psi$ to $\pi$
         $v = \psi - p$
         $p = \psi$
         remove $\psi$ from $Q$
     end while
20:  return $\pi$
```

rationale being that a more regular path will lead to a smoother signal (see for example Heinen and Plonka (2012) for a proof that, when f is sufficiently smooth, a straighter path gives smaller wavelet coefficients). This is done by computing the scalar product of the increment vector with a preferred direction vector, which at every iteration is updated to be the last increment in the path. If there are 2 possible points with the same minimum angle to the preceeding part of the path, then the preferred direction is rotated by $\pi/2$, making then only one of the two points preferrable.

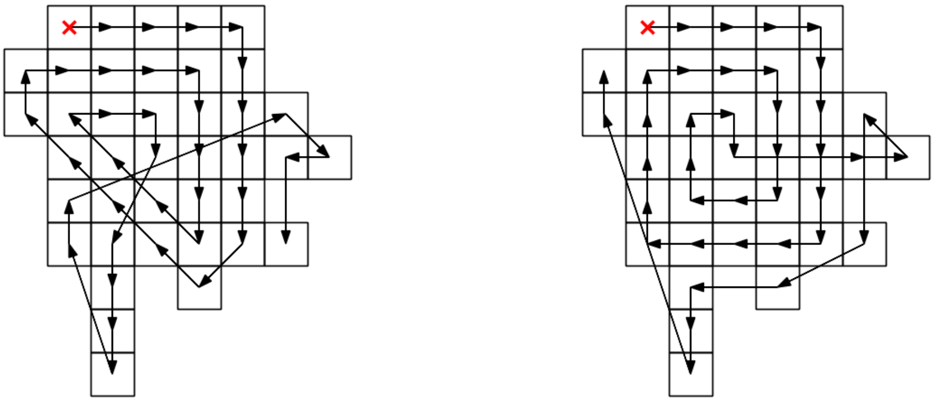


Figure 3.2: Path found by the region-easy-path procedure with the max (left) and Euclidean (right) distance.

See Figure 3.2 for an example of a path determined by region-easy-path. In our tests using the Euclidean distance gave better compression performance, so we used that in all numerical results presented here.

Another path-finding procedure is the region-grad-path, shown in Algorithm 4. It requires previous computation of the average discretized gradient for each region; these vectors have to be stored, contributing to the storage cost of the final encoding. The procedure is very similar to the region-easy-path: the closest point is always preferred. However the preferred direction is always perpendicular to the average gradient, at each iteration the sign being updated so as to obtain the most regular path. Furthermore taking the absolute value when computing the scalar product (see line 8) means that we always prefer a path that remains as much as possible perpendicular to the average gradient, even if it means a sharper change in the path's direction. Only in case of equally distant points forming equal angles the preferred direction is temporarily rotated.

See Figure 3.3 for an example of the path generated by the gradpath algorithm. The points forming the region are the same in the two images, but the gray-values generate two different average gradients.

In our numerical tests (see Section 3.5) the region-grad-path performed only marginally

Algorithm 4 Region-grad-path algorithm

```
Input: region $R$, average gradient $g$
Output: path $\pi$
    choose starting point $p \in R$                          ▷ $p$ is the current point
    $Q = R \setminus p$                                      ▷ $Q$ is the set of avaiable points
    $v =$ rotate $g$ by $\pi/2$                              ▷ $v$ is perpendicular to the average gradient
    while $Q \neq \emptyset$ do
 5:     $\bar{h} = \min\{h \in \mathbb{N},\ B_h(p) \cap Q \neq \emptyset\}$
        $C = \arg\min_{\kappa \in B_{\bar{h}}(p) \cap Q} ||p - \kappa||$
        if $|C| \geq 2$ then
            $C = \arg\max_{\phi \in C} |< \phi - p, v >|$
            if $|C| \geq 2$ then
10:             rotate $v$ by $\pi/2$
                goto 8
            end if
        end if
        pick $\psi \in C$                                    ▷ there is no choice to be done here: $C = \{\psi\}$
15:     append $\psi$ to $\pi$
        $w =$ rotate $g$ by $\pi/2$
        $v = \arg\max_{\gamma = -w, w} < \psi - p, \gamma >$
        $p = \psi$
        remove $\psi$ from $Q$
20: end while
    return $\pi$
```

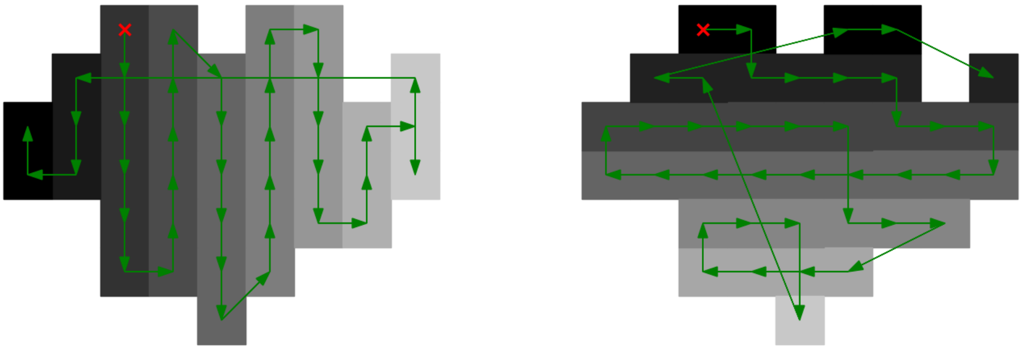


Figure 3.3: Gradpath on same region with different gray-values

better than the region-easy-path - this is without taking into account the additional storage cost.

3.3.2 Remarks on the Segmentation Method

A fundamental goal of the segmentation procedure is to produce regions such that, when its points are reordered according to the region-easy-path procedure, the gray-values considered in this order give a sequence with few jumps. A potentially tricky situation would be the image depicted in Figure 3.4 (a): segmentations (b) and especially (c) are here clearly preferable to (d), because the region-easy-path procedure applied to the latter one would pass many times through the central vertical line of the image and produce a signal with many jumps. In our tests we used the Felzenszwalb-Huttenlocher procedure described in subsection 1.7.4 and the TBES procedure proposed in Rao et al. (2010); the latter is much slower but gives better quality in the reconstructed images since it is specifically designed to segment into regions that optimize the coding length of the boundaries and the therein contained textures. For speed convenience we mostly used the Felzenszwalb-Huttenlocher method, about which we wish to make two remarks: firstly, the regions produced are such that if we use the Minimum Spanning Tree[2] of that region to move between points, we will use only edges of low weight. But when we apply our method, we will be moving along the path generated by the region-easy-path procedure, and thus we will not have any guarantee of not obtaining a signal with jumps. Secondly, since in (1.7.12) the minimum edge weight is considered, the method can behave badly if there is a portion of the image similar to Figure 3.4a of size much smaller than the scale parameter k. This could be avoided by substituting the minimum edge weight in (1.7.12)

---

[2] we're here considering the weighted graph $G = (V, E)$ used for the Felzenszwalb-Huttenlocher segmentation: the vertices are given by the pixels, the edges connect neighboring pixels and the weights are given by the difference in absolute value between the corresponding grayvalues.

with the median edge weight or another quantile, but this is shown in Felzenszwalb and Huttenlocher (2004) to make the segmentation problem NP-hard.

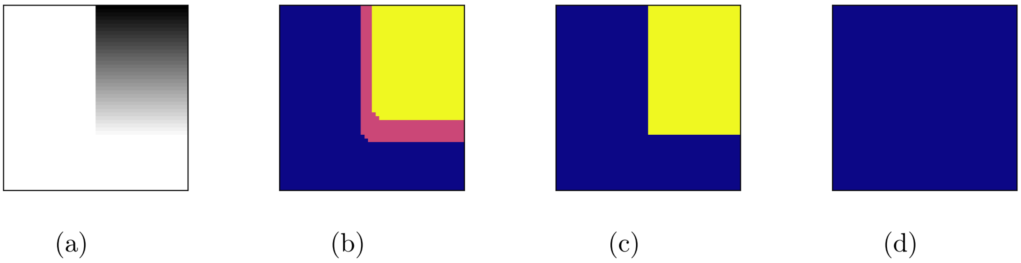

(a) (b) (c) (d)

Figure 3.4: (a) Original image, (b) Felzenszwalb-Huttenlocher segmentation for some scale parameter k_0 and smoothing parameter $\sigma > 0$, (c) $\sigma = 0$, and finally (d) for $\sigma = 0$ and scale parameter $k_1 \gg k_0$.

3.3.3 Encoding Cost and Quality Measurement

When comparing different lossy compression methods, what we are really interested in is the ratio between quality of the reconstruction and encoding storage cost; we thus define

$$Q = K \frac{\text{HaarPSI}}{\text{Encoding Cost}}$$

where K is a normalization constant; see Section 1.6 for details on the HaarPSI measure and why it is preferable to the PSNR. If we call b the number of bits we're using to encode a floating point number, then in the case of the full wavelet tensor transform the encoding cost is $n^2 b$ (since n^2 is the number of pixels in the image). Since we want Q to be equal to 1 in this case, we thus define $K := n^2 b$. If we are encoding a sparse representation of the image and store only $N \ll n^2$ coefficients, then the storage cost can be estimated using Shannon's source coding Theorem (see for example Cover and Thomas (2012)) as $Nb + n^2 H$ where H is the entropy of the *positional* binary sequence, i.e. the binary string of length n^2 identifying which N coefficients are nonzero. Note that since we know that this positional sequence will consist of N ones and $n^2 - N$ zeroes, we can approximate $H = -\frac{N}{n^2} \log_2 \frac{N}{n^2} - \frac{n^2-N}{n^2} \log_2 \frac{n^2-N}{n^2}$. In the wavelet tensor case we thus have

$$Q = n^2 b \frac{\text{HaarPSI}}{Nb + n^2 H} \, .$$

In the case of the RBEPWT we need to take this cost $Nb + n^2 H$ into account as well as the storage cost of the segmentation information. We propose here a simple encoding procedure for the segmentation (or equivalently for the label image Λ) with the purpose of giving a rough theoretical estimate of the storage cost. The idea is to encode the borders between regions as if they were a path: we save the starting pixel and then a sequence of direction changes (so that a straight long line corresponds to a sequence of zeroes). When we arrive at a bifurcation (i.e. a point where 3 or more regions meet) we save the pixel coordinates and continue on one of the possible directions randomly; later when we arrive at the border of the image, we go back to one of the saved pixels and continue on any of the not previously chosen paths. The final storage cost is given by the coordinates of all the bifurcation points plus the set of direction changes.

Formally a segmentation is encoded as a string $p_1 s_{11} s_{12} \dots p_2 s_{21} s_{22} \dots p_n$ where $p_i \in [n-1] \times [n-1] \times [3]$ and $s_{kj} \in [2]$. The idea is to identify a segmentation border element as a pixel in the image and a number in [3] identifying on which side of the pixel the segmentation border is located; from then on the string of s_{jk} is an encoding of the direction change along the pathway of the segmentation borders (where 0 signifies no direction change and 1 and 2 a $\pi/2$ direction change to the left or right respectively). When this pathway reaches a bifurcation point (i.e. a point where segmentation borders from multiple regions come together), a random direction is chosen and the bifurcation coordinates are saved; when it reaches a border of the image or an already found bifurcation point with no new routes avaible, this path part is considered over. One of the previous found bifurcation points is chosen, this point again stored alongside a number in [3] encoding the side on which the segmentation is present, and the procedure starts again from there.

The storage cost of all the borders defining the segmentation is approximated as $\nu\beta + lH'$ where ν is the number of bifurcation points, β is the number of bits needed to store an element of $[n-1] \times [n-1] \times [3]$, H' and l are respectively the entropy and the length of the string $s_{11} s_{12} \dots s_{21} \dots s_{n1} \dots$. Assuming to use 2 bytes for each axis of the image (i.e. $n < 2^{16}$), we have $\beta = 16 + 2 = 18$. In the RBEPWT case we thus have

$$Q = n^2 b \frac{\text{HaarPSI}}{Nb + n^2 H + \nu\beta + lH'} \ .$$

3.4 Thresholding for a Region of Interest

It is worth noting that the RBEPWT method can be applied without modification to images with arbitrary boundary shape. In particular, it can be used to encode only a region of interest (ROI) of a whole image - see Figure 3.5. One could then think of encoding with different qualities the ROI and the rest of the image, simply by viewing them as two separate images with non-regular boundaries. To this end however we can do something a little bit more clever, leveraging the fact that many coefficients of the full encoding of the image, especially at low levels, contain highly non-local information, shared by both the ROI and its complementary. This allows us fine control over the quality of the encoding for a region of arbitrary shape (and its complementary), something that's not possible with the classical 2-dimensional wavelet transform.

For simplicity we will restrict ourselves to the case of Haar wavelets and we will suppose that the region of interest R is one of the regions found in the segmentation step; we

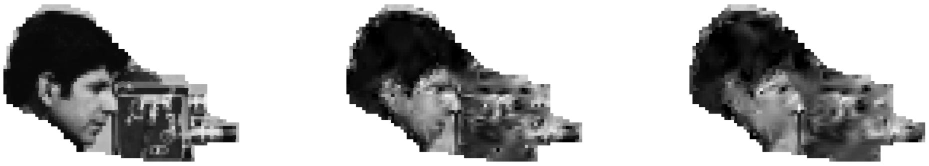
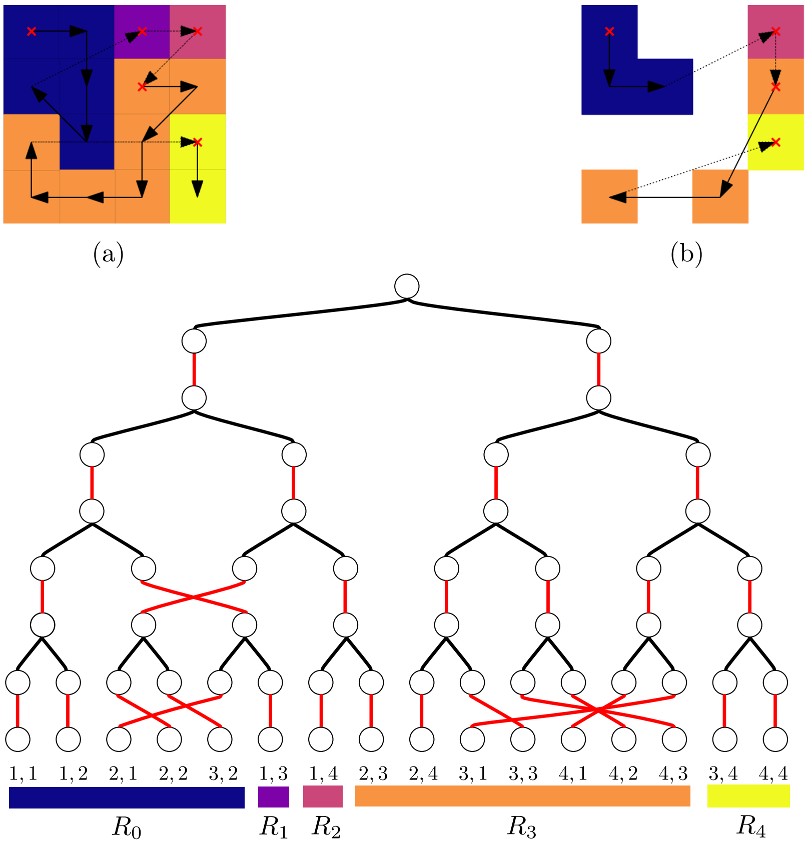

(a) (b) (c)

Figure 3.5: Encoding of a ROI for the cameraman image using the CDF 9/7 wavelet and 11 levels. (a): full quality with 2101 coefficients, (b): thresholded to 210 coefficients and (c): thresholded to 100 coefficients.

will also suppose the image to be of size $2^n \times 2^n$ and to encode using the maximum number of levels $L = 2\log_2 2^n = 2n$. The key observation is that each value of f^{L-1} will be determined only by two values of f^L; this follows from (3.1.2) and the fact that the low and high pass Haar filters have support length 2. This fact is what we used in the construction of the coefficient dependency tree in Chapter 2; in Figure 3.6 an example of such a tree derived from a simple segmentation is shown.

The points in R correspond to a set of leaf nodes in the tree; from the definition of the coefficients on the tree nodes it follows that only the RBEPWT coefficients corresponding to nodes ancestors of these leaf nodes are influenced by the values of the points in R. The farther up the tree a node is, the larger is its zone of influence in the image - the root node is affected by all points in the image, but closer to the leafs the coefficients contain more local information. For example in Figure 3.6 5 coefficients (4 detail and one approximation) are needed to perfectly reconstruct R_4, and of these only one (the root node) is common to R_0. In general if we wish to encode only region R then it suffices to follow the edges of the tree up until the root node, starting from the leafs associated to R and preserving the coefficients encountered in this visit while setting to 0 all the others. In this way all the information necessary to perfectly reconstruct R will be preserved alongside some information of the rest of the image; (see Figure 3.11b). To encode with different qualities R and the rest of the image, we must divide the coefficients into two sets: those that are ancestors to points in R in the tree and those that are not. Then we threshold the coefficients, preserving a certain percentage of those in the first set and another percentage of those in the second but not in the first, always giving precedence to coefficients larger in absolute value (see Figure 3.11c).

The general non-Haar case is more complicated: one point in I^l will influence $\lfloor s/2 \rfloor$ coefficients at level $l-1$, where s is the length of the filter and is usually greater than 2. This means we would have to generalize definition (2.3.1) in order to obtain the relevant graph which would not be a tree anymore, since there would be leafs with more than one parent. Furthermore in the non-orthogonal case the synthesis filters may have different lengths than the analysis, thus making it not obvious from the graph representation which coefficients are needed for perfect reconstruction of the ROI.



(a) (b)

(c)

Figure 3.6: (a): A segmentation of the image in Figure 3.1 (a) with the path found by the region-easy-path procedure at the fourth level. (b): the regions and path at the third level. (c): the coefficient dependency tree with permutation associated to the RBEPWT with Haar wavelets and 4 levels. Here the implied order of the pixels is given by row-stacking (i.e. left to right and top to bottom). The leafs are denoted by their coordinates in the image and are grouped by region. The red edges represent the permutation given by the path finding procedure, while the black edges represent the convolution with the low and high pass filters: see also Chapter 2.

Finally, this idea could be applied to the standard EPWT and tensor wavelet transform as well, however both cases would require more coefficients to encode a ROI compared to the RBEPWT case. For the EPWT the path at each level is searched globally on the whole set of points and not for each region separately, thus the coefficients could potentially get mixed more, with points that are nearby in the image sharing less common coefficients in the graph. For the tensor wavelet transform, the number of coefficients at level $l-1$ influenced by a single coefficient at level l is of the order of $(s/2)^2$ [3], due to the nature of the transform that uses two-dimensional convolution.

3.5 Numerical Experiments

We present here some numerical results, which were all obtained using the python code available at the software page of http://na.math.uni-goettingen.de/ [4]. We used the biorthogonal Cohen-Daubechies-Feauveau wavelets (known also as CDF 9/7) because, analogously to other natural images compression methods, they showed very good performance in our method compared to other wavelets. In Figures 3.8-3.10 our method is compared to the classical two-dimensional tensor wavelet (4 levels) transform and the EPWT (16 levels), both using again the CDF 9/7 wavelets. For the RBEPWT we used mainly the Felzenszwalb-Huttenlocher method which for 256×256 images takes around $40-60$ seconds on the MacBook Pro Mid 2012 with an Intel Ivy Bridge i5 2.5Ghz CPU used for testing. On the `cameraman` image we used also the TBES segmentation, which gives a better segmentation but takes around 15-20 minutes. The encoding and decoding procedures for the RBEPWT method for 256×256 images take around 4 minutes each with most of the time being spent searching for the nearest neighbor in the path-finding procedure: the code could probably be substantially optimized.

It is interesting to note in the Figures how thresholding the coefficients introduces different types of distortion in the different transforms: the 2D wavelet transform blurs the image, while the EPWT and RBEPWT methods preserve edges much better, while introducing a higher frequency noise. In particular, as should be expected, the RBEPWT preserves the segmentation information: for example in the `cameraman` image with the Felzenszwalb-Huttenlocher segmentation (Figure 3.8(e)-(f)), the skyscraper to the right of the camera tripod can be seen only in the RBEPWT images, because it was identified as a separate region in the segmentation step. In the house image the texture of the bricks is completely gone, but the borders are still perfectly visible - thus our method is particularly well-suited for cartoon-like images.

In Tables 3.1-3.4 we compare the qualities of these reconstructions by showing values of the PSNR, HaarPSI and the Q-index (with $b = 64$) defined in subsection 3.3.3. We remark that while the HaarPSI and PSNR values for the two segmentations of the `cameraman` image are very similar, the Felzenszwalb-Huttenlocher segmentation has much more irregular borders and thus gives a lower value of Q: especially for lower number of coefficients the TBES segmentation is clearly better. Please note that for the values of Q for the EPWT we took into account the cost of storing permutations for all the levels and for the RBEPWT with region-grad-path the cost of storing the gradient vector for each region. The RBEPWT clearly outperforms the EPWT when taking the adaptivity costs into account.

In Figure 3.11 we show an example of the ROI based thresholding described in section

---

[3] supposing the low and high pass filter both have length s

[4] also available at `https://github.com/nareto/rbepwt`

3.4. The selected region is an union of several regions identified by the Felzenszwalb-Huttenlocher method.

Finally, we observe that by setting one coefficient to 1 and the rest to 0 and then decoding, it is possible to view the elements of the adaptive basis produced by our method. In Figure 3.12 some of such elements are shown for the `cameraman` image, both for the RBEPWT (using the TBES segmentation) and the EPWT. As should be expected, the basis for our method clearly preserves the segmentation structure, while the basis for the EPWT appears to be totally unstructured.

Table 3.1: `cameraman` with TBES segmentation

encoding	coefficients	PSNR	HaarPSI	Q (with $b = 64$)
easypath	4096	29.34173	0.817805	11.93
gradpath	4096	29.78339	0.830736	12.07
epwt	4096	30.07550	0.838153	2.93
tensor	4096	30.09824	0.799467	11.80
easypath	2048	26.26069	0.718576	20.45
gradpath	2048	26.66623	0.725867	20.49
epwt	2048	26.68888	0.739299	2.92
tensor	2048	26.81330	0.681513	19.82
easypath	1024	24.02702	0.635026	34.89
gradpath	1024	24.37309	0.647067	35.03
epwt	1024	24.29738	0.644697	2.73
tensor	1024	24.07389	0.561290	32.19
easypath	512	22.43470	0.577053	60.10
gradpath	512	22.61402	0.582777	59.00
epwt	512	22.41991	0.548260	2.4
tensor	512	21.64632	0.459870	52.01

Table 3.2: `cameraman` with Felzenszwalb-Huttenlocher segmentation

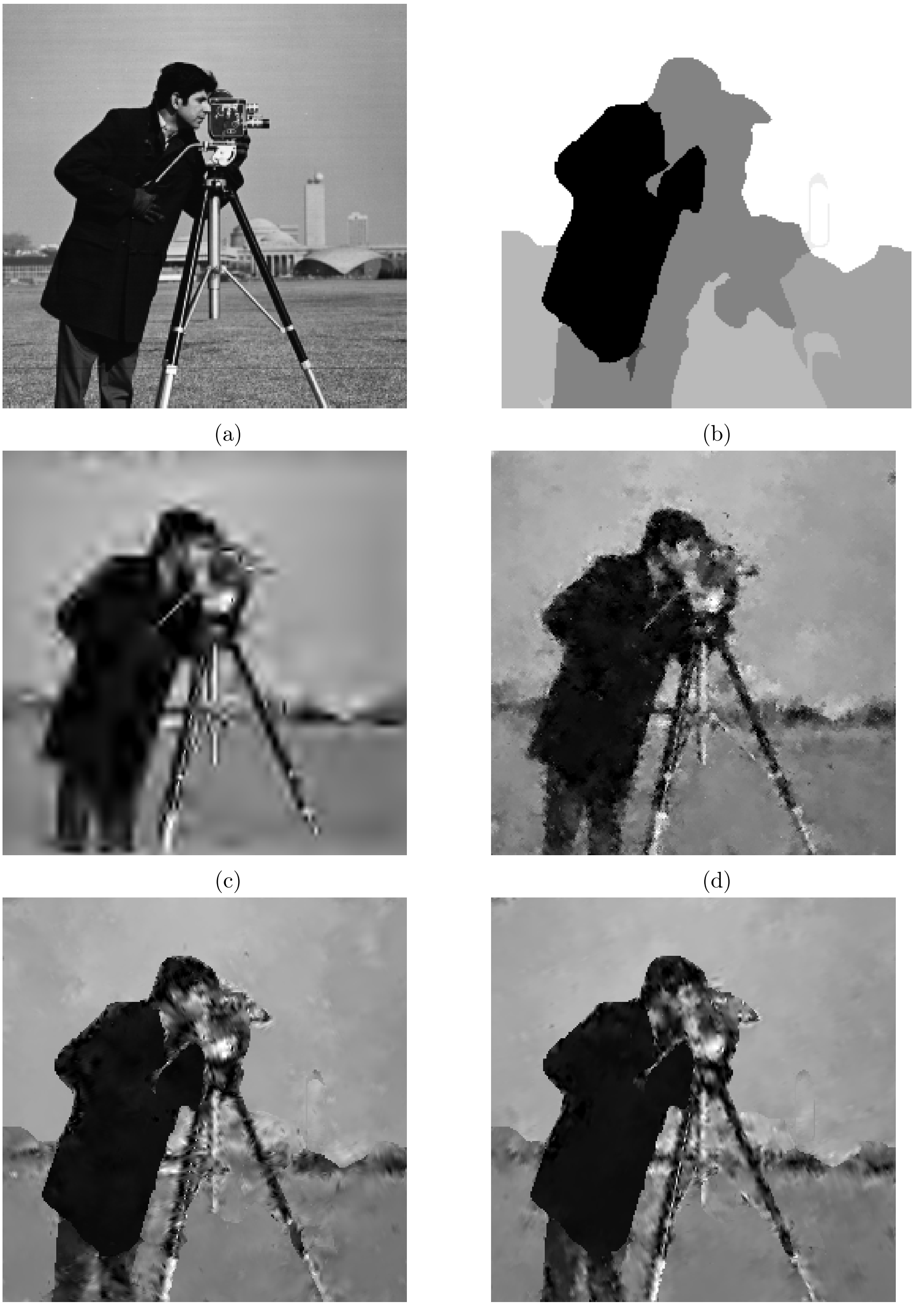
encoding	coefficients	PSNR	HaarPSI	Q (with $b = 64$)
easypath	4096	29.16731	0.818763	11.69
gradpath	4096	29.30971	0.827619	11.62
epwt	4096	30.07550	0.838153	2.93
tensor	4096	30.09824	0.799467	11.80
easypath	2048	25.82304	0.701544	19.15
gradpath	2048	25.99465	0.706431	18.66
epwt	2048	26.68888	0.739299	2.92
tensor	2048	26.81330	0.681513	19.82
easypath	1024	23.49562	0.608677	30.90
gradpath	1024	23.56097	0.600446	28.7
epwt	1024	24.29738	0.644697	2.73
tensor	1024	24.07389	0.561290	32.19
easypath	512	21.77823	0.524272	47.24
gradpath	512	21.76525	0.523178	42.47
epwt	512	22.41991	0.548260	2.4
tensor	512	21.64632	0.459870	52.01

Table 3.3: `peppers` with Felzenszwalb-Huttenlocher segmentation

encoding	coefficients	PSNR	HaarPSI	Q (with $b = 64$)
easypath	4096	29.19485	0.803416	11.32
gradpath	4096	29.12663	0.804430	11.06
epwt	4096	30.13342	0.845705	2.95
tensor	4096	31.45586	0.853261	12.59
easypath	2048	25.63808	0.686624	18.28
gradpath	2048	25.66487	0.690167	17.53
epwt	2048	26.72259	0.749571	2.96
tensor	2048	27.10352	0.726001	21.11
easypath	1024	23.15176	0.580086	28.13
gradpath	1024	23.15506	0.583367	26.02
epwt	1024	24.17585	0.646684	2.74
tensor	1024	23.77188	0.602107	34.53
easypath	512	21.33941	0.502156	41.76
gradpath	512	21.34081	0.495794	35.86
epwt	512	22.05342	0.548996	2.41
tensor	512	21.40979	0.489784	55.39

Table 3.4: `house` with Felzenszwalb-Huttenlocher segmentation

encoding	coefficients	PSNR	HaarPSI	Q (with $b = 64$)
easypath	4096	34.12162	0.899774	12.84
gradpath	4096	34.03853	0.903711	12.69
epwt	4096	33.91521	0.903794	3.15
tensor	4096	35.07878	0.882650	13.02
easypath	2048	30.77357	0.825761	22.51
gradpath	2048	30.86708	0.826890	21.87
epwt	2048	30.67060	0.837226	3.3
tensor	2048	31.55244	0.779743	22.68
easypath	1024	28.05230	0.734101	37.19
gradpath	1024	28.30740	0.750079	35.94
epwt	1024	28.19099	0.761196	3.22
tensor	1024	27.97587	0.611971	35.09
easypath	512	25.96022	0.654926	58.78
gradpath	512	26.16496	0.660720	53.84
epwt	512	26.09769	0.668225	2.94
tensor	512	24.78272	0.457352	51.72

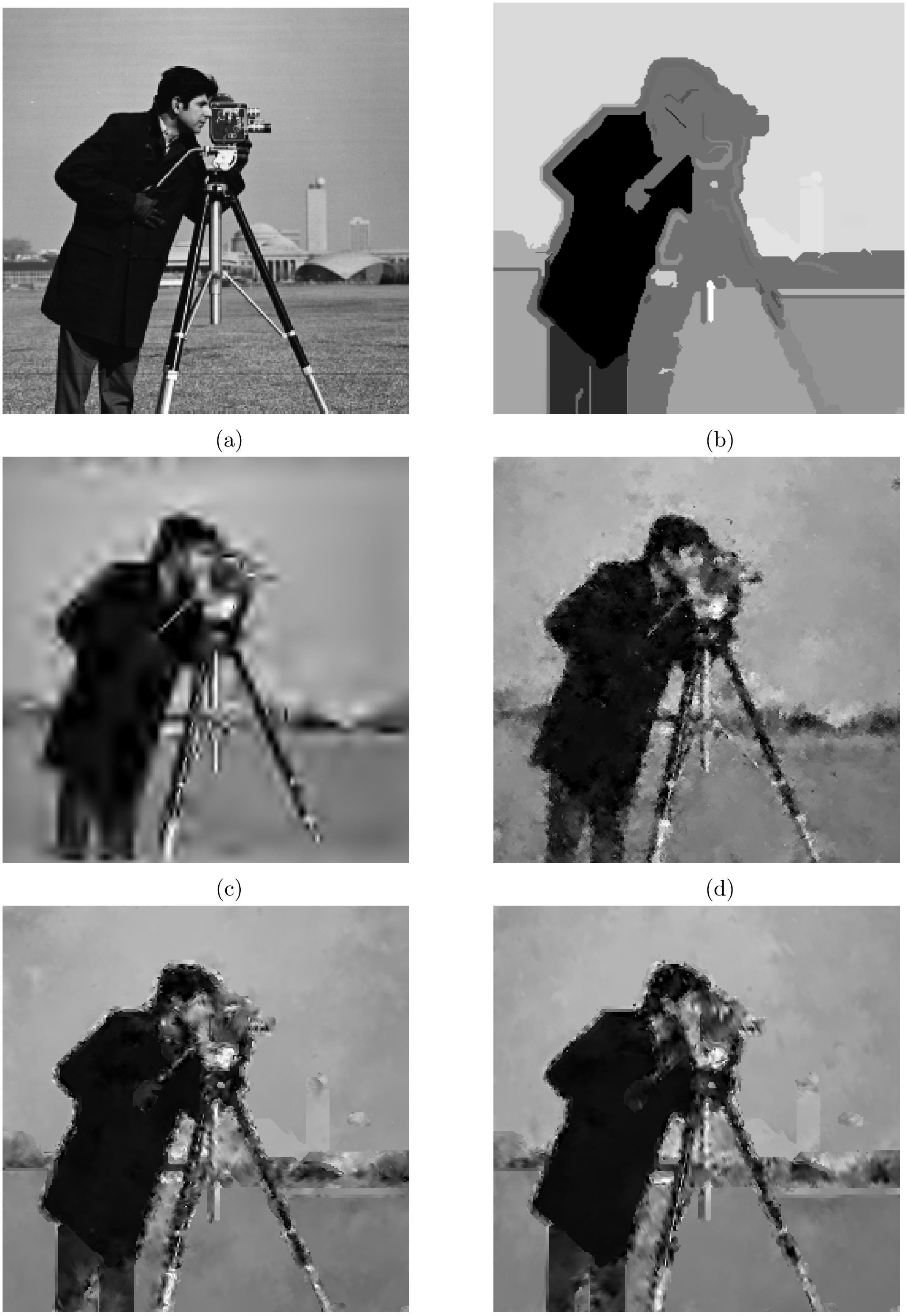


(a) (b)

(c) (d)

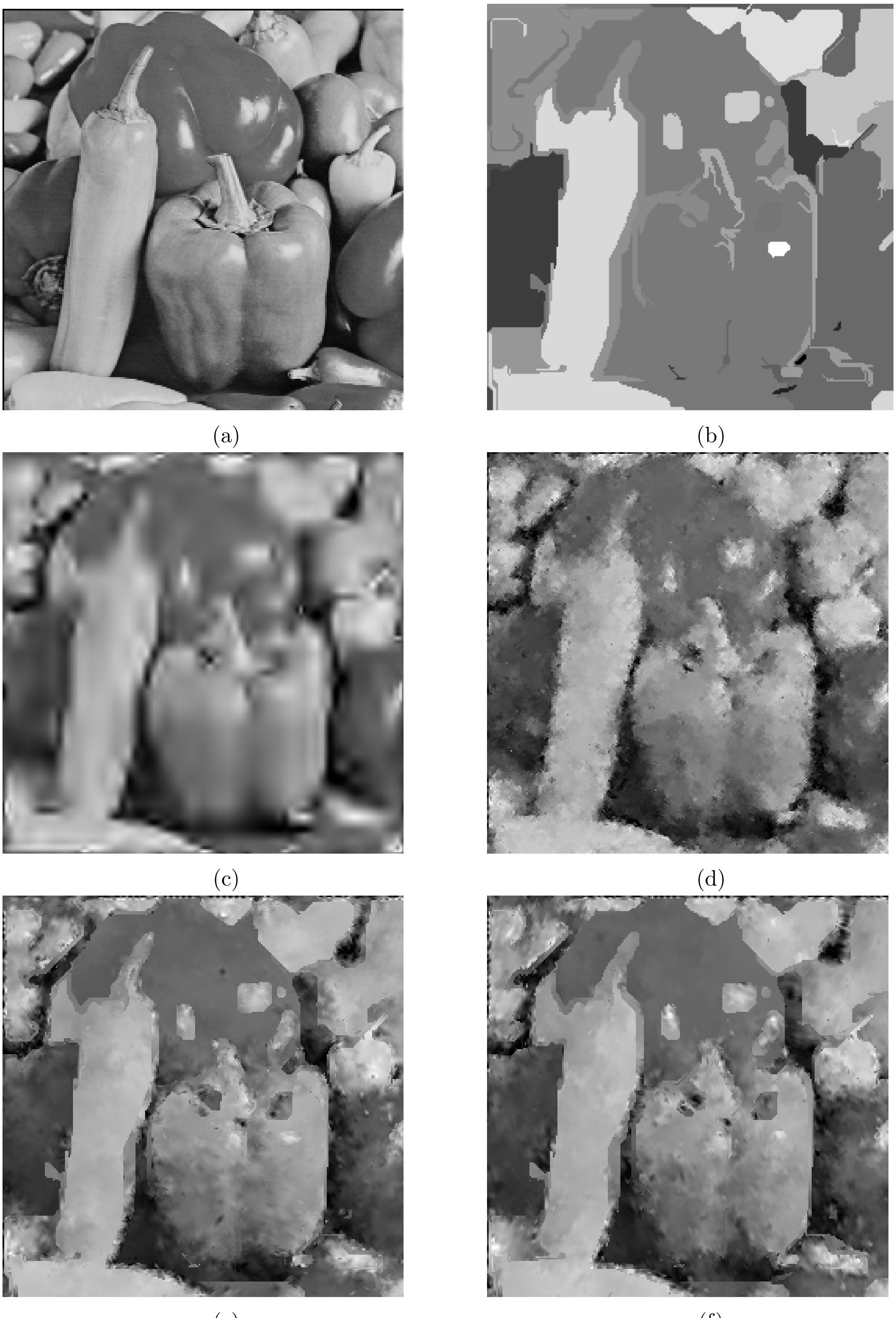
(e) (f)

Figure 3.7: (a): Original 256×256 `cameraman` image. (b): Segmentation obtained from the TBES method with scale parameter $\epsilon = 0.0005$. (c)-(f): Image compressed using 512 coefficients and the CDF 7/9 filter: (c) the classical 2D tensor wavelet transform, (d) EPWT, (e) RBEPWT with region-easy-path and (f) RBEPWT with region-grad-path transforms.



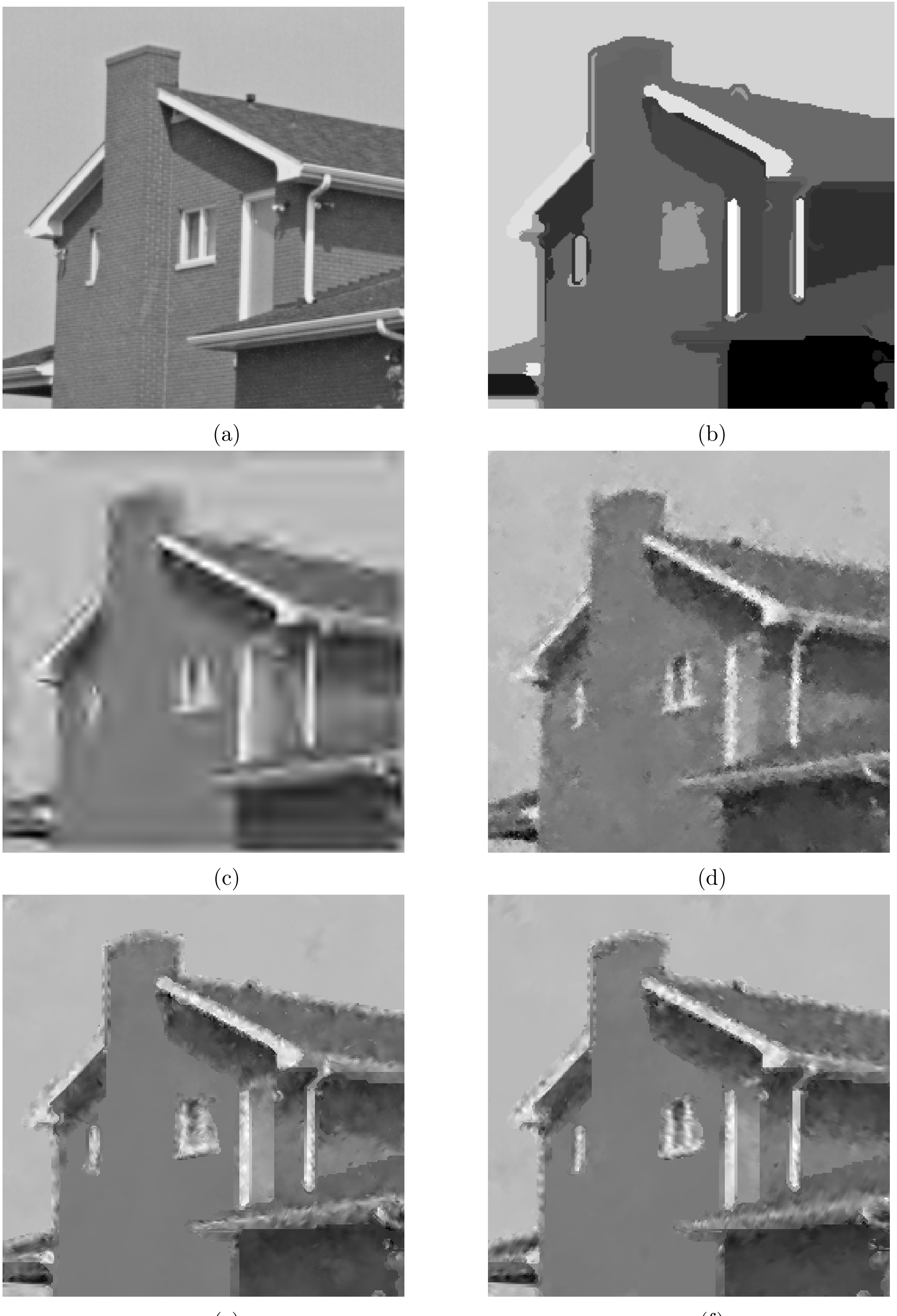
(a) (b) (c) (d) (e) (f)

Figure 3.8: (a): Original 256×256 `cameraman` image. (b): Segmentation obtained from the Felzenszwalb-Huttenlocher algorithm with scale parameter 200, $\sigma = 2$ and minimum cardinality 10. (c)-(f): Image compressed using 512 coefficients and the CDF 7/9 filter: (c) the classical 2D tensor wavelet transform, (d) EPWT, (e) RBEPWT with region-easy-path and (f) RBEPWT with region-grad-path transforms.



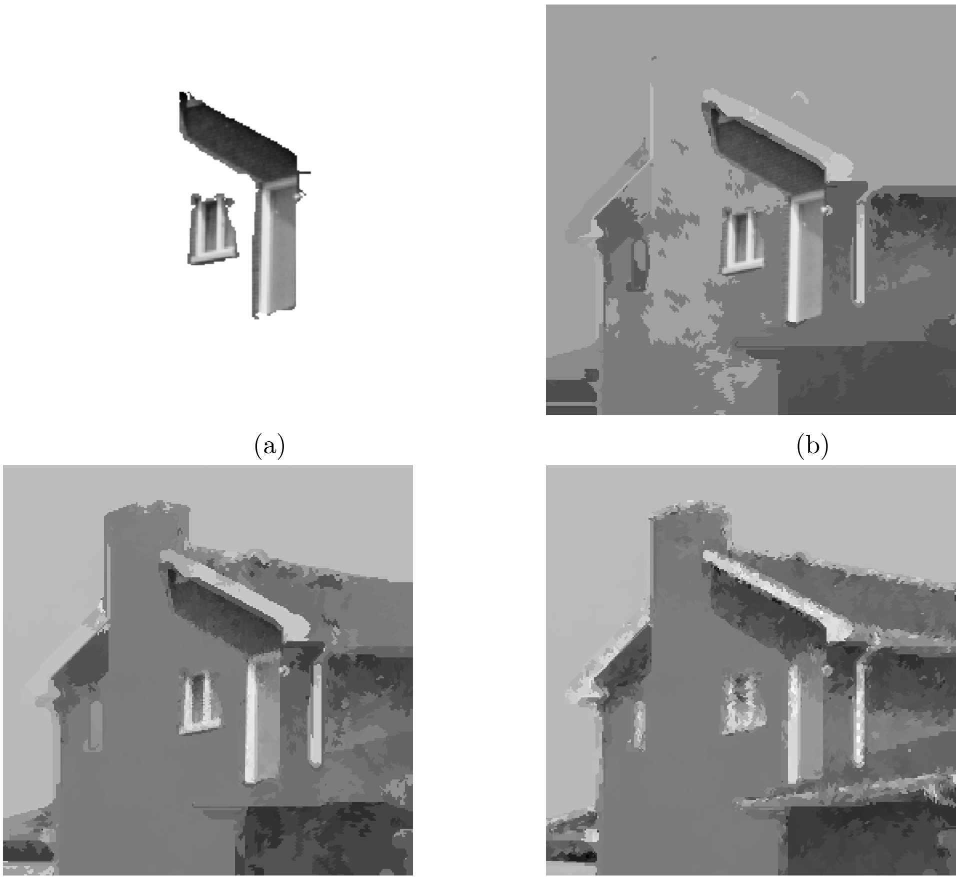
(a) (b) (c) (d) (e) (f)

Figure 3.9: (a): Original 256×256 `peppers` image. (b): Segmentation obtained from the Felzenszwalb-Huttenlocher algorithm with scale parameter 200, $\sigma = 2$ and minimum cardinality 10. (c)-(f): Image compressed using 512 coefficients and the CDF 7/9 filter: (c) the classical 2D tensor wavelet transform, (d) EPWT, (e) RBEPWT with region-easy-path and (f) RBEPWT with region-grad-path transforms.



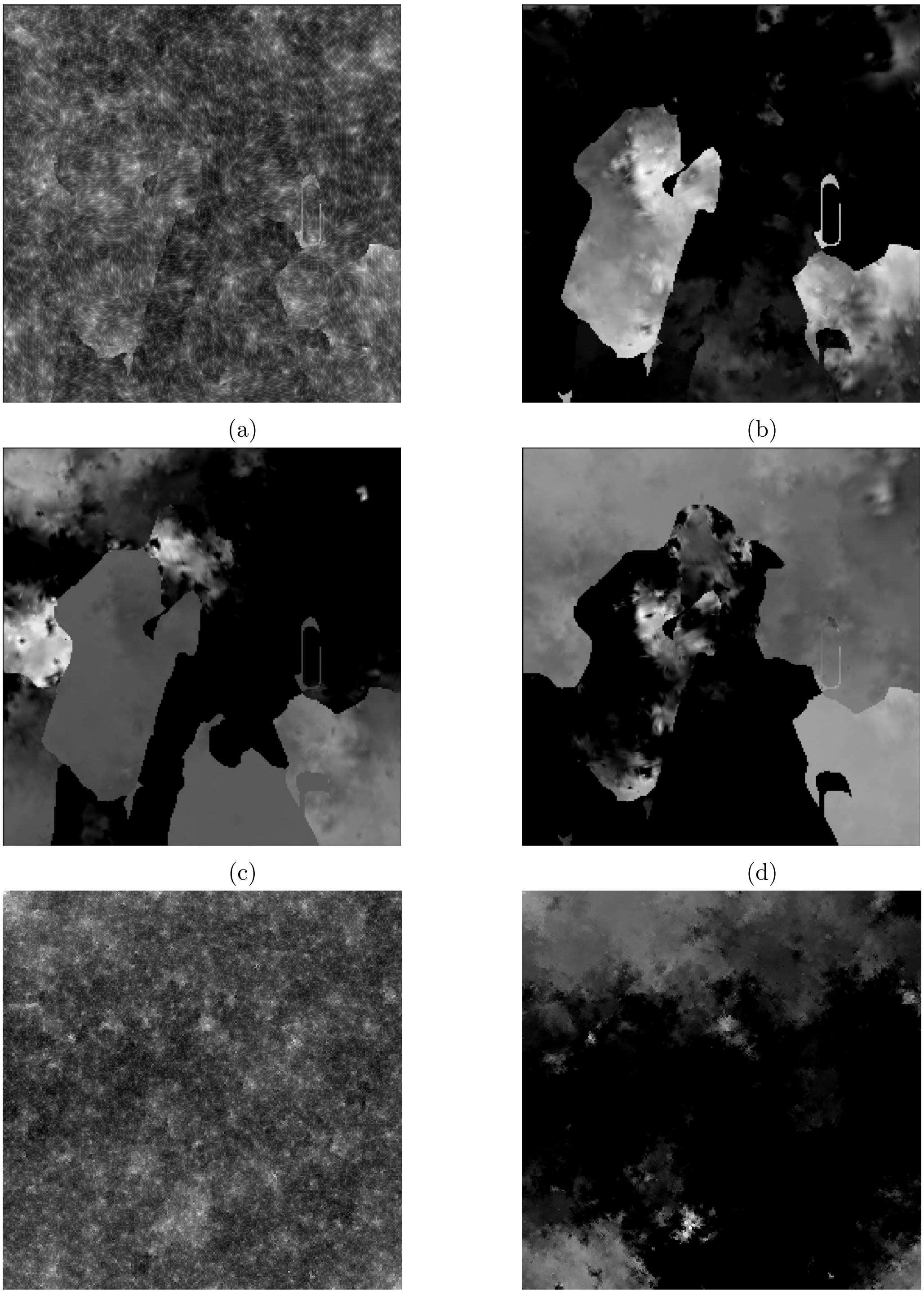
(a) (b) (c) (d) (e) (f)

Figure 3.10: (a): Original 256×256 `house` image. (b): Segmentation obtained from the Felzenszwalb-Huttenlocher algorithm with scale parameter 200, $\sigma = 2$ and minimum cardinality 10. (c)-(f): Image compressed using 512 coefficients and the CDF 7/9 filter: (c) the classical 2D tensor wavelet transform, (d) EPWT, (e) RBEPWT with region-easy-path and (f) RBEPWT with region-grad-path transforms.



(a) (b) (c) (d)

Figure 3.11: (a): The selected ROI for the house image, roughly corresponding to the window, the light part of the wall and the area under the roof; (b): decoded image where all and only the coefficients ancestors to the ROI where preserved (5042 non-zero coefficients); (c): decoded image where 10% of coefficients ancestors to the ROI and 0.1% not ancestors were preserved, for a total of 551 non-zero coefficients; (d): decoded image with standard thresholding with 551 non-zero coefficients (inserted here for comparison with (c)).



(a) (b) (c) (d) (e) (f)

Figure 3.12: Basis elements for the `cameraman` image encoded with 16 levels. (a)-(b): approximation and detail coefficients at the first level of the RBEPWT with TBES segmenation and easypath variant; (c)-(d) the two detail coefficients at the second level of the same RBEPWT; (e)-(f) approximation and detail coefficients at the first level of the EPWT.

4 Sparse Coding and Dictionary Learning

Suppose we are given a signal $y \in \mathbb{R}^n$ and a matrix $D \in \mathbb{R}^{n \times K}$ representing a linear transformation : $\mathbb{R}^K \to \mathbb{R}^n$; finding an $x \in \mathbb{R}^K$ such that $y = Dx$ can be seen from an information-theoretic point of view as a re-encoding procedure of the original signal into the domain of D. In the case $K > n$ which we will be interested in the linear system is under-determined, and thus we will need to impose some further constraints on the solution x in order to shrink the space of feasible solutions. We will suppose that the signal y we are dealing with is a random variable with some unknown but fixed probability distribution which typically is incredibly complex like the distribution of natural images. We are not interested in estimating it, however we want to find a re-encoding of the data that is more efficient than the original representation: a very sensible requirement in this sense would be for the re-encoding x to have a large number of zero components, thus a lower entropy and better compressibility. This is also known to happen in some biological systems where efficiency is paramount, for example in the human V1 visual cortex where visual data is decomposed into a limited number of simple cells corresponding in our setting to the columns of D (this was actually the application that led to the birth of the dictionary learning field, see Olshausen and Field (1996) and Olshausen and Field (1997)).

Whether it is or not possible to find such efficient encodings depends entirely on D and more specifically on its columns which have to quantize $\mathbb{R}^n$ according to the distribution of the random signal y, in such a way that each realization of the random variable is at least approximated closely enough by a linear combination of only a few of these columns. We are making the strong assumption that for the classes of signals we will be treating (natural images) such D does exist, i.e. there is a domain in which the signals we are interested in are sparsely representable. Finding the transformation D is the real question we'll be interested in, but we will start in Section 4.1 by supposing that it is given to us, and consider the following known as the *sparse coding* problem:

$$\min_x ||x||_0 \text{ s.t. } y = Dx \tag{4.0.1}$$

where $D \in \mathbb{R}^{n \times K}$ and $||x||_0 := |\{x_i \neq 0, i = 1, \ldots, K\}|$ is known as the 0-norm. Note that although it has taken this name in the literature due to being the limit for $p \to 0$ of $||\cdot||^p$, this is not really a norm since $||\lambda x||_0 = ||x||_0$ for any $\lambda \neq 0$ and also because it is not convex[1]. These bad properties of the 0-norm function make problem (4.0.1) NP-Hard (see Natarajan (1995)), and thus a variety of methods have been proposed to obtain good approximate solutions.

---

[1] $\left|\left|\frac{1}{2}(1,0) + \frac{1}{2}(0,1)\right|\right|_0 = 2 > 1 = \frac{1}{2}\,||(1,0)||_0 + \frac{1}{2}\,||(0,1)||_0$

In Section 4.2 we will then review the K-SVD method that tackles the more hard *dictionary learning* problem consisting in finding both the optimal transformation and the sparse coding for a set of N signals represented as the columns of a matrix $Y \in \mathbb{R}^{n \times N}$:

$$\min_{D \in \mathbb{R}^{n \times K}, X \in \mathbb{R}^{K \times N}} ||Y - DX|| \ , \ \forall j = 1, \ldots, N \ ||X_{\cdot j}||_0 \leq S \ , \tag{4.0.2}$$

where $S \in \mathbb{N}$ is the required sparsity.

Another interpretation to the problem is to view the columns of D as so-called *dictionary atoms*, an overcomplete (i.e. $K > n$) set of vectors that is efficient in sparsely representing the N signals from our data-set. In Chapter 5 we will focus on the particular case of this data-set being patches extracted from a gray-valued image and develop our own method to solve the problem.

4.1 Sparse Coding

The first question we wish to address is, what can be said about uniqueness of solutions to problem (4.0.1)? Given $y \in \mathbb{R}^n$ and $D \in \mathbb{R}^{n \times K}$ consider the following more general problem

$$\min_{x \in \mathbb{R}^K} J(x) \text{ s.t. } y = Dx \ ,$$

for some real function J; if it is convex there are optimization algorithms guaranteed to converge to a global minimum, and if strictly convex than this solution is unique. It thus makes sense to consider

$$\min_x ||x||_1 \text{ s.t. } y = Dx \ , \tag{4.1.1}$$

i.e. substitute the 0-norm in (4.0.1) with the ℓ^1-norm which is a real norm and thus convex; the problem can also be put in Linear Programming form and can be solved using the simplex method or interior point algorithms. This is at the basis of the so called Basis Pursuit method proposed in Chen et al. (2001), where it was observed that in many instances the solutions to this problem are also sparse in the 0-norm (see also Donoho (2006)). This claim was first proven in Donoho and Huo (2001) for the special case of $D = [I|F]$ where I is the identity matrix and F the Fourier matrix:

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} ,$$

where $\omega = e^{-2\pi i/n}$ is the n-th root of unity. If there is a solution x^* to (4.1.1) for such a matrix D such that $||x^*||_0 < \frac{\sqrt{n}}{2}$, then x^* is also solution to (4.0.1). The authors also generalize this result to the case of any union of two orthonormal bases, by introducing the *mutual coherence* of a matrix $D \in \mathbb{R}^{n \times K}$

$$\mu(D) := \max_{j,k=1,\ldots,K j \neq k} \frac{|D_{\cdot j}^T D_{\cdot k}|}{||D_{\cdot j}||_2 \, ||D_{\cdot k}||_2} \ . \tag{4.1.2}$$

This measure gives an upper bound on the correlation of the columns of the matrix: it is at most 1 (when there are two equal columns in D) and is 0 for orthogonal matrices. They prove the following:

Theorem 4.1.3. *Let $y \in \mathbb{R}^n$, $D = [\Psi, \Phi]$ with $\Psi, \Phi \in \mathbb{R}^{n \times n}$ orthogonal matrices and $x \in \mathbb{R}^{2n}$ such that $Dx = y$ and*

$$||x||_0 < \frac{1}{2}(1 + \mu(D)^{-1}) \ ;$$

then x is the unique solution to (4.1.1) *and also* (4.0.1).

This means that we can solve the easier problem (4.1.1) and, if we obtain a sparse enough solution, have the certainty that this is also solution to the difficult problem (4.0.1). The "sparse enough" bound gets more relaxed with a smaller value of $\mu(D)$: we can then say that in some sense the more uncorrelated the columns of D are the easier the sparse coding problem is.

In Elad and Bruckstein (2002) the following theorem is proved:

Uncertainty Principle 4.1.4. *Let $\Psi, \Phi \in \mathbb{R}^{n \times n}$ be orthogonal matrices and $x_1, x_2 \in \mathbb{R}^n$ such that $y = \Psi x_1$ and $y = \Phi x_2$; then we have*

$$||x_1||_0 + ||x_2||_0 \geq \frac{2}{\mu([\,\Psi \mid \Phi\,])} \ .$$

Thus if the mutual coherence of the two orthogonal bases is small, a signal y cannot be represented sparsely in both bases; in the case of Ψ being the identity matrix and Φ the Fourier matrix, this can be compared with the classic Heisenberg uncertainty principle which states that a signal cannot be tightly concentrated both in time and frequency. From the Uncertainty Principle it is easy to deduce the following:

Theorem 4.1.5. *Let $y \in \mathbb{R}^n$, $D = [\Psi, \Phi]$ with $\Psi, \Phi \in \mathbb{R}^{n \times n}$ orthogonal matrices and $x_1, x_2 \in \mathbb{R}^{2n}$ such that $Dx_1 = Dx_2 = y$; then we have*

$$||x_1||_0 + ||x_2||_0 \geq \frac{2}{\mu(D)} \ .$$

Corollary 4.1.6. *Let $y \in \mathbb{R}^n$, $D = [\Psi, \Phi]$ with $\Psi, \Phi \in \mathbb{R}^{n \times n}$ orthogonal matrices and $x \in \mathbb{R}^{2n}$ such that $Dx = y$ and*

$$||x||_0 < \mu(D)^{-1} \ ;$$

then x is the unique solution to (4.0.1).

This uniqueness result can be further generalized to any matrix $D \in \mathbb{R}^{n \times K}$: see Elad (2010) for a proof of the following Theorem that generalizes Theorem 4.1.3.

Theorem 4.1.7. *Let $y \in \mathbb{R}^n$, $D \in \mathbb{R}^{n \times K}$ and $x \in \mathbb{R}^K$ such that $Dx = y$ and*

$$||x||_0 < \frac{1}{2}(1 + \mu(D)^{-1}) \ ;$$

then x is the unique solution to (4.0.1).

We now know that if we find a solution to $Dx = y$ that is sparse enough, this will also be solution to (4.0.1). In practice though we want a reliable method that finds always a reasonably sparse solution even though it might not be optimal. There have been a multitude of methods proposed to find approximated solutions to problem (4.0.1), starting from methods specific to the matrix D (for example for the case of $D = [I|F]$ with F the Fourier matrix see Dragotti and Lu (2014)), greedy methods like Matching Pursuit (MP) (Mallat and Zhang (1993)) and its improvement Orthogonal Matching Pursuit (OMP) (Pati et al. (1993)) to more recently the Iterative Thresholding method (Blumensath and Davies (2008b)). In the next subsection we will give an overview of the MP and OMP methods, which (especially the latter) are still widely used in practice due to their speed and reasonable accuracy. All these methods work to find an approximation of (4.0.1) or the following:

$$\min_{x \in \mathbb{R}^K} ||y - Dx|| \ \text{ s.t. } \ ||x||_0 \leq S \tag{$P_{0,\epsilon}$}$$

where $S \in \mathbb{N}$ is a fixed required sparsity. This is very similar to (4.0.1) but more relevant for applications, where usually one is interested only in approximated reconstruction of the original signal, the quality of which can be indirectly controlled by S.

In some applications like for example denoising, an additional regularizing term for x is added to the optimization function in ($P_{0,\epsilon}$) (see for example Section 5.1 for two examples of such instances); in these cases an ad hoc optimization procedure must be developed to minimize simultaneously the fidelity term $||y - Dx||$ and the regularizing term.

4.1.1 Matching Pursuit Algorithms

We wish to summarize here the Matching Pursuit (MP) and Orthogonal Matching Pursuit (OMP) methods; for a more detailed discussion see Elad (2010) or the original papers Mallat and Zhang (1993) and Pati et al. (1993). For ease of computation, we'll suppose in this section that the columns of D are normalized with respect to the Euclidean vector norm - it is possible to relax this assumption and we are not loosing in generality.

Matching Pursuit is a greedy method that repeatedly projects the residual (initialized as the signal itself) onto the dictionary atom with which it has the greatest scalar product, using this last quantity to update the corresponding entry in the new encoding x. The main idea of the method stems from the following question: which is the best approximation of the signal y obtained by projecting it on only one dictionary atom? Write

$y = (d^T y)d + r_1$ where $r_1 := y - (d^T y)d$ for some column d of D and observe that this residual must be orthogonal to d

$$d^T r_1 = d^T y - (d^T y)d^T d = d^T y - d^T y = 0 \ ,$$

since we are supposing the columns of D to be normalized. Thus we have

$$||y||^2 = |d^T y|^2 + ||r_1||^2 \tag{4.1.8}$$

and minimizing the residual error $||r_1||^2$ is equivalent to finding the column d with largest scalar product with y. This procedure can be iterated by defining $r_0 := y$ and $r_{n+1} := r_n - (d_n^T r_n)d_n$, where d_n is the column of D with largest scalar product with r_n; see Algorithm (5).

Algorithm 5 Matching Pursuit

```
Input: $D \in \mathbb{R}^{n \times K}, y \in \mathbb{R}^n$ and either $\epsilon \in \mathbb{R}$ or $S \in \mathbb{N}$
Output: Sparse encoding $x$
1: Initialize $x_0 = 0, r_0 = y, i = 0$
2: while STOP CRITERIA is FALSE do
3:     Increment $i$ by 1
4:     Find the column of $D$ with largest (in absolute value) scalar product with the previous residual:
           $k_i := \arg\max_{k=1,\dots,K} \left| D_{\cdot k}^T r_{i-1} \right| \ , \ d_i := D_{\cdot k_i}$
5:     Update the corresponding component in the solution:
           $x_i := x_{i-1}$ and $x_i(k_i) := x_{i-1}(k_i) + d_i^T r_{i-1}$
6:     Update the residual:
           $r_i := y - Dx_i$
              $= r_{i-1} - d_i^T r_{i-1} d_i$
7: end while
```

The equality at line (6) is obtained by substituting y with $r_{i-1} + Dx_{i-1}$ and using the definition of x_i. Depending on whether we are trying to solve (4.0.1) or $(P_{0,\epsilon})$, the stop criteria at line (2) will be different: in the first case we would check if the norm of the residual error is below ϵ and stop when this happens, in the second case we would simply perform a fixed number of S iterations.

At iteration I the approximation of y we have available is given by the sum of the chosen atoms up to that point with the corresponding coefficients, i.e.

$$\begin{aligned} y &= y_I + r_{I+1} \\ &= \sum_{i=0}^{I} (d_i^T r_i) d_i + r_{I+1} \ . \end{aligned}$$

By iterating the reasoning used to obtain (4.1.8) we have

$$||y||_2^2 = \sum_{i=0}^{I} |d_i^T r_i|^2 + ||r_{I+1}||_2^2 \ ;$$

in Mallat and Zhang (1993) it is shown that if D is complete (i.e. the columns of D generate the whole space) then $||r_i||_2 \underset{i\to\infty}{\longrightarrow} 0$. The main computational cost of the Matching Pursuit method comes from computing the scalar products at line (4); supposing then we use the variant for problem $(P_{0,\epsilon})$ that runs for S iterations, the computational cost is $O(SKn)$.

It must be noted that, since the columns of D are usually not orthogonal, y_I is not necessarily the best approximation of y obtained with atoms $d_1, \dots, d_I$ which would be the projection of y onto the space generated by these atoms. This is due to the fact that at each step we simply project the residual on the last chosen column, not on the space generated by all the previous ones. This is exactly the improvement that Orthogonal Matching Pursuit proposes; see Algorithm (6).

The same considerations as for the Matching Pursuit algorithm are valid for the stop criteria at line (2). Differently than in Matching Pursuit, the computation of the scalar products in line (4) can be done only for the columns that have not been chosen in previous iterations, since the residual is orthogonal to all those and will give a null scalar product. In line (5) we wish to find the best possible approximation of y using only atoms in Σ_i; thus we have to solve the linear least squares problem $\min ||y - D_{\Sigma_i} x_{\Sigma_i}||$ where with the subscript Σ_i we mean the restriction of the matrices columns (respectively vector entries) to indices in Σ_i. This step is computationally expensive and would require the computation of the Moore-Penrose pseudoinverse of D_{Σ_i}: this is classically done through the SVD of D_{Σ_i} with a cost of $O(ni^2)$. The total computation cost is then

$$O(n \sum_{i=1}^{S} i^2 + SKn) = O(nS^3 + SKn) \ .$$

Several approaches have been proposed to approximate the expensive step of computing the Moore-Penrose pseudoinverse and thus diminish the computational cost of the method, see Blumensath and Davies (2008a) and the references therein.

Algorithm 6 Orthogonal Matching Pursuit

```
Input: $D \in \mathbb{R}^{n \times K}, y \in \mathbb{R}^n$ and either $\epsilon \in \mathbb{R}$ or $S \in \mathbb{N}$
Output: Sparse encoding $x$
Initialize $x_0 = 0, r_0 = y, i = 0, \Sigma_0 = \emptyset$
while STOP CRITERIA is FALSE do
    Increment $i$ by 1
    Among the not yet chosen columns of $D$, choose the one with largest (in absolute value) scalar product with the previous residual:
        $k_i := \arg\max_{k \notin \Sigma_{k-1}} \left| D_{\cdot k}^T r_{i-1} \right| \ , \ d_i := D_{\cdot k_i}$
        $\Sigma_i := \Sigma_{i-1} \cup \{k_i\}$
    Update the solution by projecting $y$ on $V_i := < d_1, \dots, d_i >$:
        $x_i := \mathcal{P}_{V_i} y = \arg\min_{x \ s.t. \ \mathrm{supp}\{x\} = \Sigma_i} ||y - Dx||_2^2$
    Update the residual:
        $r_i := y - Dx_i$
end while
```

4.2 Dictionary Learning

We now turn to the dictionary learning problem (4.0.2), which we state here once again:

$$\min_{D\in\mathbb{R}^{n\times K}, X\in\mathbb{R}^{K\times N}} ||Y - DX|| \ , \ \forall j = 1, \dots, N \ ||X_{\cdot j}||_0 \leq S \ .$$

This problem is NP-Hard (see Tillmann (2015)) and thus one can only hope to find a good approximated solution. An interesting result however is that, if we suppose the dictionary atoms to be normalized, the solution D, X is unique (up to a change in the order and the sign of the columns) if S is small enough, N is big enough and some notion of non-degeneracy on the training signals in Y is satisfied; see Aharon et al. (2006b).

There are two main strategies to find a suitable dictionary D, one analytical and one that learns the atoms from the statistical characteristics of a set of training signals. In the first case, the simplest method is choosing the dictionary as an union of different bases, relying on the assumption that the signals of interest can be efficiently represented as a superposition of phenomena each of which has a natural representation in one of the bases. The most classic example of this (see for example Donoho and Huo (2001) and Dragotti and Lu (2014)) is the case where the first n columns of D are the standard orthonormal basis and the following n are the sinusoidal Fourier basis. Most such methods used in practice involve the theory of frames in Hilbert spaces, which are an overcomplete set of functions (usually generated through transformations of a kernel) providing a redundant and stable representation; these methods generally don't make explicit use of the data in Y. We will instead be interested in the second approach which relies exclusively on this data to construct the atoms in D. The first to study such methods and actually the dictionary learning problem in general (though not exactly in the form above) were Olshausen and Field in various papers in the late '90s, see for example Olshausen and Field (1996) and Olshausen and Field (1997). They were interested in finding the best basis functions to represent small patches of images, drawing an analogous with the pattern receptors in the human visual system. They used a maximum-likelihood statistical approach and the atoms they obtained had a clear two-dimensional structure resembling rotations of a two-dimensional DCT basis; for example there were many atoms that would detect an edge in the image with some particular orientation.

We can view problem (4.0.2) as a nested inner-outer minimization problem; a basic strategy would thus be to iterate through a two-step procedure where we try to approximate both minima independently. This is the general scheme illustrated in Algorithm (7), where the dictionary and encoding matrices are updated one at a time in each step. One of the first methods to use this strategy was the Method of Optimal Directions (MOD) (see Engan et al. (1999)), which considers the problem in the Frobenius norm; at iteration i Matching Pursuit is used for step 1 to compute the encoding matrix X^i and its Moore-Penrose pseudo-inverse $(X^i)^+ := (X^i)^T(X^i(X^i)^T)^{-1}$ which is used to compute the solution to the least-squares problem

$$D^i := \arg\min_{D\in\mathbb{R}^{n\times K}} \left|\left|Y - DX^i\right|\right|_F^2$$

$$= Y(X^i)^+ \ .$$

Usually the initial dictionary atoms in Algorithm (7) (columns of D^0) are chosen as random columns of Y. The stop criteria can trigger after a fixed number of iterations or when the change in the residual error $\left|\left|Y - D^i X^i\right|\right|$ is below a fixed threshold. In the next subsection we will review the K-SVD method which also conforms to this scheme; incidentally, the method we will develop in Chapter 5 does not follow this alternated optimization scheme, but rather concentrates only on learning a dictionary matrix D and when this is found delegates the computation of X to a sparse coding algorithm like OMP.

If both the steps in Algorithm (7) reduce the value of $||Y - DX||$ then the procedure must converge to a local minimum of problem (4.0.2). The pursuit algorithms like MP or OMP which are typically used for the first step offer only an approximated solution to the sparse coding problem, and thus it is not certain they will decrease the global error; however this is indeed the case in most practical applications.

Algorithm 7 General scheme for dictionary learning

```
Input: Data matrix $Y$, sparsity $S$
Output: Dictionary $D$, encoding $X$
Initialize $i = 1$, $D^0 \in \mathbb{R}^{n \times K}$ with $\ell^2$ normalized columns
while STOP CRITERIA is FALSE do
    1. Find an approximate solution $X^i$ to
       $\min_{X \in \mathbb{R}^{K \times N}} \left|\left|Y - D^{i-1}X\right|\right| \quad \text{s.t. } \forall j = 1, \dots, N \ ||X_{\cdot j}||_0 \leq S$
    2. Find an approximate solution $D^i$ to
       $\arg\min_{D \in \mathbb{R}^{n \times K}} \left|\left|Y - DX^i\right|\right|$
end while
```

4.2.1 The K-SVD method

We give here a brief review of the K-SVD method proposed in Aharon et al. (2006a) which represents the current state of the art in solving problem (4.0.2). It has been used for many applications, including denoising (see Elad and Aharon (2006)) and compression of facial images (see Bryt and Elad (2008)). The method follows the general scheme of Algorithm (7) and is most naturally formulated in the case of the Frobenius norm.

In the first sparse coding stage of each iteration i the dictionary D^i is considered fixed: we are thus left with an instance of the sparse coding problem discussed in subsection 4.1 for which the OMP method is used. Once the new coding matrix X^i has been obtained, the K-SVD method updates one by one the atoms of the dictionary. To do this, it considers iteratively a column $D^i_{\cdot k}$ of the dictionary D^i and supposes that all the other columns are fixed. It is useful to write the product D^iX^i as sum of rank 1 matrices, which allows us to isolate the error due to column $D^i_{\cdot k}$:

$$\begin{aligned}\left|\left|Y - D^iX^i\right|\right|^2_F &= \left|\left|Y - \sum_{j=1}^{K} D^i_{\cdot j}X^i_{j\cdot}\right|\right|^2_F \\ &= \left|\left|(Y - \sum_{j\neq k}^{K} D^i_{\cdot j}X^i_{j\cdot}) - D^i_{\cdot k}X^i_{k\cdot}\right|\right|^2_F \\ &= \left|\left|E^i_k - D^i_{\cdot k}X^i_{k\cdot}\right|\right|^2_F \ , \end{aligned} \tag{4.2.1}$$

where we have defined the overall representation error matrix $E^i_k = Y - \sum_{j\neq k}^{K} D^i_{\cdot j}X^i_{j\cdot}$. Theorem 1.3.7 tells us that we could use SVD to compute $\tilde{X}^i_k$ and $\tilde{D}^i_{\cdot k}$ and by substituting them to $D^i_{\cdot k}$ and $X^i_{k\cdot}$ we could achieve the best rank 1 approximation of matrix E_k, thus minimizing the error due to atom k. However, while the pursuit algorithm in the first step assures us that the columns of matrix X will respect the S-sparsity constraint, the SVD knows nothing about sparsity, and thus the new vector $\tilde{X}^i_k$ would likely be filled. There is however a simple workaround to this problem: define ω^i_k as the indices corresponding to data points $Y_{\cdot i}$ that are encoded in X^i using atom $D^i_{\cdot k}$, i.e.

$$\omega^i_k = \{h | 1 \leq h \leq N, X^i_{k\,h} \neq 0\} \ , \tag{4.2.2}$$

then restrict matrices Y and E^i_k to only those columns whose indices are in ω_k. Formally this can be done by defining an $N \times |\omega_k|$ matrix Ω^i_k whose columns are the ordered standard basis elements e_h for all $h \in \omega^i_k$; the reduced matrices can then be defined as $Y^i_R := Y\Omega^i_k$ and $E^i_{k\,R} := E^i_k\Omega^i_k$. The SVD decomposition can then be applied to $E^i_{k\,R}$ yielding matrices U_i, Δ_i and V_i such that $E^i_{k\,R} = U_i\Delta_iV_i^T$. The new dictionary atom $\tilde{d}_k$ and corresponding coding coefficients $\tilde{X}^i_k$ are then taken to be the first column of U_i and the first column of V_i multiplied by $(\Delta_i)_{1\,1}$ respectively. Note that in the dictionary update step we are getting for free with the SVD also an update of the row of the coding matrix corresponding to the dictionary atom being updated. This means that the coding matrix X is updated column by column in the sparse coding step and row by row in the dictionary update step, though each update in the sparse coding step disregards the previous information contained in X. See algorithm 8 for the pseudo-code. The dictionary update step of the Algorithm will decrease the value of the residual $||Y - DX||_F$, thus if OMP finds a better encoding at each sparse coding step (which is usually the case) the method will converge to a local minimum. The stop criteria in the main loop can be set to either trigger after a fixed number of iterations I or when the residual error goes below a certain threshold ϵ. In the first case we can compute the computational cost as follows: for each of the I iterations we must compute N sparse codings and K SVDs of a matrix with n rows and fewer than N columns. Thus the total

Algorithm 8 K-SVD algorithm

```
Input: Data matrix $Y$, sparsity $S$ and either number of iterations $I$ or error threshold $\epsilon$
Output: Dictionary $D$, encoding $X$
Initialize $i = 1$, $D^0 \in \mathbb{R}^{n \times K}$ with $\ell^2$ normalized columns
while STOP CRITERIA is FALSE do
    Use OMP to find a sparse solution $X^i$ to the coding problem:
        $$\min_{X \in \mathbb{R}^{K \times N}} \left|\left|Y - D^{i-1}X\right|\right|_F^2 \quad \text{s.t. } \forall j = 1, \dots, N \, ||X_{\cdot j}||_0 \leq S$$
    Define $D^i := D^{i-1}$
    for $k = 1, \dots, K$ do                                  ▷ For each $k$ update column $D^i_{\cdot k}$
        Compute the overall representation error matrix $E^i_k = Y - \sum_{j \neq k} D^i_{\cdot j} X^i_{j \cdot}$
        Compute $\omega_k := \{h | 1 \leq h \leq N, X^i_{k\,h} \neq 0\} = \{\rho_1, \dots, \rho_H\}$
        Define $E^i_{k\,R} := [(E^i_k)_{\cdot \rho_1} | \dots | (E^i_k)_{\cdot \rho_h}]$   ▷ Restrict $E^i_k$ to columns' indices in $\omega_k$
        Compute the SVD decomposition $E^i_{k\,R} = U_i \Delta_i V_i^T$
        Define $\tilde{d}_k := (U_i)_{\cdot 1}$ and $\tilde{X}^i_k = \Delta_{11}(V_i)_{\cdot 1}$
        Substitute the $k$-th column of $D^i$ with $\tilde{d}_k$ and the $k$-th row of $X^i$ with $\tilde{X}^i_k$
    end for
    Set $i := i + 1$
end while
Set $D = D^{J-1}$
```

cost will be smaller than

$$I(NO(nS^3 + Kn) + KO(nN^2)) = O(NnIS^3 + NIKn + nIKN^2) \ . \tag{4.2.3}$$

In Rubinstein et al. (2008) a more clever implementation of the K-SVD method that uses the so-called Batch-OMP is described; therein also an approximated version of the K-SVD is proposed, which is much faster while still maintaining good accuracy and which we will thus use in our numerical experiments.

4.2.2 Storage cost of a dictionary

Suppose we are given N sample vectors in $\mathbb{R}^n$; the cost of storing them is Nnb, where b is the number of bits we wish to use to encode a floating point number. Suppose now we are given a dictionary D learned from data with K atoms of dimension n: by solving the sparse coding problem (4.0.1) we can instead store the dictionary D and the sparse encoding matrix X, from which we will be able to recover an approximation of Y by a simple matrix multiplication. The cost of storing D is exactly Knb, while the cost of storing X consists of SNb (S floating point numbers for each column) plus the cost of the position vectors identifying where in the column the S non-zero floating point numbers are located. We will estimate this latter using Shannon's source coding theorem (see for example Cover and Thomas (2012)) by estimating the entropy of the binary sequence of length K denoting the locations of the non-zero entries in a generic column of X. Since this string will consist of S ones and $K-S$ zeroes, the probability of an entry being a one is S/K and of it being a zero is $1-S/K$. Thus we estimate the number of bits needed to store one digit of this binary string with the entropy of the sequence

$$H_p := -\frac{S}{K}\log_2\frac{S}{K} - (1-\frac{S}{K})\log_2(1-\frac{S}{K}) \ ,$$

and the storage cost of X with $SNb + NKH_p$. Thus storing D and X will be more efficient than storing the original samples if and only if

$$Knb + SNb + NKH_p \leq Nnb \tag{4.2.4}$$

holds. In the case we are computing D only once on a set of samples and then using a sparse coding procedure to compute an encoding X of new incoming data, storing X is more efficient than the data itself if and only if

$$SNb + NKH_p \leq Nnb \tag{4.2.5}$$

holds.

5 Tree-based Dictionaries

In this Chapter we introduce our new method for dictionary learning, which was developed for the specific application of image patch-based sparse representation and compression but nonetheless is completely general and can be applied to any type of data. We will use the theoretical ideas of Chapter 2 to give the data a binary tree structure just as in the classical Haar wavelet transform; we will use this analogy as inspiration in building a dictionary with the differences of the clusters that are sibling nodes in this tree. When coupled with 2-means for clustering, this procedure is fast and gives a good quality dictionary.

We will start in Section 5.1 by giving a review of the relevant previous literature, then in Section 5.2 we will formulate a model which we think is more appropriate than (4.0.2) for the application we have in mind. In Section 5.3 we will describe the blueprint of our method in detail and finally in Section 5.4 we will carry out some numerical experiments through which we wish to find the best choices to make in the various steps of our method.

5.1 Previous literature

In this subsection we review two-papers that contain many concepts and ideas which we used for our method, described in the next subsection.

One of the characteristics of the K-SVD method is the necessity to present the data as a matrix Y, where each column is a data point. This means that if the original samples are multi-dimensional their geometrical structure (i.e. their spatial correlations) is ignored during the whole process. In Zeng et al. (2015) the authors develop a novel procedure for denoising patches $Y_1, \ldots, Y_N \in \mathbb{R}^{m \times n}$ extracted from a gray-valued image through dictionary learning which doesn't require vectorizing the patches. Instead of equation (4.0.2) they consider

$$\underset{D^L, D^R, X_1, \ldots, X_N}{\arg\min} \quad \sum_{i=1}^{N} \left|\left|(D^L)^T Y_i D^R - X_i\right|\right|_F + \lambda g_1(X_1, \ldots, X_N) + \mu g_2(X_1, \ldots, X_N) \ , \tag{5.1.1}$$

where g_1 and g_2 are two regularizing terms. Thus they aim to learn what they term the *dictionary pair* $D = (D^L, D^R)$, which corresponds to a bilinear encoding of the data (represented by the coding patches X_i) instead of the usual linear one considered in problem (4.0.2). The regularizing terms are needed for denoising: the hypotheses is that the domain in which the encoding patches live (which is determined by the bilinear transformation) is such that patches corresponding to noiseless patches in the original domain are characterized by low values of g_1 and g_2. These functions are defined as

$$g_1(X_1, \ldots, X_N) = \sum_{i=1}^{N} ||X_i||_1 \ ,$$

$$g_2(X_1, \ldots, X_N) = \sum_{i,j=1}^{N} w_{ij} \, ||X_i - X_j||_1 \ ,$$

where w_{ij} is a weight which is bigger the closer patches Y_i and Y_j are in Frobenius norm, namely they are given by a super Gaussian distribution:

$$w_{ij} = \begin{cases} \frac{1}{\Gamma} \exp(-(\frac{||Y_i - Y_j||_F}{2\sigma_i})^\tau) & \text{if } Y_i \text{ is } k\text{-nearest neighbor to } Y_j \ , \\ 0 & \text{otherwise} \ , \end{cases} ,$$

where Γ is a normalization factor chosen such that $\sum_{j=1}^{N} w_{ij} = 1$ for any $i = 1, \ldots, N$, σ_i is the variance of the neighborhood samples and τ a model parameter. Asking for a small norm of the encoding vectors (patches in this case) is typical of denoising procedures, while asking for small values of g_2 amounts to ask for some notion of continuity of the bilinear encoding: patches that are similar (i.e. that give high values of the weights w_{ij}) should have similar encodings.

Problem (5.1.1) is also NP-Hard: the authors propose an approximate method that consists of decoupling the dictionary learning and encoding problems, so it does not follow the scheme of Algorithm (7), like all the methods discussed in this Chapter. For learning the dictionary pair they describe what they term a Top-Bottom 2D Subspace Partition (TTSP) method, which consists of:

1. performing a simple bilateral 2DPCA on the samples $Y_1, \ldots, Y_N$ to approximate them with 1-dimensional matrices. This means computing the left and right image covariance matrices G_L and G_R and their first eigenvectors v and u, then for each $i = 1, \ldots, N$ the feature matrices $s_i := v^T Y_i u$. See Section 1.5.

2. Recursively applying 2-means to the values $s_1, \ldots, s_N$ in a hierarchical clustering procedure (which corresponds to a hierarchical clustering of the samples $Y_1, \ldots, Y_N$), obtaining a tree like described in Section 2.2. As stop criteria they use the depth of the tree and the number of patches in each node: when the first is too large or the second too small no branching happens on that node.

3. For each leaf node λ_k compute the d-dimensional simple bilateral 2DPCA (i.e. the first d eigenvectors of the left and right image covariance matrices computed from all those samples in the associated set $\mathcal{S}_{\lambda_k}$) and save the $m \times d$ and $n \times d$ eigenvector matrices V and U as sub-dictionary elements D_k^L and D_k^R. Stack the

left sub-dictionaries vertically and the right ones horizontally, i.e.:

$$D^L = \begin{bmatrix} D_1^L \\ \vdots \\ D_K^L \end{bmatrix} \quad , \quad D^R = \left[D_1^R | \dots | D_K^R \right] \ . \tag{5.1.2}$$

4. Use the Sub-dictionary merging (SM) algorithm to merge sub-dictionaries that generate very similar subspaces. The similarity is measured as the maximum cosine between vectors of the two sub-spaces, which the authors show can be computed with an SVD.

Example 6. We will give a numerical example that goes through steps 1-3 of the TTSP Algorithm described above. Suppose we are given the following dataset $\{Y_0, \dots, Y_7\} \subset \mathbb{R}^{3\times3}$:

$$Y_0 = \begin{bmatrix} 3 & -8 & -8 \\ -4 & 7 & 9 \\ 0 & -9 & -10 \end{bmatrix}, Y_1 = \begin{bmatrix} 7 & 5 & -1 \\ -10 & 4 & -10 \\ 5 & 9 & 4 \end{bmatrix}, Y_2 = \begin{bmatrix} -6 & -10 & 6 \\ -6 & 7 & -7 \\ -8 & -3 & -8 \end{bmatrix},$$
$$Y_3 = \begin{bmatrix} 5 & 6 & -3 \\ -1 & -7 & -4 \\ -9 & -8 & -9 \end{bmatrix}, Y_4 = \begin{bmatrix} 2 & -2 & -7 \\ 0 & -5 & -10 \\ 1 & -8 & 0 \end{bmatrix}, Y_5 = \begin{bmatrix} 3 & 8 & -6 \\ 5 & 1 & 2 \\ -4 & 3 & 9 \end{bmatrix},$$
$$Y_6 = \begin{bmatrix} 6 & -4 & 4 \\ -3 & 1 & -3 \\ -9 & 1 & -5 \end{bmatrix}, Y_7 = \begin{bmatrix} 8 & 7 & 2 \\ 8 & 7 & -9 \\ 9 & 2 & -1 \end{bmatrix} .$$

For the 2DPCA step we start by computing the right and left covarinace matrices (see (1.5.3) and (1.5.5)):

$$G_R = \begin{bmatrix} 86.34 & 28.41 & 14. \\ 28.41 & 107.53 & 25.72 \\ 14. & 25.72 & 102.98 \end{bmatrix} , \ G_L = \begin{bmatrix} 85.67 & -19.53 & 24.66 \\ -19.53 & 94.84 & 9.97 \\ 24.66 & 9.97 & 116.34 \end{bmatrix} ,$$

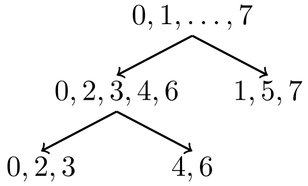
which have as first eigenvectors respectively

$$U = \begin{bmatrix} -0.46 \\ -0.7 \\ -0.56 \end{bmatrix} , \ V = \begin{bmatrix} 0.49 \\ -0.03 \\ 0.87 \end{bmatrix} .$$

We now can compute the feature scalars (one-dimensional approximations of the sampled patches) as $s_i = V^T Y_i U$ for $i = 0, \dots, 7$: we obtain

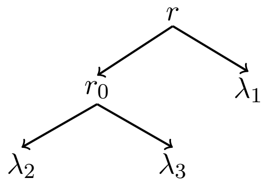
$$s_0 = -14.76,\ s_1 = 12.58,\ s_2 = -11.94,\ s_3 = -10.15,\ s_4 = -6.34,\ s_5 = 6.23,$$
$$s_6 = -4.23,\ s_7 = 8.96 \ .$$

By applying 2-means recursively on $\{s_0, \dots, s_7\}$ with a minimum cardinality of 3 we obtain the tree



,

where we're labelind the nodes with the indexes of the s_i that belong to that node. If we name the nodes like the following



,

then we have $\mathcal{S}_{\lambda_1} = \{1,5,7\}$, $\mathcal{S}_{\lambda_2} = \{0,2,3\}$ and $\mathcal{S}_{\lambda_3}\{4,6\}$. For $i = 1,2,3$ we compute the right and left covariance matrices G_R^i, G_L^i of the respective patches subsets $\mathcal{Y}_i = \{Y_k \text{ s.t. } k \in \mathcal{S}_{\lambda_i}\}$. We then compute for each $i = 1,2,3$ the first three eigenvectors of G_R^i and G_L^i, which we use as columns of U^i and V^i respectively. Finally we define $D_i^L := V^i$, $D_i^R := U^i$ and stack them to obtain the dictionaries:

$$D^L = \left[\begin{array}{ccc}
-0.97 & -0.22 & 0.03 \\
-0.22 & 0.93 & -0.29 \\
-0.04 & 0.29 & 0.96 \\
\hline
0.73 & -0.68 & 0.01 \\
-0.68 & -0.73 & 0.03 \\
0.01 & 0.03 & 1. \\
\hline
0.62 & -0.59 & 0.51 \\
-0.61 & 0.04 & 0.79 \\
0.49 & 0.81 & 0.33
\end{array}\right],$$

$$D^R = \left[\begin{array}{ccc|ccc|ccc}
1. & 0.04 & -0.08 & -0.89 & 0.46 & -0.04 & -0.62 & 0.73 & -0.3 \\
0.02 & -0.97 & -0.26 & 0.12 & 0.32 & 0.94 & 0.51 & 0.66 & 0.55 \\
0.09 & -0.26 & 0.96 & -0.45 & -0.83 & 0.34 & -0.6 & -0.19 & 0.78
\end{array}\right].$$

◇

For the coding of patch Y_i the authors don't use the complete dictionary pair (D^L, D^R), but limit themselves to the sub-dictionary pair (D_k^L, D_k^R) corresponding to the leaf to whose centroid the patch is closest in the feature domain, i.e.

$$\bar{k} \in \arg\min_{k} ||D_k^L(Y_i - C_k)D_k^R|| \ , \tag{5.1.3}$$

where C_k is the centroid of the sample set $\mathcal{S}_{\lambda_k}$. After this they use an iterative procedure to find an approximation of the optimal encoding patch X_i.

The authors then use their dictionary learning and encoding procedure to denoise the patches with the general iterative scheme described in Algorithm (9). They test their method on a variety of images with different noise levels and obtain comparable results to state of the art methods like curvelet denoising (see Starck et al. (2002)).

Algorithm 9 Patch denoising procedure through dictionary learning

```
Input: Noisy patches $\mathcal{Y} = \{Y_1, \dots, Y_N\}$
Output: Denoised patches $\mathcal{Y}_D$
Initialize $\mathcal{Y}_0 := \mathcal{Y}$, $k := 0$
while k < MAXITER do
    Learn dictionary $D_k$ from patches $\mathcal{Y}_k$
    Minimize sum of fidelity and regularizing terms
    Set $\mathcal{Y}_{k+1}$ as the set of reconstructed patches obtained from the previous step
    Set $k := k + 1$
end while
Set $\mathcal{Y}_D := \mathcal{Y}_k$
```

In Liu et al. (2018) (see also Li et al. (2018)) another procedure for denoising of two-dimensional patches extracted from images is proposed which also follows the iterative scheme of Algorithm (9). However here the coding operator is again linear: the dictionary learning procedure works with the two-dimensional patches, but once this is learned the patches and dictionary atoms are vectorized into matrices $Y = [vec(Y_1)|\dots|vec(Y_N)]$ and D respectively.

For the dictionary learning procedure (line (3) of Algorithm (9)) two procedures are proposed, both based on trees built from a hierarchical clustering of the patches like described in Section 2.2. The first proposed method recursively applies 2-means on the 1-dimensional approximation of the patches obtained through simple bilateral 2DPCA (so exactly like in the TTSP pocedure from Zeng et al. (2015)), using the number of patches in a node as stop criteria. The second method finds the patch with minimum Frobenius norm, calls it Y_1 and orders the patches by increasing distance from it, i.e. a succession $k_1 = 1, k_2, \dots, k_N$ is found such that $w_{k_1} \leq w_{k_2} \leq \dots \leq w_{k_N}$ where $w_j := ||Y_1 - Y_j||^2$. After this reordering 2-means is applied recursively on the distances w_k obtaining a hierarchical clustering tree.

Once the the binary tree is generated, for both the proposed methods the dictionary is composed by the same procedure. For each node ν in the tree let μ_ν be the centroid of the corresponding patches $\mathcal{S}_\nu$:

$$\mu_\nu = \frac{1}{|\mathcal{S}_\nu|} \sum_{s \in \mathcal{S}_\nu} s \ ,$$

and let $\tilde{\mu}_\nu$ be its rank-1 approximation given by SVD. Then for each non-leaf node ν with sons ν_0 and ν_1, the difference

$$\tilde{\mu}_{\nu_0} - \tilde{\mu}_{\nu_1} \tag{5.1.4}$$

is taken into the dictionary $\mathcal{D}$, as well as the rank-1 approximation of the global average $\tilde{\mu}_r$: these are the detail and approximation vectors associated to the tree discussed in Section 2 with the exception that here the authors use rank-1 approximations of the centroids - this is because they are interested in applying the dictionary learning procedure for denoising patches.

Example 7. We will give a numerical example of the first dictionary learning procedure proposed in Liu et al. (2018). Suppose we are given the same patches $\{Y_0, \ldots, Y_7\} \subset \mathbb{R}^{3\times 3}$ as in Example 6: the procedure to obtain the tree is exactly the same, but the dictionary elements are different. We compute the rank-1 approximation of the global average as well as, for each non-leaf node, the difference described in equation (5.1.4). Using the names for the nodes given in Example 6 we thus obtain the following 3 dictionary elements.

$$\tilde{r} = \begin{bmatrix} -0.05 & 0.02 & -0.11 \\ -1.76 & 0.62 & -4.02 \\ -1.11 & 0.39 & -2.53 \end{bmatrix}, \ \tilde{r}_0 - \tilde{\lambda}_1 = \begin{bmatrix} -6.32 & -8.29 & 0.13 \\ -4.29 & -5.41 & -1.35 \\ -7.69 & -9.14 & -5.87 \end{bmatrix},$$
$$\tilde{\lambda}_2 - \tilde{\lambda}_3 = \begin{bmatrix} -0.91 & -1.17 & -1.23 \\ 2.05 & 2.83 & 4.87 \\ -3.56 & -4.61 & -4.78 \end{bmatrix}.$$

◇

For the coding procedure (line (4) of Algorithm (9)), the problem considered consists of minimizing a fidelity term and two regularizing terms:

$$\min_X ||Y - DX||_F^2 + \lambda \, ||X||_1 + \mu Tr(XLX^T) \ . \tag{5.1.5}$$

The last term is called *graph regularizing* term and originated in the following manner: define a graph $G = (V, E)$ with $V = \{Y_1, \ldots, Y_N\}$ and $E = V \times V$ and define for any two vertices Y_i and Y_j a weight w_{ij} which is set to 1 if they are within the k-nearest neighbors to each other (measuring distances with the Frobenius norm) and to 0 otherwise. Defining the Laplacian matrix of the graph as $L = \Delta - W$, where $W = \{w_{ij}\}_{i,j=1,\ldots,N}$ and

$\Delta = diag(\sum_{j=1}^{N} w_{1j}, \dots, \sum_{j=1}^{N} w_{Nj})$, one has

$$\begin{aligned} Tr(XLX^T) &= \sum_{i,j=1}^{N} w_{ij} \left|\left|X_{\cdot i} - X_{\cdot j}\right|\right|_2^2 \\ &= \sum_{Y_i \sim Y_j} \left|\left|X_{\cdot i} - X_{\cdot j}\right|\right|_2^2 \end{aligned}$$

where we use the notation $Y_i \sim Y_j$ if and only if $w_{ij} = 1$. By supposing an analogous topology of the patches and the one induced by D on their encodings, we can suppose that $||X_{\cdot j} - X_{\cdot i}||_F$ will be small when $||Y_i - Y_j||_F$ is small. Thus the term $Tr(XLX^T)$ is inserted in (5.1.5) in order to ensure similarity of encodings for similar patches.
The authors apply the denoising scheme of Algorithm (9) to seismic two-dimensional data with artificial noise and obtain better results than the curvelet denoising procedure proposed in Hennenfent and Herrmann (2006).

5.2 A patch based model

We are interested in learning a dictionary to sparsely represent $m \times n$ patches extracted from a gray-valued image I. While we could use the vectorized patches as columns for a matrix Y and put the problem in the form of (4.0.2), this is not the real objective we're interested in and we will formulate here a more precise model that will reduce to (4.0.2) under certain conditions.
Suppose first of all we have a patch-extraction procedure Φ such that $\Phi(I) = (Y_1, \dots, Y_N)$ with $Y_i \in \mathbb{R}^{m \times n}$: in our numerical experiments we will extract non-overlapping patches, thus if the original dimension of the image is known Φ^{-1} is easily defined. Given the patches we wish to find *dictionary patches* $D_1, \dots, D_K \in \mathbb{R}^{m \times n}$ such that the sparsely reconstructed patches $\tilde{Y}_i := \sum_{j=1}^{K} x_j^i D_j$, with $\left|\left|x^i\right|\right|_0 \leq S$ give a good quality approximation $\tilde{I}$ of I when recomposed through Φ^{-1}. The optimization problem is thus

$$\min_{\substack{D_1,\dots,D_K \in \mathbb{R}^{m \times n} \\ x^1,\dots,x^N \in \mathbb{R}^K \\ \forall i=1,\dots,N \ \left|\left|x^i\right|\right|_0 \leq S}} \Delta(I, \tilde{I}) \ , \tag{5.2.1}$$

where Δ is some distortion measure and

$$\tilde{I} := \Phi^{-1}\left(\sum_{j=1}^{K} x_j^1 D_j, \sum_{j=1}^{K} x_j^2 D_j, \dots, \sum_{j=1}^{K} x_j^N D_j\right) \ .$$

The choice of the distortion measure Δ is application-dependent and should favor the properties we're interested in: if we want to approximate natural images with low storage cost (i.e. high sparsity) for the purpose of being later seen by humans, then it should correlate well with human perception of degradation of an image, and thus for example

$1-$ HaarPSI would be a good choice. If we're interested in approximating 2D data like seismographic data read from multiple sensors at multiple times, we may be interested in recovering as much as possible the correct values of the data pixel per pixel; in this case the Frobenius norm (which is classically used to formulate the dictionary learning problem) could be used.

Though minimizing $\Delta(I, \tilde{I})$ under the sparsity constraint is what we're really interested in, we will make the non-trivial assumption that the solution is obtained when we minimize distortion for each patch independently, i.e.

$$\min_{\substack{D_1,\ldots,D_K \in \mathbb{R}^{m\times n} \\ x^1,\ldots,x^N \in \mathbb{R}^K \\ \forall i=1,\ldots,N \ \|x^i\|_0 \leq S}} \sum_{i=1}^{N} \Delta(Y_i, \tilde{Y}_i) \ . \tag{5.2.2}$$

Ignoring the two-dimensional structure of the patches by using the Frobenius norm for Δ (or really any entry-wise norm), we can write the optimization problem in matrix form:

$$\begin{aligned}\min_{\substack{D_1,\ldots,D_K \in \mathbb{R}^{m\times n} \\ x^1,\ldots,x^N \in \mathbb{R}^K \\ \forall i=1,\ldots,N \ \|x^i\|_0 \leq S}} \sum_{i=1}^{N} \Delta(Y_i, \tilde{Y}_i) &= \min_{\substack{d_1,\ldots,d_K \in \mathbb{R}^{mn} \\ x^1,\ldots,x^N \in \mathbb{R}^K \\ \forall i=1,\ldots,N \ \|x^i\|_0 \leq S}} \sum_{i=1}^{N} \Delta(y^i, \sum_{j=1}^{K} x^i_j d^j) \\ &= \min_{\substack{D \in \mathbb{R}^{mn\times K} \\ X \in \mathbb{R}^{K\times N} \\ \forall i=1,\ldots,N \ \|X_{\cdot i}\|_0 \leq S}} \|Y - DX\|_F \ ,\end{aligned}$$

where $Y = [y^1|\ldots|y^N]$, $D = [d^1|\ldots|d^N]$ and $X = [x^1|\ldots|x^N]$ with $y^i := \operatorname{vec} Y_i$ and $d^j := \operatorname{vec} D_j$. This is exactly problem (4.0.2) with the choice of the Frobenius norm: we thus see that (5.2.2) and more so (5.2.1) are generalizations that take into account both the intrinsic two-dimensionality of the problem and the fact that the Frobenius norm does not always measure the type of distortion we are really interested in.

We must say that because of the huge complexity of the problem we won't try to develop a dictionary learning method specifically optimized for a certain Δ. We instead adopt an experimental approach of trying different possibilities in regards to details of our method (for example the choice of different clustering procedures), reconstruct $\tilde{I}$ and then evaluate $\Delta(I, \tilde{I})$, trying in this way to develop an empirical understanding of what works best. We will nonetheless try to preserve the two-dimensionality of the problem: this means that we'll settle for (5.2.2), as a compromise between (5.2.1) (what we really wish to optimize) and (4.0.2) (what is more simple and elegant to express mathematically). We'll see however that in practice what works best in terms of quality and time is using K-means for clustering and OMP for the sparse coding step, which both require the patches to be vectorized; we'll discuss this in more detail in Section 5.4.

Finally we want to mention the possibility to learn the dictionary not directly on the patches Y_i but on some bijective transformation of them $Z_i = f(Y_i)$. Once the dictionary patches $D_1, \ldots, D_K$ and the encoding vectors $x^1, \ldots, x^N$ have been learned then the reconstructed patches would be $\tilde{Y}_i = f^{-1}(\tilde{Z}_i) = f^{-1}(\sum_{j=1}^{K} x_j^i D_j)$. This means that if the function f is also linear then $\tilde{Y}_i = \sum_{j=1}^{K} x_j^i f^{-1}(D_j)$ and one can compute the dictionary patches on Z_i, compute $f^{-1}(D_j)$ and forget about f or its inverse; any needed encoding of a new patch can simply be obtained by using $f^{-1}(D_j)$ in lieu of D_j in the normally used sparse coding procedure. This idea is basically the same as that proposed in Rubinstein et al. (2010), where the authors propose to learn dictionary atoms as sparse linear combinations of some previously chosen basis like for example a wavelet basis. The motivation behind this procedure is that certain signals may be more sparse in a certain domain.

We can think of at least two cases in which using such a transformation f could be useful: in the presence of noise on the patches, f could be chosen as the simple bilateral 2DPCA $f(Y_i) := V^T Z_i U$ (where V and U are the eigenvector matrices of the left and right image covariance matrices respectively). In the case of dimensionality reduction though (which would be of real interest since it would remove the components with worse signal-to-noise ratio) the function f wouldn't be properly invertible: we could still learn the dictionary on the reduced patches Z_i and define the pseudo-inverse $\tilde{f}^{-1}(D_j) := V D_j U^T$. Secondarily we could think of using f to synthesize 2-dimensional information into vector form. As already mentioned above, in practice it is convenient to use the K-means algorithm for clustering which operates on vectors. We could then think of applying some wavelet or wavelet packet transform to the patches so as to re-encode their two-dimensional structure into vectors and then learn a sparse dictionary on those.

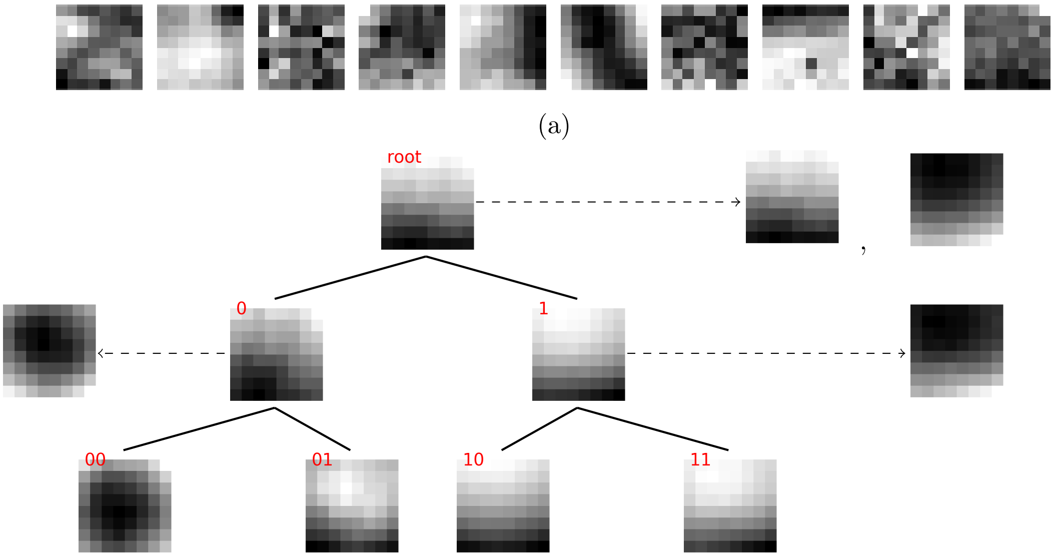
5.3 Tree-based dictionary

We are interested in using the main idea from Zeng et al. (2015) and Liu et al. (2018) to propose a solution to (5.2.2), namely recursively split the dataset in two and from the so obtained binary tree build a dictionary. We tried to simplify the construction in these two papers avoiding some of the arbitrary choices therein made and test the robustness of the main idea.

As anticipated in Section 2.2, we need a clustering method that can recursively split the dataset into classes sharing similar salient features. In subsection 1.7 we described some clustering algorithms, each of which provides a function $\mathcal{C} : 2^{\mathcal{S}} \to 2^{\mathcal{S}} \times 2^{\mathcal{S}}$ that will split any subset of $\mathcal{S}$ in two:

$$\forall S \subset \mathcal{S} \quad S = \mathcal{C}(S)_0 \sqcup \mathcal{C}(S)_1 \ .$$

Given a clustering method $\mathcal{C}$ we can build a binary tree that gives a hierarchical clustering of the data. This tree is constructed top-to-bottom, starting by splitting in two the whole dataset and then recursively applying the same procedure to the obtained clusters. The leafs of the tree, which we will also call *final clusters*, represent the smallest clusters we are willing to accept: we define a *branching criteria* that controls whether to cluster the set further or stop there, and this can be set for example to check if some quantity that somehow represents the cluster size (for example the cardinality or the variance) is above



(a)

(b)

Figure 5.1: (a) A small subset of the 8 × 8 sample patches obtained from the `flowers-pool` image (see Figure (5.10) on page 128). (b) The tree (thick edges) obtained from recursively applying 2-means to the sample set: the nodes in the tree are the centroids of the corresponding clusters. The dashed arrows show the Haar dictionary elements associated to the nodes (see text).

a fixed threshold. We will give more details on the branching criteria and the order of the tree visit in subsection 5.3.1; it should be already clear that the lower the threshold for the branching criteria is set the tighter the final clusters will be around the data, in the sense that the centroids of the final clusters will be a better approximation of the therein contained data. Equivalently a deeper tree will produce final clusters with fewer and more concentrated elements.

We try to give an intuitive idea of this hierarchical clustering procedure in Figure (5.1), where a small subset of sample patches together with the centroids corresponding to clusters of the first 3 levels of the tree are shown. As can be seen, each branching of the tree corresponds to a separation of a certain visual feature: the clustering procedure used (in this case 2-means) separates data points that are far away, thus determining a sum decomposition of the cluster centroid: in this representation a centroid patch is sum of its two sons in the tree[1].

Once the binary tree has been computed, we wish to use the information it encodes to define the dictionary atoms; these have to somehow give a good discretization of the

---

[1] this is actually not exactly true in Figure (5.1) because of a normalization factor that `matplotlib` automatically chooses such that the maximum value in a patch is 1 (white) and the minimum 0 (black).

sample set, in the sense that it should be possible to approximate each patch in $\mathcal{S}$ as a linear combination of very few (think 2 or 3) of these atoms. The most natural idea would be to pick the centroids of the final clusters as atoms; when taken together with the global average of all the samples, we'll call this dictionary the *centroids dictionary.* This dictionary potentially incurs in the non ideal situation of having many highly correlated atoms. In fact, in the regions where the sample set is clustered more finely (i.e. the subtrees branch deeper), the centroids of the clusters will be very close together. Thus we instead propose to take as dictionary atoms the differences of sibling cluster centroids, where by sibling we mean they correspond to two nodes in the tree which are sons of the same node; when taken together with the global average of all the samples, we'll call this dictionary the *Haar dictionary* or Haar-dict. If we think of the patches as elements of a vector space, the atoms we are taking are describing the direction needed to move from one centroid to the one of its sibling cluster. We are thus mitigating the risk of taking many highly correlated patches as dictionary atoms: even if the vectors that represent the centroids are very close to parallel, their differences are not necessarily.

To summarize, our method consists of recursively applying a bipartite clustering procedure to produce a binary tree from which we can build two dictionaries: the centroids dictionary composed by the centroids of the leaf nodes or the Haar dictionary composed of difference of the centroids of each non-leaf node's sons. In both cases we'll also take in the dictionary the global average (which is nothing else but the centroid corresponding to the root node of the tree). Incidentally both these procedures give the same number K of atoms, as the number of leafs in a binary tree is equal to the number of non-leaf nodes. See Figure (5.1b). In the following we will always suppose to be dealing with the Haar dictionary (unless otherwise specified) because of its superior performance; in the notation of Chapter 2, this means we are taking as dictionary

$$\mathbf{D} = \{\mathcal{A}(r)\} \cup \{\mathcal{D}(\nu) \mid \nu \in N \setminus N_\ell\} \ . \tag{5.3.1}$$

Like in Zeng et al. (2015) and Liu et al. (2018) and unlike the MOD or K-SVD method, we separate the dictionary learning and coding procedures: once we have learned the dictionary $\mathbf{D} = \{D_0, \ldots, D_{K-1}\}$ we define Y as the $mn \times N$ matrix with $\{\text{vec}(Y_i)\}_{i=1,\ldots,N}$ as columns and D as the $mn \times K$ matrix with $\text{vec}(D_k)/\left\|D_k\right\|_F$ as columns. We then use OMP to solve the sparse coding problem (4.0.1). The normalization of the dictionary atoms is necessary so that during sparse coding OMP won't assign artificially large scalar values to atoms with very small norm.

It should already be apparent that the choice and tuning of the clustering procedure is fundamental: it completely determines the clusters and the tree structure. The only way we're measuring the quality of a clustering procedure (a part from execution time) is by using the dictionary produced by our method for various sparse reconstruction tasks, and using standard quality assessment measures for the reconstruction; in other words we adopt a trial-and-error approach and check which choices give better solutions for problem (5.2.1). In some sense this is all we are interested in, nonetheless if we found some correlation between certain attributes of the clustering procedure and the quality of the generated dictionary, we could work to improve directly the details of the clustering procedure (possibly designing a new ad hoc one) to benefit our method. However we have not yet found a precise way to directly measure the quality of a clustering method, and thus we will simply somewhat generally ask that it should ideally partition the sets in such a way that data points with similar visual salient features end up in the same class.

5.3.1 The Algorithm

Algorithm 10 Haar-like tree based dictionary procedure

```
Input: Patches $\mathcal{Y} = \{Y_1, \dots, Y_N\}$, clustering procedure $C : 2^{\mathcal{S}\to 2^{\mathcal{S}}} \times 2^{\mathcal{S}}$, dictionary
    cardinality $K$ or parameters mincard and $\epsilon$
Output: Haar dependency tree $((V,E), r, \{\mathcal{S}_\nu\}_{\nu\in V}, \mathcal{A}, \mathcal{D})$, Dictionary $\mathbf{D} =$
    $\{D_0, D_1, \dots, D_{K-1}\}$
 1: Initialize $r := \mathcal{S}_r := \mathcal{Y}$, $\mathbf{D} := \{D_0\}$ with $D_0 := \mathcal{A}_r = \frac{1}{N}\sum_{i=1}^{N} Y_i$ and $E := \emptyset$
 2: Initialize tovisit = DataStructure()        ▷ This determines the type of tree visit
 3: tovisit.put($r$)
 4: while tovisit $\neq \emptyset$ do
 5:     $\nu$ = tovisit.get()
 6:     Partition $\mathcal{S}_\nu$ into $A, B = C(S_\nu)$
 7:     if BRANCH CRITERIA($\nu$) is TRUE then        ▷ This determines the tree depth
 8:         Define $\nu_0 := \mathcal{S}_{\nu_0} = A$, $\nu_1 := \mathcal{S}_{\nu_1} = B$, add edges $(\nu, \nu_0)$ and $(\nu, \nu_1)$ to $E$
 9:         Compute $\mathcal{A}_{\nu_0} := \frac{1}{|\mathcal{S}_{\nu_0}|}\sum_{Y\in A} Y$ and $\mathcal{A}_{\nu_1} := \frac{1}{|\mathcal{S}_{\nu_1}|}\sum_{Y\in B} Y$
10:         Compute $\mathcal{D}_\nu := \mathcal{A}_{\nu_0} - \mathcal{A}_{\nu_1}$
11:         Add $\mathcal{D}_\nu$ to $\mathbf{D}$
12:         tovisit.put($\nu_0$), tovisit.put($\nu_1$)
13:     end if
14: end while
```

In Algorithm (10) an outline of our procedure 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 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[2] 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 $\mathcal{S}_\nu$ is above a threshold `mincard`,

2. whether the *tightedness* of the cluster is above a threshold ϵ, where by tightedness we mean the value of the clustering minimization function (WCSS for 2-means and

---

[2] first in first out

Ncut for spectral clustering)[3].

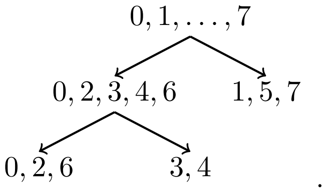
In this case we do not have a direct control over the cardinality K of the produced dictionary, we simply know that it will be a decreasing function of ϵ. On the other hand we have the certainty that the final clusters will be very small: they either must have fewer than `mincard` elements or, when partitioned 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 are 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 $\mathcal{S}_\nu$ is above a threshold `mincard`,
2. whether the number of branchings already occurred is smaller than a threshold `nbranchings`.

This means that, if the sample set is large enough, exactly `nbranchings` branchings will occur, and thus the Haar-dictionary will consist of `nbranchings` $+ 1$ dictionary elements. In this case we then have direct control over the number K of dictionary elements. In our tests (especially when doing comparing our methods to K-SVD) we will always use this priority queue variant because of the convenience of setting the dictionary cardinality K. Depending on the application though (if it is not so important to achieve a particular K) the FIFO variant might be more appropriate.

Example 8. We will describe the steps of the Algorithm using the same patches $\{Y_0, \dots, Y_7\} \subset \mathbb{R}^{3\times 3}$ as in Example 6 and 7. Suppose at first to use 2-means clustering and the FIFO queue variant, set `mincard` to 2 and the threshold for the WCSS value to 100; we then obtain the following tree



In fact 2-means applied to all the patches clusters them into $A = \{0, 2, 3, 4, 6\}$ and $B = \{1, 5, 7\}$ which give $\mathrm{WCSS}(A, B) = 195.3$; furthermore the root node contains 8 patches which is above the minimum cardinality of 2, thus the first branching occurs.

---

[3] otherwise we could use for example the variance of the corresponding set $\mathcal{S}_\nu$

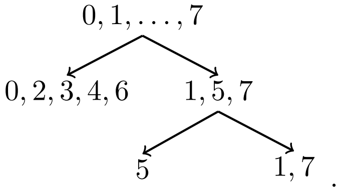
The two children nodes are put in the FIFO queue (starting with the one on the left), and thus at the next iteration the node correspoding to patches $\{0, 2, 3, 4, 6\}$ is visited. The 2-means clustering applied to these patches gives a WCSS value of 129.2 which is still above the threshold, furthermore there are 5 patches in the set, thus even here a branching on the tree occurs and the two sons are added to the FIFO queue. Now node r_1 corresponding to patches $\{1, 5, 7\}$ is visited, because it is on top of the FIFO queue: this doesn't satisfy the branching requirement since the WCSS value given by the eventual 2-means clustering is 73. Finally the two nodes $\{0, 2, 6\}$ and$\{3, 4\}$ are visited, both of which don't satisfy the branching criteria: the first one would give a WCSS value of 45.2 and the second doesn't have more than 2 patches. The FIFO queue is now empty and the tree visit stops. We now compute the global average $\mathcal{A}(r)$ and the detail vectors $\mathcal{D}(r)$ and $\mathcal{D}(r_0)$:

$$\mathcal{A}(r) = \frac{1}{8}\sum_{i=0}^{7} Y_i = \begin{bmatrix} 3.5 & 0.25 & -1.62 \\ -1.38 & 1.88 & -4. \\ -1.88 & -1.62 & -2.5 \end{bmatrix}$$

$$\mathcal{D}(r) = \frac{1}{5}\sum_{i\in\{0,2,3,4,6\}} Y_i - \frac{1}{3}\sum_{i\in\{1,5,7\}} Y_i = \begin{bmatrix} -4. & -10.27 & 0.07 \\ -3.8 & -3.4 & 2.67 \\ -8.33 & -10.07 & -10.4 \end{bmatrix}$$

$$\mathcal{D}(r_0) = \frac{1}{3}\sum_{i\in\{0,2,6\}} Y_i - \frac{1}{2}\sum_{i\in\{3,4\}} Y_i = \begin{bmatrix} -2.5 & -9.33 & 5.67 \\ -3.83 & 11. & 6.67 \\ -1.67 & 4.33 & -3.17 \end{bmatrix} .$$

The Haar-dictionary is then composed of the normalized versions (under the Frobenius norm) of these 3 patches. Suppose now that instead we want to use the priority queue variant along with the 2-means clustering; we set `nbranchings` to 3 and `mincard` to 2. The tree obtained is now the following:



In fact the first branching consists of 2-means applied to all the patches, thus the first two children are the same as in the FIFO queue case. However what is different is that now the two children are put in the priority queue using as key the variances of their corresponding patch-sets, which are 30.2 and 32.9 for the left and right son respectively. The second being highest means that node $\{1, 5, 7\}$ is extracted first to visit: this satisfies the cardinality requirement and thus the second and final branching occurs. Its sons are put in the priority queue with their variances which are 0 and 38.5 respectively. Now the branching criteria will always evaluate to `False` because the `nbranchings` threshold has been reached; it is interesting to note that the nodes still in the priority queue will be visited in order of decreasing variance, so first $\{1,7\}$, then $\{0,2,3,4,6\}$ and finally $\{5\}$. However if `nbranchings` were set to 4, the third branching would occur on node $\{0,2,3,4,6\}$, since $\{1,7\}$ wouldn't satisfy the cardinality requirement. We now compute the approximation and detail vectors: the first branching being the same as in the FIFO queue case means that $\mathcal{A}(r)$ and $\mathcal{D}(r)$ are the same. The third Haar-dictionary element here is instead the normalized version of

$$\mathcal{D}(r_1) = Y_5 - \frac{1}{2}\sum_{i\in\{1,7\}} Y_i = \begin{bmatrix} -4.5 & 2. & -6.5 \\ 6. & -4.5 & 11.5 \\ -11. & -2.5 & 7.5 \end{bmatrix} .$$

◇

Regarding the computational complexity of our method (including the computation of OMP at the end), this of course depends on the clustering procedure used and the number of branchings done (i.e. the number of nodes in the tree). Supposing we use the 2-means clustering by computing $\mathcal{I}$ iterations of Lloyd's algorithm, for each non-leaf node $\nu \in N \setminus N_\ell$ we require $O(|\mathcal{S}_\nu| n\mathcal{I})$ elementary operations for the clustering (see subsection 1.7.1) and $O(|\mathcal{S}_\nu|)$ for computing the associated dictionary element. Thus in this case the total computational cost is

$$\sum_{\nu\in N\setminus N_\ell} O(|\mathcal{S}_\nu| n\mathcal{I}) + NO(nS^3 + SKn) \le O(KNn\mathcal{I} + NnS^3 + NSKn) \ . \tag{5.3.2}$$

By comparing this with (4.2.3) we can see how the K-SVD is quadratic in the number of samples N while our method is linear.

5.4 Numerical experiments

In this Section we will carry out various reconstruction tasks using K-SVD and different variants of our method; we will compare the quality of the reconstructions and computation times. The implementation of our method was done in python[4] while for K-SVD we used the KSVD-box Matlab software[5]. All the numerical tests were run on a MacBook Pro Mid 2012 with an Intel Ivy Bridge i5 2.5Ghz CPU.

To build the binary Haar dependency tree we experimented with two clustering methods: the 2-means (K-means with K = 2) and spectral clustering. For details on these methods refer to Sections 1.7.1 and 1.7.3 respectively. Generally speaking, we will see that the 2-means would almost always be preferable: it is much faster, requires no choice of parameters and gives comparable if not better quality reconstructions.

Spectral clustering being a graph-based method, one has to define the similarity measure σ and the lower threshold $\rho \in [0,1)$ on the similarities in order to define the weights of the affinity matrix (see (1.7.6)). We experimented with three different similarity measures: HaarPSI and the gaussian kernels of the Frobenius and Earth Mover's Distance

---

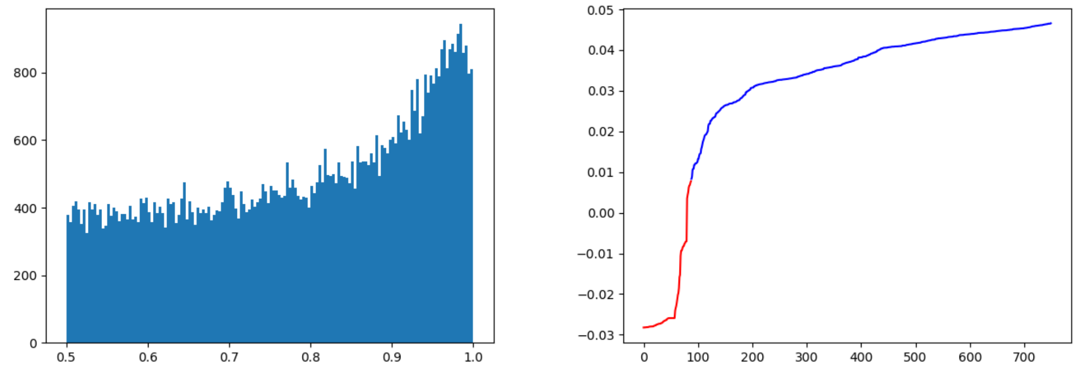
[4] code available at `https://github.com/nareto/haardict`

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

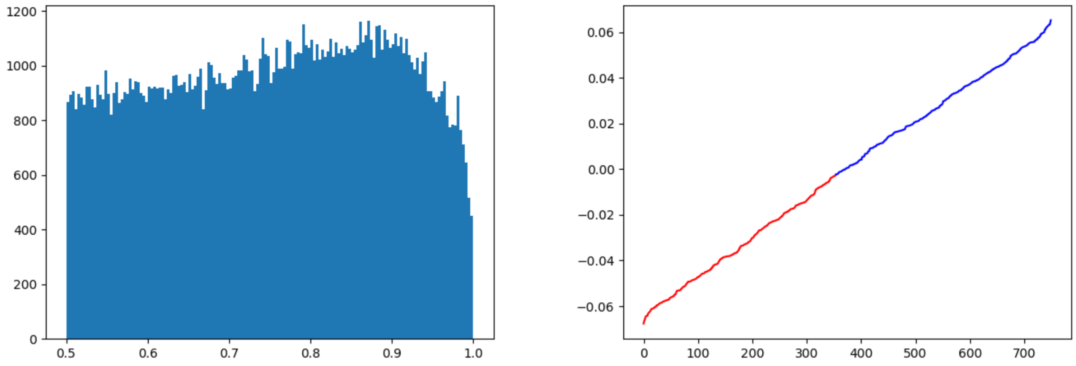
(see below). Suppose the data-set is composed of patches $Y_1, \dots, Y_N$, then according to Subsection 1.7.2 the entries of the respective affinity matrices W_H, W_F and W_E are defined as

$$
\begin{aligned}
(W_H)_{ij} &:= \mathrm{HaarPSI}(Y_i, Y_j) \\
(W_F)_{ij} &:= \mathbb{1}_{(\rho,1]}(e^{-\beta \frac{||Y_i - Y_j||_F^2}{\sigma^2}}) \\
(W_F)_{ij} &:= \mathbb{1}_{(\rho,1]}(e^{-\beta \frac{\mathrm{EMD}(Y_i, Y_j)^2}{\sigma^2}}) \ ,
\end{aligned}
$$

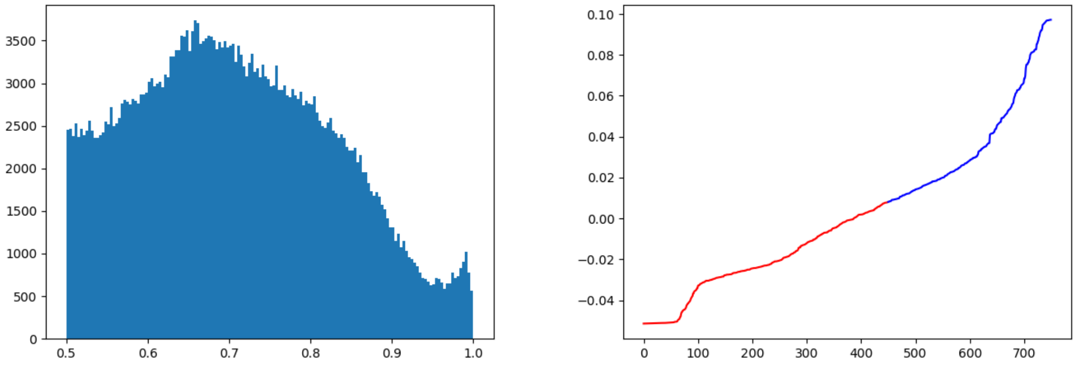
where σ is the variance of the data-set, $\rho \in (0, 1]$ and $\beta > 0$ are parameters and EMD stands for Earth Mover's Distance. The EMD (or Wasserstein's metric) is a distance between probability densities that represents the minimum cost of transforming one probability distribution into the other. It can be thought of as the minimum work necessary to transform one pile of dirt into another (with the same volume since the probability density functions are normalized); see Rubner et al. (2000). Here we are using the $p \times p$ patches themselves as probability densities defined on $[p] \times [p] \subset \mathbb{N} \times \mathbb{N}$. In the actual implementation we are using the `pyemd` python library which requires one-dimensional vectors as input; we identify the row-stacked pixel coordinates with $\gamma_0, \dots \gamma_{p^2}$, i.e. $\gamma_k = (\lfloor k/p \rfloor, k \text{ (mod p)})$, and then define the required metric matrix $M = \{m_{ij}\} \in \mathbb{R}^{p^2 \times p^2}$ as $m_{ij} = ||\gamma_i - \gamma_j||_2$.



(a) Earth mover's distance



(b) Frobenius norm



(c) HaarPSIl

Figure 5.2: On the left histograms of the entries of the affinity matrices W_E, W_F and W_H. The data set used consists of 750 8×8 patches randomly extracted from the `flowers-pool` image (see Figure (5.3)), and the parameter values used are $\rho = 0.5$ and $\beta = 0.06$. On the right the sorted values of the second eigenvectors of the corresponding Laplacian matrix L: the values below the threshold τ found by Otsu's method are colored in red, those above in blue.

To give a partial idea of the similarity distribution on a real set of image patches, we extracted 750 8×8 patches from the `flowers-pool` image (see Figure (5.3)) and plotted in Figure (5.2) the histograms for entries of W_E, W_F and W_H as well as the sorted values of the second eigenvectors of the corresponding Laplacian. These have been thresholded using Otsu's method (see Otsu (1979)), which steps through all possible threshold levels choosing the one that minimizes the sum of the variances of the values above and below the threshold. Incidentally in the following we always used the spectral clustering implementation in the `scikit-learn` python library, which instead uses k-means to threshold the eigenvector values.

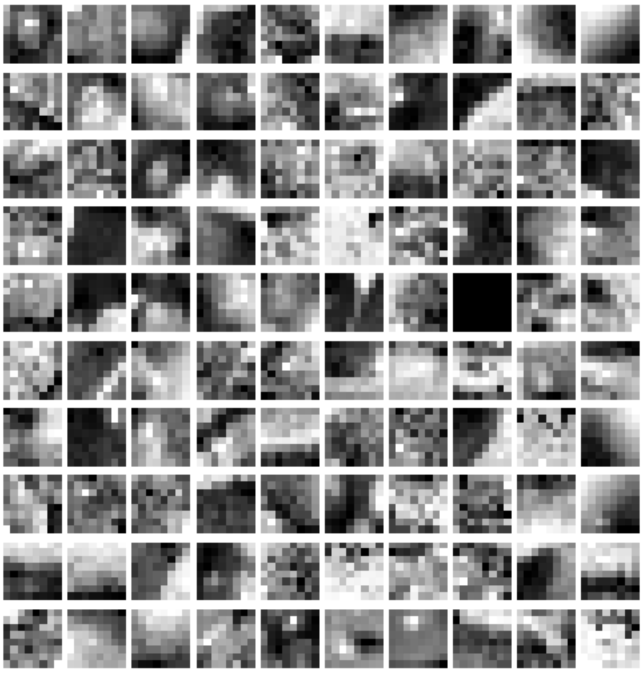


Figure 5.3: 100 8×8 patches randomly extracted from the `flowers-pool` image (Figure (5.10)).

As a first test to compare our method with different clustering procedures and K-SVD, we used patches of different sizes extracted from the `flowers-pool` image (see Figure (5.10)) to learn the dictionaries which we used to reconstruct with OMP the same patches in the image. We used patch sizes of 8×8, 16×16 and 32×32 and respectively dictionary cardinalities K equal to 100, 350 and 300, the latter being so small (and giving effectively an undercomplete dictionary) because the image contains only 1040 32×32 patches. For spectral clustering we used $\rho = 0.5$ and $\beta = 0.5$ to build the matrices W_H, W_E and W_F; for this clustering we had to limit the input data to 500 randomly chosen patches in order to keep the computation times reasonable and for this reason the requested dictionary cardinality wasn't always reached. For all the three similarity measures and for patch sizes 16×16 and 32×32 the computed dictionaries had cardinalities of approximately

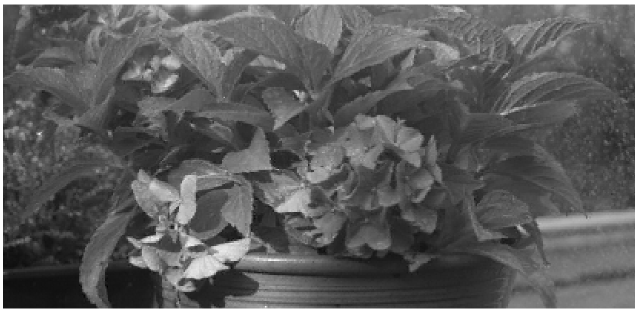
160. All the encoding procedures were done with a sparsity of $S = 5$. See Figures (5.4),(5.5) and tables 5.1-5.3 for the reconstruction results.

It can be seen that the spectral clustering, despite taking only a small fraction of image patches in input, is unfortunately too slow for practical purposes. The bottleneck is the computation of all the pairwise similarities needed to compute the affinity matrix W, which can get particularly expensive when the size of the patches increases; this is especially the case for the EMD distance, whose computation with the `pyemd` python library is unacceptably slow for patch sizes above 8×8 - therefore we didn't run the tests for larger patch sizes. The 2-means clustering is clearly preferable: it is much faster and gives better reconstructions. The K-SVD almost always gives the best quality reconstruction but is slower than our 2-means variant, especially for increasing patch-size, as was to be expected by simply comparing the complexities of K-SVD and our method. The apparent incoherence in the computation times for K-SVD in Table 5.3 is due to the number of patches rapidly decreasing with the increase of the patch size. It has also to be noted that the K-SVD code is quite mature and optimized while our implementation was designed more around being flexible to test many different variants rather than being fast.

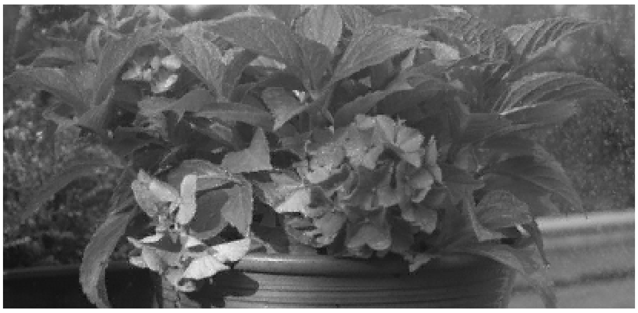
Here and in the following by most used atoms we mean the ones with greatest values of η_k where

$$\forall k = 1, \dots K \quad \eta_k := \sum_{j=1}^{N} |X_{kj}| \ , \tag{5.4.1}$$

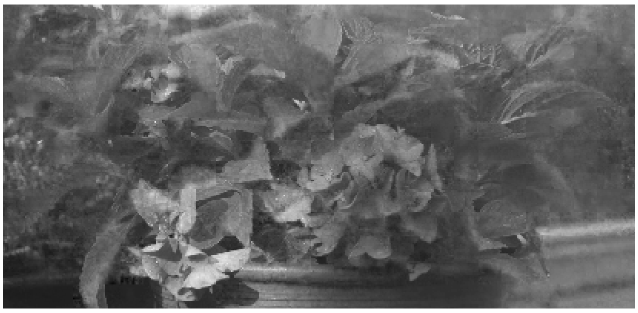
i.e. η_k is the total absolute value sum of the coefficients involving the dictionary atom k in the reconstruction of all patches. In Figure 5.6 we represented these vectors for two dictionaries: it can be seen here (and this is mostly the case in all the tests we've conducted) that there are few atoms that are used very frequently in the reconstruction and other atoms that are used with far less frequency. While for K-SVD the most used atoms seem to concentrate in the last part of the dictionary, the opposite is true for our method. In our code the dictionary is computed by revisiting the tree breadth-first, so the atoms D_k with low k 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 centroids of large clusters, i.e. they distinguish between features of the data at a very coarse level.



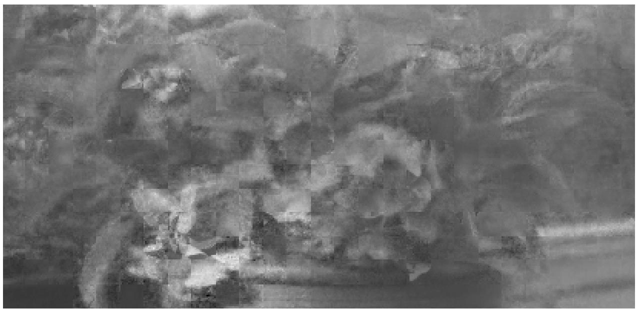
(a) 8 × 8 patches, `2-means` haar-dictionary



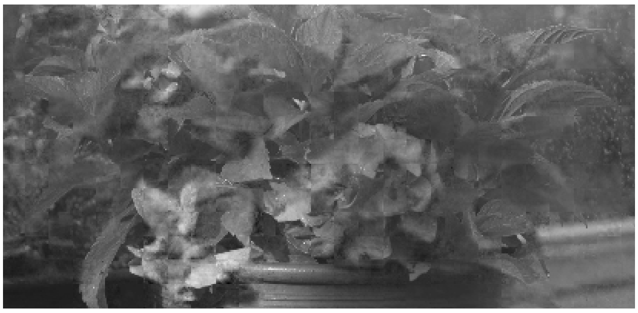
(b) 8 × 8 patches, `spectral-haarpsi` haar-dictionary



(c) 32 × 32 patches, `2-means` haar-dictionary

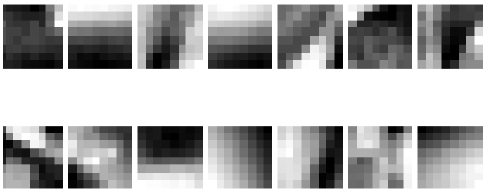


(d) 32 × 32 patches, `spectral-haarpsi` haar-dictionary

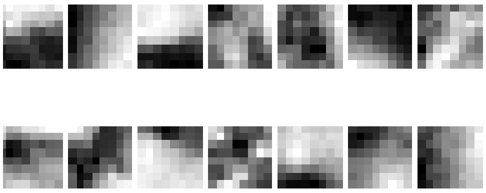


(e) 32 × 32 patches, `K-SVD` dictionary

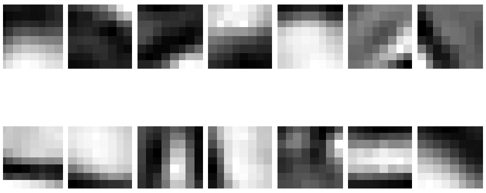
Figure 5.4: Cropped versions of the reconstructions of the `flowers-pool` image with various dictionaries.



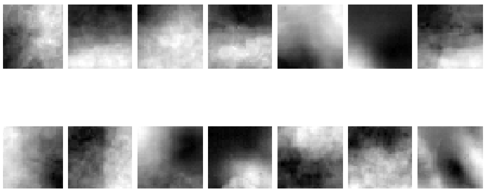
(a) 8 × 8 patches, `2-means` haar-dictionary



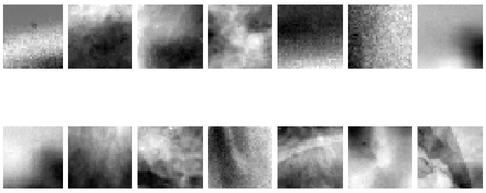
(b) 8 × 8 patches, `spectral-haarpsi` haar-dictionary



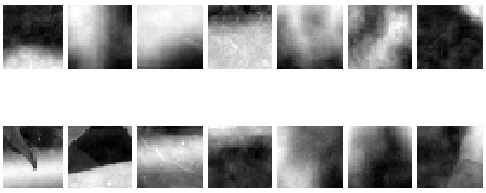
(c) 8 × 8 patches, `K-SVD` dictionary



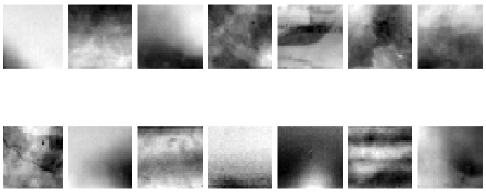
(d) 32 × 32 patches, `2-means` haar-dictionary



(e) 32 × 32 patches, `spectral-haarpsi` haar-dictionary



(f) 32 × 32 patches, `K-SVD` dictionary



(g) 32 × 32 patches, `spectral-frobenius` haar-dictionary

Figure 5.5: 14 most used atoms in reconstructing the `flowers-pool` image with various dictionaries.

Table 5.1: HaarPSI values for reconstruction of the `flowers-pool` image

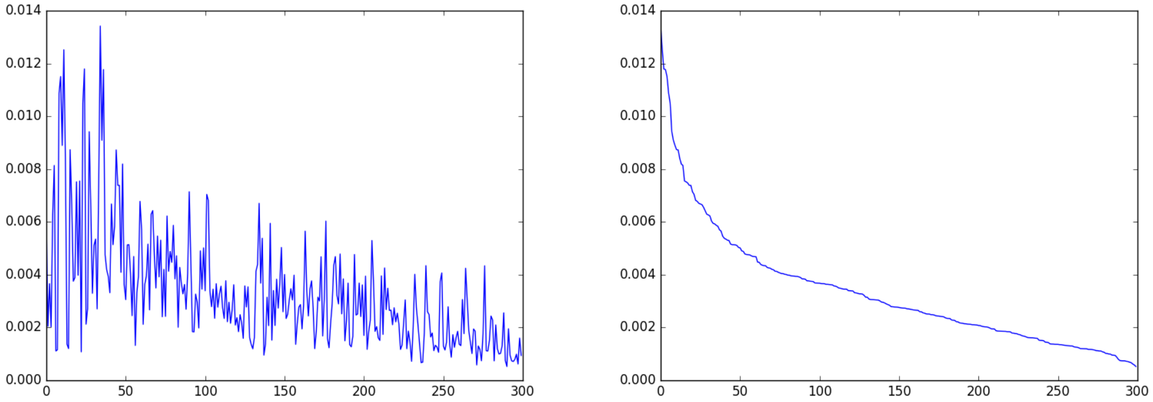
Patch size Method	8×8	16×16	32×32
2-means	0.807662	0.668130	0.591810
spectral-frobenius	0.787870	0.605407	0.489182
spectral-haarpsi	0.788303	0.588473	0.468913
spectral-emd	0.783695	-	-
K-SVD	0.819504	0.697072	0.577935

Table 5.2: PSNR values for reconstruction of the `flowers-pool` image

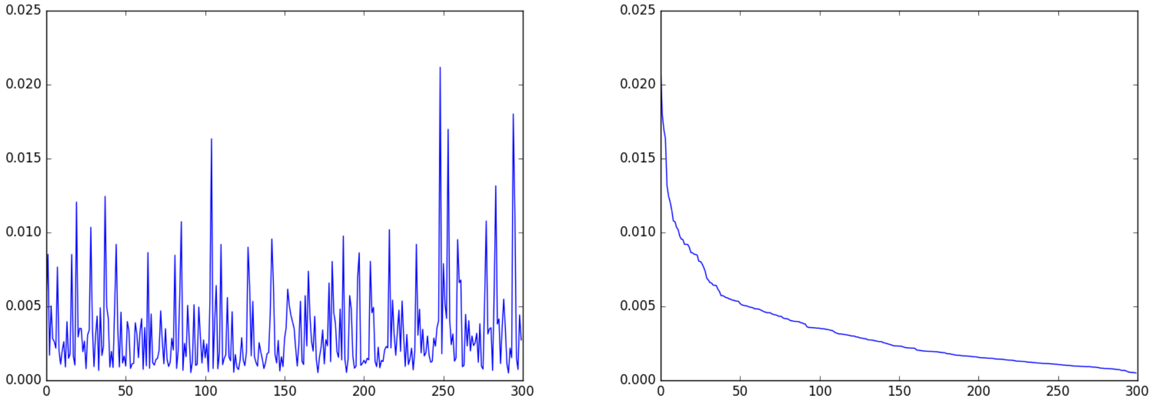
Patch size Method	8×8	16×16	32×32
2-means	36.73741	33.79289	32.32366
spectral-frobenius	36.56342	32.83815	30.38745
spectral-haarpsi	36.27714	32.35491	29.90539
spectral-emd	36.51739	-	-
K-SVD	37.57608	34.82362	32.20141

Table 5.3: Time (in seconds) for learning the dictionaries from the `flowers_pool` image

Patch size Method	8×8	16×16	32×32
2-means	3.03	8.35	8.18
spectral-frobenius	9.34	11.5	11.30
spectral-haarpsi	121.63	173.88	352.23
spectral-emd	172.40	-	-
K-SVD	18.37	21.25	13.59



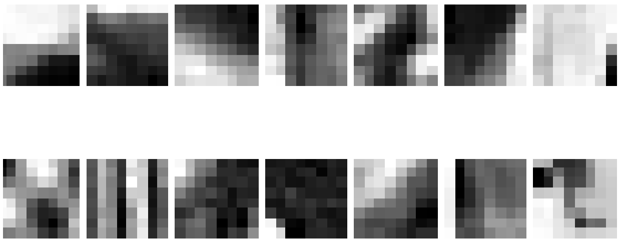
(a) Haar-dictionary



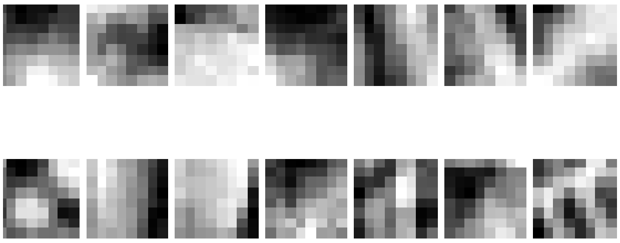
(b) K-SVD dictionary

Figure 5.6: Values of η_k defined in (5.4.1) for the Haar 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. The plots on the right show the sorted values of the corresponding η_k vector.

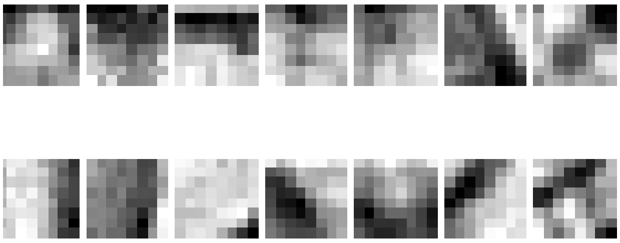
In the second test we wanted to give a fair comparison to spectral clustering: if we give it the same input patches as the other methods, does it do better? We randomly selected 300 patches of 8×8, 16×16 and 32×32 from the `cameraman`, `lena`, `barbara` and `peppers` images from Figure (5.9) and we learned dictionaries with cardinality $K = 85$ using the K-SVD and our method with both 2-means and spectral clustering (with $R = 0.5$ and $\beta = 0.5$) with the three similarity measures; we then used the so obtained dictionaries to reconstruct patches of the respective size from the `boat` image with sparsity $S = 5$. In Figure (5.7) the patches that were most used in this reconstruction are shown for the various dictionaries. Results for the quality of the reconstruction and time needed for learning the dictionaries are shown in Tables 5.4, 5.5 and 5.6. It can be seen that the spectral clustering results are indeed closer (with respect to `flowers_pool` image) to the 2-means variant and the K-SVD method, but not as much better as hoped and anyways much slower.



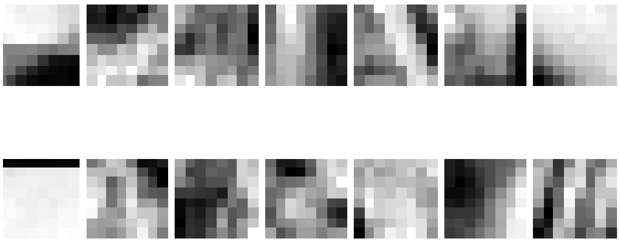
(a) `2-means`



(b) `spectral-frobenius`



(c) `spectral-haarpsi`



(d) `spectral-emd`

(e) `K-SVD`

Figure 5.7: 14 most used atoms (in reconstructing the `boat` image) for various 8×8 dictionaries learnt from the other images in Figure (5.9).

Table 5.4: HaarPSI values for reconstruction of the `boat` image

Patch size Method	8×8	16×16	32×32
2-means	0.598513	0.415514	0.286448
spectral-frobenius	0.617493	0.410687	0.288477
spectral-haarpsi	0.589914	0.413590	0.287448
spectral-emd	0.621066	-	-
K-SVD	0.653035	0.444661	0.293208

Table 5.5: PSNR values for reconstruction of the `boat` image

Patch size Method	8×8	16×16	32×32
2-means	29.98228	25.79827	22.42466
spectral-frobenius	30.50196	25.39697	22.34544
spectral-haarpsi	29.56240	25.31583	22.16454
spectral-emd	29.46130	-	-
K-SVD	30.59927	25.90940	22.37480

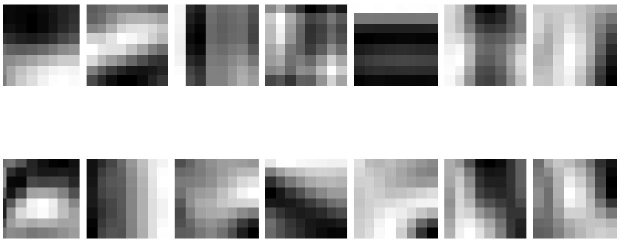
Table 5.6: Time (in seconds) for learning the dictionaries from the `cameraman`, `lena`, `barbara` and `peppers` images

Patch size Method	8×8	16×16	32×32
2-means	1.02	1.13	1.91
spectral-frobenius	4.37	4.86	5.01
spectral-haarpsi	49.04	61.80	125.92
spectral-emd	90.86	-	-
K-SVD	1.39	1.02	1.92

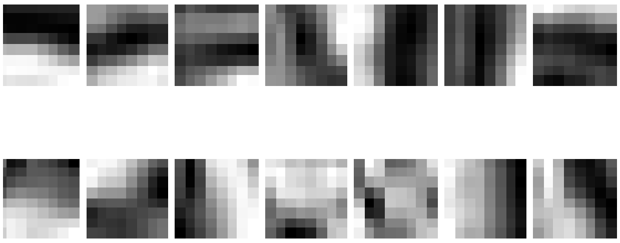
In the third test, following the idea mentioned in Section 5.2, we tried applying a 2DPCA transform on the same patch-set as in the second test before learning the dictionaries, and later applying the pseudo-inverse 2DPCA on the learned dictionary patches so as to obtain a dictionary in the original size. The purpose of this transformation is to improve the clustering step, in speed obviously (because clustering lower dimensional vectors is faster, especially for the similarity evaluation step in graph-based methods) and also in quality if we suppose a certain level of noise in the original patches. Some of the dictionary patches are shown in Figure (5.8) and the reconstruction results are shown in Tables 5.7, 5.8 and 5.9.

In general the 2DPCA procedure is very beneficial especially for the variants of our method: time computation (which in Table 5.9 is inclusive of the computation of the

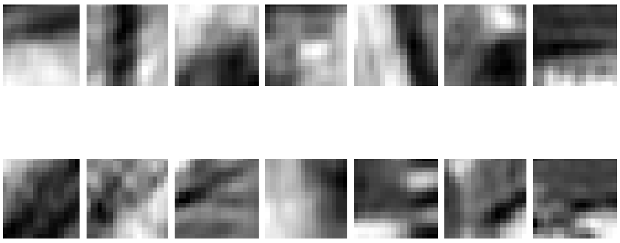
2DPCA transforms of the patches) goes down and the reconstruction quality goes up. The patches in the dictionaries look smoother and less noisy, though they might get too much smoothed if the size of the 2DPCA is too small with respect to the original patch size. We wish to remark that we repeated these tests with the full patch sets for the 2-means variant and K-SVD method and the results were different: though we still had a slight improvement in time, the quality of the reconstruction got worse. We deduce that the 2DPCA transformation may be useful when the input sample set is small.



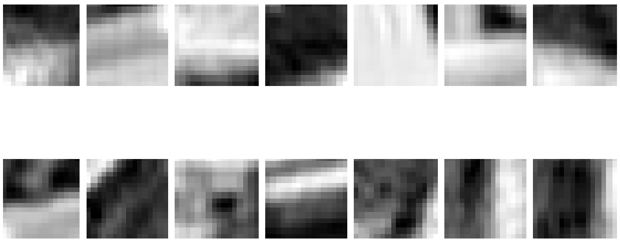
(a) 8×8 patches, `2-means` dictionary learned on 4×4 2DPCA



(b) 8×8 patches, K-SVD dictionary learned on 4×4 2DPCA



(c) 16×16 patches, `2-means` dictionary learned on 8×8 2DPCA



(d) 16×16 patches, K-SVD dictionary learned on 8×8 2DPCA

Figure 5.8: Various dictionary patches from the methods applied to the 2DPCA-transformed patches; the images always shows the 14 most used patches in the reconstruction of the `boat` image.

Table 5.7: 2DPCA HaarPSI values

Method \ Patch size	8×8 (4×4 2DPCA)	16×16 (4×4 2DPCA)	16×16 (8×8 2DPCA)
2-means	0.708048	0.438838	0.432387
spectral-frobenius	0.694233	0.443614	0.417306
spectral-haarpsi	0.696331	0.445463	0.441723
spectral-emd	0.695014	0.444121	0.437372
K-SVD	0.716982	0.451594	0.439915
	32×32 (8×8 2DPCA)	32×32 (16×16 2DPCA)	
2-means	0.308566	0.301944	
spectral-frobenius	0.301619	0.286519	
spectral-haarpsi	0.303371	0.287050	
K-SVD	0.308466	0.294213	

Table 5.8: 2DPCA PSNR values

Method \ Patch size	8×8 (4×4 2DPCA)	16×16 (4×4 2DPCA)	16×16 (8×8 2DPCA)
2-means	29.46641	25.55474	25.67997
spectral-frobenius	29.65418	25.52338	25.66238
spectral-haarpsi	29.67304	25.65105	25.97086
spectral-emd	29.51099	25.85567	25.62074
K-SVD	29.55693	26.34099	25.96887
	32×32 (8×8 2DPCA)	32×32 (16×16 2DPCA)	
2-means	22.59070	22.64134	
spectral-frobenius	23.33165	22.39385	
spectral-haarpsi	22.73070	23.56746	
K-SVD	22.66237	23.29147	

Table 5.9: Time (in seconds) for computing the 2DPCA and learning the dictionaries on the transformed patches

Patch size Method	8×8 (4×4 2DPCA)	16×16 (4×4 2DPCA)	16×16 (8×8 2DPCA)	32×32 (8×8 2DPCA)	32×32 (16×16 2DPCA)
2-means	0.99	0.98	1.18	1.05	1.20
spectral-frobenius	4.40	4.7	4.80	3.73	4.91
spectral-haarpsi	42.90	41.07	44.56	47.93	62.62
K-SVD	0.72	0.73	0.81	0.88	1.44

In the fourth test we wish to compare our method with the 2-means clustering with K-SVD on a larger number of patches. We used the `flowers-pool`, `house`, `cameraman` and `barbara` images (see Figure 5.9 and 5.10) to train the dictionaries which we used to re-encode the patches of the `landscape` image (see Figure 5.11). Alongside the Haar-dictionary we also computed the centroids dictionary, which is given by the centroids of the final clusters (the leafs) in the hierarchical clustering tree: it can be seen from Tables 5.10-5.11 that this is almost always worse than the Haar-dictionary. Furthermore from Table 5.12 it is clear that for bigger data-sets our method is much faster than K-SVD, especially considering (as already mentioned) that we are comparing our prototype code to the highly optimized KSVD-box software.

Table 5.10: PSNR values for reconstruction of the `landscape` image

Patch Size (# of patches)	K	K-SVD	Haar-dict	centroids-dict
8x8 (23264)	160	34.54001	34.40603	33.61384
16x16 (5776)	310	31.12586	31.27282	30.19550
24x24 (2496)	620	30.01312	29.51400	28.80906
32x32 (1424)	700	29.83463	28.64327	27.56030

Table 5.11: HaarPSI values for reconstruction of the `landscape` image

Patch Size (# of patches)	K	K-SVD	Haar-dict	centroids-dict
8x8 (23264)	160	0.702584	0.709226	0.699642
16x16 (5776)	310	0.518525	0.514718	0.508387
24x24 (2496)	620	0.448921	0.436503	0.442441
32x32 (1424)	700	0.402541	0.390120	0.392871

Table 5.12: Time (in seconds) for learning the dictionaries from the `flowers-pool`, `house`, `cameraman` and `barbara` images

Patch Size (# of patches)	K	K-SVD	Haar-dict
8x8 (23264)	160	21.91	4.51
16x16 (5776)	310	18.99	7.88
24x24 (2496)	620	33.18	13.02
32x32 (1424)	700	43.90	14.57

Finally we wish to compare our dictionary learning method in the specific application of image compression with the RBEPWT method presented in Chapter 3. To do this we took the 256×256 `peppers` image and applied our Haar-dictionary method with 2means clustering and different values of K and S to reconstruct the image. We then estimated the storage cost in bits of the dictionary as explained in Subsection 4.2.2 and used it to compute the Q index

$$Q = b256^2 \frac{\text{HaarPSI}}{Knb + SNb + NKH_p} ;$$

see also Subsection 3.3.3. The achieved values are reported in Table 5.13; a comparison with Table 3.3 (which we show again for convenience below as 5.14) shows that the RBEPWT gives much better compression quality. We must note though that for code simplicity in the haar-dictionary method we are reconstructing the image patch by patch with no overlap, and thus for low levels of K or S (i.e. high levels of compression) the image will show obvious signs of patchiness. We could achieve better results by following the usual overlap procedure in patch-based methods: for each pixel in the image reconstruct the patch centered around it and then average overlapping patches. In any case the haar-dictionary method is much faster, requiring in this case around 1 second while the Felzenszwalb-Huttenlocher segmentation and the RBEPWT encoding procedure require together around 5 minutes.

Table 5.13: Various reconstruction values for the `peppers` image with the Haar-dictionary method; see text.

K	sparsity	PSNR	HaarPSI	Q (with $b = 64$)
45	3	27.63951	0.542138	5.97
45	5	29.08661	0.611601	5.01
85	3	28.45634	0.606299	4.67
85	5	30.18983	0.668595	4.15

Table 5.14: `peppers` with Felzenszwalb-Huttenlocher segmentation

encoding	coefficients	PSNR	HaarPSI	Q (with $b = 64$)
easypath	4096	29.19485	0.803416	11.32
gradpath	4096	29.12663	0.804430	11.06
epwt	4096	30.13342	0.845705	2.95
tensor	4096	31.45586	0.853261	12.59
easypath	2048	25.63808	0.686624	18.28
gradpath	2048	25.66487	0.690167	17.53
epwt	2048	26.72259	0.749571	2.96
tensor	2048	27.10352	0.726001	21.11
easypath	1024	23.15176	0.580086	28.13
gradpath	1024	23.15506	0.583367	26.02
epwt	1024	24.17585	0.646684	2.74
tensor	1024	23.77188	0.602107	34.53
easypath	512	21.33941	0.502156	41.76
gradpath	512	21.34081	0.495794	35.86
epwt	512	22.05342	0.548996	2.41
tensor	512	21.40979	0.489784	55.39

(a) 256 × 256 `cameraman` image (b) 512 × 512 `lena` image

(c) 512 × 512 `barbara` image (d) 256 × 256 `peppers` image

(e) 512 × 512 `boat` image (f) 256 × 256 `house` image

Figure 5.9: A set of images used for some of the numerical experiments (see Text)

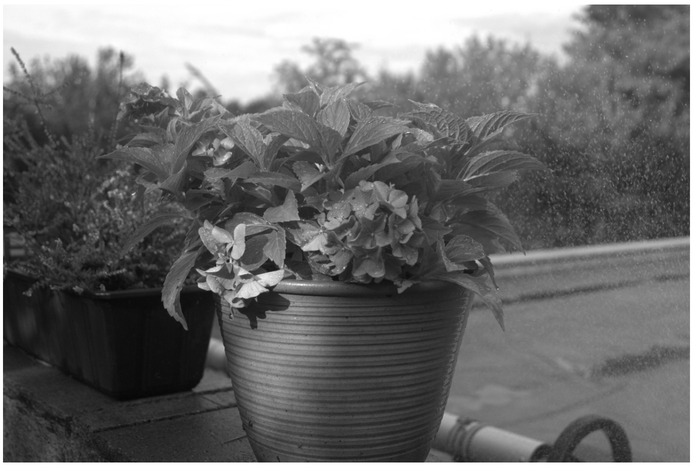


Figure 5.10: The 1287 × 857 `flowers-pool` image

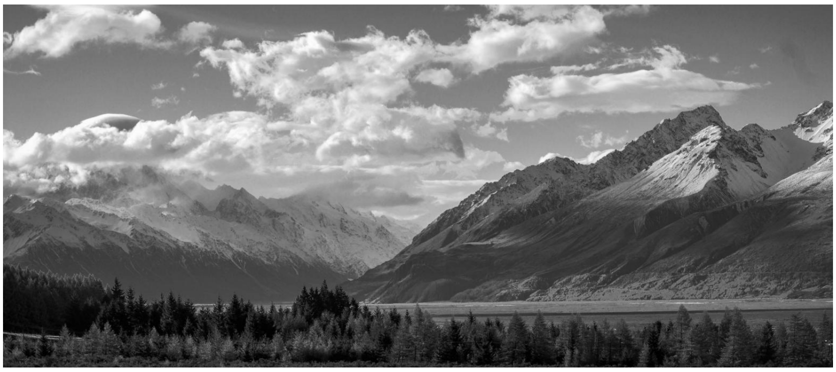


Figure 5.11: The 1555 × 680 `landscape` image

6 Conclusion

In this thesis we described two adaptive methods that are effective in sparsely representing images, the first (RBEPWT) being specific to this application and the second (Haar-dict) inserted in the more general context of dictionary learning. The RBEPWT finds a basis that depends only on the segmentation of the image: when sparsity is enforced on this basis by keeping only a small number of the largest coefficients, the reconstructed images preserve very effectively the edges corresponding to segmentation borders and the textures therein. In most of the numerical trials the method is slightly worse than EPWT in terms of quality of the reconstructed image but is cheaper in storage cost, as can be seen if we compare the ratio between reconstruction quality and storage cost of the encoding. Furthermore the RBEPWT allows one to encode and reconstruct a single region of interest in the image with different quality: this is a unique feature of the transform which works by leveraging the inherent tree structure in the coefficients of a wavelet transform. In this case there is also a permutation of the samples at each level of the tree due to the path-finding procedure; since this is carried out independently in each region, there is no mixing of coefficients from different regions in the image and it is possible to use a close to minimal number of coefficients to perfectly reconstruct the region of interest while still maintaining some information on the rest of the image. This phenomena is due to coefficients higher up in the tree being responsible for the encoding of non-local information. Future developments of the RBEPWT should include some ad hoc segmentation method which is coupled with the path-finding procedure: this would enable to obtain regions optimized for the transform and possibly prove some error bounds on the reconstructed image.

We then used another generalization of the Haar wavelet tree structure to define the fast, non-iterative Haar-dict method. This employs a clustering method (our numerical results suggest 2-means as the most practical) to recursively partition the data-set in two, thus building a tree where every branching corresponds to a splitting of the data. The computational cost of the method is linear in the number of data points used for learning, while state-of-the-art K-SVD depends quadratically on it. Furthermore the Haar-dict method provides a geometrical perspective into the dictionary learning problem: the tree gives a concept of resolution level of the space, with branches going deeper where variation in the data-set is higher. The dictionary atoms, taken as normalized differences of centroids of clusters at the same resolution level, have different degrees of importance in the final dictionary: OMP seems to prefer dictionary atoms from higher levels in the tree which correspond to difference vectors between very large clusters, i.e. those that distinguish at a coarse level between fundamental features in the data set. Future studies on this method should include experimenting with other clustering methods and non-binary trees (corresponding to wavelet transforms with longer filters) as well as further investigating the relation between a certain patch collocated in the data space and the dictionary atoms used to reconstruct it. Would it be possible for example to use the tree and wavelet synthesis formulas to define a new sparse coding procedure, alternative to the Matching Pursuit methods?

Bibliography

Aharon, M., Elad, M., and Bruckstein, A. (2006a). K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322.

Aharon, M., Elad, M., and Bruckstein, A. M. (2006b). On the uniqueness of overcomplete dictionaries, and a practical way to retrieve them. *Linear algebra and its applications*, 416(1):48–67.

Aloise, D., Deshpande, A., Hansen, P., and Popat, P. (2009). NP-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248.

Blumensath, T. and Davies, M. E. (2008a). Gradient pursuits. *IEEE Transactions on Signal Processing*, 56(6):2370–2382.

Blumensath, T. and Davies, M. E. (2008b). Iterative thresholding for sparse approximations. *Journal of Fourier analysis and Applications*, 14(5-6):629–654.

Bryt, O. and Elad, M. (2008). Compression of facial images using the k-svd algorithm. *Journal of Visual Communication and Image Representation*, 19(4):270–282.

Budinich, R. (2017a). Image compression with the region based easy path wavelet transform. *PAMM*, 17(1):831–832.

Budinich, R. (2017b). A region-based easy-path wavelet transform for sparse image representation. *International Journal of Wavelets, Multiresolution and Information Processing*, 15(05). 1750045.

Candes, E., Demanet, L., Donoho, D., and Ying, L. (2006). Fast discrete curvelet transforms. *Multiscale Modeling & Simulation*, 5(3):861–899.

Candès, E. J. and Donoho, D. L. (2004). New tight frames of curvelets and optimal representations of objects with piecewise c2 singularities. *Communications on pure and applied mathematics*, 57(2):219–266.

Chen, S. S., Donoho, D. L., and Saunders, M. A. (2001). Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159.

Cover, T. M. and Thomas, J. A. (2012). *Elements of information theory.* John Wiley & Sons.

Damelin, S. B. and Miller Jr, W. (2012). *The mathematics of signal processing*, volume 48. Cambridge University Press.

Daubechies, I. (1988). Orthonormal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 41(7):909–996.

Daubechies, I. (1992). *Ten lectures on wavelets*, volume 61. Siam.

Ding, C. and He, X. (2004). K-means clustering via principal component analysis. In *Proceedings of the twenty-first international conference on Machine learning*, page 29. ACM.

Donoho, D. L. (2006). For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution. *Communications on pure and applied mathematics*, 59(6):797–829.

Donoho, D. L. and Huo, X. (2001). Uncertainty principles and ideal atomic decomposition. *IEEE transactions on information theory*, 47(7):2845–2862.

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

Echeverri, E. (2006). Limits of capacity for the exchange of information in the human nervous system. *IEEE Transactions on Information Technology in Biomedicine*, 10(4).

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

Elad, M. and Aharon, M. (2006). Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image processing*, 15(12):3736–3745.

Elad, M. and Bruckstein, A. M. (2002). A generalized uncertainty principle and sparse representation in pairs of bases. *IEEE Transactions on Information Theory*, 48(9):2558–2567.

Engan, K., Aase, S. O., and Husoy, J. H. (1999). Method of optimal directions for frame design. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 5, pages 2443–2446. IEEE.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181.

Gavish, M., Nadler, B., and Coifman, R. R. (2010). Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *ICML*, pages 367–374.

Guo, K. and Labate, D. (2007). Optimally sparse multidimensional representation using shearlets. *SIAM journal on mathematical analysis*, 39(1):298–318.

Guo, K., Labate, D., Lim, W.-Q., Weiss, G., and Wilson, E. (2004). Wavelets with composite dilations. *Electronic research announcements of the American Mathematical Society*, 10(9):78–87.

Heinen, D. and Plonka, G. (2012). Wavelet shrinkage on paths for denoising of scattered data. *Results in Mathematics*, 62(3):337–354.

Hennenfent, G. and Herrmann, F. J. (2006). Seismic denoising with nonuniformly sampled curvelets. *Computing in Science & Engineering*, 8(3):16–25.

Kalcsics, J., Nickel, S., Puerto, J., and Tamir, A. (2002). Algorithmic results for ordered median problems. *Operations Research Letters*, 30(3):149–158.

Kong, H., Wang, L., Teoh, E. K., Li, X., Wang, J.-G., and Venkateswarlu, R. (2005). Generalized 2d principal component analysis for face image representation and recognition. *Neural Networks*, 18(5):585–594.

Larson, E. C. and Chandler, D. M. (2010). Most apparent distortion: full-reference image quality assessment and the role of strategy. *J. Electronic Imaging*, 19(1):011006.

Le Pennec, E. and Mallat, S. (2005). Bandelet image approximation and compression. *Multiscale Modeling & Simulation*, 4(3):992–1039.

Li, H.-y., Chao, Y., and Wang, H.-y. (2018). A fast dictionary learning algorithm for image denoising. *DEStech Transactions on Computer Science and Engineering*, (mso).

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

Mallat, S. (2008). *A wavelet tour of signal processing: the sparse way*. Academic press.

Mallat, S. (2009). Geometrical grouplets. *Applied and Computational Harmonic Analysis*, 26(2):161–180.

Mallat, S. G. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415.

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

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

Nickel, S. and Puerto, J. (1999). A unified approach to network location problems. *Networks: An International Journal*, 34(4):283–290.

Olshausen, B. A. and Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607.

Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66.

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.

Peyré, G. (2011). A review of adaptive image representations. *IEEE Journal of Selected Topics in Signal Processing*, 5(5):896–911.

Plonka, G. (2009). The easy path wavelet transform: A new adaptive wavelet transform for sparse representation of two-dimensional data. *Multiscale Modeling & Simulation*, 7(3):1474–1496.

Plonka, G., Iske, A., and Tenorth, S. (2013). Optimal representation of piecewise Hölder smooth bivariate functions by the easy path wavelet transform. *Journal of Approximation Theory*, 176:42–67.

Plonka, G., Tenorth, S., and Iske, A. (2012). Optimally sparse image representation by the easy path wavelet transform. *International Journal of Wavelets, Multiresolution and Information Processing*, 10(01):1250007.

Plonka, G., Tenorth, S., and Rosca, D. (2011). A new hybrid method for image approximation using the easy path wavelet transform. *IEEE Transactions on Image Processing*, 20(2):372–381.

Ponomarenko, N., Jin, L., Ieremeiev, O., Lukin, V., Egiazarian, K., Astola, J., Vozel, B., Chehdi, K., Carli, M., Battisti, F., and Kuo, C.-C. J. (2015). Image database tid2013: Peculiarities, results and perspectives. *Signal Processing: Image Communication*, 30:57 – 77.

Ram, I., Elad, M., and Cohen, I. (2011). Generalized tree-based wavelet transform. *IEEE Transactions on Signal Processing*, 59(9):4199–4209.

Rao, S., Mobahi, H., Yang, A., Sastry, S., and Ma, Y. (2010). Natural image segmentation with adaptive texture and boundary encoding. *Computer Vision–ACCV 2009*, pages 135–146.

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

Rubinstein, R., Zibulevsky, M., and Elad, M. (2008). Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit.

Rubinstein, R., Zibulevsky, M., and Elad, M. (2010). Double sparsity: Learning sparse dictionaries for sparse signal approximation. *IEEE Transactions on signal processing*, 58(3):1553–1564.

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

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98.

Starck, J.-L., Candès, E. J., and Donoho, D. L. (2002). The curvelet transform for image denoising. *IEEE Transactions on image processing*, 11(6):670–684.

Stoer, J. and Bulirsch, R. (2013). *Introduction to numerical analysis*, volume 12. Springer Science & Business Media.

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

Trefethen, L. N. and Bau III, D. (1997). *Numerical linear algebra*, volume 50. Siam.

Turk, M. and Pentland, A. (1991). Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

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.

Yang, J. and Liu, C. (2007). Horizontal and vertical 2DPCA-based discriminant analysis for face verification on a large-scale database. *IEEE Transactions on Information Forensics and Security*, 2(4):781–792.

Yang, J., Zhang, D., Frangi, A. F., and Yang, J.-y. (2004). Two-dimensional PCA: a new approach to appearance-based face representation and recognition. *IEEE transactions on pattern analysis and machine intelligence*, 26(1):131–137.

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

Zhang, D., Jing, X., and Yang, J. (2006). *Biometric image discrimination technologies.* IGI Global.

Zhang, D. and Zhou, Z.-H. (2005). (2D) 2PCA: Two-directional two-dimensional PCA for efficient face representation and recognition. *Neurocomputing*, 69(1):224–231.

Zhang, L., Shen, Y., and Li, H. (2014). VSI: A visual saliency-induced index for perceptual image quality assessment. *IEEE Transactions on Image Processing*, 23(13).

Renato Budinich | Curriculum Vitae

Institut für Numerische und Angewandte Mathematik, Lotzestrasse 16-18
37083 Göttingen – Germany

✉ r.budinich@math.uni-goettingen.de • nareto

Qualifications

University of Göttingen **Göttingen**
Doctoral student in the program "Mathematical sciences" *since 2015*
Adivsor: Prof. Gerlind Plonka-Hoch

University of Pisa **Pisa**
Master of Science in Mathematics, 110/110 *July 2015*
Thesis title: *A constrained maximum entropy principle for data reduction via pattern selection*
Advisors: Professors Giovanni Punzi, Maria Michela Del Viva, Vladimir Georgiev

University of Trieste **Trieste**
Bachelor of Mathematics, 110/110 cum laude *2011*
Thesis title: *Fourier Series: some classical results on convergence and non convergence*
original Italian title: "Serie di Fourier: alcuni risultati classici di convergenza e di non convergenza"
Advisor: Prof. Alessandro Fonda

Pubblications

- Budinich, R. (2017). A region-based easy-path wavelet transform for sparse image representation. *International Journal of Wavelets, Multiresolution and Information Processing*, 15(05):1750045.
- Budinich, R. (2017). Image compression with the region based easy path wavelet transform. *PAMM*, 17(1):831–832.

Conferences and Workshops

Bebra, Germany
3rd RTG2088 Workshop (Talk) *09/2018*

Bologna, Italy
SIAM Conference on Imaging Science (Talk) *06/2018*

Eddigehausen, Germany
2nd RTG2088 Workshop (Talk) *09/2017*

Bremen, Germany
GAMM-MSIO Mathematical Signal Processing and Data Analysis (Talk) *09/2017*

Speyer, Germany
GEMMI Geomathematics Meets Medical Imaging (Talk) *09/2017*

Shanghai, China	
International Conference on Computational Harmonic Analysis 2017 (Talk)	*05/2017*
Weimar, Germany	
88th GAMM Annual Meeting (Talk)	*03/2017*
Bestwig, Germany	
27th Rhein-Ruhr-Workshop (Talk)	*01/2017*
Luisenthal, Germany	
1st RTG2088 Workshop (Talk)	*10/2016*
Alba di Canazei, Italy	
4th Dolomites Workshop on Constructive Approximation and Applications (Talk)	*09/2016*
Berlin, Germany	
BMS Summer School on Mathematical and Numerical Methods in Image Processing	*08/2016*
Osnabrück, Germany	
International Workshop on Mathematical Imaging and Emerging Modalities	*06/2016*
Bestwig, Germany	
26th Rhein-Ruhr-Workshop	*01/2016*
Paris, France	
Mathematics and Image Analysis	*01/2016*

Further Experiences

Trieste, Italy	
Apprenticeship at Oceanography and Experimental Geophysics Institute (OGS)	*02/2011*