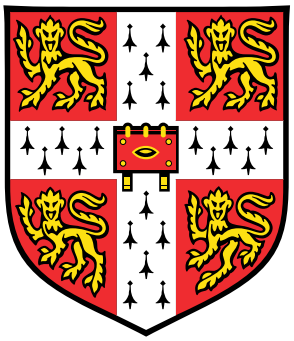
Compressive Sensing in Video Reconstruction



Brian Azizi

Supervisor: Dr Anita Faul

Department of Physics
Centre for Scientific Computing
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Selwyn College August 2016

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 15,000 words including appendices, bibliography, footnotes and tables.

Brian Azizi
August 2016

Acknowledgements

Abstract

This thesis details the development of a Compressive Sensing algorithm that allows for near-perfect reconstruction of video signals from measurement sets as small as 30% the size of the original signal.

Table of contents

Chapter 1

Introduction

The goal of the MPhil project is an implementation of a generic Compressive Sensing algorithm that can efficiently reconstruct video signals from a small number of measurements.

Compressive Sensing (CS) [2, 5] is a relatively recent framework within digital signal processing. Using the techniques in CS, we can measure signals *directly in a compressed format*, leading to highly efficient data acquisition protocols. In application areas such as magnetic resonance imaging (MRI), which has an inherently slow data acquisition process, this could potentially result in large scan time reductions, thus helping patients and doctors.

There are two major components to any CS system. Let $\boldsymbol{v} \in \mathbb{R}^M$ denote the digital signal of interest. The first component is a *sensing mechanism* that acquires the measurements

$$\boldsymbol{\Theta v} = \boldsymbol{y} \tag{1.1}$$

where $\boldsymbol{\Theta}$ is a known $N \times M$ matrix with $N << M$. The second component is a *reconstruction algorithm* that recovers the original signal $\boldsymbol{v}$ from the CS measurements $\boldsymbol{y}$ by finding a solution to the under-determined system (1.1).

In order to solve the system, we make the assumption that $\boldsymbol{v}$ has a sparse representation. This means that we can perform a change-of-basis transformation

$$\boldsymbol{v} = \boldsymbol{\Psi w} \tag{1.2}$$

such that the transformed signal $\boldsymbol{w}$ is *sparse*, i.e. it has only a small number of non-zero entries This is a realistic assumption for a wide range of common signal types, such as digital images, audio, video and many kinds communication signals.

Letting $\boldsymbol{\Phi} = \boldsymbol{\Theta}\boldsymbol{\Psi}$, the system (1.1) can then be expressed as

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{w}. \tag{1.3}$$

A variety of deterministic CS algorithms have been developed that attempt to find sparse solutions to (1.3). For our implementation, however, we choose to approach the problem from the point of view of *machine learning*.

In general, the aim of supervised machine learning is to learn a relationship $t = f(\boldsymbol{x})$ between an input vector $\boldsymbol{x}$ and a target t. If the target is a discrete variable the problem is known as *classification*, whereas if t is real-valued we refer to the problem as *regression*.

To apply regression techniques to the CS problem (1.3), we regard the CS measurements $y^{(i)}$ as targets and treat the corresponding rows $\boldsymbol{\varphi}^{(i)}$ of the matrix $\boldsymbol{\Phi}$ is input vectors. In this context, $\boldsymbol{y}$ is known as a *target vector* and $\boldsymbol{\Phi}$ as the corresponding *design matrix*. A linear regression algorithm takes these quantities as its input and "learns" a set of *weights* $\boldsymbol{w}$. Given $\boldsymbol{w}$, the original signal can be computed using equation (1.2).

For our code, we implemented the *Relevance Vector Machine* (RVM) [14, 15] as it tends to give very sparse solutions $\boldsymbol{w}$. This algorithm was used by [9] to successfully reconstruct highly undersampled image signals. Comparing it to a range of classic deterministic CS reconstruction algorithms, [9] found that the RVM gave superior reconstruction performance.

In practical CS systems, the sensing mechanism $\boldsymbol{\Theta}$ is typically implemented within the actual sensing hardware. To provide a range of reconstruction scenarios, our code simulates three types of sensing matrices: random Gaussian, random Bernoulli and random signal masks.

Of particular interest is the case when $\boldsymbol{\Theta}$ corresponds to a signal mask, so that $\boldsymbol{y}$ is a highly undersampled version of the signal $\boldsymbol{v}$. The problem of solving (1.1) is then equivalent to the problem *signal interpolation*. For this particular class of sensing mechanisms, [9] developed an extension to the RVM called *Multi-Scale Cascade of Estimations* (MSCE). The MSCE algorithm utilizes the *multiresolution properties of Haar wavelets* to form a cascade of RVMs. [9] investigated its performance in image interpolation and found that it can provide a significant boost in the reconstruction

quality over a single RVM. For our implementation, we extended the MSCE to the case of video interpolation.

Thesis Organization

In this thesis, we discuss the underlying theory and implementation details for all the building blocks of our Compressive Sensing implementation.

We begin in Chapter 2 with a discussion of the relevant background in digital signal processing. A brief overview of the conventional approaches to signal acquisition and compression is provided and then contrasted with the novel Compressive Sensing framework.

In order to ensure a high quality reconstruction, it is important to find sparse representations of the signal. Chapter 3 describes the Discrete Cosine Transform and the Discrete Wavelet Transform. These two basis transforms are often used for their sparsifying properties, especially when applied to digital image and video signals.

In Chapter 4, we derive the Sparse Bayesian Learning framework and discuss two different algorithms for training the RVM.

Chapter 5 combines Compressive Sensing with Sparse Bayesian Learning to form the Bayesian Compressive Sensing framework. We show how CS can be viewed from a Bayesian perspective and discuss the MSCE algorithm.

In Chapter 6 we provide further implementation details, including a discussion on blockwise signal reconstruction and a parallel implementation.

Finally, in Chapter 7 we demonstrate the performance of our algorithm and show that it provides near-perfect reconstructions of video signals with $N = 0.3M$.

We conclude in Chapter 8.

Chapter 2

Background

In this chapter, we will introduce the theory of *Compressive Sensing* (also known as *Compressed Sensing, Compressive Sampling* or simply, *CS*). CS is a framework within signal processing that allows for acquiring signals (i.e. measure or *sense*) directly in a *compressed* format.

To motivate the discussion, we will first review the conventional approach to signal acquisition and compression.

2.1 Coventional Signal Processing

2.1.1 Signal Acquisition

In order to work with information within analog signals (continuous streams of data) such as sounds, images or video, we rely on reducing the analog signals to digital (discrete) signals that can be processed with computers. This digitization is done by taking discrete measurements of the analog signal at certain points in time or space, a process known as *sampling*.

Conventional approaches to sampling are based on the *Shannon/Nyquist Sampling Theorem* [11]: When sampling a signal uniformly, we are able to *perfectly reconstruct* the signal from its samples if the sampling rate is at least twice the highest frequency present in the signal.

Consider an analog signal $x(t)$ that varies with time, such as an audio wave. Let f be the highest frequency present in $x(t)$.

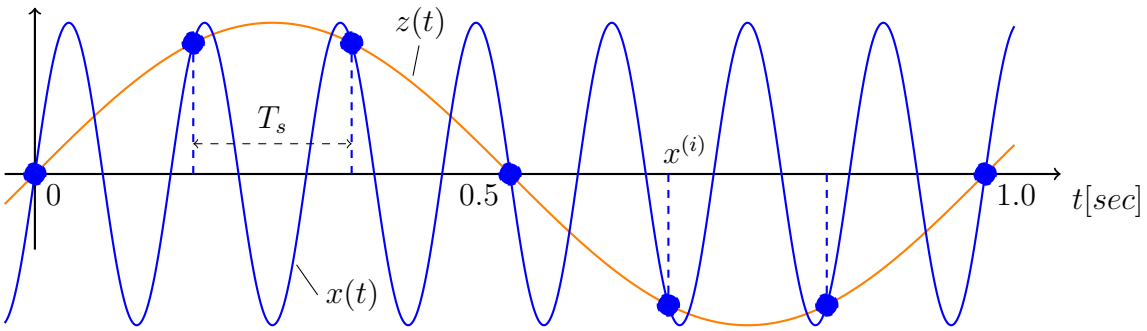


Fig. 2.1 Illustration of the Shannon/Nyquist Sampling Theorem. The blue curve is the original signal $x(t)$ which is a sinusoid with frequency $f = 7$ Hz. The blue points are discrete samples $x^{(i)}$ taken from $x(t)$ at a sampling rate $f_s = 7$ Hz, which is below the Nyquist Rate $2f = 14$ Hz. Thus, aliasing occurs and interpolation algorithms will reconstruct an alias $z(t)$ (orange curve) of $x(t)$.

In order to digitise $x(t)$, we measure x at discrete points in time $t^{(0)}, \cdots, t^{(n)}$ and store the samples $x^{(i)} \equiv x(t^{(i)})$. We sample x uniformly, measuring a sample every T_s seconds, so that $t^{(i)} = iT_s$. The sampling rate is therefore $f_s = 1/T_s$.

Suppose we wish to reconstruct $x(t)$ by interpolating the samples. There is an infinite number of continuous functions that fit this set of samples. However, it can be shown that only one of them has a bandwidth of no more than $f_s/2$. Thus, if $f < f_s/2$ (the *Nyquist Criterion*), then $x(t)$ is the unique function that will be approximated by interpolation algorithms such as the *Whittaker-Shannon interpolation formula* [11].

In Figure 2.1, we have a sinusoidal signal $x(t)$ with frequency f. The sampling rate is $f_s = f$ and therefore below the signal's *Nyquist rate* $2f$. Thus, we are unable to reconstruct $x(t)$ from the samples. Instead, we will reconstruct an *alias* $z(t)$ which, in this case, is another sinusoid with frequency $f/7$. The original signal x is lost.

We have illustrated Nyquist sampling in the 1-dimensional case. The same principles hold for higher dimensional signals such as images and videos.

For signals that vary with space, the sampling rate is governed by the desired spatial resolution. In order to recover the finer details (the high-frequency components) of an image, we require a higher pixel density (i.e. a larger number of pixels per centimeter (ppcm)).

Nyquist sampling underlies almost all signal acquisition protocols that are found in practice. It is the basis of medical imaging, audio and video recording and radio receivers.

2.1.2 Signal Compression

The sampling theorem imposes a lower bound on the sampling rate above which we are able to perfectly reconstruct the desired signal. This lower bound is often very high and we end up with a very large number of measurements. Storage and transfer of such signals becomes prohibitively expensive as the size of the signal grows. Thus, a need for *data compression* arises.

We will discuss a particular type compression algorithm known as *transform coding*. It is the standard compression method for a wide range of manmade signals such as audio, photos, and video and is the basis of many common signal formats such as JPEG (images), MPEG (video) and MP3 (audio).

Let $\boldsymbol{v}$ by a real-valued digital signal of length M, $\boldsymbol{v} \in \mathbb{R}^M$. Without loss of generality, $\boldsymbol{v}$ is assumed to be a one-dimensional signal. If we are working with a multi-dimensional signal, we may first vectorize it into a long vector. When compressing digital signals, we are usually interested in *lossy compression*.

Any vector in $\mathbb{R}^M$ can be expressed as a linear combination of M *basis vectors* $\boldsymbol{\psi}_j \in \mathbb{R}^M$:

$$\boldsymbol{v} = \sum_{j=1}^{M} w_j \boldsymbol{\psi}_j \tag{2.1}$$

where w_j is the coefficient (or weight) associated with $\boldsymbol{\psi}_j$.

By forming the *basis matrix* $\boldsymbol{\Psi} = [\boldsymbol{\psi}_1 \cdots \boldsymbol{\psi}_M]$, we can express equation (2.1) in matrix form

$$\boldsymbol{v} = \boldsymbol{\Psi}\boldsymbol{w}$$

where $\boldsymbol{w} = (w_1, \cdots, w_M)^T$. For simplicity, we assume that the basis $\boldsymbol{\Psi}$ is orthonormal, so that $\boldsymbol{\Psi}\boldsymbol{\Psi}^T = \boldsymbol{I}_M$ and $\boldsymbol{\psi}_i^T \boldsymbol{\psi}_j$ is 1 if $i = j$ and 0 otherwise. Thus, the coefficient w_j is given by $w_j = \boldsymbol{v}^T \boldsymbol{\psi}_j$.

We now have two equivalent representations of the same signal: $\boldsymbol{v}$ in the original basis and $\boldsymbol{w}$ in the $\boldsymbol{\Psi}$ basis. Since $\boldsymbol{\Psi}$ is orthogonal, $\boldsymbol{v}$ and $\boldsymbol{w}$ have the same ℓ_2-norm, $||\boldsymbol{v}||_2 = ||\boldsymbol{\Psi}\boldsymbol{w}||_2 = ||\boldsymbol{w}||_2$. However, in the original signal, $\boldsymbol{v}$, the energy is typically spread over many of its components. On the other hand, it is possible to find a basis $\boldsymbol{\Psi}$ such that the energy of the transformed signal, $\boldsymbol{w}$, is concentrated in only a few large components w_j and a large fraction of its entries are very close to zero.

Suppose that we delete the entries w_j that are very small and replace them with zero to obtain $\hat{\boldsymbol{w}}$. Let $\hat{\boldsymbol{v}} = \boldsymbol{\Psi}\hat{\boldsymbol{w}}$ be the approximate signal in the original domain. Since

(a) Uncompressed Image (b) Compressed Image via DCT

Fig. 2.2 The uncompressed image has a resolution of 512×512, i.e. 262,144 pixels. We compress the image by performing a Discrete Cosine Transform (see Section 3.1) and storing only the largest 27,832 coefficients. The compression ratio is 9.42.

$\hat{\boldsymbol{w}}$ is very close to $\boldsymbol{w}$, it follows that

$$||\hat{\boldsymbol{v}} - \boldsymbol{v}||_2 = ||\boldsymbol{\Psi}\hat{\boldsymbol{w}} - \boldsymbol{\Psi}\boldsymbol{w}||_2 = ||\boldsymbol{\Psi}(\hat{\boldsymbol{w}} - \boldsymbol{w})||_2 = ||\hat{\boldsymbol{w}} - \boldsymbol{w}||_2$$

is very small.

Thus, a viable method for lossy compression of the signal $\boldsymbol{v} \in \mathbb{R}^M$ would be the following:

1. Compute the full set of transform coefficients $\{w_j\}_{j=1}^M$ via $\boldsymbol{w} = \boldsymbol{\Psi}^T\boldsymbol{v}$.
2. Locate all the coefficients w_j whose absolute value is above a certain threshold (suppose there are K of them).
3. Discard all the $(M - K)$ small coefficients
4. Store the values and locations of the K large coefficients

In order to view the compressed signal in the original domain, we reconstruct it via the transform: $\boldsymbol{\Psi}\hat{\boldsymbol{w}} = \hat{\boldsymbol{v}}$, where $\hat{\boldsymbol{w}}$ is $\boldsymbol{w}$ with the $(M - K)$ smallest coefficients replaced by zero.

It is possible to find basis matrices $\boldsymbol{\Psi}$ that result in very high compression ratios for a wide range of signals without any noticable reduction in the signal quality.

Furthermore, many of the commonly used basis transforms can be computed very efficiently.

Audio signals and a wide class of communication signals are highly compressible in the localized Fourier basis. Images and video signals, on the other hand, can often be compressed via the *Discrete Cosine Transform* (DCT) or the *Discrete Wavelet Transform* (DWT). For instance, the JPEG standard for image compression is based on the DCT [17], while the more modern JPEG2000 format uses the CDF 9/7 wavelet transform or the CDF 5/3 wavelet transform [16].

In Figure 2.2, we compress the standard test image "Lenna" via a DCT. We are only storing about 10% of the transform coefficients. Yet, the difference between the original image and the compressed image is hardly noticable.

We will discuss the DCT and the DWT in more detail in Chapter 3.

2.2 Compressive Sensing

The conventional approach to data acquisition and compression is very effective and has been highly influential. However, it is also extremely wasteful. We acquire a huge amount of data at the signal sampling stage and then proceed to discard a large part of it at the compression stage.

Compressive Sensing [2, 5] is a more general approach that lets us *acquire signals directly in a compressed format*. This is clearly more efficient as it allows us to skip the intermediate stage of taking N samples.

In this section we will formulate the Compressive Sensing problem. For simplicity, the focus will be on discrete signals such as digital images or videos.

2.2.1 The Compressive Sensing Problem

Let $\boldsymbol{v} \in \mathbb{R}^M$ be the signal of interest. As an example, $\boldsymbol{v}$ could be a digital photograph, such as Figure 2.2a, that has been unrolled into a long vector of length M, where M is the number of pixels.

Suppose $\boldsymbol{v}$ is currently unknown and we want to acquire a compressed representation of it without measuring $\boldsymbol{v}$ directly. To do so, consider the following linear sensing scheme: We measure inner products between the signal $\boldsymbol{v}$ and a collection of M-dimensional vectors $\{\boldsymbol{\theta}_i\}_{i=1}^N$ to obtain the measurements $y_j = \boldsymbol{v}^T\boldsymbol{\theta}_i$ for $i = 1, \cdots, N$.

This relation can be expressed more succinctly as

$$\boldsymbol{y} = \boldsymbol{\Theta}\boldsymbol{v} \tag{2.2}$$

where $\boldsymbol{y} = (y_1, \cdots, y_N)^T$ and $\boldsymbol{\Theta}$ is the $N \times M$ *sensing matrix* given by

$$\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\theta}_1^T \\ \vdots \\ \boldsymbol{\theta}_N^T \end{bmatrix}. \tag{2.3}$$

We are interested in the *undersampled* situation where $N < M$. In particular, we would like to acquire a compressed representation $\boldsymbol{y}$ of $\boldsymbol{v}$ using as few measurements as possible, while still being able to recover the original signal.

There are two problems that stand out at this point.

1. Given that $N << M$, how can we ensure that, during the measurement process, we do not lose any information contained in $\boldsymbol{v}$ (i.e. $\boldsymbol{y}$ captures all the information in $\boldsymbol{v}$)?

2. Given our measurements $\boldsymbol{y}$ and knowledge of the sensing matrix $\boldsymbol{\Theta}$, how do we recover the signal of interest $\boldsymbol{v}$?

At an intuitive level, we would expect at least some information to be destroyed during the measurement process, especially if the number of measurements, N, is substantially lower than the length of the desired signal, M. Moreover, even if $\boldsymbol{y}$ did contain the same information as $\boldsymbol{v}$, we are still faced with solving the linear system in equation (2.2). This system is underdetermined and hence there are infinitely many vectors $\boldsymbol{v}$ that satisfy $\boldsymbol{\Theta}\boldsymbol{v} = \boldsymbol{y}$.

2.2.2 Sparse Signals

In general, we cannot resolve these issues. However, if we restrict ourselves to a certain class of signals, it is possible to make progress.

In particular, we will focus on signals that have *sparse representations*. A signal $\boldsymbol{v} \in \mathbb{R}^M$ has a sparse representation if there exists a $M \times M$ basis matrix $\boldsymbol{\Psi}$ so that the transformed signal $\boldsymbol{w}$, where $\boldsymbol{v} = \boldsymbol{\Psi}\boldsymbol{w}$, is *sparse*.

We say that $\boldsymbol{w}$ is K-*sparse* if $\boldsymbol{w}$ has K non-zero components. Equivalently, $\boldsymbol{w}$ is K-sparse if [1] $||\boldsymbol{w}||_0 = K$.

In Section 2.1.2 we noted that many manmade signals are compressible. They have an almost-sparse representation, meaning that, when expressed in the basis $\boldsymbol{\Psi}$, almost all of their energy is contained in only K components and the remaining components are very close to zero. Such signals are well-approximated by K-sparse representations.

2.3 Solving the Compressive Sensing Problem

So let us suppose then, that the desired signal $\boldsymbol{v}$ is K-sparse when expressed in the $\boldsymbol{\Psi}$ basis, where K is small ($K << M$). We can substitute $\boldsymbol{v} = \boldsymbol{\Psi w}$ into (2.2) to get

$$\boldsymbol{y} = \boldsymbol{\Theta v} = \boldsymbol{\Theta \Psi w}$$

Let us define $\boldsymbol{\Phi} = \boldsymbol{\Theta \Psi}$, so that

$$\boldsymbol{y} = \boldsymbol{\Phi w}. \tag{2.4}$$

We have arrived at another underdetermined linear system. The CS measurements $\boldsymbol{y}$ are still the same as in (2.2), but the sensing matrix $\boldsymbol{\Theta}$ has been replaced by the new sensing matrix $\boldsymbol{\Phi}$.

However, we now want to recover the signal $\boldsymbol{w}$ and we can use the fact that $\boldsymbol{w}$ is K-sparse. This allows us to address the problems above.

The information in $\boldsymbol{w}$ is highly localized. It is fully contained in only $K << M$ of its entries. Thus, as long as $N \geq K$, it should, at least in principle, be possible for a measurement vector $\boldsymbol{y}$ of length $N << M$ to completely capture the information within $\boldsymbol{w}$.

[1] The ℓ_p-norm of a vector $\boldsymbol{z} \in \mathbb{R}^n$ is defined as follows for $p > 0$:

$$||\boldsymbol{z}||_p \equiv \left(\sum_{i=1}^{n} |z_i|^p \right)^{\frac{1}{p}}.$$

If $p = 0$ we can define the ℓ_0 "norm" of $\boldsymbol{z}$ to be the number of its non-zero entries:

$$||\boldsymbol{z}||_0 \equiv \sum_{i=1}^{n} \mathbb{1}\{z_j \neq 0\}$$

Furthermore, since $(M - K)$ entries of $\boldsymbol{w}$ are zero, it follows that in (2.4), $\boldsymbol{y}$ is actually a linear combination of only K of the columns of $\boldsymbol{\Phi}$. So, if $N \geq K$, we might expect to find suitable constraints that allow us to recover $\boldsymbol{w}$. Once $\boldsymbol{w}$ is recovered, we can compute $\boldsymbol{v}$ via $\boldsymbol{v} = \boldsymbol{\Psi w}$.

2.3.1 Constructing the Sensing Mechanism

In practice, we do not know the locations of the K nonzero entries in $\boldsymbol{w}$. So the first challenge in a compressive sensing system is to design a $N \times M$ sensing matrix $\boldsymbol{\Theta}$ which ensures that, for any signal $\boldsymbol{v}$ that has a K-sparse representation in the basis $\boldsymbol{\Psi}$, $\boldsymbol{y} = \boldsymbol{\Theta v}$ contains all the essential information within the signal $\boldsymbol{v}$.

We will not go into the theoretical details of designing an optimal CS measurement process. Instead we will refer the reader to [3] and simply state three particular sensing matrices that have been used successfully in practice:

Gaussian: Form $\boldsymbol{\Theta}$ by sampling its entries θ_{ij} independently from the Gaussian distribution with mean zero and variance $1/M$: $\theta_{ij} \overset{iid}{\sim} \mathcal{N}(0, 1/M)$.

Bernoulli: Sample the entries θ_{ij} independently from a symmetric Bernoulli distribution, $P(\theta_{ij} = \pm 1/\sqrt{M}) = 1/2$.

Signal Mask: Form $\boldsymbol{\Theta}$ by starting with the $M \times M$ identity matrix I_M and deleting $M - N$ of its rows at random. This sensing matrix corresponds to measuring a down-sampled version of the signal $\boldsymbol{v}$ directly. Throughout the thesis, will refer to such sensing matrices as "signal masks".

2.3.2 Signal Recovery

Using the sensing mechanisms outlined above, we are able to measure the signal $\boldsymbol{v}$ directly in a compressed format $\boldsymbol{y}$. The final part of a Compressive Sensing system is a signal reconstruction algorithm that decompresses $\boldsymbol{y}$ and recovers $\boldsymbol{v}$ or, equivalently, its sparse representation $\boldsymbol{w}$.

One way of achieving this is by searching for the sparsest solution $\boldsymbol{w}$ that satisfies equation (2.4) [1]. That is, the desired signal is the solution to the optimization problem:

$$\min_{\boldsymbol{w}} ||\boldsymbol{w}||_0 \qquad \text{subject to} \qquad \boldsymbol{\Phi w} = \boldsymbol{y} \tag{2.5}$$

Unfortunately, (2.5) is an NP-complete problem that can only be solved by an exhaustive search through all possible combinations for the locations of the non-zero entries in $\boldsymbol{w}$.

Luckily, and somewhat surprisingly, it can be shown that, if $N = O(K \log(M/K))$, it is possible to reconstruct K-sparse signals exactly by solving the ℓ_1-optimization problem [1, 3]

$$\min_{\boldsymbol{w}} ||\boldsymbol{w}||_1 \qquad \text{subject to} \qquad \boldsymbol{\Phi w} = \boldsymbol{y} \tag{2.6}$$

In practice, signals $\boldsymbol{v}$ usually only have an almost-sparse representation. Thus, the reconstruction is not completely exact and the solution $\hat{\boldsymbol{v}}$ will be a close approximation to $\boldsymbol{v}$.

A large part of the CS literature is focused on developing algorithms that recover a signal $\boldsymbol{v}$ from a set CS measurement $\boldsymbol{y}$, either by solving (2.6) or through some alternative route.

For a discussion and comparison of some of the most widely used algorithms, see [9].

In Chapter 5 we will approach the CS signal reconstruction problem from a Bayesian perspective. Before we do so, however, we need to introduce the Sparse Bayesian Learning framework (Chapter 4) and provide a more thorough discussion about basis functions (Chapter 3).

Chapter 3

Basis Functions

In Section 2.1.2, we introduced transform coding. We said that any discrete signal $\boldsymbol{v} \in \mathbb{R}^M$ can be expressed in a different basis via a basis transform:

$$\boldsymbol{v} = \boldsymbol{\Psi}\boldsymbol{w}$$

where $\boldsymbol{\Psi}$ is the $M \times M$ basis matrix and $\boldsymbol{w} \in \mathbb{R}^M$ is the representation of $\boldsymbol{v}$ in the $\boldsymbol{\Psi}$ basis.

The particular classes of signals $\boldsymbol{v}$ that we are interested in are digital images and digital video. The aim of this chapter is to construct a basis matrix $\boldsymbol{\Psi}$ that gives us a sparse representation of a wide range of such signals $\boldsymbol{v}$. Finding a set of basis functions $\boldsymbol{\Psi}$ that achieve such a transformation lies at the heart of many lossy compression techniques.

3.1 Discrete Cosine Transform

The first basis transform that we will use is the Discrete Cosine Transform (DCT), one of the most widely used transforms in signal processing. It underlies JPEG image compression and is used in various video compression algorithms such as MJPEG, MPEG, H.261 and H.263 [18]. [1]

A DCT decomposes a signal in terms of cosine functions with different frequencies. Its extensive use in lossy compression algorithms is due to the DCT's *energy compaction*

[1] A related transform, known as the *Modified DCT* is used in many lossy audio compression formats such as MP3, AAC and Vorbis.

properties. The majority of a transformed signal's energy is contained within relatively few coefficients - typically those corresponding to the lower frequency basis functions.

The DCT comes in a various versions that have minor differences between them. In the following, we will describe the most widely used version, known as the *DCT-II*, as well as its inverse transform, the *DCT-III*. We will refer to them simply as "the DCT" and "the Inverse DCT" (IDCT), respectively.

3.1.1 One-Dimensional DCT

Formally, the DCT $\boldsymbol{w}$ of a one-dimensional signal $\boldsymbol{v}$ of length M is given by

$$w_k = c_k \sum_{n=1}^{M} v_n \cos\left(\frac{\pi(2n-1)(k-1)}{2M}\right) \qquad k = 1, \cdots, M \tag{3.1}$$

where

$$c_k = \begin{cases} \sqrt{\frac{1}{M}} & \text{if } k = 1 \\ \sqrt{\frac{2}{M}} & \text{otherwise} \end{cases}$$

This transforms a signal $\boldsymbol{v}$ in the original domain (time or space) into its representation $\boldsymbol{w}$ in the DCT domain.

Conversely, given a signal $\boldsymbol{w}$ in the DCT domain, we can transform it back to the orignal (time or space) domain via the IDCT defined by

$$v_n = \sum_{k=1}^{M} c_k w_k \cos\left(\frac{\pi(2n-1)(k-1)}{2M}\right) \qquad n = 1, \cdots, M \tag{3.2}$$

We can express equation (3.2) in the desired form $\boldsymbol{v} = \boldsymbol{P}\boldsymbol{w}$. The entries of the basis matrix $\boldsymbol{P}$ are given by

$$P_{n,k} = c_k \cos\left(\frac{\pi(2n-1)(k-1)}{2M}\right). \tag{3.3}$$

It can be verified that the basis matrix $\boldsymbol{P}$ is orthogonal, $\boldsymbol{P}^T\boldsymbol{P} = \boldsymbol{I}_M$.

3.1.2 Multi-Dimensional DCT

Once we know how to perform the DCT on a one-dimensional signal, we can easily extend the transform to multi-dimensional signals (images, video, etc). To do so, we

(a) Original signal $\boldsymbol{v}$ (b) DCT of $\boldsymbol{v}$

Fig. 3.1 Panel (a) shows the original signal $\boldsymbol{v}$, a 256×256 grayscale image known as "cameraman". Panel (b) illustrates the 2-D DCT of $\boldsymbol{v}$. The brightness of a an element increases with the absolute value of the corresponding DCT coefficient. (The high-frequency coefficients have been enhanced to show more detail).

simply perform successive 1-D transforms along each dimension of the signal. This is the *seperability* of the DCT.

Suppose the signal of interest is a digital image. That means that $\boldsymbol{v}$ is a $M_1 \times M_2$ matrix where $M_1 \times M_2$ is the resolution of the image. To transform the signal, we first perform the DCT on every row of the matrix. Following that, we perform the DCT on every column of the resulting matrix to get the final transformed signal.

Figure 3.1 shows an example of a 2-D signal $\boldsymbol{v}$ and its transform $\boldsymbol{w}$. Note that the majority of the energy of the transformed signal is concentrated in the top left corner. Most of the DCT coefficients are zero or very close to zero.

In Figure 3.2, we show the 2-D basis functions that would used by the DCT to decompose a signal of size 4×4. Each basis function is characterised by a horizontal and vertical spatial frequency. Typically, natural images are mostly made up of low-frequency components and the corresponding coefficients are therefore relatively large. The highest-frequency components are usually only needed to describe very fine details.

To compute the DCT of a video, we first compute the 2-D DCT for each individual frame of the video followed by a 1-D DCT across the temporal axis for each pixel. The DCT basis functions in 3-D are characterised by a horizontal and vertical spatial frequency, as well as an additional temporal frequency component.

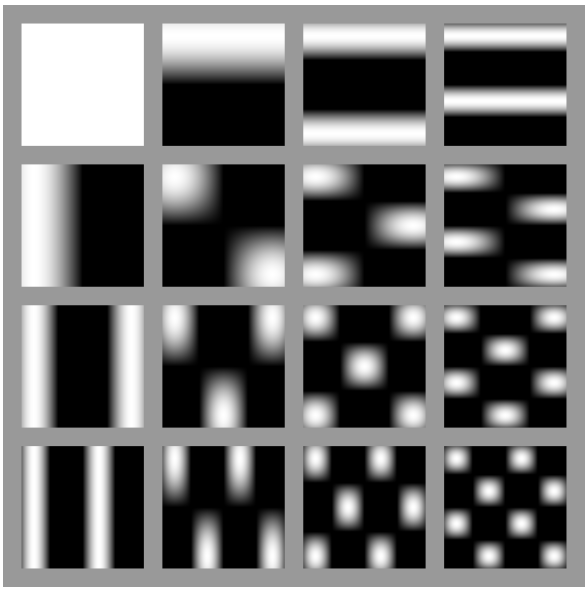


Fig. 3.2 The 2-D DCT basis functions that are used by the DCT to decompose a 4×4 image. The spatial frequency increases towards the bottom right corner.

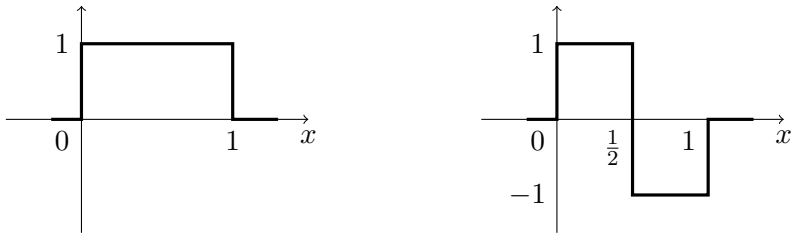
For a discussion on the properties of the DCT, see [7]

3.2 Discrete Wavelet Transform

Wavelets have become a very popular tool in signal processing. Their energy compaction properties are on par and often superior to those of the DCT for a wide range of signal classes. Moreover, the multiresolution properties of wavelets allow us to analyze signals at different scales.

In 2000, the JPEG committee released a new image coding standard, JPEG2000, that is gradually replacing the original JPEG standard. The new format moved away from the DCT and uses a Discrete Wavelet Transform (DWT) instead [16].

The aim of the following sections is to provide some intuition on the general properties of wavelets. We will discuss how a wavelet basis can be constructed at multiple resolution levels and how the DWT can be computed in practice. We will refrain from going to deep into mathematical detail and instead direct the interested reader to [8, 4].



(a) Haar scaling function $\phi(x)$ (b) Haar mother wavelet $\psi(x)$

Fig. 3.3 The scaling function and wavelet function for the Haar wavelets.

3.2.1 Introduction to Wavelets

To motivate wavelets, consider again the one-dimensional signal $\boldsymbol{v}$ of length M. Suppose, for simplicity, that M is a power of 2, $M = 2^q$ say. We can view $\boldsymbol{v}$ as a piecewise-constant function $v(x)$ on the half-open interval $[0, 1)$, where $v(x) = v_i$ if $x \in [\frac{i-1}{M}, \frac{i}{M})$.

Let V^j denote the vector space containing all piecewise-constant functions f defined on the interval $[0, 1)$ that consist of 2^j pieces, each of which is constant across a sub-interval of size 2^{-j}. Thus, V^0 consists of all functions that are constant on $[0, 1)$, while V^1 consists of all functions that have two constant pieces, one over $[0, 1/2)$ and one over $[1/2, 1)$. In particular, our signal $v(x)$ resides in the space V^q.

Note that if $f \in V^j$, then $f \in V^{j+1}$. Thus, the vector spaces V^j are nested: $V^0 \subset V^1 \subset V^2 \subset \cdots$.

We need to choose a basis for each vector space V^j. To do so, we introduce a *scaling function* (also known as *scalet*, or *father wavelet*) that is usually denoted $\phi(x)$. The form of the scaling function depends on the particular choice of wavelet decomposition.

For example, for the *Haar wavelets*, the scaling function is given by

$$\phi(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

See Figure 3.3a for a plot of $\phi(x)$.

Given the scaling function $\phi(x)$, we can define the following basis for V^j:

$$\phi_k^j(x) := 2^{j/2}\phi(2^j x - k) \qquad k = 0, \cdots, 2^j - 1 \tag{3.5}$$

Using this basis, we can decompose our signal $v(x) \in V^q$ as

$$v(x) = \sum_{k=0}^{2^q-1} c_k^q \phi_k^q(x)$$

For the particular scaling function defined in equation (3.4), we can see that $c_k^q = v_{k+1}$.

To obtain *wavelets*, consider the *orthogonal complement* of V^j in V^{j+1} and denote it W^j. That is, $W^j = \{f \in V^{j+1} : \langle f, g\rangle = 0 \ \forall g \in V^j\}$ where the inner product $\langle f, g\rangle$ is defined by

$$\langle f, g\rangle = \int_0^1 f(x)g(x)dx.$$

By forming a basis for W^j, we obtain a set of *wavelet functions* $\{\psi_k^j,\ k = 0, \cdots, 2^j - 1\}$. Wavelet functions can be constructed by scaling and shifting a so-called *mother wavelet* $\psi(x)$ as follows:

$$\psi_k^j(x) = 2^{j/2}\psi(2^j x - k) \qquad k = 0, \cdots, 2^j - 1 \tag{3.6}$$

For the Haar wavelets, the mother wavelet is given by:

$$\psi(x) = \begin{cases} 1 & 0 \le x < 1/2 \\ -1 & 1/2 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

The Haar mother wavelet is shown in Figure 3.3b.

Note that, since the scaling functions ϕ_k^j form a basis of V^j and the wavelet functions ψ_k^j form a basis of W^j, and since W^j is the orthogonal complement to V^j in V^{j+1}, it follows that the set $\{\phi_k^j, \psi_k^j : k = 0, \cdots, 2^j - 1\}$ forms a basis of the vector space V^{j+1}.

This allows us to express our signal $v \in V^q$ as

$$v(x) = \sum_{k=0}^{2^{q-1}-1} d_k^{q-1}\psi_k^{q-1}(x) + \sum_{k=0}^{2^{q-1}-1} c_k^{q-1}\phi_k^{q-1}(x)$$

This gives us the first level of the discrete wavelet transform of v. The coefficients c_k and d_k are sometimes referred to as "approximation" coefficients and "detail" coefficients, respectively.

We can continue the decomposition by splitting the basis for V^{q-1} into the bases for V^{q-2} and W^{q-2} to get the next level of the transform:

$$v(x) = \sum_{k=0}^{2^{q-1}-1} d_k^{q-1}\psi_k^{q-1}(x) + \sum_{k=0}^{2^{q-2}-1} d_k^{q-2}\psi_k^{q-2}(x) + \sum_{k=0}^{2^{q-2}-1} c_k^{q-2}\phi_k^{q-2}(x)$$

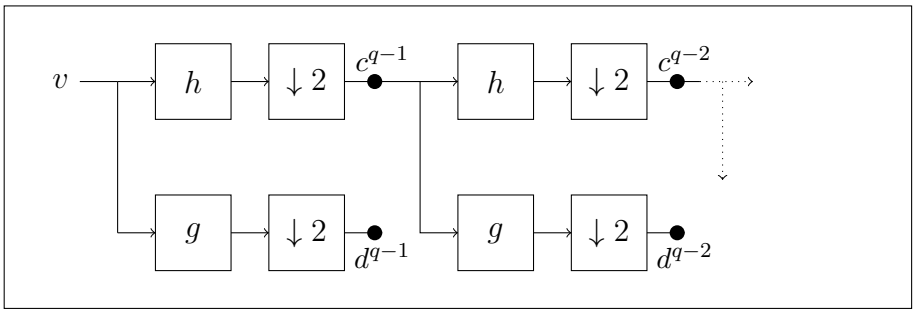


Fig. 3.4 The first two levels of the DWT of the signal $\boldsymbol{v}$ of length 2^q via a filter bank.

To get the full decomposition, we continue in this fashion up to the qth level:

$$v(x) = \sum_{j=0}^{q-1} \sum_{k=0}^{2^j-1} d_k^j \psi_k^j(x) + c_0^0 \phi(x)$$

The full Discrete Wavelet Transform of $\boldsymbol{v}$ consists of the coefficients

$$\{c_0^0, d_k^j : \, j = 0, \cdots, q-1, \, k = 0, \cdots, 2^j - 1\}.$$

3.2.2 Computing the DWT

In practice, we can compute one level of the DWT coefficients by passing the signal v through a *low-pass filter* h and a high-pass filter g, respectively, and then downsampling the results by a factor of two. Passing v through the filters h and g results in the *convolutions*:

$$(v \star h)(x) := \sum_{k=-\infty}^{\infty} v(k) h(x-k)$$
$$(v \star g)(x) := \sum_{k=-\infty}^{\infty} v(k) g(x-k)$$

To downsample by a factor of two, we remove every second sample:

$$(v \downarrow 2)(x) := v(2x)$$

Overall, these computations can be performed by multiplying the vector $\boldsymbol{v}$ by a matrix $\boldsymbol{H}$ and a matrix $\boldsymbol{G}$ to get the approximation and detail coefficients, respectively.

To compute the next stage, we take the approximation coefficients of the current stage and pass them again through the filter bank. The procedure is depicted in Figure 3.4.

The coefficients of the filters $h(x)$ and $g(x)$ depend on our choice of the scaling function $\phi(x)$ and mother wavelet function $\psi(x)$ and must satisfy the relations

$$\phi(x) = \sqrt{2} \sum_{k=-\infty}^{\infty} h(k)\phi(2x-k)$$
$$\psi(x) = \sqrt{2} \sum_{k=-\infty}^{\infty} g(k)\phi(2x-k)$$

as well as

$$g(k) = (-1)^k h(1-k).$$

For the Haar wavelets (3.4,3.7), the filter coefficients are

$$h(0) = h(1) = \frac{1}{\sqrt{2}}, \qquad h(k) = 0 \text{ if } k \neq 0, 1$$
$$g(0) = -g(1) = \frac{1}{\sqrt{2}}, \qquad g(k) = 0 \text{ if } k \neq 0, 1$$

The approximation coefficients for the first stage of the Haar DWT of signal $\boldsymbol{v} \in \mathbb{R}^M$, with $M = 2^q$, are thus given by $\boldsymbol{H}^q\boldsymbol{v}$ where $\boldsymbol{H}^q$ is the $2^{q-1} \times 2^q$ matrix defined by

$$\boldsymbol{H}^q_{haar} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{bmatrix} \tag{3.8}$$

To get the corresponding detail coefficients, we multiply $\boldsymbol{v}$ by the $2^{q-1} \times 2^q$ matrix $\boldsymbol{G}^q$ defined by

$$\boldsymbol{G}^q_{haar} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & -1 \end{bmatrix} \tag{3.9}$$

Overall, to perform the discrete Haar wavelet transform at the first scale, we multiply $\boldsymbol{v}$ by

$$\boldsymbol{P}_{haar}^{(1)} = \begin{bmatrix} \boldsymbol{H}^q \\ \boldsymbol{G}^q \end{bmatrix}$$

where we dropped the *haar*-subscript on $\boldsymbol{H}$ and $\boldsymbol{G}$ for ease of viewing.

To compute the transform at the second scale, we replace $\boldsymbol{H}^q$ by $\boldsymbol{H}^{q-1}\boldsymbol{H}^q$ and $\boldsymbol{G}^{q-1}\boldsymbol{h}^q$:

$$\boldsymbol{P}_{haar}^{(2)} = \begin{bmatrix} \boldsymbol{H}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^q \end{bmatrix}$$

We can continue this process until the qth scale to get the full Haar wavelet transform matrix:

$$\boldsymbol{P}_{haar}^{(q)} = \begin{bmatrix} \boldsymbol{H}^1\boldsymbol{H}^2\cdots\boldsymbol{H}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^1\boldsymbol{H}^2\cdots\boldsymbol{H}^{q-1}\boldsymbol{H}^q \\ \vdots \\ \boldsymbol{G}^{q-2}\boldsymbol{H}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^{q-1}\boldsymbol{H}^q \\ \boldsymbol{G}^q \end{bmatrix}$$

Note that the size of the transform matrix $\boldsymbol{P}_{haar}^{j}$ is $M \times M$ and is not affected by the scale j.

To form the basis matrix $\boldsymbol{\Psi}$ such that $\boldsymbol{v} = \boldsymbol{\Psi}\boldsymbol{w}$ corresponding to the DWT at scale j, we can invert the matrix $\boldsymbol{P}_{haar}^{(j)}$:

$$\boldsymbol{\Psi} = \left(\boldsymbol{P}_{haar}^{(j)}\right)^{-1} \tag{3.10}$$

In practice, it is desirable to work with orthonormal wavelet bases because they lead to orthogonal transform matrices, so that $\left(\boldsymbol{P}^{(j)}\right)^{-1} = \left(\boldsymbol{P}^{(j)}\right)^{T}$. This orthoganility property is provided by Haar wavelets. Therefore, the basis matrix $\boldsymbol{\Psi}$ from equation (3.10) can also be obtained as follows:

$$\boldsymbol{\Psi} = \left(\boldsymbol{P}_{haar}^{(j)}\right)^{T}. \tag{3.11}$$

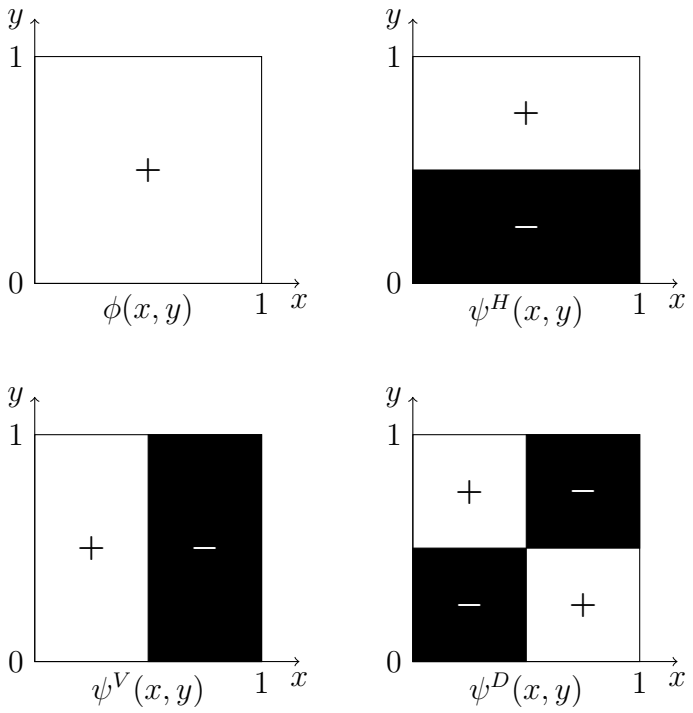


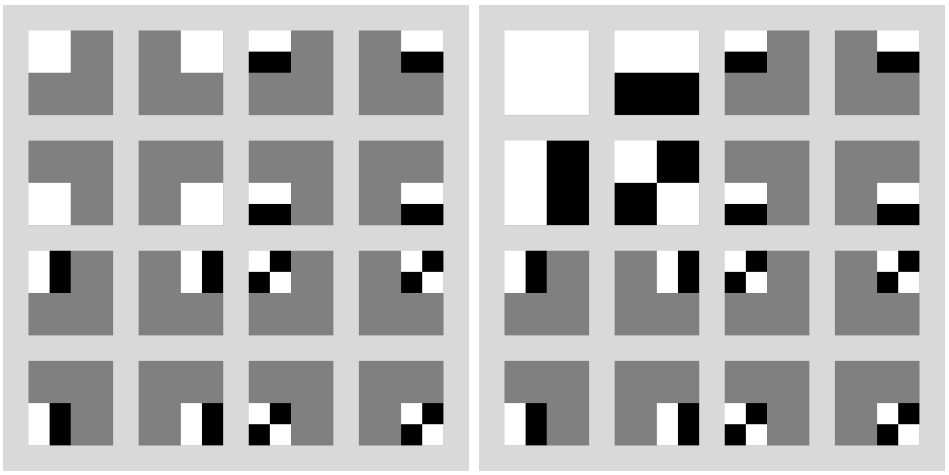
Fig. 3.5 The 2-D Haar scaling function and wavelet functions. Inside the $[0,1] \times [0,1]$ square, white corresponds to $+1$ and black corresponds to -1. The functions are zero outside the unit square.

3.2.3 Two-Dimensional Haar Wavelets

In Figure 3.3, we showed the 1-dimensional Haar scaling function $\phi(x)$ and wavelet function $\psi(x)$. In general, we can define the 2-dimensional scaling wavelet functions by taking the products of their one-dimensional versions:

$$\begin{aligned} \phi(x,y) &= \phi(x)\phi(y) \\ \psi^H(x,y) &= \phi(x)\psi(y) \\ \psi^V(x,y) &= \psi(x)\phi(y) \\ \psi^D(x,y) &= \psi(x)\psi(y) \end{aligned} \tag{3.12}$$

where the superscripts H, V and D refer to the high-pass filters in the horizontal, vertical and diagonal direction, respectively. These functions are illustrated in Figure 3.5 for the Haar wavelet system. The 2-D Haar wavelet functions act as *edge detectors* in their respective directions. For example, if we pass a 2×2 image patch through the ψ^V filter, the corresponding detail coefficient will be small unless there is sharp change



(a) Scale 1 (b) Scale 2

Fig. 3.6 The haar basis functions at the first scale (a) and the second scale (b). The DWT would use these basis functions to decompose a 4×4 image.

in pixel intensities between the left side and right side of the patch, i.e. unless there is a vertical edge.

The two-dimensional DWT of a signal is computed sequentially for each scale. We start by computing the 1-D DWT at the first scale for each individual column. Next, we do the same on each row of the intermediate result. This gives us the 2-D DWT at the first scale.

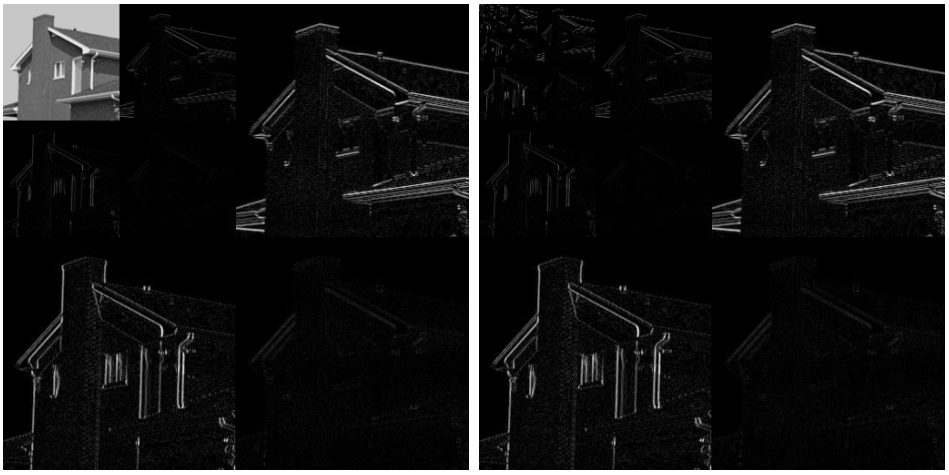
To get the next highest scale, we perform the scale 1 computations on the approximation coefficients of the current scale. This process process needs to be repeated j times to obtain the DWT at jth scale. [2]

In Figure 3.6, we illustrate the basis functions resulting corresponding to the 2-D DWT with Haar wavelets.. We show two sets of basis functions that can be used to decompose a signal of size 4×4, one corresponding to the first scale decomposition (panel (a)) and another set corresponding to the scale 2 (panel (b)).

In Figure 3.7, we have computed the DWT of a digital image at various scales. One can clearly see that the high-pass filters corresponding to the Haar wavelets act as edge detectors.

[2] This construction procedure is the so-called "non-standard" construction. For an alternative construction strategy that leads to a different set of 2-D basis functions, see [12].

(a) Original 512×512 image (b) Scale 1 DWT



(c) Scale 2 DWT (d) Scale 9 DWT

Fig. 3.7 The 2-D DWT using Haar wavelets of a digital image (a) of size 512×512. We compute the DWT at the first and the second scale (b,c) as well as the full DWT (d). In (d), we have a single approximation coefficient at the top left corner. In (b,c,d), the brightness of a pixel increases monotonically with the size of the corresponding DWT coefficient. We have enhanced the detail coefficients (relative to the approximation coefficients) for better visibility.

3.2.4 Haar Wavelets for Videos

A straightforward way of approaching the Discrete Wavelet Transform of a video would be to perform the two-dimensional DWT on each individual frame. This "pseudo 3D" approach is relatively simple, but typically also very inefficient. We expect there to be continuity between successive frames of a video. The pseudo 3D approach does not let us exploit any of the temporal correlations that are common in video signals.

A better approach would be to perform a full 3-D wavelet decomposition using three-dimensional scaling and wavelet functions. In 3-D, we have one scaling function $\phi(x, y, t)$ and a total of $2^3 - 1 = 7$ wavelet functions. These can be obtained by multiplying the 2-D functions in (3.12) with the scaling function $\phi(t)$ and the wavelet function $\psi(t)$ in the time domain:

$$\begin{aligned}
\phi(x, y, t) &= \phi(t)\phi(x)\phi(y) \\
\psi^H(x, y, t) &= \phi(t)\phi(x)\psi(y) \\
\psi^V(x, y, t) &= \phi(t)\psi(x)\phi(y) \\
\psi^D(x, y, t) &= \phi(t)\psi(x)\psi(y) \\
\psi^T(x, y, t) &= \psi(t)\phi(x)\phi(y) \\
\psi^{HT}(x, y, t) &= \psi(t)\phi(x)\psi(y) \\
\psi^{VT}(x, y, t) &= \psi(t)\psi(x)\phi(y) \\
\psi^{DT}(x, y, t) &= \psi(t)\psi(x)\psi(y)
\end{aligned}$$

We use the T superscript to denote the high-pass filters in the temporal direction.

Figure 3.8 shows the 3-D scaling and wavelet functions for the Haar wavelets. To obtain the basis functions corresponding to the 3-D Haar wavelet domain, we compute scaled and shifted versions of these functions:

$$\phi^j_{k_1,k_2,k_3}(x, y, t) = 2^{3j/2}\phi(2^j t - k_3)\phi(2^j x - k_1)\phi(2^j y - k_2)$$

At the first scale, the 3-D wavelet decomposition of a video results in 8 distinct channels: one corresponding to a low-pass filter, and 7 corresponding to high-pass filters in various directions. The approximation and detail coefficients of the various filters will be arranged as in Figure 3.9.

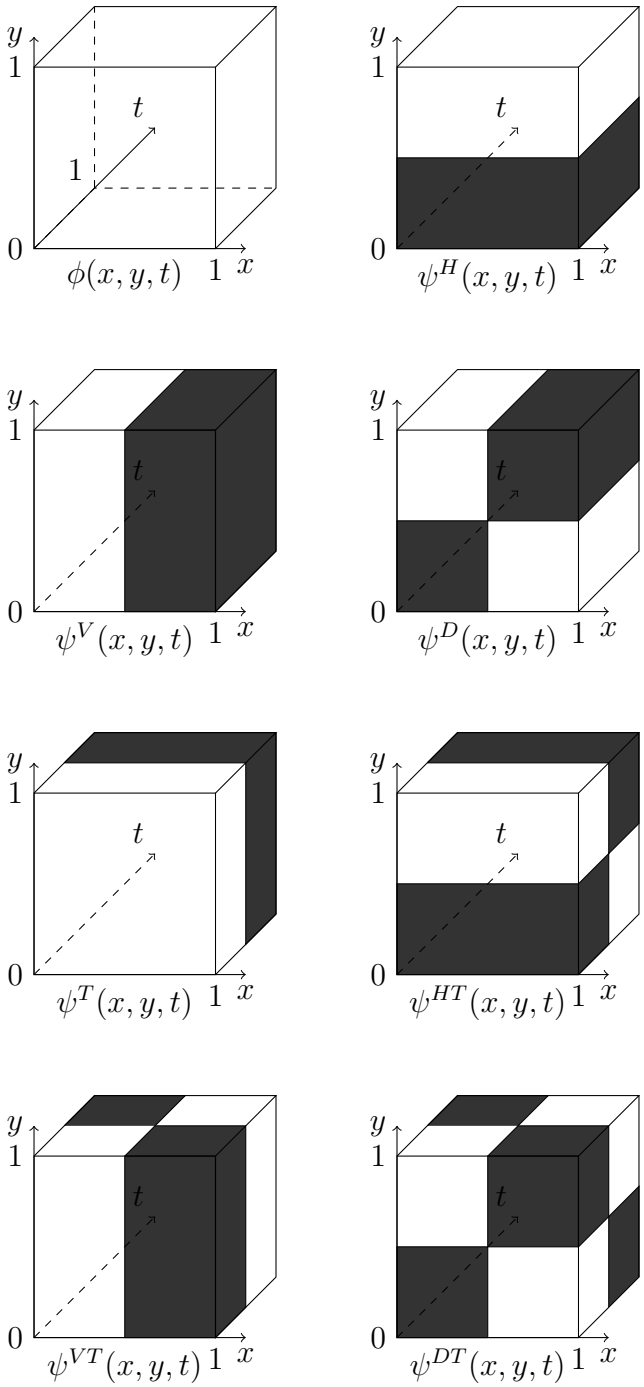


Fig. 3.8 The 3-D Haar scaling function and wavelet functions. Inside the $[0,1] \times [0,1] \times [0,1]$ cube, white corresponds to $+1$ and black corresponds to -1. The functions are zero outside of the cube.

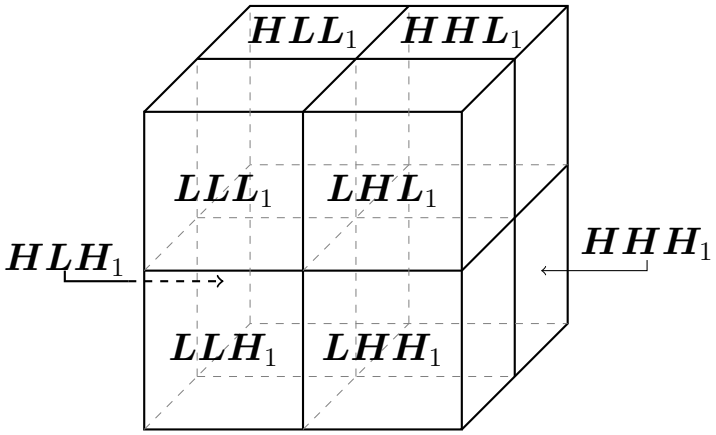
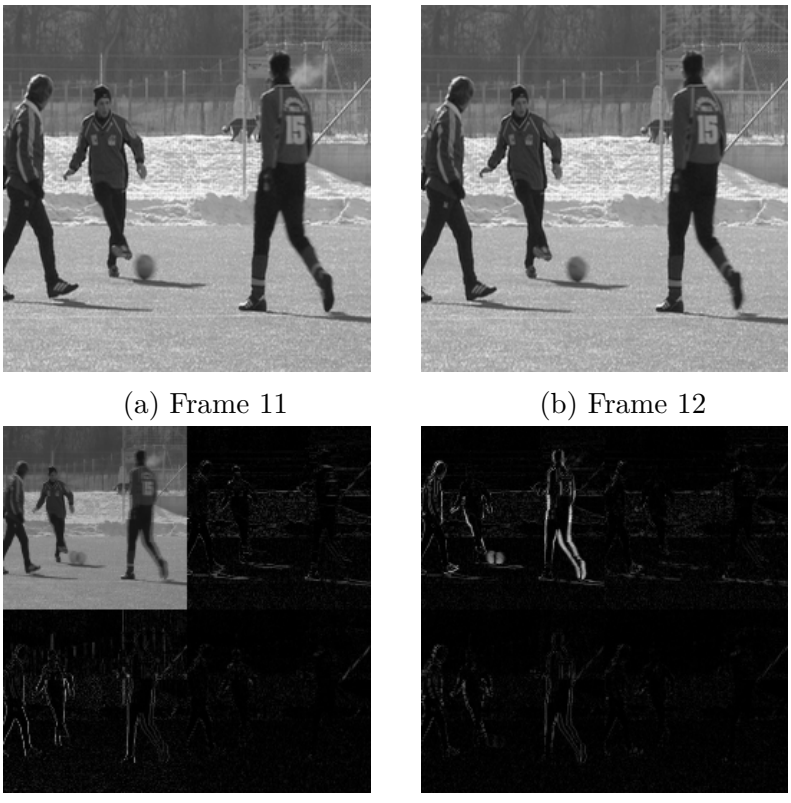


Fig. 3.9 The wavelet decomposition of a three-dimensional signal $v(x, y, t)$ at the first scale. L stands for "low" and H stands for "high". To obtain the decomposition at higher scales, we recursively decompose the low-pass channels $\boldsymbol{LLL}_j$ in a similar way.



(a) Frame 11 (b) Frame 12

(c) Scale 1 DWT: "frame" 6 (d) Scale 1 DWT: "frame" 38

Fig. 3.10 Frames 11 and 12 of a 64-frame video with a resolution of 256×256 (a,b). We perform the three-dimensional DWT at the first scale using Haar wavelets and show two "slices" of the resulting coefficients (c,d). The detail coefficients have been enhanced to improve visibility.

For the Haar wavelets, the high-pass filters act as edge detectors, just like in the 2-D case. However, on top of detecting edges along spatial directions, we also get high-pass filters that detect sudden changes in the temporal direction, i.e. motion detectors.

As an example, consider Figure 3.10. In panels (a) and (b), we show two adjacent frames of the "soccer" test video. We perform the 3-D DWT using Haar basis functions at the first scale. At scale one, the 3-D Haar basis functions have support over a $2 \times 2 \times 2$ volume. Thus, all the DWT coefficients that are affected by the data in frames 11 and 12 of the original video, are contained within two "frames", or "slices", of the transformed signal. For frames 11 and 12 of the original 64-frame video, these slices are at positions $12/2 = 6$ and $(6 + 64)/2 = 38$ in the transformed video.

In panel (c), we see the spatial edge detectors (the LLH, LHL and LHH channels) that are similar to those in the 2-D case (Figure 3.7). The top left corner corresponds to the low-pass filter LLL and can be regarded as an "average" of the video.

Panel (d), shows the four channels associated with a high-pass filter in the temporal direction. The HLL channel (top left corner of panel (d)) picks up overall differences in adjacent frames. In this example, we note that it detects the motion of the football players and the ball, (as well as a slight motion of the background due to the panning of the camera). The remaining channels in panel (d) combine motion detection with edge detection.

3.3 Construction of the Basis Matrix $\boldsymbol{\Psi}$

So far, when discussing basis transforms in two or three dimensions, we regarded the signal $\boldsymbol{v}$ as a 2-D or 3-D array, respectively. However, the theory in Chapter 2 was developed for signals in vectorized forms. Moreover, in Chapter 4 we will discuss the RVM which also operates on vectorized data. Therefore, it is important that we discuss how to vectorize our multi-dimensional signals.

In this section we will define the vectorization scheme that we used in our implementation and describe how to construct the basis matrix $\boldsymbol{\Psi}$ from the respective transform matrices in a way that is consistent with the vectorization.

3.3.1 Basis Matrix for One-Dimensional Signals

We start with 1-D signals. This case is trivial, since a one-dimensional signal $\boldsymbol{v}$ can directly be treated as a vector in $\mathbb{R}^M$, where M is the length of the signal.

The basis matrix matrix, in this case, is simply the inverse transform matrix. So, for the DCT, the basis matrix is $\boldsymbol{\Psi} = \boldsymbol{P}$, where $\boldsymbol{P}$ is defined in equation (3.3). We have defined the basis matrix corresponding to the discrete Haar wavelet transform in equation (3.11).

3.3.2 Basis Matrix for Two-Dimensional Signals

Suppose $\boldsymbol{A}$ is a two-dimensional signal of size $r \times c$ (i.e. r rows and c columns). Suppose further that r and c are powers of 2, $r = 2^{q_1}$ and $c = 2^{q_2}$, say.

We vectorize $\boldsymbol{A}$ in a row-major fashion. To be explicit, let $\boldsymbol{a}_j^T$ be the jth row of $\boldsymbol{A} \in \mathbb{R}^{r \times c}$. The vectorized form of $\boldsymbol{A}$ is then given by

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_r \end{bmatrix}$$

Thus, we end up with a long vector $\boldsymbol{v} \in \mathbb{R}^{rc}$.

DCT

To form the basis matrix corresponding to the two-dimensional DCT, we use the DCT's seperability property. Let $\boldsymbol{\Psi}_r$ and $\boldsymbol{\Psi}_c$ be the DCT basis matrices corresponding to one-dimensional signals of length r and c, respectively. The basis matrix corresponding to $\boldsymbol{v}$ is then constructed by forming the *kronecker product* of $\boldsymbol{\Psi}_c$ and $\boldsymbol{\Psi}_r$:

$$\boldsymbol{\Psi} = \boldsymbol{\Psi}_c \otimes \boldsymbol{\Psi}_r$$

The kronecker product $P \otimes Q$ between matrices P and Q with dimensions $m_P \times n_P$ and $m_Q \times n_Q$, respectively, is defined by the block matrix

$$\begin{bmatrix} P_{1,1}Q & P_{1,2}Q & \cdots & P_{1,n_P}Q \\ P_{2,1}Q & P_{2,2}Q & \cdots & P_{2,n_P}Q \\ \vdots & \vdots & \ddots & \vdots \\ P_{m_P,1}Q & P_{m_P,2}Q & \cdots & P_{m_P,n_P}Q \end{bmatrix} \tag{3.13}$$

of size $(m_P m_Q) \times (n_P n_Q)$.

DWT

For the 2-D Haar DWT at the first scale, the basis matrix is constructed in a similar fashion. Let $\boldsymbol{H}^{q_1}$ and $\boldsymbol{H}^{q_2}$ be defined according to equation (3.8), and let $\boldsymbol{G}^{q_1}$ and $\boldsymbol{G}^{q_2}$ be formed according to (3.9) (where we dropped the *haar* subscript). The basis matrix $\boldsymbol{\Psi}$ is constructed as follows:

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{H}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \end{bmatrix}^T .$$

For a signal of size $2^{q_1} \times 2^{q_2}$, we can build a cascade up to the qth scale, where $q = \min\{q_1, q_2\}$. The resulting basis matrix is given by

$$\boldsymbol{\Psi} = \begin{bmatrix} (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ \vdots \\ (\boldsymbol{G}^{q_2-2}\boldsymbol{H}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-2}\boldsymbol{H}^{q_1-1}\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-1}\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-1}\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-1}\boldsymbol{H}^{q_1}) \\ \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \end{bmatrix}^T .$$

Note that the dimensions of the basis matrix are $(rc) \times (rc)$, regardless of the scale that we use.

3.3.3 Basis Matrix for Three-Dimensional Signals

Let $\boldsymbol{V}$ be a video signal of size $r \times c \times f$ (r rows, c columns and f frames), where $r = 2^{q_1}$, $c = 2^{q_2}$ and $f = 2^{q_3}$.

We vectorize $\boldsymbol{V}$ by first vectorizing each individual frame in a row-major fashion (as in the 2-D case), followed by stacking the vectorized frames on top of each other. The result $\boldsymbol{v}$ is a long vector of length rcf.

DCT

The basis matrix corresponding to the DCT is given by

$$\boldsymbol{\Psi} = \boldsymbol{\Psi}_f \otimes \boldsymbol{\Psi}_c \otimes \boldsymbol{\Psi}_r$$

where $\boldsymbol{\Psi}_r$, $\boldsymbol{\Psi}_c$ and $\boldsymbol{\Psi}_f$ are the DCT basis matrices for 1-D signals (3.3) of length r, c and f, respectively.

Haar

The Haar basis matrix for video signals at scale 1 is constructed as follows:

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{H}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \end{bmatrix}^T .$$

where the matrices $\boldsymbol{H}$ and $\boldsymbol{G}$ are defined in equations (3.8) and (3.9).

The extension to higher scales is similar to equation (3.3.2). We can build the matrix up until the qth scale, where $q = \min\{q_1, q_2, q_3\}$. The full basis matrix is

$$\boldsymbol{\Psi} = \begin{bmatrix} (\boldsymbol{H}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{G}^{q_3-q}\boldsymbol{H}^{q_3-q+1}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q}\boldsymbol{H}^{q_2-q+1}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q}\boldsymbol{H}^{q_1-q+1}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q+1}\boldsymbol{H}^{q_3-q+2}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{H}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ (\boldsymbol{H}^{q_3-q+1}\boldsymbol{H}^{q_3-q+2}\cdots\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-q+1}\boldsymbol{H}^{q_2-q+2}\cdots\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{H}^{q_1-q+1}\boldsymbol{H}^{q_1-q+2}\cdots\boldsymbol{H}^{q_1}) \\ \vdots \\ (\boldsymbol{G}^{q_3-1}\boldsymbol{H}^{q_3}) \otimes (\boldsymbol{G}^{q_2-1}\boldsymbol{H}^{q_2}) \otimes (\boldsymbol{G}^{q_1-1}\boldsymbol{H}^{q_1}) \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{H}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{H}^{q_2} \otimes \boldsymbol{G}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{H}^{q_1} \\ \boldsymbol{G}^{q_3} \otimes \boldsymbol{G}^{q_2} \otimes \boldsymbol{G}^{q_1} \end{bmatrix}^T$$

For video signals, the basis matrix $\boldsymbol{\Psi}$ has dimensions $(hwf) \times (hwf)$.

Chapter 4

Sparse Bayesian Learning

Sparse Bayesian Learning [14] is a general Bayesian framework within supervised Machine Learning. It can be applied to both regression and classification tasks.

In this chapter, we will derive the Sparse Bayesian Learning model for regression. We will discuss model inference and summarise both the original inference algorithm [14] and also the faster "Sequential Sparse Bayesian Learning Algorithm" [15].

The *Relevance Vector Machine*, or *RVM*, is a particular specialisation of the Sparse Bayesian Learning model which has identical functional form to the Support Vector Machine (SVM). However, the RVM comes with a number of key advantages over the SVM. The solution produced by a RVM is typically much sparser than the solution by a comparable SVM. Furthermore, the RVM is a probabilistic model and as such, allows us to estimate error bounds for its predictions.

Throughout the thesis, we are only interested in the Sparse Bayesian Learning model. The terms "Sparse Bayesian Learning" and "RVM" will be used interchangibly,[1] but they always refer to the more general Sparse Bayesian Learning model.

4.1 Model Specification

We are given a data set of N input vectors $\{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ and their associated *targets* $\{y^{(i)}\}_{i=1}^{N}$. The input vectors live in D-dimensional space, $\boldsymbol{x} \in \mathbb{R}^D$. The targets are real values, $y \in \mathbb{R}$. [2]

[1] Except for the previous paragraph.

[2] When using the Sparse Bayesian model for regression, we assume the targets are real-valued. It is also possible to use the model for classification in which case the targets are assumed to be discrete class labels.

We model the data using a linearly-weighted sum of M fixed basis functions $\{\phi_j(\cdot)\}_{j=1}^M$ and base our predictions on the function $f(\cdot)$ defined as

$$f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{4.1}$$

where $\boldsymbol{w} = [w_1, \ldots, w_M]^T$ and $\boldsymbol{\phi}(\cdot) = [\phi_1(\cdot), \ldots, \phi_M(\cdot)]^T$. Using a large number M of non-linear basis functions $\phi_j : \mathbb{R}^D \to \mathbb{R}$ allows for a highly flexible model.

To train the model (4.1), i.e. find values for $\boldsymbol{w}$ that are optimal in some sense, we make the standard assumption that our training data are samples from the model with additive noise:

$$\begin{aligned} y^{(i)} &= f(\boldsymbol{x}^{(i)}; \boldsymbol{w}) + \epsilon^{(i)} \\ &= \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) + \epsilon^{(i)} \qquad i = 1, \ldots, N. \end{aligned} \tag{4.2}$$

The errors $\{\epsilon^{(i)}\}_{i=1}^N$ are assumed to be independent samples from a zero-mean Gaussian distribution with variance σ^2

$$p(\epsilon^{(i)}) = \mathcal{N}(\epsilon^{(i)} \,|\, 0, \sigma^2) \qquad i = 1, \ldots, N. \tag{4.3}$$

Combining equation (4.2) with equation (4.1), we may express the model for the complete data using matrix notation:

$$\boldsymbol{y} = \boldsymbol{\Phi} \boldsymbol{w} + \boldsymbol{\epsilon} \tag{4.4}$$

where $\boldsymbol{\epsilon} = \left[\epsilon^{(1)}, \cdots, \epsilon^{(N)}\right]^T$. The $N \times M$ matrix $\boldsymbol{\Phi}$ is known as the design matrix. The ith row of $\boldsymbol{\Phi}$ is given by $\boldsymbol{\phi}(\boldsymbol{x}^{(i)})^T$. The jth column of $\boldsymbol{\Phi}$ is given by $\boldsymbol{\phi}_j = \left[\phi_j\left(\boldsymbol{x}^{(1)}\right), \cdots, \phi_j\left(\boldsymbol{x}^{(N)}\right)\right]^T$, which is also referred to as the jth *basis vector*. Thus

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}_1 & \cdots & \boldsymbol{\phi}_M \end{bmatrix} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}^{(1)})^T \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}^{(N)})^T \end{bmatrix}$$

Combining equation (4.4) and equation (4.3), we find that the complete data likelihood function is given by

$$\begin{aligned} p\left(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2\right) &= \mathcal{N}\left(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2 \boldsymbol{I}_M\right) \\ &= (2\pi\sigma^2)^{-N/2} \exp\left\{-\frac{1}{2\sigma^2}||\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}||^2\right\} \end{aligned} \tag{4.5}$$

where $\boldsymbol{I}_M$ is the $M \times M$ identity matrix.

So far, we have specified the general linear regression model. To get to the sparse Bayesian formulation, we define a zero-mean Gaussian prior distribution over the parameters $\boldsymbol{w}$

$$p(\boldsymbol{w} \,|\, \boldsymbol{\alpha}) = \prod_{j=1}^{M} \mathcal{N}\left(w_j \,|\, 0, \alpha_j^{-1}\right) \tag{4.6}$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_M]^T$ is a vector of M hyperparameters. It is important to note that each hyperparameter α_j is solely responsible for controlling the strength of the prior of its associated weight w_j. If α_j is large, the prior over w_j is very strongly peaked at zero. This form of the prior distribution is, more than anything, responsible for the dramatic sparsity in the final model.

To complete the specification, we must define a prior over the noise parameter σ^2 and a hyperprior over the hyperparameters $\boldsymbol{\alpha}$. Following the derivation in [14], we use the following Gamma [3] priors

$$p(\boldsymbol{\alpha} \,|\, a, b) = \prod_{j=1}^{M} \text{Gamma}(\alpha_j \,|\, a, b) \tag{4.7}$$

$$p(\beta \,|\, c, d) = \text{Gamma}(\beta \,|\, c, d) \tag{4.8}$$

where $\beta \equiv \sigma^{-2}$.

As a side note, consider the prior of $\boldsymbol{w}$ after marginalising out the dependence on the hyperpriors $\boldsymbol{\alpha}$. Since each w_j is normally distributed with an unknown precision parameter α_j and since the (hyper)prior over α_j is the Gamma distribution and

[3] The Gamma distribution is defined by

$$\text{Gamma}(z \,|\, a, b) = \Gamma(a)^{-1} b^a z^{a-1} \exp(-bz) \qquad z, a, b > 0$$

where $\Gamma(.)$ is the Gamma function defined by

$$\Gamma(z) = \int_0^\infty t^{z-1} \exp(-t) dt.$$

therefore conjugate to $p(w_j \,|\, \alpha_j)$, it follows that the resulting integral can be evaluated analytically

$$\begin{aligned} p(\boldsymbol{w} \,|\, a, b) &= \int p(\boldsymbol{w} \,|\, \boldsymbol{\alpha}) p(\boldsymbol{\alpha} \,|\, a, b) d\boldsymbol{\alpha} \\ &= \prod_{j=1}^{M} \int \mathcal{N}(w_j \,|\, 0, \alpha_j^{-1}) \,\mathrm{Gamma}(\alpha_j |\, a, b) d\alpha_j \\ &= \prod_{j=1}^{M} \frac{b^a \Gamma(a + \frac{1}{2})}{(2\pi)^{\frac{1}{2}} \Gamma(a)} \left(b + \frac{w_j^2}{2} \right)^{-(a+\frac{1}{2})}. \end{aligned}$$

This corresponds to a product of independent Student-t density functions over the weights w_j. For small positive values of a and b, the distribution $p(\boldsymbol{w} \,|\, a, b)$ will be strongly peaked at zero. As discussed in [14], it is this hierarchical formulation of the weight prior that is ultimately responsible for encouraging sparse solutions.

4.2 Model Inference

We have specified the likelihood model for the data and a prior distribution over the model parameters. The next step in Bayesian inference is to compute the posterior distribution of the parameters. We begin by setting up Bayes' Rule

$$p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y}) = \frac{p(\boldsymbol{y} \,|\, \boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2)}{\int p(\boldsymbol{y} \,|\, \boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) d\boldsymbol{w} d\boldsymbol{\alpha} d\sigma^2} \tag{4.9}$$

The integral in the denominator of (4.9) is computationally intractable and we must resort to an alternative strategy. First, we decompose the left-hand-side of equation (4.9) as

$$p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y}) = p(\boldsymbol{w} \,|\, \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{\alpha}, \sigma^2 \,|\, \boldsymbol{y}).$$

Next, we use Bayes' Rule to compute the posterior distribution of the weights given $\boldsymbol{\alpha}$ and σ^2

$$p(\boldsymbol{w} \,|\, \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) = \frac{p(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2) p(\boldsymbol{w} \,|\, \boldsymbol{\alpha})}{p(\boldsymbol{y} \,|\, \boldsymbol{\alpha}, \sigma^2)} \tag{4.10}$$

The denominator of the right-hand-side is known as the *marginal likelihood* and given by

$$p(\boldsymbol{y} \,|\, \boldsymbol{\alpha}, \sigma^2) = \int p(\boldsymbol{y} \,|\, \boldsymbol{w}, \sigma^2) p(\boldsymbol{w} \,|\, \boldsymbol{\alpha}) d\boldsymbol{w} \tag{4.11}$$

Since $\boldsymbol{\alpha}$ and σ^2 are treated as fixed quantities in equation (4.10), the Gaussian density $p(\boldsymbol{w} \mid \boldsymbol{\alpha})$ is the conjugate prior to the Gaussian likelihood function $p(\boldsymbol{y} \mid \boldsymbol{w}, \sigma^2)$. Thus, the integral in equation (4.11) is a convolution of two Gaussians and therefore equal to another Gaussian:

$$\begin{aligned} p(\boldsymbol{y} \mid \boldsymbol{\alpha}, \sigma^2) &= \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{C}) \\ &= (2\pi)^{-N/2} |\boldsymbol{C}|^{-1/2} \exp\left\{-\frac{1}{2}\boldsymbol{y}^T \boldsymbol{C}^{-1} \boldsymbol{y}\right\} \end{aligned}$$

where

$$\boldsymbol{C} = \sigma^2 \boldsymbol{I}_N + \boldsymbol{\Phi} \boldsymbol{A}^{-1} \boldsymbol{\Phi}^T. \tag{4.12}$$

The posterior distribution for $\boldsymbol{w}$ is a also a Gaussian:

$$p(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{4.13}$$

Its mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ are given by

$$\boldsymbol{\Sigma} = \left(\sigma^{-2} \boldsymbol{\Phi}^T \boldsymbol{\Phi} + \boldsymbol{A}\right)^{-1} \tag{4.14}$$

$$\boldsymbol{\mu} = \sigma^{-2} \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \boldsymbol{y} \tag{4.15}$$

with $\boldsymbol{A} = \mathrm{diag}(\boldsymbol{\alpha})$.

Thus, to train the model, we need to find the hyperparameters $\boldsymbol{\alpha}$ and σ^2. In the context of the RVM, these hyperparameters are usually estimated from the data by performing a maximization of the marginal likelihood, or equivalently, its logarithm

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha}, \sigma^2) &= \log p(\boldsymbol{y} \mid \boldsymbol{\alpha}, \sigma^2) = \log \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{C}) \\ &= -\frac{1}{2}\left[N \log 2\pi + \log |\boldsymbol{C}| + \boldsymbol{y}^T \boldsymbol{C}^{-1} \boldsymbol{y}\right] \end{aligned} \tag{4.16}$$

4.2.1 Original Training Algorithm

The original training algorithm in [14] can be obtained by setting the derivatives of (4.16) to zero and rearring to get the following update equations for $\boldsymbol{\alpha}$ and σ^2:

$$\alpha_j^{\text{new}} = \frac{\gamma_j}{\mu_j^2} \tag{4.17}$$

$$(\sigma^2)^{\text{new}} = \frac{||\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}||^2}{N - \sum_j \gamma_j} \tag{4.18}$$

Algorithm 1 Sparse Bayesian Learning: Original Training Algorithm

Choose some initial positive values for σ^2 and α_j for $j = 1, \cdots, M$
repeat
  $\boldsymbol{A} = \text{diag}(\boldsymbol{\alpha})$
  $\boldsymbol{\Sigma} = \left(\sigma^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \boldsymbol{A}\right)^{-1}$
  $\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{y}$

  for $j = 1, \cdots, M$ **do**
    $\gamma_j = 1 - \alpha_j\Sigma_{jj}$
    $\alpha_j = \gamma_j/\mu_j^2$
  end for
  $\sigma^2 = ||\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}||^2/(N - \sum_j \gamma_j)$
until Convergence

where μ_j is the jth component of the posterior mean $\boldsymbol{\mu}$ (4.15). The quantities γ_j are defined by

$$\gamma_j = 1 - \alpha_j\Sigma_{jj}$$

where Σ_{jj} is the jth diagonal element of the posterior covariance $\boldsymbol{\Sigma}$ (4.14).

To train the model, we can start by giving $\boldsymbol{\alpha}$ and σ^2 some initial values and evaluate the mean and covariance of the weights posterior using equations (4.15) and (4.14), respectively. Next we alternate between re-estimating the hyperparamaters $\boldsymbol{\alpha}$ and σ^2 using (4.17) and (4.18) and updating the posterior mean and covariance parameters using (4.15) and (4.14). We continue until a relevant convergence criterion is met. For example, we may choose to stop if the change in the marginal likelihood between two consecutive iterations is below a certain pre-defined threshold.

This procedure is summarised in Algorithm 1.

During training, it is typically observed that many of the hyperparameters α_j tend to infinity. Equations (4.15) and (4.14) imply that the weights w_j corresponding to these hyperparameters have a posterior distribution where the mean and the variance are both zero, meaning their posterior is infinitely peaked at zero. As a consequence, the corresponding basis functions $\phi_j(\cdot)$ are effectively removed from the model and we achieve sparsity.

4.2.2 Sequential Sparse Bayesian Learning Algorithm

A central drawback of the training algorithm discussed in the previous section is its speed. The computational complexity scales with the cube of the number of basis

functions. During training, as basis functions are pruned from the model, the algorithm accelerates. Nevertheless, if M is very large, the procedure can be very expensive to run.

An alternative strategy of maximising the marginal likelihood (4.16) was developed by [15], resulting in a highly accelerated training algorithm: the *Sequential Sparse Bayesian Learning Algorithm* (SSBL). It starts with a single basis function and maximises the marginal likelihood by sequentially adding, re-estimating or deleting candidate basis functions. This significantly reduces the computational complexity of the algorithm.

To derive the algorithm, we follow the analysis in [13] and consider the dependence of the marginal likelihood $\mathcal{L}(\boldsymbol{\alpha}, \sigma^2)$ on a single hyperparameter α_j. First, we decompose the matrix $\boldsymbol{C}$, defined in (4.12), as follows:

$$\begin{aligned}\boldsymbol{C} &= \sigma^2 \boldsymbol{I}_N + \sum_{m \neq j} \alpha_m^{-1} \boldsymbol{\phi}_m \boldsymbol{\phi}_m^T + \alpha_j^{-1} \boldsymbol{\phi}_j \boldsymbol{\phi}_j^T \\ &= \boldsymbol{C}_{-j} + \alpha_j^{-1} \boldsymbol{\phi}_j \boldsymbol{\phi}_j^T\end{aligned}$$

where $\boldsymbol{C}_{-j} \equiv \sigma^2 \boldsymbol{I}_N + \sum_{m \neq j} \alpha_m^{-1} \boldsymbol{\phi}_m \boldsymbol{\phi}_m^T$ is $\boldsymbol{C}$ without the contribution of the jth basis vector $\boldsymbol{\phi}_j$. Making use of the Woodbury identity and the matrix determinant lemma, we can express $|\boldsymbol{C}|$ and $\boldsymbol{C}^{-1}$ as

$$\boldsymbol{C}^{-1} = \boldsymbol{C}_{-j}^{-1} - \frac{\boldsymbol{C}_{-j}^{-1} \boldsymbol{\phi}_j \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1}}{\alpha_j + \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1} \boldsymbol{\phi}_j}$$

$$|\boldsymbol{C}| = |\boldsymbol{C}_{-j}| \left| 1 + \alpha_j^{-1} \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1} \boldsymbol{\phi}_j \right|$$

This allows us to decompose the marginal likelihood:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\alpha}, \sigma^2) &= \mathcal{L}(\boldsymbol{\alpha}_{-j}, \sigma^2) + \frac{1}{2} \left[\log \alpha_j - \log(\alpha_j + s_j) + \frac{q_j^2}{\alpha_j + s_j} \right] \\ &\equiv \mathcal{L}(\boldsymbol{\alpha}_{-j}, \sigma^2) + \ell(\alpha_j, \sigma^2)\end{aligned} \tag{4.19}$$

This conveniently separates terms in α_j in $\ell(\alpha_j, \sigma^2)$ from the remaining terms in $\mathcal{L}(\boldsymbol{\alpha}_{-j}, \sigma^2)$, which is the (log) marginal likelihood with the basis vector $\boldsymbol{\phi}_j$ excluded.

The quantity s_j is the *sparsity factor*, defined as

$$s_j = \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1} \boldsymbol{\phi}_j.$$

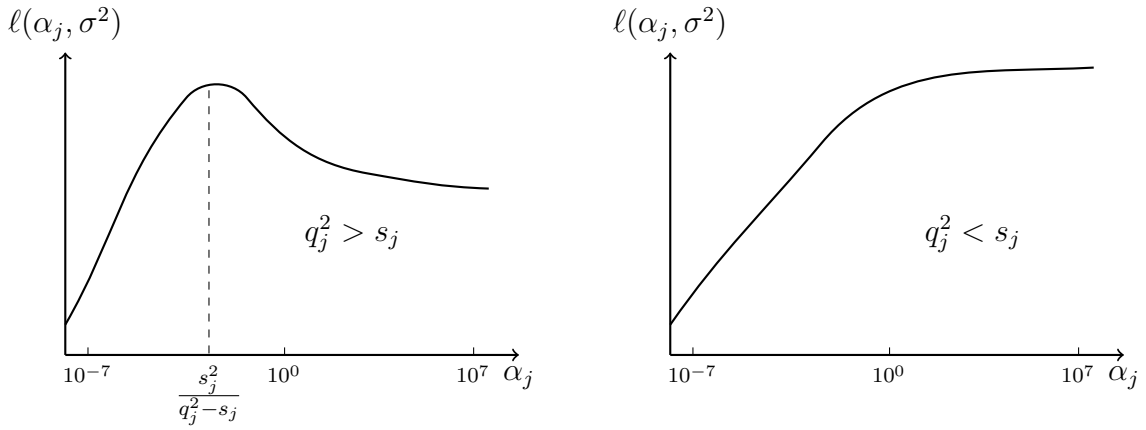


Fig. 4.1 Example plots of $\ell(\alpha_j, \sigma^2)$ against α_j illustrating the stationary points when $q_j^2 > s_j$ (left) and $q_j^2 < s_j$ (based on [13]).

It serves as a measure of how much the marginal likelihood would decrease if we added $\boldsymbol{\phi}_j$ to the model. The quantity q_j, on the other hand, is known as the *quality factor*. It is defined as

$$q_j = \boldsymbol{\phi}_j^T \boldsymbol{C}_{-j}^{-1} \boldsymbol{y}$$

and measures the extent to which ϕ_j increases $\mathcal{L}(\boldsymbol{\alpha}, \sigma^2)$ by helping to explain the data $\boldsymbol{y}$. Thus, a particular basis vector $\boldsymbol{\phi}_j$ should not be included in the model if its sparsity factor s_j is large, unless it is offset by a large quality factor q_j.

We can see this more explicitly if we consider the first derivative of $\ell(\alpha_j, \sigma^2)$ with respect to α_j [13]

$$\frac{\partial \ell(\alpha_j, \sigma^2)}{\partial \alpha_j} = \frac{\alpha_j^{-1} s_j^2 - (q_j^2 - s_j)}{2(\alpha_j + s_j)^2}$$

Equating it to zero (and noting that α_j is an inverse-variance and therefore positive), we obtain the following solution for α_j:

$$\alpha_j = \begin{cases} s_j^2/(q_j^2 - s_j) & \text{if } q_j^2 > s_j \\ +\infty & \text{otherwise} \end{cases} . \tag{4.20}$$

The solution (4.20) is illustrated in Figure 4.1.

It follows that, if, during training, a candidate basis vector $\boldsymbol{\phi}_j$ is currently included in the model (meaning $\alpha_j < \infty$) even though $q_j^2 \leq s_j$, then α_j should be set to ∞ and $\boldsymbol{\phi}_j$ should be pruned from the model. On the other hand, if $\boldsymbol{\phi}_j$ is currently excluded from the model (i.e. $\alpha_j = \infty$), but $q_j^2 > s_j$, then α_j should be set to $s_j^2/(q_j^2 - s_j)$ and $\boldsymbol{\phi}_j$ should be added to the model. Furthermore, if $\boldsymbol{\phi}_j$ is included and $q_j^2 > s_j$, then we

Algorithm 2 Sequential Sparse Bayesian Learning Algorithm [15]

1: Initialise σ^2.
2: Add basis function $\boldsymbol{\phi}_j$ to the model, where $j = \arg\max_m \left\{ ||\boldsymbol{\phi}_m^T \boldsymbol{y}||^2 / ||\boldsymbol{\phi}_m||^2 \right\}$. Set $\alpha_j = \frac{||\boldsymbol{\phi}_j||^2}{||\boldsymbol{\phi}_j^T \boldsymbol{y}||^2 / ||\boldsymbol{\phi}_j||^2 - \sigma^2}$. Set $\alpha_m = \infty$ for $m \neq j$.
3: Compute $\boldsymbol{\Sigma} = \left(\sigma^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \boldsymbol{A}\right)^{-1}$ and $\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{y}$ which are scalars initially. Compute S_m, Q_m, s_m and q_m for $m = 1, \cdots, M$ using (4.21) − (4.24).
4: **repeat**
5: Select some candidate basis vector $\boldsymbol{\phi}_j$.
6: **if** $q_j^2 > s_j$ and $\alpha_j = \infty$ **then add** $\boldsymbol{\phi}_j$ to the model and update α_j.
7: **if** $q_j^2 > s_j$ and $\alpha_j < \infty$ **then re-estimate** α_j.
8: **if** $q_j^2 < s_j$ and $\alpha_j < \infty$ **then delete** $\boldsymbol{\phi}_j$ from the model and set $\alpha_j = \infty$.
9: Update $\sigma^2 = ||\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}||/(N - M + \sum_m \alpha_m \boldsymbol{\Sigma}_{mm})$[14].
10: Update $\boldsymbol{\Sigma}$, $\boldsymbol{\mu}$ and S_m, Q_m, s_m, q_m for $m = 1, \cdots, M$.
11: **until** Convergence

may also re-estimate α_j. Each step in the algorithm increases the marginal likelihood. Thus we are guaranteed to find a maximum.

During the algorithm, we must maintain and update values of the quality factors and sparsity factors for all basis functions, as well as the posterior mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ of the weights $\boldsymbol{w}$. In practice, it easier to keep track of the quantities $Q_m = \boldsymbol{\phi}_m^T \boldsymbol{C}^{-1} \boldsymbol{\phi}_m$ and $S_m = \boldsymbol{\phi}_m^T \boldsymbol{C}^{-1} \boldsymbol{y}$ which can also be written as (using the Woodbury Identity)

$$S_m = \sigma^{-2}\boldsymbol{\phi}_m^T\boldsymbol{\phi}_m - \sigma^{-4}\boldsymbol{\phi}_m^T\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{\phi}_m \tag{4.21}$$

$$Q_m = \sigma^{-2}\boldsymbol{\phi}_m^T\boldsymbol{y} - \sigma^{-4}\boldsymbol{\phi}_m^T\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{y} \tag{4.22}$$

where $\boldsymbol{\Sigma}$ and $\boldsymbol{\Phi}$ contain only the basis functions that are currently included in the model.

The factors s_m and q_m can by obtained from S_m and Q_m as follows:

$$s_m = \frac{\alpha_m S_m}{\alpha_m - S_m} \tag{4.23}$$

$$q_m = \frac{\alpha_m Q_m}{\alpha_m - S_m} \tag{4.24}$$

Note that if $\alpha_m = \infty$, then $q_m = Q_m$ and $s_m = S_m$.

We have summarized the procedure in Algorithm 2. After initializing the standard deviation σ^2 in step 1, we add the first basis function $\boldsymbol{\phi}_j$ to the model. We could

initialize with any basis vector, but in step 2, we pick the one with the largest normalized projection on the target vector $\boldsymbol{y}$, i.e. we choose $j = \arg\max_m \left\{||\boldsymbol{\phi}_m^T \boldsymbol{y}||^2/||\boldsymbol{\phi}_m||^2\right\}$. In step 3 we compute the model statistics and in step 4 we begin the large loop of the algorithm.

There are two things to note here. First, in step 5, we need to select a candidate basis vector $\boldsymbol{\phi}_j$. We are free to pick one at random. Alternatively, it is possible to compute the change in the marginal likelihood for each candidate basis vector and choose the one that would give us the largest increase.

Second, we would usually like to estimate the noise variance σ^2 from the data, as is done in step 9. However, in practice, we may decide to set σ^2 in advance in step 1 and keep it fixed throughout the algorithm. If we decide to do so, then we can perform the updates in step 10 using very efficient update formulae that do not require matrix inversions. The formulae can be found in the appendix of [15]. If we do decide to update σ^2 in step 9, then we must use the full equations (4.14), (4.15) and (4.21)-(4.24). Alternatively, we could also choose to update σ^2 after a set number of iterations.

4.3 Making Predictions

Once we have trained the model, we may use it to predict the target y^* for a new input vector $\boldsymbol{x}^*$. To do so, we would like to compute the *predictive distribution*

$$p(y^* \mid \boldsymbol{y}) = \int p(y^* \mid \boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2) p(\boldsymbol{w}, \boldsymbol{\alpha}, \sigma^2 \mid \boldsymbol{y})\, d\boldsymbol{w}\, d\boldsymbol{\alpha}\, d\sigma^2.$$

We cannot compute this integral analytically, nor do we actually know the posterior of all the model parameters. Instead, we use the type-II maximum likelihood solutions for $\boldsymbol{\alpha}$ and σ^2 that we obtained during training and base our predictions on the posterior distribution of the weights conditioned on $\boldsymbol{\alpha}$ and σ^2. The predictive distribution for y^* is then:

$$p(y^* \mid \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) = \int p(y^* \mid \boldsymbol{w}, \sigma^2) p(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2)\, d\boldsymbol{w} \tag{4.25}$$

Both factors in the integrand are Gaussians, and we can therefore readily compute the integral to get

$$p(y^* \mid \boldsymbol{y}, \boldsymbol{\alpha}, \sigma^2) = \mathcal{N}(y^* \mid \mu^*, (\sigma^2)^*) \tag{4.26}$$

The predictive mean is given by

$$\mu^* = \boldsymbol{\mu}^T \boldsymbol{\phi}(\boldsymbol{x}^*) \tag{4.27}$$

and the predictive variance is given by

$$(\sigma^2)^* = \sigma^2 + \boldsymbol{\phi}(\boldsymbol{x}^*)^T \boldsymbol{\Sigma} \boldsymbol{\phi}(\boldsymbol{x}^*) \tag{4.28}$$

Equation (4.27) implies that, if we want to produce point predictions, we may simply set the weights $\boldsymbol{w}$ equal to posterion mean $\boldsymbol{\mu}$ which is typically very sparse.

If we are also interested in error bars for our predictions, we can obtain them using Equation (4.28). The error bars consist of two parts, the noise in the data σ^2 and the uncertainty in the weights. We may need to be careful when using the error bars as they become inreliable the further we move away from the training set [10].

For more details and derivations on Sparse Bayesian Learning, see [14, 13, 15].

Chapter 5

Bayesian Compressive Sensing

In this chapter, we combine the Compressive Sensing framework from Chapter 2 with the Sparse Bayesian Learning framework from Chapter 4 to formulate the *Bayesian Compressive Sensing* (BCS) algorithm.

Following that, we discuss the scenario where the sensing mechanism $\boldsymbol{\Theta}$ corresponds to a signal mask. We describe how the performance of the BCS algorithm can be boosted via the *Multi-Scale Cascade of Estimations* algorithm.

5.1 Compressive Sensing as Linear Regression

In the Bayesian Compressive Sensing (BCS) [6] framework, we attempt to approach the Compressive Sensing problem from the point of view of linear regression.

Recall the setup from Chapter 2. We are intersted in recovering the signal $\boldsymbol{v} \in \mathbb{R}^M$ from the CS measurements $\boldsymbol{\Theta v} = \boldsymbol{y} \in \mathbb{R}^N$, where $N << M$. We also assume that $\boldsymbol{v}$ is compressible in the basis $\boldsymbol{\Psi}$.

This means that $\boldsymbol{w} = \boldsymbol{\Psi}^T \boldsymbol{v}$ is well approximated by the K-sparse vector $\boldsymbol{w}_K$ which is identical to $\boldsymbol{w}$ for the K elements in $\boldsymbol{w}$ with the largest magnitude.

Thus, $\boldsymbol{w}_e = \boldsymbol{w} - \boldsymbol{w}_K$ is assumed to be very small in magnitude. Letting $\boldsymbol{\Phi} = \boldsymbol{\Theta\Psi}$, we have

$$\boldsymbol{y} = \boldsymbol{\Theta v} = \boldsymbol{\Theta\Psi w} = \boldsymbol{\Phi w} = \boldsymbol{\Phi w}_K + \boldsymbol{\Phi w}_e = \boldsymbol{\Phi w}_K + \boldsymbol{n}_e$$

where $\boldsymbol{n}_e = \boldsymbol{\Phi w}_e$ will be regarded as noise. Furthermore, there could be an additional noise source $\boldsymbol{n}$ in the CS measurements $\boldsymbol{y}$ (e.g. due to finite precision in the measurement

device) so that, overall,

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{w}_K + \boldsymbol{n}_e + \boldsymbol{n} = \boldsymbol{\Phi}\boldsymbol{w}_K + \boldsymbol{\epsilon} \tag{5.1}$$

with $\boldsymbol{\epsilon} = \boldsymbol{n}_e + \boldsymbol{n}$.

In the BCS literature [6], $\boldsymbol{\epsilon}$ is typically approximated by a zero-mean Gaussian noise variable with an unknown variance σ^2:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I}_N). \tag{5.2}$$

Although this assumption may not always be completely accurate in practice, it is often used for its desirable analytical properties. A possible justification is that the sensing matrix $\boldsymbol{\Theta}$ is usually constructed randomly (often using Gaussian samples) and that the noise source $\boldsymbol{n}$ in the CS measurements is typically represented by a zero-mean Gaussian distribution.

Note the similarity between equations (5.1) and (5.2) and the geneneral linear regression model in equations (4.4) and (4.3). The only difference is that, in equation (5.1), we have the addtional constraint that $\boldsymbol{w}_K$ is *sparse*.

Since in the BCS framework, we wish to solve the CS problem from a Bayesian standpoint, we are interested in computing a full posterior distribution of the weights $\boldsymbol{w}_K$ and the noise variance σ^2. The sparsity constraint can be entered into the model by placing a sparseness-inducing prior distribution over the weights $\boldsymbol{w}_K$.

There is a range of popular sparseness priors. However, to make the connection to the RVM and exploit the analysis from Chapter 4, we choose the prior (4.6)

$$p(\boldsymbol{w}_K \,|\, \boldsymbol{\alpha}) = \prod_{j=1}^{M} \mathcal{N}\left(w_j \,|\, 0, \alpha_j^{-1}\right)$$

To summarize, we solve the CS problem by passing the CS measurements $\boldsymbol{y}$ and the matrix $\boldsymbol{\Phi}$ to train a RVM. The SBBL algorithm (Algorithm 2) is then used to compute the posterior distribution of the weights $\boldsymbol{w} \in \mathbb{R}^M$ (4.13).

The desired signal $\boldsymbol{v} \in \mathbb{R}^M$ can then by recovered by computing

$$\hat{\boldsymbol{v}} = \boldsymbol{\Psi}\boldsymbol{\mu} \tag{5.3}$$

where $\boldsymbol{\mu}$ is the posterior mean of $\boldsymbol{w}$.

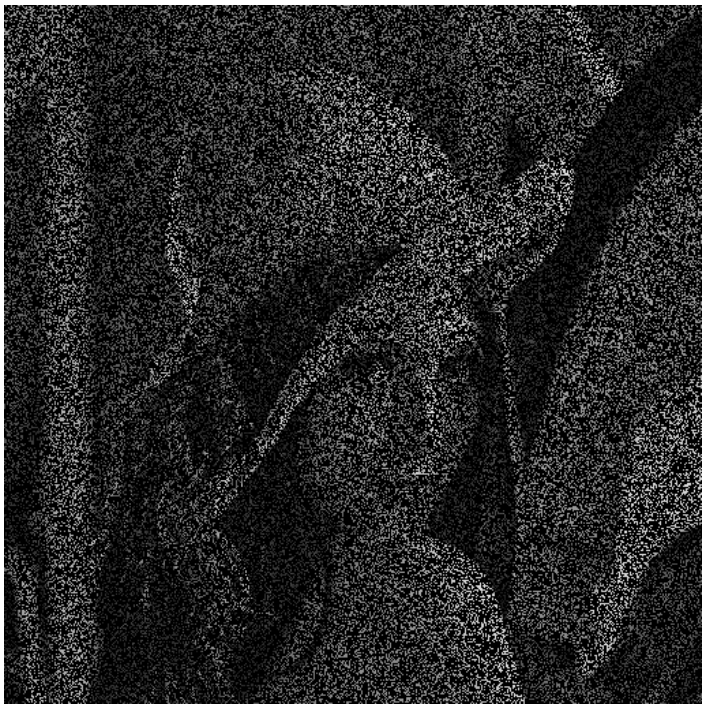


Fig. 5.1 Example of an image with missing pixel values. The missing data points are set to zero and displayed in black

The method of applying the RVM to CS inversion is sometimes referred to as the Bayesian Compressive Sensing (BCS) algorithm. The BCS algorithm was compared against state-of-the-art deterministic CS algorithms by [6, 9] and shown to give comparable performance.

We have implemented the BCS algorithm as part of our Compressive Sensing algorithm. Using BCS, our implementation is able to reconstruct video signals from a small number of CS measurements that were sampled by a general sensing mechanism $\boldsymbol{\Theta}$.

5.2 The Case when $\boldsymbol{\Theta}$ is a Signal Mask

5.2.1 Signal Interpolation

If the sensing matrix $\boldsymbol{\Theta}$ is a signal mask (see Section 2.3.1), the CS measurements y of an image $\boldsymbol{v}$ can be visualized as an image with missing pixel values as in Figure 5.1. The problem of reconstructing such signals is sometimes referred to as *interpolation*.

In this context, the design matrix $\boldsymbol{\Phi}$ can be computed efficiently from $\boldsymbol{\Psi}$ by simply deleting all the rows in $\boldsymbol{\Psi}$ that correspond to the entries of $\boldsymbol{v}$ that have missing data.

Moreover, it is possible to get a slight performance boost in the reconstruction by filling in the BCS solution $\hat{\boldsymbol{v}}$ (5.3) with the measured at data $\boldsymbol{y}$ at the appropriate locations.

5.2.2 Reconstruction using Haar Wavelets

Using the BCS model from above with a Haar basis matrix, we reconstruct the image in Figure 5.1. We display the output in Figure 5.2 for various scales of the Haar basis.

We see that the scale of the wavelet basis has a strong effect on the quality of the reconstruction. Wavelets at lower scales can fail to reconstruct the entire image, but generally give a more accurate reconstruction at the portion of the image where they succeed. On the other, wavelets at larger scales typically achieve a reconstruction of the entire image at the cost of increased blurring or pixelation. This conflict can be seen in Figure 5.2 by contrasting panel (c) with panel (d): They both achieve a reconstruction of the entire image, but scale 3 outperforms scale 4 in this example. Moreover, if we compare panel (b) with panel (d), we see that, in the part of the image where the recovery succeeds, the scale 2 reconstruction manages to recover finer details than both scale 3 and scale 4.

To explain why this tradeoff exists, recall that two-dimensional Haar wavelets at scale s have a support of size up to $2^s \times 2^s$. Thus, at small scales, the support of the individual basis functions is small. For example, suppose $s = 1$. In this case, each Haar basis functions covers an area of 2×2 pixels (see Figure 3.6). Using these wavelets, we can capture very local relationships between the pixels. This allows us to recover finer details in the reconstruction.

However, it can also lead to the formation of black pixels. At scale 1, each column j of the Haar basis matrix $\boldsymbol{\Psi}$ contains exactly 4 non-zero entries corresponding to the four pixel locations that are covered by the jth basis function. If the masked image happens to be missing data at all four of these locations, the corresponding rows of the basis matrix $\boldsymbol{\Psi}$ will be deleted when forming the design matrix $\boldsymbol{\Phi} = \boldsymbol{\Theta}\boldsymbol{\Psi}$. Therefore, column j of the design matrix will be zero. The same is true for the three columns that correspond to the remaining locations that were covered by basis function j.

The SSBL algorithm will generally not select any columns that are completely zero, since they offer no change to the marginal likelihood. The consequence is that the

(a) Scale 1: PSNR = 11.75 (b) Scale 2: PSNR = 21.78

(c) Scale 3: PSNR = 24.15 (d) Scale 4: PSNR = 23.72

Fig. 5.2 Reconstruction if the image in Figure 5.1 using the RVM with Haar wavelets at various scales. We use the *Peak Signal-to-Noise Ratio (PSNR)* as our performance metric. The larger the PSNR, the more accurate the reconstruction. See Chapter 7 for a definition of the PSNR.

corresponding entries in the posterior mean $\boldsymbol{\mu}$ of $\boldsymbol{w}$ will remain zero. The resulting reconstruction (5.3) will therefore be unable to recover the 2×2 patch covered by basis function j.

This problem can be mitigated by using larger scales since, for any given total number of missing pixels, it is less likely that larger square patches of the masked image are completely void of data. However, the SSBL Algorithm generally prefers to add basis functions with larger support to the model, since they typically cause a larger increase in the Marginal Likelihood. The result is that we get a more blurred reconstruction and lose the finer details, even in parts of the image that would have been accurately recovered at smaller scales.

5.2.3 Multi-Scale Cascade of Estimations Algorithm

The *Multi-Scale Cascade of Estimations* (MSCE) algorithm was developed by [9] to this tradeoff between inaccurate but complete reconstructions and accurate but incomplete reconstructions.

To explain the algorithm, note that, in the interpolation problem, the BCS reconstruction (5.3) is equivalent to computing the mean (4.27) of the predictive distribution (4.25) of the entire signal $\boldsymbol{v}$. Moreover, we can compute the predictive variance (4.28) (or error-bars) for each estimate:

$$(\sigma^2)^* = \sigma^2 + \boldsymbol{\psi}(\boldsymbol{x}^*)^T \boldsymbol{\Sigma} \boldsymbol{\psi}(\boldsymbol{x}). \tag{5.4}$$

The MSCE algorithm uses these error bars to construct a cascade of BCS algorithms, that builds up the scale of the Haar wavelets.

We begin by executing the BCS algorithm using Haar wavelets at the first scale. Next, we compute equation (5.4) for each location in the signal in which originally had missing data. If $(\sigma^2)^*$ is small, but larger than the noise variance σ^2, we trust the estimate and accept it. If $(\sigma^2)^*$ is large, we do not trust the estimate and reject it.

If $(\sigma^2)^*$ is equal to σ^2, then $\boldsymbol{\psi}(\boldsymbol{x}^*)^T \boldsymbol{\Sigma} \boldsymbol{\psi}(\boldsymbol{x}^*) = 0$. This means that $\boldsymbol{\psi}(\boldsymbol{x}^*) = 0$ since, by postive definiteness of $\boldsymbol{\Sigma}$, $\boldsymbol{\psi}(\boldsymbol{x}^*)^T \boldsymbol{\Sigma} \boldsymbol{\psi}(\boldsymbol{x}^*) > 0$ unless $\boldsymbol{\psi}(\boldsymbol{x}^*) = 0$. Thus, the RVM predicted the pixel value at location $\boldsymbol{x}^*$ to be zero. Since the recovery of the pixel was unsuccessful in this case, we reject the estimate.

The output of the current cascade becomes the target vector of the next cascade. The new basis matrix is constructed using Haar wavelets that are one scale higher than in the previous cascade.

Algorithm 3 Multi-Scale Cascade of Estimations [9]

Set $s = 1$
Let $\boldsymbol{y}_1 = \boldsymbol{y}$
while $s \leq s_{max}$ **do**
    Form basis matrix $\boldsymbol{\Psi}$ using Haar wavelets at scale s
    Call Sequential Sparse Bayesian Learning Algorithm with target vector $\boldsymbol{y}_j$ and design matrix $\boldsymbol{\Phi} = \boldsymbol{\Theta}\boldsymbol{\Psi}$ to get the estimate $\hat{\boldsymbol{v}}_s$
    Compute $(\sigma^2)^*$ for all newly reconstructed pixel values using equation (5.4)
    If $(\sigma^2)^* = \sigma^2$ or $(\sigma^2)^* > \tau$, discard the corresponding estimate
    Set $s = s + 1$
    Let $\boldsymbol{y}_s$ be $\hat{\boldsymbol{v}}_s$ after the discarded estimates are deleted
end while
return $\boldsymbol{v} = \boldsymbol{y}_s$

This process it repeated until we no more signal values marked as missing, or until some pre-defined maximum for the scale of the Haar basis is reached.

We have summarized the MCSE in Algorithm 3.

Chapter 6

Further Implementation Details

All the theoretical building blocks that underlie our implementation are now in place. We have developed a code that can efficiently reconstruct a video signal $\boldsymbol{v}$ from a set of CS measurements $\boldsymbol{y}$. In order to do so, it uses the BCS algorithm, representing the signal either in the DCT domain or in the Haar wavelet domain.

If the sensing mechanism is a signal mask, we can obtain an additional performance boost over the BCS by calling the MSCE algorithm.

Moreover, by treating images as single-frame videos, our algorithm can apply these techniques to provide efficient reconstruction of digital images as well as videos.

In this chapter, we will discuss some further implementation details and optimization strategies, that improve the memory requirements and execution time of out algorithm.

6.1 Blockwise Reconstruction

Let $\boldsymbol{V}$ be a video signal consisting of f frames with a width of w and a height h. Vectorization $\boldsymbol{V}$ gives us a vector $\boldsymbol{v}$ of length hwf. In order to reconstruct such signals in the BCS framework, we need to form the basis matrix $\boldsymbol{\Psi}$ which has dimensions $(hwf) \times (hwf)$.

Even for relatively small videos, the memory requirements can easily reach the order of terabytes. As an example, consider the commonly used *CIF (Common Intermediate Format)*. CIF videos have a spatial resolution of 352×288. For a CIF video containing 100 frames, the required memory for storing $\boldsymbol{\Psi}$ as in single-precision is

$$(288 \times 352 \times 100)^2 \times 4 \text{ bytes} = 4.11 \times 10^{14} \text{ bytes} = 411 \text{ TB}$$

In our code, we address the problem by performing a *blockwise reconstruction*. We split the input signal into small sub-blocks of size $2^{j_1} \times 2^{j_2} \times 2^{j_3}$. A typical size of such a block is $8 \times 8 \times 8$.

The blocks are sequentially passed to our algorithm and after each block has been reconstructed , we reassemble to obtained the recovered video.

Note that the size of the block restricts the number of cascades in the MSCE algorithm. To run the algorithm up to scale s, we require a block size of at least $2^s \times 2^s \times 2^s$.

6.2 Code Optimization

6.2.1 Choice of Programming Language

We implemented our algorithm in C++ as it is one of the most efficient programming languages. In order to boost the performance of matrix multiplications and inversions, we make use of the BLAS and LAPACK linear algebra libraries. To simulate the CS measurements, we use the random number generation facilities that were introduced into the C++11 standard.

6.2.2 Parallelization

In the blockwise reconstruction, each block is processed independently from the others. This allows for very large speedups by implementing the code as a distributed algorithm.

We have added a parallel mode to our implementation. Using the *Message Passing Interface* (MPI), we run the program on several processors and split the workload evenly between them. This generally leads to a very significant speedup.

However, if the blocksize is too small, the communication between processes (when gathering the results) will dominate over the actual computation time. Initial tests seem to indicate that, in order to achieve a significant increase in the execution time, the blocks should be at least of size $4 \times 4 \times 4$ before switching to parallel mode.

6.2.3 Fast Update Formulae

We chose to keep the noise variance σ^2 of the RVM fixed. As mentioned in Section 4.2.2, this lets us use the efficient update formulae in [15], greatly speeding up training times.

However, the downside to this is that we now have to set an additional model parameter. We tested range of values for σ^2, and setting $0.5 < \sigma^2 < 2$ gives consistently good performance.

6.2.4 Modified RVM Training

When training the RVM, we use a slightly modified version of SSBL algorithm in which we only consider additions of candidate basis functions and no deletions or re-estimations.

At each iteration, we add the basis vector $\boldsymbol{\phi}$ that results in the largest increase of the marginal likelihood. We continue to do so until none of the remaining candidate basis functions cause an increase in log marginal likelihood that is above the convergence threshold.

For the problem of image and video reconstruction, this modified algorithm gives qualitatively similar results to the unmodified version [9]. However, it can lead to a significant reduction in the runtime of the algorithm.

Chapter 7

Simulations

In this chapter we get to test our implementation.

We evaluate the reconstruction performance, we use the relative reconstruction error and the peak signal-to-noise ratio. We conduct four experiments in which we aim to compare the performance under different scenarios.

To close off with, we show some examples.

7.1 Performance Metrics

In this section, we will introduce three performance metrics that are often used to measure the quality of reconstructed images and videos.

Let $\hat{\boldsymbol{v}} \in \mathbb{R}^M$ be a reconstructed signal (in vectorized form) and let $\boldsymbol{v}$ be the corresponding orginal signal. The *mean square error* (MSE) of the reconstruction is defined as

$$\mathrm{MSE}(\hat{\boldsymbol{v}}) = \frac{\sum_{i=1}^{M}(\hat{v}_i - v_i)^2}{M}$$

The MSE is zero if and only if we $\hat{\boldsymbol{v}}$ is an exact reconstruction of $\boldsymbol{v}$.

Using the MSE, we can compute the *Peak Signal-to-Noise Ratio* (PSNR) of the reconstruction:

$$\mathrm{PSNR}(\hat{\boldsymbol{v}}) = 10 \cdot \log_{10}\left(\frac{R^2}{\mathrm{MSE}(\hat{\boldsymbol{v}})}\right)$$

where R is the maximum fluctuation in the input signal data type. For grayscale images or videos in which the pixel values are stored as 8-bit unsigned integers, we have that $R = 256$.

The PSNR is usually expressed in term of decibel (dB). Higher values of the PSNR correspond to more accurate reconstructions. The PSNR is widely used in the image and video compression literature to measure the quality of a compressed signal. Generally, when comparing the reconstruction quality, the PSNR should only be used if it was measured on the same signal.

The final performance metric that we compute is the *relative reconstruction error*. It is given by

$$\text{RRE}(\hat{\boldsymbol{v}}) = \frac{||\hat{\boldsymbol{v}} - \boldsymbol{v}||_2}{||\boldsymbol{v}||_2}$$

This is the metric that was used in [6] and [9].

7.2 Experiments

Four experiments were conducted that show what sort of performance one can expect under different settings. All simulations were run using the "foreman" test video.

For each experiment, we plot the performance against the number of CS measurements as a percentage of the signal size. Two plots are generated per experiment, one using the RRE and the other using the PSNR as performance measure.

Throughout the simulations, the block size is kept constant at $8 \times 8 \times 8$.

7.2.1 MSCE vs BCS

The results of the first experiment are shown in Figure 7.1. In the first experiment, we compare the performance of the MSCE algorithm to the BCS algorithm in the problem of video interpolation.

We run 3 instances of the BCS algorithm using Haar wavelet basis at scales 1, 2 and 3, respectively. We see that when the number of CS measurements N is small compared to the size of the signal M, a BCS using Haar wavelets at scale 1 performs very poorly. As N increases, its performance begins to accelarate and for $N > 0.5M$, the scale 1 Haar basis functions outperform the Haar bases at scales 2 and 3.

We should expect this behaviour from our analysis in Section 5.2.2. At low measurement rates, the BCS with level 1 Haar wavelets is unable to fill in the entire signal. However, at the portions of the signal where the reconstruction succeeds, it will generally be of higher quality than that of the BCS using wavelets at higher scales.

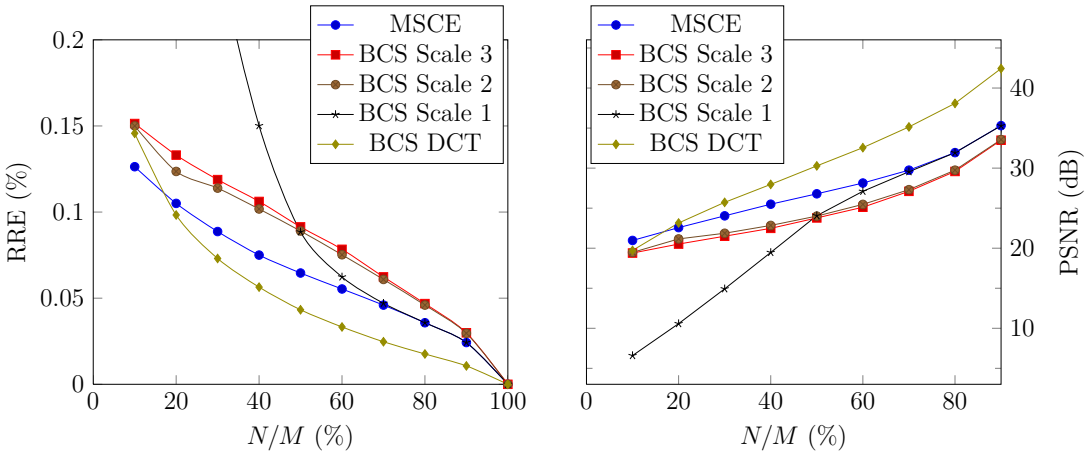


Fig. 7.1 Performance of MSCE with 3 cascades and uniform Masks

Next, we run the MSCE algorithm with 3 cascades. We can see immediately that a cascade of BCS algorithms is superior to any individual BCS instance that uses Haar wavelets.

The performances of the MSCE and the BCS using level 1 wavelets converge as the number of measurements M increases. That is because, for large M, the later stages of the cascade become redundant as Haar wavelets at scale 1 manage to cover any missing signal values.

Finally, we also run an instance of the BCS algorithm that uses DCT basis functions. We see that it easily outperforms even the cascade. Only in extremely undersampled situations ($N < 0.2M$) does the MSCE have an advantage over the BCS with DCT basis functions.

A possible explanation is that the sparsifying properties of the DCT are superior to those of the Haar wavelet DWT. Haar wavelets are the simplest of all wavelet functions. It would be interesting to compare these performances to that of a MSCE which uses more sophisticated wavelet functions, such as the CDF 9/7 wavelets.

7.2.2 DCT vs Haar with Gaussian Θ

In the second experiment we compare the performance of DCT and Haar basis functions in the case when $\boldsymbol{\Theta}$ consists of Gaussian samples.

We simulate Compressive Sensing measurements that were acquired via a random Gaussian sensing mechanism $\boldsymbol{\Theta}$.

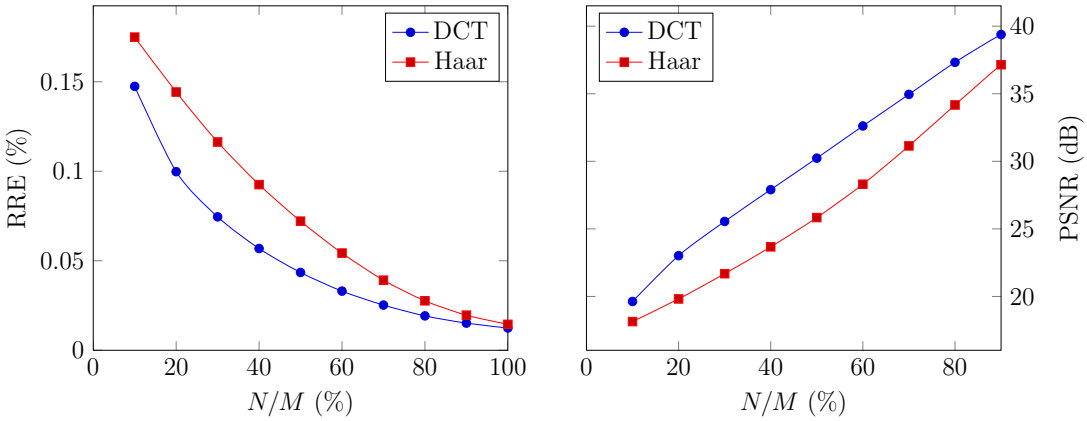


Fig. 7.2 DCT vs Haar DWT (scale 3) with Gaussian measurements

The reconstruction quality of the BCS with DCT basis functions and the BCS with level 3 Haar basis functions is shown Figure 7.2. In this case, we see that the DCT outperforms the Haar basis function at all compression levels.

7.2.3 Gaussian vs Bernoulli vs Random Signal Masks

The aim of the third experiment is to see how reconstruction differs if we change the sensing mechanism.

We simulate three sets of CS measurements using Gaussian, Bernoulli and random signal mask sensing matrices. The reconstruction is done using via the BCS algorithm and uses DCT basis functions.

Figure 7.3 shows the result of that experiment. We get near-identical performance in all three scenarios.

Only when the number of measurements is relatively high ($N \geq 0.8M$) do we begin to see a slight difference in the reconstruction quality.

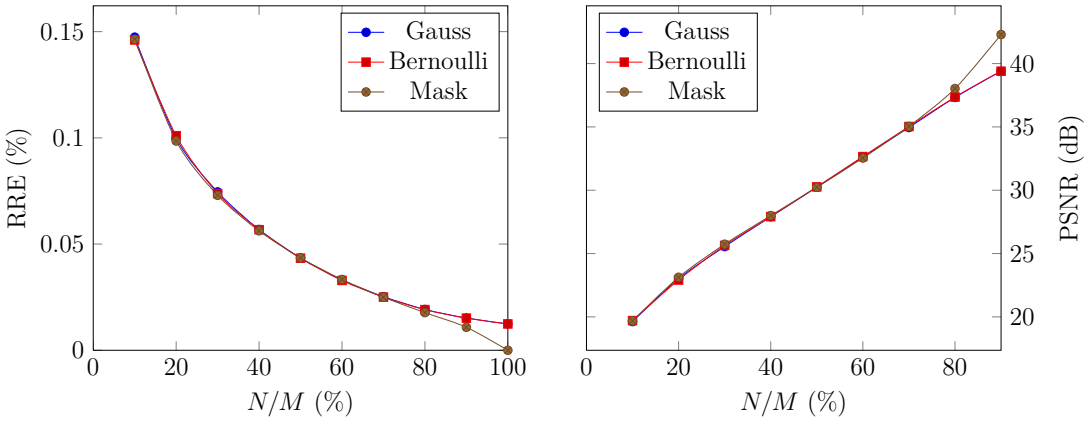
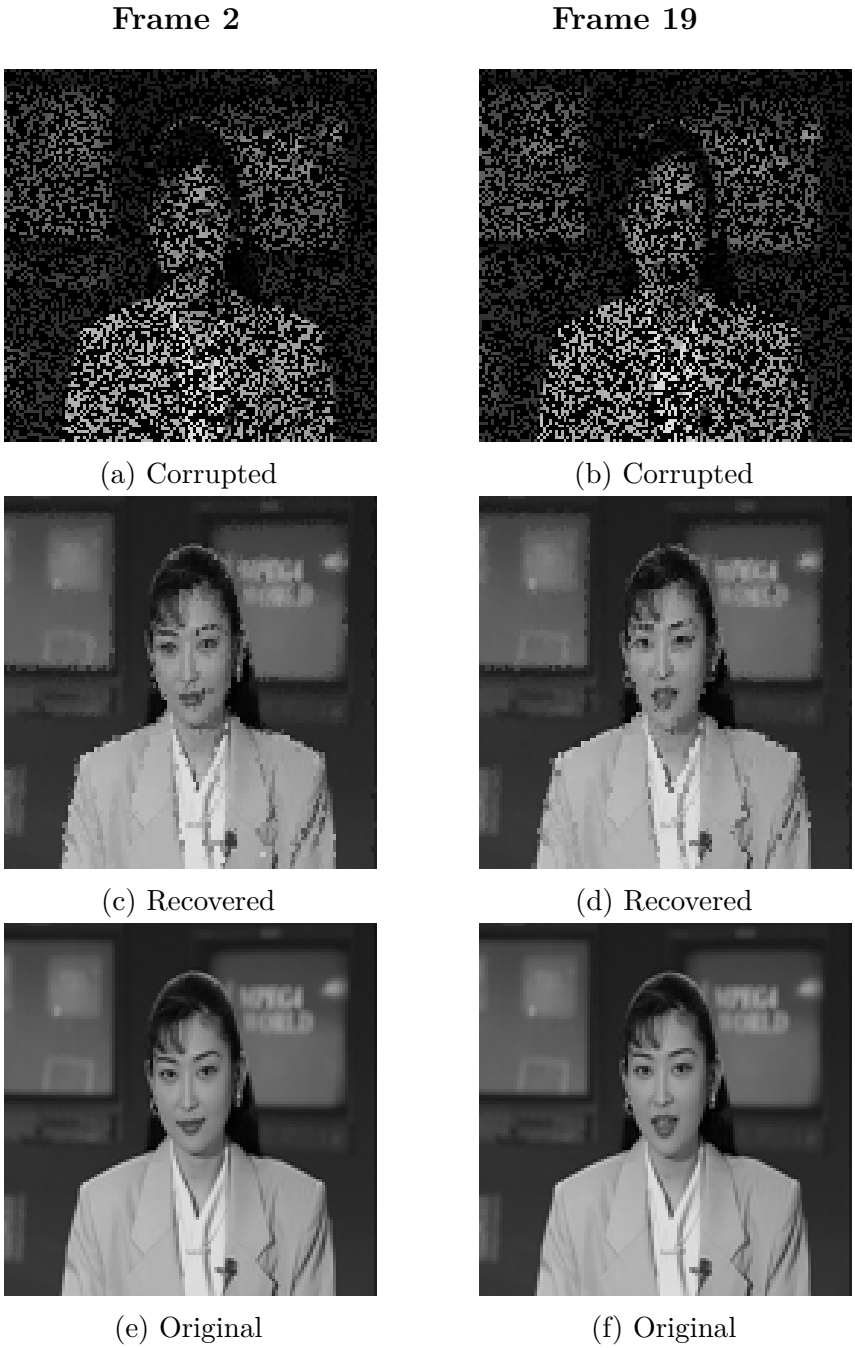


Fig. 7.3 Gaussian vs Bernoulli vs Mask (uniform) with DCT basis

7.3 Example Results



(a) Corrupted (b) Corrupted

(c) Recovered (d) Recovered

(e) Original (f) Original

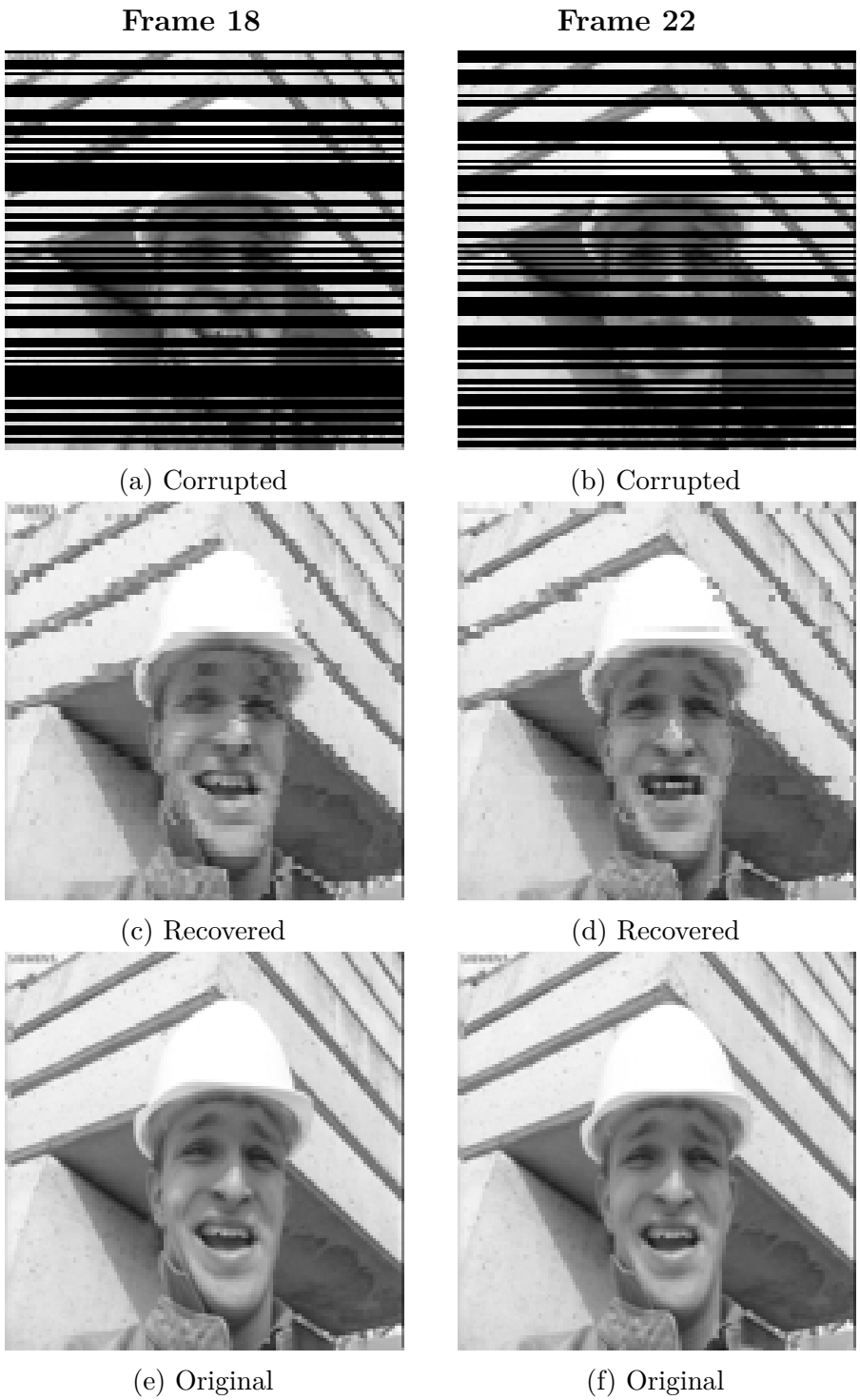
Fig. 7.4 Reconstruction of $128 \times 128 \times 128$ video signal "akiyo" from 40% of the measurement using the MSCE with 3 cascades. Mask decimation pattern is uniform. PSNR = 30.02

(a) Corrupted (b) Corrupted

(c) Recovered (d) Recovered

(e) Original (f) Original

Fig. 7.5 Reconstruction of $128 \times 128 \times 128$ video signal "akiyo" from 70% of the measurement using the MSCE with 3 cascades. Mask decimation pattern is uniform. PSNR = 36.86

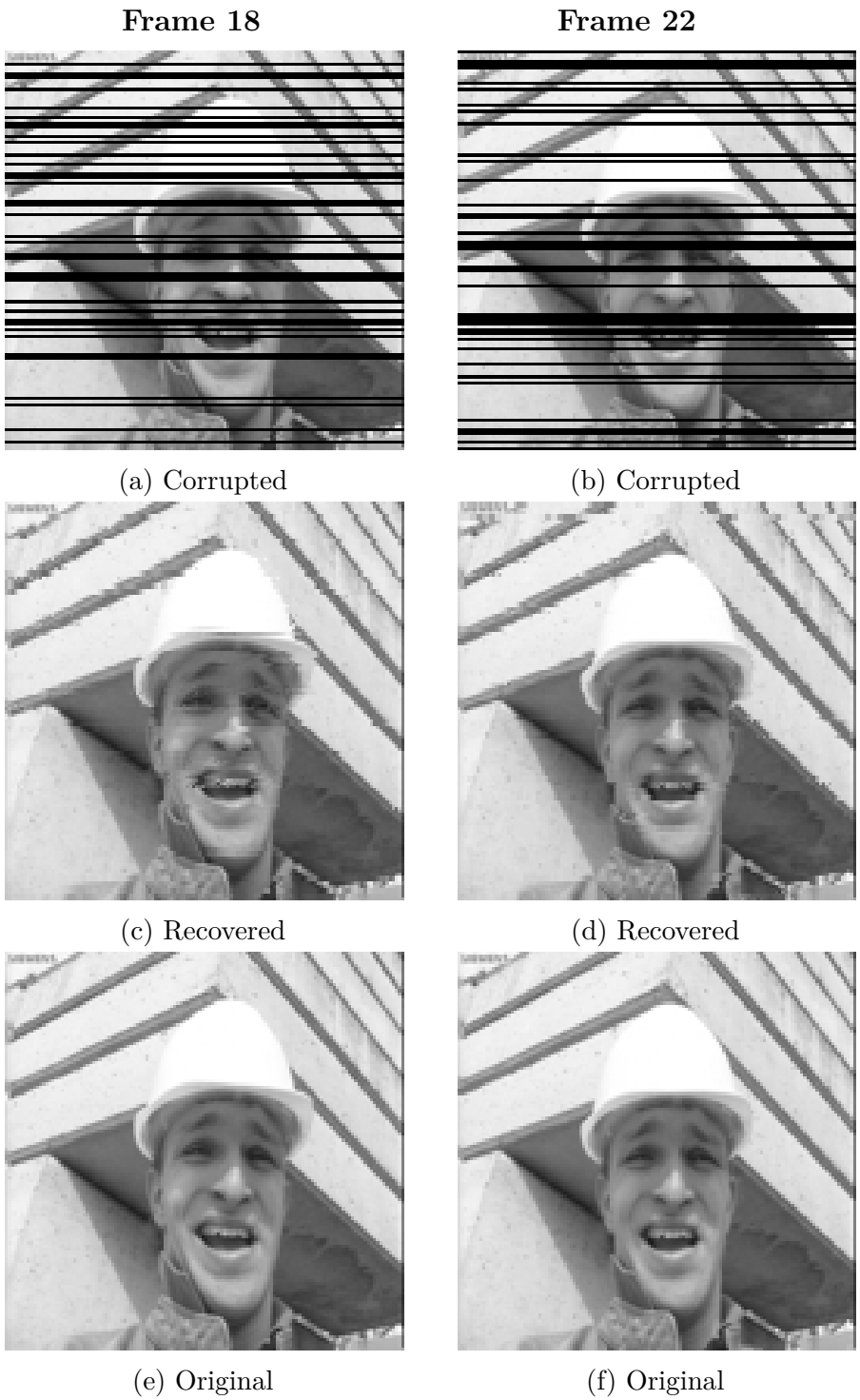


(a) Corrupted (b) Corrupted

(c) Recovered (d) Recovered

(e) Original (f) Original

Fig. 7.6 Reconstruction of $128 \times 128 \times 128$ video signal "foreman" from 40% of the measurement using the MSCE with 3 cascades. Mask decimation pattern is: horizontal lines that are randomly generated for each frame. PSNR = 25.50

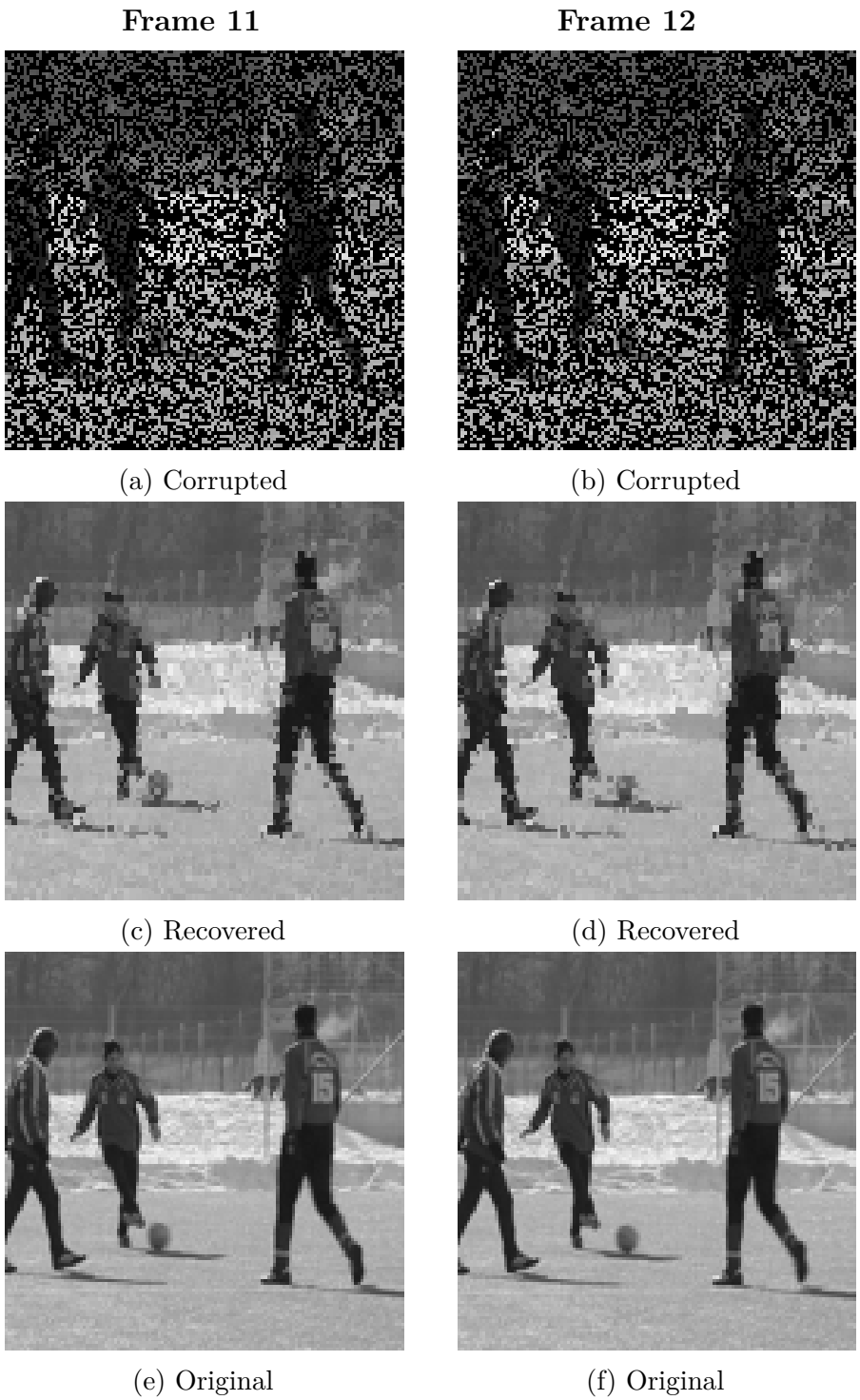


(a) Corrupted (b) Corrupted

(c) Recovered (d) Recovered

(e) Original (f) Original

Fig. 7.7 Reconstruction of $128 \times 128 \times 128$ video signal "foreman" from 70% of the measurement using the MSCE with 3 cascades. Mask decimation pattern is: horizontal lines that are randomly generated for each frame. PSNR = 30.32

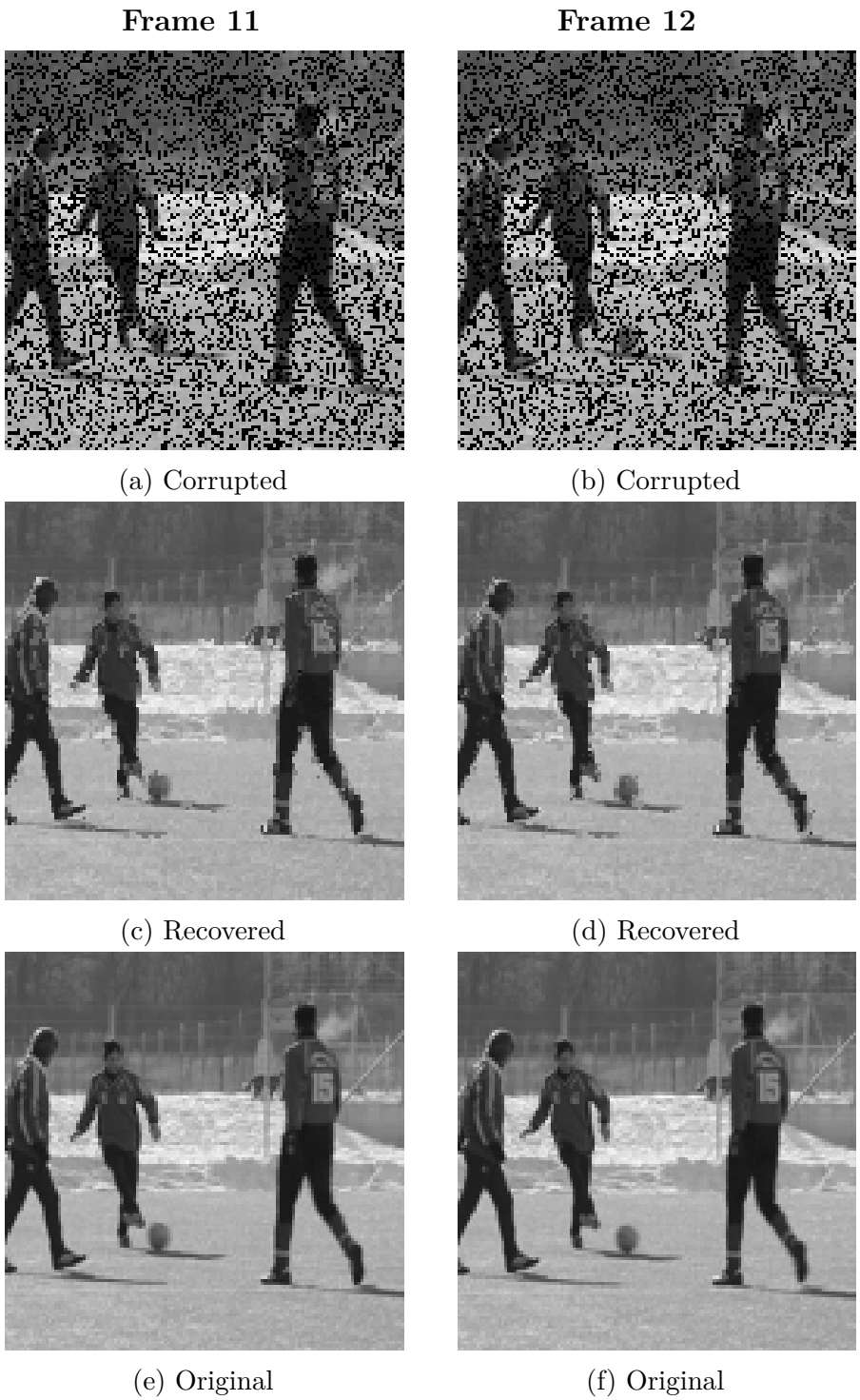


(a) Corrupted (b) Corrupted

(c) Recovered (d) Recovered

(e) Original (f) Original

Fig. 7.8 Reconstruction of $128 \times 128 \times 128$ video signal "soccer" from 40% of the measurement using the MSCE with 3 cascades. Mask decimation pattern is: missing pixels that are constant across the frames. PSNR = 24.84

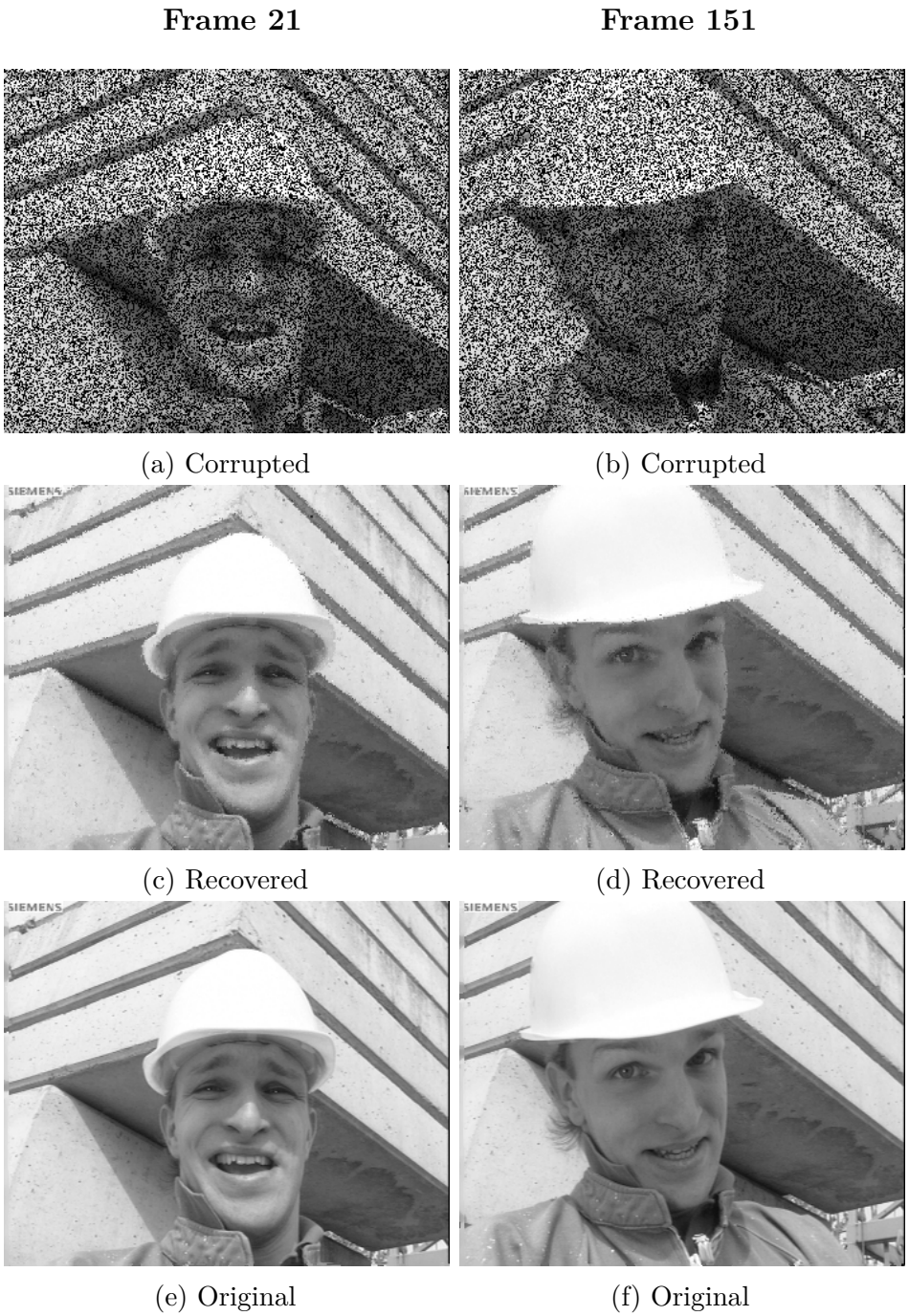


(a) Corrupted (b) Corrupted

(c) Recovered (d) Recovered

(e) Original (f) Original

Fig. 7.9 Reconstruction of $128 \times 128 \times 128$ video signal "soccer" from 70% of the measurement using the MSCE with 3 cascades. Mask decimation pattern is: missing pixels that are constant across the frames. PSNR = 29.35



(a) Corrupted (b) Corrupted

(c) Recovered (d) Recovered

(e) Original (f) Original

Fig. 7.10 Example of a masked video signal, where only 60% of the pixel values are known (a-b). Reconstruction via the MSCE Algorithm using 2 cascades (PSNR: 28.6) (c-d). Original video (e-f).

Chapter 8

Conclusion

For the MPhil thesis, we set out to develop a generic Compressive Sensing algorithm that provides high quality reconstructions of video signals from relatively small numbers of samples.

In this thesis, we investigated the efficacy of the Bayesian Compressive Sensing framework for efficient reconstruction of highly under-sampled video signals. We developed an extension to the Multi-Scale Cascade of Estimations algorithm that achieves near-perfect reconstructions of videos from a very small set of measurements.

To do so, we constructed three-dimensional wavelet basis functions that allow for a highly compressible representation of the video signal. Compressive Sensing inversion is then formulated as a machine learning problem and the Relevance Vector Machine was employed to find highly sparse solutions. To boost performance, a cascade of RVMs it built that exploits the multi-resolution properties of wavelet basis functions.

In order to deal with the large memory requirements of the algorithm, the reconstruction is performed in blockwise fashion. We have also implemented the method as a distributed program, resulting in dramatically reduced execution times.

Future research could improve performance by extending these methods in various ways. In order to fully harness the power of the MSCE for video interpolation, the implentation should be extended to accommodate different kinds of wavelets. Using the simple Haar wavelets, the MSCE struggles to outperform a BCT that uses DCT basis functions. It only shows its advantage in extremely undersampled situations ($N < 0.2M$). Alternative sets of waveles, such as the CDF-9/7 wavelet that is used by the JPEG2000 format, may lead to sparser representations of the video signals.

Furthermore, the speed of the algorithm can be increased by using a multi-threaded implementation of the Sequential Sparse Bayesian Learning Algorithm.

Development of Bayesian approaches to Compressive Sensing systems is an active area of research.

References

[1] Baraniuk, R. G. (2007). Compressive sensing. *IEEE signal processing magazine*, 24(4).

[2] Candès, E. J., Romberg, J., and Tao, T. (2006). Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509.

[3] Candès, E. J. and Wakin, M. B. (2008). An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30.

[4] Daubechies, I. et al. (1992). *Ten lectures on wavelets*, volume 61. SIAM.

[5] Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306.

[6] Ji, S., Xue, Y., and Carin, L. (2008). Bayesian compressive sensing. *IEEE Transactions on Signal Processing*, 56(6):2346–2356.

[7] Khayam, S. A. (2003). The discrete cosine transform (dct): theory and application. *Michigan State University*, 114.

[8] Mallat, S. (1999). *A wavelet tour of signal processing*. Academic press.

[9] Pilikos, G. (2014). Signal reconstruction using compressive sensing. MPhil thesis, University of Cambridge.

[10] Rasmussen, C. E. and Quinonero-Candela, J. (2005). Healing the relevance vector machine through augmentation. In *Proceedings of the 22nd international conference on Machine learning*, pages 689–696. ACM.

[11] Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21.

[12] Stollnitz, E. J., DeRose, T. D., and Salesin, D. H. (1995). Wavelets for computer graphics: a primer. 1. *Computer Graphics and Applications, IEEE*, 15(3):76–84.

[13] Tipping, A. and Faul, A. (2002). Analysis of sparse bayesian learning. *Advances in neural information processing systems*, 14:383–389.

[14] Tipping, M. E. (2001). Sparse bayesian learning and the relevance vector machine. *The journal of machine learning research*, 1:211–244.

[15] Tipping, M. E., Faul, A. C., et al. (2003). Fast marginal likelihood maximisation for sparse bayesian models. In *AISTATS*.

[16] Usevitch, B. E. (2001). A tutorial on modern lossy wavelet image compression: foundations of jpeg 2000. *IEEE signal processing magazine*, 18(5):22–35.

[17] Wallace, G. K. (1992). The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv.

[18] Zeng, J., Au, O. C., Dai, W., Kong, Y., Jia, L., and Zhu, W. (2013). A tutorial on image/video coding standards. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*, pages 1–7. IEEE.