ABSTRACT

Title of dissertation:	IMAGE REPRESENTATION AND COMPRESSION VIA SPARSE SOLUTIONS OF SYSTEMS OF LINEAR EQUATIONS
	Alfredo Nava Tudela, Doctor of Philosophy, 2012
Dissertation directed by:	Professor John J. Benedetto Department of Mathematics

We are interested in finding sparse solutions to systems of linear equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is underdetermined and fully-ranked. In this thesis we examine an implementation of the *orthogonal matching pursuit* (OMP) algorithm [11], an algorithm to find sparse solutions to equations like the one described above, and present a logic for its validation and corresponding validation protocol results. The implementation presented in this work improves on the performance reported in previously published work [2] that used software from SparseLab [19].

We also use and test OMP in the study of the compression properties of $\mathbf{A}$ in the context of image processing. We follow the common technique of image blocking used in the JPEG and JPEG 2000 standards [3, 22, 25]. We make a small modification in the stopping criteria of OMP that results in better compression ratio vs image quality as measured by the structural similarity (SSIM) and mean structural similarity (MSSIM) indices which capture perceptual image quality [26]. This results in slightly better compression than when using the more common peak

signal to noise ratio (PSNR) [30].

We study various matrices whose column vectors come from the concatenation of waveforms based on the discrete cosine transform (DCT), and the Haar wavelet. We try multiple linearization algorithms and characterize their performance with respect to compression.

An introduction and brief historical review on the topics of information theory, quantization and coding, and the theory of rate-distortion leads us to compute the distortion D properties of the image compression and representation approach presented in this work. A choice for a lossless encoder γ is left open for future work in order to obtain the complete characterization of the rate-distortion properties of the quantization/coding scheme proposed here. However, the analysis of natural image statistics is identified as a good design guideline for the eventual choice of γ. The lossless encoder γ is to be understood under the terms of a quantizer (α, γ, β) as introduced in [6].

IMAGE REPRESENTATION AND COMPRESSION VIA SPARSE SOLUTIONS OF SYSTEMS OF LINEAR EQUATIONS

by

Alfredo Nava Tudela

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2012

Preface

During her excellent Scientific Computing course in Spring 2010, Professor Dianne P. O'Leary offered a variety of articles on which to base our final course projects. Among them was an article from *SIAM Review* by Alfred M. Bruckstein, David L. Donoho, and Michael Elad [2] that caught my attention and on which I based my work. However, it was not until the Advanced Scientific Computing year-long, two-course sequence that followed that I fully developed what would become the core of this dissertation, further inspired by this article. During this period, Professors Radu Balan and Manuel Tiglio offered valuable input that shaped some of the results presented in this work.

Enthused by my discovery of the field of *Sparseland*, I talked to Professor John J. Benedetto, my long-time friend and advisor, about switching gears on what had been until then my dissertation work—trying to improve Magnetic Resonance Imaging—and instead exploring further image representation and compression by means of sparse solutions of systems of linear equations. Dr. Benedetto agreed to let me explore my newfound interest. As they commonly say, the rest is history. I am deeply indebted to all these wonderful professors. Especially to you John, who patiently over the years believed in me and, through our regular weekly meetings, saw me come to the end of this long and winding road.

I would also like to thank Professor Konstantina Trivisa, director of the Applied Mathematics & Statistics, and Scientific Computation Program at the University of Maryland, College Park, for persuading me to graduate while she was still director of the program. Her energy was a huge motivation to finish.

Dedication

To Kelly and Lucía. Thank you for your love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In recent years, interest has grown in the study of sparse solutions of underdetermined systems of linear equations because of their many and potential applications [2]. In particular, these types of solutions can be used to describe images in a compact form, provided one is willing to accept an imperfect representation.

We are interested in studying this approach to image compression because of the ever-increasing volume of images in use by many multimedia channels. The basic idea is the following. Suppose that we have a full-rank matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, where $n < m$, and that we want to find solutions to the equation,

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where $\mathbf{b}$ is a given "signal." Since the matrix $\mathbf{A}$ is full-rank, and we have more unknowns than equations, we have an infinite number of solutions to Equation 1.1. What if from all possible solutions we could find $\mathbf{x}_0$, the "sparsest" one, in the sense of having the least amount of nonzero entries? Then, if the number of nonzero entries in $\mathbf{x}_0$ happens to be less than the number of nonzero entries in $\mathbf{b}$, we could store $\mathbf{x}_0$ instead of $\mathbf{b}$, achieving a representation of the original signal $\mathbf{b}$ in a compressed way.

There are many questions that arise to this approach for image representation and compression. For example, is there a unique "sparsest" solution of Equation 1.1? How do you find such solutions? What are the practical implications for this approach to image compression?

We note that this idea can be framed in the context of signal transform compression techniques. For example, the JPEG and JPEG 2000 standards have at their core transformations that result in different representations of the original image which can be truncated to achieve compression at the expense of some acceptable error [3, 22, 25].

1.2 Finding sparse solutions

The *orthogonal matching pursuit* (OMP) algorithm [2, 11] is one of the existing techniques to find sparse solutions of systems of linear equations, as in Equation 1.1, where $\mathbf{A} \in \mathbb{R}^{n\times m}$ is a full-rank matrix, and $n < m$. This is one of many greedy algorithms that attempt to solve the general problem,

$$(P_0^\epsilon): \qquad \min_{\mathbf{x}} \|\mathbf{x}\|_0 \quad \text{subject to} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 < \epsilon. \tag{1.2}$$

Here, $\|\mathbf{x}\|_0 = \#\{j : |x_j| > 0\}$ is the "zero-norm" of vector $\mathbf{x}$, which counts the number of nonzero entries in $\mathbf{x}$. A greedy algorithm approach is necessary in the solution of the optimization problem defined in Equation 1.2 because this is an NP-complete problem [12]. Moreover, it can be proven that under certain circumstances there is a unique sparsest solution to (P_0^ϵ); and, under those same circumstances, OMP is then guaranteed to find it [2].

We have implemented OMP using a QR matrix decomposition [20] in one of the steps of the algorithm in a way that takes advantage of calculations performed in previous steps, resulting in a significant speedup. For details, see Section 2.2. Our implementation improves on the convergence to the sparsest solution of Equation 1.2 as compared to results previously published and obtained using software from [19], see Section 2.4. Moreover, compared to our initial naive implementation of OMP as defined in [2], we modified certain aspects of the algorithm that resulted in further speedup.

Armed with a tested and validated implementation of OMP (Section 2.3), we proceeded to study image representation and compression as explained before.

1.3 Image representation and compression

To make a practical implementation of the compression technique described in the introduction, we looked for inspiration in the JPEG and JPEG 2000 standards [25, 22]. Also, in order to test our image representation and compression approach, we selected a set of five commonly used test images, four of them from [24].

All images in our image database are 512 by 512, 8-bit depth, grayscale images. We proceeded to partition each image in subsets of 8 by 8 non-overlapping sub-images to process them individually. Partitioning a signal to work with more manageable pieces is a common technique [29].

Subsequently, we vectorize each 8 by 8 sub-image into a vector $\mathbf{b} \in \mathbb{R}^{64}$ to be used as a right hand side in Equation 1.1. There are many ways to do this vectorization and we explored three of them in detail, see Sections 3.4 and 3.5. To complete the setup, we needed a matrix $\mathbf{A}$.

We decided initially on $\mathbf{A} = [\mathrm{DCT}_1\ \mathrm{Haar}_1] \in \mathbb{R}^{64\times 128}$, where DCT_1 represents a basis of one-dimensional discrete cosine transform waveforms, and Haar_1 a basis of Haar wavelets, respectively. See Section 3.2.1 for details. That is, we concatenated two bases of $\mathbb{R}^{64}$, where $\mathbf{b}$ lives. Our initial thought for choice of basis elements drew from one-dimensional waveforms given that we are trying to approximate a vector—a one-dimensional object—, however we also considered bases for $\mathbb{R}^{64}$ built from tensor products of the one-dimensional waveforms mentioned above to capture the two-dimensional nature of the underlying problem, the representation of an image, an inherently two-dimensional object. We constructed in specific matrices $\mathbf{A} = [\mathrm{DCT}_{2,j}\ \mathrm{Haar}_{2,j}]$ defined in Section 3.2.2.

We have compared the compression properties of $\mathbf{A} = [\mathrm{DCT}_1\ \mathrm{Haar}_1]$ to those of $\mathbf{B} = [\mathrm{DCT}_1]$ and $\mathbf{C} = [\mathrm{Haar}_1]$ that only use the one-dimensional discrete cosine transform waveforms, or the one-dimensional Haar wavelets, respectively, and found that combining bases results in a representation $\mathbf{x}_0$ for each sub-image that requires fewer nonzero entries. We found similar compression results for the two-dimensional bases. A comparison between the one-dimensional and two-dimensional concatenation of bases showed that up to a range of tolerance values ϵ between approximately 3 and higher the two-dimensional basis elements perform better than the one-dimensional ones. See Section 3.6.

Also, as part of our research we have measured the quality of the reconstruction from these compressed representations by way of the *peak signal-to-noise ratio* (PSNR) [22, 30], the *structural similarity index* (SSIM), and the *mean structural similarity index* (MSSIM) [26], all as functions of the tolerance ϵ chosen when solving (P_0^ϵ), see Sections 3.7 and 3.8. This led us to a novel modification of the termination criteria in the OMP algorithm in order to achieve a desired PSNR or MSSIM value for the resulting decompressed image. There is still some work left to do on how to modify OMP to optimize for a desired MSSIM value.

Even though this is a promising compression technique, the full potential of this approach to image representation and compression cannot be assessed until a bit-stream $\mathbf{c}$ is produced to be able to quantify the net compression ratio, among other criteria. That is, we still need to tackle the elements found in complete image compression standards that draw on the information-theoretical paradigms raised by the work of Shannon [16], that are incorporated into working JPEG and JPEG 2000 implementations, and in general information transmission/storage systems.

Along these lines, we gave a brief review and historical introduction to the topics of information theory, quantization and coding, and the theory of rate-distortion, see Chapter 4. The structure of the image compression approach that we have chosen is amenable to a fine computation of its distortion D properties, for details see Section 4.1

Finally, we offered a few new directions where future research can lead into in Chapter 5, and in that spirit we briefly explored variations on the matrices we used in the bulk of our work. For details see Section 3.9.

Chapter 2

Finding sparse solutions

2.1 Orthogonal Matching Pursuit

In the first half of this project, we are interested in implementing and validating one of the many greedy algorithms (GAs) that attempt to solve (P_0^ϵ). The general idea is as follows. Starting from $\mathbf{x}^0 = \mathbf{0}$, a greedy strategy iteratively constructs a k-term approximation $\mathbf{x}^k$ by maintaining a set of active columns—initially empty—and, at each stage, expanding that set by one additional column. The column chosen at each stage maximally reduces the residual ℓ^2 error in approximating $\mathbf{b}$ from the current set of active columns. After constructing an approximation including the new column, the residual error ℓ^2 is evaluated; if it now falls below a specified threshold, the algorithm terminates.

Orthogonal Matching Pursuit (OMP)—a GA for calculating the solution to (P_0^ϵ):

Task: Calculate the solution to (P_0^ϵ) : $\min_{\mathbf{x}} \ \|\mathbf{x}\|_0$ subject to $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 < \epsilon$.

Parameters: We are given the matrix $\mathbf{A}$, the vector $\mathbf{b}$, and the threshold ϵ_0.

Initialization: Initialize $k = 0$, and set

- The initial solution $\mathbf{x}^0 = \mathbf{0}$.

- The initial residual $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0 = \mathbf{b}$.

- The initial solution support $\mathcal{S}^0 = \mathit{Support}\{\mathbf{x}^0\} = \emptyset$.

Main Iteration: Increment k by 1 and perform the following steps:

- **Sweep:** Compute the errors $\epsilon(j) = \min_{z_j} \|z_j\mathbf{a}_j - \mathbf{r}^{k-1}\|_2^2$ for all j using the optimal choice $z_j^* = \mathbf{a}_j^T\mathbf{r}^{k-1}/\|\mathbf{a}_j\|_2^2$.

- **Update Support:** Find a minimizer j_0 of $\epsilon(j)$: $\forall j \notin \mathcal{S}^{k-1}$, $\epsilon(j_0) \leq \epsilon(j)$, and update $\mathcal{S}^k = \mathcal{S}^{k-1} \cup \{j_0\}$.

- **Update Provisional Solution:** Compute $\mathbf{x}^k$, the minimizer of $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ subject to $\mathit{Support}\{\mathbf{x}\} = \mathcal{S}^k$.

- **Update Residual:** Compute $\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k$.

- **Stopping Rule:** If $\|\mathbf{r}^k\|_2 < \epsilon_0$, stop. Otherwise, apply another iteration.

Output: The proposed solution is $\mathbf{x}^k$ obtained after k iterations.

This algorithm is known in the literature of signal processing by the name *orthogonal matching pursuit* (OMP), and this is the algorithm we have implemented and validated. OMP solves, in essence, (P_0^ϵ) for $\epsilon = \epsilon_0$, a given positive threshold. See Equation 1.2 in Section 1.2 for details.

2.2 An OMP implementation

For a given matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$, if the approximation delivered by OMP has k_0 zeros, the method requires $\mathcal{O}(k_0 mn)$ flops in general; this can be dramatically better than the exhaustive search, which requires $\mathcal{O}(nm^{k_0}k_0^2)$ flops.

2.2.1 Least-squares approximation by QR decomposition

We would like to make the following observations about the OMP algorithm described in Section 2.1. The step that updates the provisional solution seeks to minimize $\|\mathbf{Ax}-\mathbf{b}\|_2^2$, subject to *Support*$\{\mathbf{x}\} = \mathcal{S}^k$. This is equivalent to solving the least-squares approximation problem $\min_{\tilde{\mathbf{x}}} \|\mathbf{A}^{(k)}\tilde{\mathbf{x}} - \mathbf{b}\|_2^2$ for the matrix $\mathbf{A}^{(k)}$ that results from using only the k active columns of $\mathbf{A}$ defined by $\mathcal{S}^k$, and $\tilde{\mathbf{x}}$ is the vector in $\mathbb{R}^k$ whose i-th entry corresponds to the column of $\mathbf{A}$ that was chosen during the i-th iteration of the main loop. See Figure 2.1.

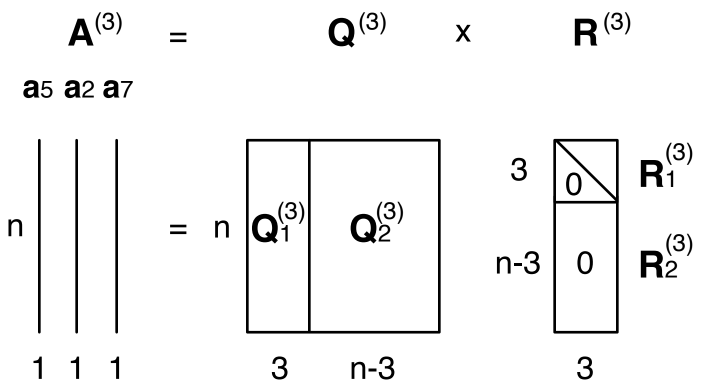


Figure 2.1: Suppose that after $k = 3$ iterations of the main loop, OMP has chosen, in the following order, columns $\mathbf{a}_5$, $\mathbf{a}_2$, and $\mathbf{a}_7$ from matrix $\mathbf{A}$. We form sub-matrix $\mathbf{A}^{(3)} = (\mathbf{a}_5\ \mathbf{a}_2\ \mathbf{a}_7)$, and its QR decomposition $\mathbf{A}^{(3)} = \mathbf{Q}^{(3)}\mathbf{R}^{(3)}$, which we use to solve the least-squares problem $\|\mathbf{A}^{(3)}\tilde{\mathbf{x}} - \mathbf{b}\|_2^2 = 0$, with $\tilde{\mathbf{x}} \in \mathbb{R}^3$.

For the case when $\mathbf{A}$ is a relatively small matrix, we can solve this problem, for example, by factorizing $\mathbf{A}^{(k)} = \mathbf{Q}^{(k)}\mathbf{R}^{(k)}$ with the QR-algorithm, and then observing that

$$\mathbf{A}^{(k)} = \mathbf{Q}^{(k)}\mathbf{R}^{(k)} = \mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)} + \mathbf{Q}_2^{(k)}\mathbf{R}_2^{(k)} = \mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)} + \mathbf{0} = \mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)}, \tag{2.1}$$

where—using Matlab notation—

$$\begin{aligned} \mathbf{Q}_1^{(k)} &= \mathbf{Q}^{(k)}(:, 1{:}k), \\ \mathbf{Q}_2^{(k)} &= \mathbf{Q}^{(k)}(:, k{+}1{:}n), \\ \mathbf{R}_1^{(k)} &= \mathbf{R}^{(k)}(1{:}k, :), \\ \text{and } \mathbf{R}_2^{(k)} &= \mathbf{R}^{(k)}(k{+}1{:}n, :). \end{aligned}$$

Then, from Equation 2.1, we have

$$\begin{aligned} \mathbf{A}^{(k)}\tilde{\mathbf{x}}_0 = \mathbf{b} &\Leftrightarrow \mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)}\tilde{\mathbf{x}}_0 = \mathbf{b} \\ &\Rightarrow \mathbf{Q}_1^{(k)\mathrm{T}}\mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)}\tilde{\mathbf{x}}_0 = \mathbf{Q}_1^{(k)\mathrm{T}}\mathbf{b} \\ &\Leftrightarrow \mathbf{R}_1^{(k)}\tilde{\mathbf{x}}_0 = \mathbf{Q}_1^{(k)\mathrm{T}}\mathbf{b} \\ &\Leftrightarrow \tilde{\mathbf{x}}_0 = (\mathbf{R}_1^{(k)})^{-1}\mathbf{Q}_1^{(k)\mathrm{T}}\mathbf{b}, \end{aligned}$$

where $\tilde{\mathbf{x}}_0 \in \mathbb{R}^k$ is the solution to the equivalent minimization problem described above, and the inverse of $\mathbf{R}_1^{(k)}$ exists because $\mathbf{A}^{(k)}$ is full-rank. Finally, when OMP returns successfully after k_0 iterations, we embed $\tilde{\mathbf{x}}_0 \in \mathbb{R}^{k_0}$ in $\mathbf{0} \in \mathbb{R}^m$ "naturally" to obtain the solution $\mathbf{x}_0 \in \mathbb{R}^m$ to the initial least-squares approximation problem $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ subject to the final active column set $\mathcal{S}^{k_0}$. The natural embedding refers to setting the j-th entry of $\mathbf{0} \in \mathbb{R}^m$ equal to the i-th entry in $\tilde{\mathbf{x}}_0 \in \mathbb{R}^{k_0}$ if during the i-th loop of the main algorithm, OMP chose the j-th column of $\mathbf{A}$.

2.2.2 Implementation fine tuning and speedup

We went through a series of code iterations to speedup our original implementation `ompQR`, initially done from a simplistic reading of the OMP algorithm described in Section 2.1. We also had a generic implementation and a couple of dedicated implementations. In Table 2.1 we show the speedup results for the generic version, which can take any full-rank matrix $\mathbf{A}$ as input. The dedicated implementations exploited the structure of known input matrices used during OMP testing with further speedup gains, as in resorting to the FFT as part of the internal calculations, for example.

Algorithm	Seconds	Speedup
`ompQR`	617.802467	—
`ompQRf`	360.192118	1.715
`ompQRf2`	308.379138	1.168
`ompQRf3`	298.622174	1.032

Table 2.1: Algorithm performance. The speedup column refers to the speedup from the immediately previous implementation. To compute the total speedup from first to last implementations multiply all speedup values together. Total speedup from `ompQR` to `ompQRf3` is 2.068, which means we doubled the speed of our implementation for the generic matrix version of our code. We used Matlab version R2010b Service Pack 1 to run "experiment.m" which performs many OMP calls on randomized input.

The first improvement came from computing $\|\mathbf{r}^k\||\cos(\theta_j)|$ during the *Sweep* portion of the algorithm. In this case θ_j is the angle between $\mathbf{a}_j$ and the residue

$\mathbf{r}_{k-1}$. This number reflects how good an approximation $z_j\mathbf{a}_j$ to the residue is, and it is faster to compute than $\epsilon(j)$. During this step we also kept track of the best approximation to the residue so that during the *Update Support* stage we could update $\mathcal{S}^k$ more efficiently as compared to what was done in `ompQR`. Finally, we do the sweep only on the set of columns that have not been added to the support set, resulting in further time gains on the *Sweep* stage whenever $k > 1$. All these changes where incorporated into `ompQRf`.

For the next round of improvements, we stop building $\mathbf{A}_k$ at each iteration as explained in Section 2.2.1. Rather, we initialize $\mathbf{Q} = \mathbf{I}_n$ and $\mathbf{R} = \emptyset$, where $\mathbf{I}_n \in \mathbb{R}^{n\times n}$ is the identity, and for consistency we define $\mathbf{A}_0 = \mathbf{I}_n \cdot \emptyset = \emptyset$. Subsequently, we update $\mathbf{Q}$ and $\mathbf{R}$ each time we add a column vector $\mathbf{a}_j$ of $\mathbf{A}$ in the following way. Suppose that at step $k > 0$ we have a QR decomposition of $\mathbf{A}_{k-1} = \mathbf{QR}$, and that column $\mathbf{a}_{j_k}$ is chosen from the *Update Support* step. Set $\mathbf{w} = (\mathbf{a}_{j_k}^{\mathrm{T}}\mathbf{Q})^{\mathrm{T}}$ and let $\mathbf{H}$ be a Householder reflexion such that $\mathbf{Hw} = \mathbf{v}$, where $\mathbf{v} = (\#, \ldots, \#, 0, \ldots, 0)^{\mathrm{T}}$ has $n-k$ zeros after the first k entries. Then, since $\mathbf{H}^{\mathrm{T}} = \mathbf{H}$, $\mathbf{H}^2 = \mathbf{I}_n$, and $\mathbf{HR} = \mathbf{R}$, it is easy to see that

$$\begin{aligned}\mathbf{QH}^{\mathrm{T}}\big(\mathbf{R}|\mathbf{H}^{\mathrm{T}}\mathbf{w}\big) &= \mathbf{Q}\big(\mathbf{H}^{\mathrm{T}}\mathbf{R}|\mathbf{H}^2\mathbf{w}\big) = \big(\mathbf{QR}|\mathbf{Qw}\big)\\ &= \big(\mathbf{A}_{k-1}|\mathbf{QQ}^{\mathrm{T}}\mathbf{a}_{j_k}\big) = \big(\mathbf{A}_{k-1}|\mathbf{a}_{j_k}\big) = \mathbf{A}_k.\end{aligned}$$

Therefore, if we set $\mathbf{Q}' = \mathbf{QH}^{\mathrm{T}}$, and $\mathbf{R}' = \big(\mathbf{R}|\mathbf{H}^{\mathrm{T}}\mathbf{w}\big)$, we would have found a QR decomposition of $\mathbf{A}_k = \mathbf{Q}'\mathbf{R}'$ as a function of $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{a}_{j_k}$. The implementation of this update results in faster code compared to the implementation that computes a QR decomposition of $\mathbf{A}_k$ from scratch for each k. This new approach was implemented in `ompQRf2`.

A final time improvement came simply from allocating all required variables as opposed to have them grow dynamically as needed. This was implemented in the final version `ompQRf3`.

2.3 OMP validation protocol and validation results

In this section we present the validation protocol that we followed to verify the correctness of our OMP implementation.

2.3.1 Theoretical results that motivate and justify the protocol

The following results provide the foundation for the validation protocol that we adopted. This protocol can be used to validate any OMP implementation.

Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ with $n < m$, we can compute its mutual coherence defined as follows [2].

Definition 1. *The mutual coherence of a given matrix* $\mathbf{A}$ *is the largest absolute normalized inner product between different columns from* $\mathbf{A}$*. Denoting the k-th column in* $\mathbf{A}$ *by* $\mathbf{a}_k$*, the mutual coherence is given by*

$$\mu(\mathbf{A}) = \max_{1 \leq k,j \leq m,\ k \neq j} \frac{|\mathbf{a}_k^T \mathbf{a}_j|}{\|\mathbf{a}_k\|_2 \cdot \|\mathbf{a}_j\|_2}. \tag{2.2}$$

The mutual coherence gives us a simple criterion by which we can test when a solution to Equation 1.1 is the unique sparsest solution available. In what follows, we assume that $\mathbf{A} \in \mathbb{R}^{n\times m}$, $n < m$, and $\operatorname{rank}(\mathbf{A}) = n$.

Lemma 1. *If $\mathbf{x}$ solves $\mathbf{A}\mathbf{x} = \mathbf{b}$, and $\|\mathbf{x}\|_0 < \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$, then $\mathbf{x}$ is the sparsest solution. That is, if $\mathbf{y} \neq \mathbf{x}$ also solves the equation, then $\|\mathbf{x}\|_0 < \|\mathbf{y}\|_0$.*

This same criterion can be used to test when OMP will find the sparsest solution.

Lemma 2. *For a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, if a solution $\mathbf{x}$ exists obeying $\|\mathbf{x}\|_0 < \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$, then an OMP run with threshold parameter $\epsilon_0 = 0$ is guaranteed to find $\mathbf{x}$ exactly.*

The proofs of these lemmas can be found or are inspired by results in [2]. In light of these lemmas, we can envision the following roadmap to validate an implementation of OMP. We have a simple unified theoretical criterion to guarantee both solution uniqueness and OMP convergence. The following theorem simply unifies the previous lemmas into one statement.

Theorem 3. *If $\mathbf{x}$ is a solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$, and $\|\mathbf{x}\|_0 < \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$, then $\mathbf{x}$ is the unique sparsest solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$, and OMP will find it.*

In light of this result, we can establish the following protocol to validate any implementation of OMP.

2.3.2 Validation protocol

Given a full-rank matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$, with $n < m$, compute $\mu(\mathbf{A})$, and find the largest integer k smaller than or equal to $\frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$. That is, find the integer

$k = \left\lfloor \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big) \right\rfloor$.

Then, build a vector $\mathbf{x}$ with exactly k nonzero entries and produce a right hand side vector $\mathbf{b} = \mathbf{A}\mathbf{x}$. This way, you have a known sparsest solution $\mathbf{x}$ to which to compare the output of any OMP implementation.

Pass $\mathbf{A}$, $\mathbf{b}$, and ϵ_0 to OMP to produce a solution vector $\mathbf{x}_{OMP} = \texttt{OMP}(\mathbf{A}, \mathbf{b}, \epsilon_0)$.

If OMP terminates after k iterations (or less), and $\|\mathbf{A}\mathbf{x}_{OMP} - \mathbf{b}\| < \epsilon_0$, for all possible $\mathbf{x}$ and $\epsilon_0 > 0$, then the OMP implementation would have been validated.

2.3.3 Validation results

Call $\kappa_{\mathbf{A}} = \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$, the constant dependent on $\mathbf{A}$ that guarantees the results of Theorem 3 for matrix $\mathbf{A}$. To test our implementation, we ran two experiments involving two random matrices.

1. $\mathbf{A}_1 \in \mathbb{R}^{100\times 200}$, with entries in the Gaussian distribution $N(0,1)$, i.i.d., for which its mutual coherence turned out to be $\mu(\mathbf{A}_1) = 0.3713$, corresponding to $k = 1 = \lfloor \kappa_{\mathbf{A}_1} \rfloor$.

2. $\mathbf{A}_2 \in \mathbb{R}^{200\times 400}$, with entries in the Gaussian distribution $N(0,1)$, i.i.d., for which its mutual coherence turned out to be $\mu(\mathbf{A}_2) = 0.3064$, corresponding to $k = 2 = \lfloor \kappa_{\mathbf{A}_2} \rfloor$.

We first note that, with probability 1, $\mathbf{A}_i$, $(i = 1, 2)$, is a full-rank matrix [2]. Second, we would like to mention that for full-rank matrices $\mathbf{A}$ of size $n \times m$, the mutual coherence satisfies $\mu(\mathbf{A}) \geq \sqrt{(m-n)/(n\cdot(m-1))}$, with the equality being sharp [21]. We used these results to guide us into obtaining matrix $\mathbf{A}_2$ for which

$k = 2 = \lfloor \kappa_{\mathbf{A}_2} \rfloor > 1$.

For each matrix $\mathbf{A}_i$, $(i = 1, 2)$, we chose 100 compatible vectors with k nonzero entries whose positions were chosen at random, and whose entries were in the Gaussian distribution $N(0, 1)$, i.i.d..

Then, for each such vector $\mathbf{x}$, we built a corresponding right hand side vector $\mathbf{b} = \mathbf{A}_i\mathbf{x}$. Each of these vectors would then be the unique sparsest solution to $\mathbf{A}_i\mathbf{x} = \mathbf{b}$, and OMP should be able to find them.

Finally, given $\epsilon_0 > 0$, if our implementation of OMP were correct, it should stop after k steps (or less), and if $\mathbf{x}_{OMP} = \texttt{OMP}(\mathbf{A}_i, \mathbf{b}, \epsilon_0)$, then $\|\mathbf{b} - \mathbf{A}_i\mathbf{x}_{OMP}\|_2 < \epsilon_0$.

We ran these experiments for twelve values of ϵ_0 equal to 10, 1, 10^{-1}, 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}, 10^{-14}, 10^{-15}, and 10^{-16}. For each of these values of ϵ_0 we built 100 vectors as described above, with their respective right hand side vectors, both of which were fed to OMP together with the tolerance ϵ_0 being tested.

We kept track of how many iterations it took OMP to stop, and the value of the norm of the residue $\|\mathbf{b} - \mathbf{A}_i\mathbf{x}_{OMP}\|_2$ at the end of each run. We mention that our implementation of OMP had as stopping condition that either the residue would be less than the tolerance ϵ_0 given, or that n iterations of the main loop would have been executed.

Figure 2.2 shows the summary of the results for matrix $\mathbf{A}_1$. It contains two graphs, the top graph represents the average of the norm of the residue over the 100 experiments executed for a given tolerance, versus the 12 tolerances chosen. The red line represents the identity in this case. The second graph is the same but for the average number of iterations it took OMP to stop vs the tolerances chosen. The

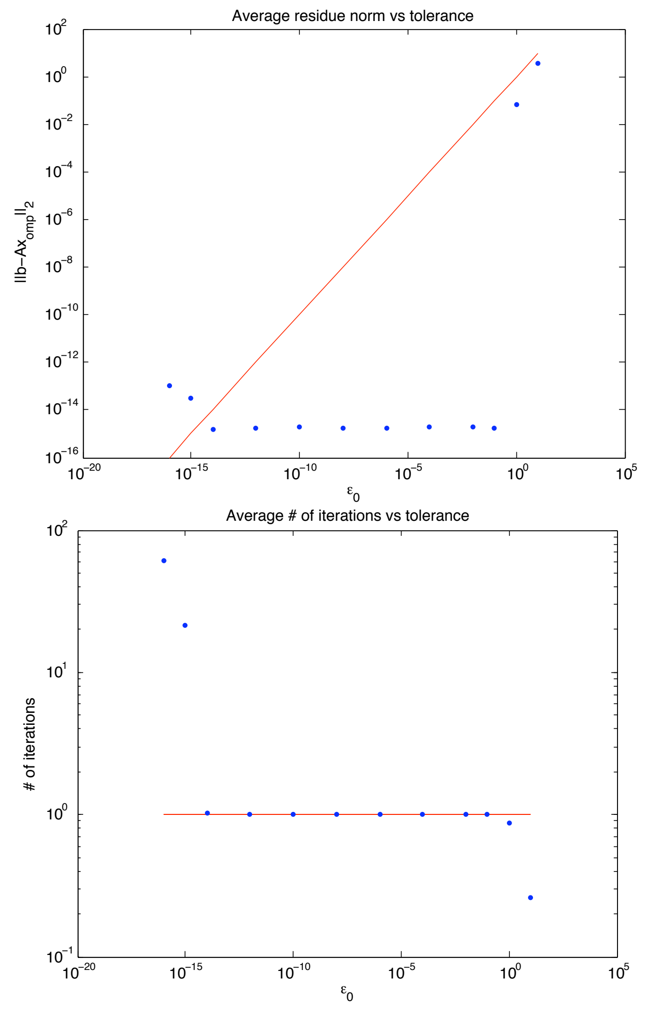


Figure 2.2: OMP behavior for a matrix $\mathbf{A}$ with $\mu(\mathbf{A}) = 0.3713$, which corresponds to $k_0 = 1$.

red line in this case is the expected number of iterations $k = \lfloor \kappa_{\mathbf{A}_1} \rfloor$ at stop time. Figure 2.3 is the same as Figure 2.2, but for matrix $\mathbf{A}_2$.

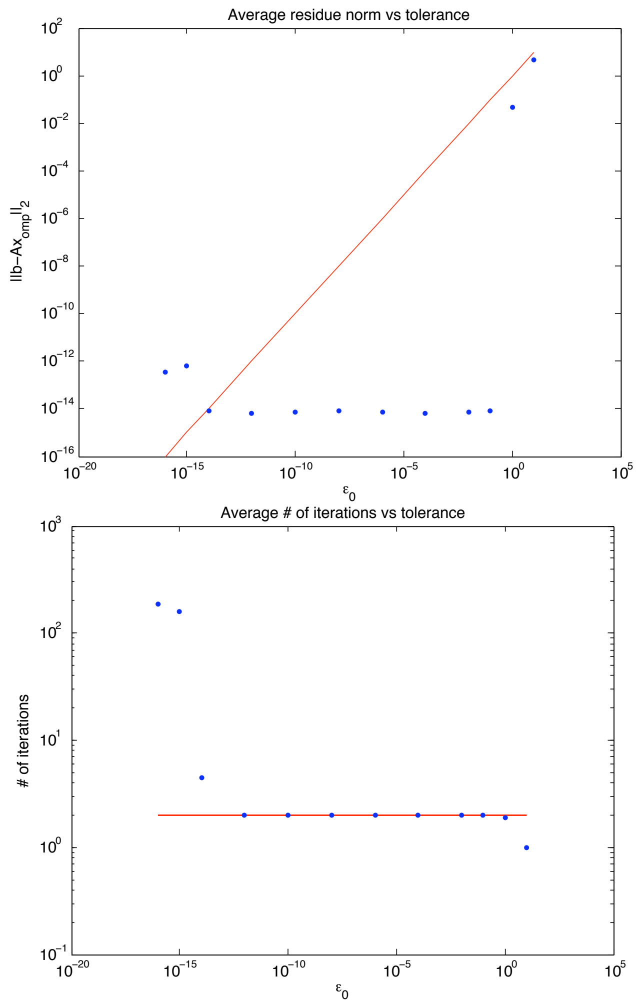


Figure 2.3: OMP behavior for a matrix $\mathbf{A}$ with $\mu(\mathbf{A}) = 0.3064$, which corresponds to $k_0 = 2$.

One can observe in both cases that there are three modal behaviors of OMP. The rightmost points in each graph correspond to tolerances ϵ_0 that are "too large". For them, OMP converges, but it does not have to do much work necessarily, since the default initial solution $\mathbf{x} = \mathbf{0}$ is already close to the right hand side $\mathbf{b}$. The typical behavior corresponds to points in the middle of the graph, they represent the cases when OMP converges in exactly k iterations to the sparsest solution within machine precision. And, finally, the leftmost points represent when OMP fails to converge because the tolerances ϵ_0 are too close to machine precision, basically trampling OMP efforts to converge due to roundoff and truncation errors.

In Figure 2.4 we exemplified each of the three modal behaviors with three values of ϵ_0 typical of each mode. The figure contains three graphs, the top graph is for $\epsilon_0 = 10$, the middle graph is for $\epsilon_0 = 10^{-6}$, and the bottom graph is for $\epsilon_0 = 10^{-16}$. Each of the graphs shows the individual results for each of the 100 experiments run for each tolerance ϵ_0. This figure corresponds to matrix $\mathbf{A}_1$. In Figure 2.5 we have the same graphs but for matrix $\mathbf{A}_2$.

We can conclude then that the validation protocol and results confirm that our implementation of OMP is correct. This implementation will return a solution $\mathbf{x} = \mathtt{OMP}(\mathbf{A}, \mathbf{b}, \epsilon_0)$ to $\mathbf{Ax} = \mathbf{b}$, within machine precision, whenever the tolerance $\epsilon_0 \geq 10^{-14}$, and provided $\|\mathbf{x}\|_0 \leq \kappa_{\mathbf{A}}$.

2.4 OMP testing protocol and results

For the first part of our testing protocol, we set out to reproduce a portion of an experiment described in [2]. The second part deals with studying the image compression properties of multiple matrices $\mathbf{A}$ which will be described in more detail in Chapter 3.

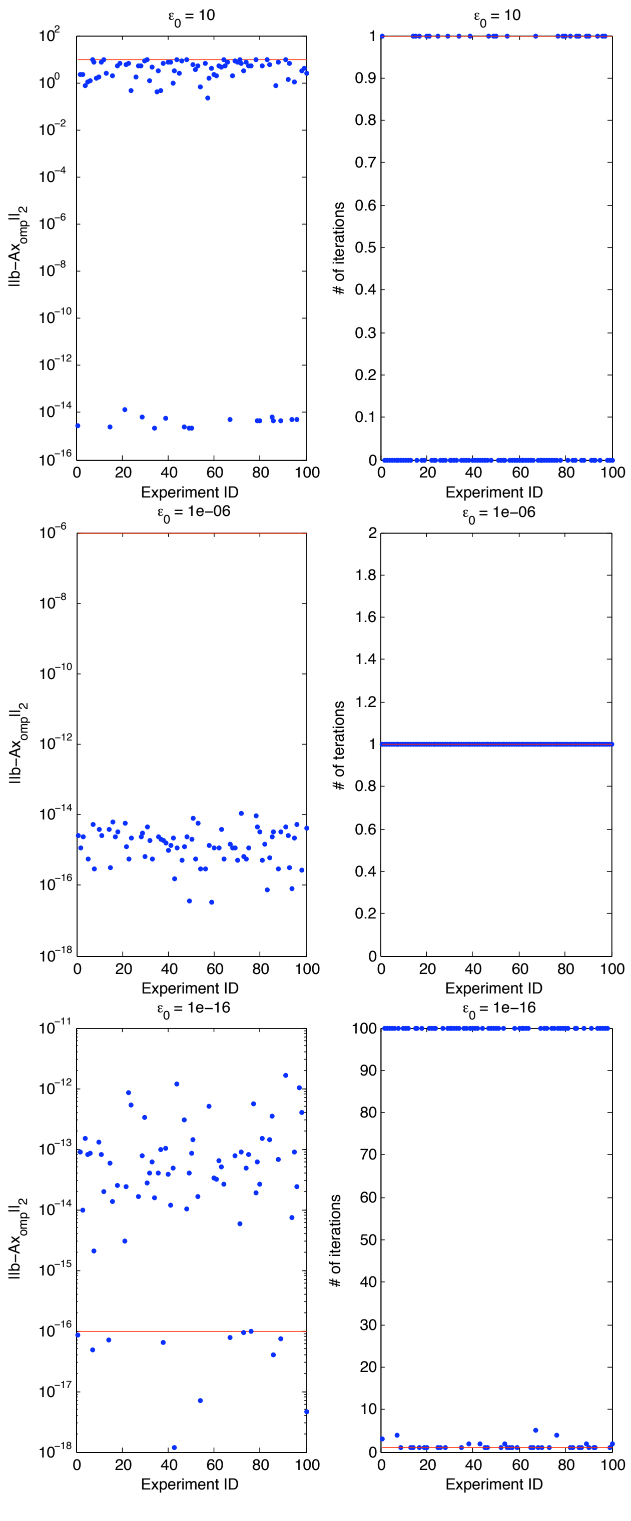


Figure 2.4: The three modal behaviors, dependent on ϵ_0, observed for the matrix $\mathbf{A}$ used in Figure 2.2.

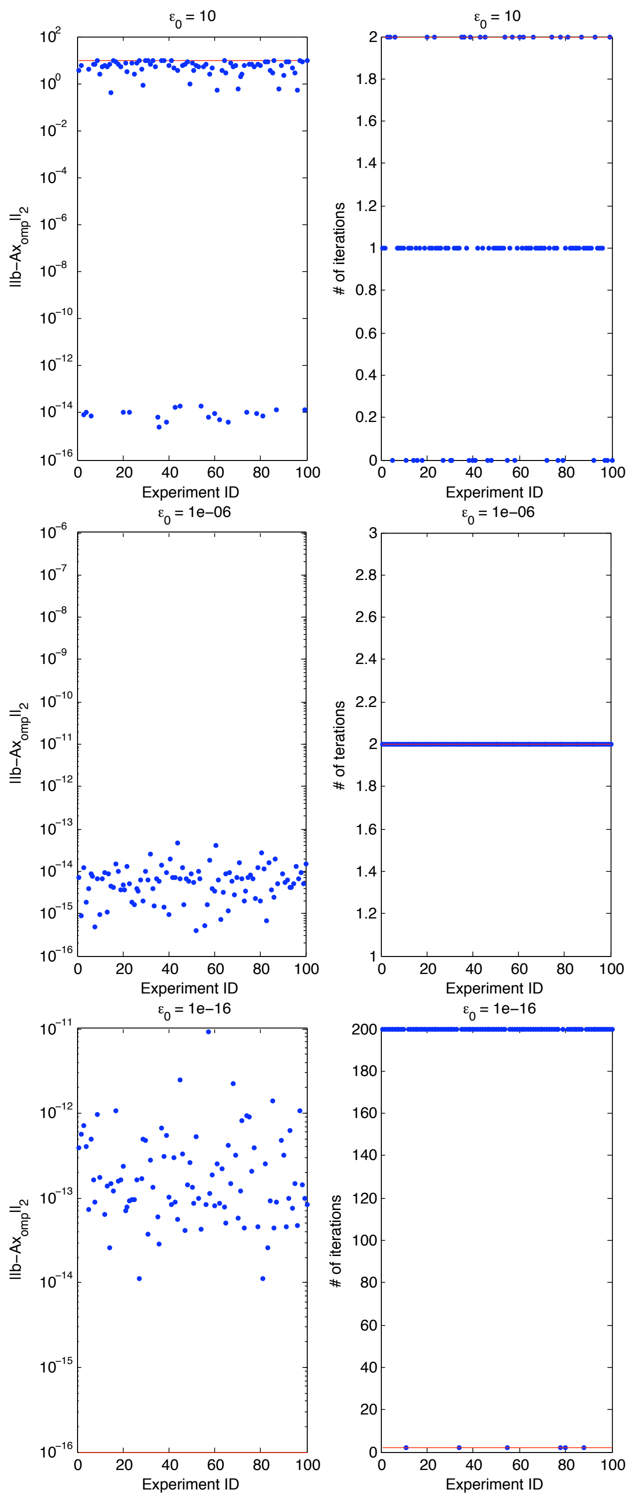


Figure 2.5: The three modal behaviors, dependent on ϵ_0, observed for the matrix $\mathbf{A}$ used in Figure 2.3.

2.4.1 Reproducing previous work

In a *SIAM Review* article by A. M. Bruckstein, D. L. Donoho, and M. Elad [2], the following experiment is presented. We set out to reproduce the portion corresponding to OMP.

Consider a random matrix $\mathbf{A}$ of size 100×200, with entries independently drawn at random from a Gaussian distribution of zero mean and unit variance, $\mathcal{N}(0,1)$. It can be proven that, with probability 1, every solution for the system $\mathbf{Ax} = \mathbf{b}$ with less than 51 entries is necessarily the sparsest one possible, and, as such, it is the solution of (P_0^ϵ), for any $\epsilon > 0$. By randomly generating such sufficiently sparse vectors $\mathbf{x}$ (choosing the nonzero locations uniformly over the support in a random way, and their values from $\mathcal{N}(0,1)$), we generate vectors $\mathbf{b}$. This way, we know the sparsest solution to $\mathbf{Ax} = \mathbf{b}$, and we shall be able to compare this to the results given by OMP.

Since we set to reproduce the results that pertain to OMP in Figure 2 of page 56 of [2], we considered cardinalities in the range of 1 to 70—even though we knew that, with probability 1, only those solutions with cardinality equal to or less than 51 were uniquely the sparsest ones possible—, and we conducted 100 repetitions and averaged the results to obtain the probability of the algorithm finding the solution with which we had generated the right hand side $\mathbf{b}$. Comparing our results with results obtained by published OMP implementations, e.g., like the ones available at SparseLab [19], Figure 2.6 shows that our implementation of OMP reproduces the published experiment, and it performs slightly better than the software found

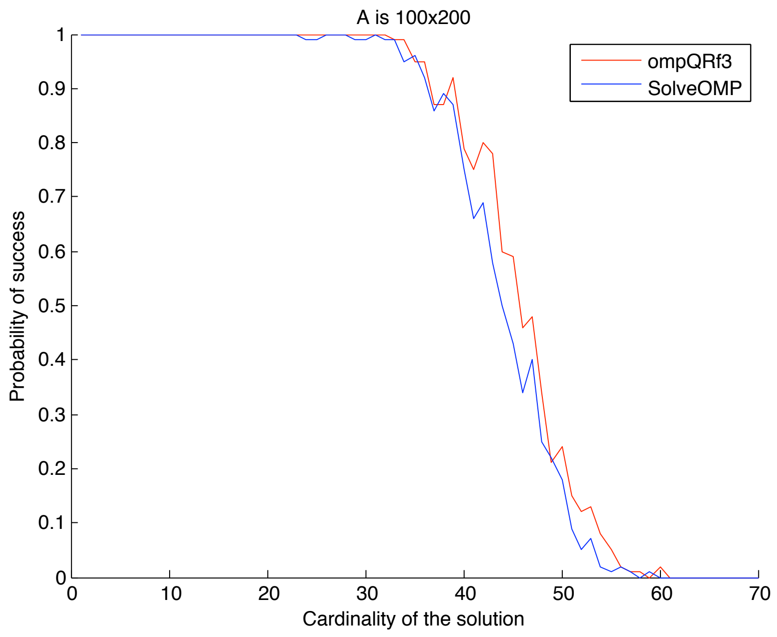


Figure 2.6: Reproduction of results in Section 3.3.1 of [2] for OMP. Our implementation `ompQRf3` is slightly better at recovering with higher probability the sparsest solution to $\mathbf{Ax} = \mathbf{b}$ when compared to `SolveOMP`, an implementation publicly available at [19].

in [19].

Chapter 3

Image representation and compression

Image compression plays a central role in modern multimedia communications. Compressed images arguably represent the dominant source of the Internet traffic today, and multiple applications ranging from medical records, to publishing—both in print and online media—, to military imagery, use them.

3.1 Elementary image representation concepts

For our purposes an image is a two dimensional sequence of sample values,

$$\mathcal{I}[n_1, n_2], \qquad 0 \leq n_1 < N_1, \qquad 0 \leq n_2 < N_2,$$

having finite extents, N_1 and N_2, in the vertical and horizontal directions, respectively. The term "pixel" is synonymous with an image sample. The first coordinate, n_1 is understood as the row index, while the second coordinate, n_2, is identified as the column index of the sample or pixel. The ordering of the pixels follows the canonical ordering of a matrix's rows and columns [22].

The sample value, $\mathcal{I}[n_1, n_2]$, represents the intensity (brightness) of the image at location $[n_1, n_2]$. The sample values will usually be B-bit signed or unsigned integers. Thus,

$$\mathcal{I}[n_1, n_2] \in \{0, 1, \ldots, 2^B - 1\} \quad \text{for unsigned imagery,}$$

$$\mathcal{I}[n_1, n_2] \in \{-2^{B-1}, -2^{B-1} + 1, \ldots, 2^{B-1} - 1\} \quad \text{for signed imagery.}$$

In many cases, the B-bit sample values are best interpreted as uniformly quantized representations of real-valued quantities, $\mathcal{I}'[n_1, n_2]$, in the range 0 to 1 (unsigned) or $-\frac{1}{2}$ to $\frac{1}{2}$ (signed). Letting $round(\cdot)$ denote rounding to the nearest integer, the relationship between the real-valued and integer sample values may be written as

$$\mathcal{I}[n_1, n_2] = round(2^B \mathcal{I}'[n_1, n_2]). \tag{3.1}$$

This accounts for the sampling quantization error which is introduced by rounding the physically measured brightness at location $[n_1, n_2]$ on a light sensor to one of the allowed pixel values [22, 29].

We will use this framework to represent grayscale images, where a pixel value of 0 will represent "black", and a value of $2^B - 1$ "white". The value of B is called the *depth* of the image, and typical values for B are 8, 10, 12 and 16. Color images are represented by either three values per sample, $\mathcal{I}_R[n_1, n_2]$, $\mathcal{I}_G[n_1, n_2]$, and $\mathcal{I}_B[n_1, n_2]$ each for the red, green, and blue channels, respectively; or by four values per pixel $\mathcal{I}_C[n_1, n_2]$, $\mathcal{I}_M[n_1, n_2]$, $\mathcal{I}_Y[n_1, n_2]$, and $\mathcal{I}_K[n_1, n_2]$, each for the cyan, magenta, yellow, and black channels commonly used in applications for color printing, respectively. We will restrict ourselves to grayscale images given that it is always possible to apply a compression system separately to each component in turn [22].

3.2 Image compression via sparsity

In this section, we give an overview of image compression via sparsity. The basic idea is that if $\mathbf{Ax} = \mathbf{b}$, $\mathbf{b}$ is dense—that is, it has mostly nonzero entries—, and $\mathbf{x}$ is sparse, we can achieve compression by storing wisely $\mathbf{x}$ instead of $\mathbf{b}$.

In specific, suppose we have a signal $\mathbf{b} \in \mathbb{R}^n$ that usually requires a description by n numbers. However, suppose that we can solve problem (P_0^ϵ), which we now recall from Section 1.2, viz.,

$$(P_0^\epsilon): \qquad \min_{\mathbf{x}} \|\mathbf{x}\|_0 \quad \text{subject to} \quad \|\mathbf{Ax} - \mathbf{b}\|_2 < \epsilon,$$

and whose solution $\mathbf{x}_0^\epsilon$ has k nonzeros, with $k \ll n$, then we would have obtained an approximation $\hat{\mathbf{b}} = \mathbf{Ax}_\mathbf{0}^\epsilon$ to $\mathbf{b}$ using k scalars, with an approximation error of at most ϵ. Thus, by increasing ϵ we can obtain better compression at the expense of a larger approximation error. We will characterize this relationship between error and compression, or equivalently, error and bit-rate per sample, later on.

The choice of matrix $\mathbf{A}$ is clearly central to our approach to compression. Inspired by the JPEG and JPEG 2000 standards that use at their core the discrete cosine transform (DCT) and a wavelet transform [25, 22], respectively, we decided to incorporate both transforms in some capacity in our choices of matrix $\mathbf{A}$.

3.2.1 Choosing $\mathbf{A}$: "One-dimensional" basis elements

Given that the signal that we are going to process comes in the form of a vector $\mathbf{b}$, an inherently one-dimensional object, a first approach is to consider the one-dimensional DCT waveforms, and any one-dimensional wavelet basis for $L^2[0,1]$. For the choice of the wavelet basis we opt for the Haar wavelet plus its scaling function, the identity on [0 1].

More specifically, we know that the one-dimensional DCT-II transform [1, 13],

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right], \qquad k = 0,\dots,N-1, \tag{3.2}$$

has at its core a sampling of the function $f_{k,N}(x) = \cos\left(\pi\left(x+\frac{1}{2N}\right)k\right)$ on the regularly spaced set of points $\mathcal{S}(N) = \left\{s_i \in [0\ 1) : s_i = \frac{i}{N},\ i = 0,\dots,N-1\right\}$. With this, we can define the vector $\mathbf{w}_{k,N} = \sqrt{2}^{\,\mathrm{sgn}(k)} \cdot (f_{k,N}(s_0),\dots,f_{k,N}(s_{N-1}))^{\mathrm{T}} \in \mathbb{R}^N$, which we call generically a "DCT waveform (of wave number k, and length N)." In specific, we will use DCT waveforms with $N = 64$ for the one-dimensional compression approach. This is because we subdivide each image in our study database into collections of 8 by 8 non-overlapping sub-images, which are then transformed into vectors $\mathbf{b} \in \mathbb{R}^{64}$, and subsequently compressed, as described above. See Section 3.4. This collection of DCT waveforms is a basis for $\mathbb{R}^{64}$, and we arrange its elements column-wise in matrix form as $\mathrm{DCT}_1 = (\mathbf{w}_{0,64} \dots \mathbf{w}_{63,64}) \in \mathbb{R}^{64\times 64}$. Note that all column vectors of DCT_1 have the same ℓ^2 norm.

The corresponding basis of $\mathbb{R}^{64}$ based on the Haar wavelet is built in the following way. Consider the Haar wavelet's mother function [10, 14],

$$\psi(x) = \begin{cases} 1 & \text{if } 0 \le x < 1/2, \\ -1 & \text{if } 1/2 \le x < 1, \\ 0 & \text{otherwise,} \end{cases} \tag{3.3}$$

and its scaling function,

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

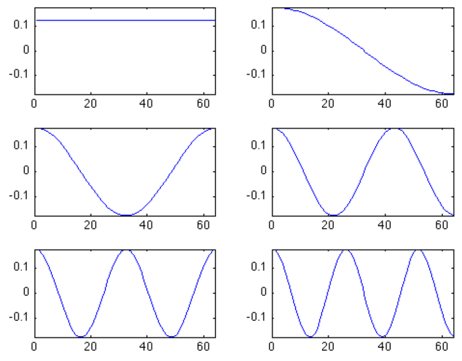
For a natural number $n \in \mathbb{N}$, build the set of functions

$$H_n = \{x \mapsto \psi_{n,k}(x) = 2^{n/2}\psi(2^n x - k) : 0 \le k < 2^n\},$$

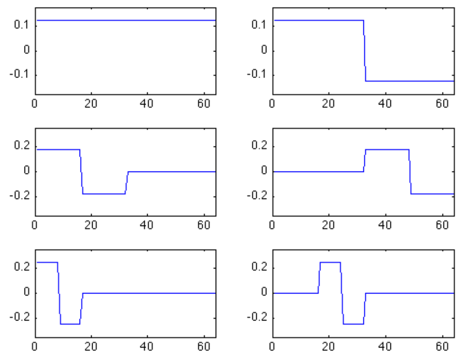
and define $H_{-1} = \{x \mapsto \phi(x)\}$. Note that $\#H_n = 2^n$ for $n \ge 0$, $\#H_{-1} = 1$, and therefore $\#\bigcup_{j=-1}^{n} H_j = 1 + \sum_{j=0}^{n} 2^j = 2^{n+1}$. Since $64 = 2^6$, a value of $n = 5$ will produce 64 functions to choose from in $\mathcal{H}(n) = \bigcup_{j=-1}^{n} H_j$. For each function $h \in \mathcal{H}(n = 5)$, create vector $\mathbf{v}_{h,64} \in \mathbb{R}^{64}$ by sampling h, as before, on the set of points $\mathcal{S}(N = 64)$. That is, $\mathbf{v}_{h,64} = (h(s_0), \ldots, h(s_{63}))^{\mathrm{T}}$. Observe that $\|\mathbf{v}_{h,64}\|_2 = \|\mathbf{w}_{0,64}\|_2$ for all $h \in \mathcal{H}(5)$. We drop the N in $\mathbf{v}_{\cdot,N}$ or $\mathbf{w}_{\cdot,N}$ when clear from the context.

Note that we can order the elements of $\mathcal{H}(5)$ in a natural way, namely, $h_0 = \phi$, $h_1 = \psi_{0,0}$, $h_2 = \psi_{1,0}$, $h_3 = \psi_{1,1}$, etc. It is easy to see that the set of vectors $\{\mathbf{v}_{h_j}\}_{j=0}^{63}$ is a basis of $\mathbb{R}^{64}$. As with the DCT waveforms, we arrange column-wise these vectors in matrix form as $\mathrm{Haar}_1 = (\mathbf{v}_{h_0} \ldots \mathbf{v}_{h_{63}}) \in \mathbb{R}^{64\times 64}$. Then, for the one-dimensional approach, we define $\mathbf{A} = [\mathrm{DCT}_1 \ \mathrm{Haar}_1] \in \mathbb{R}^{64\times128}$, the concatenation of both bases.

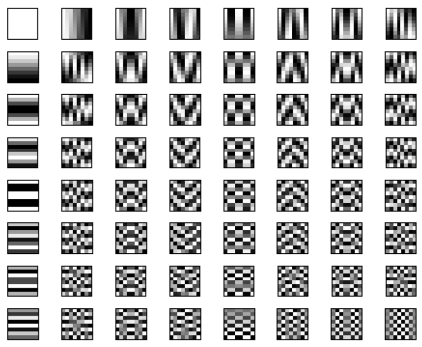
In Figure 3.1 we can visualize the columns of DCT_1 and Haar_1 in two different ways. A first method, given a column vector $\mathbf{u} = (u_1, \ldots, u_{64})^{\mathrm{T}}$ of either basis, is to plot the map $j \mapsto u_j$. A second method, slightly more elaborate, is to show the two-dimensional mapping of $\mathbf{u}$ to $c_2^{-1}(\mathbf{u})$, where the invertible map c_2 is defined in Section 3.4. We choose c_2 because the ordering of the entries of $\mathbf{u}$ induced by this invertible map results in optimal compression performance. See Table 3.2.



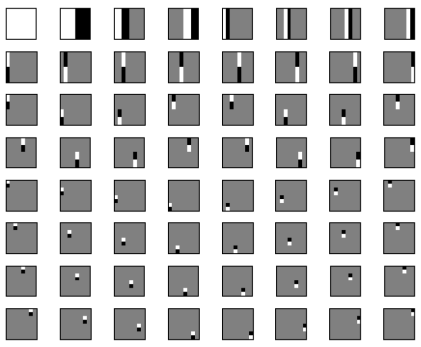
(a)



(b)



(c)



(d)

Figure 3.1: Top row: The first six waveforms of the DCT_1 (a), and the Haar_1 (b) normalized bases for $\mathbb{R}^{64}$. Bottom row: Full 2D representation using the inverse map of c_2 defined in Section 3.4 for the DCT_1 (c), and Haar_1 (d) bases for $\mathbb{R}^{64}$. White corresponds to the maximum value achieved by the basis element, black to the minimum. The intermediate shade of gray corresponds to 0.

3.2.2 Choosing $\mathbf{A}$: “Two-dimensional” basis elements

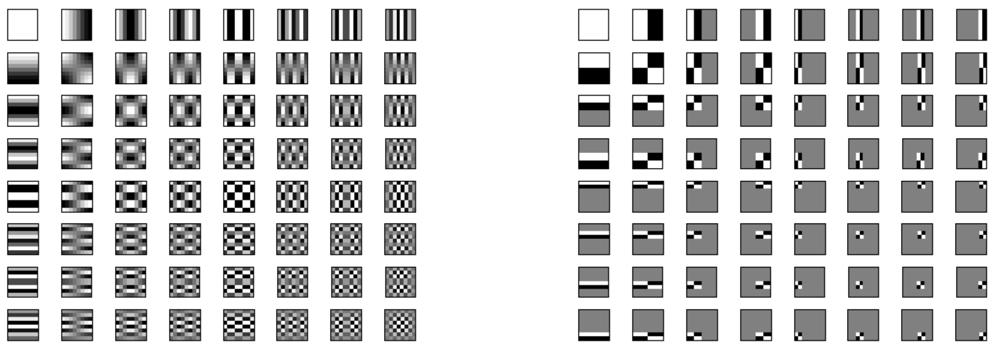
Another way to choose a basis for $\mathbb{R}^{64}$ results from taking into account the intrinsic two-dimensional nature of an image and create basis elements that reflect this fact. In specific, consider the two-dimesional DCT-II used in the JPEG standard [25],

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n1,n2} \cos\left[\frac{\pi}{N_1}\left(n_1 + \frac{1}{2}\right) k_1\right] \cos\left[\frac{\pi}{N_2}\left(n_1 + \frac{1}{2}\right) k_2\right], \qquad (3.5)$$

and, as before, consider the family of functions indexed by k_1 and k_2,

$$g_{k_1,k_2,N_1,N_2}(x,y) = \cos\left[\pi\left(x + \frac{1}{2N_1}\right) k_1\right] \cos\left[\pi\left(y + \frac{1}{2N_2}\right) k_2\right],$$

sampled on all points $(x,y) \in \mathcal{S}(N_1) \times \mathcal{S}(N_2)$, where in our case, $N_1 = N_2 = 8$, and $k_1, k_2 \in \{0, \ldots, 7\}$.



(a) (b)

Figure 3.2: Full natural 2D representation for the DCT$_2$ (a), and Haar$_2$ (b) bases for $\mathbb{R}^{64}$. White corresponds to the maximum value achieved by the basis element, black to the minimum. The intermediate shade of gray corresponds to 0.

Note that $g_{k_1,k_2,N_1,N_2}(x,y) = f_{k_1,N_1}(x)f_{k_2,N_2}(y)$, and therefore the image of $\mathcal{S}(8) \times \mathcal{S}(8)$ under $g_{k_1,k_2,8,8}$ can be naturally identified with the outer product

$$\mathbf{w}_{k_1,8} \otimes \mathbf{w}_{k_2,8}, \qquad k_1, k_2 = 0, \ldots, 7, \tag{3.6}$$

modulo the constants $\sqrt{2}^{\operatorname{sgn}(k_1)}$ and $\sqrt{2}^{\operatorname{sgn}(k_2)}$, that make $\|\mathbf{w}_{k_1}\|_2 = \|\mathbf{w}_{k_2}\|_2$, respectively. See Figure 3.2(a) for a graphical representation of each and all of these outer products.

There is a total of 64 such outer products, and for each of them we can obtain a vector $\widetilde{\mathbf{w}}_{k_1,k_2,j} = c_j^{-1}(\mathbf{w}_{k_1,8} \otimes \mathbf{w}_{k_2,8})$, where c_j^{-1} is the inverse map of any of the bijections c_1, c_2, or c_3 that take a vector into a matrix, defined in Section 3.4. It is easy to see that the vector columns of the matrix

$$\mathrm{DCT}_{2,j} = (\widetilde{\mathbf{w}}_{k_1,k_2,j}) \in \mathbb{R}^{64\times 64}, \qquad k_1, k_2 = 0, \ldots, 7, \tag{3.7}$$

form a basis for $\mathbb{R}^{64}$. The ordering of the column vectors of $\mathrm{DCT}_{2,j}$ follow the lexicographical order of the sequence of ordered pairs (k_1, k_2) for $k_1, k_2 = 0, \ldots, 7$ when k_2 moves faster than k_1, i.e., $(0,0), (0,1), \ldots, (0,7), (1,0), \ldots, (7,7)$.

In a similar fashion, we can build $\mathrm{Haar}_{2,j}$. In specific, consider first the set of functions $\mathcal{H}(2)$, which contains 8 functions. Sampling $h_k \in \mathcal{H}(2)$ on $\mathcal{S}(8)$, we obtain vector $\mathbf{v}_{h_k,8} \in \mathbb{R}^8$. Given $k_1, k_2 \in \{0, \ldots, 7\}$ we then compute vector

$$\widetilde{\mathbf{v}}_{k_1,k_2,j} = c_j^{-1}\left(\mathbf{v}_{h_{k_1},8} \otimes \mathbf{v}_{h_{k_2},8}\right) \in \mathbb{R}^{64},$$

with which, for all $k_1, k_2 \in \{0, \dots, 7\}$, and following the same order for the ordered pairs (k_1, k_2) mentioned above, we can create

$$\text{Haar}_{2,j} = (\widetilde{\mathbf{v}}_{k_1,k_2,j}) \in \mathbb{R}^{64\times 64}, \qquad k_1, k_2 = 0, \dots, 7. \tag{3.8}$$

For a visual representation of the outer products $\mathbf{v}_{h_{k_1},8} \otimes \mathbf{v}_{h_{k_2},8}$, see Figure 3.2(b).

Finally, we can define $\mathbf{A} = \mathbf{A}(j) = [\text{DCT}_{2,j}\ \text{Haar}_{2,j}] \in \mathbf{R}^{64\times 128}$, the concatenation of both bases.

3.2.3 Some properties of $[\text{DCT}_1\ \text{Haar}_1]$ and $[\text{DCT}_{2,j}\ \text{Haar}_{2,j}]$

From the definitions of DCT_1, Haar_1, $\text{DCT}_{2,j}$, and $\text{Haar}_{2,j}$, with $j = 1, 2,$ or 3 (from now on, we won't make explicitly clear that the index j runs through 1, 2, and 3 as we will assume that this is always the case when it appears,) it is easy to see that they are matrices whose column vectors are pairwise orthogonal and with a common norm, i.e., $\|\mathbf{a}\|_2 = c$ for any column vector $\mathbf{a}$ of any of these matrices. Also, it is easy to see in this case that $c = 8$. This means that if we were to consider $\frac{1}{8}\text{DCT}_1$, $\frac{1}{8}\text{Haar}_1$, $\frac{1}{8}\text{DCT}_{2,j}$, and $\frac{1}{8}\text{Haar}_{2,j}$, separately, they would all be orthogonal matrices, i.e., matrices whose columns are pairwise orthonormal. We present now some properties that are derived from these facts.

Lemma 4 (Parseval's identity). *Let U be an orthogonal basis for $\mathbb{R}^n$ such that $\forall\, \mathbf{u} \in U\ \|\mathbf{u}\|_2 = c$. Then $\forall\, \mathbf{w} \in \mathbb{R}^n$*

$$\sum_{\mathbf{u}\in U} (\mathbf{w}^{\mathrm{T}}\mathbf{u})^2 = c^2 \|\mathbf{w}\|_2^2. \tag{3.9}$$

Proof. Let $\mathbf{w} \in \mathbb{R}^n$. Since U is a basis for $\mathbb{R}^n$, there exists a linear combination of elements of $U = \{\mathbf{u}_k \in \mathbb{R}^n : k = 1, \ldots, n\}$ such that

$$\sum_k x_k \mathbf{u}_k = \mathbf{w}, \quad \text{where } x_k \in \mathbb{R}.$$

Let $j \in \{1, \ldots, n\}$, then we have that

$$\begin{aligned} \sum_k x_k \mathbf{u}_k = \mathbf{w} \quad &\Longrightarrow \quad \sum_k x_k \mathbf{u}_j^{\mathrm{T}} \mathbf{u}_k = \mathbf{u}_j^{\mathrm{T}} \mathbf{w}, \\ &\Leftrightarrow \quad x_j \|\mathbf{u}_j\|_2^2 = \mathbf{u}_j^{\mathrm{T}} \mathbf{w}, \text{ since } U \text{ is an orthogonal basis,} \\ &\Leftrightarrow \quad x_j = \frac{\mathbf{u}_j^{\mathrm{T}} \mathbf{w}}{\|\mathbf{u}_j\|_2^2}. \end{aligned}$$

Hence $\sum_k \frac{\mathbf{u}_k^{\mathrm{T}} \mathbf{w}}{\|\mathbf{u}_j\|_2^2} \mathbf{u}_k = \mathbf{w}$, but $\forall j \; \|\mathbf{u}_j\|_2 = c$ by assumption. Hence $\sum_k \mathbf{u}_k^{\mathrm{T}} \mathbf{w} \, \mathbf{u}_k = c^2 \mathbf{w}$. From this equation, premultiplying by $\mathbf{w}^{\mathrm{T}}$, we obtain that

$$\begin{aligned} \sum_k \mathbf{u}_k^{\mathrm{T}} \mathbf{w} \, \mathbf{u}_k = c^2 \mathbf{w} \quad &\Longrightarrow \quad \sum_k \mathbf{u}_k^{\mathrm{T}} \mathbf{w} \, \mathbf{w}^{\mathrm{T}} \mathbf{u}_k = c^2 \mathbf{w}^{\mathrm{T}} \mathbf{w}, \\ &\Leftrightarrow \quad \sum_k (\mathbf{w}^{\mathrm{T}} \mathbf{u}_k)^2 = c^2 \|\mathbf{w}\|_2^2. \end{aligned}$$

□

Lemma 5 (Union of two disjoint orthogonal bases)**.** *Let U and V be two orthogonal bases for $\mathbb{R}^n$ such that $\forall \, \mathbf{u} \in U, \mathbf{v} \in V \;\; \|\mathbf{u}\|_2 = \|\mathbf{v}\|_2 = c$, and $U \cap V = \emptyset$. Let $W = U \cup V$, then W is a tight uniform frame with frame bounds $A = B = 2c^2$.*

Proof. Since $\forall \, \mathbf{w} \in W$ either $\mathbf{w} \in U$ or $\mathbf{w} \in V$, we must have $\|\mathbf{w}\|_2 = c$. This takes care of uniformity, by Definition 7. See Appendix A for more basic frame definitions.

Let $\mathbf{b} \in \mathbb{R}^n$, then by Parseval's identity (Lemma 4, Equation 3.9) we have

$$\sum_k (\mathbf{b}^{\mathrm{T}}\mathbf{u}_k)^2 = c^2\|\mathbf{b}\|_2^2, \quad \text{if } U = \{\mathbf{u}_k \in \mathbb{R}^n : k = 1, \ldots, n\},$$

and

$$\sum_k (\mathbf{b}^{\mathrm{T}}\mathbf{v}_k)^2 = c^2\|\mathbf{b}\|_2^2, \quad \text{if } V = \{\mathbf{v}_k \in \mathbb{R}^n : k = 1, \ldots, n\}.$$

Hence

$$\begin{aligned}\sum_{\mathbf{w}\in W} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 &= \sum_{\mathbf{u}\in U}(\mathbf{b}^{\mathrm{T}}\mathbf{u})^2 + \sum_{\mathbf{v}\in V}(\mathbf{b}^{\mathrm{T}}\mathbf{v})^2 - \sum_{\mathbf{w}\in U\cap V}(\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 \qquad (3.10)\\ &= c^2\|\mathbf{b}\|_2^2 + c^2\|\mathbf{b}\|_2^2 - 0, \quad \text{since } U \cap V = \emptyset,\\ &= 2c^2\|\mathbf{b}\|_2^2.\end{aligned}$$

Let $A = B = 2c^2$. Then from Equation 3.10, $A\|\mathbf{b}\|_2^2 = \sum_k (\mathbf{b}^{\mathrm{T}}\mathbf{v}_k)^2 = B\|\mathbf{b}\|_2^2$. This establishes the required frame condition for a tight frame [4, 7]. $\square$

From the proof of Lemma 5, we obtain the following result.

Lemma 6 (Upper bound)**.** *Let U and V be as in Lemma 5, except that $U \cap V \neq \emptyset$. Then the following inequality holds,*

$$\forall\, \mathbf{b} \in \mathbb{R}^n \sum_{\mathbf{w}\in U\cup V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 \leq 2c^2\|\mathbf{b}\|_2^2,$$

and the inequality is tight whenever $U \cap V \neq U, V$.

Proof. From Equation 3.10 we have that, for $\forall\, \mathbf{b} \in \mathbb{R}^n$,

$$\begin{aligned}
\sum_{\mathbf{w}\in U\cup V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 &= \sum_{\mathbf{u}\in U} (\mathbf{b}^{T}\mathbf{u})^2 + \sum_{\mathbf{v}\in V} (\mathbf{b}^{\mathrm{T}}\mathbf{v})^2 - \sum_{\mathbf{w}\in U\cap V} (\mathbf{b}^{T}\mathbf{w})^2, \\
&\leq \sum_{\mathbf{u}\in U} (\mathbf{b}^{\mathrm{T}}\mathbf{u})^2 + \sum_{\mathbf{v}\in V} (\mathbf{b}^{\mathrm{T}}\mathbf{v})^2, \quad \text{since} \sum_{\mathbf{w}\in U\cap V\neq\emptyset} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 \geq 0, \\
&= c^2\|\mathbf{b}\|_2^2 + c^2\|\mathbf{b}\|_2^2, \quad \text{by Lemma 4} \\
&= 2c^2\|\mathbf{b}\|_2^2.
\end{aligned}$$

Finally, assume that $U \cap V \neq U, V$, or equivalently $U \cap V \subsetneq U, V$. This implies that $span\{U \cap V\} \subsetneq \mathbb{R}^n$, therefore there is a nonzero vector $\tilde{\mathbf{b}} \in \mathbb{R}^n$ such that $\tilde{\mathbf{b}} \perp span\{U \cap V\}$. This implies that

$$\sum_{\mathbf{w}\in U\cap V\neq\emptyset} (\tilde{\mathbf{b}}^{\mathrm{T}}\mathbf{w})^2 = 0,$$

and therefore $\sum_{\mathbf{w}\in U\cup V} (\tilde{\mathbf{b}}^{\mathrm{T}}\mathbf{w})^2 = 2c^2\|\tilde{\mathbf{b}}\|$. □

We have a similar result for the lower bound in the general case.

Lemma 7 (Lower bound). *Let U and V be as in Lemma 5, except that $U \cap V \neq \emptyset$. Then the following inequality holds,*

$$\forall\, \mathbf{b} \in \mathbb{R}^n \quad c^2\|\mathbf{b}\|_2^2 \leq \|\mathbf{b}\|_2^2 \left(2c^2 - \max_{\substack{\|\tilde{\mathbf{b}}\|_2=1 \\ \tilde{\mathbf{b}}\in\mathbb{R}^n}} \sum_{\mathbf{w}\in U\cap V} (\tilde{\mathbf{b}}^{\mathrm{T}}\mathbf{w})^2 \right) \leq \sum_{\mathbf{w}\in U\cup V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2.$$

Moreover, the second inequality is tight.

Proof. Let $\mathbf{0} \neq \mathbf{b} \in \mathbb{R}^n$, then

$$\begin{aligned}
\sum_{\mathbf{w}\in U\cap V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 &= \|\mathbf{b}\|_2^2 \sum_{\mathbf{w}\in U\cap V} \left(\frac{\mathbf{b}}{\|\mathbf{b}\|_2}^{\mathrm{T}} \mathbf{w} \right)^2, \\
&\leq \|\mathbf{b}\|_2^2 \max_{\substack{\|\tilde{\mathbf{b}}\|_2=1 \\ \tilde{\mathbf{b}}\in\mathbb{R}^n}} \sum_{\mathbf{w}\in U\cap V} \left(\frac{\tilde{\mathbf{b}}}{\|\tilde{\mathbf{b}}\|_2}^{\mathrm{T}} \mathbf{w} \right)^2, \qquad (3.11) \\
&\leq \|\mathbf{b}\|_2^2 \max_{\substack{\|\tilde{\mathbf{b}}\|_2=1 \\ \tilde{\mathbf{b}}\in\mathbb{R}^n}} \sum_{\mathbf{w}\in U} \left(\frac{\tilde{\mathbf{b}}}{\|\tilde{\mathbf{b}}\|_2}^{\mathrm{T}} \mathbf{w} \right)^2, \\
&= \|\mathbf{b}\|_2^2 c^2, \quad \text{by Lemma 4.}
\end{aligned}$$

If $\mathbf{b} = \mathbf{0}$ then $\sum_{\mathbf{w}\in U\cap V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 \leq \|\mathbf{b}\|_2^2 c^2$ trivially since both sides of the inequality are 0. Therefore Inequality 3.11 holds for all $\mathbf{b} \in \mathbb{R}^n$, and we have

$$-c^2\|\mathbf{b}\|_2^2 \leq -\|\mathbf{b}\|_2^2 \max_{\substack{\|\tilde{\mathbf{b}}\|_2=1 \\ \tilde{\mathbf{b}}\in\mathbb{R}^n}} \sum_{\mathbf{w}\in U\cap V} \left(\frac{\tilde{\mathbf{b}}}{\|\tilde{\mathbf{b}}\|_2}^{\mathrm{T}} \mathbf{w} \right)^2 \leq - \sum_{\mathbf{w}\in U\cap V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 \qquad (3.12)$$

Combining Equation 3.10 and the Inequalities 3.12 we have that, for $\forall\, \mathbf{b} \in \mathbb{R}^n$,

$$\begin{aligned}
&\sum_{\mathbf{u}\in U} (\mathbf{b}^{\mathrm{T}}\mathbf{u})^2 + \sum_{\mathbf{v}\in V} (\mathbf{b}^{\mathrm{T}}\mathbf{v})^2 - \sum_{\mathbf{w}\in U\cap V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 = \sum_{\mathbf{w}\in U\cup V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2, \\
\Rightarrow \quad & 2c^2\|\mathbf{b}\|_2^2 - \sum_{\mathbf{w}\in U\cap V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 = \sum_{\mathbf{w}\in U\cup V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2, \quad \text{by Lemma 4,} \\
\Rightarrow \quad & c^2\|\mathbf{b}\|_2^2 \leq \|\mathbf{b}\|_2^2 \left(2c^2 - \max_{\substack{\|\tilde{\mathbf{b}}\|_2=1 \\ \tilde{\mathbf{b}}\in\mathbb{R}^n}} \sum_{\mathbf{w}\in U\cap V} (\tilde{\mathbf{b}}^{\mathrm{T}}\mathbf{w})^2 \right) \leq \sum_{\mathbf{w}\in U\cup V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2.
\end{aligned}$$

Let $\mathbf{b}^* = \arg\max_{\mathbf{b}} \sum_{\mathbf{w}\in U\cap V} \left(\frac{\mathbf{b}}{\|\mathbf{b}\|}^{\mathrm{T}} \mathbf{w} \right)^2$, then Inequality 3.11 becomes an equality and by Equation 3.10 the claim of tightness follows. □

Combining Lemmas 5, 6, and 7 we obtain the following result,

Theorem 8. *Let U and V be orthogonal bases for $\mathbb{R}^n$ such that $\forall\, \mathbf{u} \in U\ \|\mathbf{u}\|_2 = c$, and $\forall\, \mathbf{v} \in V\ \|\mathbf{v}\|_2 = c$. Then $W = U \cup V$ is a uniform frame with frame bounds*

$$A = 2c^2 - \max_{\substack{\|\mathbf{b}\|_2=1 \\ \mathbf{b}\in\mathbb{R}^n}} \sum_{\mathbf{w}\in U\cap V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2, \quad \textit{and} \quad B = \begin{cases} 2c^2 & \textit{if } U \cap V \neq U, V \\ c^2 & \textit{otherwise} \end{cases}.$$

Moreover, if $U \cap V = \emptyset$ then W is a tight frame with frame bounds $A = B = 2c^2$.

Proof. The only thing left to prove is that $B = c^2$ whenever $U \cap V = U$ or $U \cap V = V$. In either case we have that $U = V$, and the claim follows from Parseval's identity (Lemma 4, Equation 3.9.) Note that in this case, $A = c^2$ since $\max_{\|\mathbf{b}\|_2=1} \sum_{\mathbf{w}\in U\cap V} (\mathbf{b}^{\mathrm{T}}\mathbf{w})^2 = c^2$, for the same reason. $\square$

In subsequent sections, we will conduct a detailed study of the compression properties of both [DCT$_1$ Haar$_1$] and [DCT$_{2,j}$ Haar$_{2,j}$], for $j = 1, 2, 3$, as well as matrices derived from them.

3.3 Image database

To carry out our experiments and test image compression via sparsity, as well as the properties of the matrices described in Section 3.2 for that purpose, we selected 5 natural images, 4 of them from the University of Southern California's Signal & Image Processing Institute (USC-SIPI) image database [24]. This database has been widely used for image processing benchmarking. The images are described in Table 3.1 and shown in Figure 3.3. All images are 512 by 512, 8-bit grayscale images, which means they are composed of $512^2 = 262{,}144$ pixels that can take integer values from 0 (black) to 255 (white).

USC-SIPI file	Name	Size	Notes
n/a	Barbara	512x512 pixels, 8-bit	
boat.512	Boat	512x512 pixels, 8-bit	
elaine.512	Elaine	512x512 pixels, 8-bit	
4.2.07	Peppers	512x512 pixels, 24-bit color image	Converted to 8-bit grayscale image.
5.2.10	Stream	512x512 pixels, 8-bit	

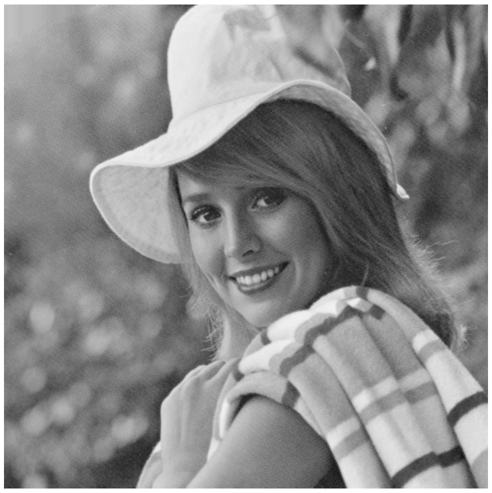
Table 3.1: List of images used. Image *Peppers* was converted to a grayscale image from the 24-bit color original using Photoshop CS3. Image *Barbara* is not in the USC-SIPI database but we included since it is widely used by the image processing community.

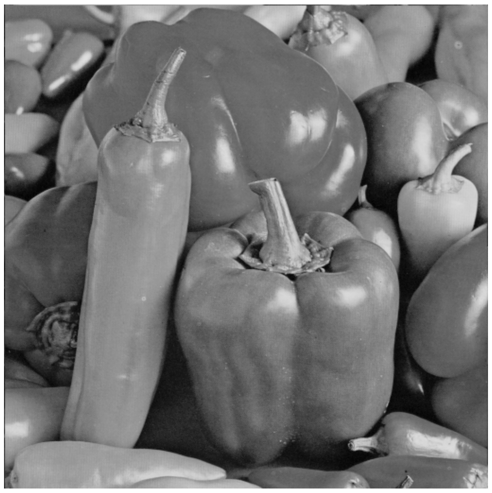
3.4 Methodology

Following the approach to image processing at the core of the JPEG image compression standard [25], we subdivide each image in our database in 8 by 8 non-overlapping squares that will be treated individually. Since we need to generate a right hand side vector $\mathbf{b}$ to implement our compression scheme via sparsity, cf. Section 3.2, a sub-image $\mathbf{Y} \in \mathbb{R}^{8\times 8}$ of size 8 by 8 pixels needs be linearized into a vector $\mathbf{y} \in \mathbb{R}^{64}$ to play the role of $\mathbf{b}$. There are many ways to do this, but we tested only three possible approaches. The first one consisted of concatenating one after the other the columns of $\mathbf{Y}$ to form $\mathbf{y}$, we shall call this method c_1, which can be thought of as a bijection $c_1 : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$ that maps $\mathbf{Y} \mapsto \mathbf{y}$ the way just described above. See Figure 3.4(a).

(a) Barbara

(b) Boat


(c) Elaine


(d) Peppers

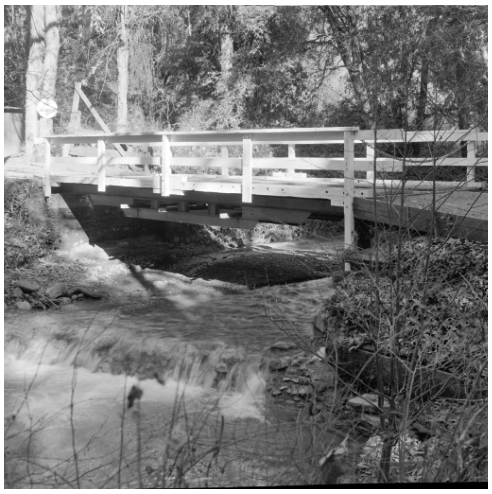
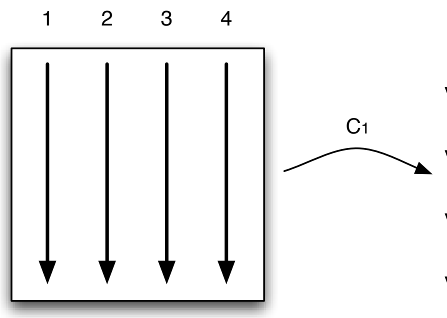
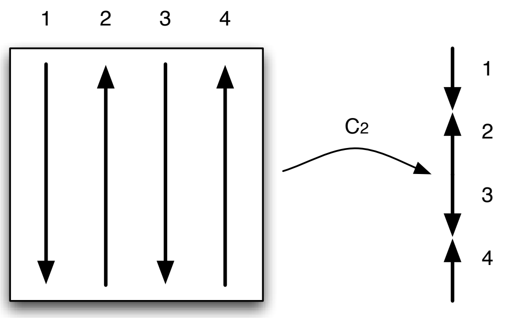

(e) Stream

Figure 3.3: Images used for our compression algorithms based on sparse image representation.



(a)



(b)

Figure 3.4: Two possible ways to vectorize a matrix. (a) Concatenate from top to bottom one after the other, from left to right, the columns of the matrix; or (b) first flip every other column, and then concatenate the columns as before. This is what functions c_1 and c_2 do, respectively.

Yet another method would be to reverse the ordering of the entries of every even column of $\mathbf{Y}$ and then concatenate the columns of the resulting matrix. That is, from $\mathbf{Y}$ first obtain the matrix $\mathbf{Y}'$, where $Y'_{i,2j} = Y_{8+1-i,2j}$, and $Y'_{\cdot,\cdot}$ and $Y_{\cdot,\cdot}$ are the entries of $\mathbf{Y}'$ and $\mathbf{Y}$, respectively. Then concatenate one after the other the columns of $\mathbf{Y}'$ to form $\mathbf{y}'$. Call this method c_2, also a bijection $c_2 : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$ that maps $\mathbf{Y} \mapsto \mathbf{y}'$ the way just described. See Figure 3.4(b).

Finally, a third method, $c_3 : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$, would traverse an 8 by 8 sub-image $\mathbf{Y}$ in a zigzag pattern from left to right, top to bottom, along the diagonal that goes from its top left to its bottom right corners and map it to a vector $\mathbf{y}'' \in \mathbb{R}^{64}$, in a similar way as the zigzag mapping shown in Figure 3.5, which depicts this mapping but for a 4 by 4 image that is mapped into a vector in $\mathbb{R}^{16}$.

These are the three methods we consider to vectorize a matrix $\mathbf{Y}$ representing an 8 by 8 sub-image from any of the images in our test database. Whether we generate a signal vector $\mathbf{b} \in \mathbb{R}^{64}$ by either setting $\mathbf{b} = c_1(\mathbf{Y})$, $\mathbf{b} = c_2(\mathbf{Y})$, or

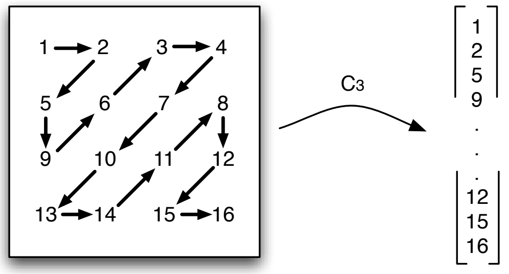


Figure 3.5: How to vectorize a matrix following a zigzag pattern. A 4 by 4 matrix is shown with its 16 elements enumerated from left to right, top to bottom. The path of arrows shows the new induced order after traversing the matrix in a zigzag pattern to obtain a vector.

$\mathbf{b} = c_3(\mathbf{Y})$, we still need to designate a matrix $\mathbf{A} \in \mathbb{R}^{64\times128}$ as well to complete the setup. This is where we use [DCT$_1$ Haar$_1$] and [DCT$_{2,j}$ Haar$_{2,j}$], with $j = 1, 2, 3$. That is, we will consider $\mathbf{A}$ equal to any of these matrices, defined in Sections 3.2.1 and 3.2.2, respectively.

Once we have chosen a way to vectorize a sub-image, say c_i—with i either 1, 2, or 3—and a matrix $\mathbf{A}$, we proceed in the following way. Given a tolerance $\epsilon_0 > 0$, and an image $\mathcal{I}$ that has been partitioned in 8 by 8 non-overlapping sub-images, say $\{\mathbf{Y}_l\}$—where $l = 1, \ldots, (512/8)^2$ for the images in our database—we obtain the approximation to $\mathbf{y}_l = c_i(\mathbf{Y}_l)$ derived from the OMP algorithm, i.e., from the sparse $\mathbf{x}_l = \texttt{OMP}(\mathbf{A}, \mathbf{y}_l, \epsilon_0)$, we compute $\tilde{\mathbf{y}}_l = \mathbf{A}\mathbf{x}_l$, where $\mathbf{A} =$ [DCT$_1$ Haar$_1$], or $\mathbf{A} =$ [DCT$_{2,j}$ Haar$_{2,j}$], for j equal to 1, 2, or 3. Using $\tilde{\mathbf{y}}_l$ we can then reconstruct a sub-image by setting $\widetilde{\mathbf{Y}}_l = c_i^{-1}(\tilde{\mathbf{y}}_l)$.

Finally, we can rebuild and approximate the original image $\mathcal{I}$ by pasting together, in the right order and position, the set of sub-images $\{\widetilde{\mathbf{Y}}_l\}$, and form that way $\widetilde{\mathcal{I}}$, the approximate image reconstruction of $\mathcal{I}$. This new image $\widetilde{\mathcal{I}}$ is necessarily an approximation, and not the original image $\mathcal{I}$, because in the process we have introduced an error by setting the tolerance $\epsilon_0 > 0$, and not $\epsilon_0 = 0$. Recall that $\|\tilde{\mathbf{y}} - \mathbf{y}\|_2 < \epsilon_0$, and therefore we will likely have $\widetilde{\mathcal{I}} \neq \mathcal{I}$. Since $\|\mathbf{x}_l\|_0 \leq \|\mathbf{y}_l\|_0$, and more likely $\|\mathbf{x}_l\|_0 \ll \|\mathbf{y}_l\|_0$, storing wisely the set $\{\mathbf{x}_l\}_{l=1}^{M}$, where M is the number of sub-images partitioning $\mathcal{I}$, will provide a compressed representation of image $\mathcal{I}$.

How to efficiently create a compressed representation of image $\mathcal{I}$ using $\{\mathbf{x}_l\}_{l=1}^{M}$, and the effects of the choice of tolerance ϵ_0, map $c_i : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$, and matrix $\mathbf{A}$ on such representation will all be addressed in subsequent sections.

3.5 Effects on image reconstruction from choice of map c_i

Given an image $\mathcal{I}$ in our database, we can follow and apply to it the methodology described in Section 3.4, and obtain at the end of this process a reconstructed image $\widetilde{\mathcal{I}}$ from it. In this section we explore the effects of the choice of map $c_i : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$ on the characteristics of image $\widetilde{\mathcal{I}}$ for the different choices of matrix $\mathbf{A}$ that we have selected to study.

We summarize the empirical observations obtained from the experiments described below, and suggest a possible explanation for them without further proof.

3.5.1 Results for $\mathbf{A} = [\mathrm{DCT}_1\ \mathrm{Haar}_1]$

We conducted the following experiments. We set $\mathbf{A} = [\mathrm{DCT}_1\ \mathrm{Haar}_1]$, and chose a tolerance of $\epsilon = 32$. Then, for each image $\mathcal{I}$ in our database—and each index $i = 1, 2, 3$—we chose map $c_i : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$ for the step in the methodology mentioned before that converts a sub-image $\mathbf{Y}_l$, from a partition in 8 by 8 sub-images of $\mathcal{I}$, into a vector $\mathbf{y}_l = c_i(\mathbf{Y}_l)$. The collection of vectors $\{\mathbf{y}_l\}_{l\in L}$ is used to eventually build image $\widetilde{\mathcal{I}}_i$, which depends on the choice of map c_i this way. See Section 3.4.

The characteristics that we used to measure the impact of the choice of map c_i are the *normalized bit-rate*, and the *peak signal-to-noise ratio* (PSNR). See Definitions 3 and 4, respectively. A smaller value for the normalized bit-rate is better than a bigger one given that this implies less bits are necessary to represent the image. A larger value for the PSNR is better than a smaller one as this means the fidelity of the representation is higher. Table 3.2 summarizes the results of the experiments. From these, we can conclude that for matrix $\mathbf{A} = [\mathrm{DCT}_1\ \mathrm{Haar}_1]$, the choice of c_2 over c_1 or c_3 produces better results. This could be because, if $\mathbf{Y} = (Y_{i,j})_{i,j=1,\ldots,8}$ is a natural image, on average $|Y_{8,j} - Y_{8,j+1}| < |Y_{8,j} - Y_{1,j+1}|$ for $j = 1, \ldots, 7$, which makes $\mathbf{y}_{c_2} = c_2(\mathbf{Y})$ change more slowly than $\mathbf{y}_{c_1} = c_1(\mathbf{Y})$. By analogy to the behavior of the DFT, this must translate into needing less column vectors from $\mathbf{A}$ to describe, within a certain error ϵ, the signal $\mathbf{y}_{c_2}$ compared to the number of columns needed to approximate signal $\mathbf{y}_{c_1}$ to the same error tolerance.

What could explain the superiority of the ordering induced by c_1 over c_3? For this, we introduce the concept of *total variation* for a vector $\mathbf{v}$.

Definition 2 (Total Variation). *Let $\mathbf{v} \in \mathbb{R}^n$. The total variation of $\mathbf{v} = (v_1, \ldots, v_n)^T$ is the quantity,*

$$V(\mathbf{v}) = \sum_{i=1}^{n-1} |v_{i+1} - v_i|. \tag{3.13}$$

Image/Function	PSNR (dB)	Normalized bit-rate (bpp)
Barbara		
c_1:	36.8996	0.1833
c_2:	36.9952	0.1863
c_3:	36.8470	0.2338
Boat		
c_1:	36.5791	0.1812
c_2:	36.6020	0.1608
c_3:	36.5615	0.2205
Elaine		
c_1:	36.5003	0.1763
c_2:	36.5155	0.1682
c_3:	36.4877	0.1885
Peppers		
c_1:	36.7936	0.1193
c_2:	36.8674	0.1024
c_3:	36.7648	0.1372
Stream		
c_1:	36.4423	0.3161
c_2:	36.4686	0.3050
c_3:	36.4400	0.3504

Table 3.2: Performance results for $\mathbf{A} = [\text{DCT}_1\ \text{Haar}_1]$ for each c_1, c_2, and c_3. For each image in our test database, we linearized each 8 by 8 sub-image using either c_1, c_2, or c_3. In all cases, the PSNR value was larger using c_2; and in all cases, except for image *Barbara*—although minimally—, the normalized bit-rate was smaller. Both of these measures make c_2 a better choice over c_1 or c_3. The values correspond to runs of OMP with an ℓ^2 termination criteria, and a tolerance $\epsilon = 32$.

Image	Averages			Maximums		
	c_1	c_2	c_3	c_1	c_2	c_3
Barbara	684.9177	624.9761	850.5647	3793	3894	4681
Boat	559.6167	471.2769	625.9233	2991	2102	4005
Elaine	552.3401	487.6824	479.7908	2564	1570	2082
Peppers	423.7781	356.2764	405.8694	2707	1940	2430
Stream	978.5122	844.6191	971.4321	3563	2826	2869

Table 3.3: Total variation averages $V_{avg}(\mathcal{I}, c_i)$ and maximums $V_{max}(\mathcal{I}, c_i)$

For an image $\mathcal{I}$ in our database, create from it the partition of non-overlapping 8 by 8 sub-images $\{\mathbf{Y}_l\}_{l \in L}$. Since our images are 512 by 512 pixels, we will have $64 \times 64 = 4096$ sub-images $\mathbf{Y}_l$. That is, $\#L = 4096$. Set $i = 1, 2, \text{or } 3$, and for each and everyone of those sub-images $\mathbf{Y}_l$ compute the total variation $V(c_i(\mathbf{Y}_l))$. Then plot in a 64 by 64 image the results to visualize the aggregate of all these values.

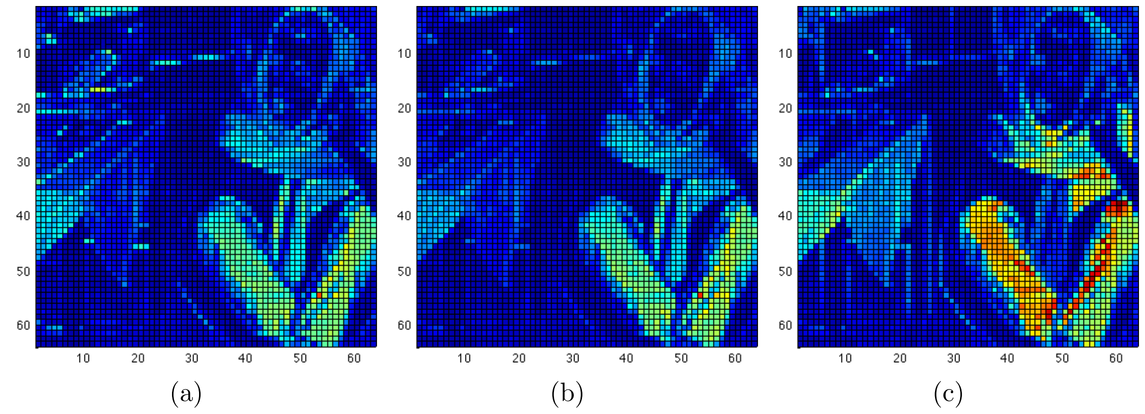


Figure 3.6: Total variation for image *Barbara* for the vectorization function (a) c_1, (b) c_2, and (c) c_3. Blue represents low values, red high values.

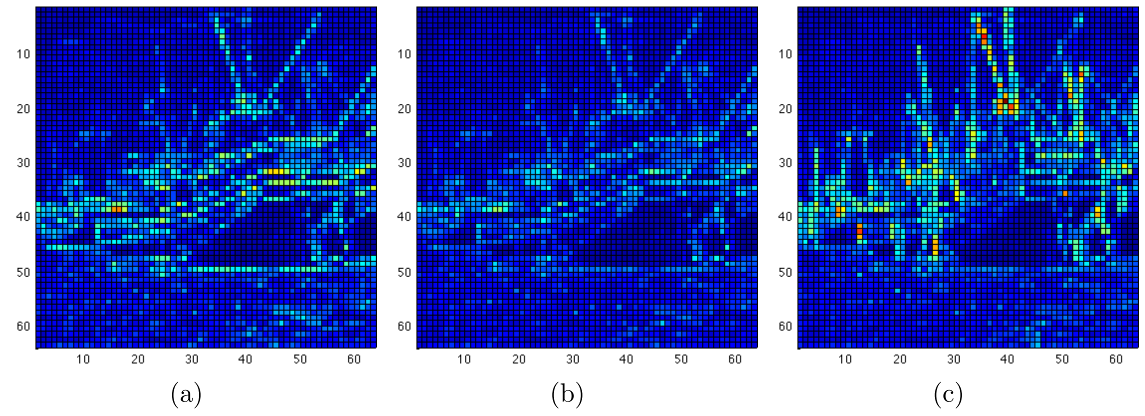


Figure 3.7: Total variation for image *Boat* for the vectorization function (a) c_1, (b) c_2, and (c) c_3. Blue represents low values, red high values.

The images obtained this way are shown in Figures 3.6 through 3.10. Also,

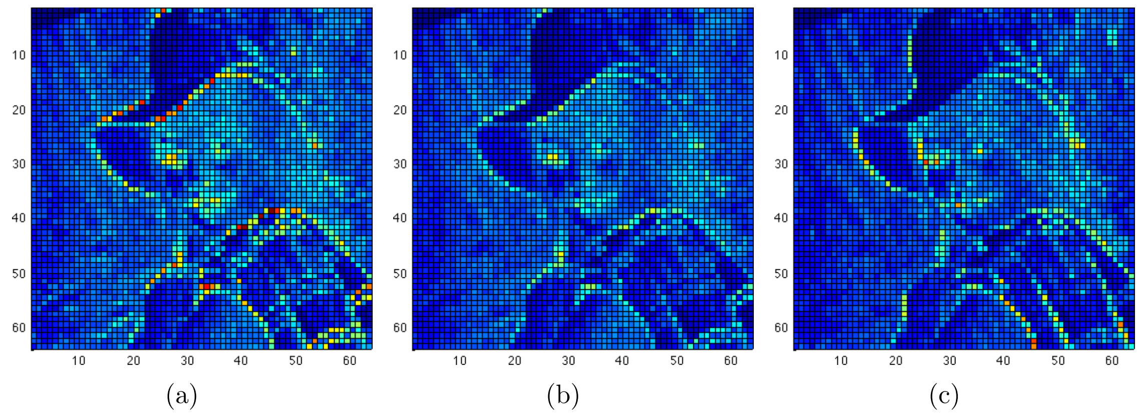


Figure 3.8: Total variation for image *Elaine* for the vectorization function (a) c_1, (b) c_2, and (c) c_3. Blue represents low values, red high values.

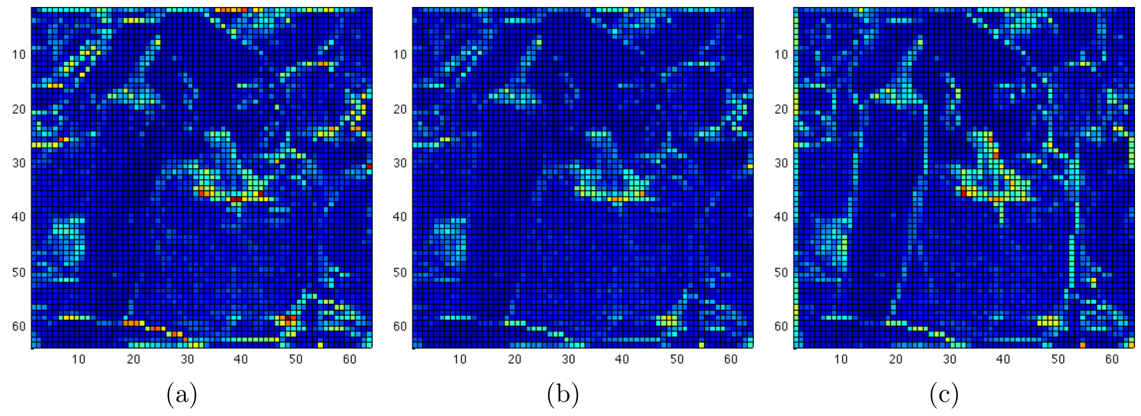


Figure 3.9: Total variation for image *Peppers* for the vectorization function (a) c_1, (b) c_2, and (c) c_3. Blue represents low values, red high values.

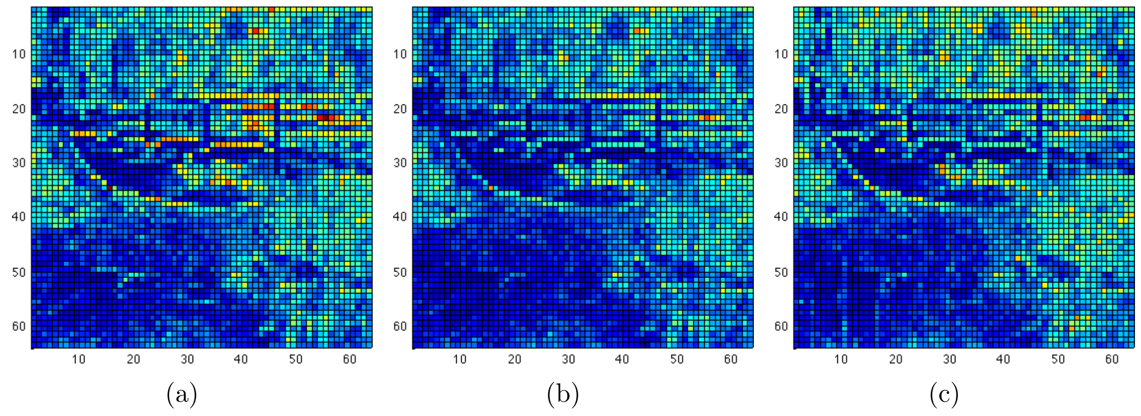


Figure 3.10: Total variation for image *Stream* for the vectorization function (a) c_1, (b) c_2, and (c) c_3. Blue represents low values, red high values.

Table 3.3 contains the maximum value,

$$V_{max}(\mathcal{I}, c_i) = \max_{l \in L} V(c_i(\mathbf{Y}_l)), \tag{3.14}$$

and the average value,

$$V_{avg}(\mathcal{I}, c_i) = \frac{1}{\#L} \sum_{l=1}^{\#L} V(c_i(\mathbf{Y}_l)), \tag{3.15}$$

for each image $\mathcal{I}$ in our database, and each and all of the vectorization functions $c_i : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$ that we considered. We observe that usually, but not always, that the smallest value of V_{avg} predicts the largest PSNR, and the smallest value of V_{max} predicts the smallest normalized bit-rate. However, as observed, this is not always the case. If we had only considered c_1 and c_3 for linearizing functions, the prediction would have failed both for the PSNR and normalized bit-rate measures for image *Elaine*, for example. This could be because the inherent properties of the matrix chosen to perform the compression, $\mathbf{A} = [\text{DCT}_1\ \text{Haar}_1]$ in this case, may have an impact as well. This is showcased in the following section, where we will see that the total variation of the different vectors resulting from different choices of linearizing functions c_i seems to have less of an impact in the PSNR and normalized bit-ratio characteristics of the reconstructed image when $\mathbf{A} = [\text{DCT}_{2,3}\ \text{Haar}_{2,3}]$.

3.5.2 Results for $\mathbf{A} = [\mathrm{DCT}_{2,j}\ \mathrm{Haar}_{2,j}]$

Proceeding in a similar fashion as in Section 3.5.1, we set $\mathbf{A} = [\mathrm{DCT}_{2,j}\ \mathrm{Haar}_{2,j}]$ for each $j = 1, 2,$ and 3, and for each such instance of $\mathbf{A}$, we perform the following experiment.

Assume that $\mathbf{A} = [\mathrm{DCT}_{2,j}\ \mathrm{Haar}_{2,j}]$, and set the vectorization function to match the same ordering that was used to create $\mathbf{A}$. This means we pick the vectorization function to be c_j. We compute $\mathbf{y}_l = c_j(\mathbf{Y}_l)$, where—as before—sub-image $\mathbf{Y}_l$ comes from the partition $\{\mathbf{Y}_l\}_{l\in L}$ of an image $\mathcal{I}$ in our image database. Then, continuing with the compression methodology described in Section 3.4, we obtain $\mathbf{x}_l = \mathtt{OMP}(\mathbf{A}, \mathbf{y}_l, \epsilon_0)$, setting $\epsilon_0 = 32$ for this experiment. Finally, from the set of vectors $\{\mathbf{x}_l\}_{l\in L}$ we obtain $\tilde{\mathbf{y}}_l = \mathbf{A}\mathbf{x}_l$, and use $\{\tilde{\mathbf{y}}_l\}_{l\in L}$ to eventually obtain the reconstructed image $\widetilde{\mathcal{I}}$ of our original image $\mathcal{I}$. Again, as in Section 3.5.1, we asses the effects of the choice of linearizing function c_i by the values of PSNR and normalized bit-rate resulting from this representation of $\mathcal{I}$ by $\widetilde{\mathcal{I}}$. We show the summary of the results of this experiment for all matrices $\mathbf{A}$ considered above, and all images in our database, in Table 3.4.

We point out that choosing c_i, with $i \neq j$, when $\mathbf{A} = [\mathrm{DCT}_{2,j}\ \mathrm{Haar}_{2,j}]$ results in worse values of both PSNR and normalized bit-rate than when $i = j$, as described above. We report only the results where $i = j$.

Remarkably, any choice of $\mathbf{A} = [\mathrm{DCT}_{2,j}\ \mathrm{Haar}_{2,j}]$ with a matching vectorization function c_j performs better than when $\mathbf{A} = [\mathrm{DCT}_1\ \mathrm{Haar}_1]$ for the normalized bit-rate metric, and almost better for the PSNR metric except for image *Stream* by 0.0008 of a dB. Also, on average, the vectorization order imposed by c_3 is slightly better than those by either c_1 or c_2, although the difference is practically imperceptible to the human eye. The normalized bit-rate figures all coincide.

Image/Function	PSNR (dB)	Normalized bit-rate (bpp)	Matrix
Barbara			
c_1 :	37.0442	0.1634	[DCT$_{2,1}$ Haar$_{2,1}$]
c_2 :	37.0443	0.1634	[DCT$_{2,2}$ Haar$_{2,2}$]
c_3 :	37.0443	0.1634	[DCT$_{2,3}$ Haar$_{2,3}$]
Boat			
c_1 :	36.6122	0.1541	[DCT$_{2,1}$ Haar$_{2,1}$]
c_2 :	36.6120	0.1541	[DCT$_{2,2}$ Haar$_{2,2}$]
c_3 :	36.6120	0.1541	[DCT$_{2,3}$ Haar$_{2,3}$]
Elaine			
c_1 :	36.5219	0.1609	[DCT$_{2,1}$ Haar$_{2,1}$]
c_2 :	36.5219	0.1609	[DCT$_{2,2}$ Haar$_{2,2}$]
c_3 :	36.5220	0.1609	[DCT$_{2,3}$ Haar$_{2,3}$]
Peppers			
c_1 :	36.8780	0.0955	[DCT$_{2,1}$ Haar$_{2,1}$]
c_2 :	36.8780	0.0955	[DCT$_{2,2}$ Haar$_{2,2}$]
c_3 :	36.8780	0.0955	[DCT$_{2,3}$ Haar$_{2,3}$]
Stream			
c_1 :	36.4678	0.2957	[DCT$_{2,1}$ Haar$_{2,1}$]
c_2 :	36.4676	0.2957	[DCT$_{2,2}$ Haar$_{2,2}$]
c_3 :	36.4677	0.2957	[DCT$_{2,3}$ Haar$_{2,3}$]

Table 3.4: Performance results for $\mathbf{A}$ = [DCT$_{2,1}$ Haar$_{2,1}$], $\mathbf{A}$ = [DCT$_{2,2}$ Haar$_{2,2}$], and $\mathbf{A}$ = [DCT$_{2,3}$ Haar$_{2,3}$] with corresponding vectorization functions c_1, c_2, and c_3, respectively. In all cases, the PSNR and normalized bit-rate values were almost identical, with minor differences. Matrix $\mathbf{A}$ = [DCT$_{2,3}$ Haar$_{2,3}$] performs slightly better on average. Mismatching function c_i with matrix $\mathbf{A}$ = [DCT$_{2,j}$ Haar$_{2,j}$] when $i \neq j$, results in degraded performance. The values correspond to runs of OMP with an ℓ^2 termination criteria, and a tolerance $\epsilon = 32$.

3.6 Normalized bit-rate vs tolerance

Following the methodology described in Section 3.4, suppose that we have an image $\mathcal{I}$ from our database and let $\{\mathbf{Y}_l\}_{l \in L}$ be a partition of $\mathcal{I}$ in $\#L$ sub-images of size 8 by 8. Let $\mathbf{y}_l = c_i(\mathbf{Y}_l)$ for some $i = 1, 2, \text{or } 3$, where $c_i : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$ is one of the three maps defined in Section 3.4. See Figures 3.4 and 3.5. Using OMP, with a full-rank matrix $\mathbf{A} \in \mathbb{R}^{64\times 128}$, and a tolerance $\epsilon_0 > 0$, obtain $\mathbf{x}_l = \texttt{OMP}(\mathbf{A}, \mathbf{y}_l, \epsilon_0)$, and compute the approximation to $\mathbf{y}_l$ given by $\tilde{\mathbf{y}}_l = \mathbf{A}\mathbf{x}_l$.

We know that $\|\tilde{\mathbf{y}}_l - \mathbf{y}_l\|_2 < \epsilon_0$, and we can count how many nonzero entries there are in $\mathbf{x}_l$, namely $\|\mathbf{x}_l\|_0$. With this, we can define the *normalized bit-rate.*

Definition 3 (Normalized Bit-Rate)**.** *Given the context above, the normalized bit-rate measured in bits per pixel (bpp) for image $\mathcal{I}$—given compression matrix $\mathbf{A}$, and tolerance ϵ_0—is the number*

$$nbr(\mathcal{I}, \mathbf{A}, \epsilon_0) = \frac{\sum_l \|\mathbf{x}_l\|_0}{N_1 N_2}, \tag{3.16}$$

where image $\mathcal{I}$ is of size N_1 by N_2 pixels ($N_1 = N_2 = 512$ in our case).

This is how to interpret this definition. Suppose that it takes a binary digit or "bit" (0 or 1, for example) to represent a nonzero coordinate in vector $\mathbf{x}_l$, and that we—for the moment—ignore how to keep track of the index l, then we will need $\|\mathbf{x}_l\|_0$ bits to store or transmit $\mathbf{x}_l$. Then, the total number of bits to represent image $\mathcal{I}$ is $\sum_l \|\mathbf{x}_l\|_0$. The average number of bits per pixel (bpp) is then obtained by dividing this quantity by $N_1 N_2$. This is the normalized bit-rate.

Compare Definition 3 with more traditional forms of compression measures. From [22], the purpose of image compression is to represent the image $\mathcal{I}$ of size N_1 by N_2 and depth B with a string of bits, called the compressed "bit-stream", denoted $\mathbf{c}$. The objective is to keep the length of $\mathbf{c}$, say $length(\mathbf{c})$, as small as possible. In the absence of any compression, we require $N_1 N_2 B$ bits to represent the image sample values. The *compression ratio* is then defined as

$$cr(\mathcal{I}, \mathbf{c}) = \frac{N_1 N_2 B}{length(\mathbf{c})}, \tag{3.17}$$

and the *compressed bit-rate*, expressed in bpp, as

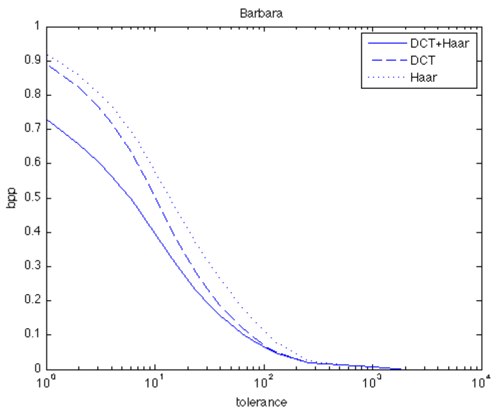
$$br(\mathcal{I}, \mathbf{c}) = \frac{length(\mathbf{c})}{N_1 N_2}. \tag{3.18}$$

The compression ratio is a dimensionless quantity that tells how many times we have managed to reduce in size the original representation of the image, while the compressed bit-rate has bpp units and tells how many bits are used on average per sample by the compressed bit-stream $\mathbf{c}$ to represent the original image $\mathcal{I}$.

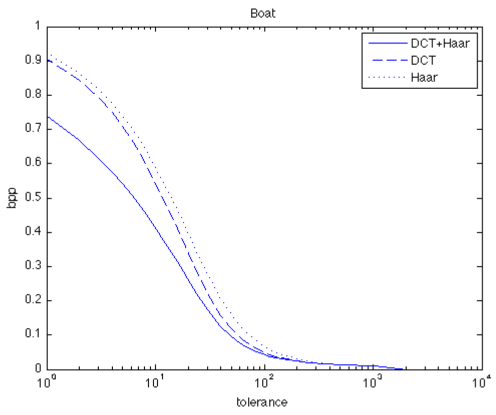
We note that from Equations 3.16 and 3.18 we must have that $nbr(\mathcal{I}, \mathbf{A}, \epsilon_0) \leq br(\mathcal{I}, \mathbf{c})$ if the bit-stream $\mathbf{c}$ is derived from the sparse representation induced by $\mathbf{A}$ and ϵ_0. The reason for this is two-fold. Firstly, it is unlikely that the coordinates in each of the resulting $\mathbf{x}_l$ vectors will be realistically represented by only one bit, and secondly, because $\mathbf{c}$ would have to somehow include a coding for the indices l for each $\mathbf{x}_l$, necessarily increasing the bit count some more. These particular issues, i.e., the number of bits to represent the entries of vectors $\mathbf{x}_l$, and the coding of the indices l for each of those vectors into a final compressed bit-stream $\mathbf{c}$, will be addressed in Section 4.

Notwithstanding the above remarks, there is still value in using the normalized bit-stream measure to quantify and plot normalized bit-rate vs tolerance graphs to gauge the compression properties of various compression matrices.

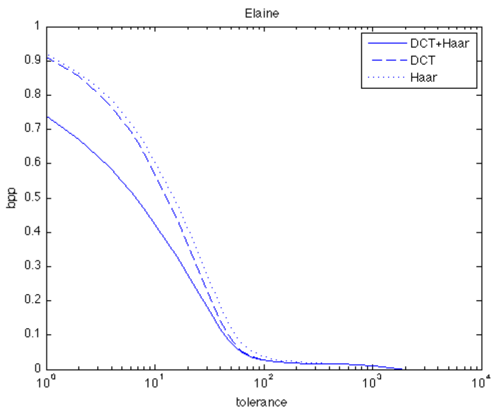
Given the results in Table 3.4, we look into the compression properties of matrices $\mathbf{A} = [\text{DCT}_1\ \text{Haar}_1]$, and $\tilde{\mathbf{A}} = [\text{DCT}_{2,3}\ \text{Haar}_{2,3}]$. We compare these properties for both matrices relative to each other, and to the compression properties of $\mathbf{B}$ and $\mathbf{C}$, which are formed from the DCT_1 or the Haar_1 submatrices of matrix $\mathbf{A}$, respectively. We plot for all images in our database their respective normalized bit-rate vs tolerance graphs. We took $\epsilon_0 \in T = \{2^k\}_{k=0}^{11} \cup \{3, 5, 6, 7, 24, 40, 48, 56, 80, 96, 112\}$ and for each image $\mathcal{I}$ in our image database we obtained the corresponding normalized bit-rates $nbr(\mathcal{I}, \mathbf{A}, \epsilon_0)$, $nbr(\mathcal{I}, \mathbf{B}, \epsilon_0)$, and $nbr(\mathcal{I}, \mathbf{C}, \epsilon_0)$ to obtain the plots in Figure 3.11. In Figure 3.12 we compare $\mathbf{A} = [\text{DCT}_1\ \text{Haar}_1]$ with $\tilde{\mathbf{A}} = [\text{DCT}_{2,3}\ \text{Haar}_{2,3}]$. We observe that up to a tolerance $\epsilon_{\mathcal{I}}$, dependent on image $\mathcal{I}$, matrix $\tilde{\mathbf{A}}$ performs better for tolerance values $\epsilon \geq \epsilon_{\mathcal{I}}$. That is, the value of the normalized bit-rate is smaller when performing compression utilizing matrix $\tilde{\mathbf{A}}$. For values of $\epsilon \leq \epsilon_{\mathcal{I}}$ compression with matrix $\mathbf{A}$ results in better normalized bit-rate values. We shall see in Section 3.7 that for values of $\epsilon = 32$, and smaller, the quality of the image reconstruction is very good. We note from Figure 3.12 that, for all images in our database, $\epsilon_{\mathcal{I}} < 32$. This means that, for most practical cases, the use of $[\text{DCT}_{2,3}\ \text{Haar}_{2,3}]$ results in slightly smaller normalized bit-rate values than when using $[\text{DCT}_1\ \text{Haar}_1]$.



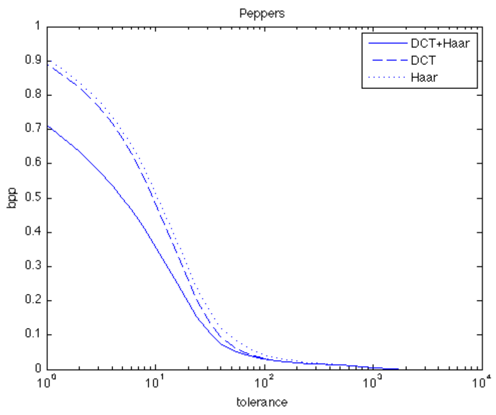
(a) Barbara



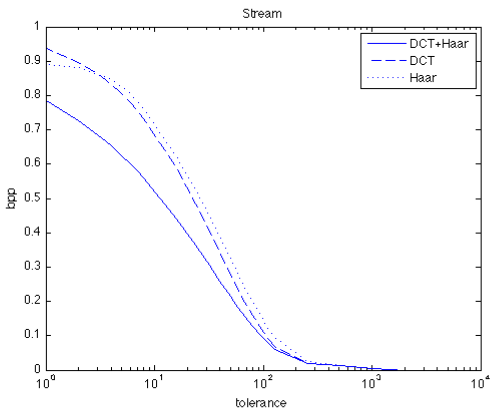
(b) Boat



(c) Elaine

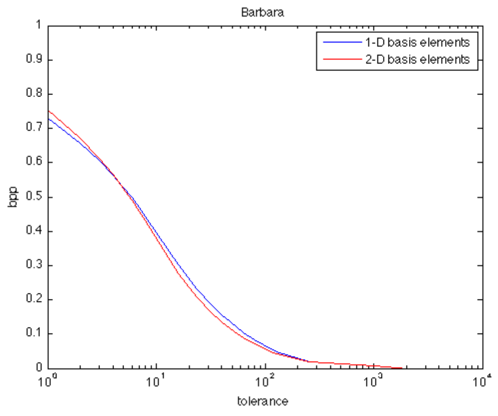


(d) Peppers

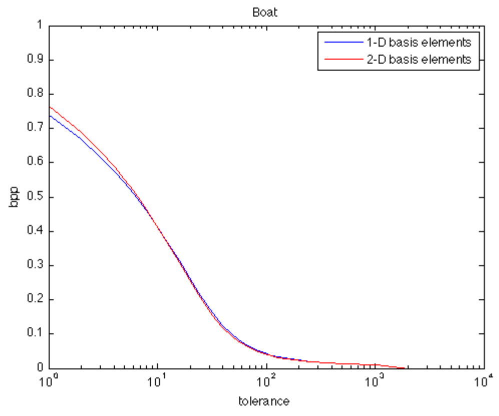


(e) Stream

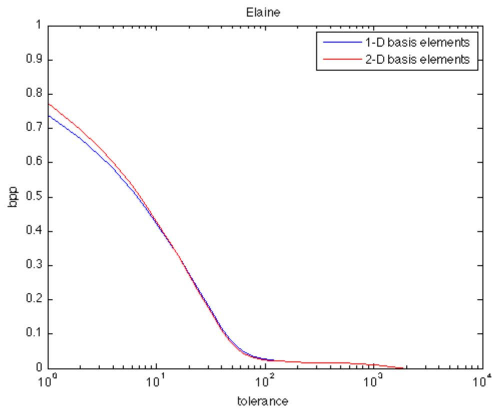
Figure 3.11: Normalized bit-rate vs tolerance: "One-dimensional" basis elements. We can observe that for all images the best normalized bit-rate for a given tolerance is obtained for matrix $\mathbf{A} = [\text{DCT}_1\ \text{Haar}_1]$ which combines both the DCT_1 and Haar_1 bases for $\mathbb{R}^{64}$.



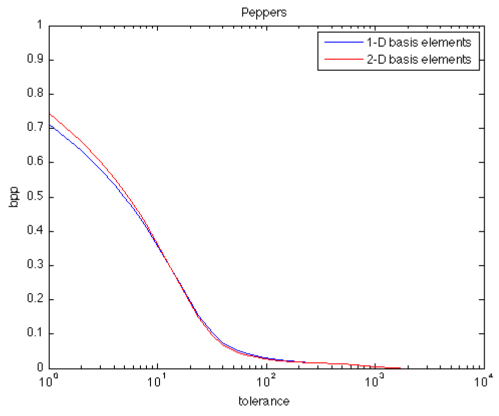
(a) Barbara



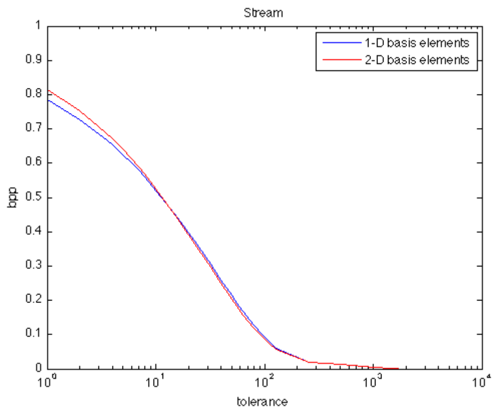
(b) Boat



(c) Elaine



(d) Peppers



(e) Stream

Figure 3.12: Normalized bit-rate vs tolerance. We show results to compare the performance of $\mathbf{A} = [\mathrm{DCT}_1\ \mathrm{Haar}_1]$ and $\tilde{\mathbf{A}} = [\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$.

From the results shown in Figure 3.11, we can see that the DCT_1 basis elements perform better compression for any given tolerance than when using the Haar_1 basis elements, except for image *Stream*: when the tolerance ϵ is approximately less than 3, the Haar_1 basis elements actually result in a smaller normalized bit-rate value. Moreover, and more importantly, combining both the DCT_1 and Haar_1 bases results in better compression than if either basis is used alone. The same is true for $\text{DCT}_{2,3}$ and $\text{Haar}_{2,3}$. This is very encouraging. However, what can be said of the quality of the reconstructed images? For what range of the tolerances tested is the image quality acceptable? What does an “acceptable” image quality mean? To answer these and related questions, we need to address the issues of image reconstruction and error estimation.

3.7 Image reconstruction and error estimation

For the work in this and further sections, unless otherwise noted, we will work with $\mathbf{A} = [\text{DCT}_1\ \text{Haar}_1]$ and the vectorization function c_2.

Given a 512×512 image $\mathcal{I}$ in our database, we can proceed to compress it using the methodology described in Section 3.4. If $\mathcal{I}$ is broken down in 8×8 non-overlapping sub-images $\{\mathbf{Y}_l\}_{l=1,\ldots,4096}$, we can obtain for each of them a corresponding reconstructed sub image $\widetilde{\mathbf{Y}}_l = c_2^{-1}(\tilde{\mathbf{y}}_l)$, and from those reconstruct an approximation $\widetilde{\mathcal{I}}$ to $\mathcal{I}$. Here, $\tilde{\mathbf{y}}_l = \mathbf{A}\mathbf{x}_l$, $\mathbf{x}_l = \texttt{OMP}(\mathbf{A}, \mathbf{y}_l, \epsilon_0)$, and $\mathbf{y}_l = c_2(\mathbf{Y}_l)$, as before. The compression would come from storing wisely $\{\mathbf{x}_l\}$. We can summarize this procedure with the following notation: $\widetilde{\mathcal{I}} = rec(\mathcal{I}, \mathbf{A}, \epsilon_0)$.

How do we assess the quality of $\widetilde{\mathcal{I}}$ when compared to the original image $\mathcal{I}$? We introduce two error estimators—three in actuality, two of them being related.

Traditionally, the signal processing community has relied on the *peak signal-to-noise ratio*, or *PSNR* [22, 30].

Definition 4 (PSNR)**.** *The Peak Signal-to-Noise Ratio between two images $\widetilde{\mathcal{I}}$ and $\mathcal{I}$ is the quantity, measured in dB,*

$$\mathrm{PSNR}(\widetilde{\mathcal{I}}, \mathcal{I}) = 20 \log_{10} \left(\frac{\max_{\mathcal{I}}}{\sqrt{\mathrm{mse}(\widetilde{\mathcal{I}}, \mathcal{I})}} \right),$$

where $\max_{\mathcal{I}}$ is the maximum possible value for any given pixel in $\mathcal{I}$, $\max_{\mathcal{I}} = 2^B - 1$ typically; and $\mathrm{mse}(\widetilde{\mathcal{I}}, \mathcal{I}) = \frac{1}{N_1 N_2} \sum_{i,j} \left(\widetilde{\mathcal{I}}[i,j] - \mathcal{I}[i,j] \right)^2$ is the mean square error between both images. Here N_1 and N_2 represent the dimensions of $\mathcal{I}$, and $\mathcal{I}[i,j]$ represents the value of the pixel at coordinates $[i,j]$ in image $\mathcal{I}$—similarly for $\widetilde{\mathcal{I}}[i,j]$. In our case $N_1 = N_2 = 512$, and $\max_{\mathcal{I}} = 255$.

PSNR has the advantage that it is easy to compute and has widespread use, but it has been criticized for poorly correlating with perceived image quality [26, 27]. However, in recent years extensive work on other error estimators that take into account the human visual system have arisen. In particular, we present and define the *structural similarity* and *mean structural similarity* indices [26].

Definition 5 (SSIM)**.** *Let $\widetilde{\mathcal{I}}$ and $\mathcal{I}$ be two images that have been decomposed in $L \times L$ non-overlapping sub images $\{\widetilde{\mathbf{Y}}_l\}$ and $\{\mathbf{Y}_l\}$, respectively. Then the Structural Similarity index for two corresponding sub-image vectorizations, say $\tilde{\mathbf{y}}_l = c_2(\widetilde{\mathbf{Y}}_l)$*

and $\mathbf{y}_l = c_2(\mathbf{Y}_l)$*, is defined as follows*

$$\mathrm{SSIM}(\tilde{\mathbf{y}}_l, \mathbf{y}_l) = \frac{(2\mu_{\tilde{\mathbf{y}}_l}\mu_{\mathbf{y}_l} + C_1)(2\sigma_{\tilde{\mathbf{y}}_l\mathbf{y}_l} + C_2)}{(\mu_{\tilde{\mathbf{y}}_l}^2 + \mu_{\mathbf{y}_l}^2 + C_1)(\sigma_{\tilde{\mathbf{y}}_l}^2 + \sigma_{\mathbf{y}_l}^2 + C_2)},$$

where $\mu_{\mathbf{y}_l}$ *and* $\sigma_{\mathbf{y}_l}$ *represent the mean and standard deviation of* $\mathbf{y}_l$*, respectively; and similarly for* $\widetilde{\mathbf{y}}_l$*. The term* $\sigma_{\tilde{\mathbf{y}}_l\mathbf{y}_l}$ *is the correlation between* $\tilde{\mathbf{y}}_l$ *and* $\mathbf{y}_l$*. The values* C_1 *and* C_2 *are two small constants.*

For our purposes, we used the default values of $L = 11$, $C_1 = 0.01$, and $C_2 = 0.03$ used in [26] when assessing the SSIM of an image in our database and its reconstruction. We used a value of $L = 4$ when we modified OMP to use internally the SSIM as a stopping criteria. More on this later.

From the above definition, we can see that the SSIM index is a localized quality measure that can be represented on a plane that maps its values. It can take values from 0 to 1 and when it takes the value of 1 the two images are identical. In practice, we usually require a single overall quality of measure for the entire image. In that case we use the mean SSIM index to evaluate the overall image quality.

Definition 6 (MSSIM)**.** *Let* $\widetilde{\mathcal{I}}$ *and* $\mathcal{I}$ *be two images, where the former is the approximation and the later is the original. Then the Mean Structural Similarity index is*

$$\mathrm{MSSIM}(\widetilde{\mathcal{I}}, \mathcal{I}) = \frac{1}{M}\sum_{l=1}^{M} \mathrm{SSIM}(\tilde{\mathbf{y}}_l, \mathbf{y}_l),$$

where $\tilde{\mathbf{y}}_l$ *and* $\mathbf{y}_l$ *are the image contents at the* l*-th local sub-image, and* M *is the number of local sub-images in the image.*

Finally, we take a look at the relationship between the size of the sub-image and the tolerance, and how this affects the quality of the approximation. We analyze the idealized error distribution in which all pixels of the approximation are c units apart from the original. Consider an $L \times L$ sub image that has been linearized to a vector $\mathbf{y}$ of length L^2. Assume that the OMP approximation within ϵ has distributed the error evenly, that is, if $\mathbf{x} = \texttt{OMP}(\mathbf{A}, \mathbf{y}, \epsilon)$ and $\tilde{\mathbf{y}} = \mathbf{A}\mathbf{x}$, then

$$\begin{aligned}
\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2 < \epsilon \Leftrightarrow & \|\tilde{\mathbf{y}} - \mathbf{y}\|_2^2 < \epsilon^2, \\
& \Leftrightarrow \sum_{j=1}^{L^2} \left(\tilde{\mathbf{y}}(j) - \mathbf{y}(j)\right)^2 < \epsilon^2, \\
& \Leftrightarrow L^2 c^2 < \epsilon^2, \\
& \Leftrightarrow c < \frac{\epsilon}{L}.
\end{aligned} \tag{3.19}$$

That is, if we want to be within c units from each pixel, we have to choose a tolerance ϵ such that $c = \epsilon / L$.

We note that the least-squares approximation at the core of OMP approximates the idealized error distribution. This can be seen in Figure 3.13 where the black dashed line represents this idealized error approximation. For tolerances $\epsilon > 40$, we can see that the PSNR for all images considered is above this idealized error distribution. This can be explained by noting that, for example, for $\epsilon = 2048$, we would have from Equation 3.19 that $c = 2048/8 = 256$, but the maximum pixel value is only 255. Therefore, unless the original image $\mathcal{I}$ is just a white patch, the initial value of the OMP approximation being an all black image, there are matching pixels in the original and the approximation image $\widetilde{\mathcal{I}} = rec(\mathcal{I}, \mathbf{A}, 2048)$ that are less than 256 units apart. This would necessarily, by Definition 4, imply PSNR$(\widetilde{\mathcal{I}}, \mathcal{I}) > 0$, a value above the value of the PSNR for the idealized error distribution when $\epsilon = 2048$, which is a small negative value.

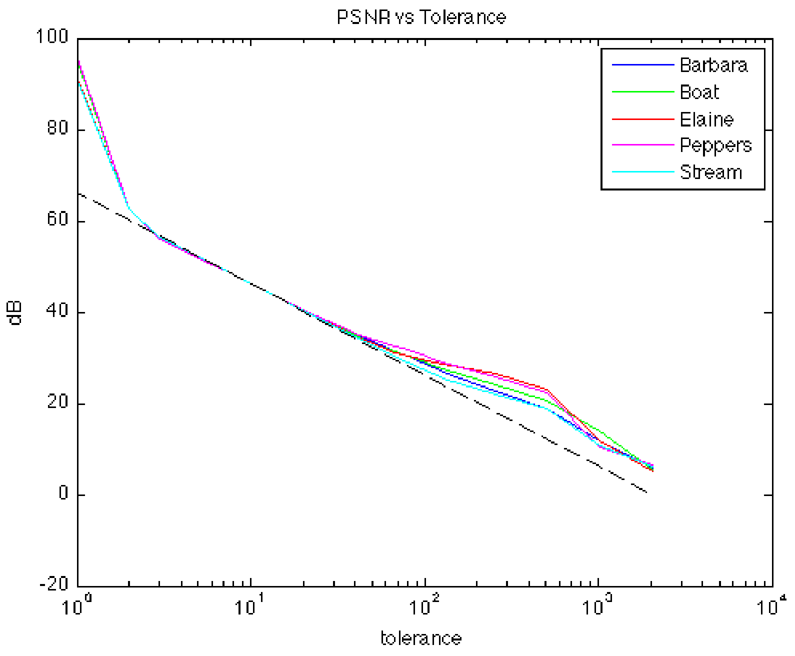


Figure 3.13: Peak Signal-to-Noise Ratio vs tolerance. We observe three typical behaviors for all images. For large values of the tolerance, about $\epsilon > 40$, the PSNR of all images is above the PSNR value for the idealized error distribution marked by the black dashed line. This behavior is also observed for very small values of the tolerance, about $\epsilon < 3$. Then for values between these two extreme behaviors, all images conform very closely to the idealized error distribution, a fact that is expected from the least-squares approximation at the core of the OMP algorithm.

On the other hand, for really small tolerances, about $\epsilon < 3$, we observe that the PSNR value for all images jumps again above the PSNR for the idealized error model. This is a happy case when roundoff error actually helps. What happens is that for such small tolerances, the roundoff to the closest integer for all entries in $\tilde{\mathbf{y}}_l = \mathbf{A}\mathbf{x}_l$ when we form the sub image approximation $\widetilde{\mathbf{Y}}_l = c_2^{-1}(\tilde{\mathbf{y}}_l)$, coincides with the true value of the pixels in the original sub image $\mathbf{Y}_l$. Again, by Definition 4, this increases the value of PSNR$(\widetilde{\mathcal{I}}, \mathcal{I})$ compared to the case where roundoff would not have taken place.

3.8 PSNR and MSSIM comparison

Now that we have some tools to asses the quality of a reconstruction, how do they compare?

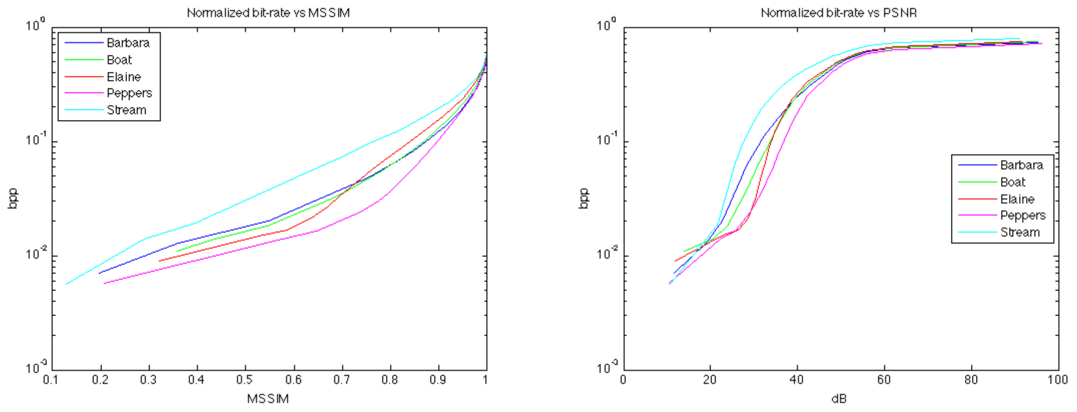


Figure 3.14: Normalized bit-rate vs MSSIM, PSNR

In Figure 3.14 we have plotted the normalized bit-rate versus both error indices MSSIM and PSNR. The first thing that we observe is that the sensitivity for PSNR varies more dramatically than the sensitivity for MSSIM over the range of tolerances chosen.

From Figure 3.15 we can observe that for the range of 20 to 40 dB in PSNR, the MSSIM index ranges from about 0.33 to 0.98. Since a value of 1 in MSSIM corresponds to two identical images, we can focus on values of PSNR no greater than 40 dB in our analysis. Also in Figure 3.15 we corroborate the criticism that

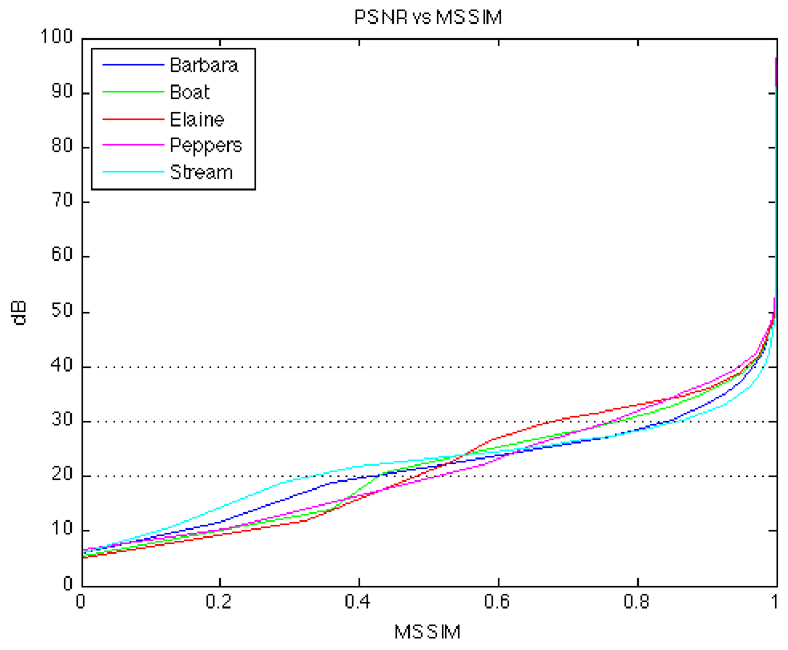


Figure 3.15: Peak Signal-to-Noise Ratio vs Mean Structural Similarity

has been addressed to PSNR as a measure of image quality. For example, for image *Stream* at 20 dB we have an MSSIM value of 0.33, whereas for image *Peppers* we have an MSSIM value of 0.51. A similar wide range between 0.69 (*Elaine*) and 0.86 (*Stream*) for MSSIM is observed for 30 dB in PSNR. It is not until 40 dB that we have a much smaller range of MSSIM values, 0.95 (*Peppers*) to 0.98 (*Stream*). Therefore, if SSIM and MSSIM capture more accurately the human visual system's perception of image quality, then the PSNR index is shown to be not so good at it until after values larger than or equal to 35 dB.

We therefore drop from the rest of our analysis the PSNR index, other than for an occasional reference point or comparison, and focus on the SSIM and MSSIM indices. We retake the questions at the end of Section 3.6 and answer them with Figure 3.16. From it, if we were to consider desirable values of MSSIM to be above or equal to 0.9, we would see that this would correspond to a tolerance ϵ of less than 32 for the *Peppers* and of 48 for *Stream*, all other tolerances for the rest of the images falling in between these two values.

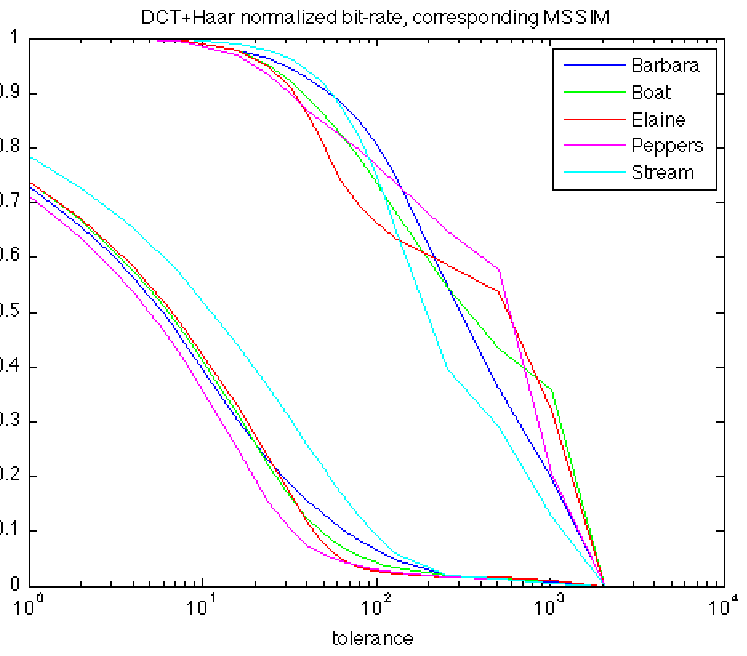
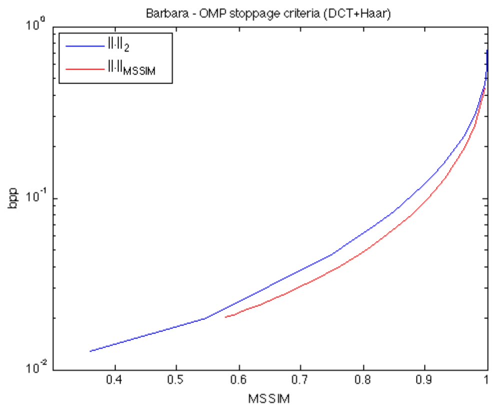


Figure 3.16: Normalized bit-rate and corresponding MSSIM vs tolerance. In this graph we have plotted together the best normalized bit-rate obtained by combining the DCT and Haar bases, and the corresponding value of the MSSIM index for a given tolerance. The normalized bit-rate graphs are on the bottom left, and the MSSIM index values are above these. This figure combines results for all images.

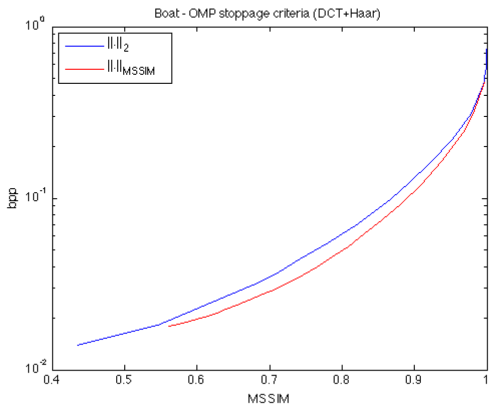
This means that if we wanted all images to have an MSSIM index of 0.9 or better, we would have to pick a tolerance no larger than $\epsilon = 32$. This tolerance corresponds, according to Equation 3.19, to a distance of no more than $32/8 = 4$ units per pixel between the reconstructed image and the original, on average. Under these circumstances we would achieve a normalized bit-rate of 0.102 to 0.305 bits per pixel. But how good would images with MSSIM greater than or equal to 0.9 actually look? Moreover, what if we could modify OMP as to guarantee a certain minimum MSSIM quality level? It turns out that this modification is possible.

Consider the following change in the termination condition for the OMP algorithm from $\|\mathbf{Ax} - \mathbf{b}\|_2 < \epsilon_0$ to $\|\mathbf{Ax} - \mathbf{b}\|_{MSSIM} \equiv \text{MSSIM}(c_2^{-1}(\mathbf{Ax}), c_2^{-1}(\mathbf{b})) > \delta_0$, where δ_0 is a desired minimum MSSIM index value to achieve in each individual sub image of the reconstruction of $\mathcal{I}$. When we make this change, and recompute the normalized bit-rate vs MSSIM graphs, we obtain the plots shown in Figure 3.17. In this figure, we observe that changing the termination condition for OMP leads to an improvement in the normalized bit-rate without sacrificing image quality. Or, to see this from the opposite perspective, given a normalized bit-rate, we can achieve a better image quality index MSSIM when we use the new stopping criteria. As we shall see in the pictures below, this change redistributes the work that OMP performs more evenly across the image.

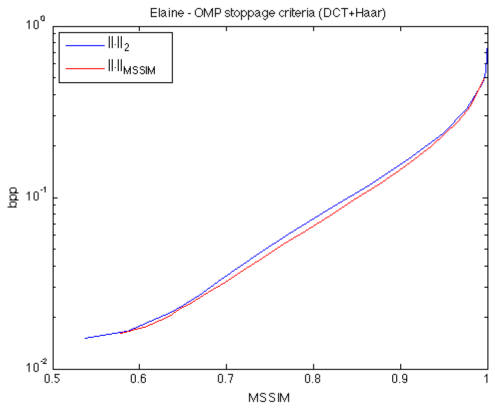
Finally, to address the question of how good the images actually look, we let the images speak for themselves, and let you—the reader—be the judge. See Figures 3.18 through 3.25. All figures consist of two images, the reconstruction from the original, to the left, and the SSIM index map to the right. The SSIM map represents the localized quality of the image reconstruction. Lighter values are values closer to 1 (“white” = 1), whereas darker values are values closer to 0 (“black” = 0). For each image we obtained a reconstruction for $\epsilon_0 = 32$ for the ℓ^2 stopping criteria, and a reconstruction with value of δ_0 close to the MSSIM from the former for the MSSIM stopping criteria, for comparison purposes.



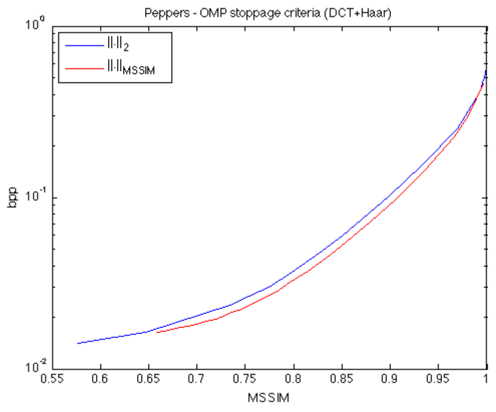
(a) Barbara



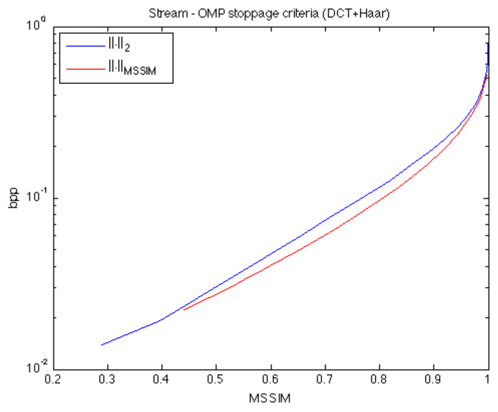
(b) Boat



(c) Elaine

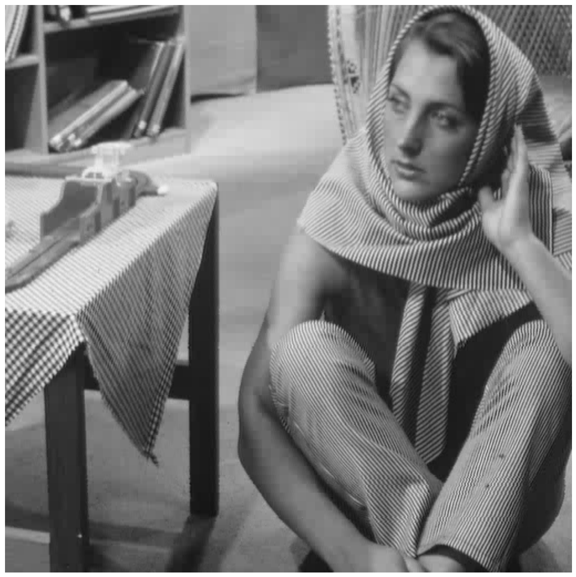


(d) Peppers



(e) Stream

Figure 3.17: Normalized bit-rate vs MSSIM. Comparison of different termination criteria for OMP.


(a) Barbara

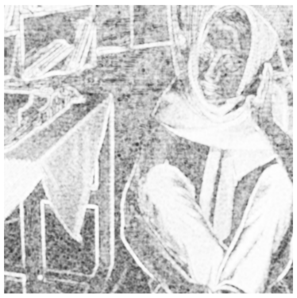
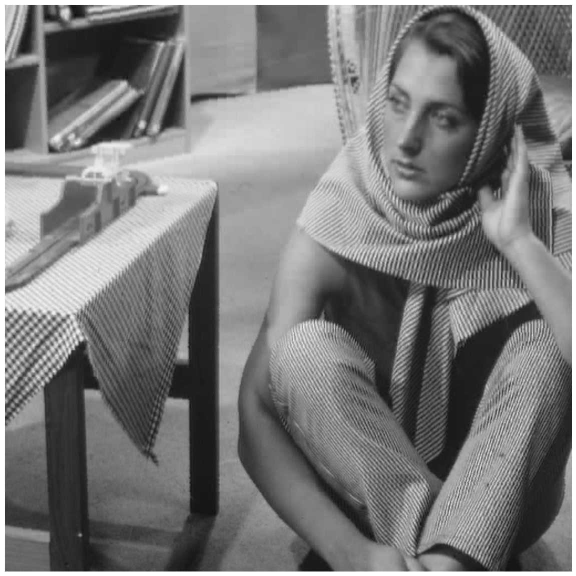

(b) SSIM

Figure 3.18: Barbara: $\epsilon_0 = 32$, PSNR = 36.9952 dB, MSSIM = 0.9444, normalized bit-rate = 0.1863 bpp, termination criteria: $\| \cdot \|_2$.


(a) Barbara

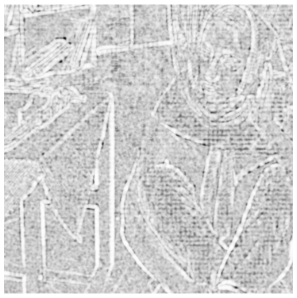
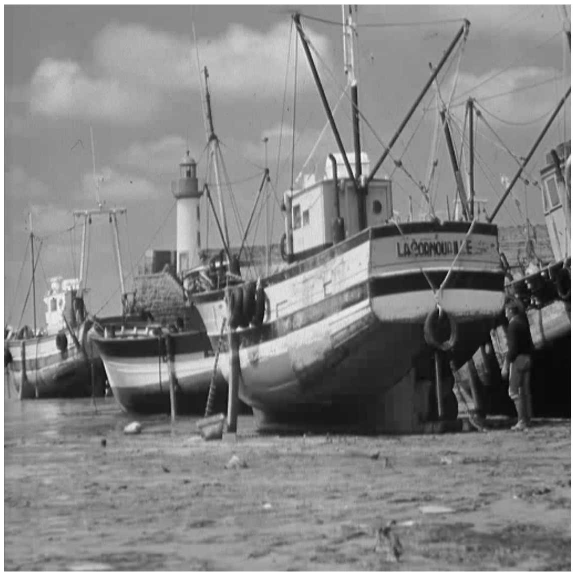

(b) SSIM

Figure 3.19: Barbara: $\delta_0 = 0.94$, PSNR = 32.1482 dB, MSSIM = 0.9462, normalized bit-rate = 0.1539 bpp, termination criteria: $\| \cdot \|_{MSSIM}$.


(a) Boat

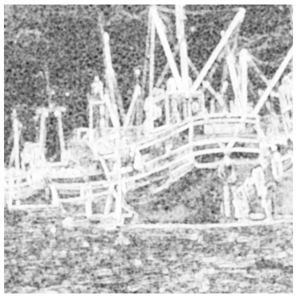
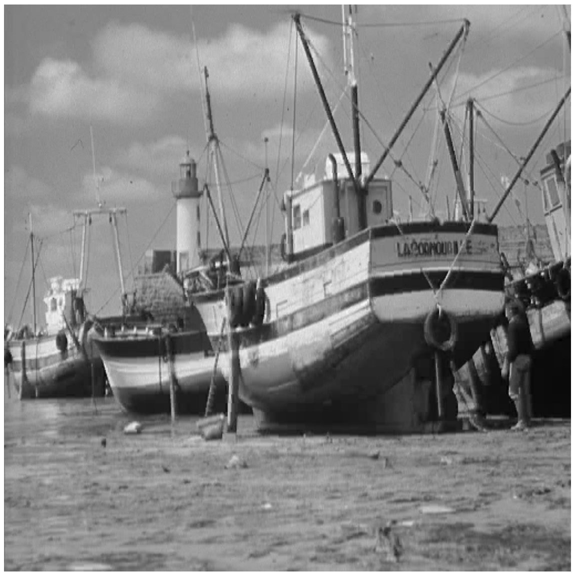

(b) SSIM

Figure 3.20: Boat: $\epsilon_0 = 32$, PSNR = 36.6020 dB, MSSIM = 0.9210, normalized bit-rate = 0.1608 bpp, termination criteria: $\| \cdot \|_2$.


(a) Boat

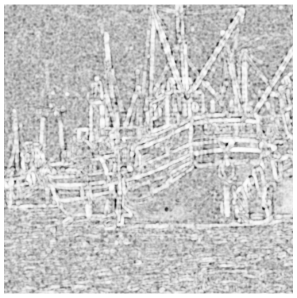

(b) SSIM

Figure 3.21: Boat: $\delta_0 = 0.92$, PSNR = 34.1405 dB, MSSIM = 0.9351, normalized bit-rate = 0.1595 bpp, termination criteria: $\| \cdot \|_{MSSIM}$.

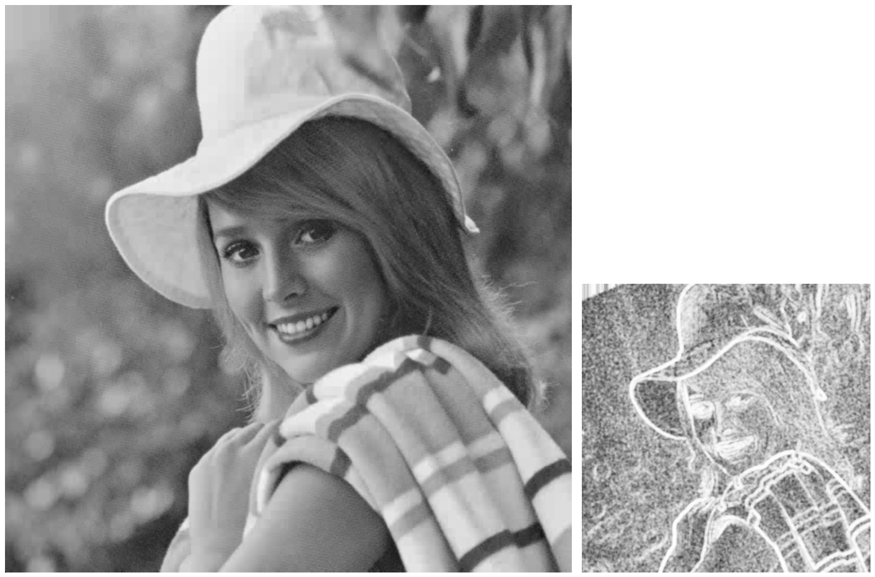

(a) Elaine (b) SSIM

Figure 3.22: Elaine: $\epsilon_0 = 32$, PSNR = 36.5155 dB, MSSIM = 0.9096, normalized bit-rate = 0.1682 bpp, termination criteria: $\| \cdot \|_2$.

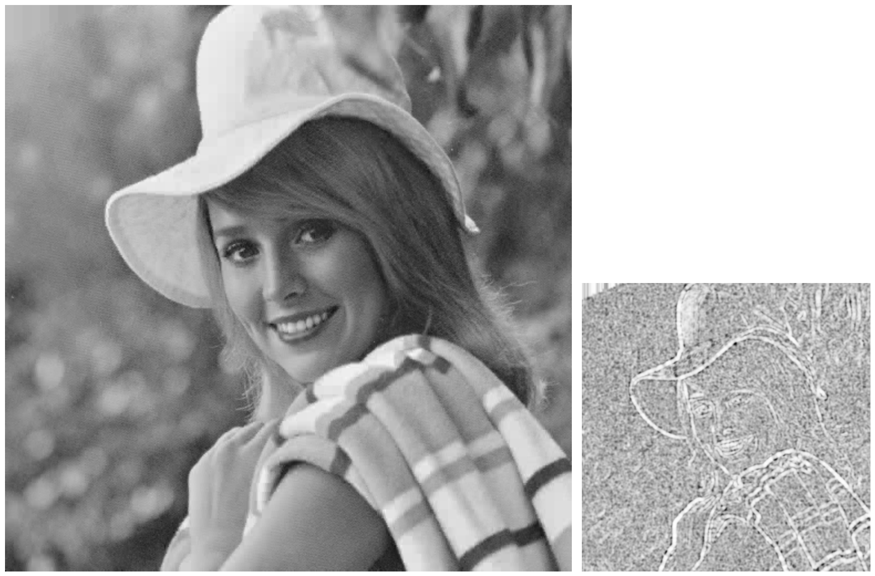

(a) Elaine (b) SSIM

Figure 3.23: Elaine: $\delta_0 = 0.90$, PSNR = 35.6288 dB, MSSIM = 0.9168, normalized bit-rate = 0.1686 bpp, termination criteria: $\| \cdot \|_{MSSIM}$.

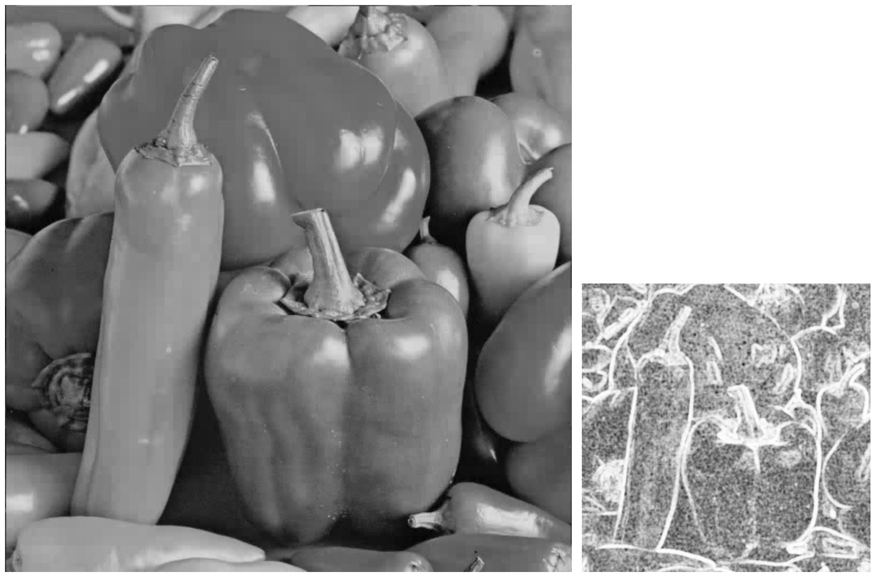

(a) Peppers (b) SSIM

Figure 3.24: Peppers: $\epsilon_0 = 32$, PSNR = 36.8674 dB, MSSIM = 0.8983, normalized bit-rate = 0.1024 bpp, termination criteria: $\| \cdot \|_2$.

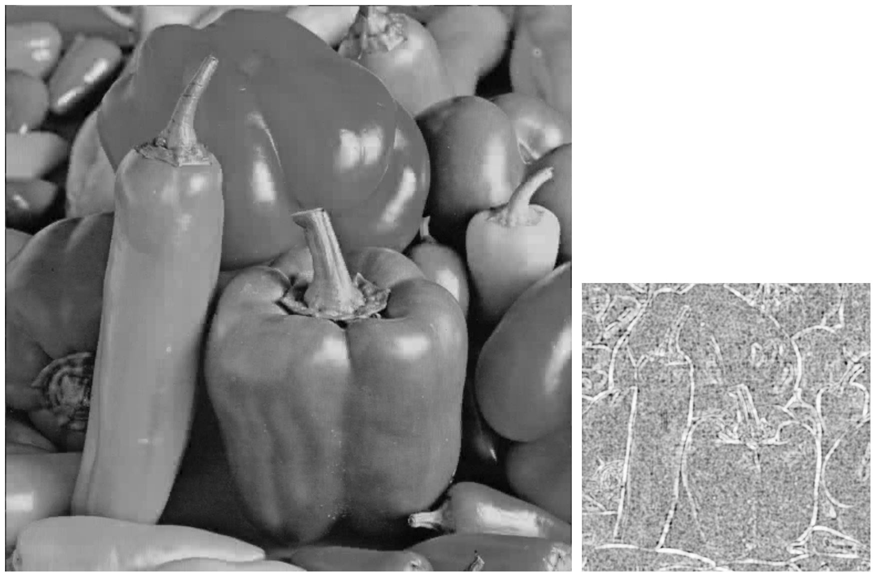
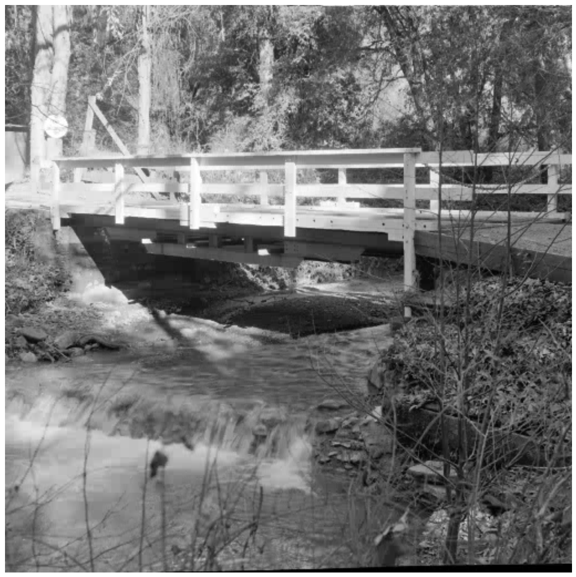
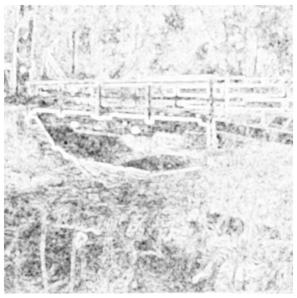
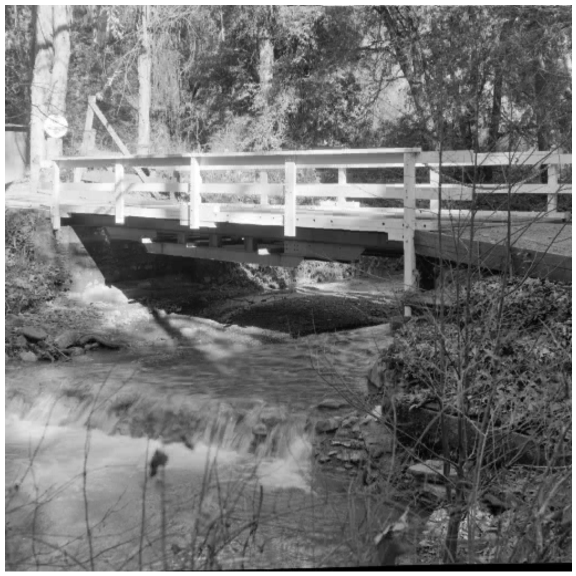
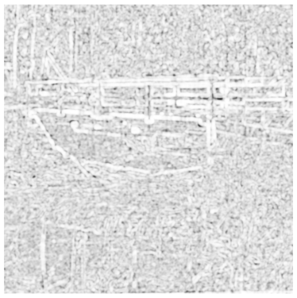

(a) Peppers (b) SSIM

Figure 3.25: Peppers: $\delta_0 = 0.89$, PSNR = 35.4309 dB, MSSIM = 0.9080, normalized bit-rate = 0.1011 bpp, termination criteria: $\| \cdot \|_{MSSIM}$.

(a) Stream (b) SSIM

Figure 3.26: Stream: $\epsilon_0 = 32$, PSNR = 36.4686 dB, MSSIM = 0.9622, normalized bit-rate = 0.3050 bpp, termination criteria: $\| \cdot \|_2$.

(a) Stream (b) SSIM

Figure 3.27: Stream: $\delta_0 = 0.95$, PSNR = 34.1398 dB, MSSIM = 0.9634, normalized bit-rate = 0.2853 bpp, termination criteria: $\| \cdot \|_{MSSIM}$.

3.9 Other matrices

We have devoted a lot of attention to specific matrices $\mathbf{A}$ to solve problem (P_0^ϵ), see Equation 1.2, which we reproduce below for convenience,

$$(P_0^\epsilon): \qquad \min_{\mathbf{x}} \|\mathbf{x}\|_0 \quad \text{subject to} \quad \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2 < \epsilon.$$

Namely, we have studied [DCT$_1$ Haar$_1$], and [DCT$_{2,j}$ Haar$_{2,j}$], for $j = 1, 2, 3$, defined in Sections 3.2.1 and 3.2.2, respectively. But there are many other matrices we could potentially use, an infinite number in fact. We briefly present here some results for other matrices that could be of interest but for which further study would be desirable.

3.9.1 "Rotations" of the basis elements of DCT$_{2,3}$

Consider the matrix DCT$_{2,3}$ defined in Section 3.2.2, and recall that its columns are obtained from the outer products of DCT waveforms, see Section 3.2.1, and Equations 3.6 and 3.7. We recall that the two-dimensional DCT-II transform plays a fundamental role in the definition of the column vectors of matrix DCT$_{2,3}$, in specific see Equation 3.5, which we reproduce below for convenience,

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n1,n2} \cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right] \cos\left[\frac{\pi}{N_2}\left(n_1+\frac{1}{2}\right)k_2\right].$$

As pointed out in Section 3.2.2, at the core of this transform we find the family of functions indexed by k_1 and k_2,

$$g_{k_1,k_2,N_1,N_2}(x,y) = \cos\left[\pi\left(x + \frac{1}{2N_1}\right)k_1\right]\cos\left[\pi\left(y + \frac{1}{2N_2}\right)k_2\right],$$

sampled on all points $(x,y) \in \mathcal{S}(N_1) \times \mathcal{S}(N_2)$, where in our case, $N_1 = N_2 = 8$, and $k_1, k_2 \in \{0,\ldots,7\}$. Recall that $\mathcal{S}(N) = \left\{s_i \in [0\ 1) : s_i = \frac{i}{N},\ i = 0,\ldots,N-1\right\}$.

Now observe that given that we are fundamentally sampling the function $g_{k_1,k_2,N_1,N_2} : \mathbb{R}^2 \to \mathbb{R}$ that takes a vector $\mathbf{v} = (x,y)^{\mathrm{T}} \in \mathbb{R}^2$ and maps it to $g_{k_1,k_2,N_1,N_2}(x,y) \in \mathbb{R}$, we could transform $\mathbf{v}$ first by applying to it a rotation in the $\mathbb{R}^2$ plane, and then apply to the transformed vector the function g_{k_1,k_2,N_1,N_2}. Let $\rho(\theta)$ be the counter-clockwise rotation by angle θ in the $\mathbb{R}^2$ plane, and identify it with its matrix representation, i.e.,

$$\rho(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}.$$

Then, we have the following construction for a new matrix $\mathbf{A}$. Let $\theta_{n,N} = \frac{\pi}{2}\frac{n}{N}$, with $n = 0,\ldots,N-1$, be N equidistant points in $\left[0, \frac{\pi}{2}\right)$. For each ordered pair $(k_1,k_2) \in \{0,\ldots,7\} \times \{0,\ldots,7\}$, and setting vector $\mathbf{v}_{i,j} = \left(\frac{i}{8}, \frac{j}{8}\right)^{\mathrm{T}} \in \mathcal{S}(8) \times \mathcal{S}(8)$, we can form the matrix,

$$\mathbf{W}_{k_1,k_2}(\theta_{n,N}) = \begin{pmatrix} g_{k_1,k_2,8,8}\big(\rho(\theta_{n,N})\mathbf{v}_{0,0}\big) & \ldots & g_{k_1,k_2,8,8}\big(\rho(\theta_{n,N})\mathbf{v}_{0,7}\big) \\ \vdots & & \vdots \\ g_{k_1,k_2,8,8}\big(\rho(\theta_{n,N})\mathbf{v}_{7,0}\big) & \ldots & g_{k_1,k_2,8,8}\big(\rho(\theta_{n,N})\mathbf{v}_{7,7}\big) \end{pmatrix} \in \mathbb{R}^{8\times 8}.$$

Using this matrix, form the column vector,

$$\mathbf{w}_{k_1,k_2,j}(\theta_{n,N}) = c_j\big(\mathbf{W}_{k_1,k_2}(\theta_{n,N})\big),$$

where the vectorization functions c_j were defined in Section 3.4, see Figures 3.4 and 3.5.

Choosing $j = 3$, we can compare $\mathbf{w}_{k_1,k_2,3}(\theta_{n,N})$ with $\widetilde{\mathbf{w}}_{k_1,k_2,3}$ in Equation 3.7. Basically, when $n = 0$, we have $\mathbf{w}_{k_1,k_2,3}(\theta_{0,N}) = \mathbf{w}_{k_1,k_2,3}(0) = \widetilde{\mathbf{w}}_{k_1,k_2,3}$. We recall that $\{\widetilde{\mathbf{w}}_{k_1,k_2,3} : k_1, k_2 = 0, \ldots, 7\}$ form the columns of the $\mathrm{DCT}_{2,3}$ submatrix in $\mathbf{A} = [\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$, hence the name for this section.

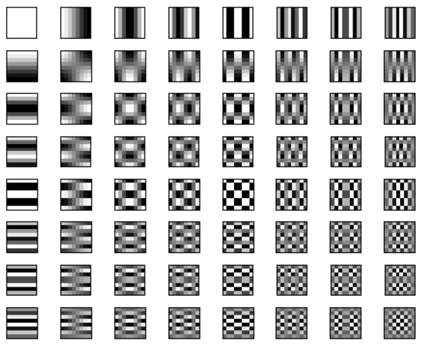
Now, finally, traversing k_1 and k_2 in lexicographical order for the sequence of ordered pairs (k_1, k_2) with $k_1, k_2 = 0, \ldots, 7$ and k_2 moving faster than k_1, i.e., $(0,0), (0,1), \ldots, (0,7), (1,0), \ldots, (7,7)$, we can form a new matrix,

$$\mathrm{DCT}_{2,3}(n, N) = \big(\mathbf{w}_{0,0,3}(\theta_{n,N}) \ldots \mathbf{w}_{7,7,3}(\theta_{n,N})\big).$$

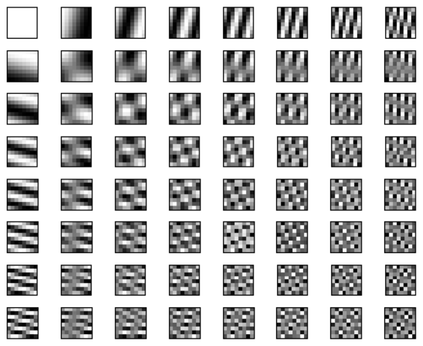
Suppose that we choose $N = 6$, and that we concatenate the six matrices $\{\mathrm{DCT}_{2,3}(n, 6)\}_{n=0}^{5}$, to form the matrix below,

$$\mathbf{A} = [\mathrm{DCT}_{2,3}(0, 6) \ldots \mathrm{DCT}_{2,3}(5, 6)]. \tag{3.20}$$

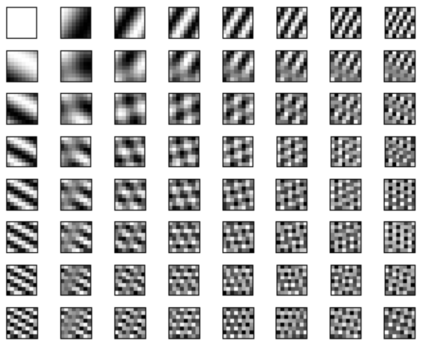
This is a matrix that uses “rotations”—as introduced above—of some of the column vectors of $[\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$ that could be used in problem (P_0^ϵ), for example. We have run experiments for comparison purposes for a tolerance $\epsilon = 32$ using



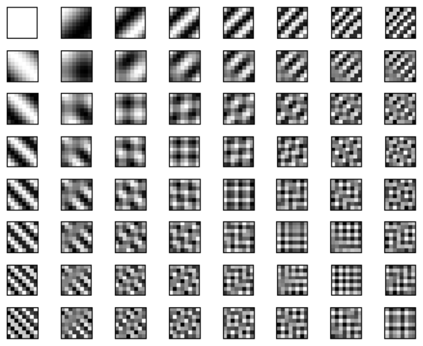
(a)



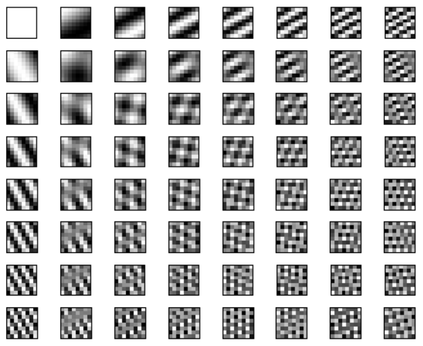
(b)



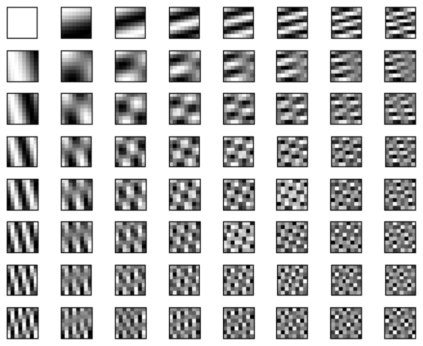
(c)



(d)



(e)



(f)

Figure 3.28: Full natural 2D representation for (a) $\mathrm{DCT}_{2,3}(0,6)$, (b) $\mathrm{DCT}_{2,3}(1,6)$, (c) $\mathrm{DCT}_{2,3}(2,6)$, (d) $\mathrm{DCT}_{2,3}(3,6)$, (e) $\mathrm{DCT}_{2,3}(4,6)$, and (f) $\mathrm{DCT}_{2,3}(5,6)$, bases for $\mathbb{R}^{64}$. White corresponds to the maximum value achieved by the basis element, black to the minimum. The intermediate shade of gray corresponds to 0.

A as defined in Equation 3.20. The results are shown in Table 3.5. For a visual representation of the basis elements of **A**, see Figure 3.28.

Image	PSNR (dB)	Normalized bit-rate (bpp)	MSSIM
Barbara	37.1180	0.1325	0.9464
Boat	36.6579	0.1321	0.9234
Elaine	36.5671	0.1359	0.9118
Peppers	36.9145	0.0859	0.9001
Stream	36.5089	0.2482	0.9636

Table 3.5: Performance results for $\mathbf{A} = [\mathrm{DCT}_{2,3}(0,6)\ldots\mathrm{DCT}_{2,3}(5,6)]$. In all cases both the PSNR and normalized bit-rate values were better than the values obtained for $\mathbf{A} = [\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$, i.e., larger and smaller, respectively. See Table 3.4 for comparison. The values correspond to runs of OMP with an ℓ^2 termination criteria, and a tolerance $\epsilon = 32$.

Chapter 4

Quantization and coding

In 1948 Claude E. Shannon published a seminal paper in two parts in *The Bell Systems Technical Journal* named “A mathematical theory of communication” [16]. This work marked the dawn of both information theory and coding theory [8, 9]. In it, Shannon defined precisely what “information” meant mathematically. He drew from work done by his colleagues at Bell Labs, Harry Nyquist and Ralph Hartley, and he certainly was aware of the work on the topic by Norbert Wiener, with whom he had taken a class at MIT.

Hartley wrote the following equation to quantify “information” in a discrete setting,

$$H = n \log s,$$

where H is the amount of information, n is the number of symbols transmitted, and s is the size of a given alphabet from which the symbols are drawn [5].

Shannon pushed this notion by identifying the amount of information with entropy, as Norbert Wiener commented on a five paragraph review in *Physics Today*, September 1950, of the book that Shannon and Weaver published together in 1949. The book, “The mathematical theory of communication” [17], contained in a single tome the two-part original paper by Shannon and an expanded and slightly more technical essay that Warren Weaver, the director of natural sciences for the Rockefeller Foundation, had written earlier for *Scientific American* in 1949 [28] about Shannon's work.

More specifically, in the case of a discrete information source, Shannon represented this as a Markov process, and asked if we could "define a quantity which will measure, in some sense, how much information is 'produced' by such a process, or better, at what rate information is produced?"

He equated this quantity to entropy,

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i, \tag{4.1}$$

where in Equation 4.1 we have supposed that we have a set of n possible events whose probabilities of occurrence are $p_1, p_2, \ldots, p_n$. To develop some intuition about this definition, we review and compare the material in [8] with Shannon's work [16]. Assume that we are given a random variable X on the finite set $\{1, 2, \ldots, n\}$ with probability distribution p. The elements $X(1) = x_1, X(2) = x_2, \ldots, X(n) = x_n$ are distinct and $p(x_1), p(x_2), \ldots, p(x_n)$ are nonnegative real numbers with

$$p(x_1) + p(x_2) + \ldots + p(x_n) = 1.$$

We are using $p_i = p(x_i)$ as a shorthand for $\text{prob}(X = x_i)$. Now think of the following. The smaller the probability $p(x_i)$, the more uncertain we are that an observation of X will result in x_i. Thus we can regard $1/p(x_i)$ as a measure of the uncertainty of x_i. The smaller the probability, the larger the uncertainty.

Shannon thought of uncertainty as information (contrary to common intuition.) This is because if an event has probability 1, there is no information gained in asking the outcome of such an event given that the answer will always be the same. Moreover, he also thought the following three properties as desirable for a function that could quantify information:

1. H should be continuous in the p_i.
2. If all the p_i are equal, $p_i = \frac{1}{n}$, then H should be a monotonic increasing function of n. With equally likely events there is more choice, or uncertainty, when there are more possible events.
3. If a choice can be broken down into two successive choices, the original H should be the weighted sum of the individual values of H.

Shannon proved that the only H satisfying these three assumptions is of the form

$$H = -K \sum_{i=1}^{n} p_i \log p_i.$$

Hence, if we define the *uncertainty* of x_i to be

$$\log_2 \frac{1}{p(x_i)} = -\log_2 p(x_i),$$

measured in *bits*, the *entropy* of the random variable X is defined to be the expected value

$$H(X) = \sum_{i=1}^{n} p(x_i) \log_2 \frac{1}{p(x_i)} = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i),$$

of the uncertainty for the random variable X, i.e., the entropy of X will measure the information gained from observing X. This reasoning establishes Equation 4.1.

Moreover, given a communication channel—like a pair of twisted copper for telephony—Shannon identified a number called the capacity of the channel and proved, in a nonconstructive way, that arbitrarily reliable communication is possible at any rate below the channel capacity. For example, he defined the capacity C of the discrete noiseless channel as

$$C = \lim_{T \to \infty} \frac{\log N(T)}{T},$$

where $N(T)$ is the number of allowed signals of duration T.

The link between entropy and channel capacity was stated by Shannon in the following theorem.

Theorem 9 (Shannon, Fundamental Theorem for a Noiseless Channel [16]). *Let a source have entropy H (bits per symbol) and a channel have a capacity C (bits per second). Then it is possible to encode the output of the source in such a way as to transmit at the average rate $\frac{C}{H} - \epsilon$ symbols per second over the channel where ϵ is arbitrarily small. It is not possible to transmit at an average rate greater than $\frac{C}{H}$.*

However, in reality, all communication channels exhibit some noise. That is, one cannot expect a communication channel to transmit with 100% fidelity the source message. Shannon included this fact in his schematic diagram for a general communication system, which we reproduce below.

Notwithstanding the limitation imposed by noise in the channel, Shannon

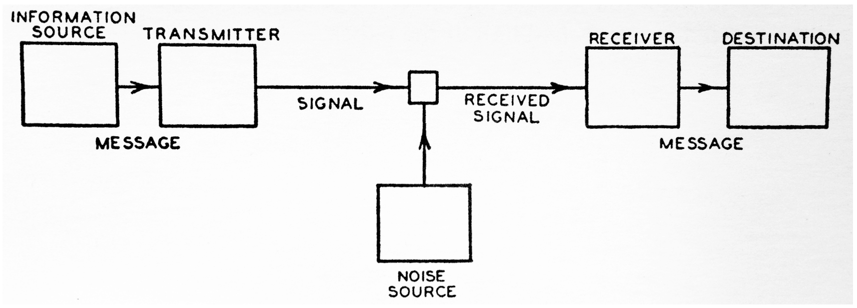


Figure 4.1: Schematic diagram of a communication system, from [16].

also proved that we can be as sure as we want (but not absolutely sure!) of the information in a received message while transmitting at any rate below channel capacity. The conclusions of Shannon's theorem are not without some trade-off. In order to communicate reliably at a rate close to channel capacity, we must increase the complexity of our communication scheme [8, 10]. To a large extent, coding theory has been the quest for these good codes.

"Quantization", the extraordinary invited paper on the occasion of the 50th anniversary of "A mathematical theory of communication" surveying this quest from an engineering perspective by Robert Gray and David Neuhoff [6], describes the history of the theory and practice of quantization up until 1998. In this paper the term *quantization* encompasses *transform coding* as one of several approaches to exploit redundancy in the source signal. The topic of redundancy in messages was of great interest to Shannon and he studied it in [16] as well. In transform encoding, the source samples are collected into a vector of, say, dimension n that is multiplied by an orthogonal matrix (an orthogonal transform) and the resulting transform coefficients are scalar quantized.

A scalar quantizer can be defined as consisting of a set of intervals or *cells*

$\mathcal{S} = \{S_i \subset \mathbb{R} : i \in I\}$ that form a partition of the real line, where the index set I is ordinarily a collection of consecutive integers beginning with 0 or 1, together with a set of *reproduction values* or *points* or *levels* $\mathcal{C} = \{y_i \in \mathbb{R} : i \in I\}$ so that the overall quantizer q is defined by $q(x) = y_i$ for $x \in S_i$, which can be expressed concisely as

$$q(x) = \sum_{i \in I} y_i \mathbb{1}_{S_i}(x), \tag{4.2}$$

where the indicator function $\mathbb{1}_S(x)$ is 1 if $x \in S$ and 0 otherwise [6].

More generally, a vast class of memoryless quantizers can be described as follows. A quantizer of dimension k, a positive integer, takes as input a vector $\mathbf{x} = (x_1, \ldots, x_k)^{\mathrm{T}} \in \mathcal{A} \subset \mathbb{R}^k$. Memoryless refers to a quantizer which operates independently on successive vectors. The set $\mathcal{A}$ is called the *alphabet* and it is often called the *support* of the source distribution. If $k = 1$ the quantizer is *scalar*, otherwise it is *vector*. The quantizer consists then of three components—a *lossy encoder* $\alpha : \mathcal{A} \to I$, where the index set I is an arbitrary countable set, usually taken as a collection of consecutive integers, a *reproduction decoder* $\beta : I \to \hat{\mathcal{A}}$, where $\hat{\mathcal{A}} \subset \mathbb{R}^k$ is the *reproduction alphabet*, and a *lossless encoder* $\gamma : I \to J$, an invertible mapping (with probability 1) into a collection J of variable-lenght binary vectors that satisfies the *prefix condition*, that is, no vector in J can be the prefix of any other vector in the collection [6].

Alternatively, a lossy encoder is specified by a partition $\mathcal{S} = \{S_i \subset \mathbb{R} : i \in I\}$ of $\mathcal{A}$; a reproduction decoder is specified by a *codebook* $\mathcal{C} = \{\beta(i) \in \hat{\mathcal{A}} : i \in I\}$ of *points*, *codevectors*, or *reproduction codewords*, also known as the *reproduction* *codebook*; and the lossless encoder γ can be described by its *binary codebook* $J = \{\gamma(i) : i \in I\}$ containing *binary* or *channel codewords*. The *quantizer rule* is the function $q(\mathbf{x}) = \beta(\alpha(\mathbf{x}))$ or, equivalently, $q(\mathbf{x}) = \beta(i)$ whenever $\mathbf{x} \in S_i$ [6].

This type of quantizers are known as "vanilla" vector quantizers or block-source codes. In the literature the term *code* is used as a generic substitute for *quantizer*.

The instantaneous rate of a quantizer applied to a particular input is the normalized length $r(\mathbf{x}) = \frac{1}{k} l(\gamma(\alpha(\mathbf{x})))$ of the channel codeword, the number of bits per source symbol that must be sent to describe the reproduction. If all binary codewords have the same length, we talk of a *fixed-length* or *fixed-rate* quantizer.

To measure the quality of the reproduction, we assume the existence of a nonnegative distortion measure $d(\mathbf{x}, \hat{\mathbf{x}})$ which assigns a distortion or cost to the reproduction of input $\mathbf{x}$ by $\hat{\mathbf{x}}$. Ideally, one would like a distortion measure that is easy to compute, useful in analysis, and perceptually meaningful in the sense that small (large) distortion means good (poor) perceived quality. No single distortion measure accomplishes all three goals [6]. However, $d(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$ satisfies the first two.

We also assume that $d(\mathbf{x}, \hat{\mathbf{x}}) = 0$ if and only if $\mathbf{x} = \hat{\mathbf{x}}$. In this light we say that a code is *lossless* if $d(\mathbf{x}, \beta(\alpha(\mathbf{x}))) = 0$ for all inputs $\mathbf{x}$, *lossy* otherwise.

Finally, the overall performance of a quantizer applied to a source is characterized by the normalized rate

$$\begin{aligned} R(\alpha,\gamma) = E[r(X)] &= \frac{1}{k}E[l(\gamma(\alpha(X)))] \\ &= \frac{1}{k}\sum_i l(\gamma(i)) \int_{S_i} f(\mathbf{x})\, d\mathbf{x}, \end{aligned} \tag{4.3}$$

and the normalized average distortion

$$\begin{aligned} D(\alpha,\beta) &= \frac{1}{k}E[d(X,\beta(\alpha(X)))] \\ &= \frac{1}{k}\sum_i \int_{S_i} d(\mathbf{x},\mathbf{y}_i) f(\mathbf{x})\, d\mathbf{x}. \end{aligned} \tag{4.4}$$

Every quantizer (α,γ,β) is thus described by a rate-distortion pair $(R(\alpha,\gamma), D(\alpha,\beta))$. The goal of compression system design is to optimize the rate-distortion trade-off [6].

In light of the results by Shannon, compression system design will also have to take into account the characteristics of the communication channel in managing the rate-distortion trade-off. Also, from the definitions above, it is clear that knowledge of the probability density function of the source messages is very important.

4.1 Image quantization and coding

Under the perspective from the previous introduction, we return to the topic of image quantization and coding. The image standards JPEG and JPEG 2000 are framed in the transform coding paradigm, and contain two more steps besides their respective discrete cosine transform (DCT) and the Cohen-Daubechies-Feauveau

5/3 (lossless) and 9/7 (lossy) biorthogonal wavelet transforms at the core of their compression engines. Both have a *quantization* and an *coding* step, as described in the literature, with their respective "inverses", to complete their definitions [3, 22, 25]. Note how the use of the words "quantization" and "coding" refer to α and γ from the previous introduction, respectively.

The general schematic for transform coding, under which both JPEG and JPEG 2000 techniques—as well as our approach—fall under, is described in the schematic drawing shown in Figure 4.2.

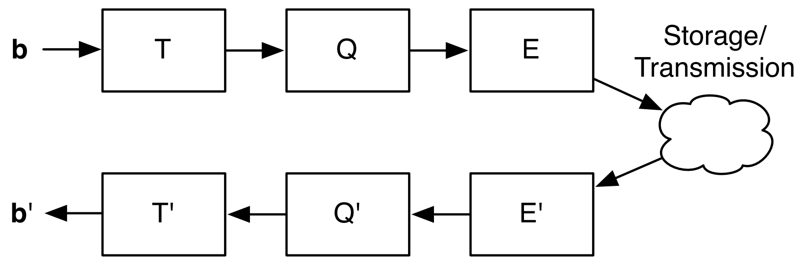


Figure 4.2: Schematic diagram of a general transform coding system.

The transform T is meant to exploit the redundancy in the source signal $\mathbf{b}$ and decorrelate it. It has an inverse T^{-1}, or at the very least a left inverse T' such that $T'T\mathbf{b} = \mathbf{b}$. In our approach we have $\mathbf{A} = T'$, and T is defined via OMP by $T\mathbf{b} = \mathtt{OMP}(\mathbf{A}, \mathbf{b}, \epsilon_0)$. In this case, we have $\|T'T\mathbf{b} - \mathbf{b}\|_2 < \epsilon$. Therefore, our compression scheme is lossy. Q is a non-invertible scalar quantizer that will be applied to the coefficients of vector $T\mathbf{b}$, and Q' is its reproduction function. Finally, we have an invertible lossless encoder E, γ in the introduction, with $E' = E^{-1}$. The function composition QT is equivalent to the lossy encoder α, and the function composition $T'Q'$ corresponds to the reproduction decoder β from the introduction above.

In order to describe the overall performance of our quantizer $(QT, E, T'Q')$, we would need to characterize the rate-distortion pair $(R(QT, E), D(QT, T'Q'))$.

For the scope of the present work, we have studied the error sensitivity for $\mathbf{Ax} = \mathbf{y}$, in terms of $\mathbf{A} \in \mathbb{R}^{n\times m}$ and $\mathbf{x} \in \mathbb{R}^m$. Assume that the columns $\mathbf{a}_j$ of $\mathbf{A} = (\mathbf{a}_j)$ all have the same norm $\|\mathbf{a}_j\|_2 = c$. Let $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$, where $\epsilon = (\epsilon_1, \ldots, \epsilon_m)^{\mathrm{T}}$. What can we say of the distance between $\tilde{\mathbf{y}} = \mathbf{A}\tilde{\mathbf{x}}$ and $\mathbf{y} = \mathbf{Ax}$?

$$\begin{aligned}
\|\tilde{\mathbf{y}} - \mathbf{y}\|_2 &= \|\mathbf{A}\tilde{\mathbf{x}} - \mathbf{Ax}\|_2 = \|\mathbf{A}(\mathbf{x} + \epsilon) - \mathbf{Ax}\|_2 \\
&= \|\mathbf{A}\epsilon\|_2 \\
&= \Big\| \sum_{j=1}^{m} \mathbf{a}_j \epsilon_j \Big\|_2 = \Big\| \sum_{\epsilon_j \neq 0} \mathbf{a}_j \epsilon_j \Big\|_2 \\
&\leq \sum_{\epsilon_j \neq 0} \|\mathbf{a}_j \epsilon_j\|_2 = \sum_{\epsilon_j \neq 0} \|\mathbf{a}_j\|_2 |\epsilon_j| \\
&= \sum_{\epsilon_j \neq 0} c|\epsilon_j| \quad , \|\mathbf{a}_j\|_2 = c \; \forall j \\
&\leq \sum_{\epsilon_j \neq 0} c\|\epsilon\|_\infty \quad , \|\epsilon\|_\infty = \max_j |\epsilon_j| \\
&= c\|\epsilon\|_\infty \|\epsilon\|_0 \quad , \|\epsilon\|_0 = \#\{j : |\epsilon_j| > 0\}. \qquad (4.5)
\end{aligned}$$

Hence, the error $\|\tilde{\mathbf{y}} - \mathbf{y}\|_2$ is bound by $c\|\epsilon\|_\infty \|\epsilon\|_0$. Moreover, the value of $\|\epsilon\|_0$ is linked to the sparsity of $\mathbf{x}$, because in our case the error ϵ comes from scalar quantizing the entries of $\mathbf{x}$. That is, if—for example—$\tilde{\mathbf{x}} = \mathit{round}(\mathbf{x})$, where $\tilde{\mathbf{x}}$ is the vector whose entries are exactly those of $\mathbf{x}$ but rounded to the closest integer, then necessarily

$$\|\epsilon\|_0 = \|\tilde{\mathbf{x}} - \mathbf{x}\|_0 \leq \|\mathbf{x}\|_0 \tag{4.6}$$

Hence, in the case where a scalar quantization scheme satisfies (4.6), Equation 4.5 gives

$$\|\tilde{\mathbf{y}} - \mathbf{y}\|_2 \leq c\|\epsilon\|_\infty \|\mathbf{x}\|_0, \tag{4.7}$$

with $\mathbf{A} = (\mathbf{a}_j)$, $\tilde{\mathbf{y}} = \mathbf{A}\tilde{\mathbf{x}}$, $\mathbf{y} = \mathbf{A}\mathbf{x}$, and $c = \|\mathbf{a}_j\|_2$ for all j. Observe that, in particular, when $\tilde{\mathbf{x}} = round(\mathbf{x})$, we have $\|\epsilon\|_\infty = 1/2$.

From Equation 4.7 we see that the error in the reconstruction due to scalar quantization is linked to the size of c, $\|\epsilon\|_\infty$, and the sparsity of $\mathbf{x}$. We are tempted to make c as small as possible, or modify the quantization scheme to make $\|\epsilon\|_\infty$ smaller to reduce the reconstruction error due to scalar quantization.

The problem is that the magnitude of c is linked to the norm of $\mathbf{x}$, which affects the number of bits we would need to represent $\mathbf{x}$. To see the link between c and the norm of $\mathbf{x}$, consider the following example. We will come back to the issue of the representation of $\mathbf{x}$ shortly.

Assume that $\|\mathbf{a}_j\|_2 = c$ for all j, where $\mathbf{A} = (\mathbf{a}_j)$, and $\mathbf{a}_j$ are the columns of $\mathbf{A}$. Suppose that $\mathbf{A}\mathbf{x} = \mathbf{y}$ and $\mathbf{x} = (x_1, 0, \ldots, 0)^{\mathrm{T}}$, i.e. $\mathbf{y} = x_1\mathbf{a}_1$. Keeping $\mathbf{y}$ fixed, as a given right hand side, what happens to $\|\mathbf{x}\|_2 = |x_1|$ as we modify c? Let $a > 0$ and consider $\mathbf{A}_a = (a\mathbf{a}_j)$. Then $\mathbf{A}_a\mathbf{x} = x_1(a\mathbf{a}_1) = a(x_1\mathbf{a}_1) = a\mathbf{y}$, hence $\mathbf{A}_a(a^{-1}\mathbf{x}) = \mathbf{y}$.

It is easy to see by linearity from this example that the general case, where $\mathbf{A}\mathbf{x} = \mathbf{y}$ and $\mathbf{x} = (x_1, \ldots, x_m)^{\mathrm{T}}$, also implies $\mathbf{A}_a(a^{-1}\mathbf{x}) = \mathbf{y}$. The norm of the columns of $\mathbf{A}_a$ is $\|a\mathbf{a}_j\|_2 = a\|\mathbf{a}_j\|_2 = ac$, $a > 0$. Making ac small implies making the parameter a small, since c is fixed by the choice of $\mathbf{A}$, and therefore making $\|a^{-1}\mathbf{x}\|_2 = a^{-1}\|\mathbf{x}\|_2$ big.

Hence, if a vector $\mathbf{x}_0$ solves $\mathbf{A}\mathbf{x} = \mathbf{y}$, then the norm of $a^{-1}\mathbf{x}_0$, which solves $\mathbf{A}_a\mathbf{x} = \mathbf{y}$, will increase as $a \to 0$. It is easy to see that if $\mathbf{x}_0$ is the sparsest vector that solves $\mathbf{A}\mathbf{x} = \mathbf{y}$, then $a^{-1}\mathbf{x}_0$ is the sparsest vector that solves $\mathbf{A}_a\mathbf{x} = \mathbf{y}$, and $\|\mathbf{x}_0\|_0 = \|a\mathbf{x}_0\|_0$.

From the above discussion we conclude that the norm of $a^{-1}\mathbf{x}_0$ has an inversely proportional relationship with the size of ac. Therefore, if we are to use a finite predetermined number of bits to represent $a^{-1}\mathbf{x}_0$, the solution of $\mathbf{A}_a\mathbf{x} = \mathbf{y}$, we need to choose a carefully.

Generalizing Equation 4.7 to the class of matrices $\mathbf{A}_a$, $a > 0$, the error bound due to the scalar quantization of the entries of $\mathbf{x}$ becomes

$$\|\tilde{\mathbf{y}} - \mathbf{y}\|_2 \leq ac\|\epsilon\|_\infty \|\mathbf{x}\|_0, \tag{4.8}$$

where $\mathbf{A} = \mathbf{A}_1 = (\mathbf{a}_j)$, $\|\mathbf{a}_j\|_2 = c$ for all j, $\tilde{\mathbf{y}} = \mathbf{A}_a\tilde{\mathbf{x}}$, $\mathbf{y} = \mathbf{A}_a\mathbf{x}$, and $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$ with $\|\epsilon\|_0 \leq \|\mathbf{x}\|_0$.

From the results of Section 3.2.3 we can see that the magnitude of the coordinates of $\mathbf{x}$ are bound by a multiple of $\|\mathbf{y}\|_2$. This also has an impact on how many bits would be needed to represent $\mathbf{x}$. Therefore the choice of a has to take into consideration the maximum value that the value of $\|\mathbf{y}\|_2$ will impose on the magnitude of the coordinates of $\mathbf{x}$.

Finally, let us recall that our image compression approach comes from the

OMP solution $\mathbf{x}_0$ to problem $(P_0^{\epsilon_0})$ for a given matrix $\mathbf{A} = (\mathbf{a}_j)$ whose column vectors satisfy $\|\mathbf{a}_j\|_2 = c$, a vector $\mathbf{b}$, and a tolerance $\epsilon_0 > 0$ (such that $\|\mathbf{A}\mathbf{x}_0 - \mathbf{b}\|_2 < \epsilon_0$.) Then, choosing $a > 0$, and following the description of T at the beginning of this section, if we set $\mathbf{x}_0 = T\mathbf{b} = \texttt{OMP}(a\mathbf{A}, \mathbf{b}, \epsilon_0)$ for a given signal $\mathbf{b}$ and a tolerance $\epsilon_0 > 0$, $a\mathbf{A} = T'$, Q a scalar quantizer that satisfies Inequality 4.6, and Q' its corresponding reproduction function, the triangle inequality and Inequality 4.8 give

$$\begin{aligned} d(\beta(\alpha(\mathbf{b})), \mathbf{b}) &= \|T'Q'QT\mathbf{b} - \mathbf{b}\|_2 \\ &= \|T'Q'QT\mathbf{b} - T'T\mathbf{b} + T'T\mathbf{b} - \mathbf{b}\|_2 \\ &= \|a\mathbf{A}\tilde{\mathbf{x}}_0 - a\mathbf{A}\mathbf{x}_0 + a\mathbf{A}\mathbf{x}_0 - \mathbf{b}\|_2 \\ &\leq \|a\mathbf{A}\tilde{\mathbf{x}}_0 - a\mathbf{A}\mathbf{x}_0\|_2 + \|a\mathbf{A}\mathbf{x}_0 - \mathbf{b}\|_2 \\ &= ac\|\delta\|_\infty \|\mathbf{x}_0\|_0 + \epsilon_0, \end{aligned}$$

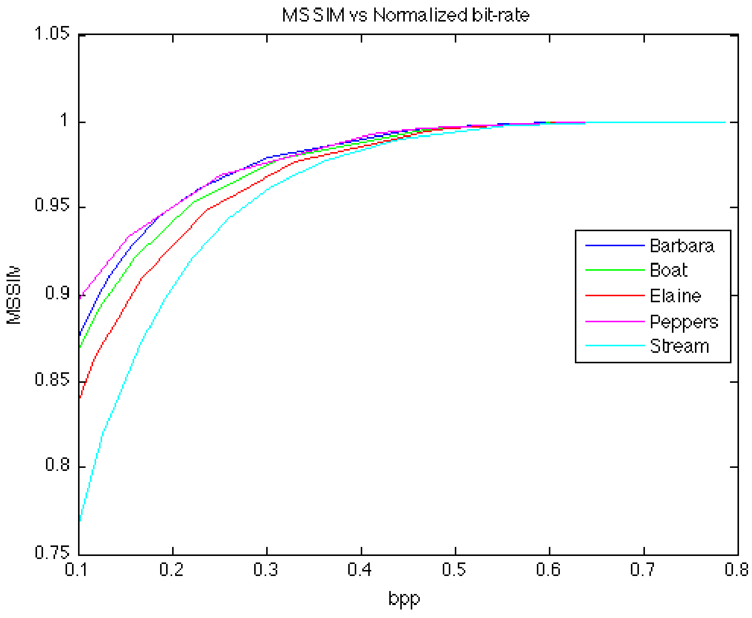
where $\delta = \tilde{\mathbf{x}}_0 - \mathbf{x}_0$. This inequality would give us a footing in the computation of the normalized average distortion $D(\alpha, \beta)$, see Equation 4.4.

From the definition of $D(\alpha, \beta)$, it is clear that we need to know something about the probability density function of the input sources, i.e., the statistics of the 8×8 linearized sub images into which each image is partitioned, if we are to compute D. As a surrogate of such knowledge, we can observe the distribution of the coefficients for each of the vectors resulting from the analysis of the images in our database, and their corresponding histograms. This is what Figures 4.5 through 4.9 show. For each image $\mathcal{I}$ in our database, we used matrix $\mathbf{A} = [\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$ with a tolerance of $\epsilon = 32$ to compute its statistics. On the x-axis of each subfigure we have matched column $\mathbf{a}_i$ at position i to integer i for each of the columns of $\mathbf{A} = (\mathbf{a}_i)$. Hence, positions 1 to 64 correspond to the DCT waveforms, and positions 65 to 128 to the Haar waveforms. A subfigure labeled (a) corresponds to the histogram for the frequency each column vector of $\mathbf{A}$ is chosen. For example, since there are 4096 sub images of size 8×8 in a 512×512 image, column vector $\mathbf{a}_1$ will be chosen 4096 times since it corresponds to the constant vector, which computes the mean or DC component for each sub image. Subfigure (b) corresponds to a partial representation of the distribution of the coefficients that multiply each and every column vector whenever such vector is chosen in the representation/approximation of an input $\mathbf{b}$. For example, suppose that column $\mathbf{a}_{74}$ was multiplied by a coefficient $a_{74} = 3.2310$ to obtain the representation of some input $\mathbf{b} = a_{74}\mathbf{a}_{74} + \mathbf{r}$ within a tolerance of $\epsilon = 32$; then we would have plotted point $(74, 3.2310)$ in subfigure (b). We have not plotted the coefficients for $\mathbf{a}_1$ since they correspond to the DC components of the sub images of $\mathcal{I}$, which vary between 0 and 255. We note that all images in our database have a similar structure.

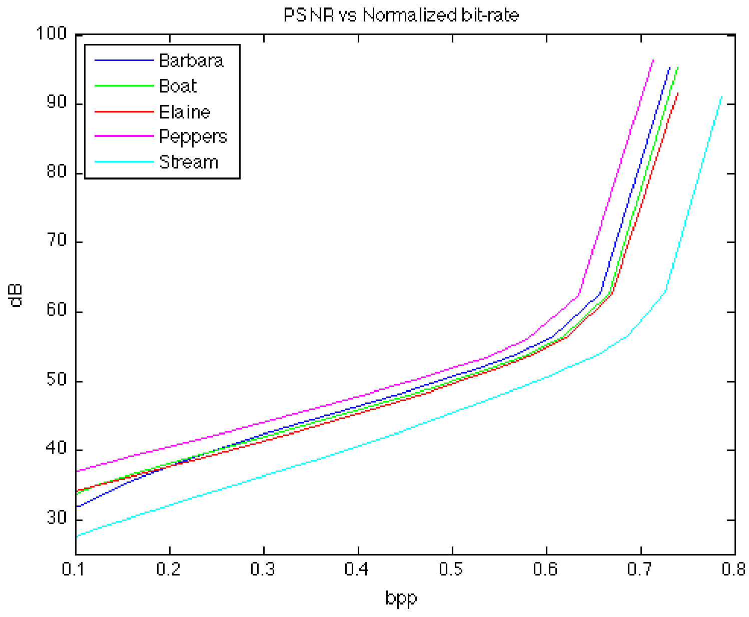
For comparison purposes, we obtained the histogram and the distribution of coefficients for a randomly generated image with a uniform distribution on [0 255], see Figure 4.10. The first thing we note is that unlike for the natural images in our database, all column vectors—except $\mathbf{a}_1$ and $\mathbf{a}_{65}$, which correspond to the constant vectors (one for the DCT and one for the Haar waveforms)—are chosen about the same number of times regardless of their position; and the distribution of the values of the coefficients is uniform as well. It is also clear that this is the image that requires the most nonzero coefficients to be reconstructed within the tolerance $\epsilon = 32$ chosen. This is consistent with the definition of information by Shannon: the more the uncertainty in a source, the more there is information in it.

Regarding the computation of the rate $R(\alpha, \gamma)$ for our image quantizer, we would have to choose a lossless encoder γ, which we have not done here.

On the other hand, we have plotted the distortion as measured by the MSSIM index and the PSNR versus the idealized normalized bit-rate, defined in Section 3.6. This bit-rate is unattainable in practice, as observed in that section, but this gives us nonetheless an idea of an upper bound in the rate-distortion trade-off, and lets us compare how much room we have to select a lossless encoder γ to complete the implementation of a quantizer using our sparse image representation approach. Figure 4.3 shows the MSSIM and PSNR versus normalized bit-rate trade-off. Figure 4.4(a) shows the PSNR versus normalized bit-rate trade-off for image *Lena*, which we computed to compare with Figure 4.4(b) which shows results for that image for three published fully implemented quantizers. We observe that there is enough room to pick an encoder γ that could compete with these implementations.

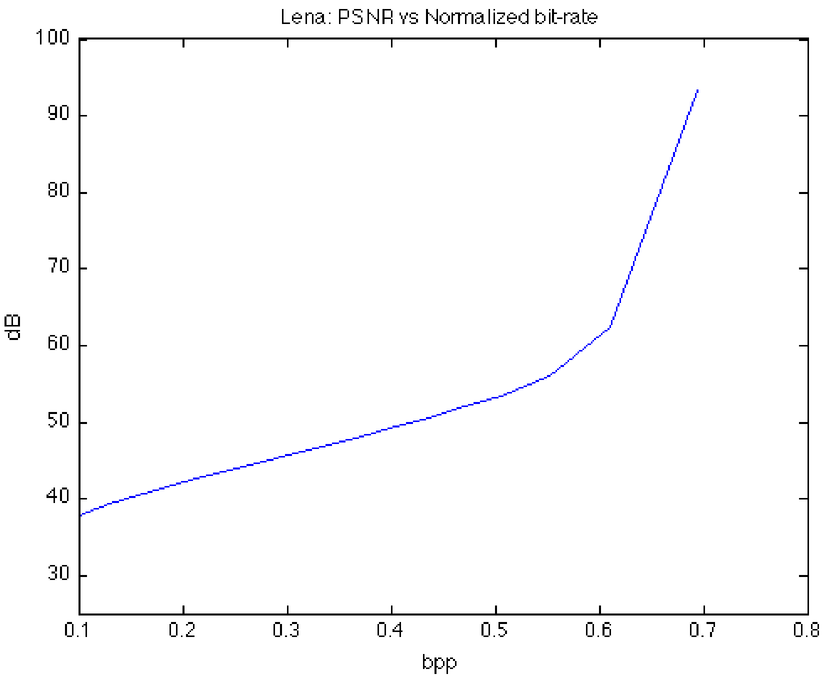


(a)

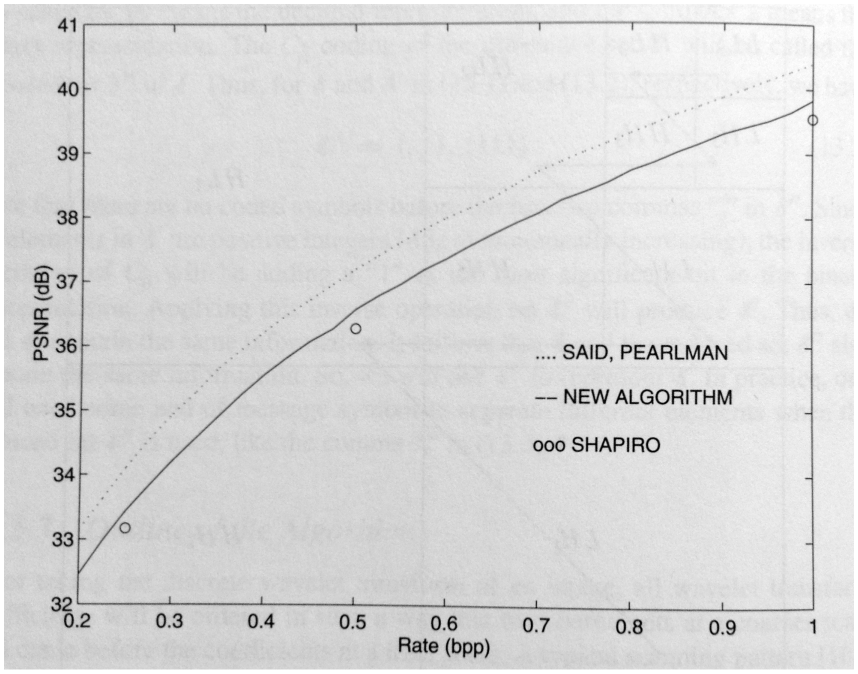


(b)

Figure 4.3: (a) MSSIM vs Normalized bit-rate, (b) PSNR vs Normalized bit-rate.

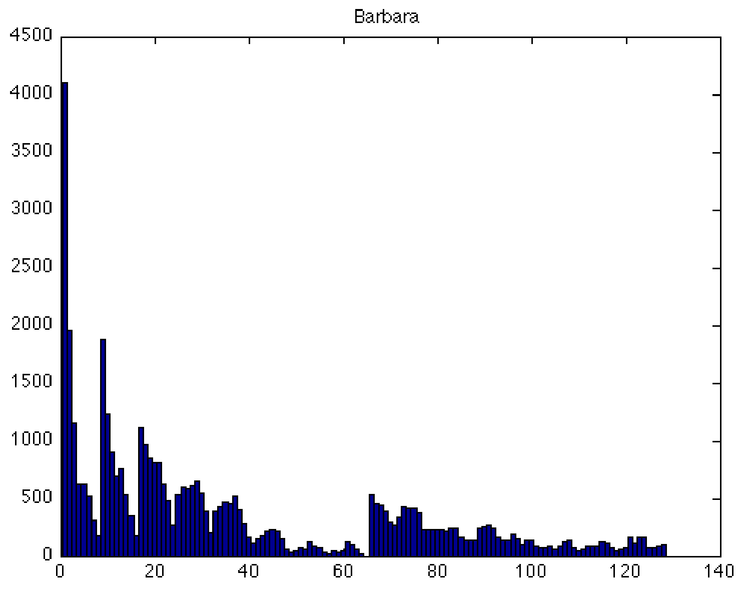


(a)

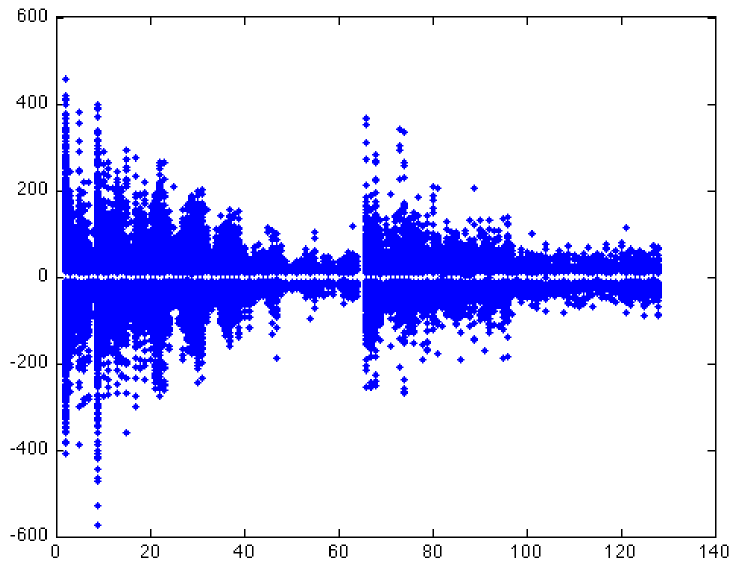


(b)

Figure 4.4: PSNR vs bit-rate: (a) Normalized bit-rate results for $\mathbf{A} = [\mathrm{DCT}_1\ \mathrm{Haar}_1]$ prior to any γ coding, and (b) bit-rate coding performances published in [14] for image *Lena*: Said and Pearlman's SPIHT algorithm [15], Embeded Coding and the Wavelet-Difference-Reduction compression algorithm ("new algorithm") [23], and Shapiro's EZW algorithm [18].

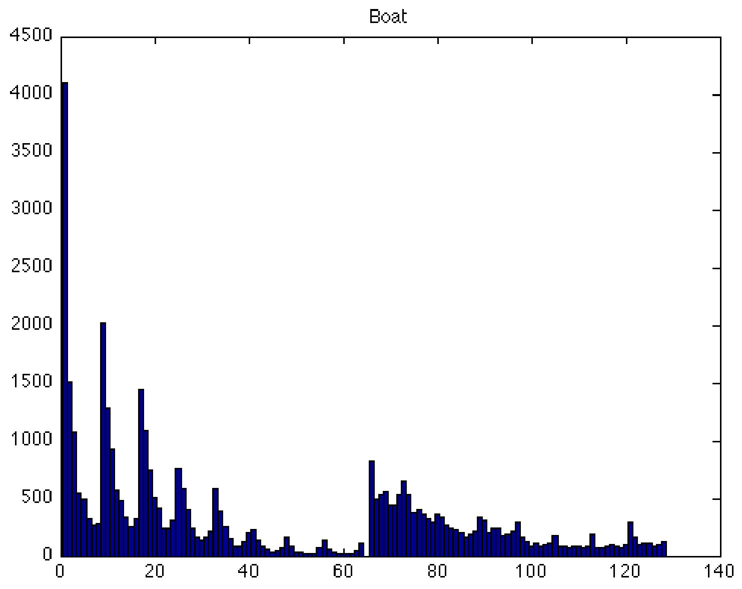


(a)

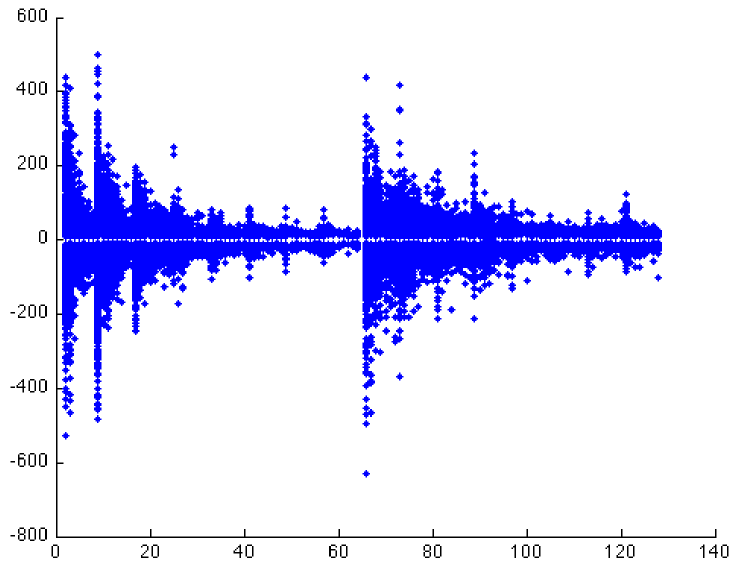


(b)

Figure 4.5: Histograms for image *Barbara*. $\mathbf{A} = [\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$, tolerance $\epsilon = 32$.

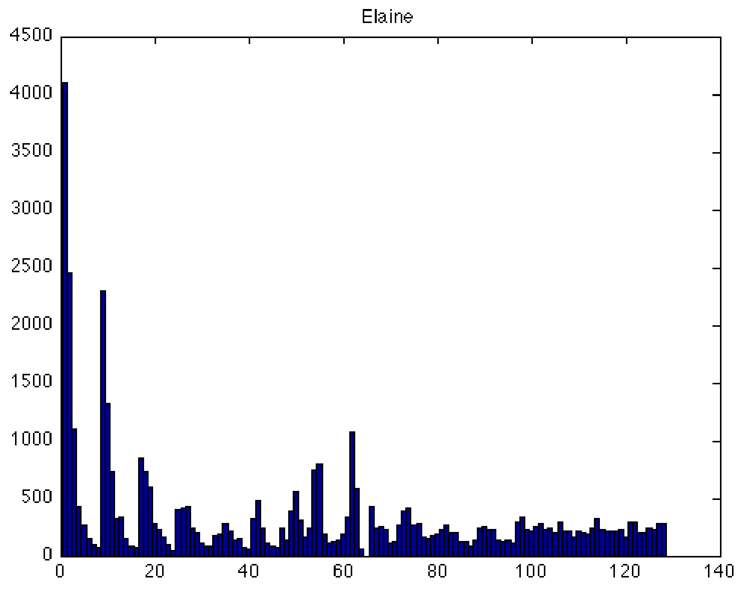


(a)

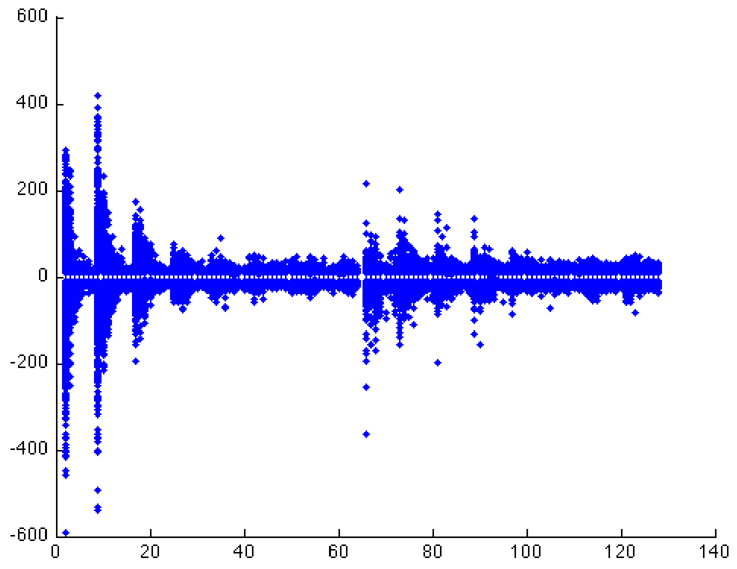


(b)

Figure 4.6: Histograms for image *Boat*. $\mathbf{A} = [\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$, tolerance $\epsilon = 32$

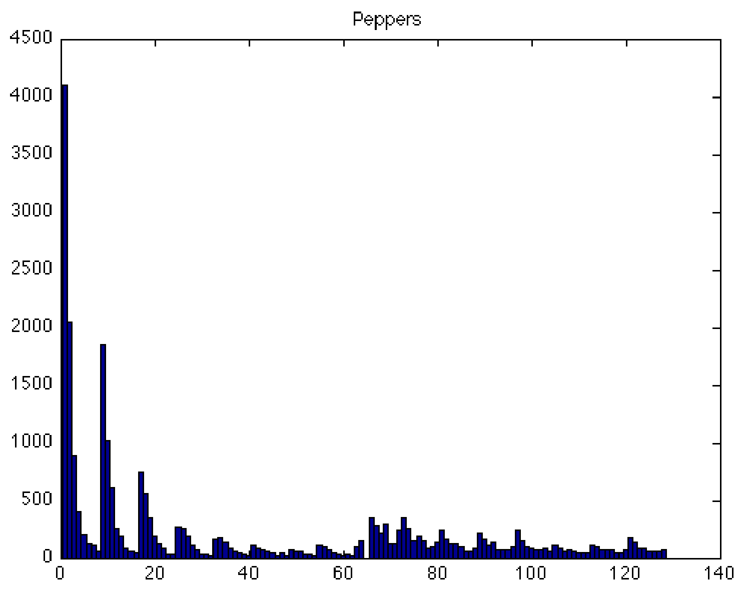


(a)

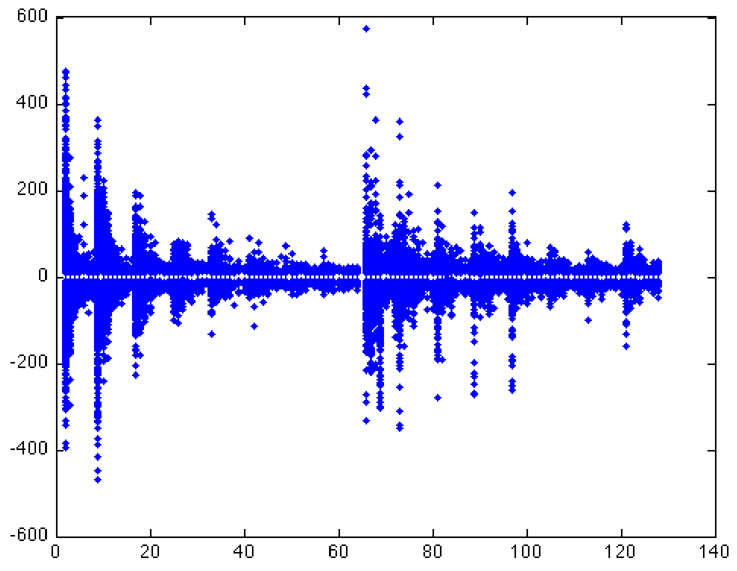


(b)

Figure 4.7: Histograms for image *Elaine*. $\mathbf{A} = [\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$, tolerance $\epsilon = 32$

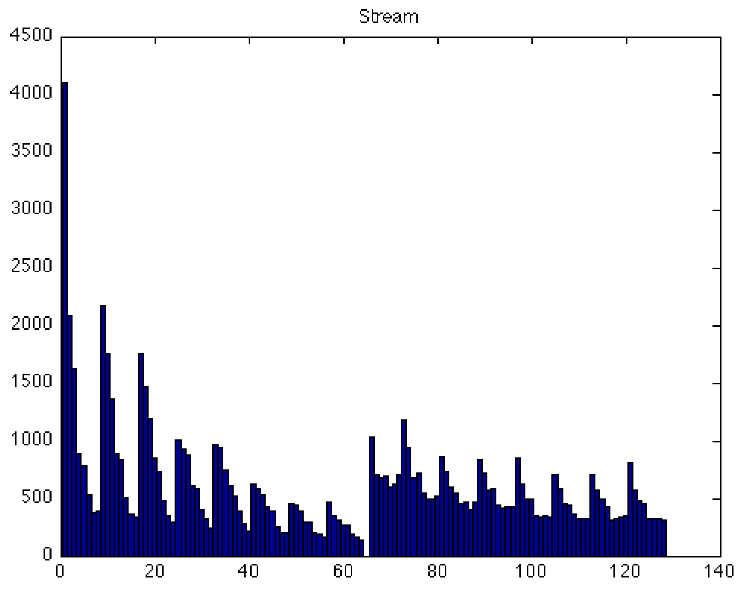


(a)

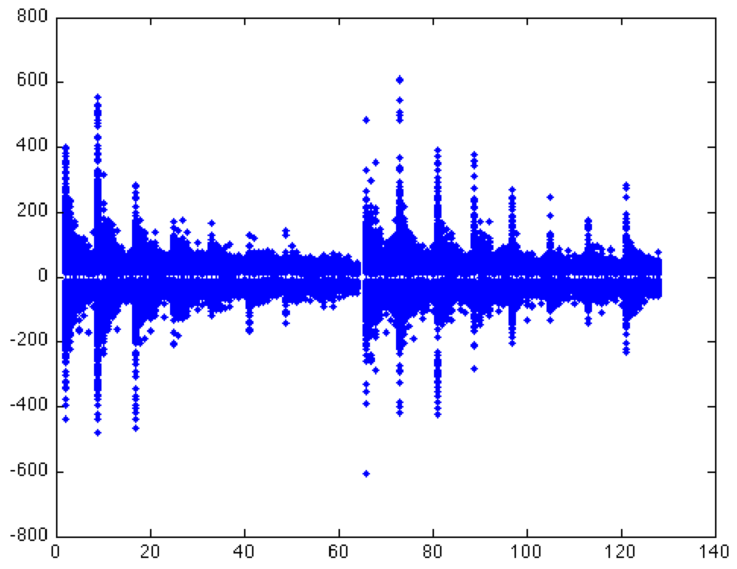


(b)

Figure 4.8: Histograms for image *Peppers*. $\mathbf{A} = [\text{DCT}_{2,3}\ \text{Haar}_{2,3}]$, tolerance $\epsilon = 32$

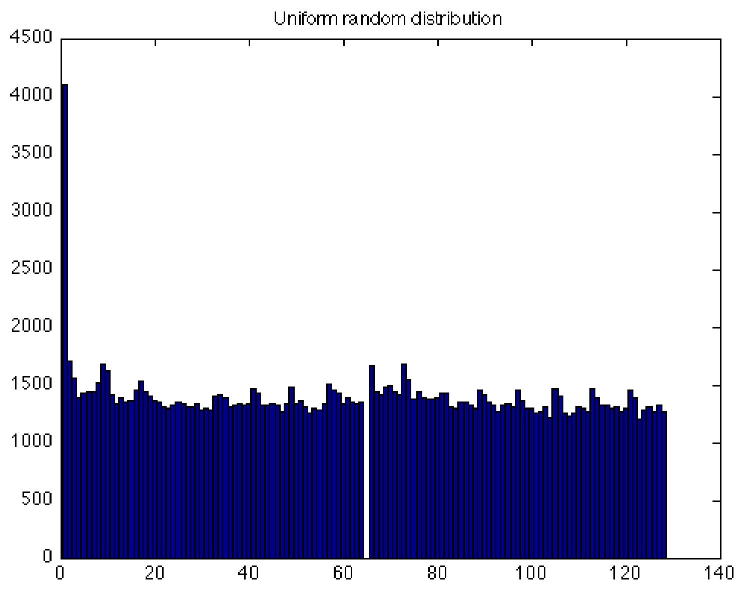


(a)

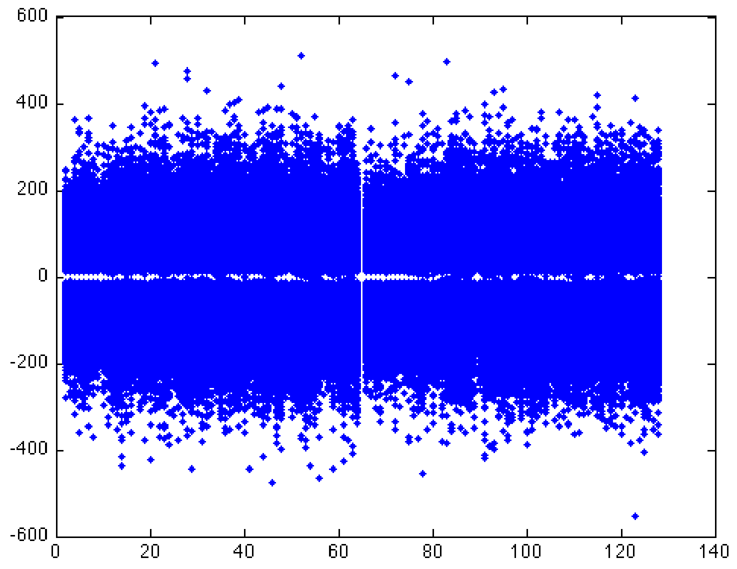


(b)

Figure 4.9: Histograms for image *Stream*. $\mathbf{A} = [\text{DCT}_{2,3}\ \text{Haar}_{2,3}]$, tolerance $\epsilon = 32$



(a)



(b)

Figure 4.10: Histograms for uniform random input. $\mathbf{A} = [\mathrm{DCT}_{2,3}\ \mathrm{Haar}_{2,3}]$, tolerance $\epsilon = 32$

Chapter 5

New directions

From the discussion in Chapter 4, it is clear that to complete an implementation of a quantizer based in the sparse image representation we would have to choose a lossless encoder γ that would allow us to match or beat the results shown in Figure 4.4(b). Such an encoder would have to be able to take advantage of the sparsity of the solution $\mathbf{x}_0$ to problem

$$(P_0^\epsilon): \qquad \min_{\mathbf{x}} \|\mathbf{x}\|_0 \quad \text{subject to} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 < \epsilon,$$

for a given right hand side $\mathbf{b}$ and tolerance $\epsilon > 0$. A combination of techniques such as Huffman encoding and run-legth encoding come to mind, see for example [10] for details. Said and Pearlman's SPIHT algorithm [15], and Shapiro's EZW algorithm [18] would be sources of inspiration as well.

As noted in Chapter 4, knowledge of natural image statistics is essential to the computation of rate R and distortion D quantities. This knowledge can also be used to optimize the design of the scalar quantizer in the setting of transform encoding by allocating more cells where most of the coordinates x_i of solution vector $\mathbf{x}_0 = (x_1, \ldots, x_m)^{\mathrm{T}}$ for (P_0^ϵ) would fall, as illustrated by Figures 4.5 through 4.9

Moreover, as pointed out in Section 3.9 at the end of Chapter 3, a study of other matrices $\mathbf{A}$ other than the matrices studied in this work would be desirable. A promising approach would be to study the properties of Gabor systems, for example.

Finally, given that the NP-completeness of the solution to problem (P_0^ϵ) in the general case [12] does not preclude a priori the existence of fast and efficient algorithms for the computation of sparse solutions to a restricted class of matrices $\mathbf{A}$, it would be most desirable to discover such algorithms and identify the class of matrices for which they would exist, if at all.

Appendix A

Basic frame definitions

Definition 7 (Frame). *A frame for a Hilbert space $\mathcal{H}$ is a sequence of vectors $\{x_i\} \subset \mathcal{H}$ for which there exists constants $0 < A \leq B < \infty$ such that, for every $x \in \mathcal{H}$,*

$$A\|x\|^2 \leq \sum_i |\langle x, x_i \rangle|^2 \leq B\|x\|^2. \tag{A.1}$$

The constants A and B are known respectively as lower and upper *frame bounds.* We will often presume without stating it that A is the greatest lower frame bound and B is the least upper frame bound. A frame is called a *tight frame* if the optimal upper and lower frame bounds are equal; $A = B$. A frame is a *Parseval frame* if $A = B = 1$. A *uniform frame* is a frame in which all the vectors have equal norm [7]. Equation A.1 is usually called the *frame condition.*

Bibliography

[1] William L. Briggs and Van Emden Henson. *The DFT, an owner's manual for the discrete Fourier transform.* SIAM, 1995.

[2] Alfred M. Bruckstein, David L. Donoho, and Michael Elad. From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM Review*, 51(1):34–81, 2009.

[3] Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi. The JPEG 2000 still image coding system: an overview. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, November 2000.

[4] R. J. Duffin and A. C. Schaeffer. A class of nonharmonic Fourier series. *Transactions of the American Mathematical Society*, 72(2):341–366, 1952.

[5] James Gleick. *The information: a history, a theory, a flood.* Pantheon Books, 2011.

[6] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, October 1998.

[7] Deguang Han, Keri Kornelson, David Larson, and Eric Weber. *Frames for undergraduates*, volume 40 of *Student Mathematical Library.* American Mathematical Society, 2007.

[8] W. Cary Huffman and Vera Pless, editors. *Handbook of coding theory*, volume 1, chapter 1. Elsevier Science B. V., 1998.

[9] W. Cary Huffman and Vera Pless. *Fundamentals of error-correcting codes.* Cambridge University Press, 2010.

[10] Stéphane G. Mallat. *A wavelet tour of signal processing.* Academic Press, 1998.

[11] Stéphane G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, December 1993.

[12] Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[13] K. R. Rao and P. Yip. *Discrete cosine transform: algorithms, advantages, applications.* Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[14] Howard L. Resnikoff and Raymond O. Wells, Jr. *Wavelet analysis. The scalable structure of information.* Springer-Verlag, 1998. Corrected 2nd printing.

[15] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, June 1996.

[16] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[17] Claude E. Shannon and Warren Weaver. *The mathematical theory of communication.* The University of Illinois Press: Urbana, 1949.

[18] Jerome M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, December 1993.

[19] SparseLab. Seeking sparse solutions to linear systems of equations: Software tools. http://sparselab.stanford.edu/.

[20] G. W. Stewart. *Introduction to matrix computations.* Academic Press, 1973.

[21] T. Strohmer and R. W. Heath. Grassmanian frames with applications to coding and communication. *Appl. Comput. Harmon. Anal.*, 14(3):257–275, 2003.

[22] D. S. Taubman and Michael W. Marcellin. *JPEG 2000: image compression fundamentals, standards and practice.* Kluwer Academic Publishers, 2001.

[23] J. Tian and R. O. Wells, Jr. A lossy image codec based on index coding. In *IEEE Data Compression Conference, DCC '96*, page 456, 1996.

[24] Viterbi School of Engineering, University of Southern California. The USC-SIPI image database. http://sipi.usc.edu/database/.

[25] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

[26] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error measurement to structural similarity. *IEEE Transactions on Image Processing*, 13(1):1–14, 2004.

[27] Andrew B. Watson, editor. *Digital images and human vision.* The MIT Press, 1993.

[28] Warren Weaver. The mathematics of communication. *Scientific American*, 181(1):11–15, July 1949.

[29] Mladen V. Wickerhauser. *Adapted wavelet analysis from theory to software.* A K Peters, Ltd., 1996.

[30] Wikipedia. Peak signal-to-noise ratio. http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.