Learning Sparse Orthogonal Wavelet Filters

by

Daniel Recoskie

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	David Fleet Professor Department of Computer Science University of Toronto Department of Computer and Mathematical Science University of Toronto Scarborough
Supervisor:	Richard Mann Associate Professor David R. Cheriton School of Computer Science University of Waterloo
Internal Member:	Robin Cohen Professor David R. Cheriton School of Computer Science University of Waterloo
Internal Member:	Justin Wan Professor David R. Cheriton School of Computer Science University of Waterloo
Internal-External Member:	Edward R. Vrscay Professor Department of Applied Mathematics University of Waterloo

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The wavelet transform is a well studied and understood analysis technique used in signal processing. In wavelet analysis, signals are represented by a sum of self-similar wavelet and scaling functions. Typically, the wavelet transform makes use of a fixed set of wavelet functions that are analytically derived. We propose a method for learning wavelet functions directly from data. We impose an orthogonality constraint on the functions so that the learned wavelets can be used to perform both analysis and synthesis. We accomplish this by using gradient descent and leveraging existing automatic differentiation frameworks. Our learned wavelets are able to capture the structure of the data by exploiting sparsity. We show that the learned wavelets have similar structure to traditional wavelets.

Machine learning has proven to be a powerful tool in signal processing and computer vision. Recently, neural networks have become a popular and successful method used to solve a variety of tasks. However, much of the success is not well understood, and the neural network models are often treated as black boxes. This thesis provides insight into the structure of neural networks. In particular, we consider the connection between convolutional neural networks and multiresolution analysis. We show that the wavelet transform shares similarities to current convolutional neural network architectures. We hope that viewing neural networks through the lens of multiresolution analysis may provide some useful insights.

We begin the thesis by motivating our method for one-dimensional signals. We then show that we can easily extend the framework to multidimensional signals. Our learning method is evaluated on a variety of supervised and unsupervised tasks, such as image compression and audio classification. The tasks are chosen to compare the usefulness of the learned wavelets to traditional wavelets, as well as provide a comparison to existing neural network architectures. The wavelet transform used in this thesis has some drawbacks and limitations, caused in part by the fact that we make use of separable real filters. We address these shortcomings by exploring an extension of the wavelet transform known as the dual-tree complex wavelet transform. Our wavelet learning model is extended into the dual-tree domain with few modifications, overcoming the limitations of our standard model. With this new model we are able to show that localized, oriented filters arise from natural images.

Acknowledgements

I would like to begin by thanking my supervisor, Professor Richard Mann, for his guidance and encouragement over these past years. Our many discussions were instrumental to the writing of this thesis. I would also like to thank my committee: Professors Robin Cohen, Justin Wan, Edward Vrscay, and David Fleet, for taking the time to read and critique this thesis.

I also thank my parents for always encouraging me to pursue my passions, regardless of any practical applications. I would not be the person I am today without them. Finally, I want to thank my partner, Kayli, whose support has kept me sane throughout the years.

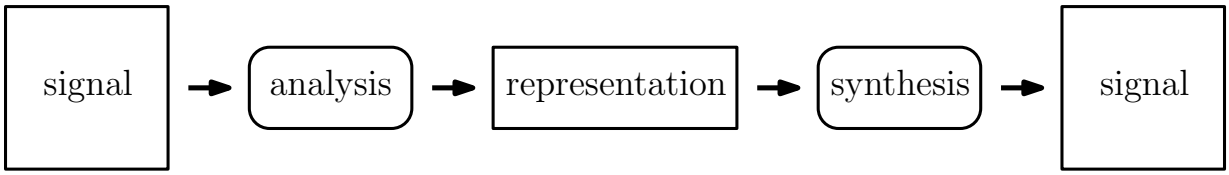
Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction



Broadly speaking, this thesis is about learning representations of data. That one sentence both summarizes this thesis, and yet is too vague to tell you much about it. And so, because specificity is the soul of narrative[1], we begin our story with a few definitions:

- Learning: In this context we mean machine learning. We define machine learning as the ability of an algorithm to adapt to, and find patterns in, data [Russell and Norvig, 1995].
- Data: What exactly do we mean by data? Data is used in different contexts to refer to many different things. However, it is important to be precise, and so this thesis will consider data in the form of signals (i.e. functions from $\mathbb{R}^d \to \mathbb{R}$). The reader may wonder why we restrict ourselves to signals. We argue that it is not much of a restriction as signals are a broad class. For example, audio, images, and video can all be considered types of signals.

[1] These wise words are attributed to John Hodgman, more commonly known as the "PC guy" from a series of successful commercials from a certain computer company.

(a) (b) (c)

Figure 1.1: Three bases found using different methods of matrix factorization applied to a dataset of face images [Lee et al., 2005]. (a) PCA (top principal components shown), (b) NMF, and (c) VQ. These experiments are inspired by those in [Lowe, 1999].

- Representation: Finally we must explain what we mean by representation. In general, algorithms require discrete data as input. Thus, we define a representation to be a discrete, real vector in $\mathbb{R}^n$ (though we will sometimes consider complex vectors). We are interested in representations for signal analysis and synthesis. In other words, we wish to find mappings from signal space to representation space, and vice versa.

Finding good data representations is an important area of study. The problem has been approached for many years, from a variety of different fields. We now give a brief overview of several popular methods in order to help position the thesis. One common approach is to represent data as a sum of components. Under such a model, discrete signals can be represented as a product of matrices. This representation is commonly called matrix factorization and includes methods such as: principal component analysis (PCA) [Pearson, 1901, Hotelling, 1933], nonnegative matrix factorization (NMF) [Lee and Seung, 1999], and vector quantization [Gersho and Gray, 2012]. Other methods include sparse dictionary learning and sparse coding [Olshausen and Field, 1997].

Many data representation techniques have been developed in the context of signal processing. Perhaps the most famous is the Fourier transform [Fourier, 1822]. Others include linear time-frequency transforms such as the Gabor transform, the more general windowed Fourier transform [Gabor, 1946], and the wavelet transform [Haar, 1910].

In the machine learning community, mappings from data to representations are often called feature transforms. Generally, feature transforms can be categorized into either

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

Figure 1.2: (a) SIFT, (b) SURF, and (c) HOG feature representations overlaid on the original image. (d) Wavelet representation of the same image.

fixed (or handcrafted) and learned. Handcrafted feature transforms have been long used by the machine learning community for tasks such as classification. It is common in audio to use a feature transform based on the windowed Fourier transform such as the popular mel-frequency cepstrum coefficients (MFCCs) [Mermelstein, 1976, Davis and Mermelstein, 1990]. Image feature transforms often make use of local descriptors such as the scale-invariant feature transform (SIFT) [Lowe, 1999], speeded up robust features (SURF) [Bay et al., 2008], and the histogram of gradients (HOG) (popularized by [Dalal and Triggs, 2005]).

Figures 1.1–1.3 illustrate different feature representations applied to signals. The details of the methods are not important at the moment, but the reader should note the varied representations that machine learning practitioners can choose from. In fact, it is often unclear what the best data representation is. This can be true even for experts. For example, consider the paper that popularized NMF [Lee and Seung, 1999]. The authors demonstrate that NMF is able to find a parts based representation of faces, with basis components corresponding to eyes, ears, mouths, etc.. However, our results shown in Figure 1.1b are not as compelling, as some of the basis components contain large portions of faces. Our results are not surprising when we consider that the NMF model is based on the idea that a signal is composed as a sum of nonnegative components. Natural images are usually not generated by such a simple additive process. Audio signals, on the other hand, are well suited for NMF because of their additive structure. This was demonstrated in the case of music signals for the task of music transcription [Smaragdis and Brown, 2003]. Figure 1.3c shows the representation matrix found by NMF for a short music sample. Note that the rows correspond very closely to the original notes. We mention this example to illustrate both the difficulty and the importance of choosing a representation.

One way to address the problem of choosing a good data representation is to incorporate feature learning into learning algorithms. More and more, handcrafted features are being replaced with learned features. This change has been driven largely by the success of neural networks. Unlike traditional machine learning approaches that work with data in a feature space, neural networks often process data in its original space. Some popular architectures include: autoencoders for representation learning [Hinton and Salakhutdinov, 2006], convolutional neural networks (CNNs or ConvNets) for image processing [LeCun et al., 1990], and long short-term memory networks (LSTMs) for time series [Hochreiter and Schmidhuber, 1997]. We note here that when neural networks are applied to the audio domain, it is typical to apply a fixed feature transform first. However, there has been recent success in processing raw audio data [Palaz et al., 2013, Tüske et al., 2014, Hoshen et al., 2015, Sainath et al., 2015, Oord et al., 2016].

Connections have been made between traditional representations and more modern neural network models. For example, there is a correspondence between PCA and a particular class of autoencoders [Bourlard and Kamp, 1988, Bengio, 2009]. Oriented bandpass filters (and wavelet filters in particular) have been used as a way of understanding CNNs [Bruna and Mallat, 2011, Mallat, 2012, Bruna and Mallat, 2013, Mallat, 2016, Hadji and Wildes, 2017]. More details about the connections will be discussed in later chapters.

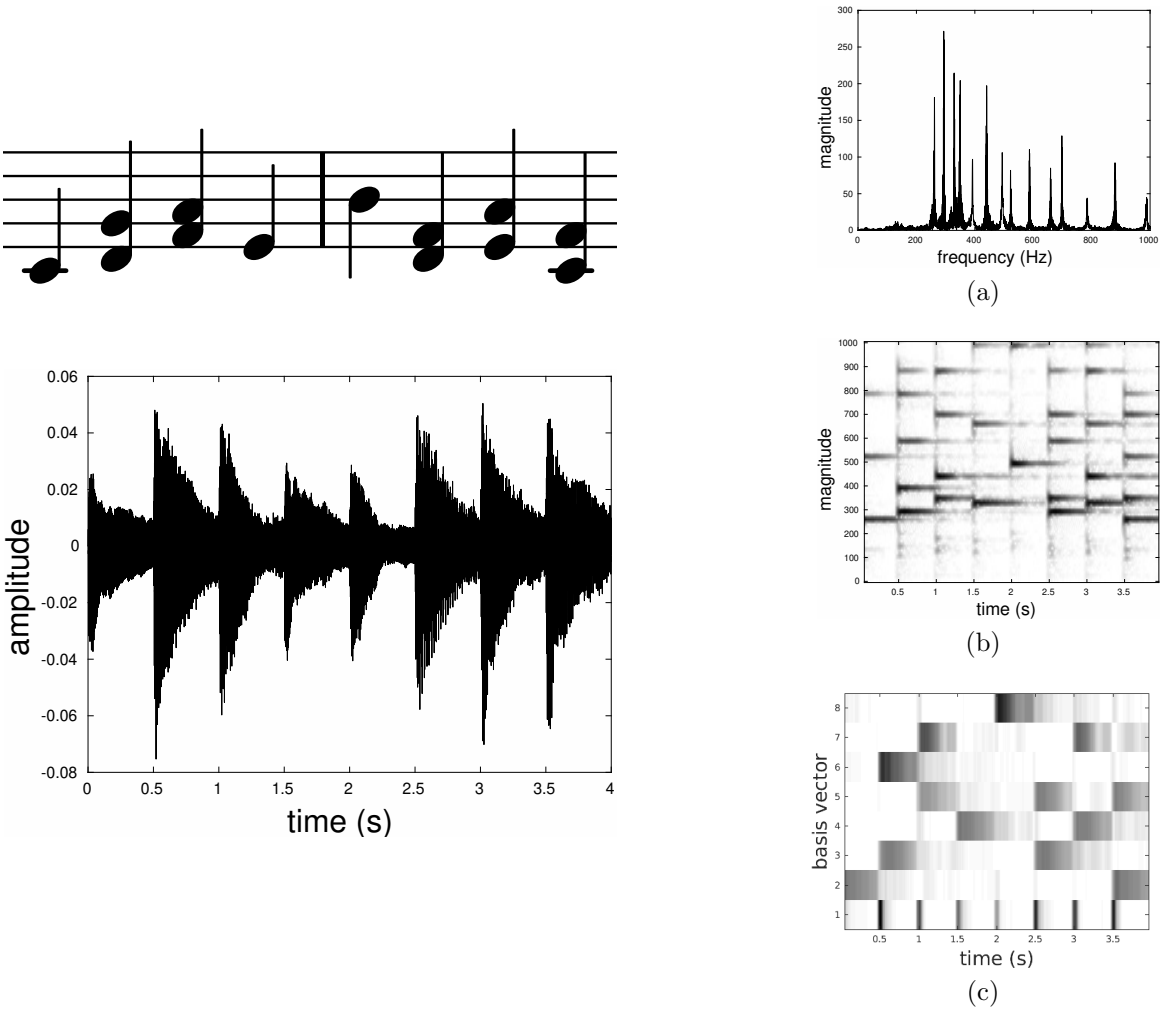


Figure 1.3: Left: Music signal with corresponding transcription. Right: Three different representations of the music signal. (a) Magnitude of the Fourier transform, (b) log magnitude of the windowed Fourier transform, (c) NMF representation of (b). Figure adapted from [Recoskie, 2014].

1.1 Thesis Outline

The first sentence of this thesis stated that our goal is to learn data representations. To make our work tractable, we will restrict ourselves to a particular class of linear time-frequency representations: wavelets. Wavelets are a class of functions that are constrained to be localized in time (or space) and frequency. The wavelet model represents signals as a sum of these localized functions. More details about wavelets can be found in later chapters. The main contribution of this thesis is the introduction of a novel method for learning wavelet filters.

At a high level this thesis aims to answer the following questions:

1. Can useful wavelets be learned?
2. Do traditional wavelets arise from natural data?

By the end of this thesis, we hope to convince the reader that each question can be answered in the affirmative. We do not claim to be the first to ask such questions, but we do claim to introduce a novel framework in which to address them.

Before we can proceed to answer any questions, we will require some mathematical background. We begin with an introduction to signal processing and wavelets in Chapter 2. The chapter starts with a discussion of sampling and the Fourier transform. From these concepts we build towards linear time-frequency transforms, and finally wavelets. The chapter ends with a formal definition of multiresolution analysis, and a short discussion on the precise type of wavelets we are interested in.

Chapter 3 formally states the thesis objectives and outlines our contributions. We also discuss other works that overlap with our own. The main related topics are sparse dictionary learning, finding optimal wavelets using parameterizations, and neural network architectures that have structure based on the wavelet transform.

Our main contribution, the wavelet autoencoder framework, is introduced in Chapter 4. Various experiments are run to demonstrate how the model behaves to different stimuli. The chapter concludes with a probabilistic treatment of the proposed model. In Chapter 5 we extend our wavelet model to two dimensions. We demonstrate that the model behaves similarly to the one-dimensional case.

The particular formulation of the wavelet transform considered in this thesis is not without some limitations. In Chapter 6 we address some of these problems by extending our model to make use of Kingsbury's dual-tree wavelet transform [Kingsbury, 1998]. Our

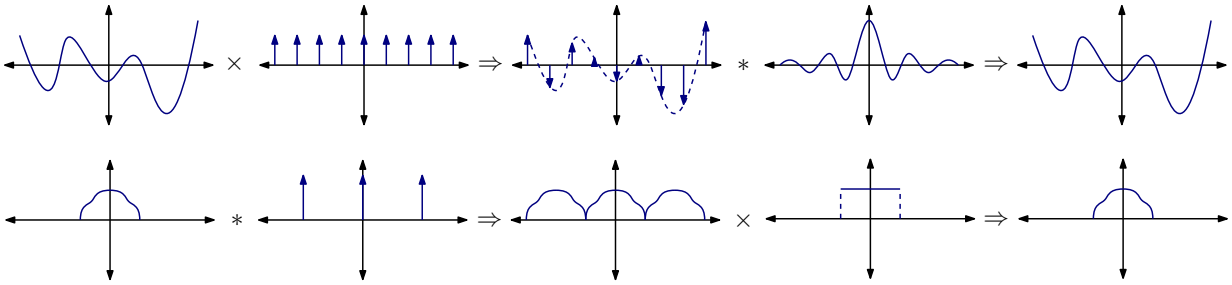
extension to the dual-tree wavelet transform yields two interesting results. The first is that filters with oriented, Gabor-like impulse responses arise from natural images (reminiscent of the work of [Olshausen and Field, 1996]). The second is that our model learns wavelet filters are structurally similar to Kingsbury's q-shift filters [Kingsbury, 2000, Kingsbury, 2003].

In Chapter 7 we explore applications of our learning model, as well as discuss some avenues of future work. We begin by developing a simple classification algorithm, and show some initial results on a dataset of bird songs. We then discuss how we can adapt the classification algorithm for clustering unlabeled signals. Next, we consider two computer vision tasks: compression and object recognition. In the compression task, we show that our learned wavelets are able to better compress images than the wavelet used in the JPEG2000 standard. In the object recognition task, we compare the performance of our learned wavelet representation to that of a standard CNN architecture. We finish the chapter by discussing two ways in which we might extend our model: (1) relaxing the orthogonality constraint in order to learn biorthogonal wavelets, and (2) incorporating variational methods to allow for better signal generation.

Chapters 4, 5, and 6 contain some overlap with our work in [Recoskie and Mann, 2018c], [Recoskie and Mann, 2018b], and [Recoskie and Mann, 2018a] respectively. Any overlapping section was written solely by the author of this thesis. Finally, this thesis contains many figures which are best viewed in colour. However, each figure should be readable in greyscale.

Chapter 2

Signal Processing



This chapter will give an introduction to signal processing in order to provide the reader with enough background to understand the area of multiresolution analysis. The reader may wonder why we are choosing to focus on signals specifically. Signals are a natural type of data to consider since most of the information humans encounter is in the form of signals. Most notably, audio and visual information are forms of signals. Humans demonstrate a natural ability to process and understand signals (e.g. recognizing speech, identifying objects in a scene, etc.).

In general terms, a continuous signal is a function $f : \mathbb{R}^d \to \mathbb{R}$, where d is the dimension of the signal. Typically we consider signals with dimension $d \in \{1, 2\}$. One-dimensional signals include audio signals, where the domain represents time. Images are examples of two-dimensional signals, where the domain is space. For simplicity, the following sections will consider only one-dimensional signals.

2.1 Sampling

In order to do computation on the continuous signals defined above, they must first be discretized. In other words, the signal must be transformed from a continuous function $f : \mathbb{R} \to \mathbb{R}$ to a discrete function $f' : \mathbb{Z} \to \mathbb{R}$. This discretization process is called sampling. Typically, sampling is done by setting $f'(n) = f(ns)$ where $n \in \mathbb{Z}$ and s is the distance (or time in the case of audio) between samples. For convenience, let us define $f[n] \equiv f(ns)$.

An important consideration in sampling is choosing a value for s. We will usually speak in terms of $1/s$, also known as the sampling rate. The choice is often determined by the limitations of the sensor we use to record the signal, and also the degree of information we wish to preserve. We will ignore the first issue and focus only on the preservation of information. Indeed, it should be no surprise that we sacrifice some information in order to discretize a continuous signal since we are moving away from an infinitely precise continuous space. The question still remains: How do we choose a proper sampling rate? For example, compact disc audio has a sampling rate of 44100 samples per second. To understand why this sampling rate was chosen, we will first need some background in Fourier analysis.

2.2 Fourier Analysis

Fourier analysis is one of the most important tools in signal processing. It received its name from Joseph Fourier, who showed that we can represent signals as infinite harmonic sums [Fourier, 1822]. In more precise terms, we can write:

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos\left(\frac{\pi n t}{L}\right) + b_n \sin\left(\frac{\pi n t}{L}\right) \tag{2.1}$$

for a function, f, that is periodic on the interval $[-L, L]$ where

$$\begin{aligned} a_0 &= \frac{1}{2L}\int_{-L}^{L} f(t)\,dt \\ a_n &= \frac{1}{L}\int_{-L}^{L} f(t)\cos\left(\frac{\pi n t}{L}\right)\,dt \\ b_n &= \frac{1}{L}\int_{-L}^{L} f(t)\sin\left(\frac{\pi n t}{L}\right)\,dt \end{aligned} \tag{2.2}$$

Equation 2.1 can be rewritten using Euler's formula

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{\pi i n t/L} \tag{2.3}$$

for appropriate complex coefficients c_n. In the limit as $L \to \infty$ Equation 2.3 becomes

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(\omega) e^{i\omega t}\, d\omega \tag{2.4}$$

where $\hat{f}(\omega)$ is the complex Fourier coefficient for angular frequency ω (in radians per second). We can compute the Fourier coefficients using:

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(t) e^{-i\omega t}\, dt \tag{2.5}$$

Equation 2.5 is known as the Fourier transform and Equation 2.4 as the inverse Fourier transform. Note that continuous signals are used in the above definitions, even though the signals we work with will be discrete. We will sometimes switch between continuous and discrete signals in our derivations, using the form that is most convenient. The discrete signals considered will have finite duration, unlike the infinite duration used in the definition of continuous signals.

We can view the Fourier transform as a change of basis. In the discrete case, our basis is

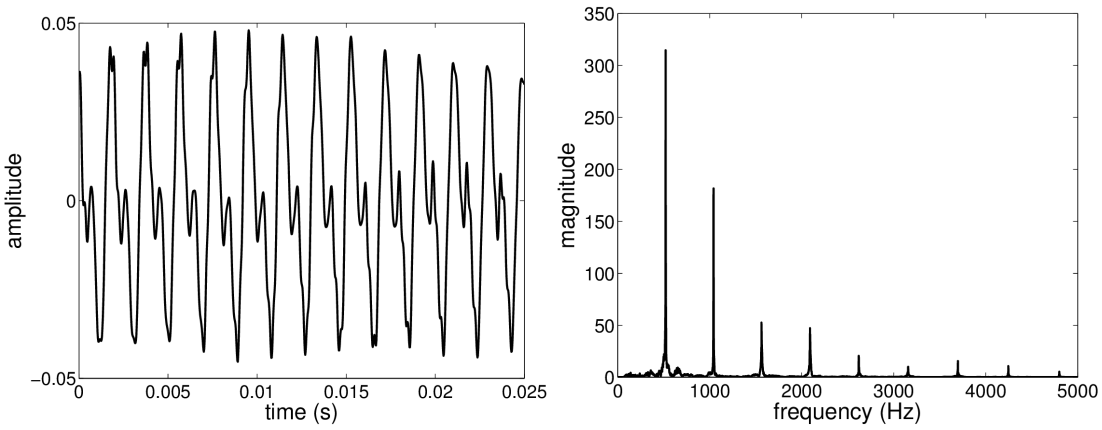
$$\left\{ e_k[n] = e^{i2\pi kn/N} \right\}_{0 \le k < N} \tag{2.6}$$

Since the basis is orthogonal, we have

$$f = \sum_{k=0}^{N-1} \frac{\langle f, e_k \rangle}{||e_k||_2^2} e_k \tag{2.7}$$

where angle brackets indicate the inner product and $||\cdot||_2$ is the ℓ_2-norm. By noting that $||e_k||_2^2 = N$ and letting $\hat{f}[k] = \langle f, e_k \rangle$, we arrive at the discrete version of Equations 2.4 and 2.5:

$$f[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{f}[k] e^{i2\pi kn/N} \tag{2.8}$$



(a) (b)

Figure 2.1: (a) Portion of a time-pressure wave of a C note (∼520 Hz) played on a piano. (b) Magnitude of the Fourier transform of the audio signal in (a). The recording was taken from [Emiya et al., 2010].

$$\hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{-i2\pi kn/N} \tag{2.9}$$

where k is the number of oscillations in a length N window. Note that we assume our signals start at $n = 0$.

Understanding the Fourier transform will be important for our later discussion of multiresolution analysis. Intuitively, the Fourier transform is a decomposition of a signal into the sum of its harmonic components. Consider the signal of a musical note (∼520 Hz) in Figure 2.1a. We can see that there is some periodic structure in the signal. The Fourier transform will make this structure very clear. Figure 2.1b shows the magnitude of the Fourier transform as it is applied to the musical note. For simplicity, we will generally concern ourselves only with the magnitudes of the complex valued Fourier coefficients (i.e. we ignore phase). Note the well defined peaks in the Fourier spectrum. Each of these peaks corresponds to a harmonic component of the note. The first (lowest frequency) peak is called the fundamental frequency. Each of the following peaks occurs at integer multiples of the fundamental frequency. The presence and relative sizes of the harmonics are what differentiate the timbre of musical instruments.

2.2.1 Sampling Theorem

With our knowledge of the Fourier transform we can now answer the question of sampling rate, and more specifically, what information we lose by sampling a continuous signal. The answer may surprise the reader: under certain conditions, we lose no information when sampling. In other words, given a discretized signal we can perfectly reconstruct the original continuous signal. This is demonstrated formally with the following theorem attributed to Shannon [Shannon, 1948], Whittaker [Whittaker, 1915], and others [Luke, 1999]:

Theorem 1. *(Sampling Theorem). If f contains no frequencies higher than half the sampling rate, then*

$$f(t) = \sum_{n=-\infty}^{+\infty} f(ns)\phi_s(t - ns) \tag{2.10}$$

where

$$\phi_s(t) = \frac{\sin(\pi t/s)}{\pi t/s} \tag{2.11}$$

In other words, we can perfectly reconstruct a continuous signal from its sampled version so long as it does not contain too high of frequencies. A formal proof of the sampling theorem is not included here, but we will outline the high-level steps of the proof. Before we begin, we require the following theorems relating convolution,

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)\, d\tau \tag{2.12}$$

and element-wise multiplication.

Theorem 2. *(Convolution Theorem). Let f and g be two continuous signals with $h = f*g$. Then,*

$$\hat{h}(\omega) = \hat{f}(\omega)\hat{g}(\omega). \tag{2.13}$$

Proof.

$$\hat{h}(\omega) = \int_{-\infty}^{+\infty} \exp^{-i\omega t} \left(\int_{-\infty}^{+\infty} f(\tau)g(t - \tau)\, d\tau \right) dt$$

By Fubini's theorem we have

$$\hat{h}(\omega) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \exp^{-i\omega t} f(\tau)g(t - \tau)\, d\tau\, dt$$

Substituting $u = t - \tau$ yields

$$\begin{aligned}\hat{h}(\omega) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \exp^{-i\omega(\tau+u)} f(\tau)g(u)\, d\tau\, du \\ &= \left(\int_{-\infty}^{+\infty} \exp^{-i\omega\tau} f(\tau)\, d\tau\right)\left(\int_{-\infty}^{+\infty} \exp^{-i\omega u} g(u)\, du\right) \\ &= \hat{f}(\omega)\hat{g}(\omega)\end{aligned}$$

□

An alternate form of the convolution theorem concerns the opposite direction.

Theorem 3. *(Convolution Theorem). Let f and g be two continuous signals with $h(t) = f(t)g(t)$. Then,*

$$\hat{h} = \hat{f} * \hat{g}. \tag{2.14}$$

The proof of Theorem 3 is similar to that of Theorem 2.

With these two theorems, we can outline the proof of the sampling theorem. Figures 2.2 and 2.3 show a visualization of the major steps. The first step is to note that uniformly sampling a continuous signal, f, is equivalent to multiplying the signal with a dirac comb whose spacing is equal to the inverse of the sampling rate (left column of Figure 2.2). By Theorem 3, this is equivalent to convolving the Fourier transform of the signal, $\hat{f}$, with the Fourier transform of the dirac comb (right column of Figure 2.2). The Fourier transform of the dirac comb is itself a dirac comb with spacing equal to the sampling rate. Thus, in the Fourier domain the sampled version of f is a repetition of $\hat{f}$ spaced at integer multiples of the sampling rate. Note that to avoid aliasing caused by overlapping copies of $\hat{f}$, the signal must not contain frequencies greater than half the sampling rate (called the Nyquist rate). We can recover $\hat{f}$ by multiplying the sampled version by a rectangular window centred at zero whose width is equal to the sampling rate (right column of Figure 2.3). By Theorem 2, this is equivalent to convolving the sampled version of f with the inverse Fourier transform of the rectangular window (i.e. a sinc function, left column of Figure 2.3). In other words, we can reconstruct f according to Theorem 1.

We can now understand why a sampling rate of 44.1 kHz was chosen for compact disc audio. Humans can hear audio approximately in the range of 20 Hz to 20 kHz [Rosen and Howell, 2011]. According to Theorem 1 we require a sampling rate of at least 40 kHz in order to reconstruct the original signal so that it is perceptibly identical to the original. The specific value of 44.1 kHz was chosen due to compatibilities with existing video equipment which was used to record digital audio at the time [Pohlmann, 1989].

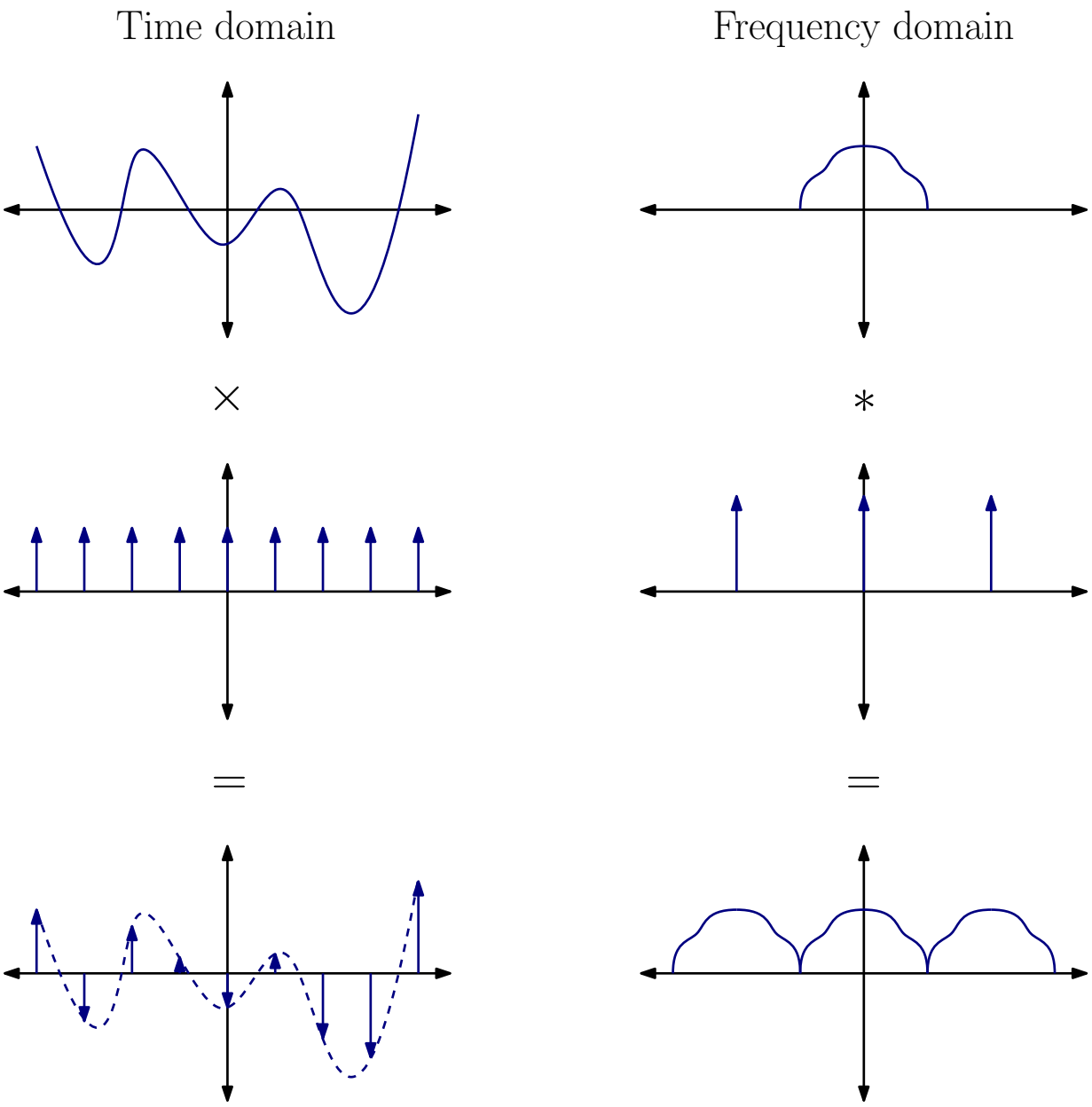


Figure 2.2: Sampling a signal in the time domain compared to the frequency domain.

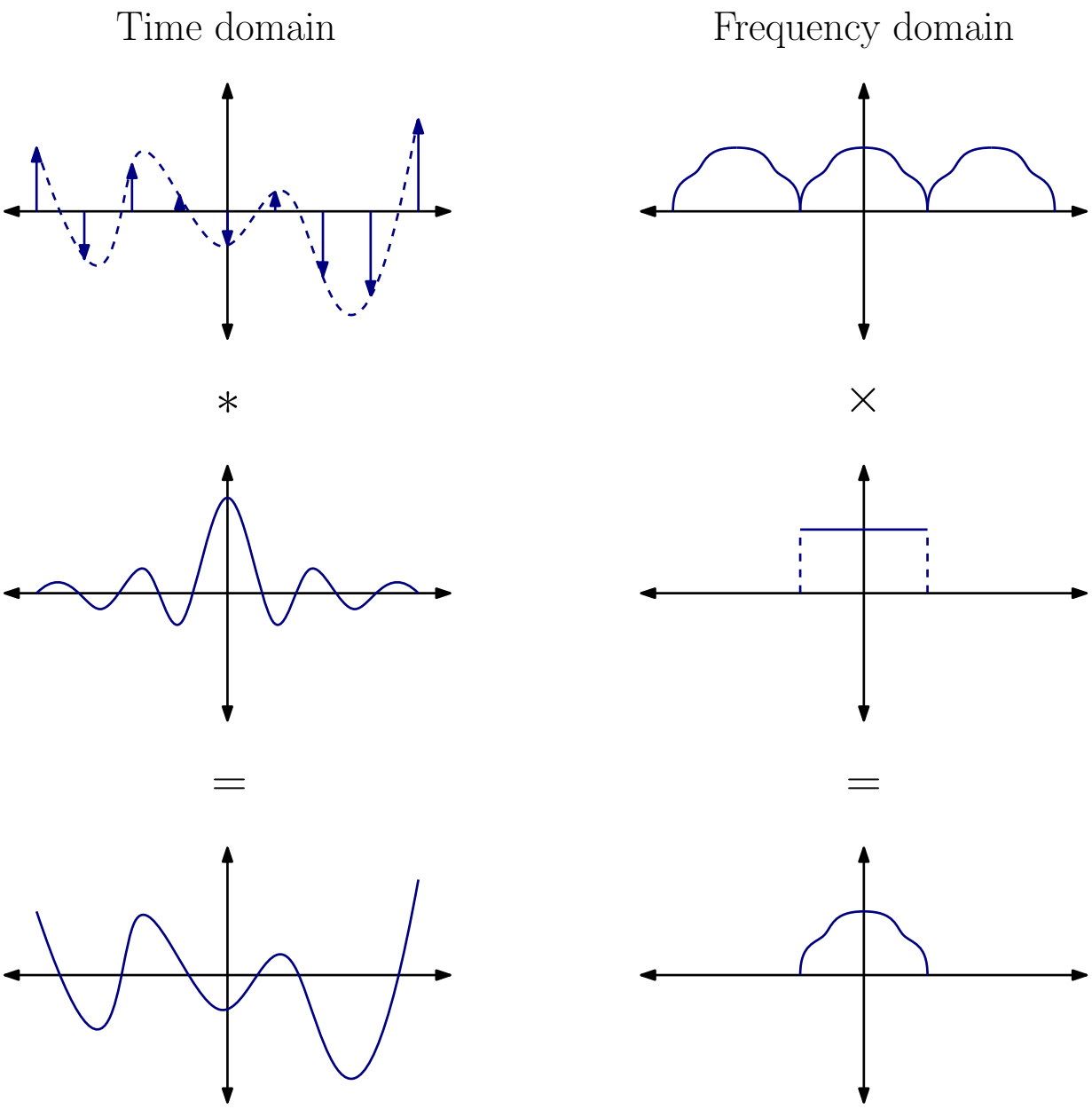


Figure 2.3: Reconstructing a continuous signal from sampled points in the time domain compared to the frequency domain.

2.3 Time-frequency Analysis

The coefficients of the Fourier transform are complex values. Typically the magnitude (or modulus) of the coefficients are used instead since it often easier to work with real values. As a consequence, we sacrifice the phase of the transform. In the case of audio signals, phase corresponds to temporal information. Consider the two signals in Figure 2.4. The first signal is an 8 Hz pulse followed by a 16 Hz pulse. The second signal contains the same two pulses, but in opposite order. Note that the magnitudes of their Fourier transforms are identical. We can overcome this limitation by moving to the domain of time-frequency transforms.

2.3.1 Windowed Fourier Transform

Up until now we have viewed signals in either the time or frequency domain with the use of the Fourier transform. The *windowed Fourier transform* (or short-time Fourier transform) is a method of viewing a signal in both time and frequency simultaneously [Gabor, 1946]. Suppose we would like to compute a Fourier transform of a signal that is localized in time. We can do so by introducing a symmetric windowing function $g : \mathbb{R} \to \mathbb{R}$, with $||g||_2 = 1$, where

$$||g||_2 = \int_{-\infty}^{+\infty} |g(x)|^2 dx \tag{2.15}$$

We can then consider the time-localized signal

$$f_u(t) = f(t)g(t-u) \tag{2.16}$$

where the signal f is localized around the time u. The Fourier transform of f_u is

$$\hat{f}_u(\xi) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f_u(t) e^{-i\xi t}\, dt = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f(t)g(t-u)e^{-i\xi t}\, dt \tag{2.17}$$

The windowed Fourier transform is computed by considering many different time localized versions of f, and is defined as

$$Sf(u,\xi) = \hat{f}_u(\xi) = \int_{-\infty}^{+\infty} f(t)g(t-u)e^{-i\xi t}\, dt \tag{2.18}$$

The discrete windowed Fourier transform is thus

$$Sf[m,l] = \sum_{n=0}^{N-1} f[n]g[n-m]e^{-i2\pi ln/N} \tag{2.19}$$

See Figure 2.5 for an example of the windowed Fourier transform applied to speech.

2.3.2 Linear Time-frequency Transforms

Let us now generalize the notion of the windowed Fourier transform to a *linear time-frequency transform*. Consider a dictionary $\{\phi_\gamma\}_{\gamma\in\Gamma}$ of *time-frequency atoms* where $||\phi_\gamma||_2 = 1$. We can think of each atom as being a function that is localized in both time and frequency. The linear time-frequency transform is defined as

$$\Phi f(\gamma) = \langle f, \phi_\gamma \rangle = \int_{-\infty}^{+\infty} f(t)\phi_\gamma^*(t)\, dt \tag{2.20}$$

where ϕ^* is the complex conjugate of ϕ.

The windowed Fourier transform can be viewed as a linear time-frequency transform. When the windowed Fourier transform was introduced earlier, the intuition was that we were taking the Fourier transform of time-localized sections of the signal. That is, a windowing function was applied to the signal and then its Fourier transform was computed as in Equation 2.17. Alternatively, let us consider windowing the Fourier basis functions instead. Let $g_{u,\xi}(t) = g(t-u)e^{i\xi t}$ be the time-frequency atoms in our dictionary (See Figure 2.6 for examples). Applying Equation 2.20, the windowed Fourier transform is a linear time-frequency transform of the form

$$Sf(u,\xi) = \langle f, g_{u,\xi} \rangle = \int_{-\infty}^{+\infty} f(t)g_{u,\xi}^*(t)\, dt \tag{2.21}$$

which is equivalent to Equation 2.18.

2.3.3 Wavelet Transform

One important class of linear time-frequency transforms is known as the *wavelet transform*, first introduced by [Haar, 1910]. To define the wavelet transform we must first generate a dictionary of wavelets. A wavelet is a function ψ localized at time $t = 0$ with zero average and $||\psi||_2 = 1$. A wavelet dictionary can then be generated from ψ:

$$\left\{ \psi_{u,s}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right) \right\}_{u\in\mathbb{R}, s\in\mathbb{R}^+} \tag{2.22}$$

Note that $||\psi_{u,s}||_2 = 1$. The wavelet transform is then defined as the following linear time-frequency transform

$$Wf(u,s) = \langle f, \psi_{u,s} \rangle = \int_{-\infty}^{+\infty} f(t)\psi_{u,s}^*(t)\, dt \tag{2.23}$$

See Figure 2.7 for an example of the wavelet transform applied to the signals from Figure 2.4.

To complete our discussion, the notion of a *scaling function* will be introduced. Let us define a function ϕ such that

$$|\hat{\phi}(\omega)|^2 = \int_1^{+\infty} \frac{|\hat{\psi}(s\omega)|^2}{s} ds = \int_\omega^{+\infty} \frac{|\hat{\psi}(\xi)|^2}{\xi} d\xi \tag{2.24}$$

with the phase of $\hat{\phi}$ being arbitrary. We call ϕ the scaling function of ψ, because it is a sum of all wavelets with scales greater than one. The scaling function is important, since we can only consider a finite number of wavelet scales in the discrete case. As such, we require a function that represents all remaining scales. Section 2.5 will give more intuition about the scaling function. See Figures 2.8 and B.2 for examples of typical wavelet and scaling functions.

There are several advantages to using the wavelet transform over the windowed Fourier transform:

1. Unlike the Fourier basis, there is no fixed wavelet basis. We can create wavelets that have different properties that may be useful for a specific task. For example, in the JPEG2000 standard, wavelets are chosen so that the wavelet decomposition of images is sparse, which is useful for compression [Christopoulos et al., 2000, Mallat, 2008]. In the lossy version of the transform, the biorthogonal 4,4 wavelet is used (also called the Cohen-Daubechies-Feauveau (CDF) 9/7 wavelet). See Figure 2.9 for an illustration of the wavelet and scaling functions.

2. We can adaptively tile the time-frequency plane with appropriate time-frequency atoms. Consider the time-frequency atoms of the windowed Fourier transform. Since the windows are fixed, each atom has a fixed spread in both time and frequency as in Figure 2.10a. On the other hand, we can choose our wavelet basis such that each atom has a different time frequency spread as in Figure 2.10b for example.

3. Building upon point 2, we can use dyadic scales ($s = 2^j$) so that we have a nonredundant transform. In other words, if our signal has N points we can choose our wavelet basis such that we have exactly N wavelet coefficients. Making this choice allows for the derivation of a linear time wavelet transform algorithm.

2.3.4 Discrete Wavelet Transform

Recall the wavelet dictionary

$$\left\{\psi_{u,s}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-u}{s}\right)\right\}_{u\in\mathbb{R},s\in\mathbb{R}^+} \tag{2.25}$$

In order to analyse discrete signals, a discrete wavelet basis is required. First, let us create a discrete set of continuous wavelets by discretizing s and u. Let $u = ks$ and $s = a^j$ for $k, j \in \mathbb{Z}$ and $a > 1$. Let

$$\psi_j[n] = \frac{1}{a^j}\psi\left(\frac{n}{a^j}\right). \tag{2.26}$$

The discrete wavelet transform is defined as [Mallat, 2008]

$$Wf[n, a^j] = \sum_{m=0}^{N-1} f[m]\psi_j^*[m-n] = f \circledast \bar{\psi}_j[n] \tag{2.27}$$

where $\bar{\psi}_j[n] = \psi_j^*[-n]$ and $\circledast$ is circular convolution.

Similarly, let us define the discrete scaling function

$$\phi_j[n] = \frac{1}{a^j}\phi\left(\frac{n}{a^j}\right) \tag{2.28}$$

and $\bar{\phi}_j[n] = \phi_j^*[-n]$. Let

$$Lf[n, a^j] = \sum_{m=0}^{N-1} f[m]\phi_j^*[m-n] = f \circledast \bar{\phi}_j[n]. \tag{2.29}$$

be an approximation of f without high frequency components. This transform corresponds to all scales greater than a^j. We require a scaling function to fully characterize f since the wavelet transform can only be computed up to a finite scale. We will assume $a = 2$ for the remainder of the document (otherwise known as dyadic sampling). This means that each wavelet scale will cover one octave of the signal as in Figure 2.10b.

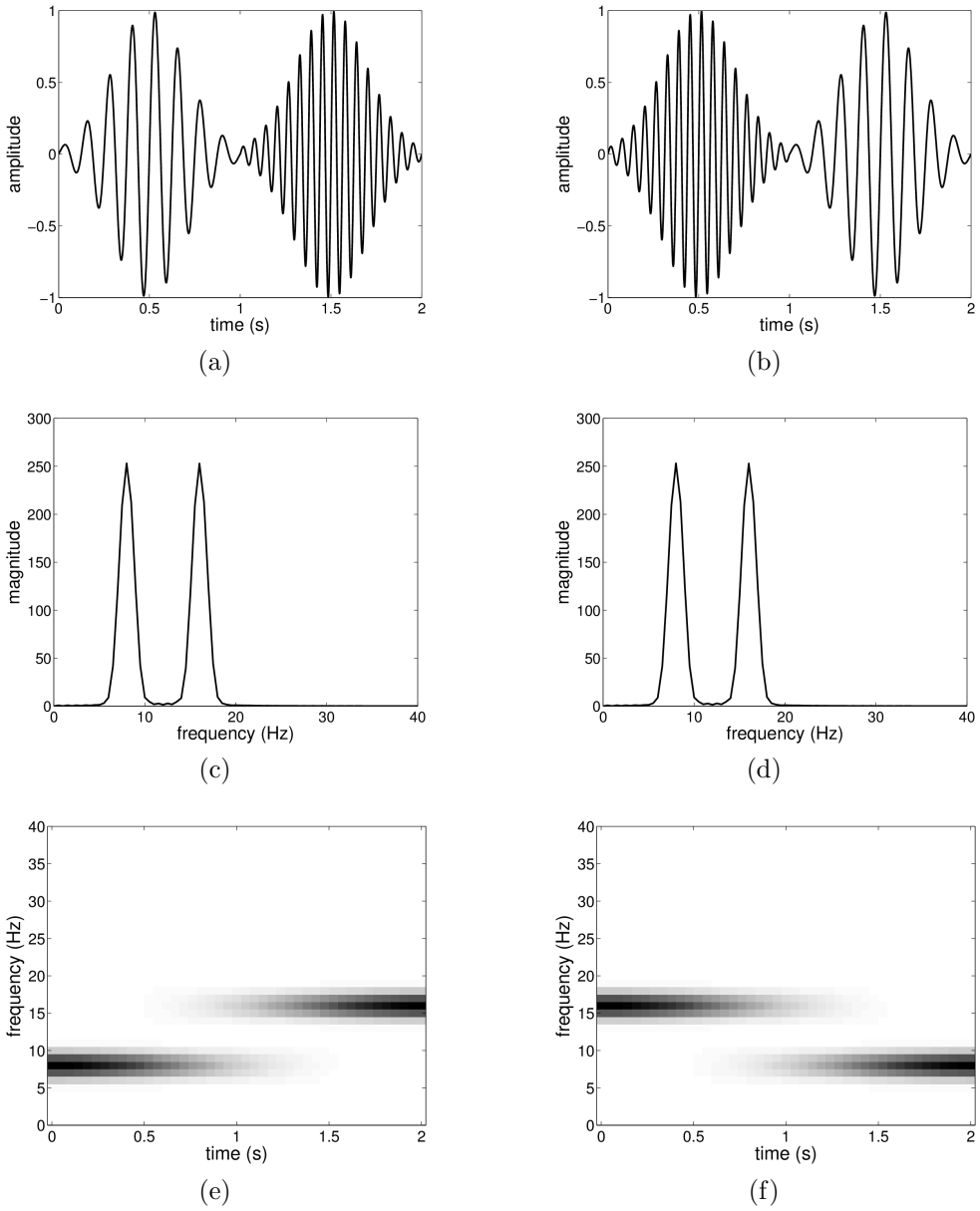


Figure 2.4: (a) A low frequency pulse followed by a high frequency pulse. (b) A signal opposite to (a). (c,d) Magnitudes of the Fourier transform of (a,b) respectively. (e,f) Magnitudes of the short-time Fourier transform of (a,b) respectively.

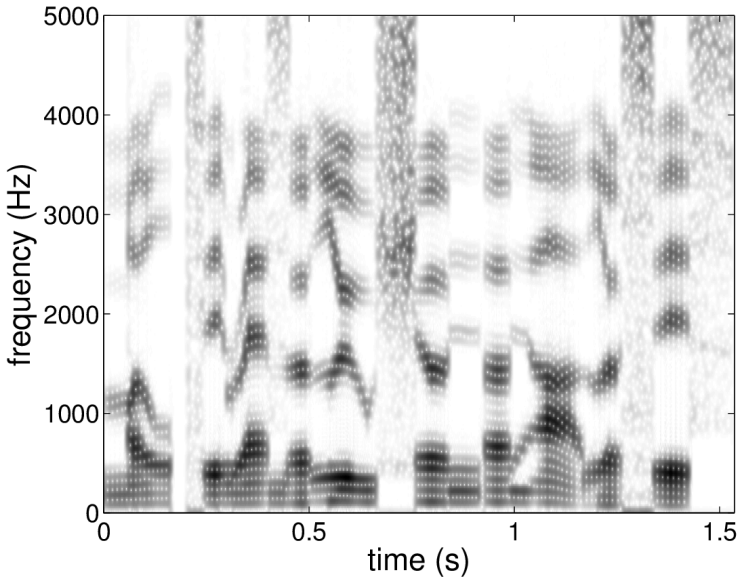


Figure 2.5: Log magnitude of the windowed Fourier transform of synthetic speech. The text is: "multiresolution analysis". The speech was generated using the software Praat [Boersman and Weenink, 2013].

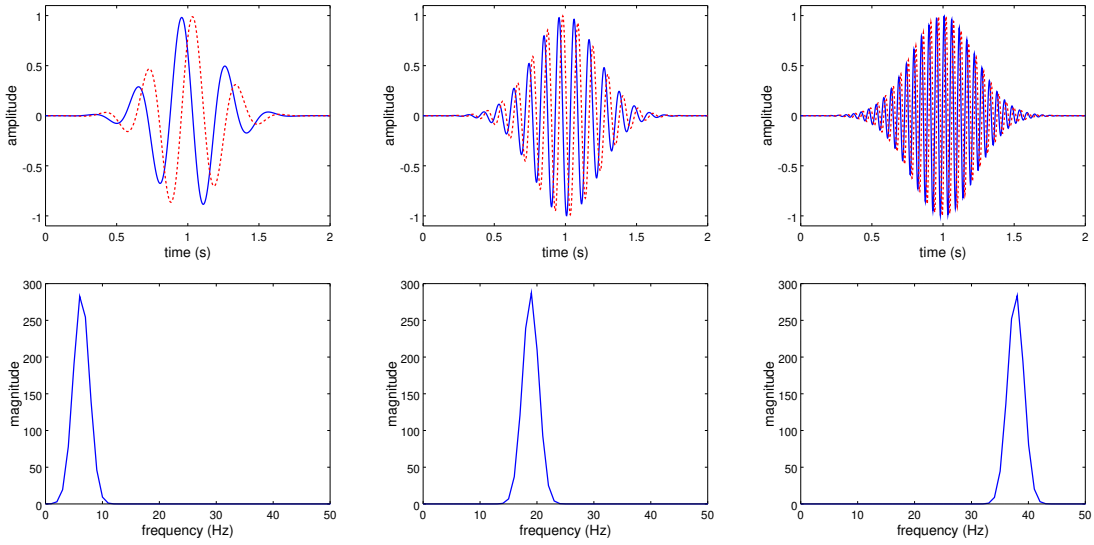


Figure 2.6: Top: Plots of $g_{u,\xi}(t) = g(t-u)e^{i\xi t}$ for various frequencies with real parts shown in solid blue and imaginary parts shown in dashed red. Bottom: Magnitudes of the Fourier transforms of the functions. Note that the spread in the Fourier domain is constant.

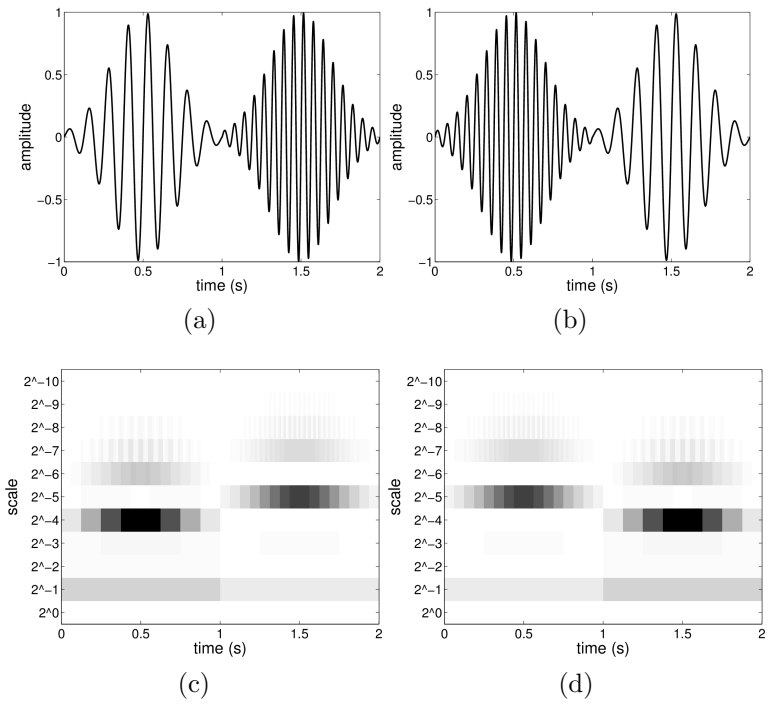


Figure 2.7: (a) A low frequency pulse followed by a high frequency pulse. (b) A signal opposite to (a). (c,d) Magnitudes of the Haar wavelet transform of (a,b) respectively.

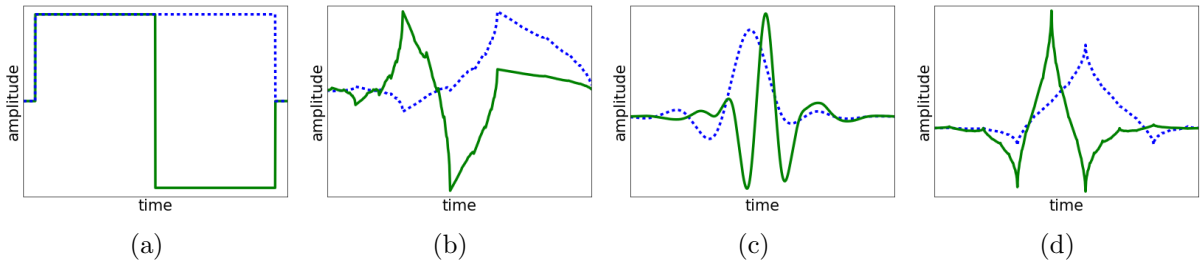


Figure 2.8: Examples of wavelet (solid) and scaling (dashed) functions: (a) Haar, (b) Daubechies, (c) Symlets, and (d) Coiflets.

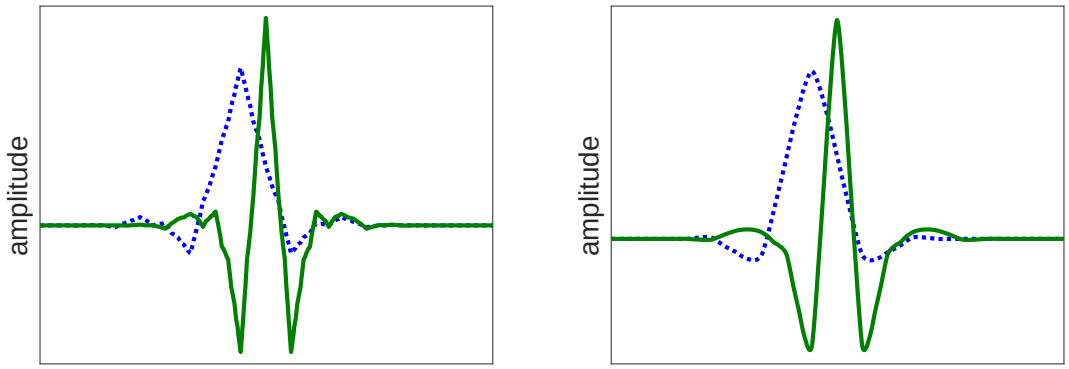


Figure 2.9: The biorthogonal 4,4 wavelet (solid) and scaling (dashed) functions used in JPEG2000 lossy compression. Since the functions are biorthogonal, there are separate (a) analysis and (b) synthesis functions.

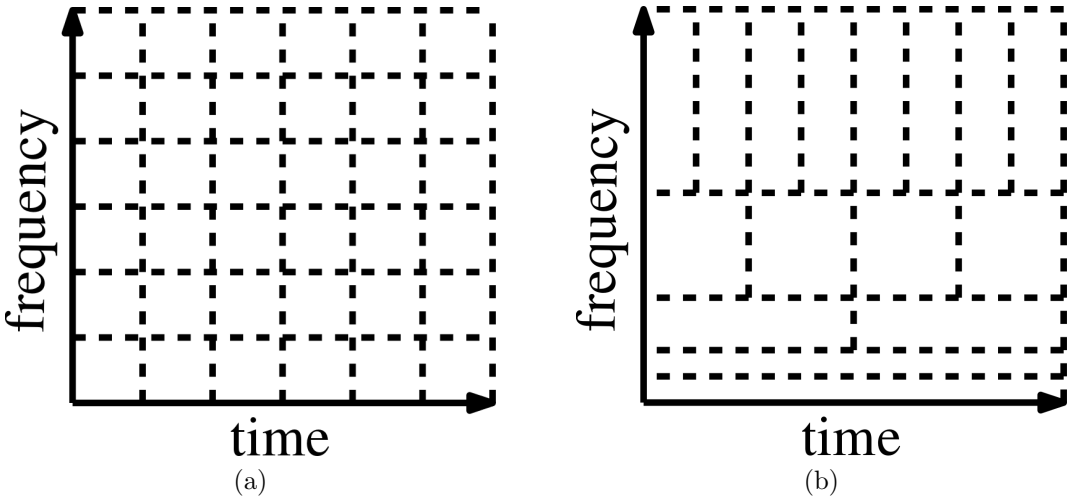


Figure 2.10: Tiling of the time-frequency plane by (a) windowed Fourier transform, and (b) the wavelet transform with dyadic sampling.

2.4 A Fast Discrete Wavelet Transform Algorithm

In this section the wavelet transform is redefined in terms of filters. Let

$$a_j[n] = \langle f, \phi_{j,n} \rangle \tag{2.30}$$

and

$$d_j[n] = \langle f, \psi_{j,n} \rangle \tag{2.31}$$

be the projections of f onto $\{\phi_{j,n}\}_{n\in\mathbb{Z}}$ and $\{\psi_{j,n}\}_{n\in\mathbb{Z}}$. We use a and d to represent "approximate" (lowpass) and "detail" (highpass). The d_j's and a_j's correspond to the wavelet and scaling coefficients in Equations 2.27 and 2.29 respectively. Let us define two filters,

$$h[n] = \left\langle \frac{1}{\sqrt{2}}\phi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle \tag{2.32}$$

$$g[n] = \left\langle \frac{1}{\sqrt{2}}\psi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle \tag{2.33}$$

Equations 2.32 and 2.33 are equivalent to

$$\phi(t) = \sum_n \sqrt{2}h[n]\phi(2t-n) \tag{2.34}$$

$$\psi(t) = \sum_n \sqrt{2}g[n]\phi(2t-n) \tag{2.35}$$

The following two theorems connect the wavelet coefficients to the filters h and g, and give rise to a recursive algorithm for computing the wavelet transform [Mallat, 2008]:

Theorem 4. *Wavelet Filterbank Decomposition*

$$a_{j+1}[p] = \sum_{n=-\infty}^{+\infty} h[n-2p]a_j[n] = (a_j * \bar{h})[2p] \tag{2.36}$$

$$d_{j+1}[p] = \sum_{n=-\infty}^{+\infty} g[n-2p]a_j[n] = (a_j * \bar{g})[2p] \tag{2.37}$$

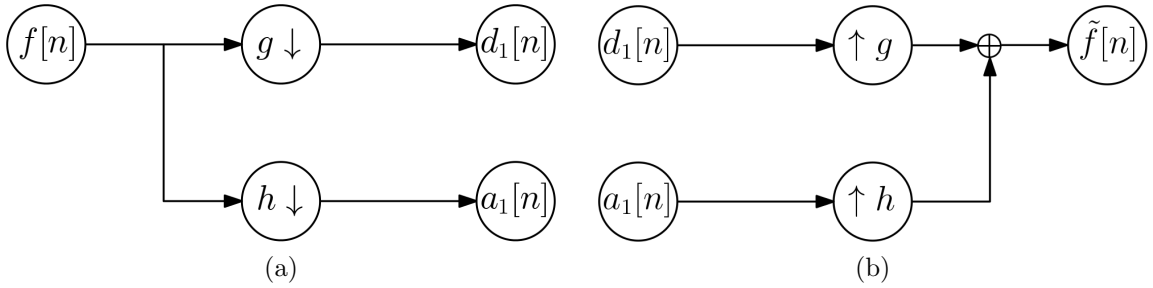


Figure 2.11: A graphical representation of a single step of the wavelet (a) decomposition and (b) reconstruction. The down and up arrows correspond to downsampling and upsampling respectively.

Theorem 5. *Wavelet Filterbank Reconstruction*

$$\begin{aligned} a_j[p] &= \sum_{n=-\infty}^{+\infty} h[p-2n]a_{j+1}[n] + \sum_{n=-\infty}^{+\infty} g[p-2n]d_{j+1}[n] \\ &= (\check{a}_{j+1} * h)[p] + (\check{d}_{j+1} * g)[p] \end{aligned} \tag{2.38}$$

where $\check{}$ denotes that the signal is upsampled by inserting zeros at odd indices, and an overbar represents the reverse of the filter.

Theorems 4 and 5 demonstrate how we can compute the wavelet transform as a cascade of filter convolutions and downsamplings. Figure 2.11 shows a graphical representation of a single step of the wavelet decomposition and reconstruction. Figure 2.12 demonstrates the filter cascade. The discrete wavelet transform can be computed as follows: set $a_0[n] = f[n]$, then recursively compute the approximation and detail coefficients as per Theorem 4. The forward and inverse algorithms are shown in Algorithms 2.1 and 2.2. In the remainder of this thesis, we will refer to these algorithms simply as the wavelet transform and the inverse wavelet transform.

2.4.1 Algorithm Analysis

The wavelet transform algorithm can be computed efficiently using the decimating algorithm described in Algorithm 2.1. A simple way to determine the time complexity is to make use of the master theorem [Bentley et al., 1980]. The master theorem is a method for determining the time complexity of recurrent algorithms and is defined below:

Algorithm 2.1 Discrete wavelet transform

Require: f is a discrete signal, L is the maximum level of the transform.

1: **function** DWT(f, L)
2: $a_0 \leftarrow f$
3: **for** $j \leftarrow 0$ to $L-1$ **do**
4: $a_{j+1} \leftarrow downsample(a_j * \bar{h})$
5: $d_{j+1} \leftarrow downsample(a_j * \bar{g})$
6: **end for**
7: **return** $\langle d_1, d_2, \ldots, d_L, a_L \rangle$
8: **end function**

Theorem 6. *(Master Theorem, simplified). If an algorithm has the recurrence relation*

$$T(n) = aT(n/b) + \Theta(n^c), \quad a \geq 1, b > 1 \tag{2.39}$$

where
n *is the input size*
$a \geq 1$ *is the number of subproblems*
n/b *is the input size to the subproblems*
$f(n)$ *is the time complexity of work outside of the recursive calls*
then the time complexity of the algorithm can be computed as follows:

Case 1: $c < \log_b a \quad \Rightarrow \quad T(n) = \Theta(n^{\log_b a})$

Case 2: $c = \log_b a \quad \Rightarrow \quad T(n) = \Theta(n^c \log n)$

Case 3: $c > \log_b a \quad \Rightarrow \quad T(n) = \Theta(n^c)$

Though we presented the discrete wavelet transform in an iterative way in Algorithm 2.1, we can equivalently view the algorithm recursively. In this view Figure 2.12 can be interpreted as a computation tree after three recursive calls. The recursive algorithm is shown in Algorithm 2.3. The initial call is $DWT(f, L)$.

In order to determine the time complexity of Algorithm 2.3 , we must first determine the time complexity of a single recursive call. In the base case the input is simply returned, so the time complexity is $O(1)$. Otherwise, two convolutions are performed, followed by downsampling. Recall that we defined downsampling as discarding every second element of the signal. Hence, the time complexity of downsampling is $O(n)$.

Algorithm 2.2 Inverse discrete wavelet transform

Require: $\langle d_1, d_2, \ldots, d_L, a_L \rangle$ as computed per Algorithm 2.1.

1: **function** IDWT($\langle d_1, d_2, \ldots, d_L, a_L \rangle$)
2: **for** $j \leftarrow L - 1$ to 0 **do**
3: $a_j \leftarrow (upsample(a_{j+1}) * h) + (upsample(d_{j+1}) * g)$
4: **end for**
5: **return** a_0
6: **end function**

Algorithm 2.3 Discrete wavelet transform (recursive)

Require: a is initially a discrete signal, L is the maximum level of the transform.

1: **function** DWT(a, L)
2: **if** $L = 0$ **then return** $\langle a \rangle$
3: **else**
4: $a' \leftarrow downsample(a * \bar{h})$
5: $d \leftarrow downsample(a * \bar{g})$
6: **return** concat($\langle d \rangle, \langle DWT(a', L - 1) \rangle$)
7: **end if**
8: **end function**

The time complexity of convolution will take a bit more consideration. Let f and g be discrete signals of lengths n and m respectively. Without loss of generality, we will assume $n \geq m$. Convolution can then be computed per its definition in $O(nm)$ time. In the case that $n \cong m$, this time complexity becomes $O(n^2)$. We can do better than $O(n^2)$ by making use of the convolution theorem. Recall that

$$h = f * g \quad \Rightarrow \quad \hat{h}(\omega) = \hat{f}(\omega)\hat{g}(\omega). \tag{2.40}$$

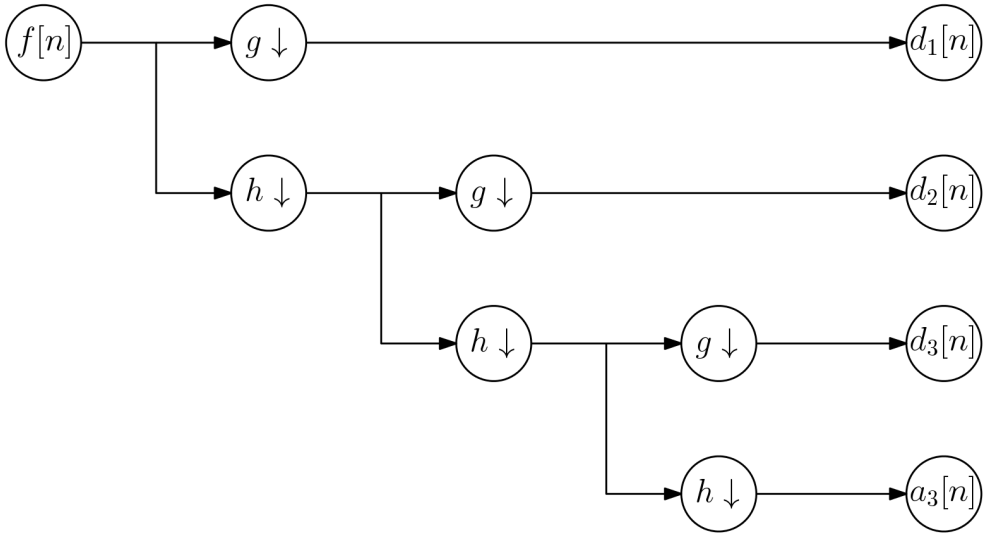
Therefore, we can compute $f * g$ by computing their Fourier transforms, performing an element-wise multiplication, then performing an inverse Fourier transform on the result. The Fourier transform can be computed in $O(n \log n)$ time by the fast Fourier transform algorithm [Cooley and Tukey, 1965]. The inverse transform can be computed in the same time complexity. Thus, the time complexity for computing convolution in the Fourier domain is $O(n \log n)$.

The above analysis of convolution considered the worst case where the signal and filter are of similar sizes. In the case of the wavelet transform, the filter size is not only generally

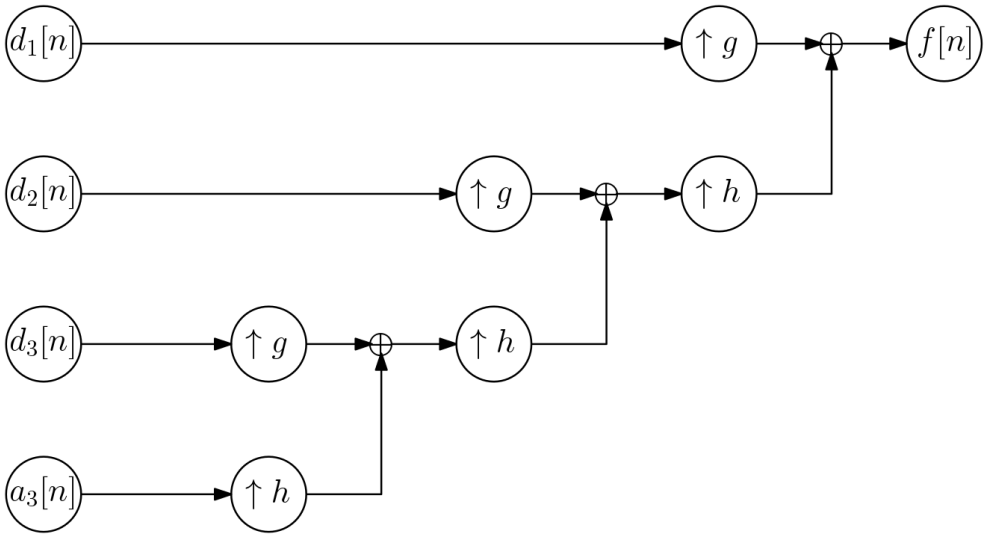
much smaller than the signal size, but its size is constant. This means that the convolution can be computed in $O(n)$ time (or more precisely, $\Theta(n)$ time). Therefore, the recurrence relation of Algorithm 2.3 is

$$T(n) = T(n/2) + \Theta(n). \tag{2.41}$$

In other words, each recursive call of the algorithm performs $\Theta(n)$ work and makes a recursive call on a signal half the size of its original input. Using the master theorem we arrive at a final time complexity of $\Theta(n)$. The inverse wavelet transform can similarly be shown to have $\Theta(n)$ time complexity.



(a)



(b)

Figure 2.12: A graphical representation of the cascade of filters in the wavelet (a) decomposition and (b) reconstruction. The down and up arrows correspond to downsampling and upsampling respectively.

2.5 Multiresolution Analysis

This section aims to give a formal definition of multiresolution analysis [Mallat, 2008]. In our previous discussion, we motivated the wavelet transform as a general method of time-frequency representation. In order to compute an invertible (i.e. lossless) transform using a finite number of wavelets, we found the need to introduce a scaling function. This treatment of the wavelet transform led to a somewhat awkward transition to the filterbank implementation. In this section, the wavelet transform is introduced by way of function approximation. The discussion will begin with the notion of the scaling function, and then motivate the usefulness of a wavelet function. This treatment will provide further intuition into the filterbank view of the wavelet transform.

Sampling continuous signals in order to produce discrete versions is a natural application for function approximation. In the case of audio signals, we can view sampling as approximating a continuous function $f \in L^2(\mathbb{R})$ (i.e. f is a square-integrable function) using a basis of piecewise constant functions whose width is the inverse of the sampling frequency. In other words, let us define

$$\phi(t) = \begin{cases} 1 & \text{if } 0 \leq t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.42}$$

This function is called the Haar scaling function. We can then approximate f by representing it as a sum of shifted versions of ϕ. If we shift ϕ by integer values, then each shifted version will be orthogonal to the other shifted versions. See Figure 2.13 for an illustration of sampling a continuous signal.

We can achieve a better approximation of f by decreasing the width of ϕ. Recall that we previously defined

$$\phi_{u,s}(t) = \frac{1}{\sqrt{s}}\phi\left(\frac{t-u}{s}\right) \tag{2.43}$$

Let us set $s = 2^j$ and $u = k2^j$, for integers j and k so that at each scale we halve the width of the scaling function. For convenience we will use the notation

$$\phi_{(j,k)}(t) = \phi_{k2^j,2^j}(t) = 2^{-j/2}\phi\left(2^{-j}t - k\right) \tag{2.44}$$

Note that the scale and shift parameters have been swapped.

We then have $\phi_{(0,k)}(t) = \phi(t-k)$. Let us consider the set of $L^2(\mathbb{R})$ functions spanned by $\{\phi_{(0,k)}\}_{k\in\mathbb{Z}}$. That is, the set of functions in $L^2(\mathbb{R})$ that are constant over the intervals

$[k, k+1)$. We will denote this set as V_0. We can similarly define V_{-1} as the set of functions spanned by $\{\phi_{(-1,k)}\}_{k\in\mathbb{Z}}$ (the functions that are constant over $[\frac{k}{2}, \frac{k+1}{2})$). What can we say about the relationship between V_0 and V_{-1}? It is easy to see that $V_0 \subset V_{-1}$ since any function that is constant over $[k, k+1)$ is also constant over $[\frac{k}{2}, \frac{k+1}{2})$.

We can go further and define V_j to be the set of functions in $L^2(\mathbb{R})$ spanned by the scaling functions at scale j. Then we have $V_{j+1} \subset V_j$ for all $j \in \mathbb{Z}$. Since $V_0 \subset V_{-1}$ we can write

$$\phi_{(0,k)}(t) = \sum_{k=-\infty}^{+\infty} h_k \phi_{(-1,k)}(t) \tag{2.45}$$

for coefficients $h_k \in \mathbb{R}$. Equation 2.45 is more commonly expressed as

$$\phi(t) = \sum_{k=-\infty}^{+\infty} h_k \sqrt{2}\phi(2t-k) \tag{2.46}$$

For the Haar scaling function $h_0 = h_1 = 1/\sqrt{2}$, with all other h_k being zero.

Let us now consider the set of functions in V_{-1} that are orthogonal to the functions in V_0. We will denote this set W_0. We then have

$$V_{-1} = V_0 \oplus W_0 \tag{2.47}$$

where $\oplus$ is the direct sum. Thus any function in V_{-1} can be represented as a sum of functions from V_0 and W_0. Suppose we wish to find an orthogonal set of functions that span W_0. Similar to the scaling function, we can define a function ψ such that shifted versions of ψ form an orthonormal basis for W_0. Since $\psi \in W_0 \subset V_{-1}$, we have

$$\psi(t) = \sum_{k=-\infty}^{+\infty} g_k \sqrt{2}\phi(2t-k) \tag{2.48}$$

for coefficients $g_k \in \mathbb{R}$. Notice the similarity to Equation 2.46. For the case of the Haar scaling function, ψ is defined as

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \le t < 1/2 \\ -1 & \text{if } 1/2 \le t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.49}$$

In this case $g_0 = 1/\sqrt{2}, g_1 = -1/\sqrt{2}$, with all other g_k being zero.

Since $\{\phi_{(0,k)}\}_{k\in\mathbb{Z}}$ and $\{\psi_{(0,k)}\}_{k\in\mathbb{Z}}$ form an orthonormal basis for V_0 and W_0 respectively, any $f \in V_{-1}$ can be written as

$$f(t) = \sum_{k=-\infty}^{+\infty} a_k \phi_{(0,k)}(t) + \sum_{k=-\infty}^{+\infty} d_k \psi_{(0,k)}(t) \tag{2.50}$$

where $a_k = \langle f, \phi_{(0,k)} \rangle$ and $d_k = \langle f, \psi_{(0,k)} \rangle$.

We can repeat this process for V_{-2} and find an orthonormal set of functions that span W_{-1} such that $V_{-1} \cap W_{-1} = \{0\}$ and $V_{-2} = V_{-1} \oplus W_{-1}$. These functions are exactly $\{\psi_{(-1,k)}\}_{k\in\mathbb{Z}}$. Substituting into Equation 2.47 yields

$$V_{-2} = V_0 \oplus W_0 \oplus W_{-1} \tag{2.51}$$

Any $f \in V_{-2}$ can be written as

$$f(t) = \sum_{k=-\infty}^{+\infty} a_k \phi_{(0,k)}(t) + \sum_{k=-\infty}^{+\infty} d_{0,k} \psi_{(0,k)}(t) + \sum_{k=-\infty}^{+\infty} d_{-1,k} \psi_{(-1,k)}(t) \tag{2.52}$$

where $d_{j,k} = \langle f, \psi_{(j,k)} \rangle$. We can generalize this to any scale j so that

$$V_j = V_0 \oplus W_0 \oplus W_{-1} \oplus \cdots \oplus W_{j+1} \tag{2.53}$$

Since positive scales are also valid, $L^2(\mathbb{R})$ can be represented as an infinite sum of $W_j's$

$$L^2(\mathbb{R}) = V_0 \oplus \left[\bigoplus_{j=0}^{-\infty} W_j \right] \tag{2.54}$$

or

$$L^2(\mathbb{R}) = \bigoplus_{j=-\infty}^{+\infty} W_j \tag{2.55}$$

Equation 2.54 is illustrated in Figure 2.14. Generalizing Equation 2.52 to all scales using Equation 2.54 is exactly the discrete wavelet decomposition from Section 2.4.

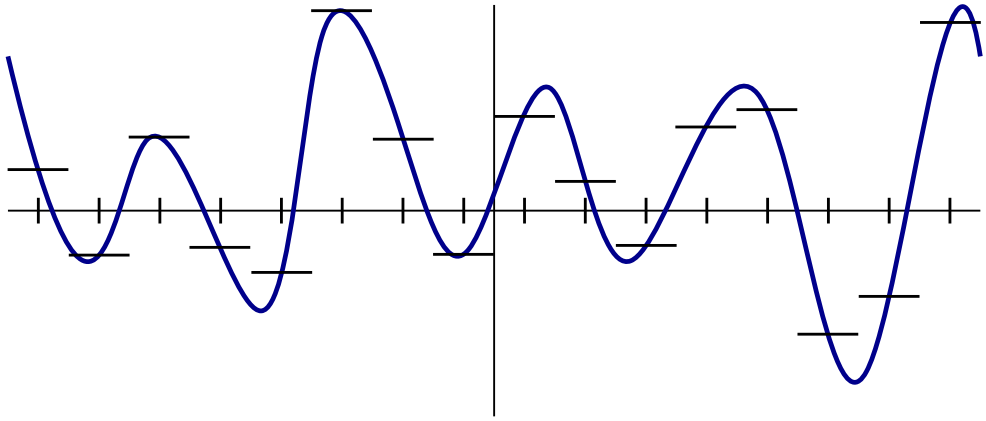


Figure 2.13: Approximation of a continuous signal with piecewise constant functions of equal width.

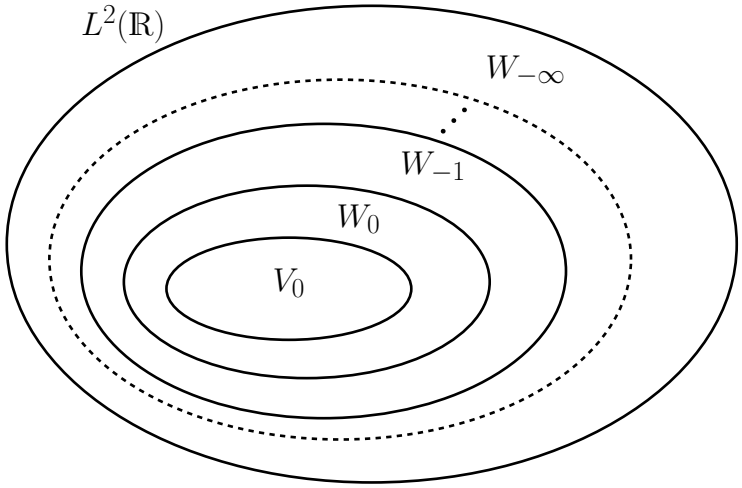


Figure 2.14: An illustration of Equation 2.54. We can represent $L^2(\mathbb{R})$ as the direct sum of function spaces.

2.6 Wavelets with Compact Support

In order to compute the wavelet transform as described above, we require filters that have a finite number of nonzero coefficients. These filters correspond to scaling and wavelet functions with compact (or finite) support. That is, the functions are zero outside of a particular interval. There are several necessary conditions on the filter coefficients so that the functions have finite support [Burrus et al., 1997]:

Finite L^1-norm

$$\sum_n h[n] = \sqrt{2} \tag{2.56}$$

Orthogonality:

$$\forall p \in \mathbb{Z}, \sum_n h[n]h[n-2p] = \begin{cases} 1 & \text{if } p = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.57}$$

Finite energy (follows from Equation 2.57 with $p = 0$):

$$\sum_n h[n]^2 = 1 \tag{2.58}$$

Odd-even equality (follows from Equations 2.56 and 2.57):

$$\sum_n h[2n] = \sum_n h[2n+1] \tag{2.59}$$

These conditions will be important when we introduce our wavelet learning model in Chapter 4. We note here that these conditions are merely *necessary*, and not sufficient. Consider the following filter with $h[0] = h[3] = 1/\sqrt{2}$, and zero elsewhere. We can easily verify that such a filter satisfies Equations 2.56-2.59. Now let us consider the scaling function induced by the filter by substituting h into Equation 2.34:

$$\phi(t) = \phi(2t) + \phi(2t-3) \tag{2.60}$$

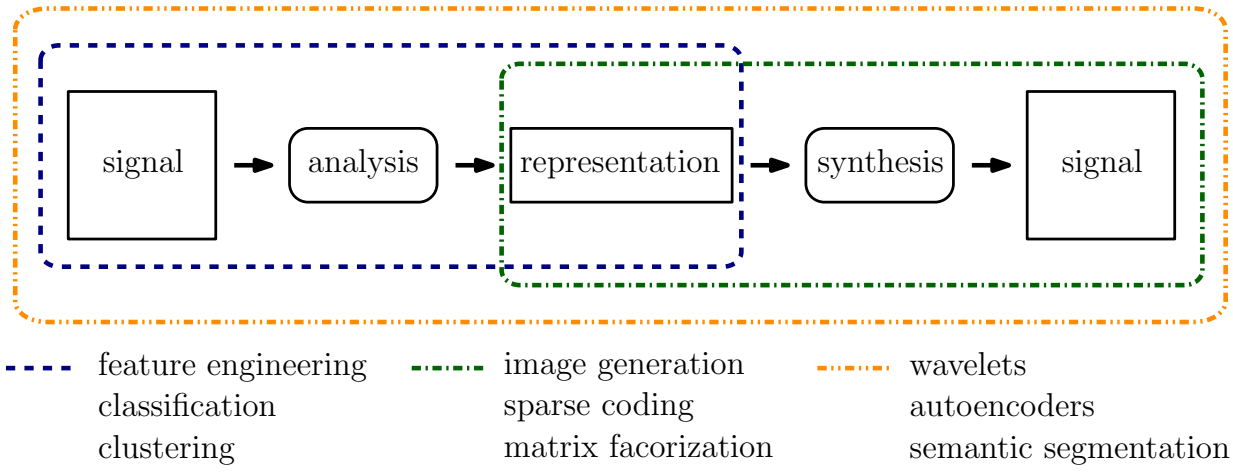
which is satisfied by

$$\phi(t) = \begin{cases} 1 & \text{if } 0 \le t \le 3 \\ 0 & \text{otherwise} \end{cases} \tag{2.61}$$

Such a ϕ does not have the property that integer translates are orthogonal. Hence, the integer translates do not form an orthonormal basis for V_0. This particular "degenerate" solution is pointed out in [Burrus et al., 1997] during their discussion of parameterized wavelets (introduced later).

Chapter 3

Motivation and Related Work



We have seen in Chapter 2 that wavelets provide a useful representation for signals. Furthermore, the fast discrete wavelet transform algorithm provides a non-redundant representation that is efficient with a time complexity of $O(n)$. Compare this to the $O(n \log n)$ time complexity of the fast Fourier transform. One downside of the wavelet transform is that the user must choose a particular wavelet function. It can be unclear, even for an expert, what wavelet function to use for a particular task. As such, the main goal of this thesis is to propose a method for choosing a wavelet function that is tuned for the task and data at hand. Adaptively choosing wavelets has a long history, dating back to early work in [Tewfik et al., 1992]. We will take a different approach that is more in line with the current trend in machine learning.

Though the wavelet transform has many useful properties, it is seldom used in modern neural network architectures that dominate the machine learning literature. Part of the reason may be that the theory of wavelets is intimidating without enough background knowledge. We hope that this thesis provides a short and concise introduction to wavelets that is accessible to the general machine learning audience. Furthermore, a requirement of our work will be to provide a framework that is compatible with current neural network architectures. In other words, we want our method to be defined in terms of neural networks so that it is approachable to current practitioners.

3.1 Thesis Contribution

In this thesis, we present a novel method for learning wavelet filters that are tuned to data. Though we are not the first to consider this problem, our method is unique in its particular formulation. The method of wavelet learning presented in this thesis has the following properties that make it unique from other methods:

1. the learned bases are non-redundant and orthogonal, leading to a fast implementation
2. the ℓ_1 norm is used for sparsity
3. any dimension signal can be considered
4. wavelet filter values are learned directly (no paramerization)
5. the learning method makes use of only gradient descent
6. the model can be incorporated into neural network architectures

Properties 1 and 2 are chosen as restrictions to the framework to simplify discussion and limit the scope of the thesis. In Chapter 7 we will discuss how these restrictions can be relaxed. The remaining properties will make our method robust, and usable in many different tasks.

Our proposed method will not restrict itself to a fixed signal dimension. It is common in the literature to propose methods only for a particular dimension (such as audio or images). Without this restriction, our model can be applied to wide variety of data. We will explore several different types of data in later chapters including: audio, natural images, x-rays, and MRI scans. We hope that this flexibility will allow our method to be attractive to a wide audience.

The choice of learning the wavelet filter values directly may seem arbitrary. Indeed, the reader may wonder what advantages this has over using parameterized wavelets. This property was chosen for two reasons. The first is that learning filter values directly is the method employed in neural networks. Hence, it is in line with our goal of making our method compatible with neural network architectures. The second reason, and possibly the more important, is that it provides flexibility to our model. Though we will only consider orthogonal wavelets in this work, it is a simple task to modify the framework to consider other classes of wavelets. For example, we may wish to consider biorthogonal wavelets. We can do so by making small changes to our framework, without the need of deriving new parameterizations.

The restriction of the learning method to gradient descent is necessary if we desire our method to be incorporated into modern neural network architectures. The most prevalent method of training neural networks is stochastic gradient descent, or its variants (See Appendix A for details and discussion). Furthermore, many neural networks are implemented in one of several popular software frameworks. As such, we will implement our model purely in Tensorflow [Abadi et al., 2015].

3.2 Related Work

The work in this thesis overlaps with several different areas of research. In this section, we provide a short review of the most relevant work. The main relevant areas are sparse dictionary learning, wavelet optimization, and second generation wavelets. We also discuss how fixed wavelet representations have been used in neural network architectures.

3.2.1 Sparse Dictionary Learning

A closely related area to our work is sparse dictionary learning (or sparse coding). Suppose we have a set of M signals of length N represented by the matrix $X \in \mathbb{R}^{N \times M}$. Our goal is to to find a dictionary, $D \in \mathbb{R}^{N \times K}$, of K basis atoms and a code (or representation), $A \in \mathbb{R}^{K \times M}$, such that

$$||X - DA||_2^2 + L(A) \tag{3.1}$$

is minimized. We represent the sparsity penalty as L. Traditionally, L is the zero norm $||\cdot||_0$ (i.e. the number of non-zero elements of A). The ℓ_1 norm is also commonly used since it can be shown to give sparse solutions for many linear systems [Donoho, 2006]. Often

the dictionary is over-complete so that there are more basis atoms than the dimension of the signals ($K > N$).

When D is fixed, the problem of finding a sparse representation is known as sparse approximation. Even with this constraint, the problem is NP-hard [Natarajan, 1995]. Approximate solutions can be found using algorithms such as Matching Pursuit [Mallat and Zhang, 1993] or LASSO [Tibshirani, 1996].

In our case, we are interested in learning the dictionary. Common techniques to dictionary learning include PCA (principal component analysis) [Pearson, 1901, Hotelling, 1933], ICA (independent component analysis) [Jutten and Herault, 1991], and NMF (nonnegative matrix factorization) [Lee and Seung, 1999]. Perhaps the most well-known work in this area is by Olshausen and Field, who showed that oriented, bandpass basis atoms arise from natural images [Olshausen and Field, 1996]. Other methods for dictionary learning include MOD (method of optimal directions) which relies on finding the pseudoinverse of A [Engan et al., 1999], and K-SVD (a generalization of K-means) [Aharon et al., 2006]. These methods do not generally impose constraints on the structure of the dictionary, which leads to computational costs. In particular, the methods tend to be applied to small signal patches instead of full signals [Ophir et al., 2011].

Multi-scale dictionary learning is a restriction to general dictionary learning. In this problem we seek a dictionary that uses a set of basis atoms that share structure at different scales. This notion is very similar to multiresolutional analysis. Of particular interest is a technique making use of overcomplete, non-orthogonal wavelet atoms [Olshausen et al., 2001, Sallee and Olshausen, 2003]. The method was restricted to images, but was later extended to one-dimensional signals [Sallee, 2004]. The learned wavelet bases were oriented and shared similar properties to the steerable pyramid basis [Simoncelli and Freeman, 1995]. An alternative multi-scale method that does not make use of wavelets was introduced in [Mairal et al., 2007, Mairal et al., 2008b]. Their work is an extension of the K-SVD method, and also makes use of an overcomplete basis. Another variant of K-SVD can be found in [Rubinstein et al., 2010] where dictionary atoms are constrained to be linear combinations of a fixed set of basis elements (such as the discrete cosine basis). Further work has been done on extending K-SVD using wavelets [Ophir et al., 2011].

In general, the multi-scale dictionary learning methods described in this section make use of a overcomplete (or redundant) basis. On the other hand, this thesis will focus on non-redundant representations. Specifically, we will concern ourselves with orthogonal, non-oriented wavelet bases (though we will show that we can extend our work to learn oriented bases). An advantage of this approach is that our learned dictionary will be fully defined by a single mother wavelet atom. Furthermore, orthogonality will mean that we

can reconstruct signals from their code without having to learn a different synthesis basis.

3.2.2 Optimal Wavelets

Finding optimal orthogonal wavelets for a given signal is not a new idea. For example, [Mandal et al., 1996] find optimal wavelets by searching over known wavelets, and [Szu et al., 1992, Coifman and Wickerhauser, 1992] construct optimal wavelets from a predefined set of wavelets. Of particular interest is the work of [Zhuang and Baras, 1994], who propose a method for finding optimal wavelet filter values by gradient descent using an entropy-based cost function. The original work did not demonstrate any experimental results, but they later showed how their method can be used for the task of image compression [Zhuang and Baras, 1996]. However, the implementation was only applied to single images, and the algorithm needed to be initialized to a known wavelet. Furthermore, minimal experimental results are shown so it is difficult to draw any conclusions about the performance of the method.

A similar method for learning orthogonal wavelets based on entropy is proposed in [Nielsen et al., 2006] for the task of signal compression. This work does not use gradient descent, but rather searches over a low dimensional parameterized wavelet space [Zou and Tewfik, 1993]. The idea behind parameterized wavelets is to observe that the necessary filter conditions from Section 2.6 have $k/2 - 1$ degrees of freedom for filters of length k [Burrus et al., 1997]. Thus, we can reduce the dimension of our search space from k to $k/2 - 1$. In the case of $k = 2$, there is exactly one filter that satisfies the conditions (i.e. the Haar scaling function). When $k = 4$, we are left with one degree of freedom. One parameterization is given in [Burrus et al., 1997]:

$$\begin{aligned} h[0] &= \frac{1 - \cos(\alpha) + \sin(\alpha)}{2\sqrt{2}} \\ h[1] &= \frac{1 + \cos(\alpha) + \sin(\alpha)}{2\sqrt{2}} \\ h[2] &= \frac{1 + \cos(\alpha) - \sin(\alpha)}{2\sqrt{2}} \\ h[3] &= \frac{1 - \cos(\alpha) - \sin(\alpha)}{2\sqrt{2}} \end{aligned} \tag{3.2}$$

We can thus learn filters of length four by optimizing over a single variable.

It appears that using wavelet parameterization is ideal, as it halves the search space dimension. However, there are drawbacks to this approach. For example, a parameterization

must be derived for each filter length. Parameterizations up to length six can be found in [Burrus et al., 1997]. Parameterizations for longer filters are more complex to derive [Vaidyanathan and Hoang, 1988]. The complexity of the parameterizations increase significantly as the filter length increases, which may counter the gain from reducing the search space dimension. See Appendix C for examples of filter parameterizations for lengths eight and ten. Filter parameterizations for longer lengths would be impractical to include, but can be generated using software [Selesnick, 1997].

Examples of parameterized optimization methods can be found in [Tewfik et al., 1992, Gopinath et al., 1994, Mandal et al., 1996]. These methods concern themselves with finding the best signal approximations up to a particular wavelet scale. Since they rely on parameterizations, it is unlikely that they can be used to find long wavelet filters. Our method does not have this limitation, as it learns the filter coefficients directly.

3.2.3 Second Generation Wavelets

Second generation (SG) wavelets are a different approach to wavelet filter design. SG wavelets, and more specifically, the lifting scheme method used in the transform are generally accredited to [Sweldens, 1998]. SG wavelets were introduced with the following ideas in mind:

- Simplification of the construction of wavelets by eliminating the need of Fourier methods.
- The consideration of signals that are not defined on the domain of $\mathbb{R}^n$.
- Allowing the analysis of signals that are irregularly sampled.

One consequence of the above considerations is that SG wavelets are no longer translations and dilations of a single mother wavelet function. Secondly, the computation of the SG wavelet transform is done using the lifting scheme which differs from wavelet transform algorithm discussed earlier [Sweldens, 1998].

A notable work in the area of learning second generation wavelets can be found in [Rustamov and Guibas, 2013]. Similar to the work presented here, the goal is to learn wavelets that give sparse representations. An autoencoder framework is used, which is similar (at least at a coarse grain level) to the approach that will be presented in a later chapter. However, there are some notable differences:

- The domain of the signals were over the vertices of graphs, not $\mathbb{R}^n$.
- A quasi-Newton method, L-BFGS, was used instead of gradient descent [Liu and Nocedal, 1989, Ngiam et al., 2011].
- A layerwise training method is used, reminiscent of pre-training stacked autoencoders [Hinton and Salakhutdinov, 2006, Bengio et al., 2007].

Other related works include [Quellec et al., 2008], where optimization of SG wavelets was performed for the purpose of template matching, and [Grasemann and Miikkulainen, 2004], where the lifting scheme is combined with evolutionary algorithms.

3.2.4 Fixed Wavelets in Neural Networks

The idea of combining wavelet analysis and neural networks is not new. However, past approaches generally make use of fixed (not learned) wavelets. Fixed wavelet representations have been used in neural network architectures for a variety of tasks including power disturbance recognition [Gaing, 2004], seizure prediction [Petrosian et al., 2000, Ghosh-Dastidar et al., 2007], freeway incident detection [Ghosh-Dastidar and Adeli, 2003], and other time series prediciton [Wang et al., 2001].

An early proposed variant of neural networks is the wavelet network [Zhang and Benveniste, 1992]. In the one-dimensional case, the wavelet network computes [Zhang, 1997]:

$$f_n(x) = \sum_{i=1}^{n} u_i \psi(a_i(x - t_i)) \tag{3.3}$$

for $u_i, a_i, t_i \in \mathbb{R}$. The wavelet function, ψ, is fixed and the learned parameters of the network are u_i, a_i, and t_i which control the time-frequency tiling. Note that no scaling function is present in Equation 3.3, and so it is not a multiresolution decomposition [Billings and Wei, 2005].

A notable model that uses a fixed wavelet representation is the scattering transform [Mallat, 2012]. The scattering transform is a translation invariant transform that is Lipschitz-continuous. Lipschitz-continuous means (approximately) that a small deformation of the input results in a small deformation of the output. Practically, the scattering transform is computed using a cascade of complex wavelets and absolute value operators. A "node" at layer k computes

$$\big|\,|\,|x * \psi_1| * \psi_2|\cdots| * \psi_k|\,\big| \tag{3.4}$$

for input signal x. In order to introduce invariance to small perturbations, the output of each wavelet cascade is convolved with a scaling function:

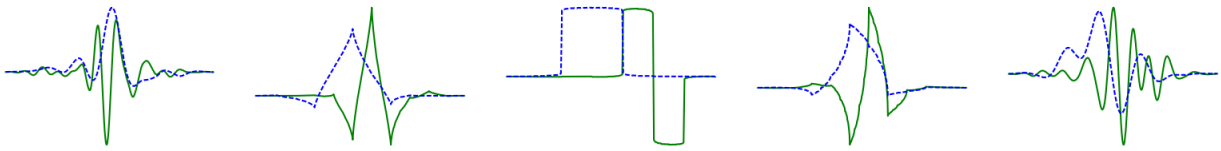
$$\big|\,|\,|x * \psi_1| * \psi_2|\cdots| * \psi_k|\,\big| * \phi_k \tag{3.5}$$

The transform can thus be viewed as a type CNN. The wavelet filters in the scattering transform are not learned, however, and are also complex. Output from each convolutional layer is kept, unlike typical CNNs which only consider the output of the final layer.

The development of the scattering transform was partially motivated by the many theoretical questions surrounding the training of CNNs [Bruna and Mallat, 2013]. Like the scattering transform, a CNN applies a cascade of filters and pooling. Furthermore, it must learn a feature representation that is invariant to irrelevant variations in the input signal [LeCun et al., 2010].

Chapter 4

Learning Wavelets for One-Dimensional Signals



The wavelet transform has several useful properties that make it a good choice for a feature representation including: a linear time algorithm, perfect reconstruction, and the ability to tailor wavelet functions to the application. However, the wavelet transform is not widely used in the machine learning community. Instead, methods like the Fourier transform and its variants are often used instead (e.g. [Graves et al., 2013]). We believe that one cause of the lack of use is the difficulty in designing and selecting appropriate wavelet functions. Wavelet filters are typically derived analytically using Fourier methods. Furthermore, there are many different wavelet functions to choose from. Without a deep understanding of wavelet theory, it can be difficult to know which wavelet to choose. This difficulty may lead many to stick to simpler methods.

We propose a method that learns wavelet functions directly from data using a neural network framework. As such, we can leverage the theoretical properties of the wavelet transform without the difficult task of designing or choosing a wavelet function. An advantage of this method is that we are able to learn directly from raw audio data. Learning from raw audio has shown recent success in audio generation [Oord et al., 2016].

We are not the first to propose using wavelets in neural network architectures. There has been previous work in using fixed wavelet filters in neural networks such as the wavelet network [Zhang and Benveniste, 1992] and the scattering transform [Mallat, 2012]. Unlike our proposed method, these works do not learn wavelet functions from data. See Chapter 3 for more discussion of related work

We begin our discussion with the wavelet transform. We will provide some mathematical background as well as outline the discrete wavelet transform algorithm. Next, we outline our proposed wavelet transform model. We show that we can represent the wavelet transform as a modified convolutional neural network. We then evaluate our model by demonstrating that we can learn useful wavelet functions using an architecture similar to traditional autoencoders [Hinton and Salakhutdinov, 2006].

4.1 The Wavelet Transform

The wavelet transform makes use of a dictionary of wavelet functions that are dilated and shifted versions of a mother wavelet, ψ. The wavelet functions can be thought of as a bandpass filter bank. The wavelet transform is then a decomposition of a signal, x, with this filter bank.

The discrete wavelet transform and its inverse can be computed via a fast decimating algorithm. Recall in Section 2.4 that we defined two filters

$$h[n] = \left\langle \frac{1}{\sqrt{2}}\phi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle \tag{4.1}$$

$$g[n] = \left\langle \frac{1}{\sqrt{2}}\psi\left(\frac{t}{2}\right), \phi(t-n) \right\rangle \tag{4.2}$$

With these two filters, an iterative algorithm for computing the wavelet coefficients can be derived. The iterative steps are included below:

$$a_{j+1}[p] = \sum_{n=-\infty}^{+\infty} h[n-2p]a_j[n] \tag{4.3}$$

$$d_{j+1}[p] = \sum_{n=-\infty}^{+\infty} g[n-2p]a_j[n] \tag{4.4}$$

Both the forward (decomposition) and inverse (reconstruction) algorithms are computed through repeated convolution and downsampling/upsampling. In other words, the algorithms compute a cascade of convolutions, with each iteration using the output of the previous iteration. This type of iterative computation is very similar to the computation done within neural networks. A neural network also performs an iterative computation, with each layer making use of the output of the previous layer. Convolutional neural networks (CNN) in particular, perform a cascade of convolutions. The similarity between the discrete wavelet transform algorithm and CNNs is important, as it allows us to view the wavelet transform as a specialized neural network.

4.2 Proposed Model

In this section we will frame the discrete wavelet transform algorithm in terms of a neural network. In particular, we propose a method for learning wavelet functions by defining the discrete wavelet transform as a modified CNN architecture. CNNs compute a feature representation of an input signal through a cascade of filters. They have seen success in many signal processing tasks, such as speech recognition and music classification [Sainath et al., 2013, Choi et al., 2017]. Generally, CNNs are not applied directly to raw audio data. Instead, a transform is first applied to the signal (such as the windowed Fourier transform). This representation is then used as the input of the network.

Our proposed method works directly on the raw audio signals. We accomplish this by implementing the discrete wavelet transform as a modified CNN. Figure 4.1 shows a graphical representation of our model, which consists of repeated applications of Equations 4.3 and 4.4. The parameters (or weights) of this network are the wavelet and scaling filters g and h. Thus, the network computes the wavelet coefficients of a signal, but allows the wavelet filter to be learned from the data. We can similarly define an inverse network using Equation 2.38.

We can view our network as an unrolling of the discrete wavelet transform algorithm similar to unrolling a recurrent neural network (RNN) [Pascanu et al., 2013]. Unlike an RNN, our model takes as input the entire input signal and reduces the scale at every layer through downsampling. Each layer of the network corresponds to one iteration of the algorithm. At each layer, the detail coefficients are passed directly to the final layer. The final layer output, denoted $W(x)$, is formed as a concatenation of all the computed detail coefficients and the final approximation coefficients. We propose that this network be used as an initial module as part of a larger neural network architecture. This would

allow a neural network architecture to take as input raw audio data, as opposed to some transformed version.

4.2.1 Comparison to CNNs

In this section we will explain our model in terms of neural networks. In particular, we will discuss how we can view our model as a restricted linear CNN. It is important to note that since our model is performing a linear transform, it is equivalent to a single layer CNN. However, we believe a comparison to a multilayer CNN is a useful exercise. See Appendix A for a background on neural networks.

It is important to discuss the differences between traditional CNNs and our proposed wavelet transform network. A CNN is parameterized by a set of filters (i.e. a filter bank) at each layer of the network. In general, there are no constraints placed on the filter parameters. Furthermore, the filters do not share weights across layers. In contrast, our wavelet transform network is parameterized by a single filter pair. The filter pair is reused at each layer of the network. More precisely, each layer of the network computes the approximation and detail coefficients as in Equations 4.3 and 4.4. The approximations coefficients, a_j, are passed on to the next layer. The detail coefficients, d_j, are passed directly to the final layer. One may view this similarly to skip connections used in recent neural network architectures [Ronneberger et al., 2015, Long et al., 2015]. See Figure 4.2 for an illustration of a single layer of the wavelet network.

The output of the wavelet network is thus a concatenation of the detail coefficients at every scale, and the final approximation coefficients. Since the network performs down-sampling by a factor of two at each layer, the maximum depth of the network is on the order of $\log_2(N)$ (for a signal of length N). In practice, we can use any depth that is less than or equal to the maximum depth. CNNs, on the other hand, do not generally have a maximum depth. Of course, computational restrictions may limit the depth of a CNN. See Appendix A for more discussion.

Filters learned in a CNN are generally unconstrained (other than having a fixed size). Any real vector is a valid filter. Our wavelet network will make use of filter constraints designed so that the learned filters meet the wavelet properties. Otherwise, the wavelet network will not compute a valid wavelet transform. The constraints we consider are: finite energy, finite L^1 norm, and zero mean wavelet filters (see Section 2.6).

$$\sum_k h[k]^2 = 1 \tag{4.5}$$

$$\sum_k h[k] = \sqrt{2} \tag{4.6}$$

$$\sum_k g[k] = 0 \tag{4.7}$$

Finally, we restrict ourselves to quadrature mirror filters. That is, we set

$$g[n] = (-1)^n h[-n] \tag{4.8}$$

By making this restriction, we reduce our parameters to only the scaling filter h.

The model parameters will be learned by gradient descent. As such, we must modify the "hard" constraints in Equations 4.5-4.7 to "soft" differentiable penalties. We define the following soft quadratic wavelet penalties:

$$\sum_k h[k]^2 = 1 \;\Rightarrow\; \left(\sum_k h[k]^2 - 1\right)^2 \tag{4.9}$$

$$\sum_k h[k] = \sqrt{2} \;\Rightarrow\; \left(\sum_k h[k] - \sqrt{2}\right)^2 \tag{4.10}$$

$$\sum_k g[k] = 0 \;\Rightarrow\; \left(\sum_k g[k]\right)^2 \tag{4.11}$$

If these penalties are equal to zero, then Equations 4.5-4.7 are satisfied exactly.

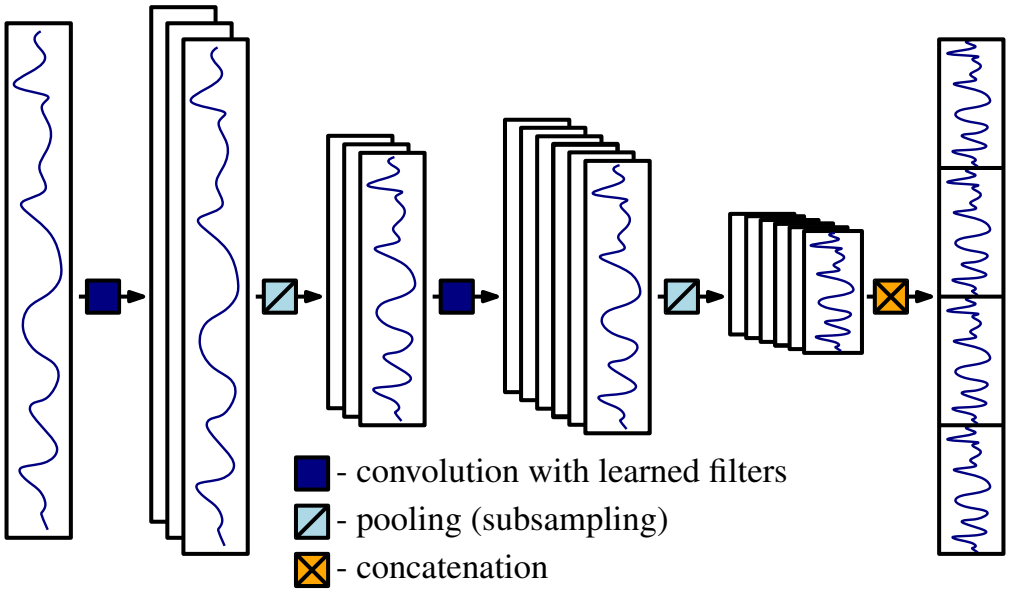
Equations 4.9-4.11 are summarized with the following loss term

$$L_w(h, g) = \left(\sum_k h[k]^2 - 1\right)^2 + \left(\sum_k h[k] - \sqrt{2}\right)^2 + \left(\sum_k g[k]\right)^2 \tag{4.12}$$

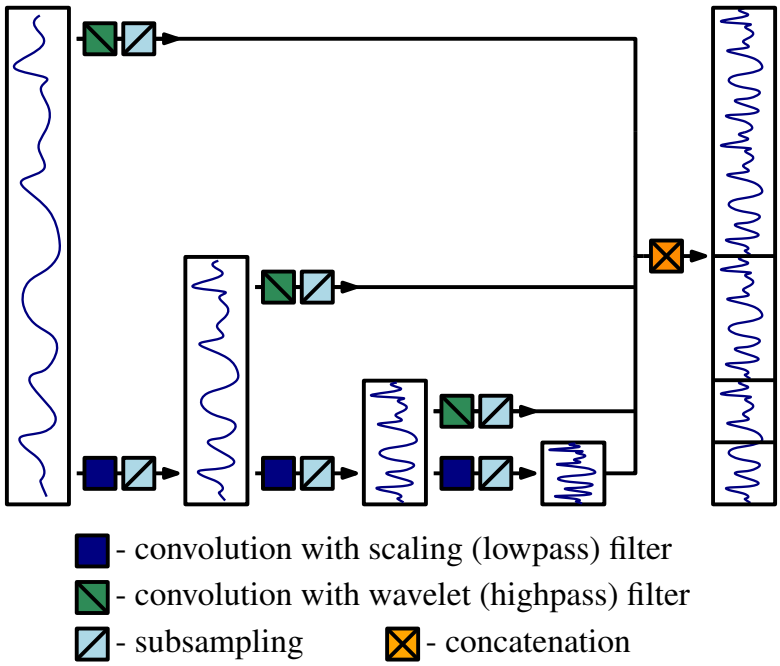
or equivalently

$$L_w(h, g) = (||h||_2^2 - 1)^2 + (\mu_h - \sqrt{2}/k)^2 + \mu_g^2 \tag{4.13}$$

where μ_h and μ_g are the filter means. The wavelet penalty, L_w, is not sufficient for learning wavelet functions. We can demonstrate this by randomly sampling filters that minimize L_w. See Figure 4.3 for examples of randomly chosen wavelet functions derived from filters that minimize Equation 6.10. These filters have no apparent structure. We have not explored the connection between the space of wavelets that minimize Equation 6.10 and those of parameterized wavelet families [Burrus et al., 1997].



(a)



(b)

Figure 4.1: (a) A standard 1D CNN architecture. (b) Our wavelet transform CNN architecture.

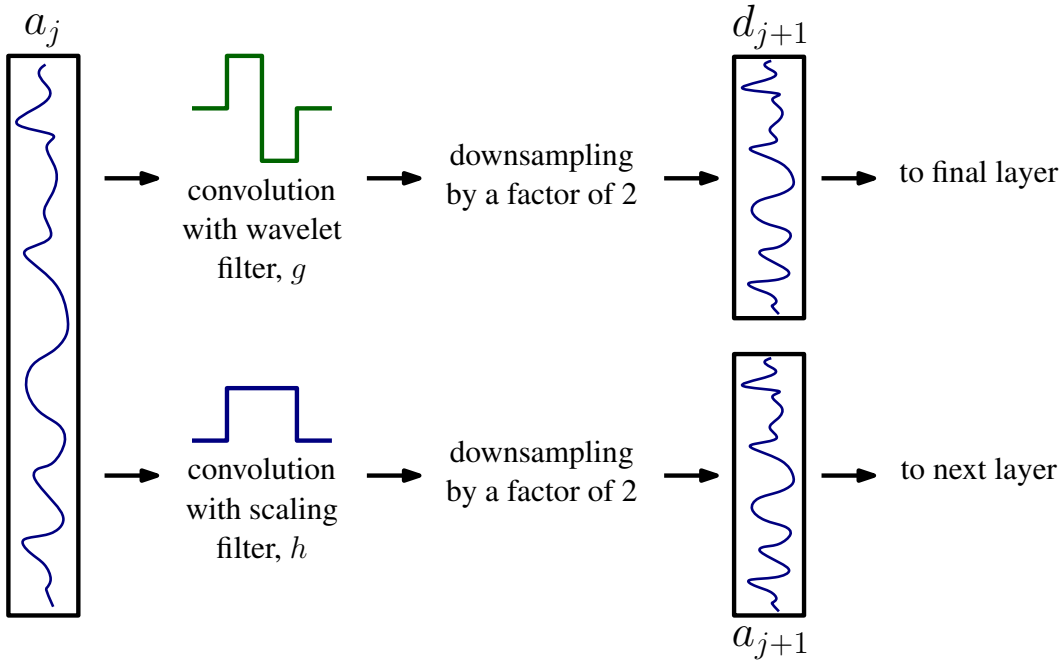


Figure 4.2: A single layer of our wavelet network.

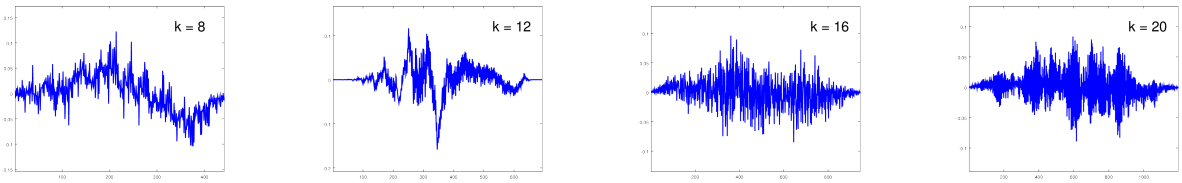


Figure 4.3: Examples of random wavelet functions that satisfy Equation 6.10 for different filter lengths.

4.3 Learning Wavelets by Sparse Representations

One way of learning wavelets is by learning sparse representations. That is, we wish to learn a wavelet that gives rise to a sparse code for each of our data samples. A common architecture developed for learning codes is the autoencoder. Autoencoders are used in unsupervised learning in order to learn useful data representations [Hinton and Salakhutdinov, 2006]. An autoencoder is composed of two neural network components: and encoder and decoder. The encoder acts as a function that maps input data to a code, while the decoder reconstructs the data from the code.

At first glance, such a model does not seem very useful as it merely computes the identity function. However, by constraining the intermediate representation the network can learn useful structure. For example, it is common to restrict the size of the code to have smaller dimension than that of the input. This forces the autoencoder to perform dimensionality reduction. We will note here that there is a correspondence between such dimensionality reduction and that of PCA. Consider a linear, single layer autoencoder with a code dimension of k trained with MSE reconstruction loss. It can be shown that the k components learned by the network will span the same space as the first k principal components [Bourlard and Kamp, 1988, Bengio, 2009]. Of course, this result does not tell us anything about other types of autoencoders (e.g. multilayer, non-linear architectures), but it does provide some connection between matrix factorization methods and neural networks.

Another approach (and the one we will make use of) is constraining the code to be sparse. Such an approach shares similarities to the problem of sparse coding. Unlike traditional sparse coding, an autoencoder learns an explicit mapping from the data to the code. Furthermore, the reconstruction is not constrained to be a linear combination of dictionary elements.

Our proposed model is an autoencoder as illustrated in Figure 4.4. It is composed of a wavelet transform network followed by an inverse wavelet transform network. The loss function is made up of a reconstruction loss, a sparsity term, and the wavelet constraints. Let $\hat{x}_i$ denote the reconstructed signal. The loss function is defined as

$$L(X; h, g) = \frac{1}{MN} \sum_{i=1}^{M} ||x_i - \hat{x}_i||_2^2 + \lambda_1 \frac{1}{MN} \sum_{i=1}^{M} ||W(x_i)||_1 + \lambda_2 L_w(h, g) \tag{4.14}$$

where $W(x_i)$ are the wavelet coefficients of x_i, and $X = \{x_1, x_2, \ldots, x_M\}$ is a dataset of signals of length N. In our experiments, we fix $\lambda_1 = \lambda_2 = 1/2$.

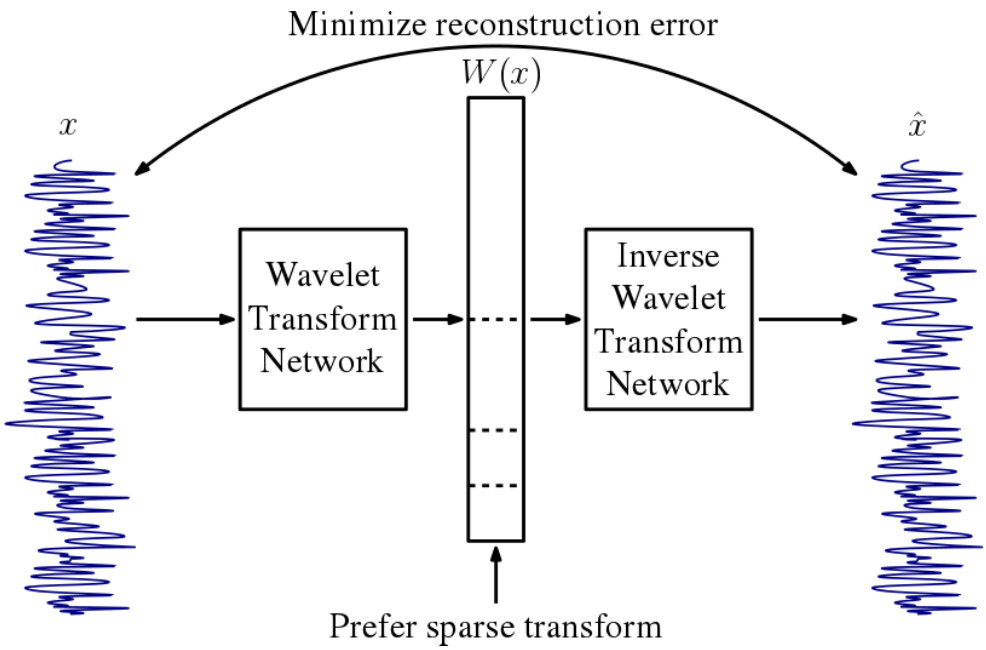


Figure 4.4: The reconstruction network is composed of a wavelet transform followed by an inverse wavelet transform.

A similar method for learning wavelet filters can be found in [Søgaard, 2017] (unpublished). A neural network model (derived independently from ours) is used to learn wavelet filters. The loss function used in the model has two components: a wavelet constraint term (similar to Equation 4.14), and a sparsity penalty based on the Gini index [Gini, 1912, Gini, 1921]. Without a reconstruction term, the model performs only signal analysis. Furthermore, the learned filters may not be valid wavelets because the wavelet constraint term relies on necessary, but not sufficient, conditions (see Section 2.6). The model is implemented using a fully connected neural network, as opposed to the CNN-based architecture in our model. As a result, a full matrix is constructed (as in Equation A.11) and multiplied by the signal at each layer. This computation is less efficient than directly performing a convolution, and becomes intractable for long signals.

4.3.1 Sparsity

In our model we choose to use the ℓ_1 norm as our sparsity constraint. This choice is important, as it will influence the behaviour of our model. Since we will be training our model by gradient descent, we require a sparsity function that is differentiable. This

eliminates the ℓ_0 norm commonly used in sparse coding. The ℓ_1 norm also sufferes from this problem since it is not differentiable at zero. However, we can still make use of gradient descent as long as we take some liberties when computing the gradient. Consider the ℓ_1 norm of some vector x:

$$||x||_1 = \sum_i |x_i| \tag{4.15}$$

The partial derivative with respect to some x_i is thus

$$\frac{\partial ||x||_1}{\partial x_i} = \begin{cases} 1 & x_i > 0 \\ -1 & x_i < 0 \\ \text{undefined} & x_i = 0 \end{cases} \tag{4.16}$$

We can fix this simply by defining the derivative at zero to be a reasonable value (such as zero):

$$\frac{\partial ||x||_1}{\partial x_i} = \begin{cases} 1 & x_i > 0 \\ -1 & x_i < 0 \\ 0 & x_i = 0 \end{cases} \tag{4.17}$$

Though this is may be mathematically suspect, it allows us to make use of certain continuous, non-differentiable loss functions with gradient descent.

This gives us the ability to use the ℓ_1 norm, but it does not provide any justification for using it. One theoretical justification is that ℓ_1 can be shown to give sparse solutions to many linear systems [Donoho, 2006]. But what about other norms? Consider the squared ℓ_2 norm

$$||x||_2^2 = \sum_i x_i^2 \tag{4.18}$$

At first this may seem like a reasonable sparsity penalty to apply. However, there is a major problem with this particular norm. The wavelets considered in this thesis are orthogonal. As such, the wavelet transform preserves energy. In other words the energy of the input signal is equal to the energy of the wavelet representation. Therefore, the ℓ_2 norm of the wavelet coefficients are constant for any orthogonal wavelet. This would make such a term in the loss function useless.

Disregarding the energy preservation problem, there is another property of the ℓ_2 norm that makes it unsuitable as a sparsity term. The goal of the sparsity cost is to prefer representations with many zero elements. Let us consider the behaviour of the derivative of a single element, x_i, of the representation:

$$\frac{\partial ||x||_2^2}{\partial x_i} = x_i \tag{4.19}$$

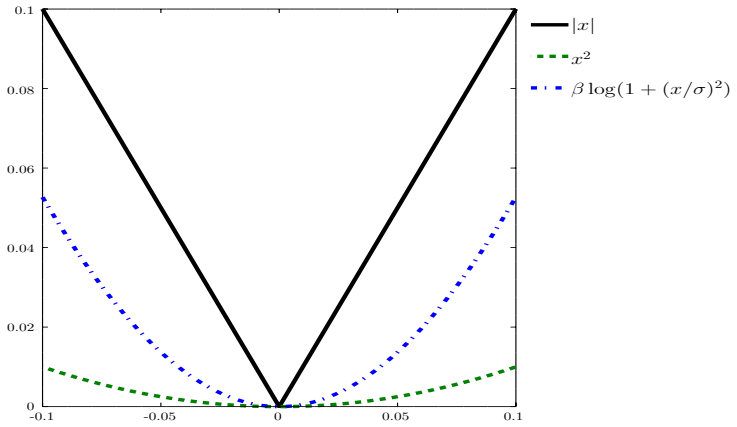


Figure 4.5: Comparison of three different sparsity penalties for small values of x.

As x_i approaches zero, so too does its derivative. Thus, elements close to zero do not contribute much to the overall gradient. Hence, when a step is taken in gradient descent, these elements will not be pushed exactly to zero. This problem occurs with any sparsity penalty whose derivative smoothly approaches zero as elements approach zero. For example, the sparsity penalty used by [Olshausen et al., 2001], $\beta \log(1 + (x/\sigma)^2)$, has a similar problem[1]. The ℓ_1 norm, on the other hand, has a constant derivative as values approach zero. Using gradient descent with ℓ_1 tends to push elements exactly to zero. Figure 4.5 illustrates different sparsity penalties. Intuitively, we can conclude that functions whose derivative smoothly approaches zero for elements near zero will be unsuitable for a sparsity penalty.

4.4 Evaluation

In this section we will evaluate our proposed model with a variety of experiments. We first show that given data that fits the model assumptions, our learning method will recover the correct wavelet. This is true even when the length of the original filter is unknown. We also demonstrate that the learned wavelets are able to capture the structure of the training data. We implemented our model using Google's Tensorflow library [Abadi et al., 2015] and make use of the Adam algorithm for stochastic gradient descent [Kingma and

[1]This issue was later addressed by using a mixture of a Gaussian and a Dirac delta function [Sallee and Olshausen, 2003].

Ba, 2014]. In all our experiments the signals are of length 1024 and were scaled to be in the range of $[-1, 1]$.

4.4.1 Model Verification

Our first set of experiments are meant to verify the validity of our wavelet model. We will generate data by first randomly sampling sparse wavelet coefficients, followed by performing an inverse wavelet transform with a fixed wavelet. Each wavelet coefficient has a 90% chance of being equal to zero, and is otherwise drawn randomly from $[-1, 1]$. We restrict ourselves to three families of orthogonal wavelets: Daubechies, Symlets, and Coiflets. The model should recover wavelets similar to the wavelet that was used to generate the data. In order to compare the the similarity between wavelets, we will first define a distance measure between filters of length k:

$$dist(h_1, h_2) = \min_{0 \leq i < k} 1 - \frac{\langle h_1, \text{shift}(h_2, i) \rangle}{||h_1||_2 \cdot ||h_2||_2} \tag{4.20}$$

where $\text{shift}(h_2, i)$ is h_2 circular shifted by i samples. This measure is the minimum cosine distance under all circular shifts of the filters. To compare different length filters, we zero-pad the shorter filter to the length of the longer.

In the first experiments we set the learned filter length, k, equal to the length of the wavelet used to generate the data. Figures 4.6-4.8 show a comparison of the learned scaling filter to the generative scaling filter. Above each plot is the distance between the filters. In each case the model was able to recover a filter very close to the filter used to generate the data. In most cases, there is no qualitative difference between the learned filter and the original filter.

In the previous experiments, we chose k to be equal to the length of the filter used to generate the data. For data in general, we will not know the true value of k. Hence, in our next experiments we fix the generative wavelet to be Daubechies 2 (length four) and vary the length of the learned filter. If our model is robust to the choice of k, then we should be able to recover the generative filter as long as $k \geq 4$. Figure 4.9 shows the learned filter values for $k \in \{6, 8, 10, 20\}$. In each case the four nonzero Daubechies 2 scaling filter values are recovered, while the other values tended to zero. This result is interesting as we put no constraints (such as sparsity) on the filter values. Yet, the model was able to recover exactly four nonzero scaling filter coefficients.

Finally, we test the effect of noise on our model. We repeat our previous experiments, but add varying levels of Gaussian noise to the input signals. Table 4.1 displays the distance between the learned wavelet and the generative wavelet for a selection of traditional

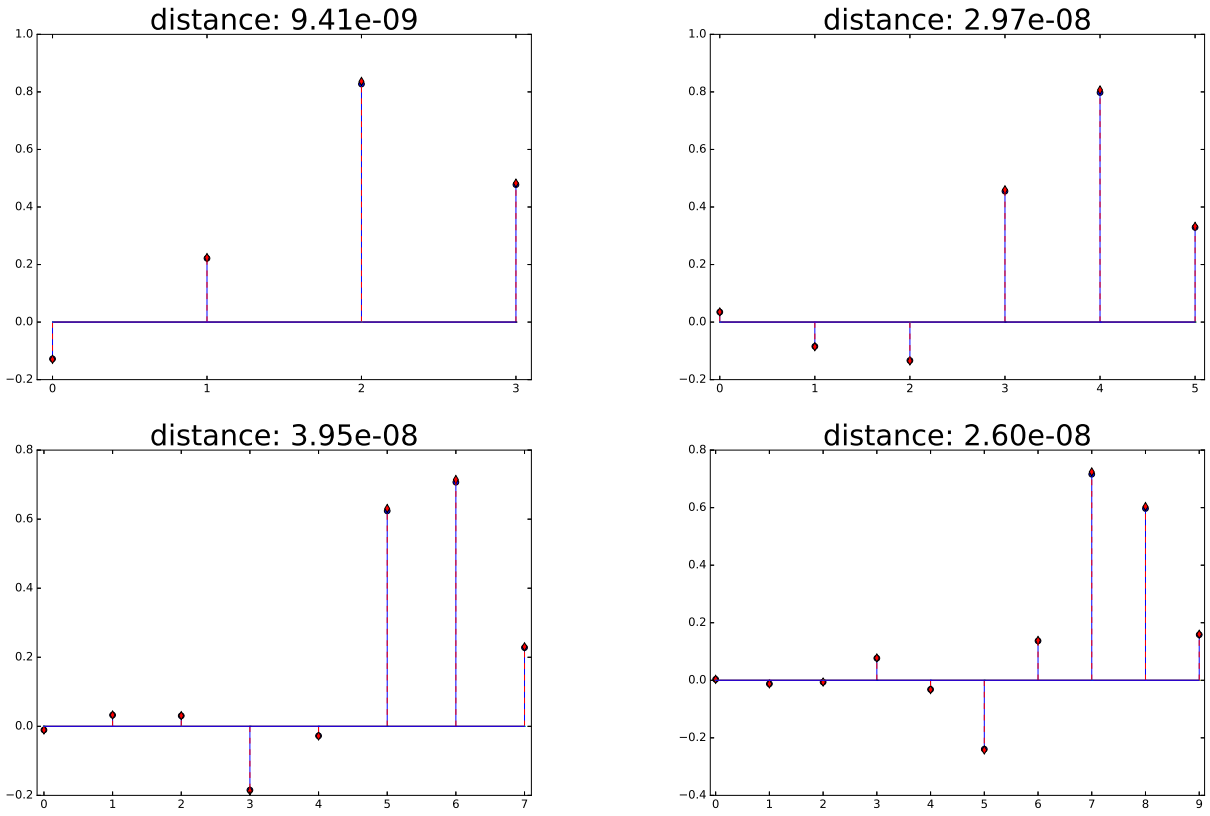


Figure 4.6: Learned scaling filters (blue circles) compared to Daubechies scaling filters (red diamonds).

wavelets and noise levels. In each trial the learned filter had length 20. Increasing the noise generally led to increased error between the learned wavelet and the generative wavelet. This was most likely caused by the model attempting to fit the noise along with the signal. However, in all cases the model was able to recover filters close to the generative filter (with a maximum error distance of 5.58e-3).

4.4.2 Structure Learning

Our second set of experiments are meant to test whether the model is able to learn wavelets whose structure is tuned to that of the data. We make use of synthetic and real harmonic data. The real data consists of segments of piano music taken from the MIDI aligned

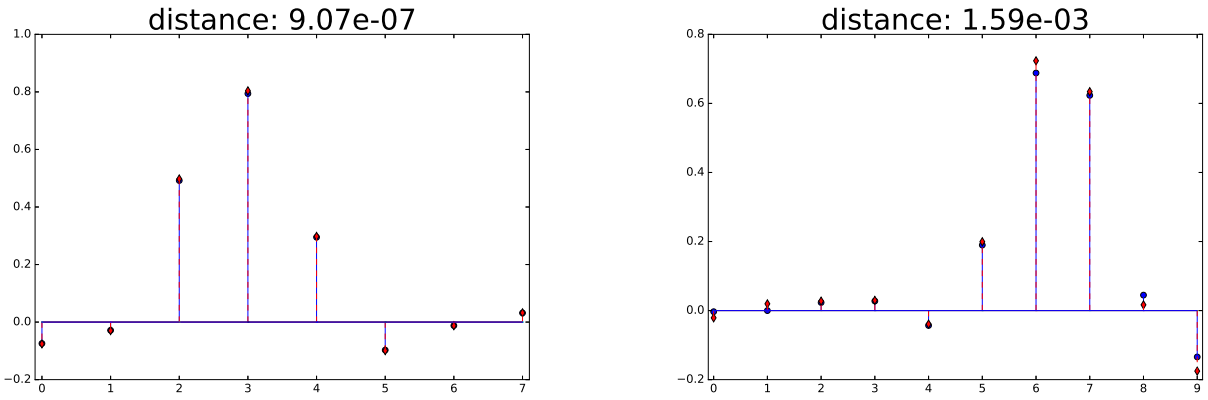


Figure 4.7: Learned scaling filters (blue circles) compared to Symlet scaling filters (red diamonds).

Noise σ	Daubechies 3 ($k = 6$)	Symlet 5 ($k = 10$)	Coiflet 2 ($k = 12$)
0.00	3.10e-05	3.09e-05	2.61e-06
0.02	6.47e-05	6.80e-05	6.56e-05
0.04	1.27e-04	6.84e-4	5.00e-4
0.06	8.68e-04	1.32e-3	2.24e-4
0.08	9.40e-04	3.23e-4	3.04e-4
0.10	1.02e-03	3.99e-4	5.58e-3

Table 4.1: Distance between learned filters and generative filters for various levels of Gaussian noise. The length of the learned filters was 20 in all trials. The generative filter lengths are indicated in brackets.

piano dataset (MAPS) [Emiya et al., 2010]. The synthetic data consists of harmonic data generated from simple periodic waves. We construct a synthetic signal, x_i, from a base periodic wave function, s, as follows:

$$x_i(t) = \sum_{k=0}^{K-1} a_k \cdot s(2^k t + \phi_k) \tag{4.21}$$

where $\phi_k \in [0, 2\pi]$ is a phase offset chosen uniformly at random, and $a_k \in \{0, 1\}$ is the k^{th} harmonic indicator which takes the value of 1 with probability 1/2. We considered three different base waves: sine, sawtooth, and square. Example synthetic signals are shown in Figure 4.10. A second type of synthetic signal was created similarly to Equation 6.14 by

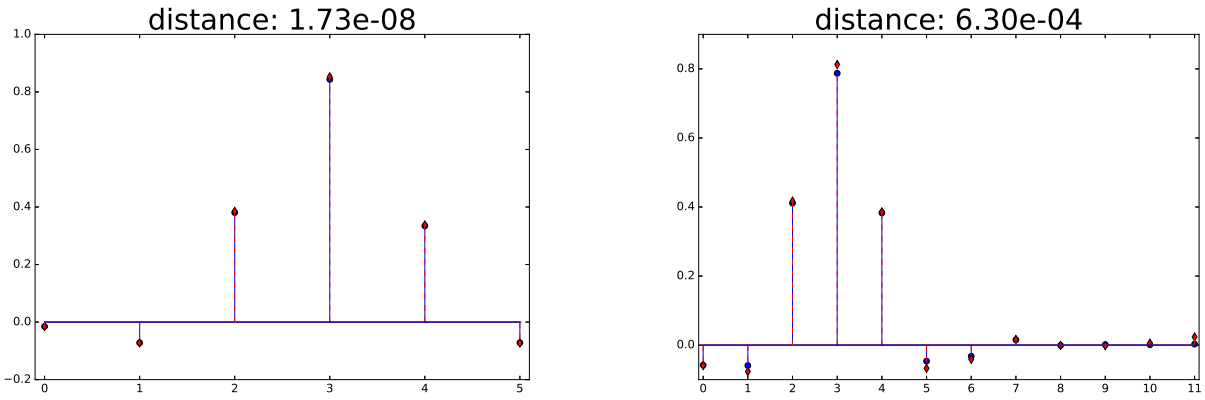


Figure 4.8: Learned scaling filters (blue circles) compared to Coiflet scaling filters (red diamonds).

windowing the base wave at each scale with randomly centered Gaussian windows (multiple windows at each scale were allowed). The length of the learned filters is 20 in all trials. An overview of the learning method is shown in Figure 4.11.

The learned wavelet filters are unique to the type of data that is being reconstructed. Example wavelet functions are included in Figure 4.12 (left column). These functions are computed from the scaling filter coefficients using the cascade algorithm [Strang and Nguyen, 1996]. Note that the learned functions are highly structured, unlike the random wavelet functions in Figure 4.3. Furthermore, each learned wavelet appears to have structure similar to the training data. For example, the wavelet learned from sawtooth data is triangular, and the wavelet learned from sine waves is similar to a windowed sinusoid. The music wavelet is also sinusoidal, which may correspond to the harmonic structure of the music data. Also included in Figure 4.12 are the closest traditional wavelets according to Equation 5.12 (middle column) and the corresponding filter values (right column). We restrict our consideration to the following traditional wavelet families: Haar, Daubechies, Symlets, and Coiflets. Note that the learned wavelets share similar structure to certain traditional wavelets.

One interesting property of the learned wavelets is that they are all asymmetric, with the exception of the square wavelet. In fact, this property is true for all compactly supported orthogonal wavelets other than the Haar wavelet [Daubechies, 1992]. Symmetric filters, however, can be obtained by either relaxing the orthogonality constraint, or by allowing complex valued filters.

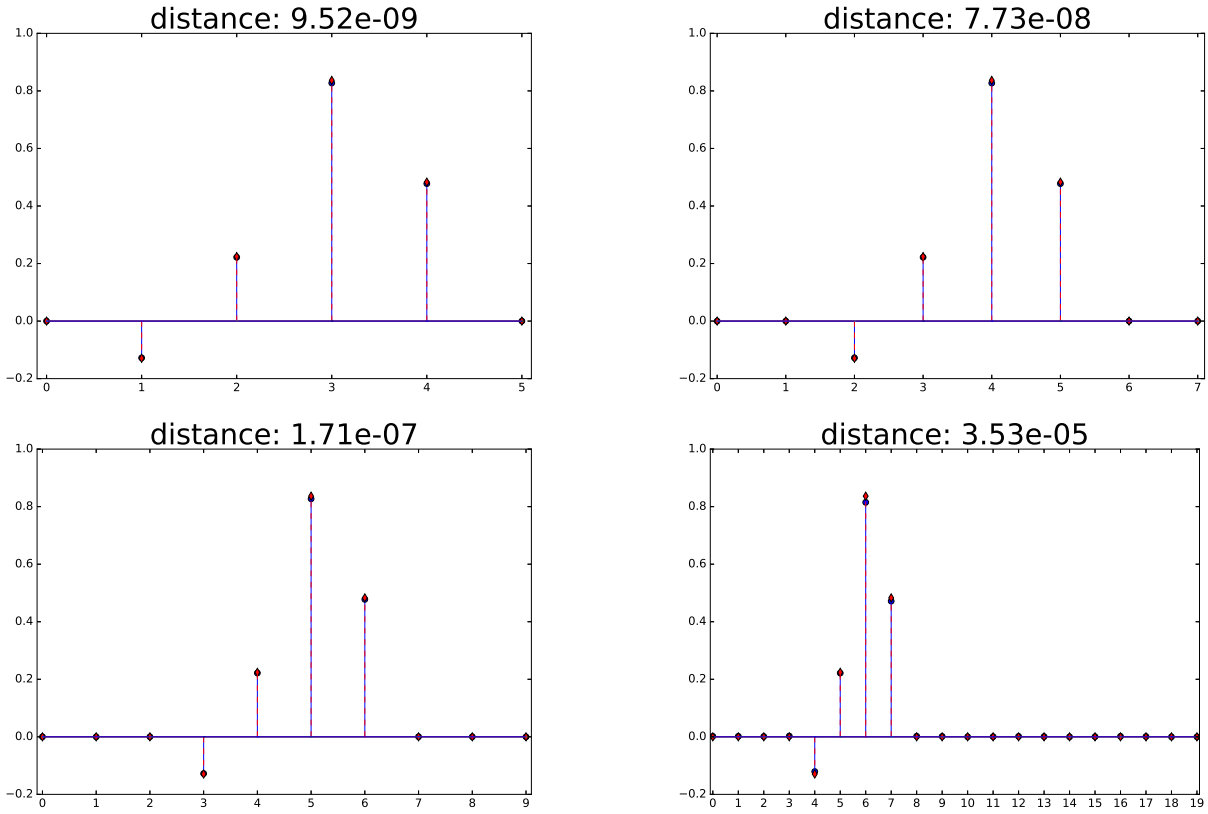


Figure 4.9: Learned scaling filters (blue circles) compared to the Daubechies 2 scaling filter (red diamonds) for various values of k.

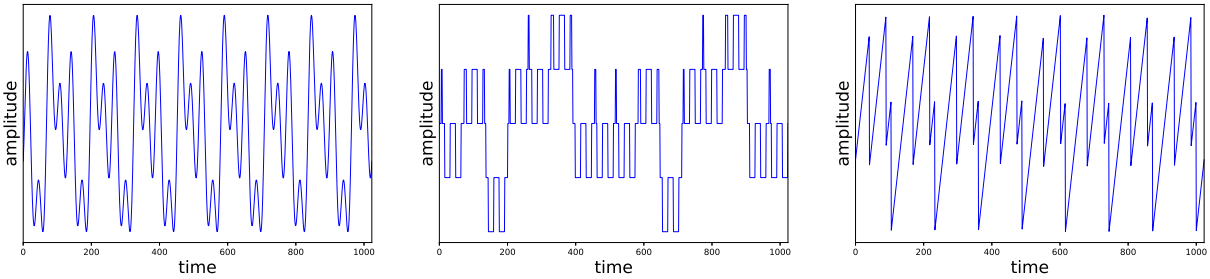


Figure 4.10: Examples synthetic data generated from Equation 6.14.

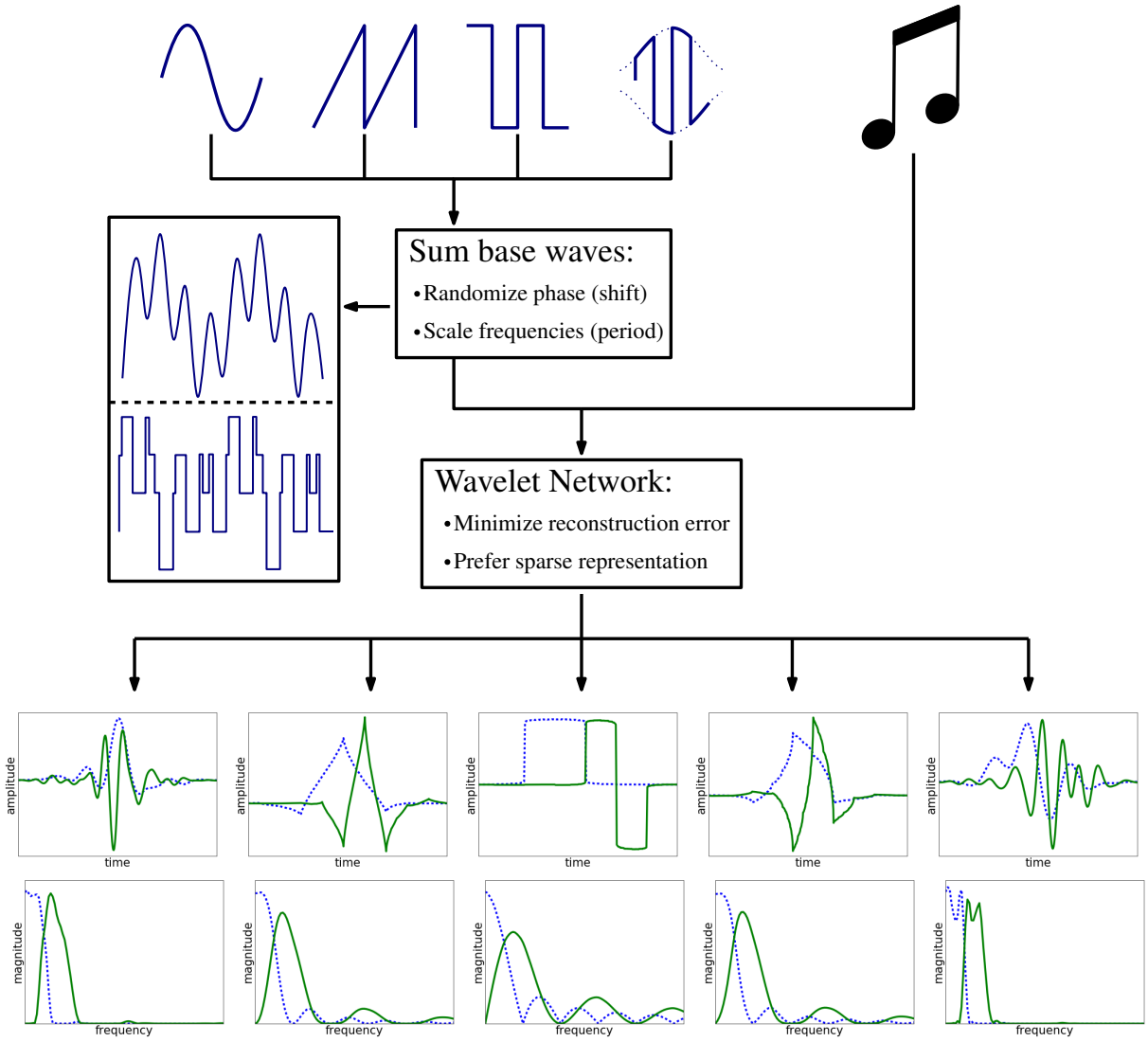


Figure 4.11: Overview of the wavelet learning process. Different input waves result in the different types of wavelets shown in the bottom rows (top: wavelet (solid) and scaling (dashed) functions, bottom: Fourier spectrum of the functions). The input data from left to right: sine waves, sawtooth waves, square waves, windowed square waves, piano music.

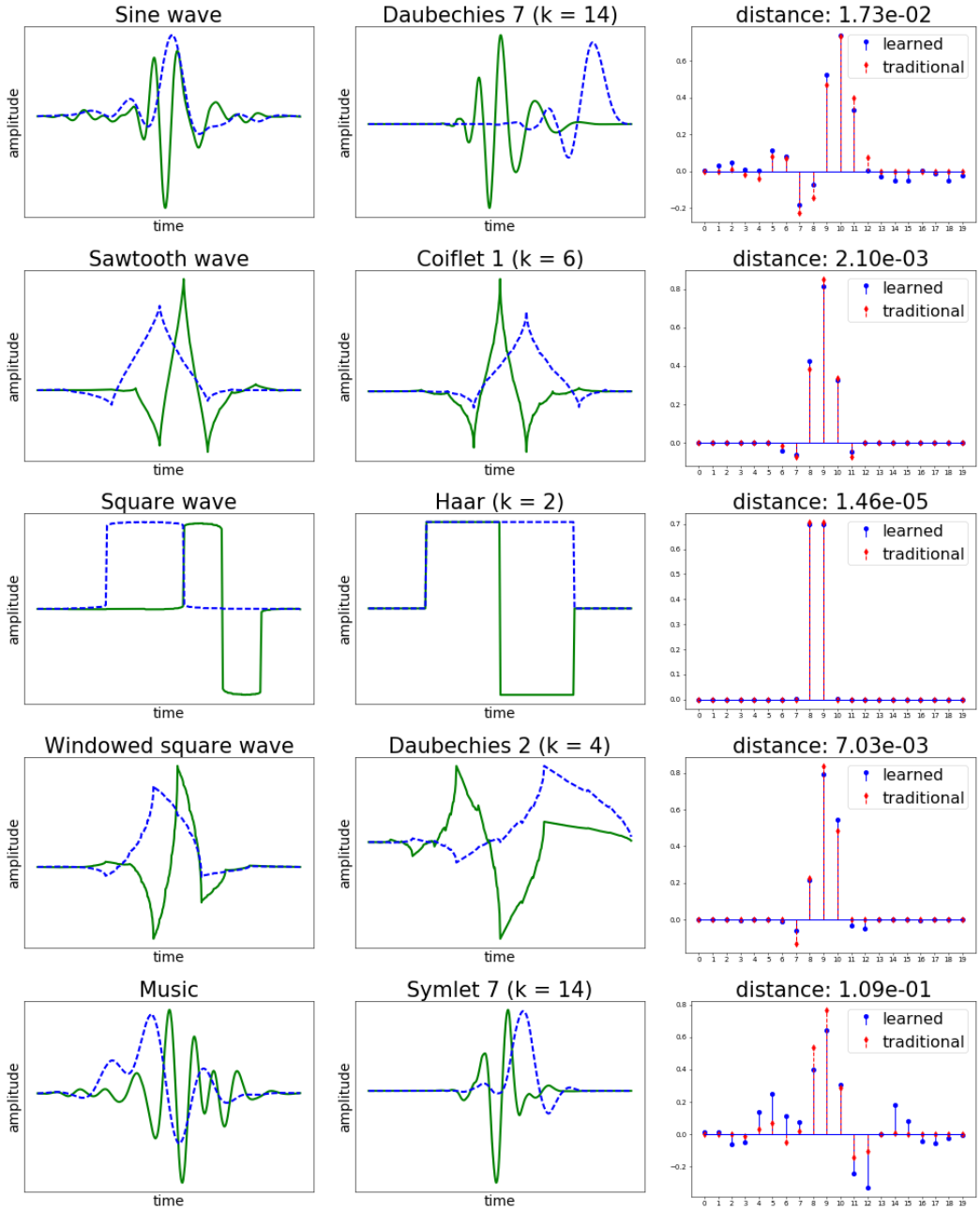


Figure 4.12: Left column: Learned wavelet (solid) and scaling (dashed) functions. Middle column: Closest traditional wavelet (solid) and scaling (dashed) functions. Right column: Plots of the scaling filters from the first two columns with corresponding distance measure.

4.5 Probabilistic Interpretation

Up to now, we have defined our learning method in terms of minimizing a cost (or energy) function. In this section we show that this minimization is equivalent to maximizing a particular probability. In doing so we will gain an alternative view of our model that may provide further intuition to the reader. Suppose we have a signal x and a model parameterized by θ. We wish to find the parameters that are most probable given the data,

$$\begin{aligned} \theta^* &= \arg\max_{\theta} P(\theta \,|\, x) \\ &= \arg\max_{\theta} P(x \,|\, \theta) P(\theta) \end{aligned} \tag{4.22}$$

where $P(x \,|\, \theta)$ is the likelihood of our data given our model parameters. In other words, $P(x \,|\, \theta)$ is the probability that the signal would be generated by the model given a particular setting of the parameters. Let us introduce a latent variable a, representing the wavelet coefficients that generated the signal x. We then can rewrite $P(x \,|\, \theta)$ as

$$P(x \,|\, \theta) = \int P(x \,|\, a, \theta) P(a)\, da \tag{4.23}$$

It is important to note that Equation 4.23 makes the assumption that $P(a \,|\, \theta) = P(a)$ (i.e. the code is independent of the model parameters).

We must now define the probability distributions in Equations 4.22 and 4.23. We begin by setting

$$P(x \,|\, a, \theta) = \frac{1}{Z_r} e^{-||x - Da||_2^2} \tag{4.24}$$

and

$$P(a) = \frac{1}{Z_s} e^{-\lambda_1 ||a||_1} \tag{4.25}$$

where D is the orthogonal matrix comprised of our synthesis wavelet basis (i.e. Da computes the inverse wavelet transform), and Z_r and Z_s are normalization constants. Equations 4.24 and 4.25 are equivalent to assuming Gaussian noise on the reconstructed signal and a Laplacian prior over the wavelet coefficients (see Section 4.5.2). Finally, we set

$$\begin{aligned} P(\theta) &= \frac{1}{Z_\theta} e^{-\lambda_2 L_w(h,g)} \\ &= \frac{1}{Z_\theta} e^{-\lambda_2 \left(||h||_2^2 - 1\right)^2} e^{-\lambda_2 \left(\mu_h - \sqrt{2}\right)^2} e^{-\lambda_2 \mu_g^2} \end{aligned} \tag{4.26}$$

In other words, $P(\theta)$ is the product of three independent Gaussians (each corresponding to one of the terms in $L_w(h, g)$).

One problem remains, Equation 4.22 requires the evaluation of an integral. To avoid this, we will evaluate the integral only at the maximum of $P(x \mid a, \theta)$. If we assume D is orthogonal, then the maximum of $P(x \mid a, \theta)$ occurs when $a = D^{-1}x = W(x)$. Equation 4.22 now becomes

$$\theta^* = \arg\max_\theta P(x \mid W(x), \theta)P(W(x))P(\theta) \tag{4.27}$$

We can then derive our loss function from Equation 4.14 by substituting Equations 4.24-4.26 into Equation 4.27

$$\begin{aligned}\theta^* &= \arg\max_\theta P(x \mid W(x), \theta)P(W(x))P(\theta) \\ &= \arg\max_\theta \log P(x \mid W(x), \theta) + \log P(W(x)) + \log P(\theta) \\ &= \arg\min_\theta ||x - \hat{x}||_2^2 + \lambda_1||W(x)||_1 + \lambda_2 L_w(h, g)\end{aligned} \tag{4.28}$$

Note that we ignore the $1/N$ factor from Equation 4.14 for convenience. Thus, our method can be seen as finding the maximum a priori (MAP) estimate of θ.

4.5.1 Relation to Sparse Coding

Now that we have a probalistic interpretation of our model, we can make a more formal comparison to sparse coding. In particular, we will compare our model to that of [Olshausen and Field, 1997]. Recall that the goal of sparse coding is to find a dictionary, D, and code, a, so as to minimize the error between an input signal and Da subject to a sparsity constraint on the code:

$$\arg\min_{D,a} ||x - Da||_2^2 + L(a) \tag{4.29}$$

where L is the sparsity penalty (often ℓ_0 or ℓ_1). We now summarize the probabilistic derivation from [Olshausen and Field, 1997]. Let us parameterize D by θ (in this case θ represents the basis functions in D). We can then state the probability of a signal being generated from the model as

$$P(x \mid \theta) = \int P(x \mid a, \theta)P(a)\, da \tag{4.30}$$

where we have marginalized over the code.

It is now necessary to define the probabilities in Equation 4.35. The reconstruction probability is given by

$$P(x \mid a, \theta) = \frac{1}{Z_r} e^{-\frac{1}{2\sigma_r^2}||x-Da||_2^2} \tag{4.31}$$

The code prior, $P(a)$, is assumed to be factorial

$$P(a) = \prod_i P(a_i) \tag{4.32}$$

where

$$P(a_i) = \frac{1}{Z_L} e^{-\beta S(a_i)} \tag{4.33}$$

for some function S.

Learning is done by finding the maximum likelihood estimate of $P(x \mid \theta)$

$$\begin{aligned} \theta^* &= \arg\max_\theta P(x \mid \theta) \\ &= \arg\max_\theta \int P(x \mid a, \theta) P(a)\, da \end{aligned} \tag{4.34}$$

In order to compute Equation 4.34, we would have to integrate over all possible values of a. In order to avoid this problem, the integral in Equation 4.34 is evaluated only at its maximum:

$$\theta^* = \arg\max_\theta \left[\max_a P(x \mid a, \theta) P(a) \right] \tag{4.35}$$

This approximation is valid as long as $P(x \mid a, \theta)P(a)$ is tightly peaked [Olshausen and Field, 1997]. Equation 4.34 can be re-written using Equations 4.30-4.33 as

$$\begin{aligned} \theta^* &= \arg\max_\theta \left[\max_a P(x \mid a, \theta) P(a) \right] \\ &= \arg\max_\theta \left[\max_a \log P(x \mid a, \theta) + \log P(a) \right] \\ &= \arg\min_\theta \left[\min_a ||x - Da||_2^2 + \lambda \sum_i S(a_i) \right] \end{aligned} \tag{4.36}$$

where $\lambda = 2\sigma_r^2 \beta$. Since Equation 4.36 contains a nested minimization, optimization proceeds in two phases. The first (inner) stage is to find the a that minimizes Equation 4.36 given a fixed θ. The second stage, similarly, finds the θ that minimizes Equation 4.36 given a fixed a. Details of the algorithm can be found in [Olshausen and Field, 1996].

We can now see the similarities between our model and that of the sparse coding model as presented in [Olshausen and Field, 1996]. Indeed, we may view our model as a restricted version of sparse coding. In our model, θ is the scaling and wavelet filter values and a represents the wavelet coefficients of x (i.e. $W(x)$).

There are two major differences between our model, and the sparse coding model as described above. The first is how we estimate θ. We find the θ that maximizes $P(\theta \mid x)$, whereas Olshausen and Field maximize $P(x \mid \theta)$. In other words, we use MAP estimation as opposed to maximum likelihood estimation (MLE). We can view MLE as equivalent to MAP if we choose a uniform prior for θ. The second difference is that we restrict ourselves to using a complete, orthogonal basis. We thus make use of the approximation in Equation 4.27, leading to a learning objective (Equation 4.28) that can proceed in a single stage.

4.5.2 Sparsity Revisited

We can also approach the choice of sparsity function from a probabilistic viewpoint. Sparsity penalties used in the loss function can be interpreted as particular choices of prior distributions over the representation elements. We will see that using the ℓ_1 norm corresponds to placing a Laplacian (or double exponential distribution) prior over the wavelet coefficients. A Laplacian is a useful distribution for sparsity, as it has a steep peak at zero and long tails.

We now outline a probabilistic interpretation of sparse codes, using a derivation similar to [Olshausen et al., 2001]. Let us consider the posterior of the wavelet coefficients

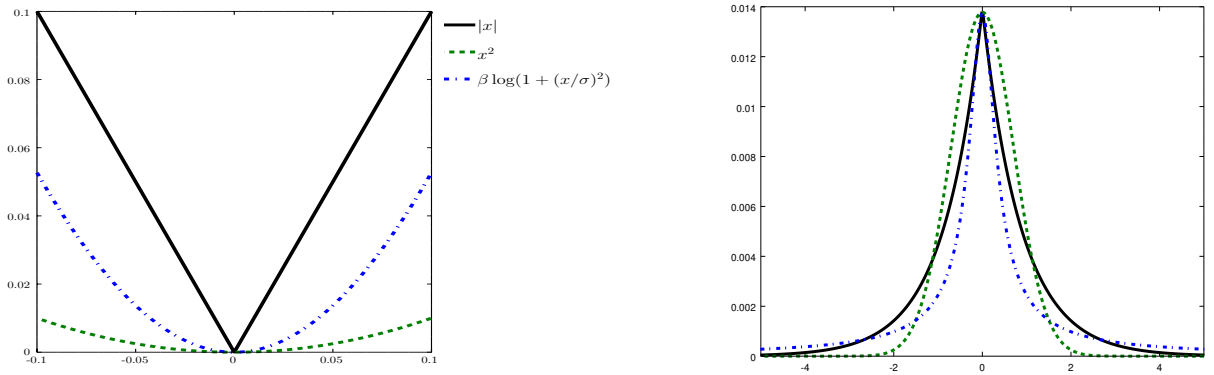
$$P(a \mid x, \theta) \tag{4.37}$$

where the parameters, θ, correspond to wavelet and scaling filters. In order to find the coefficients that maximize the posterior (i.e. the MAP estimate), we must find

$$\begin{aligned} \arg\max_a P(a \mid x, \theta) &= \arg\max_a P(x \mid a, \theta) P(a \mid \theta) \\ &= \arg\min_a \left[\log P(x \mid a, \theta) + \log P(a \mid \theta) \right] \end{aligned} \tag{4.38}$$

We make use of the ℓ_2 norm for our reconstruction error, and so we have

$$P(x \mid a, \theta) = \frac{1}{Z} e^{-||x - Da||_2^2} \tag{4.39}$$



(a) (b)

Figure 4.13: (a) Comparison of three different sparsity functions. (b) Comparison of Laplacian, Gaussian, and Cauchy-like priors.

where Z is a normalization factor. Hence, using MSE corresponds to assuming a Gaussian error on our reconstruction. Let us further assume that the prior over a, represented as $P(a)$, can be factored as follows:

$$P(a) = \prod_i P(a_i) \tag{4.40}$$

If we restrict ourselves to a distribution from the exponential family, we have

$$P(a_i) = \prod_i \frac{1}{Z_L} e^{-L(a_i)} \tag{4.41}$$

for some sparsity penalty L. It now becomes clear that choosing a sparsity penalty is equivalent to choosing a particular distribution for the prior of the wavelet coefficients. The ℓ_1 norm, squared ℓ_2 norm, and penalty from [Olshausen et al., 2001] correspond to Laplacian, Gaussian, and Cauchy-like priors respectively. Figure 4.13b shows plots of these distributions normalized to have equal heights. A Laplacian prior works better than a Gaussian for sparsity because it has a sharper peak at zero, and has long tails that allow for large non-zero elements. The Cauchy prior seems like it should be an even better choice of prior, but as we discussed earlier, suffers problems when using gradient descent.

4.5.3 Signal Generation

In order to determine how well the learned wavelets capture the structure of the training data signals, we will consider signals randomly generated from the learned wavelets. To generate signals we will randomly sample wavelet coefficients from the following Laplacian density function

$$\frac{1}{4}e^{-2|x|} \tag{4.42}$$

The generated signal is obtained by performing an inverse wavelet transform of the sparse coefficients. Qualitative results are shown in Figure 4.14. Typical training examples are shown in the left column. Example generated signals are shown in the middle column. Note that the generated signals have visually similar structure to the training examples. This provides evidence that the learned wavelets have captured the structure of the data.

The generation process does not exactly match the generation process from Equation 6.14. Most notably, we do not account for periodicity. Thus, the generated signals do not match the global structure of the training signals very well. We can impose periodicity in our generation process simply by repeatedly concatenating the randomly sampled coefficients at each scale. The new generative process is as follows. For each wavelet scale, k, we set the period to be 2^k (the same as in Equation 6.14). We then sample $n/2^k$ coefficients from the Laplacian as defined in Equation 4.42, where n is the number of wavelet coefficients at scale k. These $n/2^k$ coefficients are then repeated 2^k times to form the n wavelet coefficients. Finally, after this process is complete for all scales, we apply an inverse wavelet transform. Some sample generated signals are shown in Figure 4.14 (right column). Note that their structure is qualitatively similar to the training signals.

These experiments are included to demonstrate that our wavelet model is not generally suited for signal generation. In this case we required more information about the signals (i.e. periodicity) in order to generate signals that were similar to training examples. This is an inherent limitation of standard autoencoders because they do not explicitly model generative processes. Two examples of generative neural network models are variational autoencoders [Kingma and Welling, 2013] and generative adversarial networks [Goodfellow et al., 2014]. It may be possible to adapt our model to allow for better signal generation by incorporating ideas from these generative network models. We will return to this topic in Chapter 7.

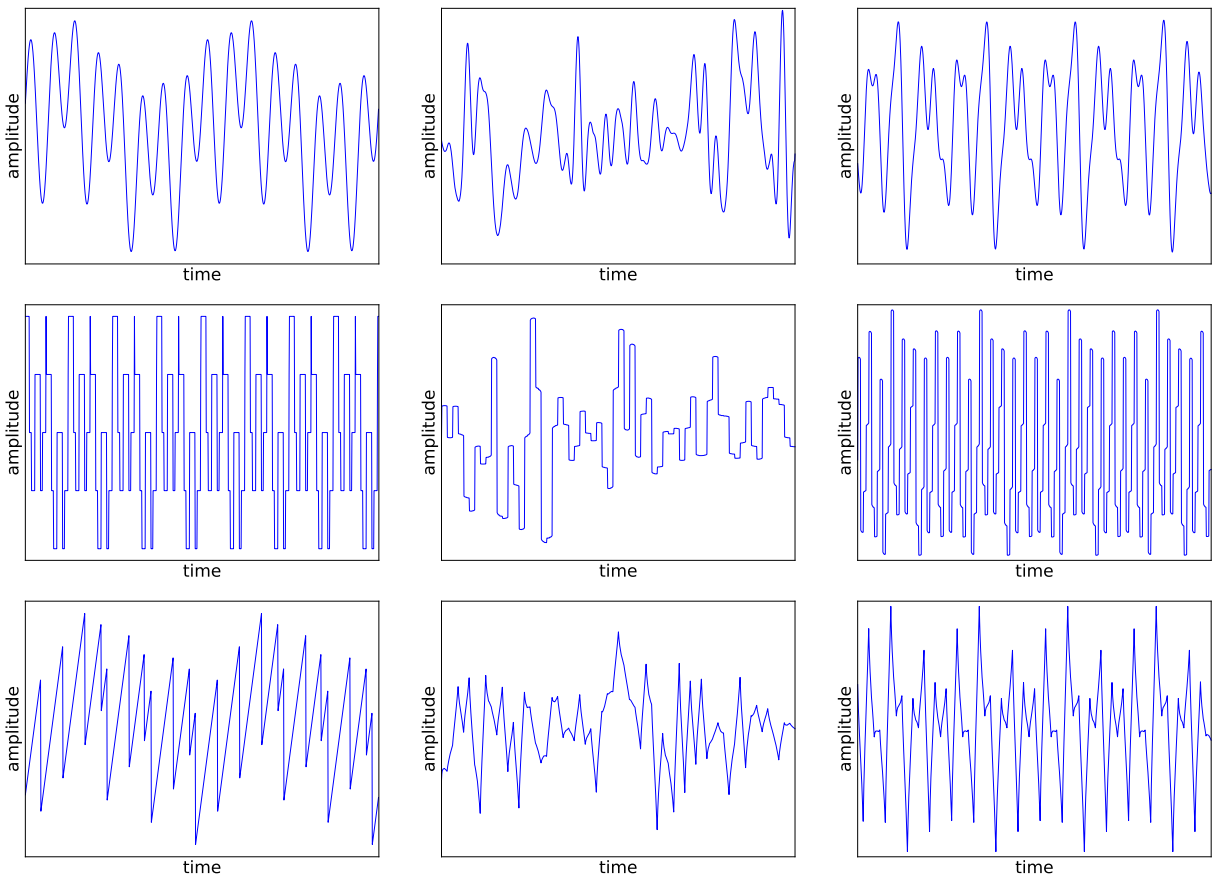


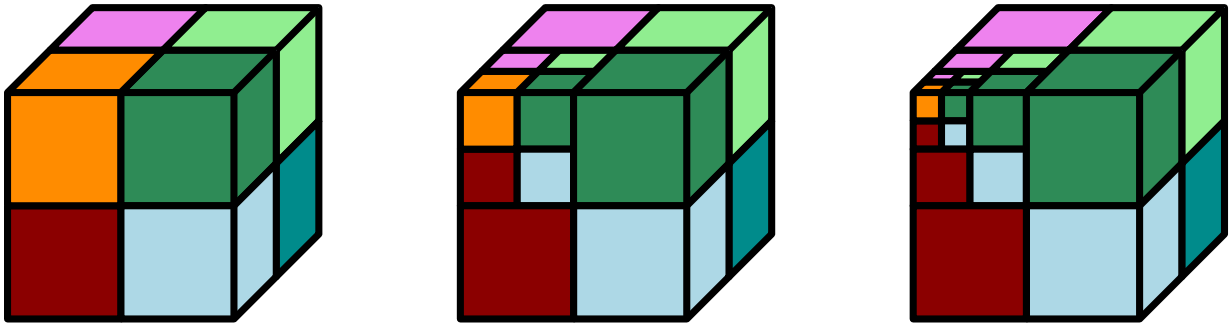
Figure 4.14: Left column: Examples of synthetic training signals. Middle column: Examples of signals generated from the corresponding learned wavelet filters by sampling coefficients from a Laplacian prior. Right column: Same as middle column, but with a periodicity constraint in each wavelet band. Base waves from top to bottom: sine, square, sawtooth.

4.6 Conclusion

We have proposed a new model capable of learning useful wavelet representations from data. We accomplish this by framing the wavelet transform as a modified CNN. We show that we can learn useful wavelet filters by gradient descent, as opposed to the traditional derivation of wavelets using Fourier methods. The learned wavelets are able to capture the structure of the training data. We hope that our work leads to wider use of the wavelet transform in the machine learning community. Framing our model as a neural network has the benefit of allowing us to leverage deep learning software frameworks, and also allows for simple integration into existing neural network architectures (which we will see in Chapter 7). An advantage of our method is the ability to learn directly from raw audio data, instead of relying on a fixed representation such as the Fourier transform.

Chapter 5

Multidimensional Signals



The goal of this chapter is to extend the one-dimensional wavelet model to multiple dimensions. To accomplish this, we will make use of a straightforward generalization of the one-dimensional wavelet transform algorithm that makes use of separable filters. In this way, our model will be able to process multidimensional signals while still learning one-dimensional filters. Most of the derivations from Chapter 4 will not change, including the loss function and probabilistic interpretation of our model. We will focus on two-dimensional signals since many applications in the literature are concerned with images. We also briefly explore some image generation methods.

5.1 2D Wavelet Transform

A motivation of this section is to learn feature representations of images directly from data. This is in contrast to the more traditional method of using a fixed feature representation

such as SIFT [Lowe, 1999] or SURF [Bay et al., 2008]. One of the most notable methods of learning directly from image data is the CNN [LeCun et al., 1990], which learns a set of 2D filters that are applied in a cascade. Our model instead learns 1D filters that are applied along each dimension of the image. Thus, the filters still have a 2D receptive field, but fewer parameters are required. Furthermore, we reuse the filters in each layer of the network, unlike in a traditional CNN where separate filters are used at each layer.

The discrete wavelet transform can be extended to two dimensions by computing the filter convolutions along each axis separately [Mallat, 1989]. In the 1D case, we computed two components at each iteration of the algorithm (highpass and lowpass). In the 2D case, we will compute four components. Let LR and LC correspond to convolving the scaling (lowpass) filter along the rows and columns respectively. We can similarly define HR and HC for the wavelet (highpass) filter. At every iteration of the 2D wavelet transform algorithm, we compute the four components as illustrated in Figure 5.1:

$$\text{approximation: } LC(LR(x)) \tag{5.1}$$

$$\text{detail horizontal: } LC(HR(x)) \tag{5.2}$$

$$\text{detail vertical: } HC(LR(x)) \tag{5.3}$$

$$\text{detail diagonal: } HC(HR(x)) \tag{5.4}$$

where x is now a 2D discrete signal. The three components computed using at least one wavelet filter are kept as the detail coefficients. The single approximation component computed by convolving the scaling filter along both axes is passed to the next iteration of the algorithm. The output of the transform is structured as in Figure 5.3a. Note that after each convolution, the image is downsampled by a factor of two along the direction of the convolution. Computing the 2D wavelet transform in this fashion is used in the JPEG2000 standard [Christopoulos et al., 2000].

The three components containing the detail coefficients each correspond to a different orientation: horizontal, vertical, and diagonal. Each component responds to changes along its corresponding direction. Figure 5.2 shows one level of the wavelet transform applied to an image. Note that the three detail coefficient components highlight different edge orientations.

As in the 1D case, we subsample by a factor of two after each convolution. Thus, the total number of coefficients is equal to the number of pixels in our original image. Figure 5.3 shows how the coefficients are typically represented graphically. Note that we can discard the approximation coefficients after each iteration as they can be reconstructed by the coefficients at the next level.

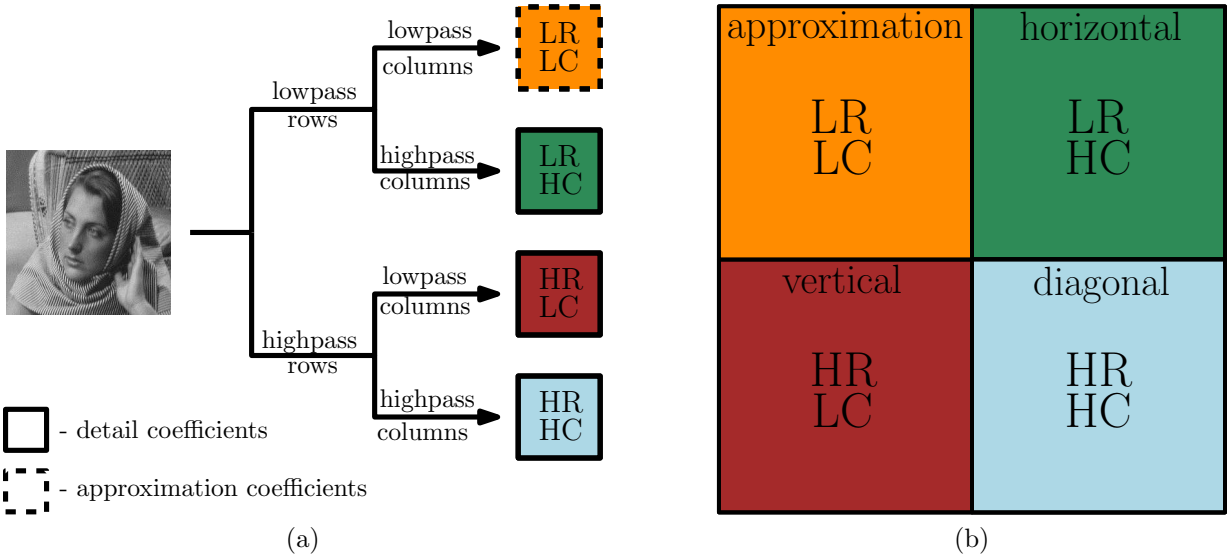


Figure 5.1: (a) One iteration of the 2D discrete wavelet transform produces four coefficient components. (b) Typical visual layout of the four coefficient components.

To get an idea of how the 1D wavelet filters behave in 2D, we can compute the impulse responses associated with each orientation. The impulse responses are computed by an inverse wavelet transform on wavelet coefficients that have a single nonzero value in each of the three detail coefficient components. The impulse responses can be thought of as the images that each filter orientation maximally responds to. Figure 5.4 shows the impulse responses for a typical wavelet filter. The first two impulse responses correspond to the horizontal and vertical components. The third impulse response has a checkerboard appearance since it is effectively the product of the first two. The 2D wavelet transform thus suffers from poor directional selectivity, which can be addressed by using a dual-tree version of the transform [Selesnick et al., 2005]. We return to this idea in Chapter 6.

5.1.1 Algorithm Analysis

The forward and inverse algorithms are outlined in Algorithms 5.1 and 5.2. A bar over over an operation indicates the inverse (e.g $\overline{LR}$ indicates upsampling the rows followed by a convolution with the inverse scaling filter). One may view the 2D transform as applying a 1D transform along each of the rows and columns of an image. Recall from Section 2.4.1 that the time complexity of the 1D transform is linear in the length of the signal. Thus,

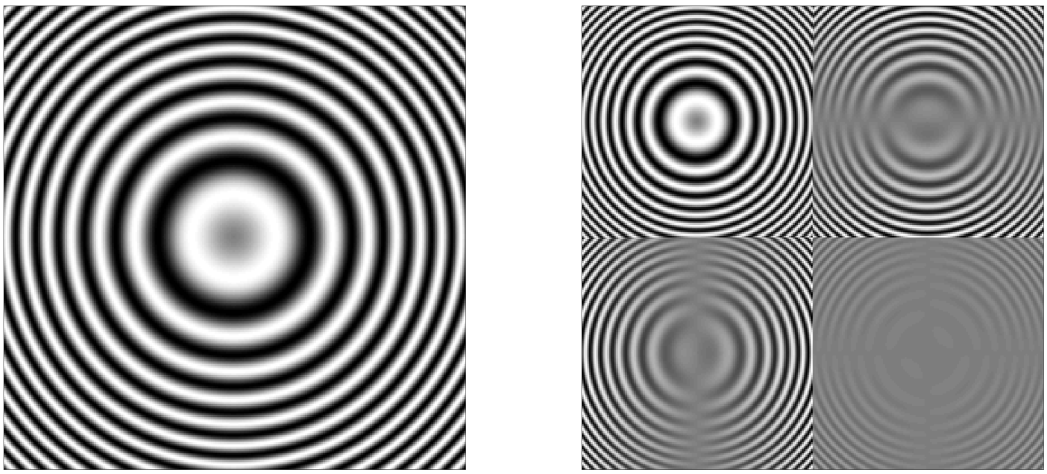


Figure 5.2: One iteration of the 2D discrete wavelet transform applied to a circular image. Not that the different coefficient components pick up different edge orientations.

for an $n \times n$ image, we would expect the time complexity of the 2D transform to be $O(n^2)$. This intuition is correct, and will be proven below.

Consider one iteration of the loop in Algorithm 5.1. Each of the four statements involving convolution have equivalent time complexities, so it is sufficient to analyze only the first:

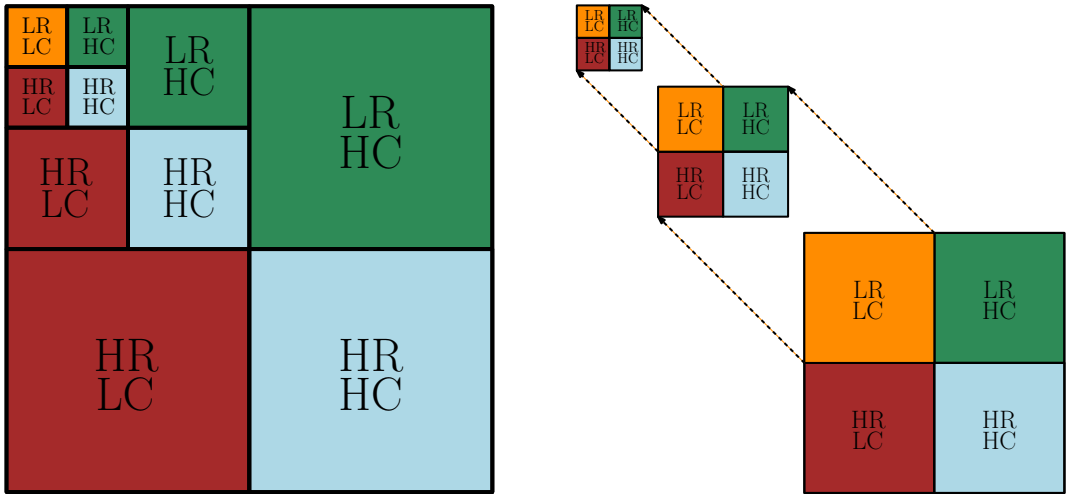
$$a_{j+1} \leftarrow LC(LR(a_j)) \tag{5.5}$$

Let us suppose we are in the first loop iteration. The row convolutions, $LR(a_j)$, each take $O(n)$ time (Section 2.4.1). Since n of them are performed, $LR(a_j)$ can be computed in $O(n^2)$ time. Computing the column convolutions can similarly be performed in $O(n^2)$ time, so the total time complexity of $LC(LR(a_j))$ is $O(n^2)$. Each loop iteration is composed of four of these operations (followed by a simple concatenation operation that we can ignore here), and thus the time complexity for the first loop iteration is $O(n^2)$. Since the dimensions of the image are halved at each iteration, the time complexity of all iterations can be represented as

$$O(n^2 + (n/2)^2 + (n/4)^2 + \cdots + 1) \tag{5.6}$$

By observing that the sum is a geometric series that is of order $O(n^2)$, we complete our analysis.

We could also apply the same method in Section 2.4.1 by viewing the algorithm recur-



(a) (b)

Figure 5.3: (a) Wavelet coefficient matrix after three iterations of the wavelet transform algorithm. (b) The wavelet coefficients are computed from the scaling coefficients of the previous iteration.

sively. Using the arguments above, the recurrence relation is

$$T(n) = O(n^2) + T(n/2) \tag{5.7}$$

By the master theorem the total time complexity is $O(n^2)$. This analysis can be easily generalised to the multidimensional case, resulting in a time complexity of $O(n^d)$ for a d dimensional signal. However, it should be noted that the algorithm is still linear in the total number of elements in the signal.

Figure 5.4: Impulse responses of a typical wavelet filter.

Algorithm 5.1 2D Discrete wavelet transform

Require: f is a discrete 2D signal, L is the maximum level of the transform.

function 2D-DWT(f, L)
    $a_0 \leftarrow f$
    for $j \leftarrow 0$ to $L-1$ **do**
        $a_{j+1} \leftarrow LC(LR(a_j))$
        $d^h_{j+1} \leftarrow LC(HR(a_j))$
        $d^v_{j+1} \leftarrow HC(LR(a_j))$
        $d^d_{j+1} \leftarrow HC(HR(a_j))$
        $d_{j+1} \leftarrow \langle d^h_{j+1}, d^v_{j+1}, d^d_{j+1} \rangle$
    end for
    return $\langle d_1, d_2, \ldots, d_L, a_L \rangle$
end function

Algorithm 5.2 Inverse 2D discrete wavelet transform

Require: $\langle d_1, d_2, \ldots, d_L, a_L \rangle$ as computed per Algorithm 5.1.

function 2D-IDWT($\langle d_1, d_2, \ldots, d_L, a_L \rangle$)
    for $j \leftarrow L-1$ to 0 **do**
        $a_j \leftarrow \overline{LR}(\overline{LC}(a_{j+1})) + \overline{HR}(\overline{LC}(d^h_{j+1})) + \overline{LC}(\overline{HR}(d^v_{j+1})) + \overline{HC}(\overline{HR}(d^d_{j+1}))$
    end for
    return a_0
end function

5.2 The Wavelet Transform as a Neural Network

Just as in the 1D case, our 2D wavelet network can be viewed in terms of neural networks. The wavelet transform is computed by a cascade of convolutions. This computation is similar in structure to that of CNNs. As such, we propose a modified CNN that directly computes the wavelet transform. By doing so, we are able to leverage the mathematical properties of the wavelet transform into the successful deep learning architectures. In simplest terms, our model is an unrolling of the discrete wavelet transform algorithm. Figure 5.5 shows an overview of our wavelet transform network. Each layer of the network computes one iteration of the discrete wavelet transform. The detail coefficients are passed directly to the final output layer. The approximation coefficients are passed as input to the next layer. The parameters of the network are the wavelet and scaling filters. These filters are reused at each layer of the network, and hence the model is only required to learn a single filter pair. This property is similar to that of recurrent neural networks, where weights are reused at each time step [Pascanu et al., 2013]. In our network, however, weights are reused at each scale.

As in the 1D case, we restrict ourselves to quadrature mirror filters

$$g[n] = (-1)^n h[-n]. \tag{5.8}$$

and make use of the same penalty terms

$$L_w(h, g) = (||h||_2^2 - 1)^2 + (\mu_h - \sqrt{2}/k)^2 + \mu_g^2 \tag{5.9}$$

where μ_h and μ_g are the filter means. The first two terms prefer a scaling function with unit L^2 norm and finite L^1 norm respectively. The last term is a relaxed orthogonality constraint which prefers a wavelet filter with zero mean. See Figure 5.7 for a selection of random filters that minimize Equation 5.9. Like in the 1D case, we can see that the constraints by themselves are not sufficient for learning meaningful filters.

We reuse our autoencoder framework from Chapter 4 (see Figure 5.6). The goal is to reconstruct an input image by first computing the forward wavelet transform, imposing sparsity constraints on the coefficients, and then performing an inverse wavelet transform. We choose to use an ℓ_1 sparsity constraint in order to prefer mostly zero coefficients. We argue that a wavelet that gives a sparse representation of an image must exploit inherent structure present in the data. Thus, the model must learn something useful about the images used for training.

To verify this claim, we test our model with real and synthetic data. In our experiments we make use of images of faces [Lee et al., 2005] and synthetic images containing harmonic

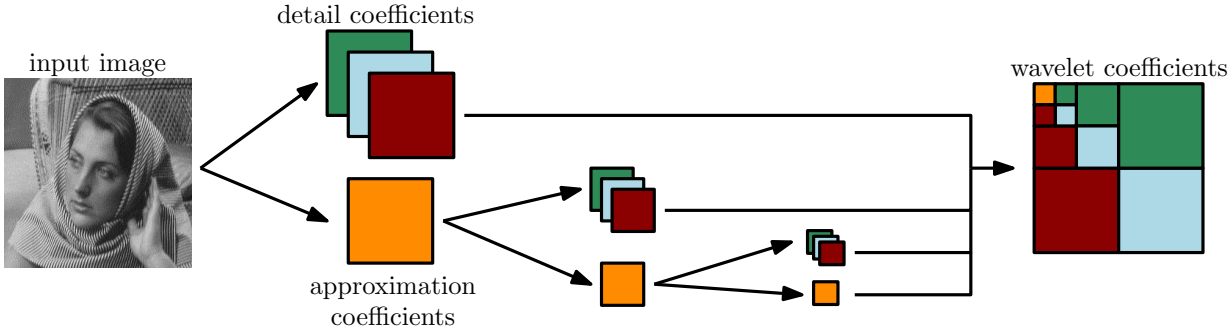


Figure 5.5: The wavelet transform as a neural network. Each layer of the network computes the three detail components and single approximation component. The approximation coefficients are passed to the next layer, while the detail coefficients are passed to the final layer. The number of layers corresponds to the number of iterations of the wavelet transform algorithm.

waves of different shapes. We extend the synthetic generation process for 1D data from Chapter 4:

$$x(t) = \sum_{k=0}^{K-1} a_k \cdot s(2^k t + \phi_k) \tag{5.10}$$

where ϕ_k is a phase offset chosen uniformly at random from $[0, 2\pi]$, and a_k is the k^{th} harmonic indicator which takes the value of 1 with probability $1/2$ and zero otherwise. In order to generate images, we first choose a random orientation angle for each harmonic wave. In the case of axis-aligned waves, the angle is 0 or $\pi/2$. The waves are then added to the image along the chosen orientation and extended orthogonally to fill the image. Three base waves are considered: square waves, sawtooth waves, and sine waves. See Figure 5.9 for samples of synthetic images and Figure 5.8 for sample faces.

Our loss function from Chapter 4 can be reused

$$L(X; g, h) = \frac{1}{MN} \sum_{i=1}^{M} ||x_i - \hat{x}_i||_2^2 + \lambda_1 \frac{1}{MN} \sum_{i=1}^{M} ||W(x_i)||_1 + \lambda_2 L_w(h, g) \tag{5.11}$$

We use squared error for our reconstruction loss and the ℓ_1 norm for the sparsity penalty. The parameters λ_1 and λ_2 control the trade-off between the three loss terms. In our experiments we set $\lambda_1 = \lambda_2 = 1/2$ and used a filter length of ten. Our model was implemented using Google's Tensorflow library and makes use of automatic differentiation [Abadi et al., 2015]. We trained using the Adam gradient descent algorithm with a batch size of four [Kingma and Ba, 2014].

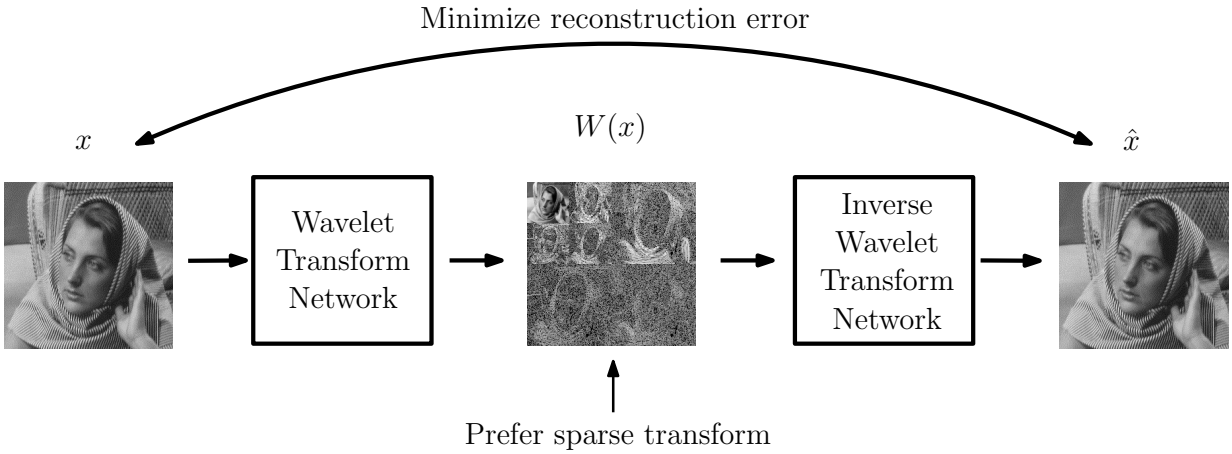


Figure 5.6: The autoencoder framework used in our experiments. An ℓ_1 penalty is put on the wavelet coefficients so that the learned wavelets must exploit the structure in the data.

A sample of the learned wavelet functions can be found in Figure 5.10. These functions were learned from axis-aligned data where applicable. The wavelets learned from the randomly oriented data had similar structure, so are not included here. The wavelet and scaling functions are computed from the filter coefficients using the cascade algorithm [Strang and Nguyen, 1996]. Note that the learned wavelet functions are able to capture the structure of the different base waves present in the data. We compared the learned filters to traditional wavelets from the following families: Haar, Daubechies, Symlets, and Coiflets. The most similar traditional wavelets are included in Figure 5.10. The distance measure used was the cosine distance under all rotations of the filters (the same as in Chapter 4):

$$dist(h_1, h_2) = \min_{0 \le i < k} 1 - \frac{\langle h_1, \text{shift}(h_2, i) \rangle}{||h_1||_2 \cdot ||h_2||_2} \tag{5.12}$$

where $\text{shift}(h, i)$ is the circular shift of h by i samples. If the two filters are of different lengths, the shorter filter is zero-padded to match the length of the longer filter.

We can also consider the impulse responses of the filters. Figure 5.11 shows the impulse responses of the learned filters from Figure 5.10. Surface plots of the impulse responses are also included. Note that the first two impulse responses are axis-aligned. The shape of each impulse response is similar in structure to its corresponding training data. Square data yields rectangular filters, sawtooth data yields triangular filters, and sine data yields filters that appear Gabor-like.

One way to determine how well the learned wavelets capture the structure of the data

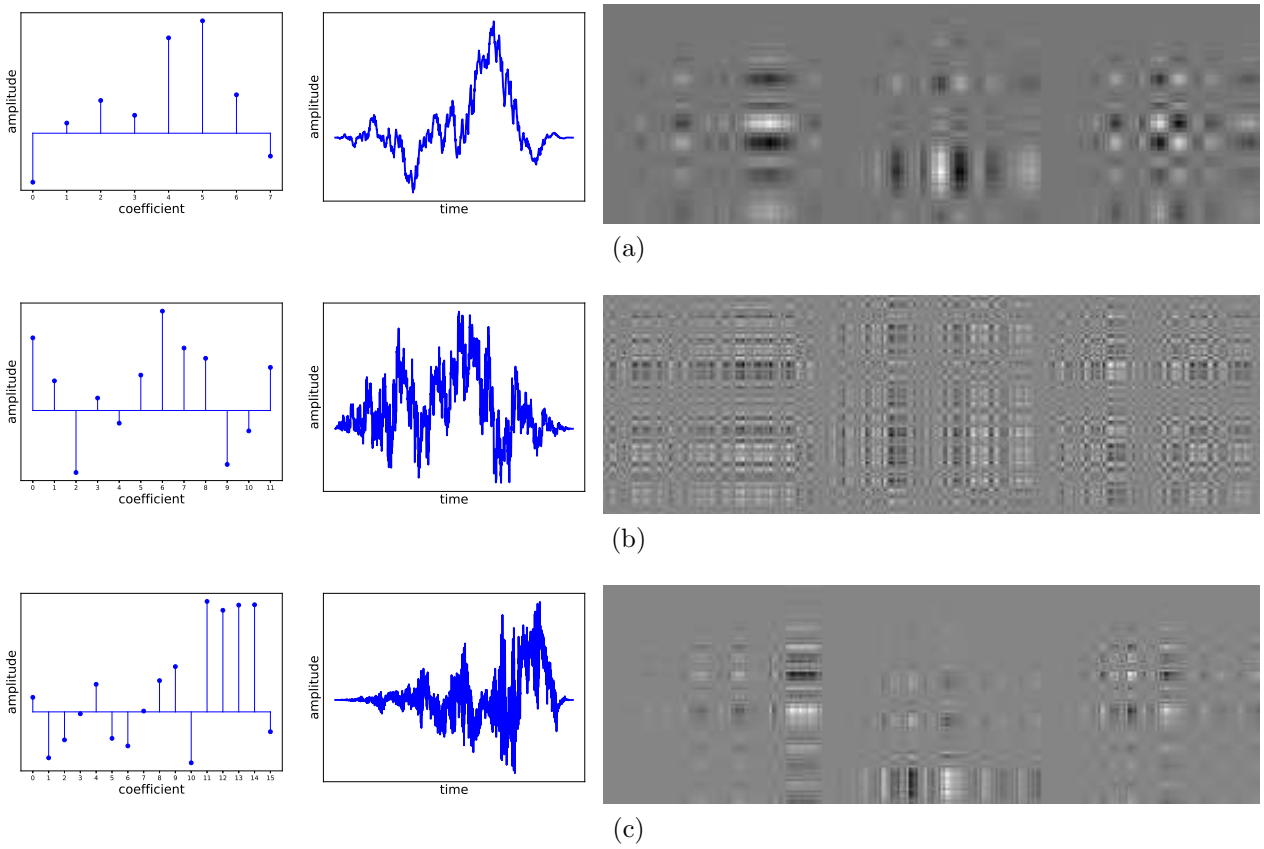


Figure 5.7: Impulse responses of random filters satisfying Equation 5.9 with lengths (a) 8, (b) 12, and (c) 16.

is to generate from the model. Image generation has a long history in computer science [Mumford and Desolneux, 2010]. Our generative process consists of sparsely populating random wavelet coefficients, and then computing an inverse transform using one of the learned wavelets. The density of coefficients is the same across all wavelet scales, and is equal to 10% (i.e. 90% of coefficients are randomly set to zero). Figure 5.12 shows examples of generated images using various learned wavelets. The first column uses coefficients of equal magnitude and includes coefficients at each scale. It is difficult to see any structure in this data as it appears similar to white noise due to the high frequency components. The second column is similar, but excludes the three highest frequency scales. We can see that different wavelets generate different structured images. The third column scales the magnitudes of the coefficients so that low frequency scales are more pronounced (similar to pink noise). The scaling factor is set to 2^i for scale i. We again see the different structure

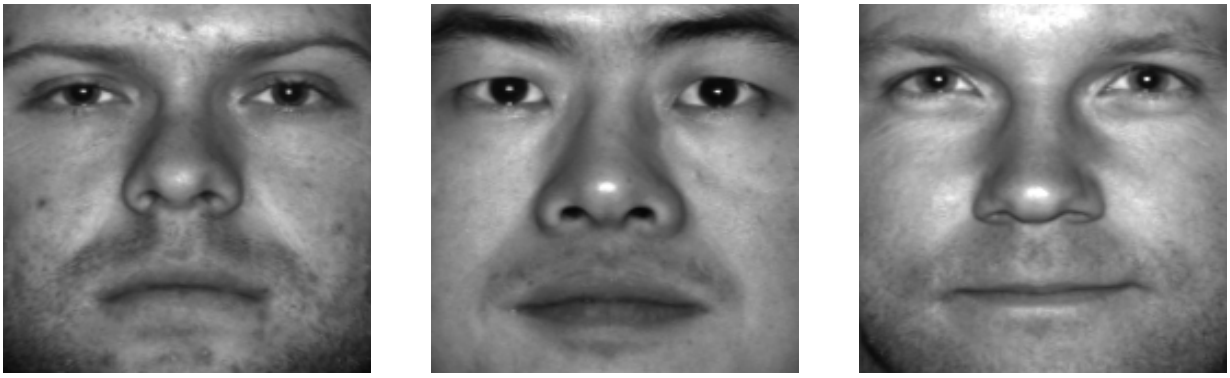
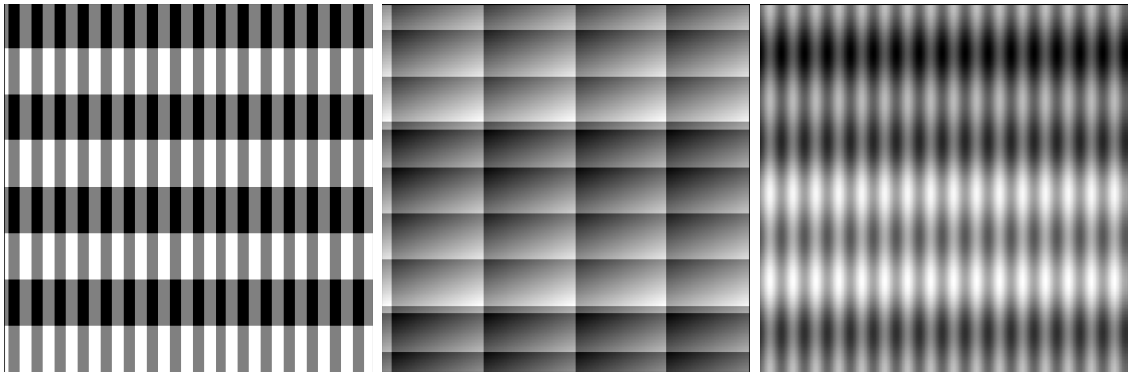


Figure 5.8: Samples of faces from the Yale face dataset.

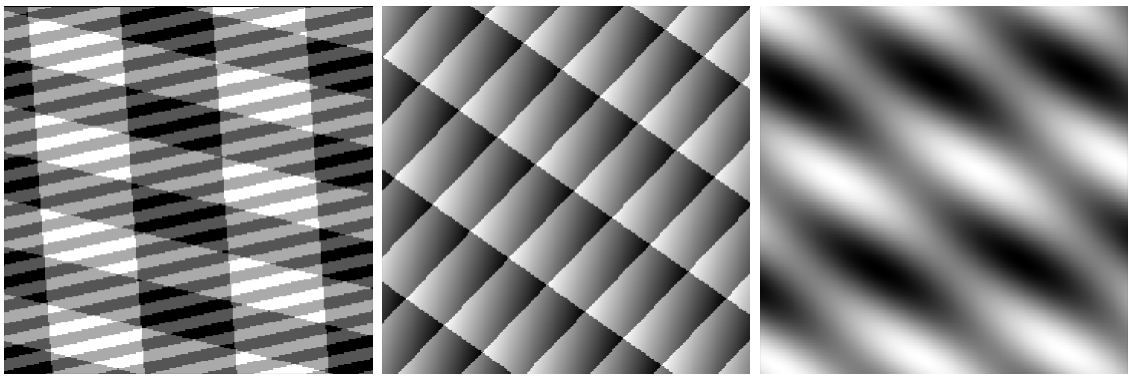
of the wavelets.

Like in the 1D generation results from Chapter 4, the generated images share similar local structure to the training images. However, the global image structure is not captured by the model. Generating images that are visually similar to the training data will require modifications to the autoencoder framework. Building a generative wavelet model is left as future work.

Our final set of experiments consider image reconstruction from a small number of wavelet coefficients. The sparsity penalty in our wavelet model prefers wavelets that give rise to very few nonzero coefficients. Thus, images should be well approximated by a small number of wavelet coefficients. However, wavelets that are not tuned to the data should not provide the same quality of reconstruction. We test this by comparing the performance of the learned face wavelet to the Haar wavelet. Figure 5.13a shows an image of a face reconstructed using different numbers of wavelet scales after a five level wavelet decomposition. The leftmost image is reconstructed using only scaling coefficients. As wavelet coefficients are added (starting from the lowest frequency), we see that the learned face wavelet is qualitatively better at reconstructing the image. The Haar wavelet reconstruction results in noticeable blocky artifacts. Figure 5.13b shows a similar process, but here the image is reconstructed using the wavelet coefficients with largest magnitude. Keeping only the largest 5% wavelet coefficients results in reasonable reconstructions for both wavelets. The learned wavelet, however, performs better as we decrease the number of coefficients. These results suggest that: 1) wavelets may be useful for image compression, and 2) learned wavelets may achieve better compression than fixed wavelets. We will return to the task of image compression in Chapter 7.



(a)



(b)

Figure 5.9: Samples of synthetic images with harmonics that are (a) axis-aligned and (b) randomly oriented. Base waves from left to right: square, sawtooth, and sine.

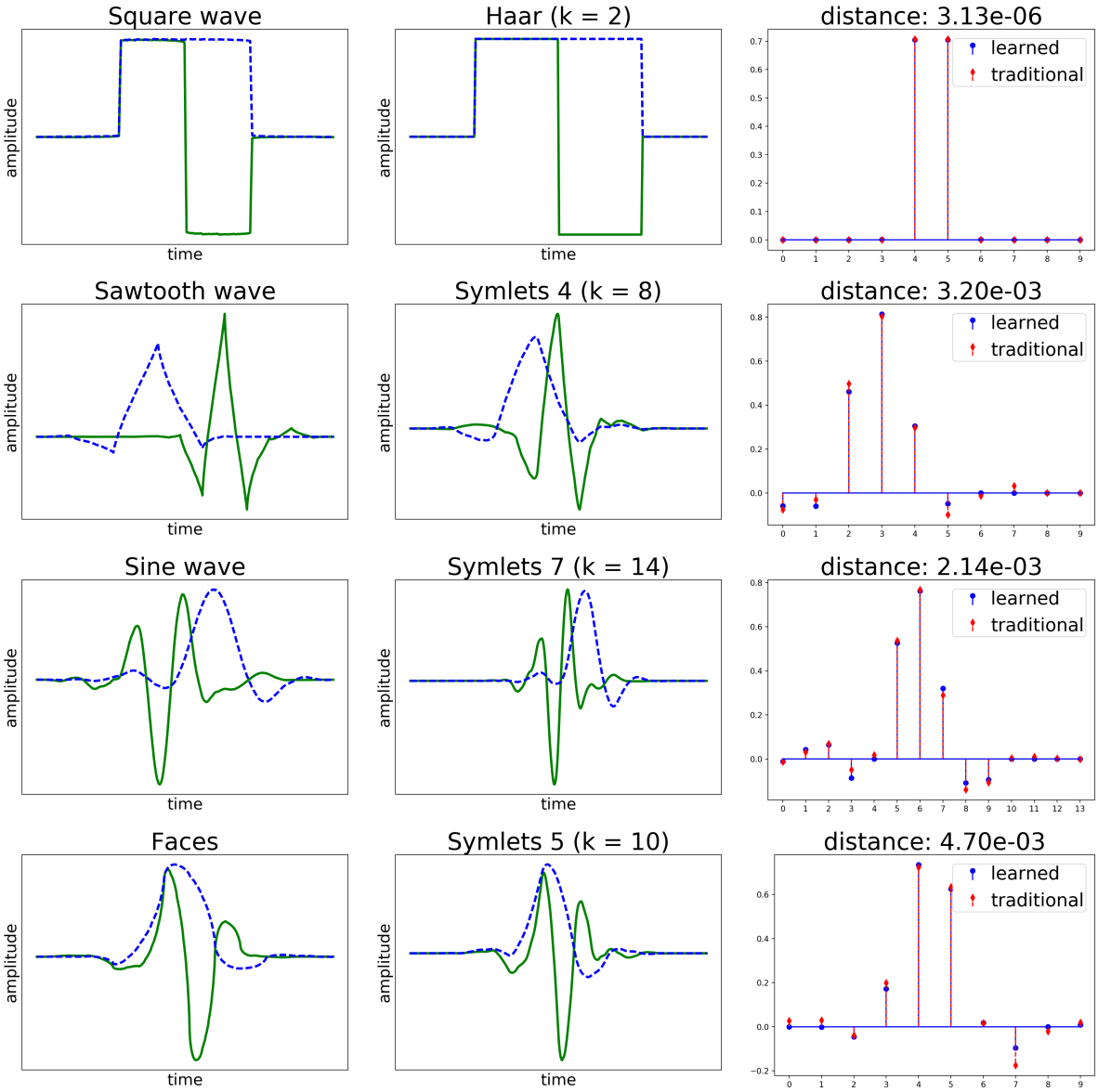
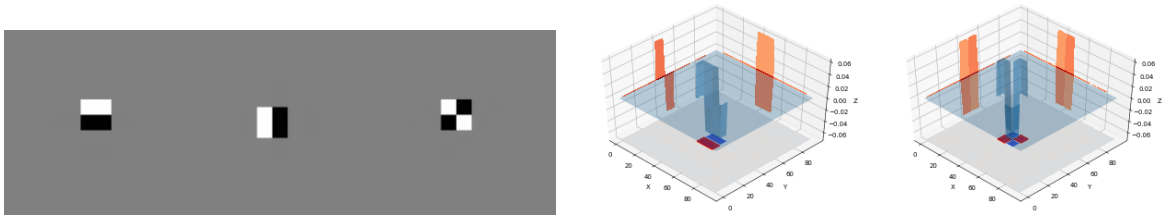
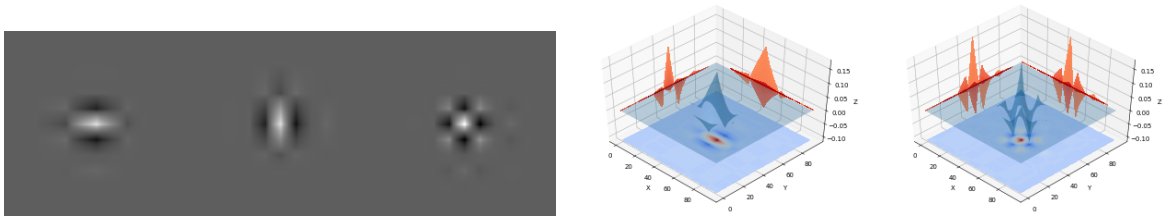


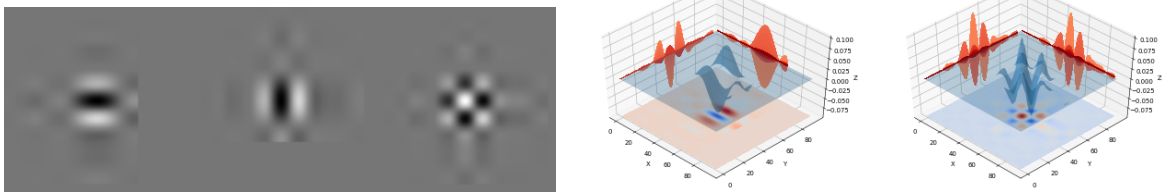
Figure 5.10: Summary of learned filters. Left column: Learned wavelet (solid) and scaling (dashed) functions. Type of training data is shown above the plots. Middle column: Closest traditional wavelet (solid) and scaling (dashed) functions according to (5.12). Right column: Plots of the scaling filters from the first two columns with corresponding distance measure.



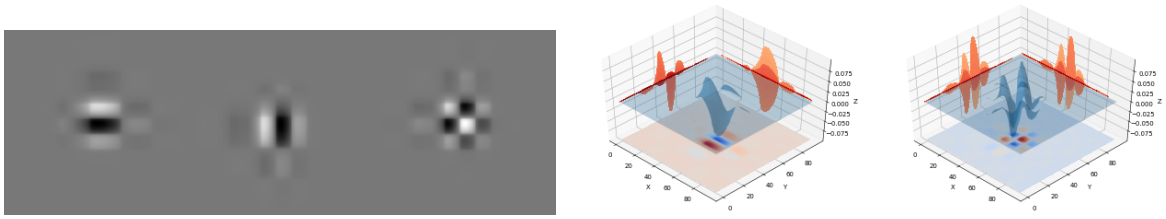
(a)



(b)

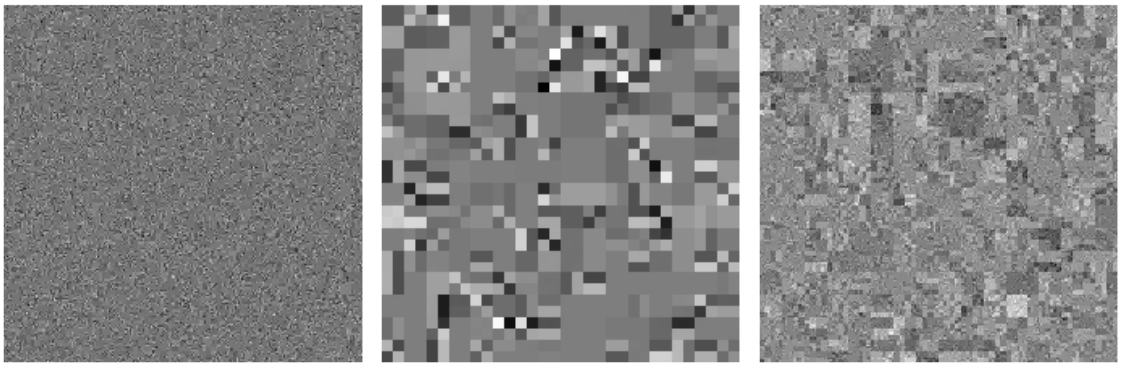


(c)



(d)

Figure 5.11: Impulse responses of the learned filters from Figure 5.10. The training data was (a) square waves, (b) sawtooth waves, (c) sine waves, and (d) faces. Surface plots of the horizontal and diagonal orientations are also shown shown.

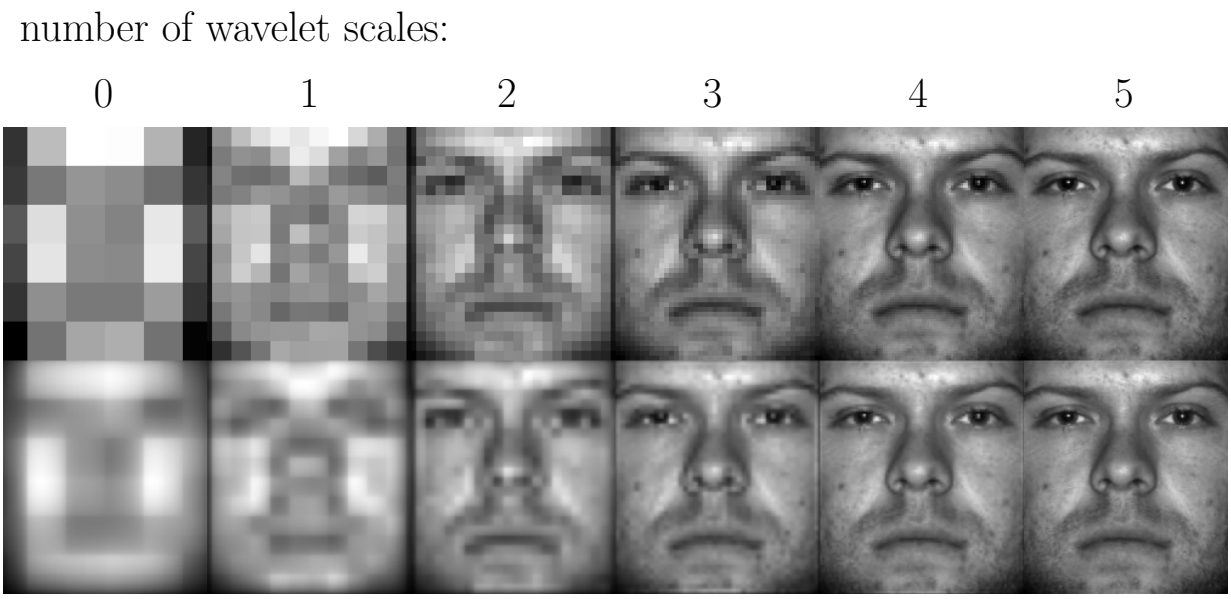


(a)

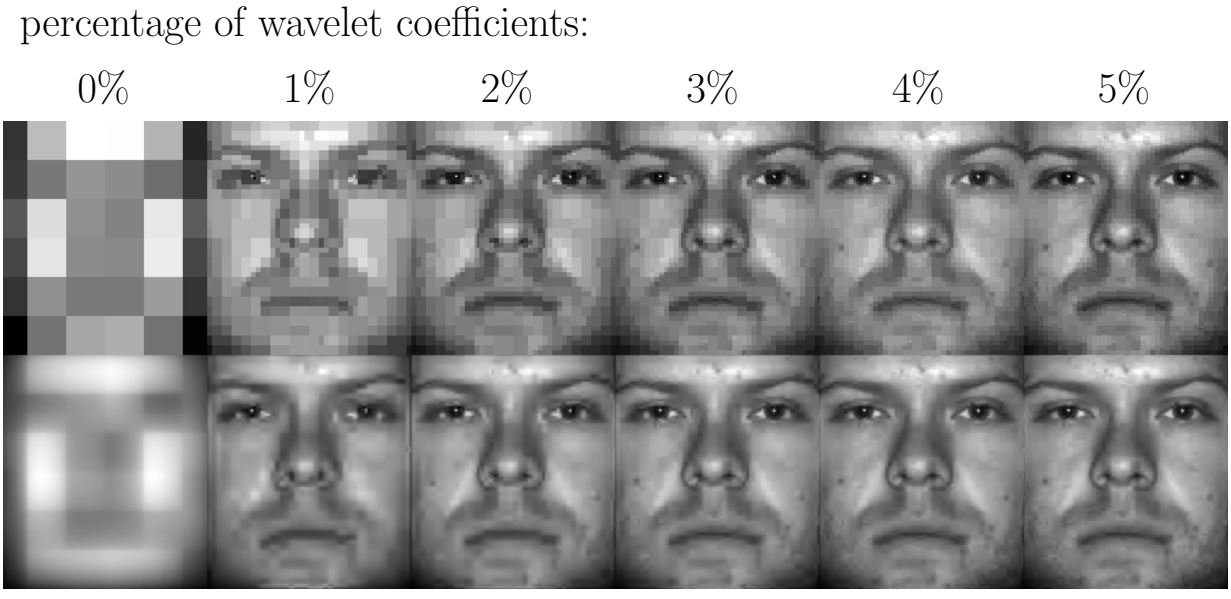
(b)

(c)

Figure 5.12: Generated image data with (a) square wavelet, (b) sawtooth wavelet, and (c) sine wavelet. Left column: equal power at each scale. Middle column: equal power at each scale with three highest scales removed. Right column: coefficients scaled so that lower frequencies are more pronounced.



(a)



(b)

Figure 5.13: A face image reconstructed from different amounts of wavelet coefficients using a Haar wavelet (top row in (a) and (b)) and the learned face wavelet (bottom row in (a) and (b)). A five level wavelet transform was used in all cases. All scaling coefficients were used in each reconstruction.

5.3 3D Wavelet Transform

The wavelet transform is easily extended into multiple dimensions. For practicality, we will end our discussion with the third dimension. In 3D, our data will have height, width, and depth dimensions. A typical example of 3D data is an MRI image (see Figure 5.18). Videos can also be viewed as 3D images, with the depth dimension representing time.

In the 2D case we computed four components per iteration:

$$LC(LR(x)),\ LC(HR(x)),\ HC(LR(x)),\ HC(HR(x)).$$

In the 3D case we introduce LD and HD to represent convolving with the scaling and wavelet filters respectively along the depth dimension. Thus we have eight components:

$$\begin{aligned}\text{approximation: } & LD(LC(LR(x)))\\ \text{detail: } & LD(LC(LR(x))),\ LD(LC(LR(x))),\ LD(LC(LR(x))),\ LD(LC(LR(x))),\\ & LD(LC(LR(x))),\ LD(LC(LR(x))),\ LD(LC(LR(x)))\end{aligned}$$

Figure 5.14 shows the eight components computed at each stage of the algorithm. Figure 5.15 shows the typical component arrangement. Note that like the 1D and 2D case, the total number of coefficients is equal to the number of elements in the input.

The 3D discrete wavelet transform algorithm is shown in Algorithm 5.3. The inverse transform is defined similarly in Algorithm 5.4. The algorithms are essentially identical to the 2D versions, but with one extra dimension of computation. The time complexity can be derived using a similar argument to Section 5.1.1. For an $n \times n \times n$ signal, the time complexity is $O(n^3)$ (or linear in the total number of samples).

Our wavelet neural network model is easily extended to the 3D case by modifying each layer to compute one iteration of Algorithm 5.3. A visualization of the 3D forward wavelet network can be found in Figure 5.16. Like in the 2D case, a single learned filter pair is applied at each layer. An autoencoder framework can then be constructed using our standard loss term (Equation 5.11).

In the 2D wavelet transform, the detail coefficient components corresponded to one of three directions in 2D space: horizontal, vertical, and diagonal. In the 3D case there are seven detail coefficient components. Each of them also correspond to a different direction, but in 3D space. Viewing the impulse responses of filters in 3D is more difficult than in the 2D case. Figure 5.17 shows a representation of the seven impulse response directions for the Haar wavelet. We sample the space using eight equally spaced grid points, and plot

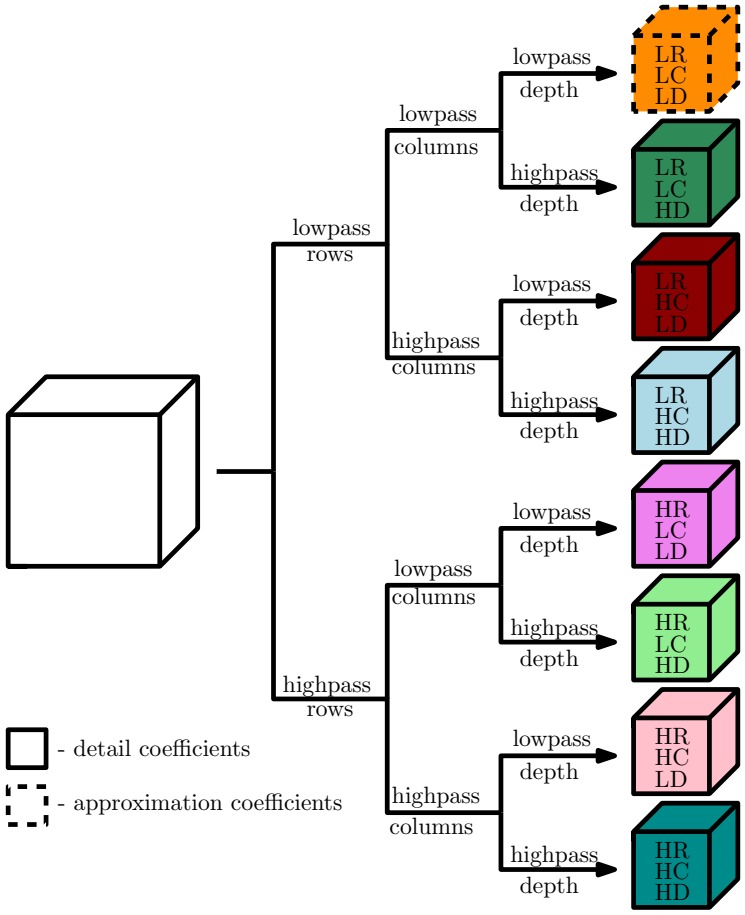
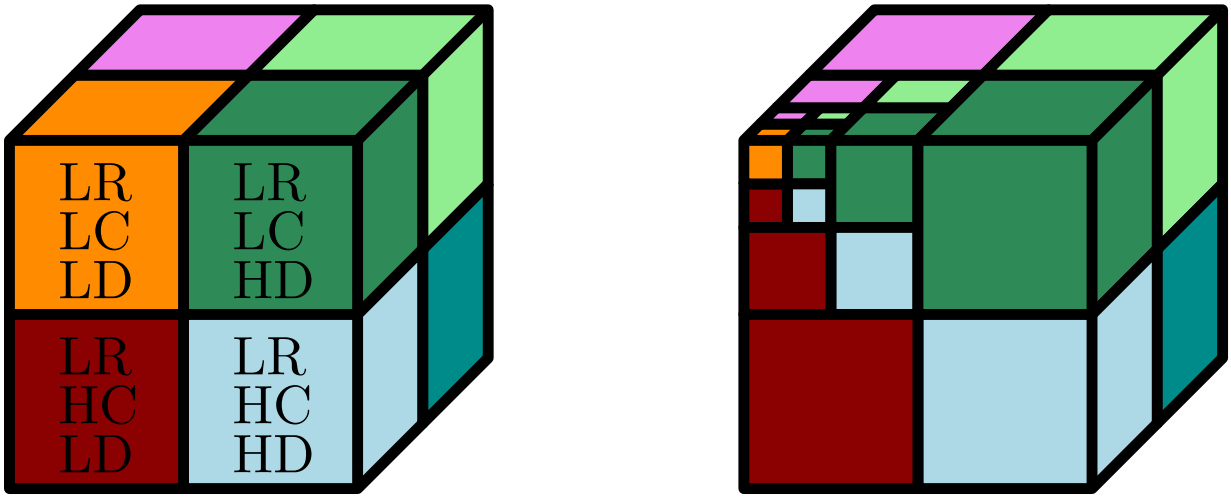


Figure 5.14: One iteration of the 3D wavelet transform. Note that there are seven detail components and one approximation component.

the sign of each point. Note that projecting the plots in Figures 5.17b, 5.17d, and 5.17f along the depth axis would recover the three directions that we saw in the 2D case.

We evaluate our 3D model by learning wavelet filters for a MRI dataset. The data is from [Schonberg et al., 2012] and can be accessed from OpenfMRI [Poldrack et al., 2013]. See Figure 5.18 for an example of 2D slices from an MRI image. In our experiments we found issues with exploding gradients (see Section A.2.4). This may be due to the extra level of convolution at each layer in the network. Gradients were clipped to $[-1, 1]$ to prevent any numerical issues. Though we have not experimented with higher dimensions, we predict exploding gradients may be an issue for any dimension greater than two. Furthermore, the model was more prone to local minima than lower dimensional models, leading to



(a) (b)

Figure 5.15: (a) A visualization of the eight components computed after a single iteration of the 3D wavelet transform. (b) The wavelet coefficient volume after three iterations of the wavelet transform algorithm.

some degenerate filters. This problem may be related to the exploding gradients problem.

Figure 5.19 shows one valid filter (top) and one degenerate filter (bottom) learned using the same parameters settings as the previous sections. The model was trained using 16 MRI scans downsampled by a factor of four due to computational limitations. The learned wavelets tended to be close to the Haar wavelet. This may be due to sharp edges in the MRI images. In particular, note how the top wavelet function in Figure 5.19 has a pronounced vertical edge. The degenerate filter, though similar to the other filter, does not lead to meaningful wavelet and scaling functions.

Algorithm 5.3 3D Discrete wavelet transform

Require: f is a discrete 3D signal, L is the maximum level of the transform.

function 3D-DWT(f, L)
  $a_0 \leftarrow f$
  for $j \leftarrow 0$ to $L-1$ **do**
    $lr, hr \leftarrow LR(a_j), HR(a_j)$
    $lclr, hclr \leftarrow LC(lr), HC(lr)$
    $lchr, hchr \leftarrow LC(hr), HC(hr)$
    $a_{j+1}, d^1_{j+1} \leftarrow LD(lclr), HD(lclr)$
    $d^2_{j+1}, d^3_{j+1} \leftarrow LD(hclr), HD(hclr)$
    $d^4_{j+1}, d^5_{j+1} \leftarrow LD(lchr), HD(lchr)$
    $d^6_{j+1}, d^7_{j+1} \leftarrow LD(hchr), HD(hchr)$
    $d_{j+1} \leftarrow \langle d^1_{j+1}, \ldots, d^7_{j+1} \rangle$
  end for
  return $\langle d_1, d_2, \ldots, d_L, a_L \rangle$
end function

Algorithm 5.4 Inverse 3D discrete wavelet transform

Require: $\langle d_1, d_2, \ldots, d_L, a_L \rangle$ as computed per Algorithm 5.3.

function 3D-IDWT($\langle d_1, d_2, \ldots, d_L, a_L \rangle$)
  for $j \leftarrow L-1$ to 0 **do**
    $lclr \leftarrow \overline{LD}(a_{j+1}) + \overline{HD}(d^1_{j+1})$
    $hclr \leftarrow \overline{LD}(d^2_{j+1}) + \overline{HD}(d^3_{j+1})$
    $lchr \leftarrow \overline{LD}(d^4_{j+1}) + \overline{HD}(d^5_{j+1})$
    $hchr \leftarrow \overline{LD}(d^6_{j+1}) + \overline{HD}(d^7_{j+1})$
    $lc \leftarrow \overline{LC}(lclr) + \overline{HC}(hclr)$
    $hc \leftarrow \overline{LC}(lchr) + \overline{HC}(hchr)$
    $a_j \leftarrow \overline{LR}(lr) + \overline{HR}(hr)$
  end for
  return a_0
end function

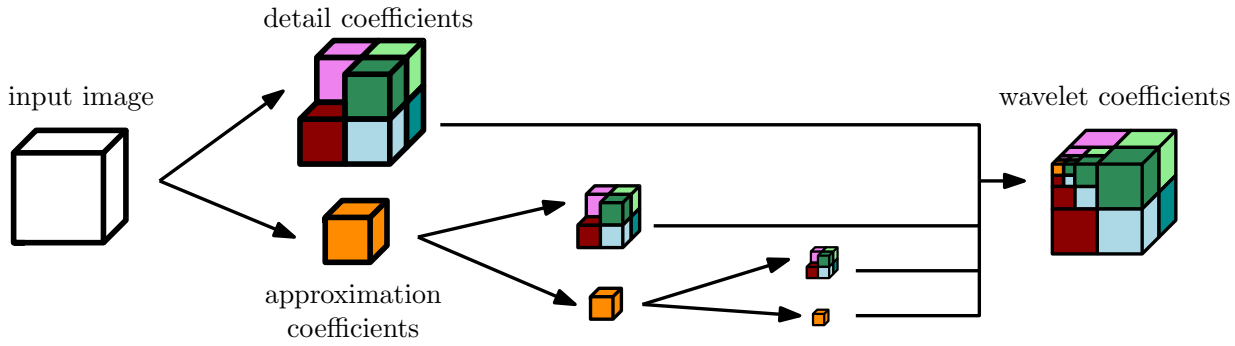


Figure 5.16: The 3D wavelet neural network architecture.

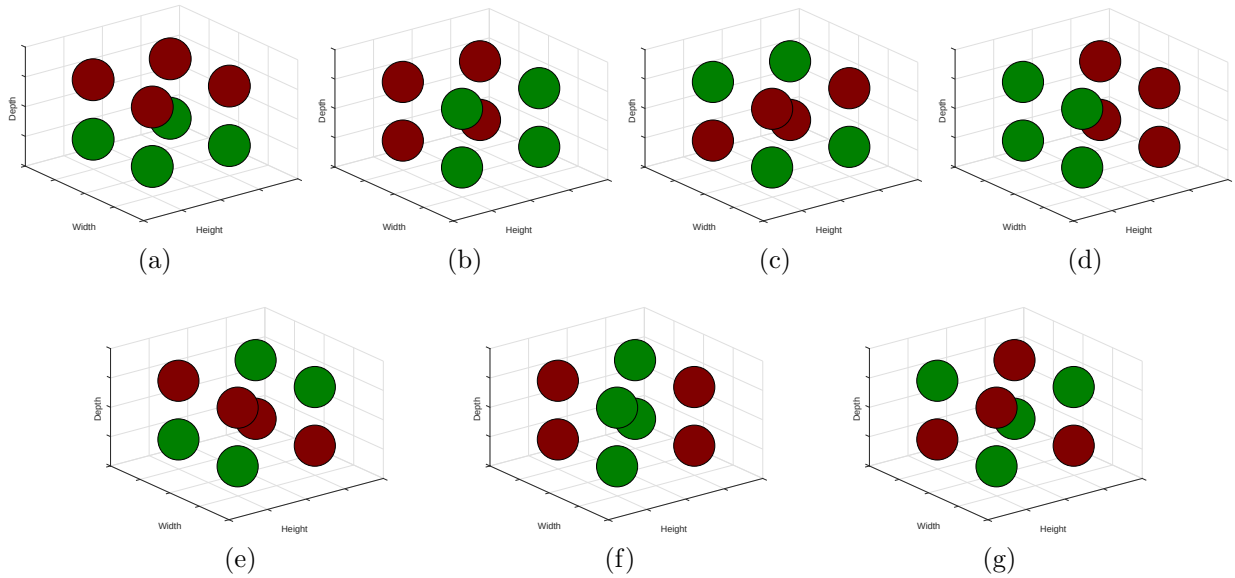


Figure 5.17: A representation of the seven directions of the impulse responses of the Haar wavelet in the 3D wavelet transform. Each point in the plot has equal magnitude. Positive values are lighter and negative values are darker.

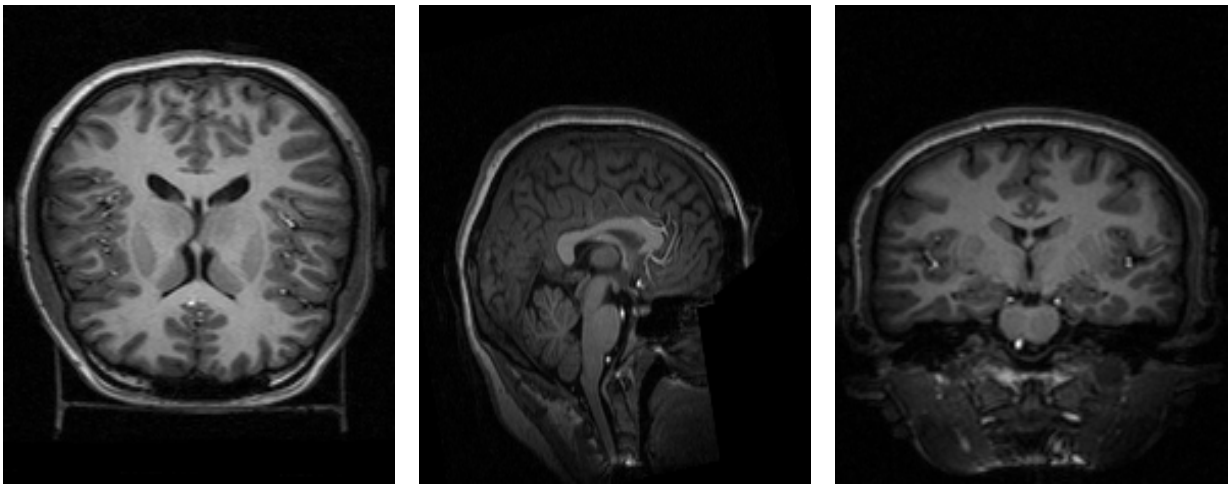


Figure 5.18: Slices of an MRI scan at three different orientations.

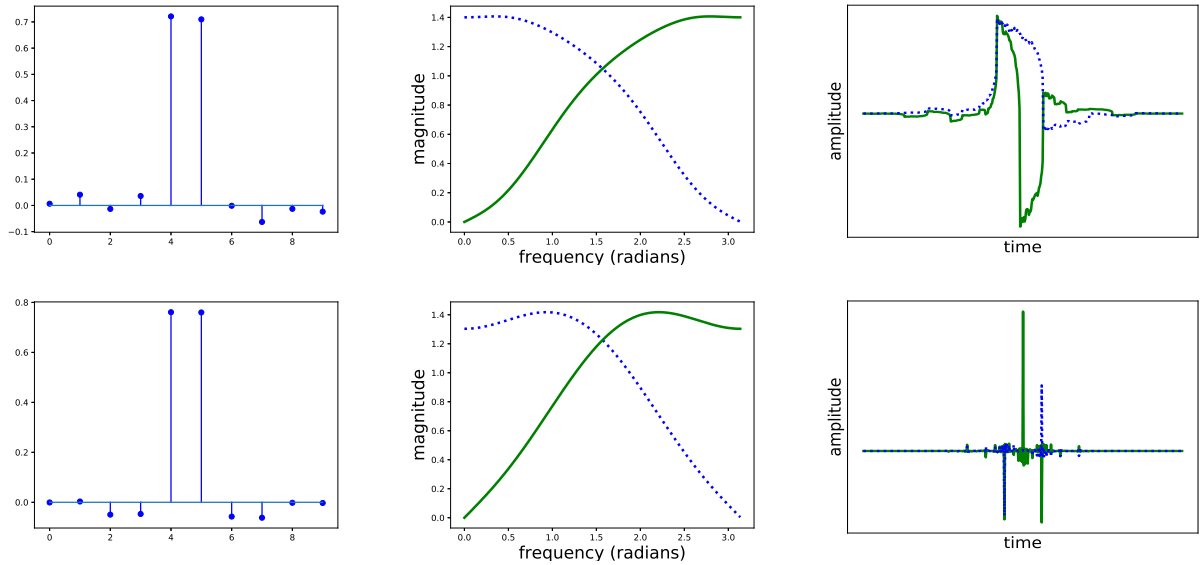


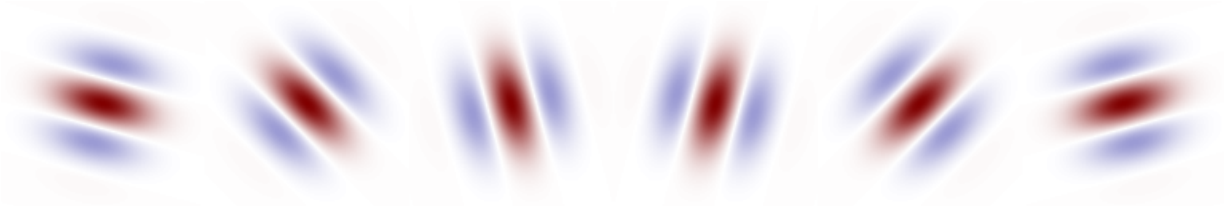
Figure 5.19: Scaling filter, filter frequency response, and wavelet (solid) and scaling (dashed) functions learned from MRI data. The bottom row shows a degenerate filter.

5.4 Conclusion

We have proposed a new model for learning wavelet filters directly from image data by extending our 1D model. We have shown that useful wavelets can be learned using an autoencoder framework with sparsity constraints. The autoencoder is comprised of a 2D discrete wavelet transform followed by an inverse transform. Preferring a sparse representation forces the model to learn wavelets that exploit structure in the training data. A reconstruction constraint means that the learned filters are (nearly) orthogonal, and so can be used for both analysis and synthesis. We frame our model as a modified CNN, making it easy to incorporate into existing neural network architectures. A benefit of this model over traditional CNNs is that we require very few parameters. This is due to two properties: 1) the filters used are one-dimensional, and 2) the filters are reused at each layer of the network. In the final section of the chapter, we demonstrated how the model could be extended to 3D, and showed some initial experiments with MRI data.

Chapter 6

The Dual-tree Wavelet Transform



In this chapter we explore learning filters for the dual-tree complex wavelet transform [Kingsbury, 1998, Selesnick et al., 2005]. This transform was introduced to address several shortcomings of the separable, real-valued wavelet transform algorithm. However, the dual-tree transform requires greater care when designing filters. The added complexity makes the transform a good candidate to replace the traditional filter derivations with learning. We demonstrate that it is possible to learn filters for the dual-tree complex wavelet transform in a similar fashion to our model from previous chapters. We will show that very few changes to our original autoencoder framework are necessary to learn filters that overcome the limitations of the separable wavelet transform. This chapter should be viewed as exploratory, and as such, we restrict our implementation to the 2D case only. 1D and multidimensional data could be treated in a similar manner, but that is left as future work.

Wavelet representations have been shown to perform well on a variety of machine learning tasks. Specifically, wavelet scattering networks have shown state-of-the-art results despite the fact they use a fixed representation (in contrast to the learned representations of CNNs) [Bruna and Mallat, 2011, Mallat, 2012, Bruna and Mallat, 2013, Mallat, 2016].

Similar work has been done using oriented bandpass filters in the SOE-Net [Hadji and Wildes, 2017]. More recently, it has been shown that extending this work using the dual-tree complex wavelet transform can lead to improved results [Singh and Kingsbury, 2017a, Singh and Kingsbury, 2017b]. In all of these examples, the filters used in the networks are fixed. Building upon our work in the previous chapters, the goal of this chapter is to demonstrate that it is possible to learn valid filters for the dual-tree complex wavelet transform for use in neural network architectures.

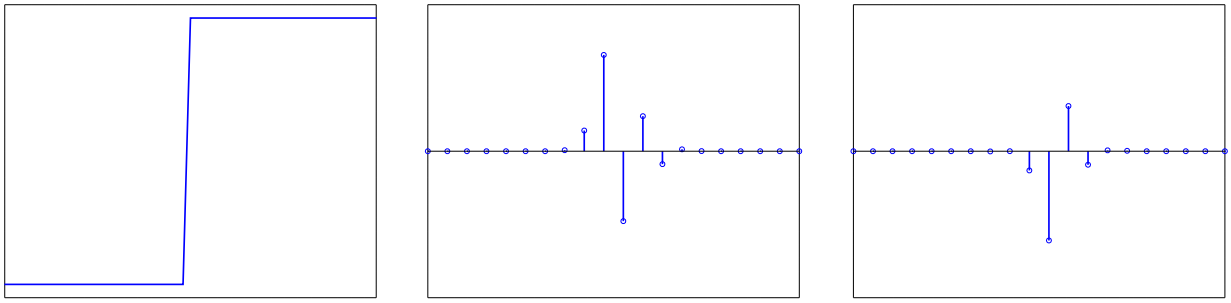
The experimental results at the end of the chapter are reminiscent of the sparse coding work of [Olshausen and Field, 1996]. They show that oriented Gabor-like filters arise when learning a sparse code of natural images. One limitation of their approach is that it can only be applied to small image patches. A hierarchical extension to their work that overcomes this limitation can be found in [Olshausen et al., 2001] (with later improvements in [Sallee and Olshausen, 2003]). In all cases, the learned basis functions were real-valued. Our model, thanks to Kingsbury's dual-tree framework, is able to learn both real and complex filters.

6.1 Problems with the Wavelet Transform

The discrete real wavelet transform has many desirable properties, such as a linear time algorithm and basis functions (wavelets) that are not fixed. However, the transform does have some drawbacks. The four major limitations that we will consider are: shift variance, oscillations, lack of directionality, and aliasing. We will see later in the chapter that complex wavelets can overcome these problems. We will restrict ourselves to a single formulation of the complex wavelet transform known as the dual-tree complex wavelet transform (DTCWT) [Kingsbury, 1998]. The DTCWT is able to overcome the limitations of the standard wavelet transform, but comes with some computational overhead. Before going into the details of the DTCWT, we will discuss the issues with the standard wavelet transform. A more detailed discussion can be found in [Selesnick et al., 2005].

6.1.1 Shift Variance

A major problem with the real-valued wavelet transform is shift variance. A desirable property for any representation is that small perturbations of the input should result only in small perturbations in the feature representation. In the case of the wavelet transform, translating the input (even by a single sample) can result in large changes to the wavelet



(a) (b) (c)

Figure 6.1: (a) A signal containing a single step edge. (b) The third level Daubechies 6 wavelet coefficients of (a). (c) Same as (b) but with the signal in (a) shifted by a single sample.

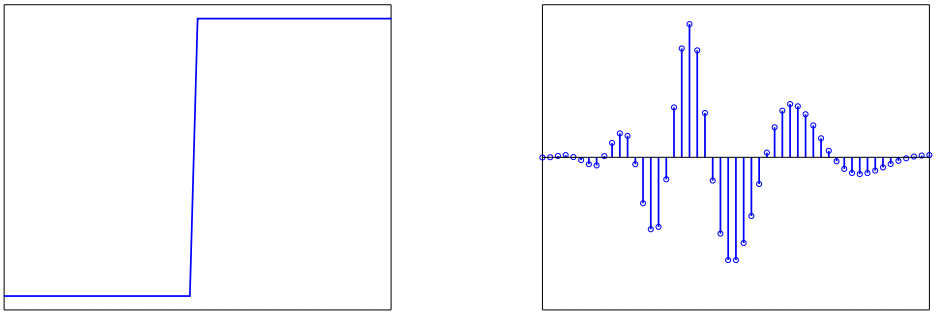
coefficients. Figure 6.1 demonstrates the issue of shift variance. We plot the third scale wavelet coefficients of a signal composed of a single step edge. The signal is shifted by a single sample, and the coefficients are recomputed. Note that the coefficient values change significantly after the shift.

6.1.2 Oscillations

Figure 6.1 also demonstrates that wavelet coefficients are not stable near signal singularities such as step edges or impulses. Note how the wavelet coefficients change in value and sign near the edge. We can see this problem is more detail by looking at the non-decimated wavelet coefficients of the step edge in Figure 6.2. The coefficients necessarily oscillate because the wavelet filter is highpass.

6.1.3 Lack of Directionality

The standard multidimensional wavelet transform algorithm makes use of separable filters. In other words, a single 1D filter is applied along each dimension of the input. Though this method lends itself to an efficient implementation of the algorithm, there are some problems in the impulse responses of the filters. Namely, we can only properly achieve horizontal and vertical directions of the filters, while the diagonal direction suffers from checkerboard artifacts. Figure 6.3 shows impulse responses of a typical wavelet filter in the 2D transform. The impulse responses demonstrate these problems.



(a) (b)

Figure 6.2: (a) A signal containing a single step edge. (b) The (undecimated) third level Daubechies 6 wavelet coeficients corresponding to the edge.

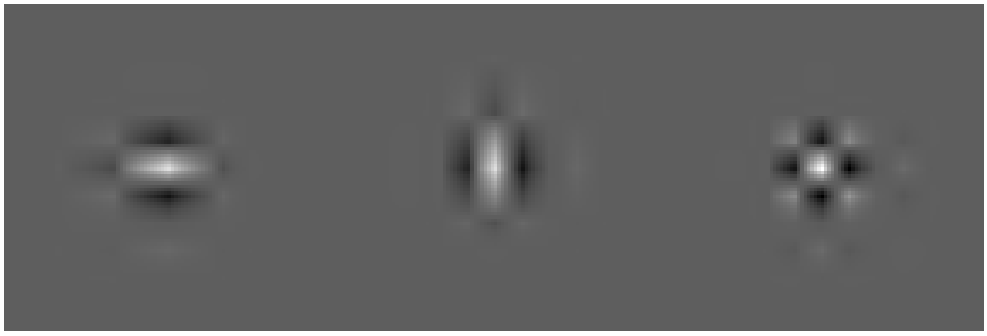


Figure 6.3: Typical impulse response of filters used in the 2D wavelet transform.

The problems caused by the lack of directionality can be seen by reconstructing simple images from wavelet coefficients at a single scale. Figure 6.4 shows two images reconstructed from their fourth scale wavelet coefficients. Note that the horizontal and vertical edges in the first image appear without any artifacts. The diagonal edges, on the other hand, have significant irregularities. The curved image illustrates this further, as almost all edges are off-axis.

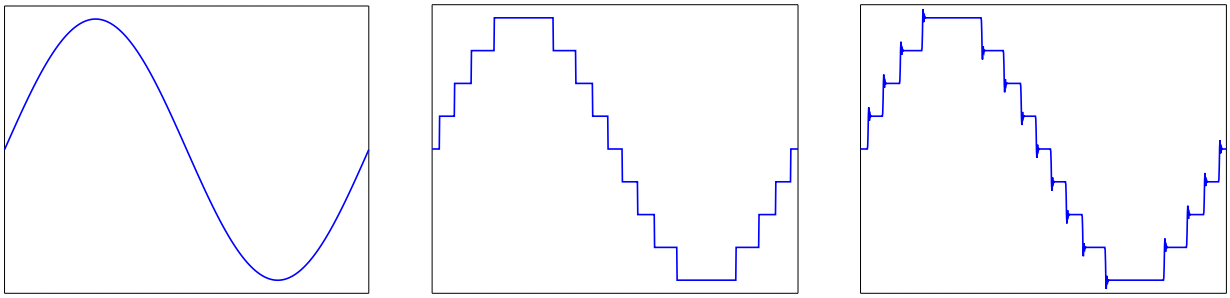
6.1.4 Aliasing

The final problem we will discuss is aliasing. The standard decimating discrete wavelet transform algorithm downsamples the wavelet coefficients by a factor of two after the wavelet filters are applied. Normally we must apply a lowpass (antialiasing) filter prior

(a) (b)

Figure 6.4: (a) Left: An image with three edge orientations. Right: Reconstruction of the image using only the fourth band Daubechies 2 wavelet coefficients. Note the irregularities on the edges that are not axis-aligned. (b) Same as (a), but using an image with a curved edge.

to downsampling a signal to prevent aliasing. Aliasing occurs when downsampling is performed on a signal that contains frequencies above the Nyquist rate. The energy of these high frequencies is reflected onto the low frequency components, causing artifacts. The wavelet transform avoids aliasing by careful construction of the wavelet filter. Thus, the original signal can be perfectly reconstructed from the downsampled wavelet coefficients. However, any changes made to the wavelet coefficients prior to performing the inverse transform (such as quantization) can cause aliasing in the reconstructed signal. Figure 6.5 shows the effects of quantization of a simple 1D signal. In the first case, quantization is performed in the sample domain, leading to step edges in the signal. In the second case, a one level wavelet transform is first applied to the signal. Quantization is then performed on the wavelet coefficients prior to reconstruction. This second method of quantization introduces artifacts that did not occur when quantizing in the sample domain.



(a) (b) (c)

Figure 6.5: (a) Original signal. (b) The signal from (a) quantized to nine levels. (c) A one level wavelet transform is first applied to (a), the wavelet coefficients are quantized to nine levels, and an inverse transform is applied. Note the artifacts near the step edges.

6.2 The Dual-tree Complex Wavelet Transform

The problems with the standard wavelet transform discussed in the previous section can be overcome with the DTCWT [Selesnick et al., 2005] (see Figures 6.6, 6.7, and 6.8). We will restrict our discussion to the 2D version of the transform. As its name suggests, the dual-tree wavelet transform makes use of multiple computational trees. The trees each use a different wavelet and scaling filter pair.

The intuition behind the transform is to compute the real and imaginary parts of a complex wavelet transform separately. Doing so allows us to make use of real valued filters, while getting the benefits of a complex transform. The filters used in each tree need to be chosen so that one is approximately a half-sample shift of the other. This can be achieved by having the filters obey the q-shift property introduced in [Kingsbury, 2000].

We will first discuss the real version of the dual-tree transform. Let us denote each filter pair by h_i and g_i, where $i = 1$ for the first tree and $i = 2$ for the second tree. The dual-tree algorithm proceeds similarly to the 2D version. Like in the 2D case, we compute four components of coefficients for each tree: $LR_i(LC_i(x))$, $LR_i(HC_i(x))$, $HR_i(LC_i(x))$, and $HR_i(HC_i(x))$, where the index corresponds to the filter index. Each tree computes its own coefficient matrix W_i. The final coefficient matrices are computed as follows:

$$W_1 \leftarrow \frac{W_1(x) + W_2(x)}{\sqrt{2}} \tag{6.1}$$

$$W_2 \leftarrow \frac{W_1(x) - W_2(x)}{\sqrt{2}} \tag{6.2}$$

We therefore have six bands of detail coefficients at each level of the transform, as opposed to three in the 2D case. Example impulse responses for real filters is shown in Figure 6.12a. Note that they are oriented along six different directions and there is no checkerboard effect. The drawback of the real dual-tree transform is that it is not approximately shift invariant [Selesnick et al., 2005].

The complex transform is similarly computed. The main difference is that there are a total of four trees instead of two (for a total of twelve detail bands). The twelve bands are of the form

$$LR_i(HC_j(x)),\ HR_i(LC_j(x)),\ HR_i(HC_j(x)) \tag{6.3}$$

for $i, j \in \{1, 2\}$. Let $W_{i,j}$ represent the three bands of detail coefficients using filter i for the rows and filter j for the columns. The final complex detail coefficients are computed as:

$$W_{1,1} \leftarrow \frac{W_{1,1}(x) + W_{2,2}(x)}{\sqrt{2}} \tag{6.4}$$

$$W_{2,2} \leftarrow \frac{W_{1,1}(x) - W_{2,2}(x)}{\sqrt{2}} \tag{6.5}$$

$$W_{1,2} \leftarrow \frac{W_{1,2}(x) + W_{2,1}(x)}{\sqrt{2}} \tag{6.6}$$

$$W_{2,1} \leftarrow \frac{W_{1,2}(x) - W_{2,1}(x)}{\sqrt{2}} \tag{6.7}$$

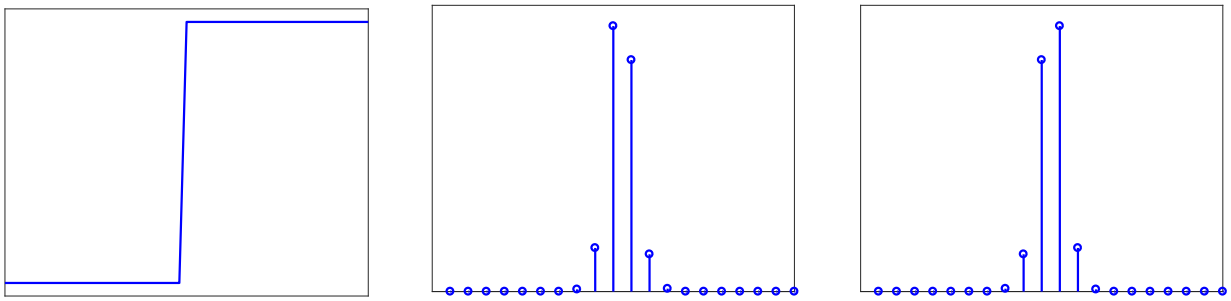
In this work we will focus on q-shift filters [Kingsbury, 2000]. That is, we restrict

$$h_2[n] = h_1[-n]. \tag{6.8}$$

In other words, the filters used in the second tree are the reverse of the filters used in the first tree. Furthermore, the filters used in the first iteration of the algorithm must be different than the filters used for the remaining iterations [Selesnick et al., 2005]. The initial filters must also obey the q-shift property, but be offset by one sample. We will denote these filters using a prime symbol (e.g., h'_i is the initial filter in the i^{th} tree). Finally, we restrict ourselves to quadrature mirror filters, i.e.,

$$g[n] = (-1)^n h[-n]. \tag{6.9}$$

Thus, we can derive the wavelet filters directly from the scaling filter by reversing it and negating alternating indices. Equations 6.8 and 6.9 mean we can completely define the transform with the filters h_1 and h'_1.

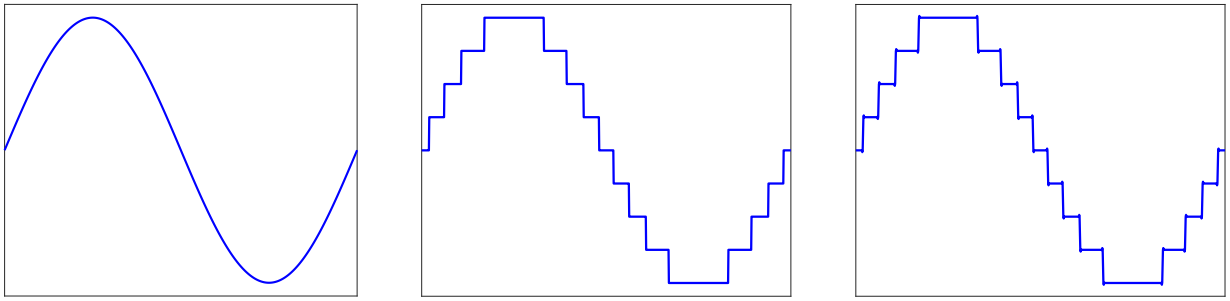


(a) (b) (c)

Figure 6.6: (a) A signal containing a single step edge. (b) The magnitude of the third level DTCWT coefficients of (a). (c) Same as (b) but with the signal in (a) shifted by a single sample.

(a) (b)

Figure 6.7: (a) Left: An image with three edge orientations. Right: Reconstruction of the image using only the fourth band DTCWT coefficients.



(a) (b) (c)

Figure 6.8: (a) Original signal. (b) The signal from (a) quantized to nine levels. (c) A one level DTCWT is first applied to (a), the wavelet coefficients are quantized to nine levels, and an inverse transform is applied. Note how the artifacts are much less pronounced.

6.3 The Dual-tree Wavelet Transform as a Neural Network

Designing filters for the dual-tree wavelet transform takes more care than designing regular wavelets. Two methods proposed by Kingsbury can be found in [Kingsbury, 2000] and [Kingsbury, 2003]. Inspired by the results from the previous chapters, we propose an alternative method of filter design based on our wavelet neural network model. The first task is to modify our previous model to fit the dual-tree framework.

The dual-tree wavelet transform is computed using two main operations: convolution and downsampling. These two operations are the same as those used in traditional CNNs. With this observation in mind, we propose framing the dual-tree wavelet transform as a modified CNN architecture. See Figures 6.9 and 6.10 for illustrations of the real and complex versions of the network respectively. This network directly implements the dual-tree wavelet transform algorithm, with each layer representing a single iteration. The output of the network are the wavelet coefficients of the input image.

To demonstrate how this model behaves, we construct an autoencoder framework [Hinton and Salakhutdinov, 2006] consisting of a dual-tree wavelet network followed by an inverse network. The inverse network is structured similarly to the forward network. The intermediate representation in the autoencoder are exactly the dual-tree wavelet coefficients of the input image. Like earlier, we will impose a sparsity constraint on the wavelet coefficients so that the model will learn filters that can summarize the structure of the training set.

The wavelet loss term from the previous chapters is reused here:

$$L_w(h,g) = \ (||h||_2^2 - 1)^2 + (\mu_h - \sqrt{2}/k)^2 + \mu_g^2 \tag{6.10}$$

where μ_h and μ_g are the means of the scaling and wavelet filters respectively. The first two terms are necessary so that the scaling function will have unit L^2 norm and finite L^1 norm. The third term requires that the wavelet filter has zero mean.

We will later demonstrate that this loss term alone is not enough to learn useful filters. Thus, in order to avoid degenerate filters, we will require an additional constraint. The degenerate filters lack localization and good orientation, and so we propose a loss that prefers impulse responses that are close to Gaussian. Let M^k be the matrix corresponding to the magnitude of the impulse response for the k^{th} band (i.e. the sum of the squares of the real and imaginary impulse responses). And let G be a Gaussian matrix of the form:

$$G_{i,j} = \alpha e^{-[(i-i_o)^2+(j-j_0)^2]/(2\sigma^2)} \tag{6.11}$$

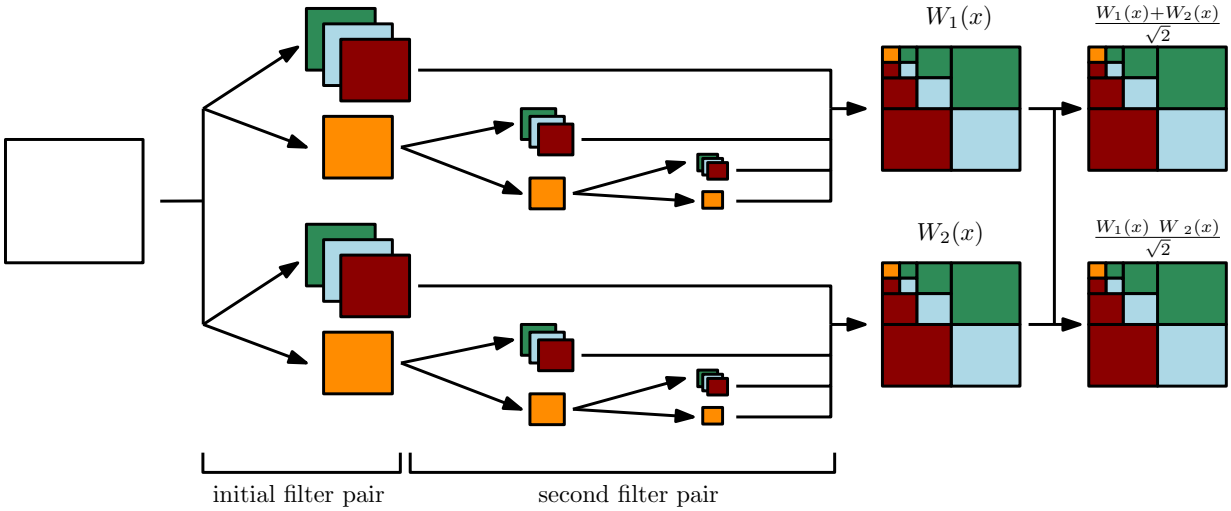


Figure 6.9: The real dual-tree wavelet transform network. Each W_1 and W_2 are computed as in Equations 6.1 and 6.2.

with i_0 and j_0 chosen to center the Gaussian in the matrix. The new loss term is of the form

$$L_g = \frac{1}{m} \sum_{k=1}^{6} ||G - M^k||_2^2 \tag{6.12}$$

where m is the number of elements in M^k.

We will learn the network parameters using gradient descent, and so must define a loss function over a dataset of images $X = \{x_1, x_2, \ldots x_M\}$:

$$\begin{aligned} L(X; h_1, h_1') = & \frac{1}{MN} \sum_{k=1}^{M} ||x_k - \hat{x}_k||_2^2 \\ & + \lambda_1 \frac{1}{MN} \sum_{k=1}^{M} \sum_{i,j} ||W_{i,j}(x_k)||_1 \\ & + \lambda_2 [L_w(h_1, g_1) + L_w(h_1', g_1')] + \lambda_3 L_g \end{aligned} \tag{6.13}$$

where $\hat{x}_k$ is the reconstructed image. We use squared error for our reconstruction loss and the ℓ_1 norm for the sparsity penalty. The λ parameters control the trade-off between the loss terms. For the real valued network, the index j is removed from the sparsity summation.

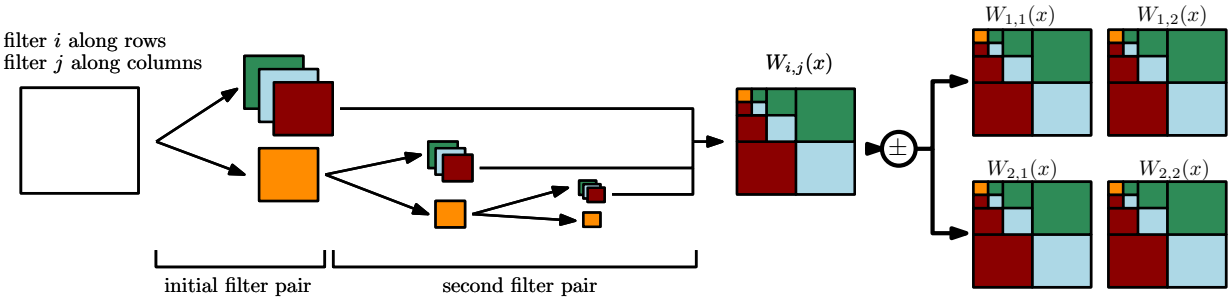


Figure 6.10: The complex dual-tree wavelet transform network. Each $W_{i,j}$ is computed as in Equations 6.4–6.7.

6.4 Experiments

In this section we show that oriented filters can arise naturally from data. The learned filters are compared to the q-shift filters of [Kingsbury, 2000, Kingsbury, 2003]. In the first set of experiments we consider synthetic data composed of random sinusoids. We demonstrate the feasibility of our model, as well as the importance of the Gaussian impulse constraint (Equation 6.12). The second set of experiments demonstrate that similar filters arise from natural images.

6.4.1 Synthetic Data

We first make use of synthetic images to demonstrate that our model is able to learn filters with localized directional structure. The generation process is the same one used in Chapter 5, but with a restriction to sine waves:

$$x(t) = \sum_{k=0}^{K-1} a_k \cdot sin(2^k t + \phi_k) \tag{6.14}$$

where $\phi_k \in [0, 2\pi]$ and $a_k \in \{0, 1\}$ are chosen uniformly at random. In Equation 6.14, ϕ_k is a phase offset and a_k is an indicator variable that determines whether a particular harmonic is present. We fix the size of the images to be 128×128. Figure 6.11 shows some example synthetic images.

We set $\lambda_1 = .1$, $\lambda_2 = 1$, $\lambda_3 = $ 4e-5 (real network), and $\lambda_3 = $ 4e-4 (complex network). For the L_g cost, we compute the impulse response at the fourth scale and set $\alpha = .02$ and $\sigma = 10$ (found empirically). We train using stochastic gradient descent using the Adam

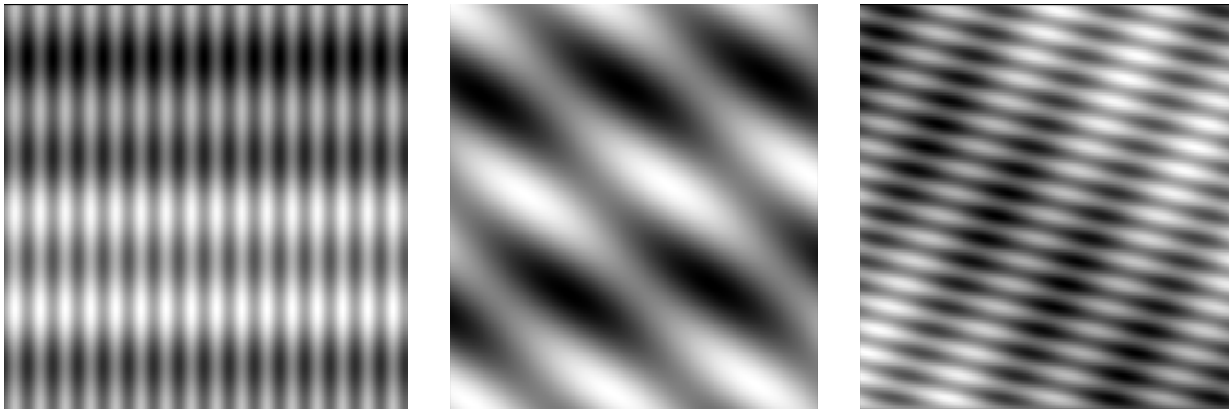


Figure 6.11: Examples of synthetic images.

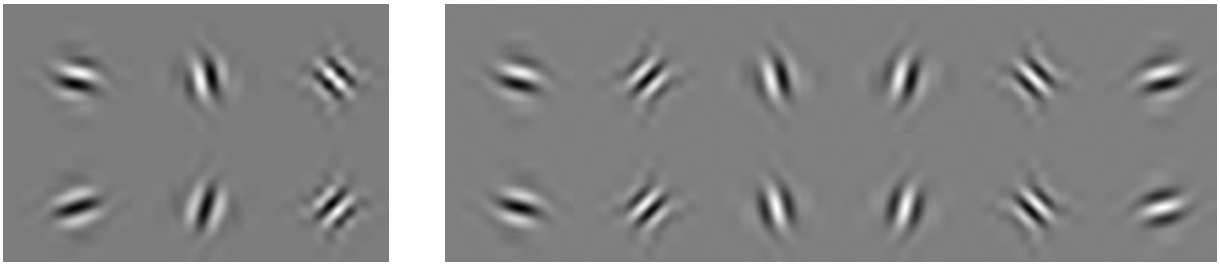
algorithm [Kingma and Ba, 2014]. Our implementation is done using Tensorflow [Abadi et al., 2015].

Figure 6.12 shows the impulse responses of the filters learned by our real and complex models. In this case, all filters were of length ten. We can see that the model is able to learn localized, directional filters from the data. Figure 6.13 shows a comparison of the learned h filter from Figure 6.12b to Kingsbury's q-shift filters. Note that h has a very similar structure. The shifted cosine distance measure from previous chapters is used, and is shown above each plot. See Appendix D for more filters and plots.

The Gaussian loss term is important for learning directional filters. Figure 6.14 shows a selection of filters learned without the impulse response loss term (i.e. $\lambda_3 = 0$ in Equation 6.13). We can see that the filters are no longer directional, and share similar structure to the impulse responses from Figure 6.3. Analysing these "degenerate" filters is left for future work.

6.4.2 Natural Images

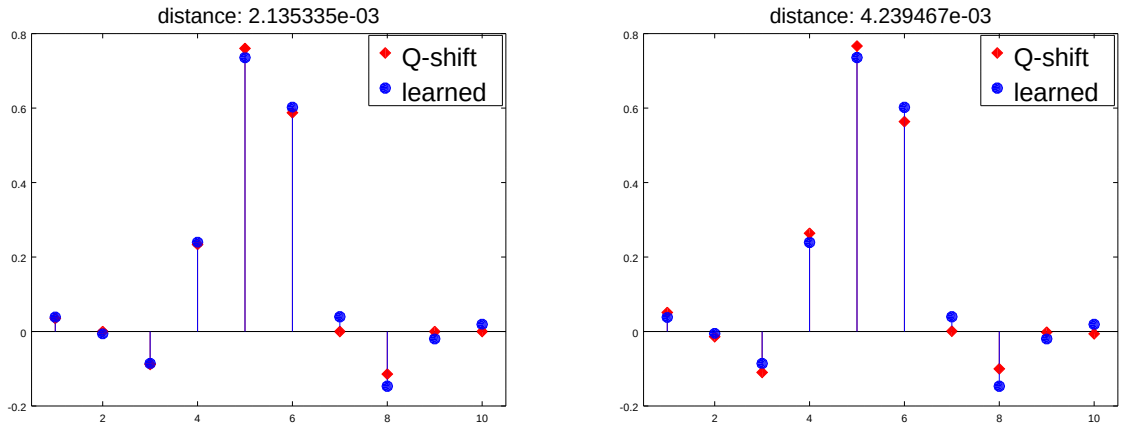
Up to now we have shown that oriented filters arise from a particular class of synthetic images (random sinusoids). We now demonstrate that similar filters also arise from natural images. The images are taken from the Kodak Lossless True Color Image Suite [Franzen, 1999] and can be found in Appendix B. We repeat the experiments from the previous section using the natural image dataset. A selection of results are summarized in Figures 6.15 and 6.16.



(a) Real transform (b) Complex transform

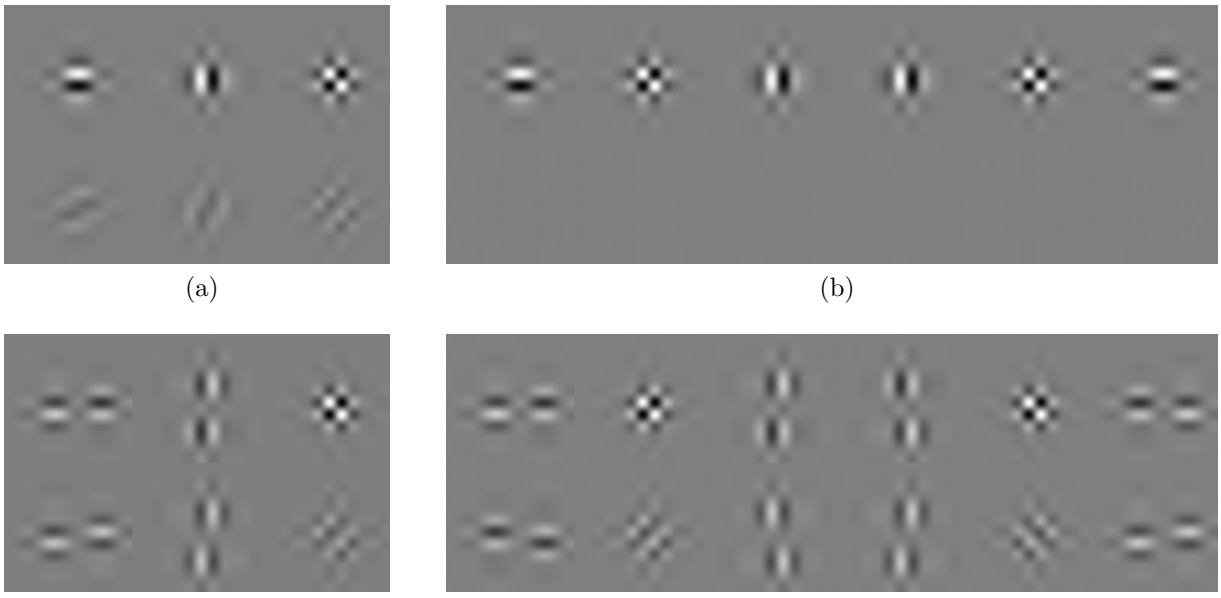
Figure 6.12: Example impulse responses of filters learned using our (a) real and (b) complex model. The top and bottom rows of (b) correspond to the real and complex parts of the filter responses respectively.

Just as in the synthetic case, the learned filters have impulse responses that are localized and oriented. Figure 6.15 shows that the structure of the filters appear to be invariant to filter length. Indeed, the impulse responses from various learned filter lengths are almost qualitatively imperceptible. Furthermore, as shown in Figure 6.15, the learned filters are very similar to filters derived using Kingsbury's design methods [Kingsbury, 2000, Kingsbury, 2003]. In particular, the initial fall off in their frequency responses are very similar. Further analysis of our dual-tree model is left as future work.



(a) (b)

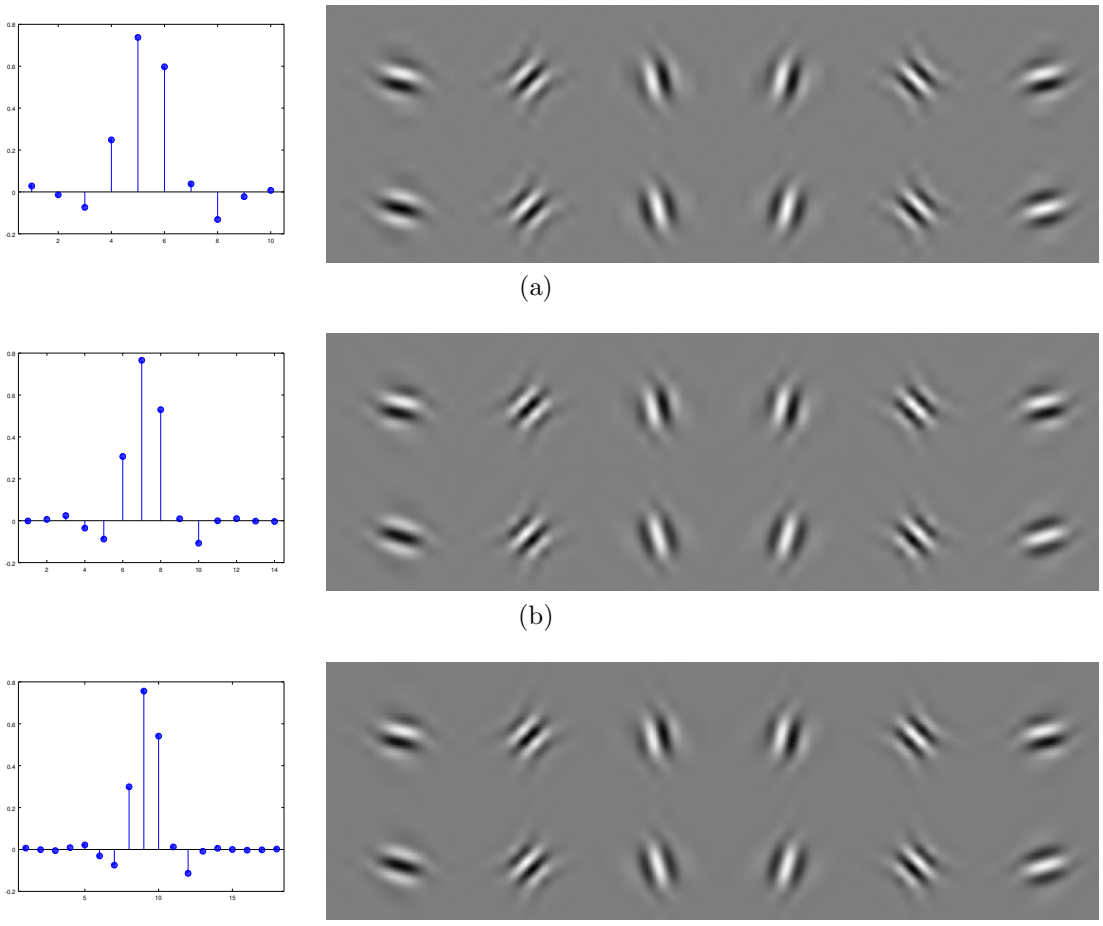
Figure 6.13: Comparison of Kingsbury's q-shift (a) 6 tap, and (b) 10 tap filters with the learned h filter from Figure 6.12b. Note that h has been reversed.



(a) (b)

(c) (d)

Figure 6.14: Example impulse responses of filters learned using our (a,c) real and (b,d) complex model without the impulse response loss term (i.e. $\lambda_3 = 0$).



(a)

(b)

(c)

Figure 6.15: Example learned filters with impulse responses using our complex model on natural images with varying filter lengths: (a) 10, (b) 14, and (c) 18.

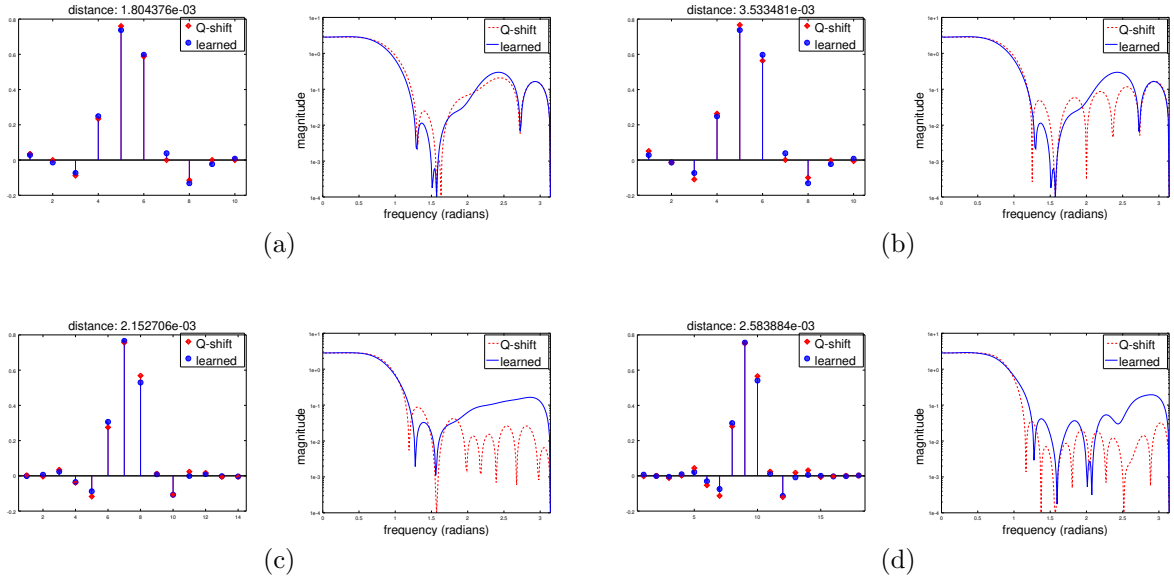


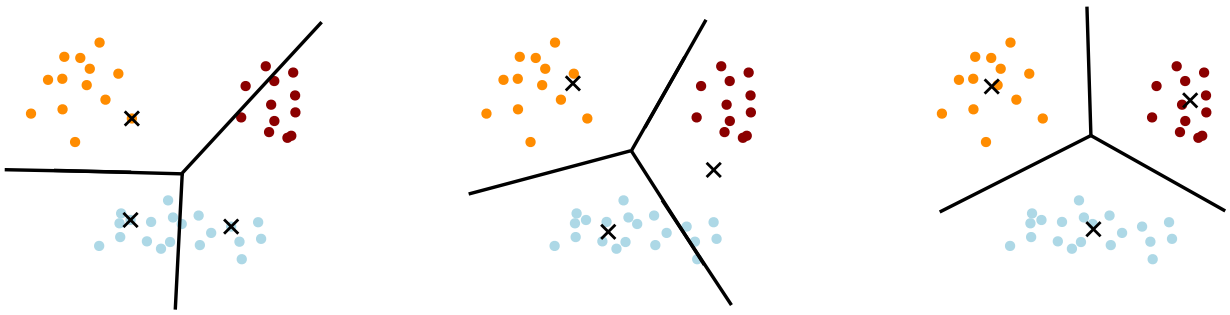
Figure 6.16: A comparison of the learned filters from Figure 6.15 with Kingsbury's q-shift filters [Kingsbury, 2000, Kingsbury, 2003]. Filter lengths: (a) 10 (6-tap q-shift), (b) 10, (c) 14, and (d) 18. Both filter values and frequency responses are shown.

6.5 Conclusion

In this chapter we presented a model based on the dual-tree complex wavelet transform that is capable of learning directional filters that overcome the limitations of the standard 2D wavelet transform. We made use of an autoencoder framework, and constrained our loss function to favour orthogonal wavelet filters that produced sparse representations. This work builds upon our previous work on the 1D and 2D wavelet transform, and introduces a locality constraint in order to avoid degenerate filters. We propose this method as an alternative to the filter design methods proposed in [Kingsbury, 2000] and [Kingsbury, 2003].

Chapter 7

Applications and Future Work



This chapter will explore some applications of our learning model, as well as some avenues of future work. The purpose of this chapter is not to demonstrate state-of-the-art results in any particular task. Instead, the purpose is to demonstrate that our wavelet learning model can be easily adapted to different tasks. We begin with the task of signal classification, and propose a novel wavelet classification algorithm. The algorithm is evaluated on a dataset of bird songs. We then show how we can adapt the classification algorithm to perform clustering of unlabeled data. Next, we consider the tasks of image compression and classification, and compare our model to standard methods. Finally, we discuss how we might extend our model to learn biorthogonal wavelets and perform signal generation.

7.1 Audio Classification and Extensions

This section will explore the use of our learning method for the task of signal classification. Suppose that we have a set of signals from discrete classes. We will denote this set

$$X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_M, y_M)\} \tag{7.1}$$

where $1 \leq y_i \leq k$ is the integer label of signal x_i. Hence, our dataset is composed of M signals with k possible class labels. How may we use our wavelet model to classify each signal? Recall that for a set of signals, our model learns a wavelet and scaling filter pair that allows for both a sparse representation and an accurate reconstruction. With this in mind, we propose a simple multiclass classification method as follows:

1. Partition X into k subsets according to the labels.
2. For each subset, learn a scaling and wavelet filter pair that minimizes Equation 4.14.
3. Given a signal of unknown label, choose the class corresponding to the filter pair that gives the best fit.

The algorithm first learns a separate filter pair for each class of signals according to our autoencoder loss function. This portion of the classification algorithm is summarized in Algorithm 7.1. At this point we have learned the parameters of the wavelet classifier: $\{(g_j, h_j)\}_{1 \leq i \leq k}$. The classification rule from step three relies on the following argument: the filter pair (g_i, h_i) should provide the best data fit for all signals from class i. Formally,

$$L(x; g_i, h_i) = \min_{g,h} L(x; g, h) \tag{7.2}$$

where x is from class i. Note that we will set $\lambda_2 = 0$ in L in order to focus only on the data fitting terms in the loss function (i.e. we remove the wavelet constraints). Given this observation, a signal can be classified by choosing the class that minimizes L:

$$\arg\min_i L(x; g_i, h_i) \tag{7.3}$$

Our classification procedure shares similarities to sparse dictionary classification methods, particularly those that take a reconstructive approach. Generally, these methods proceed as follows [Mairal et al., 2008a]: (1) Learn a set of dictionaries $\{D_1, D_2, \ldots, D_k\}$ (one for each class). (2) Given an input x, choose the class according to

$$\arg\min_i \left[\min_a R(x, D_i, a)\right] \tag{7.4}$$

Algorithm 7.1 Fit the parameters of a wavelet classifier.

Require: $X = \{(x_i, y_i)\}$ is a set of labeled signals.

```
function WaveletFit(X)
    for j ← 1 to k do
        X_j ← {x_i | y_i = j}
        (g_j, h_j) ← arg min_{g,h} L(X_j; g, h)
    end for
    return {(g_j, h_j)}_{1≤i≤k}
end function
```

where R is a reconstruction error function, often some variation of

$$R(x, D_i, a) = ||x - D_i a||_2^2 \tag{7.5}$$

as in [Yang et al., 2007].

We can derive our wavelet classification method with few modifications. We first note that a scaling and wavelet filter pair define a multiscale dictionary. Therefore, learning a separate pair of filters for each class of signals is equivalent to learning a set of dictionaries. Second, the sparse code in our model (i.e. the wavelet coefficients) is a function of the input. This means we can ignore the inner minimization in Equation 7.4, which becomes

$$\arg\min_i R(x, D_i, a) \tag{7.6}$$

It is now clear that the major difference between our method and reconstructive based sparse dictionary classification is that we incorporate a sparsity penalty into the classification rule. That is, we modify Equation 7.6 to

$$\arg\min_i R(x, D_i, a) + \lambda_1 ||a||_1 \tag{7.7}$$

which is equivalent to Equation 7.3 when $\lambda_2 = 0$ (i.e. no wavelet constraints).

7.1.1 Application to Bird Songs

The wavelet classification algorithm will be tested on a dataset of bird songs. The songs are taken from the Xeno-canto Foundation collection [Xeno-canto Foundation, 2004]. The dataset that we use here contains the songs of the following bird species: corn bunting,

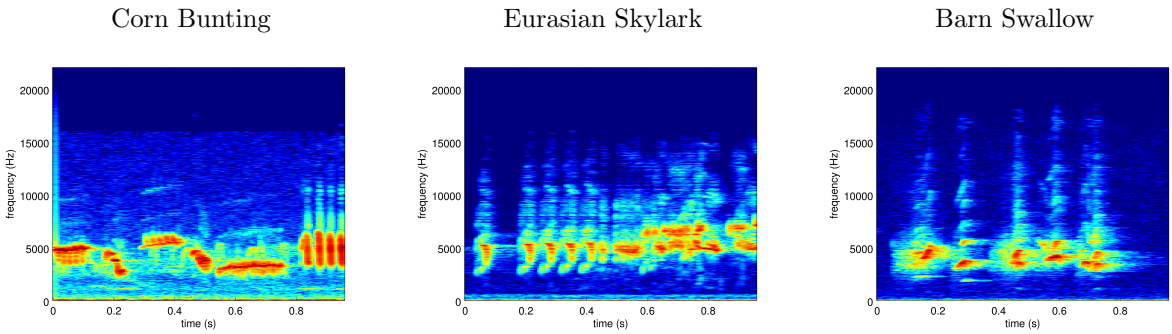


Figure 7.1: Bird song spectrums from three different species.

Eurasian skylark, barn swallow, sedge warbler, and common nightingale. These species were chosen because they each had samples recorded by the same individual. This is a desirable property, since we may assume similar recording conditions between samples. Otherwise, a classifier may learn to classify signals based on features of the recording setup and not the audio content. Each species of bird has three different audio recordings of approximately five minutes in duration. Sample spectrograms of three species are included in Figure 7.1. Note that each species has different spectral structure.

An assumption of our wavelet classification algorithm is that wavelets learned for each class will find structure unique to the class. Figure 7.2 shows wavelet and scaling functions (and corresponding scaling filters) learned from each the three different species of birds. Though the functions have similar structure, there are some differences that will allow for our classification algorithm to differentiate between classes.

In order to apply our wavelet classification algorithm, we must first construct a dataset from the bird recordings. We can see in Figure 7.1 that the majority of spectral energy is below 5000Hz. Since the original files are sampled at 44100Hz, we decimate each signal by a factor of four. This is done for efficiency purposes. Next, each signal is separated into subsignals of 2^{18} samples ($\sim$24 seconds). After this preprocessing, we are left with the following number of signals per species

- corn bunting: 30 signals
- Eurasian skylark: 24 signals
- barn swallow: 21 signals

Decimation is done using SciPy's decimation function with default parameters.

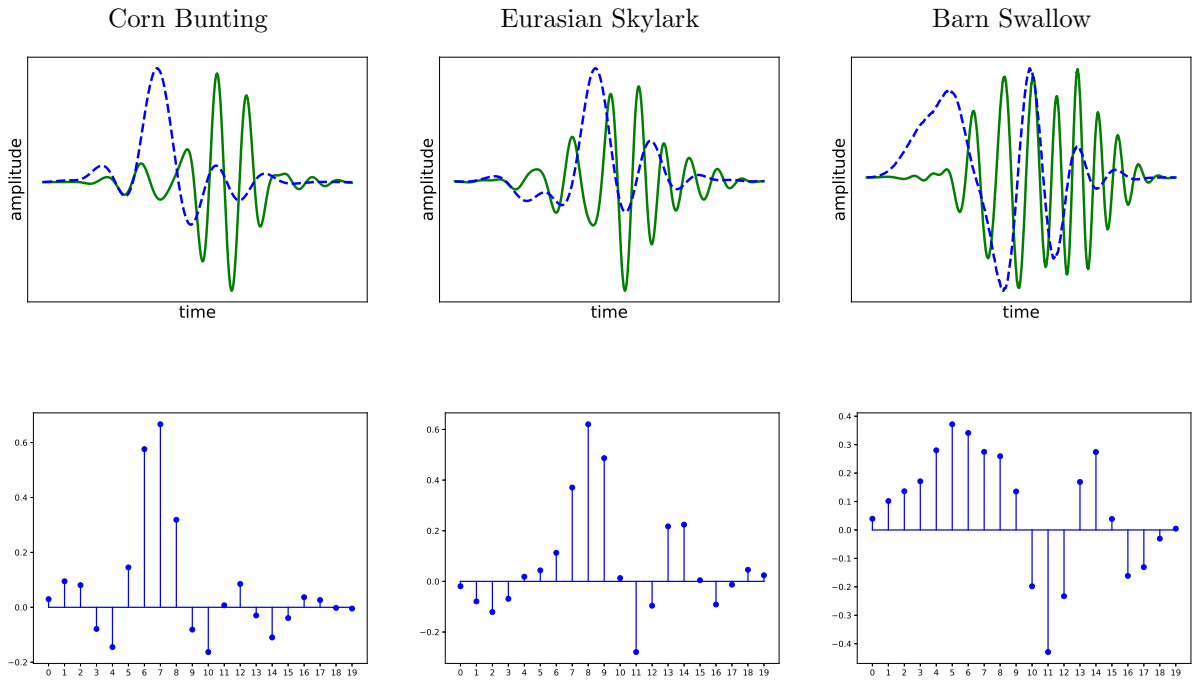


Figure 7.2: Top: Wavelet (solid) and scaling (dashed) functions learned from the bird songs. Bottom: Corresponding scaling filters.

- sedge warbler: 20 signals
- common nightingale: 20 signals

Our algorithm will be tested on subsets of the data containing different numbers of species. We begin with the corn bunting and Eurasion skylark, and add species one at a time in the order they are listed above. In all cases we learn filters of length 20, and set $\lambda_1 = \lambda_2 = 1/2$ in the learning phase. We make use of k-fold cross-validation to get a good estimate of the performance of our algorithm. We use $k = 6$ in the two species case, and $k = 5$ otherwise. These k's were chosen because they evenly divided the datasets. The accuracies are summarized in Table 7.1. The classification algorithm performs quite well for two species ($\sim$98% accuracy), but the performance begins to degrade as more species are considered. The results also suggest that the wavelet classification algorithm is not prone to overfitting, as the training and test accuracies are similar.

Number of Species	Training Accuracy	Test Accuracy
2	0.9667 ± 0.0204	0.9722 ± 0.0884
3	0.8867 ± 0.0575	0.8800 ± 0.1079
4	0.7711 ± 0.0780	0.7474 ± 0.1417
5	0.6978 ± 0.0989	0.7043 ± 0.0887

Table 7.1: Accuracies (with 95% confidence intervals) for the bird song classification experiments.

7.1.2 Probabilistic Interpretation

The signal classifier proposed in Section 7.1 was justified by a data fitting argument. Here, we will provide a probabilistic interpretation and justification. Suppose we have a signal, x, that we wish to classify. In a probabilistic framework, a simple classification rule is to choose the class with the highest probability given our data, i.e.

$$c^* = \arg\max_c P(c \,|\, x) \tag{7.8}$$

Note that by Bayes rule we have,

$$P(c \,|\, x) \propto P(x \,|\, c)P(c) \tag{7.9}$$

Thus, we must consider two terms: the likelihood of the data given the class, and the prior probability of the class. If we assume a uniform prior over the classes, then we can ignore $P(c)$ in Equation 7.9. We are left with defining $P(x \,|\, c)$ for each class. In our model, the likelihood of the data is simply the probability that the signal would be generated given the filters associated with a particular class. That is,

$$P(x \,|\, c) = P(x \,|\, \theta_c) \tag{7.10}$$

where θ_c represents the wavelet and scaling filters that best represent class c. Substituting Equations 7.9 and 7.10 into Equation 7.8 yields

$$\begin{aligned} c^* &= \arg\max_c P(x \,|\, c) \\ &= \arg\max_c P(x \,|\, \theta_c) \\ &= \arg\max_c \log P(x \,|\, W(x), \theta_c) + \log P(W(x)) \\ &= \arg\min_c ||x - \hat{x}||_2^2 + \lambda_1 ||W(x)||_1 \end{aligned} \tag{7.11}$$

which is exactly the classification rule from Equation 7.3. Therefore, our classification method can be interpreted as choosing the class with the highest probability given the data (assuming a uniform class prior).

7.1.3 Clustering

One extension of our wavelet classification algorithm is clustering. Suppose we have a dataset of signals with no labels, but with a known number of classes. In our classification algorithm, each class is summarized by a single wavelet and scaling filter pair. We might view this filter pair as being analogous to the centroid of the class. With this intuition in mind, we propose an adaptation to our classification method analogous to K-means clustering.

The standard K-means algorithm (sometimes called Lloyd's algorithm) consists of the following two iterative steps [Jain, 2010]:

1. Assign each data point to a cluster based on the current mean estimates

$$\arg\min_k ||\mu_k - x||_2^2 \tag{7.12}$$

2. Update the mean estimates by setting them to the means of each of the current clusters

$$\mu_k = \frac{1}{|X_k|} \sum_{x \in X_k} x \tag{7.13}$$

These two steps are repeated until convergence. The initial cluster means are often initialized to random data points.

We can adapt the wavelet classification algorithm similarly:

1. Assign each data point to a cluster based on

$$\arg\min_k L(x; g_k, h_k) \tag{7.14}$$

2. For each cluster, choose the filters that minimize the wavelet loss function L,

$$(g_k, h_k) = \arg\min_{(g,h)} L(X_k; g, h) \tag{7.15}$$

Again, these two steps may be repeated until convergence. The initial filters can be initialized randomly.

Figure 7.3 shows the learned wavelets found by our clustering algorithm on a synthetic dataset. Each signal in the dataset was generated by randomly sampling sparse wavelet

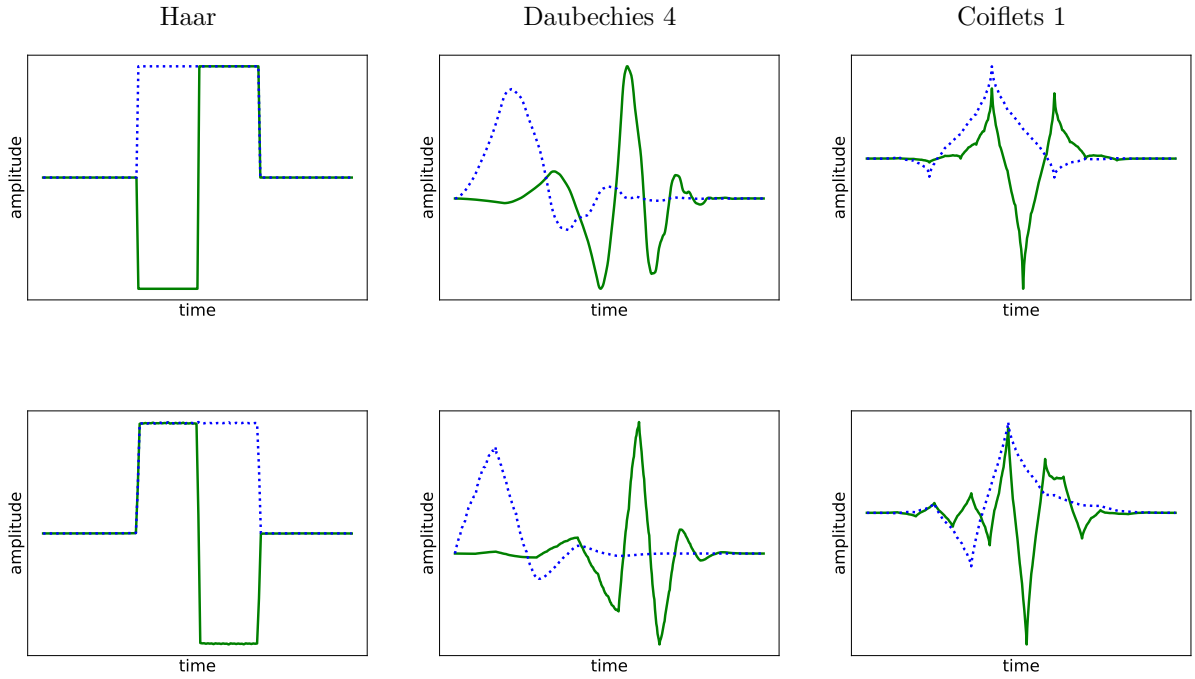


Figure 7.3: Top: Three wavelets that were used to generate a signal dataset. Bottom: The three wavelets found by our wavelet clustering algorithm.

coefficients from a Laplacian distribution, and performing an inverse transform with one of the wavelets in the top row of Figure 7.3. The algorithm recovered wavelets close to the Haar and Daubechies 4 wavelets, but struggled with the Coiflet 1 wavelet. This may be due to the hard decision rule in Equation 7.14 incorrectly clustering some of the signals. Further evaluation of the wavelet clustering algorithm is left for future work.

As we saw in Section A.1.2, a natural extension of traditional clustering is "soft" clustering where we assign cluster probabilities to each of the signals. We now briefly discuss how we might extend our wavelet clustering method to this domain using a mixture model and the EM algorithm. A mixture model is defined as (See Section A.1.2)

$$P(x) = \sum_{k=1}^{K} \alpha_k P(x \mid \theta_k) \tag{7.16}$$

where α_k are the mixing proportions, θ_k are distribution parameters, and $P(X \mid \theta_k)$ is the k^{th} distribution. Details of the exact form of $P(x \mid \theta_k)$ in our model can be found in Section

4.5. Our goal is to estimate the parameters $\{(\alpha_k, \theta_k)\}_{1 \leq k \leq K}$ for each distribution. Following the derivation in [Gupta and Chen, 2010], the expectation step estimates the probabilities that each signal belongs to a particular cluster. We can compute this as

Expectation step:

$$\gamma_{i,j} = \frac{\alpha_j^{(t)} P(x_i \mid \theta_j^{(t)})}{\sum_{k=1}^{K} \alpha_k^{(t)} P(x_i \mid \theta_k^{(t)})} \tag{7.17}$$

where $\gamma_{i,j}$ is the probability that signal x_i belongs to cluster j, and $\alpha_j^{(t)}$ and $\theta_j^{(t)}$ are the current parameter estimates.

In the maximization step, our goal is to find mixing proportions and the distribution parameters that best fit our current soft assignments $\gamma_{i,j}$. We can estimate the mixing proportions directly from the soft assignments by summing over the soft assignments. We are left with updating each filter pair. One method is to apply an iteration of gradient descent to our autoencoder loss function weighted by each $\gamma_{i,j}$. Thus, we can estimate the parameters at iteration $t+1$ as

Maximization step:

$$\alpha_j^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} \gamma_{i,j} \tag{7.18}$$

$$h_j^{(t+1)}, g_j^{(t+1)} = \arg\min_{h,g} \sum_{i=1}^{n} \gamma_{i,j} L(x_i; h, g) \tag{7.19}$$

We have yet to show that the steps above satisfy the conditions of the EM algorithm. We leave this as future work.

7.2 Image Applications

In this section we focus on two vision based applications: image compression and object classification. The purpose of this section is to demonstrate that our wavelet model can be applied relatively easily to existing tasks. However, we do not mean this chapter to be an exhaustive look into all possible vision application domains. Our two applications are chosen to show the flexibility of our model. Image compression and object classification are two very different tasks. The tasks represent two main areas of machine learning: supervised learning (object classification) and unsupervised learning (image compression). We will begin our discussion with image compression.

7.2.1 Image Compression

Image compression is an important task in computer vision. Different approaches are generally separated into lossless and lossy compression. Each approach has the same general goal: we wish to minimize the number of bits required to represent an image. However, lossless and lossy compression differ in their constraints. Lossless compression is concerned with perfect image reconstruction. Lossy compression, on the other hand, allows for some error in the reconstructed image. This allows for lossy compression techniques to achieve a better compression rate than lossless approaches.

Different image compression pipelines generally follow the same main steps. The first step is to apply some transform to the image. The goal is to transform the image into a new representation that is easier to compress than the original pixel space. This new representation is then quantized and compressed using some compression scheme. We will focus only on the first step in the process (i.e. the feature transform).

One of the most common compression methods is JPEG [Wallace, 1992]. The original JPEG standard makes use of a block-wise discrete cosine transform for its feature representation. The more recent JPEG2000 standard makes use of the wavelet transform [Christopoulos et al., 2000]. The particular wavelet used is the biorthogonal 4,4 wavelet, as it generally provides excellent compression results [Villasenor et al., 1995]. This wavelet is also called the Cohen-Daubechies-Feauveau 9/7 wavelet. Note that is not orthogonal and so the analysis and synthesis filters are different. Despite this difference, the JPEG2000 standard (particularly the biorthogonal wavelet used) provides an excellent benchmark for our method.

Since we are concerned only with the performance of learned wavelets to the fixed JPEG2000 wavelet, we will use a simplification to the full JPEG2000 compression scheme (similar to [Olshausen et al., 2001]). After performing a wavelet transform of the input image, we will use the embedded zerotree wavelet (EZW) algorithm to compress the wavelet coefficients. The EZW algorithm uses an iterative compression technique that allows for variable compression levels. This will allow us to compare wavelet compression performance at different bits per pixel (BPP). Consider a tree whose nodes correspond to the wavelet coefficients. A node's parents are the coefficients at the next scale that correspond to the same spatial location in the image. The lowest frequency scale is the root of the tree, and the highest frequency coefficients are the leaves. The EZW algorithm exploits the fact that there will often be large subtrees where the coefficients are close to zero. Details of the algorithm, as well as a review of other wavelet based image compression techniques, can be found in [Walker and Nguyen, 2001].

We will compare the compression performance of the biorthogonal 4,4 wavelet to that of

wavelets learned from our model on a variety of images. The evaluation metric we will use is peak signal-to-noise ratio (PSNR), which is commonly used to measure the performance of image compression techniques. For eight bit greyscale images, PSNR is defined as

$$\text{PSNR}(x, \hat{x}) = 10 \cdot \log_{10} \frac{255^2}{\text{MSE}(x, \hat{x})} \tag{7.20}$$

where x and $\hat{x}$ are the original and compressed images respectively, and MSE is the mean squared error. The higher the PSNR, the more alike the original and reconstructed image.

Figure 7.4 shows plots of PSNR versus the amount of compression used (bits per pixel) for three common test images. In each case, a length ten filter was learned using our wavelet model from Chapter 5 with $\lambda_1 = 1/10$ and $\lambda_2 = 1/2$. These parameter settings were chosen as they performed well across different images. We see that the learned wavelets perform similarly to the biorthogonal 4,4 wavelet. This result is in line with the empirical results from [Olshausen et al., 2001], which may indicate that the biorthogonal 4,4 wavelet basis is close to optimal for natural image compression. See Appendix B for more compression results performed on images from the Kodak Lossless True Color Image Suite [Franzen, 1999]. Note that there are some images where the learned wavelets outperform the biorthogonal 4,4 wavelet over a range of BPP.

Figure 7.6 shows a selection of images pairs with similar PSNR values, but large BPP differences. Each image sees a BPP improvement between 22% and 50% with no qualitative differences to image quality. Values are summarized in Table 7.2. These results suggest that there may be classes of images where the biorthogonal 4,4 wavelet is far from being optimal, allowing our learned wavelets to improve compression results.

To further explore this idea, we repeat our experiments on three unique images: a fingerprint, a brick texture [Wozniak, 2009], and a chest x-ray [Wang et al., 2017] (See Figure 7.5). The PSNR curve for the fingerprint image shows that the learned wavelet performs nearly identical to the biorthogonal 4,4 wavelet. This result may be surprising at first, since one would expect a learned wavelet to be able to exploit the unique structure in fingerprints. However, the biorthogonal 4,4 wavelet is empirically well suited for compressing fingerprints, even being adopted by the FBI to compress their fingerprint database [Brislawn et al., 1996]. The second image is of a brick wall, and was chosen as it has repeating and axis-aligned structure. Again, we see the learned wavelet performs similarly to the biorthogonal 4,4 wavelet. It is possible the learned wavelet does not perform better because it cannot exploit the symmetric structure of the image, unlike biorthogonal wavelets. The last image in Figure 7.5 is a chest x-ray. Here, we see the learned wavelet is able to achieve better compression than the biorthogonal 4,4 wavelet for higher bitrates, indicating that

Image	BPP	BPP (learned)	change	PSNR	PSNR (learned)	change
door	4.13	3.20	-22.59%	48.74	48.95	+0.43%
hats	4.34	2.50	-42.46%	45.57	44.65	-2.02%
sails 1	4.31	3.23	-25.11%	43.55	44.44	+2.03%
sails 2	4.25	3.09	-27.36%	44.13	43.75	-0.85%
water	4.68	2.36	-49.55%	43.97	44.28	+0.70%
x-ray	4.02	2.89	-28.12%	43.20	43.93	+1.68%

Table 7.2: PSNR and BPP values for the image pairs in Figure 7.6.

x-rays may a good candidate for learned wavelet compression. Further experiments on different classes of images is left as future work.

7.2.2 Image Classification

The second task we will consider is object classification. This is a supervised learning task in which each image is associated with a particular discrete label. We will consider four datasets in our experiments. The datasets are chosen because they each have unique structures and challenges. Each dataset comes with a predefined training and test set meant for evaluation, which we use in our experiments. The datasets are briefly described below:

- `convex`: The convex polygon dataset is composed of black and white images of simple polygons [University of Montreal, 2007a]. Each image is labelled as either containing convex or concave polygons.
- `rectangles/rectangles-im`: The rectangles dataset is composed of images of axis aligned rectangles [University of Montreal, 2007b]. The goal is to determine if the rectangle present in the image has a greater width than height, or vice versa. The dataset contains two type of images: (1) black and white images of rectangles, and (2) images where the background and rectangular regions are sampled from random images.
- `MNIST`: The MNIST (Modified National Institute of Standards and Technology) dataset is composed of grayscale images of handwritten digits [LeCun et al., 1998]. MNIST is a widely used benchmark for classification systems.

- `CIFAR-10`: The CIFAR-10 (Canadian Institute for Advanced Research) dataset is composed of colour images from ten different object classes (airplanes, birds, cars, cats, deer, dogs, frogs, horses, ships, and trucks) [Krizhevsky and Hinton, 2009]. Like MNIST, CIFAR-10 is a commonly used benchmark dataset for object classification. We convert each image to greyscale prior to processing.

The purpose of this task is to measure the effectiveness of using a learned wavelet for a feature representation. We are not concerned with achieving state-of-the-art performance on the datasets. Rather, we are interested in comparing the learned wavelet representation to a traditional CNN representation. The different representations considered are:

- `raw`: Raw pixel data (baseline).
- `wavelet`: Learned wavelet representation with length six filters.
- `cnn-same-param`: Learned CNN representation with a single 3×3 filter.
- `cnn-same-comp`: Learned CNN representation with four 6×6 filters per layer.

The raw pixel data representation is included as a baseline for the other feature representations. Two different CNN architectures will also be considered, acting as approximate lower and upper bounds on the performance of the learned wavelet representation. The first representation is a standard CNN with approximately the same number of learned parameters as the wavelet neural network. This corresponds to a single layer CNN with a single filter. The second CNN architecture is one that does approximately the same amount of computation as our wavelet network. Since the 2D wavelet transform has four effective filters per level, the CNN architecture will have four filters per layer, with the number of layers equaling the number of wavelet levels computed in the wavelet transform. Both CNN architectures will use max pooling and rectified linear activation functions after every layer.

We will compare the performance of the different feature representations by using them as input to a neural network classifier. We use a simple neural network structure to simplify training the networks. The neural network classifier has two hidden layers with 16 nodes, followed by a final layer with the same number of nodes as classes. The activation function of the hidden layers is chosen to be the rectified linear function, and batch normalization is used at each layer [Ioffe and Szegedy, 2015]. Finally, we use cross entropy as our classification loss.

Model / Dataset	convex	rectangles	rectangles-im	MNIST	CIFAR-10
raw	0.7426	0.9400	0.7637	0.9617	0.3676
cnn-same-comp	0.9406	0.9997	0.9078	0.9878	0.5387
cnn-same-param	0.7841	0.9800	0.7631	0.9629	0.3927
wavelet	0.8315	0.9992	0.7895	0.9764	0.4944

Table 7.3: Highest test accuracies achieved for each model and dataset.

We train each model using the Adam gradient descent algorithm with a batch size of 32 [Kingma and Ba, 2014]. In all trials we terminate training after 100000 batches. The test accuracies (best of four runs) are shown in Figure 7.8. The highest test accuracies achieved for the different models are summarized in Table 7.3. In all cases, the `cnn-same-comp` model performs best. This was expected as it has the most parameters. Our `wavelet` model is the second best performing model, in some cases achieving a test accuracy within 1% of the `cnn-same-comp` model. Both the `cnn-same-param` and `raw` models perform comparably, with the exception of the rectangles dataset.

Our results suggest that our wavelet model is much more parameter efficient than standard CNNs. However, restricting the filters to be wavelets results in lower accuracies than the `cnn-same-comp` model. Allowing our wavelet model to learn different filters at each layer of the network may improve performance (at a cost of increasing the number of parameters). Another possible extension would be to learn multiple filter pairs at each layer. Though this would result in a redundant representation, each filter pair may be able to adapt to different structure in the image. Exploring these modifications is left as future work.

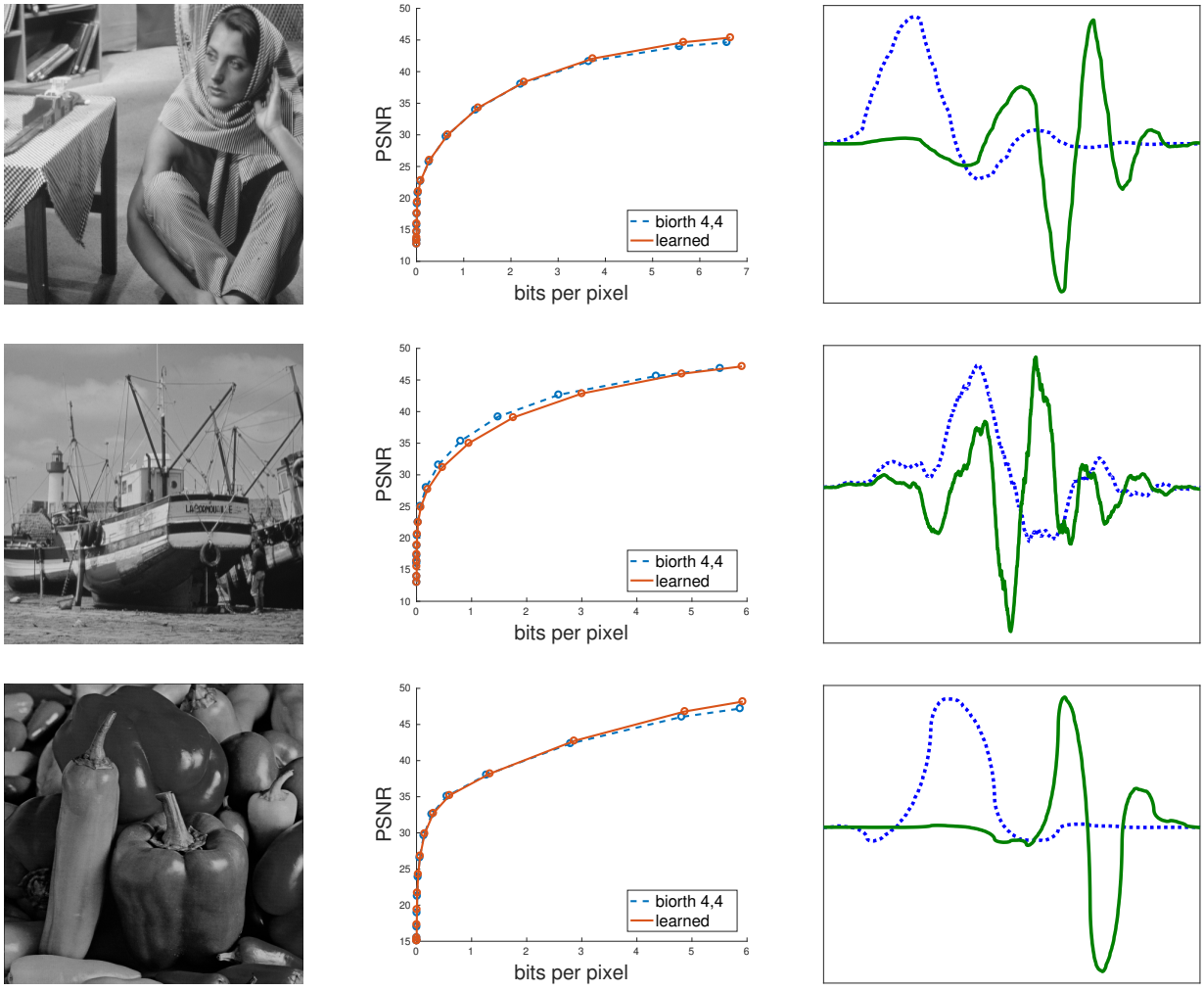


Figure 7.4: Compression results (PSNR curves) for three commonly used test images. The corresponding learned wavelet and scaling function are shown to the right.

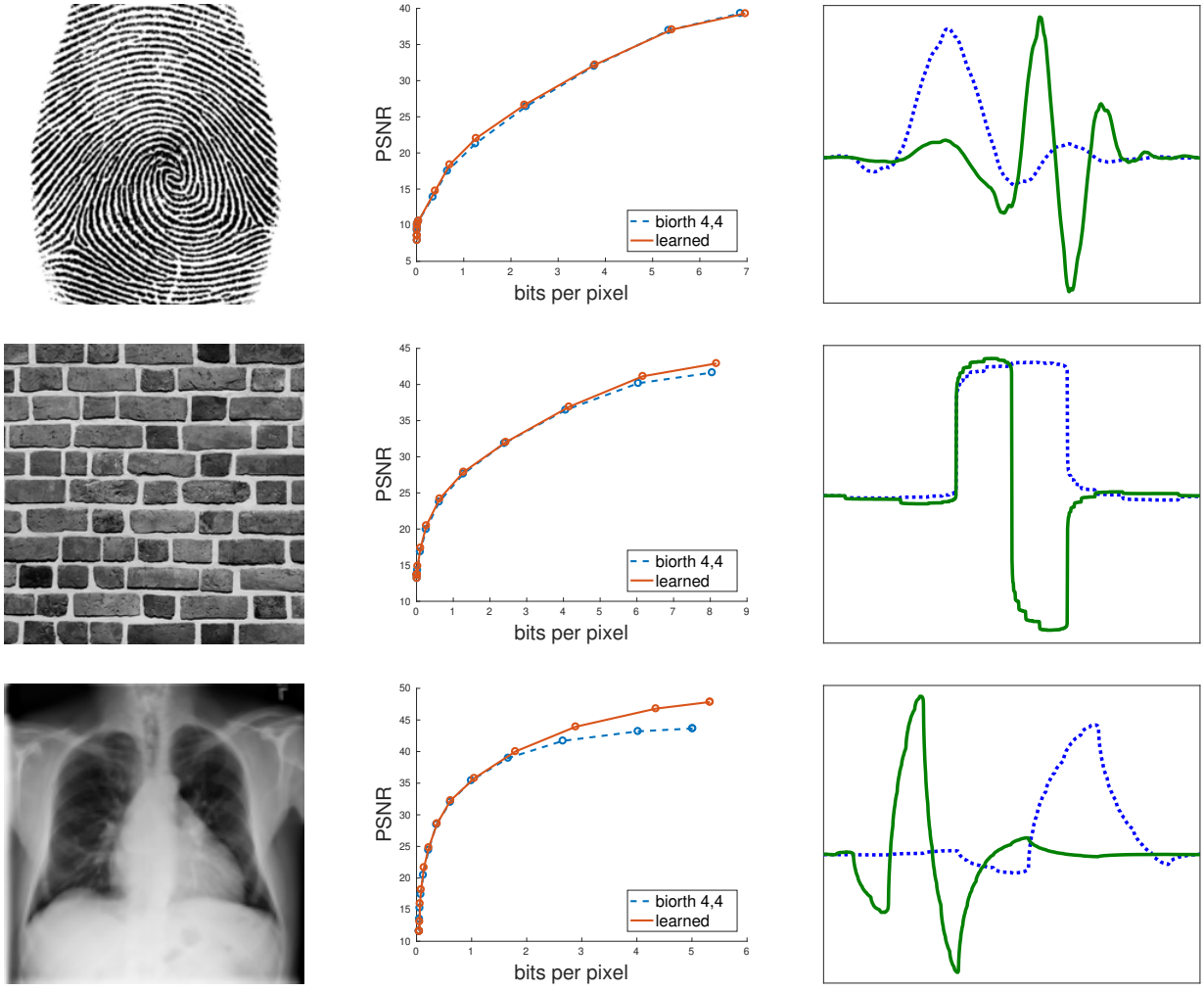
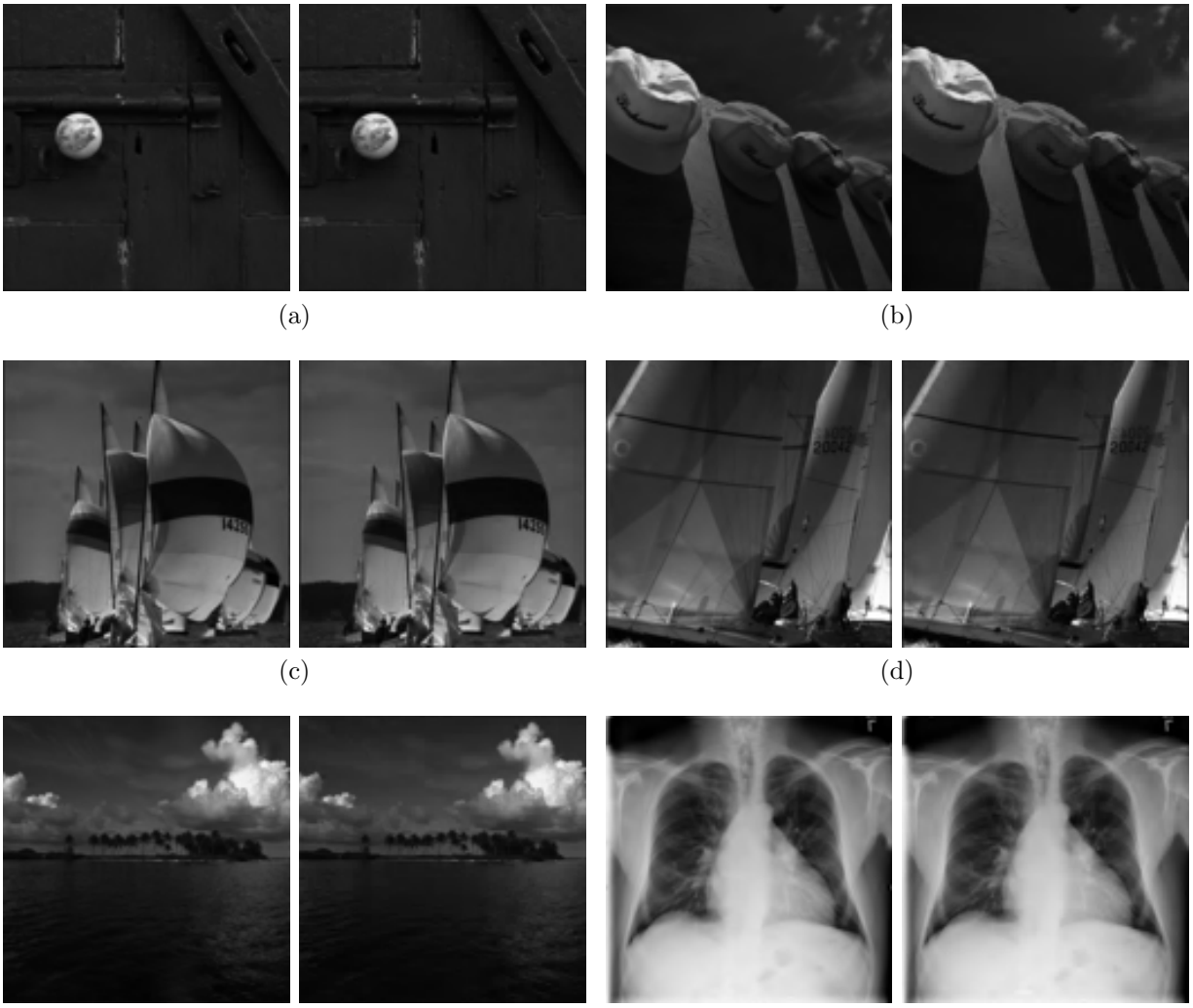


Figure 7.5: Compression results (PSNR curves) for three unique images. The corresponding learned wavelet and scaling function are shown to the right.



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

Figure 7.6: Image pairs with similar PSNR values, but large BPP differences. Left: biorthogonal 4,4 wavelet. Right: learned wavelet. See Table 7.2 for PSNR and BPP values.

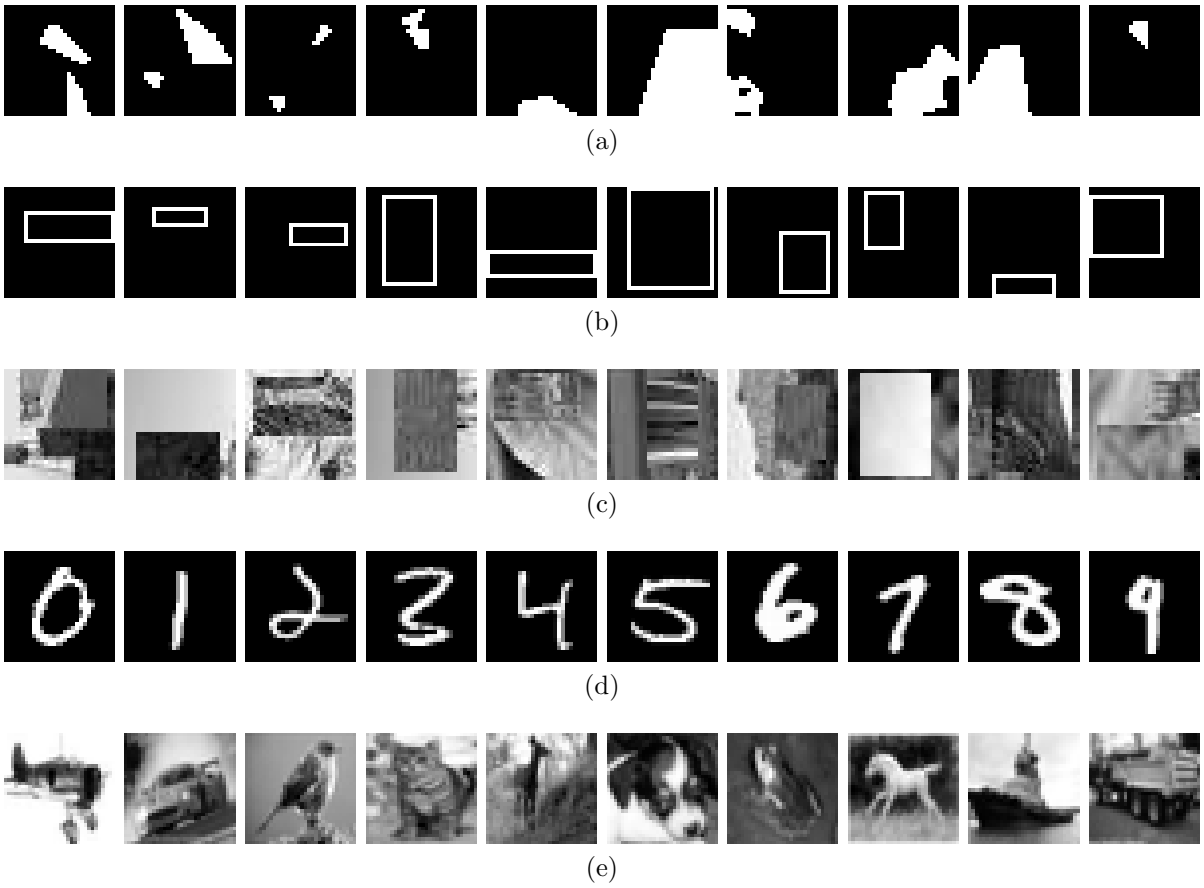
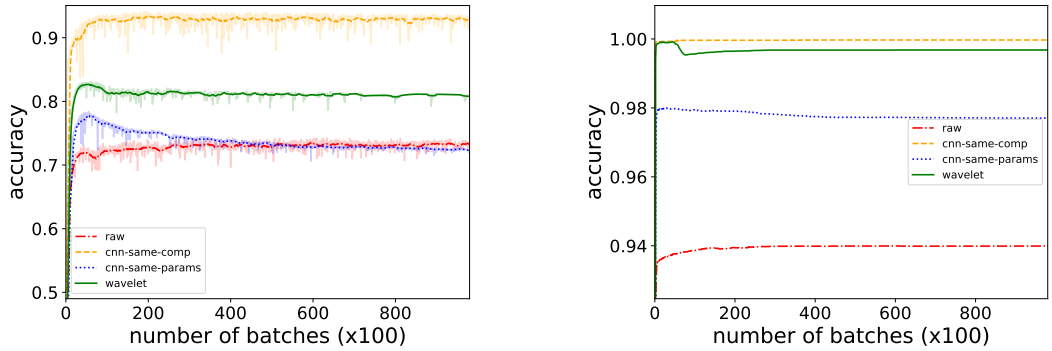
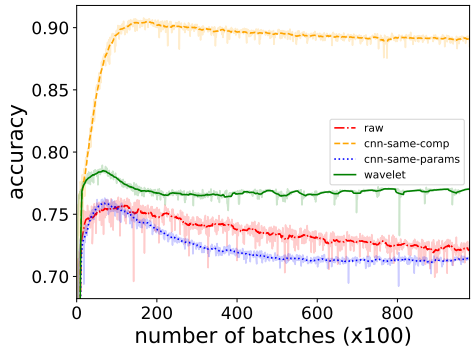


Figure 7.7: Sample images from (a) convex, (b) rectangles, (c) rectangles-im, (d) MNIST, and (e) CIFAR-10.

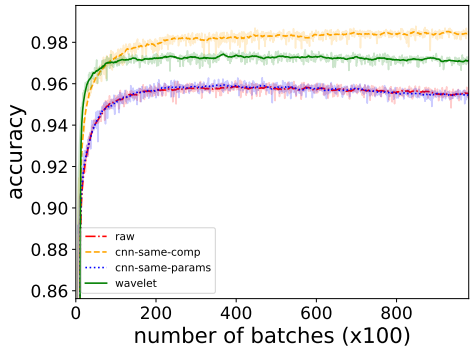


(a)

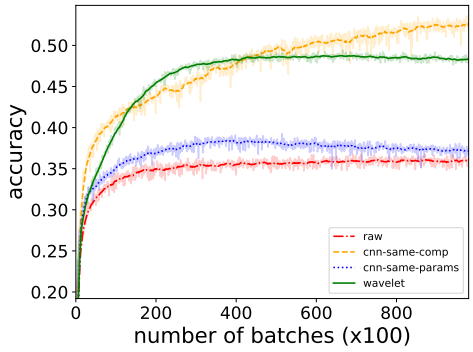
(b)



(c)



(d)



(e)

Figure 7.8: Test accuracies for the following datasets: (a) convex, (b) rectangles, (c) rectangles-im, (d) MNIST, and (e) CIFAR-10. Each plot shows the best accuracies out of four independent runs of each model. A 20 point moving average is used to smooth the plots. Actual values are shown in lighter colours.

7.3 Model Extensions

In this section we discuss two possible extensions to our wavelet model. We first consider relaxing the orthogonality constraint, allowing our model to learn biothogonal wavelets. Initial experimental results are shown. We then discuss how we might modify our model to allow for better signal generation.

7.3.1 Relaxing Constraints

Up to now we have restricted ourselves to orthogonal wavelets. In other words, the analysis and synthesis filters are the same. What happens if we relax the orthogonality constraint, and instead consider biorthogonal wavelets? Biorthogonal wavelets use a different filter pair for analysis and synthesis, but still guarantee perfect reconstruction. Our wavelet model can incorporate this change by learning two filter pairs instead of one. The loss function becomes

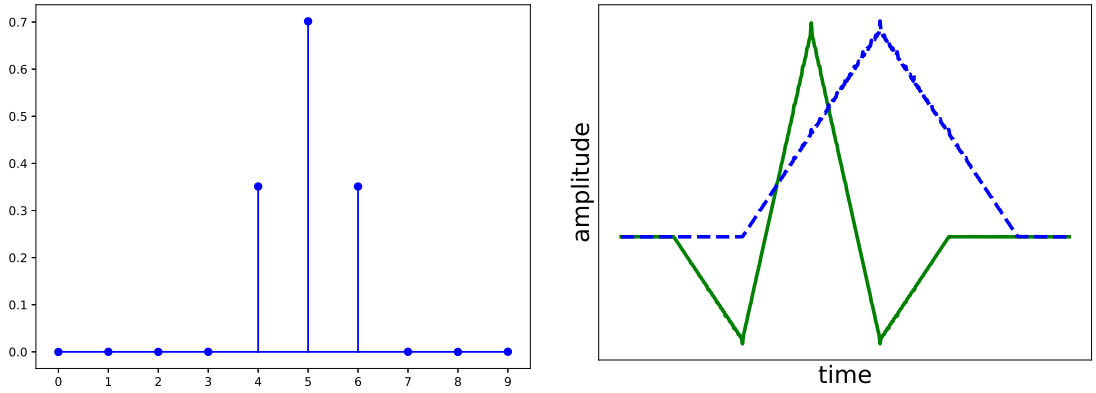
$$L(X; g, h) = \frac{1}{M}\sum_{i=1}^{M} ||x_i - \hat{x}_i||_2^2 + \lambda_1 \frac{1}{M}\sum_{i=1}^{M} ||W(x_i)||_1 + \lambda_2 \left[L_w(h_1, g_1) + L_w(h_2, g_2)\right] \quad (7.21)$$

where (h_1, g_1) and (h_2, g_2) are the analysis and synthesis filter pair respectively. The wavelet coefficients, $W(x_i)$, are computed using (h_1, g_1), while $\hat{x}_i$ is computed from $W(x_i)$ using (h_2, g_2).

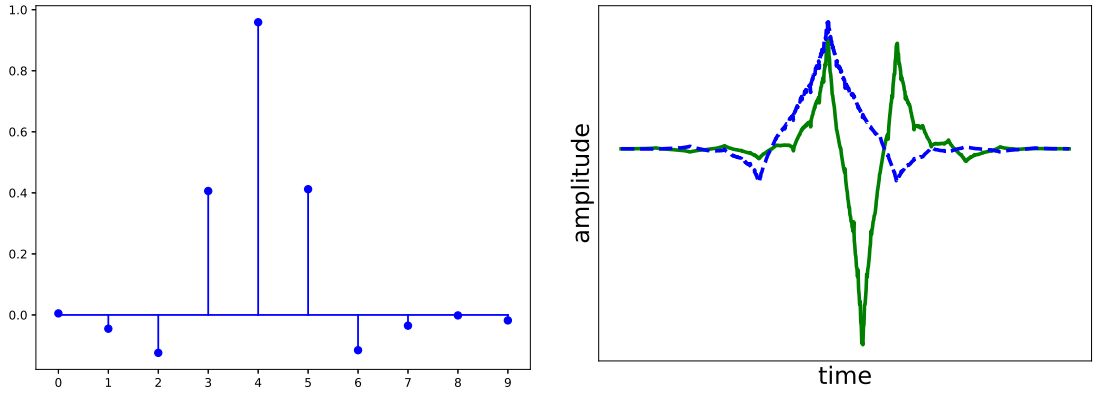
Figure 7.9 shows two scaling filters learned from one-dimensional sawtooth data (See Chapter 4). An interesting property of the learned filters is that they are symmetric, unlike the learned filters from previous chapters. It turns out it is not possible to have both symmetric and orthogonal filters (with the exception of the Haar wavelet) [Cohen et al., 1992]. However, biorthogonal filters can be symmetric. We leave further analysis of this biorthogonal wavelet model as future work.

7.3.2 Variational Autoencoders

In Chapters 4 and 5 we showed how we can sample signals from our model. However, it was not easy to get samples that were qualitatively similar to training examples. Ideally we would like to generate a signal by giving the decoder a random code drawn from some distribution (a Laplacian in our case). The problem with this method is the random code



(a)



(b)

Figure 7.9: Scaling filters and corresponding wavelet and scaling functions learned from the biorthogonal loss function (Equation 7.21). (a) analysis (forward) filter, and (b) synthesis (inverse) filter.

we draw may not be correspond to a meaningful signal. This was illustrated in Figure 5.12, where the random images looked nothing like the training images.

One solution to this problem is to modify the autoencoder model so that the encoder learns to map signals to codes that are close to a known distribution, such as a Gaussian. These models are known as variational autoencoders (VAE) [Kingma and Welling, 2013]. The main idea is that instead of the encoder mapping an input signal to a real vector, it instead outputs parameters of a distribution (such as the means and standard deviations). A sample code is drawn from this distribution, which is then used as input to the decoder.

The VAE loss is composed of two terms. The first is a standard reconstruction loss, such as mean squared error. The second term is the KL divergence between the distribution output by the encoder and the desired distribution (often a unit Gaussian). The KL divergence is a similarity measure between two distributions [Kullback and Leibler, 1951]. This loss term means that samples drawn from the desired distribution should correspond to valid signals.

Unfortunately, the sampling operation is not differentiable. This problem can be avoided by moving the sampling operation to its own layer in the network using what is known as the "reparameterization trick" [Doersch, 2016]. Suppose, given an input signal x, the encoder outputs the parameters of a Gaussian distribution μ_x and Σ_x. Instead of drawing a sample from that distribution

$$z \sim \mathcal{N}(\mu_x, \Sigma_x) \tag{7.22}$$

we first draw a sample from a zero mean, unit Gaussian

$$\epsilon \sim \mathcal{N}(0, I). \tag{7.23}$$

We can then obtain an appropriate code sample by transforming ϵ

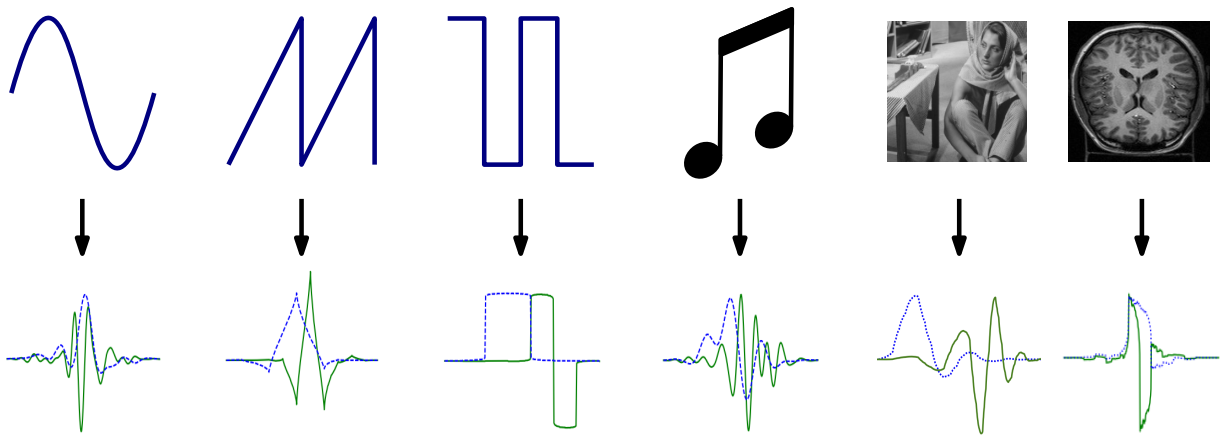
$$z = \mu_x + \Sigma_x^{1/2} \cdot \epsilon \tag{7.24}$$

Performing the reparameterization trick allows us to differentiate through the VAE. A downside of VAEs, especially ones with many layers, is that they can be difficult to train [Sønderby et al., 2016].

It is not obvious how we might adapt our autoencoder model into the VAE framework. One possibility is to introduce intermediate layers (between the encoder and decoder) that are responsible for transforming the wavelet coefficients to appropriate distribution parameters. One might think of this as inserting a VAE into the middle of the wavelet autoencoder architecture. We leave this avenue of research for future work.

Chapter 8

Conclusion



The goal of this thesis was to learn useful data representations. We chose to focus on the wavelet transform, and proposed a method of learning wavelets from data using ideas from machine learning. In this chapter we return to the two questions that began this thesis:

1. Can useful wavelets be learned?
2. Do traditional wavelets arise from natural data?

8.1 Summary

We began this thesis with a review of signal processing in Chapter 2, with a focus on the wavelet transform. In Chapter 4 contained the main contribution of this thesis: a wavelet learning method based on an autoencoder framework. We first formulate the model for one-dimensional signals, and show that a variety of wavelets can be learned depending on the structure of the training data. A probabilistic interpretation of the model is derived, and is used to compare it to similar sparse dictionary learning approaches. In Chapter 5 we extend our method to multiple dimensions with a straightforward generalization of the one-dimensional model. Similar to the one-dimensional case, it is shown that unique wavelets arise from different structured data. Furthermore, the learned wavelets are very similar to traditionally derived wavelets.

Chapter 6 addressed some limitations of wavelet transform caused by using real, separable filters. Our proposed solution was to extend our wavelet learning model into the domain of the dual-tree wavelet transform. The main result of this chapter was that localized and oriented filters can arise from natural data. The learned filters were also shown to be similar to the q-shift filters found in [Kingsbury, 2000, Kingsbury, 2003].

Finally, Chapter 7 explored various applications of our wavelet model. The first task considered was signal classification. We proposed a novel wavelet classification algorithm, and demonstrated its performance on a dataset of bird songs. An adaptation of the wavelet classification algorithm was proposed that took inspiration from K-means. We then demonstrated how we could adapt our learning method to the tasks of image compression and classification. Finally, we discussed some possible extensions to our model

Now to return to our two questions from the start of this chapter. We believe we are able to answer each with a "yes":

1. Can useful wavelets be learned?

In our various experiments from Chapters 4, 5, and 6, we demonstrated that is possible to learn wavelet filters from data. Further more, we showed in Chapter 7 that these learned wavelets could be used for a variety of applications including audio and image classification, clustering, and image compression.

2. Do traditional wavelets arise from natural data?

The synthetic and real experiments in Chapters 4, 5, and 6 yielded wavelets that were similar traditional families of wavelets. We demonstrated this by comparing the learned

filter values to various traditional wavelet filter values. Furthermore, the learned wavelets from the compression experiments in Chapter 7 are qualitatively similar to traditional wavelets.

8.2 Future Work

Much of the work presented in this thesis was exploratory, perhaps raising more questions than answers. In Chapter 7 we explored several different applications and directions of future work. The first application considered was signal classification, where we proposed a novel wavelet classification algorithm. The algorithm is a simple extension of our wavelet model, and learns class representations by way of reconstructing the training signals. The algorithm may be improved by instead learning discrimitive representations as is done in [Mairal et al., 2008a]. In other words, the learned wavelets should be tuned to discriminate well between signals of different classes, instead of being chosen to give good signal reconstructions. The next task considered was clustering, where we proposed a wavelet clustering algorithm similar to K-means. As well, an EM-like variant was proposed that may overcome some limitations of the hard clustering algorithm. Analysis and evaluation of these methods is likely a fruitful avenue of research.

Image compression using learned wavelets is another direction that deserves more investigation. We demonstrated that learned wavelets outperform the standard JPEG2000 biorthogonal 4,4 wavelet for a variety of images. We conjectured that there exist classes of images that are well suited to using a learned wavelet for compression. Investigating this claim may lead to insights into better image compression schemes. For example, it may be useful to adapt JPEG2000 to make use of different wavelets depending on the properties of the compressed image.

The image classification experiments from Chapter 7 suggest that our wavelet model performs on par with comparable CNN architectures. However, our experiments were performed on relatively small datasets and simple architectures. It will be important to conduct further experiments by incorporating wavelet filter constraints into state-of-the-art architectures in order to determine if there are any advantages. Furthermore, investigating the difference between the wavelets learned using unsupervised learning (autoencoder) and supervised learning (classification) may yield interesting results.

This thesis restricted itself to the problem of learning sparse, orthogonal wavelets. There are many extensions to our model that could be considered by either relaxing or modifying these constraints. In Chapter 7 we demonstrated that the model can learn symmetric

wavelet filters by a simple relaxation of the orthogonality constraint. Another variant could involve changing the sparsity constraint to something other than an ℓ_1 penalty. One possibility is a mixture of a Gaussian and a Dirac delta function as is used in [Sallee and Olshausen, 2003]

Signal generation is another compelling area of future work. We presented some simple generation methods in Chapters 4 and 5, but none of them garnered results that well represented the training data. In Chapter 7 we discussed how we might incorporate ideas from variational autoencoders to allow for better generative models [Kingma and Welling, 2013]. Another class of generative models to consider are generative adversarial networks, which are trained to generate signals by "fooling" a second adversarial network [Goodfellow et al., 2014].

References

[Abadi et al., 2015] Abadi, M. et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Abdi and Williams, 2010] Abdi, H. and Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459.

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

[Bay et al., 2008] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359.

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

[Bentley et al., 1980] Bentley, J. L., Haken, D., and Saxe, J. B. (1980). A general method for solving divide-and-conquer recurrences. *ACM SIGACT News*, 12(3):36–44.

[Billings and Wei, 2005] Billings, S. A. and Wei, H.-L. (2005). A new class of wavelet networks for nonlinear system identification. *IEEE Transactions on neural networks*, 16(4):862–874.

[Bilmes et al., 1998] Bilmes, J. A. et al. (1998). A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical Report TR-97-021, International Computer Science Institute.

[Boersman and Weenink, 2013] Boersman, P. and Weenink, D. (http://www.praat.org/, Version 5.3.57, 27 October 2013). Praat: doing phonetics by computer.

[Borman, 2004] Borman, S. (2004). The expectation maximization algorithm – a short tutorial. http://www.seanborman.com/publications/EM_algorithm.pdf. Accessed: July 20, 2017.

[Bourlard and Kamp, 1988] Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4):291–294.

[Bousquet and Bottou, 2008] Bousquet, O. and Bottou, L. (2008). The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168.

[Brislawn et al., 1996] Brislawn, C. M., Bradley, J. N., Onyshczak, R. J., and Hopper, T. (1996). Fbi compression standard for digitized fingerprint images. In *Applications of Digital Image Processing XIX*, volume 2847, pages 344–356. International Society for Optics and Photonics.

[Bruna and Mallat, 2011] Bruna, J. and Mallat, S. (2011). Classification with scattering operators. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1561–1566. IEEE.

[Bruna and Mallat, 2013] Bruna, J. and Mallat, S. (2013). Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886.

[Burrus et al., 1997] Burrus, C. S., Gopinath, R. A., and Guo, H. (1997). *Introduction to wavelets and wavelet transforms: a primer*. Prentice-Hall, Inc.

[Chen et al., 2016] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2016). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*.

[Choi et al., 2017] Choi, K., Fazekas, G., Sandler, M., and Cho, K. (2017). Convolutional recurrent neural networks for music classification. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 2392–2396. IEEE.

[Christopoulos et al., 2000] Christopoulos, C., Skodras, A., and Ebrahimi, T. (2000). The jpeg2000 still image coding system: an overview. *IEEE transactions on consumer electronics*, 46(4):1103–1127.

[Cohen et al., 1992] Cohen, A., Daubechies, I., and Feauveau, J.-C. (1992). Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 45(5):485–560.

[Coifman and Wickerhauser, 1992] Coifman, R. R. and Wickerhauser, M. V. (1992). Entropy-based algorithms for best basis selection. *IEEE Transactions on information theory*, 38(2):713–718.

[Collobert et al., 2011] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

[Cooley and Tukey, 1965] Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

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

[Davis and Mermelstein, 1990] Davis, S. B. and Mermelstein, P. (1990). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. In *Readings in speech recognition*, pages 65–74. Elsevier.

[Dean et al., 2012] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231.

[Doersch, 2016] Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

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

[Emiya et al., 2010] Emiya, V., Badeau, R., and David, B. (2010). Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1643–1654.

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

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[Fourier, 1822] Fourier, J. (1822). *Theorie analytique de la chaleur, par M. Fourier*. Chez Firmin Didot, père et fils.

[Franzen, 1999] Franzen, R. (1999). Kodak lossless true color image suite. http://r0k.us/graphics/kodak/. Accessed: 2018-06-01.

[Gabor, 1946] Gabor, D. (1946). Theory of communication. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 93(26):429–457.

[Gaing, 2004] Gaing, Z.-L. (2004). Wavelet-based neural network for power disturbance recognition and classification. *IEEE transactions on power delivery*, 19(4):1560–1568.

[Gersho and Gray, 2012] Gersho, A. and Gray, R. M. (2012). *Vector quantization and signal compression*, volume 159. Springer Science & Business Media.

[Ghosh-Dastidar and Adeli, 2003] Ghosh-Dastidar, S. and Adeli, H. (2003). Wavelet-clustering-neural network model for freeway incident detection. *Computer-Aided Civil and Infrastructure Engineering*, 18(5):325–338.

[Ghosh-Dastidar et al., 2007] Ghosh-Dastidar, S., Adeli, H., and Dadmehr, N. (2007). Mixed-band wavelet-chaos-neural network methodology for epilepsy and epileptic seizure detection. *IEEE transactions on biomedical engineering*, 54(9):1545–1551.

[Gini, 1912] Gini, C. (1912). Variabilità e mutabilità. *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi*.

[Gini, 1921] Gini, C. (1921). Measurement of inequality of incomes. *The Economic Journal*, 31(121):124–126.

[Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Gopinath et al., 1994] Gopinath, R., Odegard, J., and Burrus, C. (1994). Optimal wavelet representation of signals and the wavelet sampling theorem. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 41(4):262–277.

[Grasemann and Miikkulainen, 2004] Grasemann, U. and Miikkulainen, R. (2004). Evolving wavelets using a coevolutionary genetic algorithm and lifting. In *Genetic and Evolutionary Computation Conference*, pages 969–980. Springer.

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

[Gupta and Chen, 2010] Gupta, M. R. and Chen, Y. (2010). Theory and use of the em algorithm. *Signal Processing*, 4(3):223–296.

[Haar, 1910] Haar, A. (1910). Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3):331–371.

[Hadji and Wildes, 2017] Hadji, I. and Wildes, R. P. (2017). A spatiotemporal oriented energy network for dynamic texture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3066–3074.

[Hariharan et al., 2015] Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. (2015). Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 447–456.

[Hinton et al., 2012] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Holschneider et al., 1990] Holschneider, M., Kronland-Martinet, R., Morlet, J., and Tchamitchian, P. (1990). A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets*, pages 286–297. Springer.

[Hoshen et al., 2015] Hoshen, Y., Weiss, R. J., and Wilson, K. W. (2015). Speech acoustic modeling from raw multichannel waveforms. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4624–4628. IEEE.

[Hotelling, 1933] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456.

[Jain, 2010] Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666.

[Jutten and Herault, 1991] Jutten, C. and Herault, J. (1991). Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal processing*, 24(1):1–10.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kingsbury, 1998] Kingsbury, N. (1998). The dual-tree complex wavelet transform: a new efficient tool for image restoration and enhancement. In *Signal Processing Conference (EUSIPCO 1998), 9th European*, pages 1–4. IEEE.

[Kingsbury, 2000] Kingsbury, N. (2000). A dual-tree complex wavelet transform with improved orthogonality and symmetry properties. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 2, pages 375–378. IEEE.

[Kingsbury, 2003] Kingsbury, N. (2003). Design of q-shift complex wavelets for image processing using frequency domain energy minimization. In *Image Processing, 2003.*

ICIP 2003. Proceedings. 2003 International Conference on, volume 1, pages I–1013. IEEE.

[Krizhevsky and Hinton, 2009] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

[LeCun, 1985] LeCun, Y. (1985). Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In *Proceedings of Cognitiva 85, Paris, France*.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun et al., 2010] LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE.

[Lee and Seung, 1999] Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788.

[Lee and Seung, 2001] Lee, D. D. and Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562.

[Lee et al., 2005] Lee, K.-C., Ho, J., and Kriegman, D. J. (2005). Acquiring linear subspaces for face recognition under variable lighting. *IEEE Transactions on pattern analysis and machine intelligence*, 27(5):684–698.

[Lin et al., 2016] Lin, G., Milan, A., Shen, C., and Reid, I. (2016). Refinenet: Multi-path refinement networks with identity mappings for high-resolution semantic segmentation. *arXiv preprint arXiv:1611.06612*.

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528.

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. IEEE.

[Luke, 1999] Luke, H. D. (1999). The origins of the sampling theorem. *IEEE Communications Magazine*, 37(4):106–108.

[Mairal et al., 2008a] Mairal, J., Bach, F., Ponce, J., Sapiro, G., and Zisserman, A. (2008a). Discriminative learned dictionaries for local image analysis. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE.

[Mairal et al., 2007] Mairal, J., Sapiro, G., and Elad, M. (2007). Multiscale sparse image representationwith learned dictionaries. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 3, pages III–105. IEEE.

[Mairal et al., 2008b] Mairal, J., Sapiro, G., and Elad, M. (2008b). Learning multiscale sparse representations for image and video restoration. *Multiscale Modeling & Simulation*, 7(1):214–241.

[Mallat, 2008] Mallat, S. (2008). *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd edition.

[Mallat, 2012] Mallat, S. (2012). Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398.

[Mallat, 2016] Mallat, S. (2016). Understanding deep convolutional networks. *Phil. Trans. R. Soc. A*, 374(2065):20150203.

[Mallat, 1989] Mallat, S. G. (1989). A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11(7):674–693.

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

[Mandal et al., 1996] Mandal, M. K., Panchanathan, S., and Aboulnasr, T. (1996). Choice of wavelets for image compression. In *Information Theory and Applications II*, pages 239–249. Springer.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Mermelstein, 1976] Mermelstein, P. (1976). Distance measures for speech recognition, psychological and instrumental. *Pattern recognition and artificial intelligence*, 116:374–388.

[Morgan et al., 2005] Morgan, N., Zhu, Q., Stolcke, A., Sonmez, K., Sivadas, S., Shinozaki, T., Ostendorf, M., Jain, P., Hermansky, H., Ellis, D., et al. (2005). Pushing the envelope-aside [speech recognition]. *IEEE Signal Processing Magazine*, 22(5):81–88.

[Mumford and Desolneux, 2010] Mumford, D. and Desolneux, A. (2010). *Pattern theory: the stochastic analysis of real-world signals*. CRC Press.

[Murtagh and Contreras, 2012] Murtagh, F. and Contreras, P. (2012). Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97.

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

[Ngiam et al., 2011] Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Le, Q. V., and Ng, A. Y. (2011). On optimization methods for deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 265–272.

[Nielsen et al., 2006] Nielsen, M., Kamavuako, E. N., Andersen, M. M., Lucas, M.-F., and Farina, D. (2006). Optimal wavelets for biomedical signal compression. *Medical and Biological Engineering and Computing*, 44(7):561–568.

[Odena et al., 2016] Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*.

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

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

[Olshausen et al., 2001] Olshausen, B. A., Sallee, P., and Lewicki, M. S. (2001). Learning sparse image codes using a wavelet pyramid architecture. In *Advances in neural information processing systems*, pages 887–893.

[Oord et al., 2016] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

[Ophir et al., 2011] Ophir, B., Lustig, M., and Elad, M. (2011). Multi-scale dictionary learning using wavelets. *IEEE Journal of Selected Topics in Signal Processing*, 5(5):1014–1024.

[Palaz et al., 2013] Palaz, D., Collobert, R., and Doss, M. M. (2013). Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks. *arXiv preprint arXiv:1304.1018*.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

[Pearson, 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[Permuter et al., 2003] Permuter, H., Francos, J., and Jermyn, I. H. (2003). Gaussian mixture models of texture and colour for image database retrieval. In *Acoustics, Speech, and*

Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on, volume 3, pages III–569. IEEE.

[Petrosian et al., 2000] Petrosian, A., Prokhorov, D., Homan, R., Dasheiff, R., and Wunsch II, D. (2000). Recurrent neural network based prediction of epileptic seizures in intra-and extracranial eeg. *Neurocomputing*, 30(1-4):201–218.

[Pohlmann, 1989] Pohlmann, K. C. (1989). *The compact disc: a handbook of theory and use*, volume 5. AR Editions, Inc.

[Poldrack et al., 2013] Poldrack, R. A., Barch, D. M., Mitchell, J., Wager, T., Wagner, A. D., Devlin, J. T., Cumba, C., Koyejo, O., and Milham, M. (2013). Toward open sharing of task-based fmri data: the openfmri project. *Frontiers in neuroinformatics*, 7:12.

[Quellec et al., 2008] Quellec, G., Lamard, M., Josselin, P. M., Cazuguel, G., Cochener, B., and Roux, C. (2008). Optimal wavelet transform for the detection of microaneurysms in retina photographs. *IEEE Trans Med Imaging*, 27(9):1230–41.

[Recoskie, 2014] Recoskie, D. (2014). Constrained nonnegative matrix factorization with applications to music transcription. Master's thesis, University of Waterloo.

[Recoskie and Mann, 2018a] Recoskie, D. and Mann, R. (2018a). Gradient-based filter design for the dual-tree wavelet transform. *arXiv preprint arXiv:1806.01793*.

[Recoskie and Mann, 2018b] Recoskie, D. and Mann, R. (2018b). Learning filters for the 2D wavelet transform. In *Computer and Robot Vision (CRV), 2018 15th Conference on*. ©2018 IEEE.

[Recoskie and Mann, 2018c] Recoskie, D. and Mann, R. (2018c). Learning sparse wavelet representations. *arXiv preprint arXiv:1802.02961*.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer.

[Rosen and Howell, 2011] Rosen, S. and Howell, P. (2011). *Signals and systems for speech and hearing*, volume 29. Brill.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

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

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[Russell and Norvig, 1995] Russell, S. J. and Norvig, P. (1995). *Artificial intelligence: a modern approach*.

[Rustamov and Guibas, 2013] Rustamov, R. and Guibas, L. J. (2013). Wavelets on graphs via deep learning. In *Advances in Neural Information Processing Systems*, pages 998–1006.

[Sainath et al., 2013] Sainath, T. N., Mohamed, A.-r., Kingsbury, B., and Ramabhadran, B. (2013). Deep convolutional neural networks for lvcsr. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8614–8618. IEEE.

[Sainath et al., 2015] Sainath, T. N., Weiss, R. J., Senior, A., Wilson, K. W., and Vinyals, O. (2015). Learning the speech front-end with raw waveform cldnns. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[Sallee and Olshausen, 2003] Sallee, P. and Olshausen, B. A. (2003). Learning sparse multiscale image representations. In *Advances in neural information processing systems*, pages 1351–1358.

[Sallee, 2004] Sallee, P. A. (2004). *Statistical methods for image and signal processing*. PhD thesis, University of California, Davis.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

[Schonberg et al., 2012] Schonberg, T., Fox, C. R., Mumford, J. A., Congdon, E., Trepel, C., and Poldrack, R. A. (2012). Decreasing ventromedial prefrontal cortex activity during sequential risk-taking: an fmri investigation of the balloon analog risk task. *Frontiers in neuroscience*, 6:80.

[Selesnick, 1997] Selesnick, I. W. (1997). Maple and the parameterization of orthogonal wavelet bases. *Rap. tech., Rice University*.

[Selesnick et al., 2005] Selesnick, I. W., Baraniuk, R. G., and Kingsbury, N. C. (2005). The dual-tree complex wavelet transform. *IEEE signal processing magazine*, 22(6):123–151.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell Syst. Tech. J.*, 27:379–423.

[Simoncelli and Freeman, 1995] Simoncelli, E. P. and Freeman, W. T. (1995). The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *Image Processing, 1995. Proceedings., International Conference on*, volume 3, pages 444–447. IEEE.

[Singh and Kingsbury, 2017a] Singh, A. and Kingsbury, N. (2017a). Dual-tree wavelet scattering network with parametric log transformation for object classification. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 2622–2626. IEEE.

[Singh and Kingsbury, 2017b] Singh, A. and Kingsbury, N. (2017b). Efficient convolutional network learning using parametric log based dual-tree wavelet scatternet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1140–1147.

[Smaragdis and Brown, 2003] Smaragdis, P. and Brown, J. C. (2003). Non-negative matrix factorization for polyphonic music transcription. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 177–180. IEEE.

[Søgaard, 2017] Søgaard, A. (2017). Learning optimal wavelet bases using a neural network approach. *arXiv preprint arXiv:1706.03041*.

[Sønderby et al., 2016] Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746.

[Strang and Nguyen, 1996] Strang, G. and Nguyen, T. (1996). *Wavelets and filter banks*. SIAM.

[Sweldens, 1998] Sweldens, W. (1998). The lifting scheme: A construction of second generation wavelets. *SIAM journal on mathematical analysis*, 29(2):511–546.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., et al. (2015). Going deeper with convolutions. Cvpr.

[Szu et al., 1992] Szu, H. H., Telfer, B. A., and Kadambe, S. L. (1992). Neural network adaptive wavelets for signal representation and classification. *Optical Engineering*, 31(9):1907–1917.

[Tewfik et al., 1992] Tewfik, A. H., Sinha, D., and Jorgensen, P. (1992). On the optimal choice of a wavelet for signal representation. *IEEE Transactions on information theory*, 38(2):747–765.

[Theano Development Team, 2016] Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

[Tibshirani, 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

[Tüske et al., 2014] Tüske, Z., Golik, P., Schlüter, R., and Ney, H. (2014). Acoustic modeling with deep neural networks using raw time signal for lvcsr. In *Fifteenth Annual Conference of the International Speech Communication Association*.

[University of Montreal, 2007a] University of Montreal (2007a). Recognition of convex sets. http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/RectanglesData. Accessed: April 27, 2018.

[University of Montreal, 2007b] University of Montreal (2007b). Rectangles and rectangles-images data. http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/ConvexNonConvex. Accessed: April 27, 2018.

[Vaidyanathan and Hoang, 1988] Vaidyanathan, P. P. and Hoang, P.-Q. (1988). Lattice structures for optimal design and robust implementation of two-channel perfect-reconstruction qmf banks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(1):81–94.

[Villasenor et al., 1995] Villasenor, J. D., Belzer, B., and Liao, J. (1995). Wavelet filter evaluation for image compression. *IEEE Transactions on image processing*, 4(8):1053–1060.

[Walker and Nguyen, 2001] Walker, J. S. and Nguyen, T. Q. (2001). Wavelet-based image compression. *Sub-chapter of CRC Press book: Transforms and Data Compression*.

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

[Wang et al., 2001] Wang, L., Teo, K. K., and Lin, Z. (2001). Predicting time series with wavelet packet neural networks. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 1593–1597. IEEE.

[Wang et al., 2017] Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., and Summers, R. M. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3462–3471. IEEE.

[Wengert, 1964] Wengert, R. E. (1964). A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464.

[Werbos, 1982] Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer.

[Whittaker, 1915] Whittaker, E. (1915). On the functions which are represented by the expansion of interpolating theory. In *Proc. Roy. Soc. Edinburgh*, volume 35, pages 181–194.

[Wozniak, 2009] Wozniak, P. (2009). Brick wall close-up view. http://freestocktextures.com/texture/id/690. Creative Commons Attribution-Share Alike 3.0 Unported license.

[Xeno-canto Foundation, 2004] Xeno-canto Foundation (2004). Dataset of bird songs. https://archive.org/details/xccoverbl_2014.

[Yang et al., 2007] Yang, A. Y., Wright, J., Ma, Y., and Sastry, S. S. (2007). Feature selection in face recognition: A sparse representation perspective. Technical Report UCB/EECS-2007-99, EECS Department, University of California, Berkeley.

[Zhang, 1997] Zhang, Q. (1997). Using wavelet network in nonparametric estimation. *IEEE Transactions on Neural networks*, 8(2):227–236.

[Zhang and Benveniste, 1992] Zhang, Q. and Benveniste, A. (1992). Wavelet networks. *IEEE transactions on Neural Networks*, 3(6):889–898.

[Zhuang and Baras, 1994] Zhuang, Y. and Baras, J. S. (1994). Optimal wavelet basis selection for signal representation. In *Wavelet Applications*, volume 2242, pages 200–212. International Society for Optics and Photonics.

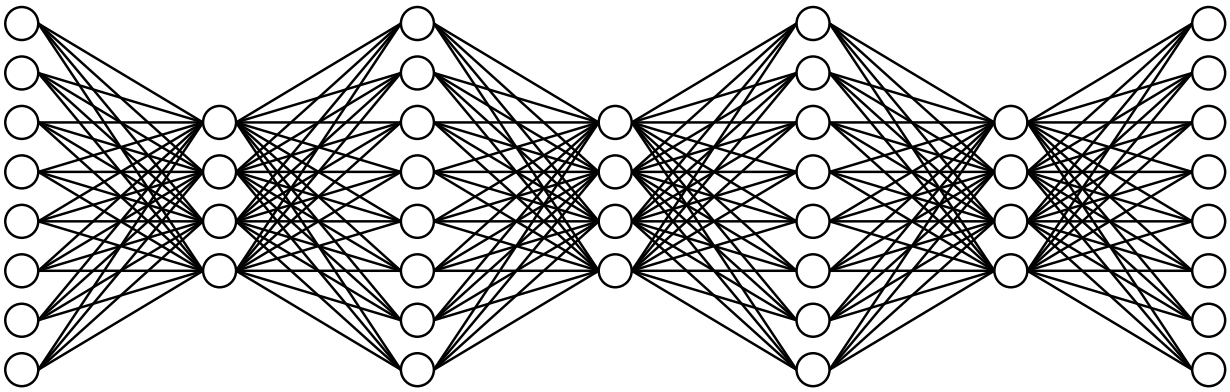
[Zhuang and Baras, 1996] Zhuang, Y. and Baras, J. S. (1996). Constructing optimal wavelet basis for image compression. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 4, pages 2351–2354. IEEE.

[Zou and Tewfik, 1993] Zou, H. and Tewfik, A. H. (1993). Parametrization of compactly supported orthonormal wavelets. *IEEE Transactions on signal processing*, 41(3):1428–1431.

APPENDICES

Appendix A

Machine Learning and Neural Networks



This chapter aims to give the reader a brief introduction to concepts from machine learning. The topics included are chosen because they will be relevant at different sections of the thesis. We focus particularly on the topic of neural networks, since it will be useful in understanding our wavelet learning model in Chapter 4. We end the chapter with a discussion of some recent neural network architectures.

A.1 Machine Learning

This section introduces the area of machine learning. We will consider two classes of machine learning problems: supervised and unsupervised learning. Supervised learning can be viewed in terms of function approximation. Let us assume that there exists some function f that we wish to approximate. The domain of f is typically $\mathbb{R}^n$ or $\{0,1\}^n$. Each dimension is called a feature of our data. The output of the function is generally divided into two types: discrete (i.e. a finite set of classes), and continuous (i.e. $\mathbb{R}$). The first case is called classification, and the second case regression. The goal of supervised learning is to approximate f with a learned function $\tilde{f}$ given a training set of the form $\{(x_i, f(x_i))\}_{1 \leq i \leq M}$. The approximation, $\tilde{f}$, is often obtained through the minimization of a loss (or error) function $L(f, \tilde{f})$. A common loss function for classification is cross entropy (CE). The binary classification formulation is below:

$$L_{CE}(f, \tilde{f}) = -\frac{1}{M} \sum_{i=1}^{M} \Big(f(x_i) \log \tilde{f}(x_i) + (1 - f(x_i)) \log(1 - \tilde{f}(x_i)) \Big) \tag{A.1}$$

In the case of regression, mean squared error (MSE) is often used:

$$L_{MSE}(f, \tilde{f}) = \frac{1}{M} \sum_{i=1}^{M} |f(x_i) - \tilde{f}(x_i)|^2 \tag{A.2}$$

We will not go into anymore detail concerning supervised learning, since the main contribution of our thesis is an unsupervised technique.

Unsupervised learning is concerned with finding structure in data. Unlike supervised learning, we are given data without labels. Typical tasks in unsupervised learning include: clustering, density estimation, and dimensionality reduction. The following sections will briefly outline these tasks, and describe commonly used methods.

A.1.1 Clustering

In clustering we wish to separate datapoints into discrete classes, but we are not explicitly told the desired classes. Formally, given a dataset $X = \{x_1, \ldots, x_m\}$ of vectors, we want to find disjoint sets $C = (X_1, \ldots, X_K)$ with $\cup_{k=1}^{K} X_k = X$, such that elements of a given cluster are similar to each other, but dissimilar to points in other clusters. The precise definition of similarity varies. One simple definition of similarity is Euclidean distance. A common

clustering technique for this similarity measure is K-means, independently proposed by several researchers [Jain, 2010]. K-means seeks to minimize:

$$J(C) = \sum_{k=1}^{K} \sum_{x \in X_k} ||\mu_k - x||_2^2 \tag{A.3}$$

where μ_k is the mean of cluster X_k. K-means is an example of centroid based clustering, since each cluster is defined in terms of its mean. Many other clustering strategies exist, such as density and hierarchical based methods [Ester et al., 1996, Murtagh and Contreras, 2012].

A.1.2 Mixture Models for Density Estimation

A disadvantage of clustering is that each datapoint must be assigned a discrete label. Instead, we may prefer to assign a probability that a datapoint belongs to a given cluster. We can view this as a form of soft clustering. A mixture model is one technique for accomplishing soft clustering. Suppose our data is generated from K different distributions. Let us represent the density function as

$$P(x) = \sum_{k=1}^{K} \alpha_k P(x \mid \theta_k) \tag{A.4}$$

where θ_k are the parameters of the k^{th} distribution, and $\alpha_k > 0$ are the mixing proportions with $\sum_{k=1}^{K} \alpha_k = 1$. The generative process of the data is straightforward: first randomly choose a distribution according to the α_k's, then draw a sample from $P(x \mid \theta_k)$. Equation A.4 is called a mixture model.

Each datapoint in our dataset has a corresponding distribution that it was sampled from. Let $z_i \in \{1, \ldots, K\}$ denote the class label of the i^{th} datapoint. The value of each z_i is unobserved (or latent). The task of learning a mixture model is equivalent to choosing values for the parameters (θ_k's and α_k's) and latent values (z_i's).

If we assume the component distributions are Gaussian, then Equation A.4 becomes:

$$P(x) = \sum_{k=1}^{K} \alpha_k \mathcal{N}(\mu_k, \sigma_k^2) \tag{A.5}$$

Equation A.5 is called a Gaussian mixture model (GMM). GMMs are a popular class of models in the machine learning community which have been applied to various tasks such as speech recognition [Morgan et al., 2005], and image classification [Permuter et al., 2003].

A popular method for learning the parameters and latent variables of a mixture model is the expectation maximization (EM) algorithm [Bilmes et al., 1998, Borman, 2004]. EM is a method for estimating the parameters that maximizes the probability of observing our data, X. Let Θ and Z be the parameters and latent values respectively. Our goal is to maximize $P(X \mid \Theta) = \sum_z P(X, z \mid \Theta) P(z \mid \Theta)$, which is typically intractable since the sum is over all possible labellings. EM attempts to maximize $\log P(X \mid \Theta)$ with two alternating steps. In the first step (expectation) a function is found that lower bounds $\log P(X \mid \Theta)$. Specifically, a function Q is computed such that

$$Q(\Theta \mid \Theta^{(t)}) \leq \log P(X \mid \Theta) \tag{A.6}$$

and

$$Q(\Theta^{(t)} \mid \Theta^{(t)}) = \log P(X \mid \Theta^{(t)}) \tag{A.7}$$

where $\Theta^{(t)}$ is our current estimate of Θ. In the second step (maximization) Θ is chosen to maximize $Q(\Theta \mid \Theta^{(t)})$. From equations A.6 and A.7, we see that our new estimate of Θ necessarily increases $\log P(X \mid \Theta)$. This process can be repeated until convergence.

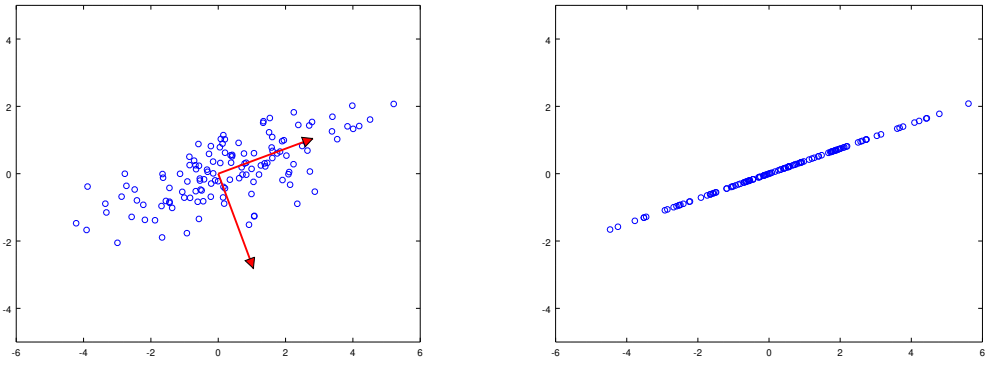
The expectation step involves an estimation of the latent values. The maximization step then updates the parameters based on the estimates. In the case of a GMM, the expectation step estimates the probability that a datapoint belongs to a given cluster. We can view this step as a soft clustering of the datapoints. The maximization step uses these membership probabilities as a weighting to estimate the parameters of each Gaussian distribution, as well as the mixing proportions. See [Gupta and Chen, 2010] for further details of EM.

A.1.3 Dimensionality Reduction

The purpose of dimensionality reduction is to project our data onto a low dimension space such that we minimize the amount of lost information. Matrix factorization is one way of achieving dimensionality reduction. In matrix factorization, our dataset X is treated as a matrix, and is factored into a product of other matrices. Two matrix factorization techniques will be described next: principal components analysis (PCA), and nonnegative matrix factorization (NMF).

The goal of PCA is to find the directions in our original space (principal components) that capture the most information about our data [Abdi and Williams, 2010]. The principal components can be found by computing the singular value decomposition of X:

$$X = P \Delta Q^{\intercal} \tag{A.8}$$

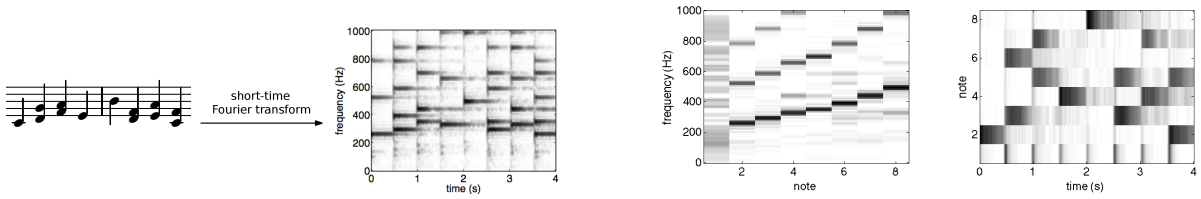


(a) (b)

Figure A.1: (a) An example of PCA applied to two dimensional data. The arrows indicate the principal components. (b) The data from (a) projected onto its first principal component.

where P and Q are matrices containing the singular vectors, and Δ is a diagonal matrix of singular values. We can view Q as a projection matrix because X can be projected onto its principal components by computing XQ. Suppose we wish to project X onto its first k principal components (thereby reducing the dimension to k). This can be achieved by multiplying X by the first k columns of Q. See Figure A.1a for an example of PCA applied to two dimensional data.

NMF attempts to factorize our data as $X = BG$ where B and G are nonnegative [Lee and Seung, 1999]. In other words, it is assumed that each datapoint is generated by a sum of nonnegative component vectors. In general, NMF is a nonconvex optimization problem. The factorization matrices, B and G, are learned with an iterative gradient descent based algorithm similar to the EM algorithm [Lee and Seung, 2001]. This model is well suited for audio, as sounds are additive. NMF has seen success in the application of music transcription [Smaragdis and Brown, 2003]. See Figure A.2 for an illustration of NMF used for music transcription from the author's master's work [Recoskie, 2014].



(a) (b)

Figure A.2: (a) The short-time Fourier transform of an music recording is taken to obtain the matrix X. (b) NMF produces note matrix B (left) and note activation matrix G (right).

A.2 Neural Networks and Deep Learning

Over the past several years, neural networks (NNs) have become a dominant method for machine learning. The term deep learning is often used to describe NN models in general. The term comes from the fact that recent NN architectures are made up of many functional layers. Recent interest in NNs is often accredited to major results on two machine learning tasks: acoustic modelling [Hinton et al., 2012], and object recognition [Krizhevsky et al., 2012]. For a thorough review, including a discussion of their history, see [LeCun et al., 2015, Schmidhuber, 2015]. Though popular lately, NNs began their history in the 1940s [McCulloch and Pitts, 1943]. However, the parameters of these early networks were fixed. The first NNs capable of learning appeared in the 1950s. The perceptron, a simple binary classifier, was one of the earliest learning models [Rosenblatt, 1958]. Since then, NN models have increased in complexity, and have demonstrated exceptional results on many tasks.

A.2.1 Feed-forward Neural Networks

Feed-forward NNs are the most basic type of NNs. In simplest terms, they are a composition of non-linear functions. More precisely, a single layer of a feed-forward NN computes $\sigma(Wx + b)$, where $x \in \mathbb{R}^n$ is an input vector, $W \in \mathbb{R}^{m \times n}$ is called the weight matrix (or just weights), $b \in \mathbb{R}^m$ is called the bias, and σ is a non-linear "activation" function which acts element-wise on its input. When W is a dense matrix (mostly nonzero entries), we call the layer fully-connected.

A feed-forward NN is a function, $f : \mathbb{R}^n \to \mathbb{R}^d$, composed of k layers. We will denote

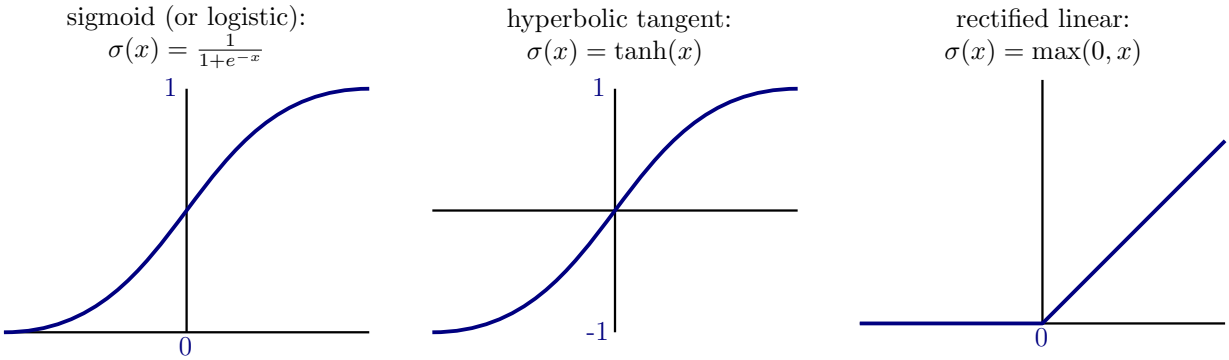


Figure A.3: Typical nonlinear activation functions used in neural networks.

$f_k(x)$ as the output of the k^{th} layer. We then have

$$f_k(x) = \sigma_k(W_k f_{k-1}(x) + b_k) \tag{A.9}$$

where W_k, b_k, and σ_k are the weights, bias, and activation function of the k^{th} layer. In this view, $f(x)$ is simply the output of the final layer. Typical activation functions can be found in Figure A.3. The rectified linear activation function is commonly used since it has several advantages compared to sigmoid functions such as: computational efficiency, a tendency to sparse representations, and less reliance on unsupervised pretraining [Glorot et al., 2011].

Figure A.4a shows a graphical representation of a two layer, fully-connected NN. The middle layer is called a hidden layer, since generally only the output of the final layer is observed. The bias vectors are typically not shown, but are implied. A downside of fully-connected NNs is that they are computationally expensive when the weight matrices are large. This problem makes them unsuitable for many real world signals. One solution to this problem is to replace the expensive matrix multiplies with more efficient convolution operations. This will be discussed in more depth in the next section.

One of the reasons NNs are widely used is because they are very flexible. Suppose we wish to construct a NN classifier for a dataset of d classes. All we must do is restrict the final layer of the network to produce a vector in $\mathbb{R}^d$. If we normalize the output using a softmax function (i.e. nonnegative elements that sum to one), we can interpret each entry as the probability that a given input belongs to a particular class. In other words, we assume

$$f(x) = P(c \,|\, x) \tag{A.10}$$

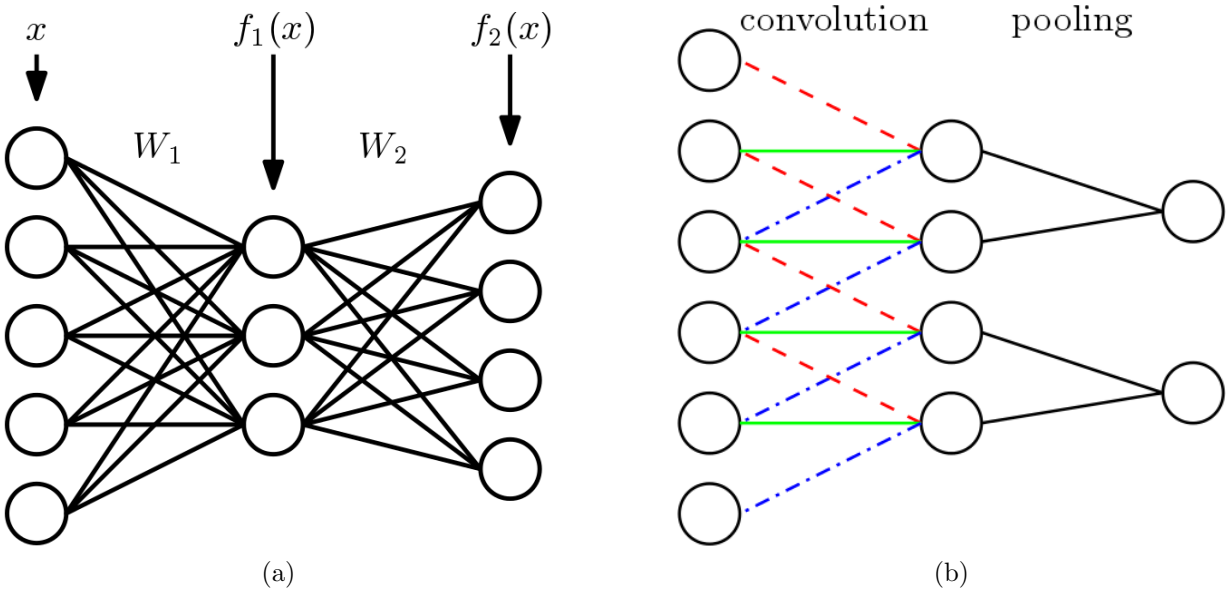


Figure A.4: Two graphical examples of neural networks. Each connection represents a single entry in the corresponding weight matrix. (a) A fully-connected neural network with a single hidden layer. Note that each node is connected to every other node in the previous and following layer. (b) A convolutional layer with a single filter. Convolutional filter weights with equal value are denoted using the same style. Zero valued weights are not shown.

where $P(c \mid x)$ is the probability distribution over the classes given a particular input x. A classification can be made by choosing the class with the highest probability.

Other tasks, such as regression and even signal generation, can be similarly performed by choosing an appropriate output structure. Of course, we have yet to explain how to set the parameters so that a NN outputs meaningful results. This will be covered in a later section.

A.2.2 Convolutional Neural Networks

Before discussing training, we will focus on a widely used class of NNs that have become popular in recent years. Convolutional neural networks (CNNs) are a variant of feed-forward NNs that are usually applied to vision applications (though they need not be). Their current formulation is generally attributed to Yann LeCun who wrote the first paper

on CNNs trained using backpropagation [LeCun et al., 1990]. CNNs share many similarities with the wavelet transform, such as filtering and downsampling. There are two main differences between CNNs and fully-connected NNs. The first is that each W_k is structured in such a way that multiplying W_k with a vector is equivalent to convolving the vector with a filter. Secondly, the output of this convolution step is pooled, which is similar to downsampling. Pooling has the effect of introducing a robustness to small variations in the signal [LeCun et al., 2010].

Let us consider the one-dimensional case. Suppose we have an input x that we wish to convolve with a k-length filter ψ. Furthermore, suppose we want to express this convolution as the matrix vector multiplication Wx. To do so we can set each row of W to shifted versions of ψ. Generally ψ is much shorter than x, and so the rows must be zero-padded. In other words, the i^{th} diagonal of W is set to $\psi[i]$:

$$W = \begin{bmatrix} \psi[0] & \psi[1] & \dots & \psi[k-1] & 0 & \dots & \dots & 0 \\ 0 & \psi[0] & \psi[1] & \dots & \psi[k-1] & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \psi[0] & \psi[1] & \dots & \psi[k-1] & 0 \\ 0 & \dots & \dots & 0 & \psi[0] & \psi[1] & \dots & \psi[k-1] \end{bmatrix} \tag{A.11}$$

The advantage of defining W this way is that Wx can be computed using a single convolution operation instead of an expensive matrix multiply. Furthermore, the number of parameters in W (i.e. its bandwidth) is drastically reduced compared to a dense matrix. Since weights are reused in each row of W, the maximum number of parameters in W is the number of columns. The reuse of weights in W is called weight sharing.

We are not restricted to using a single filter at each layer of the CNN. In fact, typically many filters are applied at each layer. A downside of applying multiple filters is that the output size is increased at each layer. To improve computational efficiency, a form of downsampling known as pooling is applied. Unlike regular downsampling where every i^{th} value of a signal is kept and the rest discarded, pooling typically applies an aggregate statistic over small nonoverlapping windows of the signal. Commonly the mean or max are used. See Figure A.4b for a graphical representation of a single layer of a CNN, and Figure A.5 for a full CNN.

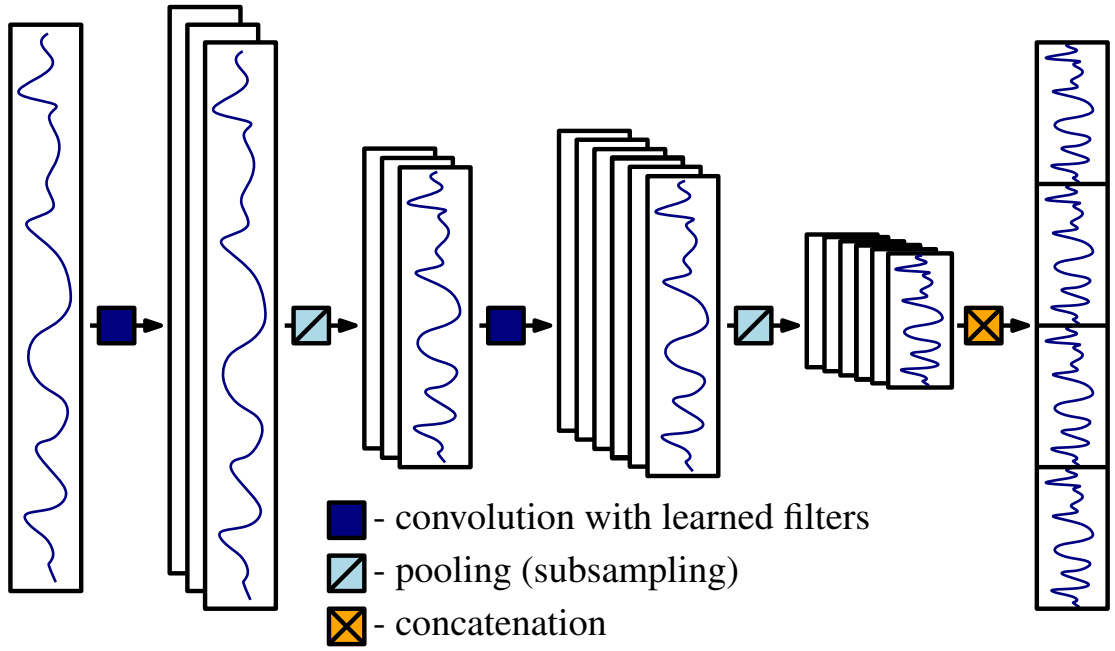


Figure A.5: Graphical representation of a convolutional neural network. Each layer of the network convolves its input with a set of learned filters to produce a set of feature maps. The feature maps are pooled to reduce dimension. The final set of feature maps are concatenated to form a single vector.

A.2.3 Backpropagation

The parameters of NN models (the weights and biases) are usually learned using stochastic gradient descent (SGD). Suppose we have a loss function $L(X;\theta)$ that is some measure of error given our parameters θ and data X. Gradient descent (GD) is an iterative optimization algorithm for choosing a value of θ. An initial value is chosen for θ (often randomly), then the following iterative update is applied:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(X;\theta^{(t)}) \tag{A.12}$$

where η is a positive step size and $\nabla L(X;\theta^{(t)})$ is the gradient of the loss function with respect to the current value of θ.

SGD is an online version of GD where a single randomly sampled datapoint is used to compute the gradient at each iteration. In practice, small random subsets of X (called minibatches) are used at each iteration instead of single datapoints in order to improve

stability and convergence [Ngiam et al., 2011]. Using minibatches is also useful for learning in large scale distributed environments [Dean et al., 2012]. Though SGD is only a first order method, second order methods are shown to provide little improvement, at least asymptotically [Bousquet and Bottou, 2008].

Calculating the gradient of the loss function is done using the backpropagation algorithm. The algorithm is an application of the chain rule for computing derivatives. It derives its name from the fact that the partial derivatives with respect to the parameters are computed from the last layer of the NN to the first layer. The practical application of the backpropagation algorithm for training NNs first appeared in the 1980s, with [Werbos, 1982, Rumelhart et al., 1985, LeCun, 1985] being notable examples.

The popularity of NNs has been helped by the release of several software frameworks. Some popular examples include: Theano [Theano Development Team, 2016], Torch [Collobert et al., 2011], and Tensorflow [Abadi et al., 2015]. These frameworks simplify the process of building and training NN models. One important feature of many of these frameworks is the use of automatic differentiation, which removes the requirement of explicitly formulating the gradient of the loss function. Since a NN is composed of many relatively simple operations, the gradient can be computed algorithmically. The idea of automatic differentiation is due to [Wengert, 1964].

A.2.4 Neural Network Depth

Theoretically, there is no limit to the depth of a NN. Of course, computational restrictions will limit the depth in practice. For example, the network sizes are often chosen so that the model parameters can fit entirely in memory. Optimization concerns also limit the depth of networks. Generally, the deeper the network, the more difficult it becomes to optimize the parameters. The problem is partly caused by vanishing or exploding gradients [Pascanu et al., 2013]. Since NNs are compositional functions, the gradient at a given layer is computed as a product of gradients at later layers (i.e. the chain rule for derivatives). Computationally, these large floating point products often vanish to zero (when the gradients are small) or explode towards infinity (when the gradients are large).

One method of dealing with exploding gradients is to simply set a maximum threshold value [Pascanu et al., 2013, Graves, 2013]. Any gradient whose magnitude is above this threshold is set to the threshold. The vanishing gradient is more difficult to remedy. Several approaches are used to combat this issue:

- Adding in skip connections (i.e. "shortcuts") has seen success in recent architectures

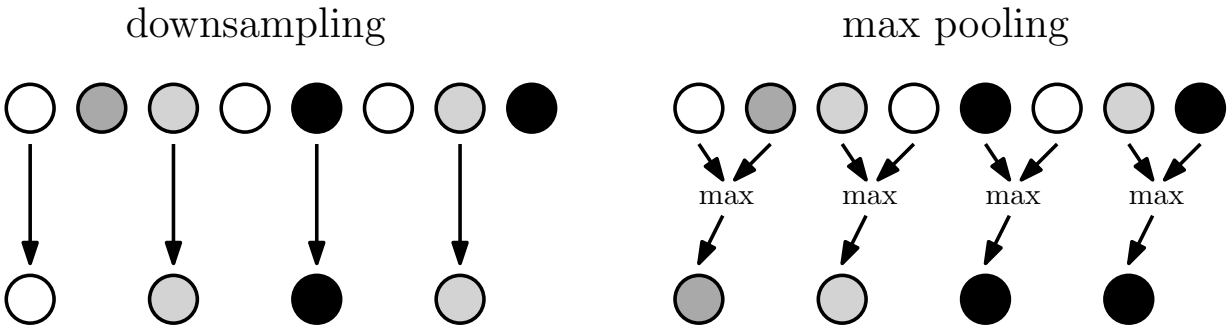


Figure A.6: A comparison of standard downsampling to max pooling. Signal values are represented by the intensity of the colours.

[Ronneberger et al., 2015, Long et al., 2015]. One explanation for their success is that they reduce the path distance from the final layer to earlier layers.

- Gradient information can be inserted at different layers of the network as in the GoogLeNet architecture [Szegedy et al., 2015].
- Long short-term memory networks are commonly used in the context of recurrent networks, and make use of gating functions to maintain the gradient [Hochreiter and Schmidhuber, 1997].
- Normalizing layer outputs to have zero mean and unit variance, called batch normalization, has also seen success in combatting the vanishing gradient problem [Ioffe and Szegedy, 2015].

A.2.5 A Wavelet Analysis of NN Architectures

We have now covered enough background to begin to answer one of the questions posed in the introduction of this thesis: Can we gain insight into NN architectures using concepts from wavelet theory? CNNs, and NNs in general, are often viewed as black box function approximators. In this section we will look at NNs from a signal processing point of view in hopes to gain some insight into their inner workings. In particular, we will perform a case study of CNNs used for semantic segmentation.

Semantic segmentation is concerned with finding regions of similar structure in images. For example, we may wish to label the pixels of an image that correspond to roadways or traffic signs for use in an automated driving system. CNNs have shown to be a successful

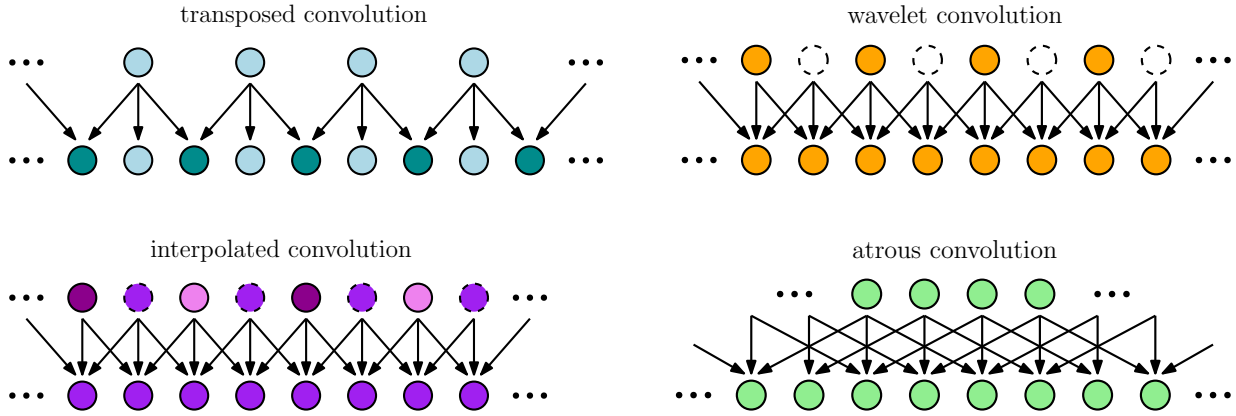


Figure A.7: A comparison of different upsampling techniques. Dashed circles indicate inserted values. The darker circles in the transposed convolution figure represent checkerboard artifacts caused by uneven filter overlap.

method used for semantic segmentation [Long et al., 2015]. One limitation of traditional CNNs is that the pooling operator removes some of the positional information of the features. This is problematic for the task since the end goal is to have a fine-grain (pixel level) labelling of the input image.

How might we explain this limitation using the concepts from Chapter 2? Let us begin by considering the pooling operation. Pooling is accomplished by applying an aggregate statistic over small windows of a signal, and then downsampling the result. Figure A.6 shows a comparison of max pooling to regular downsampling. In both cases the signals are downsampled by a factor of two. Recall that downsampling reduces the Nyquist rate of the signal, limiting the maximum frequency that can be represented. Thus, when pooling is applied at a particular layer, we necessarily lose some bandwidth and possibly introduce aliasing. This issue is addressed in the wavelet transform by careful filter construction. Downsampling is only applied after the signal has been filtered by either a highpass or lowpass filter. Doing so allows us to downsample without losing any information. However, CNNs do not typically have any constraints on the learned filters, and so we cannot guarantee that information at different bandwidths is retained. Furthermore, the intermediate representation may have lower dimension than the input depending on the number of filters used and the amount of pooling performed. This may explain why traditional CNNs are not able to recover fine-grain (i.e. high frequency) information from their intermediate representations.

To overcome this limitation, Long et al. allow their network to pass forward information

from intermediate layers [Long et al., 2015]. We can view this as a method of preserving information at multiple scales. Other methods of preserving multi-scale information can be found in [Hariharan et al., 2015, Ronneberger et al., 2015, Lin et al., 2016]. These methods share similarities with the wavelet transform, which considers output from every "layer" of filtering. The advantage of the wavelet approach is that there is no data redundancy and no signal features are lost. At each filtering step, the sampling rate is reduced by a factor of two, which results in a transform that has the same length as the input. The final output retains features at different scales in a natural way.

Another issue with using CNNs is one of upsampling. Pooling operations reduce the resolution of the signal at every layer. In order to have output with the same resolution as our input, we must have a way of reversing the reduced resolution. One method of upsampling is called transposed convolution (also fractionally strided convolution, or deconvolution) [Long et al., 2015]. Unfortunately, transposed convolution tends to introduce checkerboard artifacting due to overlapping convolutional kernels (see Figure A.7) [Odena et al., 2016].

One approach to avoid artifacting is to upsample the signal prior to performing a convolution. This approach is used in the wavelet transform by inserting zeros at every second index. A similar method has been used in [Hariharan et al., 2015], where the authors make use of bilinear interpolation to upsample the signal. This may seem like a perfectly reasonable approach to increasing the feature dimension. However, it is not quite correct. The goal of upsampling is to increase the sampling rate of a signal without introducing artifacts. This came up in the proof of the sampling theorem in Chapter 2. We saw that the correct way to upsample a discrete signal is to first convolve it with a sinc function, which allows us to perfectly reconstruct the continuous signal. The continuous signal can then be resampled at the desired sampling rate.

Another popular alternative to transposed convolution is called atrous convolution (also strided or dilated convolution) [Chen et al., 2016]. This method makes use of large sparse filters (i.e. filters with "holes"), which allow for upsampling without an increase to computation. This approach was originally developed in the wavelet literature for an undecimated version of the wavelet transform known as the "algorithme à trous" [Holschneider et al., 1990].

Through the discussion in this section, we have seen that there is significant overlap between the theory and methods from Chapter 2 and NN architectures. Taking recent CNN-based approaches to semantic segmentation as an example, we saw that several ideas were borrowed or rediscovered from wavelet analysis, such as atrous convolution and adding

Derived from the French word "trous" meaning holes.

connections to preserve multiscale information. Furthermore, some solutions (e.g. bilinear interpolation) may be better handled by techniques from signal processing. We hope that we have convinced the reader that taking a signal processing view of CNNs provides some intuition into the design and limitations of current architectures, as well as the design of future models.

Appendix B

Supplemental Figures

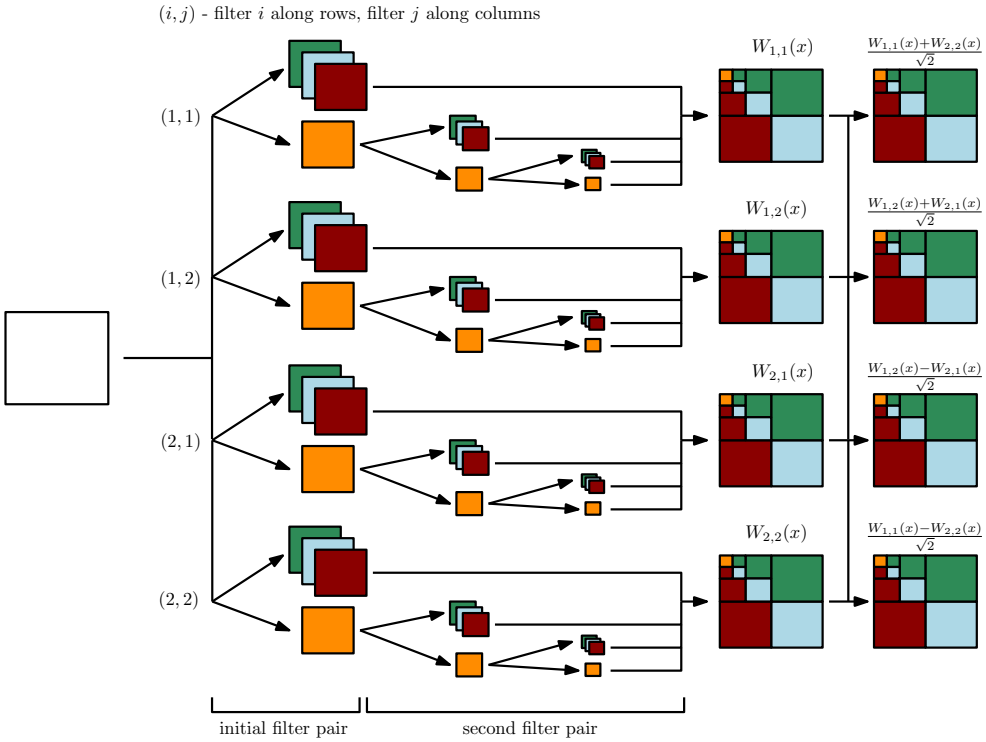


Figure B.1: The complex dual-tree wavelet transform network. Each $W_{i,j}$ is computed as in Equations 6.4–6.7.

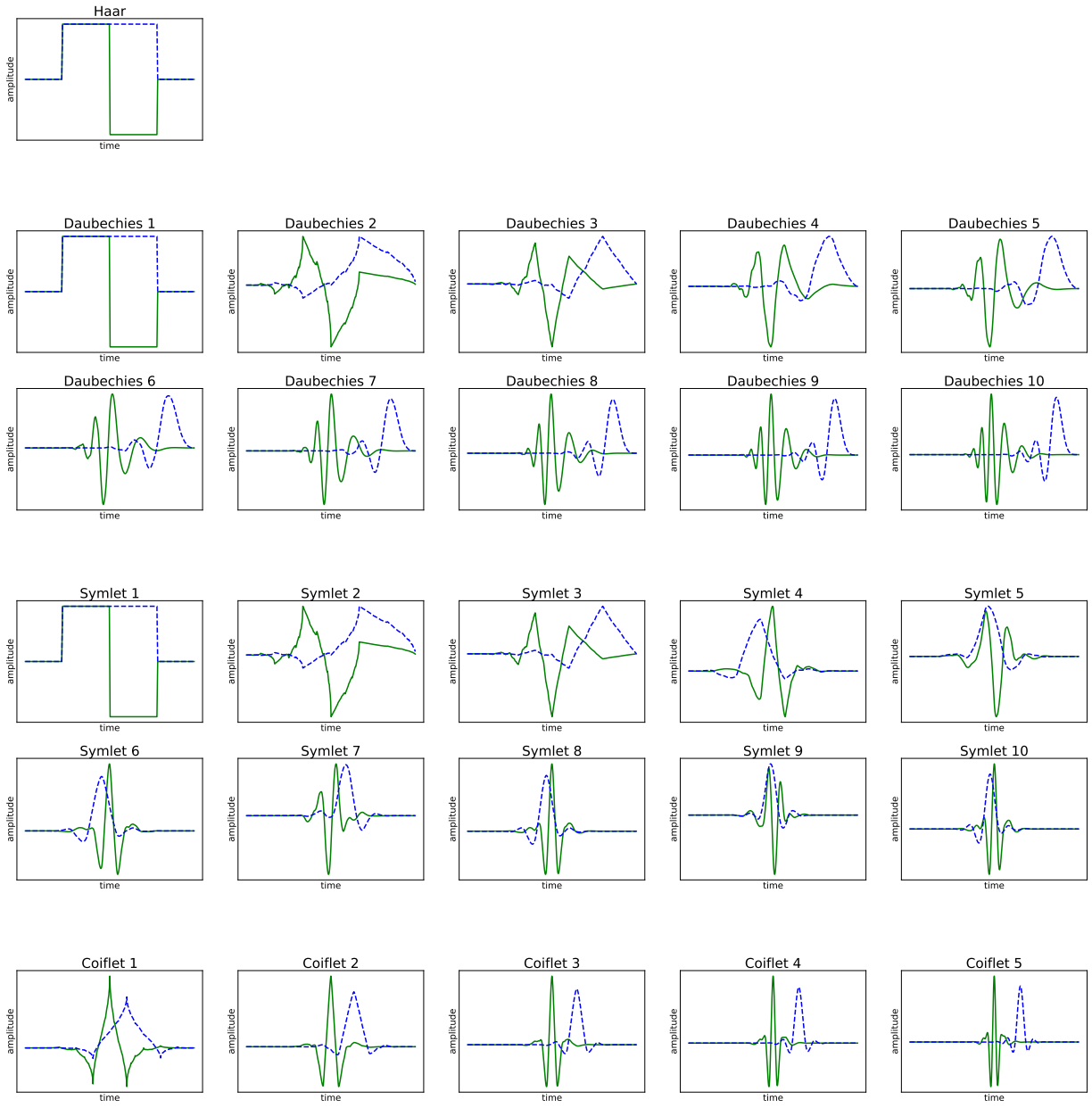


Figure B.2: Various traditional wavelet (solid) and scaling (dashed) functions.

Figure B.3: The cropped grayscale Kodak image dataset.

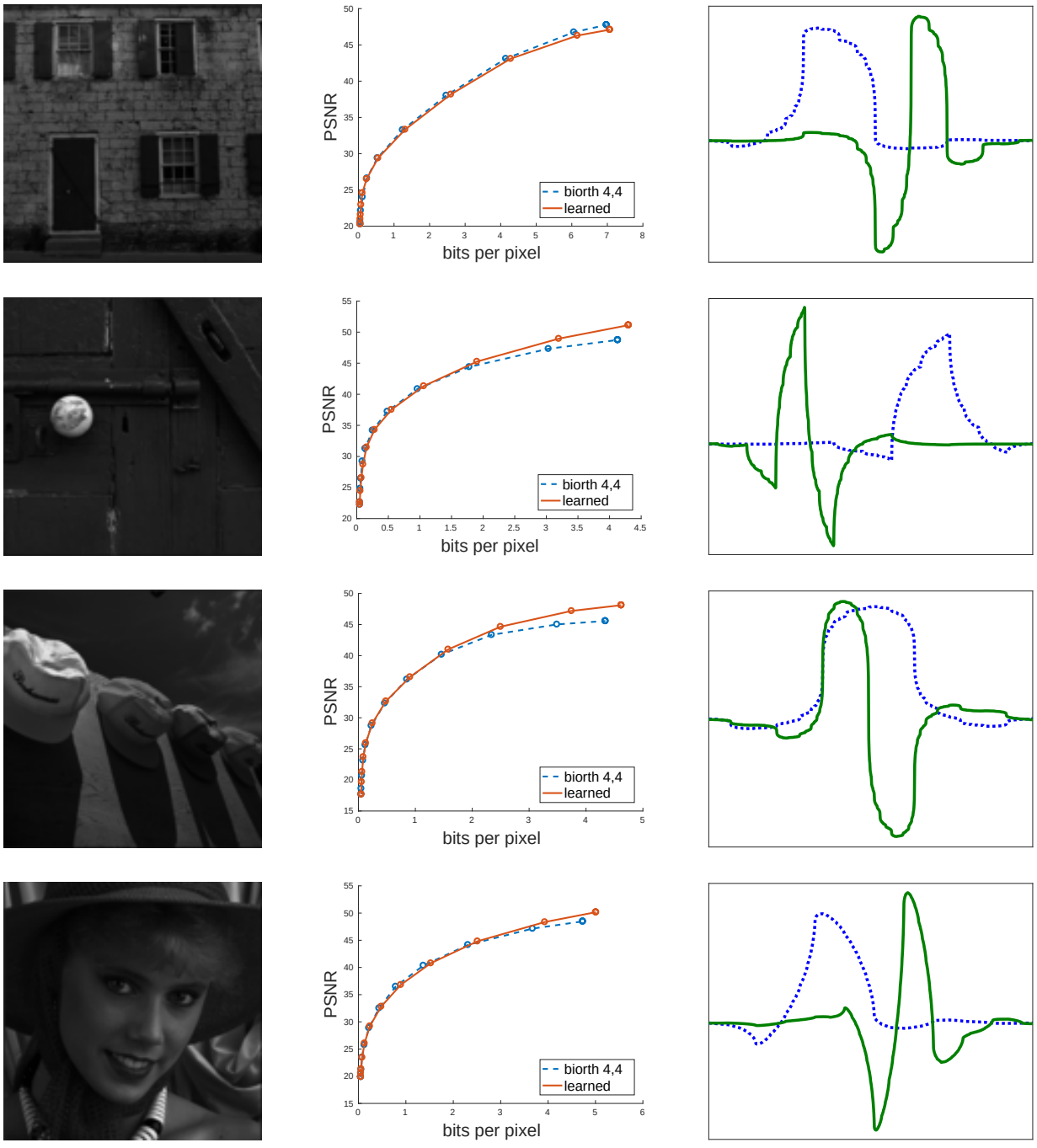


Figure B.4: Compression results for the Kodak image dataset (images 1–4).

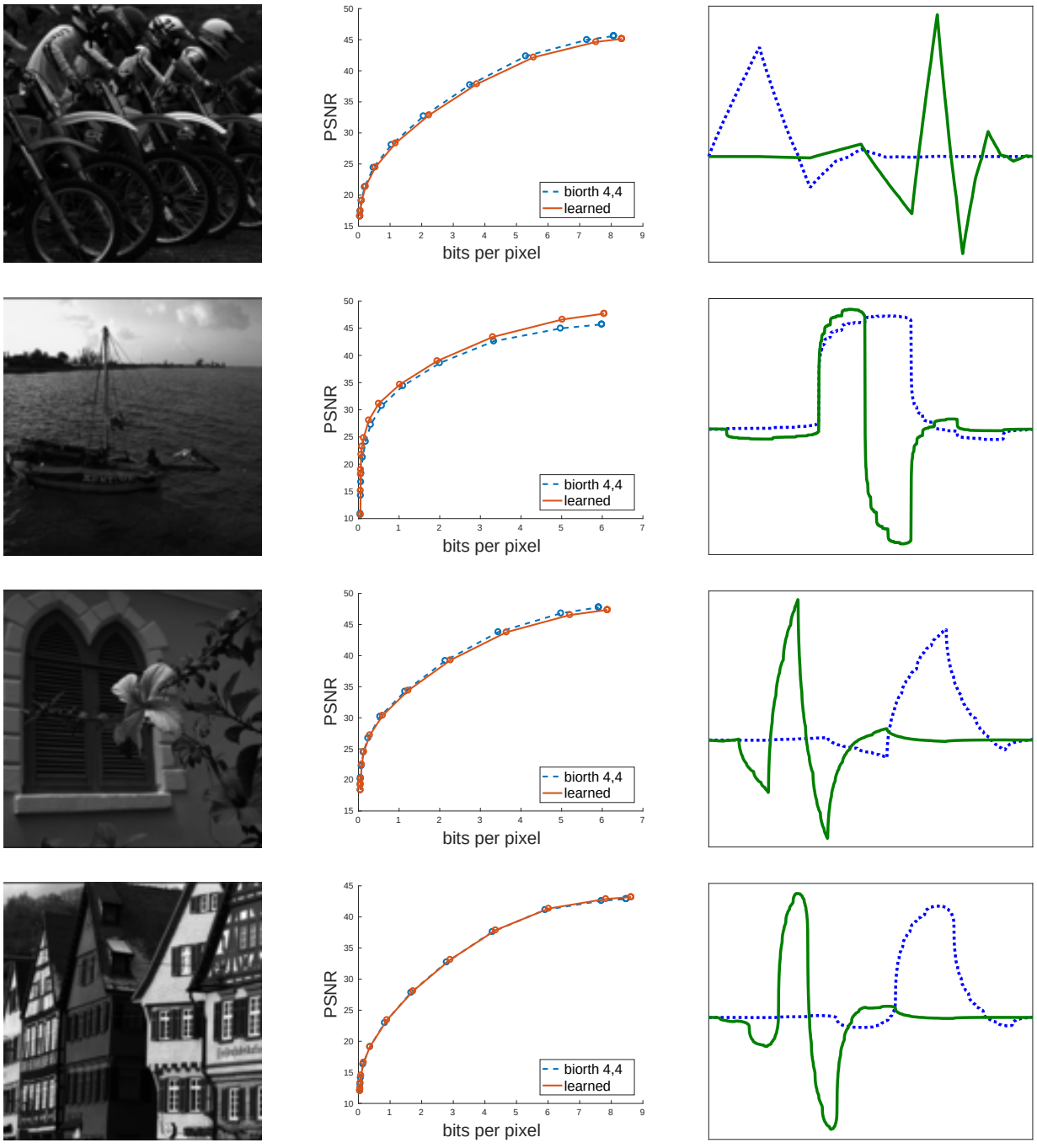


Figure B.5: Compression results for the Kodak image dataset (images 5–8).

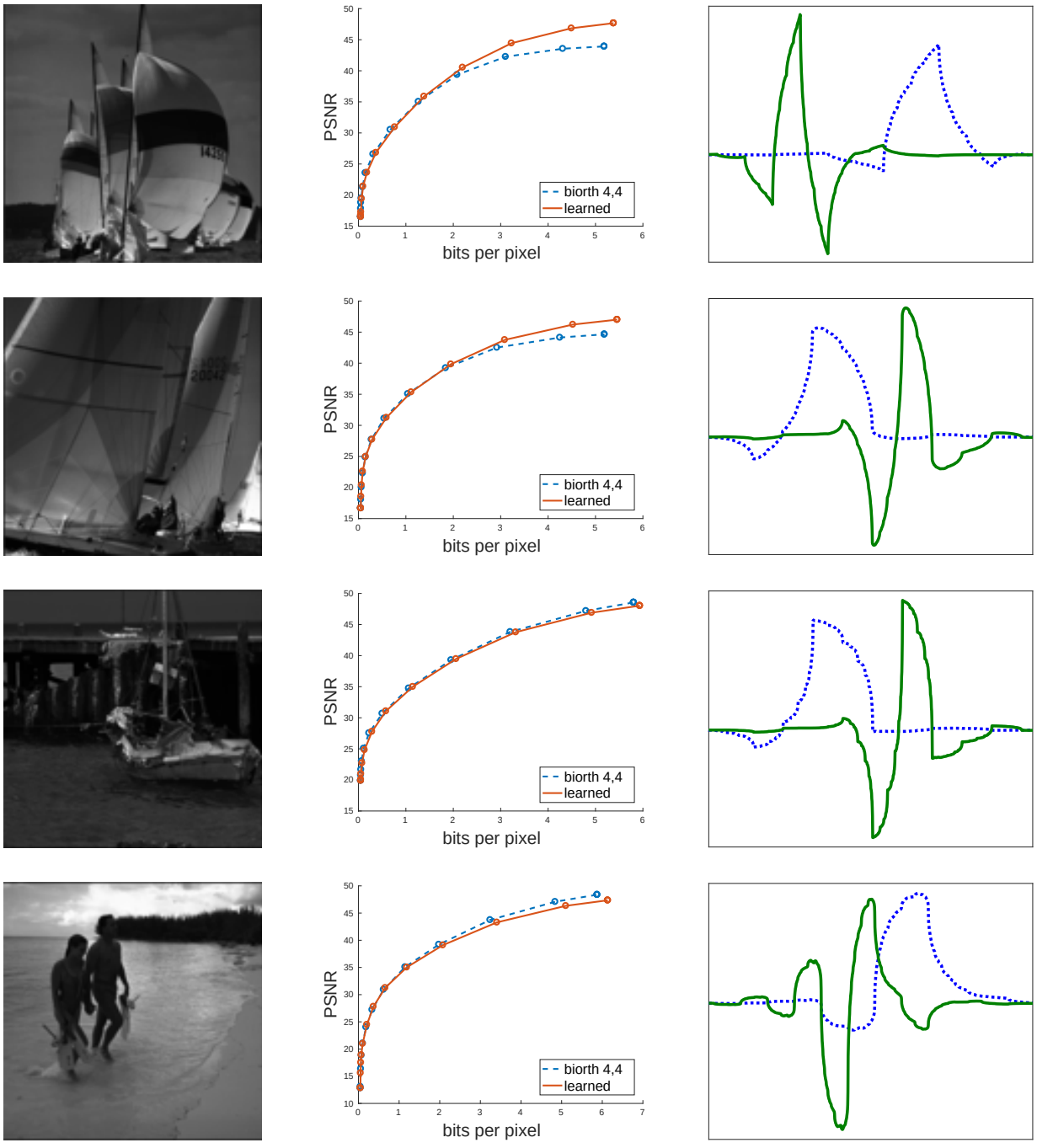


Figure B.6: Compression results for the Kodak image dataset (images 9–12).

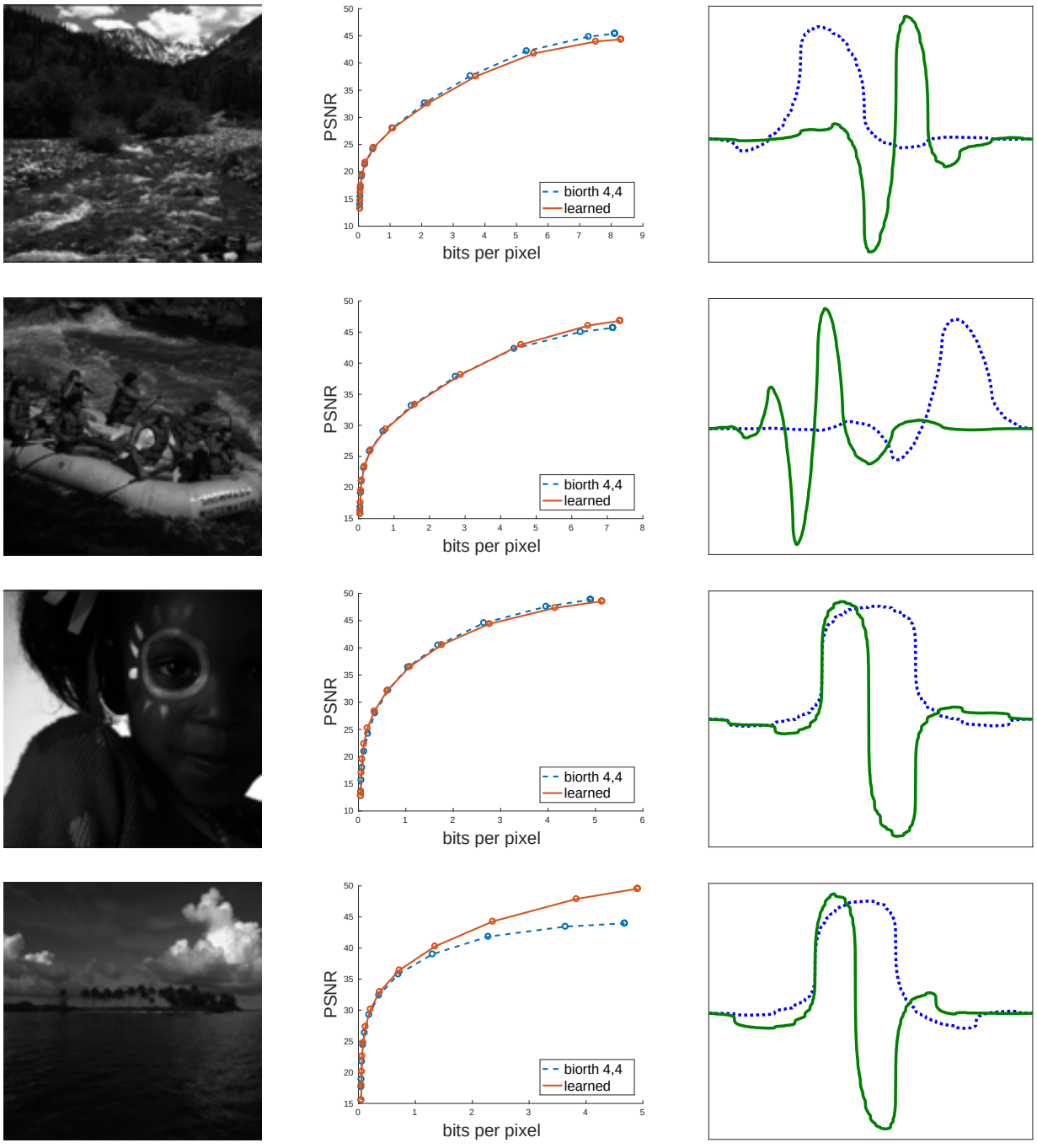


Figure B.7: Compression results for the Kodak image dataset (images 13–16).

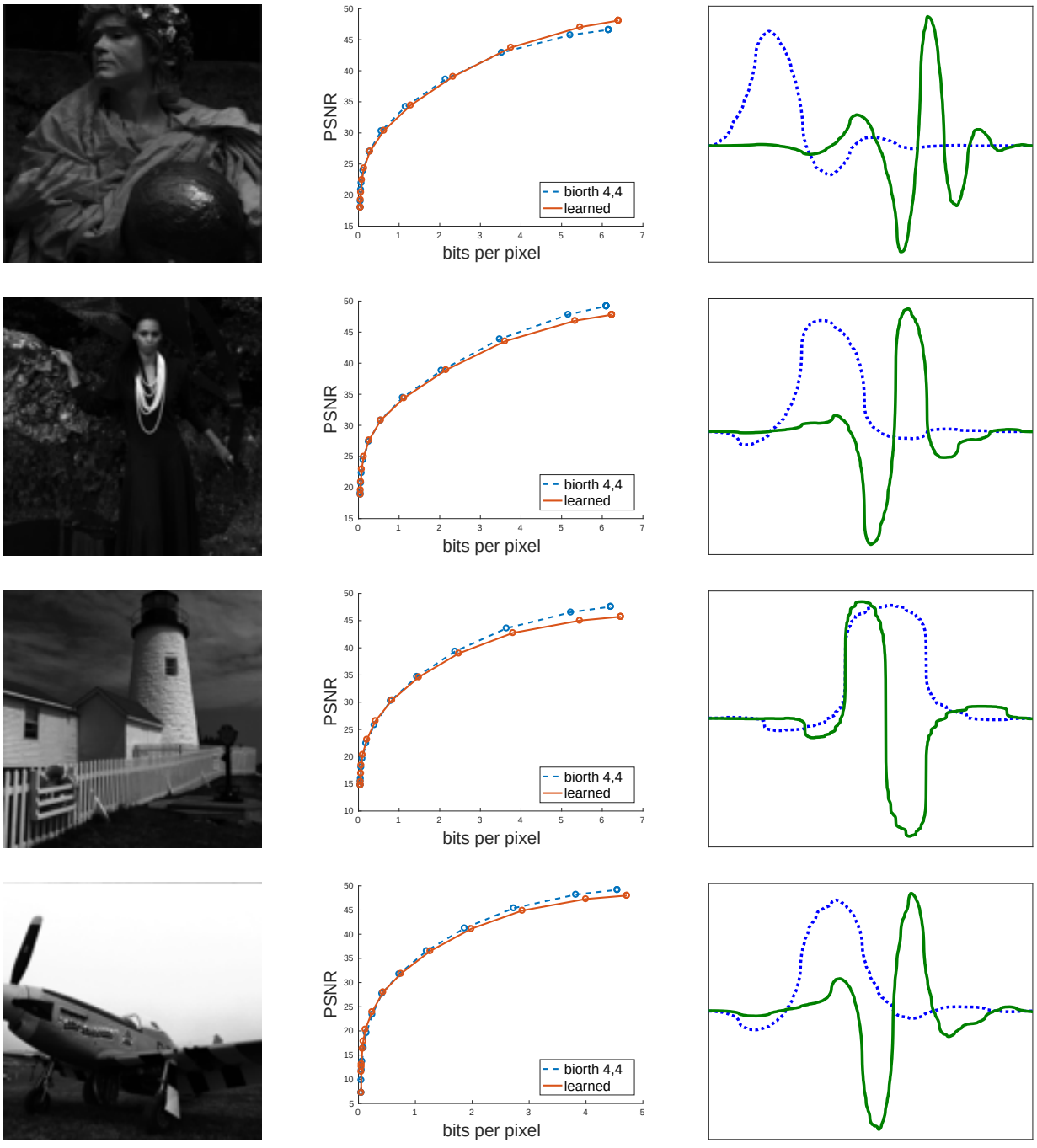


Figure B.8: Compression results for the Kodak image dataset (images 17–20).

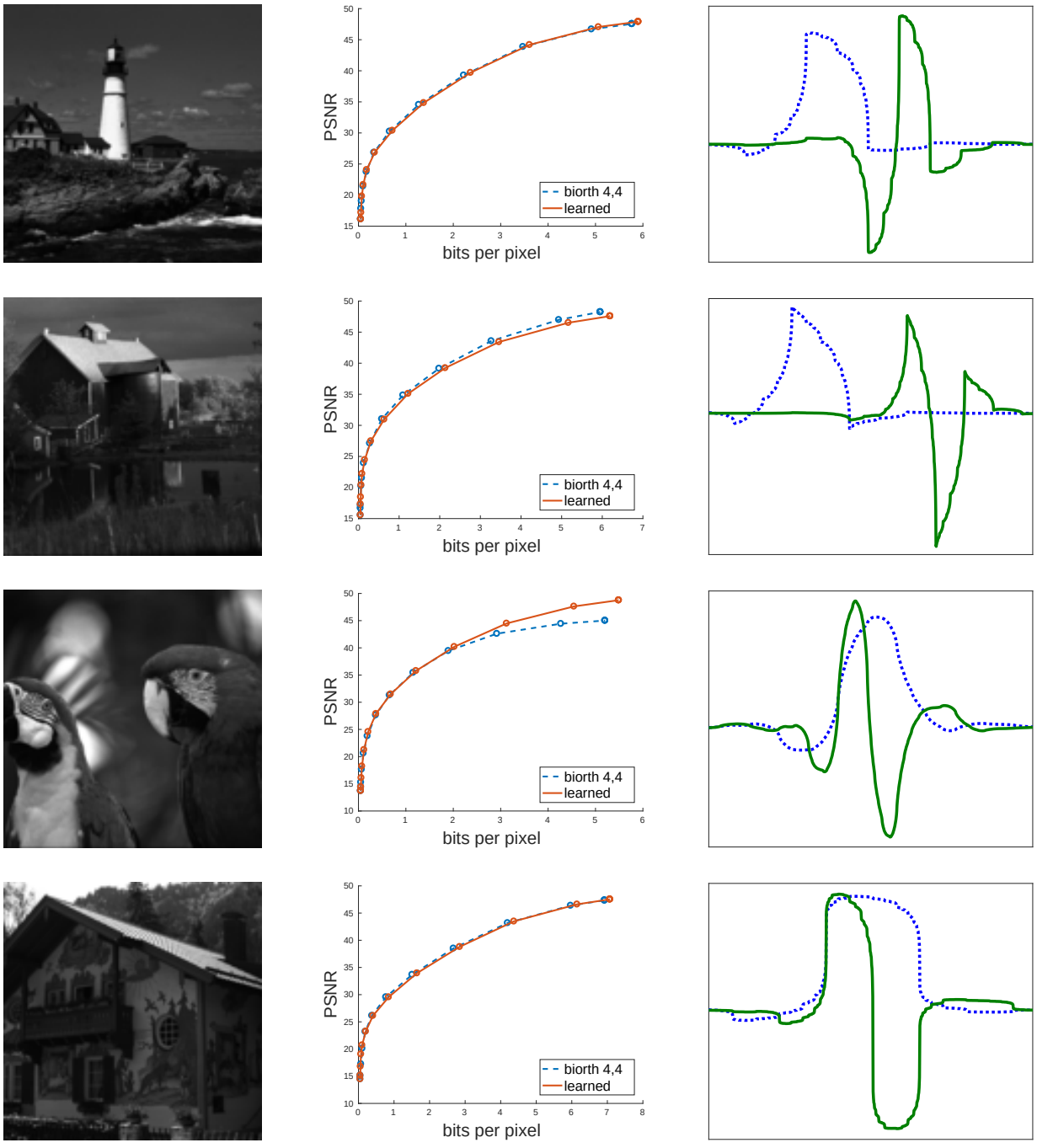


Figure B.9: Compression results for the Kodak image dataset (images 21–24).

Appendix C

Parameterized Wavelets

Scaling filter parameterization using method from [Selesnick, 1997].

Length eight filter (three free parameters):

$$
\begin{aligned}
t3 =& 1/4\,\pi - t0 - t1 - t2 \\
h_1 =& \cos(t3)\cos(t2)\cos(t1)\cos(t0) \\
h_2 =& \cos(t3)\cos(t2)\cos(t1)\sin(t0) \\
h_3 =& -\cos(t3)\cos(t2)\sin(t1)\sin(t0) - \cos(t3)\sin(t2)\sin(t1)\cos(t0) \\
& -\sin(t3)\sin(t2)\cos(t1)\cos(t0) \\
h_4 =& \cos(t3)\cos(t2)\sin(t1)\cos(t0) \; -\cos(t3)\sin(t2)\sin(t1)\sin(t0) \\
& -\sin(t3)\sin(t2)\cos(t1)\sin(t0) \\
h_5 =& -\cos(t3)\sin(t2)\cos(t1)\sin(t0) + \sin(t3)\sin(t2)\sin(t1)\sin(t0) \\
& -\sin(t3)\cos(t2)\sin(t1)\cos(t0) \\
h_6 =& \cos(t3)\sin(t2)\cos(t1)\cos(t0) - \sin(t3)\sin(t2)\sin(t1)\cos(t0) \\
& -\sin(t3)\cos(t2)\sin(t1)\sin(t0) \\
h_7 =& -\sin(t3)\cos(t2)\cos(t1)\sin(t0) \\
h_8 =& \sin(t3)\cos(t2)\cos(t1)\cos(t0)
\end{aligned}
$$

Length ten filter (four free parameters):

$$
\begin{aligned}
t4 =& 1/4\,\pi - t0 - t1 - t2 - t3 \\
h_1 =& \cos(t4)\cos(t3)\cos(t2)\cos(t1)\cos(t0) \\
h_2 =& \cos(t4)\cos(t3)\cos(t2)\cos(t1)\sin(t0) \\
h_3 =& -\cos(t4)\cos(t3)\cos(t2)\sin(t1)\sin(t0) - \cos(t4)\cos(t3)\sin(t2)\sin(t1)\cos(t0) \\
& -\cos(t4)\sin(t3)\sin(t2)\cos(t1)\cos(t0) - \sin(t4)\sin(t3)\cos(t2)\cos(t1)\cos(t0) \\
h_4 =& \cos(t4)\cos(t3)\cos(t2)\sin(t1)\cos(t0) - \cos(t4)\cos(t3)\sin(t2)\sin(t1)\sin(t0) \\
& -\cos(t4)\sin(t3)\sin(t2)\cos(t1)\sin(t0) - \sin(t4)\sin(t3)\cos(t2)\cos(t1)\sin(t0) \\
h_5 =& -\cos(t4)\cos(t3)\sin(t2)\cos(t1)\sin(t0) + \cos(t4)\sin(t3)\sin(t2)\sin(t1)\sin(t0) \\
& -\cos(t4)\sin(t3)\cos(t2)\sin(t1)\cos(t0) + \sin(t4)\sin(t3)\cos(t2)\sin(t1)\sin(t0) \\
& +\sin(t4)\sin(t3)\sin(t2)\sin(t1)\cos(t0) - \sin(t4)\cos(t3)\sin(t2)\cos(t1)\cos(t0) \\
h_6 =& \cos(t4)\cos(t3)\sin(t2)\cos(t1)\cos(t0) - \cos(t4)\sin(t3)\sin(t2)\sin(t1)\cos(t0) \\
& -\cos(t4)\sin(t3)\cos(t2)\sin(t1)\sin(t0) - \sin(t4)\sin(t3)\cos(t2)\sin(t1)\cos(t0) \\
& +\sin(t4)\sin(t3)\sin(t2)\sin(t1)\sin(t0) - \sin(t4)\cos(t3)\sin(t2)\cos(t1)\sin(t0) \\
h_7 =& -\cos(t4)\sin(t3)\cos(t2)\cos(t1)\sin(t0) + \sin(t4)\sin(t3)\sin(t2)\cos(t1)\sin(t0) \\
& +\sin(t4)\cos(t3)\sin(t2)\sin(t1)\sin(t0) - \sin(t4)\cos(t3)\cos(t2)\sin(t1)\cos(t0) \\
h_8 =& \cos(t4)\sin(t3)\cos(t2)\cos(t1)\cos(t0) - \sin(t4)\sin(t3)\sin(t2)\cos(t1)\cos(t0) \\
& -\sin(t4)\cos(t3)\sin(t2)\sin(t1)\cos(t0) - \sin(t4)\cos(t3)\cos(t2)\sin(t1)\sin(t0) \\
h_9 =& -\sin(t4)\cos(t3)\cos(t2)\cos(t1)\sin(t0) \\
h_{10} =& \sin(t4)\cos(t3)\cos(t2)\cos(t1)\cos(t0)
\end{aligned}
$$

Appendix D

Dual-tree Filter Values

In this section we include the filter values corresponding to the impulse response figures from Chapter 6.

D.1 Real dual-tree filters

h'_1	h_1
+7.538e-04	+3.207e-02
−6.960e-02	−4.052e-03
−4.274e-02	−5.705e-02
+4.233e-01	+2.732e-01
+7.917e-01	+7.357e-01
+4.234e-01	+5.615e-01
−4.291e-02	+5.348e-03
−6.972e-02	−1.456e-01
+2.573e-04	−5.864e-03
+4.025e-04	+2.406e-02

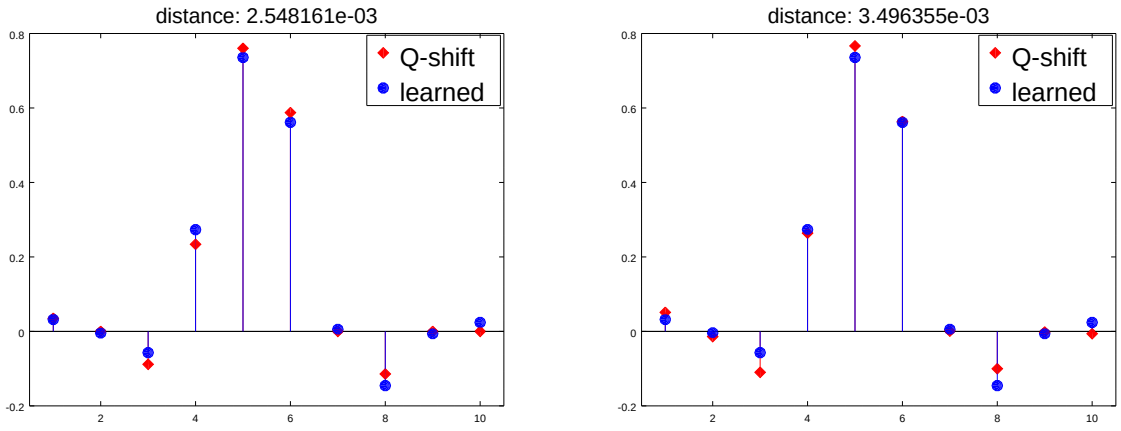
Filters from Figure 6.12a.

h'_1	h_1
−1.010e-02	+1.970e-02
+3.534e-02	−3.822e-02
+2.407e-02	−1.078e-01
−8.123e-02	+2.241e-01
+2.722e-01	+7.332e-01
+7.981e-01	+6.144e-01
+5.143e-01	+5.249e-02
−4.624e-02	−1.302e-01
−9.701e-02	+9.322e-03
−5.738e-04	+3.693e-02

Filters from Figure 6.14a.

h'_1	h_1
+1.956e-02	+1.970e-02
−5.031e-02	−3.822e-02
−7.082e-02	−1.078e-01
+4.035e-01	+2.241e-01
+8.096e-01	+7.332e-01
+4.028e-01	+6.144e-01
−7.107e-02	+5.249e-02
−4.922e-02	−1.302e-01
+1.958e-02	+9.322e-03
−1.094e-05	+3.693e-02

Filters from Figure 6.14c.



(a) (b)

Figure D.1: Comparison of Kingsbury's Q-shift (a) 6 tap, and (b) 10 tap filters with the learned h filter from Figure 6.12a.

D.2 Complex dual-tree filters

h'_1	h_1
+1.017e-02	+1.914e-02
+3.677e-02	−1.947e-02
−3.689e-02	−1.469e-01
−5.913e-02	+3.960e-02
+3.982e-01	+6.020e-01
+8.018e-01	+7.357e-01
+4.223e-01	+2.393e-01
−7.171e-02	−8.599e-02
−8.616e-02	−5.614e-03
+4.248e-05	+3.850e-02

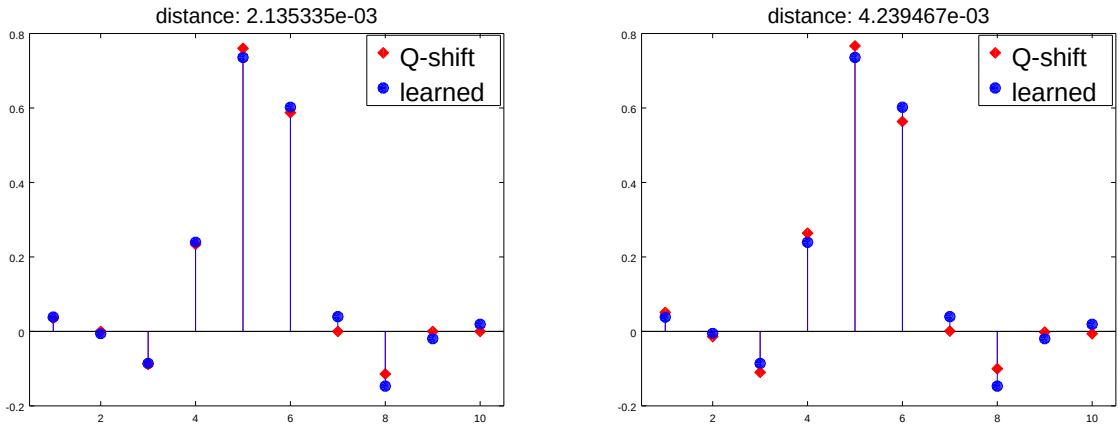
Filters from Figure 6.12b.

h'_1	h_1
+1.523e-02	+2.826e-02
−5.440e-02	−1.266e-02
−6.360e-02	−1.225e-01
+4.075e-01	+1.364e-01
+8.054e-01	+6.800e-01
+4.075e-01	+6.807e-01
−6.346e-02	+1.358e-01
−5.442e-02	−1.220e-01
+1.547e-02	−1.340e-02
+1.666e-04	+2.921e-02

Filters from Figure 6.14b.

h'_1	h_1
+1.903e-02	−8.475e-02
−4.904e-02	−8.946e-02
−7.057e-02	+3.345e-01
+4.024e-01	+7.659e-01
+8.090e-01	+5.145e-01
+4.020e-01	−1.701e-02
−7.052e-02	−9.557e-02
−4.888e-02	+5.711e-02
+1.931e-02	+3.971e-02
+5.240e-04	−9.468e-03

Filters from Figure 6.14d.



(a) (b)

Figure D.2: Comparison of Kingsbury's Q-shift (a) 6 tap, and (b) 10 tap filters with the learned h filter from Figure 6.12b. Note that h has been reversed.

h'_1	h_1
+1.640e-02	−1.571e-03
−5.179e-02	−2.007e-03
−6.775e-02	+2.725e-02
+4.049e-01	−2.220e-03
+8.068e-01	−1.437e-01
+4.050e-01	+2.654e-03
−1.189e-01	+5.598e-01
+1.654e-02	+7.526e-01
+3.910e-04	+2.865e-01
	−7.620e-02
	−2.374e-02
	+3.958e-02
	+2.277e-03
	−7.738e-03

Learned length 10 h' and length 14 h filters.

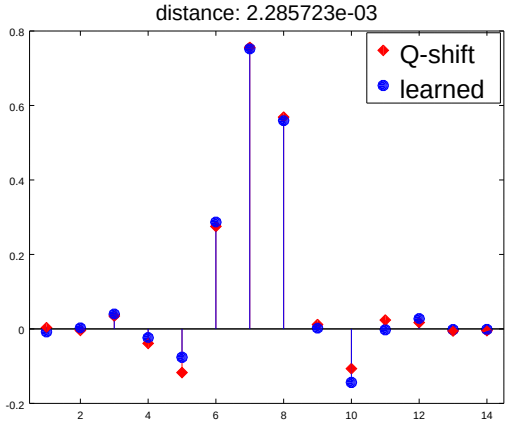


Figure D.3: Comparison of Kingsbury's Q-shift 14 tap filter with the learned length 14 h. Note that h has been reversed.

h'_1	h_1
+1.952e-02	+4.335e-04
−4.506e-02	+3.752e-04
−6.062e-02	−6.027e-03
+4.069e-01	+3.851e-03
+8.066e-01	+3.955e-02
+4.043e-01	−2.870e-02
−7.323e-02	−8.351e-02
−6.027e-02	+2.887e-01
+1.452e-02	+7.560e-01
−9.868e-05	+5.569e-01
	+3.984e-03
	−1.344e-01
	−1.460e-03
	+2.056e-02
	−2.688e-03
	−5.462e-04
	−1.798e-03
	+5.302e-05

Learned length 10 h' and length 18 h filters (λ_3 was increased to 8e-4).

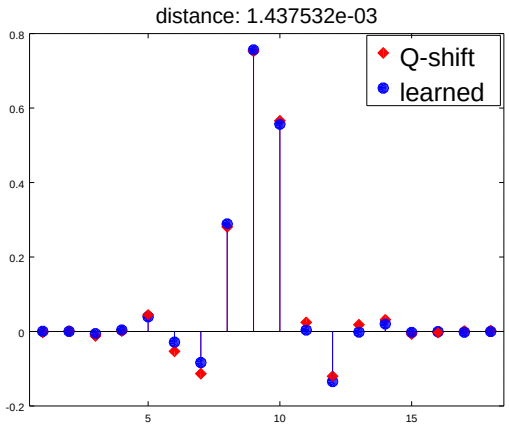


Figure D.4: Comparison of Kingsbury's Q-shift 18 tap filter with the learned length 18 h.