Learning Continuous Models for Estimating Intrinsic Component Images

by

Marshall Friend Tappen

B.S. Computer Science, Brigham Young University, 2000
S.M. Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2002

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2006
May 2006

Author .....
Department of Electrical Engineering and Computer Science
May 26, 2006

Certified by .....
Edward H. Adelson
Professor of Vision Science
Thesis Supervisor

Certified by .....
William T. Freeman
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Learning Continuous Models for Estimating Intrinsic Component Images

by

Marshall Friend Tappen

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

The goal of computer vision is to use an image to recover the characteristics of a scene, such as its shape or illumination. This is difficult because an image is the mixture of multiple characteristics. For example, an edge in an image could be caused by either an edge on a surface or a change in the surface's color. Distinguishing the effects of different scene characteristics is an important step towards high-level analysis of an image.

This thesis describes how to use machine learning to build a system that recovers different characteristics of the scene from a single, gray-scale image of the scene. The goal of the system is to use the observed image to recover images, referred to as Intrinsic Component Images, that represent the scene's characteristics. The development of the system is focused on estimating two important characteristics of a scene, its shading and reflectance, from a single image. From the observed image, the system estimates a shading image, which captures the interaction of the illumination and shape of the scene pictured, and an albedo image, which represents how the surfaces in the image reflect light. Measured both qualitatively and quantitatively, this system produces state-of-the-art estimates of shading and albedo images. This system is also flexible enough to be used for the separate problem of removing noise from an image.

Building this system requires algorithms for continuous regression and learning the parameters of a Conditionally Gaussian Markov Random Field. Unlike previous work, this system is trained using real-world surfaces with ground-truth shading and albedo images. The learning algorithms are designed to accomodate the large amount of data in this training set.

Thesis Supervisor: Edward H. Adelson
Title: Professor of Vision Science

Thesis Supervisor: William T. Freeman
Title: Professor of Computer Science and Engineering

Acknowledgments

I have been fortunate to be advised by Bill Freeman and Ted Adelson. They have taught me much about both computer vision and how to do research itself. I am grateful for the time that they have spent discussing ideas, revising papers, commenting on talks, and helping me get out of ruts in my research.

I appreciate the time that Michael Collins, the third member of my thesis committee, spent reading this thesis carefully and making suggestions for its improvements.

Many thanks go to the members of the Vision and Learning Group and Perceptual Sciences group: Bryan, Antonio, Erik, Ce, Ron, Barun, Yuanzhen, Lavanya, and Kevin. They have patiently listened to and commented on my various hare-brained ideas over the years.

My parents, John and Melanie Tappen, and my mother-in-law, Grace Chan, have been tremendous supports, both emotionally and, occasionally, financially.

My greatest thanks goes to my family. My sons, John and Jeffrey, were both born at MIT. Whether my days at school went well or poorly, I could always count on an enthusiastic reception at home. It is a humbling experience to hear a five-year-old pray for the success of his father at school. In the end, they have shaped my graduate experience more than anything else.

My wife, Joy, has continually loved and supported me. During my studies at MIT, she has

- Spent six years in small, student apartments, with two active boys.
- Made my small student stipend stretch amazingly far.
- Had to move every time that she has been expecting a child.

Her constant love and support have made life and graduate school a wonderful experience. I am blessed to have such an amazing partner for life and the eternities.

This work was supported by an NDSEG Fellowship from the Department of Defense and grants from Shell Oil, the National Geospatial-Intelligence Agency, NGA-NEGI, the National Science Foundation, the NIH, and the Nippon Telegraph and Telephone Corporation as part of the NTT/MIT Collaboration Agreement.

Contents

List of Figures

List of Tables

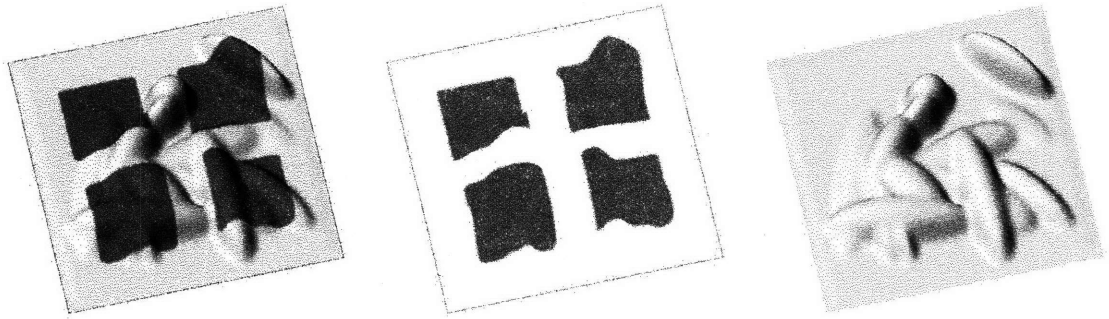
Chapter 1

Introduction

Figure 1-1: The world is a complicated place. Even a picture of a typical scene exhibits the many characteristics of a scene that affect its appearance.

An often underappreciated characteristic of the world is that it is a complicated place. Viewing an image of a real world scene, such as the image shown in Figure 1-1, shows how many different characteristics of the scene contribute to the image of that scene. The illumination of the scene and the shape of the fabric affect the appearance of the folds and shadows of the fabric. The collar of the shirt and the stripes on the socks are caused by changes in how the fabrics reflect light. In addition, there can be noise in the camera from the sensor.

The goal of computer vision is to analyze images such as Figure 1-1 and recover characteristics of the scene. Characteristics of interest include the shape of surfaces



(a) Image (b) Reflectance Image (c) Shading Image

Figure 1-2: Example of shading and albedo intrinsic images. (a) Image of a scene (b) The albedo intrinsic image. This image contains only the reflectance of each point. (c) Shading intrinsic image. The shading image results from the interaction between the illumination of the scene and the shape of the surface.

[80], the reflectance of the surfaces, and the objects present in the scene [73]. Unfortunately, the image captured by the camera is the combination of these characteristics. Recovering the characteristics of a scene from an image requires the ability to distinguish and represent each characteristic's effect on the image. For example, recovering the shape of a surface from an image requires the ability to distinguish changes caused by shading from changes in the surface's reflectance.

This thesis will focus on representing and recovering the characteristics of a scene with an image-based representation, which I will refer to as "Intrinsic Component Images." This representation is motivated by the fact that an image of a scene can be represented as a composition of a number of different images. At the image formation level, the image can be represented as the sum of a noise image and the true image of the scene. The scene image itself can also be represented as the composition of images that describe the characteristics of the surfaces in the scene. Figure 1-2 shows how an image of a painted surface can be represented as the product of an image representing just the shading of the surface and a second image representing the albedo of each point on the surface.

This thesis describes a system that uses a single image of a scene to recover images that represent the characteristics of the scene. The machinery presented in this thesis is flexible enough to be used to recover different scene characteristics. In particular, I will focus on the problem of estimating the shading and albedo of a scene and the problem of estimating a clean "scene" image from a noisy observation. The system described in this thesis achieves state-of-the-art performance on the shading

and albedo task and is competitive with state-of-the-art algorithms for denoising images.

Below, Section 1.1 discusses relevant background information, including intrinsic images and the definition of shading and albedo images. Section 1.2 discusses previous work, focusing on the task of estimating shading and albedo images.

The basic strategy for estimating images is described in Section 1.3. Section 1.6 describes the basic notation that will be used throughout this thesis and Section 1.5 gives a roadmap for the remaining chapters.

1.1 Background

This section begins by describing the intrinsic image representation and its relationship to intrinsic component images. Sections 1.1.2 and 1.1.3 then describe the specific types of intrinsic component images that this thesis will focus on.

1.1.1 Intrinsic Images

This chapter began by pointing out that a scene possesses many characteristics and that the observed image is the combination of these characteristics. In [9], Barrow and Tenenbaum propose representing each characteristic that is intrinsic to the appearance of the scene with a separate image. These images are known as *intrinsic images* because each image represents one intrinsic characteristic of the scene. The initial set of intrinsic images proposed in [9] includes an image for each physical characteristic of the scene that could be measured, such as the surface normal at each point and the illumination of each point in the scene.

This thesis focuses on a special type of intrinsic image, which will be referred to as an intrinsic component image. An intrinsic component image is an image that captures an intrinsic component of the scene and is combined with other intrinsic component images, using pointwise operations such as addition or multiplication, to produce the observed image.

The goal of this work is to decompose a single image of a scene into intrinsic component images that represent various characteristics of the scene. In particular, I will focus on two specific types of decompositions: shading/albedo and clean image/noise image.

1.1.2 Shading and Albedo Images

The meaning of shading and albedo images can be understood by considering a simple model of image formation. In this model, a shading image is formed from the interaction of the illumination of the scene and its geometry, or shape. An example of a shading image is shown in Figure 1-2(c). The observed image is produced by multiplying each point with its albedo. The albedo of a point is the fraction of light reflected from that point . Figure 1-2(b) shows the albedo image that, combined with the shading image in Figure 1-2(c), yields the final image in Figure 1-2(a). In this model, darker points in the albedo image should reflect less light than lighter points.

1.1.3 Scene and Noise Images

An image from a sensor can also be viewed as the combination of the true image of the scene and the noise from the sensor. In simple models of camera noise, this noise is additive. The observed image can thus be modelled as the sum of a scene image and a noise image.

1.2 Prior Work

1.2.1 Estimating Shading and Albedo

Much of the early work on decomposing an image into shading and reflectance images was motivated by the study of human lightness perception. When viewing a scene, the human visual system may attempt to remove the effects of illumination in order to accurately judge the reflectance of a surface [3]. Given the right stimulus, this can lead to illusions. A powerful example of this is Adelson's Checker-Shadow illusion, shown in Figure 1-3.

An influential algorithm for separating the effects of a surface's shading and albedo is the Retinex algorithm of Land and McCann [43]. The Retinex model is designed for lightness perception in a simplified world of "Mondrian" images. These images consist of a collage of patches, each with a different reflectance. The patches are lit with a slowly varying illumination. Figure 1-4(a) shows an example of a Mondrian image illuminated with smooth illumination.

In this simplified world, computing the illumination of each point in the scene requires several simple steps. These are easiest explained using a one dimensional

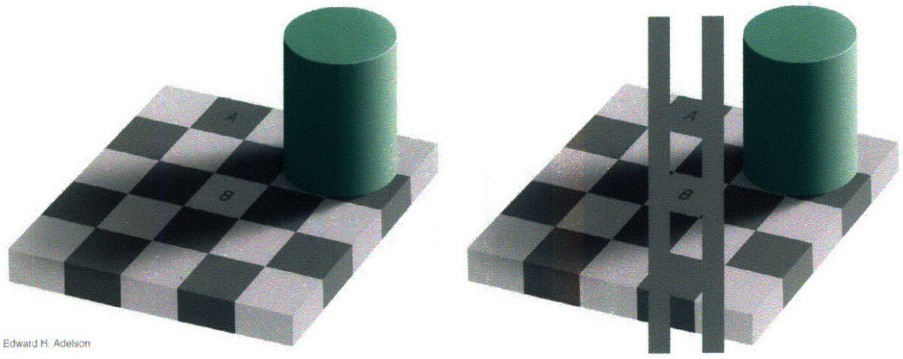
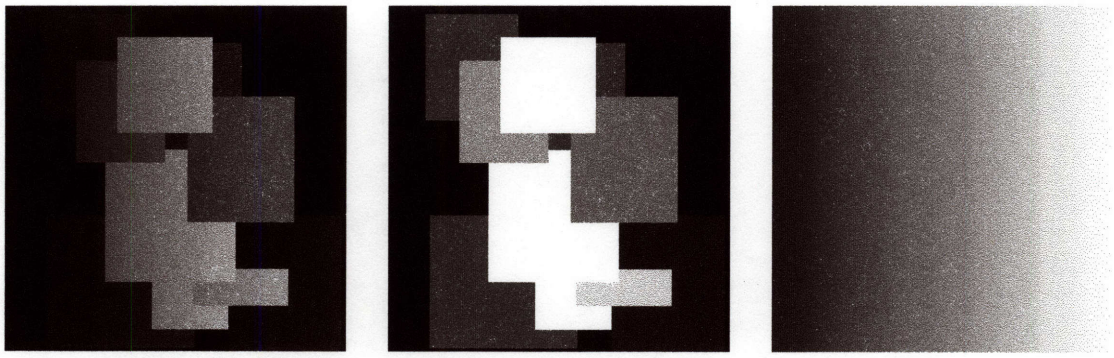


Figure 1-3: This illusion, produced by Edward Adelson, demonstrates how the human visual system attempts to correct for shading effects in a scene. The two squares, marked "A" and "B", are actually the same intensity, as revealed by the intensity reference bars on the right.

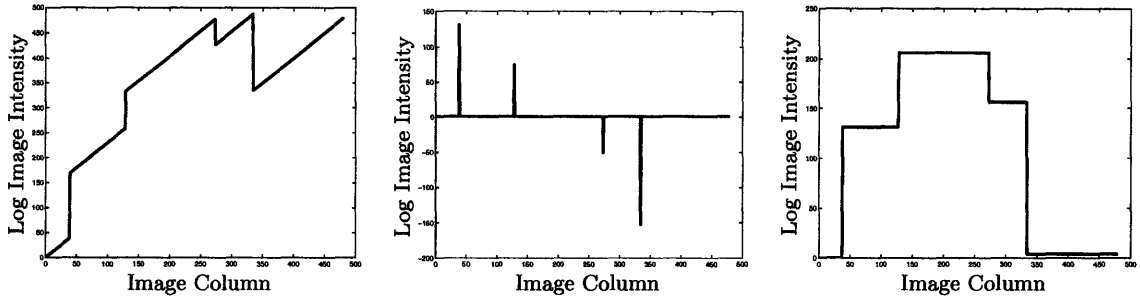


(a) An example of a Mondrian image.

(b) The reflectance pattern of the image.

(c) The illumination pattern of the image.

Figure 1-4: An image from the Mondrian world, along with the reflectance and illumination patterns used to create it.



(a) One column from the observed image.

(b) The derivative of the plot from (a).

(c) The estimate of the log shading

Figure 1-5: A one dimensional example of the Retinex process. Figure (a) shows the log image intensity along one row in the image. The first step in Retinex is to compute the derivatives, shown in (b). The albedo changes in the image are characterized by the spikes in (b). Setting these to zero and reintegrating leads to the estimate of the log albedo image in (c).

example. Figure 1-5(a) shows a plot of one row taken from the image in Figure 1-4(a). Because the observed image is the product of the albedo image and the illumination, the log is taken first to make the two components additive. The true reflectance of each patch, within a constant scale factor, is found by examining the derivatives of the log of the observation, shown in Figure 1-5(b). As can be seen in Figure 1-5(b), large magnitude derivatives coincide with changes in albedo, because the illumination varies slowly. On the other hand, derivatives with a small magnitude are probably caused by the illumination. The albedo can be recovered by setting derivatives with small magnitudes to zero, then re-integrating. This leads to a good estimate of the albedo, shown in 1-5(c).

Retinex was originally proposed to work along lines in the image, but did not specify how an albedo image could be recovered properly. Later, Horn [36] proposed a method that, while relying on this same heuristic, is able to reconstruct an image representing the albedo at each point. Since then, a number of other researchers have proposed systems based on similar assumptions [38, 40]. It is important to note that while much of the previous work uses the term "illumination" instead of "shading", the two can be viewed as equivalent.

A second heuristic, which is related to Retinex, is that the shading and reflectance images can be found by filtering the log of the input image [51]. This approach assumes that the shading component is concentrated in the low spatial frequency bands of the log input, while the reflectance image can be found from the high spatial

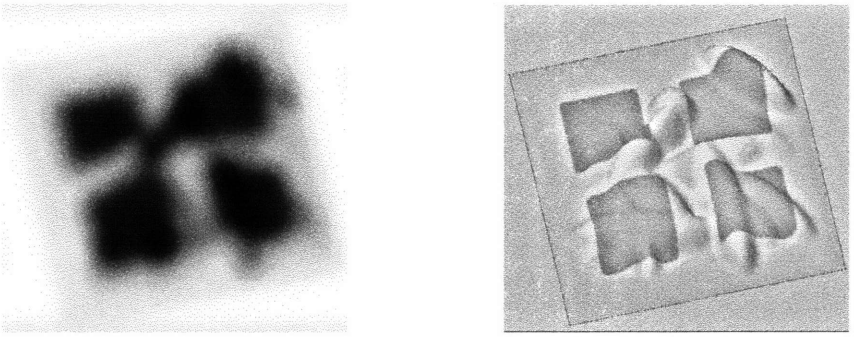


Figure 1-6: These high- and low-pass filtered versions of the log of Figure 1-2(a) show the infeasibility of solving this problem with simple filtering. Presumably, the image on the right would be the shading, while the image on the left would be the albedo of the scene. However, neither the shading nor the reflectance changes in this image are band-limited, so band-pass filtering cannot isolate one or the other.

frequencies. Like the assumption underlying Retinex, this assumption also tends not to be true in real images. Figure 1-6 shows high- and low-pass filtered versions of the log of the image shown in Figure 1-2(a). Shading and reflectance changes appear in both images because neither are band-limited. This makes it impossible for band-pass filtering to isolate either shading or reflectance changes.

1.2.2 Discriminative Approaches

Retinex and related algorithms rely on the assumption that the changes in the reflectance of a surface will lead to large derivatives, while illumination, which varies slowly, causes small derivatives. However, this assumption may not hold in real images. The image of the paper surface in Figure 1-7 is a good example of how Retinex's basic assumption can fail. This image has many sharp edges that are caused by shading. Under the Retinex assumption, these edges should be caused by changes in the surface's albedo.

To overcome this weakness in the basic model of Retinex, models have been proposed to better discriminate shading from albedo changes. Freeman and Viola [27] used a smoothness prior on the inferred shape in an image to classify the image as either entirely created by shading or all due to reflectance changes. The central drawback of this approach is that the whole image must be classified, so it is impossible to perform lighness computations at a point.

Instead of relying on just the magnitudes of the derivatives, Bell and Freeman [10] trained a classifier to use the magnitude of the output of a set of linear features. The

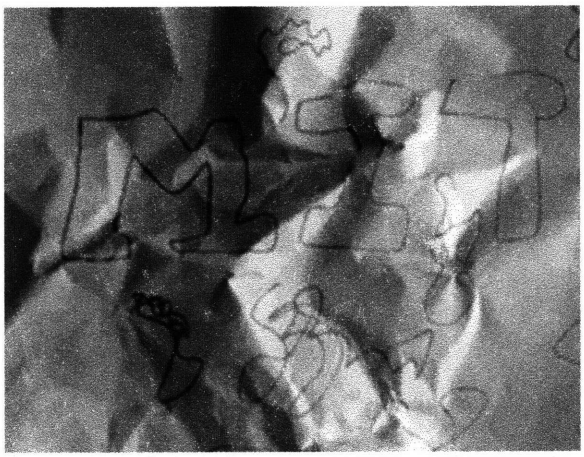


Figure 1-7: The surface pictured in this image violates the assumptions of the Retinex model. There are sharp creases and folds on the surface that lead to sharp image edges that are caused by shading. Under the Retinex model, these sharp changes should have been caused by albedo changes.

classifier was trained to label the coefficients of a steerable pyramid [60] as caused by either shading or a reflectance change. Using steerable pyramid coefficients allowed the algorithm to classify edges at multiple orientations and scales. However, the steerable pyramid decomposition has a low-frequency residual component that cannot be classified. Without classifying the low-frequency residual, only band-pass filtered copies of the shading and reflectance images can be recovered. In addition, low-frequency coefficients, which span a large region of the image, may not be naturally classified as either shading or reflectance.

1.2.3 Generative Approaches

Generative approaches are an alternative to these discriminative approaches. While discriminative approaches attempt to distinguish the effects of shading and reflectance, generative approaches create possible surfaces and reflectance patterns that explain the image, then use a model to choose the most likely surface.

Generative Models in a World of Painted Polyhedra

A good example of a generative approach is Sinha and Adelson's work on recovering the shape and reflectance of painted polyhedra [61]. This work focuses on images of polyhedra which have their faces painted different shades of gray. For these surfaces, every shading or reflectance change is manifested as a line in the image. Junctions of

lines will occur at each vertex of the polyhedron and at some areas with reflectance changes. The algorithm begins by labelling each junction as either being caused by a reflectance change or shading. Shading creates specific types of junctions, so Sinha and Adelson were able to create a catalogue of junctions that could be caused by shading.

Alone, this catalogue is not sufficient because albedo changes on the surface can have the same local appearance as shading. To overcome the limitations of local analysis, the next step is to find the most likely shape and lighting of the scene using global analysis. The authors find a set of perceptually likely shapes consistent with the input pattern, then verify that the rendered shapes could reproduce the input pattern. The final output is the most likely shape, the associated reflectance pattern, and the possible directions of the light source.

Workshop Metaphor

In [5], Adelson and Pentland use a "workshop metaphor" to describe their generative approach. In this metaphor, the world consists of images of planar surfaces. The scene is constructed from sheet metal that can be painted and shaped. In addition, lights can be placed at arbitrary locations. The scenes are created by three specialists: a lighting specialist, which places lights in the scene, sheet metal worker, which changes the shape of the surface in the scene, and a painter, who changes the reflectance of the surface. Given an image, each specialist tries to recreate the scene using as little assistance as possible from the other specialists. This scenario also includes a supervisor, who is able to use all three specialists.

The different possible scenes are evaluated according to the cost of creating each scene. Each action performed by a specialist to recreate the scene has a cost assigned to it, with more complicated actions having a higher cost. For instance, painting a rectangle is \$5, while painting a general polygon is \$5 per side. The most likely scene is computed by finding the scene with the lowest cost.

Mosaicing Shading Images

Instead of explicitly modeling the surfaces in the scene, Freeman et al. [26] proposed constructing the shading image from a mosaic of image patches taken from example images. The shading image is reconstructed by dividing the input image into patches, then for each image patch, choosing the shading and reflectance patches that best

match the image patch. These patches are mosaiced together to form the shading and reflectance images. The algorithm considers a discrete set of candidate patches for each image patch. The probability of each candidate patch being the best patch for a portion of the mosaic is influenced by two factors: how well the patch explains the image and how compatible the patch is with neighboring patches. In addition, there is a penalty based on the flatness of the shading patch applied to the log probability of each candidate. This is important because any image can be explained as a reflectance pattern painted onto a flat surface. The chief disadvantage with this approach is that the surfaces must be constructed explicitly, limiting the system's applicability to real-world images.

Gaussian Generative Approaches

In recent work, Stainvas and Lowe have proposed modelling the illumination and reflectance with a generative model that relies on a Gaussian Markov Random Field [65]. The parameters of this field are fit using a Generalized EM algorithm. Chapter 4 also relies on a Gaussian MRF, but of a much different form. In addition, Stainvas and Lowe must resort to approximate inference techniques to learn the parameters of the MRF. Chapter 4 will show how to learn the parameters without resorting to approximate inference.

1.2.4 Other approaches

The previous sections have focused on algorithms that use a single, gray-scale image of the scene to estimate the shading and albedo of the scene. Some of the difficulties inherent in this task can be avoided by considering richer inputs with more data.

Utilizing Color

Color is a strong cue for finding changes in the albedo of a scene. Instead of thresholding based on the intensity of the gradient, Funt et al. [32] proposed thresholding based on color information in the image. In more recent work, Finlayson et al. have proposed using color cues to remove shadows from images [23]. Recently, Finlayson et al. have proposed a method that does not require a calibrated camera to remove shadows [22].

Multiple Images

In a different direction, Weiss [78] proposed using multiple images where the reflectance is constant, but the illumination changes. This approach was able to create full frequency images, but required multiple input images of a fixed scene. Images with varying illumination are also used in [46] to eliminate shadows from surveillance images. More recently, Matsushita et al. have worked on the case where the range of illuminations is biased [47].

1.2.5 Broader Links

Besides the links to previous work on the specific problem of estimating shading and albedo images, the work described in this thesis has links to broader problems in computer vision and machine learning.

Shape-From-Shading

The problem of recovering the shape of a surface from a single image of a scene, also known as shape-from-shading, touches on many of the same issues as estimating the shading and albedo of a scene. Zhang et al. present a good overview of shape-from-shading methods in [80]. In the simplest formulation of the problem, it is necessary to provide the orientation of the illumination and ensure that the surface has constant albedo before the shape of the surface can be recovered.

More advanced algorithms, such as those from Zheng and Chellapa [81] and Samaras and Metaxas [57], avoid the need to explicitly provide the light source direction by estimating the light source illumination. Zheng and Chellapa's algorithm also seeks to estimate three parameters that describe the reflectance of the surface.

These algorithms differ from the approach described in this thesis in two ways. First, these algorithms can be viewed as reconstructing the illumination and surface information by explicitly recovering the surface's shape and illumination. This requires that algorithm make specific assumptions about the scene, such as the assumption that the scene is illuminated by a point-source. This thesis takes a different approach by simply estimating an image that captures the shading of the scene. This enables the intrinsic image approach described in this thesis to sidestep the inverse problem of recovering the shape and illumination of the scene.

The second difference is that these algorithms do not model albedo variations in

the scene. This is significant because most real-world scenes will have some variation in the albedo of the scene.

Markov Random Field Models of Images

As Chapter 4 will discuss, the estimates of the shading and albedo of a scene can be significantly improved by modelling the images with a Markov Random Field (MRF), then learning the parameters of this MRF.

Markov Random Field models have been used extensively to model images. Geman and Geman introduced MRF models for image denoising and image restoration [33]. Freeman et al. proposed estimating images by constructing a mosaic of image patches, chosen using a Markov Random Field Model. [26]. Rosales et al. and Tappen et al. proposed similar models where the image was created from patches that were computed directly from the observed image [54, 70]. These models were used for a number of tasks including rendering photographs as paintings, demosaicing images, and image super-resolution.

The particular type of MRF used in this thesis is a Gaussian Markov Random Field [63]. In [66], Szeliski used Gaussian Markov Random Fields as a prior on depth images.

Learning MRF Models

While much work in computer vision utilizes MRF models, the parameters of the model are most often hand-specified. This is because the types of MRF models used in vision often contain loops, which makes inference intractable [26]. A common strategy for addressing this issue is to use sampling, as in the work of Zhu and Mumford [82].

If the MRF models do not have loops, which is more common in research areas such as Natural Language Processing, learning the MRF parameters is easier. In [21], Della Pietra et al. describe an algorithm that can choose the features of the MRF. This algorithm was applied to models of text. McCallum later extended this algorithm to much larger sets of candidate features [48].

Historically, algorithms for learning MRF parameters have been posed as optimizing a probabilistic criterion. Recent work, such as that of Collins [18] and Taskar et al. [71], has proposed learning MRF parameters using different criteria. Section 4.7 discusses the relationship between this work and the models described in this thesis.

(a) Original Image (b) Shading Image (c) Reflectance Image

Figure 1-8: An example of the results produced by the system described in [69]. This is found by combining the local evidence from the color and gray-scale classifiers, then using Generalized Belief Propagation to propagate local evidence. The face that has been painted on the pillow is correctly placed in the reflectance image, while the ripples on the surface of the pillow are placed in the shading image.

1.2.6 Author's Previous Work

In [67, 68, 69], I have presented a discriminative system that takes advantage of both color and gray-scale information. Like Retinex and the work of Freeman and Bell [10], the premise of the system is that every image derivative can be classified as being caused by shading or an albedo change. The derivatives are classified using two forms of local evidence to classify the image derivatives: color and intensity patterns.

Color information is utilized by finding derivatives where the chromaticity of the neighboring pixels changes. In this system, a change in chromaticity indicates a reflectance change. Derivatives where there is a chromaticity change are classified as reflectance changes.

Intensity information is used by assuming that surfaces and illuminations in the world have regular properties that give shading patterns a unique appearance which can be discriminated from most common reflectance patterns. This regular appearance of shading patterns allows the local gray-scale image pattern surrounding a derivative to be used to classify it as either shading or an albedo change.

While combining color and gray-scale information works well, some image patterns are locally ambiguous. For instance, a sharp edge in an image such as Figure 1-7 could be caused by a crease in the paper or a strong albedo change. To properly classify derivatives in ambiguous areas, information must be propagated from areas of the image where the classification is clear into areas where the correct classification is

ambiguous.

The propagation step is implemented using a Markov Random Field (MRF). A node in the MRF is assigned to each image derivative. Each node represents the correct classification of the derivative. The compatibility functions in the MRF are configured to propagate information according to the heuristic that derivatives along an image contour should have the same labels. Using the MRF, the correct classification of each derivative is estimated using the Generalized Belief Propagation algorithm [79].

This system works well on real-world images. Figure 1-8 shows an example of the output of our system. The face and cheek patches that have been painted on the pillow are correctly placed in the reflectance image, while the ripples in the pillow are placed in the shading image.

1.2.7 Limitations of Previous Work

The system discussed in the previous section overcame issues in previous work by:

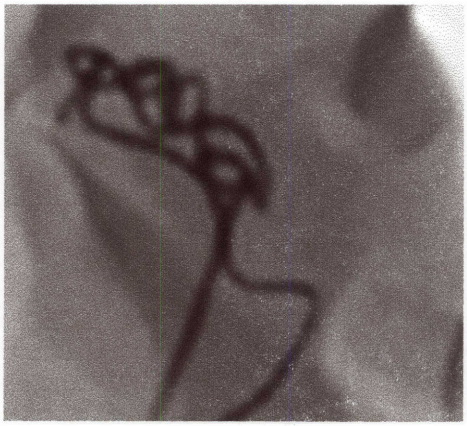
1. Using machine learning to build better models of shading and albedo images than the simple smooth/sharp model used by algorithms like Retinex.

2. Estimating full-frequency shading and albedo images by classifying derivatives.

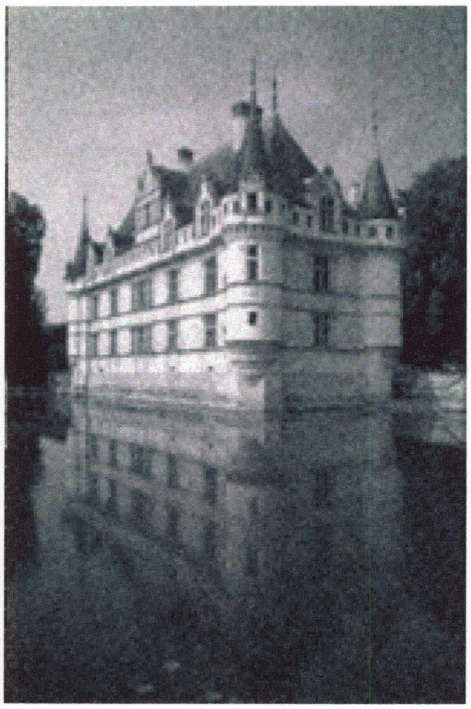
3. Using a Markov Random Field to propagate information from areas of the image with good local evidence into areas of the image where the correct classification of the derivatives is ambiguous.

However, this system has several significant limitations. The most signficant limitation is its reliance on classifying derivatives. As can be seen in Figure 1-9, shading and albedo changes often coincide. In these situations, classification is not appropriate. Another problem with classification can be seen in Figure 1-9. If the images are properly anti-aliased, strong albedo edges in the image will tend to be surrounded by derivatives with a small magnitude. Classifying these low-magnitude derivatives is difficult because classifying these derivatives as albedo changes will remove any shading around albedo changes. On the other hand, classifying these derivatives as shading will cause small albedo changes to be included in the shading image.

Relying on classification also limits the type of problem that can be addressed. Examining the noisy image in Figure 1-9(b) makes it clear that the derivatives in



(a)



(b)

Figure 1-9: Two examples of cases where the classification strategy breaks down. (a) Anti-aliasing around edges in the images leads to blurred edges that do not have a clear classification. (b) Classification will not work for denoising because noise and scene derivatives occur everywhere.

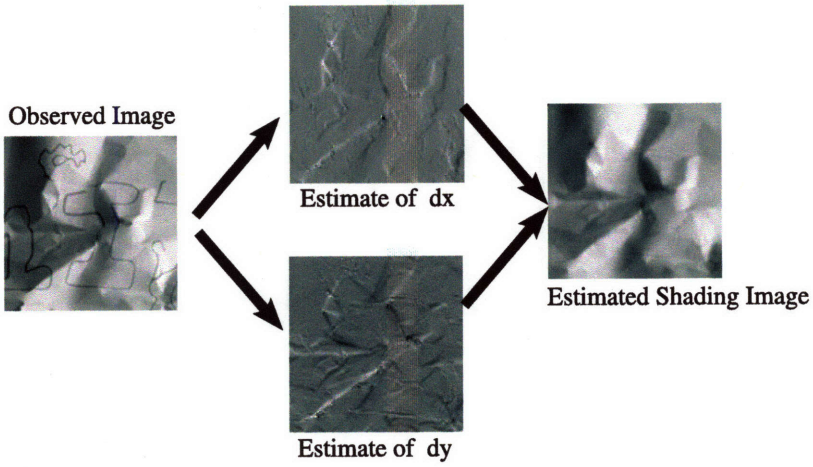


Figure 1-10: This diagram describes our basic approach to recovering intrinsic component images. In this example, we recover the shading image of a piece of crumpled paper by first estimating the horizontal and vertical derivatives of the shading image. The estimate of the shading image is produced by finding the image that best satisfies these constraints in the least squares sense.

this image cannot be classified as either belonging to the scene or the noise image. A system based on classification will be thwarted by the fact that there there are changes in the noise image at every pixel.

The second limitation of this system is that it was hand-designed. While some of the parameters in the MRF were optimized, the overall heuristic that information should propagate along contours was hand-selected, and the training was designed around this heuristic.

The third limitation of this system is that it was trained using synthetic data. Training with synthetic data is undesirable because the assumptions about the world are built into the data. In addition, it is unlikely that surfaces in synthetic data will have the complex shape and appearances that real-world surfaces possess.

1.3 Basic Strategy for Estimating Intrinsic Component Images

This thesis describes a system for estimating intrinsic component images that overcomes each of these limitations of the previous system. To overcome the limitations of classification, estimating the intrinsic component images will be viewed as a continuous estimation problem.

Viewed as a generic non-linear regression, estimating an intrinsic component image with N pixels is an N-dimensional regression problem. Even for small images, this dimensionality is too large and the space of all possible N pixel images is too complex to use black-box regression algorithms. This dimensionality issue can be overcome by making a simplifying assumption. The key simplifying assumption is that at every pixel in the image being estimated, there are quantities that can be estimated using only a small local patch of the observed image. For example, in the world of Mondrian images from the Retinex algorithm, a two-pixel image patch is sufficent to estimate the derivatives of the shading image.

Most of the previous work described in Section 1.2 has focused on using horizontal and vertical derivatives as these local quantities. This is necessary because *pixel* values cannot be estimated directly. Consider an image of a surface with two large regions of differing albedo. If the region with darker albedo is larger than the image patch used to estimate pixel values, it is impossible to correctly recover the shading of that area. The classic Retinex algorithm overcomes this by making local estimates of the *image derivatives*, rather than the pixel values. These derivative estimates are then used to estimate the image by solving Poisson's equation.

This thesis generalizes this approach by allowing the local quantities being estimated to be arbitrary linear filters. These linear filters are used to estimate the intrinsic component image in the following steps:

1. Choose a set of N_f linear filters, $f_1 \ldots f_{N_f}$. This set could be something as simple as horizontal and vertical derivative filters, or a much more complex set of filters.

2. If the observed image is denoted $\mathcal{O}$ and the true intrinsic component image being estimated is denoted $\mathcal{T}$, the next step is to use $\mathcal{O}$ to estimate $f_i * \mathcal{T}$, where $*$ denotes convolution, for all $i = 1 \ldots N_f$. In other words, the observed image $\mathcal{O}$ is used to estimate how the intrinsic component image would look if it were filtered with f_i.

   As mentioned earlier, to control dimensionality issues, it is expected that these estimates can be produced at every pixel using only local image information. Chapter 3 will describe how to use training data to learn non-linear estimators of the filtered intrinsic component image.

3. Each of these estimated values of $f_i * \mathcal{T}$ can be viewed as a constraint on

the intrinsic component image that is being estimated. The final step is to reconstruct the estimated intrinsic component image by finding the image that best satisifies these constraints. Throughout most of this thesis, the estimated intrinsic image will be found using a pseudo-inverse operation.

Figure 1-10 shows an example of how this process works when estimating a shading image. In this figure, the constraints are horizontal and vertical first derivatives. The first step is to estimate the first derivative of the shading image at every pixel. Given these estimates, which form an overcomplete linear system, the estimated shading image is produced by finding the pseudo-inverse of this linear system.

The choice of filters depends on the image statistics. Filters should capture and distinguish the characteristic statistics of the intrinsic components. In the case of classic Retinex, the assumption is that shading images are smooth, while reflectance images are piecewise constant, so that the processes can be easily separated in the derivative domain. Since real images are more complex, more sophisticated statistics must be learned in order to do the separation.

It should be noted that shading and albedo images combine multiplicatively, so the system actually estimates the log of the shading and albedo images. This makes the estimation problem additive. I have found that it is not necessary to take the log when applying the algorithm to images from uncalibrated cameras. This is because photographic tonescale is similar to a log transformation.

1.4 Contributions of this Thesis

The main focus of this thesis will be the task of learning to separate an image into albedo and shading components. The system described in this thesis achieves state-of-the-art performance on imagery that is more difficult than that used in previous work. Unlike my previous work, color information will not be used. For this work, I have chosen to focus on the problem of estimating shading and albedo images from gray-scale images. This harder problem is a better test of the capabilities of the algorithms described here.

However, this approach is not limited to estimating shading and albedo images. The same approach can be used for other image estimation problems where the statistics of different image classes are distinct. To demonstrate this, the techniques described in this thesis will also be applied to the problem of removing noise from

images. Instead of training on images with and without albedo changes, the system can be trained on examples with and without noise. As Section 3.12 will show, the results are competitive with state-of-the-art denoising methods.

In addition to the state-of-the-art system for estimating intrinsic images described in this thesis, the contributions of this work include:

1. An efficient, boosting-like method for training the estimators that predict the estimated filter outputs. This method scales to large data sets and, as will be shown in Chapter 2, produces estimators that are both smaller and make lower-error predictions than estimators trained using Support Vector Regression.

2. A data set of real-world surfaces consisting of observations and ground-truth shading images. To my knowledge, this is the first data set for the shading and albedo estimation problem with ground-truth decompositions.

3. A method for learning to how to weight the various constraints in the linear system. As Chapter 4 will show, this is important because, as discussed in Section 4.1.2, the correct estimate of a filter value may be ambiguous given only local information. Chapter 4 will show how to learn a weighting function that compensates for this ambiguity and leads to significantly better estimates.

4. As Chapter 4 will show, learning this weighting function is equivalent to learning the parameters of a Markov Random Field model of the estimated image. In contrast with the previous work of Tappen et al., the parameters of this MRF are completely learned from the training data. This enables the behavior of the system to be driven by the data, rather than hand-engineered heuristics.

1.5 Roadmap

As Section 1.3 describes, the first step in estimating images is to estimate the value of $f_i * \mathcal{T}$, the filtered intrinsic component image. Chapter 2 describes a stagewise, boosting approach for learning the estimators to predict these images.

Chapter 3 then evaluates the performance of these estimators on both the tasks of estimating shading and albedo images and denoising images. Chapter 3 also discusses the details of the pseudo-inverse step and constructing the training and test data.

Because these estimates are found using small patches of image data, the correct value of a linear filter response may be ambiguous. Chapter 4 discusses how to learn

which image patches are ambiguous and appropriately weight the estimates. This corresponds to learning the parameters of a Conditionally-Gaussian Markov Random Field. Chapter 4 describes an efficient method of learning these parameters and demonstrates how this leads to signficantly better estimates of the shading images.

1.6 Notation

Throughout this thesis, the variables $\mathcal{O}$, $\mathcal{T}$, and $\mathcal{X}$ will represent the observed image, the ground-truth "target" image, and the estimated intrinsic component image, respectively. Generally, o will represent an observation vector, such as a patch taken from an image, and t will represent a ground-truth target value. The variable $\hat{c}$ will be used to denote the estimated values of the linear constraints that are used to reconstruct the estimated image.

Chapter 2

Learning a Mixture of Experts Estimator

As the previous chapter explained, an intrinsic component image is estimated by

1. Choosing a set of linear filters, $f_1 \ldots f_{N_f}$, that will serve as constraints on the pixels in the estimated image.

2. For each f_i, learning to predict $f_i * \mathcal{T}$, from the observed image $\mathcal{O}$, where $\mathcal{T}$ is the ground-truth target image.

3. Performing a pseudo-inverse operation to reconstruct the image from the estimated filtered image values.

The intuition behind this strategy is that the correct response of some filters can be estimated from image patches. The estimated image is then reconstructed from these response estimates. Obtaining good estimates of the filtered image is the critical step in estimating high-quality intrinsic component images. This chapter focuses on the issues involved in choosing an estimator to predict these values. When choosing an estimator to use, issues beyond the error in the estimates, such as the amount of computation needed to produce the estimates, must be considered. Sections 2.2 and 2.3 explore these issues and cover background material on regression.

Section 2.4 introduces the Mixture of Experts estimator, which is the type of estimator that will be used to predict the filtered image values. Section 2.5.2 presents an efficient, stagewise algorithm for learning this Mixture of Experts estimator.

2.1 Basic Problem Statement

The goal is to learn a function $g : \mathbb{R}^M \rightarrow \mathbb{R}$ that maps from an M-dimensional observation vector o to a scalar estimate of the linear constraint value. Practically, in the basic system described in Section 1.3, g will map from an image patch to an estimated filter response at the center of the patch.

The actual estimates produced by the estimator g will depend on the parameters of that estimator, θ. In this chapter, $g(\cdot;\theta)$, will denote an estimator defined by parameters θ. While the parameters of g could be chosen by hand, it is unlikely that this estimator will capture the rich behavior of real-world images. Instead, the parameters can be found using training examples.

Formally this can be accomplished by first choosing a set of N_T training pairs, (o_i, t_i), where $i = 1 \ldots N_T$. In each pair, o_i denotes an observation, which could be a vector, while t_i denotes the scalar target value to which $g(o_i;\theta)$ should evaluate when presented with observation o_i. Due to ambiguities and noise in the training data, it is unlikely that $g(o_i;\theta)$ will equal t_i for all o_i, so a loss function, $L(g(o_i;\theta), y)$ must also be chosen. The loss function expresses the penalty for $g(o_i;\theta)$ differing from t_i. One of the most popular loss functions is the squared error:

$$L(g(o;\theta), t) = (g(o;\theta) - t)^2 \tag{2.1}$$

Given a loss function and training examples, the parameters, θ, can then be found by minimizing the sum of the loss function over the training set:

$$\theta = \arg\min_{\theta} \sum_{i=1}^{N_T} L(g(o_i;\theta), t_i) \tag{2.2}$$

This is also known as "training" or "learning" the estimator.

2.2 Issues in Choosing the Type of Estimator

The estimator $g(\cdot;\theta)$ can take a number of forms. In this work, the choice of the estimator for estimating intrinsic component images is dominated by three major issues:

1. Flexibility

Without the ability to model complex relationships between the observations and constraint values, the system will be unable to cope with the complex statistics of real-world images.

2. Computation Required for Training

   One of the primary goals in this work is to avoid simplistic, hand-constructed models of images. To capture the variation in real-world images, it is vital that the training procedure scale to large numbers of training examples.

3. Computation Required to Produce Estimates

   When producing the estimates of the intrinsic component images, the value of multiple constraint values must be estimated at each pixel in the image. For even a small 128 $\times$ 128 pixel image, this adds up to tens of thousands of estimates that must be produced. The overall system will not be useful unless these estimates can be produced quickly.

2.3 Possible Types of Estimators

With these different criteria in mind, it is possible to assess the usefulness of different types of estimators. This section considers several types of estimators that could be used to estimate the filter response values.

2.3.1 Linear Estimators

The simplest type of estimator is a linear estimator of the form

$$g(o;\theta) = \theta^T o \tag{2.3}$$

where θ is a vector of real numbers. This estimator requires relatively little computation to train and produce estimates, but can only express a simple relationship between the observation and prediction. It can be expected that the relationship between image patches and constraint values will be complex, so it is unlikely that this estimator will be accurate enough.

2.3.2 Nearest Neighbor Estimator

In the other direction, non-parametric estimators have the ability to model complex relationships between observations and predictions, with the potential cost of requiring excessive computation to produce estimates. This can be understood by considering a simple k-nearest neighbor estimator:

$$g(o;\theta) = \frac{1}{N_{\mathbb{N}}} \sum_{i \in \mathbb{N}_k(o)} t_i \tag{2.4}$$

where $\mathbb{N}_k(o)$ denotes the k training samples that are the most similar to o, and $N_{\mathbb{N}}$ denotes the number of neighbors. The patches in $\mathbb{N}_k(o)$ are known as its k-nearest neighbors. Given o, this estimator produces an estimate of y by averaging the k training samples that are the most similar to o. This estimator can be further refined by using the k-nearest neighbors to fit a polynomial, then using that polynomial to predict t. This technique is known as Locally-Weighted Regression [7, 17]. The advantage of locally-weighted regression is that more complicated relationships in the neighborhood can be captured.

This estimator can obviously predict complex relationships between o and t, as long as this relationship is locally smooth. The disadvantage of this estimator lies in the task of finding the k-nearest neighbors. Using brute-force techniques, the cost of finding the k-nearest neighbors of o grows linearly with the number of training examples.

2.3.3 The Nadaraya-Watson Estimator and Kernel Density Regression

A smoother alternative to the nearest-neighbor estimator is an estimator of the form:

$$g(o;\theta) = \frac{\sum_{i=1}^{N_T} t_i e^{\frac{-||o-o_i||^2}{h^2}}}{\sum_{i=1}^{N} e^{\frac{-||o-o_i||^2}{h^2}}} \tag{2.5}$$

where o_i and y_i are training examples from the set of N_T training examples. The parameter h is a scale factor that controls how the influence of a point falls off. This estimator is know as the *Nadaraya-Watson estimator* [50, 77, 13].

Like the k-nearest neighbor estimator, an estimate is produced by averaging the training samples. Assuming that the Euclidean distance between patches is an ap-

propriate measure of similarity between vectors, the $e^{\frac{-||o-o_i||^2}{h^2}}$ term in the numerator of Equation 2.5 will be close to 0 for dissimilar patches and close to 1 for similar patches. These similarity measures are used to compute a weighted average of the training examples.

This estimator can be viewed as a smooth version of the nearest neighbor estimator. Rather than using a set of neighbors in the training set to make a prediction, every point in the training set contributes to the prediction. However, the contribution of each example is weighted by its similarity to o. The nearest neighbors to o will contribute the most, while dissimilar training examples will contribute relatively little.

2.3.4 Probabilistic Interpretation of the Nadaraya-Watson Estimator

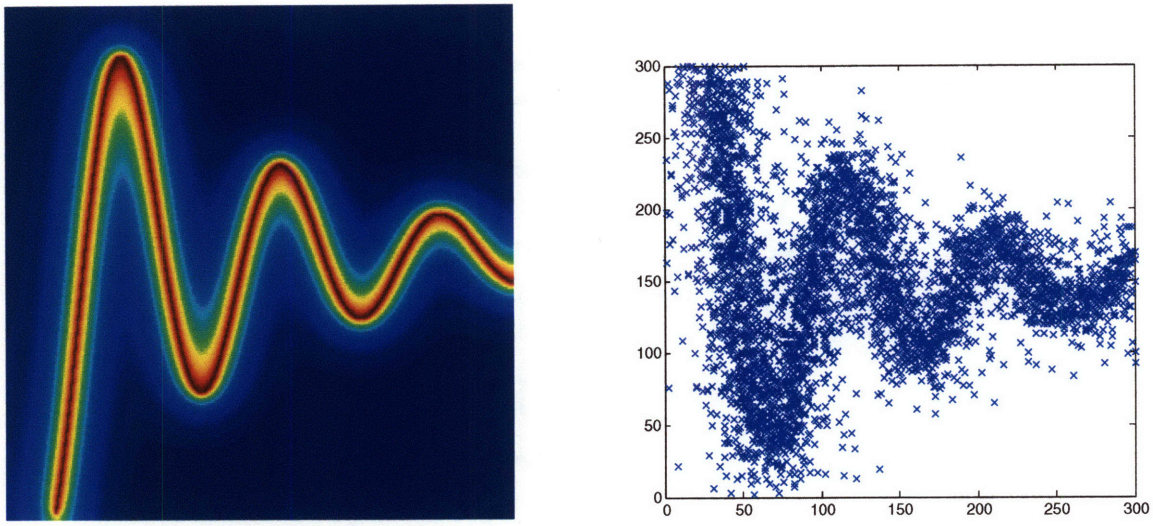
In addition to its connection to the k-nearest neighbor estimator, the Nadayara-Watson estimator is motivated well in a probabilistic framework [13]. To see the connections, I will first describe basic kernel density estimation, then show how using a density estimated in this fashion to do regression leads to the Nadaraya-Watson estimator.

Kernel Density Estimation

To begin, consider the problem of estimating a probability density function, $\hat{p}(x)$, from a set of samples, $x_1 \ldots x_N$, drawn from some unknown density, $p(x)$. For example, the samples shown in Figure 2-1(b) were drawn from the relatively complicated 2D pdf, $p(x)$, shown in Figure 2-1(a). The goal is to use the samples in Figure 2-1(b) to produce an estimate, $\hat{p}(x)$, of the density function in Figure 2-1(a).

Rather than imposing a functional form on $\hat{p}(x)$, which often overly limits the complexity of $\hat{p}(x)$, $\hat{p}(x)$ can be estimated using non-parametric methods. The unique characteristic of non-parametric methods is that $\hat{p}(x)$ is specified using the observations, rather than a small number of parameters.

Kernel density estimation, also known as Parzen density estimation, is a popular approach for estimating $\hat{p}(x)$ from $x_1 \ldots x_N$. Using this approach, $\hat{p}(x)$ is constructed by placing a small kernel at each sample x_i. Following Bishop's [13] notation, the density estimate can be constructed using a kernel function, $H(u)$:



(a) (b)

Figure 2-1: An example probability density function, shown in (a), and samples drawn from that distribution.

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} H(x - x_i) \tag{2.6}$$

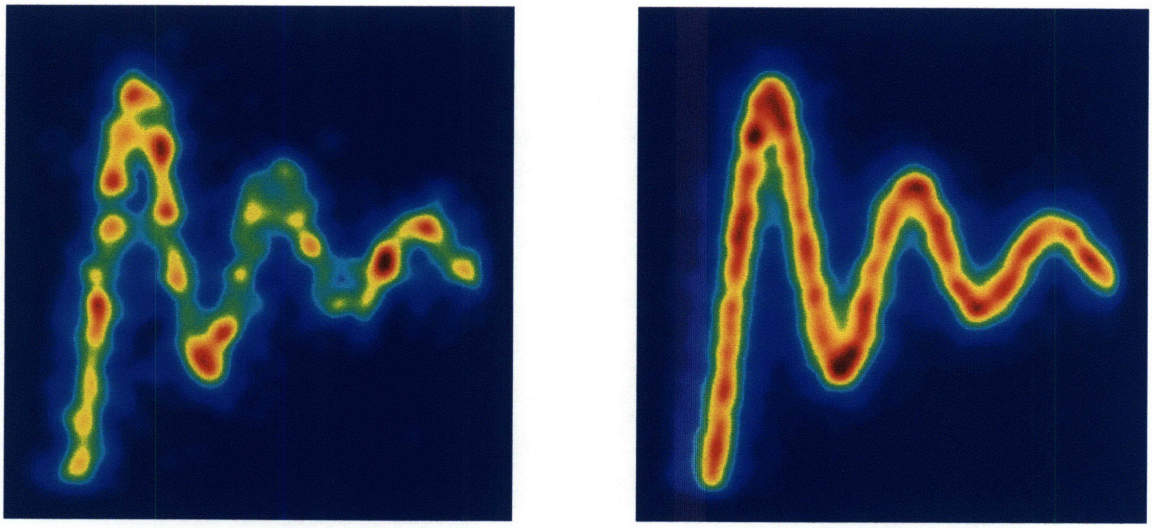
where $u \in \mathbb{R}^M$. If $H(u) > 0$ for all u and $\int H(u)du = 1$ then this estimate will be a valid probability density function.

If the kernel function is chosen to be a Gaussian kernel, with standard deviation h, the estimated kernel density function becomes:

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{Z} \exp\left(\frac{-||x - x_i||^2}{h^2}\right) \tag{2.7}$$

where $\frac{1}{Z}$ is the normalizing constant for the Gaussian kernel. The term h is sometimes referred to as the bandwidth of the kernel.

Figure 2-2(a) shows the density estimate produced using only 2000 samples of the density from Figure 2-1(a). The behavior of the kernel density estimator is clear in this figure. The estimate is the sum of a set of Gaussian bumps that have been placed at the location of the training samples. While this estimate is somewhat poor, given enough samples and if h is chosen properly, the kernel density estimation can produce good estimates of complicated densities. Figure 2-2(b) shows the density estimate produced using 20000 samples. This estimate matches the true density well.



(a) 2000 Samples

(b) 20000 Samples

Figure 2-2: Kernel density estimates of the density function shown in Figure 2-1(a). Figure (a) shows the estimate found with a relatively small number of samples. It is uneven and does not approximate the true density well. (b) With more samples, the estimate of the density improves significantly.

Deriving the Nadaraya-Watson Estimator as Estimation using the Kernel Density

The Nadayara-Watson estimator can now be derived as regression using a kernel density estimate. For convenience, we denote a sample x_i from the density $p(x)$ as a pair (o_i, t_i), where o denotes an observation that we wish to use to predict the target value t. Given an estimate of the probability density function of x_i, which can also be viewed as the joint density function of o and t, the estimate of t that minimizes the mean-squared error criterion, $\hat{t}$, is the conditional expectation of t, given o:

$$\hat{t} = E[t|o] \tag{2.8}$$

where the expectation is over the estimated density function, $\hat{p}(x)$.

Using the properties of Gaussian density functions, this is equal to

$$\hat{t} = \frac{\sum_{i=1}^{N} t_i e^{\frac{||o-o_i||^2}{h^2}}}{\sum_{i=1}^{N} e^{\frac{||o-o_i||^2}{h^2}}} \tag{2.9}$$

which is the Nadaraya-Watson estimator from Section 2.3.3.

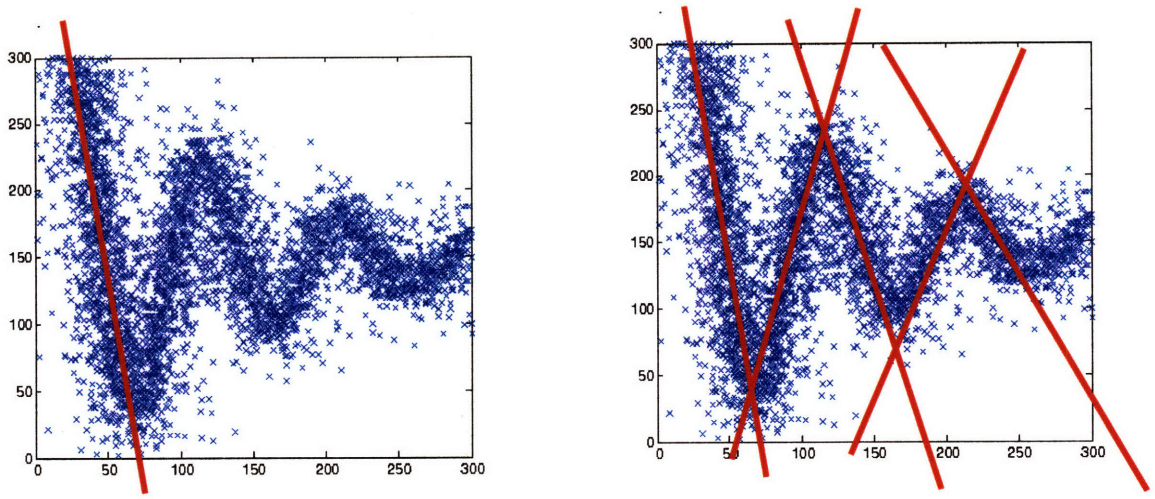
2.4 Overcoming Complexity issues using Experts

The flexibility of the Nadaraya-Watson and k-nearest neighbor estimators make them ideal estimators to use for predicting the constraint values. However, this flexibility comes at the price of high computational complexity. To predict the target value t from an observation vector, o, this vector must be compared with every sample in the training set. As the training set grows large, these comparisons become prohibitively slow. In addition, a prediction must be made for nearly every pixel in an image, making speed issues central to the usefulness of the system. Algorithms, such as the kd-tree, have been proposed to speed up these comparisons. However, the benefits of these algorithms may diminish when high-dimensional observations are used. For example, Sproull has reported that the benefits of algorithms based on trees diminish when the dimensionality of the samples is greater than 12 [64].

While recent research has proposed new algorithms that overcome some of the weaknesses of tree-approximations [20], these estimators do not provide any way of summarizing the data. One advantage of parametric estimators is that their parameters give a concise description of the relationship between the observations and the target values. The ideal estimator would offer the flexibility of the non-parametric estimators, with the ability to learn a concise description of the relationships in the training data.

This section shows how a Mixture of Experts [39, 37] estimator can offer the flexibility of a non-parametric estimator, while still summarizing the data with a relatively few number of parameters. The Mixture of Experts estimator can be motivated by noticing that the local relationship between o and t can be modelled well as a linear relationship. Figure 2-3(a) shows this property using the samples from the example distribution discussed previously. As Figure 2-3(b) shows, more complicated relationships can be expressed by considering multiple linear relationships. Each red line in Figure 2-3(b) captures the relationship between o and t for a range of o. Each of these lines can also be thought of as defining the optimal linear regression function for a range of o.

This notion of using N_E multiple linear regression functions to estimate t from o can be expressed formally by defining a function $g(o)$ that estimates t from an $M \times 1$ observation o:



(a) Single

(b) Multiple

Figure 2-3: The Mixture of Experts estimator is able to capture the relationship between the observation and the target. (a) Locally, the relationship between o and t is modelled well as a linear relationship. (b) Using multiple linear regressions, the relationship between o and t is modelled well.

$$g(o;\theta) = \sum_{i=1}^{N_E} p(i|o)(\theta_i^{\mathrm{T}} o) \tag{2.10}$$

where θ_i is a $M \times 1$ set of linear regression coefficients. The probability mass function $p(i|o)$ is the conditional probability that θ_i is the best set of linear regression coefficients for predicting t, conditioned on o. Essentially, the estimate $g(o)$ is a weighted average of linear estimates. The probability mass function, $p(i|o)$, acts as a gating function that assigns weight to the various linear estimates based on how "expert" they are for a particular observation o. This estimator is referred to as a Mixture of Experts model because it is built from a set of functions that are each experts for a different region in the space of observations.

2.4.1 Completing the Estimator by Defining $p(i|o)$

Fitting a Mixture of Experts Estimator involves choosing the number of experts, N, then fitting the regression coefficients and $p(i|o)$. Jacobs et al. proposed modeling the regression coefficients and $p(i|o)$ as the output of a neural network [37]. The parameters of this network are found by maximizing the likelihood of the data using gradient descent. In later work, Jordan and Jacobs showed how to use the EM

algorithm to fit this model [39].

While fitting the linear regression coefficients and $p(i|o)$ is relatively easy, the number of experts must be chosen beforehand. This problem is similar to the problem of choosing the number of mixture components when estimating a probability distribution using a mixture model. This problem, sometimes referred to as model selection, is discussed more in Section 2.5.

Fortunately, the tasks of fitting $p(i|o)$, learning the regression coefficients, and choosing the number of mixture coefficients can be accomplished simultaneously. This section describes an algorithm, which I will refer to as the ExpertBoost algorithm, that uses training data to fit a Mixture of Experts estimator.

Modelling $p(i|o)$

Before describing the algorithm for learning the estimator $g(o)$, the exact form of the Mixture of Experts estimator must be fixed by choosing the form of $p(i|o)$. We follow the work of Jacobs et al. [37] and use a softmax function to express $p(i|o)$:

$$p(i|o) = \frac{\exp(\gamma_i)}{\sum_{j=1}^{N} \exp(\gamma_j)} \tag{2.11}$$

where γ_i is the output of some function.

Inspired by the Nadaraya-Watson estimator and kernel density estimation, we define the $p(i|o)$ using the softmax function over the Euclidean distance between o and a set of "prototype vectors":

$$p(i|o) = \frac{\exp\left(-\frac{\|o-p_i\|^2}{h}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{\|o-p_j\|^2}{h}\right)} \tag{2.12}$$

where $p_1 \ldots p_N$ are a set of prototype vectors and h is a scale factor. For brevity, it will be assumed that $h > 0$ throughout this thesis.

The final form of the estimator can now be expressed as

$$g(o; p, \theta) = \frac{\sum_{i=1}^{N} \exp\left(-\frac{\|o-p_i\|^2}{h}\right)\left(\theta_i^{\mathrm{T}} o\right)}{\sum_{i=1}^{N} \exp\left(-\frac{\|o-p_i\|^2}{h}\right)} \tag{2.13}$$

where o is the $M \times 1$ observation and $\theta_1 \ldots \theta_N$ are $M \times 1$ vectors of linear regression coefficients. Again, $p_1 \ldots p_N$ are prototype vectors, and h is a positive scale factor.

2.5 Stagewise Training of the Mixture of Experts Estimator

Fitting the Mixture of Experts estimator in Equation 2.13 involves choosing the number of experts, N_E, fitting the linear regression coefficients, $\theta_1 \ldots \theta_{N_E}$, and choosing the prototype vectors, $p_1 \ldots p_{N_E}$. Below, I will refer to the combination of a set of linear regression coefficients, θ_i, and a prototype patch, p_i, as an expert. This section will show how all of these tasks can be accomplished quickly and easily by fitting the estimator in a stagewise fashion.

Stagewise learning can be understood by contrasting it with the traditional optimization approach. As mentioned above, the parameters of an estimator can be found by minimizing a loss function that penalizes the estimator's error on the training set. This minimization is typically done by iteratively updating the parameters of the estimator until the algorithm reaches a local minimum of the loss function. In the context of fitting a Mixture of Experts estimator, taking this approach entails fixing the number of experts, then iteratively adjusting the prototype patches and associated linear regression coefficients to minimize the loss function.

The stagewise approach to fitting the estimator is fundamentally different because it relies on the estimator being composed of a combination of experts. Like the optimization-based approach, fitting the estimator to the training data is a set of iterative steps that reduce the loss. However, instead of modifying a fixed set of parameters at each iteration, the loss is reduced at each iteration by adding a new expert to the estimator.

To understand this, consider a Mixture of Experts estimator with N_E experts:

$$g(o; p, \theta) = \frac{\sum_{i=1}^{N_E} \exp\left(-\frac{\|o-p_i\|^2}{h}\right)\left(\theta_i^{\mathrm{T}} o\right)}{\sum_{i=1}^{N_E} \exp\left(-\frac{\|o-p_i\|^2}{h}\right)} \tag{2.14}$$

This estimator can be improved in two ways. In the optimization approach, the prototype patches $p_1 \ldots p_{N_E}$ and the regression coefficients $\theta_1 \ldots \theta_{N_E}$ are modified to minimize the loss over the training set. Alternatively, this estimator could be improved by adding a new expert. This new expert would correct predictions in regions where the estimator performs poorly. The stagewise approach to fitting an estimator minimizes the loss function in just this manner. At each iteration of the training algorithm, a new expert is added so that the loss over the training set decreases.

2.5.1 Stagewise Versus Iterative Training

At first glance, iterative training may appear to be the superior strategy because the parameters of the estimator are jointly optimized to minimize the loss. However, the key disadvantage of this approach is that the number of experts must be fixed. Finding the correct number of experts for the estimator requires repeatedly fitting estimators with increasing numbers of experts until the correct number is found using some model selection criterion such AIC or BIC [6]. For complicated problems, a number of experts may be needed, making this process of repeatedly training quite slow. In addition, I speculate that optimizing over both the prototype patches and regression coefficients will be particularly difficult because the training process will be very sensitive to the initialization of the prototype patches.

The key advantage of stagewise training is that it is not necessary to fix the number of experts. The correct number of experts is determined using the stopping criterion for training. As Chapter 3 will show, estimators trained in this fashion perform very well when estimating image constraints.

2.5.2 Fitting the Mixture of Experts Estimator

Adding an expert to the estimator requires choosing a prototype patch and regression coefficients at every iteration. As shown below, once the prototype patch is chosen, finding the regression coefficients is simple. This makes choosing the prototype image patch the most difficult problem.

When deciding on a strategy for choosing the next patch, the overriding concern is computational efficiency. For a difficult image estimation problem, such as estimating shading or albedo on a real-world surface, large training sets, with hundreds of thousands of training examples, will be necessary to capture the complicated statistics of a real-world surface. In addition, a fast training algorithm is beneficial for research because it facilitates experimentation.

For this reason, I have chosen a simple, greedy strategy for choosing the next prototype patch. At each iteration, the training example with the highest loss is chosen to be the next prototype patch. This greedy algorithm will be referred to as the ExpertBoost algorithm, due to its similarities with the AdaBoost [28] algorithm for learning a classifier. The actual algorithm for fitting the estimator is shown in Algorithm 1. The following sections describe each step of the ExpertBoost algorithm.

The steps in the headings refer to the steps in Algorithm 1.

Input

The input to this algorithm is a set of M observed image patches, $o_1 \ldots o_M$ and the derivative value at the center of each patch, $c_1 \ldots c_M$. The scale factor h from the exponent in Equation 2.13 must also be provided. This scale factor can be chosen using cross-validation on the training set.

Note that an $n \times n$ image patch from the training set can be unwrapped into a $n^2 \times 1$ vector. For convenience, we will assume that each of the training samples has been unwrapped into a vector.

The linear regression associated with each expert should include a bias term. The most convenient way to add this bias term is to add another dimension to the training samples. The value of this dimension should be 1 for every training sample. If the prototype patches are augmented similarly, this will not affect the similarity value between an observation and a prototype patch. For all that follows, I will assume that the training examples have been augmented in this fashion.

Steps 1-6: Initialization

In Steps 1-6, the algorithm is initialized by setting the first prototype, p_1, to be an arbitrary training sample. The associated linear regression coefficients are chosen to be the linear regression from the observations to the ground-truth filter responses, c, with the minimum mean-squared error (MMSE). If the training examples $o_1 \ldots o_M$ are $n \times 1$ vectors, this can be computed quickly by first concatenating the M training examples into an $n \times M$ matrix, denoted O. The regression coefficients θ_1, which will be associated with p_1, can then be found using a Moore-Penrose pseudo-inverse:

$$\theta_1 = (OO^T)^{-1}O\hat{c} \tag{2.15}$$

where $\hat{c}$ is a vector consisting of $c_1 \ldots c_M$.

Once the initial prototype patch is chosen, the algorithm then adds $N_E - 1$ experts, one by one, to the estimator. The following steps occur at each of the $N_E - 1$ iterations.

Step 8: Adding the next patch

At each iteration in the algorithm, the first step is to choose the next prototype patch to be added to the system. As mentioned before, this is primarily accomplished using

Algorithm 1 ExpertBoost

Require: N_E, the number of prototype patches to be added to the model. A set of M training examples, $(o_1, c_1) \ldots (o_M, c_m)$. A scale parameter, h

$i \leftarrow 1$

Choose an arbitrary observation to be the initial prototype, p_1.

$\theta_1 \leftarrow \arg\min_\theta \sum_{m=1}^{M} (\theta_1 \cdot o_m - c_m)^2$

for $m = 1 \ldots M$ **do**

  set r_m^i, which is the current estimate of c_m, to the linear estimate of c_m from o_m

end for

for $i = 2 \ldots N$ **do**

  Add a patch, o_n, to the prototype database by setting patch $p_i = o_n$, where $n = \arg\max_n (r_n^i - c_n)^2$

  for $m = 1 \ldots M$ **do**

    $d_m \leftarrow e^{-\sum_j (o_m^j - p_n^j)^2}$, where p_i^j denotes value j in the patch with index i. A similar notation is used for the observations o.

    $d_m^0 \leftarrow \sum_{l=1}^{i} e^{-\sum_j (o_m^j - p_l^j)^2}$

  end for

  Set $\theta_i = \arg\min_\theta \sum_{m=1}^{M} \left(\frac{d_m^0 r_m^{i-1} + d_m(\theta_i^T o_m)}{d_m^0 + d_m} \right)^2$

  for $m = 1 \ldots M$ **do**

    Update r_m^i to be the current estimate of c_m from o_m.

  end for

end for

a greedy approach. At each iteration, the current estimate of each target value is updated for each training example. The training example with the largest error in the estimate is chosen to be the next prototype patch in Step 8.

Steps 9–13: Fitting the regression coefficients

Once the next prototype patch has been chosen, the next step is to fit the linear regression coefficients for the new expert. For the squared-error loss function, these coefficients can be computed efficiently.

The first step is to compute two intermediate quantities, d_m and d_m^0, where m indexes into the M training examples. The quantity d_m is the similarity between the newest prototype patch and each of the training examples. For each training example o_m in $o_1 \ldots o_M$, d_m is computed as

$$d_m \leftarrow e^{-\sum_j (o_m^j - p_n^j)^2} \tag{2.16}$$

where p_n^j denotes value j in the patch with index n, and d_m is computed for each m in $1 \ldots M$. The notation is similar for the observations o.

The other intermediate quantity that needs to be computed is d_m^0, which expresses each patch's similarity to the previous prototype patches:

$$d_m^0 \leftarrow \sum_{l=1}^{i} e^{-\sum_j (o_m^j - p_l^j)^2} \tag{2.17}$$

Again, this is computed for each m in $1 \ldots M$. The sum is over all of the previous prototype patches chosen by the system.

With these quantities, the new regression coefficients, θ_i, can be found by minimizing the error in the updated predictions:

$$\theta_i \leftarrow \arg\min_{\theta} \sum_{m=1}^{M} \left(\left(\frac{d_m^0 r_m^{i-1} + d_m(\theta_i^T o_m)}{d_m^0 + d_m} \right) - c_k \right)^2 \tag{2.18}$$

where r_m^{i-1} is the current estimate of c_m before the new expert is added.

Interestingly, Equation 2.18 can be rewritten as a weighted regression:

$$\theta_i \leftarrow \arg\min_{\theta} \sum_{m=1}^{M} \left(\frac{d_m}{d_m^0 + d_m} \right)^2 \left(\left(\frac{-d_m^0 r_m^{i-1} + c_m}{d_m} \right) - \theta_i^T o_m \right)^2 \tag{2.19}$$

In this form of the equation, the weight of a training example is determined by two factors. First, examples that are more similar to prototype patches already chosen receive less weight. At the same time, examples that are similar to the newest prototype patch receive more weight.

The regression coefficients can be found easily in closed form:

$$\left(O^T W O\right)^{-1} O^T W \tilde{c} \tag{2.20}$$

where O is the matrix of observations, as in Equation 2.15. The matrix W is a diagonal matrix with the diagonal terms set to be

$$W_m = \left(\frac{d_m}{d_m^0 + d_m}\right)^2 \tag{2.21}$$

where W_m is the value on the diagonal in the mth row. The vector $\tilde{c}$ contains the new target values:

$$\tilde{c}_m = \frac{-d_m^0 r_m^{i-1} + c_m}{d_m} \tag{2.22}$$

For stability, the regression can be regularized by adding a scaled identity matrix, αI to the $(O^T W O)$ term. For best performance, this matrix should be modified so that no regularization is applied to the bias term. This allows the bias term to be set freely and encourages the linear terms to have as low a norm as possible.

For large training sets, these steps can be accomplished much more quickly by first selecting a subset of training examples that are most similar to the new prototype.

2.5.3 Beyond Squared-Error

In the previous sections, the ExpertBoost algorithm is derived for minimizing the squared-error loss function. However, Equation 2.18 can be rewritten for any arbitrary loss function, $L(\cdot)$:

$$\theta_i \leftarrow \arg\min_{\theta} \sum_{m=1}^{M} L\left(\frac{d_m^0 r_m^{i-1} + d_m(\theta_i^T o_m)}{d_m^0 + d_m}, c_k\right) \tag{2.23}$$

While conceptually simple, there is a caveat to this statement. The squared-error loss is unique in that there is a simple closed form solution to the arg min task. It is unlikely that other loss functions will have similar simple solutions.

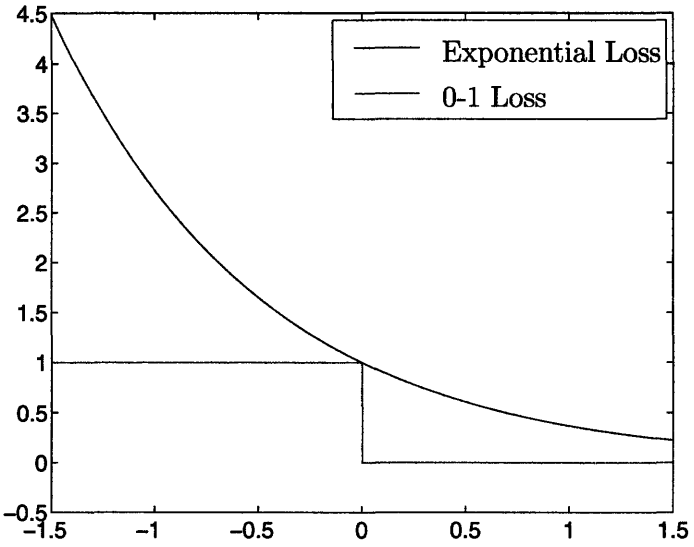


Figure 2-4: The 0-1 loss, plotted in green, is the ideal loss for classification, but is not smooth. The exponential loss is a smooth upper bound on the 0-1 loss.

2.5.4 Classification using ExpertBoost

The ExpertBoost algorithm's ability to generalize to arbitrary loss functions allow it to be used to learn a binary classifier. Binary classification can be viewed as a regression problem where every target value is $+/-1$. The typical approach to classification is to learn a function $f : \mathbb{R}^M \rightarrow \mathbb{R}$ that maps the M dimensional observation vector to a scalar. For an observation o, if $g(o) > 0$, then the classifier returns $+1$ as the estimated label of the observation. On the other hand, if $g(o) \leq 0$, the classifier returns -1. Unlike regression, the goal is not to learn to predict a certain value. Instead the goal is to predict a value with the correct sign.

In most cases, the ideal loss function for learning a classifier is the 0-1 Loss, plotted in green in Figure 2-4 for the case when the correct label is $+1$. Using this loss function, the loss only depends on the prediction having the correct sign. Unfortunately, this loss is not differentiable, which makes minimizing this loss difficult.

A differentiable alternative to the 0-1 Loss is the exponential loss [31]:

$$L(f(x_i;\theta), c) = e^{-cf(x_i;\theta)} \tag{2.24}$$

This loss, plotted in blue in Figure 2-4, is an upper bound on the 0-1 Loss. Friedman et. al also point out that the $f(x)$ that minimizes the exponential loss coincides with the maximizer of the logistic log-likelihood [31].

Training a classifier using the ExpertBoost algorithm, under the exponential loss,

can be accomplished using an algorithm similar to the GentleBoost algorithm [31]. For simplicity, the derivation will focus on a classifier of the form:

$$\hat{t}(o) = \frac{\sum_{i=1}^{N} \exp\left(-\frac{||o-p_i||^2}{h}\right)\theta_i}{\sum_{i=1}^{N} \exp\left(-\frac{||o-p_i||^2}{h}\right)} \tag{2.25}$$

where θ_i is a scalar. The primary difference between this classifier and the Mixture of Experts estimate in Equation 2.13 is that only a constant is associated with each expert, rather than a set of linear regression coefficients.

For the exponential loss, there is not a simple closed-form method of finding the regression coefficients in Step 13. However, because the exponential loss is convex, a quadratic approximation can be substituted instead. At iteration i, the exponential loss, L, for any set of regression coefficients, θ_i, is:

$$L = \sum_{m=1}^{M} \exp\left(-c_m\left[\frac{d_m^0 r_m^{i-1} + d_m\theta_i}{d_m^0 + d_m}\right]\right) \tag{2.26}$$

where c_m is the true label of example m. This can be approximated with a Taylor series, expanded around $\theta_i = 0$:

$$L \approx \sum_{m=1}^{M} \exp\left(-c_m\left[\frac{d_m^0 r_m^{i-1}}{d_m^0 + d_m}\right]\right)\left(1 + \theta_i\left(\frac{c_m}{d_m^0 + d_m}\right) + \theta_i^2\left(\frac{c_m}{d_m^0 + d_m}\right)^2\right) \tag{2.27}$$

The value of θ_i is found by minimizing this quadratic approximation:

$$\theta_i = \frac{\sum_{m=1}^{M} e^{-c_m\frac{r_m^i d_m^0}{d_m^0 + d_m^i}}\left(\frac{d_m^i c_m}{d_m^0 + d_m^i}\right)}{\sum_{k=1}^{M} e^{-c_m\frac{r_m^i d_m^0}{d_m^0 + d_m^i}}\left(\frac{d_m^i}{d_m^0 + d_m^i}\right)^2} \tag{2.28}$$

In [31], these types of stagewise updates are referred to as taking Newton steps because iteratively solving a quadratic approximation is similar to Newton's method for minimizing non-linear functions.

2.5.5 Related Work on Stagewise Methods

This stagewise approach to fitting an estimator has been used previously for fitting additive models of the form:

$$f(x,\theta) = \sum_{i=1}^{I} \theta_i h(x,\gamma_i) \tag{2.29}$$

where the functions $h(x,\gamma_1) \ldots h(x,\gamma_I)$ are often called "basis functions" or "weak learners". These basis functions are parameterized by $\gamma_1 \ldots \gamma_I$. [34] This general form subsumes many popular types of regression methods, including radial-basis function regression, spline regression, and kernel regression.

One of the first examples of the application of stagewise modelling to this additive regression model was the Projection Pursuit Regression algorithm [29]. In [44], Mallat and Zhang use this strategy, which they call matching pursuit, to choose a set of wavelet basis functions for representing signals.

This additive model is also the model produced by the popular AdaBoost [28] algorithm for learning a classifier. In the context of boosting, the functions $h_1 \ldots h_I$ are known as weak learners because each individual weak learner is expected to have poor discriminative power. However, combined together, as in Equation 2.29, these weak learners form a strong classifier. Friedman et al. [31] and Collins et al. [19] showed the connection between the AdaBoost algorithm and stagewise minimization of the exponential loss.

After the introduction of the AdaBoost algorithm, its connection to minimizing the exponential loss was not initially understood. This led to regression mechanisms that relied on using the AdaBoost algorithm to learn a classifier. An example of this type of regressor, that on the surface appears to be quite similar to the regression model that I have proposed, is the work of Avnimelech and Intrator, entitled "Boosted Mixture of Experts: An ensemble learning scheme" [8]. This algorithm operates by fitting a new expert to training examples that the previous experts predicted with low confidence. A classifier is then retrained to classify which estimator should be used for each example.

The primary difference between these additive models and the regression model presented in this chapter is the normalizing term in Equation 2.13. Section 3.10 compares the performance of an additive model, such as Equation 2.29, and the Mixture of Experts model for estimating shading images.

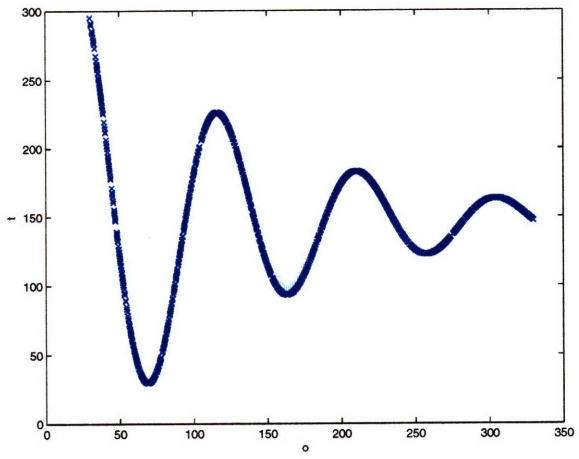


Figure 2-5: The training data used in the initial demonstrations of the ExpertBoost algorithm. In this data set, there is no noise.

2.6 Visualizing the ExpertBoost Algorithm

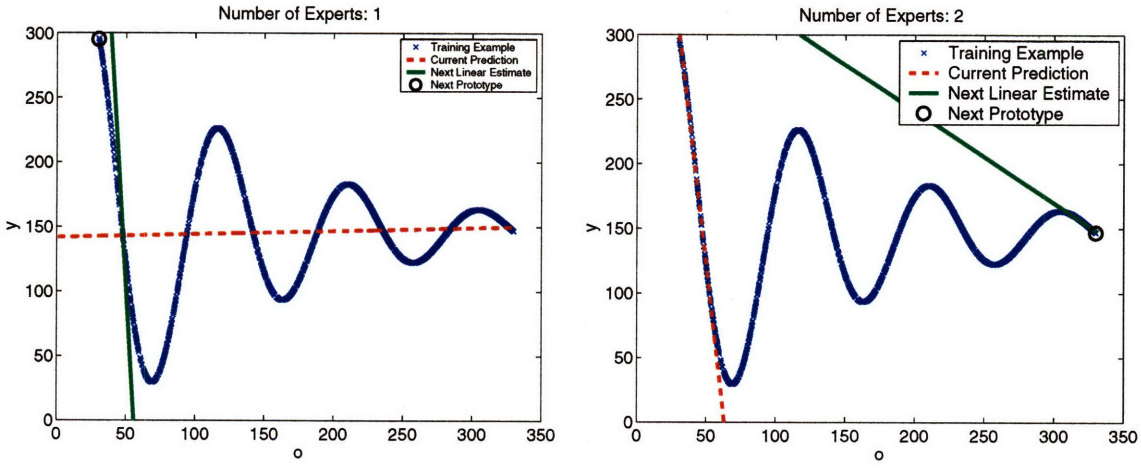
To understand the behavior of the ExpertBoost algorithm, this section will show how it learns to predict a one-dimensional function. To begin with, consider learning from a set of training pairs, $(o_1, t_1) \ldots (o_M, t_M)$, shown in Figure 2-5. In this data, given o, there is no uncertainty in the correct value of t. This data could be considered as being drawn from a distribution that would appear as a line with a similar shape to the distribution shown in Figure 2-1.

2.6.1 The Training Process

After adding the first expert, the estimator is equivalent to a linear estimator, so the regression coefficients chosen for the first expert, depicted with the red line in Figure 2-6(a), are the optimal linear regression coefficients. The next step is to choose the training sample with the highest error as the next prototype patch. This next prototype is marked with the black circle in Figure 2-6(a).

The linear regression coefficients associated with this expert are chosen to minimize the error for training examples near the next prototype. This regression, depicted as the green line in Figure 2-6(a), matches the steep relationship between o and t around the next prototype patch.

After adding this expert, the estimator, depicted as the dashed red line in Figure 2-6(b), produces good predictions where o is near zero, but very poor predictions where o is near 350. Consequently, the training example with the highest error now



(a) (b)

Figure 2-6: The first two iterations of the ExpertBoost algorithm. The estimator is initialized to be the best linear estimator of the data. See the text for details. Figure (b) shows the third prototype patch to be added to the estimator.

lies on the right side of Figure 2-6(b) and is chosen to be the next prototype. The linear regression associated with this prototype, again depicted as a green line in Figure 2-6(b), expresses the relationship between o and t on the right side of Figure 2-6(b).

After these three experts have been added, the estimator, again depicted as a red line in Figure 2-7(a), is the least accurate in the middle of Figure 2-7(a). Figure 2-7 shows how the estimator improves as more experts are added. At first, the estimator jumps wildly and has poor behavior. However, as shown in Figure 2-7(e), after 14 experts have been added, it performs quite well. After adding 24 experts, the estimator has a nearly perfect fit, as seen in Figure 2-7(f).

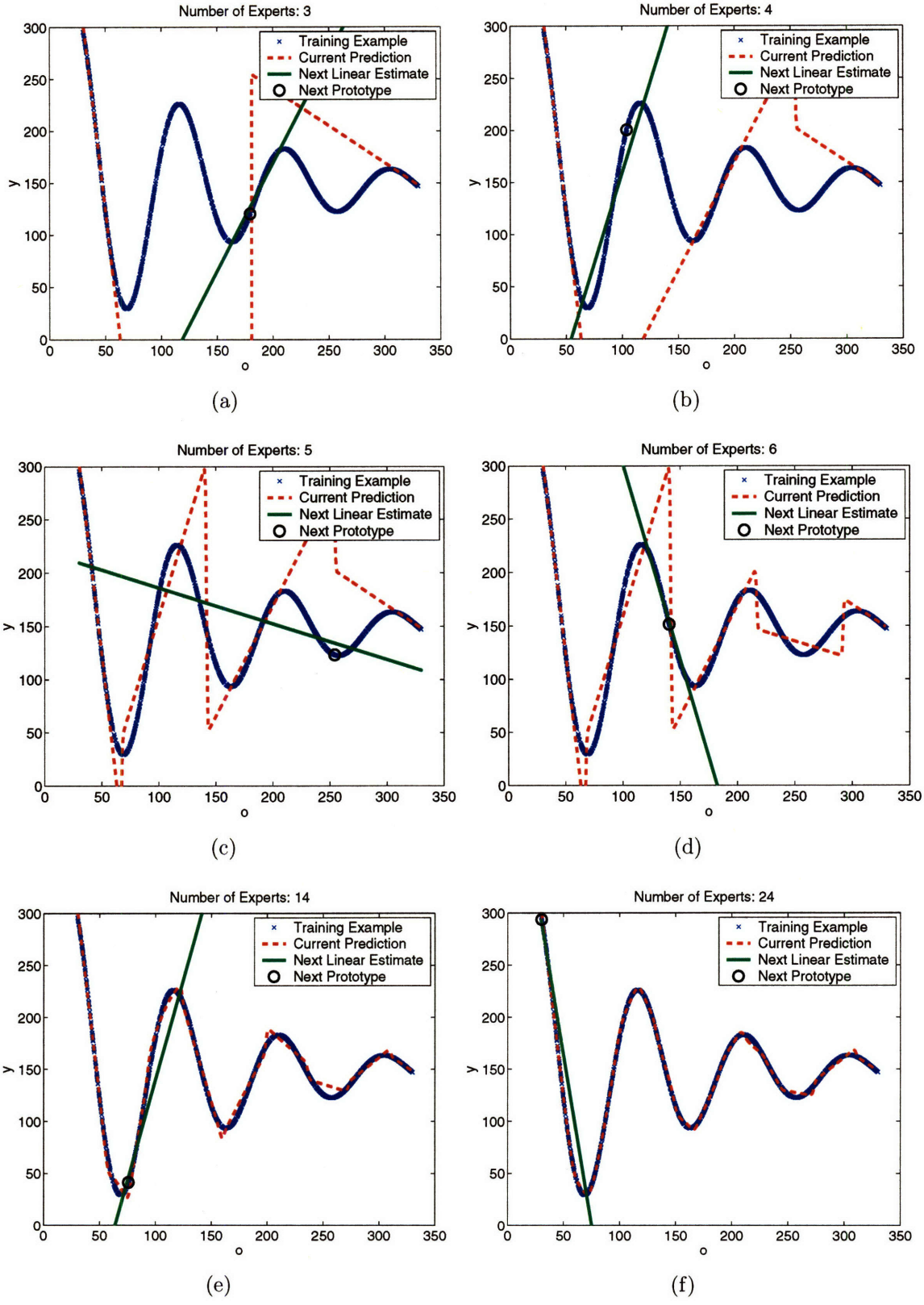
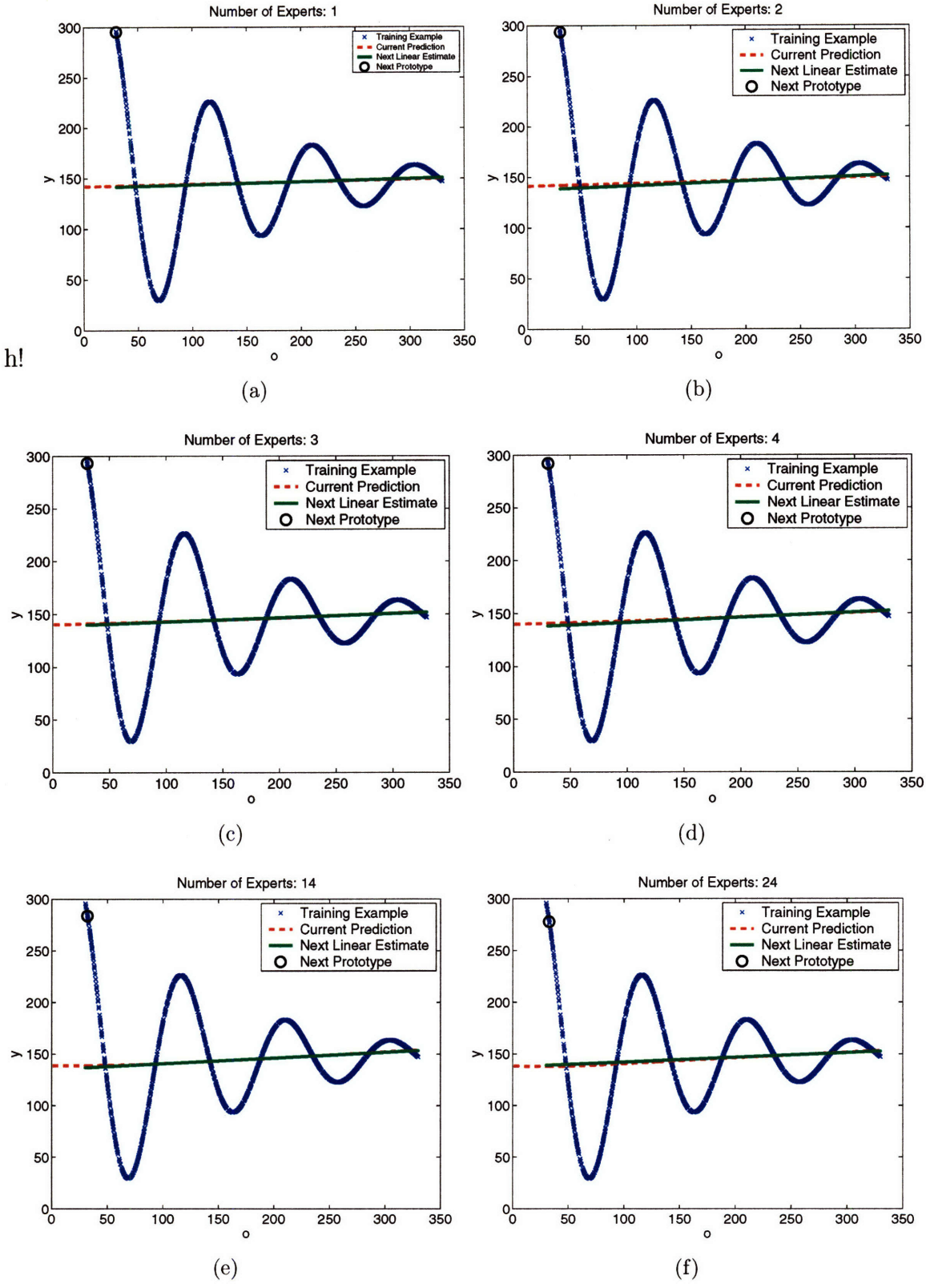


Figure 2-7: These plots show the evolution of the Mixture of Experts estimator as experts are added. At first, the estimator is a poor fit, but quickly becomes a good fit.

2.6.2 The Importance of Local Fitting

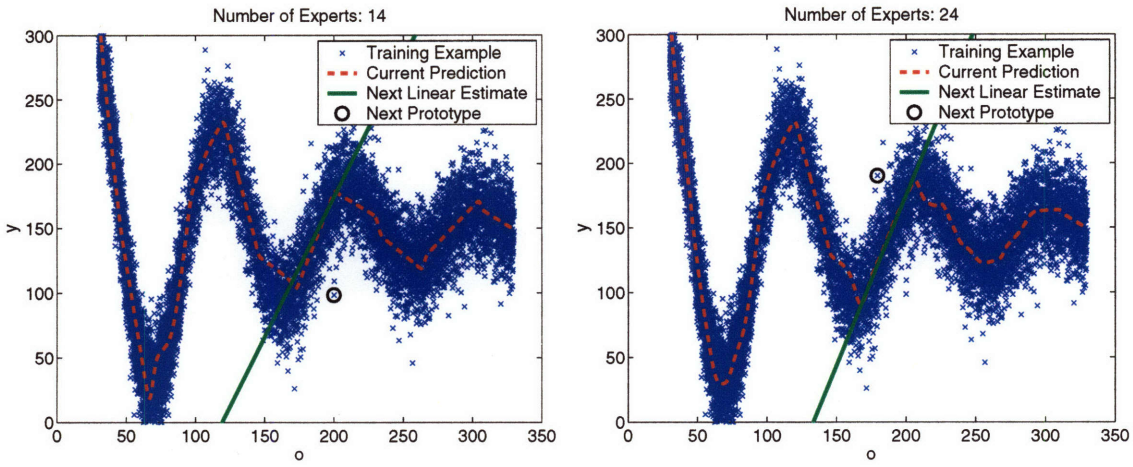
For a problem without noise, such as the simple example just demonstrated, it is vital to only use training examples near the next prototype patch when fitting the linear regression coefficients. Figure 2-8 shows the how the estimator is fit if the linear regression coefficients are fit to the whole training set each time. The initial linear fit is identical for both methods. However, as can be seen in Figure 2-8, the two methods then begin behaving quite differently. Instead of improving the estimator for different ranges of o, the second method simply selects points from the range of o close to zero, without fitting a significantly different estimator.

This occurs because the points toward the left of Figure 2-8(a) consistently have the highest error. Because the algorithm is compelled to minimize the error over the whole training set, rather than just a subset of examples near to the next prototype, the criterion for fitting the linear regression coefficients is essentially identical to the criterion used to fit the previous set of coefficients. As Figure 2-8(f) shows, this behavior still continues after large numbers of experts are added.



h!

Figure 2-8: In this example, the regression coefficients are trained to minimize the squared error over the whole training set, rather than just the subset of the training set that is close to the next prototype. This leads to pathological behavior.



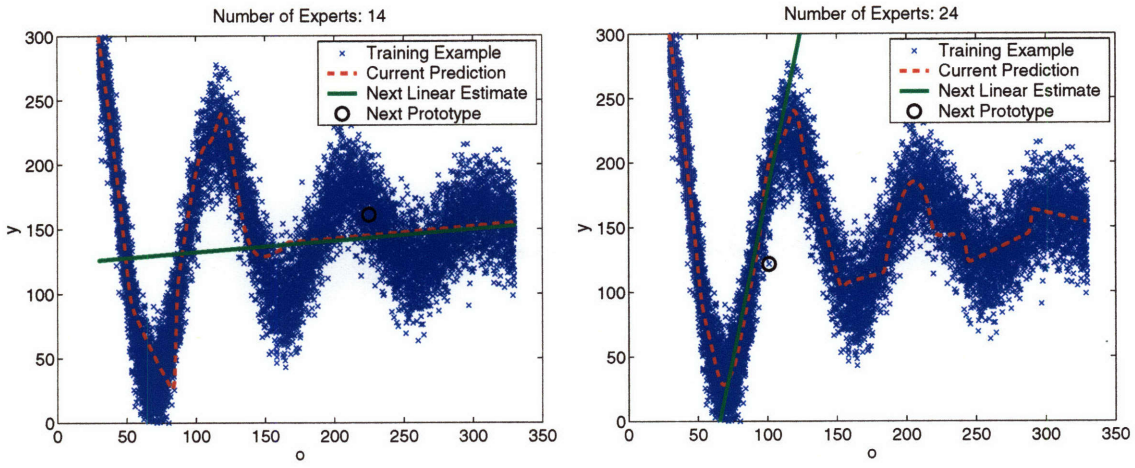
(a) (b)

Figure 2-9: These figures show the estimators trained on noisy data, with different numbers of experts added. Despite the noise in the data, the estimators trained with ExpertBoost algorithm are still a good fit.

2.6.3 Resiliency to Noise

Given the greedy nature of this algorithm, it could be expected that this algorithm would perform poorly on noisy data with outliers. However, as shown in Figure 2-9, the algorithm performs quite well. As expected, the algorithm chooses outliers as the prototype patches. Despite this, the ExpertBoost algorithm still learns a good estimator because the linear regression coefficients are fit to all the training points near each prototype. This makes the step of fitting regression coefficients robust to outliers.

Interestingly, unlike the noise-free example in Figure 2-8, fitting the regression coefficients to the whole training set, rather than just a local subset, performs well in this case. Figure 2-10 shows how the estimator develops in this case. With only a few experts, the algorithm behaves similarly to the noise-free case shown in Figure 2-10. However, as can be seen in Figure 2-10, the estimator is able to adapt to the local structure on the left side of the figures. This is possible because the noise in the data randomizes the order in which the prototypes are chosen. In addition, as more experts are added, fitting the regression coefficients to the whole training set on noisy data finds an estimator that is very similar to the estimator found when fitting the regression coefficients to subsets of the data, as can be seen in Figures 2-10(a) and 2-10(b).



(a) (b)

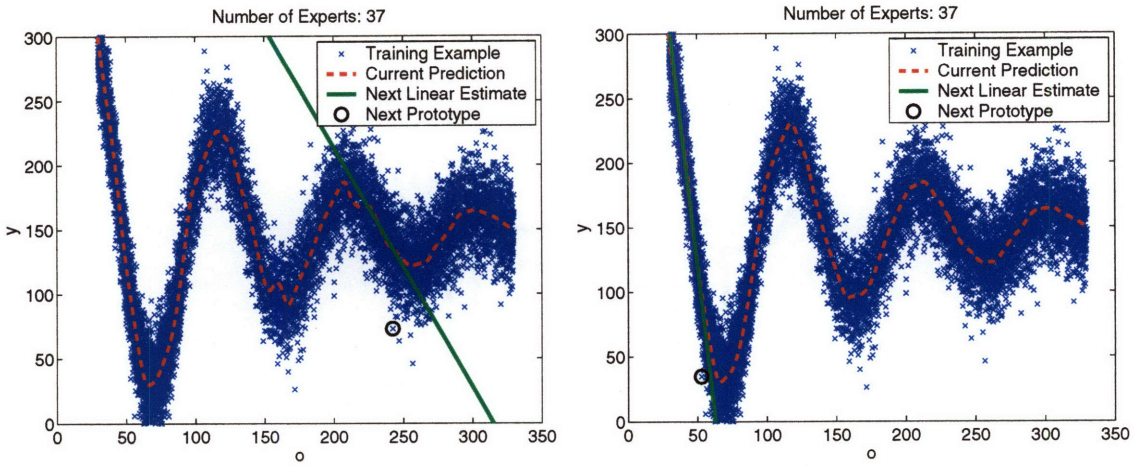
Figure 2-10: While fitting the regression coefficients to minimize the error over the whole training set did not work in the previous example, these plots demonstrate that it works well on data with noise.

2.6.4 Benefits of Randomly Chosen Patches

In addition to the greedy strategy for choosing the next prototype patch, it is helpful to occasionally choose the next prototype patch at random. The primary motivation for choosing patches at random is that it allows the algorithm to choose prototypes other than the outliers on which it tends to focus.

To examine the benefit of drawing patches at random, I changed Step 8 from Section 2.5.2 so that at each iteration there is a 40% chance of drawing a random patch. If a random patch is to be drawn, that patch is drawn from all those examples whose error is higher than the median error 65% of the time. In this experiment, these probabilities were set manually, although they could also be set using cross-validation. Generally, I found that the performance of the estimator was not sensitive to the values of these probabilities.

The advantage of choosing patches randomly can be seen after a number of experts have been added. Figure 2-11 shows the estimator trained using the strictly greedy strategy, shown in Figure 2-11(a), and the estimator found using some randomly chosen experts, shown in Figure 2-11(b). The estimator with randomly chosen prototypes has a smoother fit with fewer experts.



(a) Strictly Greedy Strategy

(b) With Random Path Draws

Figure 2-11: The plot in (b) shows the estimator found by occasionally drawing patches at random, rather than pursuing a strictly greedy strategy, as shown in (a). Adding randomly chosen prototypes leads to a smoother estimator.

2.7 Comparison with Support Vector Regression

To demonstrate the flexibility of a Mixture of Experts estimator trained using the ExpertBoost algorithm, this section will compare this estimator to an estimator found using Support Vector Regression [75] on a benchmark regression problem. Estimators trained using Support Vector Regression are useful for this comparison for several reasons:

1. Like the particular Mixture of Expert estimators used in this work, these estimators are based on a small subset of the training examples.

2. Efficient, well-tested packages for training estimators are freely available [16].

3. If Gaussian RBF kernels are used, then the estimator has a very similar form to the Mixture of Experts estimators used in this work.

Support Vector Regression, or SVR, is related to Support Vector Classification [75]. There is a large amount of research in this field, which will not be summarized in this thesis, however, excellent tutorials are available [15, 59]. Smola and Schölkopf have also published a tutorial specifically related to Support Vector Regression [62].

Assuming that a Gaussian RBF kernel is used as the kernel for the estimator, the form of the estimator is:

$$g(o) = b + \sum_{i=1}^{N_{SV}} w_i \exp\left(-\gamma ||o - p_i||^2\right) \tag{2.30}$$

where o is the observation, $w_1 \ldots w_{N_{SV}}$ are scalar coefficients, b is a scalar bias term, and $p_1 \ldots p_{N_{SV}}$ are training examples, analogous to the prototype patches chosen by the Mixture of Experts estimators. In a SVR system, these are known as support vectors.

To compare the two estimators, I used the problem of estimating the scalar function:

$$y(x_1, x_2, \ldots, x_{10}) = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 \tag{2.31}$$

When generating samples, each x was uniformly distributed between 0 and 1. In addition, zero-mean Gaussian noise, with unit variance, was added to each target value. By design, only five of the dimensions contribute to $y(x_1, x_2, \ldots, x_{10})$. This function was first proposed in [30] and has also been used for comparison in [72, 74].

The parameters of both algorithms were hand-tuned on a small sample of the training data. The scale factor was set to 2^3 for the Mixture of Experts estimators and the kernel parameter, γ, was set to 2^{-1} for the Support Vector Regression estimators. Both algorithms were trained on 30,000 examples and tested on 10,000 examples.

The Mixture of Experts model, with 1000 experts, produced estimates with a mean squared-error of 1.17, while a SVR estimator with 1137 support vectors produced estimates with a mean squared-error of 1.77. Increasing the number of support vectors to 3209, led to a mean squared error of 1.45.

Given that the two algorithms perform comparably on this simple test, the chief advantage of fitting an estimator with the ExpertBoost algorithm for this problem is that it is relatively simple to control the number of experts in the estimator. On the other hand, controlling the number of support vectors when fitting a SVR estimator is quite tricky. The number of support vectors is influenced by two parameters and altering either of these parameters can both increase and decrease the number of support vectors in the estimator.

While the performance of these algorithms is similar for this problem, Section 3.10 will show how estimators trained using the ExpertBoost algorithm will produce much better estimates of shading images.

2.8 Conclusion

This chapter has described a greedy, stagewise algorithm for learning a Mixture of Experts estimator. The greedy, stagewise approach allows the algorithm to scale to large data sets. The next chapter will show how this estimator is applied to the task of estimating intrinsic component images. In addition, Section 3.10 will show that the Mixture of Experts estimator compares very well against kernel regression.

Chapter 3

Estimating Intrinsic Component Images

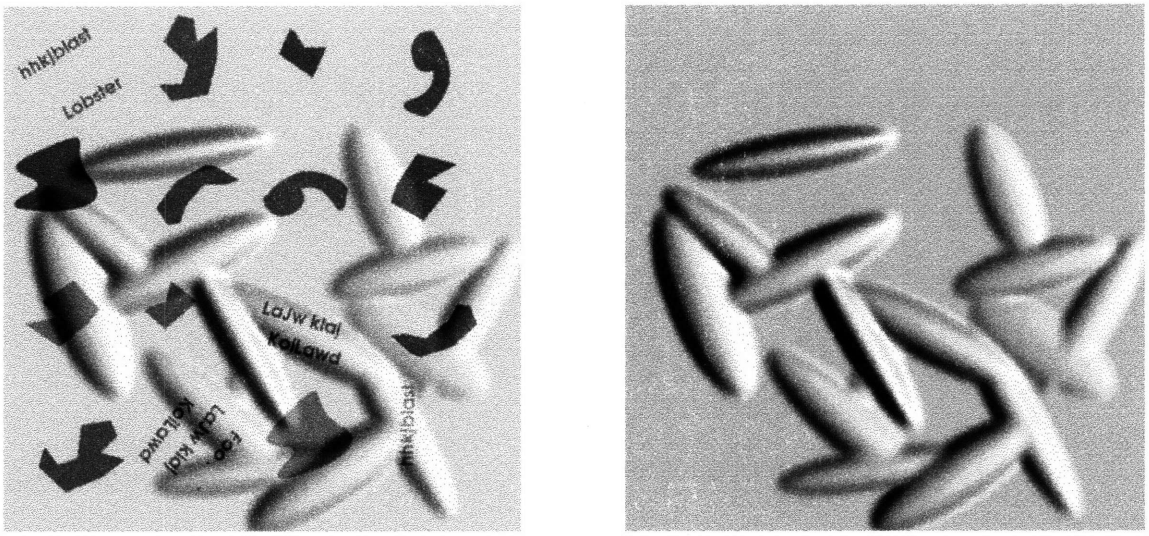
Chapter 2 has shown how to efficiently learn a Mixture of Experts estimator by minimizing the squared-error over a training set. The quality of the estimated images depends on good estimates of the filter responses. While conceptually straight-forward, there are a number of choices to be made when implementing these estimators.

This chapter focuses on evaluating these different choices, both quantitatively and qualitatively. The primary focus will be the task of estimating shading images, but Section 3.12 demonstrates the effectiveness of the system on the task of denoising images.

In addition to evaluating various configurations of the estimator, Section 3.12 demonstrates how the Mixture of Experts estimators produce much better estimates of the derivatives of the shading image than estimators based on the same heuristic as the Retinex algorithm, which was discussed in Section 1.2.2. Section 3.2 discusses the implementation of the pseudo-inverse step.

3.1 Training Data

To capture of the complicated statistics of real-world surfaces, the estimators are trained on real-world image data. This section describes how the data sets were gathered for the tasks of estimating shading images and denoising images.



(a) Observed Image (b) Shading Image

Figure 3-1: Synthetic training images for shading and albedo.

3.1.1 Creating Denoising Data

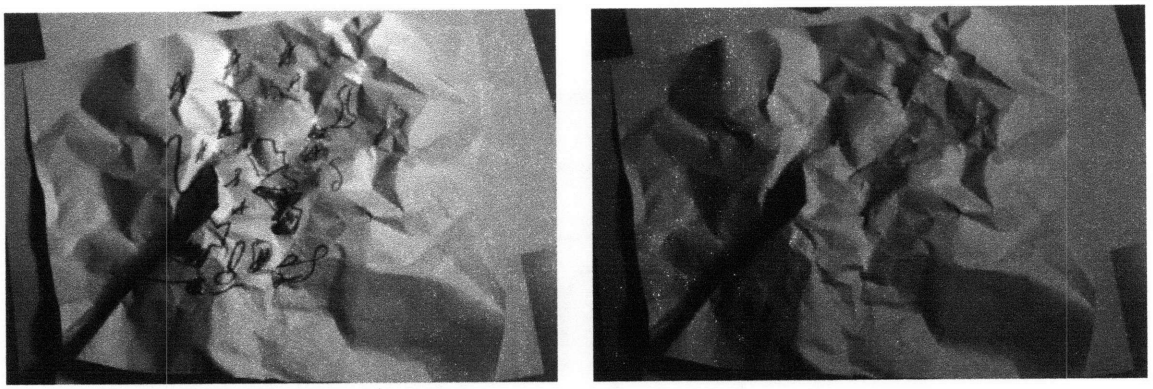
Generating training data for denoising images is simple. The training set is created from a set of 80 images taken from the Berkeley Segmention Database [45]. The details of the processing steps will be described in Section 3.12.

3.1.2 Creating Shading and Albedo Data

On the other hand, generating training data for estimating shading and albedo images can be difficult. Generating training data for this problem requires an image of surfaces both with and without albedo changes.

Gathering Real-World Data

One relatively simple option is to generate synthetic data, such as the examples in Figure 3-1. This is the approach taken in [10, 69]. The chief advantage of this approach is that there are no artifacts in the training data. This makes synthetic data especially useful in the early stages of research for validating methods and verifying intuitions. However, synthetic data is also dissatisfying because the method chosen for generating the images imposes strong assumptions about the world. For instance, training on the images shown in Figure 3-1 is equivalent to assuming that surfaces are made from smooth blobs that have a Lambertian reflectance and are illuminated with



(a) Red Channel (b) Green Channel

Figure 3-2: These images are the red and green channels of a photograph of a piece of paper colored with a green maker. The coloring can be seen in (a) but not in (b). We use this property to construct a training set.

a relatively simple illumination. In addition, the changes in albedo in these images are angular and have no correlation with the shading.

Training only on synthetic data can be viewed as only a small advance beyond Retinex. Retinex is derived by constructing a simplified model of Mondrian images and smooth shading, then using this simple model to decide the optimal method of estimating the shading and albedo. Using synthetic training data generated from a simple model of surfaces still relies on simplified models of shading and albedo images, and substitutes estimators that have been trained on overly simple models of the world.

To make meaningful progress, the training and test data must come from images of real-world surfaces. Even in controlled settings, with multiple cameras and light sources, it is difficult to create shading and albedo images without significant artifacts in the images. As an example, in theory, both basic photometric stereo methods [24] and the example-based photometric stereo method of Seitz and Hertzmann [35] offer the ability to recover both the surface normal and reflectance at each point on a surface. Unfortunately, both of these methods are insufficient because they do not account for shadows on the surface. This is a significant weakness because shadows are common characteristic of real-world surfaces.

Fortunately, the difficulty in obtaining training data can be overcome by using color to measure shading separately from albedo. Figure 3-2(a) shows the red channel of an image taken of a piece of paper colored with a Crayola "Electric Lime-Green" Washable Marker and illuminated with a standard incandescent light. Both the

markings and the surface shading are visible. The green channel of that image, shown in Figure 3-2(b), does not contain any of these markings. The red channel of the image shows the coloring because the green ink absorbs the light in the red spectrum. Hence, the coloring appears dark. Because the paper is white and the coloring is green, the light in the green spectrum is reflected uniformly, leading the paper in Figure 3-2(b) to appear as if it has not been colored. This is exactly how the shading image of this surface should appear. The red channel shows the surface with both shading and albedo changes, while the green channel shows only the shading.

In order to collect data with different types of surface statistics, the training and test sets were collected using three types of paper: office paper, paper towels, and facial tissue. After coloring the paper with a green marker, I used a Canon 10D in RAW mode, which has an approximately linear response, to capture each image. The albedo image was extracted dividing the red and green channels of the image. After finding the shading and albedo images, the data set was created from the log of these images to make the problem additive. This dataset provides photographs of real-world surfaces with high-quality shading and albedo image decompositions. The database will be available for download at `http://www.csail.mit.edu/~mtappen`.

While using paper images may seem like an overly simple data set for evaluating algorithms, examining the images in Figure 3-2 shows that the paper surfaces exhibit many of the characteristics that make this problem difficult, including occlusions, shadows, and complicated geometry. Furthermore, testing on crumpled paper images is useful because these images violate the basic assumptions of Retinex. The shading images have strong edges that would be considered albedo changes under the Retinex model. To my knowledge, previous work has not attempted to recover the shading for images as difficult as these.

Adding Synthetic Images

To further augment the range of image statistics in the set of training and test images, a set of synthetic images was also added to the set of image data. The images, such as those shown in Figure 3-1, were created by generating a random surface out of ellipses, then rendering the surface with a point source illumination and Lambertian reflectance. The albedo images were created from randomly rotated polygons with random intensity.

3.2 Recovering Intrinsic Component Images from Constraint Estimates

The overall goal of the system is to produce good estimates of the intrinsic component images. For each constraint in the set of linear constraints, such as vertical and horizontal derivatives, a Mixture of Experts estimator is used to estimate the value of that constraint for every pixel location in the estimated image. While the estimators are trained to predict constraint values as well as possible, it is also useful to consider the quality of the images reconstructed from these constraint estimates. This section describes how to reconstruct the estimated intrinsic component image from the estimated constraint values.

The variable $\mathcal{X}$ will denote the image estimated from the observed image $\mathcal{O}$. For concreteness, assume that there are two linear constraints: a horizontal discrete derivative and a vertical discrete derivative. Let C_{dx} and C_{dy} denote the matrices that express the 2-D image convolution with each filter as a matrix. It is assumed that the convolution operation produces only those pixels where the convolution kernel is fully contained inside the image. This makes each of the matrices have fewer rows than columns because the convolution is not computed at border pixels. The estimated values of each of these types of derivatives are denoted as $\hat{c}_{dx}$ and $\hat{c}_{dy}$. These estimates are found from $\mathcal{O}$ using the Mixture of Experts estimators.

The estimated image $\mathcal{X}$ is recovered using the Moore-Penrose pseudo-inverse:

$$\mathcal{X} = \left(C^T C\right)^{-1} C^T \hat{c} \tag{3.1}$$

where

$$C = \begin{bmatrix} C_{dx} \\ C_{dy} \end{bmatrix}$$

and

$$\hat{c} = \begin{bmatrix} \hat{c}_{dx} \\ \hat{c}_{dy} \end{bmatrix}$$

It is important to note that the constraints are not limited to first-order derivatives. The matrix C can be made up of arbitrary linear constraints. For some combinations of functions, such as derivatives, C will be a singular matrix. In these cases, we include a constraint specifying the value of a small number of pixels. For all of the results shown, $\mathcal{X}$ is found using Gaussian Elimination in MATLAB (the

"backslash" operator). If computation is an issue, C can also be solved with the conjugate gradient algorithm or FFT [78].

3.3 Evaluating the Mixture of Experts Estimators

The remainder of this chapter will focus on two main goals:

1. Demonstrating the effectiveness of the Mixture of Experts estimators.

2. Evaluating specific configurations of the estimator.

The second point is important because there are a number of choices available when setting the specifics of the estimator, including the choices of features and loss function. Sections 3.4 - 3.9 will quantitatively evaluate some of these choices.

While the problem of estimating shading images will be the basis of most of the evaluations, Section 3.12 will show the system's effectiveness for the problem of denoising images. After settling upon a final configuration for the estimator, Section 3.7 will show how the Mixture of Experts estimator outperforms a continuous-valued Retinex estimator. Section 3.10 will demonstrate the superior performance of the Mixture of Experts estimators when compared with estimators found with Support Vector Regression.

3.3.1 The Training and Test Sets

The performance of the method was evaluated using a training set of 22 paper images and a test set of 11 paper images. In addition, ten synthetic images were added to the training set. I found that the system generalized to the examples used in [69] much better with the addition of these images. Four similar synthetic images were also added to the test set. The paper images were all created using the method described in Section 3.1.2. The size of all the images was 127 $\times$ 127 and the gray-scale values of each image ranged roughly between 0 and 255.

3.4 Evaluating Image Patch Representations: Image Patches versus Multi-scale Representations

The evaluation will begin with a straightforward configuration. As a baseline, I trained estimators to predict horizontal and vertical derivative filters. The input for these estimators is a 5×5 image patch, with a constant subtracted from each patch so that its mean is zero. The form of the horizontal and vertical filters, c_{dx} and c_{dy}, was

$$c_{dx} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad c_{dy} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

As mentioned in Section 2.5.2, it is helpful to add a regularization term when fitting the regression coefficients. While learning this estimator, the regularization coefficient was set to 1. The ExpertBoost algorithm was used to train both the vertical and horizontal estimators. Both estimators were trained to include 800 experts. The scale factor h from Equation 2.13, which was set to 384, was chosen manually using the training data.

3.4.1 The Benefit of Multi-scale Image Patches

The first comparison will show the benefit of producing estimates from larger image patches by using a multi-scale representation.

Larger image patches are helpful for estimating constraint values because the estimators are able to see more of the image. Unfortunately, the dimensionality of the patches rises quickly as larger areas of the image are considered. To work with larger patches of the image, while avoiding the "curse of dimensionality" [11] and its problems, we use a multi-scale representation of the image patches. The multi-scale patch is constructed using a three level Laplacian Pyramid [4] using a five-tap [1 4 6 4 1] filter, without down-sampling between levels. The multi-scale patch is then created by taking a 3×3 patch from each level of the pyramid. This multi-scale representation allows each patch to effectively represent a larger area of the image with a relatively few number of dimensions. For example, an 11×11 image patch, with 121 pixels, can be represented in a multi-scale representation with only 27 coefficients. While some

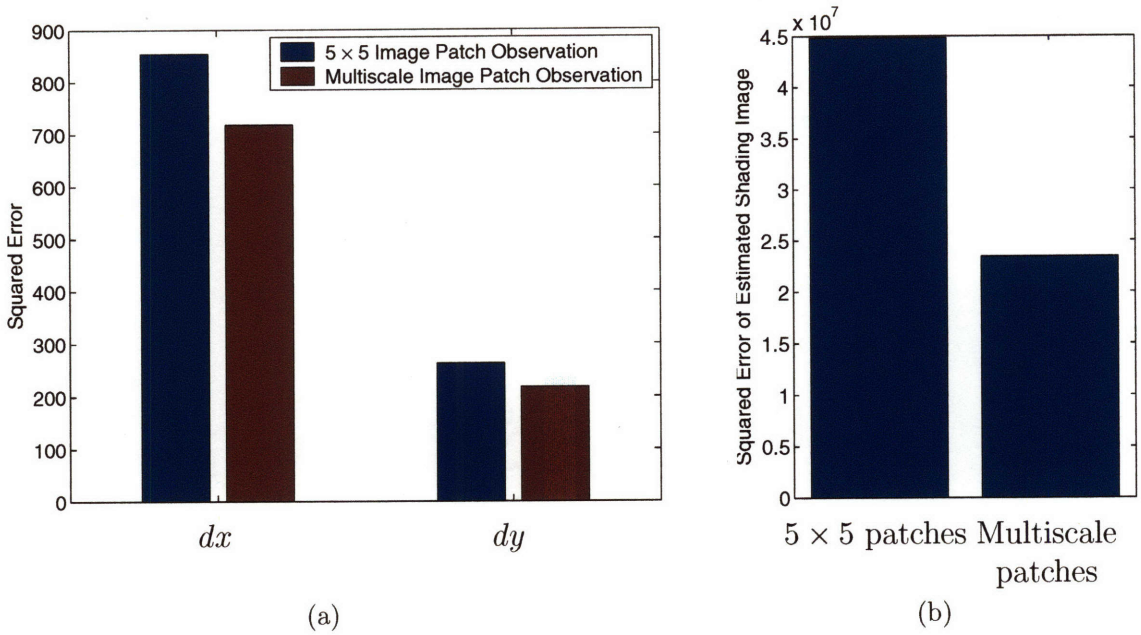


Figure 3-3: Using multi-scale image patches reduces the error both in the estimate of the derivative values and in the resulting estimated shading images. Figure (a) compares the error in the horizontal and vertical derivatives, denoted dx and dy. Figure (b) compares the error in the resulting estimates of the intrinsic component images.

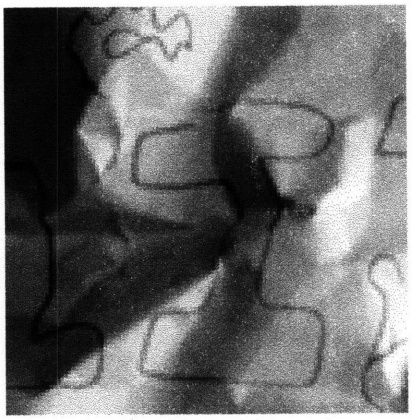
information is lost, it is assumed that the important information at the center of the patch is retained.

When estimating derivatives for shading estimation, the estimators performed better when the mean was subtracted from the 3×3 patch taken from the low-frequency residual of the Laplacian Pyramid.

Besides changing the input, the estimators are trained in the exact same fashion as described in the previous section.

Evaluating the Benefit of using Multi-scale Patches

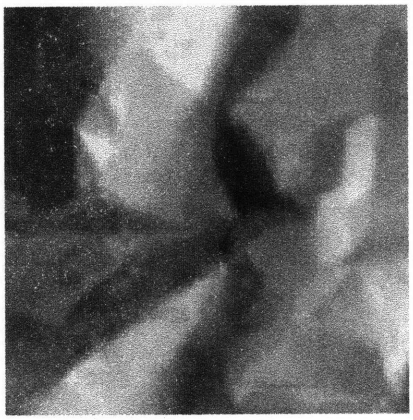
To begin, Figure 3-3 numerically compares the two input representations using three measurements: the squared error in the predictions of the horizontal derivatives, the squared error in the predictions of the vertical derivatives, and the squared-error of the estimated shading image reconstructed from the horizontal and vertical derivatives. As can be seen in Figure 3-3(a), using the multi-scale image patches improves the estimates of the derivative values. As Figure 3-3(b) shows, this has a dramatic effect on the error in the estimated shading images. Using the multi-scale image patches



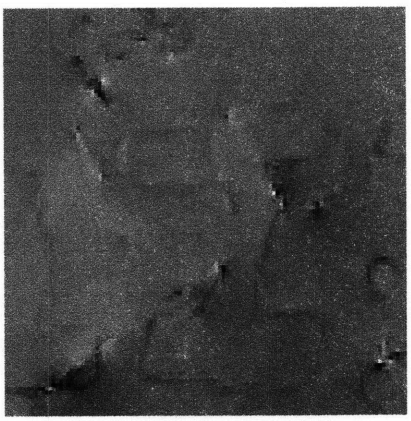
(a) Observed Image

(b) Ground-truth Shading

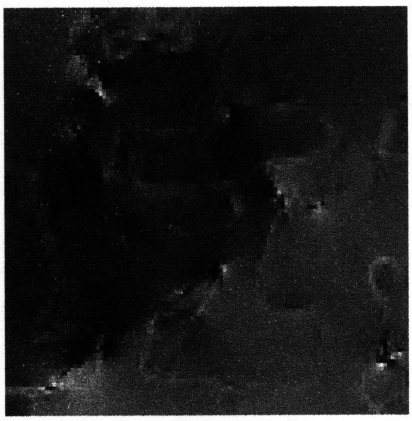
(c) Estimated Shading Image Using 5×5 Image Patches



(d) Estimated Shading Image Using Multi-scale Image Patches

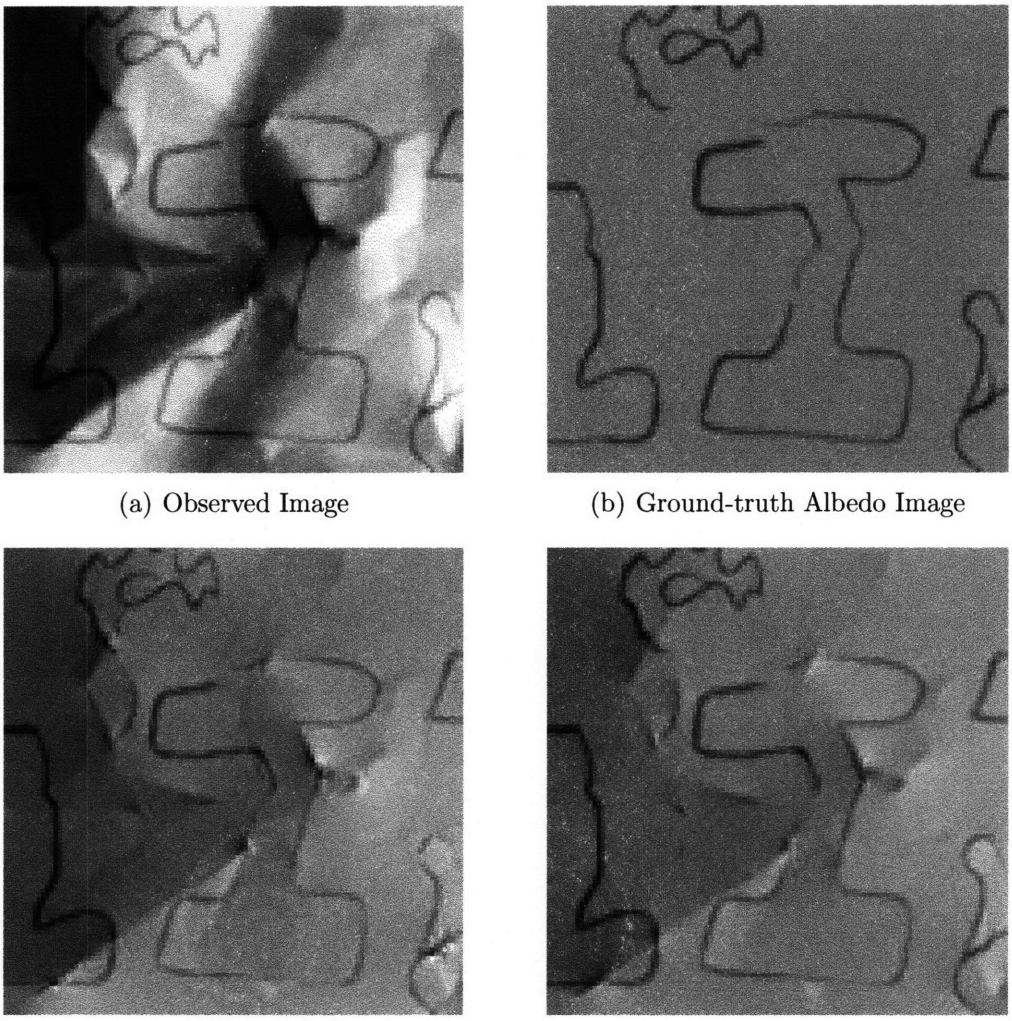


(e) Difference Between the Two Shading Estimates



(f) Absolute Value of (e)

Figure 3-4: These images demonstrate the qualitative improvement gained by using multiscale image patches. The coloring is much less noticeable in the results from using multiscale image patches, shown in (d). Examining the difference images in (e) and (f) shows that much of the difference comes from better handling of the crease in the lower-left portion of the image.



(a) Observed Image

(b) Ground-truth Albedo Image

(c) Estimated Albedo Image Using 5×5 Image Patches

(d) Estimated Albedo Image Using Multi-scale Image Patches

Figure 3-5: The albedo images recovered by subtracting the estimated shading image from the observed image. The albedo image recovered using multi-scale patches shows fewer shading artifacts.

reduces the error nearly 40%.

This difference can be observed qualitatively in Figure 3-4, which shows the shading images estimated from the observation in Figure 3-4(a). The estimate found using multi-scale image patches, shown in Figure 3-4(d), shows less of the coloring than the estimate found with 5×5 patches, shown in Figure 3-4(c). However, examining the absolute value of the difference between the two images, shown in Figure 3-4(f), shows that the major improvement from using multi-scale patches comes on the right half of the image. This difference is most likely caused by a better estimate of the shading around the crease in the lower-left portion of the image. As can be seen in Figure 3-5, which shows the albedo images recovered by subtracting the estimated shading images from the observation, less of the crease appears in the shading image recovered using the multi-scale patches. This has a large impact on the error in the rest of the image.

Figure 3-3(a) also shows another interesting characteristic of both systems. The error in the estimates of the vertical derivatives is almost one-third of the error in the estimates of horizontal derivatives. This is because the illumination in the training images generally comes from the right side of each image. This leads to strong vertical shading edges, which are ambiguous using only local information. This leads to errors like the crease which is mistakenly viewed as an albedo change in Figures 3-4 and 3-5.

3.4.2 Does Using Multiple Filters Aid Shading Estimation?

Any linear constraint can be considered in the pseudo-inverse step. A reasonable addition to the oriented derivative filters is a set of higher-order derivatives. To examine whether higher-order derivatives improve shading estimation, three second-order derivatives were added to the first-order derivatives used previously. The error over the training set only dipped slightly from 2.46×10^7 to 2.41×10^7.

3.5 Interpreting the Estimates

This thesis contains many examples of estimated shading and albedo images. Unless otherwise noted, the albedo images are produced by subtracting the estimated shading images from the observed images.

When examining the shading and albedo images, the shading image should appear as if the surface were completely made from the same material, with no painting or

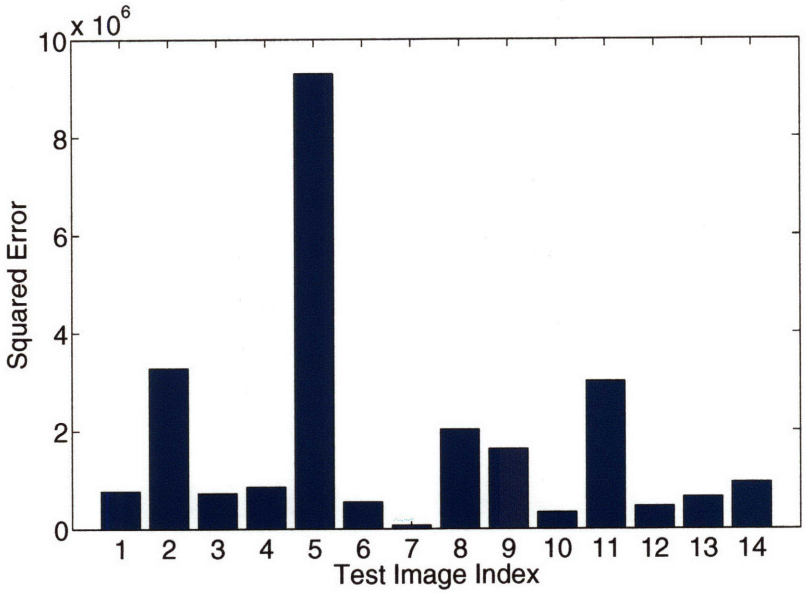


Figure 3-6: This chart shows the squared-error of the estimate of each image in the test set. Image # 5 has the largest error, due to the crease in the paper, which is ambiguous given only local information. This image will be used throughout this work as the example for a qualitative comparison.

other changes in reflectance. The albedo image should appear as a flat image with just coloring.

It should be noted that the errors reported in the test set are dominated by several images. Figure 3-6 shows the squared-error of each test image, when estimated using the multi-scale estimators from Section 3.4. In particular, image #5 accounts for a significant portion of the error. Throughout this thesis, this image will serve as the primary example for qualitative evaluation of the behavior of the algorithms. The large error incurred in this image is likely due to the crease in the lower-left portion of the image. Locally, this crease is ambiguous and mistakes in this crease propagate through the rest of the image.

3.5.1 Estimating Albedo Images

In all the results shown in this chapter, the estimation task is to estimate the shading image from the observation. The albedo image can then be recovered by subtracting the estimated shading from the observed image. If the image model includes more than two intrinsic components, then separate estimators for each characteristic would need to be trained.

(a) (b) (c)

Figure 3-7: To estimate shading images using classification, similar to [69], a label must be assigned to every horizontal and vertical derivative in the image. Pixels marked in white in Figure (b) are labelled as shading. The red x's in Figure (c) show a portion of the horizontal derivatives labelled as albedo changes.

3.6 Evaluating Different Criteria for Fitting the Derivatives

As Section 2.5.3 pointed out, the ExpertBoost algorithm can minimize any loss function. While the squared-error loss leads to a convenient algorithm, it is possible that minimizing a different criterion may lead to visually better estimated intrinsic component images. This section will compare minimizing the squared-error with two alternate criteria. First, it will be compared against minimizing the exponential loss, which essentially treats the problem as a classification problem. This will allow the results in this thesis to be compared against classification-based systems such as my previous work [69].

Next, in Section 3.6.2, minimizing the squared-error in the predictions will be compared against minimizing a robust-error norm.

3.6.1 Minimizing Squared-Error versus the Exponential Loss

Rather than minimizing the squared-error in the estimated derivative values, it is possible to instead treat the problem as a classification problem instead. This is the approach taken in Retinex and my previous work [69]. As described in Section 1.2.2, this can be thought of as classifying every derivative as either being caused by shading or an albedo change, assuming that the set of linear constraints are derivatives.

Before learning the classifiers, the first step is to label every image derivative in

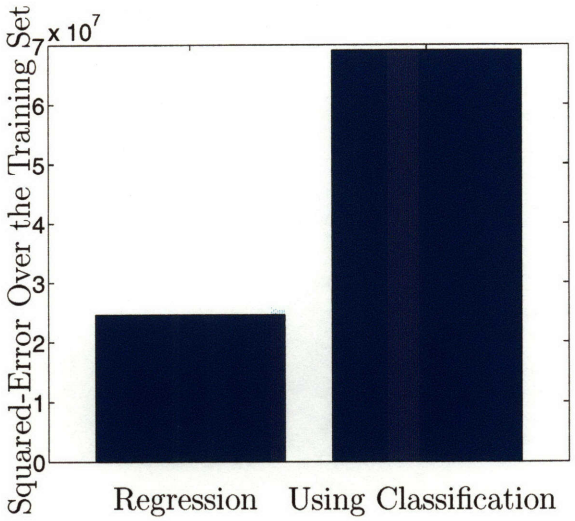


Figure 3-8: This chart shows the error on the test set when classification is used instead of regression. Using classification significantly increases the error.

the training set as either shading or a reflectance change. As the task is to predict the shading image, the training set consists of pairs of observations and the corresponding shading image. To assign each derivative in the training set a label, the albedo image is calculated for each image pair by subtracting the shading image from the observed image. The label of each derivative is based on the absolute value of the derivative in the albedo image. If this derivative has a magnitude greater than one, the derivative is classified as an albedo change. Dark points in Figure 3-7(b) show derivatives classified as albedo changes in the training image shown in 3-7(a). Figure 3-7(c) shows these points marked with red x's on the observed image.

Once every derivative in the training set has been assigned a label, the classifiers can be trained as described in Section 2.5.4. This is similar to training a classifier with the GentleBoost algorithm [31].

Results

To evaluate classification, classifiers were trained on the same training set as in previous experiments. Classifiers were trained to classify both the horizontal and vertical derivatives. Given a new test image, the shading derivatives are estimated by returning the value of the derivative in the image if the classifier labels the derivative as being caused by shading. Otherwise, the shading derivative is assumed to be zero.

Figure 3-9(g) shows one of the largest disadvantages of classification. The crease

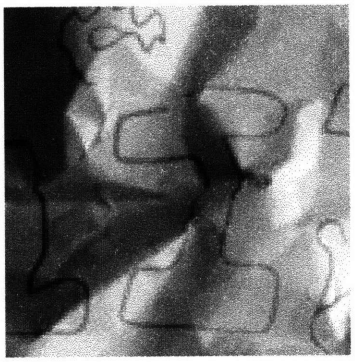
in the lower-left portion of Figure 3-9(a) is ambiguous given only local information. The estimator trained using the squared-error criterion copes with this by placing a portion of the edge in the shading image and a portion of the edge in the albedo image, as seen in Figures 3-9(c) and 3-9(f). On the other hand, a classifier cannot make this split – each derivative must be in one image or the other. As can be seen in Figures 3-9(d) and 3-9(g), this leads to the whole crease being placed in the albedo image. This significantly affects the error in the predicted image. As Figure 3-8 shows, when using classification, the squared-error over the training set grows from 2.46×10^7 to 6.9×10^7.

Perceptual Effects of Continuous Estimation Instead of Classification

The previous section has shown that estimating derivative values instead of classifying generally leads to a lower overall squared-error in the estimate of the shading image. While the error is lower, the images estimated using continous regression have a distinctive behavior. Figure 3-10 shows the intrinsic component images estimated from Adelson's Checker Shadow image. Examining the shading image, shown in Figure 3-10(b), shows that remnants of the albedo image can still be seen in the shading image. This is a natural result of using the squared-error as the criterion. If the estimator is uncertain about whether a derivative belongs in the shading image or the albedo image, under the squared-error criterion, the best choice is to split the derivative and place some in the shading image and some in the albedo image.

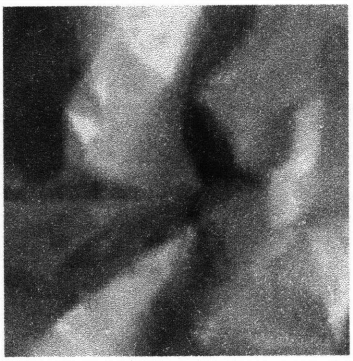
To some viewers, this may appear to be a bad result. If, instead, the goal is actually to estimate a shading image without any albedo changes, these estimates can be used to classify the derivatives. Figure 3-11 shows the shading and albedo images produced using a simple classification heuristic. Using this heuristic, if the magnitude of the derivative in the estimated shading image is less then 80% of magnitude of the corresponding derivative in the observed image, then the derivative is classified as an albedo derivative. This leads to an image that shows much less albedo in the shading image.

It is important to note that when this strategy is applied on the test images, the squared-error increases. While classifying derivatives appears to give better results, the results are actually worse numerically, using a squared-error metric.

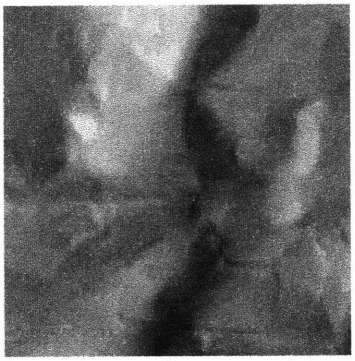


(a) Observed Image

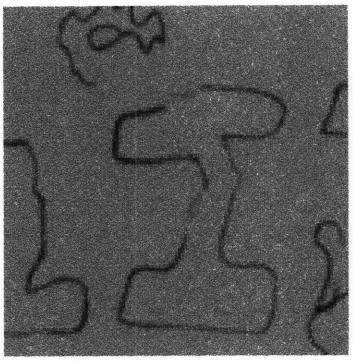
(b) Ground Truth Shading Image



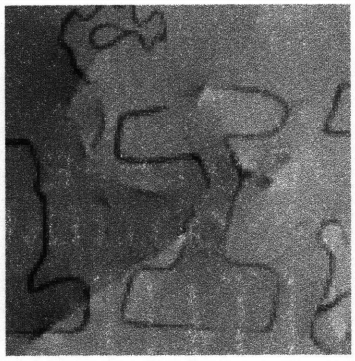
(c) Shading Image Estimated with Regression



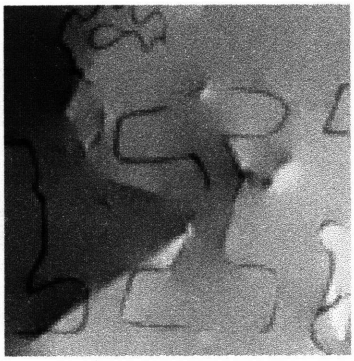
(d) Shading Image Estimated with Classification



(e) Ground Truth Albedo Image

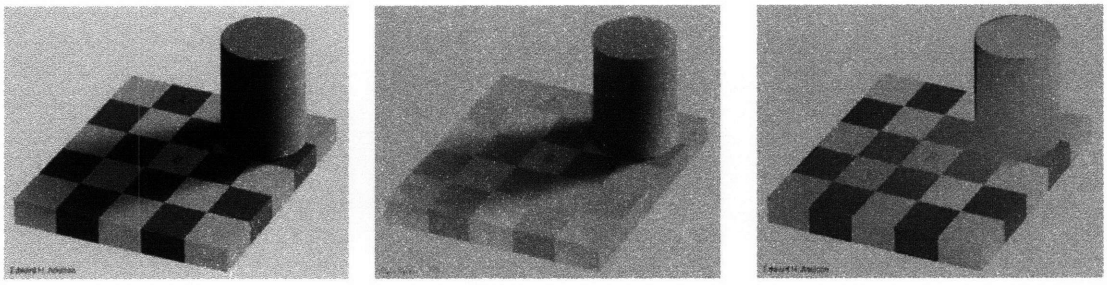


(f) Albedo Image Estimated with Regression



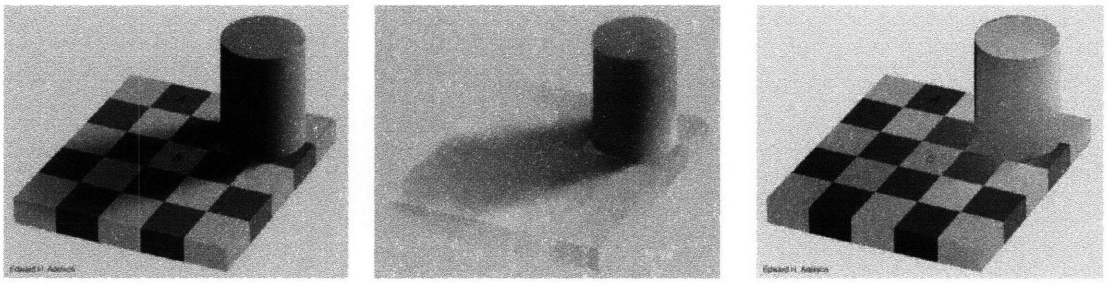
(g) Albedo Image Estimated with Classification

Figure 3-9: The shading and albedo images estimated using classification and regression. Using classification allows much more shading to leak through into the albedo image.



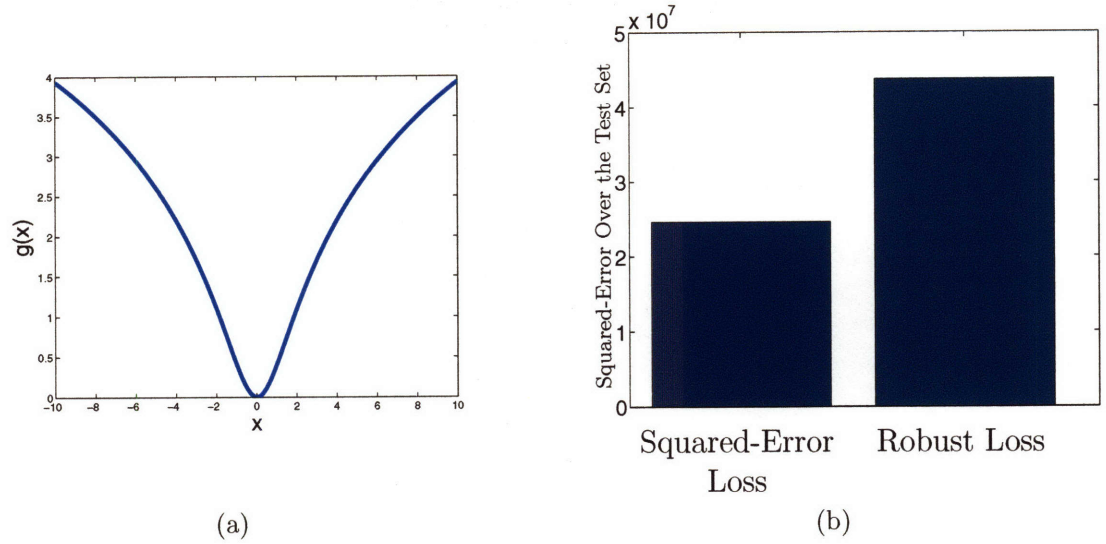
(a) Observation (b) Estimated Shading Image (c) Estimated Albedo Image

Figure 3-10: These images show the ghosting that often results from using continuous estimates of the shading derivatives, rather than estimates based on classifying derivatives.



(a) Observation (b) Estimated Shading Image (c) Estimated Albedo Image

Figure 3-11: The ghosting seen in Figure 3-10 is natural when the estimators are trained to minimize the squared-error in the derivative estimates. This leads to a lower overall squared error in the estimated shading image. However, if the real goal is to eliminate albedo changes from the shading image, at the expense of increasing the pixel-wise error, this can be accomplished by using the original prediction to classify each derivative. These figures show the results of using the estimates from Figure 3-10 to classify each derivative.



(a) (b)

Figure 3-12: (a) The Lorentzian robust loss function used in this section. (b) This chart compares the error in the test set when training the derivative estimators to minimize either a robust loss or the squared-error loss. Training using the squared-error loss leads to poorer estimates of the shading images.

3.6.2 Minimizing Squared-Error versus a Robust Loss

The ExpertBoost algorithm can also minimize loss functions other than the squared-error. An interesting robust alternative to the squared-error is the Lorentzian robust penalty function[14]:

$$L(x, y) = \log(1 + \frac{1}{2}(x - y)^2) \tag{3.2}$$

This function, plotted in Figure 3-12, is considered to be robust because the derivative of the error decreases as the error increases. This makes the Lorentzian cost function robust to the occasional outlier. In comparison, in the squared-error cost function, the derivative of the error increases as the error increases. This makes the squared-error function somewhat sensitive to outliers.

Unfortunately, it is not possible to solve the minimization in closed form, so it is minimized with the non-linear conjugate gradient routine from the NETLAB package [49]. In practice, this is much slower than solving the least-squares problem that results from using the squared-error cost function.

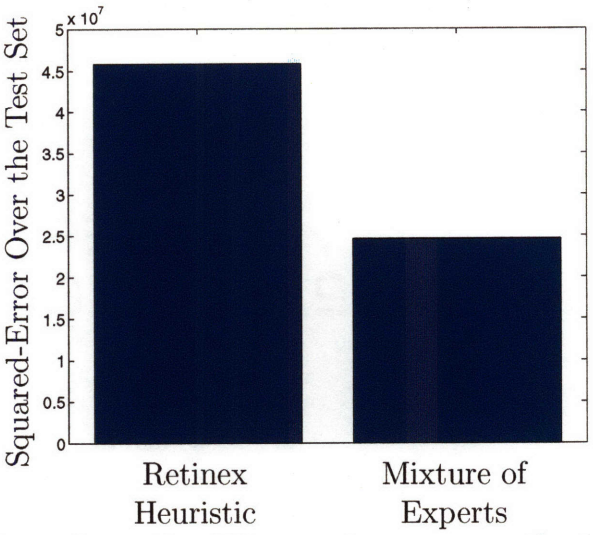


Figure 3-13: This chart shows the difference in error over the test set when using an estimator based on the Retinex heuristic versus the Mixture of Experts estimators. Using the Mixture of Experts estimators significantly reduces the error.

Results

As shown in Figure 3-12(b), training the estimators under a robust loss function raised the total squared-error in the test set from from 2.46×10^7 to 4.38×10^7. In addition to producing lower-quality image estimates, the non-linear optimization required for fitting the regression coefficients significantly increases the time necessary to train the estimators.

3.7 Learning versus the Retinex Heuristic

The previous sections have established that, out of the options evaluated, training estimators with the squared-error loss function, using multi-scale observations, is the best combination for estimating shading images when using the squared-error as the evaluation criterion. The next natural question is: how does this system compare with other approaches to this task?

The baseline system for comparison with the Mixture of Experts estimators is an estimator based on the assumption underlying the Retinex algorithm, discussed in Section 1.2.2. The assumption is that derivatives with small magnitudes are caused by shading and derivatives with large magnitudes are caused by changes in the albedo of the scene. Given the value of a derivative in the observed image, δo, the estimate of the shading derivative, $\hat{c}_{dx}$ is

$$\hat{c}_{dx} = \frac{\delta o}{1 + e^{a|\delta o| + b}} \tag{3.3}$$

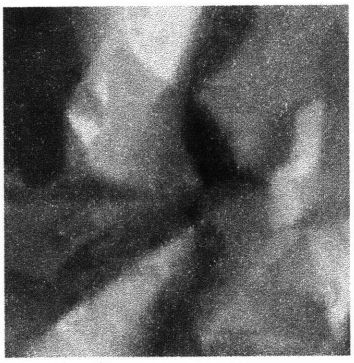
The sigmoid function in Equation 3.3 acts to set derivatives with large magnitudes to 0. The parameters a and b are found by minimizing the squared-error of the derivatives over the training set.

The results produced by the Mixture of Experts estimators were superior both visually and in terms of squared-error. Figure 3-14 shows the shading and albedo images returned by the ExpertBoost estimators and the Retinex estimators. The scribbling is still noticeable in the images from the Retinex estimators. In addition, much more of the shading image has leaked into the albedo image found using the Retinex estimators than in the image found with the ExpertBoost estimators. This is corroborated by the differences in error. As shown in Figure 3-13, the error in the test images produced using the Retinex estimators is over twice the error in the images produced using the ExpertBoost estimators.

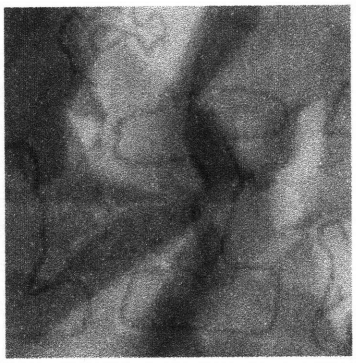
This difference can also be seen qualitatively in Figure 3-14(d), which shows the shading image recovered by using the Retinex estimators. The Retinex estimators have done a very poor job of eliminating the coloring from the shading image.

(a) Observed Image

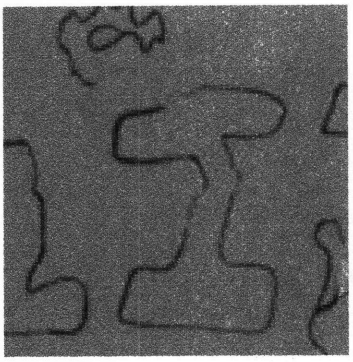
(b) Ground Truth Shading Image



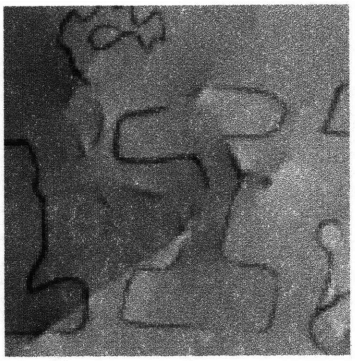
(c) Shading Image Estimated with Mixture of Experts Estimators



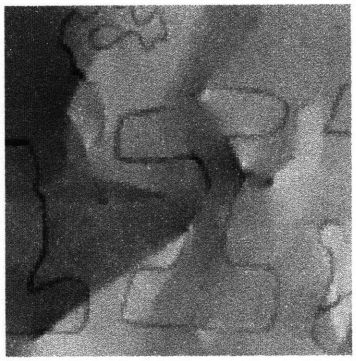
(d) Shading Image Estimated with Retinex-style Estimators



(e) Ground Truth Albedo Image



(f) Albedo Image Estimated with Mixture of Experts Estimators



(g) Albedo Image Estimated with Retinex-style Estimators

Figure 3-14: The shading and albedo images estimated using Retinex-style estimators and Mixture of Experts estimators. The Retinex-style estimators are unable to remove the coloring from the page. In addition, the albedo image shows that many more shading artifacts have leaked through into the albedo image.

3.8 Learning to Estimate Albedo Images versus Learning to Estimate Shading Images

Until now, the learning problem has been posed as the problem of estimating a shading image from an observation. However, since the observation is the sum of a shading image and an albedo image, it is equally valid to pose the problem as estimating the albedo image from the observation. If the estimators are stable, the results from learning to estimate shading images should be similar to the results from learning to estimate albedo images.

To verify the stability of the system, horizontal and vertical estimators were trained using the same method as described in Section 3.4. The only difference from Section 3.4 was that the task was to predict the derivatives of the albedo images instead.

The stability of the system can be verified by measuring the error in the shading images produced by subtracting the estimated albedo images from the observed images. As shown in the Figure 3-15, these new shading images have almost the same error as those obtained by estimating the shading images directly.

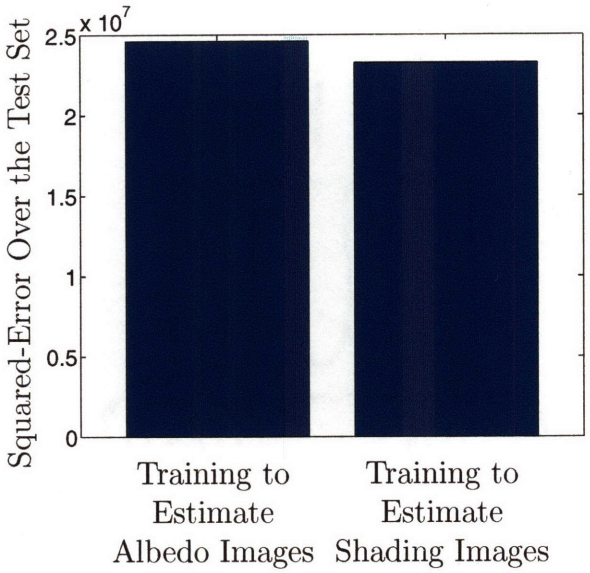


Figure 3-15: This chart compares the error in the estimated images when the system is trained to estimate shading images against the error when the system is trained to estimate albedo images. The two errors are roughly equal, demonstrating the stability of the system.

3.9 Effect of Known Lighting Orientation

To understand the importance of knowing the light direction, a new training set was produced that consisted of three copies of each of the original training images. Each of the images in the new training set was randomly rotated. Estimators of the horizontal and vertical derivatives were then trained in the same manner described in Section 3.4. The estimators were trained on a set of image patches that were similar in number to those training patches used to train the previous estimators.

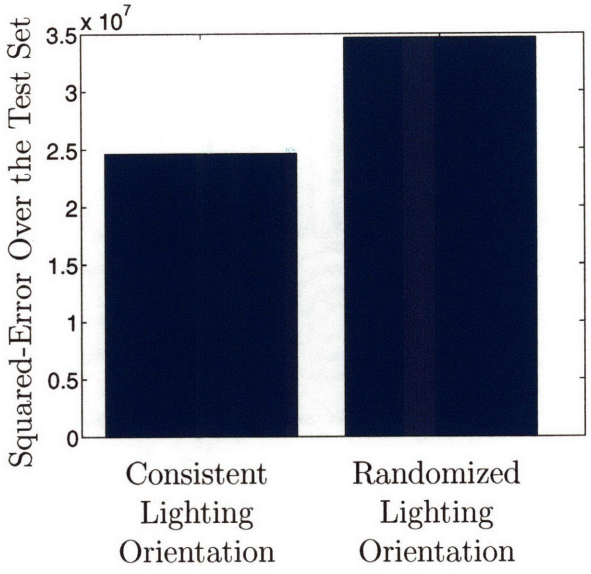
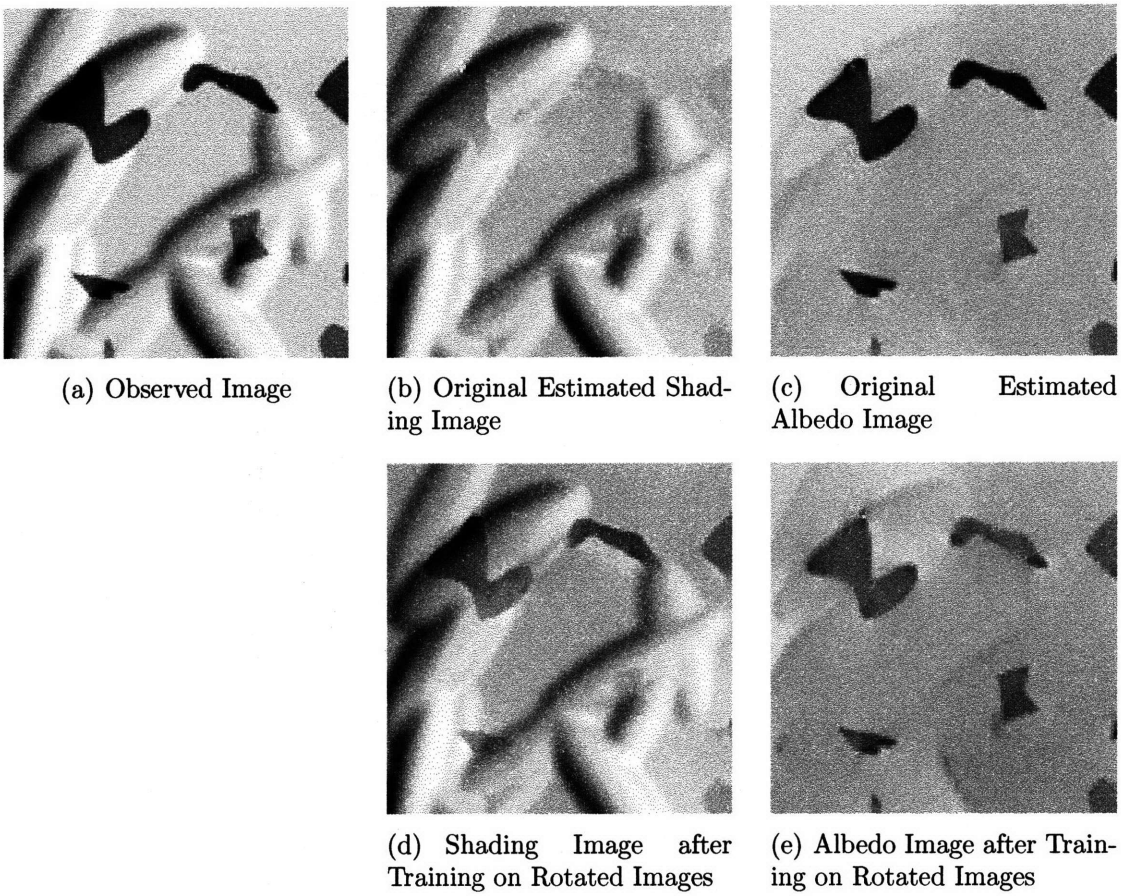


Figure 3-16: This chart compares the error in the estimated images when the orientation of the illumination is roughly known against the error when the orientation has been randomized.

To evaluate the change in performance, the system was evaluated on the same test images that were used previously. After training on the rotated images, the errors in the horizontal and vertical derivative estimates were similar, showing that cues based on the orientation of the illumination were removed by randomizing the training data.

As shown in Figure 3-16, the error in the test set increased from 2.46×10^7 to 3.46×10^7. This increase in error is expected because knowing the orientation of the illumination provides valuable information about possible shading changes. Interestingly, much of this increase is concentrated in the synthetic images in the training set. Figure 3-17 shows an example of one of the test images. As Figure 3-17 demonstrates, after training the estimators on the randomly rotated images the system has a more difficult time removing the synthetic albedo patterns from the shading image.



(a) Observed Image

(b) Original Estimated Shading Image

(c) Original Estimated Albedo Image

(d) Shading Image after Training on Rotated Images

(e) Albedo Image after Training on Rotated Images

Figure 3-17: These images show the effect of removing information about the orientation of the light source. In the synthetic test images, such as the image shown here, randomizing the light source's orientation reduces the system's ability to remove the albedo from the shading image.

The estimates of the shading and albedo images from the crumpled paper images are less affected by randomizing the illumination orientation. The mean absolute error of the derivative estimates rose slightly, but overall the estimates of the shading and albedo images changed little. This indicates that the shading on the paper surfaces is sufficiently different from the albedo patterns in the training and test sets, so as not to depend on knowing the orientation of the scene's illumination.

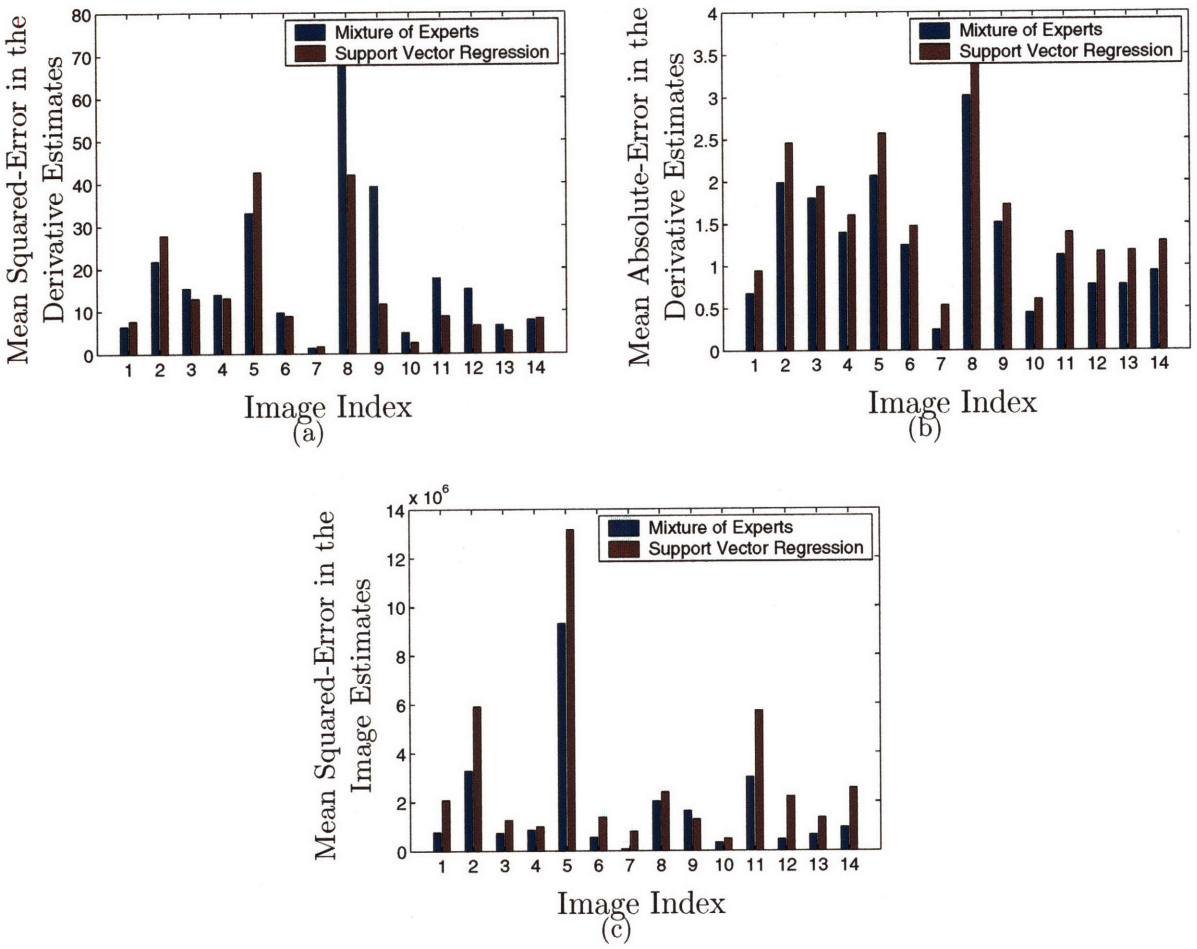


Figure 3-18: These charts compare the error between the derivatives and shading images estimated using either Support Vector Regression or the Mixture of Experts estimators. The estimators produced derivative estimates with a comparable mean squared-error, as shown in (a). However, as (b) shows, the mean absolute-error is consistently lower in the estimates produced by the Mixture of Experts estimators. This likely explains why the error in the reconstructed shading images is consistently lower when the derivative estimates from the Mixture of Experts estimators are used. The fact that the SVR estimators have a higher mean absolute-error indicates that they make more small mistakes. When the image is reconstructed, these mistakes can accumulate to produce large errors.

3.10 Boosted Regression versus Support Vector Regression

Section 2.7 showed that on a simple problem, Support Vector Regression estimators and Mixture of Experts estimators achieve similar error rates. This section considers the performance of the two different types of estimators when estimating the derivatives of the shading image. On this more complicated problem, the Mixture of Experts estimators produce much better estimates of the shading images.

To compare the Mixture of Experts estimators against estimators trained using Support Vector Regression, SVR estimators were trained on the same training set used in the previous sections. The SVR estimators were trained using the multi-scale patches as observations and the libSVM package to find the support vectors and coefficients [16]. The ν-SVR option [58] was used because it allows for easier control over the number of support vectors.

Using a held-out portion of the training set, the kernel parameter γ was set manually to optimize the estimators' generalization performance. I found that $\gamma = 2^{-9}$ worked well for estimating shading. The SVR estimator used 7421 support vectors to estimate the horizontal derivatives and 8055 support vectors to estimate the vertical derivatives.

As Figure 3-18(a) shows, the derivatives estimated using SVR often have a smaller mean squared-error than the derivative estimates from the Mixture of Experts estimators. However, as Figure 3-18(c) shows, the error in the estimated shading images is consistently higher when the shading image is reconstructed from the derivatives estimated with Support Vector Regression. Using these derivative estimates increases the total squared-error over the test set from 2.46×10^7 to 4.16×10^7.

This can be understood by examining the mean absolute error of the derivative estimates, shown in Figure 3-18(b). The mean absolute error is consistently lower in the estimates from the Mixture of Experts estimators. This difference in absolute error is likely due to the method that Support Vector Regression uses to choose a relatively small number of support vectors from a large training set – a property known as "sparsity". In Support Vector Regression, the estimator is fit by minimizing an ϵ-insensitive loss function:

$$L(g(o;\theta),t) = \begin{cases} 0 & \text{if } |g(o;\theta) - t| < \epsilon \\ |g(o;\theta) - t| - \epsilon & \text{otherwise} \end{cases} \tag{3.4}$$

where t is a target value and $g(o;\theta)$ is an estimator using o as the observation and parameters, θ, to predict t.

Minimizing this loss causes small errors to be completely ignored. For the particular estimators in this experiment, ϵ was 4.33 for the horizontal estimator and 3.34 for the vertical estimator. This means that when fitting the estimator of the horizontal derivatives, any error less than 4.33 will be ignored. This makes the estimator prone to making many small errors, as can be seen in the difference in the mean absolute errors. When the image is reconstructed from these derivatives, these small errors can compound into large errors in the image. It should be noted that the SVR training parameters have been tuned to produce an estimator that is relatively sparse. Given enough support vectors, the SVR estimators would produce similar results to the Mixture of Experts estimators.

Surprisingly, the difference in the form of the estimators does not play a significant role in the difference in the quality of the estimates. At the beginning of this work, my intuition was that the normalizing term in the Mixture of Experts estimator would significantly enhance its generalization performance. Ideally, the normalizing term would allow the linear regression to better extend to areas far from the prototype patches.

To examine the effect of removing the normalizing constant, the ExpertBoost algorithm was modified to fit an estimator of the form:

$$g(o) = \sum_{i=1}^{N_{SV}} w_i \exp\left(-\gamma ||o - p_i||^2\right) \tag{3.5}$$

The results from using this estimator were comparable to the results from using a Mixture of Experts estimator. This indicates that the primary difference in the performance of the estimators lies in the training procedures – as discussed above.

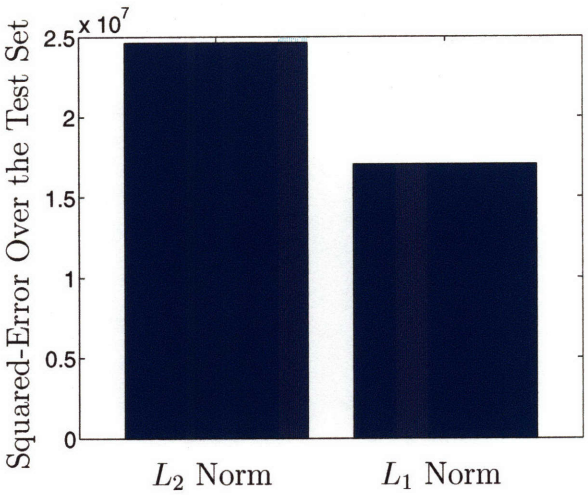


Figure 3-19: This chart compares the error when reconstructing the shading images from the derivative estimates using an L_2 norm versus an L_1 norm. Using an L_1 norm reduces the squared error in the estimated shading images.

3.11 Recovering Intrinsic Component Images Under the $L1$ Norm

Section 3.2 describes how to reconstruct the estimated intrinsic component image from the estimated linear constraint values using a Moore-Penrose pseudo-inverse. This is equivalent to finding the vector $\mathcal{X}$ that minimizes the L_2 norm of this error vector

$$\mathcal{X} = \arg\min_{\mathcal{X}} ||C\mathcal{X} - \hat{c}||^2 \tag{3.6}$$

Changing the norm in Equation 3.6 will change the value of $\mathcal{X}$ returned. The L_2 norm can be conveniently replaced with the L_1 norm, which is effectively the sum of the absolute values of the error vector. For the L_1 norm, $\mathcal{X}$ can be found by solving this linear program [12, 2]:

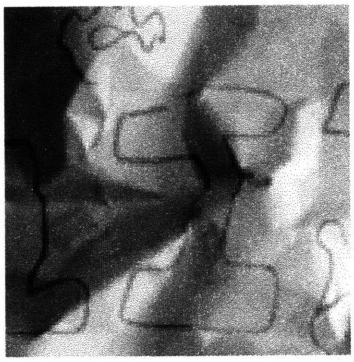
$$\begin{aligned} \text{minimize}_{\mathcal{X},z} \quad & \mathbf{1}^T z \\ \text{s.t.} \quad & C\mathcal{X} - z\mathbf{1} \leq \hat{c}, \\ & C\mathcal{X} + z\mathbf{1} \geq \hat{c} \end{aligned} \tag{3.7}$$

where $\mathbf{1}$ is a vector of all ones. Note that this solution involves introducing a new vector of variables, z. More efficient formulations of this LP can be found in [2]

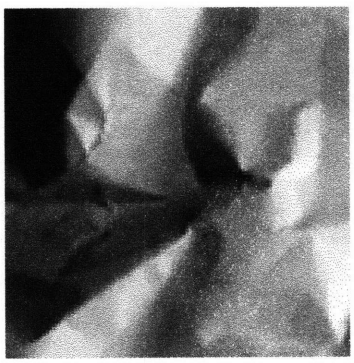
To evaluate the effect of reconstructing the estimated images using an L_1 norm instead of an L_2 norm, the shading images were estimated using the derivative estimators from Section 3.4. I used the MOSEK [1] optimization package to find $\mathcal{X}$.

As shown in Figure 3-19, reconstructing the image estimates under the L_1 norm reduces the error in the image estimates from 2.4×10^7 to 1.7×10^7. The qualitative difference between the two reconstructions can be seen in Figure 3-20. The albedo image recovered using the L_1 norm, shown in Figure 3-20(f), is better at capturing the flatness of the true albedo image than the albedo image recovered using the L_2 norm.

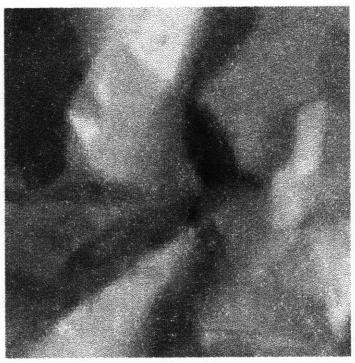
It is important to note that, in some sense, two different error criteria are being used in this evaluation. The first criterion measures how well the estimated image matches the derivative estimates, after it has been filtered with a derivative filter. The second error criterion measures how well the image reconstructed using the first error criterion matches the ground-truth image. This section shows that using the absolute error, rather than the squared-error, leads to a better shading image, according to the squared-error between it and the ground-truth image. This is likely because the absolute error is more robust to outliers, which leads to fewer artifacts in the estimated shading image.



(a) Observed Image

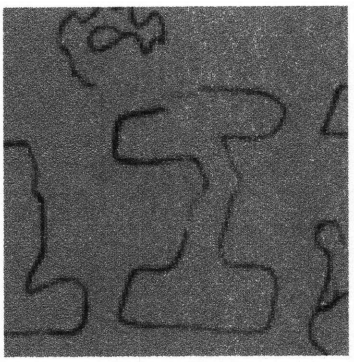


(b) Ground-Truth Shading Image

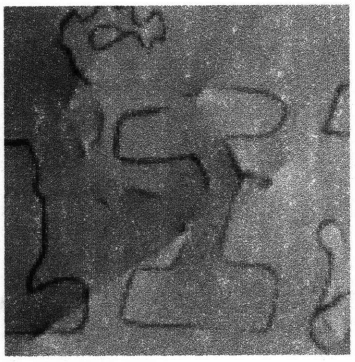


(c) Shading Image Reconstructed using an L_1 Norm

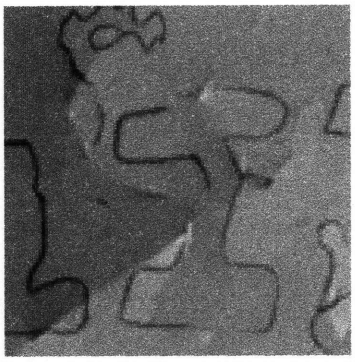
(d) Shading Image Reconstructed using an L_2 Norm



(e) Ground-Truth Albedo Image

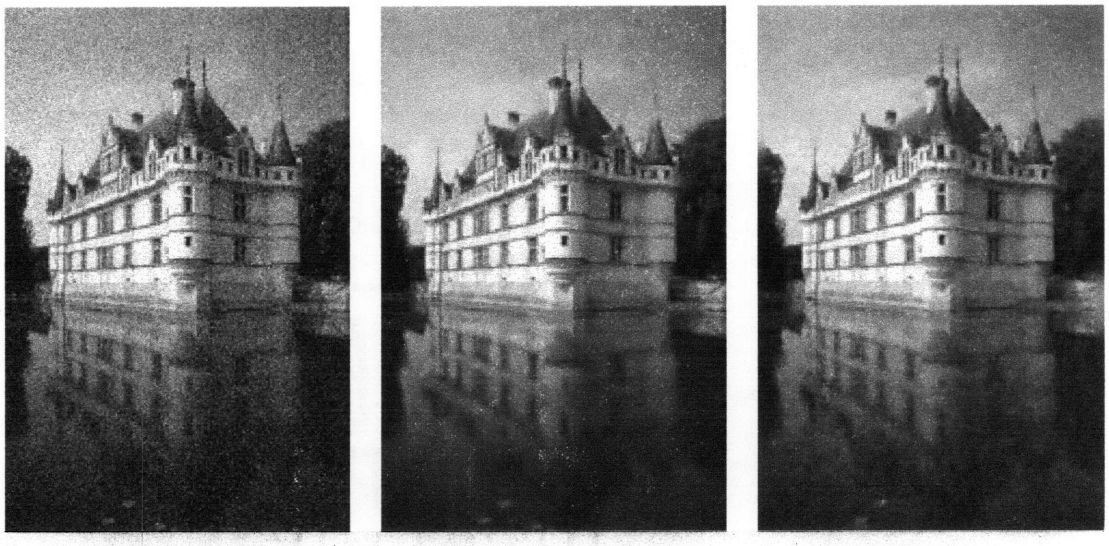


(f) Albedo Image Reconstructed using an L_1 Norm



(g) Albedo Image Reconstructed using an L_2 Norm

Figure 3-20: The shading and albedo images estimated using the L_1 and L_2 norms to reconstruct the estimated shading image from the estimated derivative values. Using an L_1 norm leads to a slightly flatter appearance in the albedo image, indicating that less shading image components appear in the albedo image.



(a) Noisy Input (b) Roth and Black (c) Intrinsic Components

Figure 3-21: An example of using our method for denoising. The image in (a) has been corrupted with white Gaussian noise ($\sigma = 10$). The results produced by our method are competitive with the recent Field of Experts Model of Roth and Black [56].

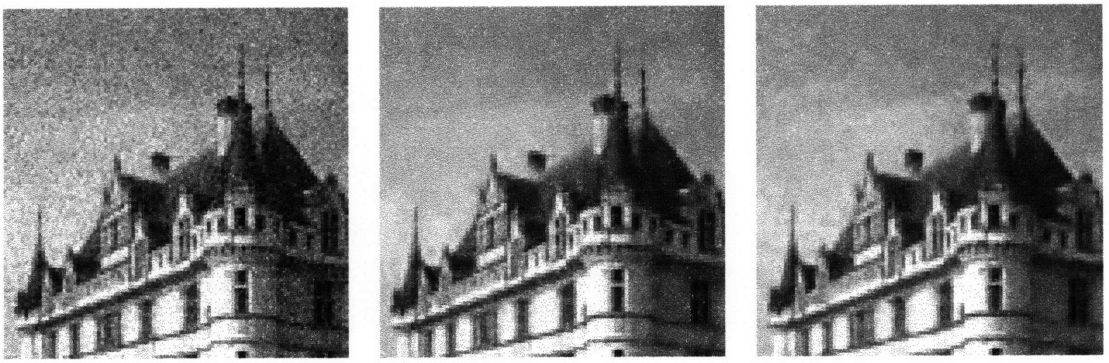
3.12 Application to Denoising

To demonstrate the ability of our method to generalize to other types of intrinsic components, this section considers the problem of estimating a scene image from an observation corrupted with Gaussian white noise. In this application, the intrinsic components are the clean image and the noise. In the initial tests, three filters served as the basis of the system: two oriented first derivative filters and a Gaussian filter with standard deviation 1 that was truncated to a 3×3 filter.

For denoising, I obtained the best results by using six filters as the basis for the system: two first derivative filters, three second derivative filters, and a Gaussian filter with standard deviation 1 that was truncated to a 3×3 filter.

The estimators were trained using a set of 80 images taken from the Berkeley Segmention Database. The images were scaled to one-half their original size, cropped to size 128×128, and added to a Gaussian white noise image with a standard deviation of 10. The ExpertBoost algorithm chose 800 prototype patches for each estimator.

For evaluation, I used a set of 40 test images from the Berkeley database. Each of these images was processed in the same manner described above. For comparison,



(a) Noisy Input (b) Roth and Black (c) Intrinsic Components

Figure 3-22: Enlarged portions of the images from Figure 3-21. The primary difference between the two methods is that the Field of Experts model removes more of the noise in the lower spatial-frequency bands.

Algorithm	PSNR	PSNR of High-Pass Filtered Image
Field of Experts	32.46	34.58
Intrinsic Components	32.21	34.51

Table 3.1: The PSNR in dB of denoised images produced by our method and the Field of Experts algorithm. The second column compares the PSNR of high-pass filtered versions of the results.

I used the Field of Experts code provided by Roth et al. [56], which has achieved results close to the those produced by Portilla et al [52], which has achieved the best-published results for denoising. As is standard in the denoising literature, the results are presented in PSNR. If the error variance is denoted as σ_e, then the PSNR for these results can be calculated as $20 \log_{10}(255/\sigma_e)$.

As shown in Table 3.1, the Mixture of Experts estimators are competitive with the Field of Experts, although the PSNR of the results from the Field of Experts algorithm is about 0.25 dB higher. The primary difference between the results of the two algorithms appears to be in the low spatial-frequency bands of the images. The second column of Table 3.1 shows the PSNR of images created by high-pass filtering the results from the two algorithms. For these images, the PSNR is almost identical. As shown in Figure 3-21, the results are also visually similar.

To see if performance improved by adding more constraints, three second derivative filters were added to the set of constraints. The PSNR only increased slightly to 32.24.

3.13 Conclusion

This chapter has considered many possible configurations of the estimators for predicting the linear constraint values. Given all of these results, the natural question is: "What is the best set-up for estimating the constraint values?" Given the results in this chapter, the best estimators:

1. Are Mixture of Experts estimators or additive estimators (see Section 3.10)
2. Use a multi-scale input representation (see Section 3.4)
3. Are trained to minimize the squared-error in the predictions (see Section 3.6)

In addition, as Section 3.11 demonstrates, it is best to reconstruct the derivatives using the L_1 error norm instead of an L_2 error norm.

Chapter 4

Optimizing Estimated Images

The previous chapters have described how to estimate intrinsic component images by first estimating local linear constraints, then finding the pseudo-inverse of the linear system defined by these constraints. Chapter 3 also demonstrated how this strategy works well when applied to the problems of estimating shading images and denoising images.

While this strategy performs well, it suffers from two major limitations, described in more detail in Section 4.1:

1. The system can be viewed as an "open-loop" system. Estimators are trained to predict local linear constraints with the hope that good estimates of the constraints will lead to good estimated intrinsic component images. However, these estimators are trained to minimize the squared error in the prediction of the *constraint values*, not the *resulting image values*.

2. Simply finding the pseudo-inverse of the constraints does not take into account that some constraints may be more reliable or useful than others. In addition, some constraints may be better than others for specific types of image patches.

This chapter focuses on overcoming these two limitations. Section 4.2 describes how to implement a closed-loop system by optimizing the linear regression coefficients, using the error in the estimated images as the criterion. Section 4.3 then addresses the second limitation by showing how to learn a function that assigns a weight to each linear constraint in the system.

4.1 Limitations of the System from Chapter 3

As described earlier, the approach to recovering intrinsic component images described in Chapter 3 suffers from two major limitations. In this section, I will provide further explanation and motivation for each.

4.1.1 Open-Loop Versus Closed Loop Approaches

The issue of an open loop versus closed-loop system can be understood by considering the criterion for finding the parameters of the system. As a reminder, the system described in Chapter 3 recovers an intrinsic component image by first estimating the values of linear constraints, which were chosen to be derivatives for the experiments.

The overall goal of this work is to produce a system that recovers the intrinsic component images as accurately as possible. However, the parameters of the estimators, which include the prototype patches and linear regression coefficients, are chosen to minimize the error in the estimates of the constraint values. The hope is then that good constraint estimates lead to good estimates of the intrinsic component image. In the rest of this chapter, I will refer to this system as an "open-loop" system because the estimator parameters are set without regard to the quality of the image recovered from those estimates.

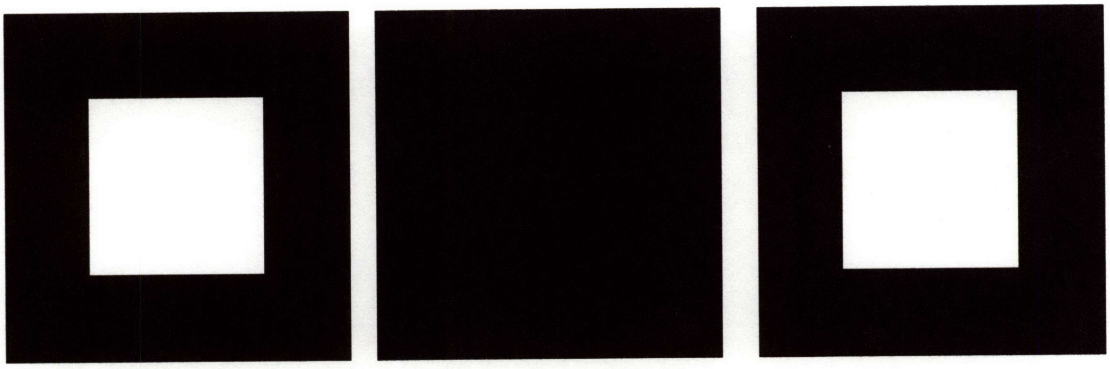
Ideally, the system should be "closed-loop", where the parameters of the constraint estimators are chosen to optimize the image recovered from the estimates of the constraints.

4.1.2 Taking Advantage of the Reliability of the Estimates

A second weakness of the system described in Chapter 3 lies in the pseudo-inverse step. As this step is implemented in Chapter 3, the estimates are assumed to all be equally valid. However, this is often not the case.

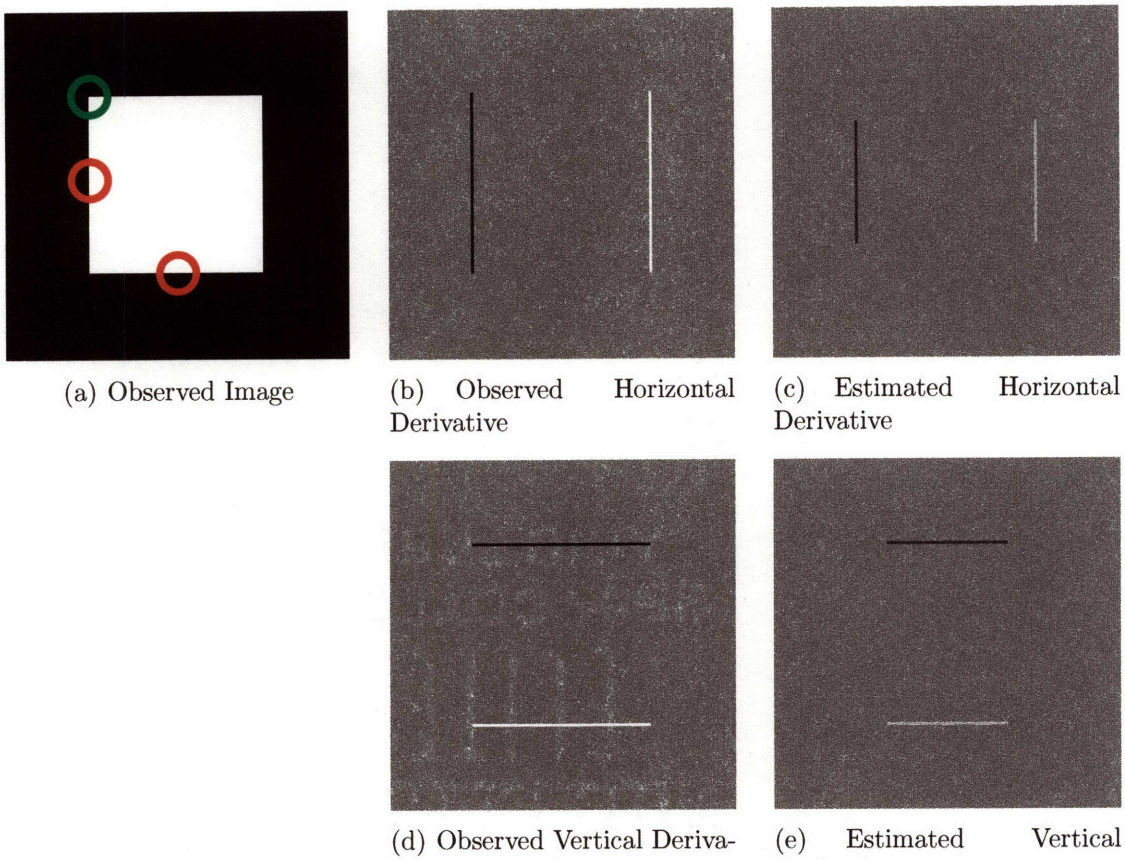
To understand this, consider the simplified, synthetic shading estimation problem shown in Figure 4-1. The observed image in Figure 4-1(a) consists entirely of albedo changes, so the flat image, shown in Figure 4-1(b), is the correct shading image. While this simple example is designed by hand, Section 4.3.3 will consider a similar example that uses trained estimators.

If only local information is used to estimate the derivatives, then good estimates of the derivatives of the shading image can be found in some areas, such as the corner in



(a) Observed Image (b) Shading Image (c) Albedo Image

Figure 4-1: A simple, synthetic example that shows the benefit of weighting or ignoring some of the constraints in the linear system. The square is an albedo change, so the shading image should be a flat image.



(a) Observed Image

(b) Observed Horizontal Derivative

(c) Estimated Horizontal Derivative

(d) Observed Vertical Derivative

(e) Estimated Vertical Derivative

Figure 4-2: Figures (c) and (e) show the estimates of the horizontal and vertical derivatives for the simple observation shown in (a). The lines in the estimates appear shorter because the corners are unambiguous albedo changes, so the derivative should clearly be zero around the corners.

the green circle in Figure 4-2(a). Local data can produce good estimates in areas like this corner because it is unlikely that shading alone could produce a corner. However, local data is insufficient to estimate both the horizontal and vertical derivatives in areas such as those marked with red circles in Figure 4-2(a). This is because a step edge could be either an albedo change or a large shading edge.

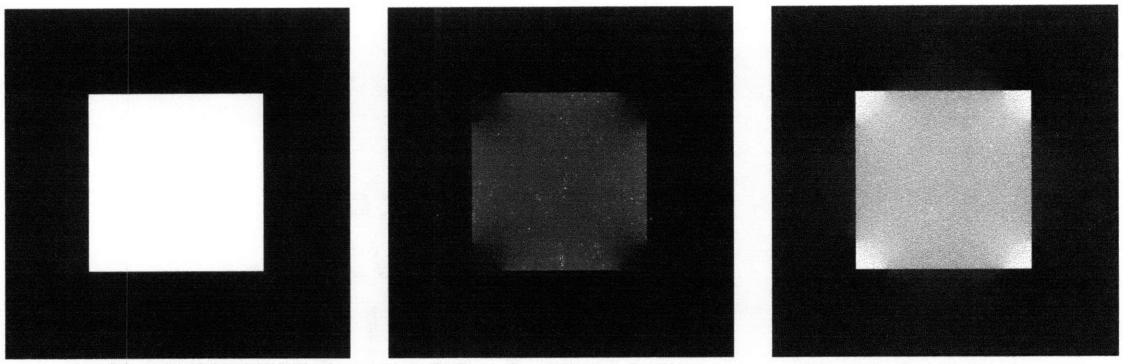
Assuming that it is equally likely that a step edge is either a shading edge or an albedo change, and that the estimates should be optimal under a least-squares criterion, the estimates of the horizontal and vertical derivatives can be found. They are shown in Figures 4-2(c) and 4-2(e).

Comparing these with the observed derivatives in Figure 4-2(b) and Figure 4-2(d), two major aspects of the estimates can be seen. First, around the corners of the square, the estimated derivative values are zero, because the corners are unambiguous albedo changes. Therefore, there should be no changes in the shading image in these areas and the derivatives are zero. However, in areas where only a step edge can be seen locally, the estimated derivatives are non-zero. Because the step edge can be explained equally well as either a shading edge or an albedo change, the optimal choice under a least squares criterion is to put derivatives with 50% of the observed magnitude in at those points. Consequently, the derivatives in Figures 4-2(b) and 4-2(d) appear more gray.

Figure 4-3(b) shows the result of including 50% of the derivatives along the edges. While most of the square has been eliminated, a faint shadow of it remains. In the albedo image, shown in Figure 4-3(c), the corners are bright, because they can be recovered unambiguously, but the edges fade to gray.

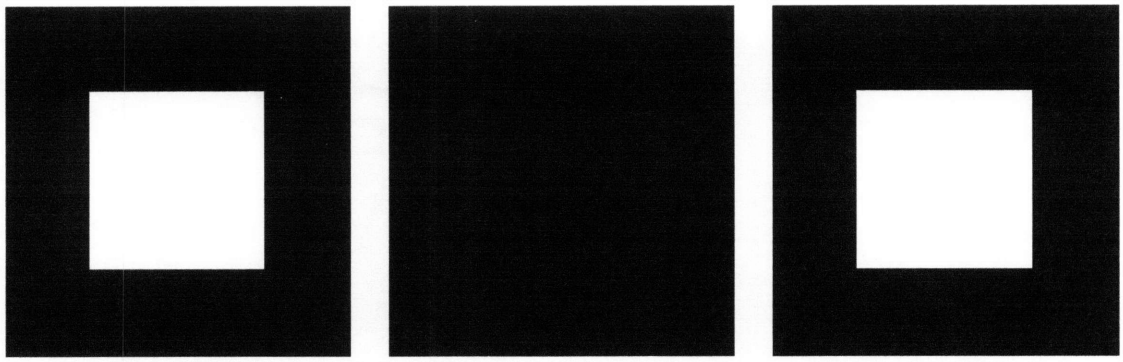
This result can be significantly improved by noticing that along the edges, one of the derivative estimates is zero. Along a vertical edge of the square, the vertical derivative will be zero, regardless of whether the edge should be classified as shading or as an albedo change. This can also be interpreted as saying, "Along a vertical edge, the vertical derivative is reliable, while the horizontal derivative is unreliable, due to ambiguity." Given this knowledge, it is reasonable to ignore the unreliable derivative estimates, while still considering the good estimates. Along the horizontal edges in this simple example, the vertical derivative will be ignored, while the horizontal derivative will be ignored.

This can also be viewed as implicitly propagating information, similar to the explicit Belief Propagation step in [69]. Along a vertical edge, the vertical derivatives



(a) Observed Image (b) Estimated Shading Image (c) Estimated Albedo Image

Figure 4-3: Equally weighting all of the constraints leads to the image in (c). The square remains in the image because of the derivative estimates along the edges, which are ambiguous given only local information.



(a) Observed Image (b) Estimated Shading Image (c) Estimated Albedo Image

Figure 4-4: As shown in (c), ignoring the constraints in certain locations, as described in the text, causes the square to be correctly removed from the image.

are constrained to be zero, which propagates information from the corners, where both the vertical and horizontal derivatives can be estimated reliably, along the edges of the square.

As can be seen in Figure 4-4, ignoring some of the derivatives leads to the correct image being recovered. As will be shown in Section 4.3.6, on real-world data, weighting or ignoring some estimates can lead to significant reductions in error.

4.2 Learning the Estimators using Image Error

4.2.1 Basic Formulation

As described in Section 4.1.1, the distinguishing feature of a closed-loop system is that the parameters of the estimators are optimized using the error in the predicted image as the optimization criterion. If there are N training images in the training set, this error, E, can be formally written as:

$$E = \sum_{n=1}^{N} (\mathcal{T}_n - \mathcal{X}_n)^T (\mathcal{T}_n - \mathcal{X}_n) \tag{4.1}$$

where $\mathcal{T}_n$ is the ground truth target image and $\mathcal{X}_n$ is the estimated image. For the rest of this chapter, it will be assumed that all images, $\mathcal{T}$ and $\mathcal{X}$ have been "unwrapped" into a vector.

If $\mathcal{X}_n$ is found using a pseudo-inverse, Equation 4.1 can be rewritten as

$$E = \sum_{n=1}^{N} (\mathcal{T}_n - (C^T C)^{-1} C^T \hat{c}_n)^T (\mathcal{T}_n - (C^T C)^{-1} C^T \hat{c}_n) \tag{4.2}$$

where C is the constraint matrix described in Section 3.2 and $\hat{c}_n$ holds the estimated constraint values. The estimated constraint values depend on the parameters of the estimators, θ.

If the parameters are to be found using an iterative optimization scheme, such as gradient descent, the gradient of E with respect to one of the estimator parameters, θ_i, can be computed directly:

$$\frac{\partial E}{\partial \theta_i} = 2 \sum_{n=1}^{N} -(\mathcal{T}_n - (C^T C)^{-1} C^T \hat{c}_n)^T (C^T C)^{-1} C^T \frac{\partial \hat{c}_n}{\partial \theta_i} \tag{4.3}$$

For computational efficiency, it is important to note that the matrix-vector product $(\mathcal{T}_n - (C^T C)^{-1} C^T \hat{c}_n)^T (C^T C)^{-1}$ need only be computed once for all θ_i. This product can be computed efficiently by viewing it as the solution to this linear system:

$$(C^T C)x = (\mathcal{T}_n - (C^T C)^{-1} C^T \hat{c}_n) \tag{4.4}$$

The vector x can be computed efficiently using techniques such as Gaussian Elimination.

4.2.2 Training the Mixture of Experts Estimators

The Mixture of Experts estimators have two basic types of parameters: the prototype patch associated with each expert and the linear regression coefficients. While both can be optimized, this section will focus primarily on optimizing the linear regression coefficients.

If one of the linear constraints in the system, such as a horizontal derivative, is indexed by i, then the estimated constraint values are computed from an observation o as:

$$g(o;\theta,p) = \frac{\sum_{i=1}^{N} \exp\left(-\frac{||o-p_i||^2}{h}\right)\left(\theta_i^{\mathrm{T}} o\right)}{\sum_{i=1}^{N} \exp\left(-\frac{||o-p_i||^2}{h}\right)} \tag{4.5}$$

where the notation is the same as that used in Equation 2.13.

If only the regression coefficients $\theta_1 \ldots \theta_N$, are being optimized, it is straightforward to calculate the gradient of the prediction error in Equation 4.2. This gradient can be computed in a particularly succinct fashion by altering the notation of the Mixture of Experts estimator. Given an observed image, $\mathcal{O}$, the estimate for every constraint in the image can be rewritten as:

$$\hat{c} = \mathbf{1}^T \left[S \circ (\bar{\mathcal{O}}\Theta)\right] \tag{4.6}$$

where $\hat{c}$ denotes the values for a linear constraint, $\circ$ is the Hadamard, entry-wise, product, and the vector $\mathbf{1}$ is a vector of all ones. Note that while Equation 4.6 does not index the constraints explicitly, multiple constraints, such as both horizontal and vertical derivatives, are used to estimate intrinsic component images. The vector $\hat{c}$ can represent the estimated values of any of these constraints. The term $\bar{\mathcal{O}}$ denotes a matrix created by concatenating the observation at every pixel in the image $\mathcal{O}$. If the observation vector consisted of a 5×5 patch of pixels, then each column of $\mathcal{O}$ would be an unwrapped version of one of these patches. Denoting each row i of $\bar{\mathcal{O}}$ as o_i, the matrix $S = [s_{i,j}]$ holds the similarity values from Equation 4.5.

$$s_{i,j} = \frac{\exp\left(-\frac{||o_i-p_j||^2}{h}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{||o_i-p_j||^2}{h}\right)} \tag{4.7}$$

The linear regression coefficients $\theta_1 \ldots \theta_N$ form the columns of the matrix Θ:

$$\Theta = [\theta_1 \theta_2 \ldots \theta_{N-1} \theta_N]$$

With these definitions, the derivative of E with respect to Θ over a training set of N images can be written as:

$$\frac{\partial E}{\partial \Theta} = 2 \sum_{n=1}^{N} \bar{\mathcal{O}}_n^T D_n S_n \tag{4.8}$$

where D_n is a diagonal matrix with the diagonal equal to

$$\text{diag}(D_n) = C_f (C^T C)^{-1} (\mathcal{I}_n - (C^T C)^{-1} C^T \hat{c}_n)$$

where $\hat{c}_n$ is the vector of constraint values for all of the constraints used to estimate the intrinsic component images. The matrix C_f is the matrix that expresses the convolution of the image with filter f. This could be the matrix C_{dx} or C_{dy} from Equation 3.1. This form is particularly useful for implementation in matrix-based programming languages such as MATLAB.

While it would be possible to reformulate the derivation in this section to find a closed-form solution for Θ, the ultimate goal is to optimize Θ with other parameters that do not have closed-form solutions. In these cases, it is sufficient to just be able to compute the gradient.

4.2.3 Evaluating Closed versus Open-Loop Training

Using the linear regression coefficients found by the ExpertBoost algorithm as the initial values, the linear regression coefficients can be optimized by minimizing Equation 4.2 using a gradient descent algorithm. Using the same training set and constraints described in Section 3.4.1, I optimized Equation 4.2 using a basic gradient descent algorithm with the "bold driver" heuristic [13]. Instead of maintaining a constant step-size at each iteration, the step size is decreased when the error rises instead of falls.

Optimizing the regression coefficients with regard to image error for 244 iterations of gradient descent led to a significant reduction in the training error, from 3.88×10^7 to 2.47×10^7. Unfortunately, this only led to a small decrease in the testing error. As Figure 4-5 shows, the testing error is only reduced a small amount from 2.46×10^7

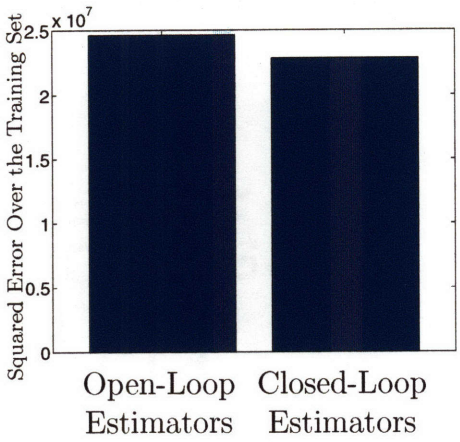


Figure 4-5: This chart compares the Mixture of Expert estimators trained in the previous chapter, referred to as Open-Loop estimators, against estimators whose regression coefficients have been jointly optimized, referred to as Closed-Loop Estimators, to minimize the error on the training set. Surprisingly, the reduction in testing error is relatively minor, showing that open-loop training is a good strategy if closed-loop training is computationally prohibitive.

to 2.44×10^7. This validates the open-loop training done in Chapter 3 as a good strategy for learning to estimate shading images – especially if closed-loop training is computationally prohibitive.

4.3 Learning to Assign Weights to Constraints

Section 4.1.2 demonstrated the usefulness of ignoring constraints in certain cases. In the example shown in that section, ignoring the vertical derivative along an ambiguous horizontal edge led to a much better estimate of the shading image. This section describes how to express this notion of weighting constraints, then describes how to learn the function that assigns weights to the different constraints.

4.3.1 Weighting Constraints

Using a pseudo-inverse to recover the estimate of an intrinsic component image can also be viewed as minimizing a least-squares cost function:

$$\mathcal{X} = \arg\min_{\mathcal{X}} \sum_{n=1}^{N_f} \sum_{p=1}^{N_p} \left[(f_n^p)^T \mathcal{X} - \hat{c}_n^p \right]^2 \tag{4.9}$$

where there are N_f types of constraints, such as vertical or horizontal derivatives. The variable p indexes all of the possible locations for a constraint to be applied, f_n^p represents the coefficients of each constraint, and $\hat{c}_n^p$ represents the estimated constraint values.

The primary point in Section 4.1.2 was that based on the observed image, some of these constraints should be ignored. This can be incorporated into Equation 4.9 by introducing a weighting function $w_n^p(\mathcal{O})$, which is a function of the observed image $\mathcal{O}$. The purpose of this function is to use $\mathcal{O}$ to assign weights to each term in Equation 4.9. Terms that receive a high weight will have a greater effect on $\mathcal{X}$ than terms that receive a relatively low weight:

$$\mathcal{X} = \arg\min_{\mathcal{X}} \sum_{n=1}^{N} \sum_{p=1}^{P} w_n^p(\mathcal{O})((f_n^p)^T \mathcal{X} - \hat{c}_n^p)^2 \tag{4.10}$$

This same idea can be expressed in the matrix form by introducing a weighting matrix W into the recovery step:

$$\mathcal{X} = \left(C^T W(\mathcal{O}) C\right)^{-1} C^T W(\mathcal{O}) \hat{c} \tag{4.11}$$

where $W(\mathcal{O})$ is a diagonal matrix. Each term in $W(\mathcal{O})$ denotes the weight that the constraint in the same row of C should receive.

4.3.2 Learning the Weighting Function

Just as the estimators should be trained in terms of the actual image error, the weighting function $W(\mathcal{O})$ should also be trained to minimize the error in the predicted intrinsic component image. Again, gradient-based optimization methods are a convenient means for minimizing this error.

The optimization criterion is the squared-error in the prediction:

$$E = \sum_{n=1}^{N} ((C^T W(\mathcal{O}) C)^{-1} C^T W(\mathcal{O}) \hat{c}_n - \mathcal{T}_n)^T ((C^T W(\mathcal{O}) C)^{-1} C^T W(\mathcal{O}) \hat{c}_n - \mathcal{T}_n) \tag{4.12}$$

Assuming that $W(\mathcal{O})$ depends on some parameter w, the derivative $\frac{\partial E}{\partial w}$ can be found by taking advantage of an identity from matrix calculus:

$$\frac{\partial A^{-1}}{\partial w} = -A^{-1} \frac{\partial A}{\partial w} A^{-1}$$

Using this identity and the matrix chain rule, the derivative of Equation 4.12 with respect to w can be rewritten as

$$\frac{\partial E}{\partial w} = 2\sum_{n=1}^{N}(\mathcal{X} - \mathcal{T}_n)^T \left[(-A^{-1}C^T\frac{\partial W(\mathcal{O})}{\partial w}C)A^{-1}C^T W(\mathcal{O})\hat{c}_n + A^{-1}C^T\frac{\partial W(\mathcal{O})}{\partial w}\hat{c}_n\right] \tag{4.13}$$

where

$$A = C^T W(\mathcal{O})C$$

and $\mathcal{X}$ is defined as in Equation 4.11.

Equation 4.13 can be rewritten more succinctly as

$$\frac{\partial E}{\partial w} = 2\sum_{n=1}^{N}(\mathcal{X}_n - \mathcal{T}_n)^T \left(A^{-1}C^T\frac{\partial W(\mathcal{O})}{\partial w}\right)(-C\mathcal{X}_n + \hat{c}_n) \tag{4.14}$$

Computationally, Equation 4.14 is significant because it make clear that in addition to computing $\mathcal{X}$, only $A^{-1}(\mathcal{X} - \mathcal{T}_n)$ needs to be computed. Similar to Equation 4.4, this can also be computed efficiently by solving a system of linear equations.

4.3.3 Using Experts as the Basis of the Weighting Function

As in the previous section, the matrix $W_i(\mathcal{O})$ will denote the weight associated with the estimator i. Each element along the diagonal of this matrix will be denoted as w_i^j for the jth element along the diagonal of $W_i(\mathcal{O})$. For a system with N_I linear filters, the complete weighting matrix, W, is a block-diagonal matrix with $W_1(\mathcal{O})\ldots W_I(\mathcal{O})$ along the diagonal.

The prototype patches that parameterize the Mixture of Experts estimators are a natural basis for constructing a function that weights the constraints. This allows image patches similar to some experts to be considered reliable, while patches similar to other experts can be flagged as unreliable. The weight of an estimated filter response is modeled as a function of the normalized similarity between the image patch surrounding the corresponding location in the observed image and the set of prototype patches, $p_1 \ldots p_{N_E}$ associated with the estimator:

$$w_i^j = \sum_{n=1}^{N_E}(\alpha_i^n)^2 \frac{e^{-\sum_s (o_j^s - p_n^s)^2}}{\sum_{n=1}^{N_E} e^{-\sum_s (o_j^s - p_n^s)^2}} \tag{4.15}$$

Each α term is squared simply to ensure that it is not negative. The coefficient α^2 should be close to zero for an expert that produces poor quality estimates. This will cause filter responses from patches similar to the prototype defining that estimator to receive low weight. One set of weights is found for each type of linear constraint.

4.3.4 Learning the Weights α

The weights, $\alpha^1 \ldots \alpha^{N_E}$, associated with a particular linear constraint, f, are found by minimizing the squared error in Equation 4.12. This can be computed succinctly in matrix form, similar to computing the gradient of the linear regression coefficients. Using the same notation as in Section 4.2.2, $\bar{\mathcal{O}}$ denotes a matrix created by concatenating the observation vectors from $\mathcal{O}$. The matrix $S = [s_{i,j}]$ holds the similarity values from Equation 4.5:

$$s_{i,j} = \frac{\exp\left(-\frac{||o_i - p_j||^2}{h}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{||o_i - p_j||^2}{h}\right)}$$

If $\hat{\alpha}_i$ is an $N \times 1$ vector of the weight values, the derivative of $\hat{\alpha}_i$ can be written as:

$$\frac{\partial E}{\partial \hat{\alpha}_i} = 4 \sum_{n=1}^{N} \left(\hat{c}_n^f - C_f \mathcal{X}_n\right)^T D_n S D_{\alpha_i} \tag{4.16}$$

where $\hat{c}_n^f$ holds the estimates of the response to filter f. The matrix C_f is a convolution matrix with the same notation as in Equation 4.8. The matrix D_n is a diagonal matrix with the diagonal equal to

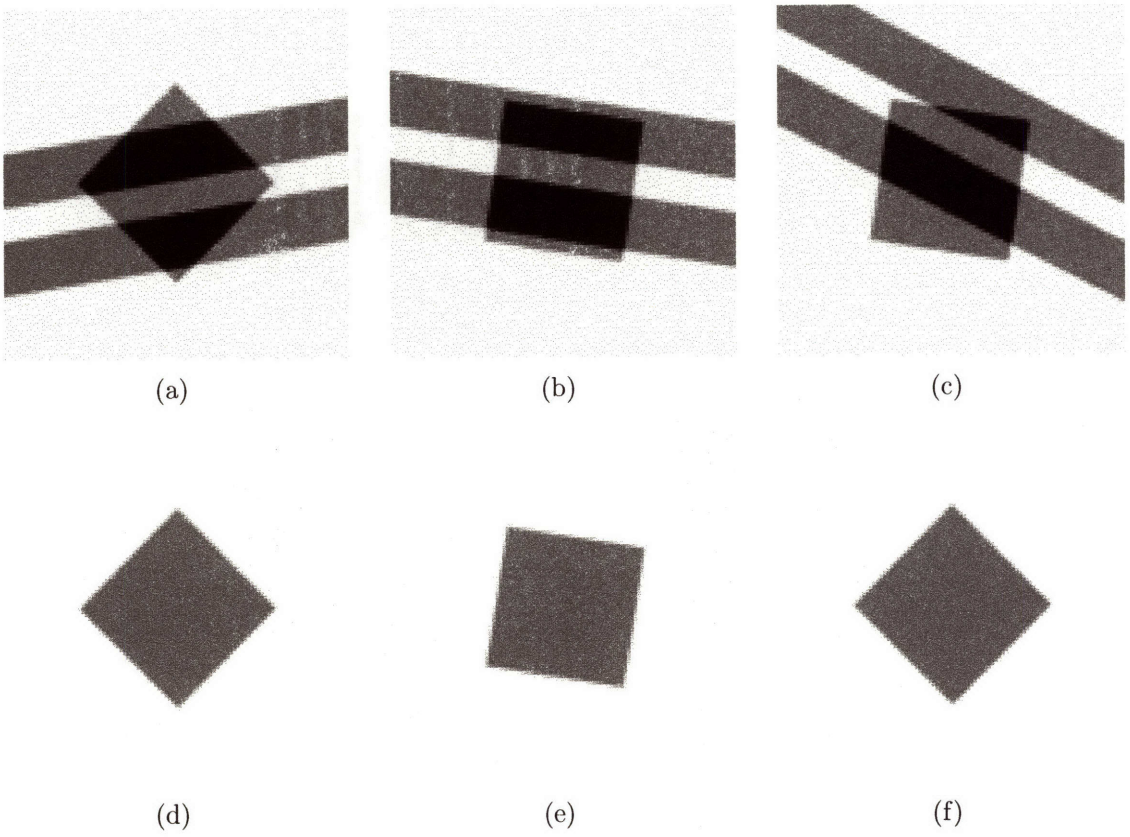
$$\text{diag}(D_n) = C_f (C^T C)^{-1} (\mathcal{I}_n - (C^T C)^{-1} C^T \hat{c}_n)$$

The final matrix D_{α_i} is also a diagonal matrix with α_i along the diagonal.

Note that while S contains similarity values in this formulation, it can be replaced with a more general set of features.

4.3.5 A simple example

To understand the value of learning these weights, this section will describe a simple example, similar to the square example in Section 4.1.2, that makes clear the benefits of learning to weight the constraints. Three of the training images in this simple



(a) (b) (c)

(d) (e) (f)

Figure 4-6: The training set used for the simple example in Section 4.3.5. The images in the top row are the observations and the images in the bottom row are the target albedo images.

example are shown in Figure 4-6. The albedo is always a square that has been been rotated. The shading is always a pair of bars that are also rotated. The training set, which has 40 images, is constructed so that for every edge in the image, an edge with the same orientation and gradient magnitude has appeared in the test set as both shading and an albedo change.

While this training set seems quite simple, from a local estimation point of view, it is quite difficult. The construction of the training set guarantees that edges are locally ambiguous. The only good local evidence is at corners and at the junctions of the bars and square.

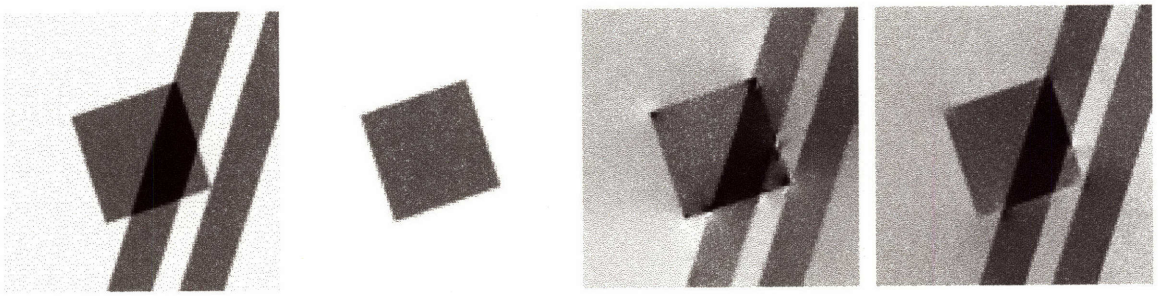
To evaluate the behavior of the intrinsic components system on this data set, I trained estimators to estimate the response of a set of Haar wavelet filters:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & -1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & -1 \\ 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

The estimators were trained to estimate the response of these filters on the albedo image, which is always a rotated square. Because these images are dominated by edges, the training set of image patches was one-third edges and one-third junctions and corners. These patches were identified with the Harris corner detector provided by Kovesi [41]. In addition, one-third of the patches were drawn at random from the training set. The interest point detector was used because most of these images are flat, so randomly sampling points would swamp the training set with a large number of identical points.

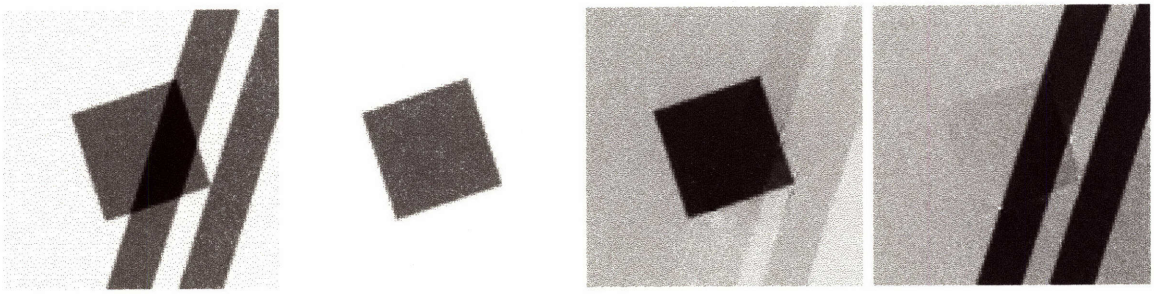
Figure 4-7(c) shows the estimated albedo image given the observation in Figure 4-7(a). Overall, the result is rather poor – the bars and the squares have basically the same intensity and the estimated shading and albedo images basically look identical. This is due to the ambiguities in the training set. Given only an edge, which could be both shading or a reflectance change, the estimators choose to "split the difference" and assign the edge to both images.

The bright spots around the corners of the squares and the junctions of the squares and bars are interesting artifacts in Figure 4-7(c). The bright spots occur in these areas because the local evidence is unambiguous. Thus, instead of splitting the difference, the estimators produce a much closer estimate of the derivative of the square. This estimate tends to have a much higher magnitude and leads to the bright spots.



(a) Observation (b) True Albedo Image (c) Estimated Albedo Image (d) Estimated Shading Image

Figure 4-7: Figure (c) shows the albedo image estimated from the observation in (a), when giving equal weight to all of the constraints. The system is basically unable to separate the shading from the albedo.



(a) Observation (b) True Albedo Image (c) Estimated Albedo Image (d) Estimated Shading Image

Figure 4-8: Figure (c) shows the albedo image estimated from the observation in (a), with each constraint weighted independently. The system is now able to separate the shading from the albedo quite well.

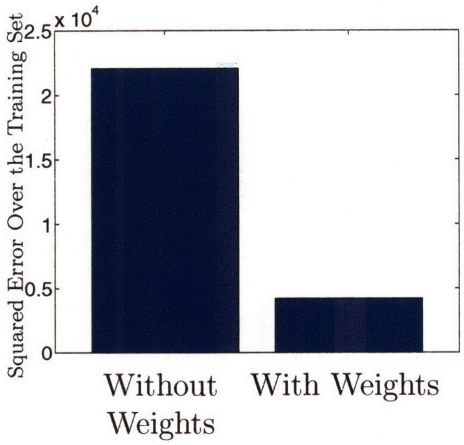
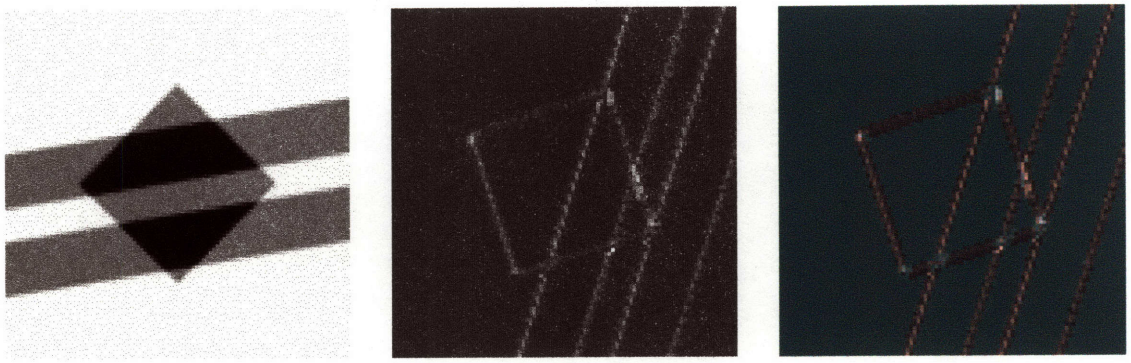


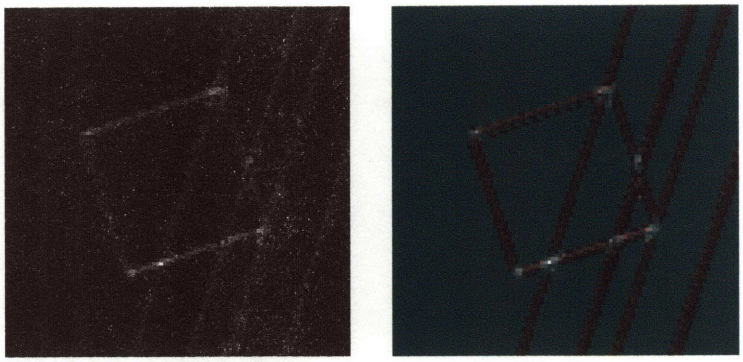
Figure 4-9: This chart shows that adjusting the weights of the constraints leads to a significant reduction in the error over the square and bar images.



(a) Observation

(b) Abs. Value of Horizontal Wavelet

(c)



(d) Abs. Value of Vertical Wavelet

(e)

Figure 4-10: The figures in (c) and (e) show the correlation between the weight assigned to a constraint and the magnitude of the constraint's value. For instance, Figure (b) shows the magnitude of the constraint values for the horizontally-oriented filter. In Figure (c), points whose weight is below a low threshold of 0.2 are marked in red. The only points with large magnitude responses that are not marked in red are points near corners and junctions. This demonstrates that the system has correctly learned that only the corners and junctions provide meaningful information.

As Figure 4-8 shows, the results can be significantly enhanced by learning to weight the constraints. The weights α were found using the same method that will be discussed in Section 4.3.4, including the use of the steerable constraint described in Section 4.3.8. The bars are almost totally removed from the albedo image. This significant increase in quality of the estimated images can also be seen in quantitative error. As shown in Figure 4-9, weighting the constraints reduces the total squared error over the training set from 2.2×10^4 to 4.2×10^3.

These weights demonstrate the correct behavior for solving this problem. Figure 4-10(b) shows the absolute value of the estimate of the horizontally oriented Haar wavelet. Next to it, Figure 4-10(c) shows the same response. In this image though,

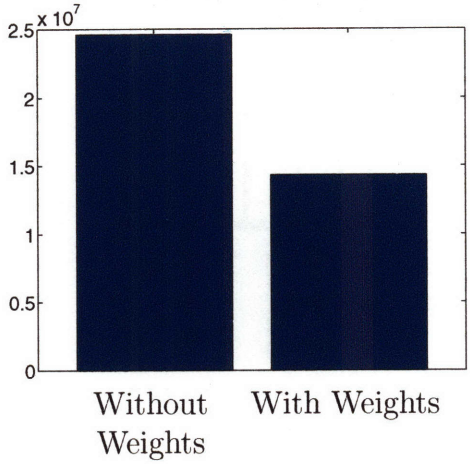


Figure 4-11: This chart shows that adjusting the weight of the constraints leads to a significant decrease in the error on the test set from Chapter 3.

pixels where the weight assigned to this constraint is below a low threshold of 0.2 are marked in red. Figures 4-10(d) and 4-10(e) show the same data for the vertically oriented filter. Notice that in most of the image, pixels are marked in red where the estimated response is large, such as along edges. The only exception to this is at the corners of the square and at the junctions of the bars and the square. Effectively, the system has correctly learned that good information can be found at the corners and junctions and that estimated response values should be ignored along edges.

4.3.6 The Benefits of Weight Estimates for Shading and Albedo

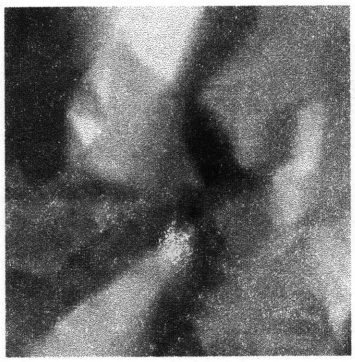
The previous section showed the benefits of weighting the constraints on a relatively simple data set. Fortunately, adjusting the weights also leads to a significant improvement in the estimates of the shading and albedo images.

Using the horizontal and vertical derivative estimators found in Section 3.4, the error in Equation 4.12 was minimized by optimizing the weighting terms α, from Equation 4.15, using the bold-driver gradient descent technique from Section 4.2.2. The gradient descent algorithm was run for 2000 iterations.

As shown in Figure 4-11, weighting the constraints significantly reduces the squared-error in the test set. The squared-error is reduced from 2.46×10^7 to 1.43×10^7. It also leads to qualitative improvements, as seen in Figure 4-12. As Figure 4-12(g) shows, the albedo image recovered with weighted constraints does a better job of capturing the flat appearance of the albedo image.

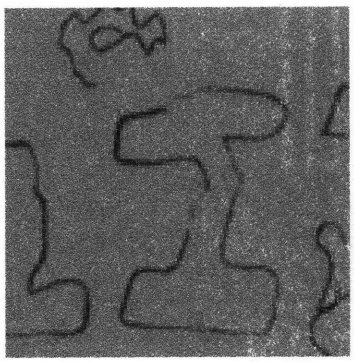
(a) Observed Image

(b) Ground Truth Shading Image

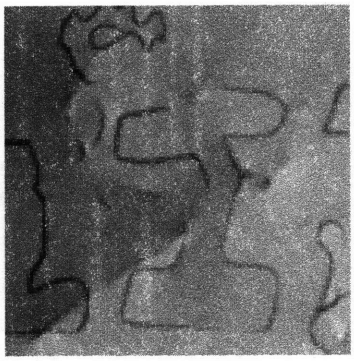


(c) Estimated Shading Image without Weights

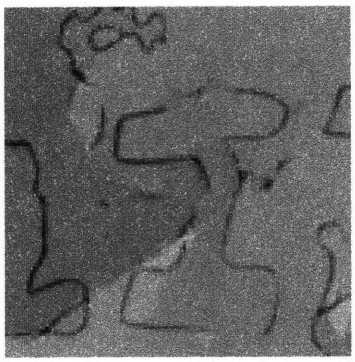
(d) Estimated Shading Image with Weights



(e) Ground Truth Albedo Image

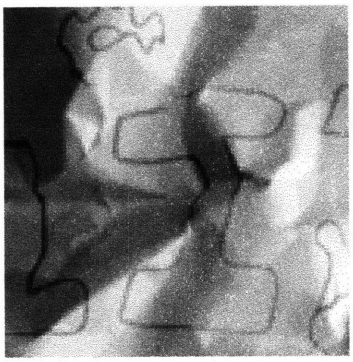


(f) Estimated Albedo Image without Weights



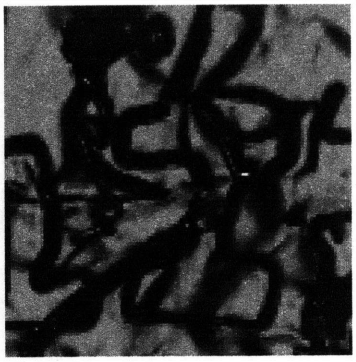
(g) Estimated Albedo Image with Weights

Figure 4-12: Shading and albedo images estimated with and without weighted constraints. The albedo image estimated with weighted constraints shows less shading and has a flatter appearance.



(a) Observed Image

(b) Weight of Horizontal Derivative



(c) Weight of Vertical Derivative

(d) Thresholded Weight of Horizontal Derivative

(e) Thresholded Weight of Vertical Derivative

Figure 4-13: These images show the weight assigned to each point in the observed image shown in (a). The weights in the second row have been thresholded to allow higher contrast.

4.3.7 What are Reliable Patches for Estimating Shading?

Examining the weight assigned to each derivative in the image is a convenient way of visualizing how the system has chosen to assign uncertainty to different types of image patches. Figure 4-13 shows the weight assigned to the horizontal and vertical derivatives, given the observed image in Figure 4-13(a). Most notably, the highest weight is assigned to smooth or flat regions. This is because these are the easiest type of patches from which to estimate derivative values. If the patch is flat, then it is very likely that both the albedo and shading images are flat at that point.

The other notable aspect of these images is that the weight around the edges is very low. This is because the correct value at the edges is more difficult to estimate. However, changing the scale so that the highest-weight portions of the image are white, shown in Figures 4-13(d) and 4-13(e), shows interesting behavior. In portions

of the image along small folds in the paper, the vertical derivative is assigned a much higher weight that the horizontal derivative. This likely because the vertical derivative will be close to zero whether those edges belong in the shading image or not. The horizontal derivative is more ambiguous because the light comes from the right of the image, which leads to many strong images caused by shading. The presence of strong edges caused by both shading and albedo changes makes it difficult to estimate the derivative of the shading image using only local information.

A second way of understanding how the system has chosen to assign weight is to rank the prototype patches by the weight assigned to them. To aid visualization, the system was retrained using 7 $\times$7 image patches. In addition, these patches were selected from only the paper images. This is because the synthetic data can be well separated using Retinex-style intensity cues, which makes interpreting the patches difficult because it is not clear whether a patch is reliable due a Retinex-style cue or a different cue. Figures 4-14, 4-15, and 4-16 show a selection of the prototype patches used for estimating the horizontal derivative and the weight assigned to them. For display purposes, the patches were sorted according to the weight assigned to each patch. The 800 patches were then divided into groups of 40 patches. For brevity, six of the groups are shown in these figures.

The patches with high weights are dominated by flat areas, where it is easy to estimate the derivative of the shading image. In addition, many of the patches with a relatively large weight are lines or bars, which are not likely to be caused by shading. These figures also show that vertical edges tend to receive relatively low weights, probably for the same reasons discussed above.

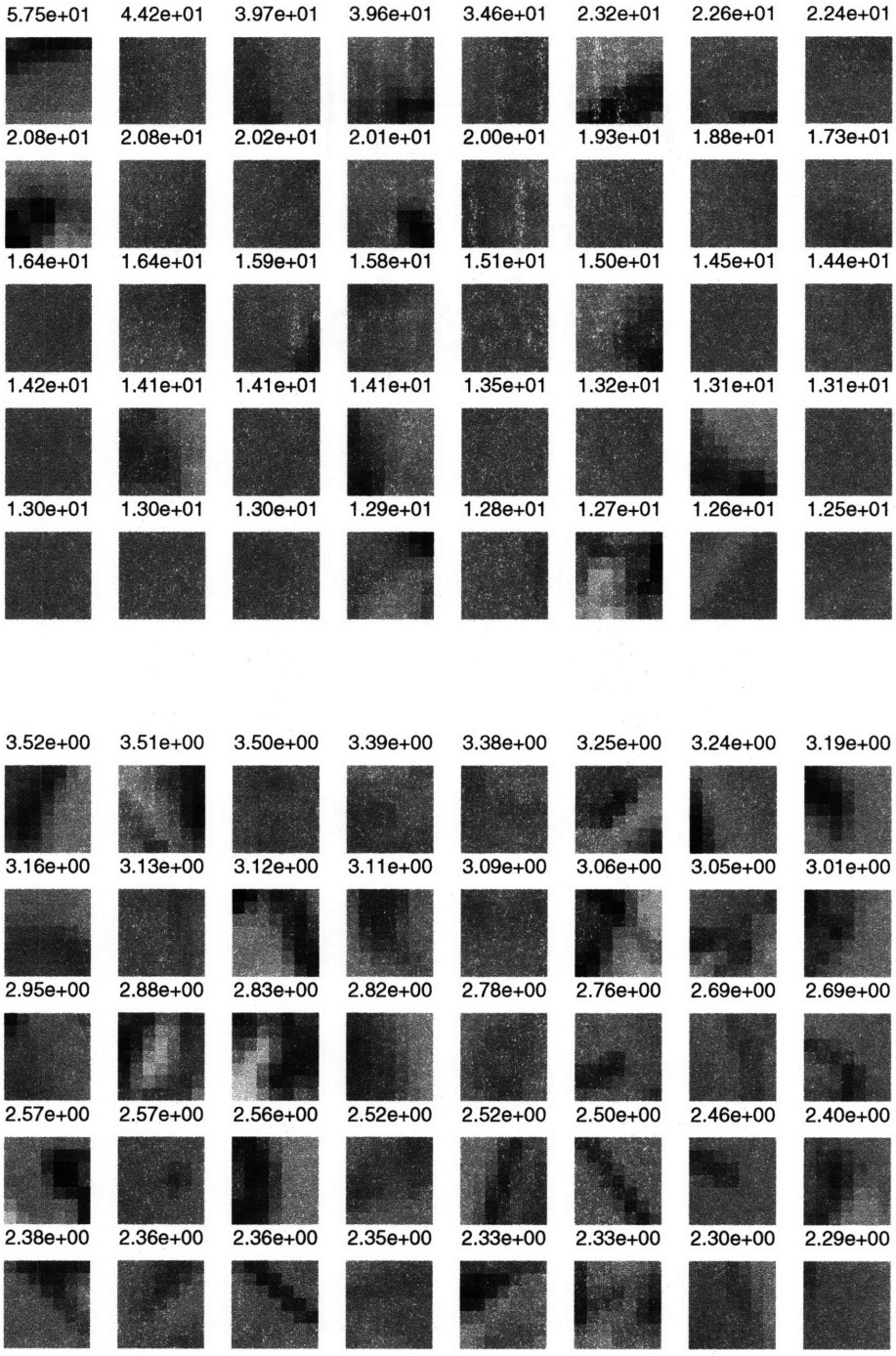


Figure 4-14: A selection of the prototype patches. The number above each patch is the weight assigned to that patch. The patches are sorted according to these weights. These patches come from a system trained on only the paper images, using 7×7 image patches as the input representation. As described in the text, each panel of patches is a sorted group of 40 patches.

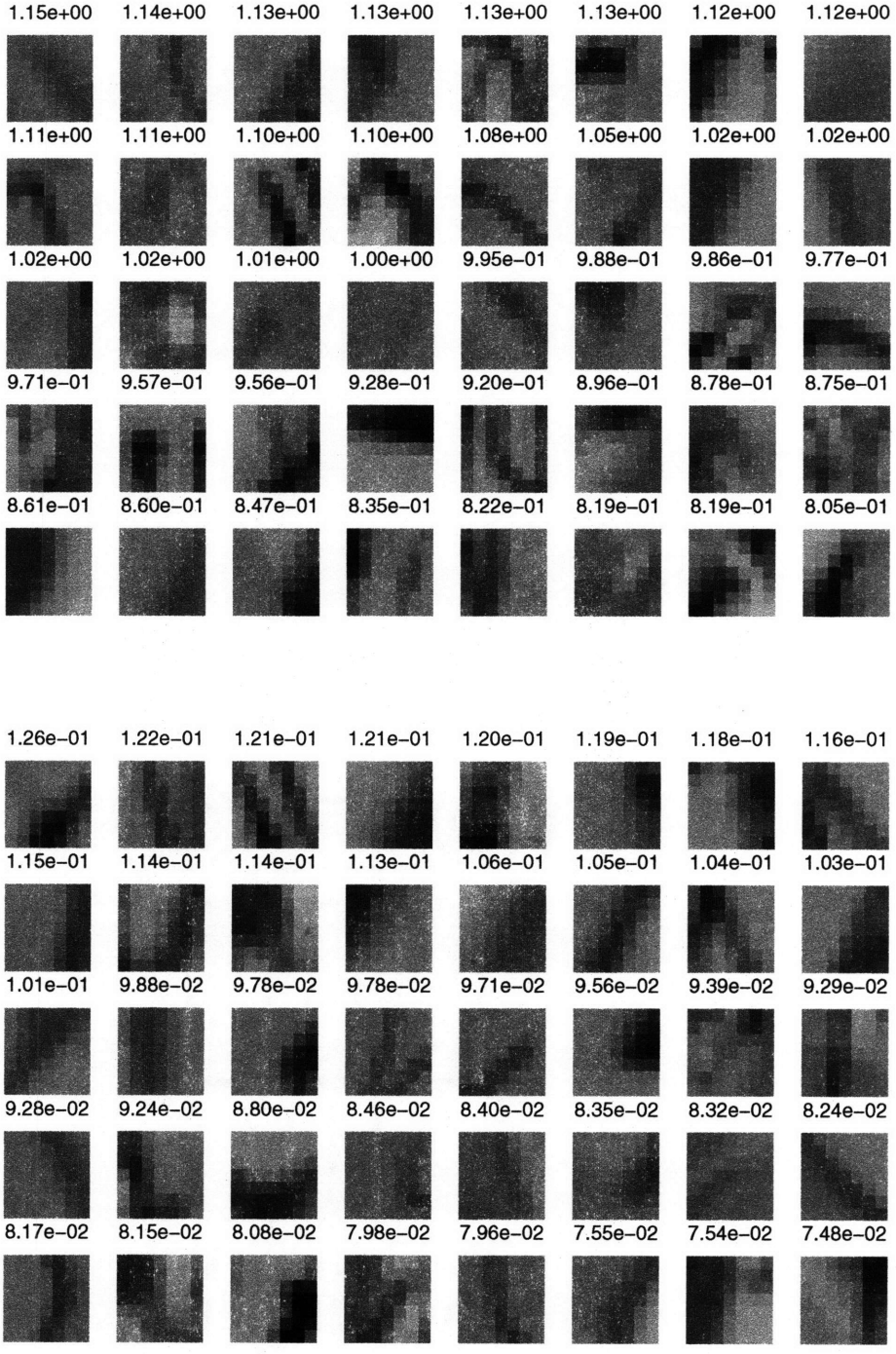


Figure 4-15: A selection of the prototype patches. The number above each patch is the weight assigned to that patch. The patches are sorted according to these weights. These patches come from a system trained on only the paper images, using 7×7 image patches as the input representation. As described in the text, each panel of patches is a sorted group of 40 patches.

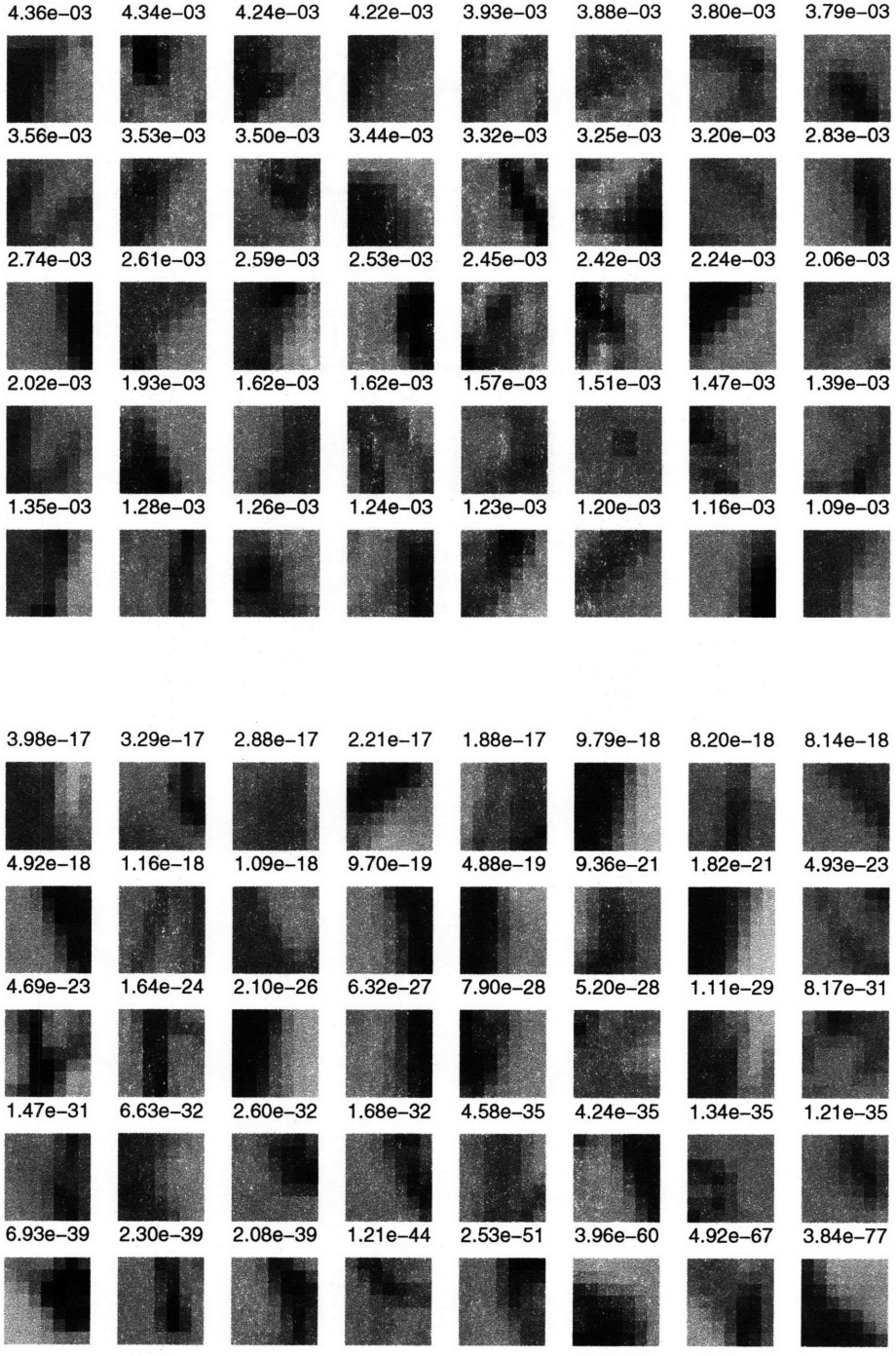


Figure 4-16: A selection of the prototype patches. The number above each patch is the weight assigned to that patch. The patches are sorted according to these weights. These patches come from a system trained on only the paper images, using 7×7 image patches as the input representation. As described in the text, each panel of patches is a sorted group of 40 patches.

4.3.8 Incorporating Steerable Constraints

The linear constraints used by the system are not limited to fixed filters. The constraints themselves can also depend on the observed image. Steerable constraints are one example of a constraint that depends on the constraint matrix. The constraint matrix C_s can be written as the sum of a horizontal and vertical derivative matrices:

$$C_S = \text{diag}(\cos(\Theta_i(\mathcal{O})))C_{dx} + \text{diag}(\sin(\Theta_i(\mathcal{O})))C_{dy} \tag{4.17}$$

where $\Theta_i(\mathcal{O})$ is a function of the observed image $\mathcal{O}$ that provides the steering orientation at location i. The matrix C_S basically represents a steered filter [25].

To test the benefit of a steered constraint, I added the constraint that the derivative steered parallel to the observed image edge should be equal to the value of zero. Since an estimator is not trained for this constraint, the prototype patches from both the horizontal and vertical derivatives are used for this constraint. The weight coefficients, α, were trained using the gradient descent method described in Section 4.2.2. This constraint can be viewed as expressing the notion that edges should have constant intensity. Incorporating this constraint reduced the error from 1.43×10^7 to 1.31×10^7.

4.4 Joint Optimization of the Linear Regression Coefficients and the Weights

Using gradient descent, it is easy to jointly optimize the regression coefficients and the weighting coefficients. Contrary to my expectations, this performed worse than simply optimizing the weighting coefficients. As Figure 4-17 shows, optimizing only the weighting coefficients achieves a lower training error than joint optimization.

However, the training error can be significantly reduced by first optimizing only the weighting coefficients, then jointly optimizing the regression and weighting coefficients. After 600 additional iterations of gradient descent, the training error decreases from 1.4169×10^7 to 9.80×10^7. However, the error on the testing set actually increases slightly after training. This indicates that the training is likely overfitting the training data.

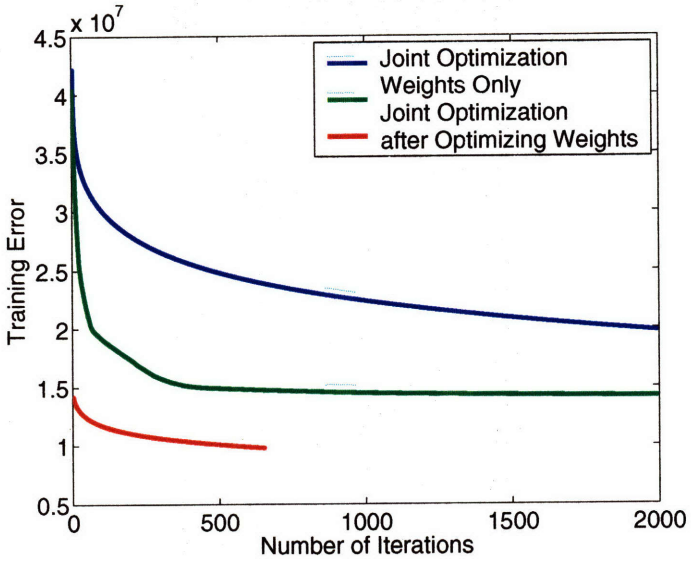


Figure 4-17: The training error at each iteration of gradient descent. The blue line is the training error when both the regression coefficients and the weighting coefficients are jointly optimized. The green line is the training error when only the weighting coefficients are optimized. The red line shows the training error when jointly optimizing the weights and filters, after first optimizing the weights.

4.5 The Advantage of an Energy Based Criterion

In the previous sections, the learning has been formulated as minimizing the squared error in the predicted intrinsic component image. This section shows how the learning can be reformulated probabilistically. The pseudo-inverse step can also be interpreted as finding the MAP estimate of Conditional Markov Random Field defined as

$$p(\mathcal{X}|\mathcal{O}) = \frac{1}{Z} \exp\left(\sum_{n=1}^{N}\sum_{p=1}^{P}((f_n^p)^T\mathcal{X} - \hat{c}_n^p)^2\right) \tag{4.18}$$

where the terms in this equation have the same meaning as in Equation 4.9. The quantity Z is the normalizing constant for the density function, also knows as the partition function. The cliques in this MRF are defined by the linear constraints f_n^p. If all the constraints take the form of a 5×5 filter, then the cliques will all be 5×5 pixel neighborhoods in the image. This MRF is conditioned on the observed image $\mathcal{O}$ because of the $\hat{c}$ terms, which depend on $\mathcal{O}$. This type of MRF is also known as a Conditional Random Field or CRF [42]. The terms in Equation 4.18 are often referred to as potential functions or compatibility functions.

In matrix form, this MRF can also be expressed as a Gauss-Markov Random Field

(GMRF) [63]:

$$p(\mathcal{X}|\mathcal{O}) = \frac{1}{(2\pi)^{\frac{N_p}{2}}|C^TC|}\exp\left(-\frac{1}{2}(\mathcal{X}-\mu)^TC^TC(\mathcal{X}-\mu)\right) \tag{4.19}$$

where

$$\mu = (C^TC)^{-1}C^T\hat{c}$$

The estimate of the conditional distribution of the intrinsic component image can be improved by weighting certain cliques more than others:

$$p(\mathcal{X}|\mathcal{O}) = \frac{1}{Z}\exp\left(\sum_{n=1}^{N}\sum_{p=1}^{P} w_n^p(\mathcal{O})((f_n^p)^T\mathcal{X} - \hat{c}_n^p)^2\right) \tag{4.20}$$

Again, the notation for the weighting functions $w(\mathcal{O})$ is the same as in Equation 4.10.

Given this probabilistic setting, the natural way to find the parameters of $w(\mathcal{O})$, denoted here as θ_w is to maximize the conditional log-likelihood of the data, $\mathcal{T}$. This can be accomplished using gradient descent. The gradient of the log likelihood with respect to θ_w is:

$$\begin{aligned}\frac{\partial \log \mathcal{T}}{\partial \theta_w} &= \sum_{n=1}^{N}\sum_{p=1}^{P}\frac{\partial w_n^p(\mathcal{O})}{\theta_w}((f_n^p)^T\mathcal{T} - \hat{c}_n^p)^2 \\ &\quad -\frac{1}{Z}\int_{\mathcal{X}}\exp\left(\sum_{n=1}^{N}\sum_{p=1}^{P}\frac{\partial w_n^p(\mathcal{O})}{\partial\theta_w}((f_n^p)^T\mathcal{X} - \hat{c}_n^p)^2\right)\end{aligned} \tag{4.21}$$

Note that because of the dominated convergence theorem, the derivative of the second term can be moved inside the integral because θ_w is constant in the integral [76].

Notice that the second term can be rewritten as an expectation:

$$\begin{aligned}\frac{\partial \log \mathcal{X}}{\partial \theta_w} &= \sum_{n=1}^{N}\sum_{p=1}^{P}\frac{\partial w_n^p(\mathcal{O})}{\theta_w}((f_n^p)^T\mathcal{T} - \hat{c}_n^p)^2 \\ &\quad -\mathbb{E}_{p(\mathcal{X};\mathcal{O},\theta_w)}\left[\sum_{p=1}^{P}\frac{\partial w_n^p(\mathcal{O})}{\partial\theta_w}((f_n^p)^T\mathcal{X} - \hat{c}_n^p)^2\right]\end{aligned} \tag{4.22}$$

The expectation, denoted $\mathbb{E}_{p(\mathcal{X};\mathcal{O},\theta_w)}$, is computed under the conditional distribution $p(\mathcal{X};\mathcal{O},\theta_w)$.

Equation 4.22 makes it clear that optimizing the parameters of the weighting

function requires the ability to compute the expected values of the potential functions. Because the problem can be formulated as a Gauss-Markov Random Field, the expected value of these potential functions can be computed easily from the covariance matrix of $p(\mathcal{X}|\mathcal{O}, \theta_w)$. This is accomplished most easily by inverting the matrix $C^T W(\mathcal{O}) C$.

Unfortunately, inverting a matrix is $\mathcal{O}(N_p^3)$, if N_p is the number of pixels in the image [53]. In addition, each gradient computation will require inverting this matrix. On the other hand, as pointed out in Section 4.3.4, calculating the gradient when minimizing the squared error only requires computing the product of a vector with an inverse matrix. Computing these products is equivalent to solving a linear system, which is $\mathcal{O}(N_p^2)$. For images, N_p will typically be very large, leading to tremendous savings in computation.

4.6 The Difficulty of Adjusting Weights under an L_1 Norm

In the previous sections, the estimated intrinsic component image, $\mathcal{X}$, is reconstructed from the estimated constraint values using a pseudo-inverse. In the previous chapter, Section 3.11 showed that the results could be significantly improved by reconstructing the images using an L_1 norm rather than an L_2 norm, which is the effective norm when using a pseudo-inverse. Given the significant improvements that come from learning to weight the estimates using the pseudo-inverse, it is likely that similar improvements can be obtained by weighting the constraints when reconstructing using an L_1 error norm.

To understand the difficulty of learning to weight the estimates under the L_1 norm, it is instructive to rewrite the error criterion from Equation 4.12 in this fashion:

$$\begin{aligned} \text{minimize}_{\mathcal{X}} \quad & \sum_{n=1}^{N} (\mathcal{X}_n - \mathcal{T}_n)^T (\mathcal{X}_n - \mathcal{T}_n) \\ \text{s.t} \quad & \mathcal{X} = \arg\min_x \sum_{n=1}^{N} \sum_{p=1}^{P} w_n^p(\mathcal{O}) || (f_n^p)^T \mathcal{X} - \hat{c}_n^p || \end{aligned} \tag{4.23}$$

where $||\cdot||$ could represent any norm. If the L_2 norm is used, then the arg min step is

a set of matrix multiplications and the value of $\mathcal{X}$ can be differentiated analytically with respect to the parameters of the weighting functions $w_n^p(\mathcal{O})$. However, if an L_1 error norm is used, then $\mathcal{X}$ is computed using a linear program similar to the one used in Section 3.11. The value of $\mathcal{X}$ can then no longer be differentiated analytically.

This problem can be avoided by considering a different approach to learning the weighting parameters. In [18], Collins describes how to estimate the parameters of models of the form:

$$\hat{x} = \arg\min_{\hat{x} \in \bar{\mathcal{X}}} \Phi(\hat{x}, \hat{y}) \cdot \alpha \tag{4.24}$$

where $\bar{\mathcal{X}}$ is the set of all possible values of $\hat{x}$, α is a vector of parameters, and $\Phi(x, y)$ is a vector of feature functions that relate possible values of $\hat{x}$ with the observation y.

To estimate the weight of the features α, Collins suggests the perceptron algorithm [55], which has historically been used to learn classifiers. The α parameters are chosen to maximize a margin between the correct estimate and all other possible estimates.

I implemented this algorithm as described in [18], including the averaging of parameters suggested in that paper. Unfortunately, the perceptron algorithm proved unstable and difficult to work with. In early experiments, it gave good results, while failing in later experiments. The primary difficulty with using the perceptron algorithm is that there are no theoretical guarantees about its performance if the data is not linearly separable. This makes it difficult to determine if the algorithm is behaving correctly and also makes it difficult to evaluate the generalization performance of the estimator returned by the algorithm.

4.6.1 A simple option

A simple option for learning the parameters of the weighting function is to train them using the L_2 norm, then use them with the L_1 norm. The drawback to this approach is that the weighting function parameters are tailored to the behavior of the L_2 norm. This actually increases the squared error over the test set. The error likely rises because the weights are tailored to the behavior of minimization under an L_2 norm.

4.7 Relationship to Other Work

Learning the parameters w to minimize the Equation 4.23 is similar to the work on the problem of structured prediction. The primary difference lies in the criteria. The work of Taskar et al. [71] and Collins [18] focuses on maximizing a margin between between the true values and all other values. In this work, the criterion is simply the loss function. It is possible to optimize only the loss because the prediction is produced from a quadratic cost function, which makes it possible to analytically differentiate the value of the arg min step.

In the context of semi-supervised learning, Zhu et al [83] have also optimized the parameters of a Gaussian field to minimize the entropy over the data.

4.8 Conclusion

This chapter has addressed two main issues: closed-loop versus open-loop learning and learning to weight the constraints implied by the estimates of the linear filter responses. Both on synthetic examples and real-world shading images, weighting the estimates leads to significantly better performance. This chapter has also shown how these weights can be learned efficiently by minimizing the loss in the predicted image. This is much more efficient than posing the problem as a maximum-likelihood learning problem.

Chapter 5

Conclusion

This thesis has presented a system that takes a single grayscale image and decomposes it into intrinsic component images that represent certain intrinsic characteristics of the scene.

This estimation process consists of two basic steps. Assuming that the desired intrinsic component image is the shading image, the first step is to use the observed image to estimate filtered versions of the shading image, using small patches of image data. Each of these estimated filter responses can be seen as a constraint on the appearance of the shading image. The second step is to find the image that best satisifies these constraints, under either a squared-error or absolute-error metric.

In Chapter 3, the effectiveness of this system was demonstrated on two types of decompositions, shading/albedo and scene/noise. On the shading and albedo task, the system achieves state-of-the-art results on images that are more difficult than those used previously. On the denoising task, this system was competitive with the recently published Field of Experts of Roth and Black [56].

5.1 Contributions of This Work

The primary purpose of this work has been to develop a state-of-the-art system for estimating shading and albedo images. However, as shown by the system's extension to denoising, the techniques described are flexibile enough to be applied to other image estimations. The unique contributions of this work include:

Treating Lightness Estimation as Regression In this work, the problem of estimating the albedo of a point as been posed as a continuous estimation problem.

This avoids the problems of classification based systems, such as [69]. In addition, it makes it easier to incorporate a wider variety of linear constraints.

Ground-Truth Data and Quantative Evaluation A unique feature of this approach is the use of ground-truth shading and albedo images to evaluate the various design decisions in the construction of the estimators. The training and test images will be made available. I hope that the availability of this new data will spur new research and approaches to this problem.

Learning to Weight the Constraints In Chapter 4, the system is extended by assigning weights to the various constraints, based on the observed image. The function that assigns these weights can be optimized on the training data. As Chapter 4 shows, this is equivalent to learning the parameters of a Conditionally Gaussian Markov Random Field. This is a unique contribution to the body of work in MRF-based techniques in computer vision. While there is much work that relies on hand-designed MRF's, systems that learn the MRF parameters are much less common. This is likely due to the computational cost of computing expectations in MRF models. As Chapter 4 shows, this can be avoided by minimizing the error in image estimates.

Bibliography

[1] The MOSEK optimization toolbox for matlab. http://www.mosek.com.

[2] The MOSEK optimization toolbox for matlab user's guide and reference manual. http://www.mosek.com.

[3] Edward H Adelson. Lightness perception and lightness illusions. In M Gazzaniga, editor, *The New Cognitive Neurosciences, 2nd Edition*, pages 339–351. MIT Press, Cambridge, MA, 2001.

[4] Edward H Adelson and P Burt. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics*, 2(4):217–236, 1983.

[5] Edward H Adelson and Alex P Pentland. The perception of shading and reflectance. In D Knill and Whitman Richards, editors, *Perception as Bayesian Inference*, pages 409–423. Cambridge University Press, 1996.

[6] H Akaike. Information theory and an extension to the maximum likelihood principle. In *Second International Symposium on Information Theory*, pages 267–281, 1973.

[7] C G Atkeson, A W Moore, and S Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1–5):11–73, 1997.

[8] Ran Avnimelech and Nathan Intrator. Boosting regression estimators. *Neural Computation*, 11(2):499–520, 1999.

[9] H G Barrow and J M Tenenbaum. Recovering intrinsic scene characteristics from images. In A. Hanson and E. Riseman, editors, *Computer Vision Systems*, pages 3–26. Academic Press, 1978.

[10] Matt Bell and William T Freeman. Learning local evidence for shading and reflection. In *Proceedings International Conference on Computer Vision*, 2001.

[11] R E Bellman. *Adaptive Control Processes.* Princeton University Press, 1961.

[12] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[13] Christopher M Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[14] M J Black and P Anandan. Robust dynamic motion estimation over time. In *IEEE CVPR*, pages 296–302, 1991.

[15] C J C Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[16] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at *http://www.csie.ntu.edu.tw/ cjlin/libsvm.*

[17] W S Cleveland and S J Delvin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of American Statistical Association*, 83(403):596–610, 1988.

[18] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP 2002*, 2002.

[19] Michael Collins, Robert E. Schapire, and Yoram Singer. Logistic regression, adaboost and bregman distances. In *Conference on Learning Theory*, 2000.

[20] M Datar, P Indyk, N Immorlica, and V. Mirrokn. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Symposium on Computational Geometry*, 2004.

[21] Stephen DellaPietra, Vincent DellaPietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, April 1997.

[22] G. Finlayson, M. Drew, and C. Lu. Intrinsic images by entropy minimization. In *Proceedings, 8th European Conference on Computer Vision, Prague*, pages 582–595, 2004.

[23] Graham D. Finlayson, Steven D. Hordley, and Mark S. Drew. Removing shadows from images. In *European Conference on Computer Vision, ECCV'02*, volume 2353 of *Lecture Notes in Computer Science*, pages 823–836, 2002.

[24] David Forsyth and Jean Ponce. *Computer Vision: A Modern Approach.* 2002.

[25] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, September 1991.

[26] William T Freeman, Egon C Pasztor, and Owen T Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[27] William T Freeman and Paul A Viola. Bayesian model of surface perception. In *Neural Information Processing Systems*, April 1998.

[28] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[29] J Friedman and W Stiestzle. Projection pursuit regression. *Journal of the American Statistical Association*, (76):817–823, 1981.

[30] Jerome Friedman. Multivariate adaptative regression splines. *Annals of Statistics*, 19(1):1–141, 1991.

[31] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 38(2):337–374, 2000.

[32] Brian V Funt, Mark S Drew, and Michael Brockington. Recovering shading from color images. In G Sandini, editor, *ECCV-92: Second European Conference on Computer Vision*, pages 124–132. Springer-Verlag, May 1992.

[33] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[34] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statisical Learning; Data Mining, Inference, and Prediction*, chapter 2. Springer, 2001.

[35] Aaron Hertzmann and Steven M. Seitz. Example-based photometric stereo: Shape reconstruction with general, varying brdfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1254–1264, August 2005.

[36] Berthold Klaus Paul Horn. *Robot Vision*, chapter 9. MIT Press, Cambridge, Massachusetts, 1986.

[37] R Jacobs, Michael Jordan, S Nowlan, and G Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[38] D J Jobson, Z Rahman, and G A Woodell. A multiscale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Transactions on Image Processing*, July 1997.

[39] Michael Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6:181–214, 1994.

[40] Ron Kimmel, Michael Elad, Doron Shaked, Renato Keshet, and Irwin Sobel. A variational framework for retinex. *International Journal of Computer Vision*, 52(1):7–23, 2003.

[41] P. D. Kovesi. MATLAB and Octave functions for computer vision and image processing. School of Computer Science & Software Engineering, The University of Western Australia. Available from: <http://www.csse.uwa.edu.au/~pk/research/matlabfns/>.

[42] John Lafferty, Fernando Pereira, and Andrew McCallum. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, 2001.

[43] Edward H Land and John J McCann. Lightness and retinex theory. *Journal of the Optical Society of America*, 61:1–11, 1971.

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

[45] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[46] Y Matsushita, K Nishino, K Ikeuchi, and M Sakauchi. Illumination normalization with time-dependent intrinsic images for video surveillance. In *Proceedings of 2003 Conference on Computer Vision and Pattern Recognition*, volume 1, pages 3–10, 2003.

[47] Yasuyuki Matsushita, Stephen Lin, Sing Bing Kang, and Heung-Yeung Shum. Estimating intrinsic images from image sequences with biased illumination. In *European Conference on Computer Vision (ECCV)*, volume 2, pages 274–286, May 2004.

[48] Andrew McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of the Conferenc on Uncertainty in Artificial Intelligence*, 2003.

[49] Ian Nabney. Netlab neural network software. http://ncrg.aston.ac.uk/netlab/index.php.

[50] E A Nadaraya. On estimating regression. *Theory of Probability and Its Applications*, 9(1):141–142, 1964.

[51] Alan V Oppenheim, Ronald W Shafer, and Jr. Thomas G Stockham. Nonlinear filtering of multiplied and convolved signals. *IEEE Transactions on Audio and Electroacoustics*, 16(3):437–466, September 1968.

[52] J Portilla, V Strela, M Wainwright, and E P Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11):1338–1351, November 2003.

[53] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 1992.

[54] R. Rosales, K. Achan, and B. Frey. Unsupervised image translation. In *ICCV03*, pages 472–478, 2003.

[55] F Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, 1962.

[56] Stefan Roth and Michael Black. Field of experts: A framework for learning image priors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 860–867, 2005.

[57] Dimitris Samaras and Dimitris Metaxas. Incorporating illumination constraints in deformable models for shape from shading and light direction estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):247–264, February 2003.

[58] B Schölkopf, A Smola, R C Williamson, and P L Bartlett. New support vector algorithms. *Neural Computation*, 12:1028–1245, 2000.

[59] B Schölkopf and Alex J Smola. *Learning with Kernels.* MIT Press, 2002.

[60] Eero P Simoncelli and William T Freeman. The steerable pyramid: a flexible architecture for multi-scale derivative computation. In *Proceedings., International Conference on Image Processing, 1995*, volume 3, pages 444–447, October 1995.

[61] Pawan Sinha and Edward H Adelson. Recovering reflectance in a world of painted polyhedra. In *Proceedings of Fourth International Conference on Computer Vision*, pages 156–163, Berlin, 1993. IEEE.

[62] Alex J Smola and Bernard Schölkopf. A tutorial on support vector regression. *Neural Computation*, 14:199–222, 2004.

[63] T P Speed and H T Kiiveri. Gaussian markov distributions over finite graphs. *Annals of Statistics*, 14(1):138–150, March 1986.

[64] R L Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, (6):579–589, 1991.

[65] Inna Stainvas and David Lowe. A generative model for separating illumination and reflectance from images. *Journal of Machine Learning Research*, (4):1499–1519, 2003.

[66] Richard Szeliski. Bayesian modeling of uncertainty in low-level vision. *International Journal of Computer Vision*, 5(3):271–301, 1990.

[67] Marshall F Tappen. Recovering shading and reflectance from a single image. Master's thesis, Massachusetts Institute of Technology, 2002.

[68] Marshall F. Tappen, William T. Freeman, and Edward H. Adelson. Recovering intrinsic images from a single image. In S. Thrun S. Becker and K. Obermayer,

editors, *Advances in Neural Information Processing Systems 15*, pages 1343–1350. MIT Press, Cambridge, MA, 2003.

[69] Marshall F. Tappen, William T. Freeman, and Edward H Adelson. Recovering intrinsic images from a single image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005.

[70] Marshall F Tappen, Bryan C Russell, and William T Freeman. Efficient graphical models for processing images. In *Proceedings of the 2004 IEEE Conference on Computer Vision and Patt ern Recognition*, volume 2, pages 673–680, 2004.

[71] Ben Taskar, Simon Lacoste-Julien, and Michael Jordan. Structured prediction via the extragradient method. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

[72] Michael E Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, June 2001.

[73] Antonio Torralba. Contextual priming for object detection. *International Journal of Computer Vision*, 53(2):169–191, 2003.

[74] Ivor W. Tsang, James T. Kwok, and Kimo T. Lai. Core vector regression for very large regression problems. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 912–919, New York, NY, USA, 2005. ACM Press.

[75] V Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[76] M J Wainwright, E P Simoncelli, and A S Willsky. Random cascades on wavelet trees and their use in modeling and analyzing natural images. *Applied Computational and Harmonic Analysis*, 11(1):89–123, July 2001.

[77] G. S. Watson. Smooth regression analysis. *Sankhya: The Indian Journal of Statistics. Series A*, (26):359–372, 1964.

[78] Yair Weiss. Deriving intrinsic images from image sequences. In *Proceedings International Conference on Computer Vision*, Vancouver, Canada, 2001. IEEE.

[79] Jonathan Yedidia, William T Freeman, and Yair Weiss. Generalized belief propagation. In T K Leen, H G Diettrich, and V Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 689–695, 2001.

[80] Ruo Zhang, Ping-Sing Tsai, James Cryer, and Mubarak Shah. Shape from shading: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, August 1999.

[81] Q Zheng and R Chellapa. Estimation of illuminant direction, albedo, and shape from shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):680–702, July 1991.

[82] Song Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1236–1250, November 1997.

[83] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *The Twentieth International Conference on Machine Learning (ICML-2003)*, 2003.