Contents

List of Figures

Chapter 1

Multiresolution Image Representations

1.1 Introduction

Our review of the organization of neural and behavioral data have led us to several specific hypotheses about how the visual system represents pattern information. Neural evidence suggests that visual information is segregated into a number of different visual streams that are specialized for different tasks. Behavioral evidence suggests that within the streams that are specialized for pattern sensitivity, information is further organized by local orientation and spatial scale and color. All of the evidence we have reviewed to this point suggest that image contrast, rather than image intensity, is the key variable represented by the visual pathways.

We will spend this chapter mainly just thinking about how these and other organizational principles might be relevant to solving various visual tasks. In addition to the intrinsic and practical interest of solving visual problems, finding principled solutions for visual tasks can also be helpful in understanding and interpreting the organization of the human visual pathways.

But, which tasks should we consider? There are many types of visual problems; only some of these have to do with tasks that are essential for human vision. One approach to thinking about visual algorithms, then, is to adopt a general approach to vision, sometimes called *computational vision*, in which we do not restrict our thinking to those problems that are important for human vision. Instead, we open our minds to visual algorithms that may have no biological counterpart.

In this book, however, we will restrict our analysis to the subset of algorithms that is related to human vision. While this is not the broadest possible class, it is a very important one because there are many potential applications for visual algorithms that emulate human performance. For example, suppose a person who wants to search through a database of images to find images with “red vehicles.” To assist the

person, the computer program must have some algorithmic representation related to human vision; after all, the words “red” and “vehicle” are defined by human perception. This is but one example from the set of computer vision algorithms that can serve to augment the human ability to manipulate, analyze, search and create images. These algorithms will be of tremendous importance over the next few decades.

There is a second reason for paying special attention to visual problems related to human vision. Many investigators have argued that studying the visual pathways and visual behavior is an efficient method for discovering novel algorithms for computational vision. The idea is that by studying the specific properties of a successful visual system, we will be led to an understanding of the general design principles of computational vision. This process is analogous to the idea of *reverse-engineering* that is often used to improve instrument design in manufacturing. This view has been suggested by many authors, but Marr (1982) has argued particularly forcefully that biology is a good source of ideas for engineering design. I have never been persuaded by this argument; it seems to me that reverse-engineering methods are most successful when one understands the fundamental principles and only wishes to improve the implementation. It is very difficult to analyze how a system works from the implementation unless one already has a set of general principles as a guide. I think engineering algorithms have done more for understanding the neuroscience than neuroscience has done for engineering algorithms.

Whichever way the benefits flow, from neuroscience to engineering or the other way around, just the presence of a flow is a good reason for the vision scientist and imaging engineer to be familiar with biological, behavioral, and computational issues. In the remainder of this book, we will spend more time engaged in thinking about computational issues related to human vision. In this chapter I will describe ideas and algorithms related to multiresolution image representations. In the following chapters I will describe work on color appearance, motion and objects. Algorithms for all of these topics continue be an important part of vision science and engineering.

1.2 Efficient Image Representations

In this chapter we will consider several different multiresolution representations. Multiresolution representations have been used as part of a variety of visual algorithms ranging from image segmentation to stereo depth and motion (e.g., Burt, 1988; Vetterli, 1992). To unify the introduction of these various representations, however, I will introduce them all by considering how they solve a single engineering problem: efficient storage of image image information.

Efficient image representations are important for systems with finite resources, which is to say all systems. No matter how much computer memory or how many visual neurons we have, we can always perform better computations, transmit more information, or store higher quality images if we use efficient storage algorithms. If we fail to consider efficiency, then we waste resource that could improve performance.

Image compression algorithms transform image data from one representation to a new one that requires less storage space. To evaluate the efficiency of a compression algorithm, we need some way to describe the amount of space required to store an image. The most common way to measure the amount of storage space necessary to encode an image is to count the total number of bytes used to represent the image[1]. Color images acquired from cameras or scanners, or color images that are about to be displayed on a monitor, are represented in terms of the intensities at a set of picture locations, called *pixels*. The color data are represented in three color bands, usually called the red, green and blue bands. We can compute the number of bytes data represented in a single image fairly easily. Suppose we have a modest size image of 512 rows and 512 columns, and that each color band represents intensity using one byte. The image representation within a single color band requires $512 \times 512 \times 3$ bytes of data, or approximately 0.75 Megabytes of data. If we have an image comprising 1024 rows and columns, we will require 3.0 Megabytes to represent the image. In a movie clip, in which we update the image sixty times a second, the numbers grow at an alarming rate; one minute requires 10 Gigabytes of information, and one hour requires 600 Gigabytes.

Notice that color image encoding already uses a significant amount of image compression that is made possible by the special characteristics of human vision. The physical signal consists of light with energy at many wavelengths, i.e., a complete spectral power distribution. The image data, however, does not encode the complete spectral power distribution of the displayed or acquired color signal. The data represent only three color bands, a very compressed representation of the image. The results of the color-matching experiment justifies the compression of information (see Chapter **??**). This part of compression is so well understood, it is rarely mentioned explicitly in discussions of image compression.

In addition to color trichromacy, two main factors permit us to compress images with little loss of quality. First, adjacent pixels in natural images tend to have similar intensity levels. We say that there is considerable *spatial redundancy* in these images. This redundancy is part of the signal, and it may be removed without any loss of information in order to obtain more efficient representations. Second, we know that human spatial resolution to certain spatial patterns is very poor (see Chapters **??** and **??**). People have very poor spatial resolution to short-wavelength light, and only

[1] A bit is a single **b**inary dig**it**, that is, 0 or 1. A byte is 8 bits and represents 256 levels (2^8). A megabyte is 10^6 bytes, while a gigabyte is 10^9 bytes.

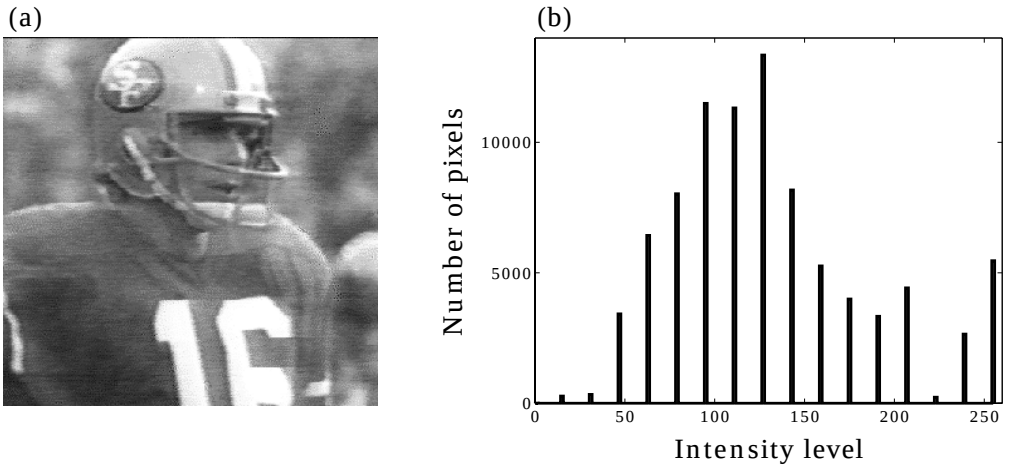

Figure 1.1: *The distribution of image intensities* is an important factor in obtaining an efficient image representation. (a) This image was acquired by a device capable of reproducing 256 gray-levels. But, the image data consists of only 16 different gray levels. (b) A histogram of the gray levels used to code the image shown in (a). Device properties limit the gray-level resolution; they do not enforce a resolution.

limited spatial resolution for colored patterns in general. Representing this information in the stored image is unnecessary because the receiver, that is the visual system, cannot detect it. By eliminating this information, we improve the efficiency of the image representation.

In this chapter we will consider efficient encoding algorithms of monochrome images, spending most of our time on issues of intensity and spatial redundancy. In the next Chapter **??**, which is devoted to color broadly, we will again touch on some of the issues of color image representation.

Intensity Redundancy in Image Data

Suppose that we have an image we wish to encode efficiently, such as the image in Figure 1.1a. The camera I used to acquire this image codes up to 256 different intensity levels (8 bits). You might imagine, therefore, that this is an 8 bit image. To see why that is not the case, let's look at the distribution of pixel levels in the image.

In Figure 1.1b I have plotted the number of points in the image at each of the 256 intensity levels the device can represent. This graph is called the image's *intensity histogram*. The histogram shows that intensity level 128 occurs quite frequently and only a few other levels occur at all.

Although the device used to acquire this image could potentially represent 256 different intensity levels, the image itself does not contain this many levels. We do

not need to represent the data at the device resolution, but only at the intrinsic resolution of the image, which is considerably smaller. Since the image in Figure 1.1a contains 16 levels, not 256, we can represent it using 4 bits per image point rather than the 8 bit resolution the device can manage. This saves us a factor of two in storage space.

The first savings in efficiency is easy to understand; we must not allocate storage space to intensity levels that do not occur in the image. We can refine this idea by taking advantage of the fact that the different intensity levels do not occur with equal frequency. Consider one method to take advantage of the fact that some intensity levels are more likely than others. Assign the one-bit sequence, 1, to code level 128. Encode the other levels using a five bit sequence that starts with a zero, say $0xxxx$, where $xxxx$ is the original four bit code. For example, the level 5 is coded by the five bit sequence 00101. We can unambiguously decode an input stream as follows. When the first bit is a one, then the current value is 128. When the first bit is a zero, read four more bits to define the intensity level at that pixel.

In this image, the intensity level 128 occupies sixty percent of the pixels. Our encoding scheme reduces the space devoted to coding these pixels from 4 bits to 1 bit, saving 3 bits at 60 percent of the locations. The encoding method costs us 1 bit of storage for 40 percent of the pixels. The average savings across the entire image is $0.6 \times 3 - 0.4 \times 1 = 1.2$ bits per pixel. Using these very simple rules, we have reduced our storage requirements to 2.8 bits per pixel.

The example I have provided here is very simple; many more elaborate and efficient algorithms exist for taking advantage of the redundancies in a data set. In general, the more we know about the input distribution the better we can do at designing efficient codes. A great deal of thought has been to the question of designing efficient coding strategies for single images and also for various classes of images such as business documents and natural images. I will not review them here, but you will find references to books on this topic in the bibliography.

Spatial Redundancy in Image Data

Normally, intensity histograms of natural images are not as coarsely discretized as the example in Figure 1.1. In natural images, intensity distributions range across many intensity levels and strategies that rely only on intensity redundancy do not save much storage space.

But there are *spatial redundancies* in natural images, and we can use the same general encoding principles we have been discussing to take advantage of these redundancies as well. Specifically, certain spatial patterns of pixel intensities are much more likely than others. There are various formal and informal ways to convince oneself of the existence of these spatial redundancies. First, consider the

Figure 1.2: *Experimental measurement of spatial redundancy in an image.* The image shows Professor Horace Barlow; random noise is added to the picture. Subjects were asked to adjust the intensity of the noisy pixels to the level they thought must have been present in the original image. Subjects are very accurate at this task, using the information present in nearby pixels. This is an experimental demonstration that people can take advantage of the spatial redundancy in image data (Source: Kersten, 1987).

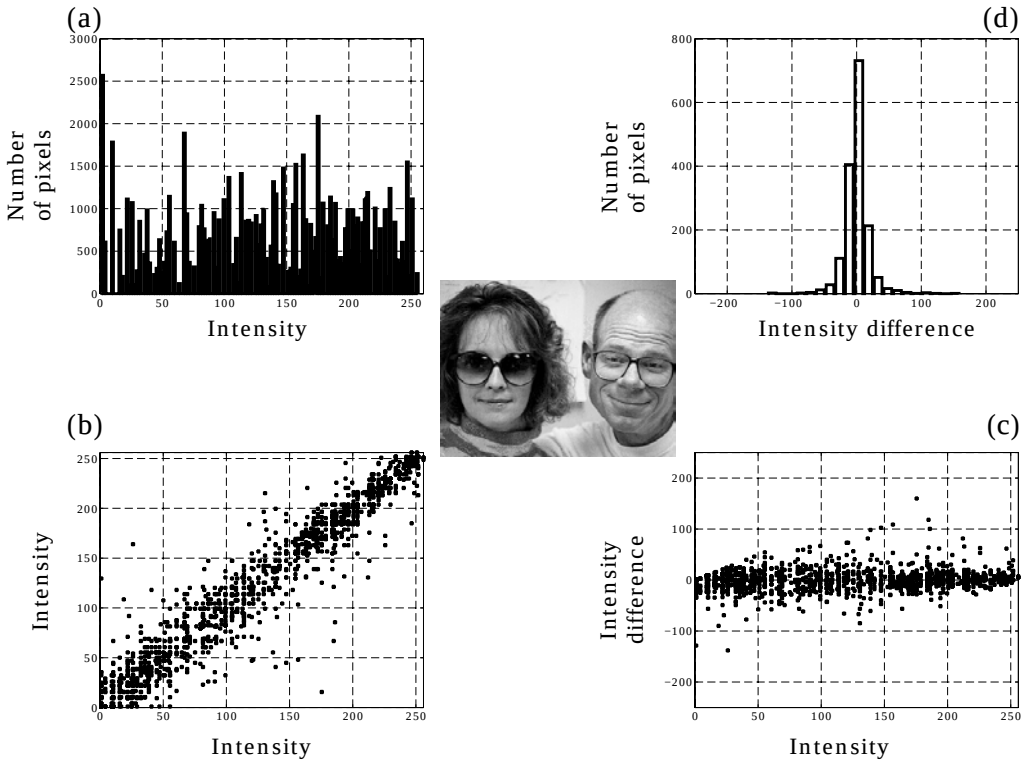

Figure 1.3: *Computational measurement of spatial redundancies in a natural image.* The natural image used for these computations is shown in the middle. (a) The image intensity histogram shows the distribution of image intensities. (b) A correlogram of the intensity at a pixel located at position (x, y) and the intensity of a pixel located at position (x,y+1). (c) A correlogram of the intensity at a pixel located at position (x, y) and the intensity difference between it and the adjacent pixel at $(x, y + 1)$. (d) A histogram of the intensity differences showing that they are concentrated near the zero level.

image in Figure 1.2. This figure contains a picture of Professor Horace Barlow, an eminent visual scientist. A few of the pixel intensities have been set randomly to a new intensity value. Kersten (1987) has shown that naive observers are quite good at adjusting the intensity of these pixels back to their original intensity. With one percent of the pixels deleted, observers correct the pixel intensity to its original value nearly 80 percent of the time. Even with forty percent of the pixels deleted observers set the proper intensity level more than half the time.

Second, we can measure the spatial redundancy in natural images by comparing intensities at neighboring pixels. Figure 1.3a shows the pixel intensities from the image shown in the center of the figure. Measured one at a time, the pixel intensities are distributed across many values and do not contain a great deal of redundancy. Figure 1.3b shows an image *cross-correlogram* that measures the intensity of a pixel, $p(x, y)$, on the horizontal axis and the intensity of its neighboring pixel, $p(x, y + 1)$,

on the vertical axis. Because adjacent pixels tend to have the same intensity level, the points in the cross-correlogram cluster near the identity line. Because the intensity of one pixel tells us a great deal about the probable intensity level of an adjacent pixel, we know that the pixel intensity levels are redundant.

We can improve the efficiency of the image representation by removing this spatial redundancy. One way of removing the redundancy is to transform the image representation. For example, instead of coding the intensities at the two pixels at adjacent locations independently, we can code one pixel level, $p(x, y)$ and the difference between the adjacent pixel values, $p(x, y + 1) - p(x, y)$. This pair of values preserves the image information since we can recover the original from $p(x, y)$ and $p(x, y + 1) - p(x, y)$ by a simple subtraction.

After transforming the data, the number of bits needed to code $p(x, y)$ is unchanged. But the difference, $p(x, y + 1) - p(x, y)$, can fall in a larger range, anywhere between 255 and -255 so that we may need as many as 9 bits to store this value. In principle, requiring an additional bit is worse, but in practice the difference between most adjacent pixels is quite small. This point is illustrated by the cross-correlogram of the transformed values shown in Figure 1.3c. The horizontal axis measures the pixel intensity $p(x, y)$, and the vertical axis measures the difference value, $p(x, y + 1) - p(x, y)$. First, notice that most of the values of the intensity difference cluster near zero. Second, notice that there is virtually no correlation between the transformed values; knowing the value of $p(x, y)$ does not help us know the value of the difference.

To build an efficient representation, we can use the same strategy I outlined in the previous section. We use a short code (say, 5 bits) to encode the small difference values that occur frequently. We use a longer code (say, 10 bits) to encode the rarely occurring large values. Because most of the pixel differences are small, the representation will more efficient.[2]

Decorrelating Transformations

We can divide the image compression strategies I have discussed into two parts. First, we linearly transformed the image intensities to a new representation by a linear transformation. The linear transformation computes $p(x, y)$ and $p(x, y) - p(x, y + 1)$ from $p(x, y)$ and $p(x, y + 1)$. The matrix form of this

[2]We could improve even this coding strategy in many different ways. For example, after the first pair of pixels we never need to encode an absolute pixel level, we can always encode only differences between adjacent pixels. This is called Differential Pulse Code Modulation, or *DPCM*. Or, we could consider the pair of pixels as a vector, calculate the frequency distribution of all possible vectors, and build an efficient code for sending communicating the values of these vectors. This is called Vector Quantization, or *VQ*. All of these methods trade on the fact that natural images are more likely to contain some spatial patterns than others.

transformation is simply

$$\begin{pmatrix} p(x,y) \\ p(x,y+1) - p(x,y) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} p(x,y) \\ p(x,y+1) \end{pmatrix} \tag{1.1}$$

We apply the linear transformation because the correlation of the transformed values is much smaller than the correlation in the original representation.

Second, we find a more efficient representation of the transformed representation. Because we have removed the correlation, in natural images the variation of the transformed values will be smaller than the variation of the original pixel intensities. Hence we will be able to encode the transformed data more efficiently than the original data.

From our example, we can identify a key property of the linear transformation that is essential for achieving efficient coding. The new transformation should convert the data to *decorrelated* values. Values are decorrelated when we gain no advantage in predicting one value from knowing the other. It should seem intuitive that decorrelation is important part of efficiency: if we can predict one value from the another, there is no reason to encode both. Generalizing this idea, if we can predict approximately predict one value from another, we can achieve some efficiencies in our representation. In our example we found that the value $p(x,y)$ is a good predictor of the value $p(x,y+1)$. Hence, it is efficient to predict that $p(x,y+1)$ is equal to $p(x,y)$ and to encode only the error in our prediction. If we have a good predictor (i.e., high correlation) the prediction error will span a smaller range than the data value. Hence, the error can be encoded using fewer bits and we can save storage space.

The transformation in Equation 1.1 does yield a pair of approximately decorrelated values. To make the example simple, I chose a simple linear transformation. We might ask how we might find a decorrelating linear transformation in general. When the set of images we will have to encode is known precisely, then the best linear transformation for lossless image compression can be found using a matrix decomposition called the *singular value decomposition*. The singular value decomposition defines a linear transformation from the data to a new representation with statistically independent values that are concentrated over smaller and smaller ranges. This representation is just what we seek for efficient image encoding. The singular value decomposition is at the heart of principal components analysis and goes by many other names including the Karhunen-Loeve transform and the Hoteling transform. The singular value decomposition may be the most important technique in linear algebra.

In practice, however, the image population is not known precisely. Nor are the image statistics for the set of natural images are known precisely. As a result, the singular value decomposition has no ready application to image compression. As a practical matter, then, selecting a good initial linear transformation remains an

engineering skill acquired by experience with algorithm design.

Lossy Compression

To this point, we have reviewed compression methods that transform the original data with no loss of information. Since we can recover the original image data perfectly from the compressed data, the methods are called *lossless* image compression. Ordinarily, we can achieve savings of a factor of two or three based on lossless compression methods, though this number is strongly image dependent.

If we are willing to tolerate some difference between the original image and the stored copy, then we can develop schemes that save considerably more space. Transformations that lose information are called *lossy* image compression methods. Using only three sensor responses to represent color information is the most successful example of a perceptually lossless encoding. We can not recover the original wavelength representation from the encoded signal. Still, we use this lossy representation because we know from the color-matching experiment that when done perfectly there should be no difference between the perceived image and the original image (see Chapter **??**).

Lossy compression is inappropriate for many types of applications, such as storing bank records. But, some amount of image distortion is acceptable for many applications. It is possible to build lossy image compression algorithms for which the difference between the original and stored image is barely perceptible, and yet the savings in storage space can be as large as a factor of five or ten. Users often judge the efficiency to be worth the image distortion. In cases when the image distortion is not visible, some authors refer to the compression as *perceptually lossless*.

As we reviewed in Chapters **??** and **??**, the human visual system is very sensitive to some patterns and wavelengths but far less sensitive to others. Perceptually lossless encoding methods are designed to account for these differences in human visual sensitivity. These schemes allocate more storage space to represent highly visible patterns and less storage space to represent poorly visible patterns (Watson and Ahumada, 1989; Watson, 1990).

Perceptually lossless image encoding algorithms follow a logic that has much in common with the lossless encoding algorithms. First, the image data are transformed to a new set of values, using a linear transformation. The transformed values are intended to represent *perceptually decorrelated features*. Second, the algorithm allocates different amounts of precision to these transformed values. In this case, the precision allocated to each transformed value depends on the visual salience of the feature the value represents; hence, salient features are allocated more storage space than barely visible features. It is at this point in the process where lossy algorithms differ from lossless algorithms. Lossless algorithms allocate enough

storage so that the transformed values are represented perfectly, yet due to the decorrelation they still achieve some savings. Lossy algorithms do not allocate enough storage to perfectly represent the initial information; the image cannot be reconstructed perfectly from the compressed representation. The lossy algorithm is designed, however, so that the lost information would not have been visible anyway. Thus, the new picture will require less storage and still look like the original image[3].

Perceptually Decorrelated Features

In my overview of perceptually lossless compression algorithms, I used – but did not define – the phrase "perceptually decorrelated features." The notion of a "perceptual feature" is widely used in a very loose way to describe the image properties that are essential for object perception. There is no widely agreed on the specific image features that comprise the perceptual features. In the context of image compression, however, we can use a very useful operational definition for perceptual feature, defined in terms of the linear transformation used to decorrelate the image data. The idea is illustrated in the matrix tableau drawn in Figure 1.4.

Suppose we represent the original image as a list of intensities, one intensity for each pixel in the image. We then apply a linear transformation to the image data, as shown in Figure 1.4a, to yield a new vector of *transform coefficients*. This is the same procedure we applied in our simple example defined by Equation 1.1.

In the transformed representation, each value represents something about the contents of the input image. One way to represent the visual significance of each transformed value is to identify an input image that is represented by that transform coefficient alone. This idea is illustrated in Figure 1.4b, in which a feature is defined as the input image that is represented by a set of transform coefficients that are zero everywhere except at one location. We call this image the *feature* represented by this transform coefficient. Using this operational definition, features are defined with respect to a particular linear transformation.

Next, we must define what we mean by "perceptually decorrelated" image features. We can use Kersten's (1987) experiment to provide an operational definition. In that experiment subjects adjusted the intensity of certain pixels to estimate the intensity level in the original image. Kersten found that observers inferred the intensity levels of individual pixels quite successfully and that observers perceived a great deal of correlation when comparing individual pixels. We can conclude that pixels are a poor choice to serve as decorrelated features.

Now, suppose we perform a variant of Kersten's experiment. Instead of randomly

[3]In practice, lossy and lossless compression are concatenated to compress image data. First a lossy compression algorithm is applied, followed by a lossless algorithm.

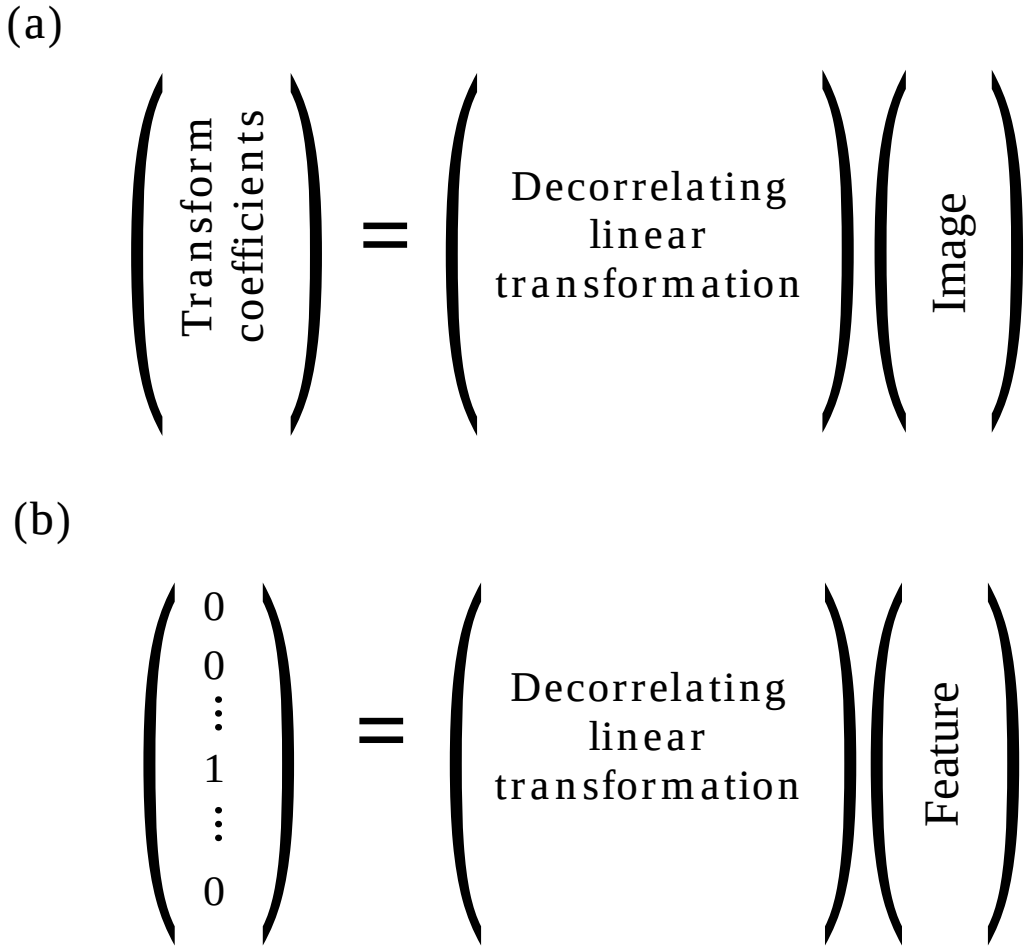

Figure 1.4: *An operational definition of perceptual features.* (a) Image compression usually begins with a linear transformation that maps image intensities into a set of transformed coefficients. (b) An operational definition of an image feature, associated with that linear transformation, is to find the image whose transformation results in a representation that is zero at all values except for one transformation coefficient.

perturbing pixel values in the image, suppose that we perturb the values of the transform coefficients. And, suppose we ask subjects to adjust the transform coefficient levels to reproduce the original image. This experiment is the same as Kersten's task except except we use the transform coefficients, rather than individual pixels, to control image features.

We concluded that individual pixels do *not* represent perceptually decorrelated features because subjects performed very well. We will conclude that a set of transform coefficients represent decorrelated features only if subjects perform badly. When knowing all the transform coefficients but one does not help the subject set the level of an unknown coefficient, we will say the features represented by the transformation are perceptually independent. I am unaware of perceptual studies analogous to Kersten's that test for the perceptual independence of image features; but, in principle, these experiments offer a means of evaluating the independence of features implicit in different compression algorithms.

The important compression step in perceptually lossless algorithms occurs when we use different numbers of bits to represent the transform coefficients. To decide on the number of bits allocated to a transform coefficient, we consider the visual sensitivity of the image feature represented by that coefficient. Because visual sensitivity to some image features is very poor, we can use very few bits to represent these features with very little degradation in the image appearance. This permits us to achieve very compact representations of image data. By saving information at the level of image features, the perceptual distortion of the image can be quite small while the efficiencies are quite large.

This compression strategy depends on the perceptual independence of the image features. If the features are not independent, then the distortions we introduce into one feature may have unwanted side effects on a second feature. If the observer is sensitive to the second feature, we will introduce unwanted distortions. Hence, discovering a set of image features that are perceptually independent is an important part of the design of a perceptually lossless image representation. If distortions of some features have unwanted effects on the appearance of other features, that is if the representation of a pair of features is perceptually correlated, then the linear transformation is not doing its job.

1.3 A Block Transformation: The JPEG-DCT

The Joint Photographic Experts Group (JPEG) committee of the International Standards Organization has defined an image compression algorithm based on a linear transformation called the *Discrete Cosine Transformation* (DCT). Because of the widespread acceptance of this standard, and the existence of hardware to implement the JPEG-DCT compression algorithm is likely to appear on your desk and in your

home within the next few years. The JPEG-DCT compression algorithm has a multiresolution character and bears an imprint from work in visual perception[4].

The JPEG-DCT algorithm uses the DCT to transform the data into a set of perceptually independent features. The image features associated with the DCT are shown in Figure 1.5. The image features are all products of cosinusoids at different spatial frequencies and two orientations. Hence, the independent features implicit in the DCT are loosely analogous to a collection of oriented spatial frequency channels. The features are not the same as the features used to model human vision since the DCT image features are comprised of high and low frequencies, while others contain signals with perpendicular orientations. Still, there is a rough similarity between these features and the oriented spatial frequency organization of models of human multiresolution representations; this is particularly so for the features pictured along the edges and along the diagonal in Figure 1.5, where the image features are organized along lines of increasing spatial frequency and within a single orientation.

The main steps of the JPEG-DCT algorithm are illustrated in Figure 1.6. First, the data in the original image are separated into blocks. The computational steps of the algorithm are applied separately to each block of image data, making the algorithm well-suited to parallel implementation. The image block size is usually 8×8 pixels, though it can be larger. Because the algorithm begins by subdividing the image into blocks, it is one of a group of algorithms called *block coding* algorithms.

Next, the data in each image block are transformed using the linear DCT. The transform coefficients for the image block are shown as an image in Figure 1.6 labeled “Transform coefficients.” In this image white means a large absolute value and black means a low absolute value. The coefficients are represented in the same order as the image features in Figure 1.5; the low spatial frequencies coefficients are in the upper left of the image and the high spatial frequency coefficients are in the lower right.

In the next step, the transform coefficients are quantized. This is one stage of the algorithm where compression is achieved. The quantization is implemented by multiplying each transform coefficient by a scale factor, rounding the result to the nearest integer, and then dividing the result by the scale factor. If the scale factor is small, then the rounding operation has a strong effect and the number of coefficient quantization levels is small. The scalar values for each coefficient are shown in the image marked “Quantization scale factor.” For this example, I chose large scalar values corresponding to the low spatial frequency terms (upper left) and small values for the high spatial frequency terms (lower right).

The quantized coefficients are shown in the next image. Notice that many of the quantized values are zero (black). Because there are so many zero coefficients, the quantized coefficients are very suitable for lossless compression. JPEG-DCT

[4]The DCT is similar to the Fourier Series computation reviewed in Chapters **??** and **??**.

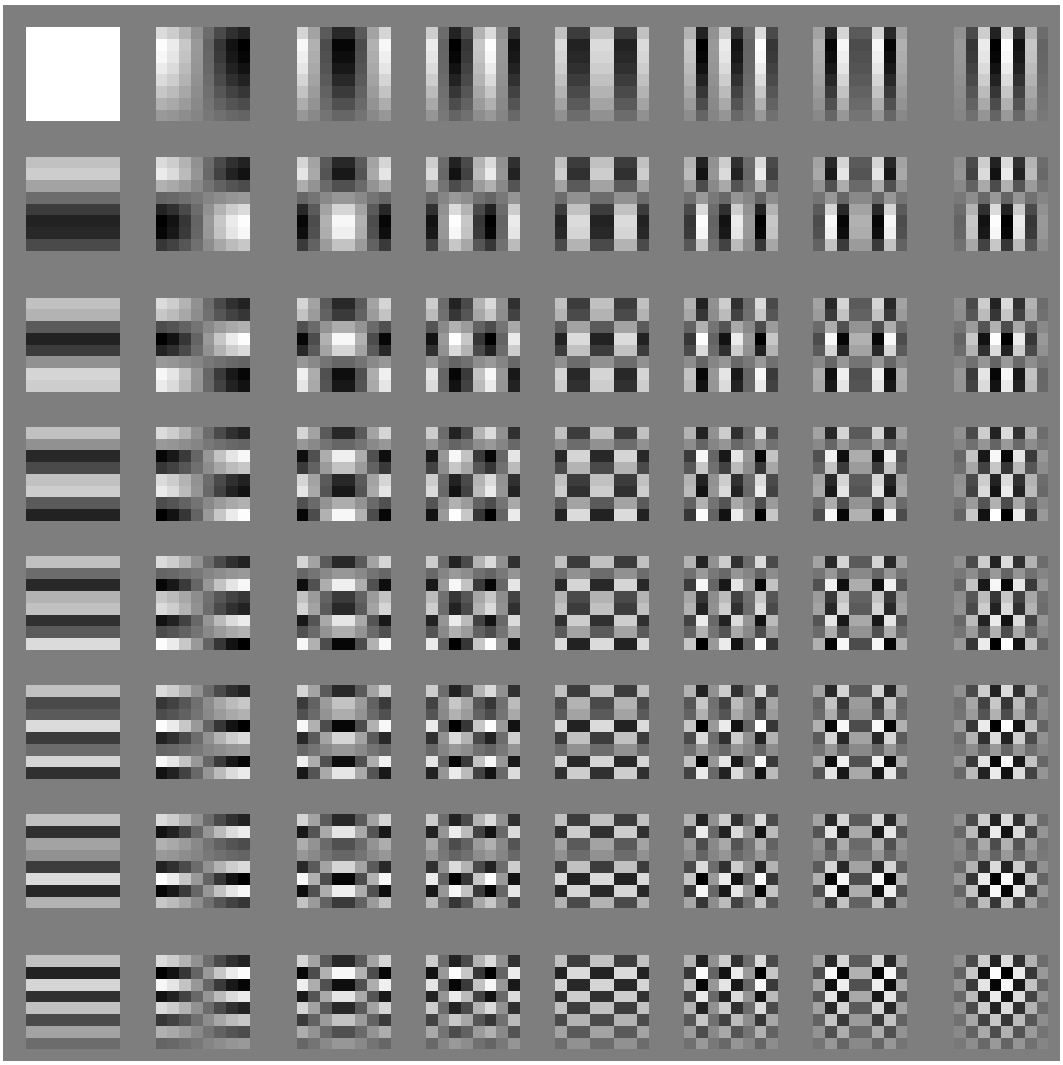

Figure 1.5: *The perceptual features of the DCT.* The DCT features are products of harmonic functions, $\cos(2\pi f_1 j)cos(2\pi f_2 k)$, where j and k refer to position along the horizontal and vertical directions. These functions have both positive and negative values, and they are shown as contrast patterns varying about a constant gray background.

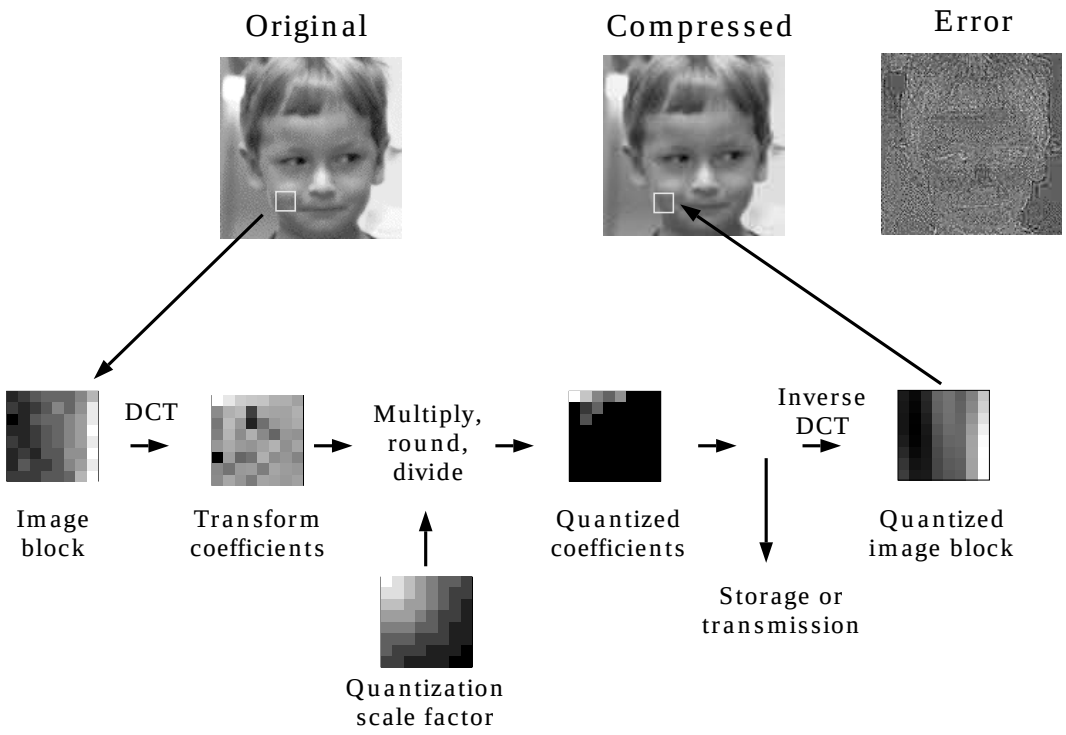

Figure 1.6: *An outline of the JPEG compression algorithm* based on the DCT. The original image is divided into a set of nonoverlapping square blocks, usually 8×8 pixels. The image data are transformed using the DCT to a new set of coefficients. The transform coefficients are quantized using using a simple multiply-round-divide operation. The quantized coefficients are zeroed by this operation, making the image well-suited for efficient lossless compression applied prior to storage or transmission. To reconstruct the image, the quantized coefficients are converted by the inverse DCT, yielding a new image that approximates the original. The error in the reconstruction, i.e., the difference between the original and the reconstruction, consists of mainly high frequency texture. The error is shown as an image on the right.

algorithm includes a lossless compression algorithm applied to the quantized coefficients. This representation is used to store or transmit the image.

To reconstruct an approximation of the original image, we only need to apply the inverse of the DCT to the quantized coefficients. This yields an approximation to the original image. Because of the quantization, the reconstruction will differ from the original somewhat. Since we have removed information mainly about the high spatial frequency components of the image, the difference between the original and the reconstruction is an image comprised of mainly fine texture. The difference image for this example is labeled "Error" in Figure 1.6.

One of the most important limiting factors in compressing images arises from the separation of the original image into distinct blocks for independent processing. Pixels located at the edge of these blocks are reconstructed without any information concerning the intensity level of the pixels that are adjacent, in the next block. One of the most important visual artifacts of the reconstruction, then, is the appearance of distortions at the edges of these blocks, which are commonly called *block artifacts*. These artifacts are visible in the reconstructed image shown in Figure 1.6.

There are two aspects of the JPEG-DCT algorithm that connect it with human vision. First the algorithm uses a roughly multiresolution representation of the image data. One way to see the multiresolution character of the algorithm is to imagine grouping together the coefficients obtained from the separate image blocks. Within each block, there are 64 DCT coefficients corresponding to the 64 image features. By collecting the corresponding transform coefficients from each block, we obtain a measure of the amount of each image feature within the image blocks. Implicitly, then, the DCT coefficients define sixty four images, each describing the contribution of the sixty four image features of the DCT. These implicit images are analogous to the collection of neural images that make up a multiresolution model of spatial vision (Chapter **??**)[5].

Second, the JPEG-DCT algorithm relies on the assumption that quantization in the high spatial frequency coefficients does not alter the quality of the image features coded by low spatial frequency coefficients. If reduced resolution of the high spatial frequencies influences very visible features in the image, then the algorithm will fail. Hence, the assumption that the transform yields perceptually *independent* features is very important to the success of the algorithm.

The independent features in the JPEG-DCT algorithm do not conform perfectly to

[5]There is something that may strike you as odd when you think about the JPEG representation in this way. Notice that each block contributes the same number of coefficients to represent low frequency information as high frequency information. Yet, from the Nyquist sampling theorem (see Chapter **??**), we know that we can represent the low frequency information using many fewer samples than are needed to represent the high frequency information. Why isn't this differential sampling rate is not part of the JPEG representation? The reason is in part due to the block coding, and in part due to the properties of the image features.

the multiresolution organization in models of human spatial vision. High and low frequency components are mixed in some of the features, components at very different orientations are also combined in a single feature. These features are desirable for efficient computation and implementation. In the next section, we will consider multiresolution computations that reflect the character of human vision a bit more closely.

1.4 Image Pyramids

Image pyramids are multiresolution image representations. Their format differs from the JPEG-DCT in several ways, perhaps the two most important being that (a) pyramid algorithms do not segment the image into blocks for processing, and (b) the pyramid multiresolution representation is more similar to the human visual representation than that of the JPEG-DCT. In fact, much of the interest in pyramid methods in image coding is born of the belief that the image pyramid structure is well-matched to the human visual encoding. This sentiment is described nicely in Pavlidis and Tanimoto's paper, one of the first on the topic.

> It is our contention that the key to efficient picture analysis lies in a system's ability, first, to find the relevant parts of the picture quickly, and second, to ignore (not waste time with) irrelevant detail. The retina of the human eye is ... structured so as to see a wide angle in a low-resolution ("high-level") way using peripheral vision, while simultaneously allowing high-resolution, detailed perception by the fovea. [Tanimoto and Pavlidis, page 104].

The linear transformations used by pyramid algorithms have image features comprising periodic patterns at a variety of spatial orientations, much like human multiresolution models. Because the coefficients in the image pyramid represent data that fall mainly in separate spatial frequency bands, it is possible to use different numbers of transform coefficients to represent the different spatial frequency bands. Image pyramids use a small number of transform coefficients to represent the low spatial frequency features and many coefficients to represent the high spatial frequency features. It is this feature, namely that decreasing number of coefficients are used to represent high to low spatial frequency features, that invokes the name pyramid.

The Pyramid Operations: General Theory

Image pyramid construction relies on two fundamental operations that are approximately inverses of one another. The first operation blurs and samples the input. The second operation interpolates the blurred and sampled image to estimate the original. Both operations are linear. I will describe the pyramid operations on one-dimensional signals to simplify notation; none of the principles change when we apply these methods to two-dimensional images. At the end of this section, I will illustrate how to extend the one-dimensional analysis to two-dimensional images.

Suppose we begin with a one-dimensional input vector, g_0, containing n entries. The first basic pyramid operation consists of convolving the input with a smoothing kernel and then sampling the result. The blurring and sampling go together, intuitively, because the result of blurring is to create a smoother version of the original, containing fewer high frequency components. Since blurring removes high frequency information, according to the sampling theorem we can represent the blurred data using fewer samples than the are needed for the original. We do this by sampling the blurred image at every other value.

As we have seen in Chapter **??**, both convolution and sampling are linear operations. Therefore, we can represent each by a matrix multiplication. We represent convolution by the matrix multiplication $B_0 g_0$, where the rows of B_0 contain the convolution kernel. We represent sampling by a rectangular matrix, S_0, whose entries are all zeroes and ones. The combined operation of blurring and sampling is summarized by the basic pyramid matrix $P_0 = S_0 B_0$. Multiplication of the input by P_0 yields a reduced version of the original, $g_1 = P_0 g_0$, containing only half as many entries; a matrix tableau representing the blurring and sampling operator, P_0, is shown in Figure 1.7a.

To create the image pyramid, we repeat the convolution and sampling on each resulting image. The first operation creates a reduced image from the original, g_1. To create the next level of the pyramid, we blur and sample g_1 to create g_2; then, we blur and sample g_2 to create g_3, and so forth. When the input is a one-dimensional signal, each successive level contains half as many sample values as the previous level. When the image is two-dimensional, sampling is applied to both the rows and the columns so that the next level of resolution contains only one-quarter as many sample values as the original. This repeated blurring and sampling is shown in matrix tableau in Figure 1.7b.

The second basic pyramid operation, interpolation, serves as an inverse to the blurring and sampling operation. Blurring and sampling transforms a vector with n entries to a vector with only $n/2$ entries. While this operation does not have an exact inverse, still, we can use g_1 to make an informed guess about g_0. If there is a lot of spatial redundancy in the input signals, our guess about the original image may not be too far off the mark. Interpolation is the process of making an informed guess

(a)

$$P_0 = \left(\quad S_0 \quad \right) \left(\quad B_0 \quad \right)$$

(b)

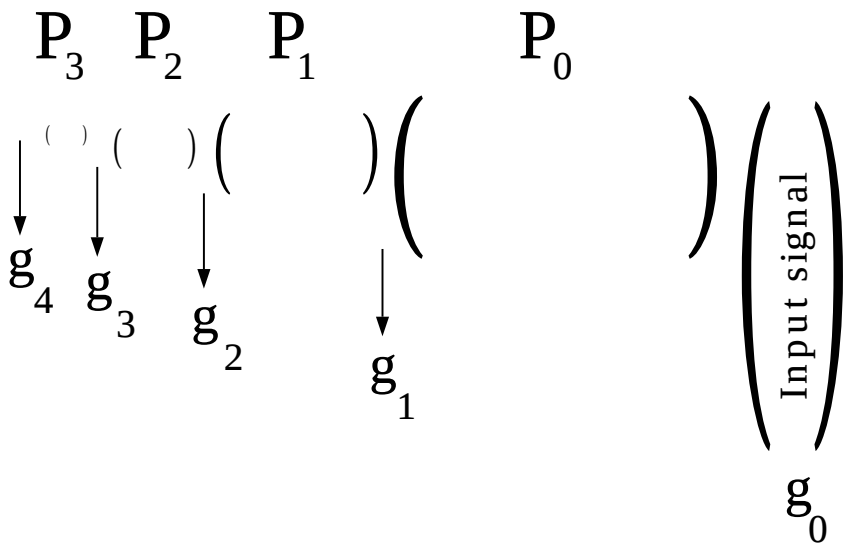

Figure 1.7: *A matrix tableau representation of the one-dimensional pyramid operations.* (a) The basic pyramid operation consists of blurring and then sampling the signal. The blurring operation is a convolution that can be represented by a square matrix whose rows are a convolution kernel. The sampling operation can be represented by a rectangular matrix, consisting of zeros and ones, that pulls out the sample values from the blurred result. (b) To create a series of images at decreasing resolution, we apply the blurring and sampling operation recursively.

about the original image from the reduced image. We interpolate by selecting a matrix, call it E_0, to estimate the input. We choose the *interpolating* matrix E_0 so that in general $E_0 g_1 \approx g_0$.

We can now put together the two basic pyramid operations into a constructive sequence will use several times in this chapter. First, we transform the input by convolution and sampling, $g_1 = P_0 g_0$. We then form our best guess about the original using the interpolation matrix, $\hat{g}_0 = E_0 g_1 = E_0 P_0 g_0$. The estimate $\hat{g}_0$ has the same size as the original image. Finally, to preserve all of the information, we create one final image to save the error. The *error* is the difference between the true signal and the interpolated signal, $e_0 = g_0 - \hat{g}_0$. This completes construction of the first level of the pyramid.

To complete the construction of all levels of the pyramid, we apply the same sequence of operations, but now beginning with first level of the pyramid, g_1. We build a new convolution matrix, B_1; we sample using the matrix, S_1; we build $g_2 = S_1 B_1 g_1$; we interpolate g_2 using a matrix E_1; finally, we form the new error image $g_1 - \hat{g}_1$, where $\hat{g}_1 = E_1 g_2$. To construct the entire pyramid we repeat the process, reducing the number of elements at each step. We stop when we decide that the reduced image, g_n, is small enough so that no further blurring and sampling would be useful.

The pyramid construction procedure defines three sequences of signals; the series of blurred and sampled signals whose size is continually being reduced, the interpolated signals, and the error signals. We can summarize their relationship to one another in a few simple equations. First, the reduced image at the i^{th} level is created by applying the basic pyramid operation to the previous level.

$$g_i = P_{i-1} g_{i-1} \tag{1.2}$$

The estimate of the image $gesti$ is created from the lower resolution representation by the calculation

$$\hat{g}_i = E_{i+1} g_{i+1} \tag{1.3}$$

Finally, the difference between the original and the estimate is the error image,

$$e_i = g_i - \hat{g}_i = g_i - E_i P_i g_i \tag{1.4}$$

Two different sets of these signals preserve the information in the original. One sequence consists of the original input and the sequence of *reduced signals*, g_0, g_1, g_2, ..., g_n. This sequence provides a description of the original signal at lower and lower resolution. It contains all of the data in the original image trivially since the original image is part of the sequence. This image sequence is of interest when we display low resolution versions of the image.

The second sequence consists of the *error signals,* $e_0, e_1, \ldots e_{n-1}, g_n$ (note that g_n is part of this sequence, too). Perhaps surprisingly, this sequence also contains all of the information in the original image. To prove this to yourself, notice that we can build the sequence of images, g_i, from the error signals. The terms g_n and e_{n-1} are sufficient to permit us to construct g_{n-1}; g_{n-1} and e_{n-2} can recover g_{n-2}, and so forth. Ultimately, we use e_0 and g_1 to reconstruct the original, g_0. This image sequence is of interest for image compression (Mallat, 1989).

Pyramids: An Example

Figure 1.8 illustrates the process of constructing a pyramid. The specific calculations used to create this example were suggested by Burt and Adelson (1983), who were perhaps the first to introduce the general notion of an image pyramid to image coding.

The example in Figure 1.8 begins with a one-dimensional squarewave input, g_0. This signal is blurred using a Gaussian convolution kernel and then sampled at every other location; the reduced signal, g_1, is shown below. This process is then repeated to form a sequence of reduced signals. When the convolution kernel is a Gaussian function, the sequence of reduced signals is called the *Gaussian pyramid.*

To interpolate the reduced signal to a higher resolution, Burt and Adelson proposed the following ad hoc procedure. Place the data in g_1 into every other entry of a vector with the same number of entries as g_0. The procedure is called *up-sampling*; it is equivalent to multiplying the vector g_i by the transpose of the sampling matrix, ${S_0}^t$. Then, convolve the up-sampled vector with (nearly) the same Gaussian that was used to reduce the image. The Gaussian used for interpolation differs from the Gaussian used to blur the signals only in that is is multiplied by a factor of 2 to compensate for the fact that the up-sampled vector only has non-zero values at one out of every two locations. In this important example, then, the interpolation matrix is equal to two times the transpose of the convolution-sampling matrix,

$$E_0 = 2{B_0}^t{S_0}^t = 2{P_0}^t. \tag{1.5}$$

The interpolated signal, that is, the estimate of the higher resolution signal, is shown in the middle column of Figure 1.8.

Next, we calculate the error signal, the difference between the estimate and the original. The error signals are shown on the right of Figure 1.8. The sequence of error signals forms the error pyramid. As I described above, we can reconstruct the original g_0 without error from the signals e_0 and g_1. Burt and Adelson called the error signals created by the combination of Gaussian blurring and interpolation functions the *Laplacian pyramid.*

Figure 1.9a shows the result of applying the pyramid process to a two-dimensional

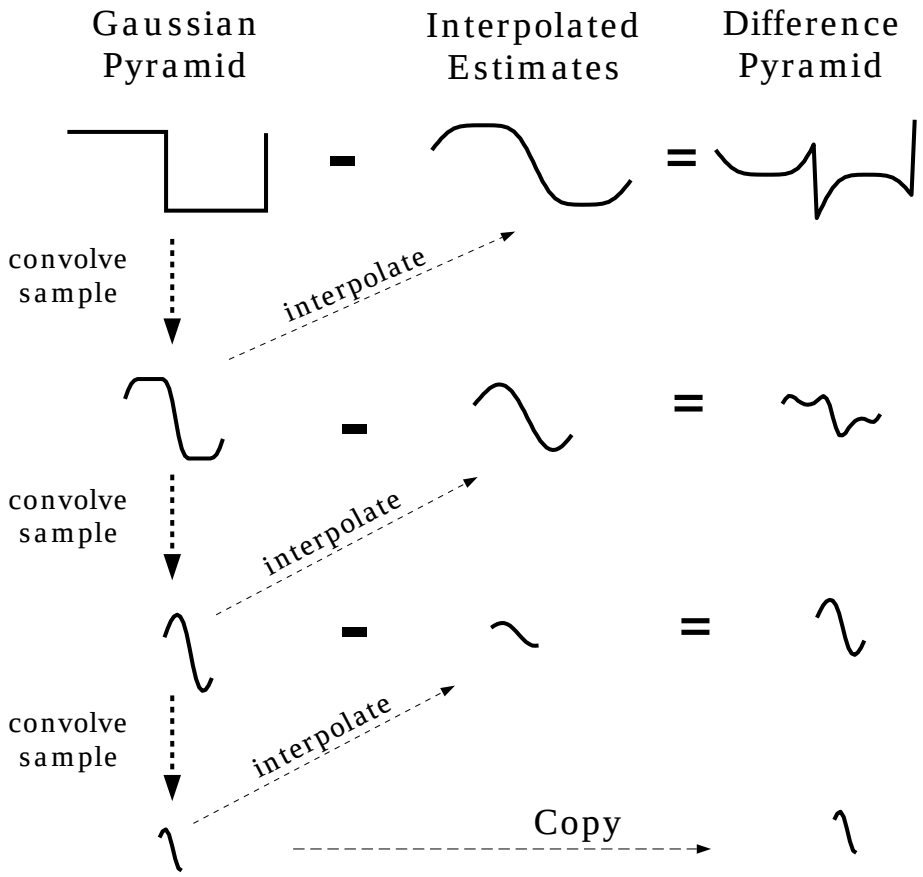

Figure 1.8: *One-dimensional pyramid construction.* The input signal (upper left) is convolved with a Gaussian kernel and the result is sampled. This creates a blurred copy of the signal at lower resolution. An estimate of the original is created by interpolating the low resolution signal, and the difference between the original and the estimate is saved in the error pyramid. The process is repeated, beginning with the blurred copy, thus creating series of copies of the original at decreasing resolution (on the left) and a series error images (on the right). The signal at the lowest resolution level is stored as the final element in the error pyramid.

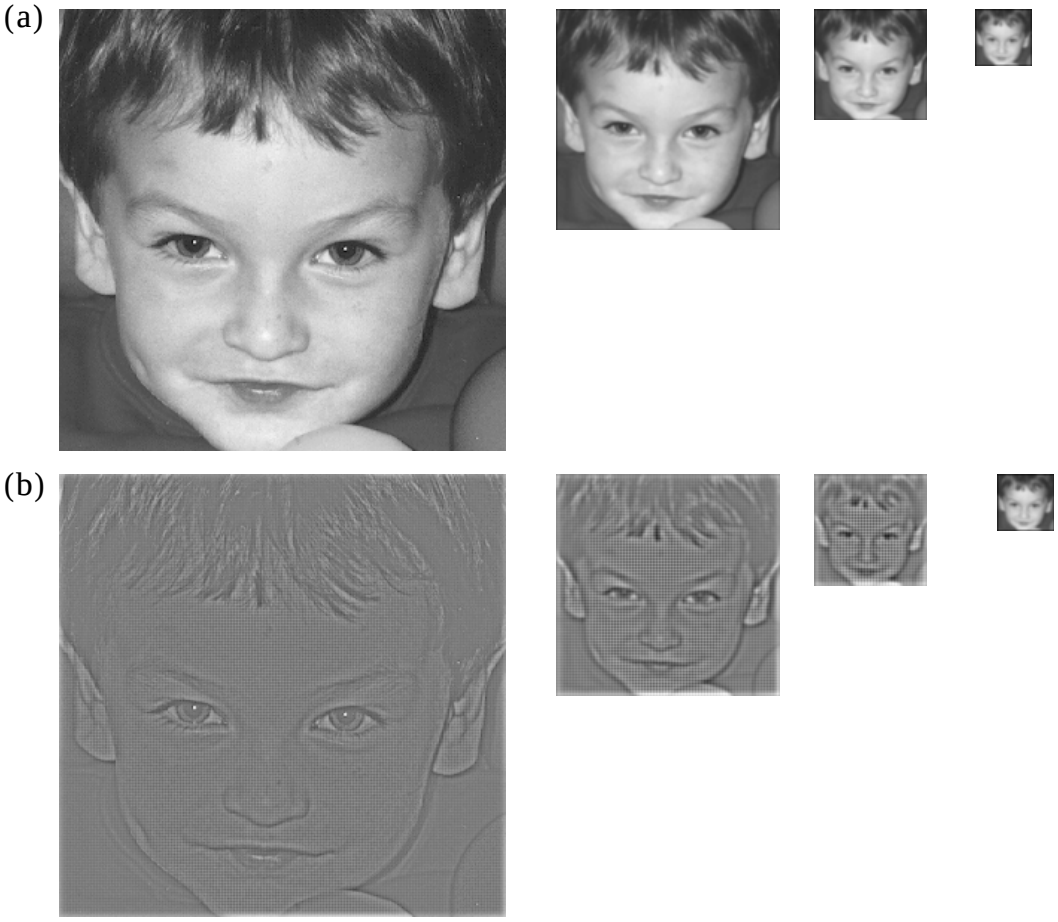

Figure 1.9: *The Gaussian and Laplacian image pyramids.* (a) The series of reduced images that form the Gaussian image pyramid begins with the original image, on the left. This image is blurred by a Gaussian convolution kernel and then sampled to form the image at a lower spatial resolution and size. (b) Each reduced image in the Gaussian pyramid can be used to estimate the image at a higher spatial resolution and size. The difference between the estimate the higher resolution image forms an error image, which in the case of Gaussian filtering is called the Laplacian pyramid. These error images can have positive or negative values, so I have shown them as contrast images in which gray represents zero error, while white and black represent positive and negative error respectively. (After Burt and Adelson, 1983).

signal, in this case an image. The sequence of reduced images forming the Gaussian pyramid is shown on the top, with the original image on the left. These images were created by blurring the original and then representing the new data at one half the sampling rate for both the rows and the columns. Thus, in the two-dimensional case each reduced image contains only one-quarter the number of coefficients as its predecessor[6].

The sequence of error images forming the Laplacian pyramid is shown in Figure 1.9b. Because the interpolation routine uses a smooth Gaussian function to interpolate the lower resolution images, the large errors tend to occur near the edges in the image. And, because the images are mainly smooth (adjacent pixel intensities are correlated) most of the errors are small[7].

Image Compression Using the Error Pyramid

From the point of view of image compression, the sequence of images in the Gaussian pyramid is not very interesting because that sequence contains the original. Rather than the use the entire sequence, we might as well just code the original. The sequence of images in the Laplacian pyramid, however, is interesting for two reasons.

First, the information represented in the Laplacian pyramid varies systematically as we descend in resolution. At the highest levels, containing the most transform coefficients, the Laplacian pyramid represents the fine spatial detail in the image. At the lowest levels, containing the fewest transform coefficients, the Laplacian pyramid represents low spatial resolution information. Intuitively, this is so because the error image is the difference between the original, which contains all of the fine detail, and an estimate of the original based on a slightly blurred copy. The difference between the original and an estimate from a blurred copy represents image information in the resolution band between the two levels. Thus, the Laplacian pyramid is a multiresolution representation of the original image.

Second, the values of the transform coefficients in the error images are distributed over a much smaller range than the pixel intensities in the original image. Figure 1.10a shows intensity histograms of pixels in the first three elements of the Gaussian pyramid. These intensity histograms are broad and not well-suited to the compression methods we reviewed earlier in this chapter. Figure 1.10b shows histograms of the pixel intensities in the Laplacian pyramid. The transform coefficients tend to cluster near zero and thus they can be represented very

[6]Therefore, in the estimation phase we multiply the interpolation matrix by a factor of 4, not 2, i.e., $E_0 = 4{P_0}^t$.

[7]In order to display the error images, which negative coefficients, the image intensities are scaled so that black is a negative value, medium gray is zero, and white is positive.

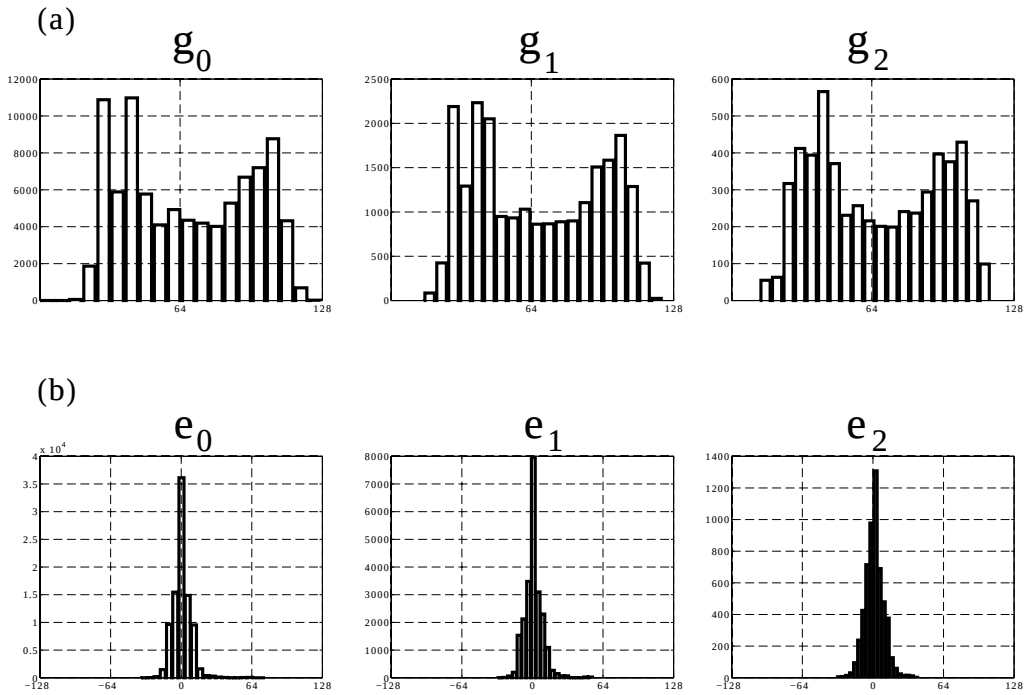

Figure 1.10: *Histograms of the Gaussian and Laplacian pyramids.* (a) The separate panels show the intensity histograms at each level of the Gaussian pyramid. The intensities are distributed across a wide range of values, making the intensities difficult to code efficiently. (b) The Laplacian pyramid coefficients are distributed over a modest range near zero and can be coded efficiently.

efficiently. The reduced range of transform coefficient values in the Laplacian pyramid arises because of the spatial correlation in natural images. The spatial correlation permits us to do fairly well in approximating images using smooth interpolation. When the approximations are close, the errors are small, and they can be coded efficiently.

There is one obvious problem with using the images in the Laplacian pyramid as an efficient image representation: there are more coefficients in the error pyramid than pixels in the original image. When building an error pyramid from two-dimensional images, for example, we sample every other row and every other column. This forms a sequence of error images equal to 1, $1/4$, $1/16$ the size of the original; hence, the error pyramid contain 1.33 times as many coefficients as the original (see Figure 1.9). Because of the excess of coefficients, the error image representation is called *overcomplete*. If one is interested in image compression, overcomplete representations seem to be a step in the wrong direction.

Burt and Adelson (1983) point out, however, that there is an important fact pertaining to human vision that reduces the significance of the overcompleteness:

The vast majority of the the transform coefficients represent information in the highest spatial frequency bands where people have poor visual resolution. Therefore, we can quantize these elements very severely without much loss in image quality. Quantization is the key step in image compression, so that having most of the transform coefficients represent information that can be heavily quantized is an advantage.

The ability to quantize severely many of the transform coefficients with little perceptual loss, coupled with the reduced variance of the transform coefficients, make the Laplacian pyramid representation practical for image compression. Computing the pyramid can be more complex than the DCT, depending on the block size, but special purpose hardware has been created for doing the computation efficiently. The pyramid representation performs about as well or slightly better the JPEG computation based on the DCT. It is also applicable to other visual applications, as we will discuss later (Burt, 1988).

1.5 QMFs and Orthogonal Wavelets

Pyramid representations using a Gaussian convolution kernel have many useful features; but, they also have several imperfections. By examining the problematic features of Gaussian and Laplacian pyramids, we will see the rationale for using a different convolution kernel, *quadrature mirror filters* (QMFs), in creating image pyramids.

The first inelegant feature of the Gaussian and Laplacian pyramids is an inconsistency in the blurring and sampling operation. Suppose we had begun our analysis with the estimated image, $\hat{g}_0$, rather than g_0. From the pyramid construction point of view, the estimate should be equivalent to the original image. It seems reasonable to expect, therefore, that the reduced image derived from $\hat{g}_0$ should be the same as the reduced image derived from g_0. We can express this condition as an equation,

$$g_1 = P_0(2{P_0}^t)g_1 = P_0\hat{g}_0. \tag{1.6}$$

Equation 1.6 implies that the square matrix $P_0(2{P_0}^t)$ must be the identity matrix. This implies that the columns of the matrix, P_0 should be *orthogonal* to one another[8]. This is not a property of the Gaussian and Laplacian pyramid.

A second inelegant feature of the Gaussian and Laplacian pyramid is that the representation is overcomplete, i.e., there are more transform coefficients than there

[8]Orthogonality is defined in Chapter **??** and the Appendix. Two vectors are orthogonal when $a^t b = 0$.

are pixels in the original image. The increase in the transform coefficients can be traced to the fact that we represent an image g_i with N_i pixels by a reduced signal and an error signal that contain more than N_i coefficients. For example, we represent the information in the i^{th} level of the pyramid using the reduced image g_{i+1} and the error image e_i.

$$g_i = (2{P_i}^t)g_{i+1} + e_i \tag{1.7}$$

In the one-dimensional case, the error image, e_i, contains N_i transform coefficients. The reduced signal, g_{i+1}, contains $N_i/2$ coefficients. To create an efficient representation, we must represent g_i using N_i transform coefficients, not $1.5N_i$ coefficients as in the Gaussian pyramid.

The error signal and the interpolated signal are intended to code different components of the original input; the interpolated vector $\hat{g}_i = (2{P_i}^t)g_{i+1}$ codes a low resolution version of the original, and e_i codes the higher frequency terms left out by the low resolution version. To improve the efficiency of the representation, we might require that the two terms code completely different types of information about the input. One way to interpret the phrase "completely different" is to require that the two vectors be orthogonal, that is,

$$0 = {e_i}^t\hat{g}_i. \tag{1.8}$$

If we require that $\hat{g}_i$ and e_i to be orthogonal, we can obtain significant efficiencies in our representation. By definition, we know that the interpolated image $\hat{g}_i$ is the weighted sum of the columns of ${P_i}^t$. If we the error e_i image is orthogonal to the interpolated image, then the error image must be the weighted sum of a set of column vectors that are all orthogonal to the columns of ${P_i}^t$. In the (one-dimensional) Gaussian pyramid construction, ${P_i}^t$ has $N_i/2$ columns. From basic linear algebra, we know that there are $(1/2)N_i$ vectors perpendicular to the columns of ${P_i}^t$. Hence, if $\hat{g}_i$ is orthogonal to e_i, we can describe both of these images using only N_i transform coefficients, and the representation will no longer overcomplete.

But, what conditions must be met to insure that e_i and $\hat{g}_i$ are orthogonal? By substituting Equations 1.2, 1.3 and 1.4 into Equation 1.8 we have

$$\begin{aligned} 0 = {e_i}^t\hat{g}_i &= (g_i^t(2{P_i}^t)P_i)((2{P_i}^t)P_ig_i - g_i) \\ &= [g_i^t(2{P_i}^t)(P_i(2{P_i}^t))P_ig_i] - [g_i^t(2{P_i}^t)P_ig_i]. \end{aligned} \tag{1.9}$$

If the rows of P_i are an orthogonal set, then by appropriate scaling we can arrange it so that $P_i(2{P_i}^t)$ is equal to the identity matrix. In that case, the final term in Equation 1.9 simplifies and we have

$$ {e_i}^t\hat{g}_i = [{g_i}^t(2{P_i}^t)P_ig_i] - [{g_i}^t(2{P_i}^t)P_ig_i] = 0, \tag{1.10}$$

thus guaranteeing that the error signal and the interpolated estimate will be orthogonal to one another. For the second time, then, we find that the orthogonality of the rows of the pyramid matrix is a useful property.

We can summarize where we stand as follows. The basic pyramid operation has several desirable features. The rows within each level of the pyramid matrices are shifted copies of one another, simplifying the calculation to nearly a convolution; the pyramid operation represents information at different resolutions, paralleling human multiresolution representations; the rows of the pyramid matrices are localized in space, as are receptive fields, yet they are not sharply localized as the blocks used in the JPEG-DCT algorithm. Finally, from our criticisms of the error pyramid, we have added a new property we would like to have: The rows of each pyramid matrix should be an orthogonal set.

We have accumulated an extensive set of properties we would like the pyramid matrices, P_i, to satisfy. Now, one can have a wish list, but there is no guarantee that there exist any functions that satisfy all our requirements. The most difficult pair of constraints to satisfy is the combination of orthogonality and localization. For example, if we look at convolution operators alone, there are no convolutions that are simultaneously orthogonal and localized in space.

Interestingly there exists a class of discrete-valued functions, called *quadrature mirror filters*, that satisfy all of the properties on our wish list (Esteban and Galand, 1977; Simoncelli and Adelson, 1990; Vetterli; 1988). The quadrature mirror filter pair splits the input signal into two orthogonal components. One of the filters defines a convolution kernel that we use to blur the original image and obtain the reduced image. The second filter is orthogonal to the first and can be used to calculate an efficient representation of the error signal. Hence, the quadrature mirror filter pair splits the original signal into coefficients that define of the two orthogonal terms, $\hat{g}_i$ and e_i; Each set of coefficients has only $n/2$ terms, so the new representation is an efficient pyramid representation. Figure 1.11 shows an example of a pair of quadrature mirror filters. The function shown in Figure 1.11a is the convolution kernel that is used to create the reduced images, g_i. The function in Figure 1.11b is the convolution kernel needed to calculate the transform coefficients in the error pyramid directly. When the theory of these filters is developed for continuous, rather than discrete, functions the convolution kernels are called *orthogonal wavelets* (Debauchie, 1992).

The discovery of quadrature mirror filters and wavelets was a bit of a surprise. It is known that there are no nontrivial convolution kernels that are orthogonal; i.e., no convolution matrix, B_i, satisfies the property that $B_i B_i^t = I$. Hence it was surprising to discover that convolution kernels do exist for the pyramid operation, which relies so heavily on convolution, can satisfy $P_i P_i^t = I$.

The quadrature mirror filter and orthogonal wavelet representations have many

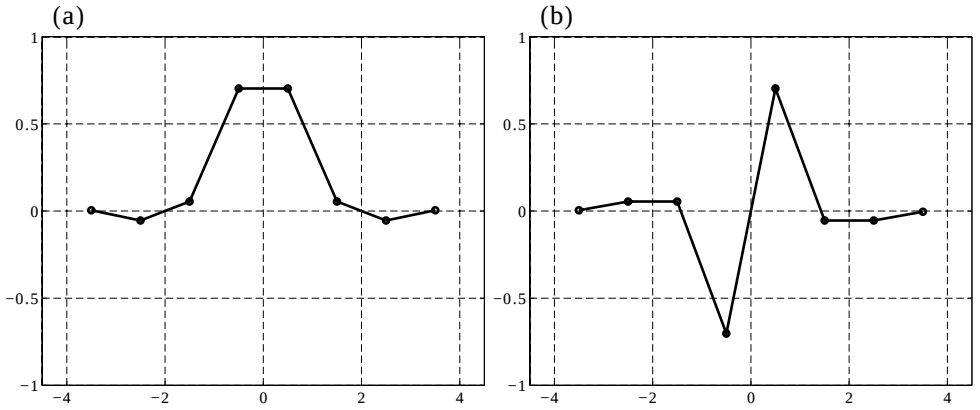

Figure 1.11: *A quadrature mirror filter pair.* One can use these two functions as convolution kernels to construct a pyramid. Convolution with the kernel in (a) followed by sampling produces the transform coefficients in the set of reduced signals. Transformation by the kernel in (b) followed by sampling yields the transform coefficients of the error pyramid (Source: Simoncelli, 1988).

fascinating properties and are an interesting area of mathematical study. They may have significant implications for compression because they remove the problem of having an overcomplete representation. But, it is not obvious that once quantization and correlation are accounted for that the savings in the number of coefficients will prove to be significant. For now, the design and evaluation of quadrature mirror filters remains an active area of research in pyramid coding of image data.

1.6 Applications of multiresolution representations

The statistical properties of natural images make multiresolution representations efficient. Were efficiency a primary concern, the visual pathways might well have evolved to use the multiresolution format. But, there is no compelling reason to think that the human visual system, with hundreds of millions of cortical neurons available to code the outputs of tens of thousands of cone photoreceptors, was subject to very strong evolutionary pressure to achieve efficient image representations. Understanding the neural multiresolution representation may be helpful when we design image compression algorithms; but, it is unlikely that neural multiresolution representations arose to serve the goal of image compression alone.

If multiresolution representations are present in the visual pathways, what other purpose might they serve? In this section, I will speculate about how multiresolution representations may be a helpful component of several visual algorithms.

Image Blending

Imagine blending refers to methods for smoothly connecting several adjacent or overlapping images of a scene into a larger photomosaic (Milgram, 1975; Carlbom, 1994). There are several different reasons why we might study the problem of joining together several pieces of an image. For example, in practical imaging applications we may find that a camera's field of view may be too small to capture the entire region of interest. In this case we would like to blend several overlapping pictures to form a complete image.

The human visual system also needs to blend images. As we saw in the early chapters of this volume, spatial acuity is very uneven across the retina. Our best visual acuity is in the fovea, and primate visual systems rely heavily on eye-movements to obtain multiple images of the scene. To form a good high acuity representation of more than the central few degrees, we must gather images from a sequence of overlapping eye fixations. How can the overlapping images acquired through a series of eye movements be joined together into the single, high resolution representation that we perceive?

Burt and Adelson (1983b) showed that multiresolution image representations offer a useful framework for blending images together. They describe some fun examples based on the pyramid representation.

We can see some of the advantages of a multiresolution image blending by comparing the method with a single resolution blend. So, let's begin by defining a simple method of joining the two pictures, based on a single resolution representation. Suppose we decide to join a picture on the left $L(x, y)$ and a picture on the right $R(x, y)$. We will blend the images by mixing their intensity values near the border where the join. A formal rule for to blend the image data must specify how to combine the data from the two images. We do this using a blending function, call it $b(x, y)$, whose values vary between 0 and 1.0. To construct our single-resolution blending algorithm we form a mixture image from the weighted average

$$M(x, y) = b(x, y)L(x, y) + (1 - b(x, y))R(x, y). \tag{1.11}$$

Consider the performance of this type of single resolution blend on an a pair of simulated astronomical images in Figure 1.12. Each of these images contain 512 rows and columns. The two images were built to simulate the appearance of a starry sky. The images contain three distortions to illustrate some of the advantages of multiresolution methods for blending images.

First, the images contain two kinds of objects (stars and clouds) whose spatial structure places them in rather different spatial frequency bands. Second, the images

have different mean levels (the image on the top right being dimmer than the one on the top left). Third, the images are slightly shifted in the vertical direction as if there was some small jitter in the camera position at the time of acquiring the pictures.

Because the images are divided along a vertical line, we need to concern ourselves only with the variation with x and join the images the same way across each row.

The most trivial, and rather ineffective, way of joining the two images is shown in panel (c). In this case the two images are simply divided in half and joined at the dividing line. Simply abutting the two images is equivalent to choosing a function $s(x, y)$ equal to

$$b(x, y) = \begin{cases} 1 & \text{if } x < m \\ 0 & \text{otherwise} \end{cases} \tag{1.12}$$

where m is the midpoint of the image, 256 in this case. This smoothing function leads to a strong artifact at the midpoint because of the difference in mean gray level.

We might use a less drastic blending function for $b(x, y)$. For example, we might choose as function that varied as a linear ramp over some central width of the image.

$$b(x, y) = \begin{cases} 1 & \text{if } x < m - w \\ 1 - \frac{x-m-w}{2w} & \text{if } m - w \leq x \leq m + w \\ 0 & \text{otherwise} \end{cases} \tag{1.13}$$

Using a ramp to join the images blurs the image at the edge, as illustrated in panels (d) and (e) of figure 1.12. In panel (d) the width parameter of the linear ramp, w, is fairly small. When the width is small the edge artifact remains visible. As the width is broadened, the edge artifact is removed (panel (e)) and elements from both images contribute to the image in the central region. At this point the vertical shift between the two images becomes apparent. If you look carefully in the central region, you will see double stars shifted vertically one above the other. Image details that are much smaller than the width of the ramp appear in the blended image and they appear at their shifted locations. The stars are small compared to the width of the linear ramp, so the blended image contains the an artifact due to the shift in the image details.

Multiresolution representations provide a natural, way for combining the two images that avoid some of these artifacts. We can state the multiresolution blending method as an algorithm.

1. Form the pyramid of error images for L and R.

2. Within each level of the pyramid, average the error images with a blend function $b(x, y)$. A simple ramp function, as in Equation 1.13 with $w = 1$, will do as the blend function.

(this is unreadable now ...just a placeholder)

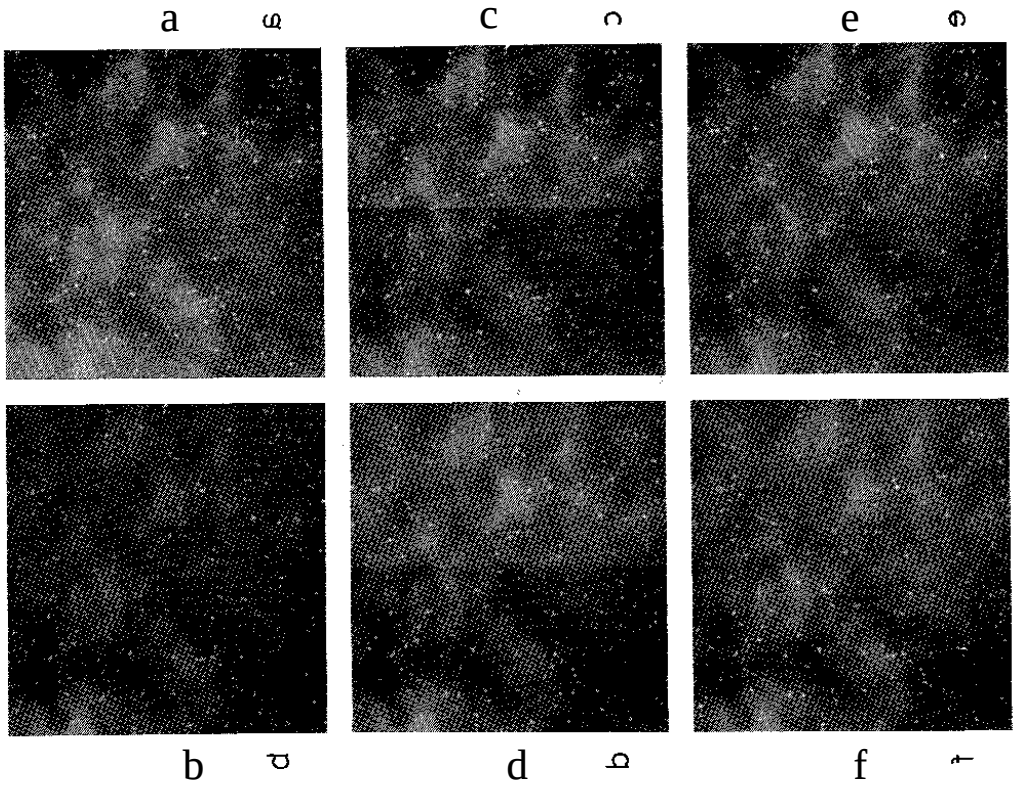

Figure 1.12: *A comparison of single-resolution and multiresolution image blending methods.* The images in (a) and (b) have a slightly different mean and are translated vertically. Abutting the right and left halves of the images is shown in (c). Spatial averaging over a small distance across the image boundary is shown in (d). Spatial averaging over a large distance across the image boundary is shown in (e). The multiresolution blend from Burt and Adelson is shown in (f). (Source: Burt and Adelson, 1983).

3. Compute the new image by reconstructing the image from the blended pyramid of error images.

The image in panel (f) of Figure 1.12 contains the results of applying the multiresolution blend to the images. The multiresolution algorithm avoids the previous artifacts because by averaging the two error pyramids, two images combine over different spatial regions in each of the resolution bands. Data from the low resolution level is combined over a wide spatial region of the image, while data from the high resolution levels are combined over a narrow spatial region of the image.

By combining low frequency information over large spatial regions, we remove the edge artifact. By combining high frequency information over narrow spatial regions, we reduce the artifactual doubling of the star images to a much narrower spatial region.

Burt and Adelson (1983b) also describe a method of blending images with different shapes. Figure 1.13 illustrates one of their amusing images. They combined the woman's eye taken from panel (a) and the hand, taken from panel (b) into a single image shown in panel (d). The method for combining images with different shapes is quite similar to the algorithm I described above. Again, we begin by forming the error images e_i for each of the two images. For the complex region, however, we must a method to define a blend function $s_i(x, y)$, appropriate for combining the data at each resolution of the pyramid over these different shapes. Burt and Adelson have a nifty solution to this problem. Build an overlay image that defines the location where second image is to be placed over the first, as in panel (c) of figure 1.13. Build the sequence of pyramid of reduced images, g_i, corresponding to the overlay image. Use the elements of the image sequence g_i to define the blend functions for combining the images at resolution e_i.

Progressive Image Transmission

For many devices, transmitting an image from its stored representation to the viewer can take a noticeable amount of time. And, in some of these cases, transmission delays may hamper our ability to perform the task. Suppose we are scanning through a database for suitable pictures to use in a drawing, or we are checking a directory to find the name of the person who recently waved hello. We may have to look through many pictures before finding a suitable one. If there is a considerable delay before we see each picture, the tasks become onerous; people just won't do them.

Multiresolution image representations are natural candidates to improve the rate of image transmission and display. The reconstruction of an image from its

Figure 1.13: *Image blending of regions with arbitrary shape.* To create a multiresolution blend of the images of the eye and hand, we must define a blending function for each level of the pyramid. The blending function can be created by building the Gaussian pyramid representation of the region where the image of the eye will be inserted. The different levels of the Gaussian pyramid can be used as the blending functions to combine the error pyramids of the two images (Source: Burt and Adelson, 1983).

multi-resolution image proceeds through several stages. The representation stores the error images e_i and the lowest reduced image g_n. We reconstruct the original by computing a set of reduced images, g_i. These reduced images are rough approximations of the original, at reduced resolution. They are represented by fewer bits than the original image, so they can be transmitted and displayed much more quickly. We can make these low resolution images available for the observer to see during the reconstruction process. If the observer is convinced that this image is not worth any more time, then he or she can abort the reconstruction and go on to the next image. This offers the observer a way to save considerable time.

We can expand on this use of multiresolution representations by allowing the observer to request a low resolution reconstruction, say at level g_i, rather than a full representation at level g_0. The observer can choose a few of the low resolutions for viewing at high resolution. Multiresolution representations are efficient because there is little wasted computation. The pyramid reconstruction method permits us to use the work invested in reconstructing the low resolution image as we to reconstruct the original at full resolution.

Th engineering issues that arise in progressive image transmission may be relevant to the internal workings of the human visual system. When we call up an image from memory, or sort through a list of recalled images, we may wish to image low resolution images rather than reconstruct each image in great detail. If the images are stored using a multi-resolution format, our ability to search efficiently through our memory for images may be enhanced.

Threshold and Recognition

Image compression methods link *visual sensitivity* measurements to an engineering application. This makes sense because threshold sensitivity plays a role in image compression; perceptually lossless compression methods, by definition, tolerate threshold level differences between the reconstructed image and the original.

For the applications apart from compression, however, sensitivity is not the key psychological measure. Since low resolution images do not look the same as the high resolution images, sensitivity to differences is not the key behavioral measure. To understand when progressive image transmissions methods work well, or which low resolution version is the best approximation to a high resolution version of an image, we need to be informed about which multiresolution representations permit people to *recognize* quickly or *search* for an item in a large collection of low resolution images quickly. Just as the design of multiresolution image compression methods requires knowing visual sensitivity to different spatial frequency bands, so too multiresolution methods for progressive image transmission requires knowing how important different resolution bands will be for expressing the information in an

image.

As we study these applications, we will learn about new properties of human vision. To emphasize some of the interesting properties that can arise, I will end this section by reviewing a perceptual study by Bruner and Potter (1964). This study illustrates some of the counter-intuitive properties we may discover as we move from threshold to recognition studies.

Bruner and Potter (1964) studied subjects ability to recognize common objects from low resolution images. Their subjects were shown objects using slides projected onto a screen. In 1964 low resolution images were created much more quickly and easily than today; rather than requiring expensive computers and digital framebuffers low resolution images were created by blurring the focus knob on the projector.

Bruner and Potter compared subjects' ability to recognize images in a few ways. I want to abstract from their results two key observations[9].

Figure 1.14 illustrates three different measurement conditions. Observers in one group saw the image develop from very blurry to only fairly blurry over a two minute period. At the end of this period the subjects were asked to identify the object in the image. They were correct on about a quarter of the images. Observers in a second group only began viewing the image after after 87 seconds. They first saw the image at a somewhat higher resolution, but then they could watch the image develop for only about a half minute. The difference between the second group and the first group, therefore, was whether they saw the image in a very blurry state, during the first 90 seconds. The second group of observers performed substantially better, recognizing the object 44 percent of the time rather than 25 percent. Surprisingly, the initial 90 seconds of viewing the image come into focus made the recognition task more difficult. A third group was also run. This group only saw the image come into focus during the last 13 seconds. The third group did not see the first 107 seconds as the image came into focus. This group also recognized the images correctly about 43 percent of the time.

Seeing these images come into focus slowly made it harder for the observers to recognize the image contents. Observers who saw the images come into focus over a long period of time formulated hypotheses as to the image contents. These hypotheses were often wrong and ultimately interfered with their recognition judgments.

Bruner and Potter illustrated the same phenomenon a different way. They showed one group of observers the image sequence coming into focus and a second group the same image sequence going out of focus. These are the same set of images, shown for the same amount of time. The difference between the stimuli is the

[9]There are a number of important methodological features of the study I will not repeat here, and I encourage the reader to return to the primary sources to understand more about the design of these experiments.

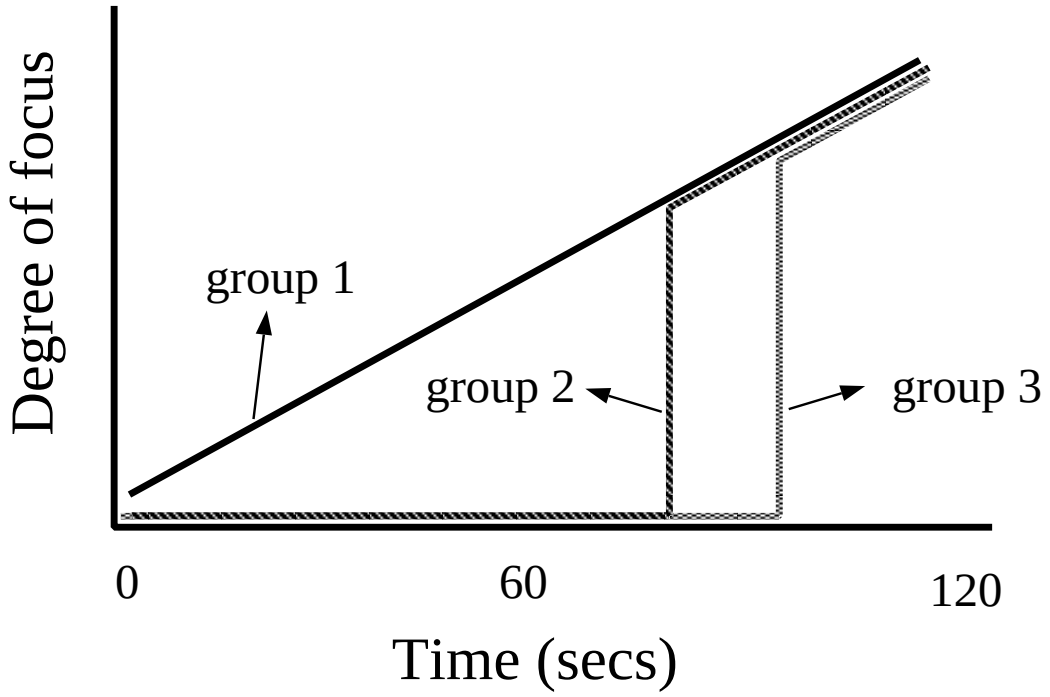

Figure 1.14: *The experimental viewing conditions used by Bruner and Potter in their recognition experiment.* One group saw the picture come into slowly and continuously over a period of 122 seconds. A second group saw nothing for 87 seconds and then watched the remainder of the image come into focus. The final group only saw the image 109 seconds. Surprisingly, the group that watched the image come into focus for the full 122 seconds had the lowest recognition rate (Source: Bruner and Potter, 1964).

time-reversal. Subjects who saw the images come into focus recognized the object correctly 44 percent of the time. Subjects who saw the image going out of focus recognized the object correctly 76 percent of the time. Seeing a low resolution version of an image can interfere with our subsequent ability to recognize the contents of an image.

Now, don't draw too strong a conclusion from this study about the problems progressive image enhancement will create. There are a number of features of this particular experiment which make the data quite unlike applications we might plan for progressive image transmission. Most importantly, in this study subjects never saw very clear images. At the best focus, only half of the subjects recognized the pictures at all. Also, the durations over which the images developed were quite slow, lasting minutes. These conditions are sufficiently unlike most planned applications of progressive image transmission that we cannot be certain the results will apply. I mention the result here to emphasize that even after the algorithms are in place, human testing will remain an important element of the system design.

Exercises

1. Answer these questions concerning the relations between image compression algorithms and visual perception.

   (a) What factors make it possible to achieve large compression ratios with natural images?

   (b) What properties of the human optics are important for compression? Do these have any implications for compression of colored images?

   (c) The *neuron doctrine* states that a stimulus that generates a strong neural response defines what the neuron codes, usually called the neuron's *trigger feature*. In this chapter we have defined the image feature corresponding to a transform coefficient. How does the definition we have used in this chapter compare with the definition of a feature used in the neuron doctrine? Explain how the definition in this chapter could be stated in terms of a physiological hypothesis.

   (d) There are certain properties, such as shadows, surface curvature, that are common across many different images. How might one take advantage of the properties of the image formation process to design more efficient image compression algorithms?

2. There are two ways in which compression can be useful. One is for representing the physical image efficiently. A second reason to compress image data is to obtain a form that is efficient for subsequent visual calculations of motion, color and form. Answer the following questions relating these two aspects of image compression.

   (a) Make a hypothesis about where in the visual pathways you would make a master representation of the input data, and how other cortical areas might retrieve data from this site.

   (b) What properties would be computed by the separate cortical areas?

   (c) What properties would define an efficient central representation of visual information?

   (d) How would you test your hypotheses with physiological or behavioral experiments?

3. Answer these questions about the JPEG-DCT image compression algorithm and image pyramids.

   (a) Describe the JPEG-DCT computation in terms of convolution and decimation.

(b) What is the main difference between the JPEG-DCT convolution / decimation process and the image pyramid convolution / decimation process?

(c) At what stage is information lost in compression algorithms?

(d) Why is it better to quantize the DCT coefficients rather than the quantize the individual pixels?

4. Here are some specific formulae concerning the DCT. Suppose that the data in an image block is $p(j,k)$. The DCT formula is

$$P(u,v) = \frac{4c(u)c(v)}{n^2} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} p(j,k) \cos(\frac{(2j+1)u\pi}{2n}) \cos(\frac{(2k+1)v\pi}{2n}) \qquad (1.14)$$

where $P(u,v)$ are the transform coefficients, and c is a normalizing function defined as

$$c(w) = \begin{cases} 1/\sqrt{2} & \text{if w = 0} \\ 1 & \text{otherwise} \end{cases} \qquad (1.15)$$

Answer the following questions about the DCT.

(a) Prove that the inverse to the DCT is

$$p(j,k) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} c(u)c(v)P(u,v) \cos(\frac{(2j+1)u\pi}{2n}) \cos(\frac{(2k+1)v\pi}{2n}) \qquad (1.16)$$

(Notice that, except for a scale factor, this formula is nearly the same as the forward DCT formula. Because of this we say the DCT is *self-inverting*.)

(b) The DCT is separable with respect to the rows and columns of the input image block. This means that we can express the DCT calculation as a matrix product $D_r P D_c$ where the matrix P represents the image data, $p(j,k)$ and the matrices D_r and D_c represent the DCT transformation. Create the matrices D_r and D_c. How are they related?

5. Answer these questions about wavelets and QMFs.

(a) What advantage(s) does orthogonality confer on the wavelet / QMF representation?

(b) Separability is a good property since it makes computation easier. Circular symmetry is another good feature since we can represent the value of a function from a single number. There is only one function that is both separable and circularly symmetric. What is it? Prove the result.

6. Wavelets have been in the news in recent years. Search recent newspaper articles in the library to see what claims have been made. Also, look for conference proceedings and companies started based on the technology.