5 Linear operators and kernels

Now I see through a glass darkly, but then, face to face

Paul of Tarsus

In this chapter,[1] we investigate linear operations on images. We first consider the derivative, probably the most common linear operator. That discussion is extended into edge detection, and we consider a variety of methods for accomplishing this objective.

5.1 What is a linear operator?

Suppose D is an operator which takes an image f and produces an image g. If D satisfies

$$D(\alpha f_1 + \beta f_2) = \alpha D(f_1) + \beta D(f_2), \tag{5.1}$$

where f_1 and f_2 are images, α and β are scalar multipliers, then we say that D is a "linear operator."

A gedankenexperiment

Consider the image operator D

$$g = D(f) = af + b \qquad a, b \in \Re$$

Is D a linear operator?

We suggest you work this out for yourself before reading the solution. It certainly LOOKS linear. Multiplication by a constant followed by addition of a constant. If f were a scalar variable, then D describes the equation of a line, which SURELY is linear (isn't it?)! OK. Let's prove it. Using Eq. (5.1), we evaluate

$$D(\alpha f_1 + \beta f_2) = a(\alpha f_1 + \beta f_2) + b = a\alpha f_1 + a\beta f_2 + b$$

[1] The authors are grateful to Bilge Karacali, Rajeev Ramanath, and Lena Soderberg for their assistance in producing the images used in this chapter.

and check to see if this is the same as

$$\begin{aligned}\alpha D(f_1) + \beta D(f_2) &= \alpha(af_1 + b) + \beta(af_2 + b)\\ &= a\alpha f_1 + \alpha b + a\beta f_2 + \beta b\\ &= a\alpha f_1 + a\beta f_2 + b(\alpha + \beta)\end{aligned}$$

so unless $\alpha + \beta = 1$, D is NOT a linear operator! This seems counter-intuitive, doesn't it? We'll have another look at this later and see if we can figure out why. In the remainder of this chapter, we will look at image operators which are linear.

5.2 Application of kernel operators in digital images

Since f is now digital, many authors choose to write f as a matrix, f_{ij}, instead of the functional notation $f(x, y)$. However, we prefer the x, y notation, for reasons which shall become apparent later. We will find it more convenient to use a single subscript f_i, at several points later, but for now, let's stick with $f(x, y)$ and remember that x and y take on only a small range of integer values, e.g. $0 < x < 511$.

Think about a one-dimensional image named f with five pixels and another one-dimensional image, which we will call a *kernel* named h with three pixels, as illustrated in Fig. 5.1.

Place the kernel down so its center, pixel h_0, is over some pixel of f, say f_2; we get $g_2 = f_1h_{-1} + f_2h_0 + f_3h_1$, which is a sum of products of elements of the kernel and elements of the image. With that understanding, let us consider the most common example of Eq. (5.2), the case we will come to call *application of a* 3×3 *kernel*:

$$g(x, y) = \sum_{\alpha}\sum_{\beta} f(x + \alpha, y + \beta)h(\alpha, \beta). \tag{5.2}$$

This case occurs when both α and β take on only the values -1, 0, and 1. In this case, Eq. (5.2) expands to

$$\begin{aligned}g(x, y) = {} & f(x - 1, y - 1)h(-1, -1) + f(x, y - 1)h(0, -1)\\ & + f(x + 1, y - 1)h(1, -1) + f(x - 1, y)h(-1, 0)\\ & + f(x, y)h(0, 0) + f(x + 1, y)h(1, 0) + f(x - 1, y + 1)h(-1, 1)\\ & + f(x, y + 1)h(0, 1) + f(x + 1, y + 1)h(1, 1).\end{aligned} \tag{5.3}$$

Remember, $y = 0$ is at the TOP of the image and y goes up as you go down in the image.

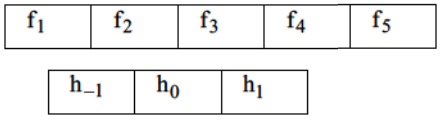


Fig. 5.1. A one-dimensional image with five pixels and a one-dimensional kernel with three pixels. The subscript is the *x*-coordinate of the pixel.

Note the order of the arguments here, x (column) and y (row). Sometimes the reverse convention is followed.

Table 5.1. *Values of the elements of a kernel.*

$h(-1,-1)$	$h(0,-1)$	$h(1,-1)$
$h(-1,0)$	$h(0,0)$	$h(1,0)$
$h(-1,1)$	$h(0,1)$	$h(1,1)$

To better capture the essence of Eq. (5.3), let us write h as a 3×3 grid of numbers (yes, we used the word "grid" rather than "array" intentionally), as in Table 5.1.

Now we imagine that we place this grid down on top of the image so that the center of the grid is directly over pixel $f(x, y)$; then each h value in the grid is multiplied by the corresponding point in the image. We will refer to the grid of h values henceforth as a "kernel."

5.2.1 On the direction of the arguments: Convolution and correlation

Let's restate two important equations. First the equation for a kernel operator, recopied from Eq. (5.2), and then the equation for two-dimensional, discrete convolution.

$$g(x, y) = \sum_{\alpha} \sum_{\beta} f(x + \alpha, y + \beta) h(\alpha, \beta) \tag{5.4}$$

$$g(x, y) = \sum_{\alpha} \sum_{\beta} f(x - \alpha, y - \beta) h(\alpha, \beta). \tag{5.5}$$

Mathematically, convolution and correlation differ in the left–right order of coordinates.

The observant student will have noticed a discrepancy in order between Eqs. (5.4) and (5.5). In formal convolution, as given by Eq. (5.5), the arguments reverse: the right-most pixel of the kernel (h_1) is multiplied by the left-most pixel in the corresponding region of the image (f_2). However, in Eq. (5.4), we think of "placing" the kernel down over the image and multiplying corresponding pixels. If we multiply corresponding pixels, left–left and right–right, we have correlation. There is, unfortunately, a misnomer in much of the literature – both may be called "convolution." We advise the student to watch for this. In many publications, the authors use the term "convolution" when they really mean "sum of products." In order to avoid confusion, in this book, we will avoid the use of the word "convolve" unless we really do mean the application of Eq. (5.5), and instead use the term "kernel operator," when we mean Eq. (5.4).

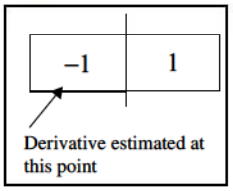
5.2.2 Using kernels to estimate derivatives

Let us examine this concept via an example – approximating the spatial derivatives of the image, $\partial f/\partial x$ and $\partial f/\partial y$.

We recall from some dimly remembered calculus class,

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

which would suggest the following kernel could be used (for $\Delta x = 1$):



But this kernel is aesthetically unpleasing – the estimate at x depends on the value at x and at $x + 1$, but not at $x - 1$; why? We actually like a symmetric definition better, such as

$$\left.\frac{\partial f}{\partial x}\right|_{x_a} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x}.$$

We cannot get Δx smaller than 1, and we end up with this kernel

−1/2	0	1/2

which, for notational simplicity, we write as

$$1/2 \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline \end{array}.$$

A major problem with derivatives is noise sensitivity. We compensate for this by taking the difference horizontally, and then averaging vertically – which produces the following kernel:

$$\left.\frac{\partial f}{\partial x}\right|_{x_0} = \frac{1}{6}\left(\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \otimes f\right) \tag{5.6}$$

where we have introduced a new symbol, $\otimes$ which will denote the sum-of-products implementation described above. The literature abounds with kernels like this one. All of them combine the concept of estimating the derivative by differences, and then averaging the result in some way to compensate for noise. Probably the best known of these ad hoc kernels is the Sobel:

$$\left.\frac{\partial f}{\partial x}\right|_{x_0} = \frac{1}{8}\left(\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \otimes f\right). \tag{5.7}$$

The Sobel operator has the benefit of being center-weighted.

5.3 Derivative estimation by function fitting

This approach presents yet another way to make use of the continuous representation of an image $f(x, y)$. Think of the brightness as a function of the two spatial coordinates, and consider a plane which is tangent to that brightness surface at a point, as illustrated in Fig. 5.2.

In this case, we may write the continuous image representation using the equation of a plane

$$f(x, y) = ax + by + c. \tag{5.8}$$

Then, we may consider the edge strength using the two numbers $\partial f/\partial x = a$, $\partial f/\partial y = b$, and the rate of change of brightness at the point (x, y) is represented by the gradient vector

$$\nabla f = \left[\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y}\right]^{\mathrm{T}} = [a \quad b]^{\mathrm{T}}. \tag{5.9}$$

The approach followed here is to find a, b, and c given some noisy, blurred measurement of f, and the assumption of Eq. (5.8).

To find those parameters, first observe that Eq. (5.8) may be written as $f(x, y) = A^{\mathrm{T}}X$ where the vectors A and X are $A^{\mathrm{T}} = [a \;\; b \;\; c]$ and $X^{\mathrm{T}} = [x \;\; y \;\; 1]$.

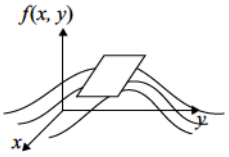
Suppose we have measured brightness values $g(x, y)$ at a collection of points $\aleph \subset Z \times Z$ (Z is the set of integers) in the image. Over that set of points, we wish to find the plane which best fits the data. To accomplish this objective, write the error as a function of the measurement and the (currently unknown) function $f(x, y)$.

A sum-squared error objective function

$$E = \sum_{\aleph}(f(x, y) - g(x, y))^2 = \sum_{\aleph}(A^{\mathrm{T}}X - g(x, y))^2.$$

Expanding the square and eliminating the functional notation for simplicity, we find

$$E = \sum_{\aleph}(A^{\mathrm{T}}X)(A^{\mathrm{T}}X) - 2A^{\mathrm{T}}Xg + g^2.$$



Remembering that for vectors A and X, $A^{\mathrm{T}}X = X^{\mathrm{T}}A$, and taking the summation

Fig. 5.2. The brightness in an image can be thought of as a surface, a function of two variables. The slopes of the tangent plane are the two spatial partial derivatives.

through, we have

$$E = \sum_{\aleph} A^{\mathrm{T}} X X^{\mathrm{T}} A - 2 \sum_{\aleph} A^{\mathrm{T}} X g + \sum_{\aleph} g^2$$
$$= A^{\mathrm{T}} \left(\sum_{\aleph} X X^{\mathrm{T}} \right) A - 2 A^{\mathrm{T}} \sum_{\aleph} X g + \sum_{\aleph} g^2.$$

Now we wish to find the A (the parameters of the plane) which minimizes E; so we may take derivatives and set the result to zero.

$$\frac{dE}{dA} = 2 \left(\sum_{\aleph} X X^{\mathrm{T}} \right) A - 2 \sum_{\aleph} X g = 0. \tag{5.10}$$

Let's call $\sum_{\aleph} X X^{\mathrm{T}} \equiv S$ (it is the "scatter matrix") and see what Eq. (5.10) means: consider a neighborhood $\aleph$ which is symmetric about the origin. In that neighborhood, suppose x and y only take on values of -1, 0, and 1, then

$$S = \sum_{\aleph} X X^{\mathrm{T}} = \sum_{\aleph} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} [x \quad y \quad 1] = \begin{bmatrix} \sum x^2 & \sum xy & \sum x \\ \sum xy & \sum y^2 & \sum y \\ \sum x & \sum y & \sum 1 \end{bmatrix}$$

which, for the neighborhood described, is

$$\begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 9 \end{bmatrix}.$$

More detail on how the elements of the scatter matrix are derived. Do not forget the positive direction for y is down.

You do not see where those values came from?

Ok, here's how you get them. Look at the top left point, at coordinates $x = -1$, $y = -1$. At that point, $x^2 = (-1)^2 = 1$. Now look at the top middle point, at coordinates $x = 0$, $y = -1$. At that point, $x^2 = 0$. Do this for all 9 points in the neighborhood, and you obtain $\sum x^2 = 6$. Got it?

Convince yourself that this is true.

Useful observation: If you make the neighborhood symmetric about the origin, all the terms in the scatter matrix which contain x or y to an odd power will be zero.

Be sure you carefully proofread whatever you write!

Also: A common miss steak is to put a 1 in the lower right corner rather than a 9 – be careful!

So now we have the matrix equation

$$2 \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 2 \begin{bmatrix} \sum g(x,y)x \\ \sum g(x,y)y \\ \sum g(x,y) \end{bmatrix}.$$

We can easily solve for a:

$$a = \frac{1}{6}\sum g(x, y)x \approx \frac{\partial f}{\partial x}.$$

So, to compute the derivative from a fit to a neighborhood at each of the nine points in the neighborhood, take the measured value at that point, multiply by its x coordinate, and add them up. Let's write down the x coordinates in tabular form:

−1	0	1
−1	0	1
−1	0	1

That is *precisely* the kernel of Eq. (5.6), which we derived intuitively. Now, we have it derived formally. Doesn't it give you a warm fuzzy feeling when theory agrees with intuition?!? (Whoops, we forgot to multiply each term by 1/6, but we can simply factor the 1/6 out, and when we get the answer, we will just divide by 6.)

We accomplished this by using an optimization method, in this case, minimizing the squared error, to find the coefficients of a function $f(x)$ in an equation of the form $y = f(x)$, where f is polynomial. Recall from section 4.1.2 that this form is referred to as an explicit functional representation.

One more terminology issue: In future material, we will use the expression *radius of a kernel*. The radius is the number of pixels from the center to the nearest edge. For example, a 3 × 3 kernel has a radius of one. A 5 × 5 kernel has a radius of 2, etc. It is possible to design kernels which are circular, but most of the time, we use squares.

Finding image gradients in hexagonal arrays of pixels

In this section, we find image gradients again, in exactly the same way, but this time with hexagonally arranged pixels. Refer to section 4A.1 for a discussion of the coordinate system. It is a different presentation of the same material, and if you read both presentations carefully, you will understand the concepts more clearly.

To find the gradient of intensity in an image, we will fit a plane to the data in a small neighborhood. This plane will be represented in the form of Eq. (5.8). We then take partial derivatives with respect to u and v to find the gradient of intensity in those corresponding directions. We choose a neighborhood of six points, surrounding a central point, and fit the plane to them. Define the set of data points as $z_i, (i = 1, \ldots, 6)$. Then the following expression represents the error in fitting these six points to a plane parameterized by a, b, and c.

$$E = \sum_{i=1}^{6}(z_i - (au_i + bv_i + c))^2. \tag{5.11}$$

In order to represent E in a form that will be easy to differentiate, we will reformulate the argument of the summation using matrix notation. Define vectors $A = [a \;\; b \;\; c]^{\mathrm{T}}$ and $Z = [u \;\; v \;\; 1]^{\mathrm{T}}$. Then E may be written using

$$E = \sum_{i=1}^{6} (z_i - A^{\mathrm{T}} Z_i)^2 \tag{5.12}$$

$$= \sum_{i=1}^{6} \left(z_i^2 - 2 z_i A^{\mathrm{T}} Z_i + A^{\mathrm{T}} Z_i A^{\mathrm{T}} Z_i \right). \tag{5.13}$$

First, we observe that $A^{\mathrm{T}} Z = Z^{\mathrm{T}} A$, and rewrite Eq. (5.13), temporarily dropping the limits on the summation to make the typing easier:

$$= \sum z_i^2 - 2A^{\mathrm{T}} \sum z_i Z_i + A^{\mathrm{T}} \left(\sum Z_i Z_i^{\mathrm{T}} \right) A. \tag{5.14}$$

The term in parentheses in Eq. (5.14) is the scatter matrix, the collection of locations of points at which data exists. We denote this matrix by the symbol S. In order to find the value of vector A which minimizes E^2, we take the partial derivative with respect to A:

$$\frac{\partial E}{\partial A} = -2 \sum z_i Z_i + 2SA. \tag{5.15}$$

Evaluating S, we find

$$S = \begin{bmatrix} \sum u_i^2 & \sum u_i v_i & \sum u_i \\ \sum u_i v_i & \sum v_i^2 & \sum v_i \\ \sum u_i & \sum v_i & \sum 1 \end{bmatrix} = \begin{bmatrix} 4 & -2 & 0 \\ -2 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}.$$

(One finds the numerical values by summing the u and v coordinates over each of the pixels in the neighborhood, as illustrated in Fig. 4.15, assuming the center pixel is at location 0, 0.)

Define

$$\sum z_i u_i \equiv \Upsilon_u$$
$$\sum z_i v_i \equiv \Upsilon_v,$$

and set the partial derivative of Eq. (5.15) equal to zero to produce a pair of simultaneous equations,

$$\begin{aligned} 4a - 2b &= \Upsilon_u \\ -4a + 8b &= 2\Upsilon_v \end{aligned} \tag{5.16}$$

with solution

$$b = \frac{1}{6}(2\Upsilon_v + \Upsilon_u). \tag{5.17}$$

Similarly,

$$a = \frac{1}{6}(\Upsilon_v + 2\Upsilon_u). \tag{5.18}$$

Substituting actual values of u and v at each pixel in the six-pixel neighborhood, we determine the gradient vector, since the gradient in the u direction is a and the gradient in the v direction is b.

We rewrite the equation for a, substituting in the definitions of the Υs:

$$a = \frac{1}{6}\left(\sum z_i v_i + 2\sum z_i u_i\right) = \frac{1}{6}\sum z_i(v_i + 2u_i).$$

Now, look at the pixel directly to the right of center, its u, v coordinates are 1, 0. So, we should multiply the image brightness, Z_i, at that point by $(v_i + 2u_i)$, or $(0 + 2 \times 1)$ which is 2. That is the value that goes into the kernel. We need to remember in using this method, that the answer we get is not quite right . . . It is actually six times the best estimate, and we need to divide by six, if the actual value of the derivative is important, rather than a result proportional to the estimated derivative. Repeating this process at each point, we obtain the kernels shown in Fig. 5.3.

The concepts of fitting described in this section are ubiquitous. They pop up all over the discipline of machine vision. You find a need to fit gray values, surfaces, lines (and we will do that in Chapter 9), curves, etc. There are even techniques for fitting when data has known statistical variations [5.7, 5.40].

5.4 Vector representations of images

Suppose we list every pixel in an image in raster scan order, as one long vector. For example, for the 4×4 image

$f(x, y) =$

1	2	4	1
7	3	2	8
9	2	1	4
4	1	2	3

$F = [1\ 2\ 4\ 1\ 7\ 3\ 2\ 8\ 9\ 2\ 1\ 4\ 4\ 1\ 2\ 3]^T$.

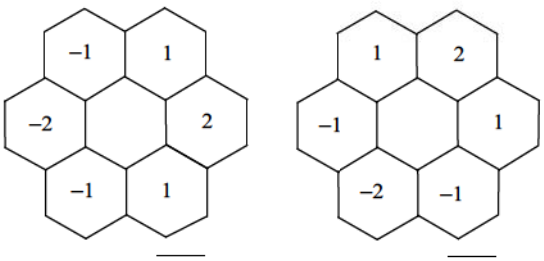


Fig. 5.3. The kernels used to estimate the gradient of brightness in the *u* direction, and in the *v* direction.

0, 0 is the UPPER left corner.

This is called the "lexicographic" representation. If we write the image in this way, each pixel may be identified by a single index, e.g., $F_0 = 1$, $F_4 = 7$, $F_{15} = 3$, where the indexing starts with zero.

Now suppose we want to apply the following kernel to this image:

$$h = \begin{bmatrix} -1 & 0 & 2 \\ -2 & 0 & 4 \\ 3 & 9 & 1 \end{bmatrix}$$

at point $x = 1$, $y = 1$, again, starting the indexing at zero.

A dot product computes application of a kernel.

Point (1, 1) corresponds to pixel F_5 in the image. We could accomplish this application of the kernel by taking the dot product of the vector F with the vector

$$H_5 = [-1 \quad 0 \quad 2 \quad 0 \quad -2 \quad 0 \quad 4 \quad 0 \quad 3 \quad 9 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^{\mathrm{T}}.$$

Now you try it. Determine what vector to use to apply this kernel at (2, 2). Did you get this?

$$H_{10} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 2 \quad 0 \quad -2 \quad 0 \quad 4 \quad 0 \quad 3 \quad 9 \quad 1]^{\mathrm{T}}.$$

Now try (2, 1) ($x = 2$, $y = 1$):

$$H_6 = [0 \quad -1 \quad 0 \quad 2 \quad 0 \quad -2 \quad 0 \quad 4 \quad 0 \quad 3 \quad 9 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]^{\mathrm{T}}.$$

Compare H_5 at (1, 1) and H_6 at (2, 1). They are the same except for a rotation. We could convolve the entire image by constructing a matrix in which each column is one such H. Doing so would result in a matrix such as the one illustrated. By producing the product $G = H^{\mathrm{T}}F$, G will be the (vector form of) convolution of image F with kernel H.

$$\begin{bmatrix}
\dots & -1 & 0 & \dots & 0 & \dots \\
\dots & 0 & -1 & \dots & 0 & \dots \\
\dots & 2 & 0 & \dots & 0 & \dots \\
\dots & 0 & 2 & \dots & 0 & \dots \\
\dots & -2 & 0 & \dots & 0 & \dots \\
\dots & 0 & -2 & \dots & -1 & \dots \\
\dots & 4 & 0 & \dots & 0 & \dots \\
\dots & 0 & 4 & \dots & 2 & \dots \\
\dots & 3 & 0 & \dots & 0 & \dots \\
\dots & 9 & 3 & \dots & -2 & \dots \\
\dots & 1 & 9 & \dots & 0 & \dots \\
\dots & 0 & 1 & \dots & 4 & \dots \\
\dots & 0 & 0 & \dots & 0 & \dots \\
\dots & 0 & 0 & \dots & 3 & \dots \\
\dots & 0 & 0 & \dots & 9 & \dots \\
\dots & 0 & 0 & \dots & 1 & \dots
\end{bmatrix}$$

H_5 H_6

Some observations about this process:

- The resulting matrix is a "circulant" matrix. Each column is simply a rotation of the adjacent column.
- The fact that we can apply kernels in this way is yet another demonstration of the fact that kernel operators are linear operators.
- This form suggests one approach for dealing with the nasty problem of boundary conditions (you *did* think about that, didn't you?). Specifically, how do you multiply by the data value *above* when you are on the top line? One answer is to rotate around the image and pull from the bottom.
- The matrix H is VERY large. If f is a typical image of 256 × 256 pixels, then H is (256 × 256) × (256 × 256) which is a large number (although still smaller than the US national debt). Do not worry about the monstrous size of H. Nobody (well, almost nobody) ever computes H and uses it in this way. This form is useful for thinking about images and for proving theorems about image operators – it is a conceptual, not a computational tool.

Finally, multiplication by a circulant matrix can be accomplished considerably faster by using the fast Fourier transform; but more about that later.

5.5 Basis vectors for images

In the previous section, we saw that we could think of an image as a vector. If we can do that for an image, surely we can do the same thing for a small subimage. Consider the nine-pixel neighborhood of a single point. We can easily construct the 9-vector which is the lexicographic representation of that neighborhood.

In Chapter 2, we learned that any vector could be represented as a weighted sum of basis vectors, and we will apply that same concept here. Let's rewrite Eq. (2.6) in the form:

$$V = \sum_{i=1}^{9} a_i \boldsymbol{u}_i$$

where now V is the 9-vector representation of this nine-pixel neighborhood, the a_i are scalar weights, and the $\boldsymbol{u}_i$ are some set of orthonormal basis vectors.

But what basis vectors should we use? More to the point, what basis set *would be useful*? The set we normally use, the Cartesian basis, is

$$\begin{aligned}
\boldsymbol{u}_1 &= [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^{\mathrm{T}} \\
\boldsymbol{u}_2 &= [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^{\mathrm{T}} \\
&\vdots \\
\boldsymbol{u}_9 &= [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^{\mathrm{T}}
\end{aligned}$$

which, while convenient and simple, does not help us at all here. Could another basis be more useful? (answer yes). Before we figure out what, do you remember how many possible basis vectors there are for this real-valued 9-space? The answer is "a zillion".[2] With so many choices, we should be able to pick some good ones. To accomplish that, recall the role of the coefficients a_i. Recall that if some particular a_i is much larger than all the other as, it means that V is "very similar" to $\boldsymbol{u}_i$. Computing the as then allows us a means to find which of a set of prototype neighborhoods a particular image most resembles.

Fig. 5.4 illustrates a set of prototype neighborhoods developed by Frei and Chen [5.12]. Notice that neighborhood ($\boldsymbol{u}_1$) is negative below and positive above the horizontal center line, and therefore is indicative of a horizontal edge, or a point where $\partial f / \partial y$ is large.

Now recall how to compute the projection a_i. The scalar-valued projection of a vector V onto a basis vector $\boldsymbol{u}_i$ is the inner product $a_i = V^{\mathrm{T}} \boldsymbol{u}_i$.

[2] Actually, the correct answer is infinity – a zillion is just an engineering approximation.

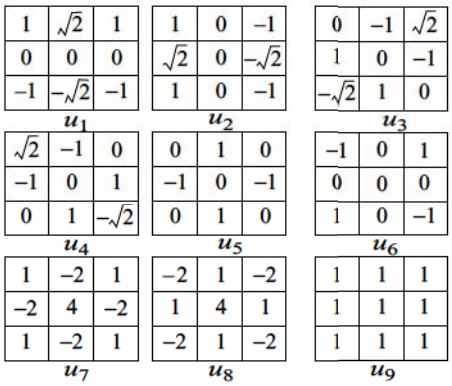


Fig. 5.4. Frei-Chen basis vectors.

Do you think you could develop a similar basis set for the seven pixels in a hexagonal neighborhood? Hint: Seven pixels defines a seven-dimensional vector space. You will need to find seven such vectors.

One way to determine how similar a neighborhood about some point is to a vertical edge is to compute the inner product of the neighborhood vector with the vertical edge basis vector. One final question: What is the difference between calculating this projection and convolving the image at that point with a kernel which estimates $\partial f / \partial x$? The answer is left as an exercise to the student. (Don't you wish they were all this easy?)

So now you know all there is to know (almost) about linear operators and kernel operators. Let's move on to an application to which we have already alluded – finding edges.

5.6 Edge detection

Edges are areas in the image where the brightness changes suddenly; where the derivative (or more correctly, *some* derivative) has a large magnitude. We can categorize edges as step, roof, or ramp [5.20], as illustrated in Fig. 5.5.

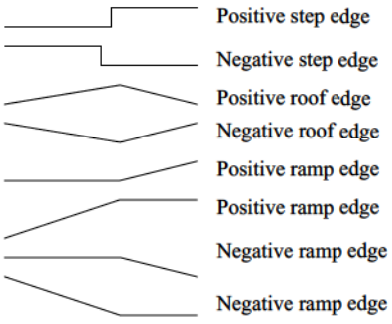


Fig. 5.5. Types of commonly occurring edges. Note that the term positive or negative generally refers to the sign of the first instance of the first derivative.

We have already seen (twice) how application of a kernel such as

$$h_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \tag{5.19}$$

approximates the partial derivative wrt x. Similarly,

Which is correct? It depends on which direction you have chosen as positive y.

$$h_y = \begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \quad \text{or} \quad \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array} \tag{5.20}$$

estimates $\partial f / \partial y$.

Some other forms have appeared in the literature that you should know about for historical purposes.

Remember the Sobel operator?

$$h_y = \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \quad \text{or} \quad \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \tag{5.21}$$

Important

(This will give you trouble for the entire semester, so you may as well start now.) In software implementations, the positive y direction is DOWN! This results from the fact that scanning is top-to-bottom, left-to-right. So pixel (0, 0) is the *upper* left corner of the image. Furthermore, numbering starts at zero, not one. We find the best way to avoid confusion is to never use the words "x" and "y" in writing programs, but instead use "row" and "column" remembering that now 0 is on top.

However, in these notes, we will use conventional Cartesian coordinates in order to get the math right, and to further confuse the student (which is, after all, what Professors are there for. Right?).

Having cleared up that muddle, let us proceed. Given the gradient vector

$$\nabla f = \left[\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y} \right]^{\mathrm{T}} \equiv [G_x \quad G_y]^{\mathrm{T}} \tag{5.22}$$

we are interested in its magnitude

$$|\nabla f| = \sqrt{G_x^2 + G_y^2} \tag{5.23}$$

(which we will call the "edge strength") and its direction

$$\angle \nabla f = \text{atan}\left(\frac{G_y}{G_x}\right). \tag{5.24}$$

One way to find an edge in an image is to compute the "gradient magnitude" image, and to threshold that. So go try it: Work homework Assignment 5.5.

While you are at it, you might want to work Assignment 5.6 too.

What did you learn from those experiments? Clearly, several problems arise when we try to find edges by simple kernel operations. During much of the remainder of the course, we will address these issues. First, let us improve our kernel-based edge detection.

5.7 A kernel as a sampled differentiable function

We hope you have realized by now that all the edge detector operators you have used so far are doing two things simultaneously; *smoothing* (read "low-pass filtering," "noise removal," "averaging," or "blurring") and *differentiation* (read "high-pass filtering" or "sharpening"). The kernel of Eq. (5.6) actually takes the vertical average of three derivative estimates. It is actually counter-intuitive, however, since it weights the center pixel the same as the lines above and below.

Consider the result you got on Assignment 5.6. If you did it correctly, the kernel values increase as they are farther from the center. That is even worse, right? Why should data points *farther away* from the point where we are estimating the derivative contribute more heavily to the estimate? Wrong! Wrong! Wrong! It is an artifact of the assumption we made that *all* the pixels fit the same plane. They obviously don't.

So here's a better way – weight the center pixel more heavily. You already saw this – the Sobel operator, Eq. (5.7) does it. But now, let's get a bit more rigorous. Let's blur the image by applying a kernel which is bigger in the middle and then differentiate. We have lots of choices for a kernel like that, e.g., a triangle or a Gaussian, but thorough research [5.28] has shown that a Gaussian works best for this sort of thing. We can write this process as

$$d = \frac{\partial}{\partial x}(g \otimes h)$$

Recall what you learned about linear systems.

where now g is the measured image, h is a Gaussian, and d will be our new derivative estimate image. Now, a crucial point from linear systems theory:

For linear operators D and $\otimes$,

$$D(g \otimes h) = D(h) \otimes g. \tag{5.25}$$

Equation (5.25) means we do not have to do blurring in one step and differentiation in the next; instead, we can pre-compute the derivative of the blur kernel and simply apply the resultant kernel.

Let's see if we can remember how to take a derivative of a 2D Gaussian (did you forget it is a 2D function?).

A d-dimensional multivariate Gaussian has the general form

$$\frac{1}{(2\pi)^{d/2}|K|^{1/2}} \exp\left(-\frac{[\boldsymbol{x}-\boldsymbol{\mu}]^{\mathrm{T}} K^{-1}[\boldsymbol{x}-\boldsymbol{\mu}]}{2}\right) \tag{5.26}$$

where K is the covariance matrix and $\boldsymbol{\mu}$ is the mean vector. Since we want a Gaussian centered at the origin (which will be the center pixel) $\boldsymbol{\mu} = 0$, and since we have no reason to prefer one direction over another, we choose K to be diagonal (isotropic)

$$K = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} = \sigma^2 I. \tag{5.27}$$

For two dimensions, Eq. (5.26) simplifies to

Learn this Gaussian blur stuff, and how to use it to compute a first- or second-derivative kernel. It is guaranteed to be on the exam.

$$h(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-[x\ y]^{\mathrm{T}}[x\ y]}{2\sigma^2}\right) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) \tag{5.28}$$

and

$$\frac{\partial}{\partial x} h(x, y) = \frac{-x}{2\pi\sigma^4} \exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right). \tag{5.29}$$

If our objective is edge detection, we are done. However, if our objective is precise estimation of derivatives, particularly higher order derivatives, use of a Gaussian kernel, since it blurs the image, clearly introduces errors which can only be partially compensated for [5.39]. Nevertheless, this is one of the most simple ways to develop effective derivative kernels.

For future reference, here are a few of the derivatives of the one-dimensional Gaussian. Even though there is no particular need for the normalizing $\sqrt{2\pi}$ for most of our needs (it just ensures that the Gaussian integrates to one), we have included it. That way these formulae are in agreement with the literature. The subscript notation is used here to denote derivatives. That is,

$$G_{xx}(\sigma, x) = \frac{\partial^2}{\partial x^2} G(\sigma, x),$$

where $G(\sigma, x)$ is a Gaussian function of x with mean of zero and standard deviation σ.

Don't scoff at third derivatives. You never know when you might need one (like finding the maxima of the second derivative).

$$\begin{aligned}
G(\sigma, x) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \\
G_x(\sigma, x) &= \frac{-x}{\sqrt{2\pi}\sigma^3} \exp\left(-\frac{x^2}{2\sigma^2}\right) \\
G_{xx}(\sigma, x) &= \left(\frac{x^2}{\sqrt{2\pi}\sigma^5} - \frac{1}{\sqrt{2\pi}\sigma^3}\right) \exp\left(-\frac{x^2}{2\sigma^2}\right) \\
G_{xxx}(\sigma, x) &= \frac{x}{\sqrt{2\pi}\sigma^5} \left(3 - \frac{x^2}{\sigma^2}\right) \exp\left(-\frac{x^2}{2\sigma^2}\right).
\end{aligned} \tag{5.30}$$

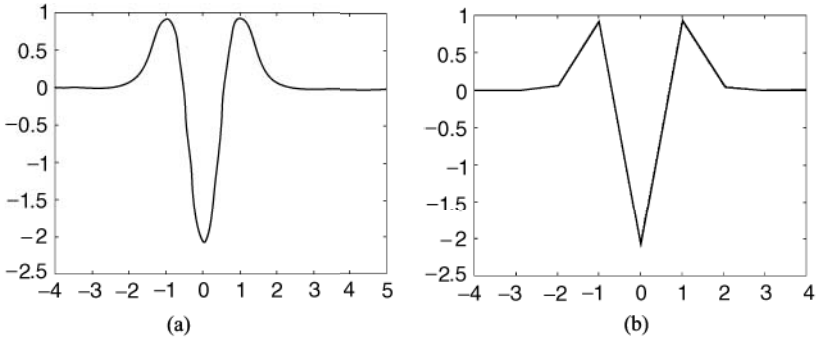


Fig. 5.6. (a) The second derivative of a one-dimensional Gaussian centered at 0. (b) A three-point approximation.

Let's look in a bit more detail about how to make use of these formulae and their two-dimensional equivalents to derive kernels.

The simplest way to get the kernel values for the derivatives of a Gaussian is to simply substitute $x = 0, 1, 2$, etc. along with their negative values, which yields numbers for the kernel. The first problem to arise is "what should σ be?" To address these questions, we will derive the elements of the kernel used for the second derivative of a one-dimensional Gaussian. The other derivatives can be developed using the same philosophy. Take a look at Fig. 5.6(a) and ask, "is there a value of σ such that the maximum of the second derivative occurs at $x = -1$ and $x = 1$?" Clearly there is, and its value is $\sigma = 1/(\sqrt{3})$. Given this value of σ, we can compute the values of the second derivative of a Gaussian at the integer points $x = \{-1, 0, 1\}$. At $x = 0$, we find $G_{xx}(1/\sqrt{3}, 0) = -2.07$, and at $x = 1$, $G_{xx}(1/\sqrt{3}, 1) = 0.9251$. So are we finished? That wasn't so hard, was it? Unfortunately, we are not done. It is very important that the elements of the kernel sum to zero. If they don't, then iterative algorithms like those described in Chapter 6 will not maintain the proper brightness levels over many iterations. The kernel also needs to be symmetric. That essentially defines the second derivative of a Gaussian. The most reasonable set of values close to those given, which satisfy symmetry and summation to zero are $\{1, -2, 1\}$.

Proving this is the correct value of σ is a homework problem.

The elements of any kernel which approximates a derivative must sum to zero.

However, this does not teach us very much. Let's look at a 5×1 kernel and see if we can learn a bit more. We require the following.

- The elements of the kernel should approximate the values of the appropriate derivative of a Gaussian as closely as possible.
- The elements must sum to zero.
- The kernel should be symmetric about its center, unless you want to do special processing.

We can calculate the elements of a five-element one-dimensional Gaussian, and if we do so, assuming $\sigma = 1/\sqrt{3}$, for $x = \{-2, -1, 0, 1, 2\}$, we get $[0.0565, 0.9251, -2.0730, 0.9251, 0.0565]$. Unfortunately, those numbers do not sum to zero. It is

very important that the kernel values integrate to zero, not quite so important that the actual values be precise. So what do we do in a case like this? We use constrained optimization. One strategy is to set up a problem to find a second derivative of a Gaussian, which has these values as closely as possible, but which integrates to zero. For more complex problems, the authors use *Interopt* [5.3] to solve numerical optimization problems, but you can solve this problem without using numerical methods. This is accomplished as follows. First, understand the problem (presented for the case of five points given above): We wish to find five numbers as close as possible to [0.0565, 0.9251, −2.0730, 0.9251, 0.0565] which satisfy the constraint that the five sum to zero. By symmetry, we actually only have three numbers, which we will denote $[a, b, c]$. For notational convenience, introduce three constants $\alpha = 0.0565$, $\beta = 0.9251$, $\gamma = -2.073$. Thus, to find a, b, and c which resemble these numbers, we write the mean squared error (MSE) form

$$H_0(a, b, c) = 2(a - \alpha)^2 + 2(b - \beta)^2 + (c - \gamma)^2. \tag{5.31}$$

H is the constrained version of H_0.

Using the concept of Lagrange multipliers, we can find the best choice of a, b, and c by minimizing a different objective function

$$H(a, b, c) = 2(a - \alpha)^2 + 2(b - \beta)^2 + (c - \gamma)^2 + \lambda(2a + 2b + c). \tag{5.32}$$

A few words of explanation are in order for those students who are not familiar with constrained optimization using Lagrange multipliers. The term with the λ in front (λ is the Lagrange multiplier) is the constraint. It is formulated such that it is exactly equal to zero, if we should find the proper a, b, and c. By minimizing H, we will find the parameters which minimize H_0 while simultaneously satisfying the constraint.

To minimize H, take the partials and set them equal to zero:

$$\begin{aligned} \frac{\partial H}{\partial a} &= 4a - 4\alpha + 2\lambda \\ \frac{\partial H}{\partial b} &= 4b - 4\beta + 2\lambda \\ \frac{\partial H}{\partial c} &= 2c - 2\gamma + \lambda. \end{aligned} \tag{5.33}$$

Setting the partial derivatives equal to zero, simplifying, and adding the constraint, we find the following set of linear equations:

$$\begin{aligned} a &= \alpha - \frac{\lambda}{2} \\ b &= \beta - \frac{\lambda}{2} \\ c &= \gamma - \frac{\lambda}{2} \\ 2a + 2b + c &= 0 \end{aligned} \tag{5.34}$$

which we solve to find the sets given in Table 5.2.

In the case of the first derivative, symmetry ensures the values always sum to zero, so no "tweaking" is necessary. Therefore the integer values are just as good as the floating point ones.

Table 5.2. *Derivatives of a one-dimensional Gaussian.*

1st deriv, 3 × 1	[0.2420, 0.0, −0.2420] or [1, 0, −1]
1st deriv, 5 × 1	[0.1080, 0.2420, 0, −0.2420, −0.1080]
2nd deriv, 3 × 1	[1, −2, 1]
2nd deriv, 5 × 1	[0.07846, 0.94706, −2.05104, 0.94706, 0.07846]

0.0261	0	–0.0261
0.1080	0	–0.1080
0.0261	0	–0.0261

Fig. 5.7. The 3×3 first derivative kernel.

We can proceed in the same way to compute the kernels to estimate the partial derivatives using Gaussians in two dimensions.

One implementation of the first derivative with respect to x, assuming an isotropic Gaussian is presented in Fig. 5.7. You will have the opportunity to derive others as homeworks.

In this chapter, we have explored the idea of edge operators based on kernel operators. We discovered that no matter what, noisy images result in edges which are:

- too thick in places
- missing in places
- extraneous in places.

That is just life – we cannot do any better with simple kernels. In Chapter 6, we will explore some approaches to these problems.

As we hope you have guessed, there are other ways of finding edges in images besides simply thresholding a derivative. In later sections, we will mention a few of them.

5.7.1 Higher order derivatives

We have just seen how second or third derivatives may be computed using derivatives of Gaussians. Since the topic has come up, and you will need to know the terminology later, we define here two scalar operators which depend on the second derivatives: the Laplacian and the quadratic variation.

The Laplacian of brightness at a point x, y is

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2},$$

whereas the quadratic variation of brightness is

$$\left(\frac{\partial^2 f}{\partial x^2}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2.$$

The Laplacian may be approximated by several kernels, including

−1	2	−1
2	−4	2
−1	2	−1

.

5.8 Computing convolutions

Remembering that the only difference between convolution and a kernel operator is the direction of x and y (section 5.2.1) for any kernel operator, there is an equivalent convolution kernel. Therefore efficient ways to calculate convolution are also efficient ways to apply kernel operators. The convolution operation may be computed directly as discussed above. It is simply a sum of products, calculated in the neighborhood of each pixel. However, it may also be computed by the Fourier transform. The Fourier transform of a convolution is the product of the Fourier transforms of the two arguments. That is (denoting convolution by the operator $\otimes$), we are concerned with computing

$$g(x, y) = f(x, y) \otimes h(x, y).$$

Let the Fourier transforms of the two images and the convolution kernel be defined by

$$\begin{aligned} G(\omega_x, \omega_y) &= F(g(x, y)) \\ F(\omega_x, \omega_y) &= F(f(x, y)) \\ H(\omega_x, \omega_y) &= F(h(x, y)) \end{aligned}$$

where the symbol F denotes the process of taking the Fourier transform. Remember from section 4.1.5, the Fourier transform of an image (a function of two variables) is itself a function of two variables. We refer to those variables, ω_x and ω_y, as the spatial frequencies in the x and y directions, respectively. Then G is the product of F and H.

$$G(\omega_x, \omega_y) = F(\omega_x, \omega_y) \bullet H(\omega_x, \omega_y). \tag{5.35}$$

The "product" of two transforms means, for each spatial frequency value (each combination of ω_x and ω_y), multiply the values of the two functions. (Just in case you do not remember the details, in general, these values are complex numbers.)

The ability to perform point-by-point multiplication is significant because of the computational complexity. Consider the complexity of convolving an $N \times N$ image with an $L \times L$ kernel. Doing it spatially, we have $L \times L$ multiplications for each pixel, totalling N^2L^2. Doing it with the Fourier transform (this assumes you are using a particular algorithm called the fast Fourier transform) works out in the following way. (The details are outside the scope of this section, but we will assume that the Fourier transform of an $N \times N$ image is itself a two-dimensional array of the same size.)

- Transform f: $N^2 \log N$.
- Transform h: $L^2 \log L$.
- Perform appropriate operations, such as padding, to get H and F the same size.
- Multiply H by F: N^2.
- Inverse transform the result: $N^2 \log N$.

If the sum of these four terms is smaller than N^2L^2, it is computationally more effective to go through the (considerable) inconvenience of using the transform domain. But for a particular image size and kernel size, what should we do? Fortunately, the relative efficiency of Fourier and spatial methods for computing convolutions of varying sizes has been analyzed, and the results are illustrated in Fig. 5.8. In that figure, we see that for kernels larger than about 15×15, we should use Fourier methods, for kernels smaller than 7×7, we should use spatial methods. The peculiar variations in the region boundary occur because the FFT requires the image size be a power of two, and images of other sizes introduce additional complications.

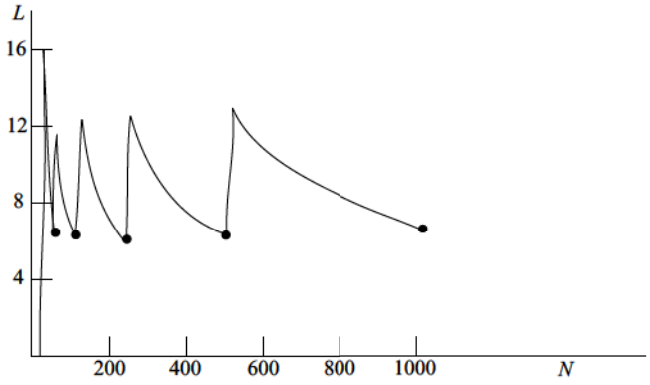


Fig. 5.8. Efficiency of computing a convolution with an $L \times L$ kernel on an $N \times N$ image. Combinations above the curve shown are more efficiently computed using Fourier methods, below the curve, by spatial methods (redrawn from Pratt [5.33]).

5.9 Scale space

"Scale space" is a recent addition to the well-known concept of image pyramids, first used in picture processing by Kelly [5.19] and later extended in a number of ways (see [5.5, 5.8, 5.30, 5.32], and many others). In a pyramid, a series of representations of the same image are generated, each created by a 2 : 1 subsampling (or averaging) of the image at the next higher level (Fig. 5.9).

In Fig. 5.10, a Gaussian pyramid is illustrated. It is generated by blurring each level with a Gaussian prior to 2 :1 subsampling. An interesting question should arise as you look at this figure. Could you, from all the data in this pyramid, reconstruct the original image? The answer is "no, because at each level, you are throwing away high-frequency information."

Usually, when we say "scale-space", we do not mean a pyramid, we mean a varying blur.

Although the Gaussian pyramid alone does not contain sufficient information to reconstruct the original image, we could construct a pyramid that does contain sufficient information. To do that, we use a "Laplacian" pyramid, constructed by computing a similar representation of the image; this preserves the high-frequency information (Fig. 5.11). Combining the two pyramid representations allows reconstruction of the original image.

In a modern scale space representation we preserve the concept that each level is a blurring of the previous level, but do not subsample – each level is the same size as the previous level, but more blurred. Normally, each level is generated by convolving the original image with a Gaussian of variance σ^2, and σ varies from one level to the next. This variance then becomes the "scale parameter." Clearly, at high levels of scale, (σ large), only the largest features are visible. We will see more about scale space later in this chapter, when we talk about wavelets.

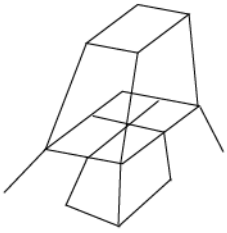


Fig. 5.9. A pyramid is a data structure which is a series of images, in which each pixel is the average of four pixels at the next lower level.

Fig. 5.10. A Gaussian pyramid, constructed by blurring each level with a Gaussian and then 2 : 1 subsampling.

Fig. 5.11. This Laplacian pyramid is actually computed by a difference of Gaussians.

5.9.1 Quad trees

A quad tree [5.21] is a data structure in which images are recursively broken into four blocks, corresponding to nodes in a tree. The four blocks are designated NW (north–west), NE, SW, and SE. The correspondence between the nodes in the tree and the image are best illustrated by an example (see Fig. 5.12).

In encoding binary images, it is straightforward to come up with a scheme for generating the quad tree for an image: If the quadrant is homogeneous (either solid black or solid white), then make it a leaf, otherwise divide it into four quadrants and add another layer to the tree. Repeat recursively until the blocks either reach pixel size or are homogeneous.

It is easy to make a quad tree representation into a pyramid. It is only necessary to keep, at each node, the average of the values of its children. Then, all the information in a pyramid is stored in the quad tree.

If an image has large homogeneous regions, a quad tree would seem to be an efficient way to store and transmit an image. However, experiments with a variety of images, even images which were the difference between two frames in a video

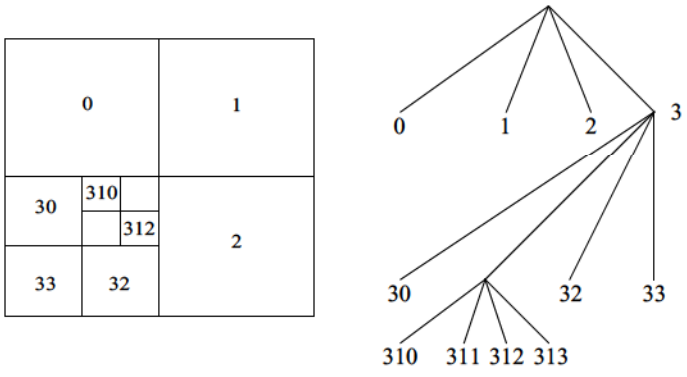


Fig. 5.12. An image is divided into four blocks. Each inhomogeneous block is further divided. This partitioning may be represented by a tree.

sequence, have shown that this is not true. Since the difference image is only nonzero where things are moving, it seems obvious that this, mostly zero, image would be efficiently stored in a quad tree. Not so. Even in that case, the overhead of managing the tree overwhelms the storage gains. So, surprisingly, the quad tree is not an efficient image compression technique. When used as a means for representing a pyramid, it does, however, have advantages as a way of representing scale space.

Another disadvantage of using quad trees is that a slight movement of an object can result in radically different tree representations, that is, the tree representation is not rotation or translation invariant. In fact, it is not even robust. Here, "robust" means a small translation of an object results in a correspondingly small change in the representation. One can get around this problem, to some extent, by not representing the entire image, but instead, representing each object subimage with a quad tree.

The generalization of the quad tree to three dimensions is called an "octree." The same principles apply.

5.9.2 Gaussian scale structures

A good way to remember large and small scale is that at large scale, only large objects may be distinguished.

We know how to blur an image. Here is a *gedankenexperiment* for you: Take an image, blur it with a Gaussian kernel of standard deviation 1. You get a new image. Call that image 1. Now, blur the original image with a Gaussian kernel of standard deviation 2. Call that image 2. Continue until you have a set of images which you can think of as stacked, and the "top" image is almost blurred away. We say the top image is a representation of the image at "large scale." This stack of images is referred to as a "scale space" representation. Clearly, we are not required to use integer values of the standard deviation, so we can create scale space representations with as much resolution in scale as desired. The essential premise of scale space representations is that certain features can be tracked over scale, and how those features vary with scale tells something about the image. A scale space has been formally defined [5.25, 5.26] as having the following properties.

- All signals should be defined on the same domain (no pyramids).
- Increasing values of the scale parameter should produce coarser representations.
- A signal at a coarser level should contain less structure than a signal at a finer level. If one considers the number of local extrema as a measure of smoothness, then the number of extrema should not increase as we go to coarser scale. This property is called "scale space causality."
- All representations should be generated by applications of a convolution kernel to the original image.

The last property is certainly debatable, since convolution formally requires a linear, space-invariant operator. One interesting approach to scale space which violates this requirement is to produce a scale space by using gray scale morphological

smoothing (we will discuss this later) with larger and larger structuring elements [5.16].

You could use scale space concepts to represent texture [4.16] or even a probability density function (in which case, your scale space representation becomes a clustering algorithm [5.24]) as well as brightness. We will see applications of scale representations as we proceed through the course.

One of the most interesting aspects of scale space representations is the behavior of our old friend, the Gaussian. The second derivative of the Gaussian (in two dimensions, the Laplacian of Gaussian: LOG) has been shown [5.27] to have some very nice properties when used as a kernel. In particular, the zero crossings of the LOG are good indicators of the location of an edge. One might be inclined to ask, "Is the Gaussian the best smoothing operator to use to develop a kernel like this?" Said another way: We want a kernel whose second derivative never generates a new zero crossing as we move to larger scale. In fact, we could state this desire in the following more general form.

Let our concept of a "feature" be a point where some operator has an extreme, either maximum or minimum. The concept of scale space causality says that as scale increases, as images become more blurred, new features are never created. The Gaussian is the ONLY kernel (linear operator) with this property [5.1, 5.2]. Studies of nonlinear operators have been done to see under what conditions these operators are scale space causal [5.22].

This idea of scale space causality is illustrated in the following example. Fig. 5.13 illustrates the brightness profile along a single line from an image, and the scale space created by blurring that single line with one-dimensional Gaussians of increasing variance. In Fig. 5.14, we see the Laplacian of the Gaussian, and the points where the Laplacian changes sign. The features in this example, the zero crossings (which are good candidates for edges) are indicated in the right image. Observe that as scale increases, feature points (in this case, zero-crossings) are never created as scale increases. As we go from top (low scale) to bottom (high scale), some features disappear, but no new ones are created.

One obvious application of this idea is to identify the important edges in the image first. We can do that by going up in scale, finding those few edges, and then tracking them down to lower scale.

5.10 Quantifying the accuracy of an edge detector

Since there are many options in the design of an edge detection algorithm, we need some objective ways to say that one edge detector works better than another. Pratt [5.33] has suggested a simple formula to address this question. The formula is just

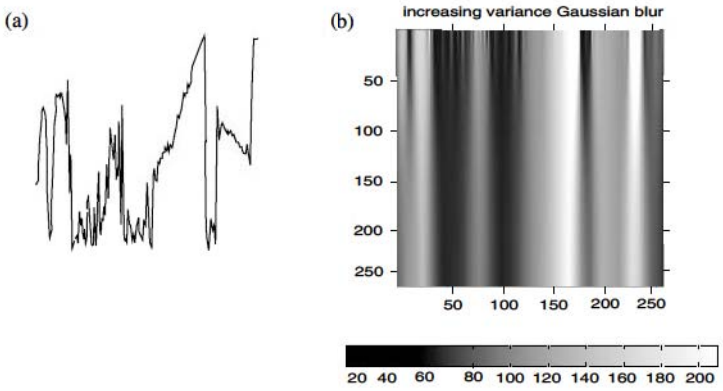


Fig. 5.13. **(a) Brightness profile of a scanline through an image. (b) Scale space representation of that scanline. Scale increases toward the bottom, so no new features should be created as one goes from top to bottom.**

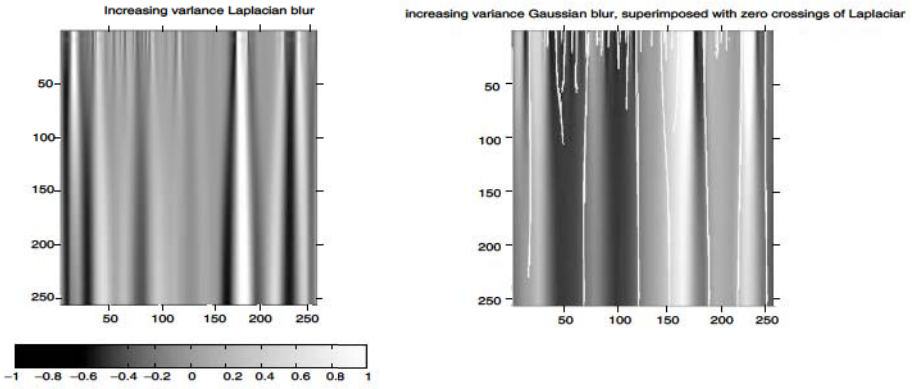


Fig. 5.14. **Laplacian of the scale space representation, and the zero crossings of the Laplacian. Since this is one-dimensional data, there is no difference between the Laplacian and the second derivative.**

a summation over the edge points

$$R = \frac{1}{I_N} \sum_{i=1}^{I_a} \frac{1}{1 + \alpha d^2} \tag{5.36}$$

where $I_N = \max(I_I, I_A)$, I_I denotes the number of edge points detected and I_A is the number of edge points actually in the image. α is a constant scale factor, and d is a measure of the distance from the edge point detected to the nearest actual edge. This formula requires, of course, some knowledge of where the edge points *should* be. Consequently, it is of primary usefulness when applied to synthetic data, since only in such data can we be absolutely sure of the actual position of edge points (see also [5.4]).

5.11 So how do people do it?

Two neurophysiologists, David Hubel and Thorsten Wiesel [5.13, 5.14] stuck some electrodes in the brains – specifically the visual cortex – first of cats[3] and later of monkeys. While recording the firing of neurons, they provided the animal with visual stimuli of various types. They observed some fascinating results: First, there are cells which fire only when specific types of patterns are observed. For example, a particular cell might only fire if it observed an edge, bright to dark, at a particular angle. There was evidence that each of the cells they measured received input from a neighborhood of cells called a "receptive field." There were a variety of types of receptive fields, possibly all connected to the same light detectors, which were organized in such a way as to accomplish edge detection and other processing. Jones and Palmer [5.17] mapped receptive field functions carefully and confirmed [5.9, 5.10] that the function of receptive fields could be accurately represented by *Gabor functions*, which have the form of Eq. (5.37):

$$G(x, y) = \frac{1}{2\pi\sigma\beta} \exp\left(-\pi\left(\frac{x^2}{\sigma^2} + \frac{y^2}{\beta^2}\right)\right) \exp(i[\xi x + \nu y]). \tag{5.37}$$

The first exponential is a two-dimensional Gaussian whose isophotes form ellipses with major and minor axes aligned with the x and y axes. (If you happen to be dealing with a receptive field which is tilted with respect to x and y, you need to rotate your coordinate system to make this equation still hold.) The second (complex) exponential represents a plane wave. Eq. (5.37) assumes the origin is at the center of the Gaussian. Fig. 5.15 illustrates a Gabor filter.

The following interesting observations have been made [5.23] regarding the values of the parameters in Eq. (5.37), when those parameters are actually measured in living organisms:

- The aspect ratio, β/α, of the ellipse is 2 : 1.
- The plane wave tends to propagate along the short axis of the ellipse.
- The half-amplitude bandwidth of the frequency response is about 1 to 1.5 octaves along the optimal orientation.

So do we have Gabor filters in our brains? Or Gabor-like wavelet generators? Well, we don't know about YOUR brain, but Young [5.41] has looked at the stimulus–response characteristics of mammalian retina and observed that those same receptive fields that are so nicely modeled with Gabor filters or wavelets may equally well be described in terms of kernels called "difference of offset Gaussians," which is essentially the LOG with an additive Gaussian offset. Fig. 5.16 illustrates a section

[3] We were going to put a dead cat joke here, something like "One of the cats died during the procedure, but its behavior was unchanged," but the publisher told us people would be offended, so we had to remove it.

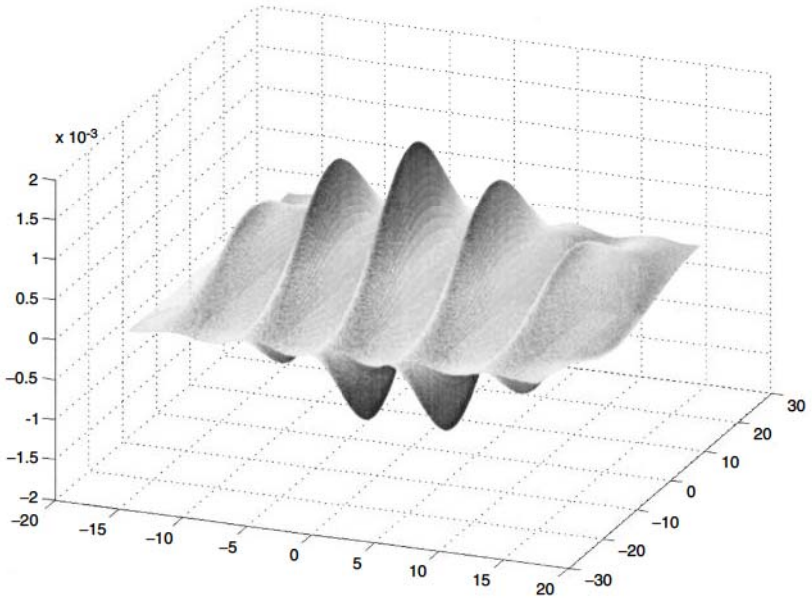


Fig. 5.15. Gabor filter. Note that the positive/negative response is very similar to those that we derived earlier in this chapter.

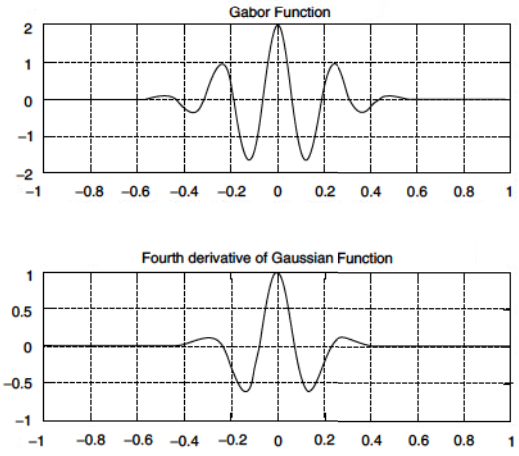


Fig. 5.16. Comparison of a section through a Gabor filter and a similar section through a fourth derivative of a Gaussian.

through a Gabor and a fourth derivative of a Gaussian. You can see noticeable differences, the principal one being that the Gabor goes on forever, whereas the fourth derivative has only three extrema. However, to the precision available to neurology experiments, they are the same. The problem is simply that the measurement of the data is not sufficiently accurate, and it is possible to fit a variety of curves to it.

Bottom line: We have barely a clue how the brain works, and we don't really know very much about the retina. There are two or three mathematical models which adequately model the behavior of receptive fields.

5.12 Conclusion

Consistency in edge detection.

Explicit use of consistency has not been made in this chapter. However, in Assignment 10.1, you will see an application of consistency to edge detection. In that problem, you will be asked to develop an algorithm which makes use of the fact that adjacent edge pixels have parallel gradients. That is, if pixel A is a neighbor of pixel B, and the gradient at pixel A is parallel (or nearly parallel) to the gradient at pixel B, this increases the confidence that both pixels are members of the same edge.

In this chapter, we have looked at several ways to derive kernel operators which, when applied to images, result in strong responses for types of edges.

- We applied the definition of the derivative.

Minimize the sum-squared error.

- We fit an analytic function to a surface, by minimizing the sum squared error.
- We converted subimages into vectors, and projected those vectors onto special basis vectors which described edge-like characteristics.
- We made use of the linearity of kernel operators to interchange the roll of blur and differentiation to construct kernels which are the derivatives of special blurring kernels. We used the constrained optimization and Lagrange multipliers to solve this problem.

Constrained optimization and Lagrange multipliers.

5.13 Vocabulary

You should know the meanings of the following terms.

```
Basis vector
Convolution
Correlation
Gabor filter
Image gradient
Inner product
Kernel operator
Lagrange multiplier
Lexicographic
Linear operator
```

LOG
Projection
Pyramid
Quad tree
Scale space
Sum-squared error

Assignment 5.1

The previous section showed how to estimate the first derivative by fitting a plane. Clearly that will not work for the second derivative, since the second derivative of a plane is zero everywhere. Use the same approach, but use a biquadratic

$$f(x, y) = ax^2 + by^2 + cx + dy + e.$$

Then $[a\ b\ c\ d\ e]^T = A$

$$\begin{bmatrix} x^2 & y^2 & x & y & 1 \end{bmatrix}^T = X.$$

Find the 3×3 kernel which estimates $\frac{\partial^2 f}{\partial x^2}$.

Assignment 5.2

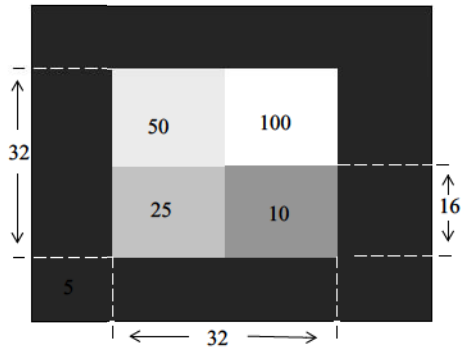
Oh no! Another part of the assignment! (hey, this one is much easier than the one above — a real piece of cake). Using the same methods, find the 5×5 kernel which estimates $\frac{\partial f}{\partial x}$ at the center point, using the equation of a plane.

Assignment 5.3

Determine whether u_1 and u_2 in Fig. 5.4 are in fact orthonormal. If not, recommend a modification or other approach which will allow all the us to be used as basis functions.

Assignment 5.4

(1) Write a program to generate an image which is 64×64, as illustrated below.



The images should contain areas of uniform brightness, with dimensions and brightness as shown. Save this in a file and call it "SYNTH1."

(2) Write a program which reads in SYNTH1, and applies the blurring kernel

$$1/10 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

and write the answer to file named "BLUR1."

(3) Add Gaussian random noise of variance $\sigma^2 = 9$ to BLUR1, and write the output to a file "BLUR1.V1."

Assignment 5.5

Write a program to apply the following two kernels (referred to in the literature as the "Sobel operators") to images SYNTH1, BLUR1, and BLUR1.V1.

$$h_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad h_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

To accomplish this, perform the following.

(1) Apply h_x to the input, save the result as a temporary image in memory (remember that the numbers CAN be negative).
(2) Apply h_y to the input, save the result as another array in memory.
(3) Compute a third array in which each point in the array is the sum of the squares of the corresponding points in the two arrays you just saved. Finally, take the square root of each point. Save the result.
(4) Examine the values you get. Presumably, high values are indicative of edges. Choose a threshold value and compute a new image, which is one whenever the edge strength exceeds your threshold and is zero otherwise.
(5) Apply steps (1)—(4) to the blurred and noisy images as well.
(6) Write a report. Include a printout of all three binary output images. Are any edge points lost? Are any points artificially created? Are any edges too thick? Discuss sensitivity of the result to noise, blur, and choice of threshold.

Be thorough; this is a research course which requires creativity and exploring new ideas, as well as correctly doing the minimum required by the assignment.

Assignment 5.6

In Assignment 5.2, you derived a 5×5 kernel. Repeat Assignment 5.5 using that kernel, for $\partial/\partial x$ and the appropriate version for $\partial/\partial y$.

Assignment 5.7

(1) Verify the mathematics we did in Eq. (5.30). Find a 3×3 kernel which implements the derivative-of-Gaussian vertical edge operator of Eq. (5.30). Use $\sigma = 1$ and $\sigma = 2$ and determine two kernels. Repeat for a 5×5 kernel. Discuss the impact of the choice of σ and its relation to kernel size. Assume the kernel may contain real (floating point) numbers.

(2) Suppose the kernel can only contain integers. Develop kernels which produce approximately the same result.

Assignment 5.8

In section 5.7, parameters useful for developing discrete Gaussian kernels were discussed. Prove that the value of σ such that the maximum of the second derivative occurs at $x = -1$ and $x = 1$. Is $\sigma = 1/\sqrt{3}$?

Assignment 5.9

Use the method of fitting a polynomial to estimate $\partial^2 f/\partial y^2$. Which of the following polynomials would be most appropriate to choose?

(a) $f = ax^2 + by + cxy$ (c) $f = ax^3 + by^3 + cxy$

(b) $f = ax^2 + by^2 + cxy + d$ (d) $f = ax + by + c$

Assignment 5.10

Fit the following expression to pixel data in a 3×3 neighborhood: $f(x, y) = ax^2 + bx + cy + d$. From this fit, determine a kernel which will estimate the second derivative with respect to x.

Assignment 5.11

Use the function $f = ax^2 + by^2 + cxy$ to find a 3×3 kernel which estimates $\partial^2 f/\partial y^2$. Which of the following is the kernel which results? (Note: The following answers do not include the scale factor. Thus, the best choice below will be the one proportional to the correct answer.)

(a)

6	4	0
4	6	0
0	0	4

(c)

2	6	2
−4	0	−4
2	6	2

(e)

6	4	0
4	6	0
0	0	1

(b)

1	−2	1
3	0	3
1	−2	1

(d)

1	0	−1
−2	0	2
1	0	−1

(f)

1	2	1
−2	0	−2
−1	−2	−1

Assignment 5.12

Suppose the kernel that estimates $\partial^2 f/\partial x \partial y$ is

2	−1	0
−1	0	1
0	1	−2

(It may not be true that this kernel estimates that derivative, but it does not affect the answer. Assume it is true.) Is it true then that

4	1	0
1	0	1
0	1	4

estimates $\left(\partial^2 f/\partial x \partial y\right)^2$? Explain your answer.

Topic 5A Edge detectors

The process of edge detection includes more than simply thresholding the gradient. We want to know the location of the edge more precisely than simple gradient thresholds reveal. Two methods which seem to have acquired a substantial reputation in this area are the so called "Canny edge detector" [5.6] and the "facet model" [4.18]. Here, we only describe the Canny edge detection.

5A.1 The Canny edge detector

The edge detection algorithm begins with finding estimates of the gradient magnitude at each point. Canny uses 2×2 rather than 3×3 kernels as we have, but it does not affect the philosophy of the approach. Once we have estimates of the two partial derivatives, we use Eqs. (5.22) through (5.24) to calculate the magnitude and direction of the gradient, producing two images, $M(x, y)$ and THETA (x, y). We now have a result which easily identifies the pixels where the magnitude of the gradient is large. That is not sufficient, however, as we now need to thin the magnitude array, leaving only points which are maxima, creating a new image $N(x, y)$. This process is called nonmaximum[4] suppression (NMS).

NMS may be accomplished in a number of ways. The essential idea, however, is as follows: First, initialize $N(x, y)$ to $M(x, y)$. Then, at each point, (x, y), look one pixel in the direction

[4] This is sometimes written using the plural, nonmaxima suppression. The expression is ambiguous. It could be a compression of suppress every point which is not a maximum, or suppress all points which are not maxima. We choose to use the singular.

of the gradient, and one pixel in the reverse direction. If $M(x, y)$ (the point in question) is not the maximum of these three, set its value in $N(x, y)$ to zero. Otherwise, the value of N is unchanged.

After NMS, we have edges which are properly located and are only one pixel wide. These new edges, however, still suffer from the problems we identified earlier – extra edge points due to noise (false hits) and missing edge points due either to blur or to noise (false misses). Some improvement can be gained by using a dual-threshold approach. Two thresholds are used, τ_1 and τ_2, where τ_2 is significantly larger than τ_1. Application of these two different thresholds to $N(x, y)$ produces two binary edge images, denoted T_1 and T_2 respectively. Since T_1 was created using a lower threshold, it will contain more false hits than T_2. Points in T_2 are therefore considered to be parts of true edges. Connected points in T_2 are copied to the output edge image. When the end of an edge is found, points are sought in T_1 which could be continuations of the edge. The continuation is continued until it connects with another T_2 edge point, or no connected T_1 points are found.

In [5.6] Canny also illustrates some clever approximations which provide significant speedups.

5A.2 Improvements to edge detection

The derivative of $g(x)$ is the second derivative of the image, so look for zero crossings.

Tagare and deFigueiredo [5.34] (see also [5.1]) describe the process of edge detection as follows.

(1) The input is convolved with a filter which smoothly differentiates the input and produces high values at and near the location of the edge. The output $g(x)$ is the sum of the differentiated step edge and filtered noise.
(2) A decision mechanism isolates regions where the output of the filter is significantly higher than that due to noise.
(3) A mechanism identifies the zero crossing in the derivative of $g(x)$ in the isolated region and declares it to be the location of the edge.

A Gaussian low-pass filter followed by finding a zero in the (second) derivative accurately (to subpixel resolution) finds the exact location of the edge, but only if the edge is straight [5.38]. If the edge is curved, errors are introduced. For example, the second derivative in the gradient direction (SDGD) and the Laplacian both make errors in estimating the location of the edge, but interestingly, in opposite directions, which prompts Verbeek and van Vliet [5.38] to suggest an operator which is the sum of the two.

All the methods cited or described in this section perform signal processing in a direction normal to the edge [5.18] in order to better locate the actual edge. Taratorin and Sideman [5.35] present a way to make use of the fact that the images are known to have properties such as positivity and finite support to improve the accuracy with which derivatives may be estimated. Iverson and Zucker [5.15] improve the results of the Canny by adding logical/Boolean reasoning. This improves edge detection over simple thresholding of the derivative, but does not provide results as good as active contours (see Chapter 9) or optimization (see Chapter 6). There are a wide variety of papers on signal processing techniques applied to edge detection [5.36].

Examination of the literature in biological imaging systems, all the way back to the pioneering work of Hubel and Wiesel in the 1960s [5.13] suggests that biological systems analyze images by making local measurements which quantify orientation, scale, and motion. Keeping this in mind, suppose we wish to ask a question like "is there an edge at orientation θ at this point?" How might we construct a kernel that is specifically sensitive to edges at that orientation? A straightforward approach [5.37] is to construct a weighted sum of the two Gaussian first derivative kernels, G_x and G_y, using a weighting something like

$$G_\theta = G_x \cos\theta + G_y \sin\theta. \tag{5.38}$$

Could you calculate the orientation selectivity? What is the smallest angular difference you could detect with a 3×3 kernel determined in this way?

Unfortunately, unless quite large kernels are used, the kernels obtained in this way have rather poor orientation selectivity. In the event that we wish to differentiate across scale, the problem is even worse, since a scale space representation is normally computed rather coarsely, to minimize computation time. Perona [5.31] provides an approach to solving these problems.

5A.3 Inferring line segments from edge points

After we have chosen the very best operators to estimate derivatives, have chosen the best thresholds, and selected the best estimates of edge position, we still have nothing for a set of pixels, some of whom have been marked as probably part of an edge. If those points are adjacent, one could "walk" from one pixel to the next, eventually circumnavigating a region, and there are representations such as the chain code which make this process easy. However, the points are unlikely to be connected the way we would like them to be. Some points may be missing due to blur, noise, or partial occlusion. There are many ways to approach this problem, including relaxation labeling, and parametric transforms, both of which will be discussed in detail later in this book. In addition, there are combination methods, such as the work of Deng and Iyengar [5.11] which combines relaxation and Bayes' methods as well as other methods [5.29] which we do not have space to discuss.

5A.4 Space/frequency representations

Wavelets are very important, but a thorough examination of this area is beyond the scope of this book. Therefore we present only a rather superficial description here, and provide some pointers to literature. For example, Castleman [4.6] has a readable chapter on wavelets.

5A.4.1 Why wavelets?

Consider the image illustrated in Fig. 5.17. Clearly, the spatial frequencies appear to be different, depending on where one looks in the image. The Fourier transform has no mechanism for capturing this intuitive need to represent both frequency and location. The Fourier transform of this image will be a two-dimensional array of numbers, representing the amount of energy *in the entire image* at each spatial frequency. Clearly, since the Fourier transform is invertible, it captures all this spatial and frequency information, but there is no obvious way to answer the question: at each position, what are the local spatial frequencies?

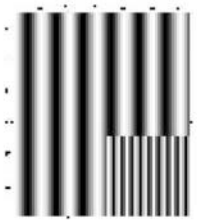


Fig. 5.17. An image in which spatial frequencies vary dramatically.

The wavelet approach is to add a degree of freedom to the representation. Since the Fourier transform is complete and invertible, all we really need to characterize the image is a single two-dimensional array. Instead however, following the space/frequency philosophy as described in section 5.8, we use a three-(or higher) dimensional data structure. In this sense, the space/frequency representation is redundant (or *overcomplete*), and requires significantly more storage than the Fourier transform.

5A.4.2 The basic wavelet and wavelet transform

We define a basic[5] wavelet $\psi(x, y)$ as any function of the two spatial variables x and y, which meets a certain criterion that we need not concern ourselves with here. Basically, we desire a function which is symmetric about the origin and has *almost finite support*. By "almost finite support," we mean that the magnitude of this function drops off to zero rapidly (in a particular way defined by the admissibility criterion) as it goes away from its center. A one-dimensional example basic wavelet is

$$\psi(x) = \frac{2}{\sqrt{3}\sqrt{\pi}}(1 - x^2)\exp\left(-\frac{x^2}{2}\right) \tag{5.39}$$

graphed in Fig. 5.18. From one such wavelet, one may then generate a (potentially infinite) set of similar functions by translation and scaling of the original. That is, we define a translated, scaled version of ψ by (again, in one dimension)

$$\psi_{a,b}(x) = \frac{1}{\sqrt{a}}\psi\left(\frac{x - b}{a}\right). \tag{5.40}$$

The wavelet transform of a function f is computed by the inner products of f with the wavelet at each of the possible values of a and b.

$$W_f(a, b) = \int_{-\infty}^{\infty} f(x)\psi_{a,b}(x)\,dx. \tag{5.41}$$

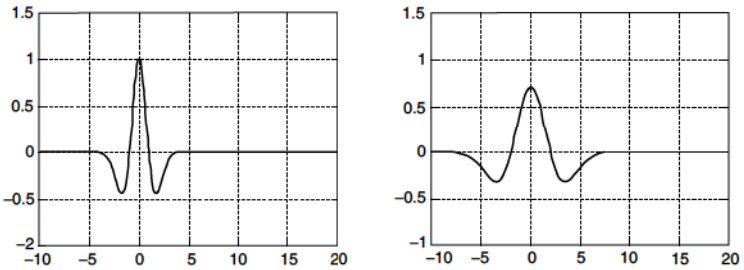


Fig. 5.18. A basic wavelet, and a wavelet generated by scale change.

[5] The basic wavelet is often referred to as the *mother* wavelet.

Fig. 5.19. Original and the result of the inner product of the original and three different two-dimensional wavelets (three slices through the wavelet transform).

Observe that the transform is a function of the scale and the shift. The same idea holds in two dimensions, where the equation for the inner product becomes

$$W_f(a, b_x, b_y) = \int_{-\infty}^{\infty} f(x, y)\psi_{a,b_x,b_y}(x, y)\, dx\, dy. \tag{5.42}$$

Fig. 5.19 illustrates a cross section of W for different values of a. Clearly, this process produces a scale space representation.

Lee [5.23] takes the neurophysiological evidence and derives a mother wavelet of the following form.

$$\psi(x, y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\left(\frac{4x^2 + y^2}{8}\right)\right)\left[\exp(i\kappa x) - \exp\left(-\frac{\kappa^2}{2}\right)\right], \tag{5.43}$$

where κ is a constant whose value depends on assumptions about the bandwidth, but is approximately equal to 3.5. Scaling and translating this "mother wavelet" produces a collection of filters which (in the same way that the Frei–Chen basis set did) represents how much of an image neighborhood resembles the feature, and from which the image can be completely reconstructed.

5A.5 Vocabulary

You should know the meaning of the following terms.

```
Canny edge detector
Nonmaximum suppression
Wavelet
```

Assignment 5.A1

```
At high levels of scale, only            objects are
visible. (Fill in the blank.)
```

Assignment 5.A2

What does the following expression estimate? $(f \otimes h_1)^2 + (f \otimes h_2)^2 = E$, where the kernel h_1 is defined by

0.05	0.08	0.05
–0.136	–0.225	–0.136
0.05	0.08	0.05

,

and h_2 is created by the transpose.

Choose the best answer:

(a) the first derivative with respect to x (where x is the horizontal direction)
(b) the second derivative with respect to x
(c) the Laplacian
(d) the second derivative with respect to y
(e) the quadratic variation.

Assignment 5.A3

You are to use the idea of differentiating a Gaussian to derive a kernel. What variance does a one-dimensional (zero mean) Gaussian need to have to have the property that the extrema of its first derivative occur at $x = \pm 1$?

Assignment 5.A4

What is the advantage of the quadratic variation as compared to the Laplacian?

Assignment 5.A5

Let $E = (f - Hg)^T (f - Hg)$. Using the equivalence between the kernel form of a linear operator and the matrix form, write an expression for E using kernel notation.

Assignment 5.A6

The purpose of this assignment is to walk you through the construction of an image pyramid, so that you will more fully understand the potential utility of this data structure, and can use it in a coding and transmission application. Along the way, you will pick up some additional understanding of the general area of image coding. Coding is not a principal learning objective of this course, but in your research career, you are sure to encounter people who deal with coding

all the time. It will prove to be useful to have a grasp of some of the more basic concepts.

(1) Locate the image "asterix512.ifs," and verify that it is indeed 512×512 (use ifs info). If it is not 512×512, then first write a program which will pad it out to that size.

(2) Write a **program** which will take a two-dimensional $n \times n$ image and produce an image, of the same data type, which is $n/2 \times n/2$. The program name should be **ShrinkByTwo** and called by

ShrinkByTwo inimg outimg

The program should NOT simply take every other pixel. Instead, each pixel of the output image should be constructed by averaging the corresponding four pixels in the input image. Note the requirement that the output image be of the same type as the input. That is easily accomplished by using **ifscreate** with the first argument being the character string defining the type of the input. For example ifscreate (in->ifsdt, len, IFS_CR_ALL,0). Use your program to create asterix256, asterix128, asterix64, and asterix32. Don't bother going below a 32×32 image. Turn in your program and a printout of your images.

(3) Write a **subroutine** to zoom an image. It should have the following calling convention:

```
ZoomByTwo(inimg,outimg)
IFSIMG inimg, outimg;
```

The calling program is responsible for creating, reading, etc. of the images. The subroutine simply fills outimg with a zoomed version of inimg. Use any algorithm you wish to fill in the missing pixels. (We recommend that the missing pixels be some average of the input pixels.)

Before we can proceed, we need to consider a "pyramid coder." When you ran ShrinkByTwo, the set of images you produced is the pyramid representation of asterix512. In a pyramid coder, the objective is to use the pyramid representation to transmit as little information as possible over the channel. Here is the idea: First, transmit all of asterix32. Then, both the transmitter and receiver run ZoomByTwo to create a zoomed version of asterix32, something like

ZoomByTwo(a32, a64prime)

When we created asterix32 from asterix64, we threw away some information, and we cannot easily get it back, so a64prime will not be identical to asterix64. If, however, ZoomByTwo (which in the image coding literature is called the *predictor*) is pretty good, the difference between a64prime and asterix64 will be small (small in value that is; it is still 64×64). Therefore, compute diff64, the difference between a64prime and asterix64. If the predictor were perfect, the resultant difference would be a 64×64 image of all zeros, which could be coded in some clever way (by run-length encoding for example), and transmitted with very few bits. Let us now transmit diff64 to the receiver. By simply adding diff64 to the version of a64prime generated at the receiver, we can correct the errors made by the predictor, and we now have a correct version of asterix64 at the receiver, but all we transmitted was diff64. Clever, aren't we? Now, from asterix64, we play the same game and get asterix128 created by transmitting diff128, *et cetera, ad nauseam*. Now, the assignment:

(4) Create the images described above, diff64, diff128, diff256, diff512. Measure the approximate number of bits required to transmit each of them. To make that measurement: Compute the standard deviation of, say, diff64. Take the log base 2 of that standard deviation, and that is the average number of bits per pixel required to code that image. Assume you were to transmit asterix512 directly. That would require $512 \times 512 \times 8$ bits (assuming the image is 8 bit — you better verify). Now, compare with the performance of your pyramid coder by adding up all the bits/pixel you found for each of the diff images you transmitted. Did your coder work well? Discuss this in your report.

References

[5.1] V. Anh, J. Shi, and H. Tsai, "Scaling Theorems for Zero Crossings of Bandlimited Signals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(3). 1996.

[5.2] J. Babaud, A. Witkin, M. Baudin, and R. Duda, "Uniqueness of the Gaussian Kernel for Scale-space Filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **8**(1), 1986.

[5.3] G. Bilbro and W. Snyder, "Optimization of Functions with Many Minima," *IEEE Transactions on Systems, Man, and Cybernetics*, **21**(4), July/August, 1991.

[5.4] K. Boyer and S. Sarkar, "On the Localization Performance Measure and Optimal Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(1), 1994.

[5.5] P. Burt and E. Adelson, "The Laplacian Pyramid as a Compact Image Code," *Computer Vision, Graphics, and Image Processing*, **16**, pp. 20–51, 1981.

[5.6] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **8**(6), 1986.

[5.7] W. Chojnacki, M. Brooks, A. van der Hengel, and D. Gawley, "On the Fitting of Surfaces to Data with Covariances," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(11), 2000.

[5.8] J. Crowley, *A Representation for Visual Information*, Ph.D. Thesis, Carnegie-Mellon University, 1981.

[5.9] J. Daugman, "Two-dimensional Spectral Analysis of Cortical Receptive Fields," *Vision Research*, **20**, pp. 847–856, 1980.

[5.10] J. Daugman, "Uncertainly Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-dimensional Visual Cortical Filters," *Journal of the Optical Society of America*, **2**(7), 1985.

[5.11] W. Deng and S. Iyengar, "A New Probabilistic Scheme and Its Application to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(4), 1996.

[5.12] W. Frei and C. Chen, "Fast Boundary Detection: A Generalization and a New Algorithm," *IEEE Transactions on Computers*, **26**(2), 1977.

[5.13] D. Hubel and T. Wiesel, "Receptive Fields, Binocular Interaction, and Functional Architecture in the Cat's Visual Cortex," *Journal of Physiology (London)*, **160**, pp. 106–154, 1962.

[5.14] D. Hubel and T. Wiesel, 'Functional Architecture of Macaque Monkey Visual Cortex," *Proceedings of the Royal Society of London, B*, **198**, pp. 1–59, 1977.

[5.15] L. Iverson and S. Zucker, "Logical/Linear Operators for Image Curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(10), 1995.

[5.16] P. Jackway and M. Deriche, "Scale-space Properties of the Multiscale Morphological Dilation–Erosion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(1), 1996.

[5.17] J. Jones and L. Palmer, "An Evaluation of the Two-dimensional Gabor Filter Model of Simple Receptive Fields in the Cat Striate Cortex," *Journal of Neurophysiology*, **58**, pp. 1233–1258, 1987.

[5.18] E. Joseph and T. Pavlidis, "Bar Code Waveform Recognition using Peak Locations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(6), 1994.

[5.19] M. Kelly, in *Machine Intelligence*, volume 6, University of Edinburgh Press, 1971.

[5.20] M. Kisworo, S. Venkatesh, and G. West, "Modeling Edges at Subpixel Accuracy using the Local Energy Approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(4), 1994.

[5.21] A. Klinger, "Pattern and Search Statistics," in *Optimizing Methods in Statistics*, New York, Academic Press, 1971.

[5.22] P. Kube and P. Perona, "Scale-space Properties of Quadratic Feature Detectors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(10), 1996.

[5.23] T. Lee, "Image Representation Using 2-D Gabor Wavelets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(10), 1996.

[5.24] Y. Leung, J. Zhang, and Z. Xu, "Clustering by Scale-space Filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(12), 2000.

[5.25] T. Lindeberg, "Scale-space for Discrete Signals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**(3), 1990.

[5.26] T. Lindeberg, "Scale-space Theory, A Basic Tool for Analysing Structures at Different Scales," *Journal of Applied Statistics*, **21**(2), 1994.

[5.27] D. Marr and E. Hildreth, "Theory of Edge Detection," *Proceedings of the Royal Society of London, B*, **207**, pp. 187–217, 1980.

[5.28] D. Marr and T. Poggio, "A Computational Theory of Human Stereo Vision," *Proceedings of the Royal Society of London, B*, **204**, pp. 301–328, 1979.

[5.29] R. Nelson, "Finding Line Segments by Stick Growing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(5), 1994.

[5.30] E. Pauwels, L. Van Gool, P. Fiddelaers, and T. Moons, "An Extended Class of Scale-invariant and Recursive Scale Space Filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(7), 1995.

[5.31] P. Perona, "Deformable Kernels for Early Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(5), 1995.

[5.32] P. Perona and J. Malik, "Scale-space and Edge Detection using Anisotropic Diffusion", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**(7), pp. 629–639, 1990.

[5.33] W. Pratt, *Digital Image Processing*, Chichester, John Wiley and Sons, 1978.

[5.34] H. Tagare and R. deFigueiredo, "Reply to 'On the Localization Performance Measure and Optimal Edge Detection'," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(1), 1994.

[5.35] A. Taratorin and S. Sideman, "Constrained Regularized Differentiation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(1), 1994.

[5.36] F. van der Heijden, "Edge and Line Feature Extraction Based on Covariance Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(1), 1995.

[5.37] M. Van Horn, W. Snyder, and D. Herrington, "A Radial Filtering Scheme Applied to Intracoronary Ultrasound Images," *Computers in Cardiology*, September, 1993.

[5.38] P. Verbeek and L. van Vliet, "On the Location Error of Curved Edges in Low-pass Filtered 2-D and 3-D Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(7), 1994.

[5.39] I. Weiss, "High-order Differentiation Filters that Work," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(7), 1994.

[5.40] M. Werman and Z. Geyzel, "Fitting a Second Degree Curve in the Presence of Error," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(2), 1995.

[5.41] R. Young, "The Gaussian Derivative Model for Spatial Vision: I. Retinal Mechanisms," *Spatial Vision*, **2**, pp. 273–293, 1987.