CHAPTER 5

Applications of Convolution

5.1 FINDING PATTERNS

5.1.1 Representing Images with Filter Banks

In the image in Figure 4.3, the leaves of the pineapple plant look like disorganized thick stripes. The leaves of the plant at its base are quite different, and look more like repeated small spots. These are examples of textures – somewhat unstructured patterns that are quite characteristic. Textures are widespread and quite distinctive – a field of pebbles looks quite different from a stand of corn; a cluster of pine needles looks very different from an expanse of bark; and so on.

Figure 4.3 also suggests a way to represent textures, and so images. Think of a texture as a collection of small patterns, arranged in some distinctive way. An image region showing a field of pebbles would have many spots, some small, some large and most medium, but very few thin bars. In contrast, an image region showing a cluster of pine needles would have many thin bars, pointing in about the direction, but very few small or large spots. Then to build an image representation: (a) construct a vocabulary of patterns; (b) find out which patterns are present at which pixel; and then (c) building a summary of which patterns are present in a region.

Because the patterns are likely so variable, an elaborate or detailed pattern detector is likely to be unhelpful – something that precisely detects a pine needle would need to be tuned to exactly the right angle, which would be a nuisance – so it is natural to use filters as pattern detectors. However, it is helpful to distinguish between, say, light thin bars on dark backgrounds (possible pine needles) and dark thin bars on light backgrounds (possible gaps between needles).

For the moment, assume the vocabulary of patterns is given, represented as a filter bank. Then the next two steps are straightforward. To find the patterns in an image, construct the response of all the filters at all points and apply a ReLU. Stack these responses into a multi-channel image. To compute a summary, construct a local weighted average of each channel of the multi-channel image at each pixel.

5.1.2 Computing Image Gradients with Finite Differences

For an image $\mathcal{I}$, the gradient is

$$\nabla \mathcal{I} = (\frac{\partial \mathcal{I}}{\partial x}, \frac{\partial \mathcal{I}}{\partial y})^T,$$

which we could estimate by observing that

$$\frac{\partial \mathcal{I}}{\partial x} = \lim_{\delta x \to 0} \frac{\mathcal{I}(x + \delta x, y) - \mathcal{I}(x, y)}{\delta x} \approx \mathcal{I}_{i+1,j} - \mathcal{I}_{i,j}.$$

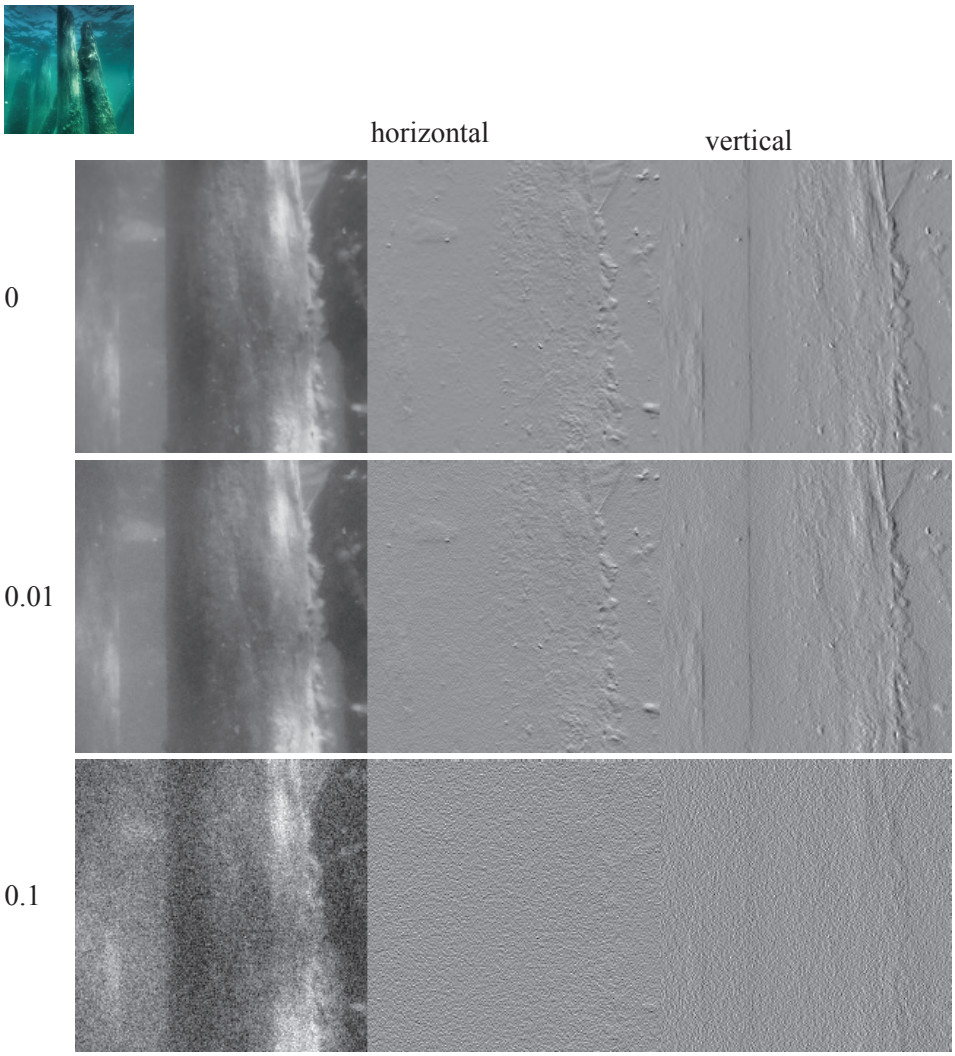

FIGURE 5.1: *Finite differences yield reasonable derivative estimates, but are strongly affected by noise.* **Top left** *shows the original image, from which a detail window is extracted and turned monochrome.* **Rows** *show image, horizontal derivative and vertical derivative, where derivatives are estimated by finite differences.* **First row** *is noise free image; others have additive Gaussian noise added, with standard deviation shown. Notice how this noise affects derivatives. The derivatives are scaled so that positive values are bright, negative values are dark, and 0 is mid-range. However the scale is chosen* per row*, which means the figure understates the effect of noise. In the noisy rows, the largest magnitude derivatives are much larger than in the clean row, which is why you can hardly see significant derivatives in the bottom row.* Image credit: *Figure shows Robert Forsyth's photograph of historical dock pilings in Lake Michigan.*

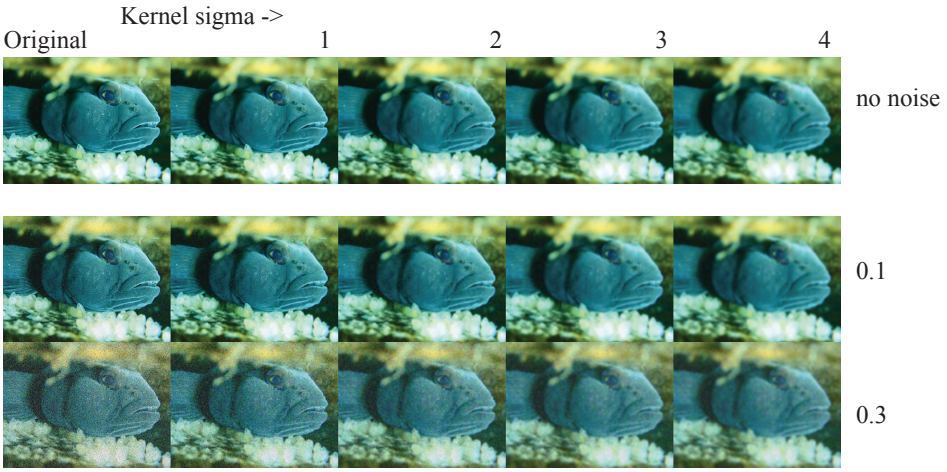

FIGURE 5.2: *Smoothing an image with a gaussian kernel is an effective way to suppress additive Gaussian noise.* **Left column** *the original image, followed by versions smoothed with a gaussian kernel with* $\sigma = 1$, $\sigma = 2$, $\sigma = 3$ *and* $\sigma = 4$. **Top row** *shows results on a noise free image;* **middle row** *shows results on an image with additive stationary gaussian noise with standard deviation 0.01, where the value of a pixel ranges from 0 to 1;* **bottom row** *shows results on an image with additive stationary gaussian noise with standard deviation 0.1. Notice how (a) smoothing blurs the original image; (b) more smoothing leads to more blur; (c) smoothing suppresses noise (so a smoothed version of a noisy image is close to the smoothed version of the original); and (d) more smoothing suppresses more noise.* Image credit: *Figure shows Robert Forsyth's photograph of a goby on its nest in Lake Michigan.*

This means a convolution with

$$-1 \quad 1$$

will estimate $\partial \mathcal{I}/\partial x$ (nothing in the definition requires convolution with a square kernel). Notice that this kernel "looks like" a dark pixel next to a light pixel, and will respond most strongly to that pattern. By the same argument, $\partial \mathcal{I}/\partial y \approx \mathcal{I}_{i,j+1} - \mathcal{I}_{i,j}$. These kinds of derivative estimates are known as *finite differences*. most unsatisfactory estimate of the derivative. This is because finite differences respond strongly (i.e., have an output with large magnitude) at fast changes, and fast changes are characteristic of noise. Roughly, this is because image pixels tend to look like one another. For example, if we had bought a discount camera with some pixels that were stuck at either black or white, the output of the finite difference process would be large at those pixels because they are, in general, substantially different from their neighbors. All this suggests that some form of noise suppression is appropriate before differentiation.

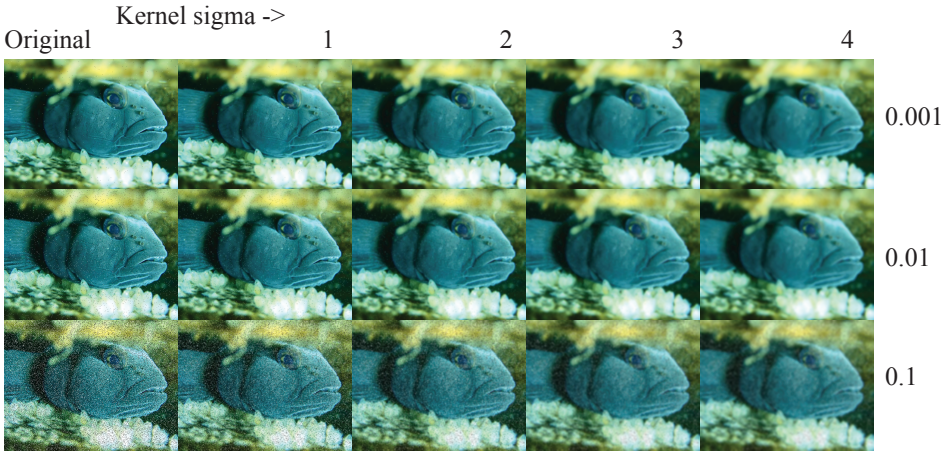

FIGURE 5.3: *Images at various noise levels smoothed with various gaussian kernels. The noise here involves picking pixel locations uniformly at random in the image, then flipping them either full light or full dark. The number on the* **far right** *shows the probability of a pixel being flipped (so at* 0.001*, a* 30×30 *window should have about one flipped pixel in it; at* 0.01*, a* 10×10 *window should have about one flipped pixel in it; and at* 0.1*, a* 3×3 *window should have one flipped pixel in it).* **Left** *the original image, followed by versions smoothed with* $\sigma = 1$*,* $\sigma = 2$*,* $\sigma = 3$ *and* $\sigma = 4$*. Notice how (a) smoothing blurs the original image; (b) more smoothing leads to more blur; (c) smoothing suppresses noise (so a smoothed version of a noisy image is close to the smoothed version of the original); and (d) more smoothing suppresses more noise. The noise-free image is top left in Figure 5.2.* Image credit: *Figure shows Robert Forsyth's photograph of a goby on its nest in Lake Michigan.*

> **Remember this:** *Images can be represented using the outputs of multiple filters, formed into a bank. Convolutions with simple filter kernels will find the* x*- and* y*-derivatives of an image, yielding the image gradient. This gradient estimate is significantly affected by image noise.*

5.2 DENOISING

5.2.1 Suppressing Noise with Filters

The simplest model of image noise is the *additive stationary Gaussian noise* (or *Gaussian noise*) model, where each pixel has added to it a value chosen independently from the same normal (Gaussian – same Gauss, different sense) probability distribution. This distribution almost always has zero mean. The standard deviation is a parameter of the model. Figure 5.2 shows some examples of additive stationary Gaussian noise.

Images can be quite effectively denoised because "pixels look like their neighbors". This important and very reliable slogan is in scare quotes because, while it is an extremely important practical guide, making it precise is neither easy nor particularly useful. Generally, pictures show objects which are span a large number of pixels, and where the shading changes relatively slowly over the surface of the object. This means that the value at a pixel is likely to be close to the value at its neighbor. Although this isn't true of every pixel – otherwise there wouldn't be edges in images – it is true of most pixels. If you have a pixel whose value is unknown, looking at its neighbors will almost always yield a good estimate. A pixel that does not look like its neighbors is suspect.

Check that the smoothing used in the downsampling strategy of Section 2.3.4 is a convolution with a gaussian kernel. This procedure is called *gaussian smoothing* or very often just *smoothing*. It turns out that this procedure is very good at suppressing many kinds of image noise. Figure 5.2 shows examples of suppressing additive Gaussian noise, and the **exercises** explore some details. Gaussian smoothing can suppress the effects of other noise processes, too (Figure 5.3).

The choice of σ (or scale) for the Gaussian follows from the following considerations:

- If the standard deviation of the Gaussian is very small—say, smaller than one pixel—the smoothing will have little effect because the weights for all pixels off the center will be very small.

- For a larger standard deviation, the neighboring pixels will have larger weights in the weighted average, which in turn means that the average will be strongly biased toward a consensus of the neighbors. This will be a good estimate of a pixel's value, and the noise will largely disappear at the cost of some blurring.

- Finally, a kernel that has a large standard deviation will cause much of the image detail to disappear, along with the noise.

Most image noise tends to result in pixels not looking like their neighbors. However, gaussian smoothing is not always effective at estimating the true value of noisy pixels. For example, look closely at Figure 5.3. The noise process – a *Poisson noise process*, sometimes called *salt and pepper noise* – picks pixel locations uniformly at random in the image, then flips the result either full light or full dark. This means that a noisy pixel contains no information, and might be very different from its neighbors. If you compute a weighted average in a region that contains a noisy pixel, that weighted average might be severely disrupted by the noise, even if the center is a clean pixel. For example, think of a dark neighborhood on the goby where noise has turned one pixel bright – the bright pixel will dominate the average unless it contains a very large number of pixels with quite large weights. And in that case, the image will be blurry.

This suggests the entirely natural alternative of computing a median in a neighborhood as an estimate of the value at a pixel. As Figure **??** shows, this can be very effective at suppressing noise. Notice an attractive feature of the median filter – it tends not to blur edges, even when it strongly smoothes the interior of image regions. Median filters are somewhat more expensive computationally than

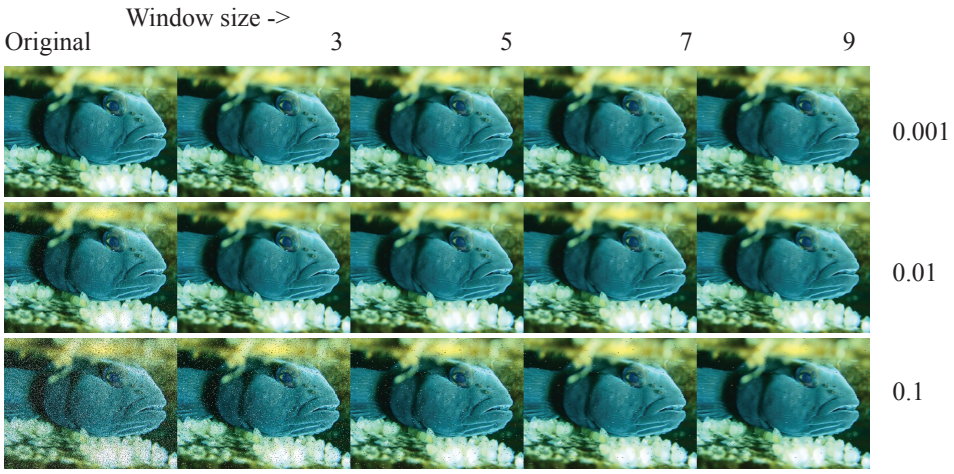

FIGURE 5.4: *Images at various noise levels smoothed with a median filter. The noise here involves picking pixel locations uniformly at random in the image, then flipping them either full light or full dark. The number on the* **far right** *shows the probability of a pixel being flipped (so at* 0.001*, a* 30×30 *window should have about one flipped pixel in it; at* 0.01*, a* 10×10 *window should have about one flipped pixel in it; and at* 0.1*, a* 3×3 *window should have one flipped pixel in it.* **Left** *the original image, followed by versions where the median is taken in windows of different sizes. Notice how (a) the median filter preserves edges rather well, even over big windows; (b) bigger windows lead to more noise suppression; and (c) texture details are suppressed by the median, with bigger windows suppressing more.* Image credit: *Figure shows Robert Forsyth's photograph of a goby on its nest in Lake Michigan.*

smoothing, but deal fairly well with additive gaussian noise as well as salt and pepper noise (Figure 5.5)

5.2.2 Application: Derivative of Gaussian Filters

Because convolution is associative, smoothing an image and then differentiating it is the same as convolving it with the derivative of a smoothing kernel. First, differentiation is linear and shift invariant. This means that there is some kernel that differentiates. Given a function $I(x, y)$,

$$\frac{\partial I}{\partial x} = K_{(\partial/\partial x)} * I.$$

Write the convolution kernel for the smoothing as S. Now

$$(K_{(\partial/\partial x)} * (S * I)) = (K_{(\partial/\partial x)} * S) * I = (\frac{\partial S}{\partial x}) * I.$$

Usually, the smoothing function is a gaussian, so an estimate of the derivative can be obtained by convolving with the derivative of the gaussian (rather than convolve

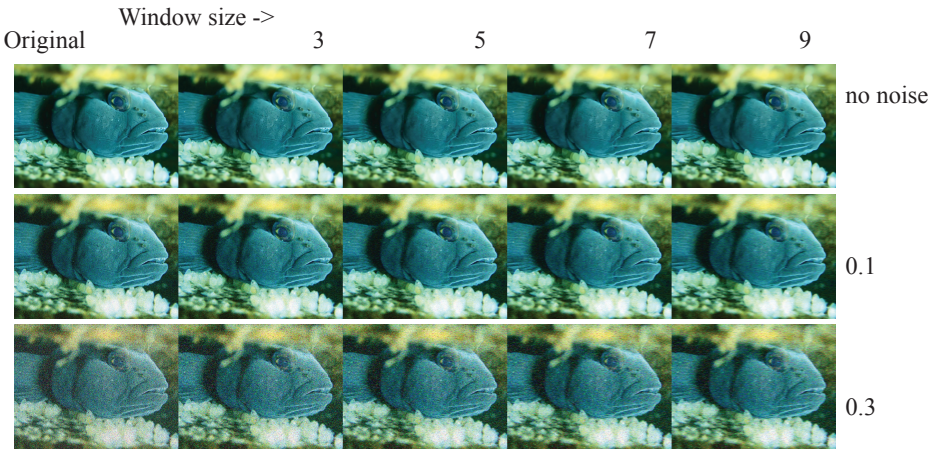

FIGURE 5.5: *Images at various noise levels smoothed with a median filter. The noise here is additive Gaussian noise.* **Left** *the original image, followed by versions where the median is taken in windows of different sizes.* **Top row** *shows results on a noise free image;* **middle row** *shows results on an image with additive stationary gaussian noise with standard deviation 0.01 (where the value of a pixel ranges from 0 to 1);* **bottom row** *shows results on an image with additive stationary gaussian noise with standard deviation 0.1. Notice how (a) the median filter preserves edges rather well, even over big windows; (b) bigger windows lead to more noise suppression; and (c) texture details are suppressed by the median, with bigger windows suppressing more.* Image credit: *Figure shows Robert Forsyth's photograph of a goby on its nest in Lake Michigan.*

and then differentiate), yielding

$$\begin{aligned}\frac{\partial g_\sigma}{\partial x} &= \frac{1}{2\pi\sigma^2}\left[\frac{-x}{2\sigma^2}\right]\exp -\left(\frac{x^2+y^2}{2\sigma^2}\right)\\ \frac{\partial g_\sigma}{\partial y} &= \frac{1}{2\pi\sigma^2}\left[\frac{-y}{2\sigma^2}\right]\exp -\left(\frac{x^2+y^2}{2\sigma^2}\right)\end{aligned}$$

As they should (Section **??**), derivative of gaussian filters look like the effects they are intended to detect. The x-derivative filters look like a vertical light blob next to a vertical dark blob (an arrangement where there is a large x-derivative), and so on.

Large gradients in images are interesting (Chapters **??** and **??**) because they tend to occur on the outlines of objects, at shadow boundaries, and so on. Generally, if there is a large x derivative at a pixel, there will be a large x derivative at neighboring pixels. Smoothing *across* the direction of the derivative may result in smeared or blurred derivatives; but smoothing *along* the direction of the derivative will tend to average the value at points with similar derivatives and improve the

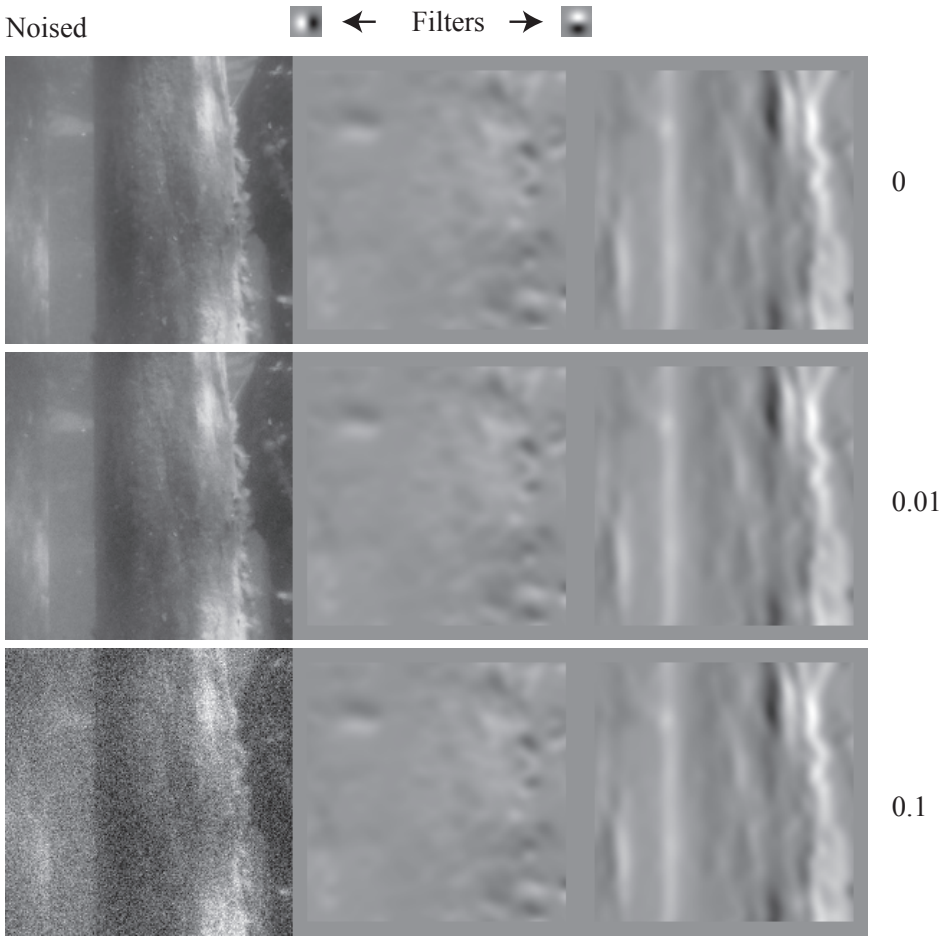

FIGURE 5.6: *Derivative of gaussian filters, equivalent to applying a finite difference to a smoothed image, very significantly improve estimates of derivatives. Compare this figure with Figure 5.1.* **Rows** *show image, horizontal derivative and vertical derivative, where derivatives are estimated by convolution with difference of gaussian filters. The filters are shown at the top. As you should expect, one looks like a dark bar next to a light bar, the other looks like a dark bar below a light bar. As should be apparent from the filters, the smoothing is the same in each direction.* **First row** *is noise free image; others have additive Gaussian noise added, with standard deviation shown on the* **right***. Notice how this noise hardly affects derivatives. The derivatives are scaled so that positive values are bright, negative values are dark, and 0 is mid-range. Although the scale is chosen* per row*, the derivative images look the same from row to row – this means that each row has about the same largest magnitude value.* Image credit: *Figure shows Robert Forsyth's photograph of historical dock pilings in Lake Michigan.*

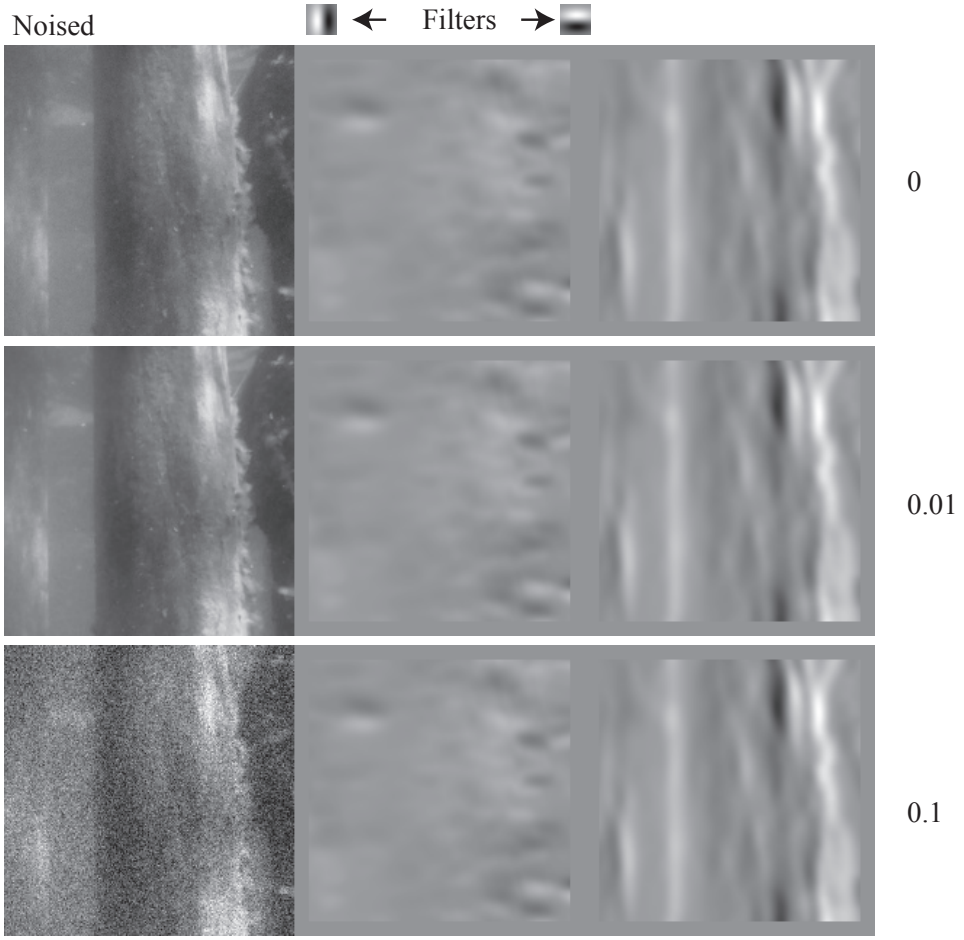

FIGURE 5.7: *Derivative of gaussian filters, equivalent to applying a finite difference to a smoothed image, very significantly improve estimates of derivatives. Compare this figure with Figure 5.1. As should be apparent from the filters, the smoothing is over a much larger range along the derivative direction than across it (compare Figure 5.6).* Image credit: *Figure shows Robert Forsyth's photograph of historical dock pilings in Lake Michigan.*

noise resistance. It is quite usual to use

$$\begin{aligned}\frac{\partial g_\sigma}{\partial x} &= \frac{1}{2\pi\sigma_s\sigma_b}\left[\frac{-x}{2\sigma_s^2}\right]\exp -\left(\frac{x^2}{2\sigma_b^2}+\frac{y^2}{2\sigma_s^2}\right)\\ \frac{\partial g_\sigma}{\partial y} &= \frac{1}{2\pi\sigma_s\sigma_b}\left[\frac{-y}{2\sigma_s^2}\right]\exp -\left(\frac{x^2}{2\sigma_b^2}+\frac{y^2}{2\sigma_s^2}\right)\end{aligned}$$

where $\sigma_b > \sigma_s$. Smoothing results in much smaller noise responses from the deriva-

tive estimates, and more smoothing yields less noisy, but more blurry, gradients (Figure 5.6 and 5.7).

Remember this: *Smoothing with a Gaussian will denoise images. An alternative that works better for some kinds of noise is a median filter, which is not linear and is not a convolution. Smoothing with a Gaussian then differentiating will obtain an image gradient estimate that is less sensitive to noise. This process is better represented as convolution with the derivative of Gaussian.*

5.3 SAMPLING, INTERPOLATION AND CONVOLUTION

5.3.1 Modelling a Sampled Function

Passing from a continuous function—like the irradiance at the back of a camera system—to a collection of values on a discrete grid —like the pixel values reported by a camera—is referred to as *sampling*. Sampling must lose information about the original function (for example, see Figure 15.10). Accounting for what is lost requires building a model of the sampling process quite carefully.

Start with a function of one dimension. Write

$$\texttt{sample}_{1D}(f)$$

for an operation that takes a continuous function and returns a sampled version. The sampled version should represent the values of f at all integer points (you can get any other uniform grid with a scale). Another important property is that

$$\int g(u)\texttt{sample}_{1D}(f)du \approx \int g(u)f(u)du$$

to the extent possible for any $g(u)$. This constraint is quite informative. Choose

$$g(u) = \begin{cases} 1/R & \text{for } 0 < u < R \\ 0 & \text{otherwise} \end{cases}$$

and notice that for this choice

$$\int g(u)f(u)du$$

is an average of $f(u)$ over the range $0 < u < R$. To preserve this property, define a function $\delta(u)$ – a *delta function* or δ *function* – by two properties

$$\delta(u) = 0 \qquad \text{for } u \neq 0$$
$$\int \delta(u)du = 1$$

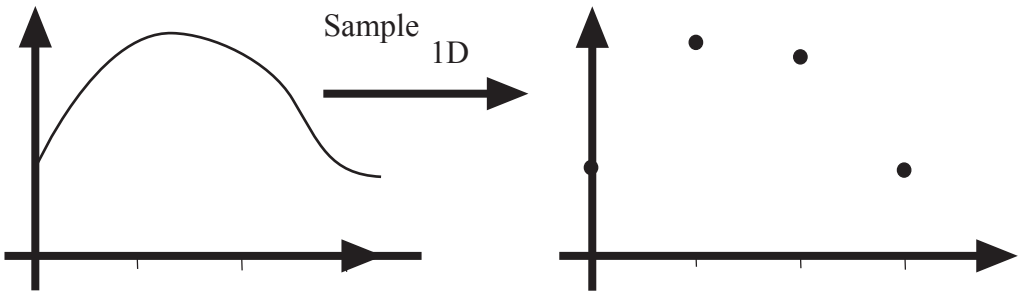

FIGURE 5.8: *Sampling in 1D takes a function and returns a vector whose elements are values of that function at all integer points. The vector is infinite to avoid having to write indices, etc.*

so that

$$\int f(u)\delta(u-x)du = f(x).$$

Now if

$$\texttt{sample}_{1D}(f) = \sum_i f(i)\delta(u-i)$$

then

$$\int g(u)\texttt{sample}_{1D}(f)du = \frac{1}{R}\sum_{0<i<R} f(i)$$

which is an approximation of the average.

Sampling in 2D is very like sampling in 1D. Although sampling can occur on nonregular grids (the best example being the human retina), the most important case has samples on a uniform grid of integer coordinates. The δ function now has the properties

$$\delta(u,v) = 0 \qquad \text{for } u \neq 0 \text{ or } v \neq 0$$
$$\int \delta(u,v)dudv = 1$$

and analogy yields

$$\texttt{sample}_{2D}(F) = \sum_{ij} F(i,j)\delta(x-i, y-j)$$

The grid is infinite in each dimension to avoid having to write ranges, etc. (Figure 5.9).

The δ function is a conceptual device to make the mathematical plumbing work properly. There is no need to place one at each sample function in an array inside your programs. Some practical systems have samples that are not evenly spaced. Older television sets had an aspect ratio of 4:3 (width:height), though 16:9

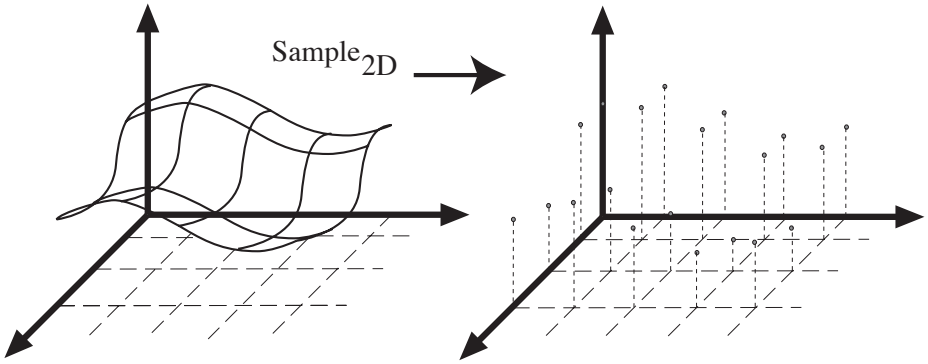

FIGURE 5.9: *Sampling in 2D takes a function and returns an array; again, we allow the array to be infinite dimensional and to have negative as well as positive indices.*

is commone for more recent sets. Cameras quite often accommodated this effect by spacing sample points slightly farther apart horizontally than vertically (in jargon, they had *non-square pixels*). It is unusual to encounter these effects now.

5.3.2 Sampling and Convolution

The sampling model may look strange to you, but respects convolution and in a useful way. Choose some continuous convolution kernel $g(x, y)$. If you convolve $\texttt{sample}_{2D}(\mathcal{I})$ with $g(x, y)$, then sample the result, you get what you would have gotten if you convolve $\mathcal{I}$ with $\texttt{sample}_{2D}(g)$. Check

$$\texttt{sample}_{2d}(\mathcal{I}) * g = \sum_{i,j} \mathcal{I}_{ij} g(x - i, y - j) = \mathcal{I} * \texttt{sample}_{2d}(g)$$

and that

$$\sum_{i,j} \mathcal{I}_{ij} g(x - i, y - j)$$

is the convolution of Chapters 15.10 if you evaluate it at integer points.

5.3.3 Interpolation: Passing from Discrete to Continuous

Recall the interpolate of Section 2.2 had the form

$$\mathcal{I}(x, y) = \sum_{i,j} \mathcal{I}_{ij} b(x - i, y - j).$$

Here b is some function with the properties $b(0, 0) = 1$ and $b(u, v) = 0$ for u and v any other grid point. This is linear and shift invariant (exercises) so it must be a convolution. The way to see the convolution is to use the model of sampling,

above. This exposes the convolution in interpolation. Notice that

$$\begin{aligned}
\texttt{sample}_{2D}(\mathcal{I}) * b &= \int\int \sum_{ij} \mathcal{I}_{ij}\delta(x-u-i, y-v-j)b(u,v)dudv \\
&= \sum_{ij} \mathcal{I}_{ij} \int\int \delta(x-u-i, y-v-j)b(u,v)dudv \\
&= \sum_{i,j} \mathcal{I}_{ij} b(x-i, y-j) \text{ from the property of a } \delta \text{ function}
\end{aligned}$$

which is the form of an interpolate.

Remember this: *The process of sampling a function is modelled using δ functions. These ensure that integrals of the sampled function have sensible values. Interpolation is a process of convolution that takes a sampled function to a continuous function.*

5.4 YOU SHOULD

5.4.1 remember these terms:

5.4.2 remember these facts:

5.4.3 remember these procedures:

5.4.4 be able to:

- Recognize a bank of filters as a way to represent small patterns in images.
- Denoise an image by smoothing with either Gaussian or median filters.
- Form a gradient estimate using derivative of Gaussian filters.
- Use the model of sampled functions in simple calculations.
- Recognize interpolation as a convolution that passes from a sampled function to a continuous function.