CHAPTER 3

Patterns, Smoothing and Filters

TODO: convolution vs filtering **TODO:** associative **TODO:** linear **TODO:** shift invariant - equivariance **TODO:** useful non-linear filters **TODO:** padding **TODO:** stride

Pictures of zebras and of dalmatians have black and white pixels, and in about the same number, too. The difference between the two – stripes vs. spots – are obvious when you look at small groups of pixels. Pixels tend to look like their neighbors, so when a pixel disagrees with its neighbors, it might have been affected by noise. So looking at small groups of pixels is a simple and effective way to suppress noise, too. In this chapter, we introduce methods for obtaining descriptions of the appearance of a small group of pixels. The key process here is forming weighted sums of pixel values using different patterns of weights. to find different image patterns.

3.1 LINEAR FILTERS AND CONVOLUTION

3.1.1 Pattern Detection by Convolution

For the moment, think of an image as a two dimensional array of intensities. Write $\mathcal{I}_{ij}$ for the pixel at position i, j. We will construct a small array (a *mask* or *kernel*) $\mathcal{W}$, and compute a new image $\mathcal{N}$ from the image and the mask, using the rule

$$\mathcal{N}_{ij} = \sum_{uv} \mathcal{I}_{i-u,j-v} \mathcal{W}_{uv}$$

which we will write

$$\mathcal{N} = \mathcal{W} * \mathcal{I}.$$

In some sources, you might see $\mathcal{W} * * \mathcal{I}$ (to emphasize the fact that the image is 2D). We sum over all u and v that apply to $\mathcal{W}$; for the moment, do not worry about what happens when an index goes out of the range of $\mathcal{I}$. This operation is known as *convolution*, and $\mathcal{W}$ is often called the *kernel* of the convolution. You should look closely at the expression; the "direction" of the dummy variable u (resp. v) has been reversed compared with what you might expect (unless you have a signal processing background). What you might expect – sometimes called *correlation* or *filtering* – would compute

$$\mathcal{N}_{ij} = \sum_{uv} \mathcal{I}_{i+u,j+v} \mathcal{W}_{uv}$$

which we will write

$$\mathcal{N} = \texttt{filter}(\mathcal{I}, \mathcal{W}).$$

This difference isn't particularly significant, but if you forget that it is there, you compute the wrong answer.

We carefully avoid inserting the range of the sum; in effect, we assume that the sum is over a large enough range of u and v that all nonzero values are taken

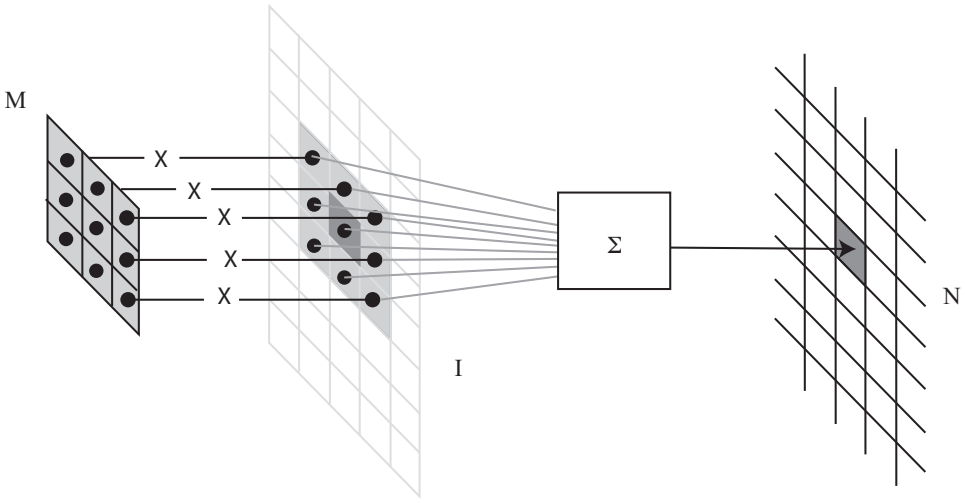

FIGURE 3.1: *To compute the value of $\mathcal{N}$ at some location, you shift a copy of $\mathcal{M}$ (the flipped version of $\mathcal{W}$) to lie over that location in $\mathcal{I}$; you multiply together the non-zero elements of $\mathcal{M}$ and $\mathcal{I}$ that lie on top of one another; and you sum the results.*

into account. Furthermore, we assume that any values that haven't been specified are zero; this means that we can model the kernel as a small block of nonzero values in a sea of zeros. We use this common convention regularly in what follows.

This operation is linear. You should check that:

- if $\mathcal{I}$ is zero, then $\mathcal{W} * \mathcal{I}$ is zero;
- $(k\mathcal{W}) * \mathcal{I}) = k(\mathcal{W} * \mathcal{I}) = \mathcal{W} * (k\mathcal{I})$;
- $(\mathcal{W}) * \mathcal{I}) = k(\mathcal{W} * \mathcal{I}) = \mathcal{W} * (k\mathcal{I})$;
- $(\mathcal{W} + \mathcal{V}) * \mathcal{I}) = \mathcal{W} * \mathcal{I} + \mathcal{V} * \mathcal{I}$;
- $\mathcal{W} * (\mathcal{I} + \mathcal{J}) = \mathcal{W} * \mathcal{I} + \mathcal{W} * \mathcal{J}$;
- $\mathcal{W} * (\mathcal{V} * \mathcal{I}) = (\mathcal{W} * \mathcal{V}) * \mathcal{I}$ (so convolution is associative).

3.1.2 Convolution as Pattern Detection

You should think of the value of $\mathcal{N}_{ij}$ as a dot-product. To see this, flip $\mathcal{W}$ in both directions to form $\mathcal{M}$. Then

$$\begin{aligned}\mathcal{N}_{ij} &= \sum_{uv} \mathcal{I}_{i-u,j-v}\mathcal{W}_{uv} \\ &= \sum_{uv} \mathcal{I}_{i+u,j+v}\mathcal{M}_{uv}\end{aligned}$$

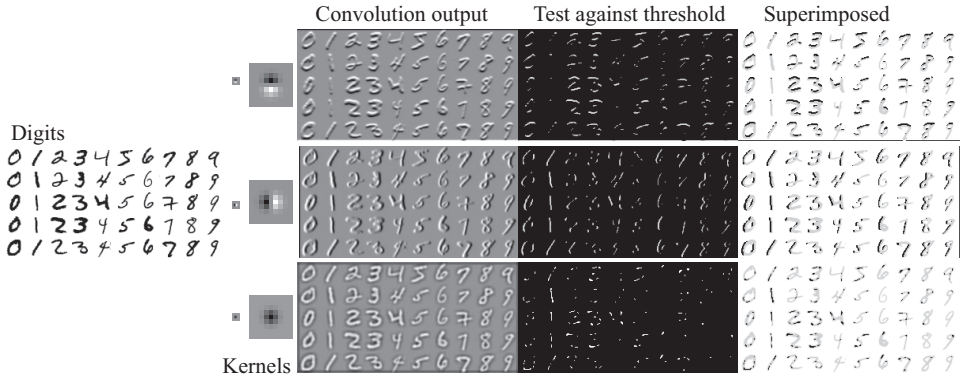

FIGURE 3.2: *On the* **far left**, *some images from the MNIST dataset. Three kernels appear on the* **center left**; *the small blocks show the kernels scaled to the size of the image, so you can see the size of the piece of image the kernel is applied to. The larger blocks show the kernels (mid-grey is zero; light is positive; dark is negative). The kernel in the top row responds most strongly to a dark bar above a light bar; that in the middle row responds most strongly to a dark bar to the left of a light bar; and the bottom kernel responds most strongly to a spot.* **Center** *shows the results of applying these kernels to the images. You will need to look closely to see the difference between a medium response and a strong response.* **Center right** *shows pixels where the response exceeds a threshold. You should notice that this gives (from top to bottom): a horizontal bar detector; a vertical bar detector; and a line ending detector. These detectors are moderately effective, but not perfect.* **Far right** *shows detector responses (in black) superimposed on the original image (grey) so you can see the alignment between detections and the image.*

equivalently

$$\begin{aligned}\mathcal{N} &= \mathcal{I} * \mathcal{W} \\ &= \texttt{filter}(\mathcal{I}, \mathcal{M})\end{aligned}$$

This means that you can think about convolution like this. To compute the value of $\mathcal{N}$ at some location, you place $\mathcal{M}$ (the flipped version of $\mathcal{W}$) at some location in the image; you multiply together the elements of $\mathcal{I}$ and $\mathcal{M}$ that lie on top of one another, ignoring everything in $\mathcal{I}$ outside $\mathcal{M}$; then you sum the results (Figure 3.1).

In turn, you can think about convolution as forming a dot product between $\mathcal{M}$ and the piece of image that lies under $\mathcal{M}$ (reindex the two windows to be vectors). This view explains why a convolution is interesting: it is a very simple pattern detector. Assume that $\mathbf{u}$ and $\mathbf{v}$ are unit vectors. Then $\mathbf{u} \cdot \mathbf{v}$ is largest when $\mathbf{u} = \mathbf{v}$, and smallest when $\mathbf{u} = -\mathbf{v}$. Using the dot-product analogy, for $\mathcal{N}_{ij}$ to have a large and positive value, the piece of image that lies under $\mathcal{M}$ must "look like" $\mathcal{M}$. Figure 3.2 give some examples.

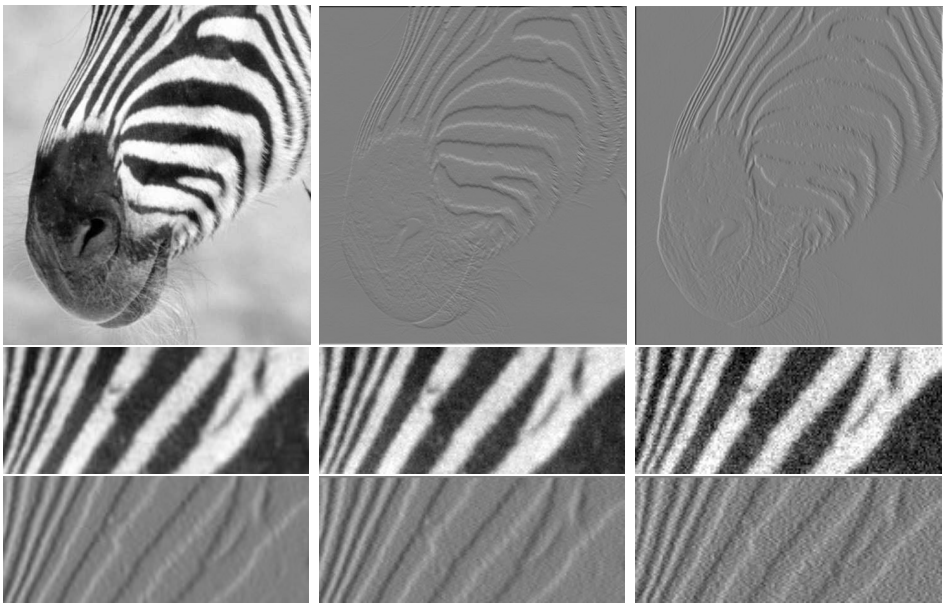

FIGURE 3.3: *The* **top row** *shows estimates of derivatives obtained by finite differences. The image at the* **left** *shows a detail from a picture of a zebra. The* **center** *image shows the partial derivative in the y-direction—which responds strongly to horizontal stripes and weakly to vertical stripes—and the* **right** *image shows the partial derivative in the x-direction—which responds strongly to vertical stripes and weakly to horizontal stripes. However, finite differences respond strongly to noise. The image at* **center left** *shows a detail from a picture of a zebra; the next image in the row is obtained by adding a random number with zero mean and normal distribution ($\sigma = 0.03$; the darkest value in the image is 0, and the lightest 1) to each pixel; and the third image is obtained by adding a random number with zero mean and normal distribution ($\sigma = 0.09$) to each pixel. The* **bottom row** *shows the partial derivative in the x-direction of the image at the head of the row. Notice how strongly the differentiation process emphasizes image noise; the derivative figures look increasingly grainy. In the derivative figures, a mid-gray level is a zero value, a dark gray level is a negative value, and a light gray level is a positive value.*

3.1.3 Convolution to Estimate Derivatives

Image derivatives can be approximated using another example of a convolution process. Because

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

we might estimate a partial derivative as a symmetric *finite difference*:

$$\frac{\partial h}{\partial x} \approx h_{i+1,j} - h_{i-1,j}.$$

This is the same as a convolution, where the convolution kernel is

$$\mathcal{H} = \begin{Bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{Bmatrix}.$$

Notice that this kernel could be interpreted as a template: it gives a large positive response to an image configuration that is positive on one side and negative on the other, and a large negative response to the mirror image.

As Figure 3.3 illustrates, finite differences give a 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 used a 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. We need some method to control image noise.

3.1.4 Smoothing with Convolution

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 Gaussian (normal) probability distribution. This distribution almost always has zero mean. The standard deviation is a parameter of the model. The model is intended to describe thermal noise in cameras and is illustrated in Figure 3.5.

Gaussian noise tends to result in pixels not looking like their neighbors. Other kinds of noise have this property too. Examples include: occasional pixels stuck at full dark or full bright; small random numbers with zero mean added to some pixel values; or some pixels multiplied by random numbers close to one. It is natural to attempt to reduce the effects of noise by replacing each pixel with a weighted average of its neighbors, a process often referred to as *smoothing*. When you smooth an image, it often looks as though it was taken by a defocussed camera, so smoothing is sometimes called *blurring*.

Replacing each pixel with an unweighted average computed over some fixed region centered at the pixel is the same as convolution with a kernel that is a block of ones multiplied by a constant (you should check this). This is a poor model of blurring; its output does not look like that of a defocused camera (Figure 3.6). The reason is clear. Assume that we have an image in which every point but the center point is zero, and the center point is one. If we blur this image by forming an unweighted average at each point, the result looks like a small, bright box, but this is not what defocused cameras do. We want a blurring process that takes a small bright dot to a circularly symmetric region of blur, brighter at the center than at the edges and fading slowly to darkness. As Figure 3.6 suggests, a set of weights of this form produces a much more convincing defocus model.

A good formal model for a fuzzy blob is the *symmetric Gaussian kernel*

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2 + y^2)}{2\sigma^2}\right)$$

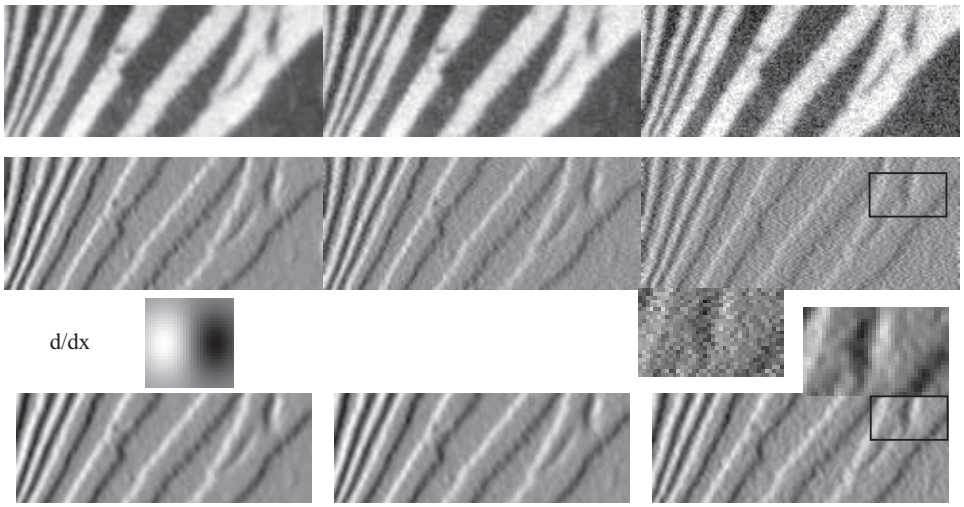

FIGURE 3.4: *Derivative of Gaussian filters are less extroverted in their response to noise than finite difference filters. The image at* **top left** *shows a detail from a picture of a zebra;* **top center** *shows the same image corrupted by zero mean stationary additive Gaussian noise, with* $\sigma = 0.03$ *(pixel values range from 0 to 1).* **Top right** *shows the same image corrupted by zero mean stationary additive Gaussian noise, with* $\sigma = 0.09$*. The second row shows the finite difference in the* x*-direction of each image. These images are scaled so that zero is mid-gray, the most negative pixel is dark, and the most positive pixel is light; we used a different scaling for each image. Notice how the noise results in occasional strong derivatives, shown by a graininess in the derivative maps for the noisy images. The final row shows the partial derivative in the* x*-direction of each image, in each case estimated by a derivative of Gaussian filter with* σ *one pixel. Again, these images are scaled so that zero is mid-gray, the most negative pixel is dark, and the most positive pixel is light; we used a different scaling for each image. The images are smaller than the input image, because we used a* 13×13 *pixel discrete kernel. This means that the six rows (resp. columns) on the top and bottom of the image (resp. left and right) cannot be evaluated exactly, because for these rows the kernel covers some points outside the image; we have omitted these values. Notice how the smoothing helps reduce the impact of the noise; this is emphasized by the detail images (between the second and final row), which are doubled in size. The details show patches that correspond from the finite difference image and the smoothed derivative estimate. We show a derivative of Gaussian filter kernel, which (as we expect) looks like the structure it is supposed to find. This is not to scale (it'd be extremely small if it were).*

illustrated in Figure 3.7. σ is referred to as the *standard deviation* of the Gaussian (or its "sigma"!); the units are interpixel spaces, usually referred to as *pixels*. This is a continuous fuzzy blob. You can get a discrete fuzzy blob by sampling it. The constant term makes the integral over the whole plane equal to one and is often

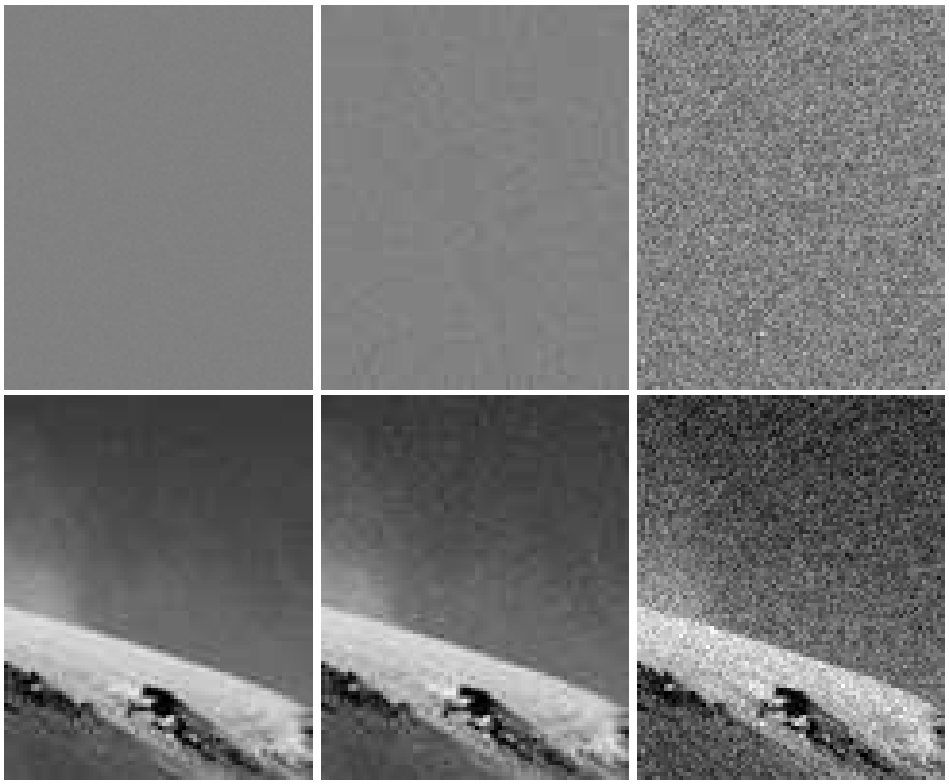

FIGURE 3.5: *The* **top row** *shows three realizations of a stationary additive Gaussian noise process. We have added half the range of brightnesses to these images to show both negative and positive values of noise. From left to right, the noise has standard deviation* $1/256$*,* $4/256$*, and* $16/256$ *of the full range of brightness, respectively. This corresponds roughly to bits zero, two, and five of a camera that has an output range of eight bits per pixel. The* **lower row** *shows this noise added to an image. In each case, values below zero or above the full range have been adjusted to zero or the maximum value accordingly.*

ignored in smoothing applications. The name comes from the fact that this kernel has the form of the probability density for a 2D normal (or Gaussian) random variable with a particular covariance.

This smoothing kernel forms a weighted average that weights pixels at its center much more strongly than at its boundaries. One can justify this approach qualitatively: Smoothing suppresses noise by enforcing the requirement that pixels should look like their neighbors. By downweighting distant neighbors in the average, we can ensure that the requirement that a pixel looks like its neighbors is less strongly imposed for distant neighbors. Choice of scale follows from the following considerations:

- If the standard deviation of the Gaussian is very small—say, smaller than one

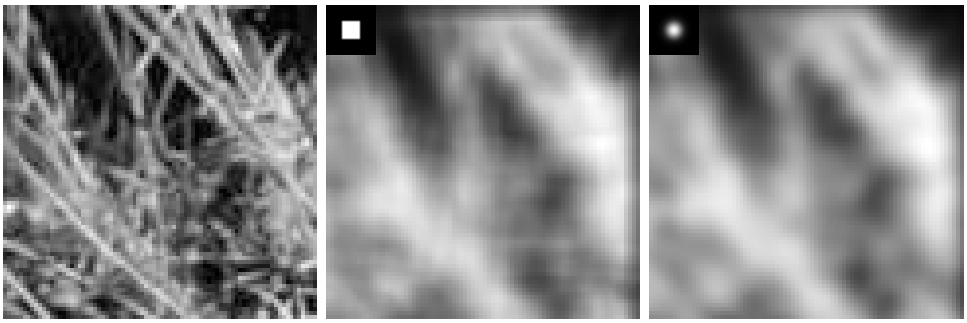

FIGURE 3.6: *Although a uniform local average may seem to give a good blurring model, it generates effects not usually seen in defocusing a lens. The images above compare the effects of a uniform local average with weighted average. The image on the* **left** *shows a view of grass; in the* **center**, *the result of blurring this image using a uniform local model; and on the* **right**, *the result of blurring this image using a set of Gaussian weights. The degree of blurring in each case is about the same, but the uniform average produces a set of narrow vertical and horizontal bars—an effect often known as* ringing. *The small insets show the weights used to blur the image, themselves rendered as an image; bright points represent large values and dark points represent small values (in this example, the smallest values are zero).*

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.

Figure 3.8 illustrates these phenomena. You should notice that Gaussian smoothing can be effective at suppressing noise.

In applications, a discrete smoothing kernel is obtained by constructing a $2k + 1 \times 2k + 1$ array whose i, jth value is

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{((i-k-1)^2 + (j-k-1)^2)}{2\sigma^2}\right).$$

Notice that some care must be exercised with σ. If σ is too small, then only one element of the array will have a nonzero value. If σ is large, then k must be large, too; otherwise, we are ignoring contributions from pixels that should contribute with substantial weight.

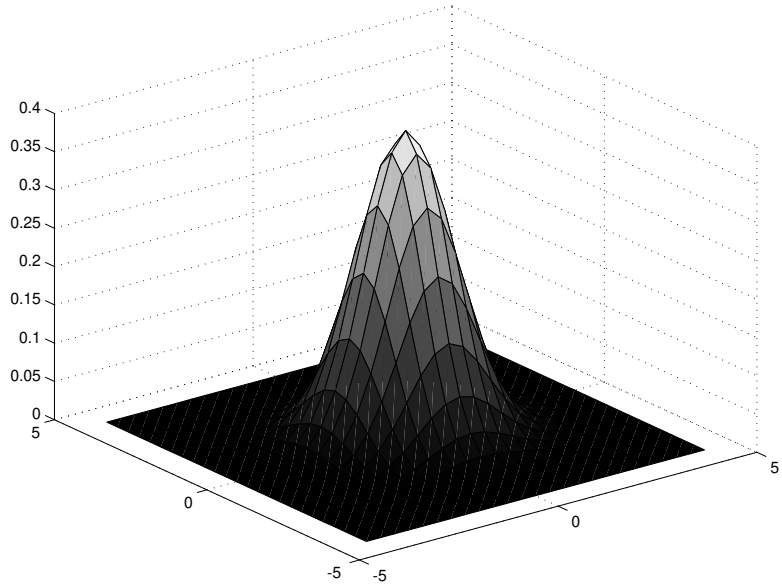

FIGURE 3.7: *The symmetric Gaussian kernel in 2D. This view shows a kernel scaled so that its sum is equal to one; this scaling is quite often omitted. The kernel shown has $\sigma = 1$. Convolution with this kernel forms a weighted average that stresses the point at the center of the convolution window and incorporates little contribution from those at the boundary. Notice how the Gaussian is qualitatively similar to our description of the point spread function of image blur: it is circularly symmetric, has strongest response in the center, and dies away near the boundaries.*

3.2 ESTIMATING GRADIENTS AND ORIENTATIONS

We have seen that finite differences give poor estimates of image gradients for noisy images. Any image gradient of significance to us has effects over a pool of pixels. For example, the contour of an object can result in a long chain of points where the image derivative is large. As another example, a corner typically involves many tens of pixels. If the noise at each pixel is independent and additive, then large image derivatives caused by noise are a local event. Smoothing the image before we differentiate will tend to suppress noise at the scale of individual pixels, because it will tend to make pixels look like their neighbors. However, gradients that are supported by evidence over multiple pixels will tend not to be smoothed out. This suggests differentiating a smoothed image (Figure 3.4).

3.2.1 Derivative of Gaussian Filters

The convolution I have described is a sampled analogue of a continuous operation. If $I(x, y)$ is a continuous image, and $W(x, y)$ is some continuous function, we have $(W * I)(x, y) = \int_{u,v} I(x-u, y-v)W(u, v)dudv$. Notice that, to make this definition meaningful, we need to make assumptions about the range of the integral and the

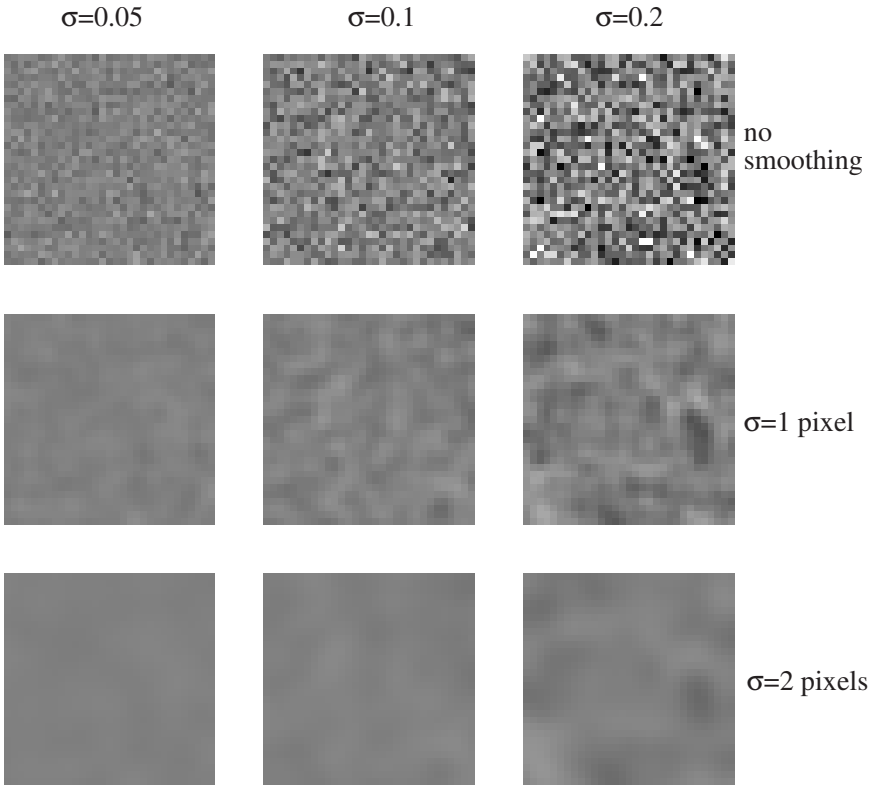

FIGURE 3.8: *The* **top row** *shows images of a constant mid-gray level corrupted by additive Gaussian noise. In this noise model, each pixel has a zero-mean normal random variable added to it. The range of pixel values is from zero to one, so that the standard deviation of the noise in the first column is about 1/20 of full range. The* **center row** *shows the effect of smoothing the corresponding image in the top row with a Gaussian filter of σ one pixel. Notice the annoying overloading of notation here; there is Gaussian noise and Gaussian filters, and both have σ's. One uses context to keep these two straight, although this is not always as helpful as it could be, because Gaussian filters are particularly good at suppressing Gaussian noise. This is because the noise values at each pixel are independent, meaning that the expected value of their average is going to be the noise mean. The* **bottom row** *shows the effect of smoothing the corresponding image in the top row with a Gaussian filter of σ two pixels.*

values of the image and kernel that are analogous to those we made for the sampled case. Context will tell whether I am referring to the sampled operation (commonly) or the continuous operation (seldom). Convolution has an important property. Write $\texttt{shift}(\mathcal{I})$ for an operation that shifts an image (so $\texttt{shift}(\mathcal{I})_{i+tx,j+t_y} = I_{ij}$, or in the continuous case, $\texttt{shift}(\mathcal{I})(\S + \sqcup_\S, \dagger + \sqcup_\dagger) = \mathcal{I}(\S, \dagger)$). Convolution is *shift invariant*, meaning that $\mathcal{W} * \texttt{shift}(\mathcal{I}) = \texttt{shift}(\mathcal{W} * \mathcal{I})$. This is true for both sampled and continuous operations. It can be proven that if an operation is (a)

linear and (b) shift invariant, then there is some convolution kernel that implements it. You should remember this result.

Smoothing an image and then differentiating it is the same as convolving it with the derivative of a smoothing kernel. This fact is most easily seen by thinking about continuous convolution. First, differentiation is linear and shift invariant. This means that there is some kernel—we dodge the question of what it looks like—that differentiates. That is, given a function $I(x, y)$,

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

Now we want the derivative of a smoothed function. We write the convolution kernel for the smoothing as S. Recalling that convolution is associative, we have

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

This fact appears in its most commonly used form when the smoothing function is a Gaussian. We can then write

$$\frac{\partial (G_\sigma * I)}{\partial x} = (\frac{\partial G_\sigma}{\partial x}) * I,$$

that is, we need only convolve with the derivative of the Gaussian, rather than convolve and then differentiate. As discussed in Section 7.3, smoothed derivative 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 (Figure 7.11). Smoothing results in much smaller noise responses from the derivative estimates (Figure 3.4).

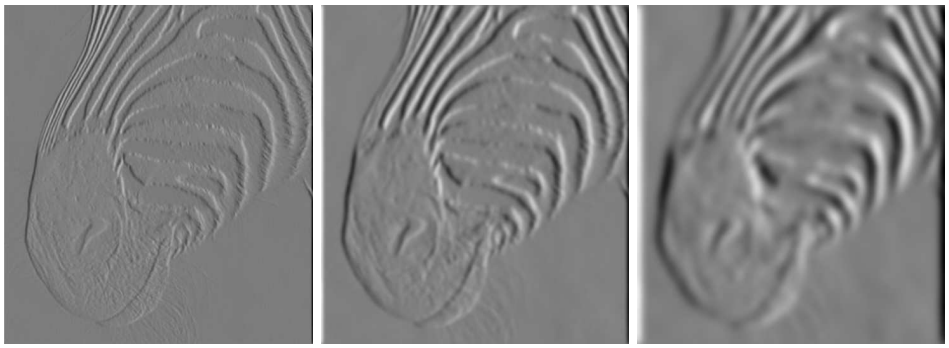

FIGURE 3.9: *The scale (i.e., σ) of the Gaussian used in a derivative of Gaussian filter has significant effects on the results. The three images show estimates of the derivative in the x direction of an image of the head of a zebra obtained using a derivative of Gaussian filter with σ one pixel, three pixels, and seven pixels (***left** *to* **right***). Note how images at a finer scale show some hair, the animal's whiskers disappear at a medium scale, and the fine stripes at the top of the muzzle disappear at the coarser scale.*

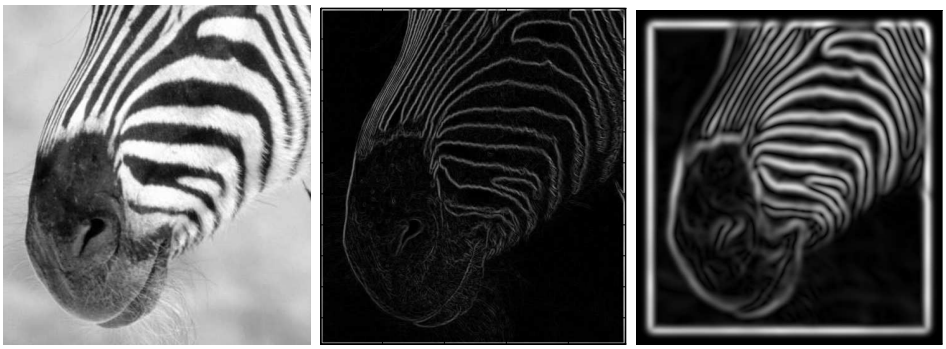

FIGURE 3.10: *The gradient magnitude can be estimated by smoothing an image and then differentiating it. This is equivalent to convolving with the derivative of a smoothing kernel. The extent of the smoothing affects the gradient magnitude; in this figure, we show the gradient magnitude for the figure of a zebra at different scales. At the* **center***, gradient magnitude estimated using the derivatives of a Gaussian with* $\sigma = 1$ *pixel; and on the* **right***, gradient magnitude estimated using the derivatives of a Gaussian with* $\sigma = 2$ *pixel. Notice that large values of the gradient magnitude form thick trails.*

The choice of σ used in estimating the derivative is often called the *scale* of the smoothing. Scale has a substantial effect on the response of a derivative filter. Assume we have a narrow bar on a constant background, rather like the zebra's whisker. Smoothing on a scale smaller than the width of the bar means that the filter responds on each side of the bar, and we are able to resolve the rising and falling edges of the bar. If the filter width is much greater, the bar is smoothed into the background and the bar generates little or no response (Figure 3.9).

3.2.2 Orientations

As the light gets brighter or darker (or as the camera aperture opens or closes), the image will get brighter or darker, which we can represent as a scaling of the image value. The image $\mathcal{I}$ will be replaced with $s\mathcal{I}$ for some value s. The magnitude of the gradient scales with the image, i.e., $\|\nabla\mathcal{I}\|$ will be replaced with $s|\nabla\mathcal{I}\|$. This creates problems for edge detectors, because edge points may appear and disappear as the image gradient values go above and below thresholds with the scaling. One solution is to represent the *orientation* of image gradient, which is unaffected by scaling (Figure 3.11). The gradient orientation field depends on the smoothing scale at which the gradient was computed. Orientation fields can be quite characteristic of particular textures (Figure **??**), and we will use this important property to come up with more complex features below.

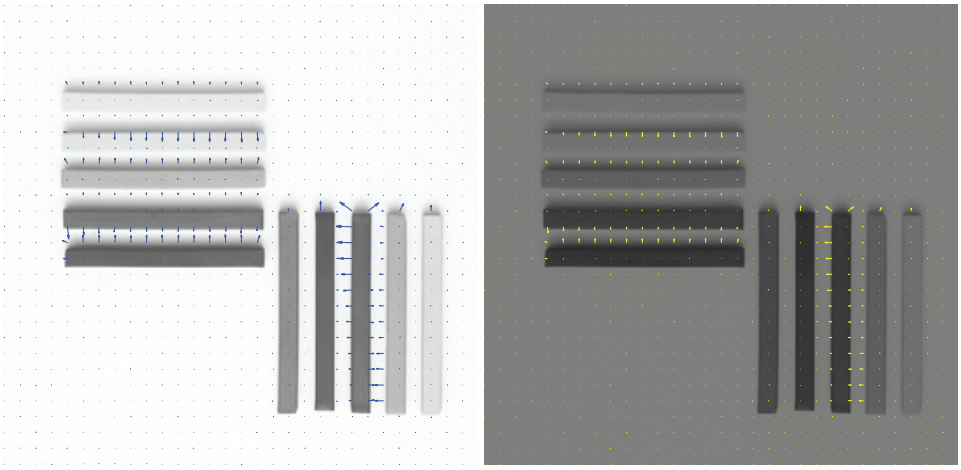

FIGURE 3.11: *The magnitude of the image gradient changes when one increases or decreases the intensity. The orientation of the image gradient does not change; we have plotted every 10th orientation arrow, to make the figure easier to read. Note how the directions of the gradient arrows are fixed, whereas the size changes.* Philip Gatward © Dorling Kindersley, used with permission.

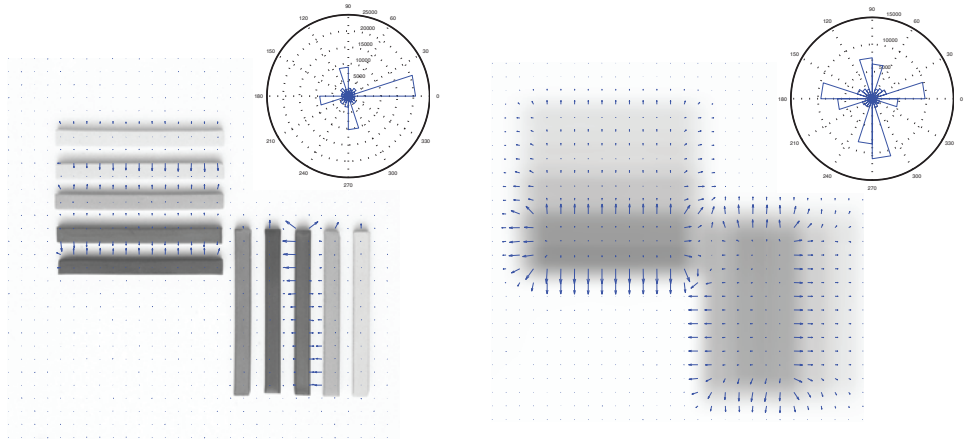

FIGURE 3.12: *The scale at which one takes the gradient affects the orientation field. We show the overall trend of the orientation field by plotting a rose plot, where the size of a wedge represents the relative frequency of that range of orientations.* **Left** *shows an image of artists pastels at a fairly fine scale; here the edges are sharp, and so only a small set of orientations occurs. In the heavily smoothed version on the* **right***, all edges are blurred and corners become smooth and blobby; as a result, more orientations appear in the rose plot.* Philip Gatward © Dorling Kindersley, used with permission.