Chapter 10

NON-LINEAR FILTERS

10.1 Filters as Templates

It turns out that filters offer a natural mechanism for finding simple patterns, because filters respond most strongly to pattern elements that look like the filter. For example, smoothed derivative filters are intended to give a strong response at a point where the derivative is large; at these points, the kernel of the filter "looks like" the effect it is 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.

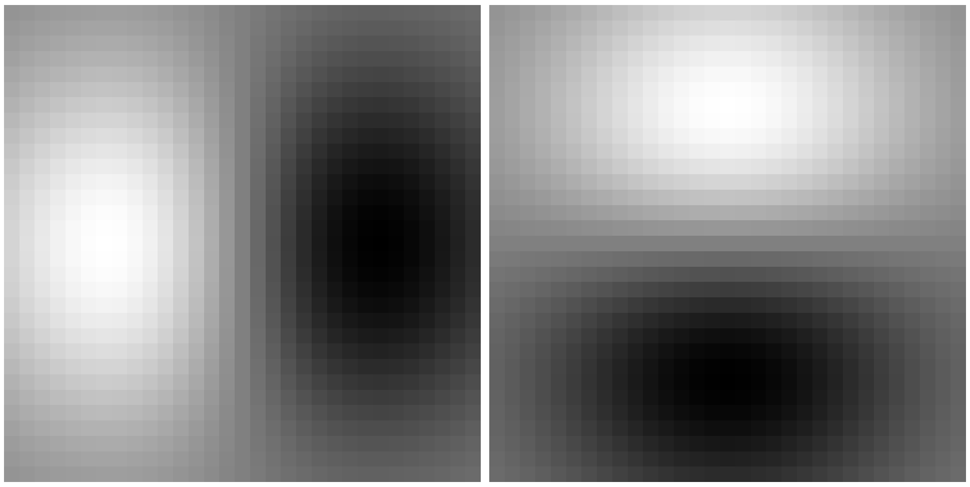

Figure 10.1. Filter kernels "look like" the effects they are intended to detect. On the left, an smoothed derivative of Gaussian filter that looks for large changes in the x-direction (such as a dark blob next to a light blob); on the right, a smoothed derivative of Gaussian filter that looks for large changes in the y-direction.

It is generally the case that filters that are intended to give a strong response to a pattern look like that pattern. This is a simple geometric result.

10.1.1 Convolution as a Dot Product

Recall from section 8.1.1 that, for $\mathcal{G}$ the kernel of some linear filter, the response of this filter to an image $\mathcal{H}$ is given by:

$$R_{ij} = \sum_{u,v} G_{i-u,j-v} H_{uv}$$

Now consider the response of a filter at the point where i and j are zero. This will be

$$R = \sum_{u,v} G_{-u,-v} H_{u,v}$$

This response is obtained by associating image elements with filter kernel elements, multiplying the associated elements, and summing. We could scan the image into a vector, and the filter kernel into another vector, in such a way that associated elements are in the same component. By inserting zeros as needed, we can ensure that these two vectors have the same dimension. Once this is done, the process of multiplying associated elements and summing is precisely the same as taking a dot-product.

This is a powerful analogy, because this dot-product, like any other, will achieve its largest value when the vector representing the image is parallel to the vector representing the filter kernel. This means that a filter responds most strongly when it encounters an image pattern that looks like the filter. The response of a filter will get stronger as a region gets brighter, too.

Now consider the response of the image to a filter at some other point. Nothing significant about our model has changed; again, we can scan the image into one vector and the filter kernel into another vector, such that associated elements lie in the same components. Again, the result of applying this filter is a dot-product. There are two useful ways to think about this dot-product.

10.1.2 Changing Basis

We can think of convolution as a dot-product between the image and *a different vector* (because we have moved the filter kernel to lie over some other point in the image). The new vector is obtained by rearranging the old one, so that the elements lie in the right components to make the sum work out (exercise **??**). This means that, by convolving an image with a filter, we are representating the image on a new *basis* of the vector space of images — the basis given by the different shifted versions of the filter. The original basis elements were vectors with a zero in all slots except one. The new basis elements are shifted versions of a single pattern. For many of the kernels we have discussed, we expect that this process will *lose* information — for the same reason that smoothing suppresses noise — so that the coefficients on this basis are redundant. This basis transformation is valuable in texture analysis (section **??**).

10.1.3 Normalised Correlation

We can think of convolution as comparing a filter with a patch of image centered at the point whose response we are looking at. In this view, the image neighbourhood corresponding to the filter kernel is scanned into a vector which is compared with the filter kernel. By itself, this dot-product is a poor way to find features, because the value may be large simply because the image region is bright. By analogy with vectors, we are interested in the cosine of the angle between the filter vector and the image neighbourhood vector; this suggests computing the root sum of squares of the relevant image region (the image elements that would lie under the filter kernel) and dividing the response by that value.

This yields a value that is large and positive when the image region looks like the filter kernel, and small and negative when the image region looks like a contrast-reversed version of the filter kernel. This value could be squared if contrast reversal doesn't matter. This is a cheap and effective method for finding patterns, often called **normalised correlation**.

10.1.4 Controlling the Television by Finding Hands by Normalised Correlation

It would be nice to have systems that could respond to human gestures. You might, for example, wave at the light to get the room illuminated, or point at the airconditioning to get the temperature changed. In typical consumer applications, there are quite strict limits to the amount of computation available, meaning that it is essential that the gesture recognition system be simple. However, such systems are usually quite limited in what they need to do, too.

Typically, a user interface is in some state — perhaps a menu is displayed — and an event occurs — perhaps a button is pressed on a remote control. This event causes the interface to change state — a new menu item is highlighted, say — and the whole process continues. In some states, some events cause the system to perform some action — the channel might change. All this means that a state machine is a natural model for a user interface.

One way for vision to fit into this model is to provide events. This is good, because there are generally very few different kinds of event, and we know what kinds of event the system should care about in any particular state. As a result, the vision system needs only to determine whether either nothing or one of a small number of known kinds of event has occurred. It is quite often possible to build systems that meet these constraints.

Controlling the Television

A relatively small set of events is required to simulate a remote control — one needs events that "look like" button presses (for example, to turn the television on or off), and events that "look like" pointer motion (for example, to increase the volume; it is possible to do this with buttons, too). With these events, the television can be

Figure 10.2. Freeman's system controlling his telly

turned on, and an on-screen menu system navigated.

Freeman *et al.* produced an interface where an open hand turns the television on. This can be robust, because all the system needs to do is determine whether there is a hand in view. Furthermore, the user will cooperate by holding their hand up and open. Because the user is expected to be a fairly constant distance from the camera — so the size of the hand is roughly known, and there is no need to search over scales — and in front of the television, the image region that needs to be searched to determine if there is a hand is quite small.

The hand is held up in a fairly standard configuration and orientation to turn the television set on (so we know what it will look like). This means that Freeman can get away with using a normalised correlation score to find the hand. Any points in the correlation image where the score is high enough correspond to hands.

This approach can be used to control volume, etc. as well as turn the television on and off. To do so, we need some notion of where the hand is going — to one side turns the volume up, to the other turns it down — and this can be obtained by comparing the position in the previous frame with that in the current frame. The system displays an iconic representation of its interpretation of hand position, so the user has some feedback as to what the system is doing (figure 10.2).

10.1.5 More Noise Models

10.1.6 Spatial Noise Models

Salt and pepper noise models cameras with defective sample sites. Pixels are chosen uniformly at random; each of these pixels is set to be either full value or zero value (again, uniformly at random). The result looks as though the image has been sprinkled with salt and pepper, whence the name. There is a basic conceptual difference between stationary additive Gaussian noise and salt and pepper noise; in the first case, we add a random quantity to each pixel, whereas in the second, we

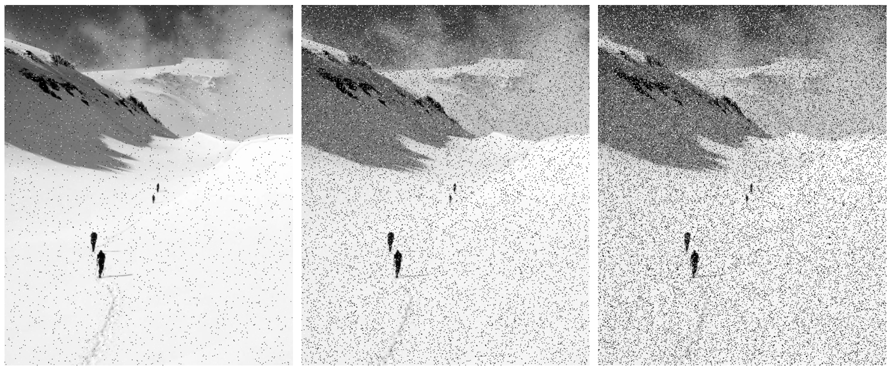

Figure 10.3. Examples of salt-and-pepper noise at different intensities (Poisson process). The snowy background means that, in some areas, the pepper is more visible than the salt.

use a random mechanism to select pixels, and then operate on the selected pixels.

Random mechanisms to select pixels are usually called **point processes**, and form a significant topic of their own. We will describe some important types of point process. In a **Poisson point process**, points on the image plane are chosen randomly so that the expected number of points in any subset is proportional to the area of the subset. The constant of proportionality is known as the intensity of the process. An instance of a Poisson point process can be obtained by sampling the number of affected pixels from a Poisson distribution whose mean is the intensity times the image area, and then drawing the coordinates of these pixels uniformly at random.

Because we need to flip some pixels to white and other to black, we need to use a **marked point process**. In this model, we use a point process to select points, then assign to each point a "mark" (for example, whether it is white or black, or some other such thing) at random using an appropriate distribution. For a camera where a non-responsive pixel is as likely as a saturated pixel, the probability that a point carries a black mark should be the same as the probability that a point carries a white mark. If (for example, because of the manufacturing process) there are fewer non-responsive pixels than responsive pixels, then the distribution on the marks can reflect this as well.

A noise process of this form results in fairly evenly distributed noise. A set of bad pixels that consists of widely separated large, tight clumps is quite unlikely. One model that would achieve this takes points chosen by a Poisson process and then marks a clump of pixels around them. The shape of the clump is chosen randomly — this is another form of mark. Another form of noise that is quite common in videotape systems involves whole scan-lines of an image being turned to noise. The line involved can be chosen with a Poisson point process as well (figure 10.4). A variety of models of this form are available; a brief exposition appears in chapter *** of [**?**]; more detail on point processes appears in [**?**].

The response of a filter to spatial noise models of this form is usually impossible to compute analytically. Instead, we rely on simulation. The basic idea is to set up a probabilistic noise model, draw a large number of samples from that model, and then apply the filter in question to the samples. One computes appropriate statistics from the result — for example, the mean and variance of the noise response. In principle it is possible to choose filters from families of filters in this way, although we are not aware of anyone doing so in the computer vision literature.

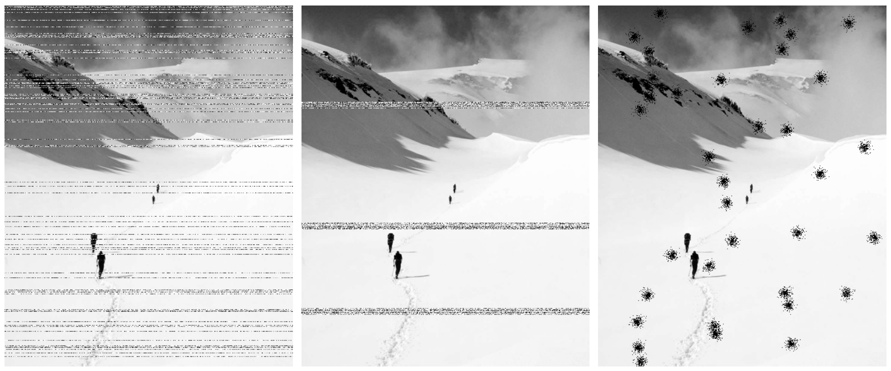

Figure 10.4. A variety of other natural spatial noise models. We show two versions. The first image shows the result of randomly replacing lines with noise, where the probability of replacing the line is uniform; in the second, a block of lines of random length is replaced with noise — this process is a reasonable model of the sort of noise that occurs in fast-forwarding or rewinding VCR's. In the third, a Poisson process chooses "noise points"; in a neighbourhood of each noise point, pixels are randomly marked black with a probability that falls off as the negative exponential of the distance from the noise point. This process simulates damage to the CCD array, or to the lens.

10.2 Non-linear Filters

An alternative approach to preserving edges while smoothing noise is to think of a filter as a statistical estimator. In particular, the goal here is to estimate the actual image value at a pixel, in the presence of noisy measurements. This view leads us to a class of filters that are hard to analyse, but can be extremely useful.

10.2.1 Robust Estimates

Smoothing an image with a symmetric Gaussian kernel replaces a pixel with some weighted average of its neighbours. If an image has been corrupted with stationary additive zero-mean Gaussian noise, then this weighted average gives a reasonable estimate of the original value of the pixel. The expected noise response is zero, and the estimate has better behaviour in terms of spatial frequency than a simple average (as the ringing effects in figure **??** show).

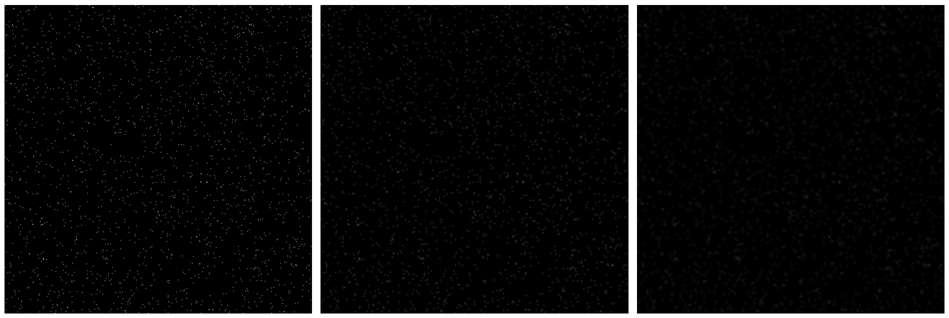

Figure 10.5. On the left, a black background with white noise pixels distributed as a Poisson point process. These pixels are outliers, in the sense that they differ radically from their neighbouring pixels. In the center image, we see the result of estimating pixels as the response of the image to a Gaussian filter with σ one pixel; we are estimating a pixel value as a weighted sum of its neighbours. Because the noise pixels are wildly different from their neighbourhood, they skew this estimate substantially. In the right hand image, we see the result of using a Gaussian filter with σ two pixels; the effect remains, but is smaller, because the effective support of the filter is larger.

However, if the image noise is not stationary additive Gaussian noise, difficulties arise. For example, consider a noise model where image points are set to the brightest or darkest possible value with a Poisson point process (section 10.5). In particular, consider a region of the image which has a constant dark value and there is a single bright pixel due to noise — smoothing with a Gaussian will leave a smooth, Gaussian-like, bright bump centered on this pixel.

The problem here is that a weighted average can be arbitrarily badly affected by very large noise values. Thus, in our example, we can make the bright bump arbitrarily bright by making the bright pixel arbitrarily bright — perhaps as result of, say, a transient error in reading a memory element. Estimators that do not have this most undesirable property are often known as **robust estimates**.

The best known robust estimator involves estimating the mean of a set of values using its **median**. For a set with $2k+1$ elements, the median is the $k+1$'th element of the sorted set of values. For a set with $2k$ elements, the median is the average of the k and the $k+1$'th element of the sorted set. It does not matter whether the set is sorted in increasing or decreasing order (exercises!).

10.2.2 Median Filters

A **median filter** is specified by giving some form of neighbourhood shape (which can significantly affect the behaviour of the filter). This neighbourhood is passed over the image as in convolution, but instead of taking a weighted sum of elements within the neighbourhood, we take the median. If we write the neighbourhood

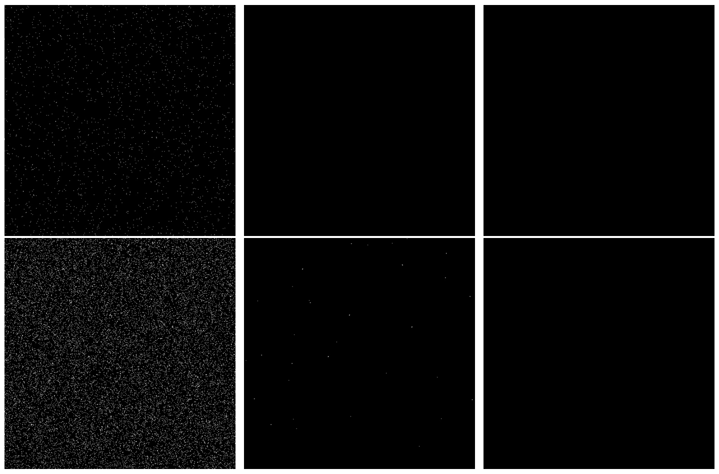

Figure 10.6. The columns on the left show Poisson noise processes of different intensities; on the top row, there are 2000 noise pixels and on the bottom row, 20000. The second column shows the effect of applying a filter that returns the median of a 3x3 neighbourhood to these images, and the third column shows the effect of applying a filter that returns the median of a 7x7 neighbourhood to these images. Notice that, if the noise is intense, then the median filter is unable to suppress it.

centered at i, j as N_{ij}, the filter can be described by:

$$y_{ij} = med(\{x_{uv} | x_{uv} \in N_{ij}\})$$

Applying a median filter to our example of a uniform dark region with a single, arbitrarily bright, pixel will yield a dark region. In this example, up to half of the elements in the neighbourhood could be noise values and the answer would still be correct (exercises!). It is difficult to obtain analytic results about the behaviour of median filters, but a number of general observations apply.

Multi-stage Median Filters

Median filters preserve straight edges, but tend to behave badly at sharp corners (figure 8.1 and exercises). This difficulty is usually dealt with by forming a **multi-stage median filter**; this filter responds with the median of a set of different medians, obtained in different neighbourhoods:

$$\begin{aligned} y_{ij} &= med(z_1, z_2, z_3, z_4) \\ z_1 &= med(\{x_{uv} | x_{uv} \in N^1_{ij}\}) \\ z_2 &= med(\{x_{uv} | x_{uv} \in N^2_{ij}\}) \\ z_3 &= med(\{x_{uv} | x_{uv} \in N^3_{ij}\}) \\ z_4 &= med(\{x_{uv} | x_{uv} \in N^4_{ij}\}) \end{aligned}$$

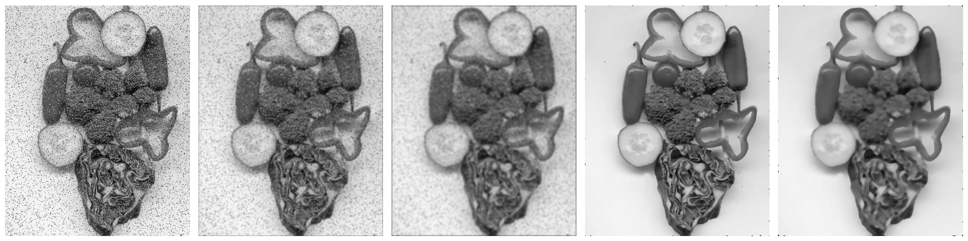

Figure 10.7. On the **left**, an image corrupted with salt-and-pepper noise (points are chosen by a Poisson process, and then with even probability marked either black or white; in this image, about 9% of the pixels are noise pixels). Gaussian smoothing (**center left** shows σ one pixel and **center** shows σ two pixels) works particularly poorly, as the contrast makes the dark regions left behind by averaging in dark pixels very noticeable. A median filter is much more successful (**center right** shows a 3x3 median filter and **right** shows a 7x7 median filter). Notice how the median filter blurs boundaries.

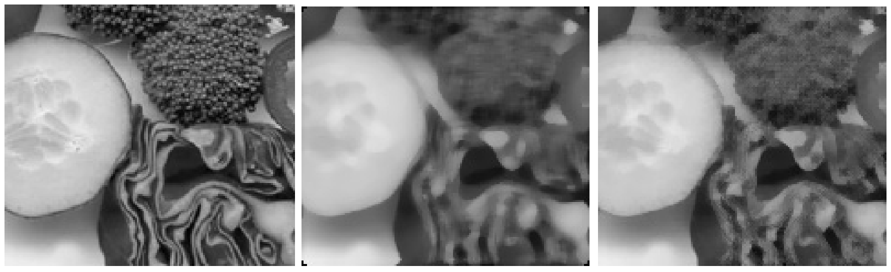

Figure 10.8. On the left, a detail from the figure of vegetables; in the center the result of applying a median filter with a 7x7 neighbourhood of support. Notice that the texture of the broccoli florets is almost completely smoothed away, and that corners around the red cabbage have been obscured. These effects could be useful in some contexts, but reduce the usefulness of the filter in suppressing long-tailed noise because they represent a reduction in image detail, too. On the right, the result of applying a multistage median filter, using 7 pixel domains that are horizontal, vertical, and along the two diagonals. Significantly less detail has been lost.

where N^1 is a vertically extended neighbourhood, N^2 is a horizontally extended neighbourhood, and N^3 and N^4 are diagonal neighbourhoods. Exercise **??** asks for an intuitive argument as to why this filter is inclined to preserve corners; the effect is illustrated in figure 10.8.

Trimmed and Hybrid Median Filters

While median filters tend to be better than linear filters at rejecting very large noise values — so called **outliers** — they tend to be poorer than linear filters at handling noise that does not have outliers. In jargon, noise that can produce occasional large values is often called **long-tailed noise**, because the probability density for the

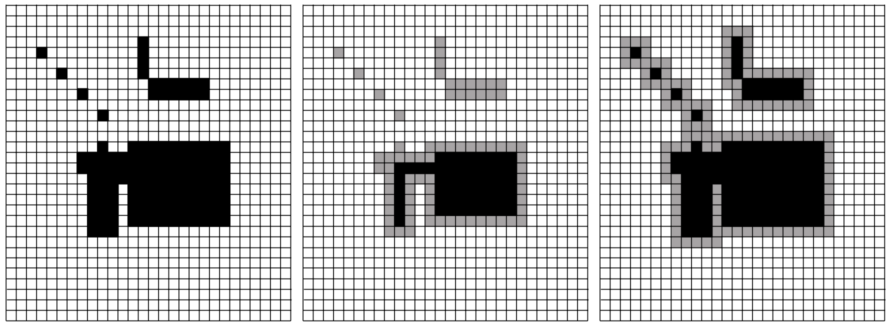

Figure 10.9. On the left, a binary image; a natural strategy for removing small groups of dark pixels is to lighten all pixels that do *not* lie at the center of a 3x3 dark neighbourhood. This process is known as erosion. In the center, the relevant pixels have been greyed. Similarly, we could fill in small gaps by marking all pixels such that a 3x3 neighbourhood around the pixel contacts a dark pixel, a process known as dilation. The relevant pixels have been greyed on the right.

noise values has "long tails" —there is significant weight in the density far from the mean; similarly, noise that does not have this property is often called **short-tailed noise**. In a neighbourhood, long-tailed noise will produce a small number of very large values, which tend not to affect the median much; however, short-tailed noise will produce values that are similar to the original pixel value and which will affect the median more. This difficulty can be handled either by using an **α-trimmed linear filter** — where $\alpha/2$ percent of the largest and smallest values in a neighbourhood are removed from consideration and the rest are subjected to a linear filter — or by using a **hybrid median filter** — where the output is the median of a set of linear filters over a neighbourhood.

Median filters can be extremely slow. One strategy is to pretend that a median filter is separable, and apply separate x and y median filters.

10.2.3 Mathematical morphology: erosion and dilation

A variety of useful operators can be obtained from considering set-theoretic operations on binary images. It often occurs that a binarised images has individual pixels or small groups of pixels that are isolated from the main body of the image. Commonly, one would like to remove very small groups of pixels and join up groups that are close together. Small groups can be removed by noticing that a block of pixels would not fit inside a small group; large groups can be joined up by "thickening" their boundaries. In figure 10.9, we illustrate removing groups of dark pixels by removing pixels that are not at the center of a 3x3 block of dark pixels (i.e. pixels where some neighbour is light). Similarly, gaps can be jumped by attaching a 3x3 neighbourhood to each pixel.

These (quite useful) tricks can be generalised. For example, there is no need to insist on a 3x3 neighbourhood — any pattern will do. The generalisation is most easily phrased in terms of sets. Assume, for the moment, we have two binary images $\mathcal{I}$ and $\mathcal{S}$ (i.e. pixel values in each can be only either 0 or 1). We can regard each image as a representation of a set of elements which belong to a finite grid. Pixels that have the value 1 are elements of the set and pixels that have the value 0 are not. Now write $\mathcal{S}_p$ for the image obtained by shifting the center of $\mathcal{S}$ to the pixel p. We can define a new set

$$\mathcal{I} \oplus \mathcal{S} = \{p : \mathcal{S}_p \cap \mathcal{I} \neq \emptyset\}$$

This is called the **dilation** of the set $\mathcal{I}$ by $\mathcal{S}$. Similarly, we can define

$$\mathcal{I} \ominus \mathcal{S} = \{p : \mathcal{S}_p \subset \mathcal{I}\}$$

which is called the **erosion** of the set $\mathcal{I}$ by $\mathcal{S}$. In these operations, $\mathcal{S}$ is usually called the **structuring element**. The properties of these operators have been widely studied []; some are explored in the exercises. Their main application in practice appears in cleaning up data sets. Typically, a predicate is available that marks "interesting" pixels — which might be skin-coloured, or red, or textured, etc. Usually, small groups of pixels pass this criterion as well as the real regions of interest. A few passes of erosion by a 3x3 neighbourhood, followed by a few passes of dilation by a 3x3 neighbourhood, will remove small groups, fill gaps, and leave an estimate of the real region of interest that is often significantly improved. Occasionally, applications arise where erosion or dilation by structuring elements different from kxk neighbourhoods is appropriate [].

10.3 Corners and orientation representations

Edge detectors notoriously fail at corners, because the assumption that estimates of the partial derivatives in the x and y direction suffice to estimate an oriented gradient becomes unsupportable. At sharp corners or unfortunately oriented corners, these partial derivative estimates will be poor, because their support will cross the corner. There are a variety of specialised corner detectors, which look for image neighbourhoods where the gradient swings sharply. More generally, the statistics of the gradient in an image neighbourhood yields quite a useful description of the neighbourhood. There is a rough taxonomy of four qualitative types of image window:

- *constant windows*, where the grey level is approximately constant;
- *edge windows*, where there is a sharp change in image brightness that runs along a single direction within the window;
- *flow windows*, where there are several fine parallel stripes — say hair or fur — within the window;

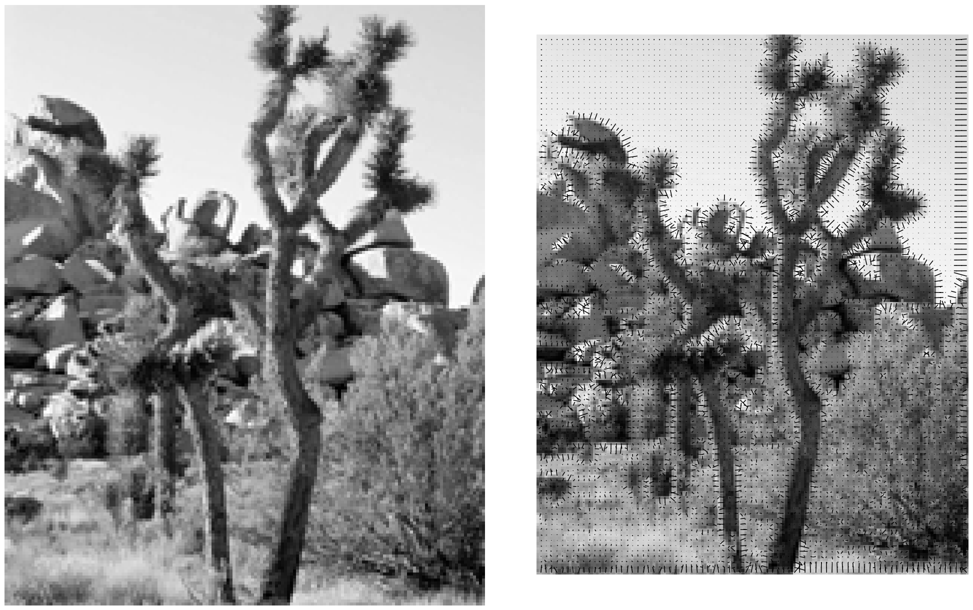

Figure 10.10. An image of a joshua tree, and its orientations shown as vectors superimposed on the image. The center image shows the orientation superimposed on top of the image as small vectors. Notice that around corners and in textured regions, the orientation vector swings sharply.

- and *2D windows*, where there is some form of 2D texture — say spots, or a corner — within the window.

These cases correspond to different kinds of behaviour on the part of the image gradient. In constant windows, the gradient vector is short; in edge windows, there is a small number of long gradient vectors all pointing in a single direction; in flow windows, there are many gradient vectors, pointing in two directions; and in 2D windows, the gradient vector swings.

These distinctions can be quite easily drawn by looking at variations in orientation within a window. In particular, the matrix

$$\mathcal{H} = \sum_{window} \left\{ (\nabla I)(\nabla I)^T \right\} \approx \sum_{window} \left\{ \begin{array}{cc} (\frac{\partial G_\sigma}{\partial x} **\mathcal{I})(\frac{\partial G_\sigma}{\partial x} **\mathcal{I}) & (\frac{\partial G_\sigma}{\partial x} **\mathcal{I})(\frac{\partial G_\sigma}{\partial y} **\mathcal{I}) \\ (\frac{\partial G_\sigma}{\partial x} **\mathcal{I})(\frac{\partial G_\sigma}{\partial y} **\mathcal{I}) & (\frac{\partial G_\sigma}{\partial y} **\mathcal{I})(\frac{\partial G_\sigma}{\partial y} **\mathcal{I}) \end{array} \right\}$$

gives a good idea of the behaviour of the orientation in a window. In a constant window, both eigenvalues of this matrix will be small, because all terms will be small. In an edge window, we expect to see one large eigenvalue associated with gradients at the edge and one small eigenvalue because few gradients will run in other directions. In a flow window, we expect the same properties of the eigenvalues,

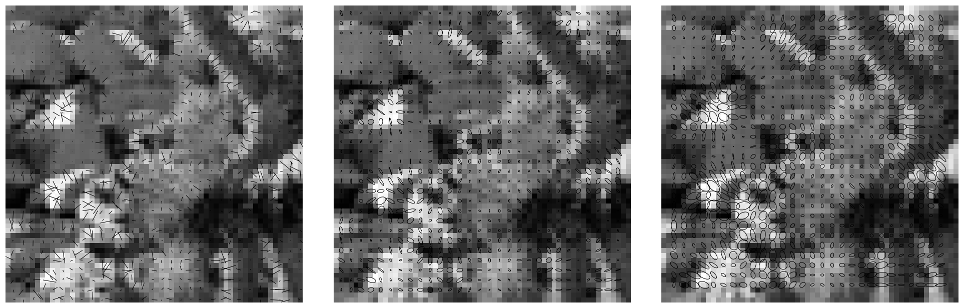

Figure 10.11. The orientation field for a detail of the joshua tree picture. On the left, the orientations shown as vectors and superimposed on the image. Orientations have been censored to remove those where the gradient magnitude is too small. The center shows the ellipses described in the text, for a 3x3 window; left shows the ellipses for a 5x5 window.

except that the large eigenvalue is likely to be larger because many edges contribute. Finally, in a 2D window, both eigenvalues will be large.

The behaviour of this matrix is most easily understood by plotting the ellipses

$$(x, y)^T \mathcal{H}^{-1}(x, y) = \epsilon$$

for some small constant ϵ. These ellipses are superimposed on the image windows. Their major and minor axes will be along the eigenvectors of $\mathcal{H}$, and the extent of the ellipses along their major or minor axes corresponds to the size of the eigenvalues; this means that a large circle will correspond to an edge window and a narrow extended ellipse will indicate an edge window as in figure 10.10. Thus, corners could be marked by marking points where the area of this ellipse is large. The localisation accuracy of this approach is limited by the size of the window and the behaviour of the gradient. More accurate localisation can be obtained, at the price of providing a more detailed model of the corner sought (see, for example, []).

10.4 Anisotropic Scaling

One important difficulty with scale space models is that the symmetric Gaussian smoothing process tends to blur out edges rather two aggressively for comfort. For example, if we have two trees near one another on a skyline, the large scale blobs corresponding to each tree may start merging before all the small scale blobs have finished. This suggests that we should smooth differently at edge points than at other points. For example, we might make an estimate of the magnitude and orientation of the gradient: for large gradients, we would then use an oriented smoothing operator that smoothed aggressively perpendicular to the gradient and very little along the gradient; for small gradients, we might use a symmetric smoothing operator. This idea used to be known as **edge preserving smoothing**.

In the modern, more formal version (details in in []), we notice the scale space representation family is a solution to the **diffusion equation**

$$\begin{aligned}\frac{\partial \Phi}{\partial \sigma} &= \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} \\ &= \nabla^2 \Phi \\ \Phi(x, y, 0) &= \mathcal{I}(x, y)\end{aligned}$$

If this equation is modified to have the form

$$\begin{aligned}\frac{\partial \Phi}{\partial \sigma} &= \nabla \cdot (c(x, y, \sigma) \nabla \Phi) \\ &= c(x, y, \sigma) \nabla^2 \Phi + (\nabla c(x, y, \sigma)) \cdot (\nabla \Phi) \\ \Phi(x, y, 0) &= \mathcal{I}(x, y)\end{aligned}$$

then if $c(x, y, \sigma) = 1$, we have the diffusion equation we started with, and if $c(x, y, \sigma) = 0$ there is no smoothing. We will assume that c does not depend on σ. If we knew where the edges were in the image, we could construct a mask that consisted of regions where $c(x, y) = 1$, isolated by patches along the edges where $c(x, y) = 0$; in this case, a solution would smooth *inside* each separate region, but not over the edge. While we do not know where the edges are — the exercise would be empty if we did — we can obtain reasonable choices of $c(x, y)$ from the magnitude of the image gradient. If the gradient is large, then c should be small, and vice-versa.