Assume a fixed scale parameter k
Find all locations and scales which are local extrema of
  $\nabla^2_\sigma \mathcal{I}(x, y)$ in location (x, y) and scale σ forming a list of triples (x_c, y_c, r)
For each such triple
  Compute an orientation histogram $H(\theta)$ for gradient orientations within
    a radius kr of (x_c, y_c).
  Compute the orientation of the patch θ_p as
    $\theta_p = \underset{\theta}{\text{argmax}}\ H(\theta)$. If there is more than one θ that
    maximizes this histogram, make one copy of the patch for each.
  Attach (x_c, y_c, r, θ_p) to the list of patches for each copy

Algorithm 5.3: Obtaining Location, Radius, and Orientation of Pattern Elements Using the Laplacian of Gaussian.

obtained by using a corner detector and then estimating scale. Corner detectors respond to a corner structure at the point of interest; the Laplacian of Gaussian looks for structures that look like a circular blob of a particular scale centered at the point of interest. Corner detectors tend to produce neighborhoods where the estimate of the center is very accurate, but the scale estimate is poor. These are most useful in matching problems where we don't expect the scale to change much. Laplacian of Gaussian methods produce neighborhoods where the estimate of the center is less accurate, but the scale estimate is better. These are most useful in matching problems where large changes of scale might appear.

As we have seen, orientation histograms are a natural representation of image patches. However, we cannot represent orientations in image coordinates (for example, using the angle to the horizontal image axis), because the patch we are matching to might have been rotated. We need a reference orientation so all angles can be measured with respect to that reference. A natural reference orientation is the most common orientation in the patch. We compute a histogram of the gradient orientations in this patch, and find the largest peak. This peak is the reference orientation for the patch. If there are two or more peaks of the same magnitude, we make multiple copies of the patch, one at each peak orientation. The whole process is summarized in Algorithms 5.2 and 5.3. These estimates of patch neighborhoods are remarkably well behaved (Figure 5.13).

5.4 DESCRIBING NEIGHBORHOODS WITH SIFT AND HOG FEATURES

We know the center, radius, and orientation of a set of an image patch, and must now represent it. Orientations should provide a good representation. They are unaffected by changes in image brightness, and different textures tend to have different orientation fields. The pattern of orientations in different parts of the patch is likely to be quite distinctive. Our representation should be robust to small errors in the center, radius, or orientation of the patch, because we are unlikely to estimate these exactly right.

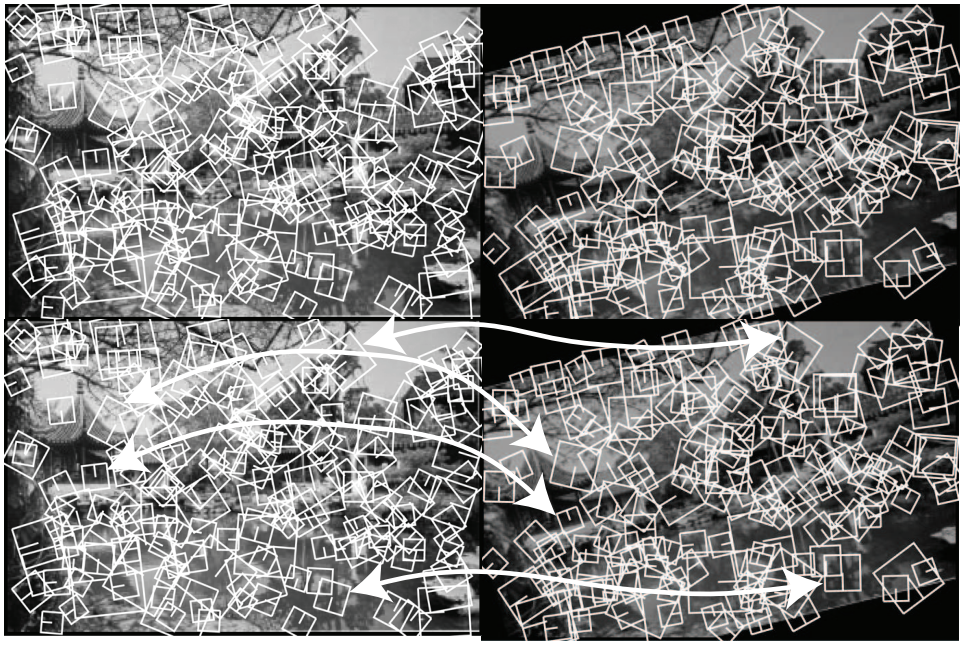

FIGURE 5.13: This figure shows local patches recovered using a method similar to that described in the text (the details of the corner detector were different). These patches are plotted as squares, rather than as circles. The location of the patch is the center of the square. The reference orientation of the patch is given by the line segment in the square, and the scale is the size of the square. The image on the **right** has been scaled, rotated, and translated to produce the image on the **left**. Notice that (a) most of the patches on the right have corresponding patches on the left and (b) the corresponding patches are translated, rotated, and scaled versions of the original patches. You can check this by looking at the grayscale version of the image. We have shown some of the many corresponding pairs of patches (**below**; the large white arrows). *This figure was originally published as Figure 1 of "Object recognition from local scale-invariant features" D.G. Lowe, Proc. IEEE ICCV, 1999* © *IEEE 1999.*

You should think of these neighborhoods as being made up of pattern elements. In this case, the pattern elements will be orientations, but we will use this trick again for other kinds of pattern element. These elements move around somewhat inside the neighborhood (because we might not get the center right), but if most elements are there and are in about the right place, then the neighborhood has the right properties. We must build features that can make it obvious whether the pattern elements are present, and whether they are in about the right place, but are not affected by some rearrangement.

The most obvious approach is to represent the neighborhood with a histogram of the elements that appear there. This will tell us what is present, but it confuses too many patterns with one another. For example, all neighborhoods with vertical stripes will get mixed up, however wide the stripe. The natural approach is to take histograms locally, within subpatches of the neighborhood. This leads to a very

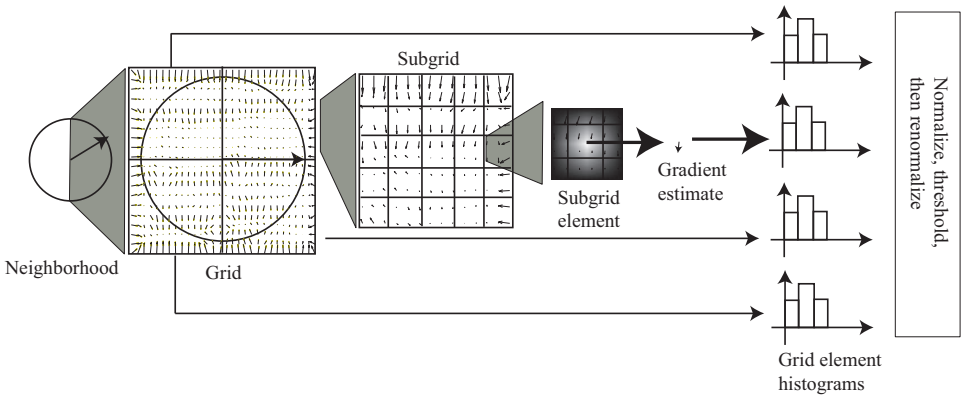

FIGURE 5.14: To construct a SIFT descriptor for a neighborhood, we place a grid over the rectified neighborhood. Each grid is divided into a subgrid, and a gradient estimate is computed at the center of each subgrid element. This gradient estimate is a weighted average of nearby gradients, with weights chosen so that gradients outside the subgrid cell contribute. The gradient estimates in each subgrid element are accumulated into an orientation histogram. Each gradient votes for its orientation, with a vote weighted by its magnitude and by its distance to the center of the neighborhood. The resulting orientation histograms are stacked to give a single feature vector. This is normalized to have unit norm; then terms in the normalized feature vector are thresholded, and the vector is normalized again.

important feature construction.

5.4.1 SIFT Features

We can now compute a representation that is not affected by translation, rotation, or scale. For each patch, we rectify the patch by translating the center to the origin, rotating so the orientation direction lies along (say) the x-axis, and scaling so the radius is one. Any representation we compute for this rectified patch will be invariant to translations, rotations, and scale. Although we do not need to rectify in practice—instead, we can work the rectification into each step of computing the description—it helps to think about computing descriptions for a rectified patch.

A *SIFT descriptor* (for Scale Invariant Feature Transform) is constructed out of image gradients, and uses both magnitude and orientation. The descriptor is normalized to suppress the effects of change in illumination intensity. The descriptor is a set of histograms of image gradients that are then normalized. These histograms expose general spatial trends in the image gradients in the patch but suppress detail. For example, if we estimate the center, scale, or orientation of the patch slightly wrong, then the rectified patch will shift slightly. As a result, simply recording the gradient at each point yields a representation that changes between instances of the patch. A histogram of gradients will be robust to these changes. Rather than histogramming the gradient at a set of sample points, we histogram local averages of image gradients; this helps avoid noise.

The standard SIFT descriptor is obtained by first dividing the rectified patch

into an $n \times n$ grid. We then subdivide each grid element into an $m \times m$ subgrid of subcells. At the center of each subcell, we compute a gradient estimate. The gradient estimate is obtained as a weighted average of gradients around the center of the cell, weighting each by $(1 - d_x/s_x)(1 - d_y/s_y)/N$, where d_x (resp. d_y) is the x (resp. y) distance from the gradient to the center of the subcell, and s_x (resp. s_y) is the x (resp. y) spacing between the subcell centers. This means that gradients make contributions to more than one subcell, so that a small error in the location of the center of the patch leads to a small change in the descriptor.

We now use these gradient estimates to produce histograms. Each grid element has a q-cell orientation histogram. The magnitude of each gradient estimate is accumulated into the histogram cell corresponding to its orientation; the magnitude is weighted by a Gaussian in distance from the center of the patch, using a standard deviation of half the patch.

We concatenate each histogram into a vector of $n \times n \times q$ entries. If the image intensity were doubled, this vector's length would double (because the histogram entries are sums of gradient magnitudes). To avoid this effect, we normalize this vector to have unit length. Very large gradient magnitude estimates tend to be unstable (for example, they might result from a lucky arrangement of surfaces in 3D so that one faces the light directly and another points away from the light). This means that large entries in the normalized vector are untrustworthy. To avoid difficulties with large gradient magnitudes, each value in the normalized vector is thresholded with threshold t, and the resulting vector is renormalized. The whole process is summarized in Algorithm 5.4 and Figure 5.14. Standard parameter values are $n = 4$, $m = 4$, $q = 8$, and $t = 0.2$.

Given an image $\mathcal{I}$, and a patch with center (x_c, y_c),
  radius r, orientation θ, and parameters n, m, q, k and t.
For each element of the $n \times n$ grid centered at (x_c, y_c) with spacing kr
  Compute a weighted q element histogram of the averaged
    gradient samples at each point of the $m \times m$ subgrid,
    as in Algorithm 5.5.
Form an $n \times n \times q$ vector $\boldsymbol{v}$ by concatenating the histograms.
Compute $\boldsymbol{u} = \boldsymbol{v}/\sqrt{\boldsymbol{v} \cdot \boldsymbol{v}}$.
Form $\boldsymbol{w}$ whose i'th element w_i is $\min(u_i, t)$.
The descriptor is $\boldsymbol{d} = \boldsymbol{w}/\sqrt{\boldsymbol{w} \cdot \boldsymbol{w}}$.

Algorithm 5.4: Computing a SIFT Descriptor in a Patch Using Location, Orientation and Scale.

There is now extensive experimental evidence that image patches that match one another will have similar SIFT feature representations, and patches that do not will tend not to. SIFT features can be used to represent the local color pattern around a sample point, too. The natural procedure is to apply SIFT feature code to a color representation. For example, one could compute SIFT features for each of the hue, saturation, and value channels (HSV-SIFT; see Bosch *et al.* (2008)); for the opponent color channels (OpponentSIFT, which uses R-G and B-Y; see van de

Given a grid cell $\mathcal{G}$ for patch with center $\boldsymbol{c} = (x_c, y_c)$ and radius r

Create an orientation histogram
For each point $\boldsymbol{p}$ in an $m \times m$ subgrid spanning $\mathcal{G}$
  Compute a gradient estimate $\nabla\mathcal{I}\,|_{\boldsymbol{p}}$ estimate at $\boldsymbol{p}$
    as a weighted average of $\nabla\mathcal{I}$, using bilinear weights centered at $\boldsymbol{p}$.
  Add a vote with weight $\|\nabla\mathcal{I}\|\frac{1}{r\sqrt{2\pi}}\exp\left(-\frac{\|\boldsymbol{p}-\boldsymbol{c}\|^2}{r^2}\right)$
    to the orientation histogram cell for the orientation of $\nabla\mathcal{I}$.

Algorithm 5.5: Computing a Weighted q Element Histogram for a SIFT Feature.

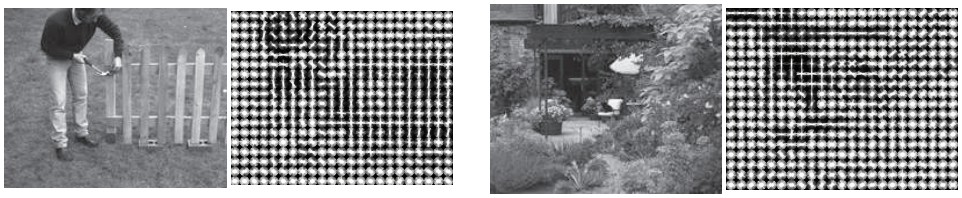

FIGURE 5.15: The HOG features for each the two images shown here have been visualized by a version of the rose diagram of Figures 5.7–5.9. Here each of the cells in which the histogram is taken is plotted with a little rose in it; the direction plotted is at right angles to the gradient, so you should visualize the overlaid line segments as edge directions. Notice that in the textured regions the edge directions are fairly uniformly distributed, but strong contours (the gardener, the fence on the **left**; the vertical edges of the french windows on the **right**) are very clear. This figure was plotted using the toolbox of Dollár and Rabaud. *Left:* © *Dorling Kindersley, used with permission. Right: Geoff Brightling* © *Dorling Kindersley, used with permission.*

Sande *et al.* (2010)); for normalised opponent color channels (C-SIFT, which uses $(R-G)/(R+G+B)$ and $(B-Y)/(R+G+B)$; see Abdel Hakim and Farag (2006); Geusebroek *et al.* (2001); or Burghouts and Geusebroek (2009)); and for normalized color channels (rgSIFT, which uses $R/(R+G+B)$ and $G/(R+G+B)$; see van de Sande *et al.* (2010)). Each of these features will behave slightly differently when the light falling on an object changes, and each can be used in place of, or in addition to, SIFT features.

5.4.2 HOG Features

The *HOG feature* (for Histogram Of Gradient orientations) is an important variant of the SIFT feature. Again, we histogram gradient orientations in cells, but now adjust the process to try and identify high-contrast edges. We can recover contrast information by counting gradient orientations with weights that reflect how significant a gradient is compared to other gradients in the same cell. This means that, rather than normalize gradient contributions over the whole neighborhood, we normalize with respect to nearby gradients only. Normalization could occur on a grid of cells that is different from the orientation subgrid, too. A single gradient

location might contribute to several different histograms, normalized in somewhat different ways; this means we will be relatively unlikely to miss boundaries that have low contrast.

Write $\|\nabla I_{\mathbf{x}}\|$ for the gradient magnitude at point $\mathbf{x}$ in the image. Write $\mathcal{C}$ for the cell whose histogram we wish to compute and $w_{\mathbf{x},\mathcal{C}}$ for the weight that we will use for the orientation at $\mathbf{x}$ for this cell. A natural choice of weight is

$$w_{\mathbf{x},\mathcal{C}} = \frac{\|\nabla I_{\mathbf{x}}\|}{\sum_{\mathbf{u}\in\mathcal{C}} \|\nabla I_{\mathbf{u}}\|}.$$

This compares the gradient magnitude to others in the cell, so that gradients that are large compared to their neighbors get a large weight. This normalization process means that HOG features are quite good at picking outline curves out of confusing backgrounds (Figure 5.15).

5.5 COMPUTING LOCAL FEATURES IN PRACTICE

We have sketched the most important feature constructions, but there is a huge range of variants. Performance is affected by quite detailed questions, such as the extent of smoothing when evaluating orientations. Space doesn't allow a detailed survey of these questions (though there's some material in Section 5.6), and the answers seem to change fairly frequently, too. This means we simply can't supply accurate recipes for building each of these features.

Fortunately, at time of writing, there are several software packages that provide good implementations of each of these feature types, and of other variations. Piotr Dollár and Vincent Rabaud publish a toolbox at `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`; we used this to generate several figures. VLFeat is a comprehensive open-source package that provides SIFT features, vector quantization by a variety of methods, and a variety of other representations. At time of writing, it could be obtained from `http://www.vlfeat.org/`. SIFT features are patented (Lowe 2004), but David Lowe (the inventor) provides a reference object code implementation at `http://www.cs.ubc.ca/~lowe/keypoints/`. Navneet Dalal, one of the authors of the original HOG feature paper, provides an implementation at `http://www.navneetdalal.com/software/`. One variant of SIFT is PCA-SIFT, where one uses principal components to reduce the dimension of the SIFT representation (Ke and Sukthankar 2004). Yan Ke, one of the authors of the original PCA-SIFT paper, provides an implementation at `http://www.cs.cmu.edu/~yke/pcasift/`. Color descriptor code, which computes visual words based on various color SIFT features, is published by van de Sande *et al.* at `http://koen.me/research/colordescriptors/`.

5.6 NOTES

Edges

There is a huge edge detection literature. The earliest paper of which we are aware is Julez (1959) (yes, 1959!). Those wishing to be acquainted with the early literature in detail should start with a 1975 survey by Davis (1975); Herskovits and Binford (1970); Horn (1971); and Hueckel (1971), who models edges and then detects the model. There are many optimality criteria for edge detectors, and rather more

"optimal" edge detectors. The key paper in this literature is by Canny (1986); significant variants are due to Deriche (1987) and to Spacek (1986). Faugeras' textbook contains a detailed and accessible exposition of the main issues Faugeras (1993). At the end of the day, most variants boil down to smoothing the image with something that looks a lot like a Gaussian before measuring the gradient. All edge detectors behave badly at corners; only the details vary.

Object boundaries are not the same as sharp changes in image values. There is a vast literature seeking to build boundary detectors; we can provide only some pointers. The reader could start with Bergholm (1987), Deriche (1990), Elder and Zucker (1998), Fleck (1992), Kube and Perona (1996), Olson (1998), Perona and Malik (1990*b*), or Torre and Poggio (1986). The best current boundary detector takes quite a lot of local information into account, and is described in Section 17.1.3.

The edges that our edge detectors respond to are sometimes called *step edges* because they consist of a sharp, "discontinuous" change in value that is sometimes modeled as a step. A variety of other forms of edge have been studied. The most commonly cited example is the *roof edge*, which consists of a rising segment meeting a falling segment, rather like some of the reflexes that can result from the effects of interreflections. Another example that also results from interreflections is a composite of a step and a roof. It is possible to find these phenomena by using essentially the same steps as outlined before (find an "optimal" filter, and do nonmaximum suppression on its outputs) (Canny 1986, Perona and Malik 1990*a*). In practice, this is seldom done. There appear to be two reasons. First, there is no comfortable basis in theory (or practice) for the models that are adopted. What particular composite edges are worth looking for? The easy answer—those for which optimal filters are reasonably easy to derive—is most unsatisfactory. Second, the semantics of roof edges and more complex composite edges is even vaguer than that of step edges. There is little notion of what one would *do* with roof edge once it had been found.

Corners, Neighborhoods, and Interest Points

The first corner detector we know of is due to Moravec (1980). Corner detectors are now very well studied (there is an excellent Wikipedia page that describes the various detectors and their relations at `http://en.wikipedia.org/wiki/Corner_detection`). The Harris and Stephens detector we described remains competitive. Important variants look at different eigenvalue criteria (Tomasi and Shi 1994); differential geometric criteria (Wang and Brady 1994); multiple scales (Lindeberg 1993); local self-similarity measures (Smith and Brady 1997, Trajkovic and Hedley 1998); and machine learning (Rosten *et al.* 2010).

For simplicity of exposition, we have elided corners and interest points (the other name under which corners are often studied). Interest points are usually thought of as a corner (or something like it) together with a neighborhood, covariant under some form of transformation. We like to see detecting the points and estimating their neighborhoods as distinct processes, though for strict covariance both the detector and the neighborhood estimator must be covariant. Various detectors are scale covariant (Mikolajczyk and Schmid 2002); affine covariant (Mikolajczyk and Schmid 2002); and illumination robust (Gevrekci and Gunturk 2009). The idea can

be extended to spatio-temporal representations (Willems *et al.* 2008, Laptev 2005). There are now detailed experimental studies of the performance of interest point detectors (Schmid *et al.* 2000, Privitera and Stark 1998, Mikolajczyk *et al.* 2005).

Descriptors

The tricks to describing neighborhoods seem to be: describe a local texture pattern within a covariant neighborhood; work with orientations, because they're illumination invariant; and use histograms to suppress spatial detail, working with more detail at the center than at the boundary. These tricks appear in numerous papers in a variety of forms (e.g., Schmid and Mohr (1997); Belongie *et al.* (2001); Berg *et al.* (2005)), but SIFT and Hog features now dominate. Comparisons between local descriptors seem to support this dominance (Mikolajczyk and Schmid 2005).

PROBLEMS

5.1. Each pixel value in 500×500 pixel image $\mathcal{I}$ is an independent, normally distributed random variable with zero mean and standard deviation one. Estimate the number of pixels that, where the absolute value of the x derivative, estimated by forward differences (i.e., $|I_{i+1,j} - I_{i,j}|$, is greater than 3.

5.2. Each pixel value in 500×500 pixel image $\mathcal{I}$ is an independent, normally distributed random variable with zero mean and standard deviation one. $\mathcal{I}$ is convolved with the $2k + 1 \times 2k + 1$ kernel $\mathcal{G}$. What is the covariance of pixel values in the result? There are two ways to do this; on a case-by-case basis (e.g., at points that are greater than $2k+1$ apart in either the x or y direction, the values are clearly independent) or in one fell swoop. Don't worry about the pixel values at the boundary.

5.3. We have a camera that can produce output values that are integers in the range from 0 to 255. Its spatial resolution is 1024 by 768 pixels, and it produces 30 frames a second. We point it at a scene that, in the absence of noise, would produce the constant value 128. The output of the camera is subject to noise that we model as zero mean stationary additive Gaussian noise with a standard deviation of 1. How long must we wait before the noise model predicts that we should see a pixel with a negative value? (Hint: You may find it helpful to use logarithms to compute the answer as a straightforward evaluation of $\exp(-128^2/2)$ will yield 0; the trick is to get the large positive and large negative logarithms to cancel.)

5.4. Show that for a 2×2 matrix $\mathcal{H}$, with eigenvalues λ_1, λ_2

(a) $\det\mathcal{H} = \lambda_1\lambda_2$

(b) $\text{trace}\mathcal{H} = \lambda_1 + \lambda_2$

PROGRAMMING EXERCISES

5.5. The Laplacian of a Gaussian looks similar to the difference between two Gaussians at different scales. Compare these two kernels for various values of the two scales. Which choices give a good approximation? How significant is the approximation error in edge finding using a zero-crossing approach?

5.6. Obtain an implementation of Canny's edge detector (you could try the vision home page; MATLAB also has an implementation in the image processing toolbox), and make a series of images indicating the effects of scale and contrast thresholds on the edges that are detected. How easy is it to set up the edge detector to mark only object boundaries? Can you think of applications where

this would be easy?

5.7. It is quite easy to defeat hysteresis in edge detectors that implement it; essentially, one sets the lower and higher thresholds to have the same value. Use this trick to compare the behavior of an edge detector with and without hysteresis. There are a variety of issues to look at:

(a) What are you trying to do with the edge detector output? It is sometimes helpful to have linked chains of edge points. Does hysteresis help significantly here?

(b) Noise suppression: We often wish to force edge detectors to ignore some edge points and mark others. One diagnostic that an edge is useful is high contrast (it is by no means reliable). How reliably can you use hysteresis to suppress low-contrast edges without breaking high-contrast edges?

5.8. Build a Harris corner detector; for each corner, estimate scale and orientation as we have described. Now test how well your list of neighborhoods behaves under rotation, translation, and scale of the image. You can do this by a simple exercise in matching. For each test image, prepare a rotated, translated, and scaled version of that image. Now you know where each neighborhood should appear in the new version of the image — check how often something of the right size and orientation appears in the right place. You should find that rotation and translation cause no significant problems, but large scale changes can be an issue.