CHAPTER 16

Classifying Images

Many modern computer vision problems can be solved by using a classifier. Here we will survey a set of applications where one passes a whole image into a classifier, and in the next chapter show an extremely important extension where one applies the classifier to windows in the image. The recipe is straightforward: one finds a labelled dataset, builds features, and then trains a classifier. Even better, there is a set of reliable feature building tricks that apply in many important cases. This recipe is so powerful and has proven so effective that it is worth a considerable effort to rephrase a problem into a form in which it applies; and whenever it does apply, it is essential to check how well it works before doing anything more elaborate.

Each application needs a set of features that can represent the image appearance usefully. Section 16.1 describes general tricks for building appearance features in the context of some particular applications. We then look at the general problem of image classification, where one takes a test image and must classify it into one of a set of categories (Section 16.2). There are two important threads in the current state of the art: Building methods that perform better on a fixed set of categories (Section 16.2.3); and building methods that apply to increasingly large numbers of categories (Section 16.2.4). Finally, Section 16.3 gives pointers to software and datasets that are useful for research in this area.

16.1 BUILDING GOOD IMAGE FEATURES

The core difficulty in applying our recipe is choosing good image features. Different feature constructions are good for different applications. The key is to build features that expose *between-class variation*, which is the tendency for classes to look different from one another, and suppress *within-class variation*, the tendency for instances within a class to look different. Some feature constructions seem to be quite good at this for many problems, but most problems have special properties.

16.1.1 Example Applications

Detecting explicit images: There are numerous reasons to try and detect images that depict nudity or sexual content. In some jurisdictions, possession or distribution of such pictures might be illegal. Many employers want to ensure that work computers are not used to collect or view such images; in fact, vendors of image filtering software have tried to persuade employers that they might face litigation if they don't do so. Advertisers want to be careful that their ads appear only next to images that will not distress their customers. Web search companies would like to allow users to avoid seeing images they find distressing.

It is difficult for jurists to be clear about what images are acceptable and what are not. In the United States, images that are not obscene have first amendment protections, but the test for whether an image is obscene is far too vague to be useful

FIGURE 16.1: Material is not the same as object category (the three cars on the **top** are each made of different materials), and is not the same as texture (the three checkered objects on the **bottom** are made of different materials). Knowing the material that makes up an object gives us a useful description, somewhat distinct from its identity and its texture. *This figure was originally published as Figures 2 and 3 of "Exploring Features in a Bayesian Framework for Material Recognition," by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz Proc. CVPR 2010, 2010* © *IEEE, 2010.*

to the technical community (even the legal community finds it tricky; see histories in, for example, O'Brien (2010) or de Grazia (1993)). For most applications, it is enough to filter pictures that likely show nakedness or sexual content, and that could be done with a classifier. Much of the research on this topic is done at large industrial laboratories, behind a wall of secrecy. All published methods rely on finding skin in the image; some then reason about the layout of the skin. For us, there are two classification steps: we need classify pixels into skin and not-skin; and we need to classify images into explicit vs. not-explicit based on the layout of skin.

Material classification: Imagine we have an image window. What material (e.g., "wood," "glass," "rubber," and "leather") does the window cover? If we could answer this question, we could decide whether an image patch was cloth—and so might be part of a person, or of furniture—or grass, or trees, or sky. Generally, different materials produce different image textures, so natural features to use for this multiclass classification problem will be texture features. However, materials tend to have some surface relief (for example, the little pores on the surface of an orange; the bumps on stucco; the grooves in tree bark), and these generate quite complex shading behavior. Changes in the illumination direction can cause the shadows attached to the relief to change quite sharply, and so the overall texture changes. Furthermore, as Figure 16.1 illustrates, texture features might indicate the material an object is made of, but objects can have the same texture and be made of quite different materials.

Scene classification: Pictures of a bedroom, of a kitchen, or of a beach show different *scenes*. Scenes provide an important source of context to use in

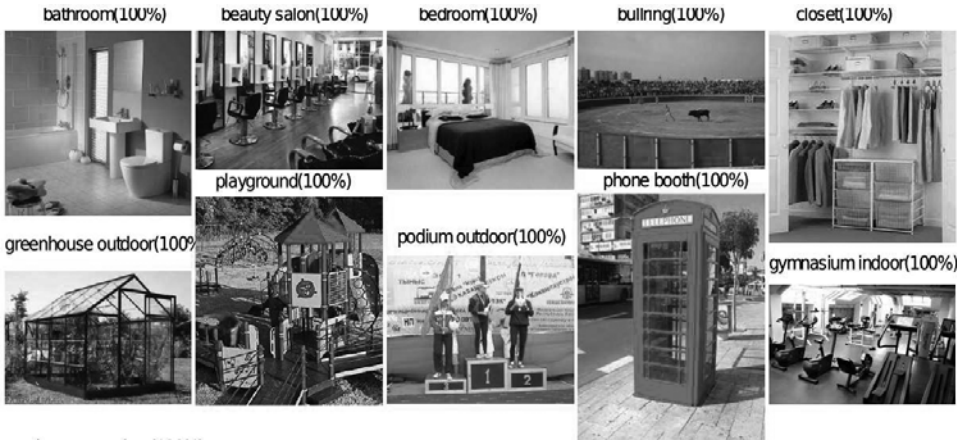

FIGURE 16.2: Some scenes are easily identified by humans. These are examples from the SUN dataset (Xiao *et al.* 2010) of scene categories that people identify accurately from images; the label above each image gives its scene type. *This figure was originally published as Figure 2 of "SUN database: Large-scale Scene Recognition from Abbey to Zoo," by J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba, Proc. IEEE CVPR 2010, © IEEE, 2010.*

interpreting images. You could reasonably expect to see a pillow, but not a toaster or a beachball, in a bedroom; a toaster, but not a pillow or a beachball, in a kitchen; or a beachball and perhaps a pillow, but not a toaster, on a beach. We should like to be able to tell what scene is depicted in an image. This is difficult, because scenes vary quite widely in appearance, and this variation has a strong spatial component. For example, the toaster could be in many different locations in the kitchen. In *scene classification*, one must identify the scene depicted in the image. Scene labels are more arbitrary than object labels, because there is no clear consensus on what the different labels should be. Some labels seem fairly clear and are easy to assign accurately (Figure 16.2), for example, "kitchen," "bedroom," and other names of rooms in a house. Other labels are uncertain: should one distinguish between "woodland paths" and "meadows", or just label them all "outdoors"? Humans seem to have a problem here, too (Figure 16.18). However, there are several scene datasets, each with its own set of labels, so methods can be compared and evaluated.

There are some important general points about these and most other applications, that can guide feature construction. Any representation should be robust to rotation, translation, or scaling of the image, because these transformations will not affect the label of the image (an explicit image is an explicit image, even when it is upside down). The driving observations behind the SIFT and HOG feature constructions are (a) **exact intensity values are not important**, because we might encounter versions of the image taken under brighter or darker illumination; and (b) **image curves are important**, because they might be object outlines. As we saw in Section 5.4, these observations justified working with gradient orientations and normalizing the resulting features in various ways. There are two more

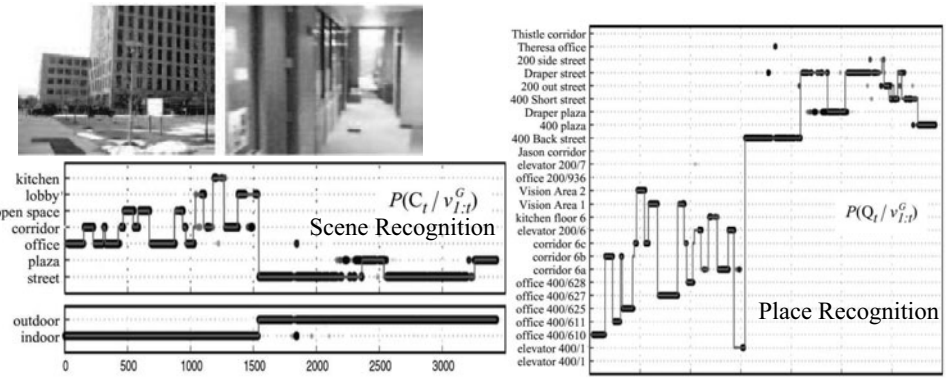

FIGURE 16.3: GIST features can be used to identify scenes, particularly the place where the image was taken. Torralba *et al.* (2003) demonstrated a vision system that moves through a known environment, and can tell where it is from what it sees using scene recognition ideas. Images (examples on the **top left**) are represented with GIST features. These are used to compute a posterior probability of place conditioned on observations and the place of the last image, which is shown on the **right**. The shaded blobs correspond to posterior probability, with darker blobs having higher probability. The thin line superimposed on the figure gives the correct answer; notice that almost all probability lies on the right answer. For places that are not known, the type of place can be estimated (**bottom left**); again, the shaded blobs give posterior probability, darker blobs having higher probability, and the thin line gives the right answer. *This figure was originally published as Figures 2 and 3 of "Context-based vision system for place and object recognition," by A. Torralba, K. Murphy, W.T. Freeman, and M.A. Rubin, Proc. IEEE ICCV 2003,* © *IEEE 2003.*

important observations. First, **image texture is important**, and is usually highly diagnostic. Although this isn't at all obvious, it turns out to be an essential part of building good features. This suggests looking at summary statistics of orientation (for example, horizontal stripes give lots of vertically oriented gradients, spotty regions should have uniformly distributed orientations, and so on). Second, **exact feature locations are not important**, because small changes in the layout of the image would not change its class. For example, moving the toaster on a shelf doesn't stop the kitchen from being a kitchen. We have seen a version of this issue before, at a finer spatial scale. The histogramming step in SIFT and HOG features tries to account for small shifts in the location of orientation components, local to a particular neighborhood, by summarizing that neighborhood. We will use similar summarization mechanisms to deal with larger scale shifts of larger structures.

16.1.2 Encoding Layout with GIST Features

One natural cue to the scene is the overall layout of a picture. If there are large, smooth regions on either side, many vertical straight lines, and relatively little sky visible, then you might be in an urban canyon; if there is a lot of sky visible, and rough brown stuff at the bottom of the picture, then you might be outdoors; and so on. There is a lot of evidence that people can make very fast, accurate

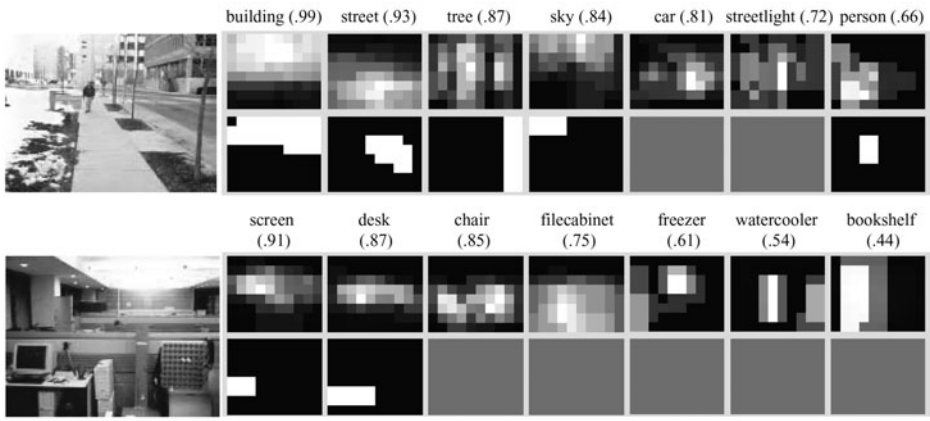

FIGURE 16.4: Scenes are important, because knowing the type of scene shown in an image gives us some information about the objects that are present. For example, street's are typically at the bottom center of street scenes. These maps show probabilities of object locations (**top row**, for each image) extracted from scene information for the image to the left; brighter values are higher probabilities. Compare these with the true support of the object (**bottom row**, for each image); notice that, while knowing the scene doesn't guarantee that an object is present, it does suggest where it is likely to be. This could be used to cue object detection processes. *This figure was originally published as Figure 10 of "Context-based vision system for place and object recognition," by A. Torralba, K. Murphy, W.T. Freeman, and M.A. Rubin, Proc. IEEE ICCV 2003,* © *IEEE 2003.*

judgments about pictures, which appear to be based on the overall layout of the picture (Henderson and Hollingworth 1999).

GIST features attempt to capture this layout. Oliva and Torralba (2001) constructed these features by reasoning about a set of perceptual dimensions that might encode the layout of a scene. The dimensions include whether the scene is natural or man-made; whether there is wide-open space or just a narrow enclosure; whether it is rugged or not. They then build features that tend to be good at predicting these dimensions. These features typically result from a spectral analysis of all or part of the scene. For example, images that show urban canyons have lots of strong vertical edges, which will mean high energy at high spatial frequencies at particular (vertical) phases; similarly, ruggedness will translate into strong energy at high spatial frequencies.

A natural feature will be comparable to the texture representations of Chapter 6, but summarized to represent the whole image. Oliva and Torralba apply a bank of oriented filters at a range of scales (eight orientations and four scales). They then average the magnitude of the filter output over a four by four grid of non-overlapping windows. The result is a 512 ($= 4 \times 4 \times 8 \times 4$) dimensional vector. This is then projected onto a set of principal components computed on a large dataset of natural images. The result is a set of features that (a) give a sense of the strength of texture activity at various scales and orientations in various blocks of the image and (b) tend to differ between natural scenes. These features are now

FIGURE 16.5: The original application of visual word representations was to search video sequences for particular patterns. On the **left**, a user has drawn a box around a pattern of interest in a frame of video; the **center** shows a close-up of the box. On the **right**, we see neighborhoods computed from this box. These neighborhoods are ellipses, rather than circles; this means that they are covariant under affine transforms. Equivalently, the neighborhood constructed for an affine transformed patch image will be the affine transform of the neighborhood constructed for the original patch (definition in Section 5.3.2). *This figure was originally published as Figure 11 of J. Sivic and A. Zisserman "Efficient Visual Search for Objects in Videos," Proc. IEEE, Vol. 96, No. 4, April 2008* © *IEEE 2008.*

very widely used. There is strong evidence that they do encode scene layout (for example Oliva and Torralba (2007); Oliva and Torralba (2001); or Torralba *et al.* (2003)), and they are widely used in applications where scene context is likely to help performance.

16.1.3 Summarizing Images with Visual Words

Features that represent scenes should summarize. It is generally more important to know that something is present (the toaster in our example), than to know where it is. This suggests using a representation that has the form of a histogram. Such histograms are useful for other cases, too. Imagine we would like to classify images containing large, relatively isolated objects. The objects might deform, the viewpoint might move, and the image might have been rotated or scaled. Apart from these effects, we expect the object appearance to be fairly stable (that is, we're not trying to match a striped object with a spotted object). This means that the absolute location of structures in the image is probably not very informative, but the presence of these structures is very important. Again, this suggests a representation that is like a histogram. The big question is what to record in the histogram.

An extremely successful answer is to record characteristic local image patches. When we discussed texture, we called these textons (Section 6.2), but in recognition applications they tend to be called *visual words*. The construction follows the same lines. We detect interest points and build neighborhoods around them (Section 5.3). We then describe those neighborhoods with SIFT features (Section 5.4). We vector quantize these descriptions, then build a representation of the overall pattern of vector-quantized neighborhoods.

There are many plausible strategies to vector quantize SIFT descriptors. For concreteness, assume we obtain a large training set of SIFT descriptors from relevant images and cluster them with k-means. We now vector quantize a new SIFT

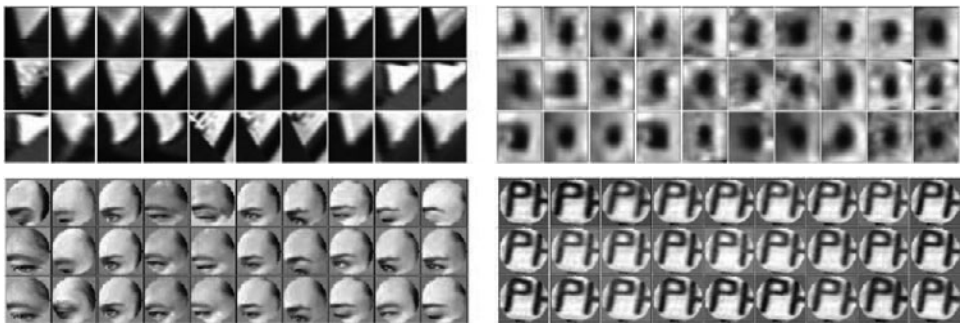

FIGURE 16.6: Visual words are obtained by vector quantizing neighborhoods like those shown in Figure 16.5. This figure shows 30 examples each of instances of four different visual words. Notice that the words represent a moderate-scale local structure in the image (an eye, one and a half letters, and so on). Typical vocabularies are now very large, which means that the instances of each separate word tend to look a lot like one another. *This figure was originally published as Figure 3 of "Efficient Visual Search for Objects in Videos," by J. Sivic and A. Zisserman, Proc. IEEE, Vol. 96, No. 4, April 2008* © *IEEE 2008.*

descriptor by replacing it with the number of the closest cluster center. The resulting numbers are very like words (for example, you can count the number of times a particular type of interest point occurs in an image, and images with similar counts are likely to be similar). Sivic and Zisserman (2003), who pioneered the approach, call these numbers *visual words*. Perhaps the most important difference between visual words and the textons of Section 6.2.1 is that a visual word could describe a larger image domain than a texton might.

Although individual visual words should be somewhat noisy, the overall picture of local patches should be the same in comparable images (or regions). For example, Figure 16.7 is relatively typical of visual word representations. Not all the neighborhoods in the query were matched in each response. Noise could be caused by the neighborhood procedure not finding the correct neighborhood, or by the vector quantization sending the neighborhood to the wrong visual word. As a result, we need to summarize the set of visual words in a way that is robust to errors. In practice, histograms are an excellent summary. If most of the words in one image match most of the words in the other image, then the histograms should be similar. Furthermore, the histograms should not be significantly affected by change of image intensity, rotations, scaling, and deformations.

In the histogram representation, two images that are similar should have similar histograms, and two images that are different will have different histograms. This means that it is somewhat unnatural (and, in practice, not particularly effective) to simply apply a linear classifier to a histogram represented as a vector. We expect positive (resp. negative) examples to lie on a fairly complex structure in this feature space, and the classification procedure should compare test examples to multiple training examples. This means that kernel methods (Section 15.2.5) are particularly well adapted to histogram features. One can use the χ-*squared kernel*,

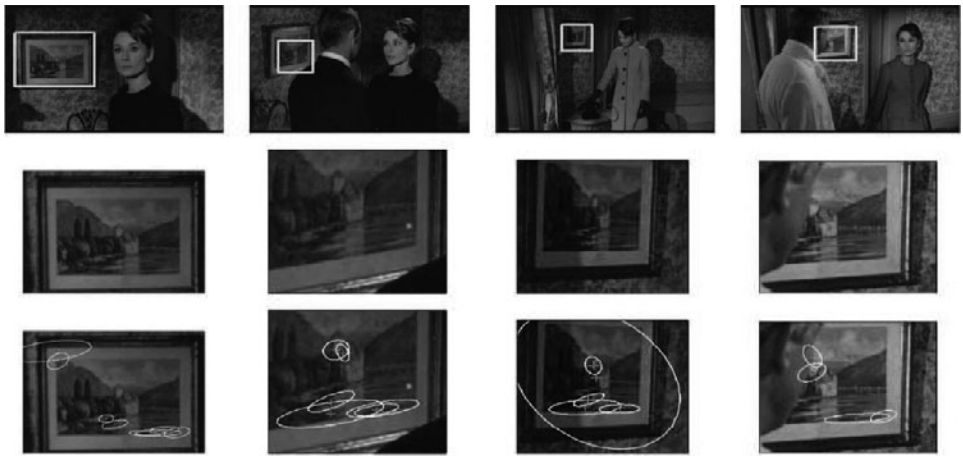

FIGURE 16.7: This figure shows results from the query of Figure 16.5, obtained by looking for image regions that have a set of visual words strongly similar to those found in the query region. The first row shows the whole frame from the video sequence; the second row shows a close-up of the box that is the result (indicated in the first row); and the third row shows the neighborhoods in that box that generated visual words that match those in the query. Notice that some, but not all, of the neighborhoods in the query were matched. *This figure was originally published as Figure 11 of J. Sivic and A. Zisserman "Efficient Visual Search for Objects in Videos," Proc. IEEE, Vol. 96, No. 4, April 2008* © *IEEE 2008.*

where

$$K(\boldsymbol{h}, \boldsymbol{g}) = \frac{1}{2} \sum_i \frac{(h_i - g_i)^2}{h_i + g_i}$$

is the χ-square distance between the histograms. A widely adopted alternative is the *histogram intersection kernel*, where

$$K(\boldsymbol{h}, \boldsymbol{g}) = \sum_i \min(h_i, g_i)$$

which will be large if $\boldsymbol{h}$ and $\boldsymbol{g}$ have many boxes of similar weight, and small otherwise. You can see this as an estimate of the number of elements of the ith type that can be matched. Notice that this applies to normalized histograms, which means that if one image has many elements of the ith kind and the other has few, then not only will the ith term in the sum be small, but others must be small, too. The histogram intersection kernel can be evaluated very quickly, with appropriate tricks (Maji *et al.* 2008).

16.1.4 The Spatial Pyramid Kernel

Histograms of visual words are a very powerful representation, as we shall see in Section 16.2. However, they suppress all spatial information, which creates problems in scene recognition. Imagine we are building a kernel to compare scene images. We should like the kernel value to be large for similar scenes, and small for

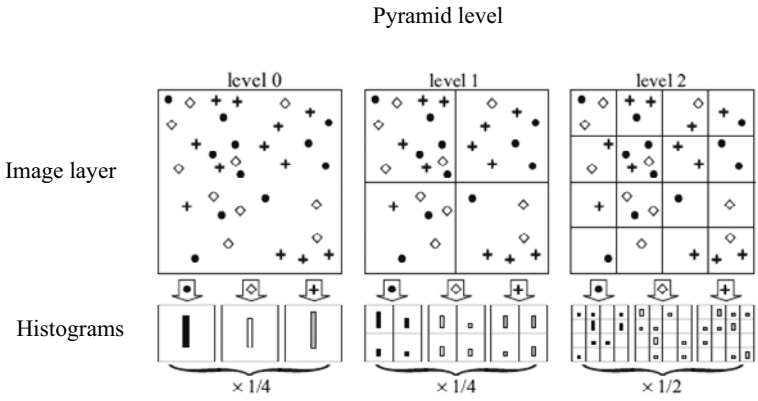

FIGURE 16.8: A simplified example of constructing a spatial pyramid kernel, with three levels. There are three feature types, too (circles, diamonds, and crosses). The image is subdivided into one-, four-, and sixteen-grid boxes. For each level, we compute a histogram of how many features occur in each box for each feature type. We then compare two images by constructing an approximate score of the matches from these histograms. *This figure was originally published as Figure 1 of "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," by S. Lazebnik, C. Schmid, and J. Ponce, Proc. IEEE CVPR 2006, © IEEE 2006.*

different ones. We expect two images of the same scene to have many objects in common, but these objects will move around somewhat. For example, two kitchen pictures should have a high similarity score. Since the ceilings, windows, counters, and floors are at about the same height in different kitchens, we need a score that respects rough spatial structure. The score should not be affected by fine spatial details (for example, moving the toaster from the counter on the left to that on the right).

Lazebnik *et al.* (2006) show how to build an important variant of a histogram of visual words that yields a kernel that has a very effective rough encoding of spatial layout. If you think of each image as a pattern, made up of elements, which could be visual words, then two images should be similar if about the same elements were present in comparable places. In particular, if the elements of one image can be matched to elements of the same kind that lie nearby in another image, then the two images should have high similarity. We cannot compute similarity by matching elements exactly between two images, because there are too many elements and computing an exact matching would be too expensive.

A rough estimate of the number of elements that can be matched is easy to get. If there are $N_{i,1}$ elements of type i in image 1 and $N_{i,2}$ elements of type i in image 2, then $\min(N_{i,1}, N_{i,2})$ elements of type i could match. This is the reasoning underlying the histogram intersection kernel (Section 16.1.3). However, this is a relatively poor estimate of the number of matching elements, because some of these elements may have to match others that are very far away. We can get an improved estimate by breaking each image into four quarters, and applying the same reasoning to each quarter to come up with the score for matching that quarter

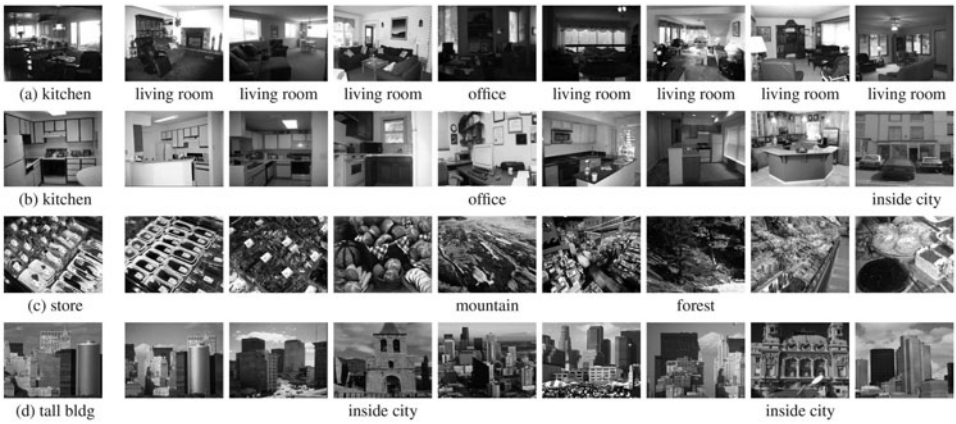

FIGURE 16.9: Measurements of similarity using a spatial pyramid kernel offer natural methods for scene classification, because similar scenes should have about the right features in about the right place. This figure shows results obtained by querying a set of scene images with the query shown on the **left**. On the **right**, images from a test collection ranked by the value of the similarity score, with the most similar image on the left. The responses on the first row are mainly wrong (the name of the room is below the image when the response is wrong), perhaps because the kitchen in the query image has an eccentric layout. Other responses are mostly right. *This figure was originally published as Figure 4 of "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," by S. Lazebnik, C. Schmid, and J. Ponce, Proc. IEEE CVPR 2006,* © *IEEE 2006.*

to the corresponding quarter of the other image. We now have five estimates (one for the whole image, and one for each quarter) and must combine them. We do so using weights that depend on the size of the image partitions for which the estimates were computed. We could subdivide the quarters again to create even smaller boxes and weight the local estimates appropriately, but boxes that are too small are not terribly informative.

We can now give a formal expression for the similarity score between two images, which is the kernel value. For simplicity, assume the patterns we work with are all the same size. We wish to compare $\mathcal{I}$ and $\mathcal{J}$. To get an estimate of the number of features that would match, we break each image into a grid of squares. We will use several different grids, indexed by l (Figure 16.8). We write $H^l_{\mathcal{I},t}(i)$ for the number of features of type t in the ith box in grid l on image $\mathcal{I}$. We assume that elements in a particular box in a particular grid over $\mathcal{I}$ can match only to elements in the corresponding box and grid over $\mathcal{J}$. We also assume that all elements that can be matched within a box, are matched. This means that the number of elements of type t in box i in grid l that match is

$$\min(H^l_{\mathcal{I},t}(i), H^l_{\mathcal{J},t}(i)),$$

and the similarity between $\mathcal{I}$ and $\mathcal{J}$, as measured at grid level l, is

$$\sum_{i \in \text{grid boxes}} \min(H^l_{\mathcal{I},t}(i), H^l_{\mathcal{J},t}(i)).$$

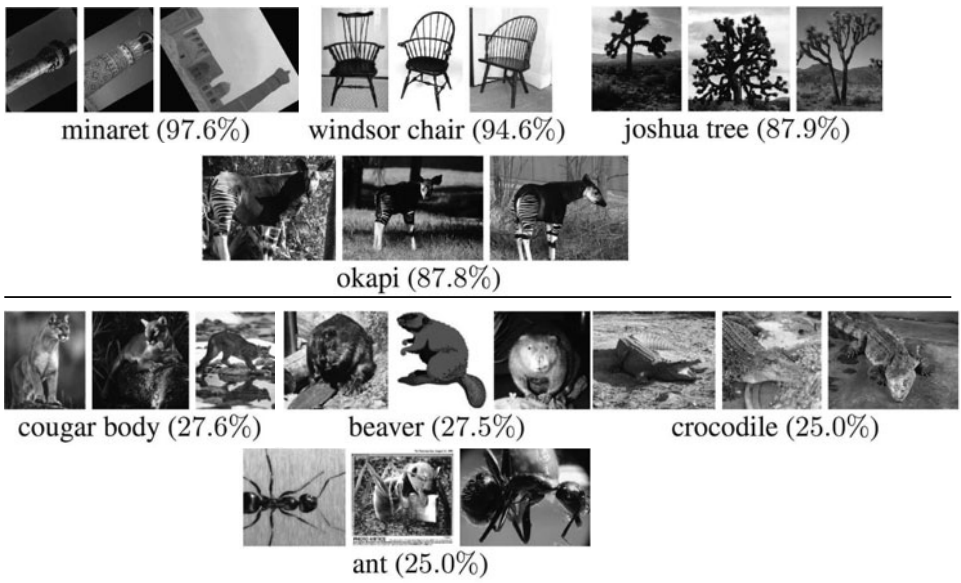

FIGURE 16.10: The spatial pyramid kernel is capable of complex image classification tasks. Here we show some examples of categories from the Caltech 101 collection on which the method does well (**top row**) and poorly (**bottom row**). The number is the percentage of images of that class classified correctly. Caltech 101 is a set of images of 101 categories of objects; one must classify test images into this set of categories (Section 16.3.2). *This figure was originally published as Figure 5 of "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," by S. Lazebnik, C. Schmid, and J. Ponce, Proc. IEEE CVPR 2006,* © *IEEE 2006.*

Each grid gives us an estimate of how well the features match, but generally we would like to place more weight on matches in fine grids and less weight on matches in coarse grids. We can do this by weighting matches by the inverse of the cell width at each level; write this weight as w_l. We will assume that matches between features of different types have the same weight, and obtain a total similarity score

$$\sum_{t \in \text{feature types}} \sum_{l \in \text{levels}} \sum_{i \in \text{grid boxes}} w_l \min(H^l_{\mathcal{I},t}(i), H^l_{\mathcal{I},t}(i)).$$

The resulting similarity estimates can be used either to rank image similarity (as in Figure 16.9), or as a kernel for a kernel-based classifier.

The spatial pyramid kernel does very well at classifying images by scene, and can outperform histogram intersection kernels on standard image classification tasks, even on datasets where the background on which objects appear varies very widely (Lazebnik *et al.* 2006). It can work well with very rich pools of features. In the work of Lazebnik *et al.* (2006), visual words are not constructed at interest points alone, but on a grid across the whole image; this means that there is a much richer—and much larger—set of visual words available to represent the image. It is notable that spatial pyramid kernels seem to represent relatively isolated objects or

natural scenes well, but have trouble with textureless objects or objects that blend well into their backgrounds (Figure 16.10).

16.1.5 Dimension Reduction with Principal Components

Our constructions tend to produce high-dimensional feature vectors. This will tend to make it more difficult to estimate a classifier accurately, because it will tend to increase the variance of the estimate. It can be useful to reduce the dimension of a feature vector. We can do so by projecting onto a low-dimensional basis. One way to choose this basis is to insist that the new set of features should capture as much of the old set's variance as possible. As an extreme example, if the value of one feature can be predicted precisely from the value of the others, it is clearly redundant and can be dropped. By this argument, if we are going to drop a feature, the best one to drop is the one whose value is most accurately predicted by the others, that is, features that have relatively little variance.

In *principal component analysis*, the new features are linear functions of the old features. We take a set of data points and construct a lower dimensional linear subspace that best explains the variation of these data points from their mean. This method (also known as the Karhunen–Loéve transform) is a classical technique from statistical pattern recognition (see, for example Duda and Hart (1973), Oja (1983), or Fukunaga (1990)).

Assume we have a set of n feature vectors $\boldsymbol{x}_i$ $(i = 1, \ldots, n)$ in $\mathbb{R}^d$. The mean of this set of feature vectors is $\boldsymbol{\mu}$ (you should think of the mean as the center of gravity in this case), and their covariance is Σ (you can think of the variance as a matrix of second moments). We use the mean as an origin and study the offsets from the mean $(\boldsymbol{x}_i - \boldsymbol{\mu})$.

Our features are linear combinations of the original features; this means it is natural to consider the projection of these offsets onto various different directions. A unit vector $\boldsymbol{v}$ represents a direction in the original feature space; we can interpret this direction as a new feature $v(\boldsymbol{x})$. The value of u on the ith data point is given by $v(\boldsymbol{x}_i) = \boldsymbol{v}^T(\boldsymbol{x}_i - \boldsymbol{\mu})$. A good feature captures as much of the variance of the original dataset as possible. Notice that v has zero mean; then the variance of v is

$$\begin{aligned}
\text{var}(v) &= \frac{1}{n-1}\sum_{i=1}^{n} v(\boldsymbol{x}_i)v(\boldsymbol{x}_i)^T \\
&= \frac{1}{n}\sum_{i=1}^{n-1} \boldsymbol{v}^T(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{v}^T(\boldsymbol{x}_i - \boldsymbol{\mu}))^T \\
&= \boldsymbol{v}^T \left\{ \sum_{i=1}^{n-1} (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T \right\} \boldsymbol{v} \\
&= \boldsymbol{v}^T \Sigma \boldsymbol{v}.
\end{aligned}$$

Now we should like to maximize $\boldsymbol{v}^T\Sigma\boldsymbol{v}$ subject to the constraint that $\boldsymbol{v}^T\boldsymbol{v} = 1$. This is an eigenvalue problem; the eigenvector of Σ corresponding to the largest eigenvalue is the solution. Now if we were to project the data onto a space *perpendicular* to this eigenvector, we would obtain a collection of $d - 1$ dimensional vectors. The highest variance feature for this collection would be the eigenvector

of Σ with second largest eigenvalue, and so on.

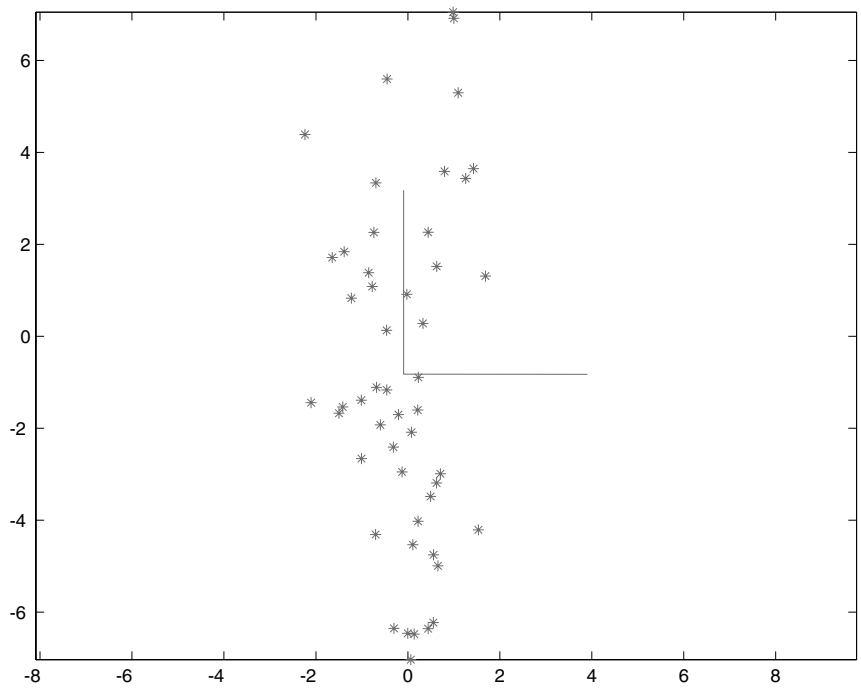

FIGURE 16.11: A dataset that is well represented by a principal component analysis. The axes represent the directions obtained using PCA; the vertical axis is the first principal component, and is the direction in which the variance is highest.

This means that the eigenvectors of Σ—which we write as $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d$, where the order is given by the size of the eigenvalue and $\boldsymbol{v}_1$ has the largest eigenvalue—give a set of features with the following properties:

- They are independent (because the eigenvectors are orthogonal).
- Projection onto the basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ gives the k-dimensional set of linear features that preserves the most variance.

You should notice that, depending on the data source, principal components can give a good or a bad representation of a data set (see Figures 16.11, 16.12, and 16.13).

16.1.6 Dimension Reduction with Canonical Variates

Principal component analysis yields a set of linear features of a particular dimension that best represents the variance in a high-dimensional dataset. There is no guarantee that this set of features is good for *classification*. For example, Figure 16.13 shows a dataset where the first principal component would yield a bad classifier, and the second principal component would yield quite a good one, despite not capturing the variance of the dataset.

Assume we have a set of n feature vectors $\boldsymbol{x}_i$ $(i = 1, \ldots, n)$ in $\mathbb{R}^d$. Write

$$\boldsymbol{\mu} = \frac{1}{n} \sum_i \boldsymbol{x}_i$$

$$\Sigma = \frac{1}{n-1} \sum_i (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$$

The unit eigenvectors of Σ—which we write as $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d$, where the order is given by the size of the eigenvalue and $\boldsymbol{v}_1$ has the largest eigenvalue—give a set of features with the following properties:

- They are independent.
- Projection onto the basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ gives the k-dimensional set of linear features that preserves the most variance.

Algorithm 16.1: Principal Components Analysis

Linear features that emphasize the distinction between classes are known as *canonical variates*. To construct canonical variates, assume that we have a set of data items $\boldsymbol{x}_i$, for $i \in \{1, \ldots, n\}$. We assume that there are p features (i.e., that the $\boldsymbol{x}_i$ are p-dimensional vectors). We have g different classes, and the jth class has mean $\boldsymbol{\mu}_j$. Write $\overline{\boldsymbol{\mu}}$ for the mean of the class means, that is,

$$\overline{\boldsymbol{\mu}} = \frac{1}{g} \sum_{j=1}^{g} \boldsymbol{\mu}_j.$$

Write

$$\mathcal{B} = \frac{1}{g-1} \sum_{j=1}^{g} (\boldsymbol{\mu}_j - \overline{\boldsymbol{\mu}})(\boldsymbol{\mu}_j - \overline{\boldsymbol{\mu}})^T.$$

Note that $\mathcal{B}$ gives the variance of the class means. In the simplest case, we assume that each class has the same covariance Σ, and that this has full rank. We would like to obtain a set of axes where the clusters of data points belonging to a particular class group together tightly, whereas the distinct classes are widely separated. This involves finding a set of features that maximizes the ratio of the separation (variance) between the class means to the variance within each class. The separation between the class means is typically referred to as the *between-class variance*, and the variance within a class is typically referred to as the *within-class variance*.

Now we are interested in linear functions of the features, so we concentrate on

$$v(\boldsymbol{x}) = \boldsymbol{v}^T \boldsymbol{x}.$$

We should like to maximize the ratio of the between-class variances to the within-class variances for $\boldsymbol{v}_1$.

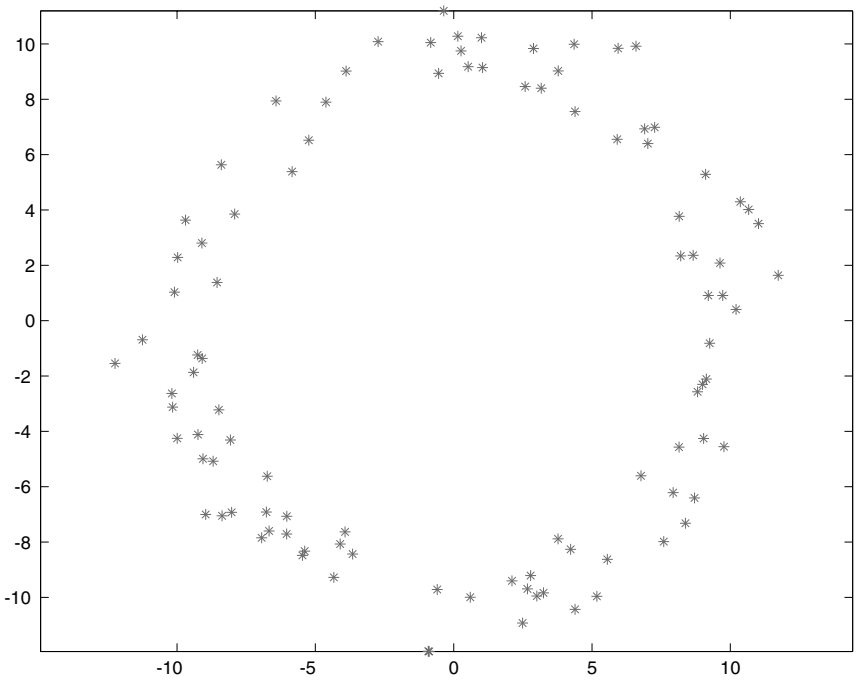

FIGURE 16.12: Not every dataset is well represented by PCA. The principal components of this dataset are relatively unstable, because the variance in each direction is the same for the source. This means that we may well report significantly different principal components for different datasets from this source. This is a secondary issue; the main difficulty is that projecting the dataset onto some axis suppresses the main feature, its circular structure.

Using the same argument as for principal components, we can achieve this by choosing $\boldsymbol{v}$ to maximize

$$\frac{\boldsymbol{v}_1^T \mathcal{B} \boldsymbol{v}_1}{\boldsymbol{v}_1^T \Sigma \boldsymbol{v}_1}.$$

This problem is the same as maximizing $\boldsymbol{v}_1^T \mathcal{B} \boldsymbol{v}_1$ subject to the constraint that $\boldsymbol{v}_1^T \Sigma \boldsymbol{v}_1 = 1$. In turn, a solution has the property that

$$\mathcal{B}\boldsymbol{v}_1 + \lambda \Sigma \boldsymbol{v}_1 = 0$$

for some constant λ. This is known as a *generalized eigenvalue problem*; if Σ has full rank, one solution is to find the eigenvector of $\Sigma^{-1}\mathcal{B}$ with largest eigenvalue. It is usually better to use specialized routines within the relevant numerical software environment, which can deal with the case Σ does not have full rank.

Now for each $\boldsymbol{v}_l$, for $2 \leq l \leq p$, we would like to find features that extremize the criterion and are independent of the previous $\boldsymbol{v}_l$. These are provided by the other eigenvectors of $\Sigma^{-1}\mathcal{B}$. The eigenvalues give the variance along the features (which are independent). By choosing the $m < p$ eigenvectors with the largest eigenvalues,

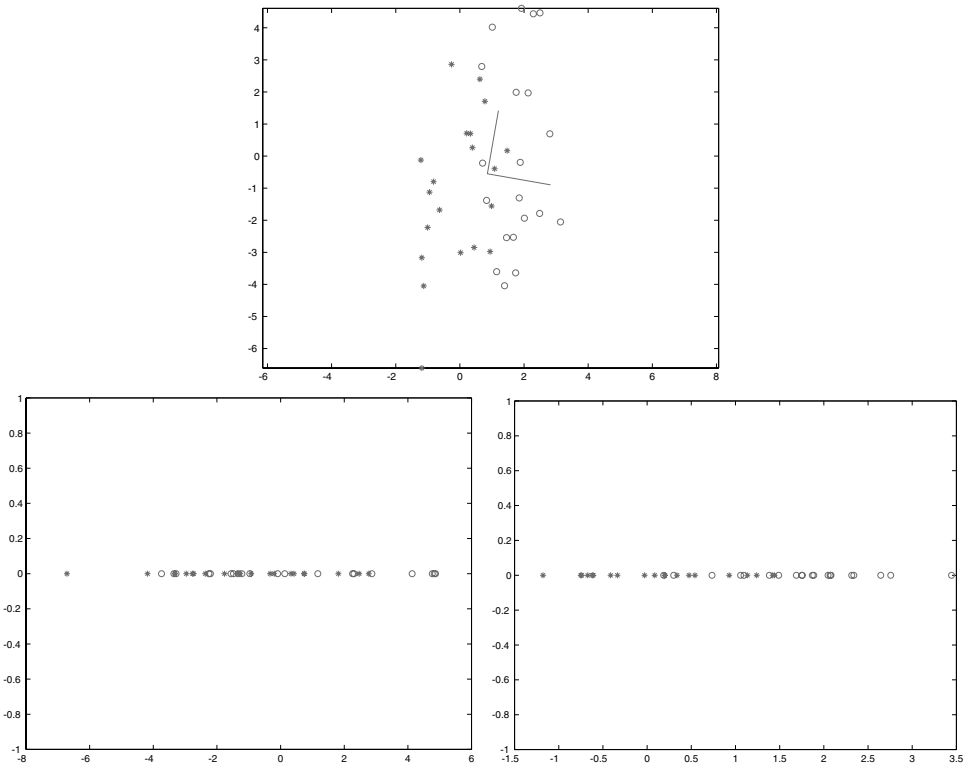

FIGURE 16.13: Principal component analysis doesn't take into account the fact that there may be more than one class of item in a dataset. This can lead to significant problems. For a classifier, we would like to obtain a set of features that reduces the number of features and makes the difference between classes most obvious. For the dataset on the **top**, one class is indicated by circles and the other by stars. PCA would suggest projection onto a vertical axis, which captures the variance in the dataset but cannot be used to discriminate between the classes, as we can see from the axes obtained by PCA, which are overlaid on the dataset. The **bottom row** shows the projections onto those axes. On the **bottom left**, we show the projection onto the first principal component, which has higher variance but separates the classes poorly, and on the **bottom right**, we show the projection onto the second principal component, which has significantly lower variance (look at the axes) and gives better separation.

we obtain a set of features that reduces the dimension of the feature space while best preserving the separation between classes. This doesn't guarantee the best error rate for a classifier on a reduced number of features, but it offers a good place to start by reducing the number of features while respecting the category structure. Details and examples appear in McLachlan and Krishnan (1996) and in Ripley (1996).

If the classes don't have the same covariance, it is still possible to construct canonical variates. In this case, we estimate a Σ as the covariance of all the offsets of each data item *from its own class mean* and proceed as before. Again, this is an

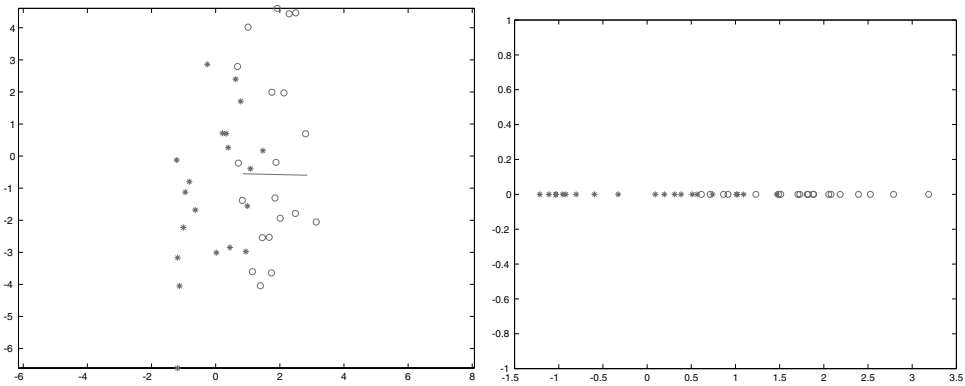

FIGURE 16.14: Canonical variates use the class of each data item as well as the features in estimating a good set of linear features. In particular, the approach constructs axes that separate different classes as well as possible. The dataset used in Figure 16.13 is shown on the **left**, with the axis given by the first canonical variate overlaid. On the **bottom right**, we show the projection onto that axis, where the classes are rather well separated.

approach without a guarantee of optimality, but one that can work quite well in practice.

16.1.7 Example Application: Identifying Explicit Images

All published explicit image classifiers start by detecting skin. Skin detection is an important problem in its own right. Skin detectors are very useful system components, because they can be used to focus searches. If one is searching for explicit images, then skin will likely be involved. But if one is looking for faces, skin is a good place to start, too (Section 17.1.1). If one is trying to interpret sign language, hands are important, and hands usually show skin (for example, systems in Buehler *et al.* (2009), Buehler *et al.* (2008), Farhadi and Forsyth (2006), Farhadi *et al.* (2007), and Bowden *et al.* (2004)). Skin detection is also a natural example of a very general recipe for detection: apply a classifier at each location likely to be of interest. For example, we can detect skin by (a) building a skin classifier, then (b) applying it to each pixel in an image independently. Formally, this involves (the clearly false) assumption that pixels are independent; in practice, pretending that pixels are independent is quite satisfactory for skin detection.

Skin is quite easy to identify by color. This does not depend on how dark the skin is. The color of human skin is caused by a combination of light reflected from the surface of the skin (which will have a white hue), from blood under the surface (which gives skin a red tinge), and from melanin under the surface (which absorbs light and darkens the apparent color). The intensity of light reflected from skin can vary quite a lot, because the intensity of illumination can change, specular reflection from oils and grease on the surface can be very bright, and skin with more melanin in it absorbs more light (and so looks darker under fixed illumination). But the hue and saturation of skin do not vary much; the hue will tend to be in the red-orange

Assume that we have a set of data items of g different classes. There are n_k items in each class, and a data item from the kth class is $\boldsymbol{x}_{k,i}$, for $i \in \{1, \ldots, n_k\}$. The jth class has mean $\boldsymbol{\mu}_j$. We assume that there are p features (i.e., that the $\boldsymbol{x}_i$ are p-dimensional vectors).
Write $\overline{\boldsymbol{\mu}}$ for the mean of the class means, that is,

$$\overline{\boldsymbol{\mu}} = \frac{1}{g} \sum_{j=1}^{g} \boldsymbol{\mu}_j,$$

Write

$$\mathcal{B} = \frac{1}{g-1} \sum_{j=1}^{g} (\boldsymbol{\mu}_j - \overline{\boldsymbol{\mu}})(\boldsymbol{\mu}_j - \overline{\boldsymbol{\mu}})^T.$$

Assume that each class has the same covariance Σ, which is either known or estimated as

$$\Sigma = \frac{1}{N-1} \sum_{c=1}^{g} \left\{ \sum_{i=1}^{n_c} (\boldsymbol{x}_{c,i} - \boldsymbol{\mu}_c)(\boldsymbol{x}_{c,i} - \boldsymbol{\mu}_c)^T \right\}.$$

The unit eigenvectors of $\Sigma^{-1}\mathcal{B}$, which we write as $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d$, where the order is given by the size of the eigenvalue and $\boldsymbol{v}_1$ has the largest eigenvalue, give a set of features with the following property:

- Projection onto the basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ gives the k-dimensional set of linear features that best separates the class means.

Algorithm 16.2: Canonical Variates

range (blue or green skin look very unnatural), and the color will not be strongly saturated. This means that a large number of image pixels can be quite reliably rejected as non-skin pixels by inspecting their color alone.

The simplest learned skin detector, which is very effective, uses a class-conditional histogram classifier (as in Section 15.2.2), and is due to Jones and Rehg (Jones and Rehg 2002). Each pixel is classified as skin or not skin based on

Create an output image $\mathcal{O}$, and fill it with zeros.

For each pixel I_{ij} in the input (binary) image
    If I_{ij} is 1 and all neighbors are 1
        $O_{ij} = 1$
    End
End

Algorithm 16.3: Dilation

FIGURE 16.15: The figure shows a variety of images together with the output of the skin detector of Jones and Rehg applied to the image. Pixels marked black are skin pixels and white are background. Notice that this process is relatively effective and could certainly be used to focus attention on, say, faces and hands. *This figure was originally published as Figure 6 of "Statistical color models with application to skin detection," by M.J. Jones and J. Rehg, Proc. IEEE CVPR, 1999* © *IEEE, 1999.*

its R, G, and B coordinates. The quantization of these color values seems not to affect the accuracy of the detector all that much; this should be read as experimental evidence that the range of skin colors is fairly tightly clustered. One important source of errors are specular reflections on skin, which tend to be bright and white. If the skin detector marks such pixels as positives, huge areas of some images will become false positives; but if it marks them as negatives, most faces will have missing chunks of skin around the nose and forehead. Pixels neighboring skin pixels are probably themselves skin pixels, and the same applies to non-skin pixels. We can exploit this with a simple trick. Regard the output of the skin detector as a binary image, with skin pixels labeled 1. We apply several steps of *erosion*, a binary image operator (Algorithm 16.3). This will tend to remove isolated skin pixels, and make holes in skin bigger. We then apply several steps of *dilation*, another binary image operator (Algorithm 16.4). This will tend to fill holes in the skin mask. More com-

Create an output image $\mathcal{O}$, and fill it with zeros.

For each pixel I_{ij} in the input (binary) image
    If any of its neighbors are 1
        $O_{ij} = 1$
    End
End

Algorithm 16.4: Erosion

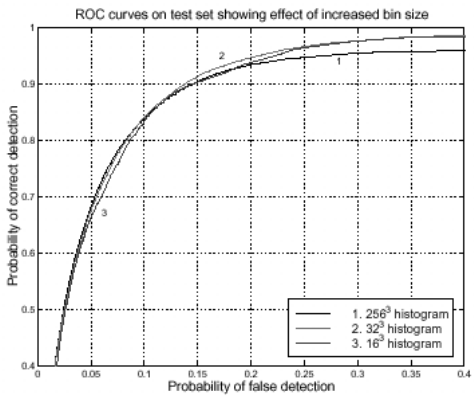

FIGURE 16.16: The receiver operating curve for the skin detector of Jones and Rehg. This plots the detection rate against the false-negative rate for a variety of values of the parameter θ. A perfect classifier has an ROC that, on these axes, is a horizontal line at 100% detection. Notice that the ROC varies slightly with the number of boxes in the histogram. *This figure was originally published as Figure 7 of "Statistical color models with application to skin detection," by M.J. Jones and J. Rehg, Proc. IEEE CVPR, 1999* © *IEEE, 1999.*

plex skin detectors rely on the fact that skin has relatively little texture, so quite simple texture features can be discriminative (Forsyth and Fleck 1999). This skin detector suggests a recipe for detecting other things: search image windows using a detector. This recipe is immensely useful, and is explored in detail in Section 17.1.

Building an explicit image detector is now quite straightforward. We detect skin, then compute features from the skin regions, then pass these features to a classifier. All these methods work tolerably well on experimental datasets. Bosson *et al.* (2002) started the tradition of building simple region layout features from the skin regions. They use, among other features, number of skin regions, fractional area of the largest skin region, fraction of skin that is accounted for by a face (see Section 17.1.1 for face detection), and classify with a support vector machine. Forsyth and Fleck (1999) find groups of skin regions that appear to be body

segments (arms, legs, etc.), then decide a naked person is present when a large enough group is found. Deselaers *et al.* (2008) use a histogram of visual words, followed by either an SVM or logistic regression. We know of no system that uses a spatial pyramid kernel for this problem, but expect that that would work rather well. Easy experiments—think of a phrase likely to produce alarming pictures, then search with the search filter turned on—with commercial image search programs suggest that these have low-false positive rates, though the false-negative rate is extremely hard to assess (and may not be known even to the manufacturers). Commercial methods may use text on pages and link topology information as well as image features to classify pictures, too.

16.1.8 Example Application: Classifying Materials

Leung and Malik (2001) show that a texton representation (see Section 6.2, and keep in mind the similarities between textons and visual words) can be used to classify materials. Assuming simple textures (where there is no relief, and so no illumination effect), they represent image patches with a vector of 48 filter responses evaluated at the center of the patch. Their textons are constructed by vector quantizing these vectors, using k-means. A single patch of texture is then represented by (a) computing the texton for each pixel in the patch, then (b) computing the overall histogram of these textons. Textures are classified by nearest neighbors, using the ξ-squared distance between histograms. Now assume that the texture is really a material. In this case, we need to have multiple images of the same patch to train, because the appearance of the patch changes widely. In their experiment, the test sample consisted of multiple images as well. The texton labels could be unreliable, because some accident of lighting might have caused a patch to look like some other patch. However, because we can record how textons change under the lighting changes, we have an estimate of what the right labels could be for each texton. To compare a test sample with a training example, we search the different available labellings of textons for the test sample, computing the ξ-squared distance between the resulting histograms and the training example's histogram. The distances will tend to be big when the two do not match, even though we can relabel textons to bring them closer. A variety of alternative constructions are available, varying mainly in how the textons are built. Varma and Zisserman (2005) show improved classification using a small set of rotationally invariant filters; in a later paper, they show even better classification results produced by vector quantizing small image patches (Varma and Zisserman 2009).

All this work uses images of isolated patches of material. When one is presented with an image of an object, Liu *et al.* (2010) show that determining its material remains very hard. They collected a new dataset of images of objects made of a single predominant material, then applied visual words and other methods to classify it. As Figure 16.17 shows, this problem remains extremely difficult.

16.1.9 Example Application: Classifying Scenes

The original scene classification method, due to Oliva and Torralba (2001), used k-nearest neighbors to classify scenes using GIST features. This classifies single images into one of eight classes. Torralba *et al.* (2003) then showed that GIST

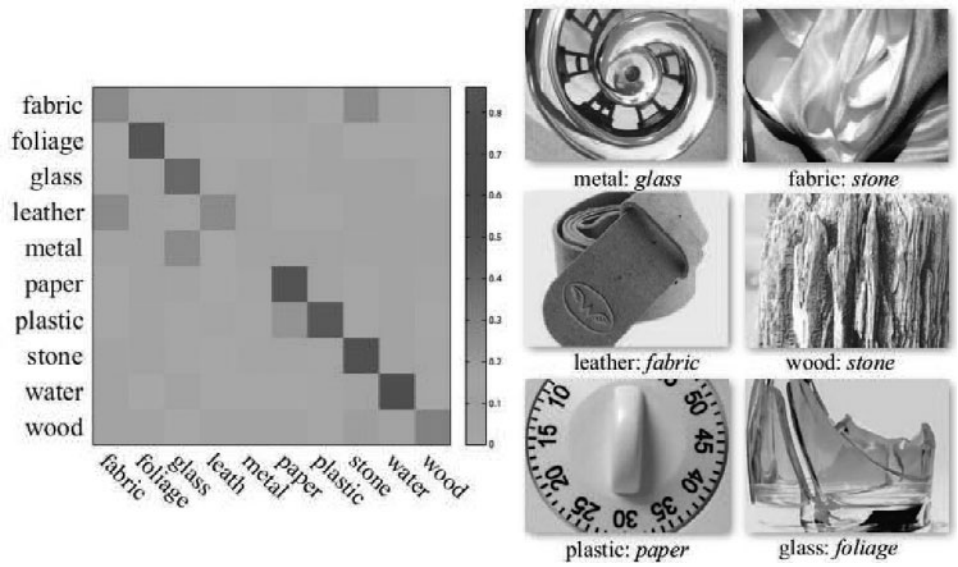

FIGURE 16.17: Liu *et al.* (2010) prepared a material classification dataset from flickr images, and used a combination of SIFT features and novel features to classify the materials. This is a difficult task, as the class confusion matrix on the **left** shows; for example, it is quite easy to mix up metal with most other materials, particularly glass. On the **right**, examples of misclassified images (the italic label is the incorrect prediction). *This figure was originally published as Figure 12 of "Exploring Features in a Bayesian Framework for Material Recognition," by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz Proc. CVPR 2010, 2010* © *IEEE, 2010.*

features could be used to identify place, that is, where an image was taken, chosen from a fixed vocabulary of known places. For images taken in places that are not in the vocabulary, their system could describe the type of place, for example, "kitchen" or "lobby" (Figure 16.3). Their system assumed the camera was attached to a moving observer, but did not explicitly use motion cues; instead, their system uses a form of conditional random field so that prior probabilities of state transitions can affect the classification of an image (Torralba *et al.* 2003). A scene is a context in which some objects are likely to occur (for example, "toasters" are more common in "kitchens" than "outdoors"). Furthermore, some objects are more likely to occur in some locations in a scene ("toasters" are likely on "tables," but not on "floors"), so we expect that knowing the scene identity offers some cues to likely object location. This turns out to be the case, as Torralba *et al* demonstrate (Figure 16.4). As we have seen above, spatial pyramid kernels do well at scene classification, too.

Xiao *et al.* (2010) report scene classification results on a recent, very large, dataset (SUN; 397 categories with at least 100 images each; see Section 16.3.2). They compare a variety of methods, the best getting a recognition rate of approximately 38% (i.e., approximately 38% of classification attempts on test data are correct). As Figures 16.2 and 16.18 suggest, we might not expect a classification performance of 100%.

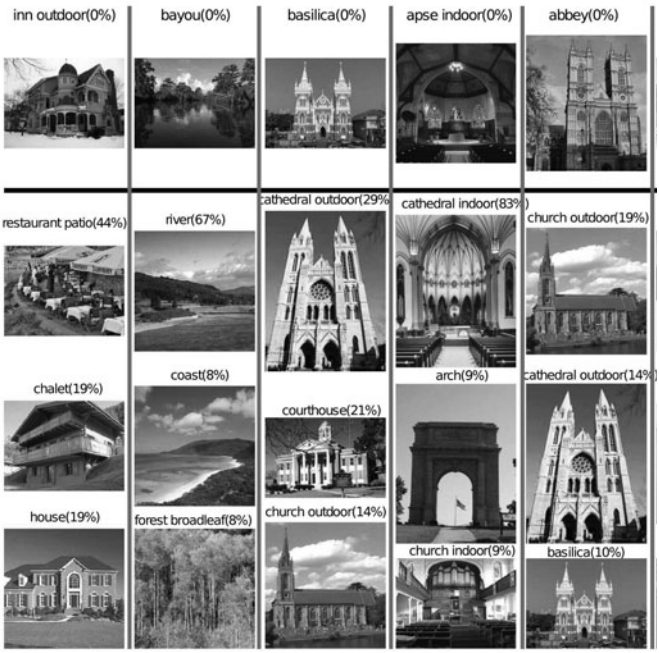

FIGURE 16.18: Not all scene categories are easily distinguished by humans. These are examples from the SUN dataset (Xiao *et al.* 2010). On the **top** of each column is an example of a difficult category; **below** are examples from three categories that are commonly confused with it by people. Such confusions might result from difficulties in knowing the category boundaries (which aren't canonical), or the terms (or categories) are unfamiliar, or because the images are ambiguous. *This figure was originally published as Figure 3 of "SUN database: Large-scale Scene Recognition from Abbey to Zoo," by J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba, Proc. IEEE CVPR 2010, © IEEE, 2010.*

16.2 CLASSIFYING IMAGES OF SINGLE OBJECTS

Many interesting images contain essentially a single object on a simple background. Some images, such as catalogue pictures, are composed this way. Others just happen to be this way. Images like this are important in image search, because searchers typically compose quite simple queries (e.g., "elephant" rather than "long shot showing mopani bush near a waterhole with an elephant in the bottom left and baboons frolicking") and so might not object to quite simple responses. Another reason that they're important is that it might be easier to learn models of object category from such images, rather than from general images. A core challenge in computer vision is to learn to classify such images.

A bewildering variety of features and classifiers has been tried on this problem; Section 16.2.1 makes some general remarks about approaches that seem successful. The classification process is rather naturally linked to image search, and so image search metrics are the usual way to evaluate classifiers (Section 16.2.2). Large-scale experimental work in image classification is difficult (one must collect and label

images; and one must train and evaluate methods on very large datasets), and as a result there are unavoidable flaws in all experimental paradigms at present. However, trends suggest significant improvements in the understanding of image classification over the last decade. There are two general threads in current work. One can try to increase the accuracy of methods using a fixed set of classes (Section 16.2.3), and so gain some insight into feature constructions. Alternatively, one might try to handle very large numbers of classes (Section 16.2.4), and so gain some insight into what is discriminative in general.

16.2.1 Image Classification Strategies

The general strategy is to compute features, and then present feature vectors to a multi-class classifier. A very great variety of methods can be produced using this strategy, depending on what features and what classifier one uses. However, there are some general statements one can make. The features we described earlier predominate (which is why we described them). Methods typically use variants of HOG and SIFT features, combined with color features. Methods commonly use dictionaries of visual words, though there is great variation in the precise way these dictionaries are built. Spatial pyramid and pyramid match kernels seem to give very strong performance in representing images. A wide variety of classifiers are then applied to the resulting features; different reasonable choices of classifier give slightly different results, but no single classifier appears to have an overwhelming advantage.

Most research in this area is experimental in nature, and building datasets is a major topic. Fortunately, datasets are freely shared, and there is much competition to get the best performance on a particular dataset. Furthermore, much feature and classifier code is also freely shared. In turn, this means that it is usually relatively straightforward to try and reproduce cutting-edge experiments. Section 16.3 gives a detailed survey of datasets and code available as of the time of writing.

There are so many methods, each with slight advantages, that it is difficult to give a crisp statement of best practice. The first thing we would do when presented with a novel image classification problem would be to compute visual words for feature locations on an image grid. These visual words would be vector quantized with a large number of types (10^4 or 10^5, if enough data is available). We would then represent images with a histogram of the visual words and classify them using a histogram intersection kernel. If we were unhappy with the results, we would first vary the number of types of visual word, then apply a spatial pyramid kernel. After that, we would start searching through the different packages for feature computation described in Section 16.3.1, and perhaps search different types of classifier.

16.2.2 Evaluating Image Classification Systems

Image retrieval is related to image classification. In image retrieval, one queries a very large collection of images with a query—which could be keywords, or an image—and wishes to get matching images back (Chapter 21 surveys this area). If the query is a set of keywords and we expect keyword matches, then there must be some form of image classification engine operating in the background to attach

keywords to images. For this reason, it is quite usual to use metrics from image retrieval to evaluate image classification methods.

Information retrieval systems take a query, and produce a response from a collection of data items. The most important case for our purposes is a system that takes a set of keywords and produces a set of images taken from a collection. These images are supposed to be relevant to the keyword query. Typically, two terms are used to describe the performance of information retrieval systems. The percentage of relevant items that are actually recovered is known as the *recall*. The percentage of recovered items that are actually relevant is known as the *precision*. It is natural to use these measures to evaluate an image classification system, because this system is attaching a label — which is rather like a keyword—to a set of test images.

It is tempting to believe that good systems should have high recall and high precision, but this is not the case. Instead, what is required for a system to be good depends on the application, as the following examples illustrate.

Patent searches: Patents can be invalidated by finding "prior art" (material that predates the patent and contains similar ideas). A lot of money can depend on the result of a prior art search. This means that it is usually much cheaper to pay someone to wade through irrelevant material than it is to miss relevant material, so very high recall is essential, even at the cost of low precision.

Web and email filtering: US companies worry that internal email containing sexually explicit pictures might create legal or public relations problems. One could have a program that searched email traffic for problem pictures and warned a manager if it found anything. Low recall is fine in an application like this; even if the program has only 10% recall, it will still be difficult to get more than a small number of pictures past it. High precision is very important, because people tend to ignore systems that generate large numbers of false alarms.

Looking for an illustration: There are various services that provide stock photographs or video footage to news organizations. These collections tend to have many photographs of celebrities; one would expect a good stock photo service to have many thousands of photographs of Nelson Mandela, for example. This means that a high recall search can be a serious nuisance, as no picture editor really wants to wade through thousands of pictures. Typically, staff at stock photo organizations use their expertise and interviews with customers to provide only a very small subset of relevant pictures.

There are a variety of ways of summarizing recall and precision data to make it more informative. *F-measures* are weighted harmonic means of precision and recall. Write P for precision and R for recall. The *F_1-measure* weights precision and recall evenly, and is given by

$$F_1 = 2\frac{PR}{P+R}.$$

The *F_β-measure* weights recall β times as strongly as recall, and is given by

$$F_\beta = (1+\beta^2)\frac{PR}{\beta^2 P + R}.$$

Usually, it is possible to adjust the number of items that a system returns in re-

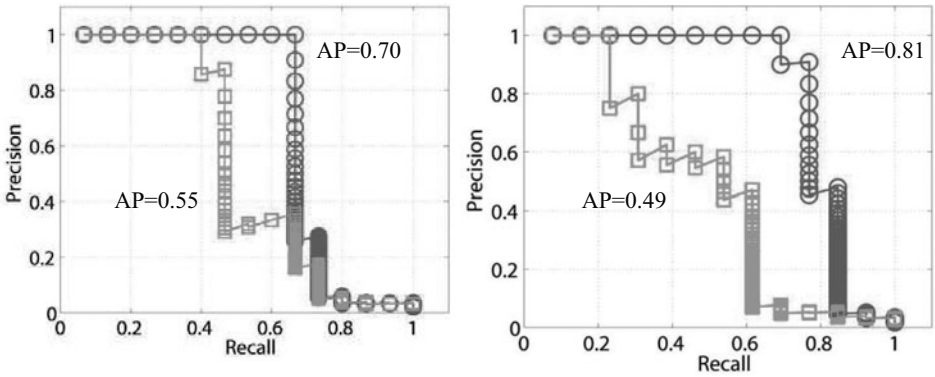

FIGURE 16.19: Plots of precision as a function of recall for six object queries. Notice how precision generally declines as recall goes up (the occasional jumps have to do with finding a small group of relevant images; such jumps would become arbitrarily narrow and disappear in the limit of an arbitrarily large dataset). Each query is made using the system sketched in Figure 16.5. Each graph shows a different query, for two different configurations of that system. On top of each graph, we have indicated the average precision for each of the configurations. Notice how the average precision is larger for systems where the precision is higher for each recall value. *This figure was originally published as Figure 9 of J. Sivic and A. Zisserman "Efficient Visual Search for Objects in Videos," Proc. IEEE, Vol. 96, No. 4, April 2008* © *IEEE 2008.*

sponse to a query. As this pool gets bigger, the recall will go up (because we are recovering more items) and the precision will go down. This means we can plot precision against recall for a given query. Such a plot gives a fairly detailed picture of the system's behavior, and particular profiles are important for particular applications (Figure 16.19). For example, for web search applications, we would typically like high precision at low recall, and are not concerned about how quickly the precision dies off as the recall increases. This is because people typically don't look at more than one or two pages of query results before rephrasing their request. For patent search applications, on the other hand, the faster the precision dies off, the more stuff we might have to look at, so the rate at which precision falls off becomes important.

An important way to summarize a precision-recall curve is the *average precision*, which is computed for a ranking of the entire collection. This statistic averages the precision at which each new relevant document appears as we move down the list. Write rel(r) for the binary function that is one when the rth document is relevant, and otherwise zero; $P(r)$ for the precision of the first r documents in the ranked list; N for the number of documents in the collection; and N_r for the total number of relevant documents. Then, average precision is given by

$$A = \frac{1}{N_r} \sum_{r=1}^{N} (P(r)\text{rel}(r))$$

Notice that average precision is highest (100%) when the top N_r documents are the relevant documents. Averaging over all the relevant documents means the

statistic incorporates information about recall; if we were to average over the top 10 relevant documents, say, we would not know how poor the precision was for the lowest-ranked relevant document. The difficulty for vision applications is that many relevant documents will tend to be ranked low, and so average precision statistics tend to be low for image searches. This doesn't mean image searches are useless, however.

All of these statistics are computed for a single query, but most systems are used for multiple queries. Each statistic can be averaged over multiple queries. The choice of queries to incorporate in the average usually comes from application logic. *Mean average precision*, the average precision averaged over a set of queries, is widely used in object recognition circles. In this case, the set of possible queries is typically relatively small, and the average is taken over all queries.

16.2.3 Fixed Sets of Classes

The Pascal Challenge is a series of challenge problems set to the vision community by members of the Pascal network. From 2005–2010, the Pascal Challenge has included image classification problems. From 2007–2010, these problems involved 20 standard classes (including aeroplane, bicycle, car and person). Examples of these images can be found at `http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2010/examples/index.html`. Table 16.1 shows average precisions obtained by the best method *per class* (meaning that the method that did best on aeroplanes might not be the same as the method that did best at bicycles) from 2007–2010. Note the tendency for results to improve, though there is by no means monotonic improvement. For these datasets, the question of selection bias does not arise, as a new dataset is published each year. As a result, it is likely that improvements probably do reflect improved features or improved classification methodologies. However, it is still difficult to conclude that methods that do well on this challenge are good, because the methods might be adapted to the set of categories. There are many methods that participate in this competition, and differences between methods are often a matter of quite fine detail. The main website (`http://pascallin.ecs.soton.ac.uk/challenges/VOC/`) is a rich mine of information, and has a telegraphic description of each method as well as some pointers to feature software.

Error rates are still fairly high, even with relatively small datasets. Some of this is most likely caused by problematic object labels. These result from the occasional use of obscure terms (for example, few people know the difference between a "yawl" and a "ketch" or what either is, but each is represented in the Caltech 101 dataset). Another difficulty is a natural and genuine confusion about what term applies to what instance. Another important source of error is that current methods cannot accurately estimate the spatial support of the object (respectively, background), and so image representations conflate the two somewhat. This is not necessarily harmful—for example, if objects are strongly correlated with their backgrounds, then the background is a cue to object identity—but can cause errors. Most likely, the main reason that error rates are high is that we still do not fully understand how to represent objects, and the features that we use do not encapsulate all that is important, nor do they suppress enough irrelevant information.

Category	2007	2008	2009	2010
aeroplane	0.775	0.811	0.881	0.933
bicycle	0.636	0.543	0.686	0.790
bird	0.561	0.616	0.681	0.716
boat	0.719	0.678	0.729	0.778
bottle	0.331	0.300	0.442	0.543
bus	0.606	0.521	0.795	0.859
car	0.780	0.595	0.725	0.804
cat	0.588	0.599	0.708	0.794
chair	0.535	0.489	0.595	0.645
cow	0.426	0.336	0.536	0.662
diningtable	0.549	0.408	0.575	0.629
dog	0.458	0.479	0.593	0.711
horse	0.775	0.673	0.731	0.820
motorbike	0.640	0.652	0.723	0.844
person	0.859	0.871	0.853	0.916
pottedplant	0.363	0.318	0.408	0.533
sheep	0.447	0.423	0.569	0.663
sofa	0.509	0.454	0.579	0.596
train	0.792	0.778	0.860	0.894
tvmonitor	0.532	0.647	0.686	0.772
# methods	2	5	4	6
# comp	17	18	48	32

TABLE 16.1: Average precision of the best classification method for each category for the Pascal image classification challenge by year (per category; the method that was best at "person" might not be best at "pottedplant"), summarized from `http://pascallin.ecs.soton.ac.uk/challenges/VOC/`. The **bottom** rows show the number of methods in each column and the total number of methods competing (so, for example, in 2007, only 2 of 17 total methods were best in category; each of the other 15 methods was beaten by something for each category). Notice that the average precision grows, but not necessarily monotonically (this is because the test set changes). Most categories now work rather well.

16.2.4 Large Numbers of Classes

The number of categories has grown quite quickly. A now little-used dataset had five classes in it; in turn, this was replaced with a now obsolete ten class dataset; a 101-class dataset; a 256-class dataset; and a 1,000-class dataset (details in Section 16.3.2). Figure 16.20 compares results of recent systems on Caltech 101 (the 101-class dataset described in Section 16.3.2) and on Caltech 256 (256-classes; Section 16.3.2). For these datasets, some care is required when one computes error statistics. Two statistics are natural. The first is the percent of classification attempts that are successful over all test examples. This measure is not widely used, for the following reason: imagine that one class is numerous, and easy to classify; then the error statistic will be dominated by this class, and improvements may just mean that one is getting better at classifying this class. However, for some applications, this might be the right thing. For example, if one is confident that the dataset represents the relative frequency of classes well, then this error rate is

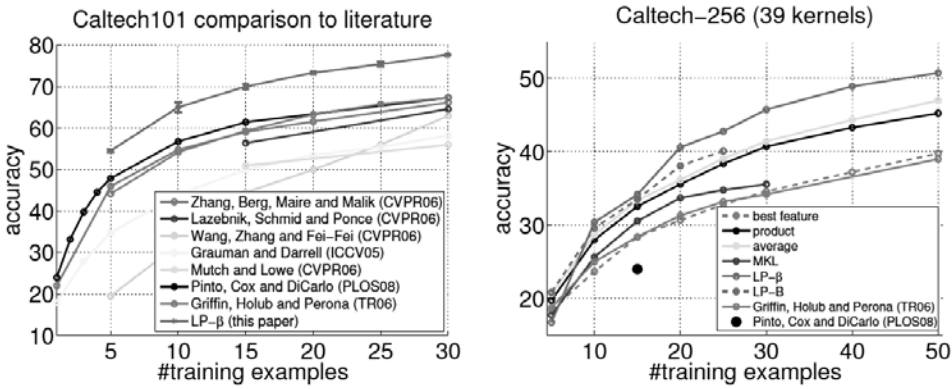

FIGURE 16.20: Graphs illustrating typical performance on Caltech 101 for single descriptor types (**left**) and on Caltech 256 for various types of descriptor (**right**; notice the vertical scale is different), plotted against the number of training examples. Although these figures are taken from a paper advocating nearest neighbor methods, they illustrate performance for a variety of methods. Notice that Caltech 101 results, while not perfect, are now quite strong; the cost of going to 256 categories is quite high. Methods compared are due to: Zhang *et al.* (2006*b*), Lazebnik *et al.* (2006), Wang *et al.* (2006), Grauman and Darrell (2005), Mutch and Lowe (2006), Griffin *et al.* (2007), and Pinto *et al.* (2008); the graph is from Gehler and Nowozin (2009), which describes multiple methods (anything without a named citation on the graph). *This figure was originally published as Figure 2 of "On Feature Combination for Multiclass Object Classification," by P. Gehler and S. Nowozin Proc. ICCV 2009, 2009* © *IEEE 2009.*

a good estimate of the problems that will be encountered when using the classifier.

The other statistic that is natural is the average of per-class error rates. This weights down the impact of frequent classes in the test dataset; to do well at this error measure, one must do well at all classes, rather than at frequent classes. This statistic is now much more widely used, because there is little evidence that classes occur in datasets with the same frequency they occur in the world.

It is usual to report performance as the number of training examples goes up, because this gives an estimate of how well features (a) suppress within class variations and (b) expose between class variations. Notice how the performance of all methods seems to stop growing with the number of training examples (though it is hard to confirm what happens with very large numbers, as some categories have relatively few examples).

Generally, strong modern methods do somewhat better on Caltech 101 than on Caltech 256, and better on Caltech 256 than on datasets with more categories, though it is difficult to be sure why. One possibility is that classification becomes a lot harder when the number of categories grows, most likely because of feature effects. Deng *et al.* (2010) show that the performance of good modern methods declines as the number of categories increases, where the set of categories is selected at random from a very large set. This suggests that increasing the number of categories exposes problems in feature representations that might otherwise go unnoticed, because it increases the chances that two of the categories are quite sim-

ilar (or at least look similar to the feature representation). As a result, performance tends to go down as the number of categories is increased. Another possibility is that Caltech 101 is a more familiar dataset, and so feature design practices have had more time to adapt to its vagaries. If this is the case, then methods that do well on this dataset are not necessarily good methods; instead, they are methods that have been found by the community to do well *on a particular dataset*, which is a form of selection bias. Yet another, equally disturbing possibility is that no current methods do well on large collections of categories because it is so hard to search for a method that does so. The pragmatics of dealing with large numbers of categories is very demanding. Simply training a single method may take CPU years and a variety of clever tricks; classifying a single image also could be very slow (Deng *et al.* 2010).

Working with large numbers of categories presents other problems, too. Not all errors have the same significance, and the semantic status of the categories is unclear (Section 18.1.3). For example, classifying a cat as a dog is probably not as offensive as classifying it as a motorcycle. This is a matter of loss functions. In practice, it is usual to use the so-called *0-1 loss*, where any classification error incurs a loss of one (equivalently, to count errors). This is almost certainly misleading, and is adopted mainly because it is demanding and because there is no consensus on an appropriate replacement. One possibility, advocated by (Deng *et al.* 2010), is to use semantic resources to shape a loss function. For example, Wordnet is a large collection of information about the semantic relations between classes (Fellbaum (1998); Miller *et al.* (1990)). Words are organized into a hierarchy. For example, "dog" (in the sense of an animal commonly encountered as a domestic pet) has child nodes (*hyponyms*), such as "puppy," and ancestors (*hypernyms*) "canid," "carnivore," and, eventually, "entity." A reasonable choice of loss function could be the hop distance in this tree between terms. In this case, "dog" and "cat" would be fairly close, because each has "carnivore" as a grandparent node, but "dog" and "motorcycle" are quite different, because their first common ancestor is many levels removed ("whole"). One difficulty with this approach is that some objects that are visually quite similar and appear in quite similar contexts might be very different in semantics (bird and aircraft, for example). Deng *et al.* (2010) advocate using the height above the correct label of the nearest ancestor, common between the correct and predicted label.

16.2.5 Flowers, Leaves, and Birds: Some Specialized Problems

Image classification techniques are valuable in all sorts of specialized domains. For example, there has been considerable recent progress in classifying flowers automatically from pictures. A natural system architecture is to query a collection of labeled flower images with a query image. If a short list of similar images contains the right flower, that might be sufficient, because geographic distribution cues might rule out all other flowers on the list. The problem is tricky because within-class variation could be high, as a result of pictures taken from different viewing directions, and between-class variation can be low (Figure 16.21). Nilsback and Zisserman (2010) describe a system for matching flower images that computes color, texture, and shape features, then learns a combination of distances in each feature that gives

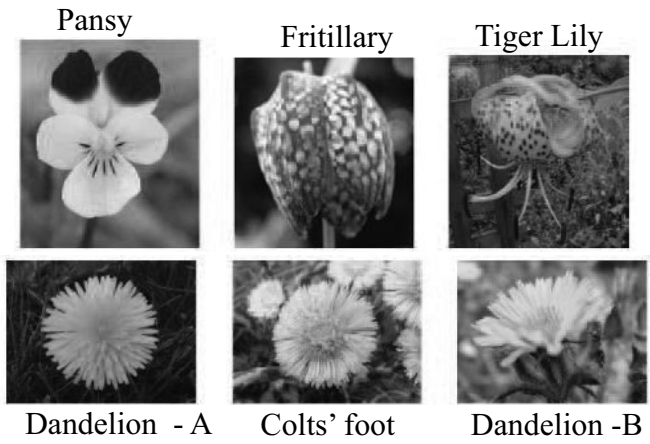

FIGURE 16.21: Identifying a flower from an image is one useful specialized application for image classification techniques. This is a challenging problem. Although some flowers have quite distinctive features (for example, the colors and textures of the pansy, the fritillary, and the tiger lily), others are easy to confuse. Notice that dandelion-A (**bottom**) looks much more like the colts' foot than like dandelion-B. Here the within-class variation is high because of changes of aspect, and the between-class variation is small. *This figure was originally published as Figures 1 and 8 of "A Visual Vocabulary for Flower Classification," by M.E. Nilsback and A. Zisserman, Proc. IEEE CVPR 2006,* © *IEEE 2006.*

the best performance for this short list; the best results on this dataset to date are due to a complex multiple-kernel learning procedure (Gehler and Nowozin 2009).

Belhumeur *et al.* (2008) describe a system for automatic matching of leaf images to identify plants; they have released a dataset at `http://herbarium.cs.columbia.edu/data.php`. This work has recently resulted in an iPad app, named Leafsnap, that can identify trees from photographs of their leaves (see `http://leafsnap.com`).

Often, although one cannot exactly classify every image, one can reduce the load on human operators in important ways with computer vision methods. For example, Branson *et al.* (2010) describe methods to classify images of birds to species level that use humans in the loop, but can reduce the load on the human operator. Such methods are likely to lead to apps that will be used by the very large number of amateur birdwatchers.

16.3 IMAGE CLASSIFICATION IN PRACTICE

Numerous codes and datasets have been published for image classification; the next two sections give some pointers to materials available at the time of writing. Image classification is a subject in flux, so methods change quickly. However, one can still make some general statements. Section 16.3.3 summarizes the difficulties that result because datasets cannot be as rich as the world they represent, and Section 16.3.4 describes methods for collecting data relatively cheaply using crowdsourcing.

16.3.1 Codes for Image Features

Oliva and Torralba provide GIST feature code at `http://people.csail.mit.edu/torralba/code/spatialenvelope/`, together with a substantial dataset of outdoor scenes.

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/`.

The pyramid match kernel is an earlier variant of the spatial pyramid kernel described in Section 16.1.4; John Lee provides a library, `libpmk`, that supports this kernel at `http://people.csail.mit.edu/jjl/libpmk/`. There are a variety of extension libraries written for `libpmk`, including implementations of the pyramid kernel, at this URL.

Li Fei-Fei, Rob Fergus, and Antonio Torralba publish example codes for core object recognition methods at `http://people.csail.mit.edu/torralba/shortCourseRLOC/`. This URL is the online repository associated with their very successful short course on recognizing and learning object categories.

VLFeat is an open-source library that implements a variety of popular computer vision algorithms, initiated by Andrea Vedaldi and Brian Fulkerson; it can be found at `http://www.vlfeat.org`. VLFeat comes with a set of tutorials that show how to use the library, and there is example code showing how to use VLFeat to classify Caltech-101.

There is a repository of code links at `http://featurespace.org`.

At the time of writing, multiple-kernel learning methods produce the strongest results on standard problems, at the cost of quite substantial learning times. Section 15.3.3 gives pointers to codes for different multiple-kernel learning methods.

16.3.2 Image Classification Datasets

There is now a rich range of image classification datasets, covering several application topics. Object category datasets have images organized by category (e.g., one is distinguishing between "bird"s and "motorcycle"s, rather than between particular species of bird). Five classes (motorbikes, airplanes, faces, cars, spotted cats, together with background, which isn't really a class) were introduced by Fergus *et al.* (2003) in 2003; they are sometimes called Caltech-5. Caltech-101 has 101 classes, was introduced in Perona *et al.* (2004) and by Fei-Fei *et al.* (2006), and can be found at `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`. This dataset is now quite well understood, but as Figure 16.20 suggests, it is not yet exhausted. Caltech-256 has 256 classes, was introduced by (Griffin *et al.* 2007), and can be found at `http://www.vision.caltech.edu/Image_Datasets/Caltech256/`. This dataset is still regarded as challenging.

LabelMe is an image annotation environment that has been used by many users to mark out and label objects in images; the result is a dataset that is changing and increasing in size as time goes on. LabelMe was introduced by Russell *et al.* (2008), and can be found at `http://labelme.csail.mit.edu/`.

The Graz-02 dataset contains difficult images of cars, bicycles, and people in natural scenes; it is originally due to Opelt *et al.* (2006), but has been recently reannotated Marszalek and Schmid (2007). The reannotated edition can be found

FIGURE 16.22: Torralba and Efros (2011) show one disturbing feature of modern classification datasets; that it is quite easy for skilled insiders to "name that dataset." Here we show a sample of images from current datasets (those not described in the text can be found by a search); you should try and match the image to the dataset. It is surprisingly easy to do. *This figure was originally published as Figures 1 of "Unbiased look at dataset bias," by A. Torralba and A. Efros, Proc. IEEE CVPR 2011,* © *IEEE 2011.*

at `http://lear.inrialpes.fr/people/marszalek/data/ig02/`.

Imagenet contains tens of millions of examples, organized according to the Wordnet hierarchy of nouns; currently, there are examples for approximately 17,000 nouns. Imagenet was originally described in Deng *et al.* (2009), and can be found at `http://www.image-net.org/`.

The Lotus Hill Research Institute publishes a dataset of images annoted in detail at `http://www.imageparsing.com`; the institute is also available to prepare datasets on a paid basis.

Each year since 2005 has seen a new Pascal image classification dataset; these are available at `http://pascallin.ecs.soton.ac.uk/challenges/VOC/`.

There are numerous specialist datasets. The Oxford visual geometry group publishes two flower datasets, one with 17 categories and one with 102 categories; each can be found at `http://www.robots.ox.ac.uk/~vgg/data/flowers/`. Other datasets include a "things" dataset, a "bottle" dataset, and a "camel" dataset, all from Oxford (`http://www.robots.ox.ac.uk/~vgg/data3.html`).

There is a bird dataset published by Caltech and UCSD jointly at `http:`

//www.vision.caltech.edu/visipedia/CUB-200.html.

Classifying materials has become a standard task, with a standard dataset. The Columbia-Utrecht (or CURET) material dataset can be found at http://www.cs.columbia.edu/CAVE/software/curet/; it contains image textures from over 60 different material samples observed with over 200 combinations of view and light direction. Details on the procedures used to obtain this dataset can be found in Dana *et al.* (1999). More recently, Liu *et al.* (2010) offer an alternative and very difficult material dataset of materials on real objects, which can be found at http://people.csail.mit.edu/celiu/CVPR2010/FMD/.

We are not aware of collections of explicit images published for use as research datasets, though such a dataset would be easy to collect.

There are several scene datasets now. The largest is the SUN dataset (from MIT; http://groups.csail.mit.edu/vision/SUN/; Xiao *et al.* (2010)) contains 130,519 images of 899 types of scene; 397 categories have at least 100 examples per category. There is a 15-category scene dataset used in the original spatial pyramid kernel work at http://www-cvr.ai.uiuc.edu/ponce_grp/data/.

It isn't possible (at least for us!) to list all currently available datasets. Repositories that contain datasets, and so are worth searching for a specialist dataset, include: the pilot European Image Processing Archive, currently at http://peipa.essex.ac.uk/index.html; Keith Price's comprehensive computer vision bibliography, whose root is http://visionbib.com/index.php, and with dataset pages at http://datasets.visionbib.com/index.html; the Featurespace dataset pages, at http://www.featurespace.org/; and the Oxford repository, at http://www.robots.ox.ac.uk/~vgg/data.html.

16.3.3 Dataset Bias

Datasets can suffer from *bias*, where properties of the dataset misrepresent properties of the real world. This is not due to mischief in the collecting process; it occurs because the dataset must be much smaller than the set of all images of an object. Some bias phenomena can be quite substantial. For example, Figure 16.22 shows that people can get quite good at telling which dataset a picture was taken from, as can computers (Figure 16.23, whose caption gives the right answers for Figure 16.22). As another example, Figure 16.24 shows the mean image of a set of Caltech 101 images. Clearly, in this case, each image in the dataset looks quite a lot like every other image in its class and not much like images in other classes. This doesn't mean that it is easy to get very strong recognition results; compare Figure 16.20. The best current strategies for avoiding bias are (a) to collect large datasets from a variety of different sources; (b) to evaluate datasets carefully using baseline methods before using them to evaluate complex methods; and (c) to try and quantify the effects of bias by evaluating on data collected using a different strategy than that used to collect the training data. Each is fairly crude. Improved procedures would be most valuable.

16.3.4 Crowdsourcing Dataset Collection

Recently, dataset builders have made extensive use of *crowdsourcing*, where one pays people to label data. One such service is Amazon's Mechanical Turk. Crowd-

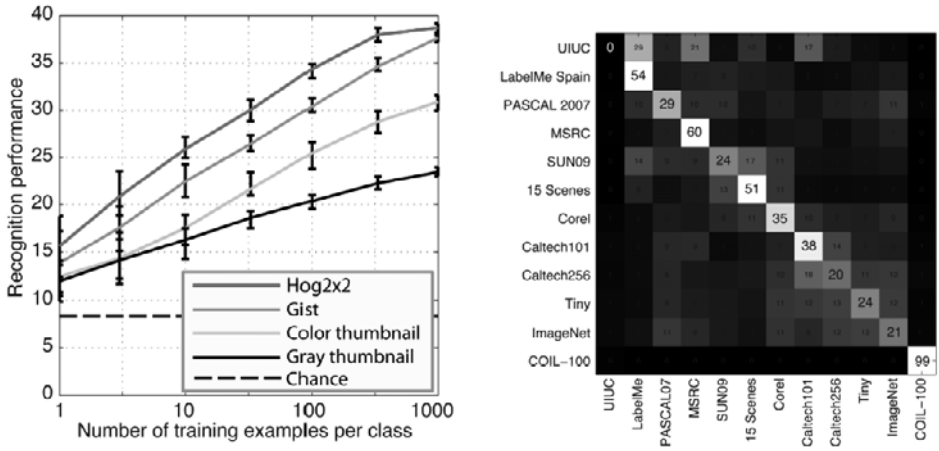

FIGURE 16.23: Computers do very well at "name that dataset." On the **left**, classification accuracy as a function of training size for some different features; notice that classifiers are really quite good at telling which dataset a picture came from. On the **right**, the class confusion matrix, which suggests that these datasets are well-separated. The answers to the question in Figure 16.22 are: (1) Caltech-101, (2) UIUC, (3) MSRC, (4) Tiny Images, (5) ImageNet, (6) PASCAL VOC, (7) LabelMe, (8) SUNS-09, (9) 15 Scenes, (10) Corel, (11) Caltech-256, (12) COIL-100. *This figure was originally published as Figure 2 of "Unbiased look at dataset bias," by A. Torralba and A. Efros, Proc. IEEE CVPR 2011,* © *IEEE 2011.*

sourcing services connect people on the Internet willing to do tasks for money with people who have tasks and money. Generally, one builds an interface to support the task (for example, your interface might display an image and some radio buttons to identify the class), then registers the task and a price. Workers then do the task, you pay the service, and they transmit money to the workers. Important issues here are quality control—are people doing what you want them to do?—and pricing—how much should you pay for a task? Quality control strategies include: prequalifying workers; sampling tasks and excluding workers who do the task poorly; and using another set of workers to evaluate the results of the first set. We are not aware of good principled pricing strategies right now. However, some guidelines can be helpful. Workers seem to move quickly from task to task, looking for ones that are congenial and well-paid. This means that all tasks seem to experience a rush of workers, which quickly tails off if the price is wrong. Paying more money always seems to help get tasks completed faster. There seems to be a pool of workers who are good at identifying overpaid tasks with poor quality control, but most workers are quite conscientious. Finally, interface design can have a huge impact on the final accuracy of labeled data. These ideas are now quite pervasive. Examples of recent datasets built with some input from Mechanical Turk include Deng *et al.* (2009), Endres *et al.* (2010), Parikh and Grauman (2011), and Xiao *et al.* (2010). Sorokin and Forsyth (2008) give a variety of strategies and methods to use the service. Vijayanarasimhan and Grauman (2011) supply good evidence that active

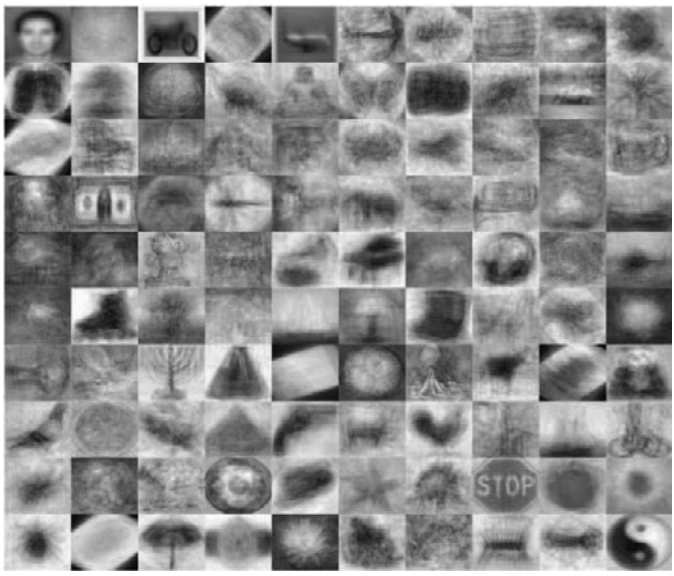

FIGURE 16.24: The average image for each of 100 categories from the Caltech 101 image classification dataset. Fairly obviously, these pictures consist of isolated objects, and the mean of each class is far away from the mean of other classes. This does not mean that these images are easy to classify (compare Figure 16.20); instead, it is an illustration of the fact that all datasets must contain statistical regularities that are not present in the world. *This figure created by A. Torralba, and used with his permission.*

learning can improve costs and quality; see also Vijayanarasimhan and Grauman (2009). Vondrick *et al.* (2010) show methods to balance human labor (which is expensive and slow, but more accurate) with automatic methods (which can propagate existing labels to expose what is already known, and can be fast and cheap) for video annotation. Crowdflower, which is a service that helps build APIs and organize crowdsourcing, can be found at `http://crowdflower.com/`.

16.4 NOTES

Generally, successful work in image classification involves constructing features that expose important properties of the classes to the classifier. The classifier itself can make some difference, but seems not to matter all that much. We have described the dominant feature constructions, but there is a particularly rich literature on feature constructions; there are pointers to this in the main text.

We suspect that the best methods for explicit image detection are not published now, but instead just used, because good methods appear to have real financial value. All follow the lines sketched out in our section, but using a range of different features and of classifiers. Experiments are now on a relatively large scale.

One application we like, but didn't review in this chapter, is sign language understanding. Here an automated method watches a signer, and tries to transcribe the sign language into text. Good start points to this very interesting literature

include Starner *et al.* (1998), Buehler *et al.* (2009), Cooper and Bowden (2009), Farhadi *et al.* (2007), Erdem and Sclaroff (2002), Bowden *et al.* (2004), Buehler *et al.* (2008), and Kadir *et al.* (2004). Athitsos *et al.* (2008) describe a dataset.

Visual words are a representation of important local image patches. While the construction we described is fairly natural, it is not the only possible construction. It is not essential to describe only interest points; one could use a grid of sample points, perhaps as fine as every pixel. The description does not have to be in terms of SIFT features. For example, one might extend it by using some or all of the color sift features described briefly in Section 5.4.1. Many authors instead compute a vector of filter responses (section 6.1 for this as a texture representation; section 16.1.8 for applications to texture material classification). An important alternative is to work directly with small local image patches, say 5×5 pixels in size. The vocabulary of such visual words could be very big indeed, and special clustering techniques are required to vector quantize. In each case, however, the main recipe remains the same: decide on a way of identifying local patches (interest points, sampling, etc.); decide on a local patch representation; vector quantize this representation to form visual words; then represent the image or the region with a histogram of the important visual words.

PROGRAMMING EXERCISES

16.1. Build a classifier that classifies materials using the dataset of Liu *et al.* (2010). Compare the performance of your system using the main feature constructions described here (GIST features; visual words; spatial pyramid kernel). Investigate the effect of varying the feature construction; for example, is it helpful to use C-SIFT descriptors?

16.2. Build a classifier that classifies scenes using the dataset of Xiao *et al.* (2010). Compare the performance of your system using the main feature constructions described here (GIST features; visual words; spatial pyramid kernel). Investigate the effect of varying the feature construction; for example, is it helpful to use C-SIFT descriptors?

16.3. Search online for classification and feature construction codes, and replicate an image classification experiment on a standard dataset (we recommend a Caltech dataset or a PASCAL dataset; your instructor may have an opinion, too). Do you get exactly the same performance that the authors claim? Why?