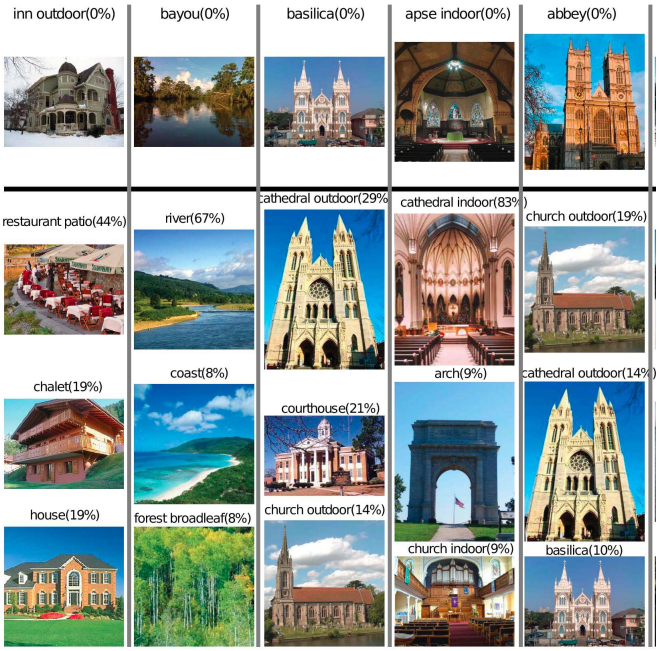

FIGURE 16.18: Not all scene categories are easily distinguished by humans. These are examples from the SUN dataset (Xiao *et al.* 2010). On the **top** of each column is an example of a difficult category; **below** are examples from three categories that are commonly confused with it by people. Such 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-

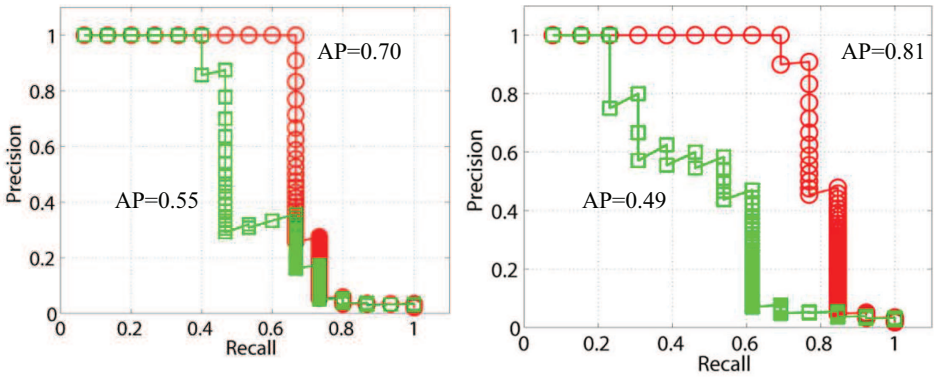

FIGURE 16.19: Plots of precision as a function of recall for six object queries. Notice how precision generally declines as recall goes up (the occasional jumps have to do with finding a small group of relevant images; such jumps would become arbitrarily narrow and disappear in the limit of an arbitrarily 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

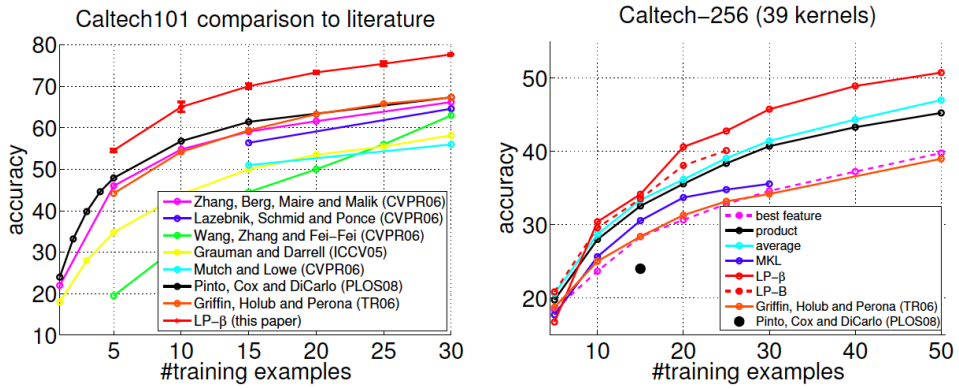

FIGURE 16.20: Graphs illustrating typical performance on Caltech 101 for single descriptor types (**left**) and on Caltech 256 for various types of descriptor (**right**; notice the vertical scale is different), plotted against the number of training examples. Although these figures are taken from a 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

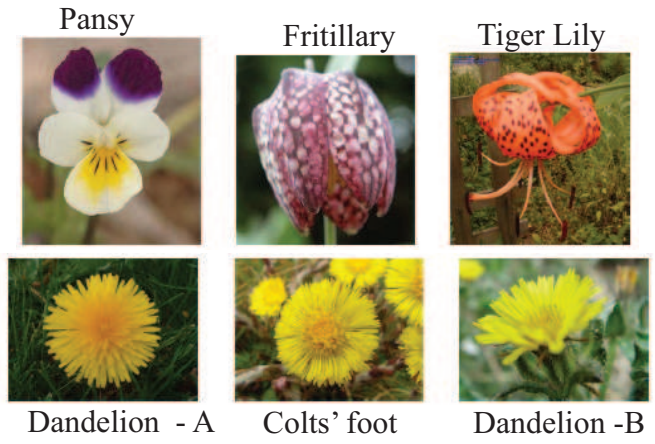

FIGURE 16.21: Identifying a flower from an image is one useful specialized application for image classification techniques. This is a challenging problem. Although some flowers have quite distinctive features (for example, the colors and textures of the pansy, the fritillary, and the tiger lily), others 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

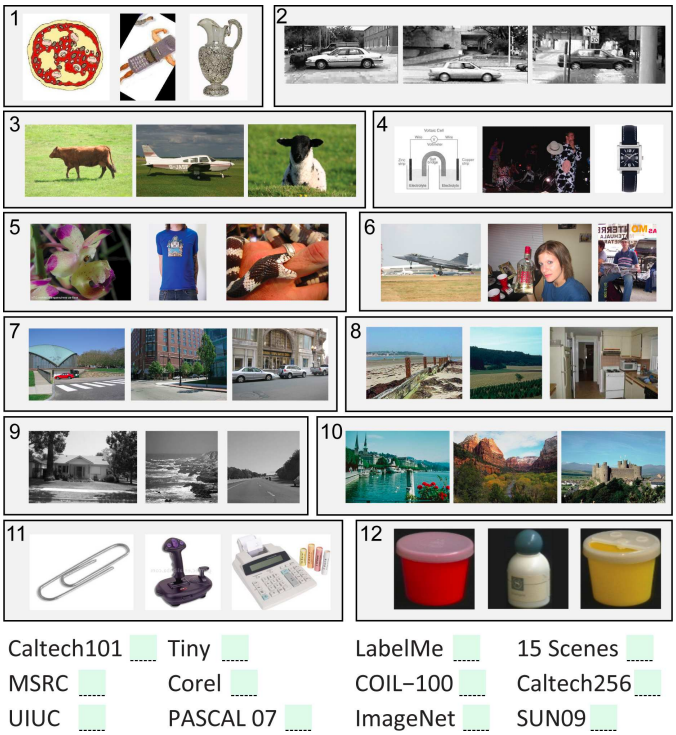

FIGURE 16.22: Torralba and Efros (2011) show one disturbing feature of modern classification datasets; that it is quite easy for skilled insiders to "name that dataset." Here we show a sample of images from current datasets (those not described in the text can be found by a search); you should try and 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-

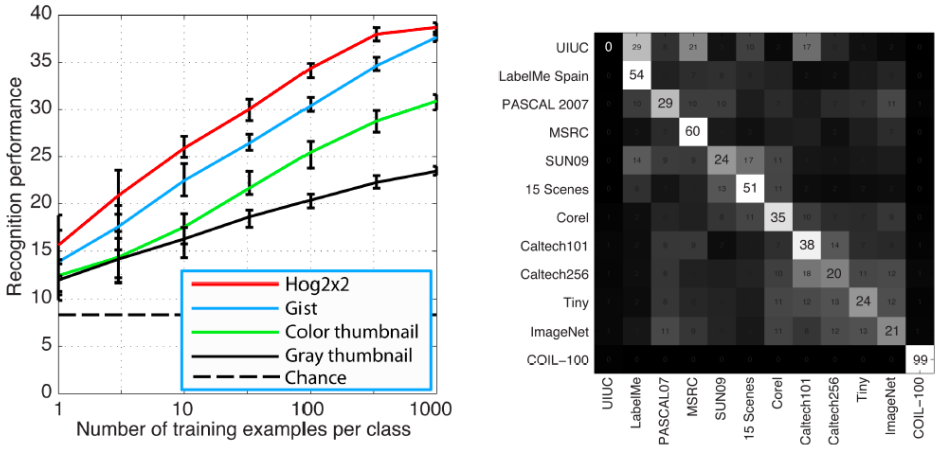

FIGURE 16.23: Computers do very well at "name that dataset." On the **left**, classification accuracy as a function of training size for some different features; notice that classifiers are really quite good at telling which dataset a picture came from. On the **right**, the class confusion matrix, which 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

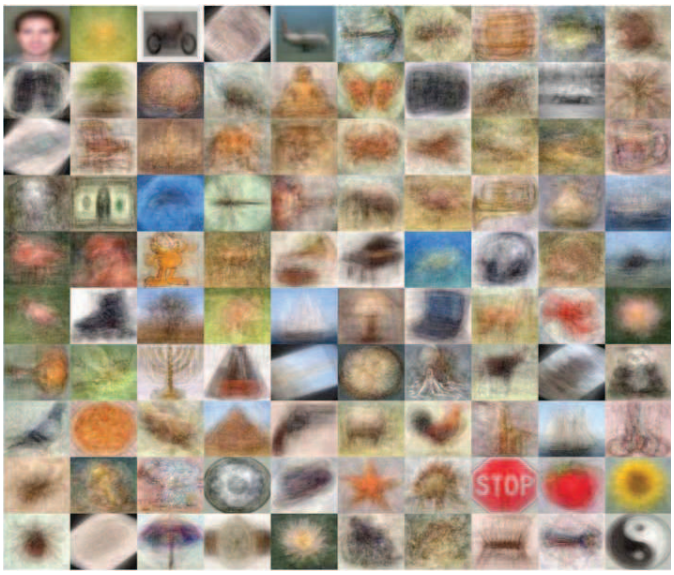

FIGURE 16.24: The average image for each of 100 categories from the Caltech 101 image classification dataset. Fairly obviously, these pictures consist of isolated objects, and the mean of each class is far away from the mean of other classes. This does not mean that these images are easy to classify (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?