CHAPTER 5

Categorizing Images and Regions

Once we have decided on how to represent the scene, we must recover that representation from images. As discussed in the background (Chapter 2), early attempts at scene understanding involved many hand-tuned rules and heuristics, limiting generalization. We advocate a data-driven approach in which supervised examples are used to learn how image features relate to the scene model parameters.

In this chapter, we start with an overview of the process of categorizing or scoring regions, which is almost always a key step in recovering the 3D scene space. Then, we present some basic guidelines for segmentation, choice of features, classification, and dataset design. Finally, we survey a broad set of useful image features.

1. Overview of Image Labeling

Although there are many different scene models, most approaches follow the same basic process for estimating them from images. First, the image is divided into smaller regions. The regions could be a grid of uniformly shaped patches, or they could fit the boundaries in the image. Then, features are computed over each region. Next, a classifier or predictor is applied to the features of each region, yielding scores for the possible labels or predicted depth values. For example, the regions might be categorized into geometric classes or assigned a depth value. Often, a post-processing step is then applied to incorporate global priors, such as that the scene should be made up of a small number of planes.

Many approaches that seem unrelated at first glance are formulated as region classification:

- **Automatic Photo Pop-up** [**51**]:
  1. Create many overlapping regions.
  2. Compute color, texture, position, and perspective features for each region.
  3. Based on the features, use a classifier to assign a confidence that the region is good (corresponds to one label) and a confidence that the region is part of the ground, a vertical surface, or the sky.
  4. Average over regions to get the confidence for each label at each pixel. Choose largest confidence to assign each pixel to "ground", "vertical", or "sky".
  5. Fit a set of planar billboards are fit to the vertical regions, compute world coordinates and texture map onto the model.
- **Make3D** [**105**]:
  1. Divide the image into small regions.
  2. Compute texture features for each region.
  3. Predict 3D plane parameters for each region using a linear regressor.
  4. Refine estimates using a Markov Random Field model [**76**], applying pairwise priors such as that neighboring regions are likely to be connected and co-planar.
- **Box Layout** [**47**]:
  1. Estimate three orthogonal vanishing points.
  2. Create regions for the walls, floor, and ceiling by sampling pairs of rays from two of the vanishing points.
  3. Compute edge and geometric context features within each region.
  4. Score each candidate (a set of wall, floor, and ceiling regions) using a linear classifier. Choose the highest scoring candidate.

In the above descriptions, critical details of features, models, and classification method were omitted. The point is that many approaches for inferring scene space follow the same pattern: divide into regions, score or categorize them, and assemble into a 3D model.

Classifiers and predictors are typically learned in a training stage on one set of images and applied in a testing stage on another set of image (Figure 5.1). For region classification, the sets of feature values computed within each image are examples. In training, both labels and examples are provided to the learning algorithm, with the goal of learning a classifier that will correctly predict the label for a new test example. The efficacy of the classifier depends on the how informative the features are, the form and regularization of the classifier, and the number of training examples.

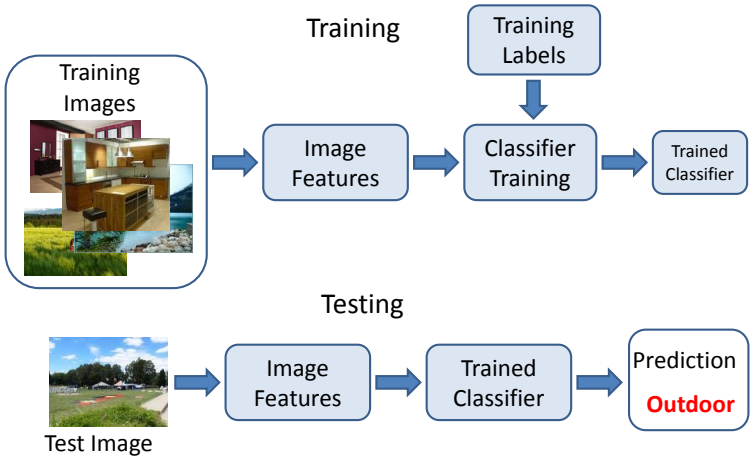

FIGURE 5.1. Overview of the training and testing process for an image categorizer. For categorizing regions, the same process is used, except that the regions act as the individual examples with features computed over each.

2. Guiding Principles

The following are loose principles based on extensive experience in designing features and in application of machine learning to vision.

2.1. Creating Regions

When labeling pixels or regions, it is important to consider the spatial support used to compute the features. Many features, such as color and texture histograms must be computed over some region, the spatial support. Even the simplest features, such as average intensity tend to provide more reliable classification when computed over regions, rather than pixels. The region could be created by dividing the image into a grid, by oversegmenting into hundreds of regions, or by attempting to segment the image into a few meaningful regions. Typically, based on much personal experience, oversegmentation works better than dividing the image into arbitrary blocks. Useful oversegmentation methods include the graph-based method of Felzenszwalb and Huttenlocher [**31, 52**], the mean-shift algorithm [**18, 129**], recursive normalized cuts [**111, 82**], and watershed-based methods [**4**]. Code for all of these is easily found online. Methods based on normalized cuts and watershed tend to be more regular but also may require more regions to avoid merging thin objects. With a single, more aggressive segmentation, the benefit of improved spatial support may be negated by errors in the segmentation. A successful alternative is to create multiple segmentations, either by varying segmentation parameters or by randomly seeding a clustering of smaller regions.

Then, pixel labels are assigned by averaging the classifier confidences for the regions that contain the pixel.

Depending on the application, specialized methods for proposing regions may be appropriate. For example, Hedau et al. [**47**] generates wall regions by sampling rays from orthogonal vanishing points, and Lee et al. [**71**] propose wall regions using line segments that correspond to the principle directions. Gupta et al. [**39**] propose whole blocks based on generated regions and geometric context label predictions.

2.2. Choosing Features

The designer of a feature should carefully consider the desired sensitivity to various aspects of the shape, albedo, shading, and viewpoint. For example, the gradient of intensity is sensitive to changes in shading due to surface orientation but insensitive to average brightness or material albedo. The SIFT descriptor [**79**] is robust to in-plane orientation, but that discarded information about the dominant orientation may be valuable for object categorization. Section 3 discusses many types of features in more detail.

In choosing a set of features, there are three basic principles:

- Coverage: Ensure that all relevant information is captured. For example, if trying to categorize materials in natural scenes, color, texture, object category, scene category, position within the scene, and surface orientation can all be helpful. Coverage is the most important principle because no amount of data or fancy machine learning technique can prevent failure if the appearance model is too poor.

- Concision: Minimize the number of features without sacrificing coverage. With fewer features, it becomes feasible to use more powerful classifiers, such as kernelized SVM or boosted decision trees, which may improve accuracy on both training and test sets. Additionally, for a given classifier, reducing the number of irrelevant or marginally relevant features will improve generalization, reducing the margin between training and test performance.

- Directness: Design features that are independently predictive, which will lead to a simpler decision boundary, improving generalization.

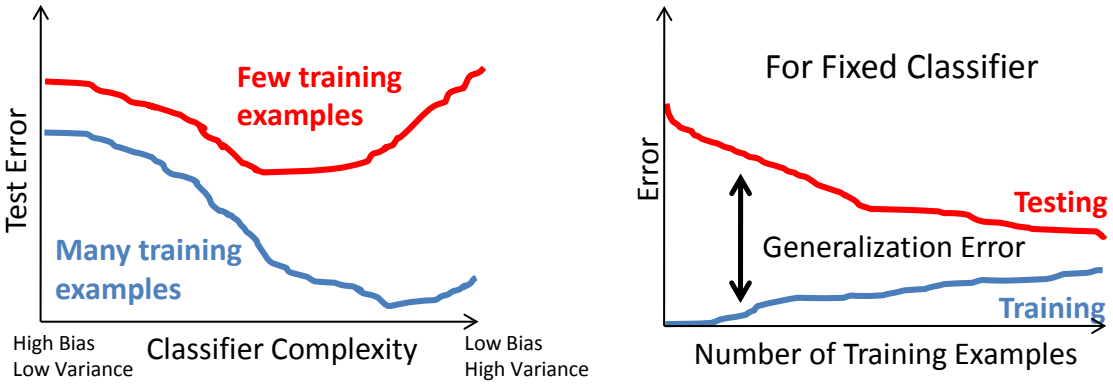

FIGURE 5.2. Left: As the complexity of the classifier increases, it becomes harder to correctly estimate the parameters. With few training examples, a lower complexity classifier (e.g., a linear classifier) may outperform. If more training examples are added, it may improve performance to increase the classifier complexity. Right: As more training examples are added, it becomes harder to fit them all, so training error tends to go up. But the training examples provide a better expectation of what will be seen during testing, so test error goes down.

2.3. Classifiers

Features and classifiers should be chosen jointly. The main considerations are the hypothesis space of the classifier, the type of regularization, the amount of training data, and computational efficiency. See Figure 5.2 for an illustration of two important principles relating to complexity, training size, and generalization error.

The hypothesis space is determined by the form of the decision boundary and indicates the set of possible decision functions that can be learned by the classifier. A classifier with a linear decision boundary has a much smaller hypothesis space than a nearest neighbor classifier, which could arbitrarily assign labels to examples, depending on the training labels.

Regularization can be used to penalize complex decision functions. For example, SVMs and L2 logistic regression include a term of summed square weights in the minimization function, which encodes a preference that no particular feature should have too much weight.

If few training examples are available, then a classifier with a simple decision boundary (small hypothesis space) or strong regularization should be used to avoid overfitting. Overfitting is when the margin between training and test error increases faster than the training error decreases. If many training examples are available, a more powerful classifier may be appropriate. Similarly, if the number of features is very large, a simpler classifier is likely to perform best. For example, if classifying a region using color and texture histograms with thousands of individually weak

features, a linear SVM is a good choice. If a small number of carefully designed features are used, then a more flexible boosted decision tree classifier may outperform.

Though out of scope for this document, it is worthwhile to study the generalization bounds of the various classifiers. The generalization bounds are usually not useful for predicting performance, but they do provide insight into expected behavior of the classifier with irrelevant features or small amounts of data.

As a basic toolkit, we suggest developing familiarity with the following classifiers: SVM [**108**] (linear and kernelized), Adaboost [**35**] (particularly with decision tree weak learners), logistic regression [**86**], and nearest neighbor [**25**] (a strawman that is often embarrassingly hard to beat).

2.4. Datasets

See Berg et al. [**12**] for a lengthy discussion on datasets and annotation. The main considerations in designing a dataset (assuming that a representation has already been decided) is the level of annotation, the number of training and test images, and the difficulty and diversity of scenes. More detailed annotation is generally better, as parts of the annotation can always be ignored, but cost of collection must be considered. More data makes it easier to use larger feature sets and more powerful classifiers to achieve higher performance.

The issue of bias in datasets must be treated carefully. Bias could be due to the acquisition or sampling procedure, to conventions in photography, or social norms. Every dataset is biased. For example, a random selection of photos from Flickr would have many more pictures of people and many more scenes from mountain tops than if you took pictures from random locations and orientations around the world. Bias is not always bad. If we care about making algorithms that work well in consumer photographs, we may want to take advantage of the bias, avoiding the need to achieve good results in photographs that are pointed directly at the ground or into the corner of a wall. The structure of our visual world comes from both physical laws and convention, and we would be silly not to take advantage of it.

But we should be careful to distinguish between making improvements by better fitting the foibles of a particular dataset or evaluation measure and improvements that are likely to apply to many datasets. As a simple example, the position of a pixel is a good predictor of its object category in the MSRC dataset [**112**], so that including it as a feature will greatly improve performance. However, that classifier will not perform well on other datasets, such as LabelMe [**101**], because the biases in photography are different. Likewise, it may be possible to greatly improve results in the PASCAL

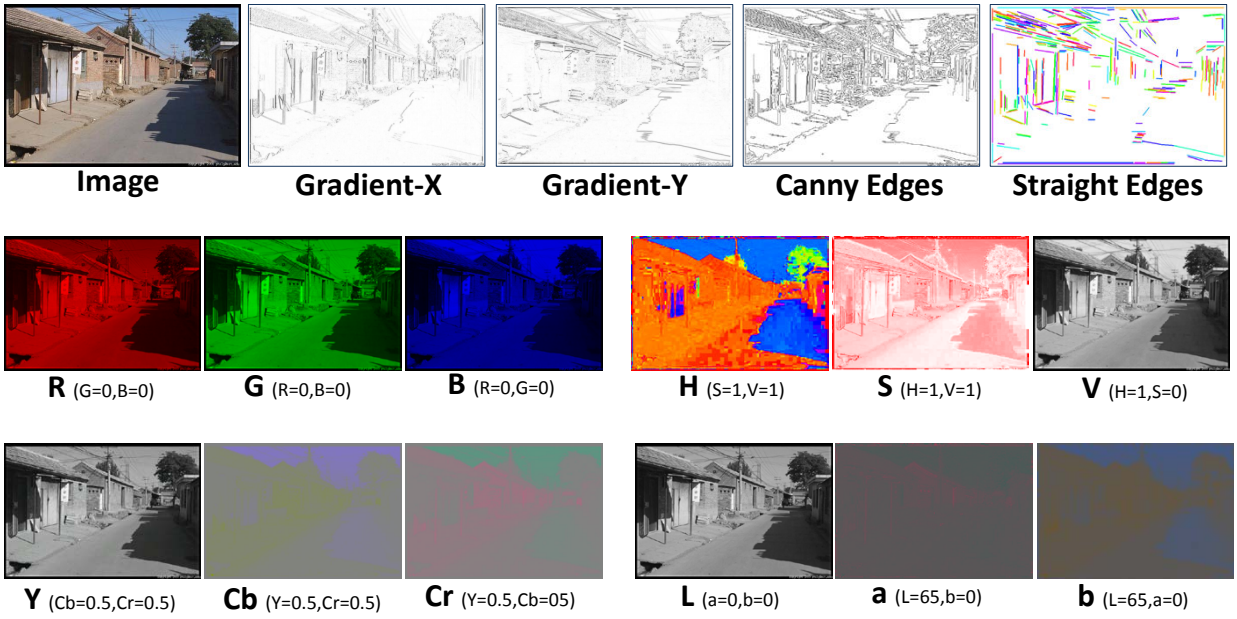

FIGURE 5.3. Gradients, edges, and examples of color spaces for the input image.

challenge [**28**] by improving the localization of bounding boxes, but that improvement may not apply to other evaluation criteria.

Although it may seem that big datasets avoid bias, they don't. Whether you sample one hundred or one hundred million examples from Flickr, a large number of them will be framed around people looking at the camera, which is different than what would be observed on an autonomous vehicle. In fact, big datasets make it easier to exploit the foibles of the dataset by making large feature sets more effective, so that care with dataset bias is more important than ever.

In summary, we cannot avoid bias, but we should consider the biases that are encoded by a particular dataset and consider whether measured improvements indicate a better fit to a dataset or a more general phenomenon. Towards this, we encourage experimentation in which training and test sets are drawn from different sources.

3. Image Features

Although there are many possible prediction tasks, the same basic features apply to all of them: color, gradients, histograms of gradients, edges, histograms of edges, position, and region shape. See Figure 5.3 for examples of several color spaces, gradient features, and edges.

3.1. Color

Color is predictive of material and lighting and is helpful for segmenting regions into separate objects and materials. By far, the greatest variation in color is due to changes in the luminance or brightness. For this reason, color spaces that decouple luminance, such as YCbCr, HSV, and CIELAB, may be preferable to RGB. **YCbCr** is quickly computed and commonly used in display electronics, such as televisions. **HSV**, which decomposes into hue, saturation, and value, is the most intuitive, though the angular measurement of hue (e.g., the maximum and minimum values are both red) can be a nuisance. **CIELAB** is designed to be perceptually uniform, meaning that small changes in color with equal Euclidean distance will be perceived by humans as having similar degrees of change.

Luminance encodes most of the useful information for scene understanding. People can understand grayscale images and movies without difficulty. From luminance alone, most changes in albedo are perceivable. Additionally, luminance encodes the shading information that is predictive of physical texture and overall shape. To better analyze shape, intensity values are typically not used directly. Instead, gradients and zero-sum filters, which better encode changes in shading, are used.

For categorization, color is often represented using averages or histograms. For example, the mean values of YCbCr indicate the overall brightness, the blueness, and the redness of a region. **Histograms** are estimates of the probability that a randomly selected pixel will have a particular color value. Histograms are computed by counting the number of times that each value is observed and dividing by the total value. It is possible to compute separate histograms for each channel or for all channels jointly. Commonly, the three color channels are quantized into a smaller number of values before counting them. One method is to divide the color space into bins (or cubes for three channels). For instance, the values with R$<$0.25, G$<$0.25, B$<$0.25 could be assigned the a value of 1, with all other possible RGB values assigned to one of 64 (4x4x4) discrete values. Then, these discrete values are counted. Another quantization method is clustering. Typically, a sampling of color values is clustered with K-means [**25**] to learn a set of cluster centers. Then, new values are assigned a number corresponding to the closest center. The K-means clustering aims to provide a quantization that minimizes the mean squared error of the reconstruction of the original values. The clustering approach is more computationally expensive but usually provides a better representation than simple binning for a fixed number of quantized values.

For segmentation, the difference of mean values or histograms is often used to measure region similarity. To compute the difference of histograms, the measures histogram intersection (HIN) and chi-squared (χ^2) are often used. HIN is faster to compute. χ^2 has attractive theoretical justification

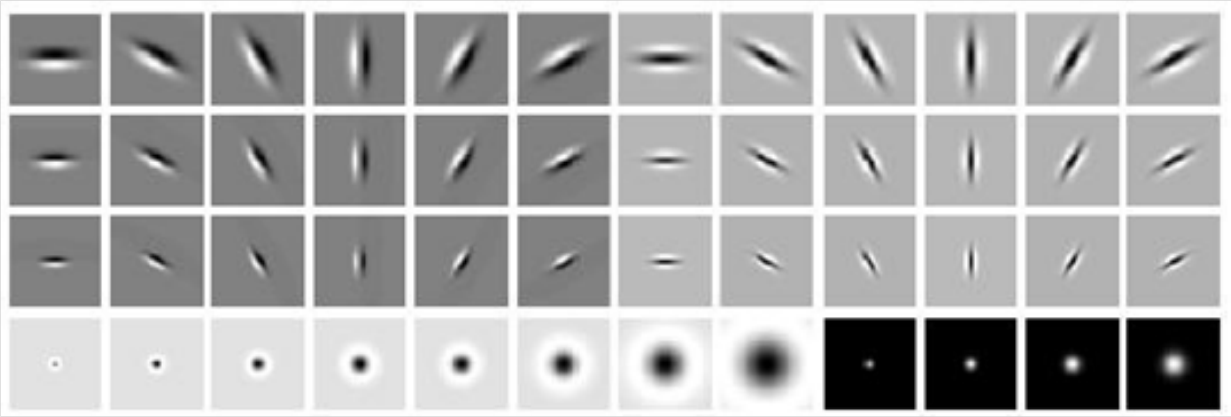

FIGURE 5.4. Leung-Malik filter bank for representing texture.

and sometimes leads to better performance. HIN or χ^2 are also often used as kernels for SVM classifiers with histogram features.

The gain a better intuition for color, we encourage students to play around with sample images, visualizing them using separate color channels in various color spaces. To compare effectiveness of quantization approaches, it may also be helpful to visualize reconstructions of the images after quantizing the colors.

3.2. Texture

Texture is predictive of material and material properties and is helpful for separating regions into different objects and materials. Texture is typically represented by computing statistics of responses to zero-mean filters, such as oriented edge filters, bar filters, or blob filters. Commonly used filter banks include those by Leung and Malik [**75**] (LM filter bank; see Figure 5.4) and Varma and Zisserman [**124**] (MR filter banks). Like color, the texture of a region is typically represented by computing the average magnitude of filter responses within the region or by computing histograms of quantized values of the filter responses. Histograms of quantized filter responses are often called "textons" [**75**].

3.3. Gradient-based

Local differences in intensity or gradients are predictive of shape, due to shading, and boundaries of objects and materials. Gradients may computed simply, with filters such as $[-1\ 1]$ or $[-1\ 0\ 1]$ for a horizontal gradient. Often, particularly for detecting important boundaries, the image is first filtered with a Gaussian filter. Gradients can be computed at multiple scales and orientations by

changing the bandwidth of the Gaussian pre-filter and by taking weighted averages of the vertical and horizontal filter responses (see the classic work on steerable filters by Freeman and Adelson [**34**] for details). The magnitude of the gradient between two regions is a strong indicator for the likelihood of an object or material boundary, a depth discontinuity, or a surface orientation discontinuity.

Gradients are also the basis of the most powerful features in object recognition. Oriented gradient values are used to compute many frequently used patch descriptors, such as SIFT [**79**] and HOG [**22**] (histogram of gradient). Most descriptors are computed by dividing a patch into different spatial bins (called cells), computing histograms within each spatial bin, and normalizing the values of the histograms. In SIFT, the descriptor is first registered to the dominant orientation so that an in-plane rotation has little effect, and normalization is achieved within each cell by division of total magnitude and thresholding. The HOG descriptor is sensitive to rotation (which is often beneficial for object categorization), and the values in each HOG cell are normalized several times based on the values in the surrounding cells.

HOG features were originally proposed for pedestrian detection [**22**] and have since been shown effective other object categories [**32**] and for a variety of other tasks, such as categorizing scenes and predicting the 3D orientation of materials. HOG features are insensitive to changes in brightness of the light source (due to use of the gradient) and small shifts in position, due to computation of histograms within spatial bins. They are also insensitive to overall changes in contrast, due to the normalization. HOG features are sensitive to shading patterns caused by surface shape and to changes in albedo, and they are sensitive to oriented textures which can predict surface orientation. Object parts, such as eyes, vertical legs, and wheels, often have distinctive HOG feature values. For object categorization, HOG features are often used directly at several fixed positions within a proposed object window. For other tasks, such as scene categorization, histograms of SIFT or HOG features are often computed, in the same way as with color and texture.

3.4. Interest Points and Bag of Words

The idea of detecting distinctive points was first developed for tracking [**44, 119**] and then for more general image matching [**107**] and object instance recognition [**78**]. Now, interest points and their descriptors, such as SIFT, are used as a general purpose region and image descriptor. After detecting interest points, or placing them at dense intervals along a grid [**89**], descriptors are computed and quantized. Then, histograms are computed for regions of interest or the image as a whole, depending on the task. Many, many papers have been written proposing variants on the descriptor, how many visual words to use, what points to sample [**89**], faster clustering

techniques [**87**], discriminative quantization techniques, and more. In most cases, simple k-means clustering of densely sampled SIFT or HOG descriptors will work well.

3.5. Image Position

Due to conventions of photography and the structure of our natural world, image position is often strongly predictive of surface orientation [**52**], material [**112**], and object category. Position can be defined within the image coordinates, or relative to an estimated horizon. Especially if defined in image coordinates, position may increase accuracy on the dataset used for training but lead to poorer generalization on other image collections. The position of a region can be encoded by a centroid, a bounding box, or percentiles of sorted positions (e.g., the 5th and 95th percentile of sorted vertical positions for a more robust top and bottom estimate).

3.6. Region Shape

Defying intuition and many efforts to devise good representations of shape, region shape tends to provide only a weak cue for categorization in natural images. In part, this is because it is difficult to obtain precise segmentations of objects and surfaces. Also, the 2D silhouette of an object often looks like an indistinct blob. Consider the silhouette of a duck or of a car. These shapes have iconic profiles, but if seen from the front or a three-quarters view, the silhouette will be indistinctive. One simple approach to shape-based categorization is to feed a binary mask into a classifier [**98**]. Another possibility is to characterize the shape with orientation, eccentricity, major axis length, minor axis length, and so on (see Matlab documentation for `regionprops` for a nice list of simple shape measures).

3.7. Perspective

Useful perspective cues include statistics of intersecting lines and detected vanishing points, histograms of edges that are oriented consistently with the vanishing points, and more general histograms of oriented straight edges [**54, 47, 71**]. Such features can provide powerful cues for surface orientation, although histograms of oriented gradient filters can often work nearly as well.