CHAPTER 17

Detecting Objects in Images

Chapter 16 described methods to classify images. When we assumed that the image contained a single, dominating object, these methods were capable of identifying that object. In this chapter, we describe methods that can detect objects. These methods all follow a surprisingly simple recipe—essentially, apply a classifier to subwindows of the image—which we describe with examples in Section 17.1. We then describe a more complex version of this recipe that applies to objects that can deform, or that have complex appearance (Section 17.2). Finally, we sketch the state of the art of object detection, giving pointers to available software and data (Section 17.3).

17.1 THE SLIDING WINDOW METHOD

Assume we are dealing with objects that have a relatively well-behaved appearance, and do not deform much. Then we can detect them with a very simple recipe. We build a dataset of labeled image windows of fixed size (say, $n \times m$). The examples labeled positive should contain large, centered instances of the object, and those labeled negative should not. We then train a classifier to tell these windows apart. We now pass every $n \times m$ window in the image to the classifier. Windows that the classifier labels positive contain the object, and those labeled negative do not. This is a search over location, which we could represent with the top left-hand corner of the window.

There are two subtleties to be careful about when applying this recipe. First, not all instances of an object will be the same size in the image. This means we need to search over scale as well. The easy way to do this is to prepare a Gaussian pyramid of the image (Section 4.7), and then search $n \times m$ windows in each layer of the pyramid. Searching an image whose edge lengths have been scaled by s for $n \times m$ windows is rather like searching the original image for $(sn) \times (sm)$ windows (the differences are in resolution, in ease of training, and in computation time).

The second subtlety is that some image windows overlap quite strongly. Each of a set of overlapping windows could contain all (or a substantial fraction of) the object. This means that each might be labeled positive by the classifier, meaning we would count the same object multiple times. This effect cannot be cured by passing to a bigger training set and producing a classifier that is so tightly tuned that it responds only when the object is exactly centered in the window. This is because it is hard to produce tightly tuned classifiers, and because we will never be able to place a window exactly around an object, so that a tightly tuned classifier will tend to behave badly. The usual strategy for managing this problem is *non-maximum suppression*. In this strategy, windows with a local maximum of the classifier response suppress nearby windows. We summarize the whole approach in Algorithm 17.1.

Train a classifier on $n \times m$ image windows. Positive examples contain the object and negative examples do not.
Choose a threshold t and steps Δx and Δy in the x and y directions

Construct an image pyramid.

For each level of the pyramid
  Apply the classifier to each $n \times m$ window, stepping by Δx and Δy, in this level to get a response strength c.
  If $c > t$
    Insert a pointer to the window into a ranked list $\mathcal{L}$, ranked by c.

For each window $\mathcal{W}$ in $\mathcal{L}$, starting with the strongest response
  Remove all windows $\mathcal{U} \neq \mathcal{W}$ that overlap $\mathcal{W}$ significantly,
    where the overlap is computed in the original image by expanding windows in coarser scales.

$\mathcal{L}$ is now the list of detected objects.

Algorithm 17.1: Sliding Window Detection.

The sliding window detection recipe is wholly generic and behaves very well in practice. Different applications require different choices of feature and sometimes benefit from different choices of feature. Notice that there is a subtle interaction between the size of the window, the steps Δx and Δy, and the classifier. For example, if we work with windows that tightly surround the object, then we might be able to use a classifier that is more tightly tuned, but we will have to use smaller steps and so look at more windows. If we use windows that are rather larger than the object, then we can look at fewer windows, but our ability to detect objects next to one another might be affected, as might our ability to localize the objects. Cross-validation is one way to make appropriate choices here. As a result, there is some variation in the appearance of the window caused by the fact our search is quantized in translation and scale; the training tricks in Section 15.3.1 are extremely useful for controlling this difficulty.

17.1.1 Face Detection

In frontal views at a fairly coarse scale, all faces look basically the same. There are bright regions on the forehead, the cheeks, and the nose, and dark regions around the eyes, the eyebrows, the base of the nose, and the mouth. This suggests approaching face finding as a search over all image windows of a fixed size for windows that look like a face. Larger or smaller faces can be found by searching coarser- or finer-scale images.

A face illuminated from the left looks different than a face illuminated from