

and Chellappa, 1988) operation reports the maximum output within a rectangular neighborhood. Other popular pooling functions include the average of a rectangular neighborhood, the  $L^2$  norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel.

In all cases, pooling helps to make the representation become approximately **invariant** to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. See figure 9.8 for an example of how this works. *Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.* For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face. In other contexts, it is more important to preserve the location of a feature. For example, if we want to find a corner defined by two edges meeting at a specific orientation, we need to preserve the location of the edges well enough to test whether they meet.

The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network.

Pooling over spatial regions produces invariance to translation, but if we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to (see figure 9.9).

Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced  $k$  pixels apart rather than 1 pixel apart. See figure 9.10 for an example. This improves the computational efficiency of the network because the next layer has roughly  $k$  times fewer inputs to process. When the number of parameters in the next layer is a function of its input size (such as when the next layer is fully connected and based on matrix multiplication) this reduction in the input size can also result in improved statistical efficiency and reduced memory requirements for storing the parameters.

For many tasks, pooling is essential for handling inputs of varying size. For example, if we want to classify images of variable size, the input to the classification layer must have a fixed size. This is usually accomplished by varying the size of an offset between pooling regions so that the classification layer always receives the same number of summary statistics regardless of the input size. For example, the final pooling layer of the network may be defined to output four sets of summary statistics, one for each quadrant of an image, regardless of the image size.