

mapping one image function to another image function, such that $I' = g(I)$ is the image function with $I'(x, y) = I(x - 1, y)$. This shifts every pixel of I one unit to the right. If we apply this transformation to I , then apply convolution, the result will be the same as if we applied convolution to I' , then applied the transformation g to the output. When processing time series data, this means that convolution produces a sort of timeline that shows when different features appear in the input. If we move an event later in time in the input, the exact same representation of it will appear in the output, just later in time. Similarly with images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output. This is useful for when we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations. For example, when processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image. In some cases, we may not wish to share parameters across the entire image. For example, if we are processing images that are cropped to be centered on an individual's face, we probably want to extract different features at different locations—the part of the network processing the top of the face needs to look for eyebrows, while the part of the network processing the bottom of the face needs to look for a chin.

Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations.

Finally, some kinds of data cannot be processed by neural networks defined by matrix multiplication with a fixed-shape matrix. Convolution enables processing of some of these kinds of data. We discuss this further in section 9.7.

9.3 Pooling

A typical layer of a convolutional network consists of three stages (see figure 9.7). In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the **detector** stage. In the third stage, we use a **pooling function** to modify the output of the layer further.

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the **max pooling** (Zhou