

It is also possible to connect a large input layer to a much smaller layer by spacing out the receptive fields, as shown in [Figure 13-4](#). The distance between two consecutive receptive fields is called the *stride*. In the diagram, a  $5 \times 7$  input layer (plus zero padding) is connected to a  $3 \times 4$  layer, using  $3 \times 3$  receptive fields and a stride of 2 (in this example the stride is the same in both directions, but it does not have to be so). A neuron located in row  $i$ , column  $j$  in the upper layer is connected to the outputs of the neurons in the previous layer located in rows  $i \times s_h$  to  $i \times s_h + f_h - 1$ , columns  $j \times s_w$  to  $j \times s_w + f_w - 1$ , where  $s_h$  and  $s_w$  are the vertical and horizontal strides.

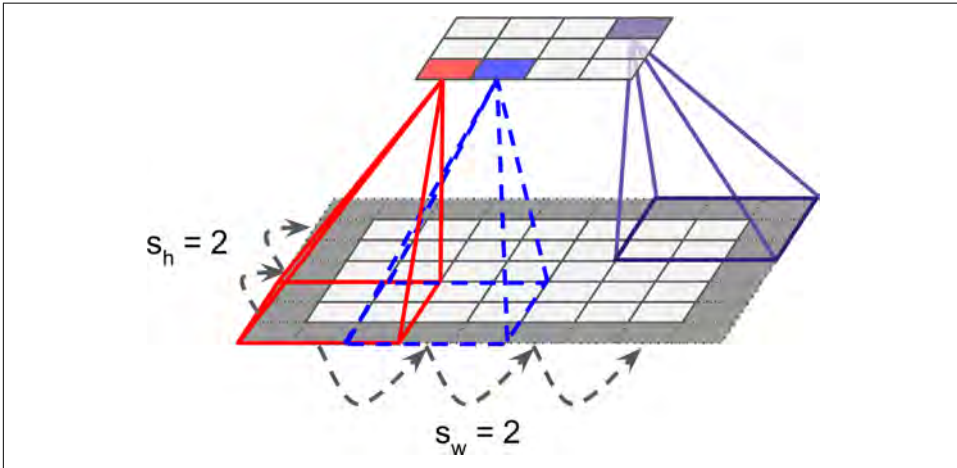


Figure 13-4. Reducing dimensionality using a stride

## Filters

A neuron's weights can be represented as a small image the size of the receptive field. For example, [Figure 13-5](#) shows two possible sets of weights, called *filters* (or *convolution kernels*). The first one is represented as a black square with a vertical white line in the middle (it is a  $7 \times 7$  matrix full of 0s except for the central column, which is full of 1s); neurons using these weights will ignore everything in their receptive field except for the central vertical line (since all inputs will get multiplied by 0, except for the ones located in the central vertical line). The second filter is a black square with a horizontal white line in the middle. Once again, neurons using these weights will ignore everything in their receptive field except for the central horizontal line.

Now if all neurons in a layer use the same vertical line filter (and the same bias term), and you feed the network the input image shown in [Figure 13-5](#) (bottom image), the layer will output the top-left image. Notice that the vertical white lines get enhanced while the rest gets blurred. Similarly, the upper-right image is what you get if all neurons use the horizontal line filter; notice that the horizontal white lines get enhanced while the rest is blurred out. Thus, a layer full of neurons using the same filter gives

you a *feature map*, which highlights the areas in an image that are most similar to the filter. During training, a CNN finds the most useful filters for its task, and it learns to combine them into more complex patterns (e.g., a cross is an area in an image where both the vertical filter and the horizontal filter are active).

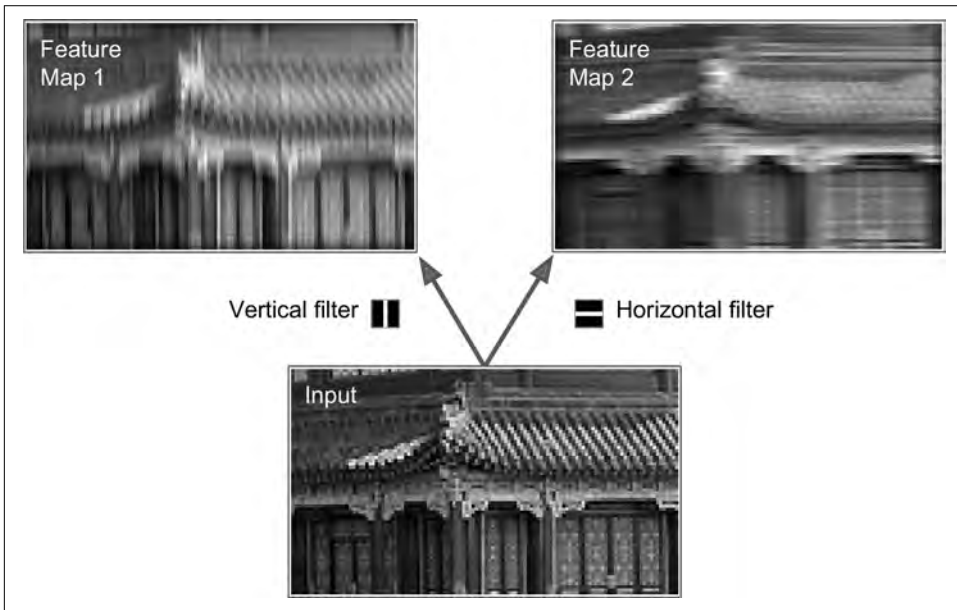


Figure 13-5. Applying two different filters to get two feature maps

## Stacking Multiple Feature Maps

Up to now, for simplicity, we have represented each convolutional layer as a thin 2D layer, but in reality it is composed of several feature maps of equal sizes, so it is more accurately represented in 3D (see [Figure 13-6](#)). Within one feature map, all neurons share the same parameters (weights and bias term), but different feature maps may have different parameters. A neuron's receptive field is the same as described earlier, but it extends across all the previous layers' feature maps. In short, a convolutional layer simultaneously applies multiple filters to its inputs, making it capable of detecting multiple features anywhere in its inputs.



The fact that all neurons in a feature map share the same parameters dramatically reduces the number of parameters in the model, but most importantly it means that once the CNN has learned to recognize a pattern in one location, it can recognize it in any other location. In contrast, once a regular DNN has learned to recognize a pattern in one location, it can recognize it only in that particular location.