

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).

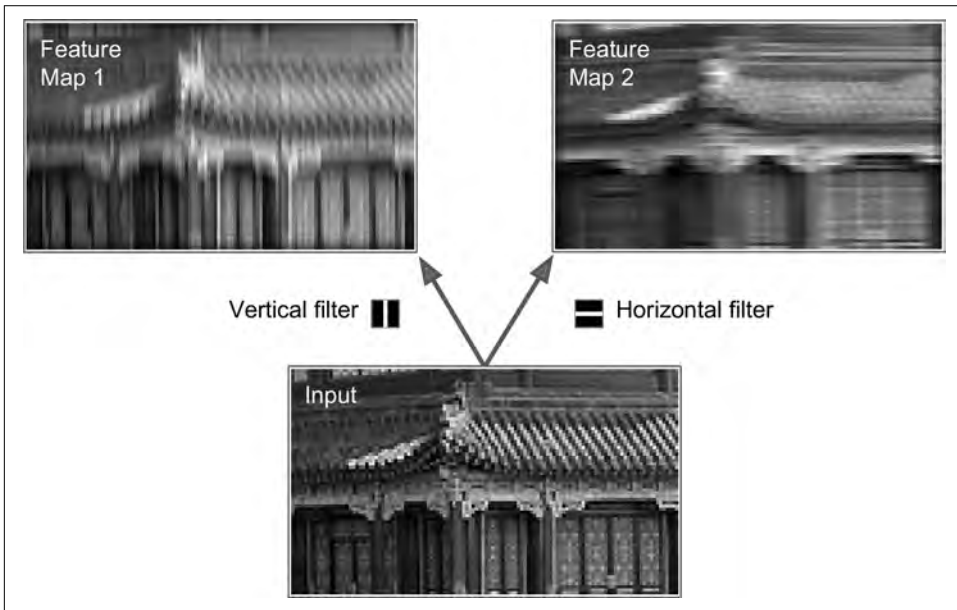


Figure 13-5. Applying two different filters to get two feature maps

## 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.

Moreover, input images are also composed of multiple sublayers: one per *color channel*. There are typically three: red, green, and blue (RGB). Grayscale images have just one channel, but some images may have much more—for example, satellite images that capture extra light frequencies (such as infrared).

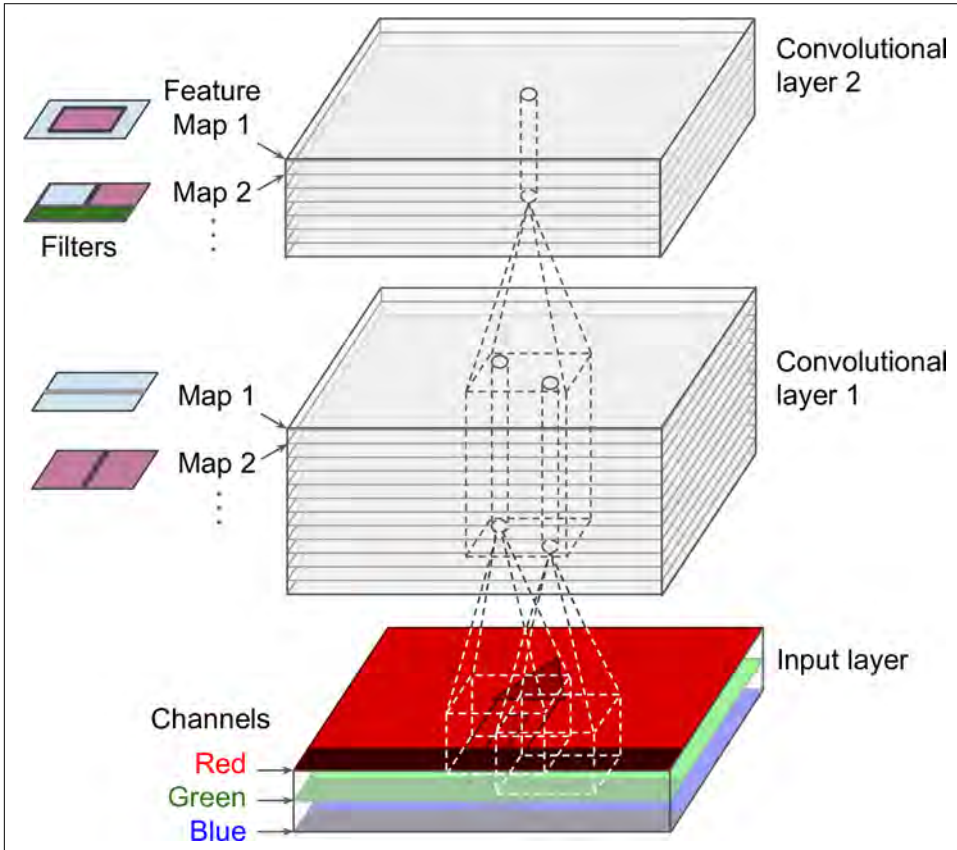


Figure 13-6. Convolution layers with multiple feature maps, and images with three channels

Specifically, a neuron located in row  $i$ , column  $j$  of the feature map  $k$  in a given convolutional layer  $l$  is connected to the outputs of the neurons in the previous layer  $l - 1$ , located in rows  $i \times s_w$  to  $i \times s_w + f_w - 1$  and columns  $j \times s_h$  to  $j \times s_h + f_h - 1$ , across all feature maps (in layer  $l - 1$ ). Note that all neurons located in the same row  $i$  and column  $j$  but in different feature maps are connected to the outputs of the exact same neurons in the previous layer.

**Equation 13-1** summarizes the preceding explanations in one big mathematical equation: it shows how to compute the output of a given neuron in a convolutional layer.

It is a bit ugly due to all the different indices, but all it does is calculate the weighted sum of all the inputs, plus the bias term.

*Equation 13-1. Computing the output of a neuron in a convolutional layer*

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f_{n'}} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \quad \begin{cases} i' = u \cdot s_h + f_h - 1 \\ j' = v \cdot s_w + f_w - 1 \end{cases}$$

- $z_{i,j,k}$  is the output of the neuron located in row  $i$ , column  $j$  in feature map  $k$  of the convolutional layer (layer  $l$ ).
- As explained earlier,  $s_h$  and  $s_w$  are the vertical and horizontal strides,  $f_h$  and  $f_w$  are the height and width of the receptive field, and  $f_{n'}$  is the number of feature maps in the previous layer (layer  $l - 1$ ).
- $x_{i',j',k'}$  is the output of the neuron located in layer  $l - 1$ , row  $i'$ , column  $j'$ , feature map  $k'$  (or channel  $k'$  if the previous layer is the input layer).
- $b_k$  is the bias term for feature map  $k$  (in layer  $l$ ). You can think of it as a knob that tweaks the overall brightness of the feature map  $k$ .
- $w_{u,v,k',k}$  is the connection weight between any neuron in feature map  $k$  of the layer  $l$  and its input located at row  $u$ , column  $v$  (relative to the neuron's receptive field), and feature map  $k'$ .

## TensorFlow Implementation

In TensorFlow, each input image is typically represented as a 3D tensor of shape [height, width, channels]. A mini-batch is represented as a 4D tensor of shape [mini-batch size, height, width, channels]. The weights of a convolutional layer are represented as a 4D tensor of shape  $[f_h, f_w, f_n, f_{n'}]$ . The bias terms of a convolutional layer are simply represented as a 1D tensor of shape  $[f_n]$ .

Let's look at a simple example. The following code loads two sample images, using Scikit-Learn's `load_sample_images()` (which loads two color images, one of a Chinese temple, and the other of a flower). Then it creates two  $7 \times 7$  filters (one with a vertical white line in the middle, and the other with a horizontal white line), and applies them to both images using a convolutional layer built using TensorFlow's `conv2d()` function (with zero padding and a stride of 2). Finally, it plots one of the resulting feature maps (similar to the top-right image in [Figure 13-5](#)).