

Additionally, the input is usually not just a grid of real values. Rather, it is a grid of vector-valued observations. For example, a color image has a red, green and blue intensity at each pixel. In a multilayer convolutional network, the input to the second layer is the output of the first layer, which usually has the output of many different convolutions at each position. When working with images, we usually think of the input and output of the convolution as being 3-D tensors, with one index into the different channels and two indices into the spatial coordinates of each channel. Software implementations usually work in batch mode, so they will actually use 4-D tensors, with the fourth axis indexing different examples in the batch, but we will omit the batch axis in our description here for simplicity.

Because convolutional networks usually use multi-channel convolution, the linear operations they are based on are not guaranteed to be commutative, even if kernel-flipping is used. These multi-channel operations are only commutative if each operation has the same number of output channels as input channels.

Assume we have a 4-D kernel tensor \mathbf{K} with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel i of the output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit. Assume our input consists of observed data \mathbf{V} with element $V_{i,j,k}$ giving the value of the input unit within channel i at row j and column k . Assume our output consists of \mathbf{Z} with the same format as \mathbf{V} . If \mathbf{Z} is produced by convolving \mathbf{K} across \mathbf{V} without flipping \mathbf{K} , then

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n} \quad (9.7)$$

where the summation over l , m and n is over all values for which the tensor indexing operations inside the summation is valid. In linear algebra notation, we index into arrays using a 1 for the first entry. This necessitates the -1 in the above formula. Programming languages such as C and Python index starting from 0, rendering the above expression even simpler.

We may want to skip over some positions of the kernel in order to reduce the computational cost (at the expense of not extracting our features as finely). We can think of this as downsampling the output of the full convolution function. If we want to sample only every s pixels in each direction in the output, then we can define a downsampled convolution function c such that

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} [V_{l,(j-1) \times s + m, (k-1) \times s + n} K_{i,l,m,n}] \cdot \quad (9.8)$$

We refer to s as the **stride** of this downsampled convolution. It is also possible