Convolution operation

Introduction

The intention of this work is not to explain how convolution and pooling operations work. There are a lot of incredible blogs on those, I mention here the two most intuitive in my opinion.

The intention is to make familiar with how I use to represent those operations in the rest of the works on this blog to ease the visualization of such operations.

[1] – Andrey Karpathy explanation on Official Stanford CS231 Course on CNNs for Visual Recognition

[2] – Dumoulin and Visin paper on Convolution and Pooling Arithmetic for Deep Learning

[3] – Jiaxin Wu perfect detailed work on the math behind convolutions and pooling

Convolution

Let's start with the simplest case. The first thing to have in mind is that an input volume is normally convolved with ***several different kernels***. Why? Because we expect each different kernel ***to extract different features from the input image***. Now we will see only 1 filter to the expand to the feature maps space.

Simple Convolution – 1 filter

The convolution operation simply consists on passing a ***kernel over an input volume***, which is the image. On this pass, the values of the image that matches the size of the kernel, ***perform a matrix multiplication between them two***, to provide the value of 1 cell on the output volume. In the Figure 1, the first shaded cell in the output will be the result of the matrix multiplication of the blue shaded matrix in the input volume with the convolutional kernel in yellow.

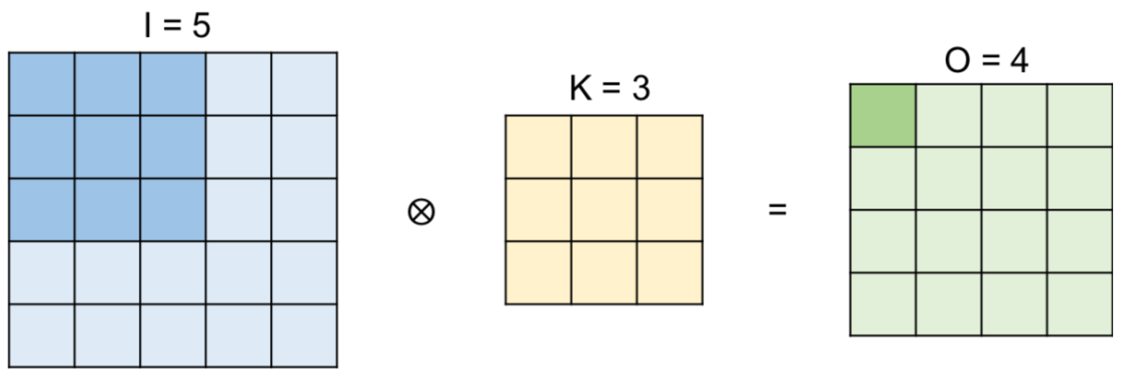

Figure 1. Convolution – 1 kernel, stride 1, no padding

Stride

The kernel then slides in the right direction until it reaches the last position. If we slide 1 by 1, called **stride** of 1, the total positions that the kernel can take is 4. This determines the dimension of the output volume.

If the stride is increased to 2, then it could only make 2 positions, as shown in Figure 2. The first position will be the same shaded area as in Figure 1, and the second position the shown in Figure 2.

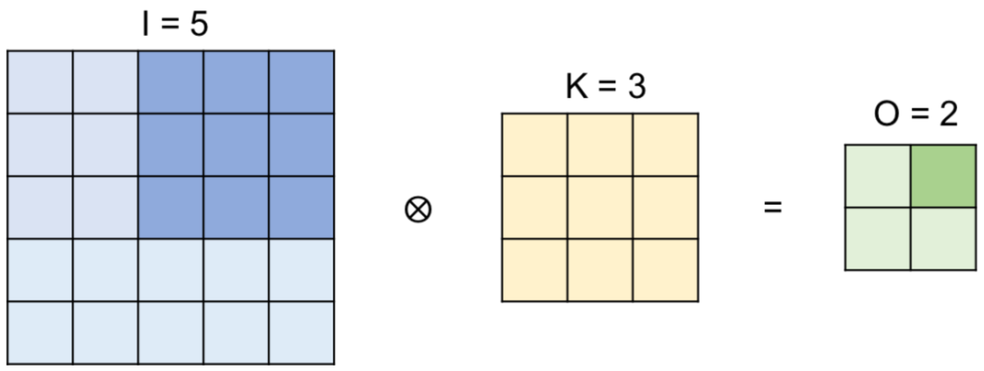

Figure 2. Convolution – 1 kernel, stride 2, no padding

There will be many occasions where we would like the output volume to have the same size as the input volume. Why so? Well, normally convolutional neural networks tend to be very deep. Therefore, we need not to decrease the volume at every convolutional layer, because we will end up with very small volumes very soon, not being able to capture the features we are interested in.

What can we do to achieve this?

Padding

Padding is a technique that simply adds zeros to the margin of the image to increase its size. More precisely, the padding required to achieve the same volume on both side of the convolution is intuitively called ***Same Padding***.

Figure 3 represents this. The stride is equal to one, but with the padding, the input gets to be 7 instead of the original 5, leading to an output size of 5.

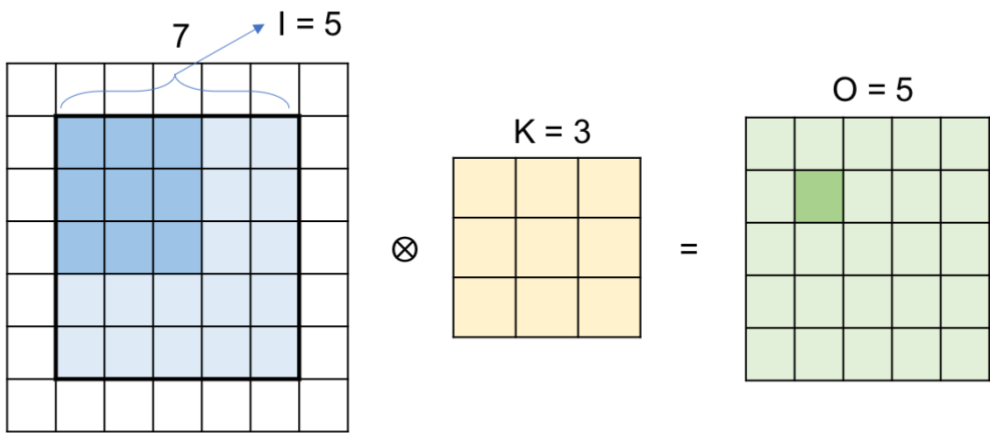

Figure 3. Convolution – 1 kernel, stride 1, padding 1

We can summarize the behavior of the output size with:

$$O = \frac{I + 2P - K}{S} + 1$$

Feature maps

Let’s take a look at Figure 4 for a better idea on what the kernel is actually doing on the image to expand later to several kernels.

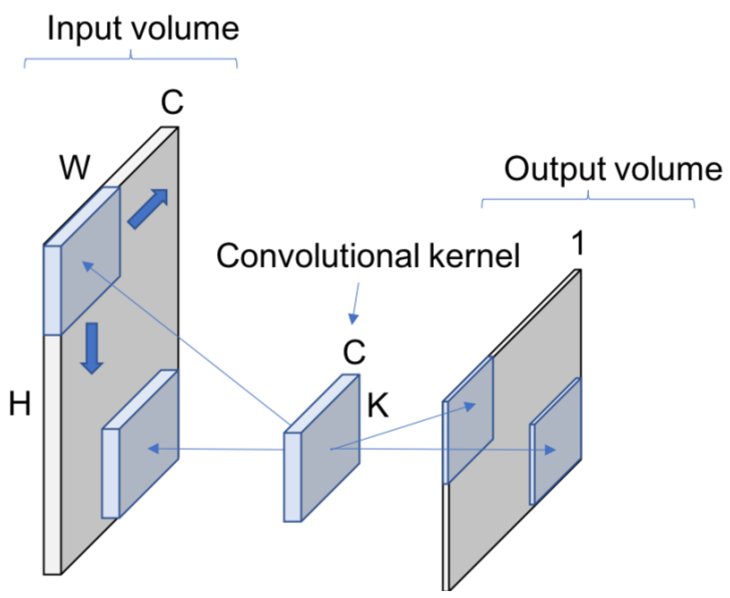

Figure 4. Convolution 2D with 1 Kernel

The main idea that you need to see is the 2D after the convolution in the name on the operation. We normally use kernels which channels’ dimension match the input volume’s one. Therefore, as can be appreciated in the Figure 4, 1 convolutional kernel will lead to a 1D plane output volume. If the size of the kernel was smaller, then we could move the kernel through the volume in the channel dimension, see Figure 5; and perform a 3D convolution operation.

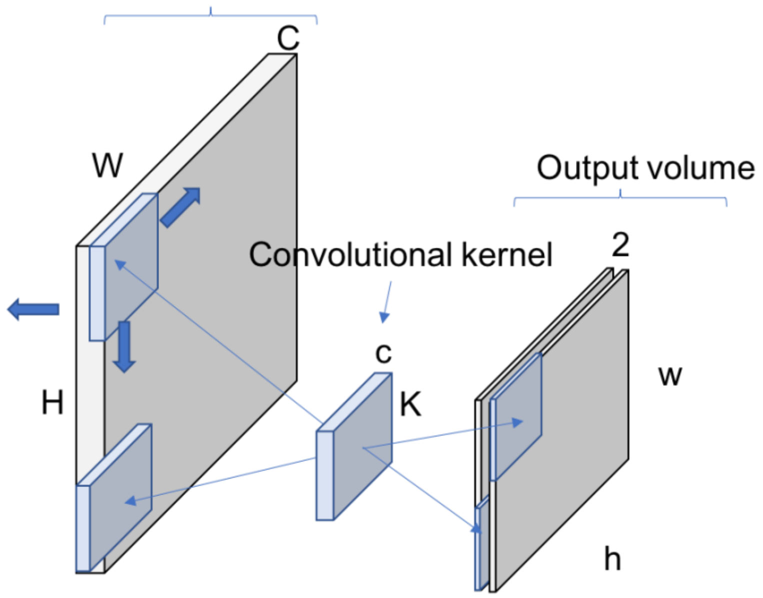

Figure 5. 3D Convolution

Convolution with multiple filters

When we apply several convolutional kernels, each of them will stack another (and different) output volume, with different features activated. This is the reason why some kernels are called edge detectors, corner detectors… because the values on the kernel allow to capture the features that are represented by and edge, corner… This can be seen in Figure 6, where ***each kernel is leading to one feature map in the output volume***. This is why the number of filters determines the channel size dimension when you are implementing your convolutions in code.

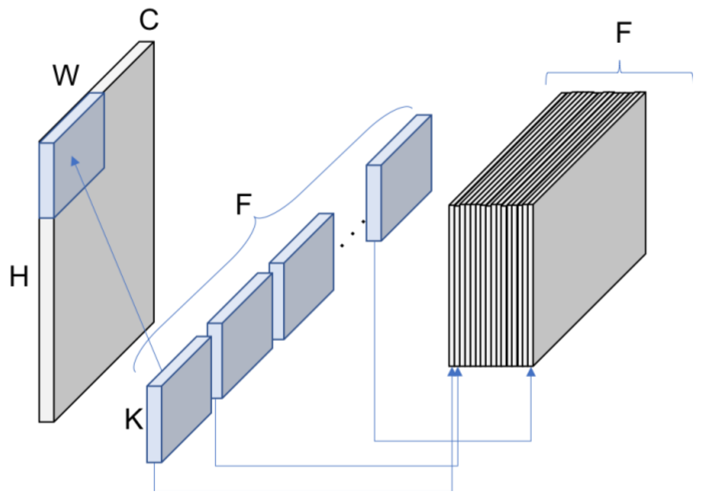

Figure 6. Convolution 2D with several kernels

Different representation

Since we have declared that the channel size of the kernels will be the same as the input volume, we can get rid of that dimension (C) in the draws and represent the feature maps dimension (F) as the depth of the filters. This will look like this:

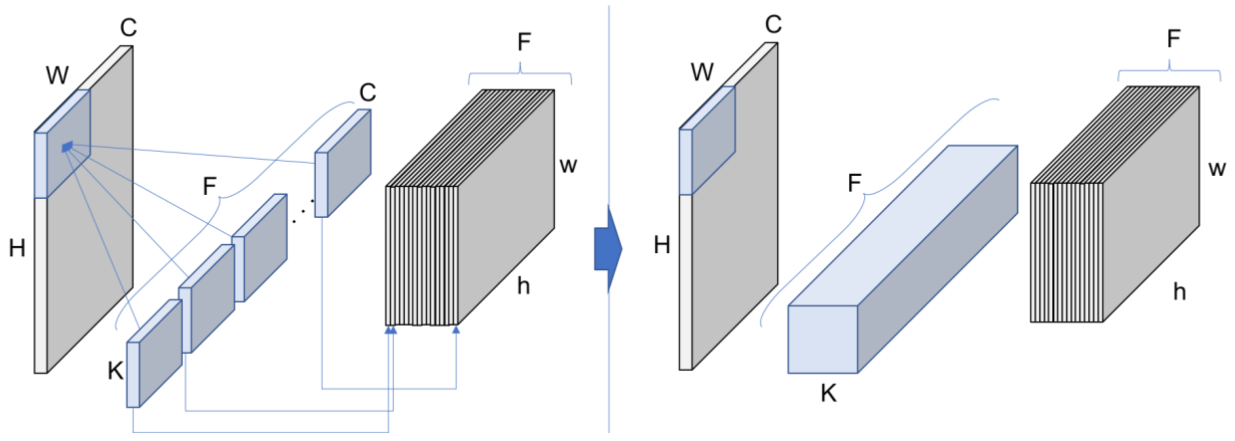

Figure 7. Different representation for convolutional kernels