Open in app Get started

Published in Towards Data Science · Follow

You have **2** free member-only stories left this month. Sign up for Medium and get an extra one

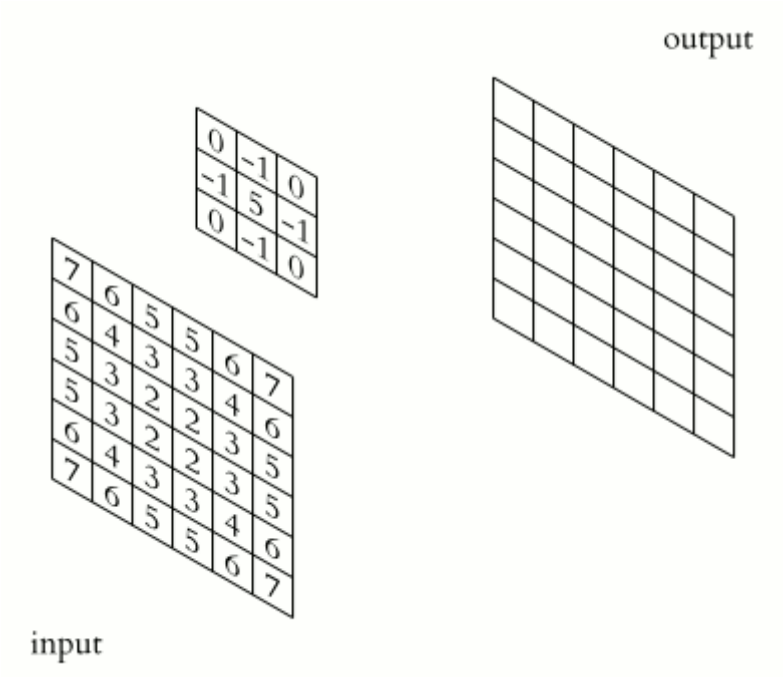
Prakhar Ganesh · Follow
Oct 18, 2019 · 7 min read ★

Types of Convolution Kernels : Simplified

An intuitive introduction to different variations of the glamorous CNN layer

Just a brief intro

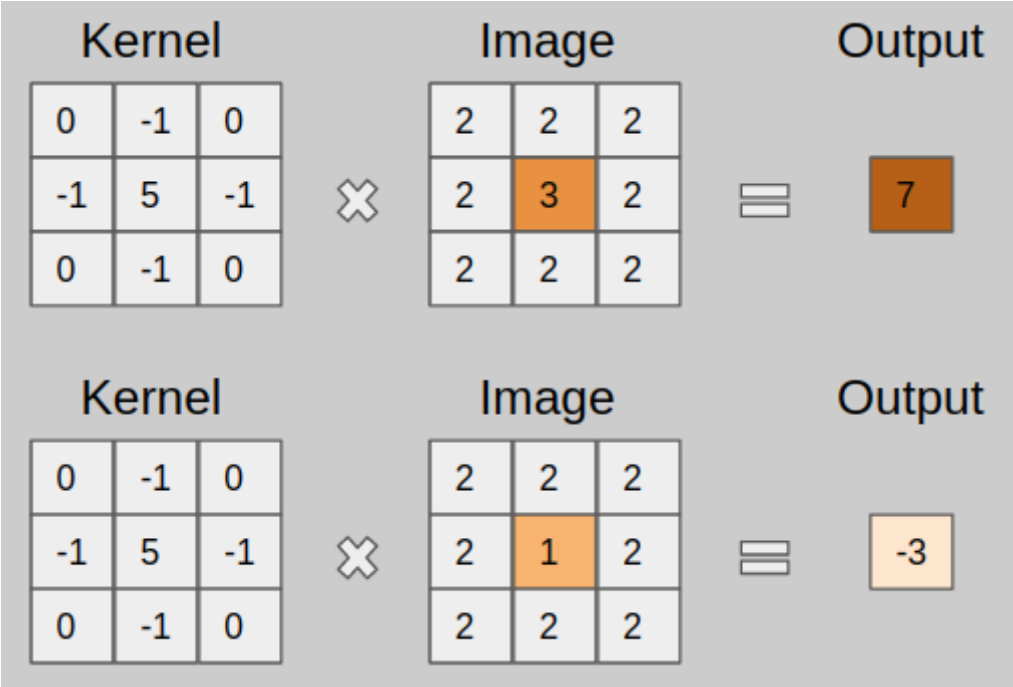
Convolution is using a 'kernel' to extract certain 'features' from an input image. Let me explain. A kernel is a matrix, which is slid across the image and multiplied with the input such that the output is enhanced in a certain desirable manner. Watch this in action below.



For example, the kernel used above is useful for sharpening the image. But what is so

Open in app Get started

2*-1 + 2*-1 = -3. The value 1 got decreased to -3. Clearly, the contrast between 3 and 1 is increased to 7 and -3, which will in turn sharpen the image.

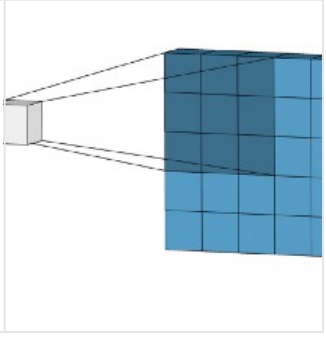


Instead of using manually made kernels for feature extraction, through Deep CNNs we can learn these kernel values which can extract latent features. For further reading into working of conventional CNNs, I would suggest this blog.

Intuitively Understanding Convolutions for Deep Learning

Exploring the strong visual hierarchies that makes them work

towardsdatascience.com



Kernel vs Filter

Before we dive into it, I just want to make the distinction between the terms ‘kernel’ and ‘filter’ very clear because I have seen a lot of people use them interchangeably. A kernel is, as described earlier, a matrix of weights which are multiplied with the input to extract

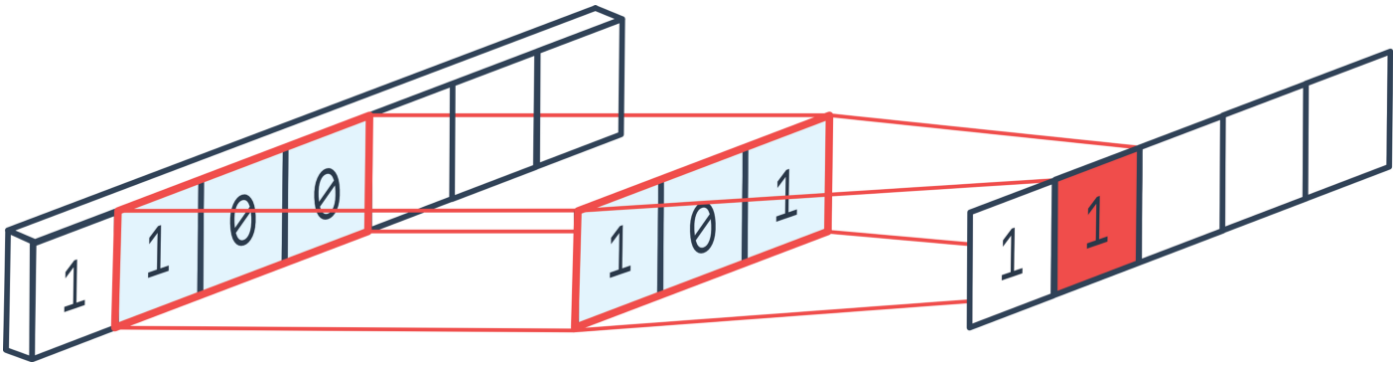
Open in app Get started

A filter however is a concatenation of multiple kernels, each kernel assigned to a particular channel of the input. Filters are always one dimension more than the kernels. For example, in 2D convolutions, filters are 3D matrices (which is essentially a concatenation of 2D matrices i.e. the kernels). So for a CNN layer with kernel dimensions h*w and input channels k, the filter dimensions are k*h*w.

A common convolution layer actually consist of multiple such filters. *For the sake of simplicity in the discussion to follow, assume the presence of only one filter unless specified, since the same behavior is replicated across all the filters.*

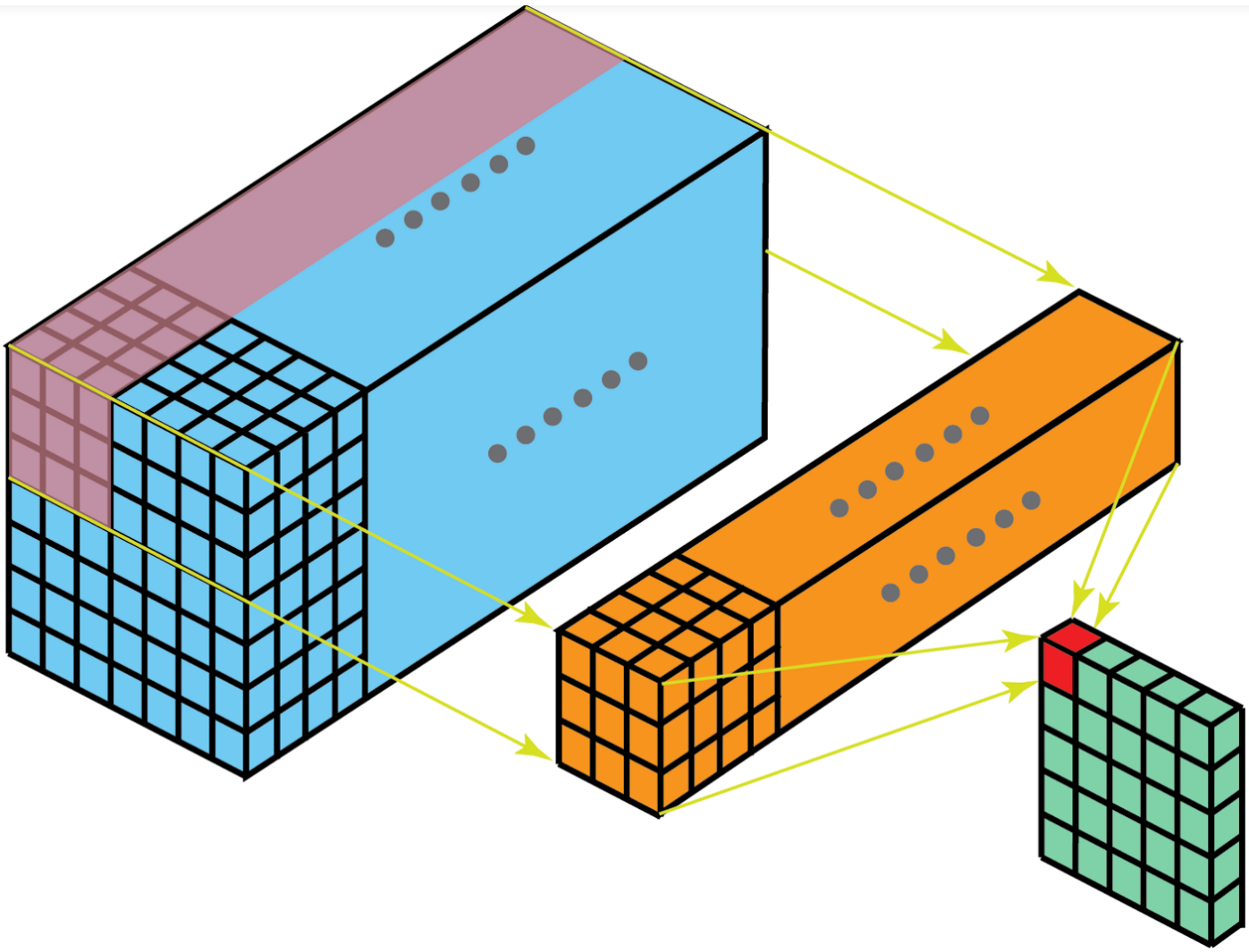
1D, 2D and 3D Convolutions

1D convolutions are commonly used for time series data analysis (since the input in such cases is 1D). As mentioned earlier, the 1D data input can have multiple channels. The filter can move in one direction only, and thus the output is 1D. See below an example of single channel 1D convolution.



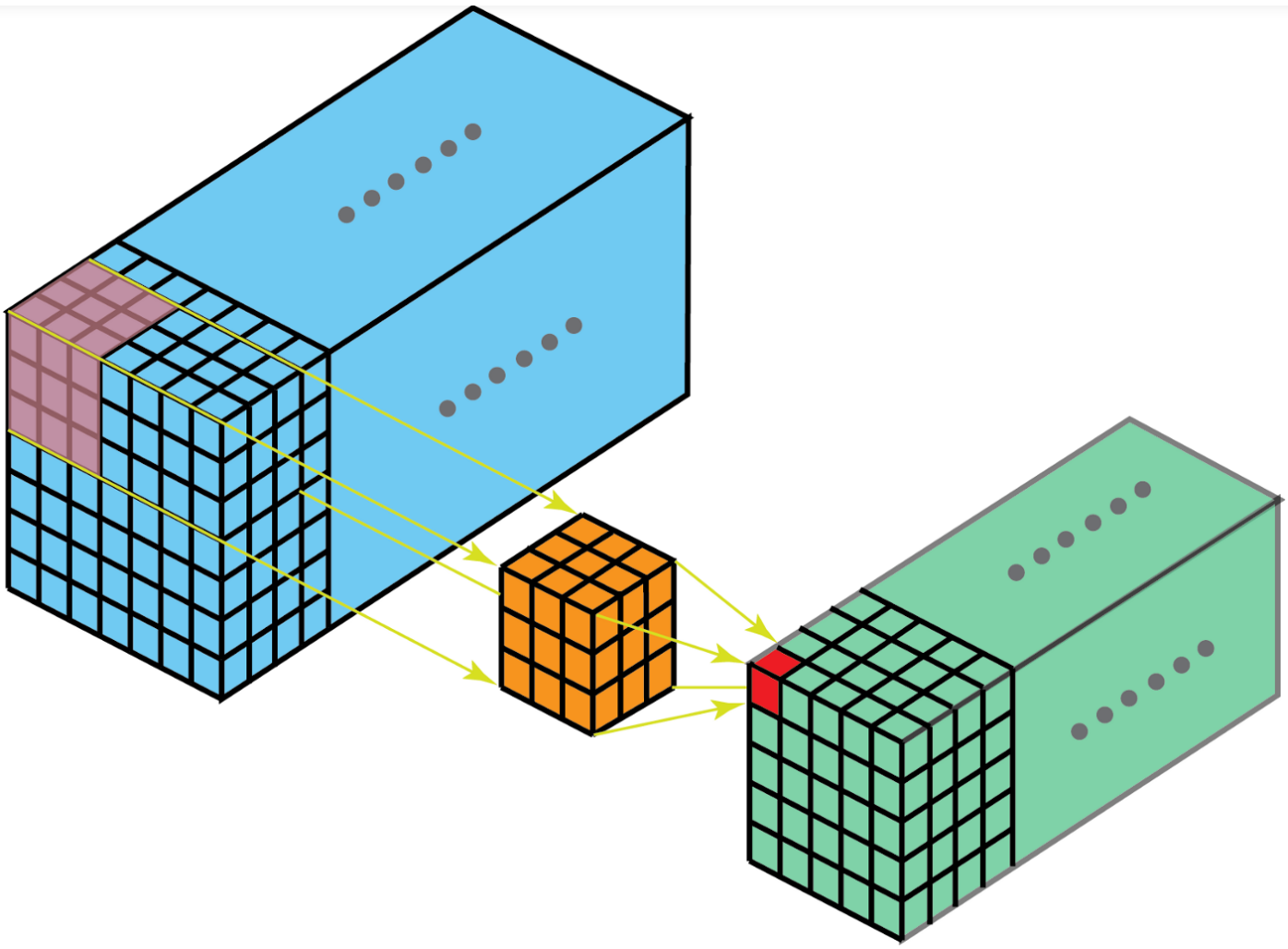
We already saw an example of single channel 2D convolution at the start of the post, so let's visualize a multi channel 2D convolution and try to wrap our heads around it. In the diagram below, the kernel dimensions are 3*3 and there are multiple such kernels in the filter (marked yellow). This is because there are multiple channels in the input (marked blue) and we have one kernel corresponding to every channel in the input. Clearly, here the filter can move in 2 directions and thus the final output is 2D. 2D convolutions are the most common convolutions, and are heavily used in Computer Vision.

Open in app Get started



It is difficult to visualize a 3D filter (since it's a 4D dimensional matrix), so we will discuss single channel 3D convolution here. As you can see from the image below, in 3D convolutions, a kernel can move in 3 directions and thus the output obtained is also 3D.

Open in app Get started

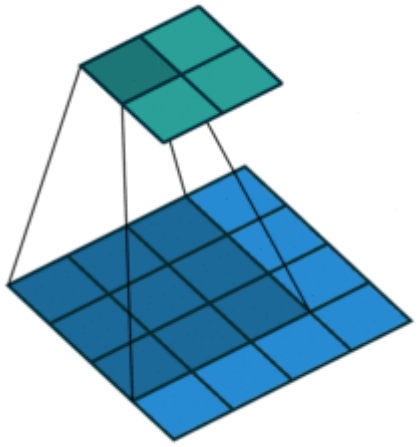


Most of the work done in modifying and customizing CNN layers have been focused towards 2D convolutions only and so from this point forward I will only be discussing these variations in context of 2D convolutions.

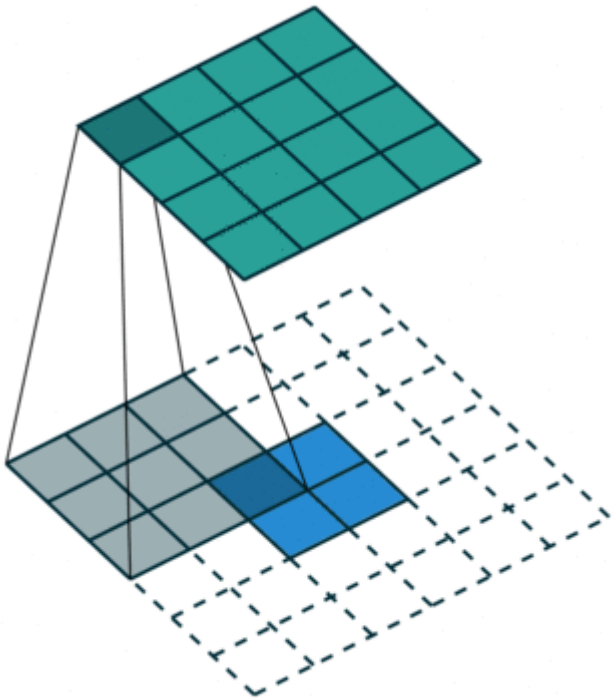
Transposed Convolution (Deconvolution)

The GIF below nicely captures how a 2D convolution decreases the dimensions of the input. But sometimes we need to do input processing such as to increase it's dimensions (also called 'upsampling').

Open in app Get started

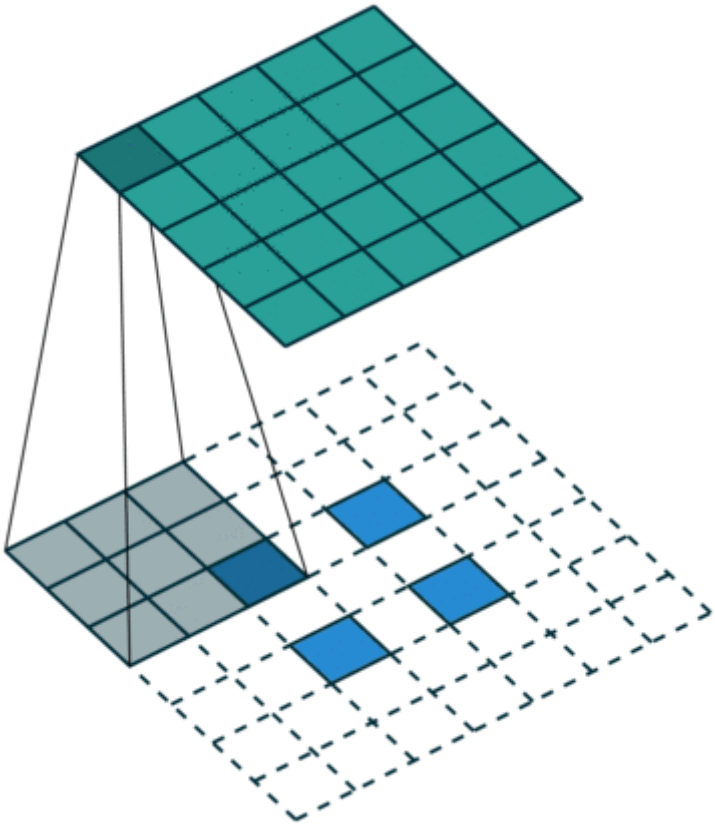


To achieve this using convolutions, we use a modification known as transposed convolution or deconvolution (although it is not truly 'reversing' a convolution operation, so a lot of people don't prefer to use this term). The dotted blocks in the GIF below represent padding.



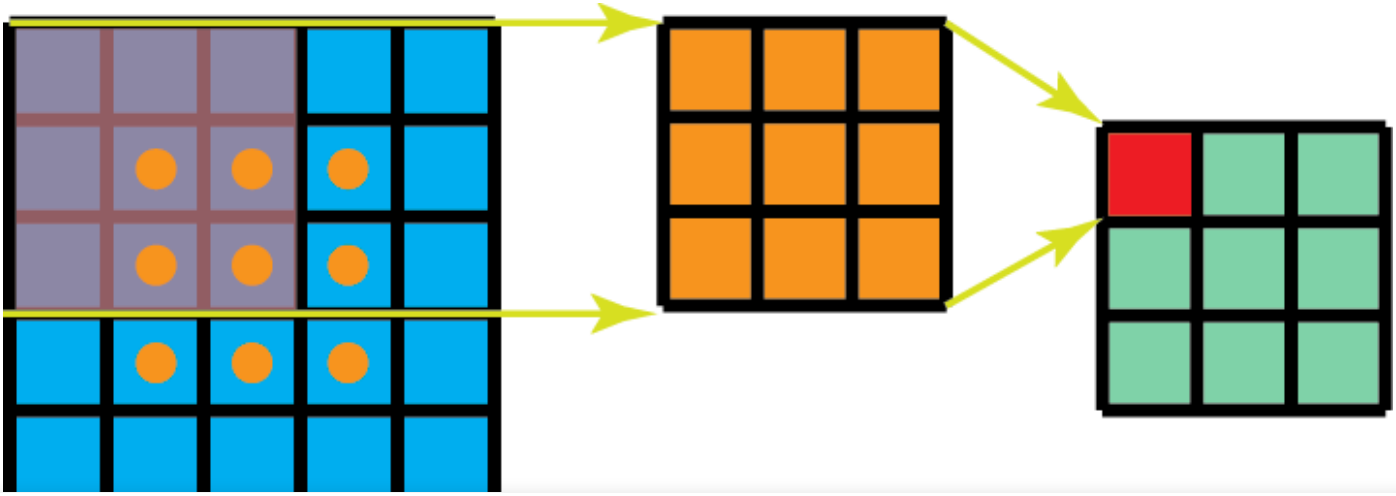
I think these animations give a good intuition of how different up-sampled outputs can be

Open in app Get started

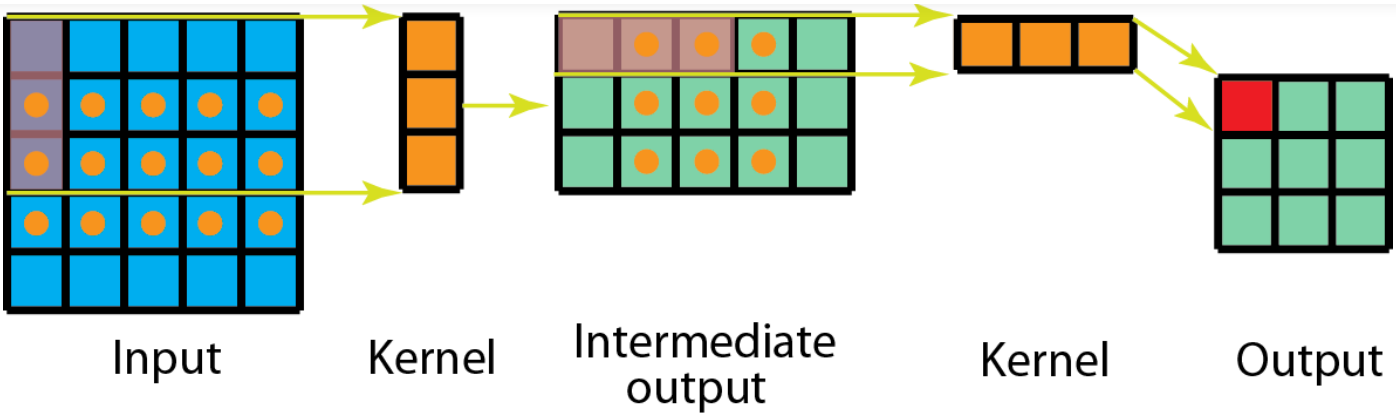


Separable Convolution

Separable Convolution refers to breaking down the convolution kernel into lower dimension kernels. Separable convolutions are of 2 major types. First are spatially separable convolutions, see below for example.

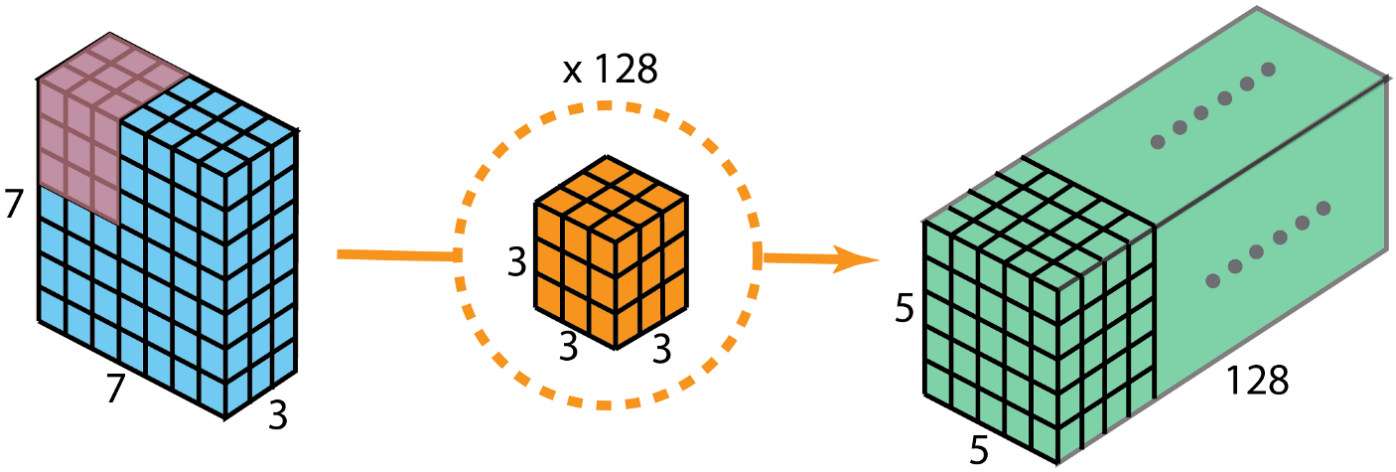


Open in app Get started

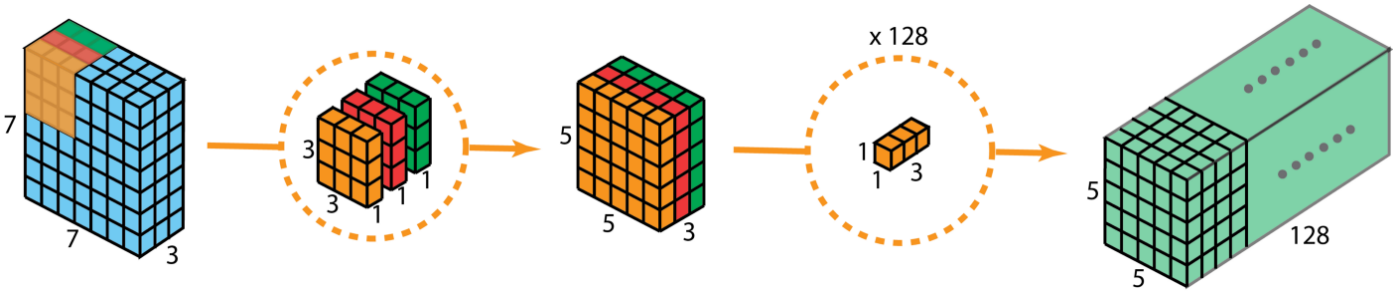


Spatially separable 2D convolution

However, spatially separable convolutions are not that common in Deep Learning. On the other hand, Depthwise separable convolutions are widely used in light weight CNN models and provide really good performances. See below for example.



A standard 2D convolution with 3 input channels and 128 filters



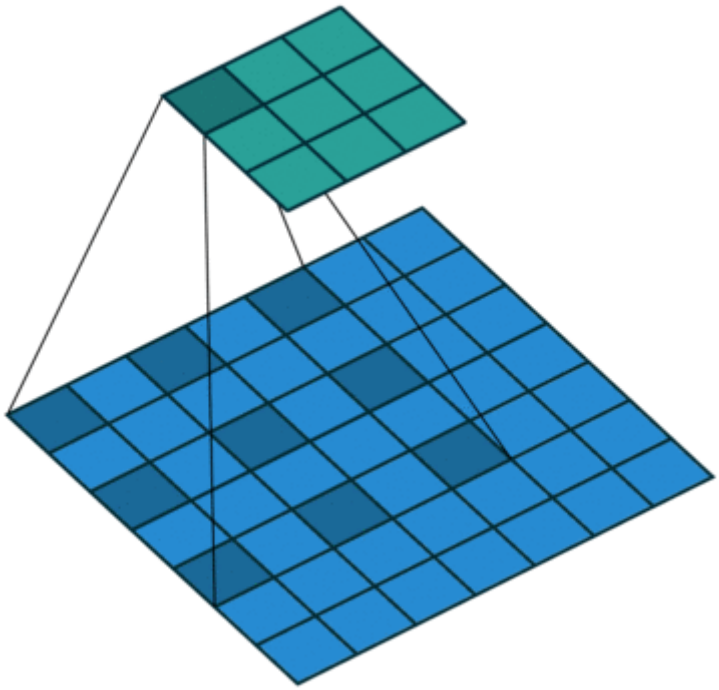
Depthwise separable 2D convolution which first processes each channel separately and then applies inter-

Open in app Get started

and tremendous size of Deep Learning networks that we have today, being able to provide similar performances with lower number of parameters is definitely a requirement.

Dilated (Atrous) Convolution

As you have seen in all the convolution layers above (without exception) that they process all the neighboring values together. However, sometimes it might be in the best interest of the pipeline to skip certain input values and this is how dilated convolutions (also called atrous convolutions) were introduced. Such a modification allows the kernel to increase it's *range of view,* without increasing the number of parameters.

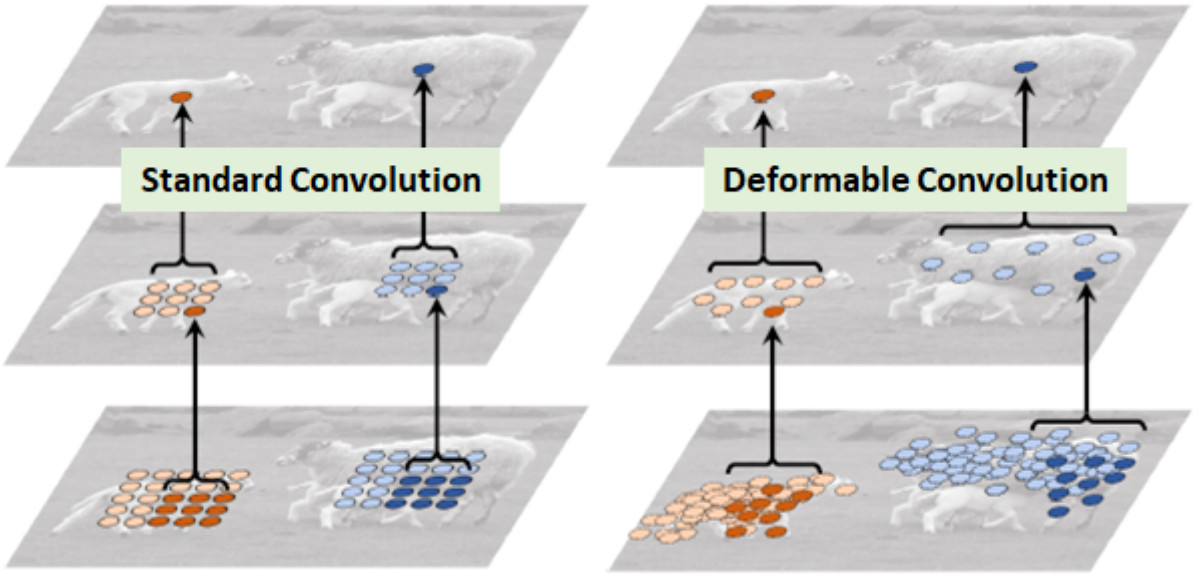


Clearly one can notice from the animation above, that the kernel is able to process a wider neighborhood with those same 9 parameters as earlier. This also means loss in information because of not being able to process the fine-grained information (since it is skipping certain values). However, the overall effect seems to be positive in certain applications.

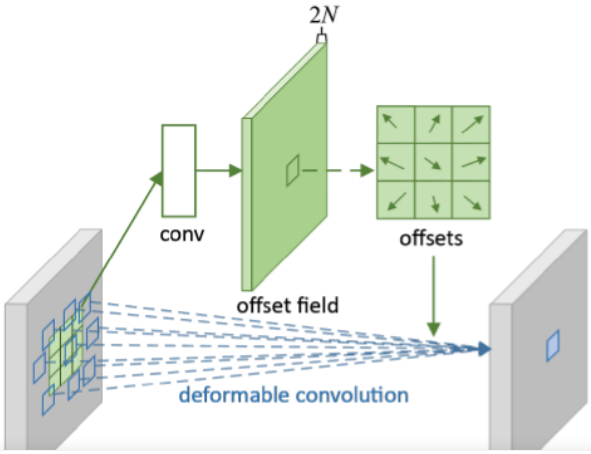
Deformable convolution

Open in app Get started

in itself was learnable? This is the core idea behind the introduction of deformable convolutions.



The implementation of a Deformable Convolution is actually very straightforward. Every kernel is actually represented with two different matrices. The first branch learns to predict the 'offset' from the origin. This offset is an indication of what inputs around the origin will be processed. Since each offset is predicted independently, they don't need to form any rigid shape between themselves, thus allowing the deformable nature. The second branch is simply the convolution branch whose input is now the values at these offsets.



Regular convolution

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n)$$

Deformable convolution

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)$$

where $\Delta\mathbf{p}_n$ is generated by a sibling branch of regular convolution

Open in app Get started

There have been multiple variations of CNN layers which have been used independently or in combination with each other to create successful and complex architectures. Each variation was born out of an intuition of how feature extraction should work. Thus I believe that while these Deep CNN networks learn weights that we cannot explain, the intuitions involved in forming them is very important to their performance and further work in that direction is important for success of highly complex CNNs.

. . .

This blog is a part of an effort to create simplified introductions to the field of Machine Learning. Follow the complete series here

Machine Learning : Simplified

Know it before you dive in

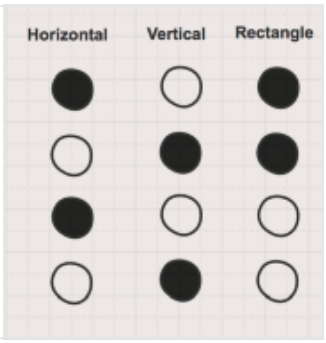
towardsdatascience.com

Or simply read the next blog in the series

Distributed Vector Representation : Simplified

Arguably the most essential feature representation method in Machine Learning

towardsdatascience.com



. . .

References

Open in app Get started

learning." arXiv preprint arXiv:1603.07285 (2016).
[3] Chen, Liang-Chieh, et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs." IEEE transactions on pattern analysis and machine intelligence 40.4 (2017): 834–848.
[4] Dai, Jifeng, et al. "Deformable convolutional networks." Proceedings of the IEEE international conference on computer vision. 2017.
[5] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
[6] https://github.com/vdumoulin/conv_arithmetic

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

Get this newsletter