Published in Towards Data Science

Chi-Feng Wang Follow
Aug 14, 2018 · 8 min read · Listen

A Basic Introduction to Separable Convolutions

Anyone who takes a look at the architecture of MobileNet will undoubtedly come across the concept of separable convolutions. But what is that, and how is it different from a normal convolution?

There are two main types of separable convolutions: spatial separable convolutions, and depthwise separable convolutions.

Spatial Separable Convolutions

Conceptually, this is the easier one out of the two, and illustrates the idea of separating one convolution into two well, so I'll start with this. Unfortunately, spatial separable convolutions have some significant limitations, meaning that it is not heavily used in deep learning.

The spatial separable convolution is so named because it deals primarily with the **spatial dimensions** of an image and kernel: the width and the height. (The other dimension, the "depth" dimension, is the number of channels of each image).

A spatial separable convolution simply divides a kernel into two, smaller kernels. The most common case would be to divide a 3x3 kernel into a 3x1 and 1x3 kernel, like so:

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Image 1: Separating a 3×3 kernel spatially

Now, instead of doing one convolution with 9 multiplications, we do two convolutions with 3 multiplications each (6 in total) to achieve the same effect. With less multiplications, computational complexity goes down, and the network is able to run faster.

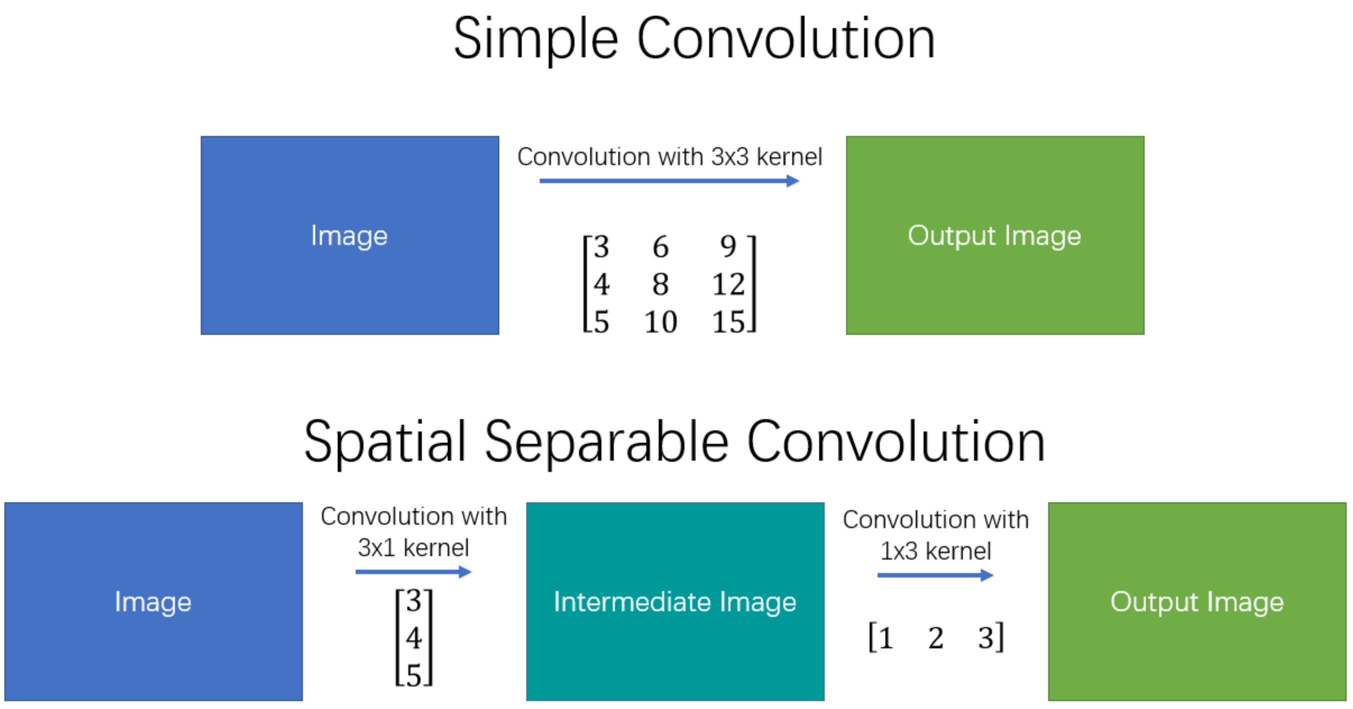

Simple Convolution
Image
Convolution with 3x3 kernel
[3 6 9; 4 8 12; 5 10 15]
Output Image
Spatial Separable Convolution
Image
Convolution with 3x1 kernel
[3; 4; 5]
Intermediate Image
Convolution with 1x3 kernel
[1 2 3]
Output Image

Image 2: Simple and spatial separable convolution

One of the most famous convolutions that can be separated spatially is the Sobel kernel, used to detect edges:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Image 3: Separating the Sobel kernel

The main issue with the spatial separable convolution is that not all kernels can be “separated” into two, smaller kerne 6.8K 42 ticularly bothersome during training, since of all the possible kernels the network could have adopted, it can only end up using one of the tiny portion that can be separated into two smaller kernels.

Depthwise Separable Convolutions

Unlike spatial separable convolutions, depthwise separable convolutions work with kernels that cannot be “factored” into two smaller kernels. Hence, it is more commonly used. This is the type of separable convolution seen in keras.layers.SeparableConv2D or tf.layers.separable_conv2d.

The depthwise separable convolution is so named because it deals not just with the spatial dimensions, but with the depth dimension — the number of channels — as well. An input image may have 3 channels: RGB. After a few convolutions, an image may have multiple channels. You can image each channel as a particular interpretation of that image; in for example, the “red” channel interprets the “redness” of each pixel, the “blue” channel interprets the “blueness” of each pixel, and the “green” channel interprets the “greenness” of each pixel. An image with 64 channels has 64 different interpretations of that image.

Similar to the spatial separable convolution, a depthwise separable convolution splits a kernel into 2 separate kernels that do two convolutions: the depthwise convolution and the pointwise convolution. But first of all, let’s see how a normal convolution works.

Normal Convolution:

If you don’t know how a convolution works from a 2-D perspective, read this article or check out this site.

A typical image, however, is not 2-D; it also has depth as well as width and height. Let us assume that we have an input image of 12x12x3 pixels, an RGB image of size 12x12.

Let’s do a 5x5 convolution on the image with no padding and a stride of 1. If we only consider the width and height of the image, the convolution process is kind of like this: 12x12 — (5x5) — >8x8. The 5x5 kernel undergoes scalar multiplication with every 25 pixels, giving out1 number every time. We end up with a 8x8 pixel image, since there is no padding (12–5+1 = 8).

However, because the image has 3 channels, our convolutional kernel needs to have 3 channels as well. This means, instead of doing 5x5=25 multiplications, we actually do 5x5x3=75 multiplications every time the kernel moves.

Just like the 2-D interpretation, we do scalar matrix multiplication on every 25 pixels, outputting 1 number. After going through a 5x5x3 kernel, the 12x12x3 image will become a 8x8x1 image.

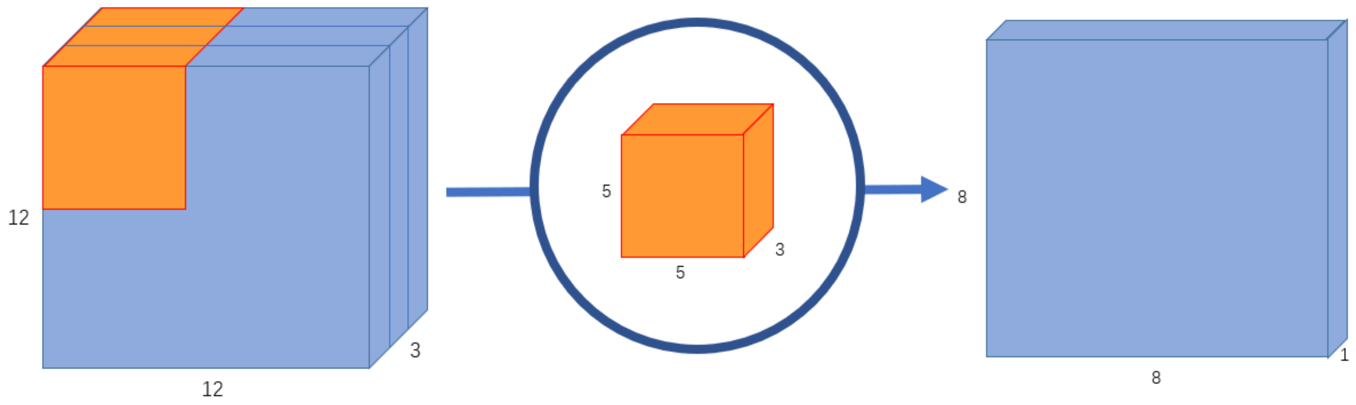


Image 4: A normal convolution with 8×8×1 output

What if we want to increase the number of channels in our output image? What if we want an output of size 8x8x256?

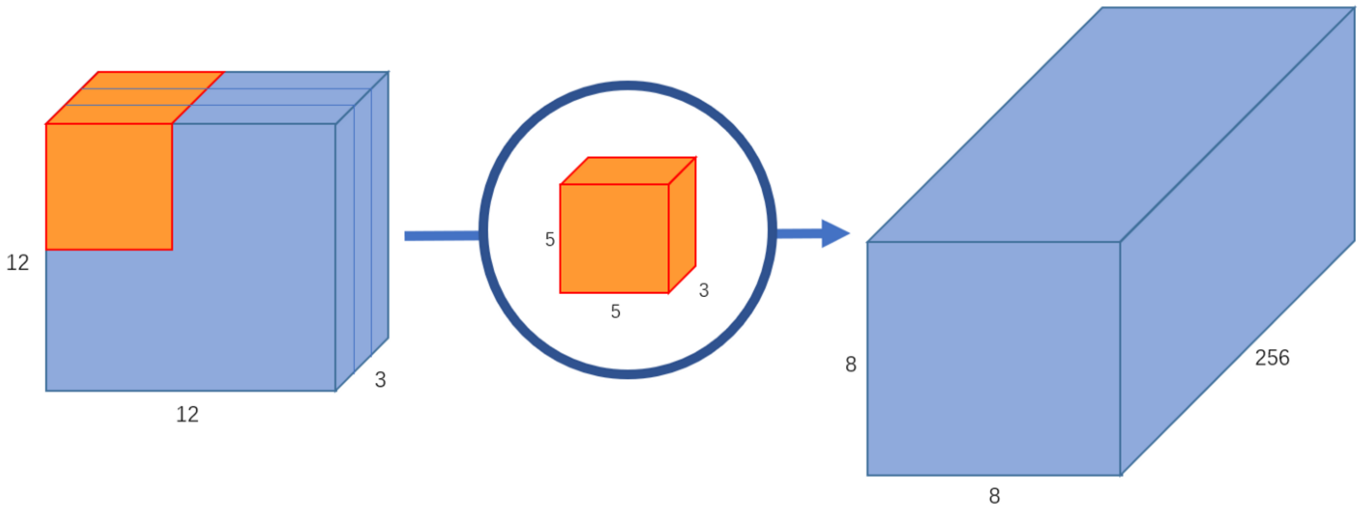


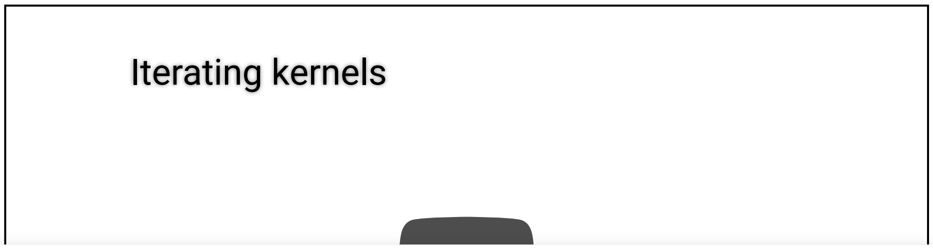
Image 5: A normal convolution with 8×8×256 output

This is how a normal convolution works. I like to think of it like a function: 12x12x3 — (5x5x3x256) — >12x12x256 (Where 5x5x3x256 represents the height, width, number of input channels, and number of output channels of the kernel). Not that this is not matrix multiplication; we're not multiplying the whole image by the kernel, but moving the kernel through every part of the image and multiplying small parts of it separately.

A depthwise separable convolution separates this process into 2 parts: a depthwise convolution and a pointwise convolution.

Part 1 — Depthwise Convolution:

In the first part, depthwise convolution, we give the input image a convolution without changing the depth. We do so by using 3 kernels of shape 5x5x1.



Video 1: Iterating 3 kernels through a 3 channel image

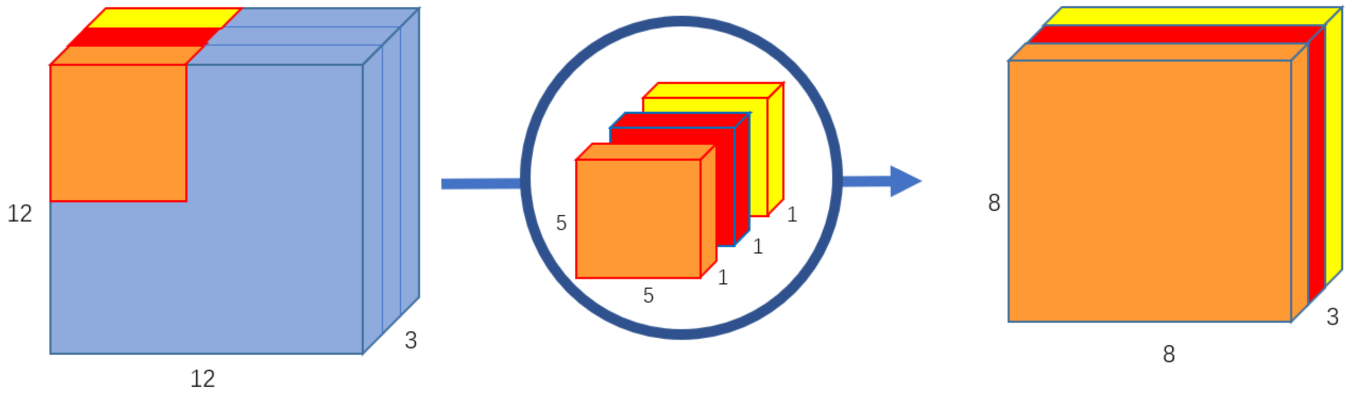


Image 6: Depthwise convolution, uses 3 kernels to transform a 12×12×3 image to a 8×8×3 image

Each 5x5x1 kernel iterates 1 channel of the image (note: **1 channel**, not all channels), getting the scalar products of every 25 pixel group, giving out a 8x8x1 image. Stacking these images together creates a 8x8x3 image.

Part 2 — Pointwise Convolution:

Remember, the original convolution transformed a 12x12x3 image to a 8x8x256 image. Currently, the depthwise convolution has transformed the 12x12x3 image to a 8x8x3 image. Now, we need to increase the number of channels of each image.

The pointwise convolution is so named because it uses a 1x1 kernel, or a kernel that

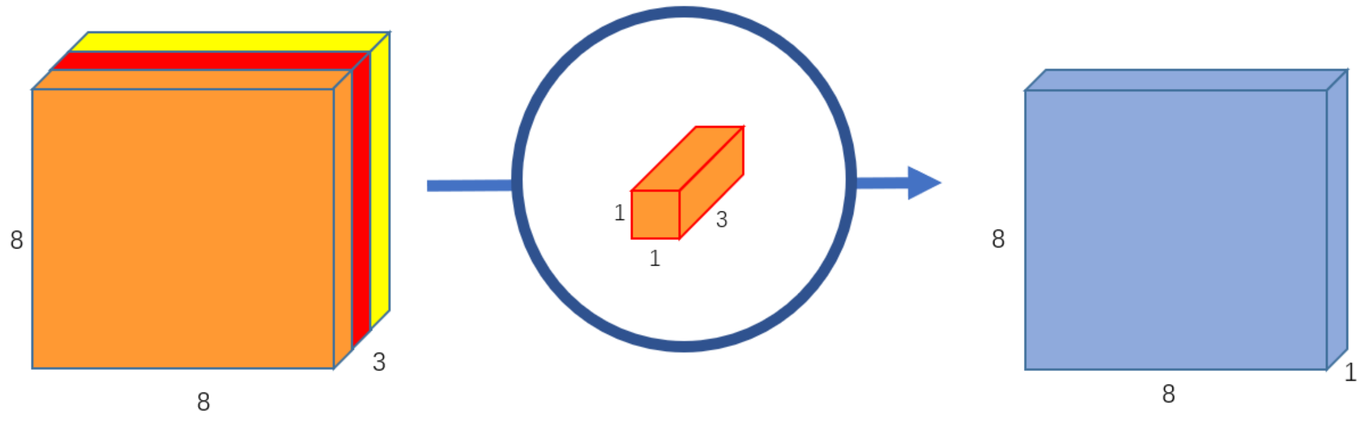


Image 7: Pointwise convolution, transforms an image of 3 channels to an image of 1 channel

We can create 256 1x1x3 kernels that output a 8x8x1 image each to get a final image of shape 8x8x256.

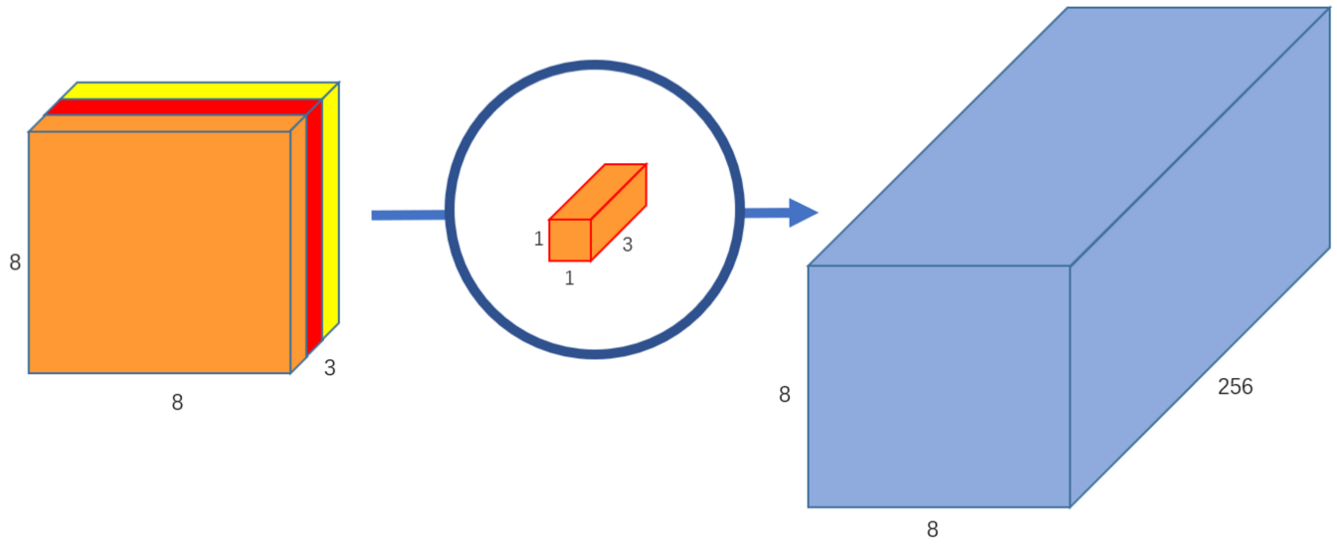


Image 8: Pointwise convolution with 256 kernels, outputting an image with 256 channels

And that's it! We've separated the convolution into 2: a depthwise convolution and a pointwise convolution. In a more abstract way, if the original convolution function is 12x12x3 — (5x5x3x256) →12x12x256, we can illustrate this new convolution as 12x12x3 — (5x5x1x1) — > (1x1x3x256) — >12x12x256.

Let’s calculate the number of multiplications the computer has to do in the original convolution. There are 256 5x5x3 kernels that move 8x8 times. That’s 256x3x5x5x8x8=1,228,800 multiplications.

What about the separable convolution? In the depthwise convolution, we have 3 5x5x1 kernels that move 8x8 times. That’s 3x5x5x8x8 = 4,800 multiplications. In the pointwise convolution, we have 256 1x1x3 kernels that move 8x8 times. That’s 256x1x1x3x8x8=49,152 multiplications. Adding them up together, that’s 53,952 multiplications.

52,952 is a lot less than 1,228,800. With less computations, the network is able to process more in a shorter amount of time.

How does that work, though? The first time I came across this explanation, it didn’t really make sense to me intuitively. Aren’t the two convolutions doing the same thing? In both cases, we pass the image through a 5x5 kernel, shrink it down to one channel, then expand it to 256 channels. How come one is more than twice as fast as the other?

After pondering about it for some time, I realized that the main difference is this: in the normal convolution, we are **transforming the image 256 times**. And every transformation uses up 5x5x3x8x8=4800 multiplications. In the separable convolution, we only really **transform the image once** — in the depthwise convolution. Then, we take the transformed image and **simply elongate it to 256 channels**. Without having to transform the image over and over again, we can save up on computational power.

It’s worth noting that in both Keras and Tensorflow, there is a argument called the “depth multiplier”. It is set to 1 at default. By changing this argument, we can change the number of output channels in the depthwise convolution. For example, if we set the depth multiplier to 2, each 5x5x1 kernel will give out an output image of 8x8x2, making the total (stacked) output of the depthwise convolution 8x8x6 instead of 8x8x3. Some may choose to manually set the depth multiplier to increase the number of parameters in their neural net for it to better learn more traits.

during training. If used properly, however, it manages to enhance efficiency without significantly reducing effectiveness, which makes it a quite popular choice.

1×1 Kernels:

Finally, because pointwise convolutions use the concept, I'd like to touch upon the usages of a 1x1 kernel.

A 1x1 kernel — or rather, n 1x1xm kernels where n is the number of output channels and m is the number of input channels — can be used outside of separable convolutions. One obvious purpose of a 1x1 kernel is to increase or reduce the depth of an image. If you find that your convolution has too many or too little channels, a 1x1 kernel can help balance it out.

For me, however, the main purpose of a 1x1 kernel is to apply non-linearlity. After every layer of a neural network, we can apply an activation layer. Whether it be ReLU, PReLU, Softmax, or another, activation layers are non-linear, unlike convolution layers. "A linear combination of lines is still a line." Non-linear layers expand the possibilities for the model, as is what generally makes a "deep" network better than a "wide" network. In order to increase the number of non-linear layers without significantly increasing the number of parameters and computations, we can apply a 1x1 kernel and add an activation layer after it. This helps give the network an added layer of depth.

Leave a comment below if you have any further questions! And don't forget to give this story some claps!