1x1 Convolution operation

Introduction

1x1 Convolutions are specially used in [1] in 2013 and [2] in 2014 where they gain a great popularity.

Since then they are the default option for gaining computational performance in convolutional layers, acting as a bottleneck to reduce the feature maps dimension.

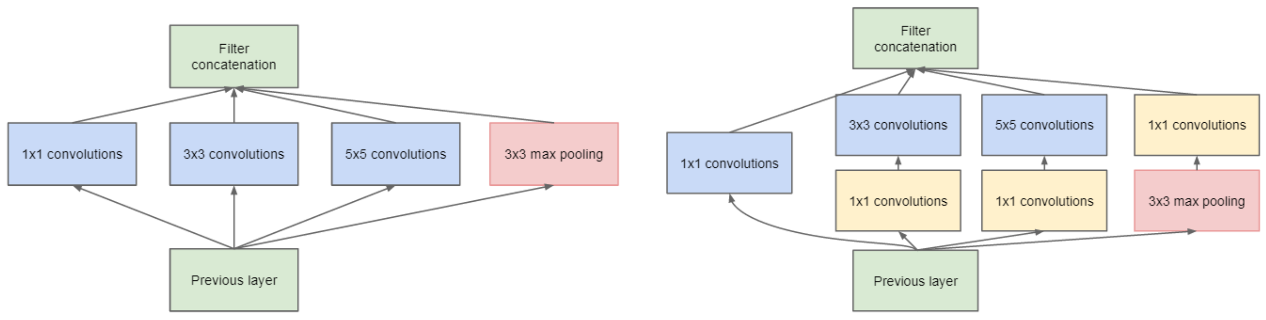
In this post we will take a look at what is going on there.

Motivation

Why do we need this trick in the first place?

Well, it turns out that convolutional operations are computationally expensive with higher kernels dimensionalities (like 3x3 or 5x5) when the input volume has a big number of feature maps.

In the case of the so popular Inception convolutional neural networks, as Figure 1 shown, the feature maps are concatenated from several parallel branches. The reason to have these branches is simply because each one is attempting to extract feature at different scales.

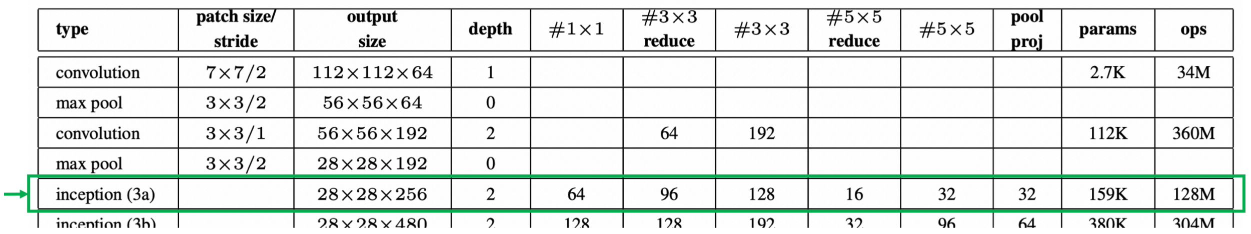


(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 1. Inception modules

A good way to understand how 1x1 is helping, is taking a look at how authors of the Inception module used them.

We will take a close look at the first inception module of the Inception-3A, where form the paper we have the following information:



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M

Figure 2. GoogLeNet incarnation of the Inception architecture from the original paper [2]

From Figure 2 we could see how the input size of that layer (which is the output size of the layer before) is 28x28x566, and from Figure 1 we could see that, for instance, the third branch is a 1x1 convolution right before a more expensive 5x5 convolution. Because the purpose of this post is just to explain 1x1 convolution, we will cover only that branch. However, check out the post on the GoogLeNet[1] to see the entire mechanism of that module or even better exercise, do it by yourself before!

Returning to that third branch on the right-hand size of Figure 1, authors implemented a 1x1 convolution before compared to the naïve implementation. Why did they do that?

1x1 convolution allows us to compress the data of the input volume into a smaller volume before performing the more expensive 5x5 convolution. This way, we encourage the weights to find a more efficient representation of the data.

1x1 convolution

According to the table then our input volume has 28x28x256 dimensions.

Let's explore the difference between performing the naïve Inception module (left-hand side figure) and do the 5x5 convolution first. Figure 3 (a) shows how to perform the 5x5 convolution. If you are not familiar with convolutions or the visualization/notation, I did a post before using the same methodology for simple convolution.

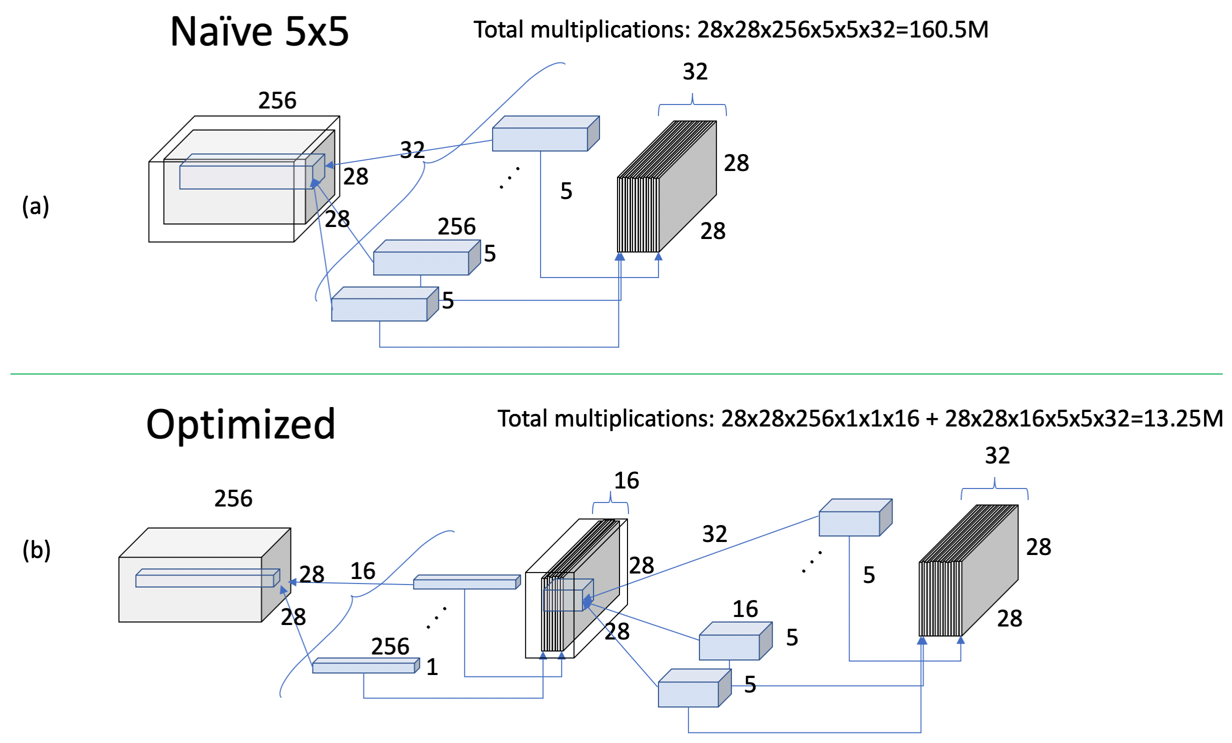

Naïve 5x5
Total multiplications: 28x28x256x5x5x32=160.5M
(a)
256
28
28
32
256
5
5
5
32
28
28
Optimized
Total multiplications: 28x28x256x1x1x16 + 28x28x16x5x5x32=13.25M
(b)
256
28
28
16
256
1
16
28
28
32
16
5
5
5
32
28
28

Figure 3. 1x1 Convolution to reduce the computations

[1] Ongoing post

The number of multiplications we have to do to achieve the 5x5 convolution with 32 filters as the authors do results in 160.5 million operations. This confirms the previous statement of being such a expensive operations, specially if we compare it with the improve we can get using a previous 1x1 convolution to reduce the feature maps dimension.

Figure 3 (b) uses 16 filters of 1x1 convolution to preserve the spatial dimensions of the image (28x28) while compressing the feature maps from 256 to 16. Next, the same 5x5 convolution with 32 filters is performed on the new volume, to provide the same output volume as the naïve implementation.

Note how by doing this ‘trick’, the number of operations required are reduced to 13.25 million!

Bottleneck

If we just focus our attention in the feature maps dimension, that would be something like looking at Figure 3 from the top, we would observe Figure 4.

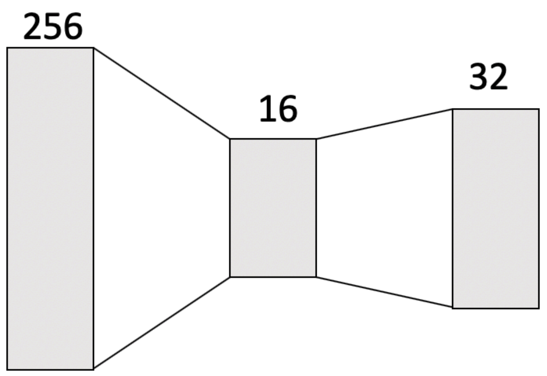


Figure 4. Bottleneck of 1x1 convolutions

We can better observe the bottleneck effect.

This is a term very used in convolutional neural networks from the work on [1] and [2] and inspired future convolutional neural architectures like ResNets and DenseNets.

Bibliography

[1] M. Lin, Q. Chen and S. Yan, Network In Network, 2013.

[2] C. Szegedy, W. Liu, Y. Jia, P. Semanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, 2014.