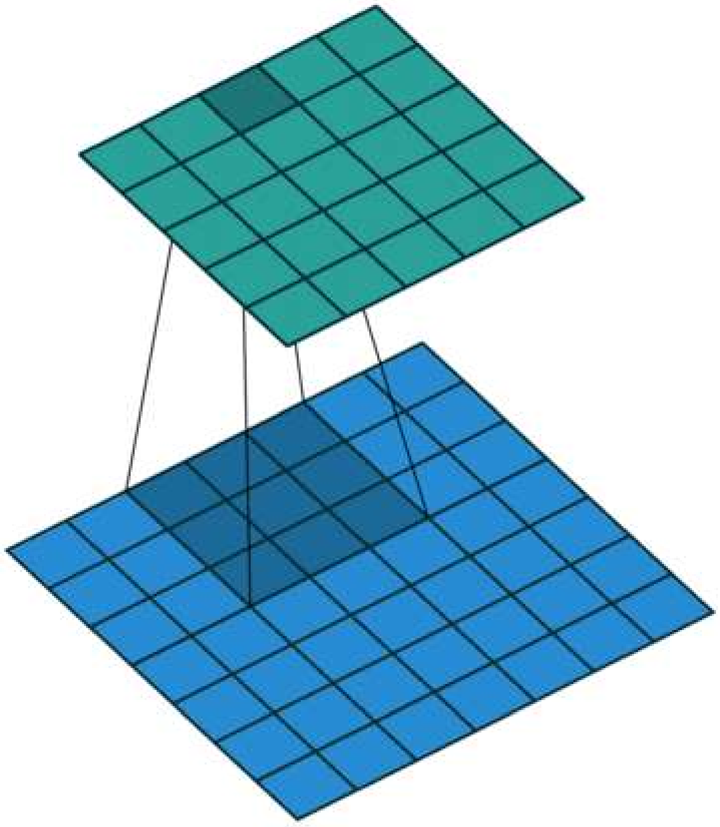
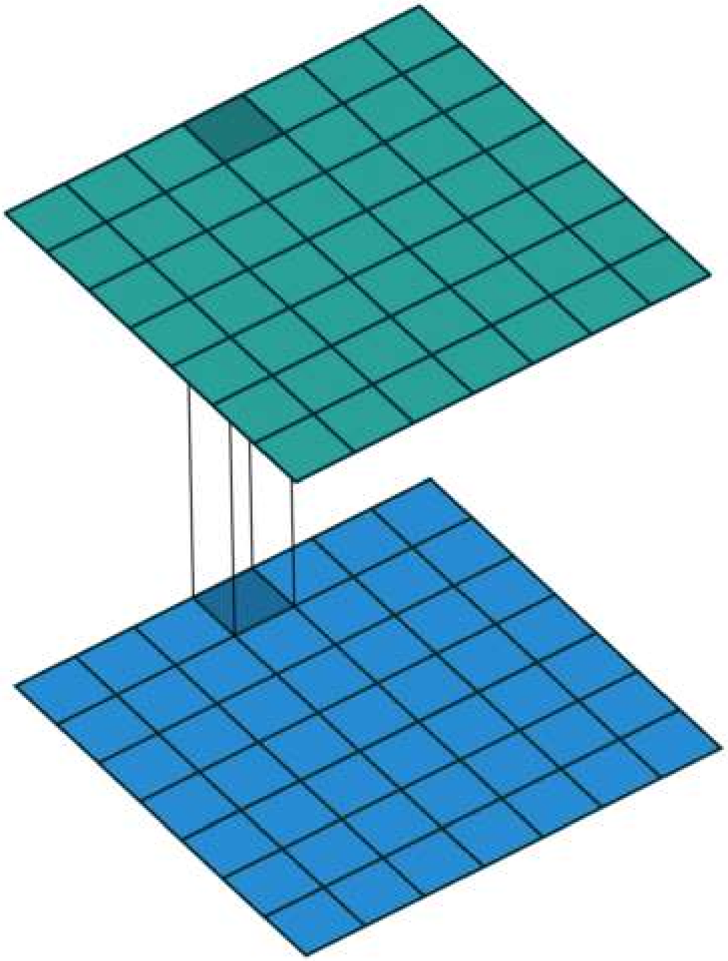
One by One [1 x 1] Convolution - counter-intuitively useful

Whenever I discuss or show GoogleNet architecture, one question always comes up -

"Why 1x1 convolution ? Is it not redundant ?





left : **Convolution with kernel of size 3x3** right : **Convolution with kernel of size 1x1**

Simple Answer

Most simplistic explanation would be that 1x1 convolution leads to dimension reductionality. For example, an image of 200 x 200 with 50 features on convolution with 20 filters of 1x1 would result in size of 200 x 200 x 20. But then again, is this is the best way to do dimensionality reduction in the convoluational neural network? What about the efficacy vs efficiency?

Complex Answer

Feature transformation

Although 1x1 convolution is a 'feature pooling' technique, there is more to it than just sum pooling of features across various channels/feature-maps of a given layer. 1x1 convolution acts like coordinate-dependent transformation in the filter space[1]. It is important to note here that this transformation is strictly linear, but in most of application of 1x1 convolution, it is succeeded by a non-linear activation layer like ReLU. This transformation is learned through the (stochastic) gradient descent. But an important distinction is that it suffers with less over-fitting due to smaller kernel size (1x1).

Deeper Network

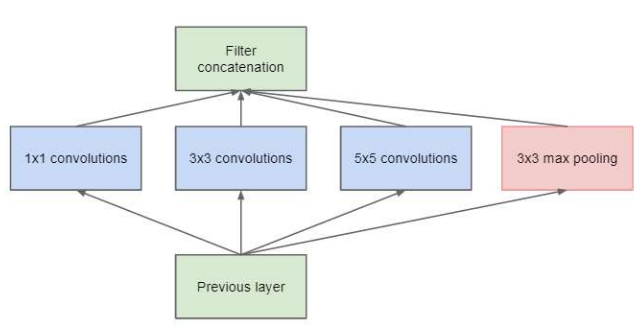
One by One convolution was first introduced in this paper titled Network in Network. In this paper, the author's goal was to generate a deeper network without simply stacking more layers. It replaces few filters with a smaller perceptron layer with mixture of 1x1 and 3x3 convolutions. In a way, it can be seen as "going wide" instead of "deep", but it should be noted that in machine learning terminology, 'going wide' is often meant as adding more data to the training. Combination of 1x1 (x F) convolution is mathematically equivalent to a multi-layer perceptron.[2].

Inception Module

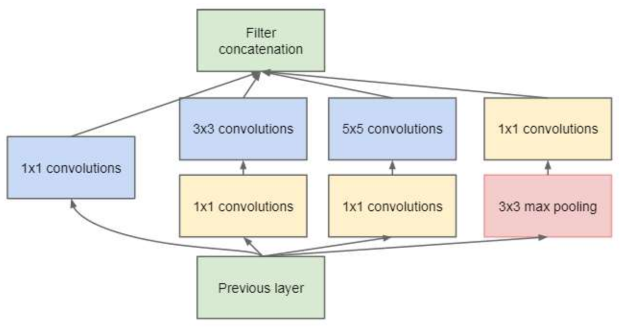
In GoogLeNet architecture, 1x1 convolution is used for two purposes

- To make network deep by adding an "inception module" like Network in Network paper, as described above.
- To reduce the dimensions inside this "inception module".
- To add more non-linearity by having ReLU immediately after every 1x1 convolution.

Here is the scresnshot from the paper, which elucidates above points :



(a) Inception module, naïve version



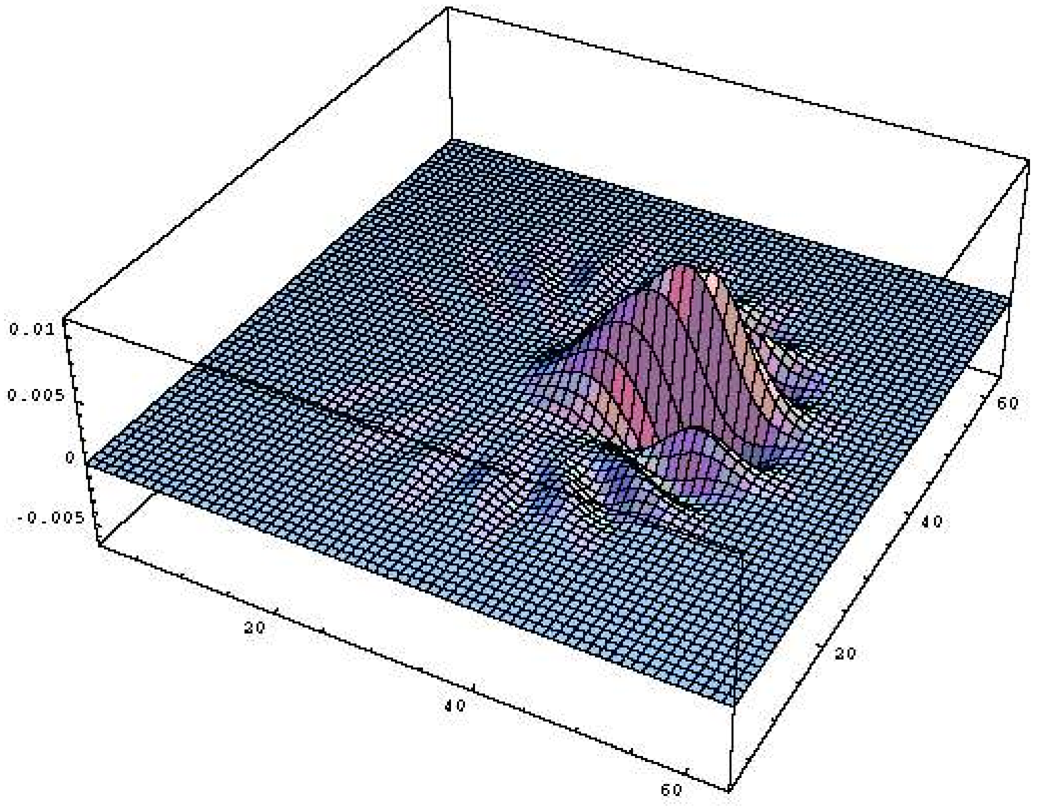
(b) Inception module with dimension reductions

1x1 convolutions in GoogLeNet

It can be seen from the image on the right, that 1x1 convolutions (in yellow), are specially used before 3x3 and 5x5 convolution to reduce the dimensions. It should be noted that a two step convolution operation can always to combined into one, but in this case and in most other deep learning networks, convolutions are followed by non-linear activation and hence convolutions are no longer linear operators and cannot be combined.

In designing such a network, it is important to note that initial convolution kernel should be of size larger than 1x1 to have a receptive field capable of capturing locally spatial information. According to the NIN paper, 1x1 convolution is equivalent to cross-channel parametric pooling layer. From the paper - “This cascaded cross channel parameteric pooling structure allows complex and learnable interactions of cross channel information”.

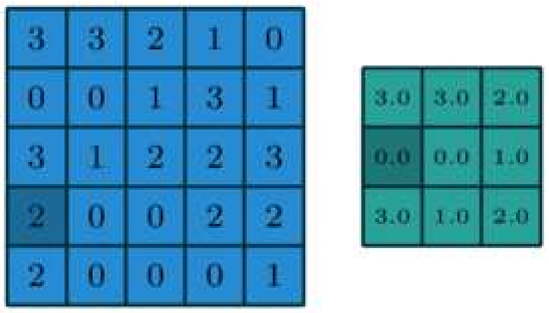
Cross channel information learning (cascaded 1x1 convolution) is biologically inspired because human visual cortex have receptive fields (kernels) tuned to different orientation. For e.g



Different orientation tuned receptive field profiles in the human visual cortex
Source

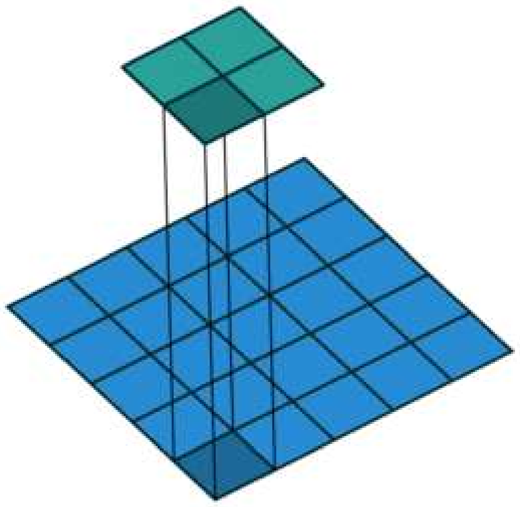
More Uses

- 1x1 Convolution can be combined with Max pooling



Pooling with 1x1 convolution

- 1x1 Convolution with higher strides leads to even more redution in data by decreasing resolution, while losing very little non-spatially correlated information.



1x1 convolution with strides

- Replace fully connected layers with 1x1 convolutions as Yann LeCun believes they are the same -

> *In Convolutional Nets, there is no such thing as "fully-connected layers". There are only convolution layers with 1x1 convolution kernels and a full connection table. – Yann LeCun*

Convolution gif images generated using this wonderful code, more images on 1x1 convolutions and 3x3 convolutions can be found here

Written on March 25, 2016

18 Comments **iamaaditya** 1 **Login**

Recommend 4 **Share** **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

HS Choi • 7 months ago

HI I've finished reading through your great post with great effort but I have one question left

I still question about what lecun said

since when using convnet we apply the same kernel through each channel and that means we share the same weights through each channel and since we only have one by one sized kernel I think it's like using the same weight to each input node to map that to one output node in the perspective of fully connected layer. So I think it's little bit of exaggeration to say that one by one conv has the same effect as fully connected layer since fully connected layer doesn't share the weight through each input node, thus resulting in higher model complexity.
If there's any misunderstanding please let me informed! Thanks

1 ^ | v • Reply • Share ›

Kirill Konevets → HS Choi • 4 months ago

That is right. You can not replace 1x1 convolution layer with fully connected, but you can do the opposite. Suppose you have 100x100 layer with 3 channels and you fully connect it with layer containing K nodes. This means you have 100x100x3xK weights. Now if you want to replace it with 1x1 convolution layer, just take it with 100x100xK filters, this means you have 1x1x3x100x100xK weights. Looks the same.

^ | v • Reply • Share ›

Felix Pîrvan → Kirill Konevets • 3 months ago

As I understand it, the convolution shares the weights across the input feature maps. So with 1x1 convolution we have 1x1x3xK distinct weights, repeated 100x100 times. It is only a particular case of fully connected layer.

^ | v • Reply • Share ›

Bazyli Debowski → Felix Pîrvan • 2 months ago

I am getting the same understanding as you Felix.

^ | v • Reply • Share ›

Gaurav Menghani • a month ago

The animation for the 1x1 at the top doesn't seem to be correct as the output doesn't seem to depend on the entire spatial dimension of the input.

^ | v • Reply • Share ›

aaditya prakash Mod → Gaurav Menghani • a month ago

Hi Gaurav. Thanks for pointing out the error. I have fixed the image. Cheer.s

1 ^ | v • Reply • Share ›

Martin Wieser • 9 months ago

HI i have another question, the last gif is also 'not showing the full output' or?

^ | v • Reply • Share ›

aaditya prakash Mod → Martin Wieser • 9 months ago

You are right. The last one is missing one row of output (the top one). It will be tough to rebuild the gif but you can use this code to generate your own tough to rebuild the gif but you can use this code to generate your own https://github.com/vdumouli... (if you do, send me the link and I will update the animation). Thanks for pointing it out.

^ | ˅ • Reply • Share ›

Rick Xie • 10 months ago

Hi, thank you for your excellent and useful post. I have a question about: "In a way, it can be seen as “going wide” instead of “deep”, but it
should be noted that in machine learning terminology, ‘going wide’ is
often meant as adding more data to the training". By using 1*1 conv kernel, the author made the net become 'wider(as you explained: more data)', but how did he achieve the goal of getting 'deeper'? Thanks.

^ | ˅ • Reply • Share ›

aaditya prakash Mod ➦ Rick Xie • 10 months ago

see more

^ | ˅ • Reply • Share ›

Rick Xie ➦ aaditya prakash • 10 months ago

Thank you for providing details. I now see the feature concatenation is the key of keeping more data remained (we have lost a lot of information when we perform pooling operations) and get stuffs deeper. By the way, I think there is something wrong with the GIF on the right of the first figure, since using 1*1 conv kernel will produce a feature map with 'width' & `height` that are identical to the inputting `width` & 'height'. Thanks.

^ | ˅ • Reply • Share ›

Sunwoo Oh • a year ago

Thanks for sharing! This was exactly what I was looking for!

Reply • Share ›

surf reta • a year ago

Hi, I have a question regarding the right one of the first picture in the very above. The input image is 7*7, why the output is 5*5. If we use 1*1 convolution with stride =1, the output should be 7*7 as well.

Reply • Share ›

aaditya prakash Mod surf reta • a year ago

You are absolutely right. It should have been 7x7 as well. I guess because I wanted to make the images similar to the left. Think of it as 'not showing the full output'.

Reply • Share ›

Alireza Nejati • 2 years ago

> In a way, it can be seen as “going wide” instead of “deep”, but it should be noted that in machine learning terminology, ‘going wide’ is often meant as adding more data to the training. Combination of 1x1 (x F) convolution is mathematically equivalent to a multi-layer perceptron.[2].

Yep. In fact, you can express a classical feed-forward multi-layer perceptron (MLP) as a single pixel image with n channels, and stacking a number of 1x1 'convolutions' on top of it.

Love your blog, by the way. Keep up the good work.

Reply • Share ›

Tejas Khot Alireza Nejati • a year ago

"Yep. In fact, you can express a classical feed-forward multi-layer perceptron (MLP) as a single pixel image with n channels, and stacking a number of 1x1 'convolutions' on top of it."
Can you please explain how this could be done, didn't understand.

Reply • Share ›

Jane Tejas Khot • 9 months ago

if your MLP has two layers: input layer of size n and output of size m. you can rearrange your input layer and look at it as if it were a single pixel image with n channels, i.e. 1x1xn input layer. Analogue, rearrange your output layer into a 1x1xm layer. For your original weight matrix W, with W_ij being the weight connecting input i with output j, create for each of the m output nodes one 1x1 filter of size 1x1xn, so in total you will have m 1x1xn kernels. Kernel j will contain the weights W_ij, i=1..n

Then you will see that both the MLP and the transformed convNet with

1x1 filters are equivalent

Reply • Share ›

aaditya prakash Mod → Alireza Nejati • 2 years ago

Thanks for the nice words.

Reply • Share ›