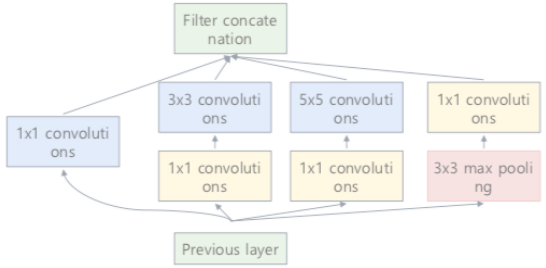
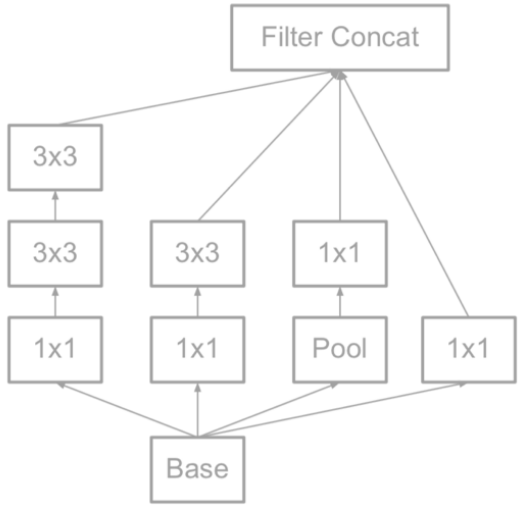
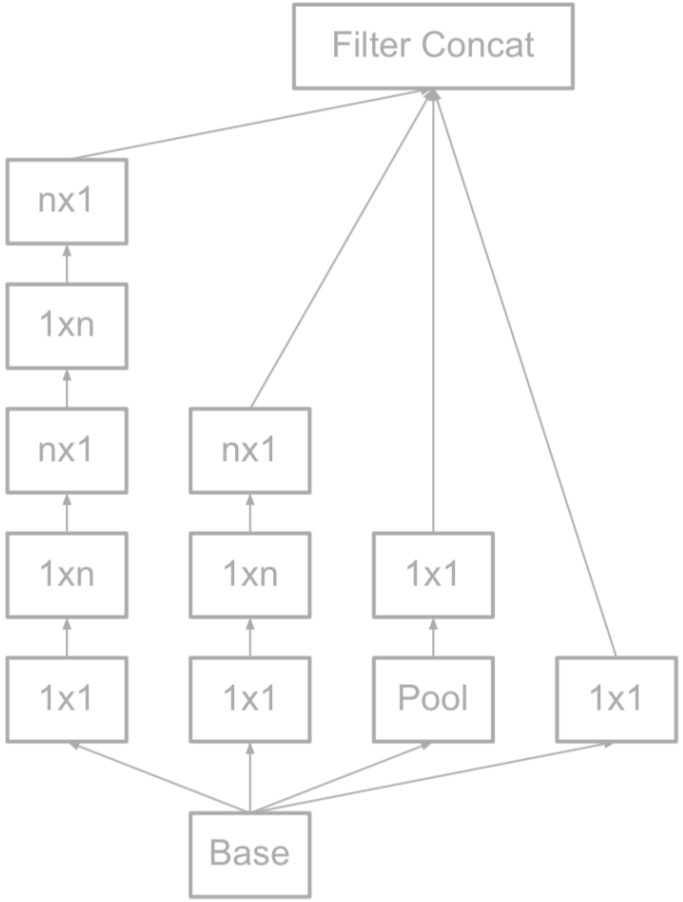
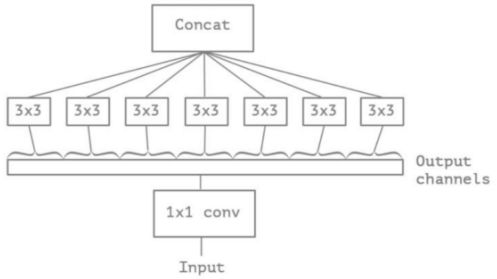








Inceptions

薛飞飞

6th July, 2018

Today's

Contents

Inceptions & Xception

- [v1]Going Deeper with Convolution
- [v2]Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
- [v3]Rethinking the Inception Architecture for Computer Vision
- [v4]Inception-v4, Inception-RestNet and the Impact of Residual Connections on Learning
- [X] Xception: Deep Learning with Depthwise Separable Convolutions

Motivation

Q) What is the best way to improve the performance of deep neural network?

A) Bigger size!
(Increase the depth and width of the model)

Problems

1. Bigger model typically means a larger number of parameters → overfitting
2. Increased use of computational resources

   → e.g. quadratic increase of computation

$$\mathbf{3\times3\times}\boldsymbol{C} \rightarrow \mathbf{3\times3\times}\boldsymbol{C}\text{: }\boldsymbol{C}^2 \text{ computations}$$

Solution: Sparsely connected architecture

Sparsely Connected Architecture

1. Mimicking biological system
2. Theoretical underpinnings

→ **Arora et al.**

Provable bounds for learning some deep representations

ICML 2014

"Given samples from a sparsely connected neural network whose each layer is a denoising autoencoder, can the net (and hence its reverse) be learnt in polynomial time with low sample complexity?"

YES!

Why Arora et al. is important

To provably solve optimization problems for general neural networks with two or more layers, the algorithms that would be necessary hit some of the biggest open problems in computer science. So, we don't think there's much hope for machine learning researchers to try to find algorithms that are provably optimal for deep networks. This is because the problem is NP-hard, meaning that provably solving it in polynomial time would also solve thousands of open problems that have been open for decades. Indeed, in 1988 J. Stephen Judd shows the following problem to be NP-hard:

> Given a general neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?

Judd also shows that the problem remains NP-hard even if it only requires a network to produce the correct output for just two-thirds of the training examples, which implies that even approximately training a neural network is intrinsically difficult in the worst case. In 1993, Blum and Rivest make the news worse: even a simple network with just two layers and three nodes is NP-hard to train!

https://www.oreilly.com/ideas/the-hard-thing-about-deep-learning

Problems

1. **Sparse matrix computation is very inefficient**

   → dense matrix calculation is extremely efficient

2. **Even ConvNet changed back from sparse connection to full connection for better optimize parallel computing**

Is there any intermediate step?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
								X	X	X			X		X	X
								X	X	X	X		X	X		X
									X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Inception Architecture

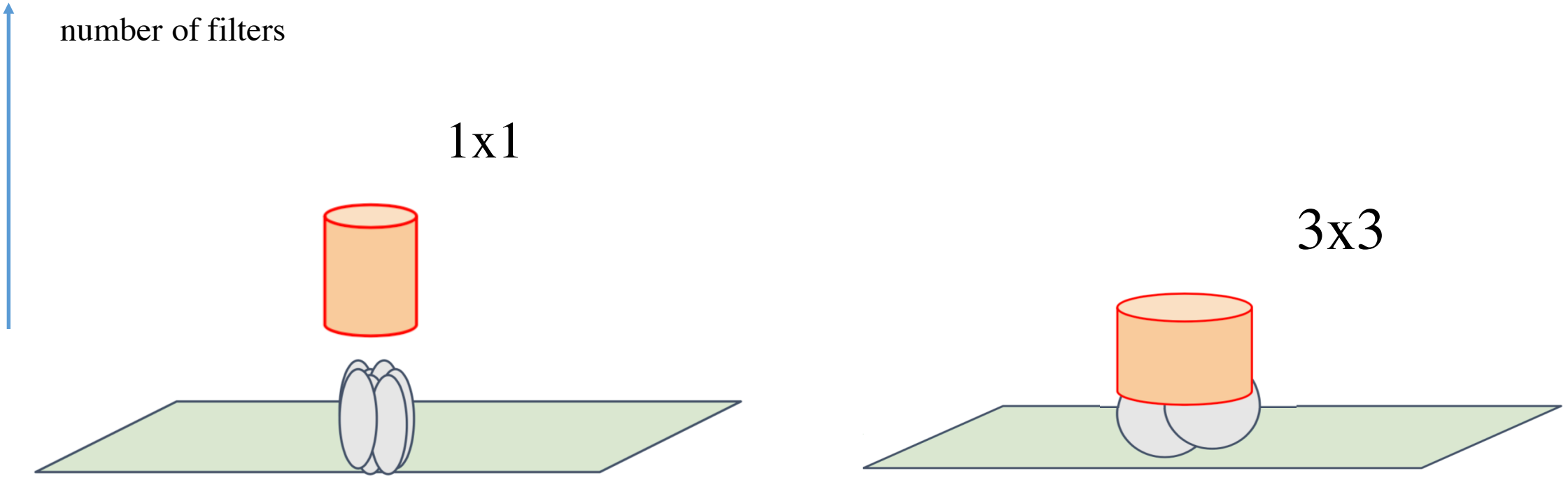
Main idea
How to find out an optimal local sparse structure in a convolutional network and how can it be approximated and covered by readily available dense components?

: All we need is to find the optimal local construction and to repeat it spatially
: Arora et al.: layer by layer construction with correlation statistics analysis

Inception

Architecture

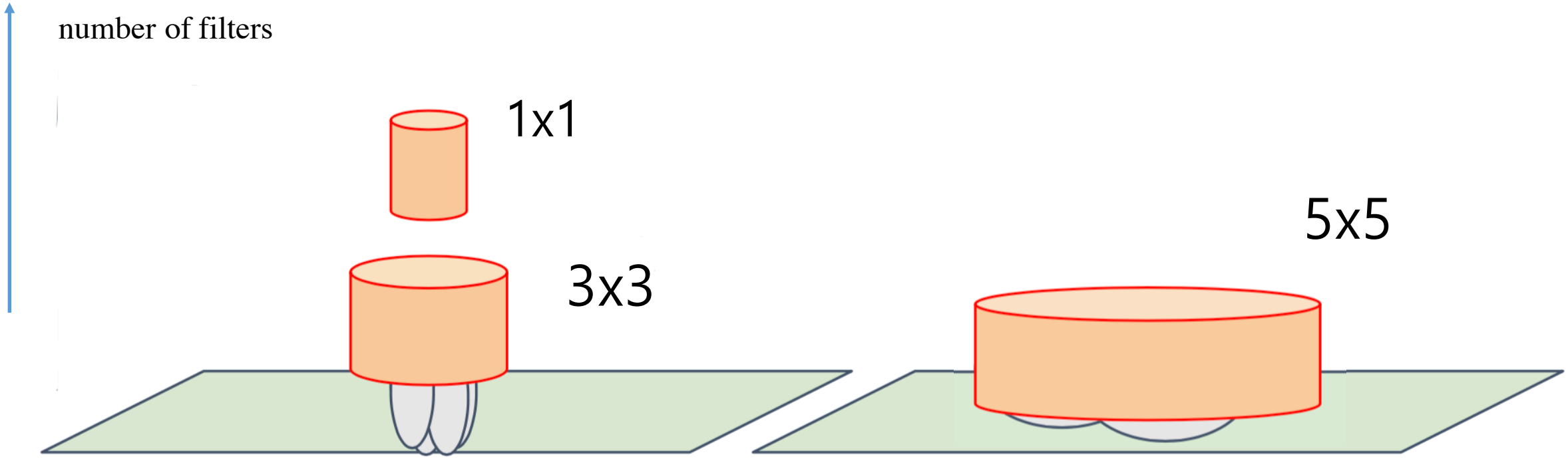
Cover images spatial patterns at different scales by 1x1, 3x3 convolutions



Inception

Architecture

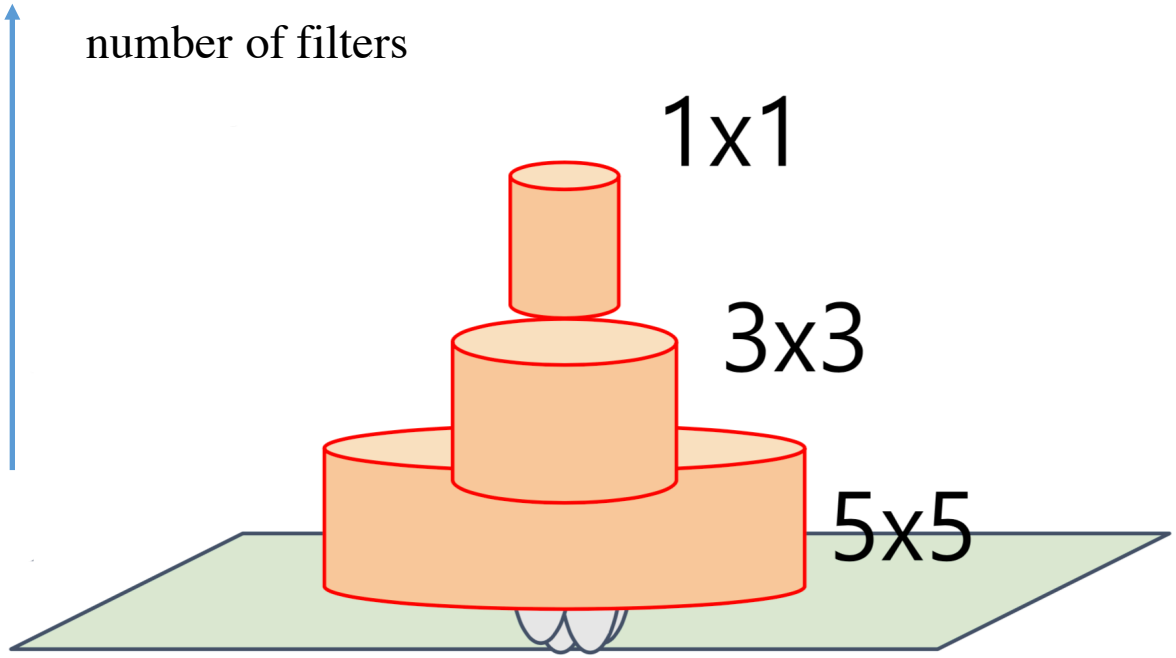
Cover more spread out clusters by 5x5 convolutions

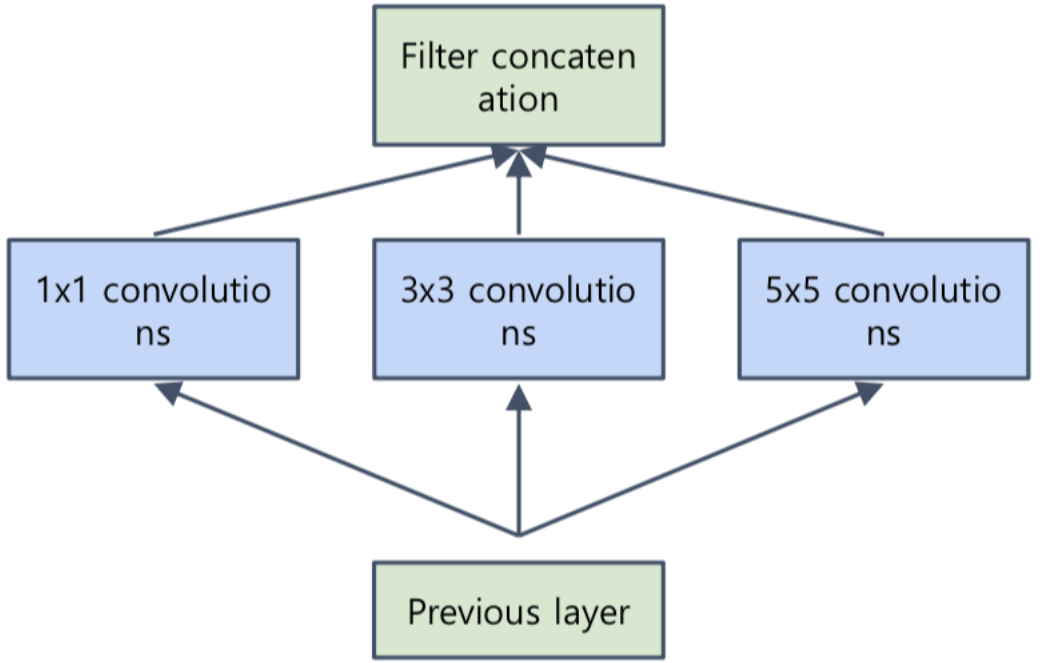


Inception

Architecture > Naïve Version

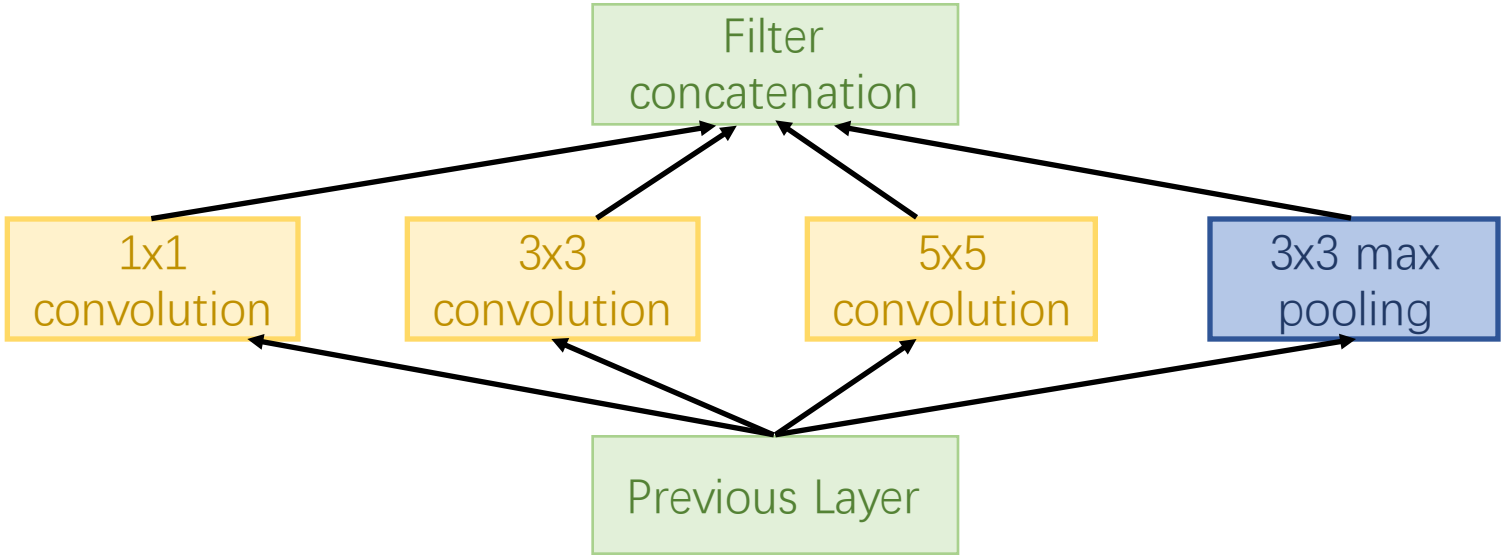
A heterogeneous set of convolutions





Inception

Architecture > Naïve Version



Naïve Inception module

Apply parallel filter operations on the input from previous layer:
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5) ?
- Pooling operation (3x3) ?

Concatenate all filter outputs together depth-wise ?

Q: What is the problem with this?
[Hint: Computational complexity]

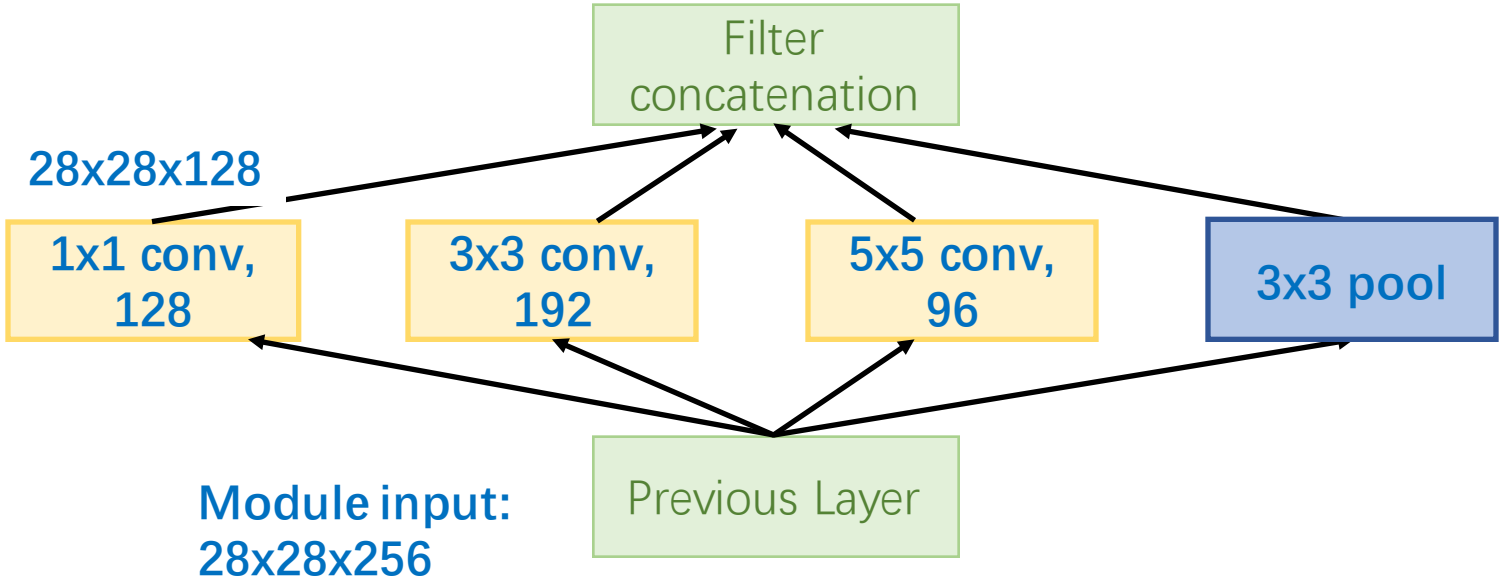
Inception

Architecture > Naïve Version

Example:

Q1:What is the output size of the 1x1 conv, with 128 filters?

Q: What is the problem with this? [Hint: Computational complexity]



Naïve Inception module

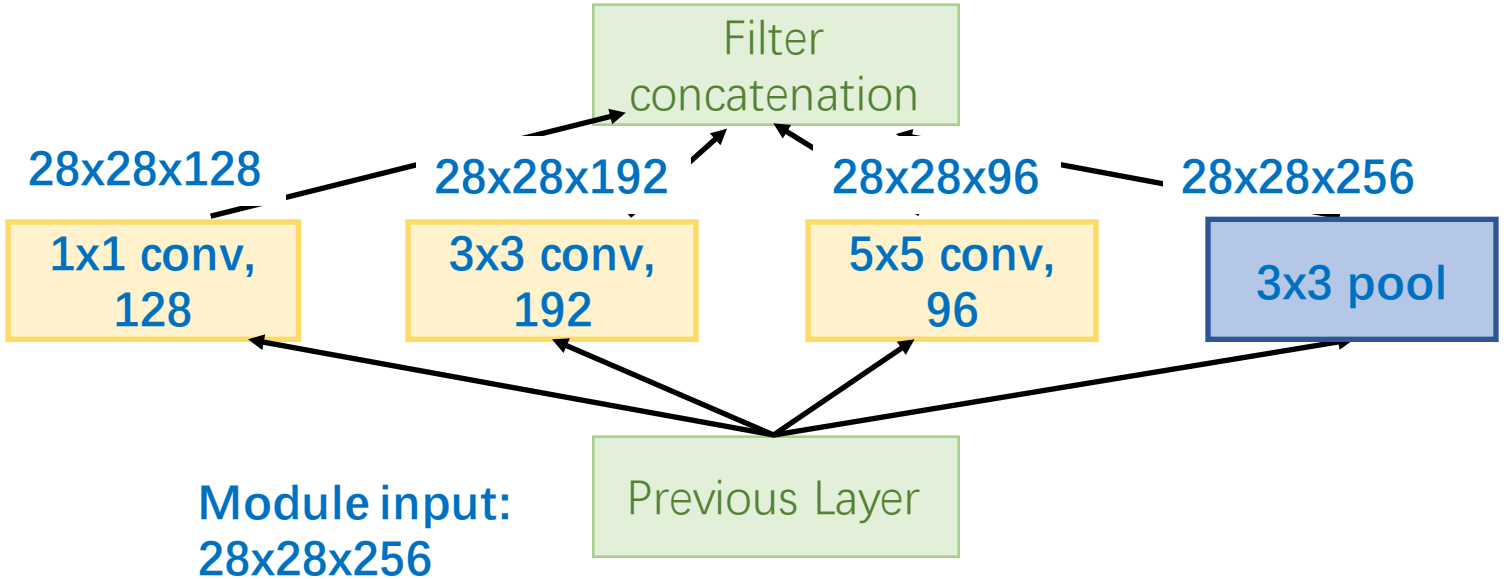
Inception

Architecture > Naïve Version

Example:

Q2:What are the output size of all different filter operations?

Q: What is the problem with this? [Hint: Computational complexity]



Naïve Inception module

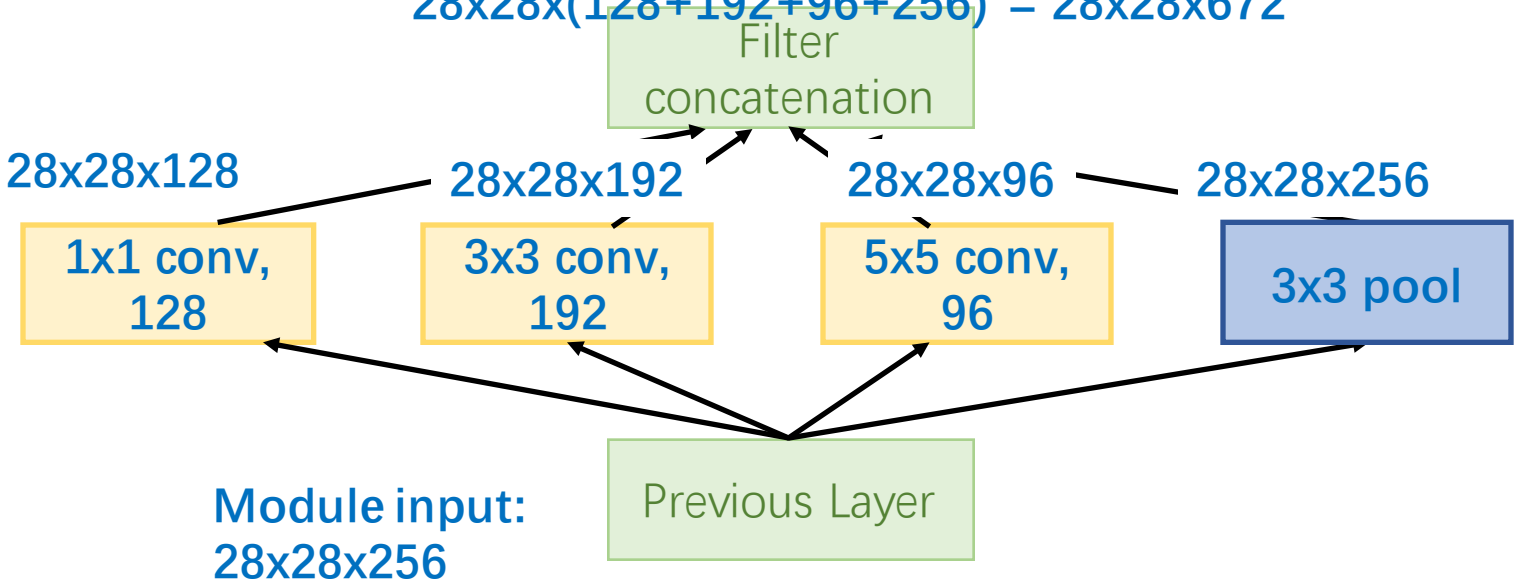
Inception

Architecture > Naïve Version

Example:

Q3:What is output size after Filter concatenation?

28x28x(128+192+96+256) = 28x28x672



Naïve Inception module

Q: What is the problem with this? [Hint: Computational complexity]

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

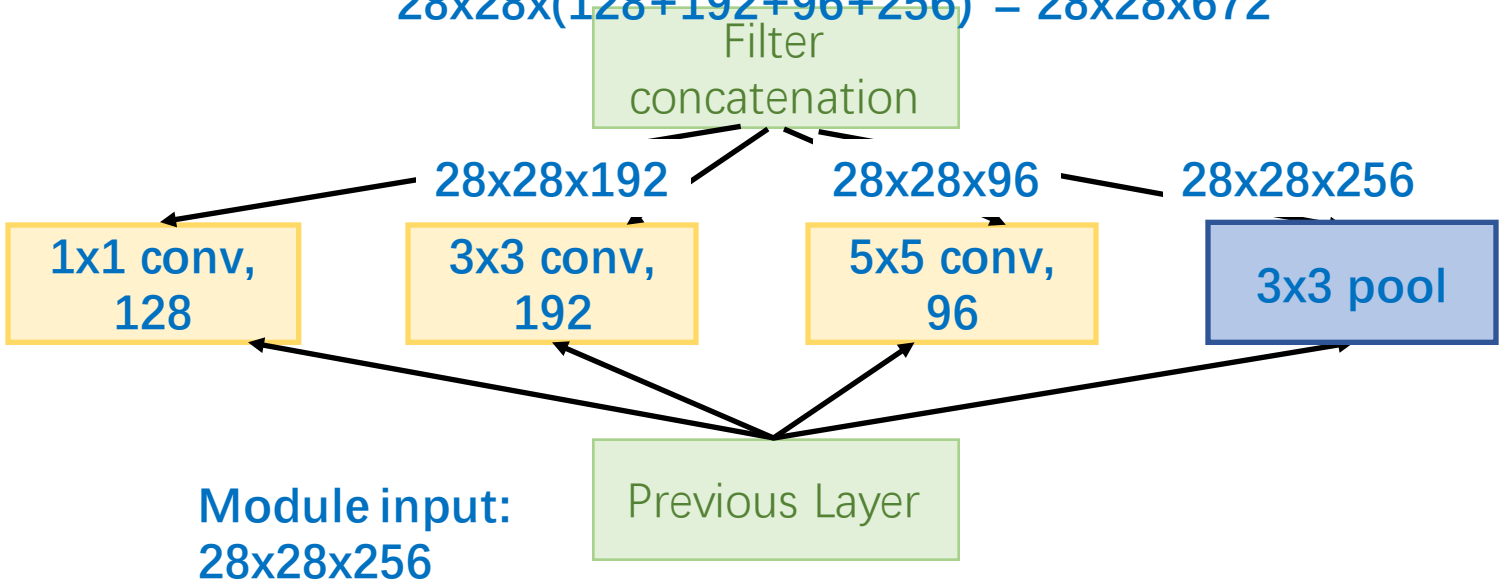
Inception

Architecture > Naïve Version

Example:

Q3:What is output size after Filter concatenation?

28x28x(128+192+96+256) = 28x28x672

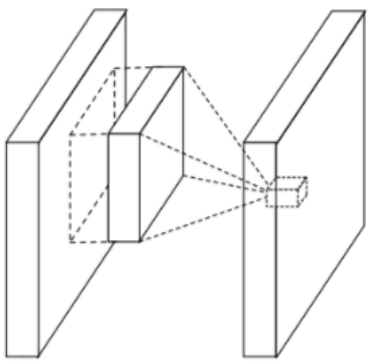


Q: What is the problem with this? [Hint: Computational complexity]

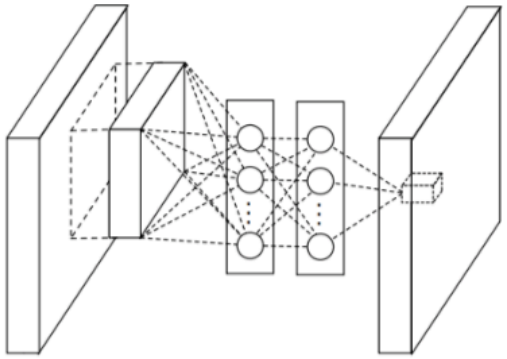
Solution: "bottleneck" layers that use 1x1 convolutions to reduce feature depth

Naïve Inception module

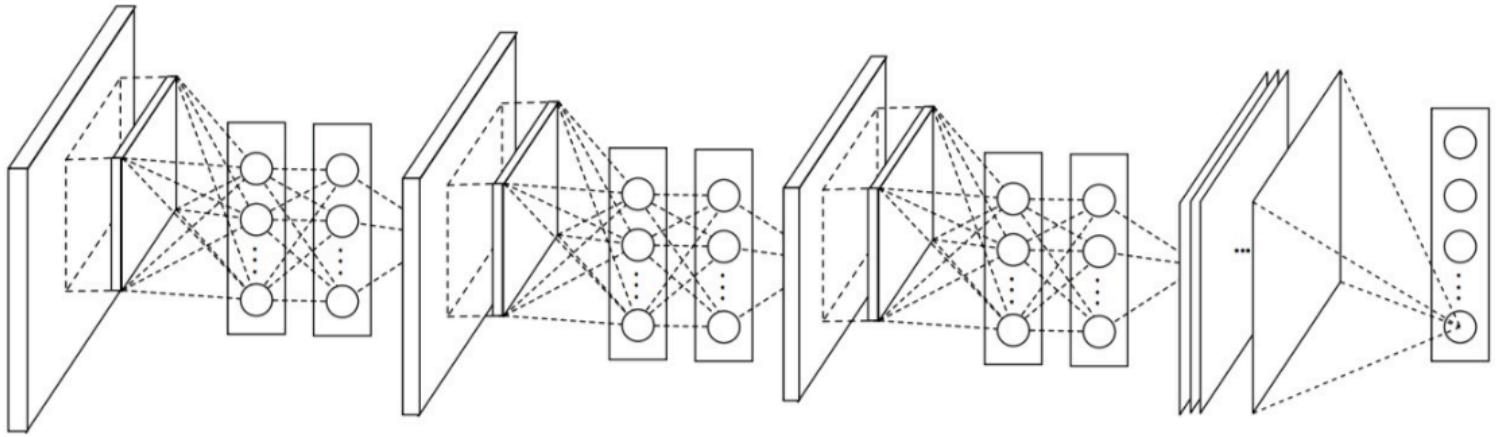
NIN



(a) Linear convolution layer



(b) Mlpconv layer

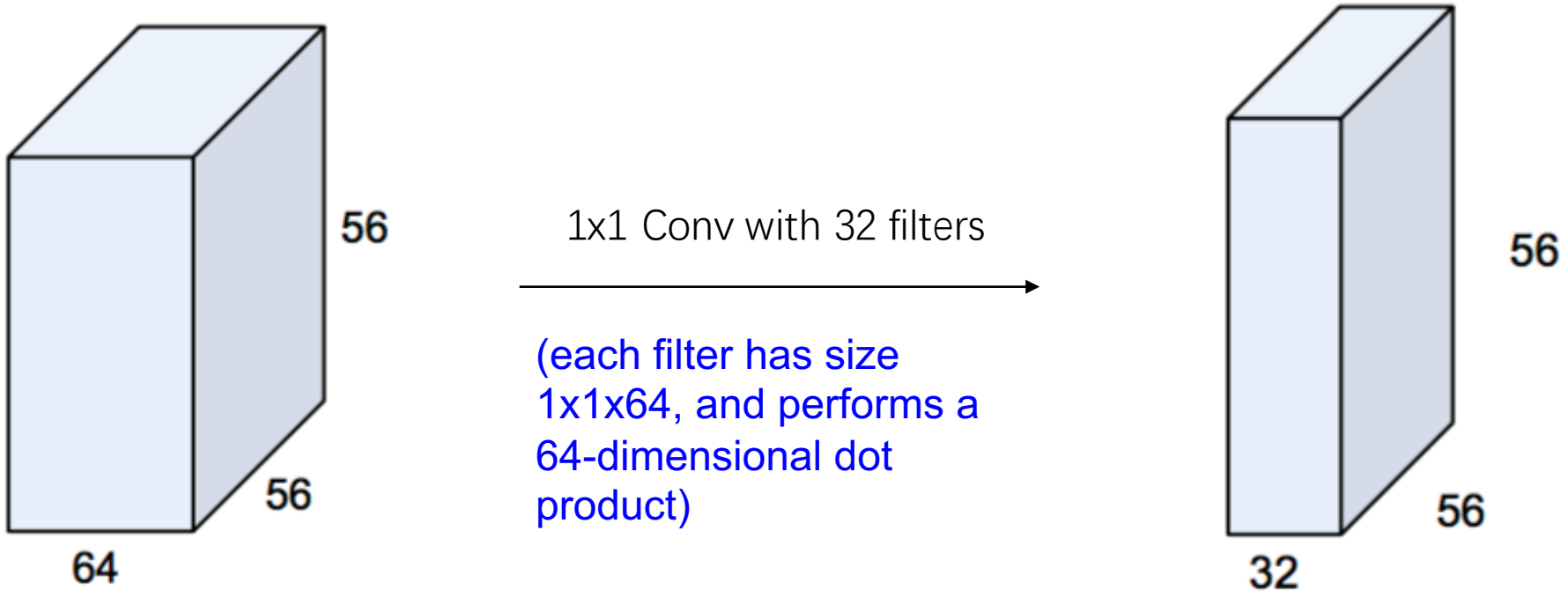


Conv = GLM **MLPConv = 1×1 Conv**

GLM is replaced with a "micro network" structure

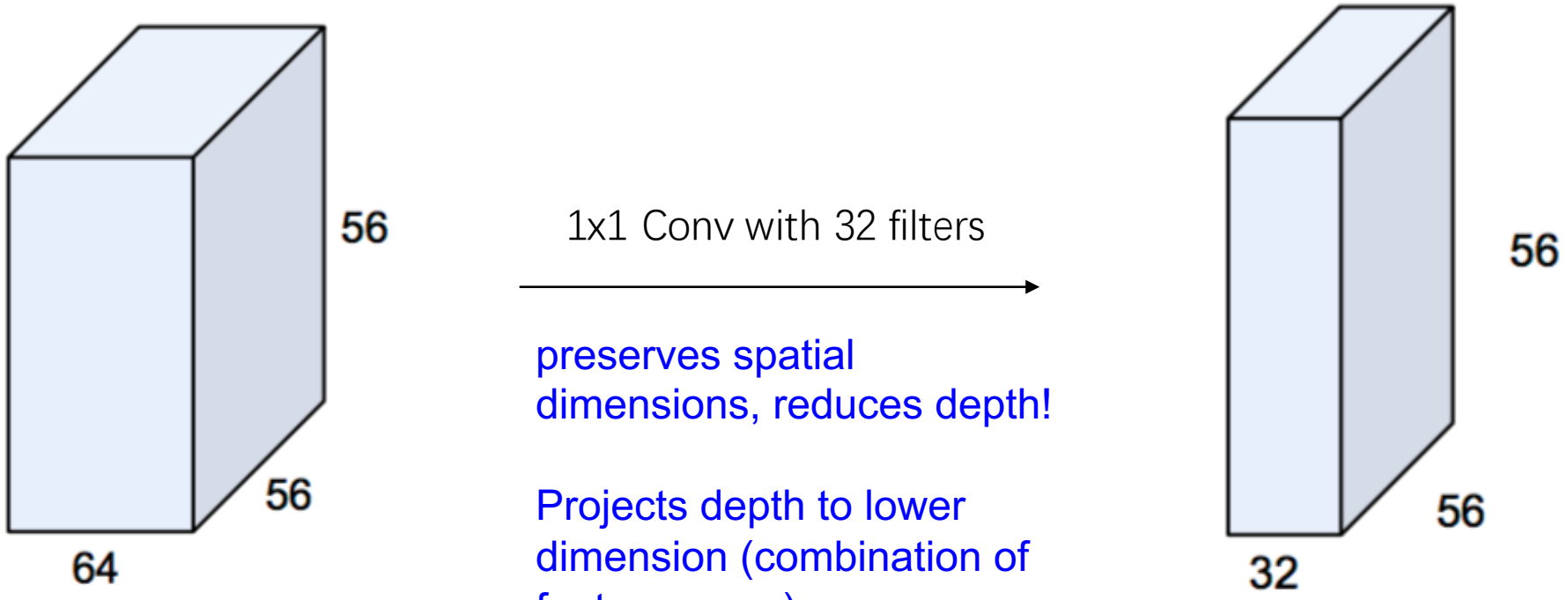
Inception

1x1 Conv



Inception

1x1 Conv

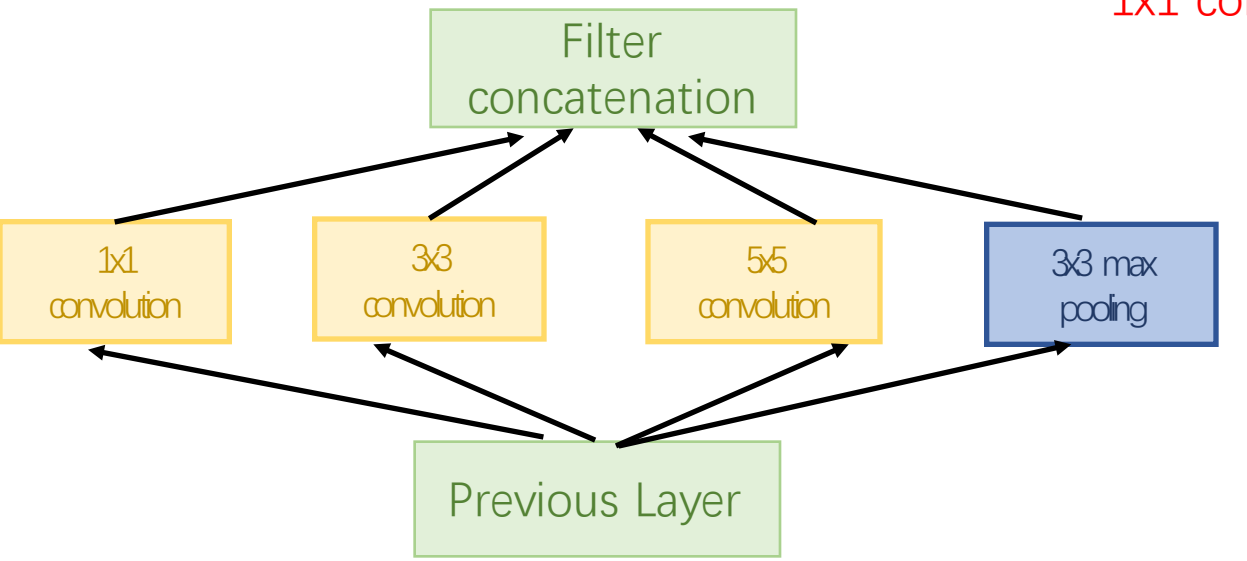


preserves spatial dimensions, reduces depth!

Projects depth to lower dimension (combination of feature maps)

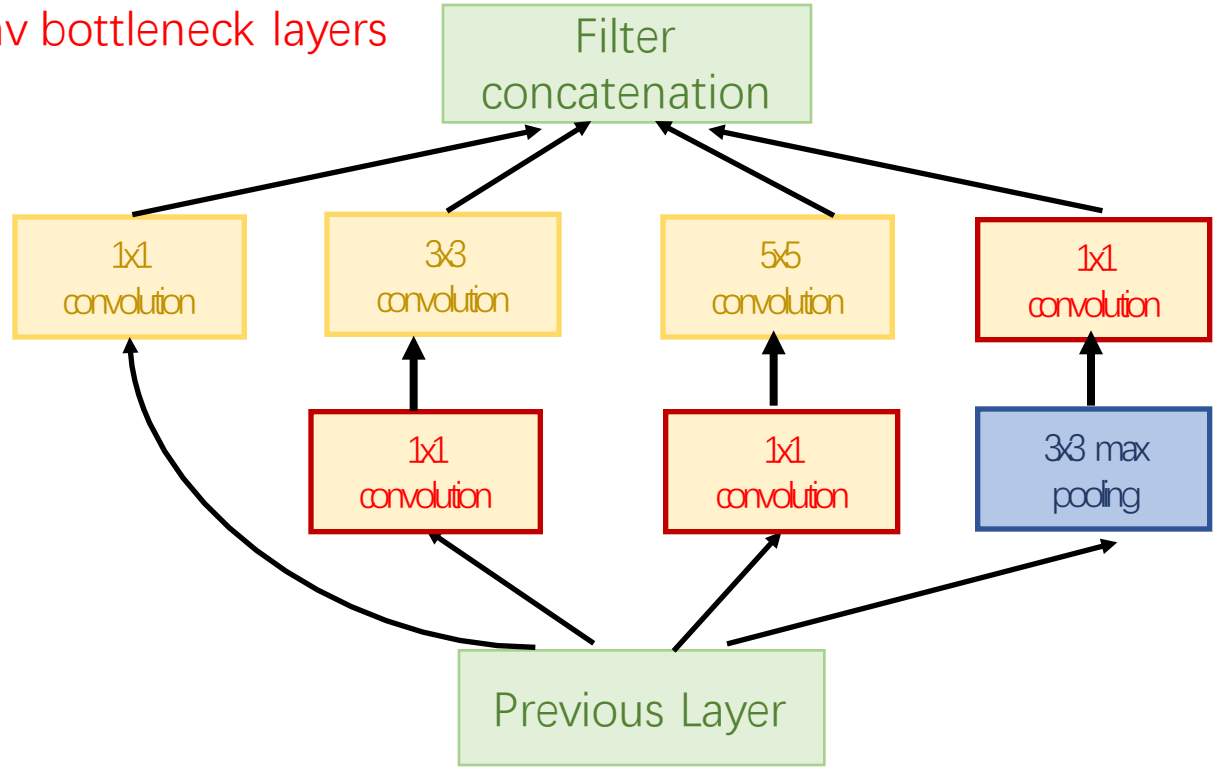
Inception

Revolution



Naïve Inception module

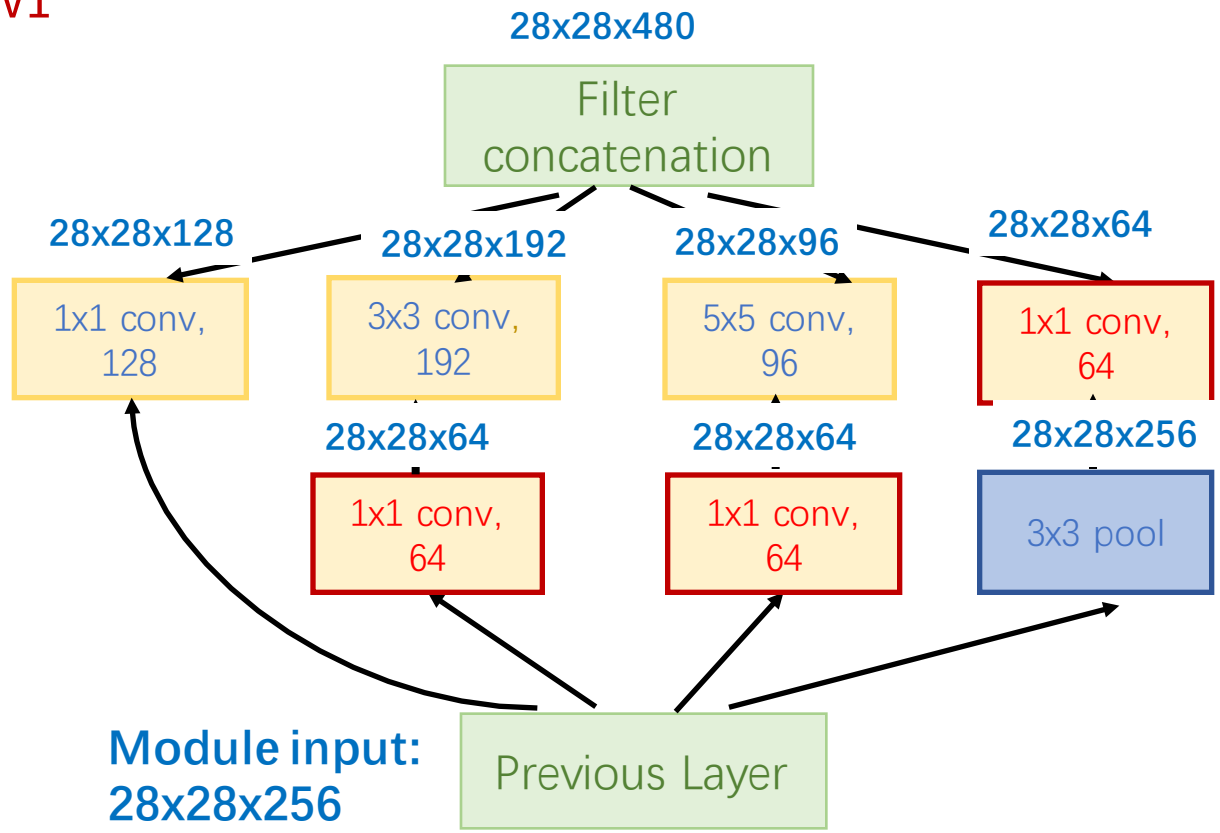
1x1 conv bottleneck layers



Inception module with dimension reduction

Inception

V1



Inception module with dimension reduction

Conv Ops:

[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256

Total: 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

Inception

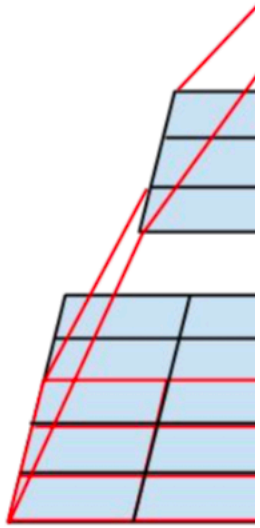
V1- GoogLeNet

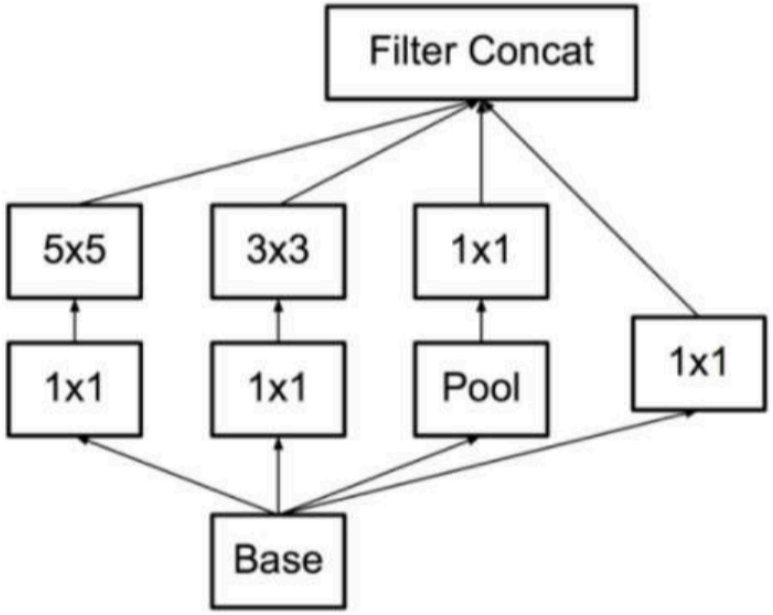
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture.

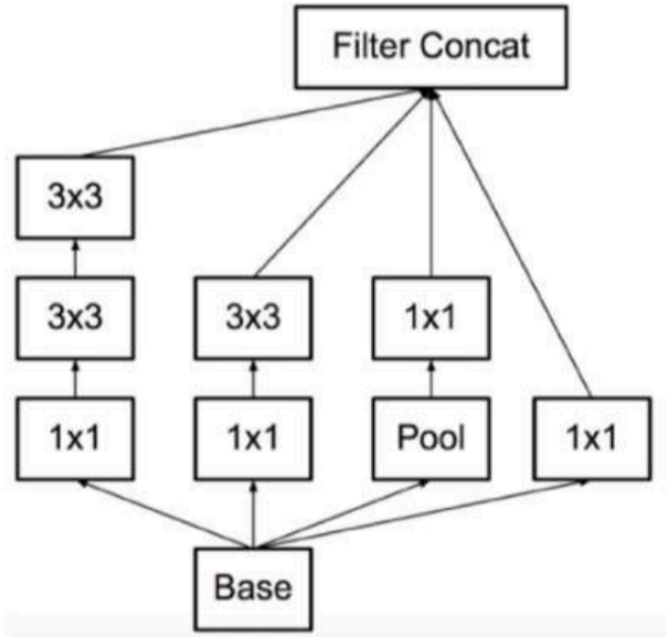
Inception

V2 > 5x5---3x3





original inception module



factorizing inception module

Inception

V2 > BN

(1) 可以使用更高的学习率。如果每层的scale不一致，实际上每层需要的学习率是不一样的，同一层不同维度的scale往往也需要不同大小的学习率，通常需要使用最小的那个学习率才能保证损失函数有效下降，Batch Normalization将每层、每维的scale保持一致，那么我们就可以直接使用较高的学习率进行优化。

(2) 移除或使用较低的dropout。dropout是常用的防止overfitting的方法，而导致overfit的位置往往在数据边界处，如果初始化权重就已经落在数据内部，overfit现象就可以得到一定的缓解。论文中最后的模型分别使用10%、5%和0%的dropout训练模型，与之前的40%-50%相比，可以大大提高训练速度。

(3) 降低L2权重衰减系数。还是一样的问题，边界处的局部最优往往有几维的权重（斜率）较大，使用L2衰减可以缓解这一问题，现在用了Batch Normalization，就可以把这个值降低了，论文中降低为原来的5倍。

(4) 取消Local Response Normalization层。由于使用了一种Normalization，再使用LRN就显得没那么必要了。而且LRN实际上也没那么work。

(5) 减少图像扭曲的使用。由于现在训练epoch数降低，所以要对输入数据少做一些扭曲，让神经网络多看看真实的数据。

https://blog.csdn.net/happynear/article/details/44238541

Inception

V2

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	double #3×3 reduce	double #3×3	Pool +proj
convolution*	7×7/2	112×112×64	1						
max pool	3×3/2	56×56×64	0						
convolution	3×3/1	56×56×192	1		64	192			
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	3	64	64	64	64	96	avg + 32
inception (3b)		28×28×320	3	64	64	96	64	96	avg + 64
inception (3c)	stride 2	28×28×576	3	0	128	160	64	96	max + pass through
inception (4a)		14×14×576	3	224	64	96	96	128	avg + 128
inception (4b)		14×14×576	3	192	96	128	96	128	avg + 128
inception (4c)		14×14×576	3	160	128	160	128	160	avg + 128
inception (4d)		14×14×576	3	96	128	192	160	192	avg + 128
inception (4e)	stride 2	14×14×1024	3	0	128	192	192	256	max + pass through
inception (5a)		7×7×1024	3	352	192	320	160	224	avg + 128
inception (5b)		7×7×1024	3	352	192	320	192	224	max + 128
avg pool	7×7/1	1×1×1024	0						

Inception

V3

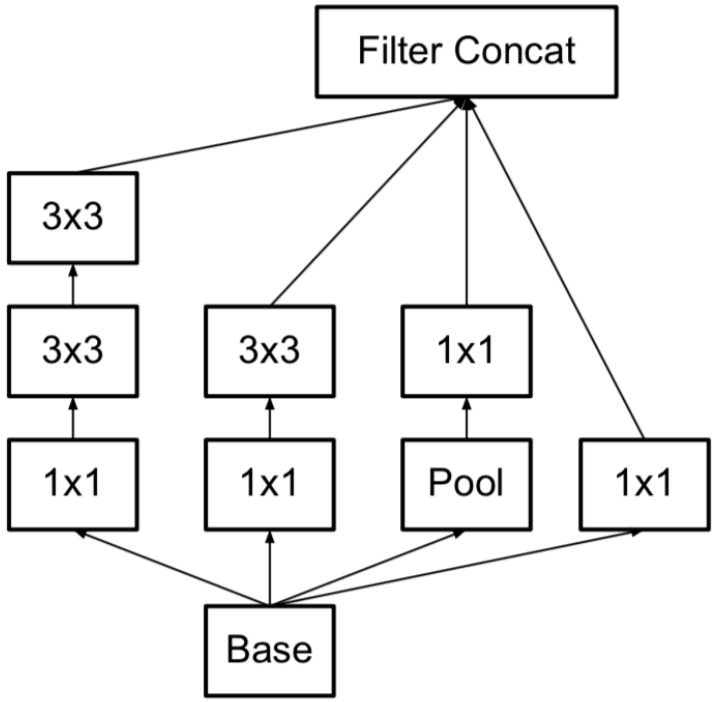


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

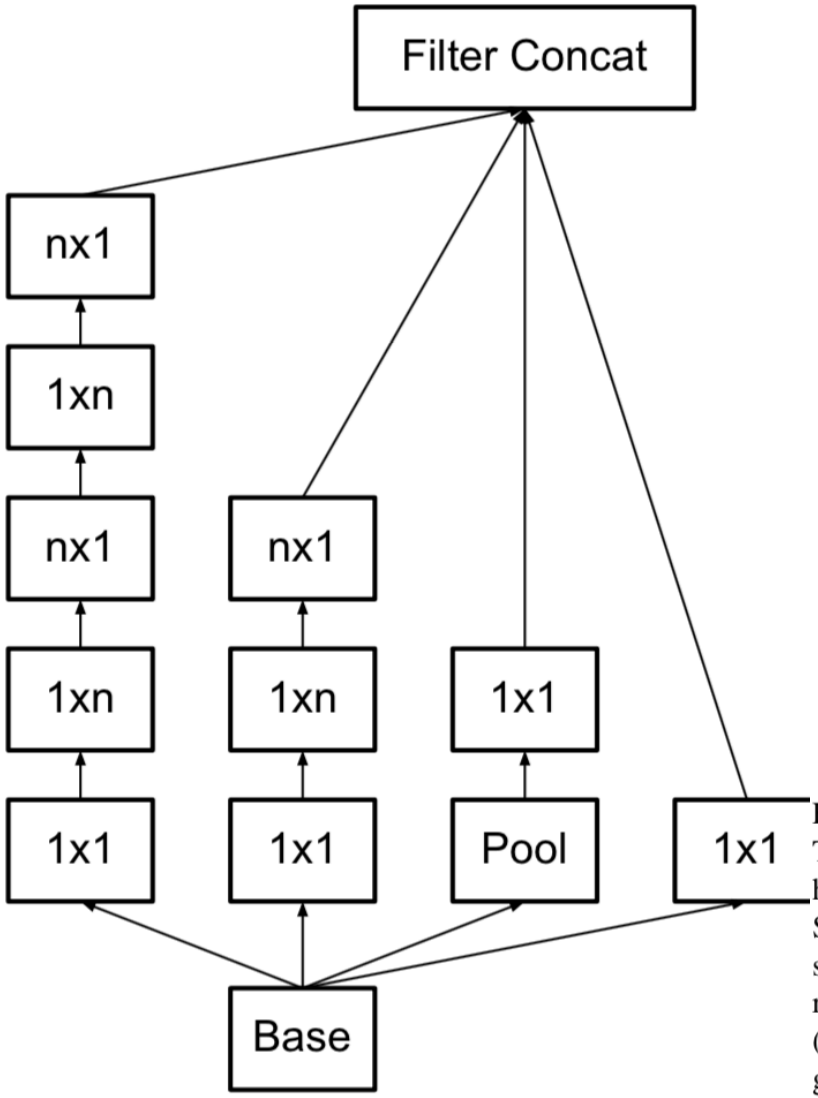


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

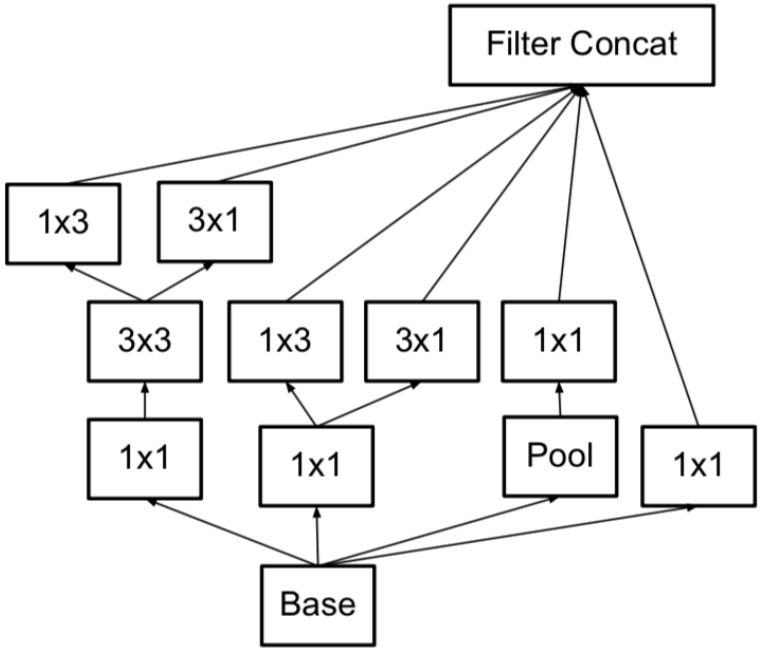


Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing high dimensional sparse representation is the most critical as the ratio of local processing (by 1×1 convolutions) is increased compared to the spatial aggregation.

Inception

V3

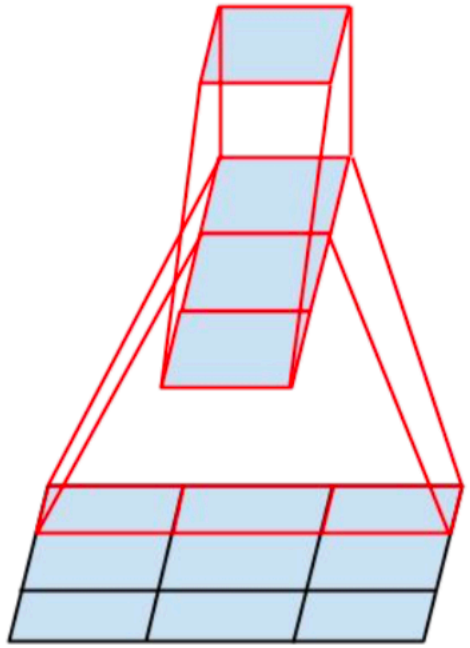


Figure 3. Mini-network replacing the 3×3 convolutions. The lower layer of this network consists of a 3×1 convolution with 3 output units.

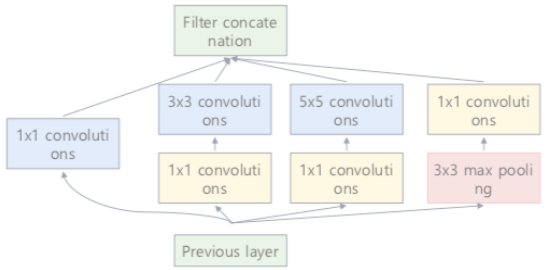
v3一个最重要的改进是分解（Factorization），将7x7分解成两个一维的卷积（1x7,7x1），3x3也是一样（1x3,3x1），这第一个样的好处，既可以加速计算（多余的计算能力可以用来加深网络），又可以将1个conv拆成2个conv，使得网络深度进一步增加，增加了网络的非线性，还有值得注意的地方是网络输入从224x224变为了299x299，更加精细设计了35x35/17x17/8x8的模块；

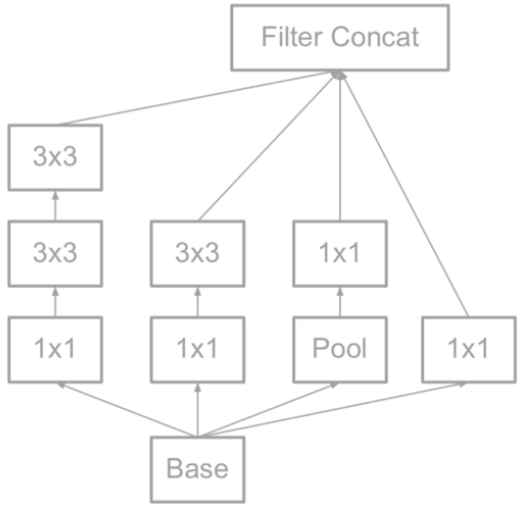
Inception

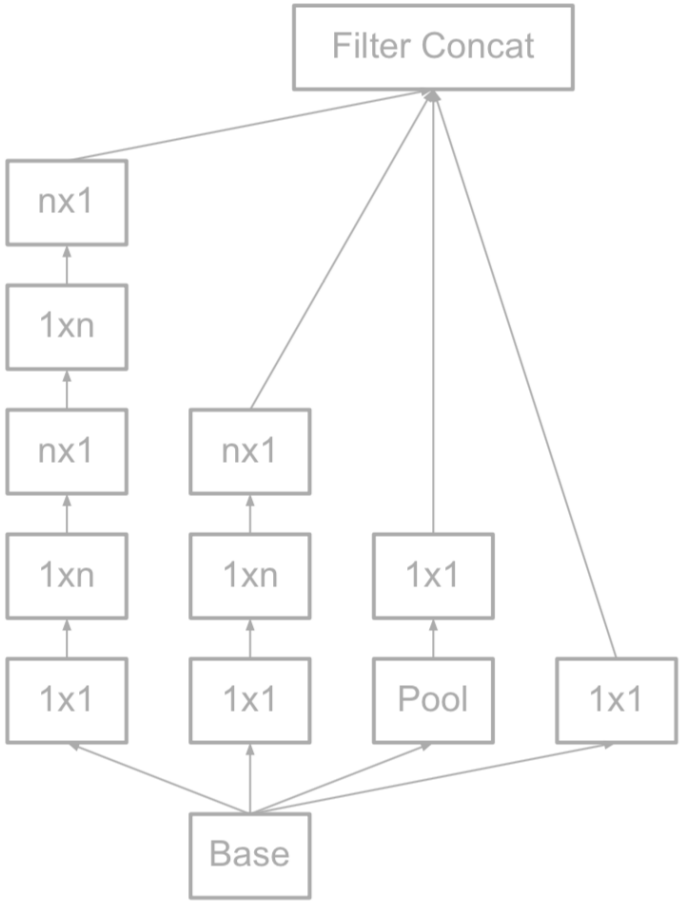
V3

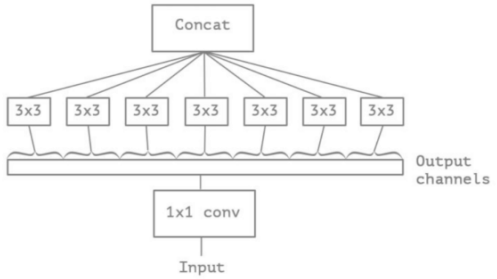
type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Table 1. The outline of the proposed network architecture. The output size of each module is the input size of the next one. We are using variations of reduction technique depicted Figure 10 to reduce the grid sizes between the Inception blocks whenever applicable. We have marked the convolution with 0-padding, which is used to maintain the grid size. 0-padding is also used inside those Inception modules that do not reduce the grid size. All other layers do not use padding. The various filter bank sizes are chosen to observe principle 4 from Section 2.









Inceptions

Thanks for watching!

薛飞飞

6th July, 2018