CS867 Computer Vision
Fall 2021

Lecture 10
Convolutional Neural Networks Architectures

Dr. Muhammad Moazam Fraz
Associate Professor
Department of Computing
NUST SEECS

Web : Lab

In the last class

Convolutional Neural Network

- Generalized Feature Representation for Visual Recognition
- From Fully Connected to Convolutional Neural Network
- A toy-conv net architecture
- Building blocks of a CNN

More on CNNs

What we covered in the last class

CNN Parameters

Alex Net

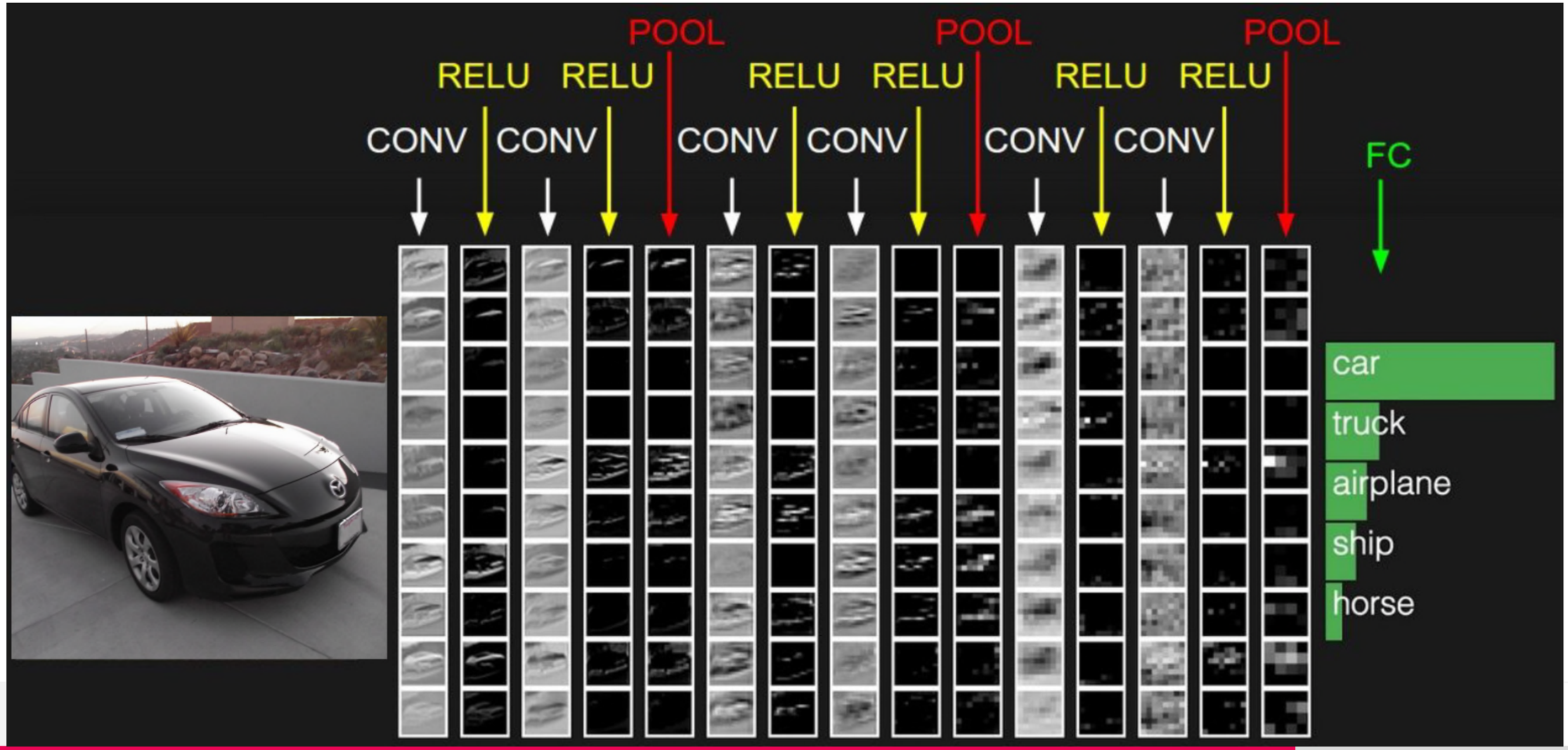
VGG Net

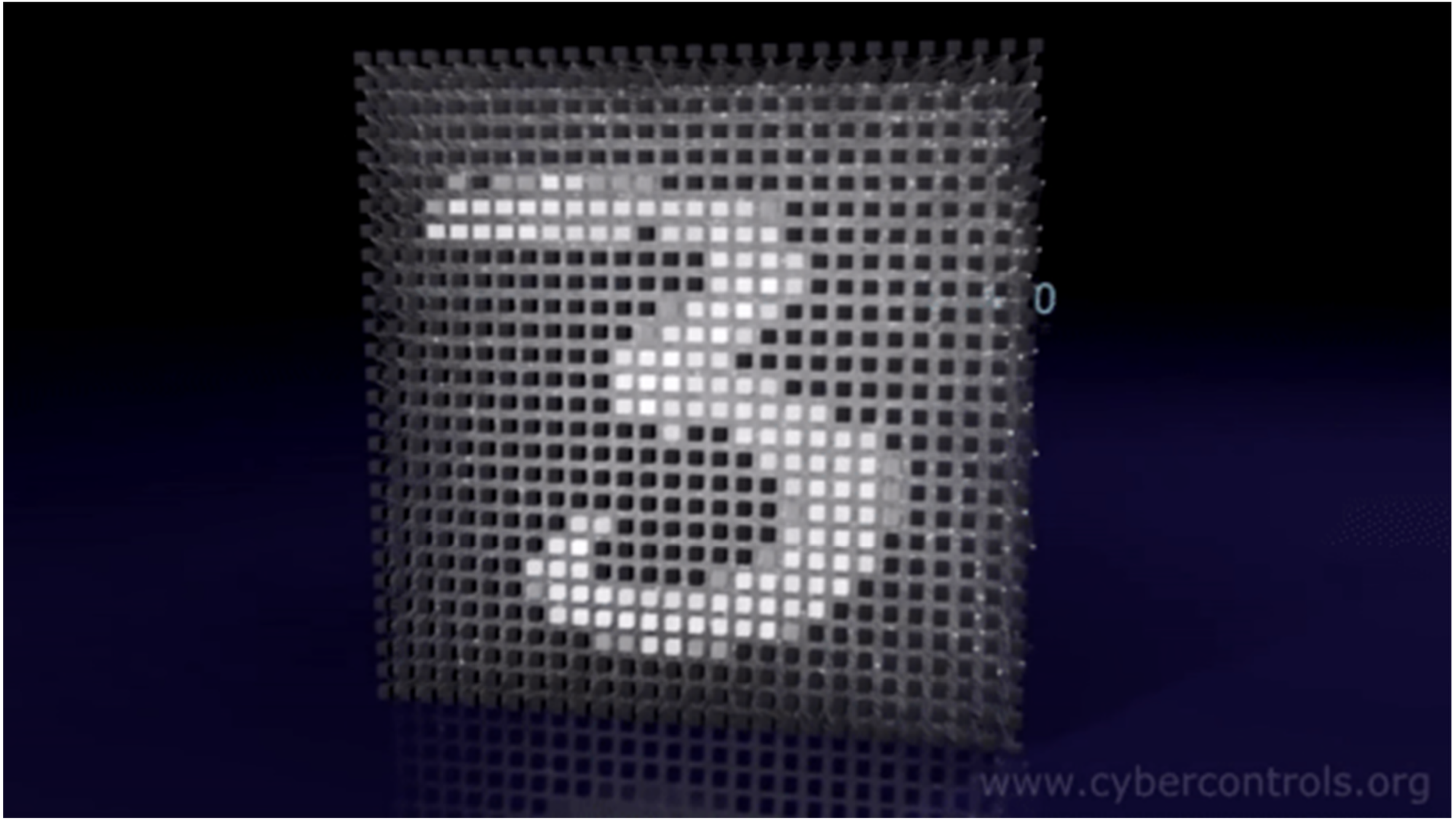
GoogleNet

CNN : Progressively aggregating spatial information to reach a global decision

Local features are extracted and aggregated throughout the network

The last layers "sees" the entire image and the features **encode global** information





Neural Network Simulation credit to Denis Dmitriev

Why silence

- Models had too many parameters
- They overfit for small datasets
- We did not have very large datasets
- Even for large sets, we did not have enough computation power to train the models until…

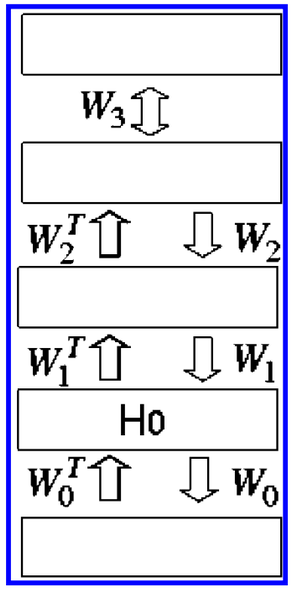
General purpose Graphical Processing Units (GPUs)
Allowed parallel processing

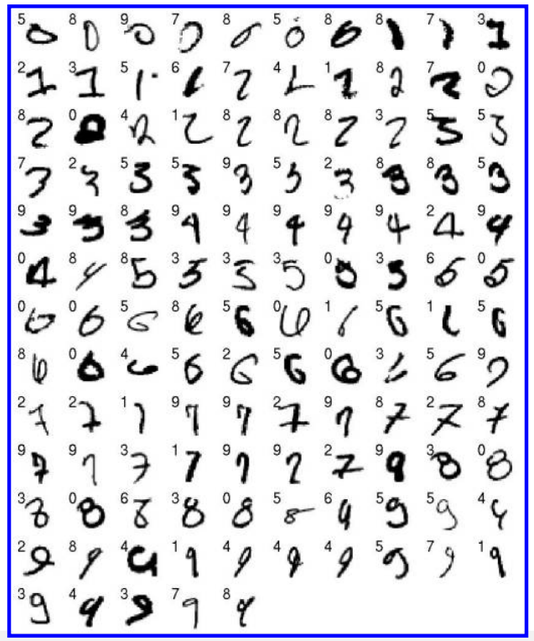
Then first this happened in 2006

A Fast Learning Algorithm for Deep Belief Nets

Geoffrey E. Hinton
hinton@cs.toronto.edu
Simon Osindero
osindero@cs.toronto.edu
Department of Computer Science, University of Toronto, Toronto, Canada M5S 3G4

Yee-Whye Teh
tehyw@comp.nus.edu.sg
Department of Computer Science, National University of Singapore, Singapore 117543





- A fast algorithm to train deep belief networks by stacking restricted Boltzmann machines
- Layer-wise pre-training with contrastive divergence
- Set a state-of-the-art performance on MNIST
- However, this did not use GPUs yet

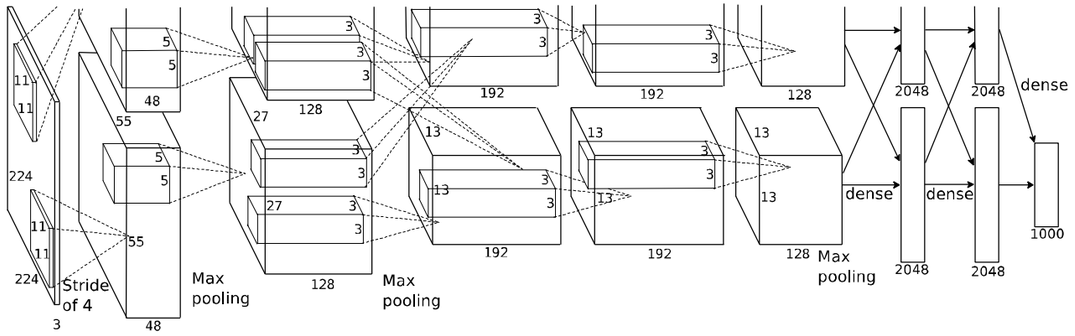
Then in 2012

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

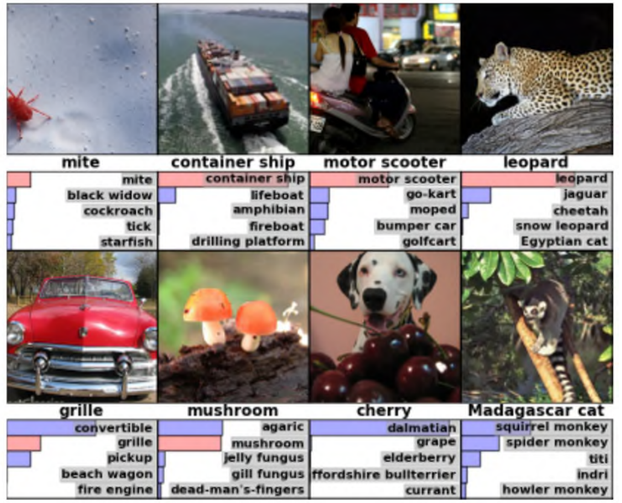
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

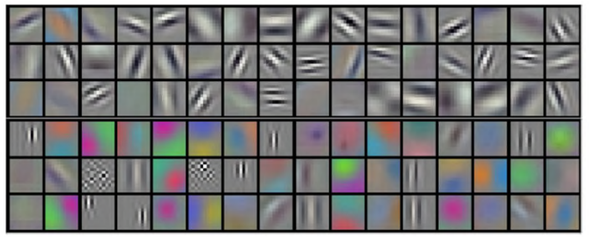
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca



Network

- Krizhevsky et al. almost halved the error rate in the ImageNet challenge
- A simple CNN





First layer filters 11x11

A word about the ImageNet challenge

http://www.image-net.org/

ImageNet: A Large-Scale Hierarchical Image Database

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei
Dept. of Computer Science, Princeton University, USA
{jiadeng, wdong, rsocher, jial, li, feifeili}@cs.princeton.edu

[CVPR 2009]

International Journal of Computer Vision
December 2015, Volume 115, Issue 3, pp 211–252 | Cite as

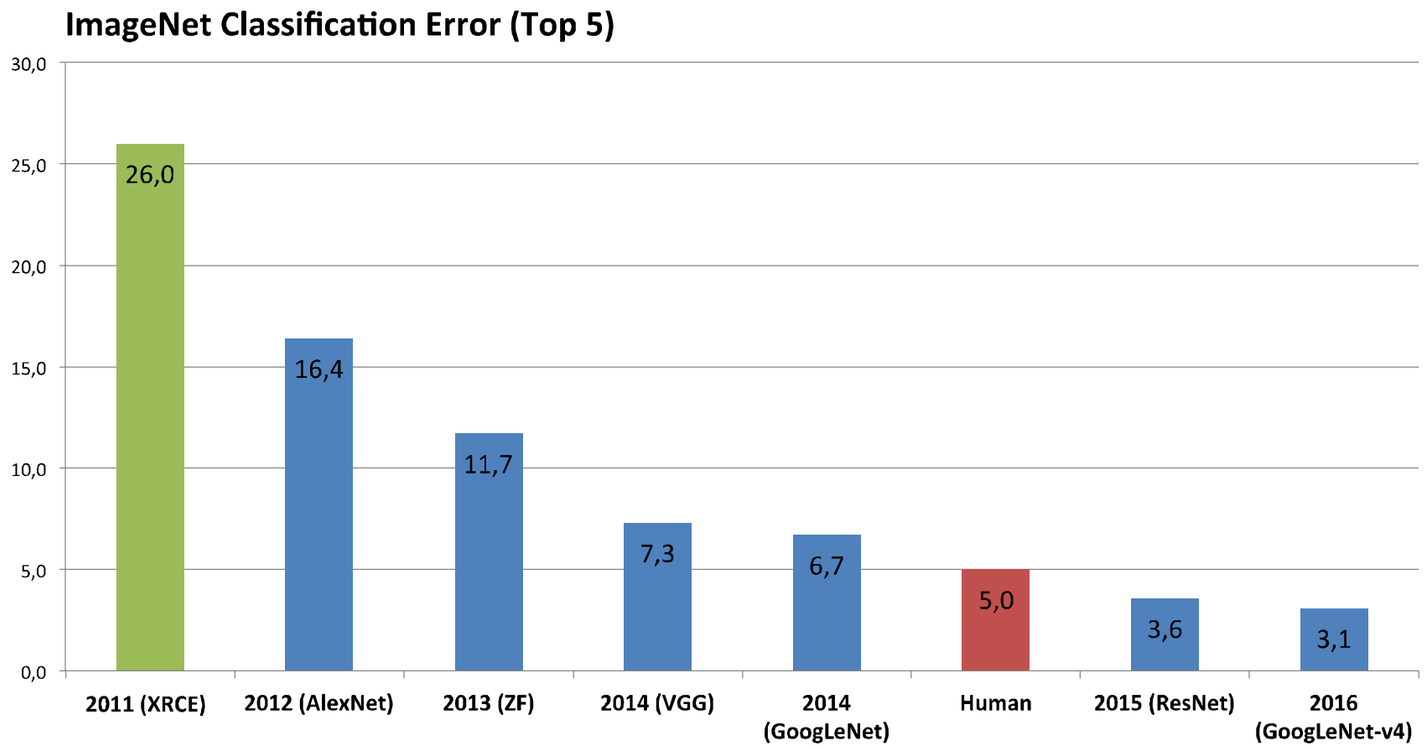
ImageNet Large Scale Visual Recognition Challenge

Authors Authors and affiliations

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei

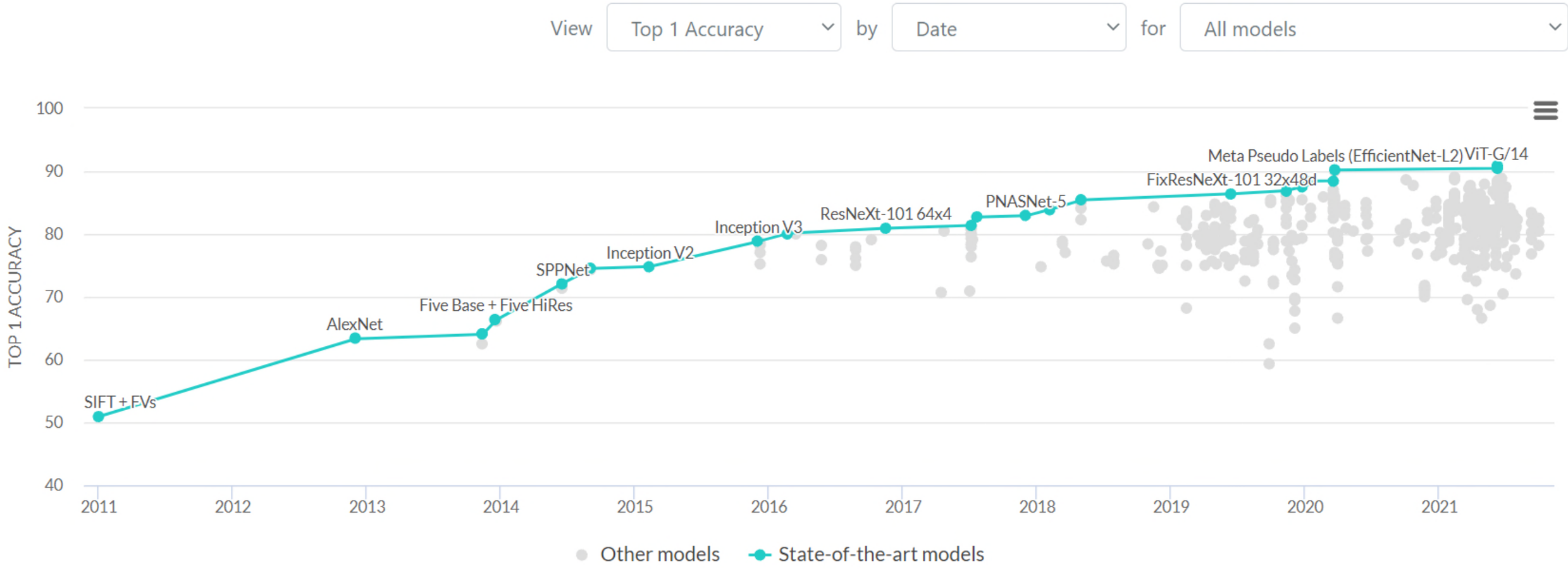
- Large scale object recognition challenge started in 2010
- 1.2 million training images
- 1000 object categories
- 200k validation and test images
- 2017 – 3 challenges:
  - Object localization
  - Object detection
  - Object detection from video

Historical evolution of the ImageNet Challenge



Historical evolution of the ImageNet Challenge

Leaderboard Dataset

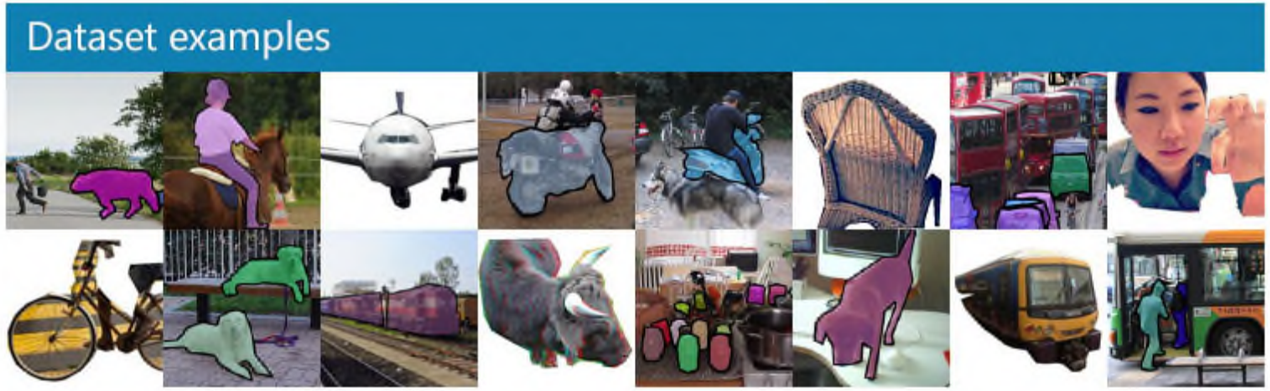


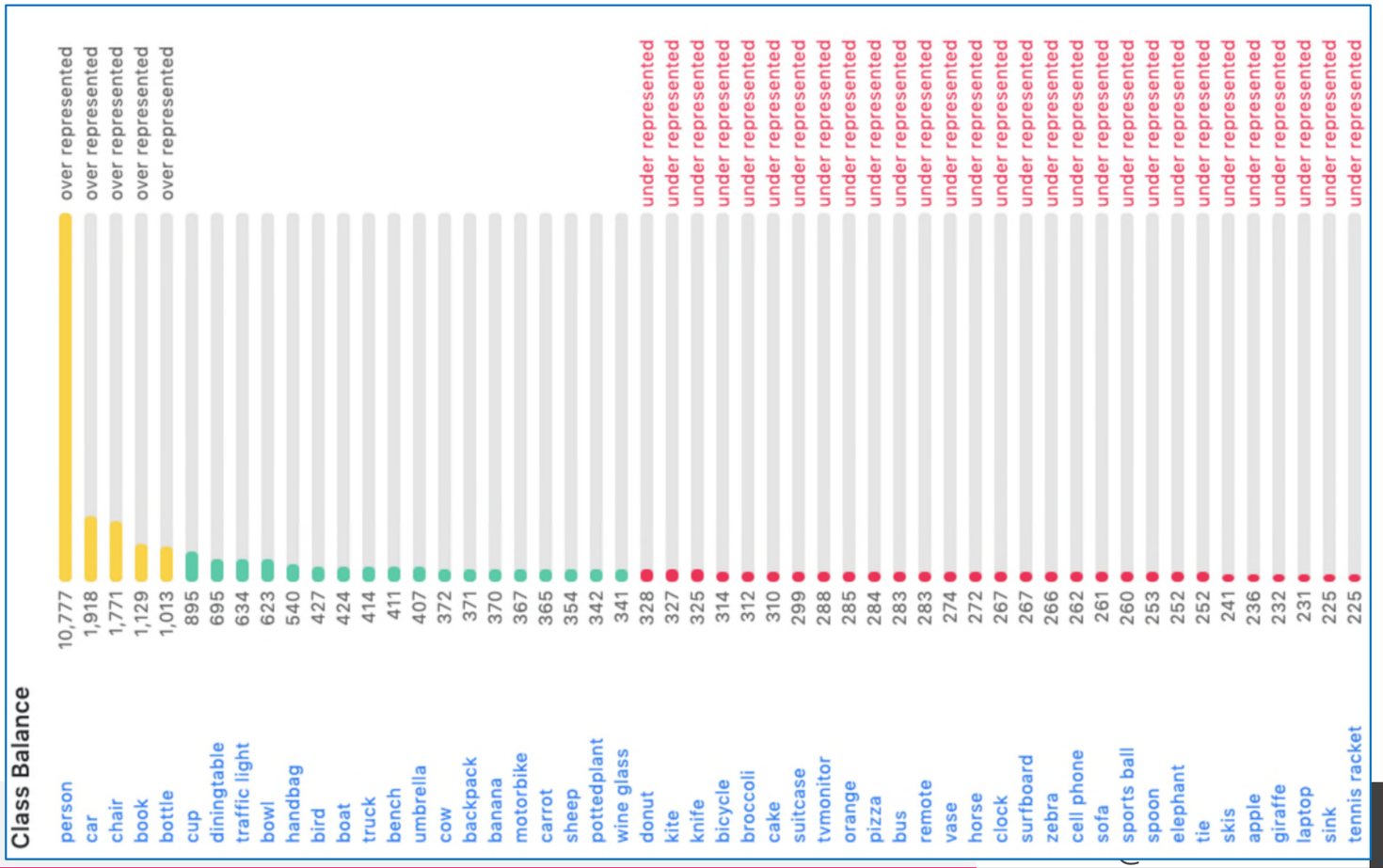
Historical evolution of the ImageNet Challenge

Rank	Model	Top 1 Accuracy	Top 5 Accuracy	Number of params	Extra Training Data	Paper	Code	Result	Year	Tags
1	**CoAtNet-7**	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
2	**ViT-G/14**	90.45%		1843M	✓	Scaling Vision Transformers			2021	Transformer JFT-3B
3	**CoAtNet-6**	90.45%		1470M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
4	**ViT-MoE-15B** (Every-2)	90.35%		14700M	✓	Scaling Vision with Sparse Mixture of Experts			2021	Transformer JFT-3B
5	**Meta Pseudo Labels** (EfficientNet-L2)	90.2%	98.8	480M	✓	Meta Pseudo Labels			2020	EfficientNet JFT-300M
6	**Meta Pseudo Labels** (EfficientNet-B6-Wide)	90%	98.7	390M	✓	Meta Pseudo Labels			2020	EfficientNet JFT-300M

MS COCO Data set

- COCO is a large-scale object detection, segmentation, and captioning dataset.
- COCO has several features:
- Object segmentation
- Recognition in context
- Superpixel stuff segmentation
- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image
- 250,000 people with keypoints





https://viso.ai/computer-vision/coco-dataset/

Open Images Dataset V6

- Open Images is a dataset of ~9M images annotated with
  - image-level labels
  - object bounding boxes
  - object segmentation masks
  - visual relationships
  - localized narratives:

  - 15,851,536 boxes on 600 categories
  - 2,785,498 instance segmentations on 350 categories
  - 3,284,280 relationship annotations on 1,466 relationships
  - 675,155 localized narratives
  - 59,919,574 image-level labels on 19,957 categories
  - Extension - 478,000 crowdsourced images with 6,000+ categories

- **LVIS Challenge** 2021
  - A new dataset for long tail object recognition.
  - https://www.lvisdataset.org/challenge_2021

LVIS Dataset

v1.0

Training set

- 1,270,141 instances (1 GB)
- 100,170 images (18 GB)

Validation set

- 244,707 instances (192 MB)
- 19,809 images (1 GB)

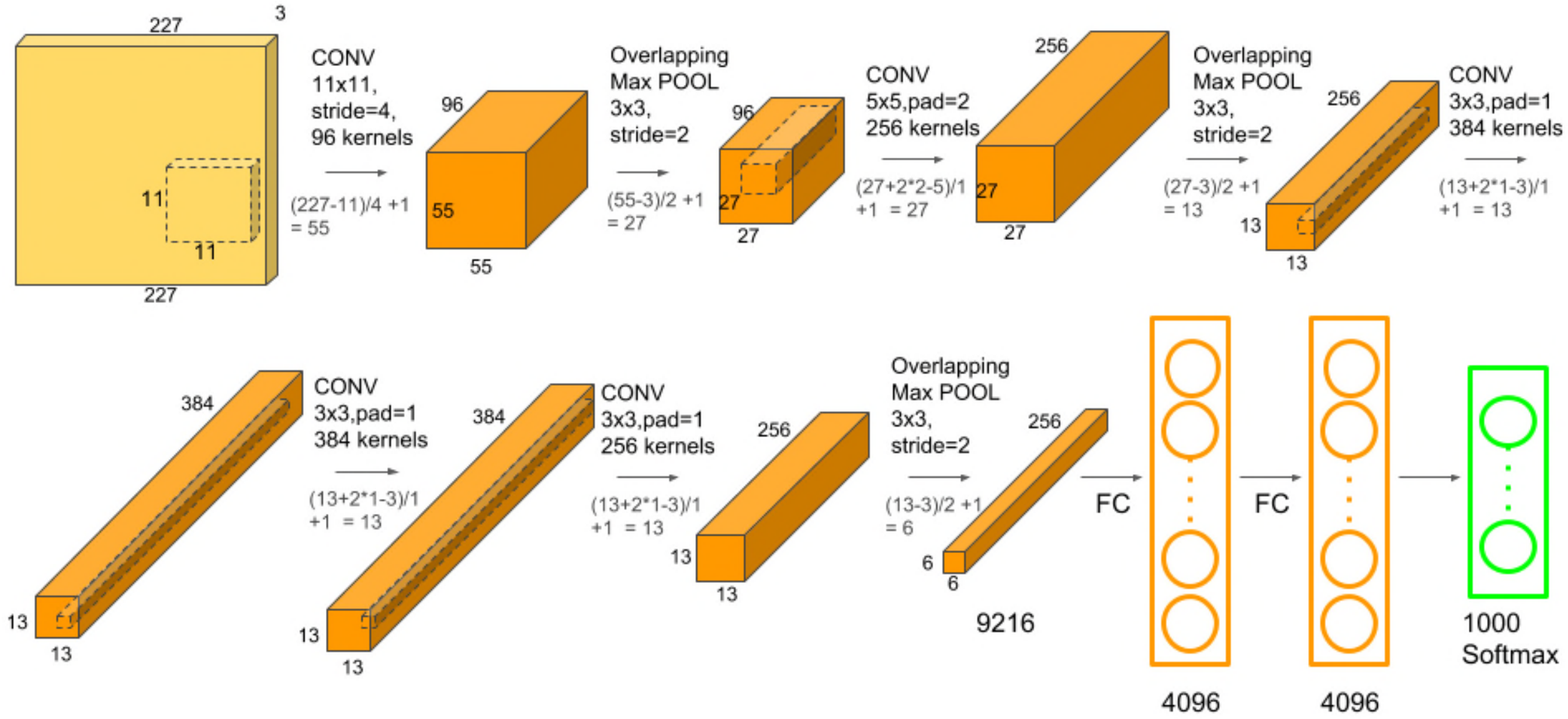
Test Dev

- info (4 MB)
- 19,822 images (6 GB)

Test Challenge

- info (4 MB)
- 19,822 images (6 GB)

Alex Net (2012)



Size of the Output Tensor (Image) of a Conv Layer

Let us define

O = Size (width) of output image.
I = Size (width) of input image.
K = Size (width) of kernels used in the Conv Layer.
N = Number of kernels.
S = Stride of the convolution operation.
P = Padding.

The size (O) of the output image is given by

$$O = \frac{I - K + 2P}{S} + 1$$

The number of channels in the output image is equal to the number of kernels

Size of the Output Tensor (Image) of a Conv Layer

- In AlexNet, the input image is of size 227x227x3.
- The first convolutional layer has 96 kernels of size 11x11x3.
- The stride is 4 and padding is 0.
- Therefore the size of the output image right after the first bank of convolutional layers is

$$O = \frac{227 - 11 + 2 \times 0}{4} + 1 = 55$$

So, the output image is of size 55x55x96 (one channel for each kernel)

Size of Output Tensor (Image) of a MaxPool Layer

Let's define

O = Size (width) of output image.
I = Size (width) of input image.
S = Stride of the convolution operation.
P_s = Pool size.

The size (O) of the output image is given by

$$O = \frac{I - P_s}{S} + 1$$

In AlexNet, the MaxPool layer after the bank of convolution filters has a pool size of 3 and stride of 2.

The image at this stage is of size 55x55x96.

The output image after the MaxPool layer is of size

$$O = \frac{55 - 3}{2} + 1 = 27$$

The output image is of size 27x27x96

A fully connected layer outputs a vector of length equal to the number of neurons in the layer

Number of Parameters of a Conv Layer

In a CNN, each layer has two kinds of parameters :

- weights
- biases

The total number of parameters is just the sum of all weights and biases.

Let's define,

W_c = Number of weights of the Conv Layer.
B_c = Number of biases of the Conv Layer.
P_c = Number of parameters of the Conv Layer.
K = Size (width) of kernels used in the Conv Layer.
N = Number of kernels.
C = Number of channels of the input image.

$$W_c = K^2 \times C \times N$$
$$B_c = N$$
$$P_c = W_c + B_c$$

In a Conv Layer, the depth of every kernel is always equal to the number of channels in the input image. So every kernel has K^2 x C parameters, and there are N such kernels.
That's how we come up with the above formula.

Number of Parameters of a Conv Layer

In AlexNet, at the first Conv Layer,

- The number of channels (C) of the input image is 3,
- The kernel size (K) is 11
- The number of kernels (N) is 96.

So the number of parameters is given by

$$W_c = 11^2 \times 3 \times 96 = 34,848$$
$$B_c = 96$$
$$P_c = 34,848 + 96 = 34,944$$

There are no parameters associated with a MaxPool layer.

The pool size, stride, and padding are hyperparameters

You can verify the number of parameters for Conv-2, Conv-3, Conv-4, Conv-5 are 614656 , 885120, 1327488 and 884992 respectively

Number of Parameters of a Fully Connected (FC) Layer connected to a FC Layer

Let's define,

W_{ff} = Number of weights of a FC Layer which is connected to an FC Layer.
B_{ff} = Number of biases of a FC Layer which is connected to an FC Layer.
P_{ff} = Number of parameters of a FC Layer which is connected to an FC Layer.
F = Number of neurons in the FC Layer.
F_{-1} = Number of neurons in the previous FC Layer.

$$W_{ff} = F_{-1} \times F$$
$$B_{ff} = F$$
$$P_{ff} = W_{ff} + B_{ff}$$

In the above equation,

- F_{-1} x F is the total number of connection weights from neurons of the previous FC Layer the neurons of the current FC Layer.
- The total number of biases is the same as the number of neurons (F).

Example: The last fully connected layer of AlexNet is connected to an FC Layer.

For this layer, F_{-1} = 4096 and F = 1000. Therefore,

$$W_{ff} = 4096 \times 1000 = 4,096,000$$
$$B_{ff} = 1,000$$
$$P_{ff} = W_{ff} + B_{ff} = 4,097,000$$

Reading Material

- What AlexNet Brought To The World Of Deep Learning | by Richmond Alake | Towards Data Science

- AlexNet. Let's understand and code it! | by Abhishek Verma | Towards Data Science

- Number of Parameters and Tensor Sizes in a Convolutional Neural Network (CNN)
  - https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

- Learnable Parameters in a Convolutional Neural Network (CNN) explained
  - https://www.youtube.com/watch?v=gmBfb6LNnZs

IMAGENET : Large Scale Visual Recognition Challenge (ILSVRC)

- The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale.
- One high level motivation is to allow researchers to compare progress in detection across a wider variety of objects -- taking advantage of the quite expensive labeling effort.
- Another motivation is to measure the progress of computer vision for large scale image indexing for retrieval and annotation.

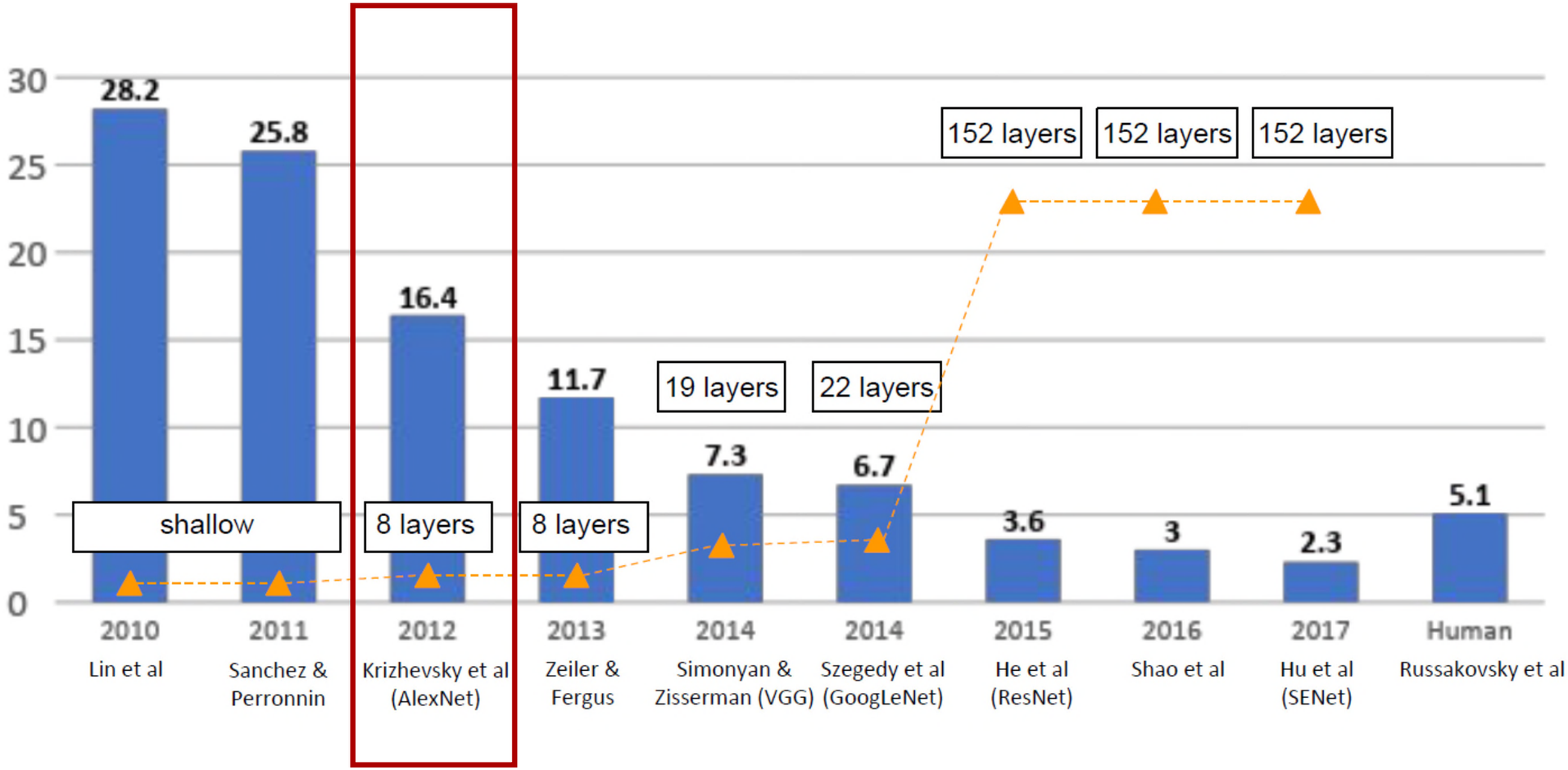
- For details about each challenge please refer to the corresponding page.
- ILSVRC 2017
- ILSVRC 2016
- ILSVRC 2015
- ILSVRC 2014
- ILSVRC 2013
- ILSVRC 2012
- ILSVRC 2011
- ILSVRC 2010

http://image-net.org/

https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/

Large Scale Visual Recognition Challenge winners : Recap

First CNN based winner : AlexNet



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

VGG Net

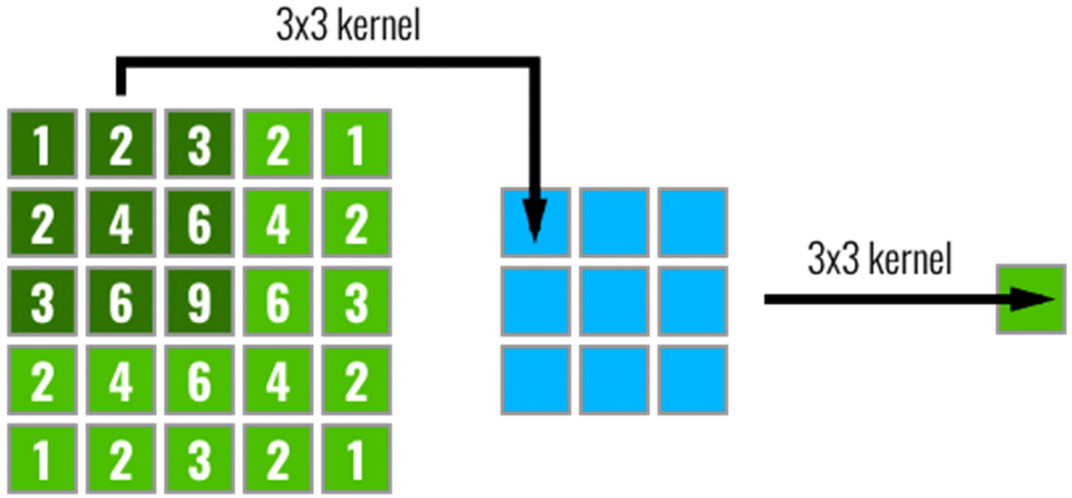
ILSVRC 2014 Runner-up

Receptive Field in CNNS

VGG Net

Receptive Field in CNN Layer

- The receptive field in Convolutional Neural Networks (CNN) is the **region of the input space** that affects a particular unit of the network.
  - Note that this input region can be not only the input of the network but also output from other units in the network, therefore this receptive field can be calculated relative to the input that we consider and also relative the unit that we are taking into consideration as the "receiver" of this input region.

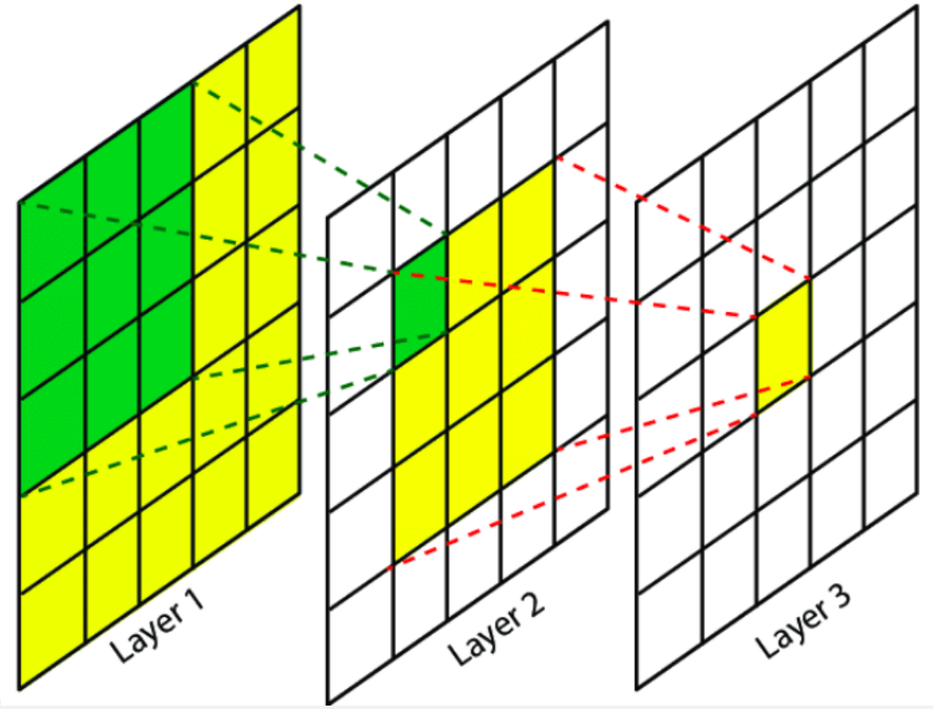


Receptive Field across 3 different layers using 3×3 filters.

The idea of the receptive field will help you dive into the architecture that you are using or developing

Receptive Field in CNN Layer

- Receptive Field (**RF**) is defined as **the size of the region in the input that produces the feature**[*].
  - Basically, it is a measure of association of an output feature (of any layer) to the input **region** (patch)
- The "receptive field" of a "neuron" in a layer would be the cross section of the previous layer from which neurons provide inputs

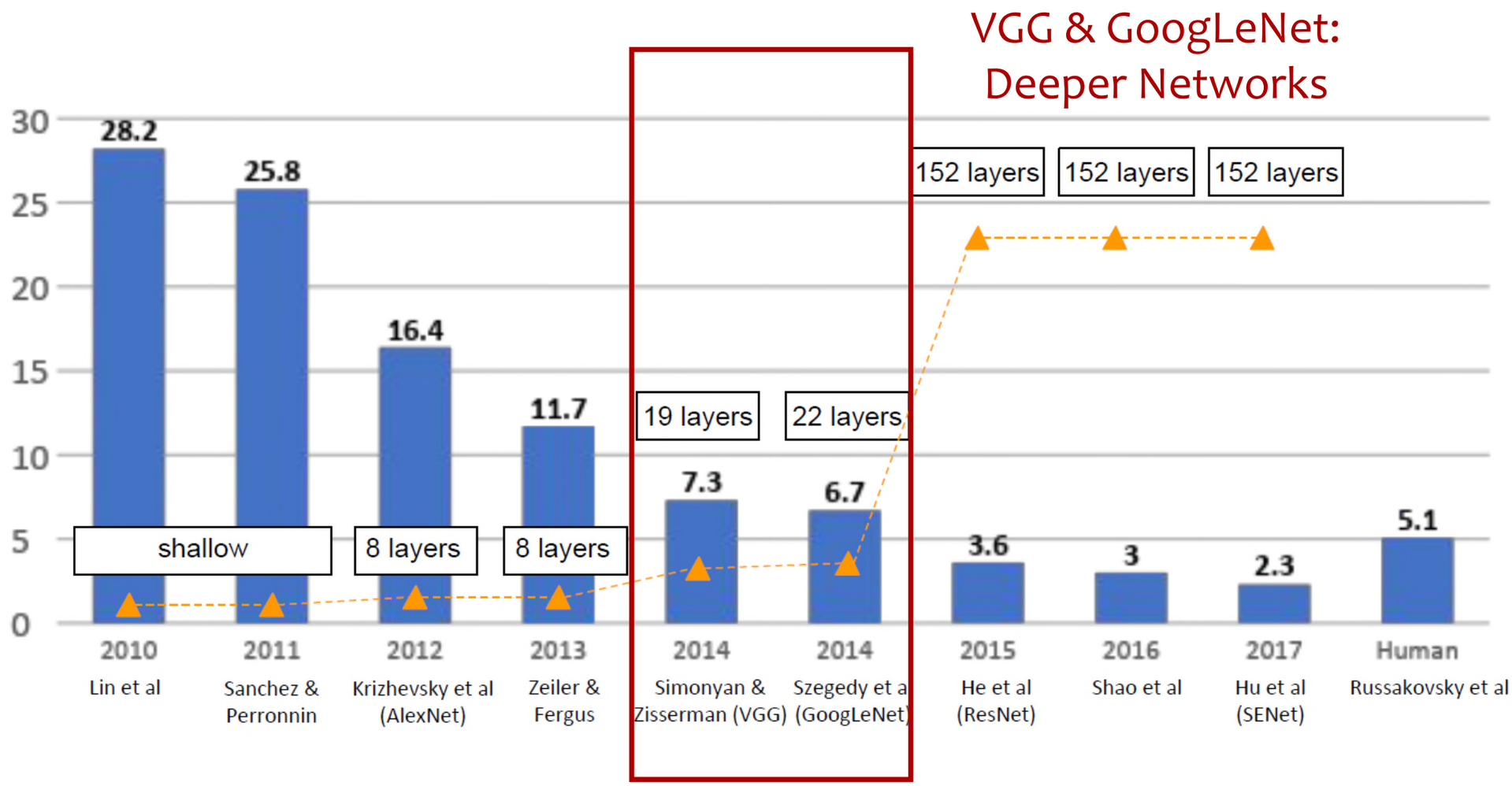


A convolutional unit only depends on a local region (patch) of the input.

That's why we never refer to the RF on fully connected layers since each unit has access to all the input region

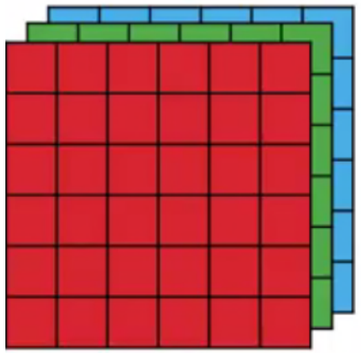
* Araujo, A., Norris, W., & Sim, J. (2019). Computing receptive fields of convolutional neural networks. Distill, 4(11), e21

Large Scale Visual Recognition Challenge winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

Case Study: VGGNet



224×224 ×3

→ [CONV 64 ×2] 224×224×64 → POOL 112×112 ×64 → [CONV 128] ×2 112×112 ×128 → POOL 56×56 ×128

→ [CONV 256] ×3 56×56 ×256 → POOL 28×28 ×256 → [CONV 512] × 3 28×28 ×512 → POOL 14×14×512

→ [CONV 512] ×3 14×14 ×512 → POOL 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

Case Study: VGGNet

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14

Q: Why use smaller filters? (3x3 conv)

Case Study: VGGNet

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14

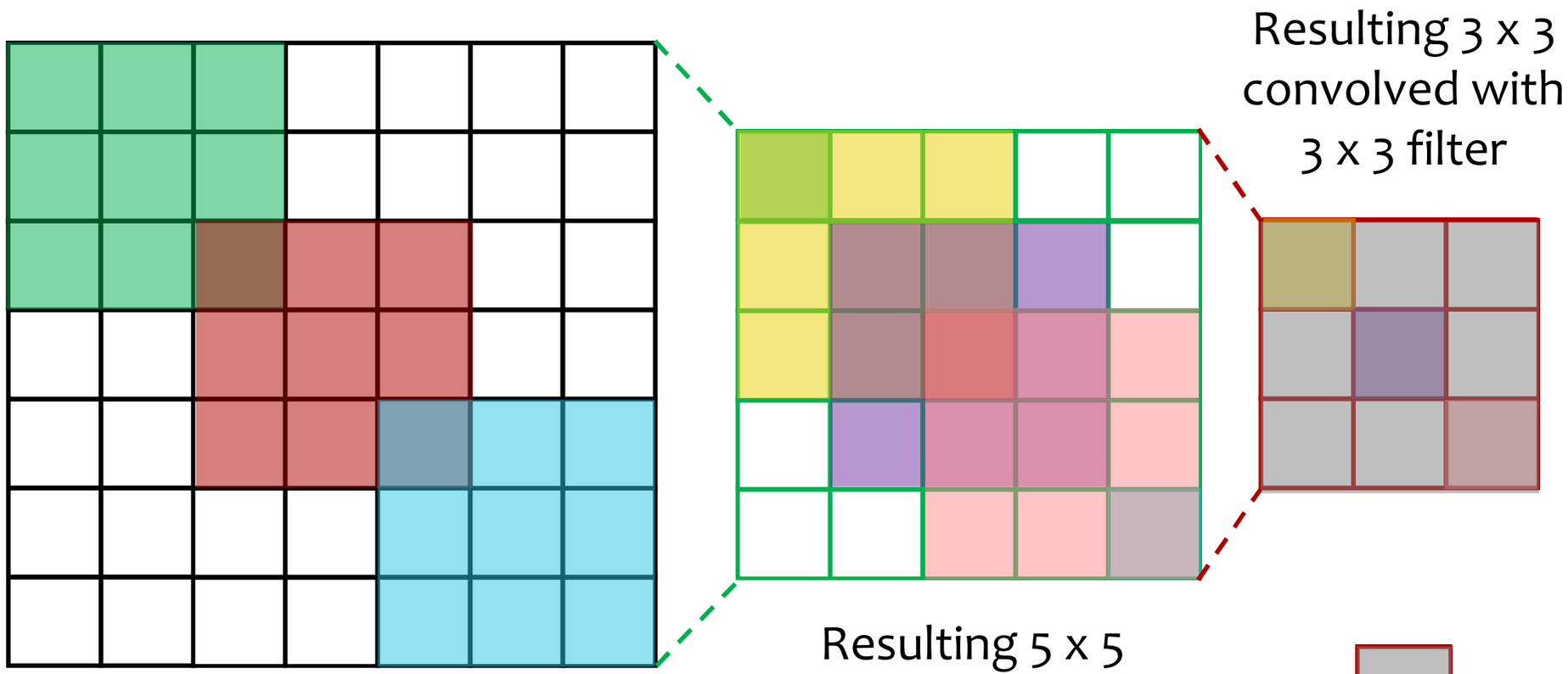
Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

And **fewer parameters**: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

C^2 if same depth is ensured



7 x 7 convolved with 3 x 3 filter

Resulting 5 x 5 convolved with 3 x 3 filter

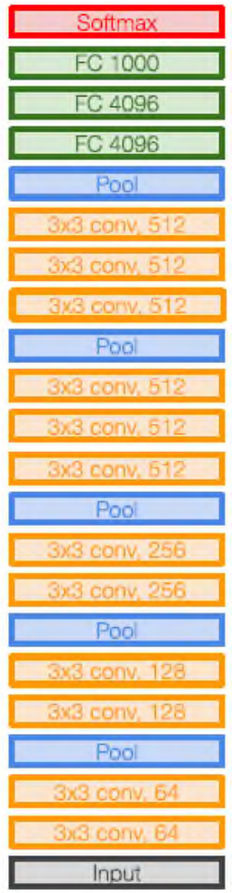
Resulting 3 x 3 convolved with 3 x 3 filter

Resulting 1 x 1 convolved with 3 x 3 filter

Case Study: VGGNet

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters



VGG16

Case Study: VGGNet

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

Most memory is in early CONV

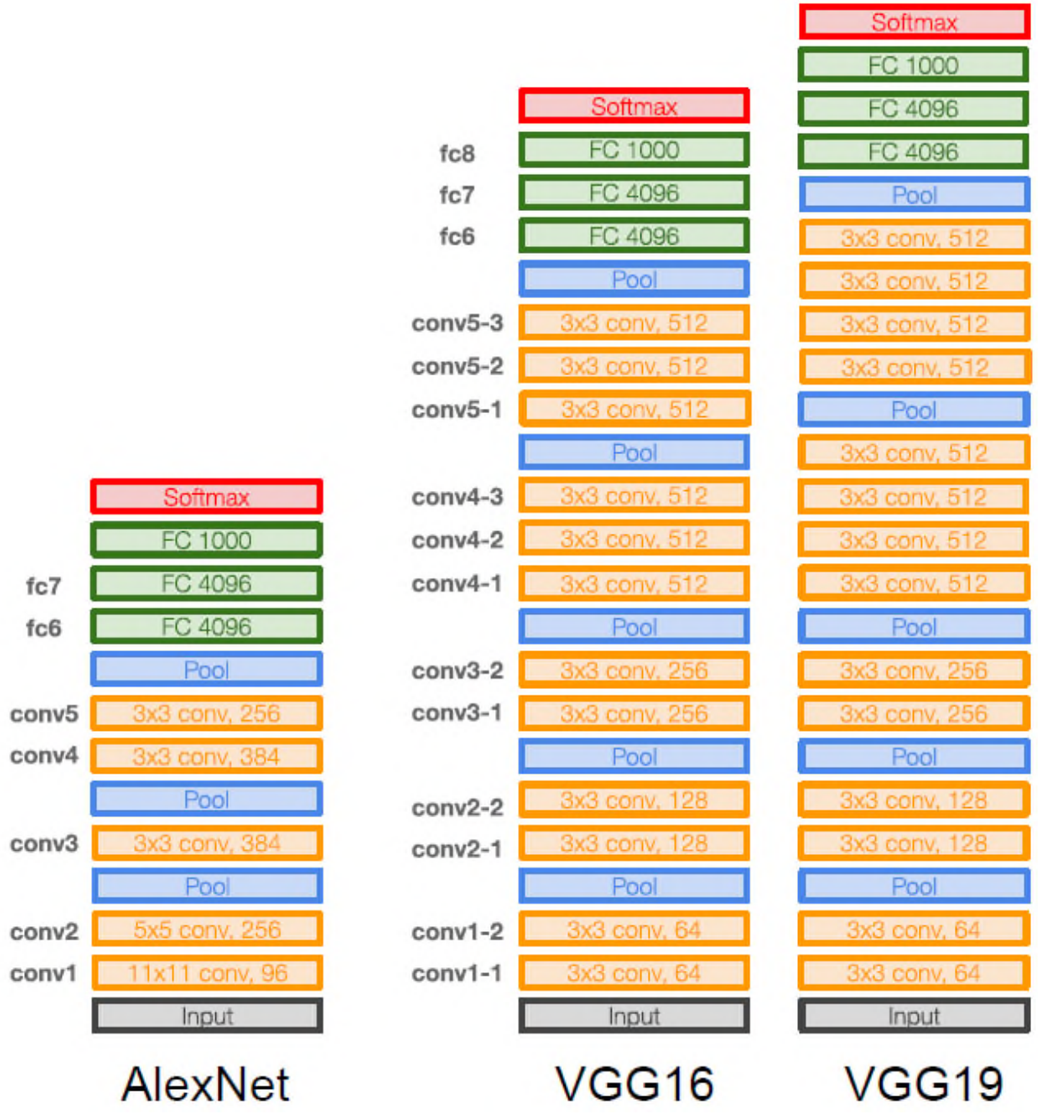
Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Case Study: VGGNet

Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



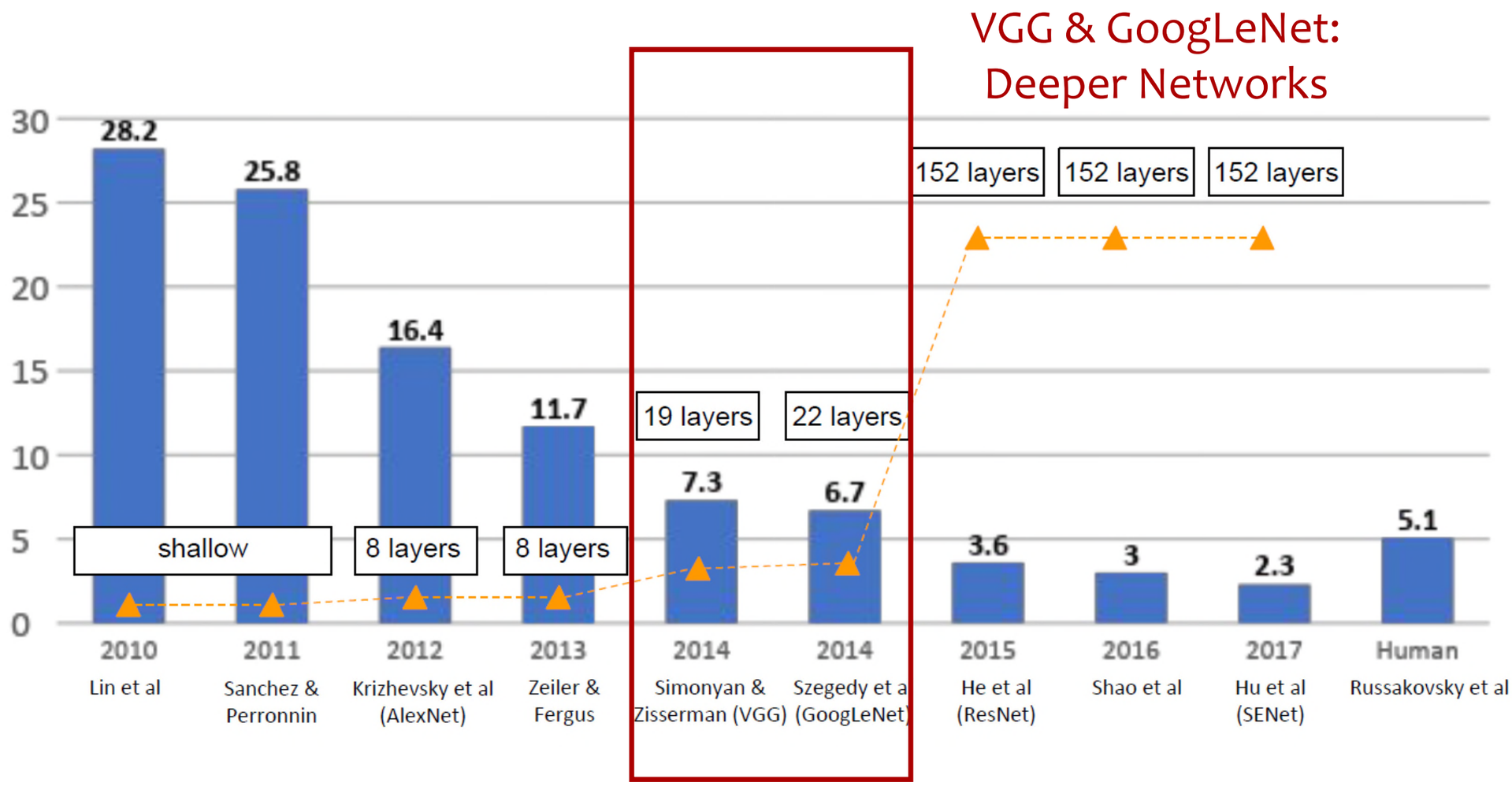
GoogleNet

ILSVRC 2014 Winner

INCEPTION MODULE

Feature Map Dimensionality Reduction

Large Scale Visual Recognition Challenge winners

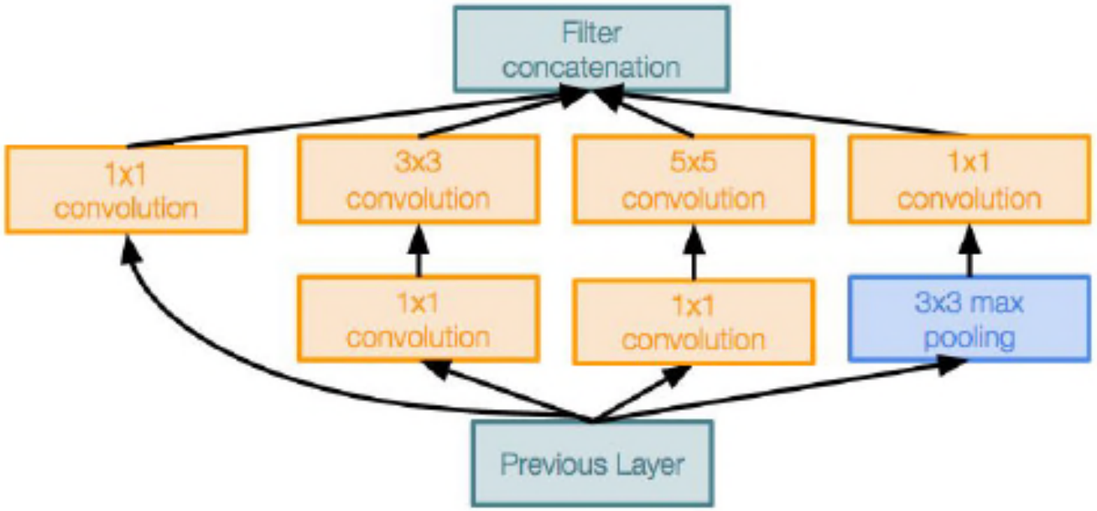


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

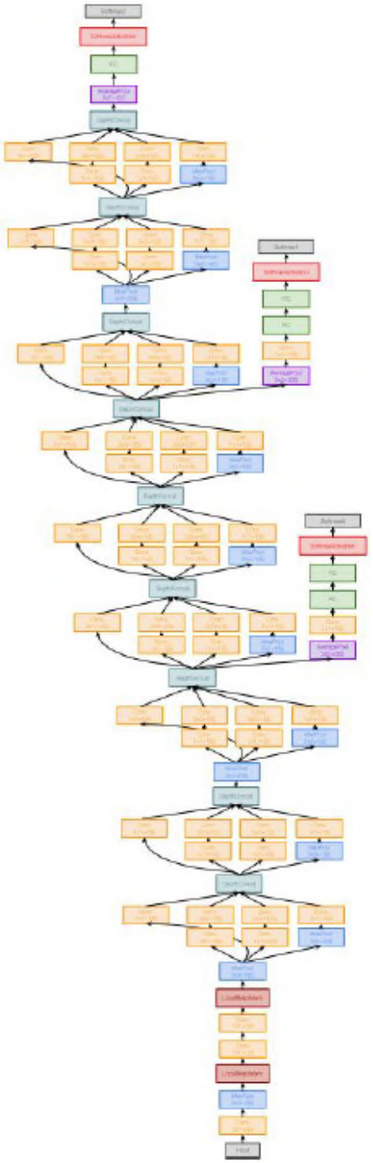
Case Study: GoogLeNet (Inception)

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC’14 classification winne
  (6.7% top 5 error)

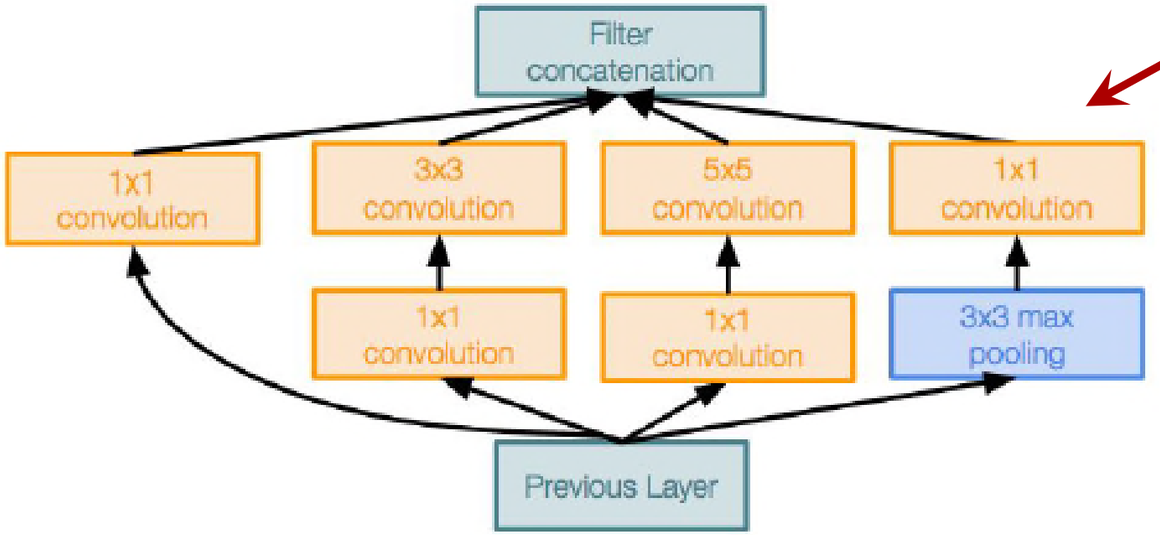


Inception module

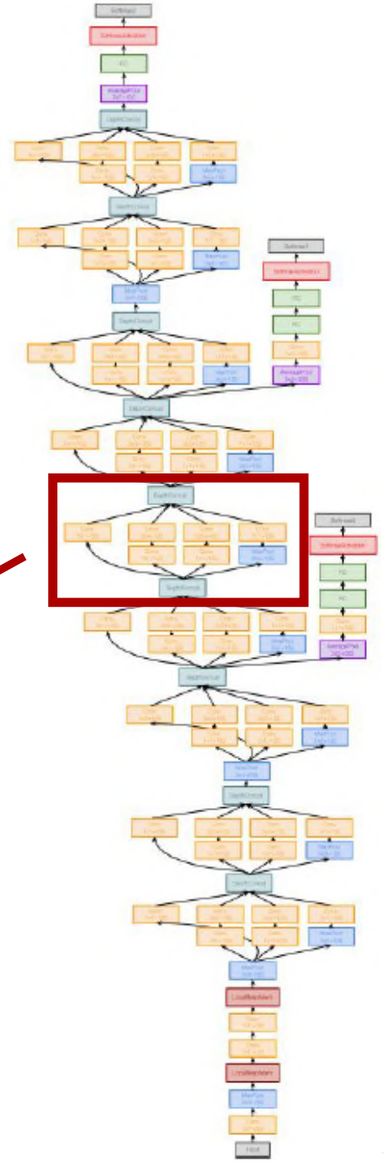


Case Study: GoogLeNet (Inception)

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other



Inception module



1×1 Convolutions to Manage Model Complexity

- Pooling can be used to **down sample the content of feature maps**, *reducing their width and height whilst maintaining their salient features*.

- A problem with deep convolutional neural networks is that the number of feature maps often increases with the depth of the network.
  - This problem can result in a dramatic increase in the number of parameters and computation required when larger filter sizes are used, such as 5×5 and 7×7

- To address this problem, a **1×1 convolutional layer** can be used that offers a channel-wise pooling, often called feature map pooling or a projection layer.
  - This simple technique can be used for dimensionality reduction, *decreasing the number of feature maps* whilst retaining their salient features.

1×1 Convolutions to Manage Model Complexity

- A filter applied to an input image or input feature map always results in a single number.
  - One filter creates one corresponding feature map. (With systematic left-to-right and top-to-bottom application)

- A filter must have the same depth or number of channels as the input,
  - Yet, regardless of the depth of the input and the filter, the resulting output is a single number and one filter creates a feature map with a single channel.

- Let's make this concrete with some examples:
  - If the input has one channel such as a grayscale image, then a 3×3 filter will be applied in 3x3x1 blocks.
  - If the input image has **three** channels for red, green, and blue, then a 3×3 filter will be applied in **3x3x3** blocks.
  - If the input is a block of feature maps from another convolutional or pooling layer and has the **depth of 64**, then the 3×3 filter will be applied in **3x3x64** blocks to create the single values to make up the single output feature map.

1×1 Convolutions to Manage Model Complexity

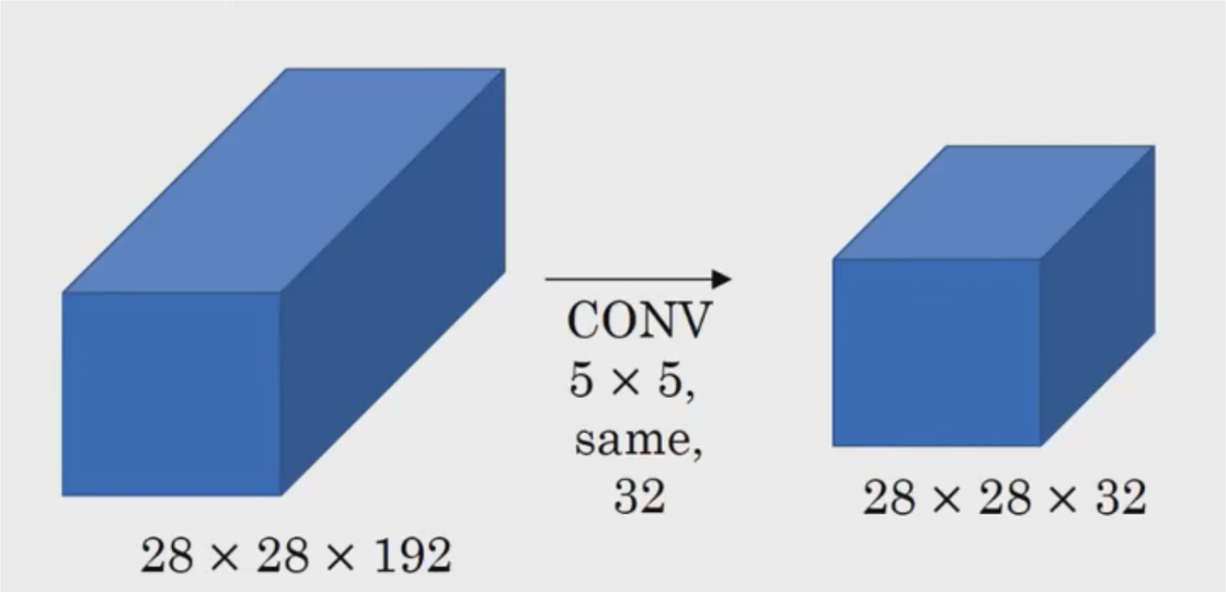
- The solution is to use a 1×1 filter to down sample the depth or number of feature maps

- The 1×1 filter is so simple that it does not involve any neighboring pixels in the input;
  - It may not be considered a convolutional operation.
  - Instead, it is a linear weighting or projection of the input. Further, a nonlinearity is used as with other convolutional layers, allowing the projection to perform non-trivial computation on the input feature maps.

- This simple 1×1 filter provides a way to usefully summarize the input feature maps.
  - The use of multiple 1×1 filters, in turn, allows the tuning of the number of summaries of the input feature maps to create, effectively allowing the depth of the feature maps to be increased or decreased as needed.

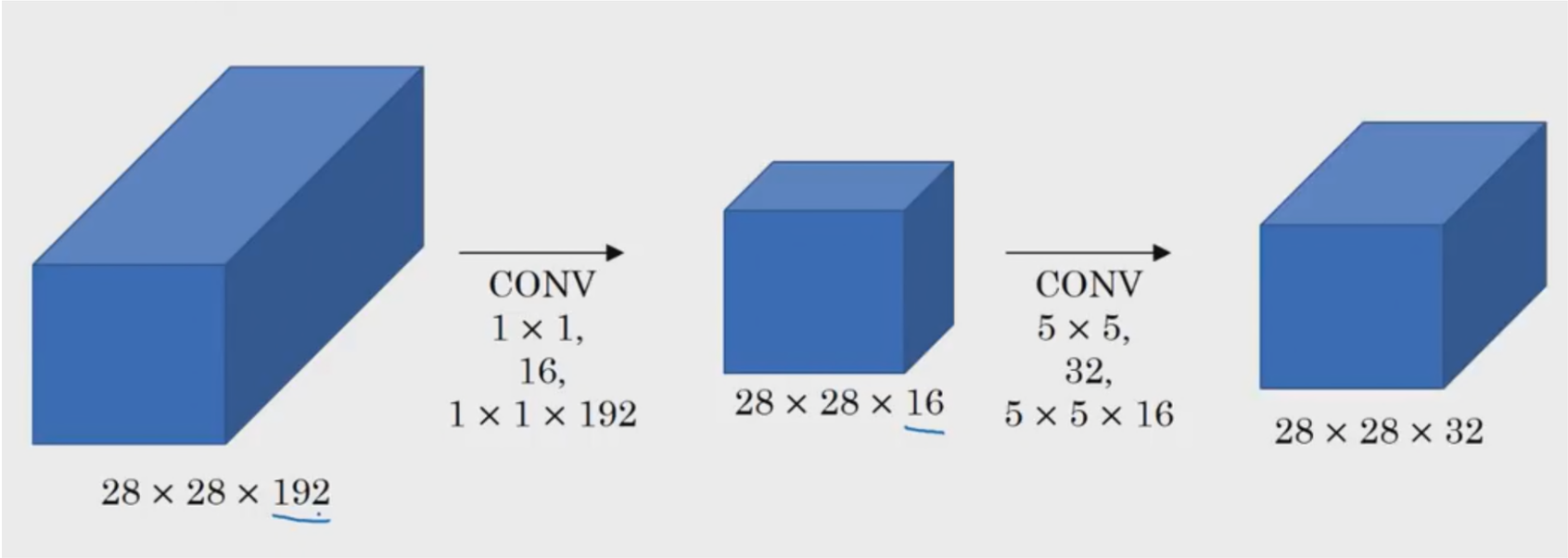
- A convolutional layer with a 1×1 filter can, therefore, be used at any point in a convolutional neural network to control the number of feature maps.

1x1 Convolution



- Filters : 32 ;
- Each filter of size 5 x 5 x 192
- Output size needed : 28 x 28 x 32
- Total computations : 120 M multiplications (ignoring biases)

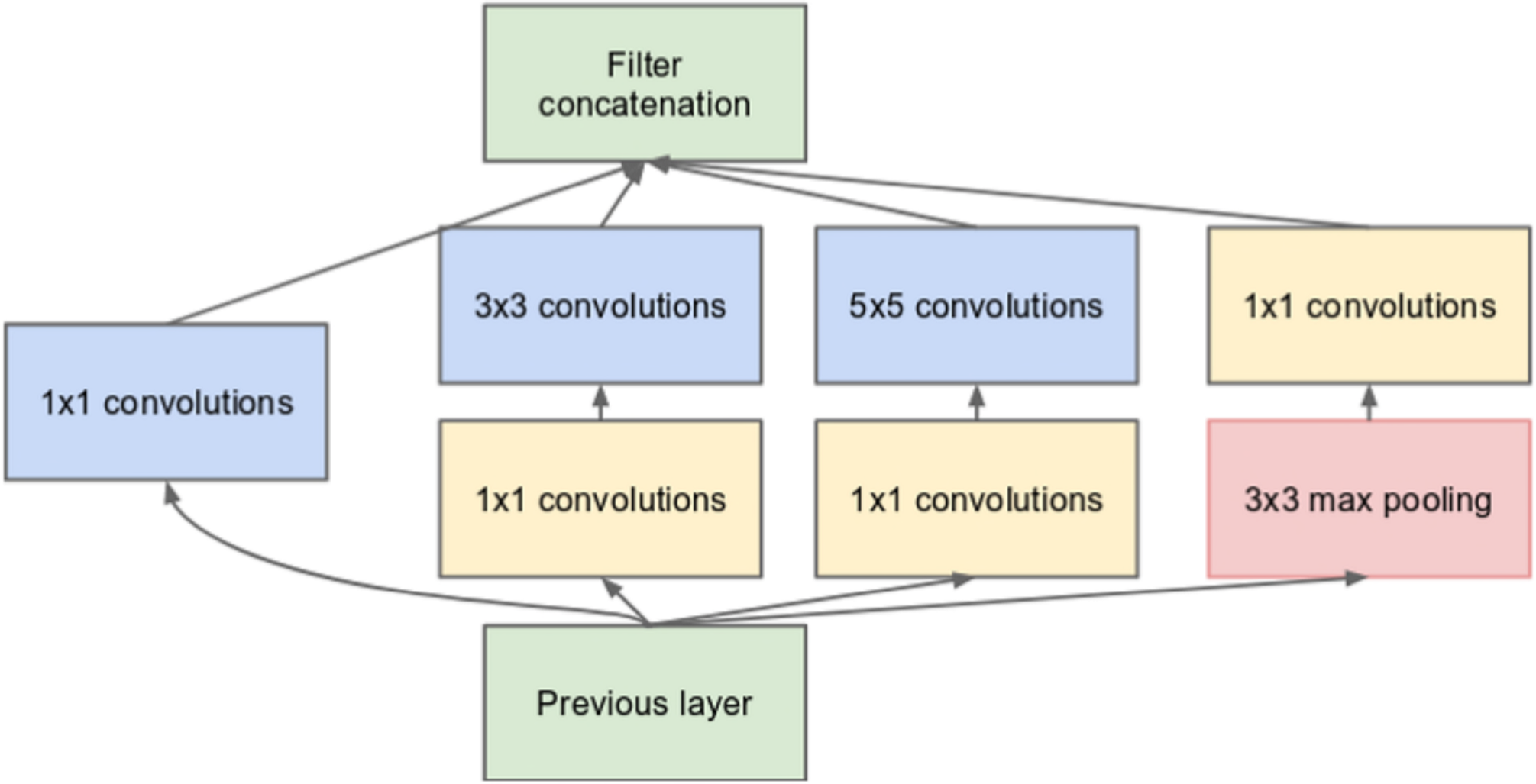
1x1 Convolution (reducing computational complexity)



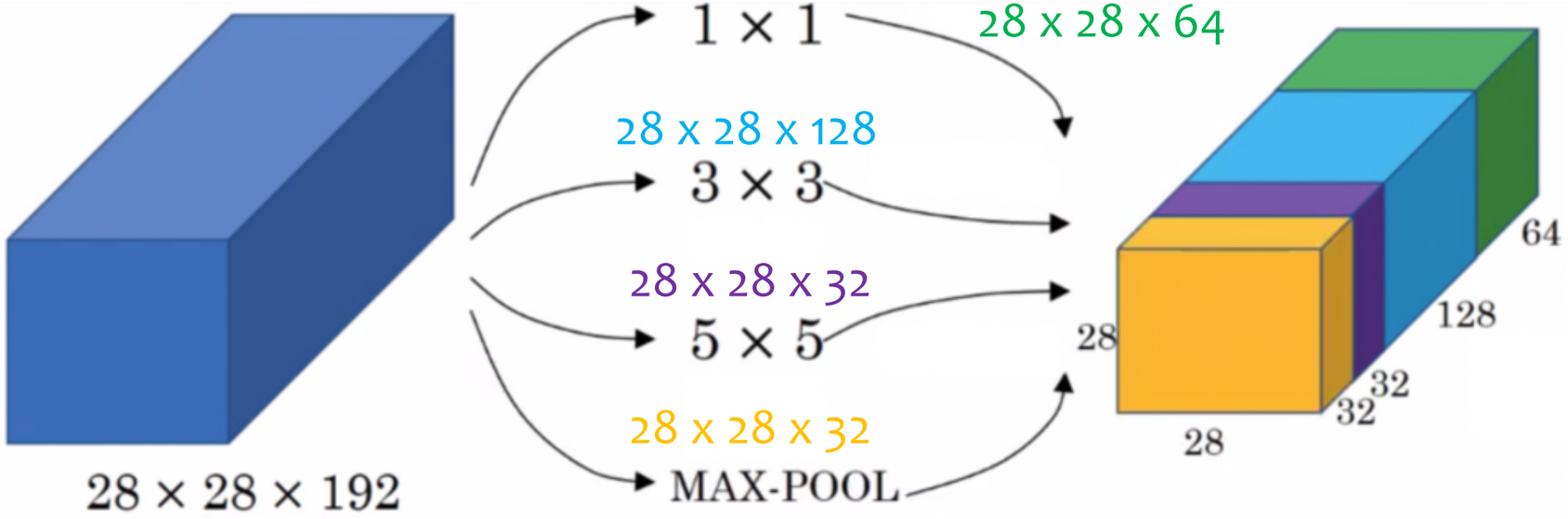
- Filters : 16
- Each filter of size 1 x 1 x 192
- Intermediate Output size : 28 x 28 x 16 (bottleneck)
- Total computations : 2.4 M multiplications (ignoring biases)

- Filters : 32
- Each filter of size 5 x 5 x 16
- Output size : 28 x 28 x 32
- Total computations : 10.0 M multiplications (ignoring biases)

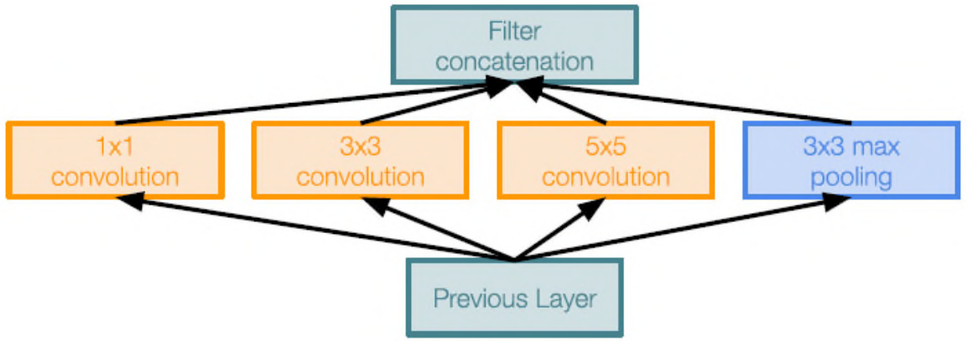
Total 12.4 M << 20 M


Filter concatenation
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer

Case Study: GoogLeNet (Inception)



Q: What is the problem with this?



Naive Inception module

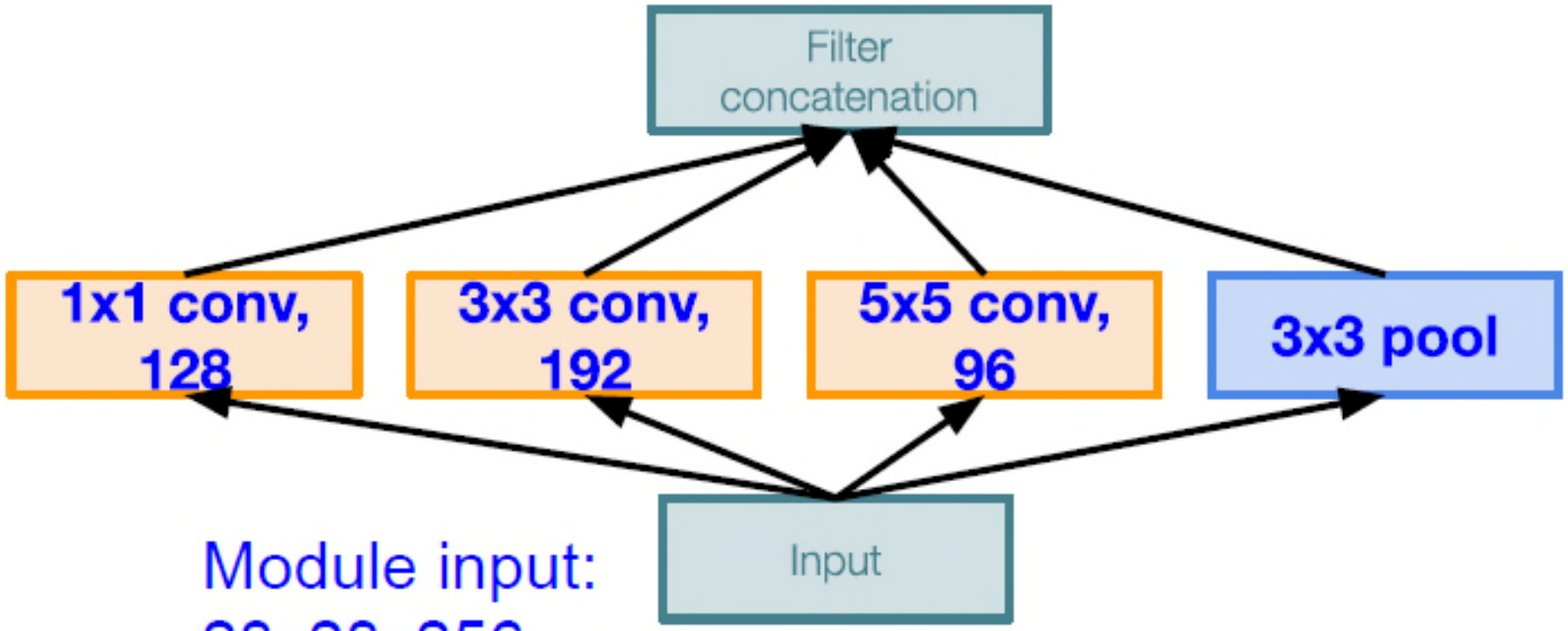
Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

Case Study: GoogLeNet (Inception)

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

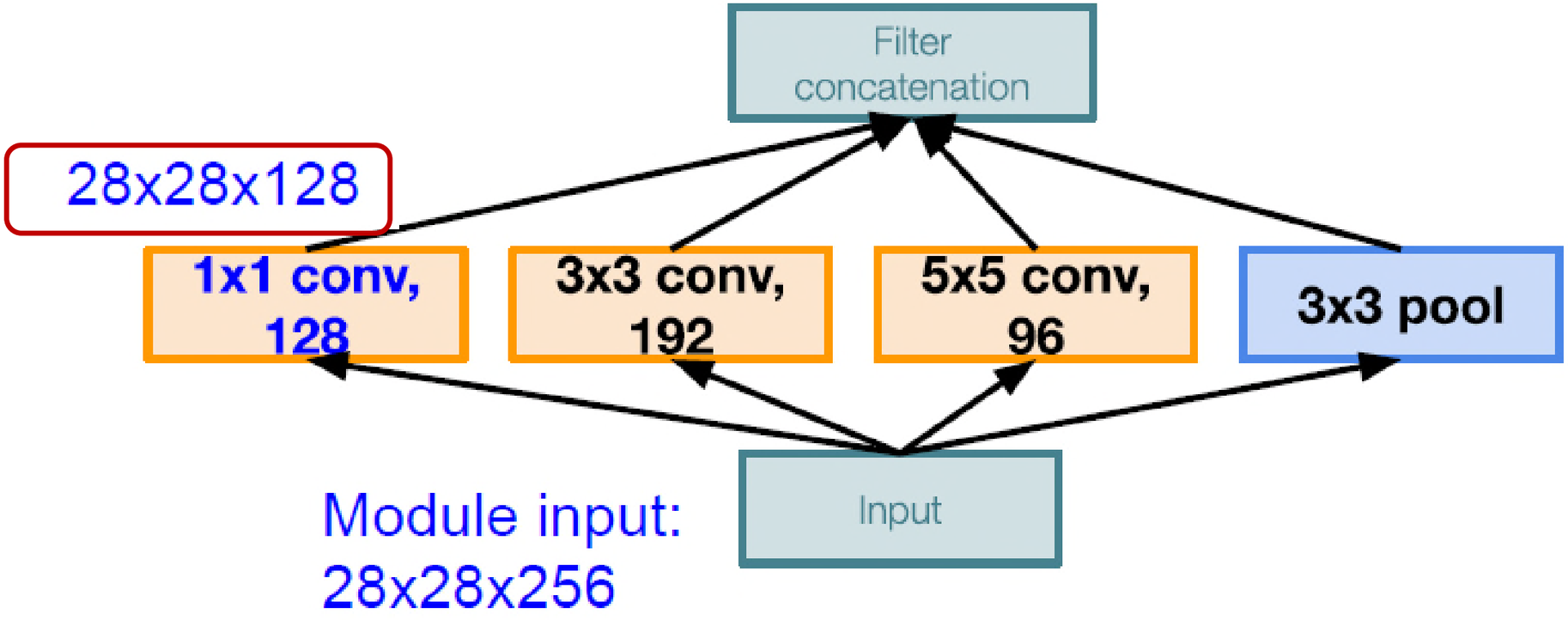


Naive Inception module

Case Study: GoogLeNet (Inception)

Example:

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

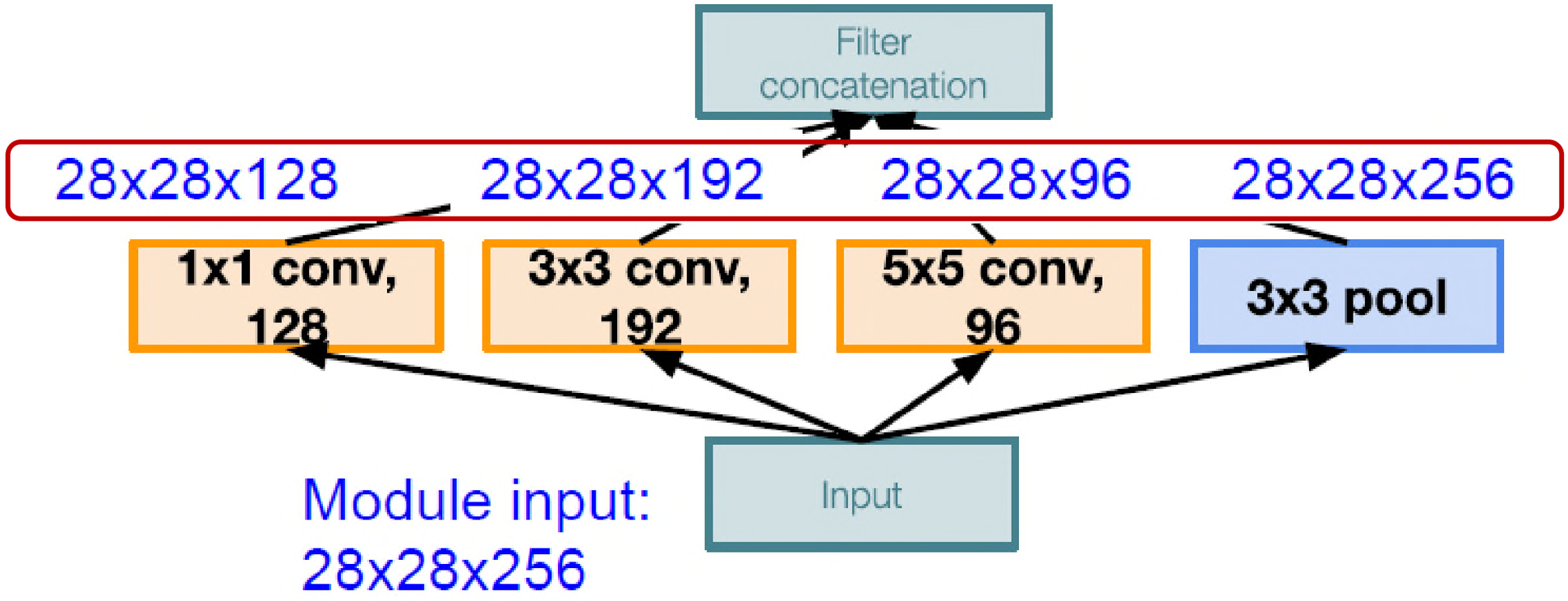


Naive Inception module

Case Study: GoogLeNet (Inception)

Example:

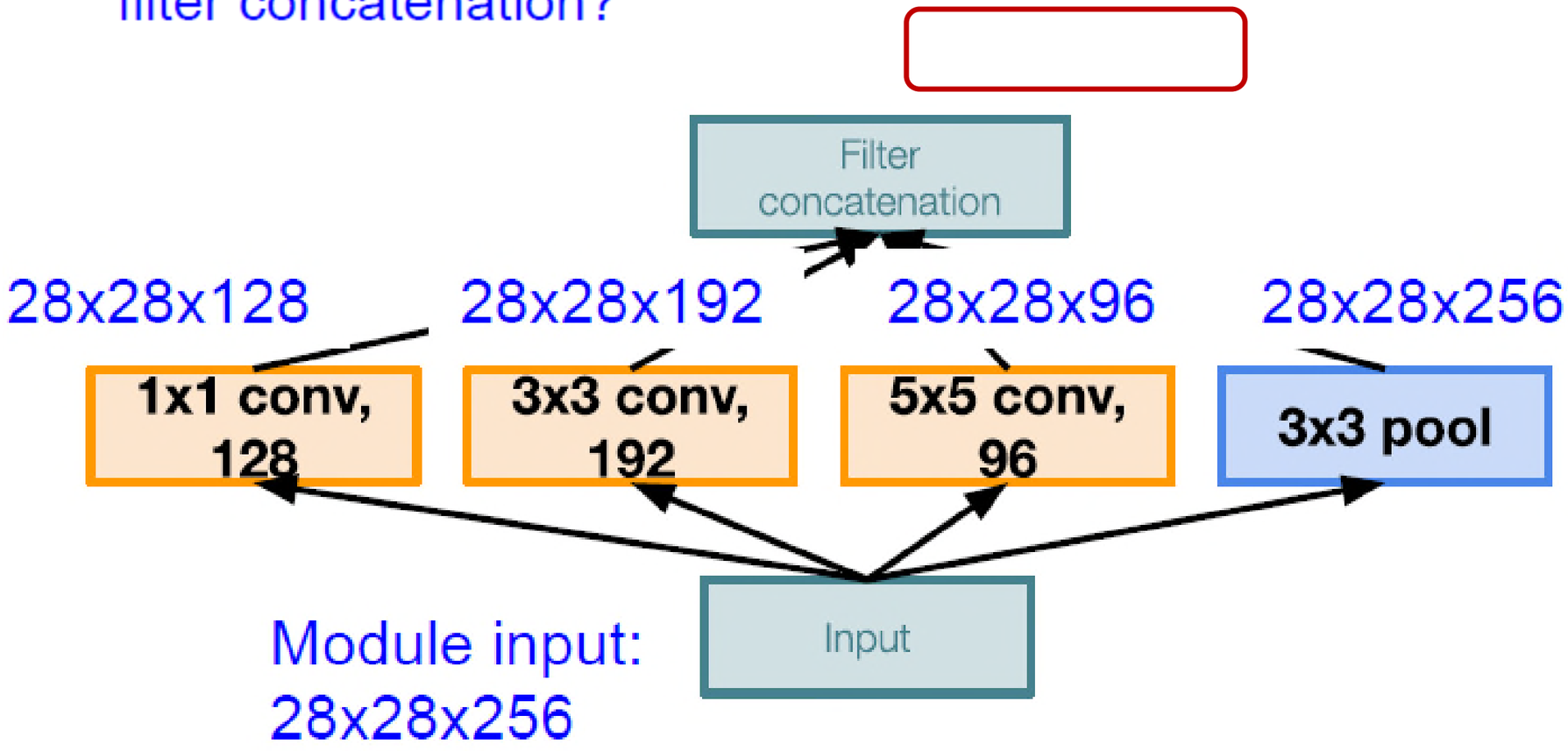
Q2: What are the output sizes of all different filter operations?



Naive Inception module

Case Study: GoogLeNet (Inception)

Example: Q3:What is output size after filter concatenation?

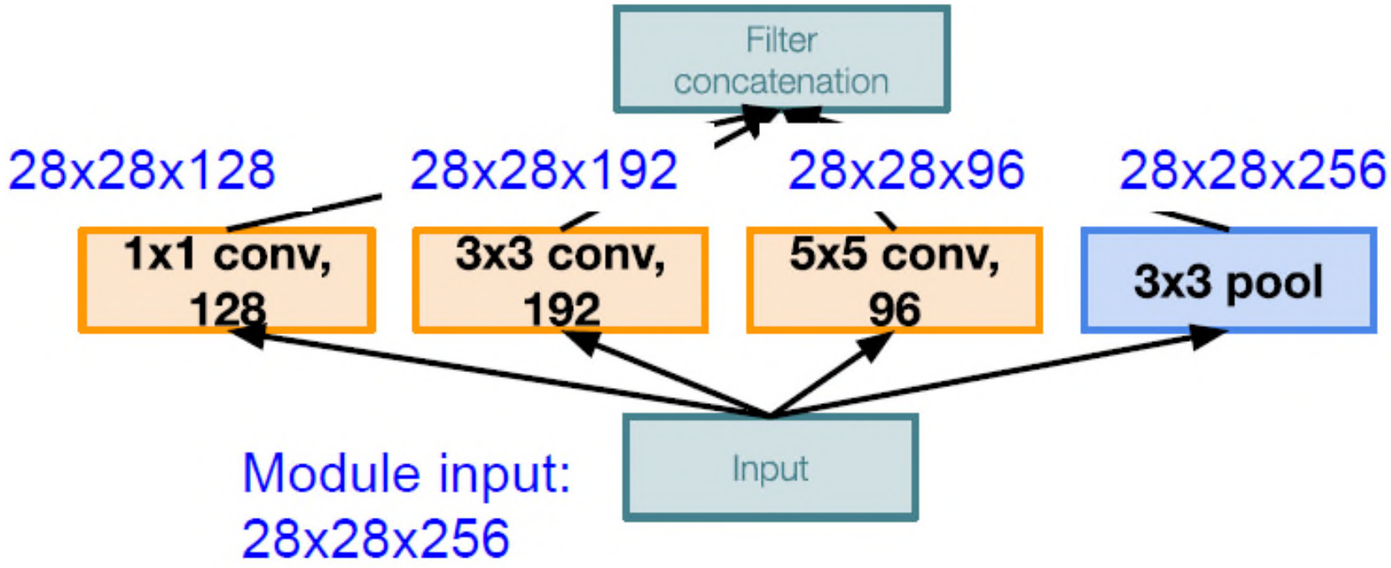


Naive Inception module

Case Study: GoogLeNet (Inception)

Example: Q3:What is output size after filter concatenation?

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



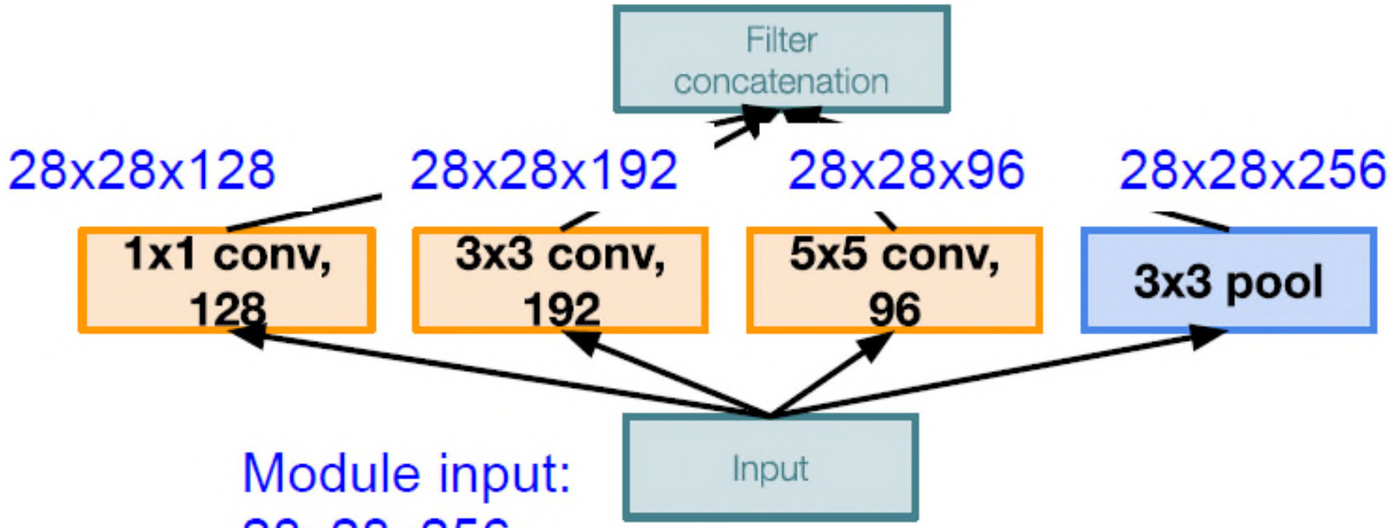
Naive Inception module

What is the Problem with this?

Case Study: GoogLeNet (Inception)

Example: Q3:What is output size after filter concatenation?

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



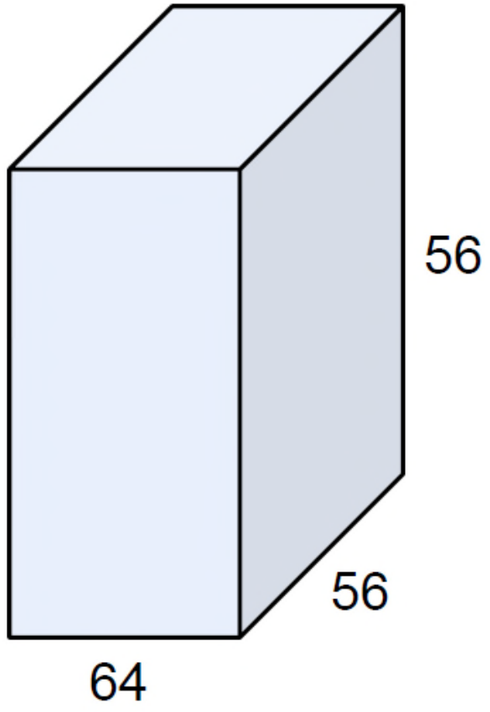
Naive Inception module

What is the Problem with this?

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

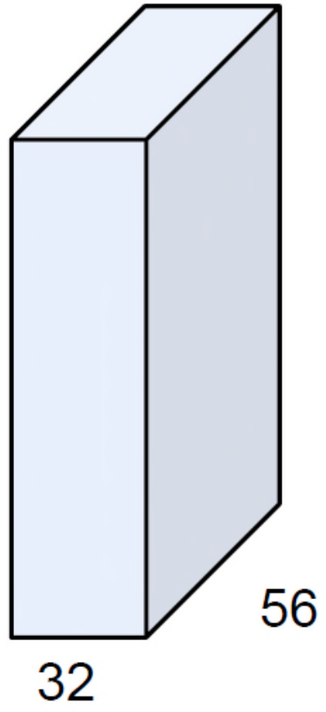
Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature depth

Reminder: 1 x 1 Convolution



1x1 CONV
with 32 filters

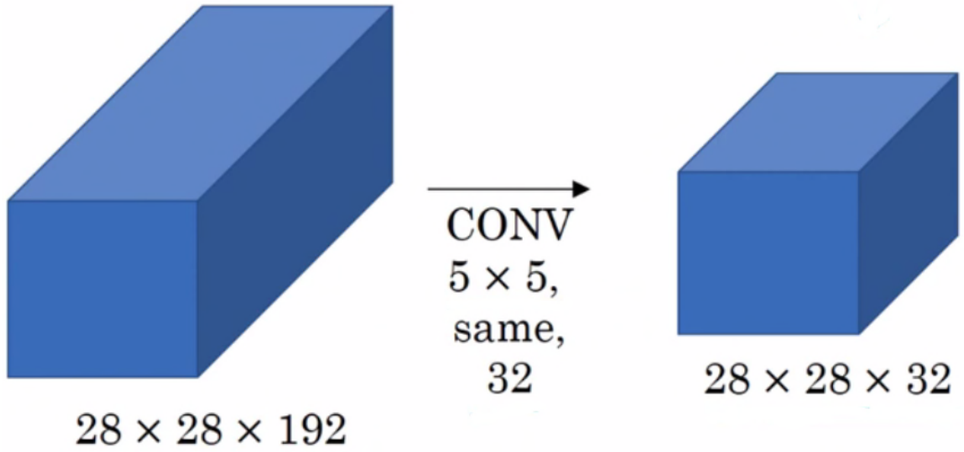
(each filter has size 1x1x64, and performs a 64-dimensional dot product)



preserves spatial dimensions, reduces depth!

Projects depth to lower dimension (combination of feature maps)

Bottleneck Layer: 1 x1 Convolution



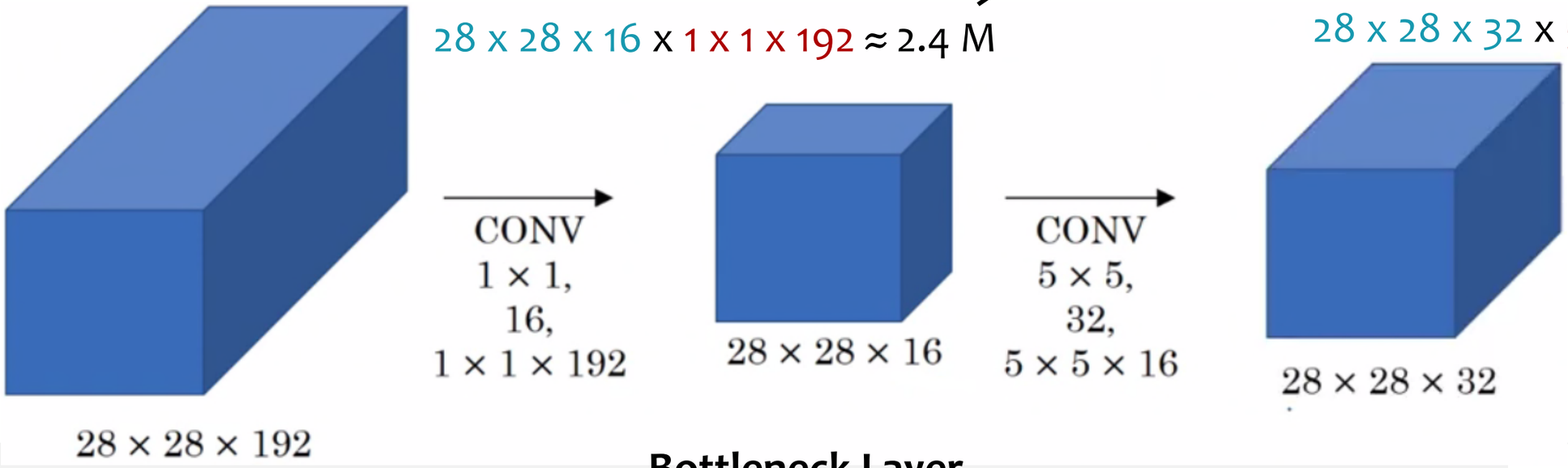
5 x 5 x 192 filters for each position resulting in 28 x 28 x 32 volume

Total operations:
28 x 28 x 32 x 5 x 5 x 192 ≈ **120 M**

2.4 M + 10 M = **12.4 M**

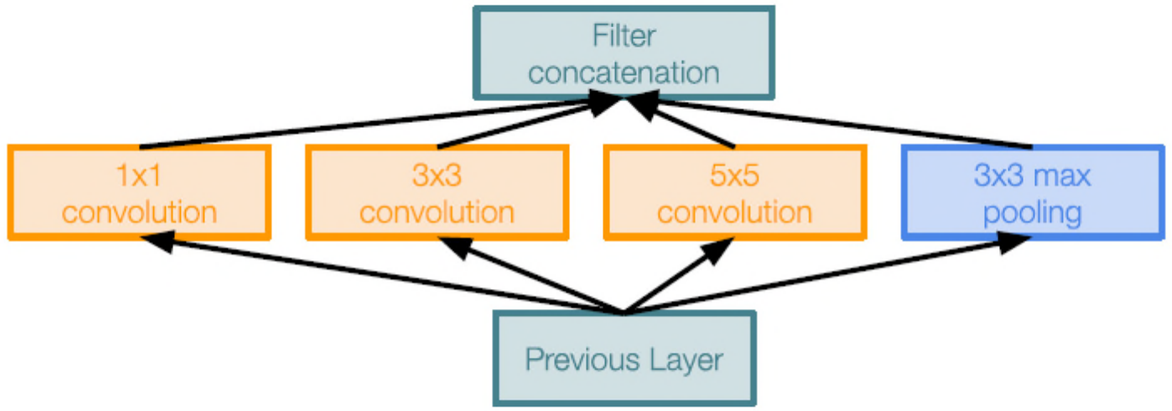
28 x 28 x 16 x 1 x 1 x 192 ≈ 2.4 M

28 x 28 x 32 x 5 x 5 x 16 ≈ 10 M



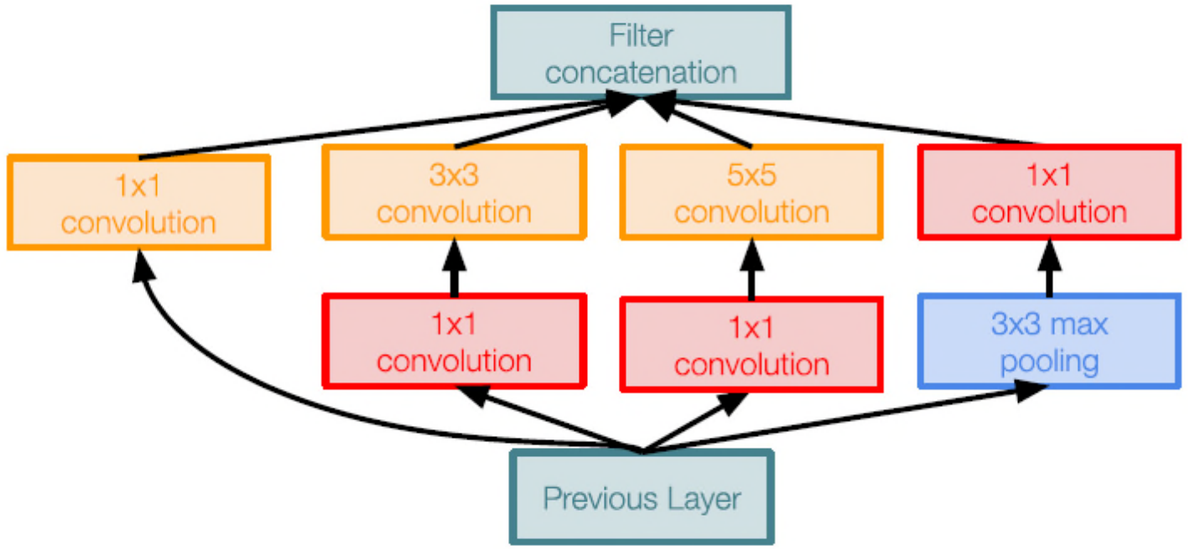
Bottleneck Layer

Case Study: GoogLeNet (Inception)



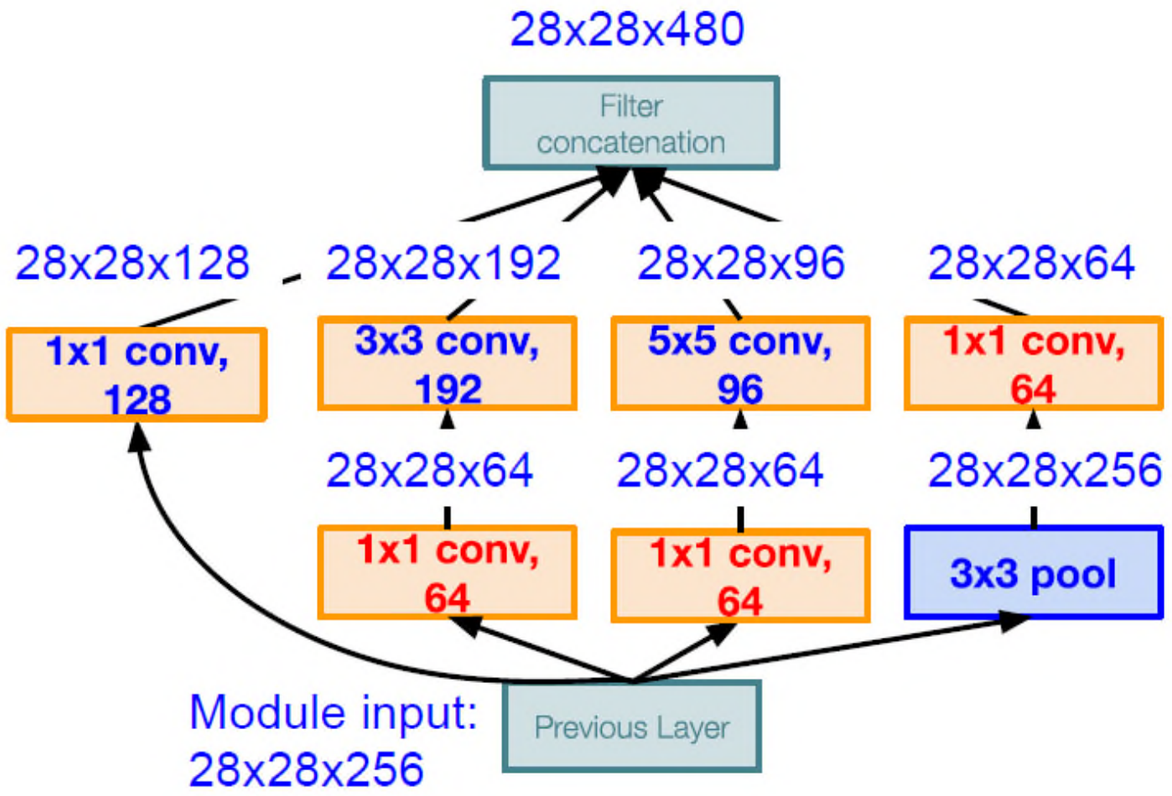
Naive Inception module

1x1 conv "bottleneck" layers



Inception module with dimension reduction

Case Study: GoogLeNet (Inception)



Inception module with dimension reduction

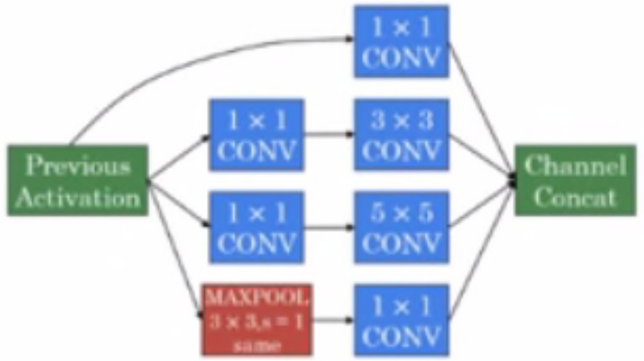
Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

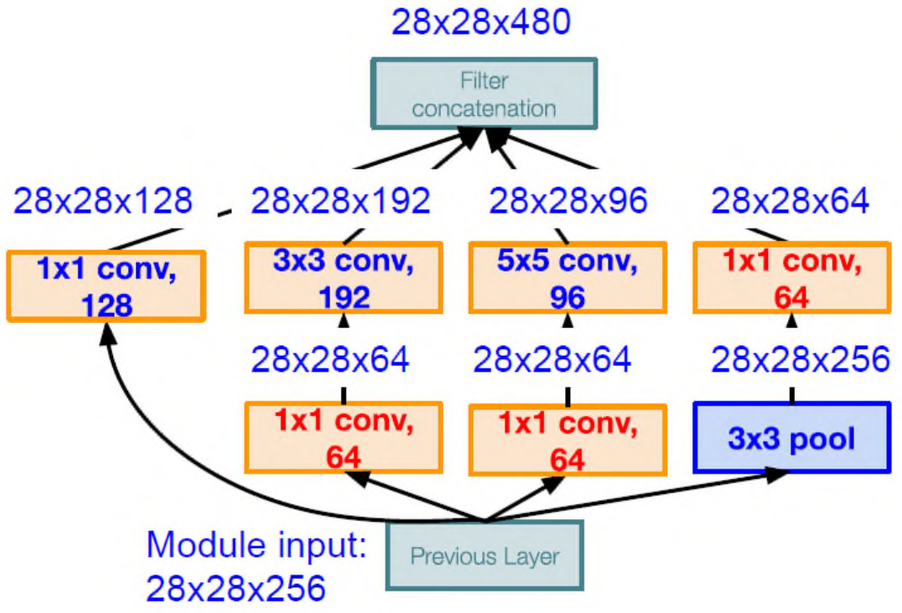
Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

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

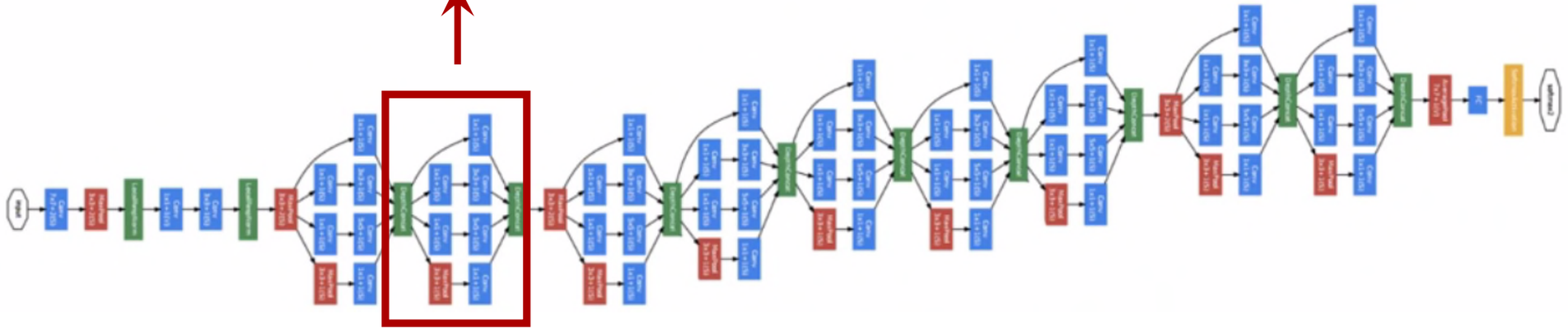
Case Study: GoogLeNet (Inception)

Stack Inception modules with dimension reduction on top of each other



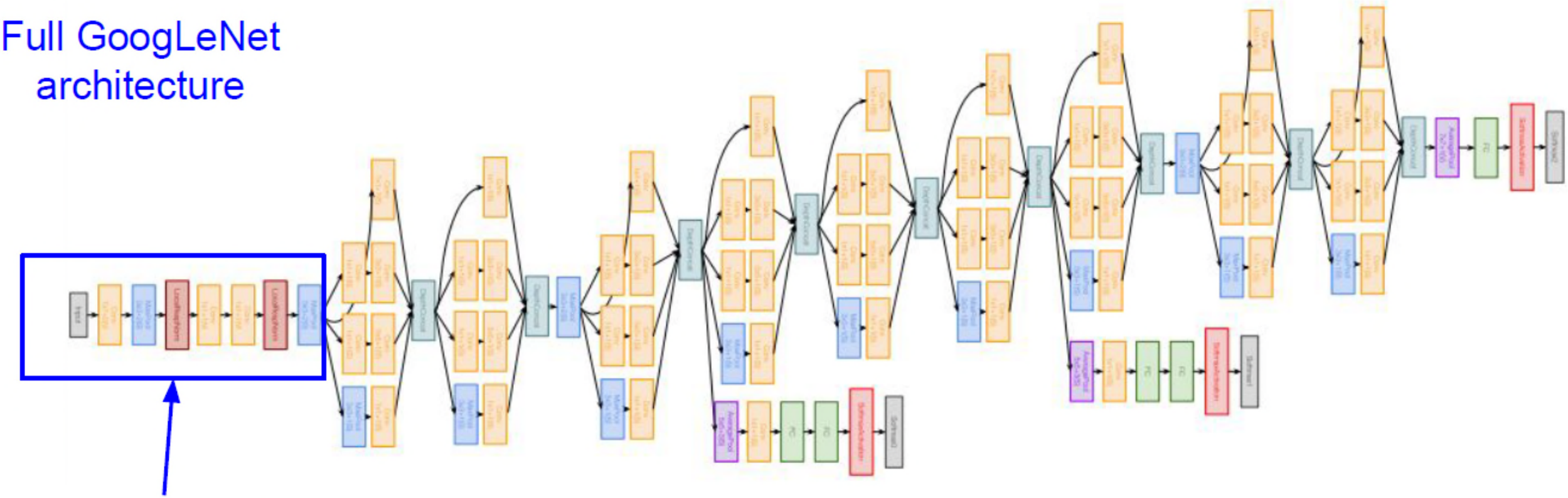


Inception module with dimension reduction



Case Study: GoogLeNet (Inception)

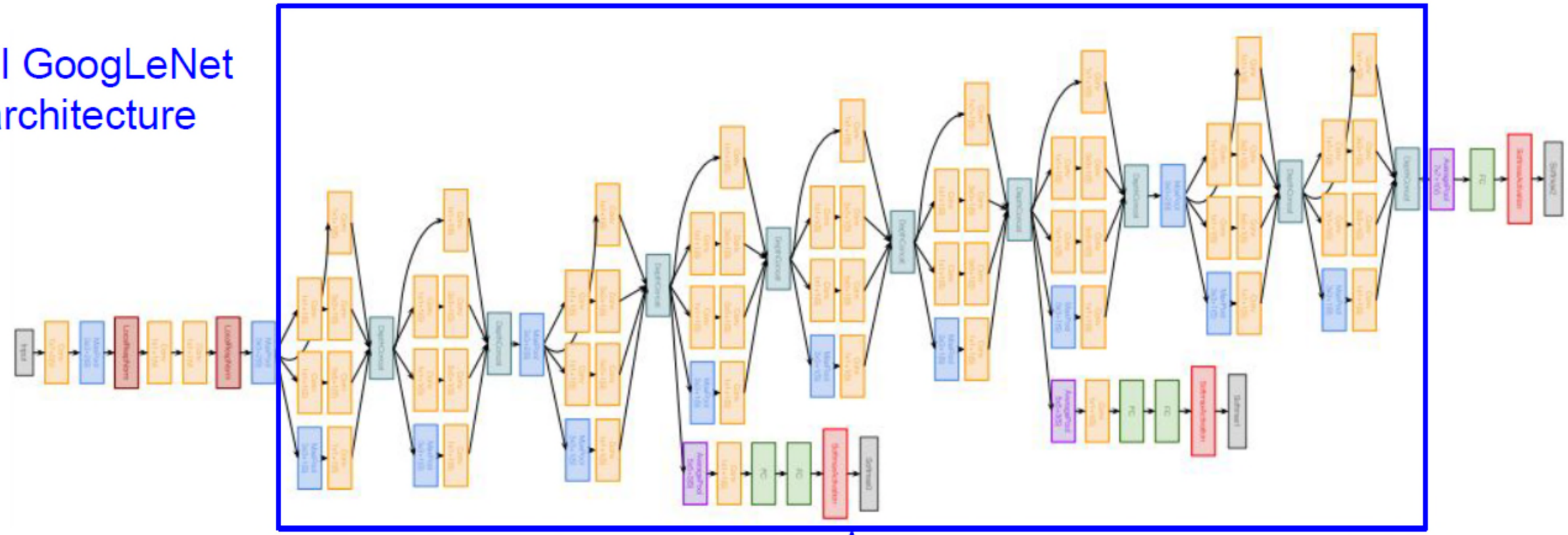
Full GoogLeNet architecture



Stem Network:
Conv-Pool-
2x Conv-Pool

Case Study: GoogLeNet (Inception)

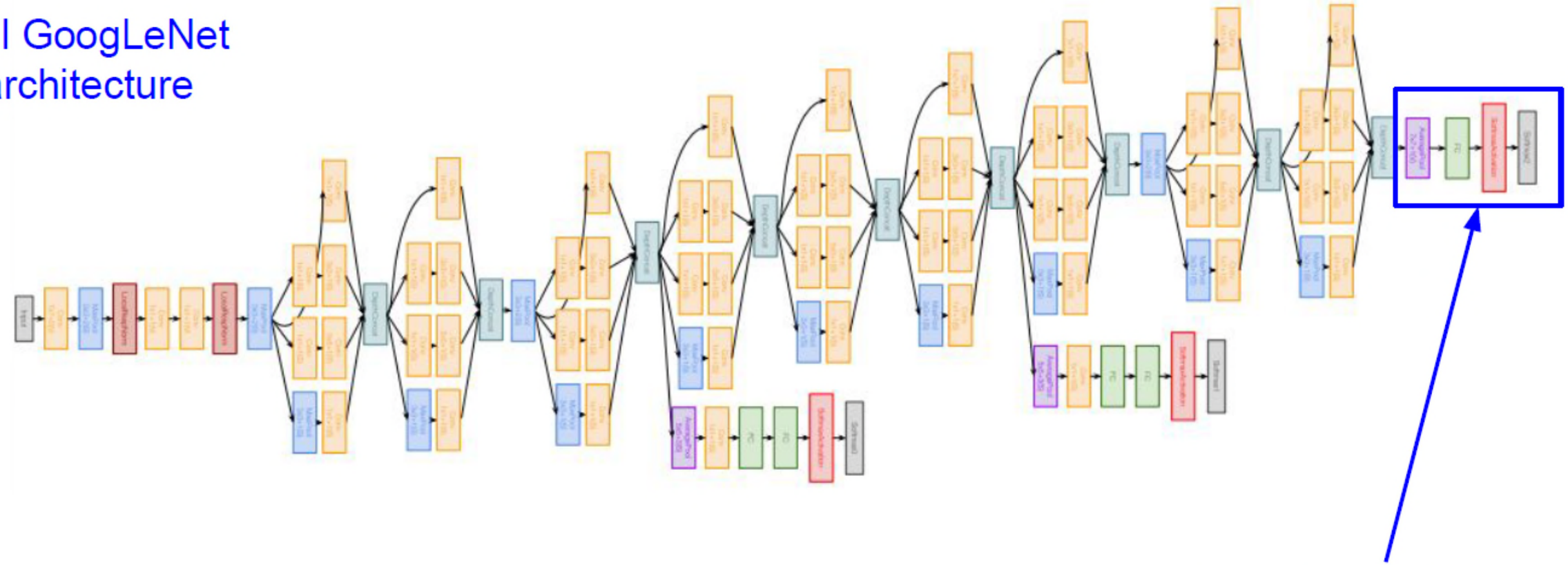
Full GoogLeNet architecture



Stacked Inception Modules

Case Study: GoogLeNet (Inception)

Full GoogLeNet architecture



Classifier output

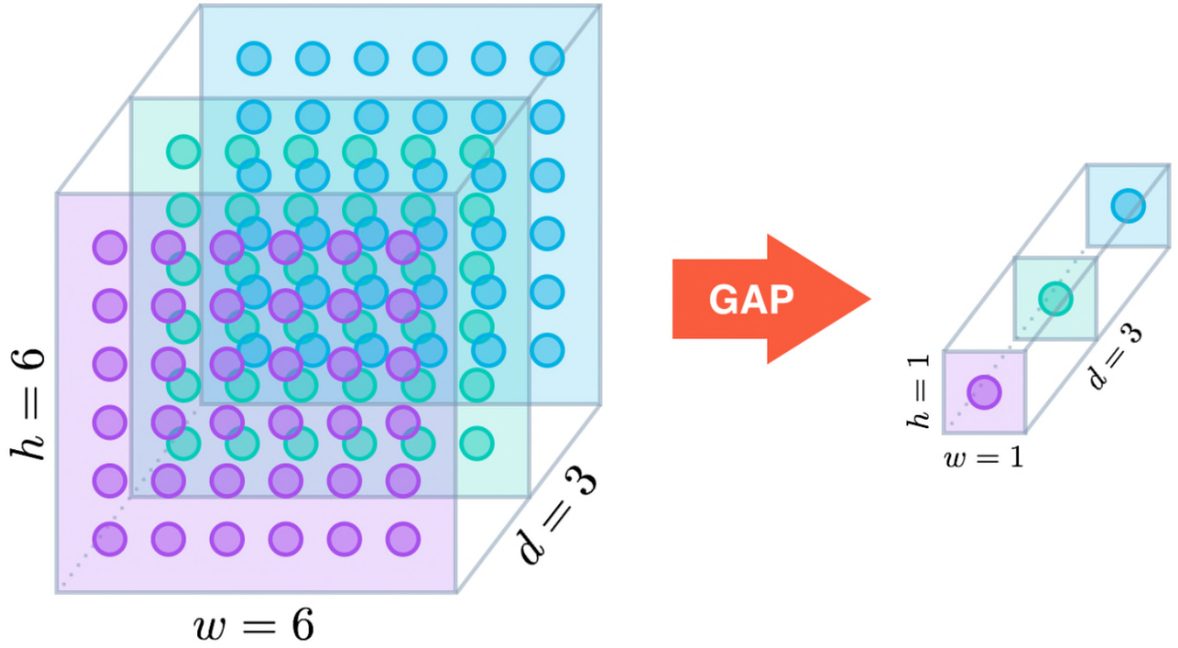
Note: after the last convolutional layer, **a global average pooling layer** is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

Global Average Pooling

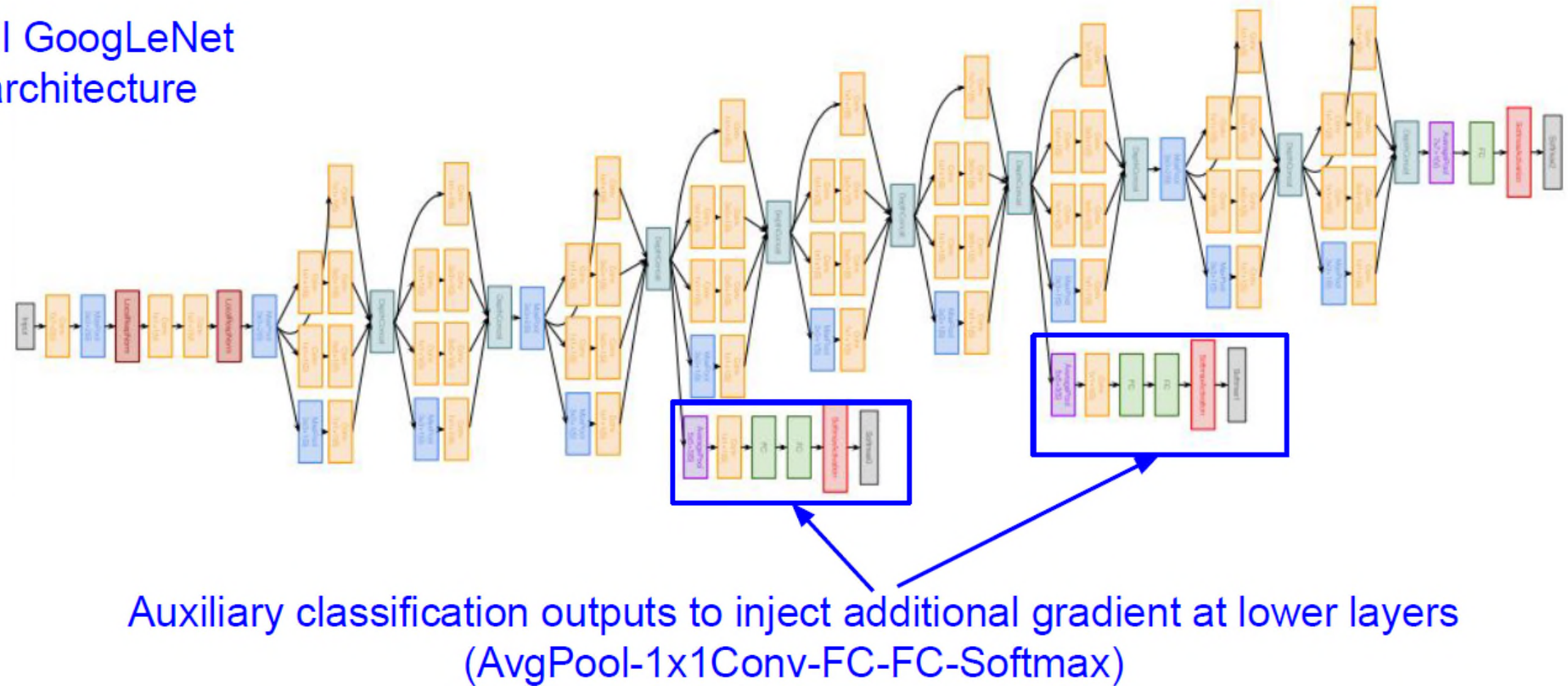
- GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor.

- However, GAP layers perform a more extreme type of dimensionality reduction, where a tensor with dimensions h×w×d is reduced in size to have dimensions 1×1×d.

- GAP layers reduce each h×w feature map to a single number by simply taking the average of all hxw values.



Case Study: GoogLeNet (Inception)

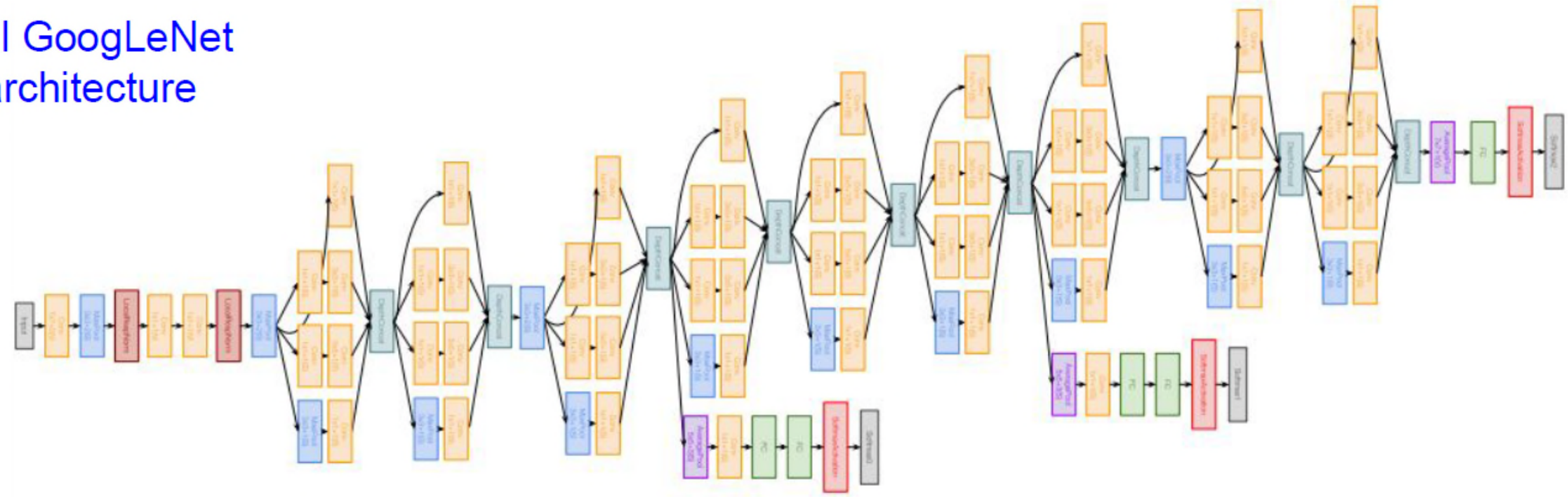


To prevent the **middle part** of the network from "**dying out**", the authors introduced **two auxiliary classifiers**. They essentially applied softmax to the outputs of two of the inception modules, and computed an **auxiliary loss** over the same labels. The **total loss function** is a **weighted sum** of the **auxiliary loss** and the **real loss**. Weight value used in the paper was 0.3 for each auxiliary loss.

```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

Case Study: GoogLeNet (Inception)

Full GoogLeNet architecture



22 total layers with weights

(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

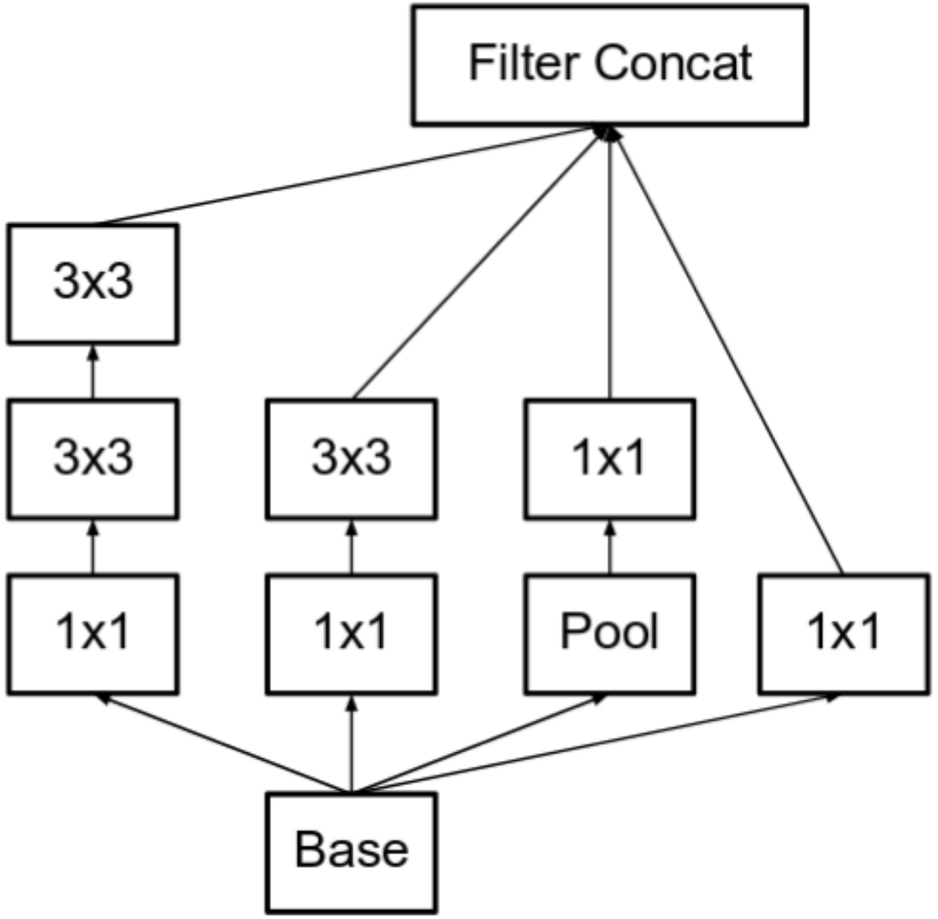
Case Study: GoogLeNet (Inception)

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								

6,797,700 parameters

Inception V2 and V3 (Module A)

- Factorize 5x5 convolution to two 3x3 convolution operations to improve computational speed.
- Although this may seem counterintuitive, a 5x5 convolution is 2.78 times more expensive than a 3x3 convolution.
- So stacking two 3x3 convolutions infact leads to a boost in performance.

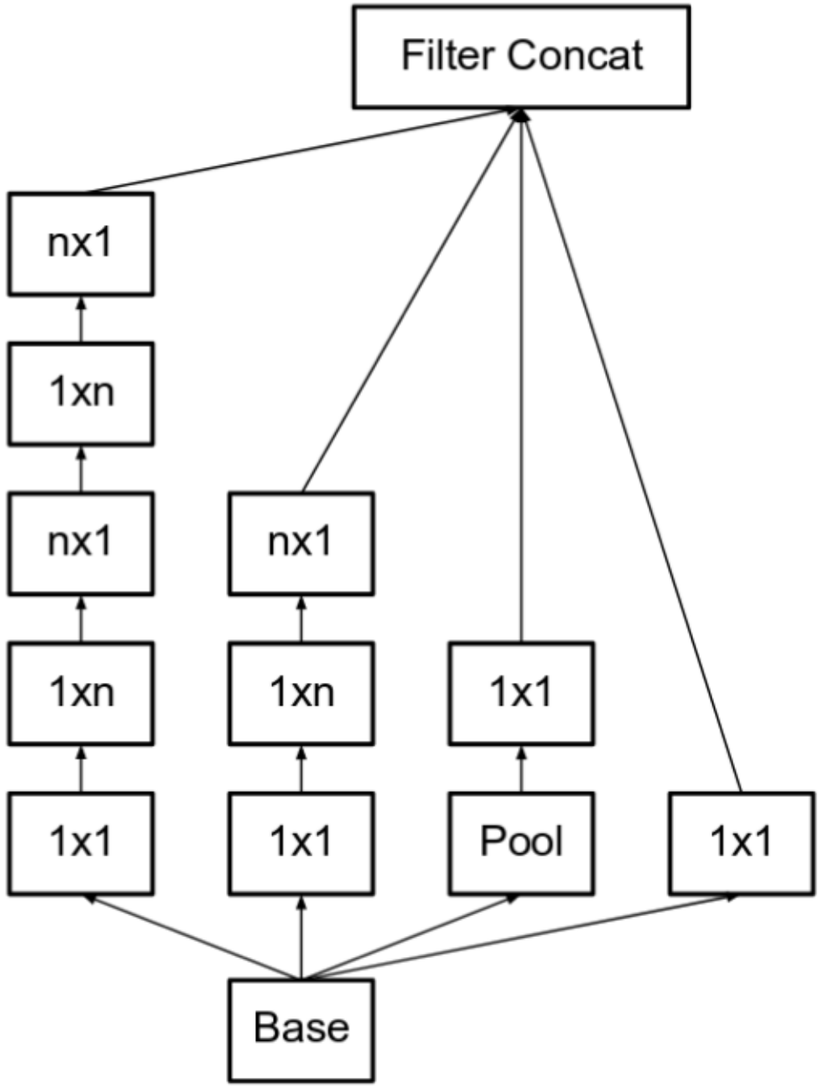


Inception V2 and V3 (Module B)

Moreover, they factorize convolutions of filter size nxn to **a combination** of 1xn and nx1 convolutions.

> For example, a 3x3 convolution is equivalent to first performing a 1x3 convolution, and then performing a 3x1 convolution on its output.

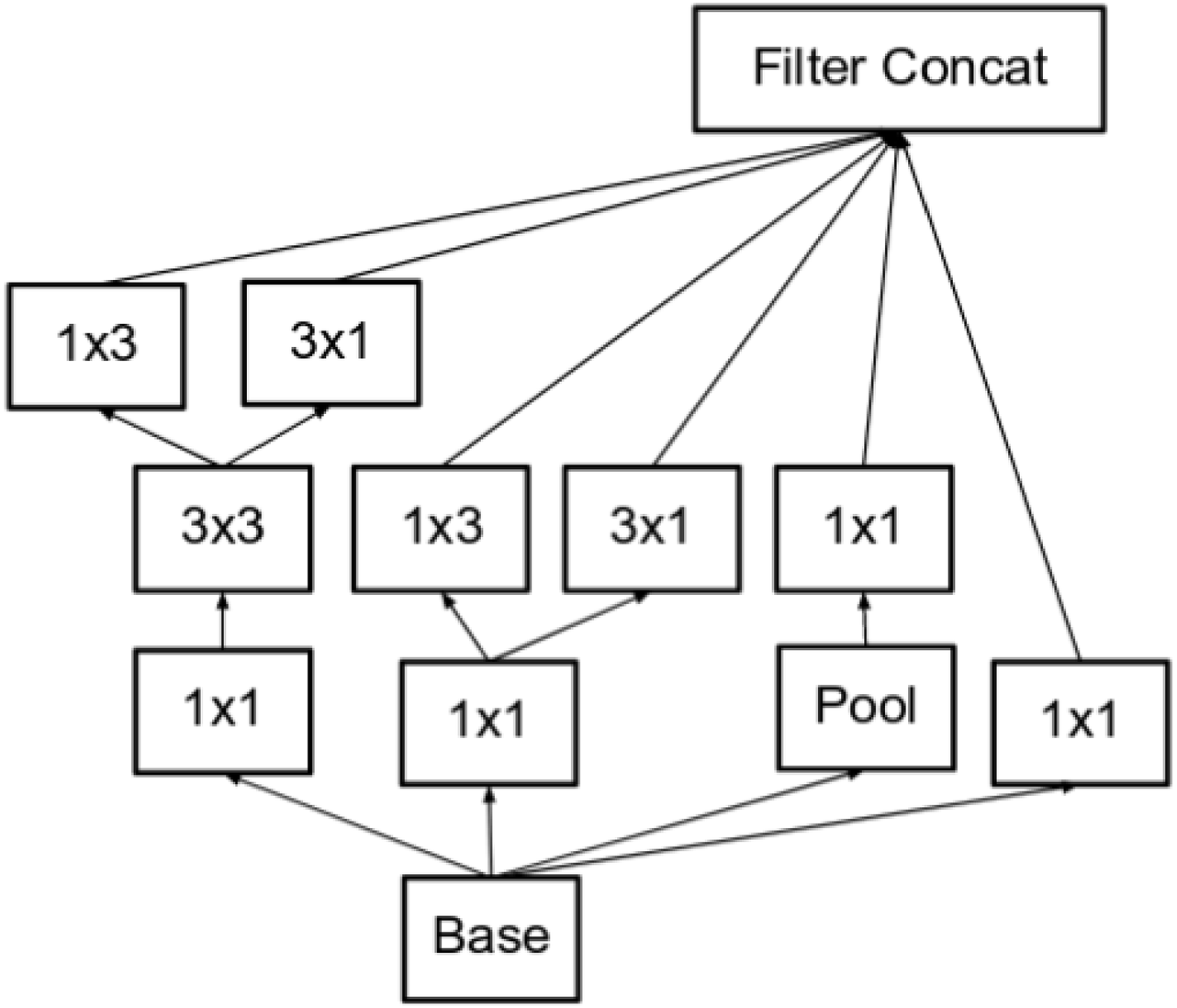
They found this method to be 33% more cheaper than the single 3x3 convolution.



Inception V2 and V3 (Module C)

The **filter banks** in the module were **expanded** (made wider instead of deeper) to remove the representational bottleneck.

If the module was made deeper instead, there would be excessive reduction in dimensions, and hence loss of information.



Inception V2

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	Module A	35×35×288
5×Inception	Module B	17×17×768
2×Inception	Module C	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Reading Material

- Receptive Filed
  - A guide to receptive field arithmetic for Convolutional Neural Networks

- ABOUT GoogleNet (INCEPTION)
  - **A Simple Guide to the Versions of the Inception Network**
    - https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202
  - **Inception (Video)**
    - https://www.youtube.com/watch?v=VxhSouuSZDY
  - **Inception Network Building Block**
    - https://www.youtube.com/watch?v=C86ZXvgpejM&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=17
  - **Combining Building Blocks to build the Inception**
    - https://www.youtube.com/watch?v=KfV8CJh7hE0&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=18

Acknowledgements

Various contents in this presentation have been taken from different books, lecture notes and the web. These solely belong to their owners, and are here used only for clarifying various educational concepts.
Any copyright infringement is *not* intended.