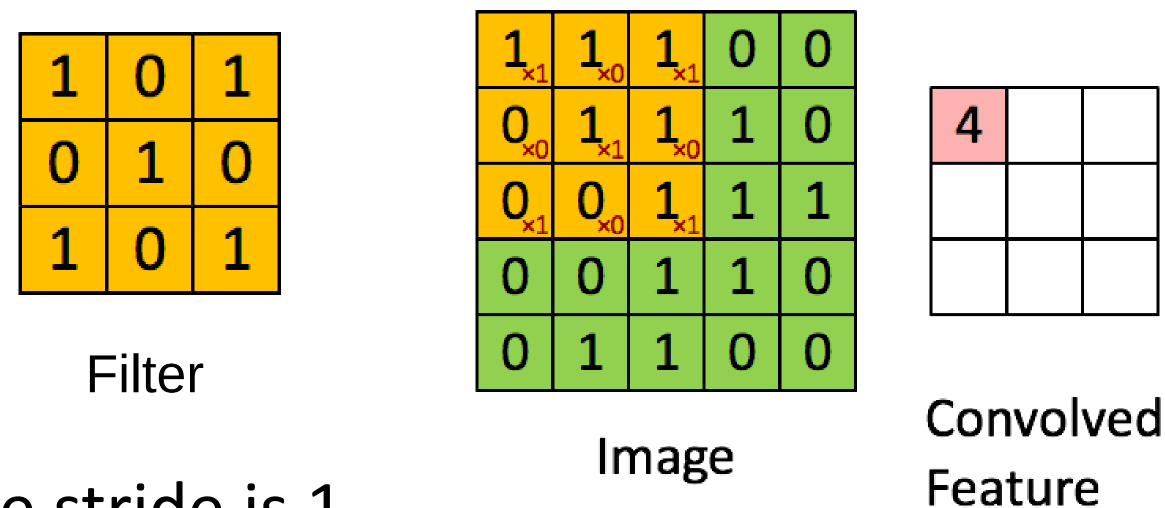
CNN Architectures

Outline

- Convolution and CONV Layers
- ImageNet Large Visual Recognition Challenge (ILSVRC)
- CNN Architectures
  - Successful beginner: LeNet
  - ILSVRC winners: AlexNet, VGGNet, GoogLeNet, ResNet
  - Advanced models: Inception, DenseNet, ResNeXt

Convolution

- For standard 2D convolution:



  - The stride is 1.
  - The height and width are changed as:

$$W_{out} = \frac{W_{in} - W_{filter}}{Stride} + 1 = (5 - 3)/1 + 1 = 3.$$

Convolution

We need **Zero-Padding** to keep image size:

0	0	0	0	0	0	0
0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0	0
0	$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0	0
0	$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

2	2	3	1	1
1	4	3	4	1
1	2	4	3	3
1	2	3	4	1
0	2	2	1	1

The width/height will become:

$$W_{out} = \frac{W_{in} - W_{filter} + 2 \times Padding}{Stride} + 1$$

CONV Layers

In CNN, the data are stored as **4D tensor**:

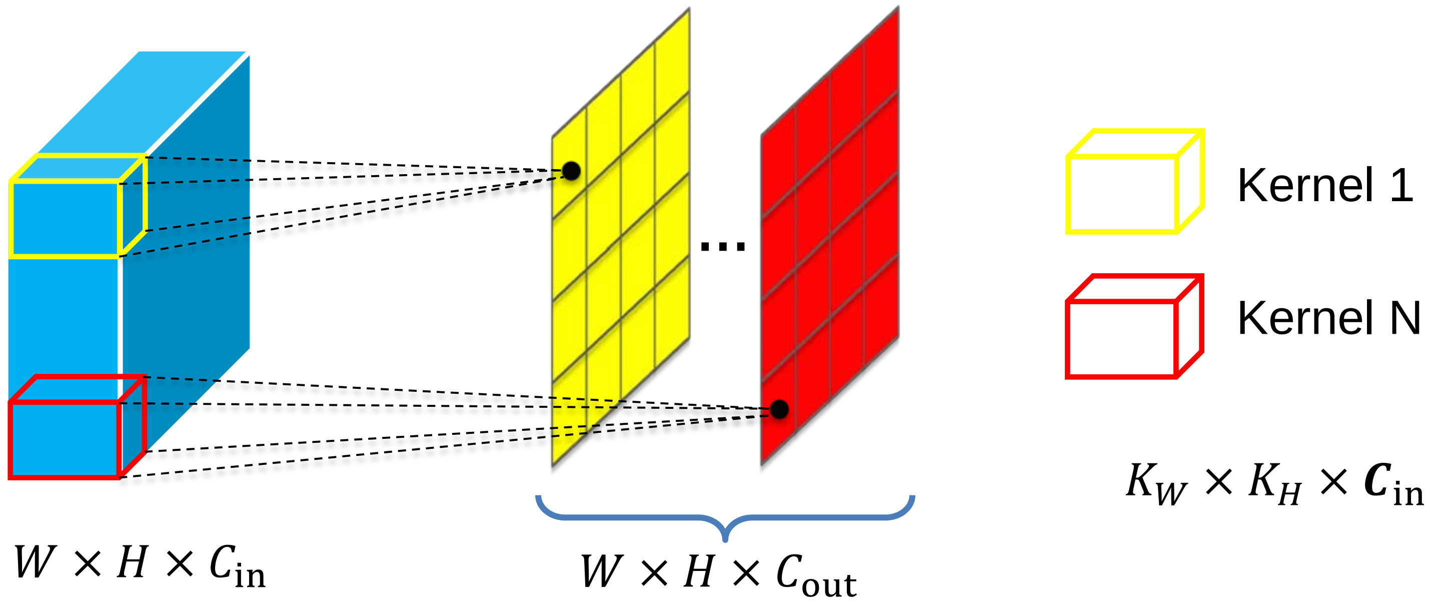
(B, C, W, H)

- B: batch size, the number of input data for each iteration.
  - After several iterations, the training data will be fully used once, which is called one **epoch**.
- C: channel, e.g. RGB image has three channels.
- W/H: width/height.
  - They are always the same in the same layer.

CONV Layers

In CONV layers:

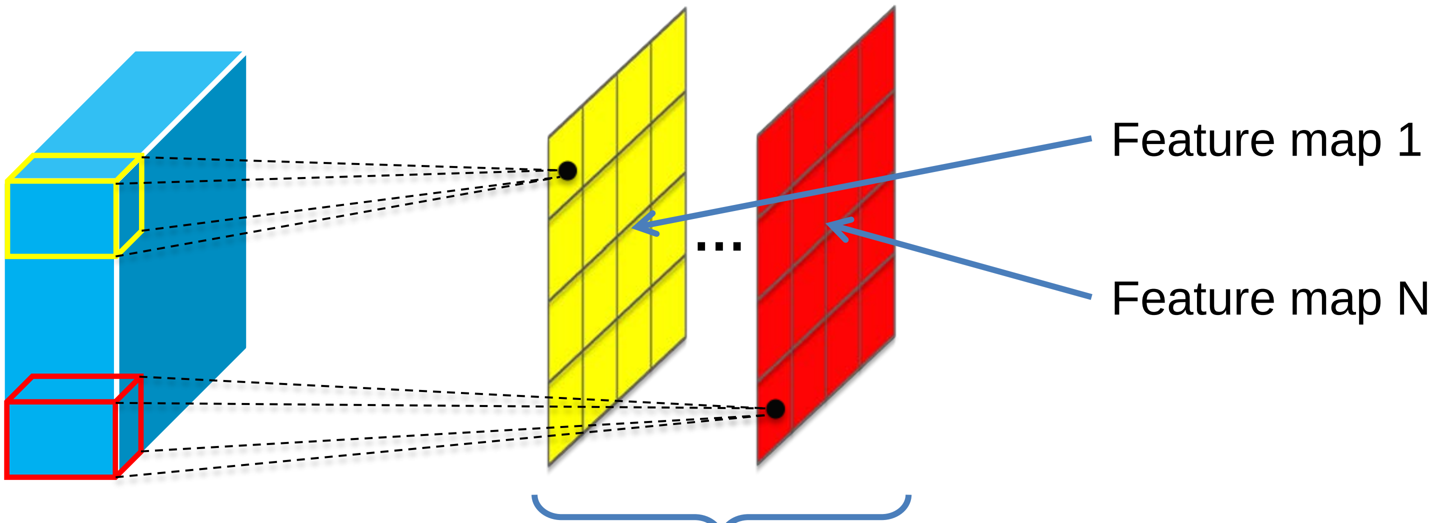
- Filters are called **Kernel**s and become 3D. The parameters of kernels are to be learned.



CONV Layers

In CONV layers:

- Feature maps are the outputs of each layer. The number of feature maps is the channel.



$W \times H \times C_{\text{in}}$

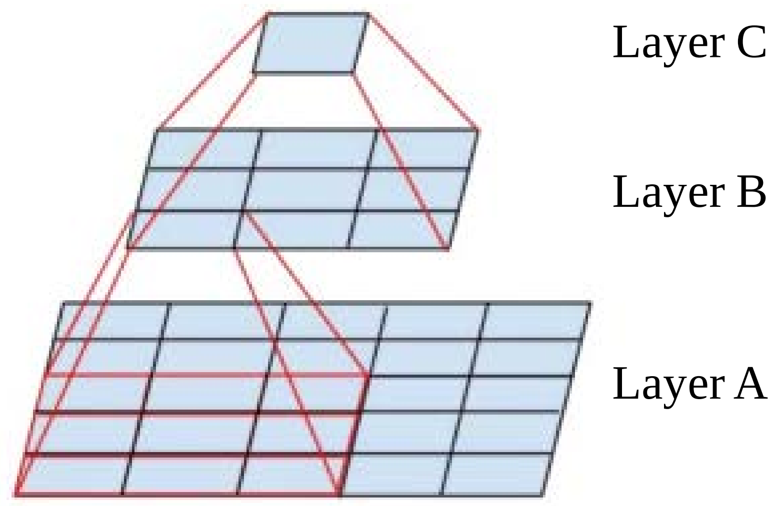
$W \times H \times C_{\text{out}}$

CONV Layers

- **Receptive Field** (RF): the region in the input space that a particular feature point is looking at (i.e. be affected by).

For the feature point in layer C:

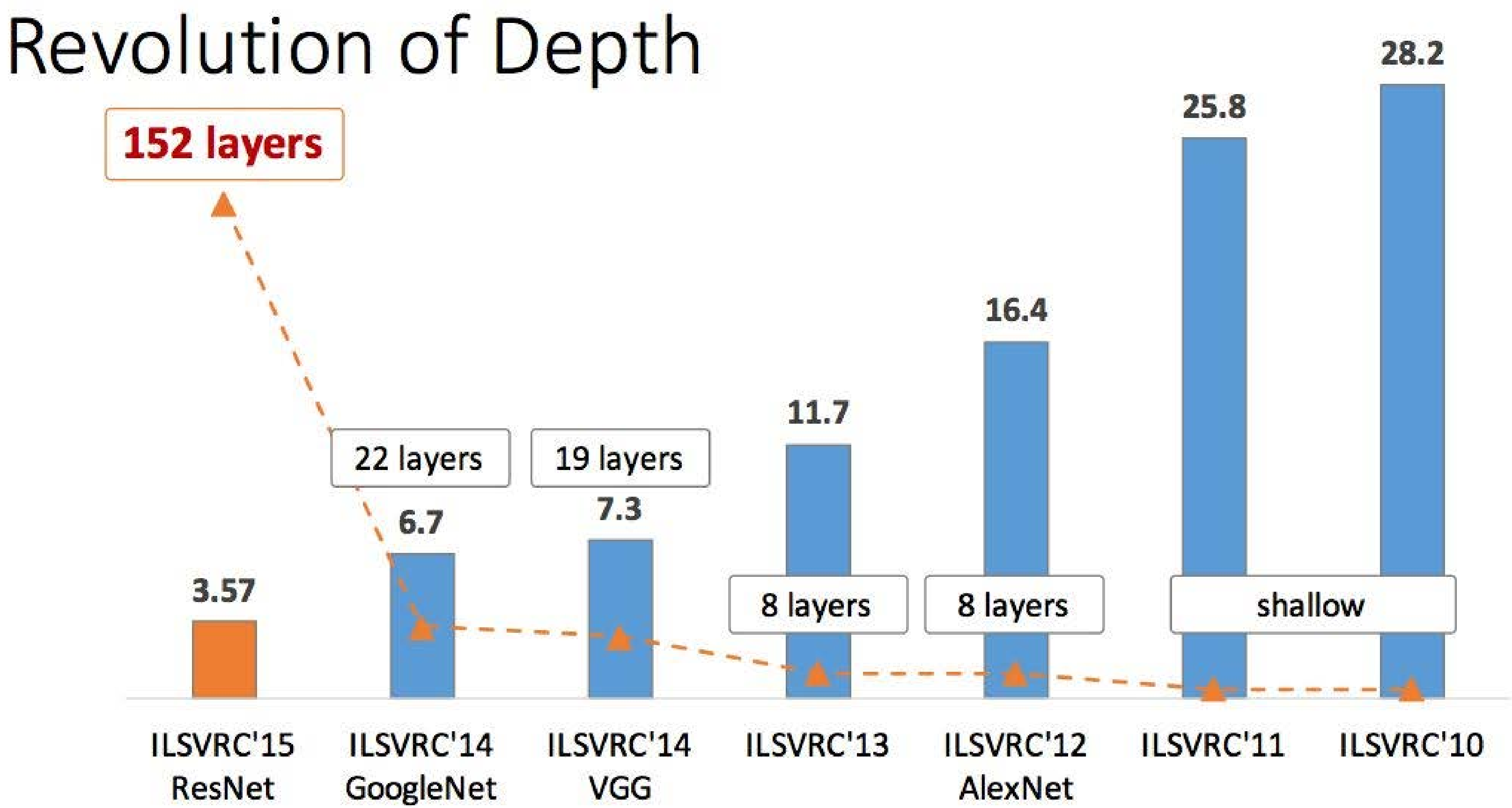
- – RF in layer B: 3 x 3
- – RF in layer A: 5 x 5



ILSVRC

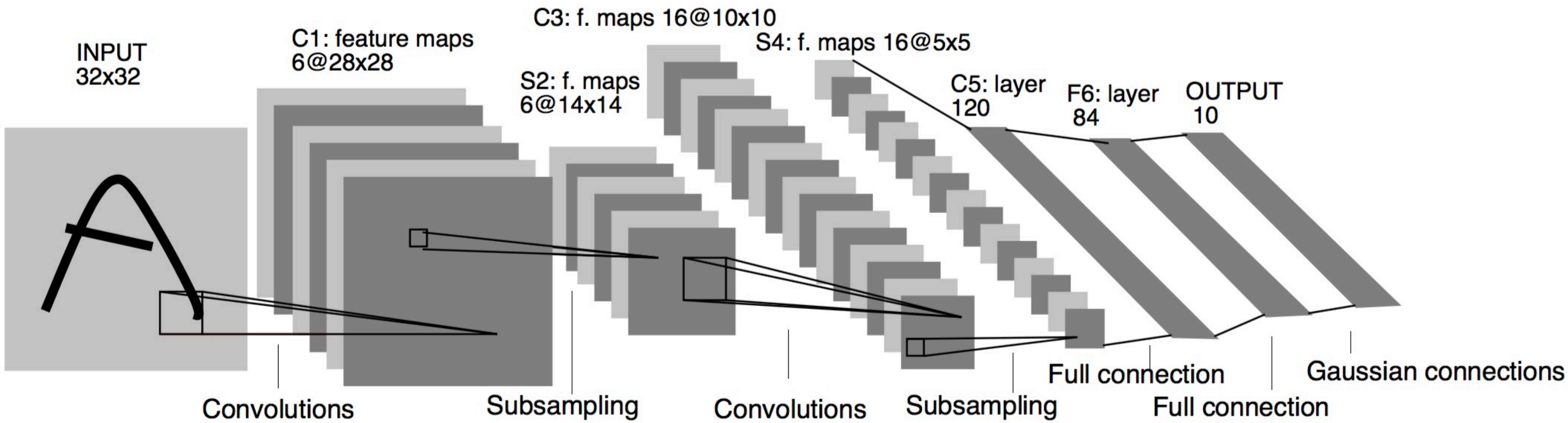
- Image Classification
  - one of the core problems in computer vision
  - many other tasks (such as object detection, segmentation) can be reduced to image classification
- ImageNet Large Scale Visual Recognition Challenge
  - start from 2010 and end in 2017
  - main tasks: classification and detection

ILSVRC



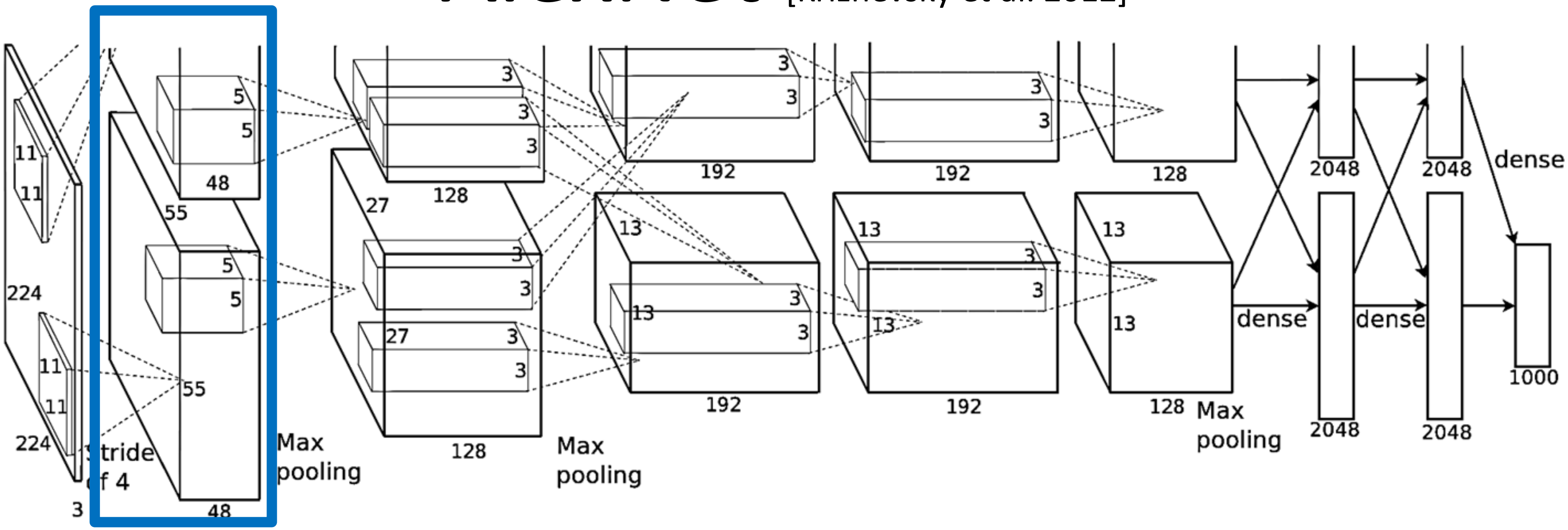
Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15
ResNet
ILSVRC'14
GoogleNet
ILSVRC'14
VGG
ILSVRC'13
ILSVRC'12
AlexNet
ILSVRC'11
ILSVRC'10

LeNet [LeCun et al., 1998]



- Architecture is [CONV-POOL-CONV-POOL-FC-FC]
- CONV: 5x5 filter, stride=1
- POOL: 2x2 filter, stride=2

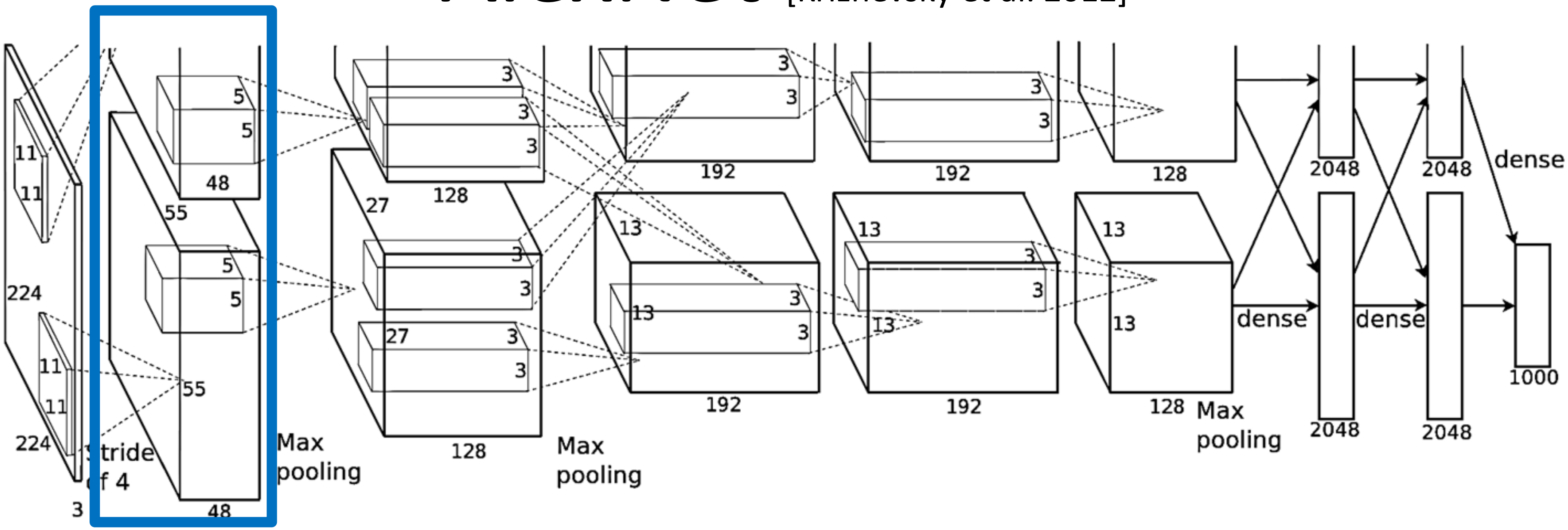
AlexNet [Krizhevsky et al. 2012]



First layer (CONV1): 11x11 filter, stride=4, 96 in total

- Input: 227 x 227 x 3 images
- What is the output volume size? Hint: (227-11)/4+1 = 55
- What is the total number of parameters in this layer?

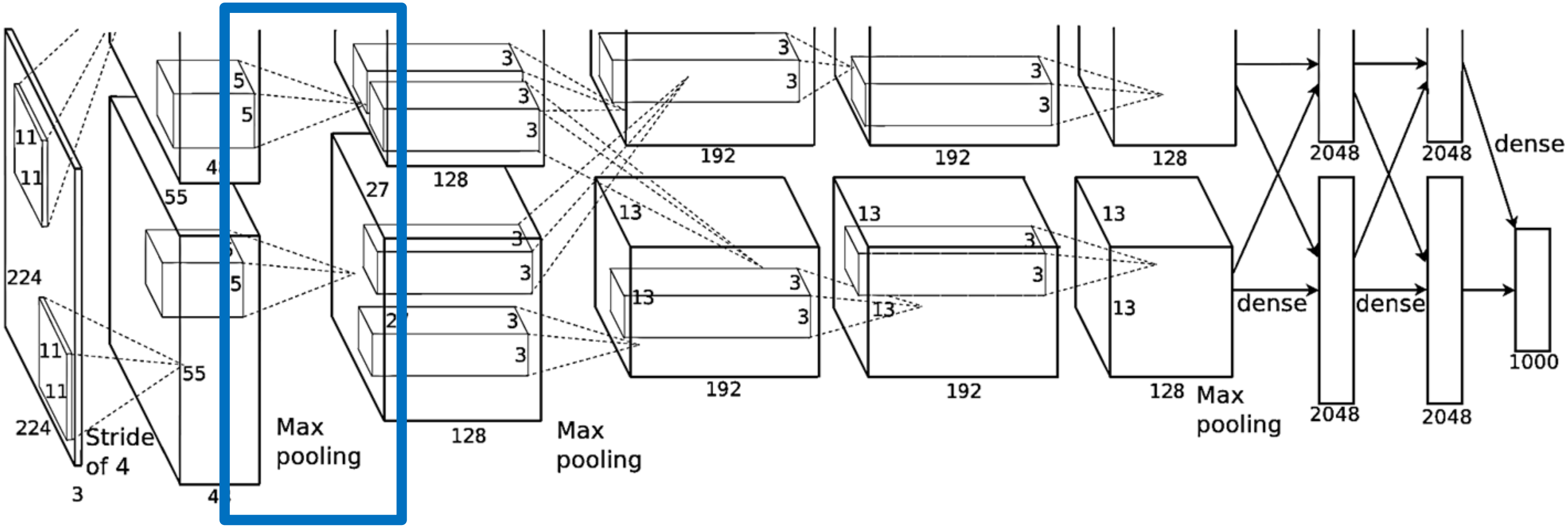
AlexNet [Krizhevsky et al. 2012]



First layer (CONV1): 11x11 filter, stride=4, 96 in total

- Input: 227 x 227 x 3 images
- Output volume: **[55x55x96]**
- Parameters: (11*11*3)*96 = **35K**

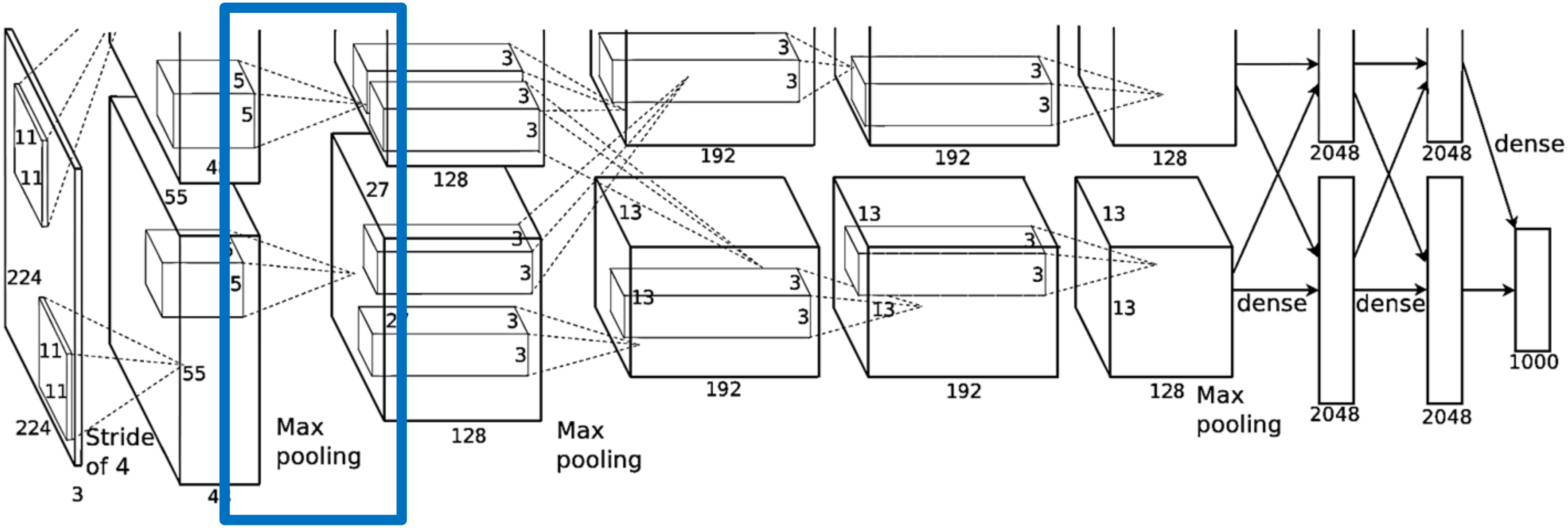
AlexNet [Krizhevsky et al. 2012]



Second layer (POOL1): 3x3 filters, stride=2

- Input: output of CONV1: 55 x 55 x 96
- What is the output volume size? Hint: (55-3)/2+1 = 27
- What is the total number of parameters in this layer?

AlexNet [Krizhevsky et al. 2012]



Second layer (POOL1): 3x3 filters, stride=2

- Input: output of CONV1: 55 x 55 x 96
- Output volume: **[27x27x96]**
- Parameters: 0

AlexNet [Krizhevsky et al. 2012]

Full AlexNet architecture:

- [227x227x3] INPUT
- [55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
- [27x27x96] MAX POOL1: 3x3 filters at stride 2
- [27x27x96] NORM1: Normalization layer
- [27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
- [13x13x256] MAX POOL2: 3x3 filters at stride 2
- [13x13x256] NORM2: Normalization layer
- [13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
- [13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
- [13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
- [6x6x256] MAX POOL3: 3x3 filters at stride 2
- [4096] FC6: 4096 neurons
- [4096] FC7: 4096 neurons
- [1000] FC8: 1000 neurons (class scores)

AlexNet [Krizhevsky et al. 2012]

Details/Retrospectives:

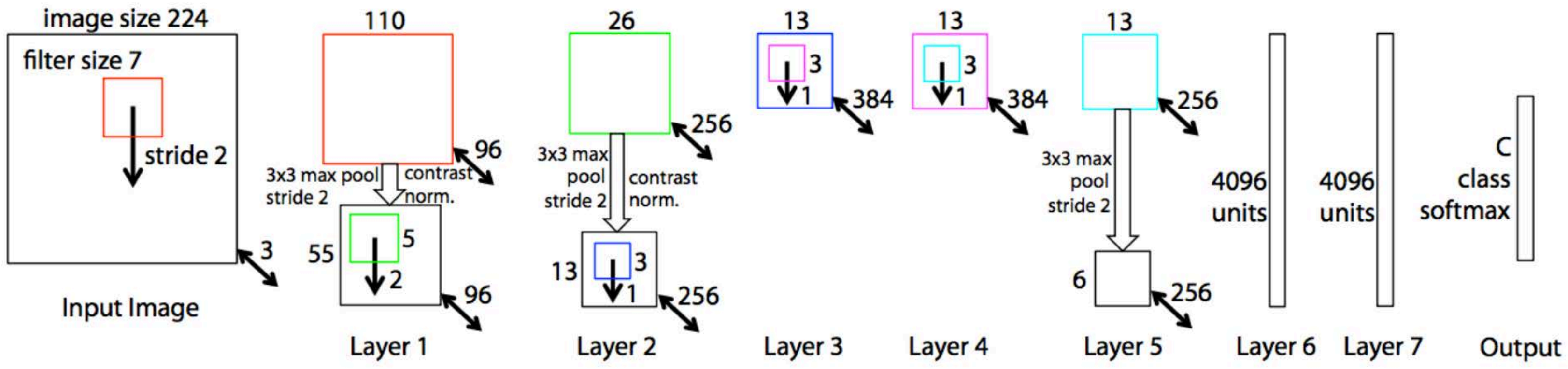
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

AlexNet [Krizhevsky et al. 2012]

Training with GPU

- Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.
- CONV1, CONV2, CONV4, CONV5: Connections only with feature maps on same GPU
- CONV3, FC6, FC7, FC8: Connections with all feature maps in preceding layer, communication across GPUs

ZF-Net [Zeiler and Fergus, 2013]



An improved version of AlexNet:

- First convolutional layer: 11x11 filter, stride=4 -> 7x7 filter, stride=2
- More channels in last 3 three convolutional layers: (384, 384, 256 -> 512, 1024, 512)

Results: top-5 error drops from 16.4% to 11.7%

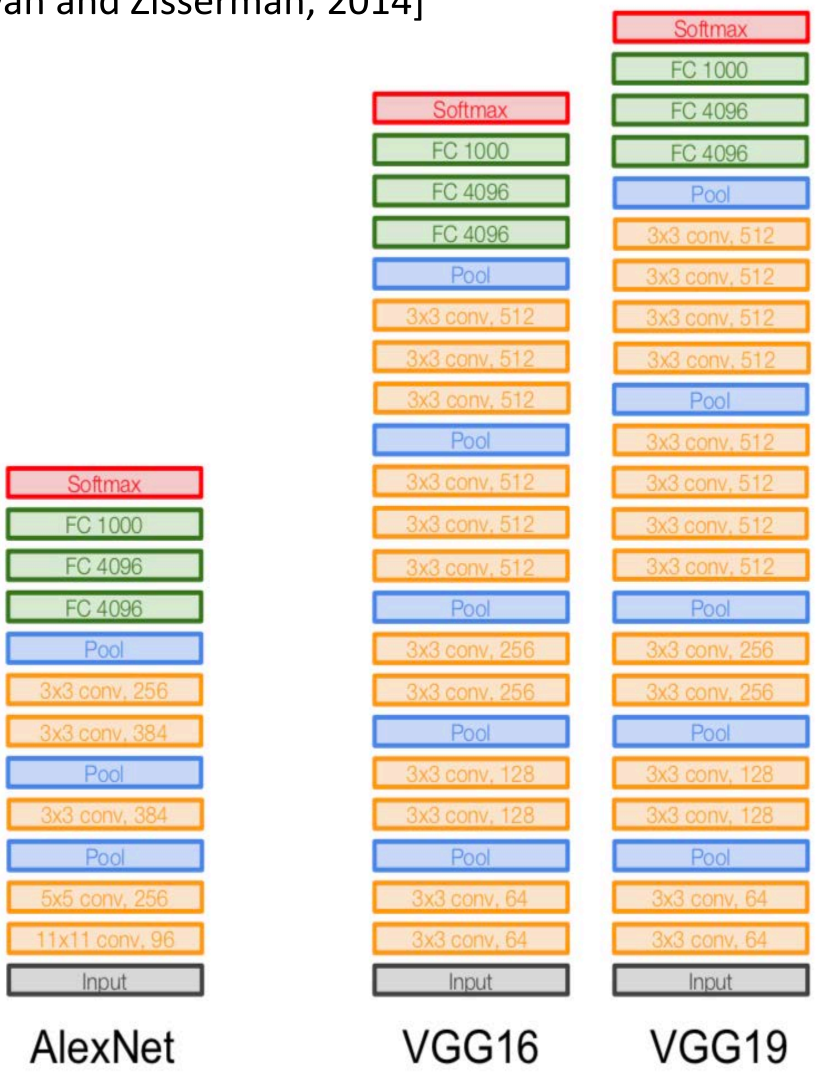
VGGNet [Simonyan and Zisserman, 2014]

Deeper networks:

- AlexNet: 8 layers
- VGGNet: 16 or 19 layers

Small filters:

- 3x3 convolutional layers (stride 1)
- 2x2 max-pooling, stride 2

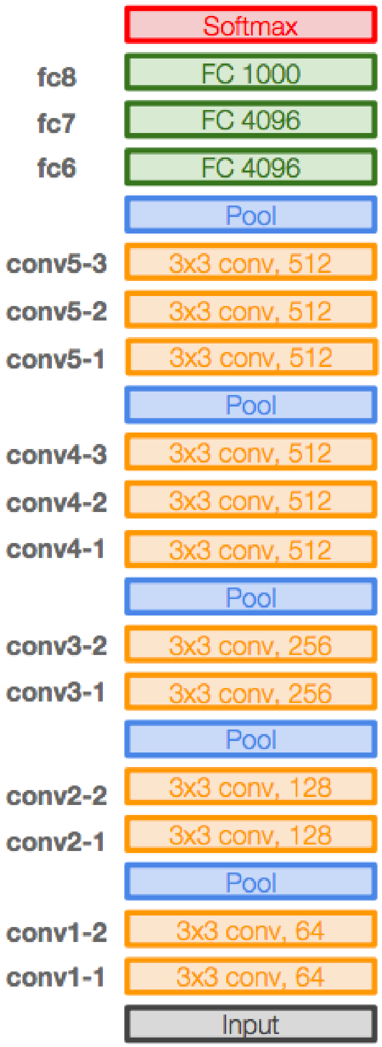


VGGNet [Simonyan and Zisserman, 2014]

Why small filters?

- stack of three 3x3 convolutional (stride 1) layers has the **same effective receptive field** as one 7x7 convolutional layer
- **deeper**, more non-linearities
- **fewer** parameters:

  $3 \times (3^2C^2)$ vs. 7^2C^2 for C channels per layer



VGG16

VGGNet [Simonyan and Zisserman, 2014]

VGG parameters (output size, memory, params)

- INPUT: [224x224x3] memory: 224*224*3=150K params: 0
- CONV1-1: [224x224x64] memory: 224*224*64=**3.2M** params: (3*3*3)*64 = 1,728
- CONV1-2: [224x224x64] memory: 224*224*64=**3.2M** params: (3*3*64)*64 = 36,864
- POOL1: [112x112x64] memory: 112*112*64=800K params: 0
- CONV2-1: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
- CONV2-2: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
- POOL2: [56x56x128] memory: 56*56*128=400K params: 0
- CONV3-1: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
- CONV3-2: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
- CONV3-3: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
- POOL3: [28x28x256] memory: 28*28*256=200K params: 0

What about CONV4, CONV5 and FC6, FC7, FC8?

VGGNet [Simonyan and Zisserman, 2014]

VGG parameters (cont'd) (output size, memory, params)

- CONV4-1: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
- CONV4-2: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
- CONV4-3: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
- POOL4: [14x14x512] memory: 14*14*512=100K params: 0
- CONV5-1: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
- CONV5-2: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
- CONV5-3: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
- POOL5: [7x7x512] memory: 7*7*512=25K params: 0
- FC6: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
- FC7: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
- FC8: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

VGGNet [Simonyan and Zisserman, 2014]

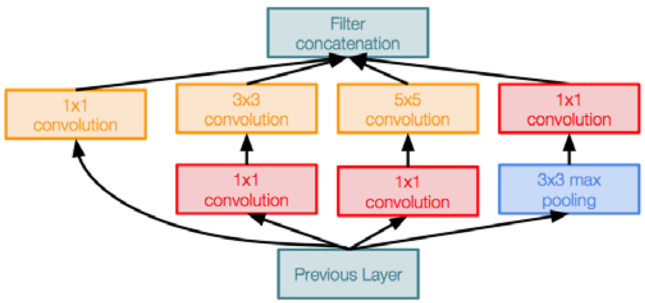
Some comments:

- Total memory: 24M * 4 bytes ~= 96MB / image (which is only forward, and around 2 times for backward)
- Total parameters: 138M parameters
- Most memory is in early CONV layers (CONV1), and most parameters are in late FC layers (FC6)
- No Local Response Normalization (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, but more memory)
- FC7 features generalize well to other tasks

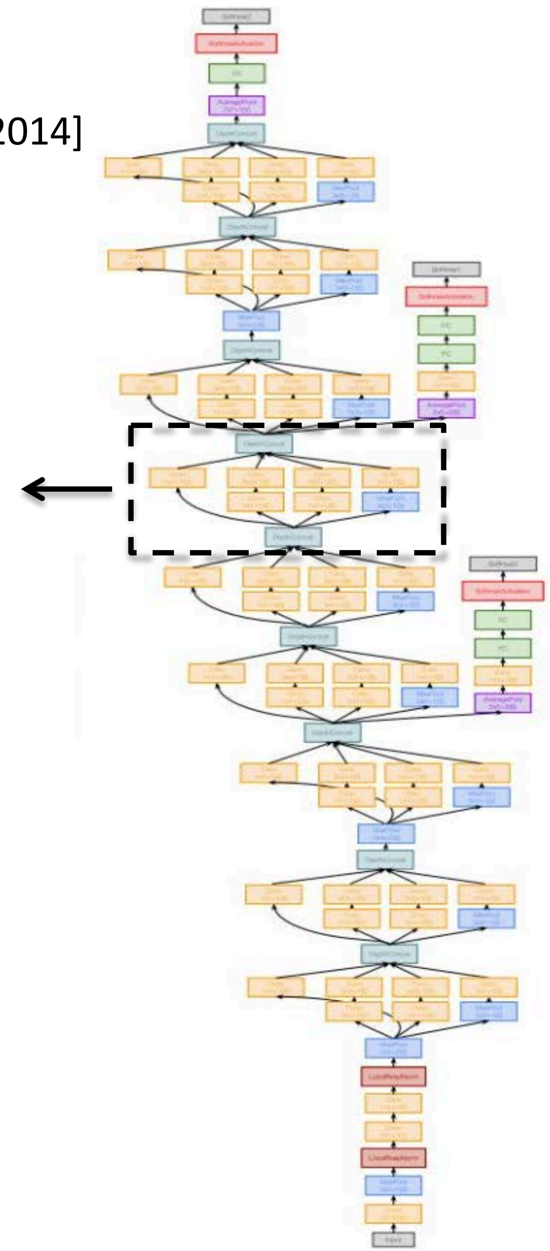
GoogLeNet [Szegedy et al., 2014]

Deeper networks

- 22 layers.
- Auxiliary loss



Inception module



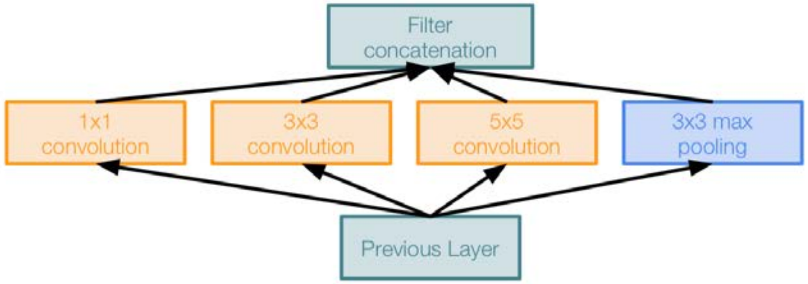
Computational efficiency

- Inception module
- Remove FC layer, use global average pooling
- 5 million parameters, 12x less than AlexNet

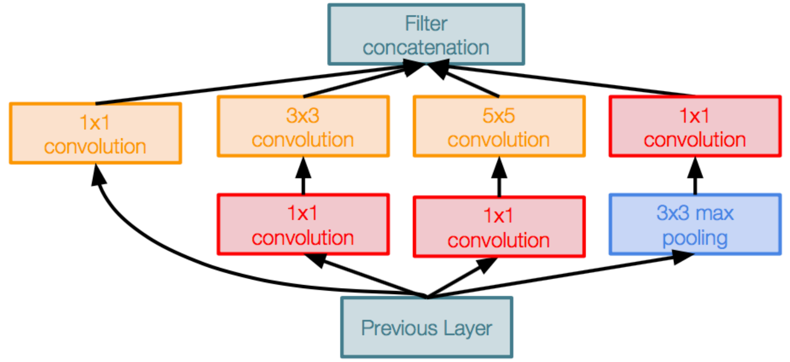
GoogLeNet [Szegedy et al., 2014]

How to build an inception module?

- Apply parallel filter operations on the input from previous layer: multiple receptive field sizes for convolution (1x1, 3x3, 5x5) and pooling operation (3x3)
- Concatenate all filter outputs together depth-wise
- Besides the naïve inception module, 1x1 convolutional layers are used as bottlenecks to reduce parameters.



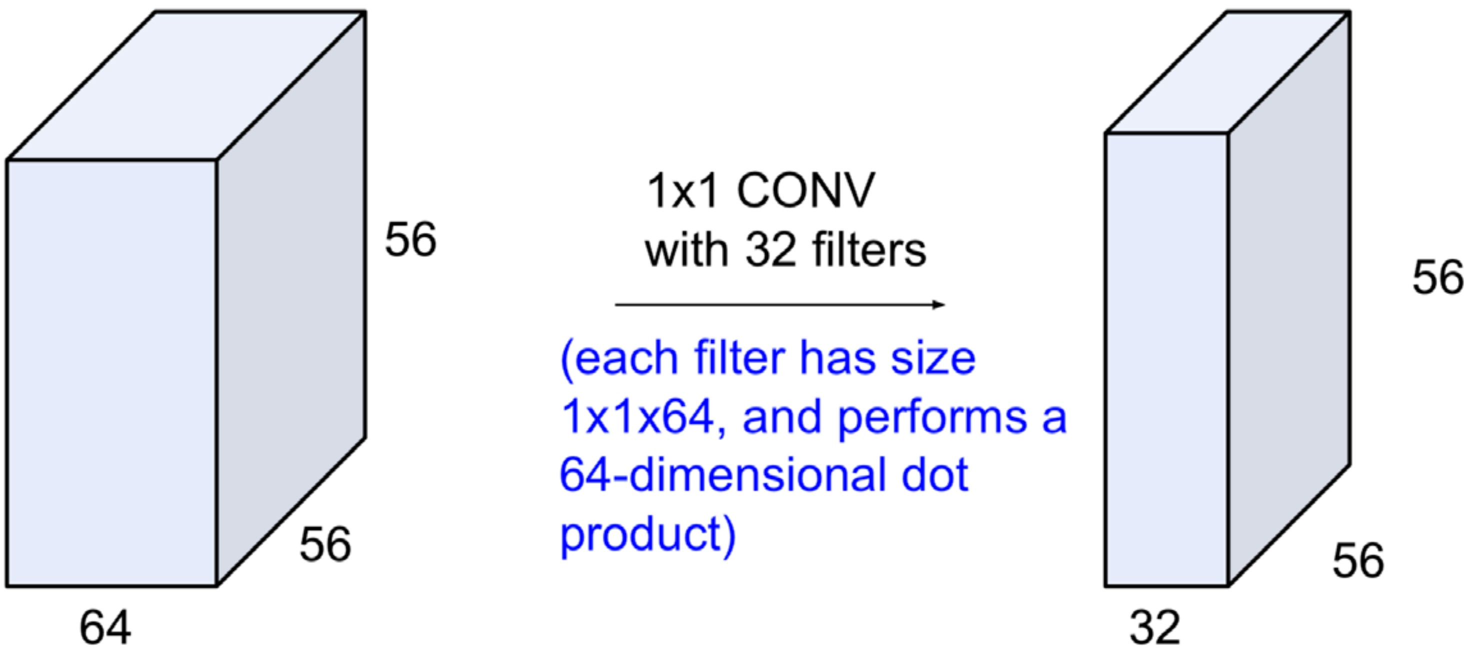
Naive Inception module



Inception module with dimension reduction

GoogLeNet [Szegedy et al., 2014]

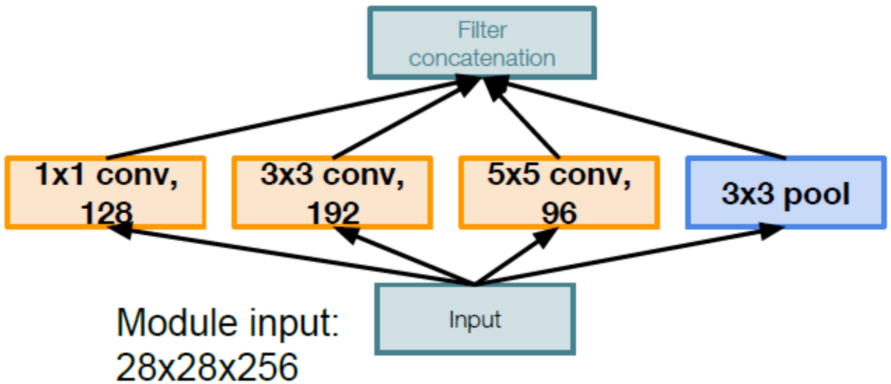
The 1x1 convolutional can preserve spatial dimensions and reduce the depth.



GoogLeNet [Szegedy et al., 2014]

Parameters of the Naïve Inception module:

- The output sizes of all filters:
  - 1x1 conv: 28 x 28 x 128
  - 3x3 conv: 28 x 28 x 192
  - 5x5 conv: 28 x 28 x 96
  - 3x3 pool: 28 x 28 x 256



- The output size after concatenating all filters:
  - 28x28x(128+192+96+256) = 28x28x672
- Total convolution operations: **854M**
  - [1x1 conv, 128] 28x28x128 x 1x1x256
  - [3x3 conv, 192] 28x28x192 x 3x3x256
  - [5x5 conv, 96] 28x28x96 x 5x5x256

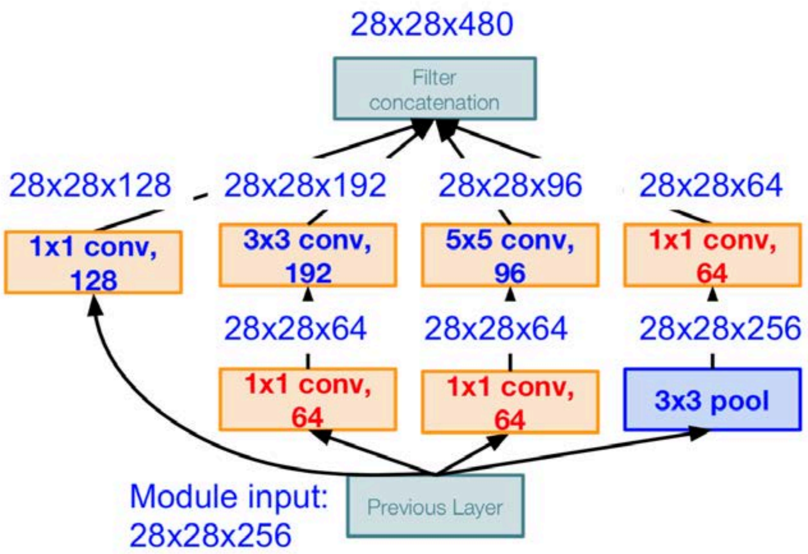
The sizes of output

The sizes of each filter

GoogLeNet [Szegedy et al., 2014]

Parameters of the Inception module (with bottelneck):

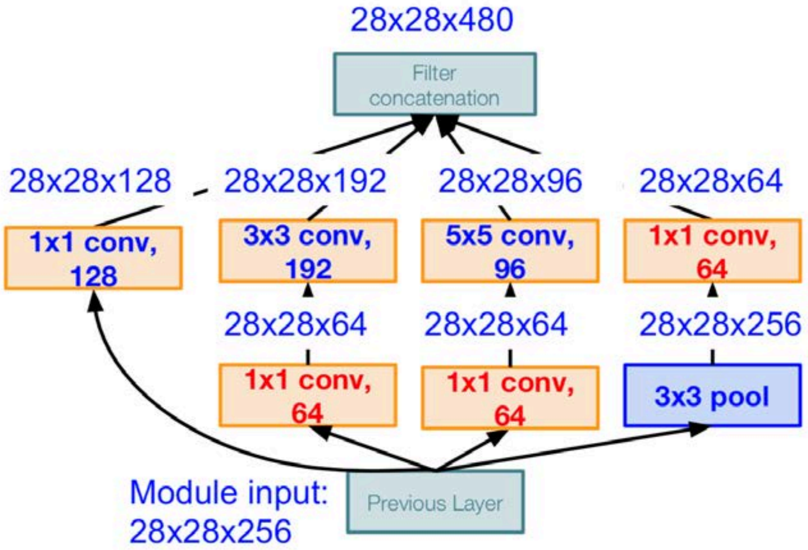
- The output sizes of all filters are as shown.
- Total convolution operations: **358M < 854M**
  - [1x1 conv, 64]
  - [1x1 conv, 64]
  - [1x1 conv, 128]
  - [3x3 conv, 192]
  - [5x5 conv, 96]
  - [1x1 conv, 64]



GoogLeNet [Szegedy et al., 2014]

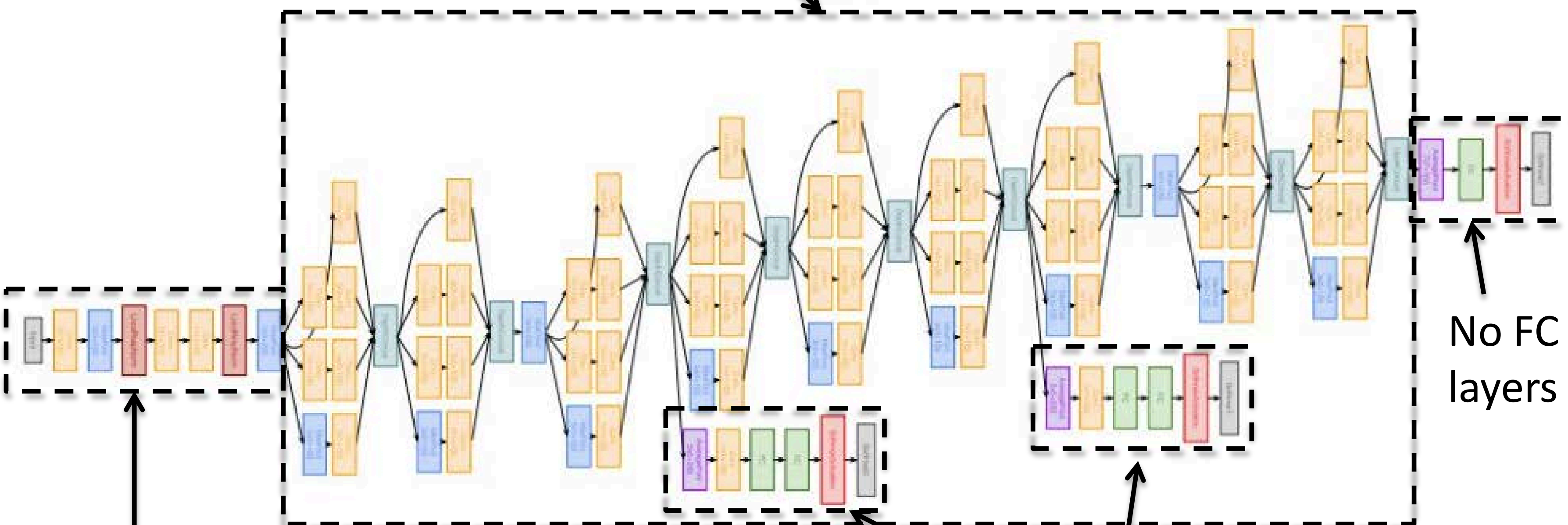
Parameters of the Inception module (with bottleneck):

- The output sizes of all filters are as shown.
- Total convolution operations: **358M < 854M**
  - [1x1 conv, 64] 28x28x64x1x1x256
  - [1x1 conv, 64] 28x28x64x1x1x256
  - [1x1 conv, 128] 28x28x128x1x1x256
  - [3x3 conv, 192] 28x28x192x3x3x64
  - [5x5 conv, 96] 28x28x96x5x5x64
  - [1x1 conv, 64] 28x28x64x1x1x256



GoogLeNet [Szegedy et al., 2014]

Stacked Inception modules



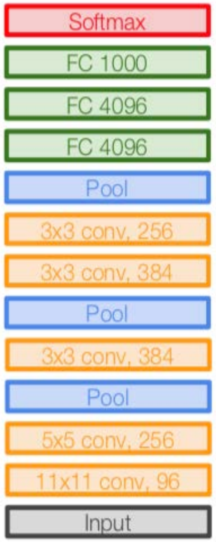
No FC layers

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

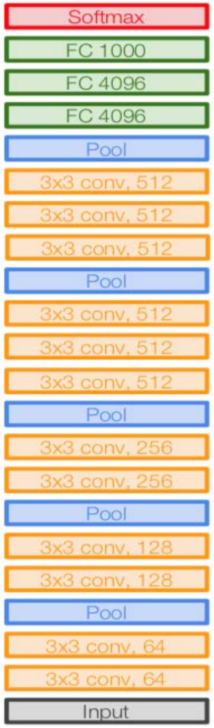
Auxiliary classification outputs to
inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

Revolution of Depth

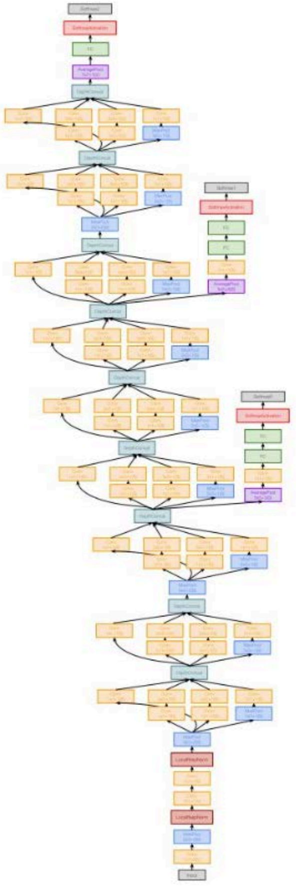
The deeper model should be able to perform at least as well as the shallower model.



AlexNet (9 layers)
top-5 error rate 16.4%



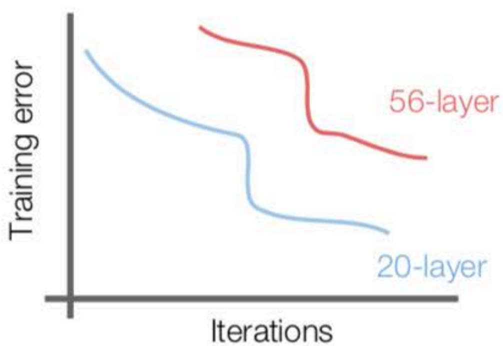
VGG16 (16 layers)
top-5 error rate 7.3%

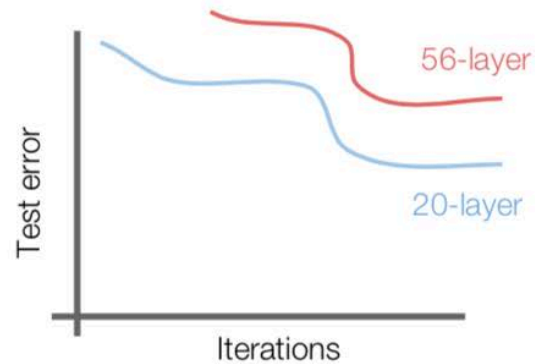


GoogLeNet (22 layers)
top-5 error rate 6.7%

Revolution of Depth

However, the strange things happen when we stack deeper layers on a "plain" convolutional neural network. The deeper model performs worse on both training and test error, but it's not caused by overfitting.



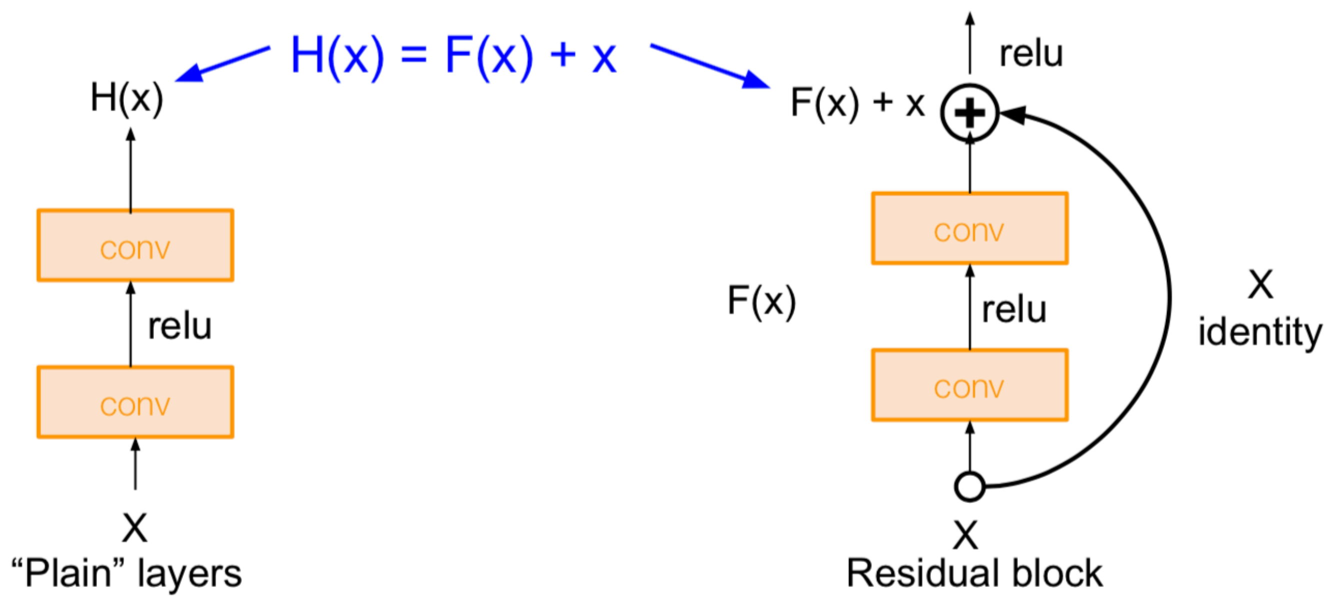


We have a hypothesis that **deeper models are harder to optimize**.

A possible solution is copying the learned layers from the shallower model and setting additional layers to identity mapping.

ResNet [He et al., 2015]

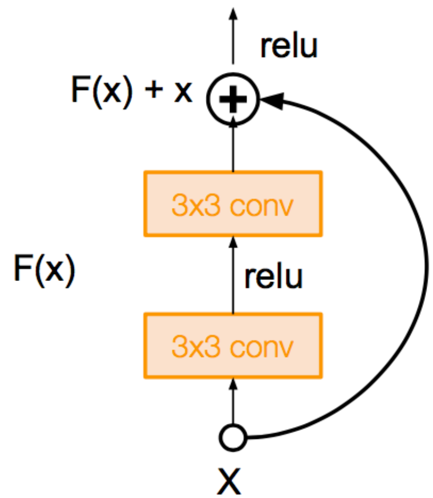
- ResNet uses network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping.
- For example, here ResNet try to fit residual F(x) = H(x) – x instead of H(x) directly



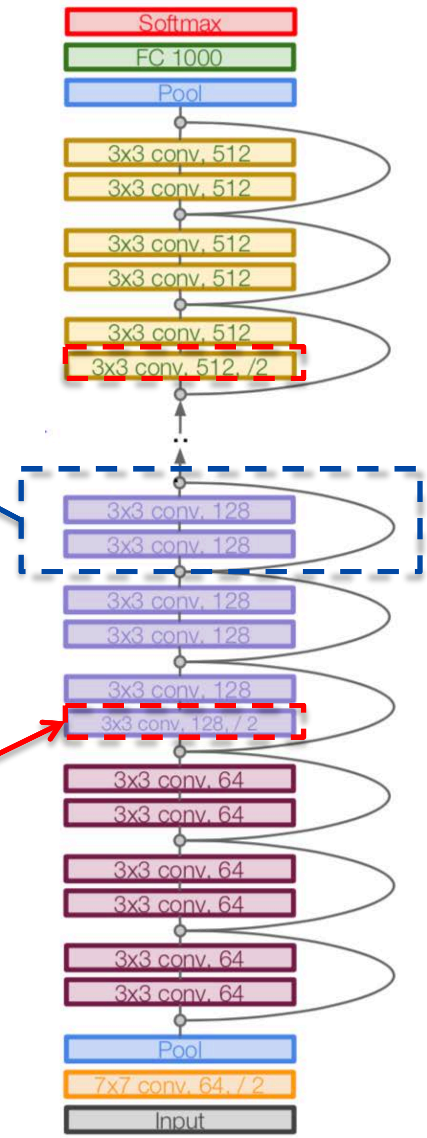
ResNet [He et al., 2015]

Basic design:

- Stack residual blocks
- Each block has two 3x3 conv layer
- No dropout layers
- No FC layers at the end, use average pooling instead (only FC 1000 to output classes)
- Additional conv layer at the beginning
- Periodically, double number of filters and downsample spatially using stride 2 (/2 in each dimension)
- Total depths are different in many versions: 34, 50, 101, 152



Residual block

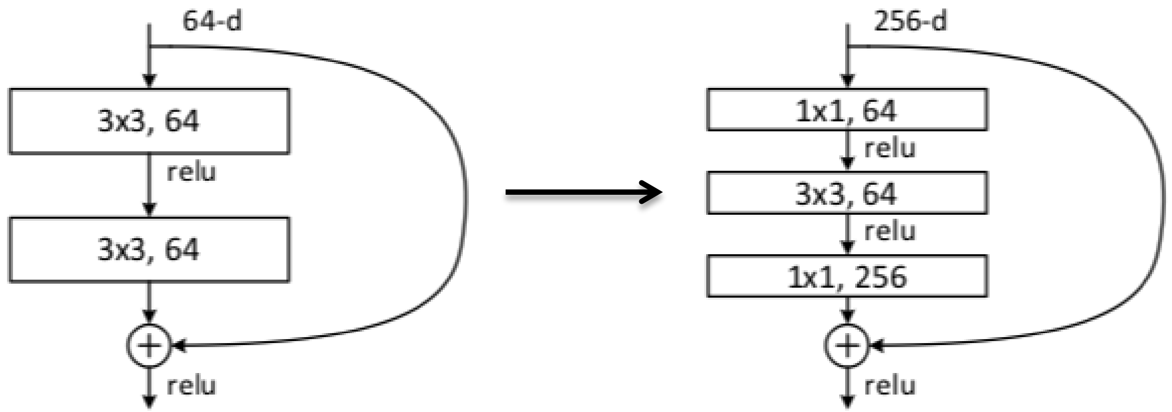


ResNet

ResNet [He et al., 2015]

For deeper networks (ResNet-50 and more), they also use "**bottleneck**" layer to improve efficiency (similar to GoogLeNet).

Suppose the input is 28x28x256, what are the differences between the two modules?



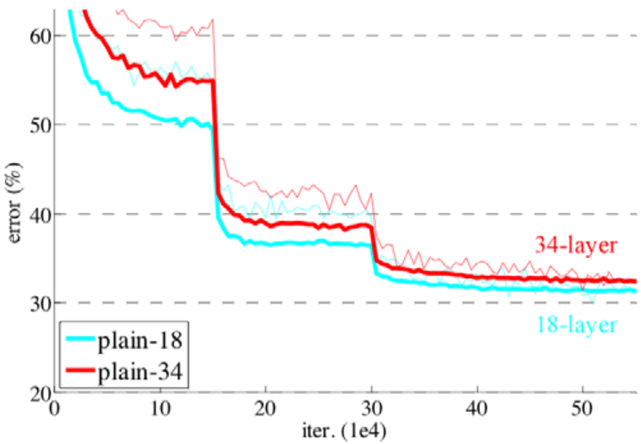
ResNet [He et al., 2015]

Training ResNet in practice:

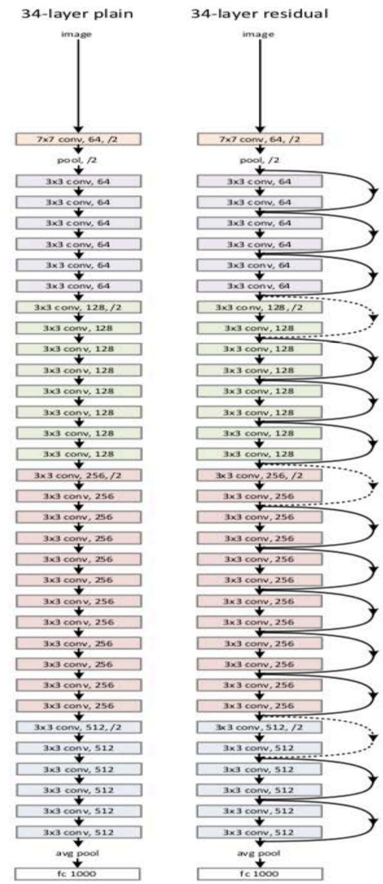
- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

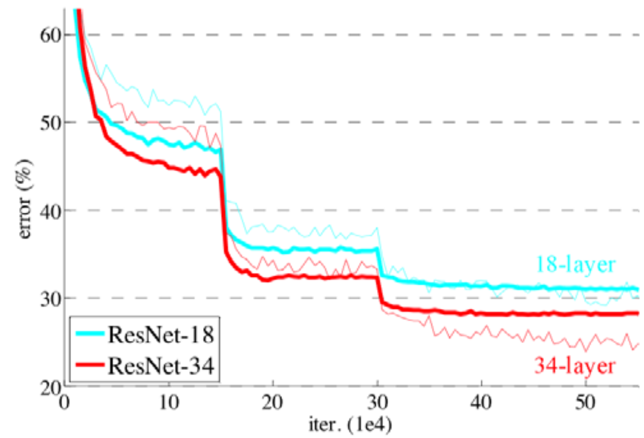
ResNet [He et al., 2015]

ResNet solved the optimization problem for deeper neural networks perfectly!



deeper != better



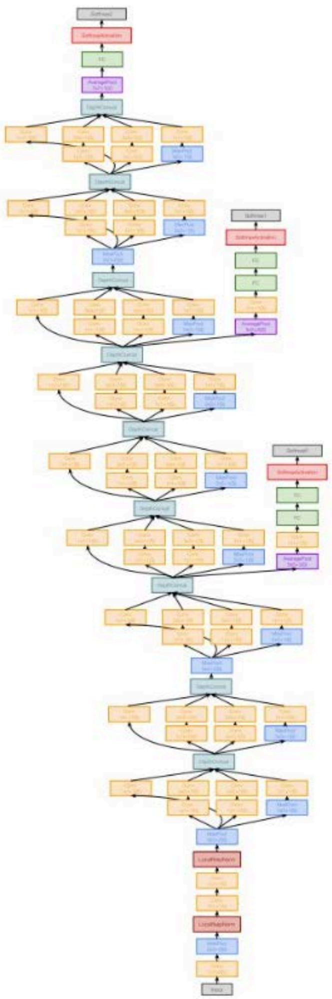


deeper = better

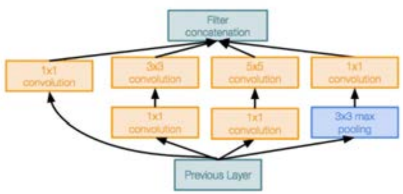
ResNet [He et al., 2015]

- ResNet is able to train very deep networks without degrading (152 layers on ImageNet)
- ResNet swept 1st place in all ILSVRC and COCO 2015 competitions.
- ResNet is ILSVRC 2015 classification winner (3.6%
- top 5 error), even better than "human performance"!
    - **1st places in all five main tracks**
        - ImageNet Classification: "*Ultra-deep*" 152-layer nets
        - ImageNet Detection: 16% better than 2nd
        - ImageNet Localization: 27% better than 2nd
        - COCO Detection: 11% better than 2nd
        - COCO Segmentation: 12% better than 2nd

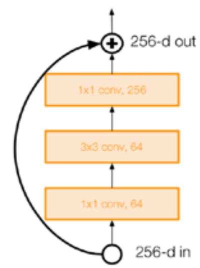
GoogLeNet vs ResNet



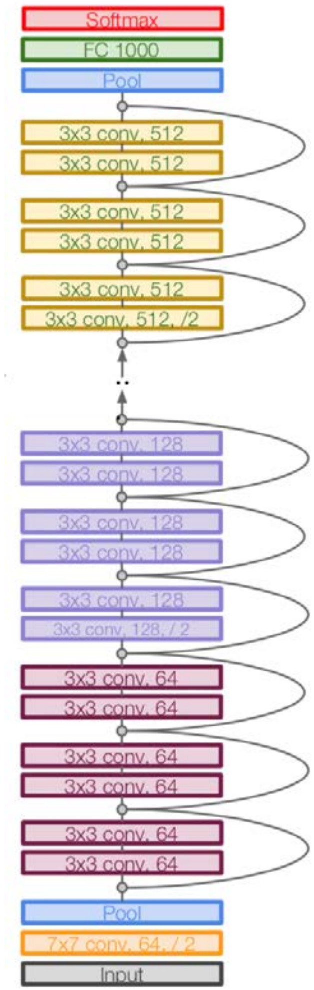
Stacked modules or blocks.



Inception module



Residual block



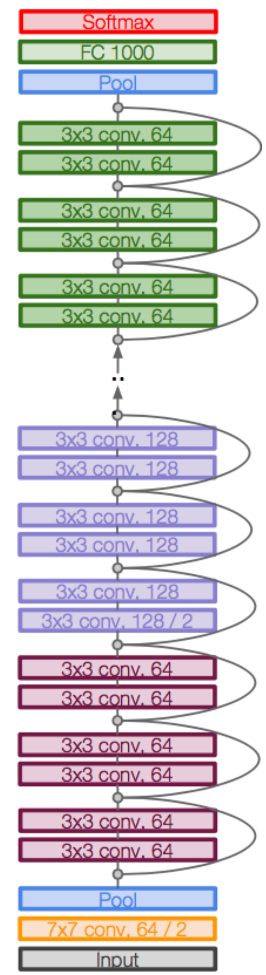
Advanced models

Inception-based:

- Inception v1,v2,v3,v4
- Inception-res-v1,v2

ResNet-based:

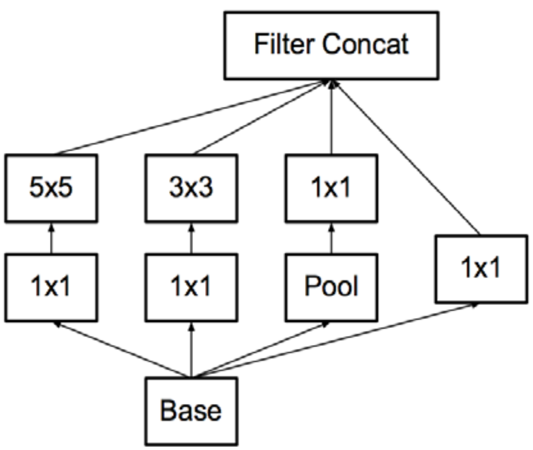
- ResNeXt
- DenseNet



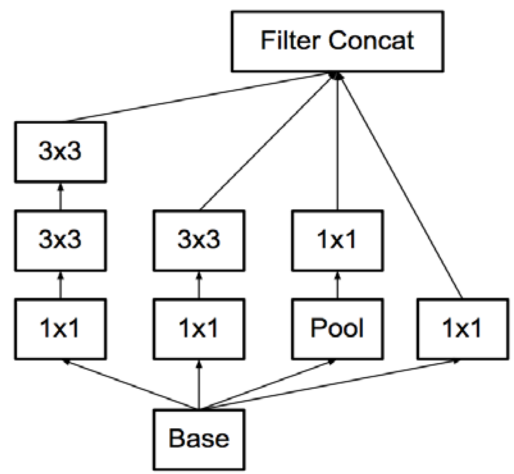
Inception

Inception v2 [Ioffe et al., 2015]

- Add batch normalization
- Remove LRN and dropout
- Use small kernels (inspired by VGG)



Inception module v1
(In GoogLeNet)

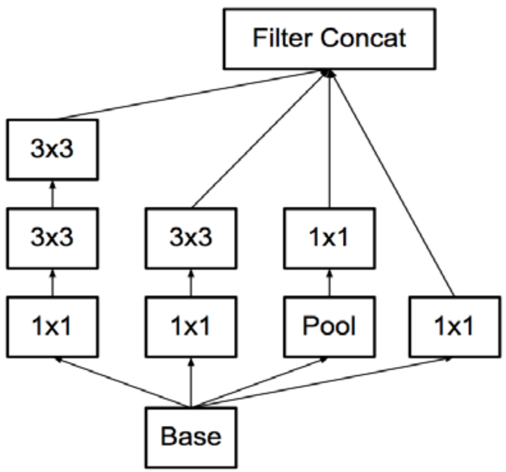


Inception module v2

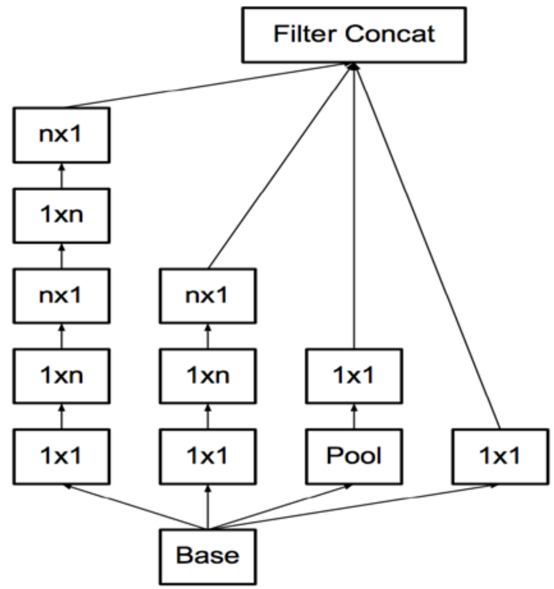
Inception

Inception v3 [Szegedy et al. 2015]

- new inception modules
- deeper with dropout



Inception module v2

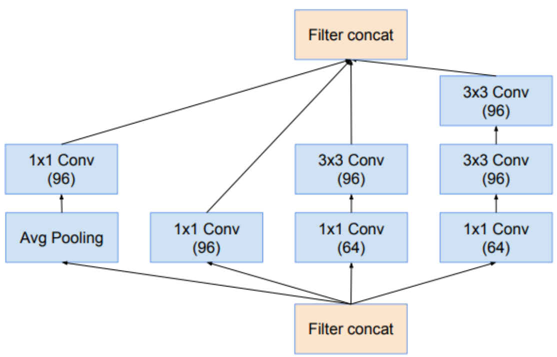


Inception module v3

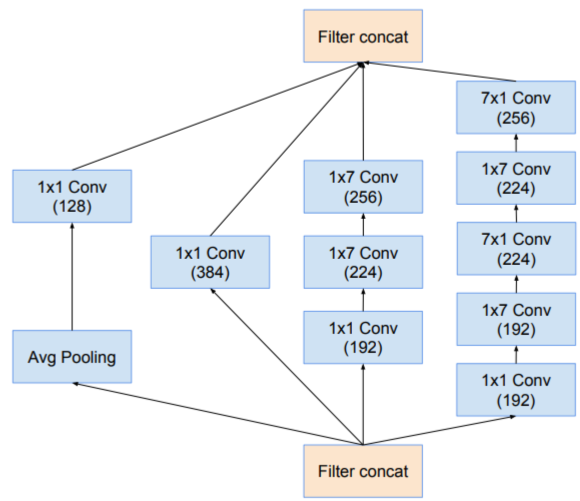
Inception

Inception v4 [Szegedy et al. 2016]

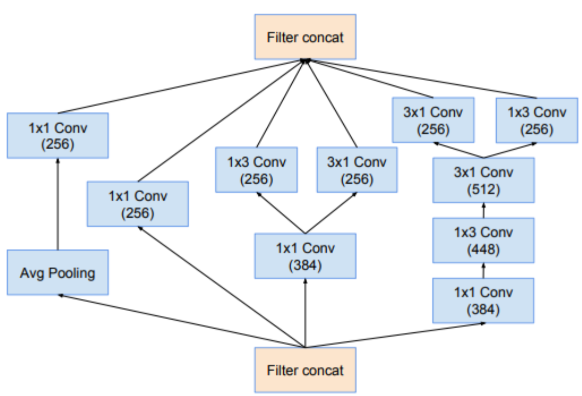
- Deeper structure with special inception modules



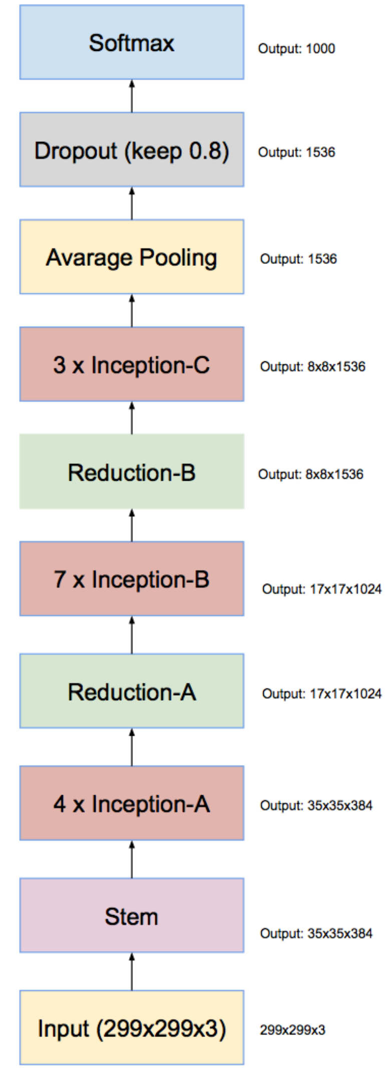
Inception-A



Inception-B



Inception-C

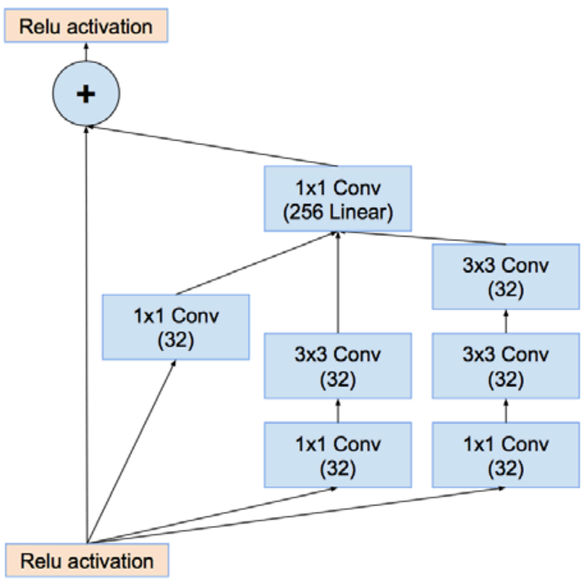


Inception V4

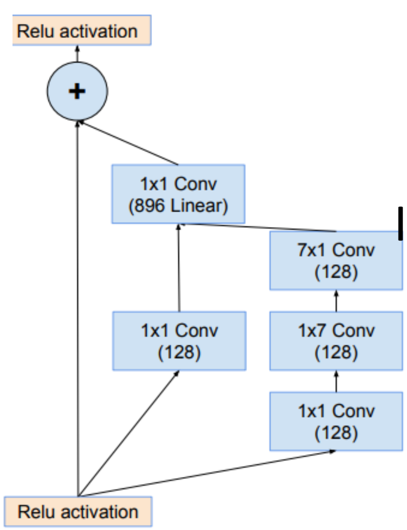
Inception

Inception-ResNet v1, v2 [Szegedy et al. 2016]

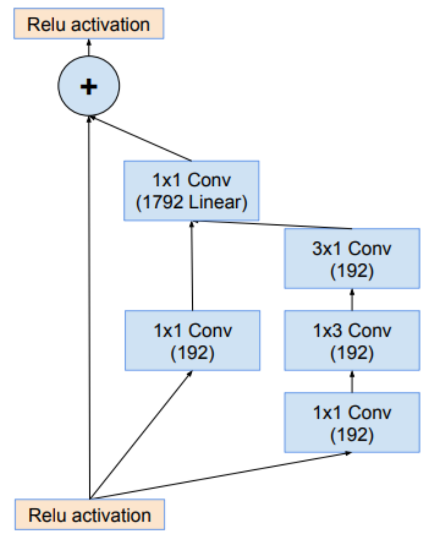
- Inception module with skip connection



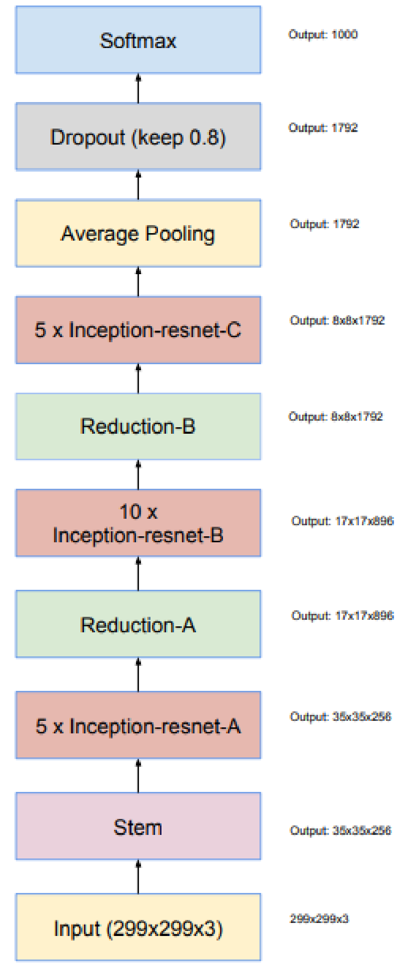
Inception-ResNet-A



Inception-ResNet-B



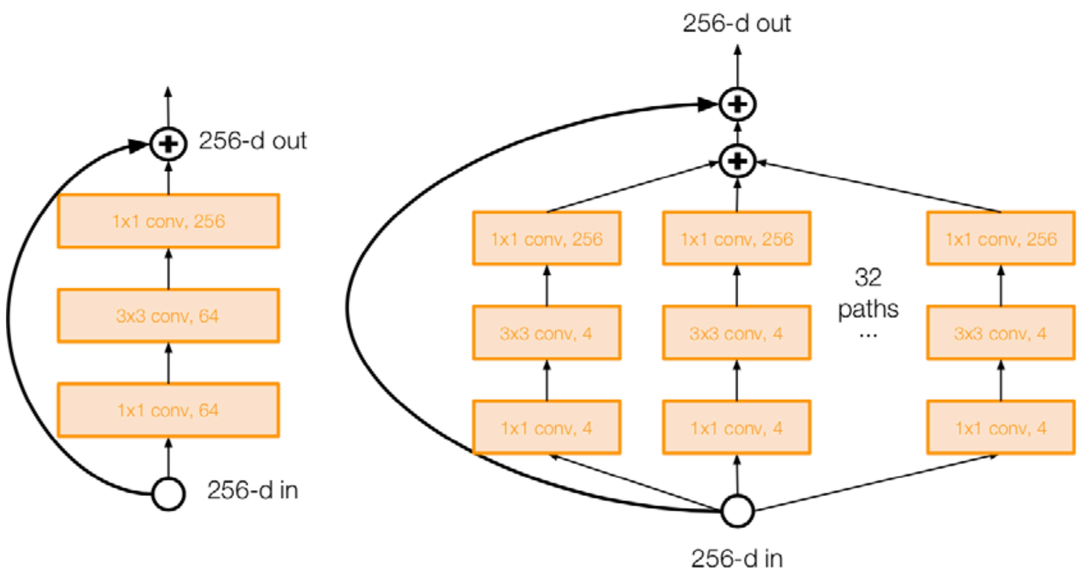
Inception-ResNet-C



Inception-ResNet

ResNeXt [Xie et al. 2016]

- Increases width of residual block through multiple parallel pathways
- Parallel pathways similar in spirit to Inception module



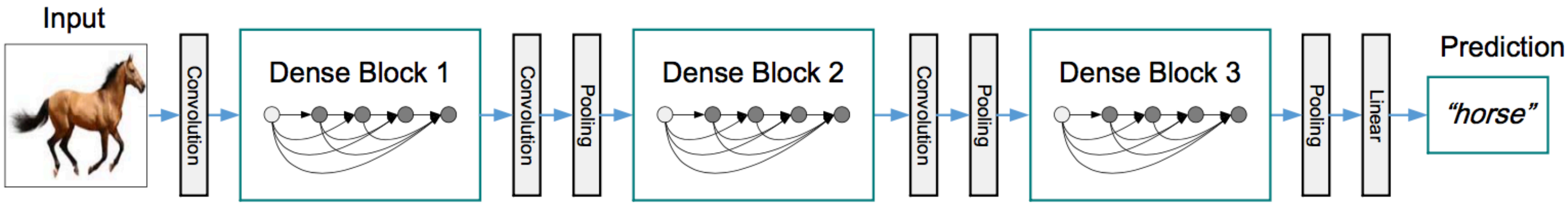
ResNeXt [Xie et al. 2016]

The structure of ResNetXt-50 compared with ResNet-50:

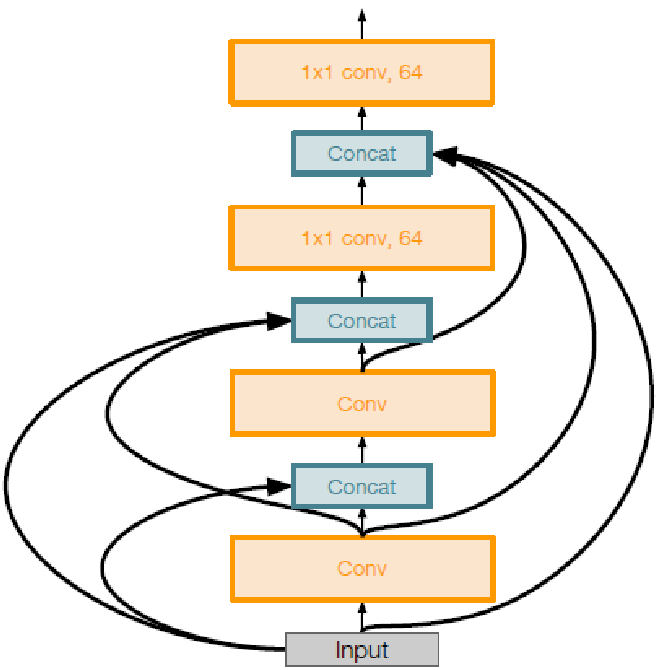
- The complexity are quite similar
- The top-1 error of ResNeXt-50 on ImageNet-1k is 22.2%, better than that of ResNet-50 (23.9%)

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

DenseNet [Huang et al. 2017]



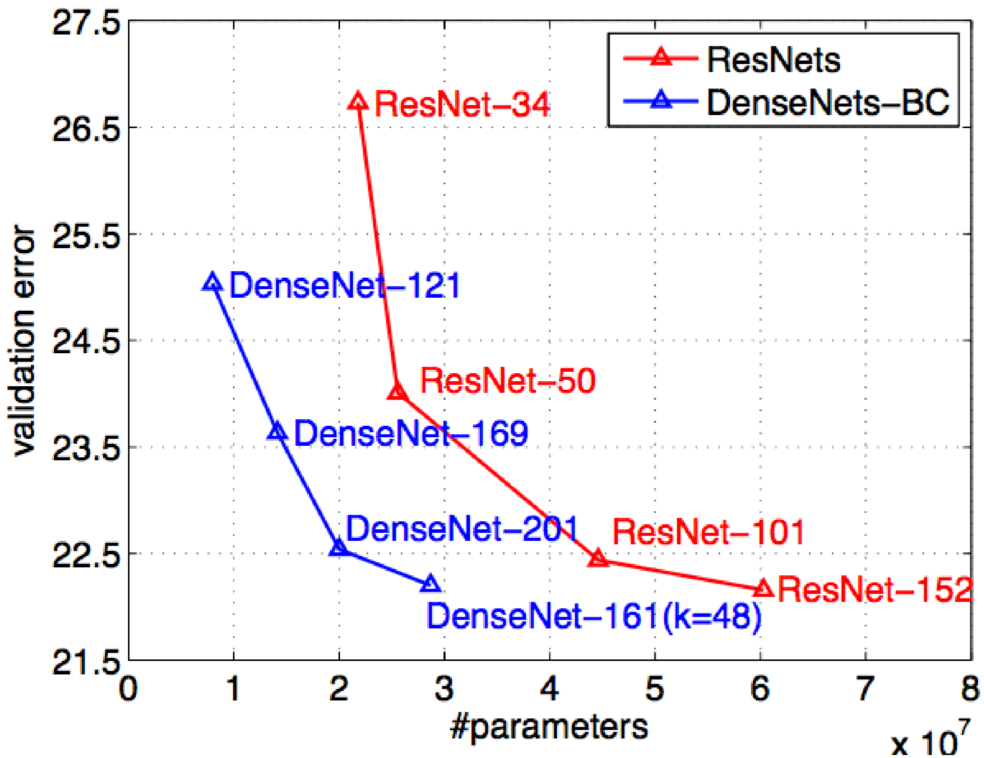
- Use dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient
- strengthens feature propagation
- encourages feature reuse
- reduce the number of parameters



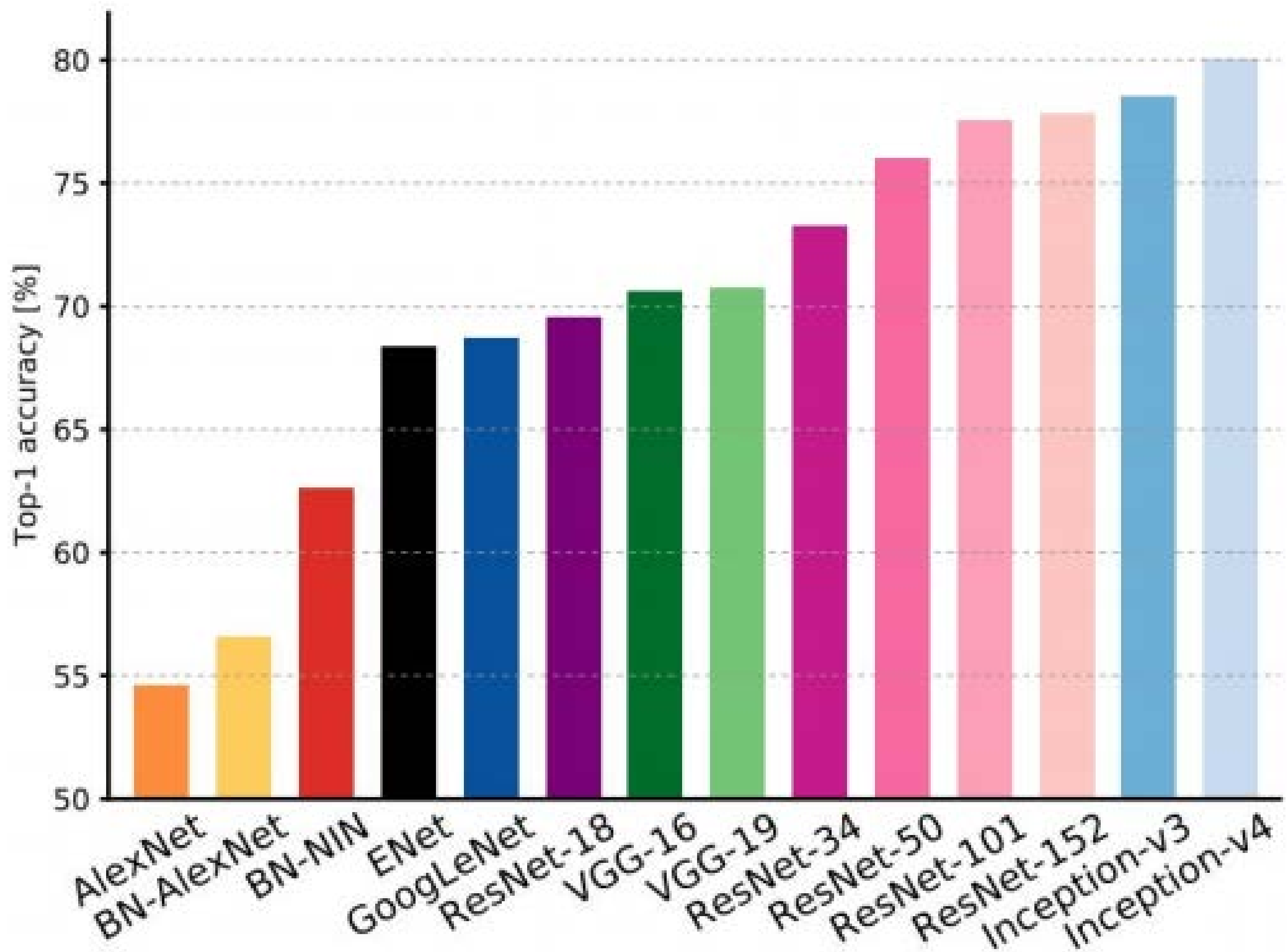
Dense Block

DenseNet [Huang et al. 2017]

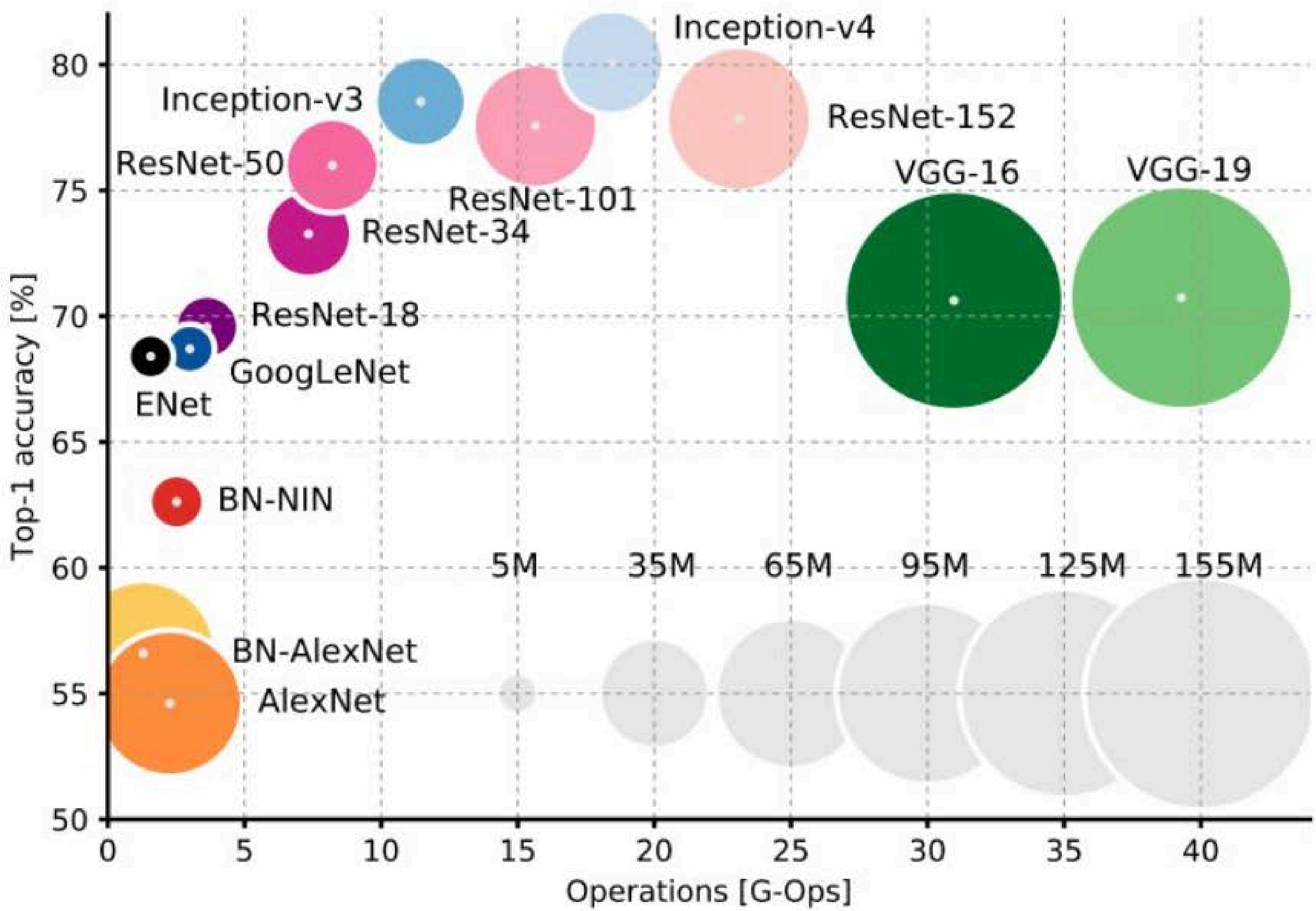
- Comparison of the DenseNet and ResNet on model size: same parameters, better performance



Compare accuracy



Compare complexity



Compare complexity

- AlexNet: smaller compute, still memory heavy, lower accuracy
- VGG: highest memory, most operations
- GoogLeNet: most efficient
- ResNet: moderate efficiency depending on model, highest accuracy

Summary

- Convolution and CONV layers
- Introduction of ILSVRC
- ILSVRC winners
- Advanced networks: Inception vs ResNet
- Complexity comparison