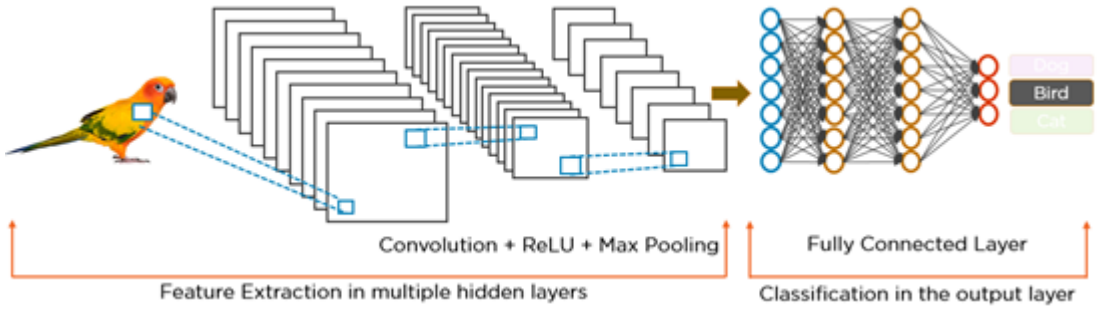
Convolutional Neural Network

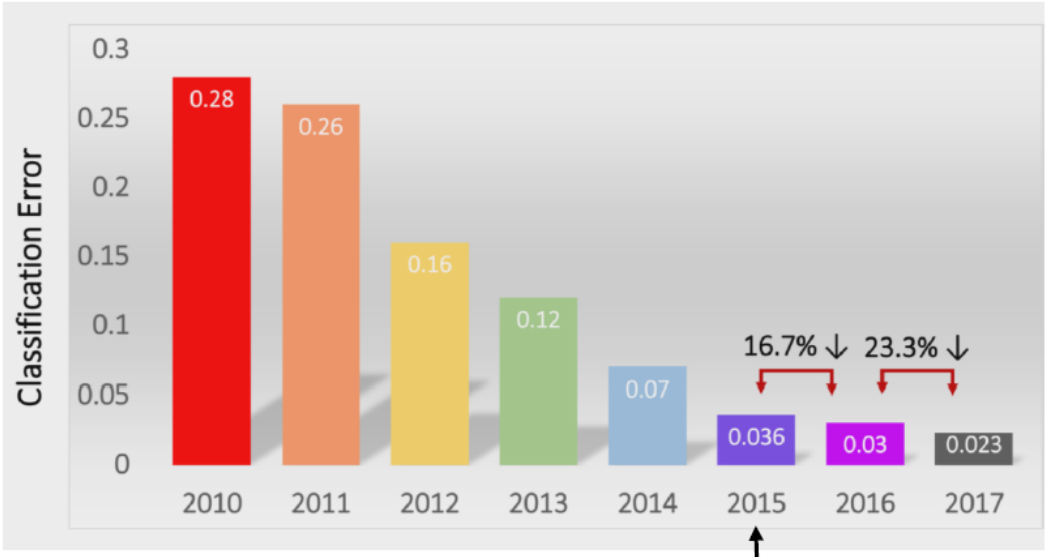
Case Study



Large Scale Image Classification

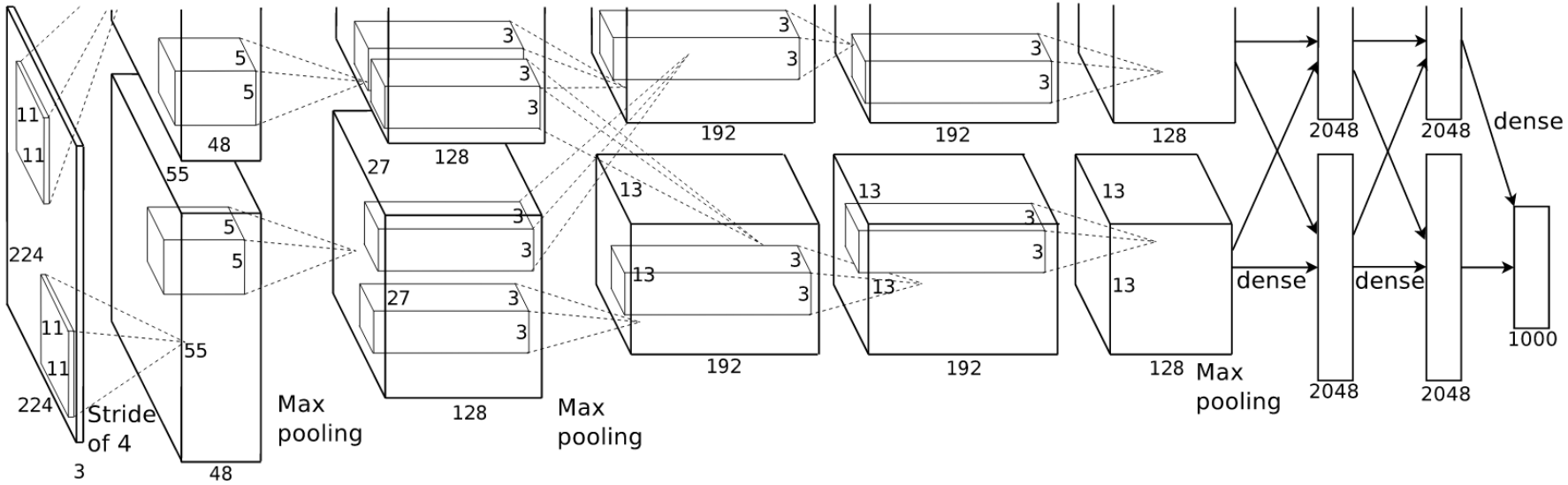
- ImageNet
  - Over 15 million labeled high-resolution images
  - Roughly 22,000 categories
  - Collected from the web
  - Labeled by human labelers using Amazon's Mechanical Turk crowd-sourcing tool
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)
  - Uses a subset of ImageNet
    - 1,000 categories
    - 1.2 million training images
    - 50,000 validation images
    - 150,000 test images
  - Report two error rates:
    - Top-1 and top-5

ImageNet Classification Results



- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models
- **SENet (2017): 2.99% to 2.251%**
  - Squeeze and excitation block: network is allowed to adaptively adjust the weighting of each feature map in the convolutional block.

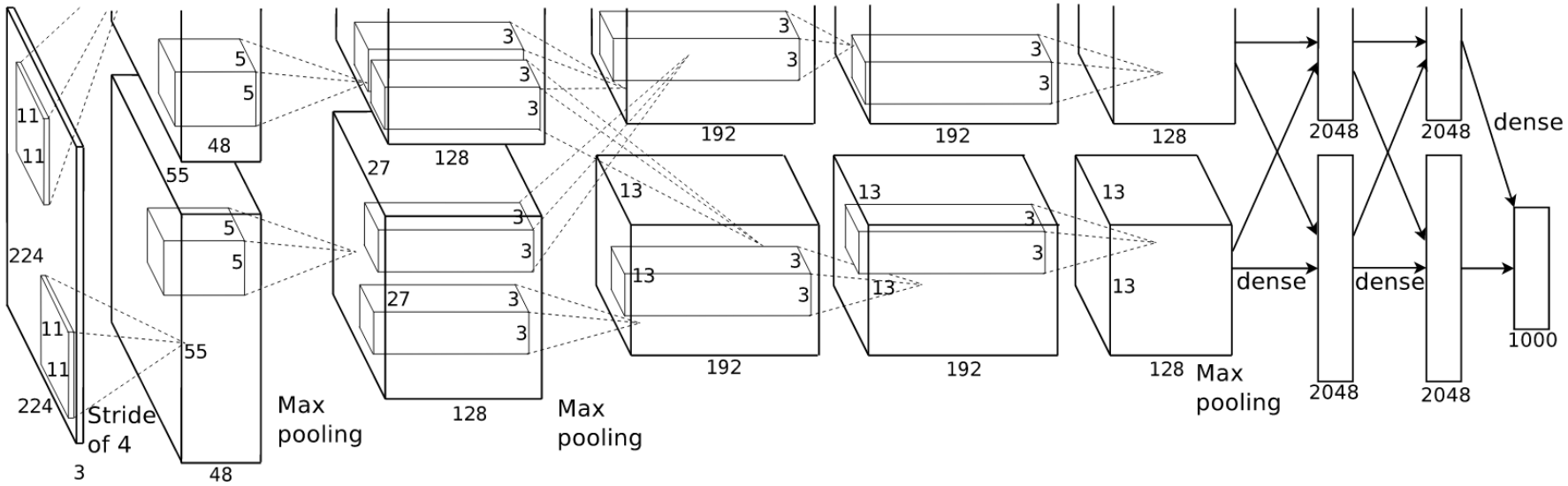
AlexNet (Krizhevsky, 2012)



- First use of ReLU
- Used norm layers(not common anymore)
- Data Augmentation
- Dropout 0.5
- Batch size 128
- SGD Momentum 0.9
- Learning rate 0.01, reduced by 10
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% → 15.4%

Slide Credit : Stanford CS231n

AlexNet (Krizhevsky, 2012)



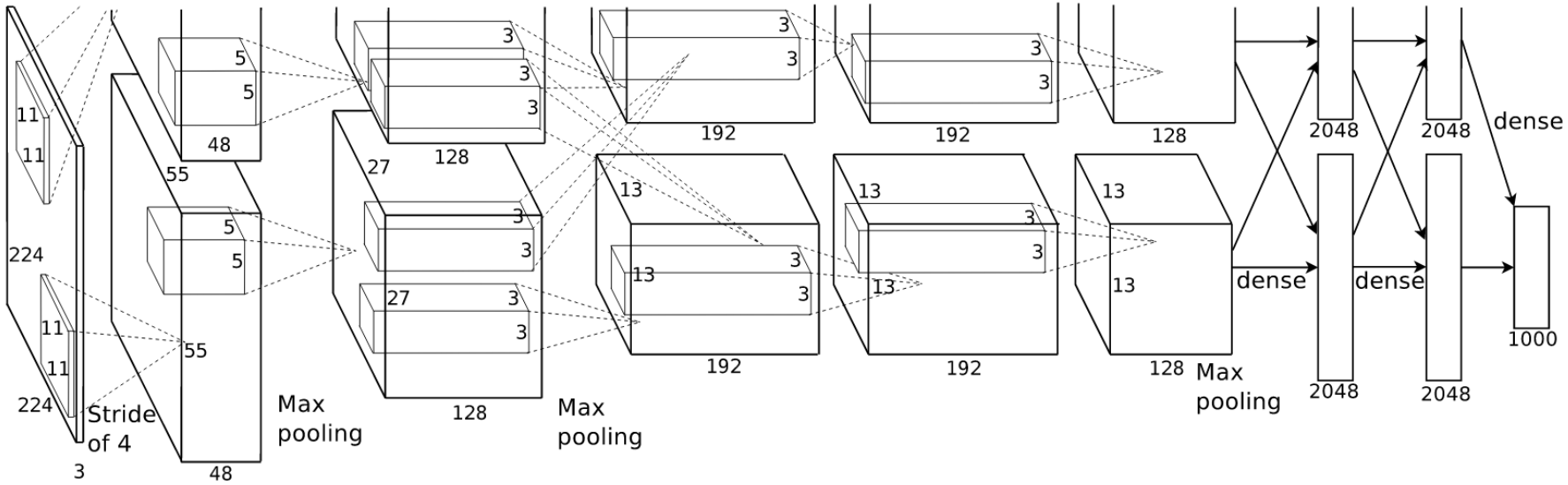
- Input: 227 x 227 x 3 images
- First layer(CONV1): 96 11x1 filters applied at stride 4

→Output size : 55 x 55 x 96

→# of parameters : (11 x 11 x 3) x 96 = 35K

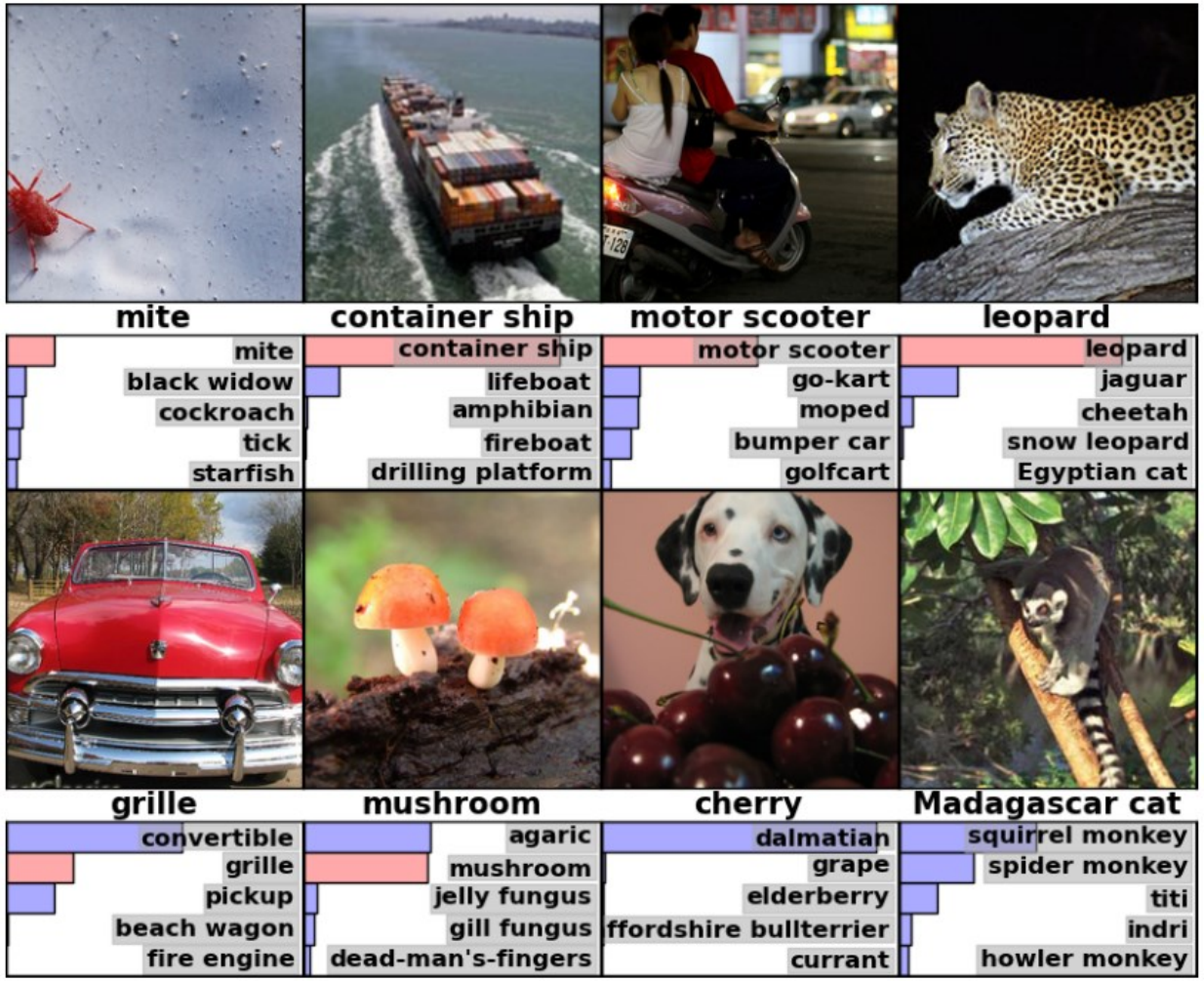
Slide Credit : Stanford CS231n

AlexNet (Krizhevsky, 2012)



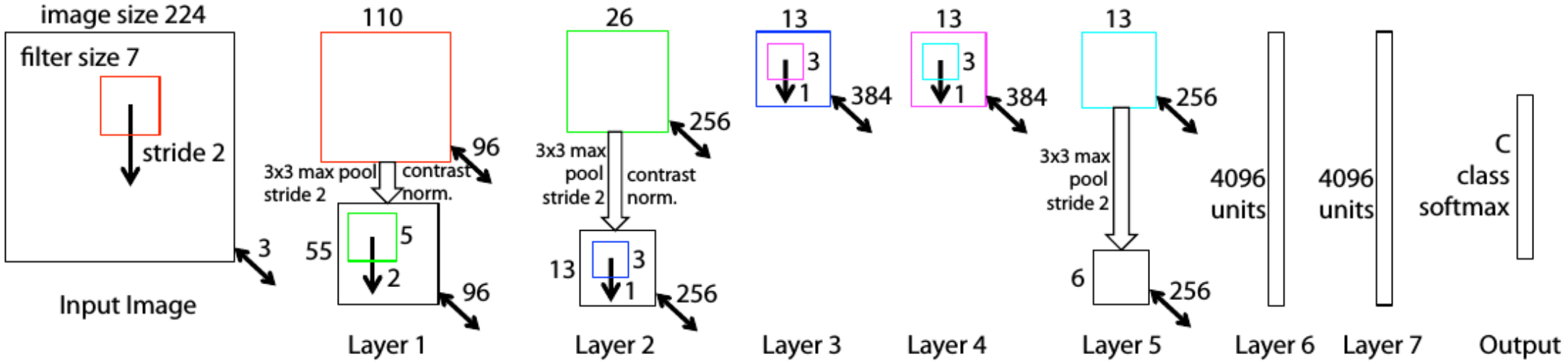
- [Conv1 – Pool1 – Norm1 – Conv2 – Pool2 – Norm2 – Conv3 – Conv4 – Conv5 – Pool 5 – FC6 – FC7 – FC8]
- 7 hidden layers, 650,000 neurons, 60M parameters
- Training for 1 week, using 2-GPUs
  - Trained on GTX 580 GPU with only 3GB of memory. Network spread across 2 GPUs, half the neurons(feature maps) on each GPU

AlexNet Result



"ImageNet Classification with Deep Convolutional Neural Networks "

ZFNet (Zeiler and Fergus, 2013)



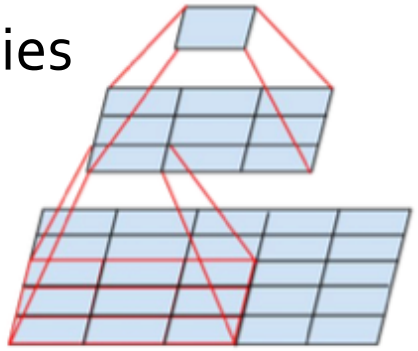
- Similar to AlexNet except:
  - Conv1 – change from (11x11 stride 4) to (7x7 stride 2)
  - Conv3, 4, 5: instead of 384, 384, 256 filters use 512, 1024, 512
  - Top 5 error : 16.4% → 11.7%

VGGNet (Simonyan and Zisserman, 2014)

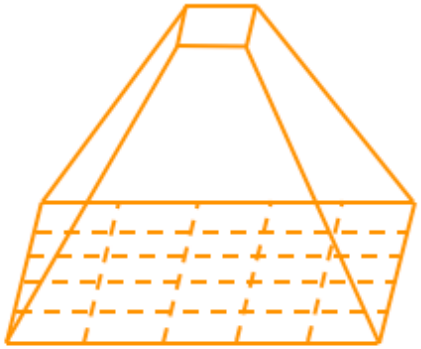
- ILSVRC'14 2nd in classification, 1st in localization
- Small filters, Deeper networks
  - 8 layers(AlexNet) → 16~19 layers(VGG16, VGG19)
  - Only use 3x3 conv stride 1, pad 1 & 2x2 maxpool stride 2
  - 11.7% top 5 error(ZFNet) → 7.3% top 5 error
- Why use only 3x3 filters?
  - Stack of 3x3 conv layers has same effective receptive field as 5x5 or 7x7 conv layer
  - Deeper means more non-linearities
  - Fewer parameters:

    2 x (3 x 3 x C) vs (5 x 5 x C)

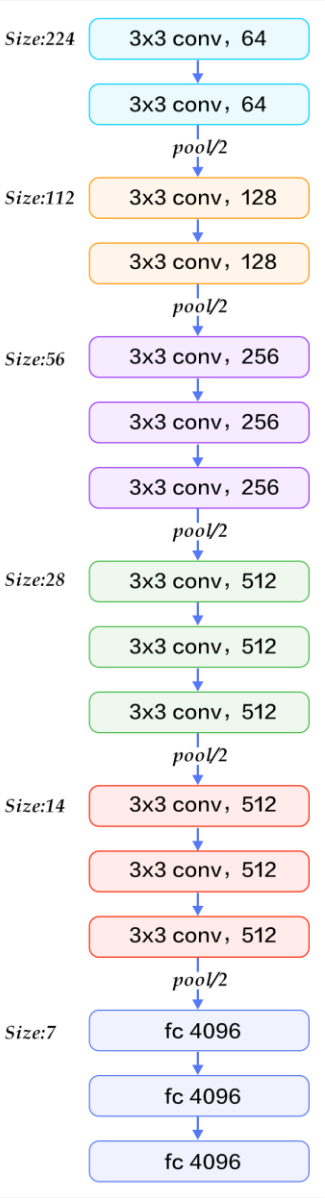
    → regularization effect



two successive 3x3 convolutions



5x5 convolution



VGGNet

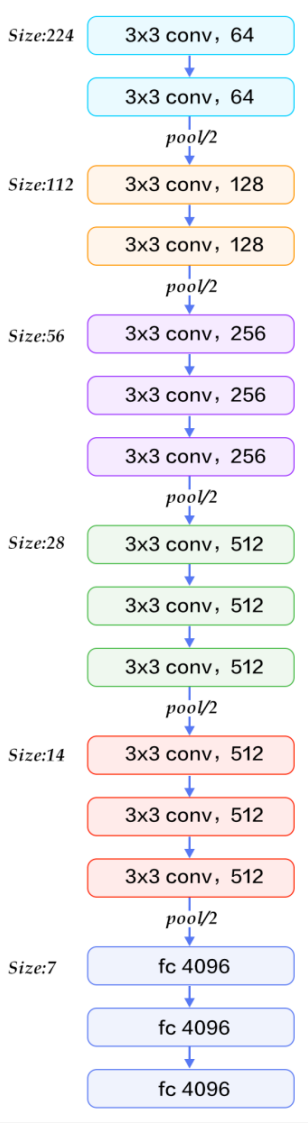
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Memory Usages and Parameters of 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

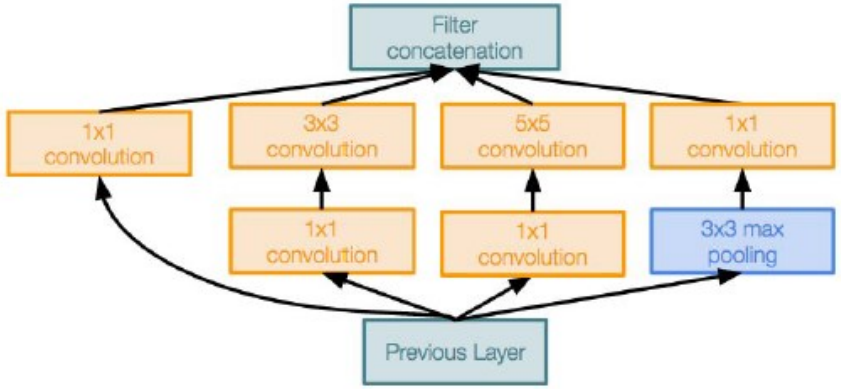
TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters



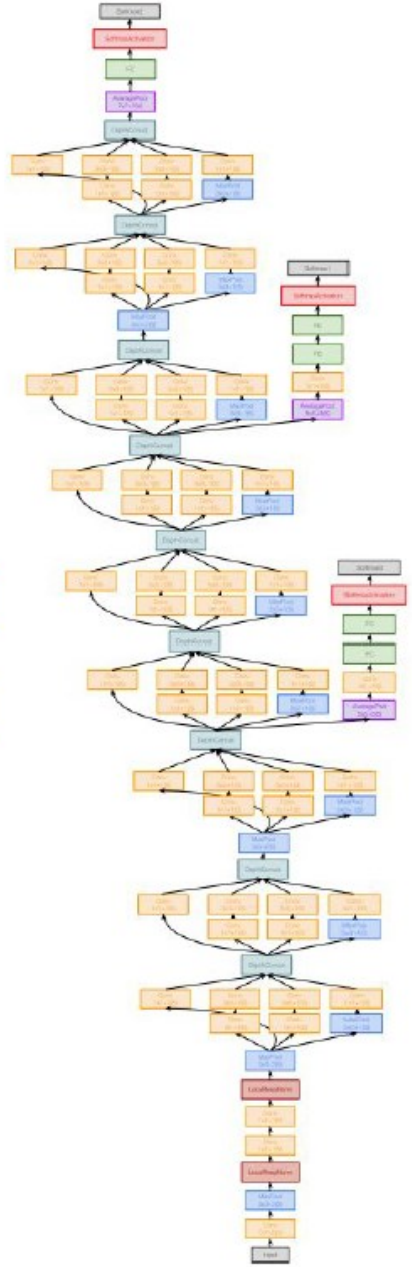
Slide Credit : Stanford CS231n

GoogLeNet (Szegedy, 2014)

- Deeper networks, with computational efficiency
  - 22 layers
  - Efficient "Inception" module
  - Global average pooling
  - Only 5 million parameters
  - ILSVRC'14 classification winner
    - ➢ 6.7% top 5 error



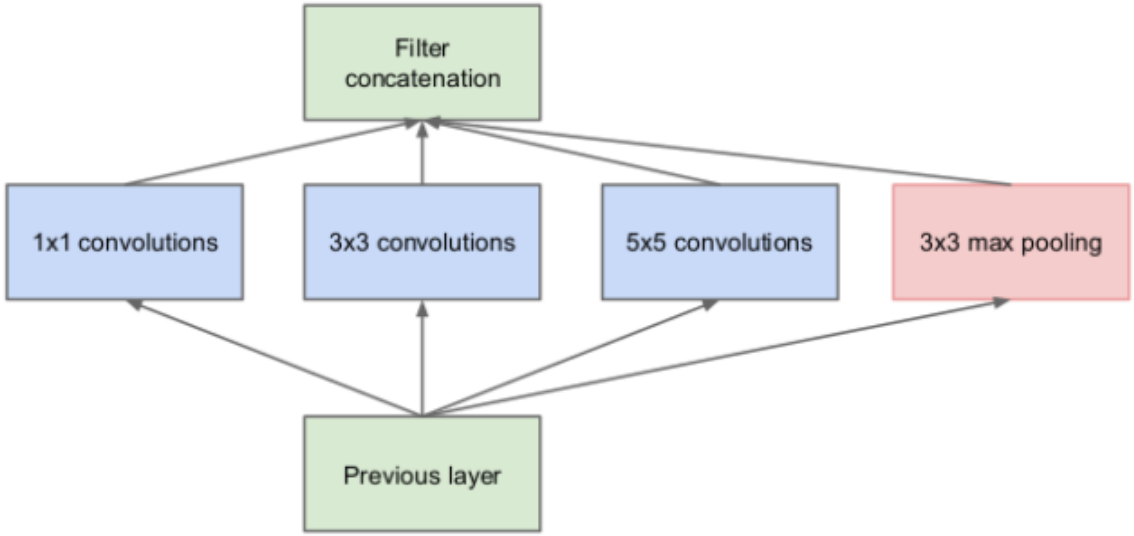
Inception module



Slide Credit : Stanford CS231n

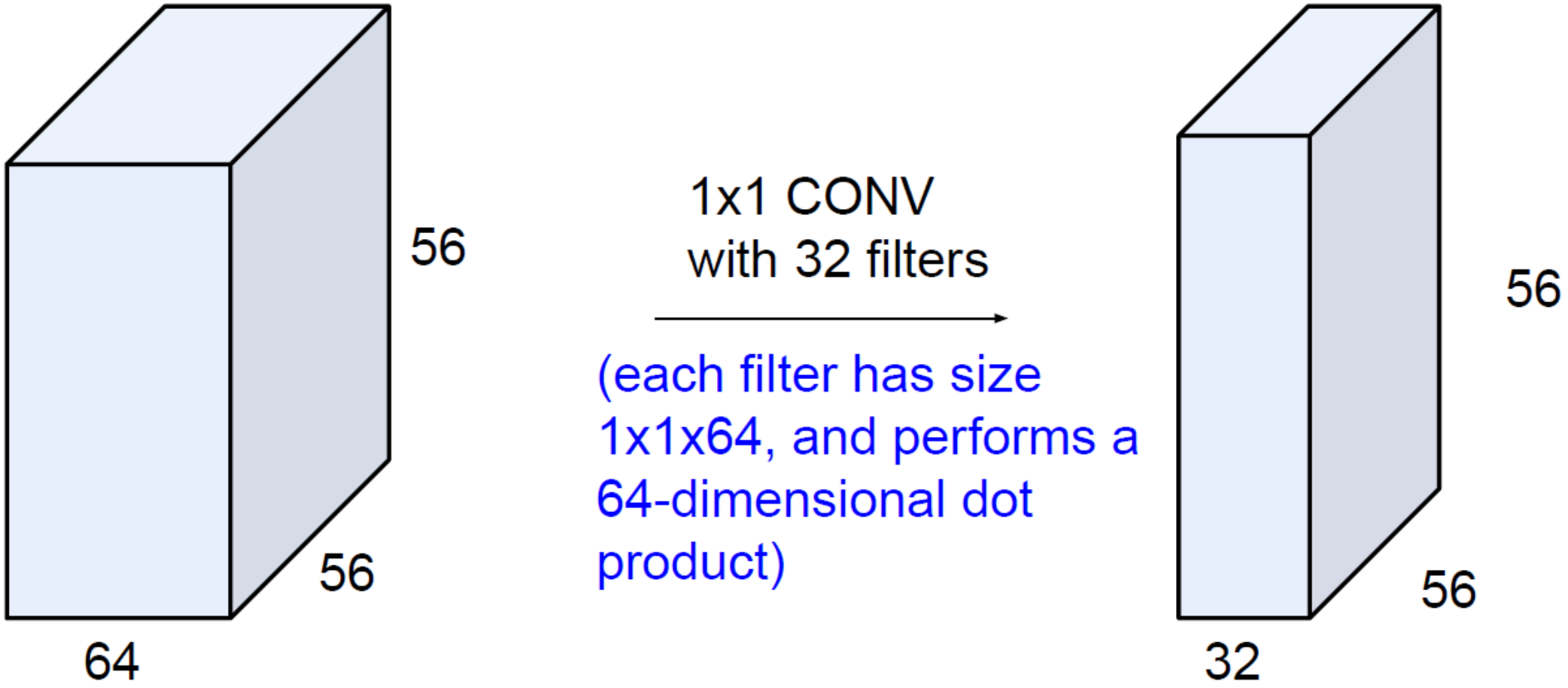
Inception Module

- Naïve Inception module
  - Apply parallel filter operations on the input from previous layer
    - ➢ Multiple receptive field sizes for convolution
    - ➢ Pooling operation(3x3)
  - Concatenate all filter outputs together channel-wise
  - What is the problem with this?
    - ➢ Very expensive compute
    - ➢ Depth after concatenation can grow and grow at every layer!
    - → Use Bottleneck layers



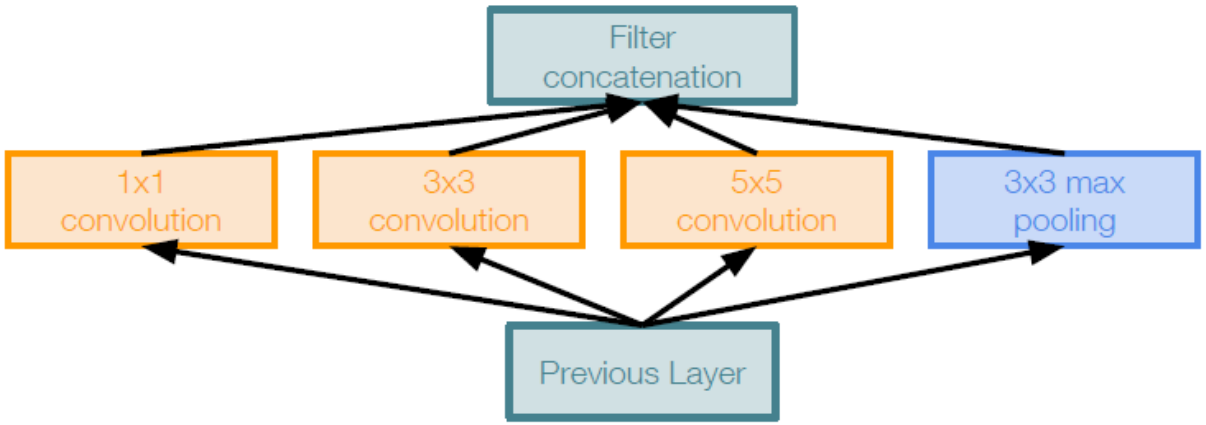
Slide Credit : Stanford CS231n

1x1 convolutions



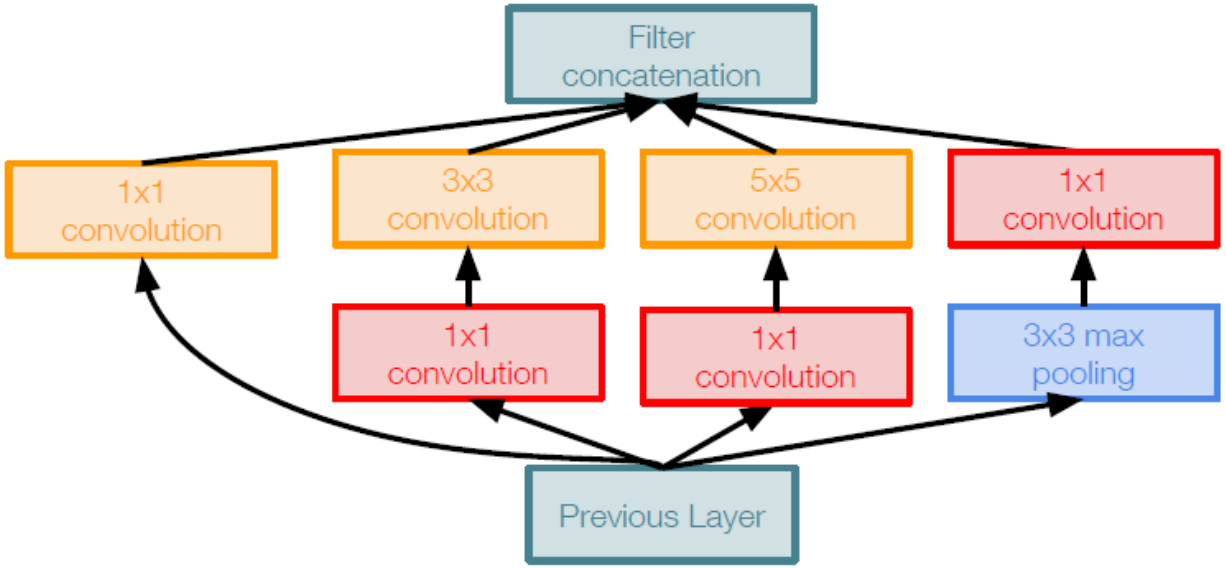
Slide Credit : Stanford CS231n

Inception Module



Naive Inception module

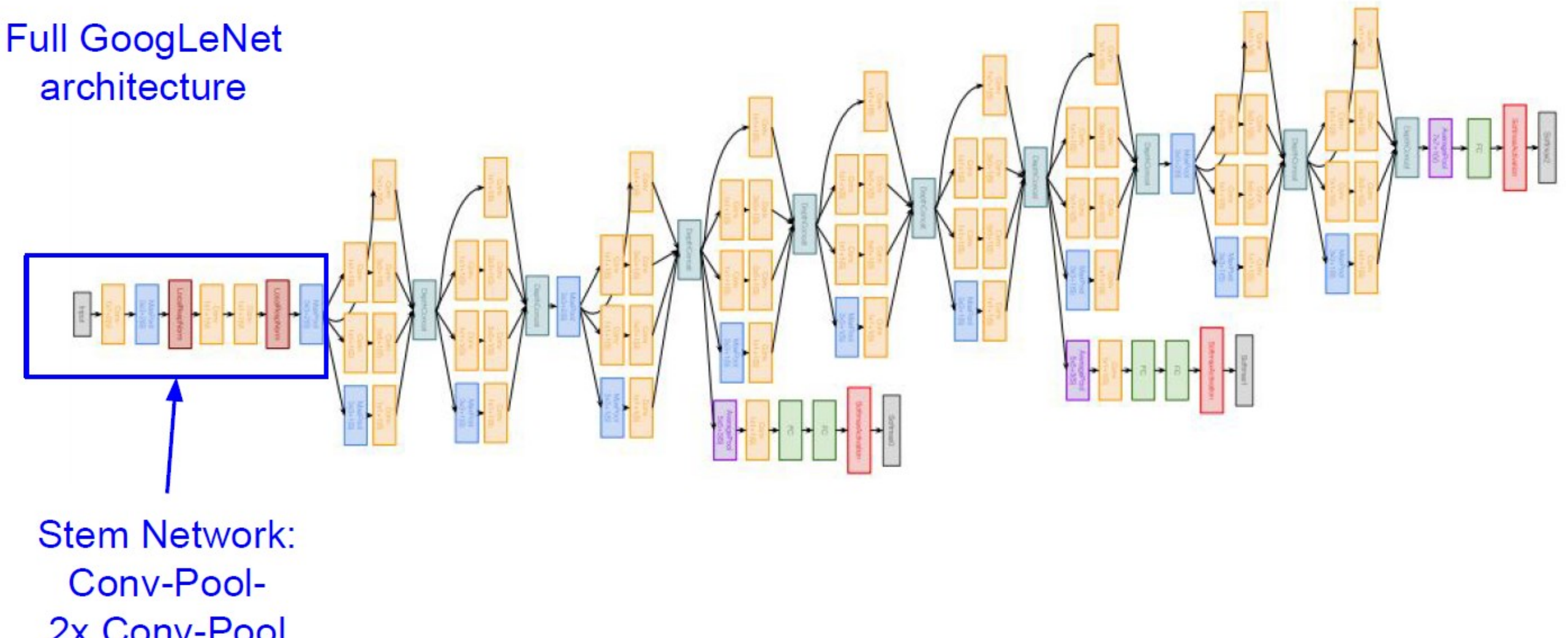
1x1 conv "bottleneck" layers



Inception module with dimension reduction

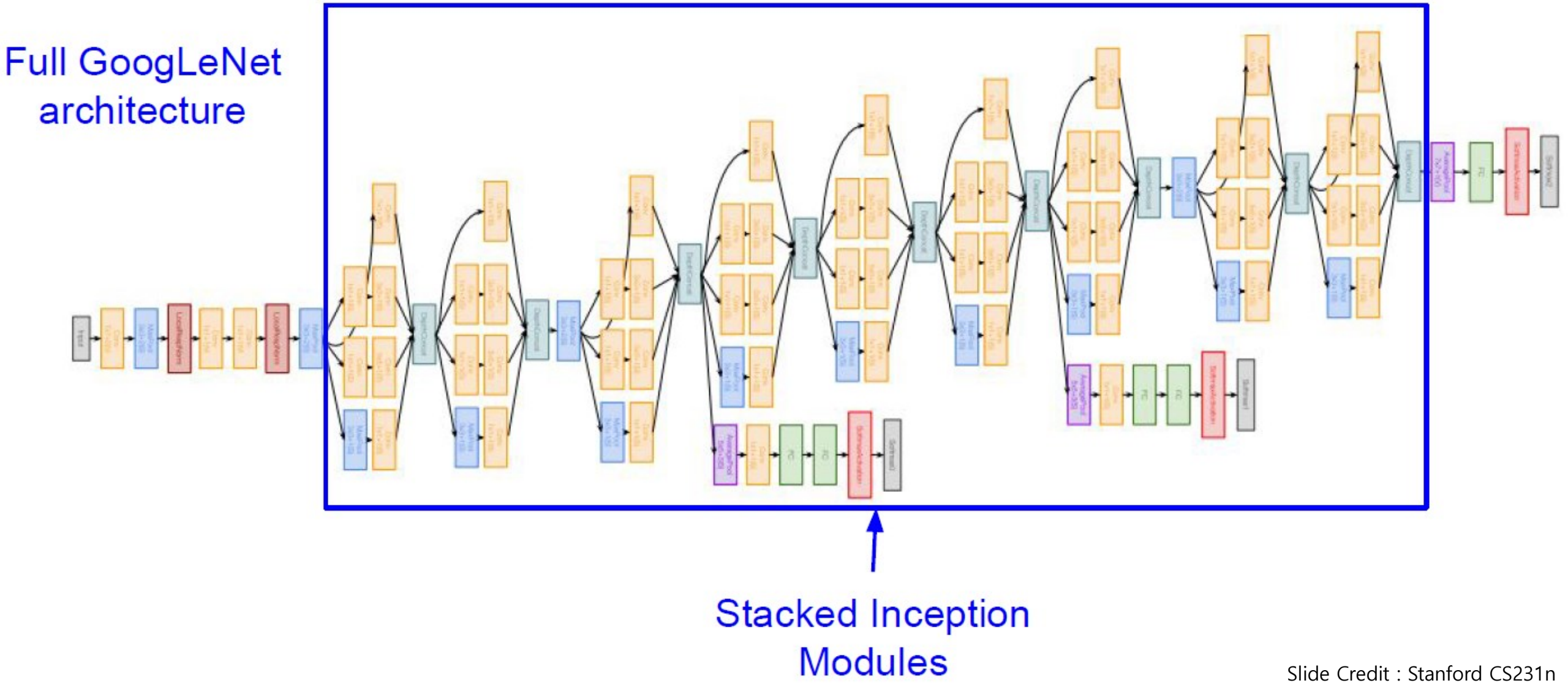
Slide Credit : Stanford CS231n

GoogLeNet



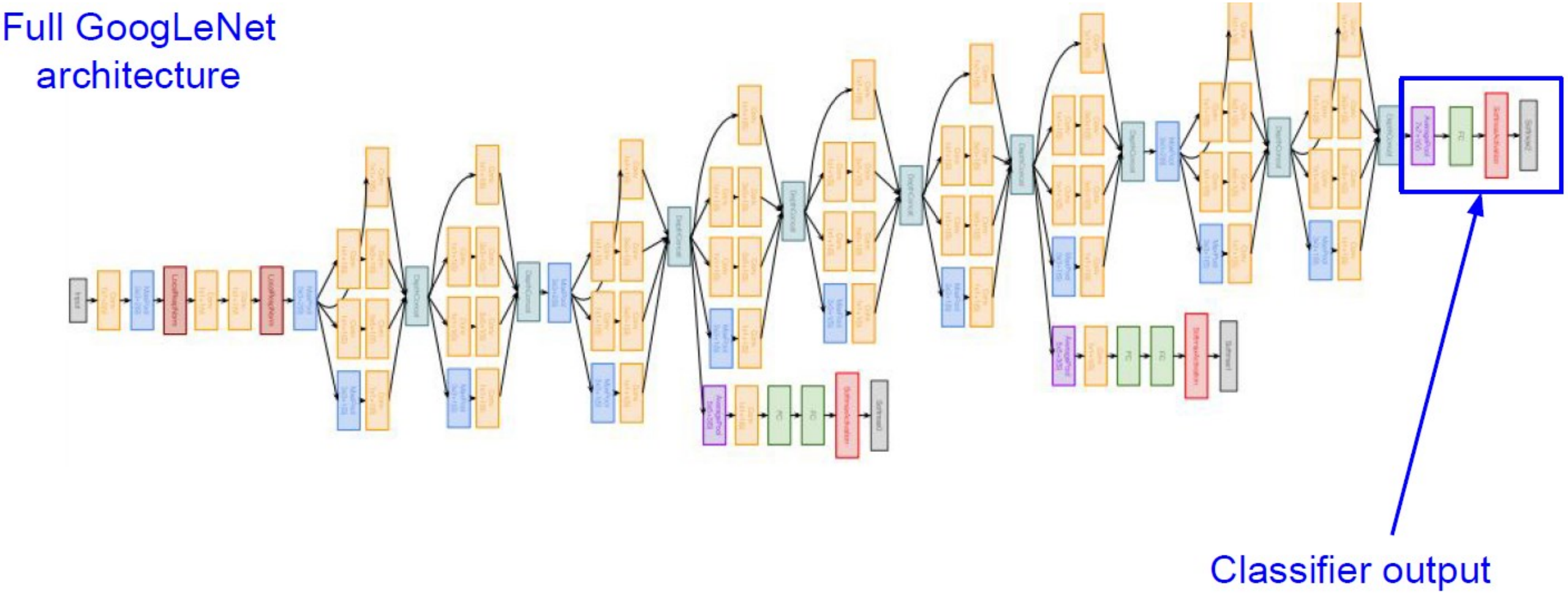
Slide Credit : Stanford CS231n

GoogLeNet



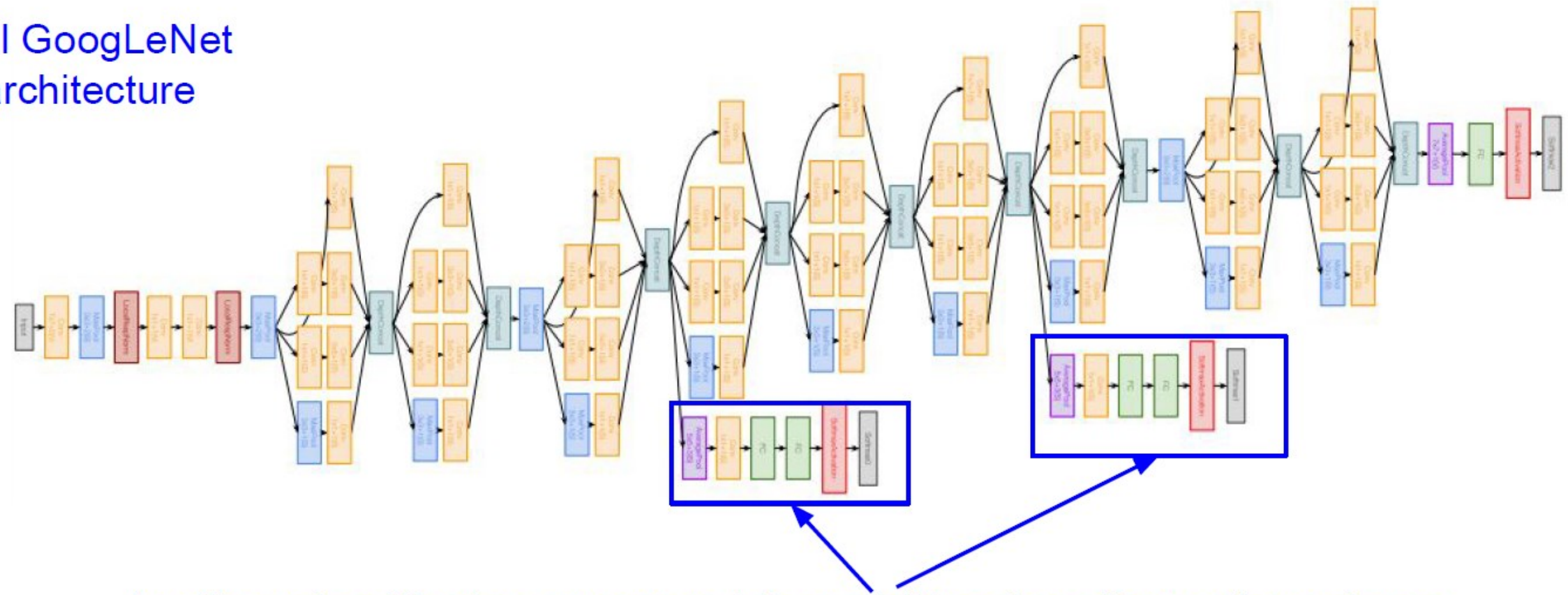
Slide Credit : Stanford CS231n

GoogLeNet



Slide Credit : Stanford CS231n

GoogLeNet



Slide Credit : Stanford CS231n

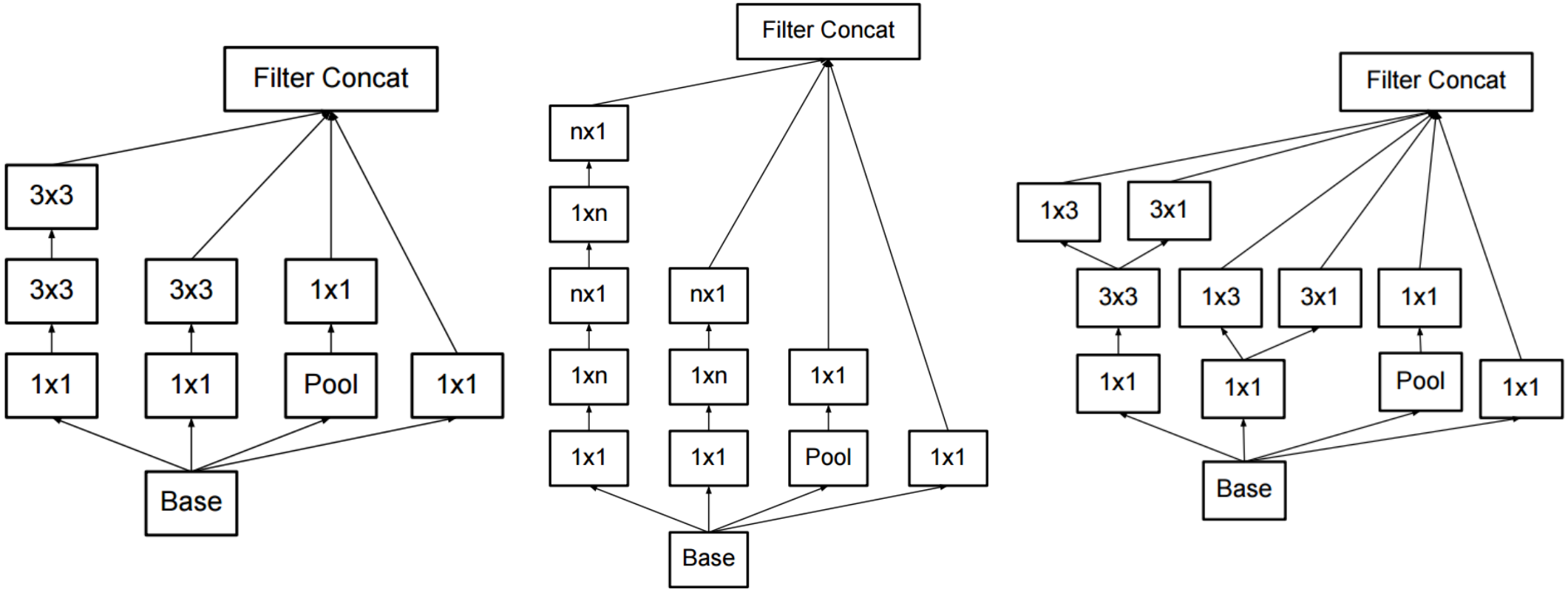
GoogLeNet

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

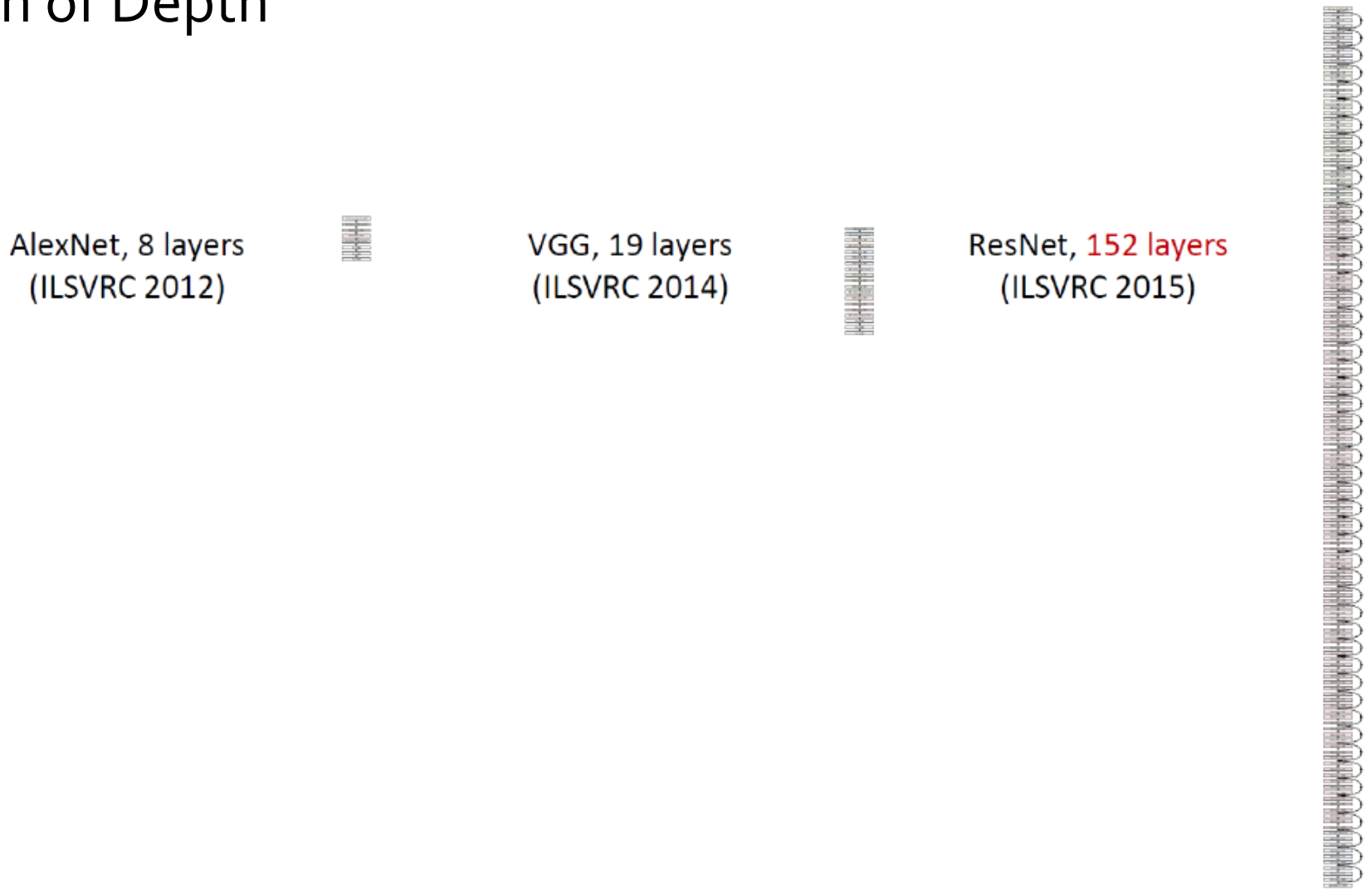
Inception-v3

- Factorization of filters



ResNet (He, 2015)

- Revolution of Depth



ResNet

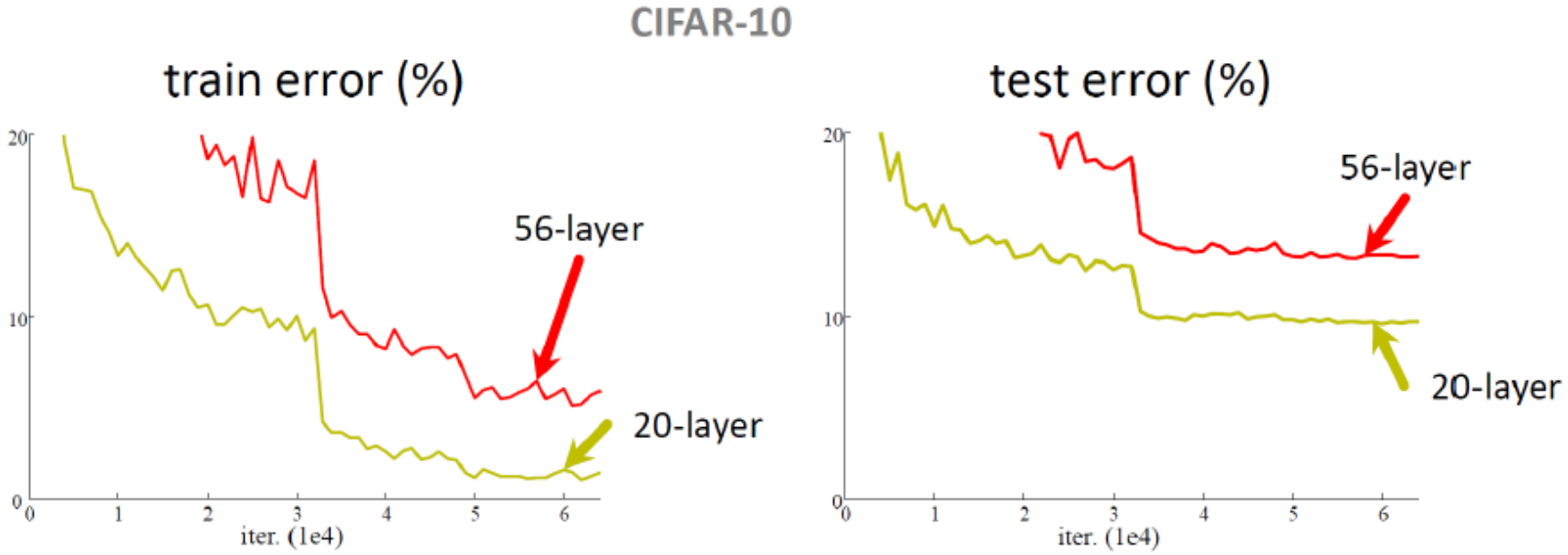
- Swept 1st place in all ILSVRC and COCO 2015 competitions

- ILSVRC'15 classification winner(3.6% top 5 error) – better than "human performance" (Russakovsky 2014)

Slide Credit : Stanford CS231n

ResNet

- What happens when we continue stacking deeper layers on a "plain" convolutional neural network?

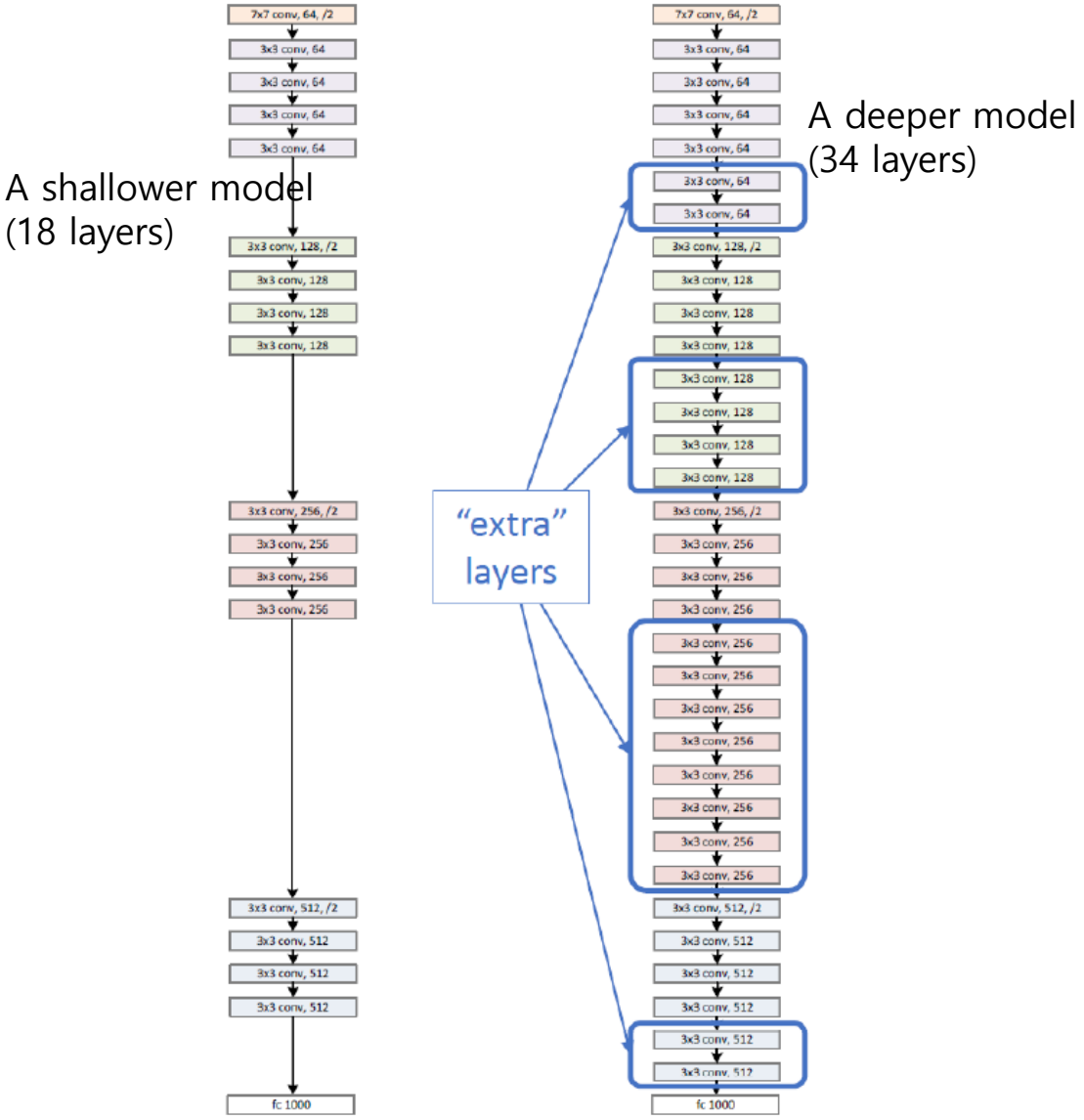


- 56-layer model performs worse on both training and test error
  - The deeper model performs worse, but it's not caused by overfitting!

Slide Credit : Stanford CS231n

ResNet

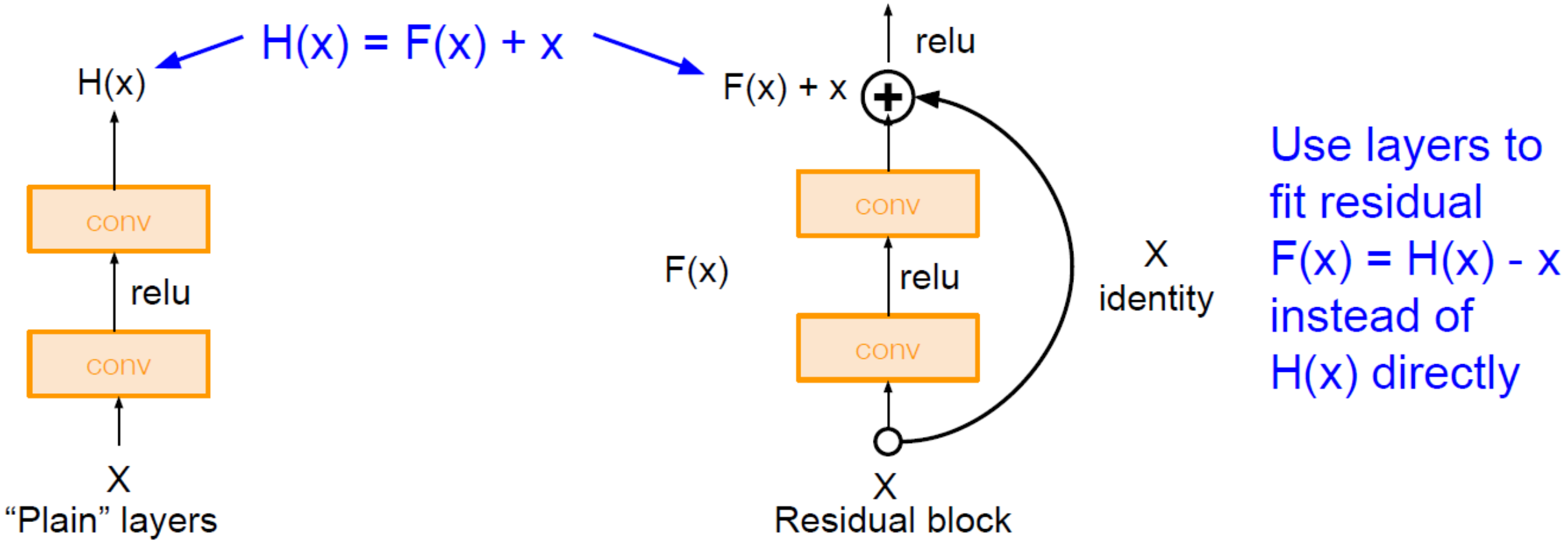
- Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize
  - The deeper model should be able to perform at least as well as the shallower model
  - A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping



Slide Credit : Stanford CS231n

ResNet

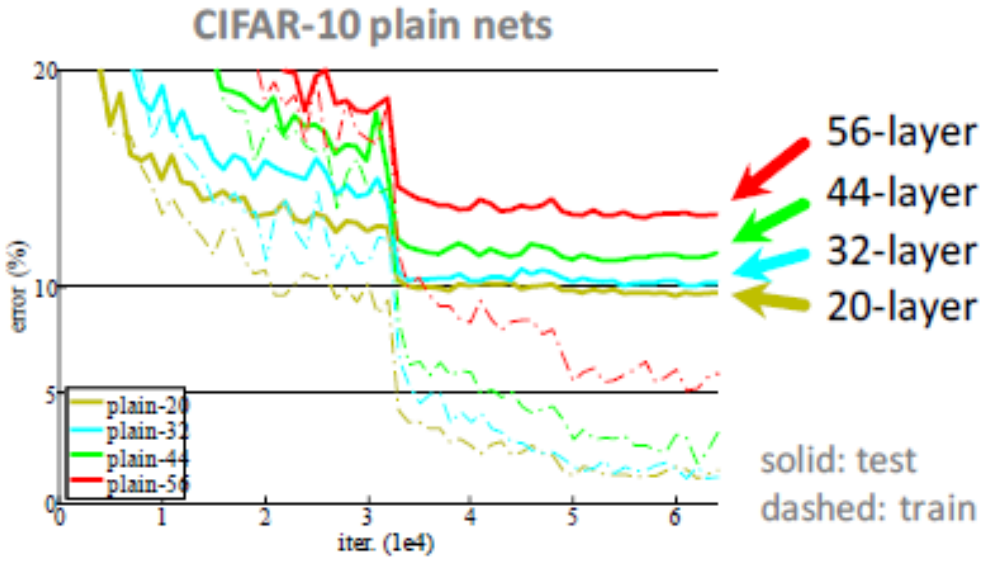
- Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

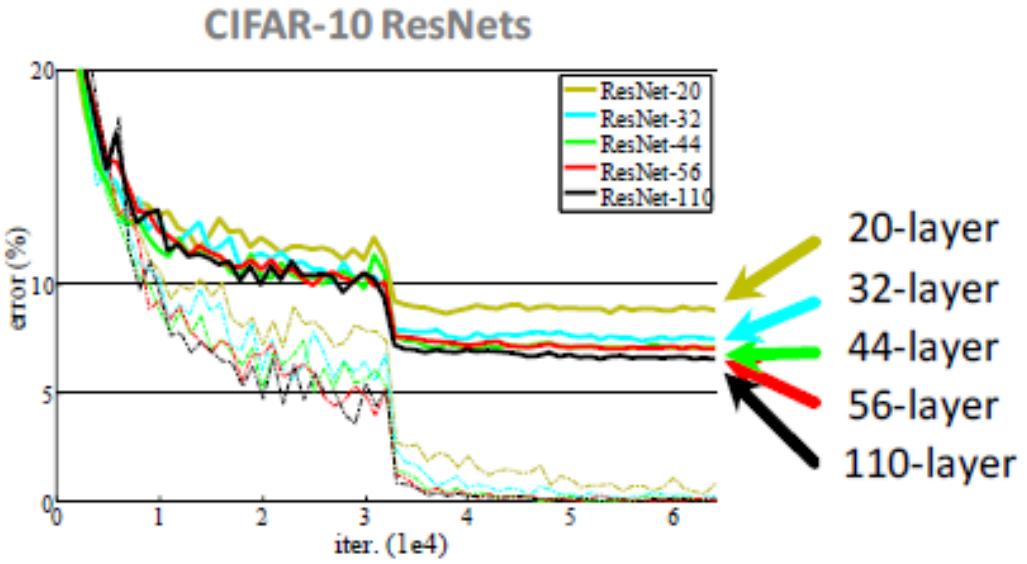


Slide Credit : Stanford CS231n

ResNet – Experimental Results

CIFAR-10 experiments

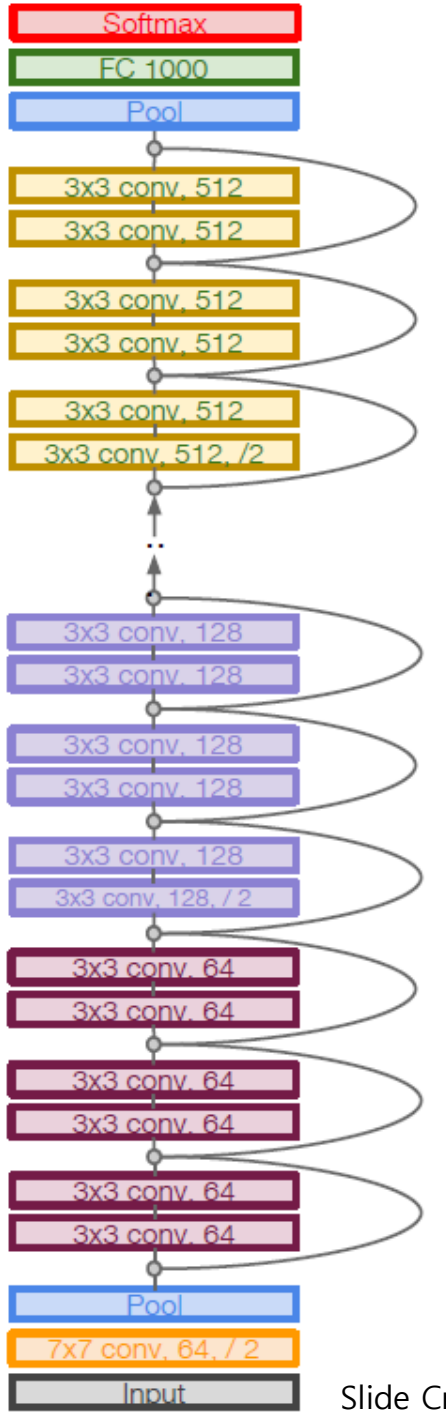




- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

ResNet

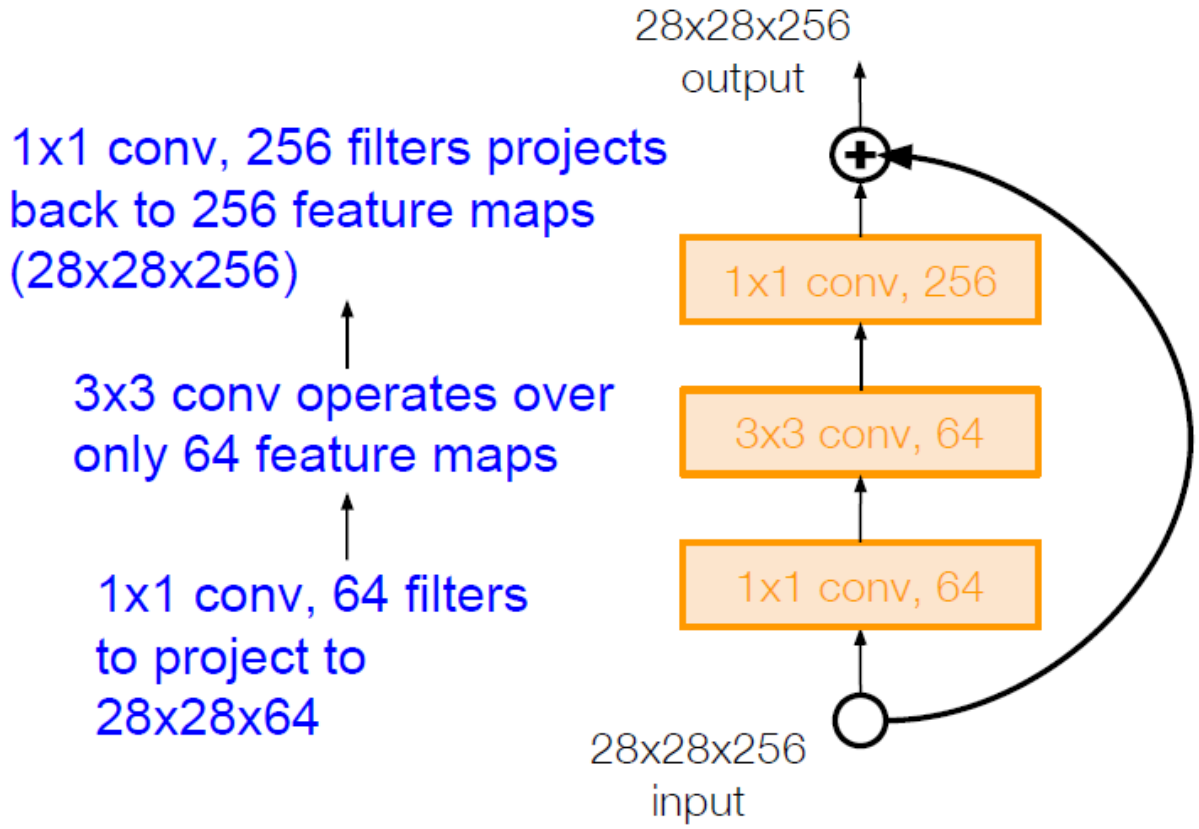
- Full ResNet architecture
  - Stack residual blocks
  - Every residual block has two 3x3 conv layers
  - Periodically double # of filters and downsample spatially using stride 2
  - Additional conv layer at the beginning
  - No FC layers at the end(only FC 100 to output classes)



Slide Credit : Stanford CS231n

Bottleneck Architecture

- For deeper networks(ResNet-50+), use "bottleneck" layer to improve efficiency(similar to GoogLeNet)



Slide Credit : Stanford CS231n

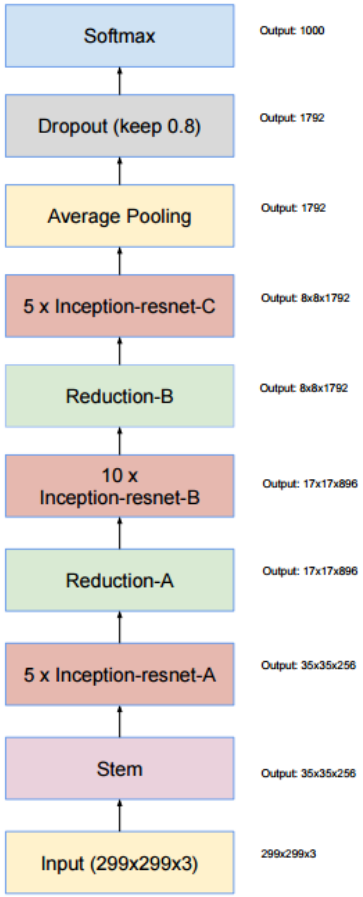
ResNet

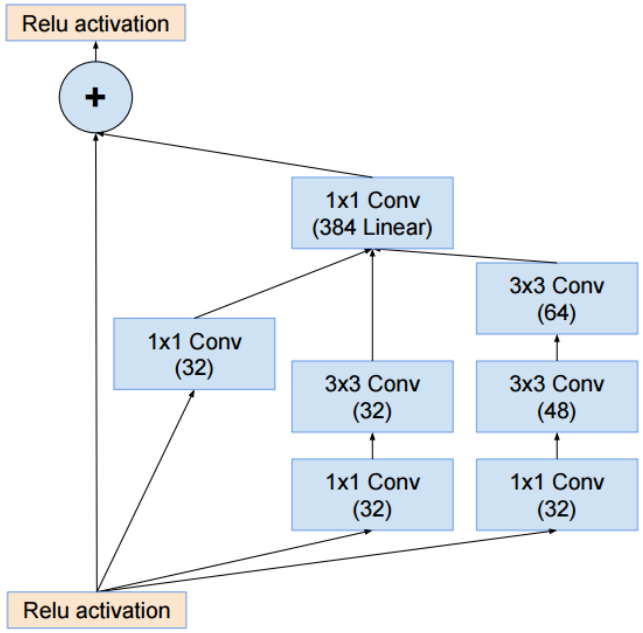
- Batch Normalization after every Conv layer
- Xavier/2 initialization from He et al.
- SGD + Momentum(0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

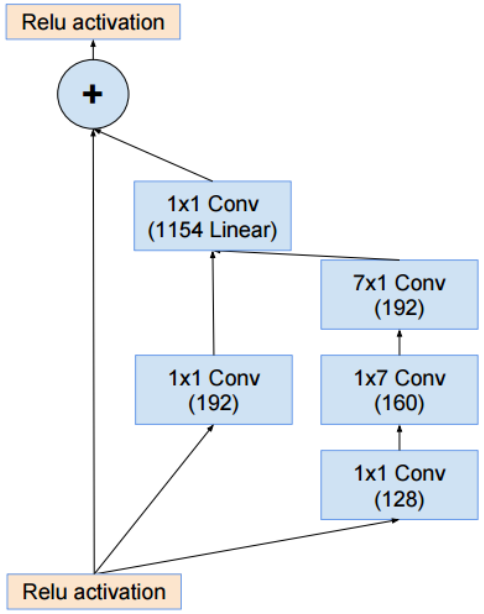
Slide Credit : Stanford CS231n

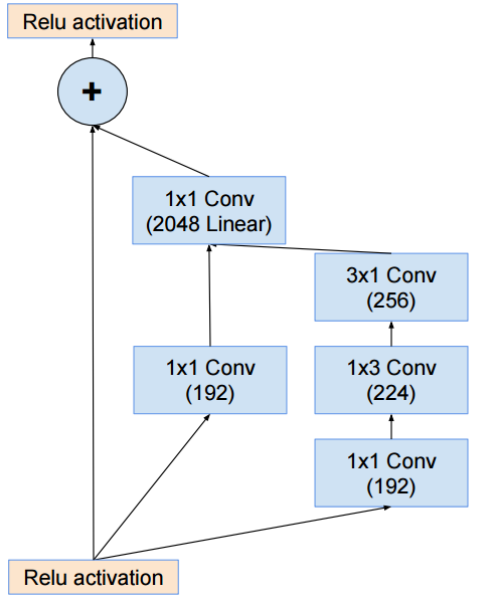
Inception-ResNet

- Inception + ResNet



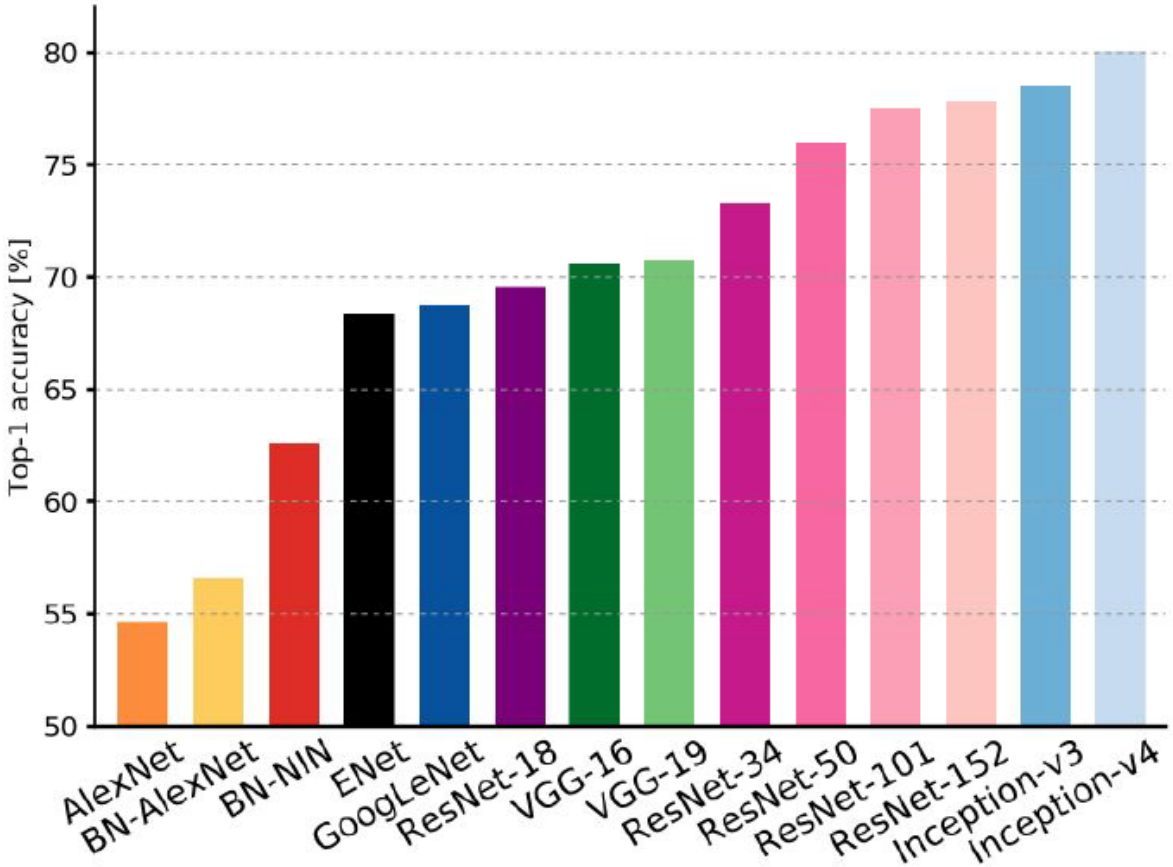


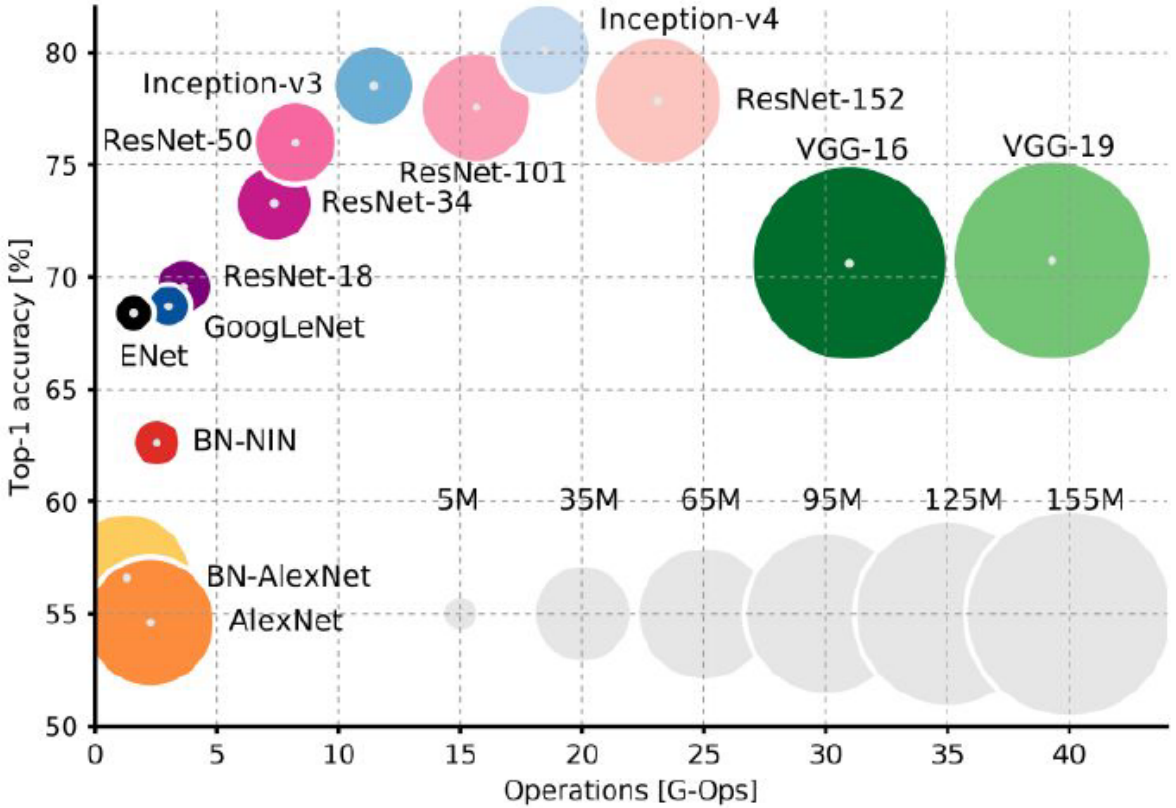




"Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning"

Comparing Complexity





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Slide Credit : Stanford CS231n

ResNeXt

Aggregated Residual Transformations for Deep Neural Networks

Saining Xie[1] Ross Girshick[2] Piotr Dollár[2] Zhuowen Tu[1] Kaiming He[2]
[1]UC San Diego [2]Facebook AI Research
{s9xie,ztu}@ucsd.edu {rbg,pdollar,kaiminghe}@fb.com

Abstract

We present a simple, highly modularized network architecture for image classification. Our network is constructed by repeating a building block that aggregates a set of transformations with the same topology. Our simple design results in a homogeneous, multi-branch architecture that has only a few hyper-parameters to set. This strategy exposes a new dimension, which we call "cardinality" (the size of the set of transformations), as an essential factor in addition to the dimensions of depth and width. On the ImageNet-1K dataset, we empirically show that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy. Moreover, in-

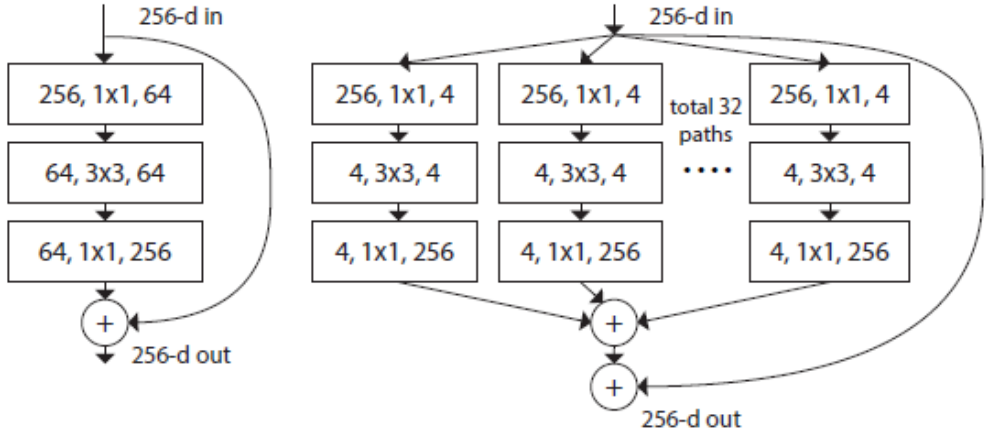


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

ImageNet 2016 Results

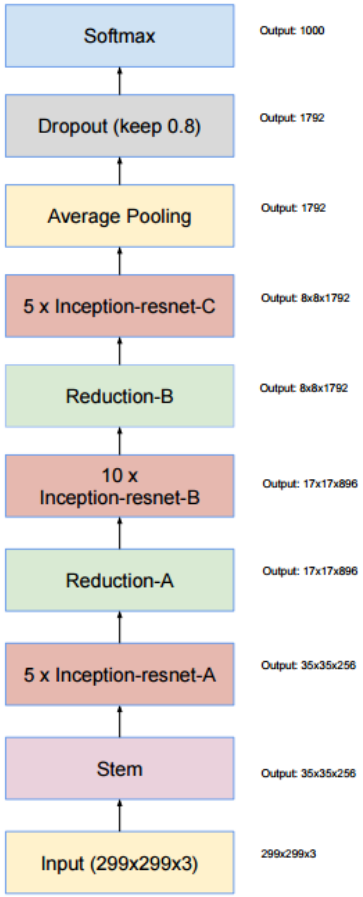
Team name	Entry description	**Classification error**	Localization error
Trimps-Soushen	Ensemble 2	0.02991	0.077668
Trimps-Soushen	Ensemble 3	0.02991	0.077087
Trimps-Soushen	Ensemble 4	0.02991	0.077429
ResNeXt	Ensemble C, weighted average, tuned on val. [No bounding box results]	0.03031	0.737308
CU-DeepLink	GrandUnion + Fused-scale EnsembleNet	0.03042	0.098892
CU-DeepLink	GrandUnion + Multi-scale EnsembleNet	0.03046	0.099006
CU-DeepLink	GrandUnion + Basic Ensemble	0.03049	0.098954
ResNeXt	Ensemble B, weighted average, tuned on val. [No bounding box results]	0.03092	0.737484
CU-DeepLink	GrandUnion + Class-reweighted Ensemble	0.03096	0.099369
CU-DeepLink	GrandUnion + Class-reweighted Ensemble with Per-instance Normalization	0.03103	0.099349
ResNeXt	Ensemble C, weighted average. [No bounding box results]	0.03124	0.737526
Trimps-Soushen	Ensemble 1	0.03144	0.079068
ResNeXt	Ensemble A, simple average. [No bounding box results]	0.0315	0.737505
SamExynos	3 model only for classification	0.03171	0.236561
ResNeXt	Ensemble B, weighted average. [No bounding box results]	0.03203	0.737681
KAISTNIA_ETRI	Ensembles A	0.03256	0.102015
KAISTNIA_ETRI	Ensembles C	0.03256	0.102056
KAISTNIA_ETRI	Ensembles B	0.03256	0.100676

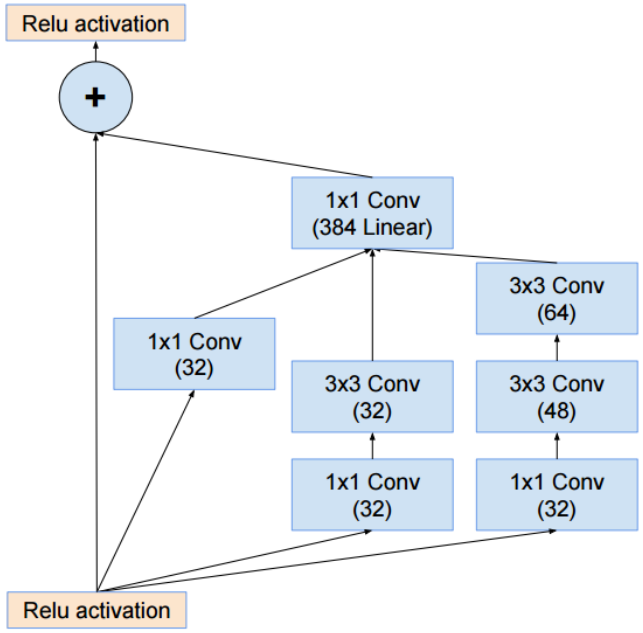
Growing Number of Hyper-parameters

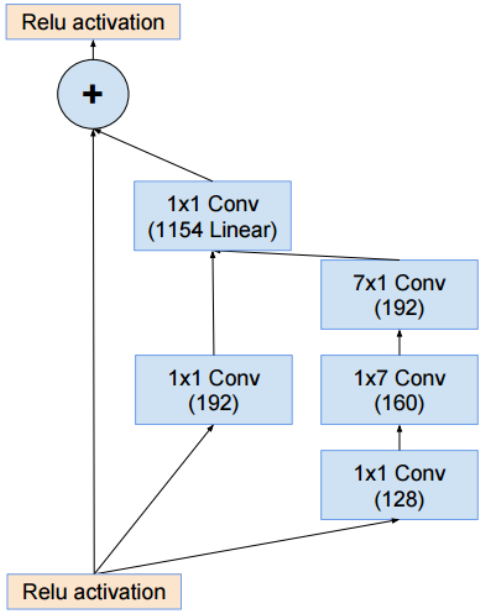
- VGGNet exhibit a simple yet effective strategy of constructing very deep network – stacking building blocks of the same shape
- ResNet inherit this strategy with stacking modules of the same topology
- Unlike VGGNet, the family of Inception models have demonstrated that carefully designed topologies are able to achieve compelling accuracy
  - Important common property is split-transform-merge strategy
  - Split – 1x1 conv, transform – 3x3, 5x5 conv, merge - concatenation

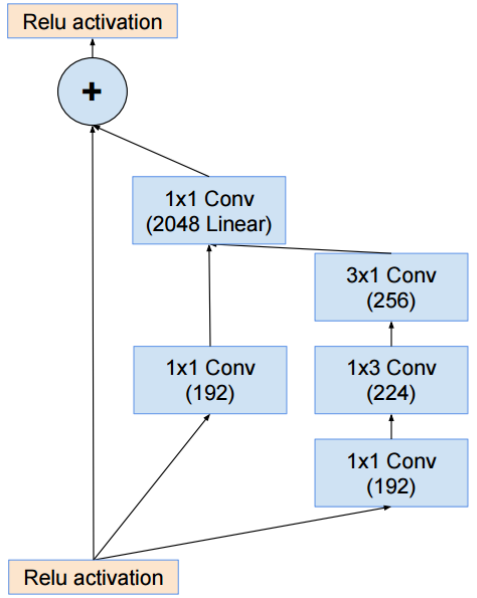
Inception Learns ResNet

- Inception + ResNet





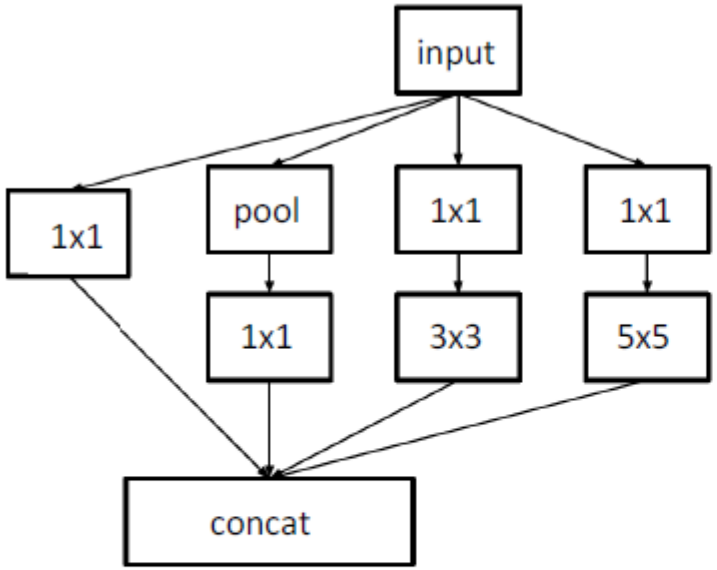




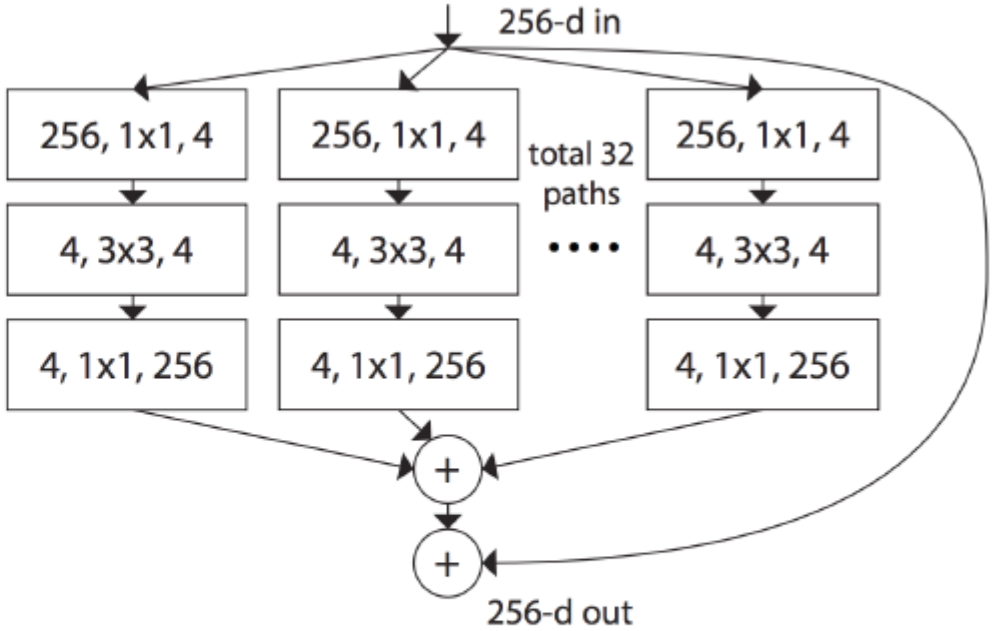
"Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning"

ResNet Learns Inception??

- Multi-branch + ResNet

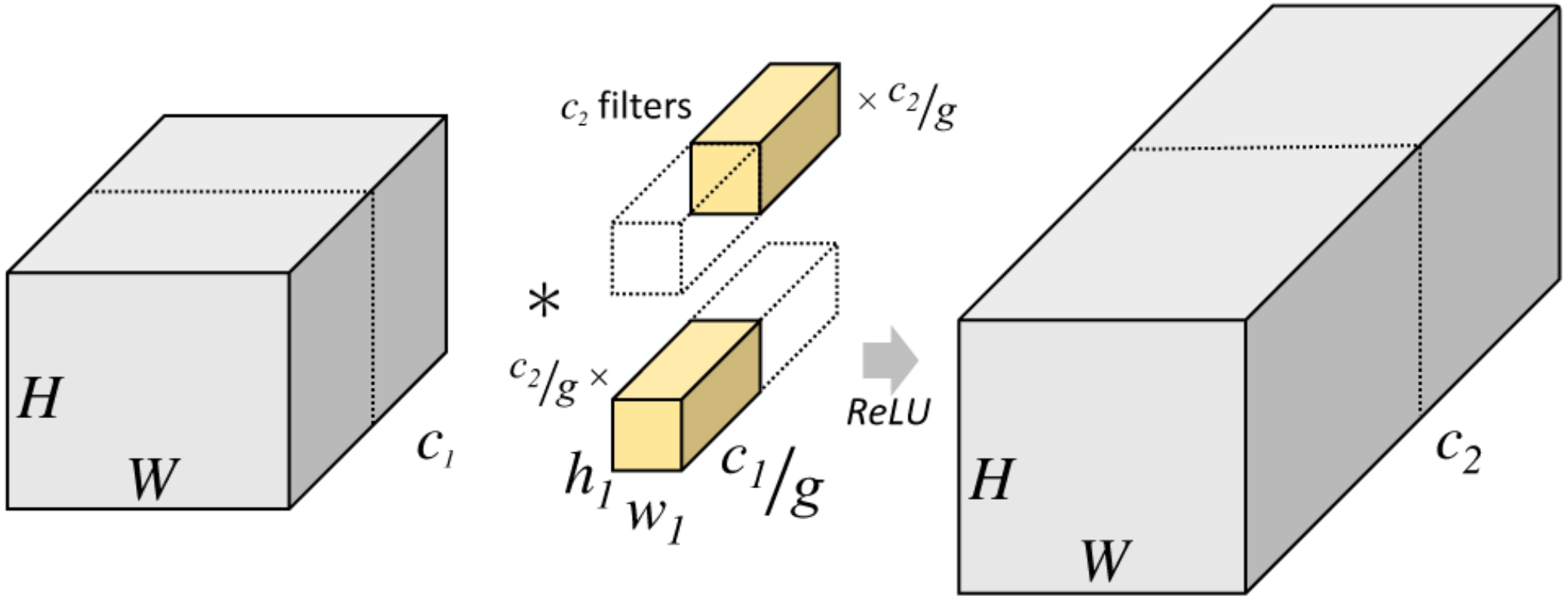


Inception:
heterogeneous multi-branch



ResNeXt:
uniform multi-branch

Group Convolution

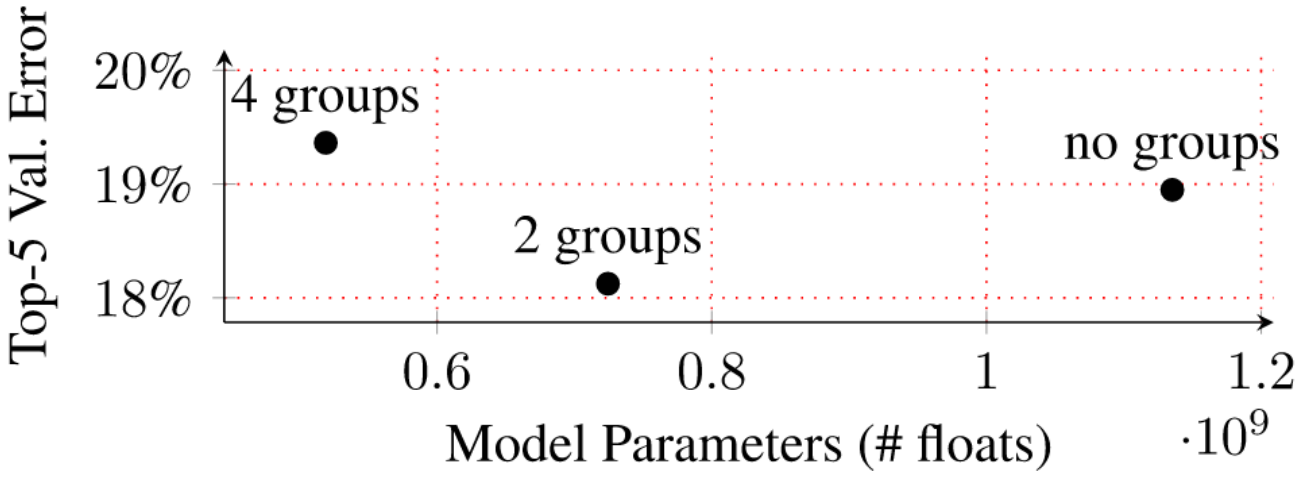


A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convolutional layer is now exactly half the depth, i.e. half the parameters and half the compute as the original filter.

Figure from https://blog.yani.io/filter-group-tutorial/

Grouped Convolution of AlexNet

- AlexNet's primary motivation was to allow the training of the network over two Nvidia GTX580 GPUs with 1.5GB of memory each
- AlexNet without filter groups is not only less efficient(both in parameters and compute), but also slightly less accurate!



AlexNet trained with varying numbers of filter groups, from 1 (i.e. no filter groups), to 4. When trained with 2 filter groups, AlexNet is more efficient and yet achieves the same if not lower validation error.

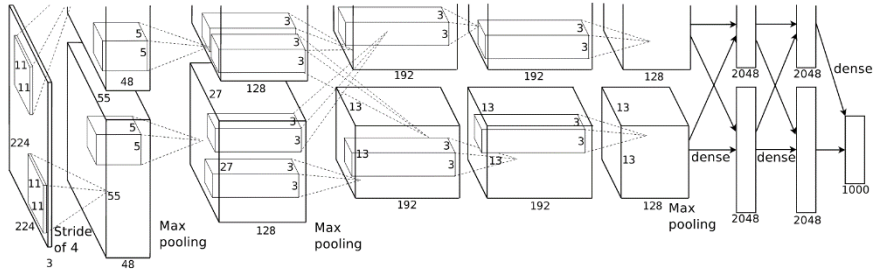
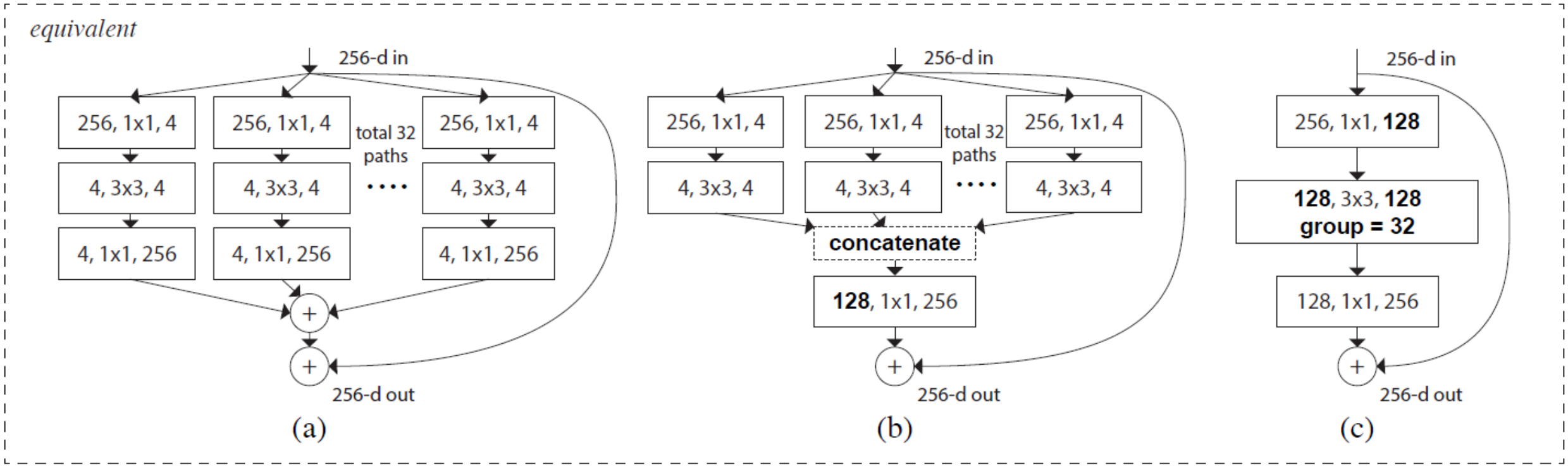


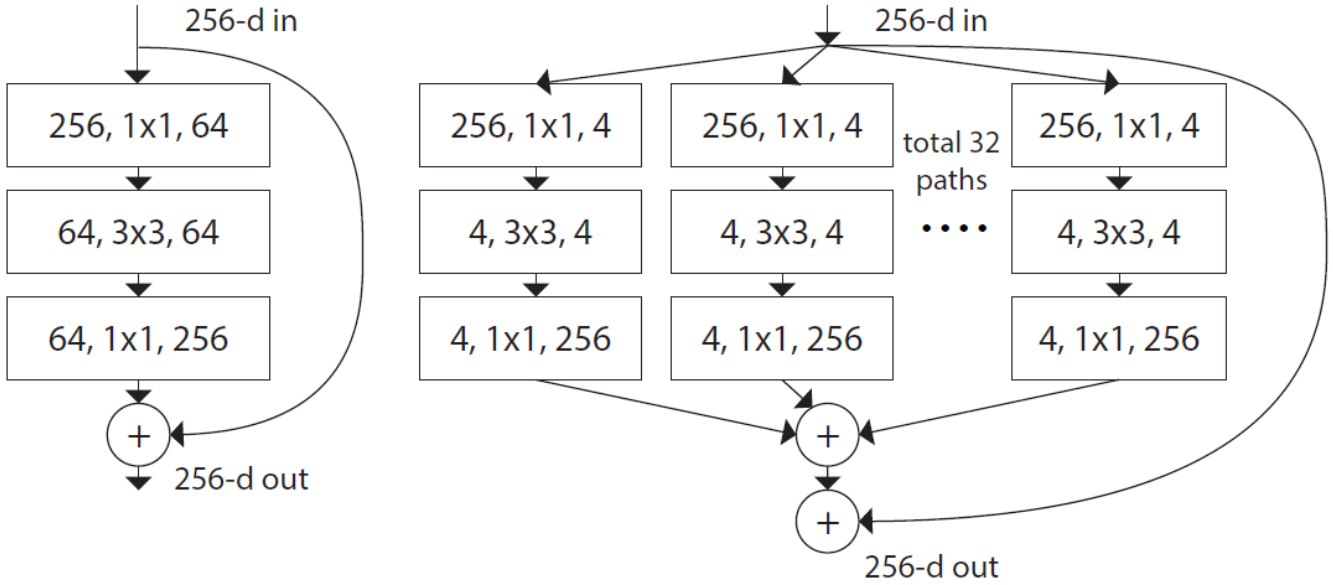
Figure from https://blog.yani.io/filter-group-tutorial/

ResNeXt

- Concatenation and Addition are interchangeable
- Uniform multi-branching can be done by group-conv



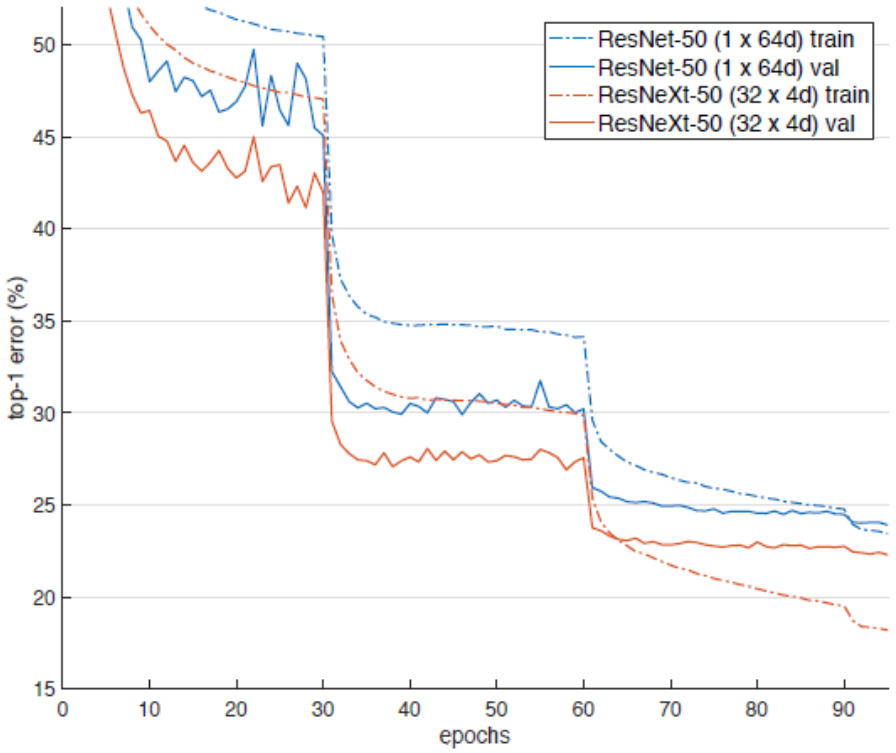
Model Capacity (# of parameters)

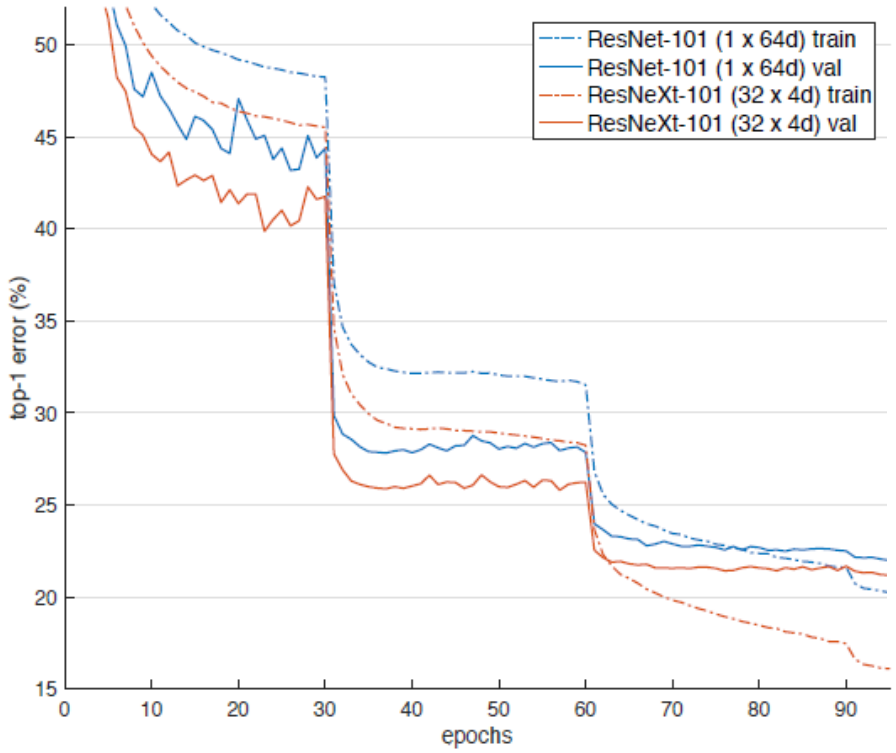


- Original ResNet(left) : 256x64 + 3x3x64x64 + 64x256 = 70k
- ResNeXt(right – with bottleneck width d and cardinality C) :

  C x (256xd + 3x3xdxd + dx256) = 70k, when C =32 and d=4

Results – Cardinality vs Width





	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Squeeze-and-Excitation Networks

Squeeze-and-Excitation Networks

Jie Hu[0000−0002−5150−1003] Li Shen[0000−0002−2283−4976] Samuel Albanie[0000−0001−9736−5134]
Gang Sun[0000−0001−6913−6799] Enhua Wu[0000−0002−2174−1428]

Abstract—The central building block of convolutional neural networks (CNNs) is the convolution operator, which enables networks to construct informative features by fusing both spatial and channel-wise information within local receptive fields at each layer. A broad range of prior research has investigated the spatial component of this relationship, seeking to strengthen the representational power of a CNN by enhancing the quality of spatial encodings throughout its feature hierarchy. In this work, we focus instead on the channel relationship and propose a novel architectural unit, which we term the "Squeeze-and-Excitation" (SE) block, that adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels. We show that these blocks can be stacked together to form SENet architectures that generalise extremely effectively across different datasets. We further demonstrate that SE blocks bring significant improvements in performance for existing state-of-the-art CNNs at minimal additional computational cost. Squeeze-and-Excitation Networks formed the foundation of our ILSVRC 2017 classification submission which won first place and reduced the top-5 error to 2.251%, surpassing the winning entry of 2016 by a relative improvement of ∼25%. Models and code are available at https://github.com/hujie-frank/SENet.

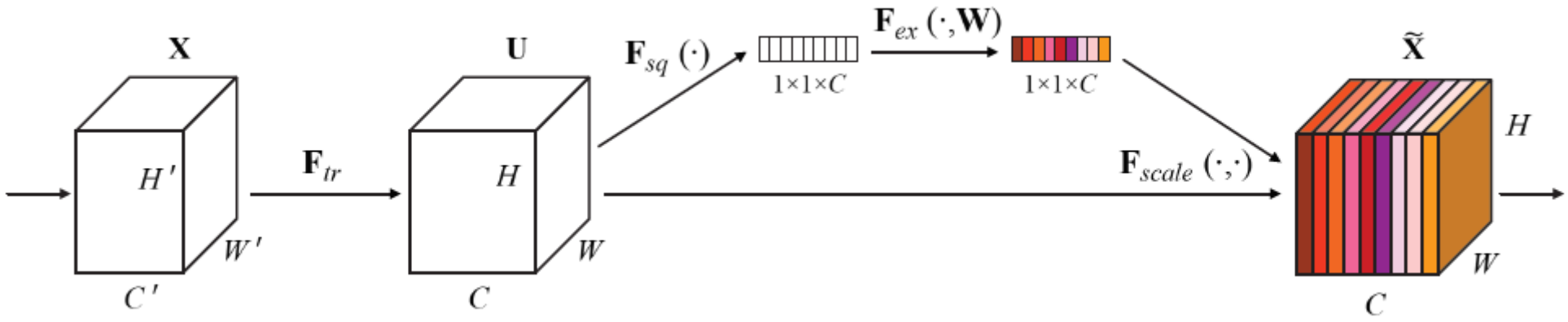
Index Terms—Squeeze-and-Excitation, Image classification, Convolutional Neural Network.

Squeeze-and-Excitation Networks

Team name	Entry description	**Classification error**	Localization error
WMW	Ensemble C [No bounding box results]	0.02251	0.590987
WMW	Ensemble E [No bounding box results]	0.02258	0.591018
WMW	Ensemble A [No bounding box results]	0.0227	0.591153
WMW	Ensemble D [No bounding box results]	0.0227	0.591039
WMW	Ensemble B [No bounding box results]	0.0227	0.59106
Trimps-Soushen	Result-1	0.02481	0.067698
Trimps-Soushen	Result-2	0.02481	0.06525
Trimps-Soushen	Result-3	0.02481	0.064991
Trimps-Soushen	Result-4	0.02481	0.065261
Trimps-Soushen	Result-5	0.02481	0.065302
NUS-Qihoo_DPNs (CLS-LOC)	[E2] CLS:: Dual Path Networks + Basic Ensemble	0.0274	0.088093
NUS-Qihoo_DPNs (CLS-LOC)	[E1] CLS:: Dual Path Networks + Basic Ensemble	0.02744	0.088269
BDAT	provide_class	0.02962	0.086942
BDAT	provide_box	0.03158	0.081392
MIL_UT	Ensemble of 9 models (classification-only)	0.03205	0.596164
SIIT_KAIST-SKT	ensemble 2	0.03226	0.128924
MIL_UT	Ensemble of 10 models (classification-only)	0.03228	0.596174

Squeeze-and-Excitation Blocks

- Improving the quality of representations produced by a network by explicitly modelling the interdependencies between the channels of its convolutional features

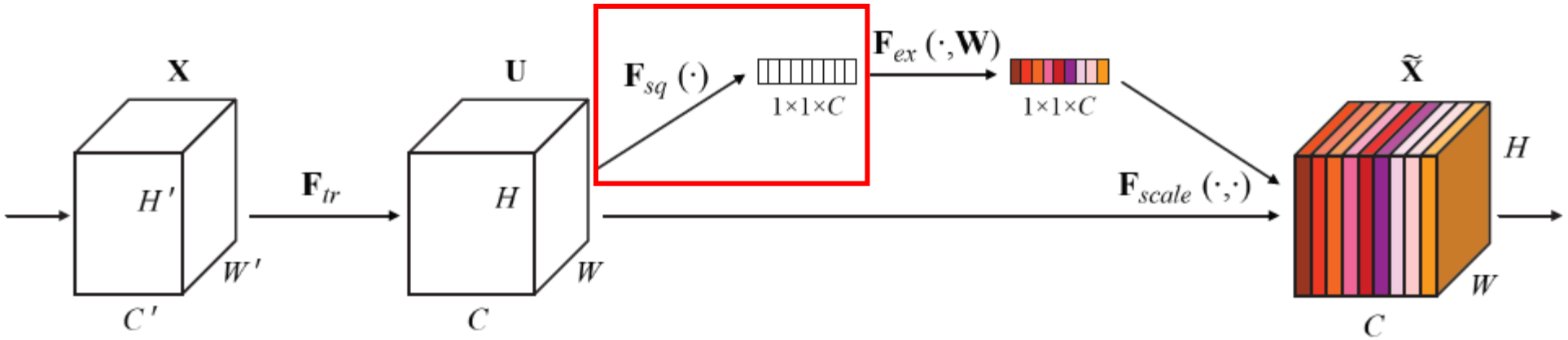


Standard Convolution

- In standard convolution case, the output is produced by a summation through all channels
- Channel dependencies are implicitly embedded in output feature maps
- But, they are entangled with the local spatial correlation captured by filters

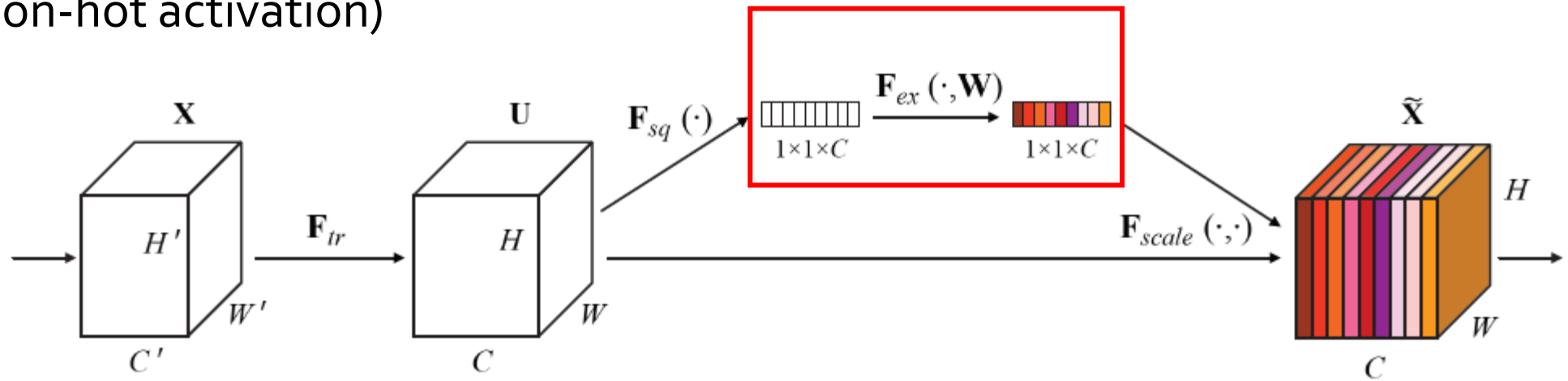
Squeeze : Global Information Embedding

- Authors propose to squeeze global spatial information into a channel descriptor.
- This is achieved by using global average pooling to generate channel-wise statistics.
- The output of the transformation(GAP) can be interpreted as a collection of the local descriptors whose statistics are expressive for whole image



Excitation : Adaptive Recalibration

- To make use of the information aggregated in the squeeze operation, authors follow it with a second operation which aims to fully captured channel-wise dependencies.
- The function must meet two criteria
  - It must be flexible(it must be capable of learning a nonlinear interaction btw channels)
  - It must learn a non-mutually-exclusive relationship(rather than enforcing a on-hot activation)

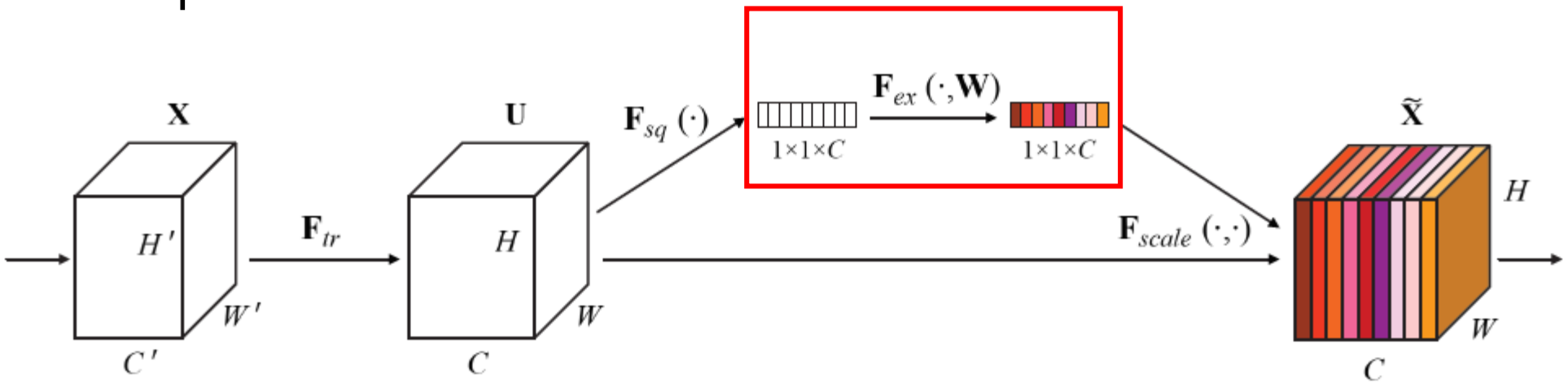


Excitation : Adaptive Recalibration

- To meet these criteria, authors opt to employ a simple gating mechanism with a sigmoid activation (σ : sigmoid, δ : ReLU)

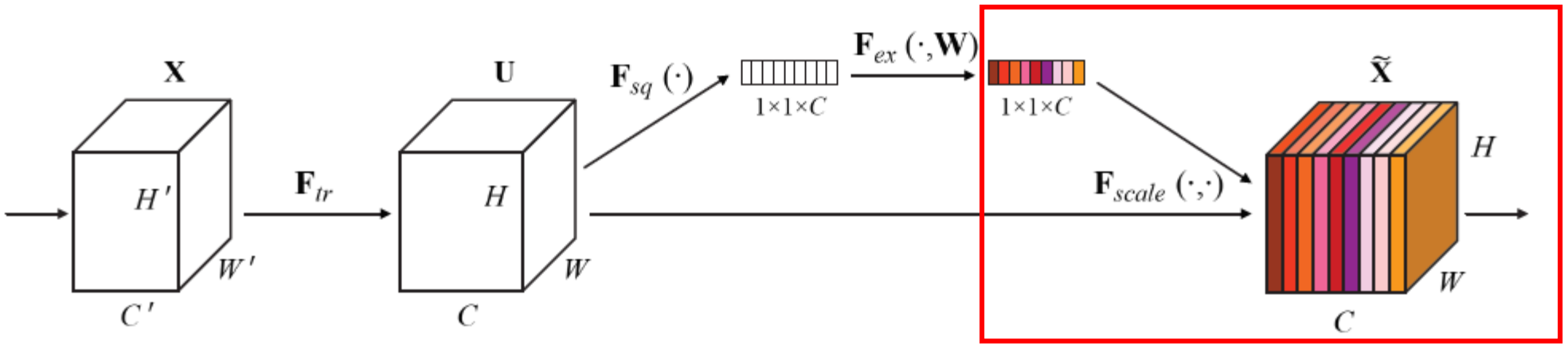
$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2\delta(\mathbf{W}_1\mathbf{z})),$$

- Forming a bottleneck with two fully connected layers around the non-linearity. A dimensionality-reduction layer with parameters W1 and reduction ratio r, a ReLU and then a dimensionality-increasing layer with parameters W2



Excitation : Adaptive Recalibration

- The final output of the block is obtained by rescaling the transformation output with the activations – channel-wise multiplication



Instantiation

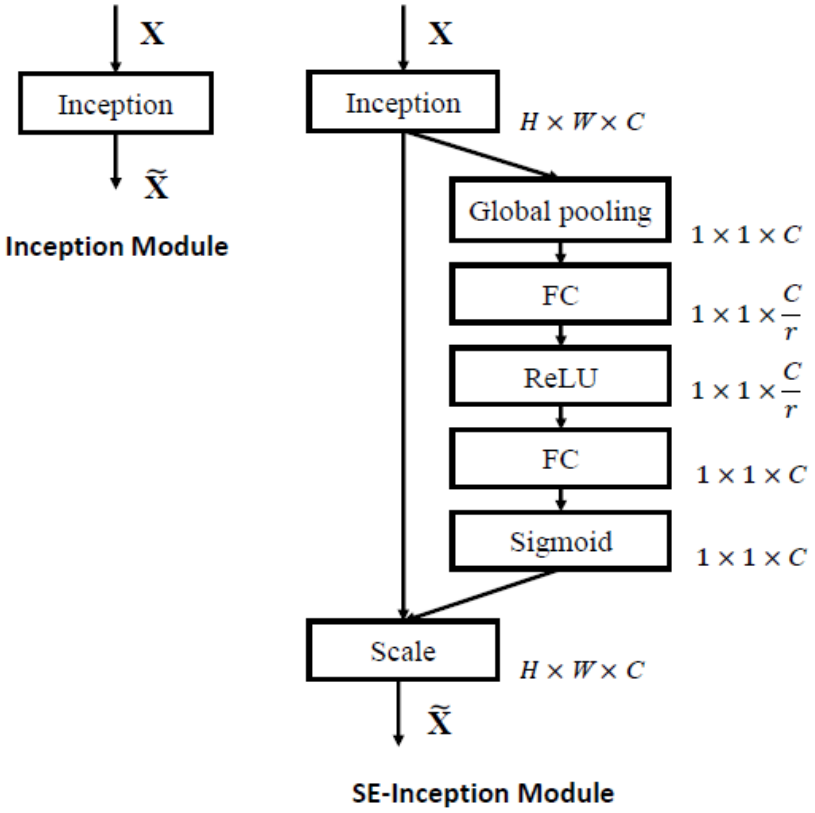


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

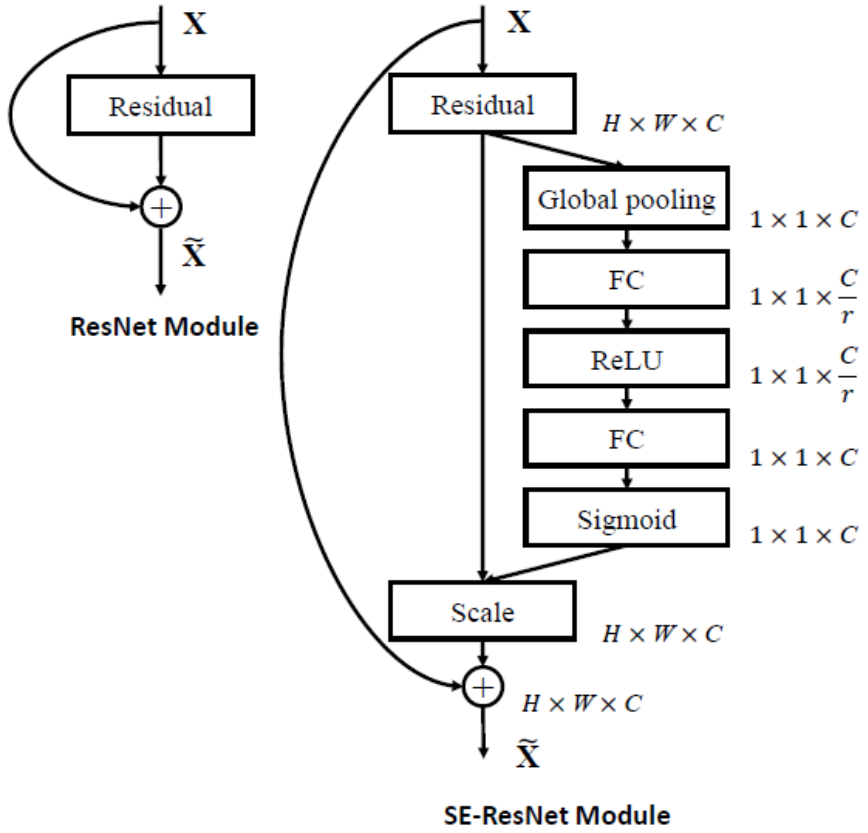


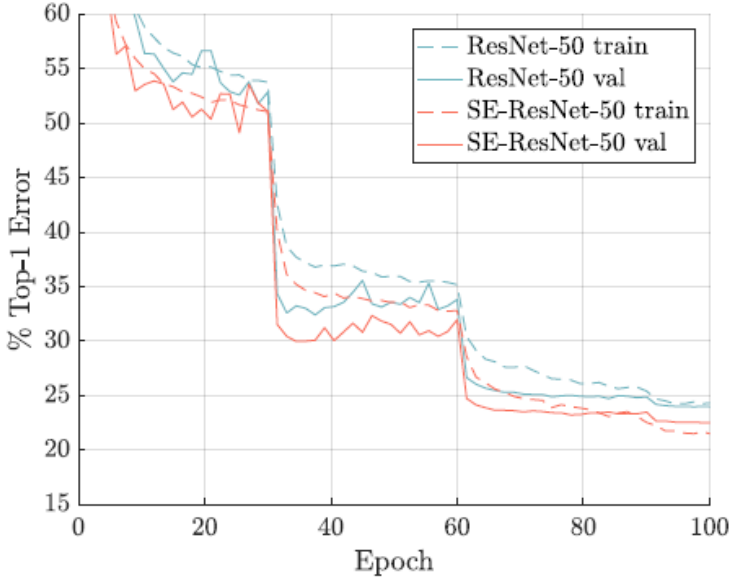
Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

Example Architecture

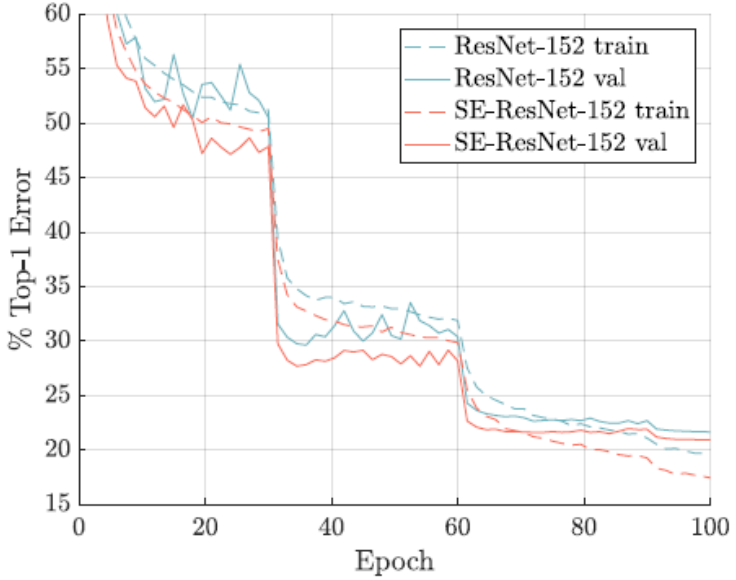
TABLE 1
(Left) ResNet-50. (Middle) SE-ResNet-50. (Right) SE-ResNeXt-50 with a 32×4d template. The shapes and operations with specific parameter settings of a residual building block are listed inside the brackets and the number of stacked blocks in a stage is presented outside. The inner brackets following by *fc* indicates the output dimension of the two fully connected layers in an SE module.

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
112 × 112	conv, 7 × 7, 64, stride 2		
56 × 56	max pool, 3 × 3, stride 2		
	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \quad C=32 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28 × 28	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \quad C=32 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14 × 14	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \quad C=32 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 1024 \\ \text{conv}, 3\times3, 1024 \quad C=32 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1 × 1	global average pool, 1000-d *fc*, softmax		

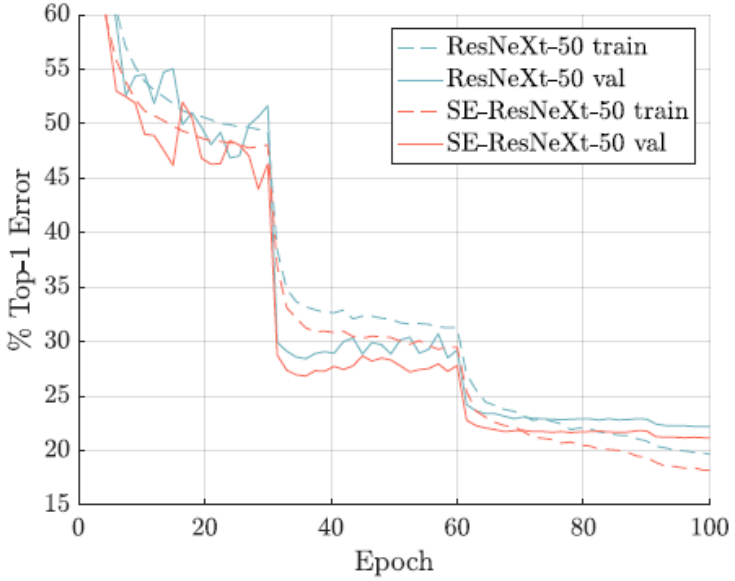
Results



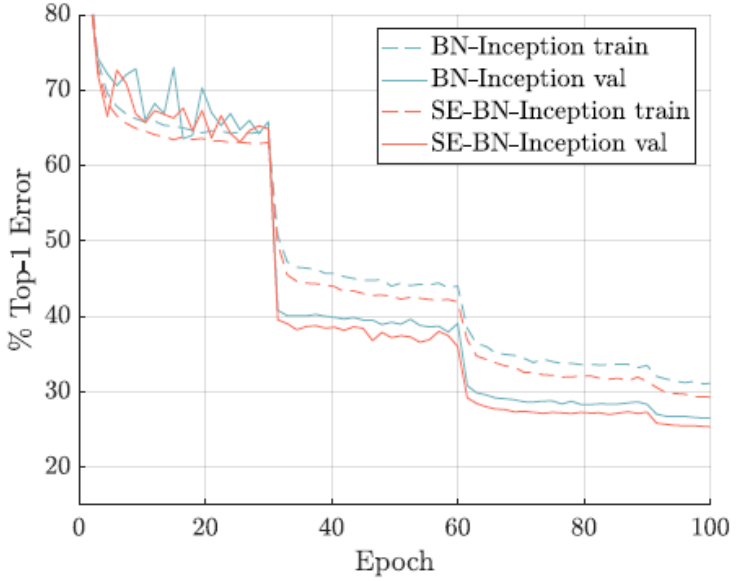
(a) ResNet-50 and SE-ResNet-50



(b) ResNet-152 and SE-ResNet-152



(c) ResNeXt-50 and SE-ResNeXt-50



(d) BN-Inception and SE-BN-Inception

Xception

Xception: Deep Learning with Depthwise Separable Convolutions

François Chollet
Google, Inc.
fchollet@google.com

Abstract

We present an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads us to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions. We show that this architecture, dubbed Xception, slightly outperforms Inception V3 on the ImageNet dataset (which Inception V3 was designed for), and significantly outperforms Inception V3 on a larger image classification dataset comprising 350 million images and 17,000 classes. Since the Xception architecture has the same number of parameters as Inception V3, the performance gains are not due to increased capacity but rather to a more efficient use of model parameters.

as GoogLeNet (Inception V1), later refined as Inception V2 [7], Inception V3 [21], and most recently Inception-ResNet [19]. Inception itself was inspired by the earlier Network-In-Network architecture [11]. Since its first introduction, Inception has been one of the best performing family of models on the ImageNet dataset [14], as well as internal datasets in use at Google, in particular JFT [5].

The fundamental building block of Inception-style models is the Inception module, of which several different versions exist. In figure 1 we show the canonical form of an Inception module, as found in the Inception V3 architecture. An Inception model can be understood as a stack of such modules. This is a departure from earlier VGG-style networks which were stacks of simple convolution layers.

While Inception modules are conceptually similar to convolutions (they are convolutional feature extractors), they empirically appear to be capable of learning richer representations with less parameters. How do they work, and how do they differ from regular convolutions? What design strategies come after Inception?

MobileNets

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko
Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam

Google Inc.

{howarda,menglong,bochen,dkalenichenko,weijunw,weyand,anm,hadam}@google.com

Abstract

We present a class of efficient models called MobileNets for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. We introduce two simple global hyper-parameters that efficiently trade off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem. We present extensive experiments on resource and accuracy tradeoffs and show

models. Section 3 describes the MobileNet architecture and two hyper-parameters width multiplier and resolution multiplier to define smaller and more efficient MobileNets. Section 4 describes experiments on ImageNet as well a variety of different applications and use cases. Section 5 closes with a summary and conclusion.

2. Prior Work

There has been rising interest in building small and efficient neural networks in the recent literature, e.g. [16, 34, 12, 36, 22]. Many different approaches can be generally

Xception

- Observation
  - Inception module try to explicitly factoring two tasks done by a single convolution kernel: mapping cross-channel correlation and spatial correlation
- Inception hypothesis
  - By inception module, these two correlations are sufficiently decoupled

→ Would it be reasonable to make a much stronger hypothesis than the Inception hypothesis?

Xception

Figure 1. A canonical Inception module (Inception V3).

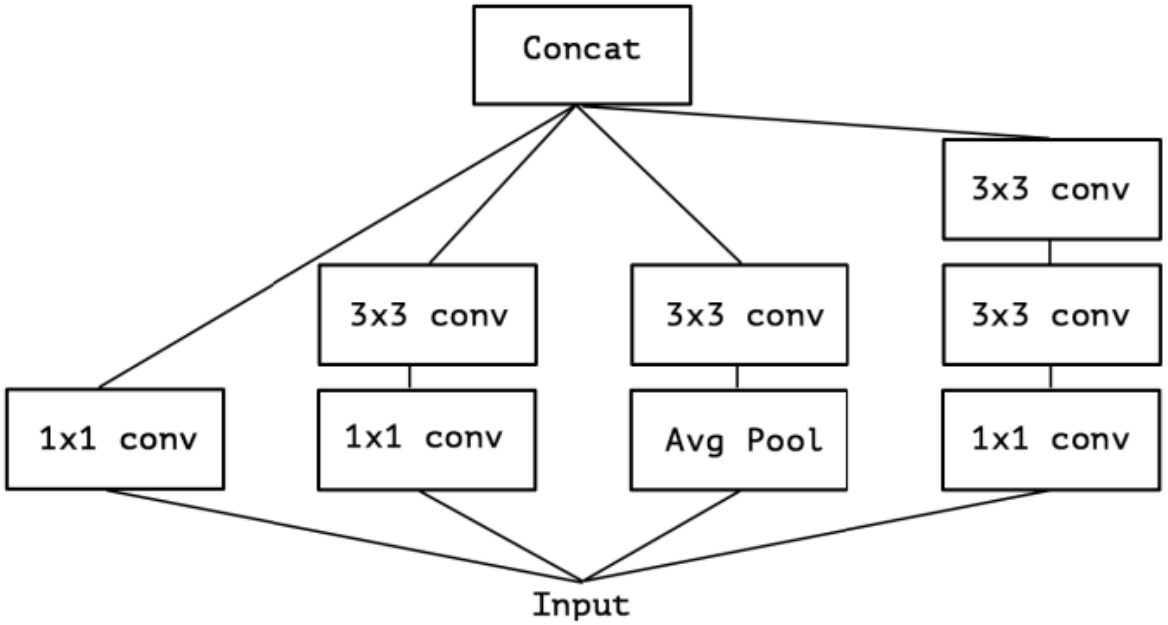
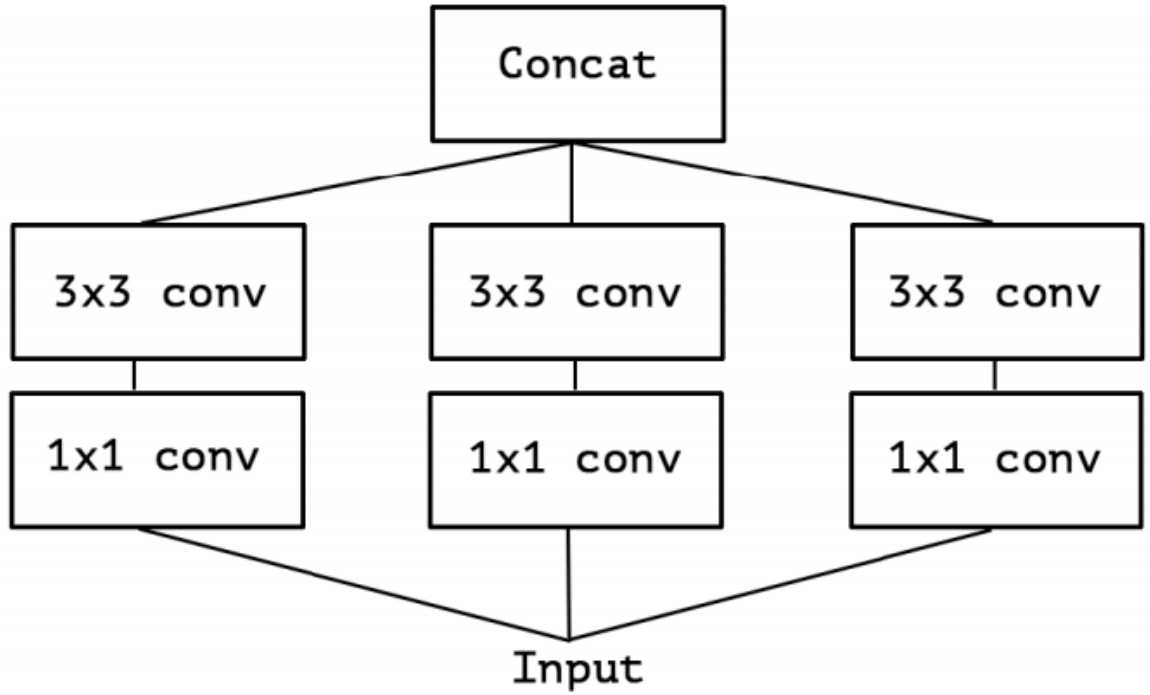


Figure 2. A simplified Inception module.



Equivalent Reformulation

Figure 2. A simplified Inception module.

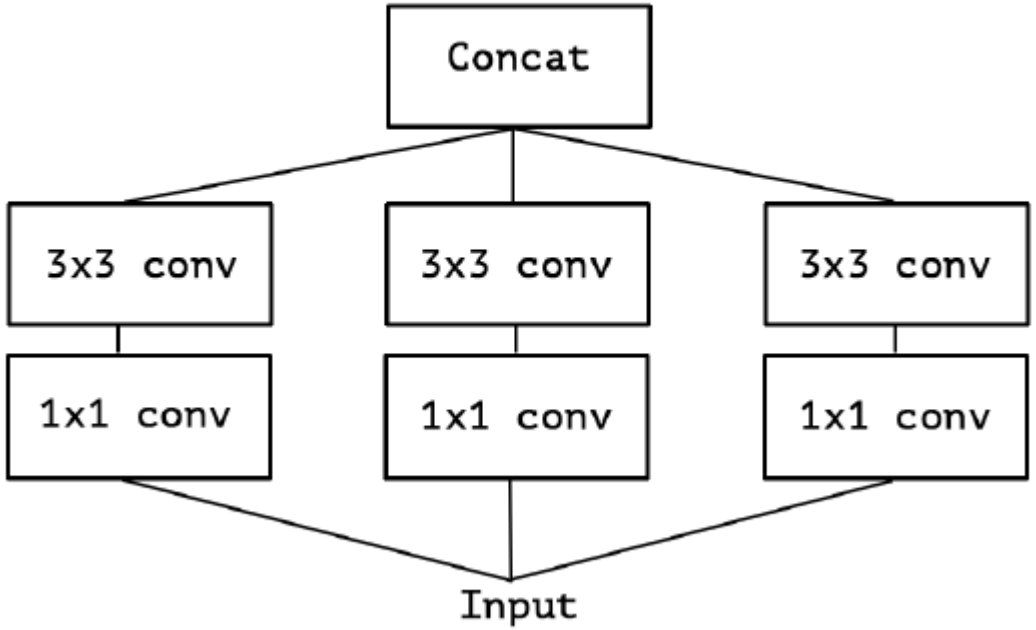
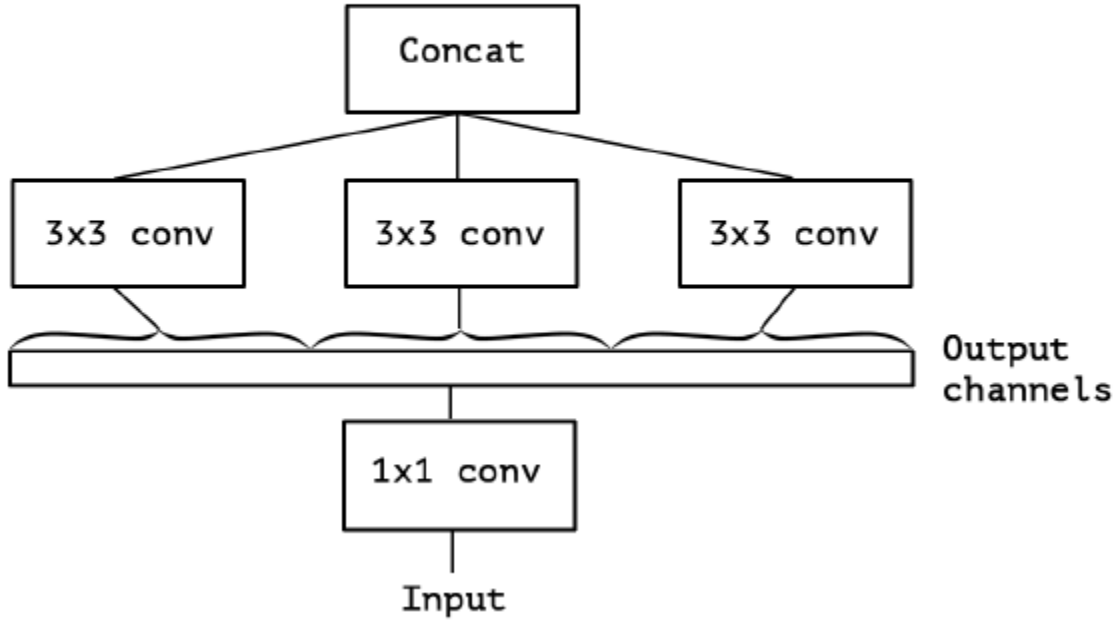
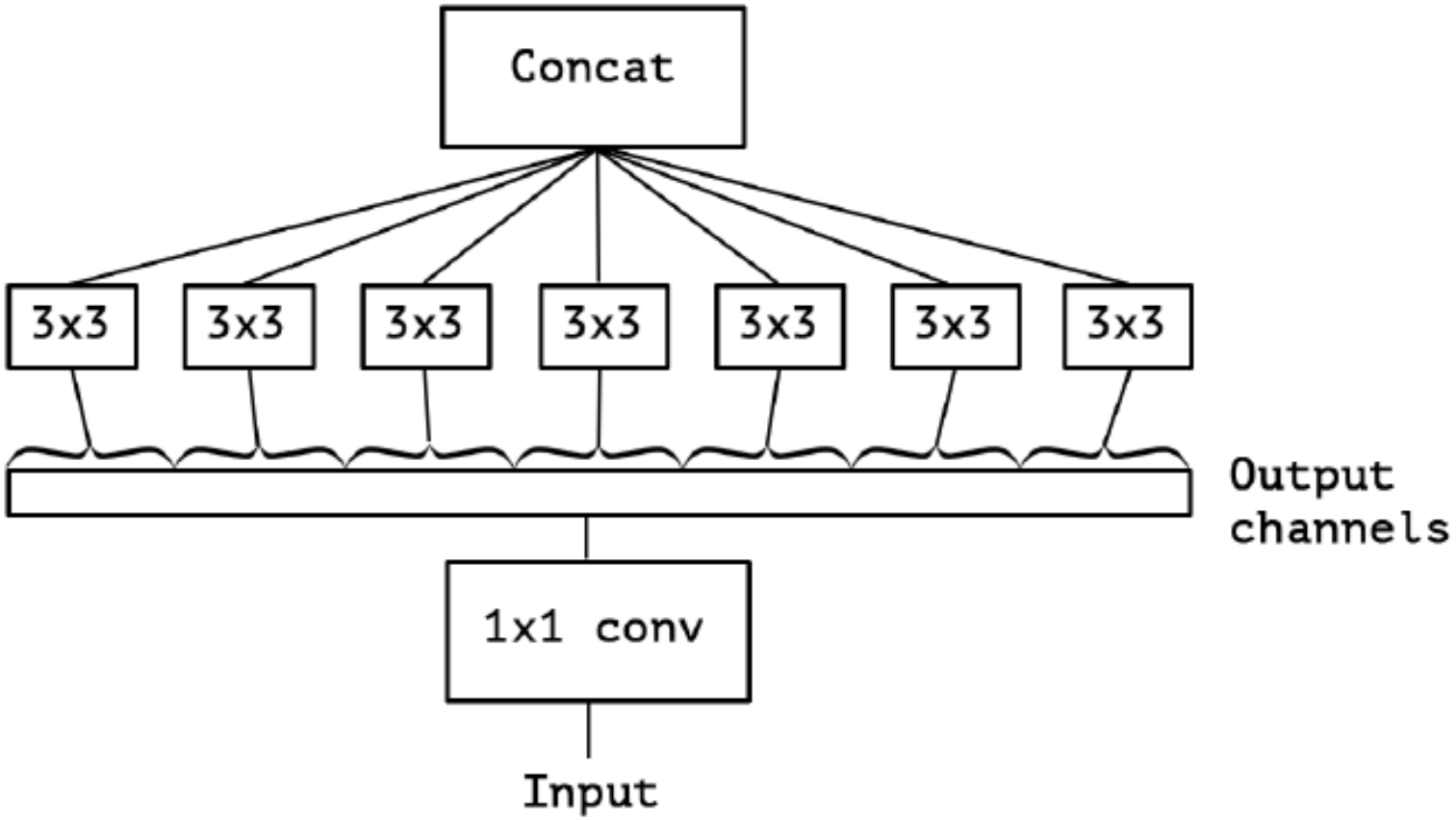


Figure 3. A strictly equivalent reformulation of the simplified Inception module.

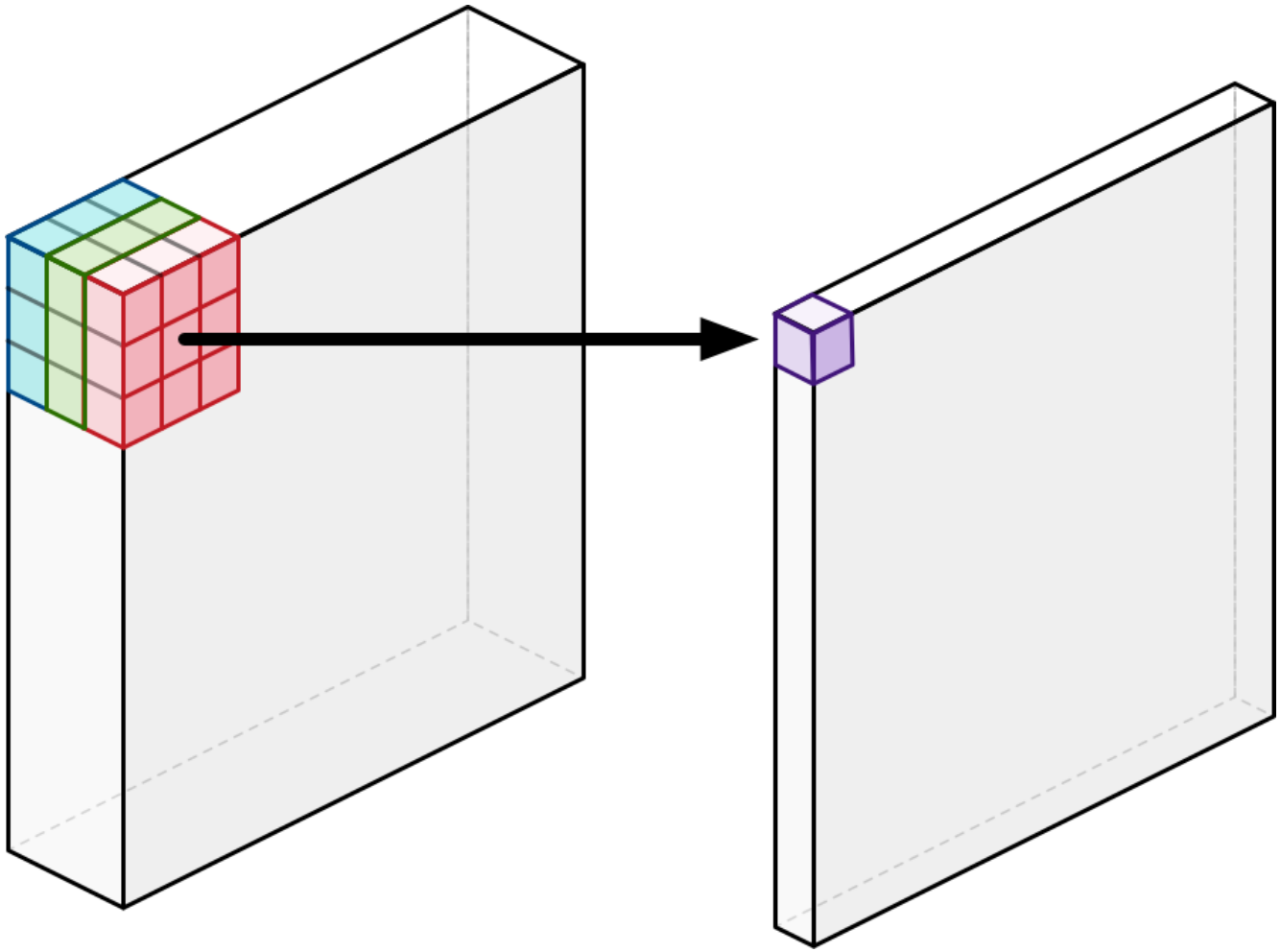


Extreme Version of Inception Module

Figure 4. An "extreme" version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



Standard Convolution

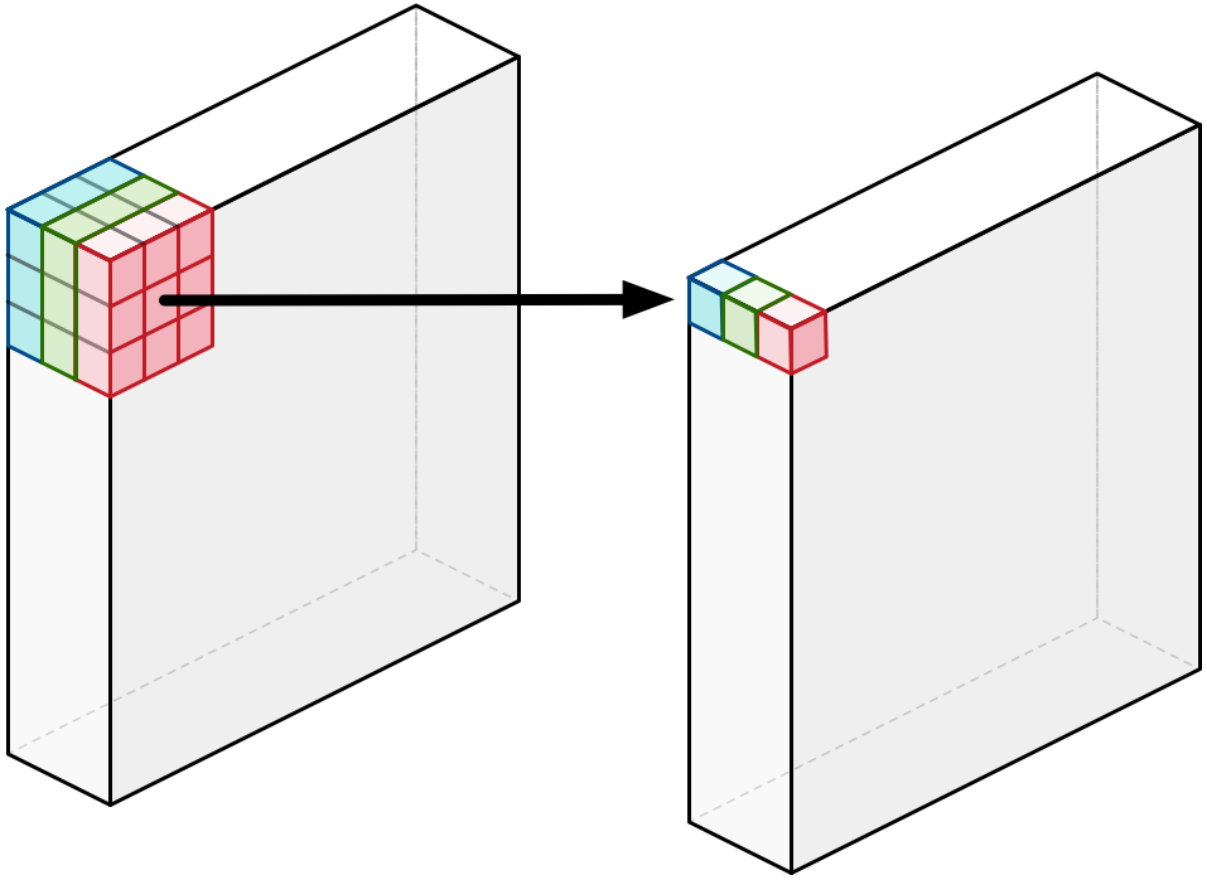


Standard convolution

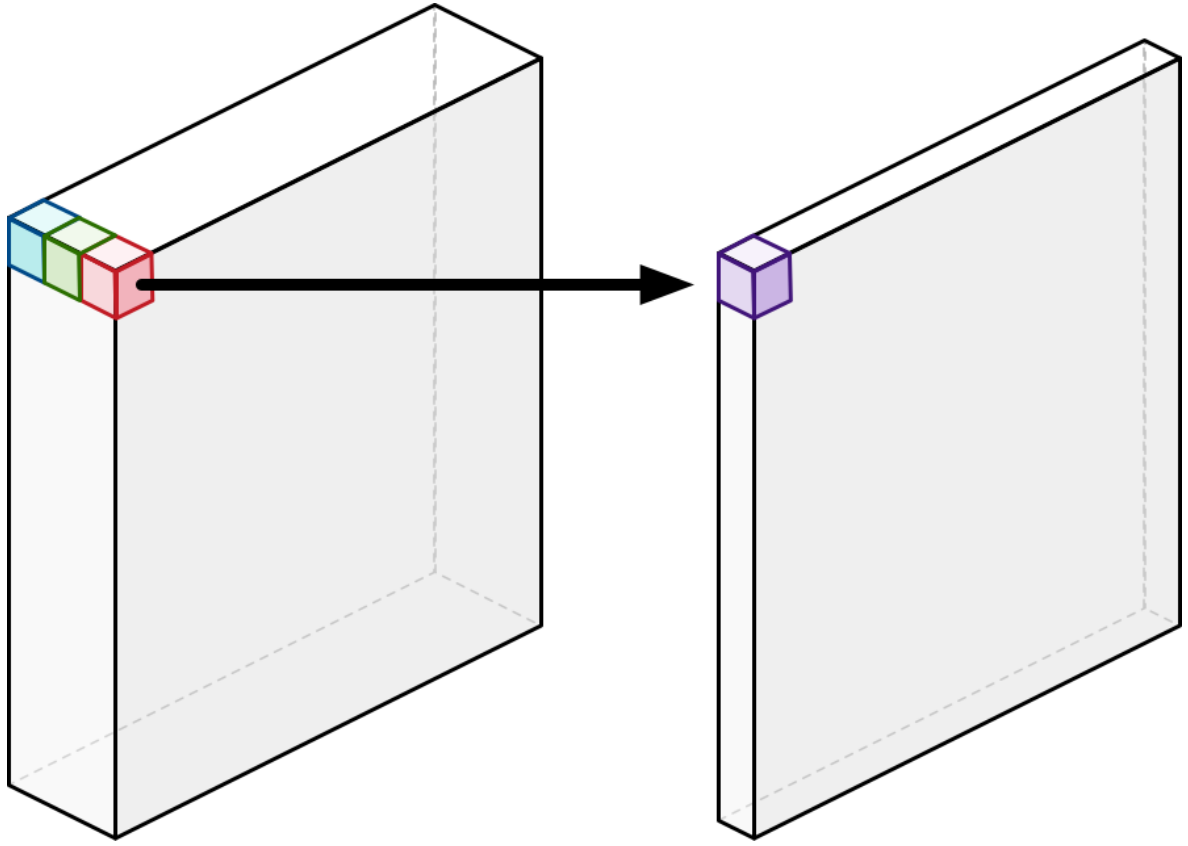
Figures from http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

Depthwise Separable Convolution

- Depthwise Convolution + Pointwise Convolution(1x1 convolution)



Depthwise convolution



Pointwise convolution

Figures from http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

Xception vs Depthwise Separable Convolution

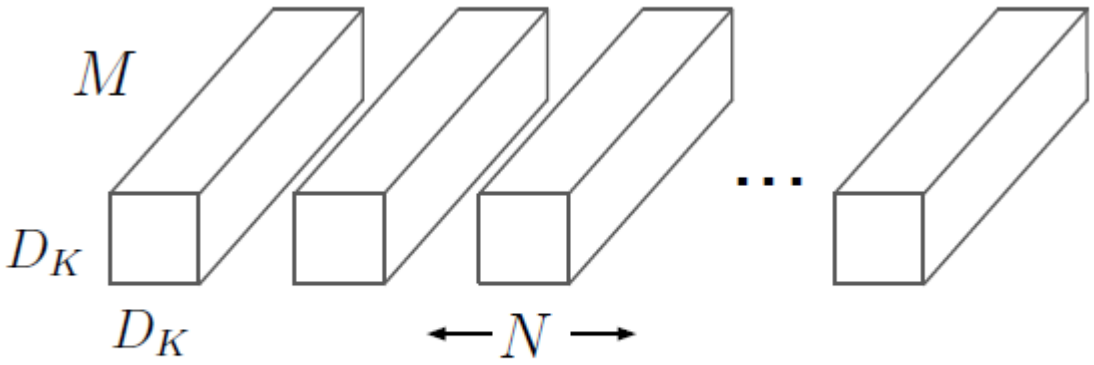
- The order of the operations
- The presence or absence of a non-linearity after the first operation

Inception modules lie in between!

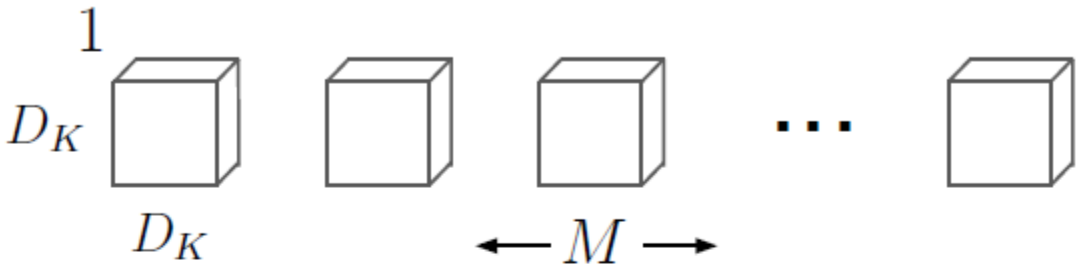
Xception Hypothesis

: Make the mapping that *entirely* decouples
the cross-channels correlations and spatial correlations

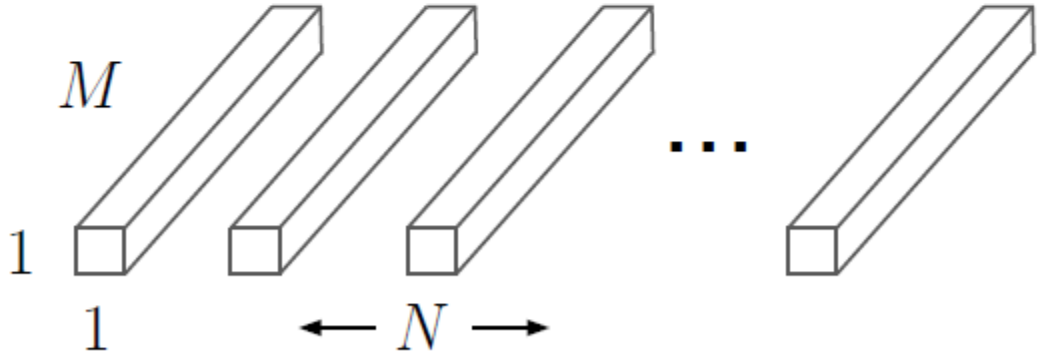
Standard Convolution vs Depthwise Separable Convolution



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Standard Convolution vs Depthwise Separable Convolution

- Standard convolutions have the computational cost of
  - $D_K \times D_K \times M \times N \times D_F \times D_F$
- Depthwise separable convolutions cost
  - $D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F$
- Reduction in computations
  - $1 / N + 1 / D_K^2$
  - If we use 3x3 depthwise separable convolutions, we get between 8 to 9 times less computations

Depthwise Separable Convolutions

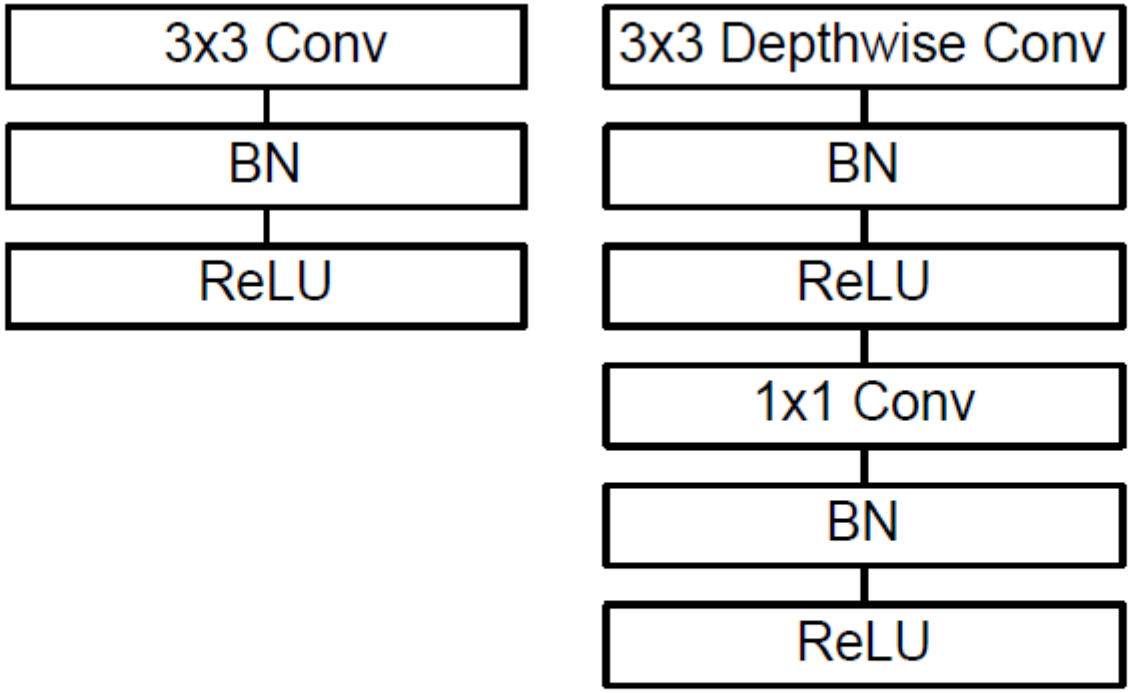


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Model Structure

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Width Multiplier & Resolution Multiplier

- For a given layer and width multiplier α, the number of input channels M becomes αM and the number of output channels N becomes αN – where α with typical settings of 1, 0.75, 0.5 and 0.25
- The second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier ρ
- Computational cost:

$$D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_F \times \rho D_F$$

Width Multiplier & Resolution Multiplier

Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the previous row. This example is for an internal MobileNet layer with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

Experiments – Model Choices

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

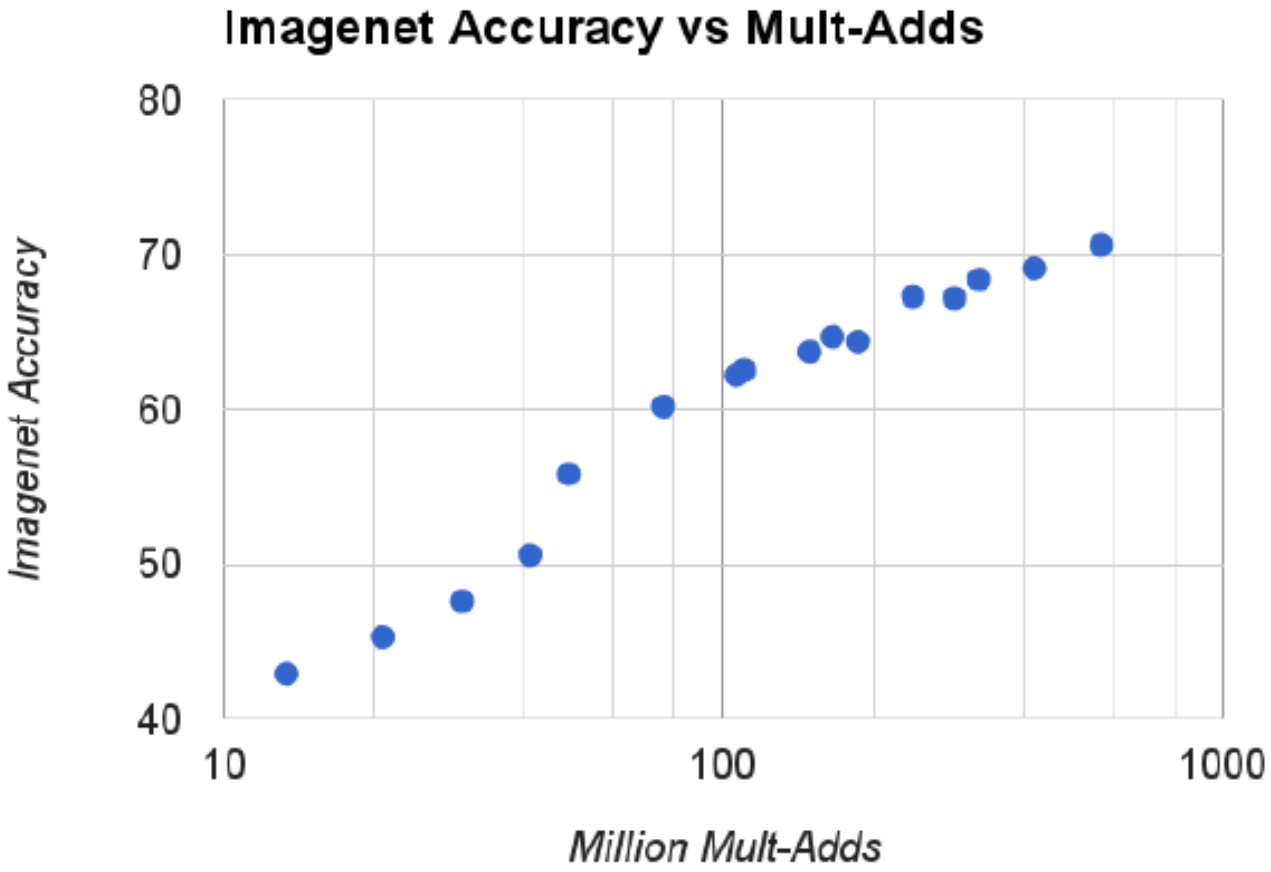
Table 6. MobileNet Width Multiplier

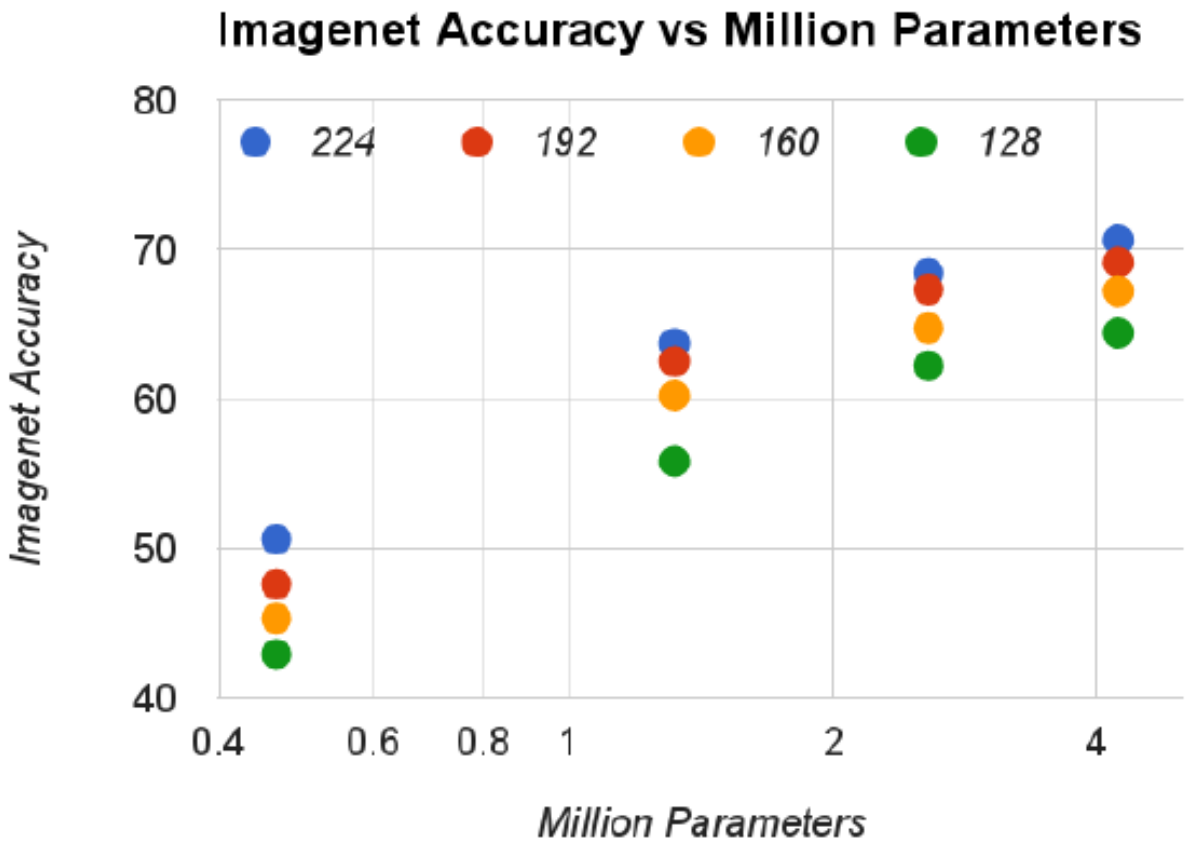
Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

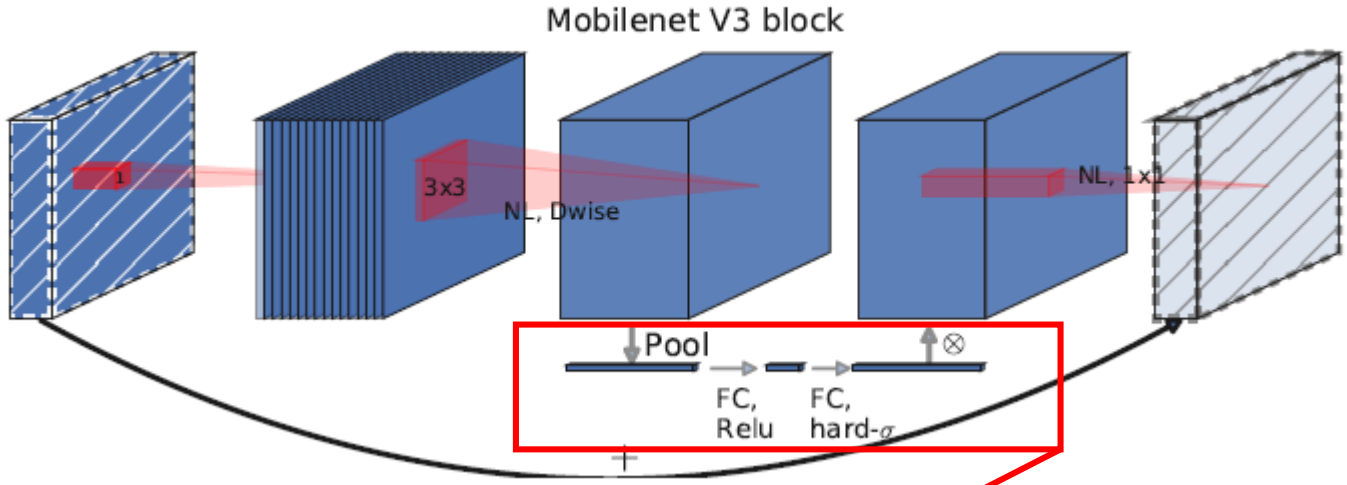
Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

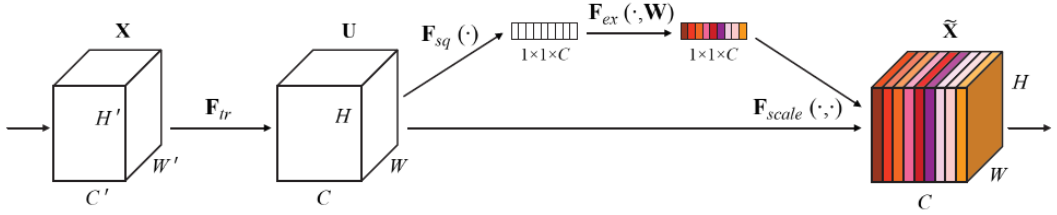
Model Shrinking Hyperparameters





MobileNet V2, V3





$$\text{swish}\, x = x \cdot \sigma(x)$$

$$\text{h-swish}[x] = x\frac{\text{ReLU6}(x+3)}{6}$$