6.5930/1

Hardware Architectures for Deep Learning

Popular DNN Models

February 12, 2024

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

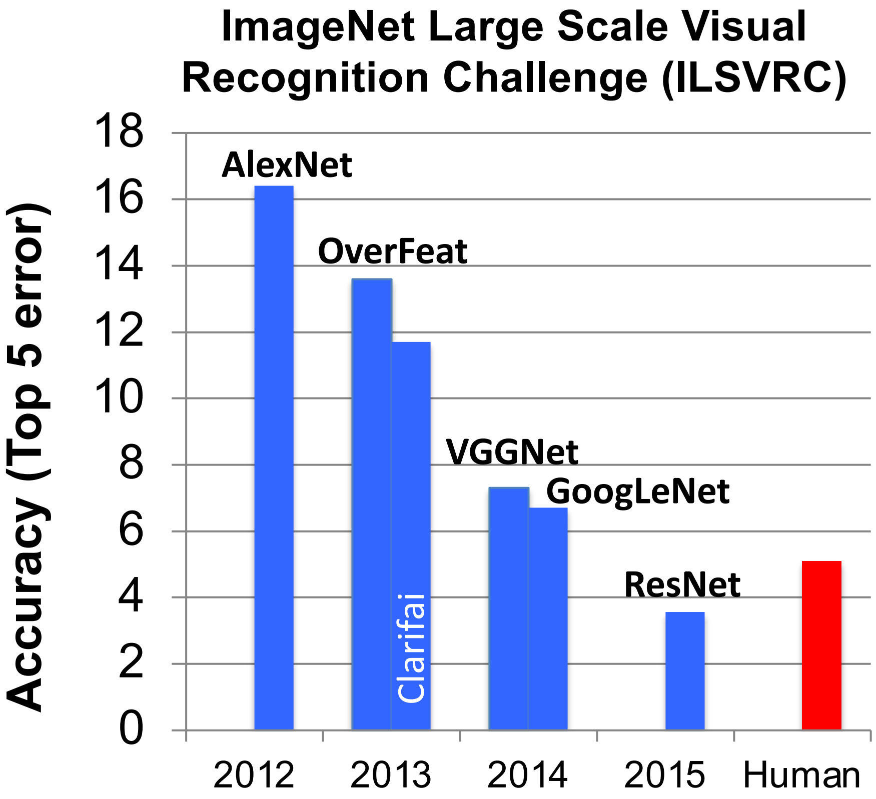
Goals of Today's Lecture

- Last lecture covered the building blocks of CNNs; this lecture describes how we put these blocks together to form a CNN.
- Overview of various well-known CNN models
  - CNN 'models' are also referred to as 'network architectures'; however, we prefer to use the term 'model' in this class to avoid overloading the term 'architecture'
- We group the CNN models into two categories
  - **High Accuracy CNN Models**: Designed to maximize accuracy to compete in the ImageNet Challenge
  - **Efficient CNN Models:** Designed to reduce the **number of weights** and **operations (specifically MACs)** while maintaining accuracy

High Accuracy CNN Models

Popular CNNs

- **LeNet** (1998)
- **AlexNet** (2012)
- **OverFeat** (2013)
- **VGGNet** (2014)
- **GoogleNet** (2014)
- **ResNet** (2015)



[**Russakovsky**, *IJCV* 2015]

MNIST

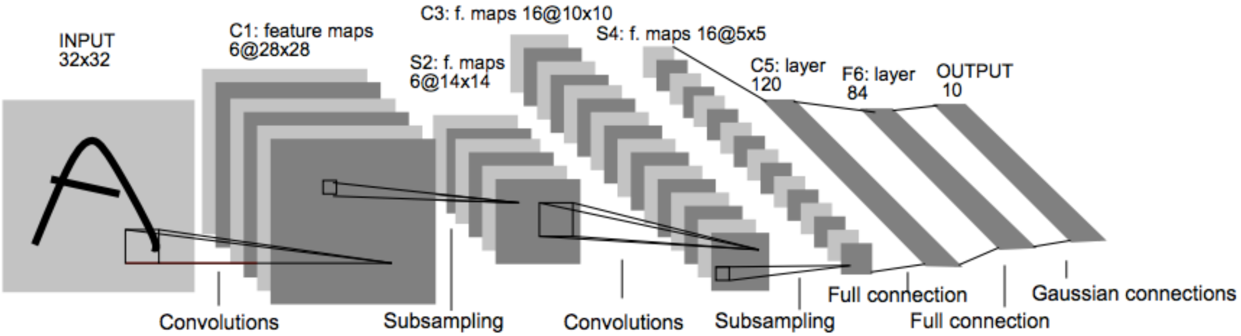
Digit Classification
28x28 pixels (B&W)
10 Classes
60,000 Training
10,000 Testing

http://yann.lecun.com/exdb/mnist/

LeNet-5

CONV Layers: 2
Fully Connected Layers: 2
Weights: 60k
MACs: 341k
Sigmoid used for non-linearity

Digit Classification!
(MNIST Dataset)



six
5x5 filters

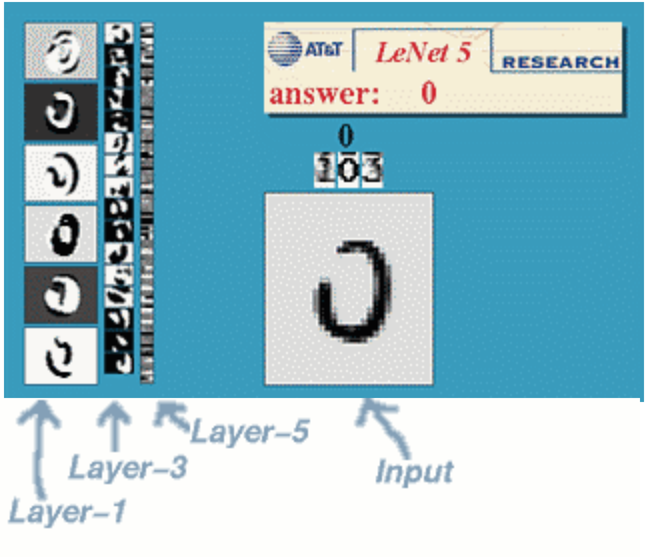
2x2
average
pooling

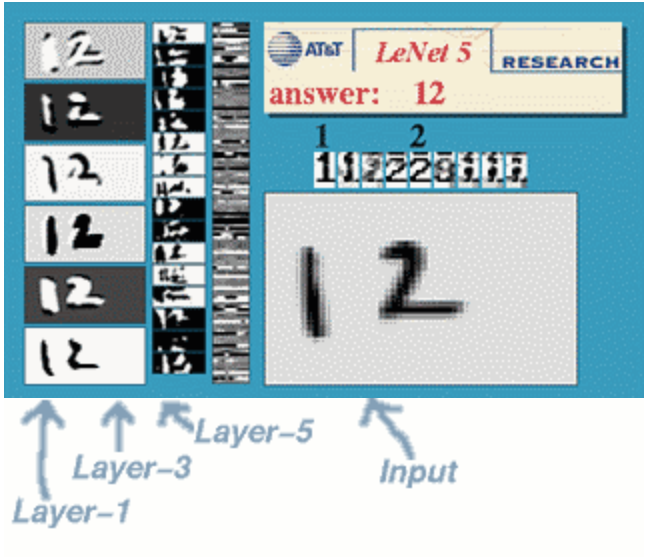
sixteen
5x5 filters

2x2
average
pooling

[**Lecun**, *Proceedings of the IEEE*, 1998]

LeNet-5





http://yann.lecun.com/exdb/lenet/

IMAGENET

Image Classification
~256x256 pixels (color)
1000 Classes
1.3M Training
100,000 Testing (50,000 Validation)

For **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** accuracy of classification task reported based on top-1 and top-5 error

Image Source: http://karpathy.github.io/

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

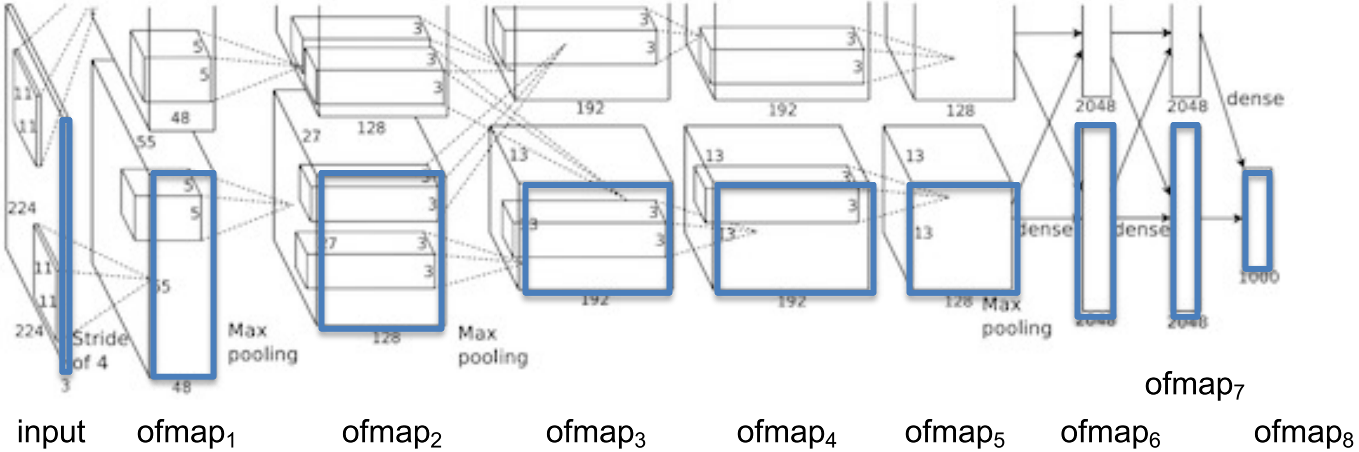
AlexNet

CONV Layers: 5
Fully Connected Layers: 3
Weights: 61M
MACs: 724M
ReLU used for non-linearity

ILSCVR12 Winner

Uses Local Response Normalization (LRN)

[**Krizhevsky**, *NeurIPS* 2012]



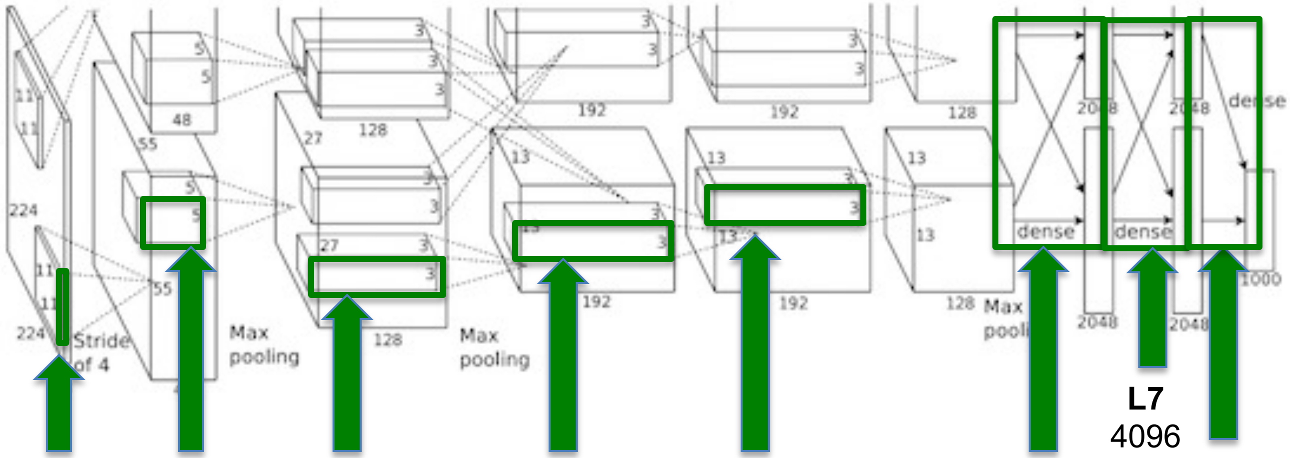
AlexNet

CONV Layers: 5
Fully Connected Layers: 3
Weights: 61M
MACs: 724M
ReLU used for non-linearity

ILSCVR12 Winner

Uses Local Response Normalization (LRN)

[**Krizhevsky**, *NeurIPS* 2012]



L1
96 filters
(11x11x3)

L2
256 filters
(5x5x48)

L3
384 filters
(3x3x256)

L4
384 filters
(3x3x192)

L5
384 filters
(3x3x192)

L6
4096 filters
(13x13x256)

L7
4096
(4096x1)

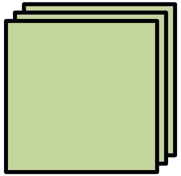
L8
1000
(4096x1)

Large Sizes with Varying Shapes

AlexNet Convolutional Layer Configurations

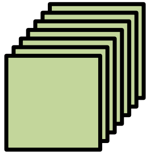
Layer	Filter Size (RxS)	# Filters (M)	# Channels (C)	Stride
1	11x11	96	3	4
2	5x5	256	48	1
3	3x3	384	256	1
4	3x3	384	192	1
5	3x3	256	192	1

Layer 1



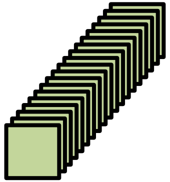
34k Params
105M MACs

Layer 2



307k Params
224M MACs

Layer 3



885k Params
150M MACs

[**Krizhevsky**, *NeurIPS* 2012]

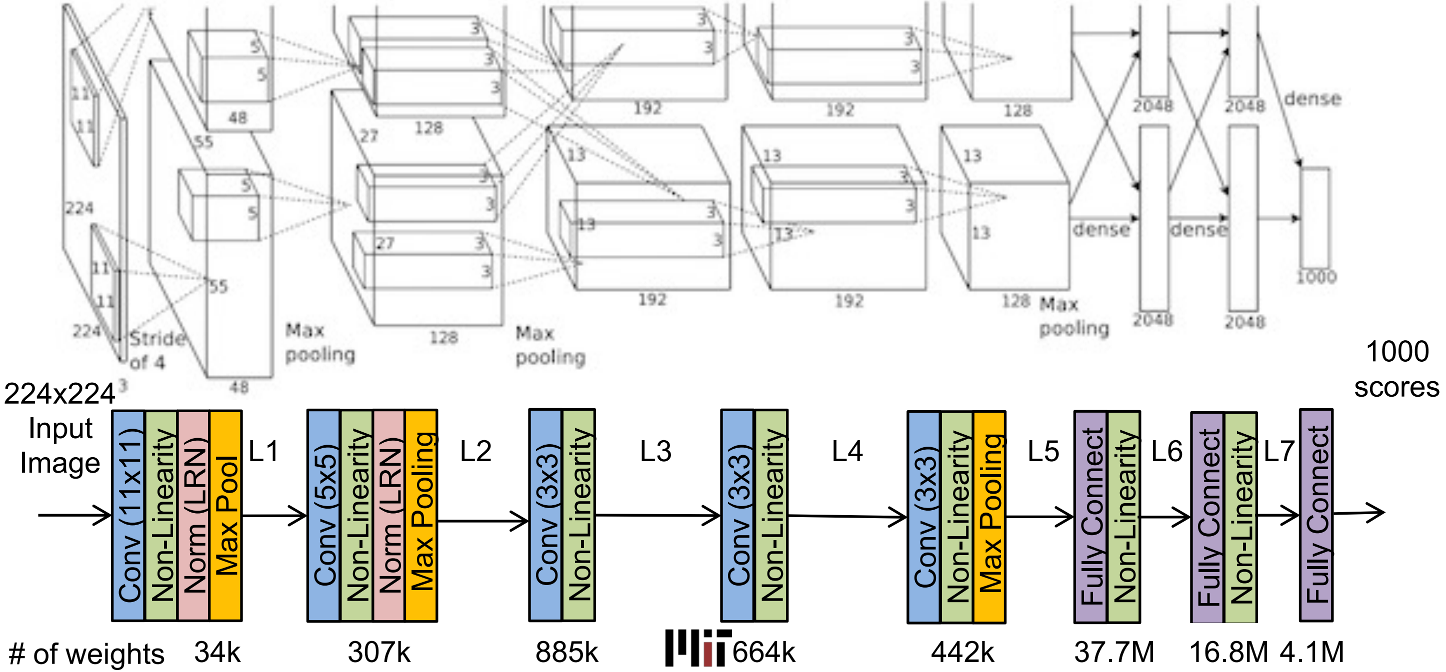
AlexNet

CONV Layers: 5
Fully Connected Layers: 3
Weights: 61M
MACs: 724M
ReLU used for non-linearity

ILSCVR12 Winner

Uses Local Response Normalization (LRN)

[**Krizhevsky**, *NeurIPS* 2012]



224x224 Input Image

L1: Conv (11x11), Non-Linearity, Norm (LRN), Max Pool
L2: Conv (5x5), Non-Linearity, Norm (LRN), Max Pooling
L3: Conv (3x3), Non-Linearity
L4: Conv (3x3), Non-Linearity
L5: Conv (3x3), Non-Linearity
Conv (3x3), Non-Linearity, Max Pooling
L6: Fully Connect, Non-Linearity
L7: Fully Connect, Non-Linearity
Fully Connect

1000 scores

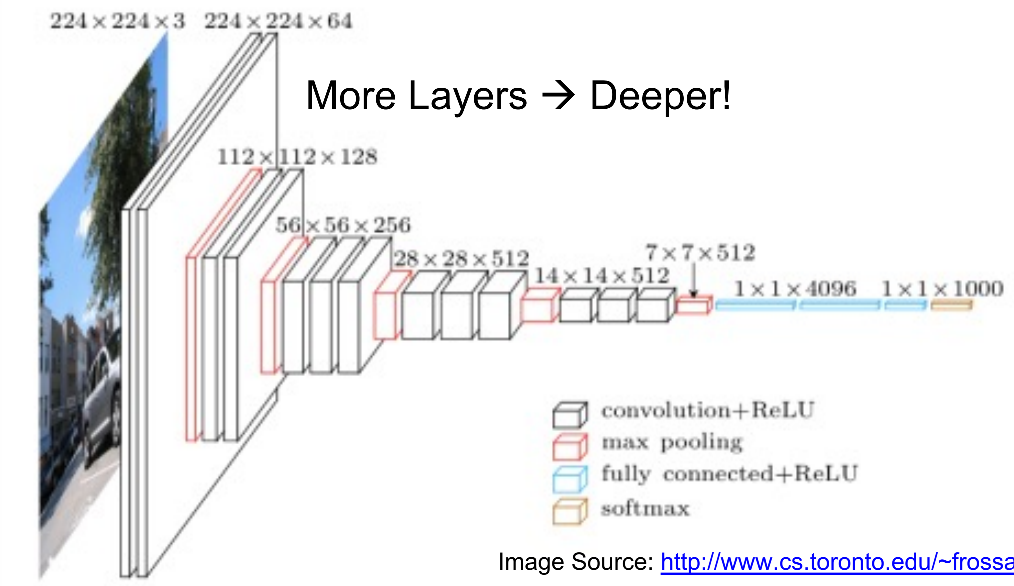
# of weights	34k	307k	885k	664k	442k	37.7M	16.8M	4.1M

VGG-16

CONV Layers: 13
Fully Connected Layers: 3
Weights: 138M
MACs: 15.5G

Also, 19-layer version

[**Simonyan**, *ICLR* 2015]



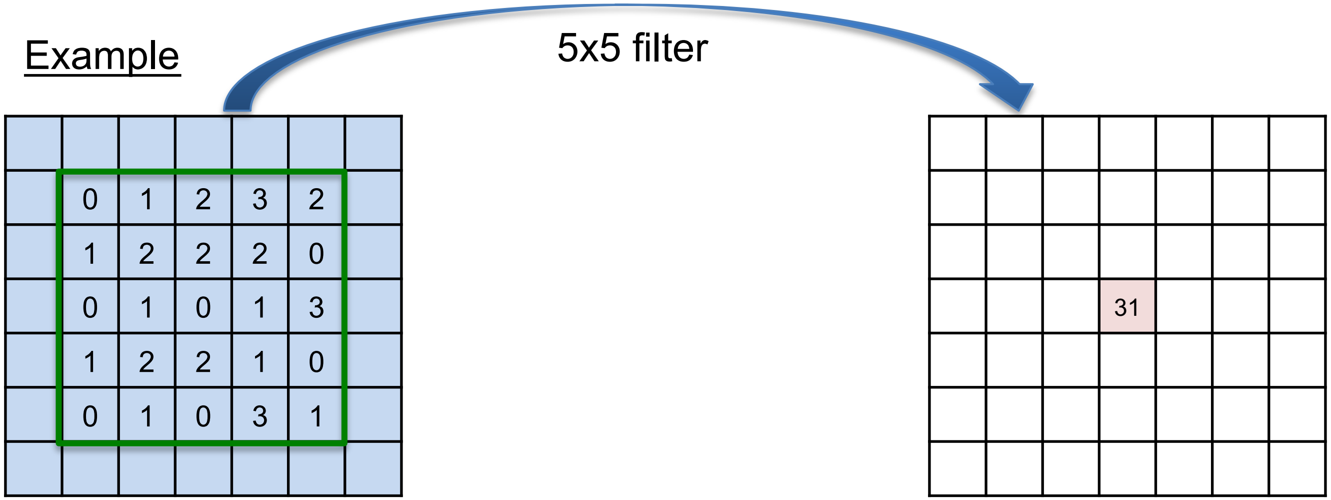
More Layers → Deeper!

Image Source: http://www.cs.toronto.edu/~frossard/post/vgg16/

Stacked Filters

- Deeper network means more weights
- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights

Example



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

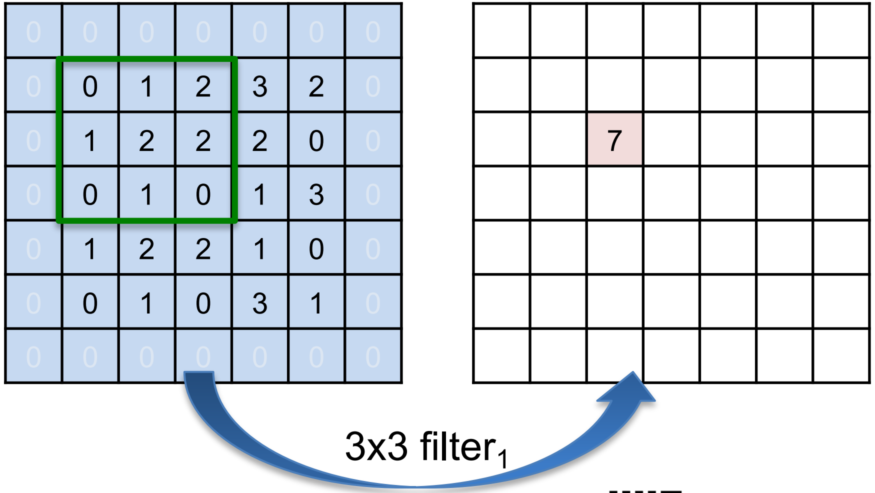
Stacked Filters

- Deeper network means more weights
- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights

filter (3x3)

0	1	0
1	1	1
0	1	0

Example



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

		7				

3x3 filter$_1$

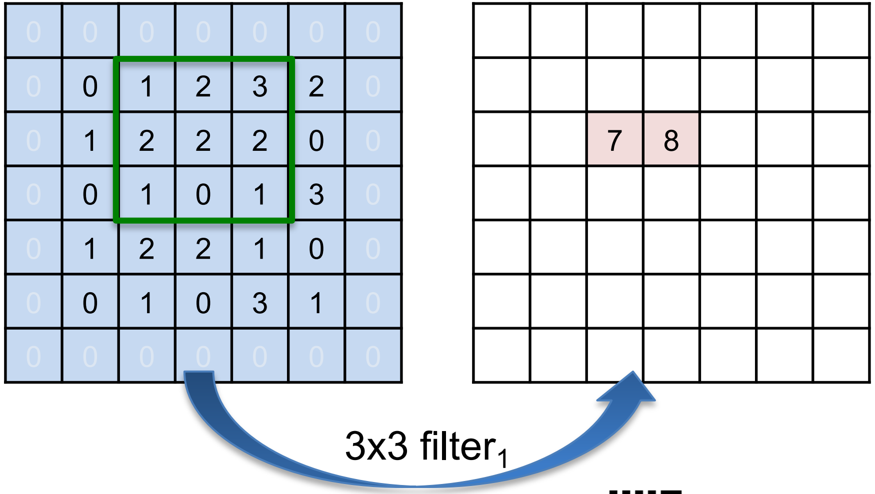
Stacked Filters

- Deeper network means more weights
- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights

filter (3x3)

0	1	0
1	1	1
0	1	0

Example



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

		7	8			

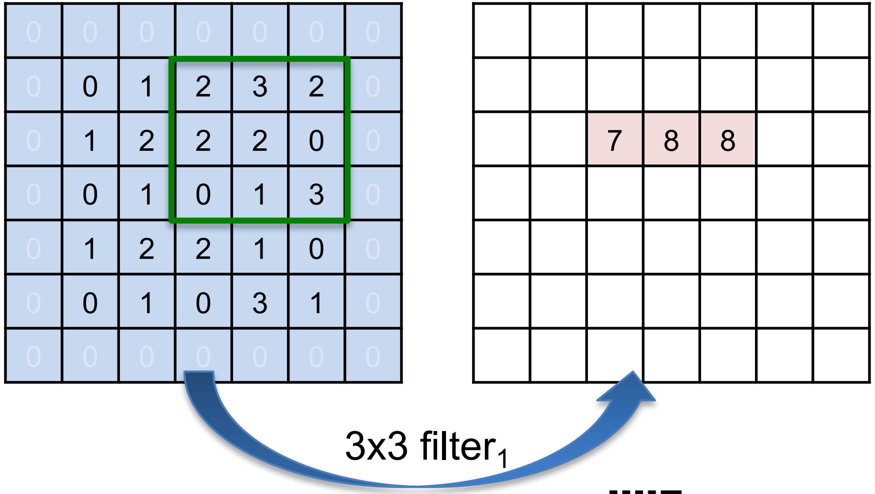
Stacked Filters

- Deeper network means more weights
- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights

filter (3x3)

0	1	0
1	1	1
0	1	0

Example



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

		7	8	8		

3x3 filter$_1$

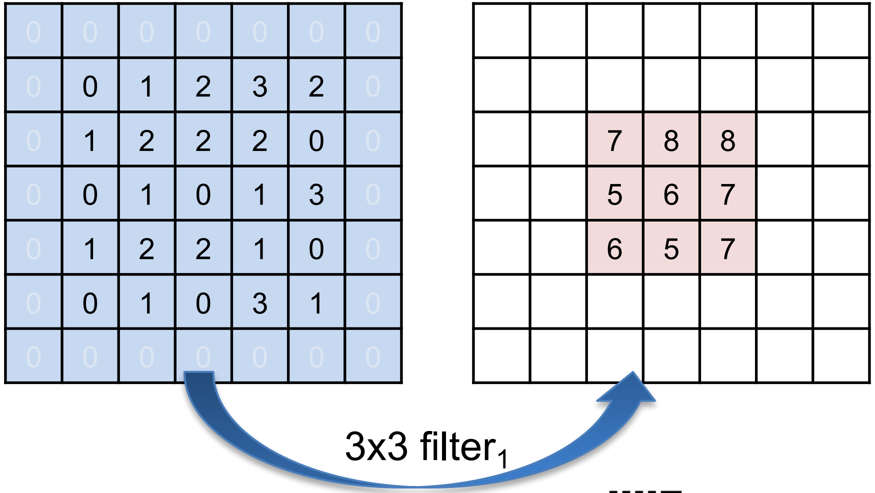
Stacked Filters

- Deeper network means more weights
- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights

filter (3x3)

0	1	0
1	1	1
0	1	0

Example



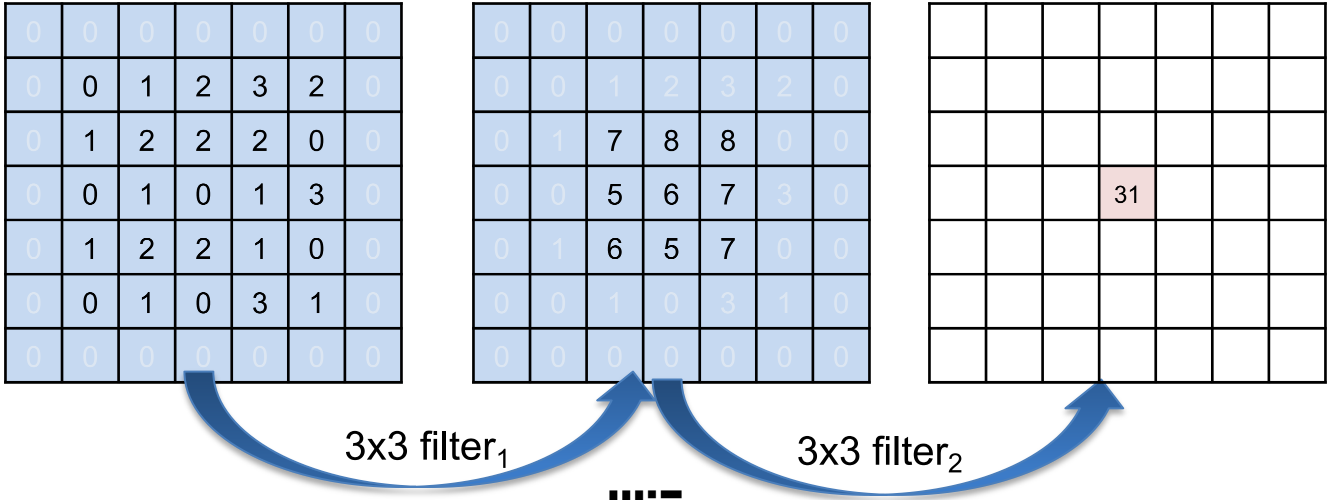
0	0	0	0	0	0	0
0	0	1	2	3	2	0
0	1	2	2	2	0	0
0	0	1	0	1	3	0
0	1	2	2	1	0	0
0	0	1	0	3	1	0
0	0	0	0	0	0	0

		7	8	8		
		5	6	7		
		6	5	7		

VGGNet: Stacked Filters

- Deeper network means more weights
- Use stack of smaller filters (3x3) to cover the same receptive field with fewer filter weights
- Non-linear activation inserted between each filter

Example: 5x5 filter (25 weights) → two 3x3 filters (18 weights)



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

0	0	0	0	0	0	0
0	0	1	2	3	2	0
0	1	7	8	8	0	0
0	0	5	6	7	3	0
0	1	6	5	7	0	0
0	0	1	0	3	1	0
0	0	0	0	0	0	0

			31			

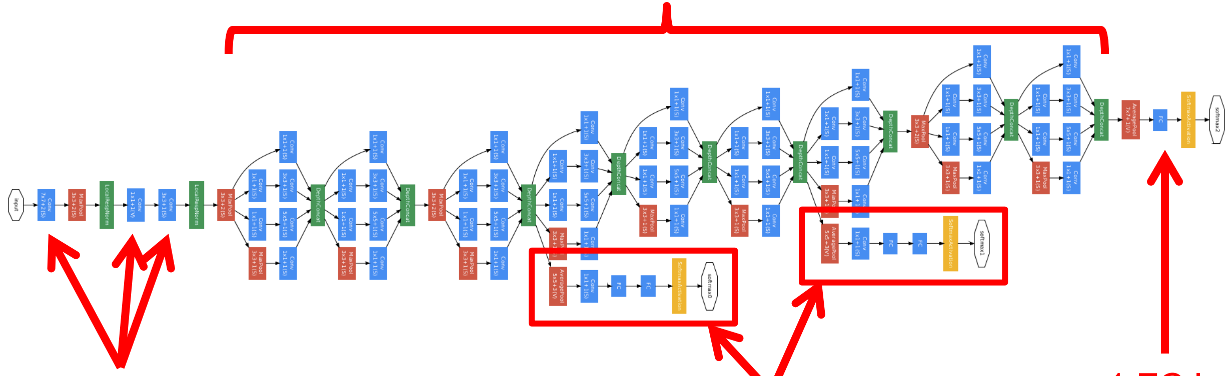
GoogLeNet/Inception (v1)

CONV Layers: 21 (depth), 57 (total)
Fully Connected Layers: 1
Weights: 7.0M
MACs: 1.43G

Also, v2, v3 and v4
ILSVRC14 Winner

[**Szegedy**, *CVPR* 2015]

9 Inception Blocks*



3 CONV layers

Auxiliary Classifiers
(helps with training,
not used during inference)

1 FC layer
(reduced from 3)

*referred to as inception **module** in textbook

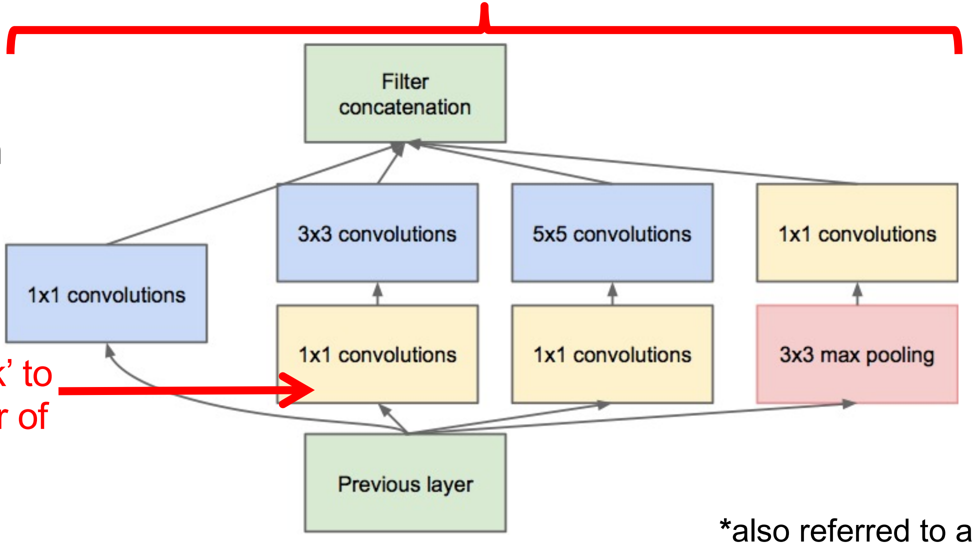
GoogLeNet/Inception (v1)

CONV Layers: 21 (depth), 57 (total)
Fully Connected Layers: 1
Weights: 7.0M
MACs: 1.43G

Also, v2, v3 and v4
ILSVRC14 Winner

[**Szegedy**, *CVPR* 2015]

Inception Block



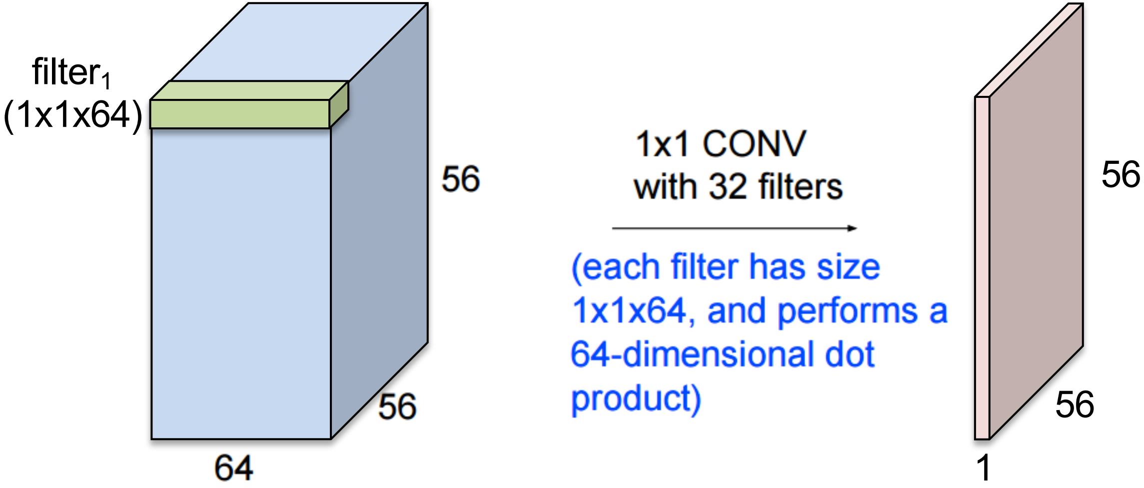
parallel* filters of different size have the effect of processing image at different scales

1x1 'bottleneck' to reduce number of weights and multiplications

*also referred to as "multi-branch" and "split-transform-merge"

1x1 Bottleneck

Use **1x1 filter** to capture cross-channel correlation, but not spatial correlation.
Can be used to reduce the number of channels in next layer (**compress**).
(Filter dimensions for bottleneck: **R**=1, **S**=1, **C** > **M**)

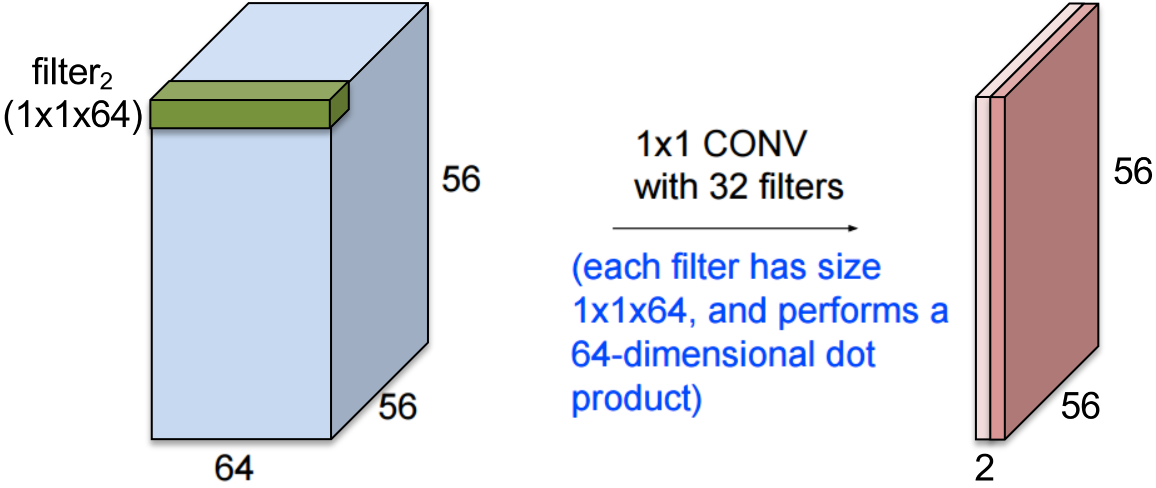


Modified image from source:
Stanford cs231n

[**Lin**, Network in Network, *ICLR* 2014]

1x1 Bottleneck

Use **1x1 filter** to capture cross-channel correlation, but not spatial correlation.
Can be used to reduce the number of channels in next layer (**compress**).
(Filter dimensions for bottleneck: **R**=1, **S**=1, **C** > **M**)

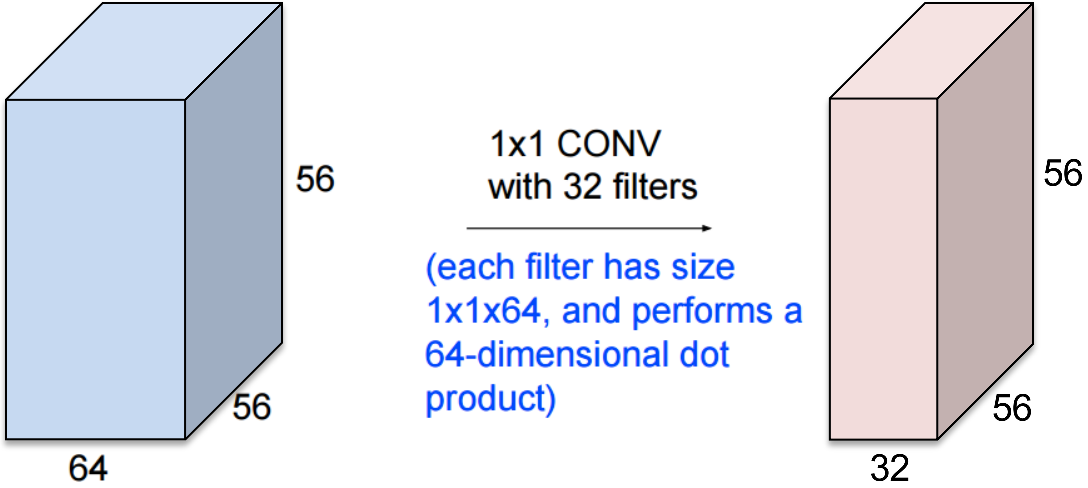


Modified image from source:
Stanford cs231n

[**Lin**, Network in Network, *ICLR* 2014]

1x1 Bottleneck

Use **1x1 filter** to capture cross-channel correlation, but not spatial correlation.
Can be used to reduce the number of channels in next layer (**compress**).
(Filter dimensions for bottleneck: **R**=1, **S**=1, **C** > **M**)



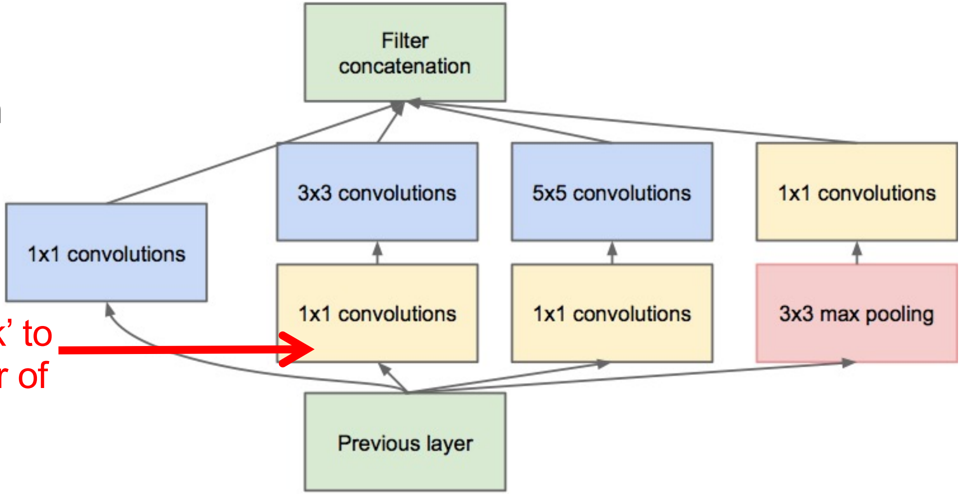
Modified image from source:
Stanford cs231n

[**Lin**, Network in Network, *ICLR* 2014]

GoogLeNet:1x1 Bottleneck

Apply 1x1 bottleneck before 'large' convolution filters.
Reduce weights such that **entire CNN can be trained on one GPU.**
Number of multiplications reduced from 854M → 358M

Inception Block



1x1 'bottleneck' to reduce number of weights and multiplications

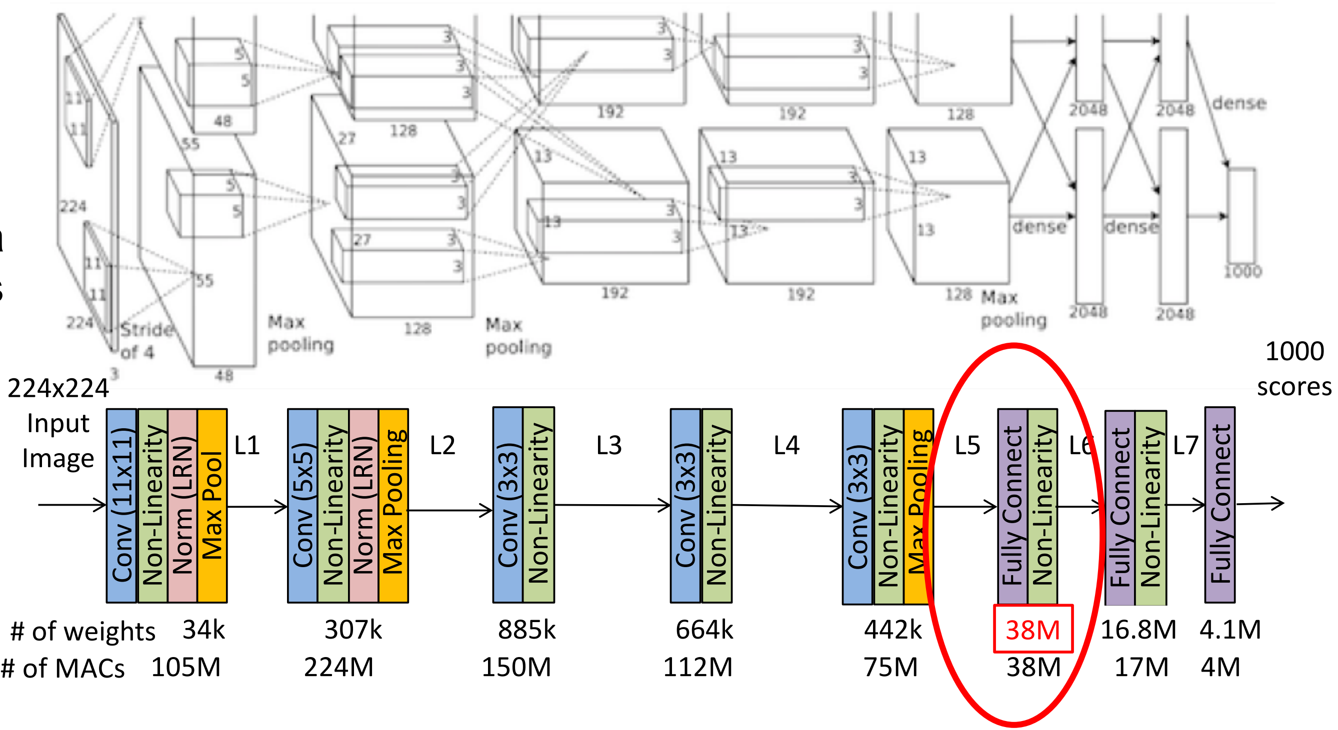
[**Szegedy**, *CVPR* 2015]

Reduce Cost of FC Layers

[**Krizhevsky**, *NeurIPS* 2012]

First FC layer accounts for a significant portion of weights

38M of 61M for AlexNet

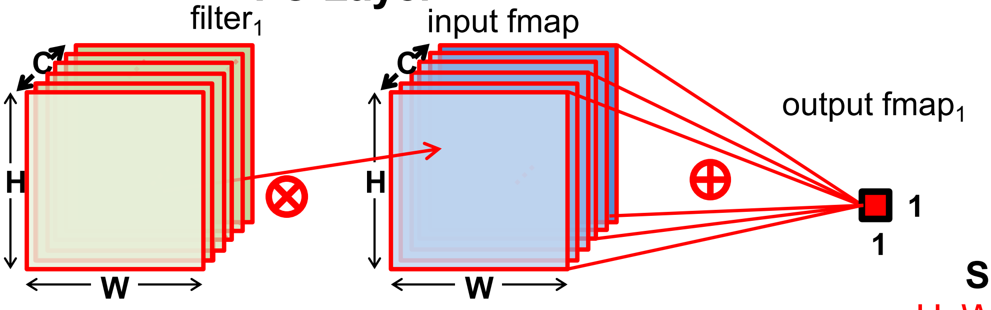


# of weights	34k	307k	885k	664k	442k	38M	16.8M	4.1M
# of MACs	105M	224M	150M	112M	75M	38M	17M	4M

Global Pooling

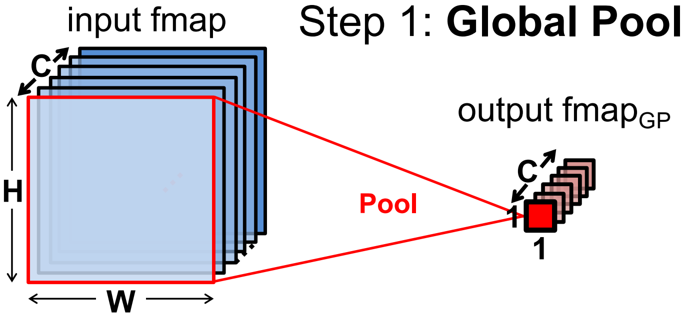
Use **Global Pooling** to reduce size of input to the **first FC layer** and the FC layer itself

FC Layer

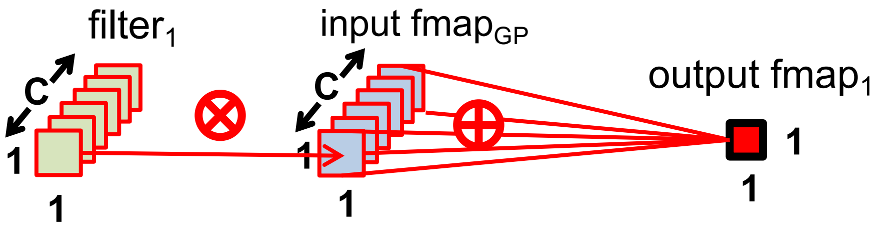


Size of FC layer:
HxWxCxM → 1x1xCxM

Step 1: **Global Pooling**



Step 2: **FC Layer**



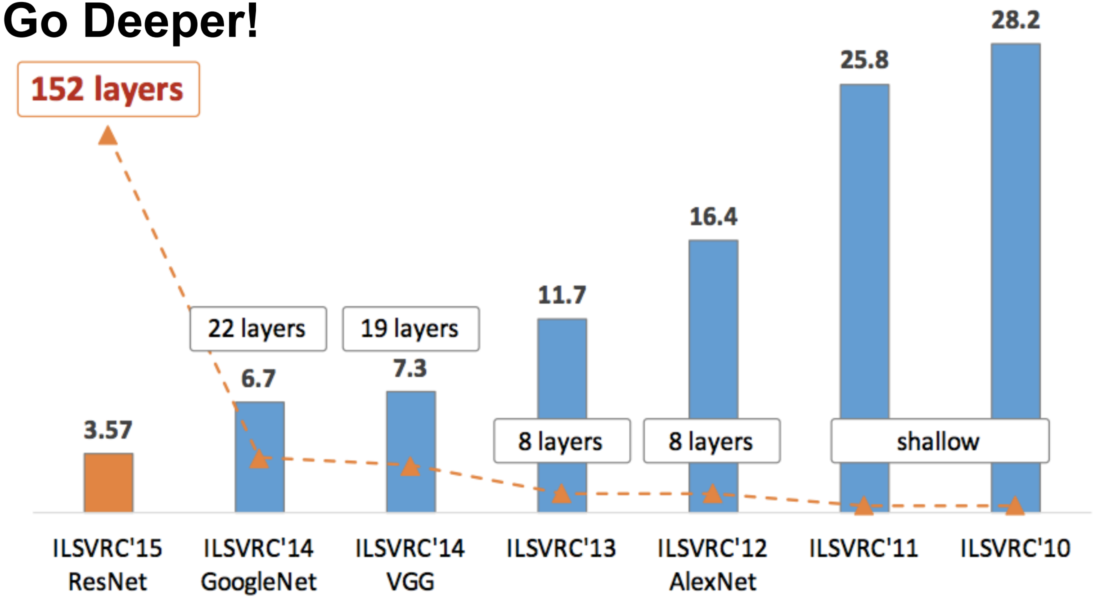
GoogLeNet uses global pooling to reduce number of FC layers from three to one

[**Lin**, *ICLR* 2014]

ResNet

ILSVRC15 Winner
(better than human level accuracy!)

Go Deeper!



ImageNet Classification top-5 error (%)

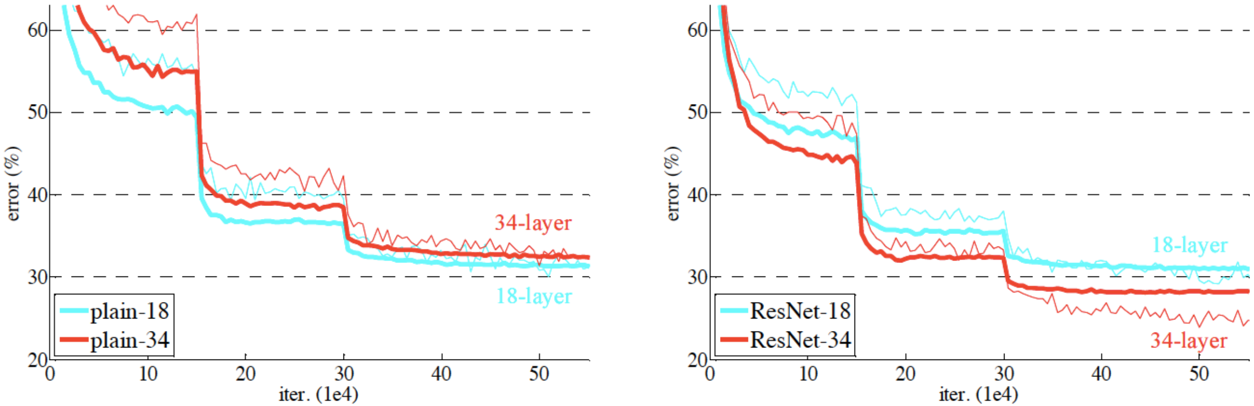
Image Source: http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

ResNet: Training

Training and validation error **increases** with more layers;
this is due to vanishing gradient, no overfitting.
Introduce **short cut block** to address this!

Without shortcut

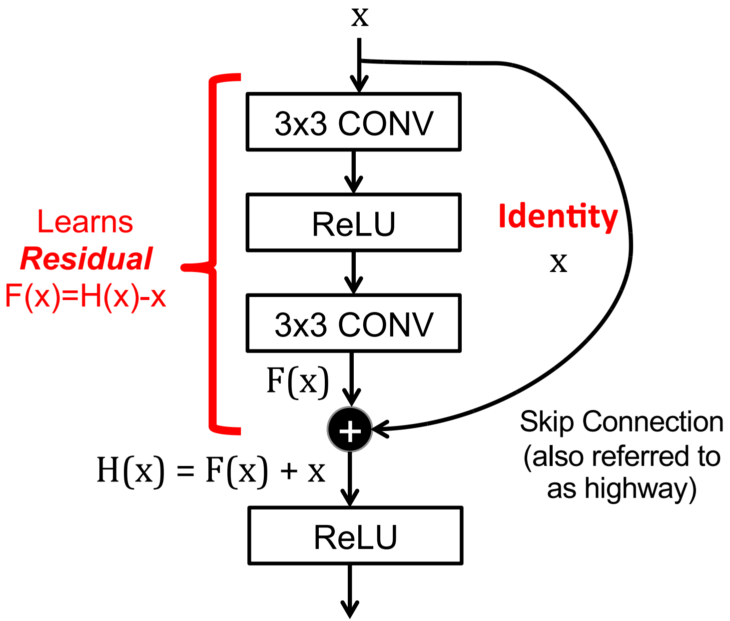
With shortcut



Thin curves denote training error, and bold curves denote validation error.

[**He**, *CVPR* 2016]

ResNet: Short Cut Block



x

3x3 CONV

ReLU

3x3 CONV

$F(x)$

Learns ***Residual*** $F(x)=H(x)-x$

Identity

x

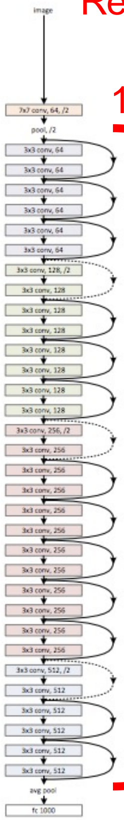
$H(x) = F(x) + x$

ReLU

Skip Connection
(also referred to
as highway)

Helps address the vanishing gradient challenge for training very deep networks

[**He**, *CVPR* 2016]



ResNet-34

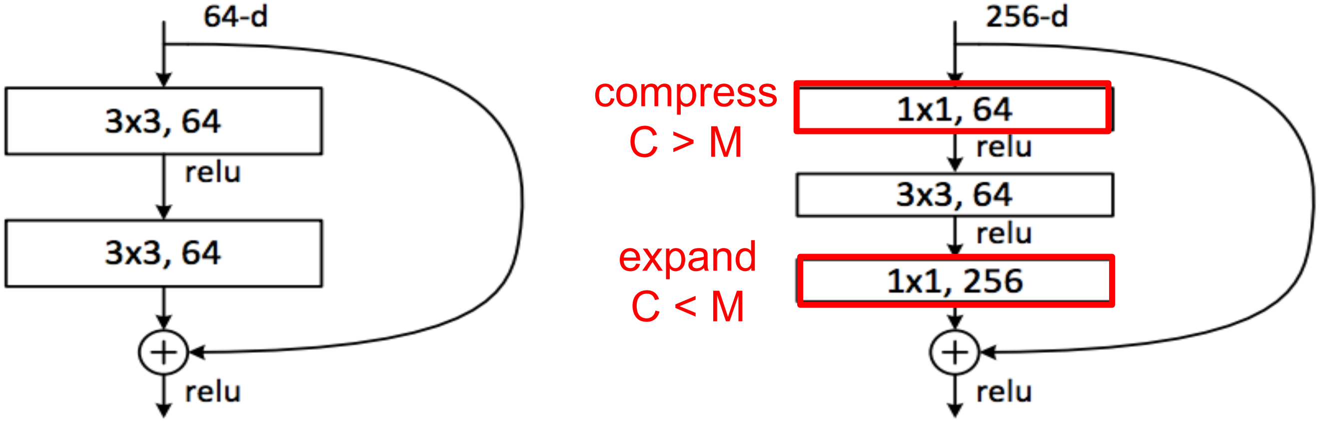
1 CONV layer

16 Short Cut Blocks

1 FC layer

ResNet: Bottleneck

Apply 1x1 bottleneck to reduce computation and size
Also makes network deeper (ResNet-34 → ResNet-50)



compress
$C > M$

expand
$C < M$

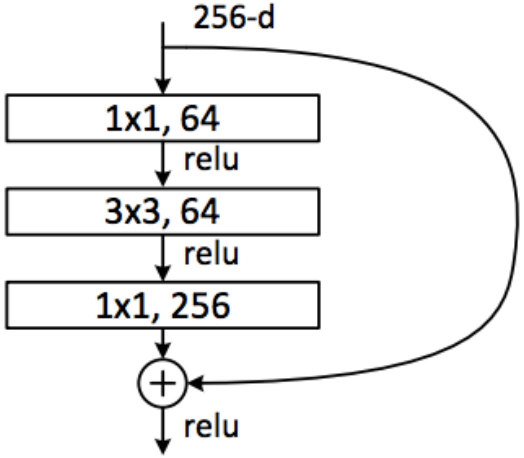
[**He**, *CVPR* 2016]

ResNet-50

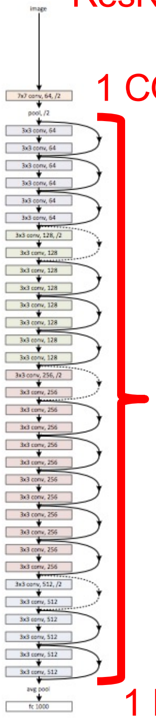
CONV Layers: 49
Fully Connected Layers: 1
Weights: 25.5M
MACs: 3.9G

Also, 34-, 152-, and 1202-layer versions
ILSVRC15 Winner

Short Cut Block



ResNet-50



1 CONV layer

16 Short Cut Blocks

1 FC layer

[**He**, *CVPR* 2016]

Summary of Popular CNNs

Metrics	LeNet-5	AlexNet	VGG-16	GoogLeNet (v1)	ResNet-50
Top-5 error	n/a	16.4	7.4	6.7	5.3
Input Size	28x28	227x227	224x224	224x224	224x224
# of CONV Layers	**2**	**5**	**16**	**21 (depth)**	**49**
Filter Sizes	5	3, 5,11	3	1, 3 , 5, 7	1, 3, 7
# of Channels	1, 6	3 - 256	3 - 512	3 - 1024	3 - 2048
# of Filters	6, 16	96 - 384	64 - 512	64 - 384	64 - 2048
Stride	1	1, 4	1	1, 2	1, 2
# of Weights	2.6k	2.3M	14.7M	6.0M	23.5M
# of MACs	283k	666M	15.3G	1.43G	3.86G
# of FC layers	**2**	**3**	**3**	**1**	**1**
# of Weights	58k	58.6M	124M	1M	2M
# of MACs	58k	58.6M	124M	1M	2M
Total Weights	**60k**	**61M**	**138M**	**7M**	**25.5M**
Total MACs	**341k**	**724M**	**15.5G**	**1.43G**	**3.9G**

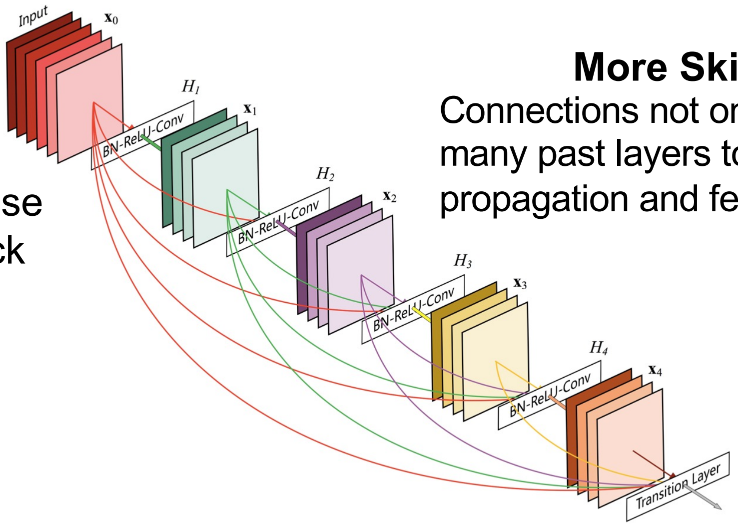
CONV Layers increasingly important!

Summary of Popular CNNs

- **AlexNet**
  - First CNN Winner of ILSVRC
  - Uses LRN (deprecated after this)
- **VGG-16**
  - Goes Deeper (16+ layers)
  - Uses only 3x3 filters (stack for larger filters)
- **GoogLeNet (v1)**
  - Reduces weights with Inception and uses Global Pooling so that only one FC layer is needed
  - Inception Block: 1x1 and parallel connections
  - Batch Normalization
- **ResNet**
  - Goes Deeper (24+ layers)
  - Short cut Block: Skip connections

DenseNet

Dense Block

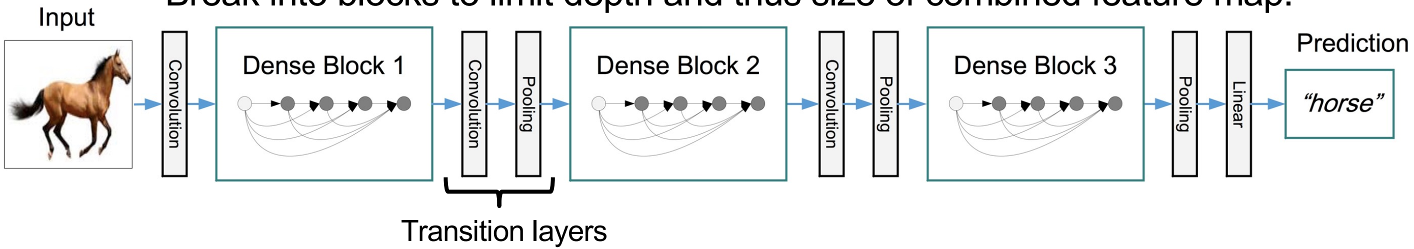


More Skip Connections!

Connections not only from previous layer, but many past layers to strengthen feature map propagation and feature reuse.

Feature maps are concatenated rather than added.

Break into blocks to limit depth and thus size of combined feature map.


Input

Convolution

Dense Block 1

Convolution

Pooling

Dense Block 2

Convolution

Pooling

Dense Block 3

Pooling

Linear

Prediction

"horse"

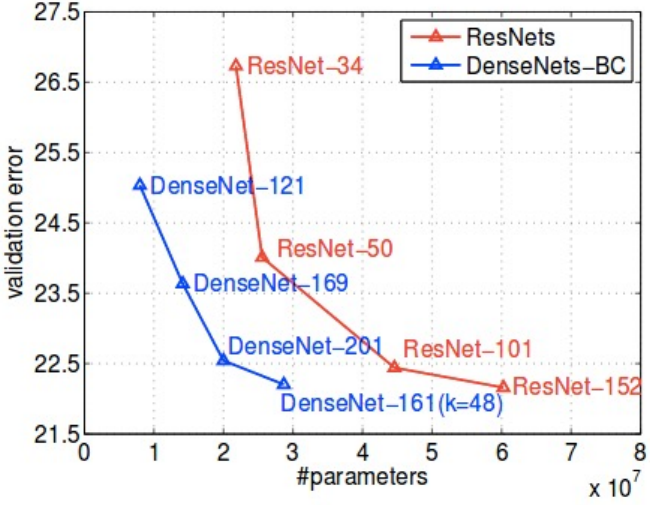
Transition layers

[Huang, *CVPR* 2017]

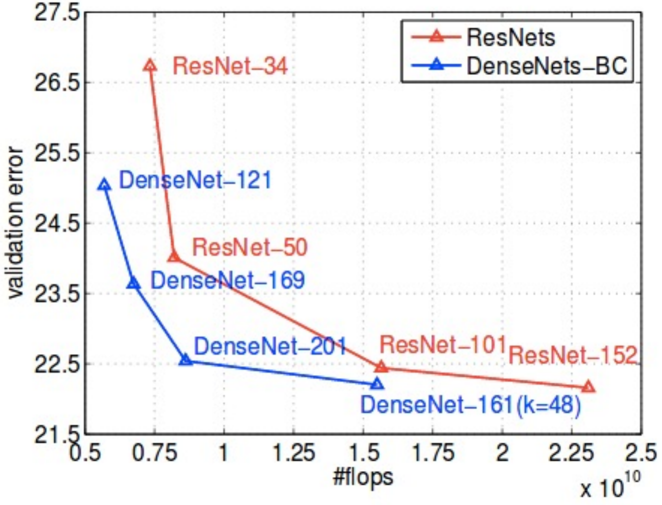
DenseNet

Higher accuracy than ResNet with fewer weights and multiplications

Top-1 error



Top-1 error



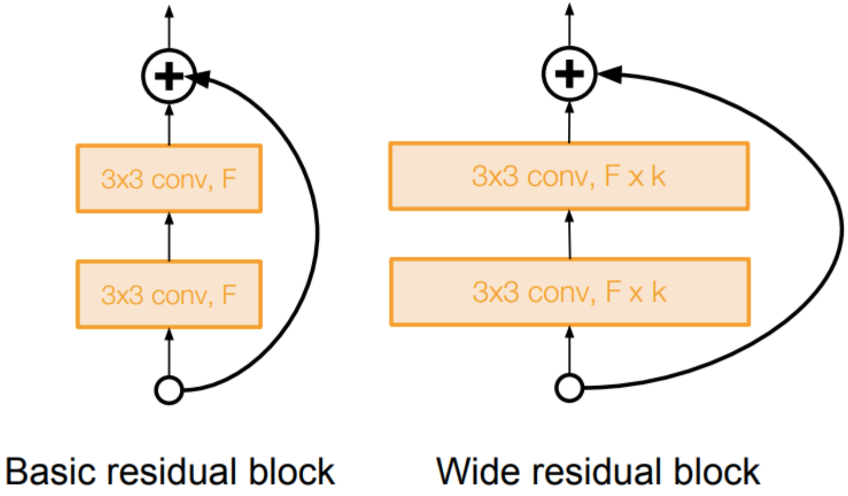
Note: 1 MAC = 2 FLOPS

[**Huang**, *CVPR* 2017]

Wide ResNet

Increase width (# of filters) rather than depth of network

- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth is also more parallel-friendly



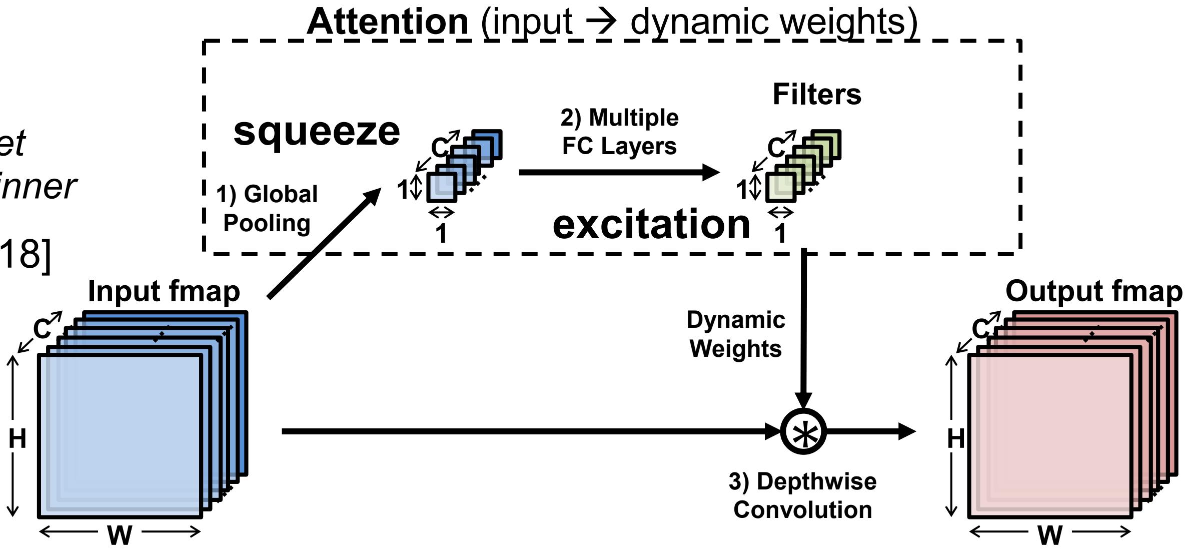
[**Zagoruyko**, *BMVC* 2016]

Image Source: Stanford cs231n

Squeeze and Excitation

Used by SENet
ILSVRC 2017 Winner

[**Hu**, *CVPR* 2018]



Depth-wise convolution with **dynamic weights**, where the weights change based on the input feature map.

- **Squeeze**: Summarize each channel of input features map with global pooling
- **Excitation**: Determine weights using FC layers to increase **attention** on certain channels of the input features map

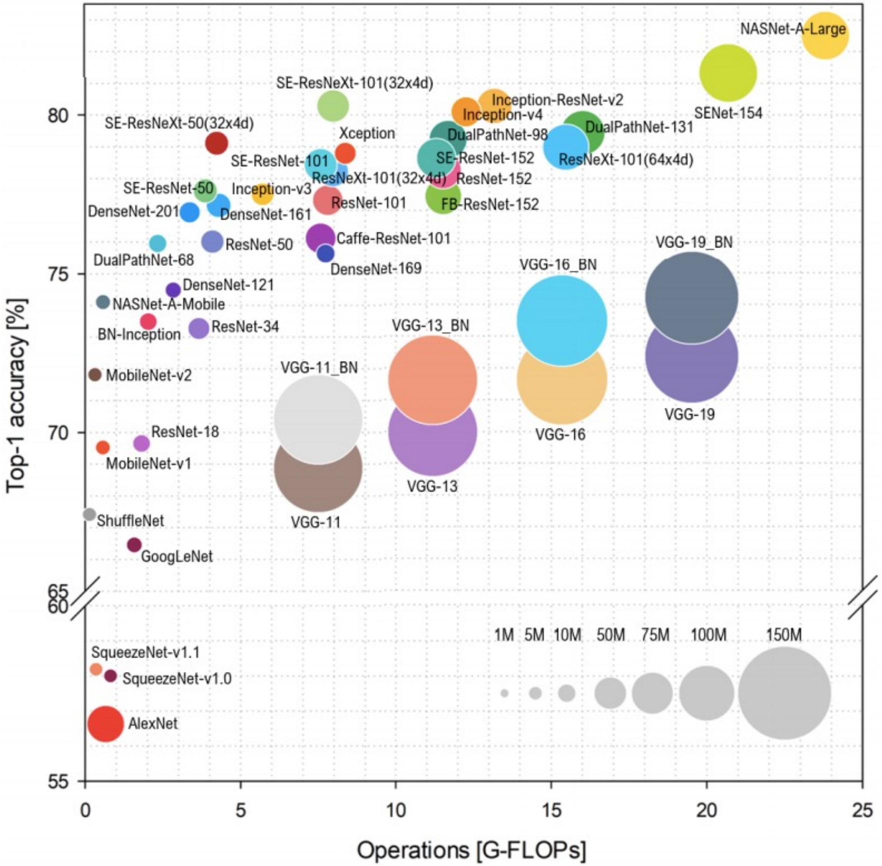
Convolution versus Attention Mechanism

- **Convolution**
  - Only models dependencies between spatial neighbors
  - Use sparsely connected layer to spatial neighbors; no support for dependencies outside of spatial dimensions of filter (R x S)
- **Attention**
  - "Allows modeling of [global] dependencies **without regard to their distance**" [**Vaswani**, *NeurIPS* 2017]
  - However, fully connected layer too expensive; develop mechanism to bias "the allocation of available **computational resources towards the most informative** components of a signal" [**Hu**, *CVPR* 2018]

- **Transformer** is a type of DNN that is built entirely using Attention Mechanism [**Vaswani**, *NeurIPS* 2017] (Next Lecture)

Efficient CNN Models

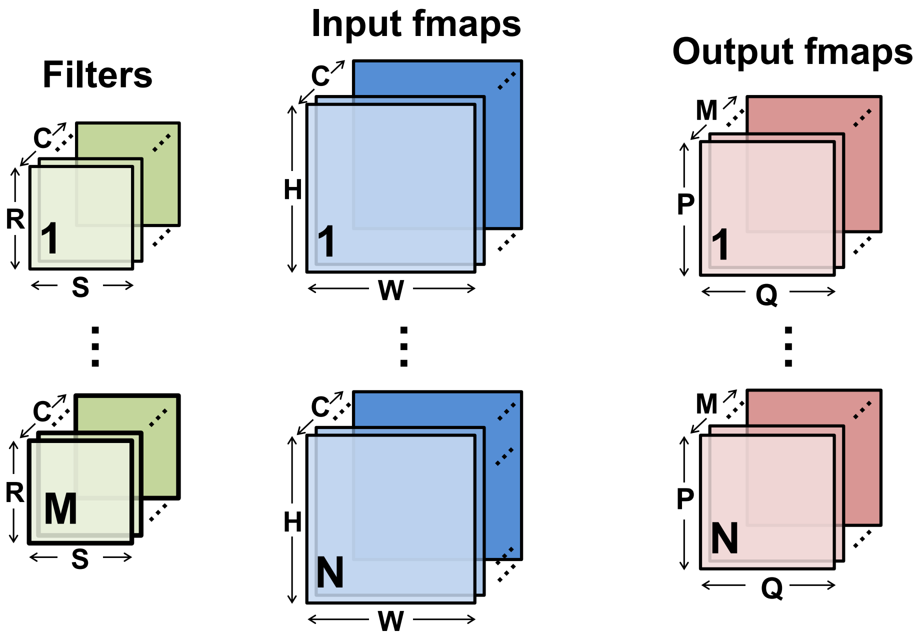
Accuracy vs. Weight & OPs



[**Bianco**, *IEEE Access* 2018]

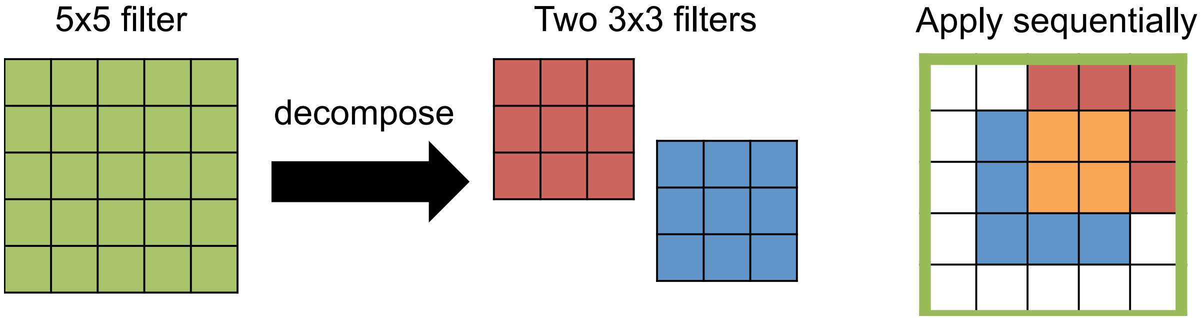
Manual Network Design

- **Reduce Spatial Size (R, S)**
  - stacked filters
- **Reduce Channels (C)**
  - 1x1 convolution, grouped convolution
- **Reduce Filters (M)**
  - feature map reuse across layers

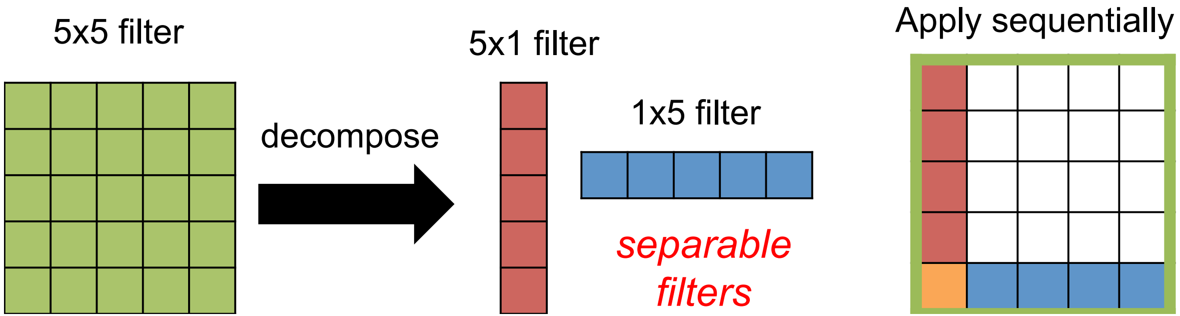


Reduce Spatial Size (R, S): Stacked Small Filters

VGG



GoogleNet/ Inception v3



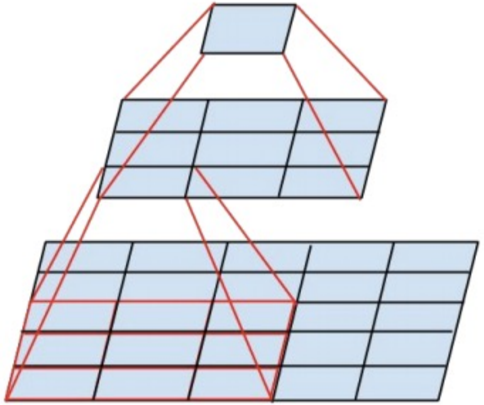
separable filters

Replace a large filter with **a series of smaller filters** (reduces degrees of freedom)

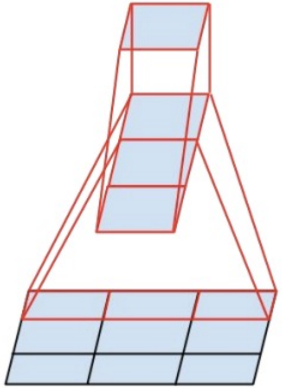
Example: Inception V3

Go deeper **(v1: 22 layers → v3: 40+ layers)** by reducing the number of weights per filter using **filter decomposition**
~3.5% higher accuracy than v1

5x5 filter → 3x3 filters



3x3 filter → 3x1 and 1x3 filters

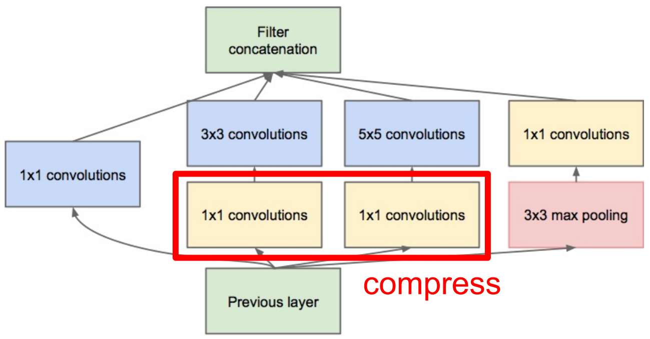


Separable filters

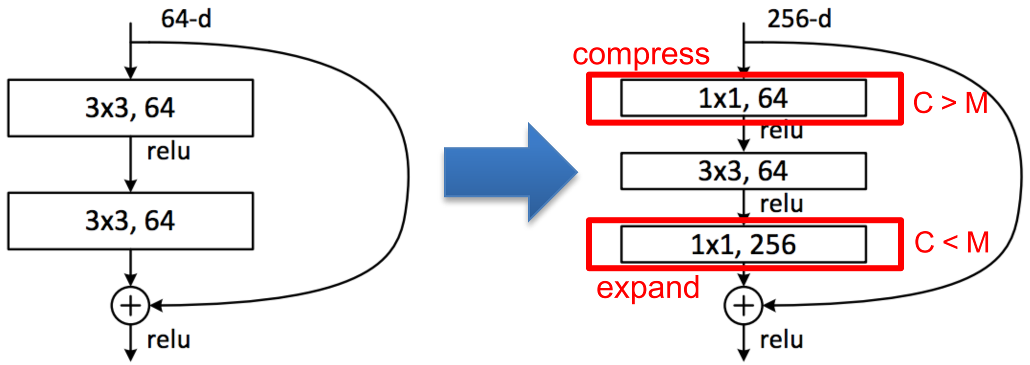
[**Szegedy**, *CVPR* 2016]

Reduce Channels (C): 1x1 Convolution

GoogLeNet

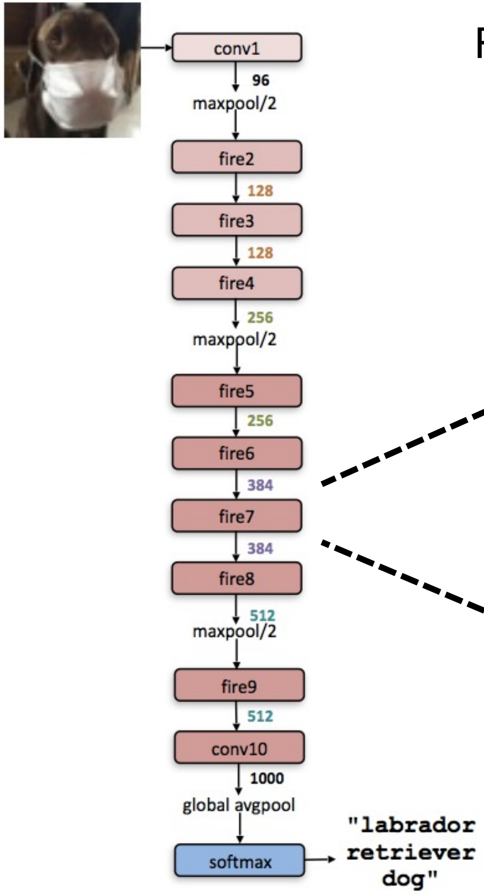

Filter concatenation
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer
compress

ResNet


64-d
3x3, 64
relu
3x3, 64
relu
256-d
compress
1x1, 64
C > M
relu
3x3, 64
relu
1x1, 256
C < M
expand
relu

- Use **1x1 (bottleneck) filter** to capture cross-channel correlation, but not spatial correlation
- Reduce the number of channels in next layer (**compress**), where **C** > **M**

Example: SqueezeNet

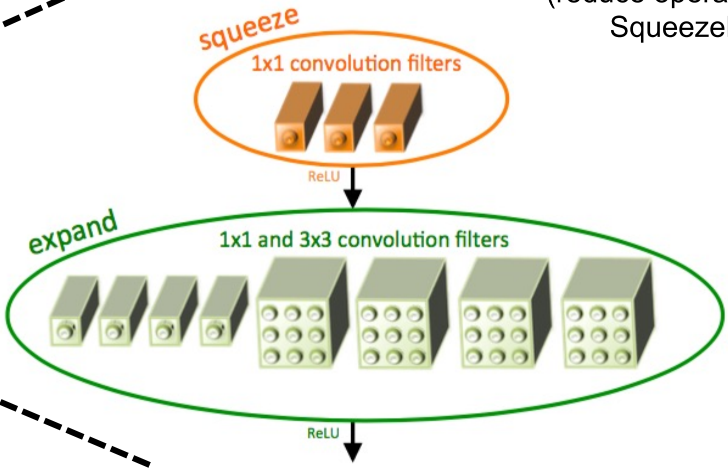


Reduce number of weights by reducing number of input channels by “squeezing” with 1x1

50x fewer weights than AlexNet (no accuracy loss)

However, 1.2x more operations than AlexNet*

Fire Block



*for SqueezeNetv1.0 (reduce operations by 2x in SqueezeNetv1.1)

[Iandola, *ICLR* 2017]

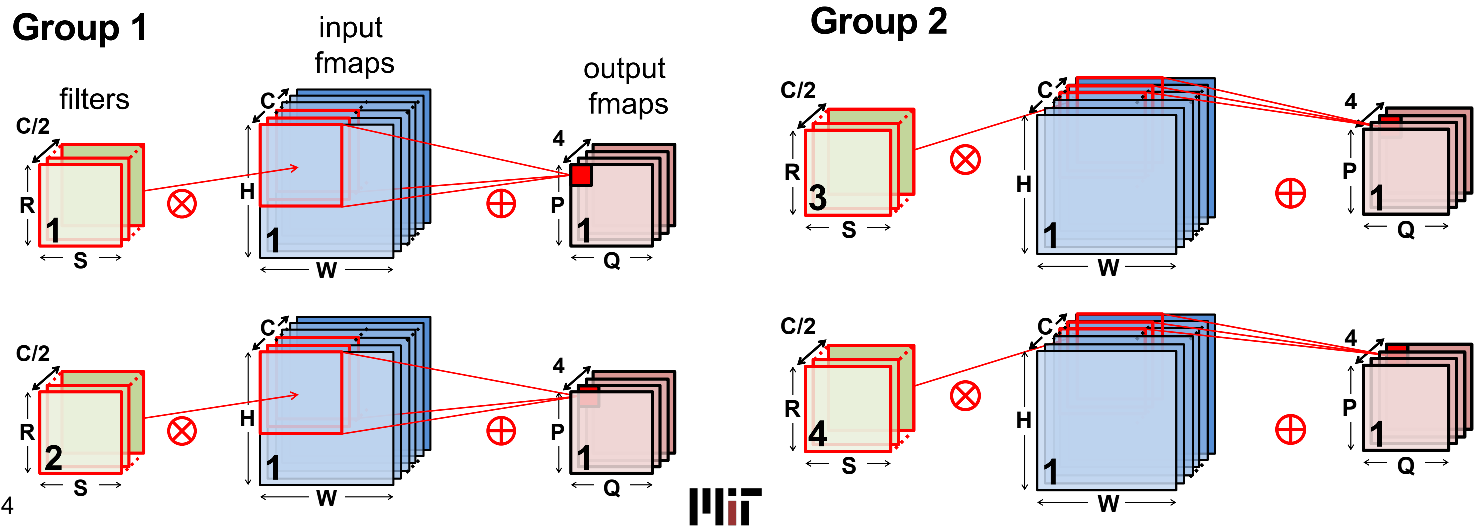
Reduce Channels (C): Grouped Convolutions

Grouped convolutions reduce the number of **weights and multiplications** at the cost of not sharing information between **groups**

- **Divide** filters into groups (**G**) operating on **subset** of channels.
- Each group has **M/G** filters and processes **C/G** channels.

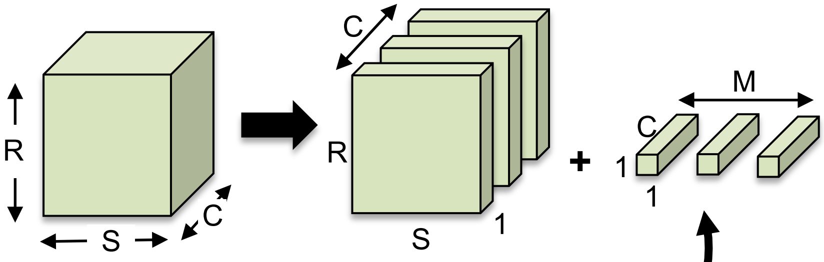
Example for **G=2**: Each filter requires **2x fewer weights and MACs (C → C/2)**

In this example, N=1 & M=4



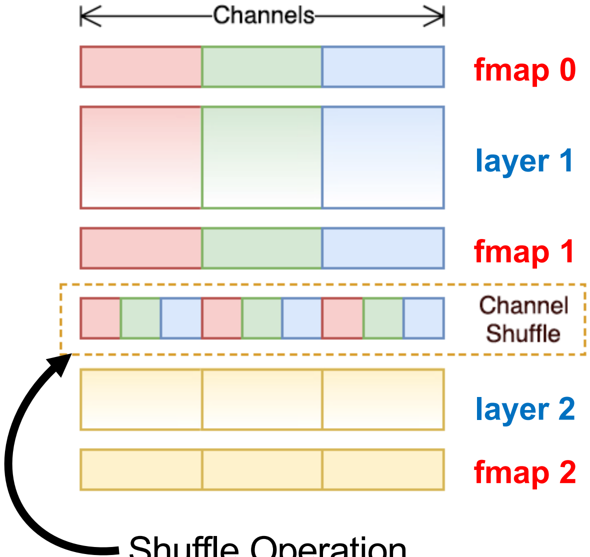
Reduce Channels (C): Grouped Convolutions

Two ways of mixing information from groups



Pointwise (1x1) Convolution
(**Mix in one step**)
MobileNet

Also referred to as **depth-wise separable**:
Decouple the cross-channels correlations and
spatial correlations in the feature maps of the CNN

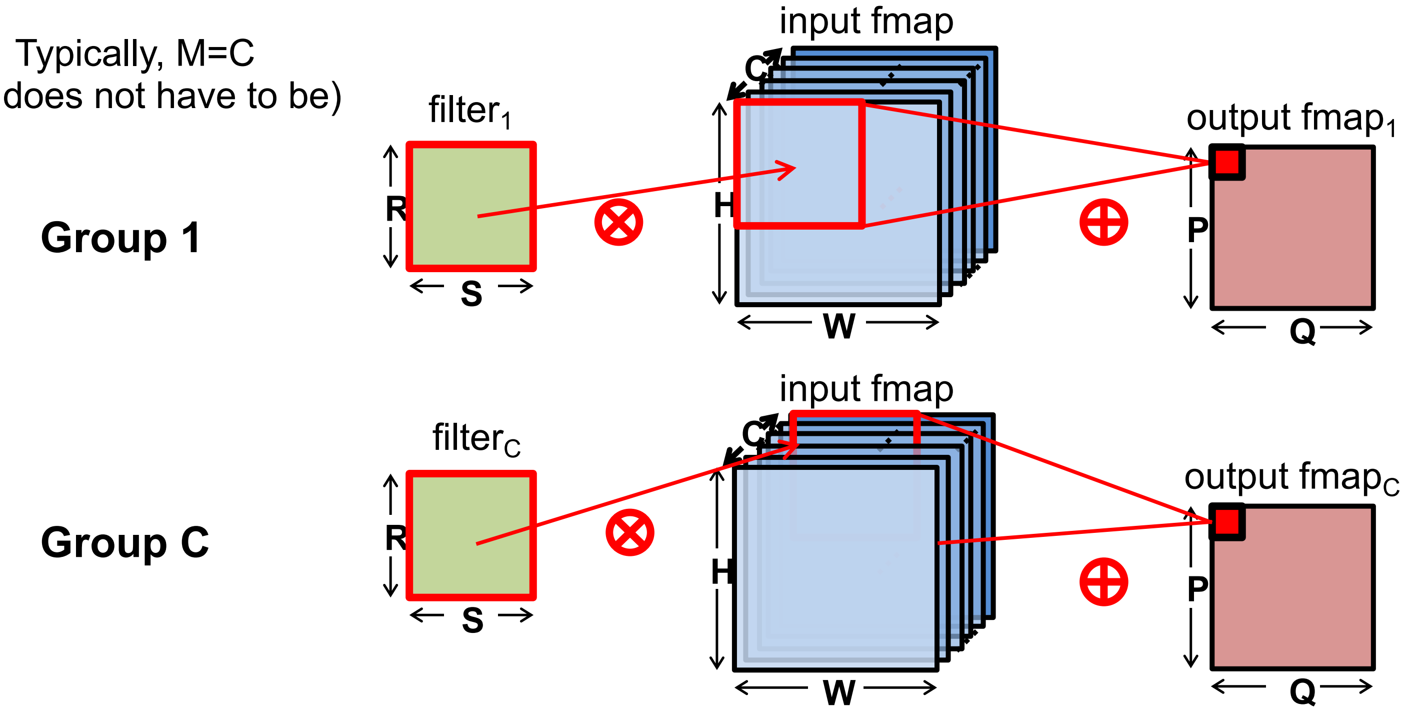


Shuffle Operation
(**Mix in multiple steps**)
ShuffleNet

Depth-wise Convolutions

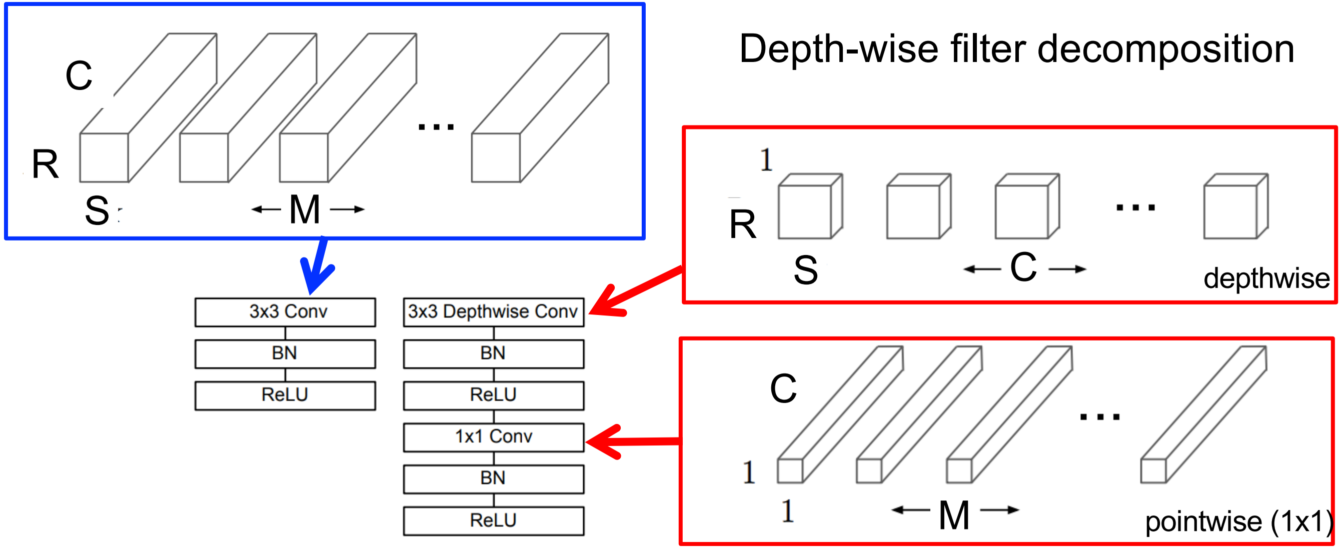
The extreme case of **Grouped Convolutions** is **Depth-wise Convolutions,** where the **number of groups (G) equals number channels (C)** (i.e., one input channel per group)

Typically, M=C
(but does not have to be)



Example: MobileNets

Depth-wise filter decomposition



Reduction in MACs

$$\frac{HWCRSM}{HWC(RS+M)} = \frac{RSM}{(RS+M)}$$

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

[**Howard**, *arXiv* 2017]

MobileNets: Comparison

Comparison with other CNN Models

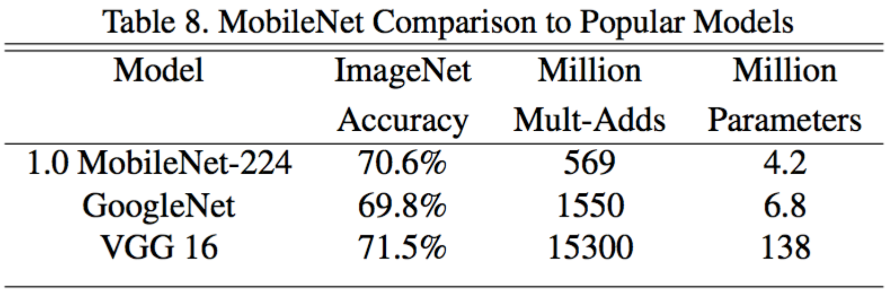


Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

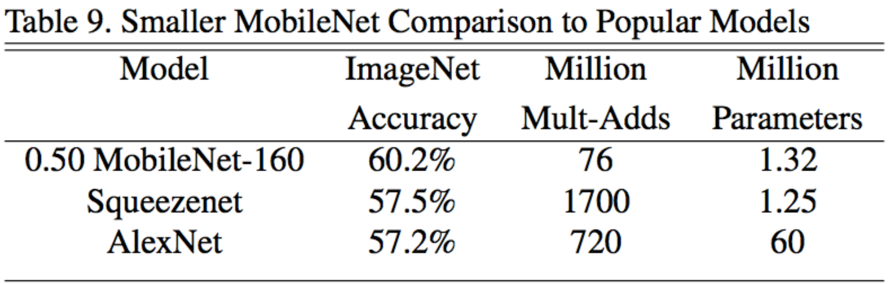
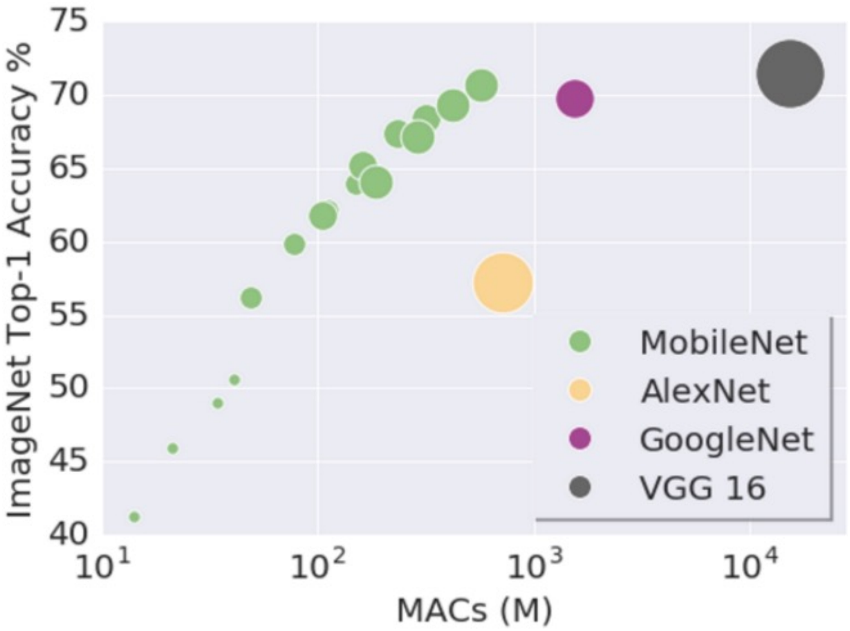


Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

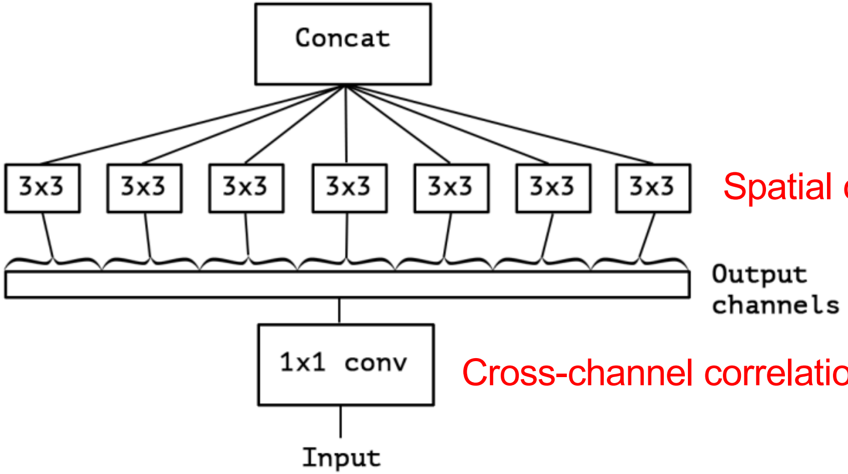


[Image source: Github]

[**Howard**, *arXiv* 2017]

Example: Xception

- An Inception block based on depth-wise separable convolutions
- Claims to learn richer features with similar number of weights as Inception V3 (i.e., more efficient use of weights)
  - Similar performance on ImageNet; 4.3% better on larger dataset (JFT)
  - However, 1.5x more operations required than Inception V3



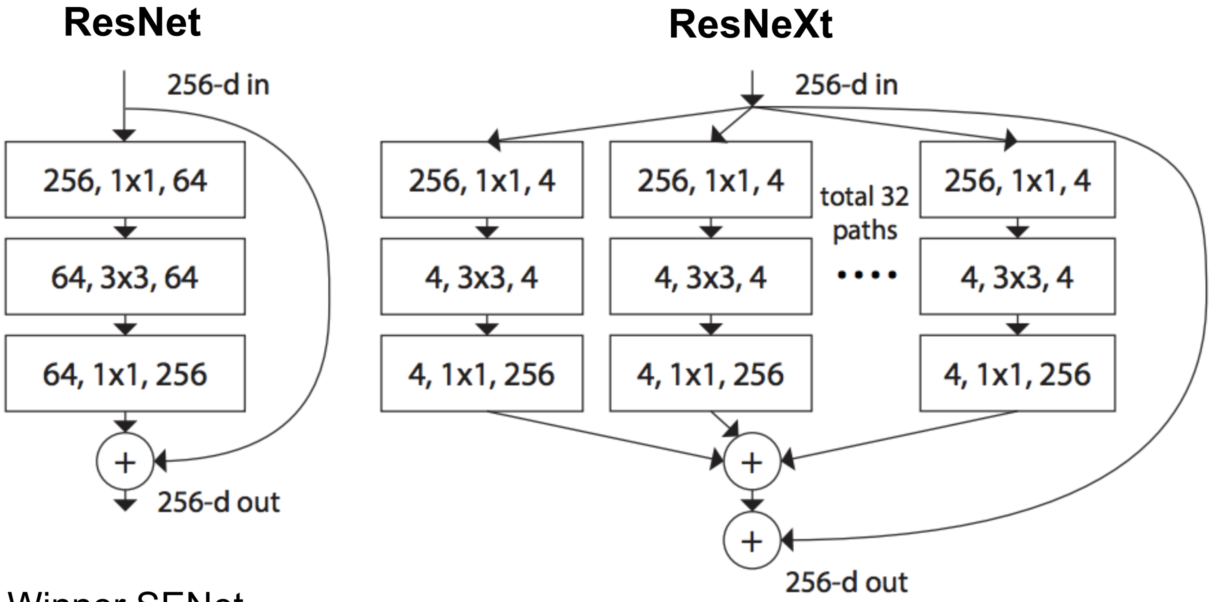
Spatial correlation

Cross-channel correlation

[**Chollet**, *CVPR* 2017]

Example: ResNeXt

Increase number of **convolution groups** (**G**) (referred to as *cardinality* in the paper) instead of depth and width of network

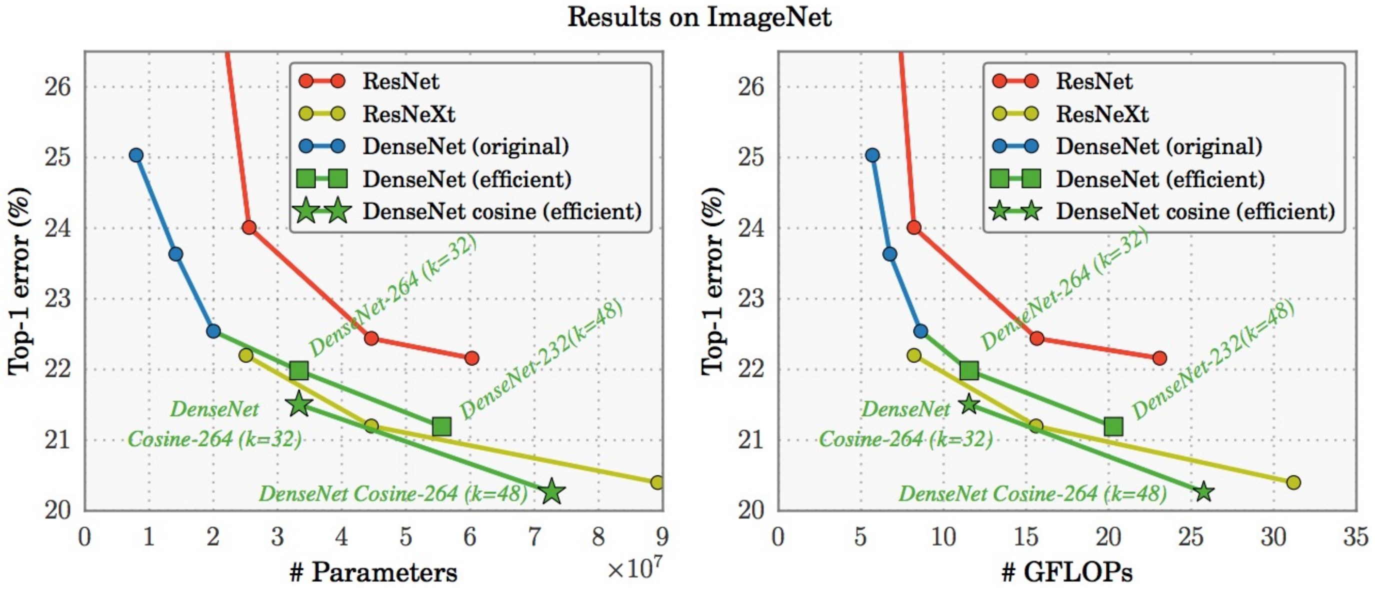


Used by ILSVRC 2017 Winner SENet
Inspired by Inception's "split-transform-merge"

[**Xie**, *CVPR* 2017]

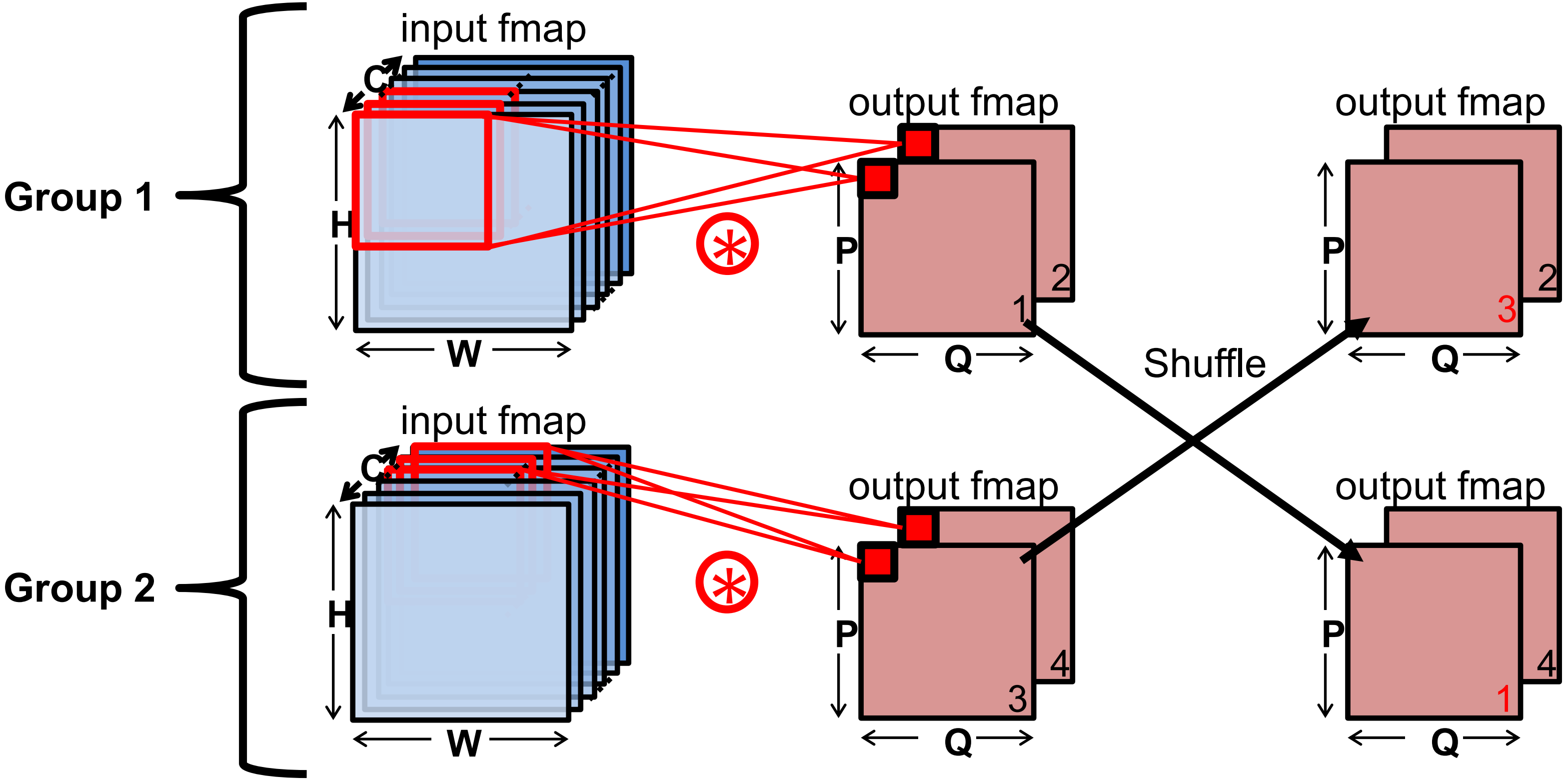
Example: ResNeXt

Improved accuracy vs. ‘complexity’ tradeoff compared to other ResNet based models



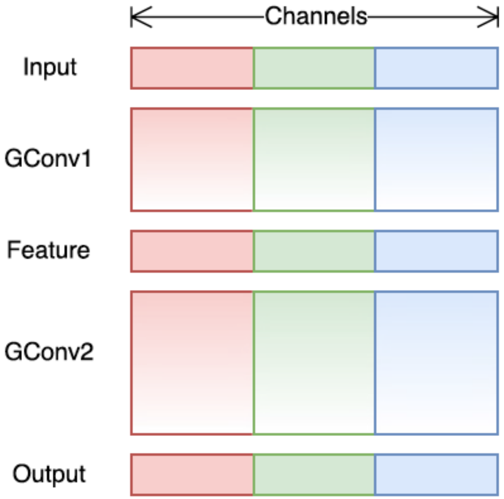
[Xie, *CVPR* 2017]

Shuffle Operation

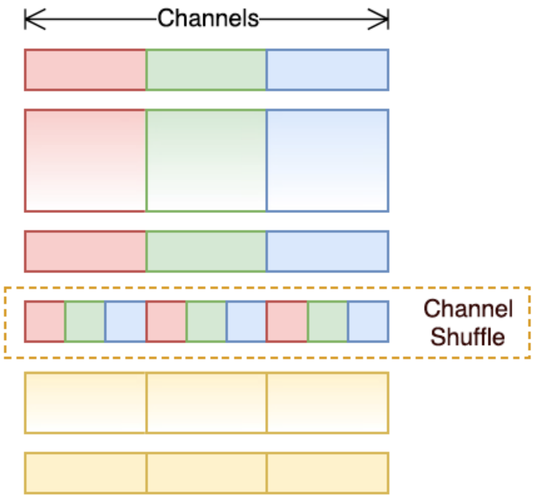


Example: ShuffleNet

Shuffle order such that channels are not isolated across groups
(up to 4% increase in accuracy)



No interaction between
channels from different groups



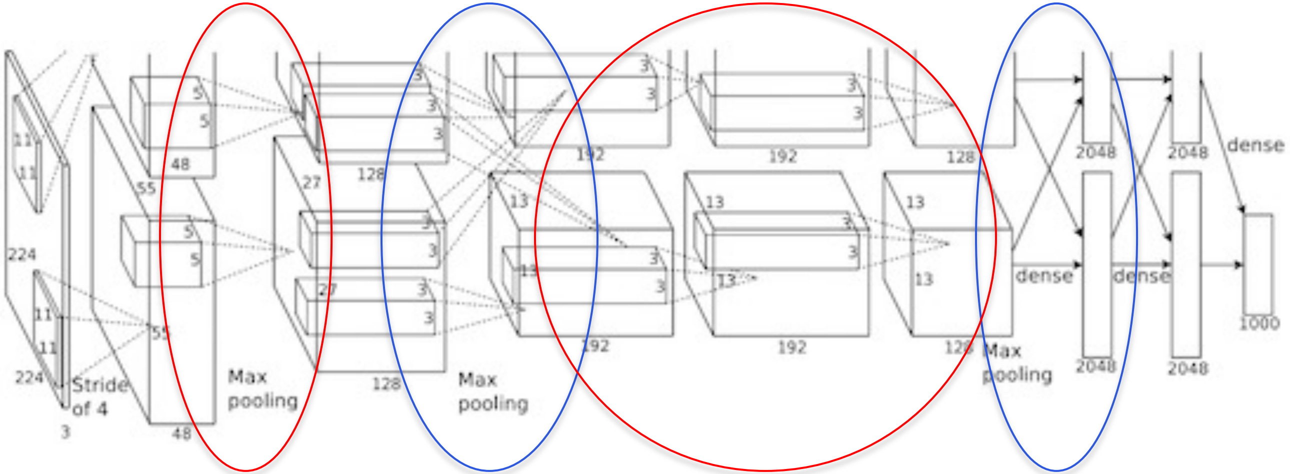
Shuffling allow interaction between
channels from different groups

[**Zhang**, *CVPR* 2018]

AlexNet: Grouped Convolutions

AlexNet uses grouped convolutions to train on two separate GPUs

(Drawback: correlation between channels of different groups is not used)



Split into 2 Groups

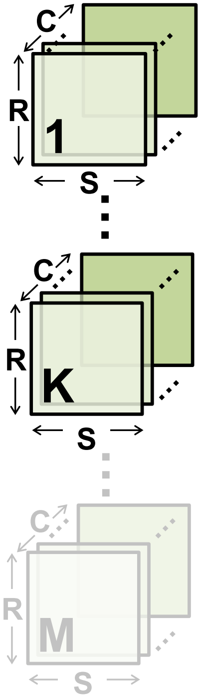
Mix Information (3x3 CONV)

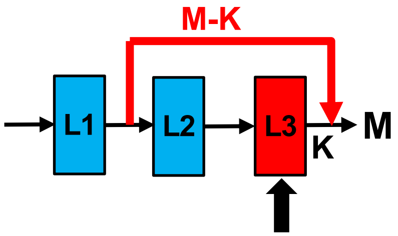
Split into 2 Groups

Mix Information (FC)

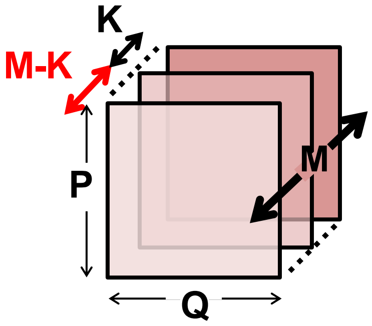
Reduce Filters (M): Feature Map Reuse

M Filters



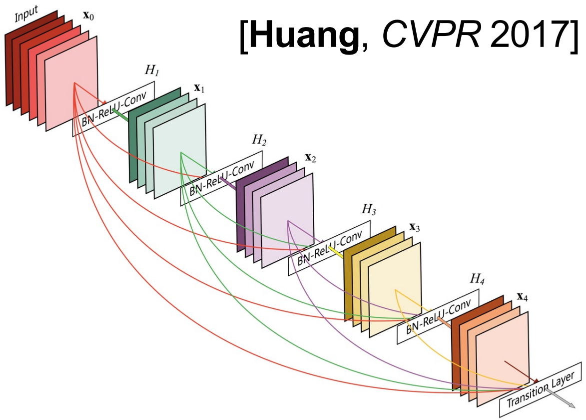


Output fmap with M channels



Reuse **(M-K)** channels in feature maps from previously processed layers

DenseNet reuses feature map from multiple layers



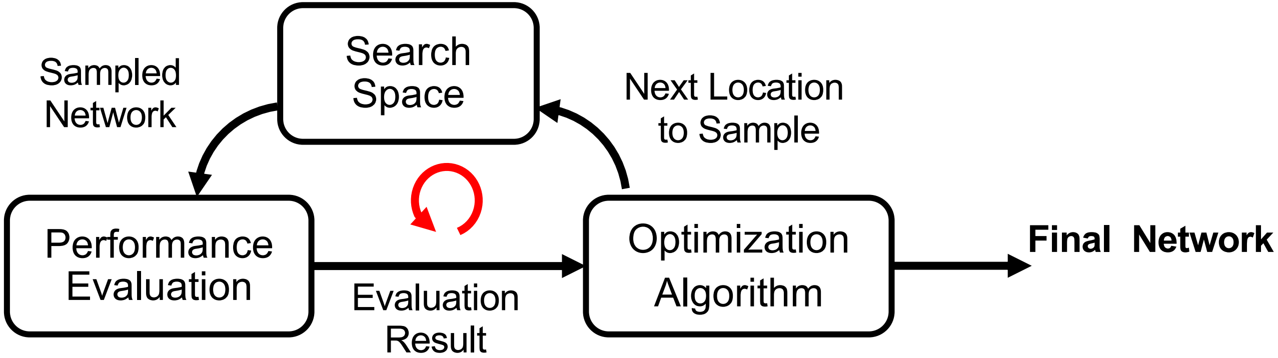
[**Huang**, *CVPR* 2017]

Neural Architecture Search (NAS)

Rather than handcrafting the model, automatically search for it

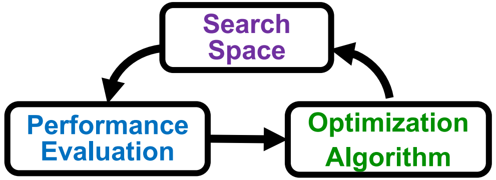
Neural Architecture Search (NAS)

- Three main components:
  - Search Space (what is the set of all samples)
  - Optimization Algorithm (where to sample)
  - Performance Evaluation (how to evaluate samples)



Key Metrics: Achievable DNN **accuracy** and required **search time**

Evaluate NAS Search Time



$$time_{nas} = num_{samples} \times time_{sample}$$

$$time_{nas} \propto \left(size_{search\_space} \times \frac{num_{alg\_tuning}}{efficiency_{alg}} \right) \times (time_{eval} + time_{train})$$

(1) Shrink the search space

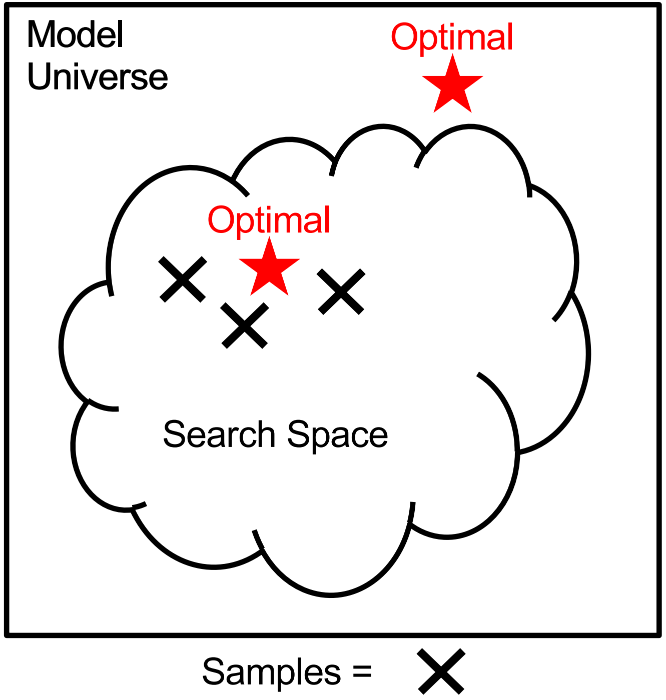
(2) Improve the optimization algorithm

(3) Simplify the performance evaluation

Goal: Improve the efficiency of NAS in the three main components

(1) Shrink the Search Space

- Trade the breadth of models for search speed
- May limit the performance that can be achieved
- Use domain knowledge from manual network design to help guide the reduction of the search space



(1) Shrink the Search Space

- **Search space = layer operations + connections between layers**

Common layer operations

- Identity
- 1x3 then 3x1 convolution
- 1x7 then 7x1 convolution
- 3x3 dilated convolution
- 1x1 convolution
- 3x3 convolution
- 3x3 separable convolution
- 5x5 separable convolution
- 3x3 average pooling
- 3x3 max pooling
- 5x5 max pooling
- 7x7 max pooling

[**Zoph**, *CVPR* 2018]

(1) Shrink the Search Space

- **Search space = layer operations + connections between layers**

Smaller Search Space

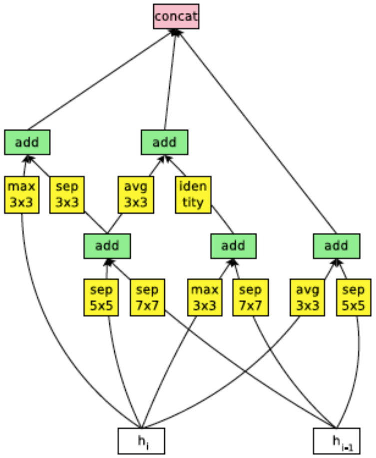
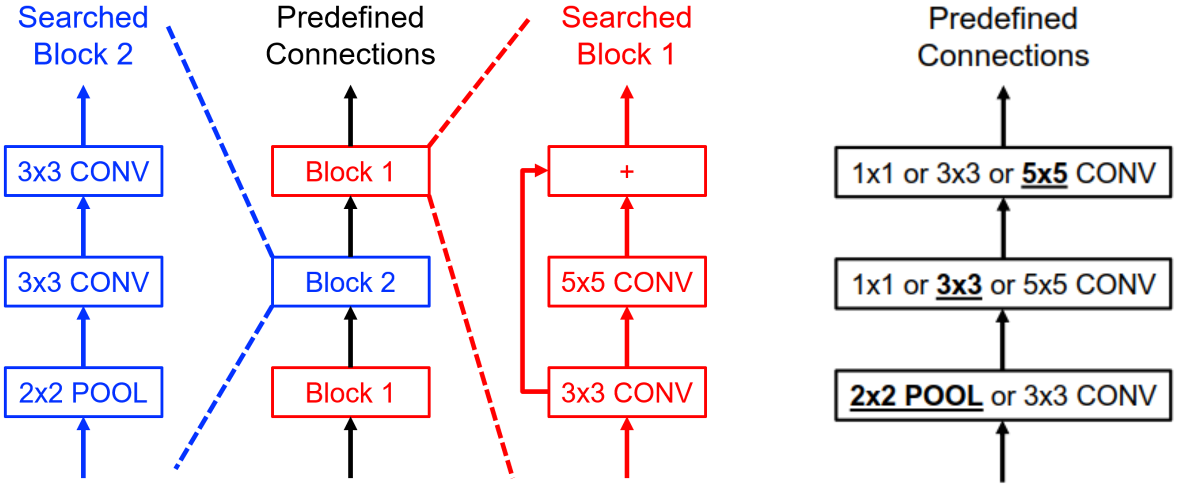


Image Source: [**Zoph**, CVPR 2018]



Searched Block 2

3x3 CONV

3x3 CONV

2x2 POOL

Predefined Connections

Block 1

Block 2

Block 1

Searched Block 1

+

5x5 CONV

3x3 CONV

Predefined Connections

1x1 or 3x3 or **5x5** CONV

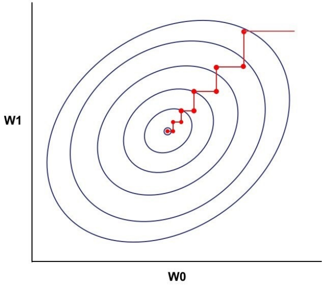
1x1 or **3x3** or 5x5 CONV

2x2 POOL or 3x3 CONV

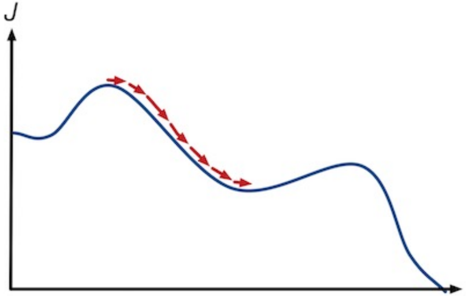
(2) Improve Optimization Algorithm

Random

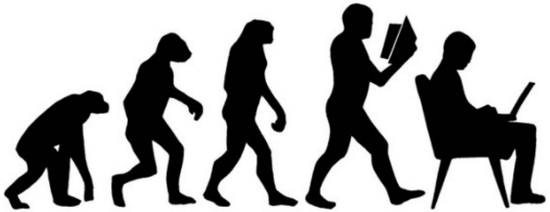
Coordinate Descent



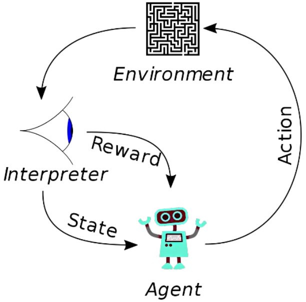
Gradient Descent



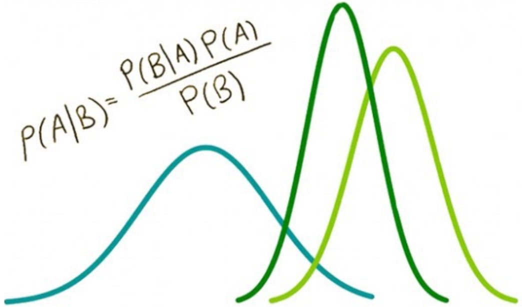
Evolutionary



Reinforcement Learning



Bayesian

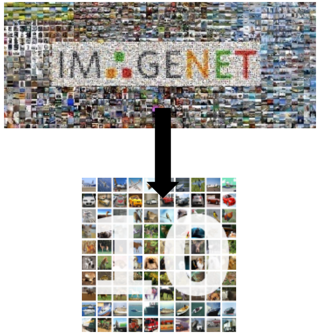


(3) Simplify the Performance Evaluation

- NAS needs only the rank of the performance values
- Method 1: approximate accuracy

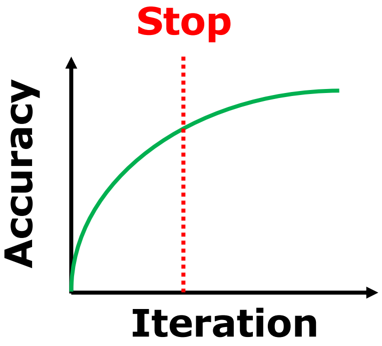
Proxy Task

E.g., Smaller resolution, simpler tasks



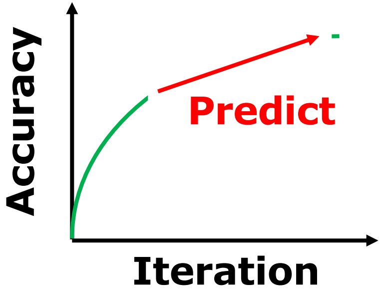
Early Termination

Stop training earlier



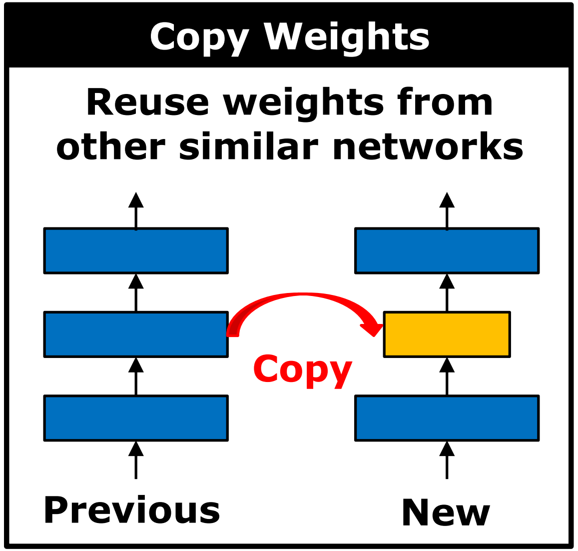
Accuracy Prediction

Extrapolate accuracy

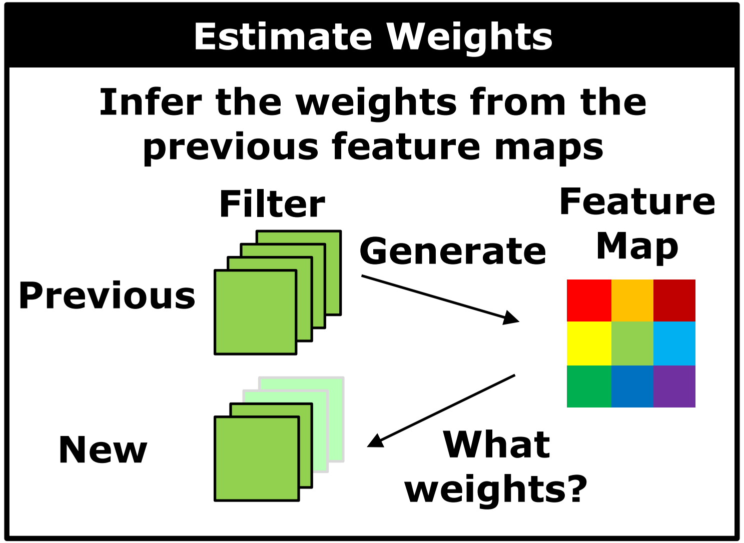


(3) Simplify the Performance Evaluation

- NAS needs only the rank of the performance values
- Method 2: approximate weights


Copy Weights

Reuse weights from other similar networks


Estimate Weights

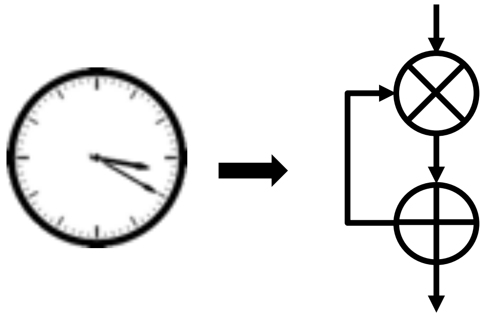
Infer the weights from the previous feature maps

(3) Simplify the Performance Evaluation

- NAS needs only the rank of the performance values
- Method 3: approximate metrics (e.g., latency, energy)

Proxy Metric

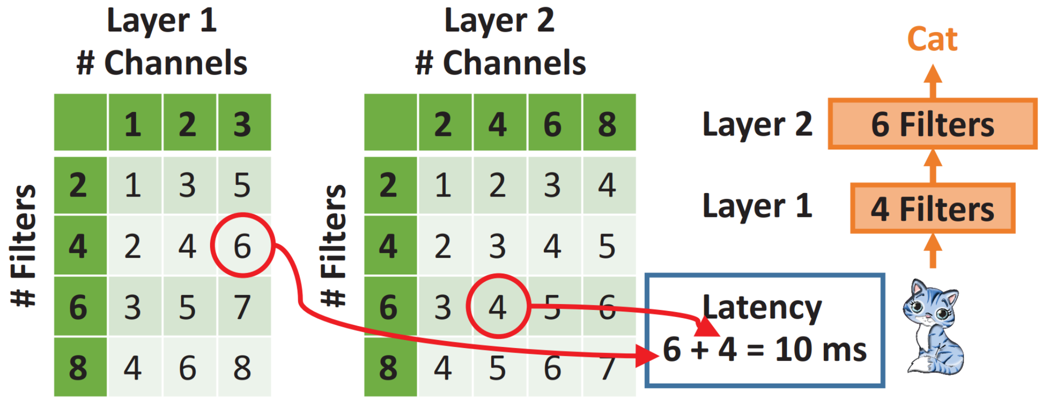
Use an easy-to-compute metric to approximate target



Latency → **# MACs**

Look-Up Table

Use table lookup



Layer 1 # Channels

# Filters	1	2	3
2	1	3	5
4	2	4	6
6	3	5	7
8	4	6	8

Layer 2 # Channels

# Filters	2	4	6	8
2	1	2	3	4
4	2	3	4	5
6	3	4	5	6
8	4	5	6	7

Latency
6 + 4 = 10 ms

Design Considerations for NAS

- The components may not be chosen individually
  - Some optimization algorithms limit the search space
  - Type of performance metric may limit the selection of the optimization algorithms

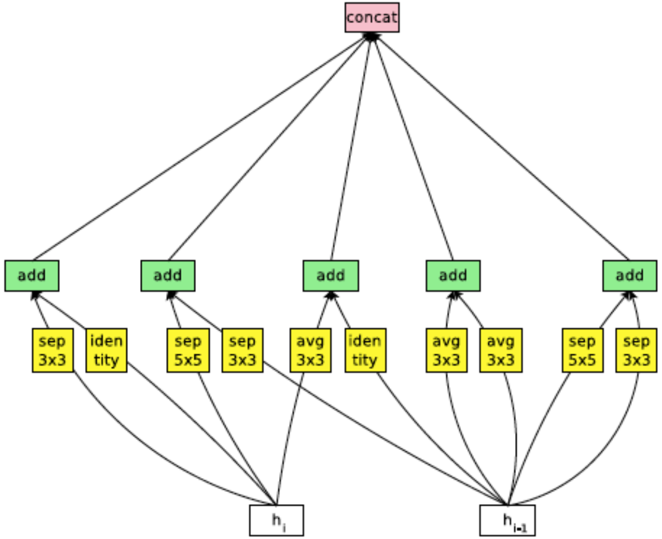
- Commonly overlooked properties
  - The complexity of implementation
  - The ease of tuning hyperparameters of the optimization
  - The probability of convergence to a good architecture

Example: NASNet

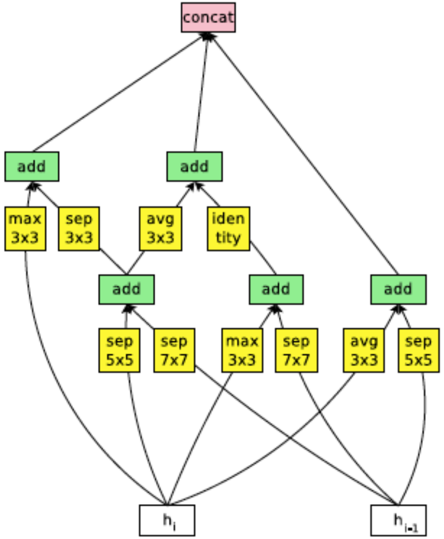
- Search Space: Build model from popular layers
  - Identity
  - 1x3 then 3x1 convolution
  - 1x7 then 7x1 convolution
  - 3x3 dilated convolution
  - 1x1 convolution
  - 3x3 convolution
  - 3x3 separable convolution
  - 5x5 separable convolution
  - 3x3 average pooling
  - 3x3 max pooling
  - 5x5 max pooling
  - 7x7 max pooling

[**Zoph**, *CVPR* 2018]

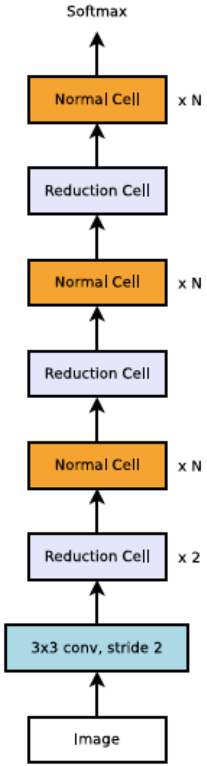
NASNet: Learned Convolutional Cells



Normal Cell



Reduction Cell

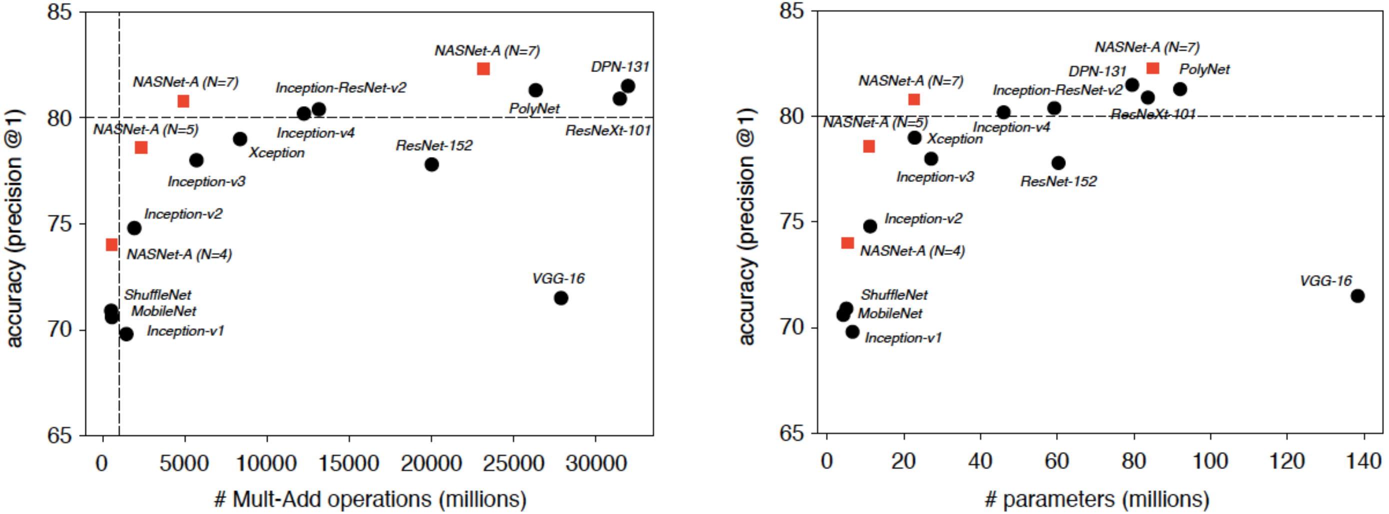


ImageNet Architecture

[**Zoph**, *CVPR* 2018]

NASNet: Comparison with Existing Networks

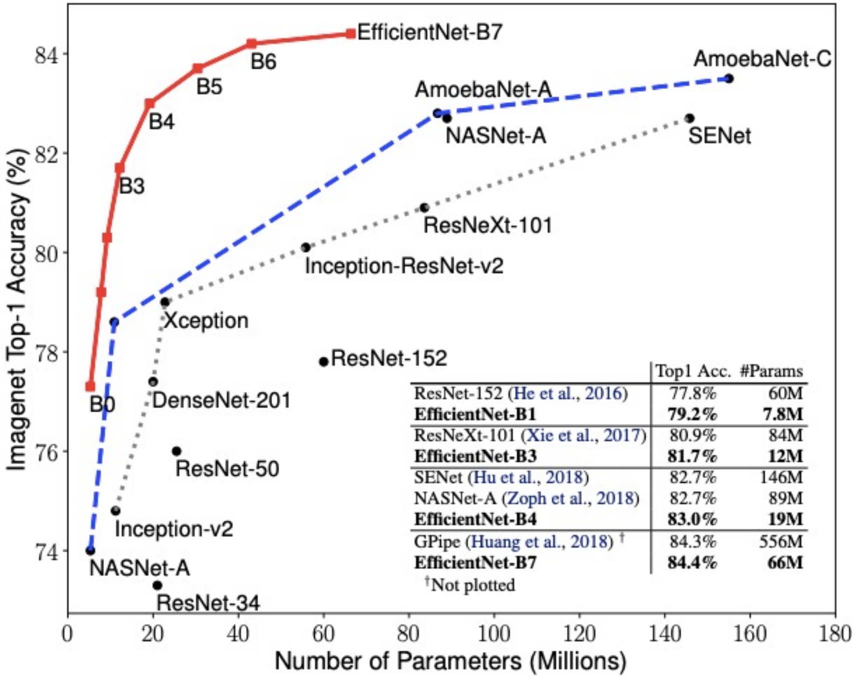
Learned models have improved accuracy vs. ‘complexity’ tradeoff compared to handcrafted models



[**Zoph**, *CVPR* 2018]

EfficientNet

Uniformly scaling all dimensions including depth, width, and resolution
since there is an interplay between the different dimensions.
Use NAS to search for baseline model and then scale up.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) [†]	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

[†] Not plotted

[Tan, *ICML* 2019]

Summary

- Approaches used to improve accuracy by popular CNN models in the ImageNet Challenge
  - Go deeper (i.e., more layers)
  - Stack smaller filters and apply 1x1 bottlenecks to reduce number of weights such that the deeper models can fit into a GPU (faster training)
  - Use multiple connections across layers (e.g., parallel and short cut)
- Efficient models aim to reduce number of weights and number of operations
  - Most use some form of filter decomposition (spatial, depth and channel)
  - Note: Number of weights and operations does not directly map to storage, speed and power/energy. Depends on hardware!
- Filter shapes vary across layers and models
  - Need flexible hardware!

Warning!

- These works often use **number of weights and operations** to measure "**complexity**"
- Number of weights provides an indication of **storage cost** for inference
- However later in the course, we will see that
  - Number of operations doesn't directly translate to latency/throughput
  - Number of weights and operations doesn't directly translate to power/energy consumption
- Understanding the underlying hardware is important for evaluating the impact of these "efficient" CNN models

References

- Book: Chapter 2 & 9
  - https://doi.org/10.1007/978-3-031-01766-7
- Other Works Cited in Lecture (increase accuracy)

  - **LeNet:** LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proc. IEEE 1998.
  - **AlexNet:** Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NeurIPS. 2012.
  - **VGGNet:** Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR 2015.
  - **Network in Network:** Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." ICLR 2014
  - **GoogleNet:** Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR 2015.
  - **ResNet:** He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR 2016.
  - **DenseNet:** Huang, Gao, et al. "Densely connected convolutional networks." CVPR 2017
  - **Wide ResNet:** Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." BMVC 2017.
  - **ResNext:** Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." CVPR 2017
  - **SENets:** Hu, Jie et al., "Squeeze-and-Excitation Networks," CVPR 2018
  - **NFNet:** Brock, Andrew, et al., "High-Performance Large-Scale Image Recognition Without Normalization," arXiv 2021

References

- Other Works Cited in Lecture (increase efficiency)
  - **InceptionV3:** Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." CVPR 2016.
  - **SqueezeNet:** Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size." ICLR 2017.
  - **Xception:** Chollet, François. "Xception: Deep Learning with Depthwise Separable Convolutions." CVPR 2017
  - **MobileNet:** Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
  - **MobileNetv2:** Sandler, Mark et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks," CVPR 2018
  - **MobileNetv3:** Howard, Andrew et al., "Searching for MobileNetV3," ICCV 2019
  - **ShuffleNet:** Zhang, Xiangyu, et al. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices." CVPR 2018
  - **Learning Network Architecture**: Zoph, Barret, et al. "Learning Transferable Architectures for Scalable Image Recognition." CVPR 2018
- Other Works Cited in Lecture (Increase accuracy and efficiency)
  - **EfficientNet:** Tan, Mingxing, et al. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019