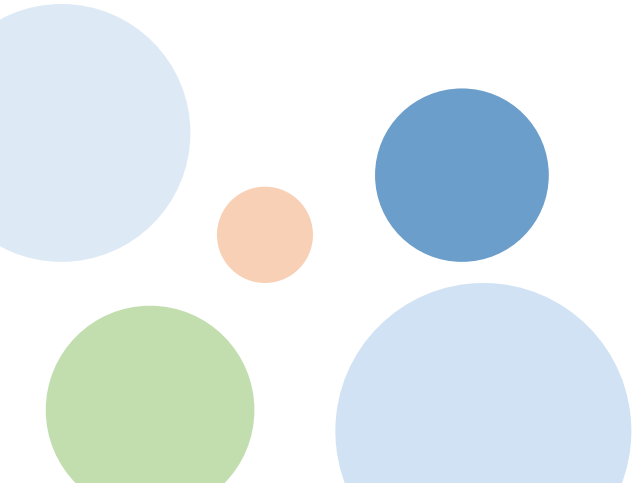
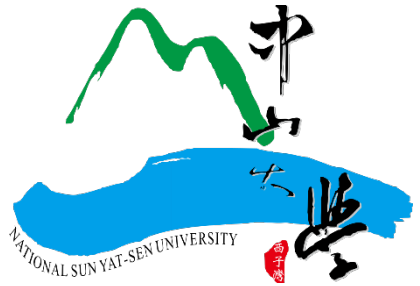
CNN Benchmark Models





Outlines

- CNN Representative Architectures
  - LeNet (1998)
  - AlexNet (2012)
  - ZFNet (2013)
  - VGG (2014)
  - GoogLeNet v1~v5 (2014~2017)
  - ResNet, ResNeXt (2015-2016)
  - SqueezeNet (2016)
  - ShuffleNet (2017)
  - MobileNet (2017)
- Model Compression
  - Pruning + Weight Sharing
  - Depthwise Separable Convolution
- Recurrent Neural Network (RNN)
- Generative Adversarial Network (GAN)

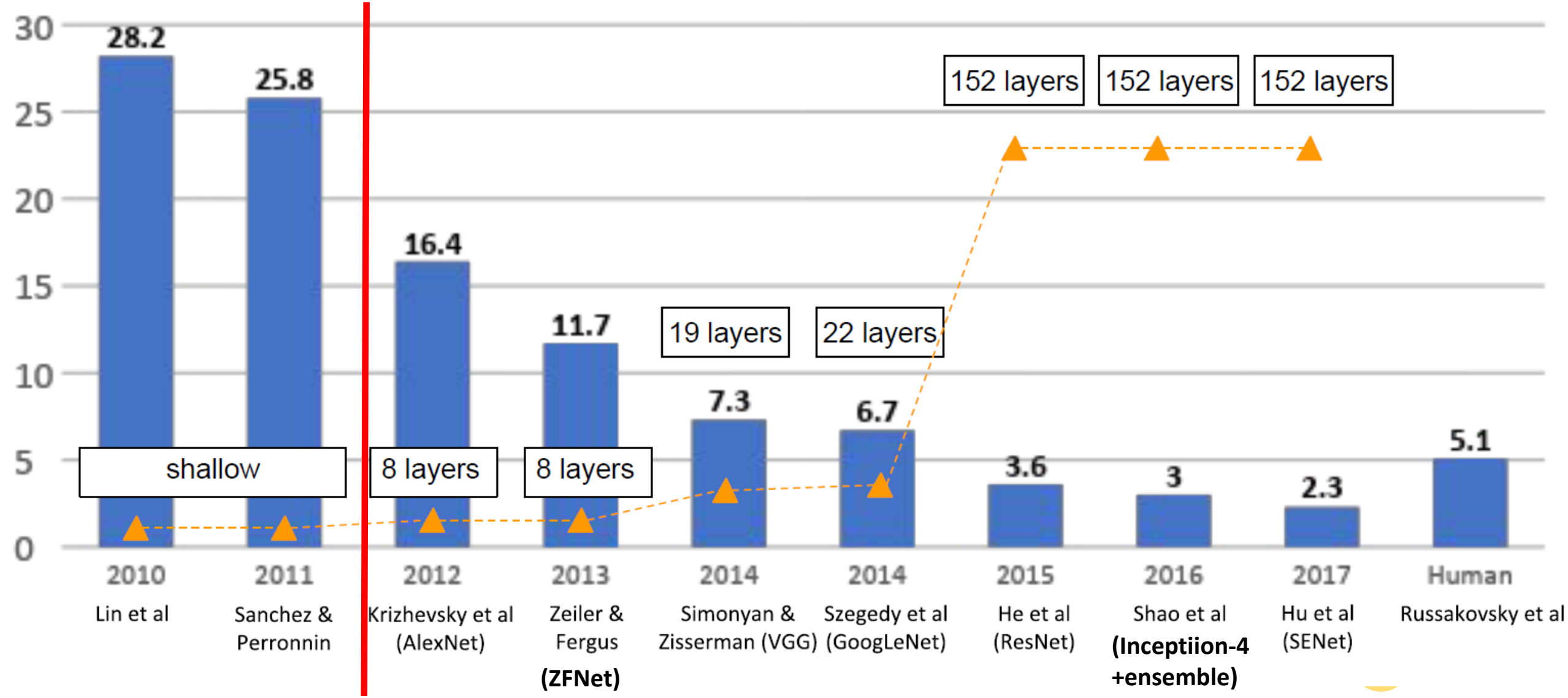
References and Credits

- Stanford CS231n, “Convolutional Neural Networks for Visual Recognition”
  - by Fei-Fei Li, Justin Johnson, and Serena Yeung
- MIT 6.S191, “Deep Learning”
  - by Alexander Amini, and Ava Soleimany
- UVA Deep Learning , Univ. of Amsterdam
  - by Efstratios Gavves
- CMSC 35264 Deep Learning, Univ. of Chicago
  - by Shubhendu Trivedi and Risi Kondor
- Deep Learning for Computer Vision
  - by 台大資工系 莊永裕 教授
- book: Deep Learning
  - by Ian Goodfellow, Yoshua Bengio, and Aaron Courville

Representative CNN Models

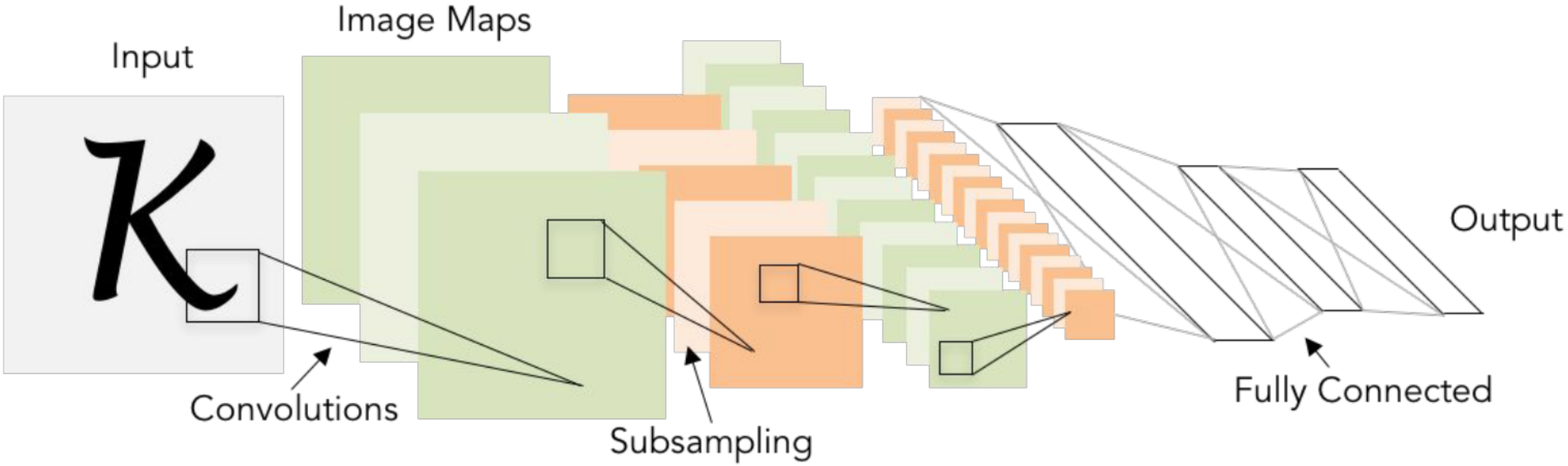
- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGG (2014)
- GoogLeNet (2014)
- ResNet (2015)
- ResNeXT (2016)
- SqueezeNet (2016)
- ShuffleNet (2017)
- MobileNet (2017)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



LeNet*

first CNN model, published in 1998
Training dataset: MNIST, recognition of 0~9 digits



Conv filters were 5x5. applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

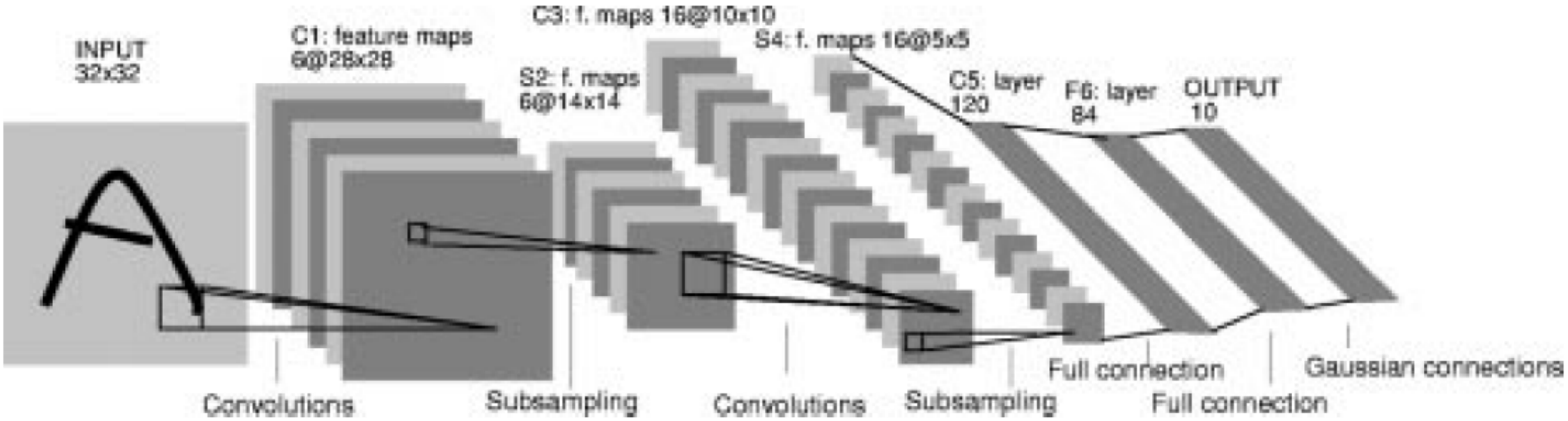
*Y. LeCun, et al., “Gradient-Based Learning Applied to Document Recognition,” Proc. IEEE, 1998. (AT&T Bell Lab)

LeNet-5

- C1: 5x5x1x6 convolution (32x32 -> 28x28)
- S2: 2x2 avg. pooling with stride=2 (28x28 -> 14x14)
- **C3**: 5x5x**{3,4,6}**x16 convolution (14x14 -> 10x10)
  - Convolution with 3, or 4, or 6 input channels (see right table)
- S4: 2x2 avg. pooling with stride=2 (10x10 -> 5x5)
- **C5**: 5x5x16x120 convolution (5x5 -> 1x1)
  - Same as Fully Connected (FC) layer 5x5x16 -> 120
- F6: 120x1 fully-connected (120 -> 84)

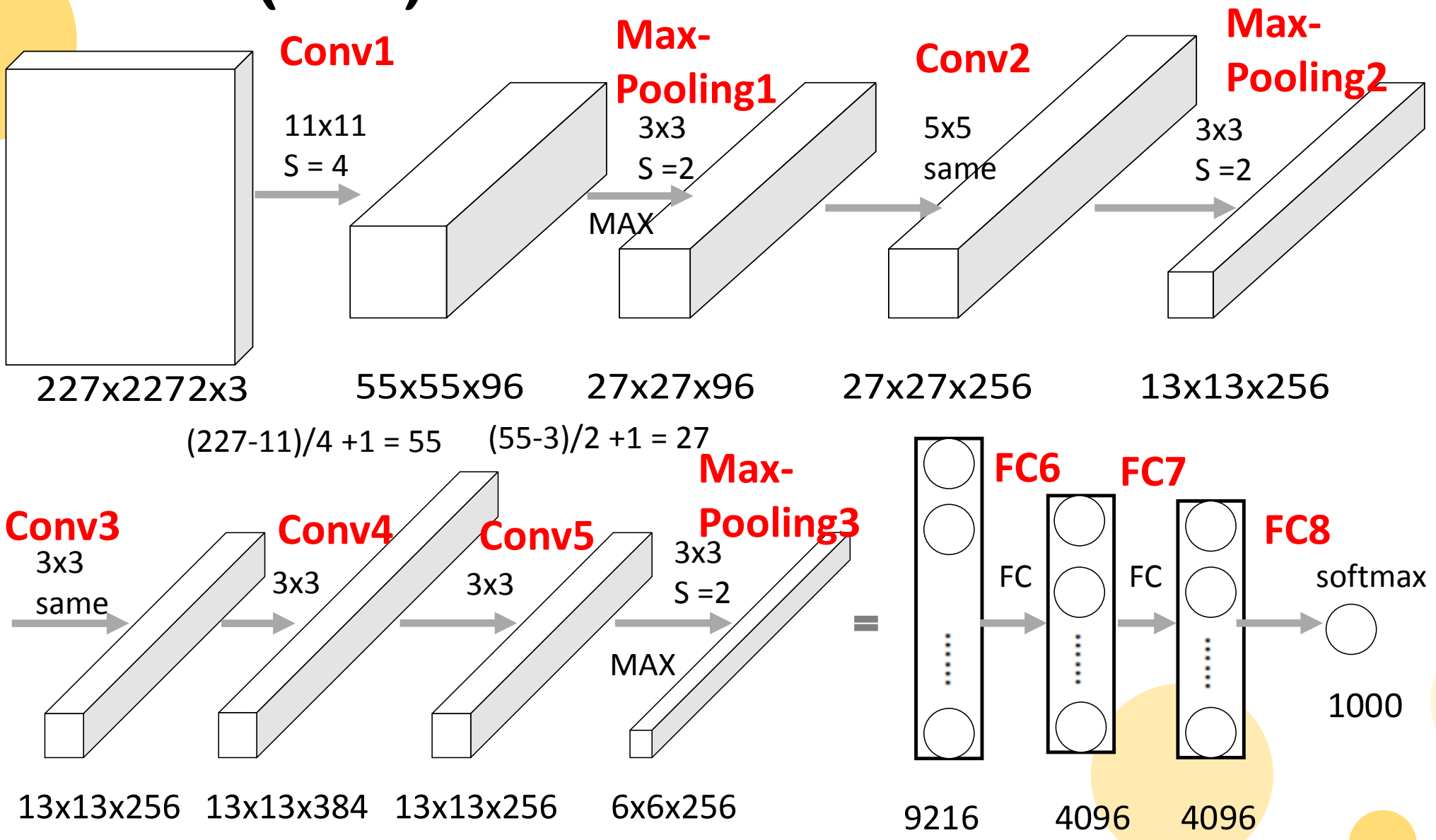
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

combination of input feature maps in C3



AlexNet (1/3)

2012 ILSVRC winner, training dataset: ImageNet

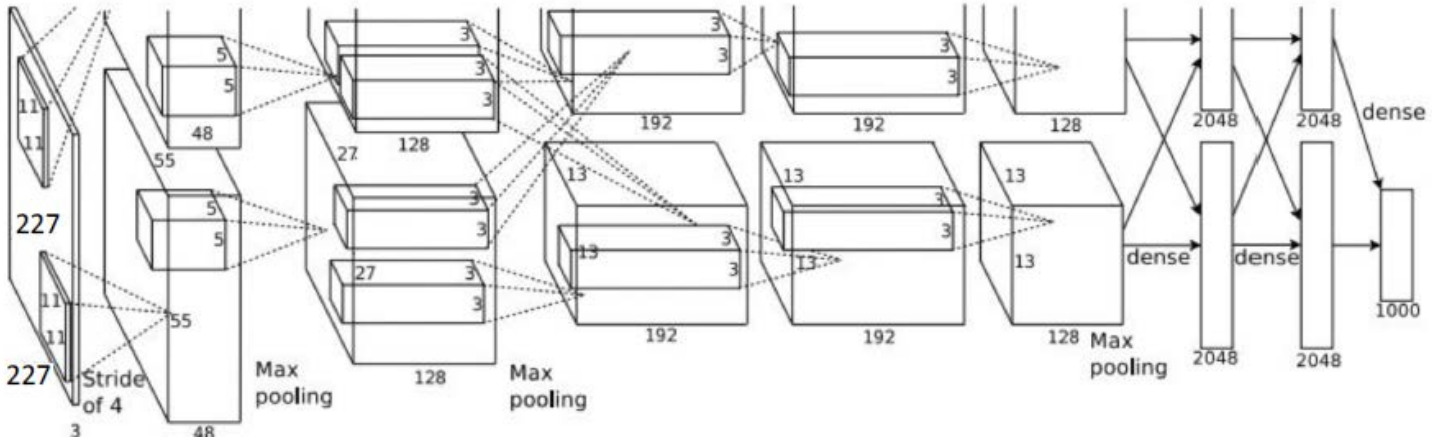


* Alex Krizhevsky, ea tal., “ImageNet Classification with Deep Convolutional Neural Networks,” NIPS 2012. (Univ. Toronto)

AlexNet (2/3)

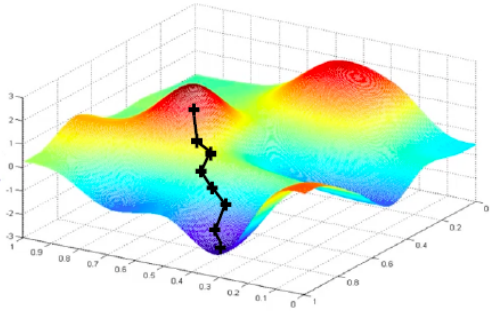
- first CNN-based ILSVRC winner
  [Krizhevsky et al. 2012]

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



Details/Retrospectives:

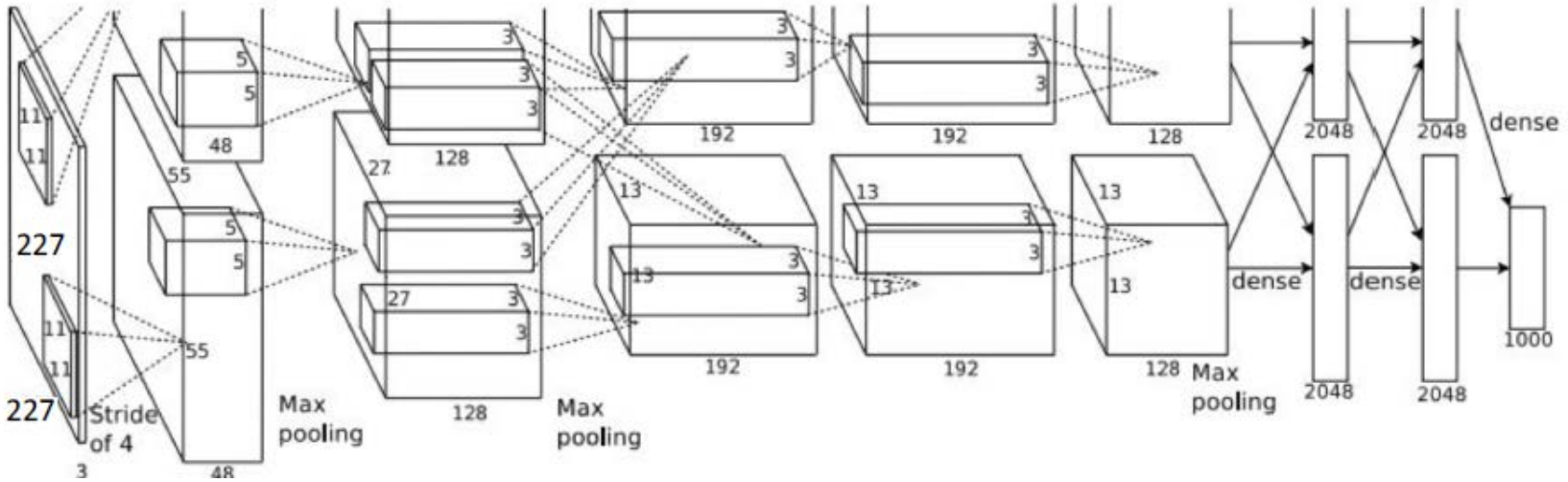
- first use of ReLU
- used Local Response Normalization (LRN) layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- mini-batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2. reduced by 10 manually
- L2 regularization with weight decay 5e-4

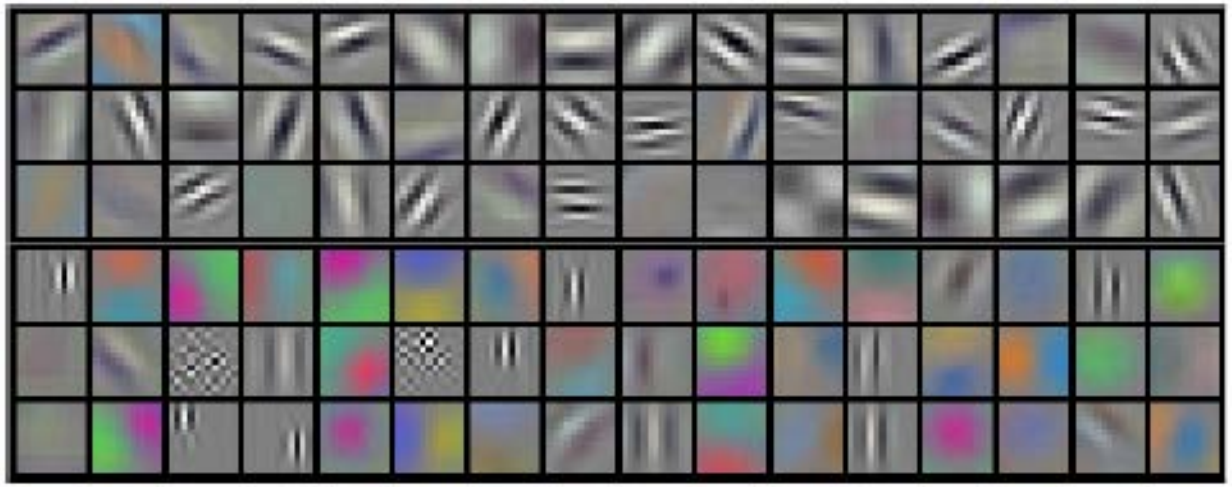


$$v \leftarrow \alpha v - \epsilon g \qquad \theta \leftarrow \theta + v$$

AlexNet (3/3)

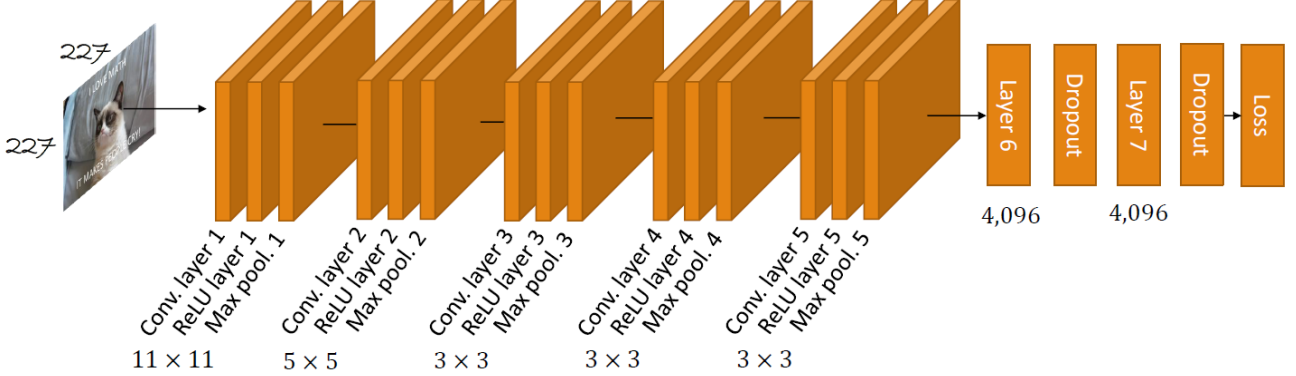
- 5 Conv + 3 FC
  - **60M** parameters
  - filter sizes: 11x11, 5x5, 3x3
- trained on two GPUs
- first to use
  - deep (8-layer) CNN in ILSVRC
  - ReLU for non-linear activation function
  - dropout for regularization
  - local response normalization (LRN)
- data augmentation
  - flip, crop
- 1st place of ILSVRC 2012 with **15.3%** top-5 error rate
  - 2nd place has 26.2% top-5 error rate



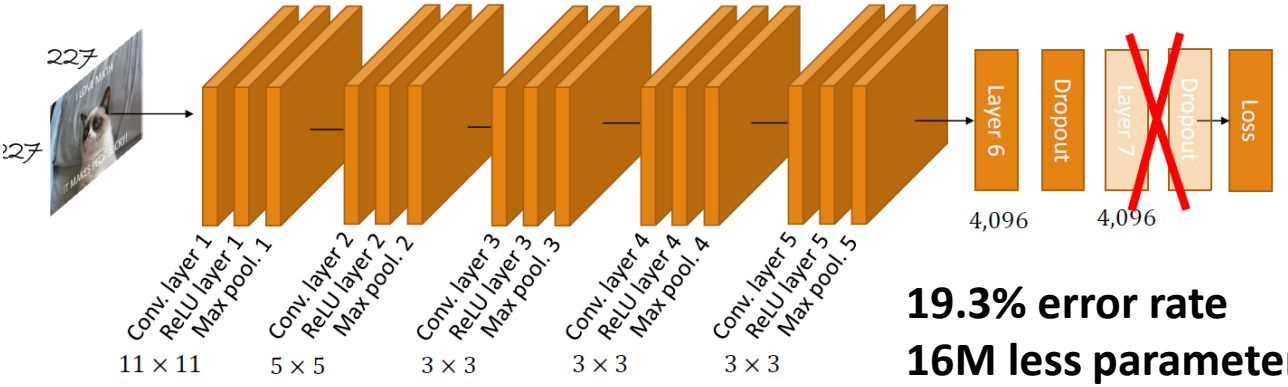


- 96 convolutional kernels of size 11x11x3 in the first Conv. layer
- learned a variety of frequency- and oriented-selected kernels, and colored blobs

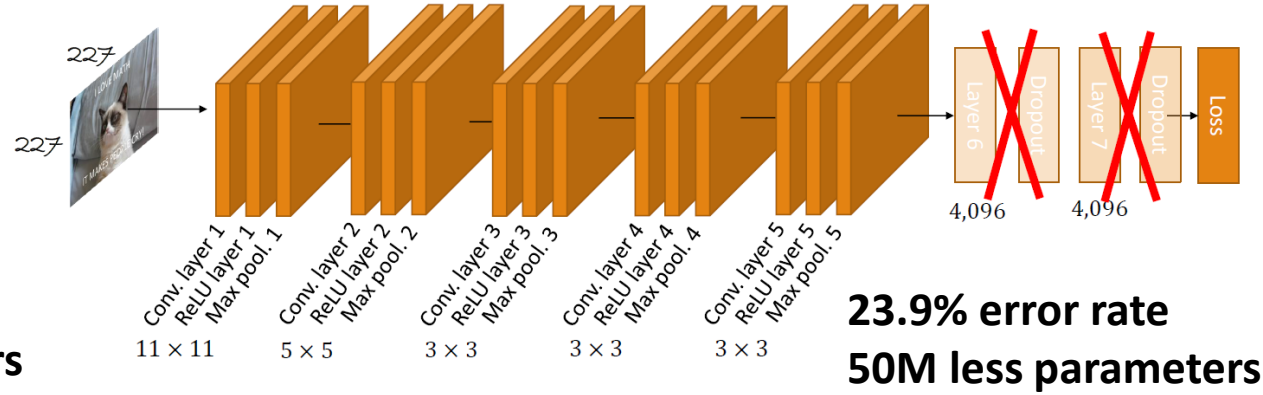
AlexNet variants*



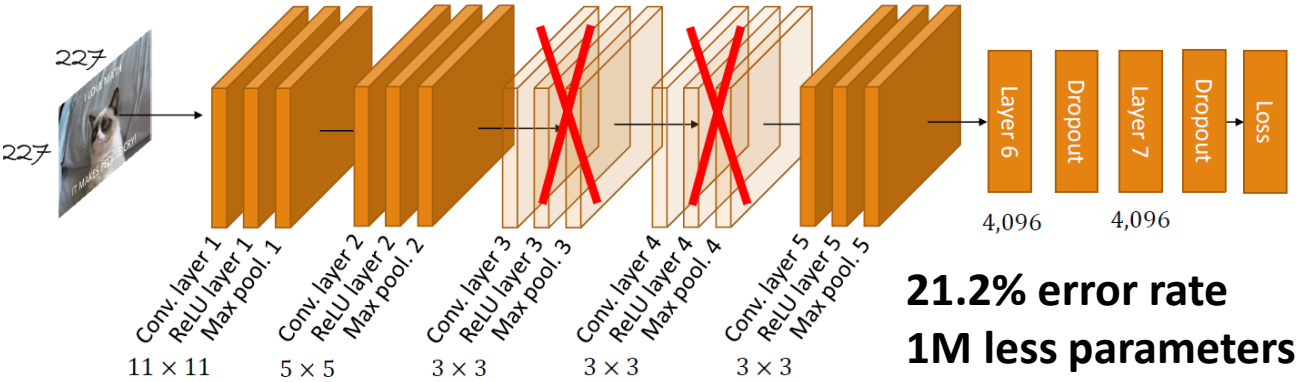
18.2% error rate
60M parameters



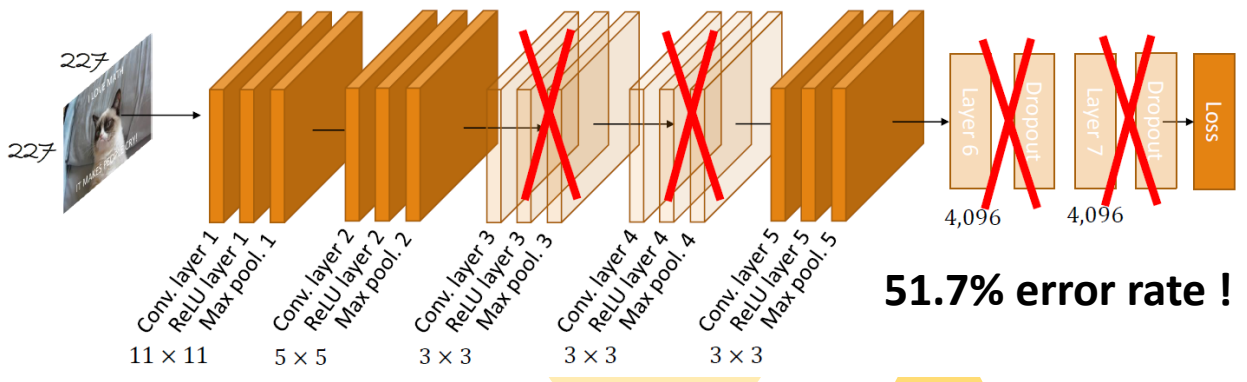
19.3% error rate
16M less parameters



23.9% error rate
50M less parameters



21.2% error rate
1M less parameters

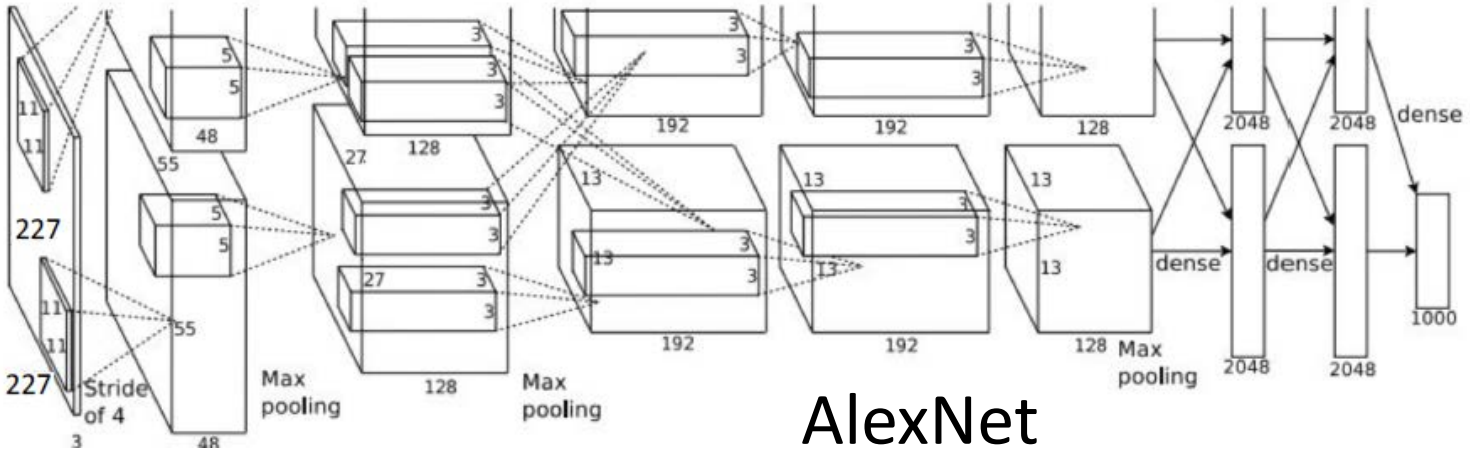


51.7% error rate !

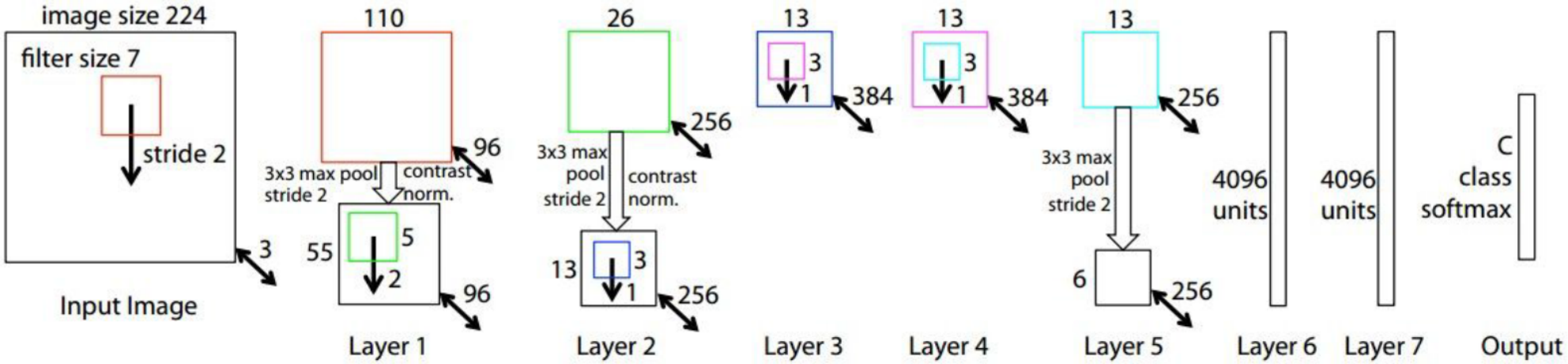
* UVA Deep Learning Course, by Efstratios Gavves

ZFNet* (1/3)

- improve hyper-parameters over AlexNet
- Visualization of CNN



ZFNet



Similar to AlexNet, but:

CONV1: change from (11 x11 stride 4) to (7x7 stride 2)

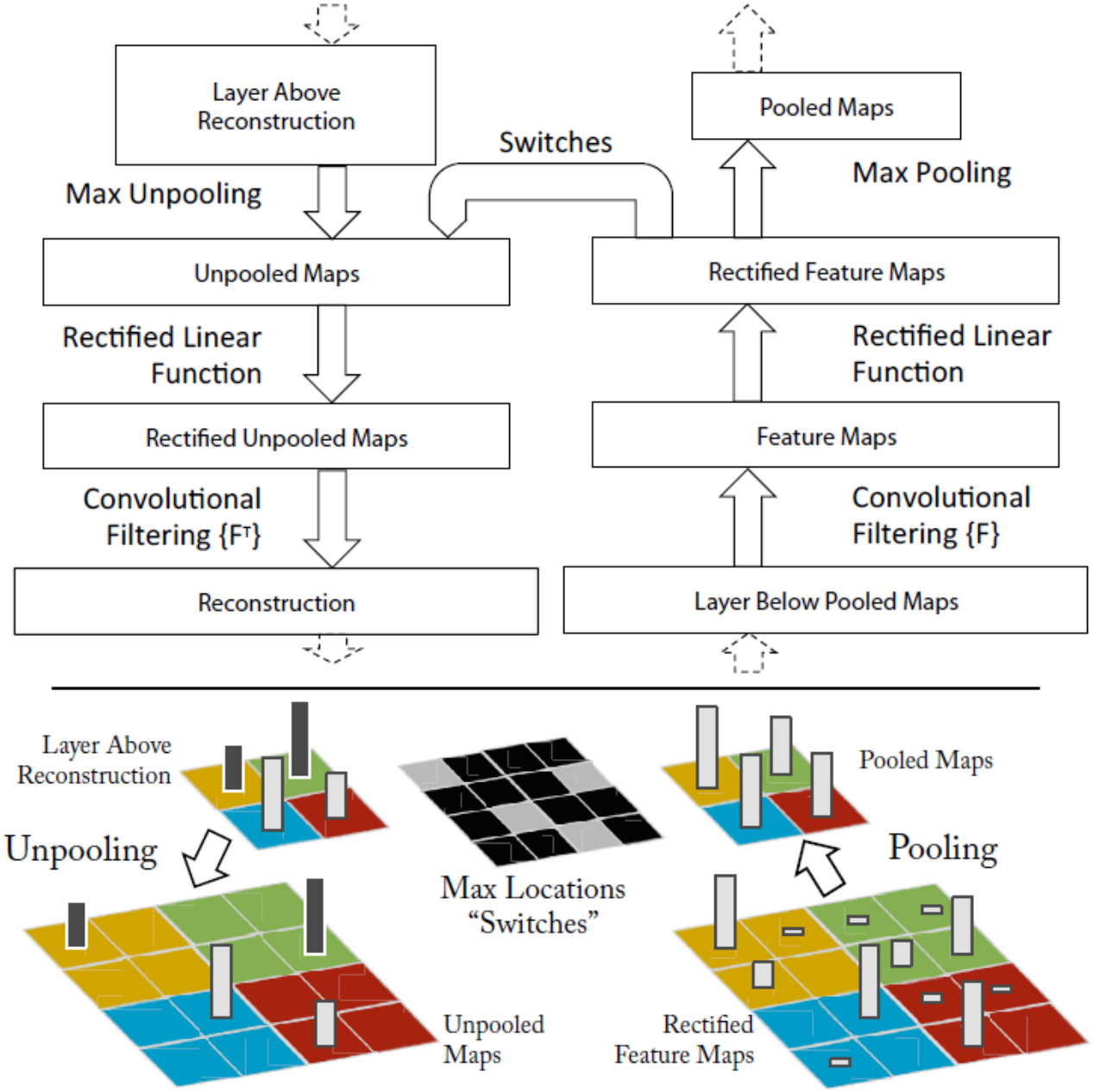
CONV3,4,5: instead of 384. 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

*M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," ECCV, 2014, (NYU)

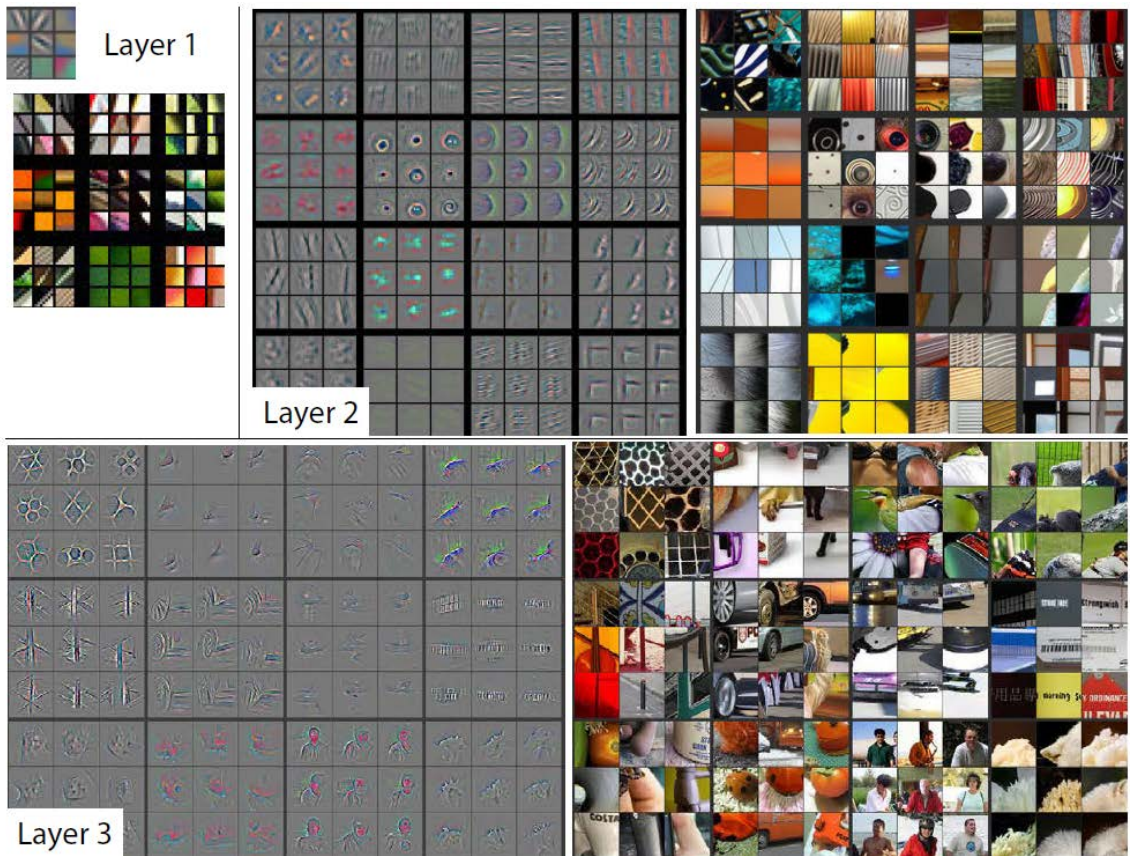
Visualization

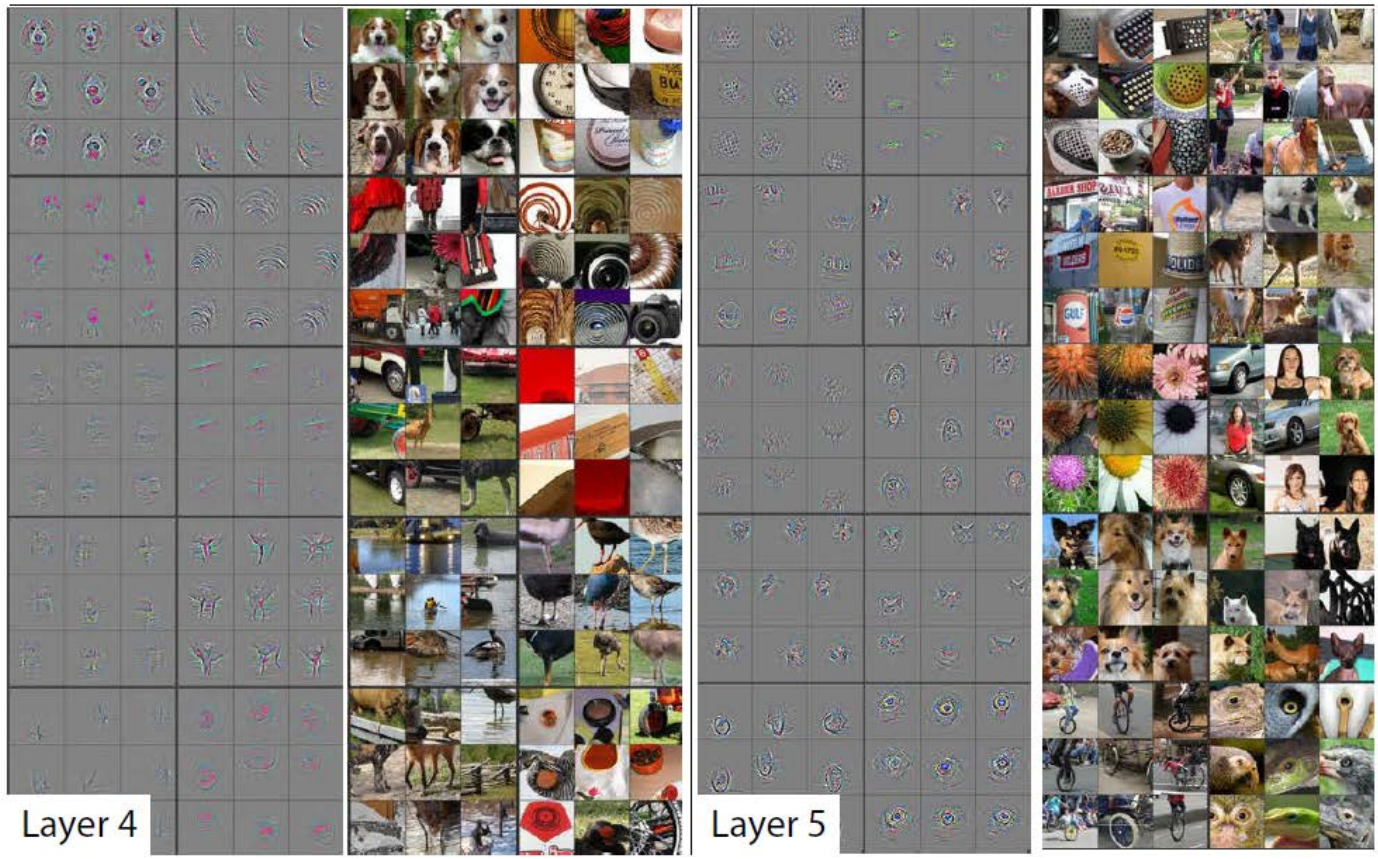
- map activations back to the input pixel space, showing what input patterns originally caused a given activation in the feature maps
- a deconvnet is attached to each layer, providing a continuous path back to image pixel
- to examine a given convent activation, set all other activations in the layer to zero and pass the feature maps as input to the deconvnet, successfully (1) unpool, (2) rectify, (3) filter, to reconstruct the activity in the layer beneath that gave rise to the chosen activations



ZF-Net (3/3)

- top 9 activations in a random subset of feature maps across validation data projected down to pixel space
  - reconstructed patterns from validation set that cause high activations in a given feature map





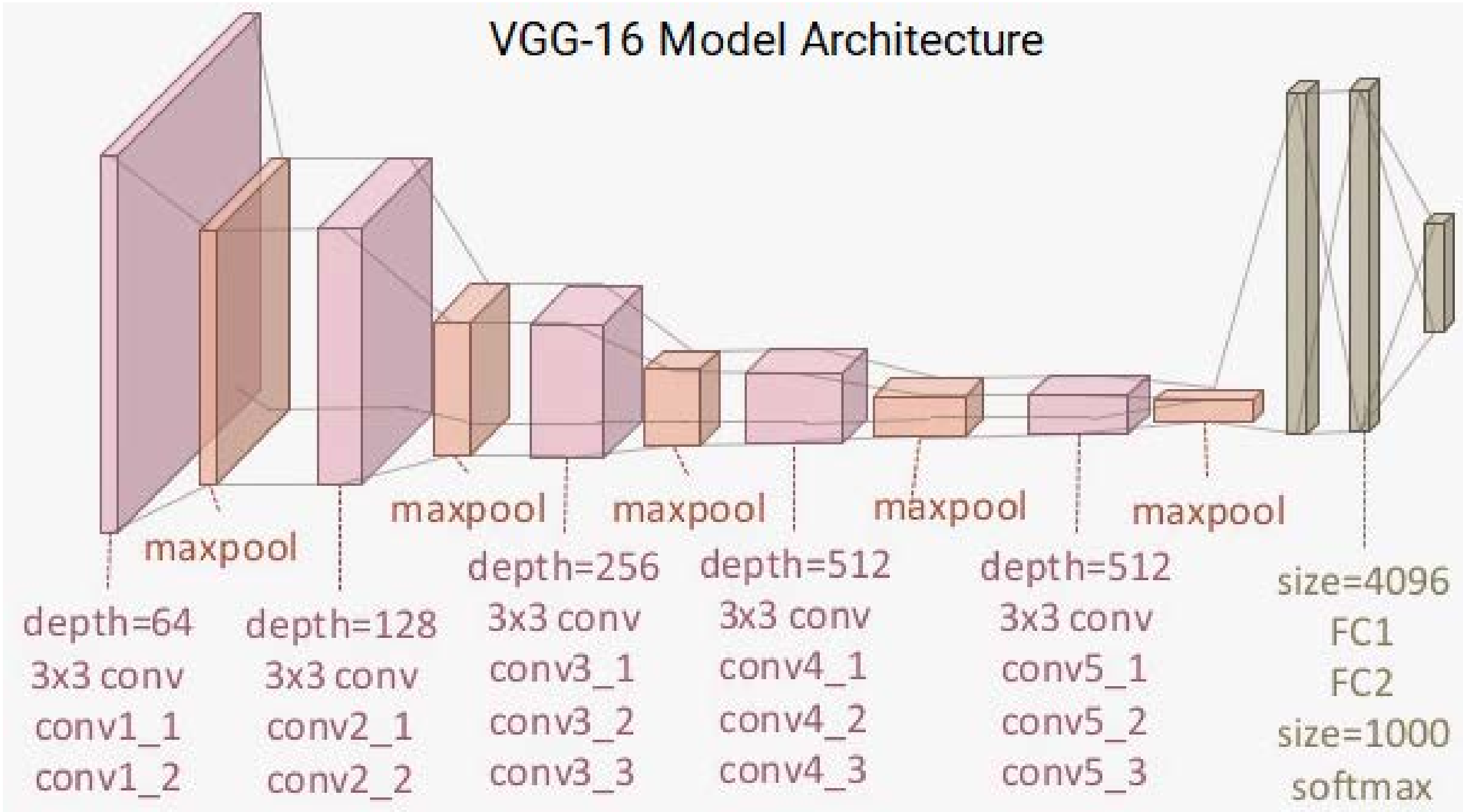
VGG* (1/4)

- 3x3 filters only, cp.
  - 11x11, 5x5, 3x3 in AlexNet
  - 7x7, 5x5, 3x3 in ZFNet
- 6 configurations
  - D configuration (VGG-16) is popular
- 138M parameters cp.
  - 60M in AlexNet
- increasing # of output channels: 64, 128, 256, 512, cp.
  - 384, 384, 256 in AlexNet
  - 512, 1024, 512 in ZFNet

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					

VGG16

*K. Simonyan and A. Zisserman, "Very Deep Convolution Networks for Large-Scale Image Recognition," ICLR 2015, (Visual Geometry Group, Oxford)


VGG-16 Model Architecture
maxpool
maxpool
maxpool
maxpool
maxpool
depth=64
3x3 conv
conv1_1
conv1_2
depth=128
3x3 conv
conv2_1
conv2_2
depth=256
3x3 conv
conv3_1
conv3_2
conv3_3
depth=512
3x3 conv
conv4_1
conv4_2
conv4_3
depth=512
3x3 conv
conv5_1
conv5_2
conv5_3
size=4096
FC1
FC2
size=1000
softmax

VGG (2/4)

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

most memory is in early Conv layers

most parameters are in late FC layers

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

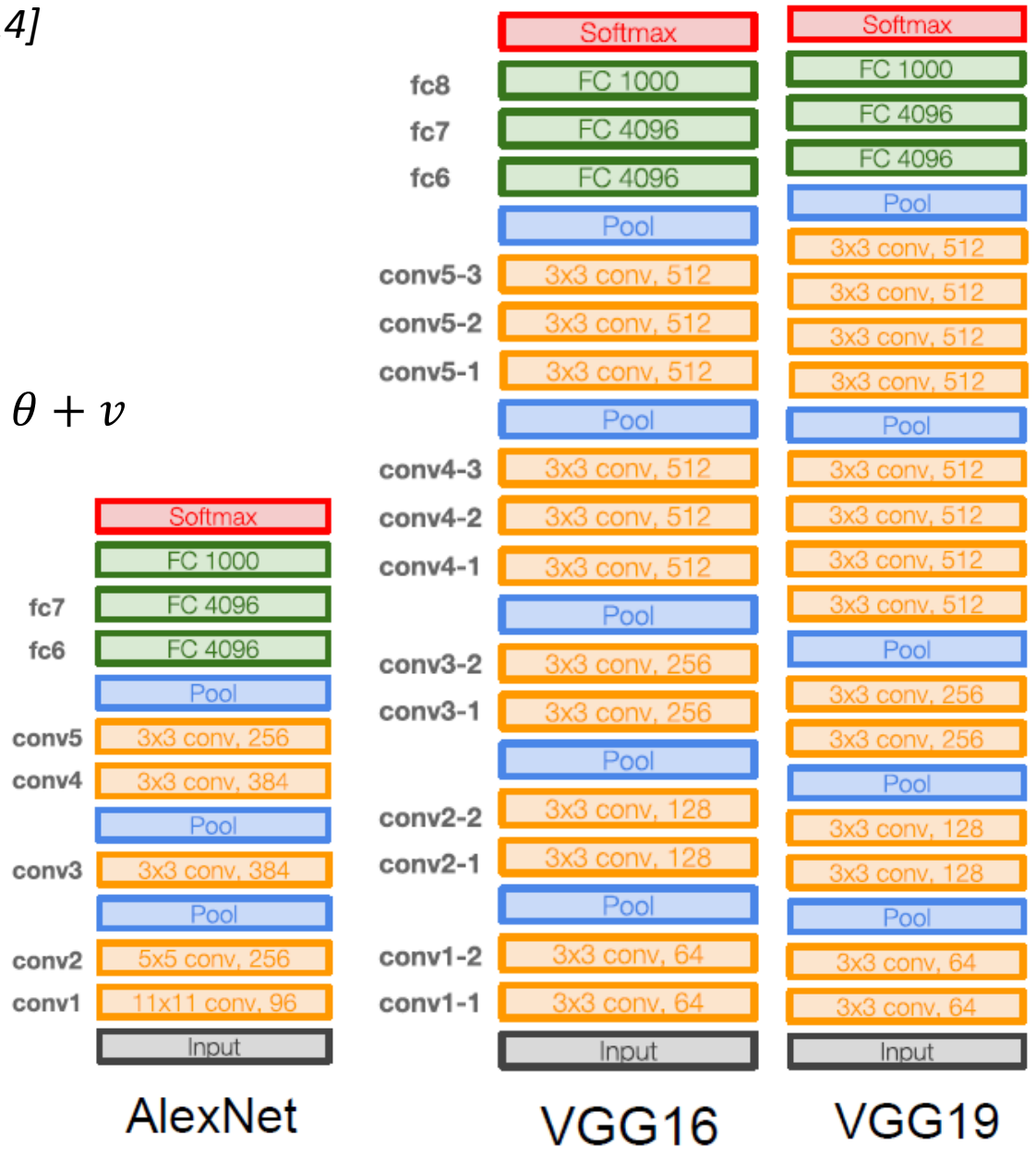
VGG16

VGG (3/4)

[Simonyan and Zisserman, 2014]

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as AlexNet
  * batch size: 256 (cp. 128 in AlexNet)
  * SGD with momentum=0.9
  * learning rate=1e-2
    - then decrease by 1/10 when validation accuracy stop improving
    - 3 times this learning rate decreasing
  * weight decay = 5e-4
  * dropout on first two FC layers
- No Local Response Normalization (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks (such as object detection)

$$v \leftarrow \alpha v - \epsilon g \qquad \theta \leftarrow \theta + v$$

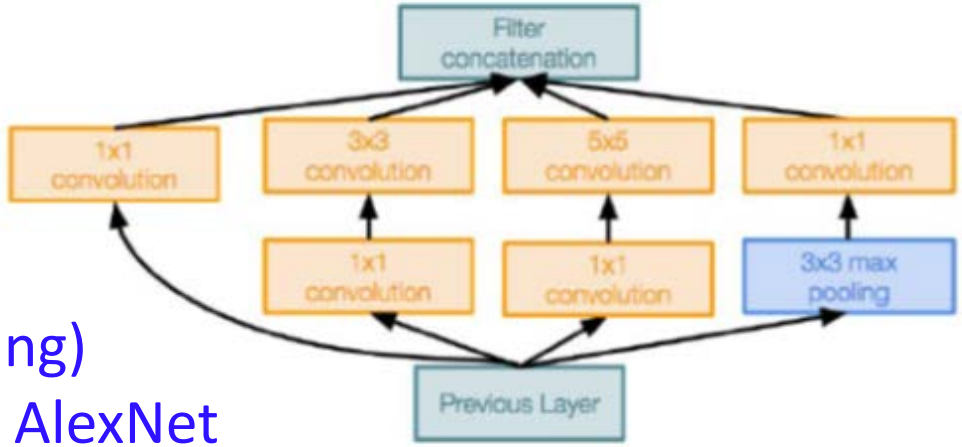


VGG (4/4)

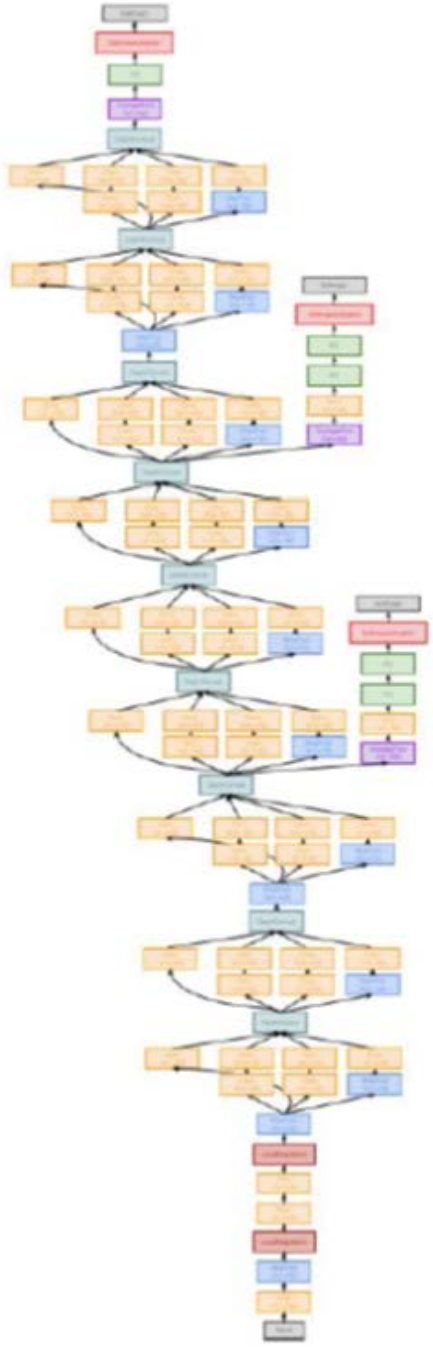
- 13~16 Conv + 3 FC
  - 138 M parameters (cp. 60M in AlexNet)
  - filter sizes: 3x3 (smallest filters to capture “up”, “down”, “left”, “right” of a pixel location
- cascade of several 3x3 Conv layers
  - 2 layers of 3x3 => receptive field of 5x5 (ops: 9*2 vs. 25)
  - 3 layers of 3x3: receptive field of 7x7 (ops: 9*3 vs. 49)
  - More nonlinearity for same “size” of pattern learning
- ILSVRC 2014 with top-5 error rate 7.3%
  - cp. 16.4% in AlexNet
- VGG is used for feature extraction in many applications such as object detection

GoogLeNet*

- inception + 1x1 convolution
  - multiple kernel filters of different sizes (1x1, 3x3, 5x5) in a layer
  - objects have great variation in size
    - receptive field should vary in size accordingly
  - 1x1 to reduce complexity



Inception module



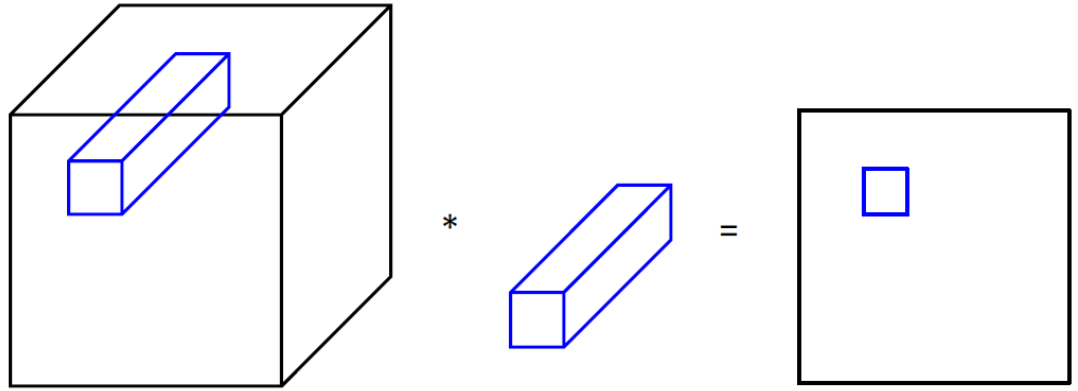
- 22 layers (+ 5 pooling layers)
- Efficient "Inception" module
- No FC layers (use Global Average Pooling)
- Only ~ 6 M parameters! (10x less than AlexNet
- ILSVRC'14 classification winner
  (6.7% top 5 error)

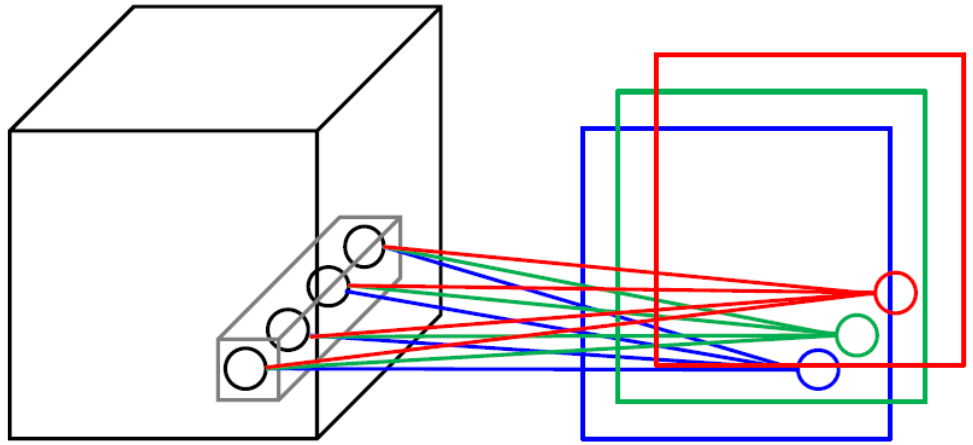
Deeper networks, with computational efficiency

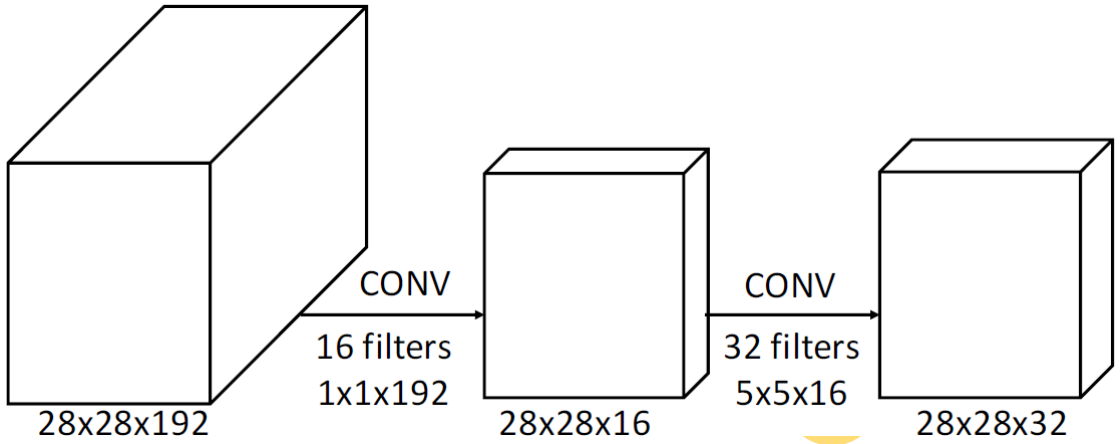
* C. Szegedy, et al., “Going Deeper with Convolutions,” CVPR, 2015. (Google)

1x1 Convolution

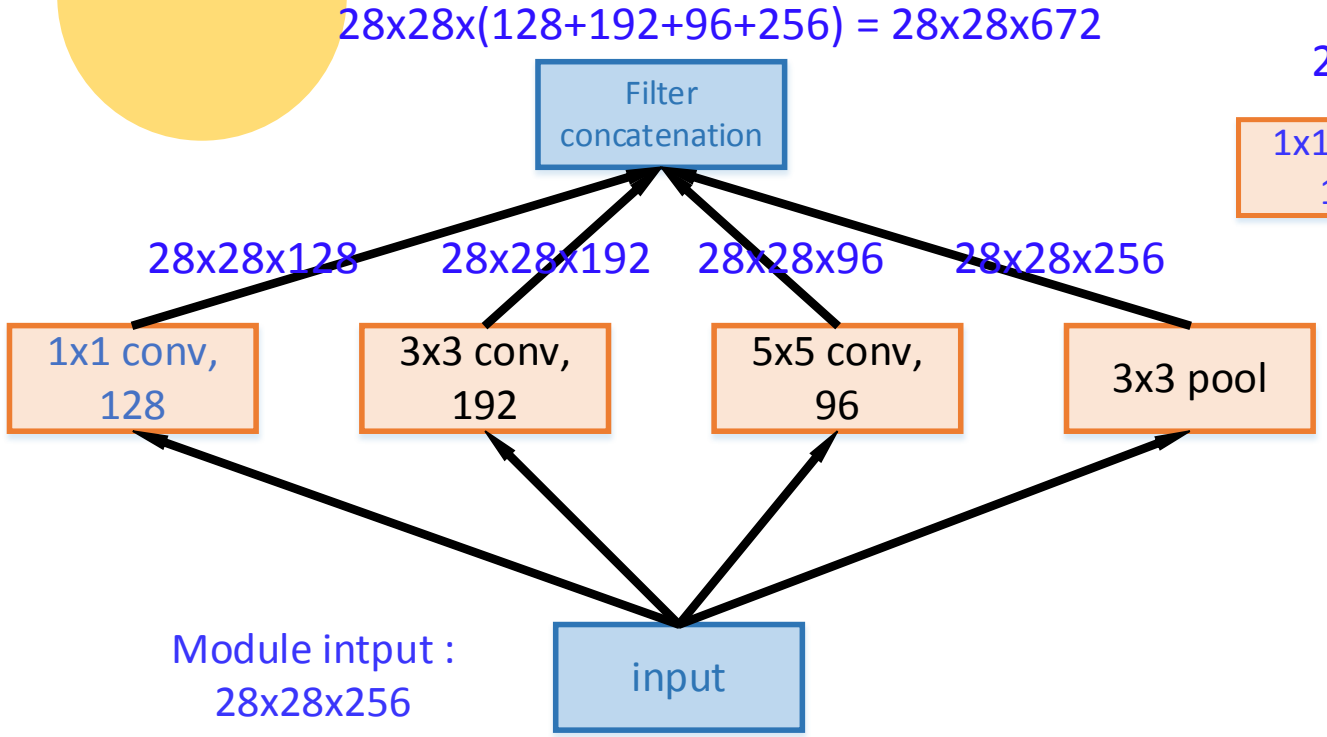
- weighted average across input channels (1x1 filter)
  - also called pointwise convolution
  - Linear combination of input channels
  - Used to reduce # of output channels for purpose of complexity reduction
- different 1x1 convolution (for different output channels) apply different weights
  - different 1x1 filters generate different features
- 1x1 convolution can be used to **shrink/expand** number of channels
- more 1x1 convolution layers provide extra non-linearity (due to more layers of activations)
- similar idea of NiN (Network in Network) in 2014





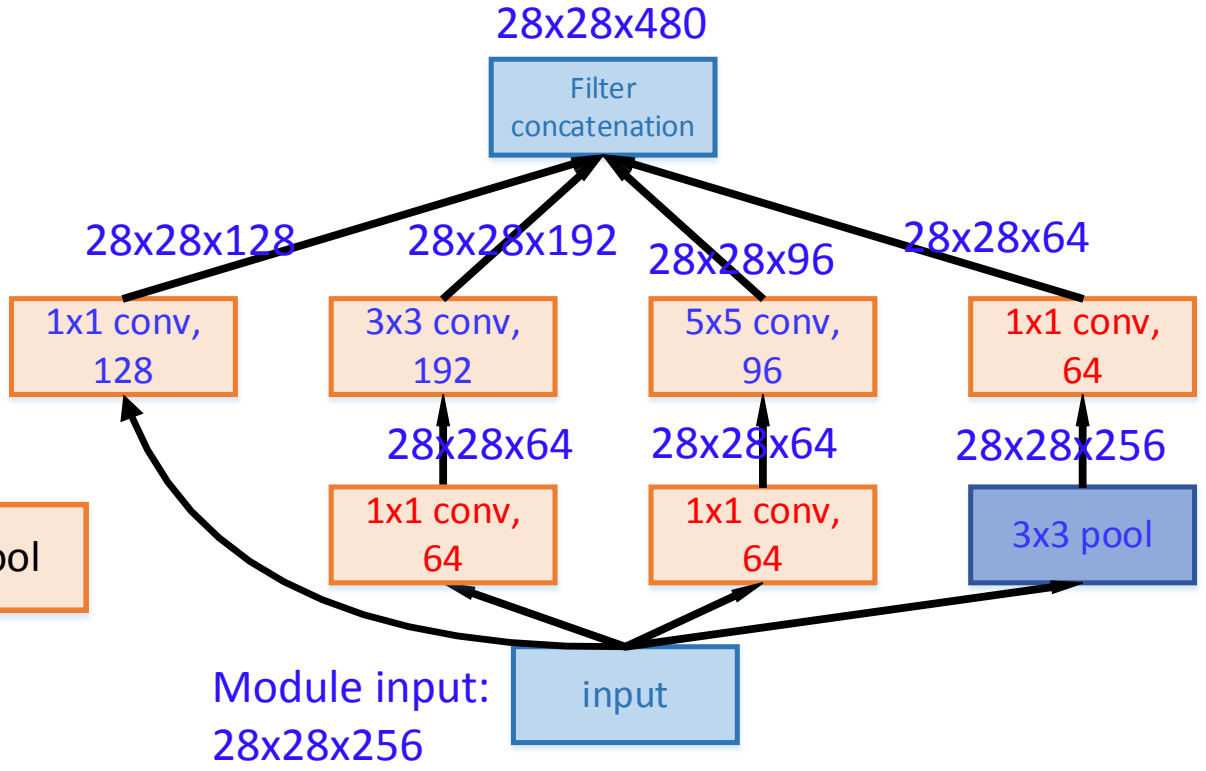


Inception Module



Conv Ops:

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

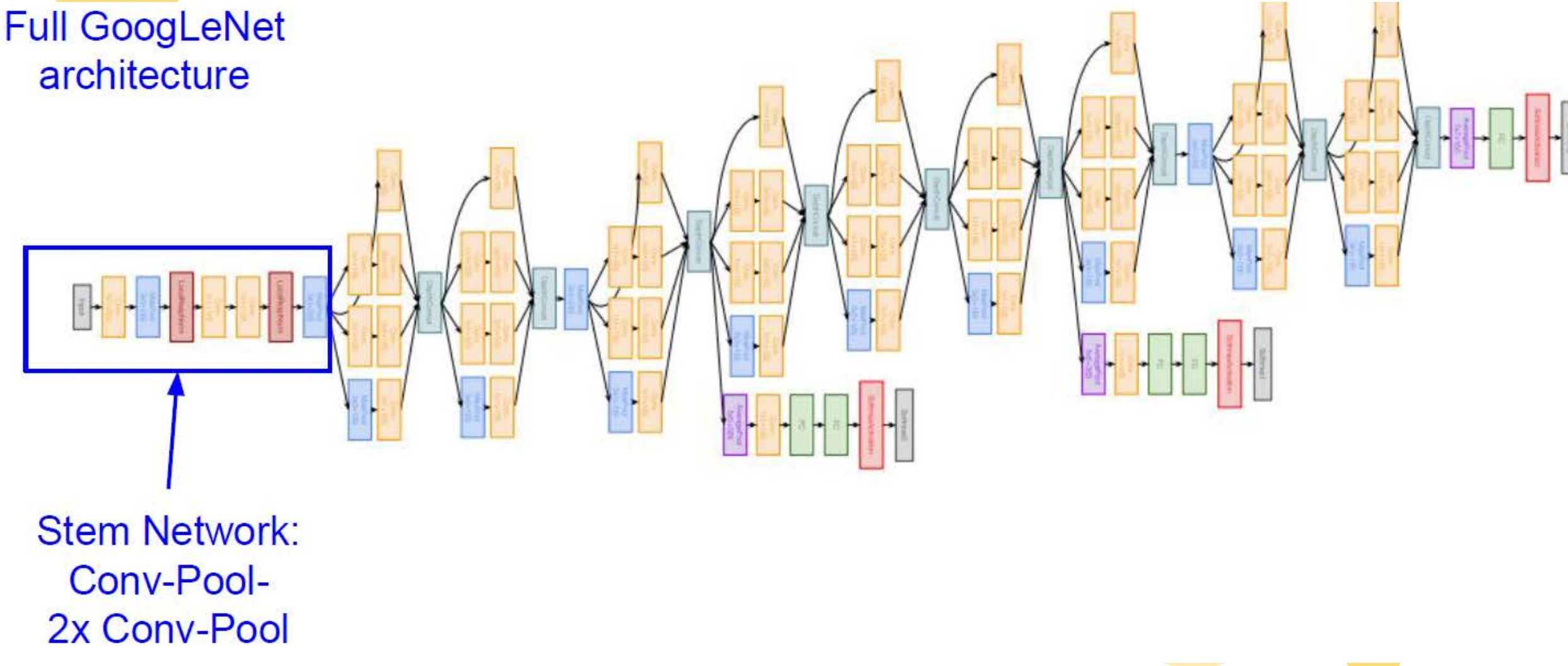


Conv Ops:

[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x25
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

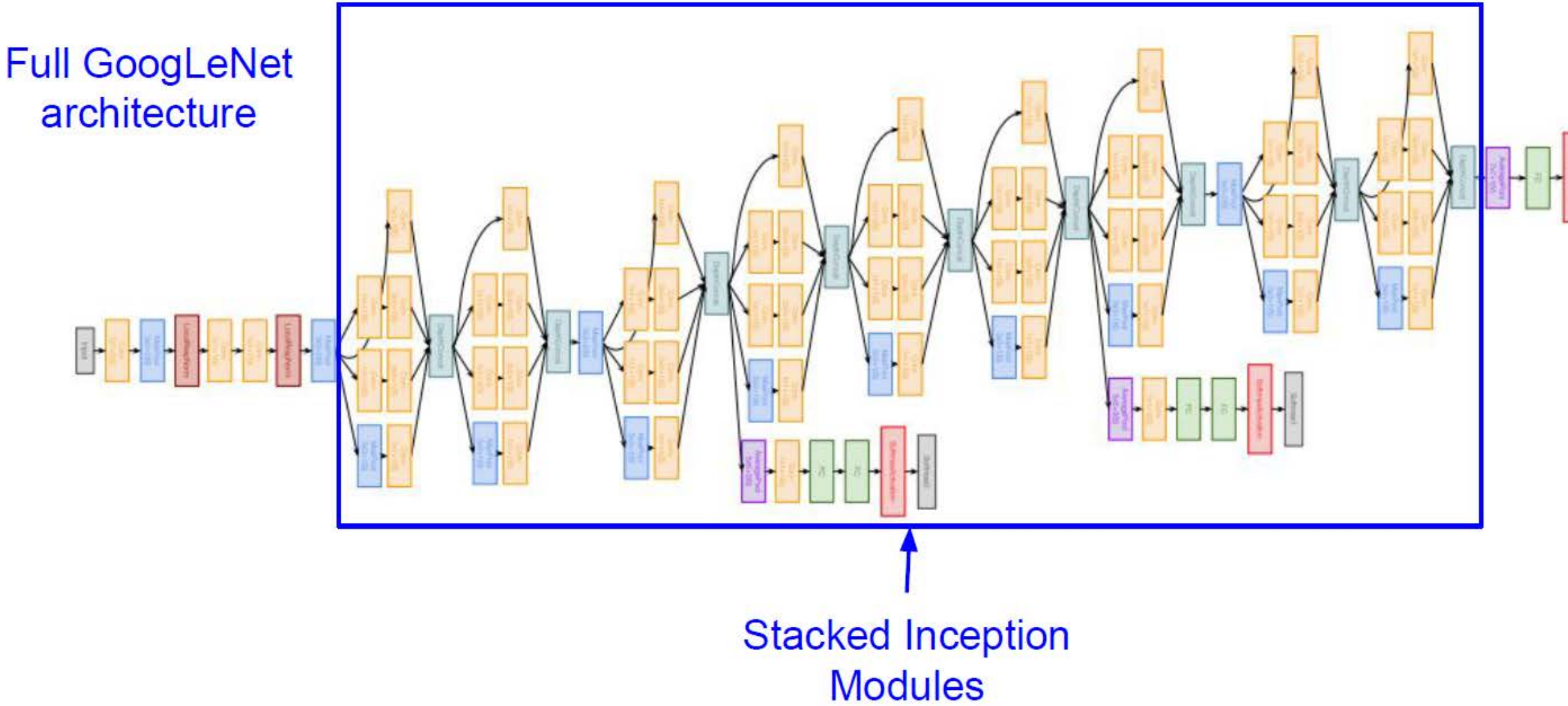
GoogLeNet Stem Network

- stem network of several Conv and Pool layers at the beginning



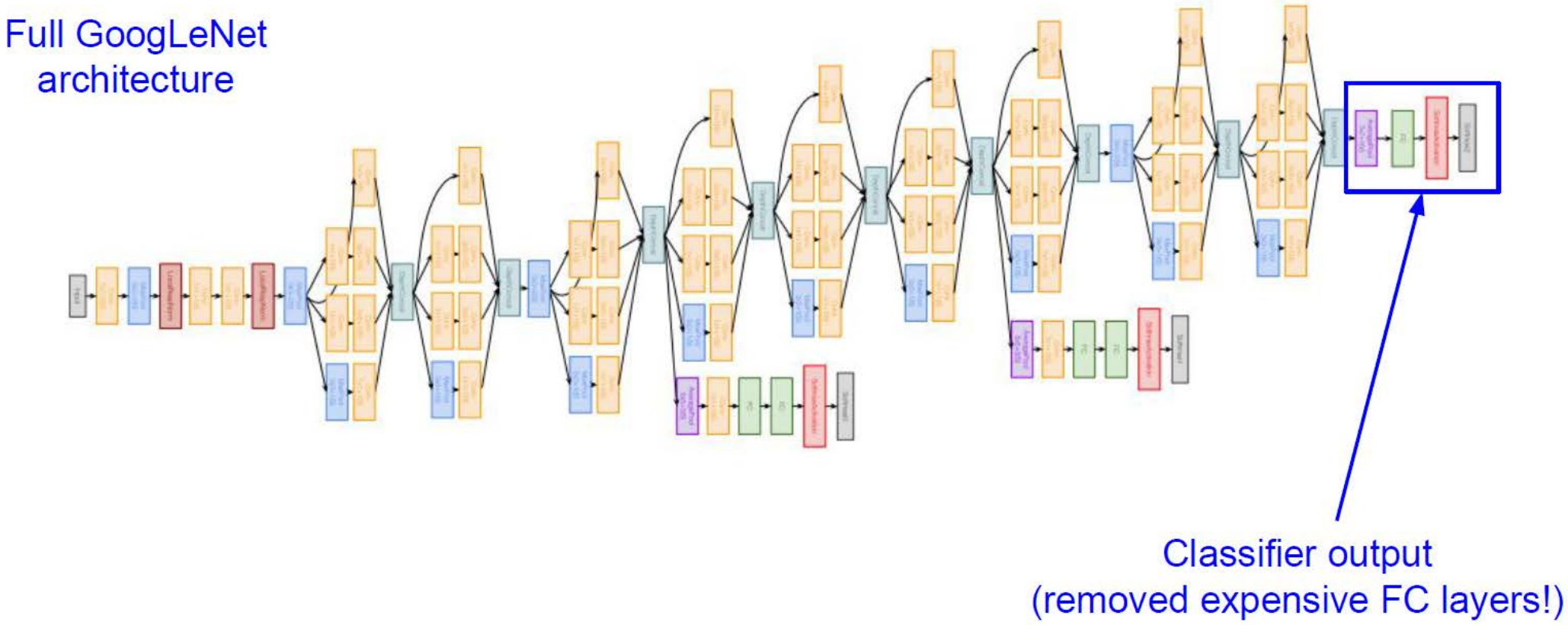
GoogLeNet Stacked Inception Modules

- 9 inception modules



GoogLeNet Classifier Output

- Global Average Pooling at the last inception module
  - Replace FC layers in previous CNN models

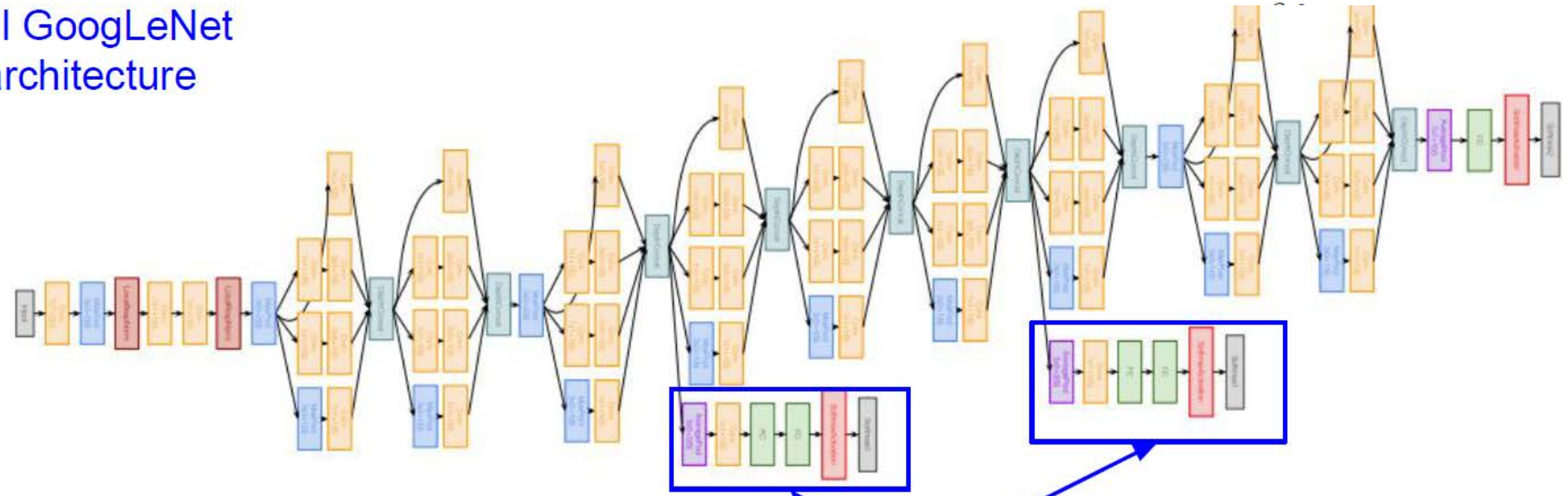


GoogLeNet Auxiliary Classification Outputs

- add intermediate classifiers to solve vanishing gradient problems
  - removed after training

$$\frac{\partial \mathcal{L}}{\partial w^l} = \frac{\partial \mathcal{L}}{\partial a^L} \cdot \frac{\partial a^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial a^{L-2}} \cdot \cdots \cdot \frac{\partial a^l}{\partial w^l}$$

Full GoogLeNet architecture



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

Inception v2~v5

- Inception v2 (2015)
  - batch normalization
- Inception v3 (2016)
  - filter factorization
- Inception v4 (2017)
  - ResNet + Inception
- v5: Xception (2017)
  - depthwise separable convolution

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

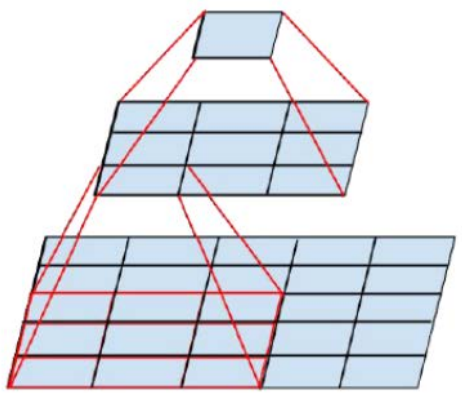
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

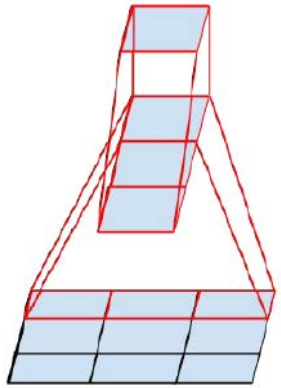
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

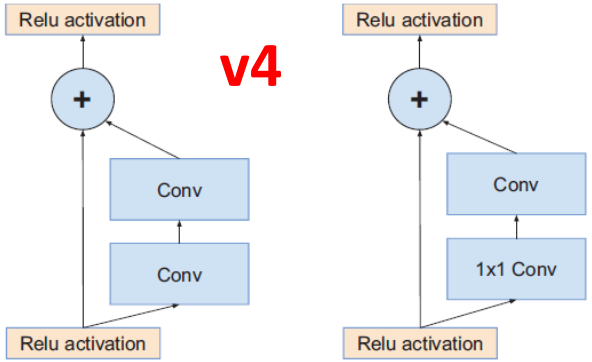
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

v2

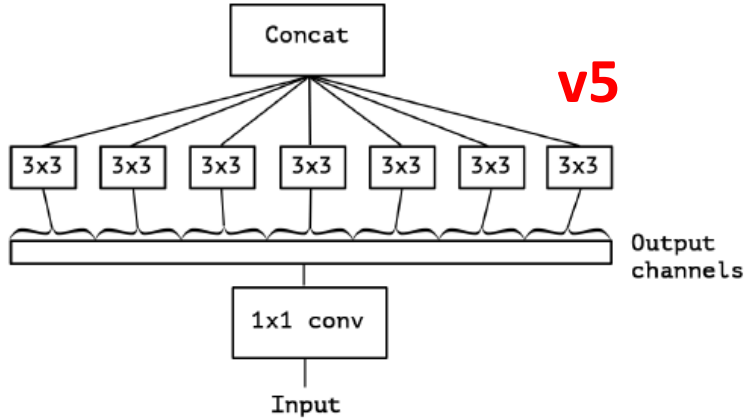


v3





v4



v5

(Inception v1): C. Szegedy, et al., "Going Deeper with Convolutions," CVPR, 2015.
(Inception v2): S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," ICML 2015.
(Inception v3): C. Szegedy et al., "Rethinking the Inception Architecture for Computer Visiion," CVPR, 2016.
(Inception v4): C. Szegedy, et al, "Inception-v4, Inception-ResNet and the Impact of Residual Connections in Learning," AAAI, 2017.
(Inception v5): F. Chollet, "Xception: Deep Learning with Datawise Separable Convolutions," CVPR, 2017.

Inception v2 (Batch Normalization)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

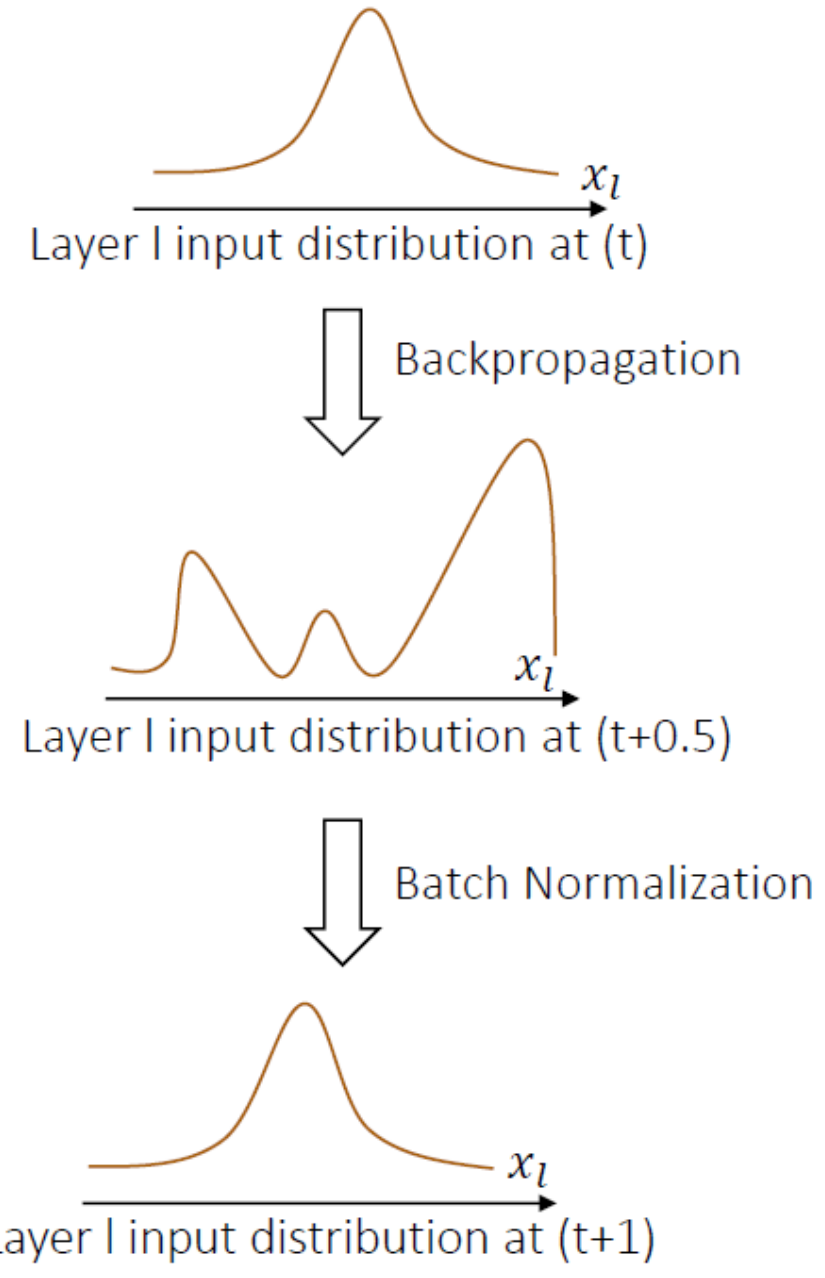
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

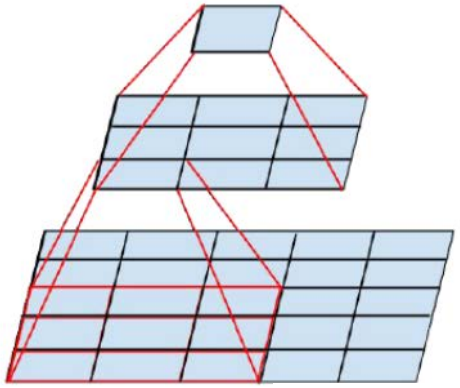
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

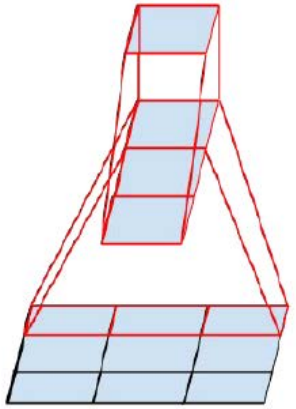
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

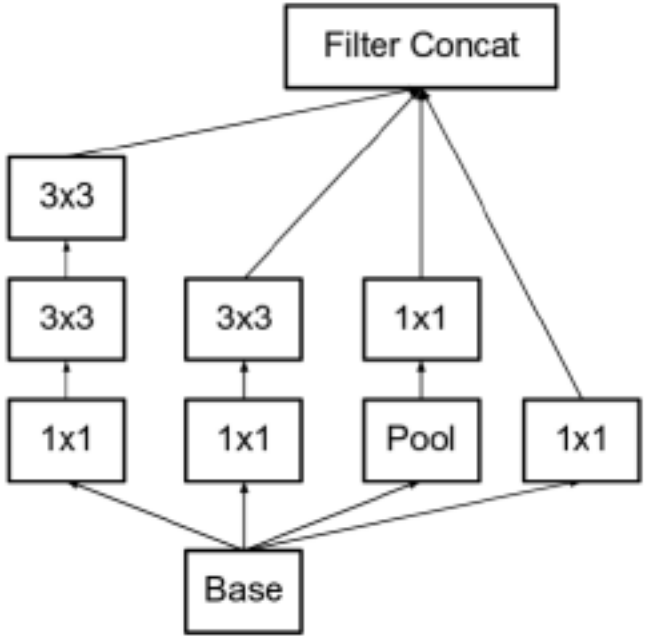


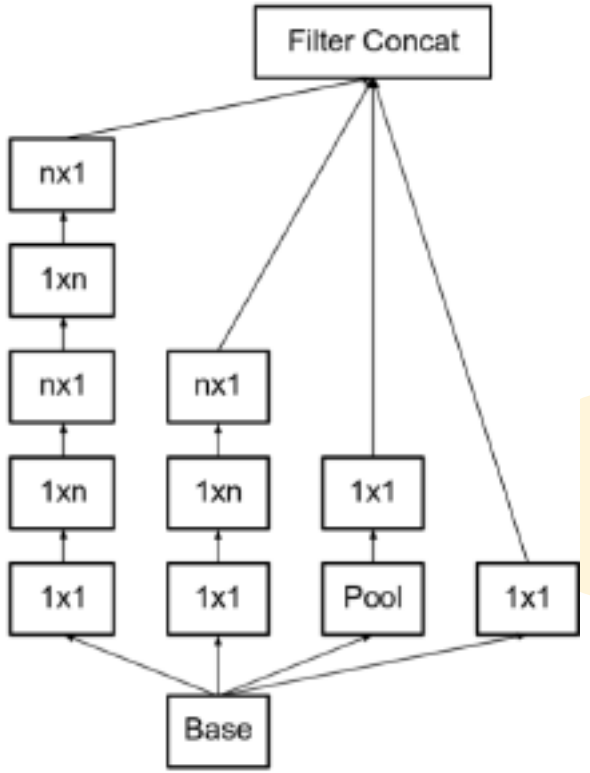
Inception v3 (factorization)

- factorize a large filter (e.g., 5x5) into several smaller filters (e.g., two 3x3)
- factorize a 3x3 filter into 3x1 and then 1x3
-



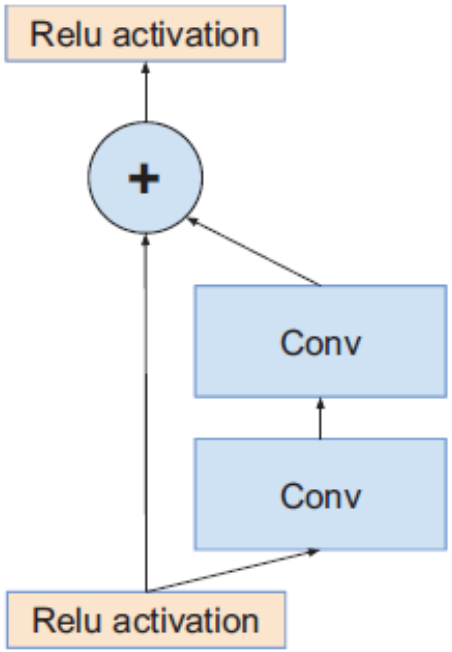


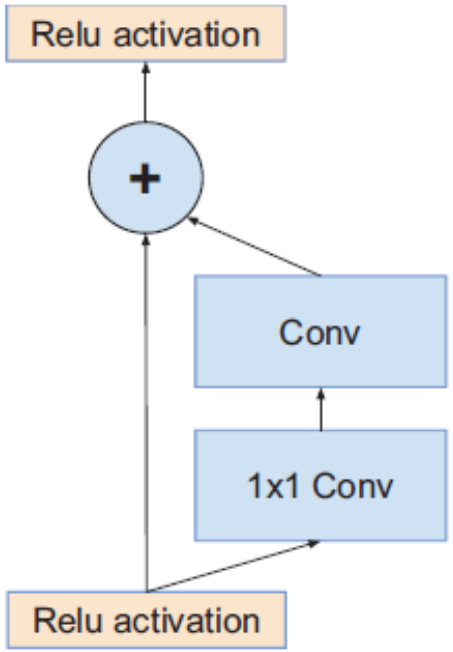


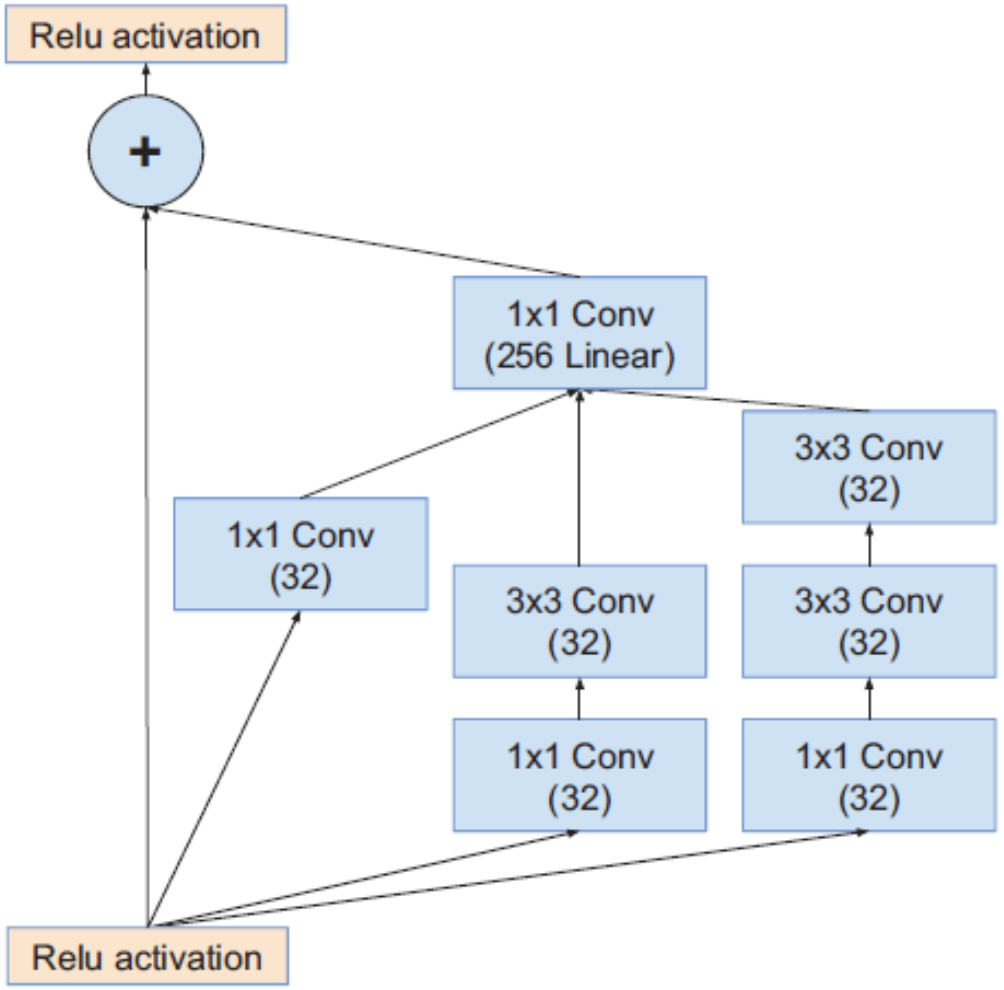


Inception v4

- Inception + ResNet

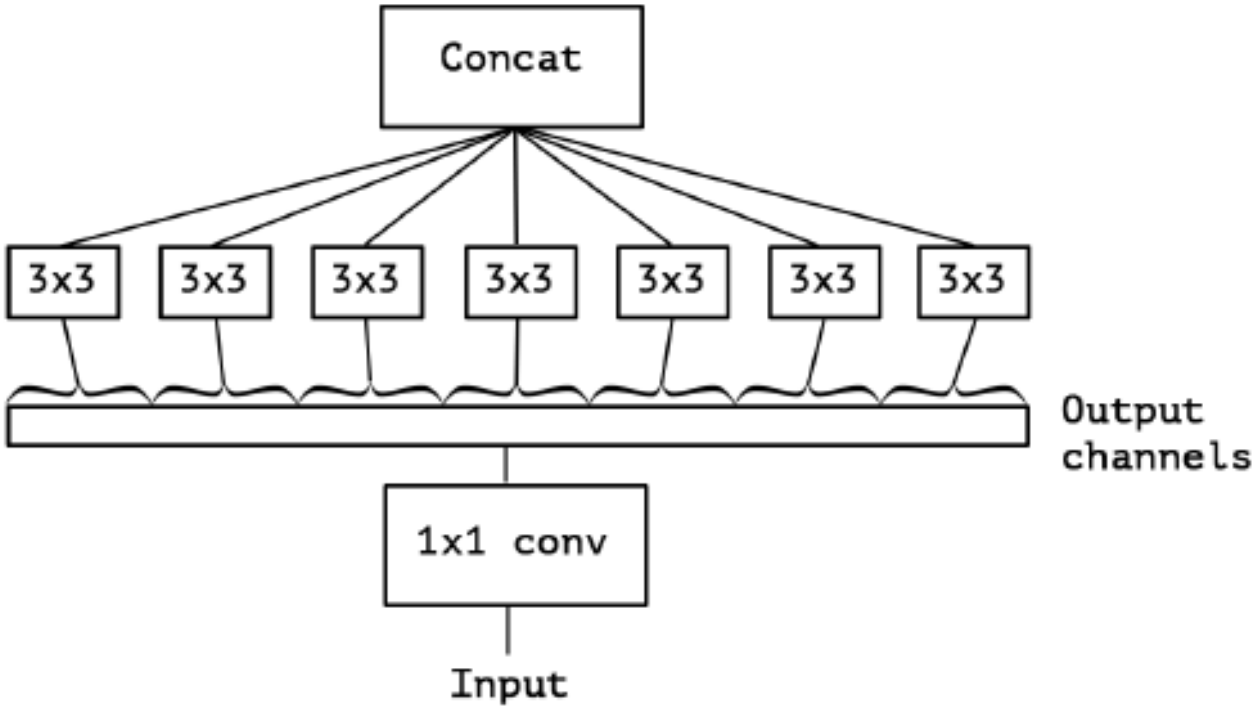






Xception (Inception v5)

- 1x1 convolution + depthwise convolution
  - cp. depthwise separable convolution = depthwise convolution + 1x1 convolution
- also group convolution (e.g., 7 groups below for output channels)



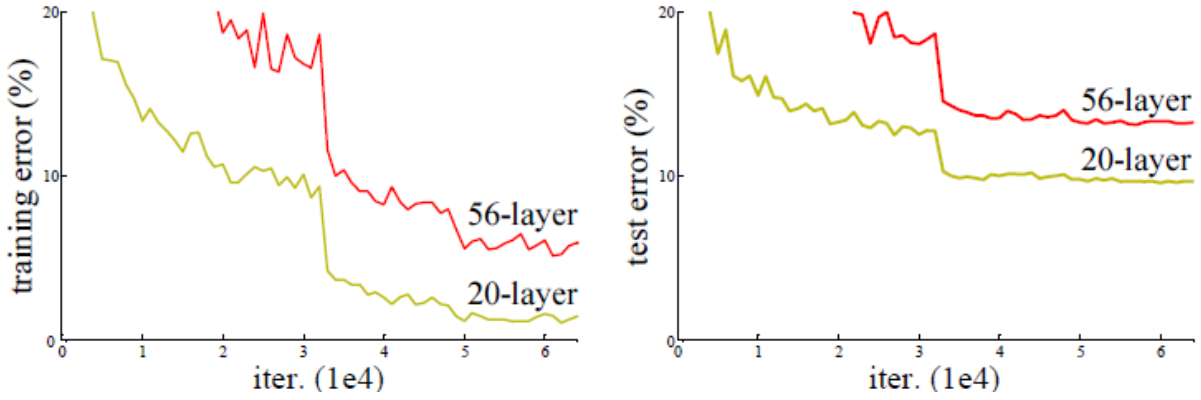
ResNet (1/3)

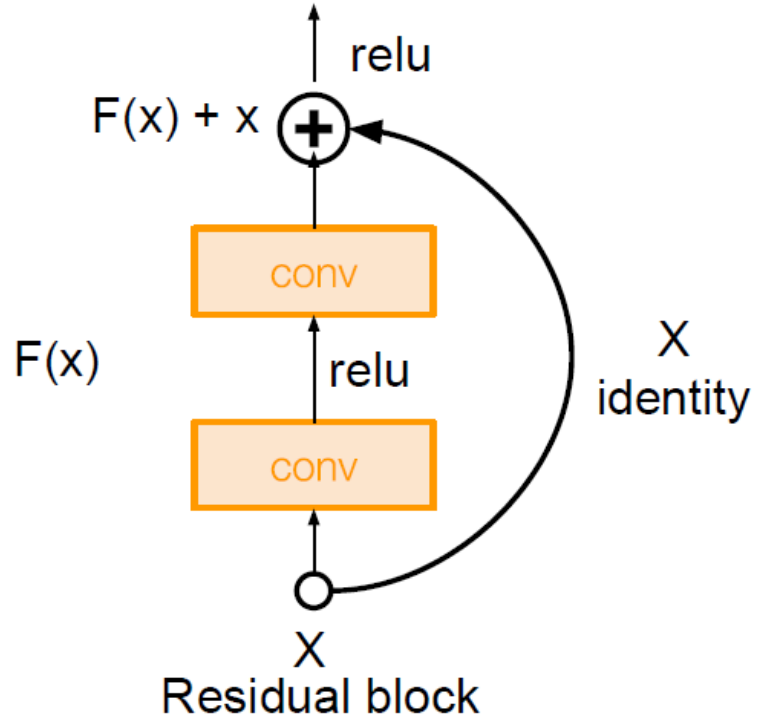
[He et al., 2015]

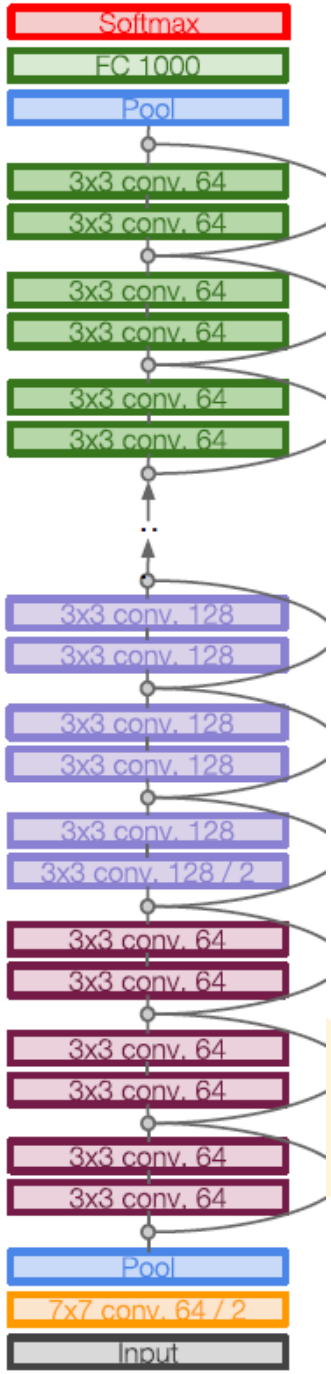
- avoid gradient vanishing problems for very deep NN

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- win over all classification and detection competitions in ILSVRC'15 and COCO'15!







* K. He, et al., "Deep Residual Learning for Image Recognition," CVPR< 2016. (Microsoft)

ResNet (2/3)

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- For deeper networks (ResNet-50+), use 1x1 "bottleneck" layer to reduce complexity
- Periodically, double # of filters and downsample spatially using stride 2
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

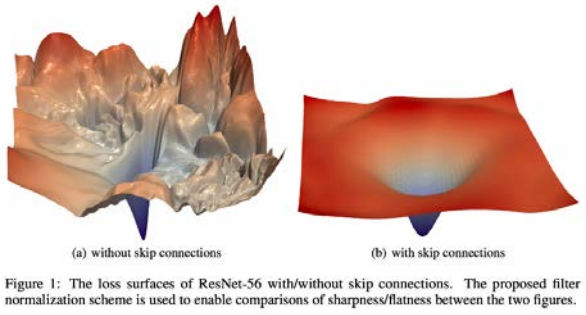
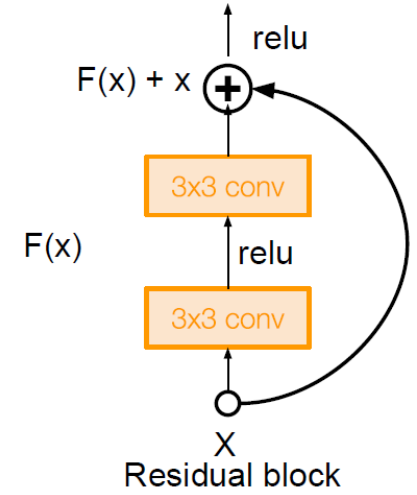
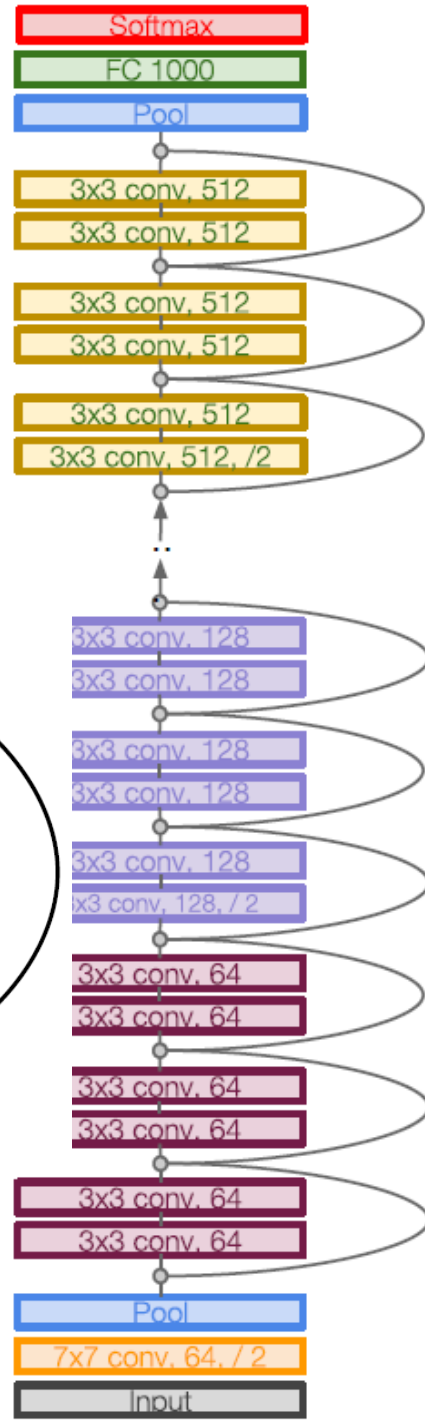
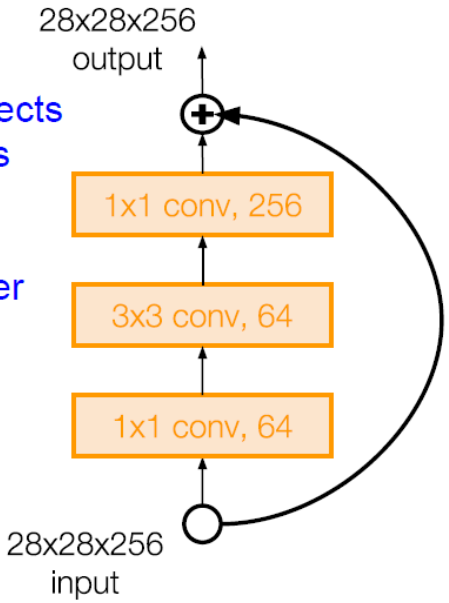


Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.



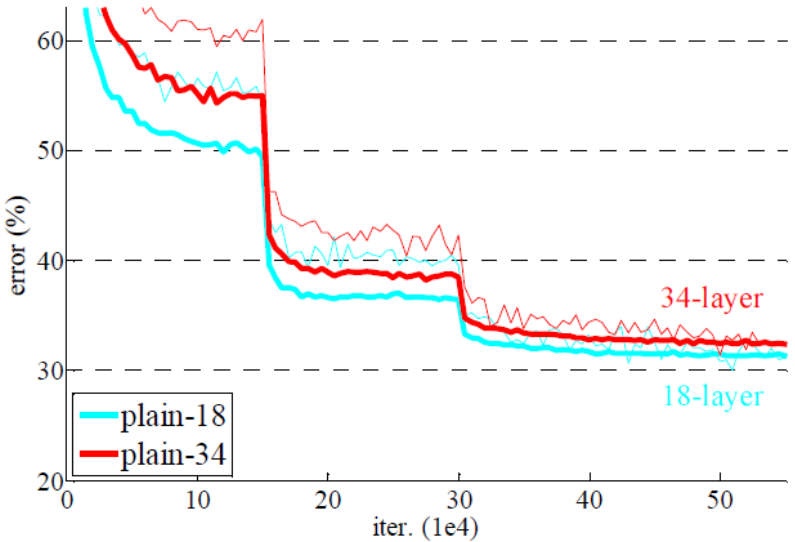


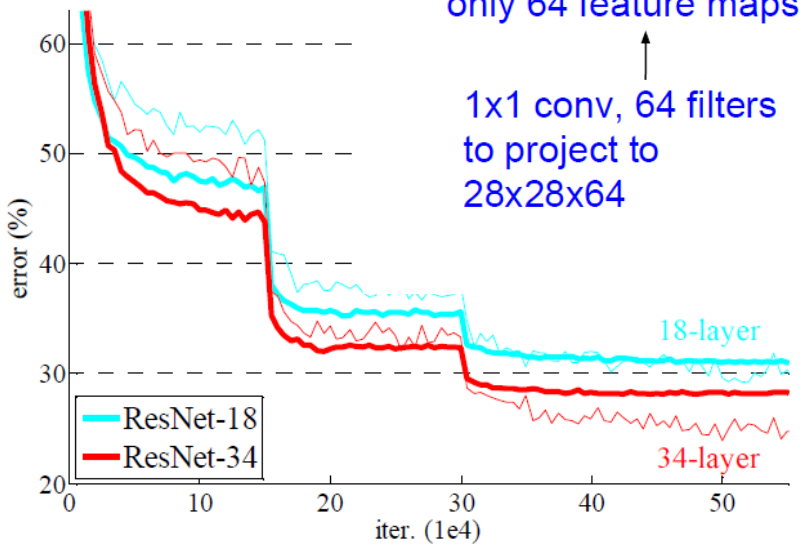


1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

1x1 conv, 64 filters to project to 28x28x64





ResNet (3/3)

- ResNet layers: 16, 34, 50, 101, 152
- Single model error rate: 4.49% (top-5)
- Ensemble error rate: 3.57% (top-5)
  - 6 independent models (including two 152-layer ones)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43†
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

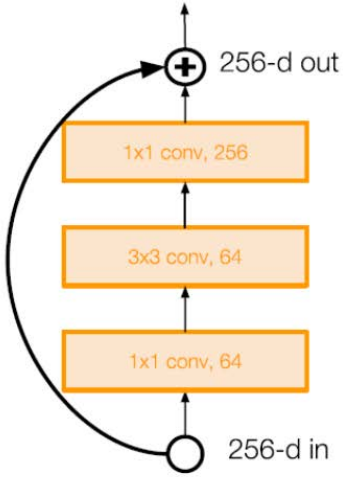
method	top-5 err. (**test**)
VGG [40] (ILSVRC'14)	7.32
GoogLeNet [43] (ILSVRC'14)	6.66
VGG [40] (v5)	6.8
PReLU-net [12]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

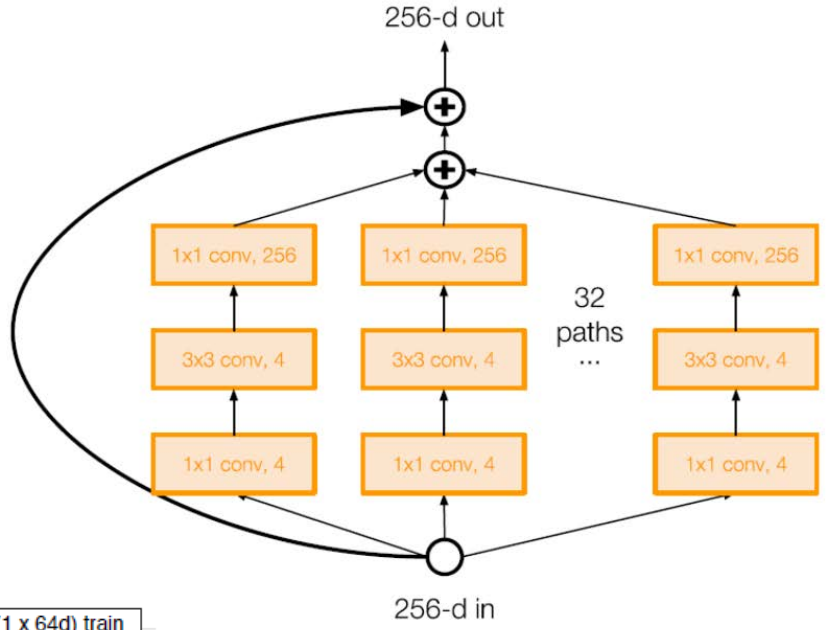
Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

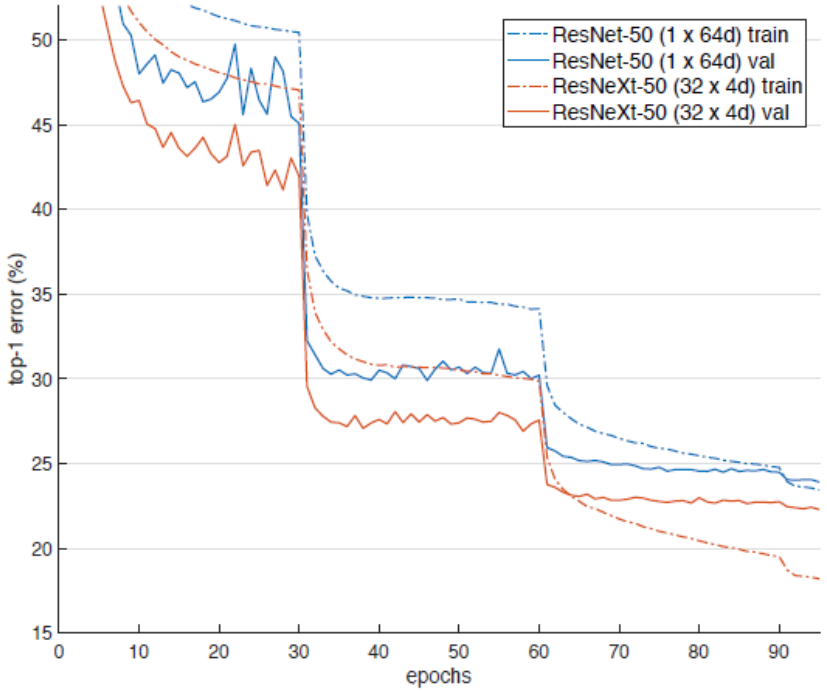
ResNeXt*

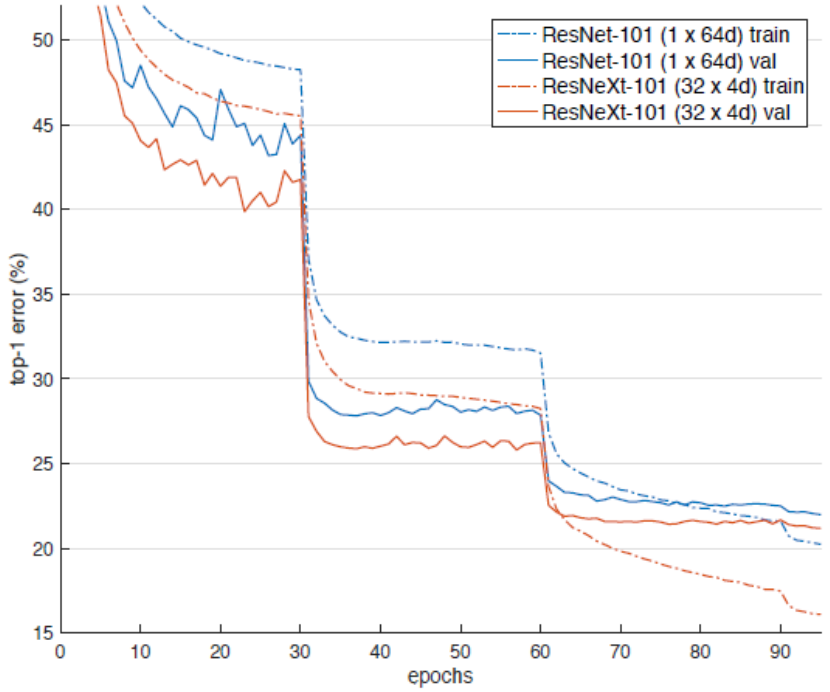
Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

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







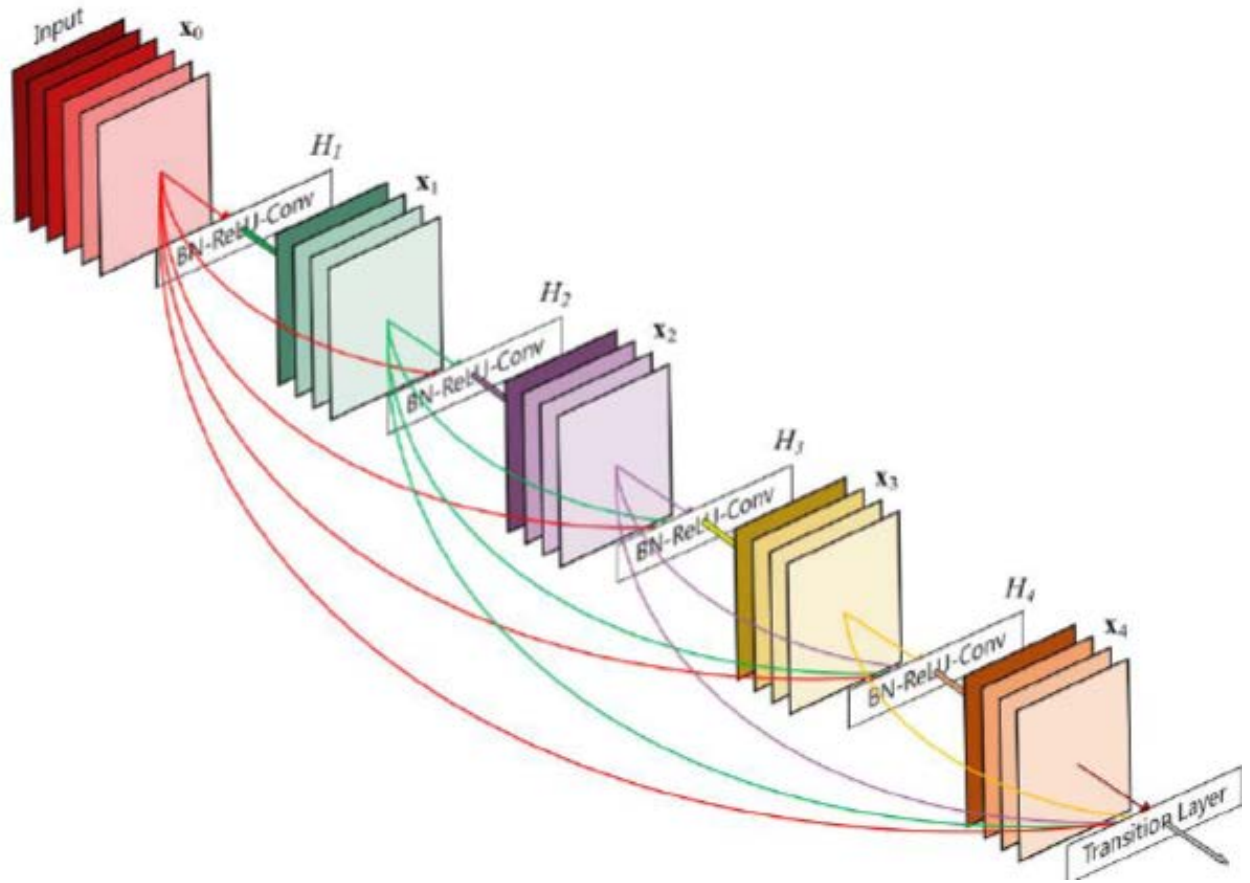


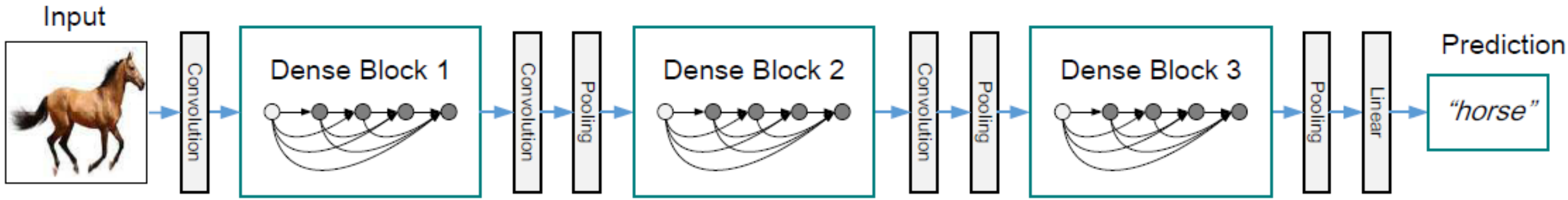
1st Runner Up in ILSVRC 2016 (Image Classification)

* S. Xie, et al., “Aggregated Residual Transformations for Deep Nerual Networks,” CVPR, 2017. (Facebook)

DenseNet

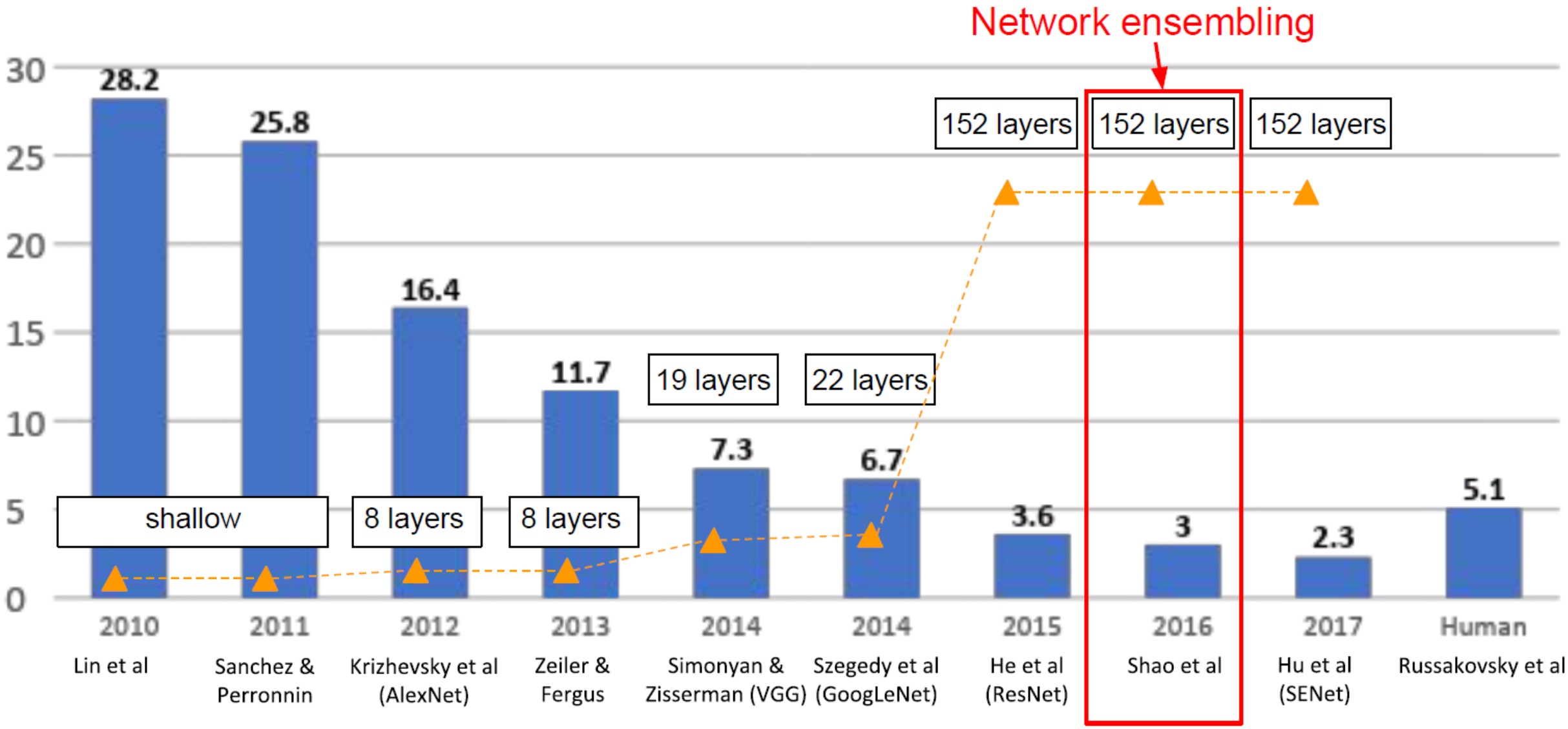
- add skip connections to multiple forward layers
- assume layer 1 captures edges, while layer 5 captures faces
- why not have layer that combines both faces and edges (e.g., to model a scarred face)
- standard CNN does not allow this
  - layer 6 combines only layer 5 patterns, not lower
- alleviate vanishing gradients, strengthen feature propagation, encourage feature reuse





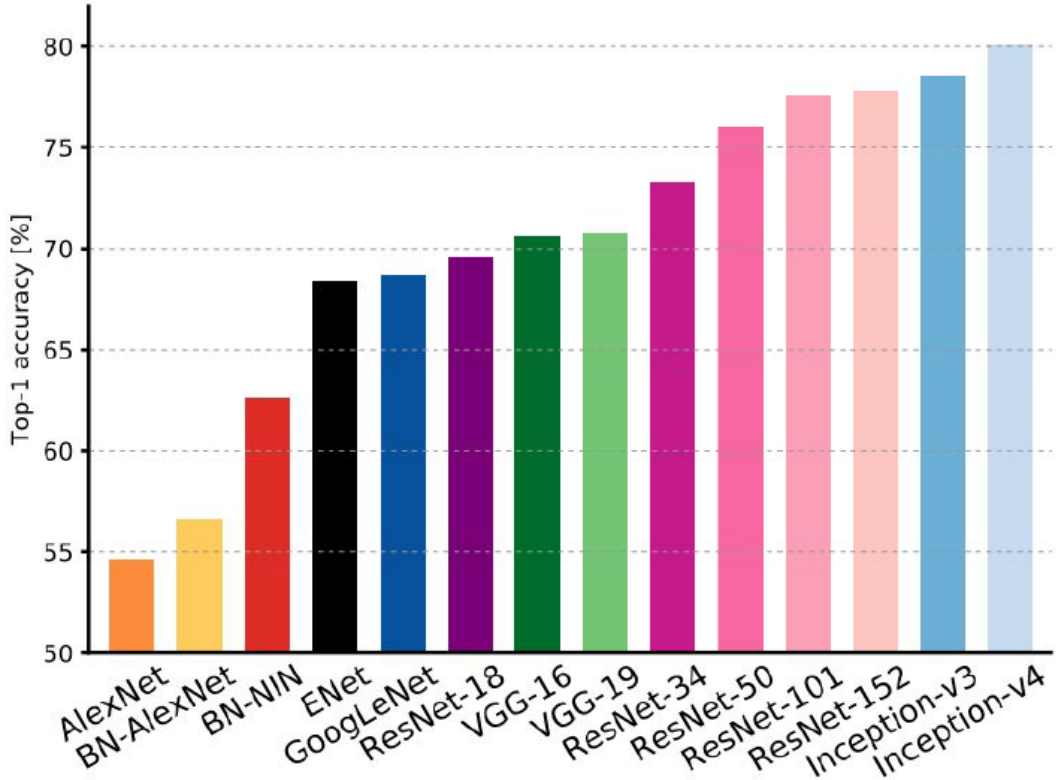
G. Huang, et al., "Densely Connected Convolutional Networks," CVPR, 2017.

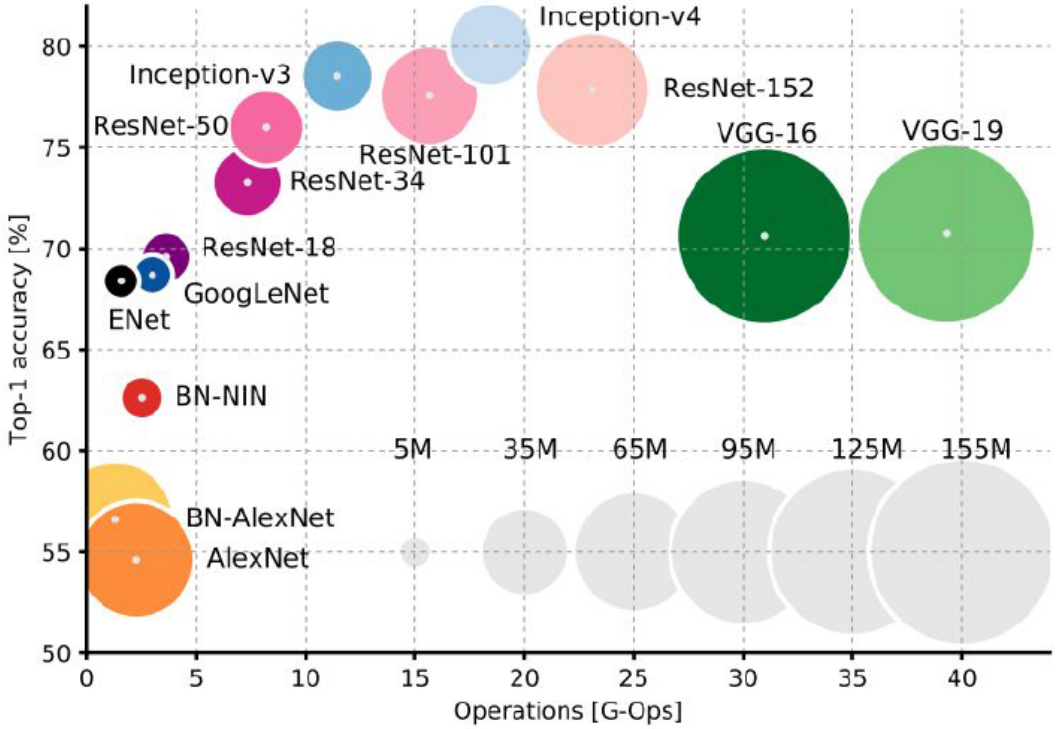
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Complexity Comparison

- VGG: highest memory and most operations
- GoogLeNet: most efficient
- AlexNet has smaller computation complexity, but still memory heavy, and lower accuracy
- ResNet: moderate efficiency depending on model depth, highest accuracy





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

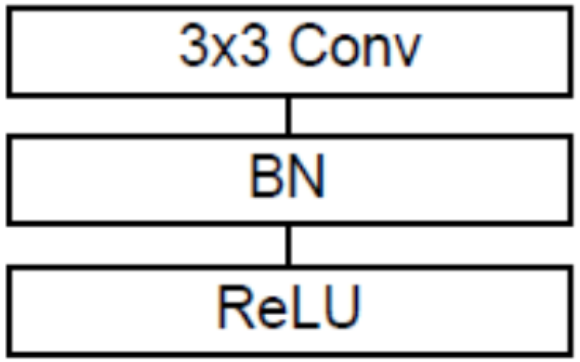
Summary of Popular CNN Models

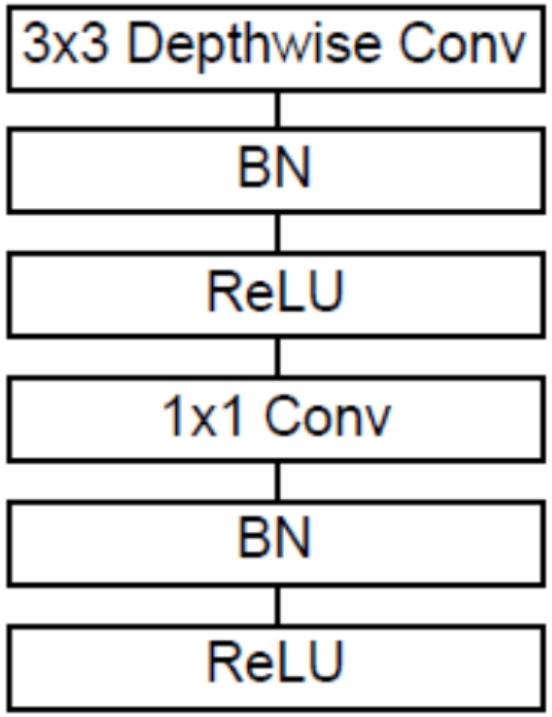
CNN		layers		Kernel size	# of parameters (M)		GOP		Top-1 val. error (%)	Top-5 val. error (%)
		Conv	FC		Conv	FC	Conv	FC		
AlexNet (2012)		5	3	3x3 5x5 11x11	60		0.67	0.05	37.5	17.0
VGG (2014)	11	8	3	3x3	133		6.5	0.12	29.6	10.4
	13	10	3	3x3	133		10.2	0.12	28.7	9.9
	16	13	3	1x1 3x3	138		15.3	0.12	27.0	8.8
	19	16	3	3x3	144		16.8	0.12	27.3	9.0
GoogLeNet (2015)		22		1x1 3x3 5x5 7x7	5.8	1	1.5	0.001	-	9.15
ResNet (2016)	20	19	1	1x1 3x3 7x7	0.27		1.8		-	8.75
	32	31	1		0.46		3.6		-	7.51
	56	55	1		0.85		3.86		-	6.97
	110	109	1		1.7		7.6		-	6.43
	1202	1201	1		19.4		11.3		-	7.93
ResNeXt (2016)	50	49	1	1x1 3x3 7x7	25		4.2		24.4	6.6
	101	100	1		44.3		7.99		22.2	5.7
PyramidNet (2017)	110 α=48	109	0	1x1 3x3	1.7		-		-	4.58
	110 α=84	109	0		3.8		-		-	4.26
	110 α=270	109	0		28.3		-		-	3.73
	164 α=270 bottle neck	163	0		27.0		-		-	3.48

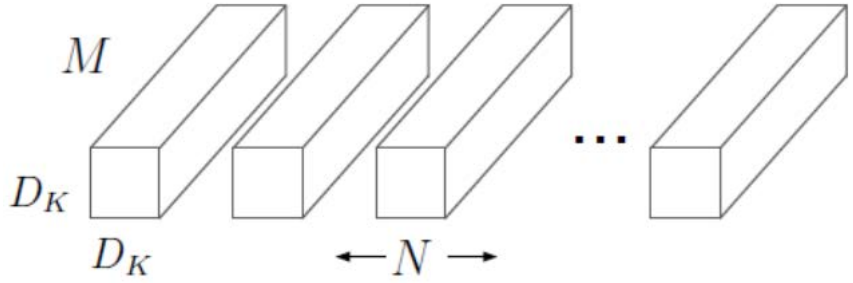
Light CNN Models: MobileNet*

- standard convolution =
  depthwise convolution (DWC)

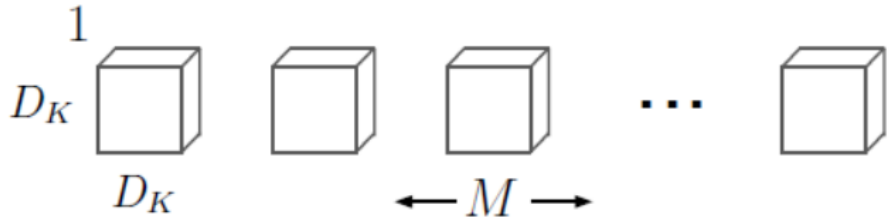
+ 1x1 convolution (pointwise convolution)



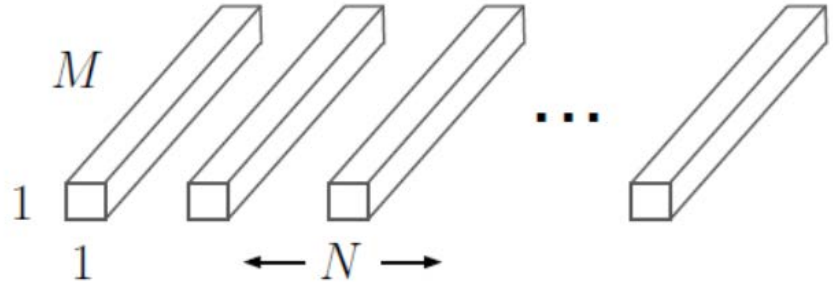




(a) Standard Convolution Filters



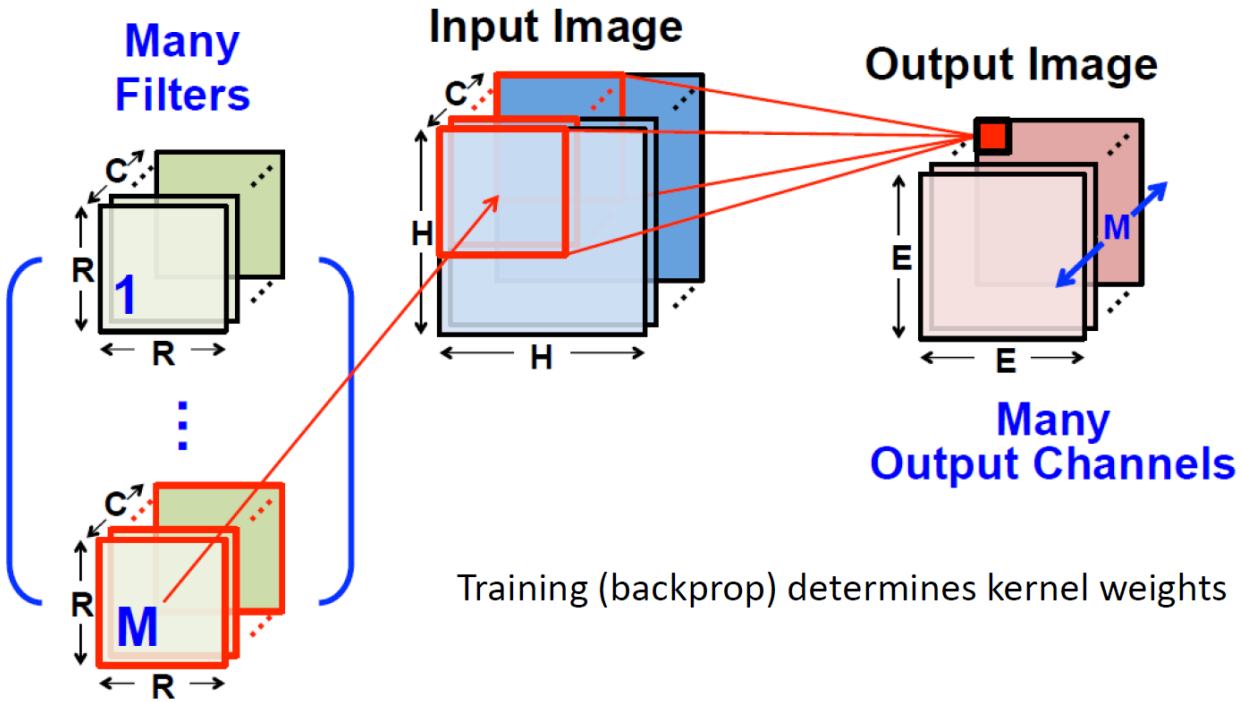
(b) Depthwise Convolutional Filters



* A. G. Howard, et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017. (Google)

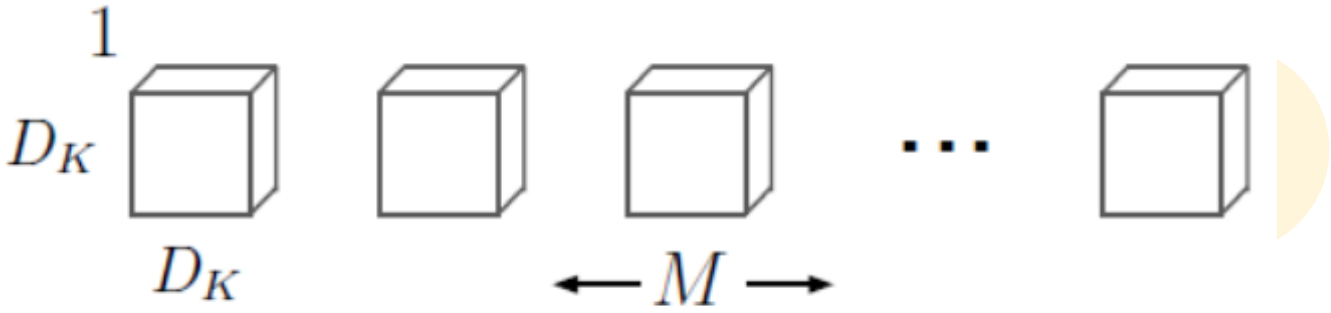
Standard vs. Depthwise convolution

- Standard convolution
  - C*M 2D filters
  - Convolution across all C input channels using 3D filters of size RxRxC
  - M 3D filters, one for each output channels



Training (backprop) determines kernel weights

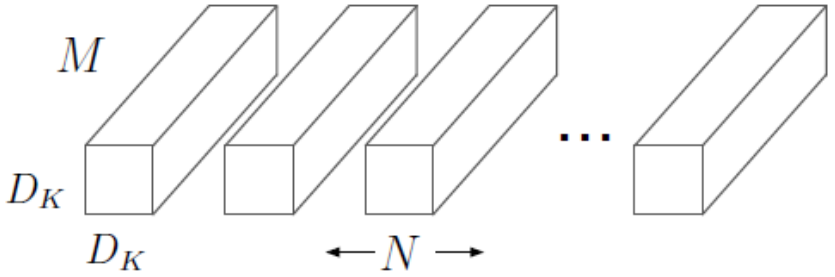
- Depthwise convolution
  - M 2D filters (M=C)
  - No convolution across input channels, i.e, no 3D filters



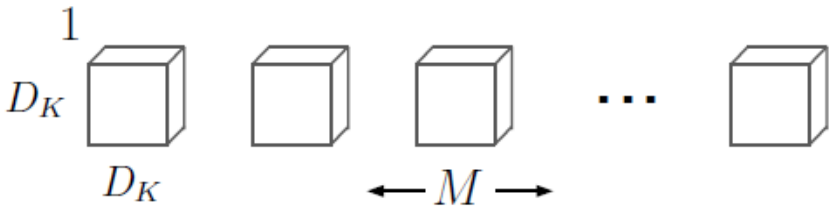
Depthwise Separable Convolution

- standard convolution
  - M input channels, N output channels
  - D_K x D_K filter kernel, feature size: D_F*D_F
  - # of parameters=D_K*D_K*M*N
  - # of operations=D_K*D_K*M*N*D_F*D_F
- depthwise convolution (DWC)
  - convolution of one input channel, generating one output channel
  - # of output channels = # of input channels
  - # of parameters=D_K*D_K*M
  - # of operations= D_K*D_K*M*D_F*D_F
- pointwise convolution (PWC)
  - 1x1 filters combining M input channels, generating N output channels
  - # of parameters=M*N
  - # of operations=M*N*D_F*D_F
- DSC = DWC + PWC
  - total # of parameters=D_K*D_K*M+M*N
  - total # of operations= D_K*D_K*M*D_F*D_F+M*N*D_F*D_F
  - e.g., M=128, N=128, D_K=3, D_F =64
  - standard convolution
    - total # of parameters=147,456
    - total # of operations= **604M**
  - DSC
    - total # of parameters=17,536 (cp. 147,456)
    - total # of operations=**72M** (cp. 604M)

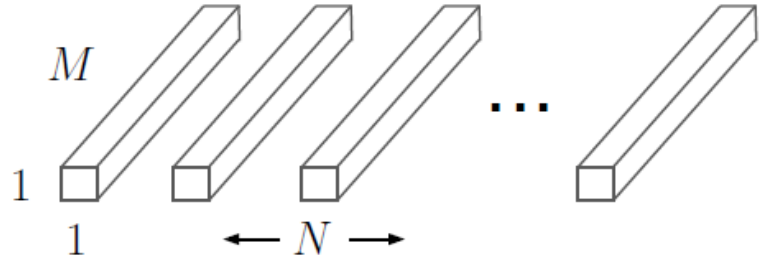
standard convolution



DWC



1x1 convolution



MobileNet

- width multiplier
  - reduced channel number
- resolution multiplier
  - reduced channel size

Table 8. MobileNet Comparison to Popular Models

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

Table 10. MobileNet for Standford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3	84%	5000	23.2
1.0MobileNet-224	83.3%	569	3.3
0.75MobileNet-224	81.9%	325	1.9
1.0MobileNet-192	81.9%	418	3.3
0.75MobileNet-192	80.5%	239	1.9

Type / Stride	Filter Shape	Input Size
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw / s1	3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32
Conv dw / s2	3 x 3 x 64 dw	112 x 112 x 64
Conv / s1	1 x 1 x 64 x 128	56 x 56 x 64
Conv dw / s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 128	56 x56 x128
Conv dw / s2	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	28 x28 x128
Conv dw / s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 256
Conv dw / s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
5x **Conv dw / s 1** **Conv / s1**	3 x 3 x 512 dw 1 x 1 x 512 x 512	14 x 14 x 512 14x14x512
Conv dw / s2	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s2	3 x 3 x 1024 dw	7 x 7 x 1024
Conv/ s1	lx 1 x 1024 x1024	7x 7 x 1024
Avg Pool / s1	Pool 7 x 7	7 x 7 x 1024
FC/s1	1024 x 1000	1 x 1 x1024
Softmax / s1	Classifier	1 x 1 x 1000

MobileNet Result

- object detection using MobileNet SSD (Single Shot Detection)
  - COCO dataset

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8
Faster-RCNN 300	VGG	22.9%	64.3	138.5
	Inception V2	15.4%	118.2	13.3
	MobileNet	16.4%	25.2	6.1
Faster-RCNN 600	VGG	25.7%	149.6	138.5
	Inception V2	21.9%	129.6	13.3
	Mobilenet	19.8%	30.5	6.1

ShuffleNet Comparison

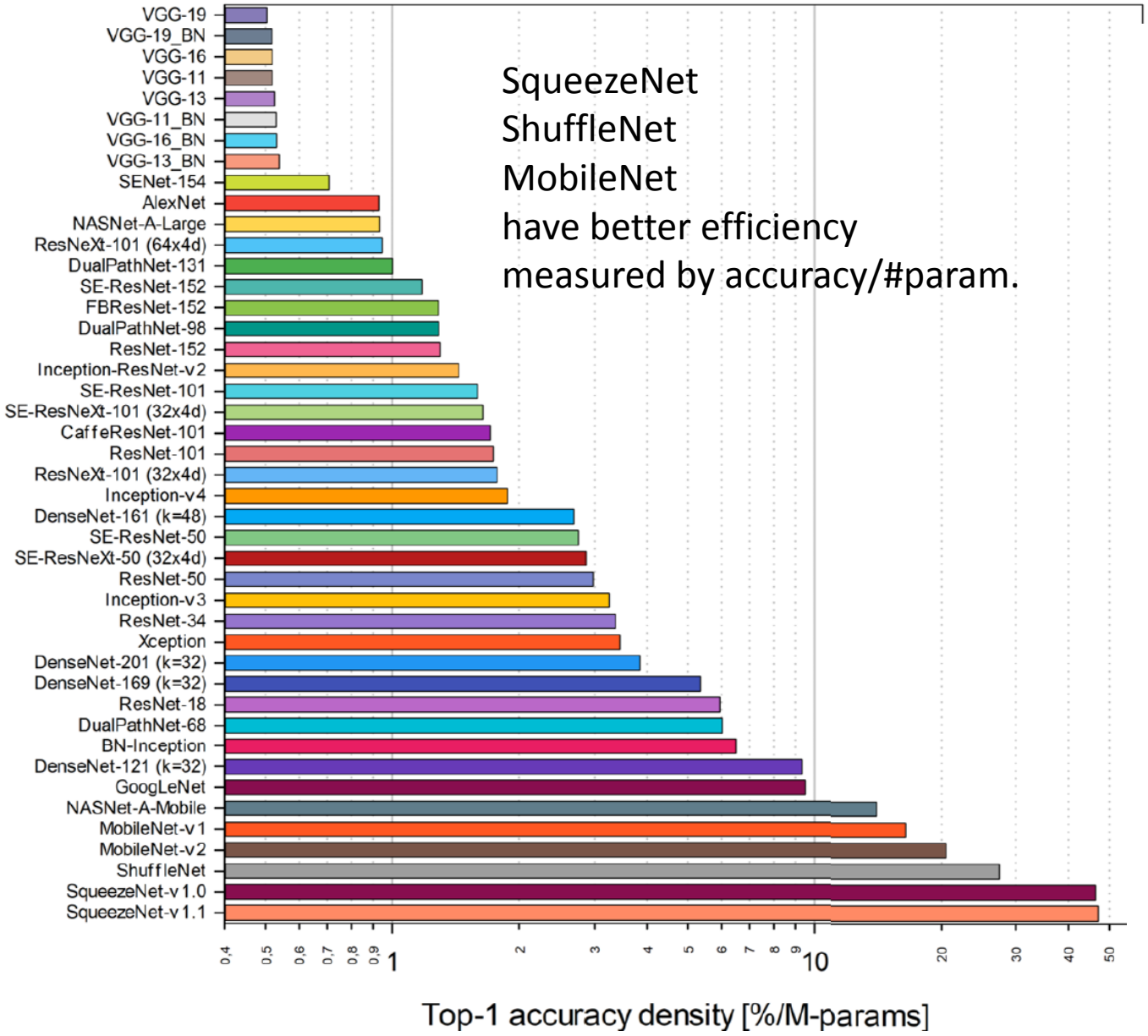
Table 4: Classification error vs. various structures (%, *smaller number represents better performance*)

Complexity (MFLOPs)	VGG-like[4]	ResNet	Xception-like	ResNeXt	ShuffleNet (ours)
140	56.0	38.7	35.1	34.3	**34.1** ($1\times$, $g = 3$)
38	-	48.9	46.1	46.3	**43.7** ($0.5\times$, $g = 4$)
13	-	61.6	56.7	59.2	**53.7** ($0.25\times$, $g = 8$)
40 (arch2)	-	48.5	45.7	47.2	**42.7** ($0.5\times$, $g = 8$)
13 (arch2)	-	61.3	56.5	61.0	**53.3** ($0.25\times$, $g = 8$)

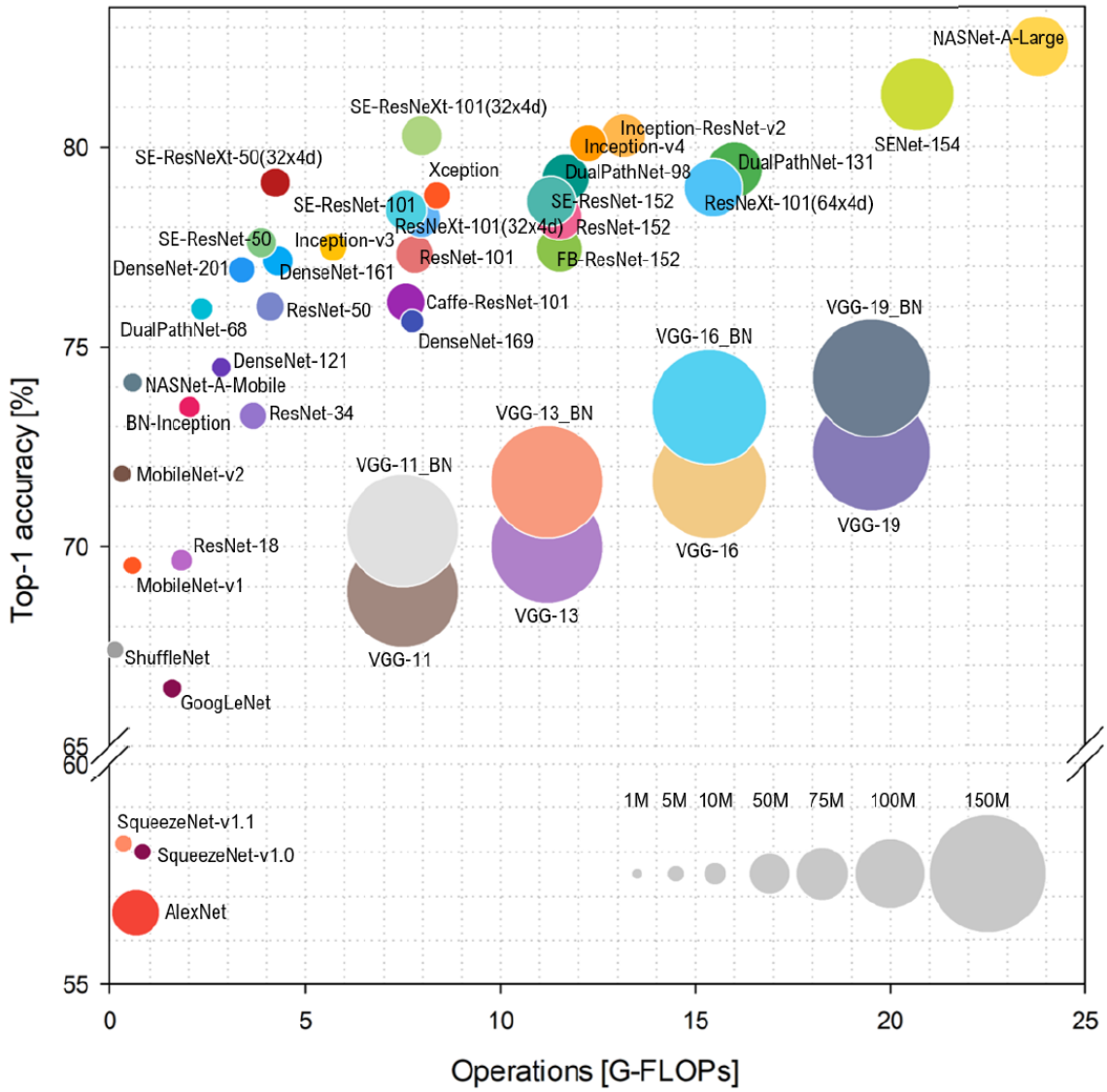
Table 5: ShuffleNet vs. MobileNet [12] on ImageNet Classification

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet $2\times$ ($g = 3$)	524	**29.1**	0.3
0.75 MobileNet-224	325	31.6	-
ShuffleNet $1.5\times$ ($g = 3$)	292	**31.0**	0.6
0.5 MobileNet-224	149	36.3	-
ShuffleNet $1\times$ ($g = 3$)	140	**34.1**	2.2
0.25 MobileNet-224	41	49.4	-
ShuffleNet $0.5\times$ (arch2, $g = 8$)	40	**42.7**	6.7
ShuffleNet $0.5\times$ (shallow, $g = 3$)	40	45.2	4.2

Accuracy vs. Complexity



SqueezeNet
ShuffleNet
MobileNet
have better efficiency
measured by accuracy/#param.



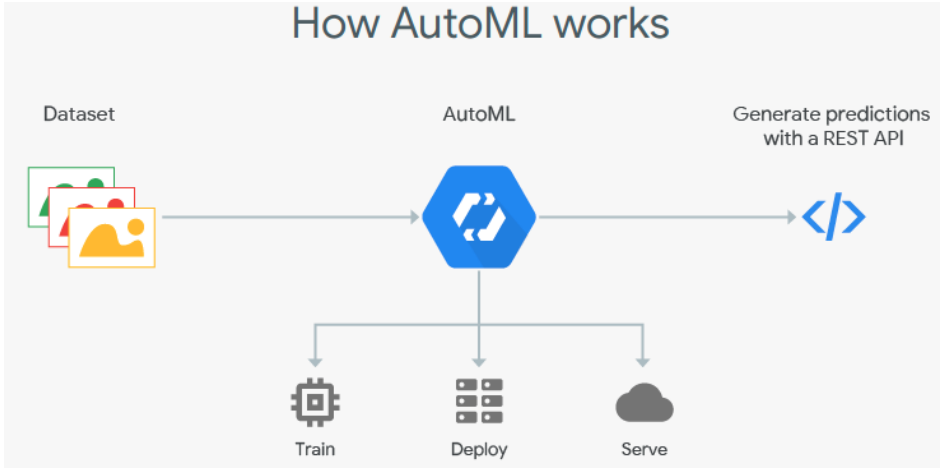
S. Bianco, et al, “Benchmark Analysis of Representative Deep Neural Network Architectures,” IEEE Access, Nov. 2018.

Neural Architecture Search (NAS)

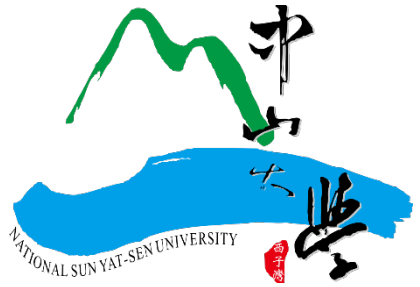
- NAS is an algorithm that automatically *searches* for the best *neural network architecture given data and building blocks*
  - Google takes 3~4 days using 450 GPUs

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

- Progressive NAS (PNAS)
  - test out blocks and search for structures in order of increasing complexity
- Efficient NAS (ENAS)
  - forces all models to share weights instead of training from scratch toconvergence
    - concept of transfer learning for blocks that were trained before
- Could AutoML
  - upload your data to Google Cloud AutoML
  - Google's NAS finds neural networks for you
  - but, cannot export your model when it is trained
  - you have to use their API to run your model on cloud



Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)

- remember previous state, e.g., in speech and image captioning

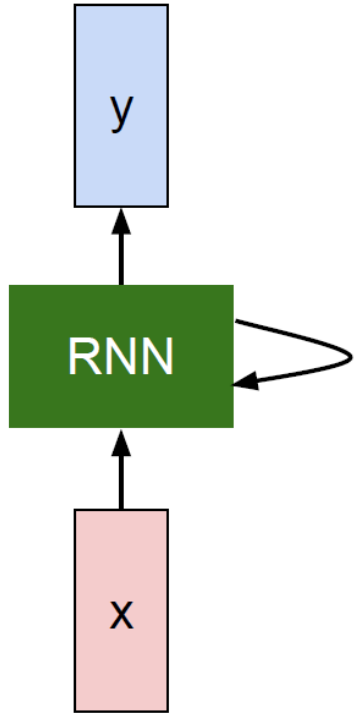
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

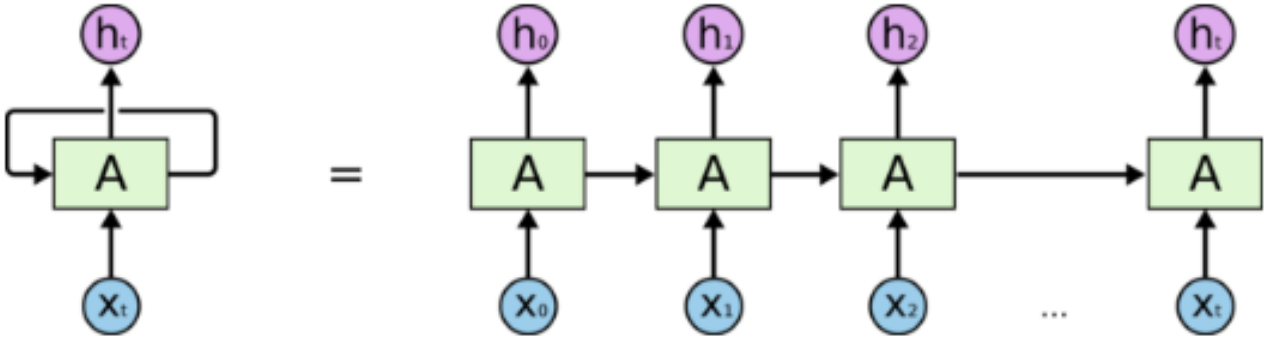
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

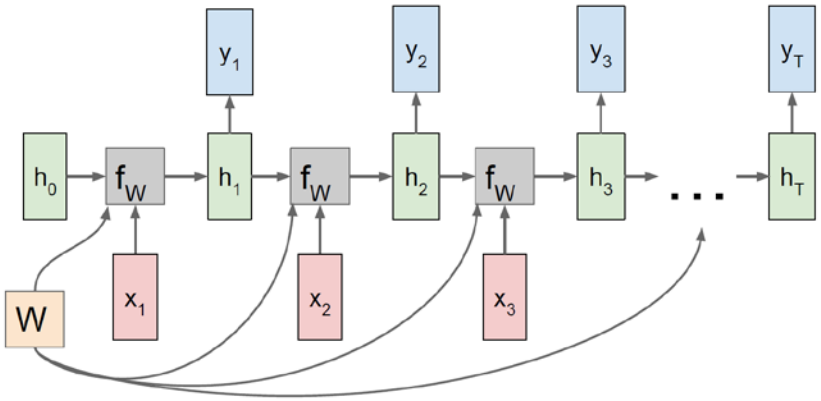
$$y_t = W_{hy}h_t$$



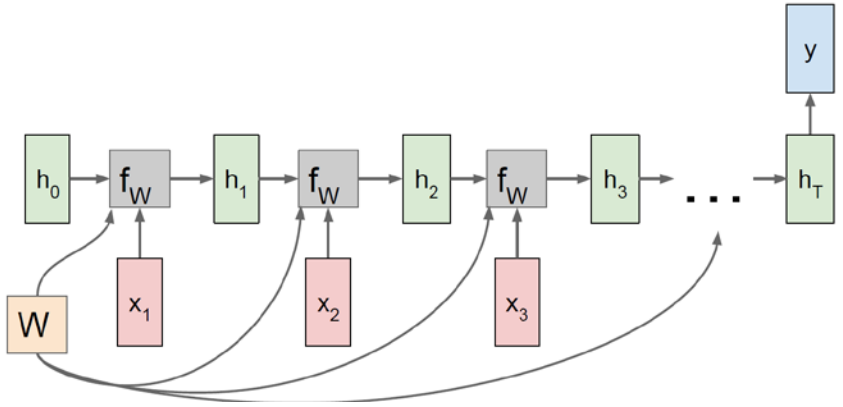


RNN unrolled in time

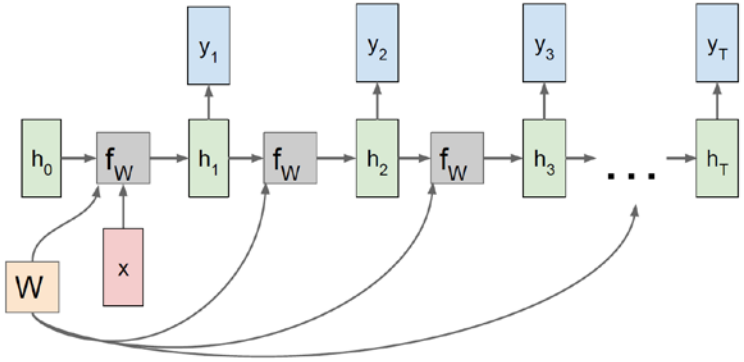
many to many
(e.g., machine translation)



many to one
(e.g., speaker recognition)



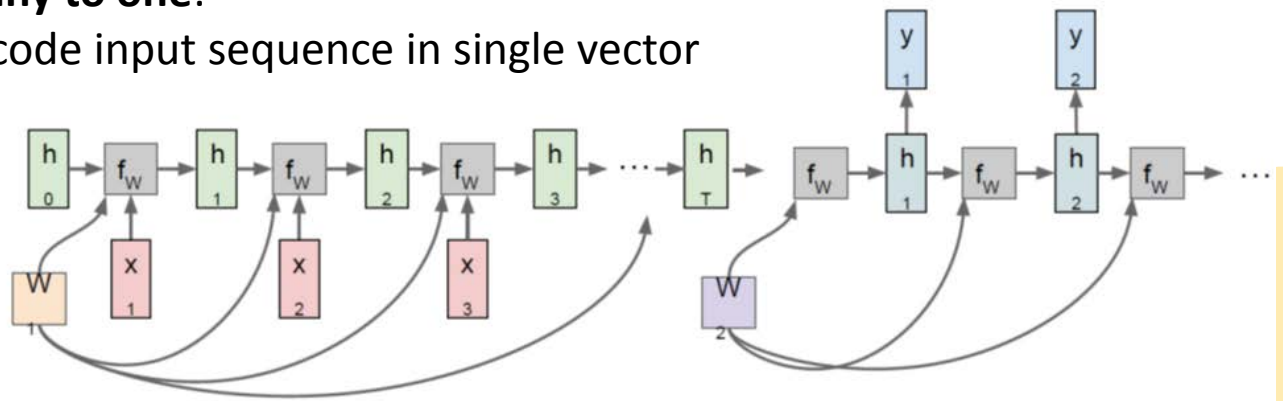
one to many
(e.g., image captioning)



many to one +
one to many
(e.g., autoencoder)

Many to one:
Encode input sequence in single vector

One to many :
Produce output sequence from single input vector

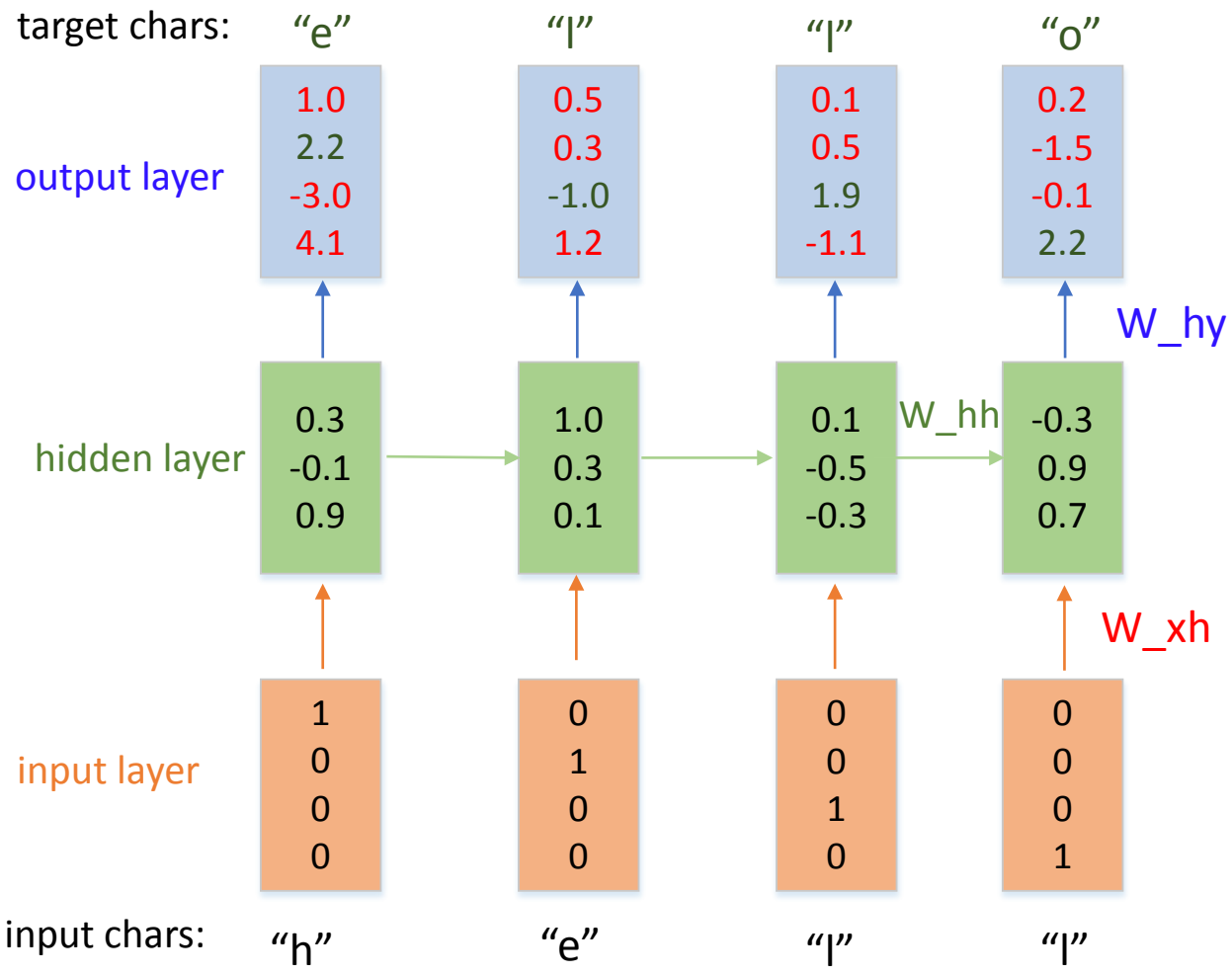


Example: Character-Level Language Model

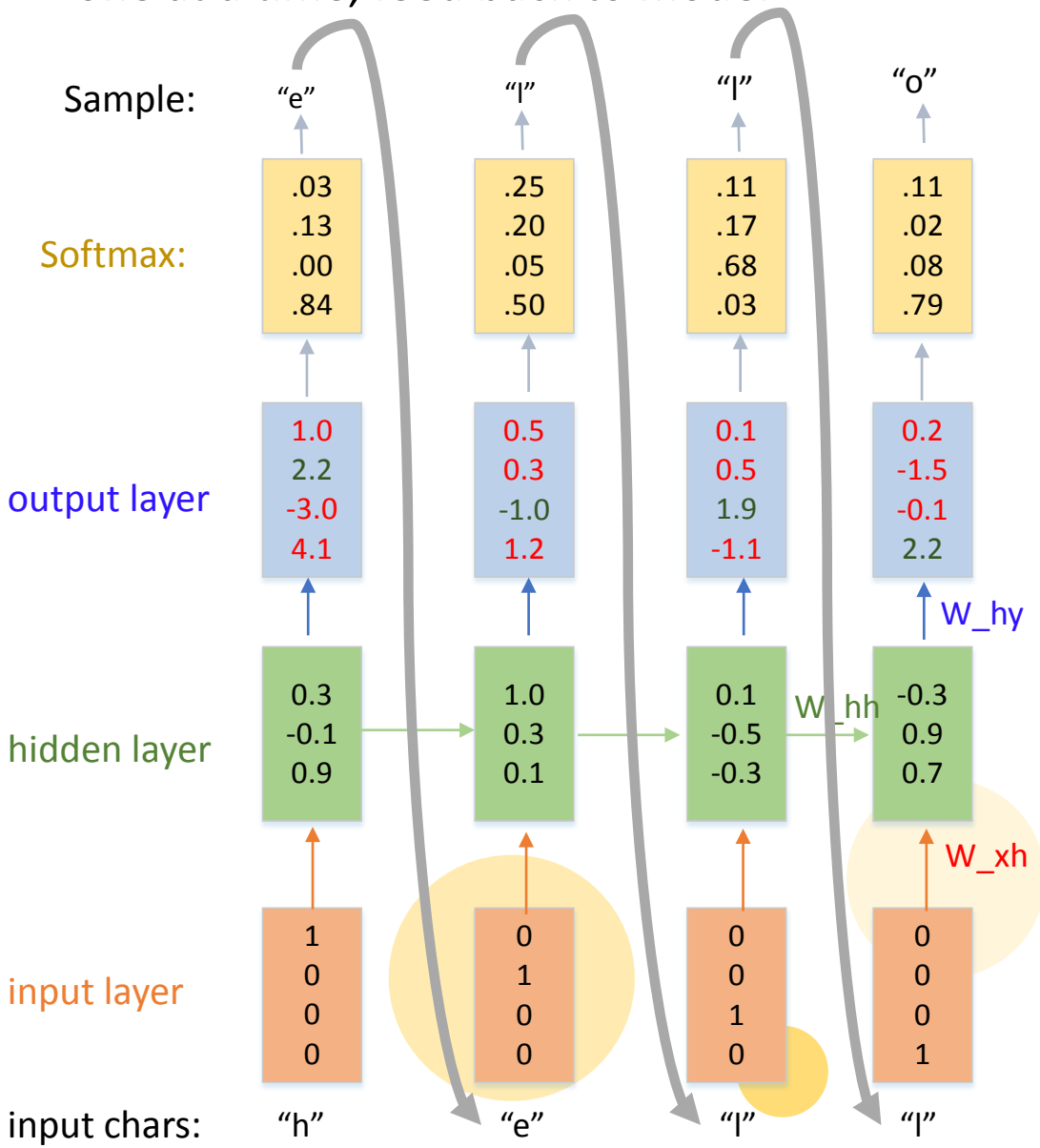
Vocabulary: {h,e,l,o}

training sequence “hello”

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

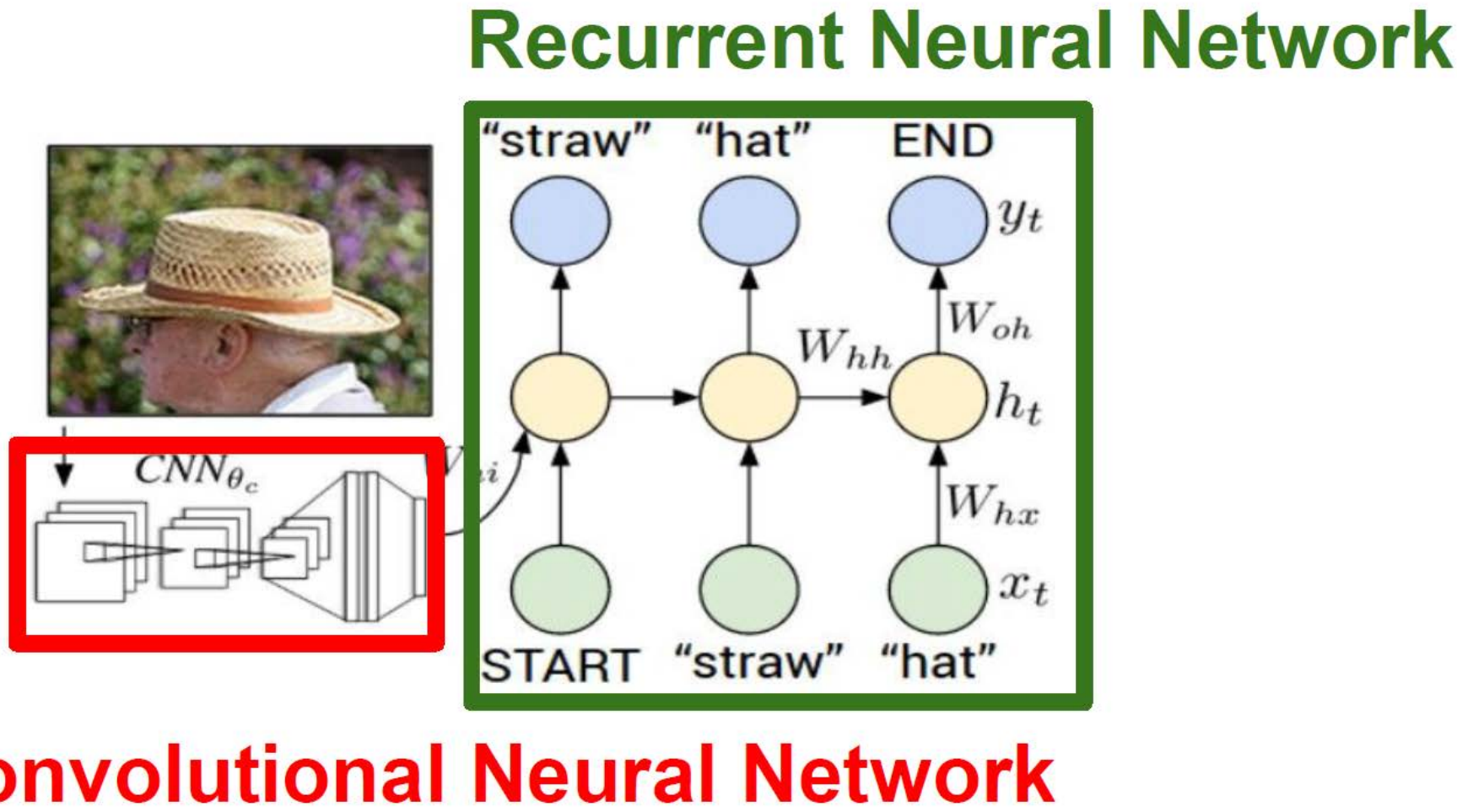


at test-time, sample characters one at a time, feed back to model



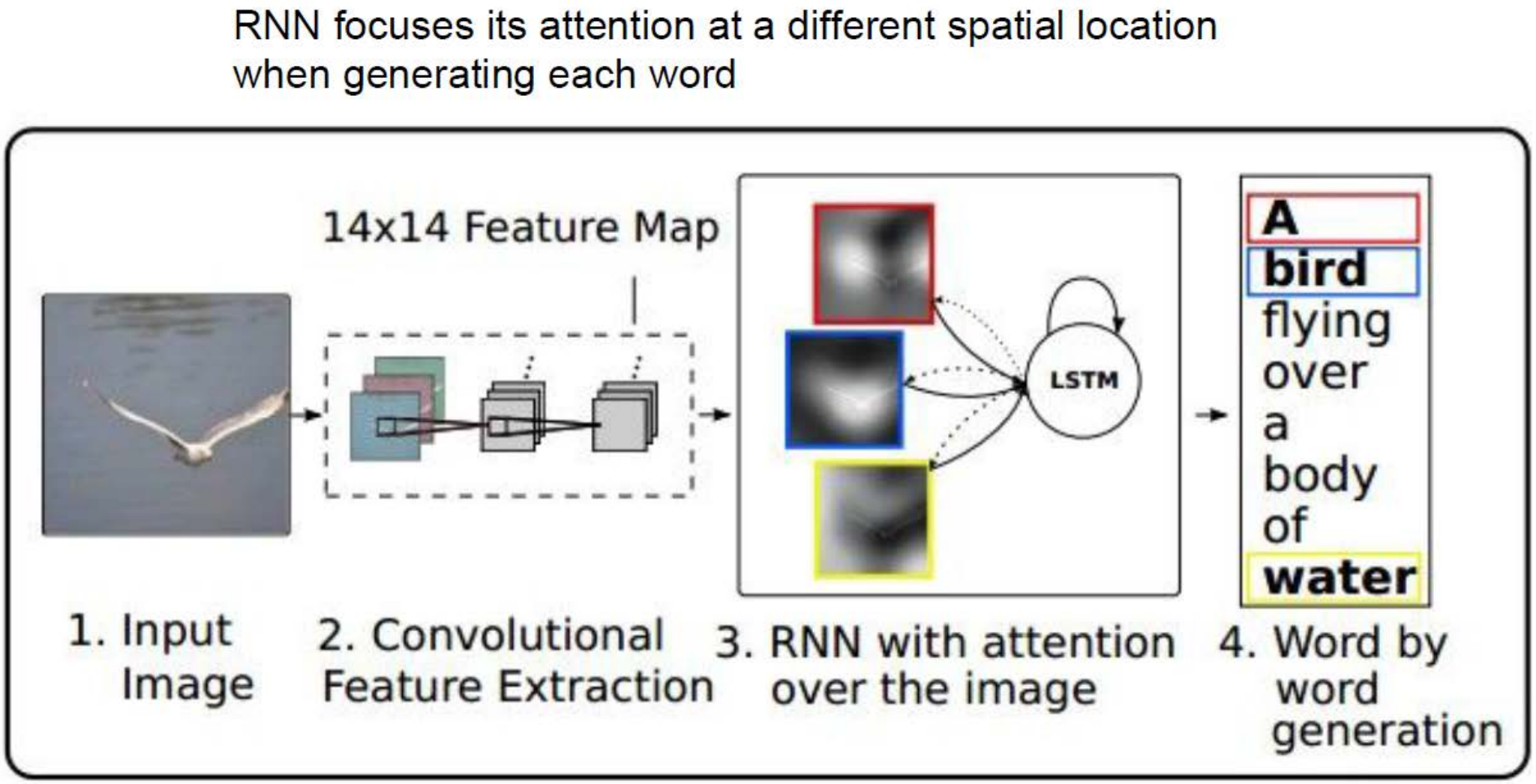
Example: Image Captioning

- CNN for feature extraction
- RNN for generating sentences



Example: Image Captioning with Attention

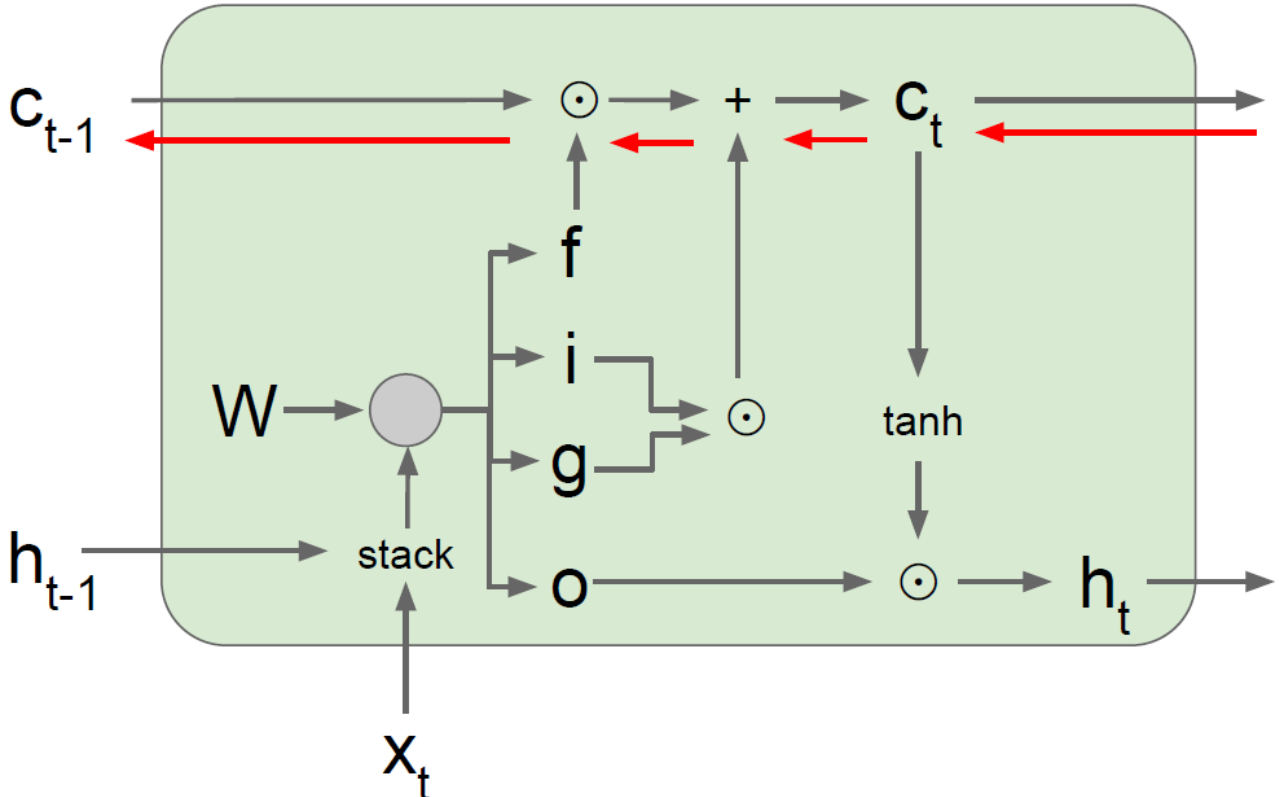
- use “attention” to emphasize the important parts in images



LSTM (Long Short Term Memory)

- solve RNN gradient vanishing (exploding) problem during back propagation

i: input gate: whether to write to cell
f: forget gate: whether to erase cell
o: output gate: how much to reveal cell
g: gate gate: how much to write to cell



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} \mathrm{W} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

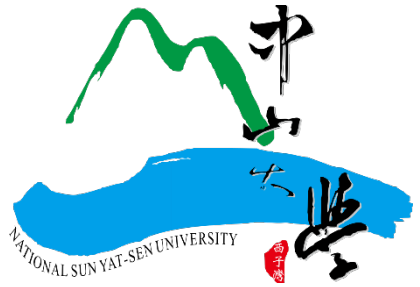
$$h_t = o \odot \tanh(c_t)$$

LSTM

$$h_t = \tanh(\mathrm{W} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix})$$

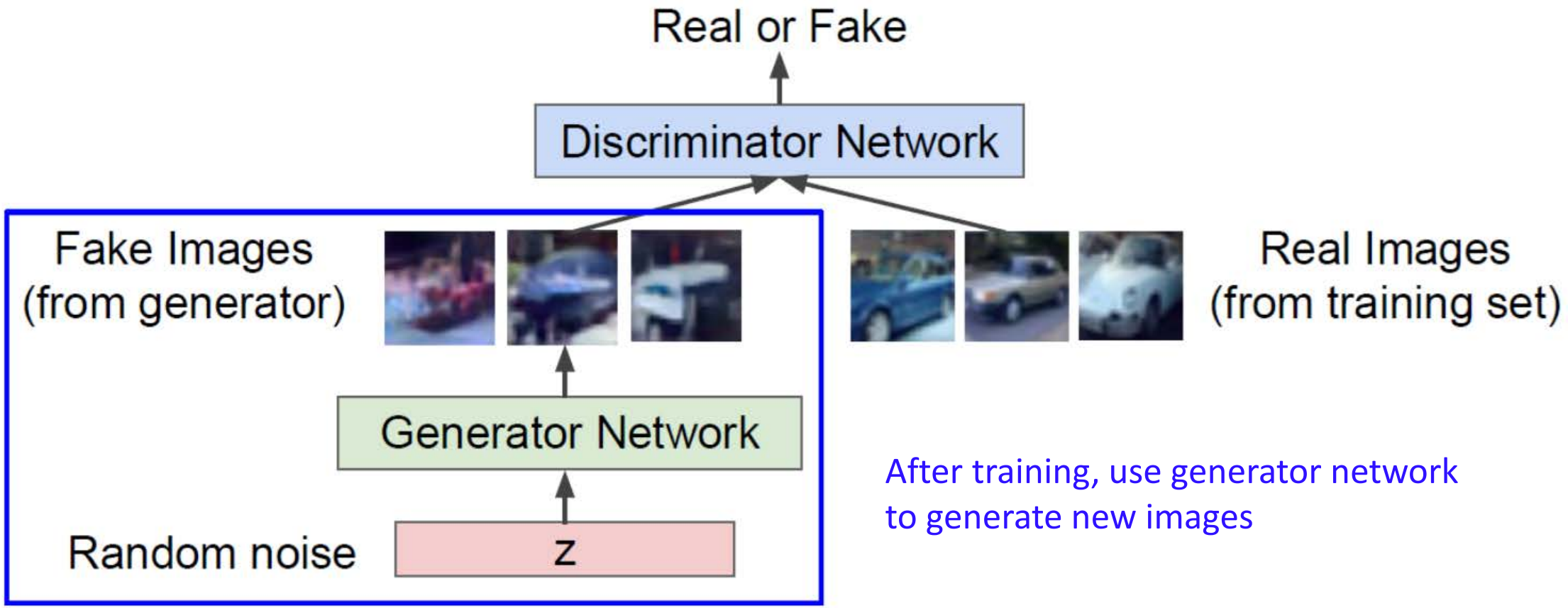
RNN

GAN
(Generative Adversarial Network)



Generative Adversarial Network (GAN)

- **Generator network**: try to fool the discriminator by generating real-looking images
- **Discriminator network**: try to distinguish between real and fake images



After training, use generator network to generate new images