

# **CS 4644-DL / 7643-A: LECTURE 8**

## **DANFEI XU**

Topics:

- Pooling
- Convolutional Neural Networks Architectures

## Recap: Image features are spatially localized!

- Relevant features repeated across the image
  - Edges
  - Color
  - Motifs (corners, etc.)
- No reason to believe one feature tends to appear in a fixed location. Need to search in entire image.



**Can we enforce a structure in the design of a neural network layer to reflect this?**

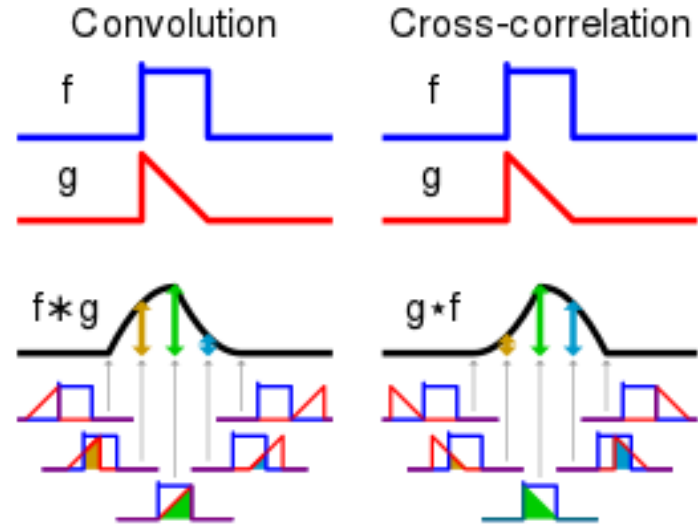
# Recap: Convolution

1-D Convolution is defined as the **integral** of the **product** of two functions after one is reflected about the y-axis and shifted.

Cross-correlation is convolution without the y-axis reflection.

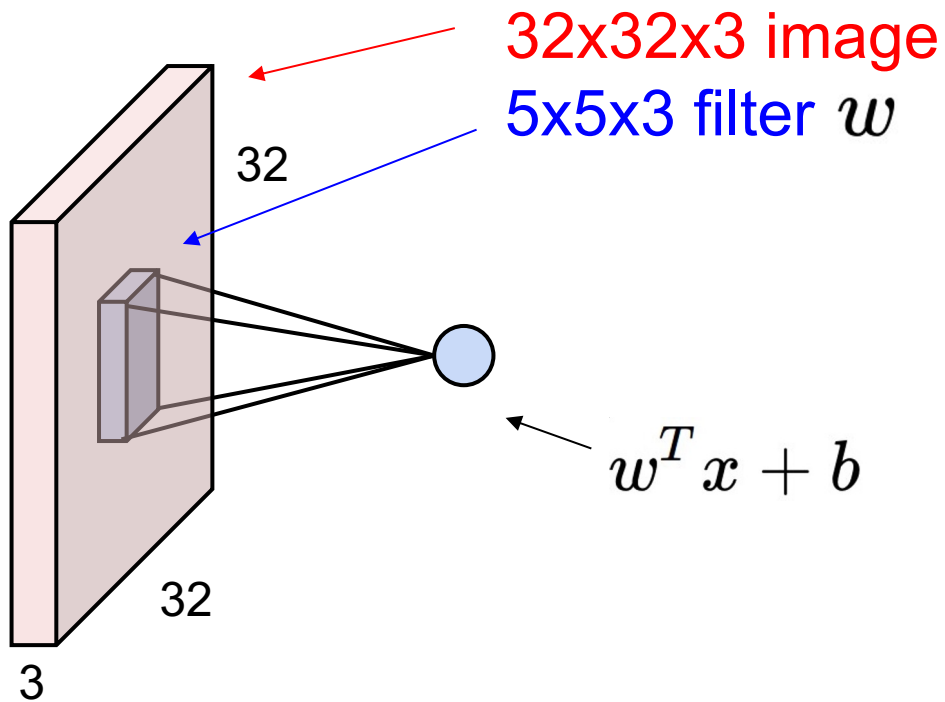
**Intuitively:** given function  $f$  and filter  $g$ . How similar is  $g(-x)$  with the part of  $f(x)$  that it's operating on.

For ConvNets, we don't flip filters, so we are really using Cross-Correlation Nets!

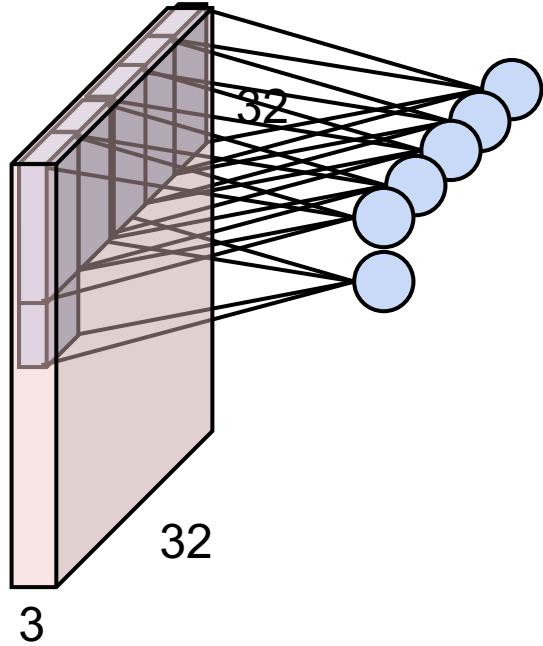


From <https://en.wikipedia.org/wiki/Convolution>

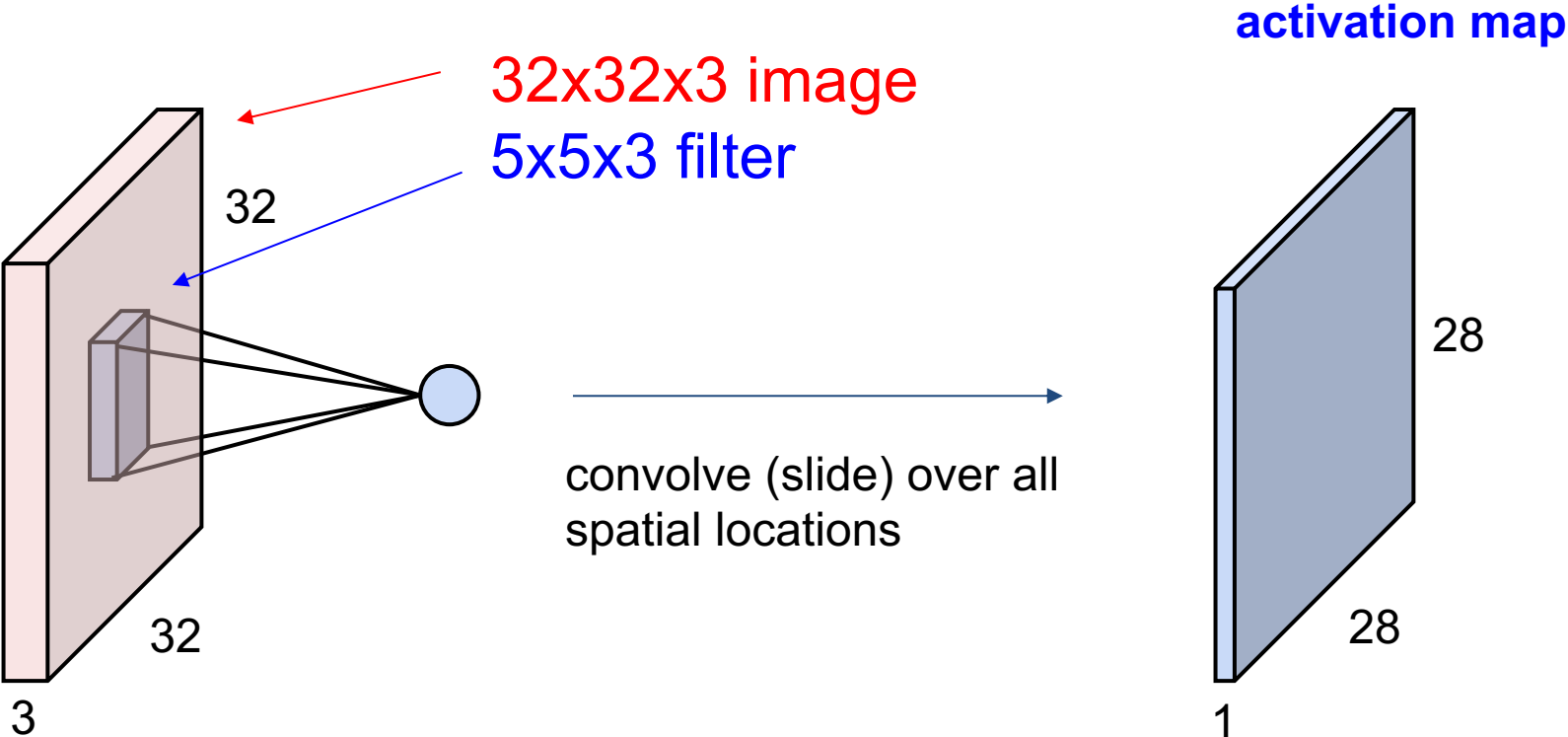
# Convolution Layer



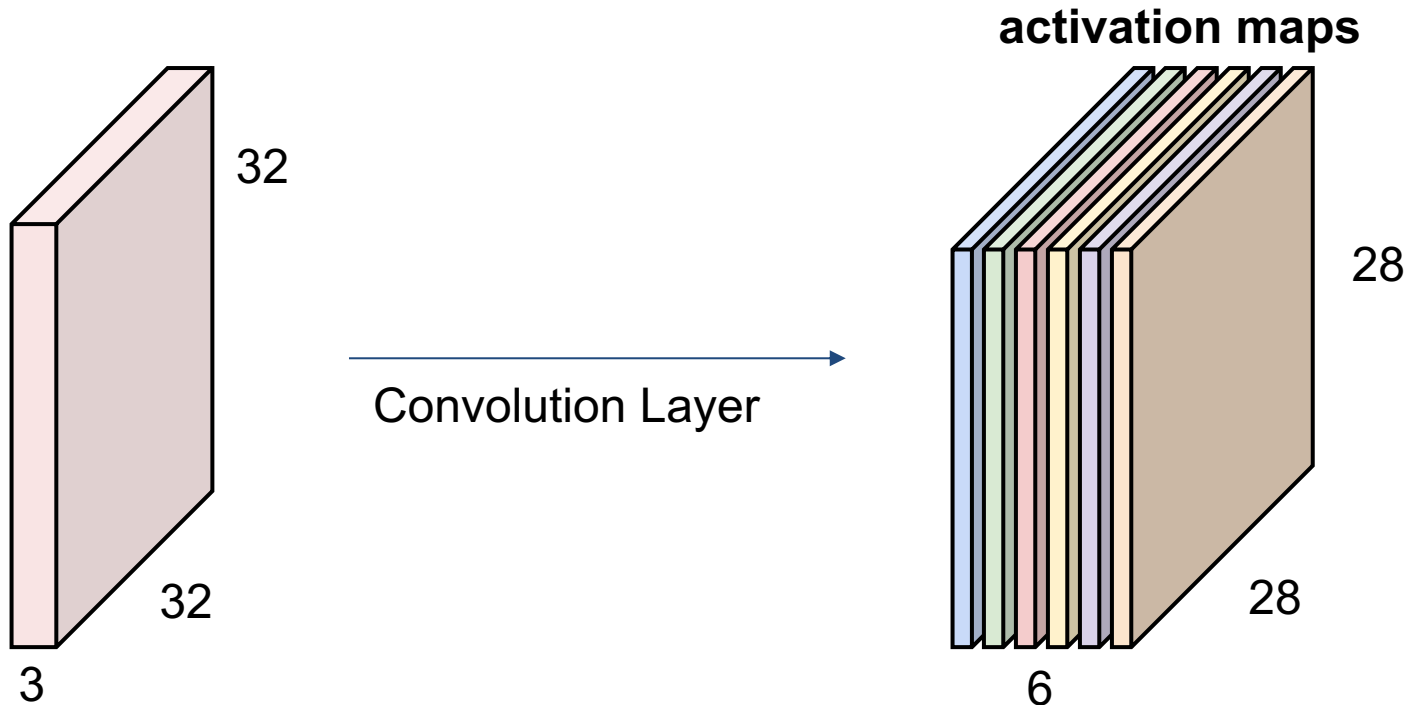
# Recap: Convolution Layer



# Recap: Convolution Layer



For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3** filter, applied with **stride 1**

**pad with 1 pixel** border => what is the output?

**7x7 output!**

N = input dimension

P = padding size

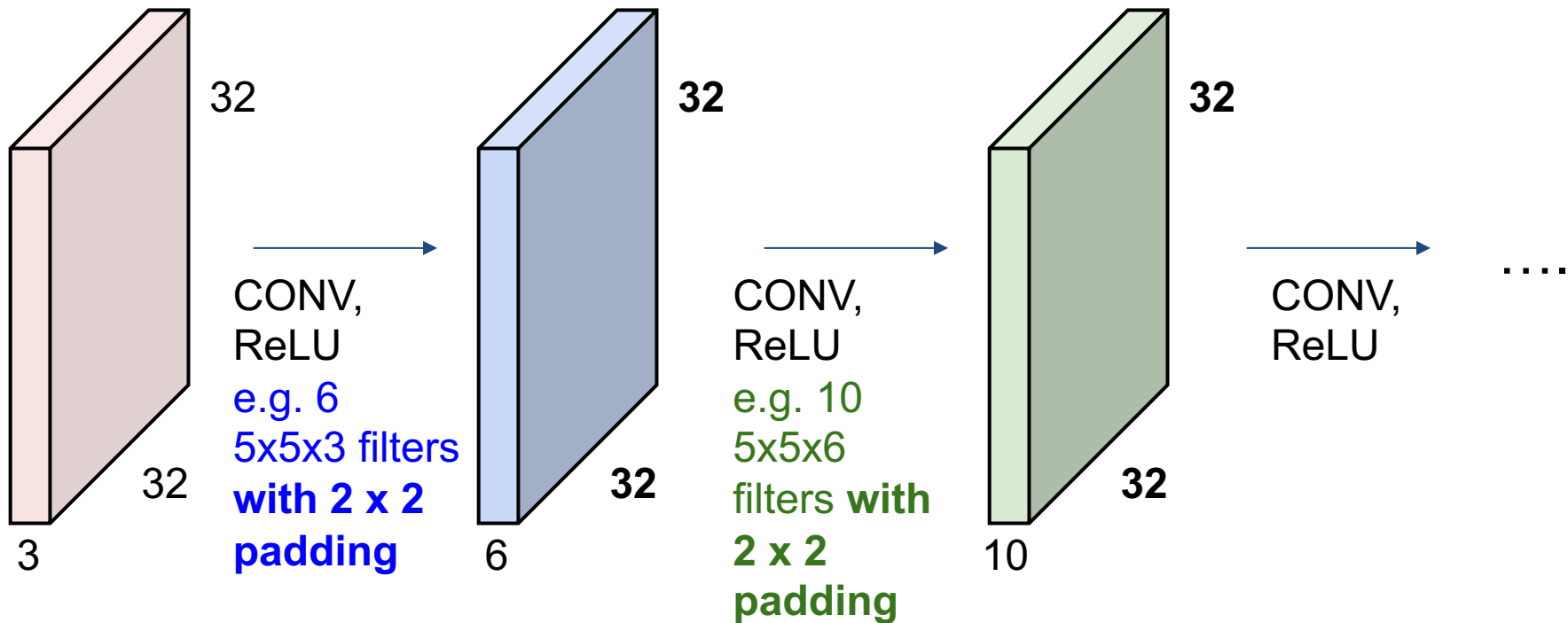
F = filter size

$$\begin{aligned}\text{Output size} &= (N - F + 2P) / \text{stride} + 1 \\ &= (7 - 3 + 2 * 1) / 1 + 1 = 7\end{aligned}$$



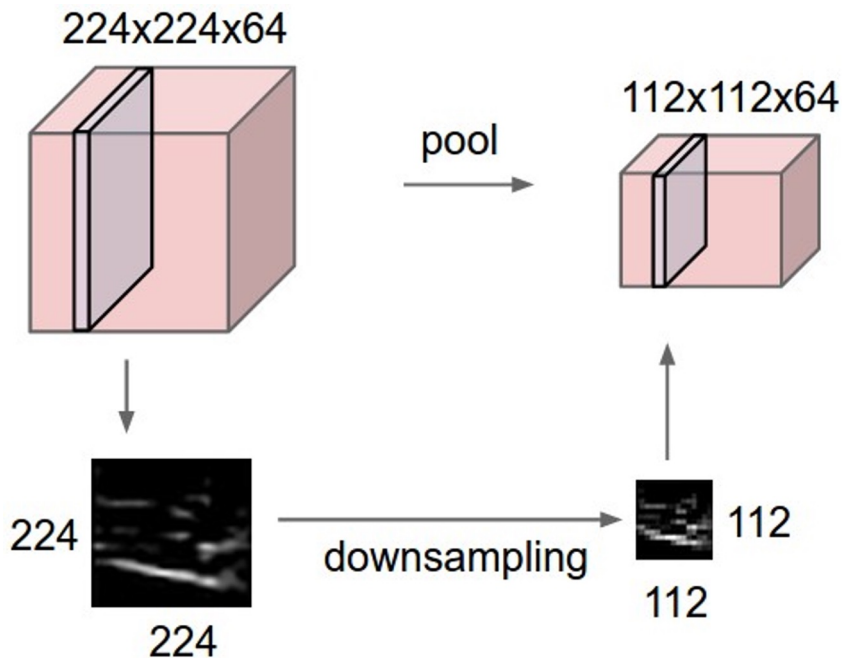
## Remember back to...

With padding, we can keep the same spatial feature dimension throughout the convolution layers.



# Pooling layer (down-sampling)

- makes the representations spatially smaller
- saves computation (GPU mem & speed), allows go deeper
- operates over each activation map independently:



# MAX POOLING

Single depth slice

x

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

y

max pool with 2x2 filters  
and stride 2

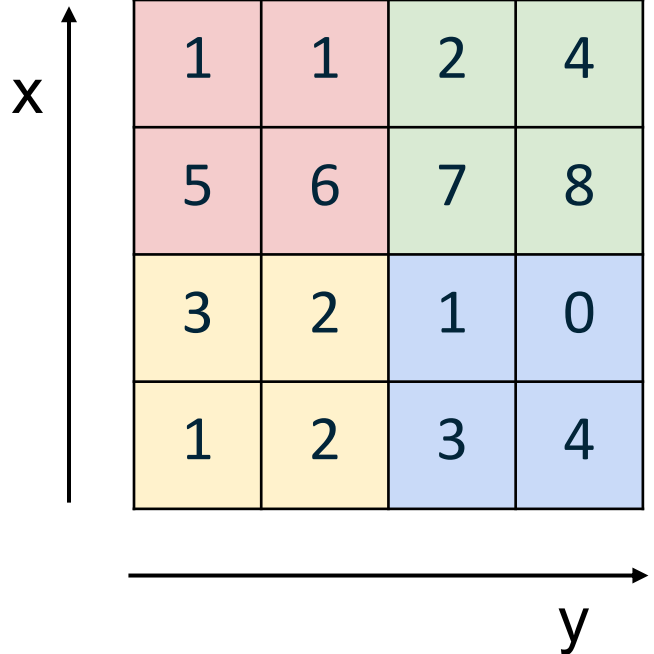


6	8
3	4

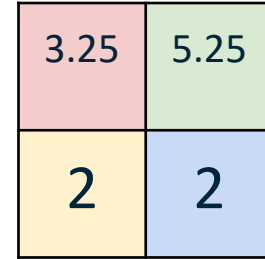
- Intuitively, only forward the most important features in the region.
- Also improve spatial invariance (output is agnostic to where the max value comes from)

# AVG POOLING

Single depth slice



average pool with 2x2 filters and stride 2



# Pooling layer: summary

Let's assume input is  $W_1 \times H_1 \times C$

Pooling layer needs 2 hyperparameters:

- The spatial extent **F** (e.g., 2)
- The stride **S** (e.g., 2)

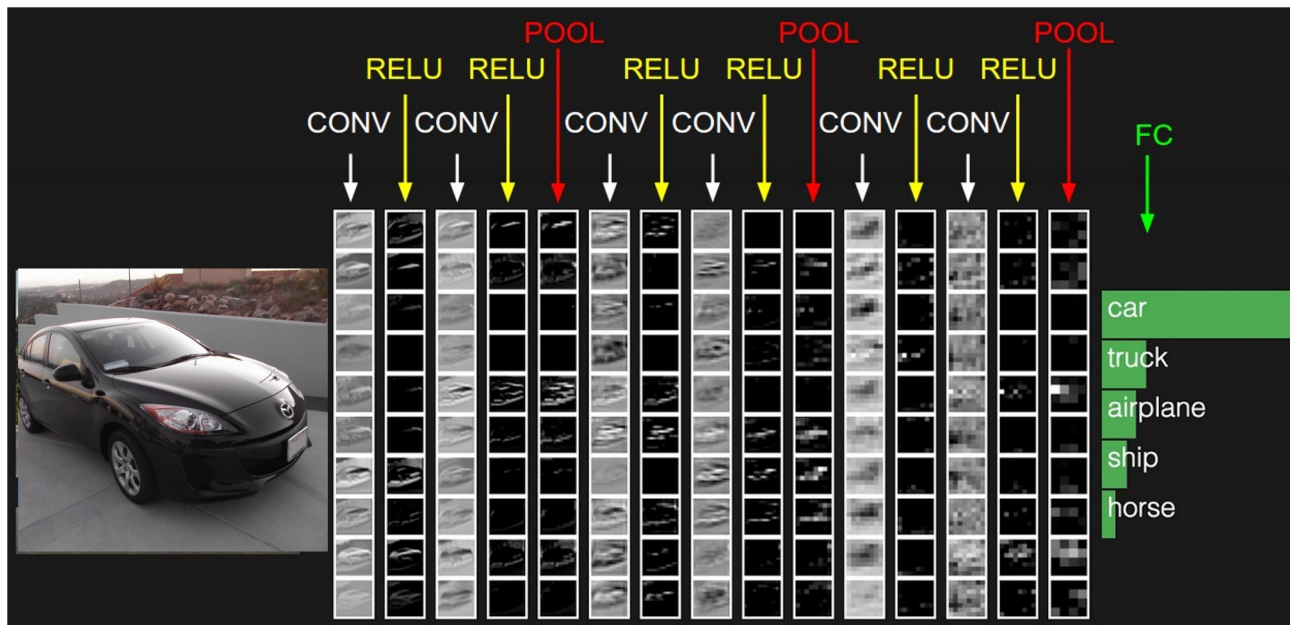
This will produce an output of  $W_2 \times H_2 \times C$  where:

- $W_2 = (W_1 - F) / S + 1$
- $H_2 = (H_1 - F) / S + 1$

Number of parameters?

0

# A canonical (shallow) convolutional neural net

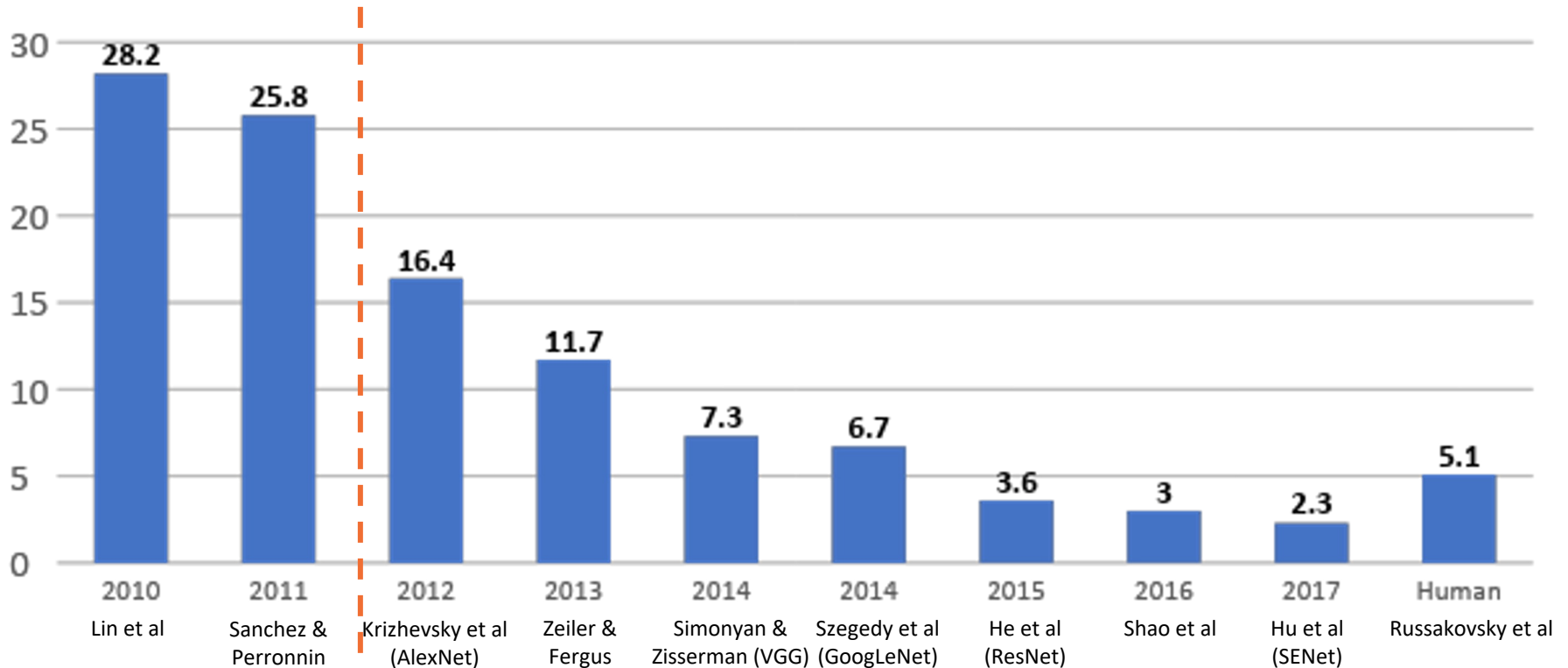


# ConvNets: Where are we today?



The **ImageNet** dataset contains 14,197,122 annotated images according to the WordNet hierarchy. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a benchmark for image classification and object detection based on the dataset.

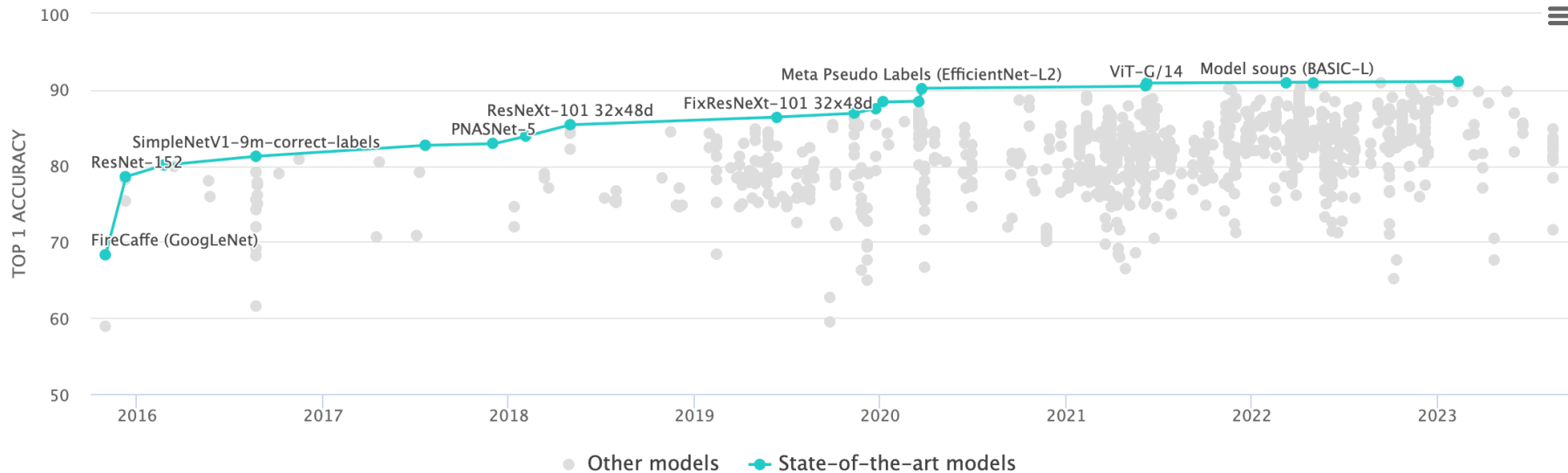
# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



“Pre- Deep Learning”



# ConvNets: Where are we today?



<https://paperswithcode.com/sota/image-classification-on-imagenet>

# CNN Architectures

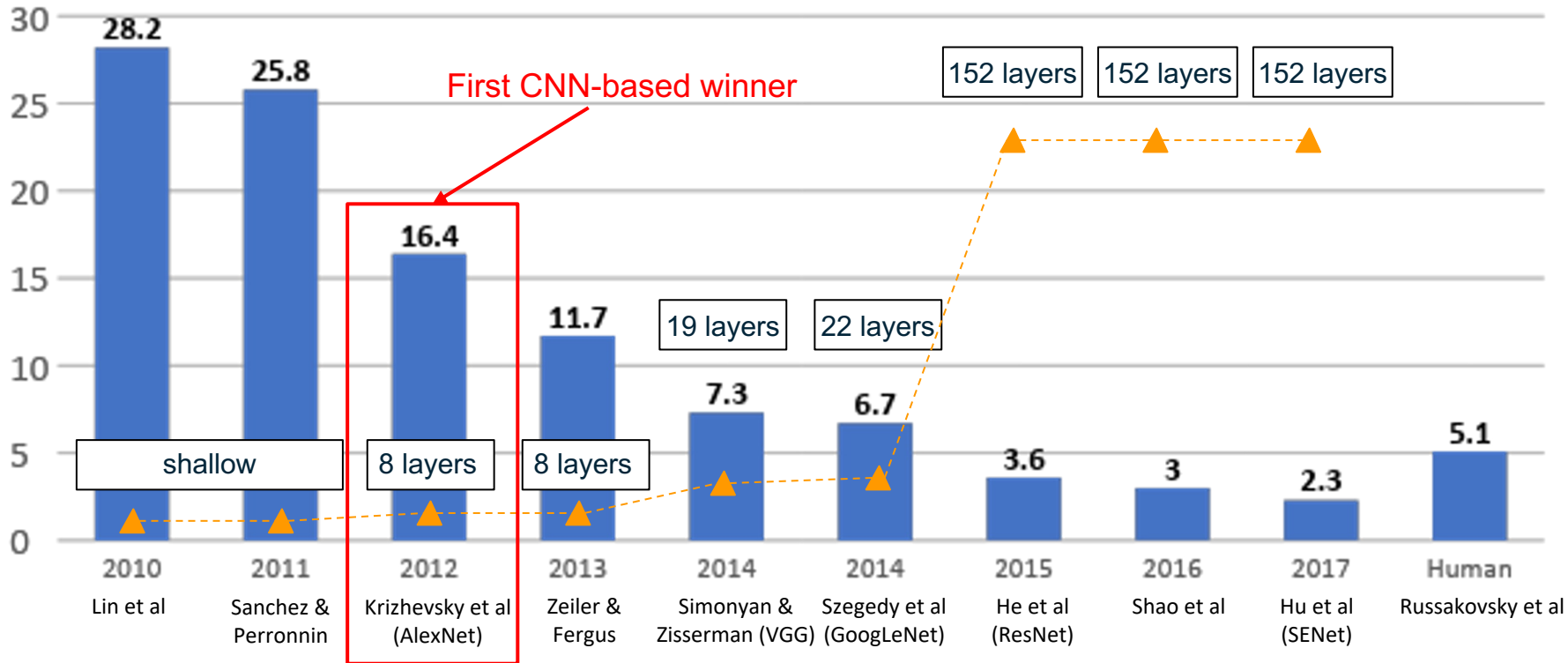
## Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

## Also.....

- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet
- EfficientNet

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

FC8

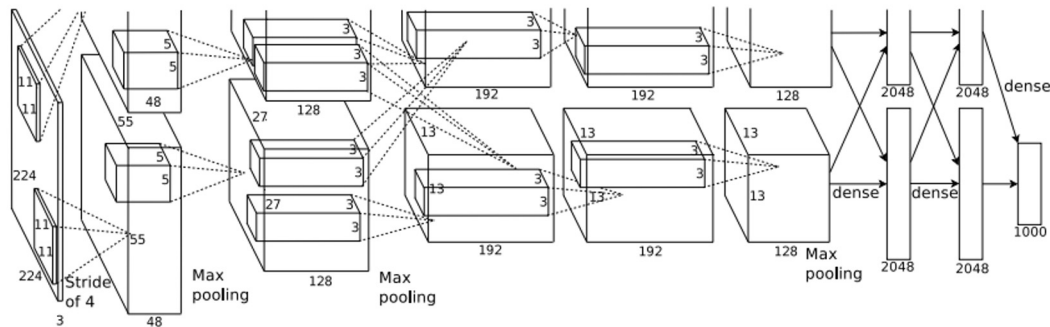
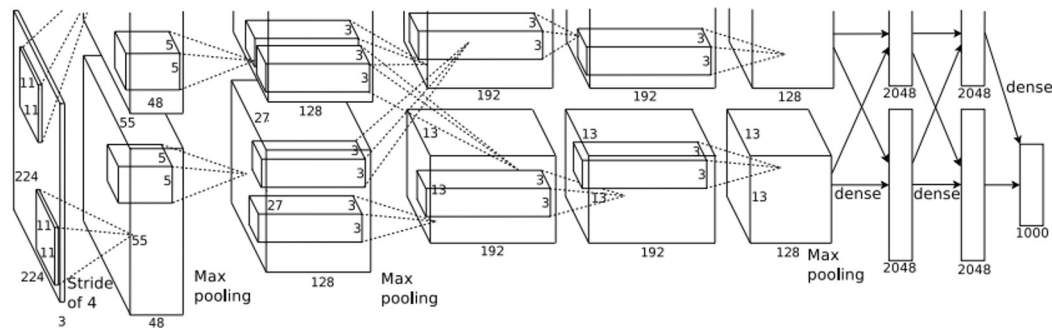


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

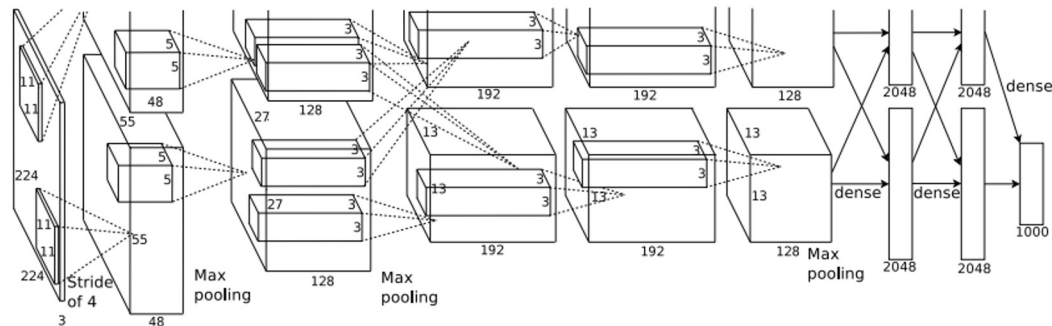
Q: what is the output volume size? Hint:  $(227-11)/4+1 = 55$

$$W' = (W - F + 2P) / S + 1$$

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

$$W' = (W - F + 2P) / S + 1$$

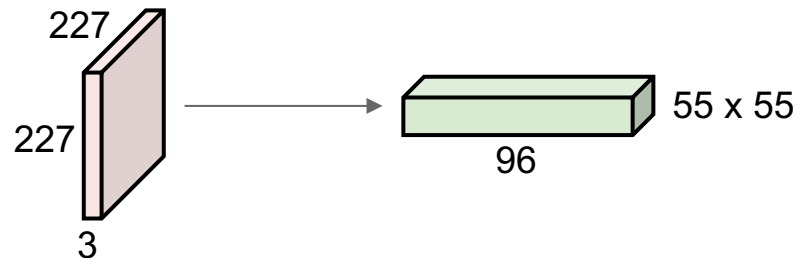
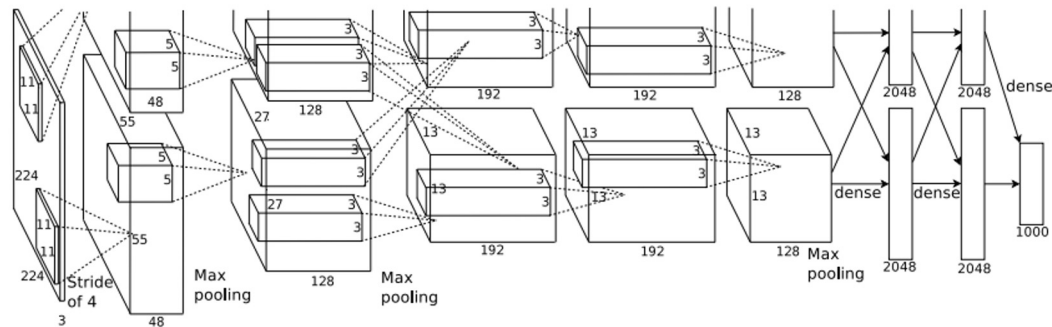


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

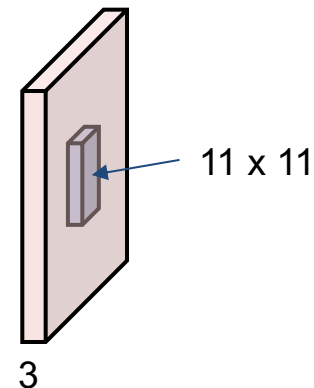
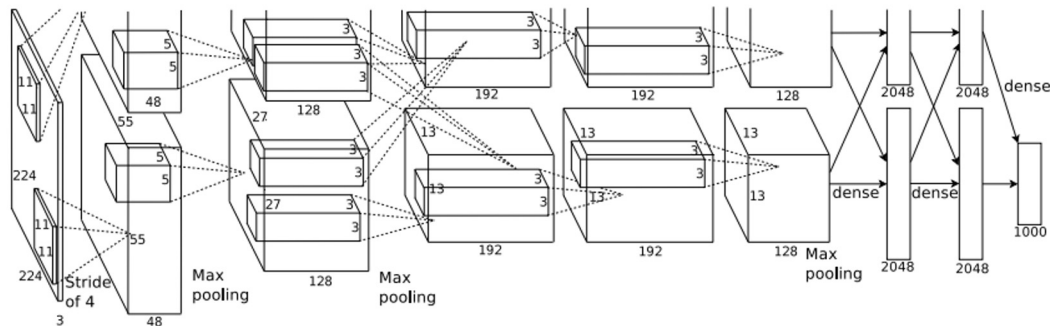


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Parameters:  $(11*11*3 + 1)*96 = \mathbf{35K}$

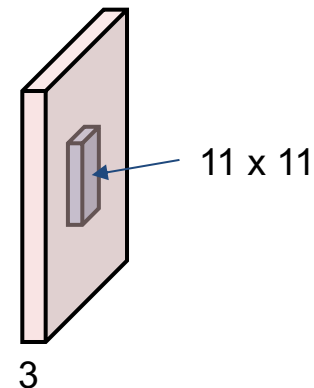
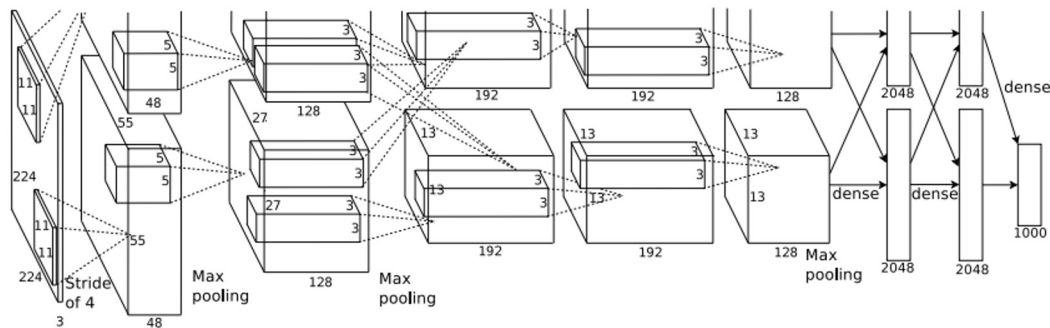


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.



# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

**Second layer (POOL1):** 3x3 filters applied at stride 2

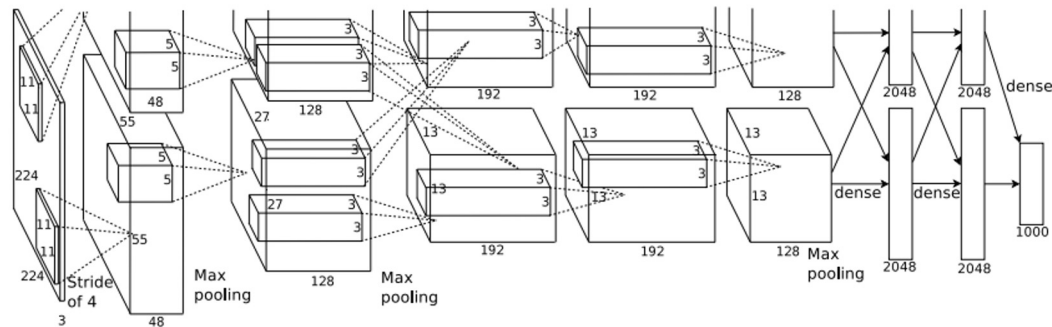
Q: what is the output volume size? Hint:  $(55-3)/2+1 = 27$

$$W' = (W - F + 2P) / S + 1$$

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

**Second layer (POOL1):** 3x3 filters applied at stride 2

Output volume: 27x27x96

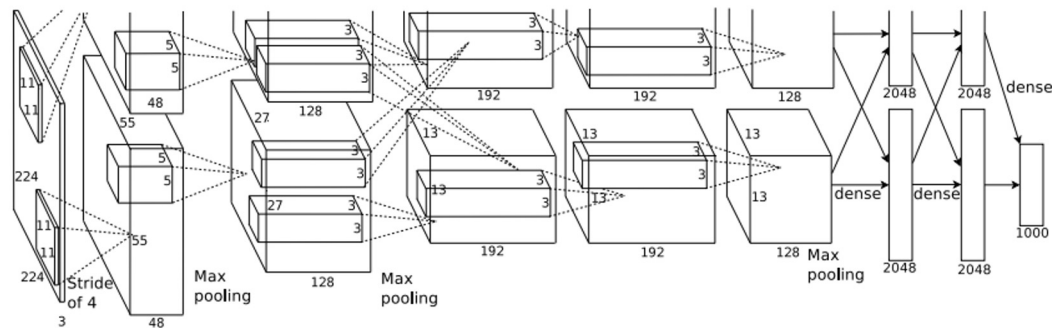
Q: what is the number of parameters in this layer?

$$W' = (W - F + 2P) / S + 1$$

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

**Second layer (POOL1):** 3x3 filters applied at stride 2

Output volume: 27x27x96

Parameters: 0!

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images  
After CONV1: 55x55x96  
After POOL1: 27x27x96  
...

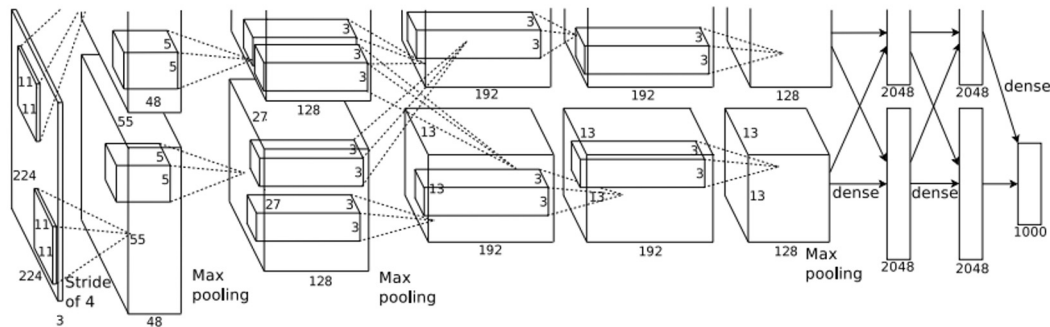


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)

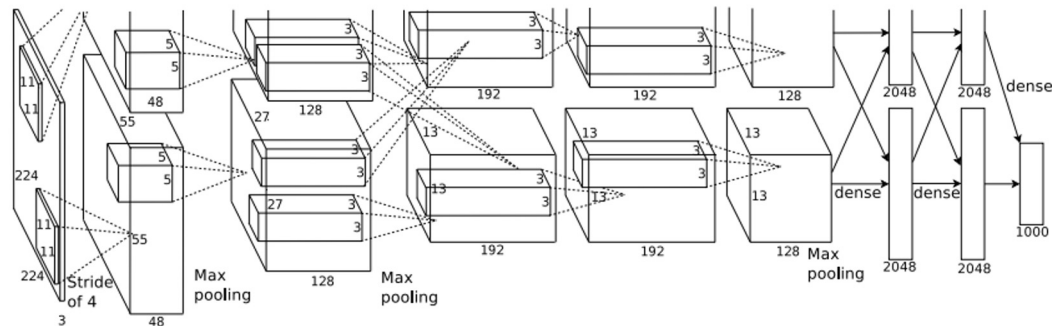


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

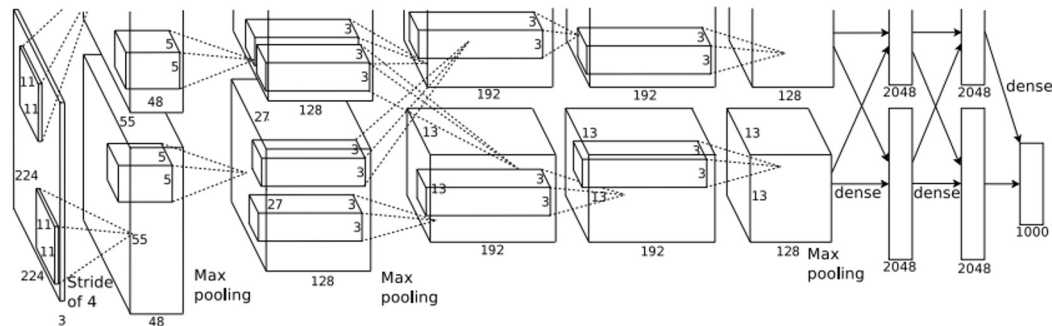
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



## Details/Retrospectives:

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

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

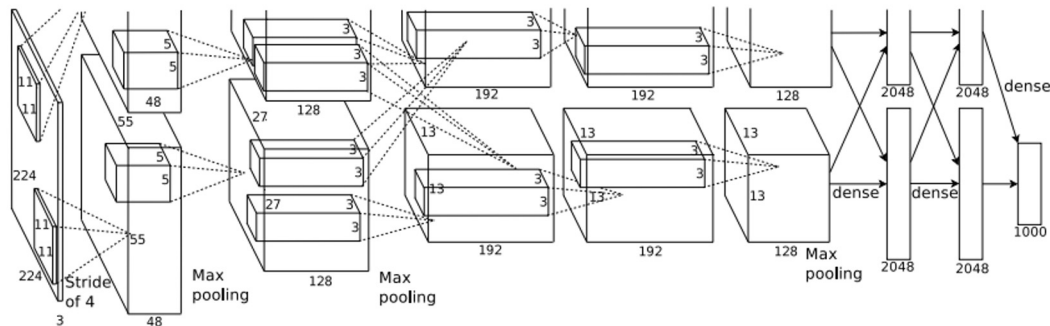
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



## Details/Retrospectives:

- first use of ReLU **Next two lectures!**

- used Norm layers (not common anymore)

- heavy data augmentation

- dropout 0.5

- batch size 128

- SGD Momentum 0.9

- Learning rate 1e-2, reduced by 10

manually when val accuracy plateaus

- L2 weight decay 5e-4

- 7 CNN ensemble: 18.2% -> 15.4%

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

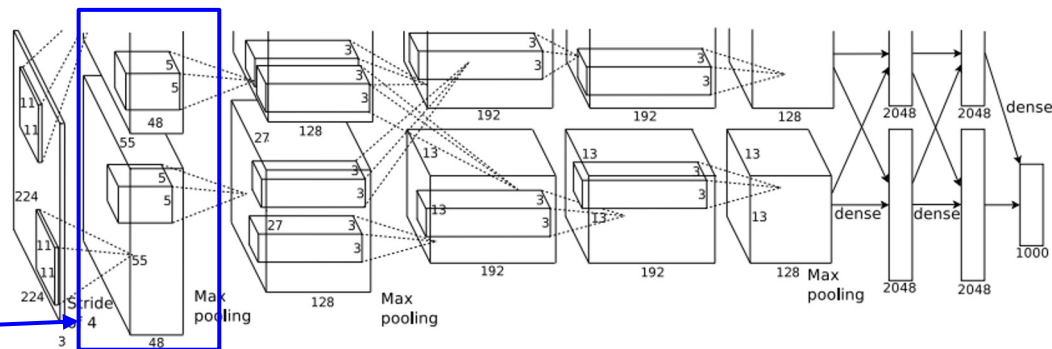
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



[55x55x48] x 2

Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.



# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

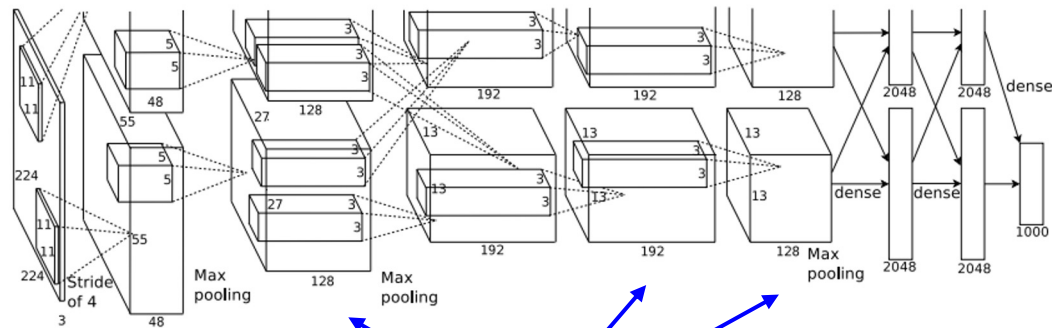
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



**CONV1, CONV2, CONV4, CONV5:**  
Connections only with feature maps  
on same GPU

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

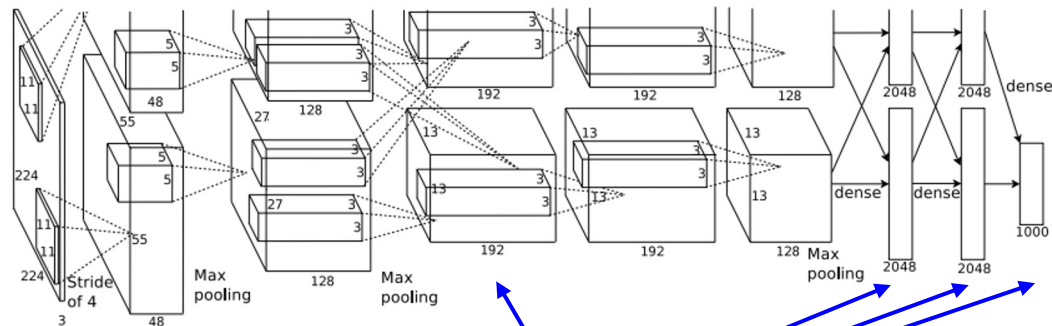
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



**CONV3, FC6, FC7, FC8:**

Connections with all feature maps in preceding layer, communication across GPUs

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

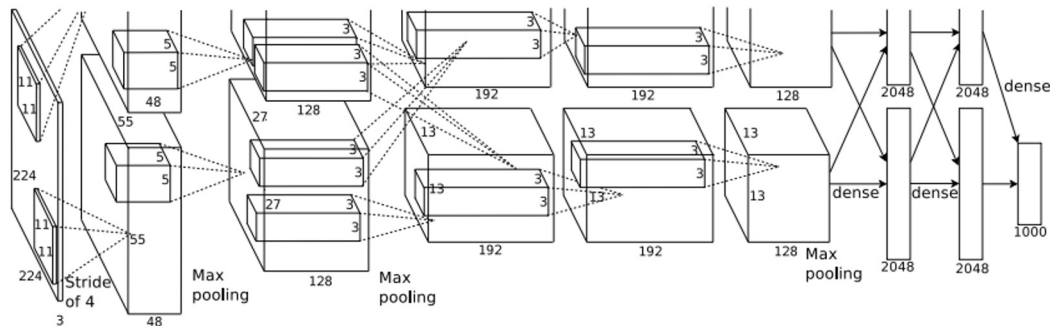
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



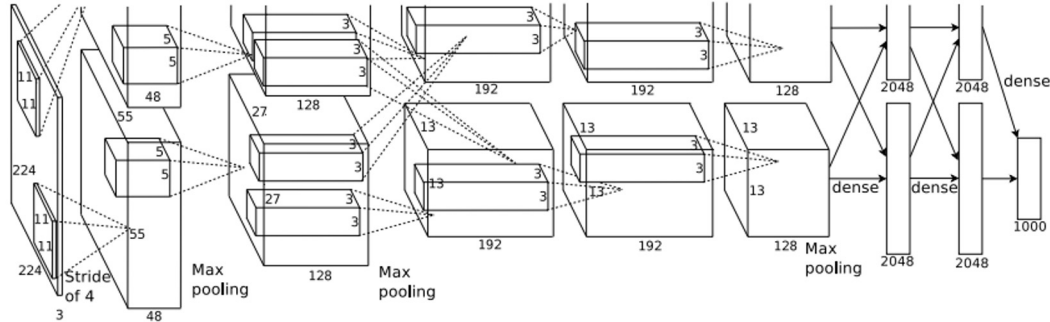
How to choose these hyperparameters?

1. Trial and error ☹️
2. **Computational cost (memory and tflops)**

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

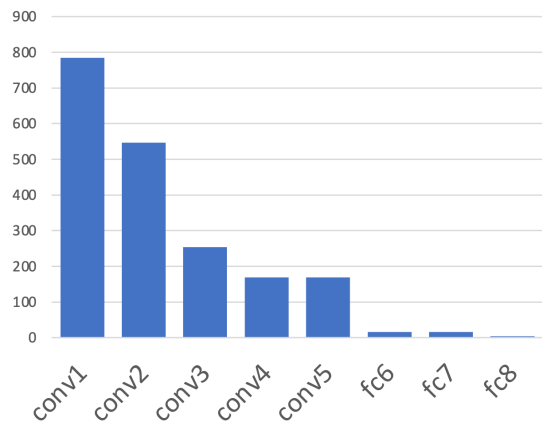
# Case Study: AlexNet

[Krizhevsky et al. 2012]



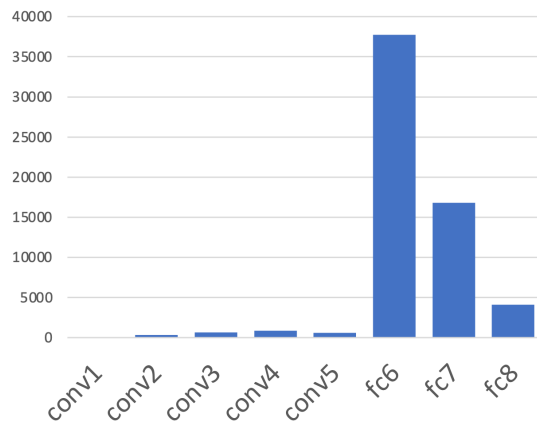
High memory (feature volume) in earlier convs

Memory (KB)



More parameters in FC than in conv

Params (K)



Most FLO occurs in conv layers

MFLOP

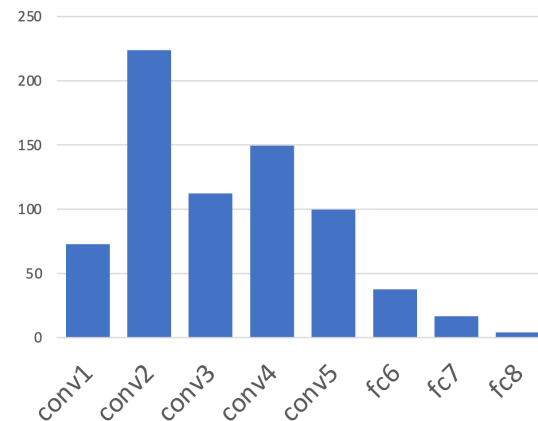
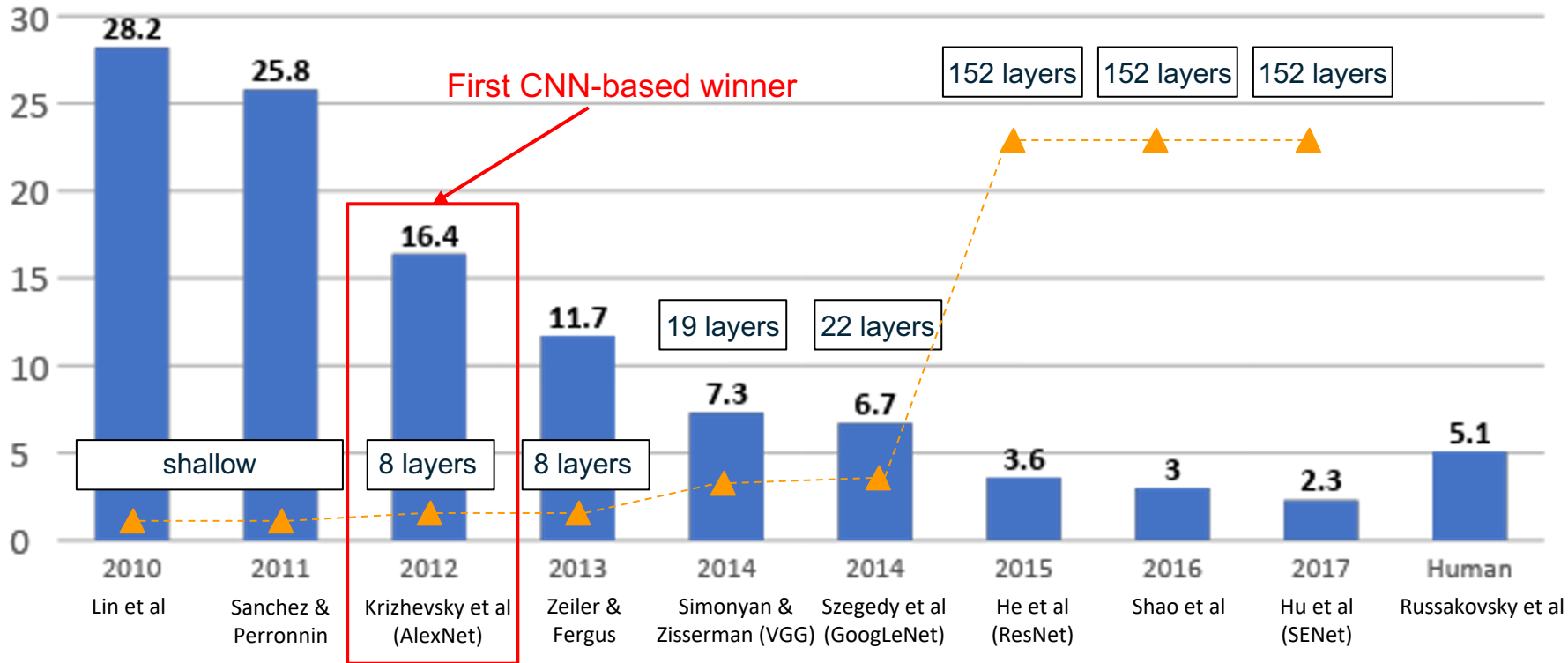
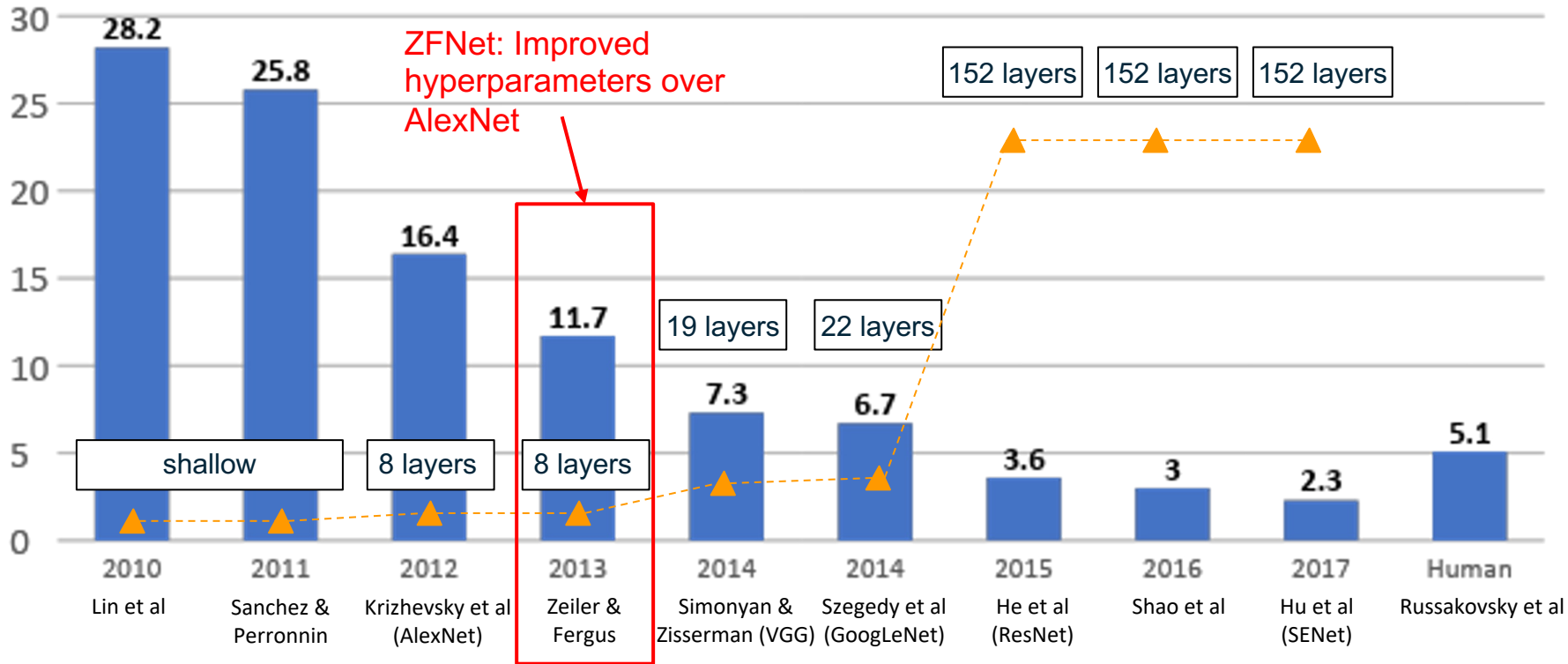


Figure credit: Umich EECS 498.008 / 598.008

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

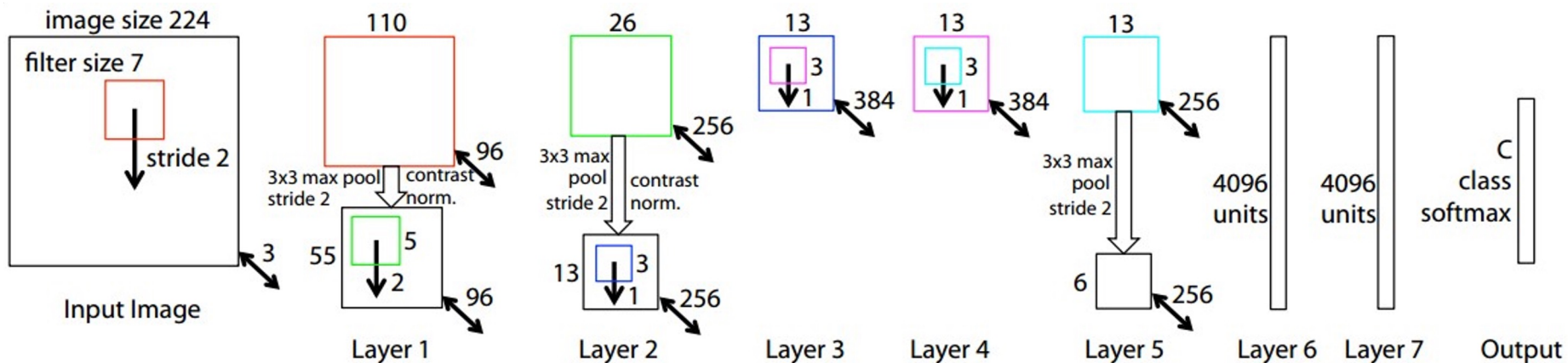


# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# ZFNet

[Zeiler and Fergus, 2013]



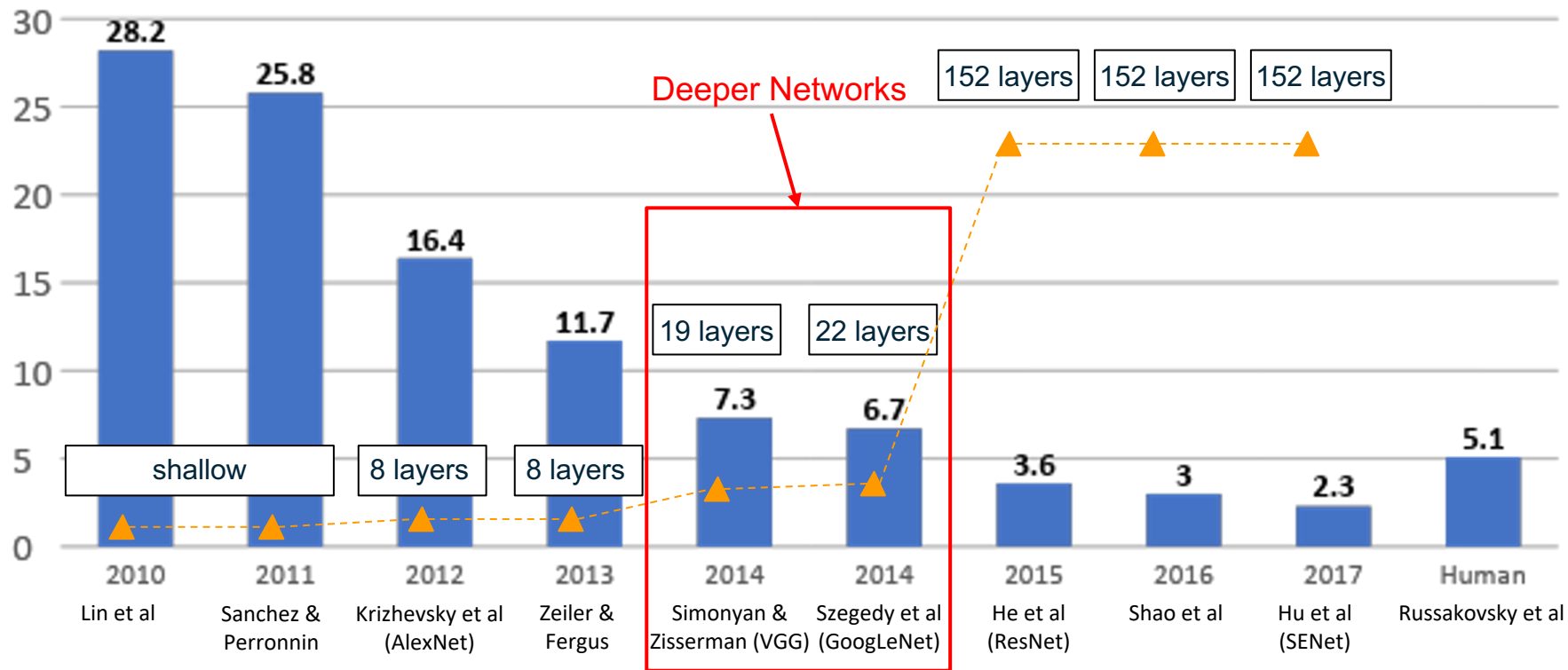
AlexNet but:

CONV1: change from (11x11 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%

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners





# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

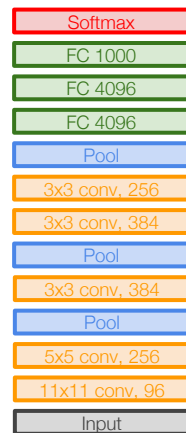
8 layers (AlexNet)

-> 16 - 19 layers (VGG16Net)

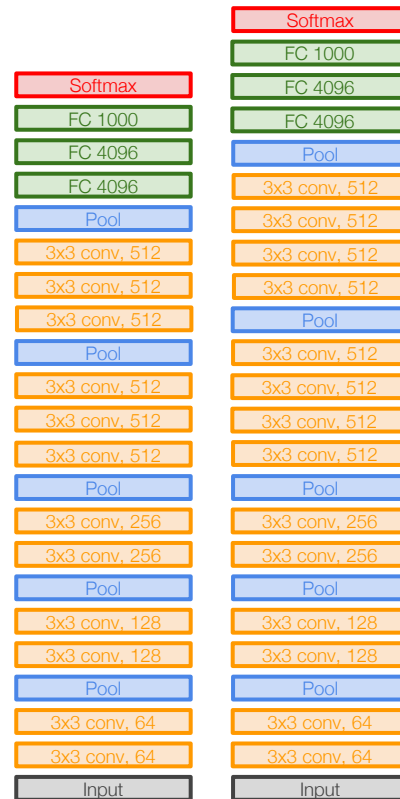
Only 3x3 CONV stride 1, pad 1  
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13  
(ZFNet)

-> 7.3% top 5 error in ILSVRC'14



AlexNet



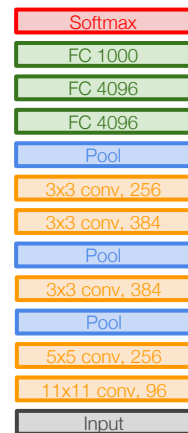
VGG16

VGG19

# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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



AlexNet



VGG16

VGG19

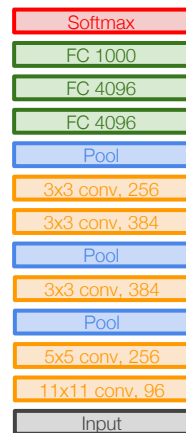
# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

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

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



AlexNet



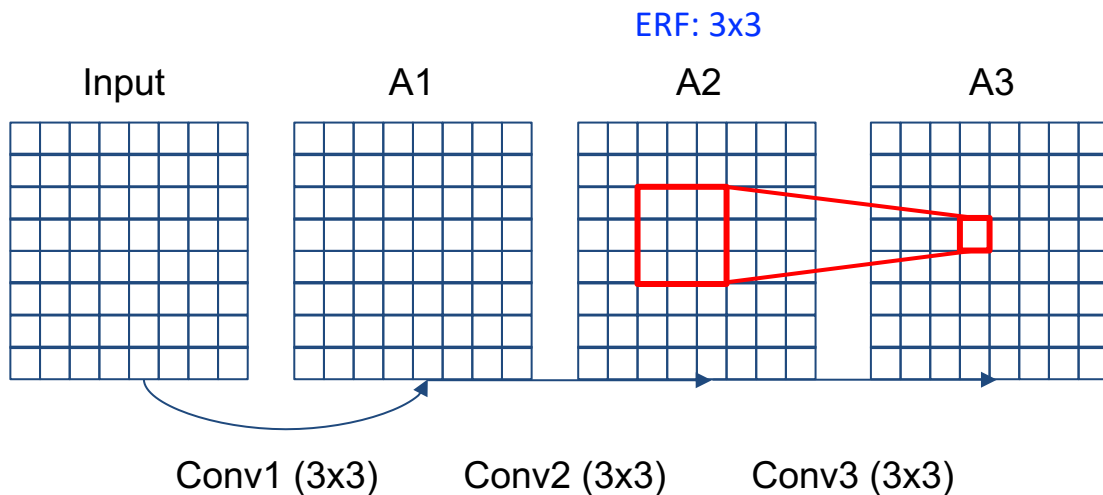
VGG16

VGG19

# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

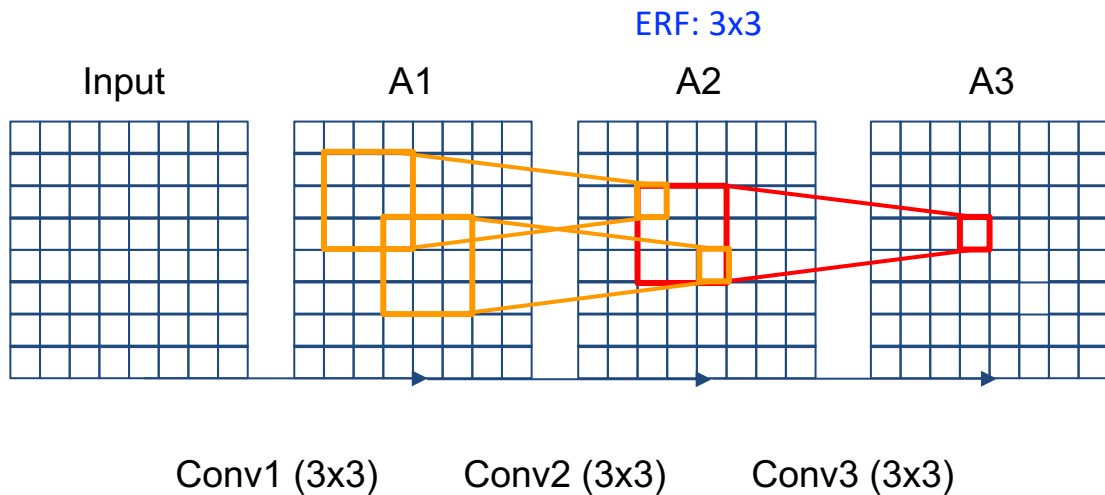
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

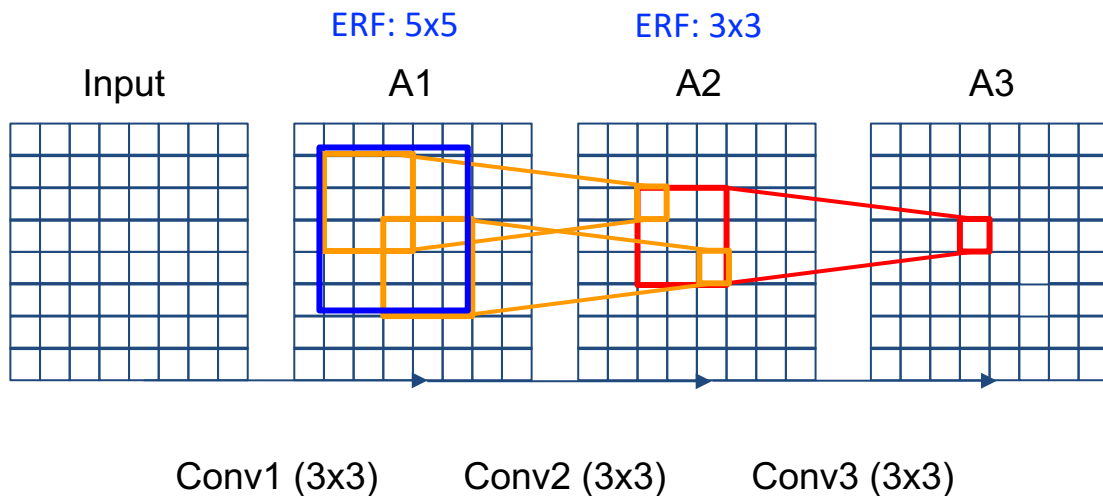
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

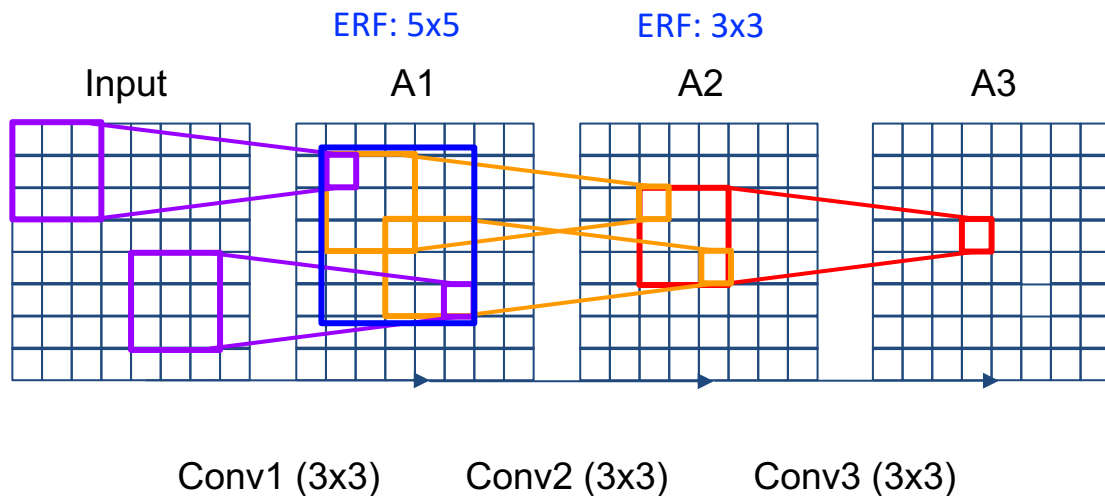
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

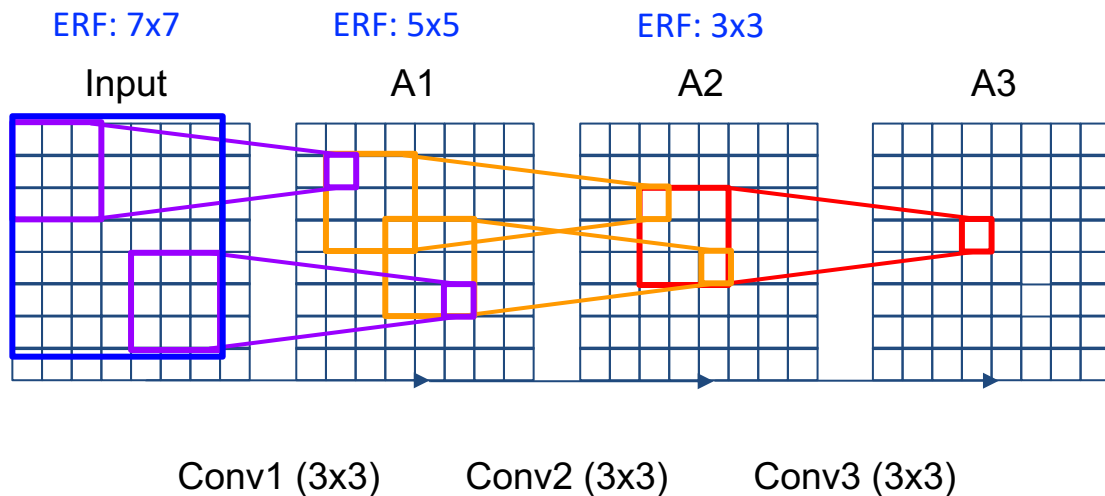
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?





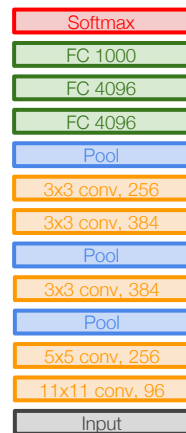
# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

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

[7x7]



AlexNet



VGG16

VGG19

# Case Study: VGGNet

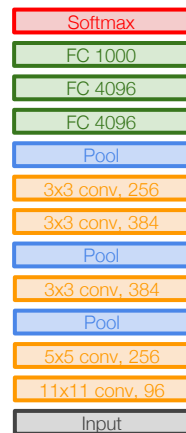
[Simonyan and Zisserman, 2014]

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

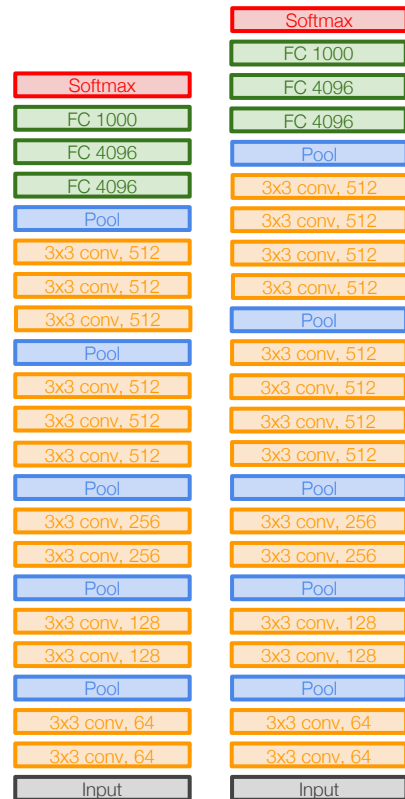
Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

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



AlexNet



VGG16

VGG19

# Case Study: VGGNet

[Simonyan and Zisserman, 2014]

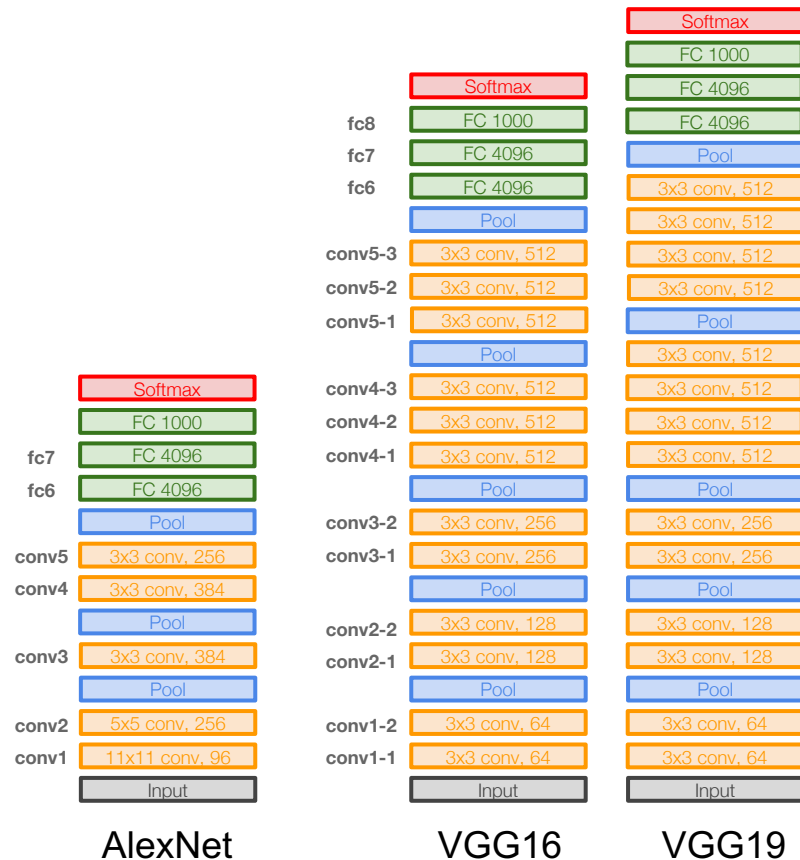
## Details:

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

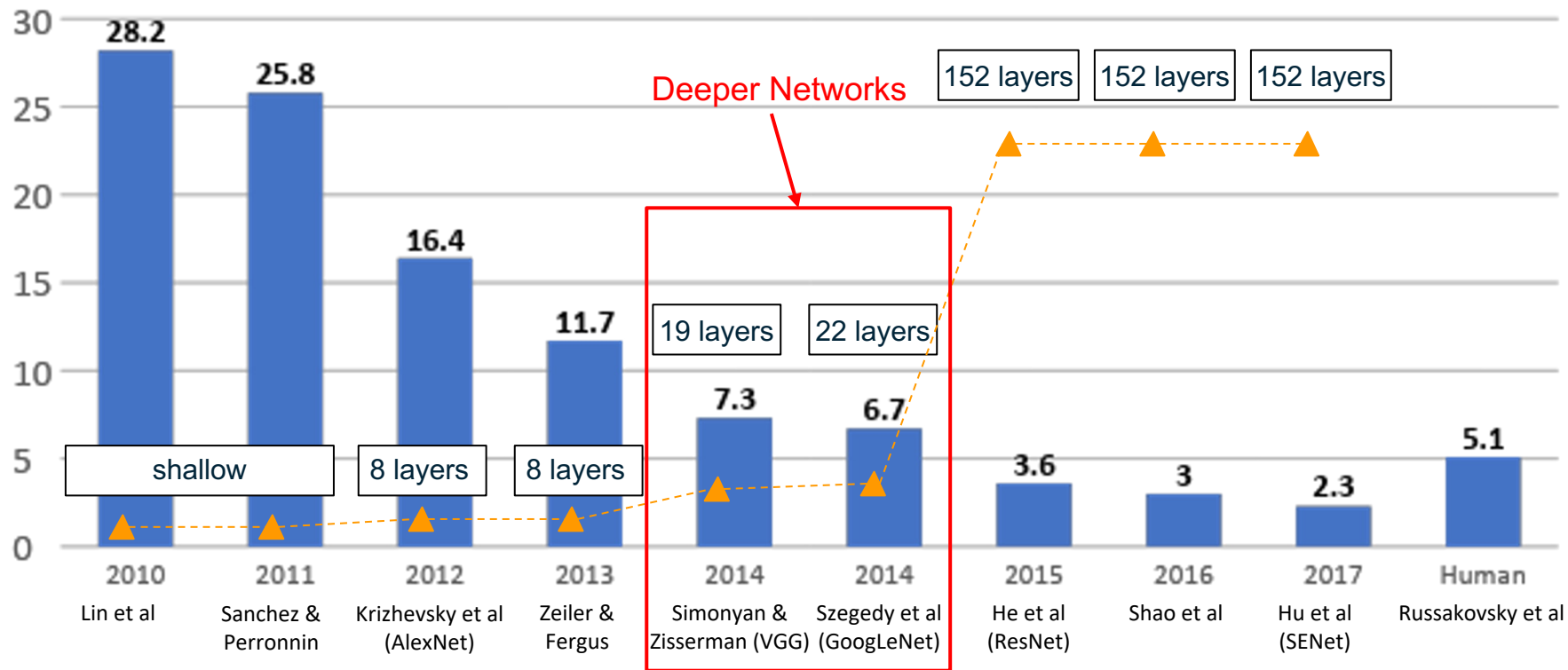
**Still very expensive!**

TOTAL memory:  $24M * 4 \text{ bytes} \approx 96MB / \text{image}$   
(only forward!  $\sim *2$  for bwd)

TOTAL params: 138M parameters



# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

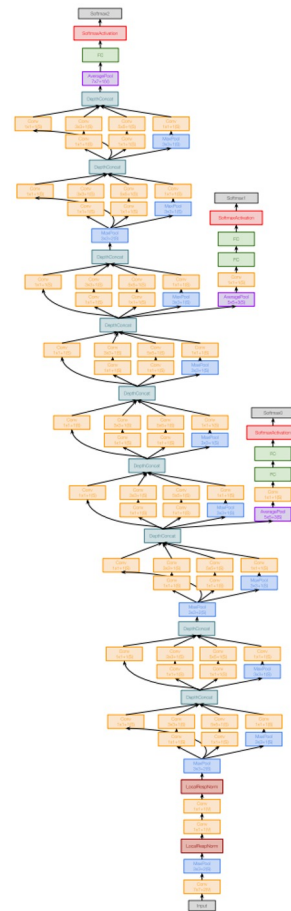


# Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, focus on computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!  
12x less than AlexNet  
27x less than VGG-16
- Efficient “Inception” module
- No FC layers



# Case Study: GoogLeNet

[Szegedy et al., 2014]

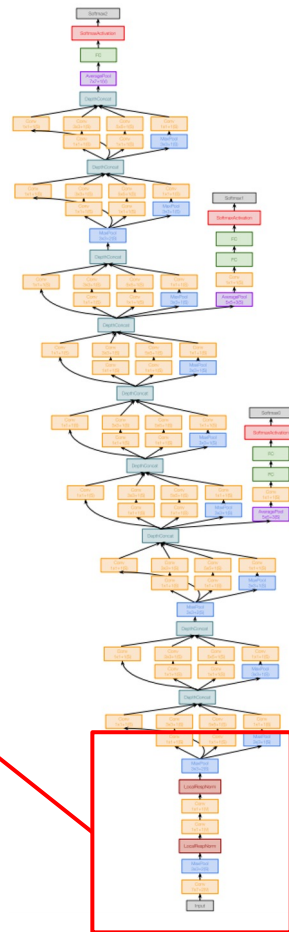
Deeper networks, focus on computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!  
12x less than AlexNet  
27x less than VGG-16
- Efficient "Inception" module
- No FC layers

**Stem Network:** aggressively reduce the input feature volume

- Conv 7 x 7 x 64 with stride 2
- MaxPool
- Conv 1 x 1 x 64
- Conv 3 x 3 x 192
- MaxPool

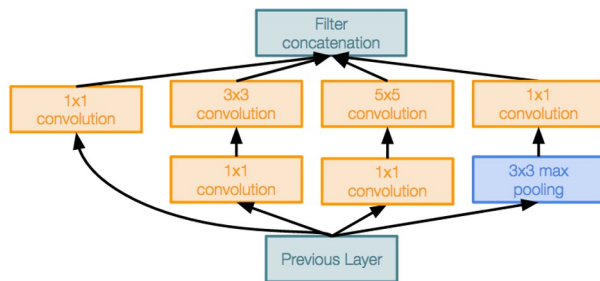
Reduce 224 x 224 spatial solution to 28 x 28 with just 418 MFLOP!  
(Comparing to 7485 MFLOP of VGG)



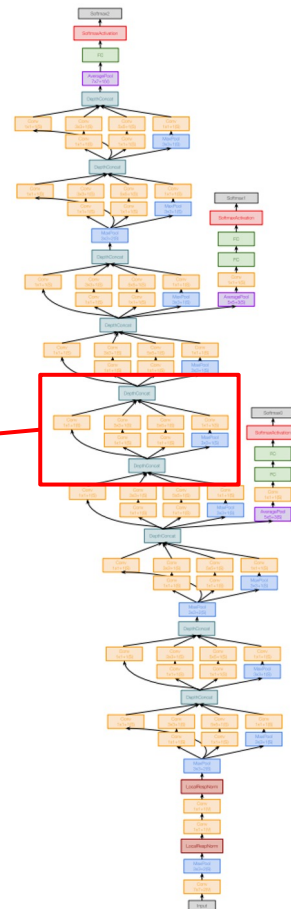
# Case Study: GoogLeNet

[Szegedy et al., 2014]

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



Inception module

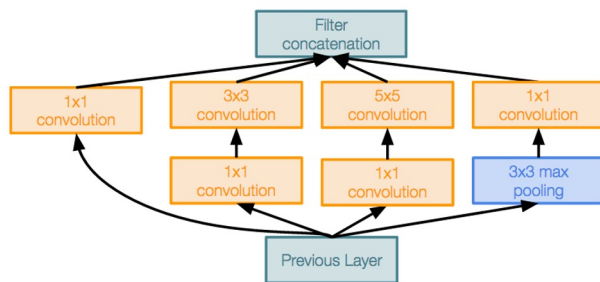


# Case Study: GoogLeNet

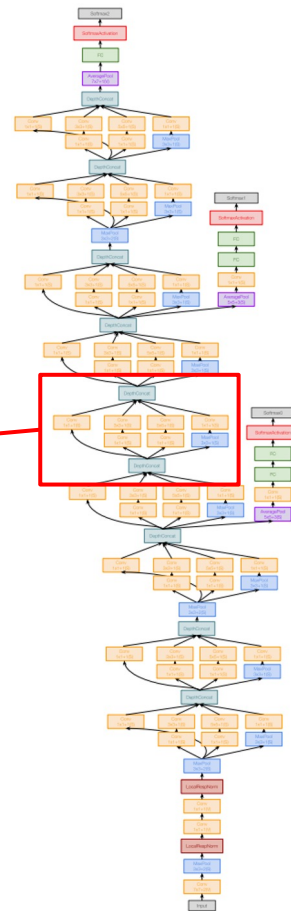
[Szegedy et al., 2014]

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

Multiple conv filter size diversifies learned features



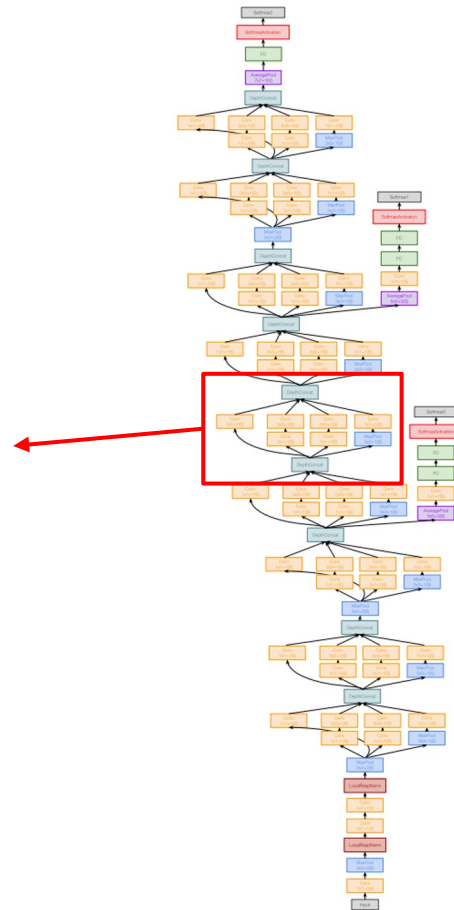
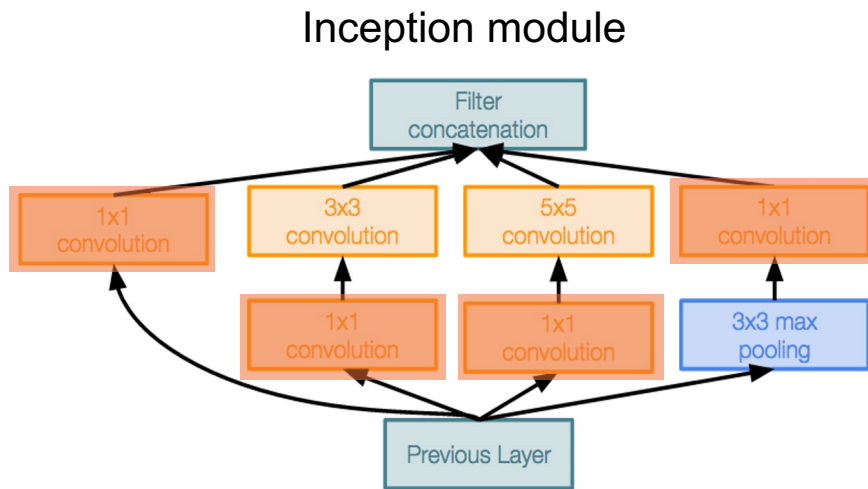
Inception module



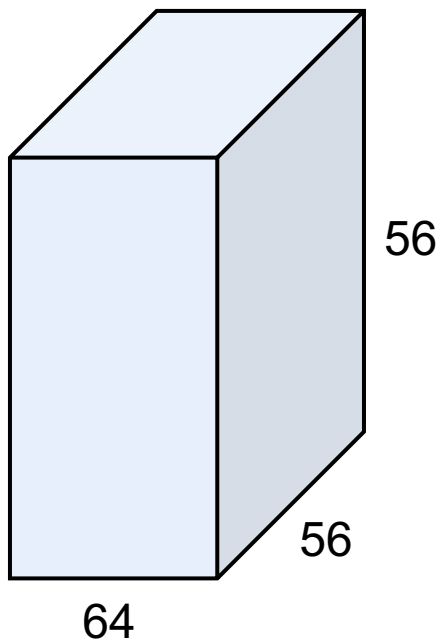


# Case Study: GoogLeNet

[Szegedy et al., 2014]



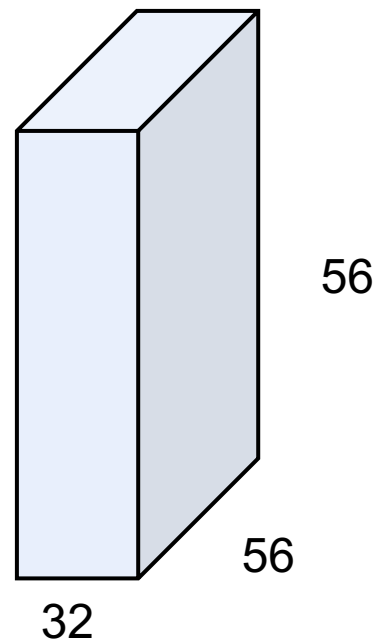
# Review: 1x1 convolutions



1x1 CONV  
with 32 filters

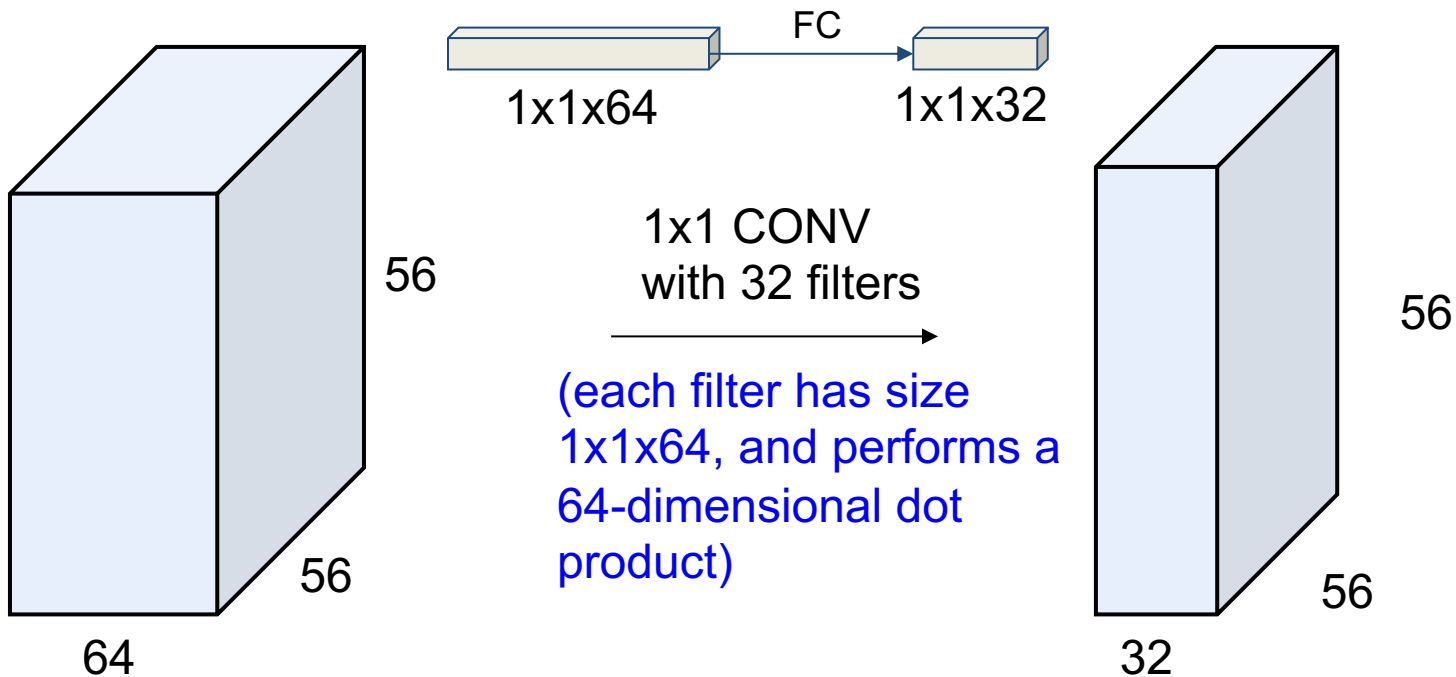


(each filter has size  
1x1x64, and performs a  
64-dimensional dot  
product)



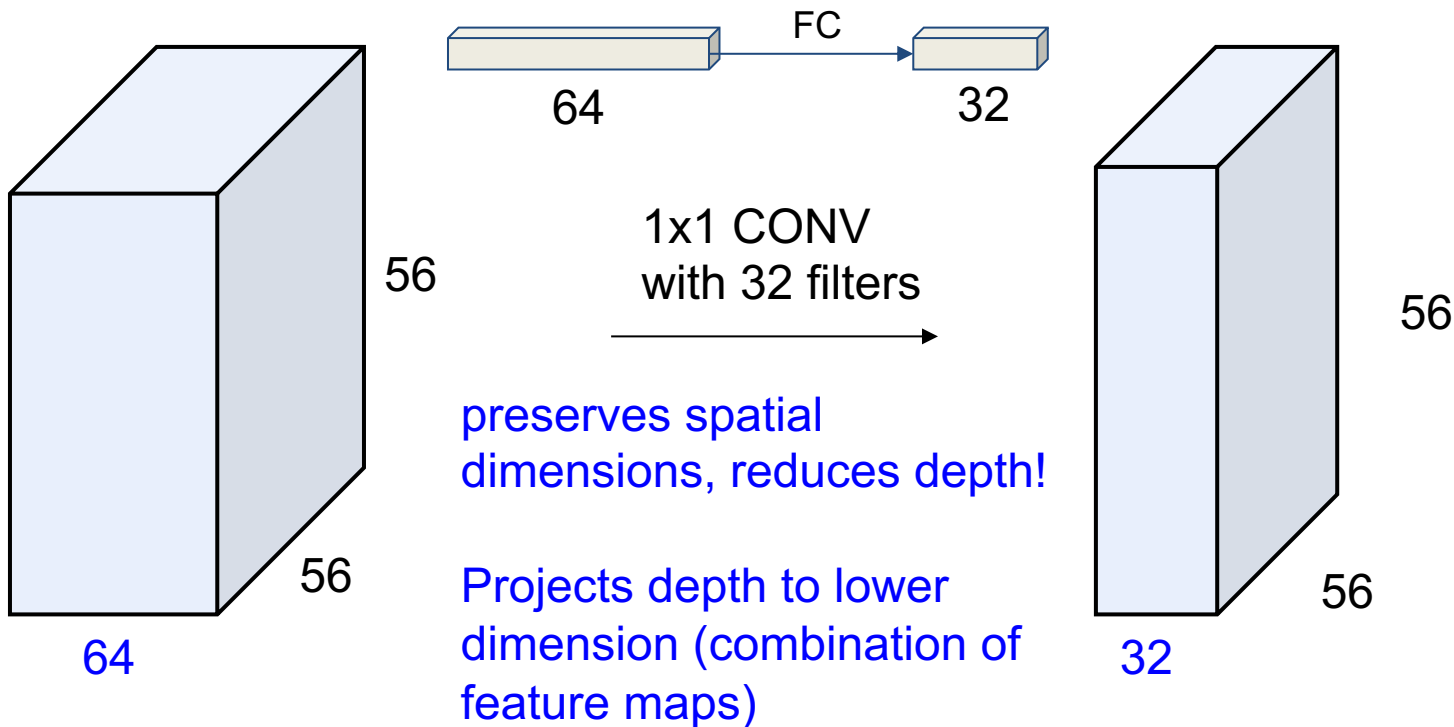
# Review: 1x1 convolutions

Alternatively, interpret it as applying the same FC layer on each input pixel



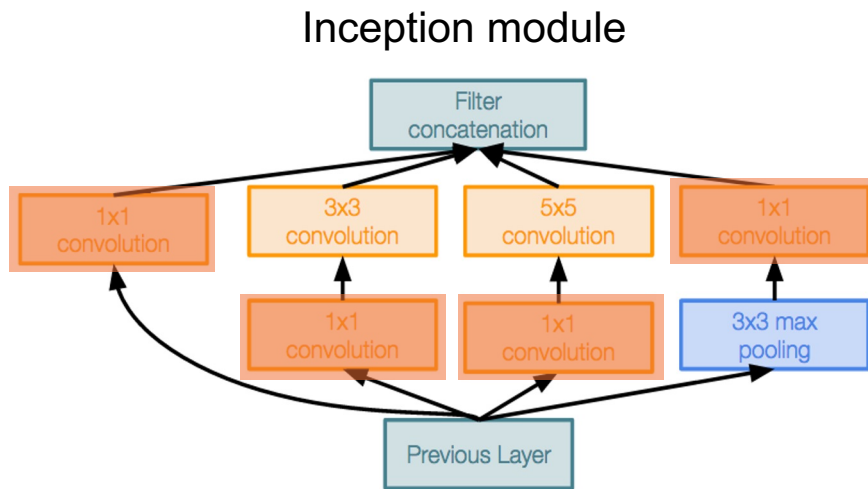
# Review: 1x1 convolutions

Alternatively, interpret it as applying the same FC layer on each input pixel

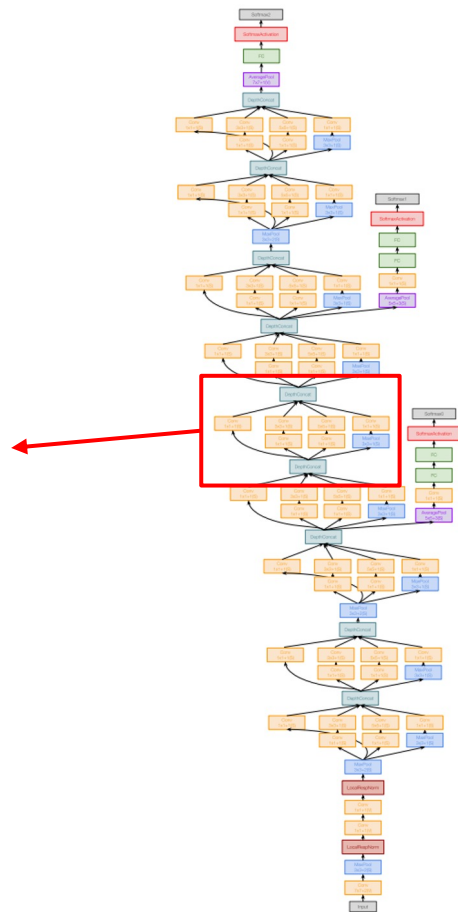


# Case Study: GoogLeNet

[Szegedy et al., 2014]



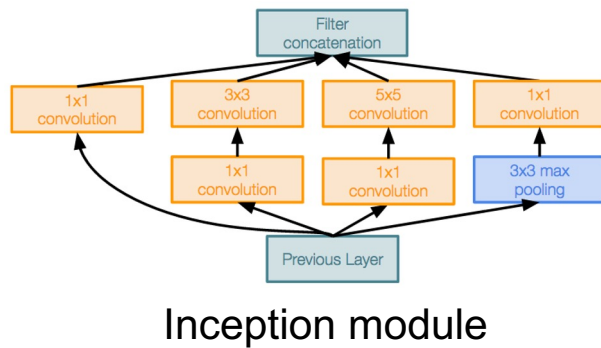
Uses 1x1 “Bottleneck” layers to reduce channel dimension before expensive conv (we will revisit this with ResNet!)



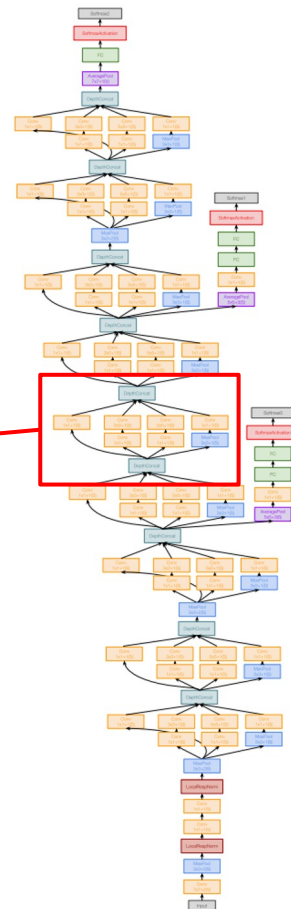
# Case Study: GoogLeNet

[Szegedy et al., 2014]

Stack Inception modules with dimension reduction on top of each other



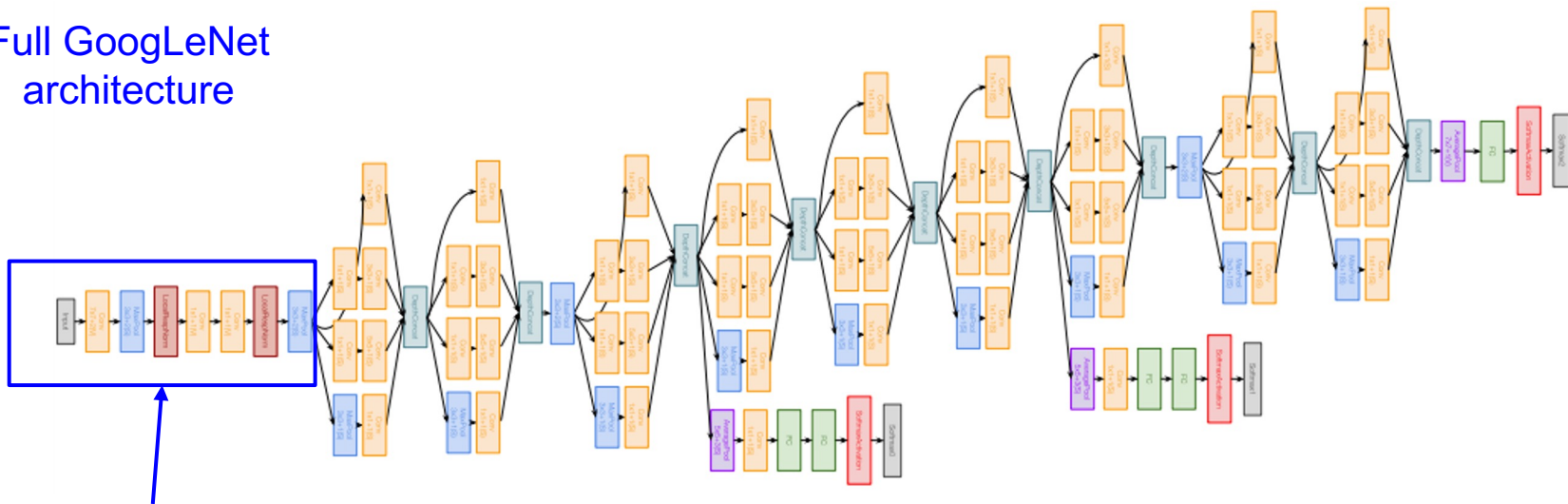
Inception module



# Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet  
architecture

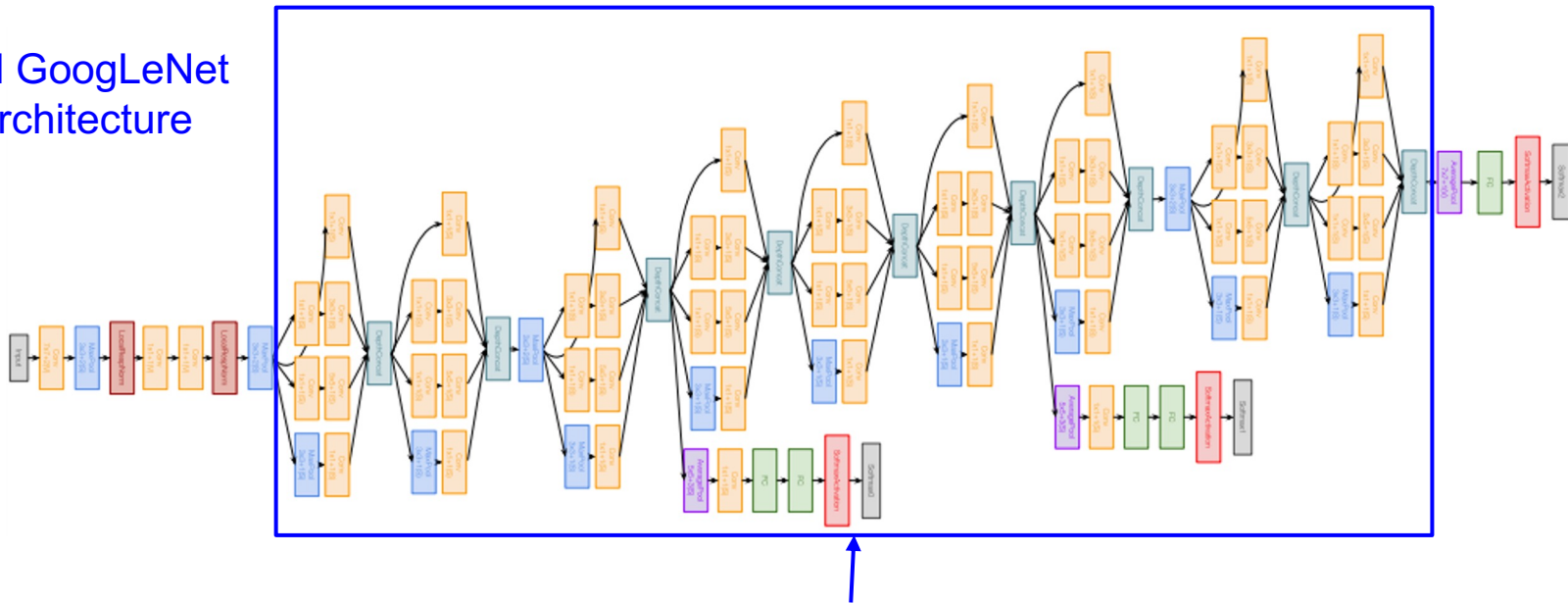


Stem Network:  
Conv-Pool-  
2x Conv-Pool

# Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet  
architecture



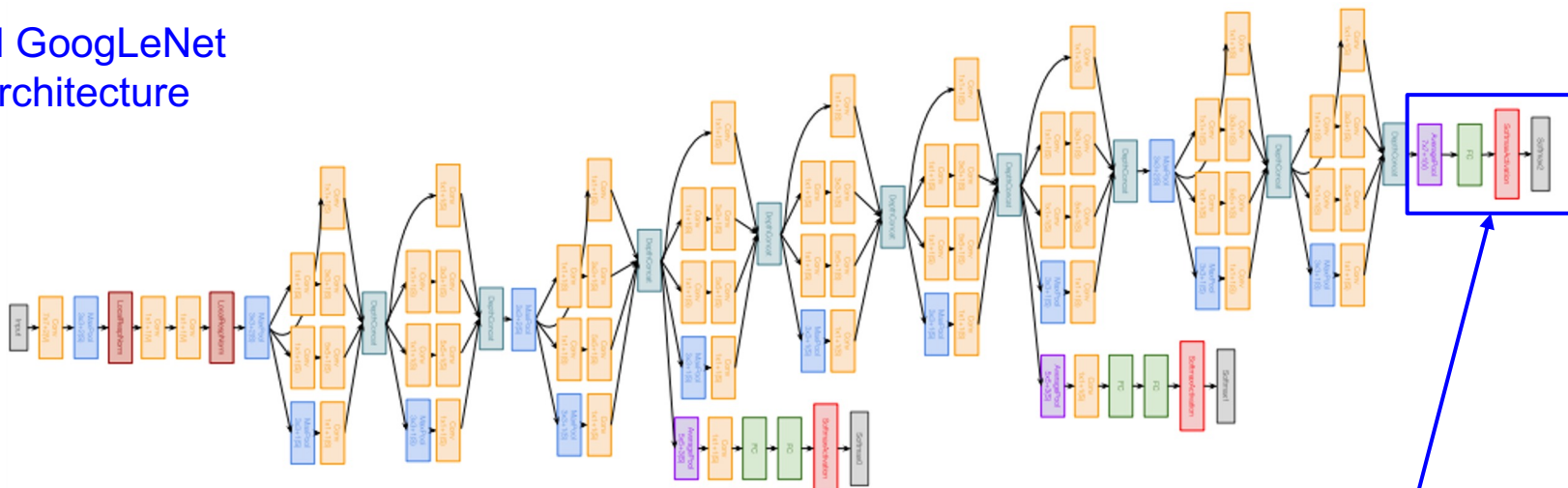
Stacked Inception  
Modules



# Case Study: GoogLeNet

[Szegedy et al., 2014]

## Full GoogLeNet architecture

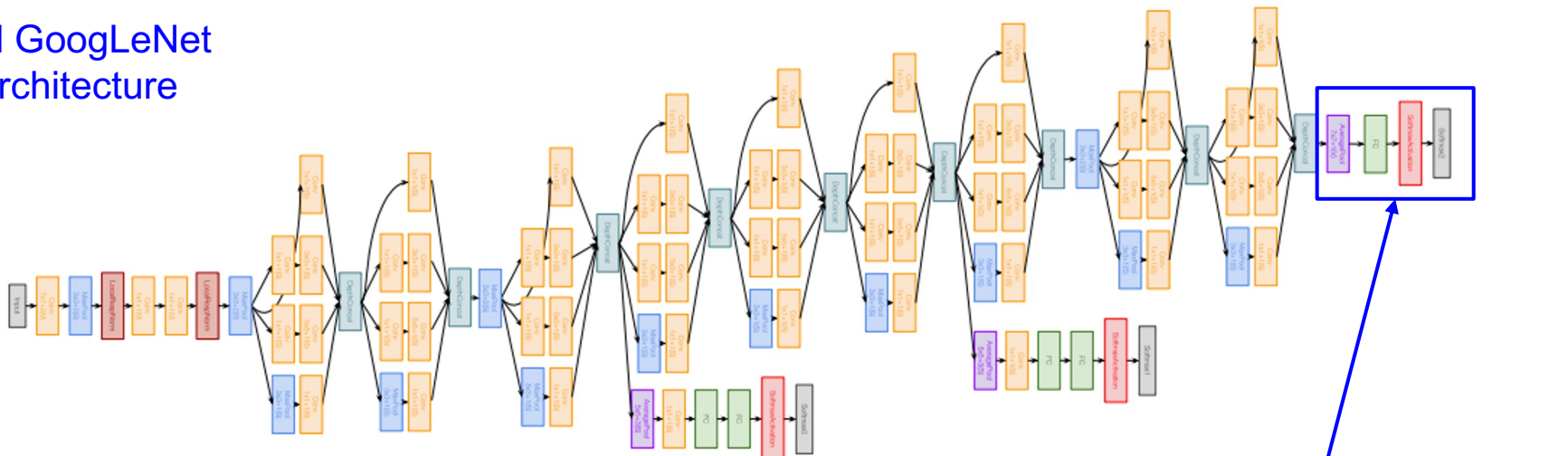


Classifier output

# Case Study: GoogLeNet

[Szegedy et al., 2014]

## Full GoogLeNet architecture



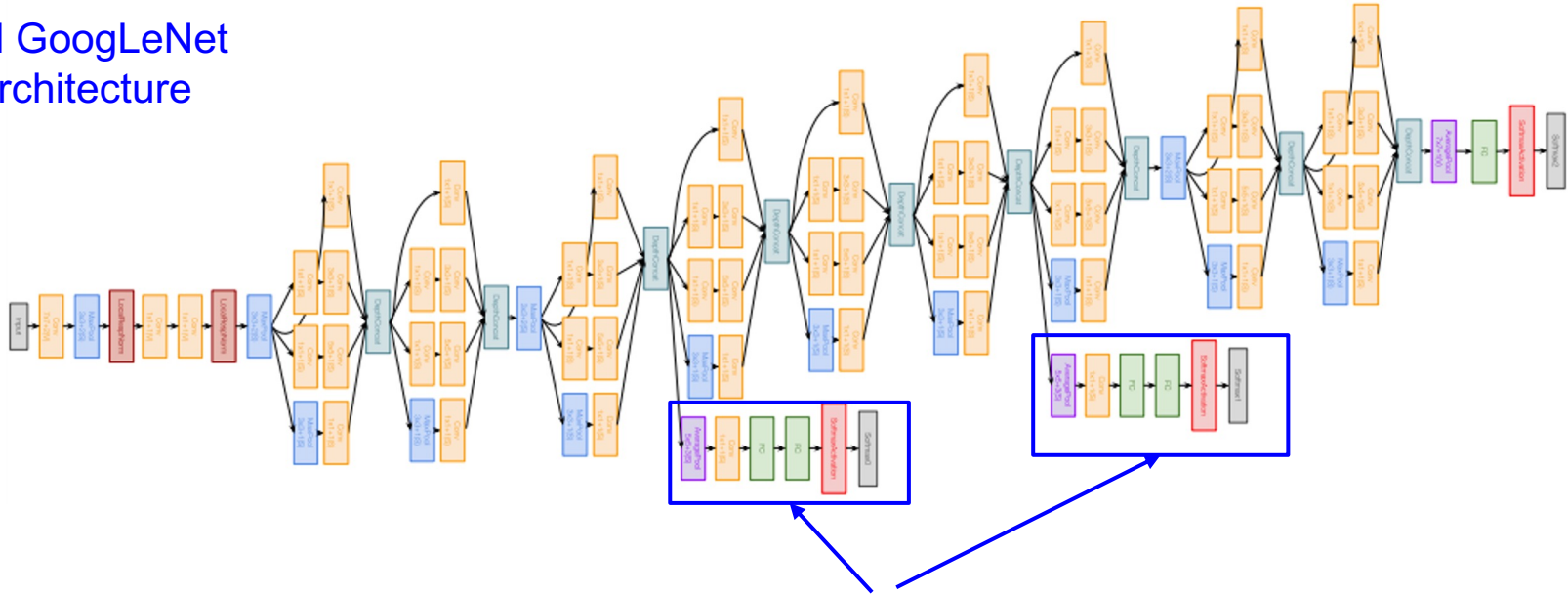
Note: after the last convolutional layer, a global average pooling layer is used that **spatially** averages across each feature map, before final FC layer. No longer multiple expensive FC layers!  
(Also used in ResNet)

Classifier output

# Case Study: GoogLeNet

[Szegedy et al., 2014]

## Full GoogLeNet architecture



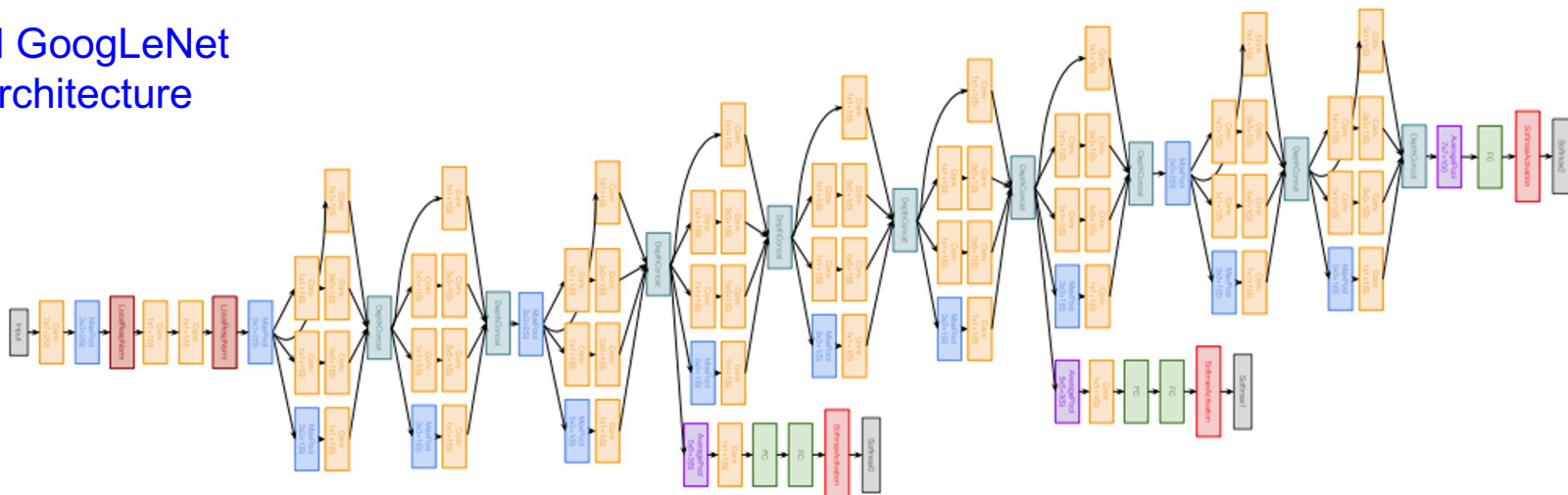
Auxiliary classification outputs to inject additional gradient at lower layers (AvgPool-1x1Conv-FC-FC-Softmax)

Related to **vanishing gradient**, will discuss further in Lecture 10

# Case Study: GoogLeNet

[Szegedy et al., 2014]

## Full GoogLeNet architecture



22 total layers with weights

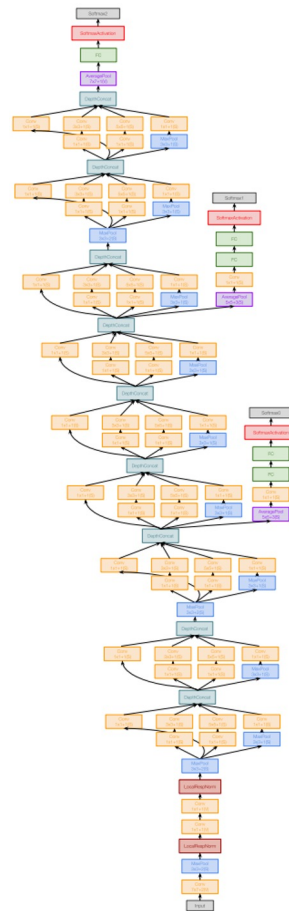
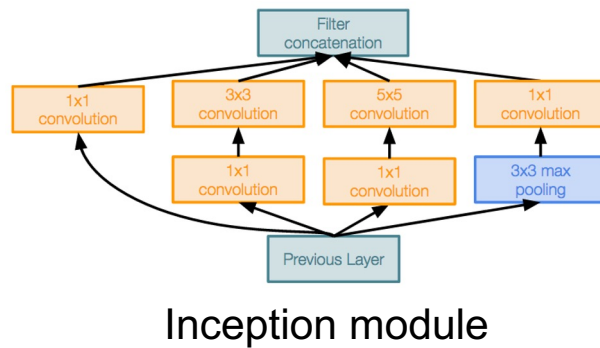
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

# Case Study: GoogLeNet

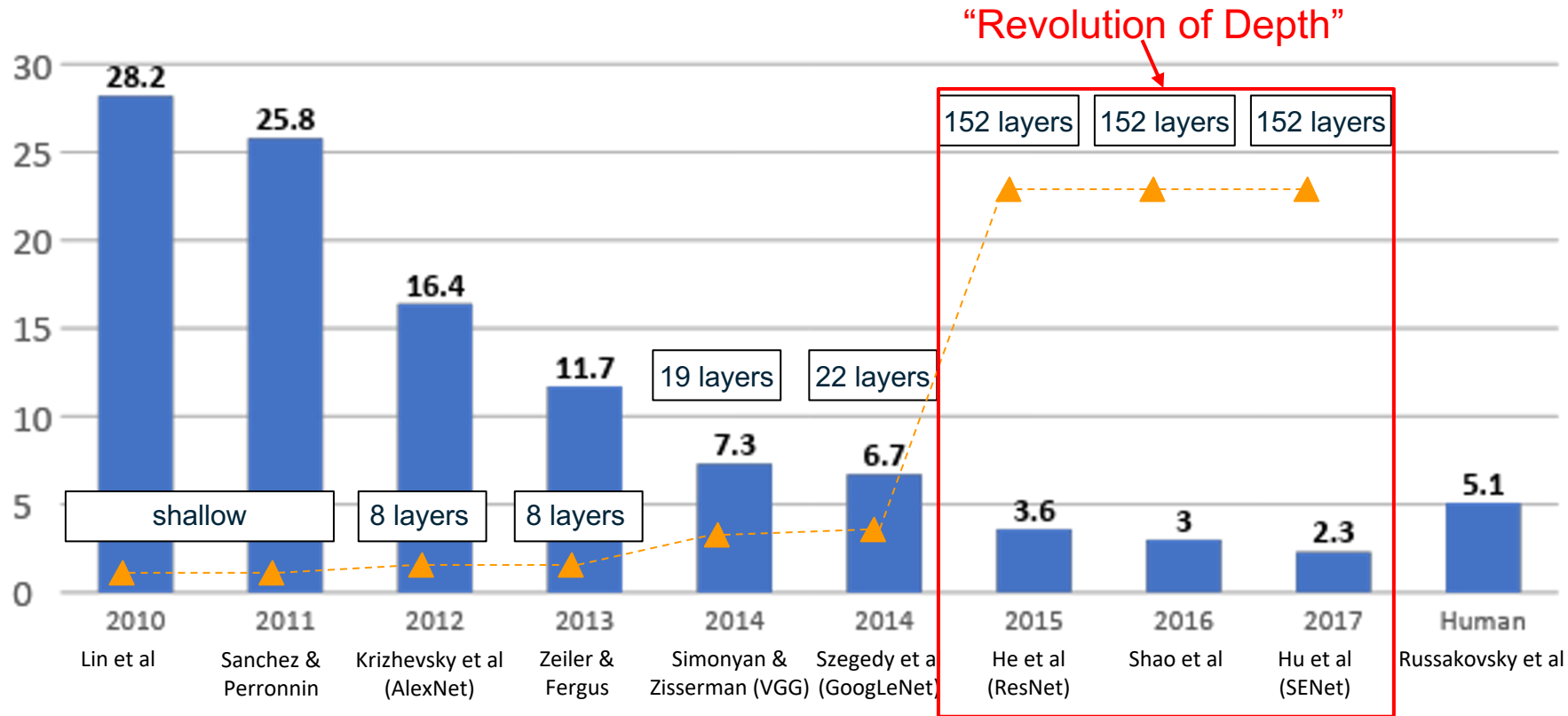
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC’14 classification winner (6.7% top 5 error)



# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

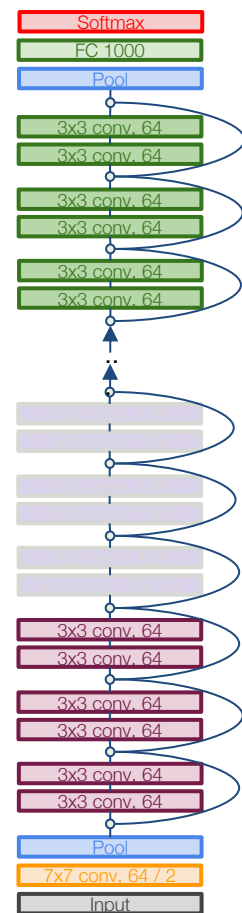
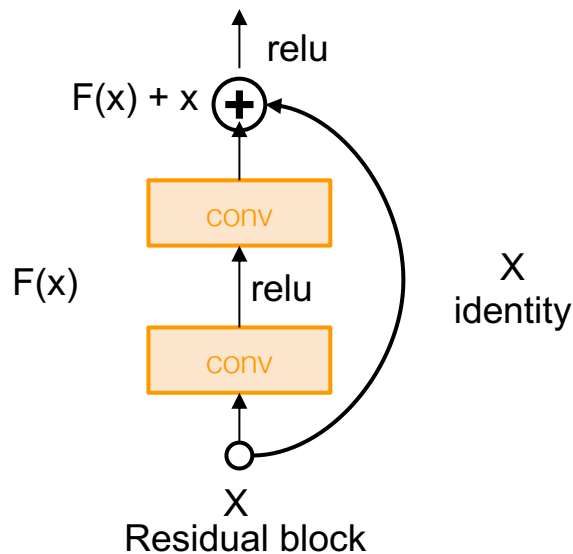


# Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

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



# Case Study: ResNet

*[He et al., 2015]*

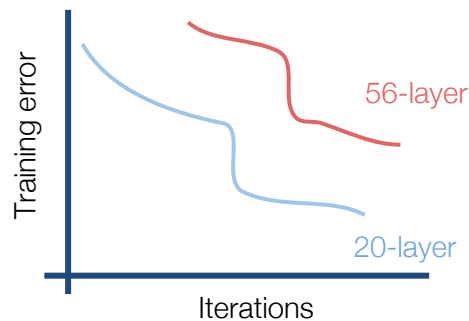
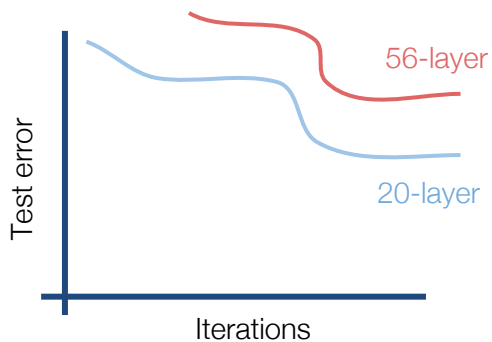
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



# Case Study: ResNet

[He et al., 2015]

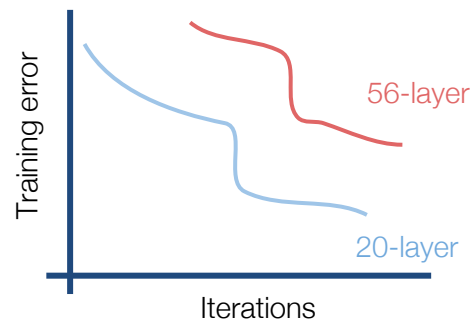
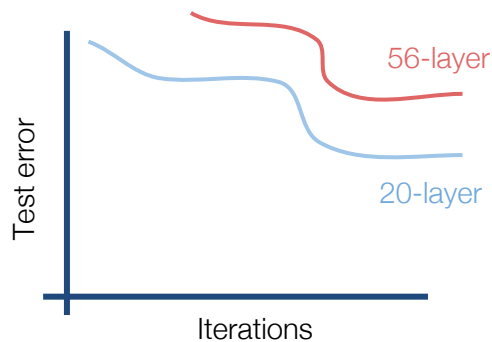
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



# Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



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

# Case Study: ResNet

[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

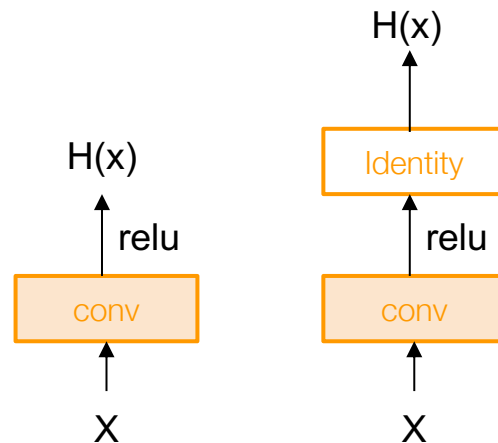
Hypothesis: the problem is an *optimization* problem,  
**deeper models are harder to optimize**

# Case Study: ResNet

[He et al., 2015]

A deeper model can **emulate** a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models



# Case Study: ResNet

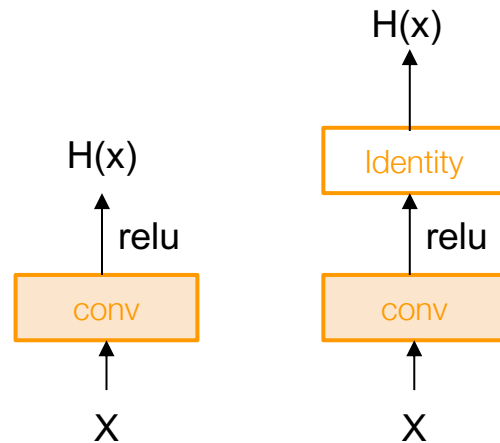
[He et al., 2015]

A deeper model can **emulate** a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models

Deeper models are harder to optimize. They don't learn identity functions to emulate shallow models

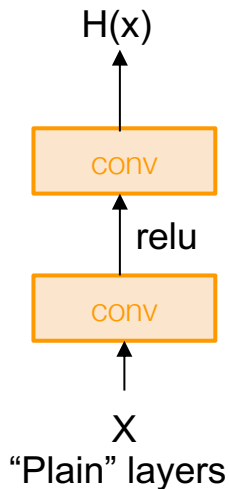
**Solution:** Change the network so learning identity functions (no-op) as extra layers is easy



# Case Study: ResNet

[He et al., 2015]

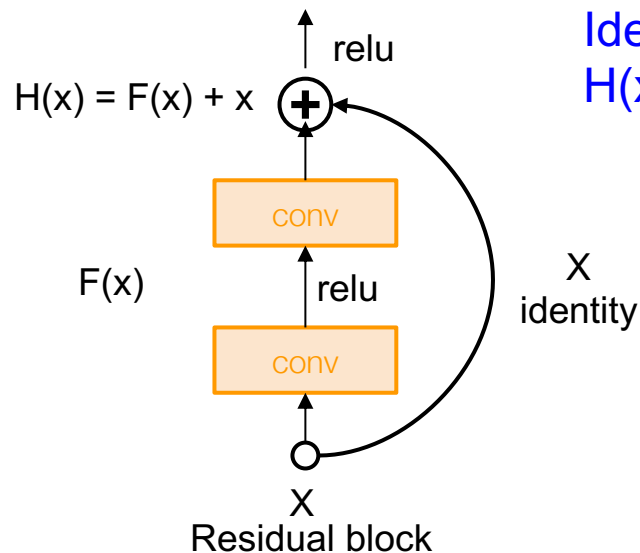
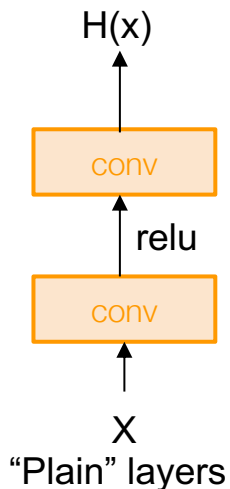
Solution: Change the network so learning identity functions as extra layers is easy



# Case Study: ResNet

[He et al., 2015]

Solution: Change the network so learning identity functions as extra layers is easy

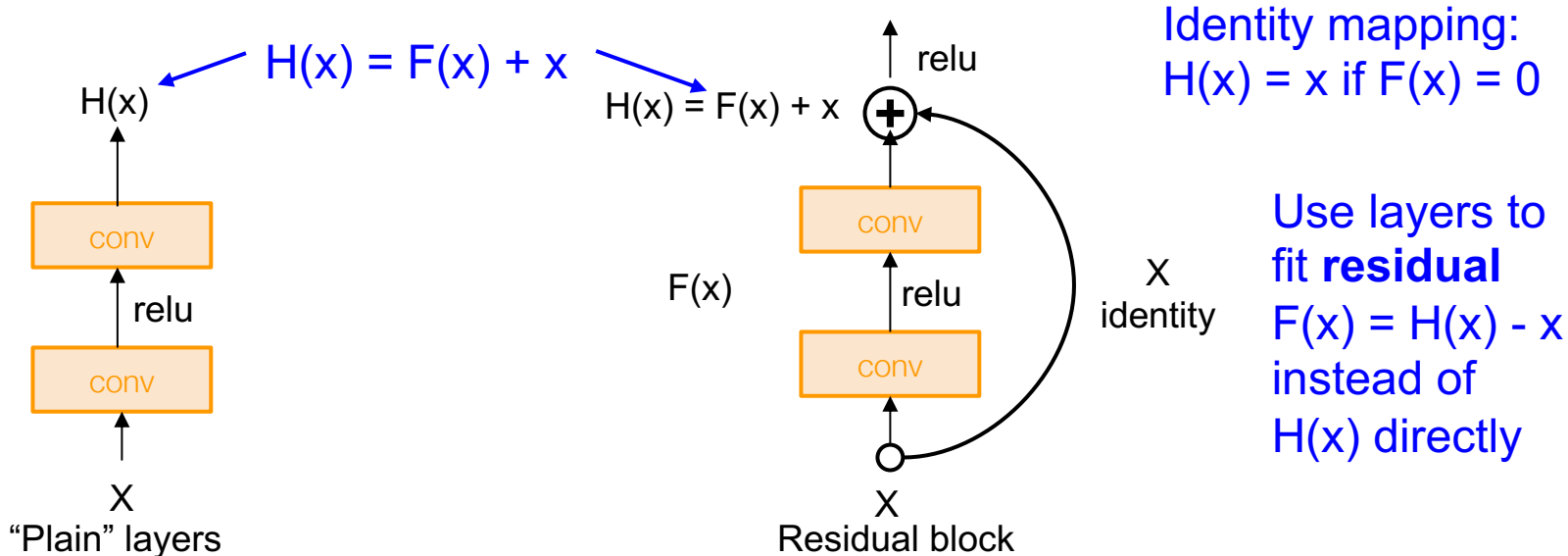


Identity mapping:  
 $H(x) = x$  if  $F(x) = 0$

# Case Study: ResNet

[He et al., 2015]

Solution: Change the network so learning identity functions as extra layers is easy



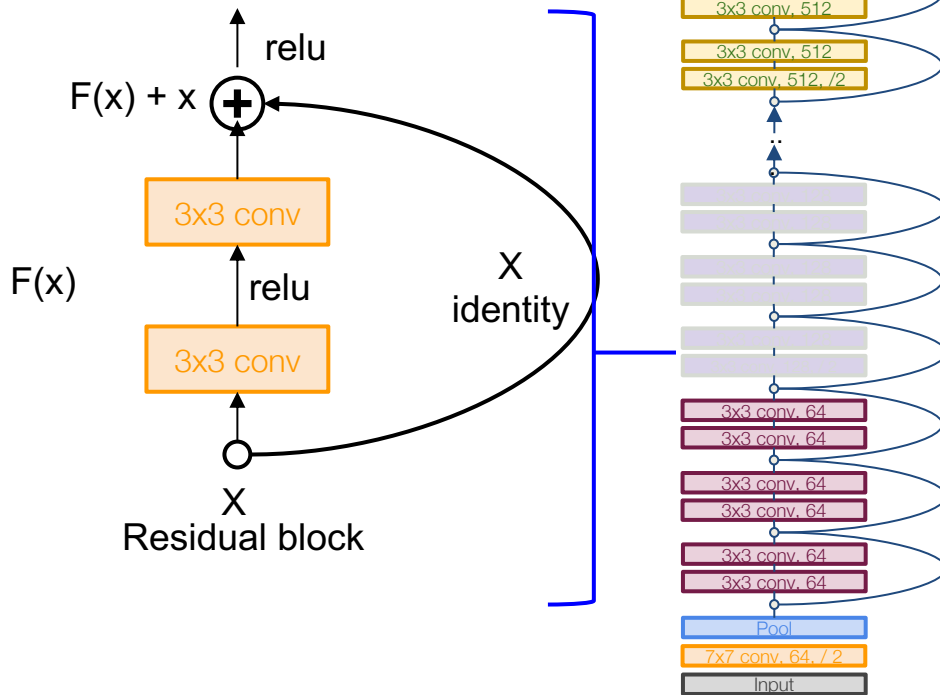


# Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers

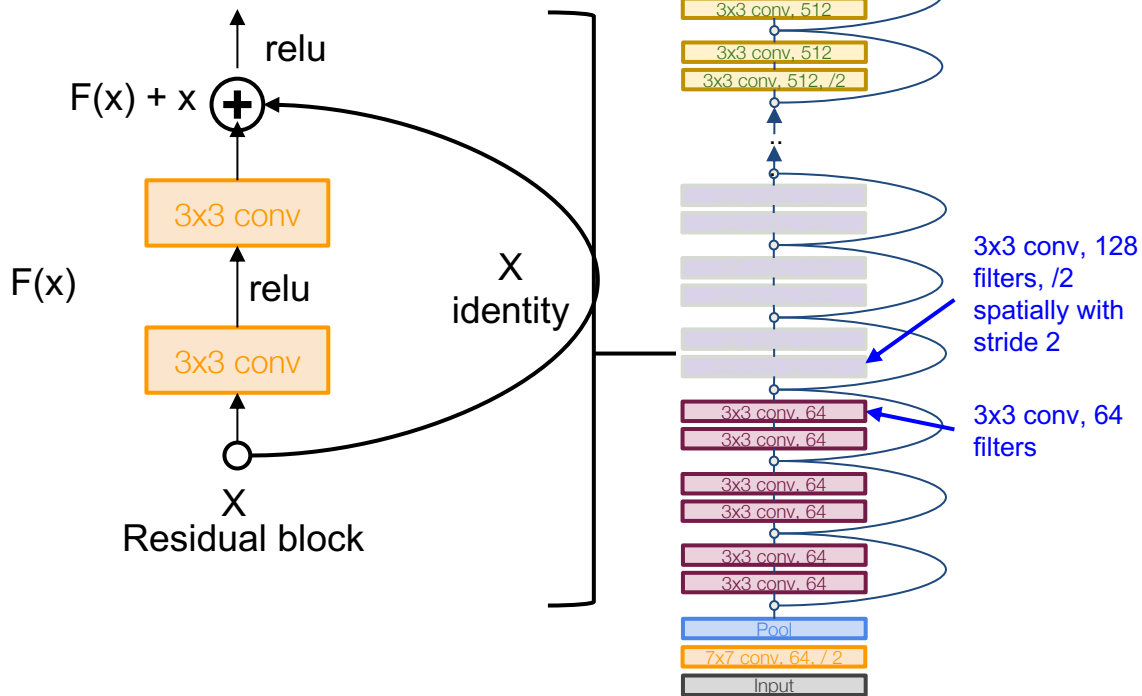


# Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
  - Every residual block has two 3x3 conv layers
  - Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Reduce the activation volume by half.

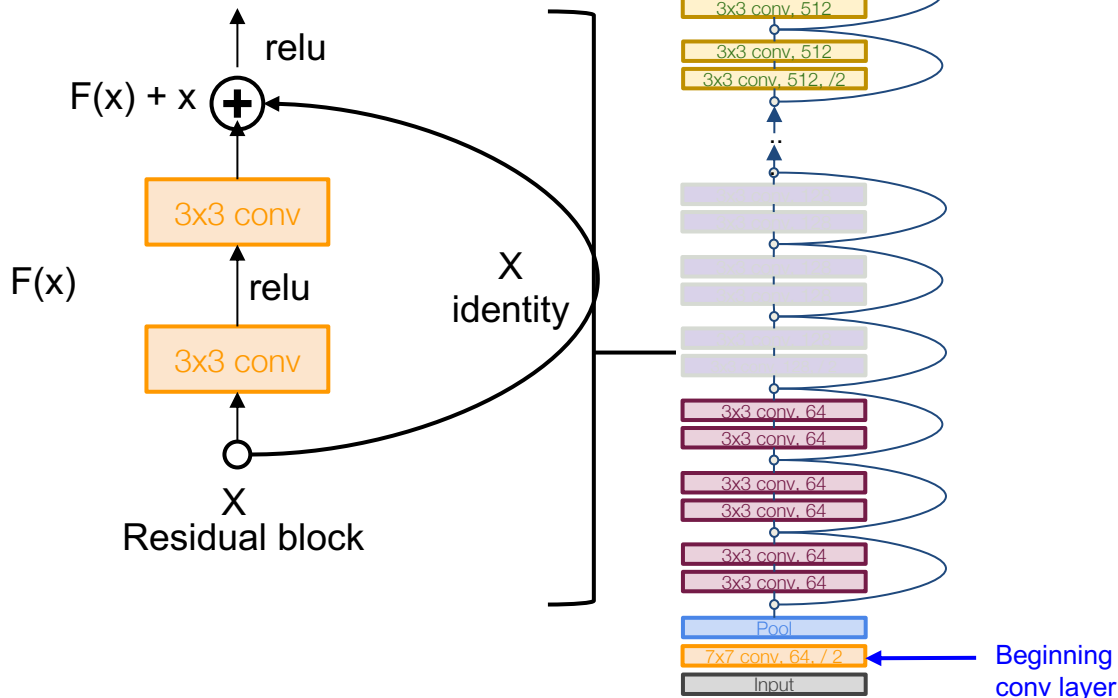


# Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Reduce the activation volume by half.
- Additional conv layer at the beginning (stem)

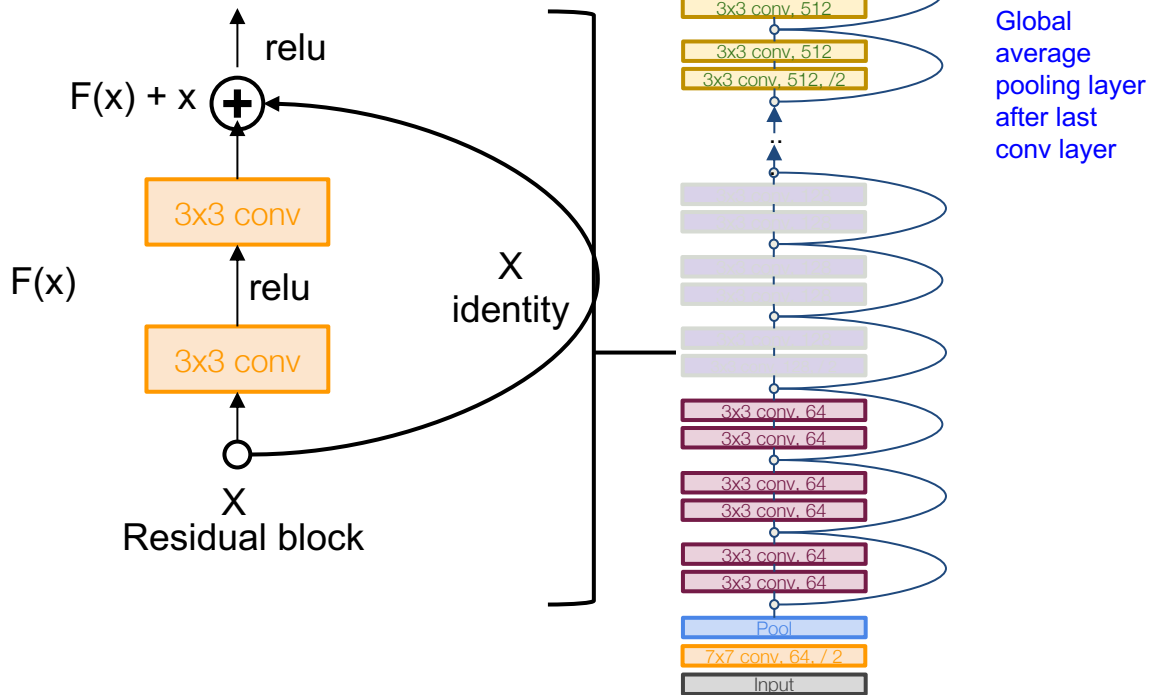


# Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

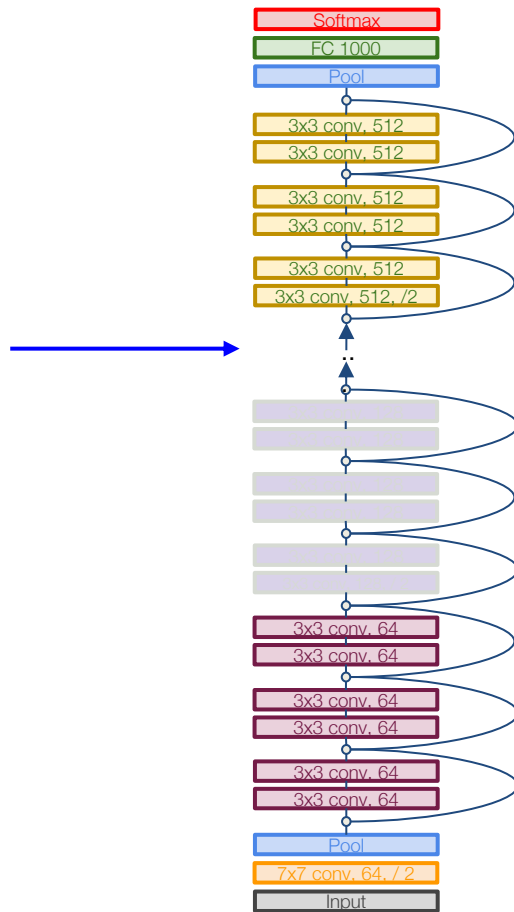
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers at the end (only FC 1000 to output classes)
- (In theory, you can train a ResNet with input image of variable sizes)



# Case Study: ResNet

[He et al., 2015]

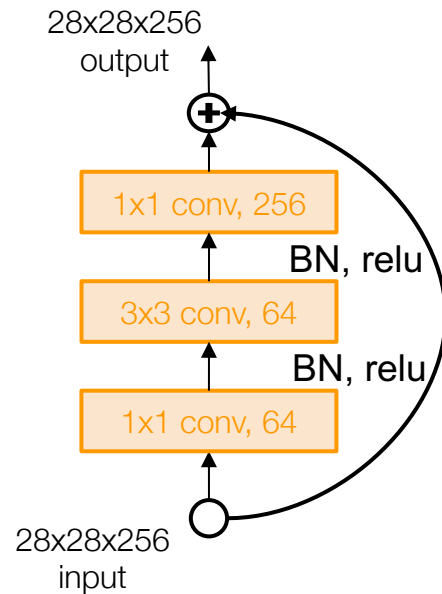
Total depths of 18, 34, 50,  
101, or 152 layers for  
ImageNet



# Case Study: ResNet

[He et al., 2015]

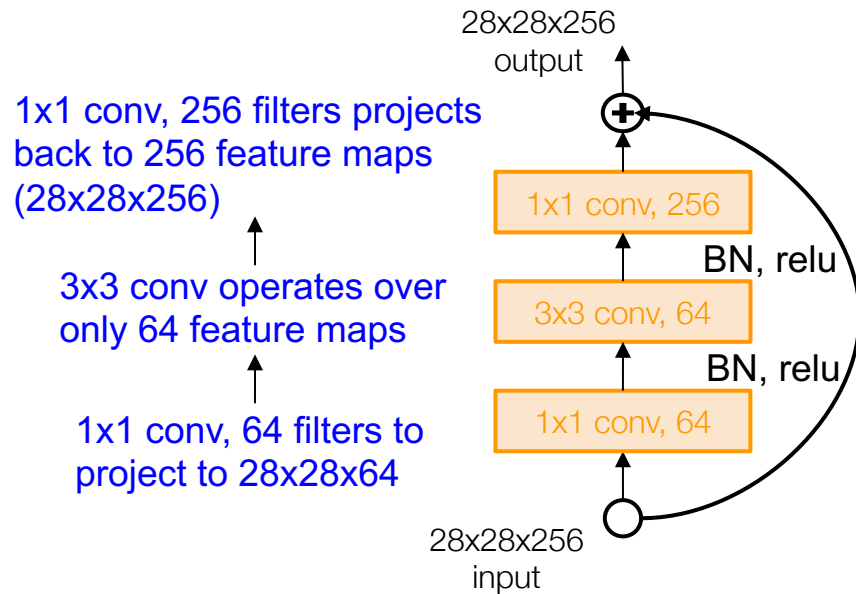
For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)



# Case Study: ResNet

[He et al., 2015]

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



# Case Study: ResNet

*[He et al., 2015]*

Training ResNet in practice:

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



# Case Study: ResNet

[He et al., 2015]

## Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

## MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**

- ImageNet Classification: *“Ultra-deep”* (quote Yann) **152-layer** nets
- ImageNet Detection: **16%** better than 2nd
- ImageNet Localization: **27%** better than 2nd
- COCO Detection: **11%** better than 2nd
- COCO Segmentation: **12%** better than 2nd

# Case Study: ResNet

[He et al., 2015]

## Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

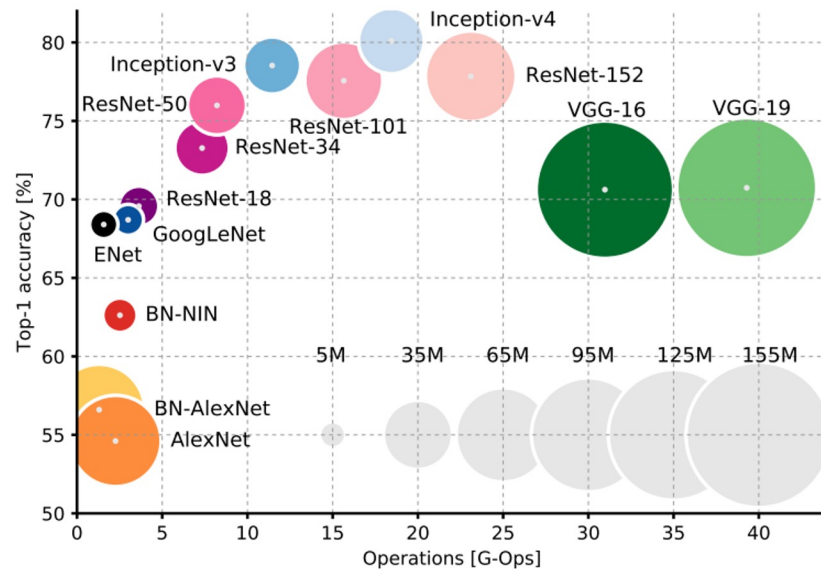
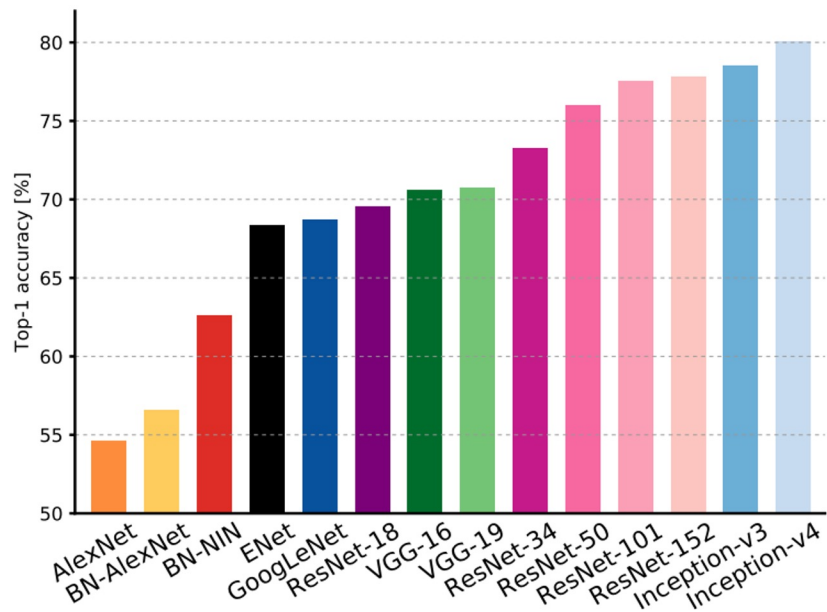
## MSRA @ ILSVRC & COCO 2015 Competitions

### • 1st places in all five main tracks

- ImageNet Classification: “Ultra-deep” (quote Yann) 152-layer nets
- ImageNet Detection: 16% better than 2nd
- ImageNet Localization: 27% better than 2nd
- COCO Detection: 11% better than 2nd
- COCO Segmentation: 12% better than 2nd

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than “human performance”! (Russakovsky 2014)

# Comparing complexity...

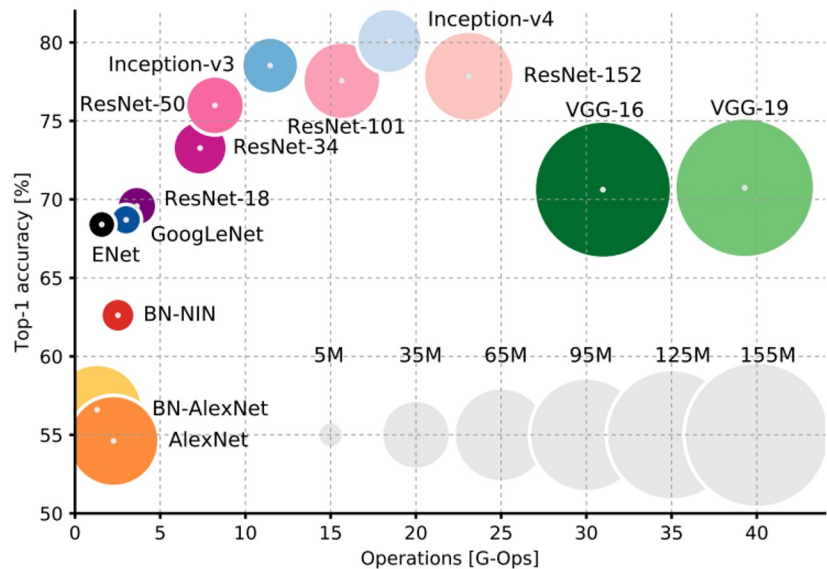
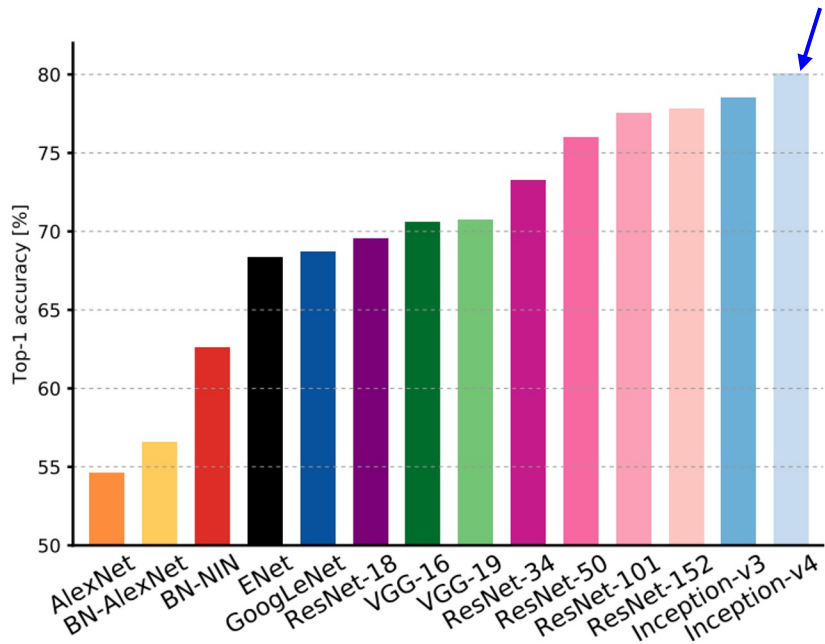


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

# Comparing complexity...

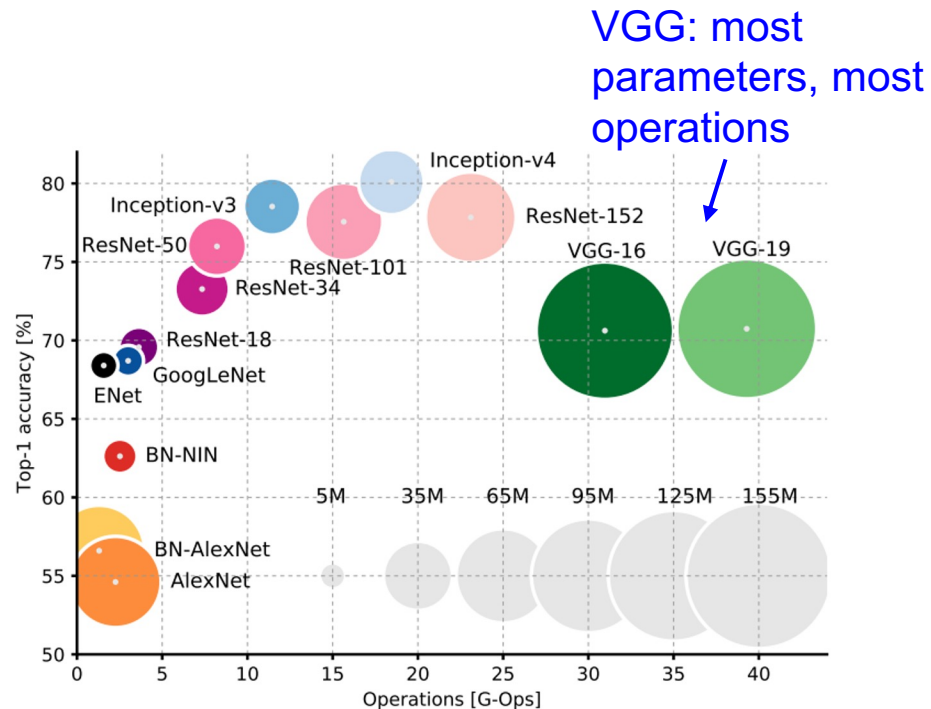
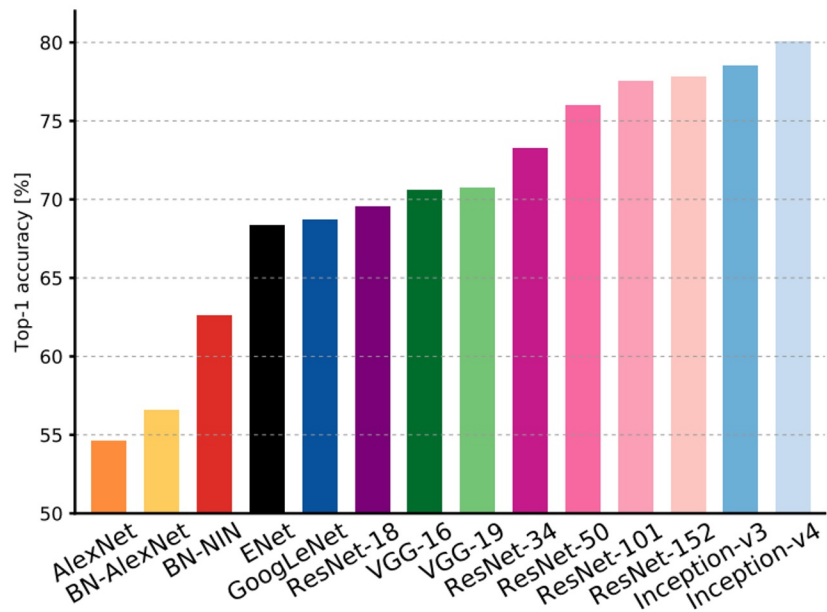
Inception-v4: Resnet + Inception!



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

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

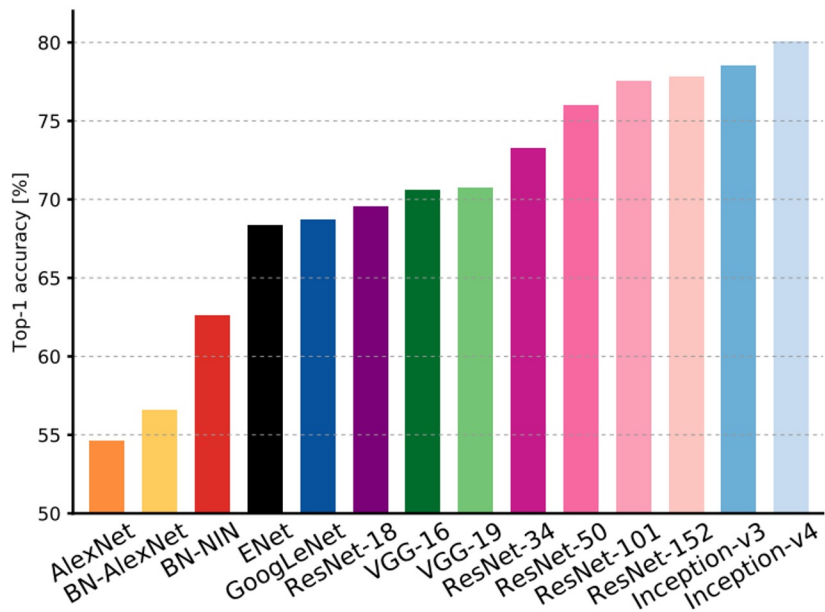
# Comparing complexity...



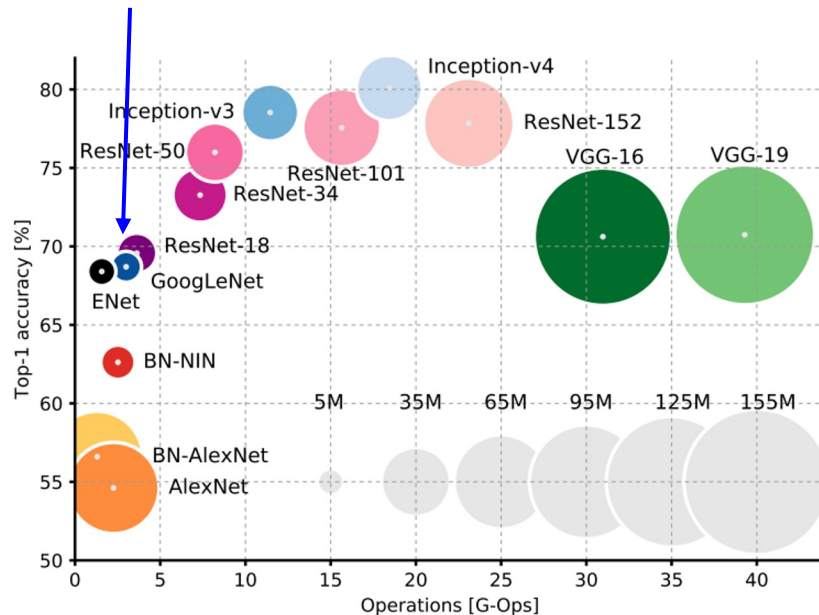
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

# Comparing complexity...



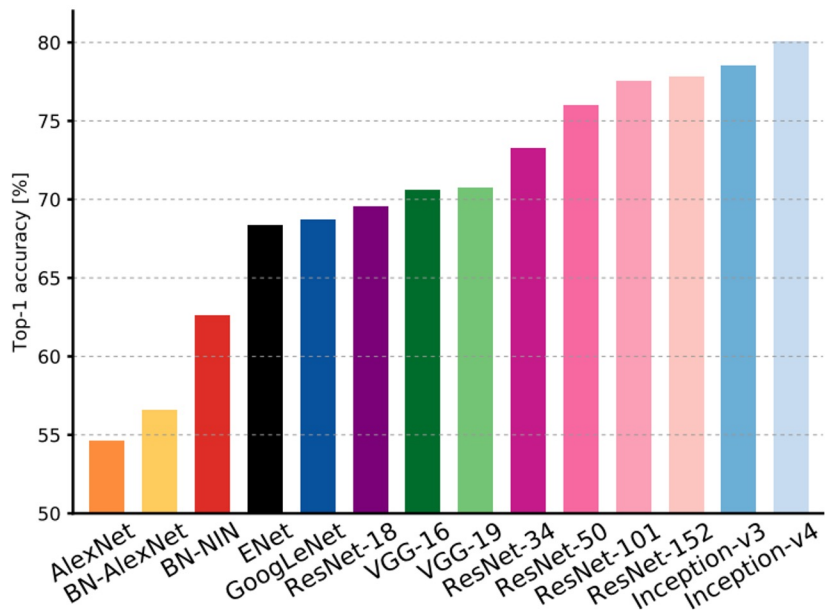
GoogLeNet:  
most efficient



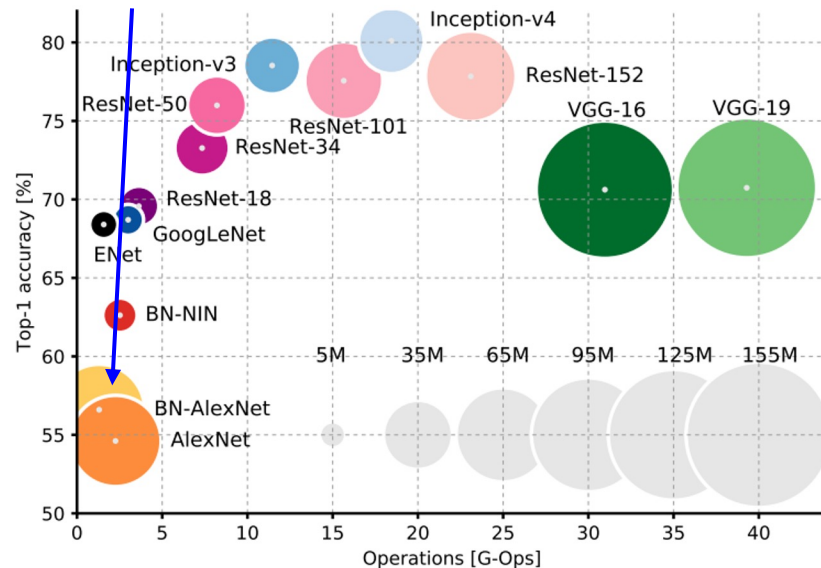
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

# Comparing complexity...



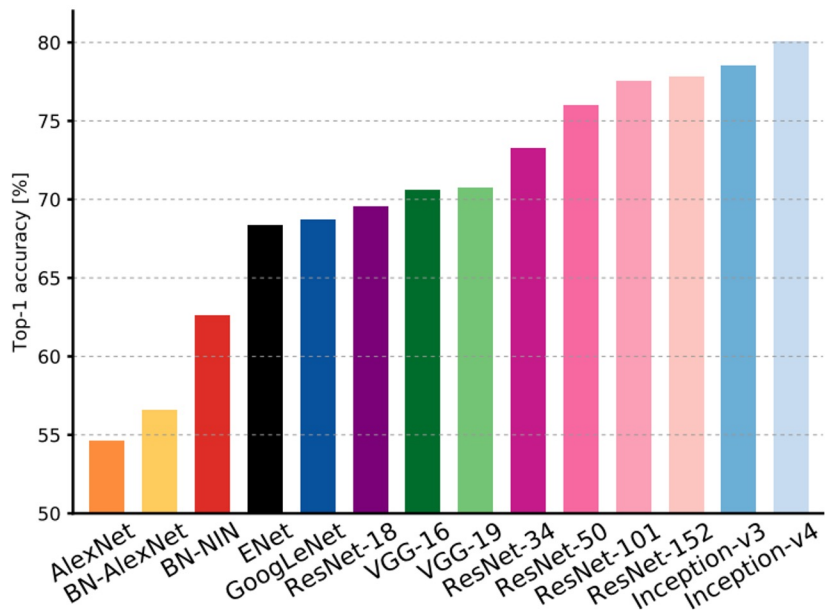
AlexNet:  
Smaller compute, still memory heavy, lower accuracy



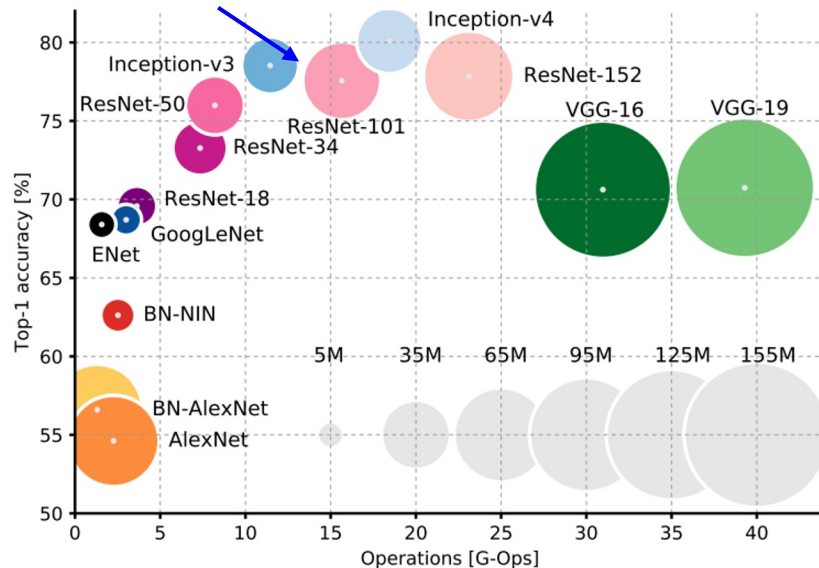
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

# Comparing complexity...



ResNet:  
Moderate efficiency depending on  
model, highest accuracy

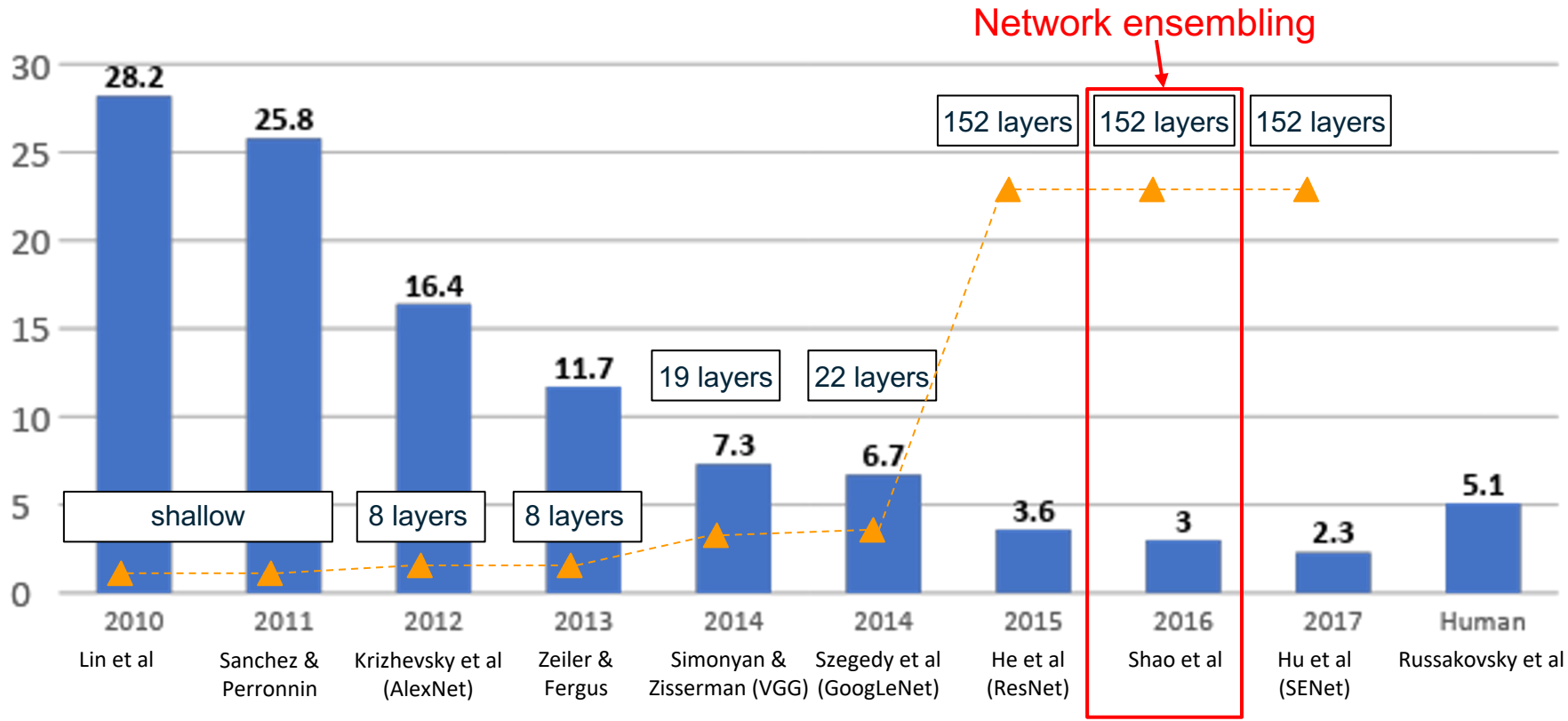


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.



# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# Improving ResNets...

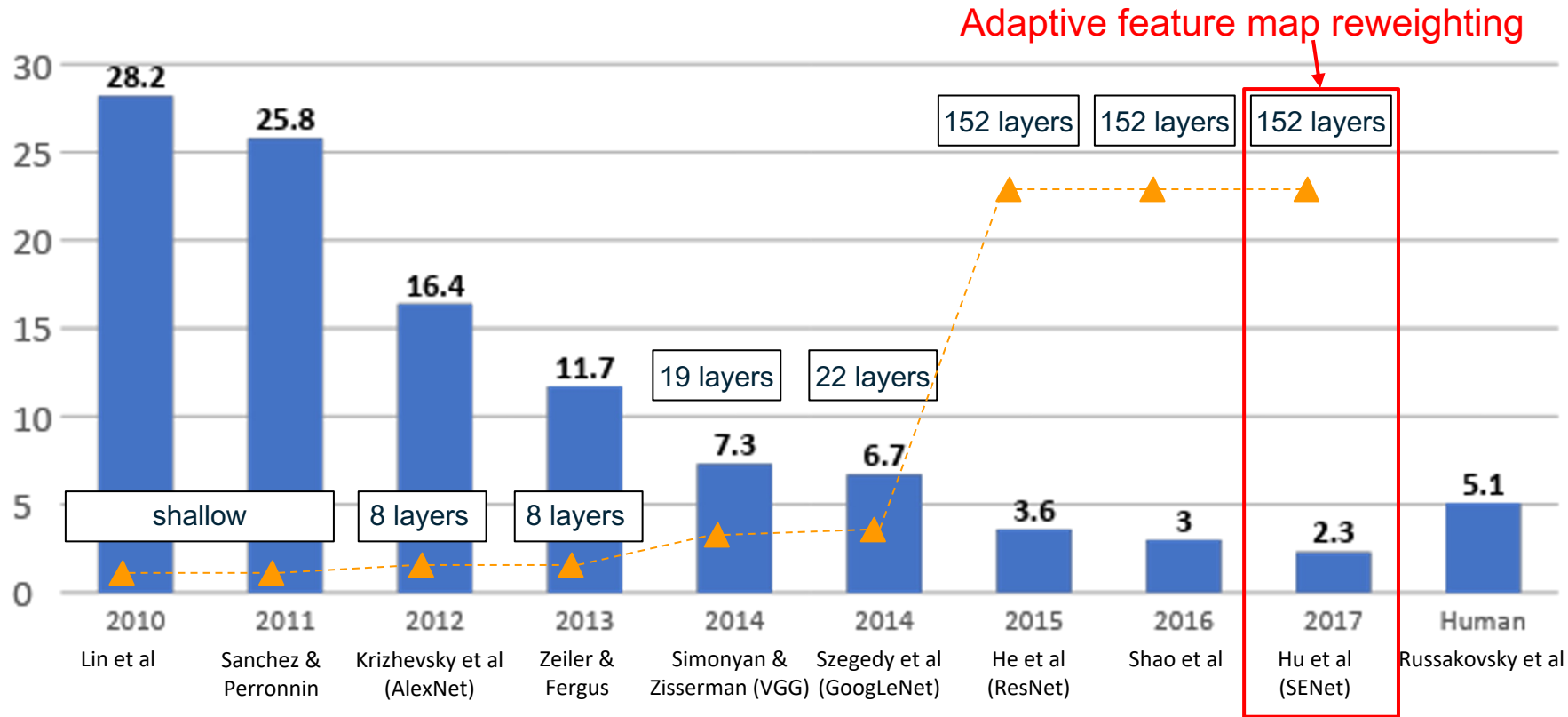
## “Good Practices for Deep Feature Fusion”

[Shao et al. 2016]

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC'16 classification winner

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion (Val.)	Fusion (Test)
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

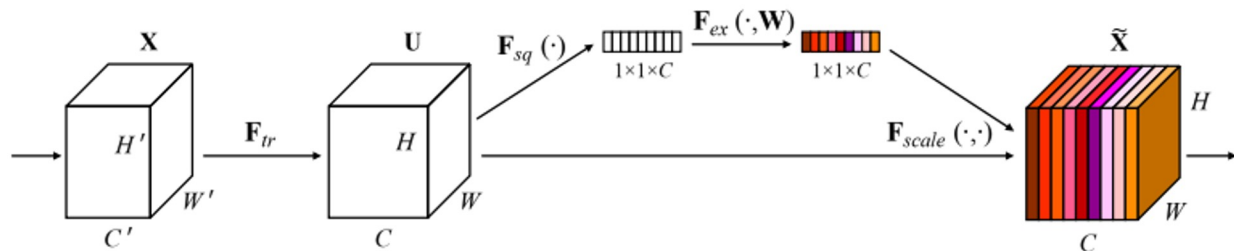
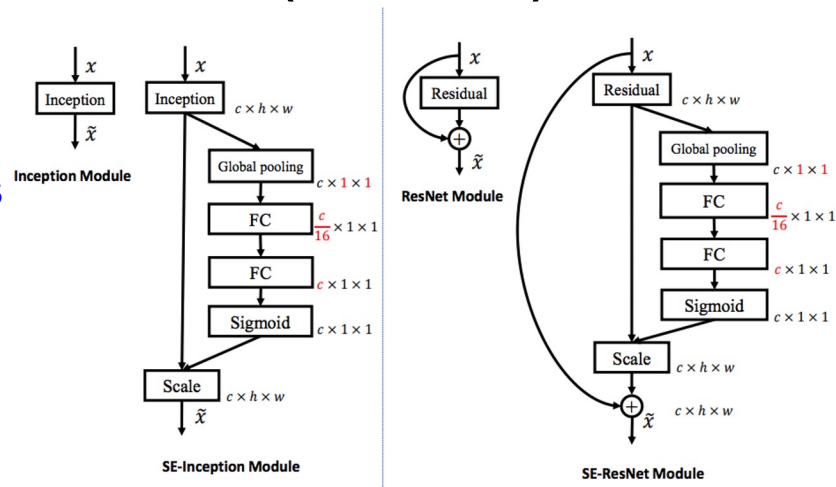


# Improving ResNets...

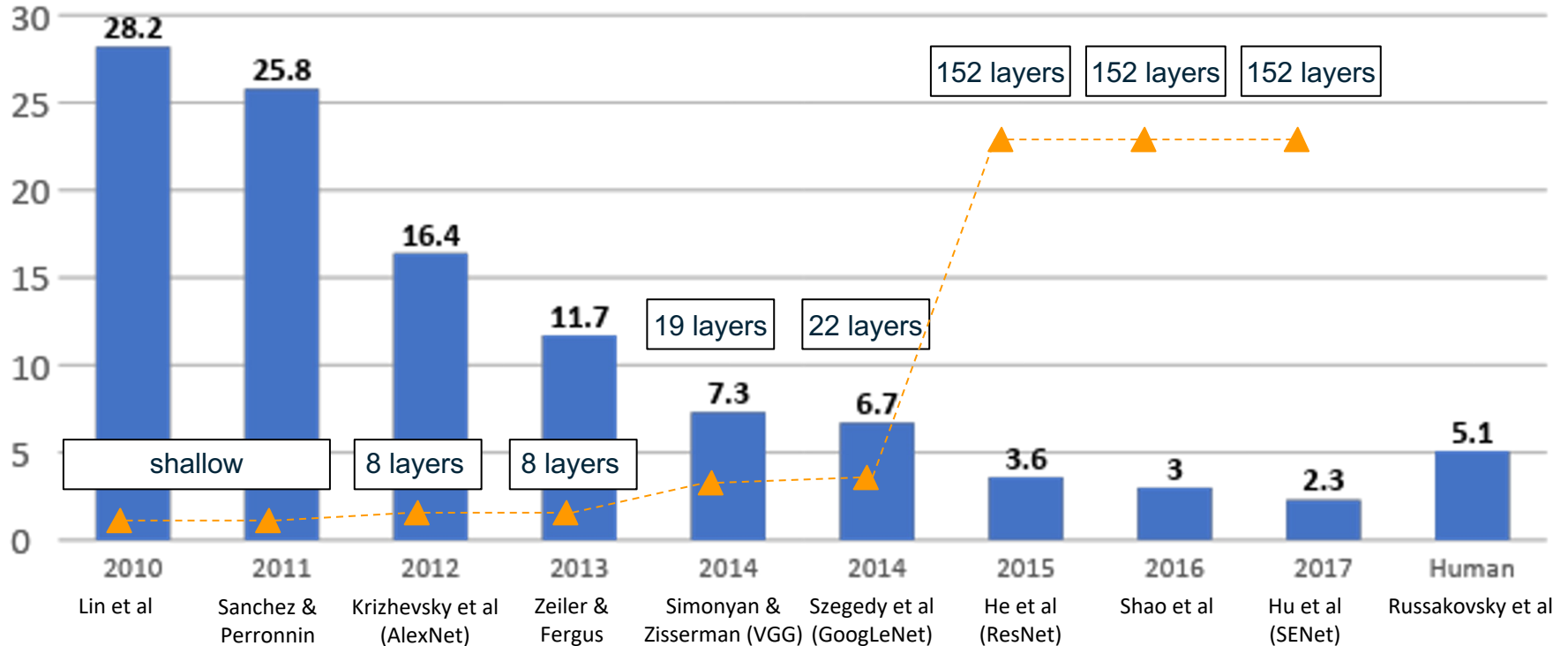
## Squeeze-and-Excitation Networks (SENet)

[Hu et al. 2017]

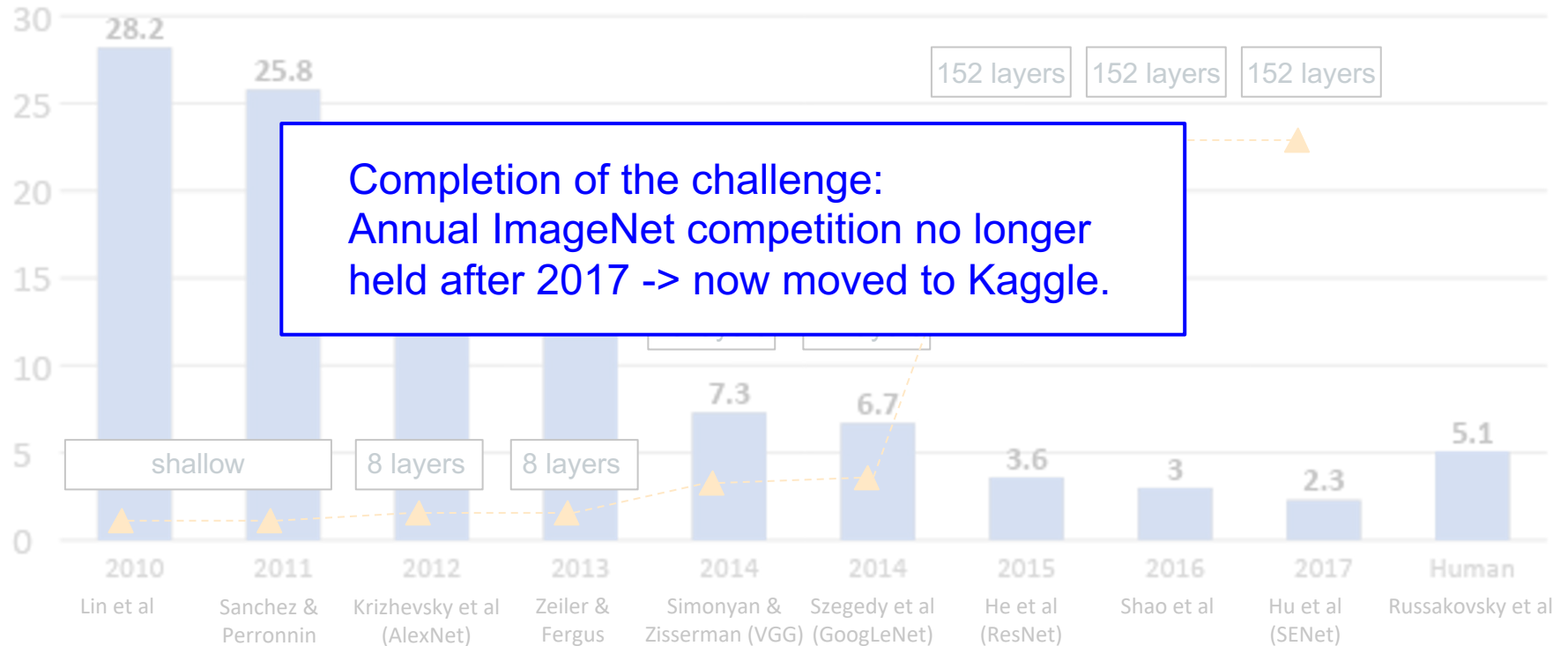
- Add a “feature recalibration” module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC'17 classification winner (using ResNeXt-152 as a base architecture)



# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



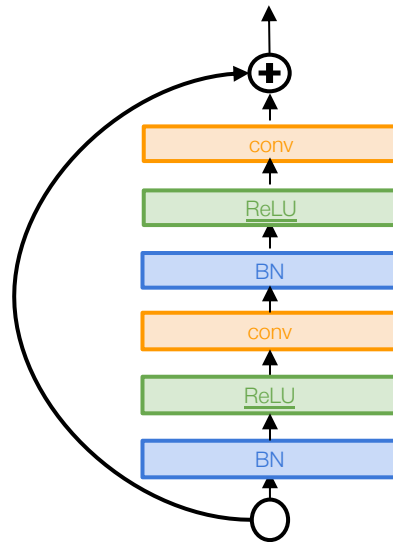
But research into CNN architectures is still flourishing

# Improving ResNets...

## Identity Mappings in Deep Residual Networks

[He et al. 2016]

- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network
- Gives better performance



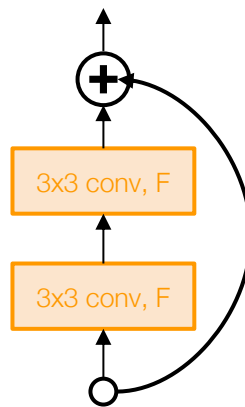


# Improving ResNets...

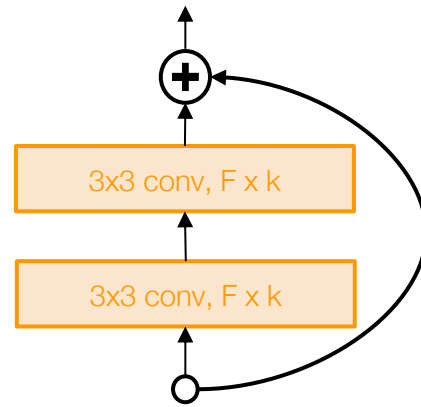
## Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- Use wider residual blocks ( $F \times k$  filters instead of  $F$  filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



Basic residual block



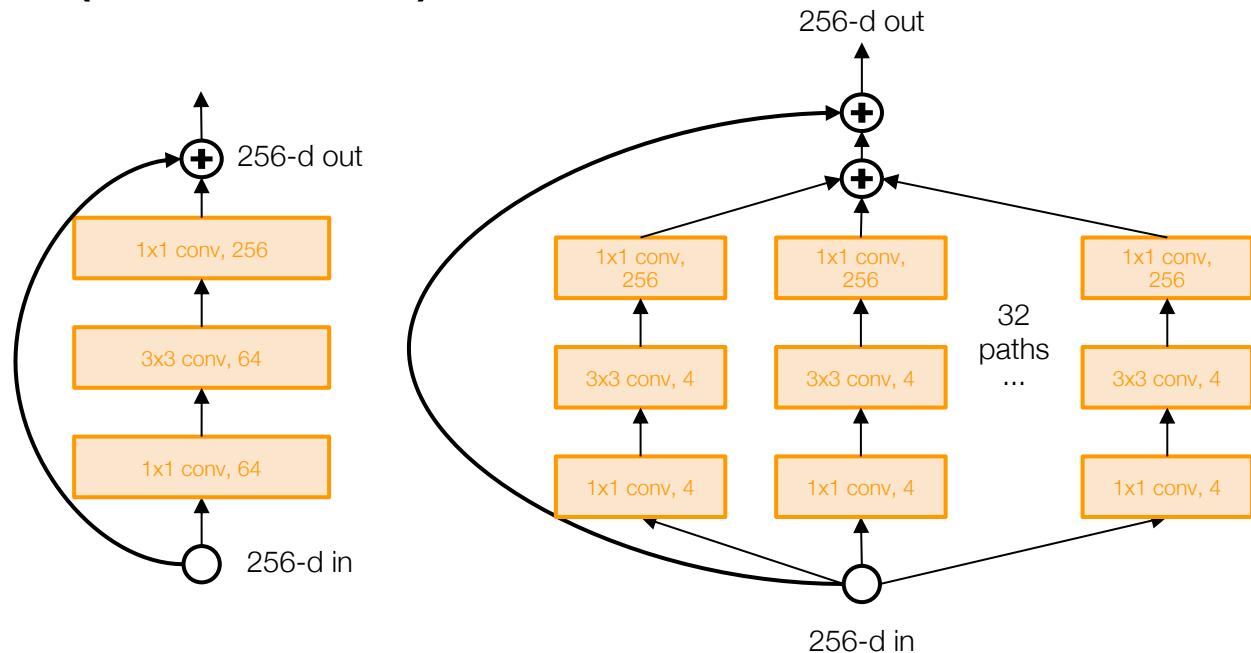
Wide residual block

# Improving ResNets...

# Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

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

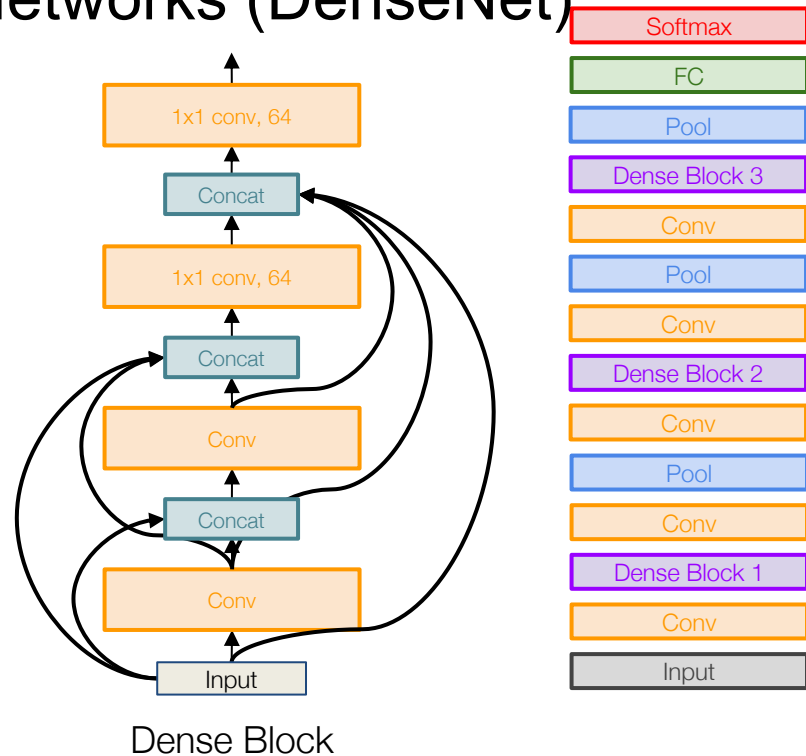


# Other ideas...

## Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet

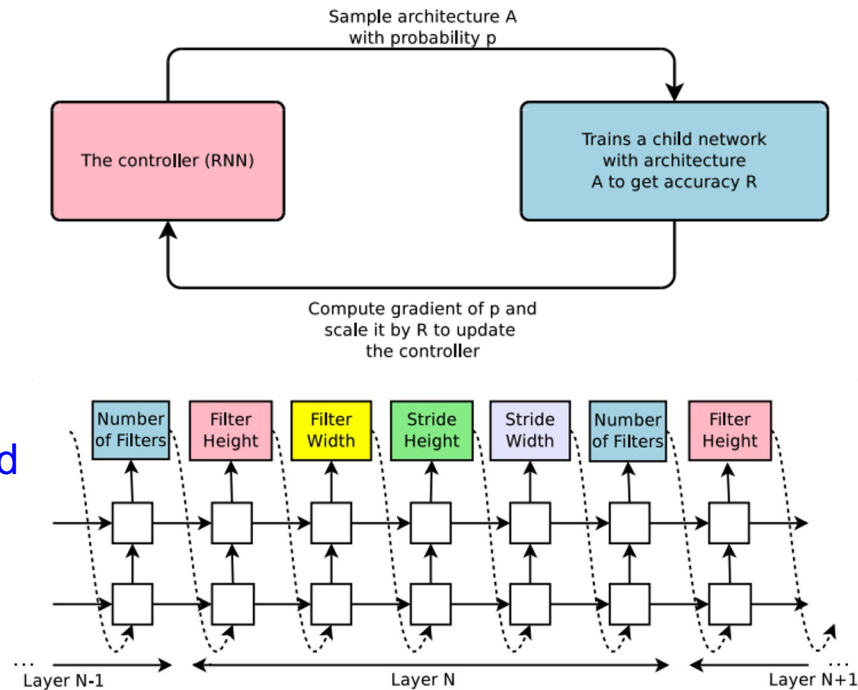


# Learning to search for network architectures...

## Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- “Controller” network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  - 1) Sample an architecture from search space
  - 2) Train the architecture to get a “reward”  $R$  corresponding to accuracy
  - 3) Compute gradient of sample probability, and scale by  $R$  to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)

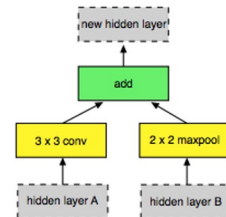
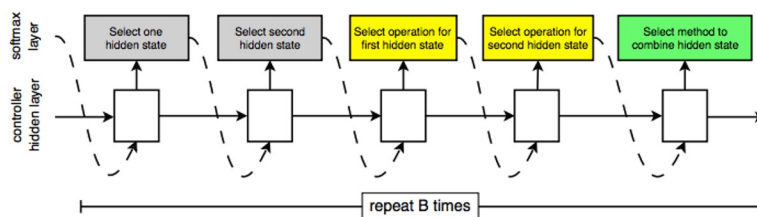
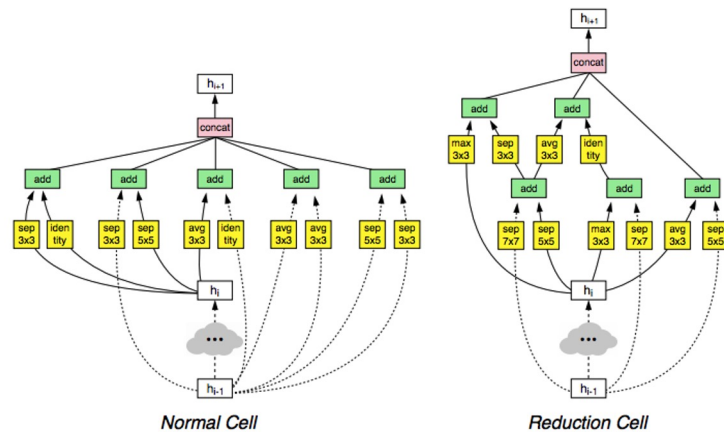


# Learning to search for network architectures...

## Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks (“cells”) that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet
- Many follow-up works in this space e.g. AmoebaNet (Real et al. 2019) and ENAS (Pham, Guan et al. 2018)



# But sometimes smart heuristic is better than NAS ...

## EfficientNet: Smart Compound Scaling

[Tan and Le. 2019]

- Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.
- Search for optimal set of compound scaling factors given a compute budget (target memory & flops).
- Scale up using smart heuristic rules

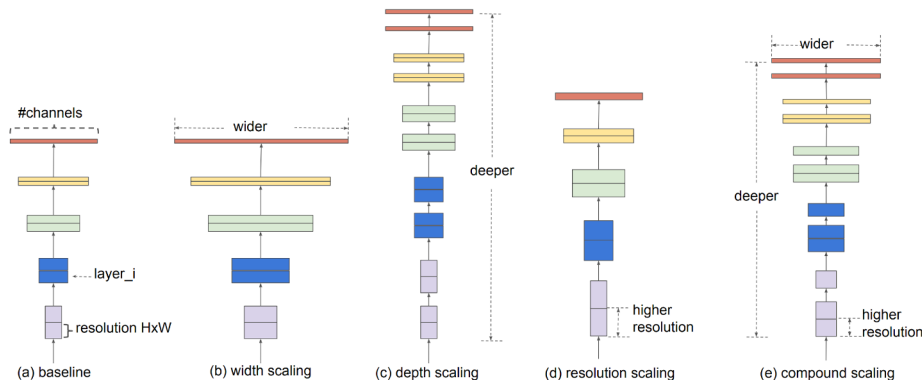
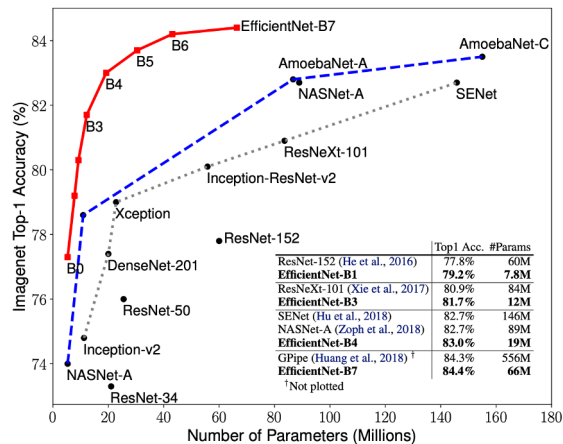
$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

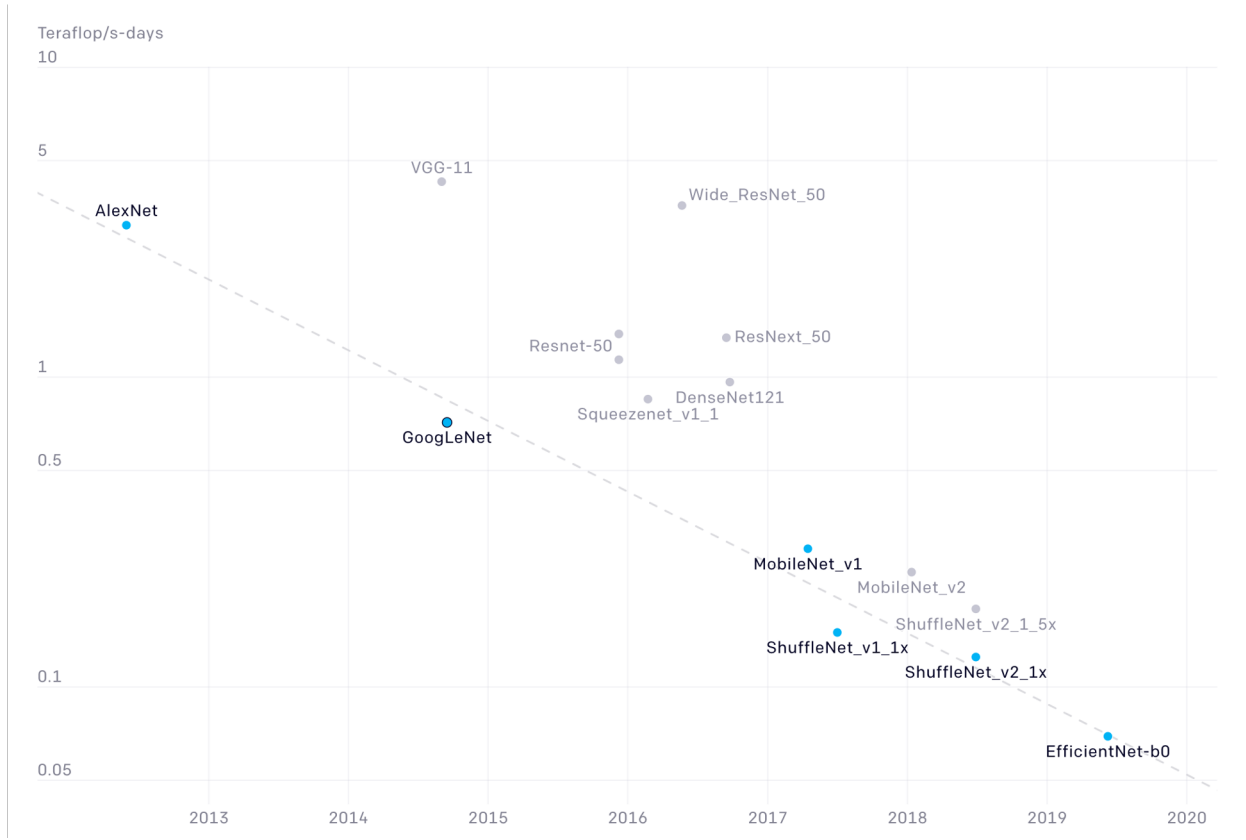
$$\text{resolution: } r = \gamma^\phi$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$



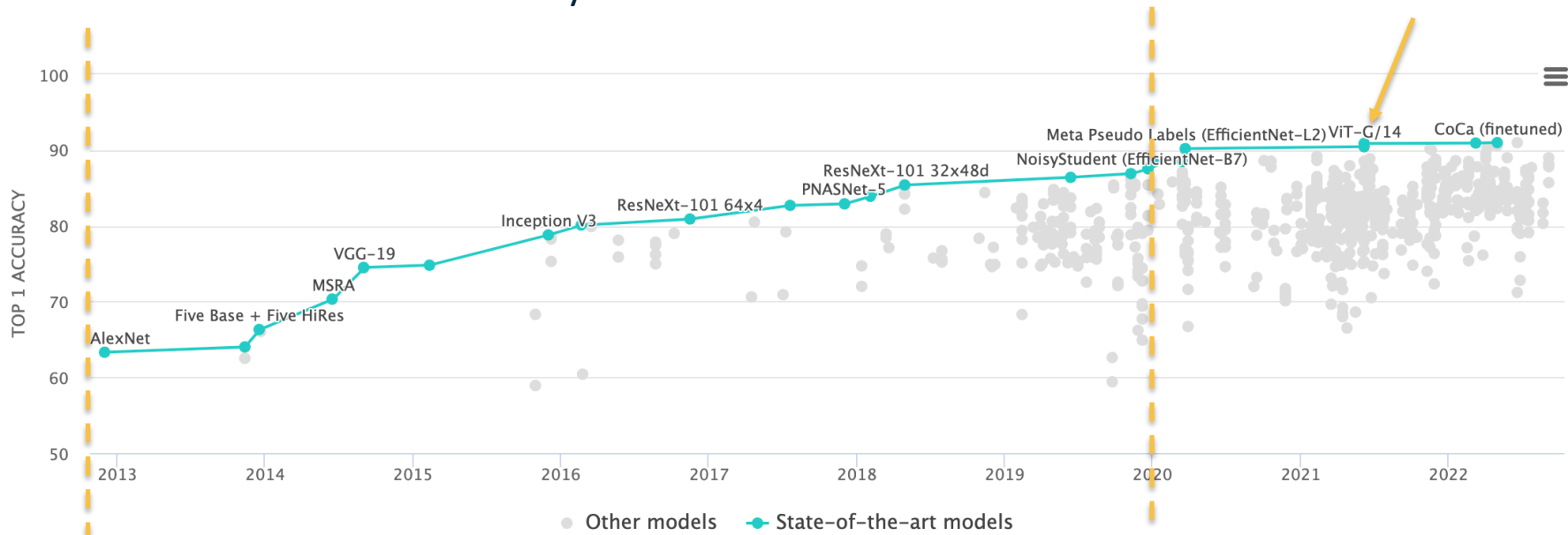
# Efficient networks...



<https://openai.com/blog/ai-and-efficiency/>

# Today's Lecture

# Transformer



<https://paperswithcode.com/sota/image-classification-on-imagenet>



# ConvNets are surviving the age of Transformers!

Plenty of real-world computer vision systems still use ConvNets as their backbone

## A ConvNet for the 2020s

---

Official PyTorch implementation of **ConvNeXt**, from the following paper:

[A ConvNet for the 2020s](#). CVPR 2022.

[Zhuang Liu](#), [Hanzi Mao](#), [Chao-Yuan Wu](#), [Christoph Feichtenhofer](#), [Trevor Darrell](#) and [Saining Xie](#)

Facebook AI Research, UC Berkeley

[ [arXiv](#) ] [ [video](#) ]

---

## 🔗 ConvNeXt V2

### Official PyTorch Implementation

---

This repo contains the PyTorch version of 8 model definitions (*Atto*, *Femto*, *Pico*, *Nano*, *Tiny*, *Base*, *Large*, *Huge*), pre-training/fine-tuning code and pre-trained weights (converted from JAX weights trained on TPU) for our ConvNeXt V2 paper.

**ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders**

[Sanghyun Woo](#), [Shoubhik Debnath](#), [Ronghang Hu](#), [Xinlei Chen](#), [Zhuang Liu](#), [In So Kweon](#) and [Saining Xie](#)

# Next Time: Training Deep NNs (Part 1)

- Activation Functions
- Data Preprocessing
- Weight Initialization
- Batch Normalization
- Transfer learning