# Deep Learning Theory and Practice

Lecture 13

Convolutional Neural Network Architectures

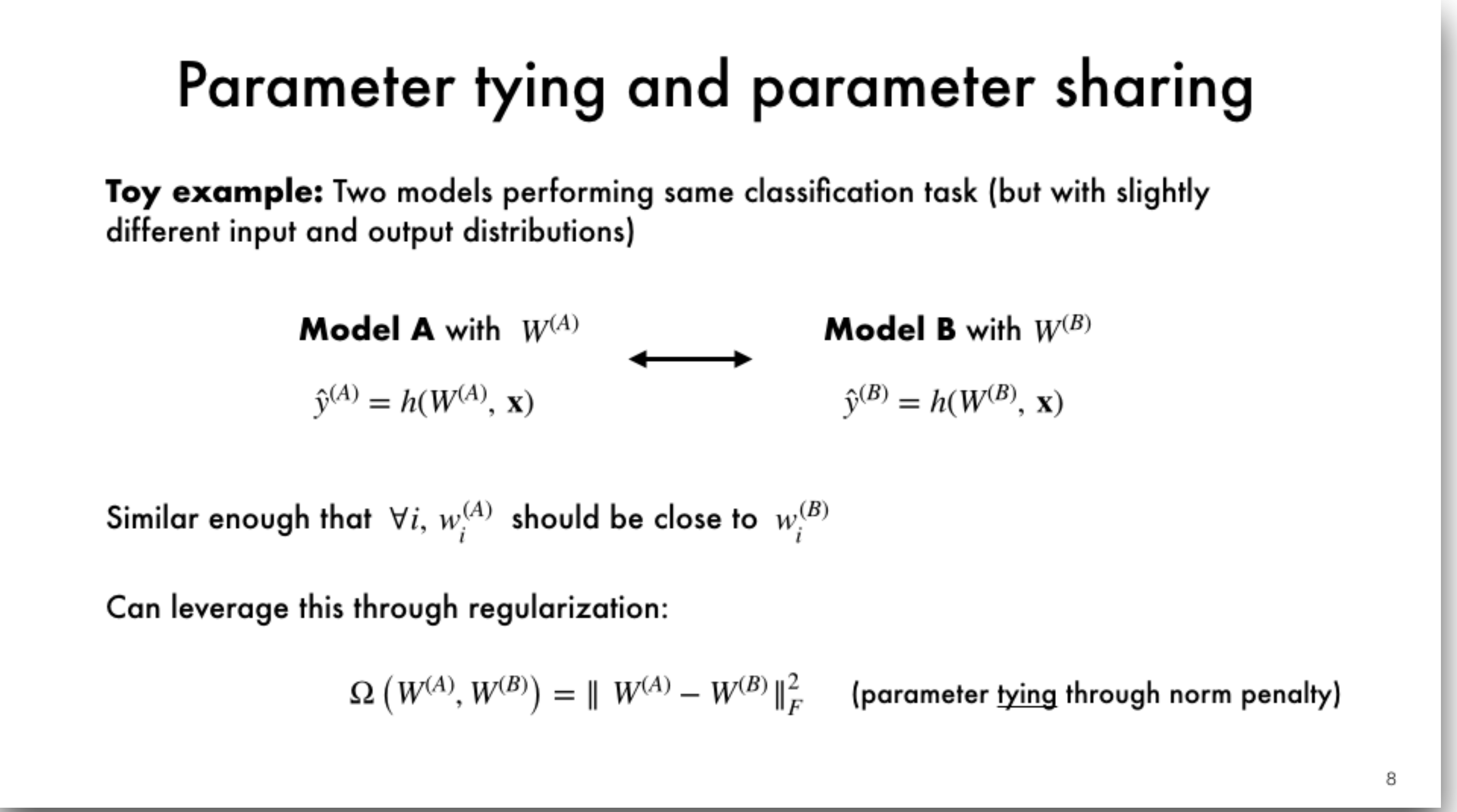
Dr. Ted Willke

willke@pdx.edu

Monday, May 13, 2019

# Review of Lecture 12

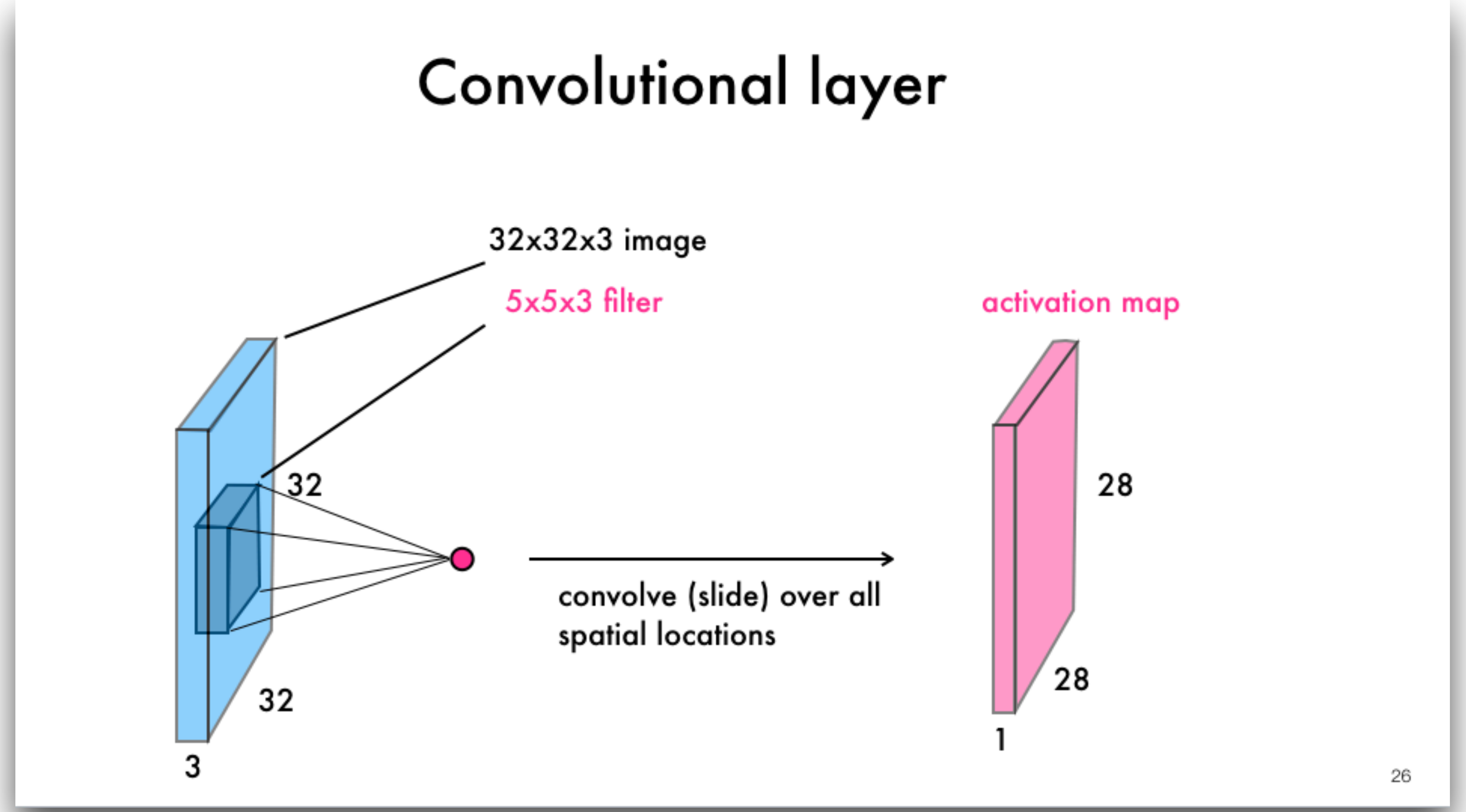
- **Parameter tying and sharing**
  - Take advantage of when you know there are dependencies between parameters




  - Additional benefits if parameters can be shared.

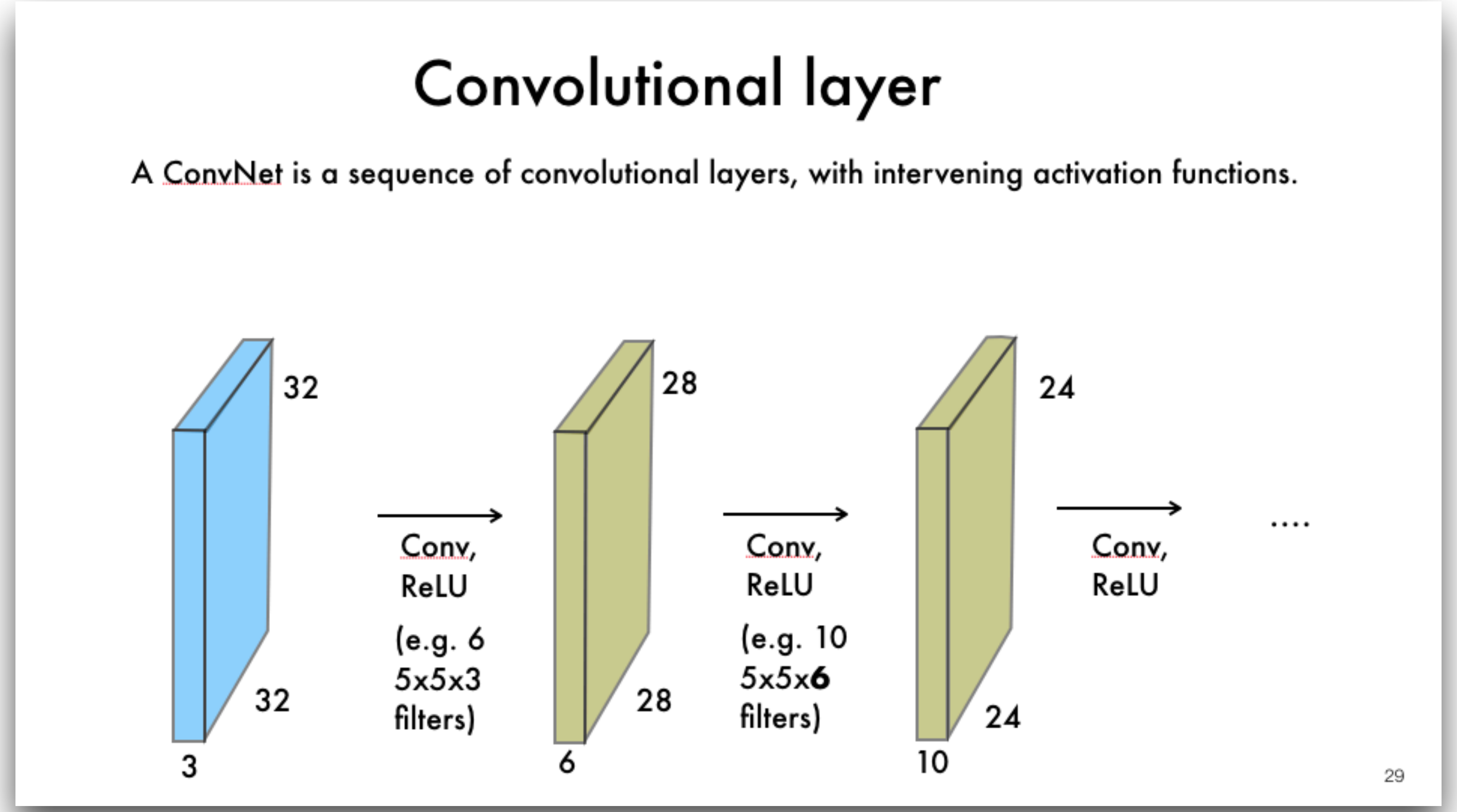
    **CNNs save memory and computation this way.**

- **Convolutional neural networks**
  - Utilizes spatial organization (receptive fields)

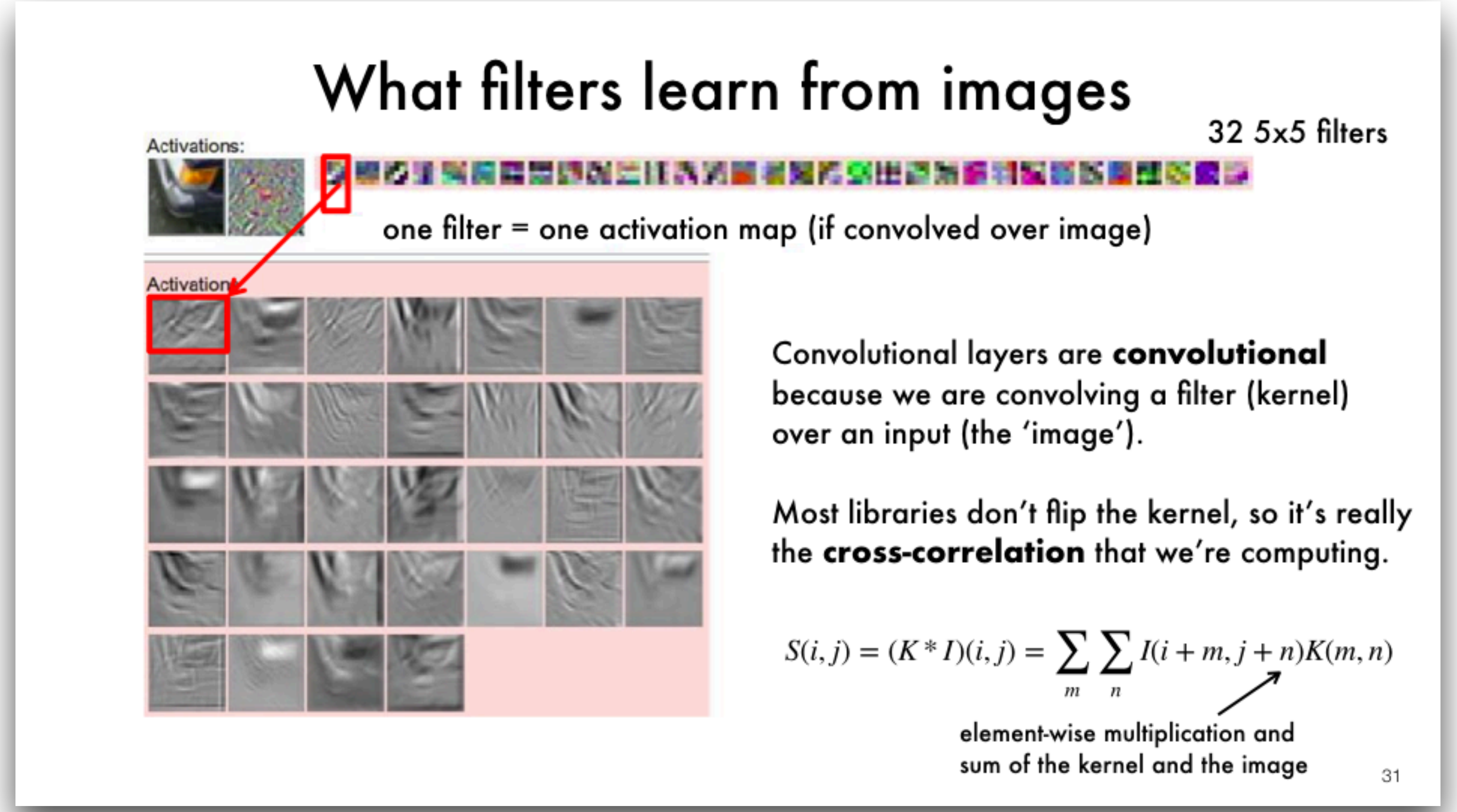


# Review of Lecture 12

- Utilizes hierarchical organization




- **Use any number of filters/activation maps**

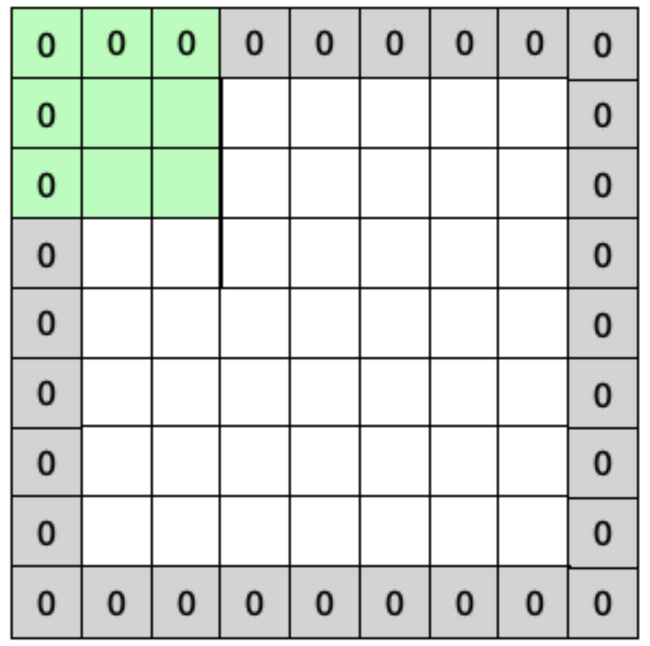


# Review of Lecture 12

- **Zero padding the border to retain dims**

## Zero padding the border

Common in practice to zero pad the border (boundary condition).




For example, input is 7x7, filter is 3x3, applied with **stride 1.**

If we **pad the border** with 1 pixel, what is the output dim?

***7x7 output!***

Common to see Conv layers with stride of 1, filters of size FxF, and zero-padding of `(F - 1) / 2`.

Will preserve size spatially:

F = 3 ⟶ zero pad with 1.
F = 5 ⟶ zero pad with 2.
F = 7 ⟶ zero pad with 3.

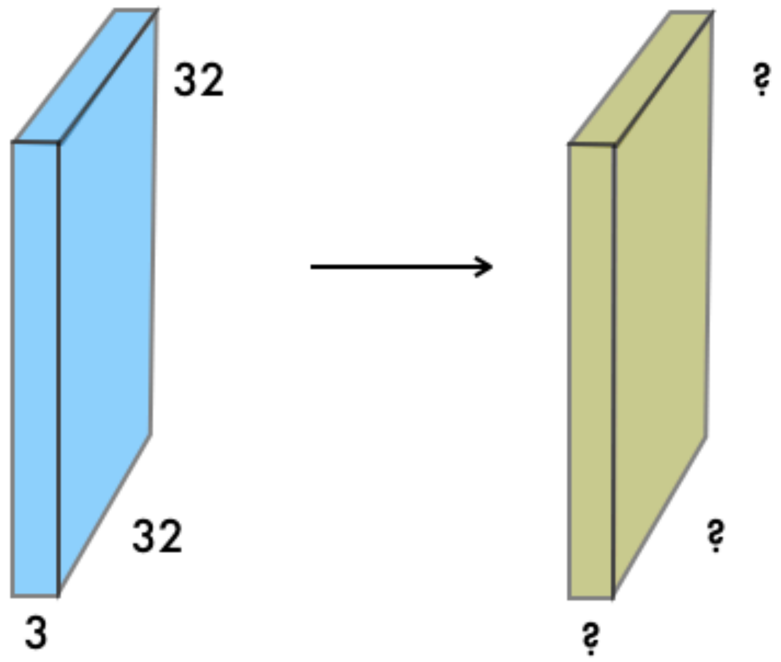
46

- **CNNs reduce number of parameters!**

## Examples

If the input volume is 32x32x3, what is the **number of parameters** in this layer?

10 5x5 filters with stride of 1, pad 2?




**Number of parameters:**

Each filter has `5*5*3 + 1 = 76` parameters
(+1 for the bias)

⟶ 76*10 = 760!

(much fewer than a FC layer with this input size)

49

# Review of Lecture 12 (Summary)

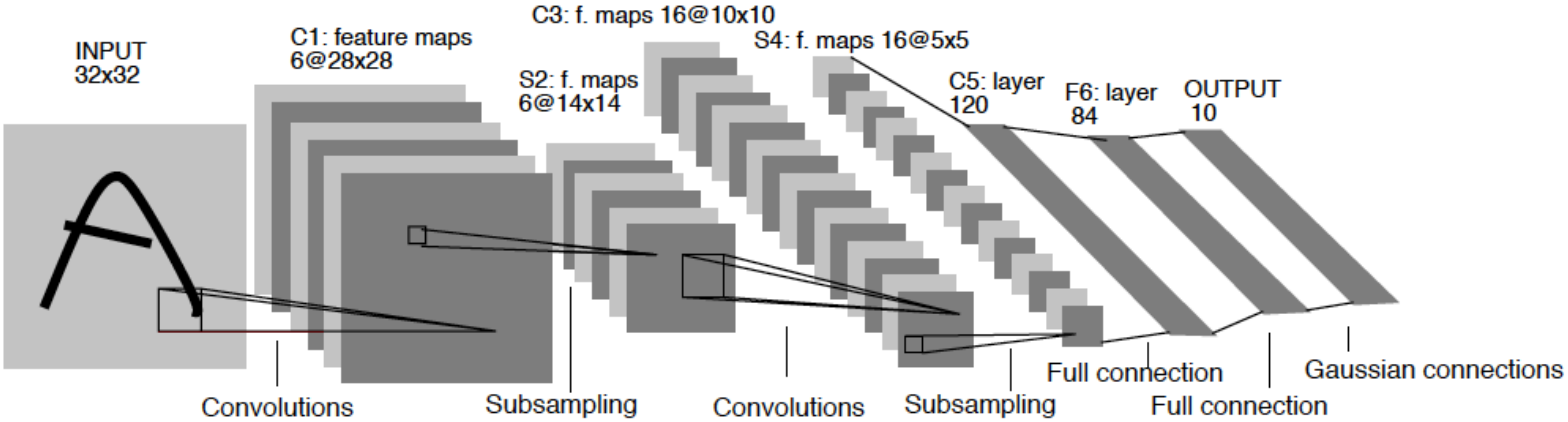
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires 4 hyperparameters:
  - Number of filters $K$,
  - Their spatial extent $F$,
  - The stride, $S$,
  - And the amount of zero-padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where
  - $W_2 = (W_1 - F + 2P) / S + 1$
  - $H_2 = (H_1 - F + 2P) / S + 1$
  - $D_2 = K$
- With parameter sharing, introduces $F*F*D_1$ weights per filters, for a total of $(F*F*D_1)*K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a convolution of the $d$-th filter over the input volume, with a stride of $S$, and offset by the $d$-th bias.

# Today's Lecture

- Convolutional Neural Network Architectures

(Many slides adapted from Stanford's excellent CS231n course. Thank you Fei-Fei Li, Justin Johnson, and Serena Young!)

# LeNet-5




- 5x5 convolutional filters applied at stride of 1
- 2x2 subsampling (pooling) applied at stride of 2
- Architecture is CONV-POOL-CONV-POOL-CONV-FC-FC

# Pooling

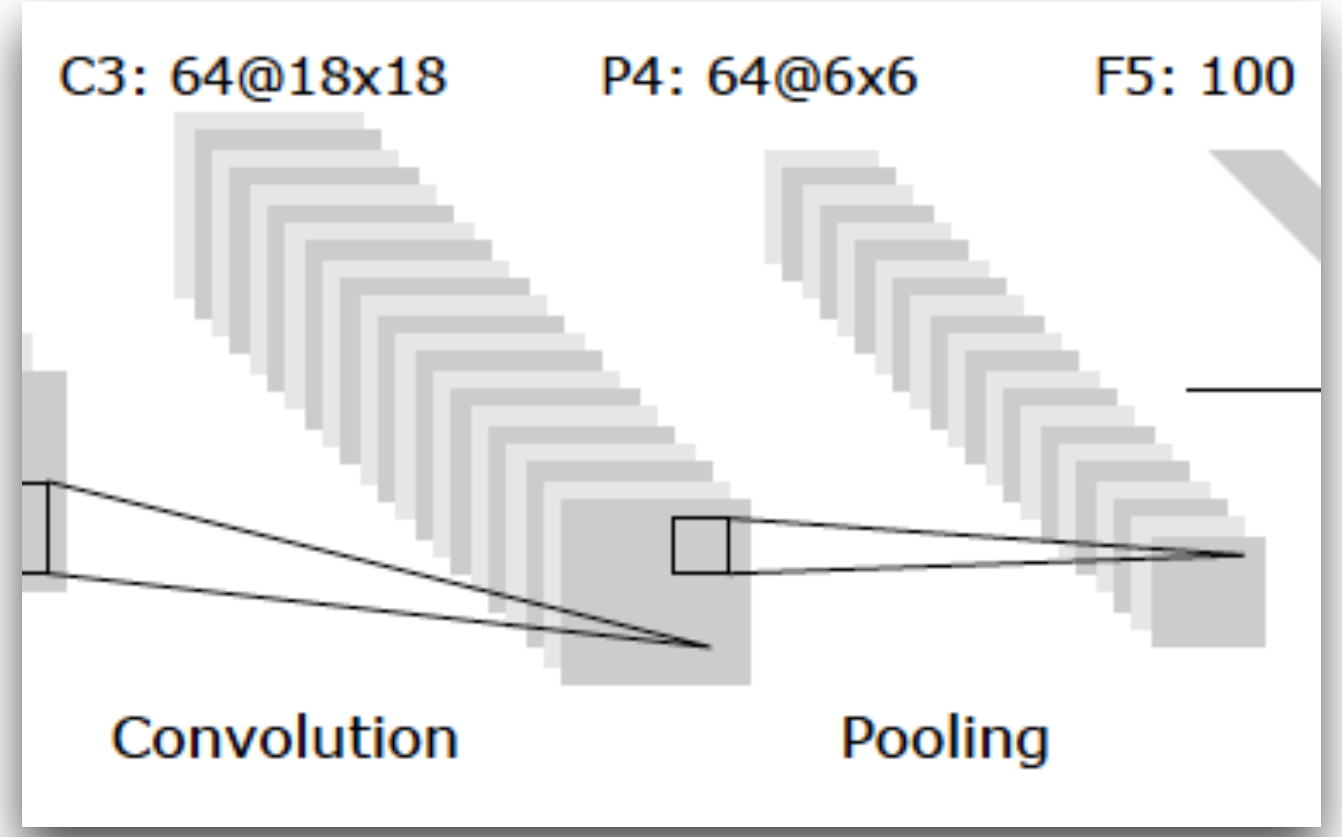
**Objective:** Achieve spatial invariance by reducing the resolution of activation maps.

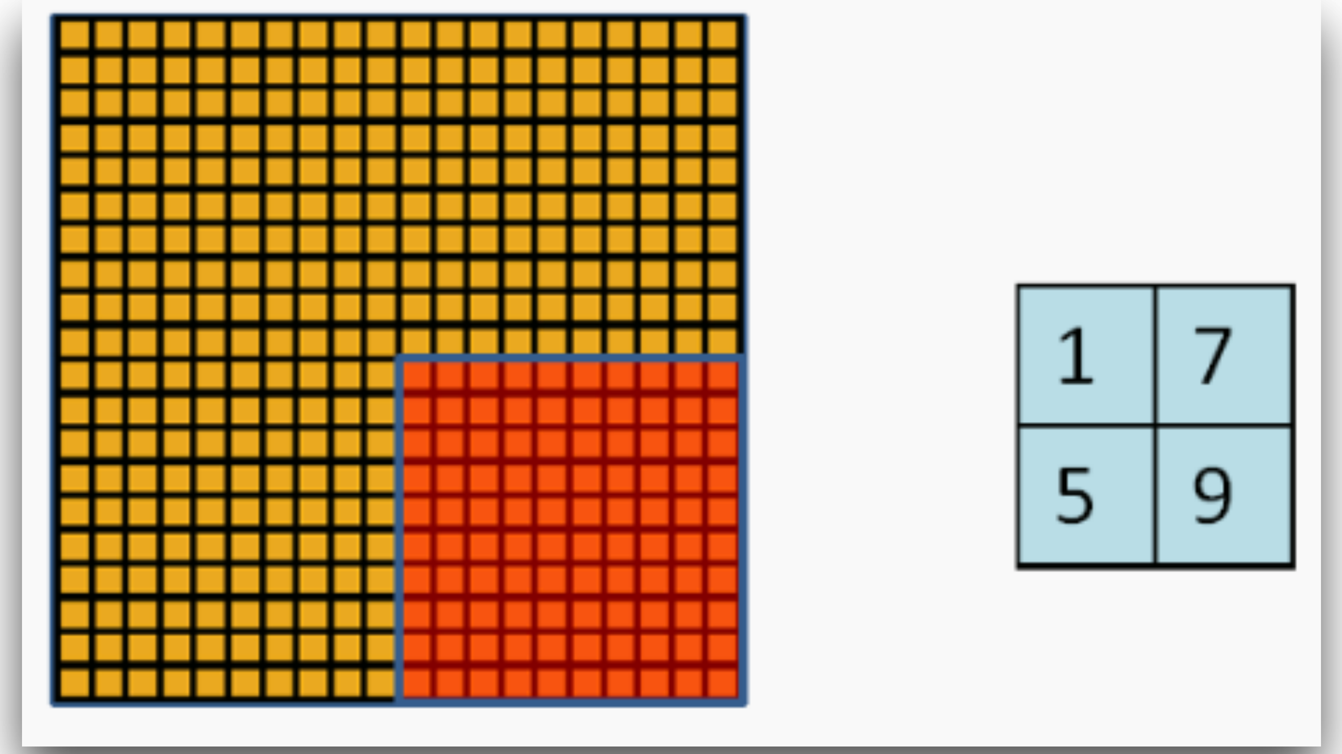
- One pooled activation map per map of the previous layer
- Pooling window of nxn can be of arbitrary size
- Compare subsampling with max pooling:

$$a_j = ReLU\left(\beta \sum_{N\times N} a_i^{n\times n} + b\right) \quad \text{(subsampling)}$$

$$a_j = \max_{N\times N}\left(a_i^{n\times n} u(n,n)\right) \quad \text{(max pooling)}$$

windowing function




Convolved feature

Pooled feature

(Scherer et al. ICANN 2010)

# Pooling

**For backpropagation,**

- Subsampling layer is treated as usual
- At pooling layer, the error signal is only propagated to the position at:

$$a_j = \arg\max_{N \times N} \left( a_i^{n \times n} u(n, n) \right)$$ (results in sparse error maps)

Subsampling is clearly more expensive. But is it superior?

No! It's inferior for selecting invariant features and generalizing.

**Use max pooling without smoothing or overlap!**

(Scherer et al. ICANN 2010)

# AlexNet

Layer architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

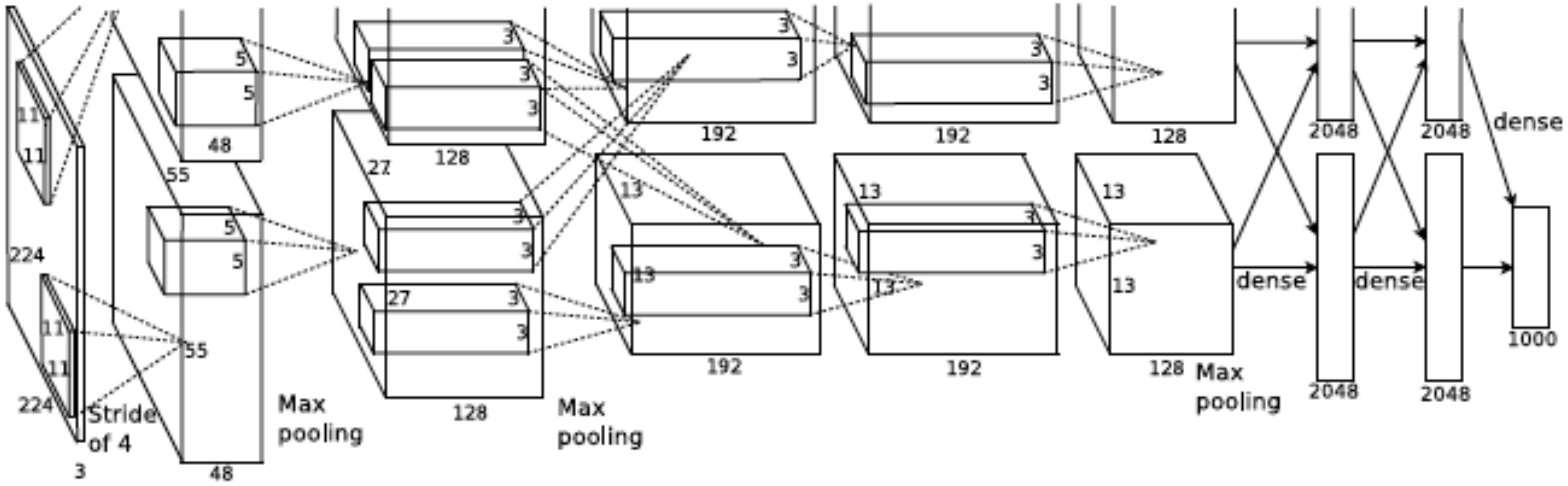
CONV5

MAX POOL3

FC6

FC7

FC8



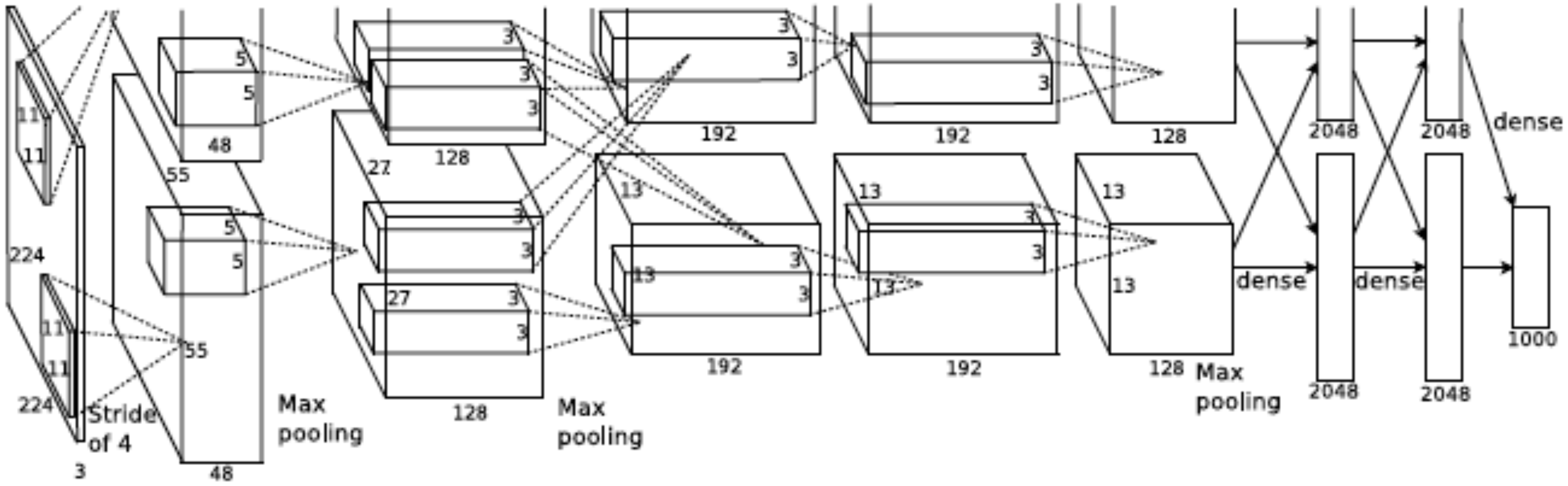

Details:

- 1000-way softmax classifier
- Convolutional layers, max pooling, ReLU
- SGD with weight decay (batch size=128)
- Dropout on fully-connected layers
- Data augmentation

(Krizhevsky et al. 2012)

# AlexNet

Input: 227x227x3 images




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

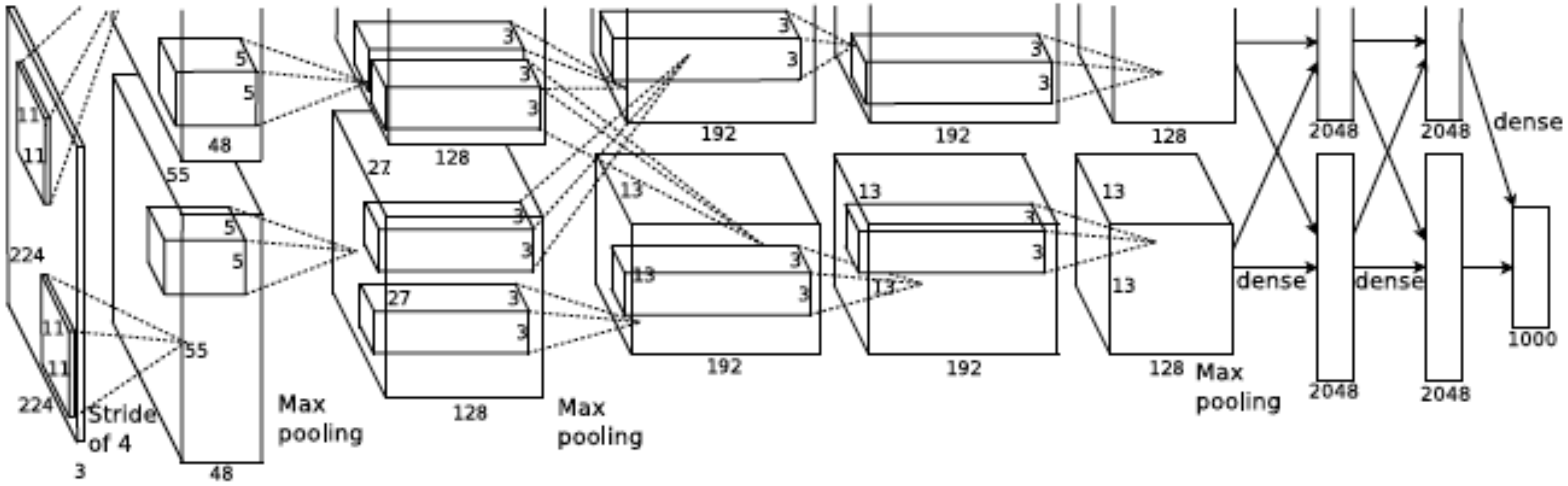
**What is the output volume size?** [Hint: (227-11)/4+1=55]

Output volume: **55x55x96** Number of parameters?

Parameters: (11*11*3)96+96=**35K**

(Krizhevsky et al. 2012)

# AlexNet




Input: 227x227x3 images
After CONV1: 55x55x96

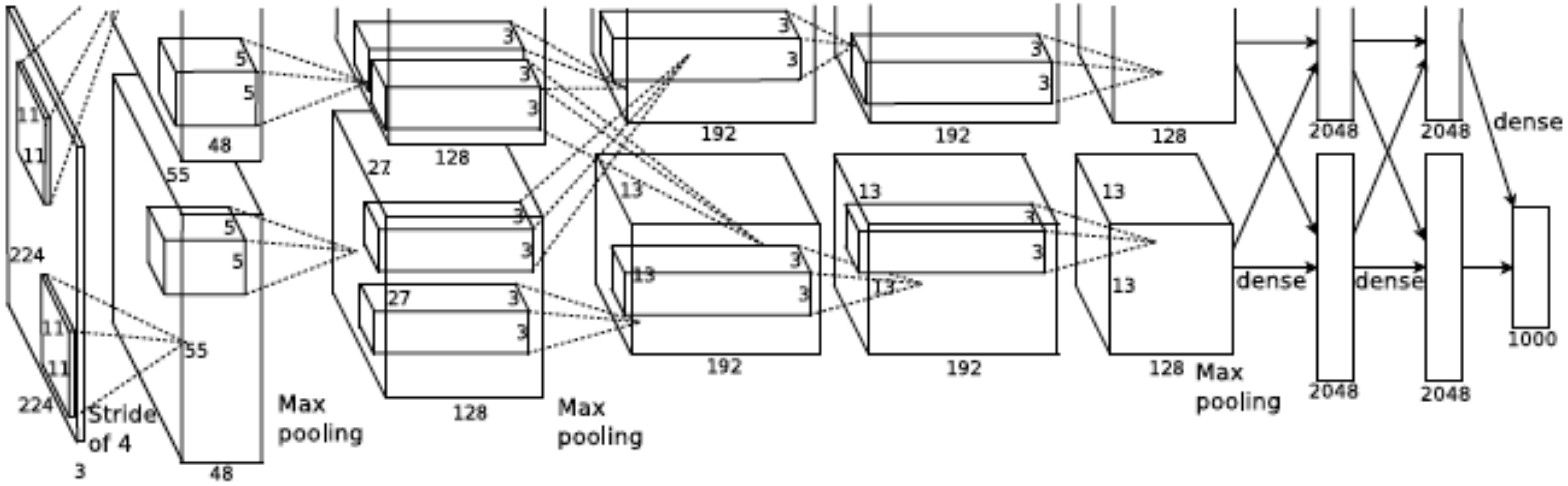
Second layer (POOL1): 3x3 filters applied at stride 2

**What is the output volume size?** [Hint: (55-3)/2+1=27]

Output volume: **27x27x96** Number of parameters?
Parameters: **0!**

(Krizhevsky et al. 2012)

# AlexNet



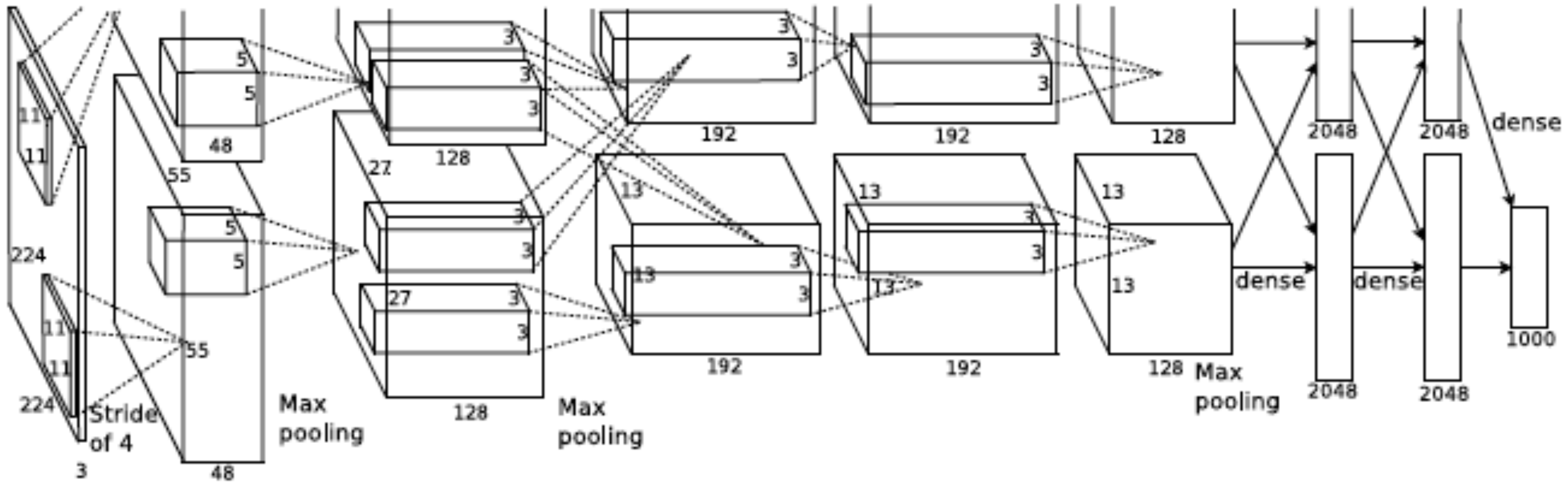

Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

(Krizhevsky et al. 2012)

# AlexNet

**AlexNet Architecture:**

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] 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] POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class outputs)




(Krizhevsky et al. 2012)

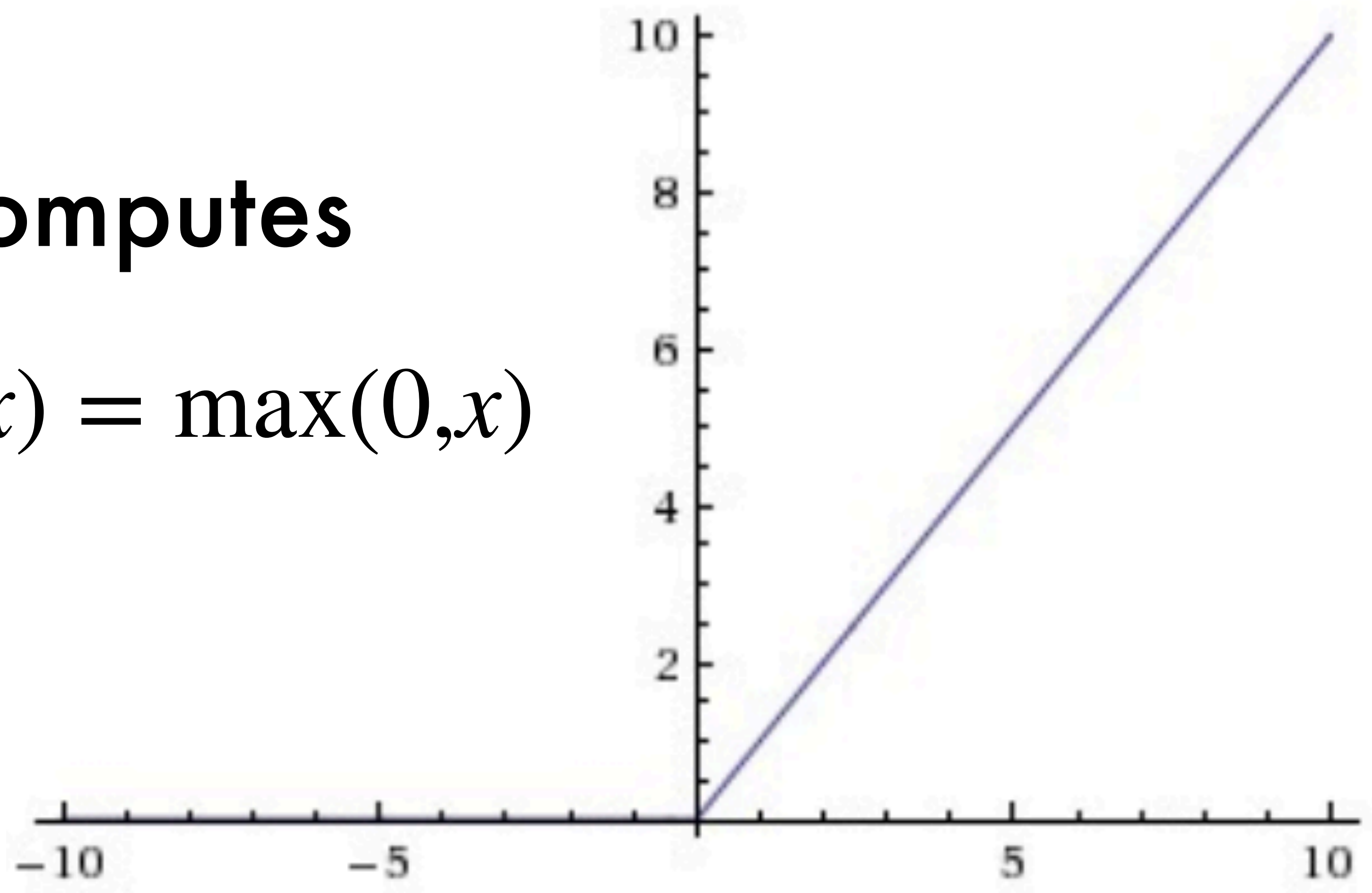
**Details:**

- First use of **ReLU**
- Used Norm layers (across channels - not used anymore)
- Heavy data augmentation
- Dropout 0.5
- Batch size 128
- SGD with **Momentum** 0.9
- LR 1e-2, reduced by 10X manually (val accuracy plateaus)
- $L_2$ weight decay 5e-4
- 7 CNN **ensemble**: 18.2% error down to 15.4%

# ReLU (Rectified Linear Unit)

Computes

$f(x) = \max(0,x)$




Pros:

1. Greatly accelerates convergence of SGD (6X) due to linear non-saturating form
2. Inexpensive to compute (threshold the matrix at zero)
3. More biologically plausible

Cons:

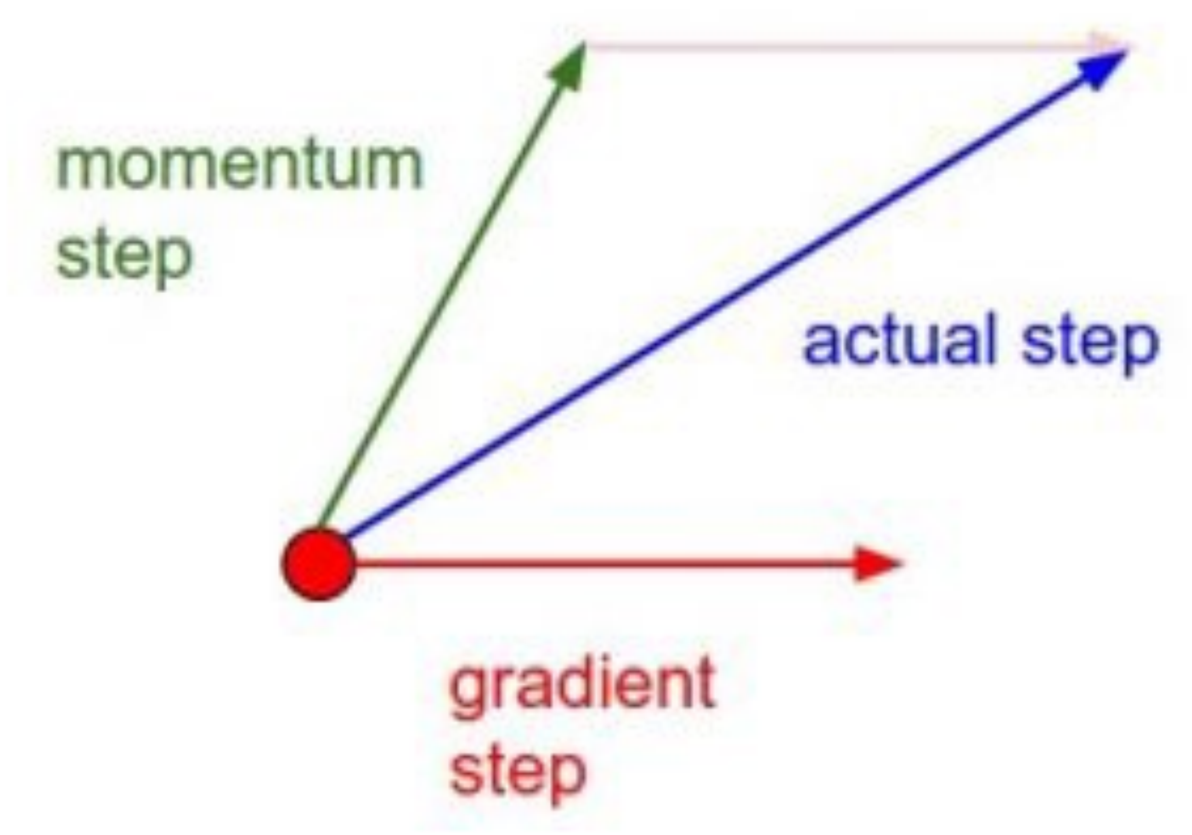
1. Can "die" (update weights such that will never activate again)
2. Non-zero centered

# Momentum

Definition modifies weight update equation:

$$V_t = \beta V_{t-1} \ + \ (1-\beta)\nabla_w L(W, X, y)$$

$$W = W - \gamma V_t$$




More commonly:

$$V_t = \beta V_{t-1} \ + \ \alpha \nabla_w L(W, X, y)$$

Why it works:

- Suppresses noise in the 'right' way
- Dampens oscillations in 'ravines'

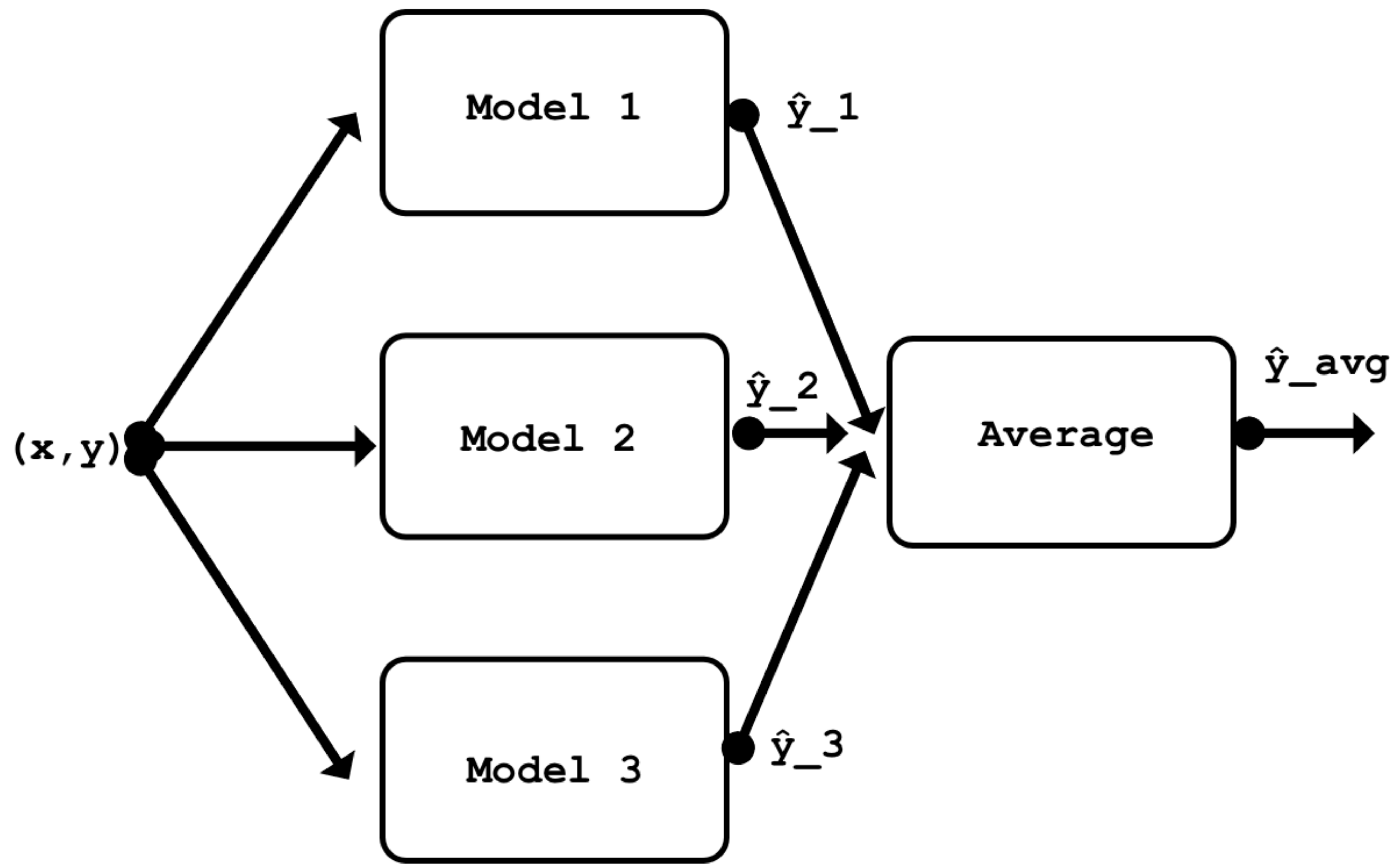
How to use:

- Available with many optimizers (SGD, RMSprop)
- Typical after cross val: $\beta = [0.5,\ 0.9,\ 0.95,\ 0.99]$
- Commonly annealed (start low, end high)

**With momentum, the parameter vector will build up "velocity" in any direction with consistent gradient.**

# Ensembles of Models

**Ensemble of Models:** Train multiple versions of a model, or multiple independent models.




**Approaches may include:**

- Same model, different initializations
- Top models discovered during cross val
- Different epoch checkpoints of same model
- Running average of parameters over last few iterations (smoothed version of weights)

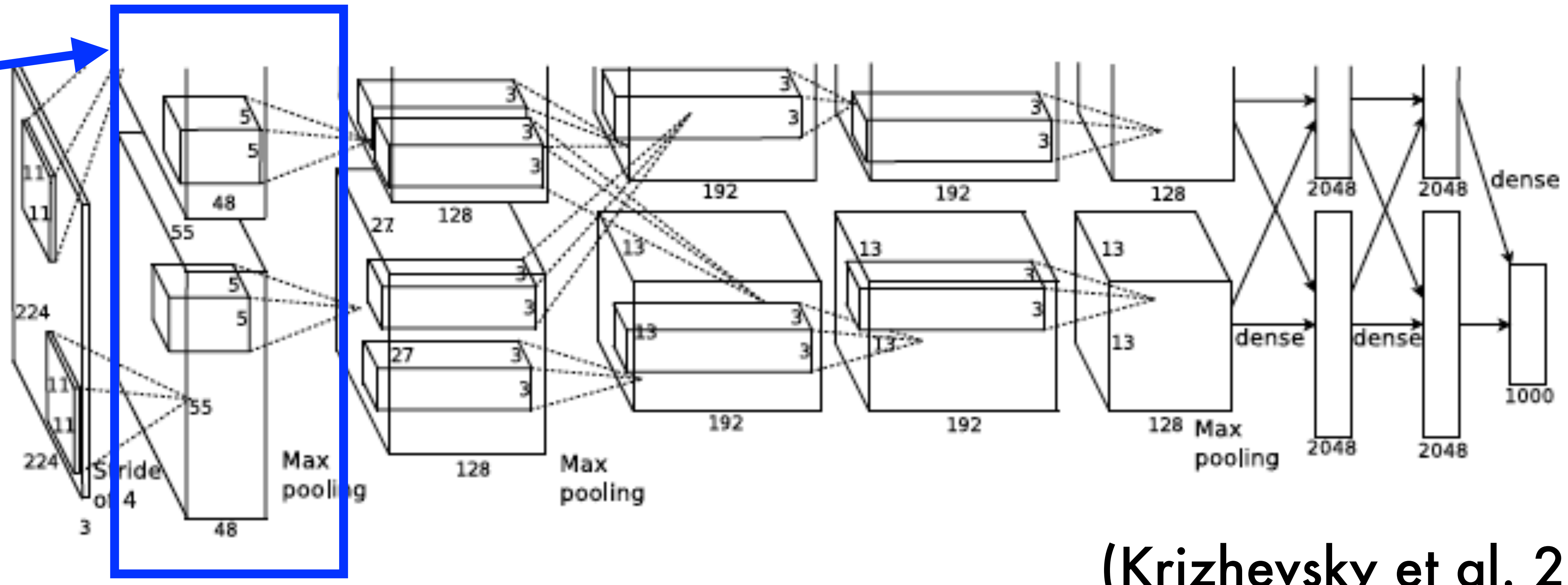
**Disadvantages? Complexity, computational cost**

Image credit: https://towardsdatascience.com/ensembling-convnets-using-keras-237d429157eb

# AlexNet

**AlexNet Architecture:**

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] 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] POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class outputs)




(Krizhevsky et al. 2012)

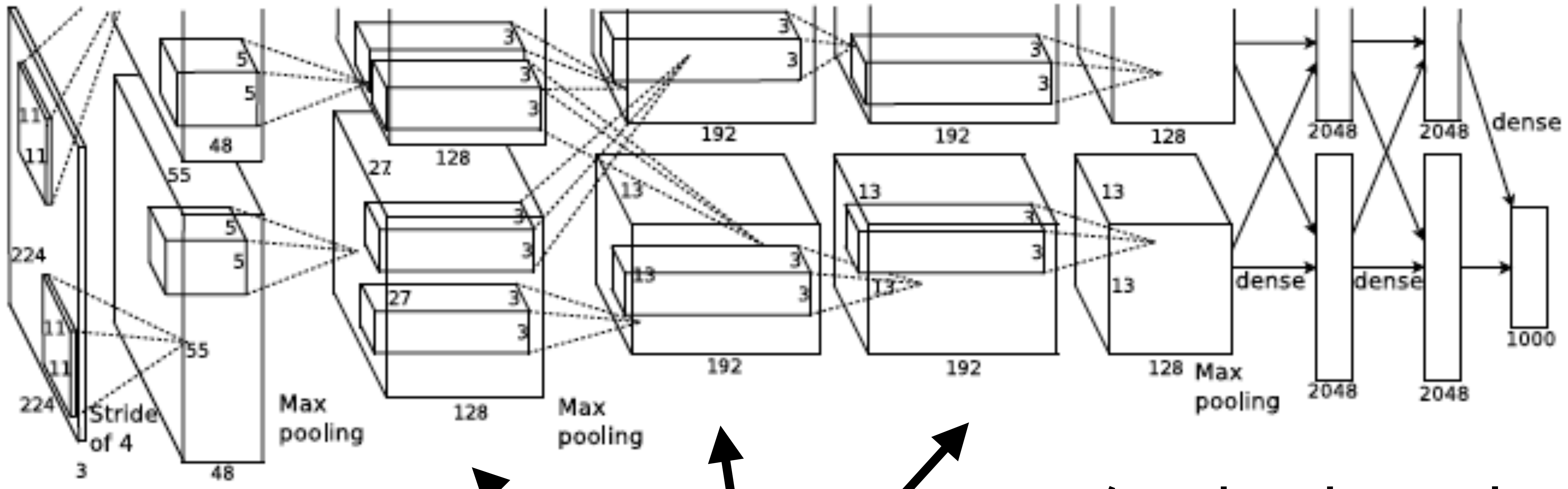
**[55x55x48] x 2!**

- Trained on GTX 580 GPU (3 GB memory)
- Network split across 2 GPUs
- Half the feature maps per GPU

# AlexNet

**AlexNet Architecture:**

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] 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] POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class outputs)




(Krizhevsky et al. 2012)

CONV1, CONV2, CONV4, CONV5 connect only with activation maps on same GPU.

# AlexNet

**AlexNet Architecture:**

[227x227x3] INPUT

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

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

[27x27x96] NORM1: Normalization layer

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

[13x13x256] 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

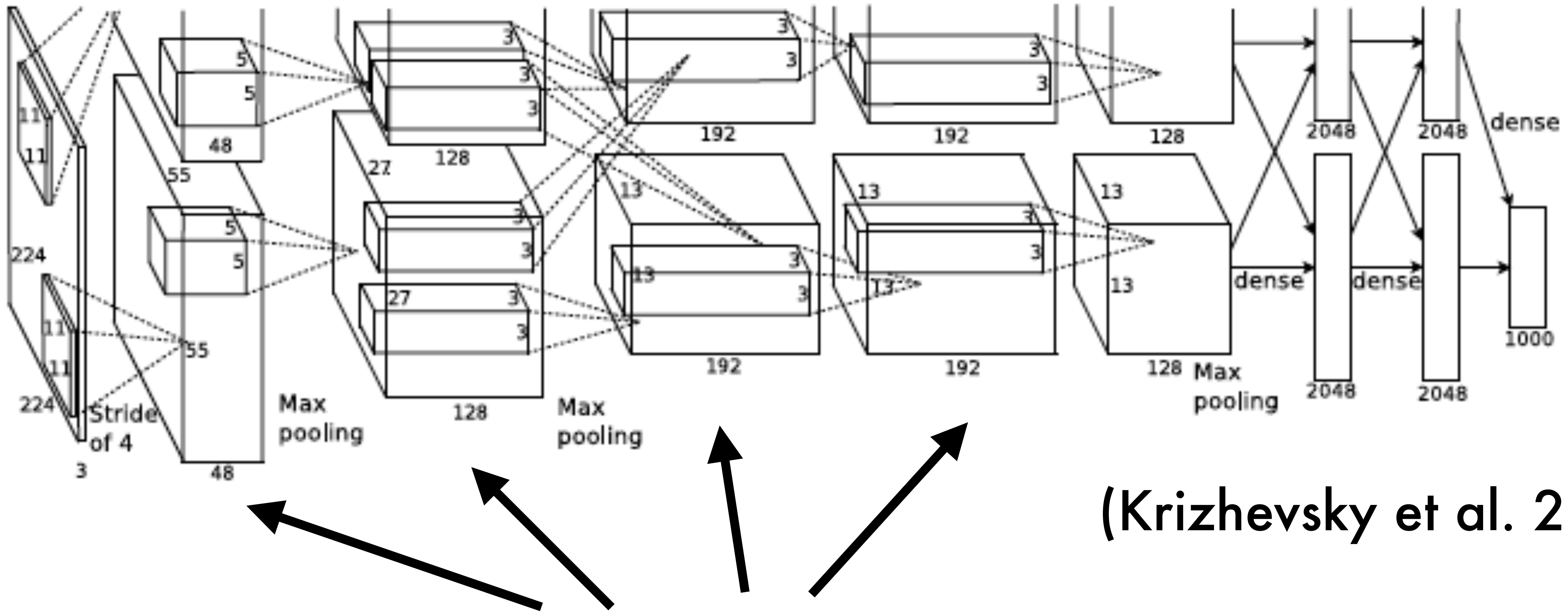
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

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

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class outputs)




(Krizhevsky et al. 2012)

CONV3, FC6, FC7, and FC8 connect with all activation maps in preceding layer (communicate across GPUs).

# AlexNet and ImageNet (ILSVRC contest)

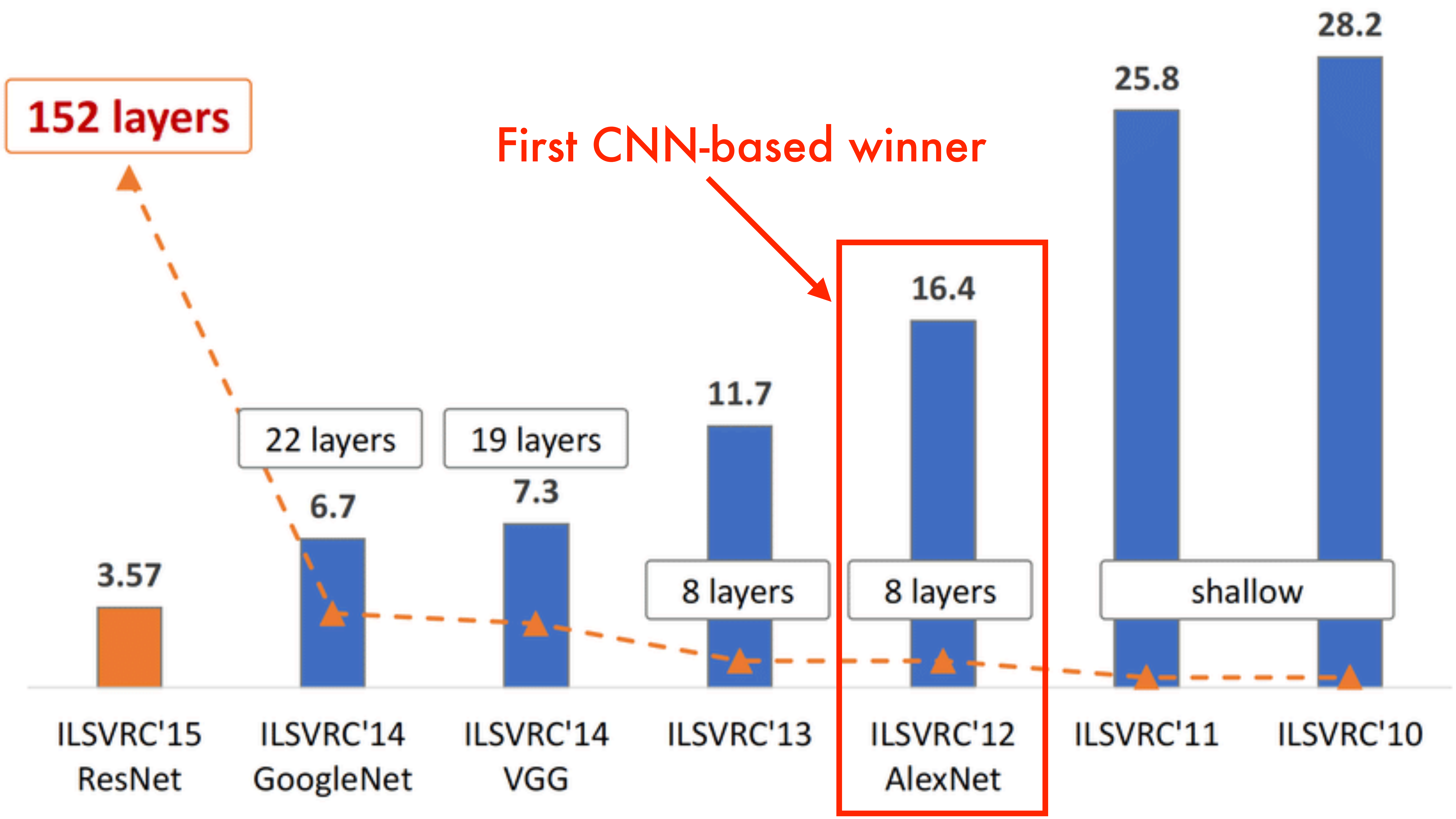



Image credit: Kaiming He

# AlexNet and ImageNet (ILSVRC contest)

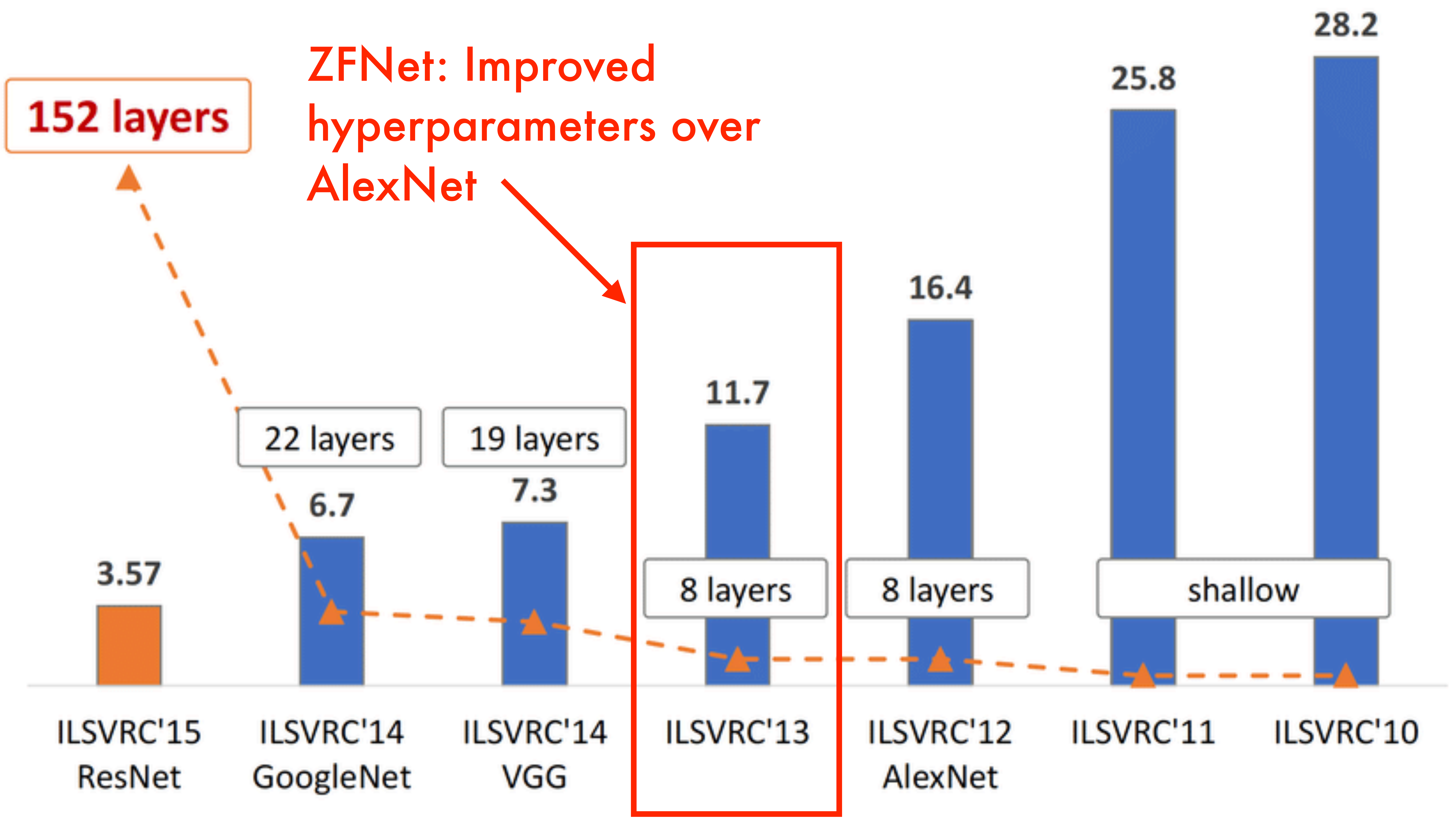
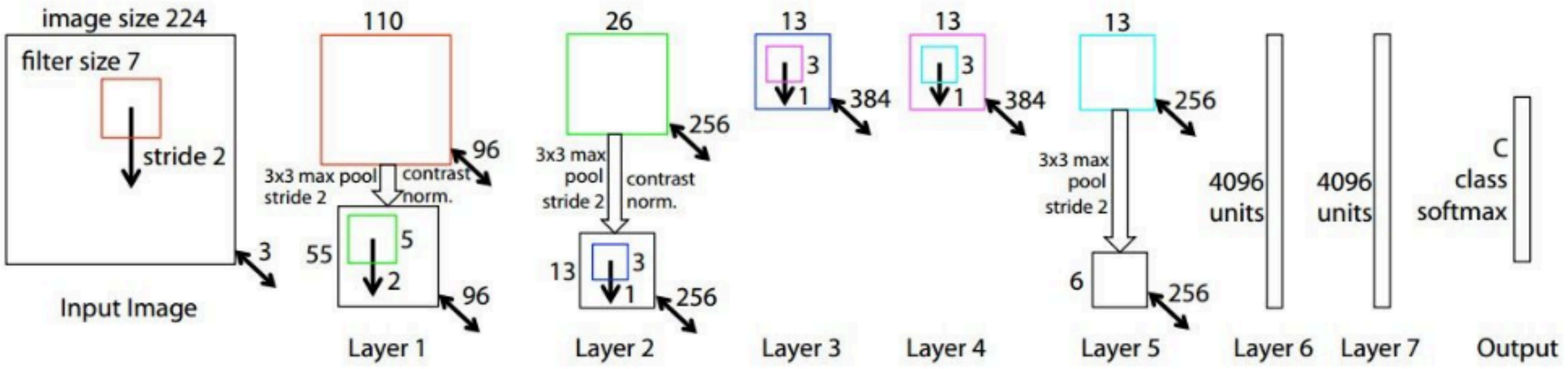



Image credit: Kaiming He

# ZFNet




You get the idea!

Like AlexNet, but:

- CONV1 changed from 11x11 with stride 4 to 7x7 with stride 2
- CONV3,4,5 changed from 384, 384, and 256 filters to 512, 1024, and 512 filters

**ImageNet top-5 error improved from 16.4% to 11.7%**

(Zeiler and Fergus 2013)

# AlexNet and ImageNet (ILSVRC contest)

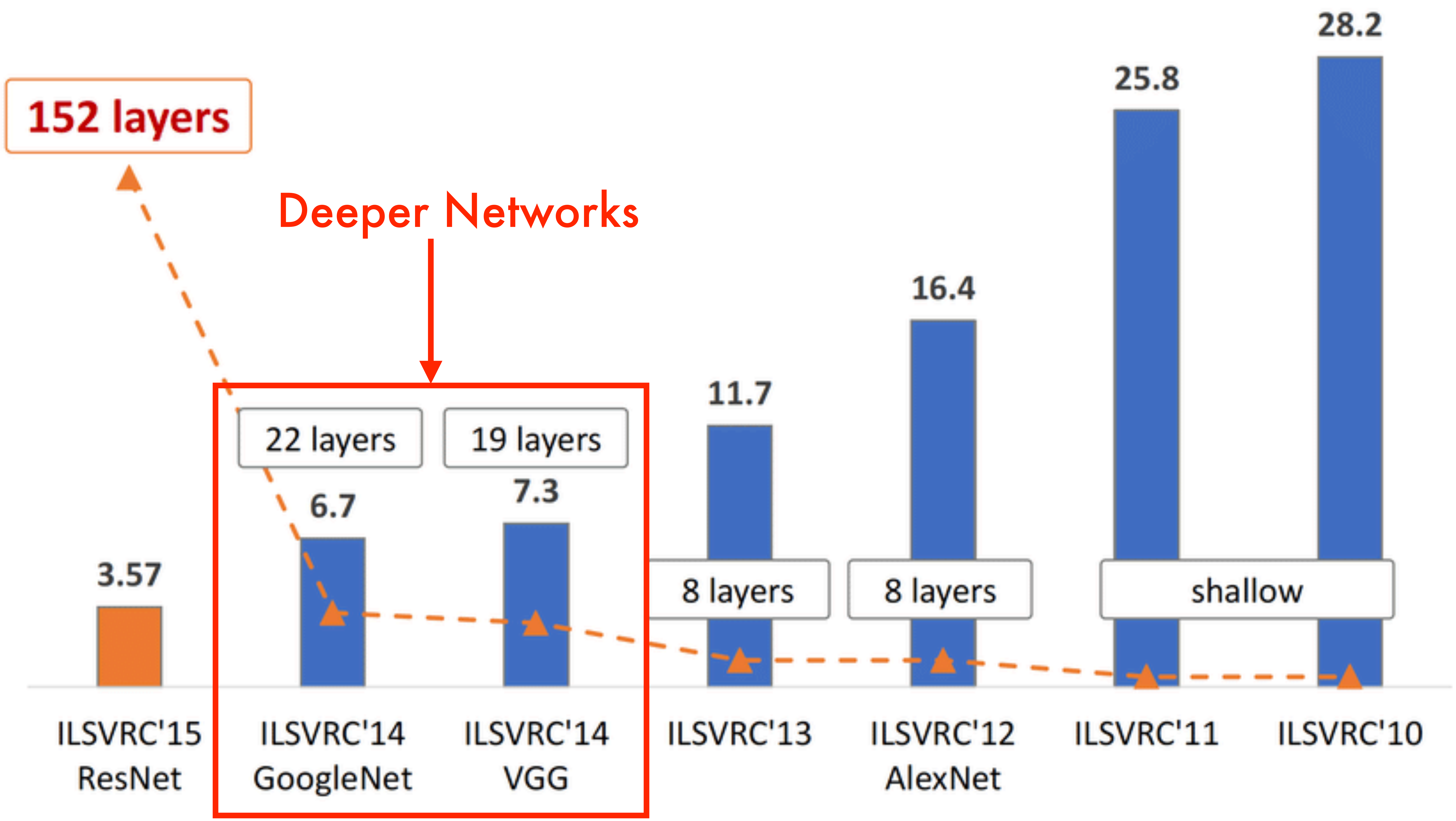



Image credit: Kaiming He

# VGGNet

**Smaller filters, deeper networks**

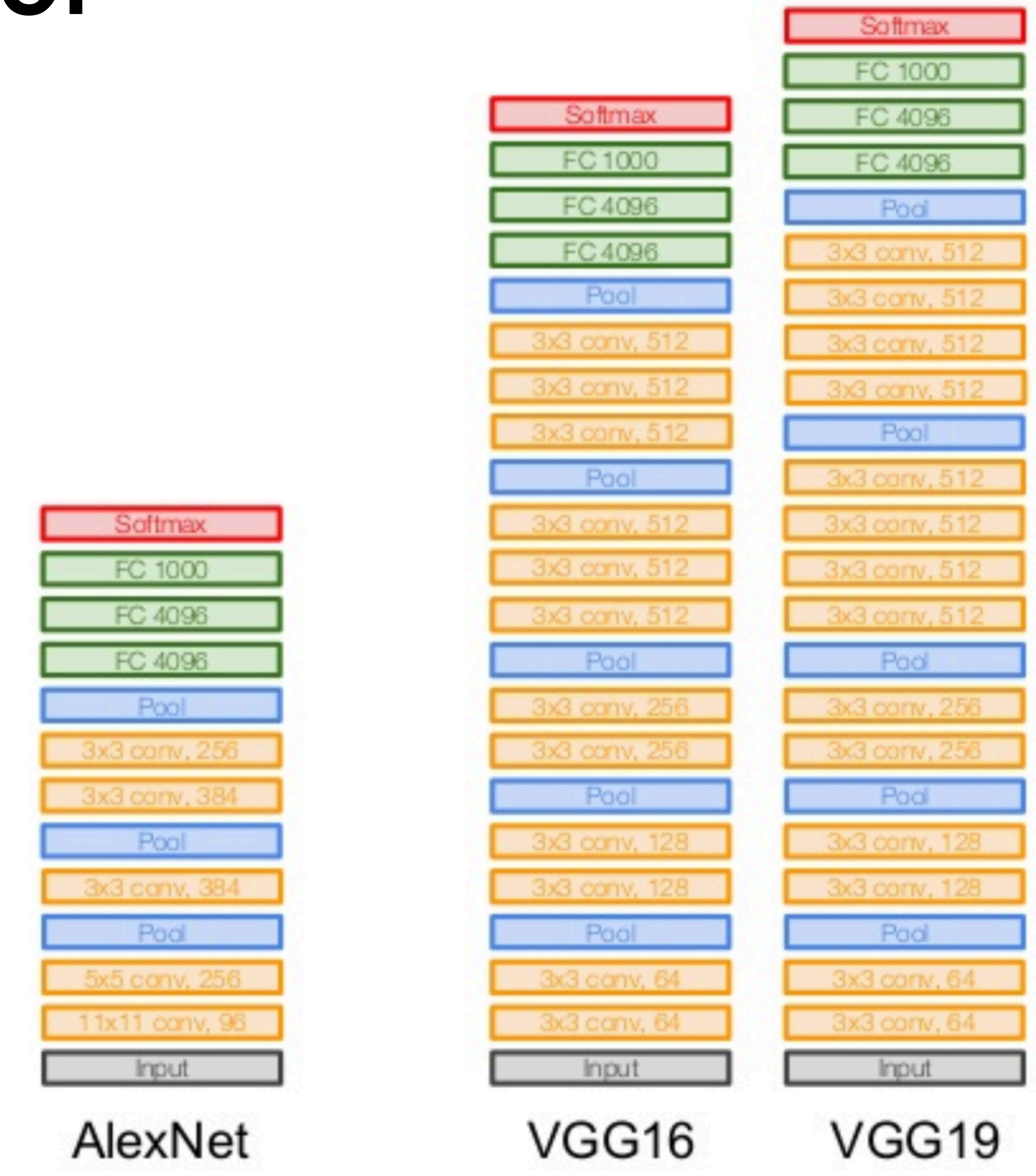
8 layers (AlexNet)

16-19 layers (VGG16Net)

Smallest that looks at neighbors!

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

**ImageNet top-5 error improved from 11.7% to 7.3%**




(Simonyan and Zisserman 2014)

# VGGNet

**Smaller filters, deeper networks**

Why use smaller filters (e.g., 3x3 conv)?

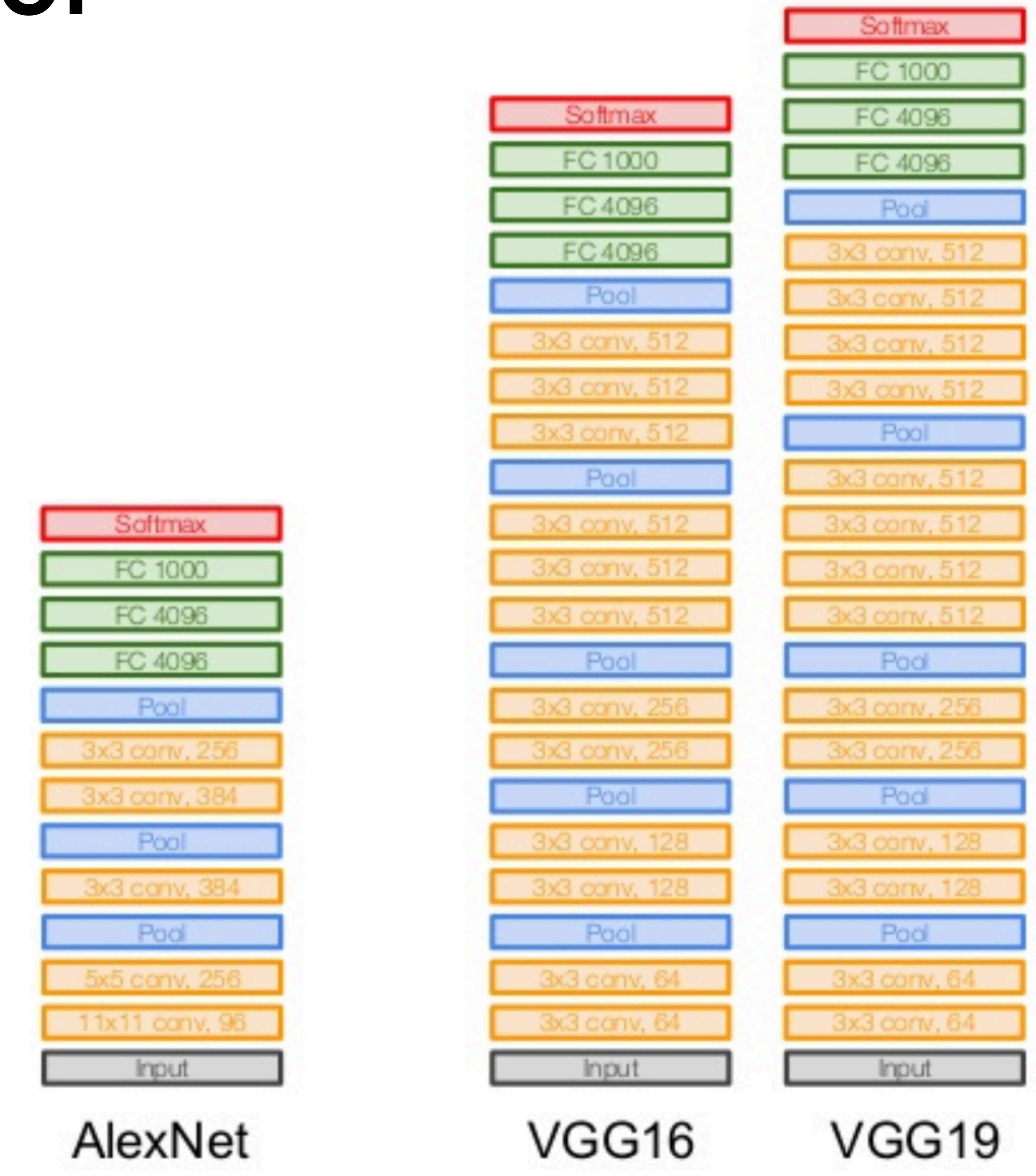
**Related:** What is the receptive field of three 3x3 conv layers (stride 1)?

Same **effective receptive field** as one 7x7 conv layer…

… but deeper (more non-linearities) …

… and fewer parameters!

$3 * (3^2 C^2)$ versus $7^2 C^2$ for $C$ channels per layer.




(Simonyan and Zisserman 2014)

# VGG16 memory usage and parameters

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Most memory is used in early conv layers

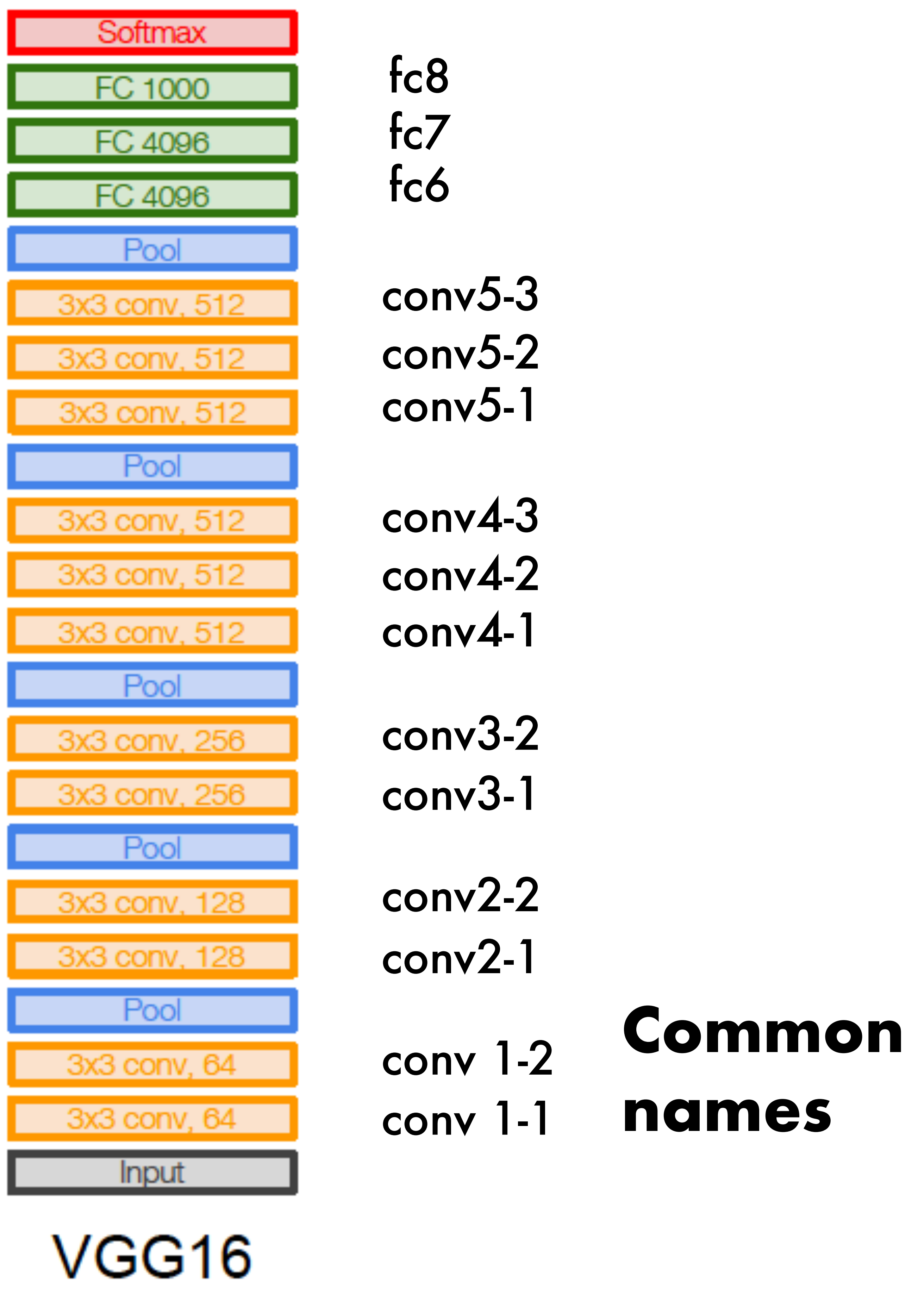
Most parameters used in FC layers

Total memory: 24M*4 bytes or approx 96 MB per image (forward pass; 2X to include backward!)
Total parameters: 138M!!

# VGG16 memory usage and parameters

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

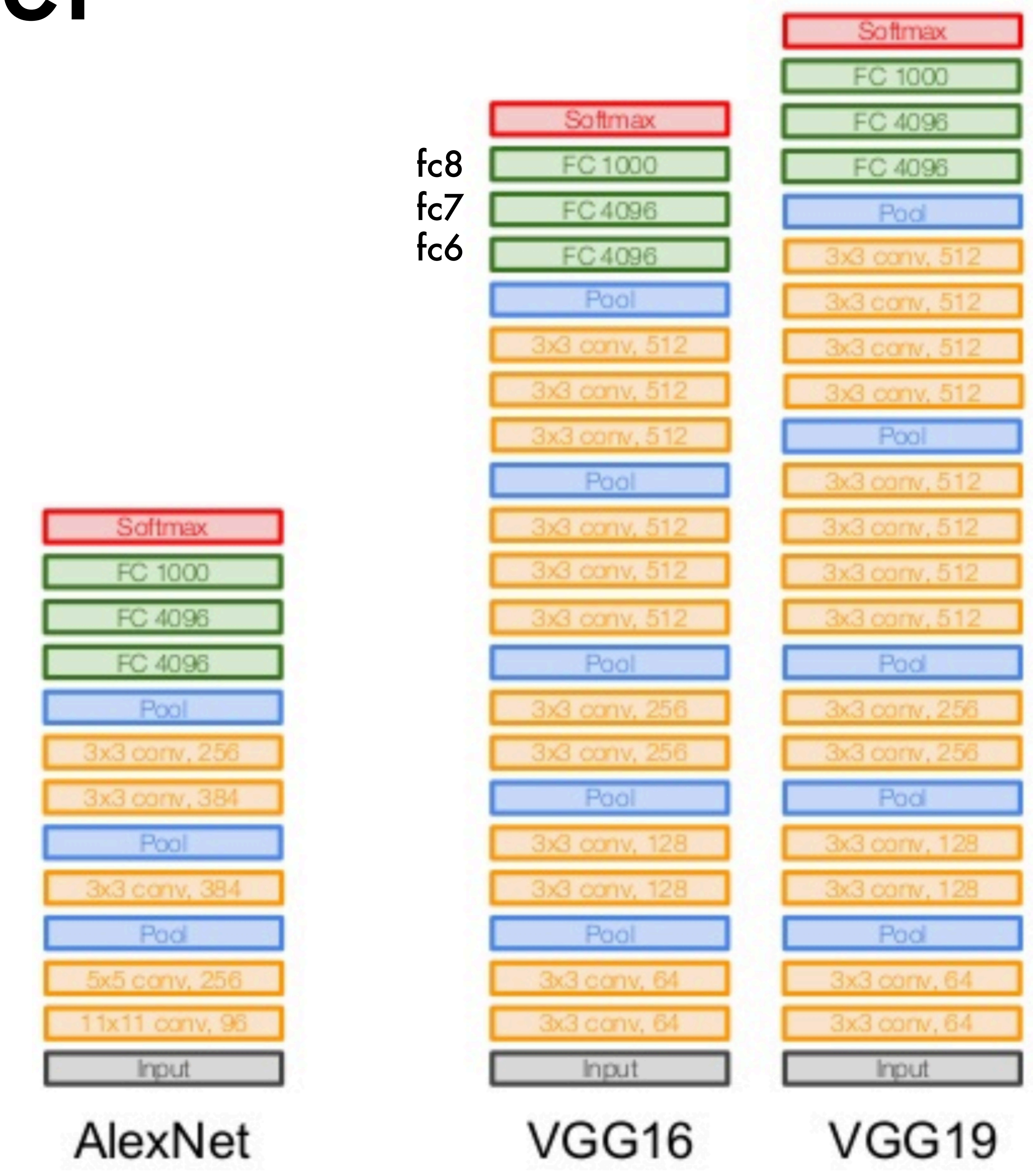



Total memory: 24M*4 bytes or approx 96 MB per image (forward pass; 2X to include backward!)
Total parameters: 138M!!

# VGGNet

**Additional notes:**

- 2nd in classification, 1st in localization in ILSVRC'14
- Similar training as AlexNet
- Didn't use the NORM
- VGG19 slightly better (more memory)
- Use ensembles for best results
- fc7 features generalize well to other tasks

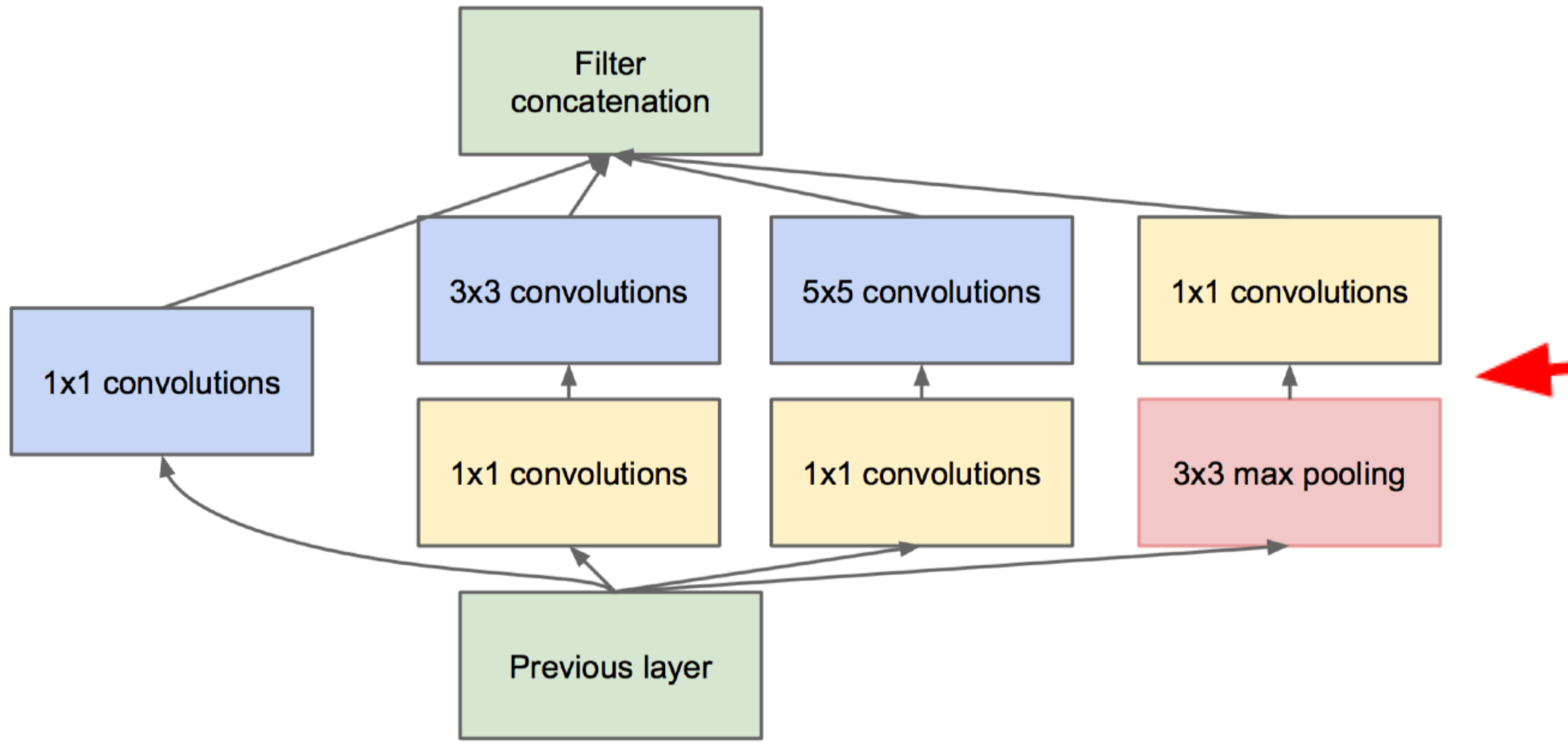



(Simonyan and Zisserman 2014)

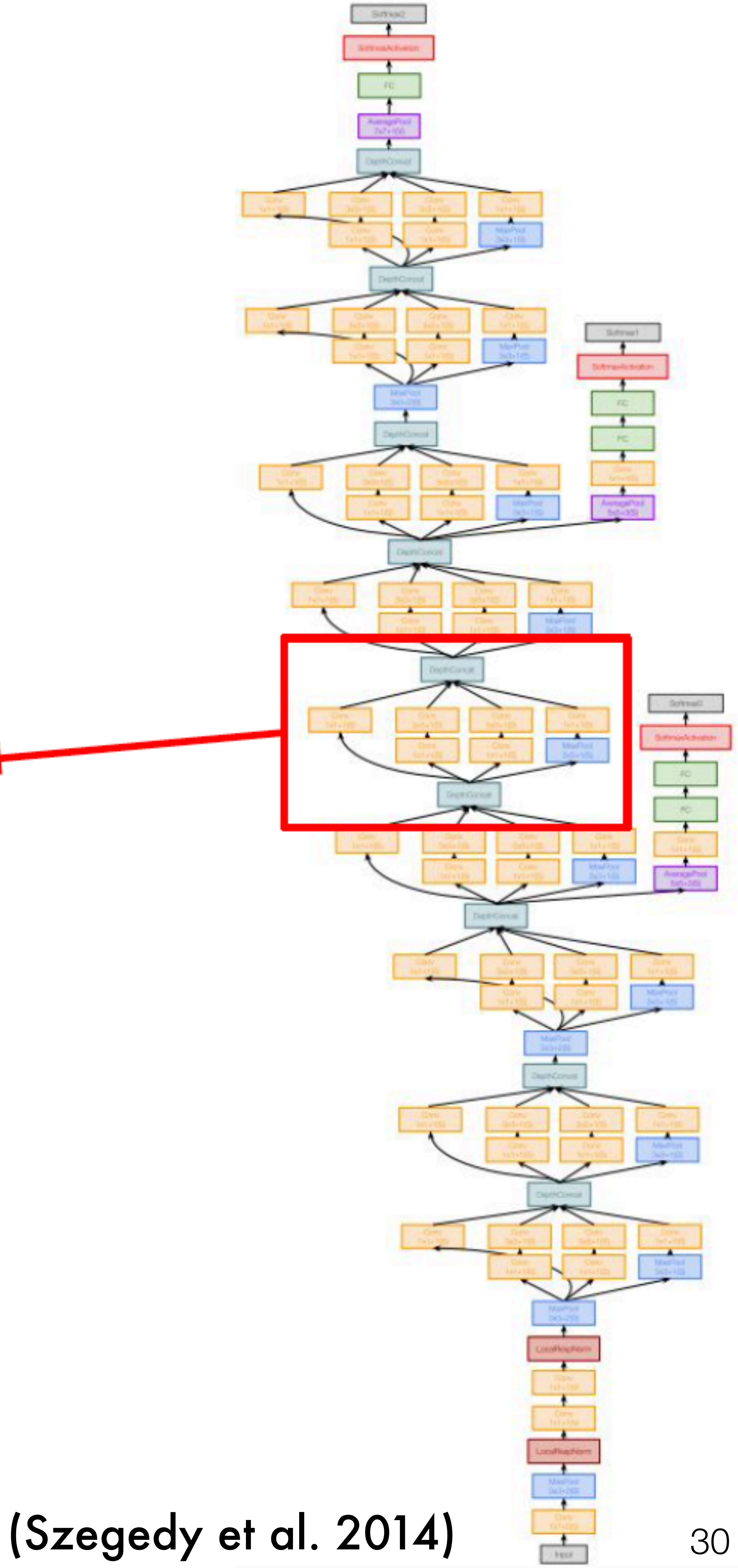
# GoogLeNet

**Deeper networks,**
**with computational efficiency**

- 22 layers
- Efficient 'inception' module
- No FC layers!
- Only 5 million parameters! (1/12th of AlexNet)
- ILSVRC'14 classification winner (6.7% top-5 error)




Inception module

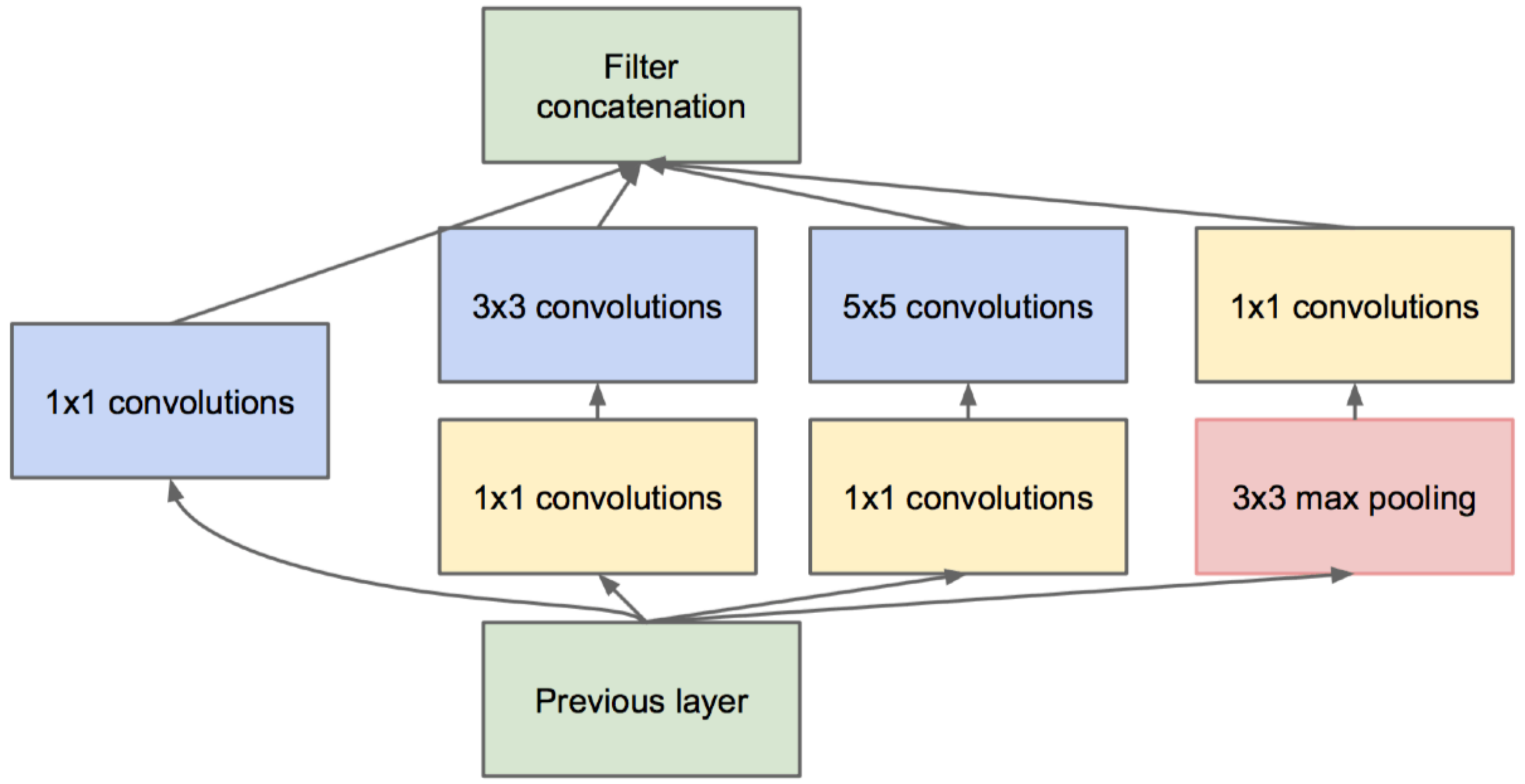


(Szegedy et al. 2014)

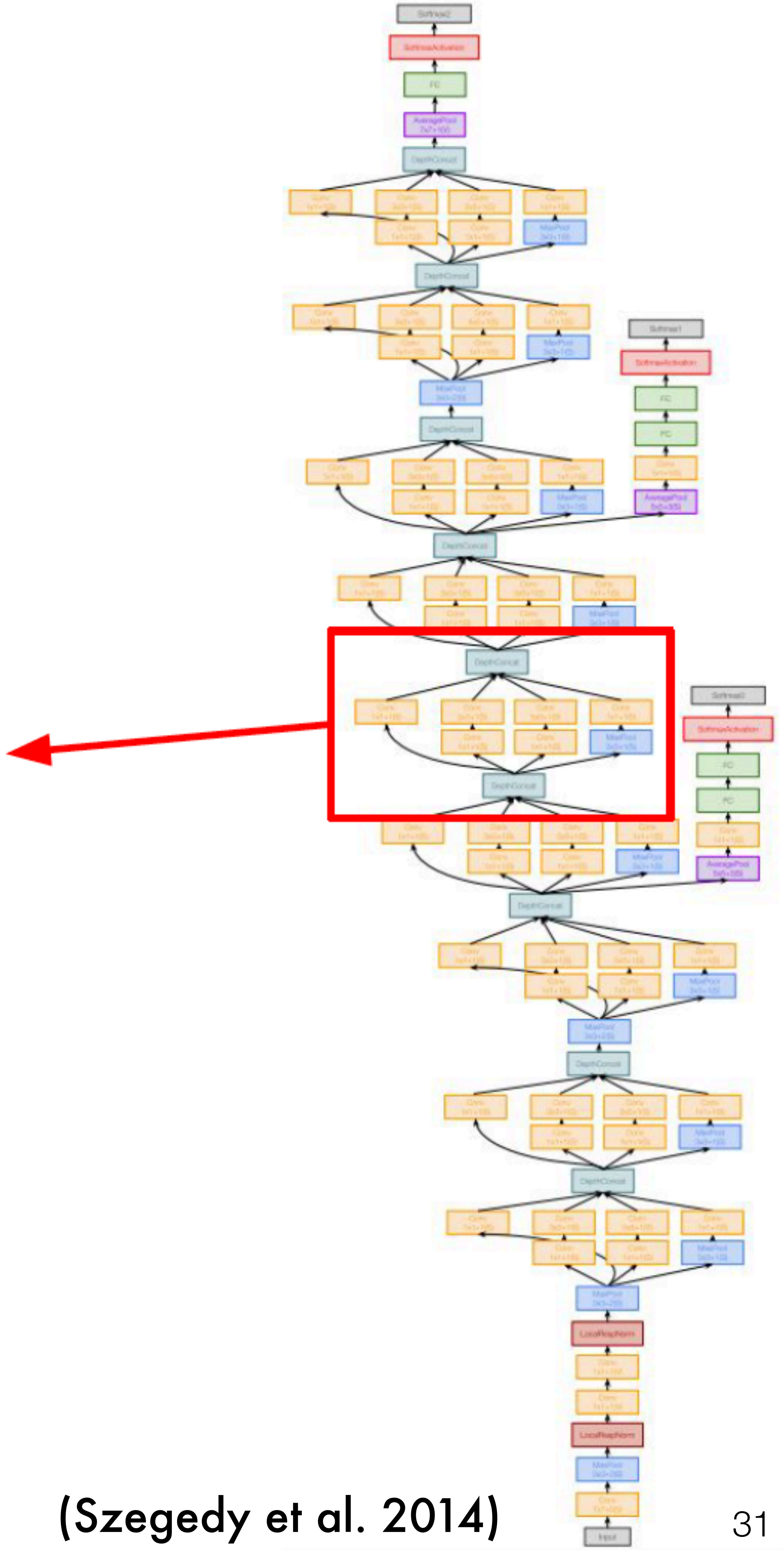
# GoogLeNet

**Deeper networks,**
**with computational efficiency**

**‘Inception model’**
Design a good local network topology (network within a network) and then stack these modules on top of each other.

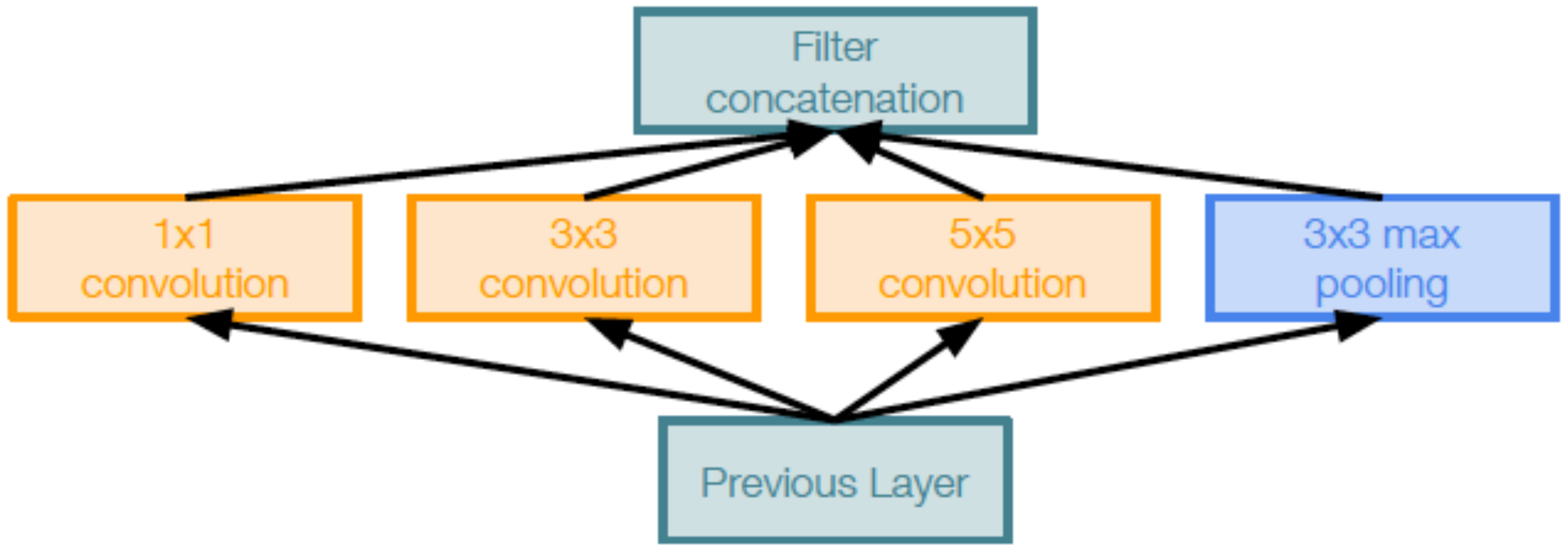



Inception module



(Szegedy et al. 2014)

# GoogLeNet




Naive Inception module

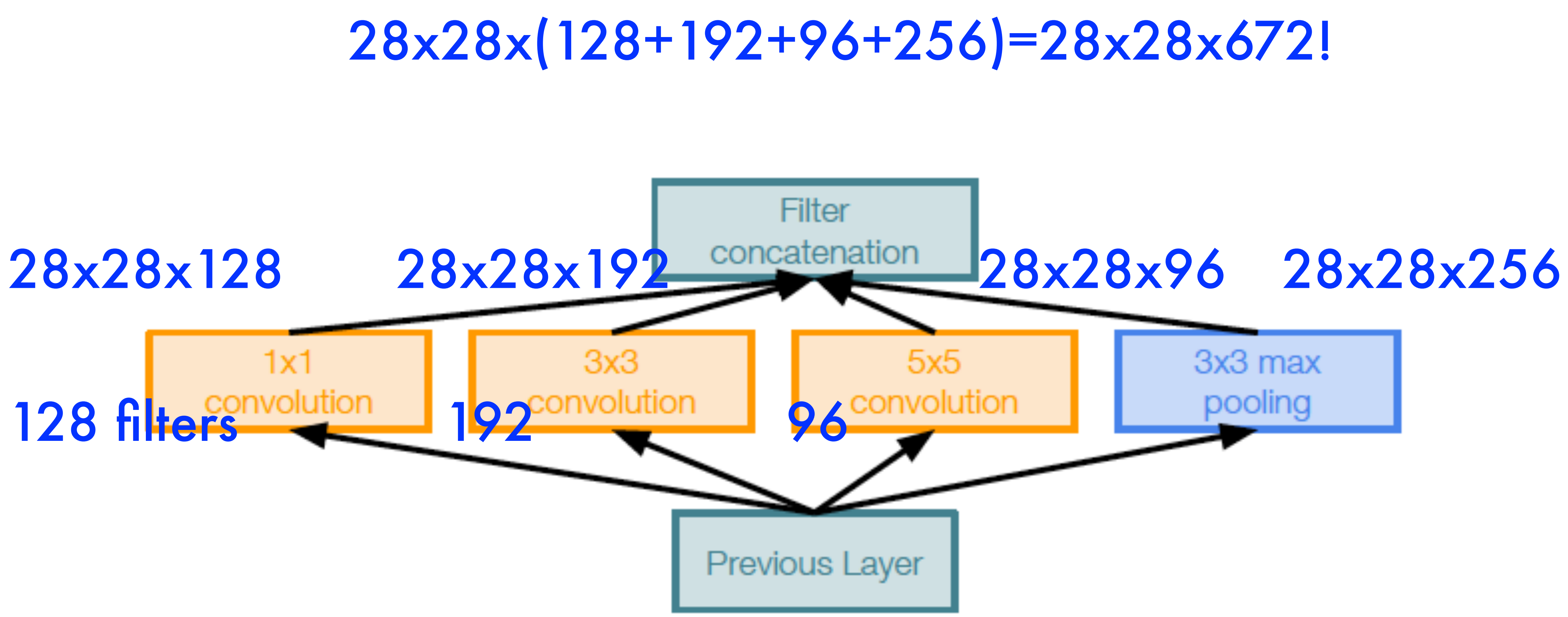
Apply parallel filter operations on the input from previous layer

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

What is the problem with this?
[Hint: Computational complexity!]

# GoogLeNet




Naive Inception module

Module input: 28x28x256

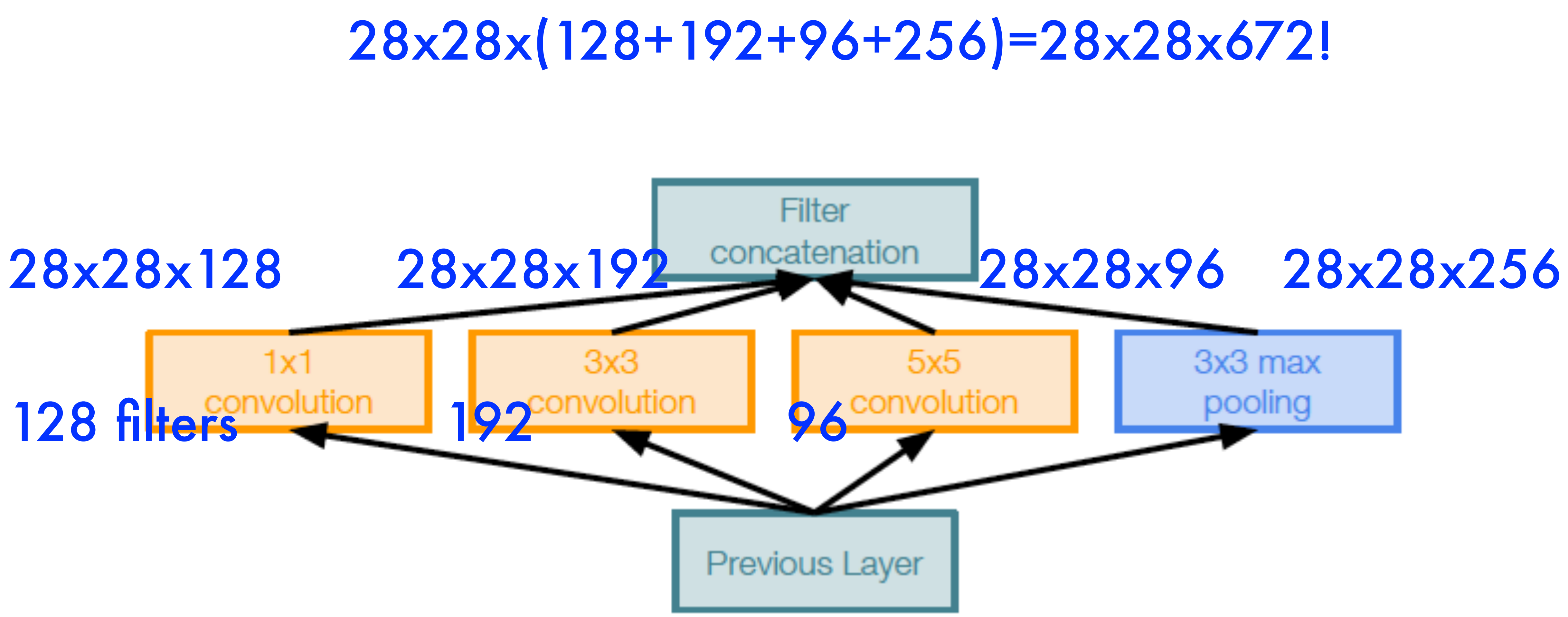
What is the problem with this?
[Hint: Computational complexity!]

What is the output size of the 1x1 conv with 128 filters?

What is the output size of all of the different filter operations?

What is the output size after filter concatenation?

# GoogLeNet




Naive Inception module

**Conv Ops:**
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x4 conv, 96] 28x28x96x5x5x256
**Total of 854M ops!**

Very expensive compute!

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

**See next lecture for solution!**

# Further reading


- Abu-Mostafa, Y. S., Magdon-Ismail, M., Lin, H.-T. (2012) Learning from data. AMLbook.com.
- Goodfellow et al. (2016) Deep Learning. https://www.deeplearningbook.org/
- Boyd, S., and Vandenberghe, L. (2018) Introduction to Applied Linear Algebra - Vectors, Matrices, and Least Squares. http://vmls-book.stanford.edu/
- VanderPlas, J. (2016) Python Data Science Handbook. https://jakevdp.github.io/PythonDataScienceHandbook/