

CNN Architectures

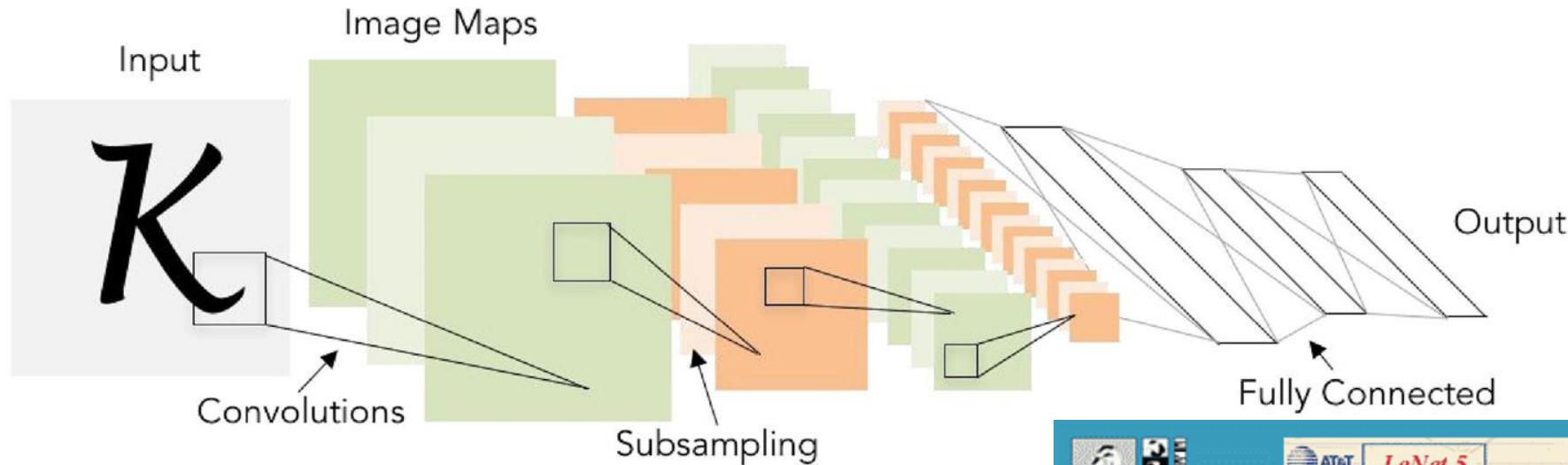
EE 5179: Deep Learning for Imaging
Instructor: Kaushik Mitra

Case study of following CNN architectures

- LeNet
- AlexNet
- VGG
- GoogLeNet
- ResNet

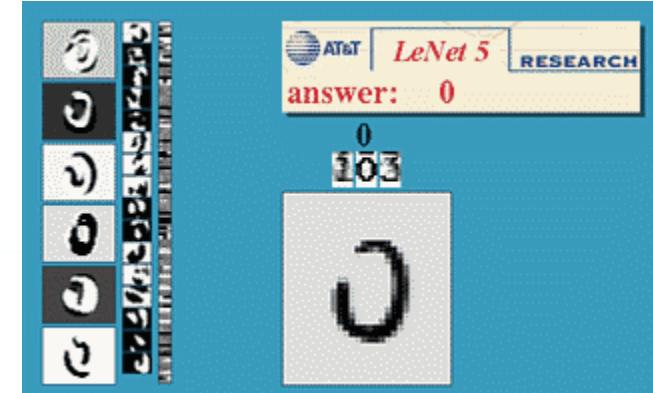
Review: LeNet-5

[LeCun et al., 1998]



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]



Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

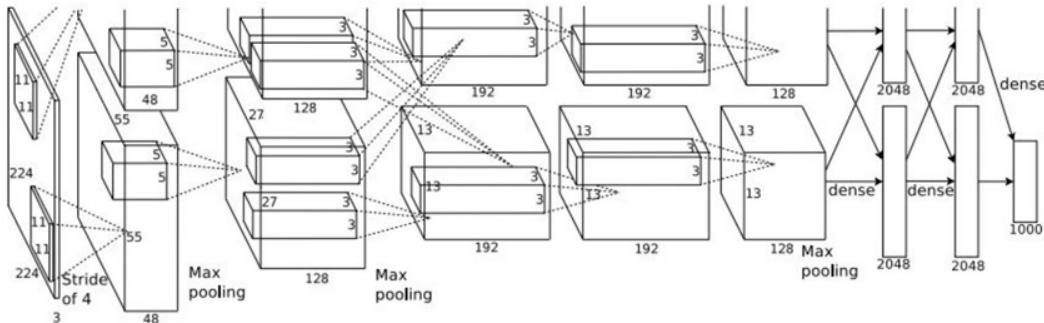
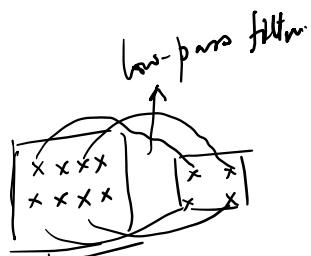
CONV5

Max POOL3

FC6

FC7

FC8



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

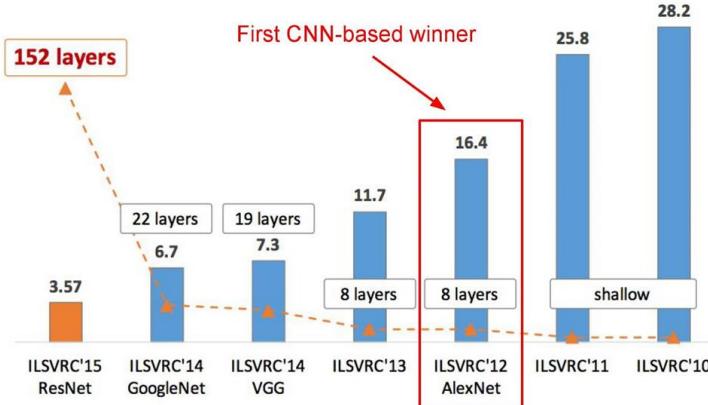


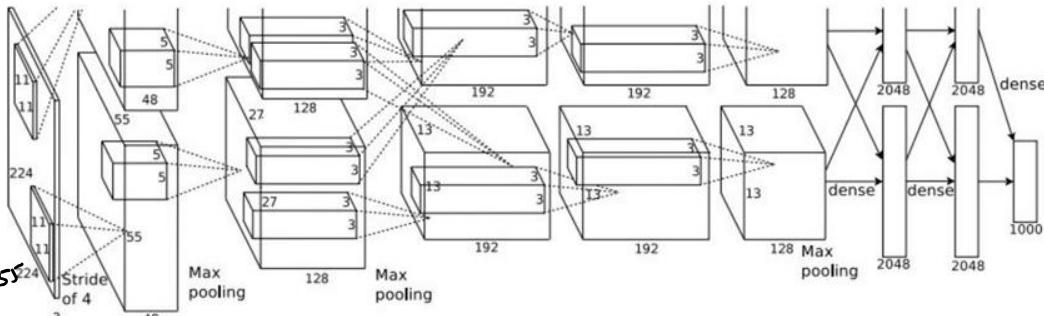
Figure copyright Kaiming He, 2016. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]

$$w_2 = \frac{w_1 - F + 2P}{S} + 1$$
$$\frac{227 - 11}{4} + 1$$
$$\frac{215}{4} + 1 = 54$$

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% \rightarrow 15.4%

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

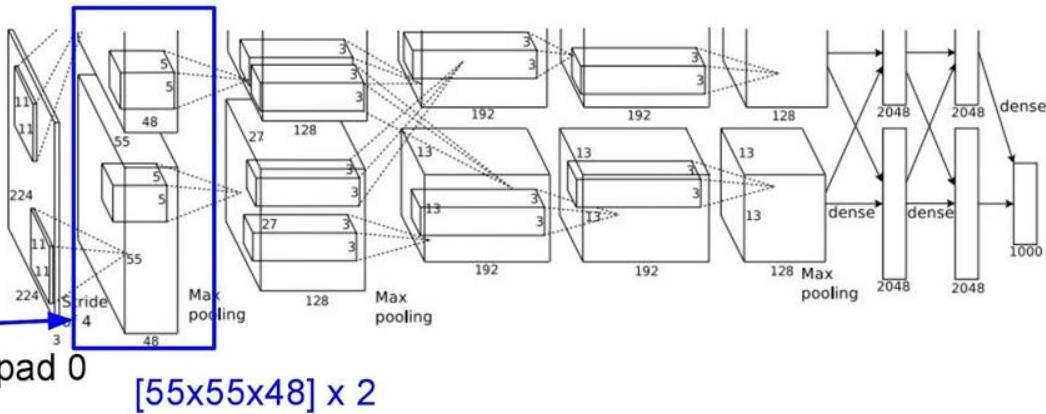
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

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

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



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.

Note on Alexnet

- Notice the use of filters of size 11x11 in the initial layer.
- It increases the number of parameters that need to be trained
- Recent CNNs use cascade of small filters of size 3*3 or 5*5.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

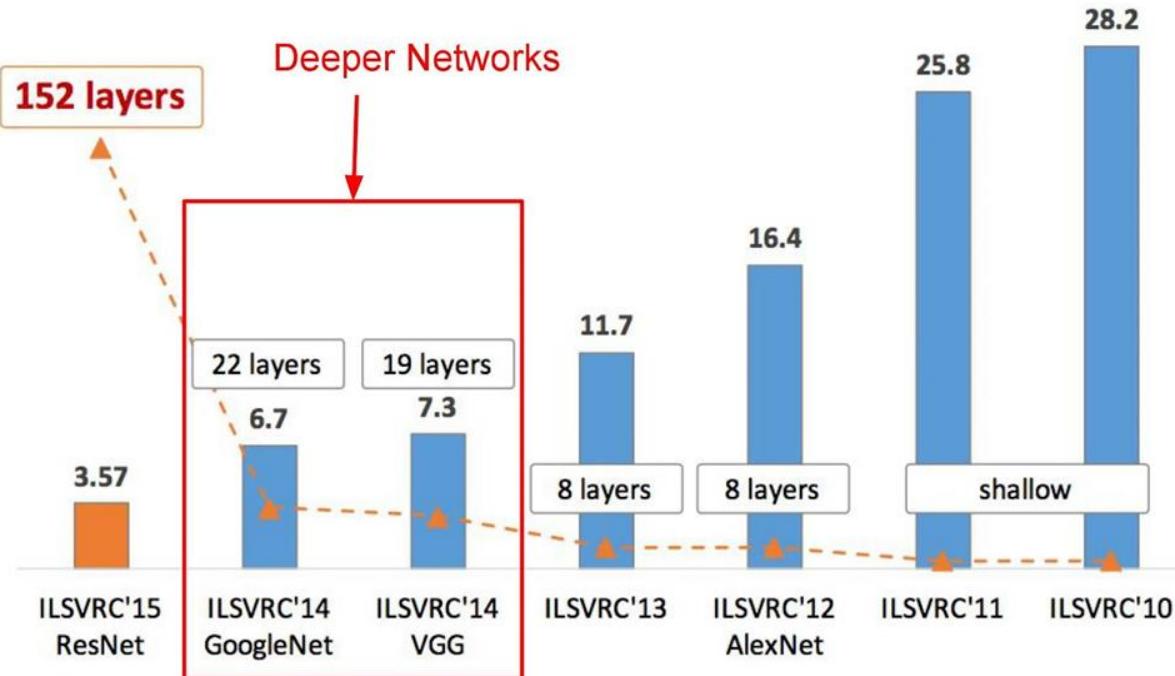


Figure copyright Kaiming He, 2016. Reproduced with permission.

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

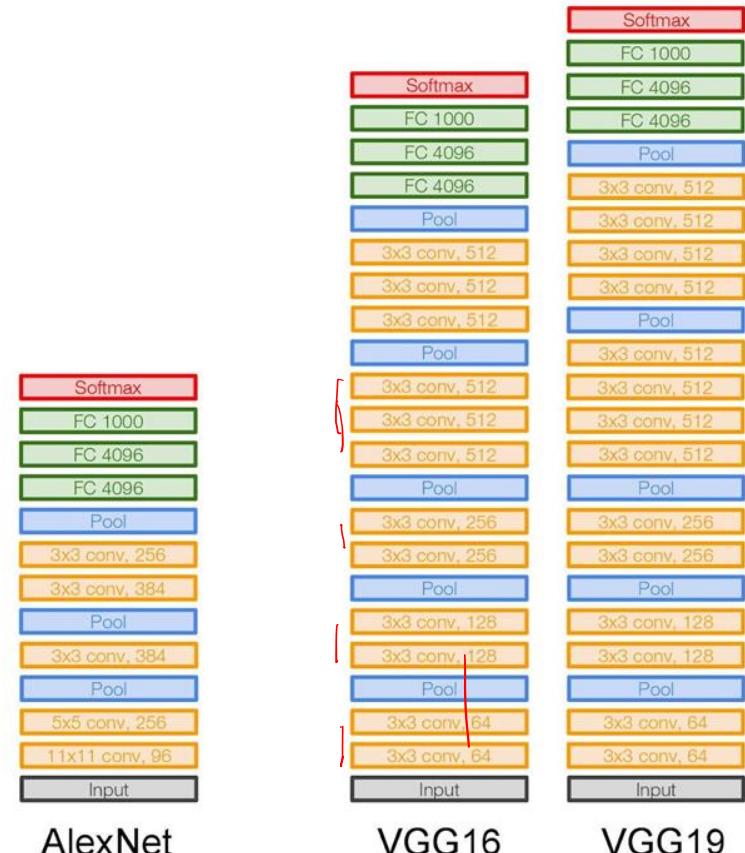
Small filters, Deeper networks

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

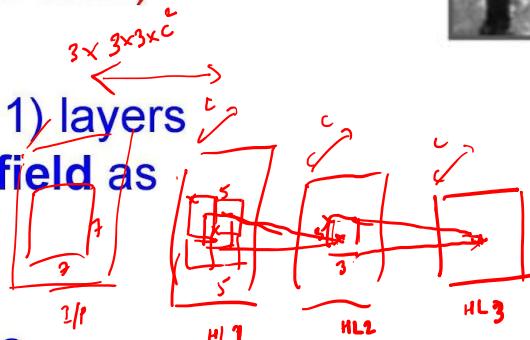


Case Study: VGGNet

[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

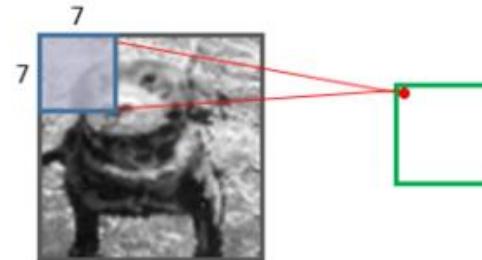


But deeper, more non-linearities

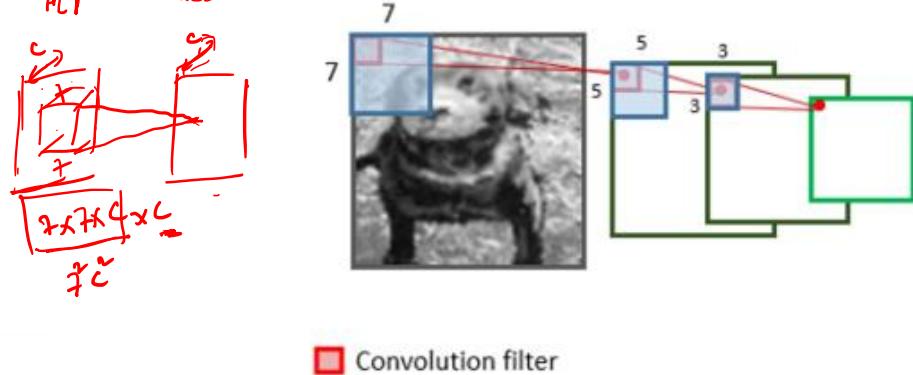
And fewer parameters: $3 * (3^2 C^2)$ vs. $7^2 C^2$ for C channels per layer

$$\begin{aligned} & \text{Stack of } 4 \text{ } 3 \times 3 \text{ conv} \Rightarrow \text{receptive field} = 9 \times 9 \\ & 4 \times 3^2 C = 36 C \quad 81 C \end{aligned}$$

« Conventional » approach
Using one convolutional layer with a large receptive field (7x7)



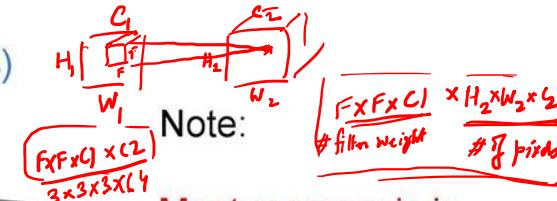
« VGG net » approach
Stacking three (3x3) convolutional layers



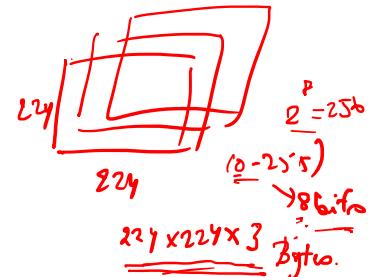
INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150\text{K}$ params: 0 (not counting biases)
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2\text{M}$ params: $(3 \times 3 \times 3) \times 64 = 1,728$
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2\text{M}$ params: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

TOTAL memory: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB} / \text{image}$ (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

$138 \text{M} \times 4 \text{ bytes}$



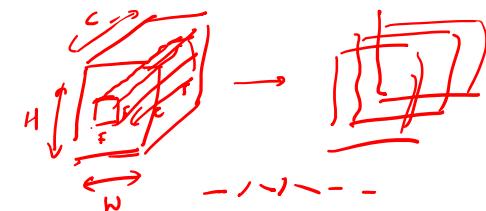
Most memory is in early CONV



Most params are in late FC

$8 \text{ bits} = 1 \text{ byte}$

float - 4 bytes



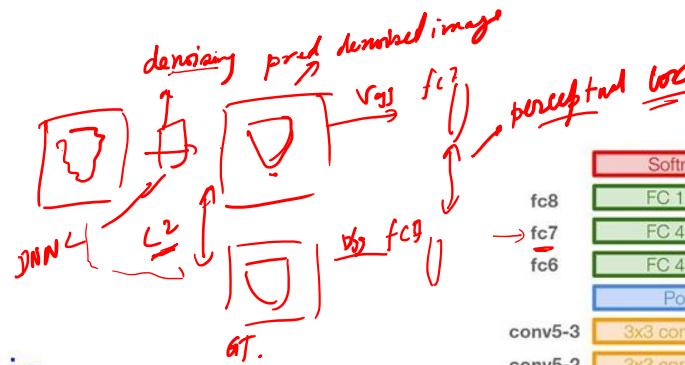
Case Study: VGGNet

[Simonyan and Zisserman, 2014]



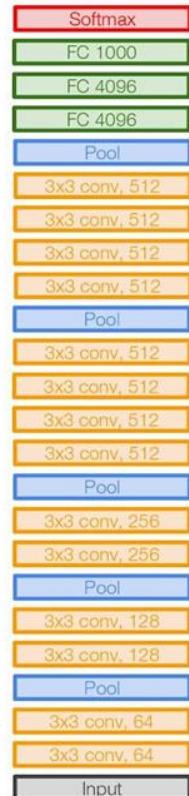
Details:

- 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



AlexNet

VGG16



VGG19

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

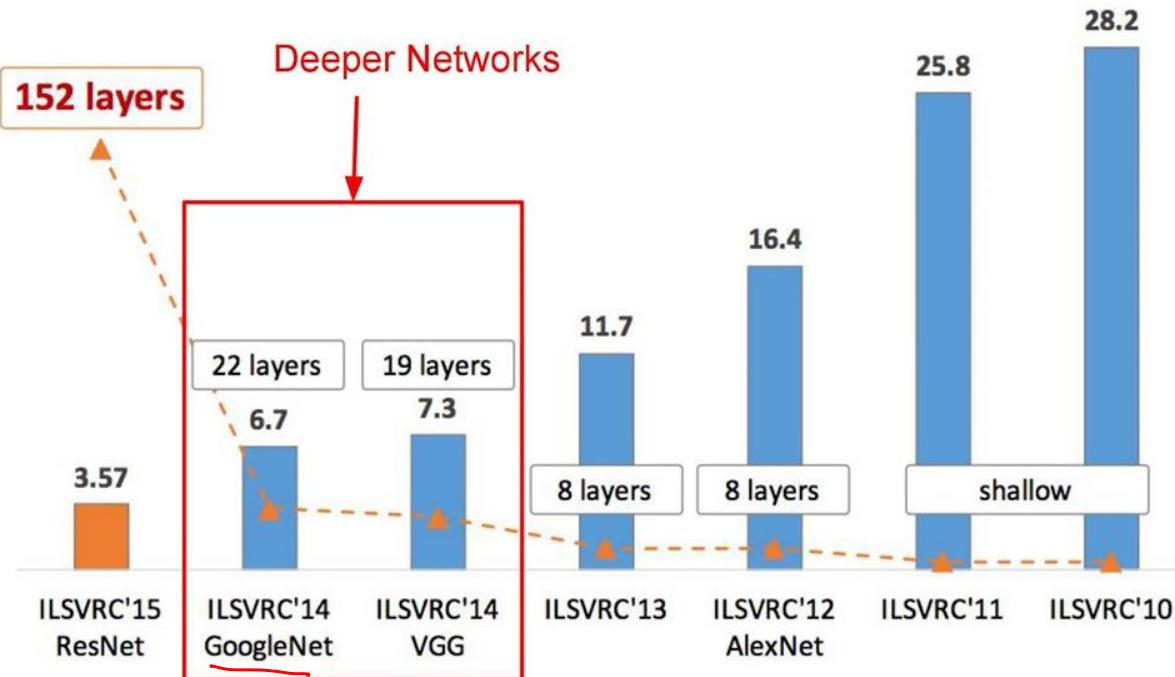


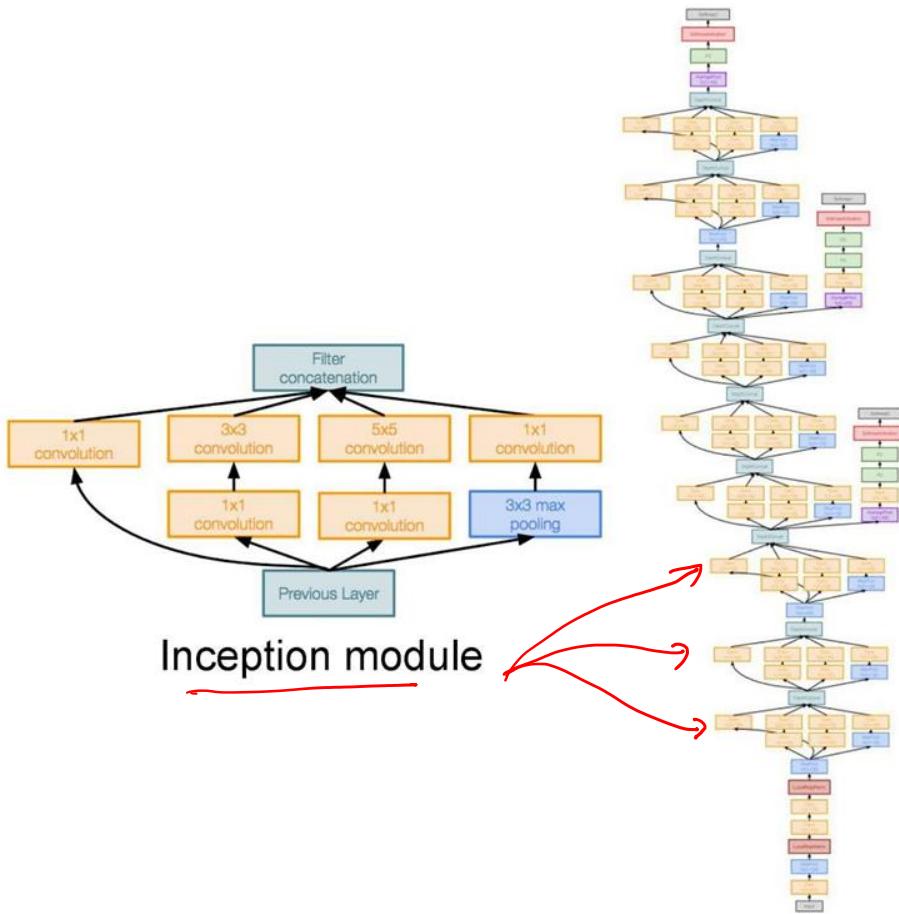
Figure copyright Kaiming He, 2016. Reproduced with permission.

Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

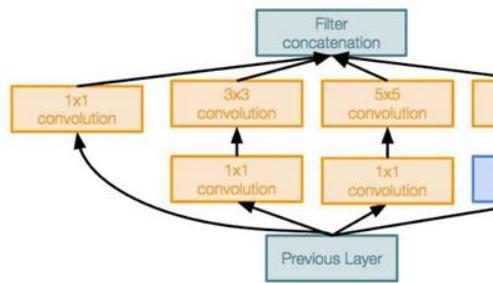
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
12x less than AlexNet
- ILSVRC’14 classification winner
(6.7% top 5 error)



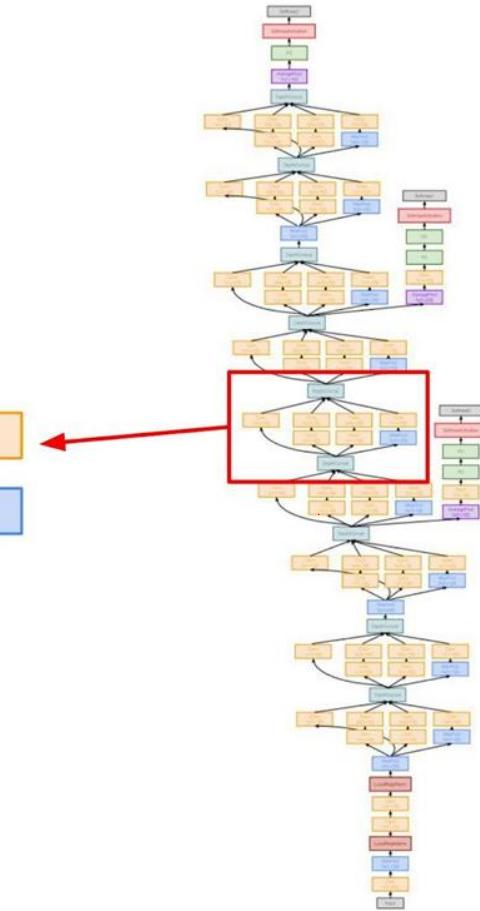
Case Study: GoogLeNet

[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

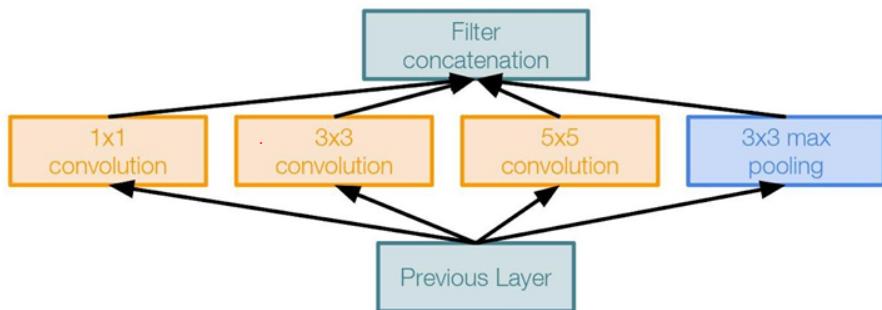


Inception module

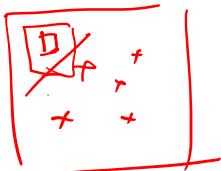


Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

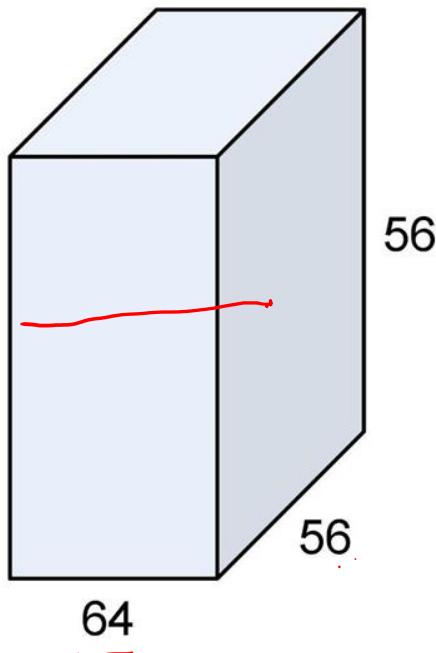


Apply parallel filter operations on the input from previous layer:

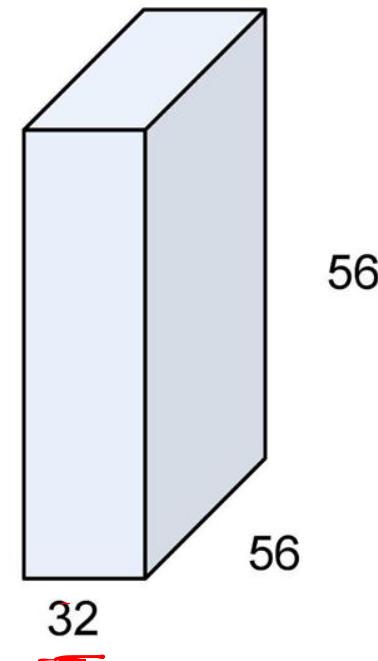
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

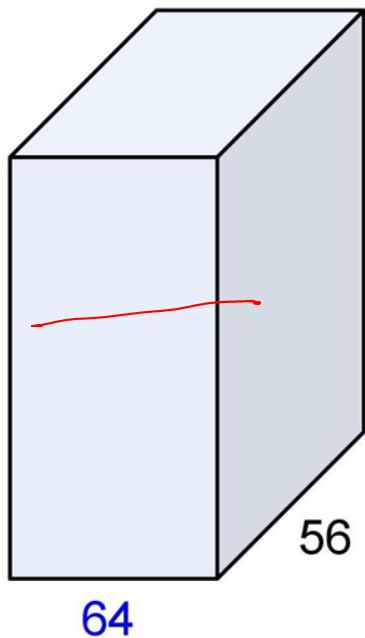
1x1 Convolutions



1x1 CONV
with 32 filters
→
(each filter has size
1x1x64, and performs a
64-dimensional dot
product)



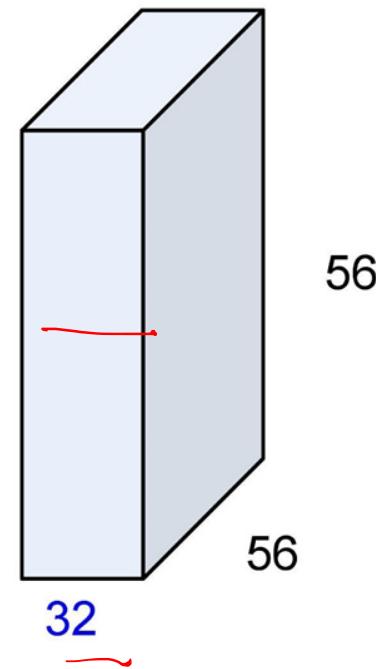
1x1 Convolutions



1x1 CONV
with 32 filters

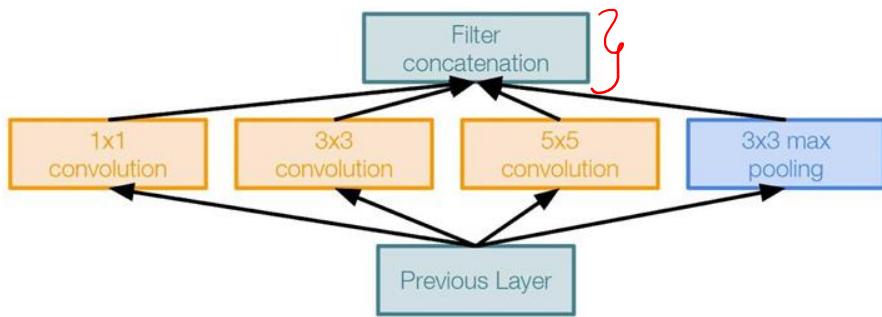
preserves spatial
dimensions, reduces depth!

Projects depth to lower
dimension (combination of
feature maps)



Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1×1 , 3×3 , 5×5)
- Pooling operation (3×3)

Concatenate all filter outputs together depth-wise

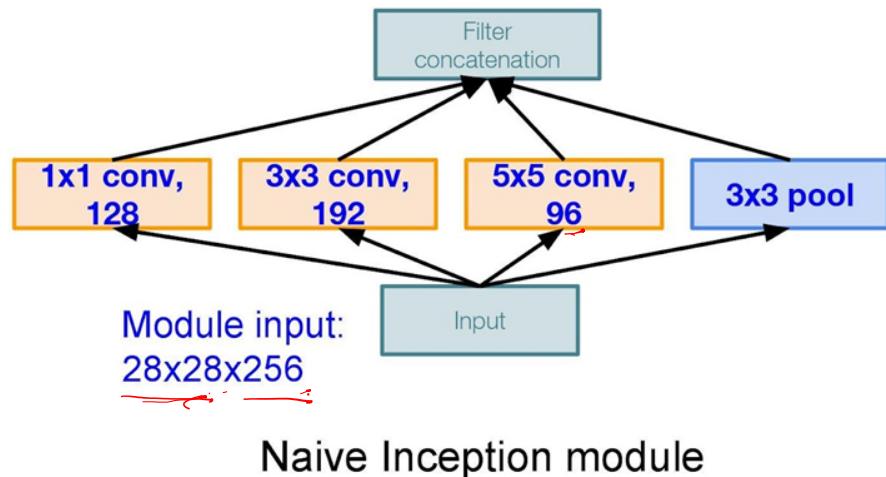
Q: What is the problem with this?
[Hint: Computational complexity]

Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:



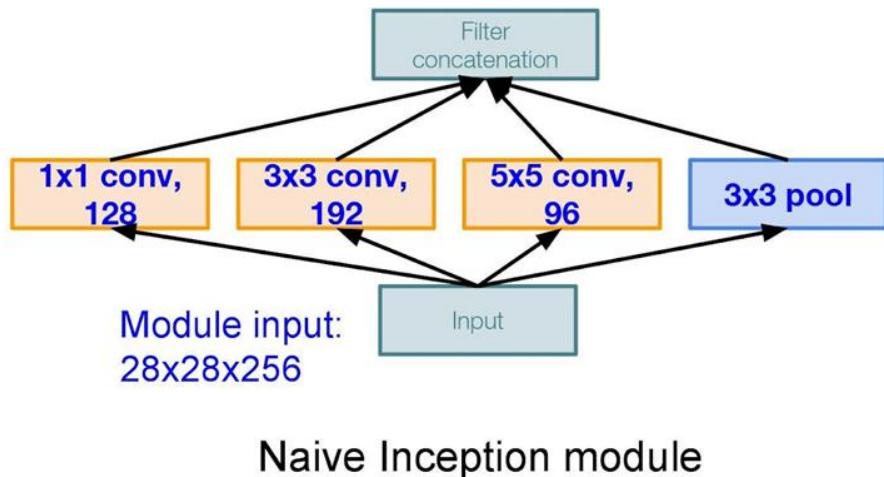
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q1: What is the output size of the
1x1 conv, with 128 filters?

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

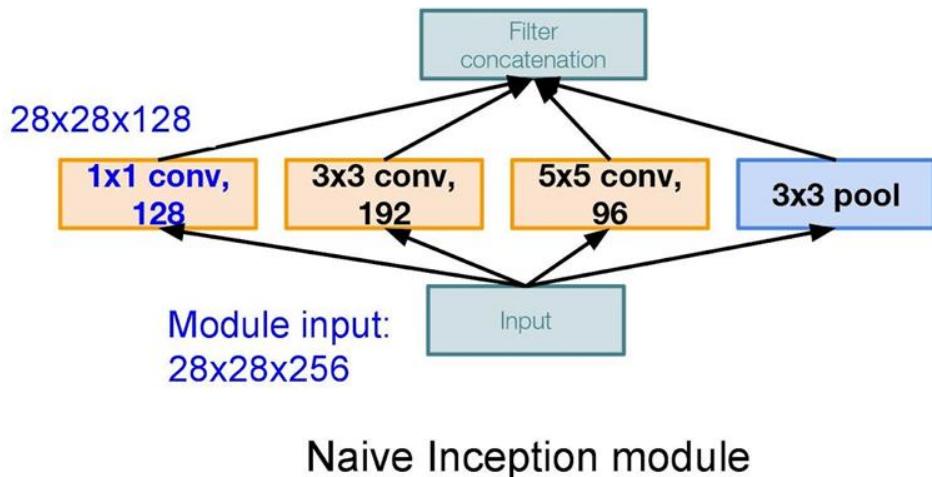


Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q1: What is the output size of the
1x1 conv, with 128 filters?



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

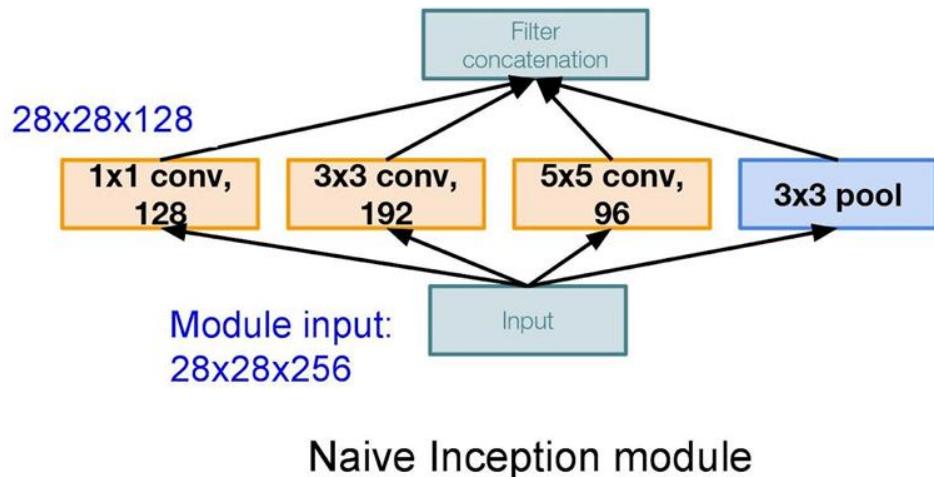
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What are the output sizes of all different filter operations?

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



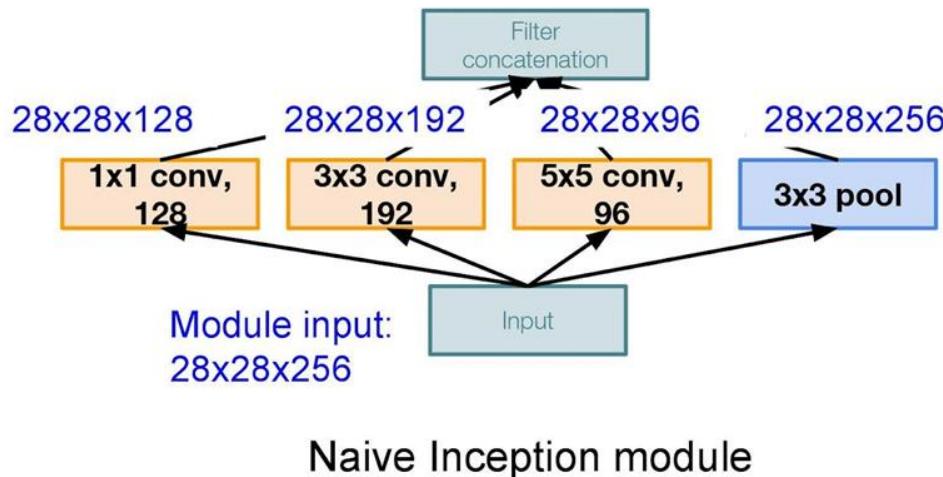
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What are the output sizes of all different filter operations?

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



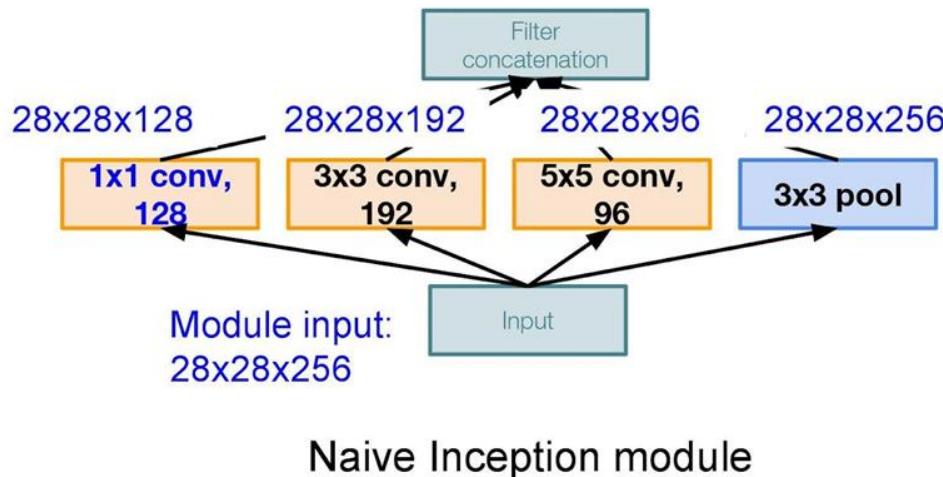
Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

Q3: What is output size after
filter concatenation?



Case Study: GoogLeNet

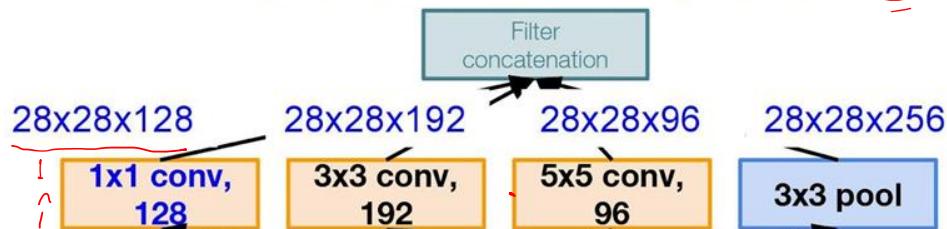
[Szegedy et al., 2014]

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

Example:

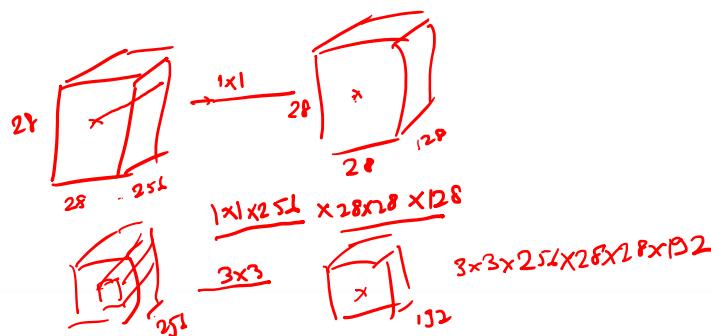
Q3: What is output size after
filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times \underline{672}$$



Module input:
 $28 \times 28 \times 256$

Naive Inception module



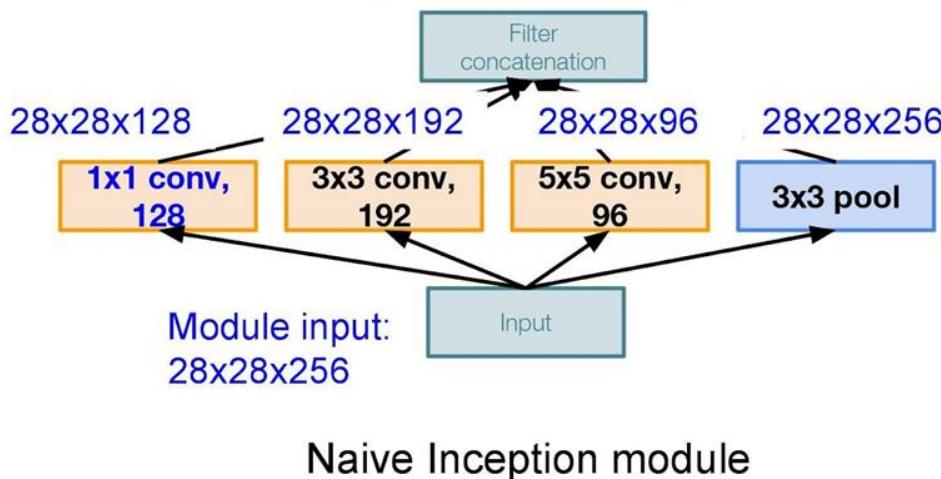
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



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

Conv Ops:

[1×1 conv, 128] $\cancel{28 \times 28 \times 128 \times 1 \times 256}$

[3×3 conv, 192] $\cancel{28 \times 28 \times 192 \times 3 \times 3 \times 256}$

[5×5 conv, 96] $\cancel{28 \times 28 \times 96 \times 5 \times 5 \times 256}$

Total: 854M ops

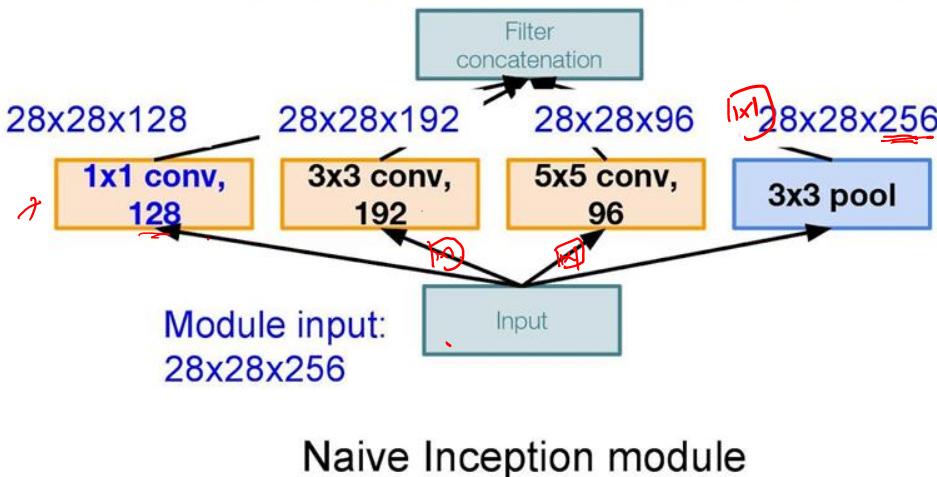
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



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

Conv Ops:

[1×1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3×3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 256$

[5×5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 256$

Total: 854M ops

Very expensive compute

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

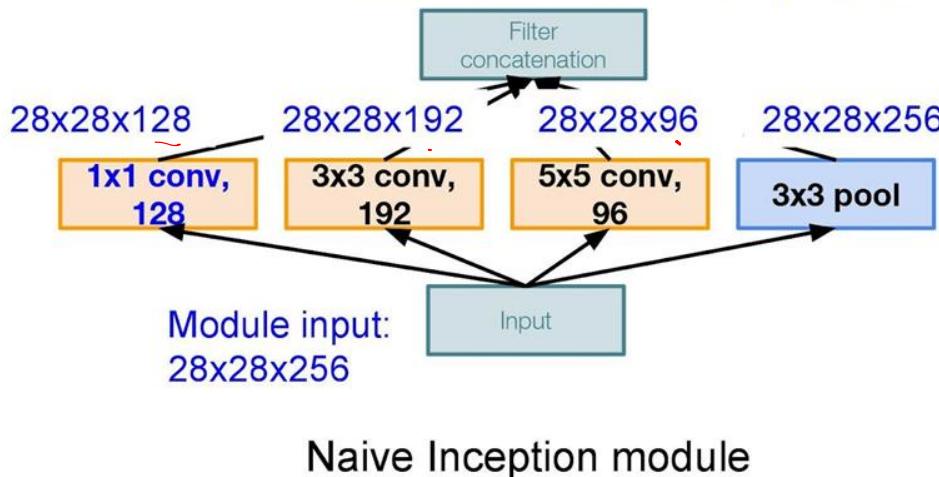
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 529k$$

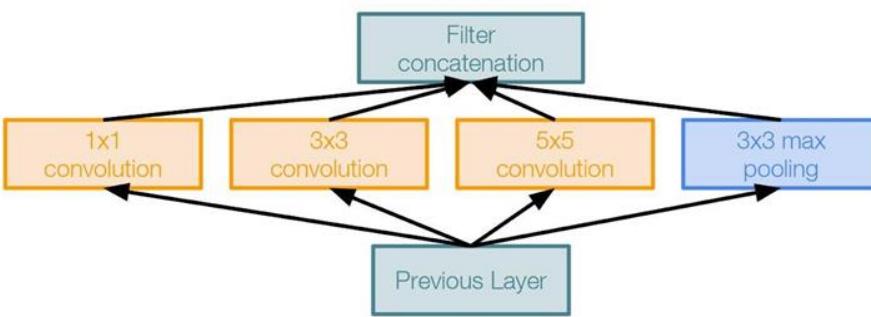


Q: What is the problem with this?
[Hint: Computational complexity]

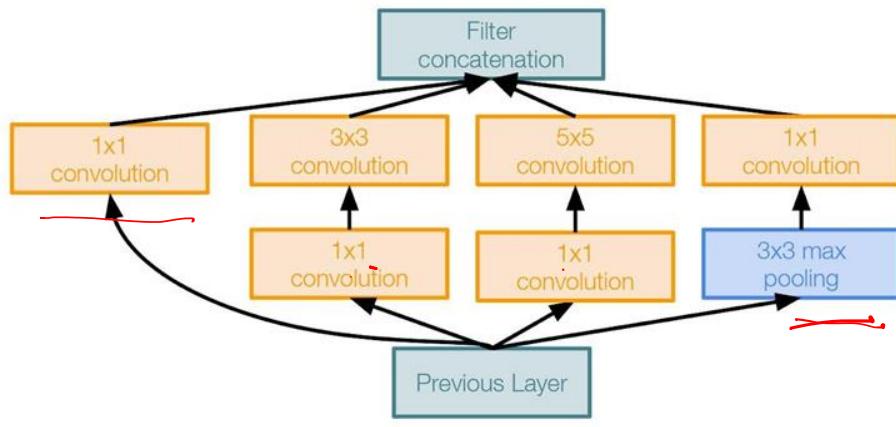
Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature depth

Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

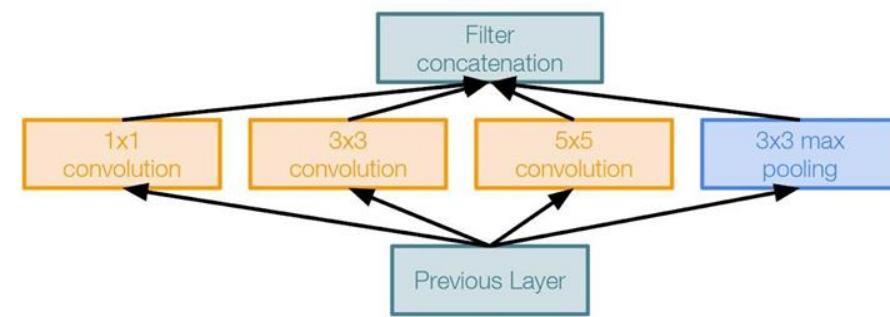


Inception module with dimension reduction

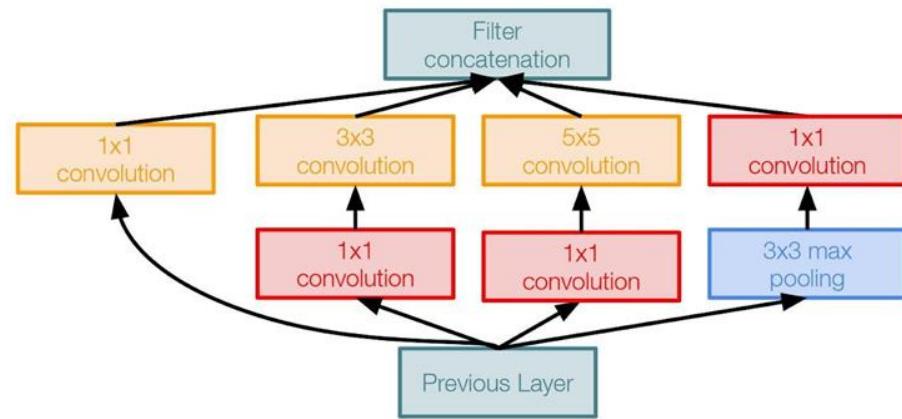
Case Study: GoogLeNet

[Szegedy et al., 2014]

1x1 conv “bottleneck”
layers



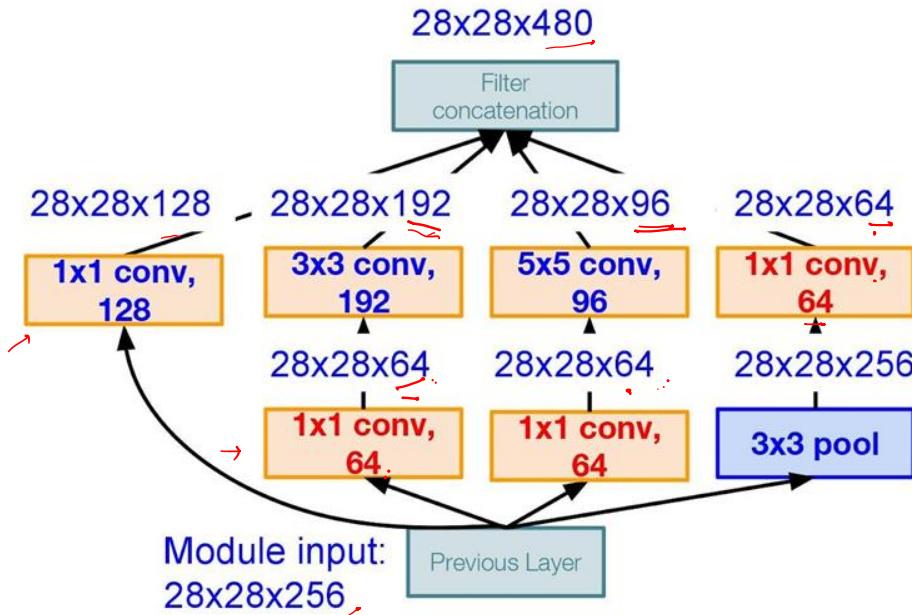
Naive Inception module



Inception module with dimension reduction

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

Conv Ops:

- [1x1 conv, 64] **28x28x64x1x1x256**
- [1x1 conv, 64] **28x28x64x1x1x256**
- [1x1 conv, 128] **28x28x128x1x1x256**
- [3x3 conv, 192] **28x28x192x3x3x64**
- [5x5 conv, 96] **28x28x96x5x5x64**
- [1x1 conv, 64] **28x28x64x1x1x256**

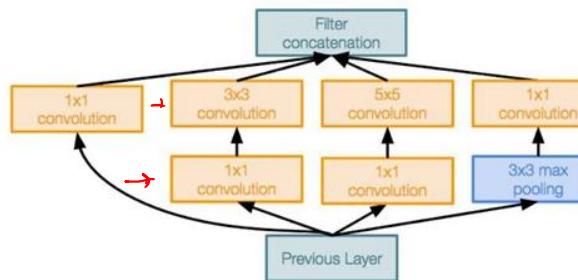
Total: 358M ops

Compared to **854M ops** for naive version
Bottleneck can also reduce depth after pooling layer

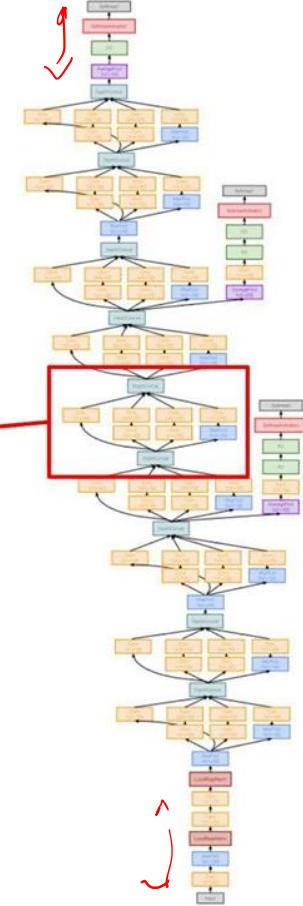
Case Study: GoogLeNet

[Szegedy et al., 2014]

Stack Inception modules
with dimension reduction
on top of each other



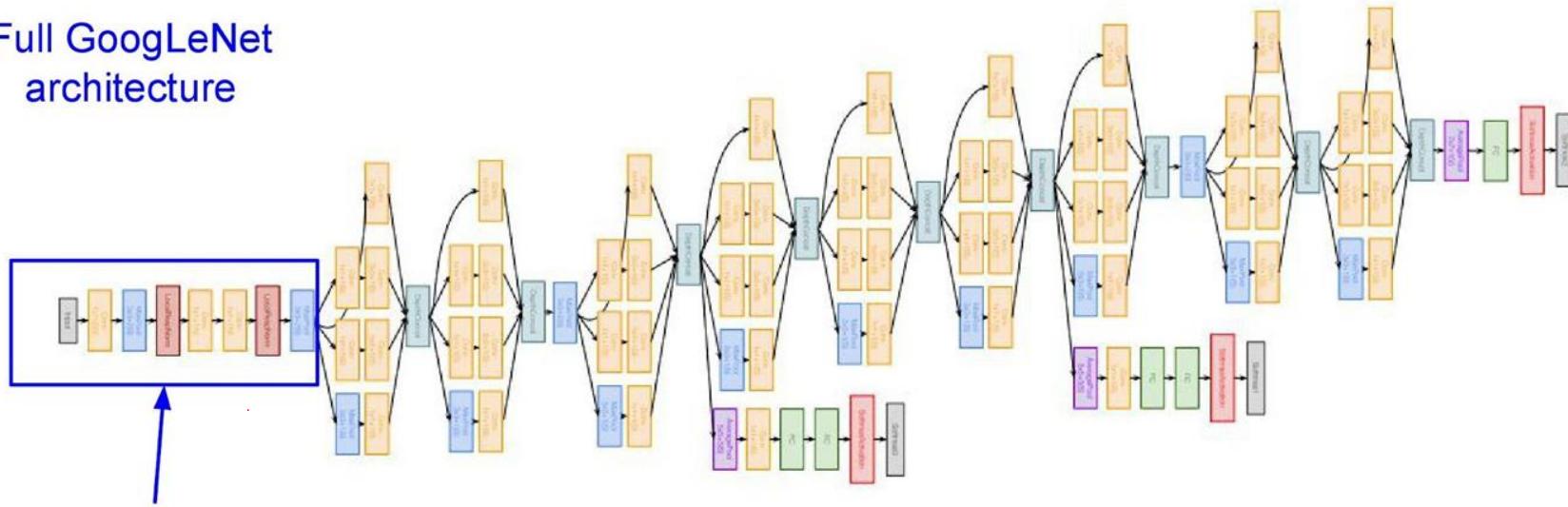
Inception module



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture

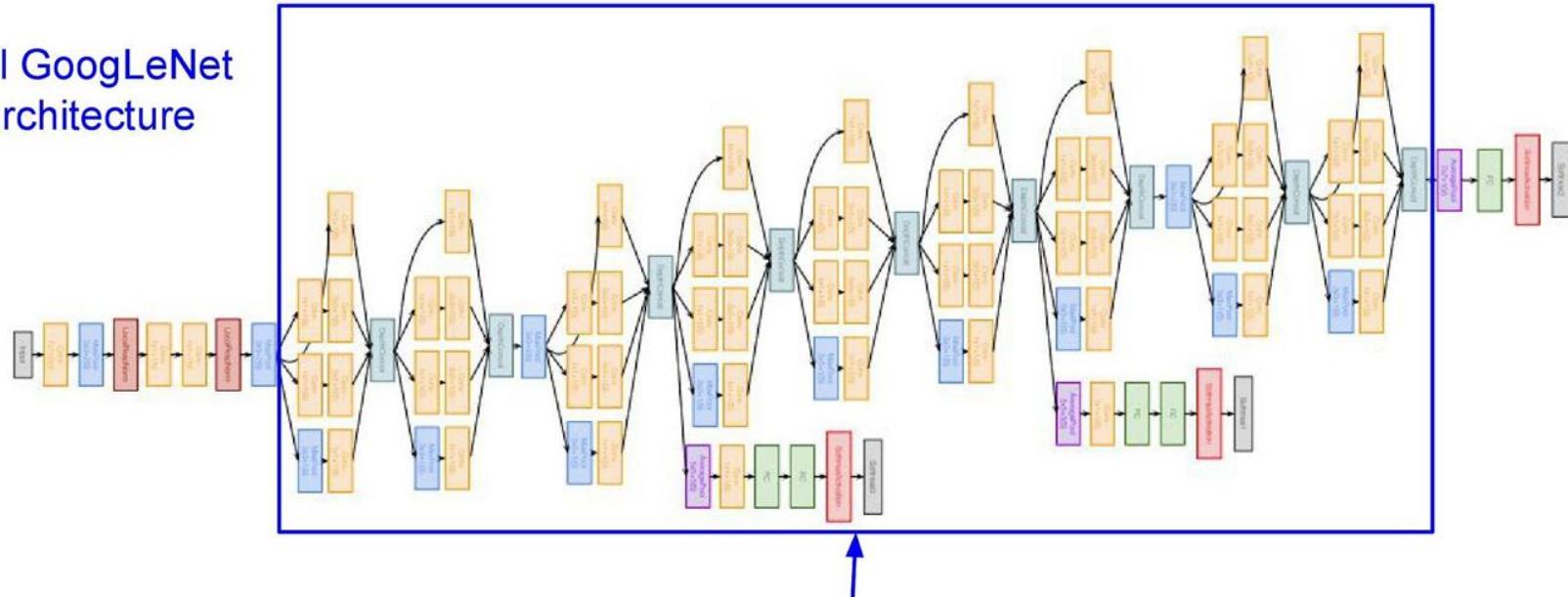


Stem Network:
Conv-Pool-
2x Conv-Pool

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture

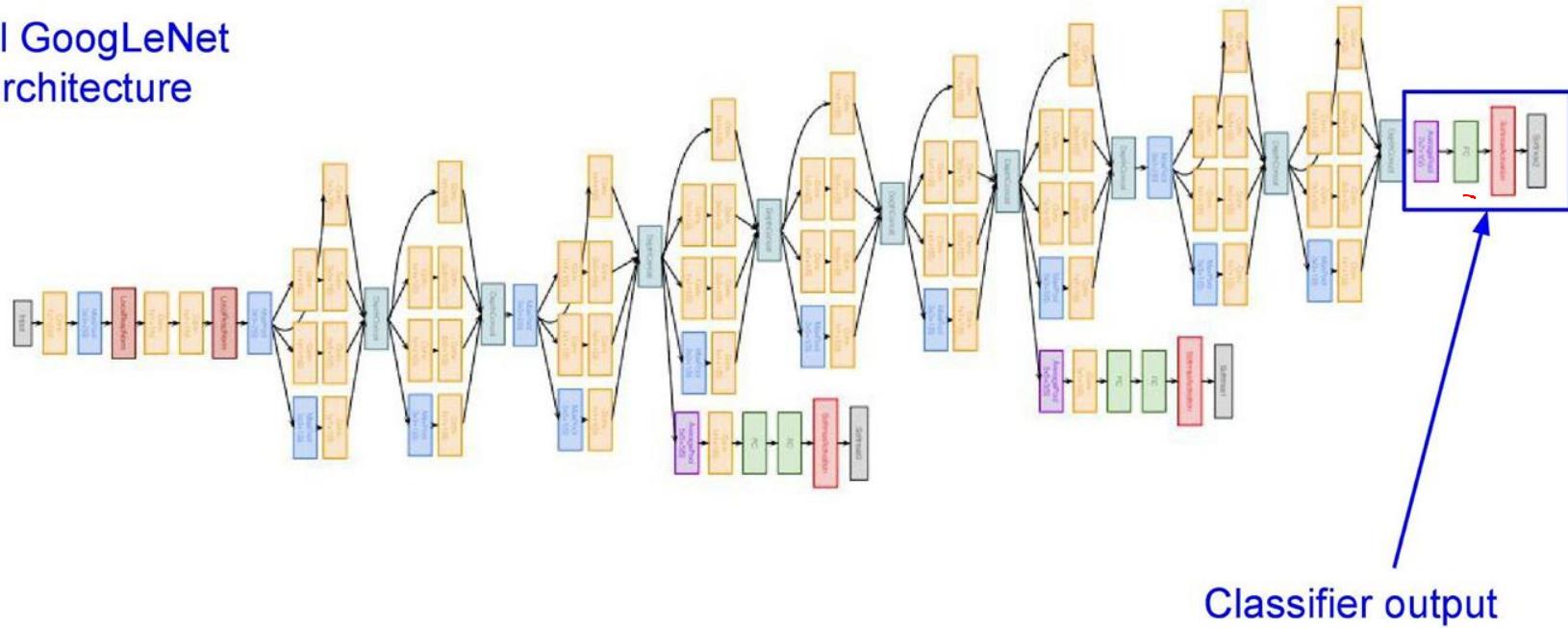


Stacked Inception
Modules

Case Study: GoogLeNet

[Szegedy et al., 2014]

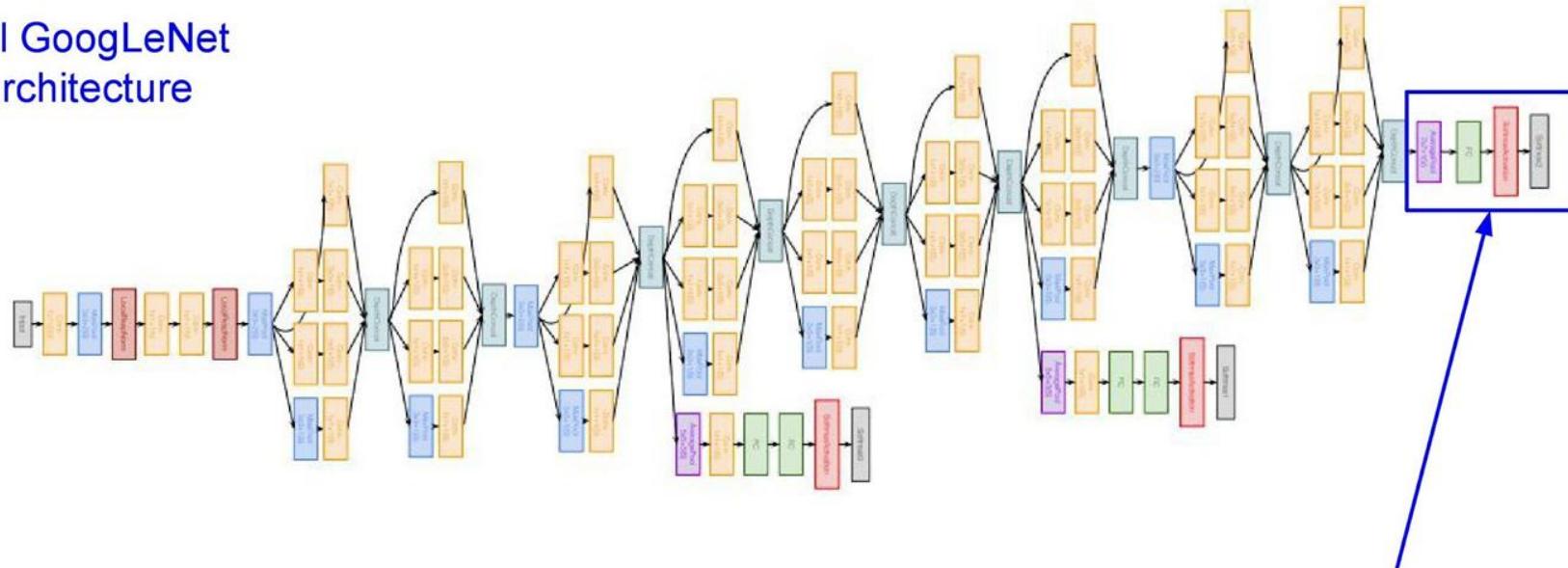
Full GoogLeNet
architecture



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture

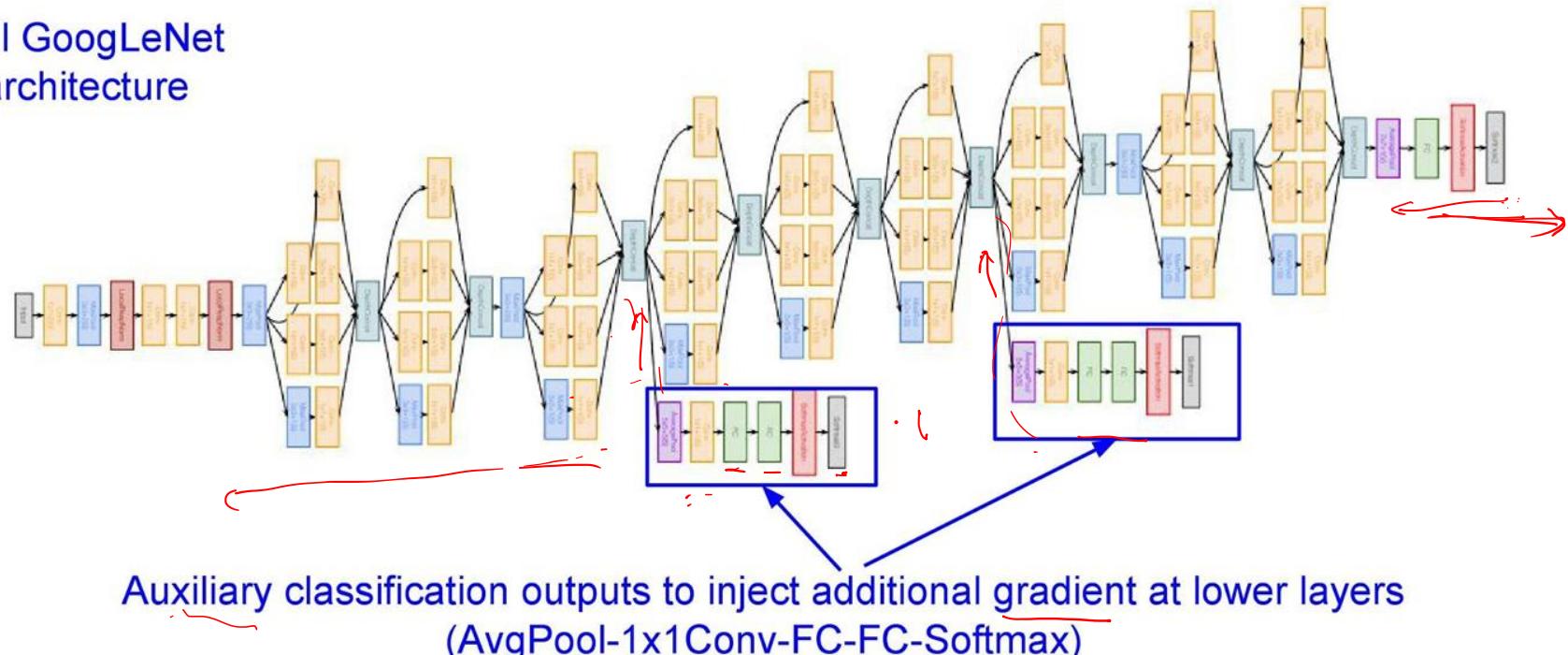


Classifier output
(removed expensive FC layers!)

Case Study: GoogLeNet

[Szegedy et al., 2014]

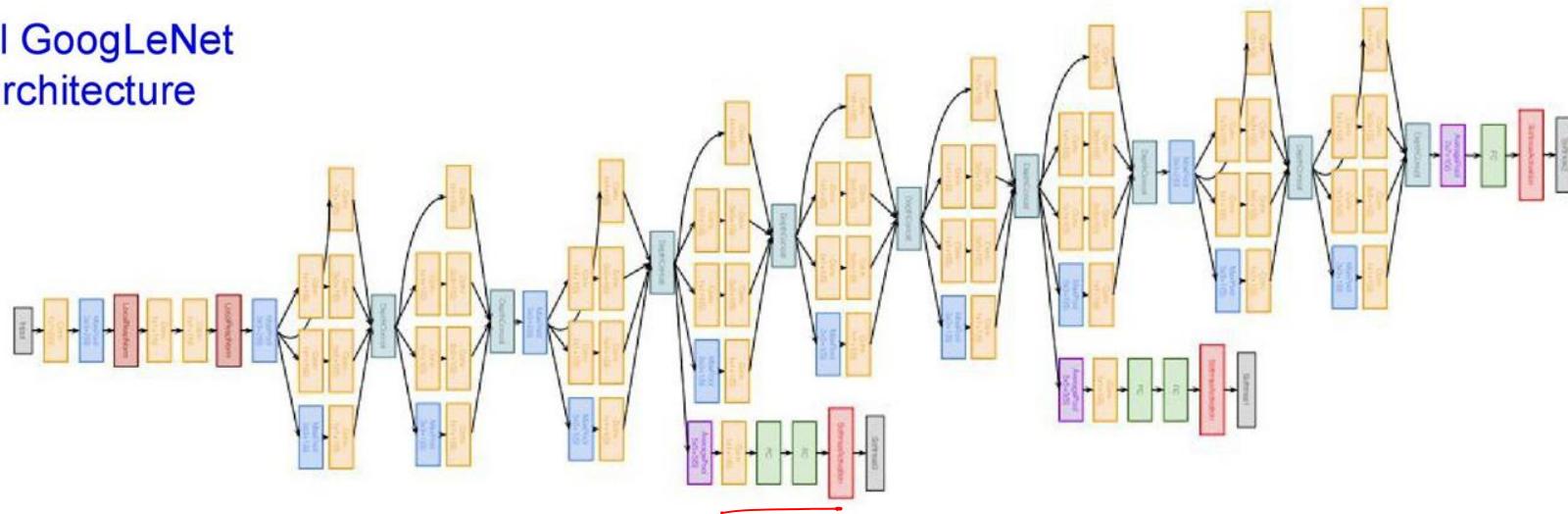
Full GoogLeNet
architecture



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



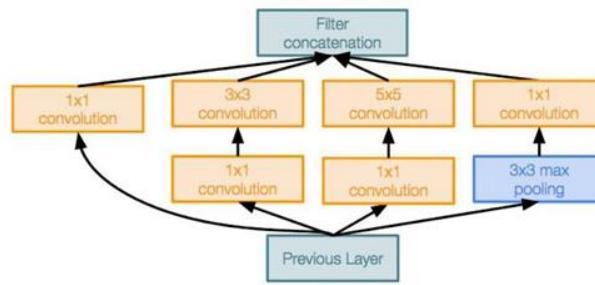
22 total layers with weights (including each parallel layer in an Inception module)

Case Study: GoogLeNet

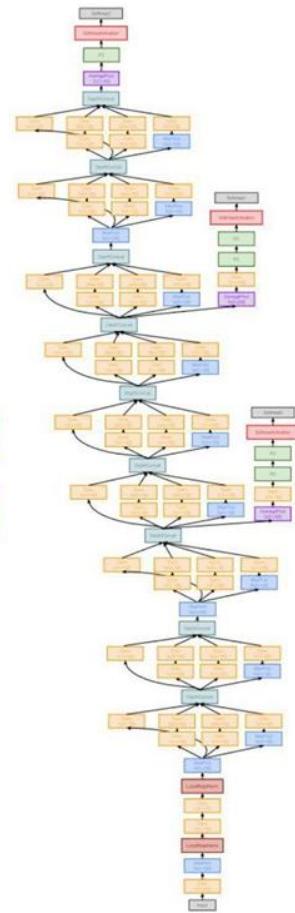
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)



Inception module



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

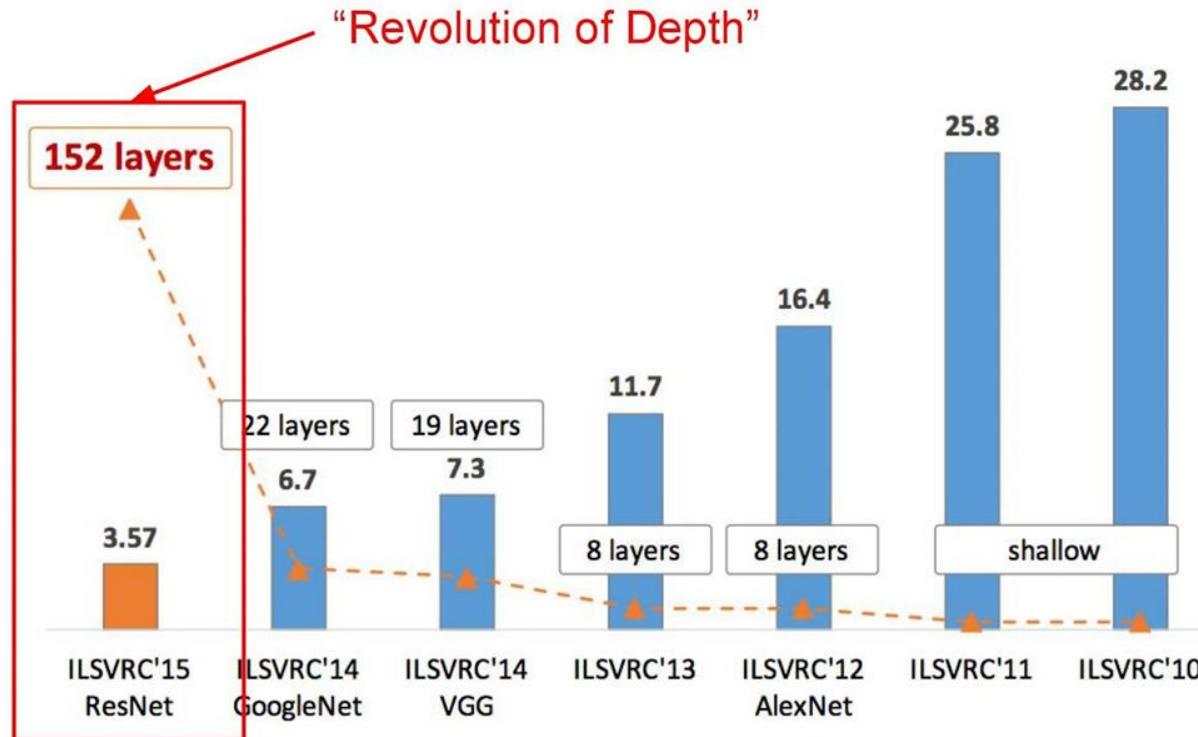


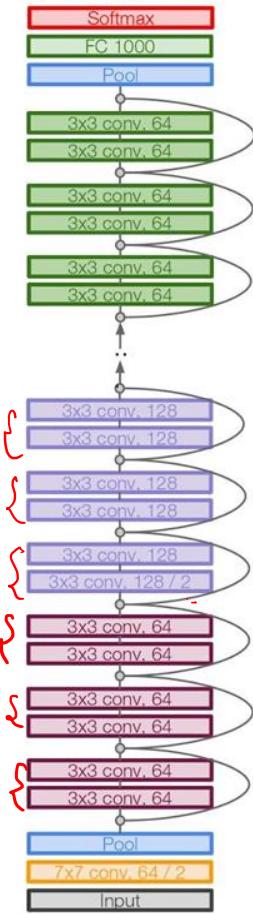
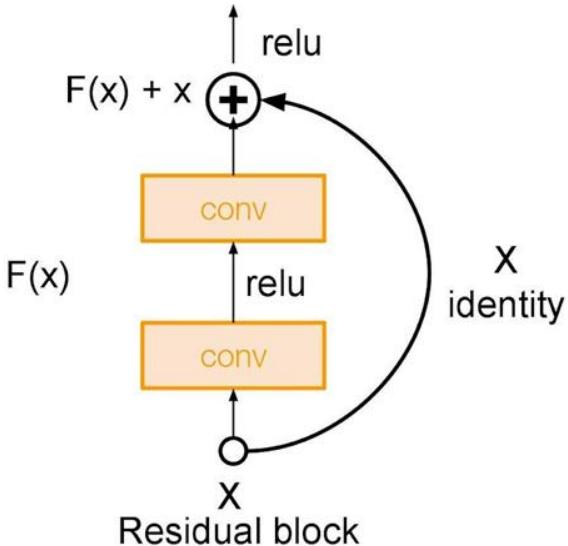
Figure copyright Kaiming He, 2016. Reproduced with permission.

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

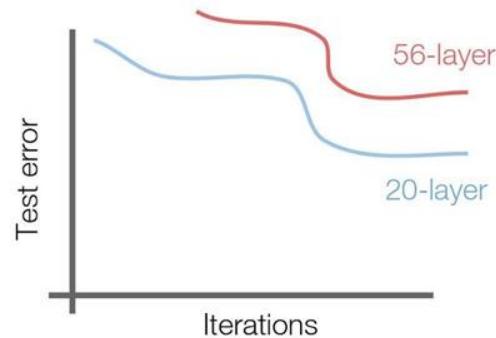
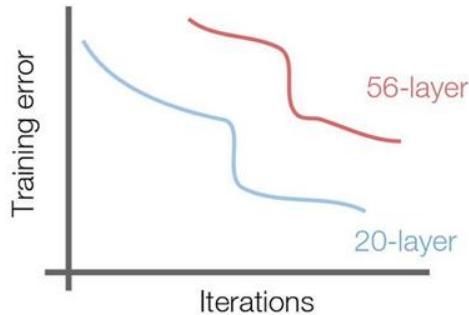
[He et al., 2015]

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?



Q: What's strange about these training and test curves?
[Hint: look at the order of the curves]

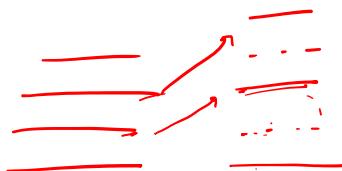
Case Study: ResNet

[He et al., 2015]

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

The deeper model should be able to perform at least as well as the shallower model.

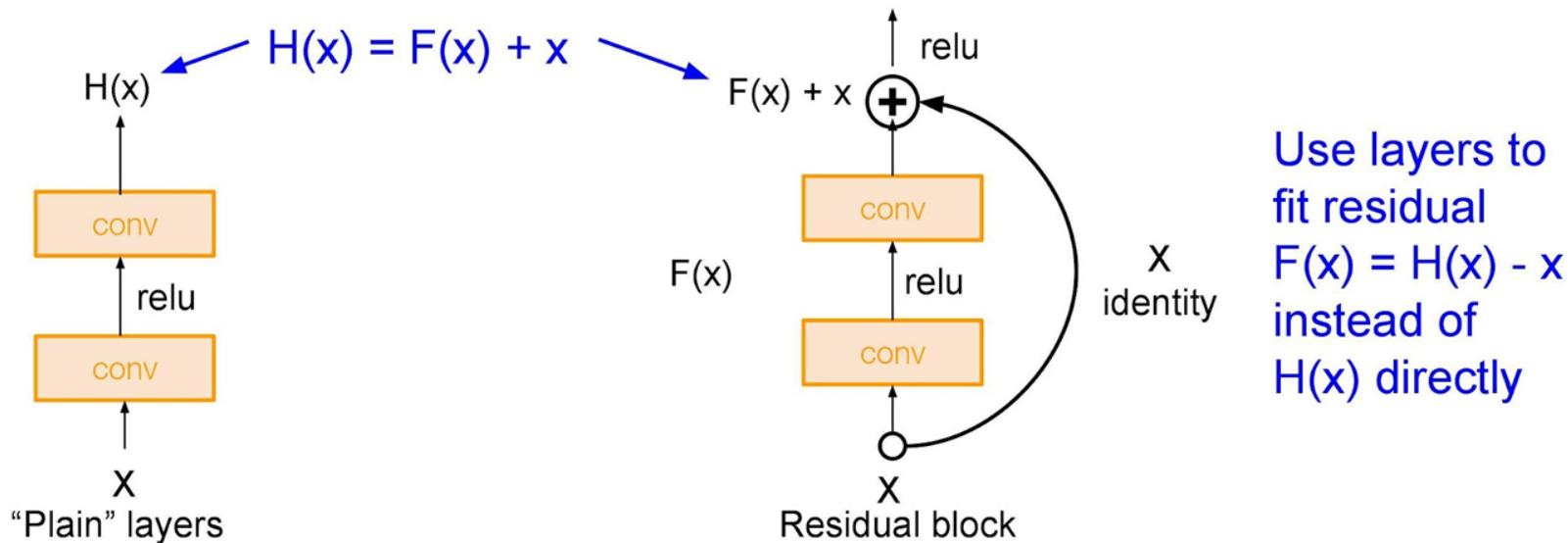
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



Case Study: ResNet

[He et al., 2015]

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

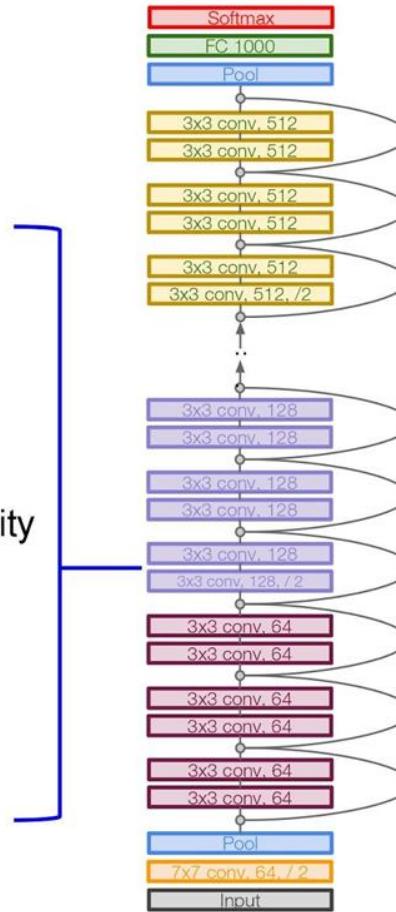
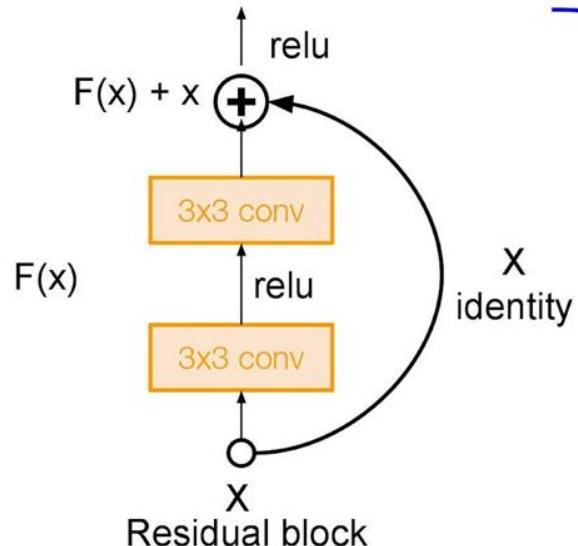


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

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

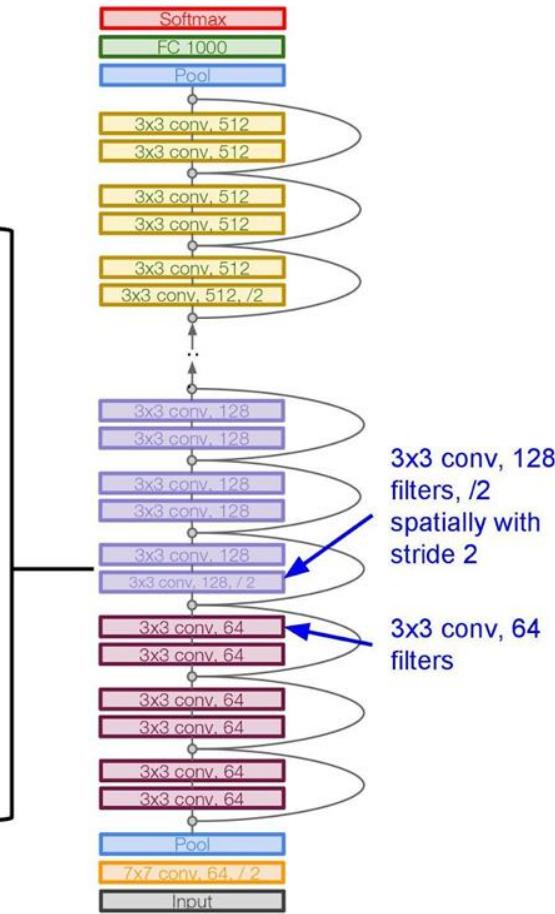
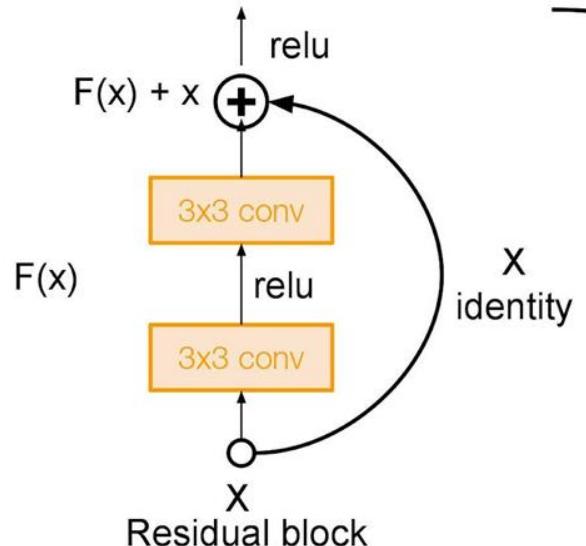


Case Study: ResNet

[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)

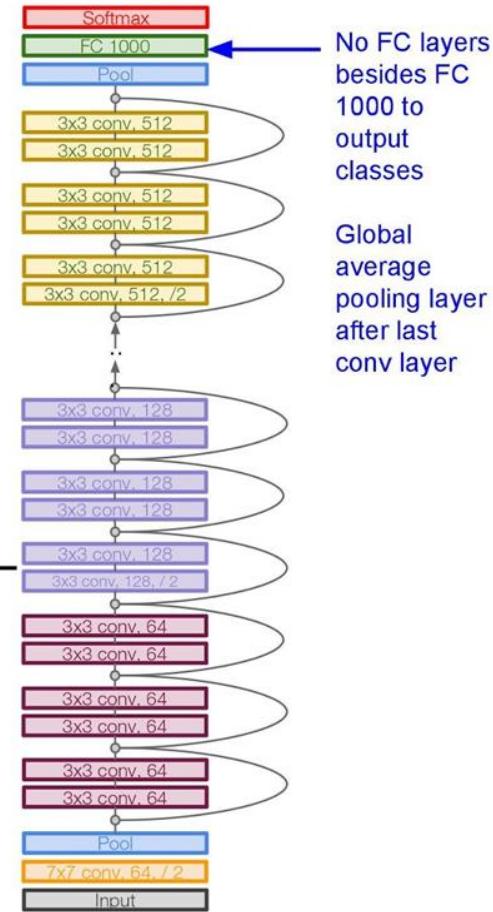
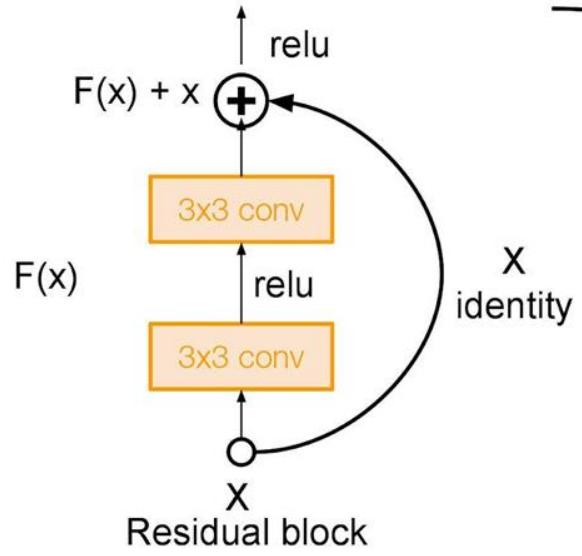


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

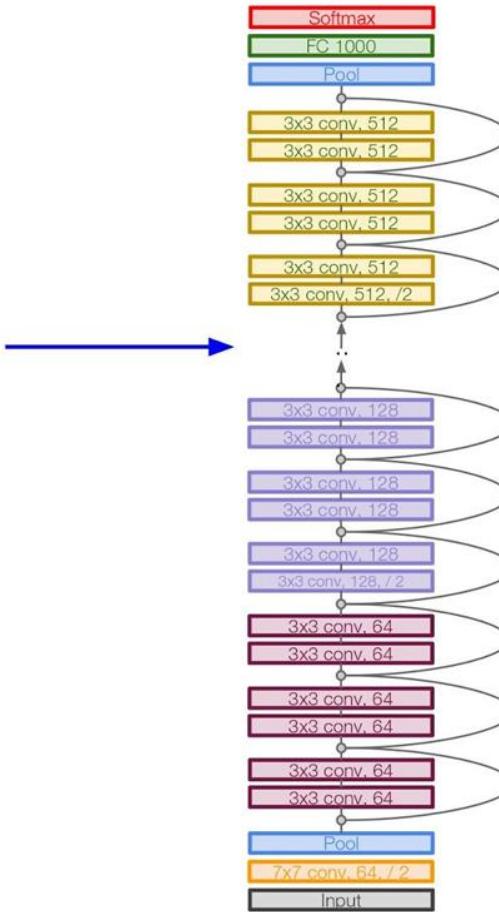
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



Case Study: ResNet

[He et al., 2015]

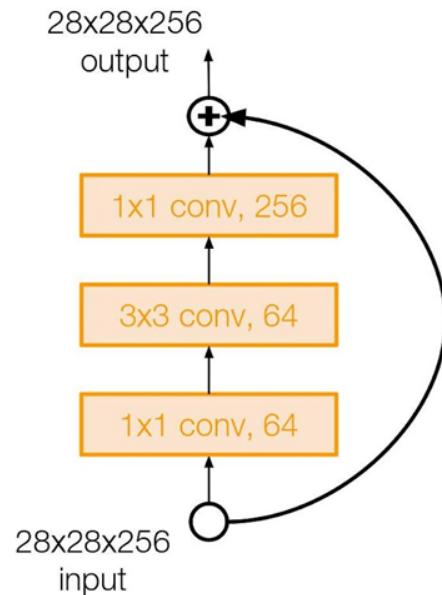
Total depths of 34, 50, 101, or
152 layers for ImageNet



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]

Training ResNet in practice:

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

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

Summary: CNN Architectures

- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Even more recent trend towards examining necessity of depth vs. width and residual connections