Lecture 13: CNNs

Announcements:

- HW #5 is due **Monday, March 4**, uploaded to Gradescope. Appreciate that you went from 35-40% accuracy with softmax on Homework 2 to 65+% accuracy with CNNs!
- Remaining schedule: Today: CNNs, 3/4: CNNs + RNNs, 3/6: RNNs + object detection, 3/11: object detection + adversarial examples, 3/13: adversarial + overview.
- The project and its accompanying data have been uploaded to Bruin Learn. It is due **March 15, 2024** (Friday of Week 10). Custom projects should be requested no later than **Friday, March 1, 2024.**
  - You will be allowed to use PyTorch, Keras, or other deep learning libraries for the project.
  - The TAs will release a Jupyter Notebook that implements a CNN + LSTM hybrid this Friday at Discussion.
  - Although we have yet to cover RNNs, you can feel free to implement them using LSTM cells in PyTorch (https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html) or Keras (https://keras.io/api/layers/recurrent_layers/lstm/) and play with hyperparameters.

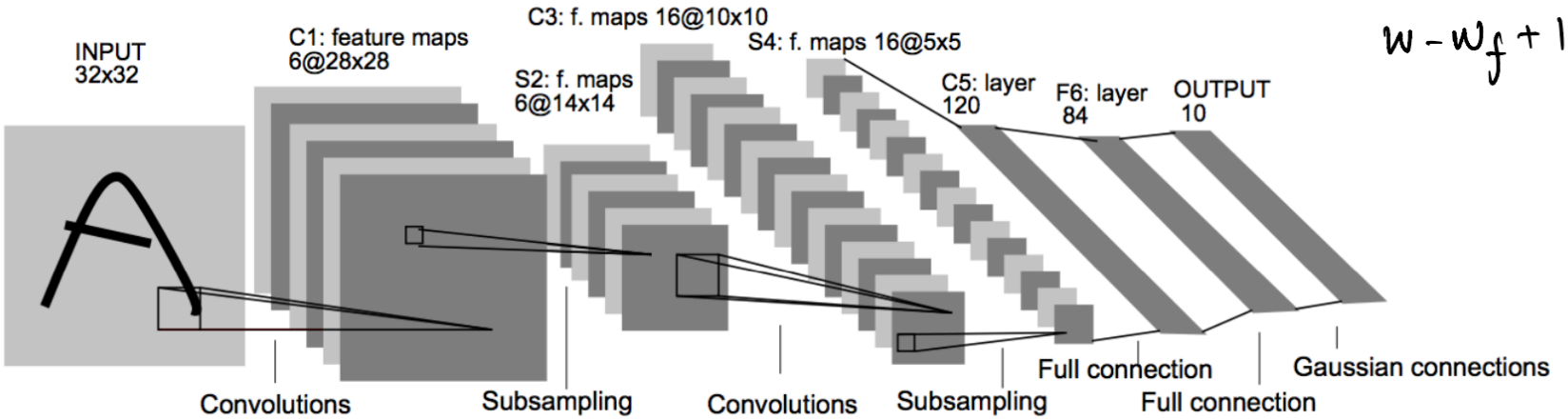
Case studies

To help get an intuition behind CNN's, we'll go over a few architectures that have been influential in recent years.

Case studies:

- LeNet (1998)
- AlexNet (2012)
- VGG (2013)
- GoogLeNet (2014)
- ResNet (2015)

Sizing examples



$W - W_f + 1$

output

C1 contains six 5x5 conv filters. stride = 1, pad = 0

Size of feature maps at C1? $32 - 5 + 1 = 28$ $(28 \times 28 \times 6)$

Number of parameters in C1 layer? $(5 \times 5 \times 1 + 1) \cdot 6 = 156$ params

S2 is a 2x2 pooling layer applied at stride 2.

Size of feature maps at S2? $(14 \times 14 \times 6)$

Number of parameters in S2 layer? 0

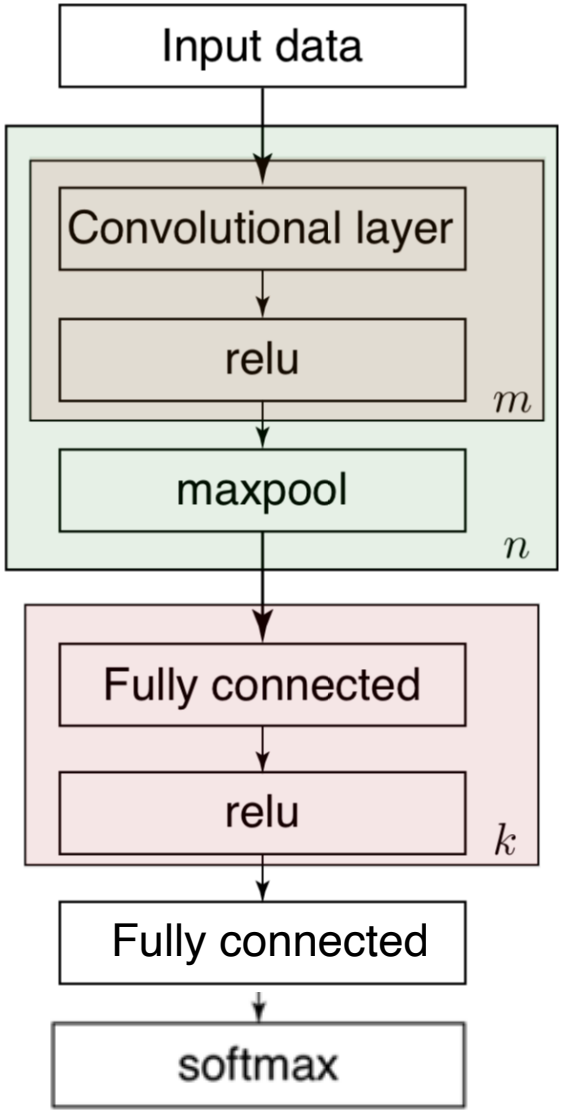
C3 contains sixteen 5x5 conv filters.

Number of parameters in C3 layer? $(5 \times 5 \times 6 + 1) \cdot 16$

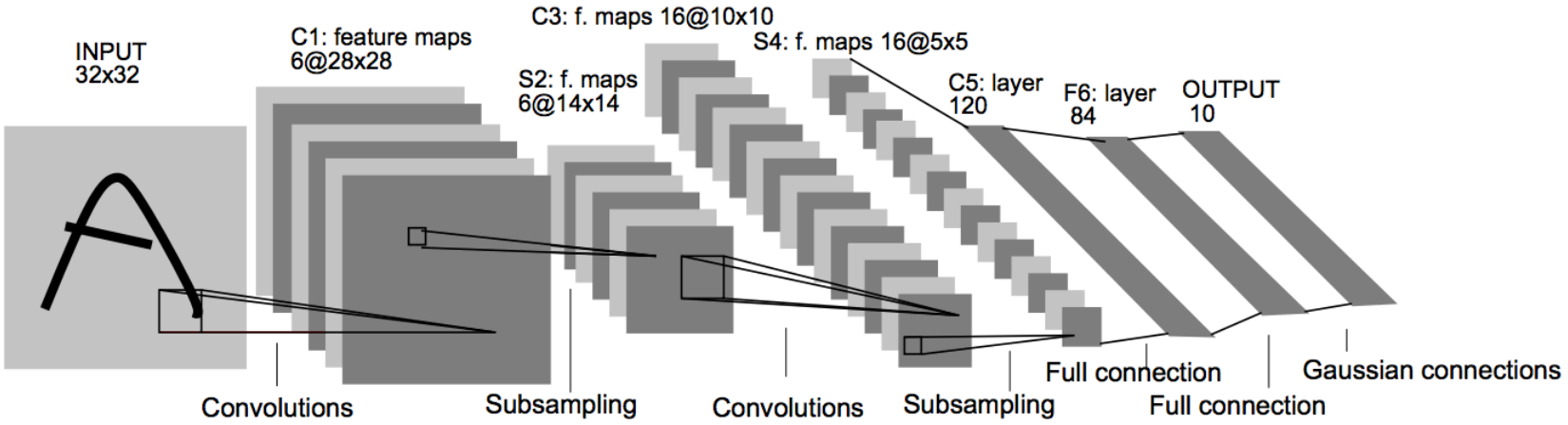
Size of feature maps at C3? $14 - 5 + 1 = 10$ $(10 \times 10 \times 16)$

Convolutional neural network architectures (cont.)

With this in mind, the following describes a fairly typical CNN architecture.



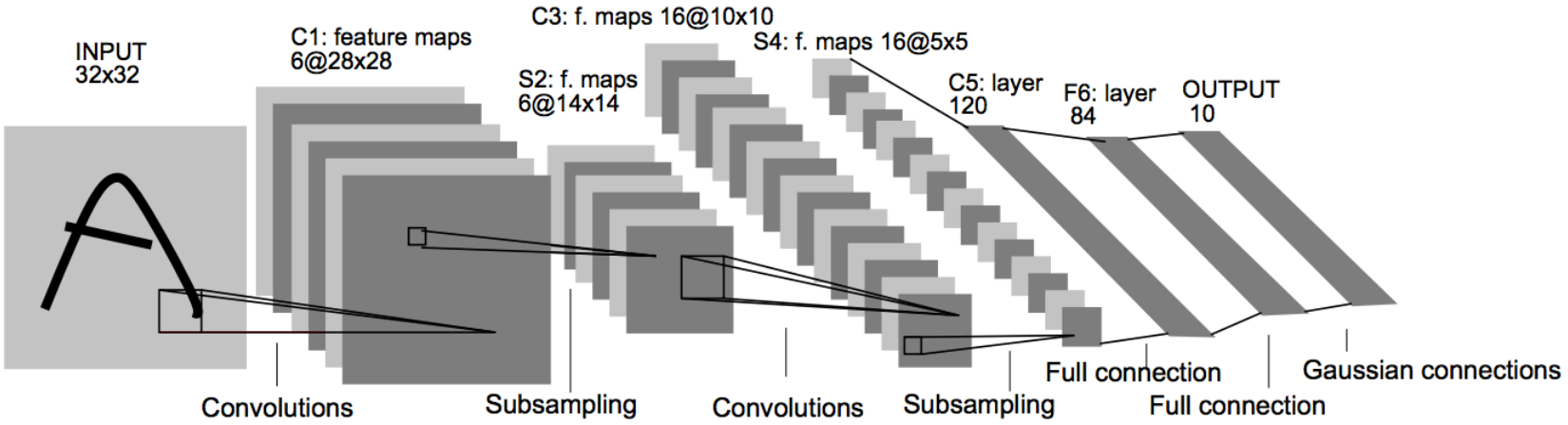
LeNet-5



4 total layers. Input is 32x32.

1. [28x28x6] **CONV**: 6 convolutional filters, 5x5 filter size, applied at stride 1.

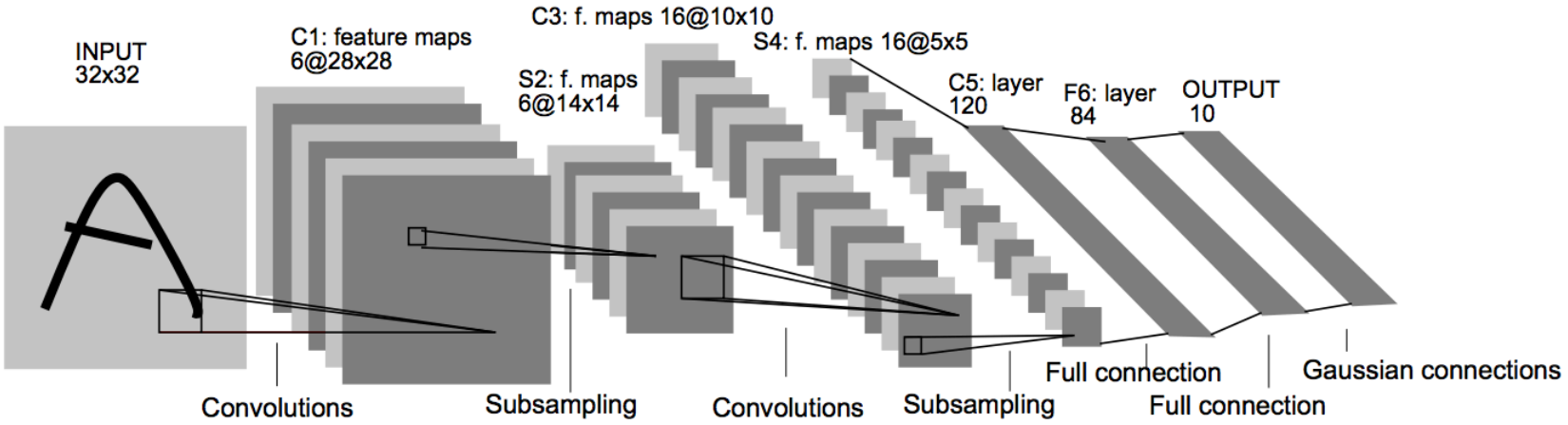
LeNet-5



4 total layers. Input is 32x32.

1. [28x28x6] **CONV**: 6 convolutional filters, 5x5 filter size, applied at stride 1.
2. [14x14x6] **POOL**: 2x2 pool with stride 2. (Adds all elems, multiplies them by trainable coefficient, then passes through sigmoid.)

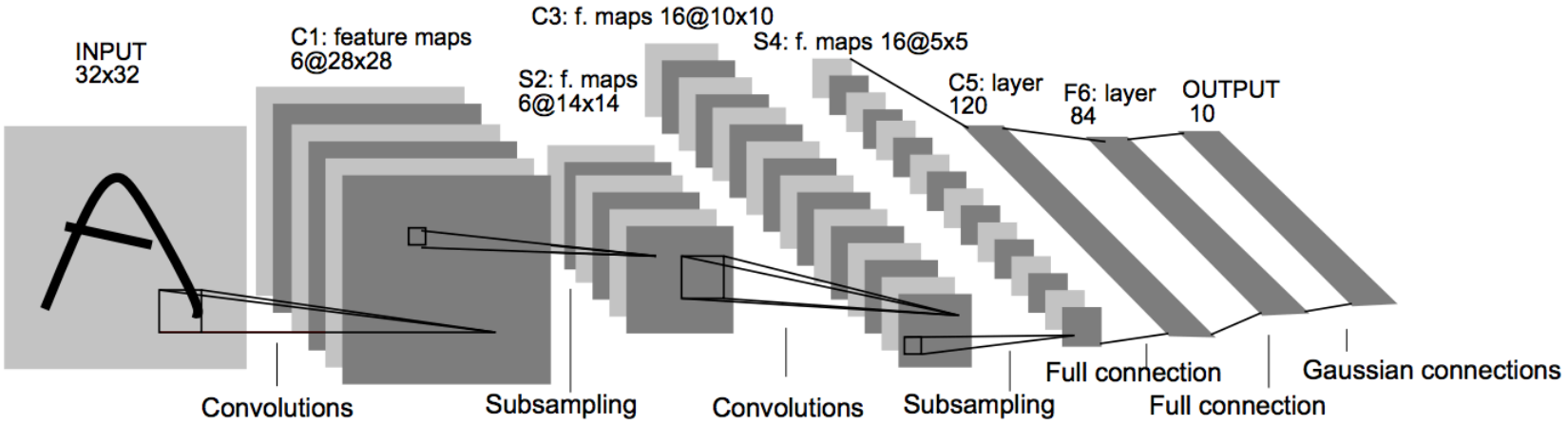
LeNet-5



4 total layers. Input is 32x32.

1. [28x28x6] **CONV**: 6 convolutional filters, 5x5 filter size, applied at stride 1.
2. [14x14x6] **POOL**: 2x2 pool with stride 2. (Adds all elems, multiplies them by trainable coefficient, then passes through sigmoid.)
3. [10x10x16] **CONV:** 16 convolutional filters, 5x5.
4. [5x5x16] **POOL**: 2x2 pool with stride 2.
5. [120] **CONV**: 120 5x5 convolutional filters.
6. [84] **FC**: FC layer: 84 x 120.
7. [10] **OUT:** MSE against a template for each digit.

LeNet-5



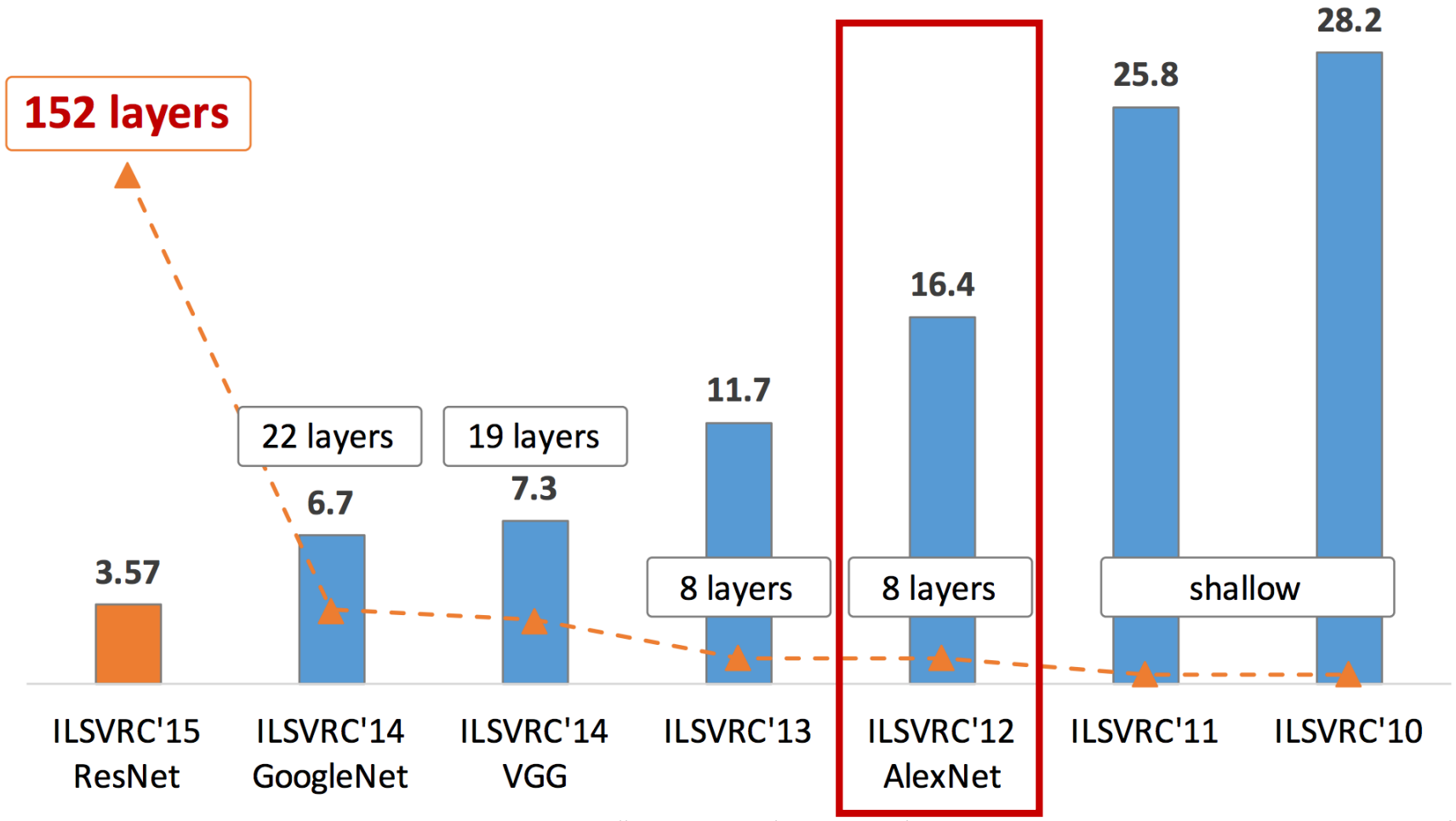
LeCun et al., 1998.

Overall architecture:

[CONV-POOL]x2 - CONV - FC - OUT

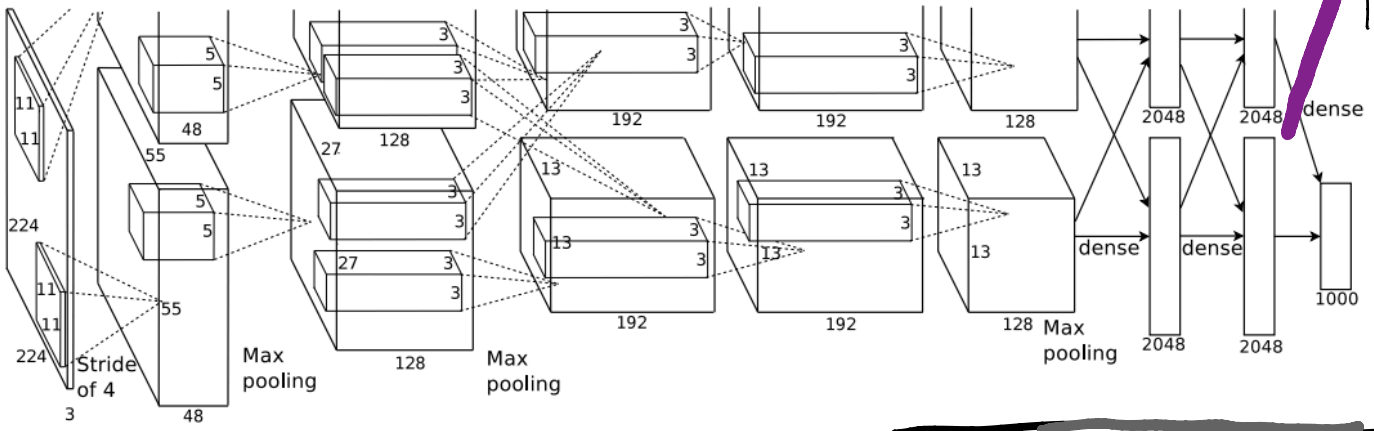
AlexNet in context

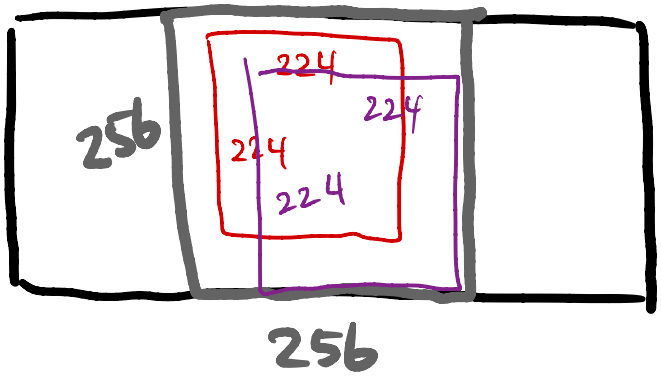
The number of layers refers to the number of convolutional or FC layers.



http://kaiminghe.com/icml16tutorial/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

AlexNet



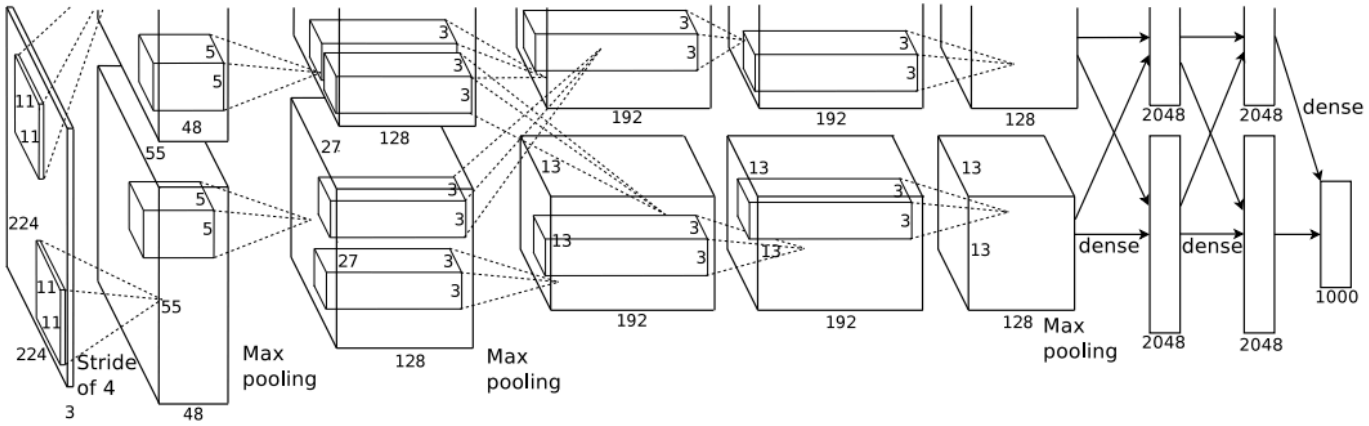


AlexNet, Krizhevsky et al., NIPS 2012.

Input processing:

- ImageNet has variable-sized images.
- Downsample or resize each image; given a rectangular image …
  - Crop so the shorter side is 256 pixels.
  - Crop out the central 256 x 256 pixels.
  - The actual input to the CNN is 224 x 224 x 3 after data augmentation.
    - However, the layer sizing doesn't quite work out, so we'll say it's 227x227x3.
- Subtracted the mean image over the training set from each pixel.

AlexNet

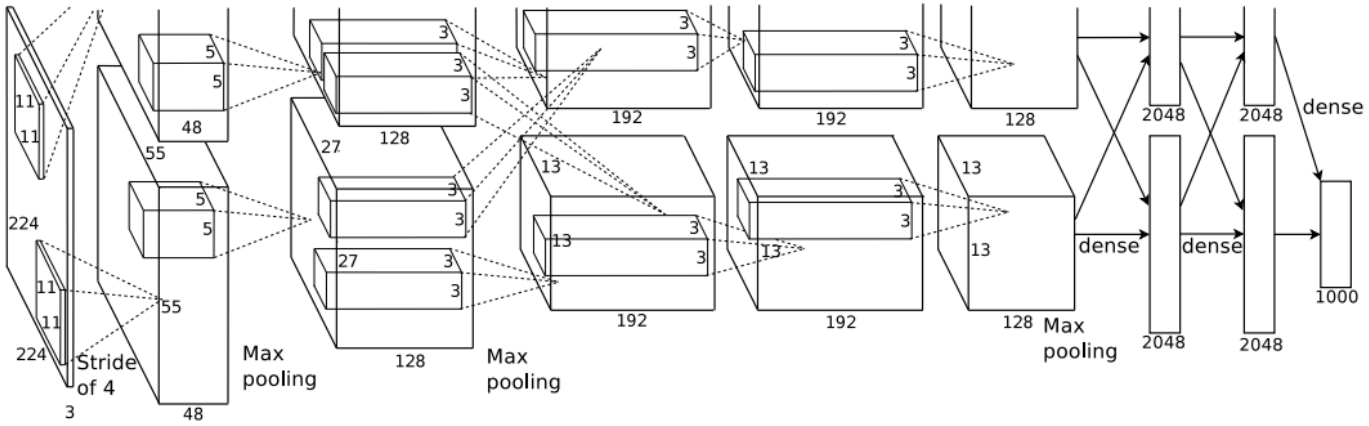


AlexNet, Krizhevsky et al., NIPS 2012.

Nonlinearity:

- Used the ReLU. It was faster than sigmoidal or tanh units.

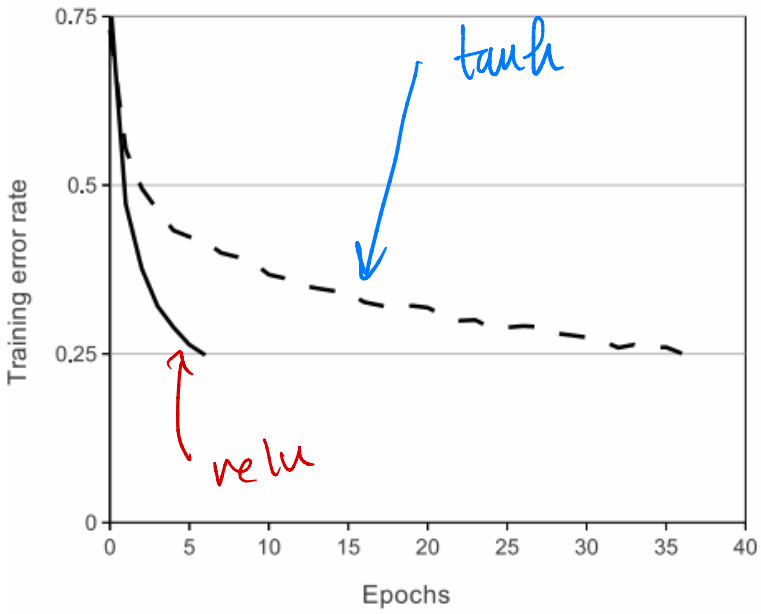
AlexNet



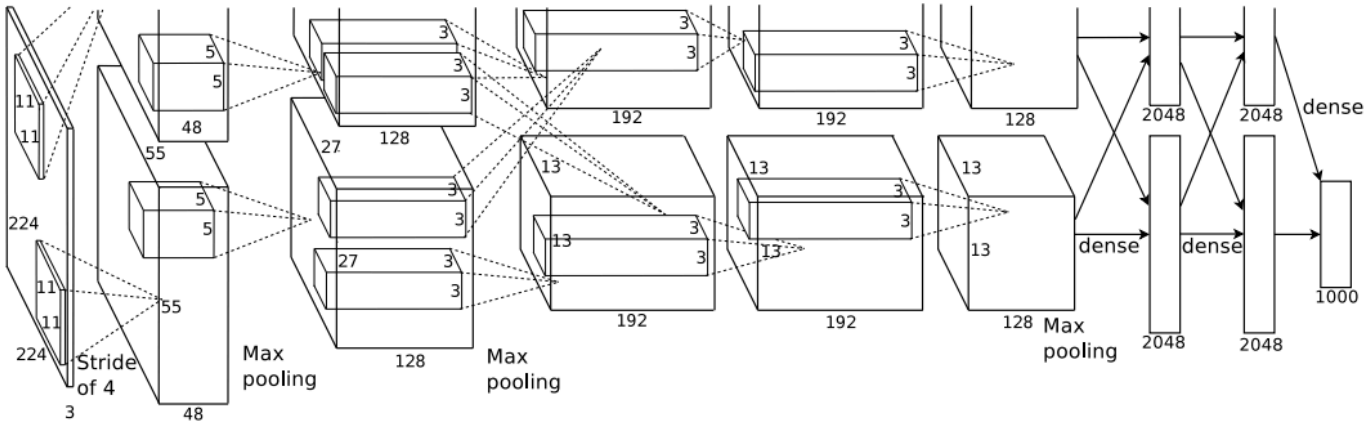
AlexNet, Krizhevsky et al., NIPS 2012.

Dotted line is tanh
Solid line is ReLU
Clearly ReLU resulted in more efficient training.

ReLU is at the output of every convolutional and fully-connected layer.



AlexNet

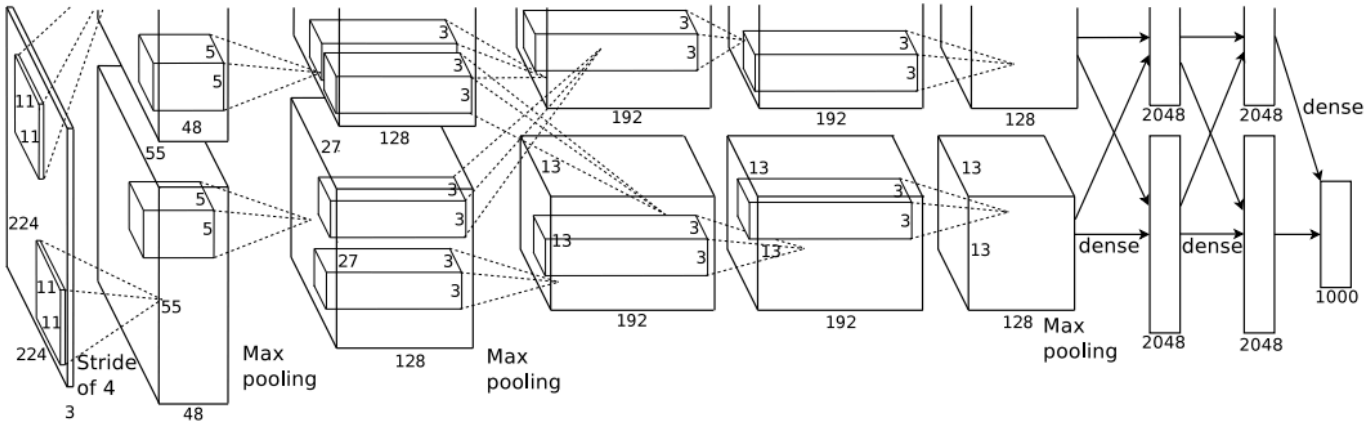


AlexNet, Krizhevsky et al., NIPS 2012.

Training on multiple GPUs.

- This is why the above image is cropped. Everything is replicated x2, and the two paths correspond to training on two GPU's.
- They trained on GPUs due to memory; they trained on 1.2 million images and they stored them on GPUs; each GPU had just 3 GB of memory.

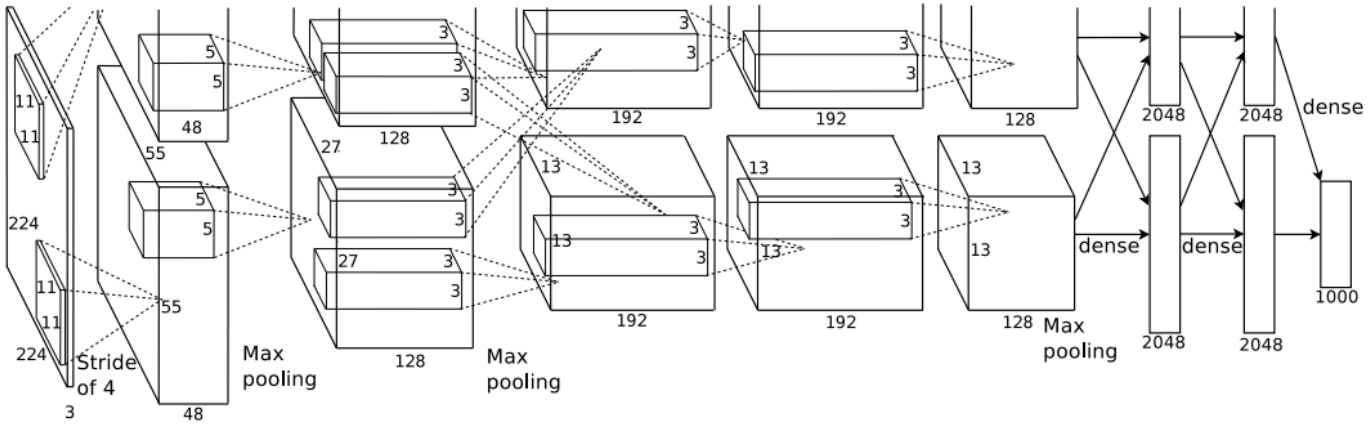
AlexNet



AlexNet, Krizhevsky et al., NIPS 2012.

- Local response normalization (not common anymore).
- Used pooling with overlapping (i.e., the stride was not the pool width).

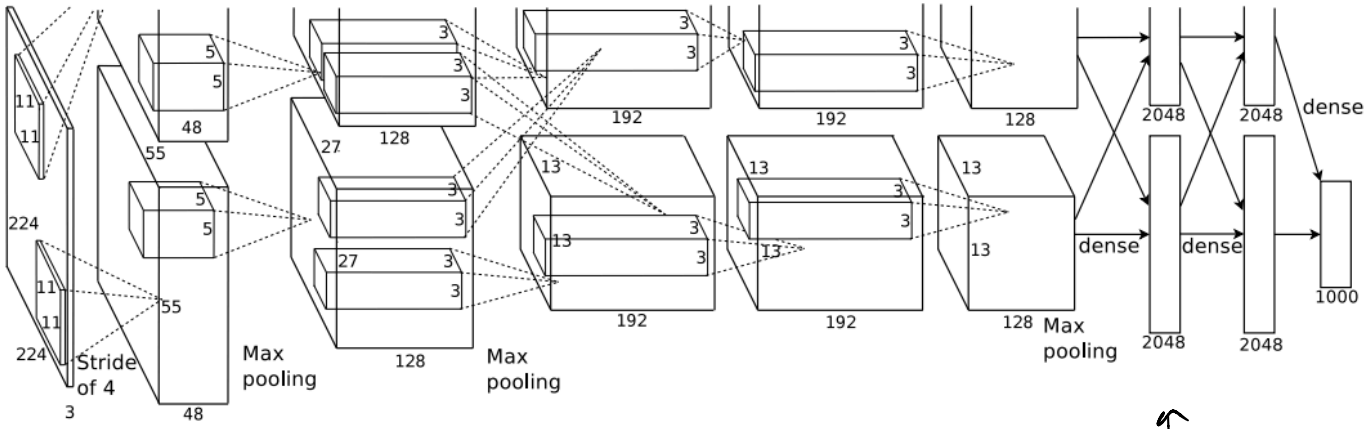
AlexNet

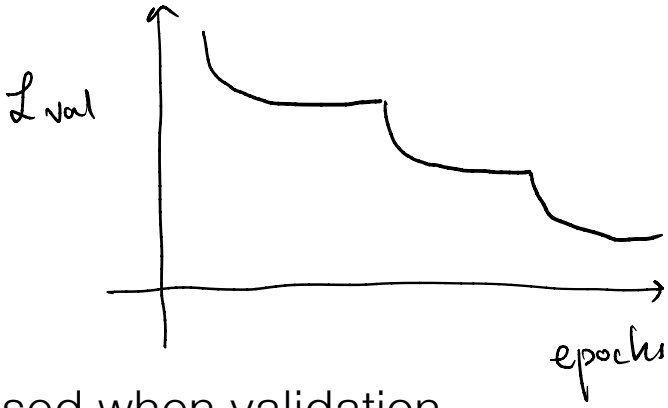


AlexNet, Krizhevsky et al., NIPS 2012.

- Data augmentation:
  - Image translations and horizontal reflections.
    - Extract out random 224 x 224 patches and their horizontal reflections.
    - At test time, extract 5 random 224 x 224 patches + reflections, and average the predictions of the 10 output softmax's. This avg'ing reduces error rate by ~1.5%.
  - Color augmentation: scale the PCs of the colors, capturing different levels of illumination and intensities.
    - Reduces the Top 1 error rate by 1%.
- Dropout with p = 0.5.
  - Substantially reduces overfitting; takes twice as long to train.

AlexNet





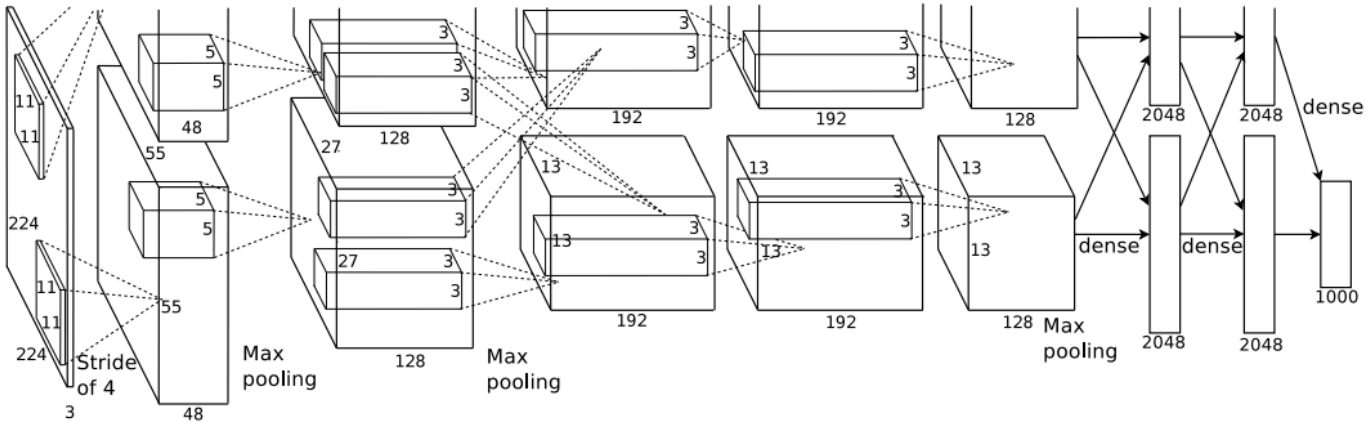
AlexNet, Krizhevsky et al., NIPS 2012.

- SGD with momentum and weight decay.
  - Batch size: 128, momentum: 0.9
  - Learning rate initialized to 0.01, manually decreased when validation error stopped improving.
  - L2 weight decay: 0.0005

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

AlexNet

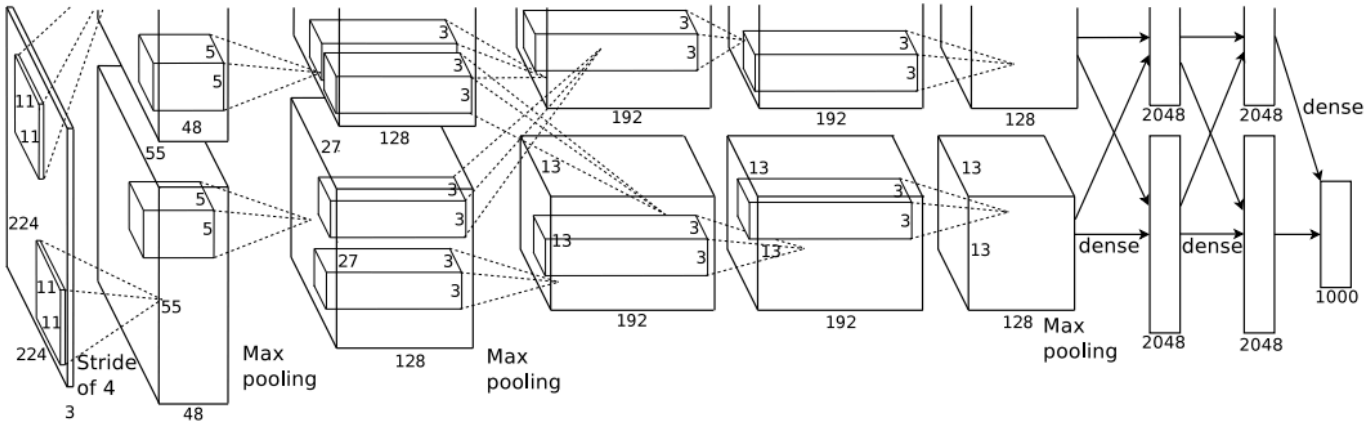


AlexNet, Krizhevsky et al., NIPS 2012.

- Training time: roughly five to six days on two GTX 580 GPUs.

reduced three times prior to termination. We trained the network for roughly 90 cycles through the training set of 1.2 million images, which took five to six days on two NVIDIA GTX 580 3GB GPUs.

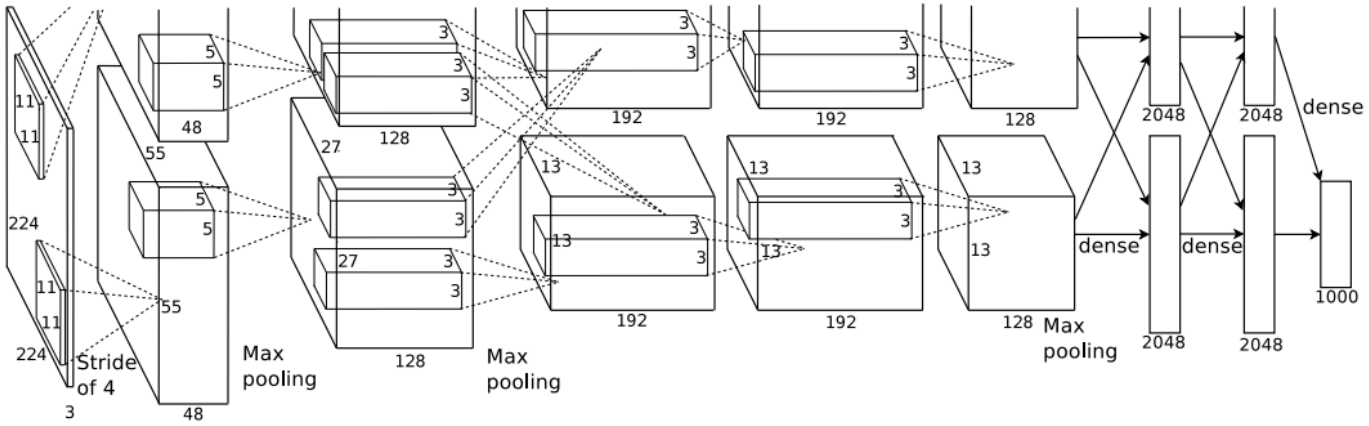
AlexNet



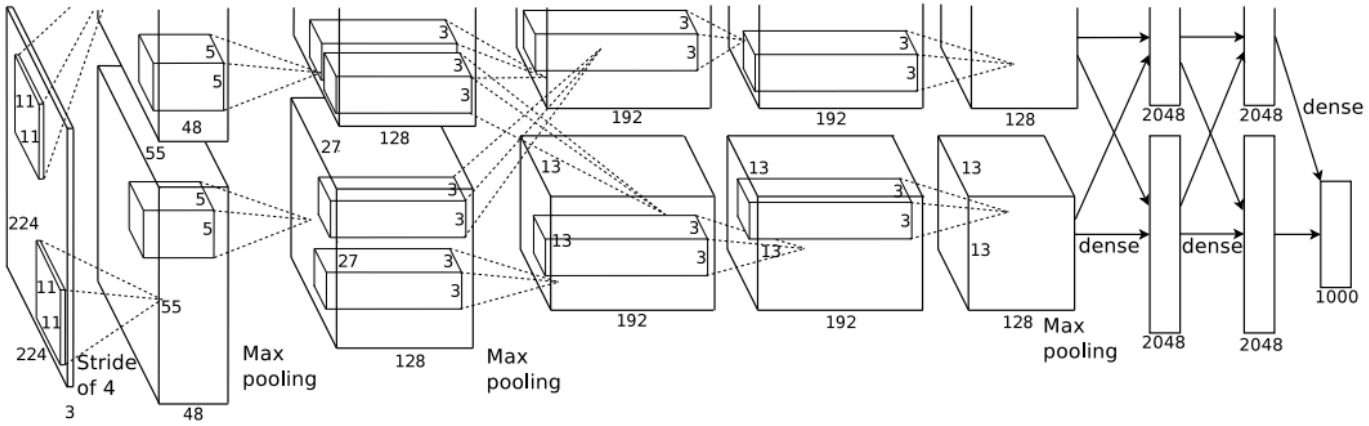
AlexNet, Krizhevsky et al., NIPS 2012.

- Averaged the output of multiple CNNs. Validation error of …
  - 1 CNN: 18.2%
  - 5 CNNs: 16.4%
  - 7 CNNs: 15.4%

AlexNet


224
224
3
11
11
Stride
of 4
55
55
48
48
5
5
Max
pooling
27
27
128
128
3
3
Max
pooling
13
13
192
192
128
Max
pooling
2048
2048
2048
2048
dense
dense
dense
1000

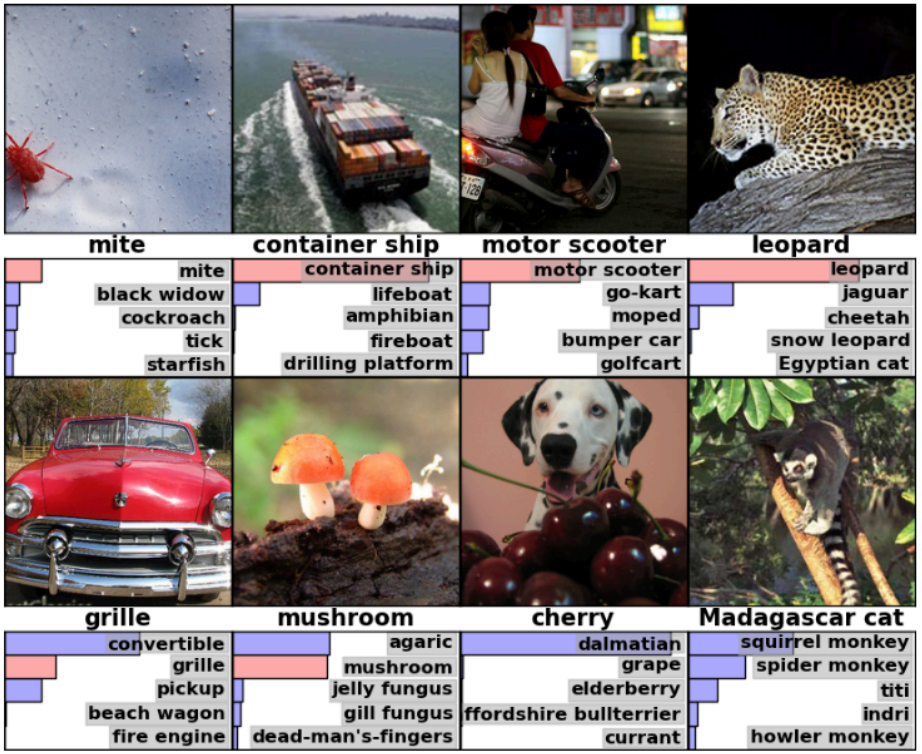
AlexNet



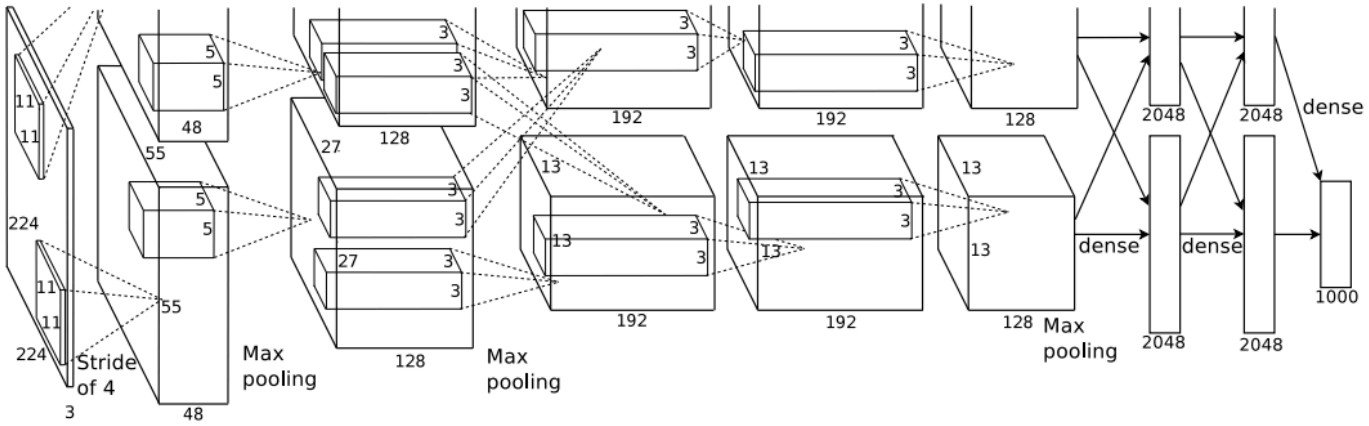
Top 5 error rate

15.4%

1000 classes



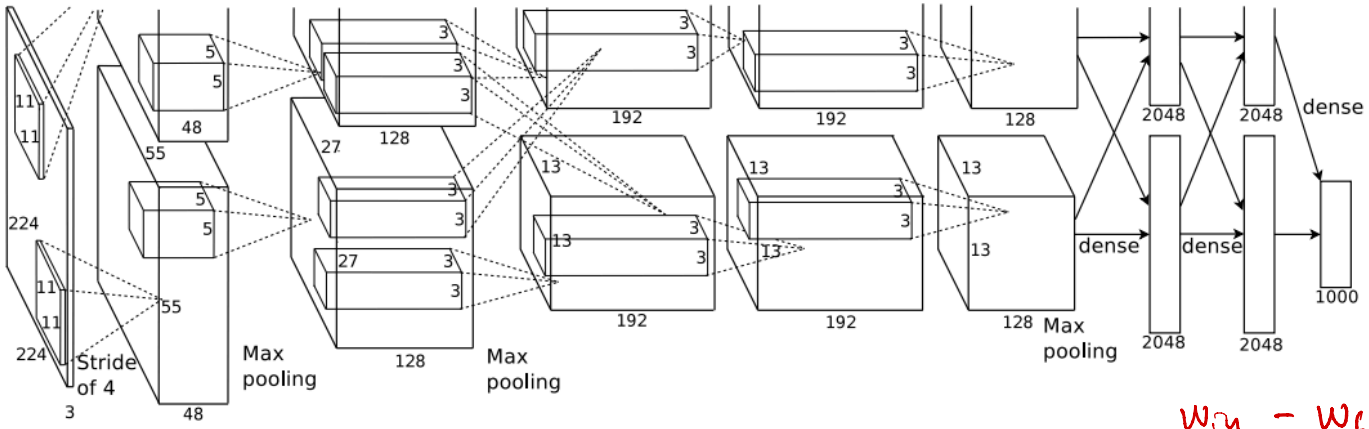
AlexNet



AlexNet, Krizhevsky et al., NIPS 2012.

- Importance of depth?
  - Validation error worsens by 2% by removing any middle layer.

AlexNet



$$W_{out} = \frac{W_{in} - W_f + 2 \cdot pad}{stride} + 1$$

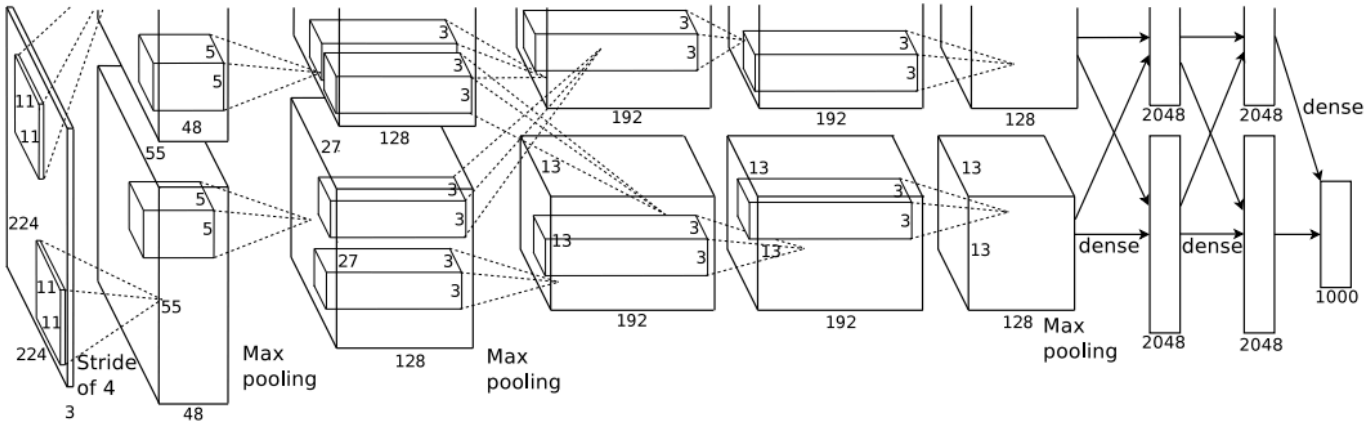
Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

Question: The input is 227x227x3. The first convolutional layer has 96 11x11 filters applied at stride 4. What is the output size?

$$\frac{227 - 11}{4} + 1 = 55$$

$(55 \times 55 \times 96)$

AlexNet

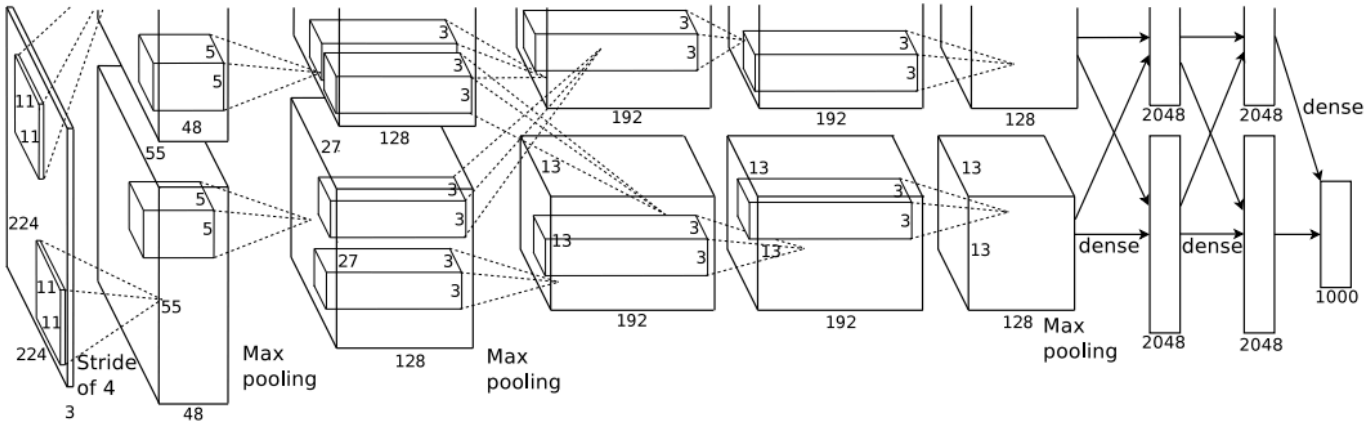


Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

Question: How many trainable parameters in the first convolutional layer? (Recall, 96 filters that are 11x11.)

$96 \cdot (11 \times 11 \times 3 + 1) = 34,944$

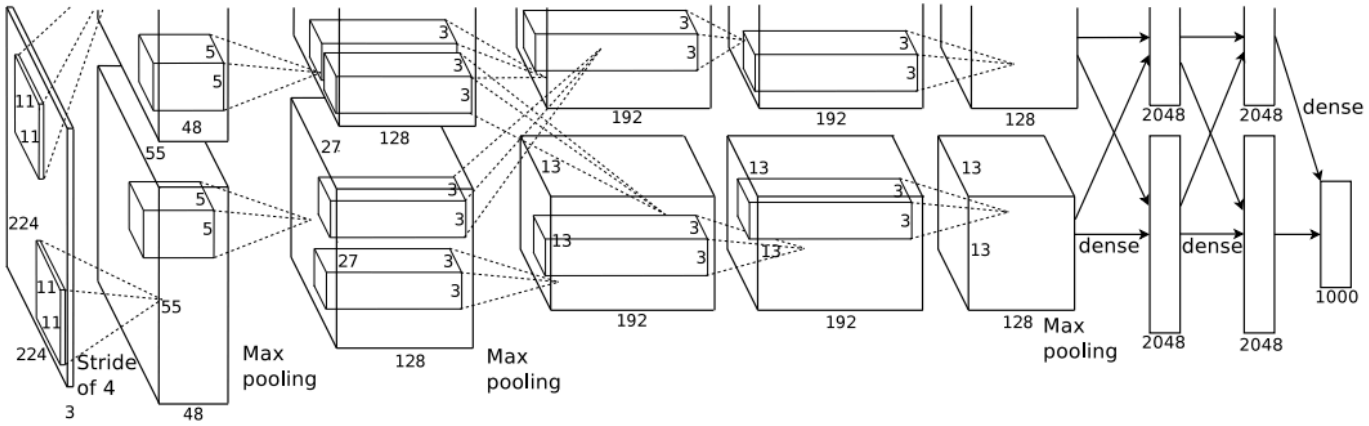
AlexNet



Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

1. [55x55x96] **CONV**: 96 filters of size 11x11x3 with stride 4.

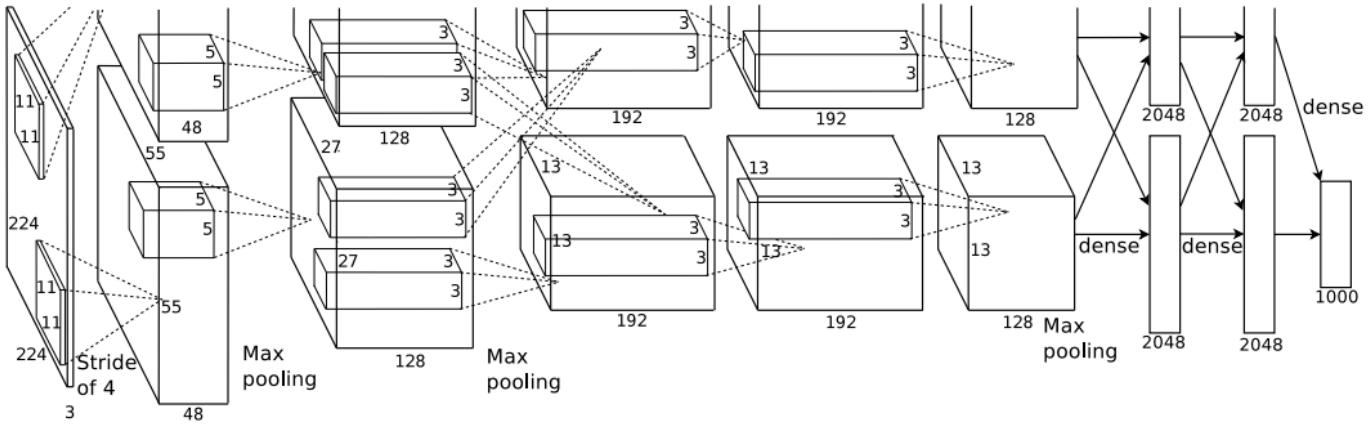
AlexNet



Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

Question: The output of the first convolutional layer is 55x55x96. The pooling layer is 3x3 filters applied at stride 2. What is the output size?

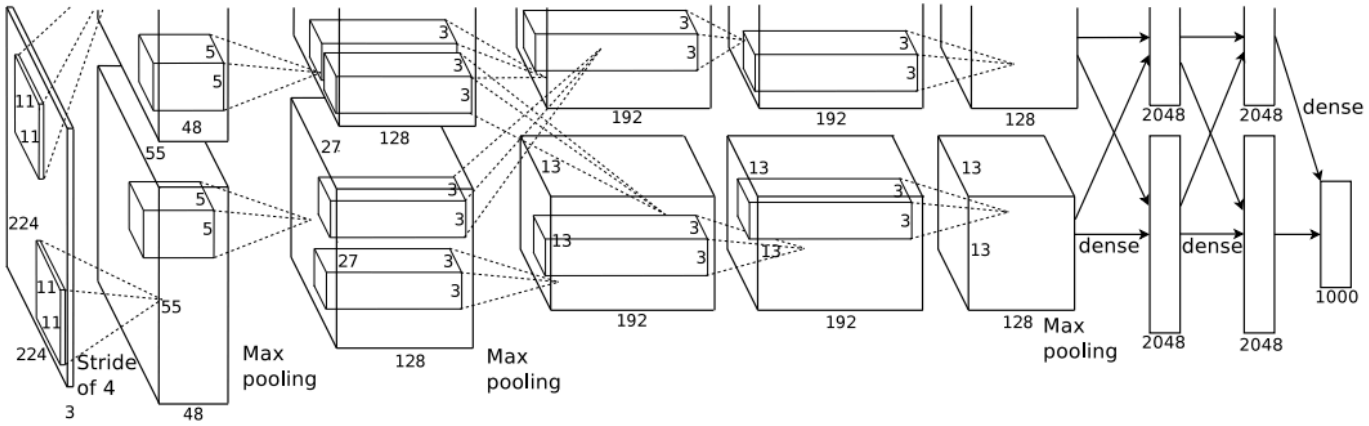
AlexNet



Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

Question: How many trainable parameters in the first pooling layer? (Recall, pool is with 3x3 filters at stride 2.)

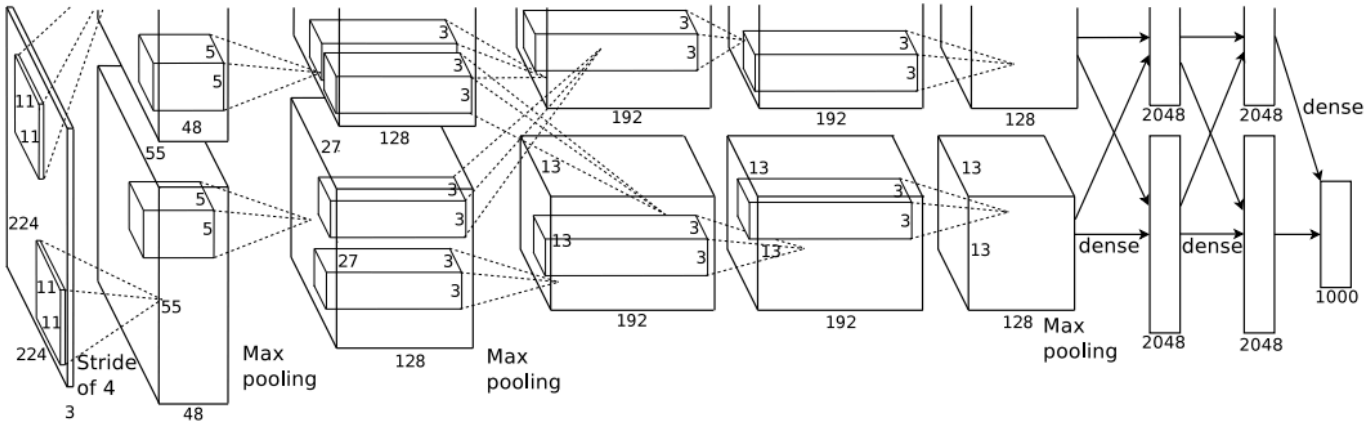
AlexNet



Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

1. [55x55x96] **CONV**: 96 filters of size 11x11x3 with stride 4.
2. [27x27x96] **POOL**: 3x3 filters with stride 2.
3. [27x27x96] **NORM**: normalization layer

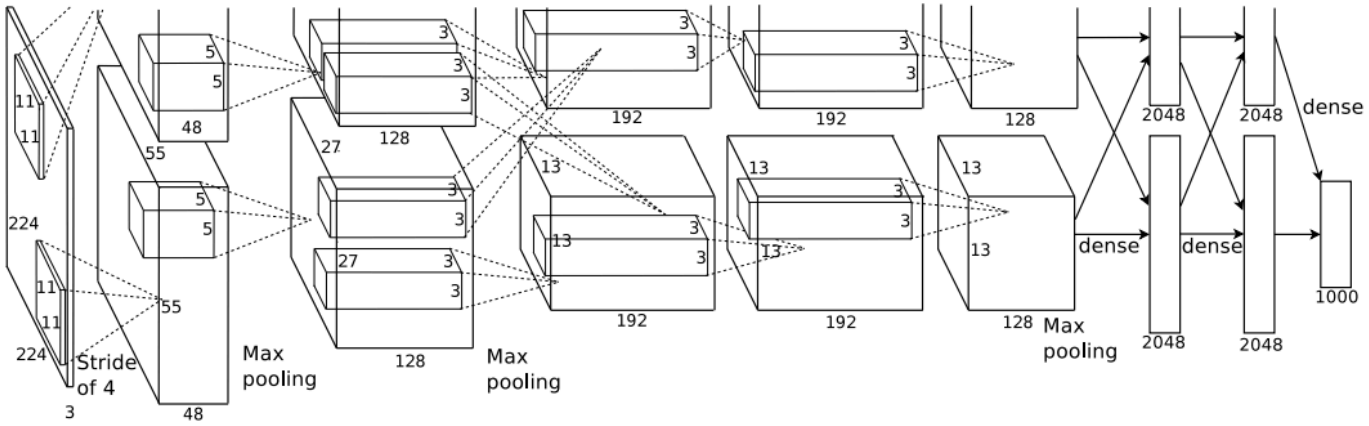
AlexNet



Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

Question: The input into the second convolutional layer is 27x27x96. The layer has 256 5x5 filters at stride 1 with pad 2. What is the output size?

AlexNet



Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

1. [55x55x96] **CONV**: 96 filters of size 11x11x3 with stride 4.
2. [27x27x96] **POOL**: 3x3 filters with stride 2.
3. [27x27x96] **NORM**: normalization layer
4. [27x27x256] **CONV**: 256 filters of size 5x5x96 with stride 1, pad 2.
5. [13x13x256] **POOL**: 3x3 filters with stride 2.

AlexNet

$W_{out} = W_{in} - W_f + 2 \cdot pad + 1$
$= 13 - 3 + 2 \cdot 1 + 1$
$= 13$

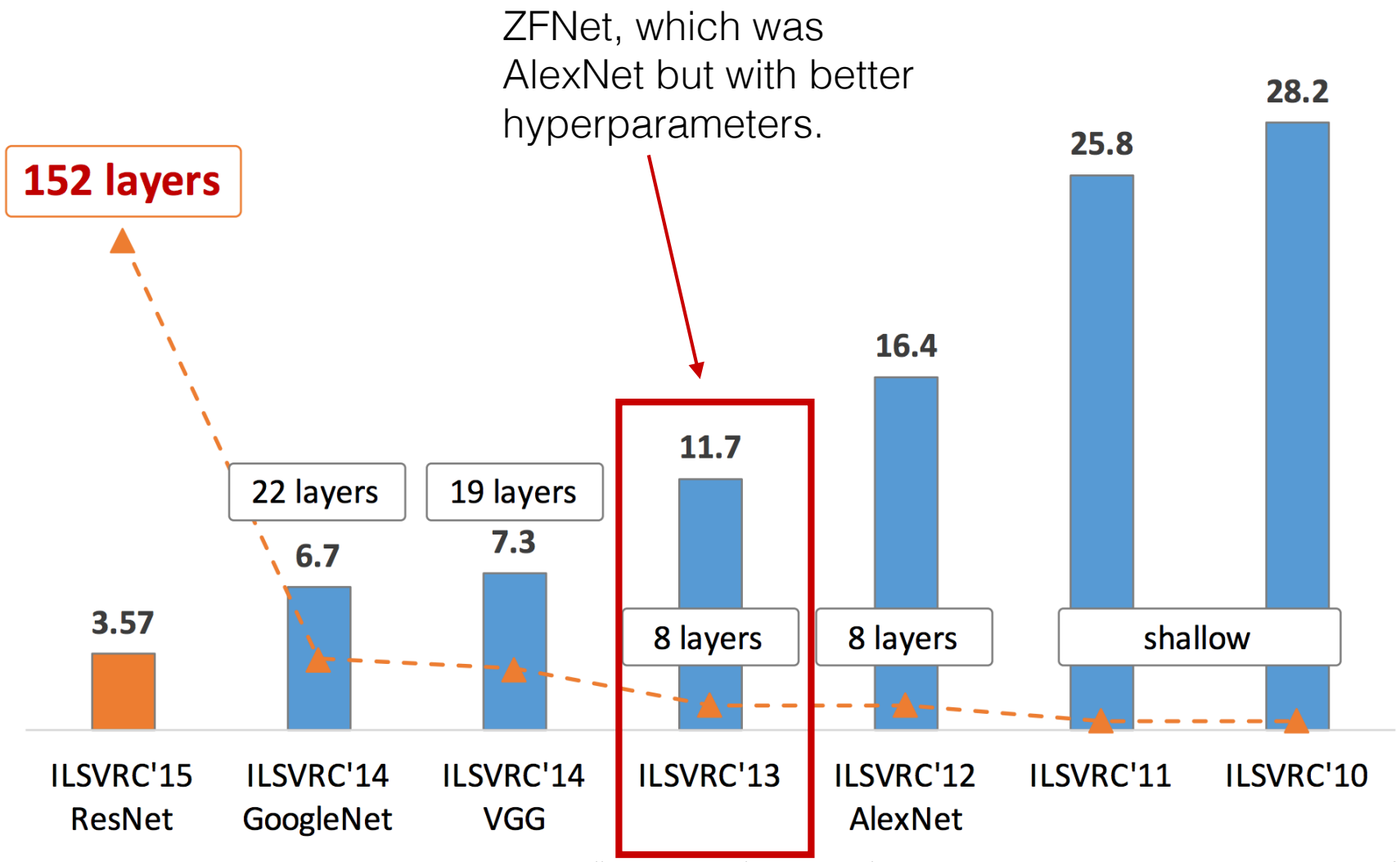
Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

1. [55x55x96] **CONV**: 96 filters of size 11x11x3 with stride 4.
2. [27x27x96] **POOL**: 3x3 filters with stride 2.
3. [27x27x96] **NORM**: normalization layer
4. [27x27x256] **CONV**: 256 filters of size 5x5x96 with stride 1, pad 2.
5. [13x13x256] **POOL**: 3x3 filters with stride 2.
6. [13x13x256] **NORM**: normalization layer
7. [13x13x384] **CONV**: 384 filters of size 3x3 at stride 1, pad 1.
8. [13x13x384] **CONV**: 384 filters of size 3x3 at stride 1, pad 1.
9. [13x13x256] **CONV**: 256 filters of size 3x3 at stride 1, pad 1.
10. [6x6x256] **POOL**: 3x3 filters at stride 2.
11. [4096] **FC**: Fully connected layer with 4096 units
12. [4096] **FC**: Fully connected layer with 4096 units
13. [1000] **FC**: Fully connected layer with 1000 units (class scores).
14. [1000] **OUT**: Softmax layer

3 x 3 x 256

2F 512 1024 512

ZFNet



http://kaiminghe.com/icml16tutorial/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

ZFNet

Zeiler & Fergus, arXiv 2013, “Visualizing and understand convolutional neural networks.”

- They introduced the deconvnet, which maps the output activations back to input pixels. This enables a visualization of features being captured by the convnets.

- With this, they made optimizations to AlexNet.

ZFNet

Differences between ZFNet and AlexNet:

- The first convolutional layer has 7x7 filters at stride 2 (AlexNet: 11x11 filters at stride 4).
- The third, fourth, and fifth convolutional layers have 512, 1024, and 512 filters (AlexNet: 384, 384, 256).

(Fig. 6(b) & (d)), various problems are apparent. The first layer filters are a mix of extremely high and low frequency information, with little coverage of the mid frequencies. Additionally, the 2nd layer visualization shows aliasing artifacts caused by the large stride 4 used in the 1st layer convolutions. To remedy these problems, we (i) reduced the 1st layer filter size from 11x11 to 7x7 and (ii) made the stride of the convolution 2, rather than 4. This new architecture retains

ZFNet

Differences between ZFNet and AlexNet:

- The first convolutional layer has 7x7 filters at stride 2 (AlexNet: 11x11 filters at stride 4).
- The third, fourth, and fifth convolutional layers have 512, 1024, and 512 filters (AlexNet: 384, 384, 256).

(Fig. 6(b) & (d)), various problems are apparent. The first layer filters are a mix of extremely high and low frequency information, with little coverage of the mid frequencies. Additionally, the 2nd layer visualization shows aliasing artifacts caused by the large stride 4 used in the 1st layer convolutions. To remedy these problems, we (i) reduced the 1st layer filter size from 11x11 to 7x7 and (ii) made the stride of the convolution 2, rather than 4. This new architecture retains

How big are these differences?

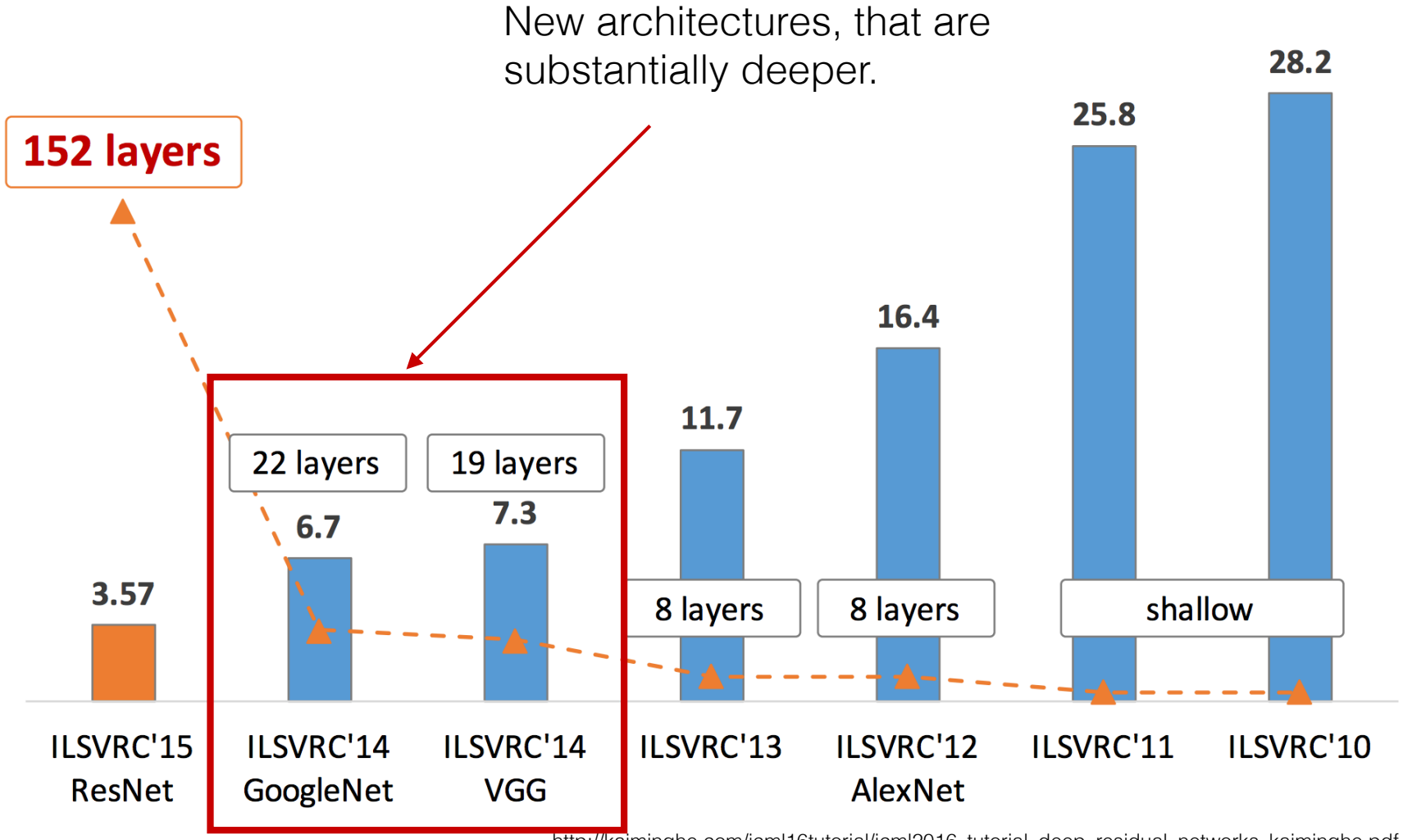
- Big!
- Dropped error rate from **16.4%** to **11.7%**.
- Measuring relative to zero error, this is approximately a 30% reduction in error.

What makes convolutional neural networks work better?

Some observations from ZFNet are:

- Smaller filters applied at smaller strides appears to help (at least in early layers).
- Having more filters later on in deeper layers appears to help.

What about depth?



http://kaiminghe.com/icml16tutorial/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

VGGNet

From the Visual Geometry Group, Dept of Eng. Sci., Oxford, “Very Deep Convolutional Neural Networks for Large-Scale Image Recognition,” Simonyan & Zisserman, arXiv 2014.

“Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers.”

Their approach: focus on a small convolutional filter (3x3) and extend the depth.

VGGNet

VGG Net:

Instead of 8 layers (AlexNet), VGGNet increased the network architecture to 16-19 layers.

ARCHITECTURE:

Input -> [CONVx2-POOL]x3 -> [CONVx3-POOL]x2 -> FC x 3 -> Softmax

All CONV filters are uniform: 3x3 with stride 1, pad 1
All POOL filters are uniform: 2x2 max pool with stride 2.

Reduction from **11.7%** to **7.3%**, approximately a 40% reduction in error rate.

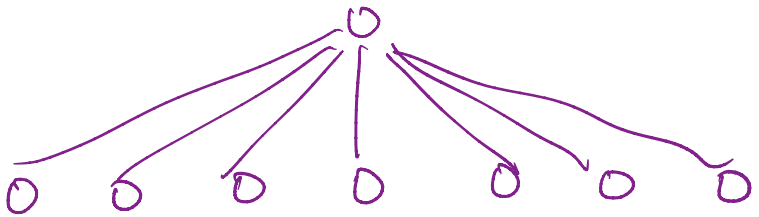
VGGNet

Small filters and depth:

What might be a con of using a small filter, and how does VGGNet address this? (Think receptive fields.)

ZFNet

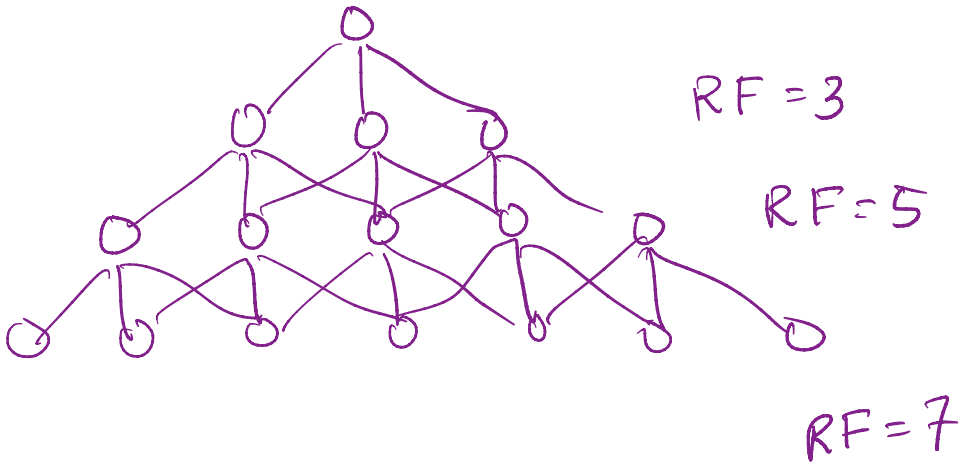
7x7



RF = 7

VGG Net

3x3



VGGNet

Small filters and depth:

Which has more parameters? One 7x7 CONV layer or three 3x3 stacked CONV layers?

ZF Net 7 x 7

input depth : $C_{in} = C$

filters : $C_{out} = C$

$(7 \times 7 \times C_{in}) \cdot C_{out}$

$49 \cdot C^2$

VGG Net

stack of three 3 x 3 filters

$((3 \times 3 \times C_{in}) \cdot C_{out}) \cdot 3$

$27 \cdot C^2$

VGGNet

Small filters and depth:

Why might stacking many 3x3 filters turn into a good thing?

"more nonlinearity"

(1 relu ZF vs. 3 relus for VGG)

VGGNet

Small filters and depth:

What is a potential con of using small filters and more layers?

ZFNet

one 7x7 filter, pad = 1

$W_{out} = W_{in} - 7 + 2 + 1$

$= W_{in} - 4$

$(W_{in} - 4, H_{in} - 4)$

VGG

three 3x3 filters, pad = 1

(W_{in}, H_{in})

(W_{in}, H_{in})

(W_{in}, H_{in})

VGGNet

INPUT	[224x224x3]
CONV (64)	[224x224x64]
CONV (64)	[224x224x64]
POOL	[112x112x64]
CONV (128)	[112x112x128]
CONV (128)	[112x112x128]
POOL	[56x56x128]
CONV (256)	[56x56x256]
CONV (256)	[56x56x256]
CONV (256)	[56x56x256]
POOL	[28x28x256]
CONV (512)	[28x28x512]
CONV (512)	[28x28x512]
CONV (512)	[28x28x512]
POOL	[14x14x512]
CONV (512)	[14x14x512]
CONV (512)	[14x14x512]
CONV (512)	[14x14x512]
POOL	[7x7x512]
FC	[1x1x4096]
FC	[1x1x4096]
FC	[1x1x1000]

64 filters, each $3 \times 3 \times 3$, pad = 1

64 filters, each $3 \times 3 \times 64$, pad = 1

ops : # neurons · (# ops / neuron)

$(112 \cdot 112 \cdot 128)(3 \cdot 3 \cdot 64)$

#ops : $(56 \cdot 56 \cdot 256)(3 \cdot 3 \cdot 128)$

VGGNet

INPUT	[224x224x3]	224*224*3	~ 150K
CONV (64)	[224x224x64]	224*224*64	~ 3.2M
CONV (64)	[224x224x64]	224*224*64	~ 3.2M
POOL	[112x112x64]	112*112*64	~ 800K
CONV (128)	[112x112x128]	112*112*128	~ 1.6M
CONV (128)	[112x112x128]	112*112*128	~ 1.6M
POOL	[56x56x128]	56*56*128	~ 400K
CONV (256)	[56x56x256]	56*56*256	~ 800K
CONV (256)	[56x56x256]	56*56*256	~ 800K
CONV (256)	[56x56x256]	56*56*256	~ 800K
POOL	[28x28x256]	28*28*256	~ 200K
CONV (512)	[28x28x512]	28*28*512	~ 400K
CONV (512)	[28x28x512]	28*28*512	~ 400K
CONV (512)	[28x28x512]	28*28*512	~ 400K
POOL	[14x14x512]	14*14*512	~ 100K
CONV (512)	[14x14x512]	14*14*512	~ 100K
CONV (512)	[14x14x512]	14*14*512	~ 100K
CONV (512)	[14x14x512]	14*14*512	~ 100K
POOL	[7x7x512]	7*7*512	~ 25K
FC	[1x1x4096]		4096
FC	[1x1x4096]		4096
FC	[1x1x1000]		1000

Every activation is 4B

24 M × 4 Bytes

= 96 MBytes

VGGNet

IGNORE BIASES

params

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

VGGNet : 138 M params

FC layers : 122 M

VGGNet

Some observations:

- Memory: 24M * 4 bytes = 96MB for one forward pass.
- Total parameters: 138M parameters
- A lot of the network parameters are in the fully connected layer.

VGGNet

Number of layers

A - 11
B - 13
C - 16
D - 16
E - 19

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

Simonyan et al., arXiv 2014

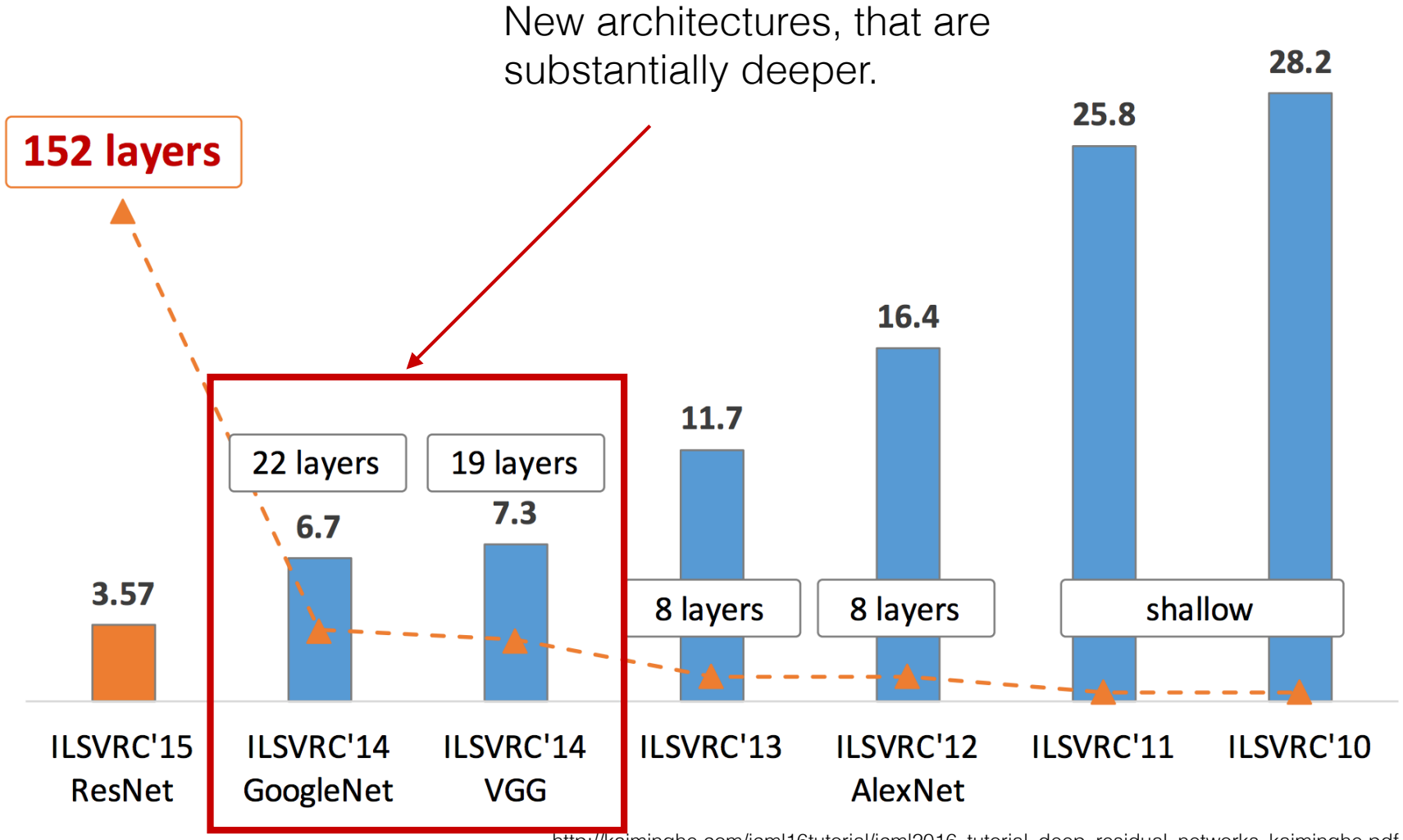
Difference between C & D: C had three 1x1 conv layers.

VGGNet

Other implementation notes about VGGNet.

- Input is 224x224 RGB image that has global mean-subtraction.
- They disposed of the local response normalization (LRN) layers from AlexNet, as they found they did not increase performance but consumed more memory & computation.
- Batch size 256, SGD + momentum 0.9.
- L2 penalty of 5e-4.
- Dropout for first two FC layers.
- Learning rate adjusted as in AlexNet.
- In initialization, they trained a shallower network and then used its weights as the initial weights for deeper networks.
  - But later on they found out the Xavier initialization performed comparably.
- Also performed the horizontal flipping, random crops, and RGB shifting that AlexNet and others used.
- Training took 2-3 weeks on a 4 GPU machine.
- Their submission averaged the output of 7 nets.

What about depth?



http://kaiminghe.com/icml16tutorial/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

GoogLeNet

From Szegedy et al., IEEE CVPR 2014.

The main take-home points of the GoogLeNet:

(1) Lets the network pick out / optimize for the most important features

(2) Reduce computational expense

- 22 layers (deeper)
- Introduces the "Inception" module
- Gets rid of fully connected layers.
- Has only 5 million parameters, which is about 12x less than AlexNet and 27x less than VGGNet.
- Also tries to keep computational budget down.
  - "… so that the [sic] they do not end up to be a purely academic curiosity…"
- Won ImageNet top 5 error (w/ error rate 6.7%).

GoogLeNet

Going deeper requires more parameters, and more computational expense.

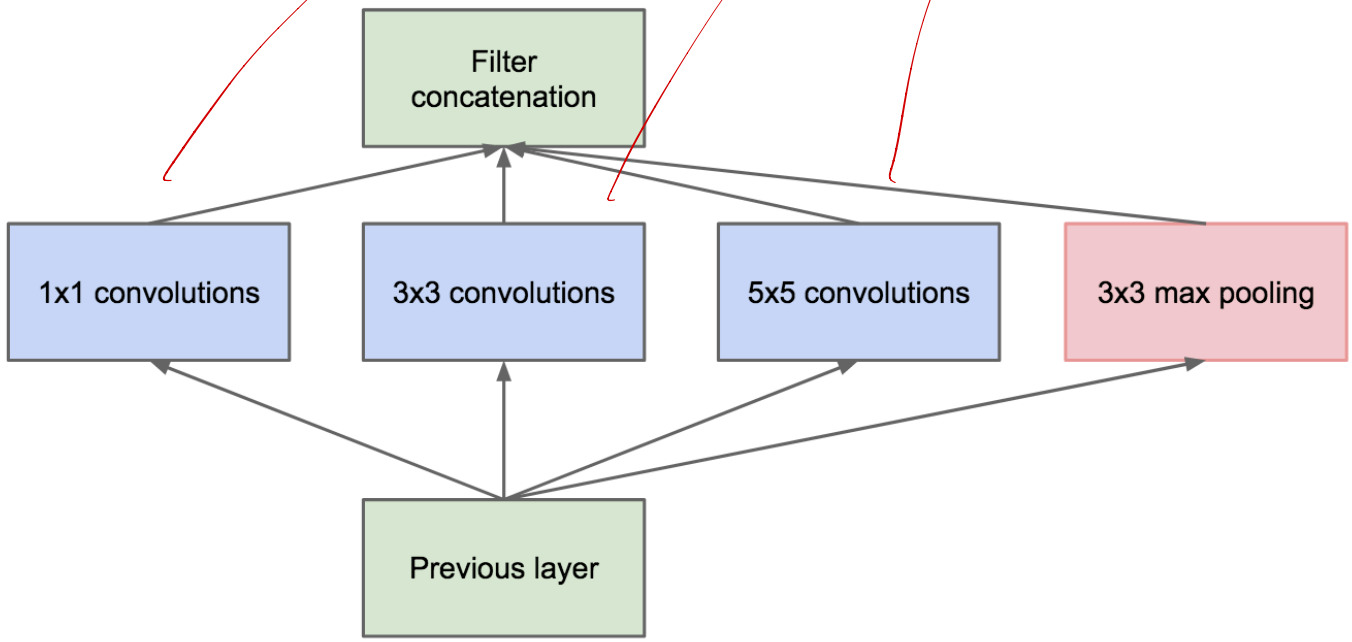
Is there a way to address this?

GoogLeNet: the inception module.

GoogLeNet

They leverage an idea called “network-in-network.”

Naive inception module:



extract diff. features of the data

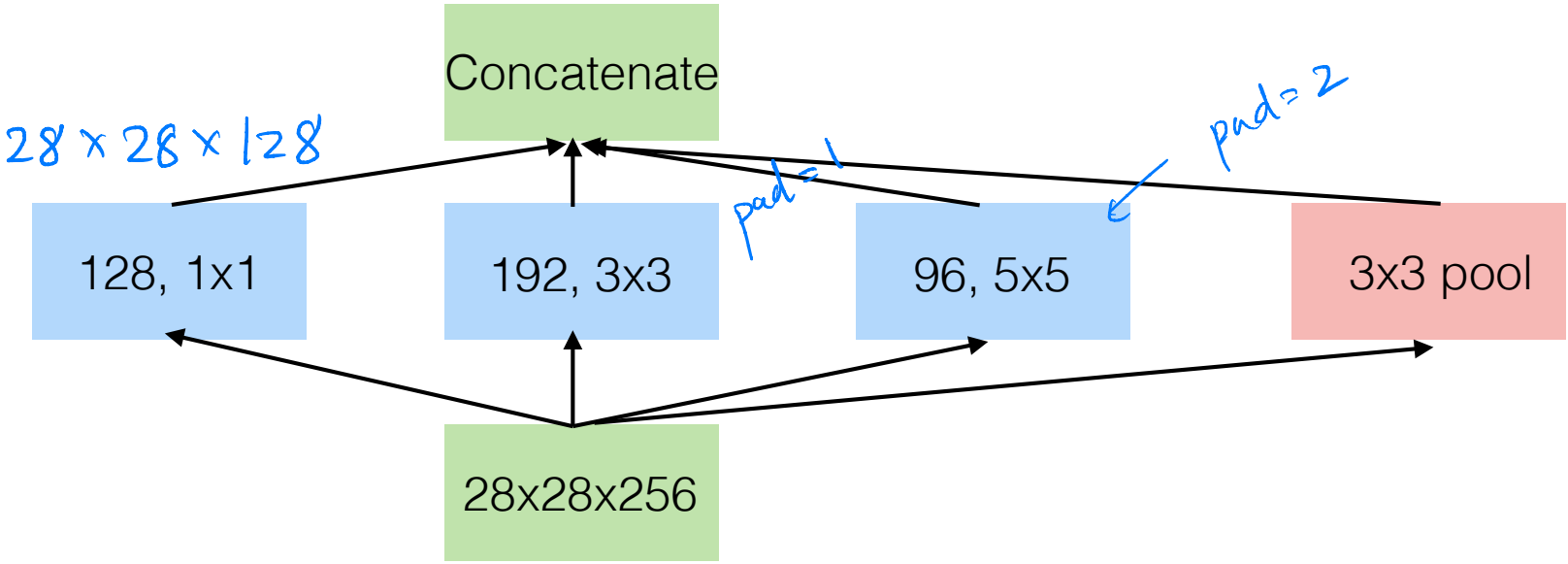
inception module

GoogLeNet

They leverage an idea called “network-in-network.”

Naive inception module:

$$W_{out} = W_{in} - W_f + 2 \cdot pad + 1$$
$$= 28 - 1 + 0 + 1$$
$$= 28$$

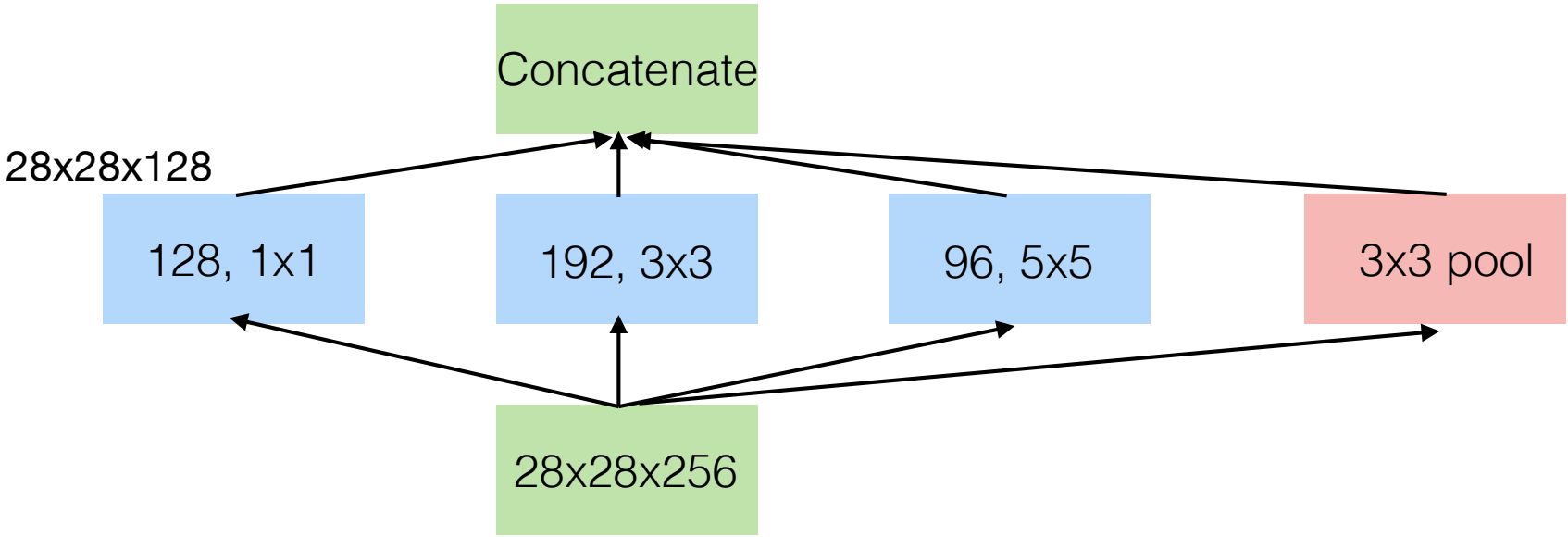


What is the size of the output of the 128 1x1 convolutions?

GoogLeNet

They leverage an idea called “network-in-network.”

Naive inception module:

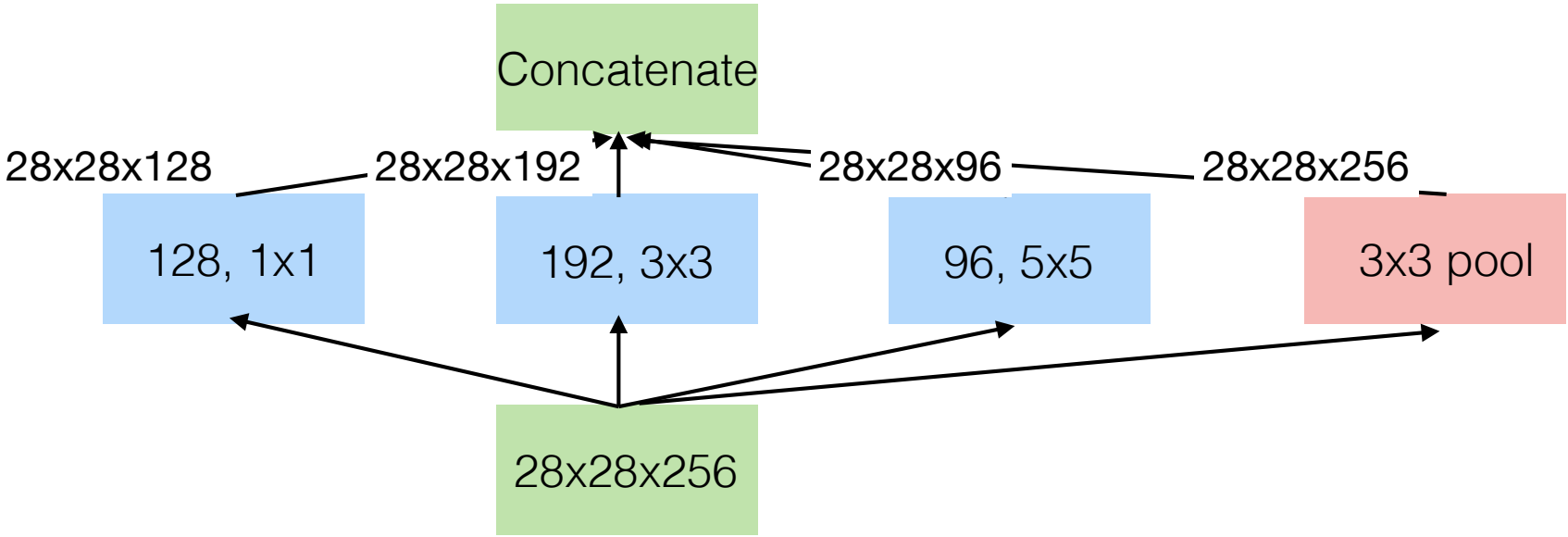


What padding do we need to keep the output size consistent for the 192 3x3 convolutional filters?

GoogLeNet

They leverage an idea called “network-in-network.”

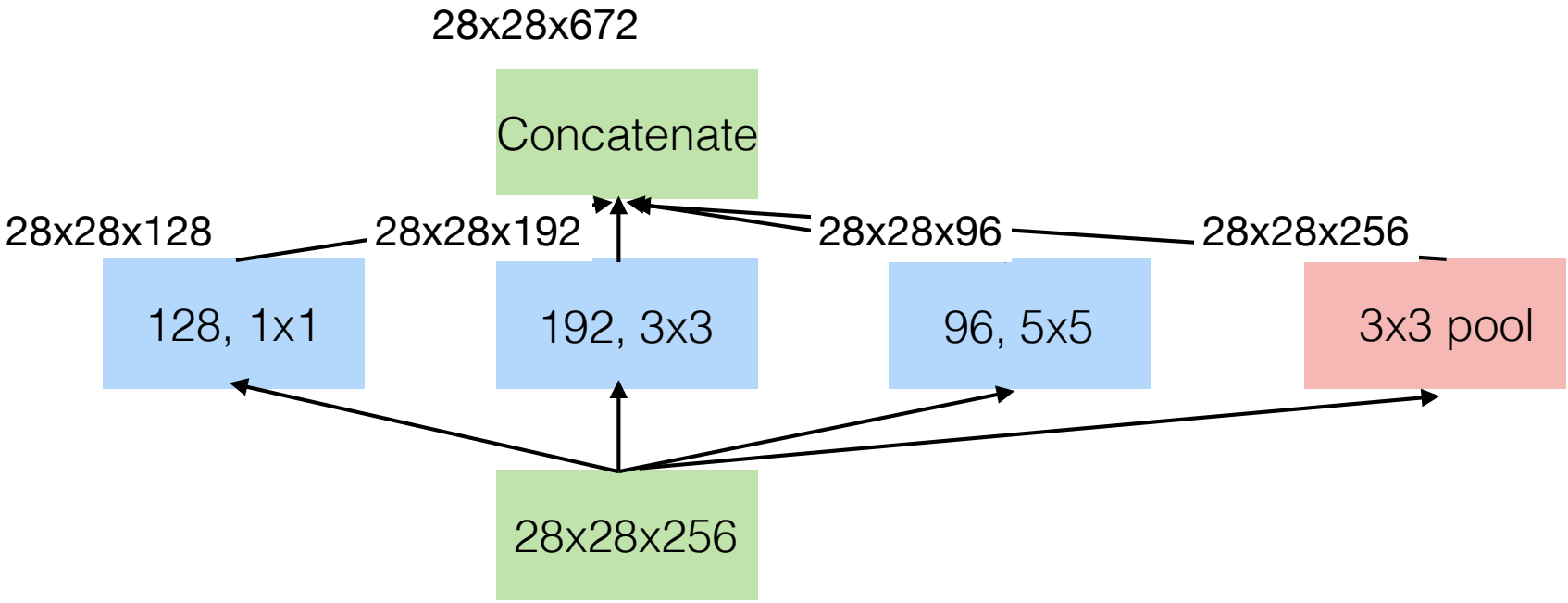
Naive inception module:



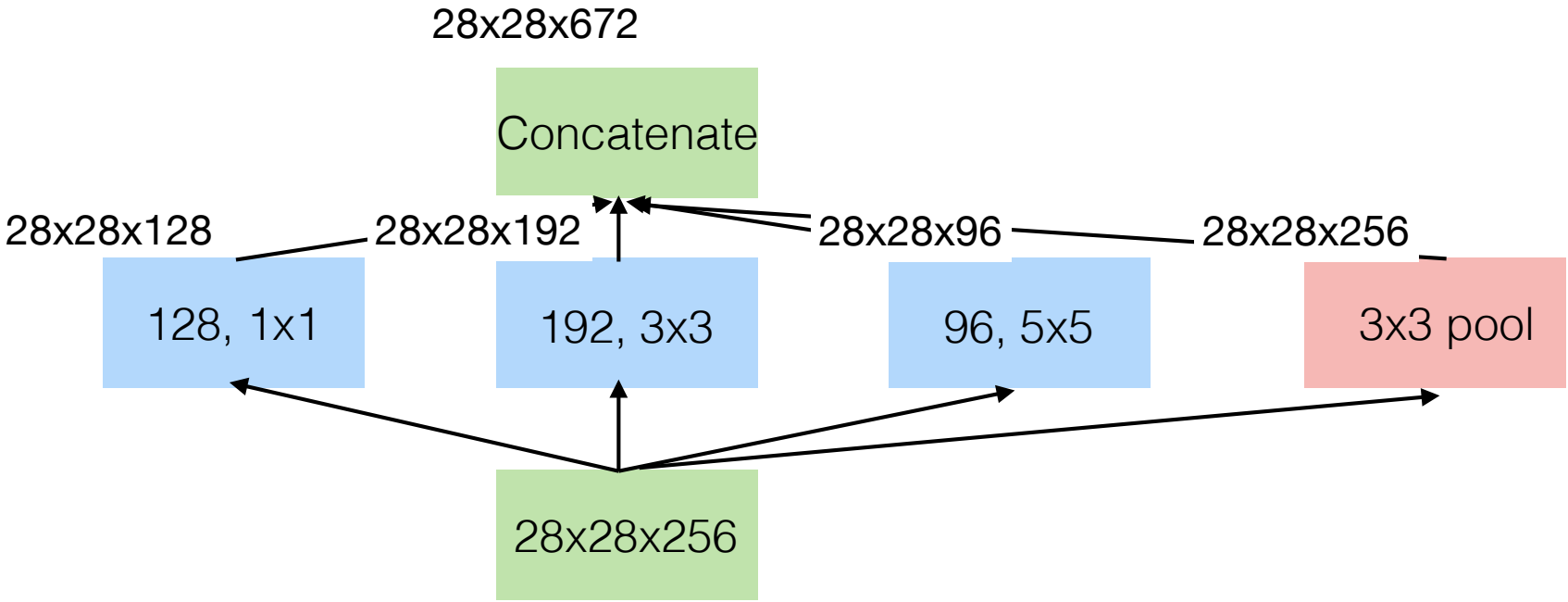
GoogLeNet

They leverage an idea called “network-in-network.”

Naive inception module:



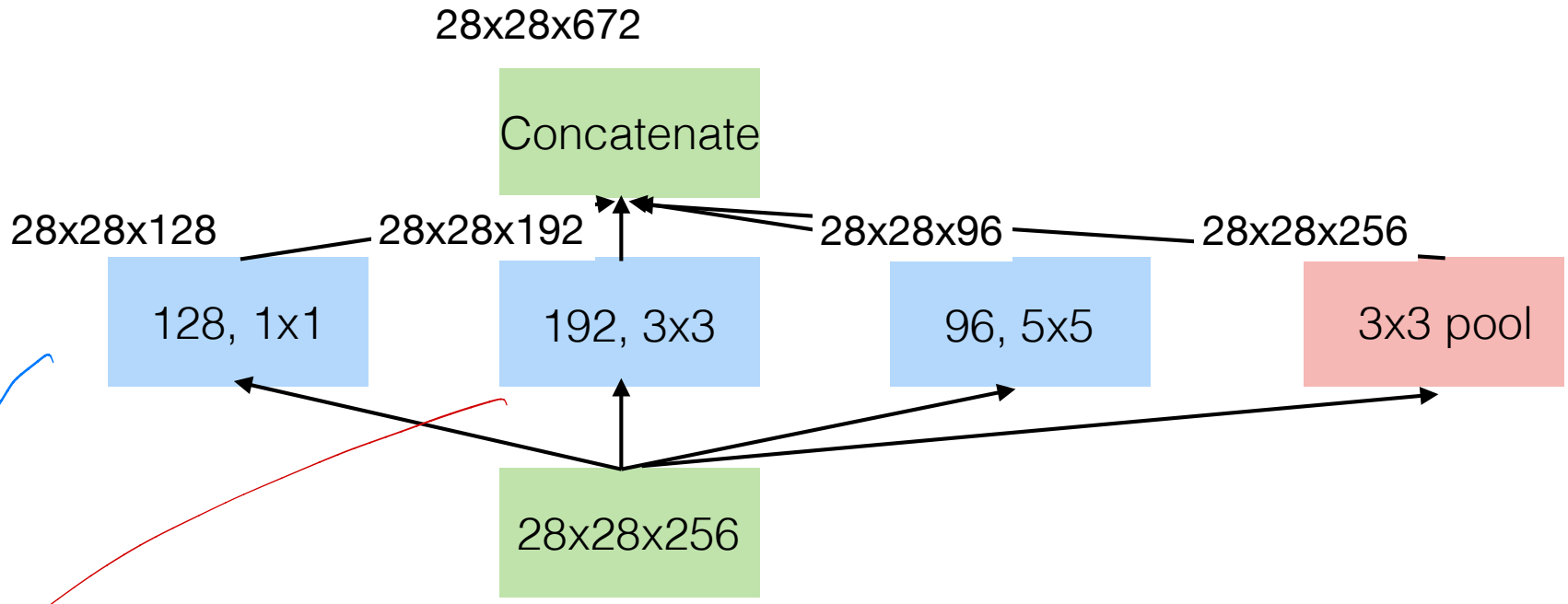
GoogLeNet



With this architecture, can the concatenated output ever have smaller depth (3rd dimension) than the input?

No.

GoogLeNet

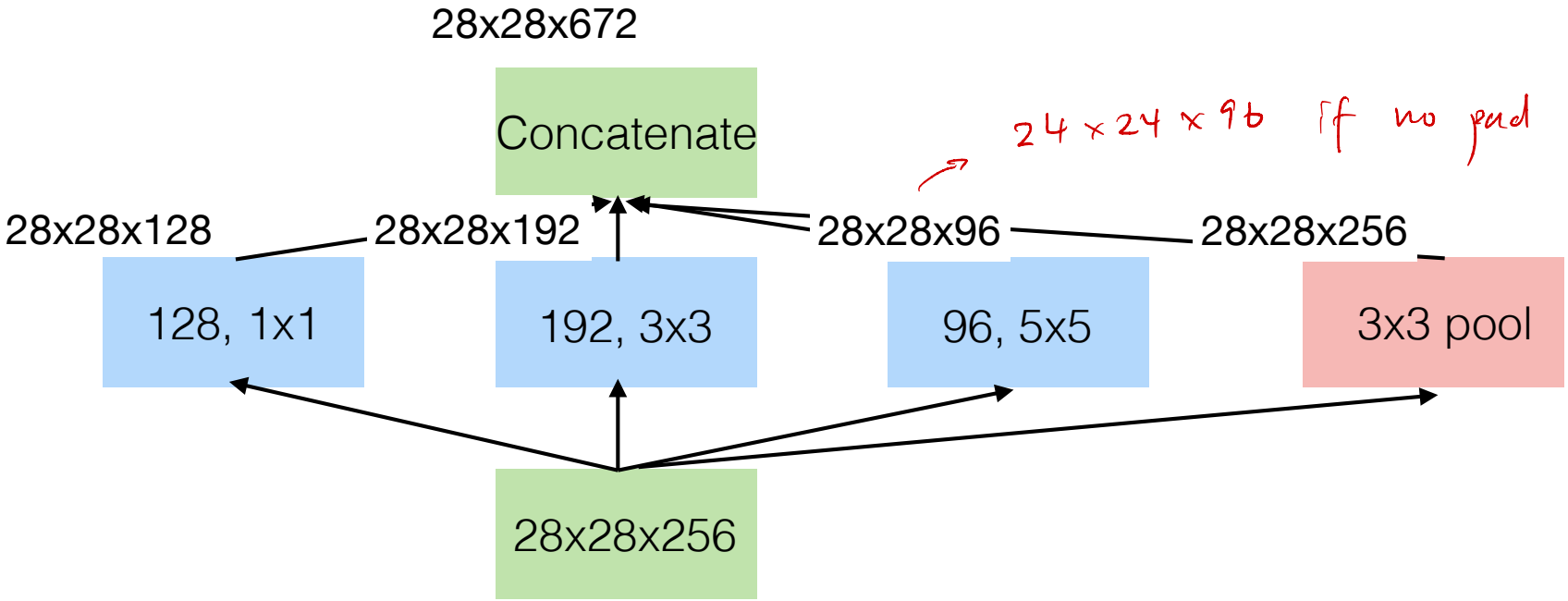


How many operations are there in this layer?

1x1 conv: $(28 \cdot 28 \cdot 128)(1 \cdot 1 \cdot 256)$

3x3 conv: $(28 \cdot 28 \cdot 192)(3 \cdot 3 \cdot 256)$

GoogLeNet



How many operations are there in this layer?

1x1x128 conv: 28*28*256*1*1*128 = 25,690,112
3x3x192 conv: 28*28*256*3*3*192 = 356,816,512
5x5x96 conv: 28*28*256*5*5*96 = 481,689,600

SUM = 864,196,224

Before

ops = (28×28×64)(1×1×256)
+ (28·28·192)(3·3·64)

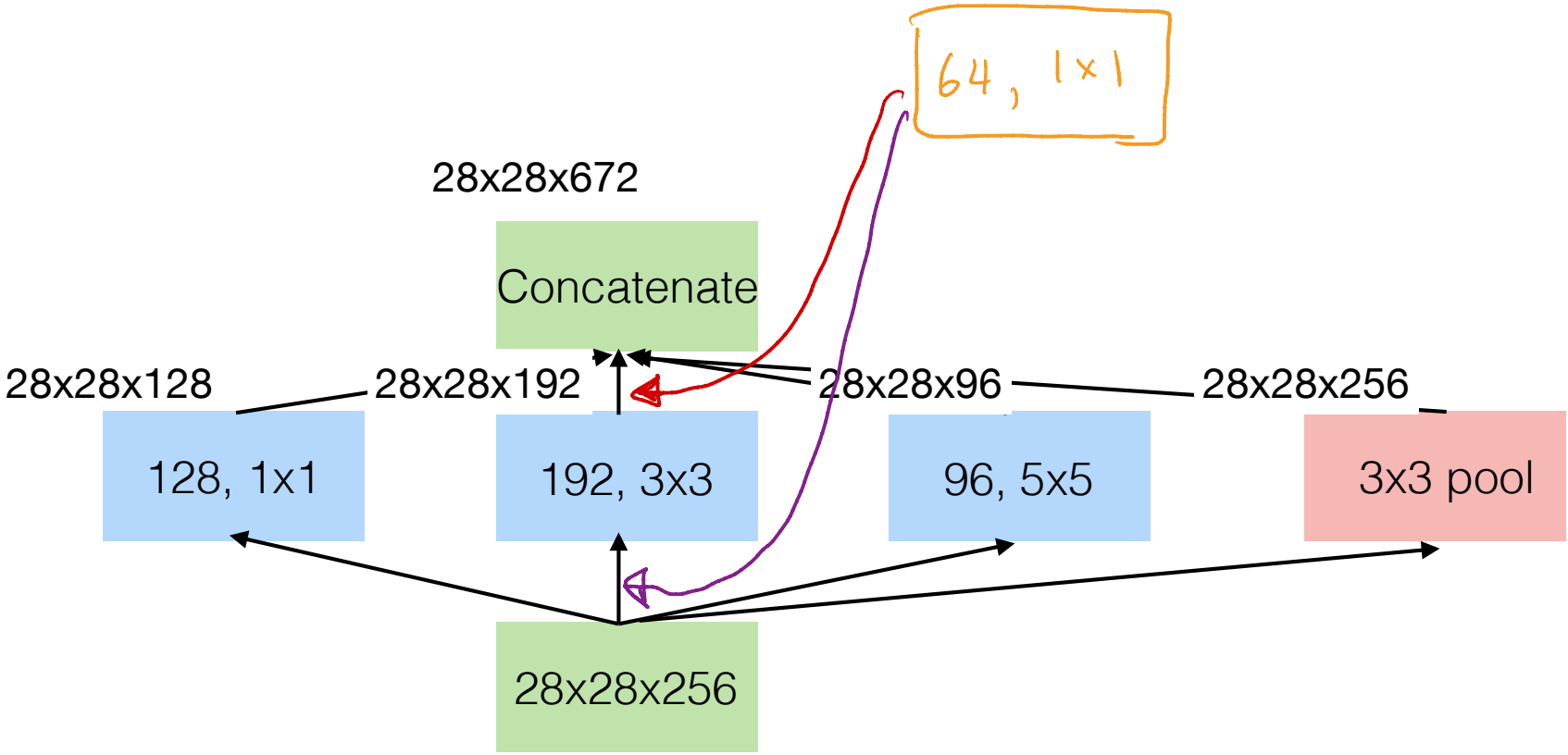
GoogLeNet

After

ops = (28×28×192)(3×3×256)
+ (28×28×64)(1×1×192)

To address this, in GoogLeNet, 1x1xF convolutional layers are added, that reduce the number of feature maps to substantially reduce the number of operations.

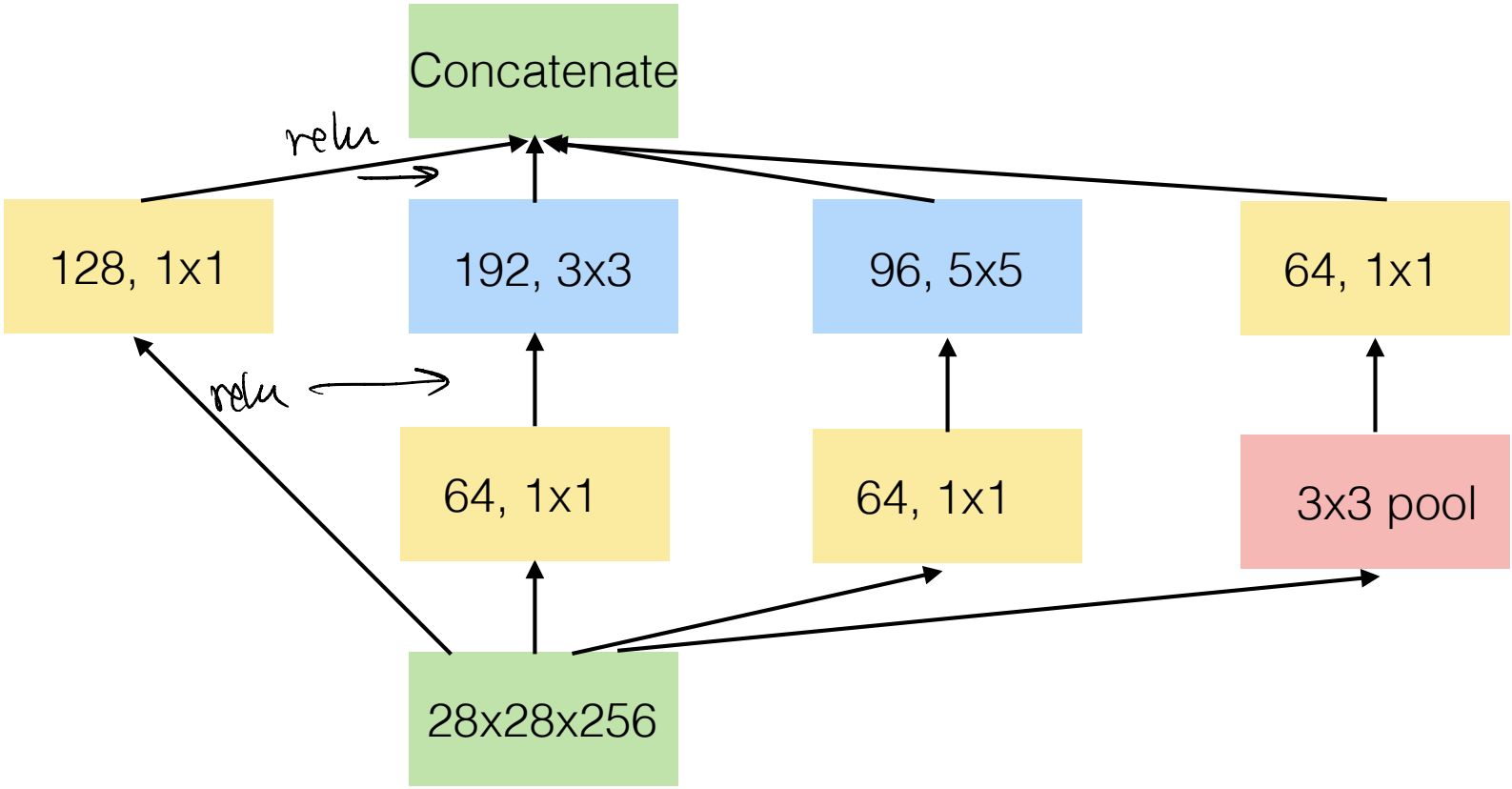
Question: Say F = 64. Where should we put these convolutional layers?



GoogLeNet

To address this, in GoogLeNet, 1x1xF convolutional layers are added, that reduce the number of feature maps to substantially reduce the number of operations.

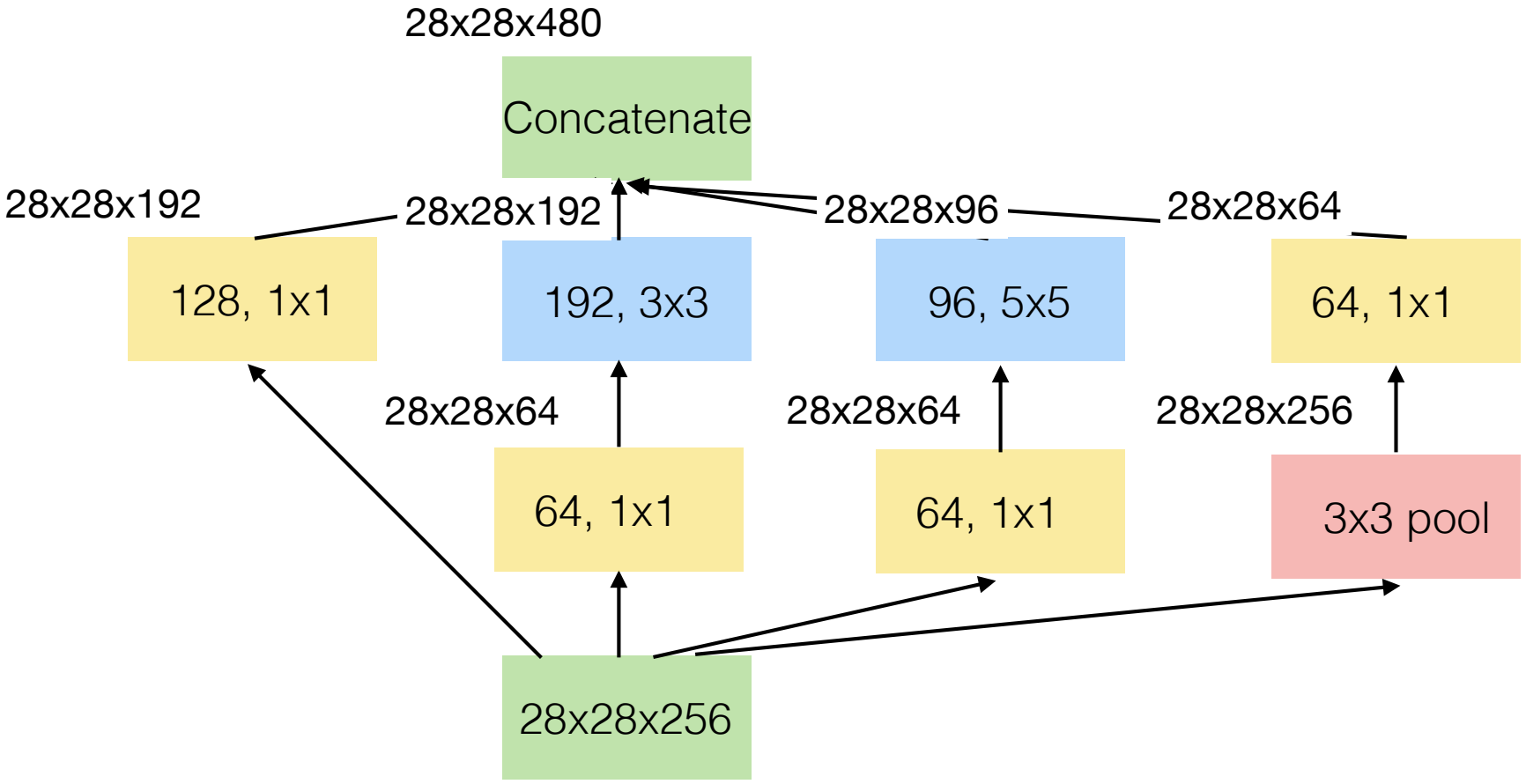
Question: Say F = 64. Where should we put these convolutional layers?



GoogLeNet

To address this, in GoogLeNet, 1x1xF convolutional layers are added, that reduce the number of feature maps to substantially reduce the number of operations.

Question: Say F = 64. Where should we put these convolutional layers?



GoogLeNet

How many operations?

1x1x128 conv: 28*28*256*1*1*128 = 25,690,112
1x1x64 conv: 28*28*256*1*1*64 = 12,845,056
1x1x64 conv: 28*28*256*1*1*64 = 12,845,056
1x1x64 conv: 28*28*256*1*1*64 = 12,845,056
3x3x192 conv: 28*28*64*3*3*192 = 86,704,128
5x5x96 conv: 28*28*64*5*5*96 = 120,422,400

SUM: 271,351,808

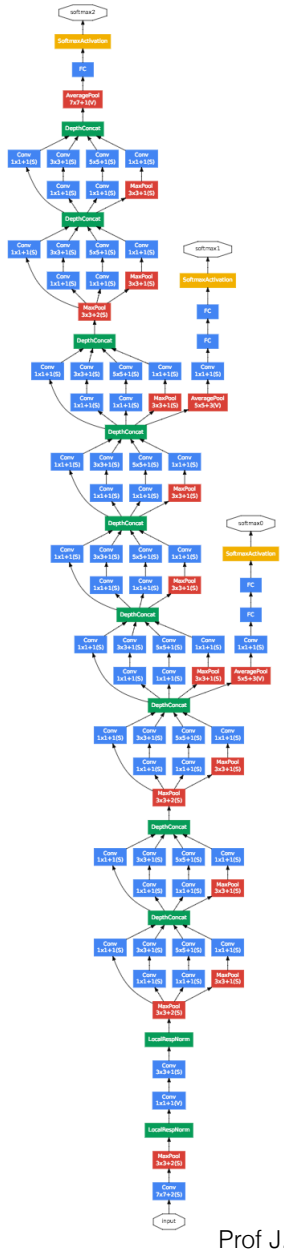
These 1x1 convolutions reduce the amount of computation by almost 4x in this example.

Is information lost in these 1x1x64 convolutions?

GoogLeNet

GoogLeNet architecture.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								



GoogLeNet

	Num layers
Convolution	1
Max pool	
Convolution (2x)	2
Max pool	
Inception (2x)	4
Max pool	
Inception (5x)	10
Max pool	
Inception (2x)	4
Avg pool	
FC (dropout)	1
Softmax	

Total layers: 22

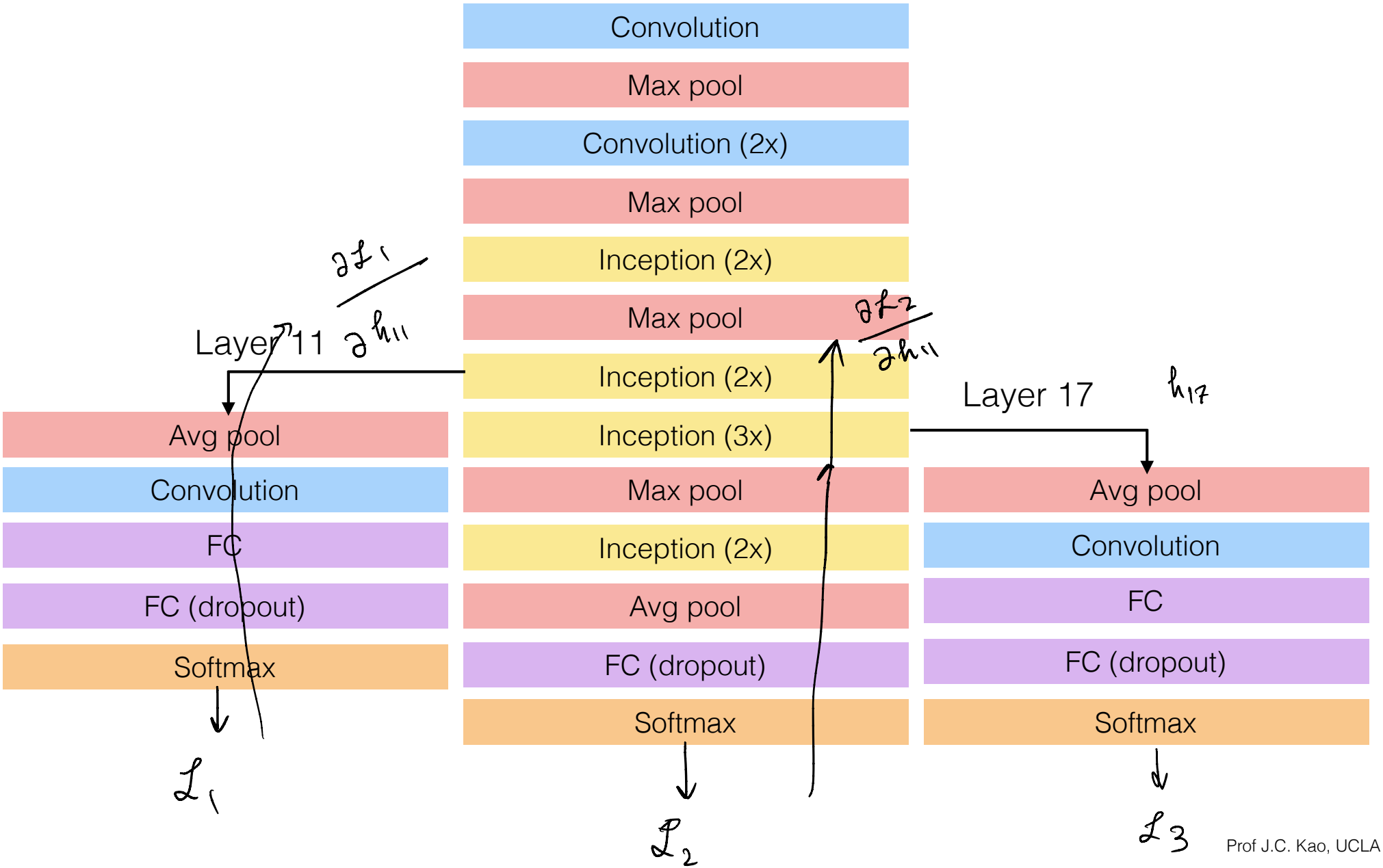
GoogLeNet

	Num layers
Convolution	1
Max pool	
Convolution (2x)	2
Max pool	
Inception (2x)	4
Max pool	
Inception (5x)	10
Max pool	
Inception (2x)	4
Avg pool	
FC (dropout)	1
Softmax	

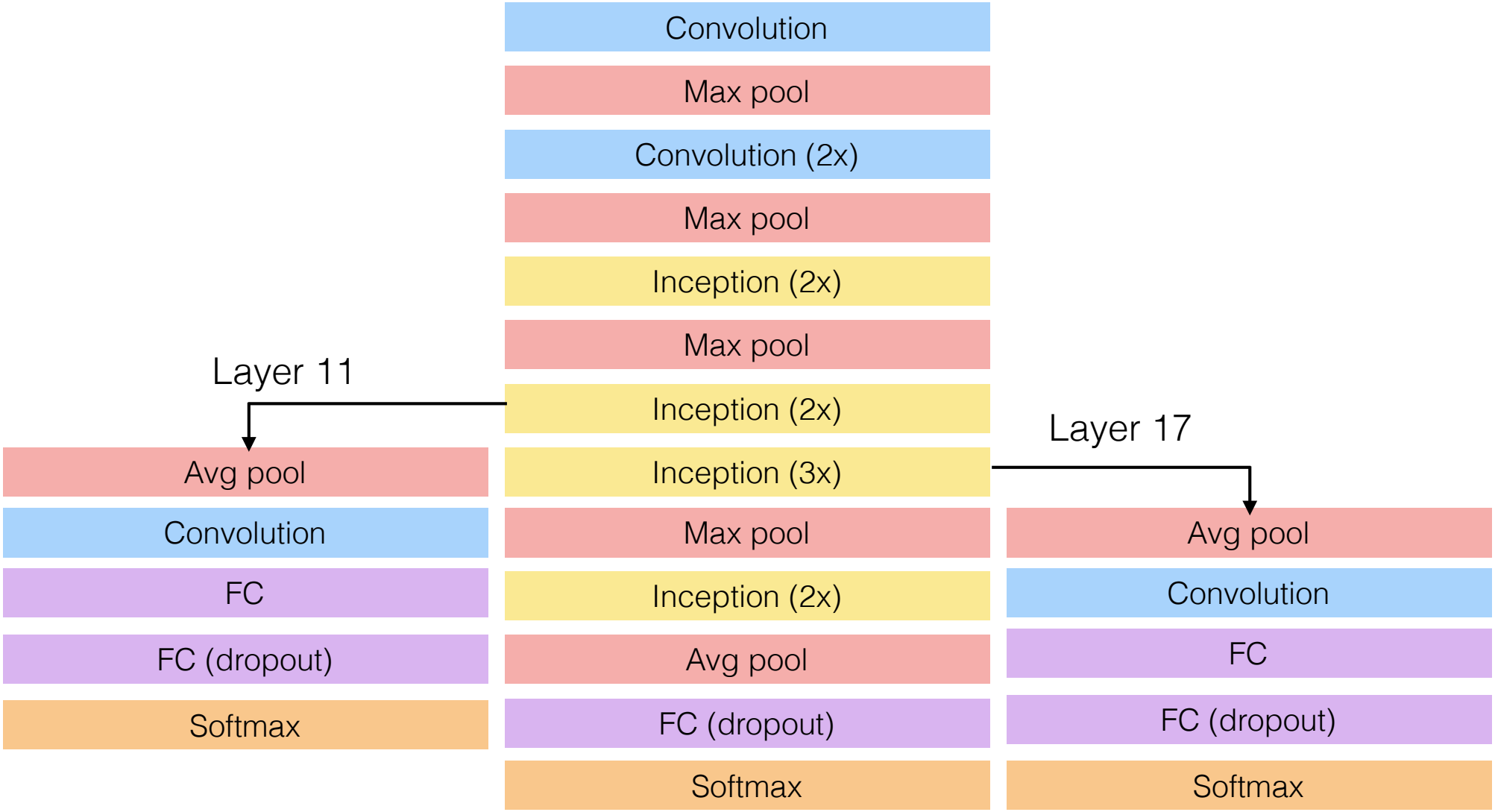
Total layers: 22

What might be a concern about this architecture?

GoogLeNet


Convolution
Max pool
Convolution (2x)
Max pool
Inception (2x)
Max pool
Inception (2x)
Inception (3x)
Max pool
Inception (2x)
Avg pool
FC (dropout)
Softmax
Layer 11
Avg pool
Convolution
FC
FC (dropout)
Softmax
Layer 17
Avg pool
Convolution
FC
FC (dropout)
Softmax

GoogLeNet



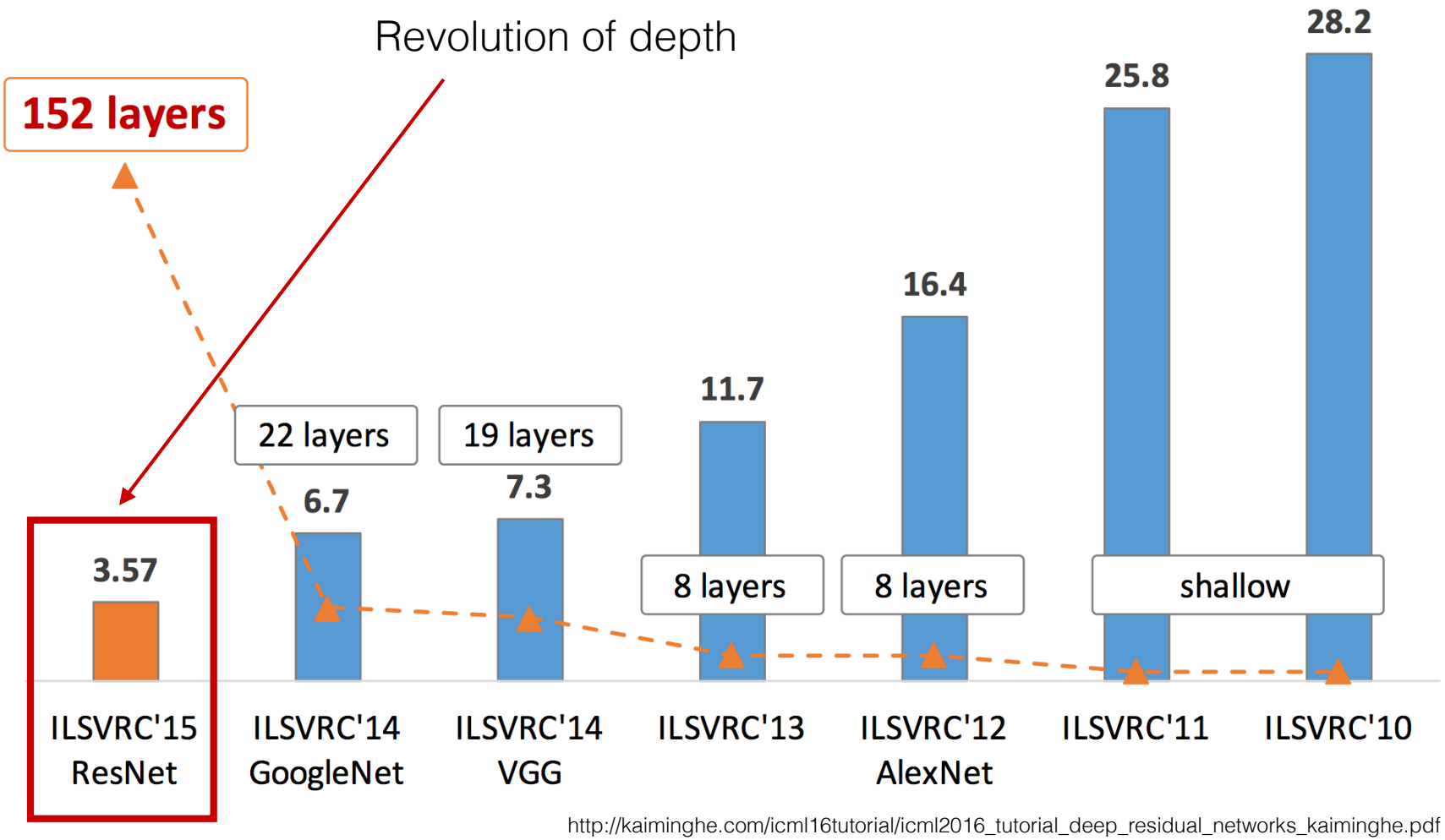
Later work showed auxiliary classifiers had a minor effect (0.5%) and you only need one of them. They're discarded at inference. Their loss is multiplied by 0.3.

GoogLeNet

Other details:

- SGD with 0.9 momentum.
- Decrease learning rate by 4% every 8 epochs.
- Image size patches from 8 to 100% of the area, having aspect ratios 3x4 or 4x3. Used 144 crops per image.
- Other “photometric” operations.
- Averaged results of 7 GoogLeNets.
- Did not use GPUs (!)
- Again, 12x less params than AlexNet. Won ILSVRC ’14. (6.7% top 5 error.)

ResNet



http://kaiminghe.com/icml16tutorial/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf