



Convolutional neural networks

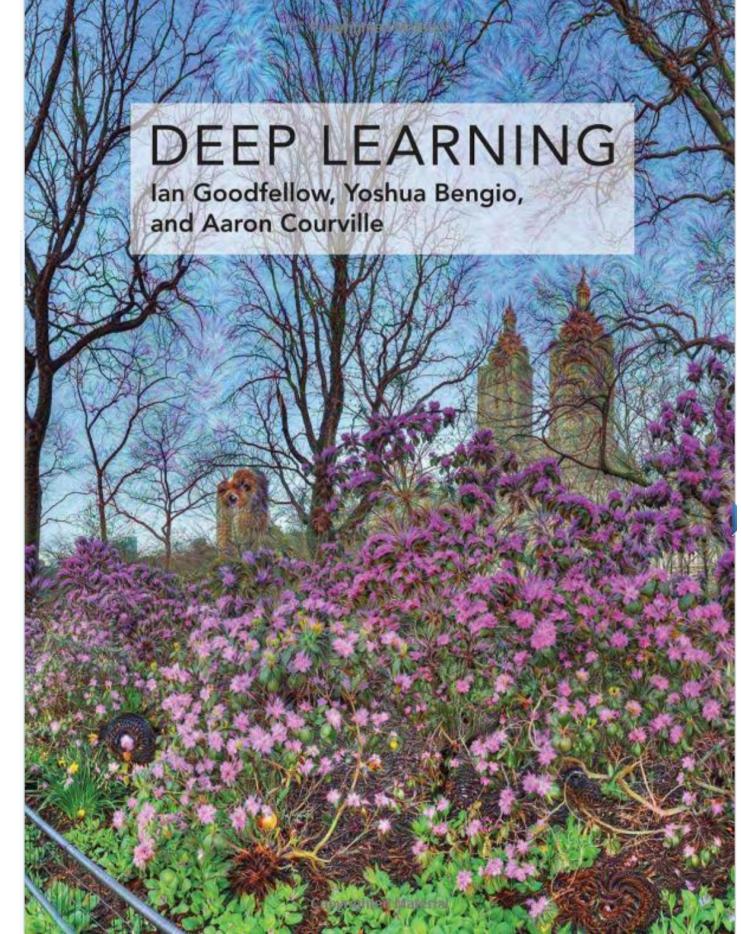
In this lecture, we'll talk about the convolutional neural network.



Reading

Reading:

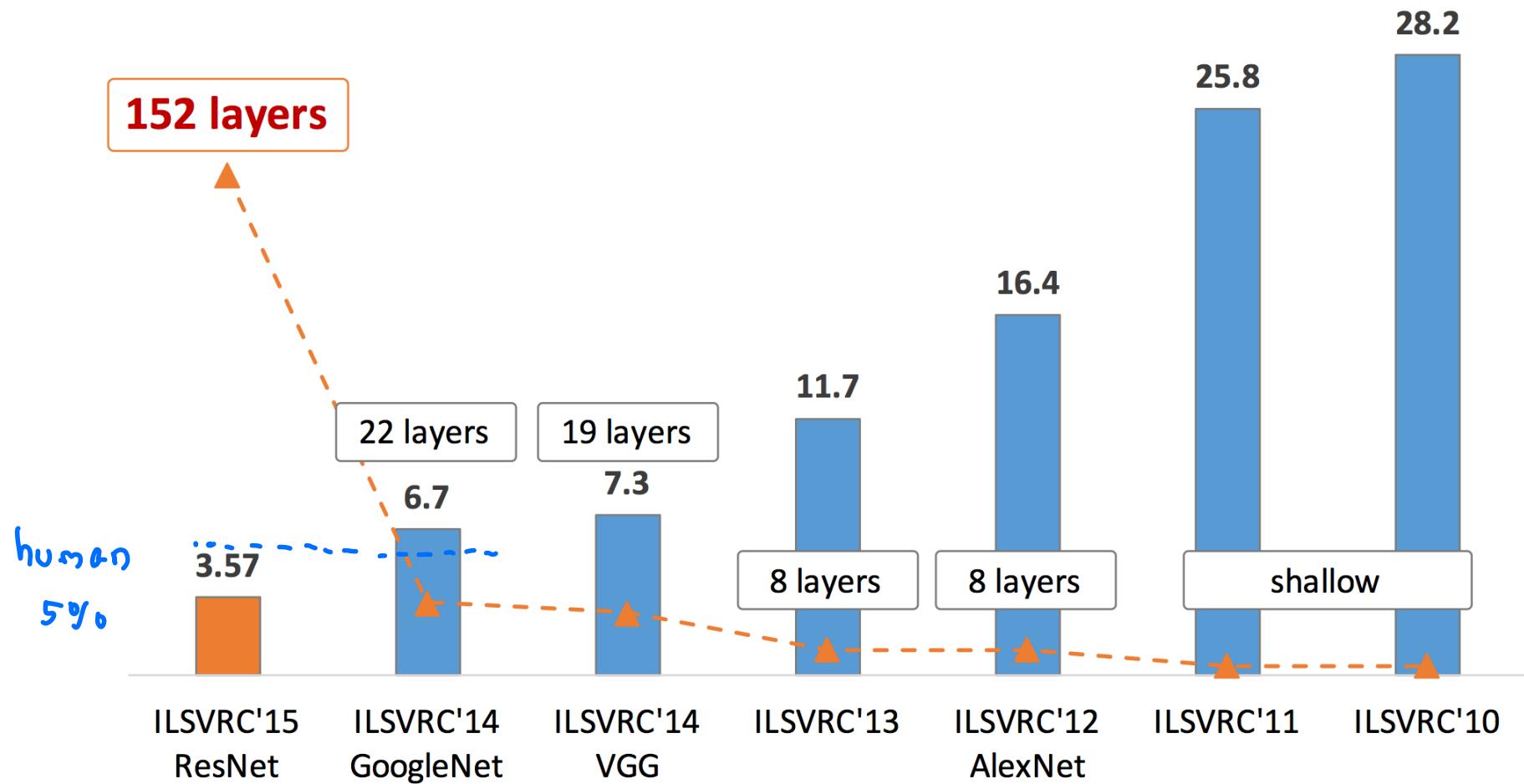
Deep Learning, Chapter 9





Background: convolutional neural networks

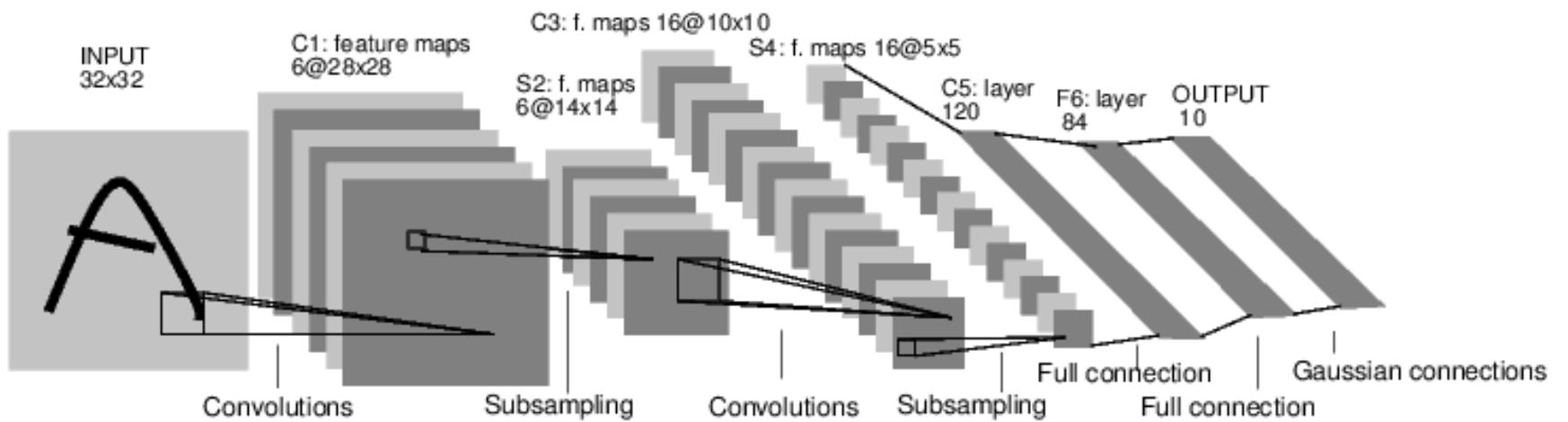
The CNN played a large role in this last “revival” of neural networks (i.e., the past 5 years).





Background: convolutional neural networks

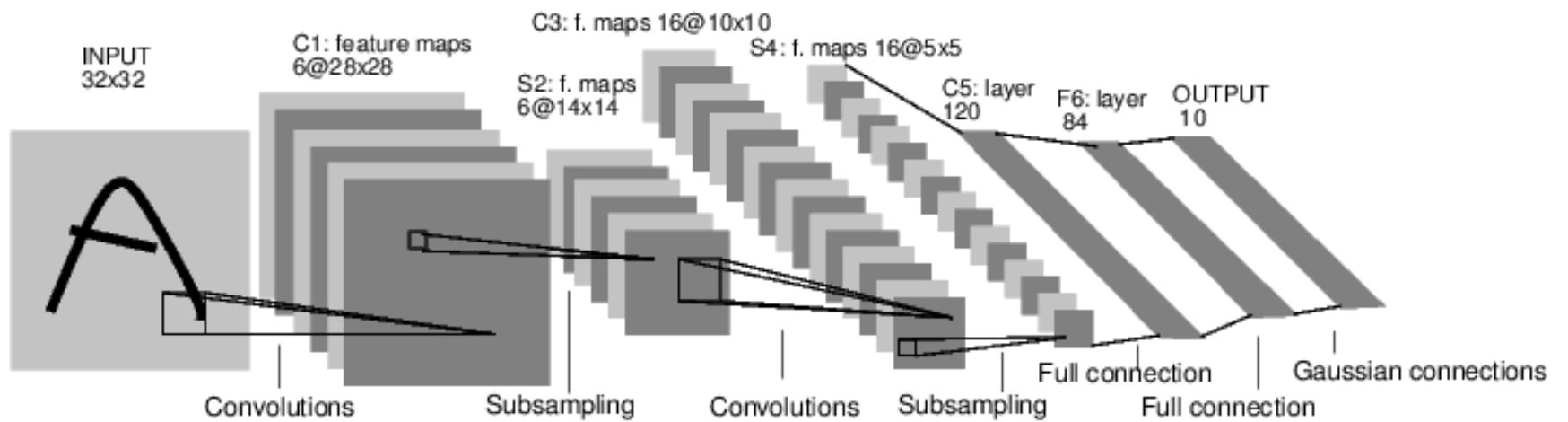
CNNs have been around since the 1990's. In 1998, Yann LeCun introduced LeNet, which is the modern convolutional neural network.





Background: convolutional neural networks

How do we arrive at this architecture? What are the principles that lead us to this?





Biological inspiration for the CNN

Biological inspiration

Principles of the convolutional neural network are inspired from neuroscience. Seminal work by Hubel and Wiesel in cat visual cortex in the 1960's (Nobel prize awarded in the 1980's) found that V1 neurons were tuned to the movement of oriented bars. Hubel and Wiesel also defined "simple" and "complex" cells, the computational properties of which were later well-described by linear models and rectifiers (e.g., Movshon and colleagues, 1978). Three key principles of neural coding in V1 are incorporated in convolutional neural networks (CNNs):

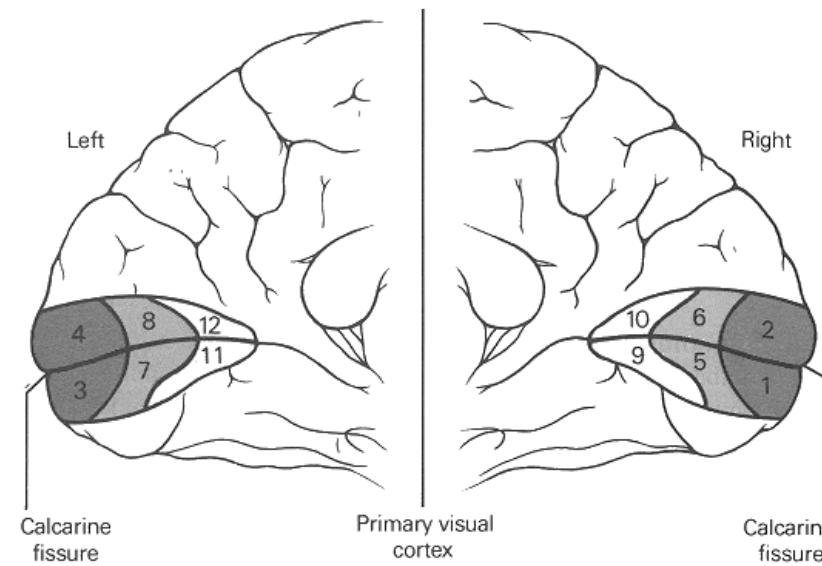
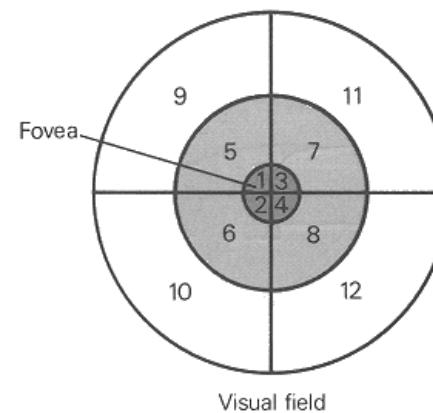
- V1 has a retinotopic map.
- V1 is composed of simple cells that can be adequately described by a linear model in a spatially localized receptive field.
- V1 is composed of complex cells that respond to similar features as simple cells, but importantly are largely invariant to the position of the feature.

If interested, you can YouTube some videos of their experiments. Warning: they do show cats in an experimental apparatus.



Biological inspiration for the CNN

- Retinotopic map: CNNs are spatially organized, so that nearby cells act on nearby parts of the input image.





Biological inspiration for the CNN

- affine transformation + thresholding (ReLU)
- Simple cells: CNNs use spatially localized linear filters, which are followed by thresholding.
 - Complex cells: CNNs use pooling units to incorporate invariance to shifts of the position of the feature.

→ "pool"



Biological inspiration for the CNN

Do CNN's compute like the visual system?

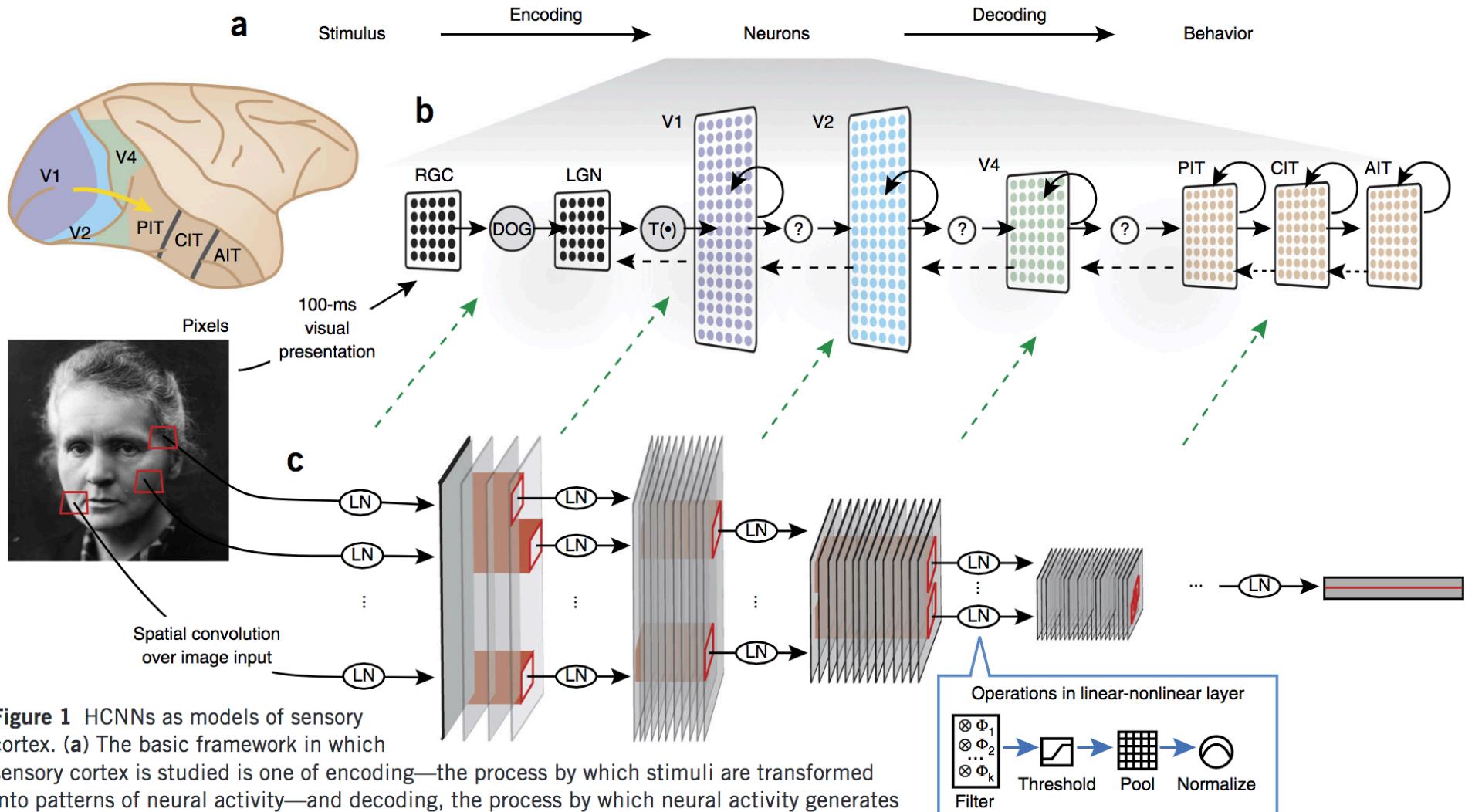
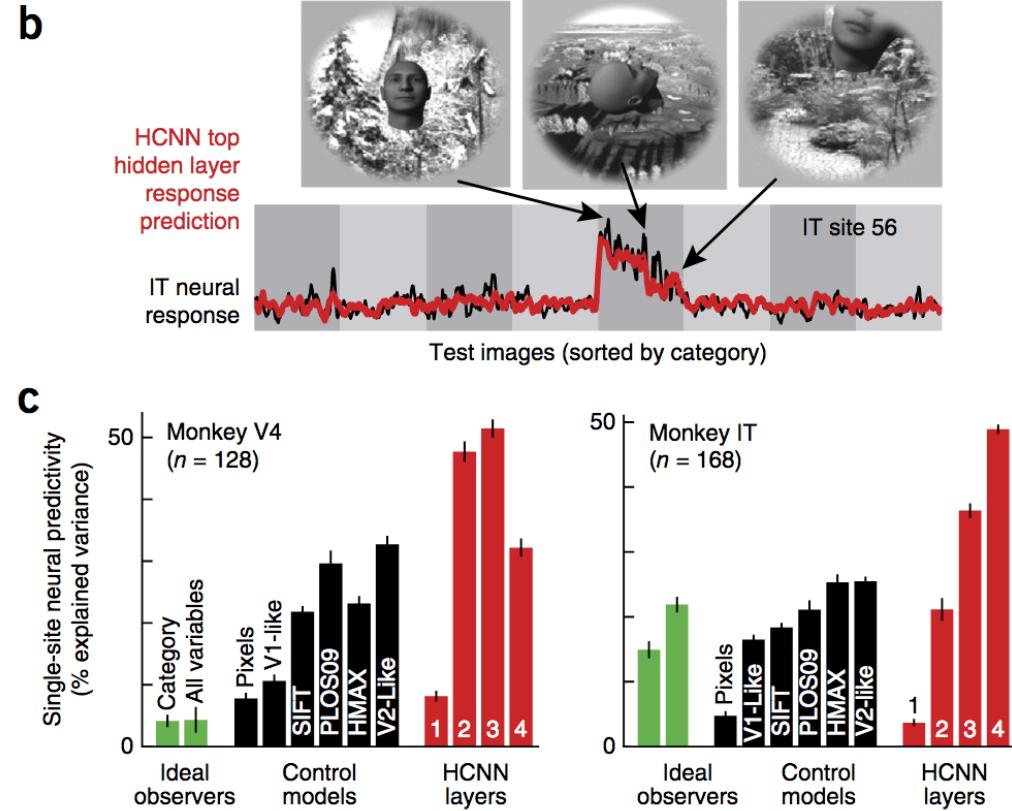


Figure 1 HCNNs as models of sensory cortex. (a) The basic framework in which sensory cortex is studied is one of encoding—the process by which stimuli are transformed into patterns of neural activity—and decoding, the process by which neural activity generates behavior. HCNNs have been used to make models of the encoding step; that is, they describe



Biological inspiration for the CNN

Do CNN's compute like the visual system?





Biological inspiration for the CNN

Do CNN's compute like the visual system?

Limitations of biological analogies

There are several limitations in these analogies. For example,

- CNNs have no feedback connections; neural populations are recurrently connected.
- The brain foveates a small region of the visual space, using saccades to focus on different areas.
- The output of the brain is obviously more than just image category classification.
- Pooling done by complex cells is an approximate way to incorporate invariance.



Motivation for CNNs

Motivation for convolutional neural networks

- For images, fully connected networks require many parameters. In CIFAR-10, the input size is $32 \times 32 \times 3 = 3072$, requiring 3072 weights for each neuron. For a more normal sized image that is 200×200 , each neuron in the *first layer* would require 120000 parameters. This quickly gets out of hand. This network, with a huge number of parameters, would not only take long to train, but may also be prone to overfitting.



Convolution

The convolution operation

The convolution of two functions, $f(t)$ and $g(t)$, is given by:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau$$

In discrete time, this is given by:

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n - m)$$



Correlation and convolution

Note, however, that in general CNNs don't use *convolution*, but instead use *cross-correlation*. Colloquially, instead of “flip-and-drag,” CNNs just “drag.” For real-valued functions, cross-correlation is defined by:

$$(f \star g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n+m)$$

We'll follow the field's convention and call this operation convolution.



Convolution in 2D

Convolution in 2D

The 2D convolution (formally: cross-correlation) is given by:

$$(f * g)(i, j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m, n)g(i + m, j + n)$$

This generalizes to higher dimensions as well. Note also: these “convolutions” are not commutative.



Convolution in 2D example

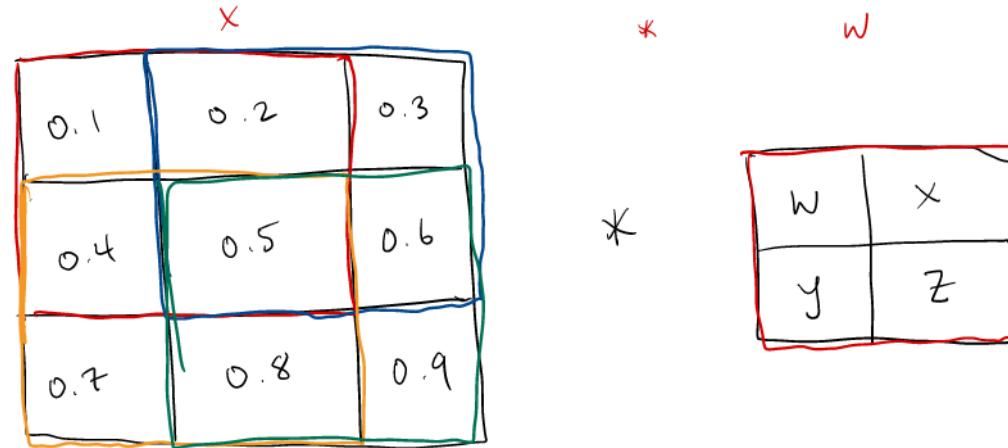
$$\begin{matrix} & \begin{matrix} 1 & 1 & 1 \end{matrix} \\ \begin{matrix} 1 & 1 & 1 \end{matrix} & * \end{matrix} \quad \begin{matrix} w & x \\ y & z \end{matrix}$$

The diagram illustrates a 3x3 input matrix and a 2x2 filter matrix. The input matrix has all elements equal to 1. The filter matrix has elements labeled w, x, y, and z. The asterisk (*) between the two matrices indicates the convolution operation.

***** All convolutions in this class will be VALID convolutions (the filter and inputs must be totally overlapping). *****



Convolution in 2D



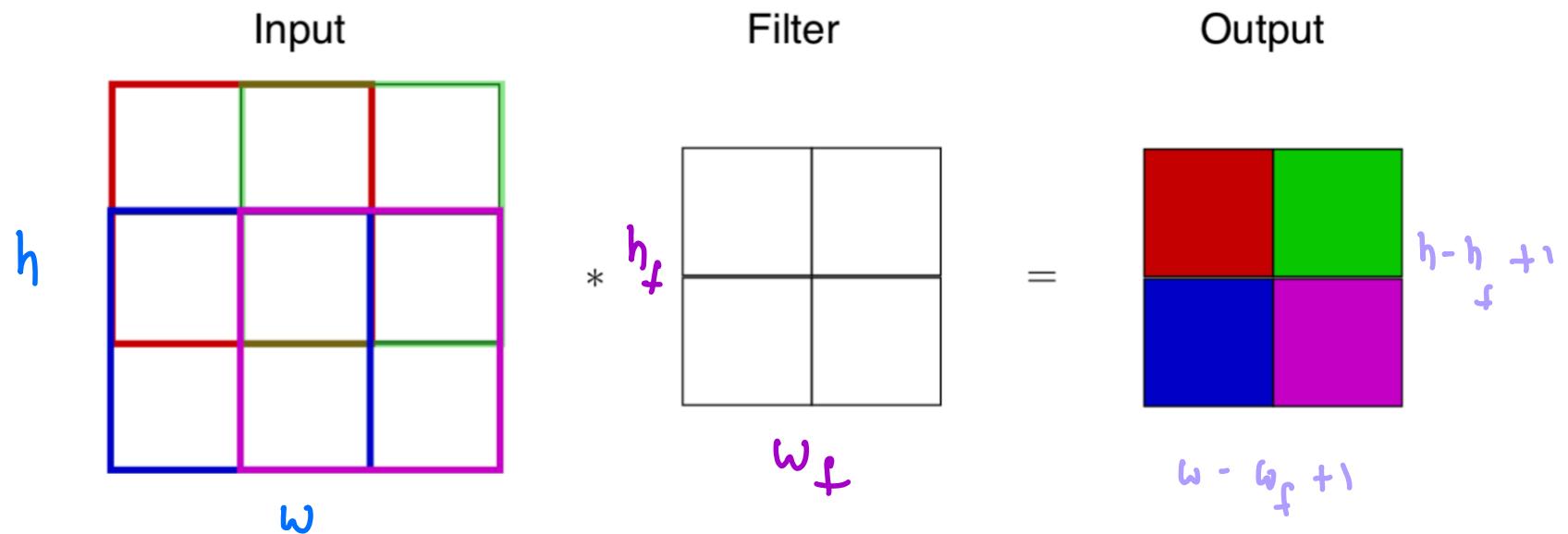
$$\begin{array}{|c|c|} \hline & 0.1w + 0.2x \\ \hline 0.2w + 0.3x & \\ \hline & + 0.4y + 0.5z \\ \hline & + 0.7y + 0.8z \\ \hline & 0.4 \cdot w + 0.5x \\ \hline & + 0.7y + 0.82 \\ \hline & 0.5 \cdot w + 0.6x \\ \hline & + 0.8y + 0.9z \\ \hline \end{array}$$

"VALID CONVOLUTION"



Convolution in 2D

*** All convolutions in this class will be VALID convolutions (the filter and inputs must be totally overlapping). ***



$$(w - w_f + 1, h - h_f + 1)$$

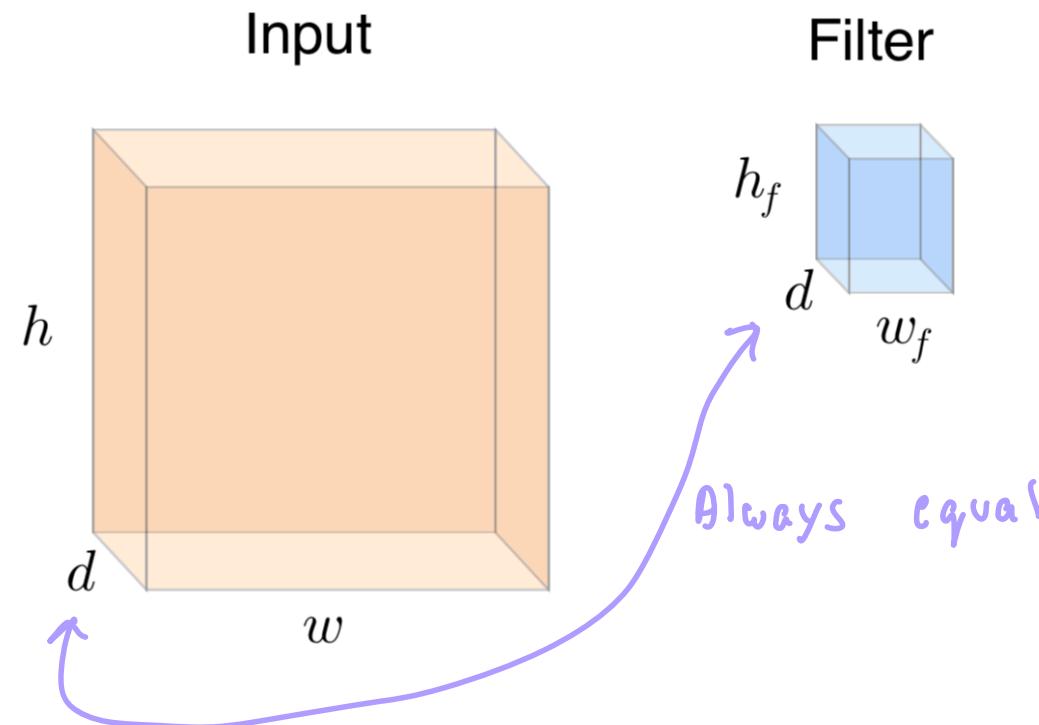


The convolutional layer of a CNN

Convolutional layer

This convolution operation (typically in 3D, since images often come with a width and height as well as *depth* for the R, G, B channels) defines the “convolutional layer” of a CNN. The convolutional layer defines a collection of filters (or activation maps), each with the same dimension as the input.

- Say the input was of dimensionality (w, h, d) .
- Say the filter dimensionality is (w_f, h_f, d) . So that the filter operates on a small region of the input, typically $w_f < w$.
- The depths being equal means that the output of this convolution operation is 2D.

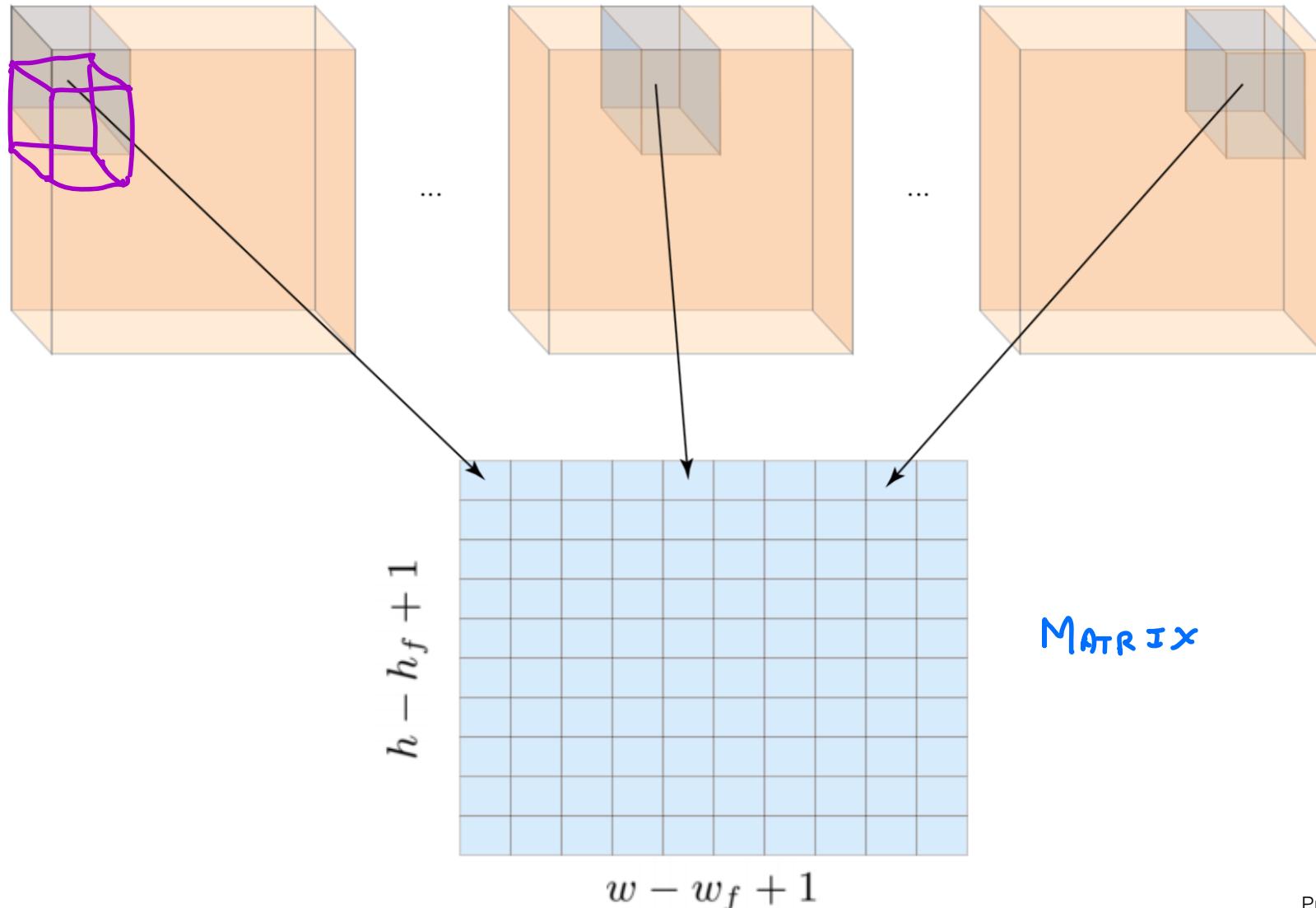




The convolutional layer of a CNN

Convolutional layer (cont.)

After performing the convolution, the output is $(w - w_f + 1, h - h_f + 1)$

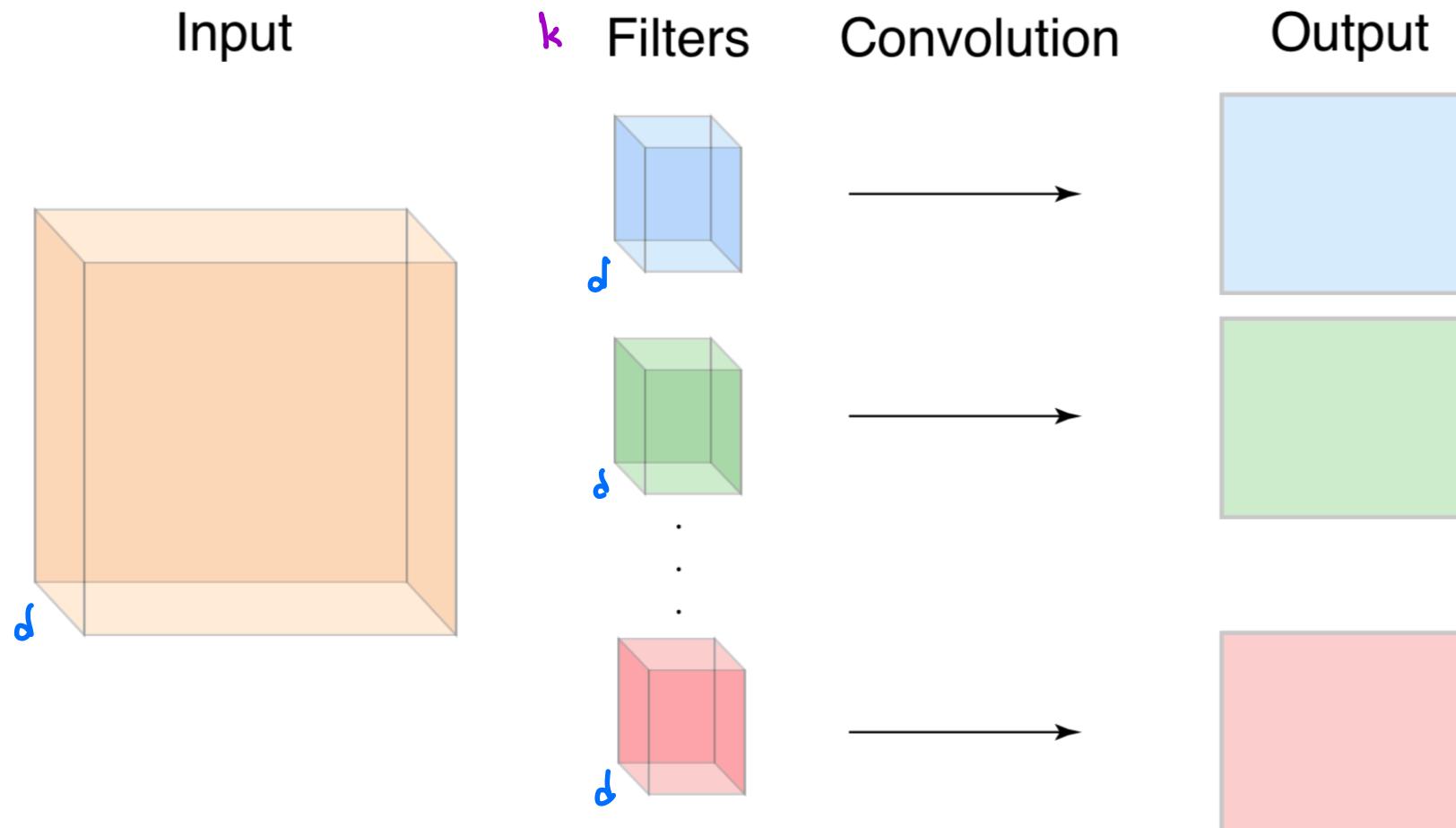




The convolutional layer of a CNN

Convolutional layer (cont.)

Now, we don't have just one filter in a convolutional layer, but multiple filters. We call each output (matrix) a slice.

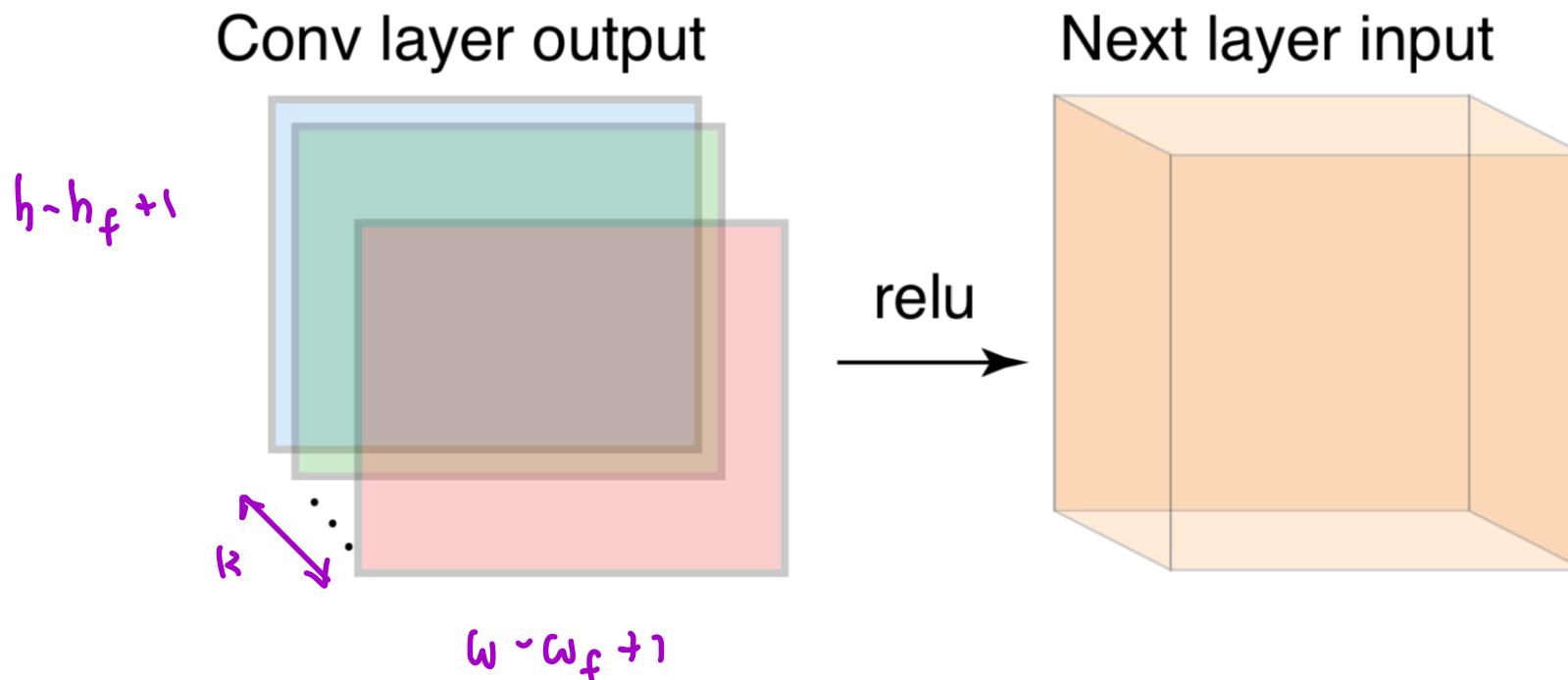




The convolutional layer of a CNN

Convolutional layer (cont.)

The output slices of the convolution operations with each filter are composed together to form a $(w - w_f + 1, h - h_f + 1, n_f)$ tensor, where n_f is the number of filters. The output is then passed through an activation nonlinearity, such as $\text{ReLU}(\cdot)$. This then acts as the input to the next layer.



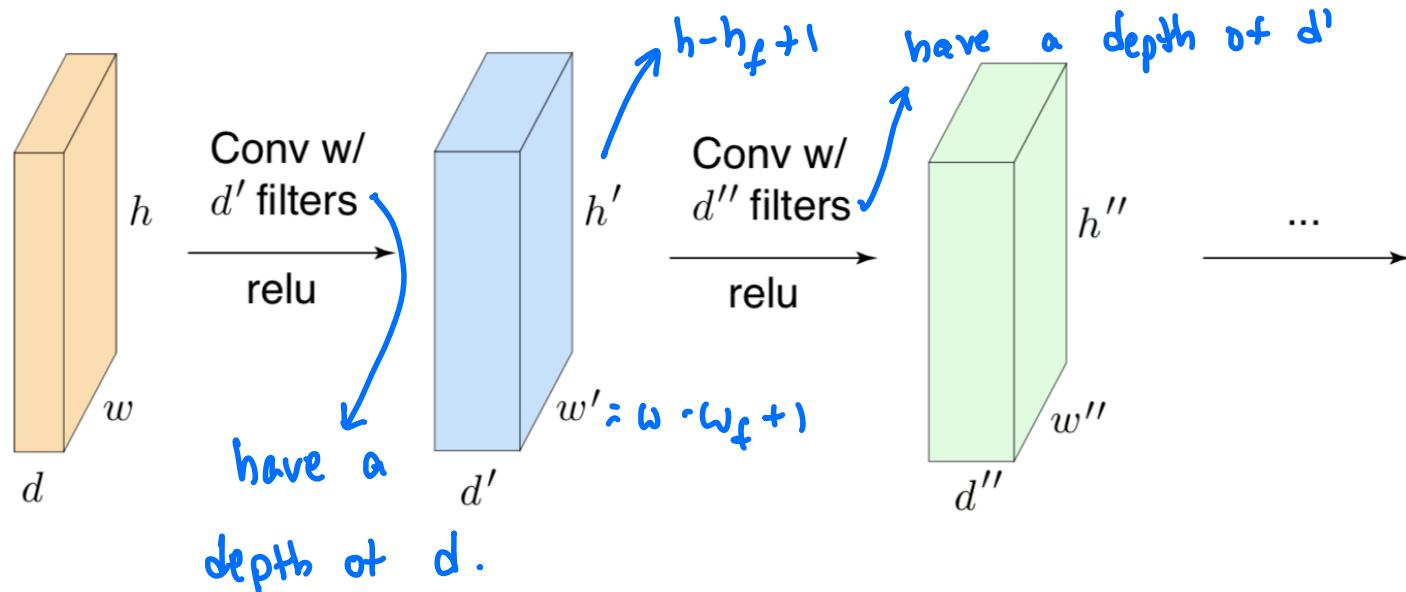


VERY IMPORTANT
FOR MIDTERM

The convolutional layer of a CNN

Composition of convolutional layers

These layers can then be composed, which comprise a large component of the CNN.



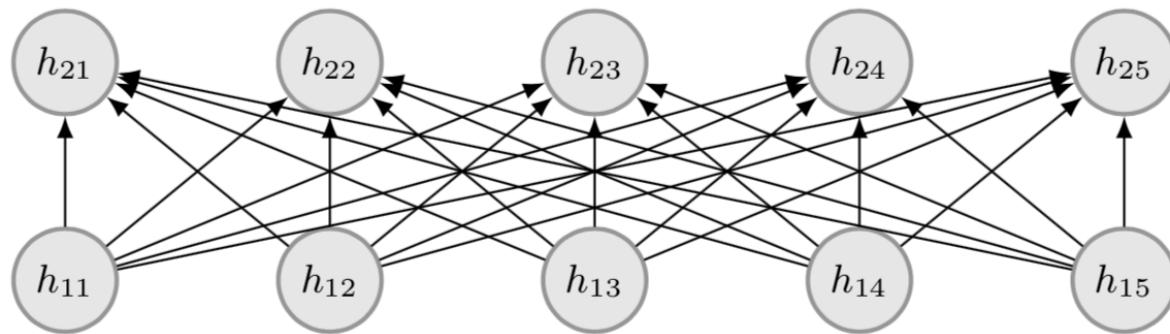


Convolutional layers have sparse interactions

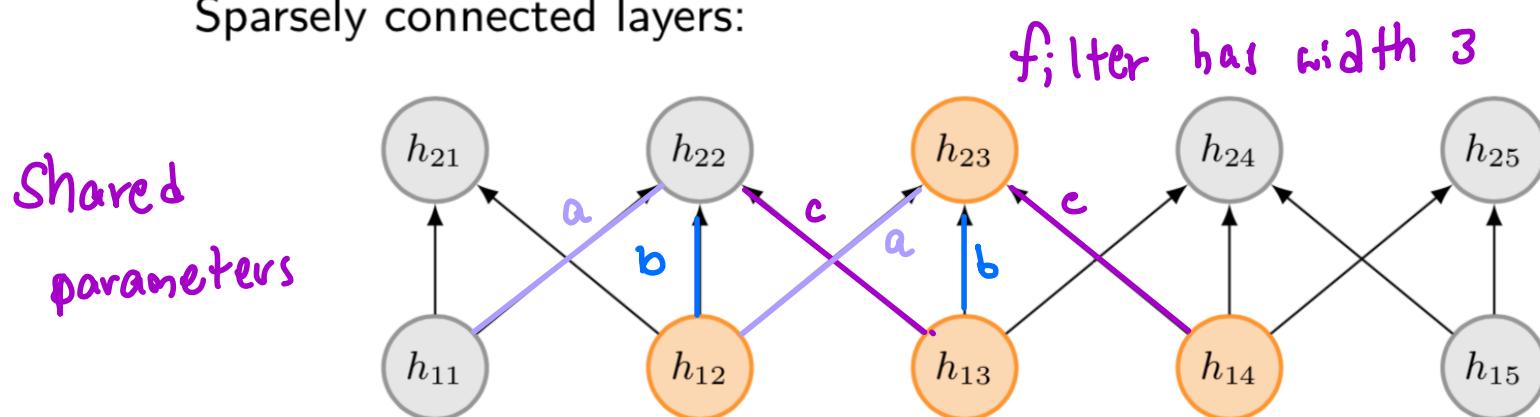
Convolutional layers have sparse interactions

Convolutional layers have *sparse interactions* or *sparse connectivity*. The idea of sparse connectivity is illustrated below, where each output is connected to a small number of inputs.

Fully connected layers:



Sparsely connected layers:





Convolutional layers have sparse interactions

Convolutional layers have sparse interactions (cont.)

Sparse interactions aid with computation.

- **Sparse interactions reduce computational memory.** In a fully connected layer, each neuron has $w \cdot h \cdot d$ weights corresponding to the input. In a convolutional layer, each neuron has $w_f \cdot h_f \cdot d$ weights corresponding to the input. Moreover, in a convolutional layer, every output neuron in a slice has the *same* $w_f \cdot h_f \cdot d$ weights (more on this later). As there are far fewer parameters, this reduces the memory requirements of the model.
- **Sparse interactions reduce computation time.** If there are m inputs and n outputs in a hidden layer, a fully connected layer would require $\mathcal{O}(mn)$ operations to compute the output. If each output is connected to only k inputs (e.g., where $k = w_f \cdot h_f \cdot d$) then the layer would require $\mathcal{O}(kn)$ operations to compute the output.

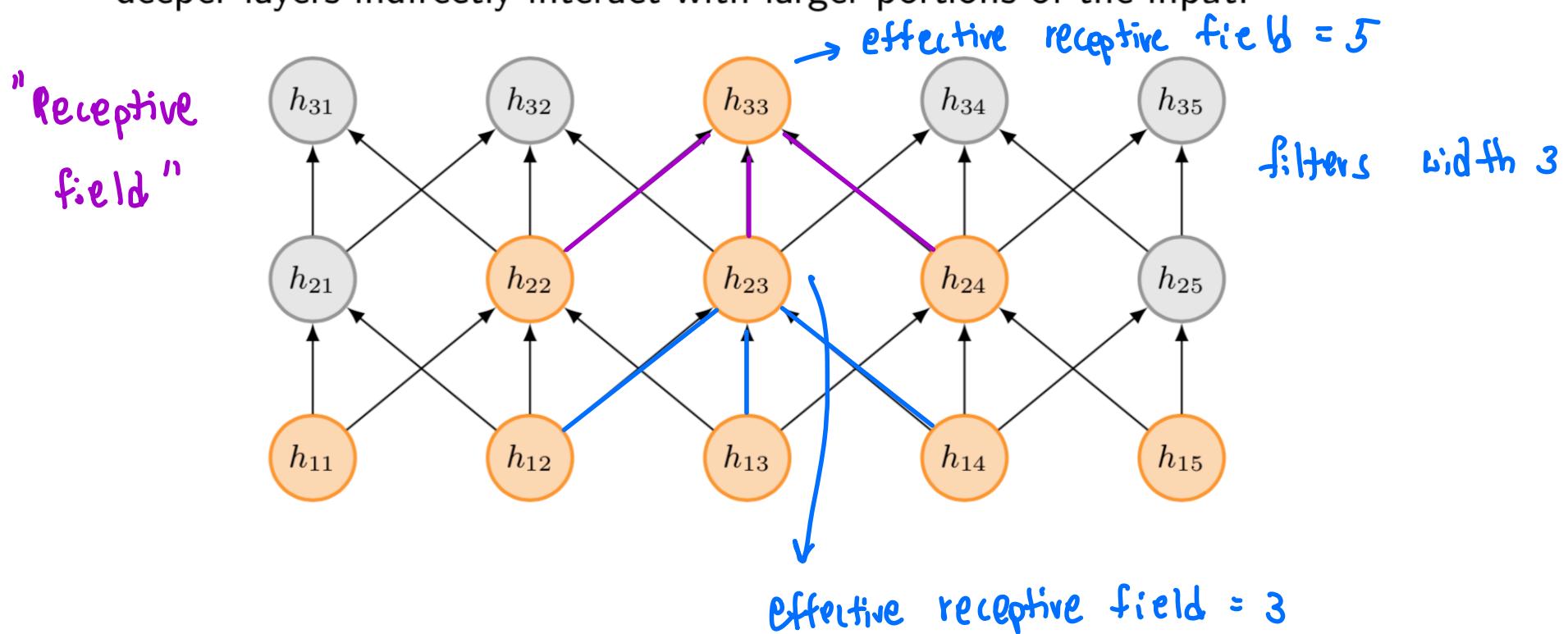


Convolutional layers have sparse interactions

Convolutional layers have sparse interactions (cont.)

A concern of sparse interactions is that information from different parts of the input may not interact. For example, a self-driving car should know where obstacles are from all over the image to avoid them.

This argues that networks should be composed of more layers, since units in deeper layers indirectly interact with larger portions of the input.





Convolutional layers share parameters

Convolutional layers share parameters

Convolutional layers have shared parameters (or *tied weights*), in every output neuron in a given slice uses the same set of parameters in the filter.

Example: Consider an input that is $(32 \times 32 \times 3)$. We have two architectures; in the fully connected architecture, there are 500 output neurons at the first layer. In the convolutional neural net there are 4 filters that are all $4 \times 4 \times 3$.
(1) How many output neurons are there in the convolutional neural network, assuming that the convolution is only applied in regions where the filter fully overlaps the kernel? (2) How many parameters are in each model?



Convolutional layers can be applied to inputs of variable size

Convolutional layers can handle variable sized inputs

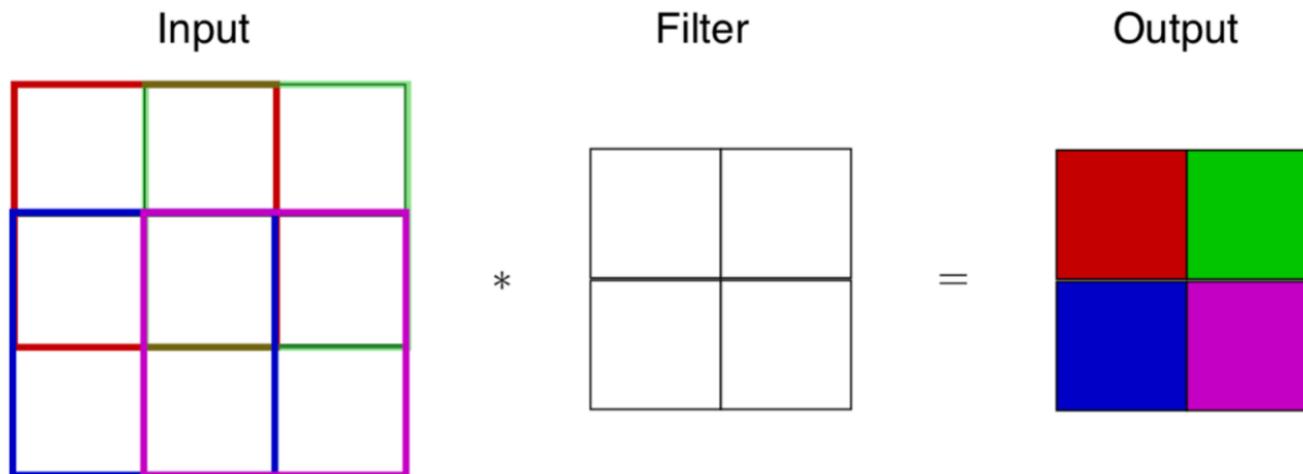
Convolutional neural networks can process inputs with varying sizes or spatial extents. In a fully connected layer, each layer specifies parameters \mathbf{W}, \mathbf{b} such that $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, and thus the inputs and output scales are defined. In a convolutional layer, the parameters are convolutional filters, which are then convolved with an input.

- If the input changes in size, the convolution operation can still be performed.
- The dimensions of the output will be different, which may be okay depending on your application.
- If the output dimensions are a fixed size, some modifications may be made to the network (e.g., if the output size needs to be decreased, pooling layers, described later, can scale with the network input size to create the correct output size).



All convolutions in this class are valid convolutions

In this class, all convolutions will be valid convolutions. We explicitly specify the amount of zero padding when we need it.



Output size: $(w - w_f + 1, h - h_f + 1)$



Convolutional padding

Convolution padding (cont.)

For the rest of the class, we'll assume convolution is only applied when the filter completely overlaps the input. Now, the input and output will remain the same size if the input is zero-padded with $w_f - 1$ total zeros (i.e., $(w_f - 1)/2$ zeros on the left and right of the matrix) and the height is zero-padded with $h_f - 1$ total zeros (i.e., $(h_f - 1)/2$ zeros on the top and bottom of the matrix).

Usually, we specify a variable pad which is the amount of zeros to pad on each border.

$$\text{pad} = 0$$

$$\text{pad} = 1$$

0	0	0	0	0
0				0
0				0
0				0
0	0	0	0	0

$$\text{pad} = 2$$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0					0
0	0					0
0	0					0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

The output of the convolution is now $(w - w_f + 1 + 2\text{pad}, h - h_f + 1 + 2\text{pad})$.



Convolutional padding

Convolution padding (cont.)

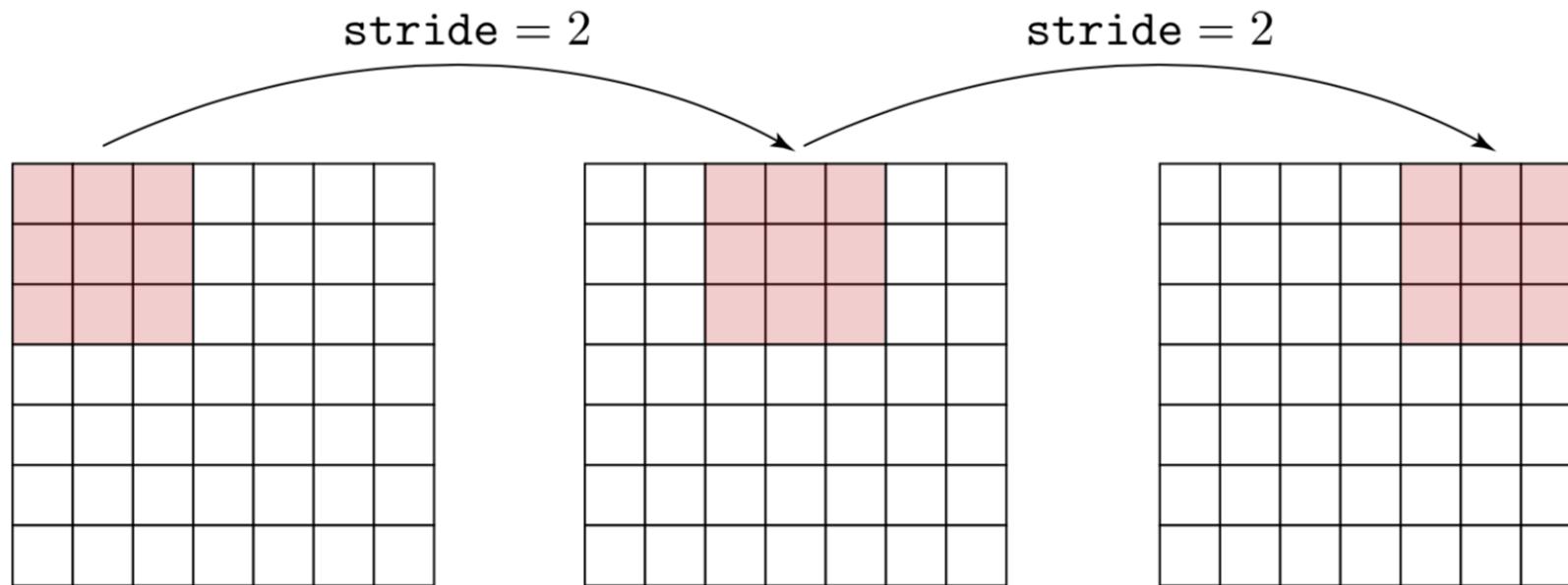
It is worth noting that Goodfellow et al. report that the optimal amount of zero padding (in terms of test accuracy) is somewhere between $\text{pad} = 0$ and the pad that causes the output and input to have the same width and height.



Convolutional stride

Convolution stride

Another variable to control is the stride, which defines how much the filter moves in the convolution. For normal convolution, $\text{stride} = 1$, which denotes that the filter is dragged across every part of the input. Consider a 7×7 input with a 3×3 filter. If convolution is only applied where the kernel overlaps the input, the output is 5×5 . With $\text{stride} = 2$, the output is 3×3 .





Convolutional stride

Convolution stride (cont.)

The stride must be sensible. In the 7×7 input with 3×3 filter example, it is not correct to use stride = 3 as it is not consistent with input. However, one can zero-pad the input so that the stride is appropriate. The output size after accounting for stride and padding is:

$$\left(\frac{w - w_f + 2\text{pad}}{\text{stride}} + 1, \frac{h - h_f + 2\text{pad}}{\text{stride}} + 1 \right)$$

Be careful when incorporating stride, as the data representation size will shrink in a divisive manner.

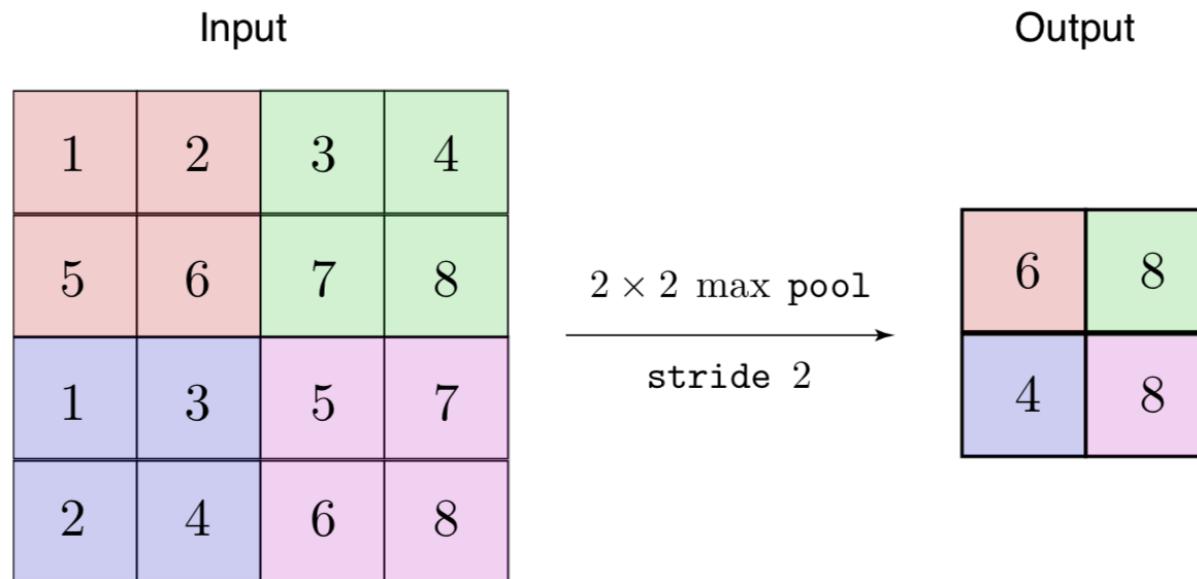


Pooling layer

Pooling layer

CNNs also incorporate pooling layers, where an operation is applied to all elements within the filtering extent. This corresponds, effectively, to downsampling.

The pooling filter has width and height (w_p, h_p) and is applied with a given stride. It is most common to use the `max()` operation as the pooling operation.





Pooling layer

Pooling layer (cont.)

A few notes on pooling layers.

- The resulting output has dimensions

$$\left(\frac{w - w_p}{\text{stride}} + 1, \frac{h - h_p}{\text{stride}} + 1 \right)$$

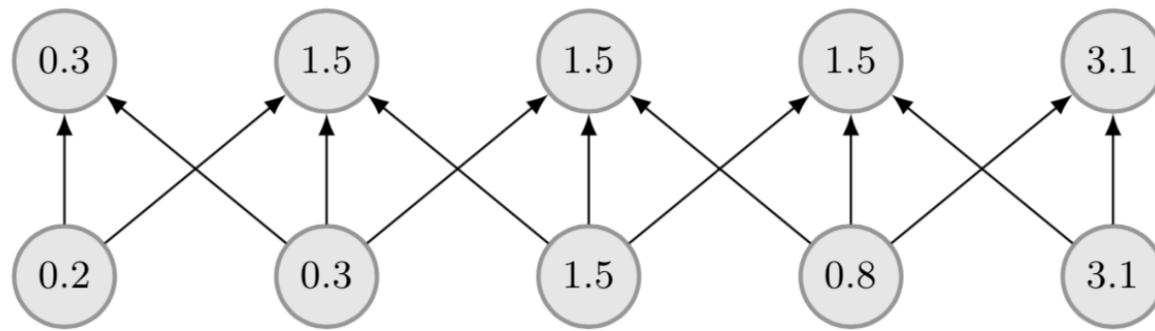
- Pooling layers introduce *no* additional parameters to the model.
- Pooling layers are intended to introduce small spatial invariances.



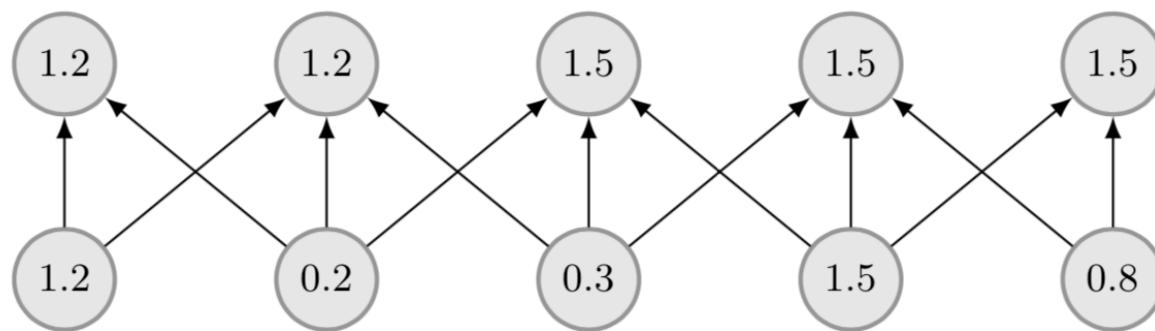
Pooling layer

Pooling layer (cont.)

Invariance example:

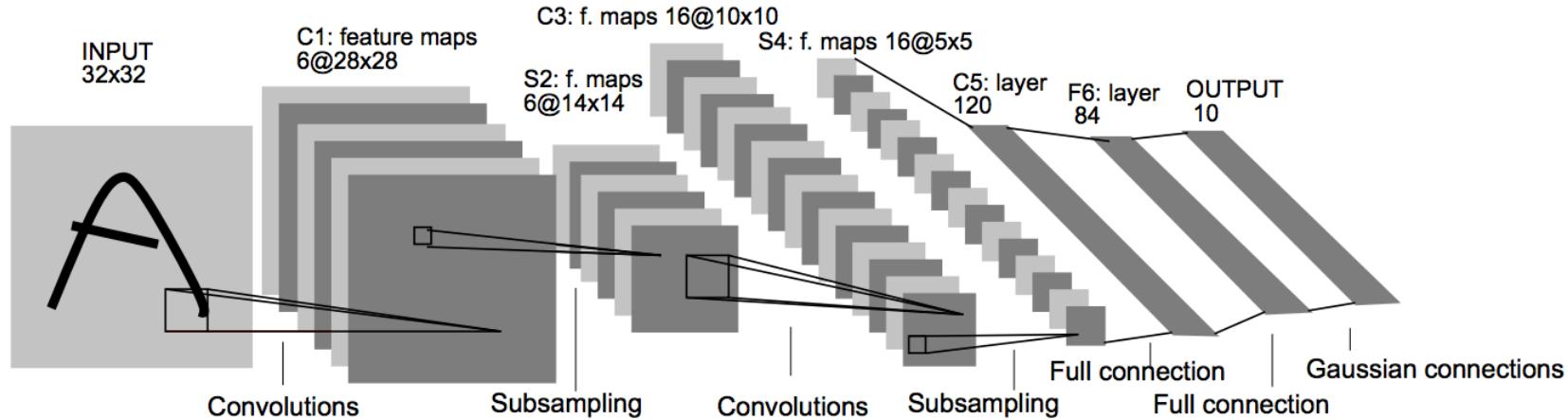


Inputs shifted by 1. Five inputs change but only three outputs change.





Sizing examples



C1 contains six 5x5 conv filters.

Size of feature maps at C1?

Number of parameters in C1 layer?

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

Size of feature maps at S2?

Number of parameters in S2 layer?

C3 contains sixteen 5x5 conv filters.

Size of feature maps at C3?



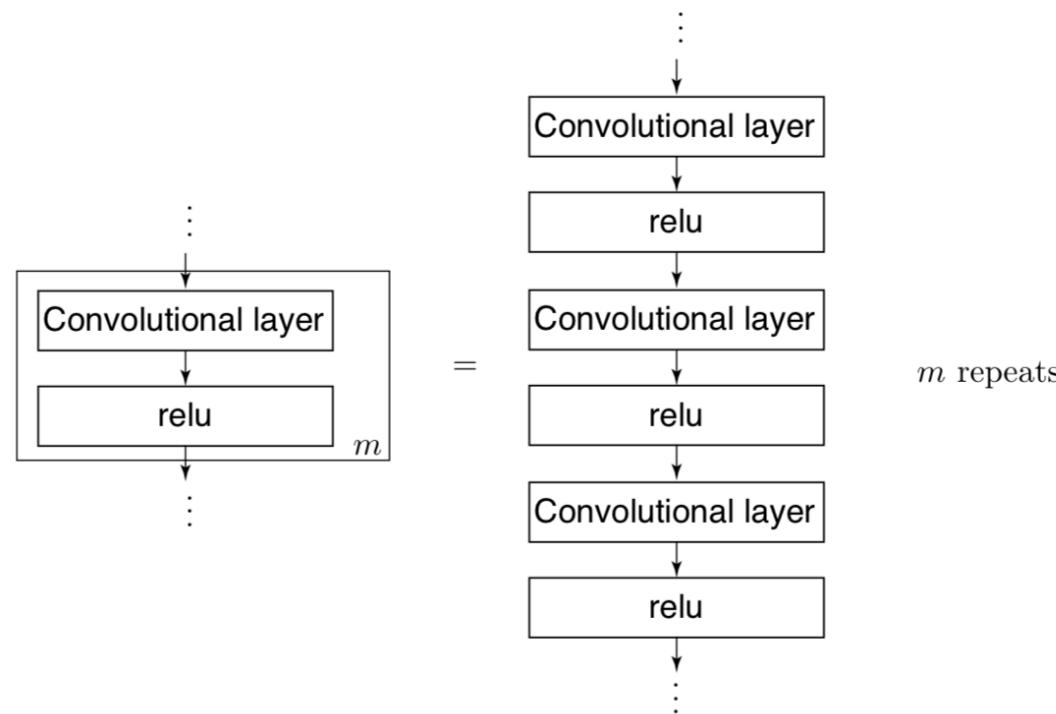
CNN architecture

Convolutional neural network architectures

Typically, convolutional neural networks are comprised of units that are composed of:

- Several paired convolutional-relu layers.
 - Potentially one max pooling layer.

These units are cascaded. Finally, a CNN may end with a fully connected-relu layer, followed by a softmax classifier. To illustrate this, we borrow the plate notation from graphical models, where:

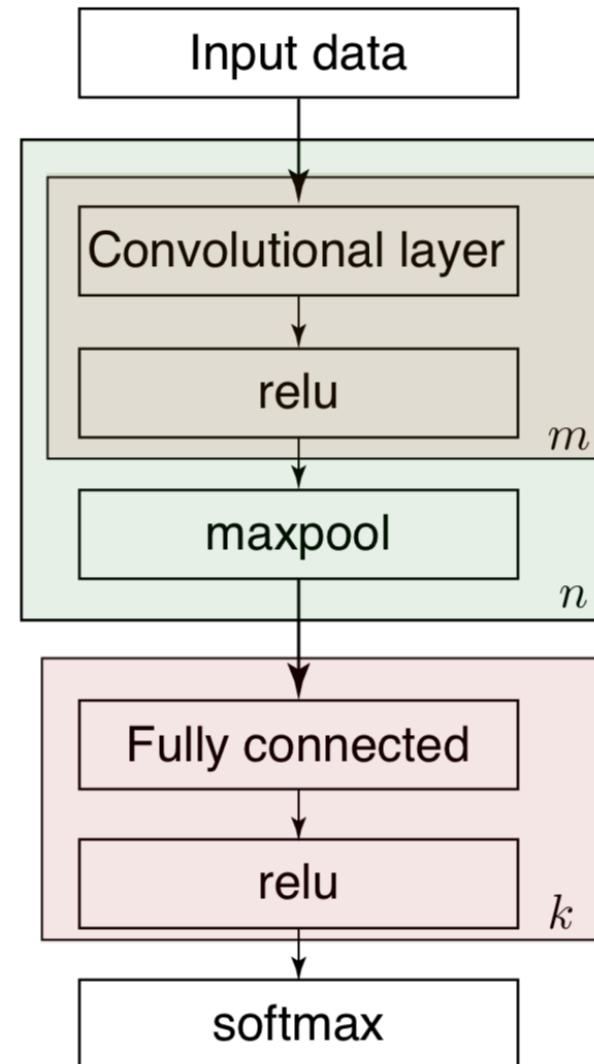




CNN architecture

Convolutional neural network architectures (cont.)

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





Case studies

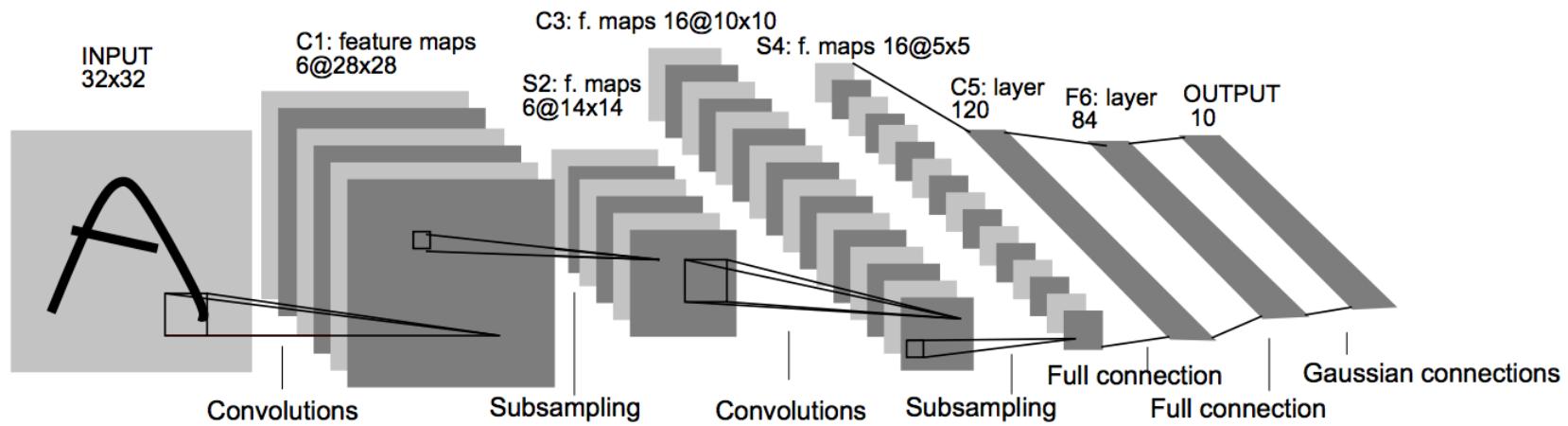
To help get an intuition behind CNN's, we'll go over a few architectures that have been influential in recent years.

Case studies:

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



LeNet-5

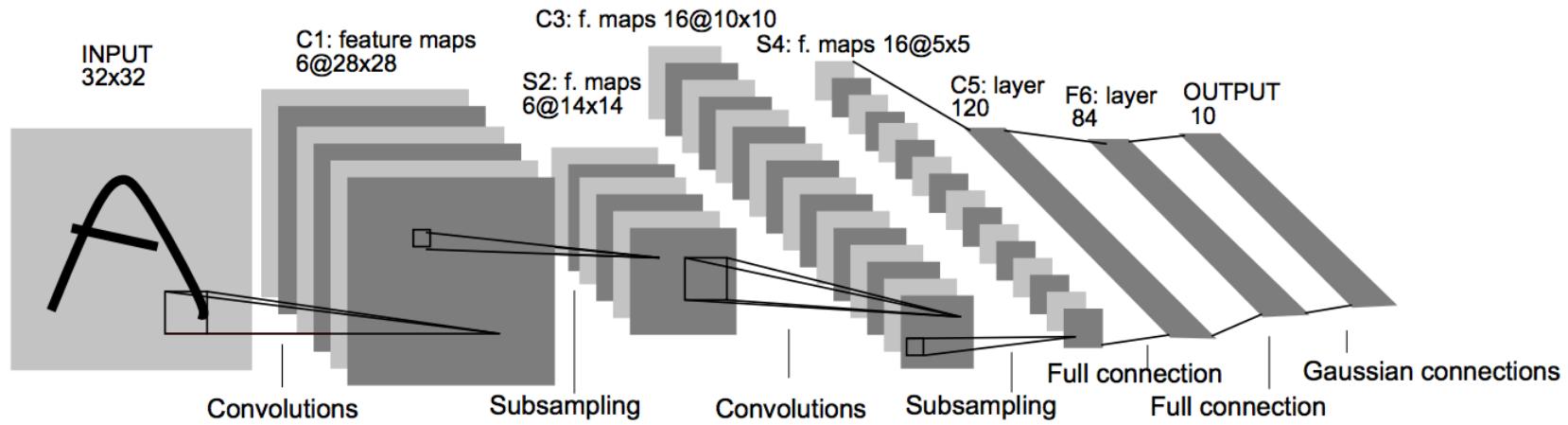


LeCun et al., 1998.

Applied to handwriting recognition.



LeNet-5

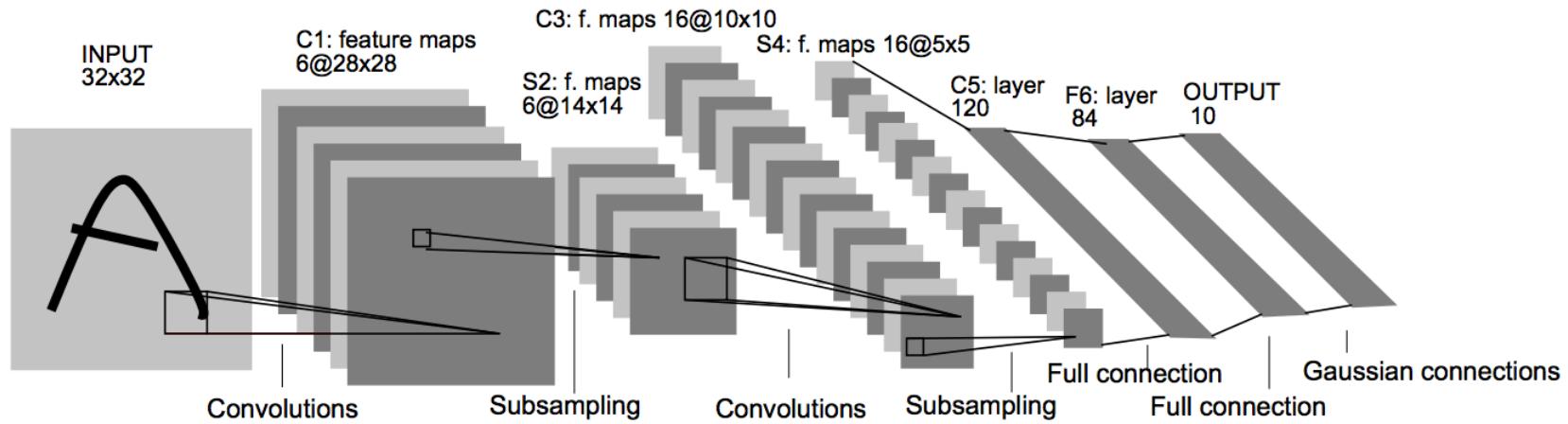


LeCun et al., 1998.

Question: The input is 32x32x3. The first convolutional layer has 6 filters that are 5x5. What is the size of the output of the first convolutional layer?



LeNet-5

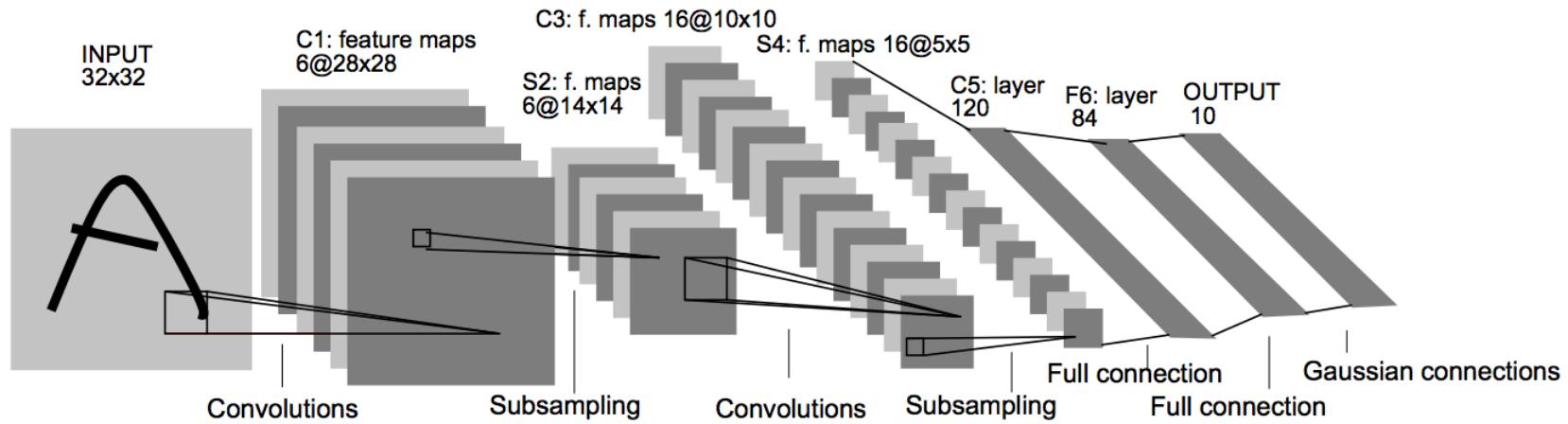


LeCun et al., 1998.

Question: How many trainable parameters are there in the first convolutional layer?



LeNet-5

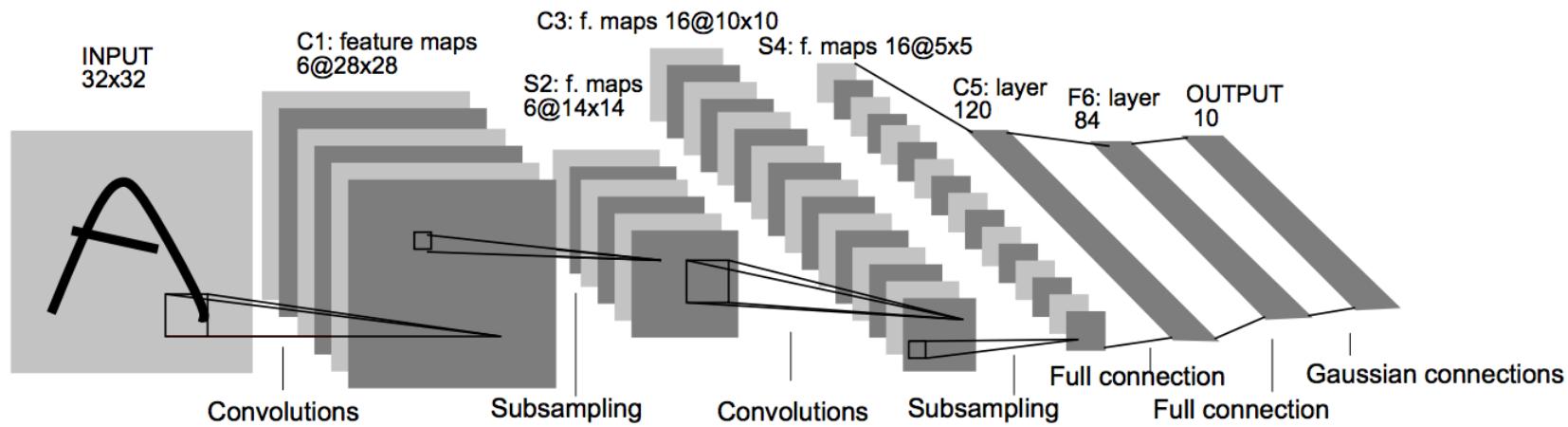


LeCun et al., 1998.

Question: How many connections are there in the first convolutional layer?



LeNet-5

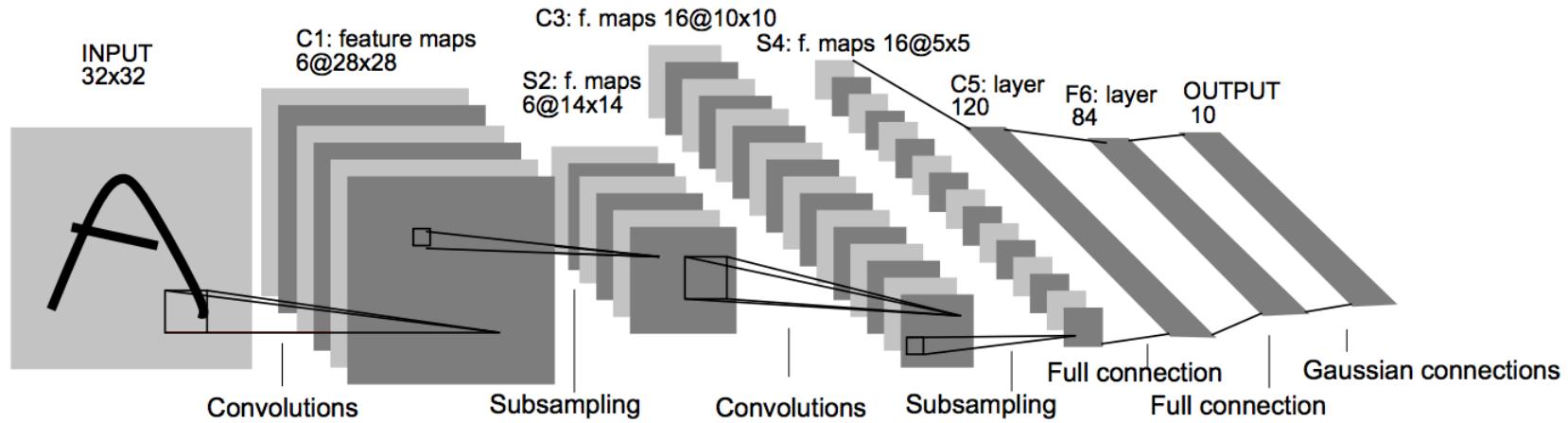


7 total layers. Input is 32x32x3.

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



LeNet-5

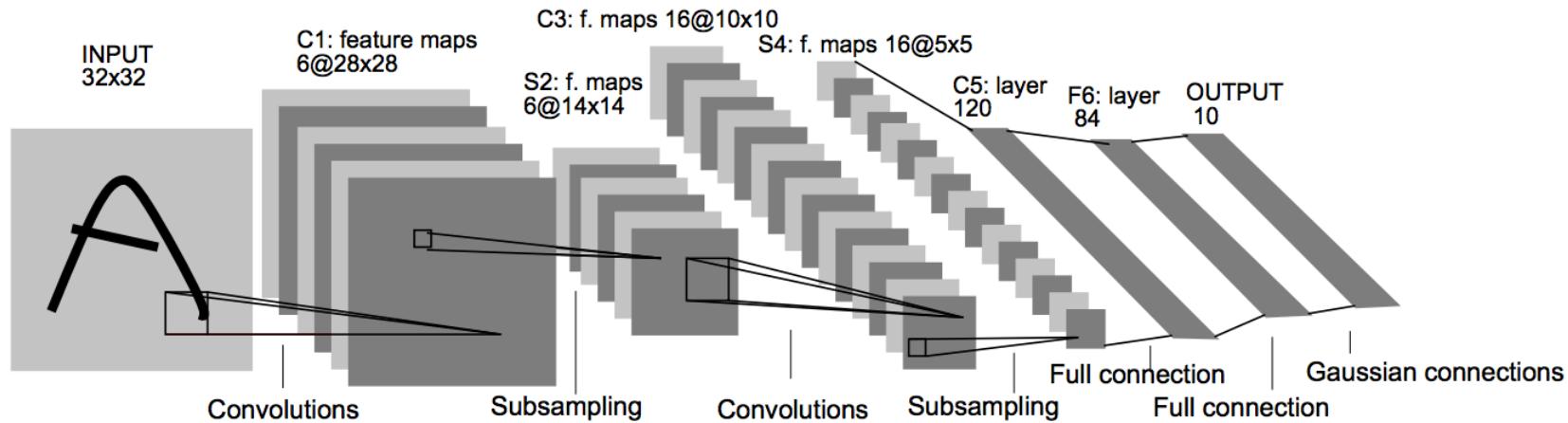


LeCun et al., 1998.

Question: The second layer is a pooling layer with 2×2 filters applied at stride 2. What is the size at the output of this layer?



LeNet-5

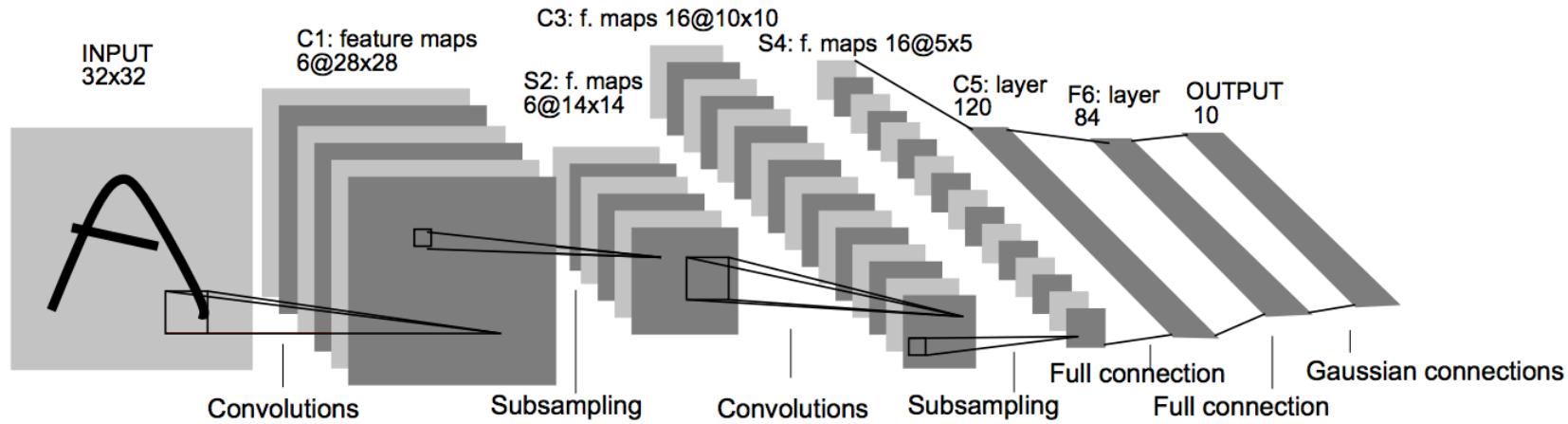


7 total layers. Input is 32x32x3.

1. [28x28x6] **CONV**: 6 convolutional filters, 5x5 features, 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

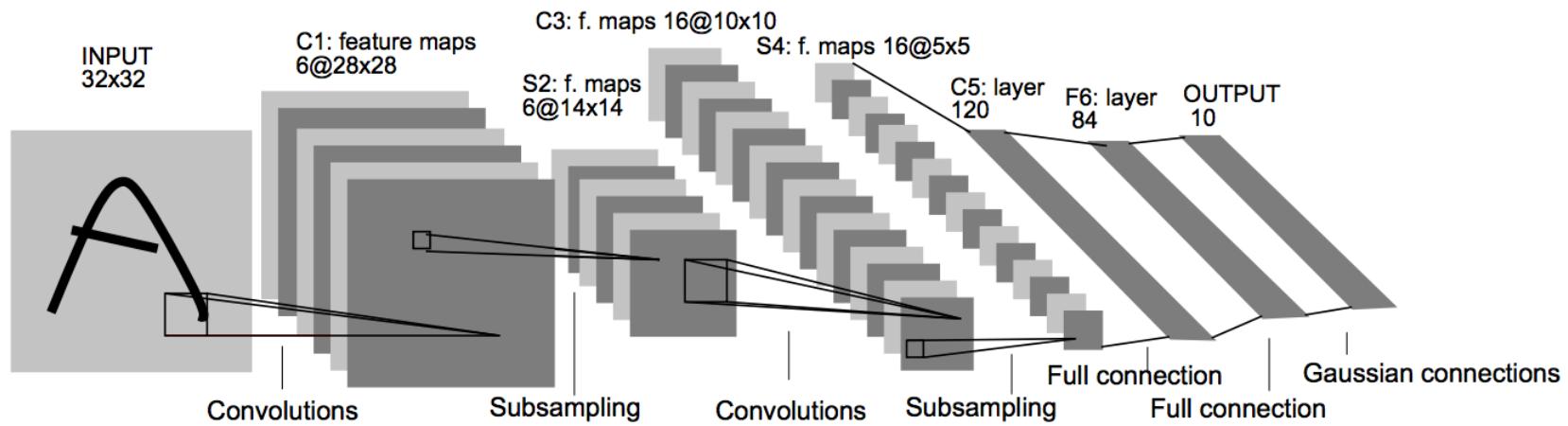


7 total layers. Input is 32x32x3.

1. [28x28x6] **CONV**: 6 convolutional filters, 5x5 features, 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



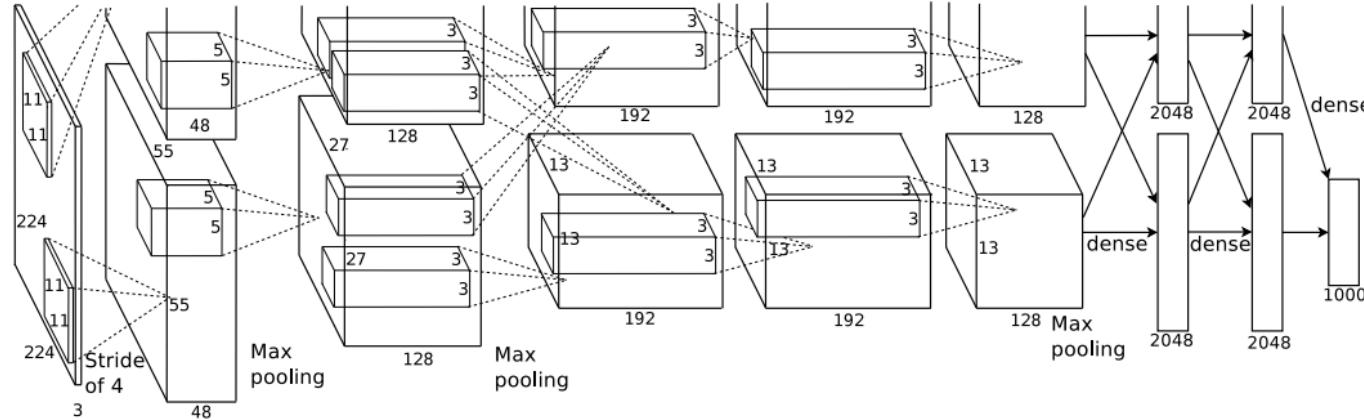
LeCun et al., 1998.

Overall architecture:

[CONV-POOL] $\times 2$ - CONV - FC - OUT



AlexNet



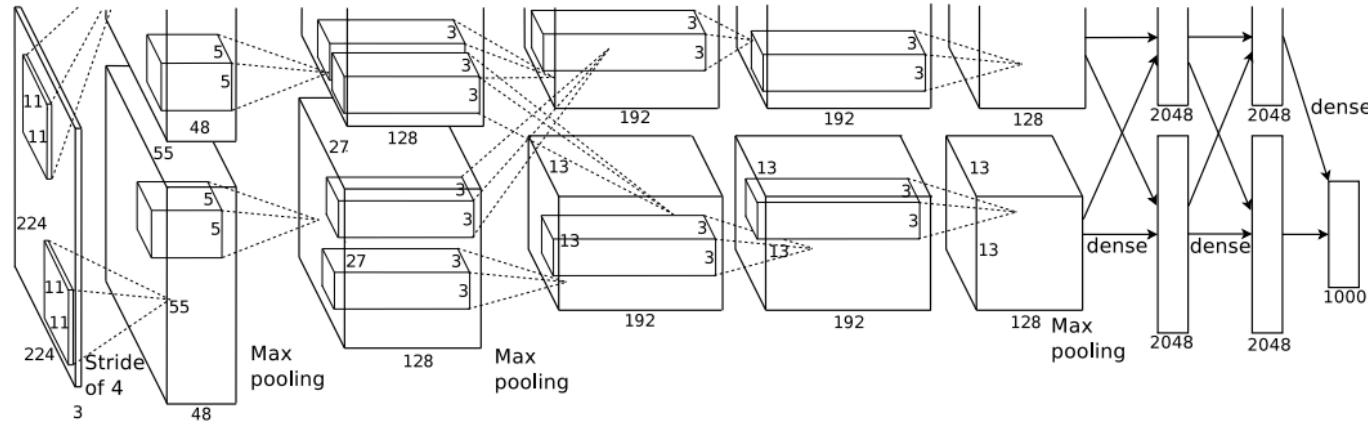
AlexNet, Krizhevsky et al., NIPS 2012.

Input processing:

- ImageNet has variable-sized images.
- Downsampled 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



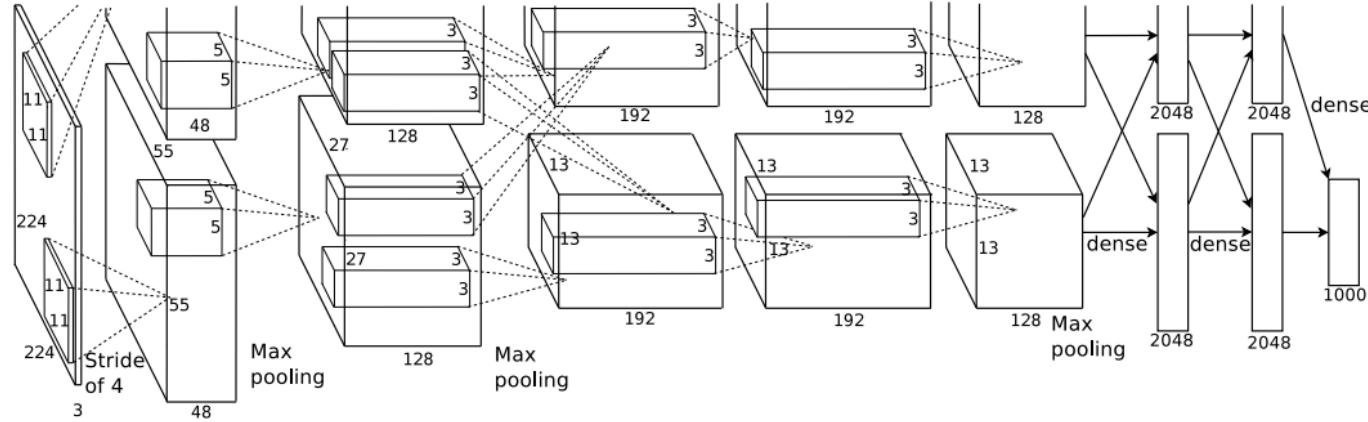
AlexNet, Krizhevsky et al., NIPS 2012.

Nonlinearity:

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



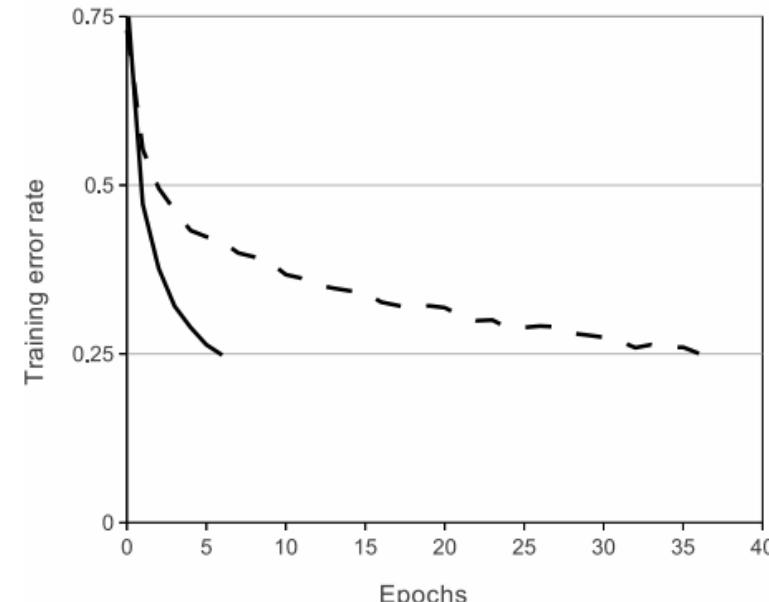
AlexNet



AlexNet, Krizhevsky et al., NIPS 2012.

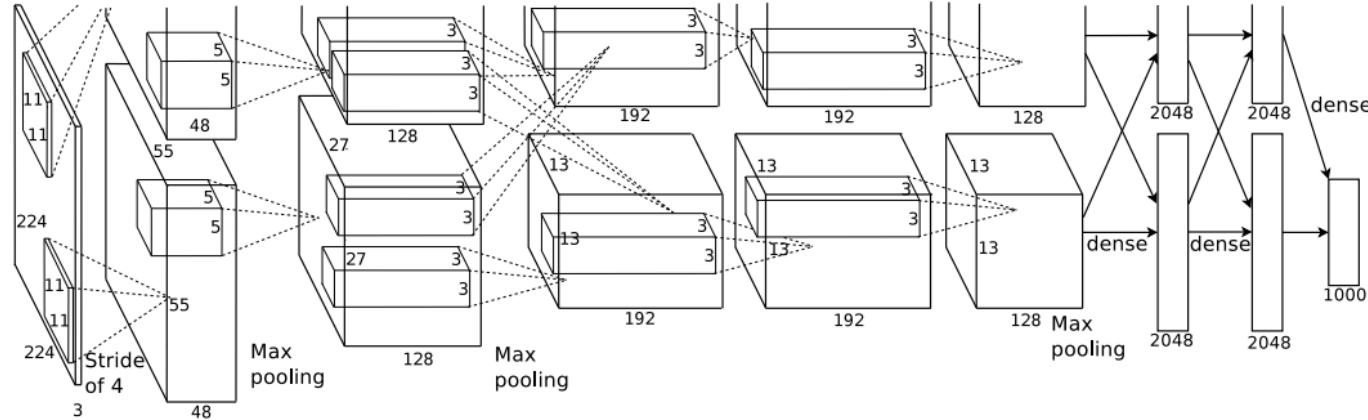
Dotted line is tanh
Solid line is ReLU
Clearly ReLU resulted in more efficient training.

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





AlexNet



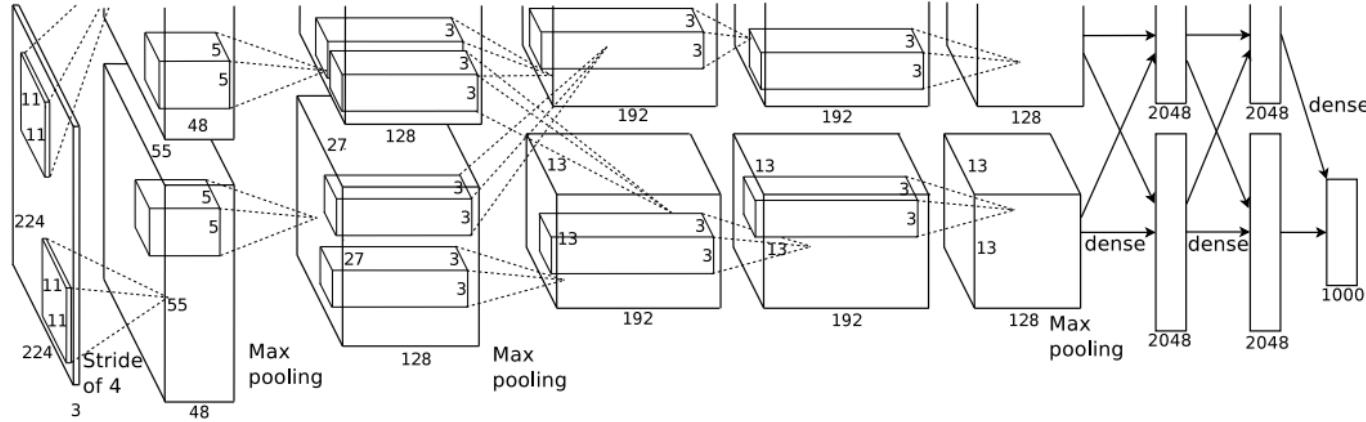
AlexNet, Krizhevsky et al., NIPS 2012.

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.
- Why do you think they trained on two GPUs?



AlexNet

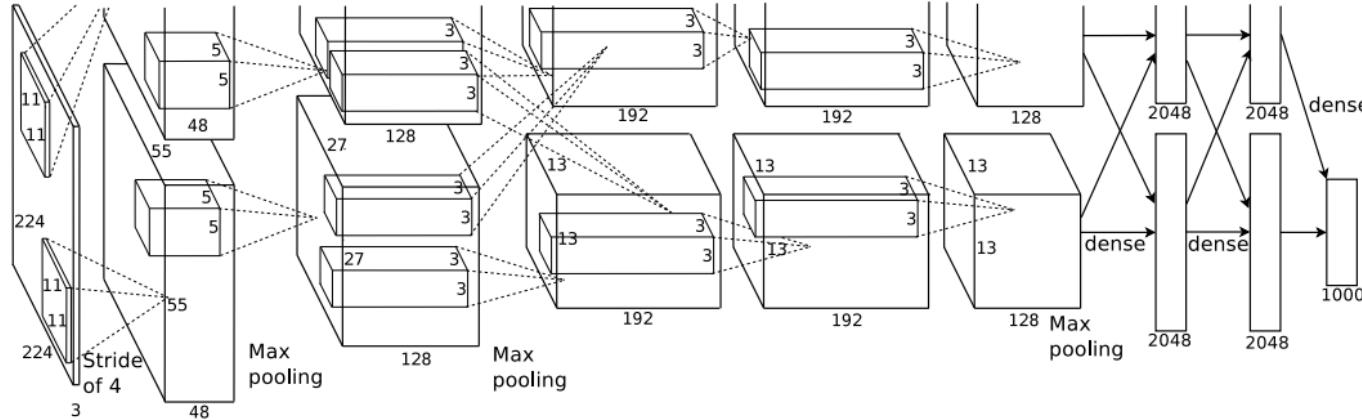


AlexNet, Krizhevsky et al., NIPS 2012.

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



AlexNet

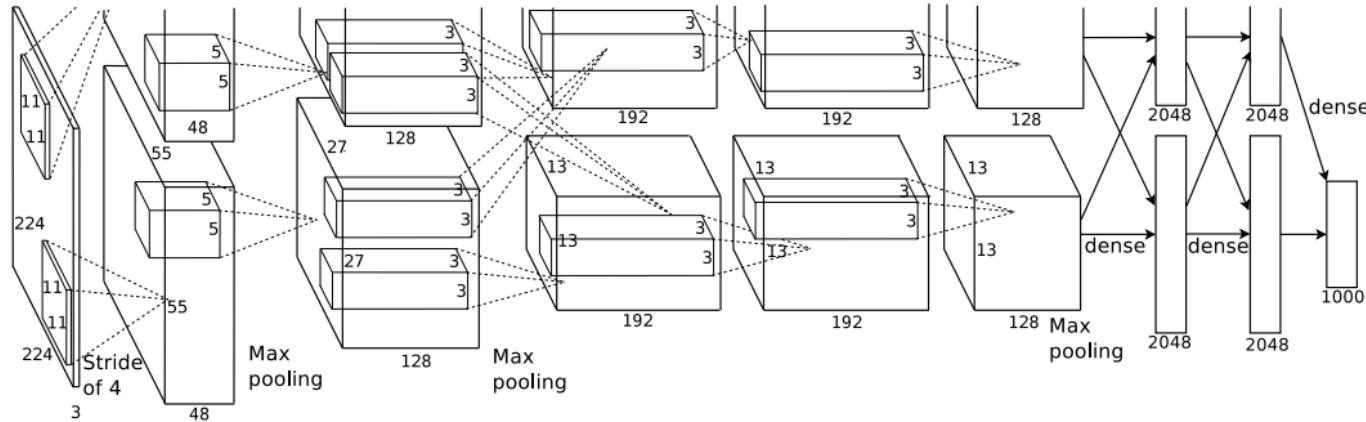


AlexNet, Krizhevsky et al., NIPS 2012.

- Data augmentation:
 - Image translations and horizontal reflections.
 - Extract out random 224×224 patches and their horizontal reflections.
 - At test time, extract 5 random 224×224 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.

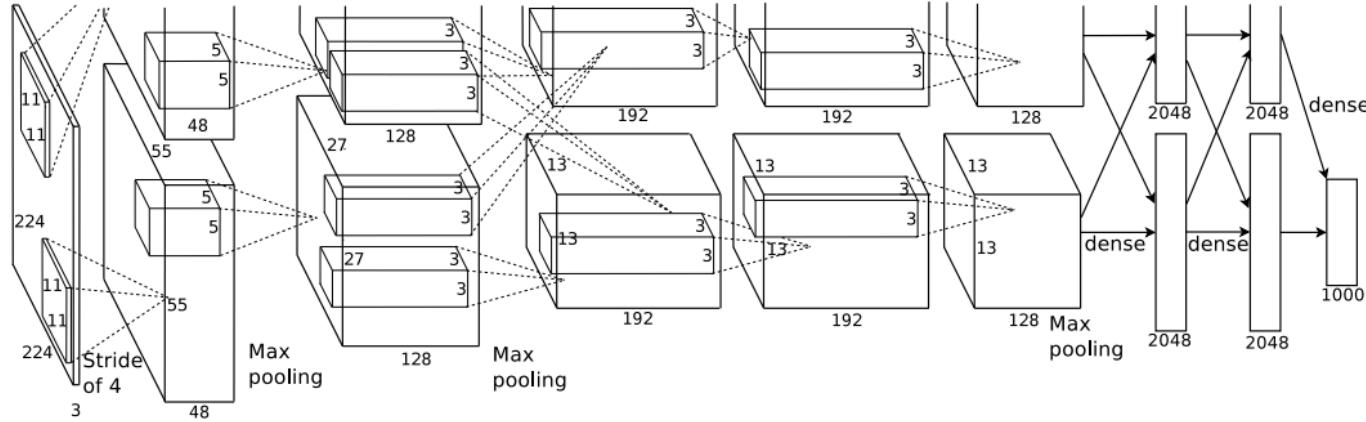
- SGD with momentum and weight decay.
 - Batch size: 128, momentum: 0.9
 - Learning rate initialized to 0.01, manually decreased when validation error stopped improving.
 - L2 weight decay: 0.0005

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

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



AlexNet



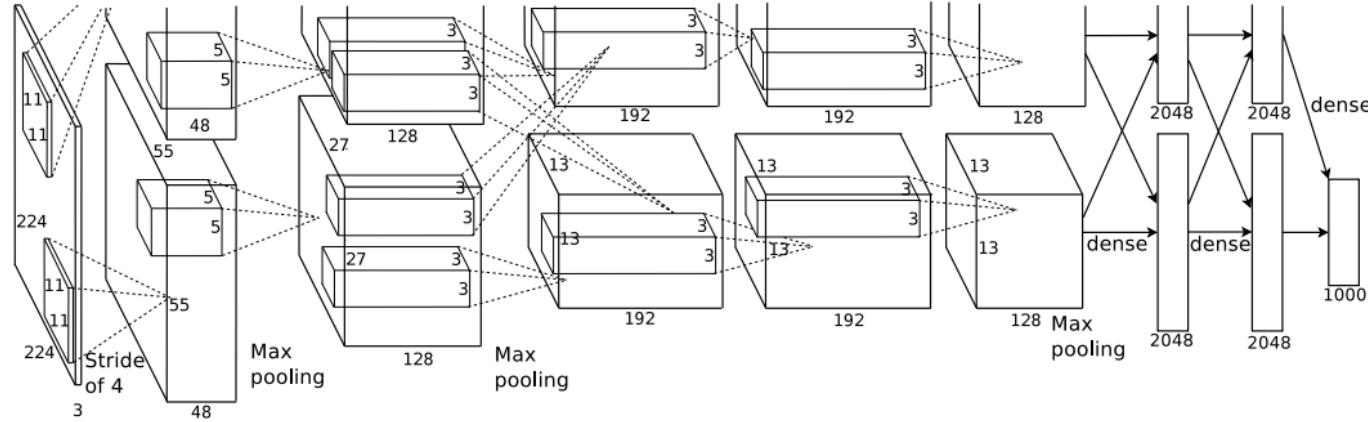
AlexNet, Krizhevsky et al., NIPS 2012.

- 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

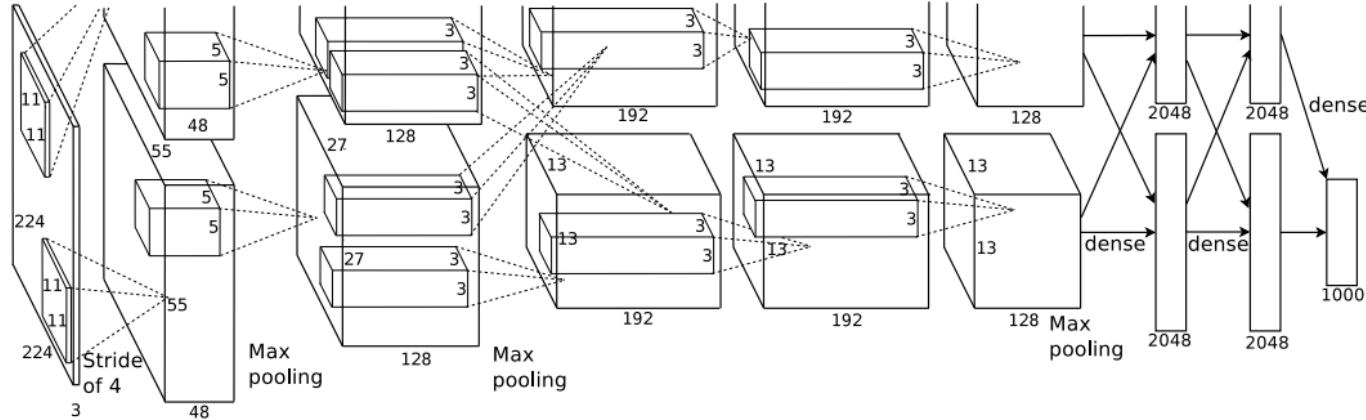


AlexNet, Krizhevsky et al., NIPS 2012.

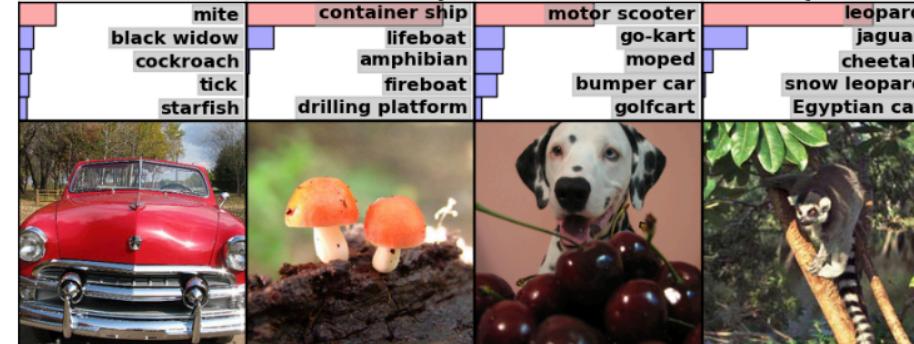
- Averaged the output of multiple CNNs. Validation error of ...
 - 1 CNN: 18.2%
 - 5 CNNs: 16.4%
 - 7 CNNs: 15.4% (was also pre-trained)



AlexNet



mite container ship motor scooter leopard

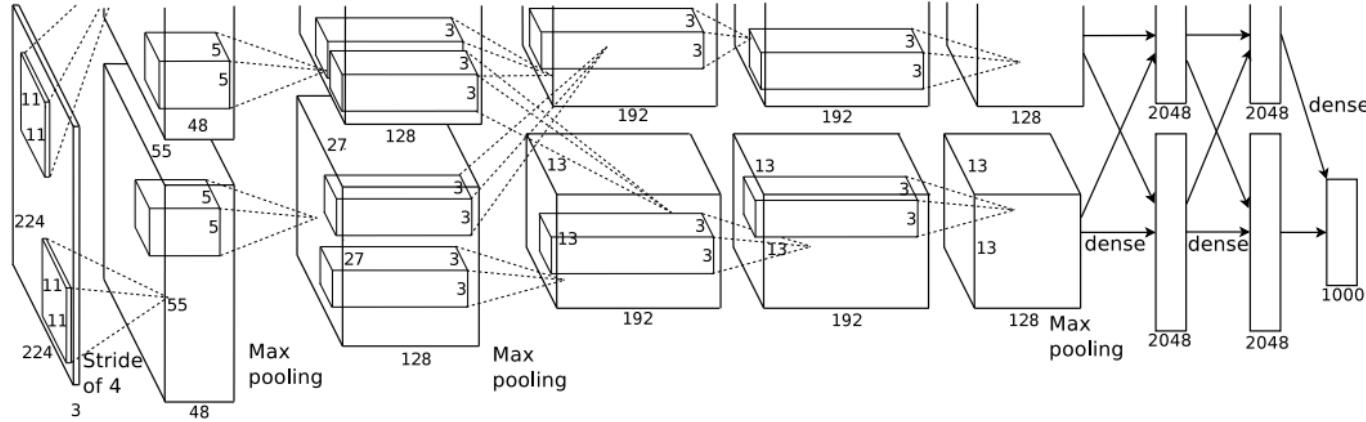


grille mushroom cherry Madagascar cat

convertible	agaric	dalmatian	squirrel monkey
grille	mushroom	grape	spider monkey
pickup	jelly fungus	elderberry	titi
beach wagon	gill fungus	ffordshire bullterrier	indri
fire engine	dead-man's-fingers	currant	howler monkey



AlexNet

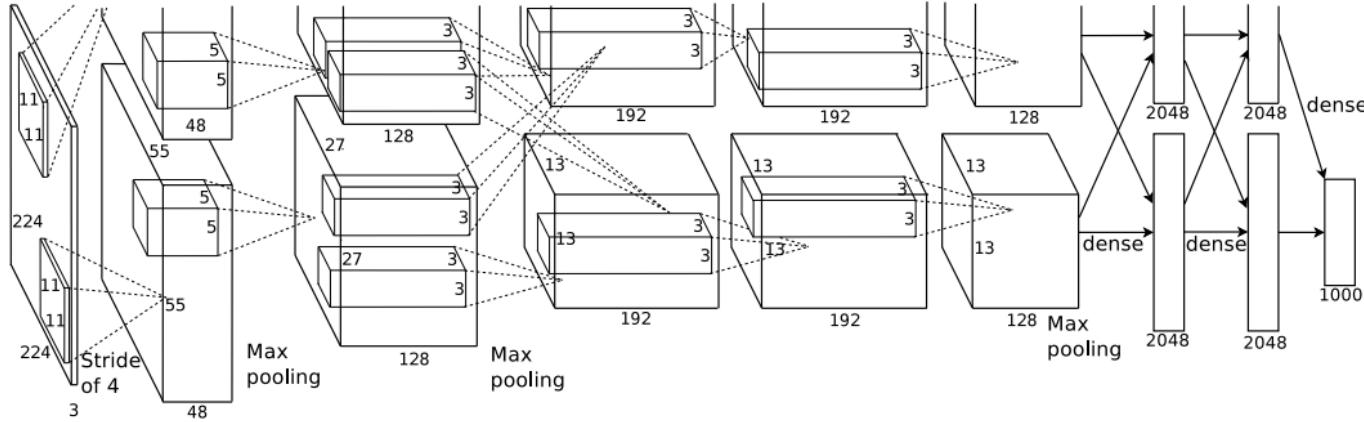


AlexNet, Krizhevsky et al., NIPS 2012.

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



AlexNet

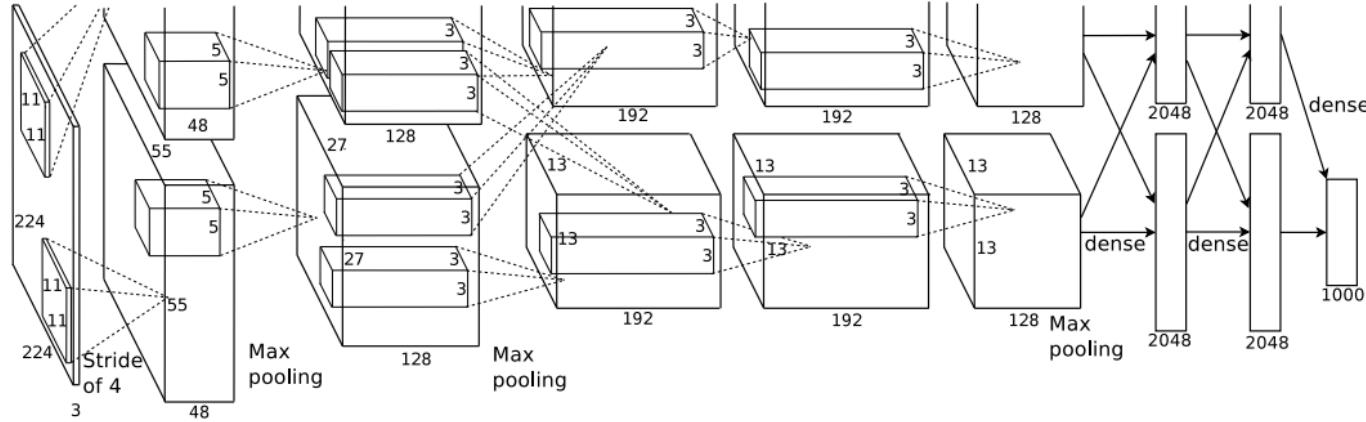


Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

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



AlexNet

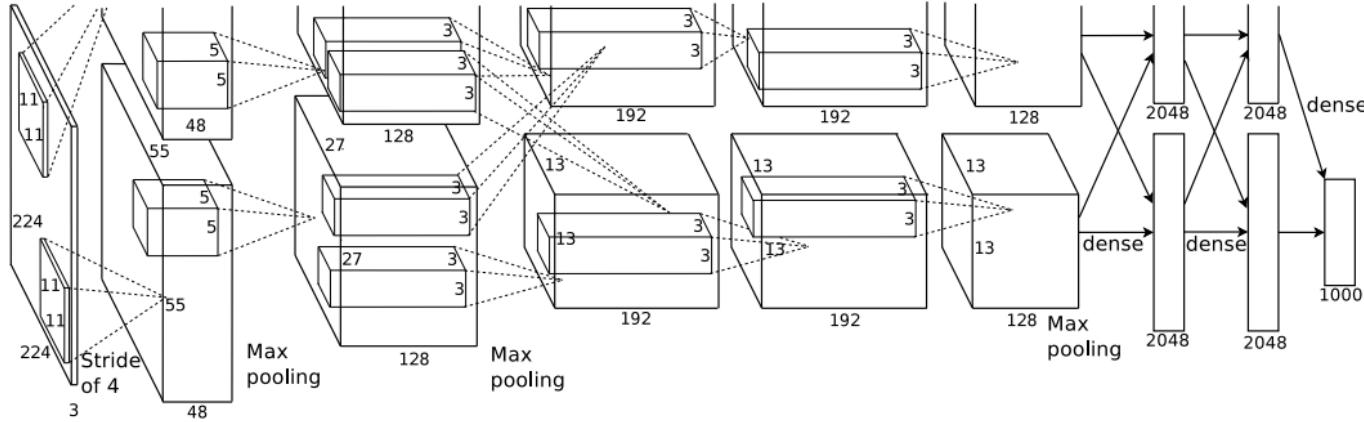


Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

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



AlexNet

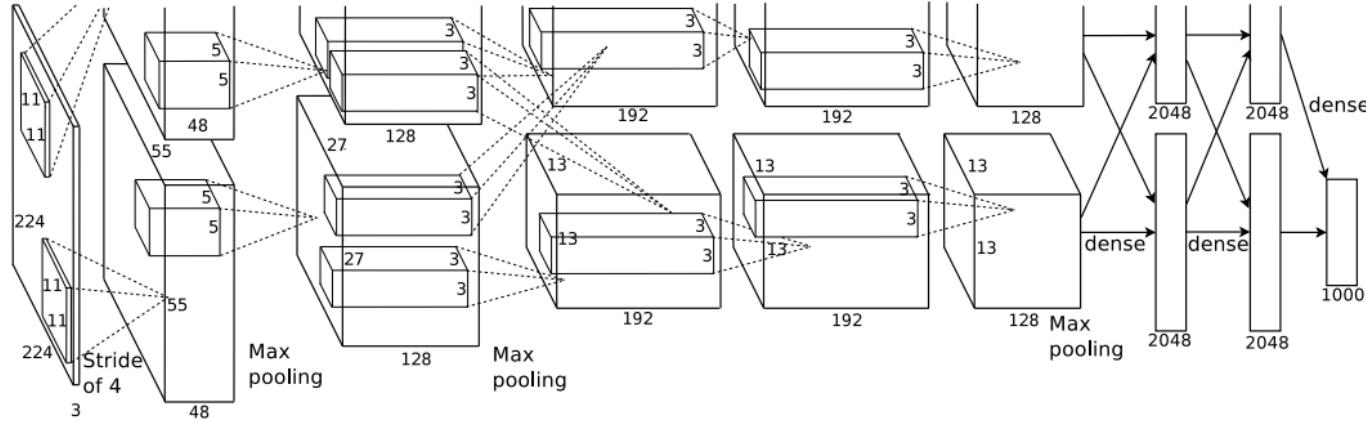


Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

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



AlexNet

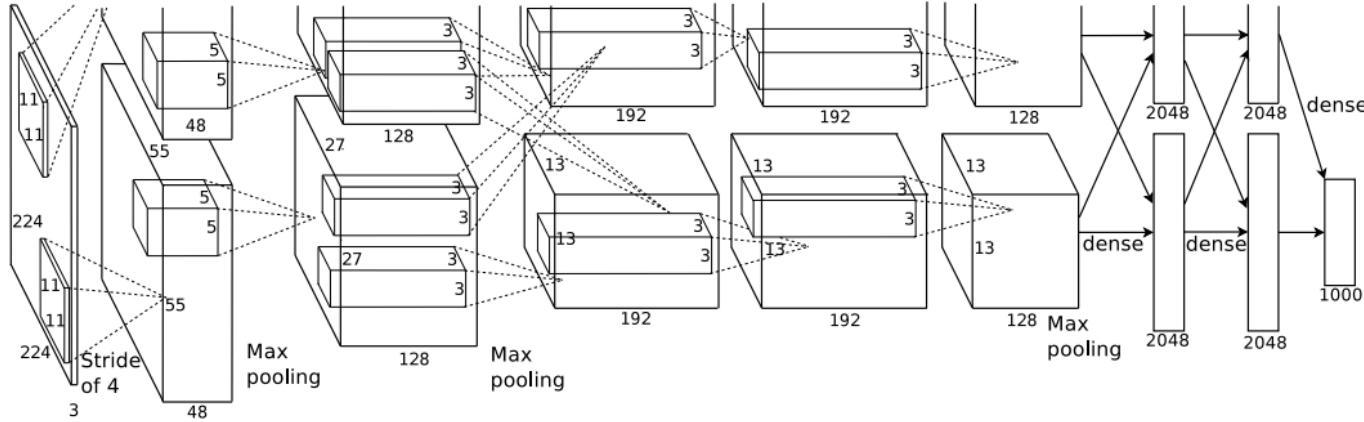


Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

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

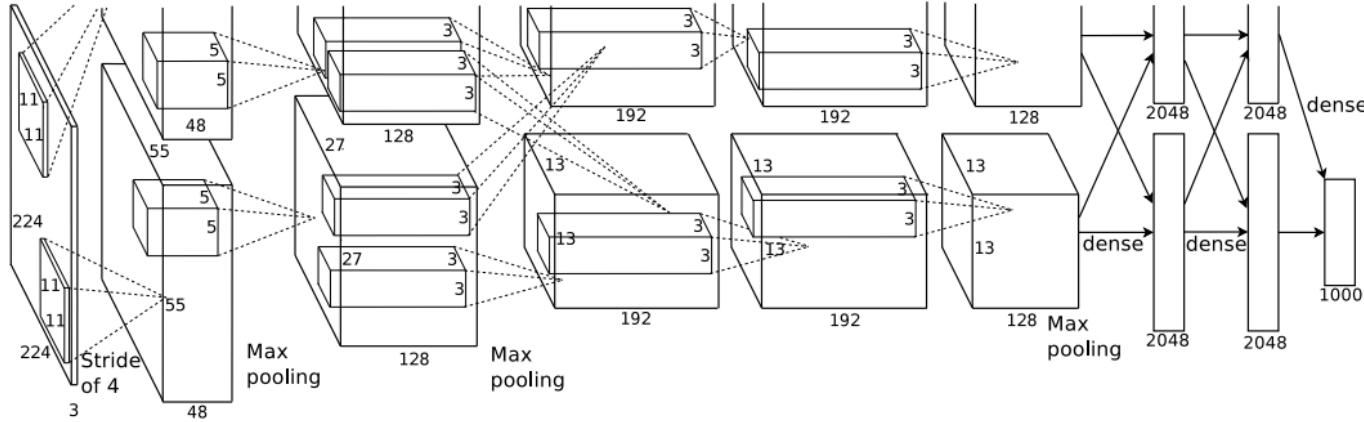


Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

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



AlexNet

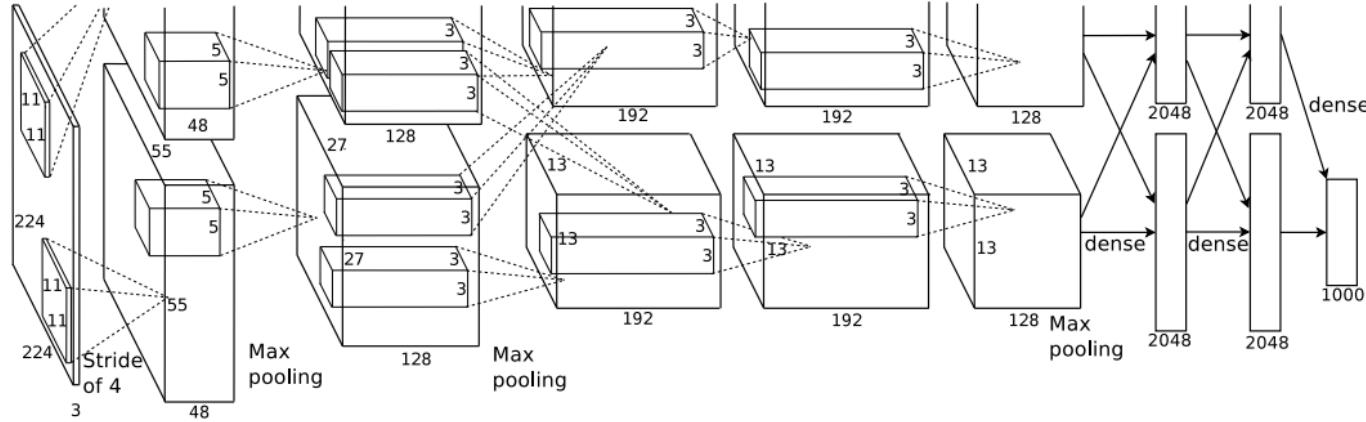


Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

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

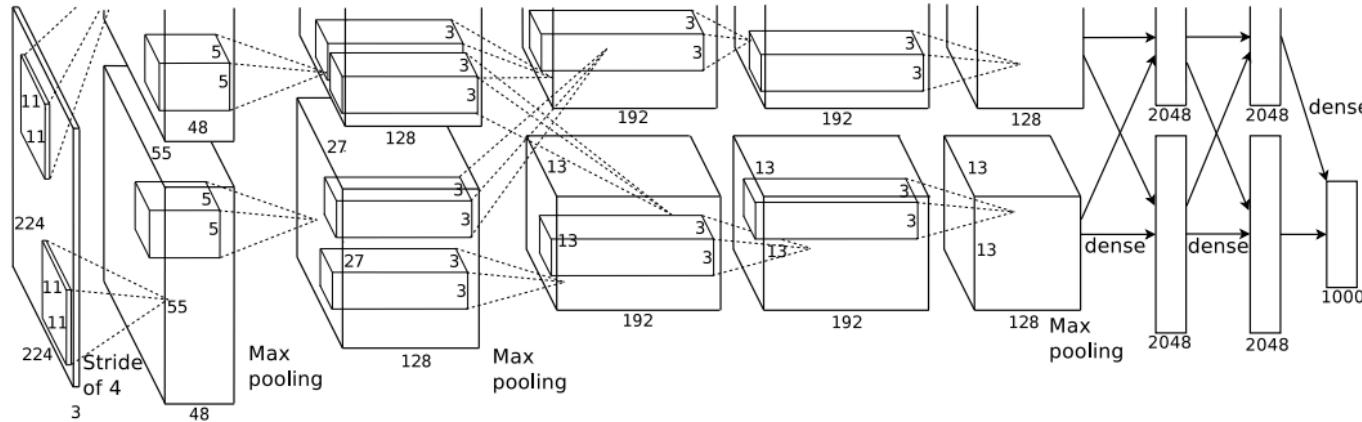


Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

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 5x5x48 with stride 1, pad 2.
5. [13x13x256] **POOL**: 3x3 filters with stride 2.



AlexNet

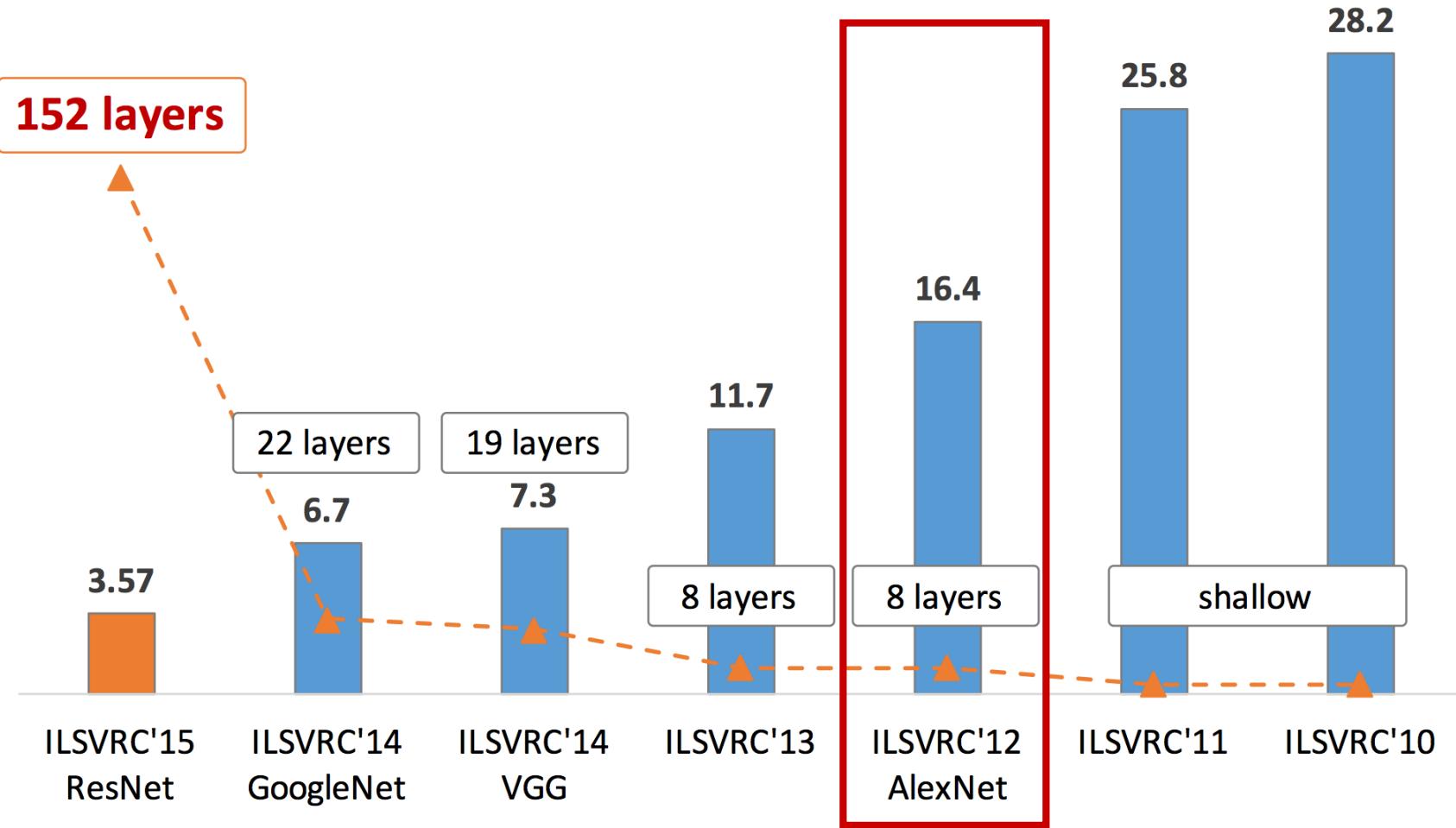
Architecture: 8 layers. Input is 227x227x3 (in paper, 224x224x3; numbers were changed so the operations work out).

1. [55x55x96] **CONV**: 96 filters of size 11x11x3 with stride 4.
2. [27x27x96] **POOL**: 3x3 filters with stride 2.
3. [27x27x96] **NORM**: normalization layer
4. [27x27x256] **CONV**: 256 filters of size 5x5x48 with stride 1, pad 2.
5. [13x13x256] **POOL**: 3x3 filters with stride 2.
6. [27x27x96] **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



AlexNet in context

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

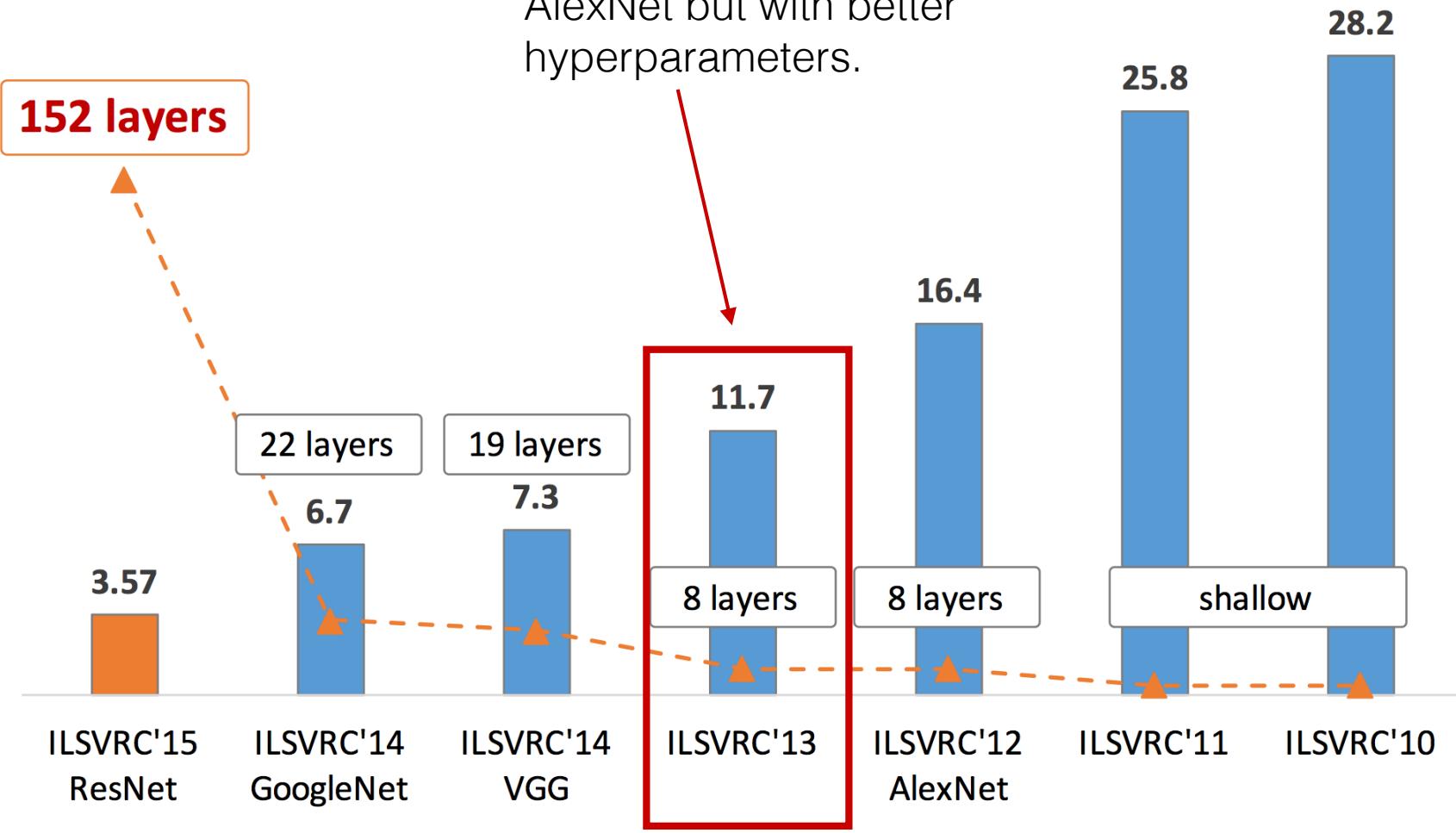


http://kaiminghe.com/icml16tutorial/icml2016Tutorial_deep_residual_networks_kaiminghe.pdf



ZFNet

ZFNet, which was AlexNet but with better hyperparameters.



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).

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?

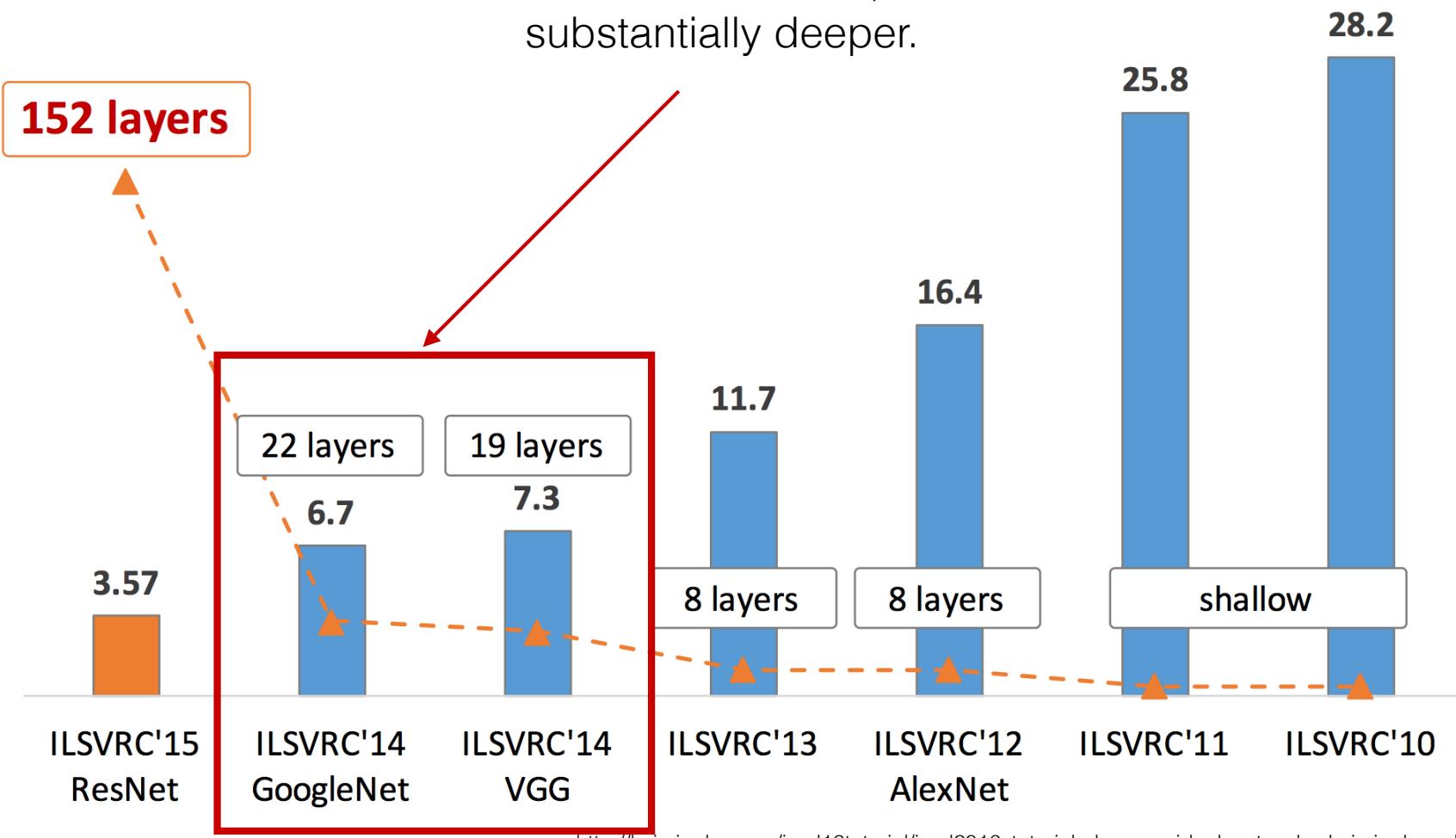
We'll unpack this more later on, but some observations from ZFNet are:

- 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?

New architectures, that are substantially deeper.





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 (3×3) 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.

Picture: cs231n.stanford.edu



VGGNet

Small filters and depth:

What might be a con of using a small filter, and how does VGGNet address this?

Picture: cs231n.stanford.edu



VGGNet

Small filters and depth:

Why might this turn into a good thing?

Picture: cs231n.stanford.edu



VGGNet

Small filters and depth:

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

Picture: cs231n.stanford.edu



VGGNet

Small filters and depth:

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

Picture: cs231n.stanford.edu



VGGNet

INPUT	[224x224x3]
CONV (64)	[224x224x64]
CONV (64)	[224x224x64]
POOL	[112x112x64]
CONV (128)	[224x224x128]
CONV (128)	[224x224x128]
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]



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)	[224x224x128]	112*112*128	~ 1.6M
CONV (128)	[224x224x128]	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



VGGNet

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)	[224x224x128]	112*112*128	~ 1.6M	$(3*3*64)*128 = 73,728$
CONV (128)	[224x224x128]	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

Some observations:

- Total memory: $24M * 4 \text{ bytes} = 96\text{MB}$ 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



VGGNet

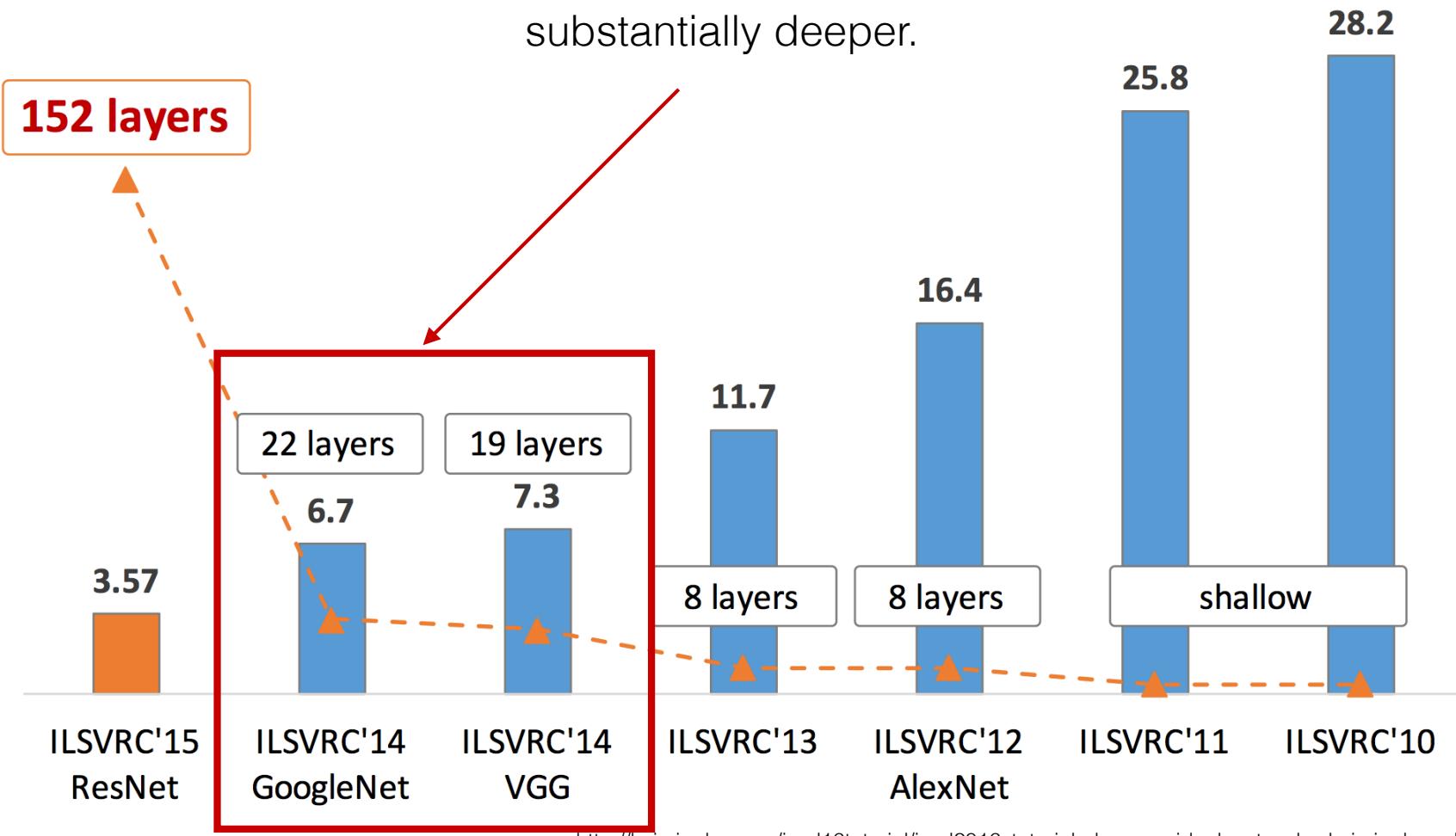
Other implementation notes about VGGNet.

- 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 randomized a shallower network and then used the weights there as the initial weights for deeper networks.
 - But later on they found out the Xavier initialization was fine.
- Also performed the horizontal flipping, random crops, and RGB shifting that Krizhevsky and others used.
- Training took 2-3 weeks on a 4 GPU machine.
- Their submission averaged the output of 7 nets.



What about depth?

New architectures, that are substantially deeper.





GoogLeNet

From Szegedy et al., IEEE CVPR 2014.

The main take-home points of the GoogLeNet:

- 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

Big question: why not just go deeper and use more neurons?

- This increases the number of parameters.
 - More overfitting, esp. if dataset sizes are not large.
 - More learning required.
- Dramatically increases the computational expenses. Further, some of this computation may be wasted (e.g., if a ReLU unit is dead).

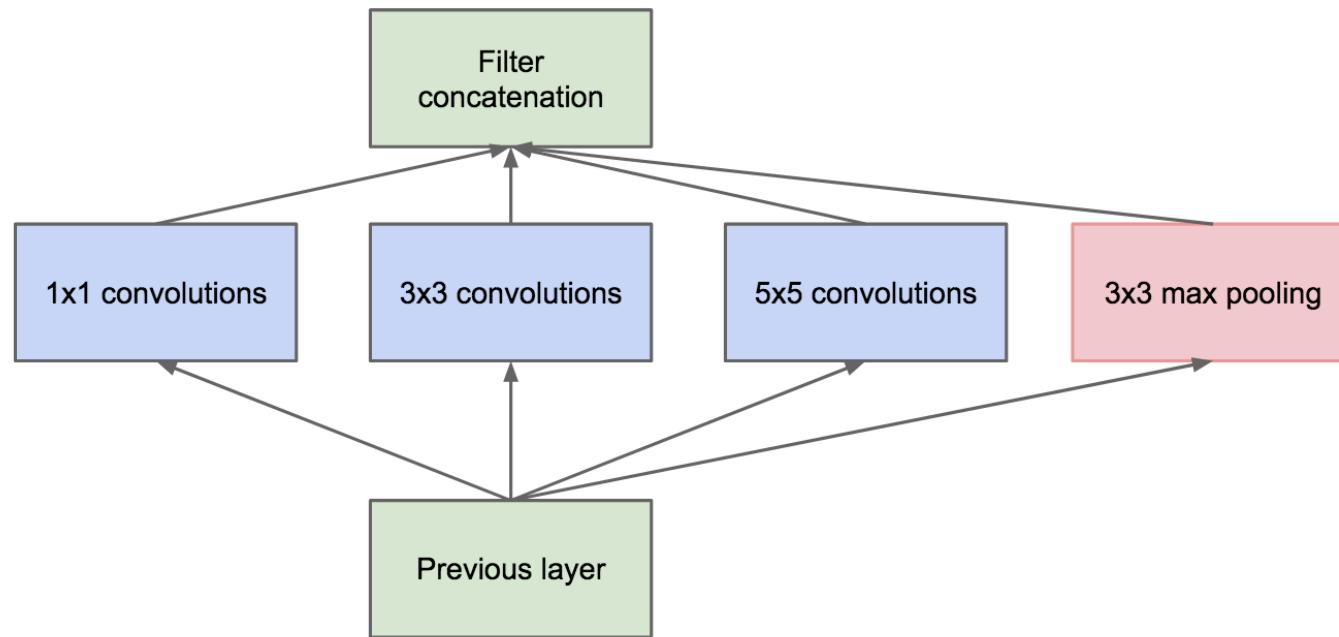
Idea to address these limitations: the inception module.



GoogLeNet

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

Naive inception module:

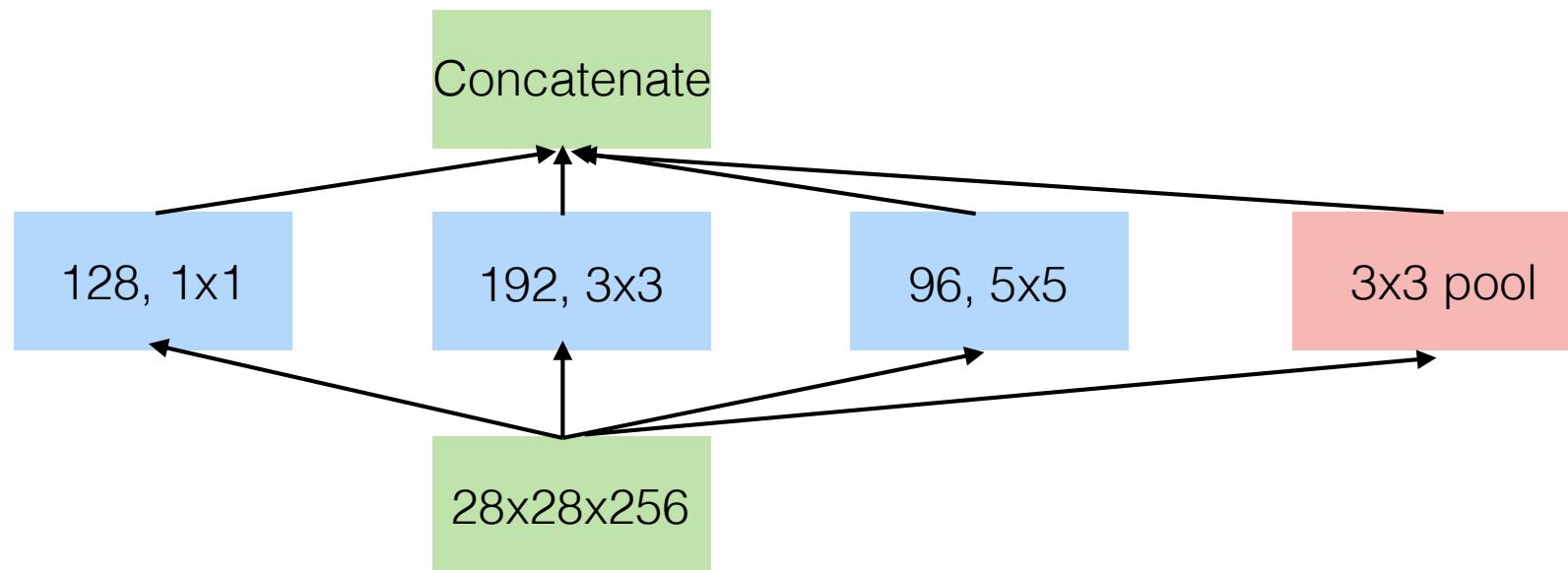




GoogLeNet

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

Naive inception module:



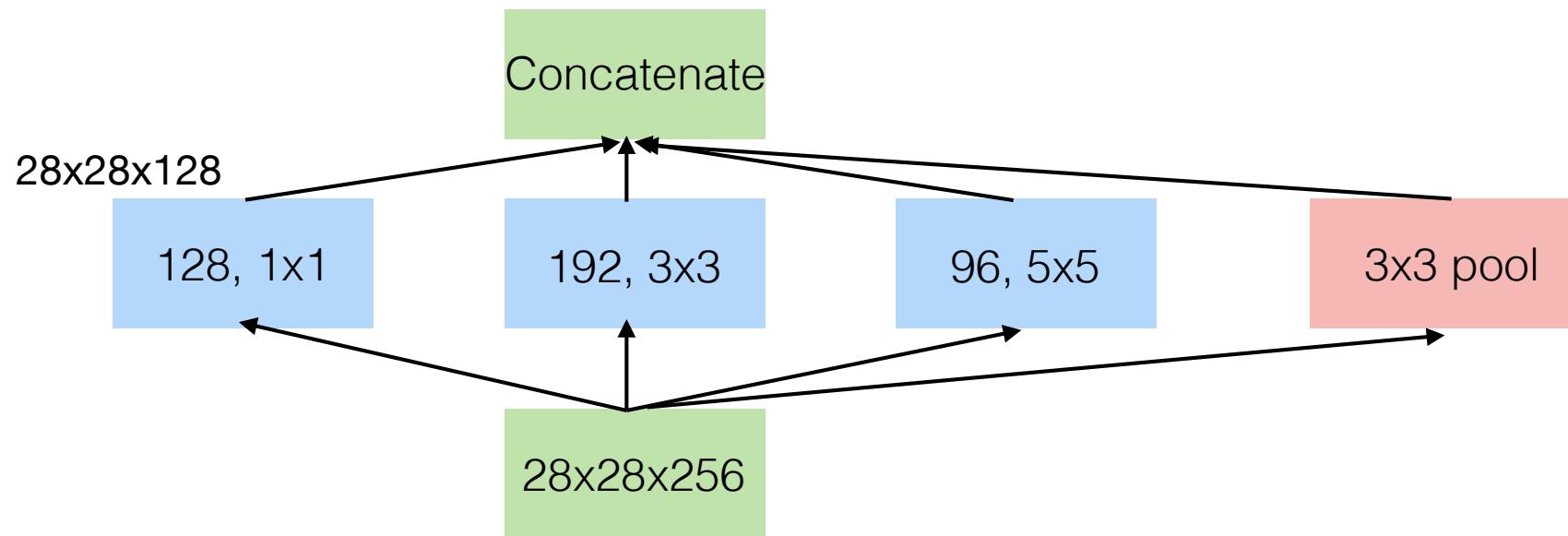
What is the size of the output of the 128 1x1 convolutions?



GoogLeNet

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

Naive inception module:



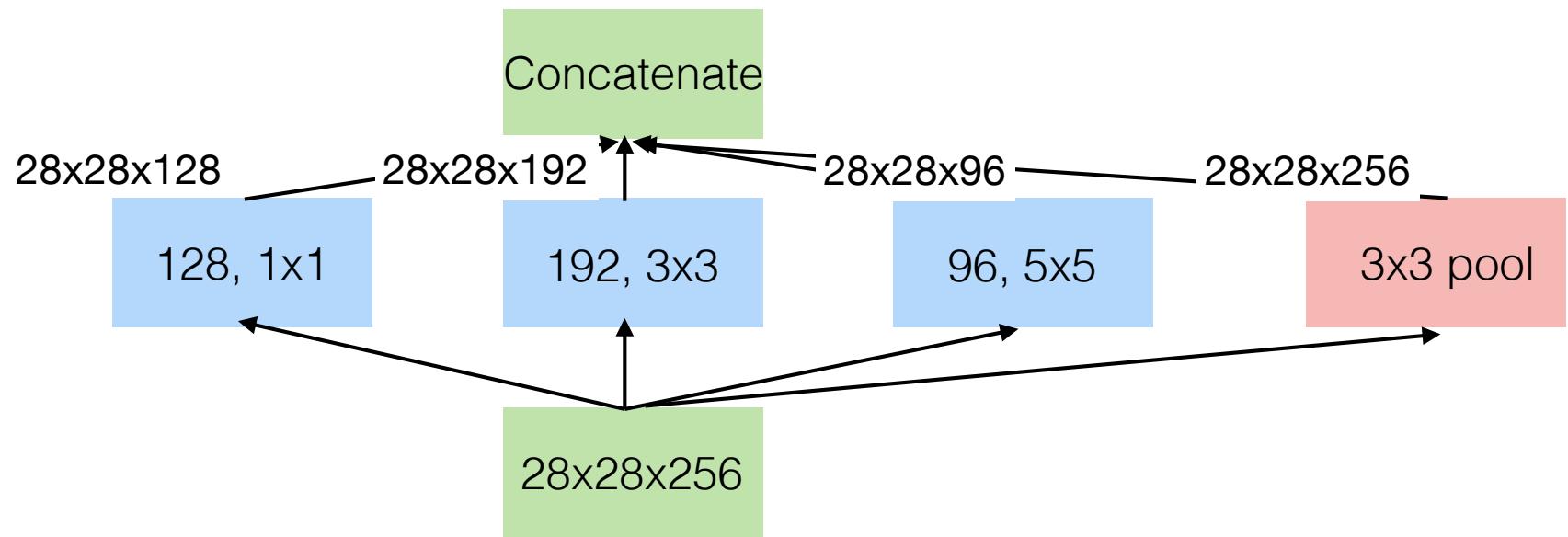
What padding do we need to keep the output size consistent for the $192, 3 \times 3$ convolutional filters?



GoogLeNet

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

Naive inception module:

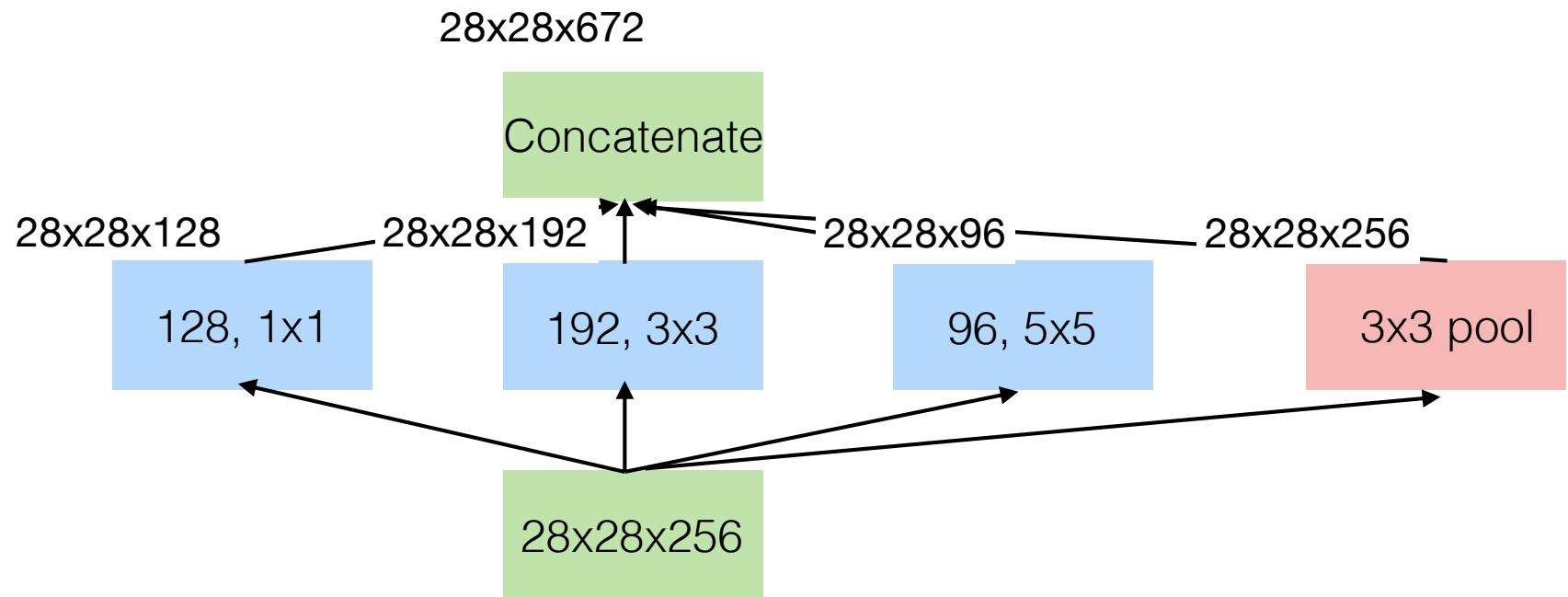




GoogLeNet

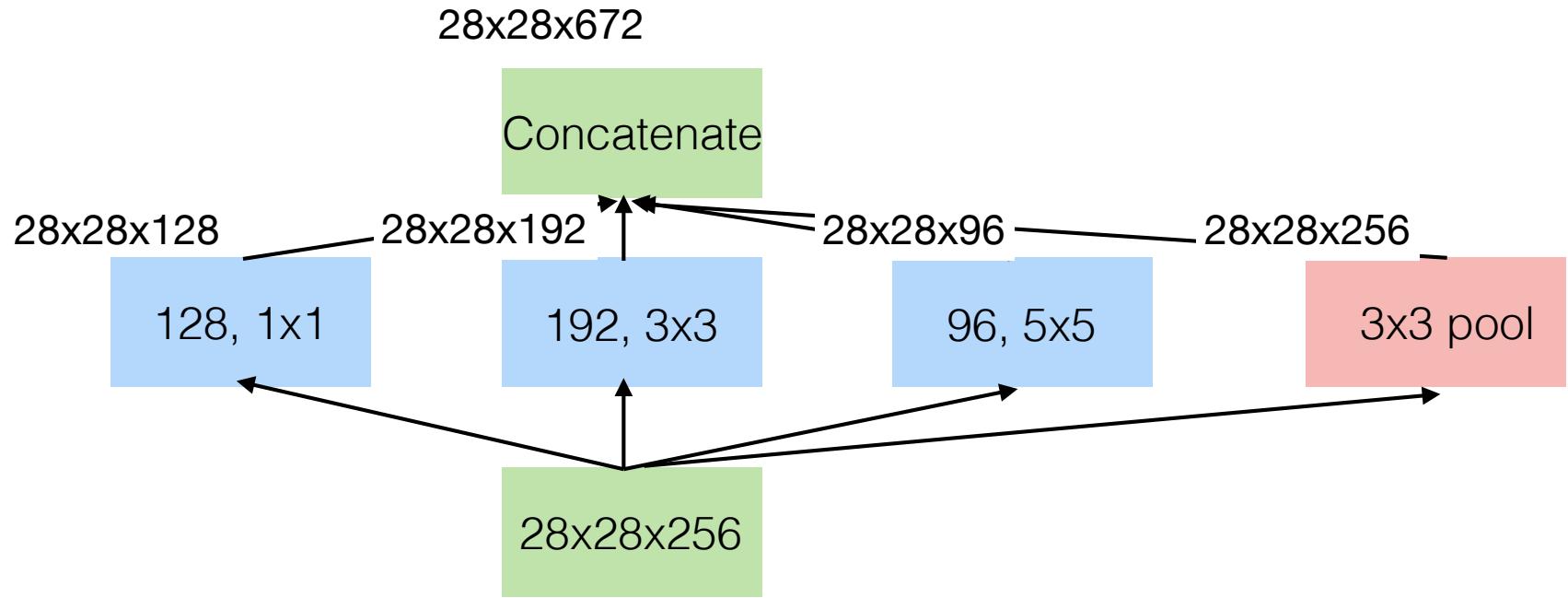
They leverage an idea called “network-in-network.”

Naive inception module:





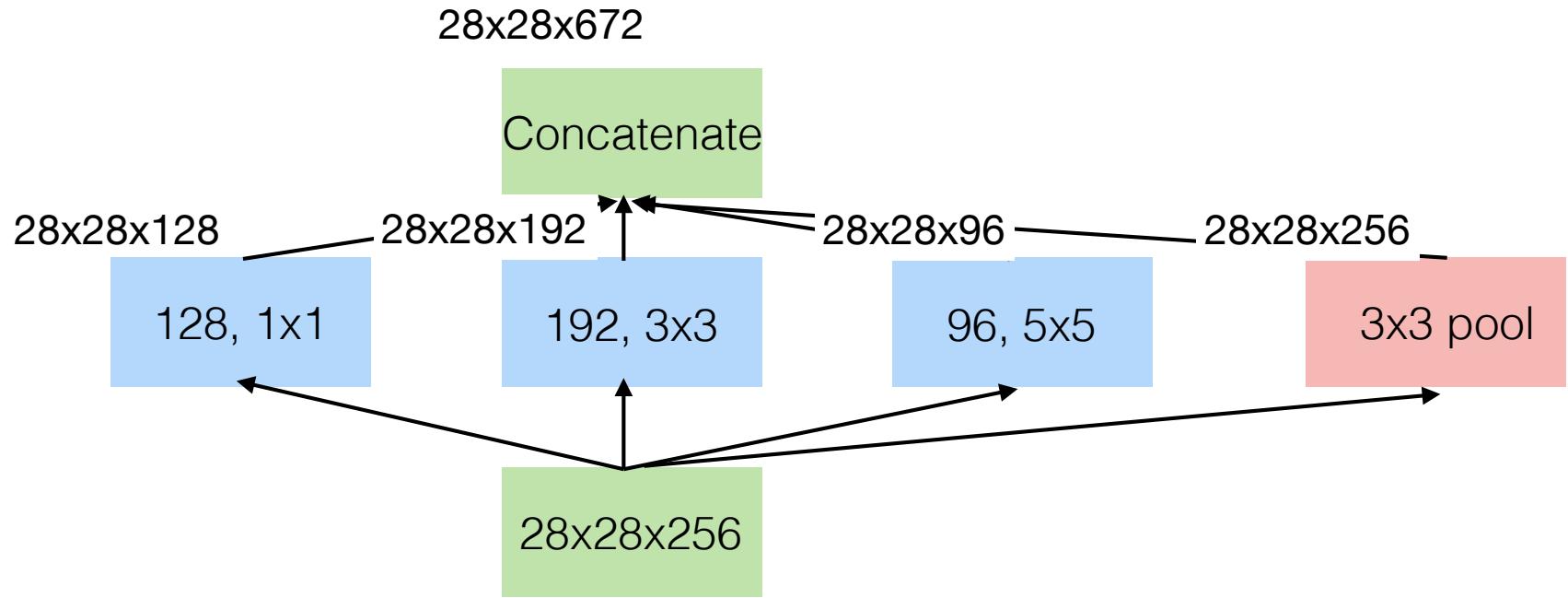
GoogLeNet



With this architecture, can the concatenated filter length ever be smaller than the input feature length?



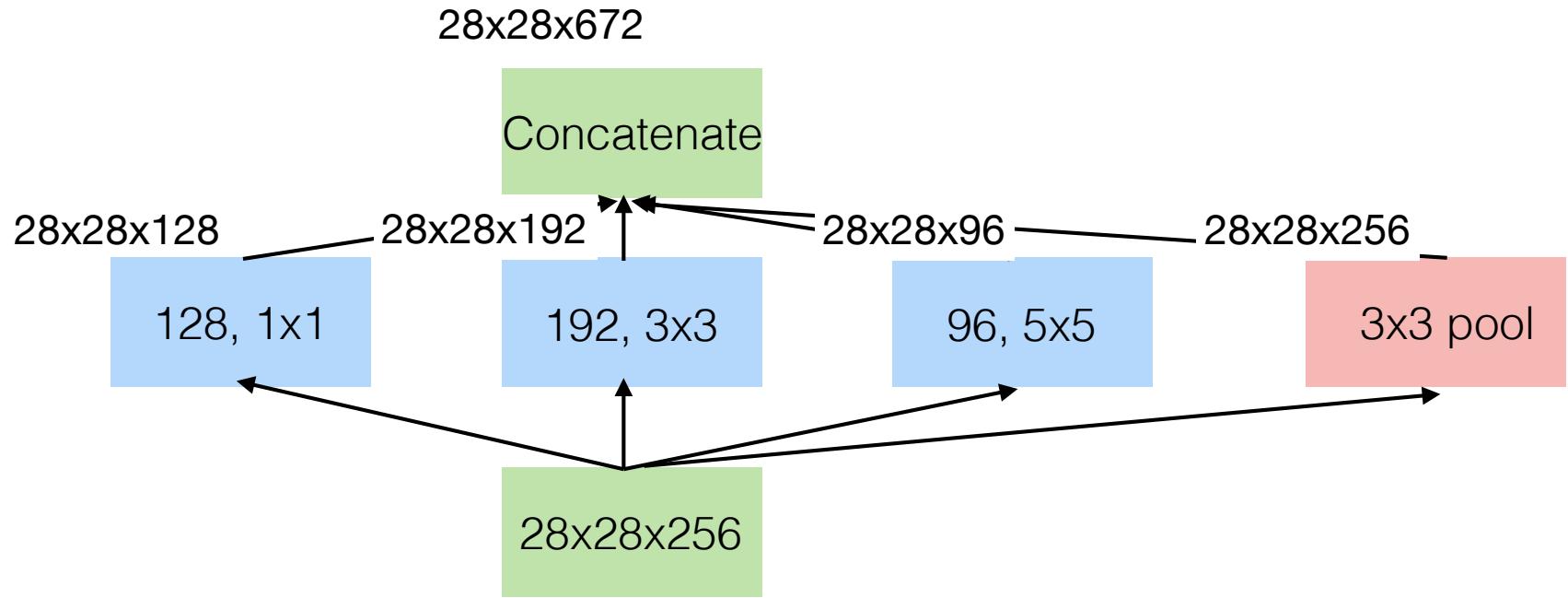
GoogLeNet



How many operations are there in this layer?



GoogLeNet



How many operations are there in this layer?

$$1 \times 1 \times 128 \text{ conv: } 28 \times 28 \times 256 \times 1 \times 1 \times 128 = 25,690,112$$

$$3 \times 3 \times 192 \text{ conv: } 28 \times 28 \times 256 \times 3 \times 3 \times 192 = 356,816,512$$

$$5 \times 5 \times 96 \text{ conv: } 28 \times 28 \times 256 \times 5 \times 5 \times 96 = 481,689,600$$

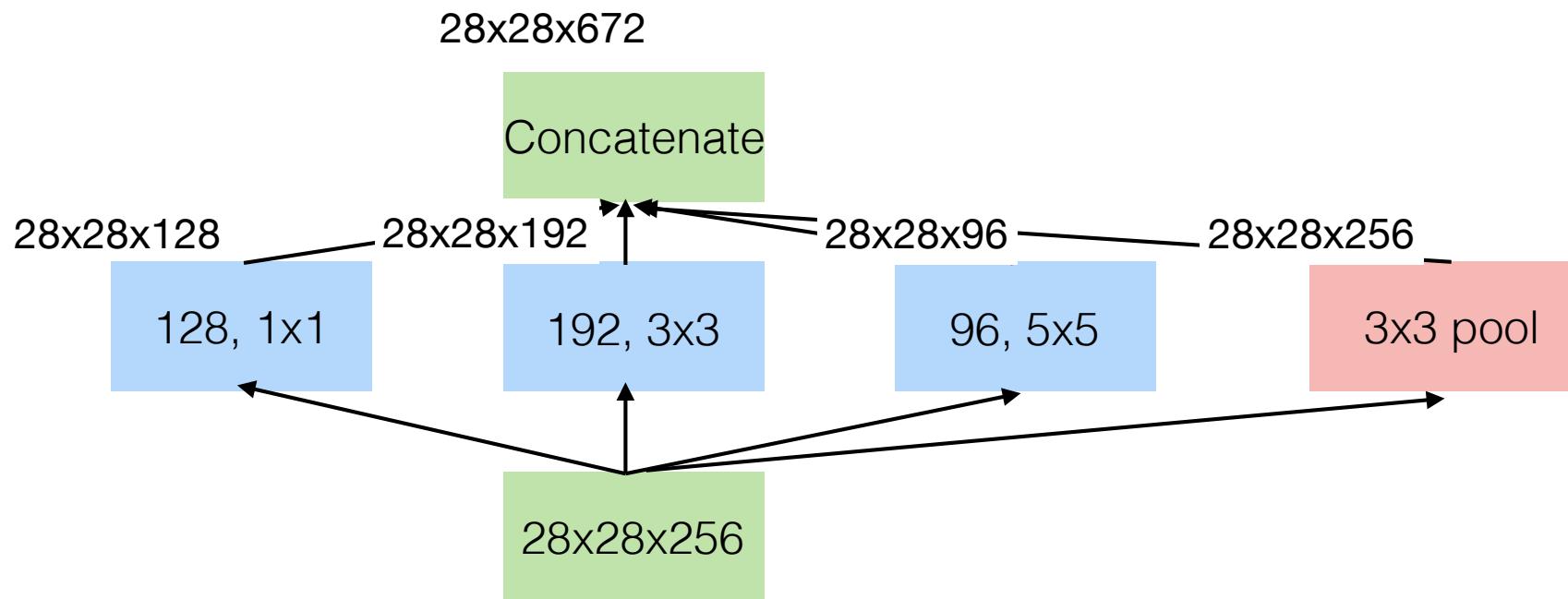
SUM = 864,196,224



GoogLeNet

To address this, in GoogLeNet, $1 \times 1 \times F$ 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?

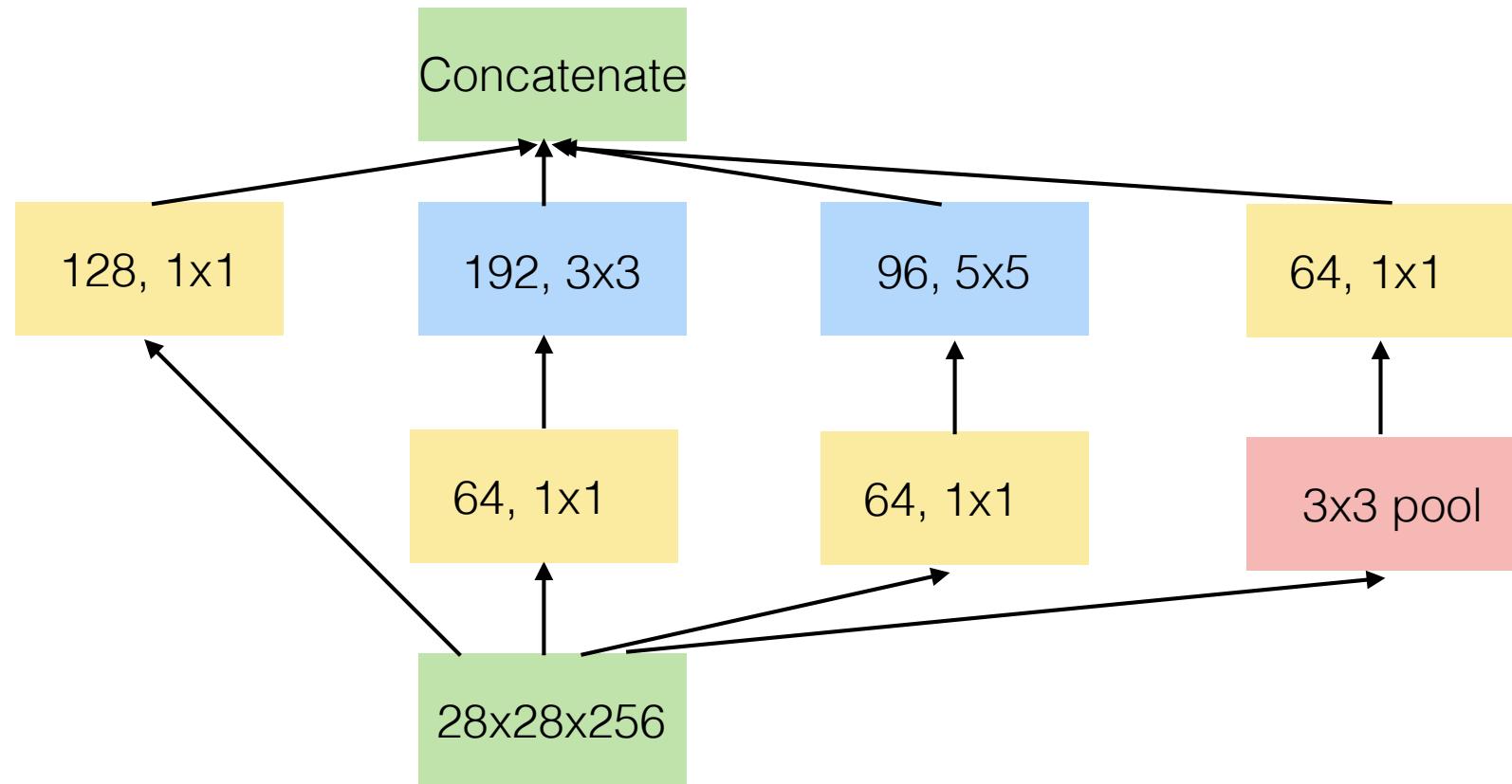




GoogLeNet

To address this, in GoogLeNet, $1 \times 1 \times F$ 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?

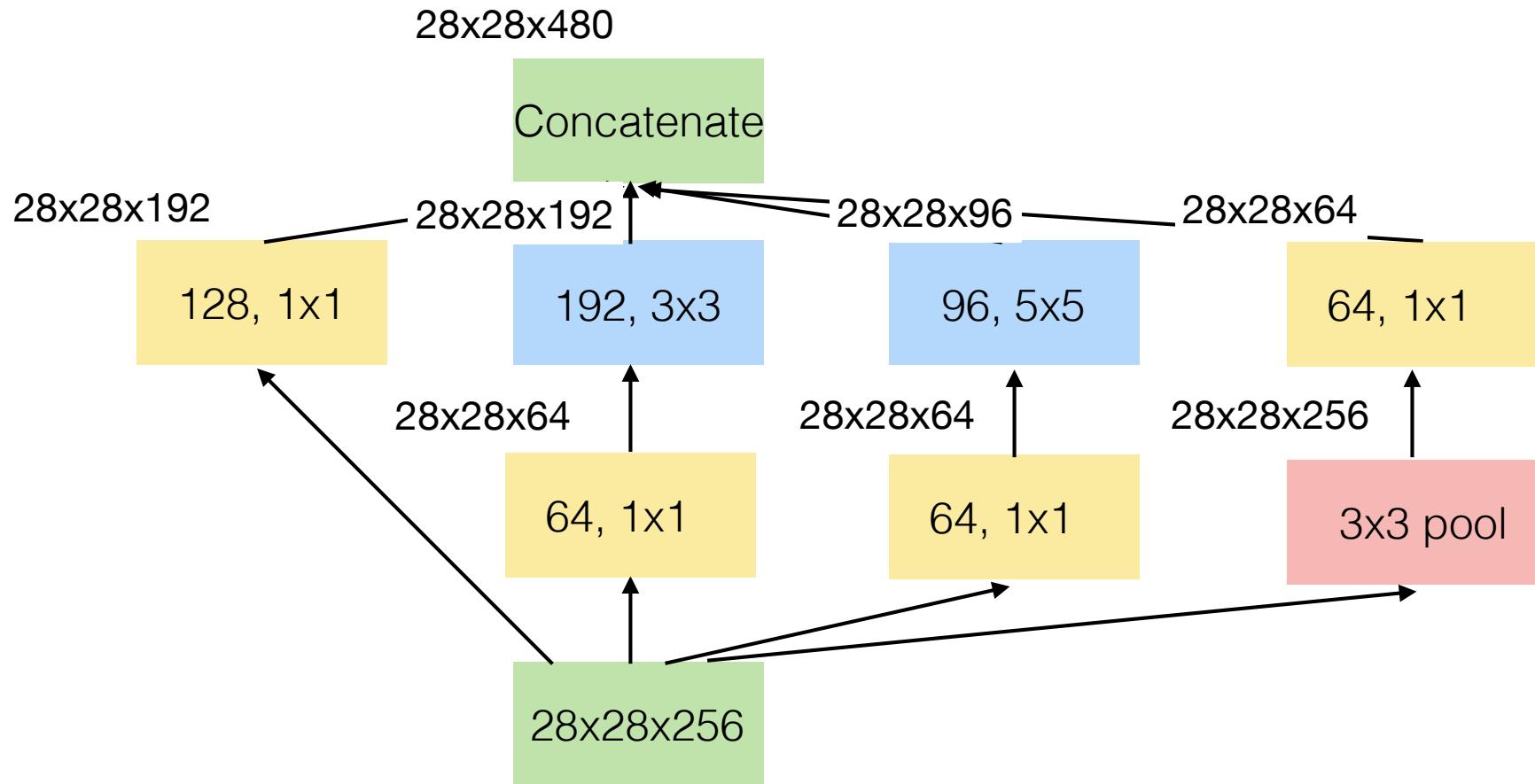




GoogLeNet

To address this, in GoogLeNet, $1 \times 1 \times F$ 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: $28 \times 28 \times 256 \times 1 \times 1 \times 128 = 25,690,112$

1x1x64 conv: $28 \times 28 \times 256 \times 1 \times 1 \times 64 = 12,845,056$

1x1x64 conv: $28 \times 28 \times 256 \times 1 \times 1 \times 64 = 12,845,056$

1x1x64 conv: $28 \times 28 \times 256 \times 1 \times 1 \times 64 = 12,845,056$

3x3x192 conv: $28 \times 28 \times 64 \times 3 \times 3 \times 192 = 86,704,128$

5x5x96 conv: $28 \times 28 \times 64 \times 5 \times 5 \times 96 = 120,422,400$

SUM: 271,351,808

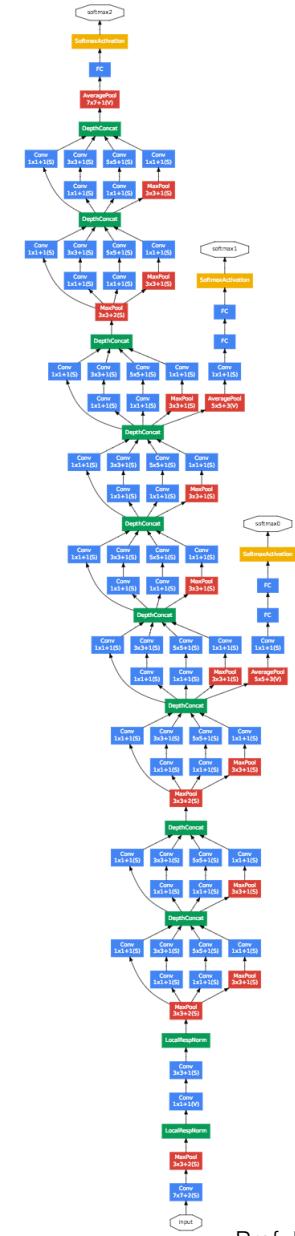
These 1x1 convolutions reduce the amount of computation by almost 4x in this example.



GoogLeNet

GoogLeNet architecture.

type	patch size/ stride	output size	depth	#1x1	#3x3 reduce	#3x3	#5x5 reduce	#5x5	pool proj	params	ops
convolution	7x7/2	112x112x64	1							2.7K	34M
max pool	3x3/2	56x56x64	0								
convolution	3x3/1	56x56x192	2		64	192				112K	360M
max pool	3x3/2	28x28x192	0								
inception (3a)		28x28x256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28x28x480	2	128	128	192	32	96	64	380K	304M
max pool	3x3/2	14x14x480	0								
inception (4a)		14x14x512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14x14x512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14x14x512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14x14x528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14x14x832	2	256	160	320	32	128	128	840K	170M
max pool	3x3/2	7x7x832	0								
inception (5a)		7x7x832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7x7x1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7x7/1	1x1x1024	0								
dropout (40%)		1x1x1024	0								
linear		1x1x1000	1							1000K	1M
softmax		1x1x1000	0								





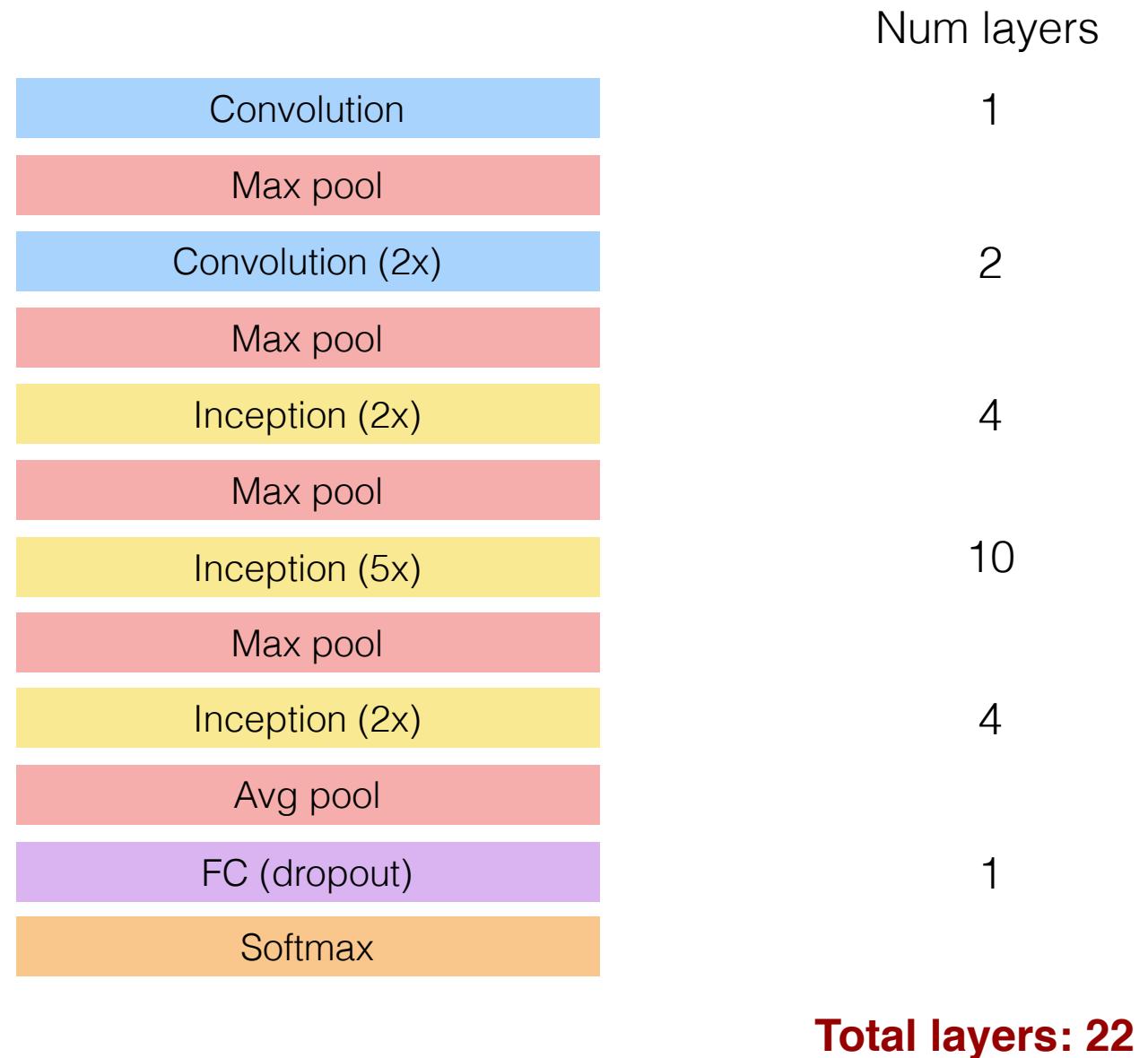
GoogLeNet

	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



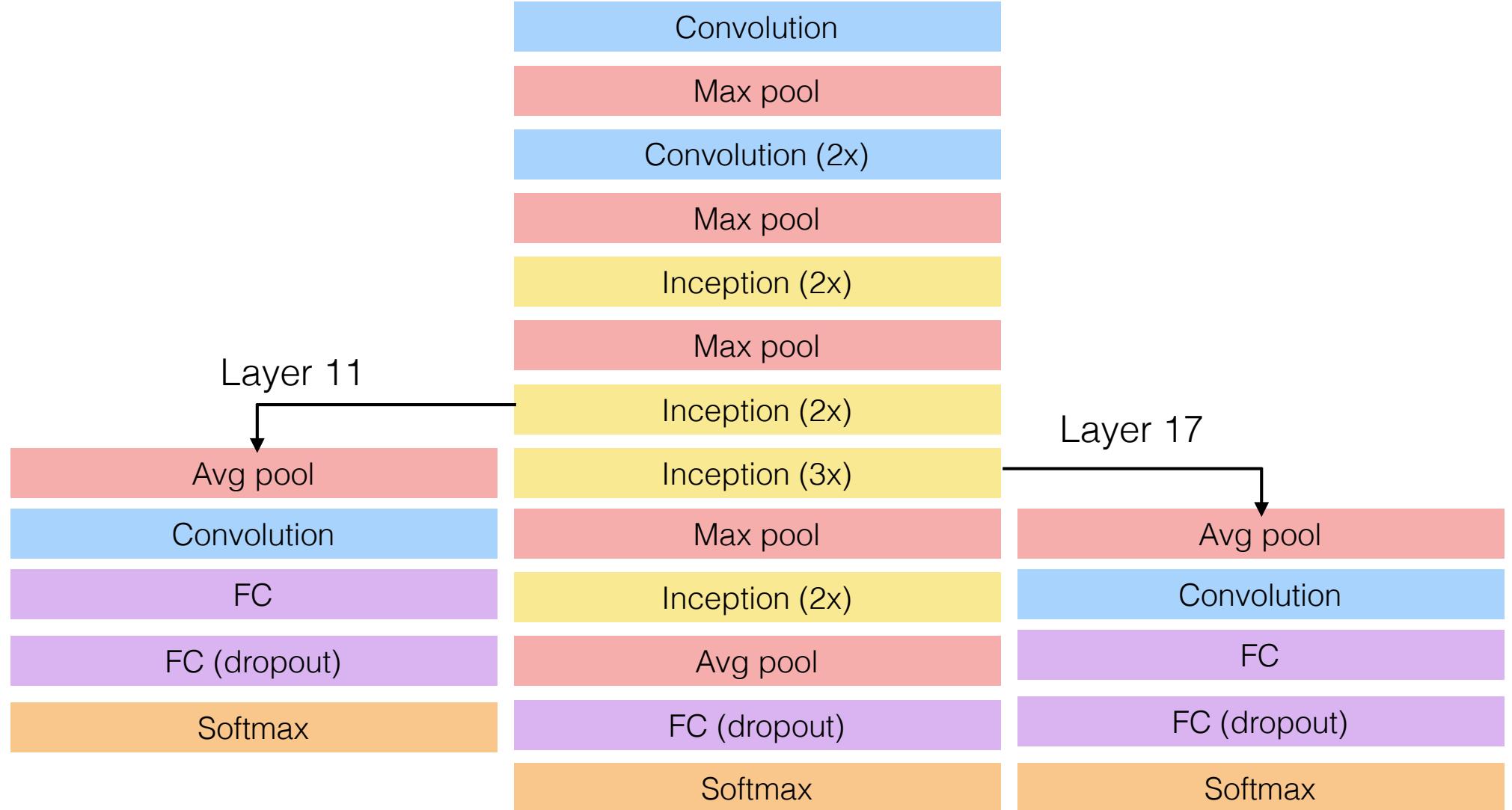
GoogLeNet



What might be a concern about this architecture?

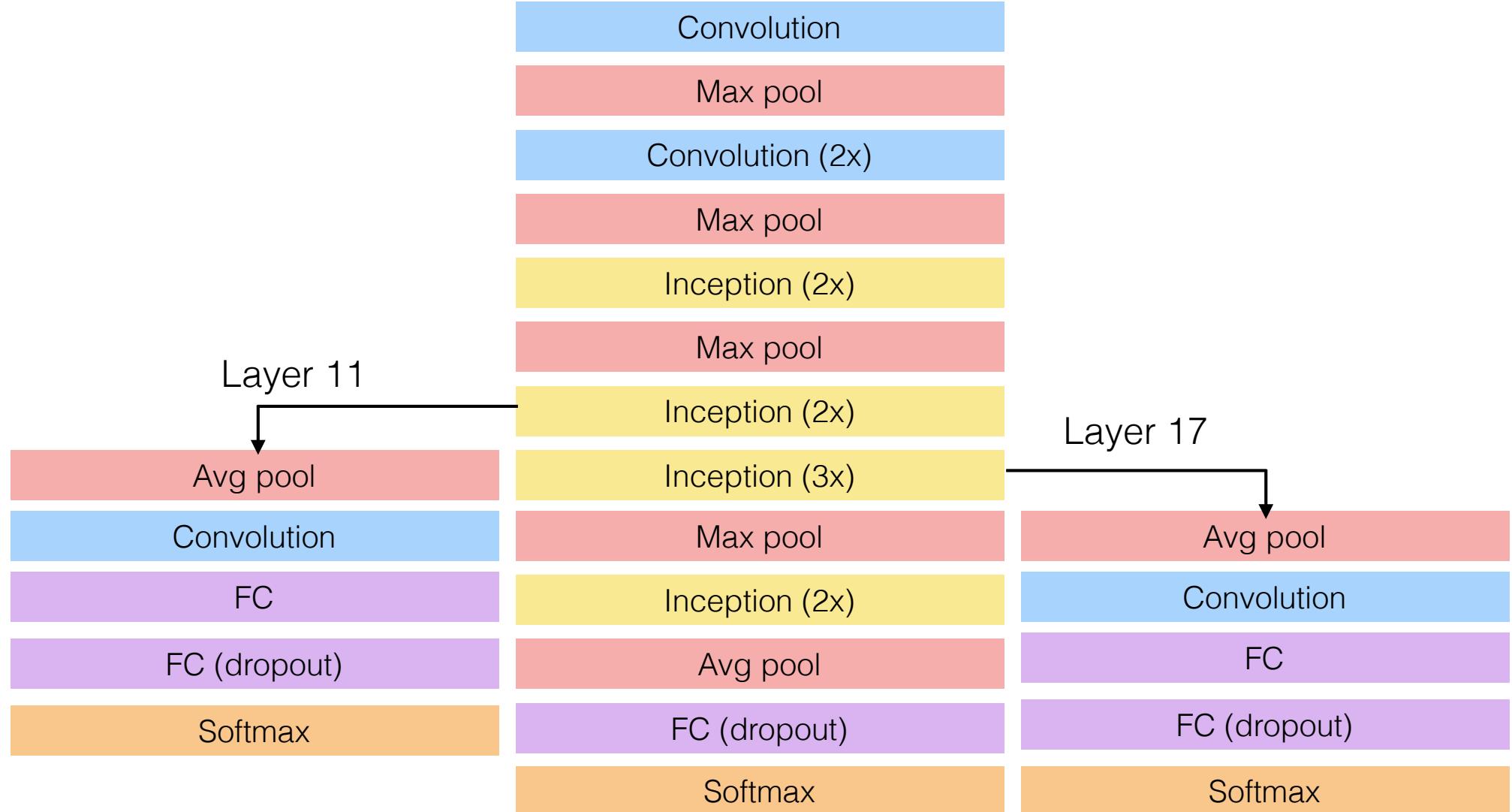


GoogLeNet





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.



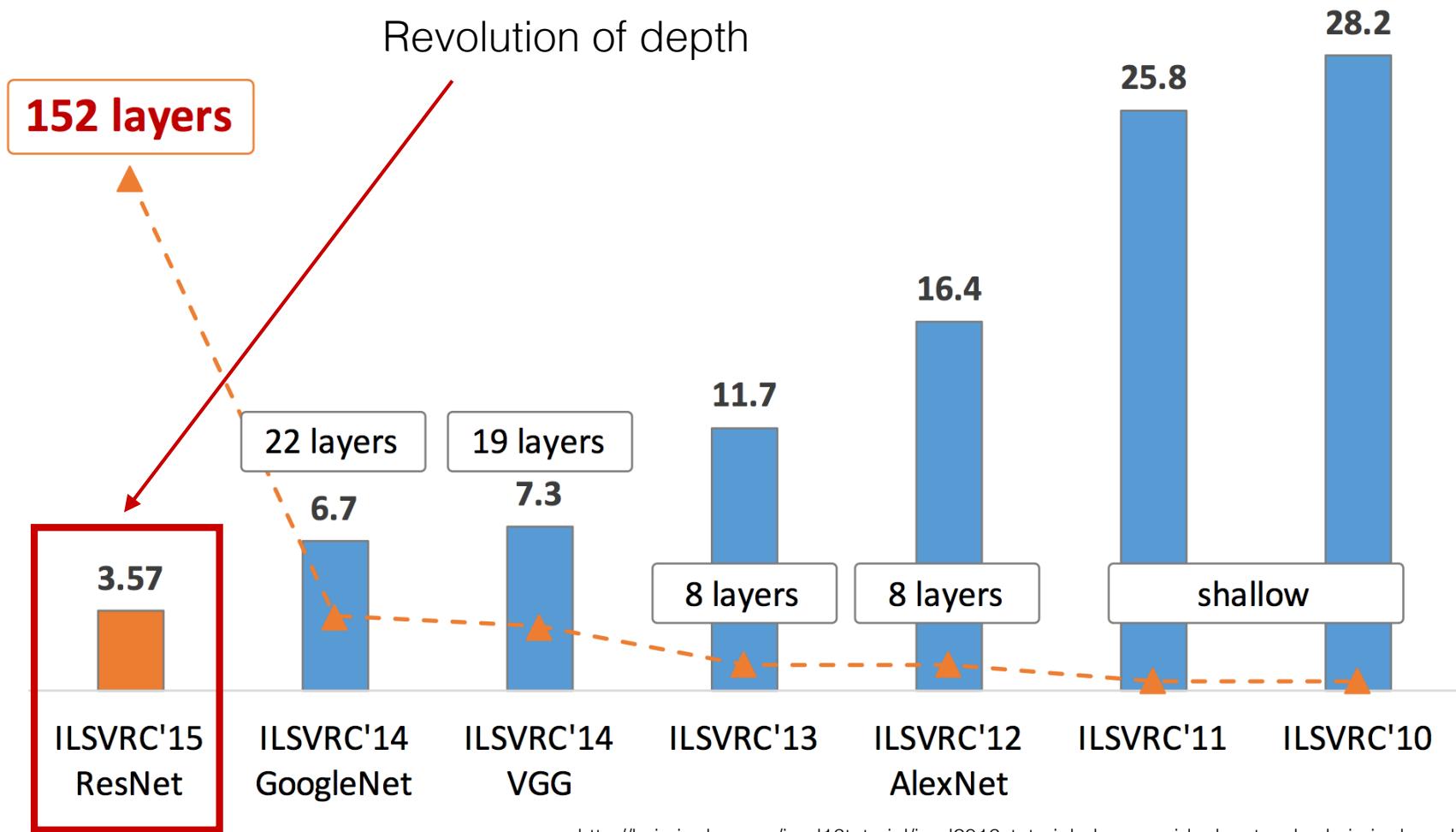
GoogLeNet

Other details:

- SGD with 0.9 momentum.
- Decrease learning rate by 4% every 8 epochs.
- Polyak averaging with exponential decay.
- 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

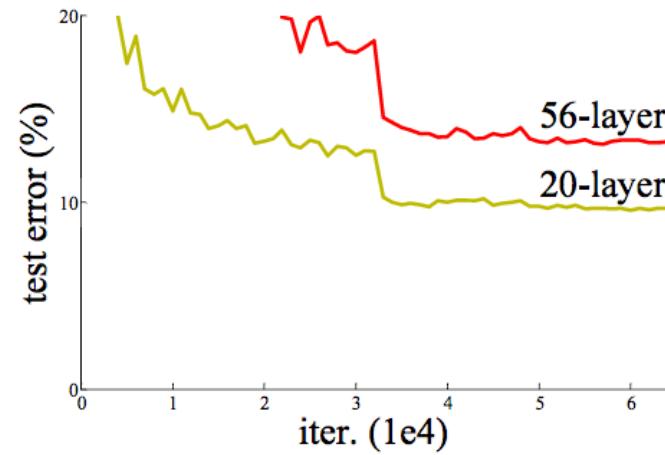
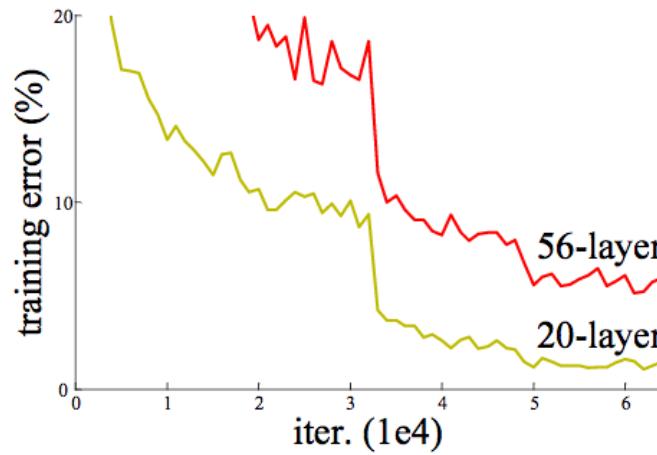




ResNet

Idea so far:

- AlexNet and ZFNet were 8 layers.
- VGG Net was 16-19 layers.
- GoogLeNet was 22 layers?
- Why not just keep adding layers?



He et al., 2016



ResNet

This result is non-intuitive. (Why?)

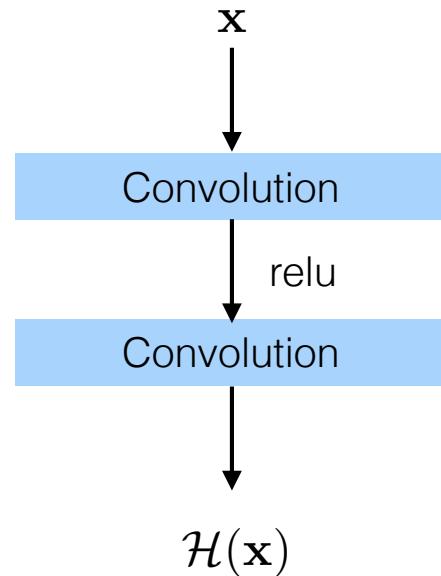
What are potential problems for why these networks do not converge?



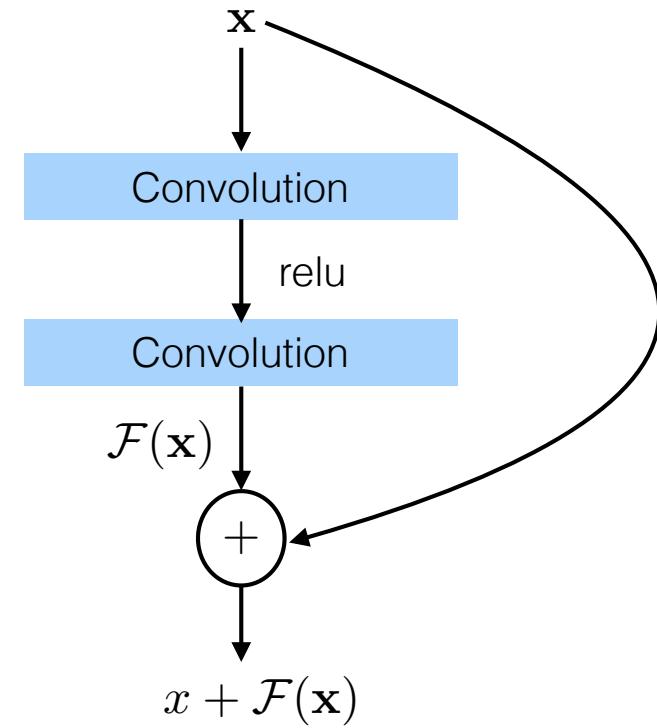
ResNet

The main idea of the ResNet architecture is to facilitate the network training by causing each layer to learn a residual to add to the input.

Normal architecture



ResNet architecture

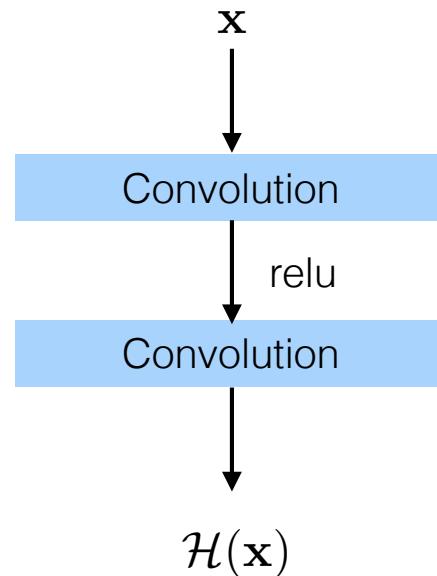




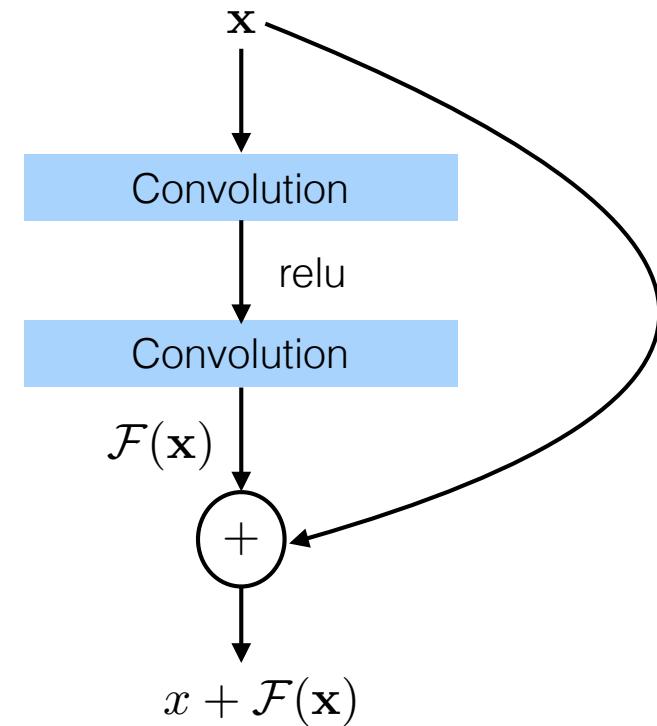
ResNet

The main idea of the ResNet architecture is to facilitate the network training by causing each layer to learn a residual to add to the input.

Normal architecture



ResNet architecture



Residual layers are used to fit $\mathcal{F}(x) \approx \mathcal{H}(x) - x$

To make dimensions work out, sometimes a linear mapping is used: $\mathbf{W}_s x$



ResNet

“We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.”

“To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.”

(He et al., 2016)



ResNet

Network architecture:

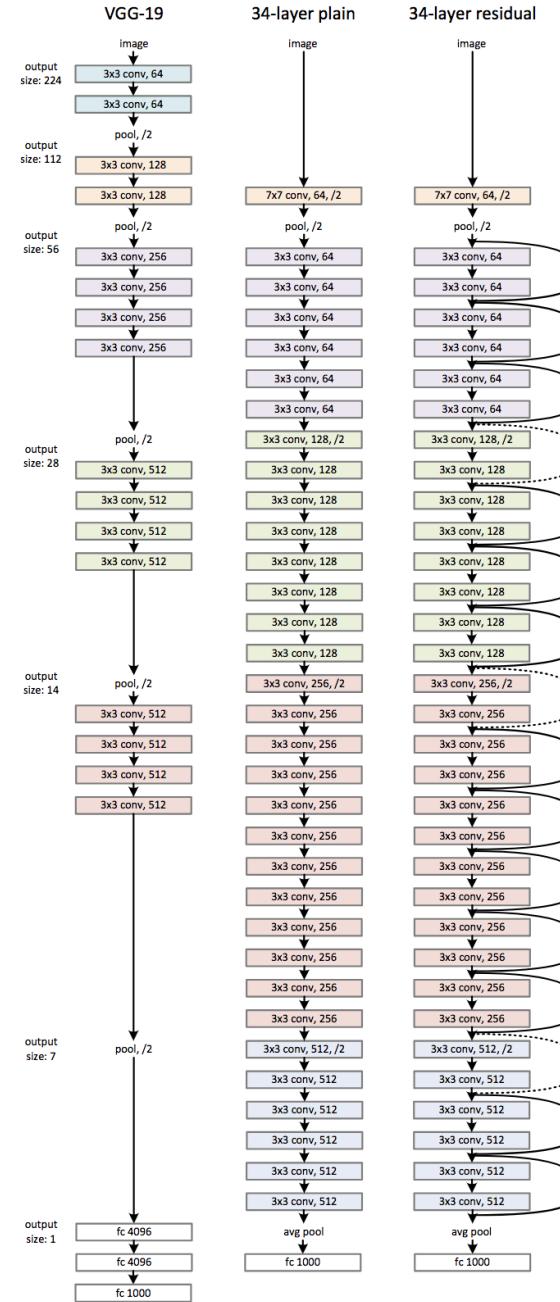
- They follow the design rules of VGG Net.
 - All conv layers are 3x3 filters with the same number of filters.
 - If the feature map size is halved, the number of filters is doubled so that the computational complexity in each layer is the same.
 - The output ends with average pooling and then a FC 1000 layer to a Softmax.
- Conv layers are residual network layers.

Other notes:

- They performed data augmentation (image scaling, different crops)
- SGD with momentum 0.9, with mini batch size 256.
- L2 regularization with weight 0.0001
- Learning rate starts off at 0.1 and is decreased by an order of magnitude when the error plateaus.
- No dropout.
- Train for 600,000 iterations.



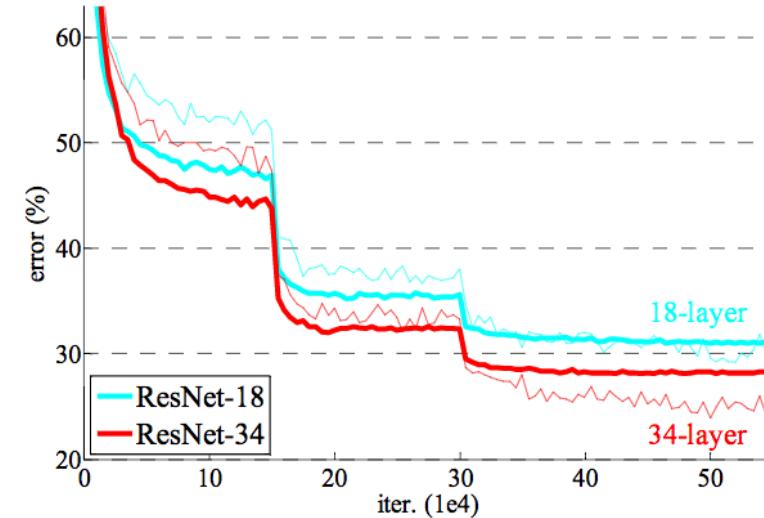
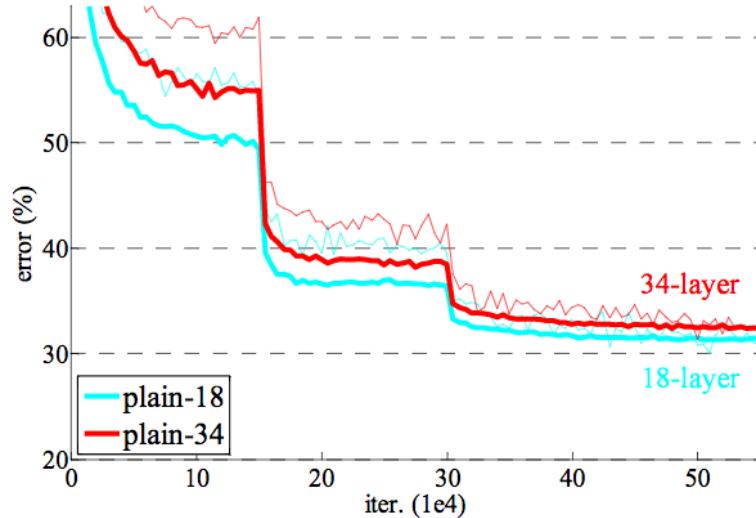
ResNet



He et al., 2016

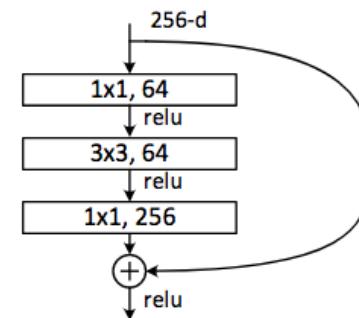
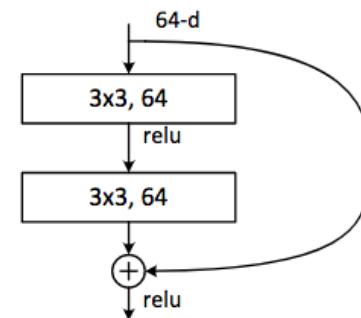


ResNet



He et al., 2016

For deeper networks, use an idea similar to inception:



He et al., 2016

Prof J.C. Kao, UCLA ECE



ResNet

CIFAR-10 performance:

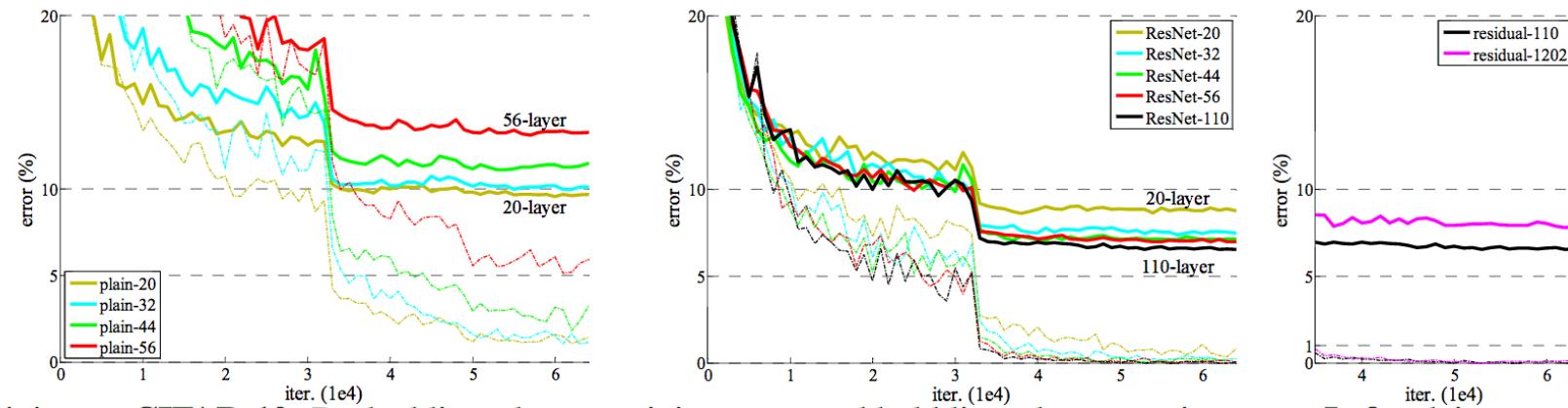
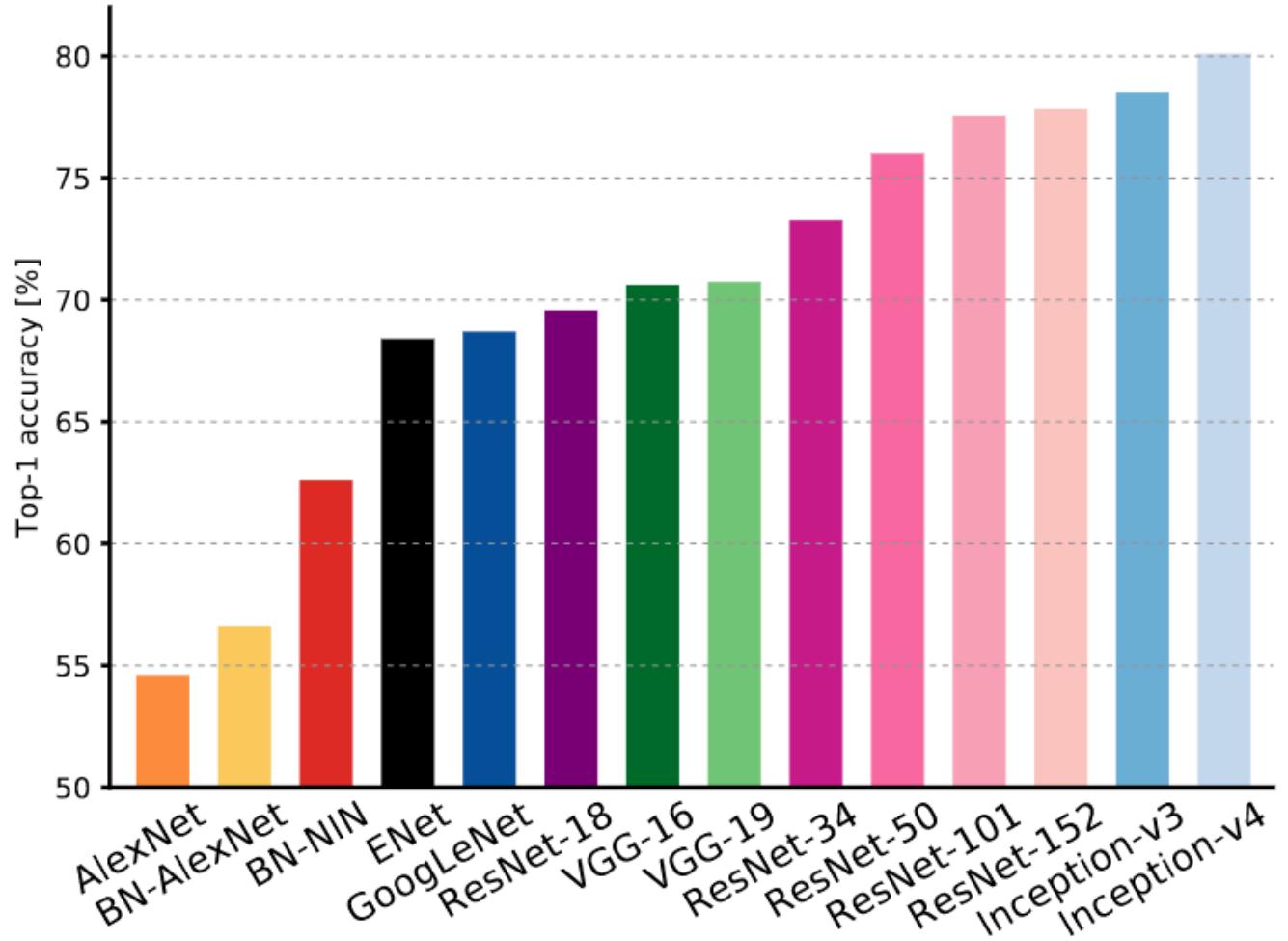


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left:** plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle:** ResNets. **Right:** ResNets with 110 and 1202 layers.

He et al., 2016



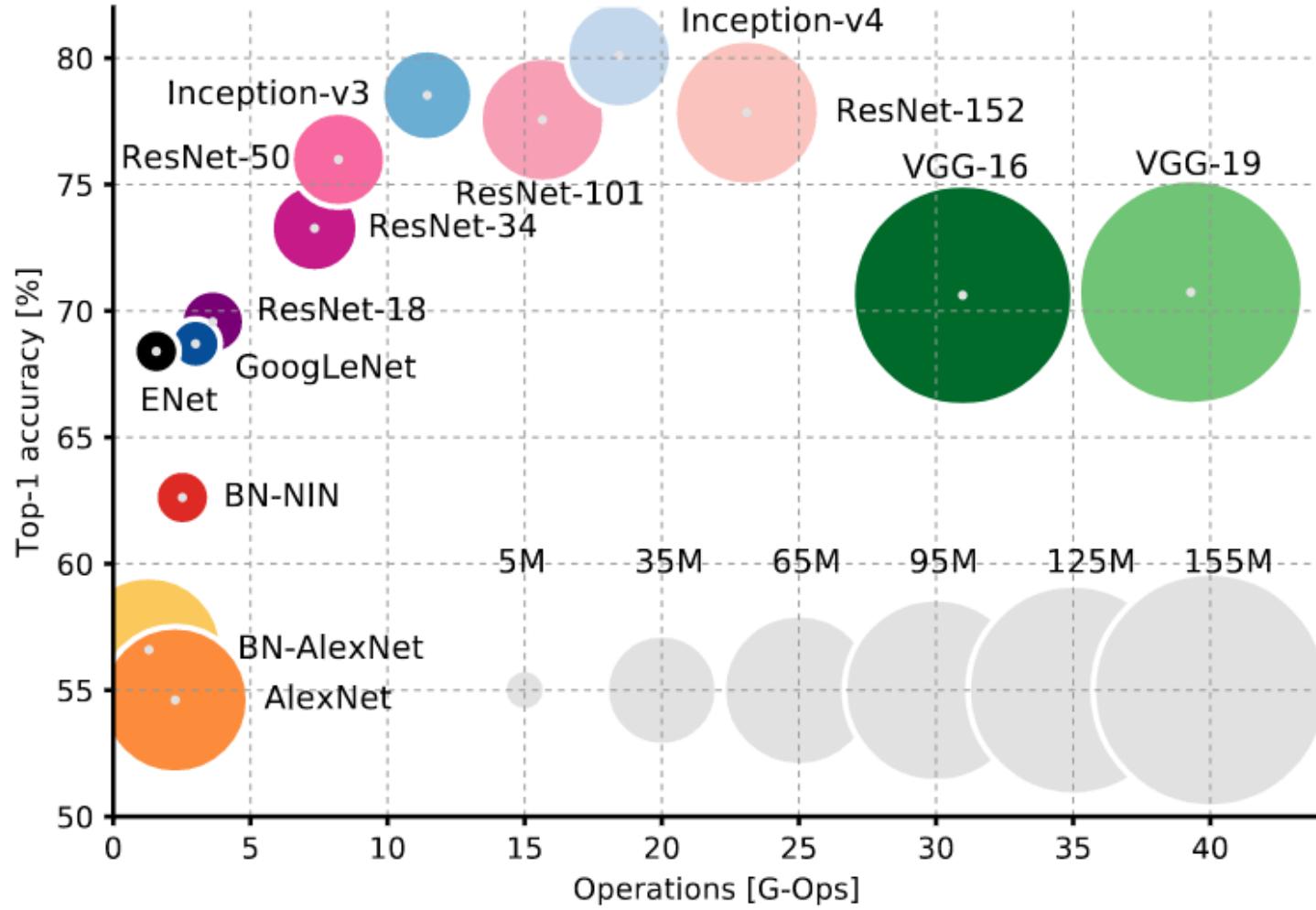
An overall view of architectures



Canziani et al., 2017



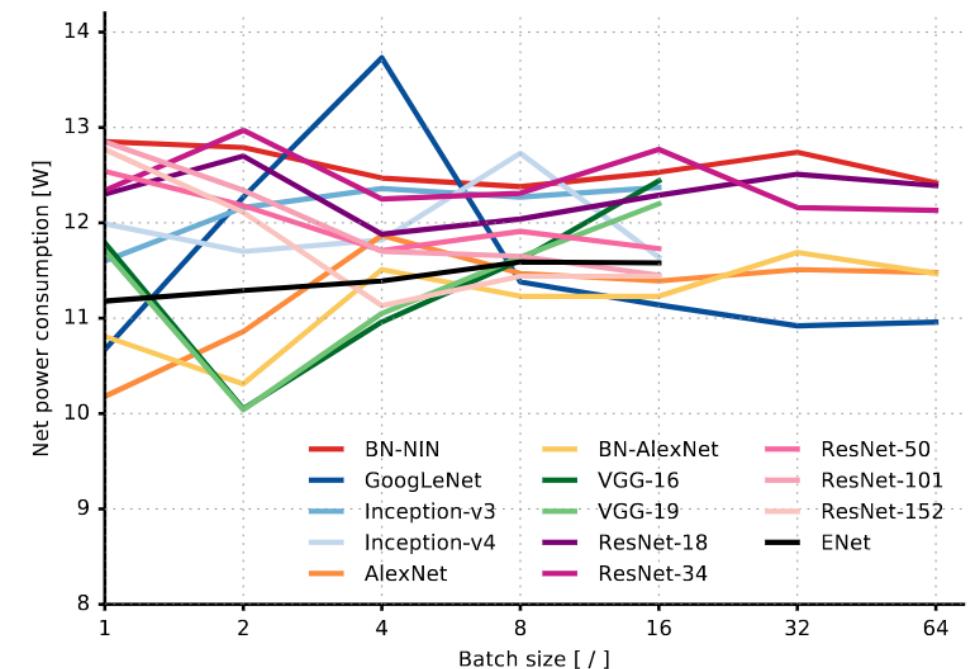
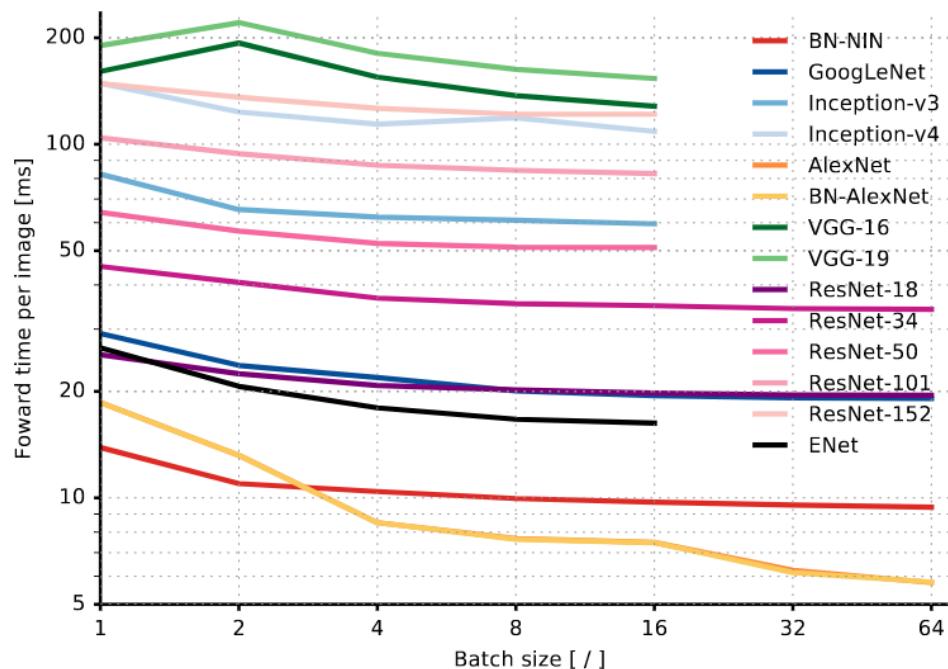
An overall view of architectures



Canziani et al., 2017



An overall view of architectures



Canziani et al., 2017



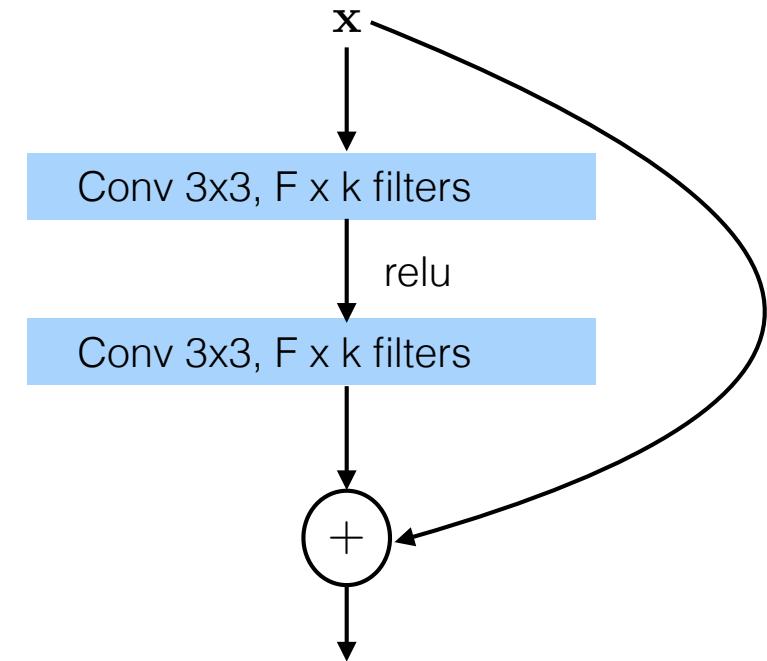
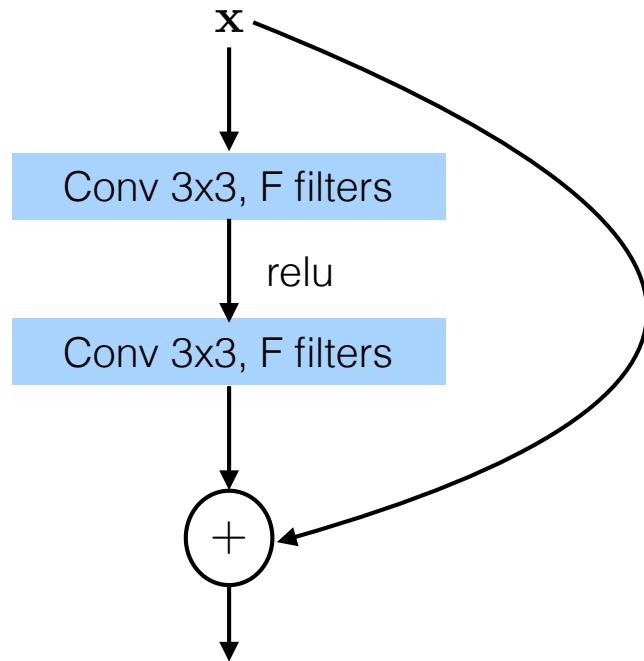
ResNets were 2015. What's been going on since then?

ImageNet errors continue to get lower and lower. Here are some architectures that have been developed since the ResNet. A lot of them are modifications on top of ResNet.



Wide ResNets

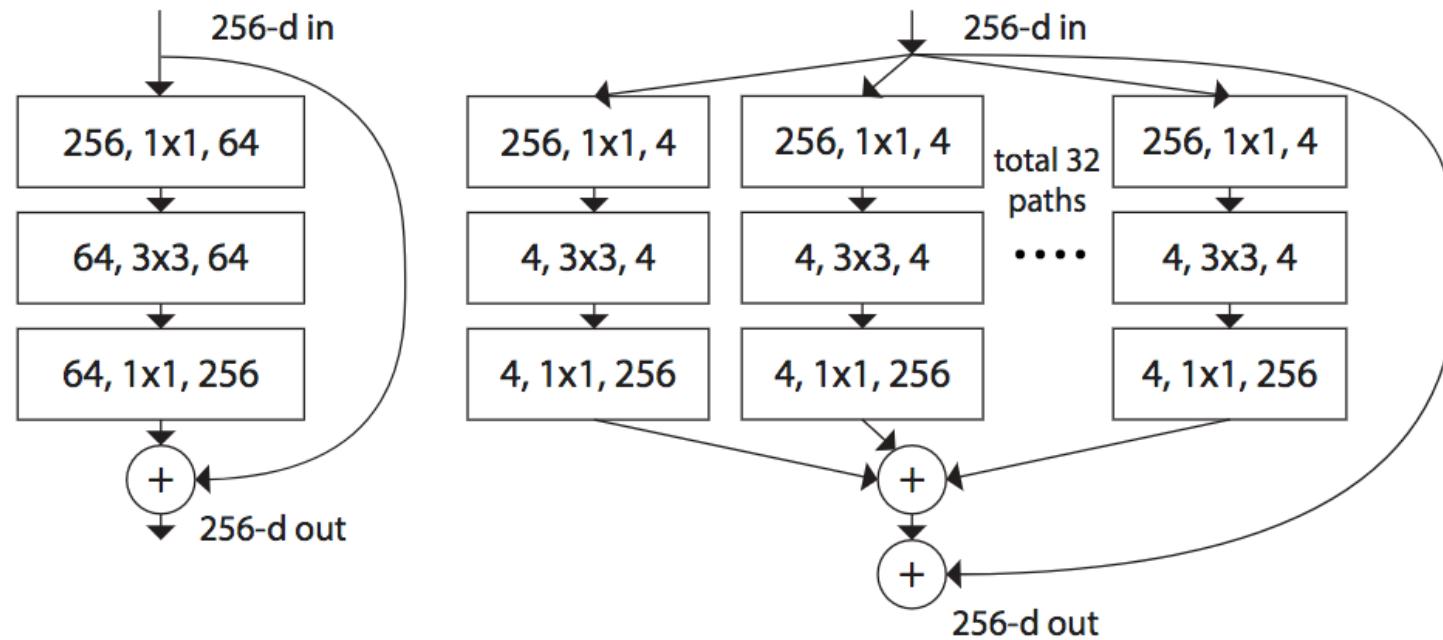
ResNets and Wide ResNets





ResNeXt (ImageNet 2016 2nd place)

ResNeXt.



Xie et al., 2017



ResNeXt (ImageNet 2016 2nd place)

ResNeXt.

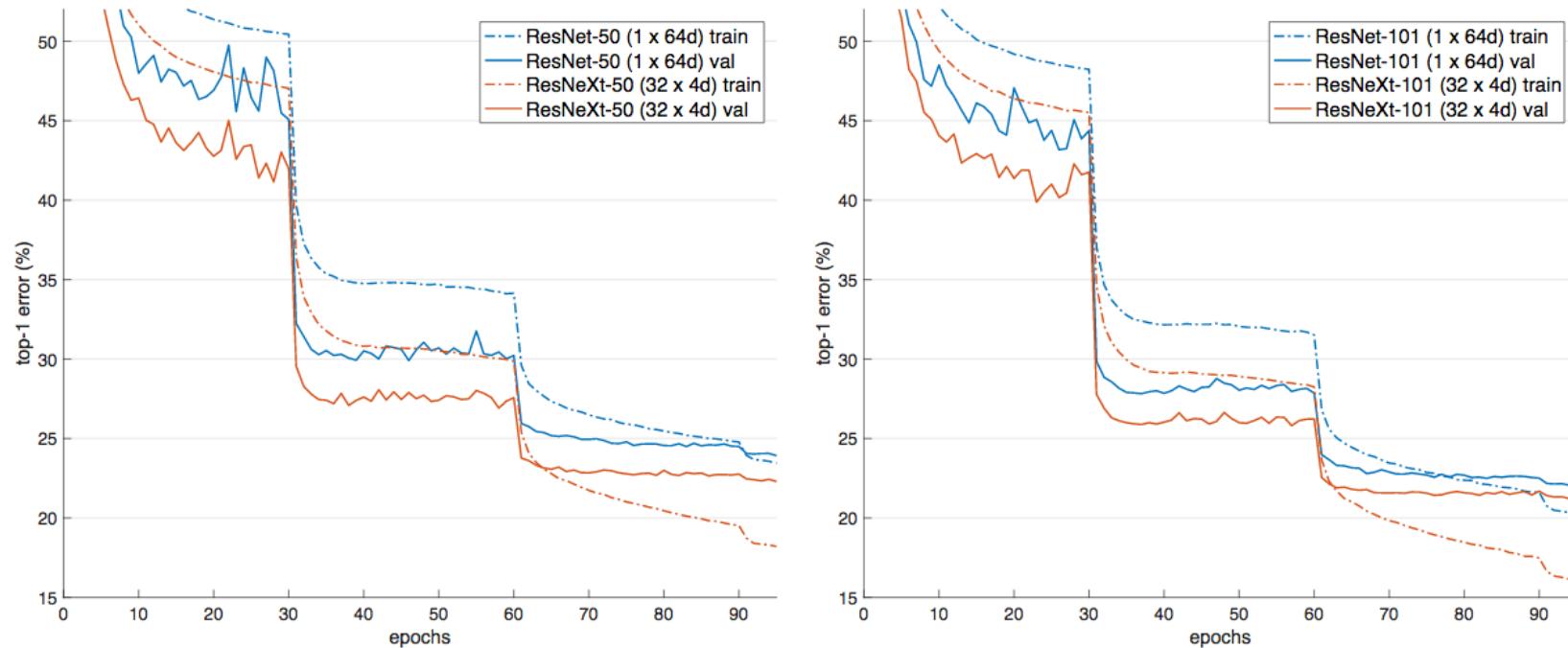


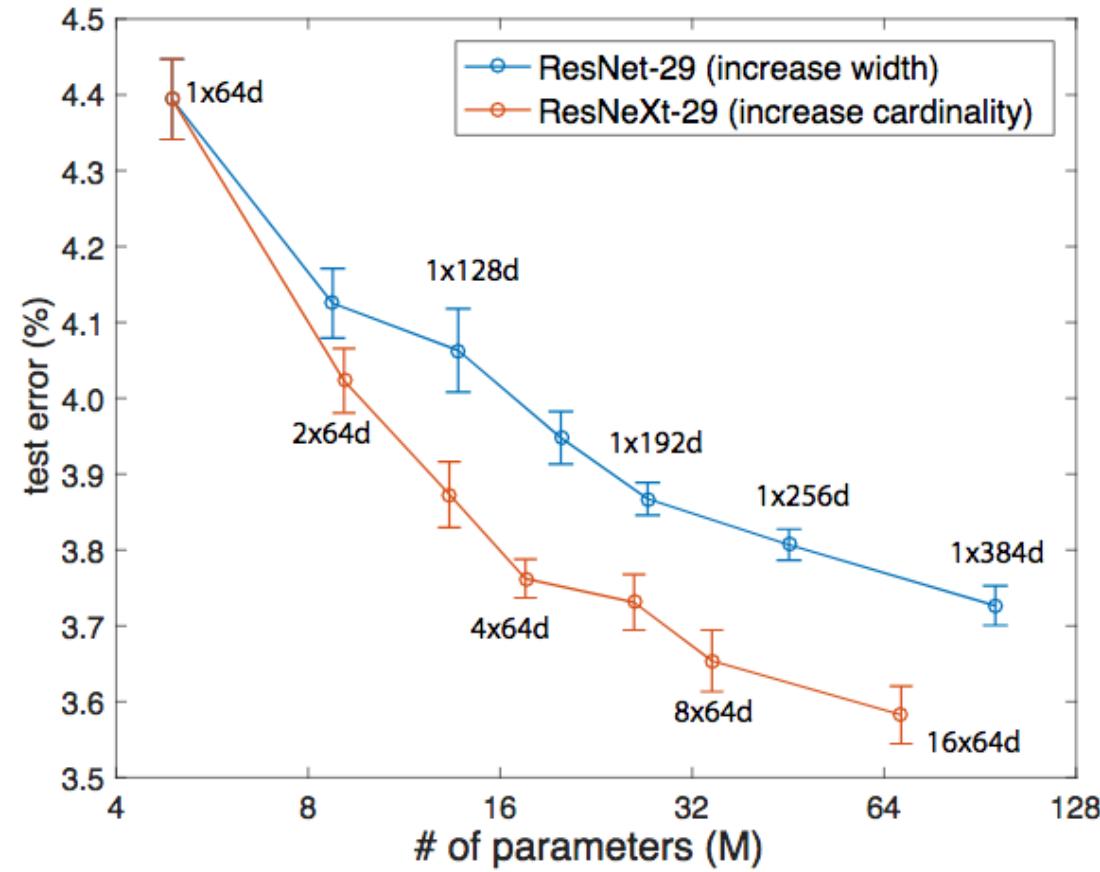
Figure 5. Training curves on ImageNet-1K. (**Left**): ResNet/ResNeXt-50 with preserved complexity (~4.1 billion FLOPs, ~25 million parameters); (**Right**): ResNet/ResNeXt-101 with preserved complexity (~7.8 billion FLOPs, ~44 million parameters).

Xie et al., 2017



ResNeXt (ImageNet 2016 2nd place)

Better than simply increasing width.



Xie et al., 2017



Stochastic depth

Randomly dropout modules of the ResNet.

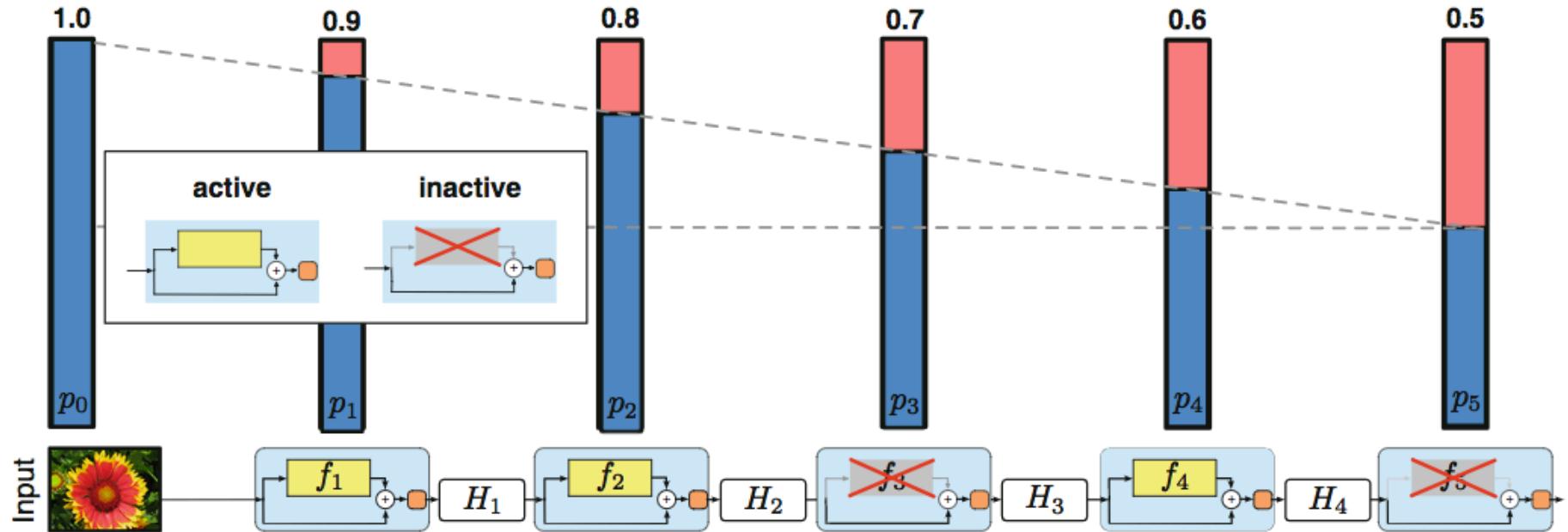


Fig. 2. The linear decay of p_ℓ illustrated on a ResNet with stochastic depth for $p_0=1$ and $p_L=0.5$. Conceptually, we treat the input to the first ResBlock as H_0 , which is always active.

Huang et al., 2016



Stochastic depth

Randomly dropout modules of the ResNet.

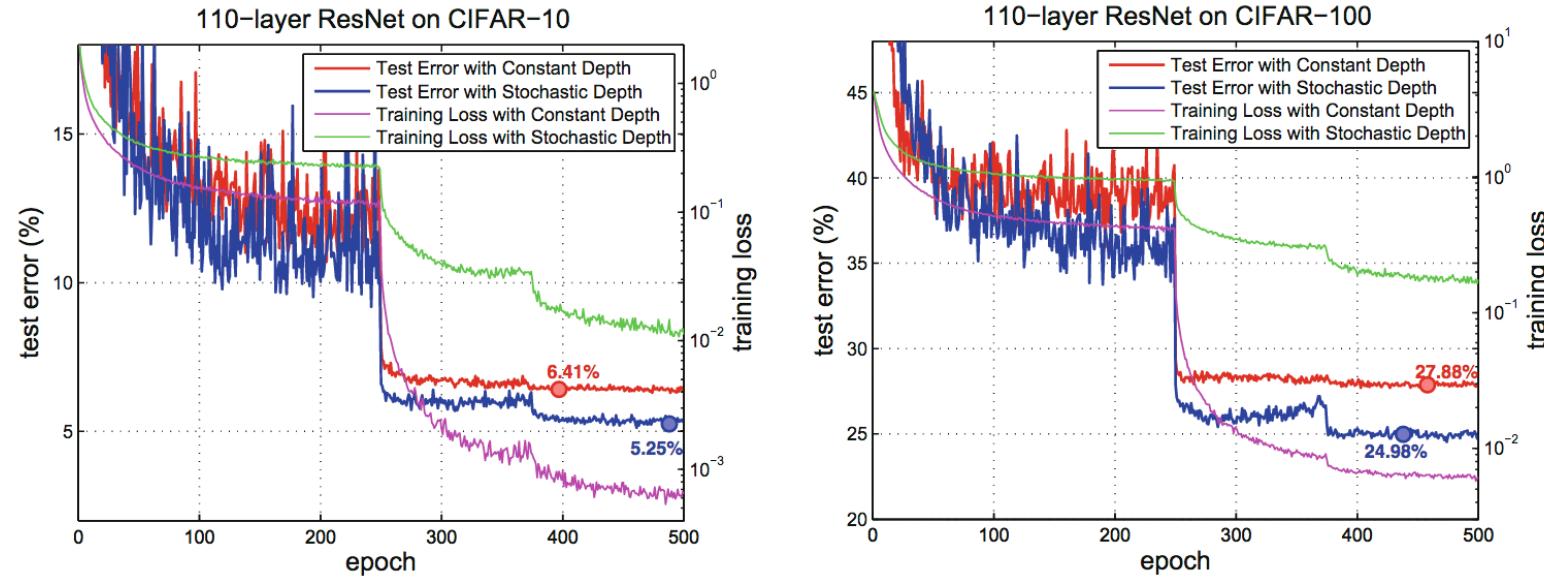


Fig. 3. Test error on CIFAR-10 (*left*) and CIFAR-100 (*right*) during training, with data augmentation, corresponding to results in the first two columns of Table 1.

Huang et al., 2016



Fractal Nets

*“...thereby demonstrating that **residual representations may not be fundamental to the success of extremely deep convolutional neural networks. Rather, the key may be the ability to transition, during training, from effectively shallow to deep.**”*

Larsson et al., 2017

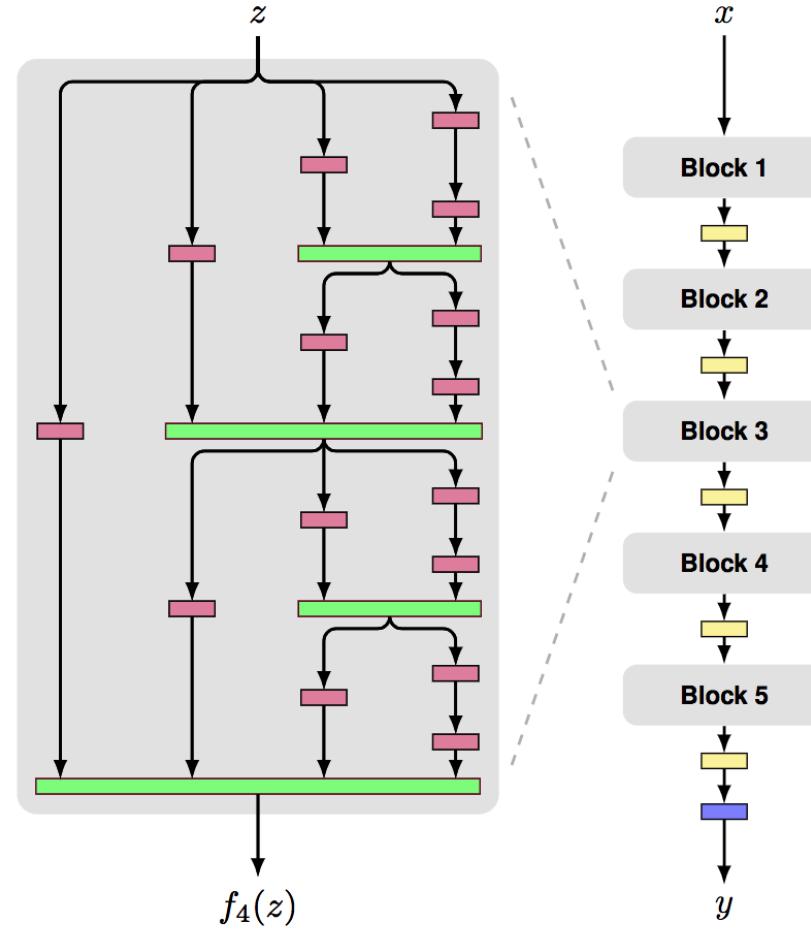
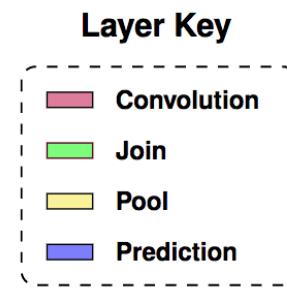
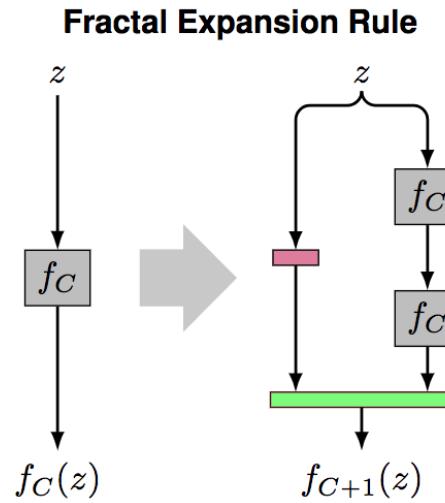
Regarding ResNet:

*First, the objective changes to learning residual outputs, rather than unreferenced absolute mappings. Second, these networks exhibit a type of deep supervision (Lee et al., 2014), as near-identity layers effectively reduce distance to the loss. **He et al. (2016a) speculate that the former, the residual formulation itself, is crucial.***

Larsson et al., 2017



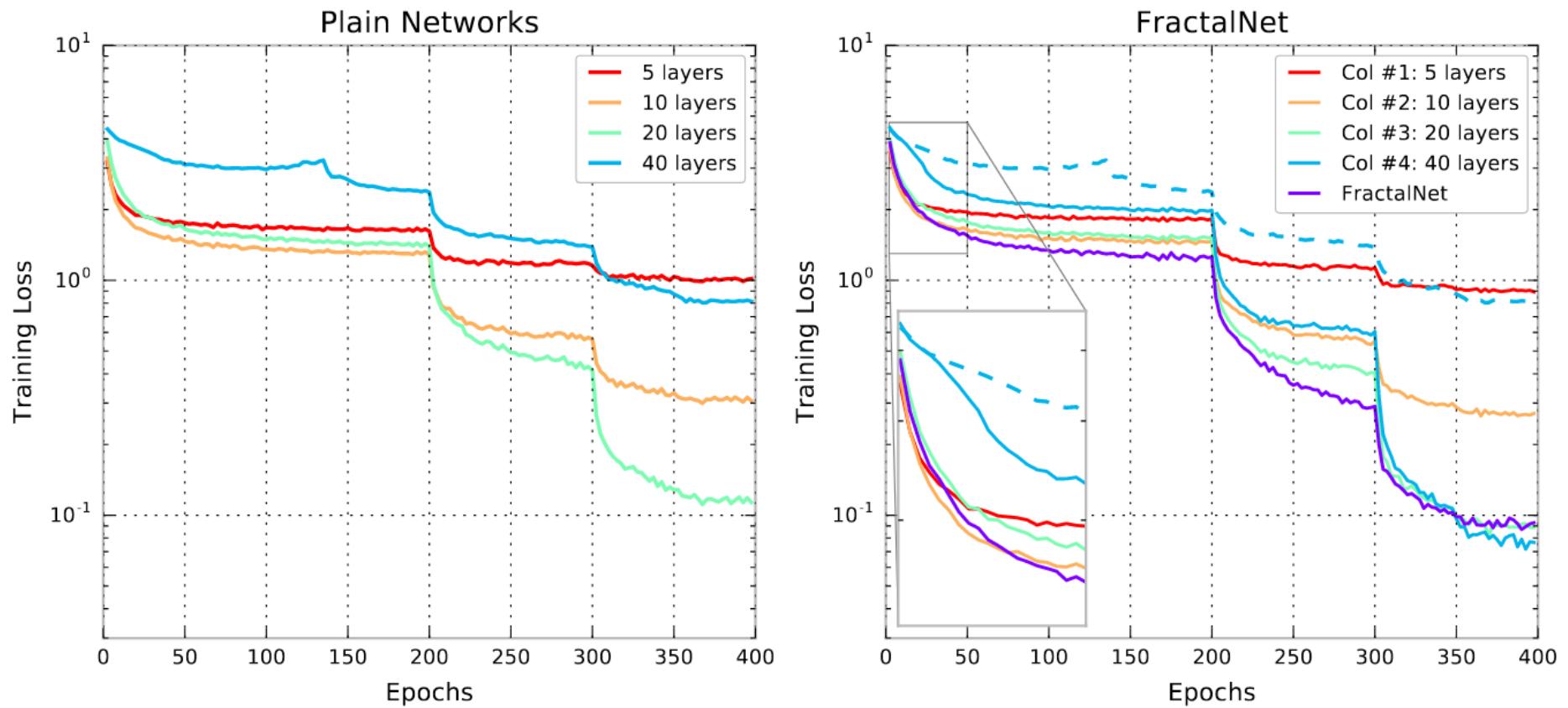
Fractal Nets



Larsson et al., 2017



Fractal Nets



Larsson et al., 2017



Take home points

For convolutional neural networks:

- Architecture can play a key role in the performance of the network.
- There is evidence that deeper and wider (i.e., more feature maps) result in better performance.
- Going deeper helps to a point; beyond that, new architectures have to be considered (like the ResNet).
- It appears that what is key about these different architectures is that they reduce the *effective depth* of the network, i.e., they shorten the longest path from the output loss to the network input. This helps to avoid the fundamental problem of deep learning.