Convolutional Neural Network

Sudeshna Sarkar

22/2/17, 23/2/17, 24/2/17

History

- In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.

Convolution

1D (continuous, discrete) :

Input

Kernel

$$f * g \ (x) = \int_{\alpha=-\infty}^{\infty} f(\alpha) g(x-\alpha) d\alpha$$

$$= \sum_{\alpha=0}^{N-1} f(\alpha) g(x-\alpha)$$

Output is sometimes called Feature map

2D (continuous, discrete) :

$$f * g \ (x,y) = \int_{\alpha=-\infty}^{\infty} \int_{\beta=-\infty}^{\infty} f(\alpha,\beta) g(x-\alpha, y-\beta) d\alpha d\beta$$

$$= \sum_{\alpha=0}^{N-1} \sum_{\beta=0}^{N-1} f(\alpha,\beta) g(x-\alpha, y-\beta)$$

Convolution Properties

- Commutative:

  $f*g = g*f$

- Associative:

  $(f*g)*h = f*(g*h)$

- Homogeneous:

  $f*(\lambda g)= \lambda f*g$

- Additive (Distributive):

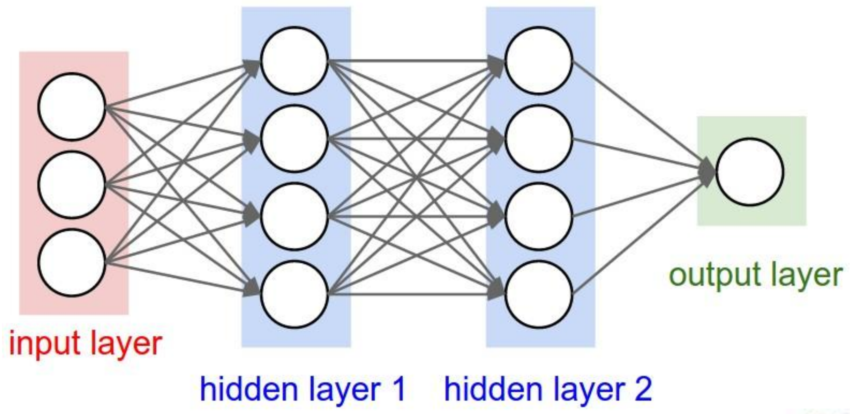
  $f*(g+h)= f*g+f*h$

- Shift-Invariant

  $f*g(x-x0,y-yo)= (f*g) (x-x0,y-yo)$

ConvNet

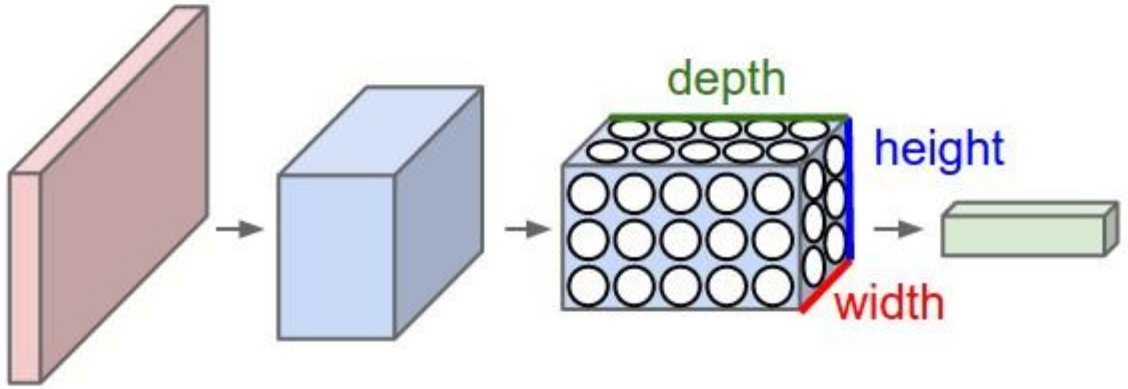
- ConvNet architectures for images:
  - fully-connected structure does not scale to large images
  - the explicit assumption that the inputs are images
  - allows us to encode certain properties into the architecture.
  - These then make the forward function more efficient to implement
  - Vastly reduce the amount of parameters in the network.
- 3D volumes: neurons arranged in 3 dimensions: width, height, depth.



Convnets

Layers used to build ConvNets:

- a stacked sequence of layers. 3 main types
- Convolutional Layer, Pooling Layer, and Fully-Connected Layer

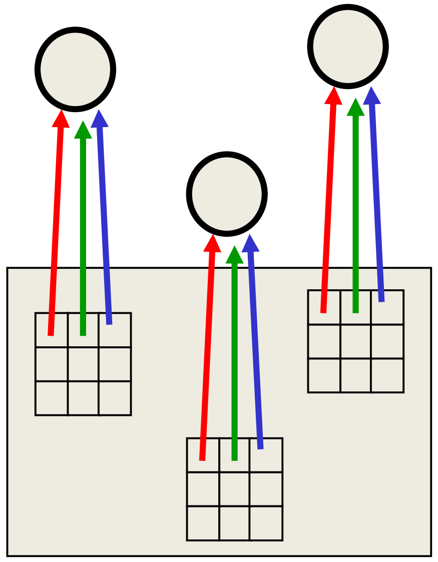


- every layer of a ConvNet transforms one volume of activations to another through a differentiable function.

The replicated feature approach

- Use many different copies of the same feature detector with different positions.
  - Could also replicate across scale and orientation (tricky and expensive)
  - Replication greatly reduces the number of free parameters to be learned.
- Use several different feature types, each with its own map of replicated detectors.
  - Allows each patch of image to be represented in several ways.

The red connections all have the same weight.



Backpropagation with weight constraints

- It’s easy to modify the backpropagation algorithm to incorporate linear constraints between the weights.
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
  - So if the weights started off satisfying the constraints, they will continue to satisfy them.

$$\textit{To constrain}: \quad w_1 = w_2$$

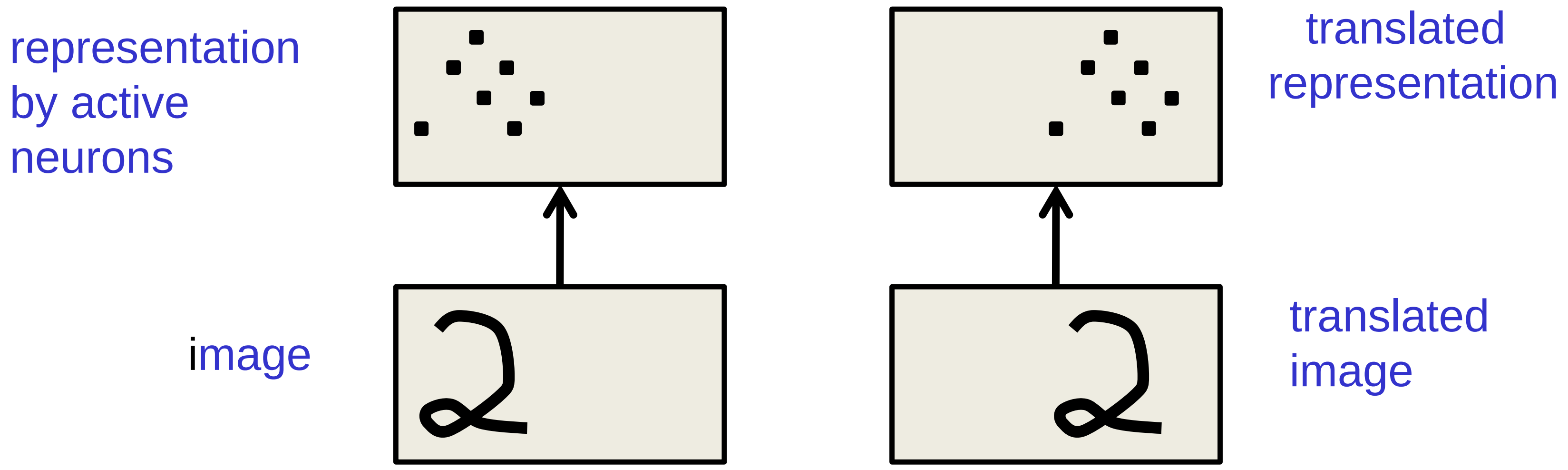
$$\textit{we need}: \quad \Delta w_1 = \Delta w_2$$

$$\textit{compute}: \quad \frac{\partial E}{\partial w_1} \quad \textit{and} \quad \frac{\partial E}{\partial w_2}$$

$$\textit{use} \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad \textit{for } w_1 \textit{ and } w_2$$

What does replicating the feature detectors achieve?

- Equivariant activities: Replicated features do not make the neural activities invariant to translation. The activities are equivariant.



- Invariant knowledge: If a feature is useful in some locations during training, detectors for that feature will be available in all locations during testing.

Pooling the outputs of replicated feature detectors

- Get a small amount of translational invariance at each level by averaging four neighboring replicated detectors to give a single output to the next level.
  - This reduces the number of inputs to the next layer of feature extraction, thus allowing us to have many more different feature maps.
  - Taking the maximum of the four works slightly better.
- Problem: After several levels of pooling, we have lost information about the precise positions of things.
  - This makes it impossible to use the precise spatial relationships between high-level parts for recognition.

Example Architecture for CIFAR-10

- [INPUT - CONV - RELU - POOL - FC]
- INPUT [32x32x3] : the raw pixel values of the image
- CONV will compute the output of neurons that are connected to local regions in the input. With 12 filters, the output volume is [32x32x12]
- RELU : apply an elementwise activation function, such as the max(0,x)
- POOL will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10

Convolution Layer

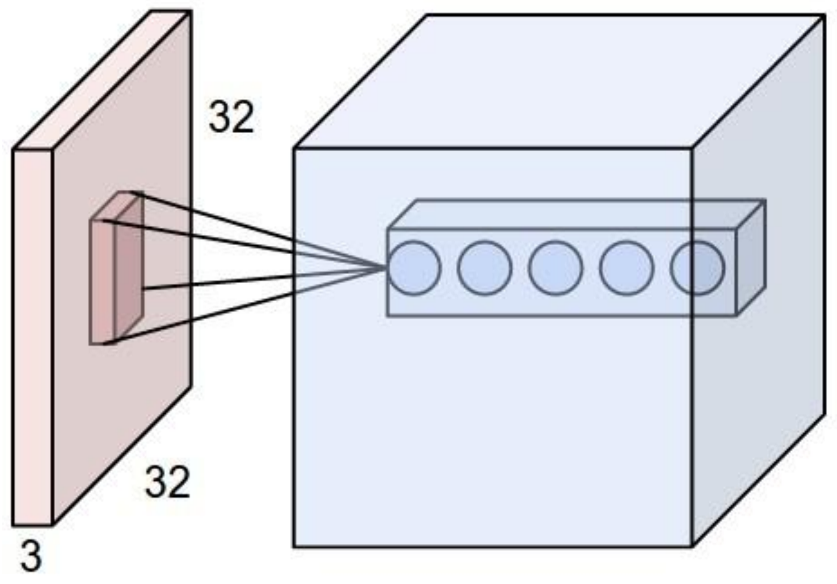
- The Conv layer is the core building block of a CNN
- The parameters consist of a set of learnable filters.
- Every filter is small spatially (width and height), but extends through the full depth of the input volume, eg, 5x5x3
- During the forward pass, we slide (convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position.
- produce a 2-dimensional activation map that gives the responses of that filter at every spatial position.
- Intuitively, the network will learn filters that activate when they see some type of visual feature
- A set of filters in each CONV layer
  - each of them will produce a separate 2-dimensional activation map
  - We will stack these activation maps along the depth dimension and produce the output volume.

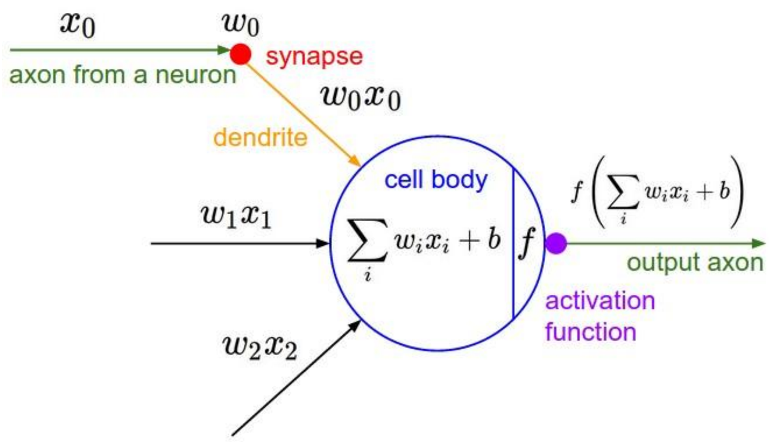
Convolutional Neural Network 2

Sudeshna Sarkar

23/2/17

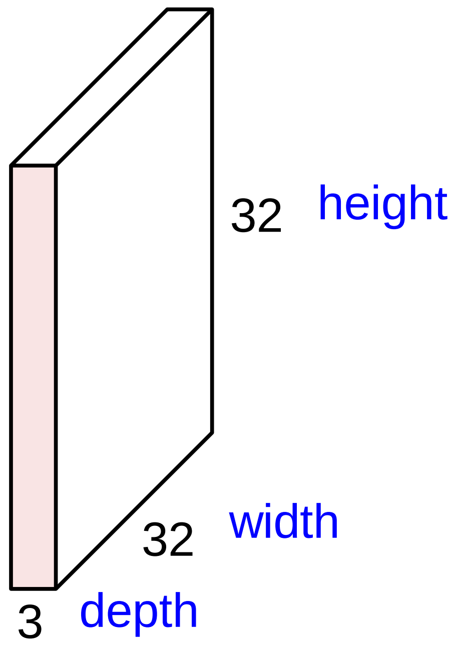
Convolution





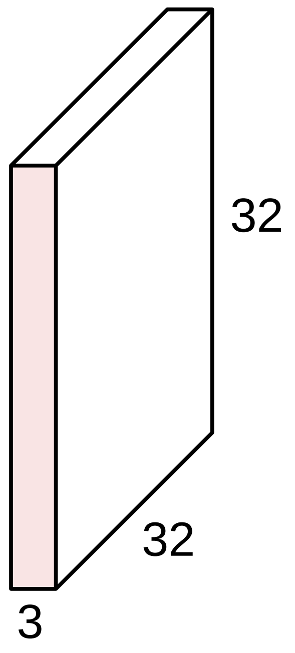
Convolutions: More detail

32x32x3 image

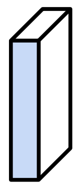


Convolutions: More detail

32x32x3 image



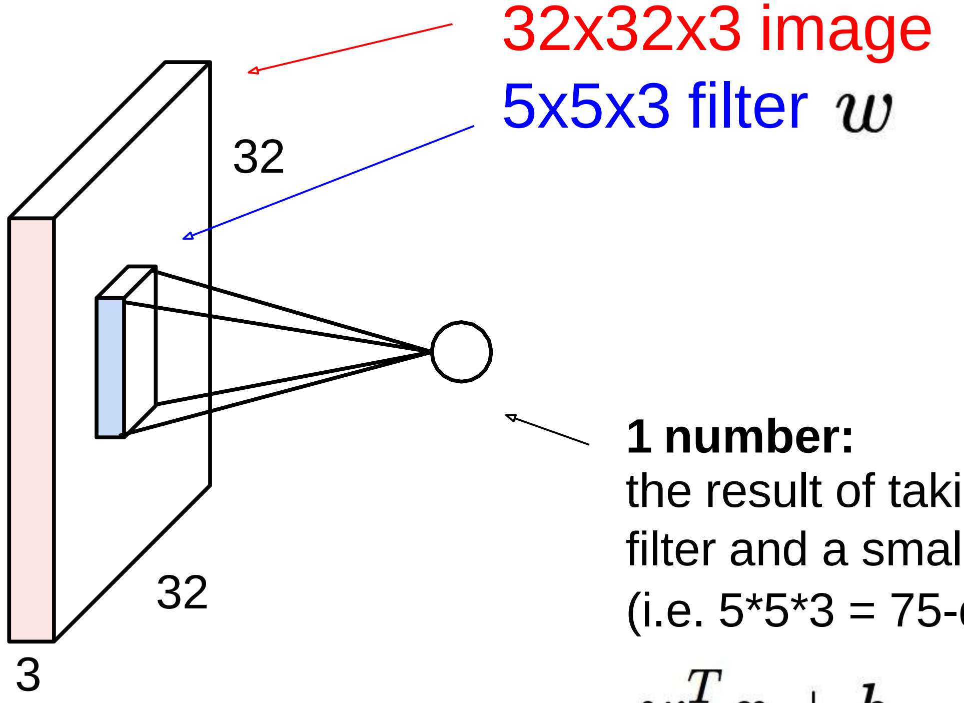
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolutions: More detail

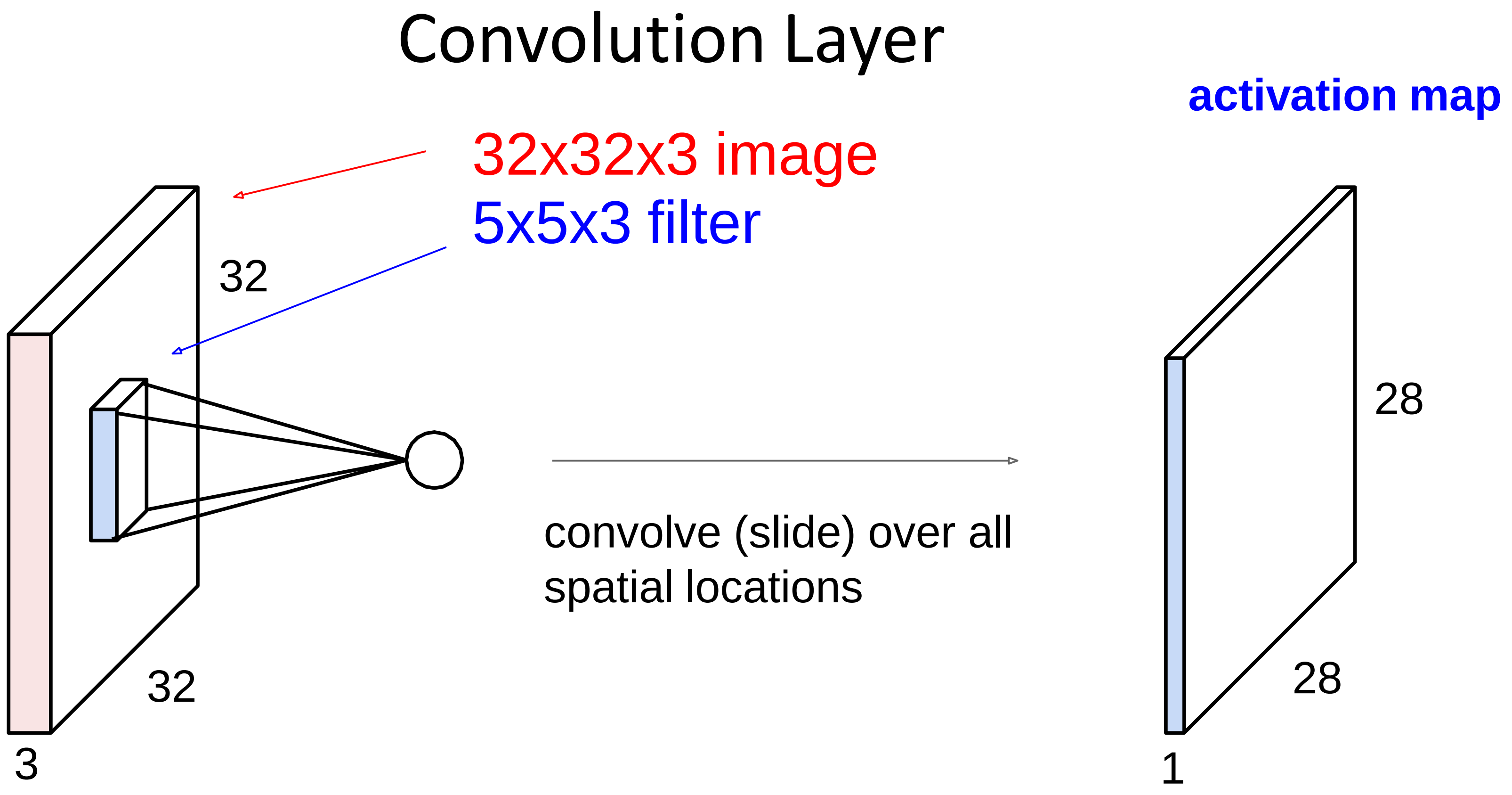
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolutions: More detail

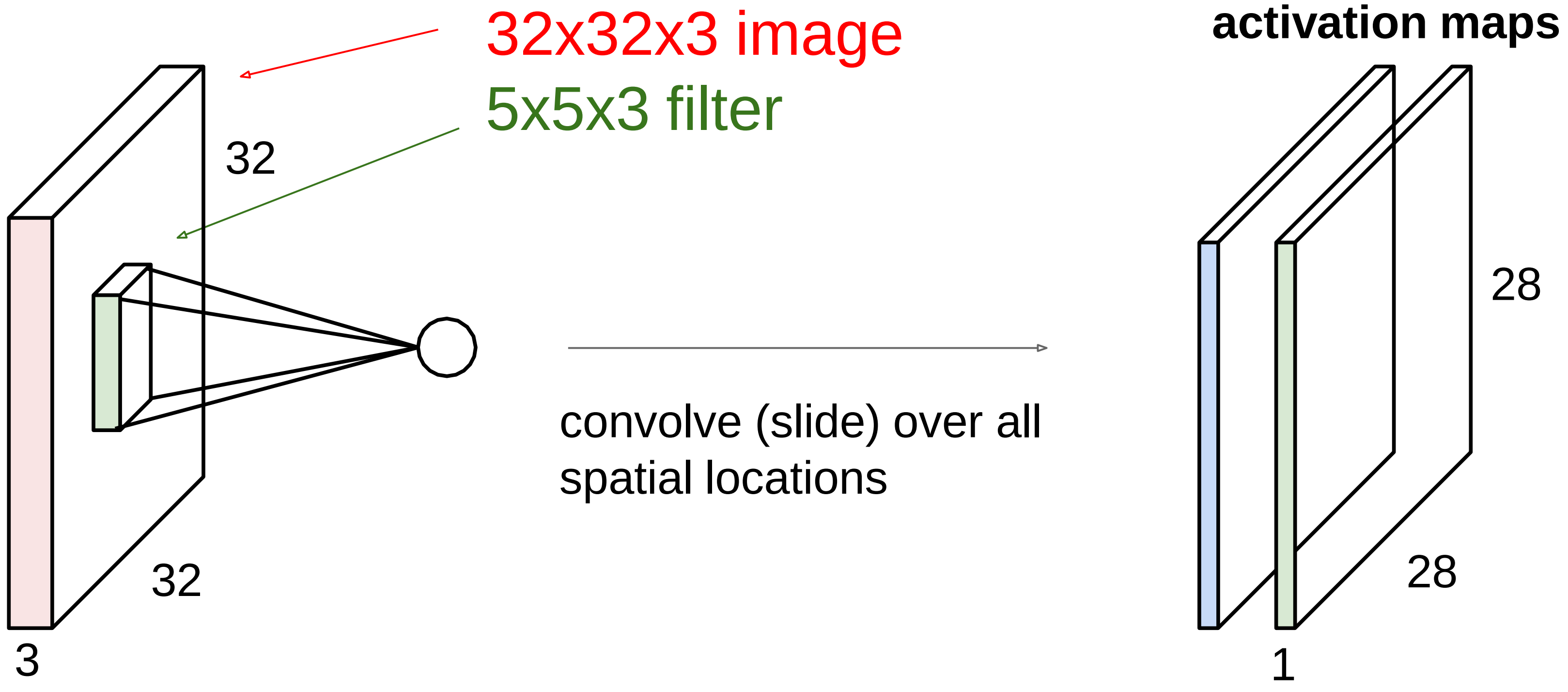


Andrej Karpathy

Convolutions: More detail

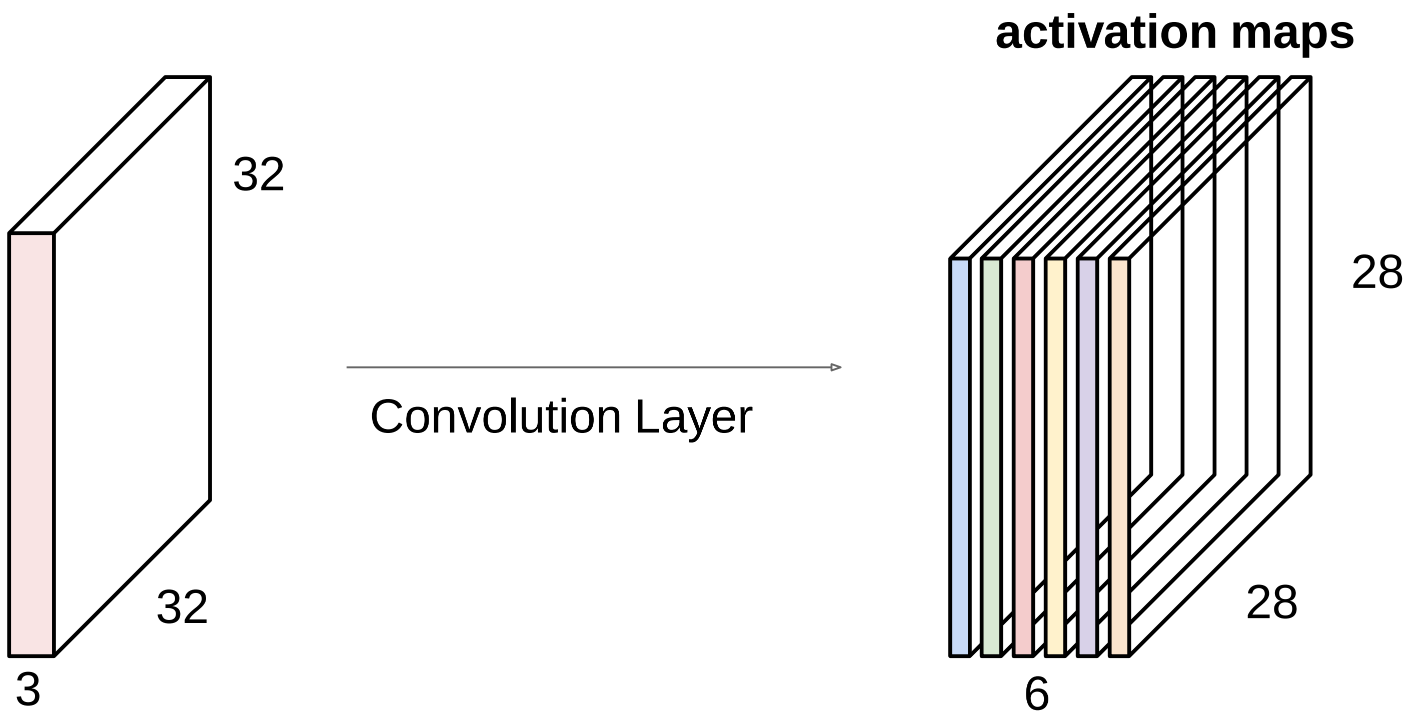
Convolution Layer

consider a second, green filter



Convolutions: More detail

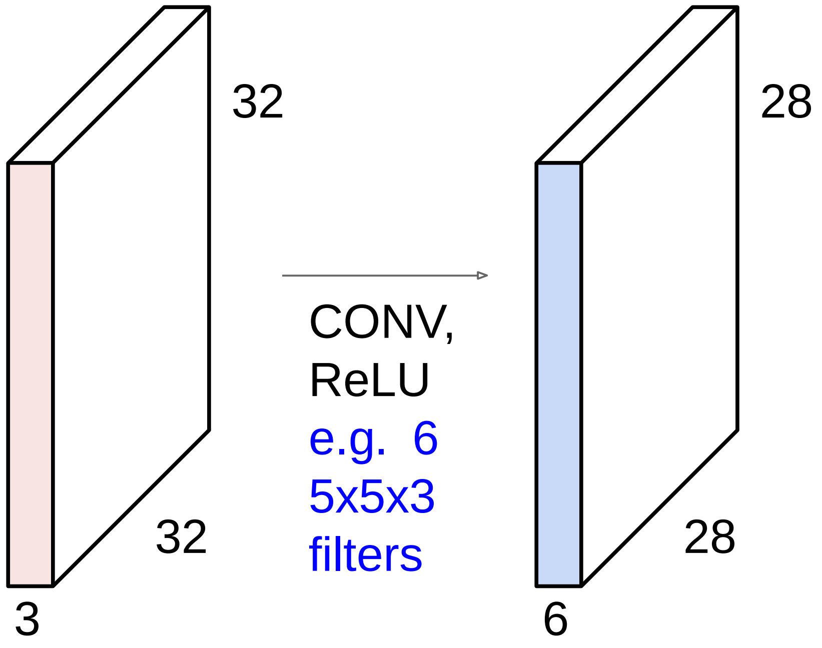
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Convolutions: More detail

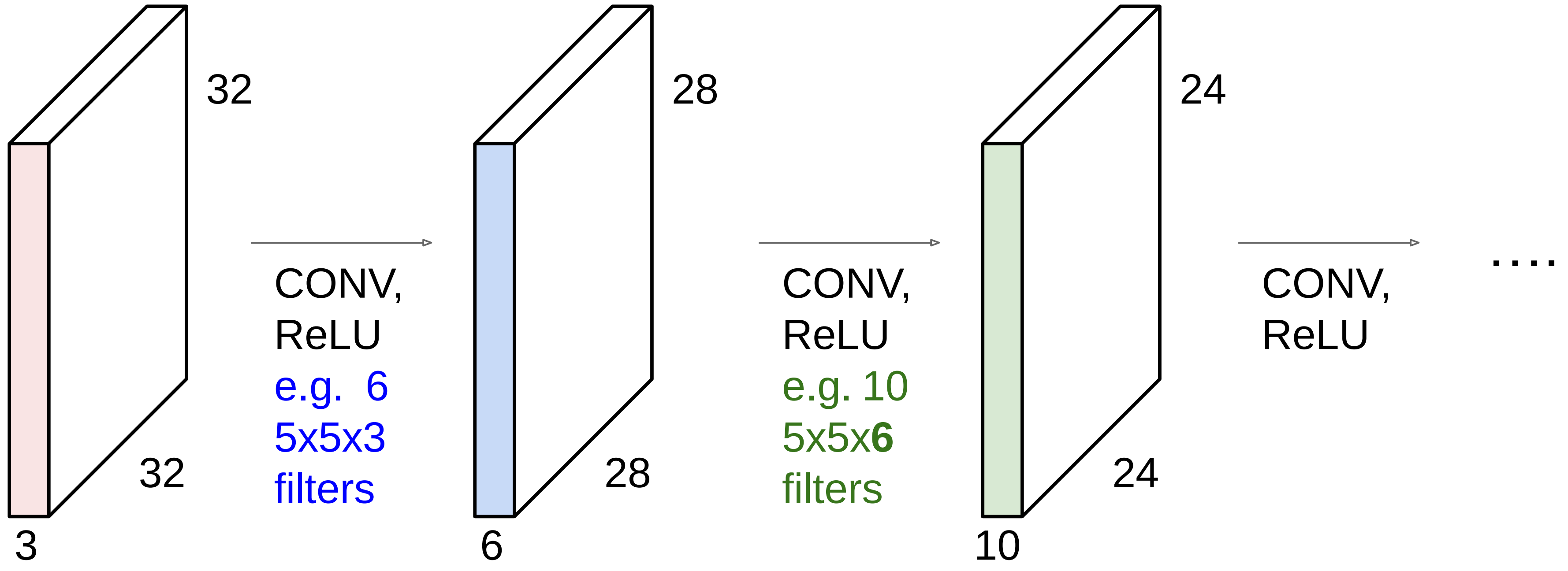
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Andrej Karpathy

Convolutions: More detail

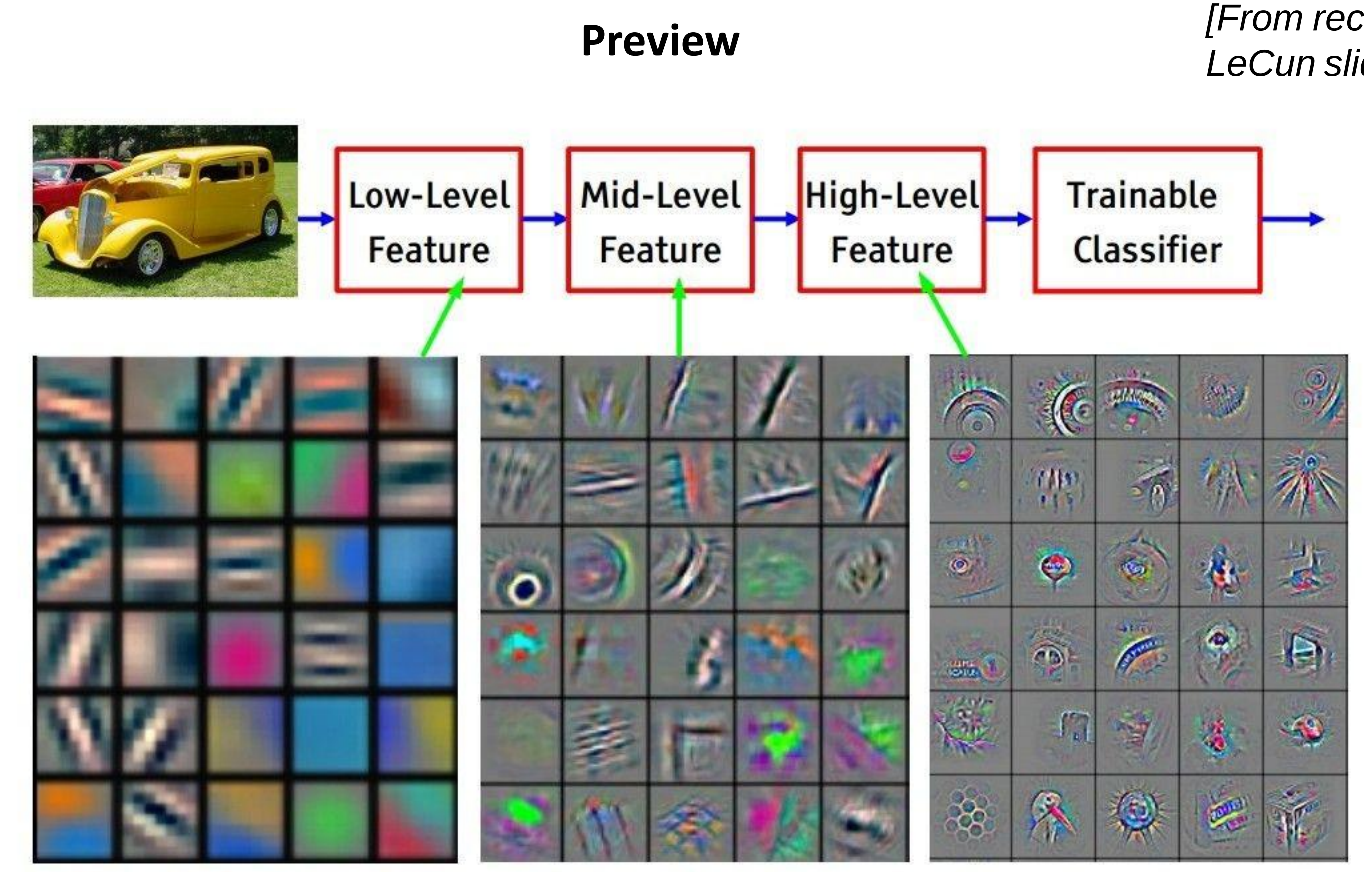
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Andrej Karpathy

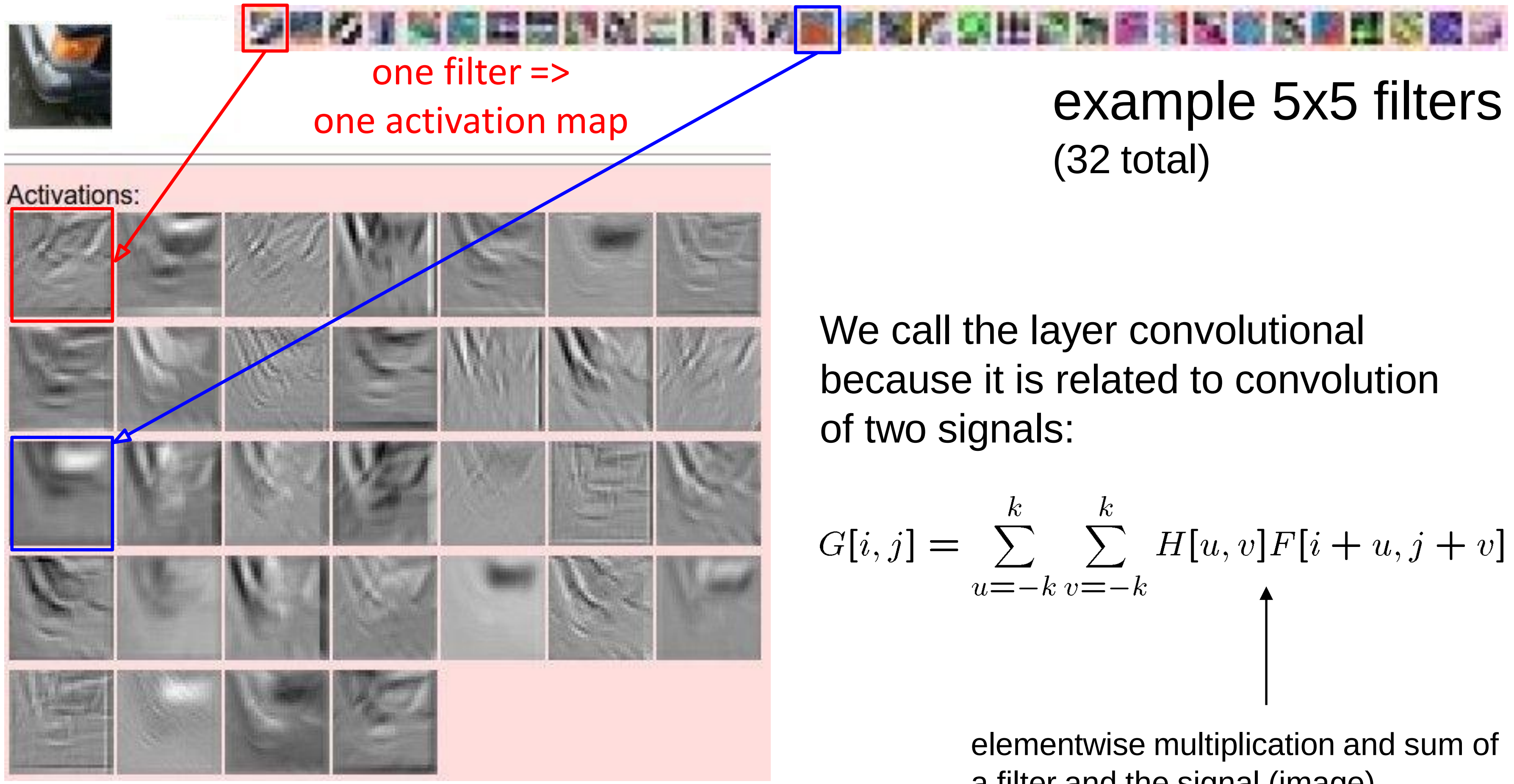
Convolutions: More detail

[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutions: More detail



example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

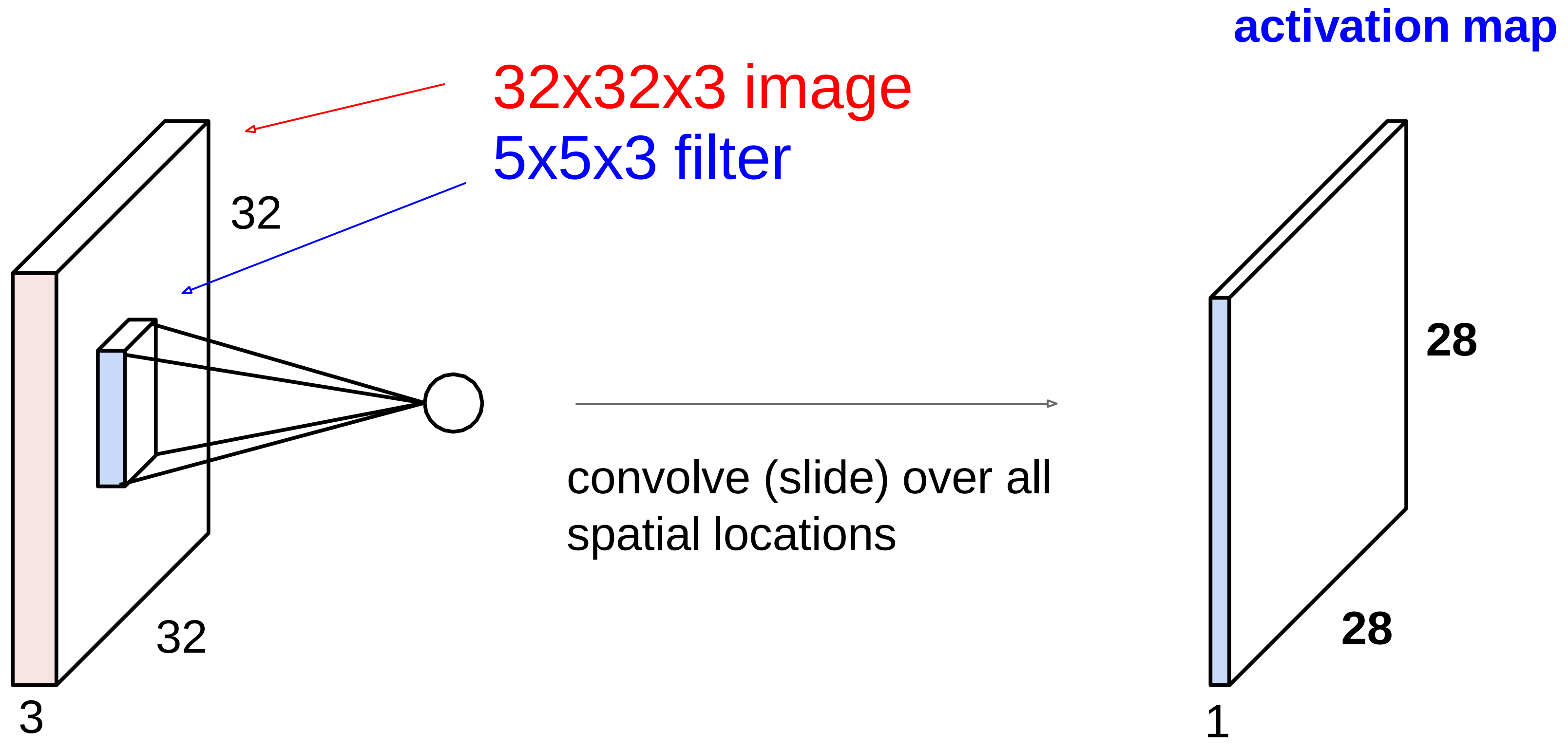
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

elementwise multiplication and sum of a filter and the signal (image)

Adapted from Andrej Karpathy, Kristen Grauman

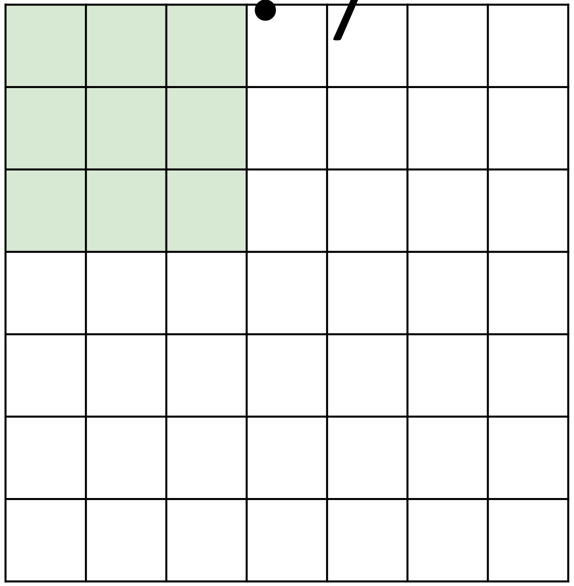
Convolutions: More detail

A closer look at spatial dimensions:



Convolutions: More detail

A closer look at spatial dimensions:



- 7

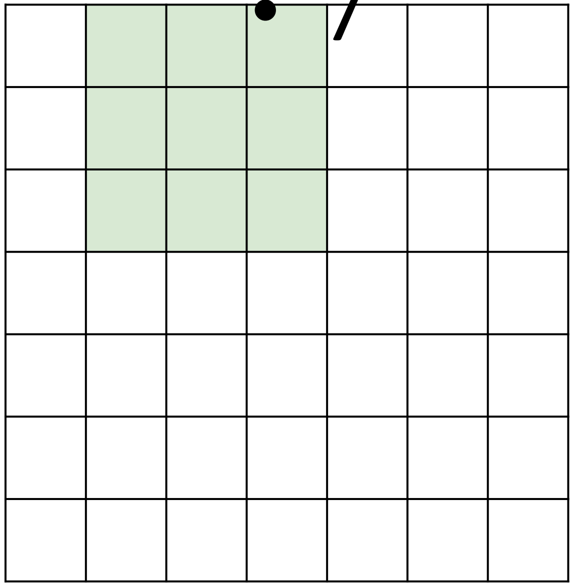
- 7x7 input (spatially) assume 3x3 filter

- 7

Andrej Karpathy

Convolutions: More detail

A closer look at spatial dimensions:



- 7

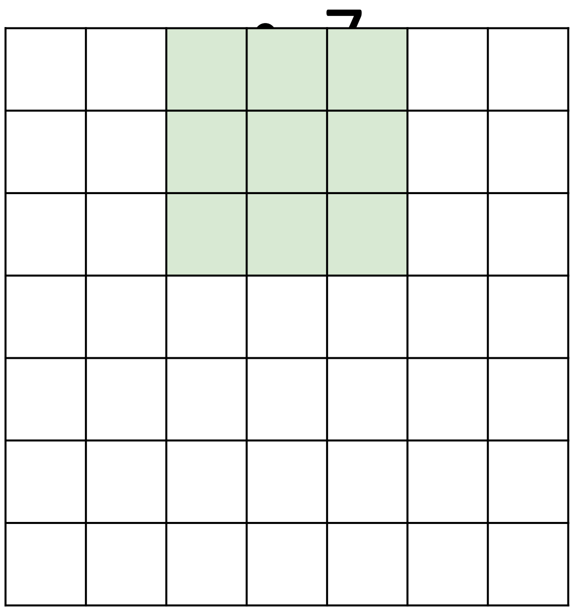
- 7x7 input (spatially) assume 3x3 filter

- 7

Andrej Karpathy

Convolutions: More detail

A closer look at spatial dimensions:



- 7x7 input (spatially) assume 3x3 filter

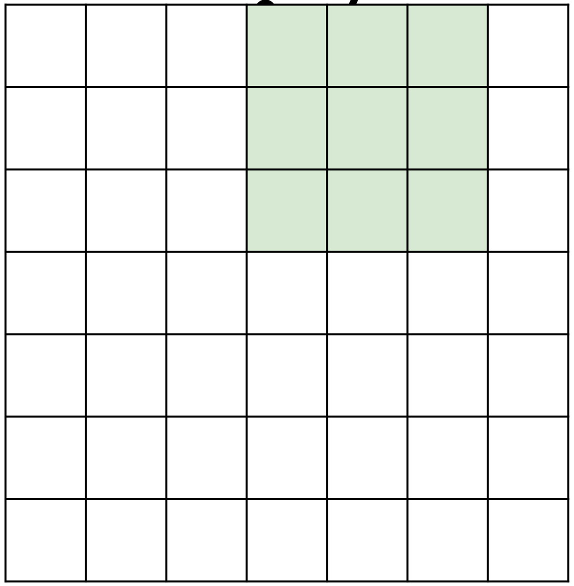
- 7

Andrej Karpathy

Convolutions: More detail

A closer look at spatial dimensions:

- 7



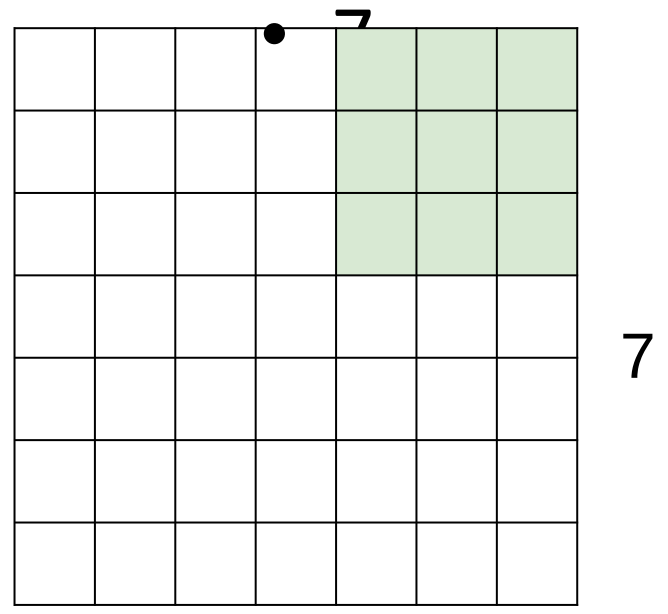
- 7x7 input (spatially) assume 3x3 filter

- 7

Andrej Karpathy

Convolutions: More detail

A closer look at spatial dimensions:

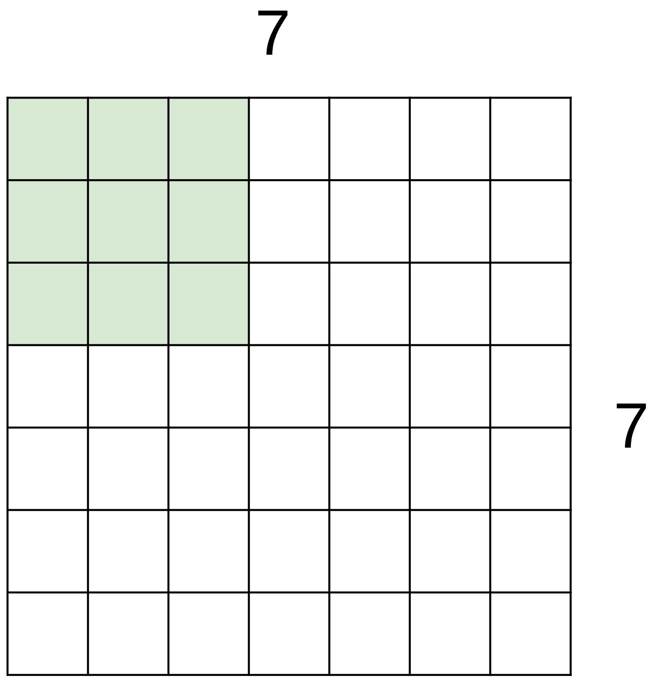


- 7x7 input (spatially)
  assume 3x3 filter

  => 5x5 output

Convolutions: More detail

A closer look at spatial dimensions:

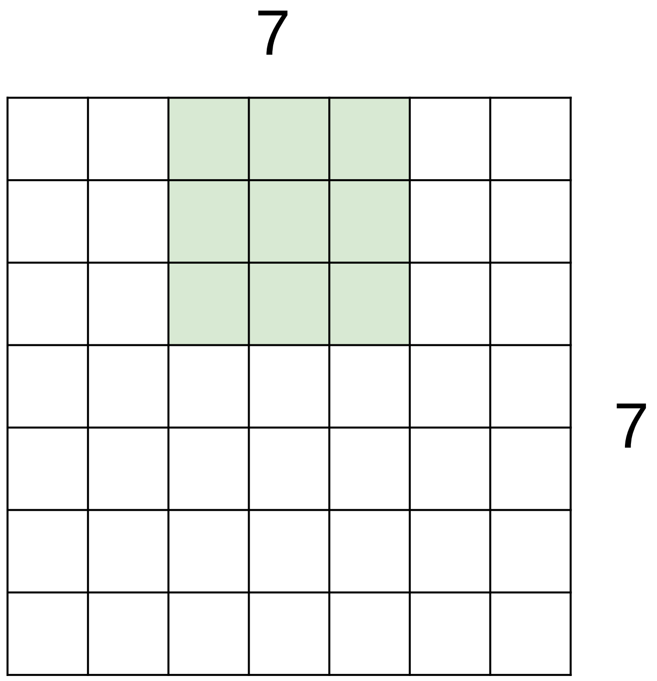


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Andrej Karpathy

Convolutions: More detail

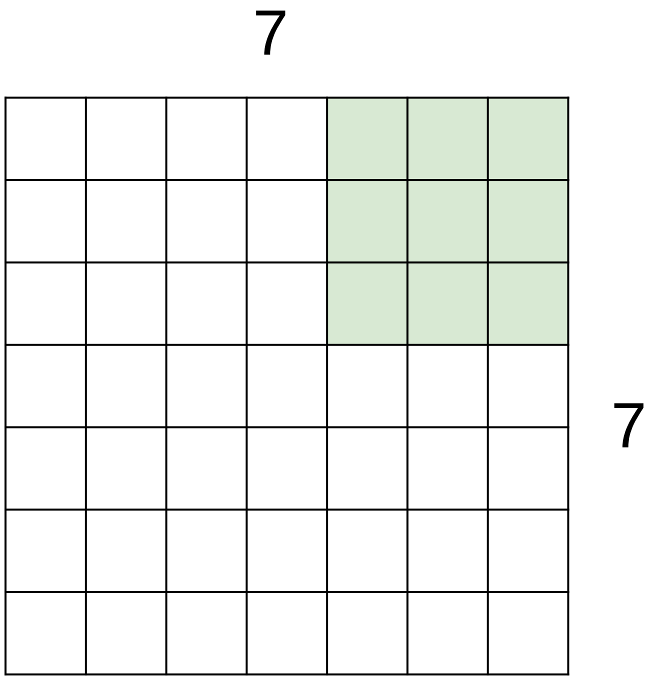
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolutions: More detail

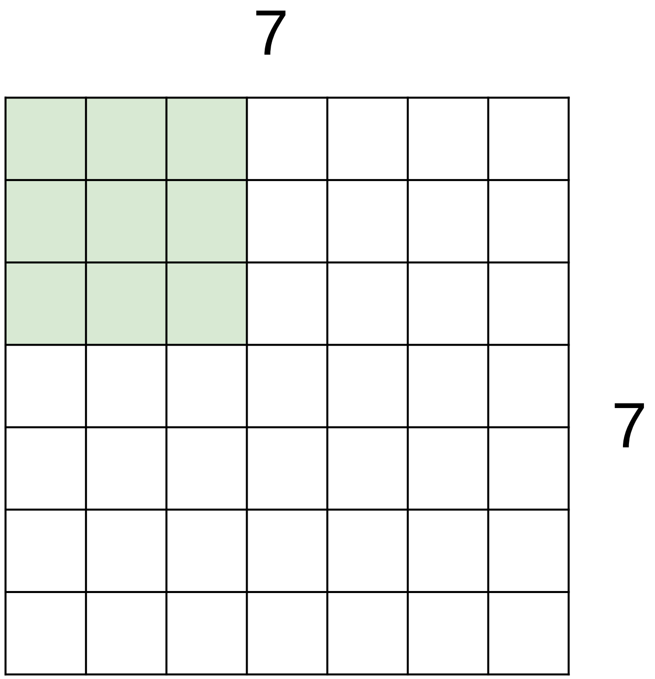
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Convolutions: More detail

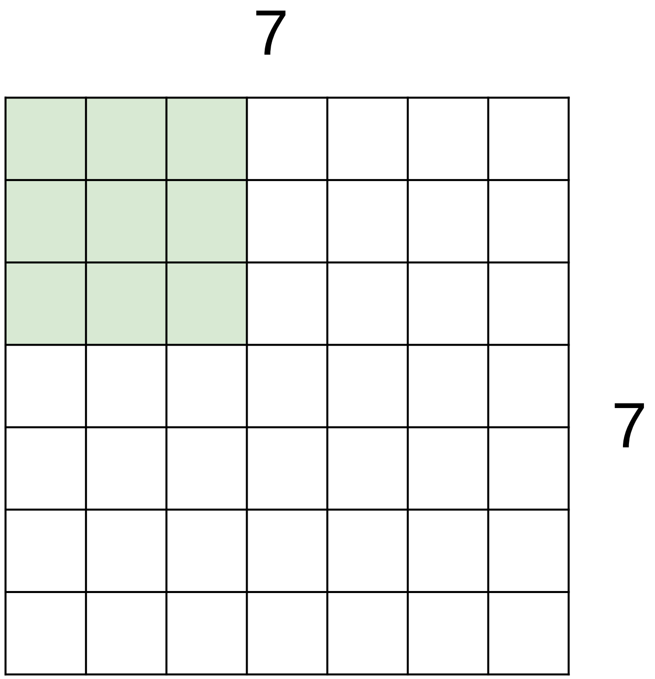
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

Convolutions: More detail

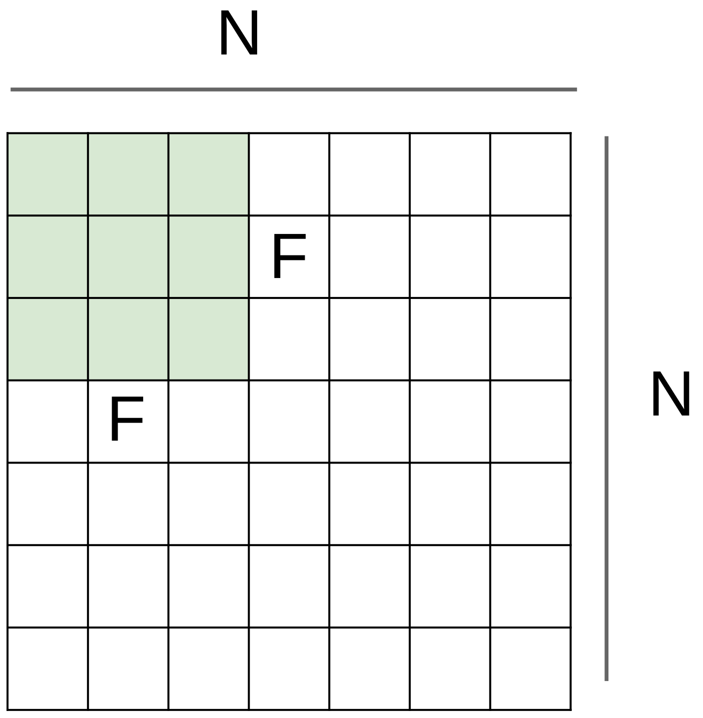
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Convolutions: More detail

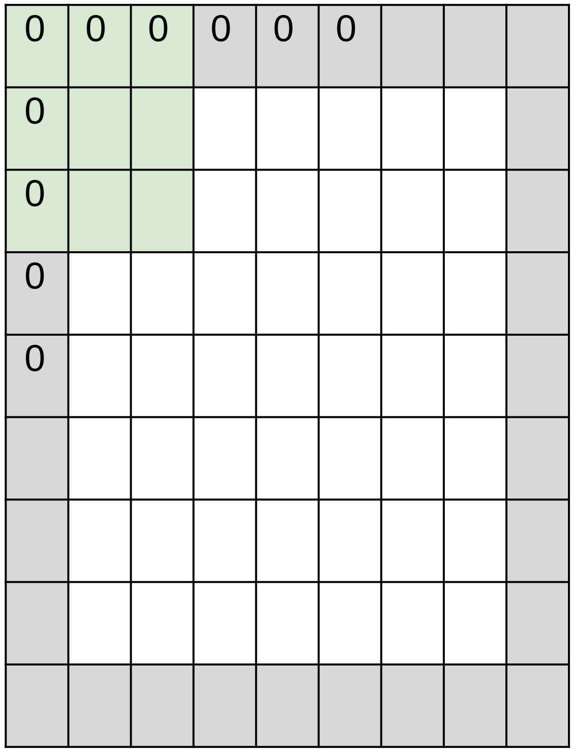


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Convolutions: More detail

In practice: Common to zero pad the border



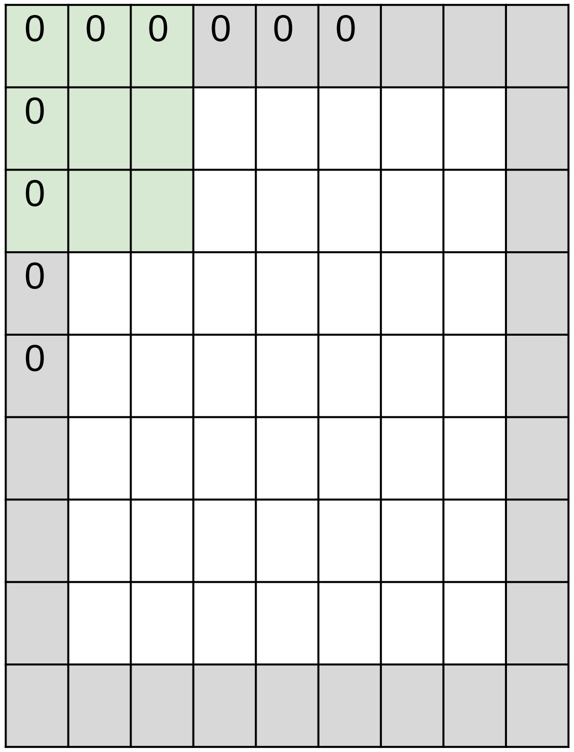
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Andrej Karpathy

Convolutions: More detail

In practice: Common to zero pad the border

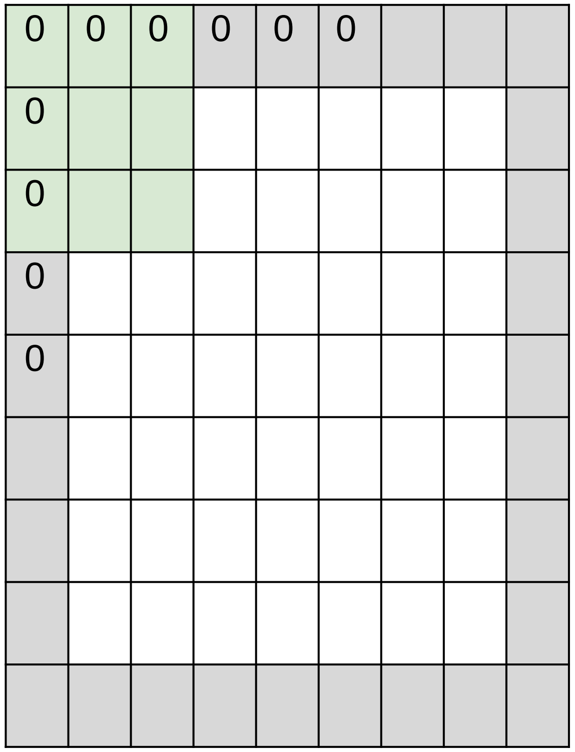


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Convolutions: More detail

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

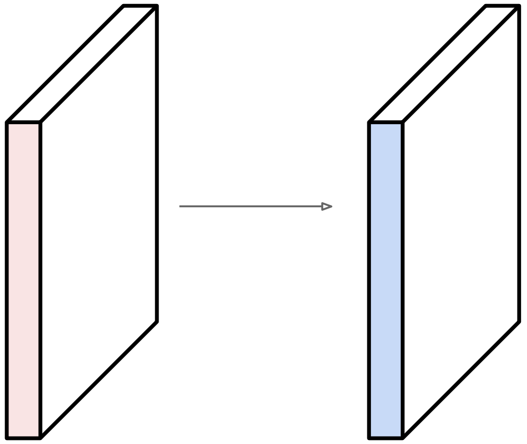
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

(N + 2*padding - F) / stride + 1

Andrej Karpathy

Convolutions: More detail

Examples time:

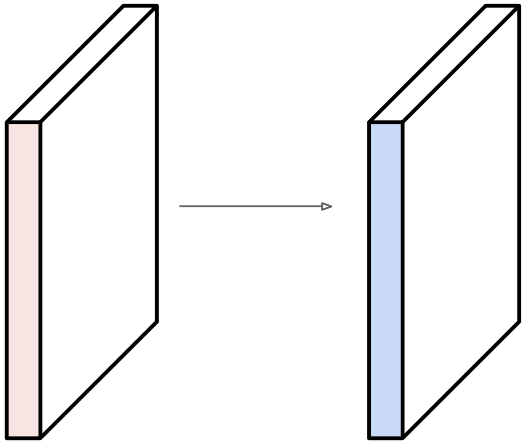


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Convolutions: More detail

Examples time:

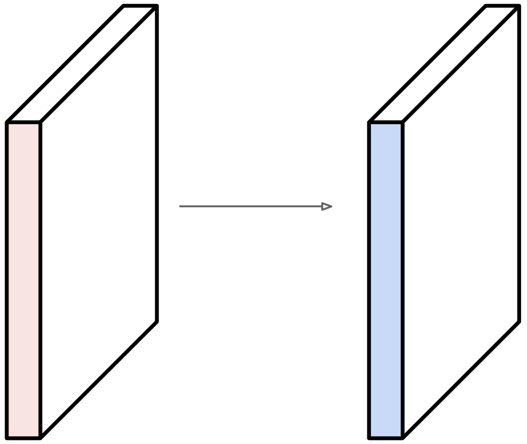


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Convolutions: More detail

Examples time:

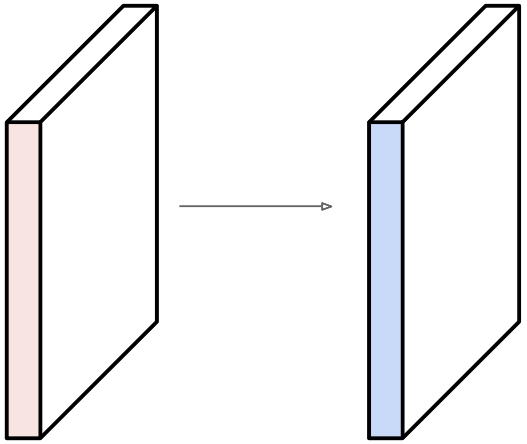


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Convolutions: More detail

Examples time:

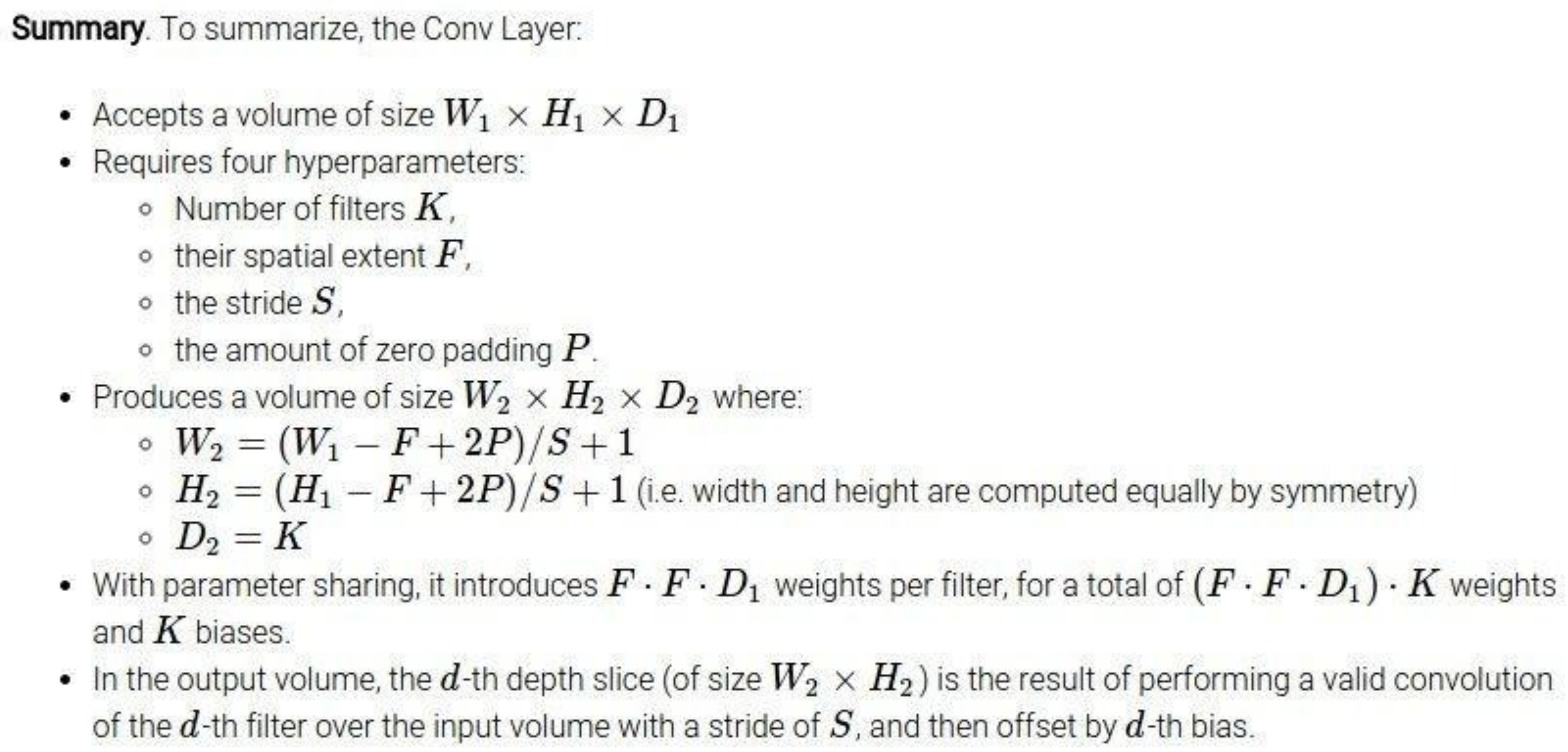


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Andrej Karpathy

Convolutions: More detail



Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

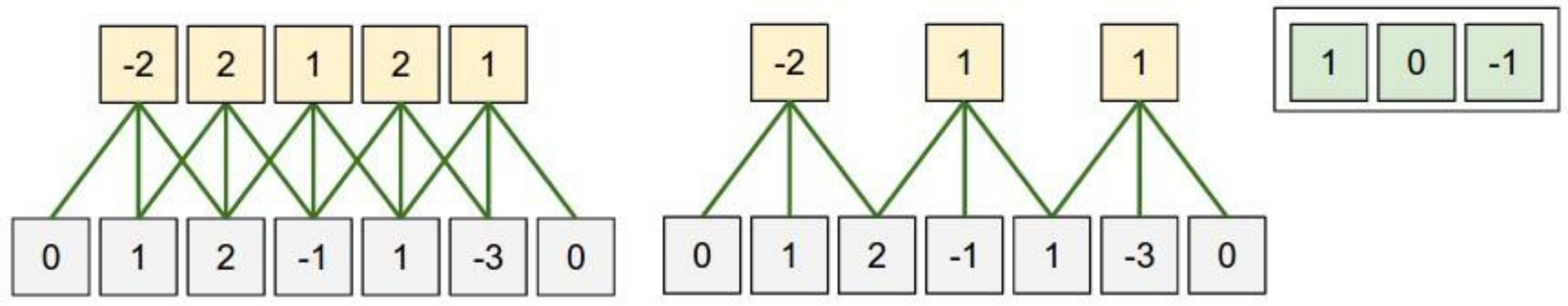
Andrej Karpathy

Spatial arrangement

- Three hyperparameters control the size of the output volume
  - Depth: no of filters, each learning to look for something different in the input.
  - the stride with which we slide the filter.
  - pad the input volume with zeros around the border.

Spatial arrangement

- We compute the spatial size of the output volume as a function of
  - the input volume size (W)
  - the receptive field size of the Conv Layer neurons (F)
  - the stride with which they are applied (S)
  - the amount of zero padding used (P) on the border.
- The number of neurons that “fit” is given by (W−F+2P)/(S+1)
  - For a 7x7 input and a 3x3 filter with stride 1 and pad 0 we would get a 5x5 output.
  - With stride 2 we would get a 3x3 output.

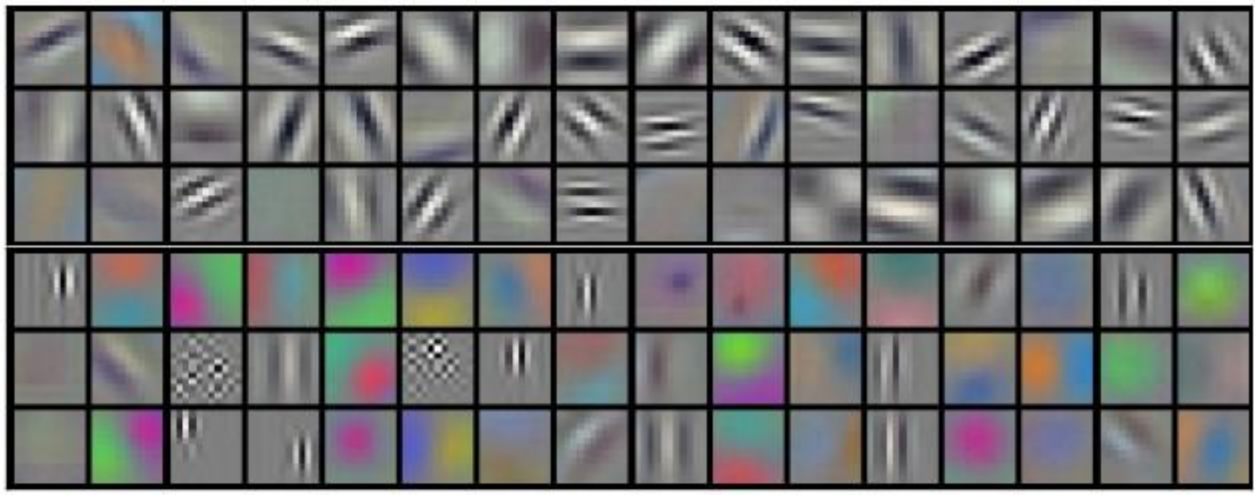


- one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and zero padding of P = 1
- Stride = 1, 2

- The Krizhevsky et al. architecture that won the ImageNet 2012
- images of size [227x227x3].
- the first Convolutional Layer, used neurons with receptive field size F=11, stride S=4, no zero padding P=0
- Since (227 - 11)/4 + 1 = 55, the Conv layer had a depth of K=96,
- the Conv layer output volume had size [55x55x96].
- Each of the 55*55*96 neurons in this volume was connected to a region of size [11x11x3] in the input volume.
- Moreover, all 96 neurons in each depth column are connected to the same [11x11x3] region of the input,

Parameter Sharing

- Parameter sharing controls the number of parameters.
- If there are 55*55*96 = 290,400 neurons in the first Conv Layer, and each has 11*11*3 = 363 weights and 1 bias. Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the ConvNet alone.
- Reduce by parameter sharing
- now have only 96 unique set of weights (one for each depth slice), for a total of 96*11*11*3 = 34,848 unique weights, or 34,944 parameters (+96 biases)
- During backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.



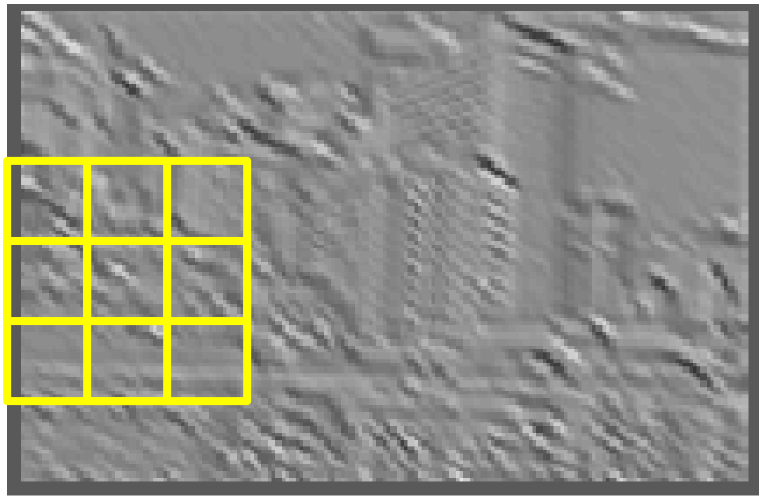
- Example filters learned by Krizhevsky.
- 96 filters each of size [11x11x3], each is shared by the 55*55 neurons in one depth slice.

Summary of Conv Layer

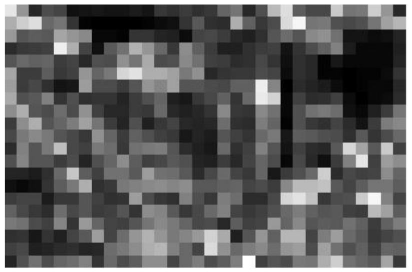
- Accepts a volume of size $W1 \times H1 \times D1$
- Requires four hyperparameters:
  - Number of filters K
  - their spatial extent F
  - the stride S
  - the amount of zero padding P
- Produces a volume of size $W2 \times H2 \times D2$
  - $W2=(W1-F+2P)/S+1$
  - $H2=(H1-F+2P)/S+1$
  - $D2=K$
- With parameter sharing, it introduces $F \cdot F \cdot D1$ weights per filter, for a total of $(F \cdot F \cdot D1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size W2×H2) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Spatial Pooling

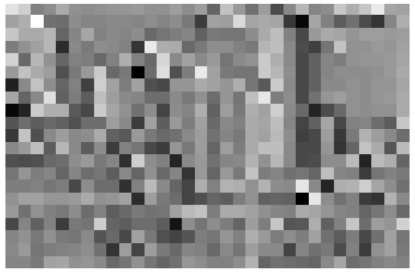
- Sum or max over non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (neurons see more of input)



Max



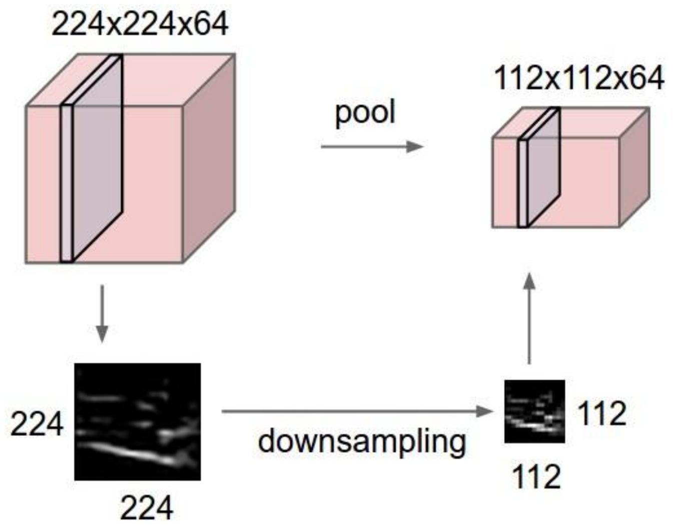
Sum

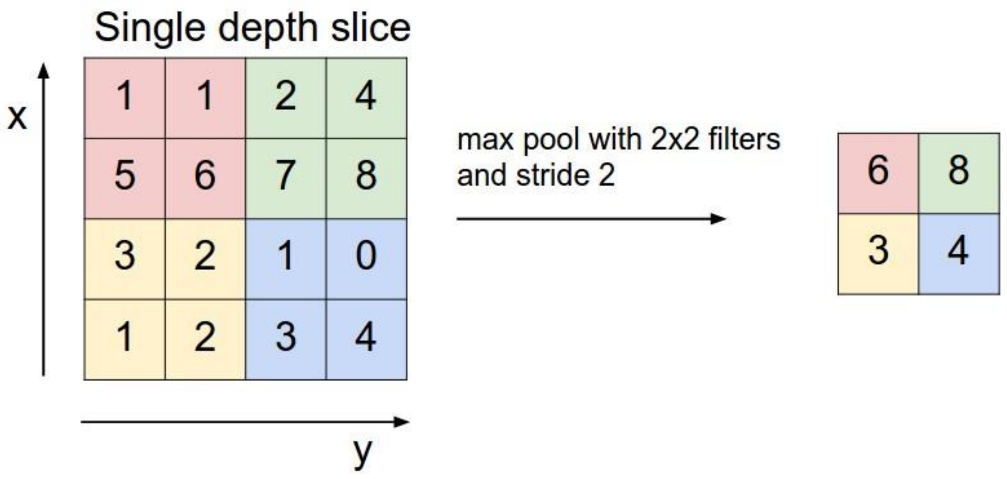


Adapted from Rob Fergus

3. Spatial Pooling

- Sum or max over non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (neurons see more of input)





Rob Fergus, figure from Andrej Karpathy

Pooling Layer

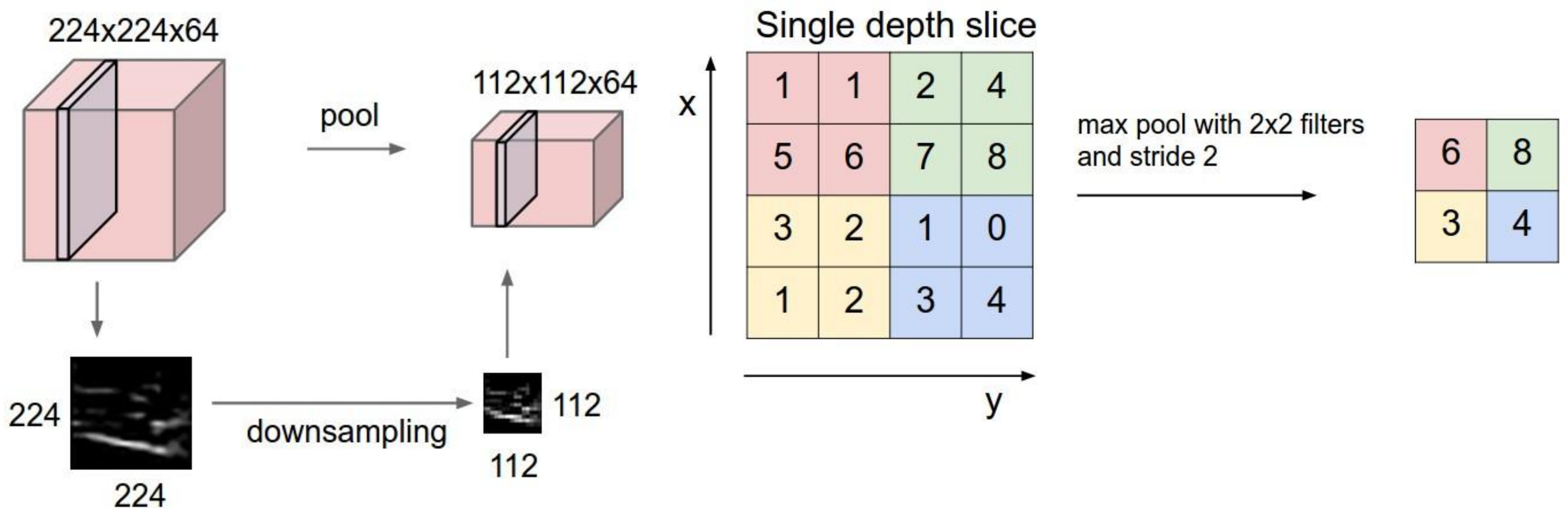
- Insertion of pooling layer:
  - reduce the spatial size of the representation

  reduce the amount of parameters and computation in the network, and hence also control overfitting.
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.
- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 -- downsamples every depth slice in the input by 2 along both width and height,
- MAX operation would in take a max over 4 numbers (little 2x2 region in some depth slice).
- The depth dimension remains unchanged.

General pooling layer

- Accepts a volume of size W1×H1×D1
- Requires two hyperparameters:
  - their spatial extent F
  - the stride S
- Produces a volume of size W2×H2×D2 where:
  - $W2=(W1-F)/S+1$
  - $H2=(H1-F)/S+1$
  - $D2=D1$
- Introduces zero parameters
- Other pooling functions: Average pooling, L2-norm pooling

General pooling



- Backpropagation. the backward pass for a max(x, y) operation routes the gradient to the input that had the highest value in the forward pass.
- Hence, during the forward pass of a pooling layer you may keep track of the index of the max activation (sometimes also called the switches) so that gradient routing is efficient during backpropagation.

Getting rid of pooling

1. **Striving for Simplicity: The All Convolutional Net** proposes to discard the pooling layer and have an architecture that only consists of repeated CONV layers.

- To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.
- Argument:
  - The purpose of pooling layers is to perform dimensionality reduction to widen subsequent convolutional layers' receptive fields.
  - The same effect can be achieved by using a convolutional layer: using a stride of 2 also reduces the dimensionality of the output and widens the receptive field of higher layers.
- The resulting operation differs from a max-pooling layer in that
  - it cannot perform a true max operation
  - it allows pooling across input channels.

Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." arXiv preprint arXiv:1412.6806 (2014).

Getting Rod of Pooling 2

2. **Very Deep Convolutional Networks for Large-Scale Image Recognition.**

- The core idea here is that hand-tuning layer kernel sizes to achieve optimal receptive fields (say, 5×5 or 7×7) can be replaced by simply stacking homogenous 3×3 layers.
- The same effect of widening the receptive field is then achieved by layer composition rather than increasing the kernel size
  - three stacked 3×3 have a 7×7 receptive field.
  - At the same time, the number of parameters is reduced:
  - a 7×7 layer has 81% more parameters than three stacked 3×3 layers.

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

Fully-connected layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer
- Their activations can hence be computed with a matrix multiplication followed by a bias offset.
- **Converting FC layers to CONV layers**
- the only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters.
- However, the neurons in both layers still compute dot products, so their functional form is identical.

Converting FC layers to CONV layers

- For any CONV layer there is an FC layer that implements the same forward function.
- The weight matrix would be a large matrix that is mostly zero except for at certain blocks (due to local connectivity) where the weights in many of the blocks are equal (due to parameter sharing).
- Conversely, any FC layer can be converted to a CONV layer.
- For example, an FC layer with K=4096 that is looking at some input volume of size 7×7×512
- can be equivalently expressed as a CONV layer with F=7,P=0,S=1,K=4096.
- In other words, we are setting the filter size to be exactly the size of the input volume, and hence the output will simply be 1×1×4096 since only a single depth column "fits" across the input volume, giving identical result as the initial FC layer.

ConvNet Architectures

Layer Patterns

- The most common architecture
- stacks a few CONV-RELU layers,
- follows them with POOL layers,
- and repeats this pattern until the image has been merged spatially to a small size.
- At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores. In other words, the most common ConvNet architecture follows the pattern:

INPUT -> [[CONV -> RELU]*N -> POOL?]*M ->[FC -> RELU]*K -> FC

- N >= 0 (and usually N <= 3), M >= 0, K >= 0

Prefer a stack of small filter CONV to one large receptive field CONV layer.

three layers of 3x3 CONV vs a single CONV layer with 7x7 receptive fields.

- The receptive field size is identical in spatial extent (7x7), but with several disadvantages.
  1. The neurons would be computing a linear function over the input, while the three stacks of CONV layers contain non-linearities that make their features more expressive.
  2. If we suppose that all the volumes have C channels, the single 7x7 CONV layer would contain $C\times(7\times7\times C)=49C^2$ parameters, while the three 3x3 CONV layers would contain $3\times(C\times(3\times3\times C))=27C^2$ parameters.
- Intuitively, stacking CONV layers with tiny filters as opposed to having one CONV layer with big filters allows us to express more powerful features of the input, and with fewer parameters.

Recent Departures

- The conventional paradigm of a linear list of layers has recently been challenged, in
  1. Google’s Inception architectures
  2. current (state of the art) Residual Networks from Microsoft Research Asia.
- Both of these feature more intricate and different connectivity structures.

Case Studies

- **LeNet**. The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990's. was used to read zip codes, digits, etc.
- **AlexNet**. popularized Convolutional Networks in Computer Vision, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton.
- The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other
- **ZF Net**. The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus.
- An improvement on AlexNet by tweaking the architecture hyperparameters, -- expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.

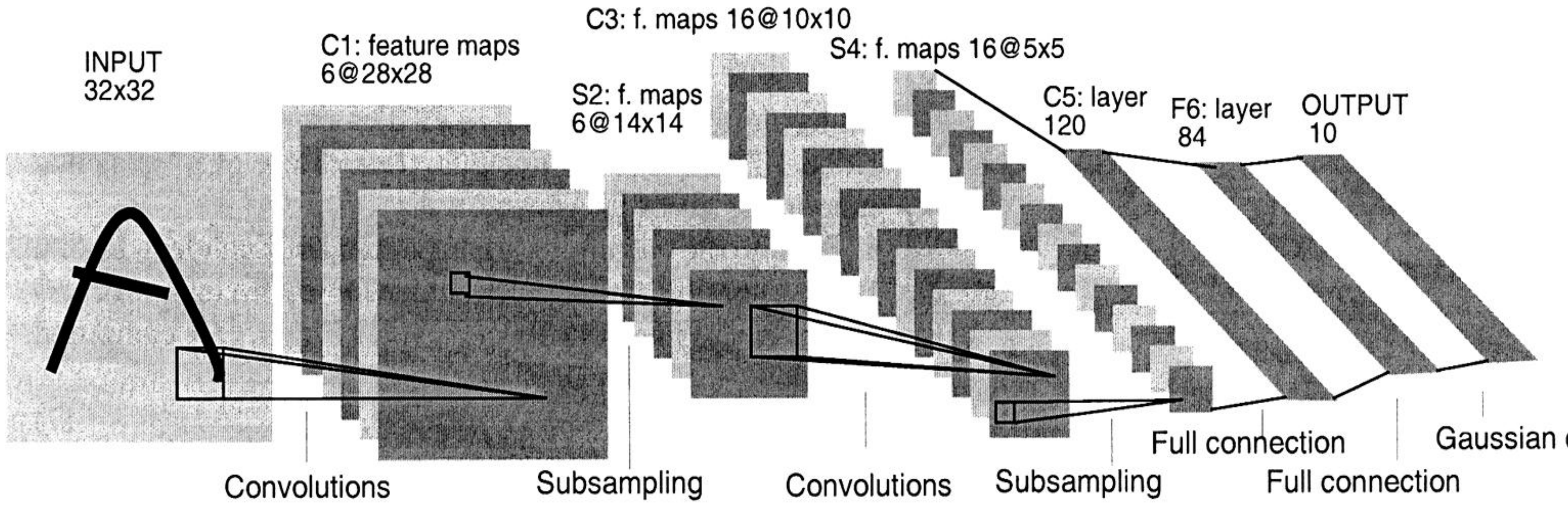
Case Studies

- **GoogLeNet**. The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google.
- Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).
- Uses Average Pooling instead of Fully Connected layers at the top of the ConvNet
- There are also several followup versions to the GoogLeNet, most recently Inception-v4.
- **VGGNet**. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman.
- Showed that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers
- features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.
- A downside: that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

LeNet

- Yann LeCun and his collaborators developed a really good recognizer for handwritten digits.
- This net was used for reading ~10% of the checks in North America.
- Demo at http://yann.lecun.com

The architecture of LeNet5



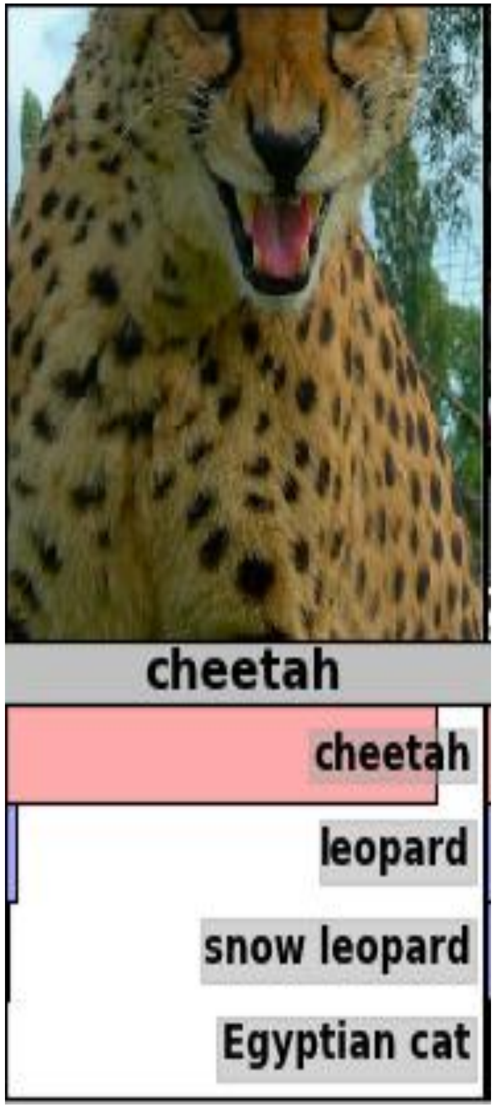
From hand-written digits to 3-D objects

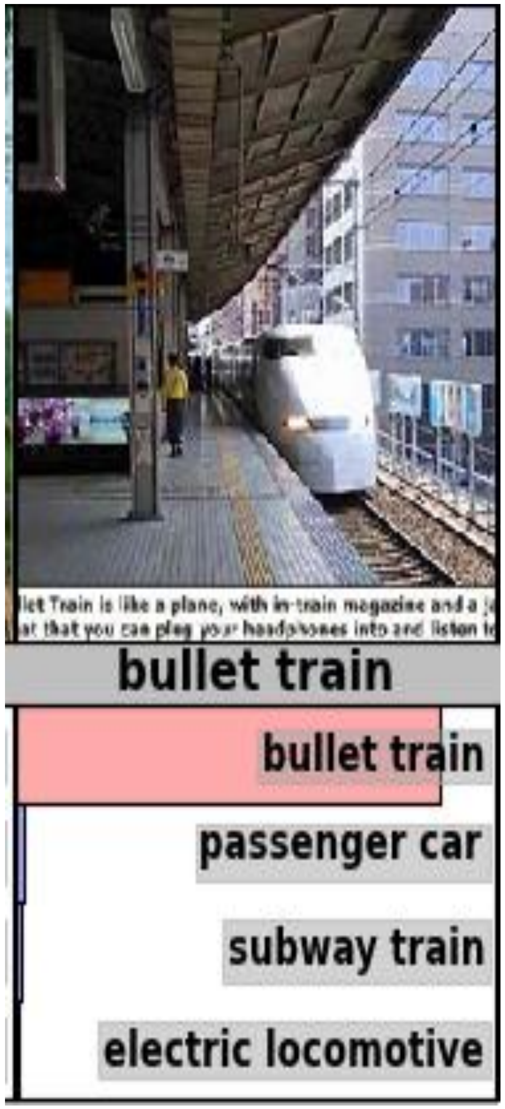
- Recognizing real objects in color photographs downloaded from the web is much more complicated than recognizing hand-written digits:
  - Hundred times as many classes (1000 vs 10)
  - Hundred times as many pixels (256 x 256 color vs 28 x 28 gray)
  - Two dimensional image of three-dimensional scene.
  - Cluttered scenes requiring segmentation
  - Multiple objects in each image.
- Will the same type of convolutional neural network work?

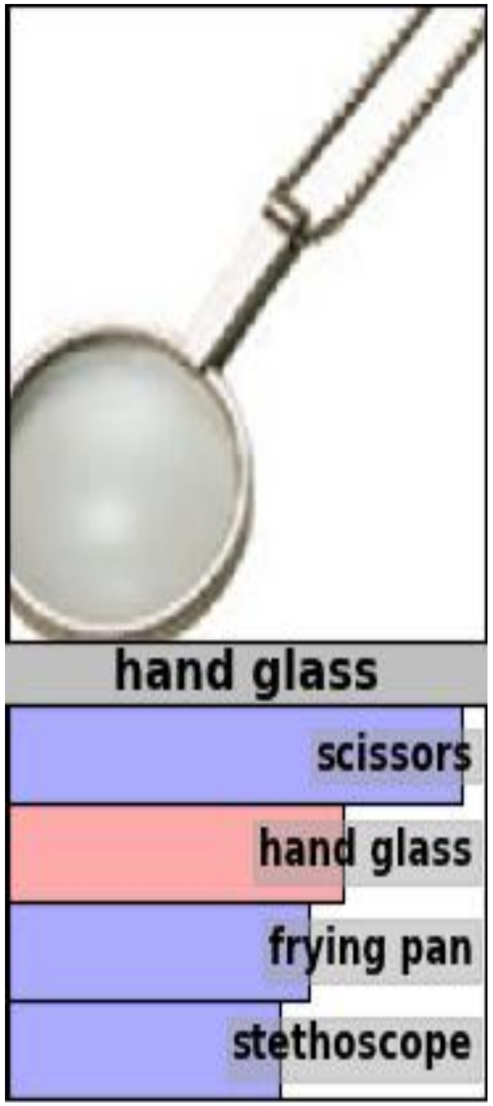
The ILSVRC-2012 competition on ImageNet

- The dataset has 1.2 million high-resolution training images.
- The classification task:
  - Get the “correct” class in your top 5 bets. There are 1000 classes.
- The localization task:
  - For each bet, put a box around the object. Your box must have at least 50% overlap with the correct box.
- Some of the best existing computer vision methods were tried on this dataset by leading computer vision groups from Oxford, INRIA, XRCE, …
  - Computer vision systems use complicated multi-stage systems.
  - The early stages are typically hand-tuned by optimizing a few parameters.

Examples from the test set (with the network’s guesses)







A neural network for ImageNet

- Alex Krizhevsky (NIPS 2012) developed a very deep convolutional neural net of the type pioneered by Yann Le Cun. Its architecture was:
  - 7 hidden layers not counting some max pooling layers.
  - The early layers were convolutional.
  - The last two layers were globally connected.
  - 650000 units, 60 million params
- The activation functions were:
  - Rectified linear units in every hidden layer. These train much faster and are more expressive than logistic units.
  - Competitive normalization to suppress hidden activities when nearby units have stronger activities. This helps with variations in intensity.

A Common Architecture: AlexNet

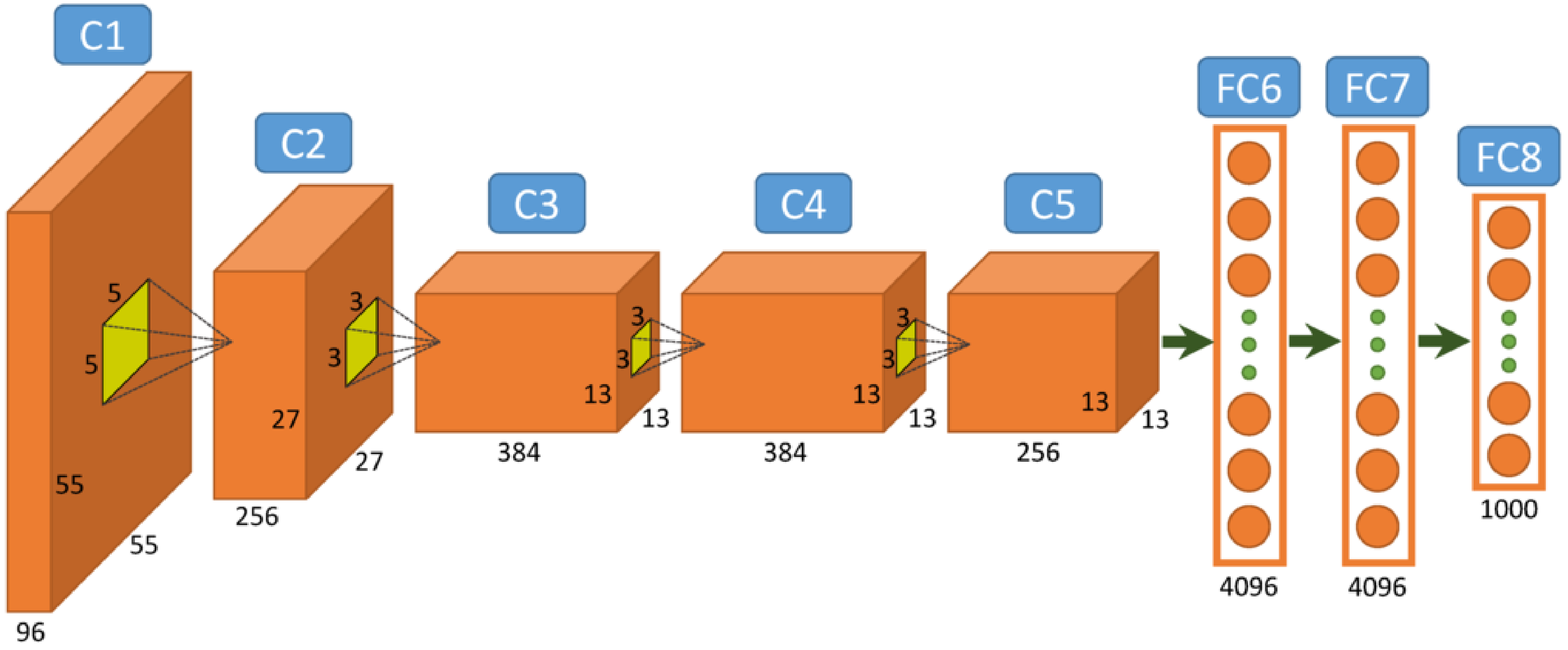


Figure from http://www.mdpi.com/2072-4292/7/11/14680/htm

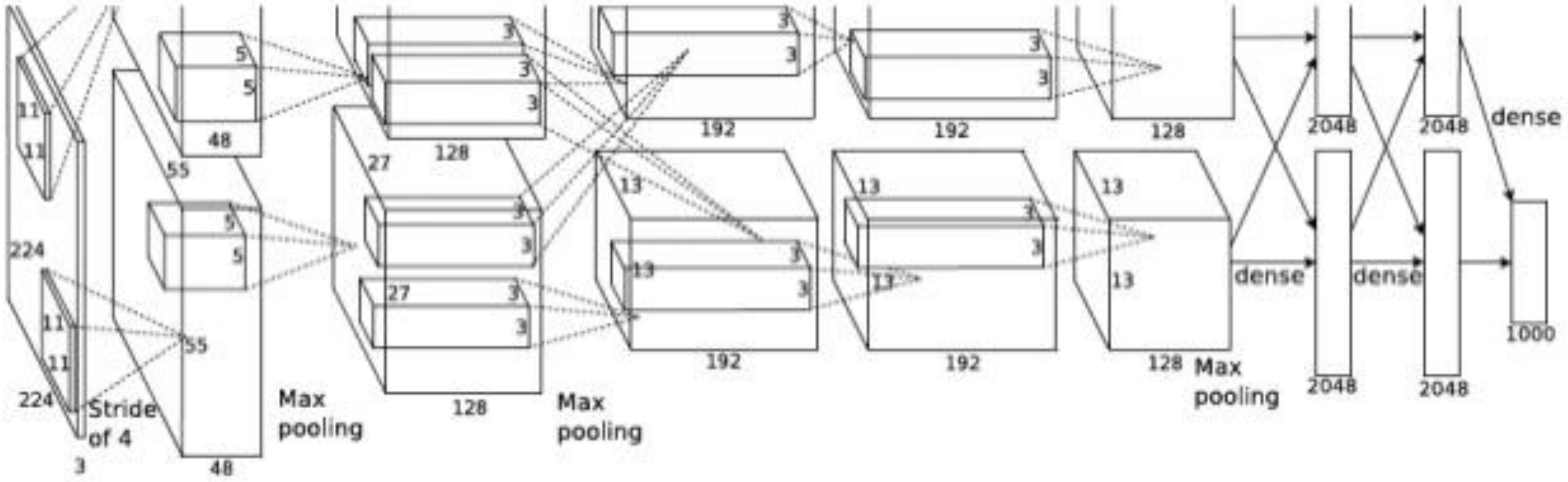
University of Toronto (Alex Krizhevsky)	16.4%	34.1%

Error rates on the ILSVRC-2012 competition

	classification	classification &localization
University of Tokyo	26.1%	53.6%
Oxford University Computer Vision Group	26.9%	50.0%
INRIA (French national research institute in CS) + XRCE (Xerox Research Center Europe)	27.0%	
University of Amsterdam	29.5%	

AlexNet

- Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data (10^6 vs. 10^3 images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week



A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Tricks that significantly improve generalization

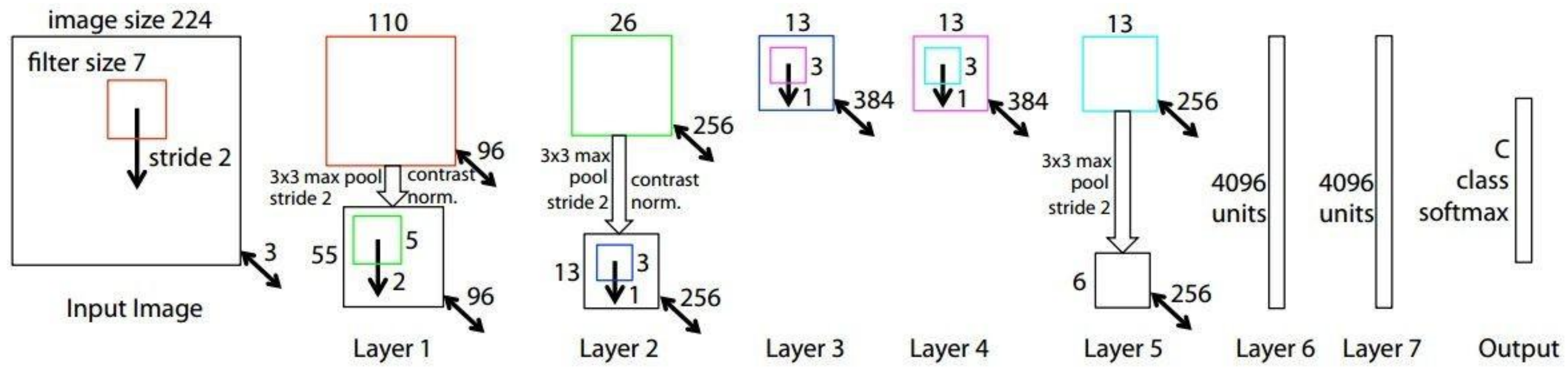
- Train on random 224x224 patches from the 256x256 images to get more data. Also use left-right reflections of the images.
    - At test time, combine the opinions from ten different patches: The four 224x224 corner patches plus the central 224x224 patch plus the reflections of those five patches.
- Use “dropout” to regularize the weights in the globally connected layers (which contain most of the parameters).
    - – half of the hidden units in a layer are randomly removed for each training example.

The hardware required for Alex’s net

- Uses a very efficient implementation of convolutional nets on two Nvidia GTX 580 Graphics Processor Units (over 1000 fast little cores)
  - GPUs are very good for matrix-matrix multiplies.
  - GPUs have very high bandwidth to memory.
  - This allows him to train the network in a week.
  - It also makes it quick to combine results from 10 patches at test time.
- We can spread a network over many cores if we can communicate the states fast enough.

Case Study: ZFNet

[Zeiler and Fergus, 2013]



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

Case Studies

- **GoogLeNet**. The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google.
- Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).
- Uses Average Pooling instead of Fully Connected layers at the top of the ConvNet
- There are also several followup versions to the GoogLeNet, most recently Inception-v4.
- **VGGNet**. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman.
- Showed that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers
- and, apfeatures an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. Their pretrained model is available for plug and play use in Caffe. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

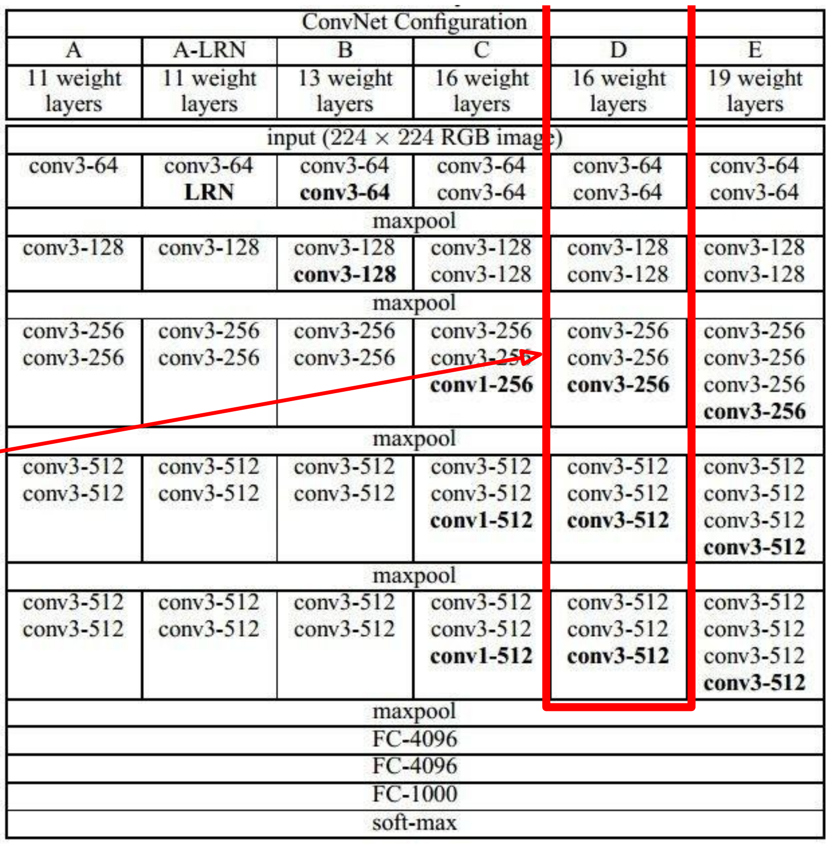
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

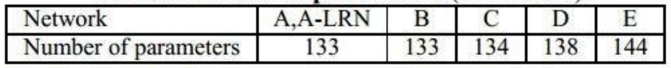
best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

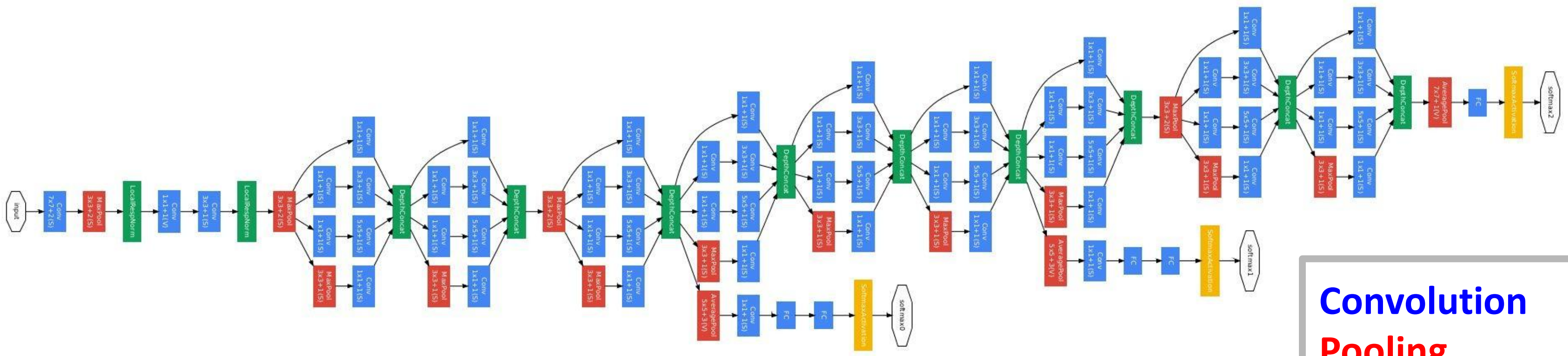
Table 2: **Number of parameters** (in millions).



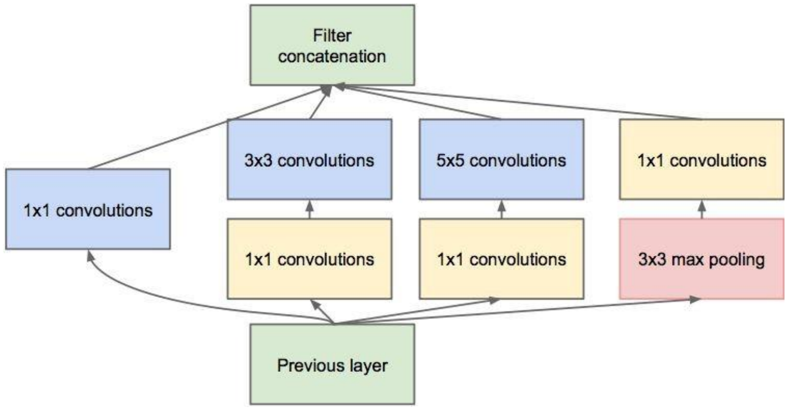
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Case Study: GoogLeNet

[Szegedy et al., 2014]



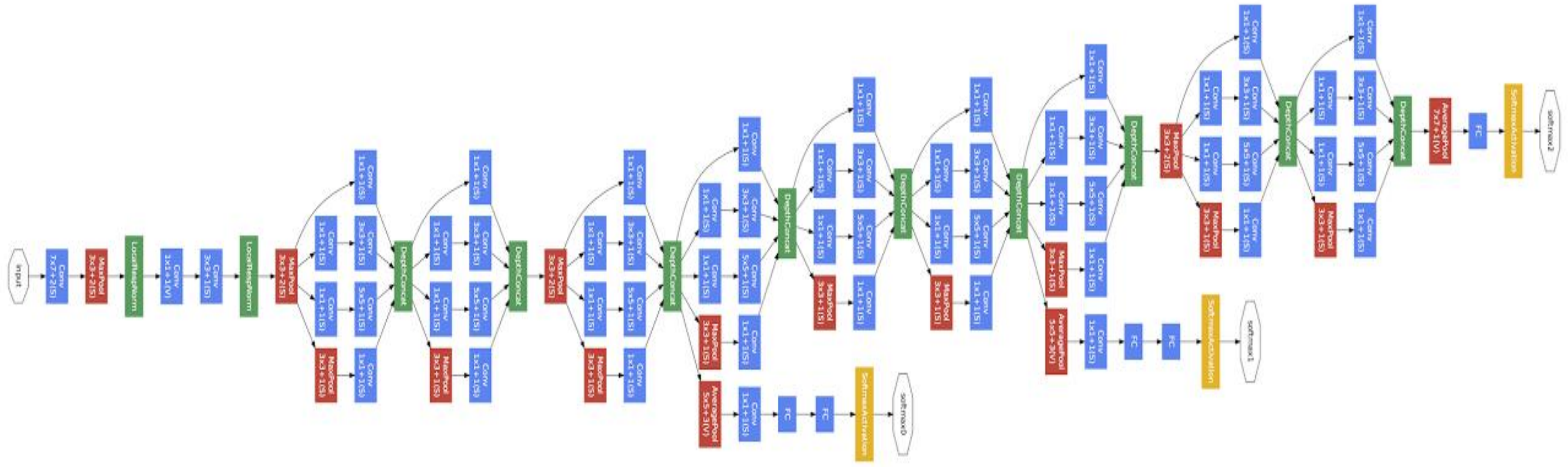
Convolution
Pooling
Softmax
Other



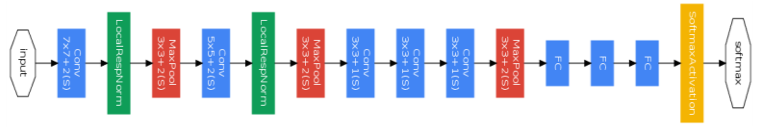
Inception module

ILSVRC 2014 winner (6.7% top 5 error)

GoogLeNet vs State of the art



GoogLeNet



Zeiler-Fergus Architecture (1 tower)

Convolution
Pooling
Softmax
Other

Residual Network

Deep Residual Learning for Image Recognition

: Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun

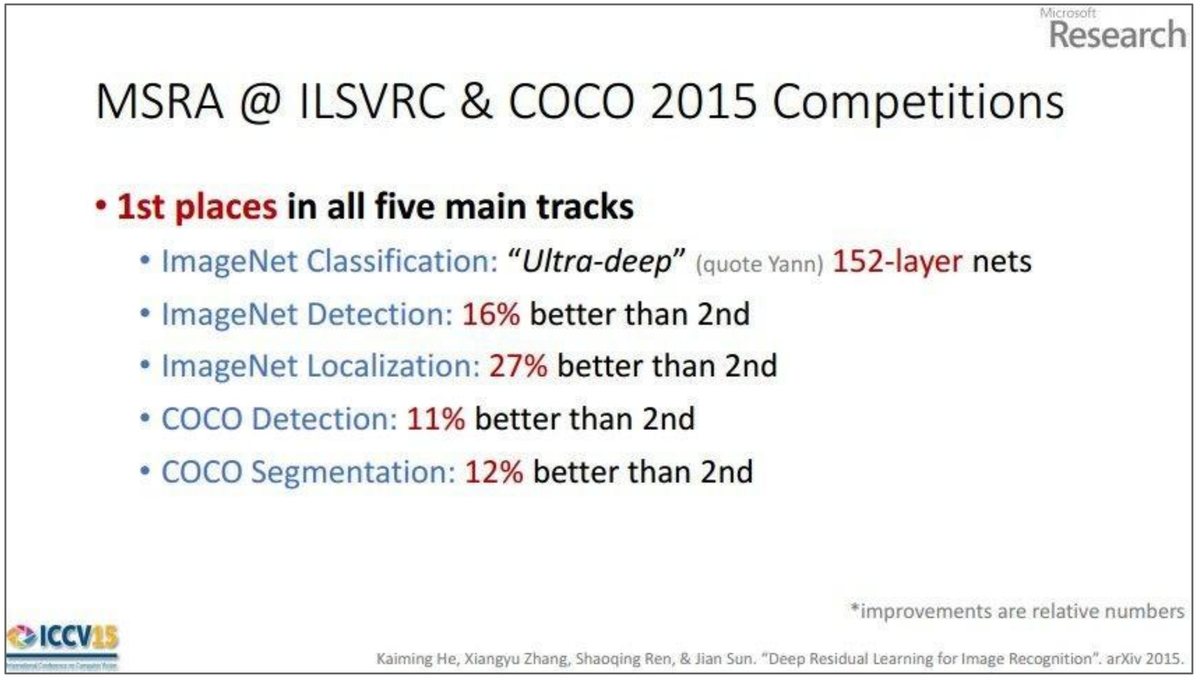
The deeper, the better

- The deeper network can cover more complex problems
  - Receptive field size ↑
  - Non-linearity ↑
- However, training the deeper network is more difficult because of vanishing/exploding gradients problem

Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



Microsoft Research

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
    - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
    - ImageNet Detection: 16% better than 2nd
    - ImageNet Localization: 27% better than 2nd
    - COCO Detection: 11% better than 2nd
    - COCO Segmentation: 12% better than 2nd

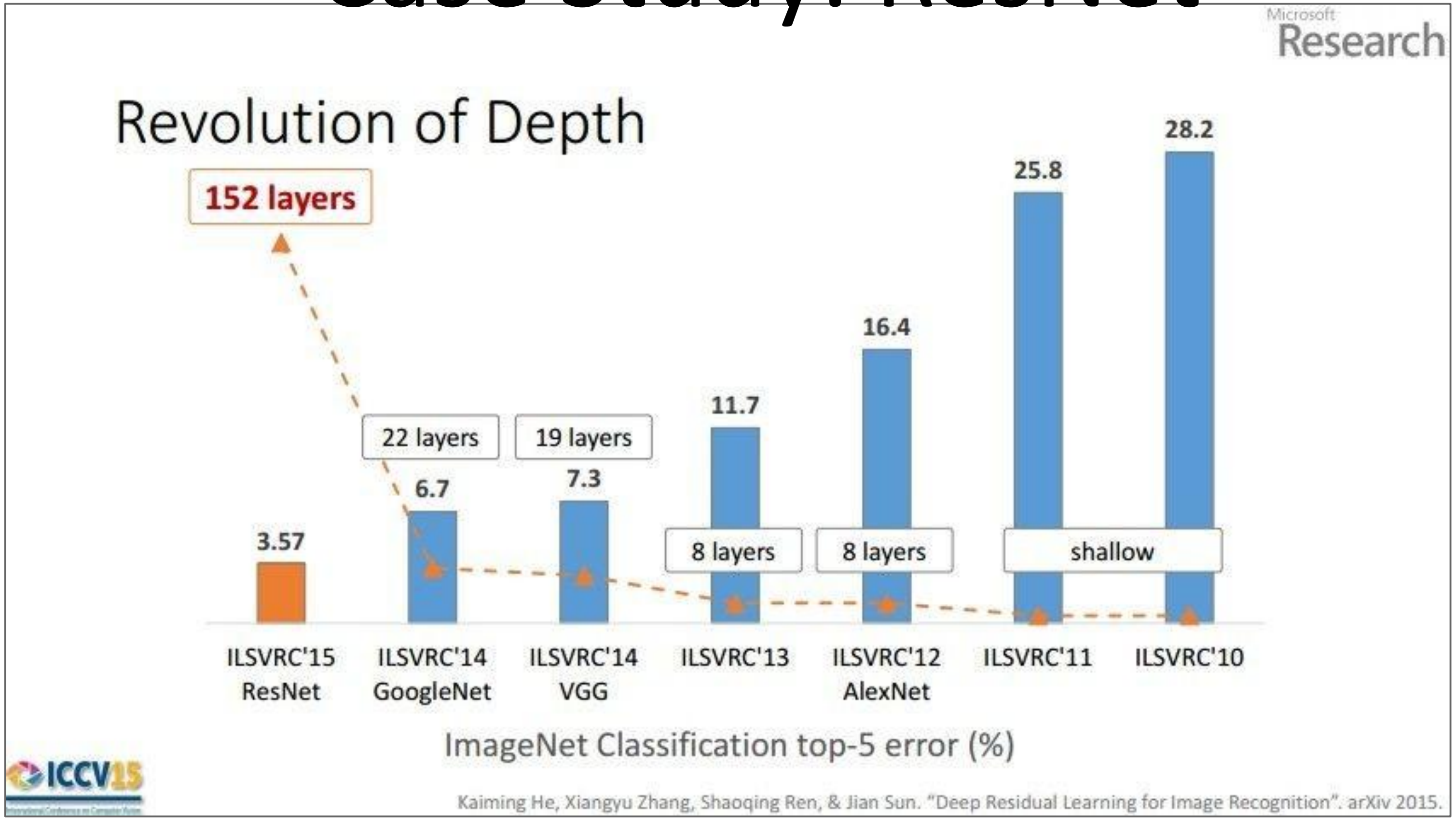
*improvements are relative numbers

ICCV15

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

Case Study: ResNet



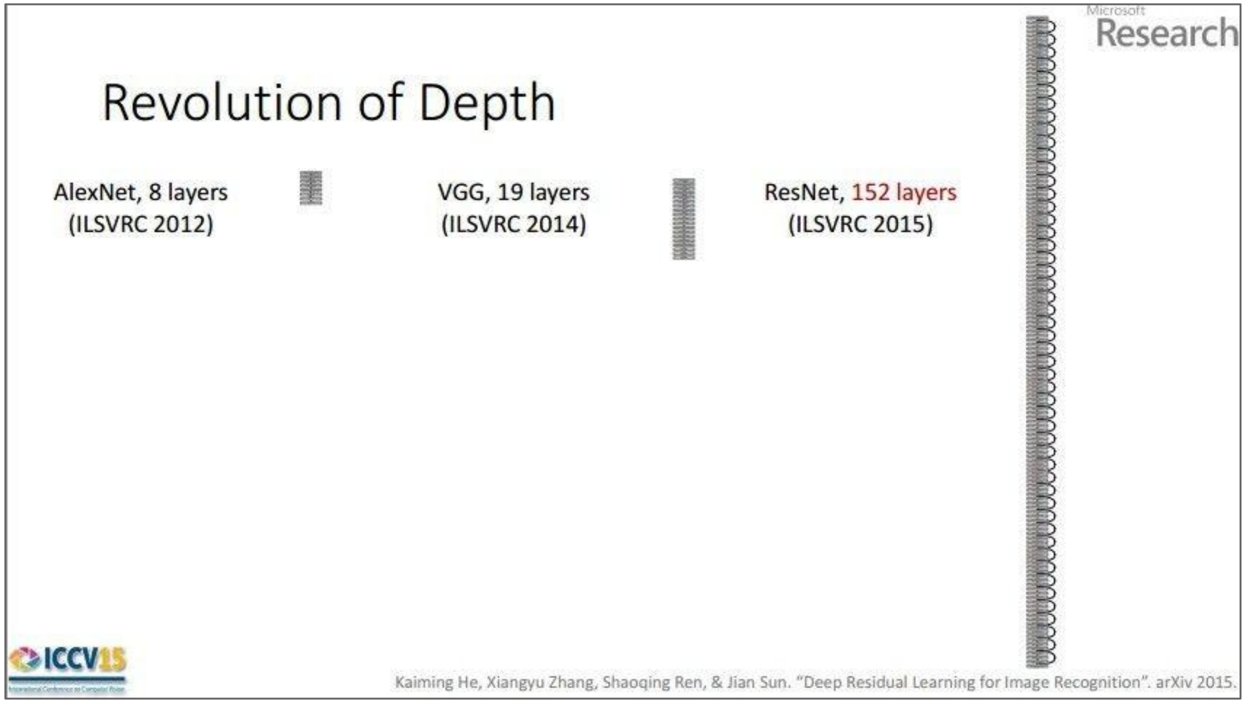
(slide from Kaiming He's recent presentation)

- Escape from few layers
  - ReLU for solving gradient vanishing problem
  - Dropout …
- Escape from 10 layers
  - Normalized initialization
  - Intermediate normalization layers
- Escape from 100 layers
  - Residual network

Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



Microsoft Research
Revolution of Depth
AlexNet, 8 layers (ILSVRC 2012)
VGG, 19 layers (ILSVRC 2014)
ResNet, 152 layers (ILSVRC 2015)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

(slide from Kaiming He's recent presentation)

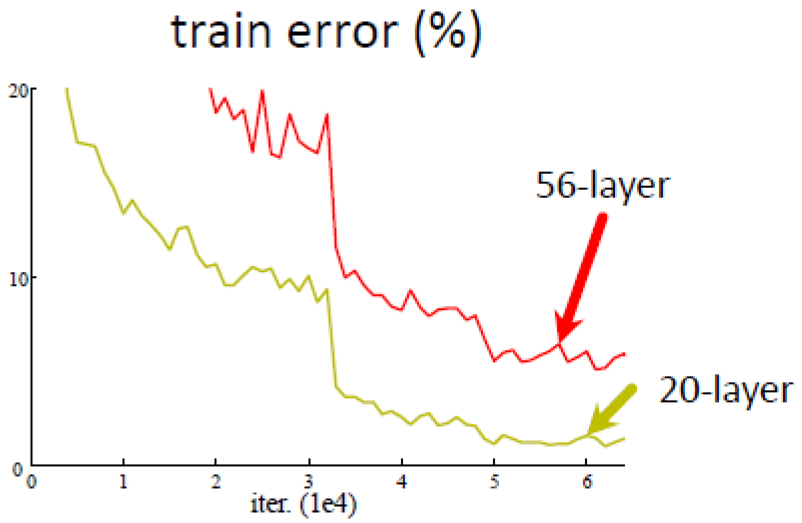
2-3 weeks of training on 8 GPU machine

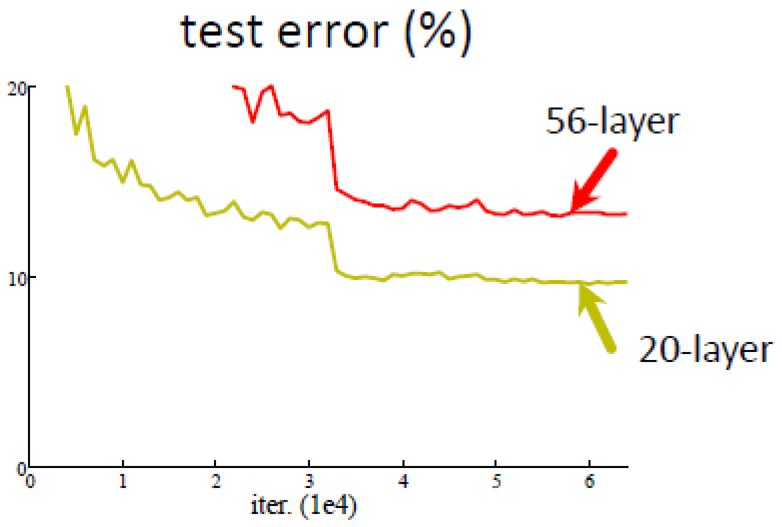
at runtime: faster than a VGGNet! (even though it has 8x more layers)

Plain Network

- Plain nets: stacking 3x3 conv layers
- 56-layer net has higher training error and test error than 20-layers net

CIFAR-10



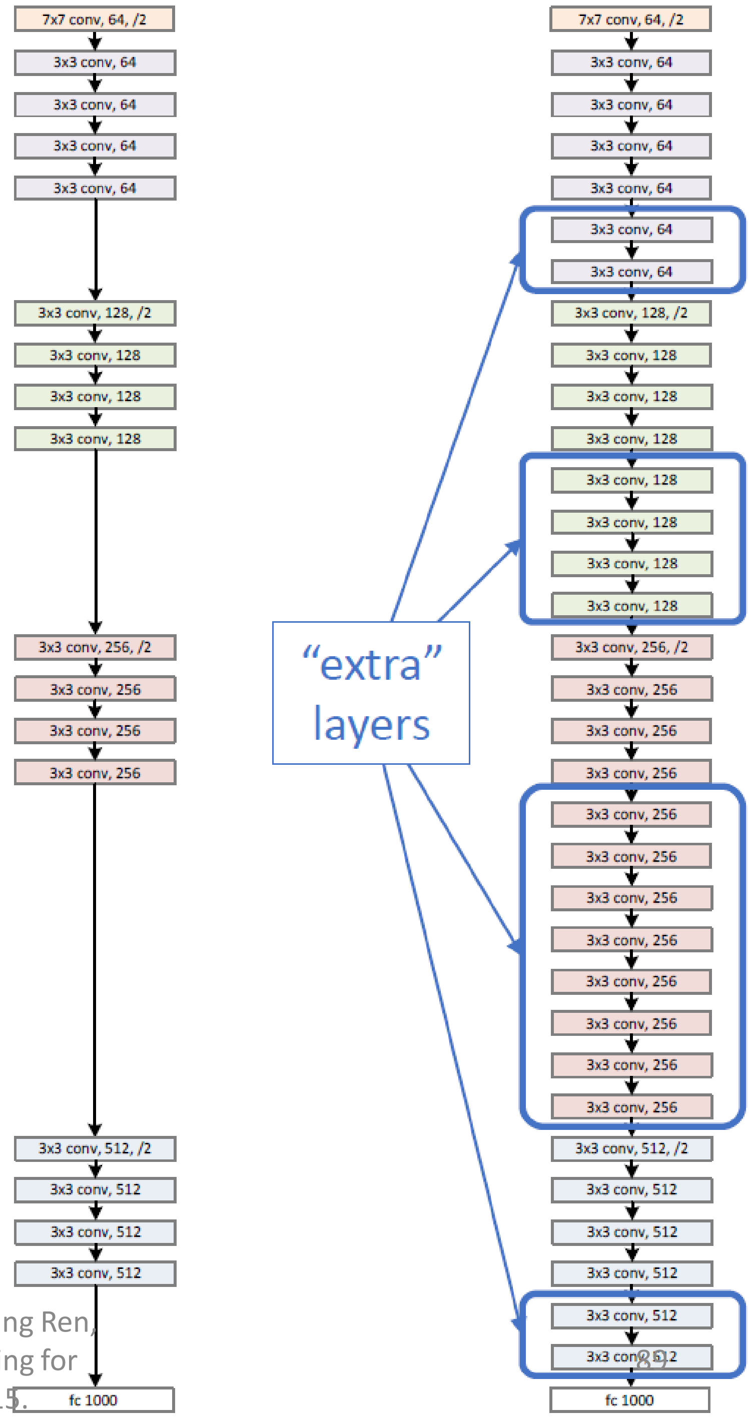


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

- **ResNet**. Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015. It features special *skip connections* and a heavy use of batch normalization.
- The architecture is also missing fully connected layers at the end of the network. The reader is also referred to Kaiming's presentation (video, slides), and some recent experiments that reproduce these networks in Torch.
- ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 10, 2016).
- More recent developments that tweak the original architecture from Kaiming He et al. Identity Mappings in Deep Residual Networks (published March 2016).

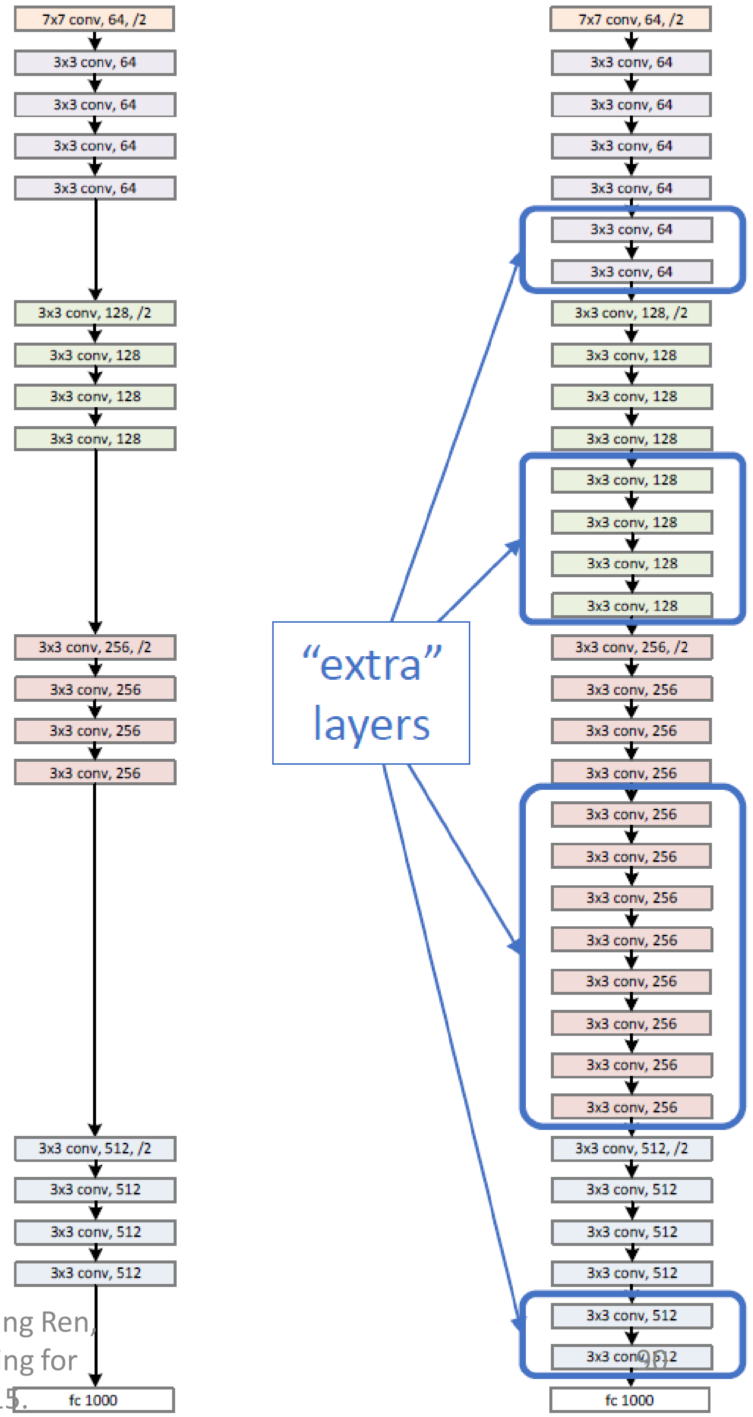
Residual Network

- Naïve solution
  - If extra layers are an identity mapping, then a training errors does not increase



Residual Network

- If extra layers are an identity mapping, then training error does not increase
- Adding layers makes smaller differences
- Optimal mappings are closer to an identity

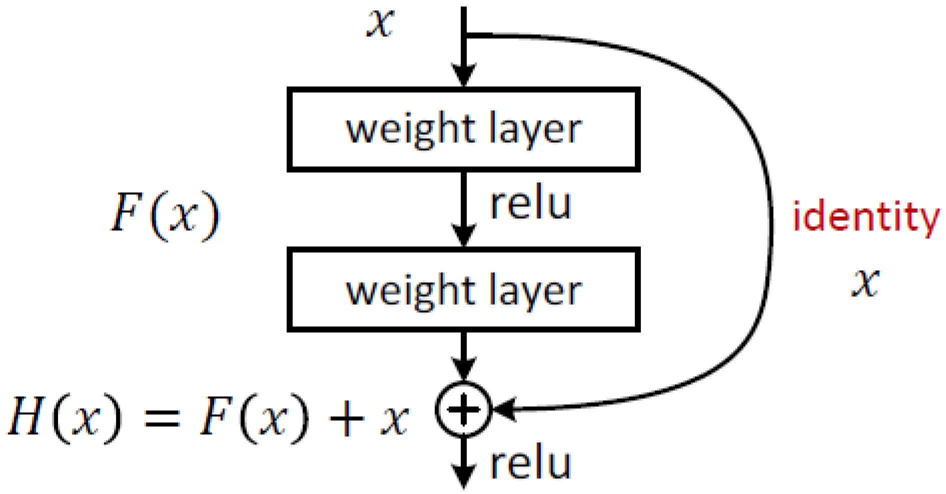


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Residual Network

Residual block

- In traditional CNNs, H(x) would just be equal to F(x)
- Instead, we're computing the term F(x), that you have to add to input x.
- Basically, the mini module is computing a "delta" or a slight change to the original input x to get a slightly altered representation

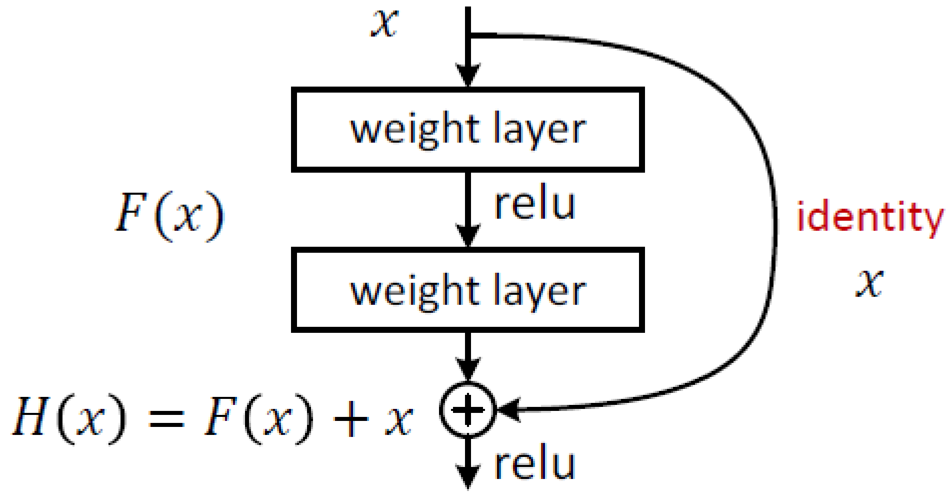


The authors believe that "it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping".

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

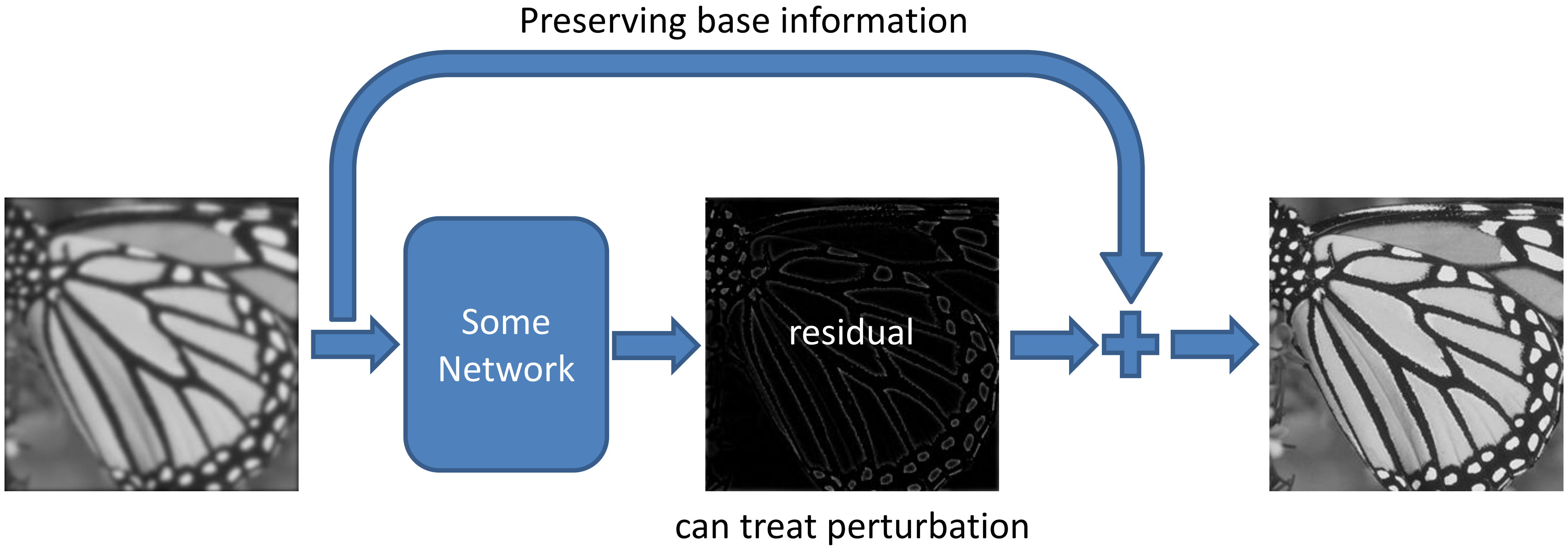
Residual Network

F(x) is a residual mapping w.r.t. identity



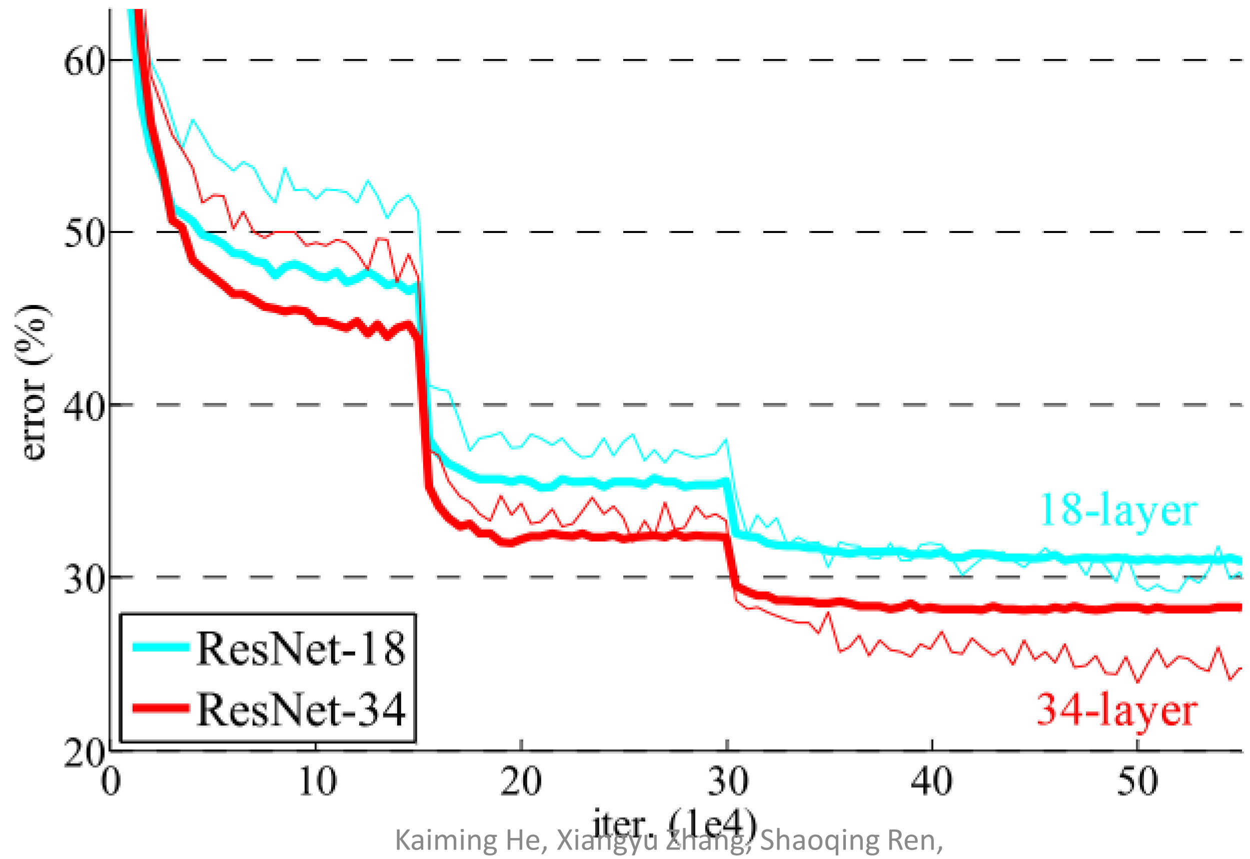
Residual Network

- Difference between an original image and a changed image



Residual Network

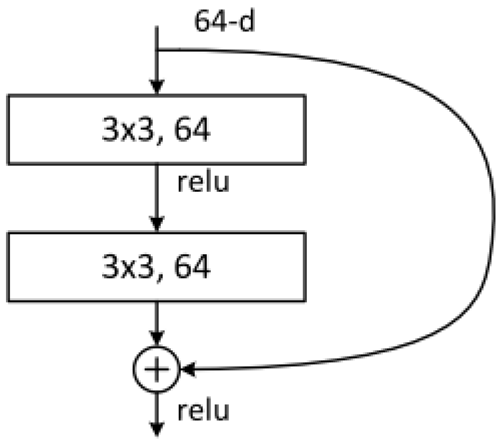
- Deeper ResNets have lower training error



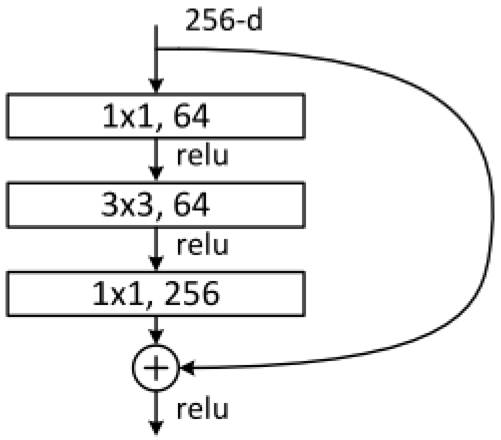
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

Residual Network

- Residual block
  - Very simple
  - Parameter-free



A naïve residual block

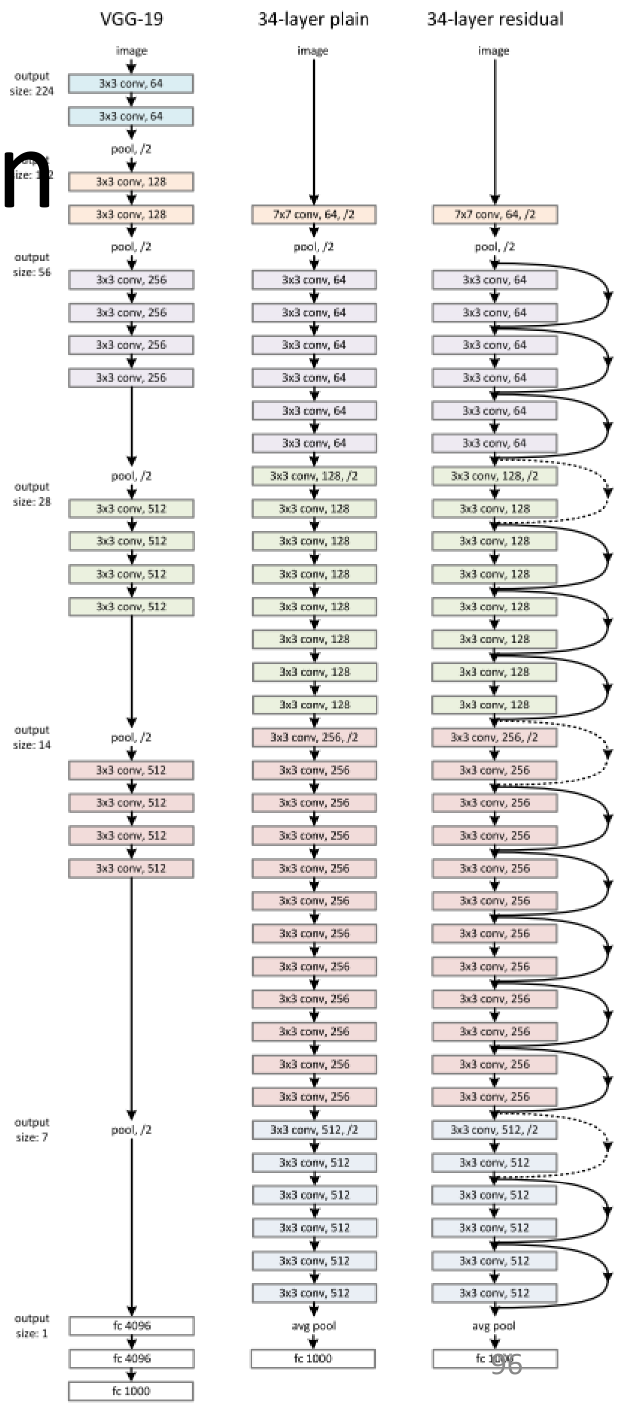


"bottleneck" residual block
(for ResNet-50/101/152)

Network Design

- Basic design (VGG-style)
  - All 3x3 conv (almost)
  - Spatial size/2 => #filters x2
  - Batch normalization
  - Simple design, just deep

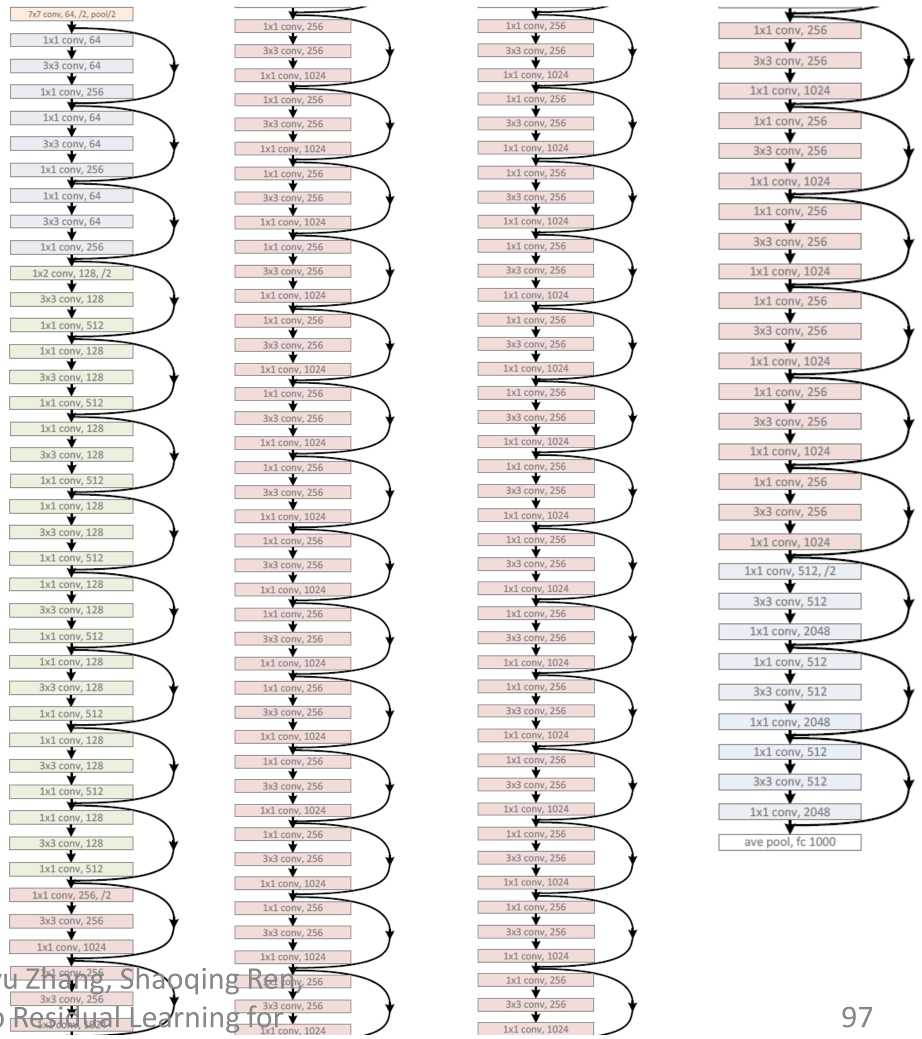
- Other remarks
  - No max pooling (almost)
  - No hidden fc
  - No dropout



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

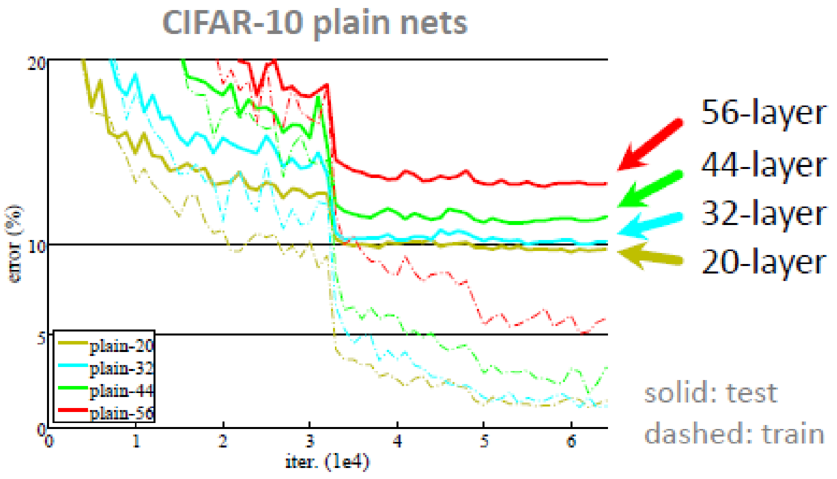
Network Design

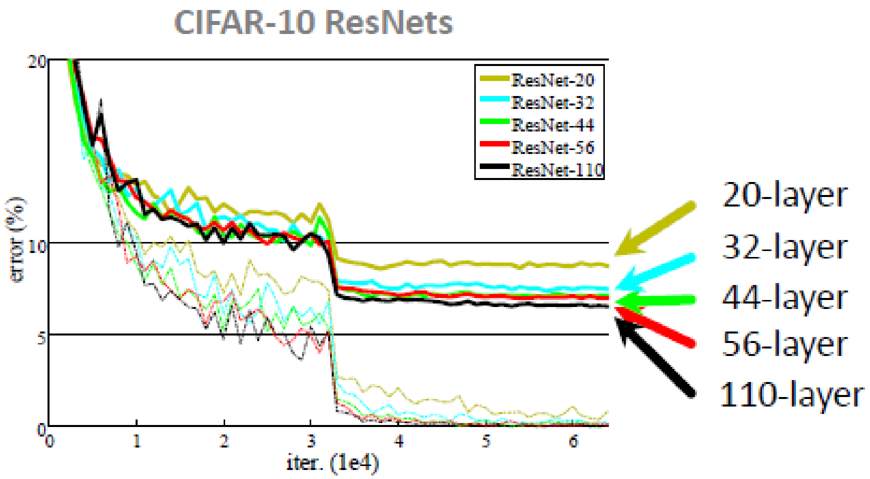
- ResNet-152
  - Use bottlenecks
  - ResNet-152(11.3 billion FLOPs) has lower complexity than VGG-16/19 nets (15.3/19.6 billion FLOPs)



Results

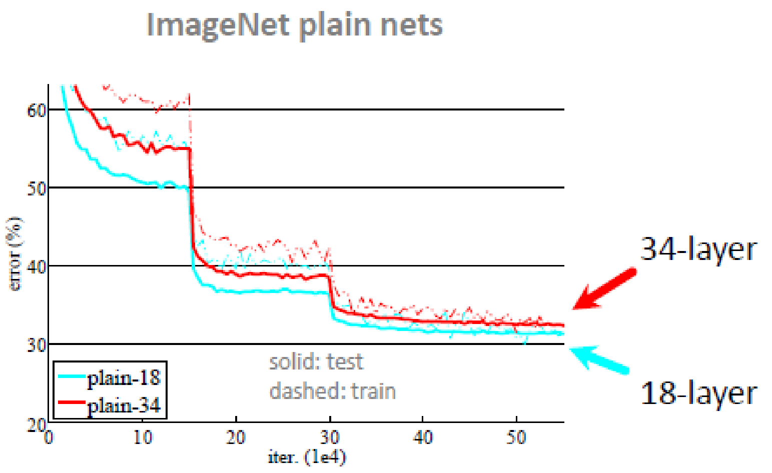
- Deep Resnets can be trained without difficulties
- Deeper ResNets have lower training error, and also lower test error

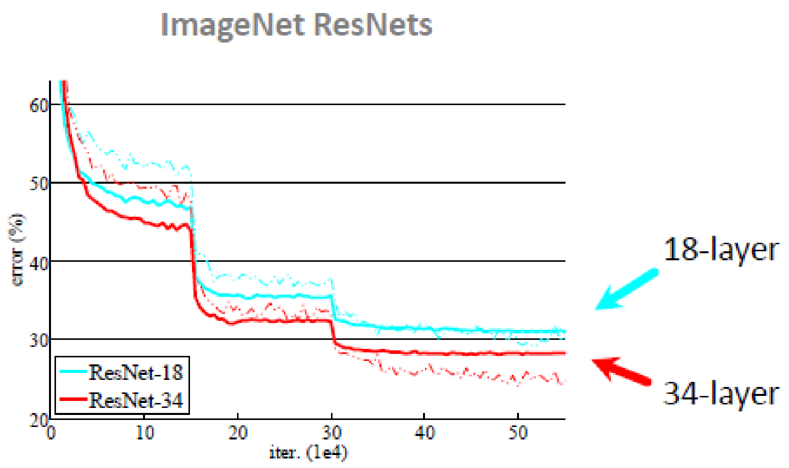




Results

- Deep Resnets can be trained without difficulties
- Deeper ResNets have lower training error, and also lower test error



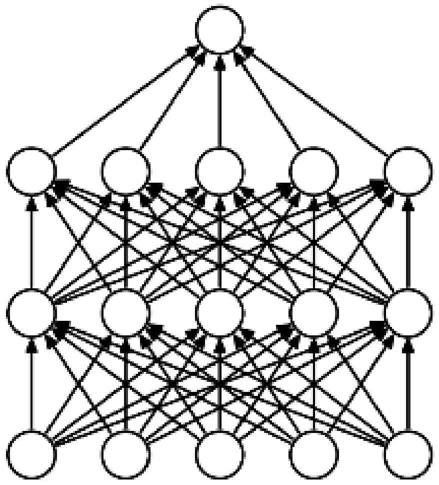


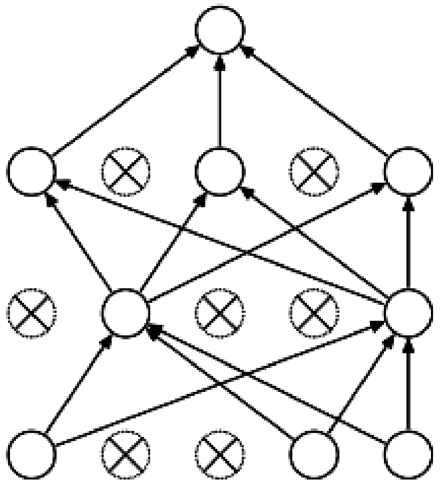
Practical matters

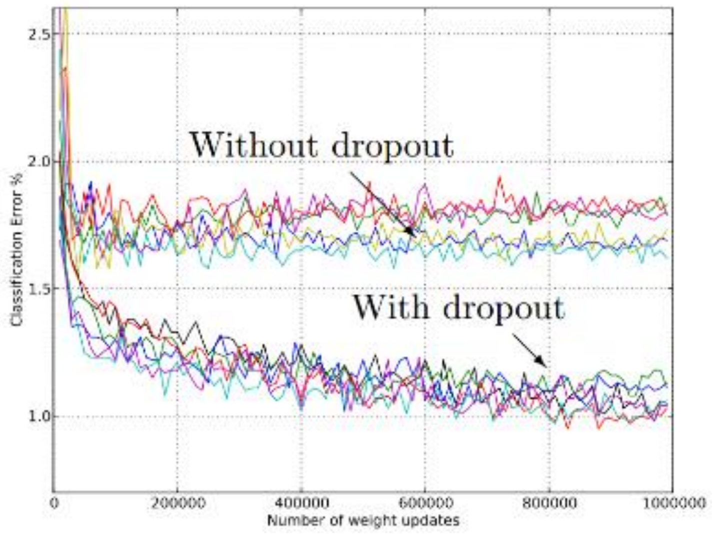
Training: Best practices

- Use mini-batch
- Use regularization
- Use gradient checks
- Use cross-validation for your parameters
- Use RELU or leaky RELU or ELU, don’t use sigmoid
- Center (subtract mean from) your data
- To initialize, use “Xavier initialization”
    - initializing the weights in your network by drawing them from a distribution with zero mean and variance $Var(W) = {}^{1}/_{n_{in}}$
- Learning rate: too high? Too low?

Regularization: Dropout


(a) Standard Neural Net


(b) After applying dropout.



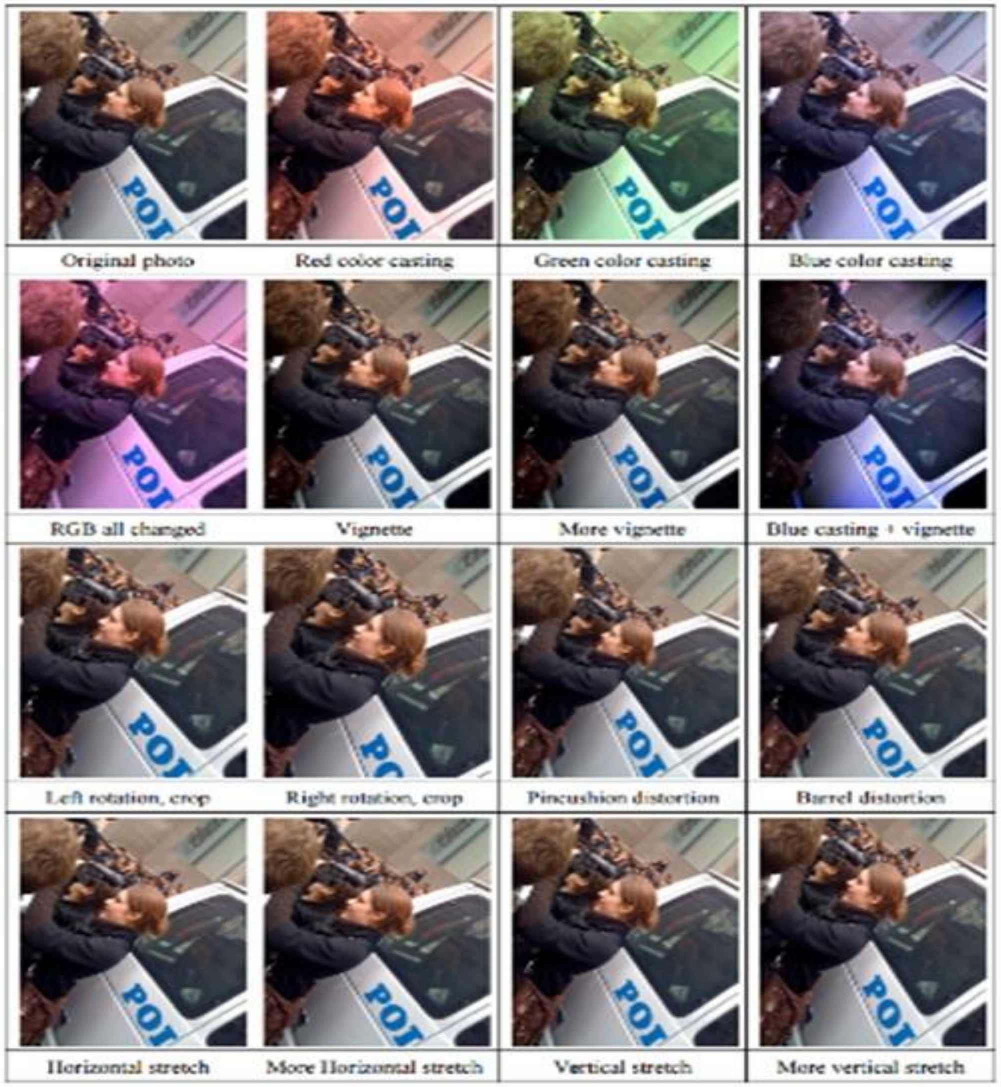
- Randomly turn off some neurons
- Allows individual neurons to independently be responsible for performance

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Adapted from Jia-bin Huang

Data Augmentation (Jittering)

- Create *virtual* trainin
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



Deep Image [Wu et al. 2015]

Transfer Learning

"You need a lot of a data if you want to train/use CNNs"

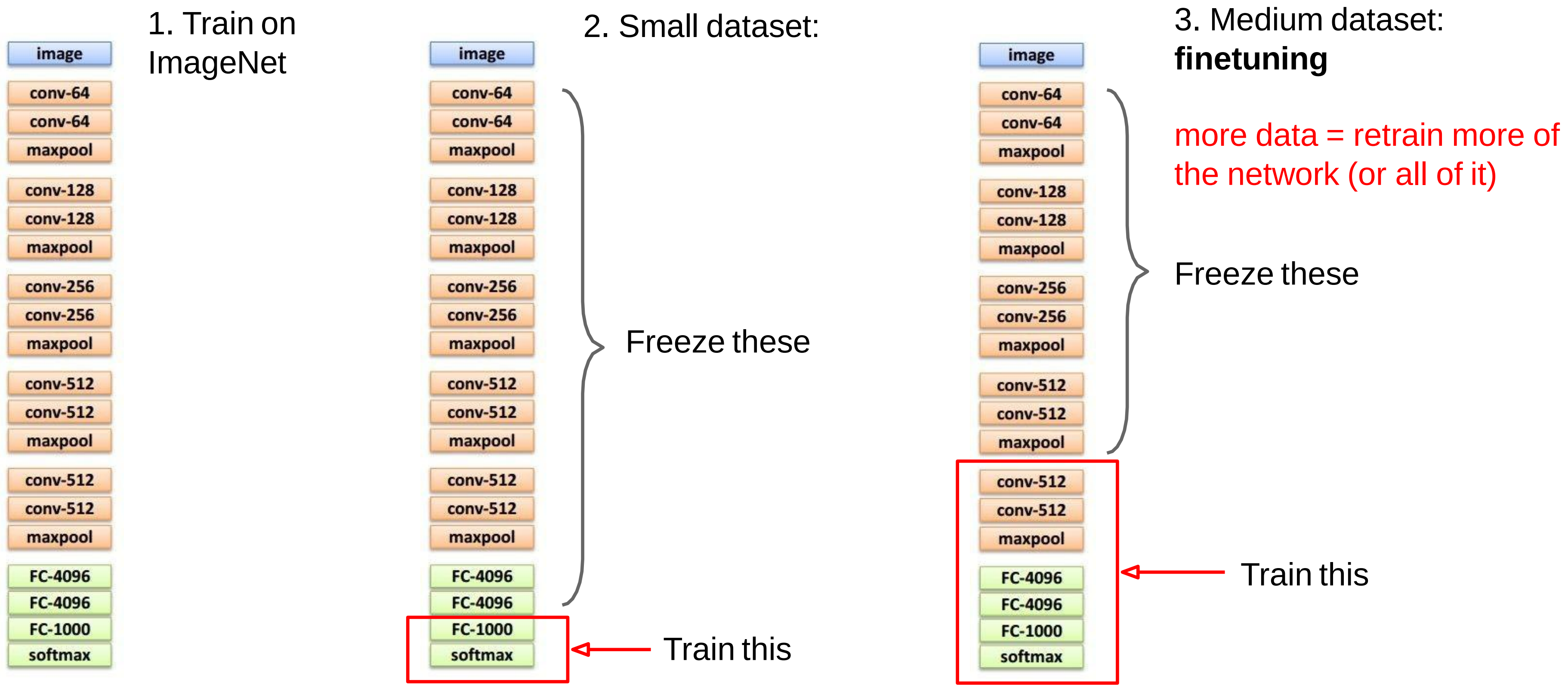
BUSTED

Andrej Karpathy

Transfer Learning with CNNs

Source: classification on ImageNet

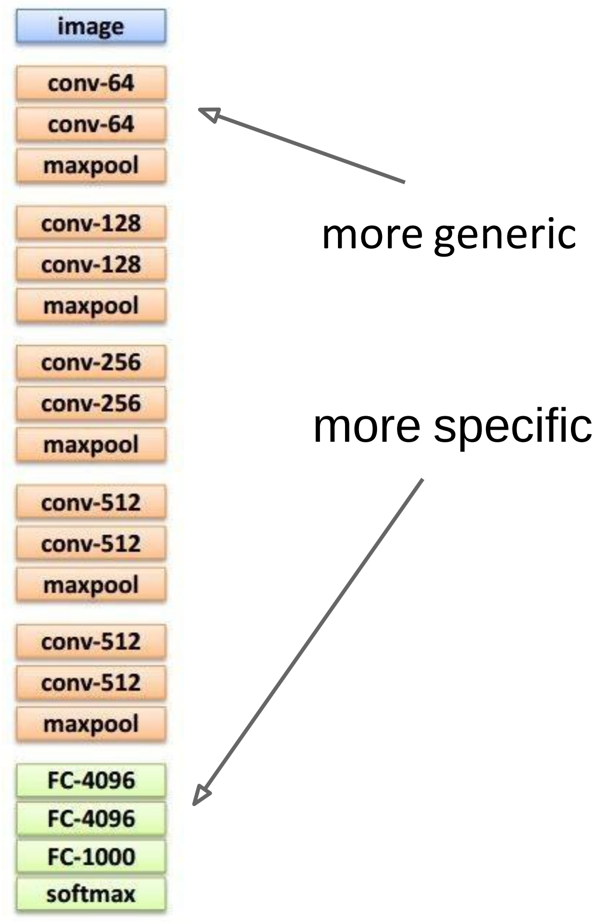
Target: some other task/data



Another option: use network as feature extractor, train SVM on extracted features for target task

Adapted from Andrej Karpathy

Transfer Learning with CNNs



	very similar dataset	**very different dataset**
very little data	Use linear classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Andrej Karpathy

Simplest Way to Use CNNs

- Take model trained on, e.g., ImageNet 2012 training set
- Easiest: Take outputs of e.g. 6th or 7th fully-connected layer, and plug features from each layer into linear SVM
    - Features are neuron activations at that level
    - Can train linear SVM for different tasks, not just one used to learn the deep net
- Better: fine-tune features and/or classifier on new dataset
- Classify test set of new dataset

Adapted from Lana Lazebnik

Packages

- Caffe and Caffe Model Zoo
- Torch
- Theano with Keras/Lasagne
- MatConvNet
- TensorFlow

Learning Resources

- http://deeplearning.net/
- http://cs231n.stanford.edu

Things to remember

- Overview
  - Neuroscience, perceptron, multi-layer neural networks
- Convolutional neural network (CNN)
  - Convolution, nonlinearity, max pooling
- Training CNN
  - Dropout; data augmentation; transfer learning
- Using CNNs for your own task
  - Basic first step: try the pre-trained CaffeNet fc6-fc8 layers as features