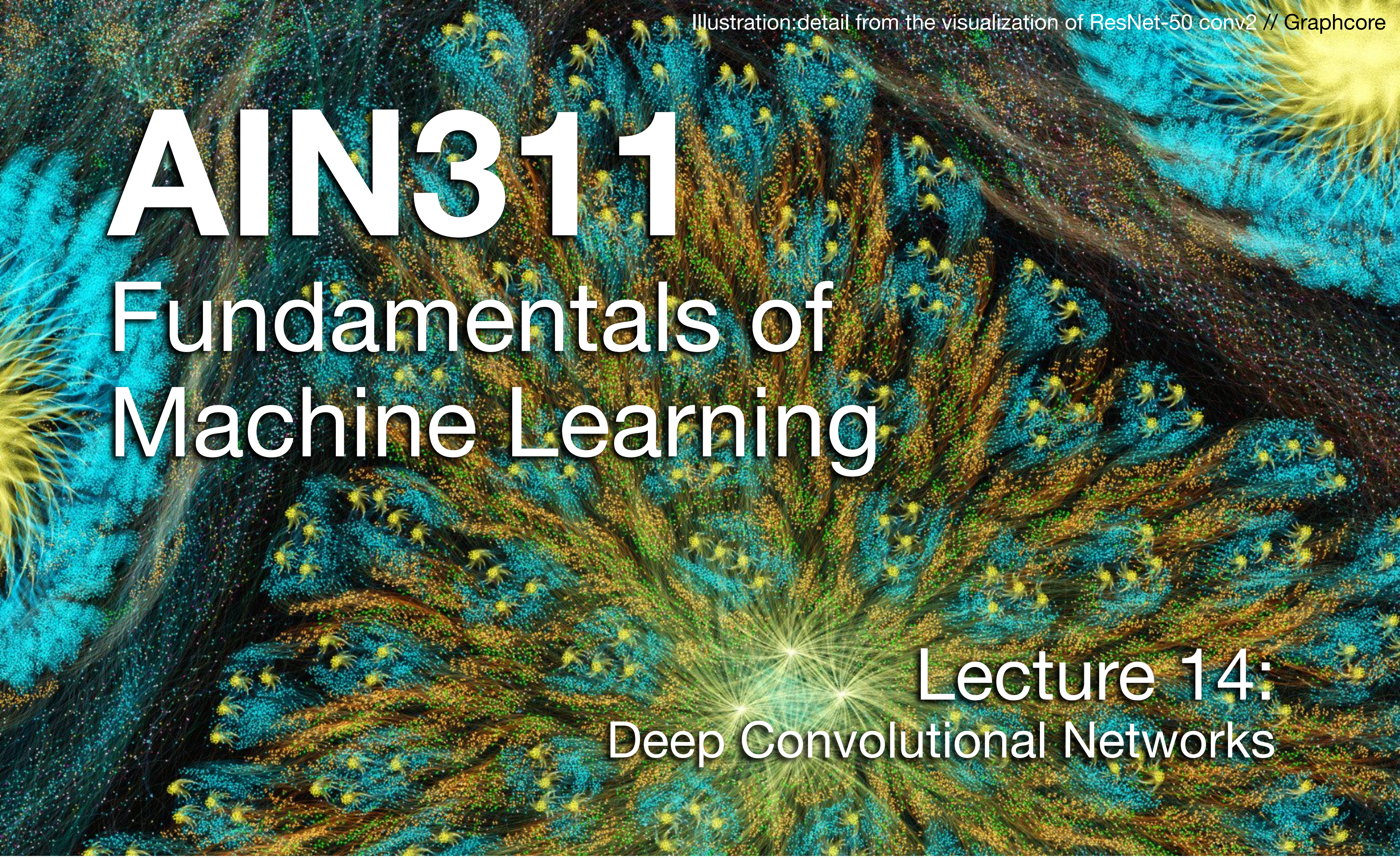


Illustration:detail from the visualization of ResNet-50 conv2 // Graphcore

AIN311

Fundamentals of Machine Learning

Lecture 14:

Deep Convolutional Networks

HACETTEPE UNIVERSITY COMPUTER VISION LAB

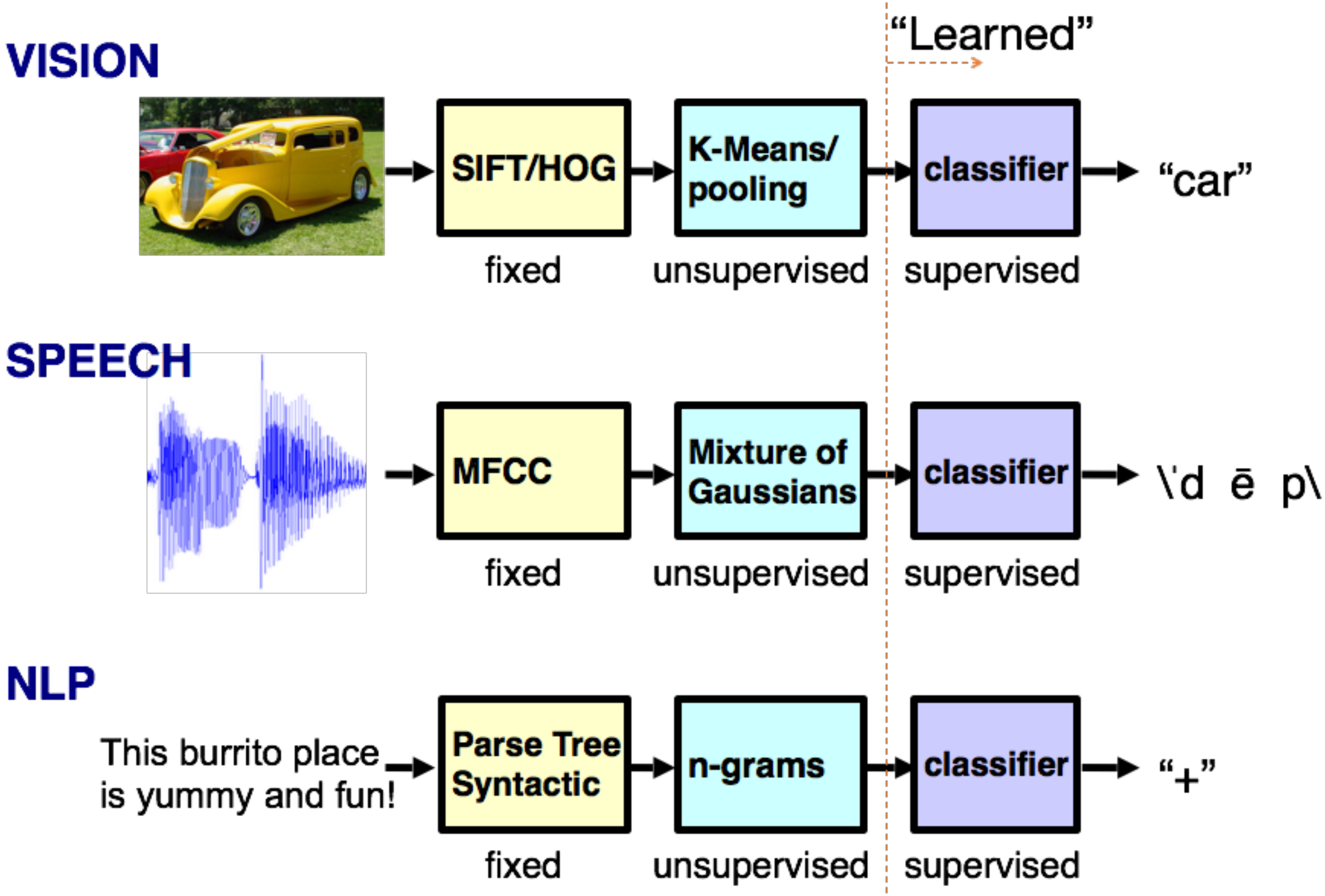
Erkut Erdem // Hacettepe University // Fall 2023

Last time... **Three key ideas**

- (Hierarchical) Compositionality
  - Cascade of non-linear transformations
  - Multiple layers of representations
- End-to-End Learning
  - Learning (goal-driven) representations
  - Learning to feature extract
- Distributed Representations
  - No single neuron “encodes” everything
  - Groups of neurons work together

slide by Dhruv Batra

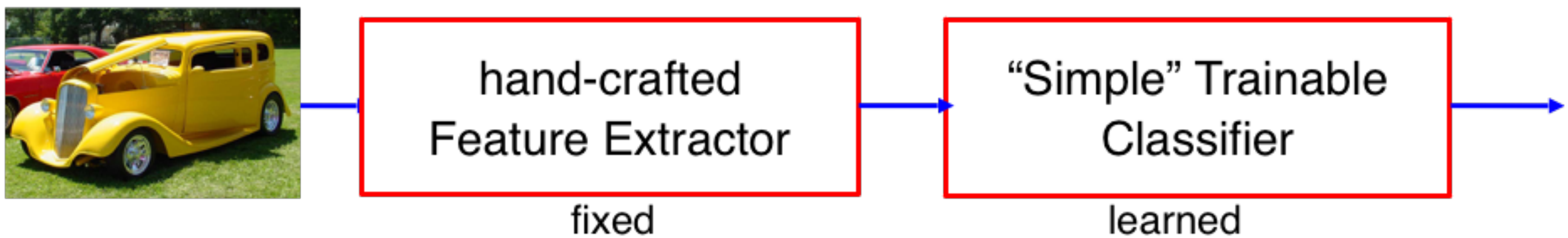
Last time… **Intro. to Deep Learning**



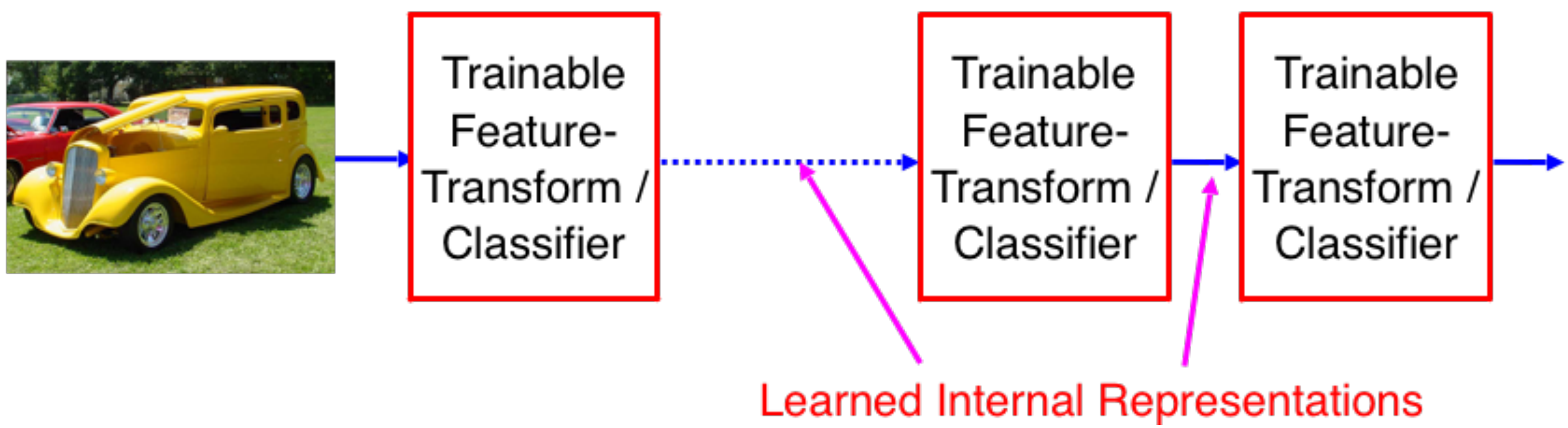
slide by Marc'Aurelio Ranzato, Yann LeCun

Last time… **Intro. to Deep Learning**

- “Shallow” models



- Deep models



slide by Marc'Aurelio Ranzato, Yann LeCun

Deep Convolutional Neural Networks

Convolutions

- Images typically have invariant patterns
  - E.g., directional gradients are translational invariant:

- Apply convolution to local sliding windows

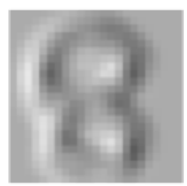
slide by Yisong Yue

Convolution Filters

- Applies to an image patch x
  - Converts local window into single value
  - Slide across image

$$\underset{\text{Local Image Patch}}{x} \otimes W = \sum_{ij} W_{ij} x_{ij}$$

$\otimes W =$



Left-to-Right
Edge Detector

-1	0	+1
-1	0	+1
-1	0	+1

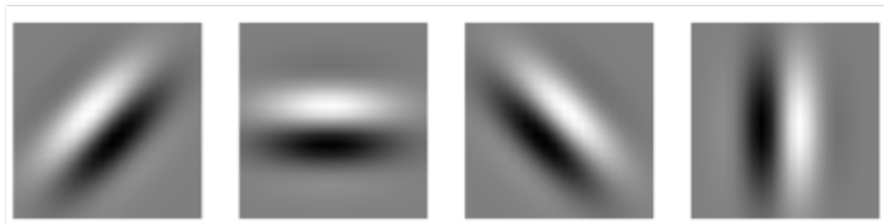
W

slide by Yisong Yue

Gabor Filters

- Most common low-level convolutions for computer vision

Example W:



-1	0	+1
-1	0	+1
-1	0	+1

W

http://en.wikipedia.org/wiki/Gabor_filter

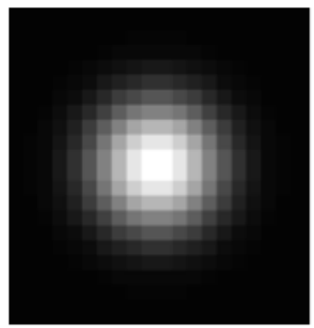
slide by Yisong Yue

Gaussian Blur Filters

- Weights decay according to Gaussian Distribution
  - Variance term controls radius

Example W:

Apply per RGB Channel



- Black = 0
- White = Positive

http://en.wikipedia.org/wiki/Gaussian_blur

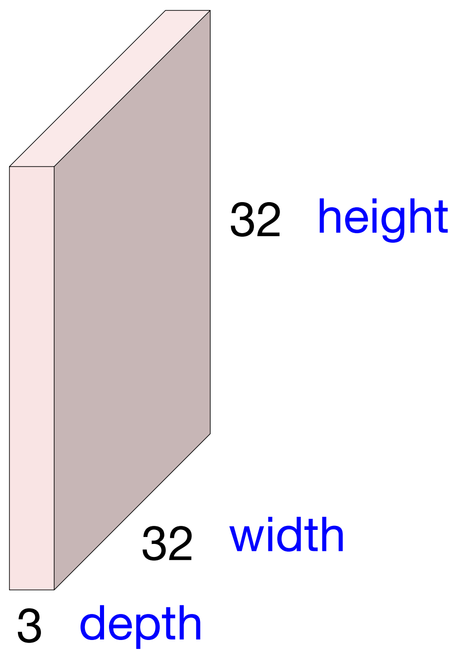
slide by Yisong Yue

Convolutional Neural Networks

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Convolution Layer

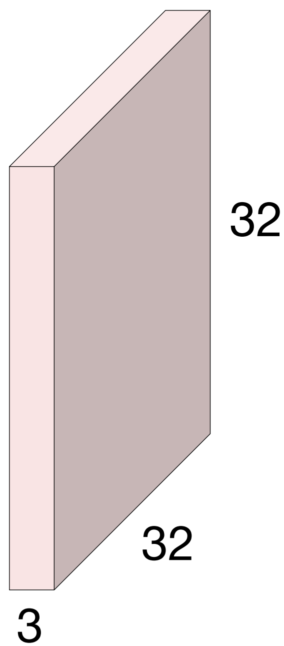
32x32x3 image



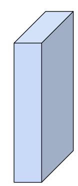
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Convolution Layer

32x32x3 image



5x5x3 filter



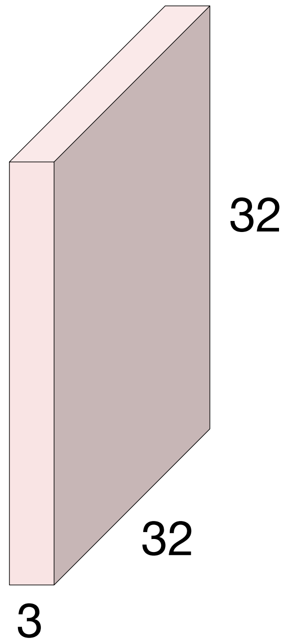
Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

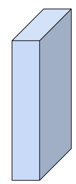
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter



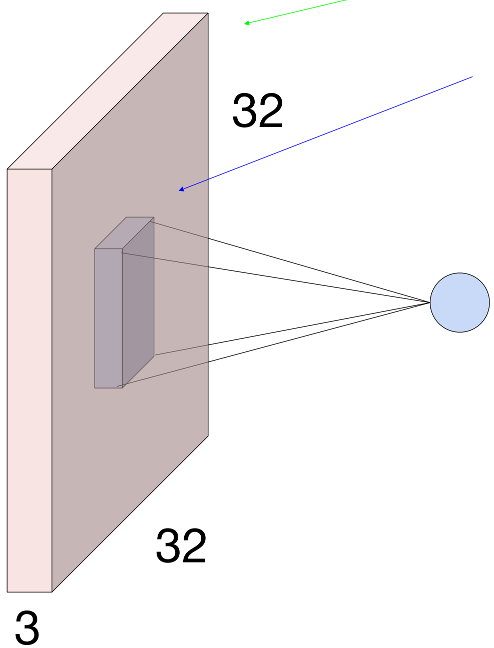
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Convolution Layer

32x32x3 image

5x5x3 filter w



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)

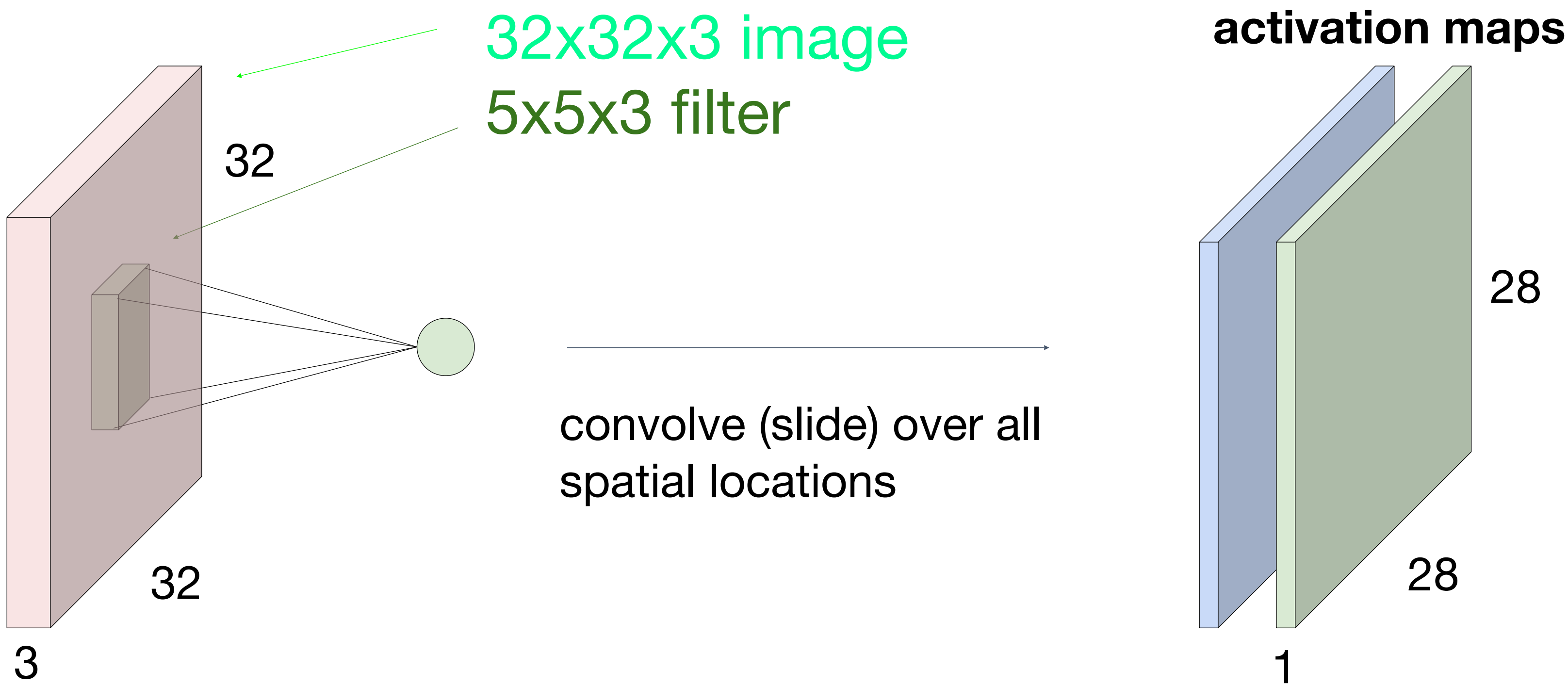
$$w^T x + b$$

Convolution Layer

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

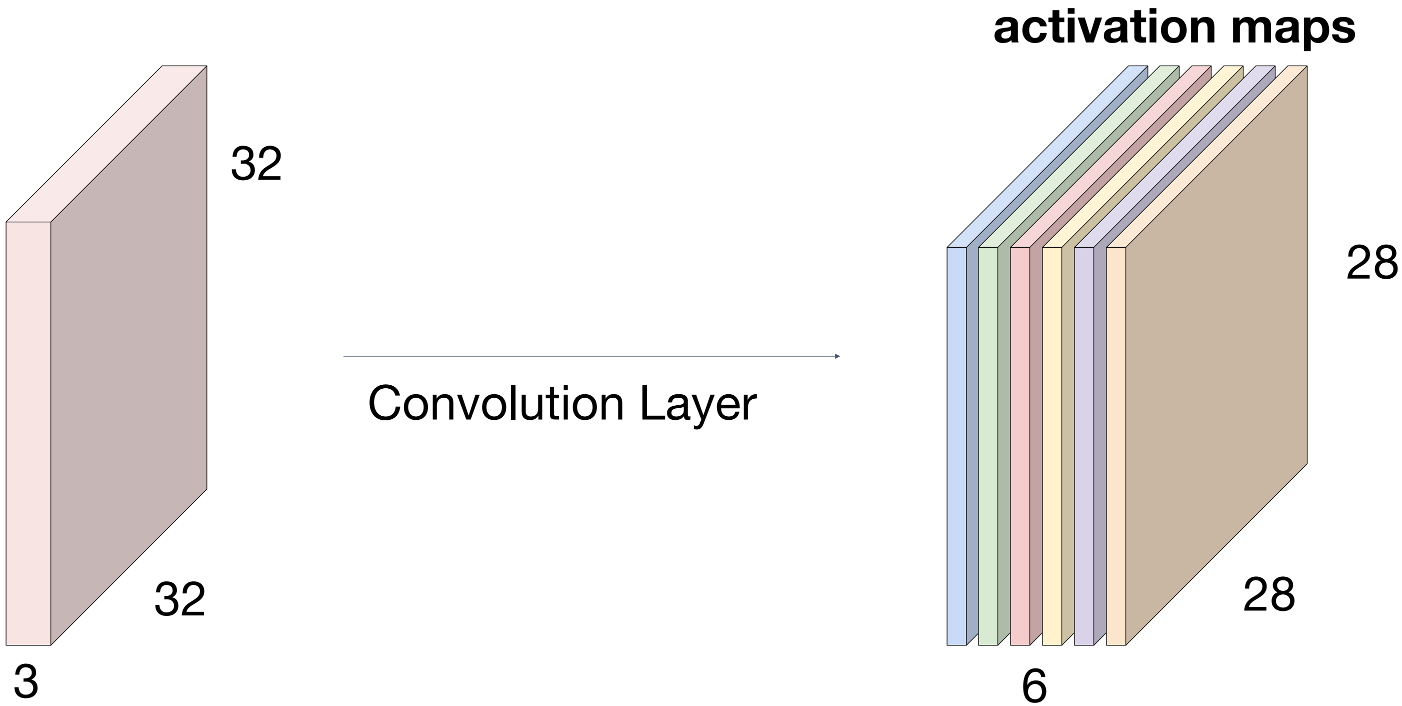
Convolution Layer

consider a second, green filter



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

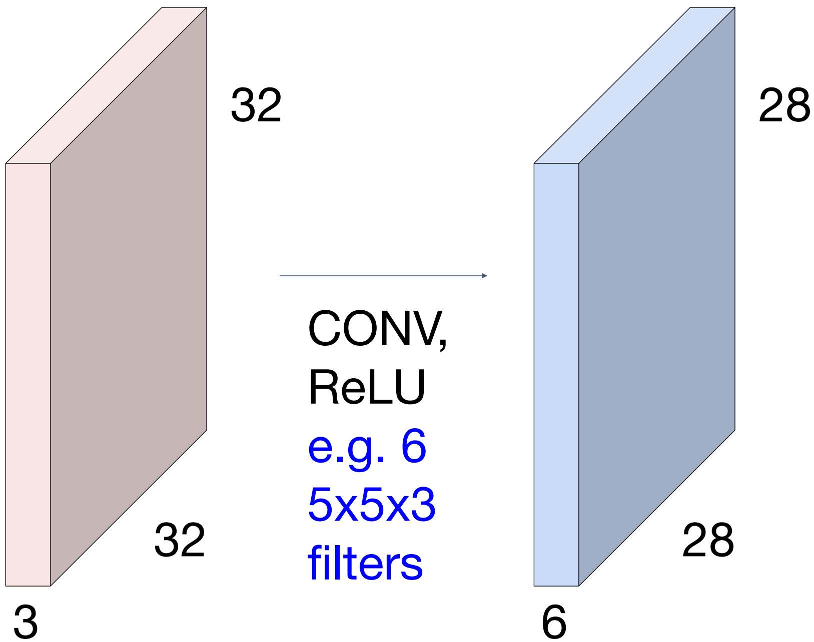
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

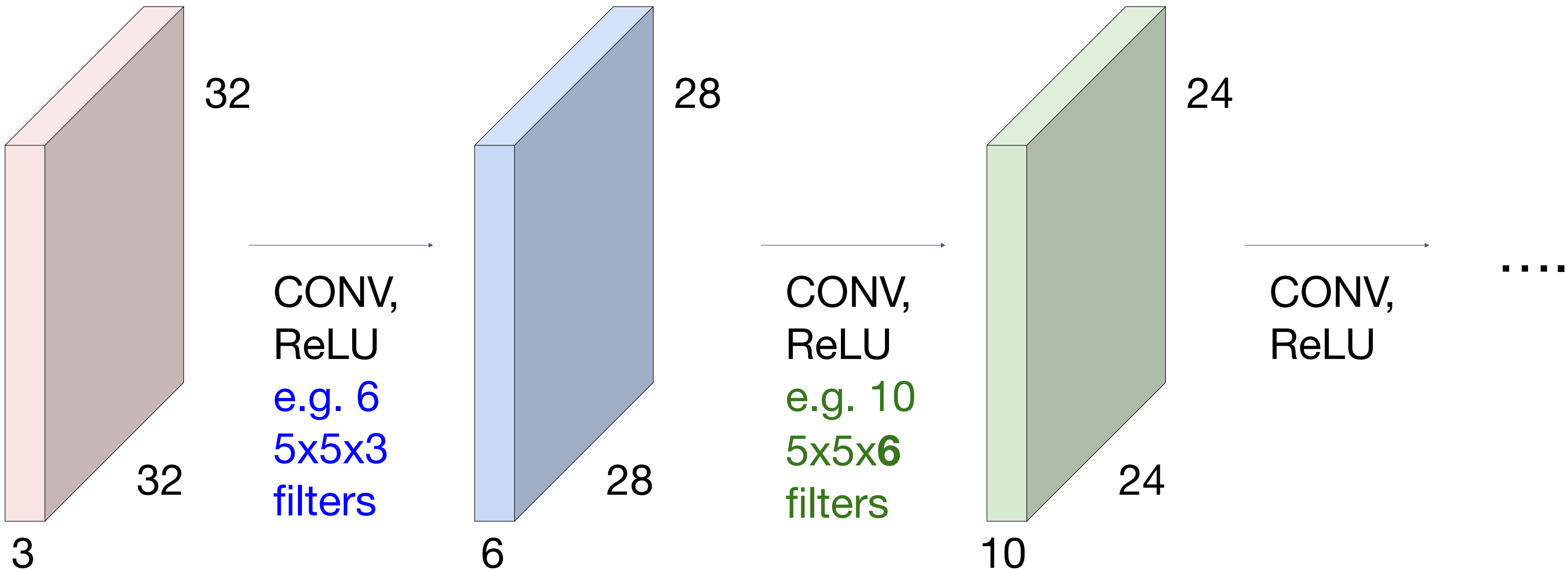
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

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



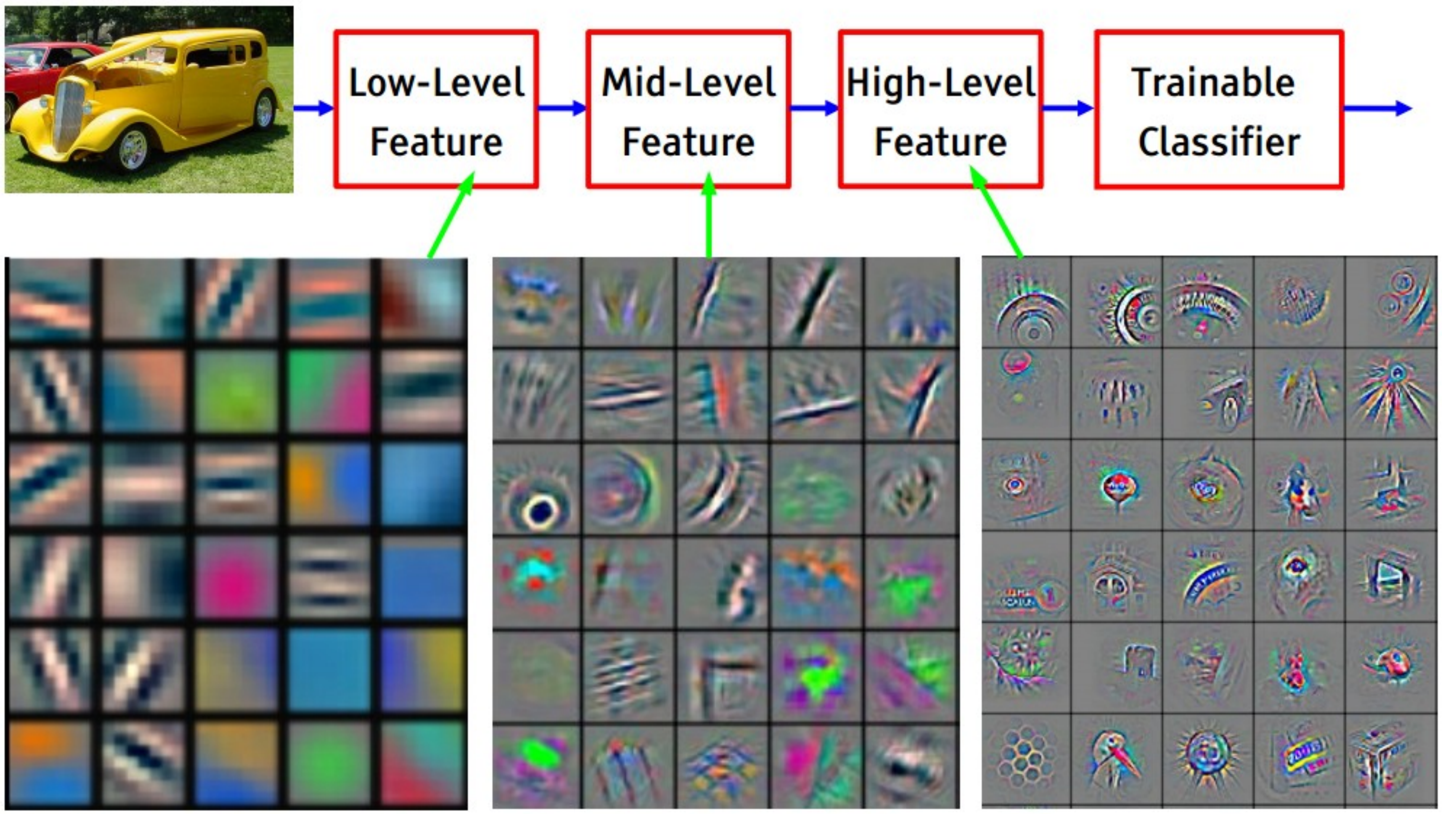
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

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



Preview

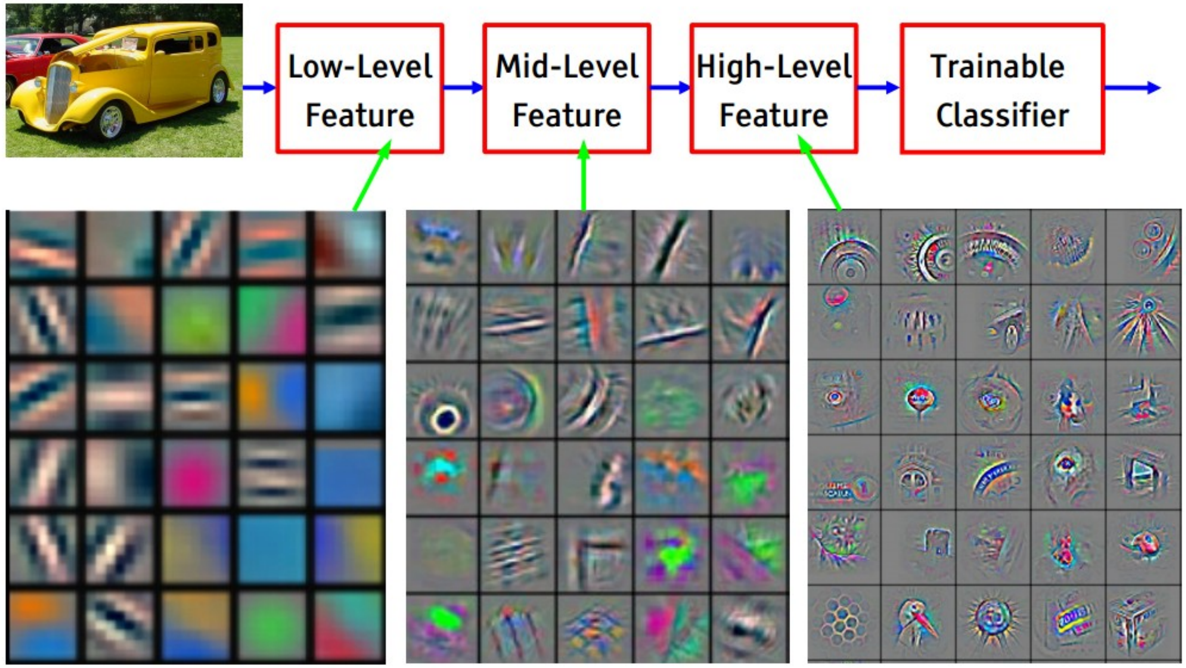
[From recent Yann LeCun slides]



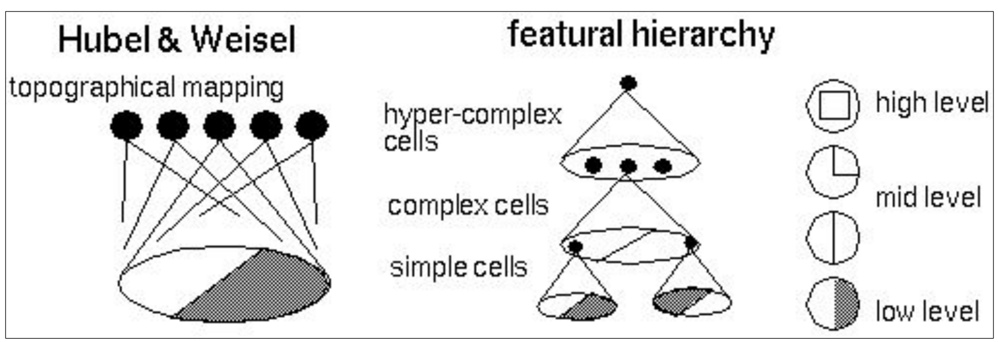
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Preview

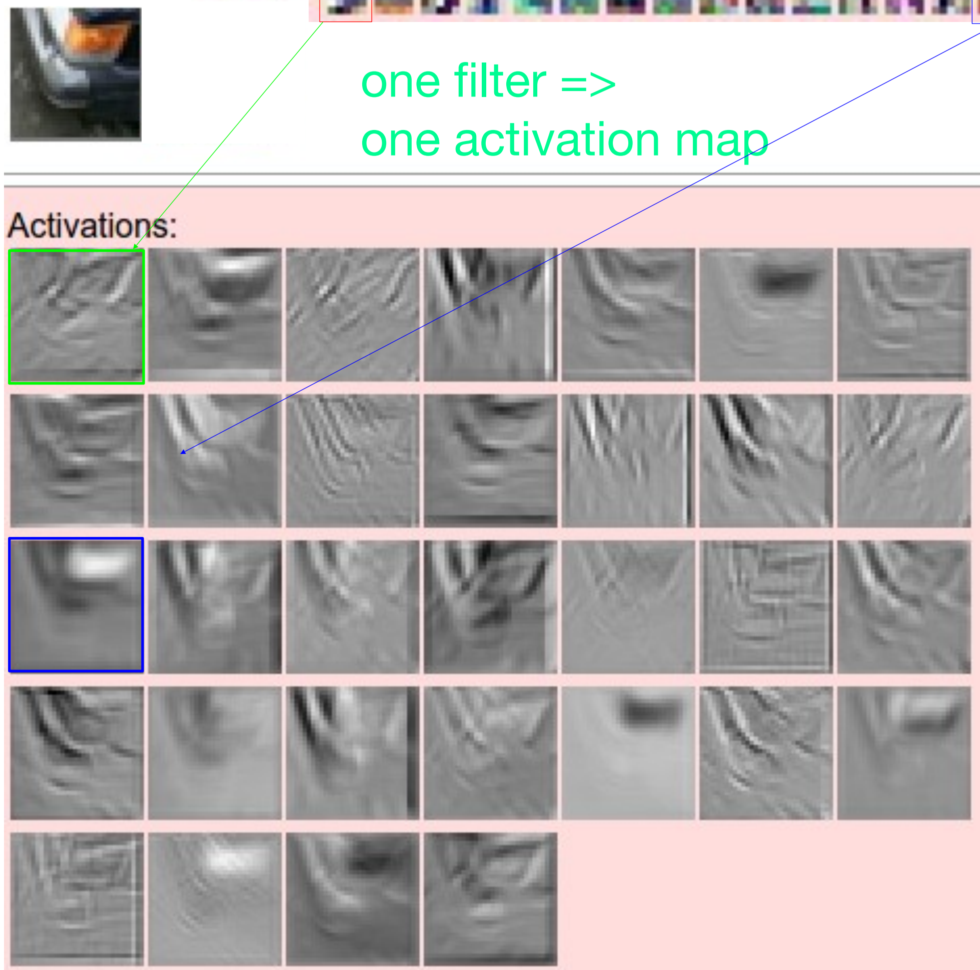
[From recent Yann LeCun slides]



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



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson



example 5x5 filters

(32 total)

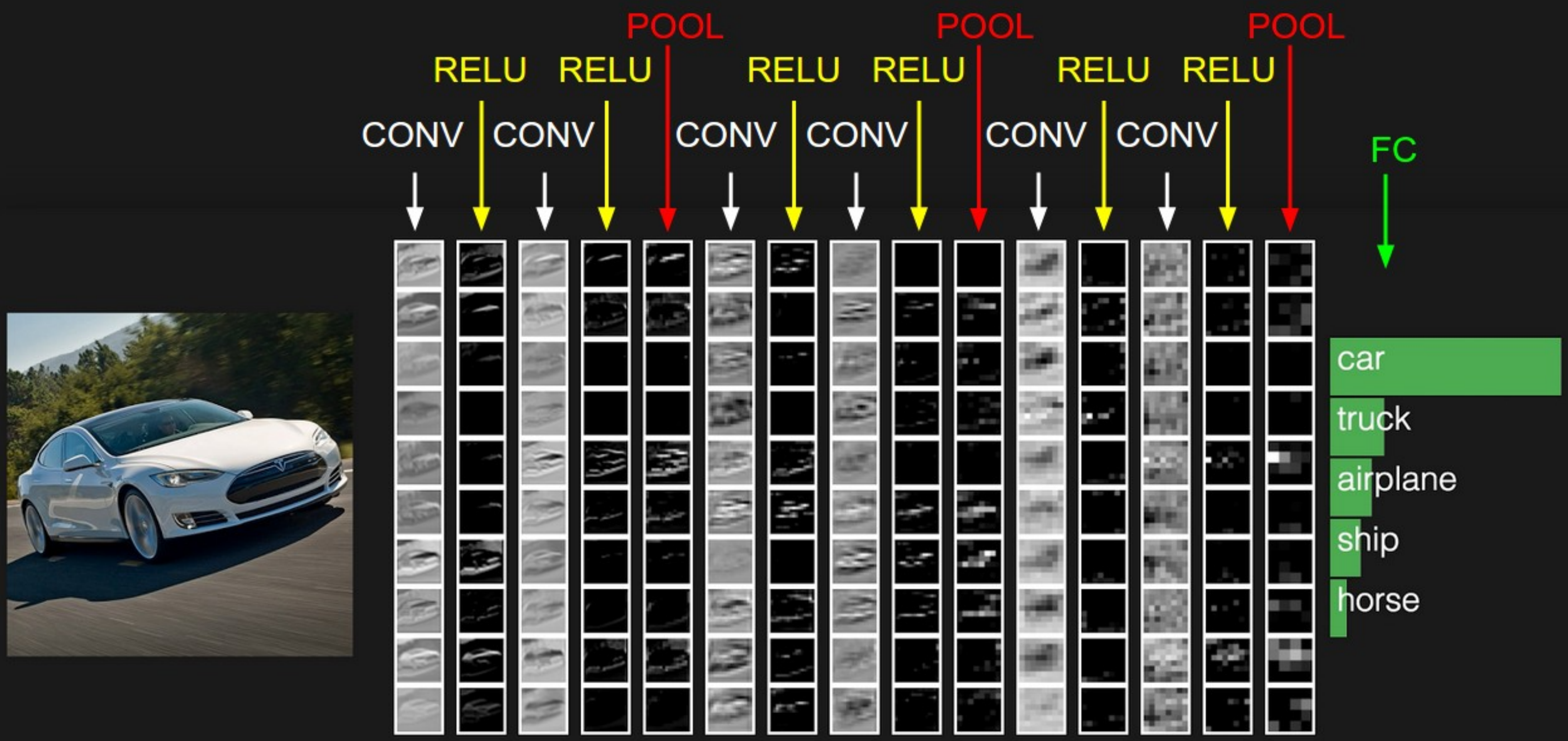
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$

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

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Preview

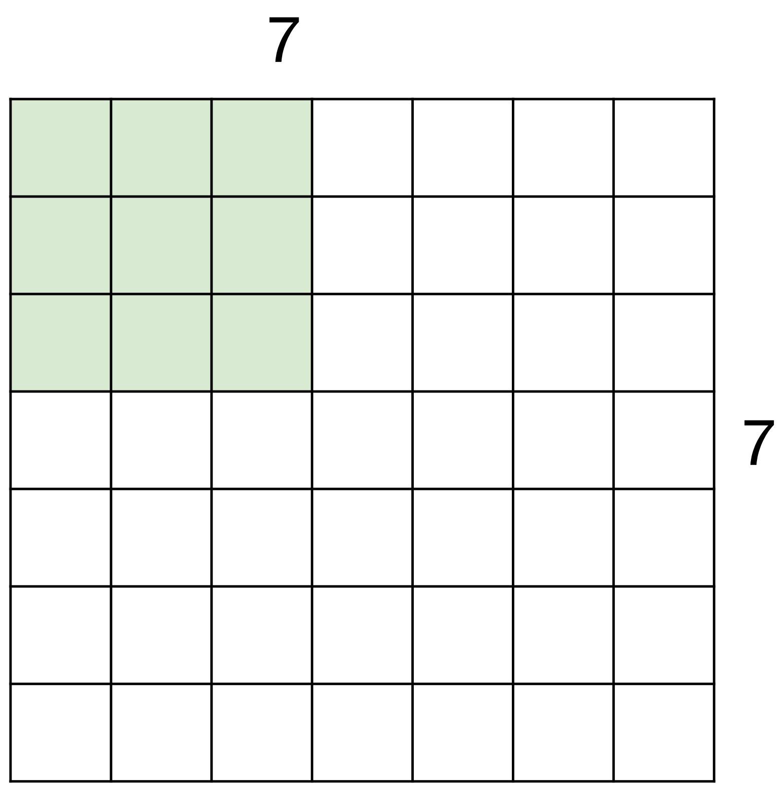


slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

A closer look at spatial dimensions:

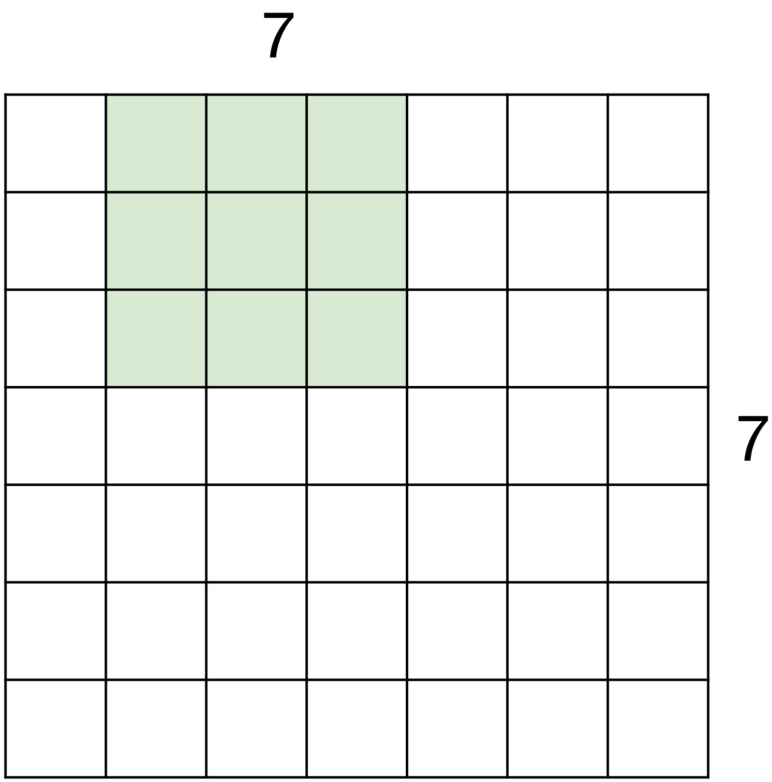
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

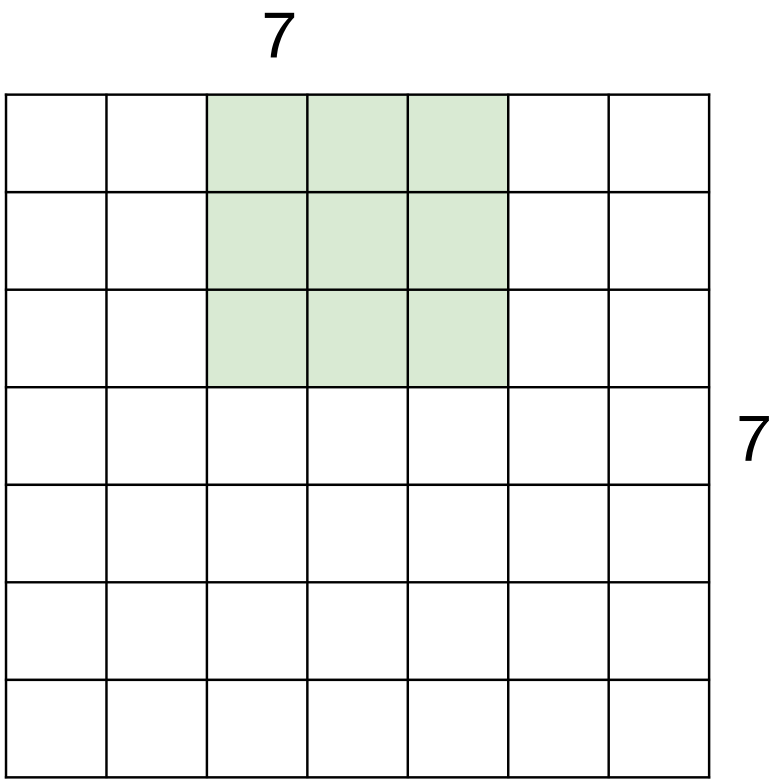
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

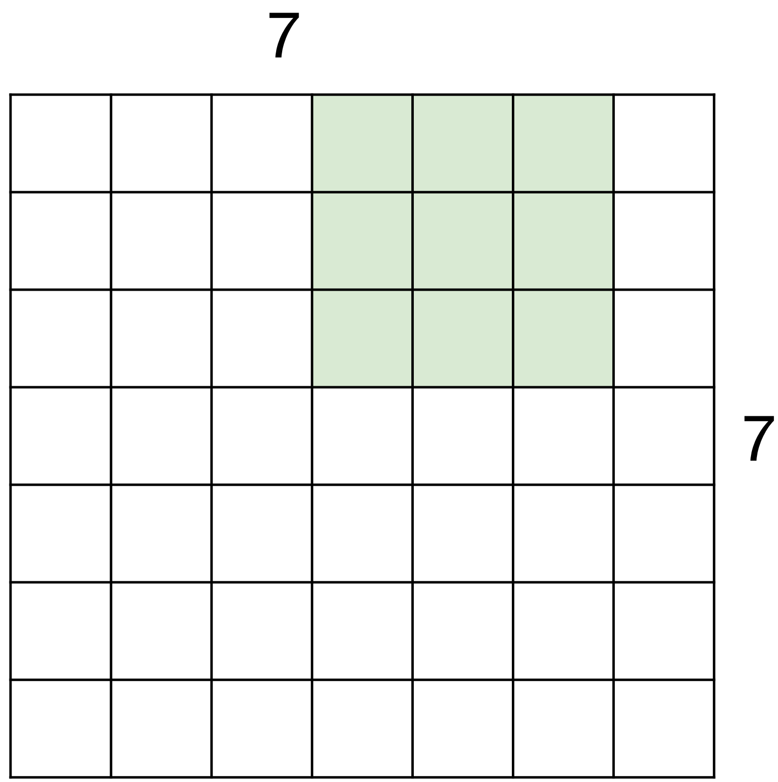
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

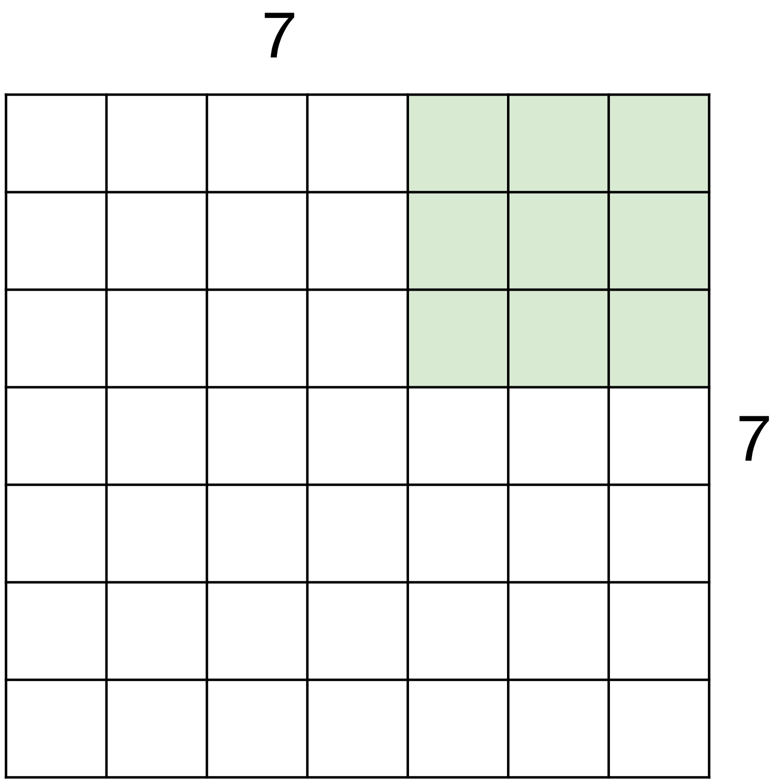
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

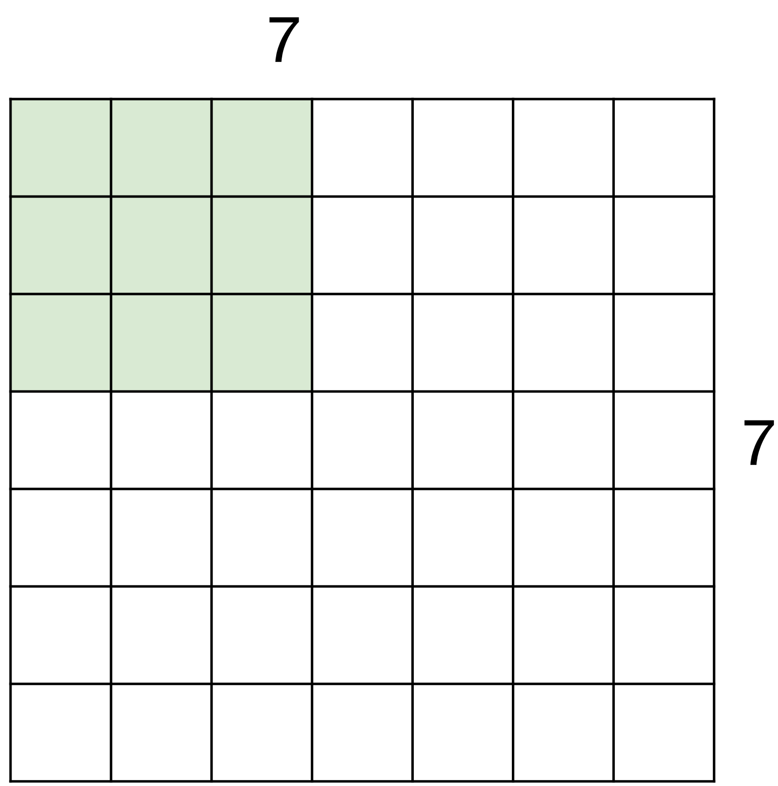
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

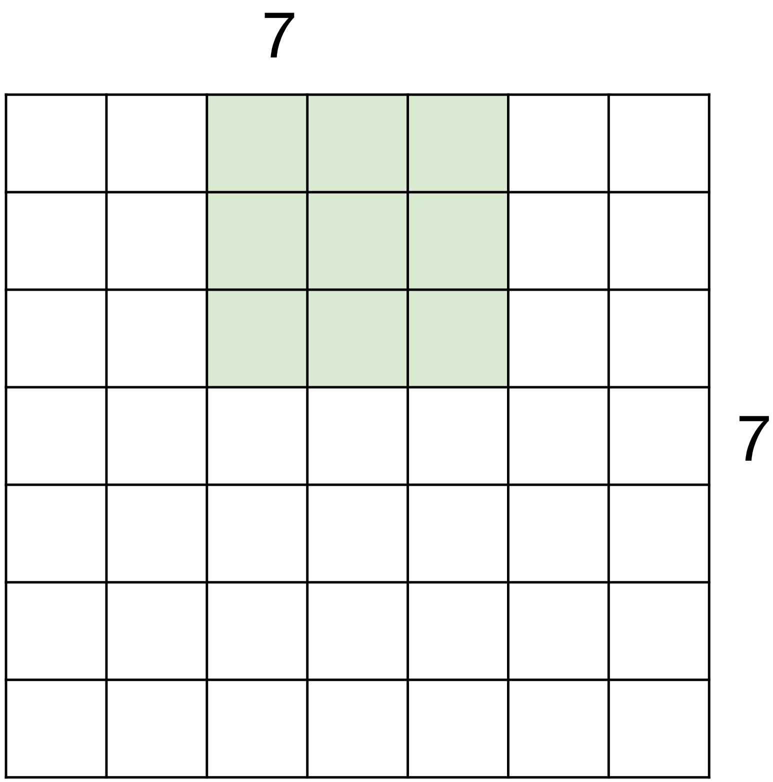
A closer look at spatial dimensions:



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

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

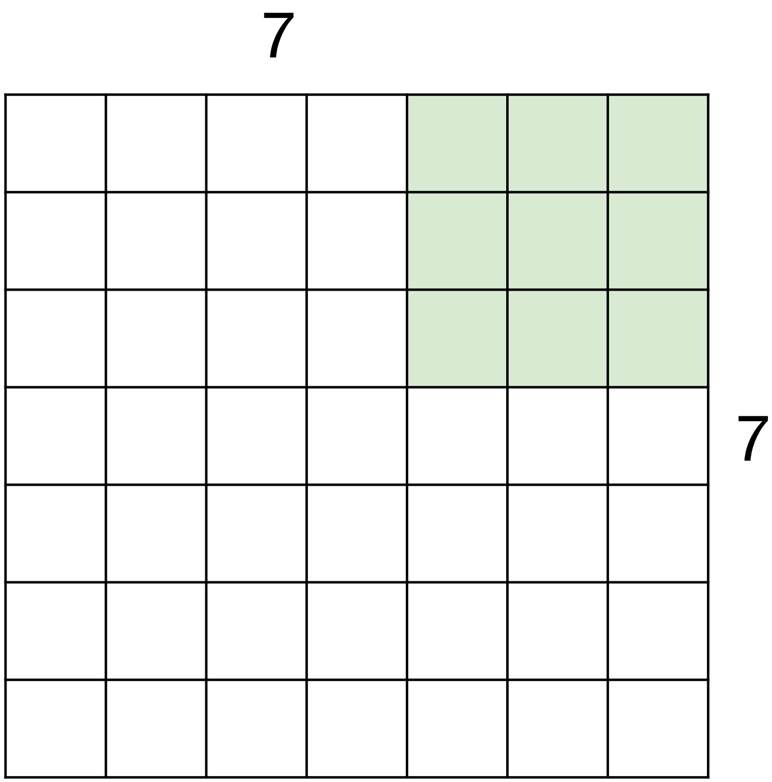
A closer look at spatial dimensions:



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

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

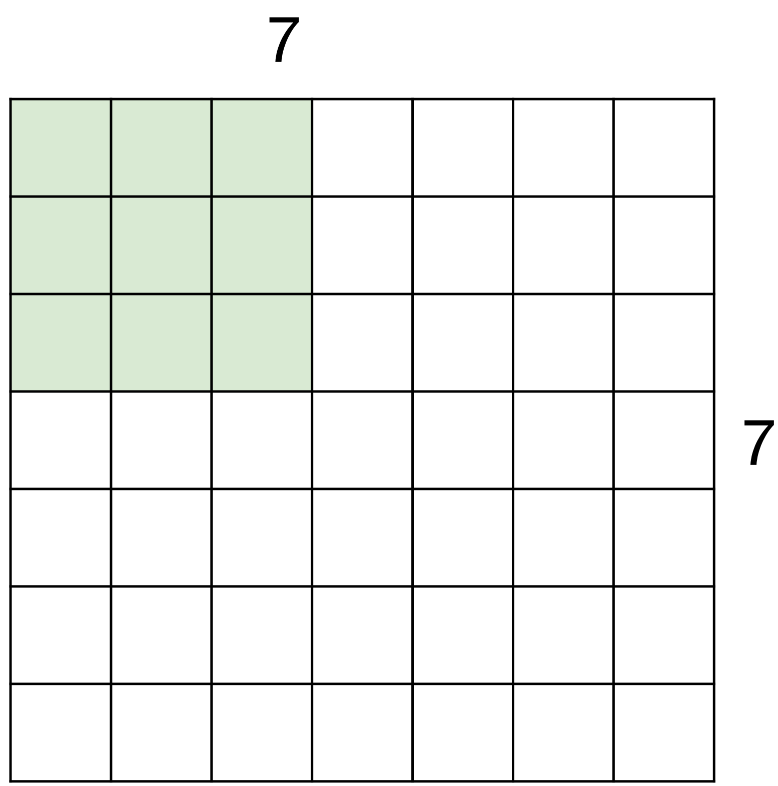
A closer look at spatial dimensions:



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

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

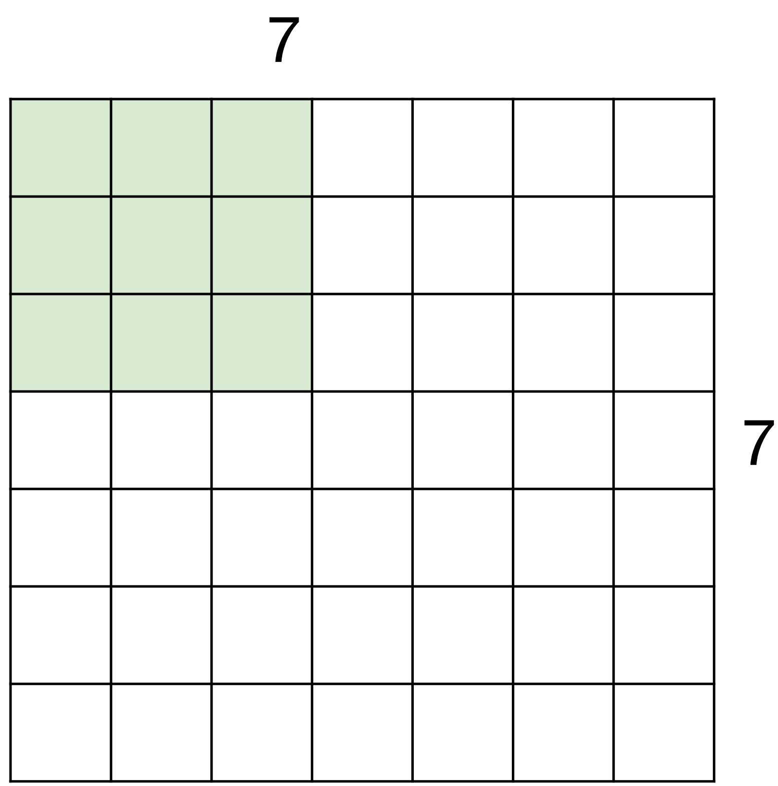
A closer look at spatial dimensions:



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

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

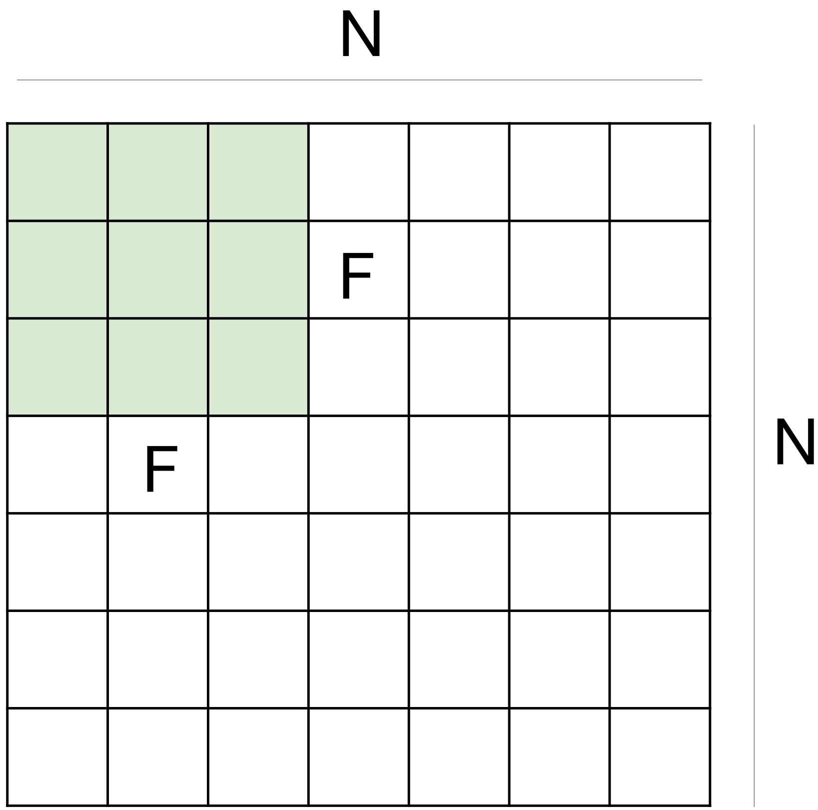
A closer look at spatial dimensions:



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

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

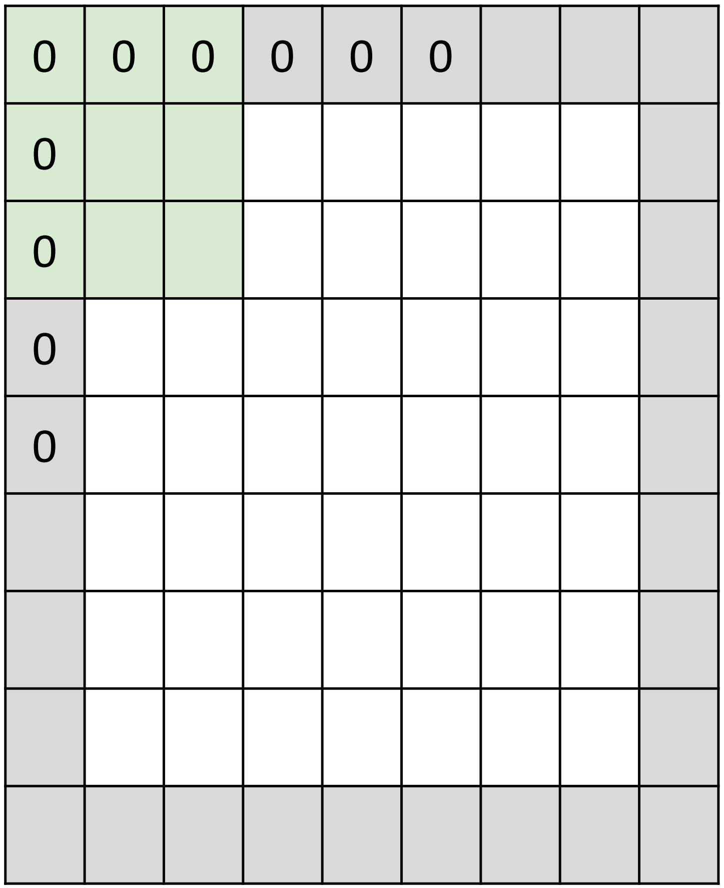
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson



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 :\

In practice: Common to zero pad the border

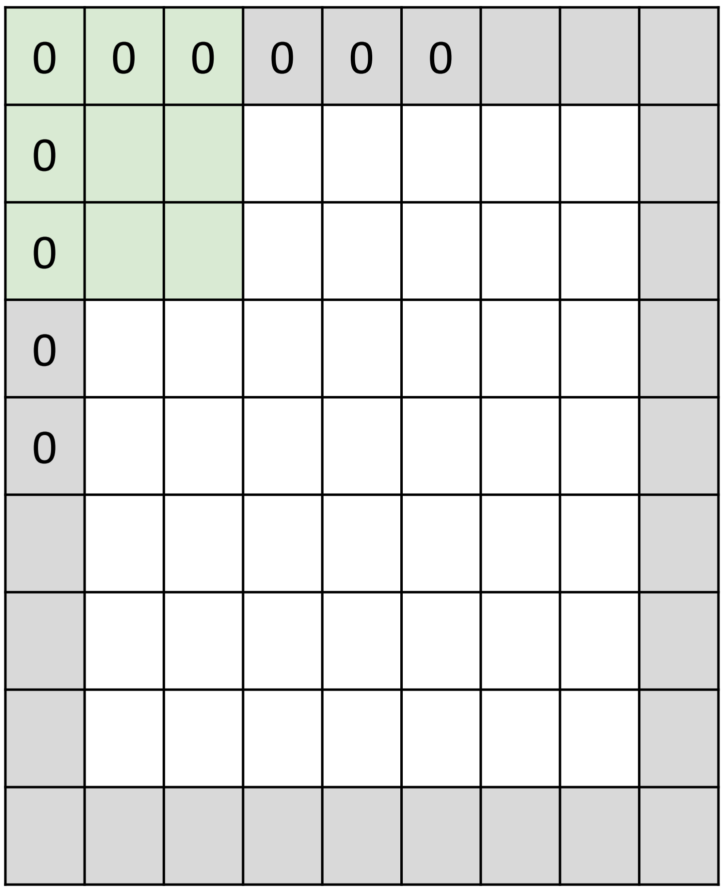


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

In practice: Common to zero pad the border

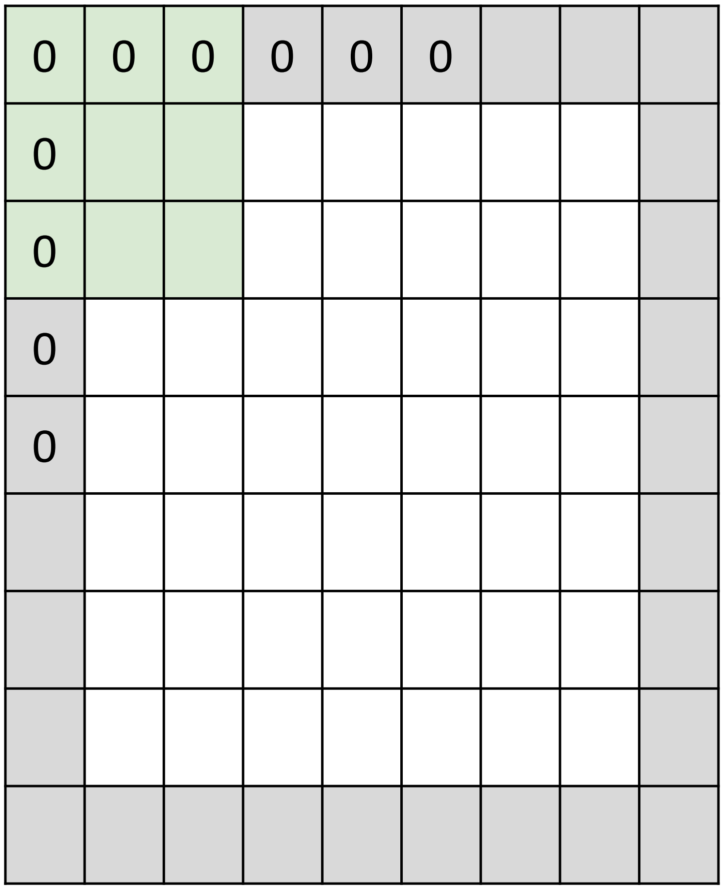


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

In practice: Common to zero pad the border



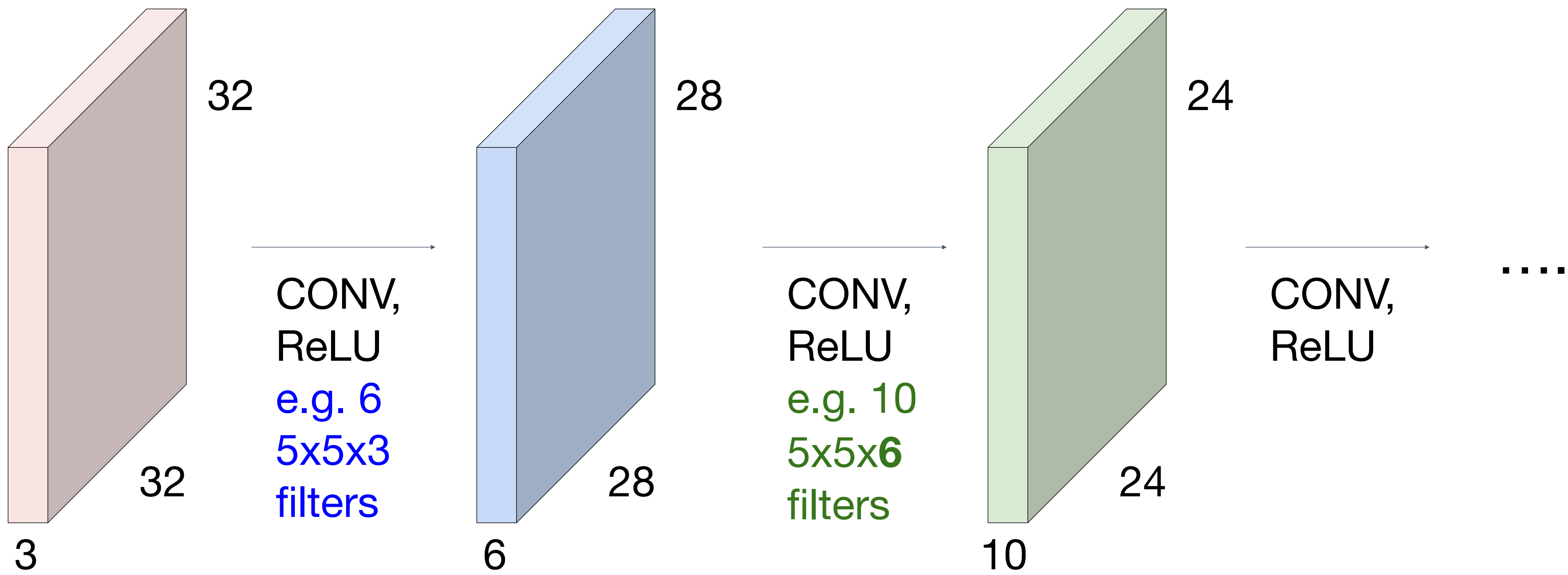
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

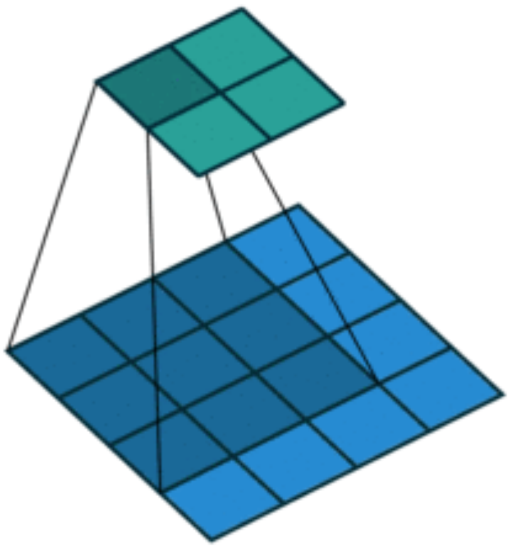
Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Recap: Convolution Layer



$$f = Wx$$

$$W = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

(No padding, no strides)
Convolving a 3 × 3 kernel over a 4 × 4 input using unit strides
(i.e., i = 4, k = 3, s = 1 and p = 0).

Image credit: Vincent Dumoulin and Francesco Visin

Computing the output values of a 2D discrete convolution

$i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, and $p_1 = p_2 = 1$

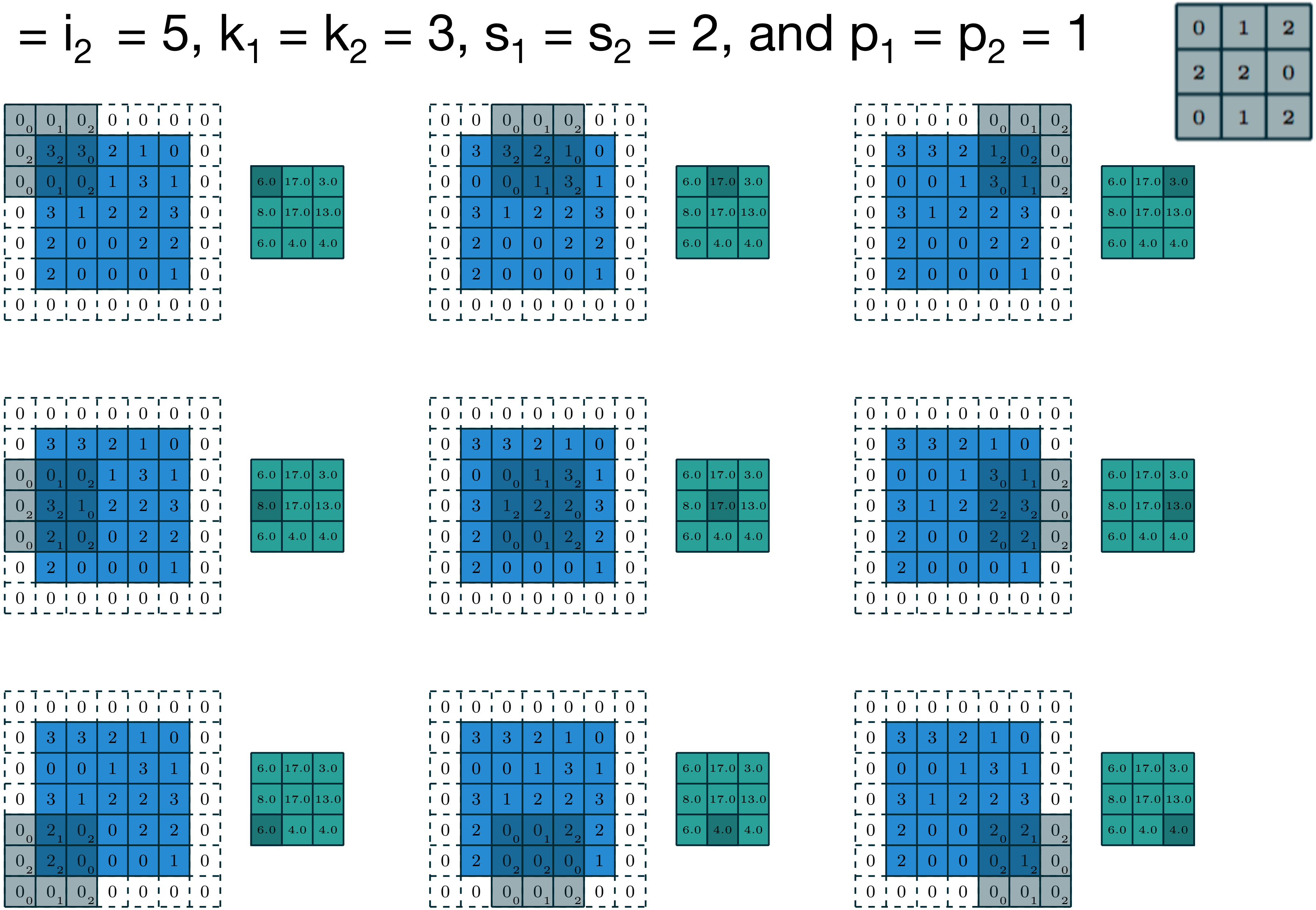
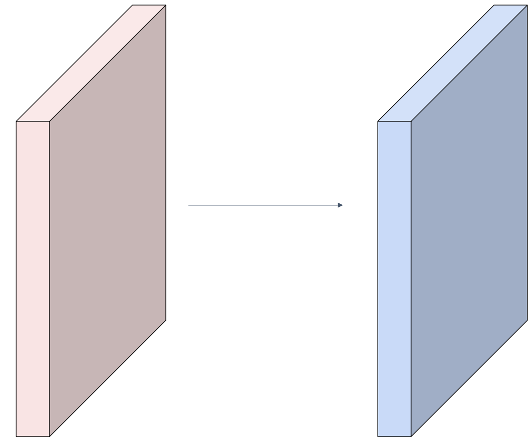


Image credit: Vincent Dumoulin and Francesco Visin

Examples time:

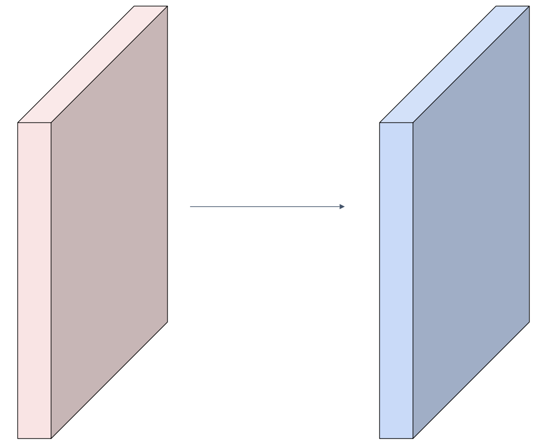


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

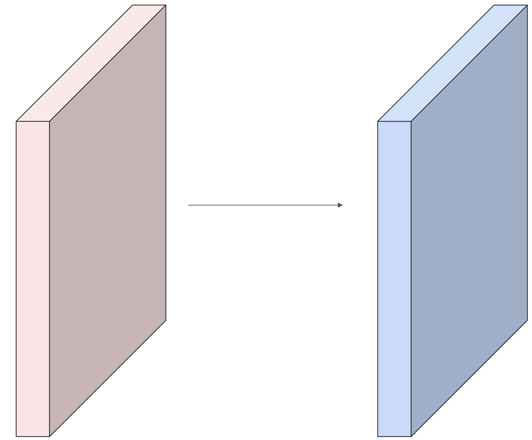
Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Examples time:

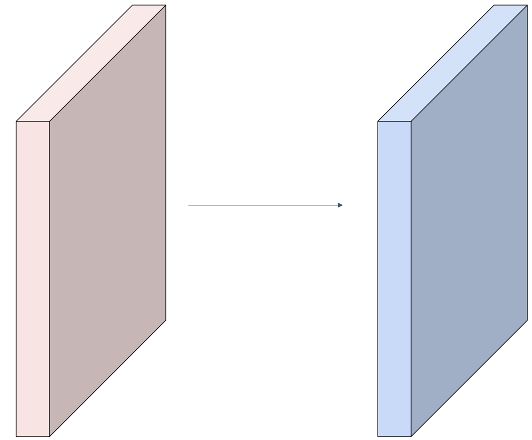


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

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**

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

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.

Common settings:

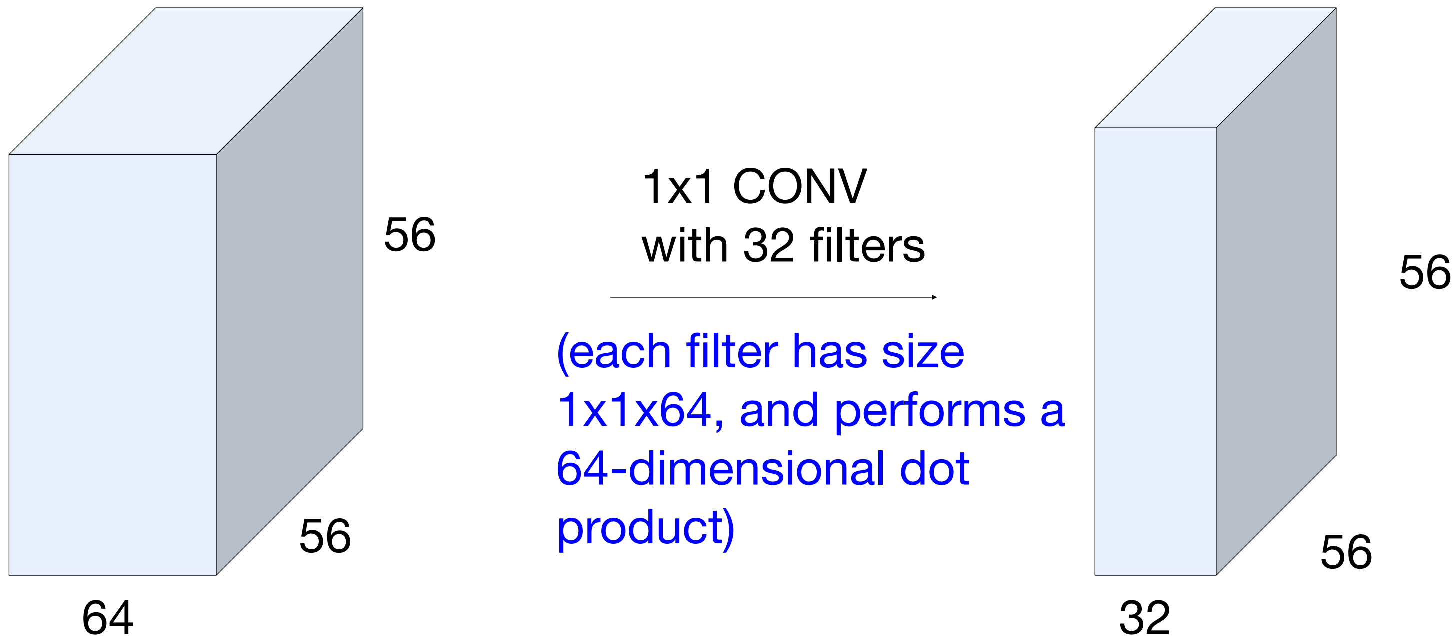
K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

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.

(btw, 1x1 convolution layers make perfect sense)



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Example: CONV layer in PyTorch

CONV2D

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

This module supports TensorFloat32.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit padding on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At groups=1, all inputs are convolved to all outputs.
  - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels and producing half the output channels, and both subsequently concatenated.
  - At groups= `in_channels`, each input channel is convolved with its own set of filters (of size $\frac{\text{out\_channels}}{\text{in\_channels}}$).

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

Summary. To summarize, the Conv Layer:

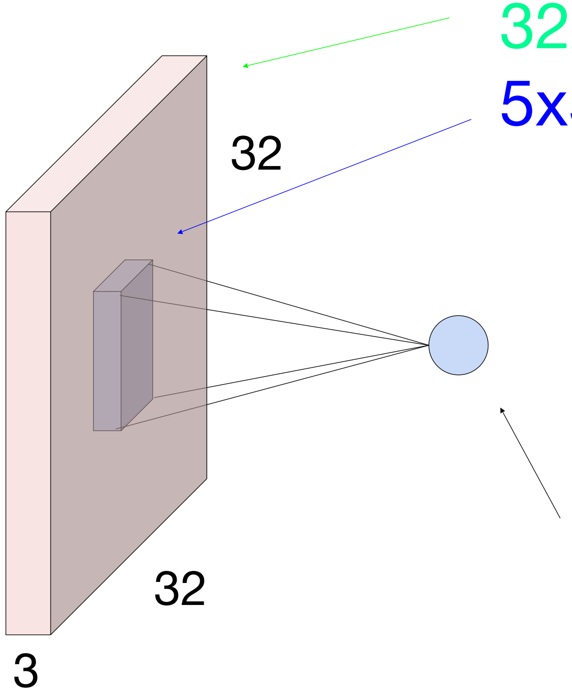
- Accepts a volume of size $W_1 \times H_1 \times$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

The brain/neuron view of CONV Layer

32x32x3 image

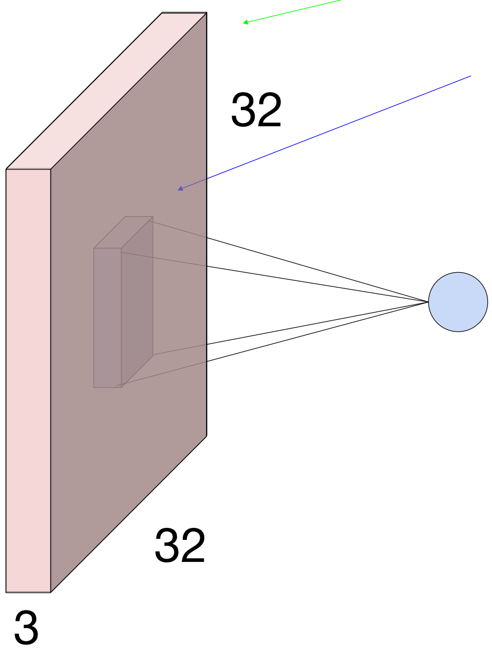
5x5x3 filter



1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

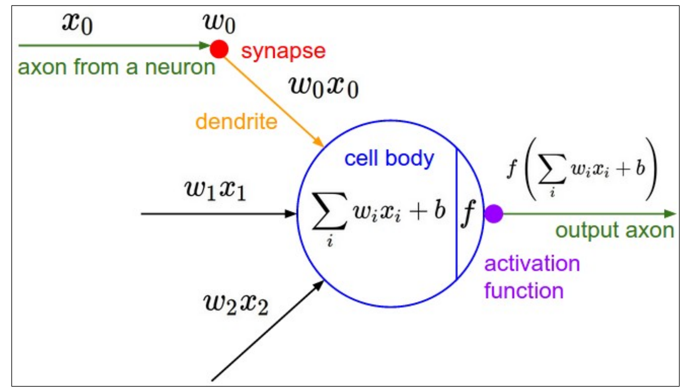
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

The brain/neuron view of CONV Layer



32x32x3 image

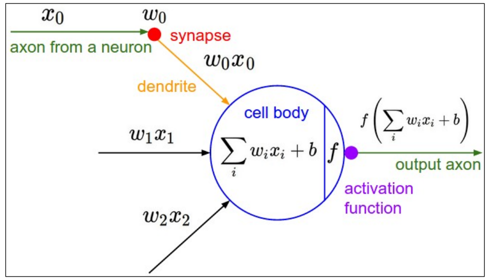
5x5x3 filter

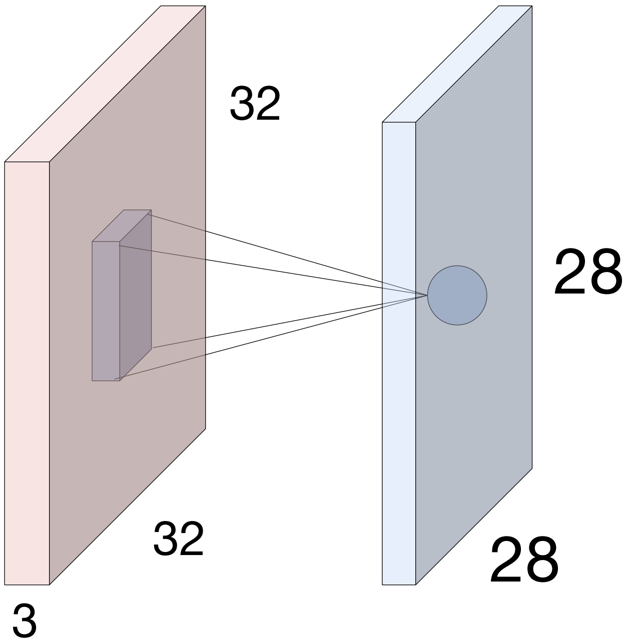


It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer





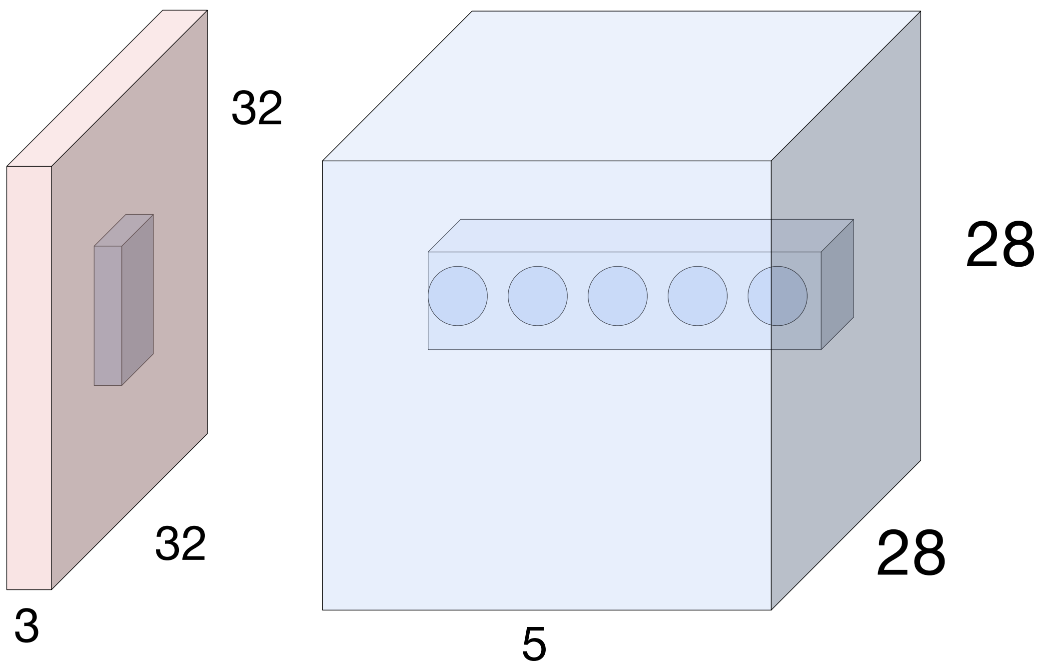
An activation map is a 28x28 sheet of neuron outputs:

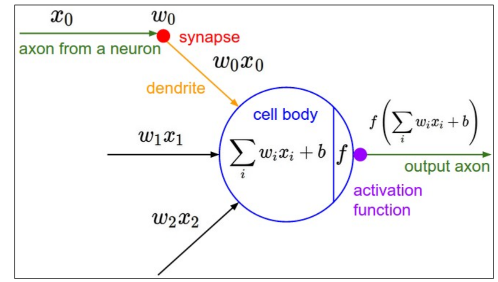
1. Each is connected to a small region in the input
2. All of them share parameters

“5x5 filter” -> “5x5 receptive field for each neuron”

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

The brain/neuron view of CONV Layer

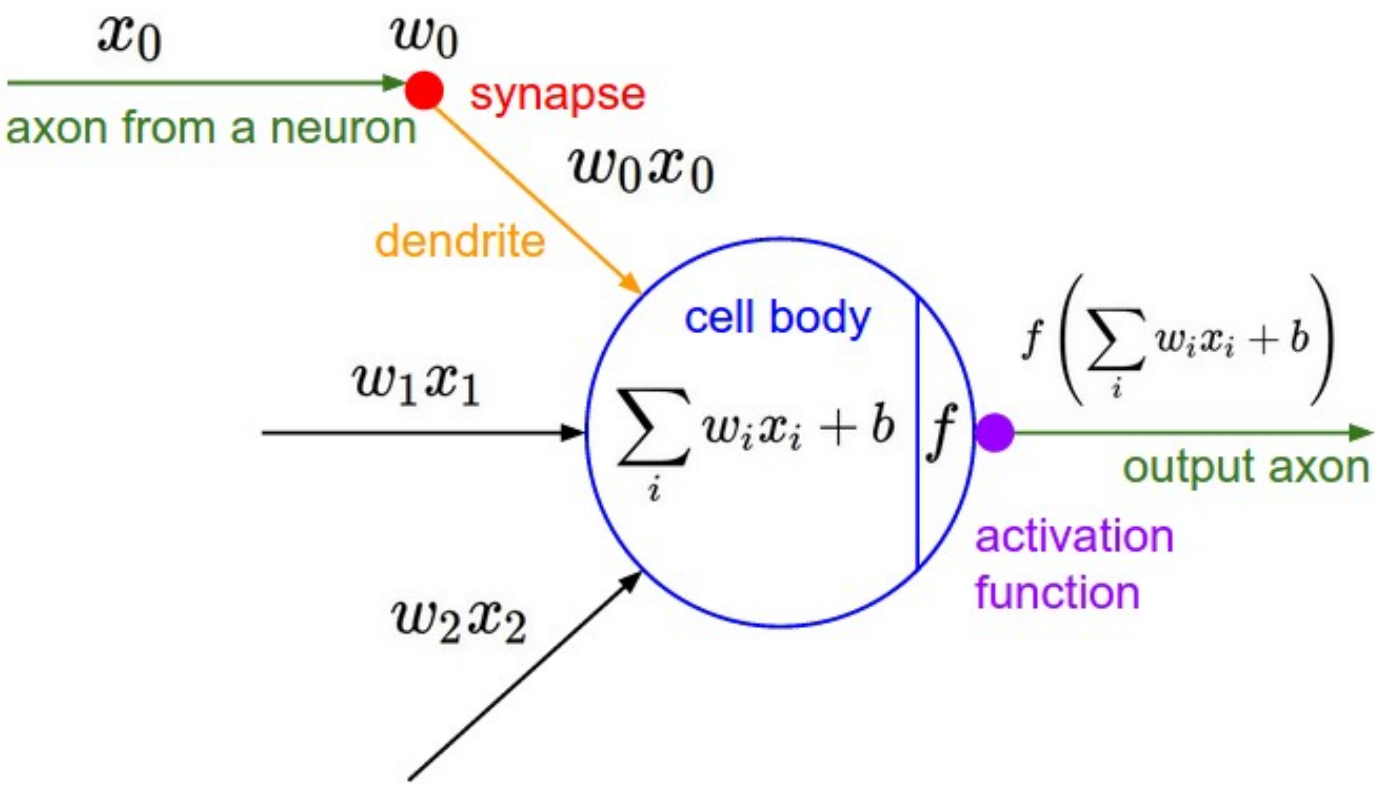




E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume

Activation Functions

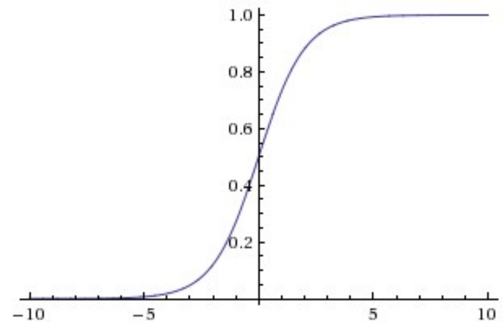


slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

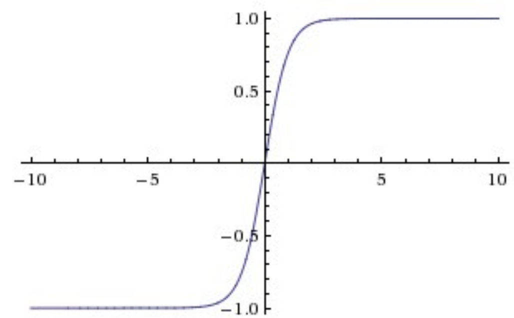
Activation Functions

Sigmoid

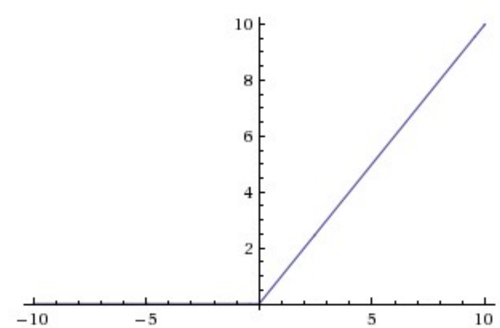
$$\sigma(x) = 1/(1 + e^{-x})$$



tanh tanh(x)



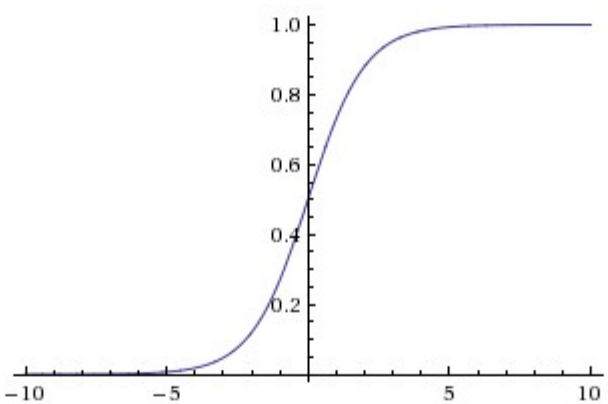
ReLU max(0,x)



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Activation Functions

$$\sigma(x) = 1/(1+e^{-x})$$



Sigmoid

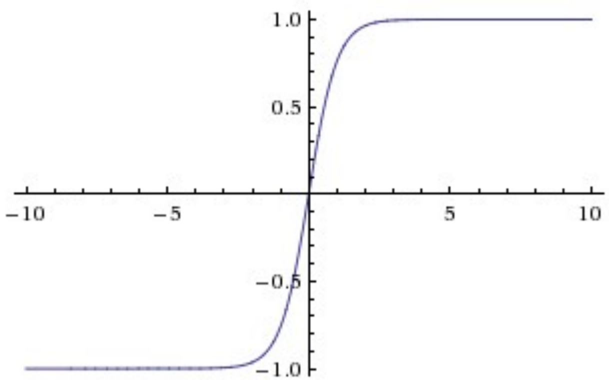
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

3 problems:

1. Saturated neurons "kill" the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Activation Functions

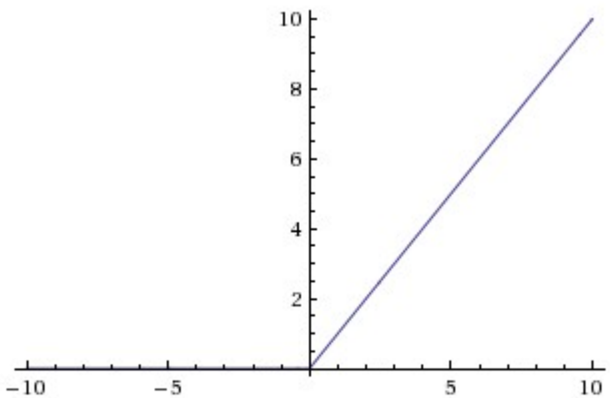


tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

[LeCun et al., 1991]

Activation Functions



ReLU
(Rectified Linear Unit)

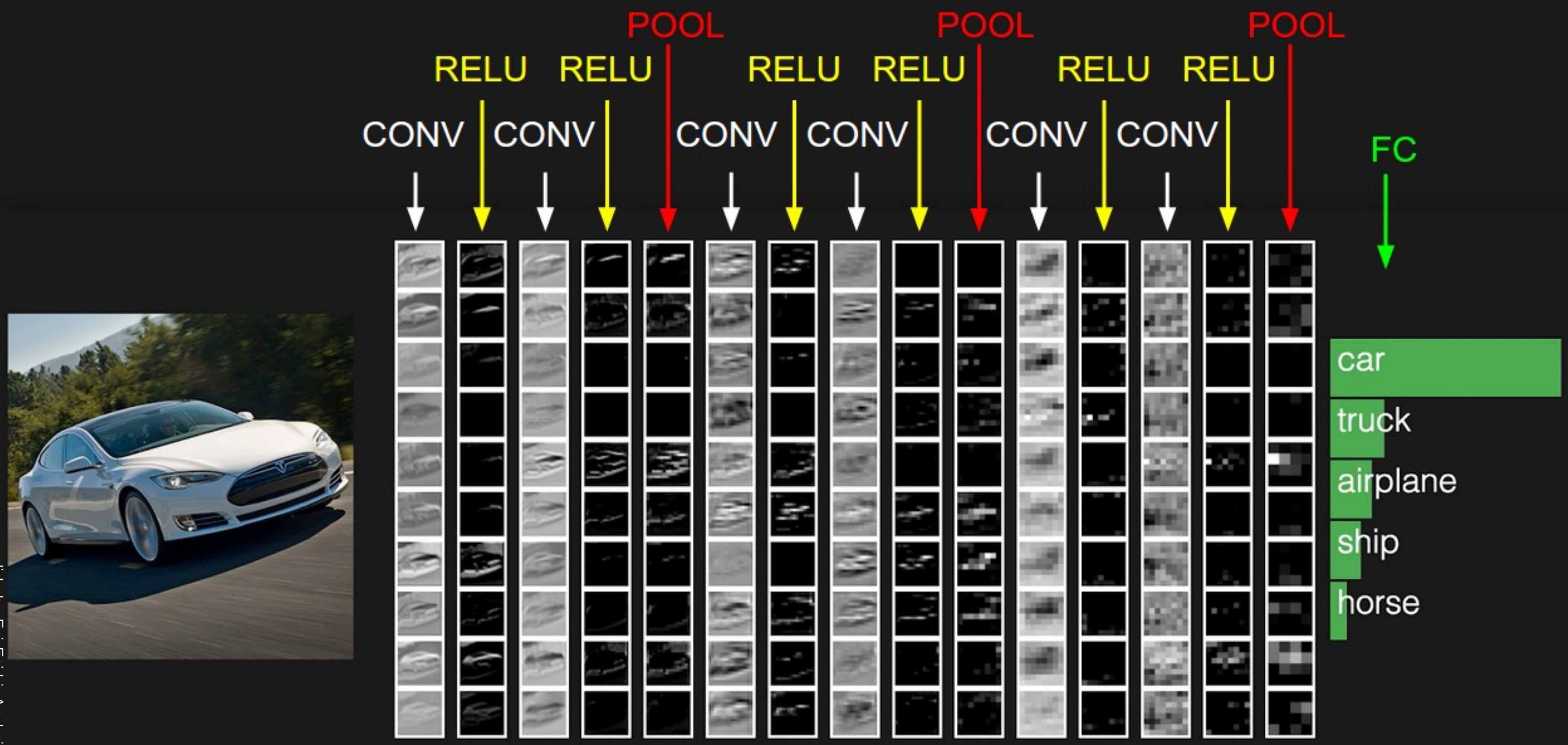
- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

[Krizhevsky et al., 2012]

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

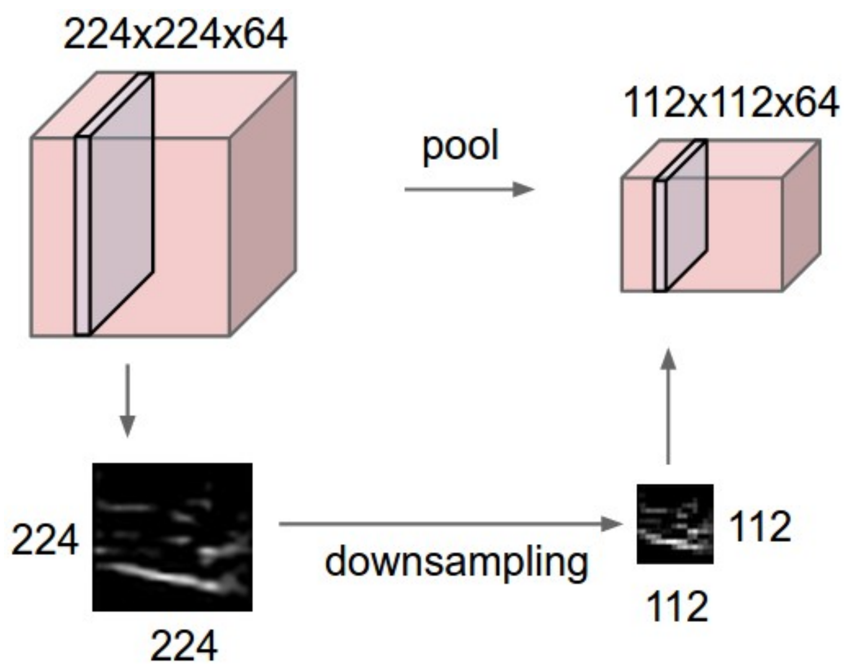
two more layers to go: POOL/FC



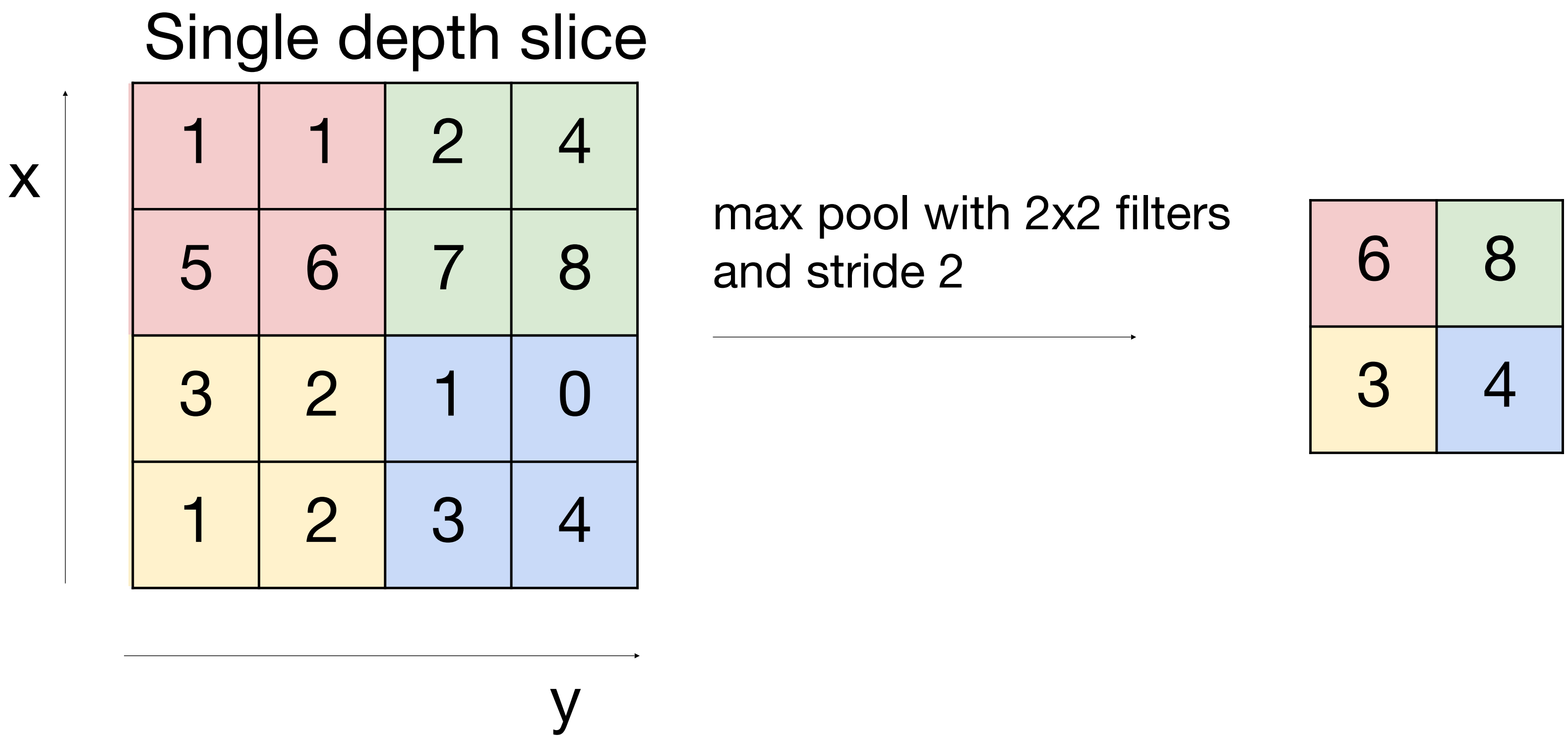
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

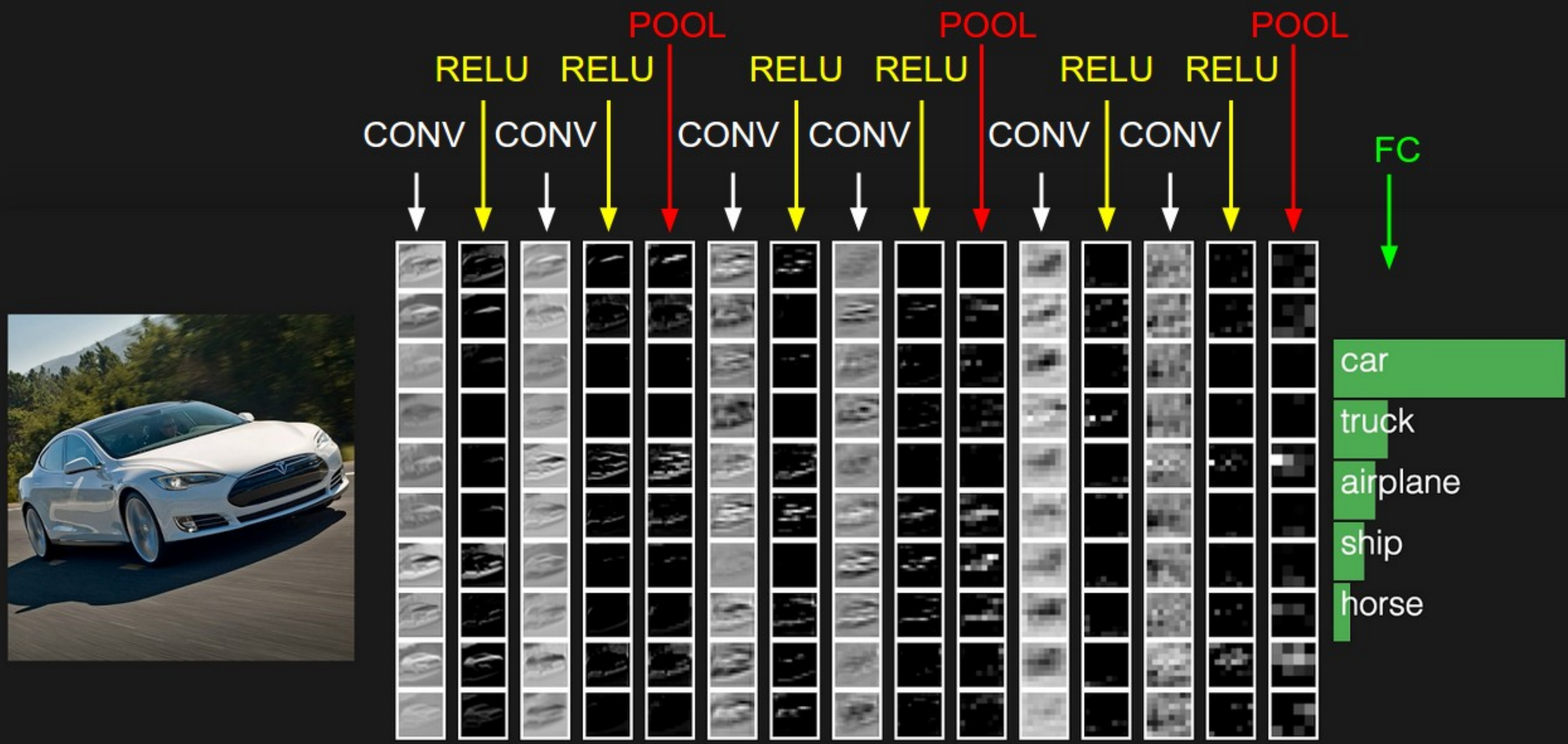
F = 2, S = 2
F = 3, S = 2

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



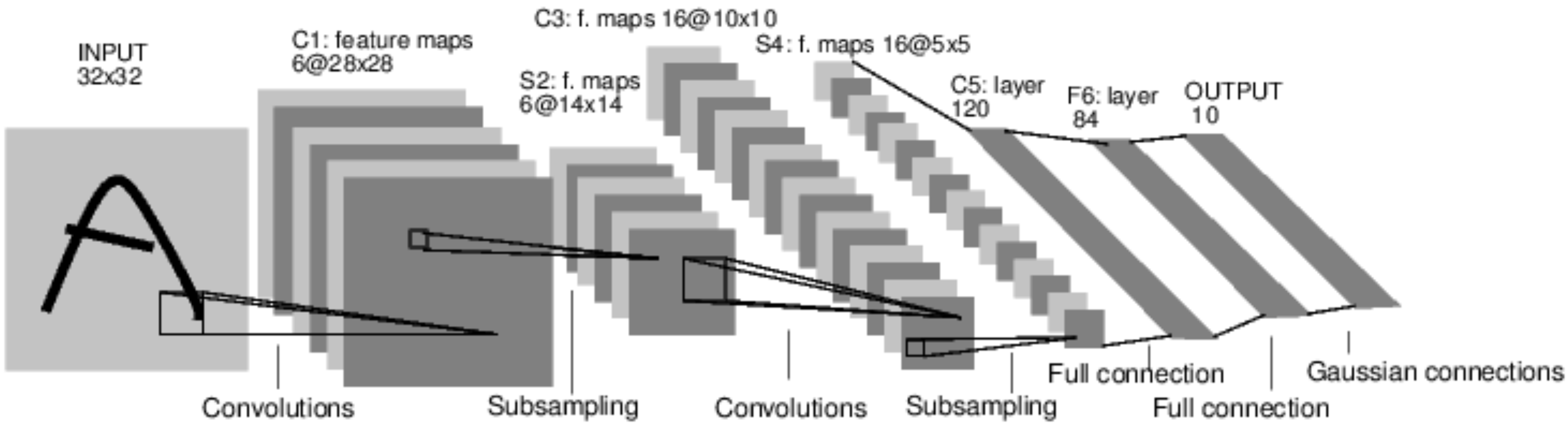
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

[ConvNetJS demo: training on CIFAR-10]

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Case studies

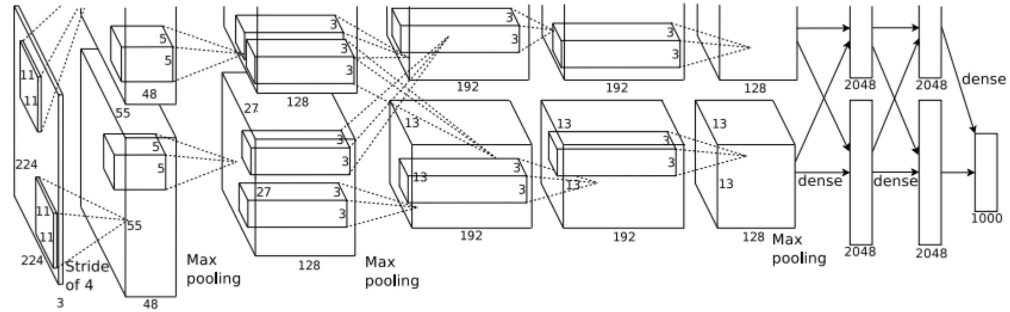
Case Study: LeNet-5 *[LeCun et al., 1998]*



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

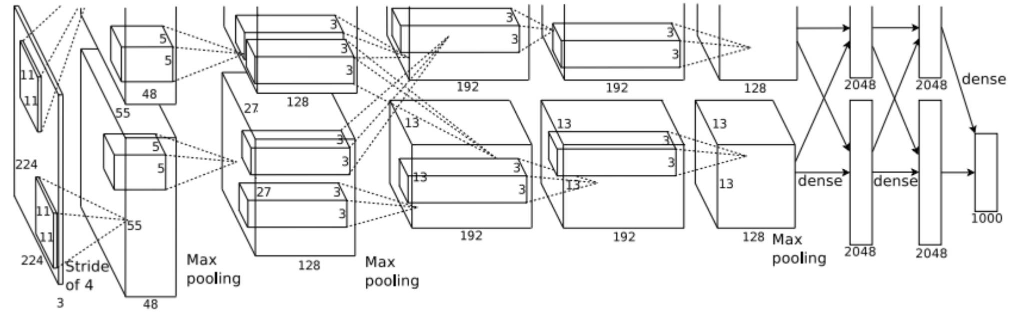
=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

=>

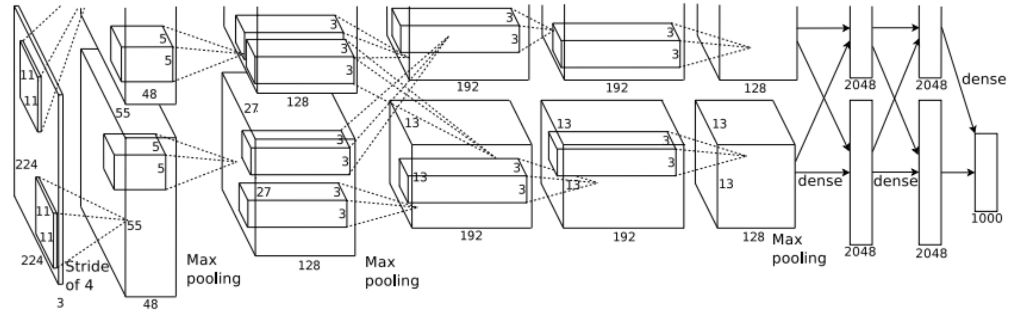
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

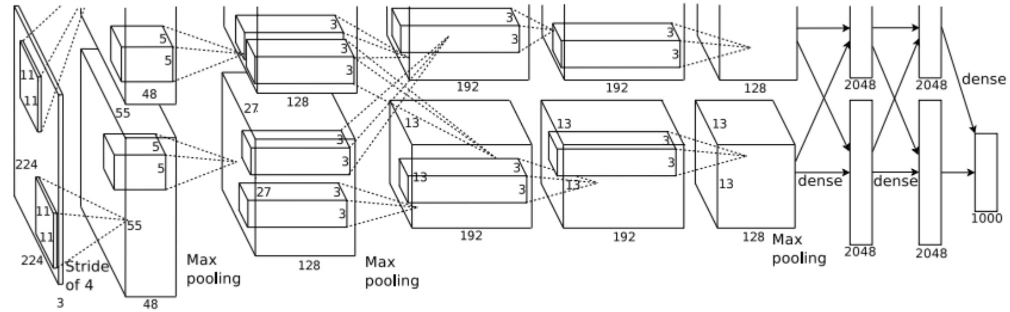
=>

Output volume **[55x55x96]**

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



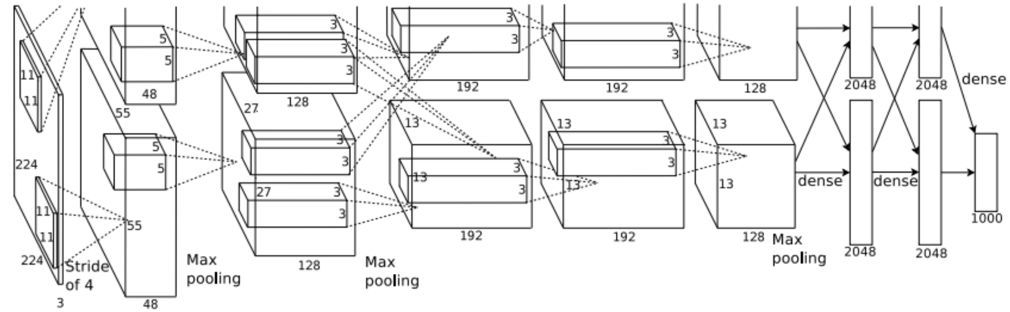
Input: 227x227x3 images
After CONV1: 55x55x96

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

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

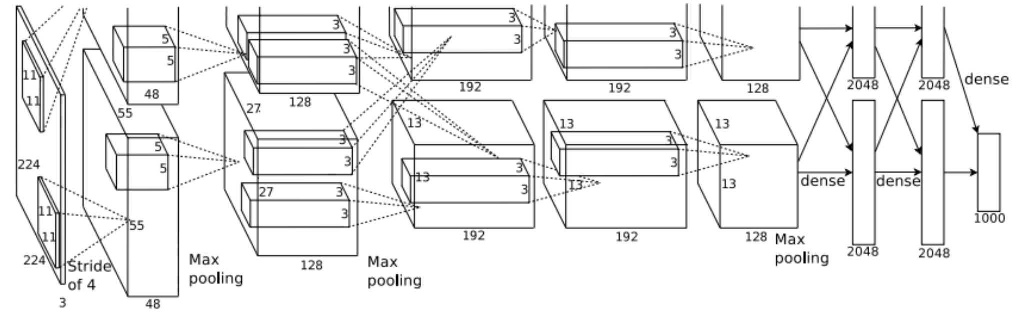
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

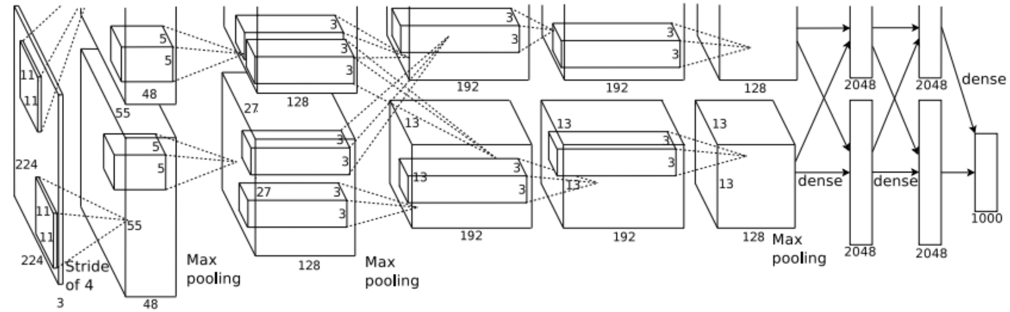
Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

Parameters: 0!

Case Study: AlexNet

[Krizhevsky et al. 2012]

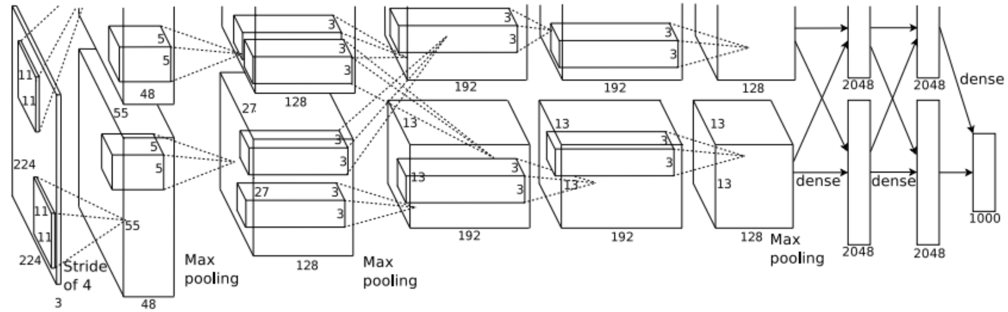


Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Case Study: AlexNet

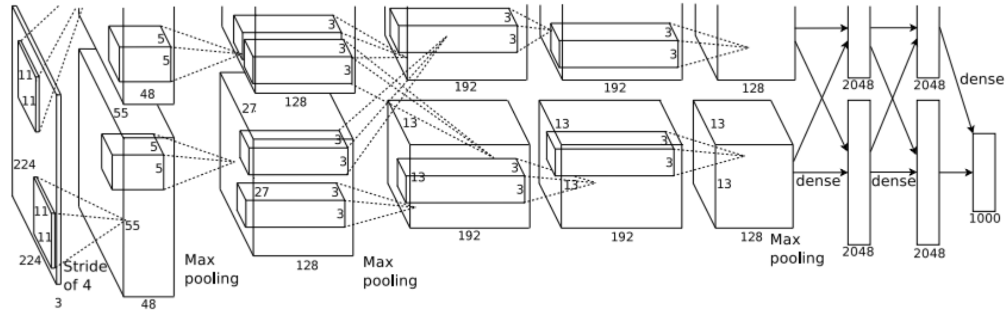
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Case Study: AlexNet

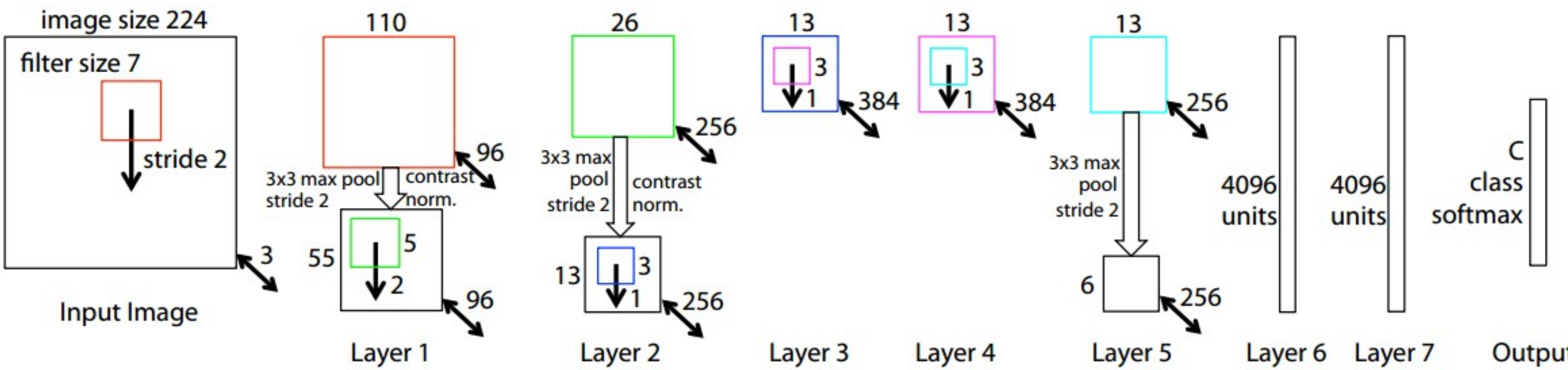
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

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 Study: VGGNet

[Simonyan and Zisserman, 2014]

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

<table>
<tr><td colspan="6">ConvNet Configuration</td></tr>
<tr><td>A</td><td>A-LRN</td><td>B</td><td>C</td><td>D</td><td>E</td></tr>
<tr><td>11 weight layers</td><td>11 weight layers</td><td>13 weight layers</td><td>16 weight layers</td><td>16 weight layers</td><td>19 weight layers</td></tr>
<tr><td colspan="6">input (224 × 224 RGB image)</td></tr>
<tr><td>conv3-64</td><td>conv3-64
LRN</td><td>conv3-64
conv3-64</td><td>conv3-64
conv3-64</td><td>conv3-64
conv3-64</td><td>conv3-64
conv3-64</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td>conv3-128</td><td>conv3-128</td><td>conv3-128
conv3-128</td><td>conv3-128
conv3-128</td><td>conv3-128
conv3-128</td><td>conv3-128
conv3-128</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td>conv3-256
conv3-256</td><td>conv3-256
conv3-256</td><td>conv3-256
conv3-256</td><td>conv3-256
conv3-256
conv1-256</td><td>conv3-256
conv3-256
conv3-256</td><td>conv3-256
conv3-256
conv3-256
conv3-256</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td>conv3-512
conv3-512</td><td>conv3-512
conv3-512</td><td>conv3-512
conv3-512</td><td>conv3-512
conv3-512
conv1-512</td><td>conv3-512
conv3-512
conv3-512</td><td>conv3-512
conv3-512
conv3-512
conv3-512</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td>conv3-512
conv3-512</td><td>conv3-512
conv3-512</td><td>conv3-512
conv3-512</td><td>conv3-512
conv3-512
conv1-512</td><td>conv3-512
conv3-512
conv3-512</td><td>conv3-512
conv3-512
conv3-512
conv3-512</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td colspan="6">FC-4096</td></tr>
<tr><td colspan="6">FC-4096</td></tr>
<tr><td colspan="6">FC-1000</td></tr>
<tr><td colspan="6">soft-max</td></tr>
</table>

best model

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

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

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

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

(not counting biases)

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

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

(not counting biases)

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

ConvNet Configuration		
B	C	D
13 weight layers	16 weight layers	16 weight layers
put (224 × 224 RGB image)		
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
maxpool		
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
FC-4096		
FC-4096		
FC-1000		
soft-max		

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

(not counting biases)

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

Note:

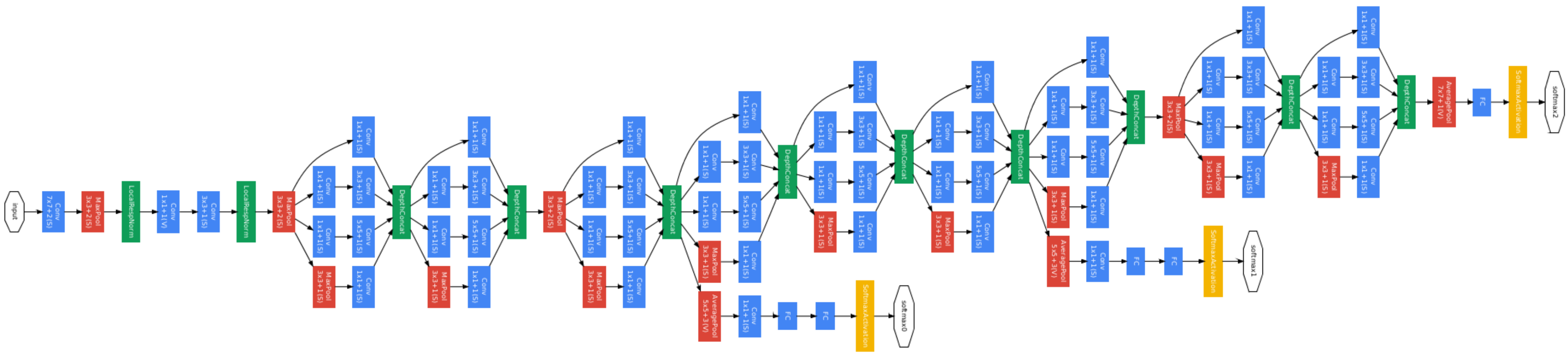
Most memory is in early CONV

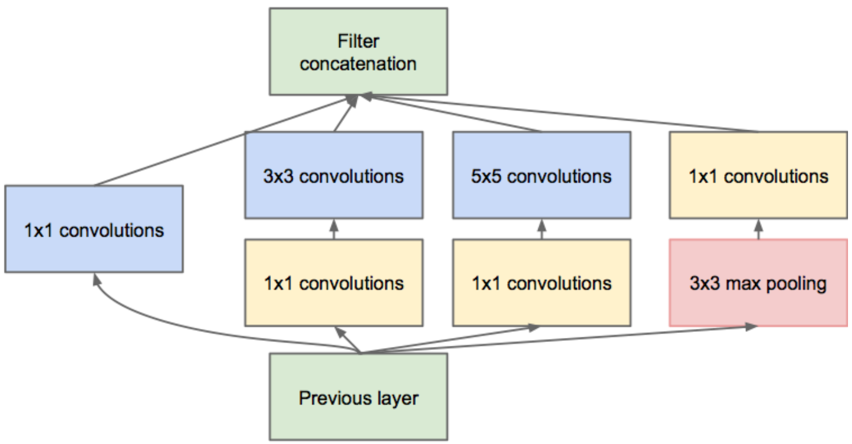
Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Case Study: GoogLeNet

[Szegedy et al., 2014]





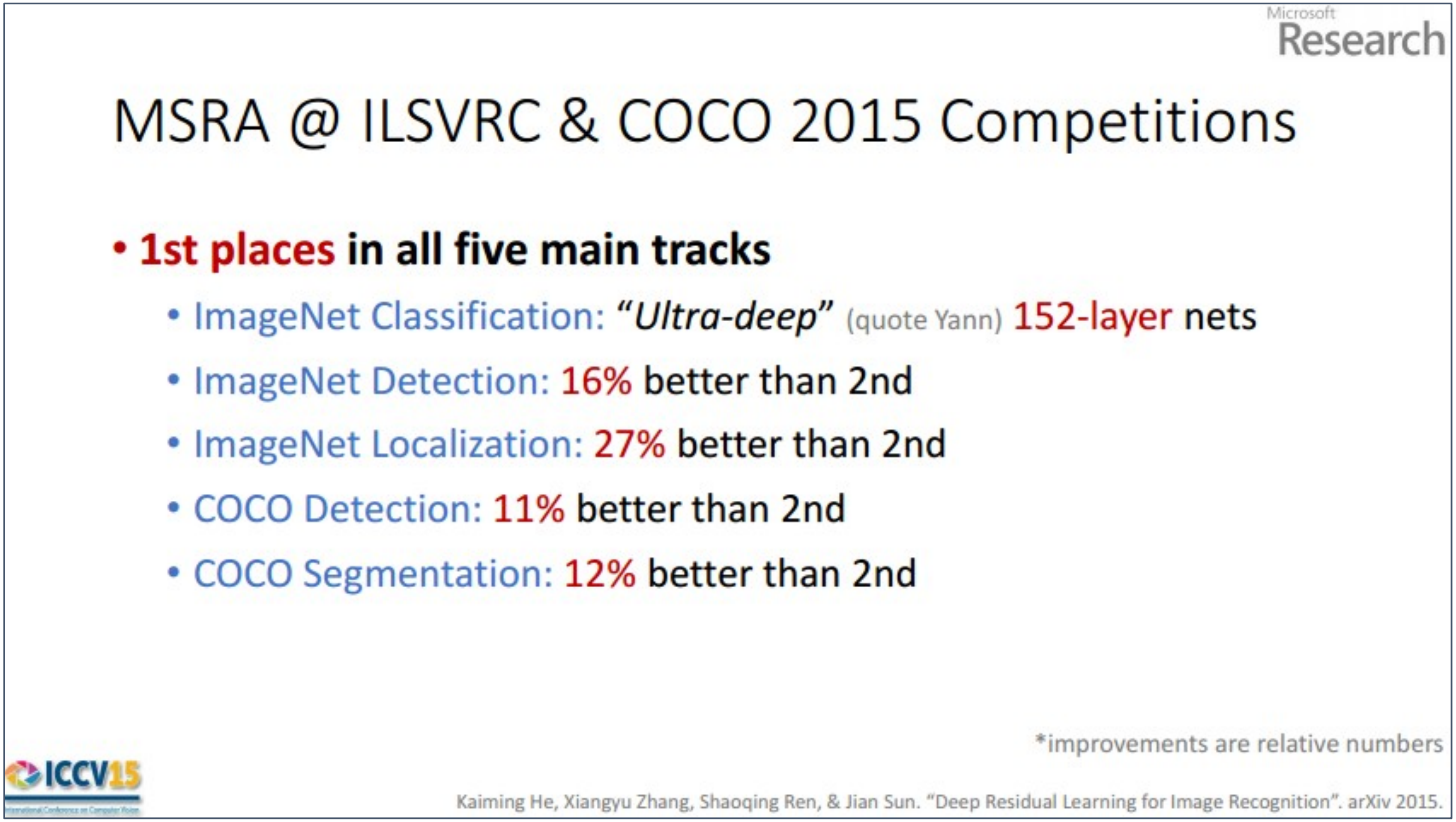
Inception module

ILSVRC 2014 winner (6.7% top 5 error)

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner
(3.6% top 5 error)

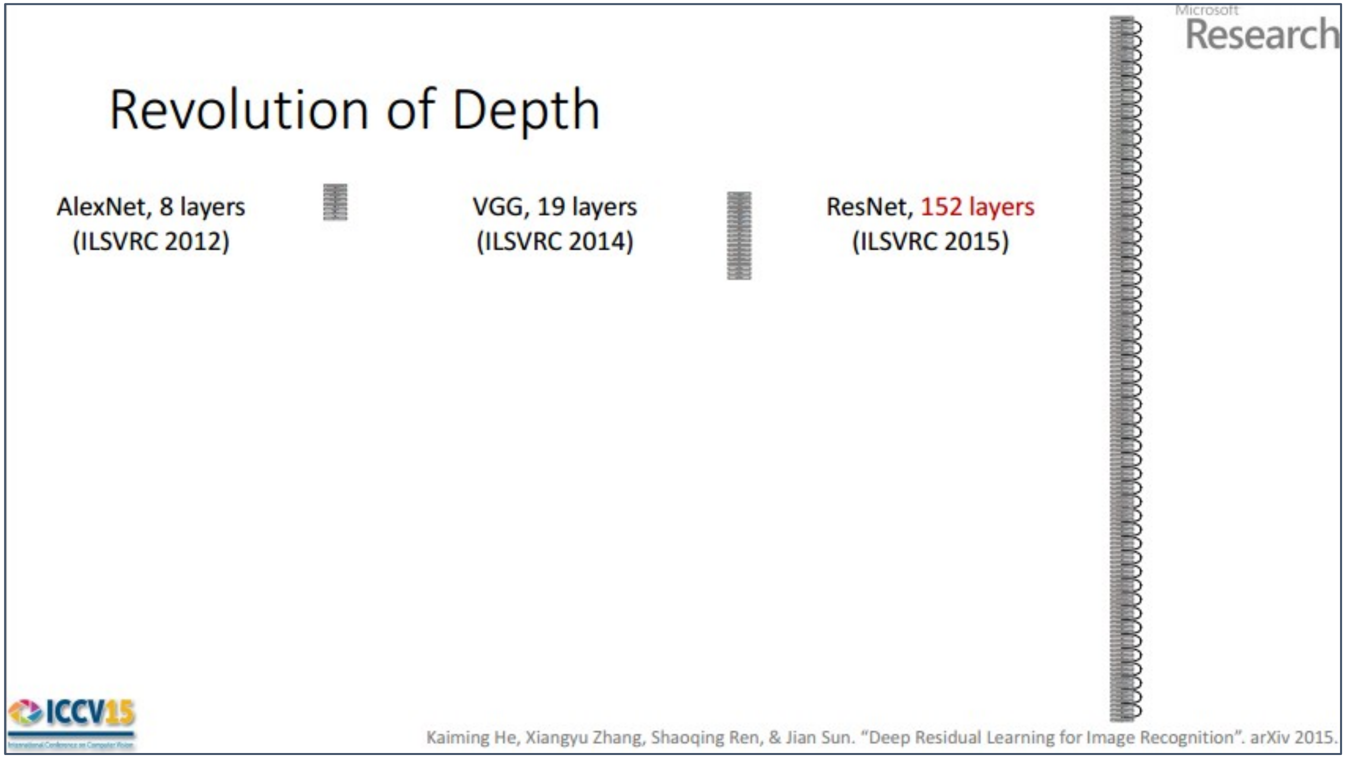


Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner
(3.6% top 5 error)



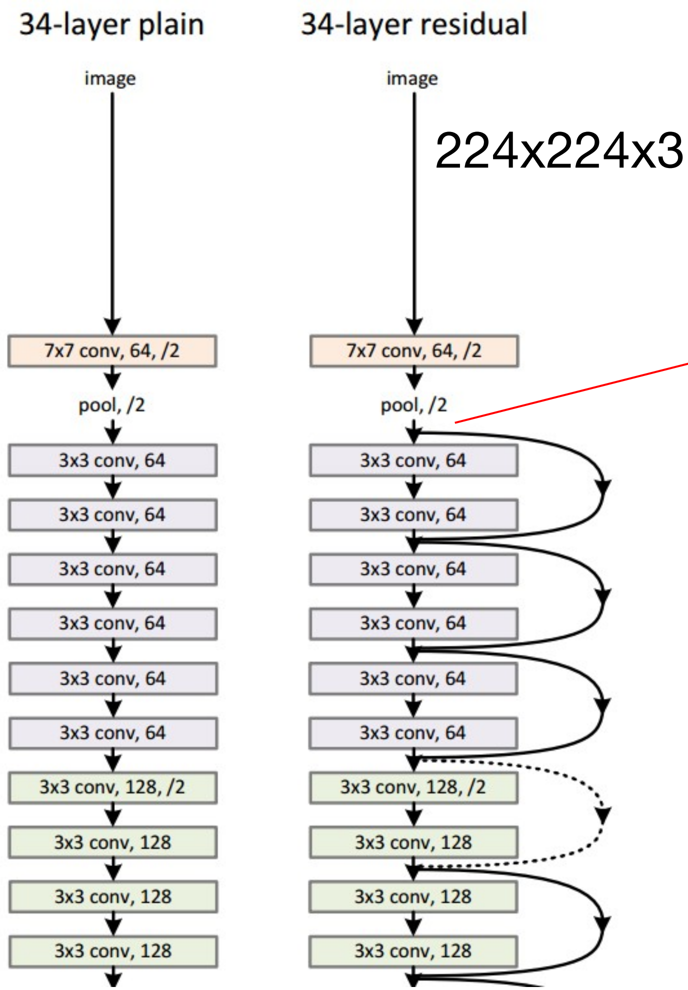
2-3 weeks of training on 8 GPU machine

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

(slide from Kaiming He's recent presentation)

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

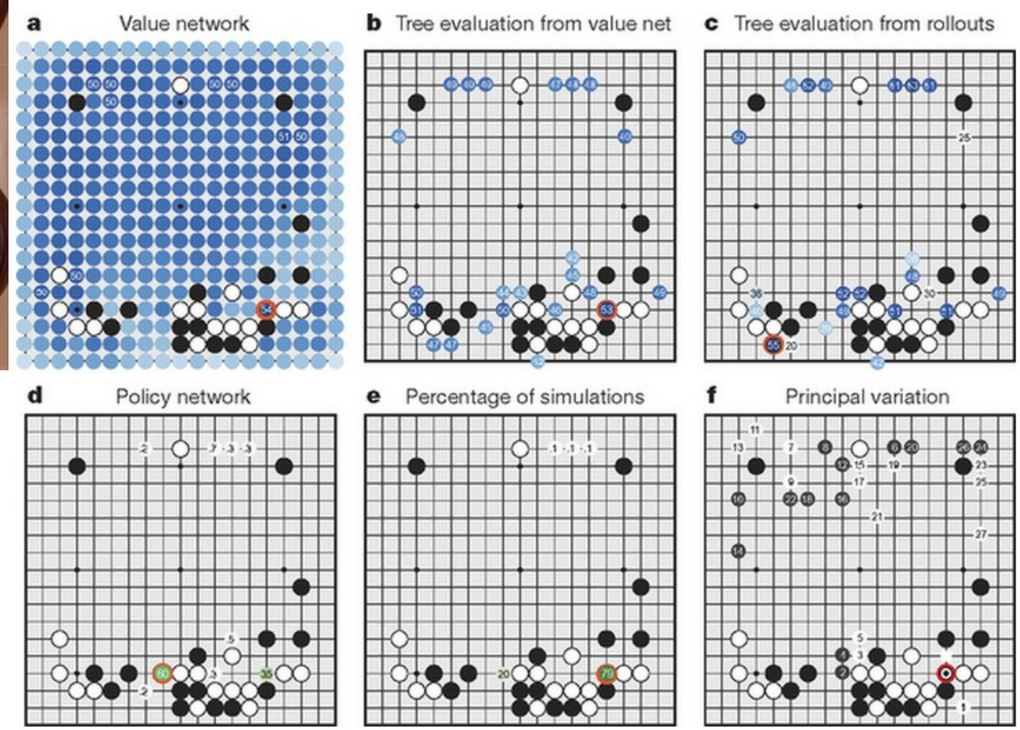
Case Study: ResNet *[He et al., 2015]*



spatial dimension
only 56x56!

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Case Study Bonus: DeepMind's AlphaGo



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like

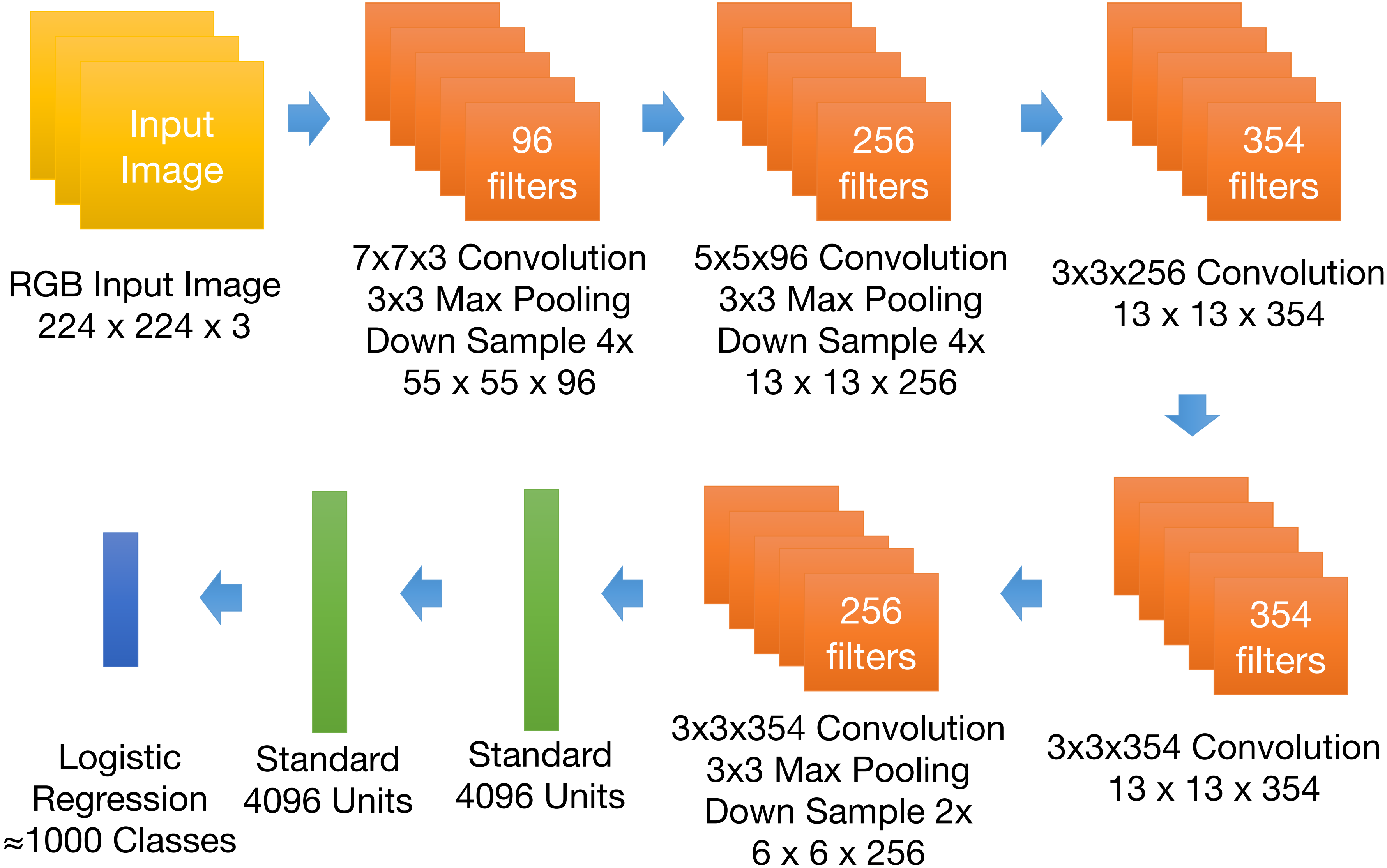
  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX

  where N is usually up to ~5, M is large, 0 <= K <= 2.

  - but recent advances such as ResNet/GoogLeNet challenge this paradigm

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Understanding ConvNets

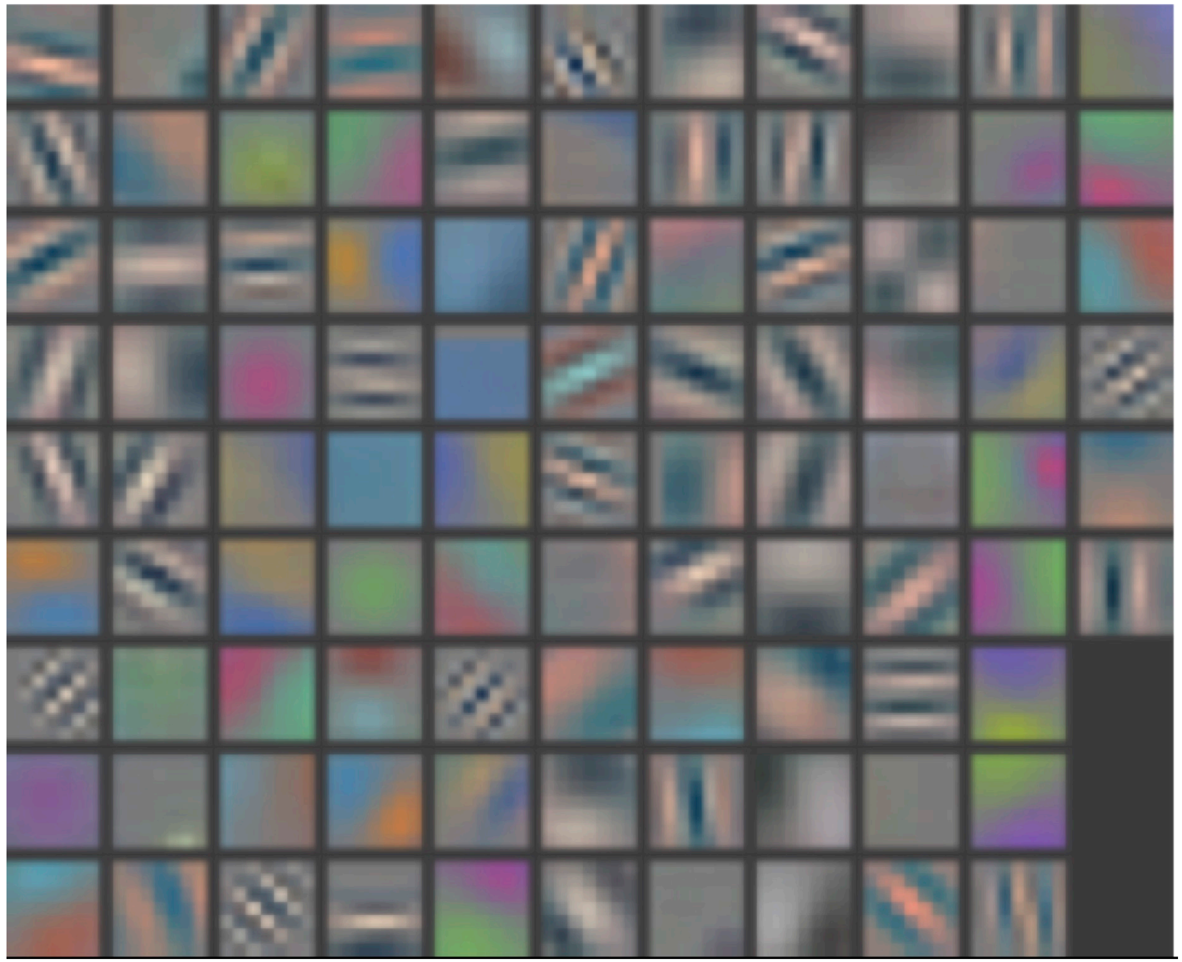


http://www.image-net.org/

http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

slide by Yisong Yue

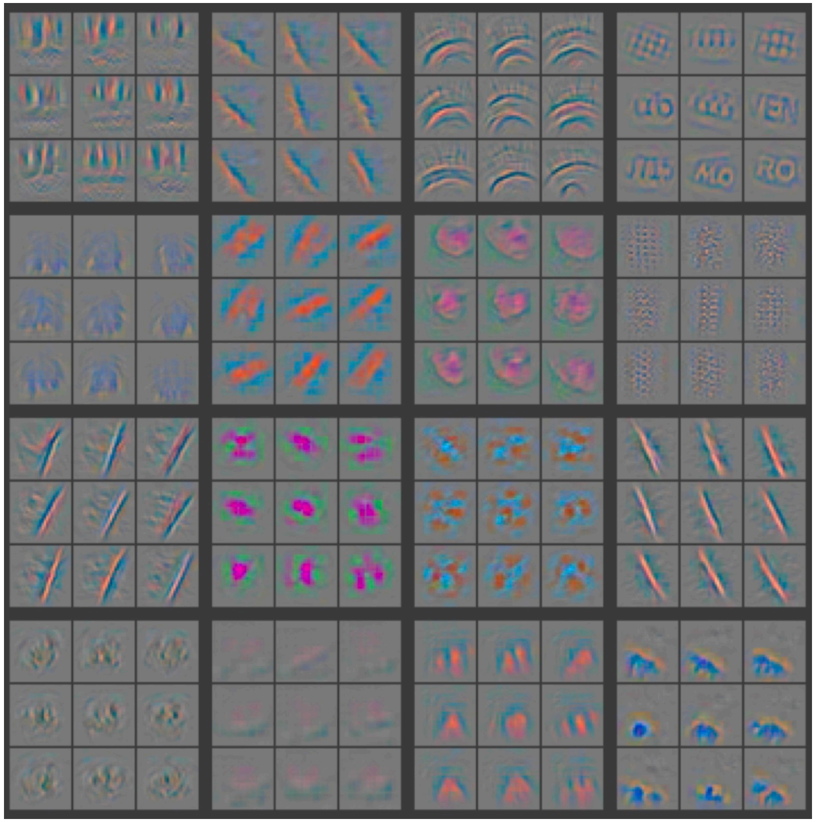
Visualizing CNN (Layer 1)



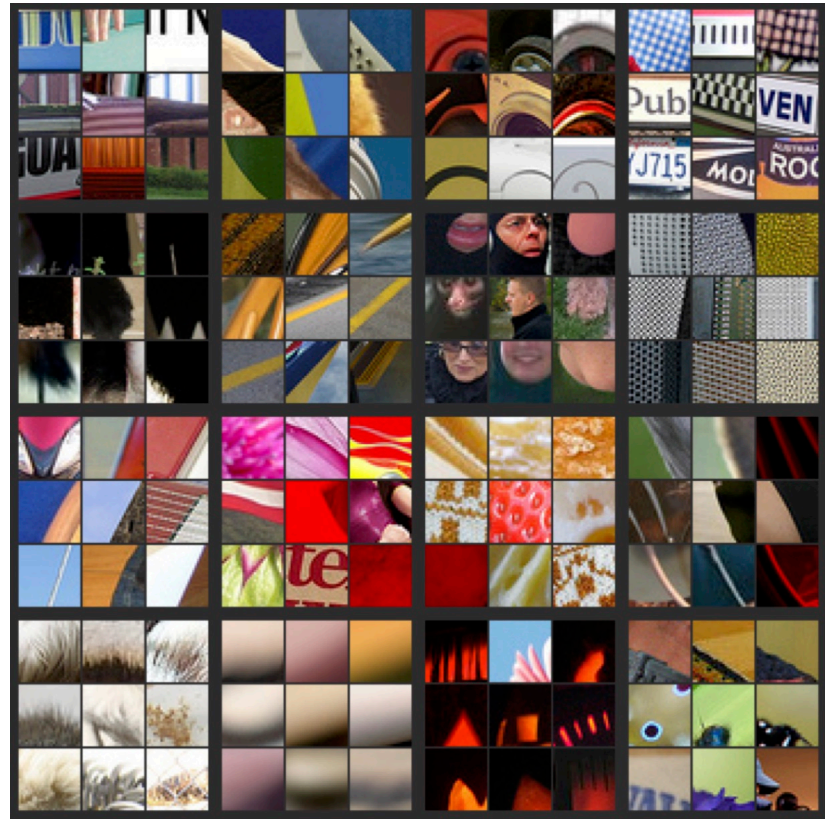
http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

slide by Yisong Yue

Visualizing CNN (Layer 2)



Part that Triggered Filter

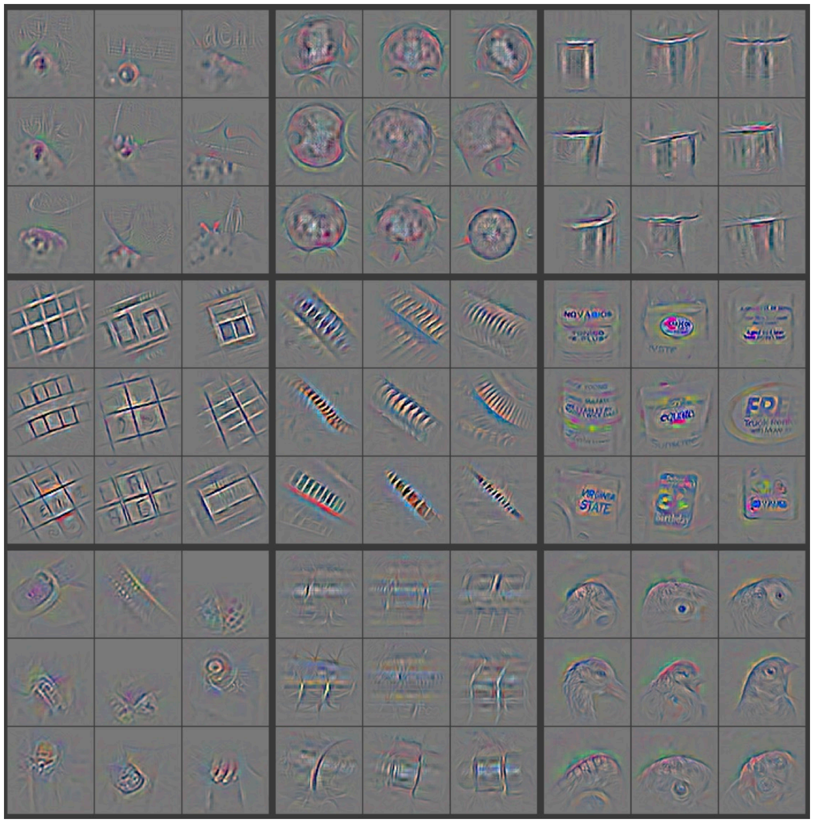


Top Image Patches

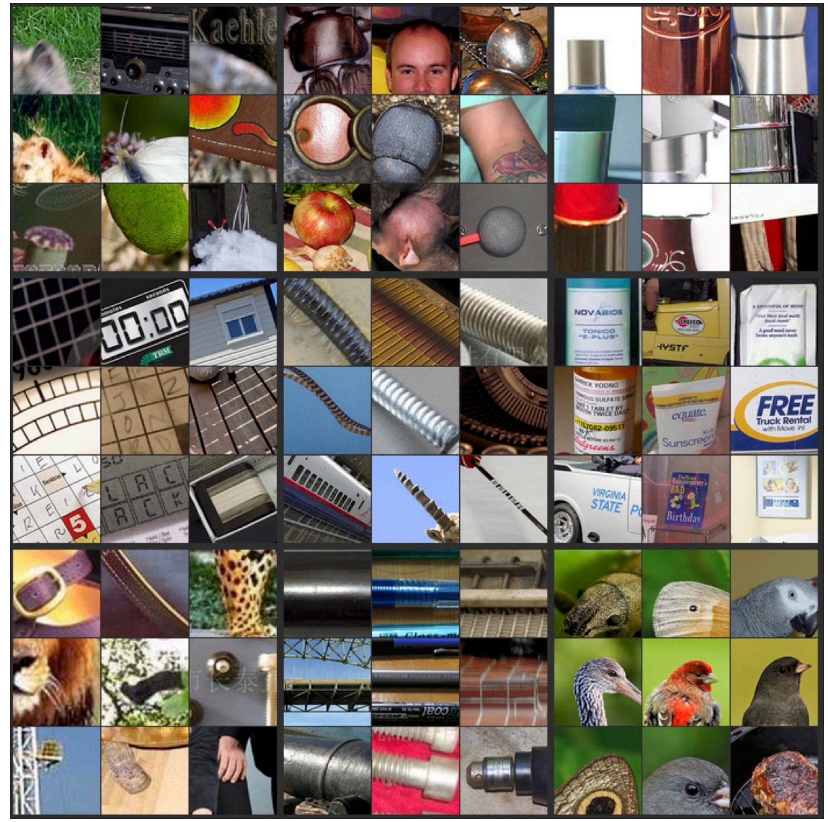
http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

slide by Yisong Yue

Visualizing CNN (Layer 3)



Part that Triggered Filter

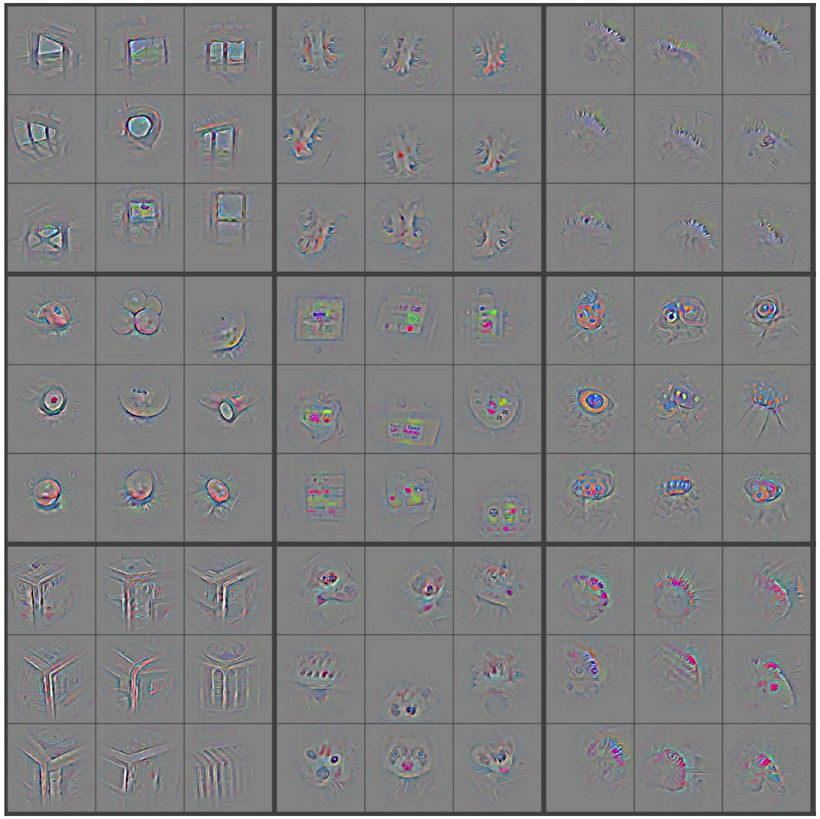


Top Image Patches

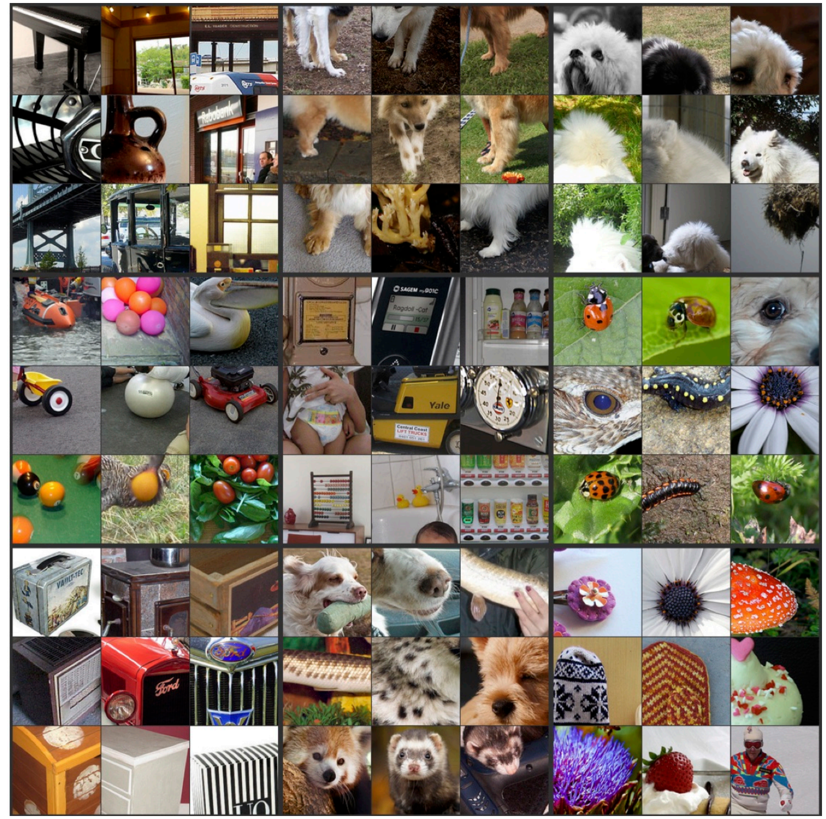
slide by Yisong Yue

http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

Visualizing CNN (Layer 4)



Part that Triggered Filter

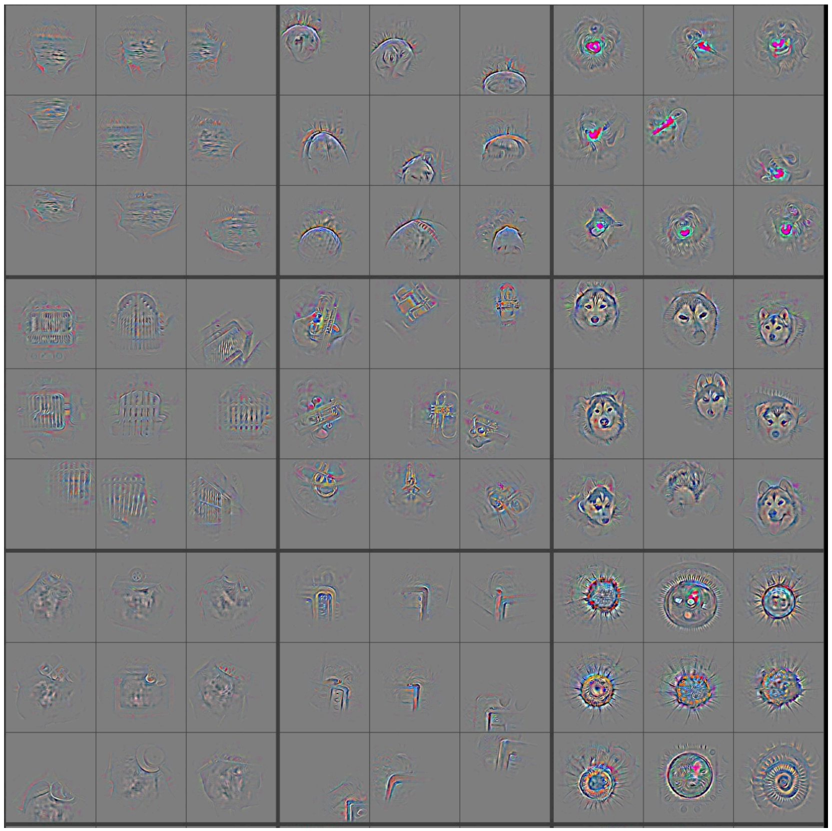


Top Image Patches

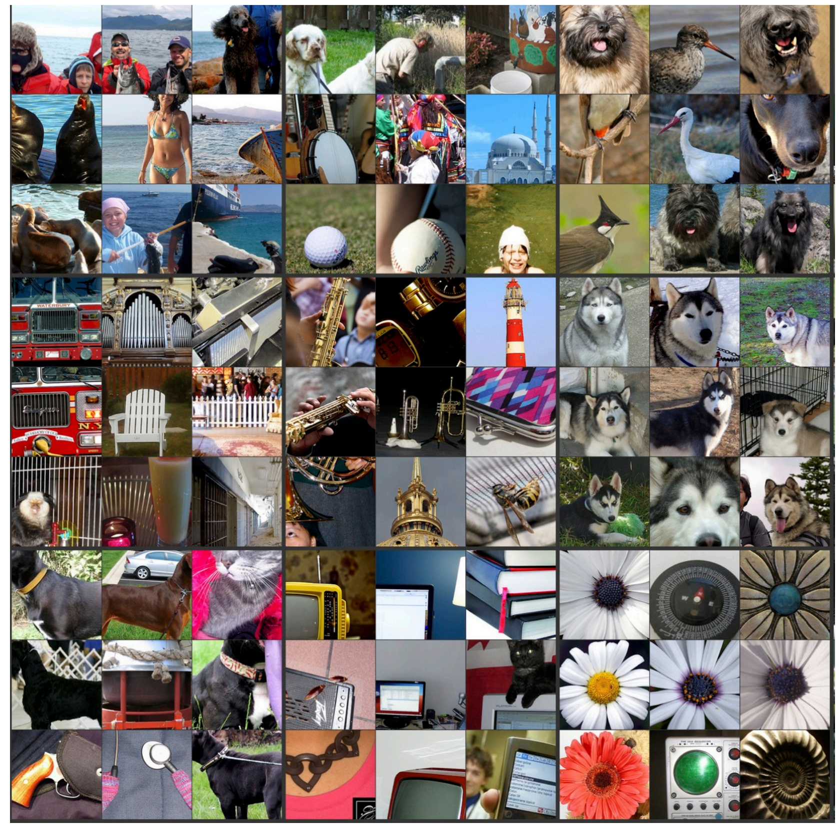
http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

slide by Yisong Yue

Visualizing CNN (Layer 5)



Part that Triggered Filter



Top Image Patches

http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

slide by Yisong Yue

Deep Visualization Toolbox

yosinski.com/deepvis

#deepvis


Jason Yosinski


Jeff Clune


Anh Nguyen


Thomas Fuchs


Hod Lipson

Cornell University

University of Wyoming

Jet Propulsion Laboratory
California Institute of Technology

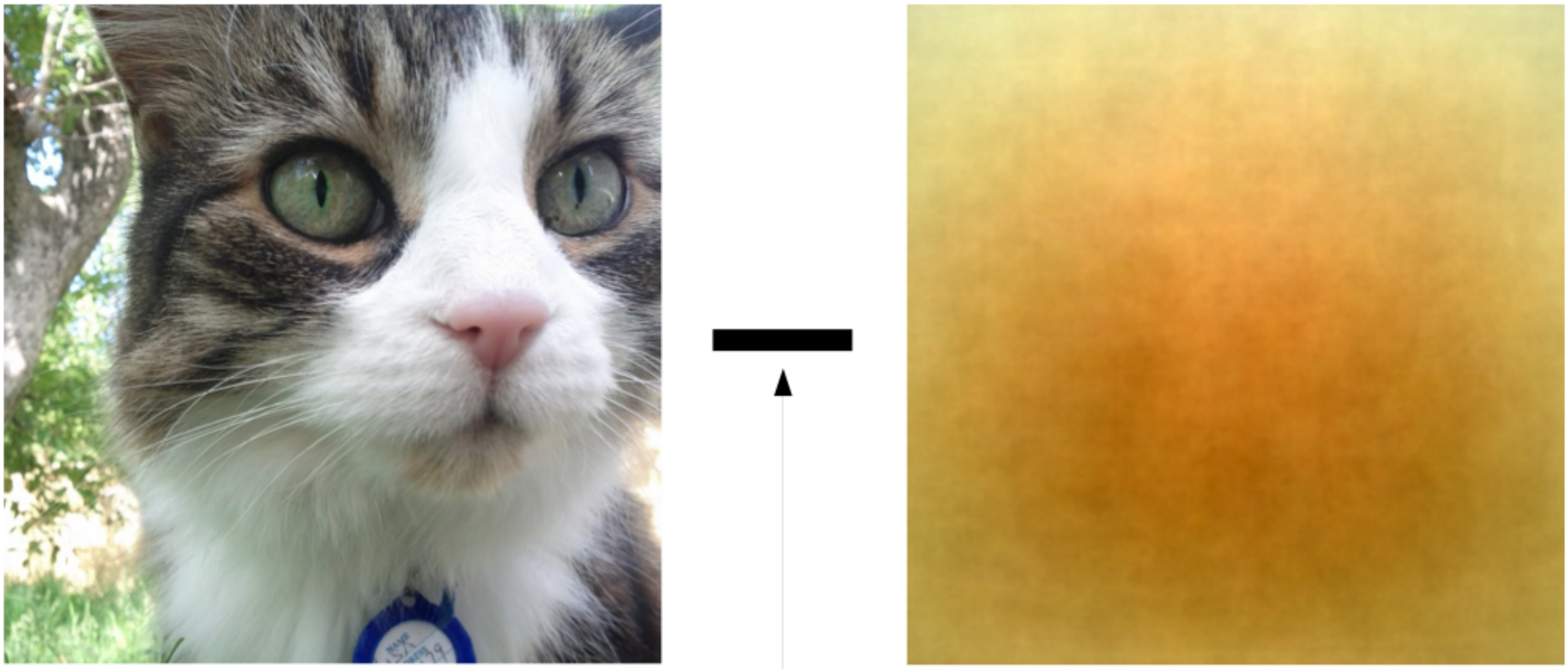
Tips and Tricks

- Shuffle the training samples
- Use Dropoout and Batch Normalization for regularization

Input representation

"Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image"

- Centered (0-mean) RGB values.



An input image (256x256) Minus sign The mean input image

slide by Alex Krizhevsky

Data Augmentation

- The neural net has 60M real-valued parameters and 650,000 neurons
- It overfits a lot. Therefore, they train on 224x224 patches extracted randomly from 256x256 images, and also their horizontal reflections.

"This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly inter- dependent."

[Krizhevsky et al. 2012]

slide by Alex Krizhevsky

Data Augmentation

- Alter the intensities of the RGB channels in training images.

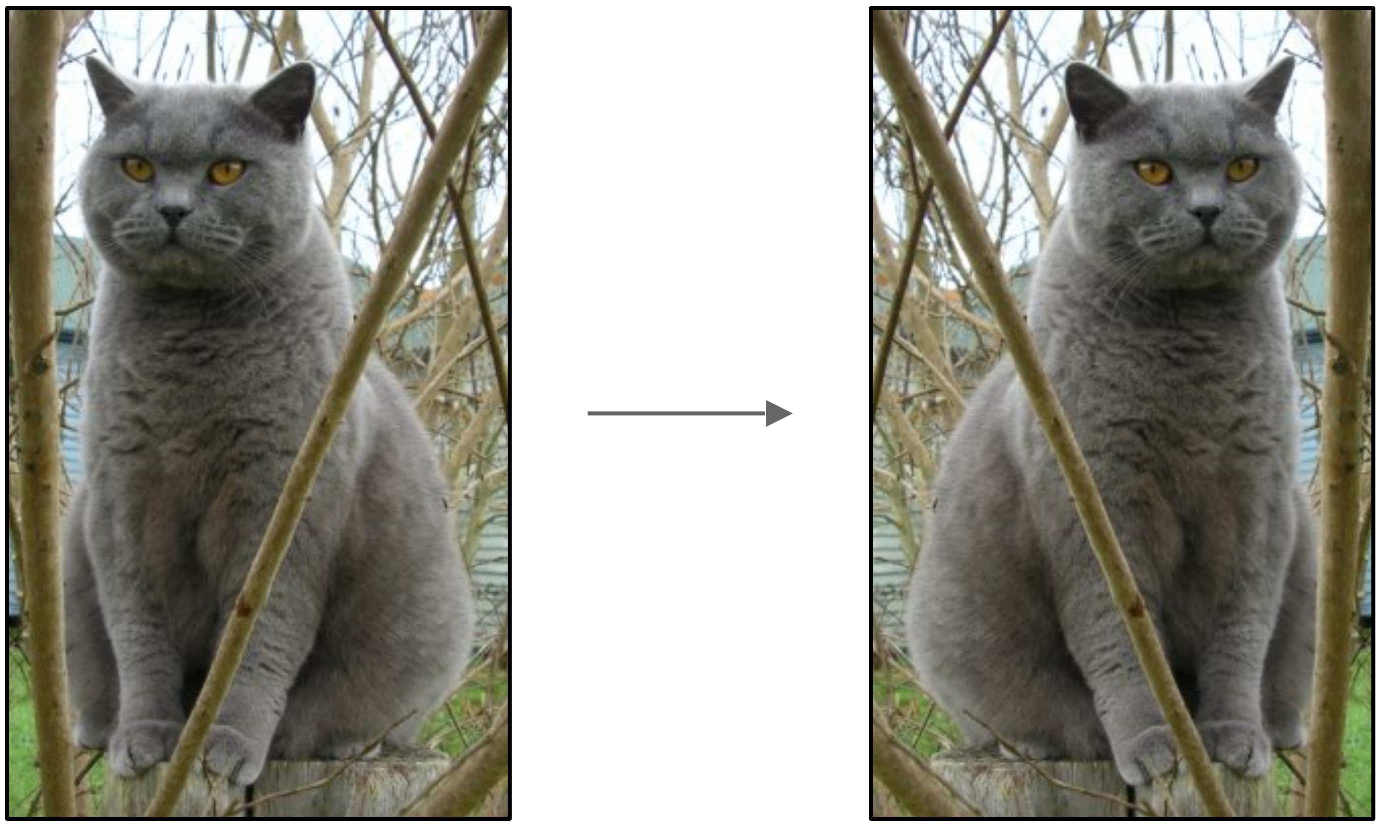
  "Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, with magnitudes proportional to the corres. ponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1…This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%."

[Krizhevsky et al. 2012]

slide by Alex Krizhevsky

Data Augmentation

Horizontal flips



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Data Augmentation

Get creative!

Random mix/combinations of :
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

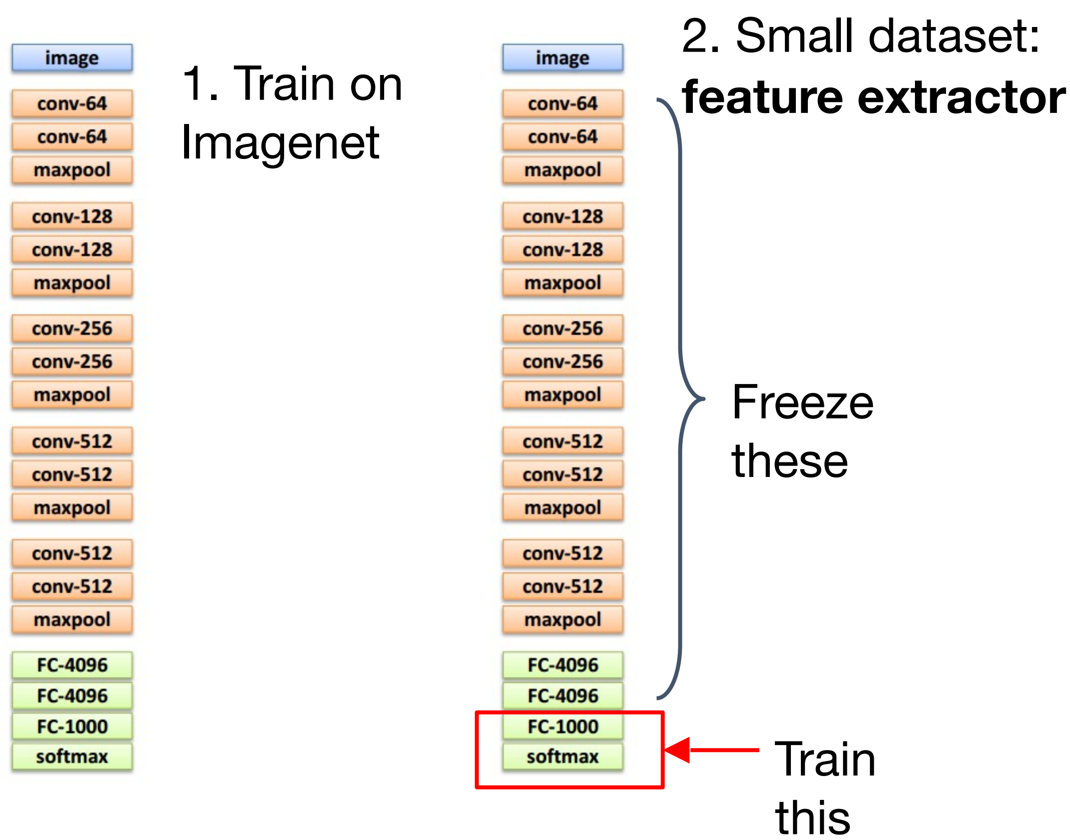
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Transfer Learning with ConvNets

1. Train on Imagenet

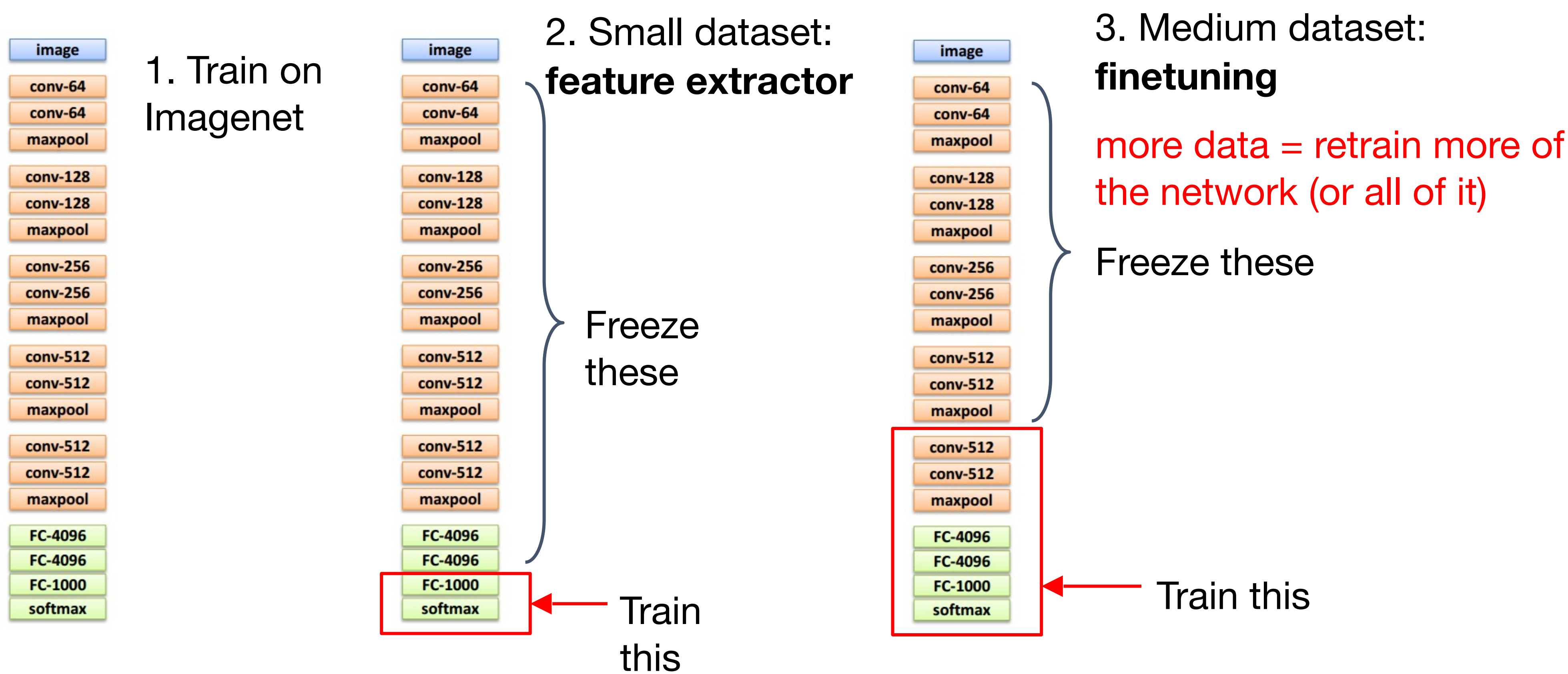
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Transfer Learning with ConvNets



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

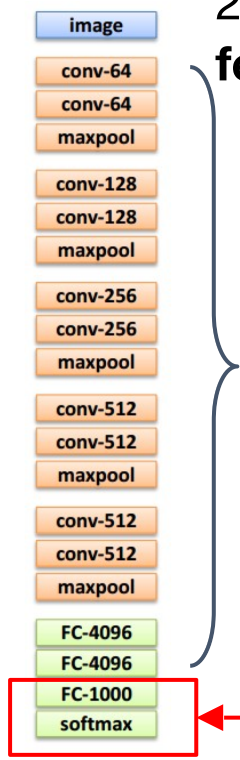
Transfer Learning with ConvNets



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Transfer Learning with ConvNets

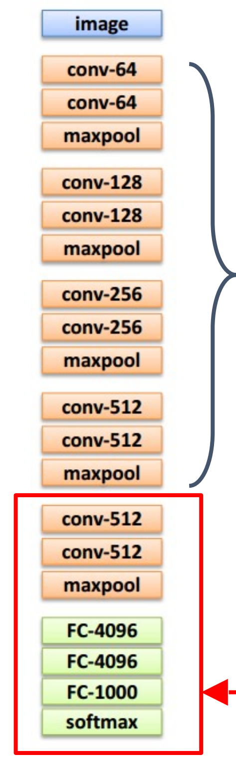
1. Train on Imagenet



2. Small dataset: **feature extractor**

Freeze these

Train this



3. Medium dataset: **finetuning**

more data = retrain more of the network (or all of it)

Freeze these

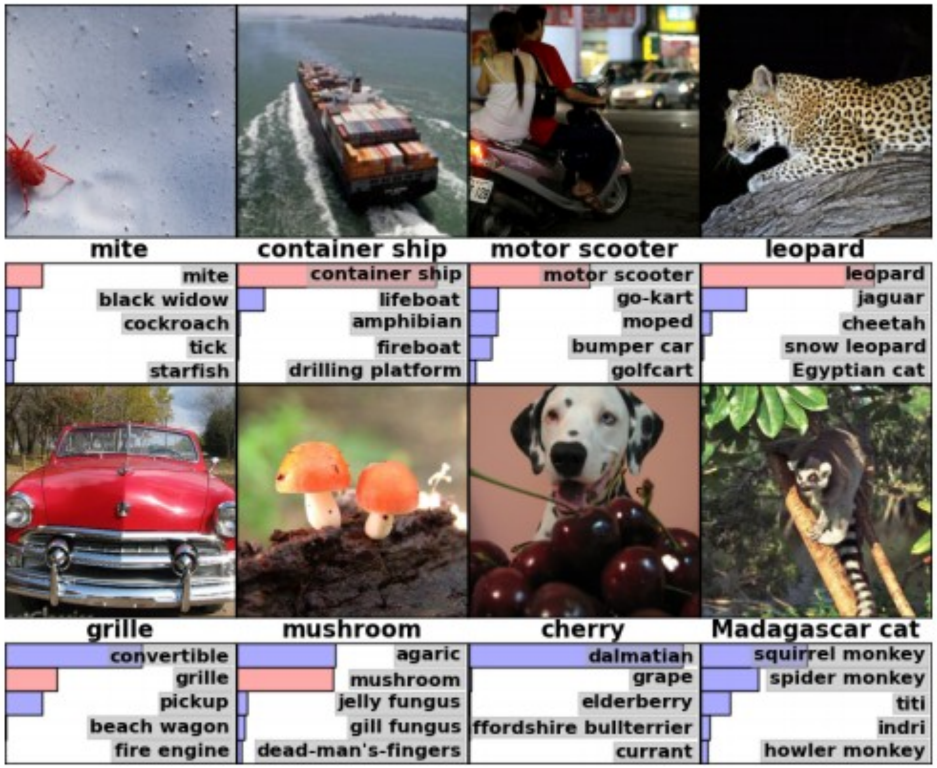
tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers

Train this

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Today ConvNets are everywhere

Classification



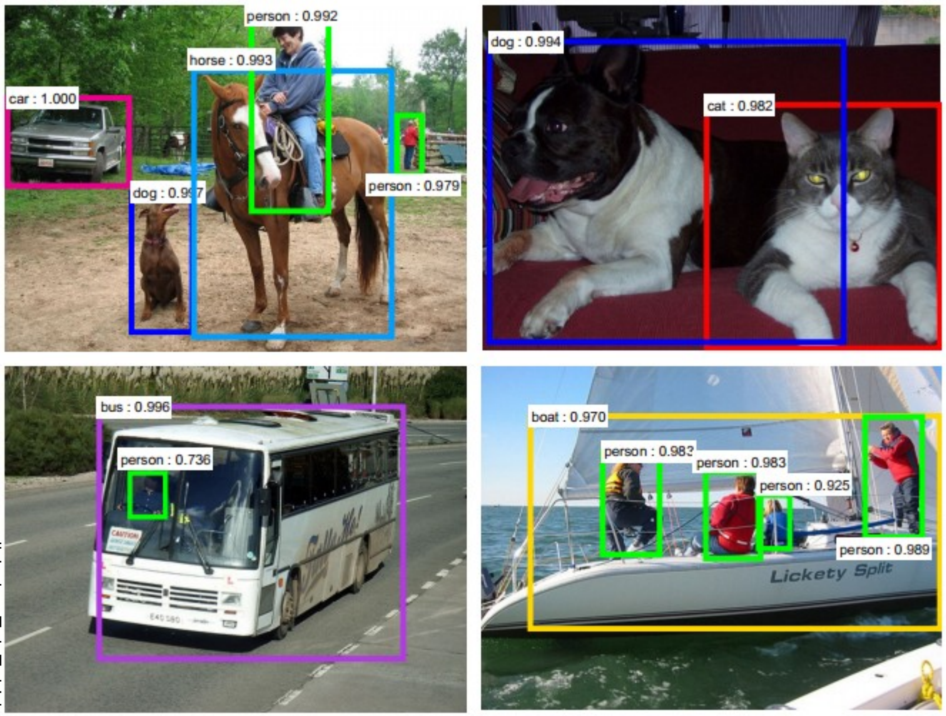
Retrieval

[Krizhevsky 2012]

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

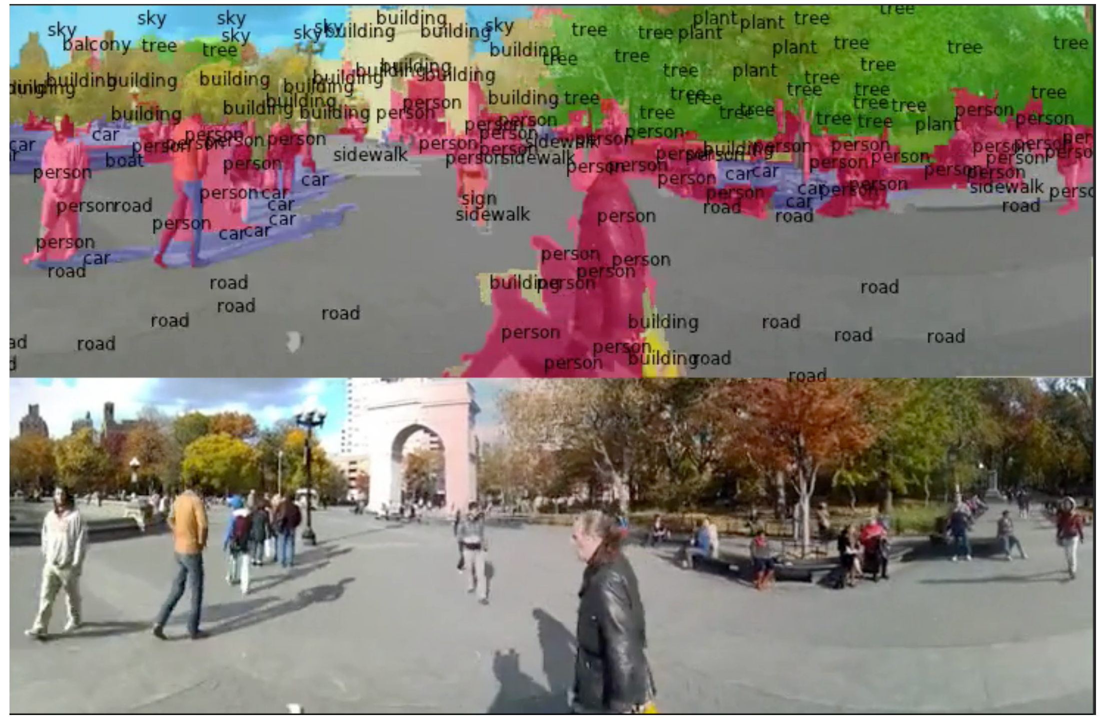
Today ConvNets are everywhere

Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

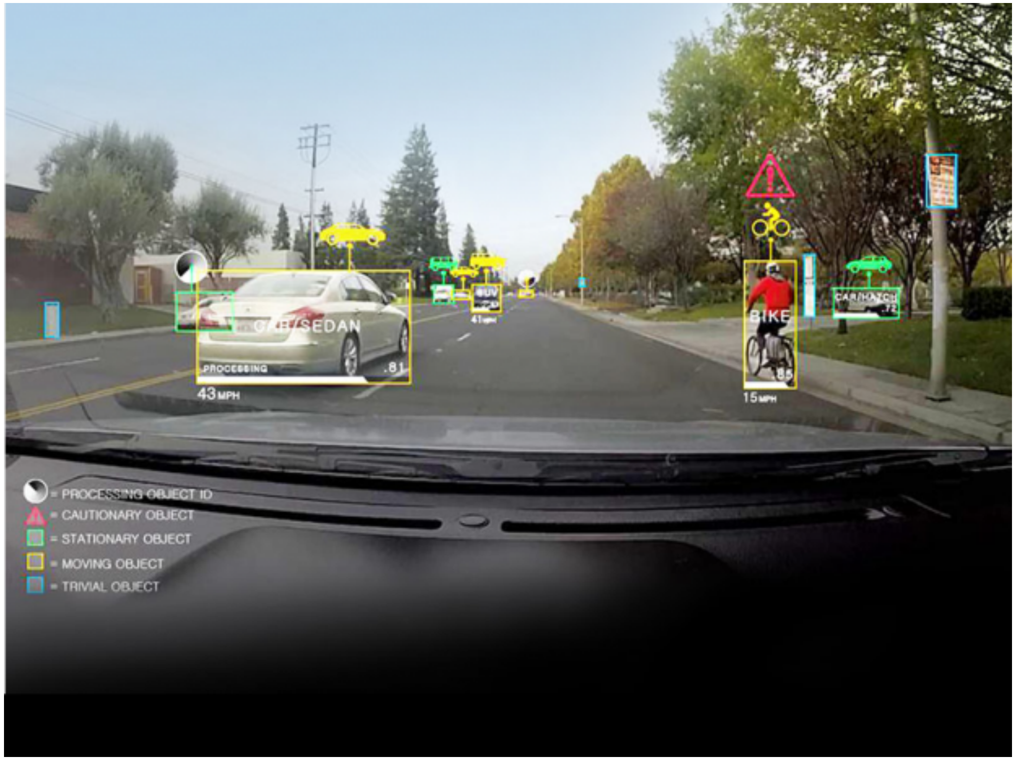
Segmentation



[Farabet et al., 2012]

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Today ConvNets are everywhere



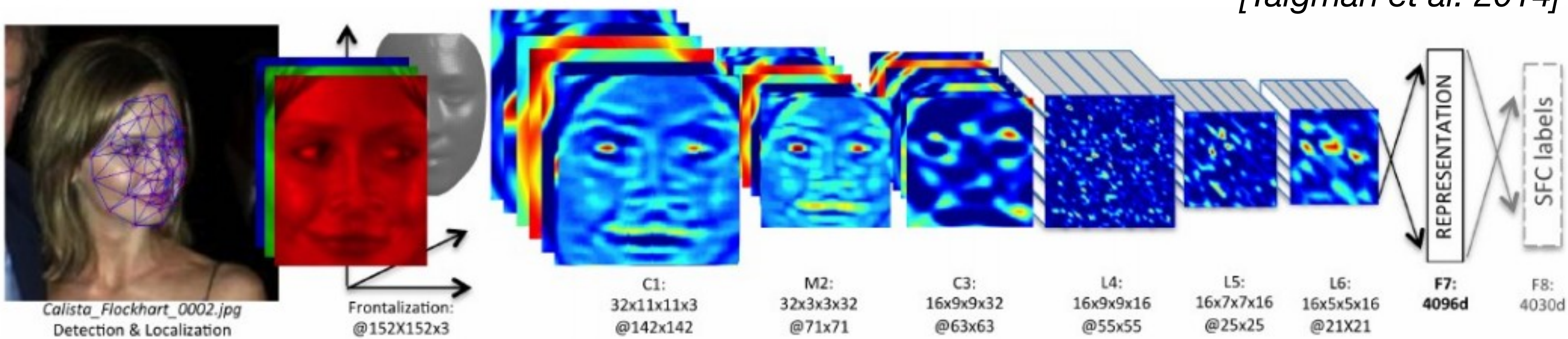
self-driving cars

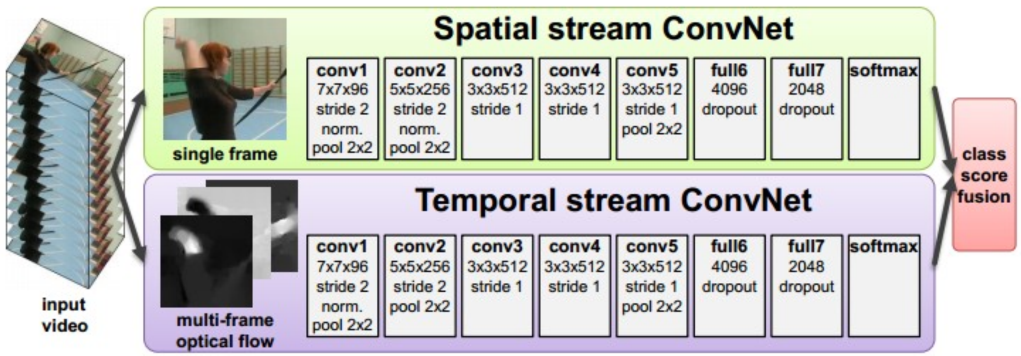
NVIDIA Tegra X1

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

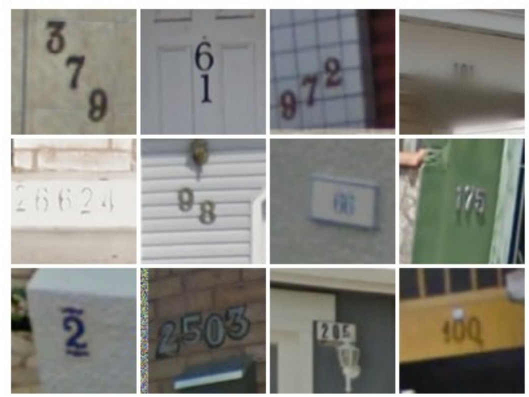
Today ConvNets are everywhere

[Taigman et al. 2014]





[Simonyan et al. 2014]



[Goodfellow 2014]

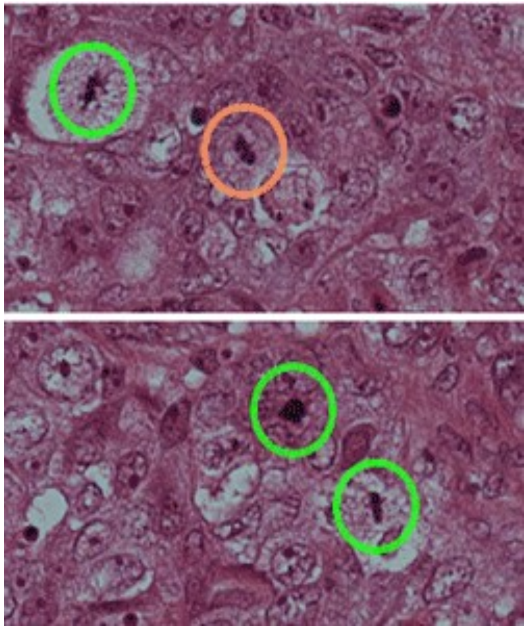
slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Today ConvNets are everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]

Today ConvNets are everywhere

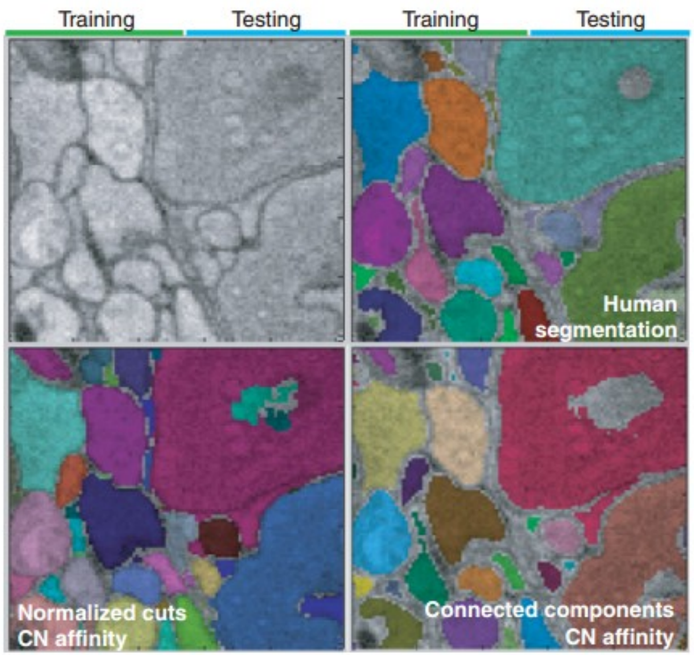


[Ciresan et al. 2013]

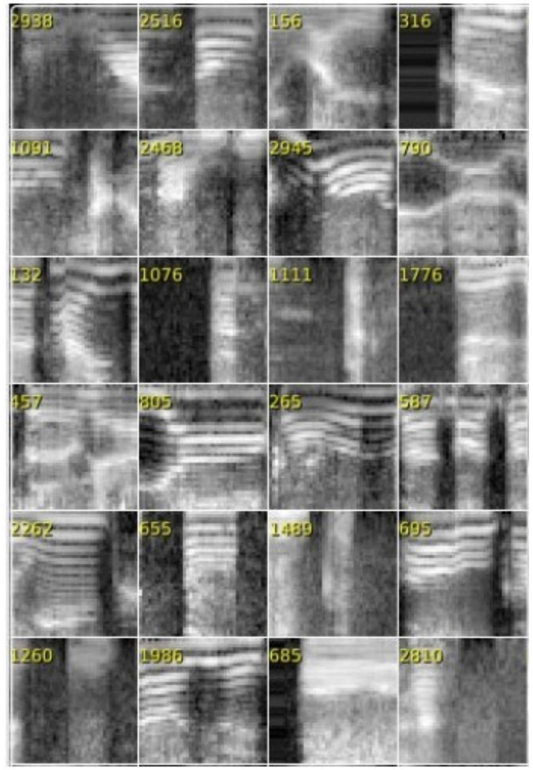
[Sermanet et al. 2011]
[Ciresan et al.]

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Today ConvNets are everywhere



[Turaga et al., 2010]



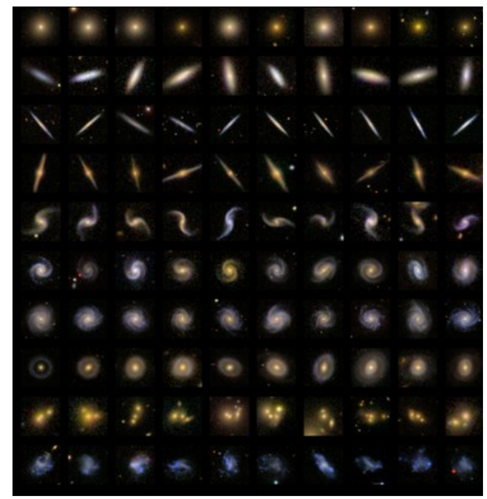
I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. **Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck hybrid decides to play cat-and-mouse with them.** Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the horror/slasher buff. **Delinitely worth checking out.**

I just saw this on a local independent station in the New York City area. **The cast showed promise but when I saw the director, George Cosmotos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmotos movie I ever saw.** He's like a stupid man's Michael Bey – with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirators on. We are left to ourselves to connect the dots from one bit of graffiti on various walls in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extremism, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

Graphics is far from the best part of the game. **This is the number one best TH game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is the kind of money that is wasted properly.** And even though graphics suck, thats doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WHO CARES? As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS3) Well, I don't know if they say that, but they might. who knows. Well, Canadian people do. Wait a minute, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it . What really makes this movie work is Judd Nelson's character and the sometimes clever script. **A pretty good script for a person who wrote the Final Destination films and the direction was okay.** Sometimes there's scenes where it looks like it was filmed using a home video camera with a grainy - look. Great made - for - TV movie. **It was worth the rental and probably worth buying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best.** I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.

[Denil et al. 2014]



slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Today ConvNets are everywhere

Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010

Today ConvNets are everywhere

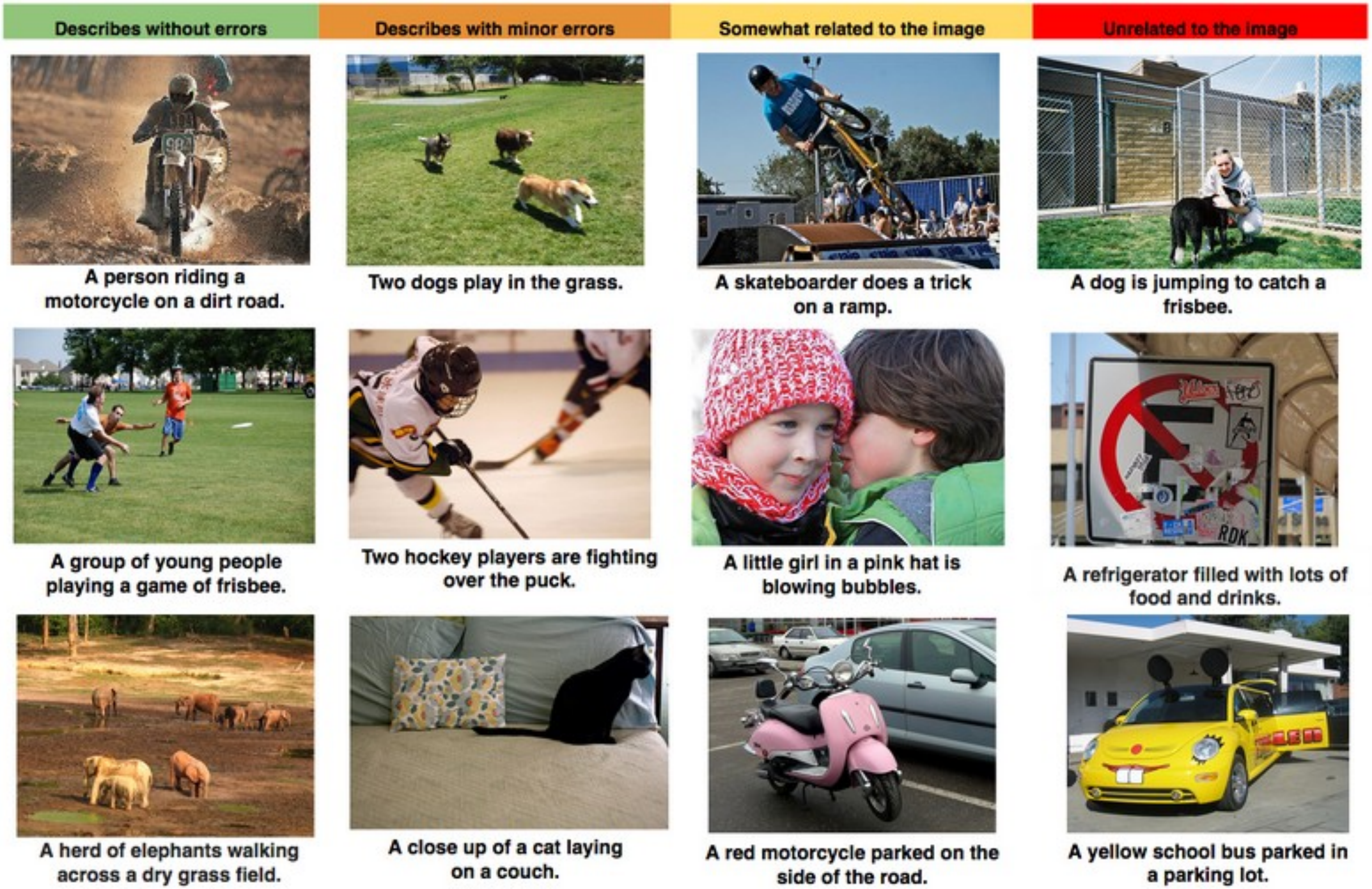


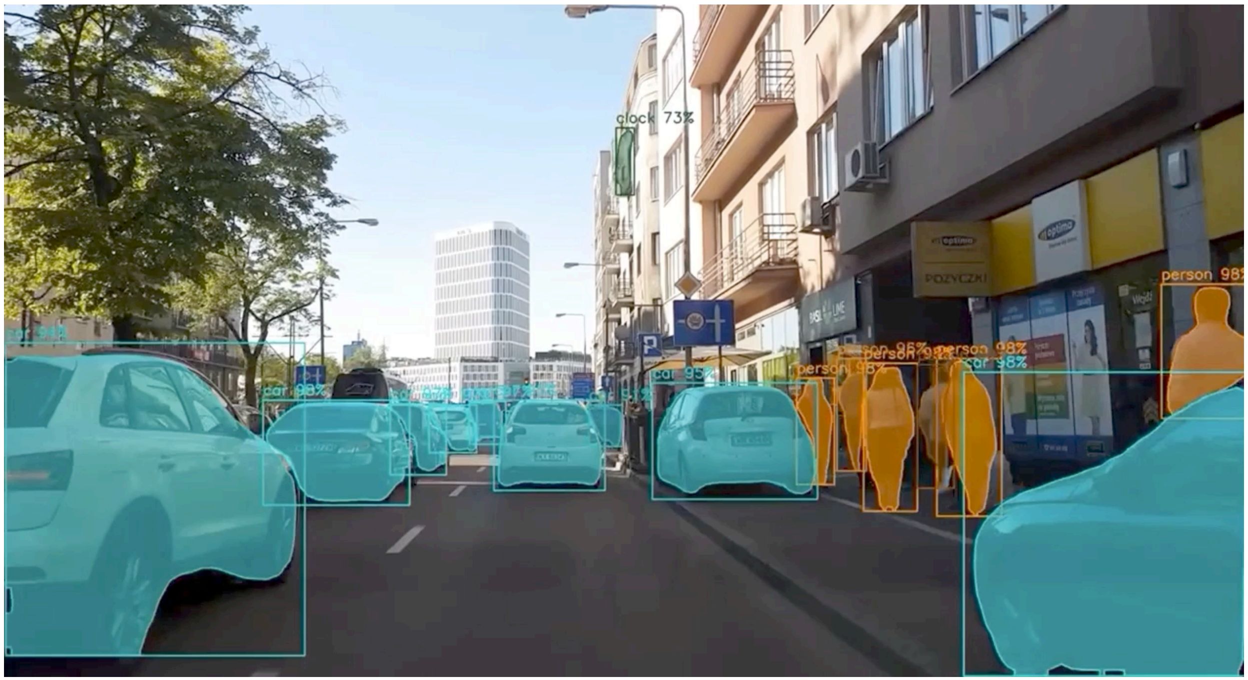
Image Captioning

[Vinyals et al., 2015]

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnso

Today ConvNets are everywhere

Detection + Segmentation = Semantic Segmentation



self-driving cars

[Mask R-CNN: He, Gkioxari, Dollár, Girshick 2017]

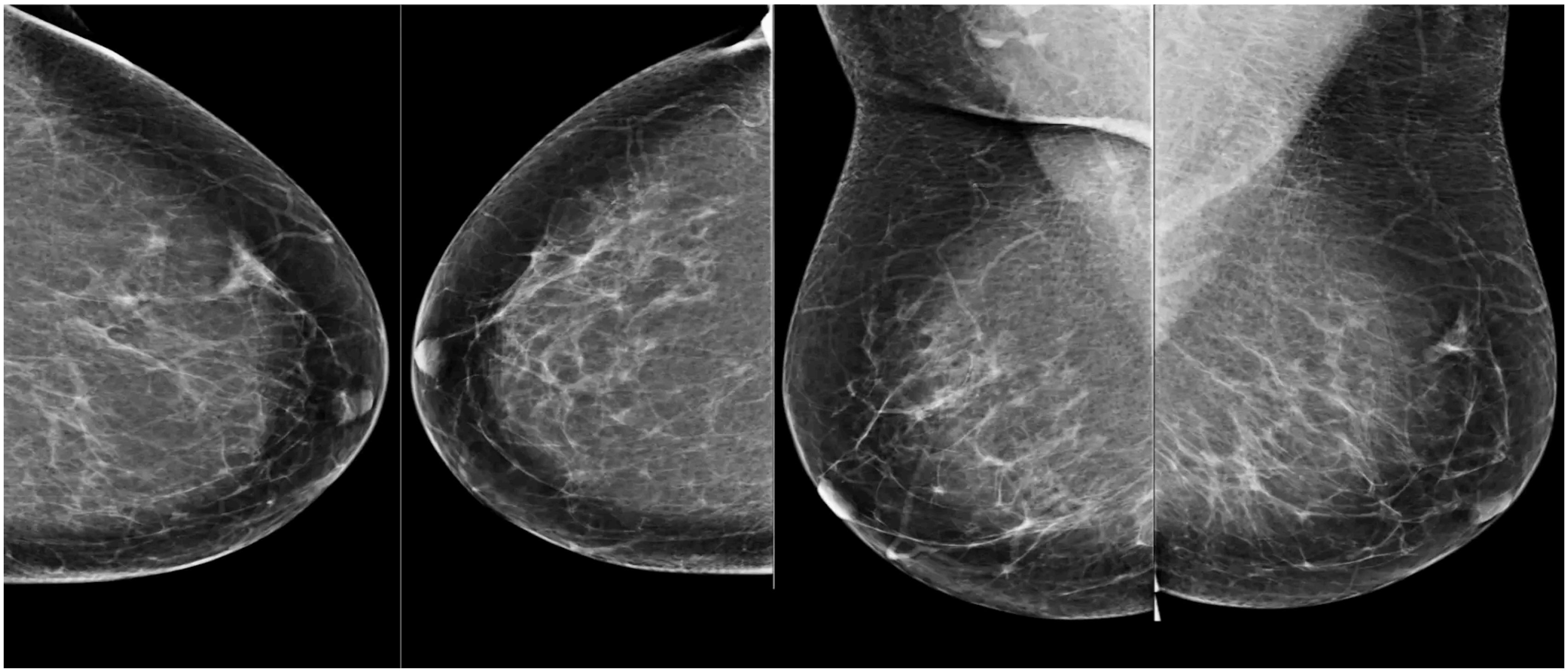
Today ConvNets are everywhere

Pose estimation

[DensePose: Güler, Neverova, Kokkinos 2018]

Today ConvNets are everywhere

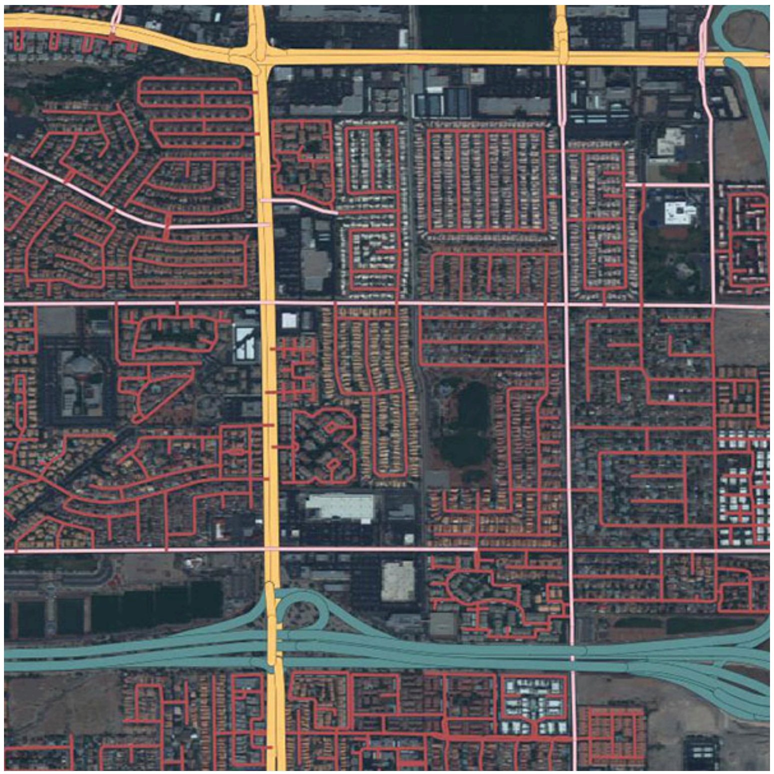
Cancer detection

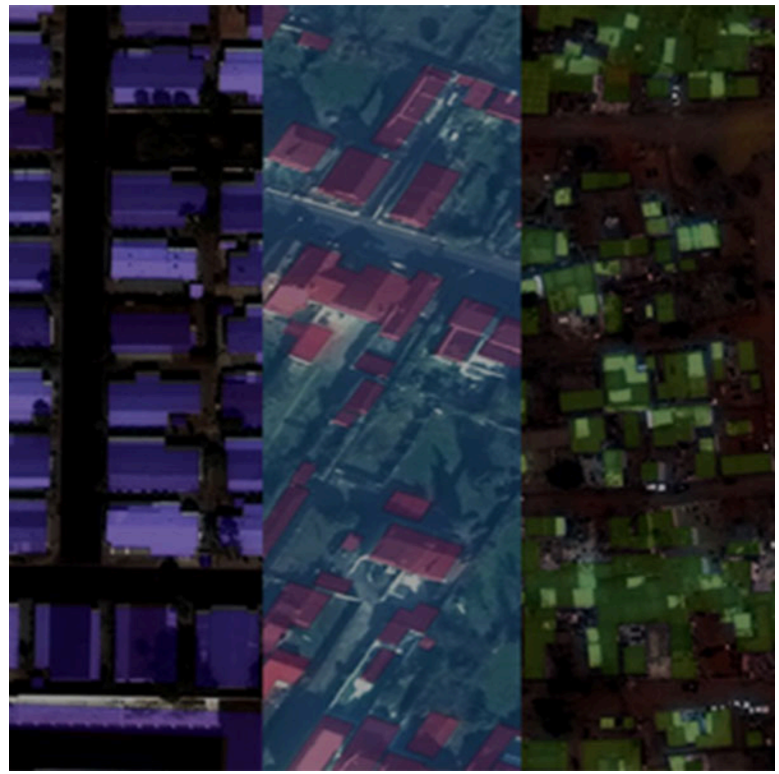


[McKinney et al. 2020]

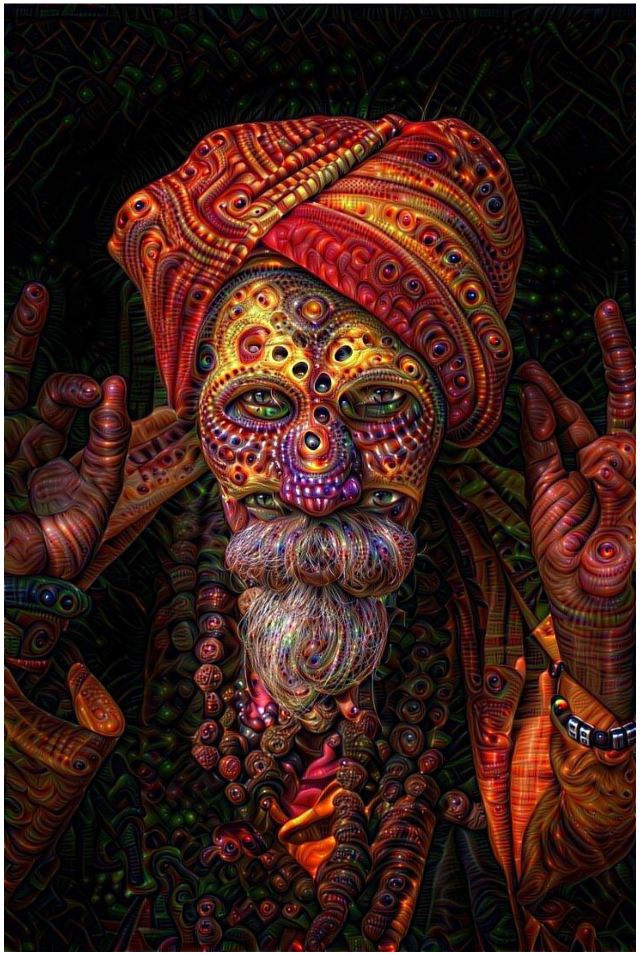
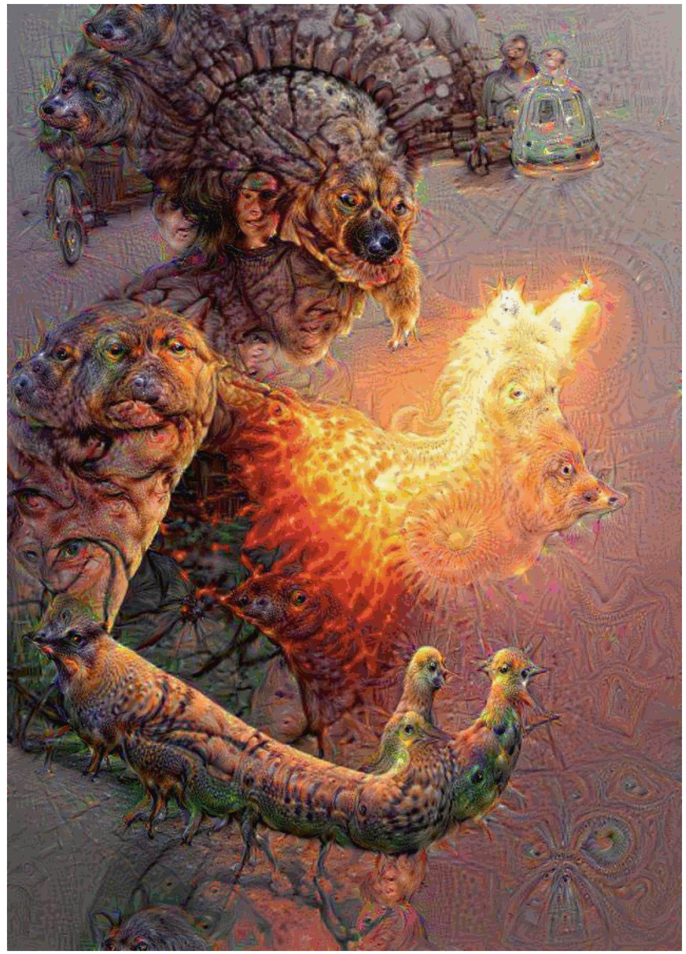
Today ConvNets are everywhere

Open challenges for remote sensing





Today ConvNets are everywhere

reddit.com/r/deepdream

slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson

Next Lecture:

Support Vector Machines