CS 1678: Intro to Deep Learning

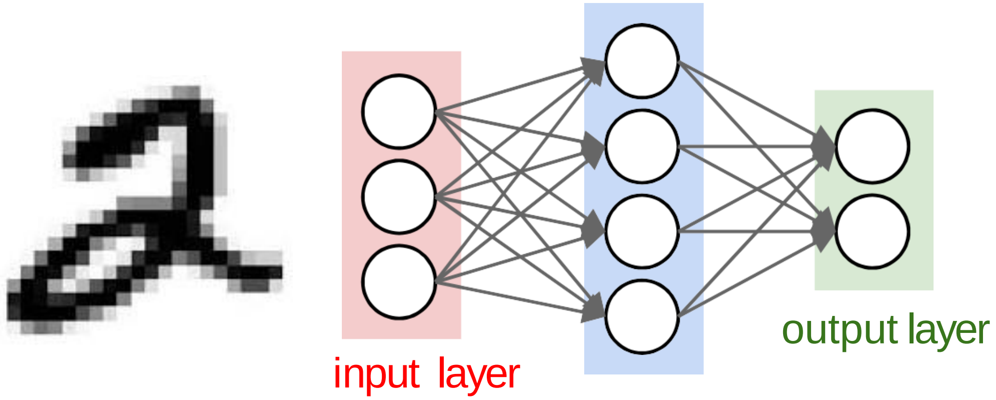
Convolutional Neural Networks

Prof. Adriana Kovashka
University of Pittsburgh
March 2, 2021

Plan for this lecture

- Motivation: Scanning for patterns
- Convolutional network operations
- Common architectures
- Visualizing convolutional networks
- Applications in computer vision

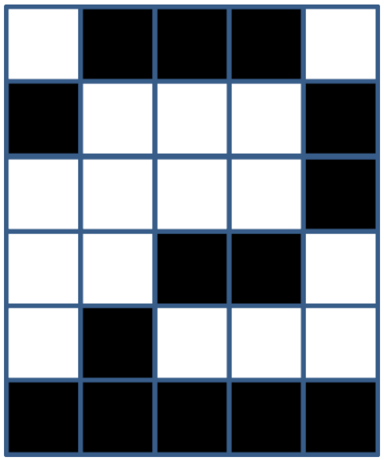
Neural networks so far



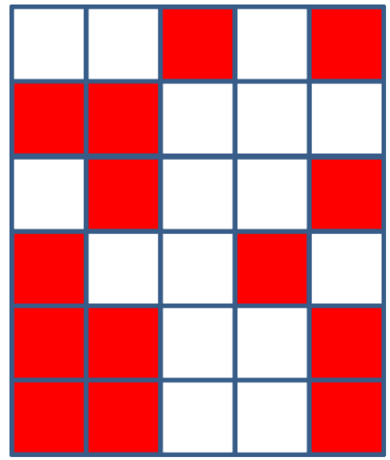
- Can recognize patterns in data (e.g. digits)

Adapted from Bhiksha Raj

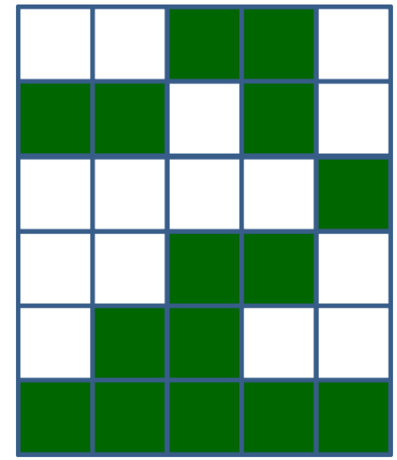
The weights look for patterns



$$y = \begin{cases} 1 \text{ if } \sum w_i x_i \geq T \\ 0 \text{ else} \end{cases}$$



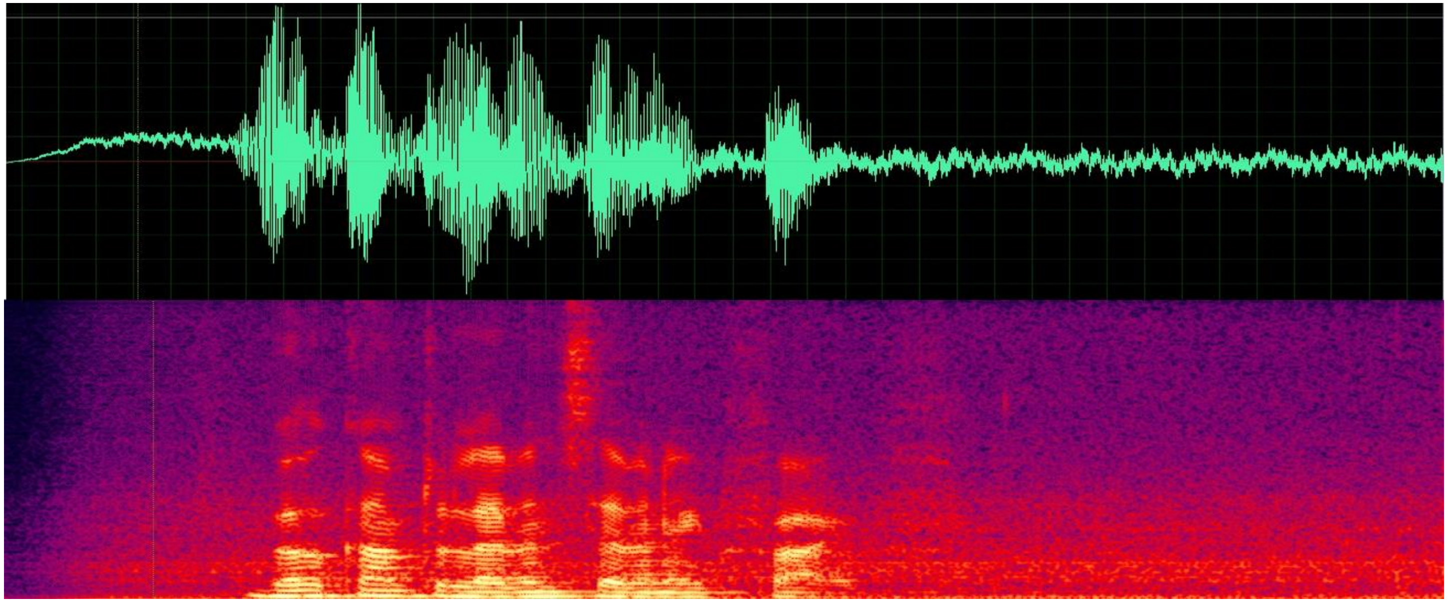
Correlation = 0.57



Correlation = 0.82

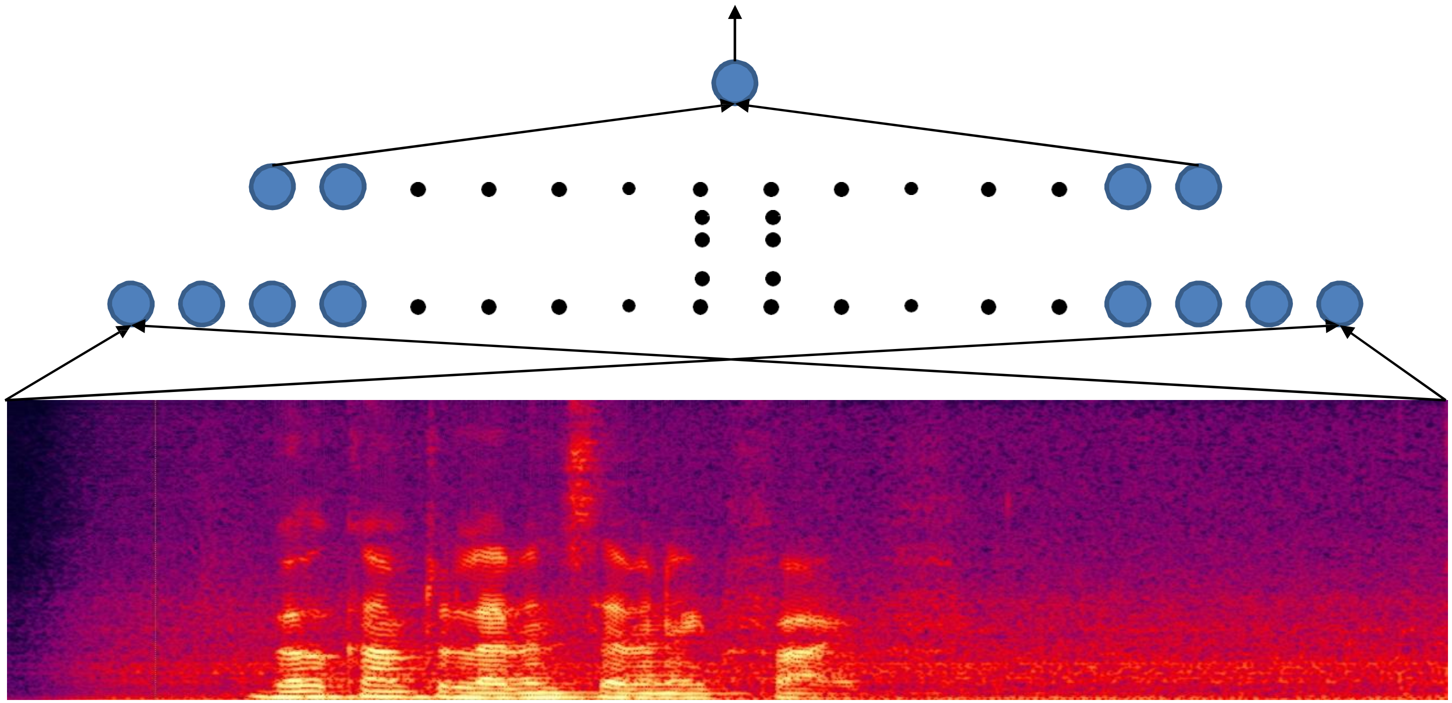
- The green pattern looks more like the weights pattern (black) than the red pattern
  - The green pattern is more *correlated* with the weights

A problem



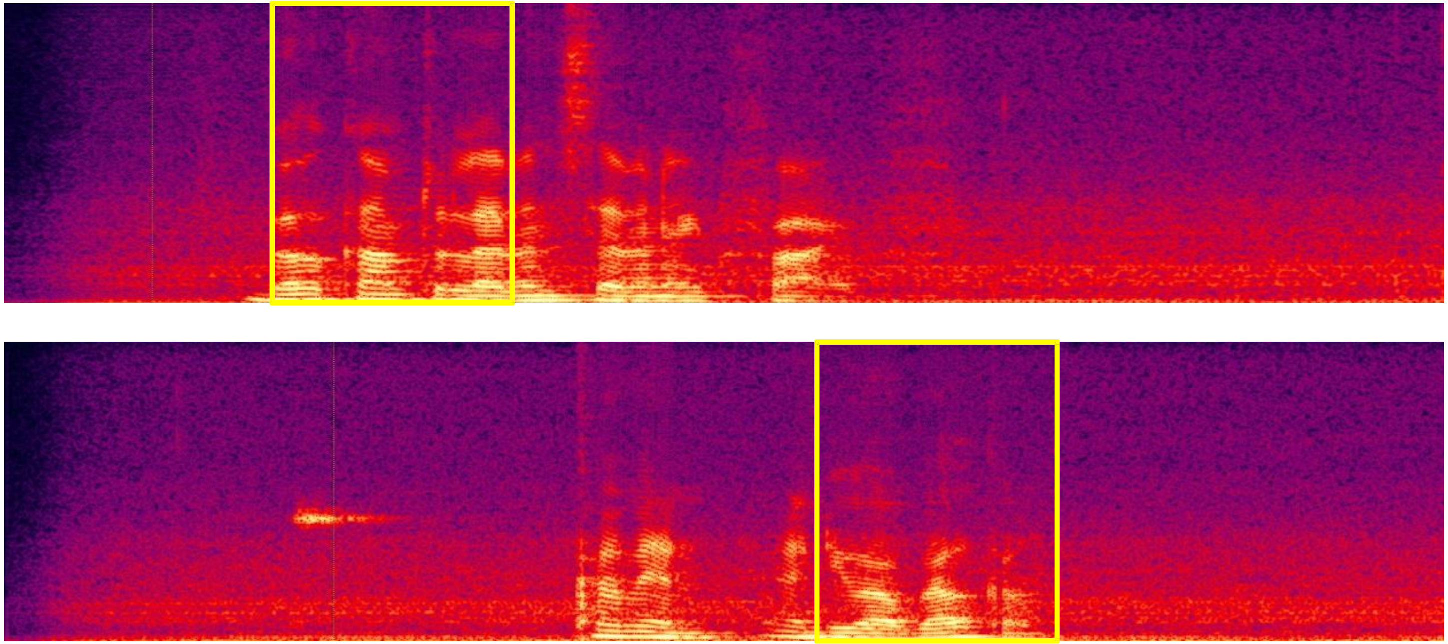
- Does this signal contain the word “Welcome”?
- Compose a NN for this problem
  - Assuming all recordings are exactly the same length

Finding a Welcome



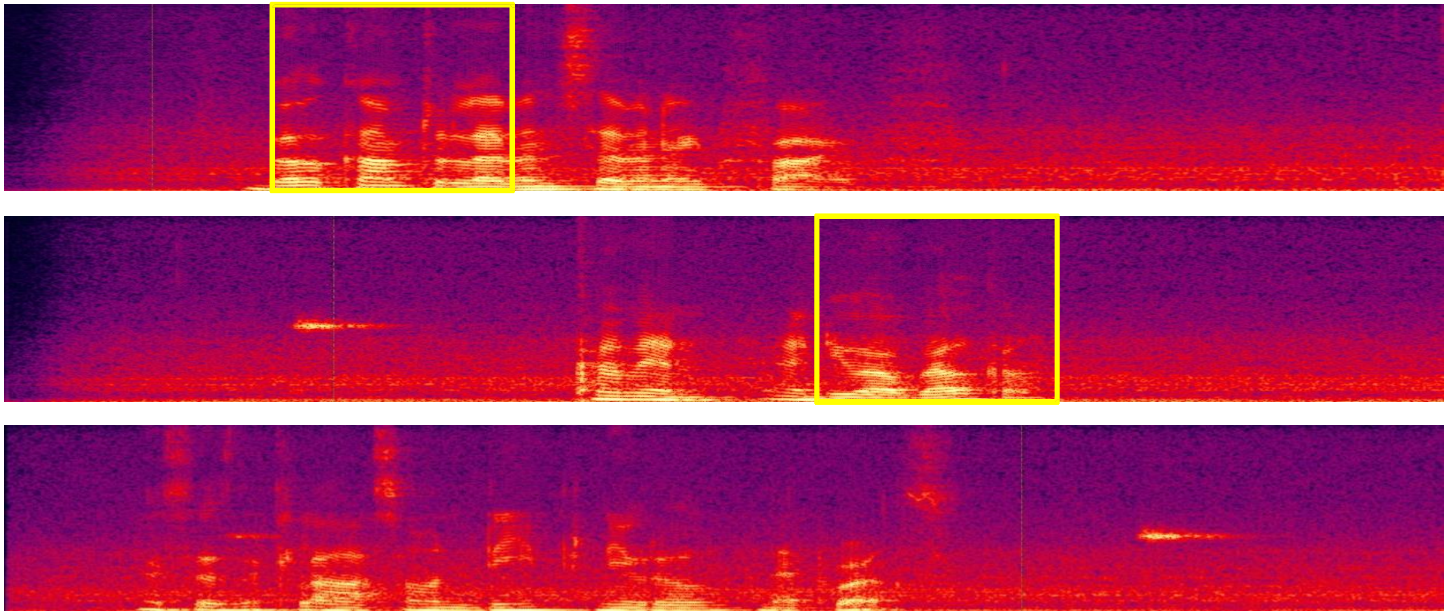
- Trivial solution:Train a NN for the entire recording

Finding a Welcome



- Problem with trivial solution: Network that finds a "welcome" in the top recording will not find it in the lower one
  - Unless trained with both
  - Will require a very large network and a large amount of training data to cover every case

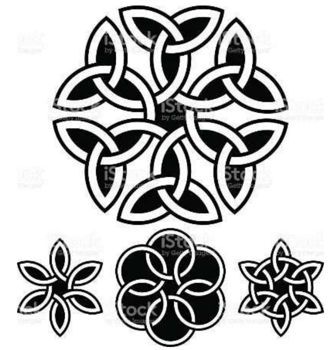
Finding a Welcome

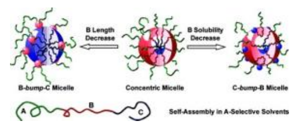


- Need a *simple* network that will fire regardless of the location of "Welcome"
  - and not fire when there is none

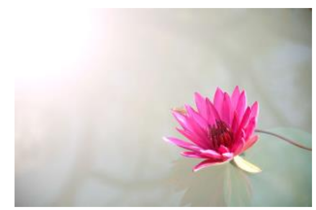
Flower





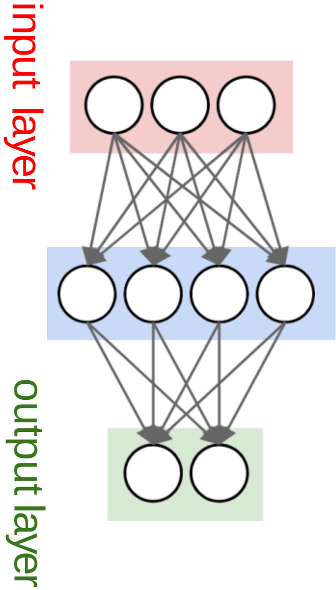







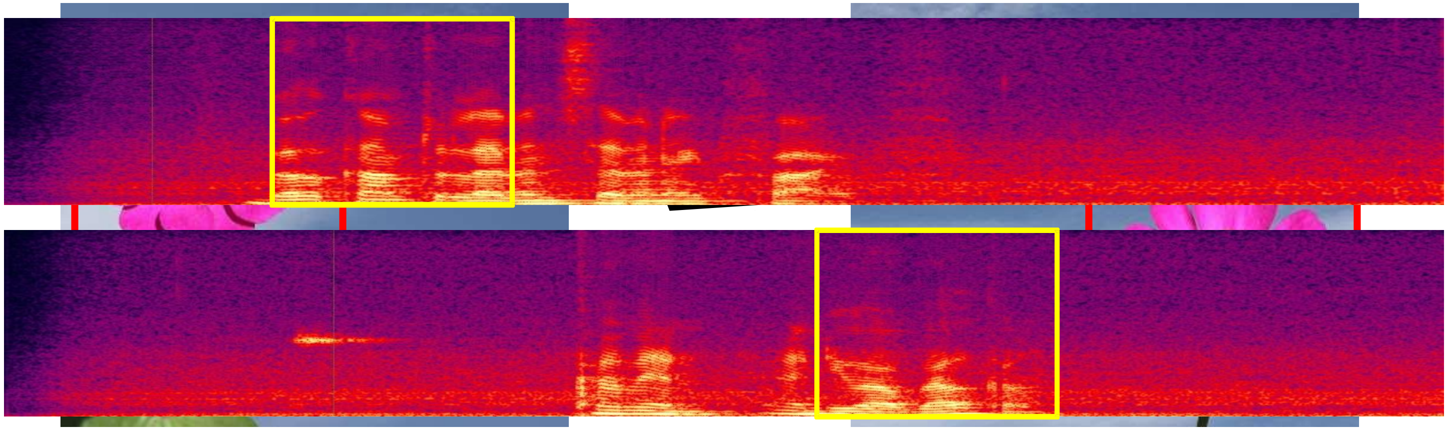
- Is there a flower in any of these images?

Flower



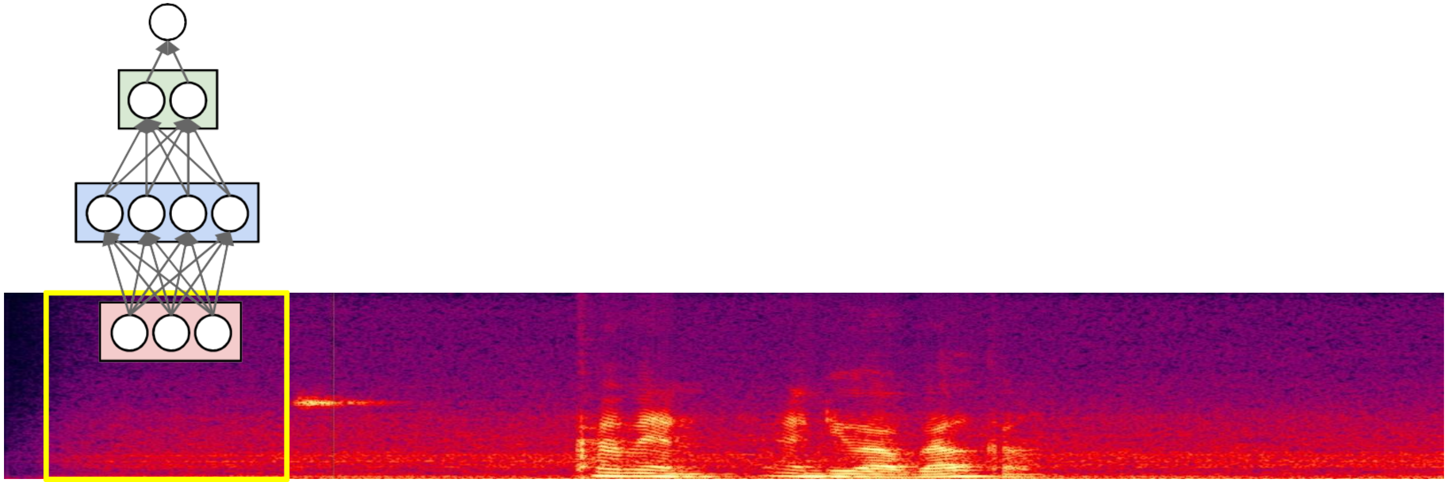
- Will a NN that recognizes the left image as a flower also recognize the one on the right as a flower?
- Need a network that will "fire" regardless of the precise location of the target object

The need for *shift invariance*



- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern is important
- Conventional NNs are sensitive to location of pattern
  - Moving it by one component results in an entirely different input that the NN won't recognize
- Requirement: Network must be *shift invariant*

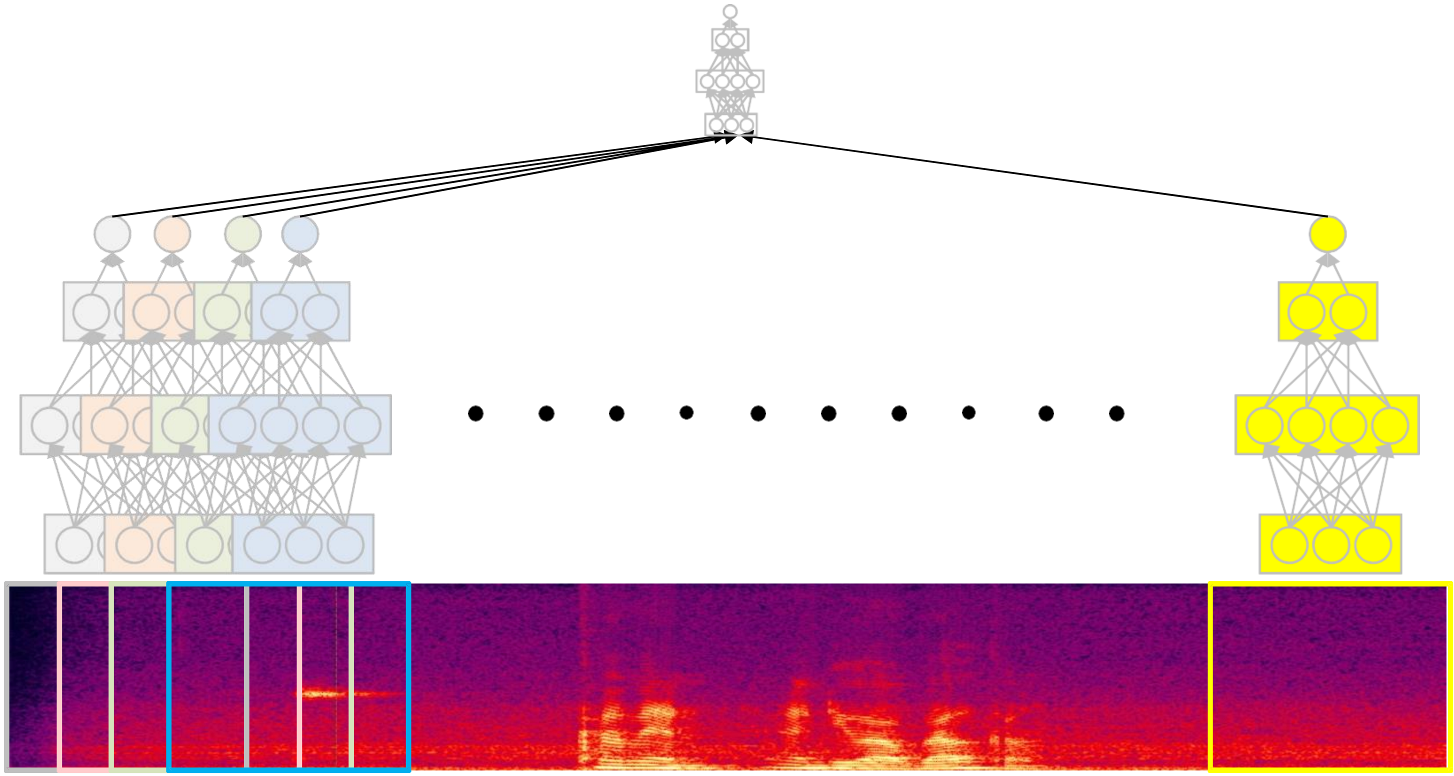
Solution: Scan



- Scan for the target word
  - The audio signals in a "window" are input to a "welcome-detector" NN

Adapted from Bhiksha Raj

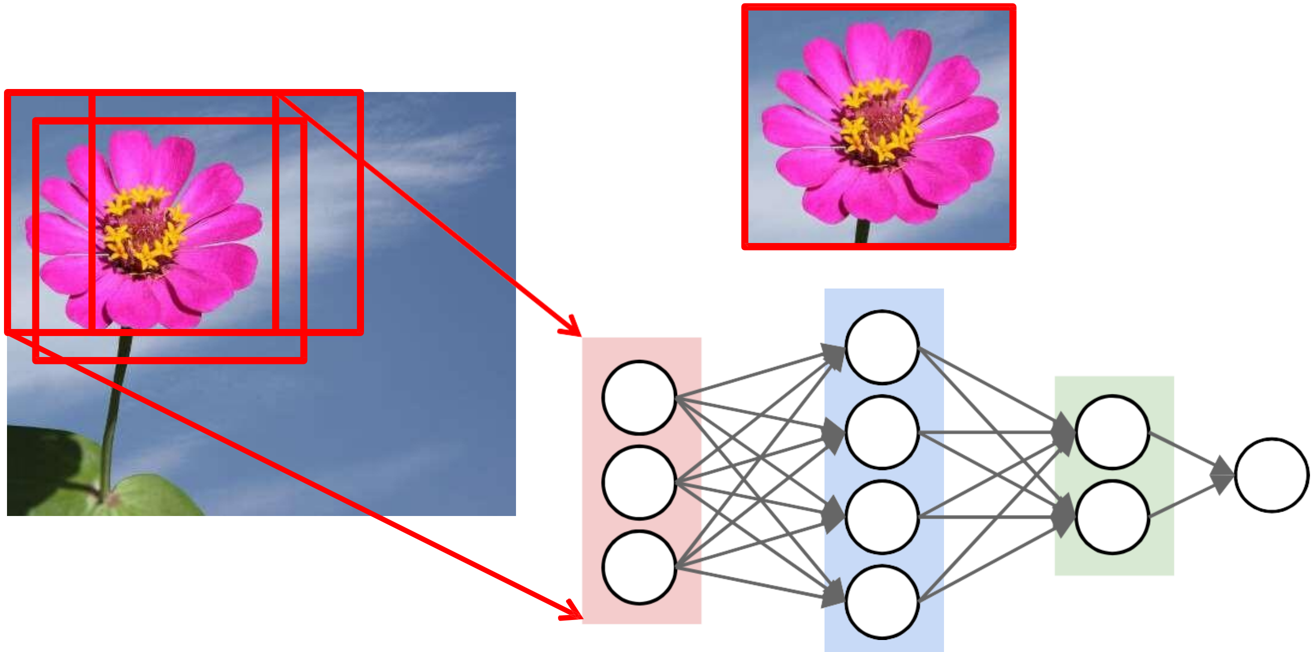
Solution: Scan



- “Does welcome occur in this recording?”
  - Maximum of all outputs (Equivalent of Boolean OR)
  - Or more complex function

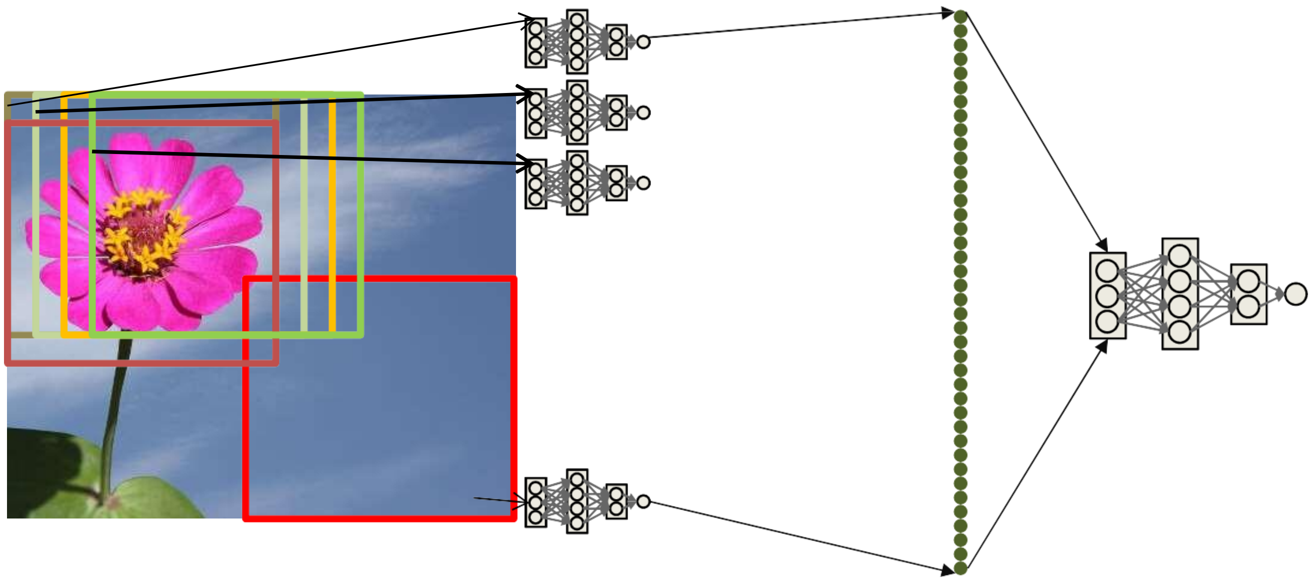
Adapted from Bhiksha Raj

2-d analogue: Does this picture have a flower?



- *Scan* for the desired object
  - “Look” for the target object at each position
  - At each location, entire region is sent through NN

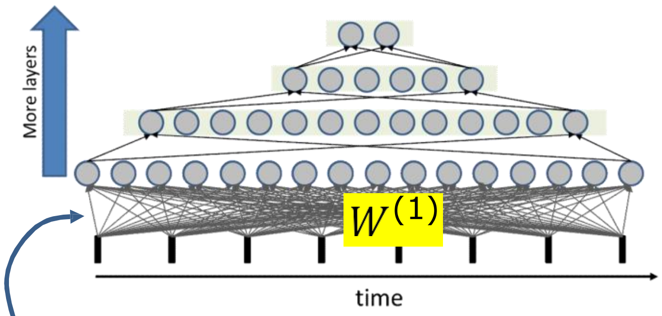
A giant net with common identical subnets



- Determine if any of the locations had a flower
  - Each dot in the right represents the output of the NN when it classifies one location in the input figure
  - Look at the maximum value
  - Or pass it through a simple NN (e.g. linear combination + softmax)

Adapted from Bhiksha Raj

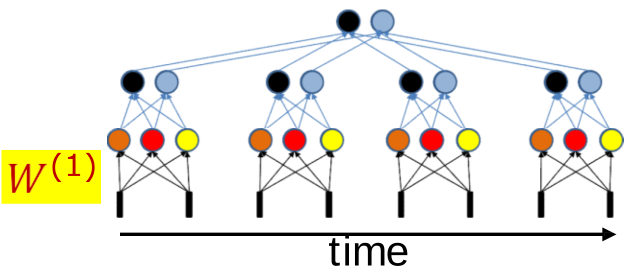
Regular network



$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & \cdots & \cdots & \cdots & w_{1M} \\ w_{21} & w_{22} & w_{23} & w_{24} & \cdots & \cdots & \cdots & w_{2M} \\ w_{31} & w_{32} & w_{33} & w_{34} & \cdots & \cdots & \cdots & w_{3M} \\ w_{41} & w_{42} & w_{43} & w_{44} & \cdots & \cdots & \cdots & w_{4M} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{N1} & w_{N2} & w_{N3} & w_{N4} & \cdots & \cdots & \cdots & w_{NM} \end{bmatrix}$$

- Consider the first layer
  - Assume *N* inputs and *M* outputs
- The weights matrix is a full *N x M* matrix
  - Requiring *N*M* unique parameters

Scanning networks

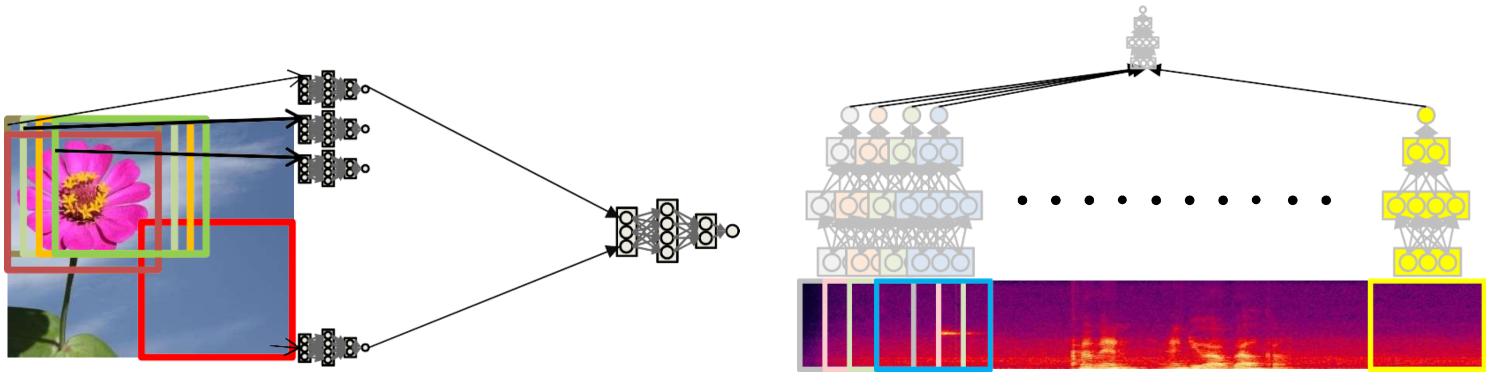


$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ w_{21} & w_{22} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ w_{31} & w_{32} & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & w_{11} & w_{12} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & w_{21} & w_{22} & 0 & 0 & 0 & \vdots & \vdots \\ 0 & 0 & w_{31} & w_{32} & 0 & 0 & 0 & \vdots & \vdots \\ 0 & 0 & 0 & 0 & w_{11} & w_{12} & 0 & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & w_{31} & w_{32} \end{bmatrix}$$

- In a ***scanning NN*** each neuron is connected to a subset of neurons in the previous layer
  - The weights matrix is sparse
  - The weights matrix is block structured ***with identical blocks***
  - **The network is a *shared parameter* model**

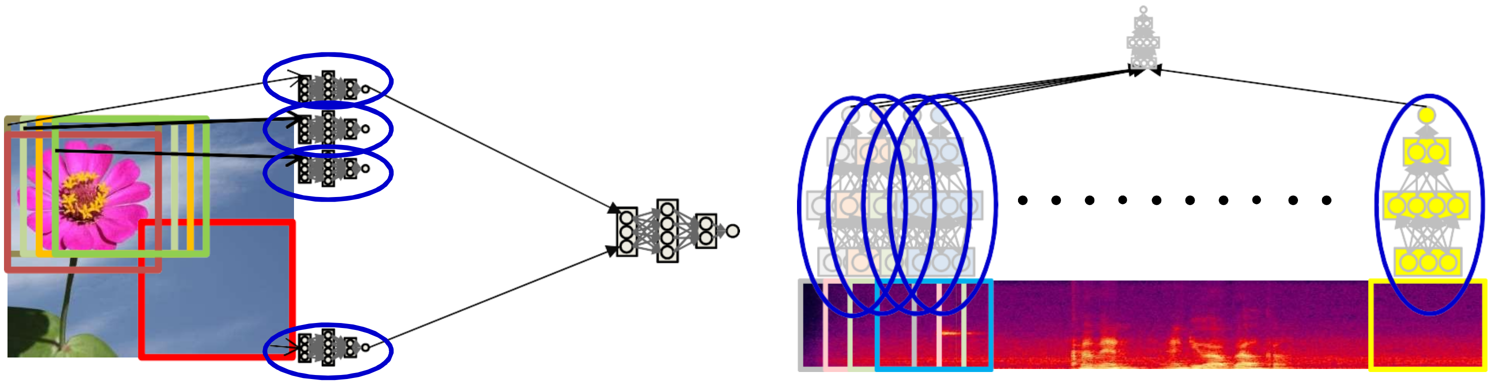
Adapted from Bhiksha Raj

Training the network



- These are really just large networks
  - Can use conventional backpropagation to learn parameters
  - Backprop learns a network that maps the training inputs to the target binary outputs

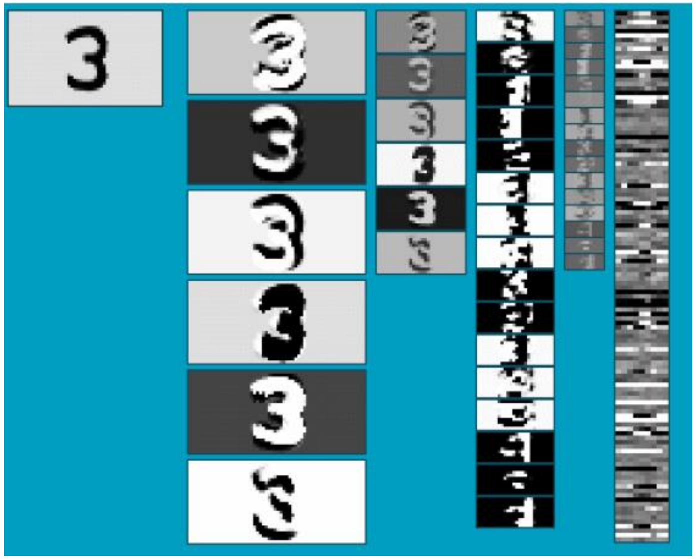
Training the network: constraint

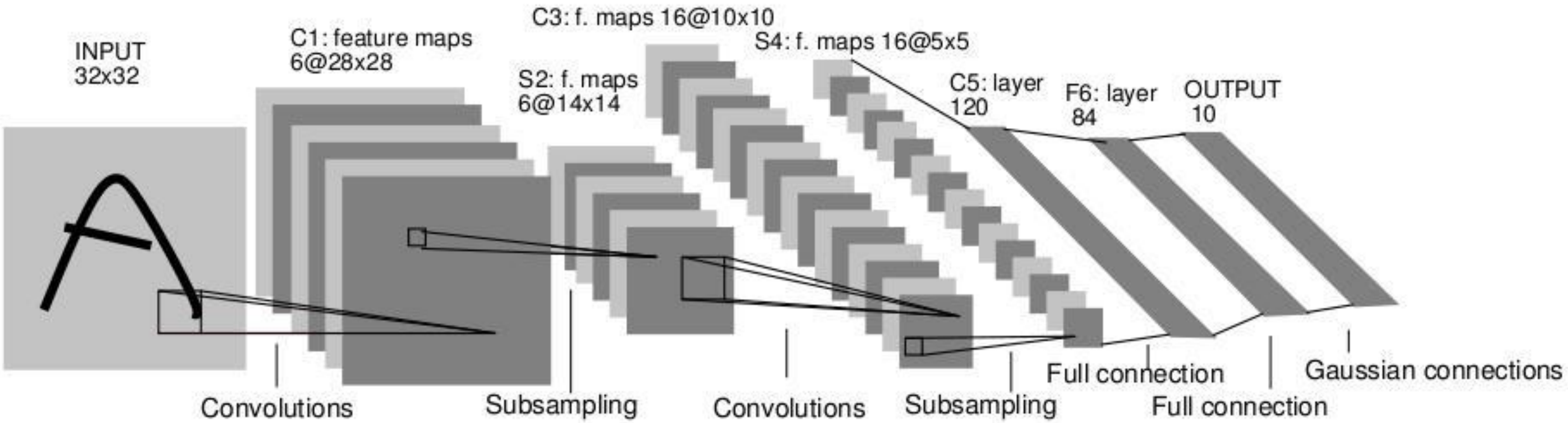


- These are *shared parameter* networks
  - All lower-level subnets are identical
    - Are all searching for the same pattern
  - Any update of the parameters of one copy of the subnet must equally update *all* copies

Convolutional Neural Networks (CNN)

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant, *more abstract* features
- Classification layer at the end





Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, [Gradient-based learning applied to document recognition](), Proceedings of the IEEE 86(11): 2278–2324, 1998.

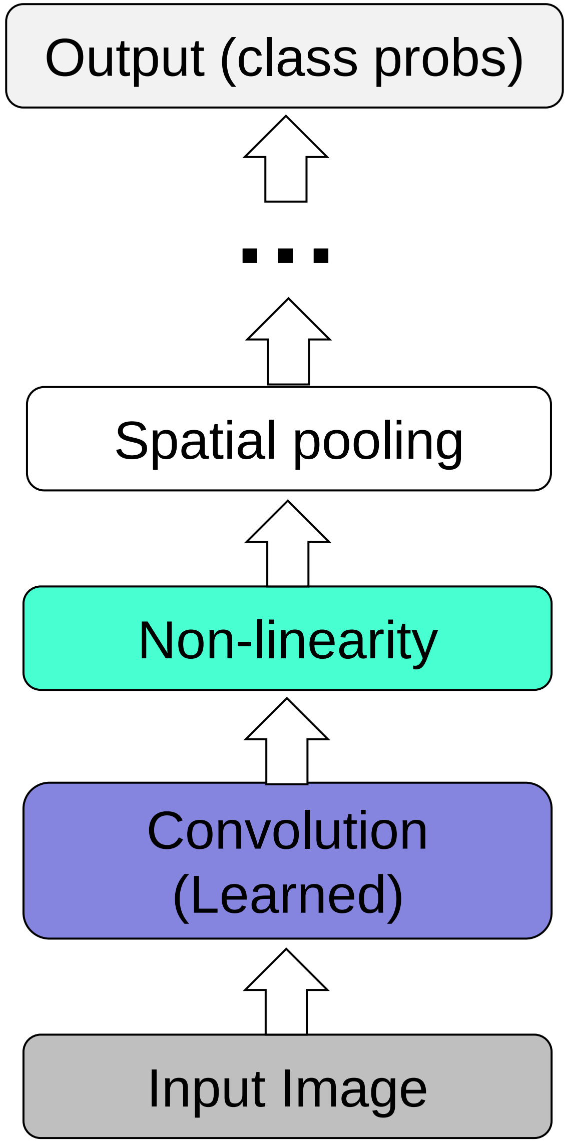
Adapted from Rob Fergus

Convolutional Neural Networks (CNN)

- Feed-forward feature extraction:
  1. *Convolve* input with learned filters
  2. Apply non-linearity
  3. Spatial pooling (downsample)

- Recent architectures have additional operations (to be discussed)

- Trained with some loss, backprop

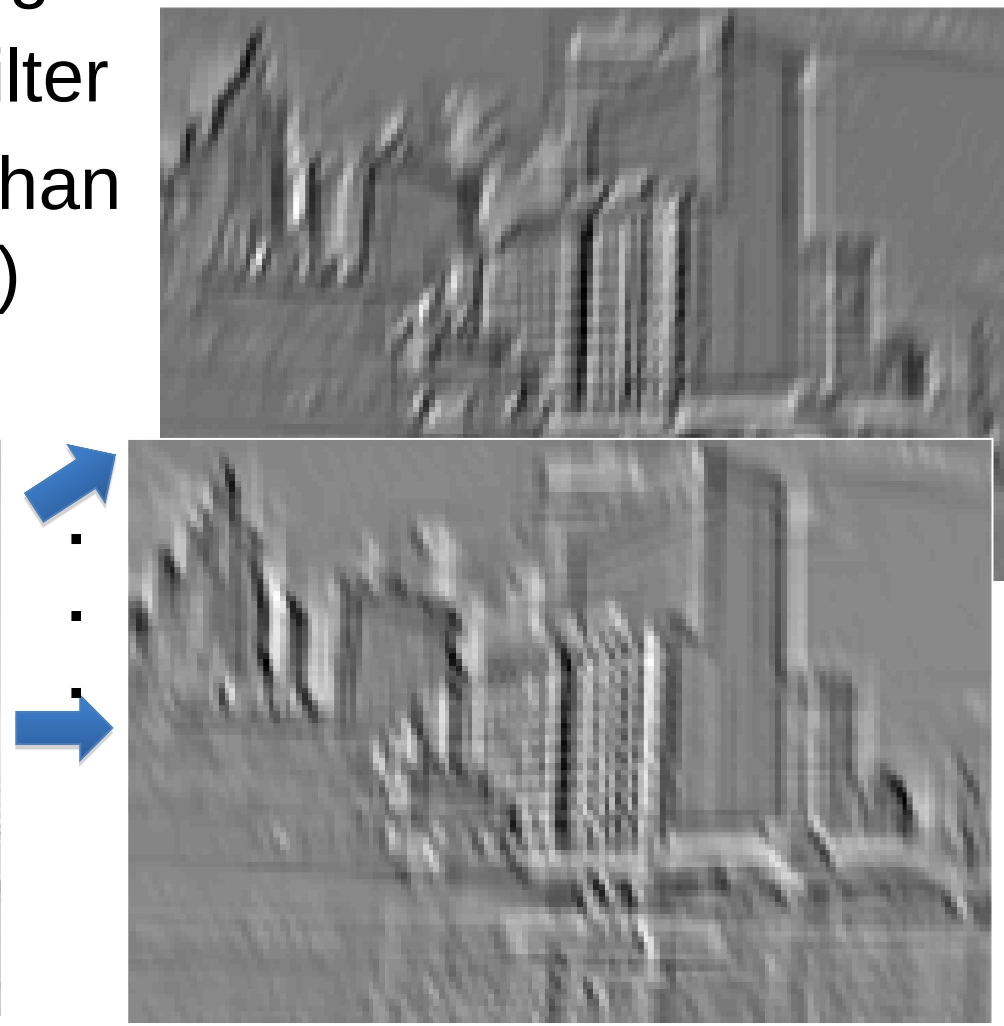


Adapted from Lana Lazebnik

1. Convolution

- Apply learned filter weights
- One feature map per filter
- Stride can be greater than 1 (faster, less memory)

Input

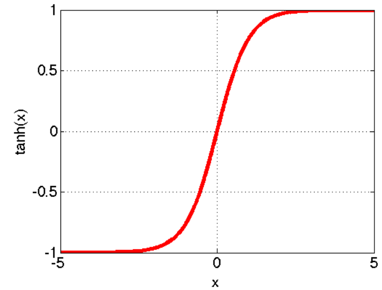


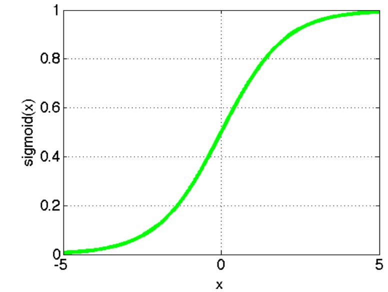
Feature Map

Adapted from Rob Fergus

2. Non-Linearity

- Per-element (independent)
- Some options:
  - Tanh
  - Sigmoid: 1/(1+exp(-x))
  - Rectified linear unit (ReLU)
    - Avoids saturation issues





Krizhevsky et al.

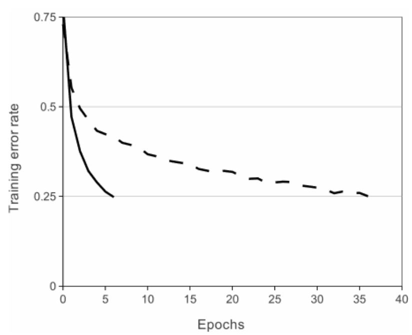
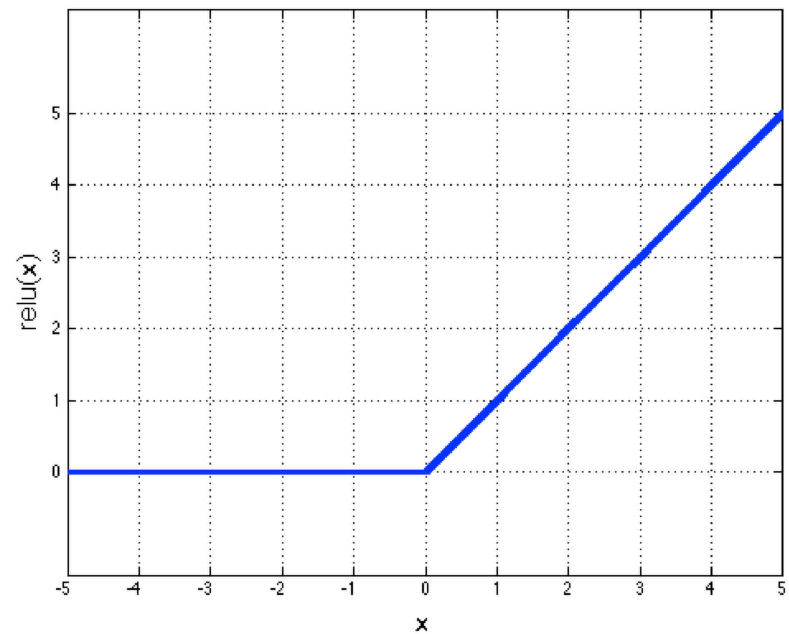


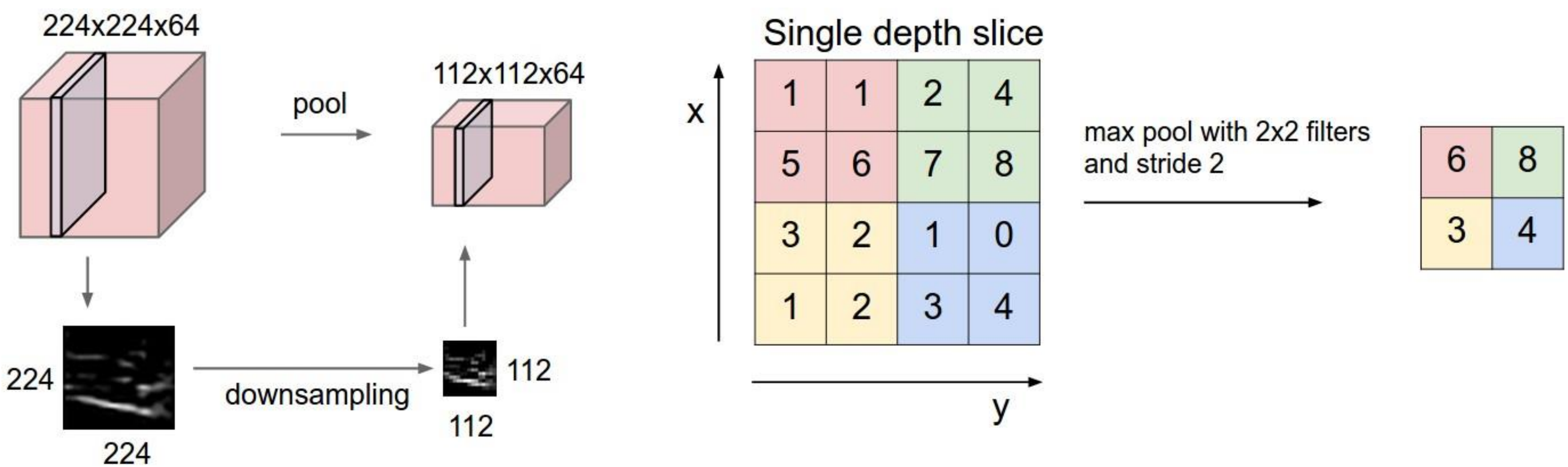
Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.



Adapted from Rob Fergus

3. Spatial Pooling

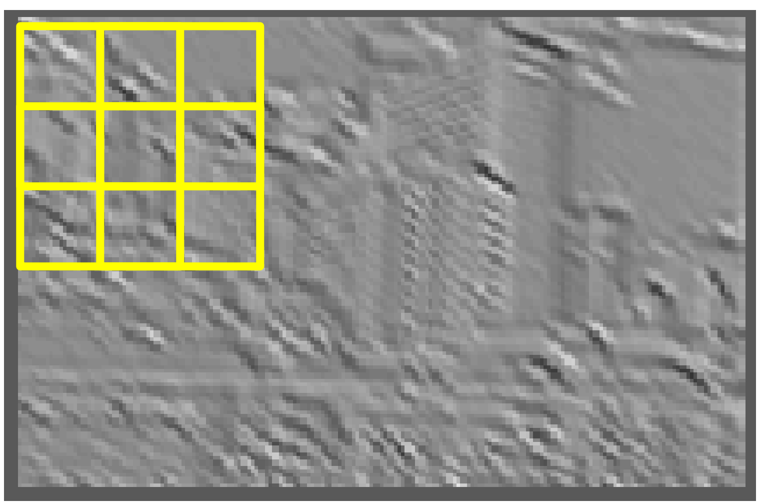
- Sum or max over non-overlapping / overlapping regions



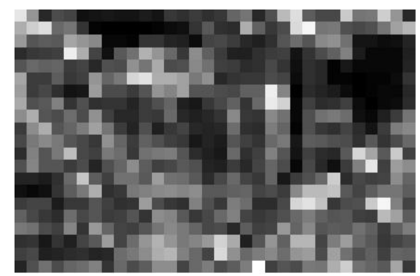
Rob Fergus, figure from Andrej Karpathy

3. Spatial Pooling

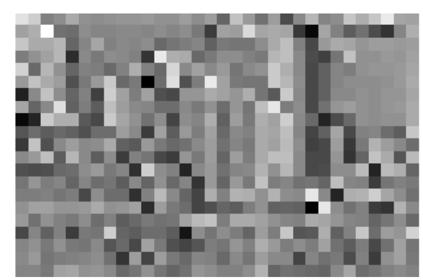
- Sum or max over non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (neurons see more of input)



Max



Sum



Adapted from Rob Fergus

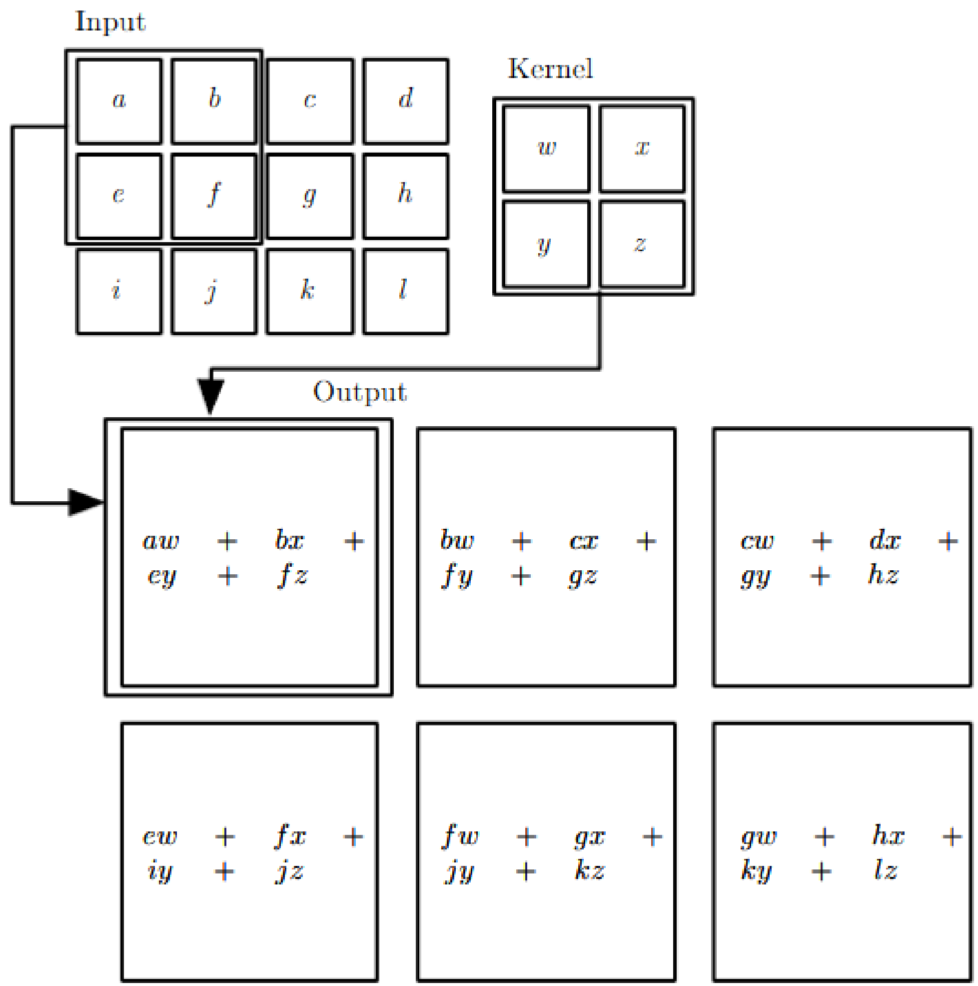


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Background: Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0								

Source: S. Seitz

Background: Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10							

Source: S. Seitz

Background: Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20						

Source: S. Seitz

Background: Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30					

Source: S. Seitz

Background: Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10	20	30	30				

Source: S. Seitz

Background: Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Source: S. Seitz

Image filtering

- Compute a function of the local neighborhood at each pixel in the image
  - Function specified by a “filter” or mask saying how to combine values from neighbors.
  - Element-wise multiplication

- Uses of filtering:
  - Enhance an image (denoise, resize, etc)
  - Extract information (texture, edges, etc)
  - **Detect patterns (template matching)**

Adapted from Derek Hoiem

Correlation filtering

Say the averaging window size is 2k+1 x 2k+1:

$$G[i,j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel F[i,j]}}$$

Now generalize to allow **different weights** depending on neighboring pixel's relative position:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Non-uniform weights}} F[i+u, j+v]$$

Correlation filtering

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

This is called **cross-correlation**, denoted $G = H \otimes F$

Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter a.k.a. kernel a.k.a. mask *H*[*u,v*] is the prescription for the weights in the linear combination.

Adapted from Kristen Grauman

Averaging filter

- What values belong in the kernel *H* for the moving average example?

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$\otimes$

$H[u, v]$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & ? & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

"box filter"

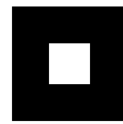
$G[x, y]$

	0	10	20	30	30				

$$G = H \otimes F$$

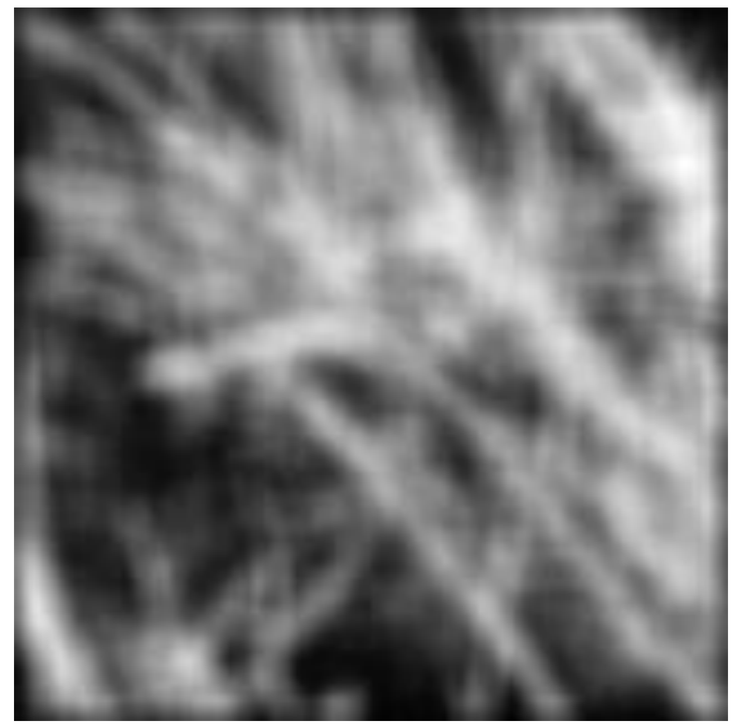
Kristen Grauman

Smoothing by averaging



depicts box filter:
white = high value, black = low value

original



filtered

What if the filter size was 5 x 5 instead of 3 x 3?

Kristen Grauman

Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

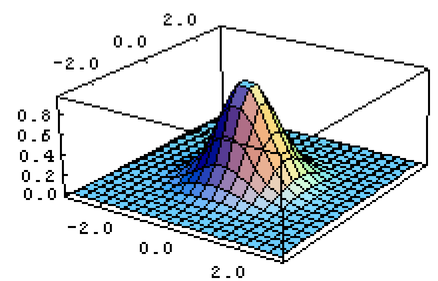
$F[x,y]$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$H[u,v]$

This kernel is an approximation of a 2d Gaussian function:

$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



Source: S. Seitz

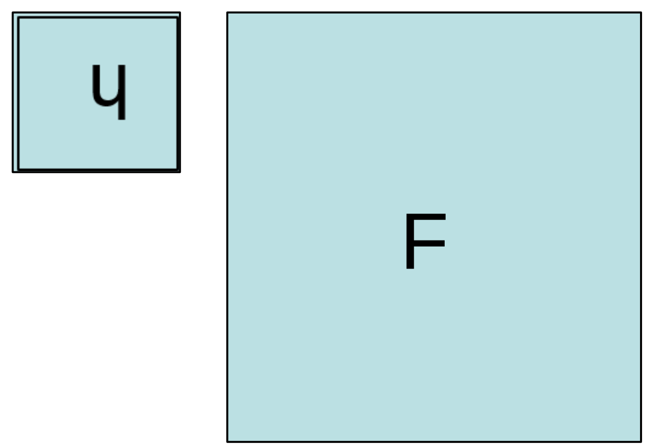
Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

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

$$G = H \star F$$

Notation for convolution operator



Kristen Grauman

Convolution vs. correlation

Convolution

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

$$G = H \star F$$

Cross-correlation

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

$$G = H \otimes F$$

For a Gaussian or box filter, how will the outputs differ?

Convolution vs. correlation

Cross-correlation

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

u = -1, v = -1

$$G = H \otimes F$$

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

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

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs. correlation

Cross-correlation

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

$$G = H \otimes F$$

u = -1, v = -1

v = 0

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

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

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs. correlation

Cross-correlation

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

$$G = H \otimes F$$

u = -1, v = -1

v = 0

v = +1

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

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

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs. correlation

Cross-correlation

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

$$G = H \otimes F$$

u = -1, v = -1
v = 0
v = +1
u = 0, v = -1

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

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

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs. correlation

Cross-correlation

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

$$G = H \otimes F$$

u = -1, v = -1

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

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

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs. correlation

Cross-correlation

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

$$G = H \otimes F$$

u = -1, v = -1

v = 0

F

5	2	5	4	4	
5	200	3	200	4	
1	5	5	4	4	(i, j)
5	5	1	1	2	
200	1	3	5	200	
1	200	200	200	1	

Convolution

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

$$G = H \star F$$

H

.06	.12	.06	
.12	.25	.12	(0, 0)
.06	.12	.06	

Convolution vs. correlation

Cross-correlation

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

$$G = H \otimes F$$

u = -1, v = -1

v = 0

v = +1

F

5	2	5	4	4
5	200	3	200	4
1	5	5	4	4
5	5	1	1	2
200	1	3	5	200
1	200	200	200	1

(i, j)

Convolution

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

$$G = H \star F$$

H

.06	.12	.06
.12	.25	.12
.06	.12	.06

(0, 0)

Convolution vs. correlation

Cross-correlation

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

$$G = H \otimes F$$

u = -1, v = -1

v = 0

v = +1

u = 0, v = -1

F

5	2	5	4	4	
5	200	3	200	4	
1	5	5	4	4	(i, j)
5	5	1	1	2	
200	1	3	5	200	
1	200	200	200	1	

Convolution

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

$$G = H \star F$$

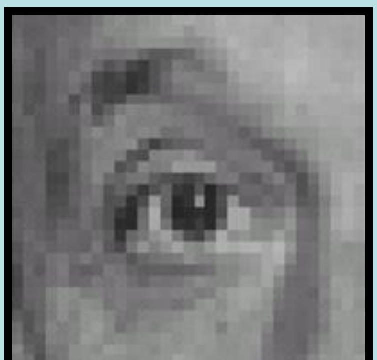
H

.06	.12	.06	
.12	.25	.12	(0, 0)
.06	.12	.06	

Predict the outputs using correlation filtering

$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = ?$$

$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = ?$$



$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

Kristen Grauman

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

Shifted left by 1 pixel with correlation

Source: D. Lowe

Practice with linear filters

Original

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

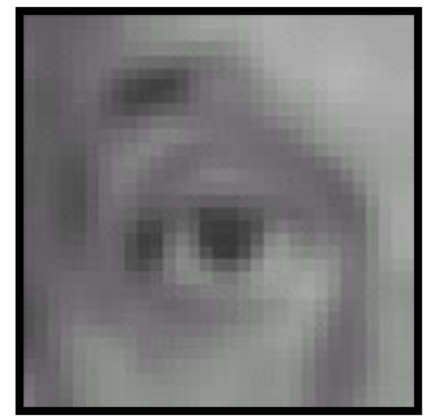
?

Source: D. Lowe

Practice with linear filters

Original

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Blur (with a box filter)

Source: D. Lowe

Practice with linear filters

Original

$$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \quad ?$$

Source: D. Lowe

Practice with linear filters

Original

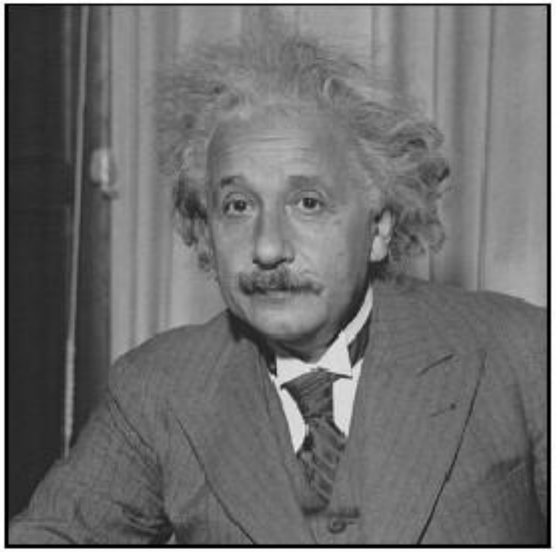
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter:

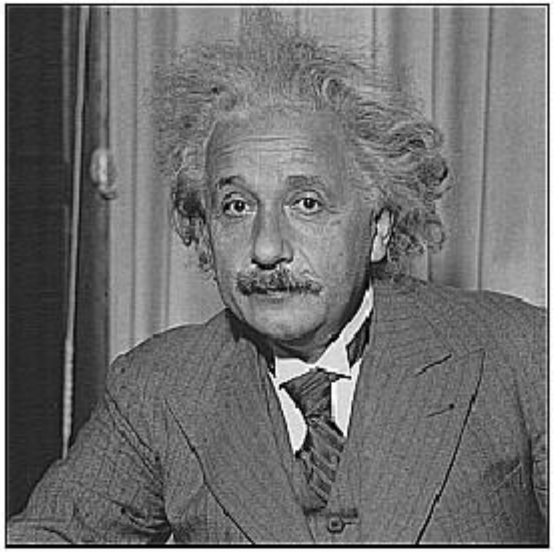
accentuates differences with local average

Source: D. Lowe

Sharpening



before

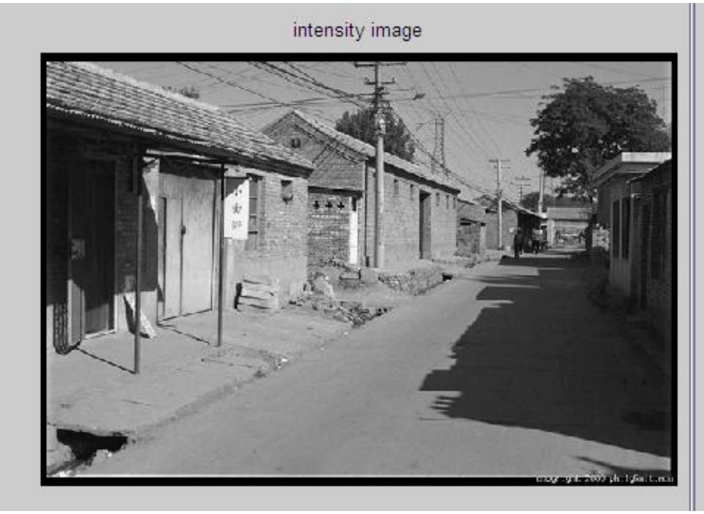


after

Kristen Grauman

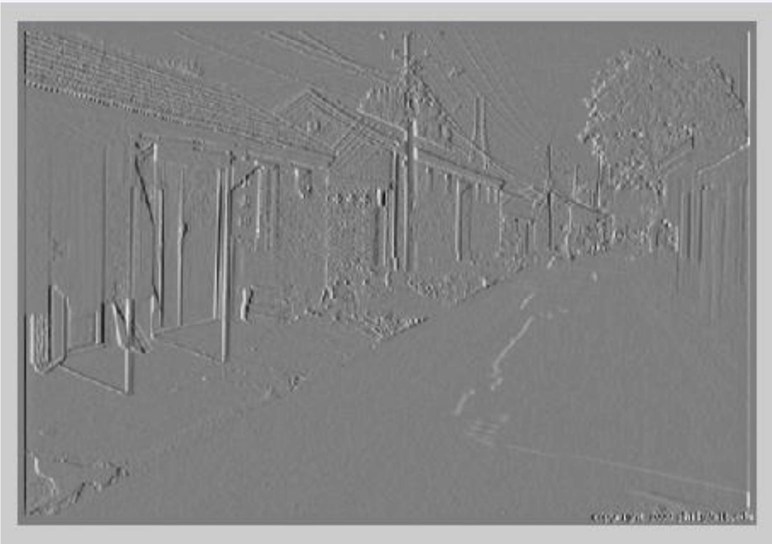
Filters for computing *gradients*

1	0	-1
2	0	-2
1	0	-1



*

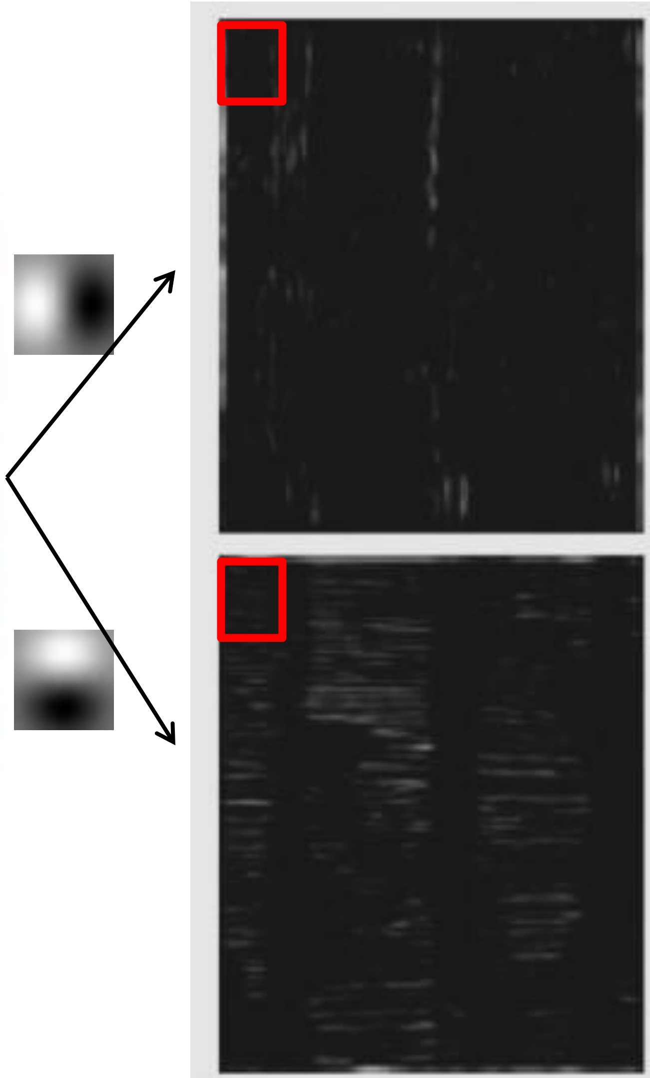
=



Slide credit: Derek Hoiem

Texture representation: example

original image



derivative filter responses, squared

	mean d/dx value	**mean d/dy value**
Win. #1	4	10

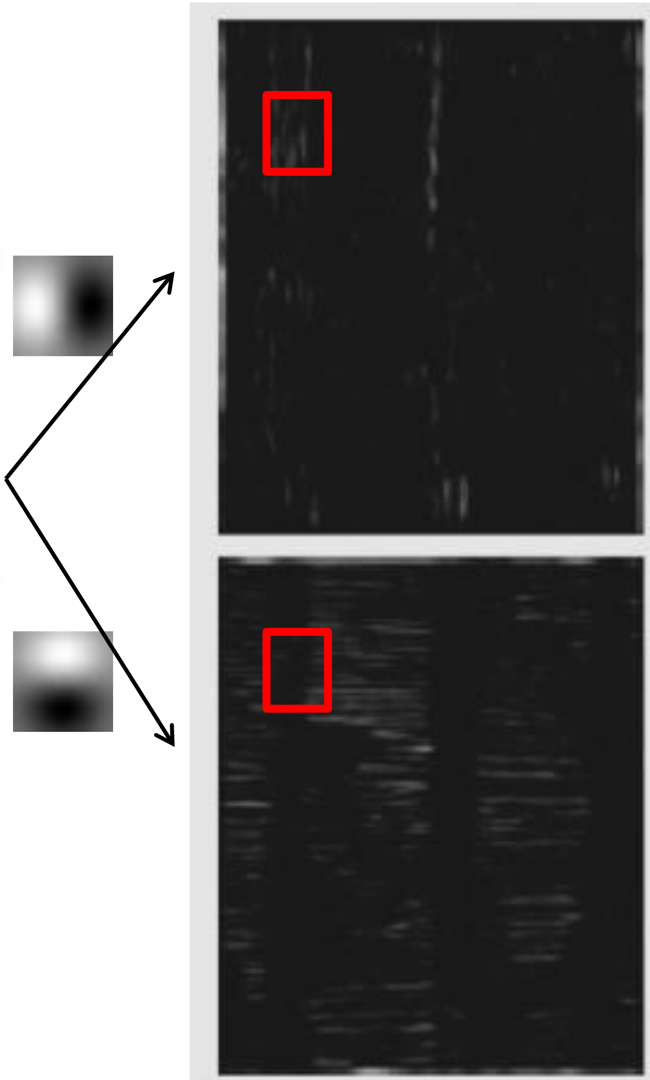
⋮

statistics to summarize patterns in small windows

Kristen Grauman

Texture representation: example

original image



derivative filter responses, squared

	mean d/dx value	mean d/dy value
Win. #1	4	10
Win.#2	18	7

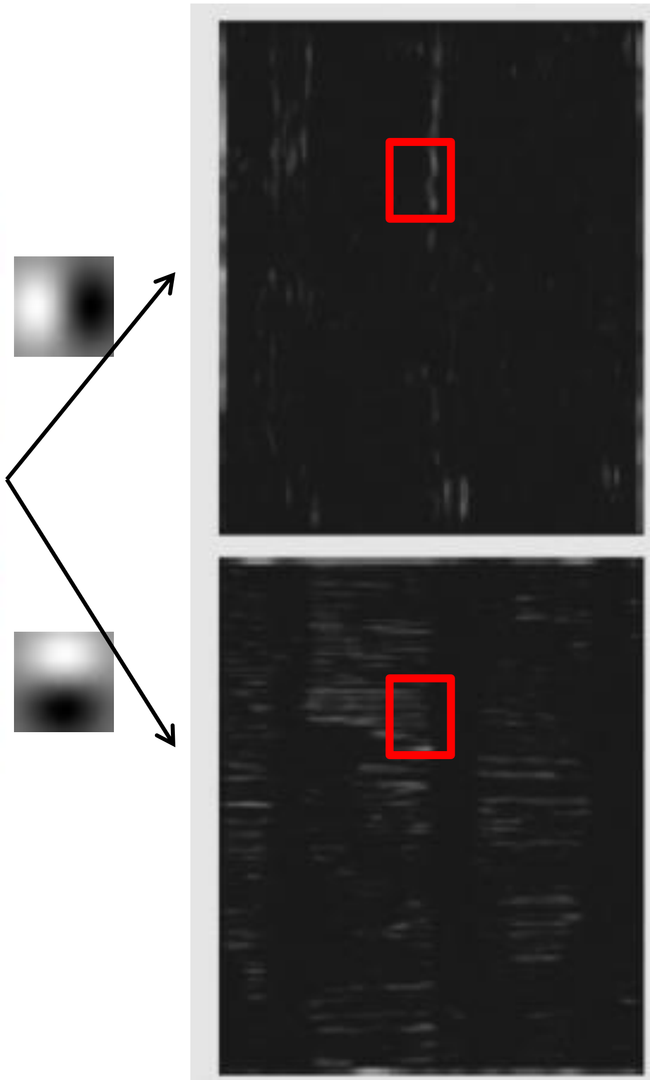
⋮

statistics to summarize patterns in small windows

Kristen Grauman

Texture representation: example

original image



derivative filter responses, squared

	mean d/dx value	mean d/dy value
Win. #1	4	10
Win.#2 ⋮	18	7
Win.#9	20	20

⋮

statistics to summarize patterns in small windows

Kristen Grauman

Vectors of texture responses

To represent pixel, form a "feature vector" from the responses at that pixel.

To represent *image*, compute statistics over all pixel feature vectors, e.g. their mean.

$[\mathtt{r}_{(1,1)}^{1},\ \mathtt{r}_{(1,1)}^{2},\ \ldots,\ \mathtt{r}_{(1,1)}^{38}]$

$[\mathtt{r}_{(1,2)}^{1},\ \mathtt{r}_{(1,2)}^{2},\ \ldots,\ \mathtt{r}_{(1,2)}^{38}]$

Pixel location (row, column)

Filter ID

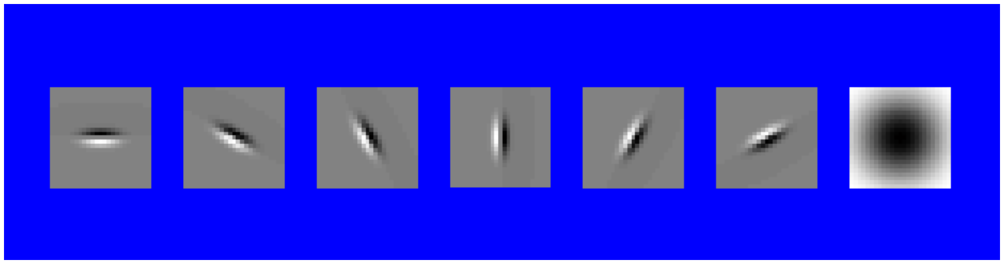
...

$[\mathtt{r}_{(W,H)}^{1},\ \mathtt{r}_{(W,H)}^{2},\ \ldots,\ \mathtt{r}_{(W,H)}^{38}]$

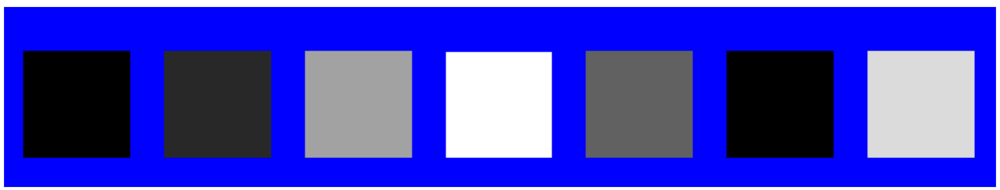
$[\mathtt{mean}(\mathtt{r}_{(:)}^{1}),\ \mathtt{mean}(\mathtt{r}_{(:)}^{2}),\ \ldots,\ \mathtt{mean}(\mathtt{r}_{(:)}^{38})]$

You try: Can you match the texture to the response?

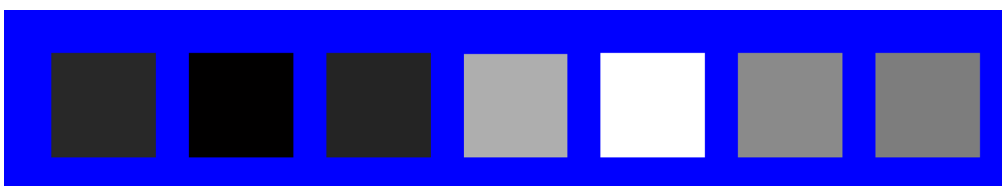
Filters



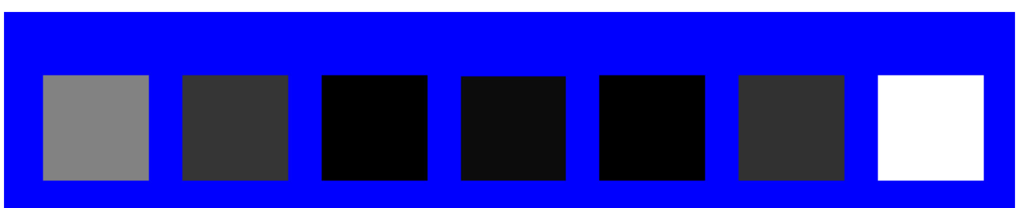
1



2

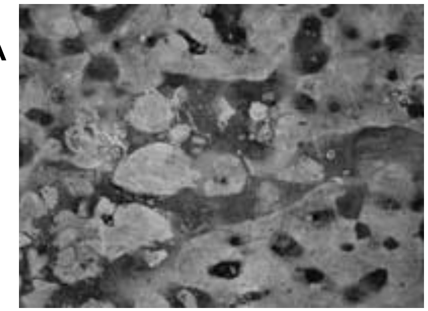


3



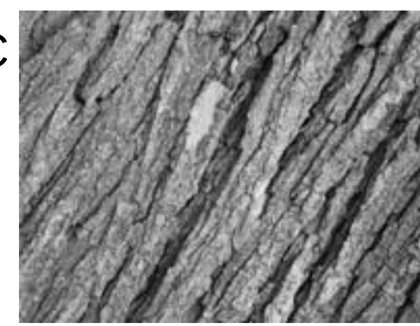
Mean abs responses

A



B

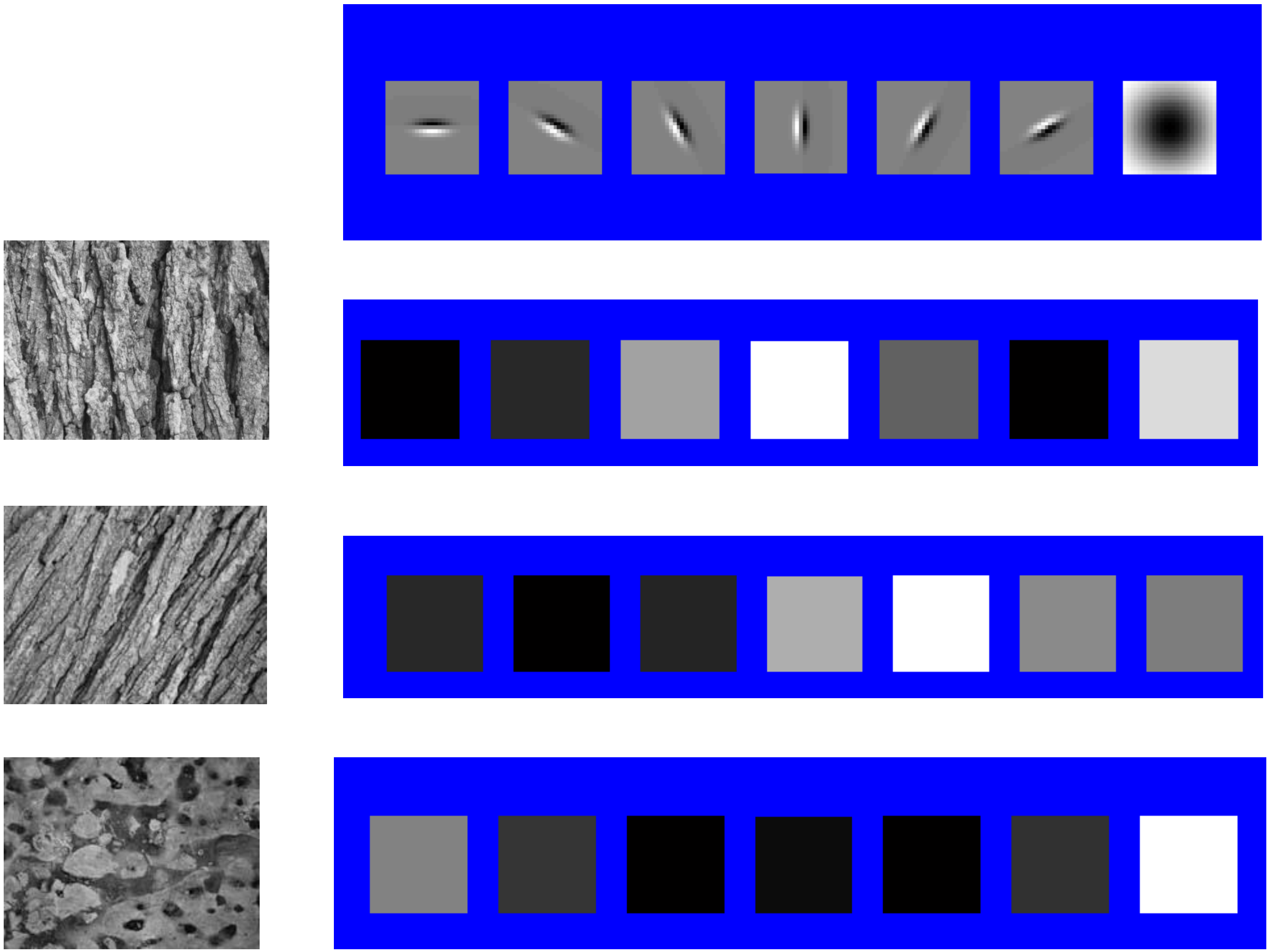
C



Derek Hoiem

Representing texture by mean abs response

Filters

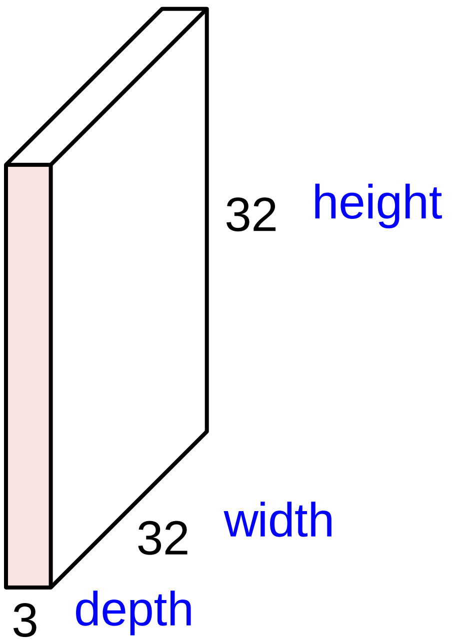


Mean abs responses

Derek Hoiem

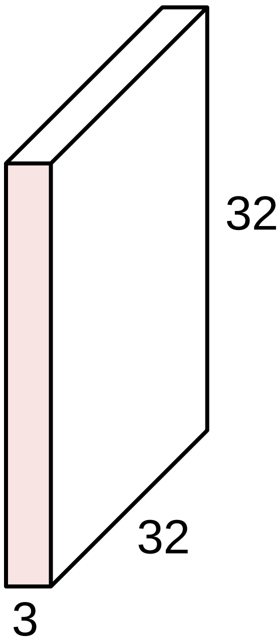
Convolutions: More detail

32x32x3 image

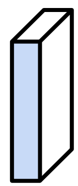


Convolutions: More detail

32x32x3 image



5x5x3 filter



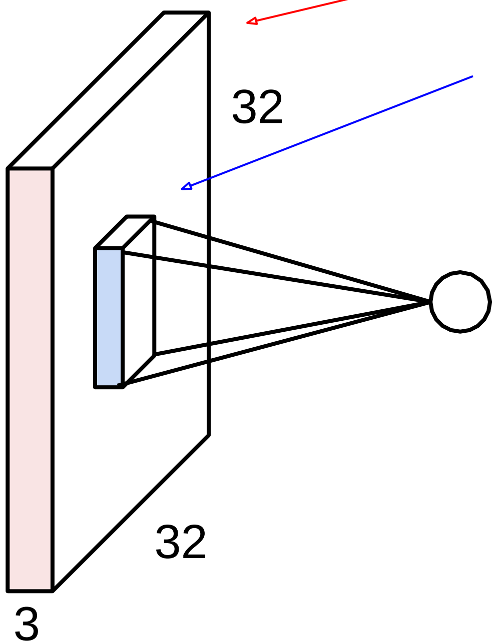
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolutions: More detail

Convolution Layer

32x32x3 image

5x5x3 filter w



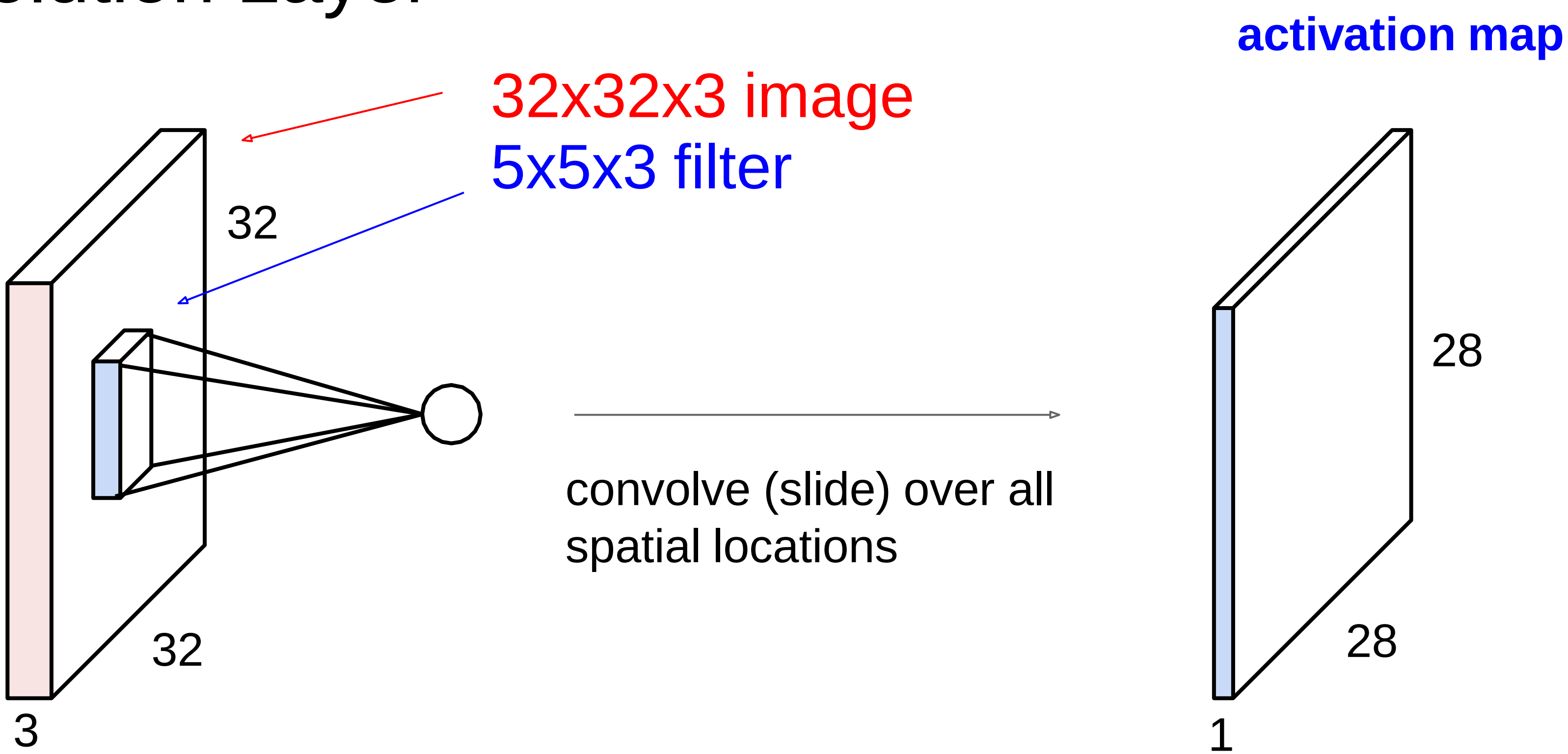
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Andrej Karpathy

Convolutions: More detail

Convolution Layer

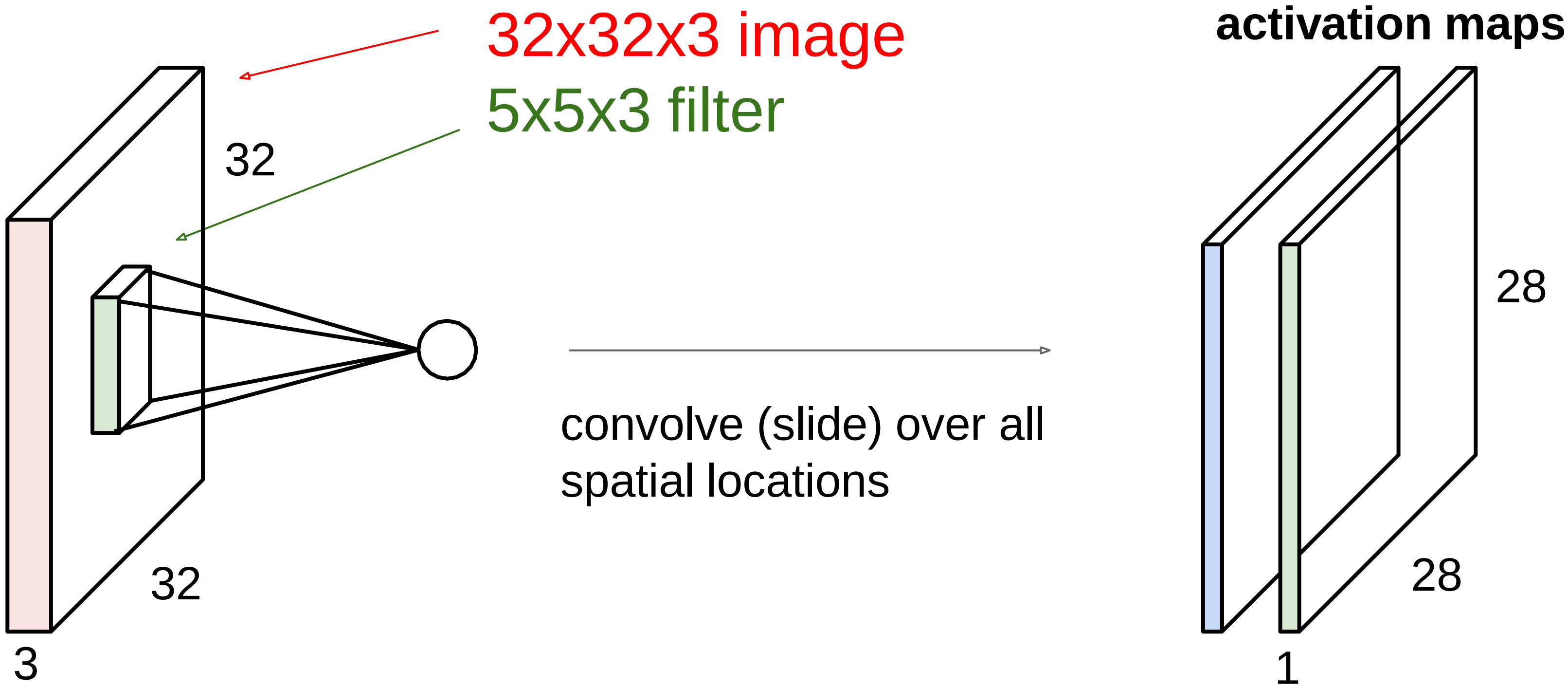


Andrej Karpathy

Convolutions: More detail

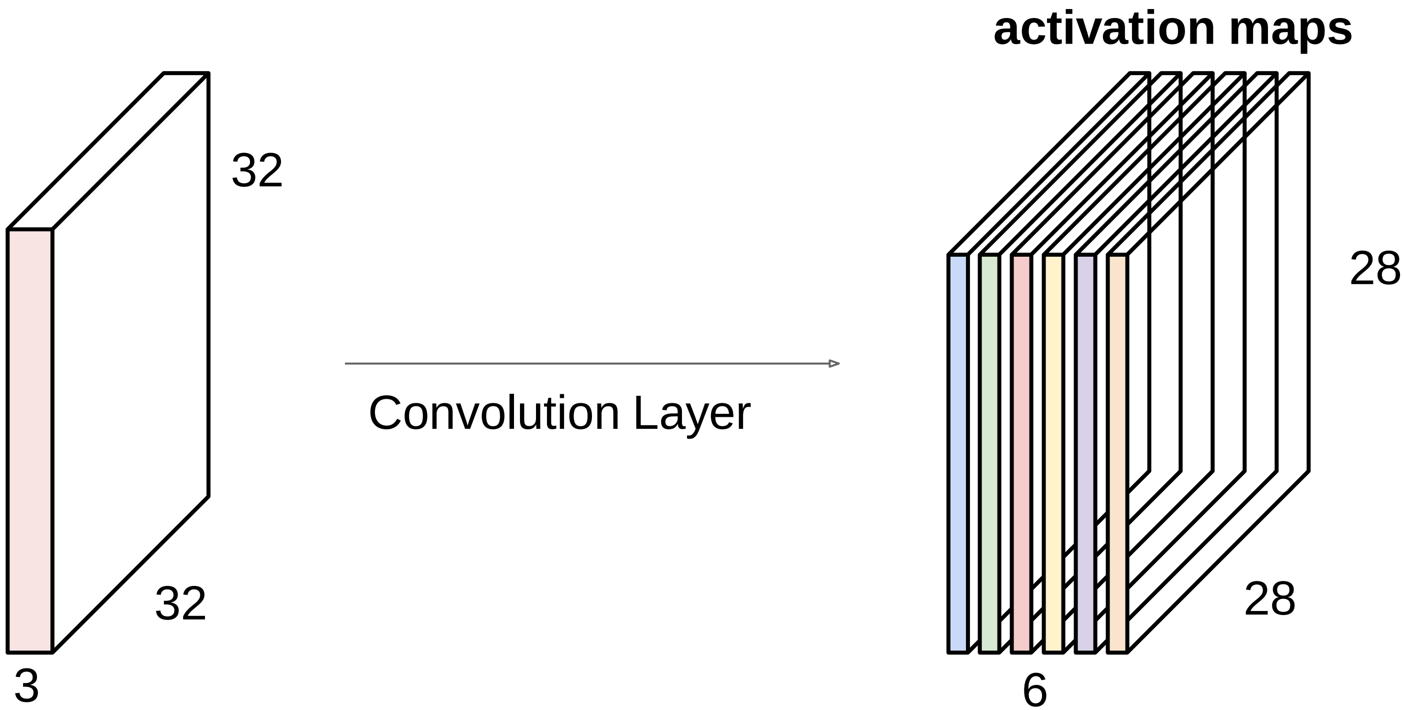
Convolution Layer

consider a second, green filter



Convolutions: More detail

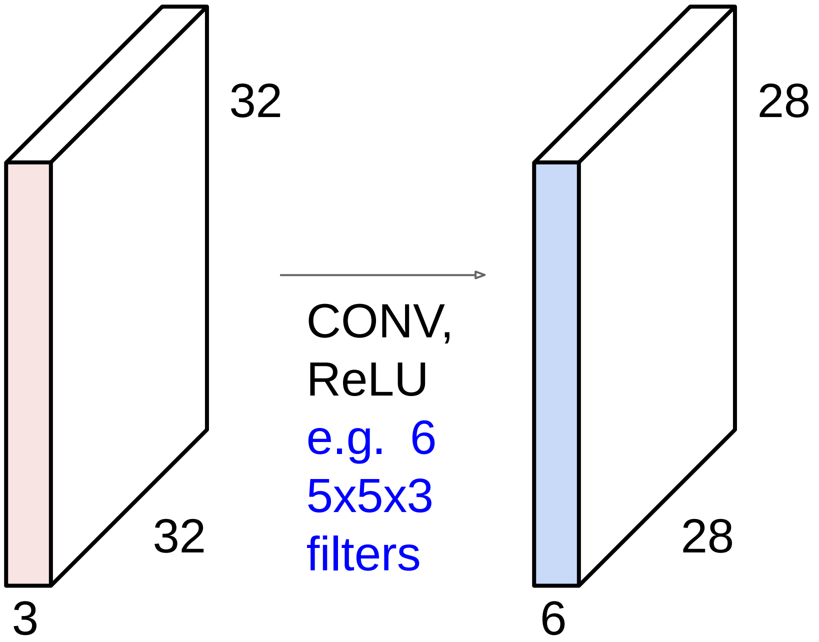
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

Convolutions: More detail

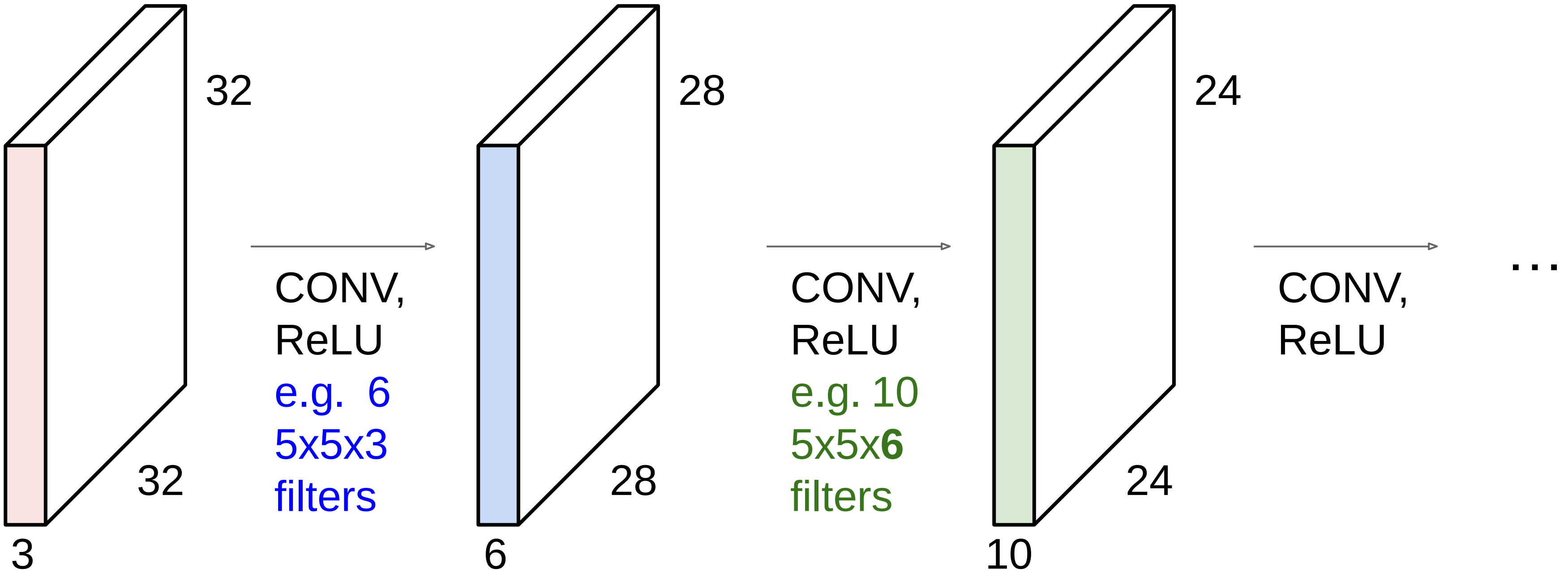
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Andrej Karpathy

Convolutions: More detail

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

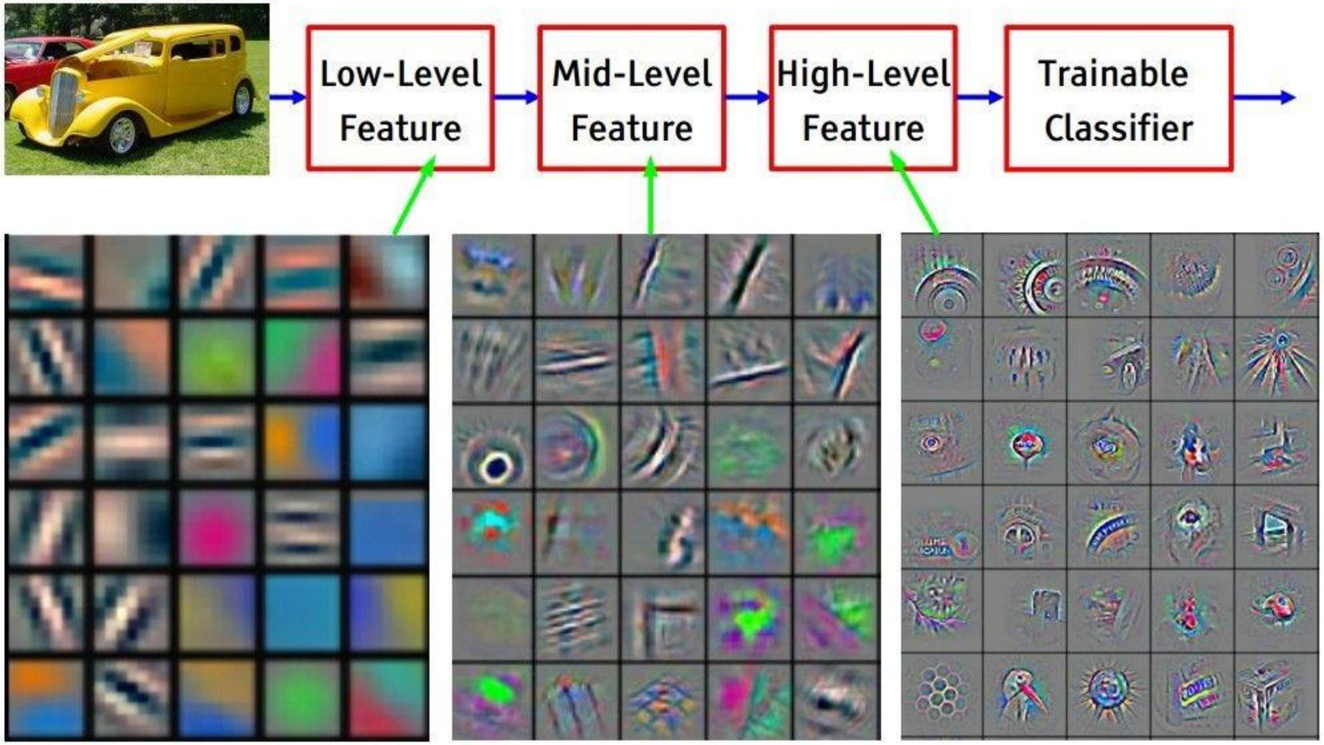


Andrej Karpathy

Convolutions: More detail

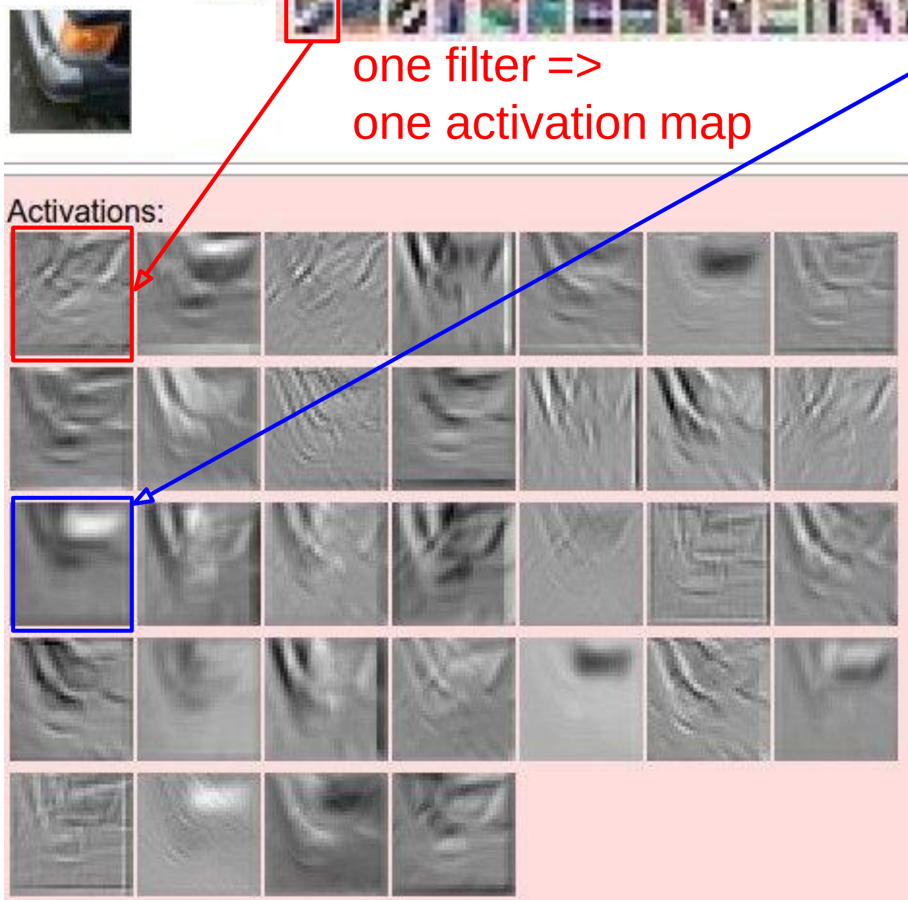
Preview

[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:

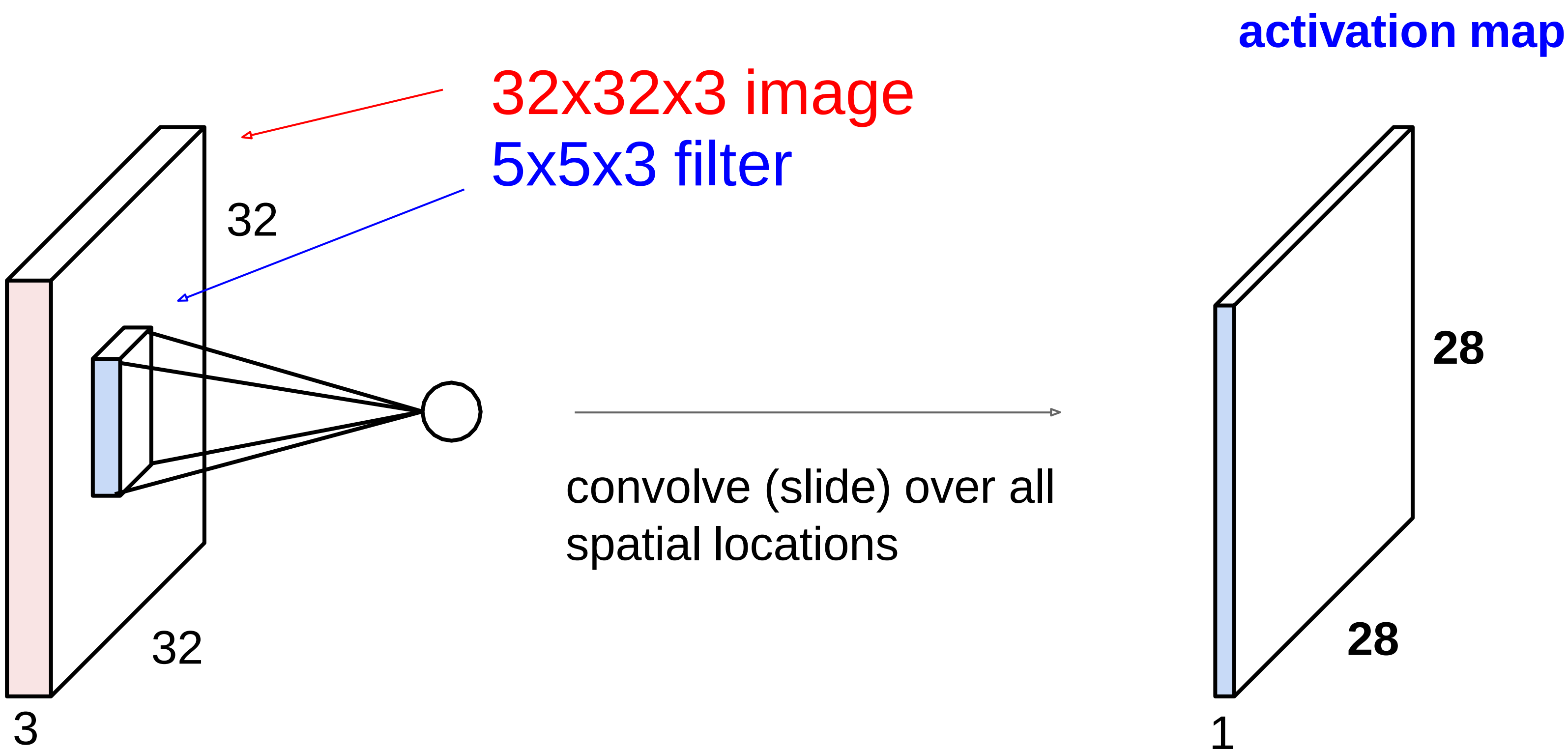
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

Element-wise multiplication and sum of a filter and the signal (image)

Adapted from Andrej Karpathy, Kristen Grauman

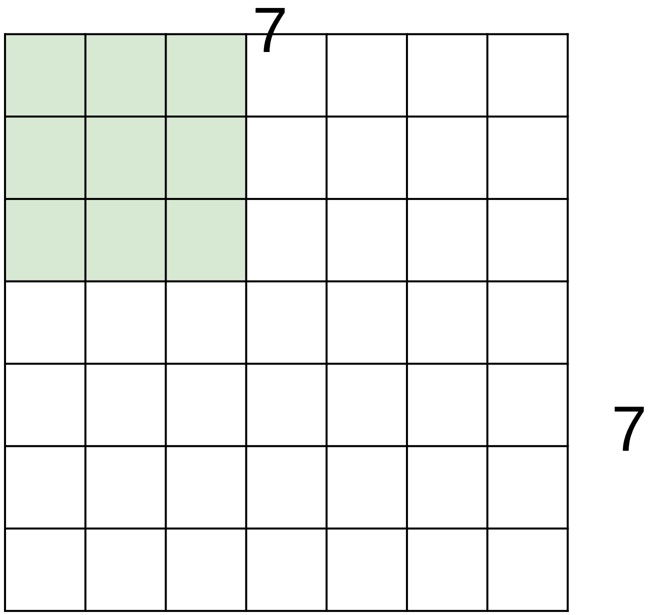
Convolutions: More detail

A closer look at spatial dimensions:



Convolutions: More detail

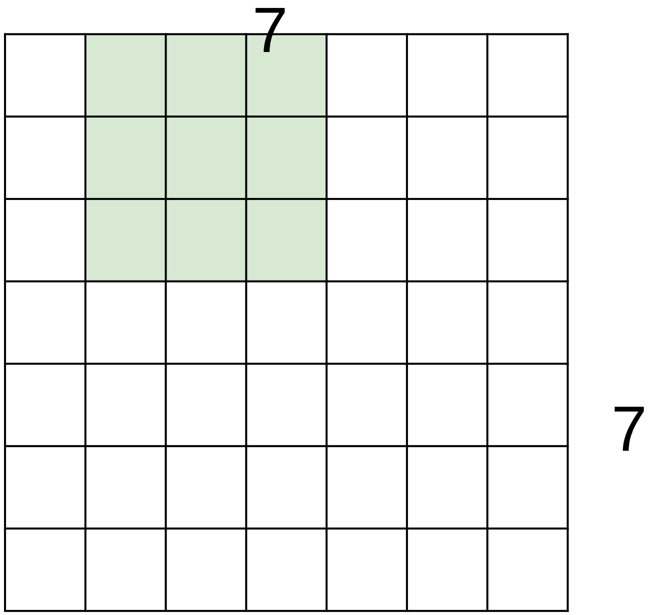
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutions: More detail

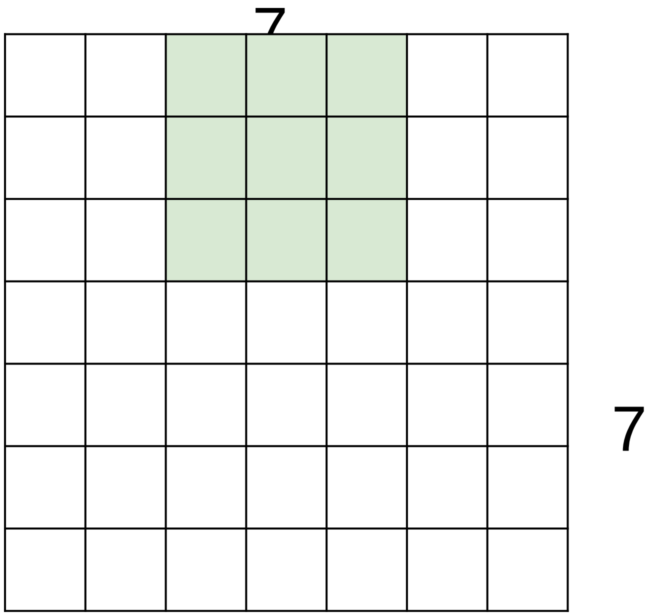
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutions: More detail

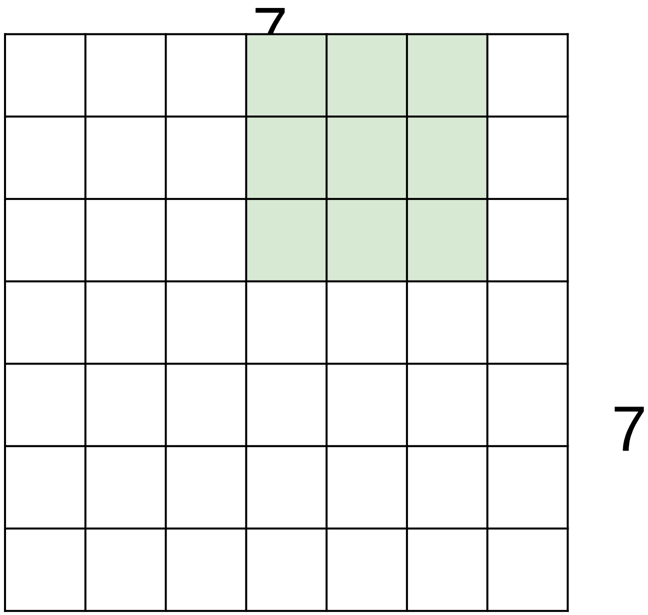
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutions: More detail

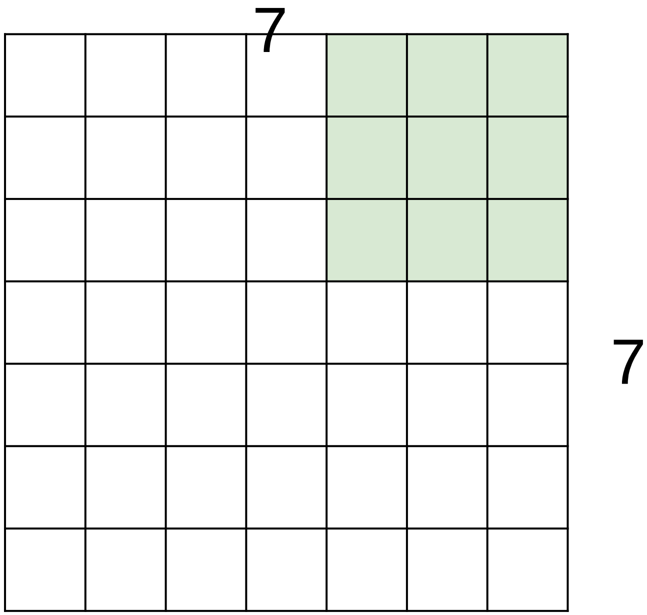
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutions: More detail

A closer look at spatial dimensions:

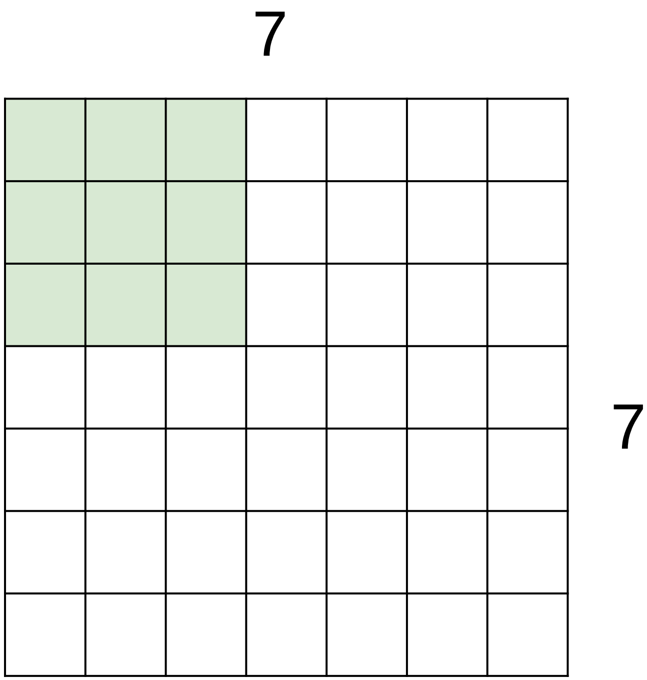


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Convolutions: More detail

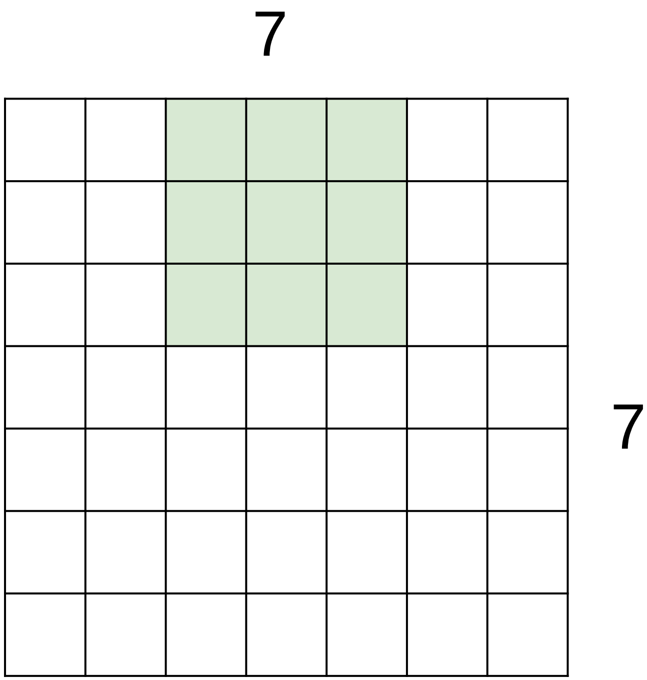
A closer look at spatial dimensions:



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

Convolutions: More detail

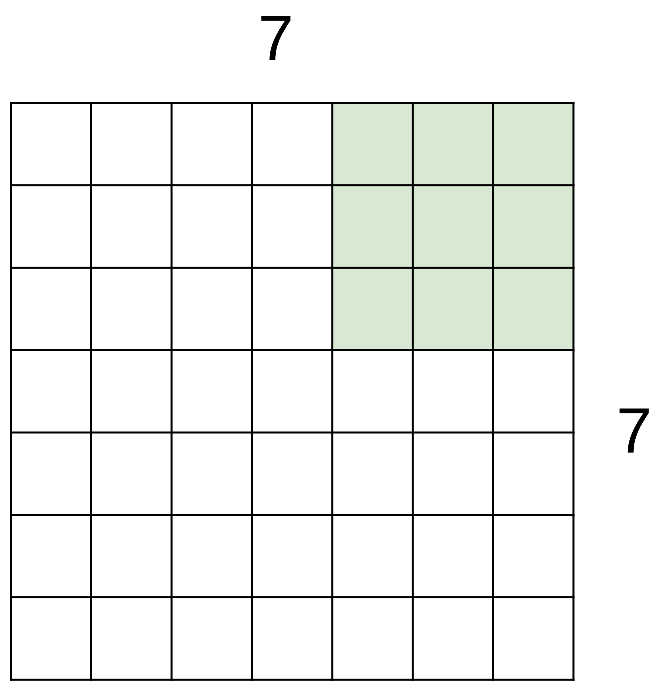
A closer look at spatial dimensions:



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

Convolutions: More detail

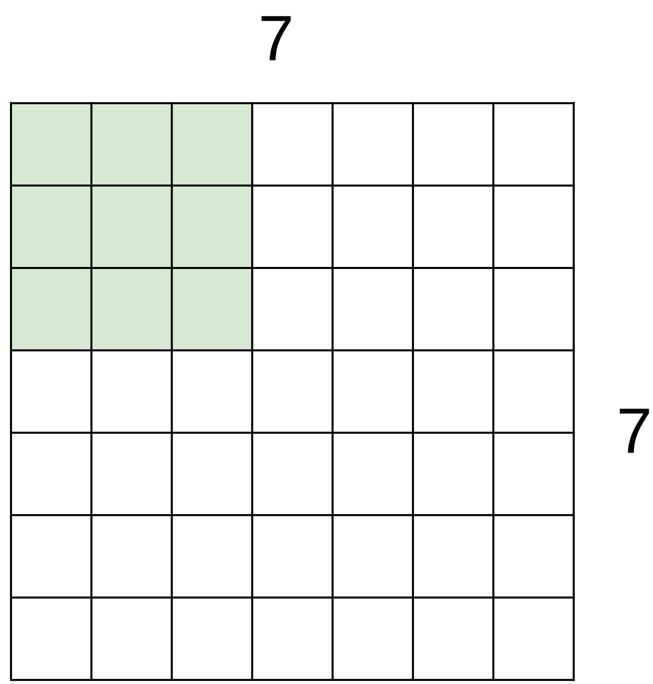
A closer look at spatial dimensions:



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

Convolutions: More detail

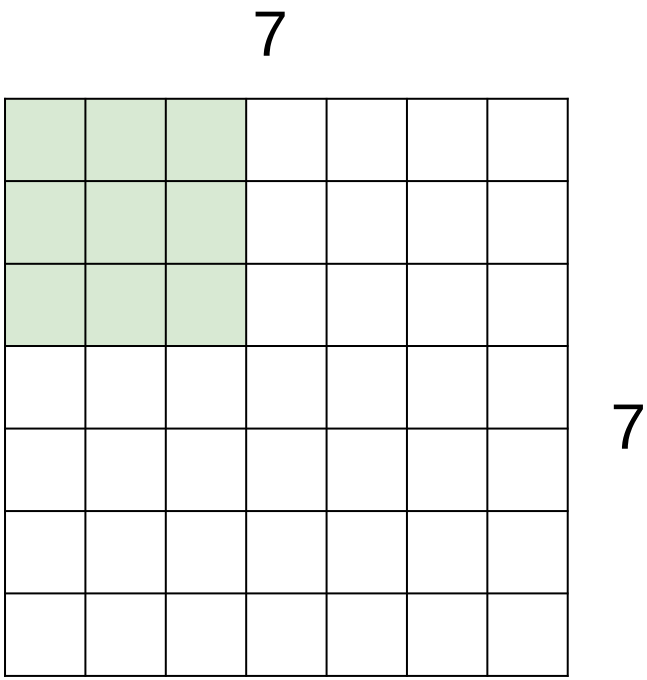
A closer look at spatial dimensions:



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

Convolutions: More detail

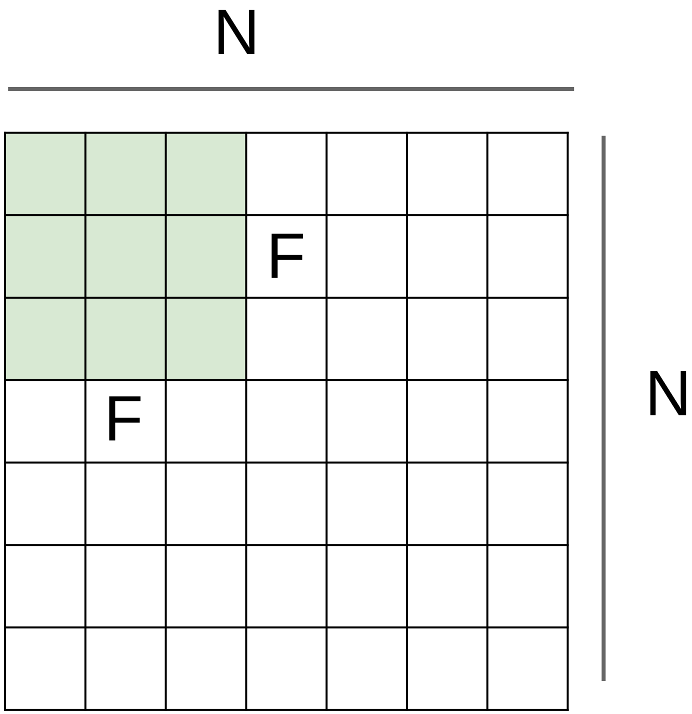
A closer look at spatial dimensions:



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

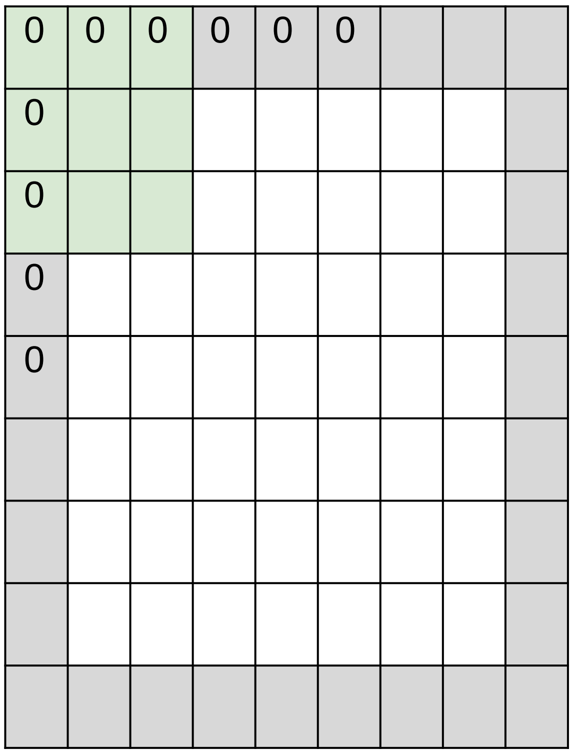


Output size:
(N - F) / stride + 1

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

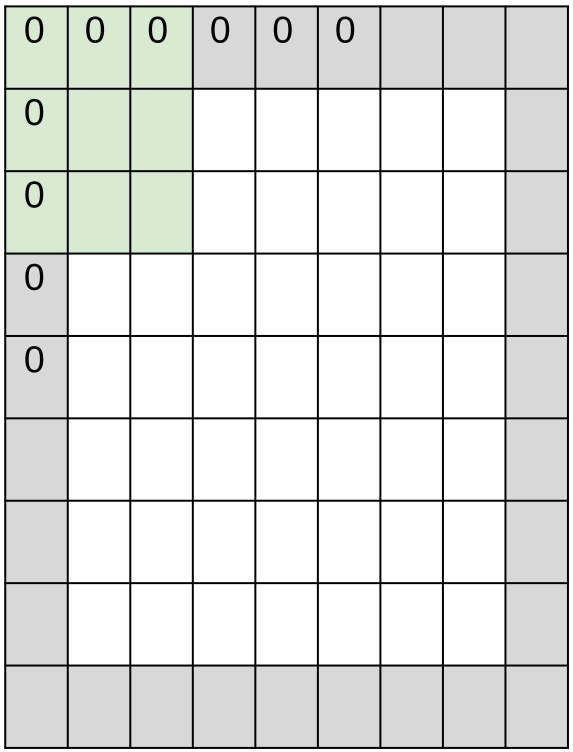


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Convolutions: More detail

In practice: Common to zero pad the border

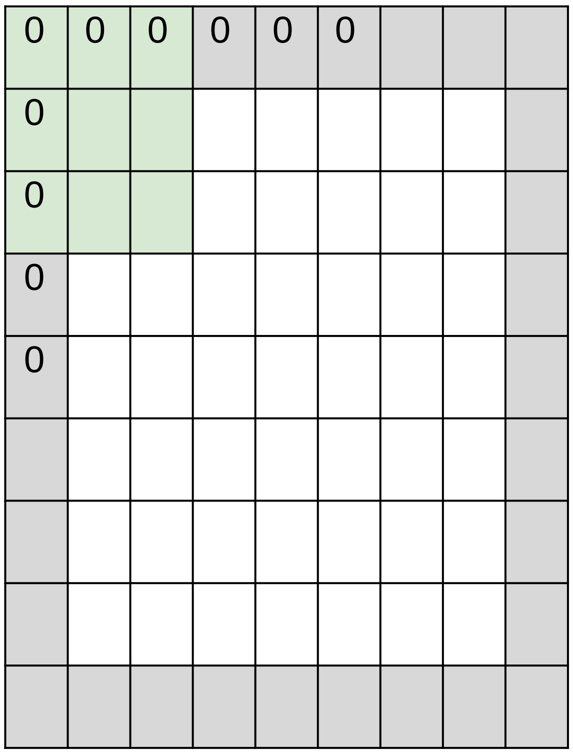


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

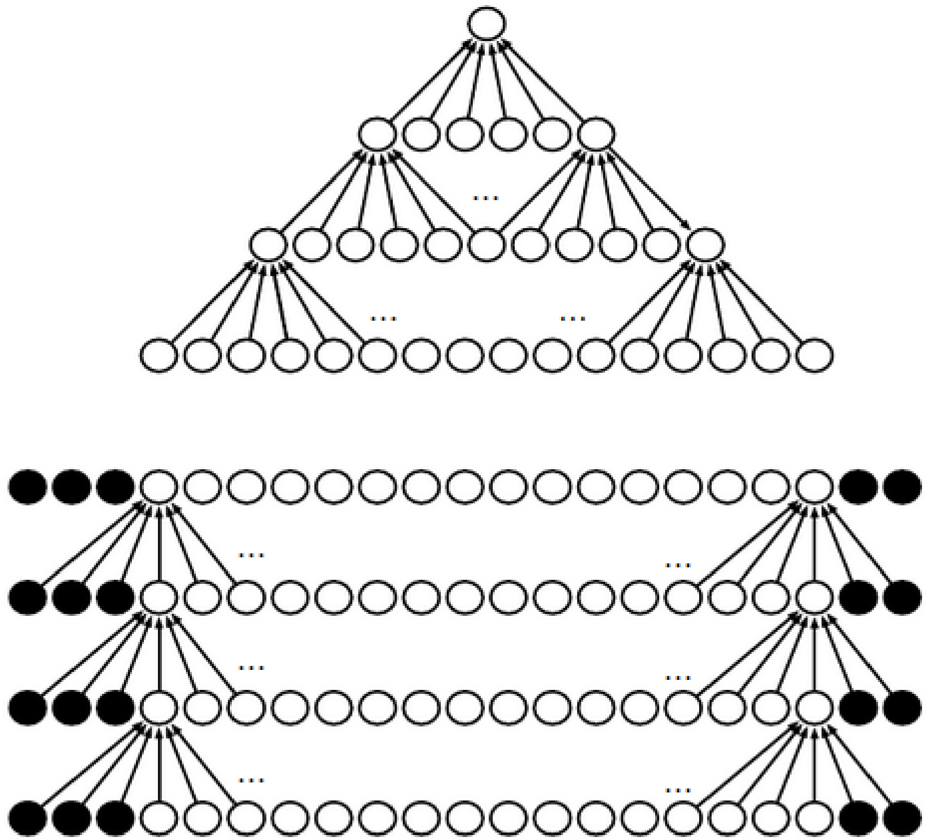
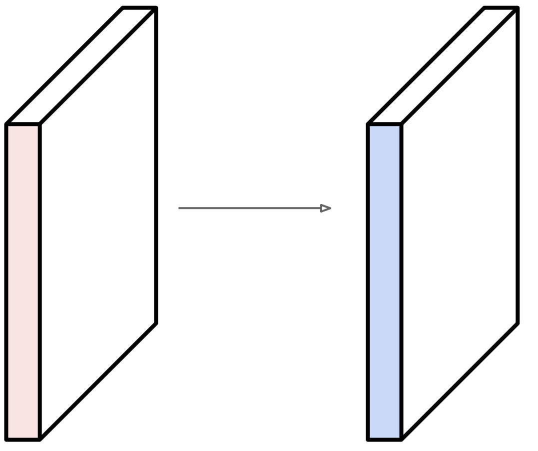


Figure 9.13: The effect of zero padding on network size. Consider a convolutional network with a kernel of width six at every layer. In this example, we do not use any pooling, so only the convolution operation itself shrinks the network size. *(Top)*In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive, and some shrinking is inevitable in this kind of architecture. *(Bottom)*By adding five implicit zeros to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

Convolutions: More detail

Examples time:

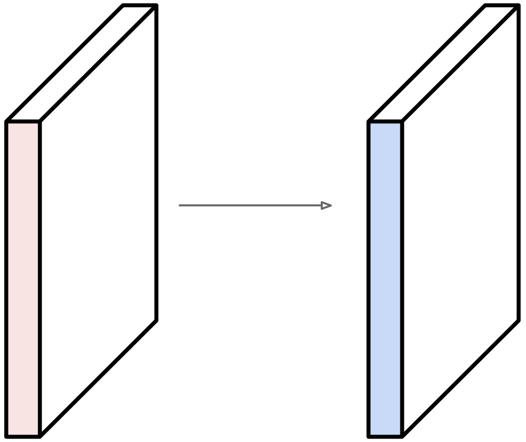


Input volume: **32x32x3**
10 5x5x3 filters with stride 1, pad 2

Output volume size: ?

Convolutions: More detail

Examples time:

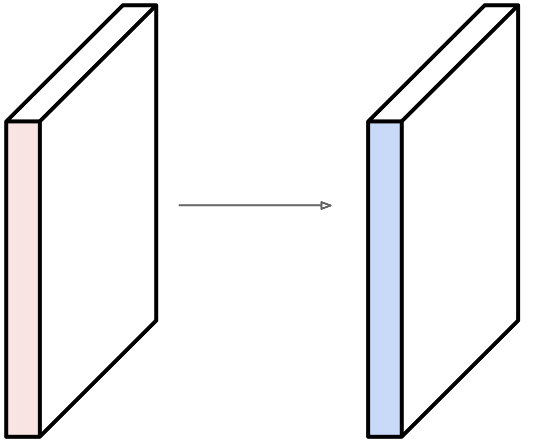


Input volume: **32x32x3**
10 5x5x3 filters with stride 1, pad 2

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

Convolutions: More detail

Examples time:



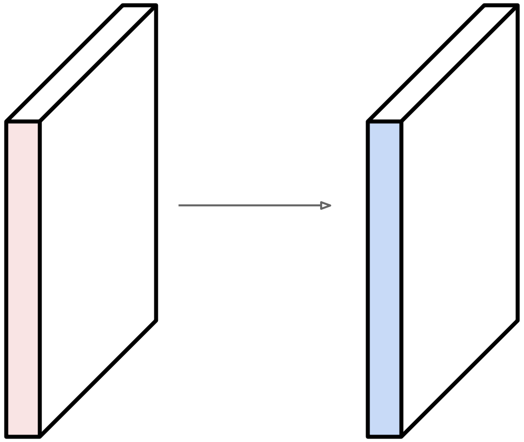
Input volume: **32x32x3**
10 5x5x3 filters with stride 1, pad 2

Number of parameters in this layer?

Andrej Karpathy

Convolutions: More detail

Examples time:



Input volume: **32x32x3**

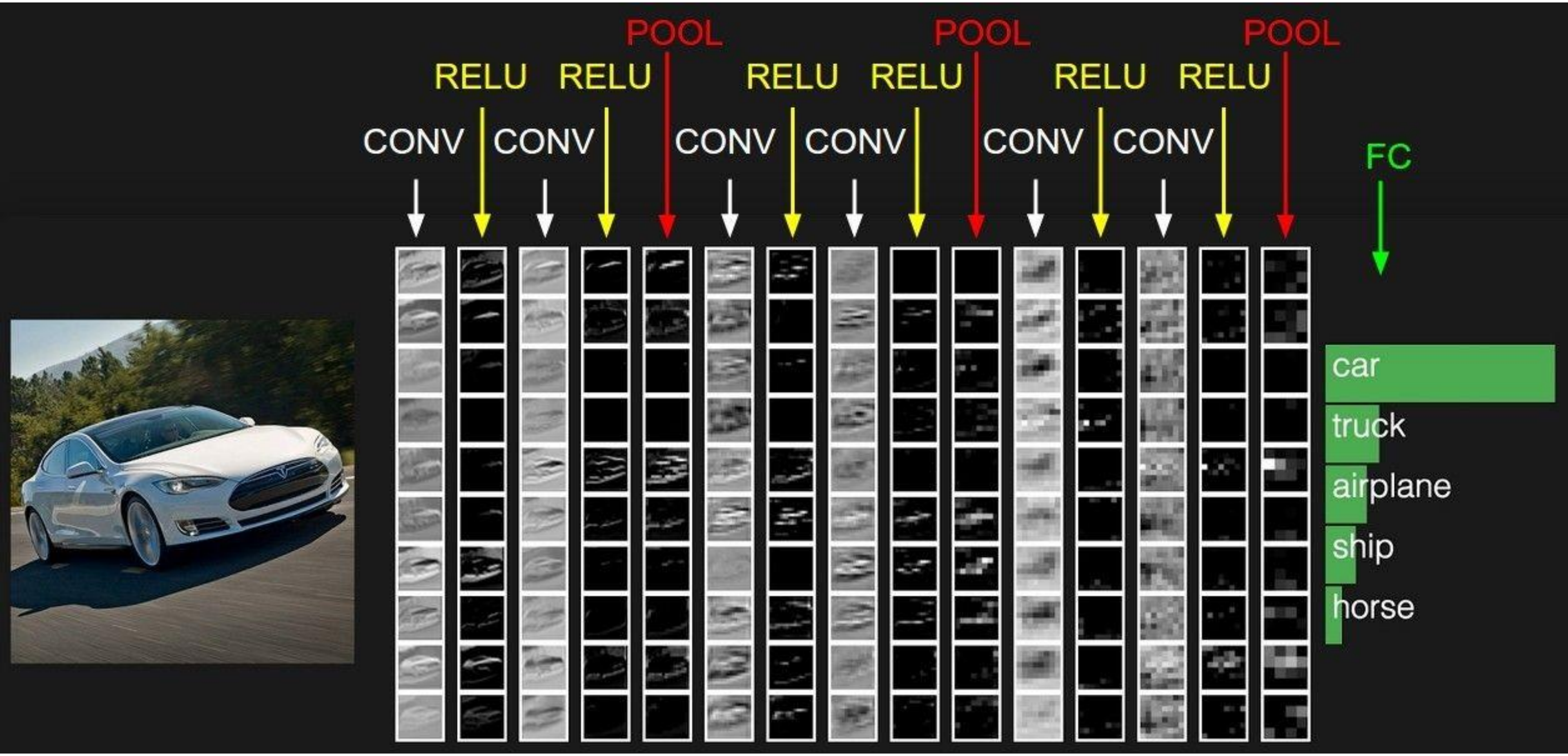
10 5x5x3 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

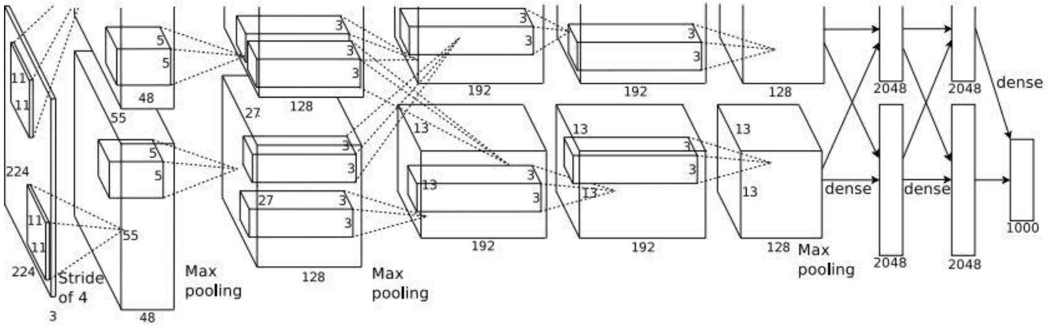
Putting it all together



Some Common Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]

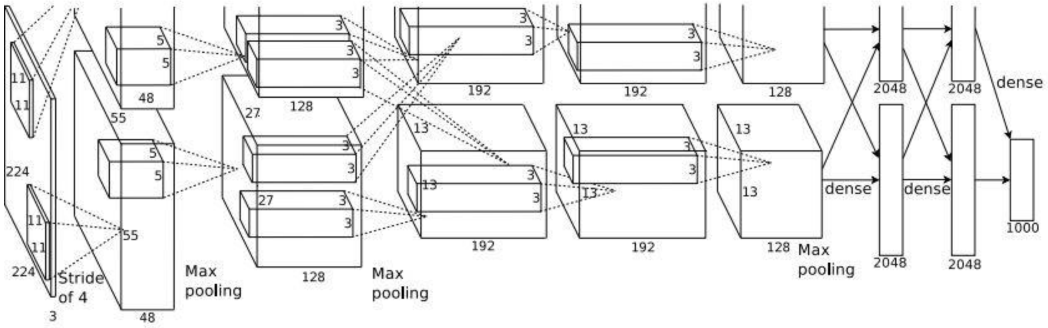


Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Case Study: AlexNet

[Krizhevsky et al. 2012]

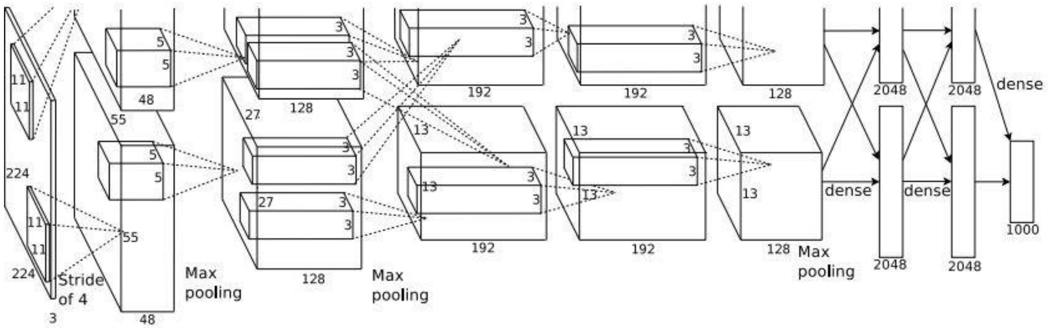


Input: 227x227x3 images

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]



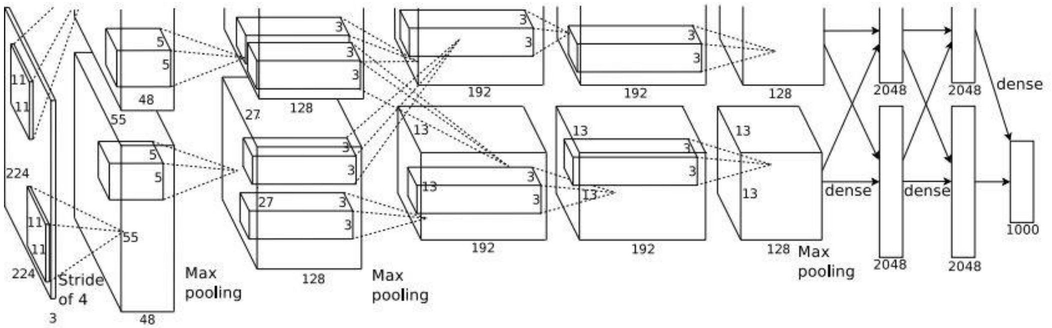
Input: 227x227x3 images
After CONV1: 55x55x96

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

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

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)

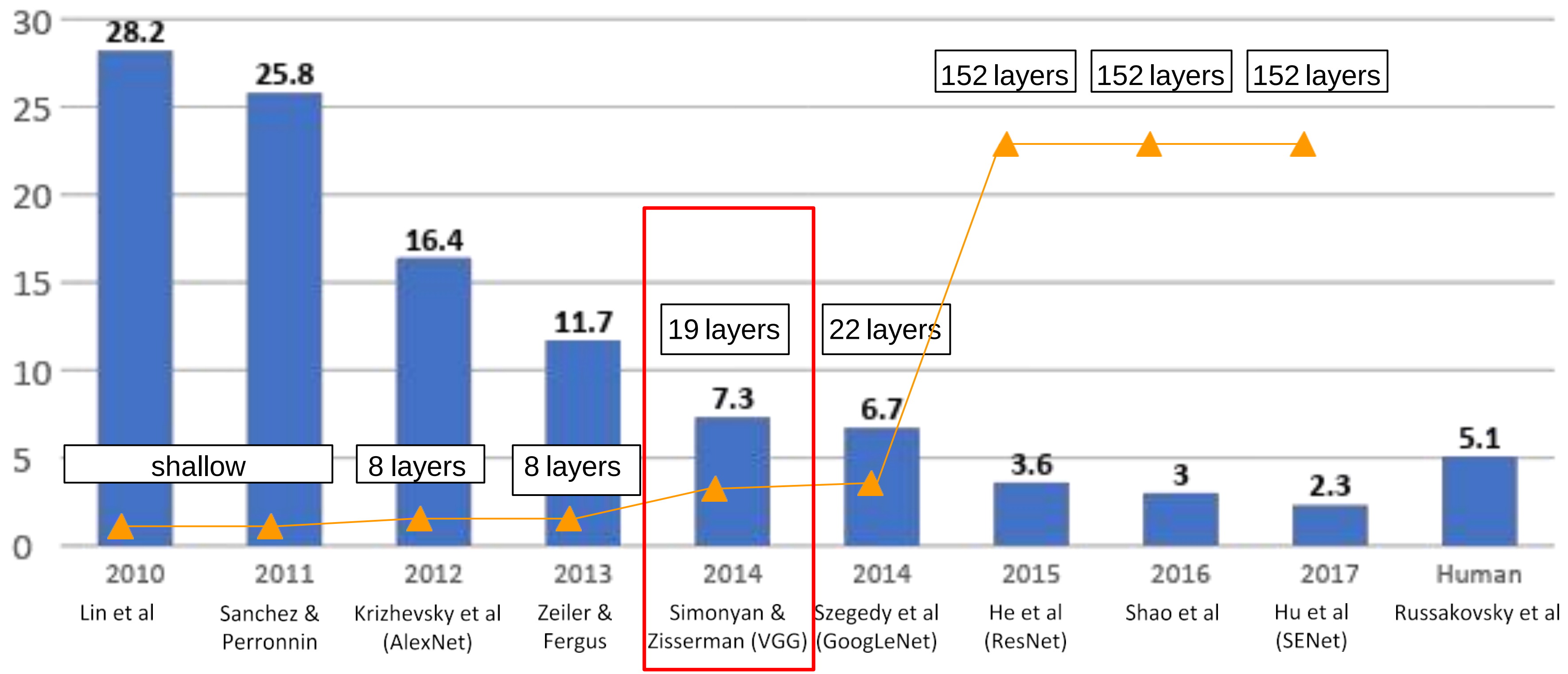
Details/Retrospectives:
-first use of ReLU
-used Norm layers (not common anymore)
-heavy data augmentation
-dropout 0.5
-batch size 128
-SGD Momentum 0.9
-Learning rate 1e-2, reduced by 10
manually when val accuracy plateaus
-L2 weight decay 5e-4

3.3 Local Response Normalization

ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that the following local normalization scheme aids generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

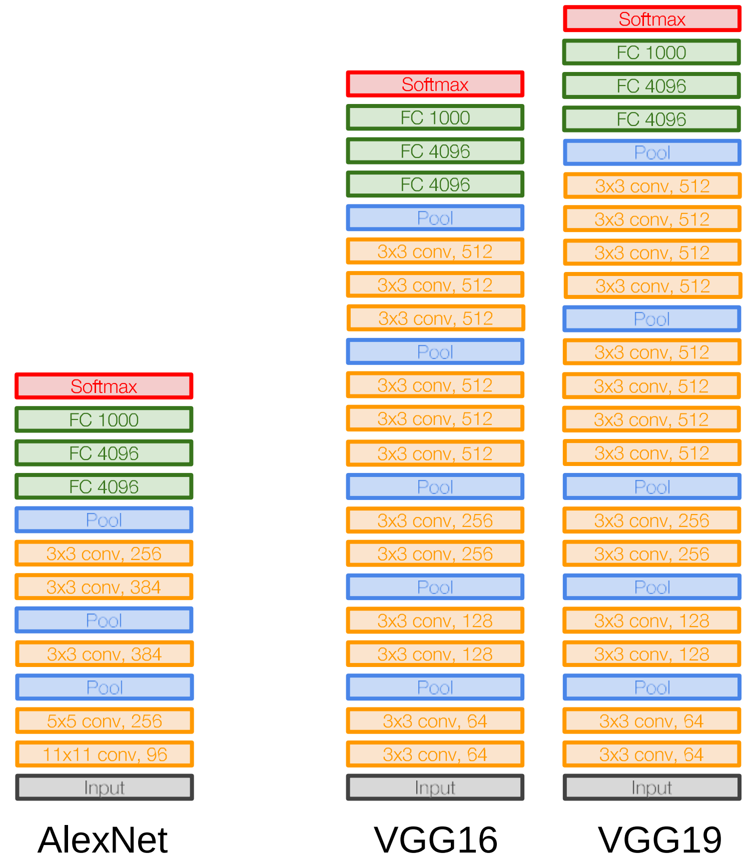
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

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

11.7% top 5 error in ILSVRC’13 (ZFNet)
-> 7.3% top 5 error in ILSVRC’14



Case Study: VGGNet

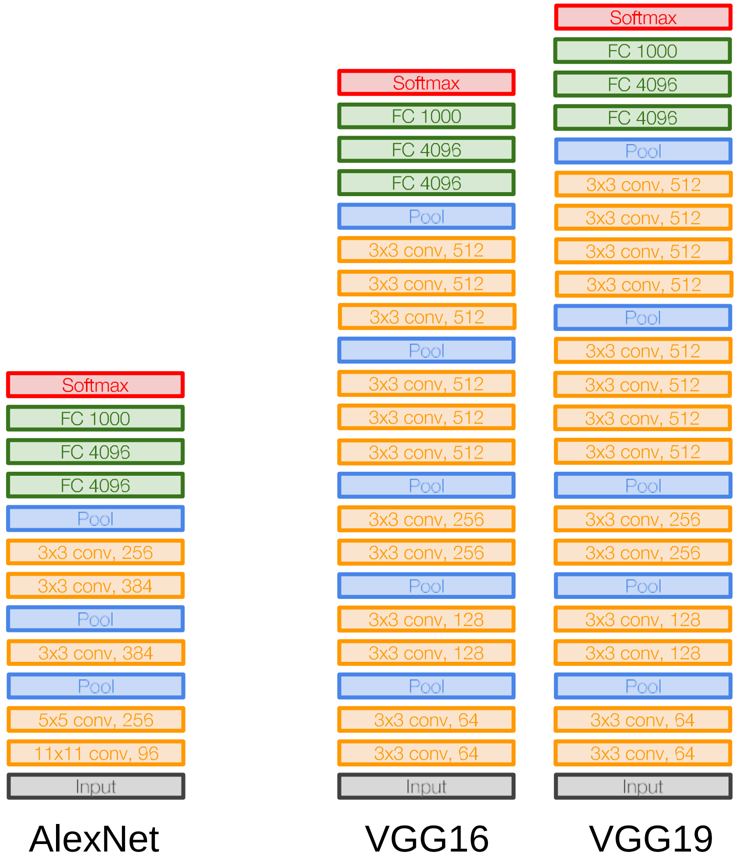
[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

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



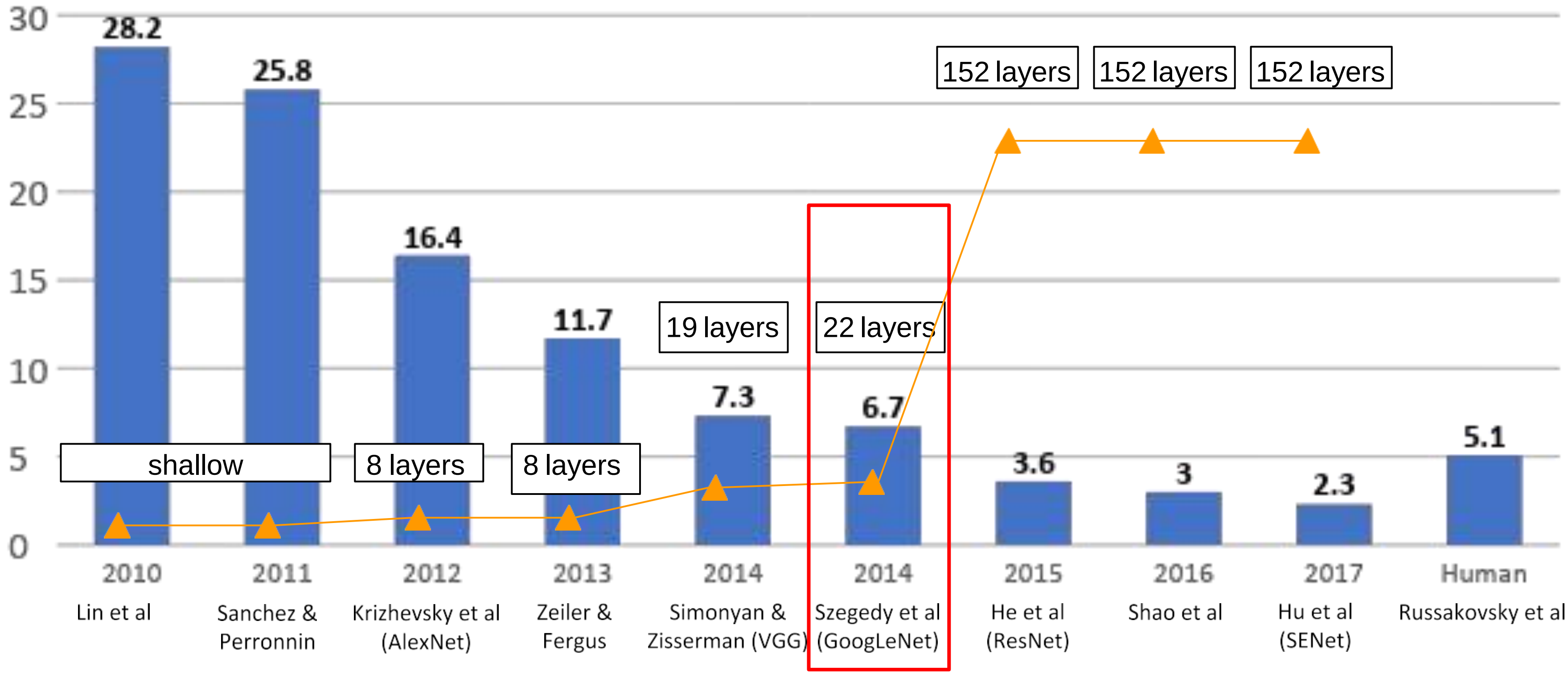
Case Study: VGGNet

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

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters

VGG16

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

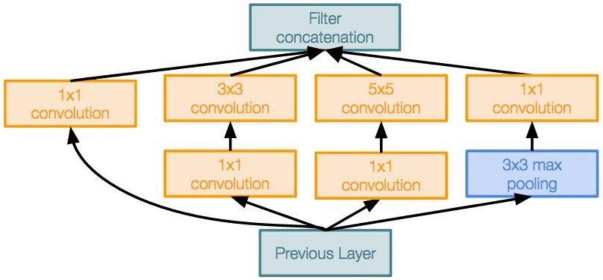


Case Study: GoogLeNet

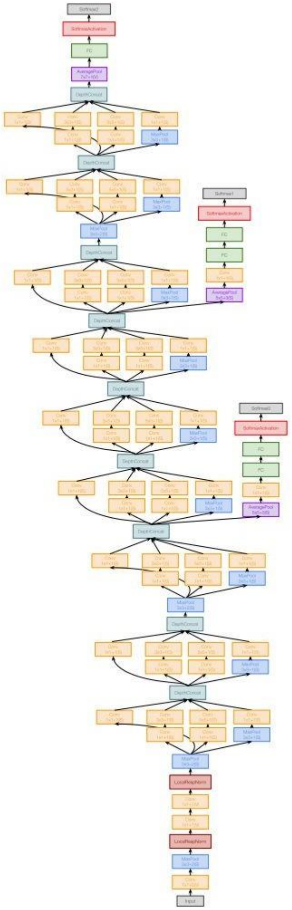
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

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



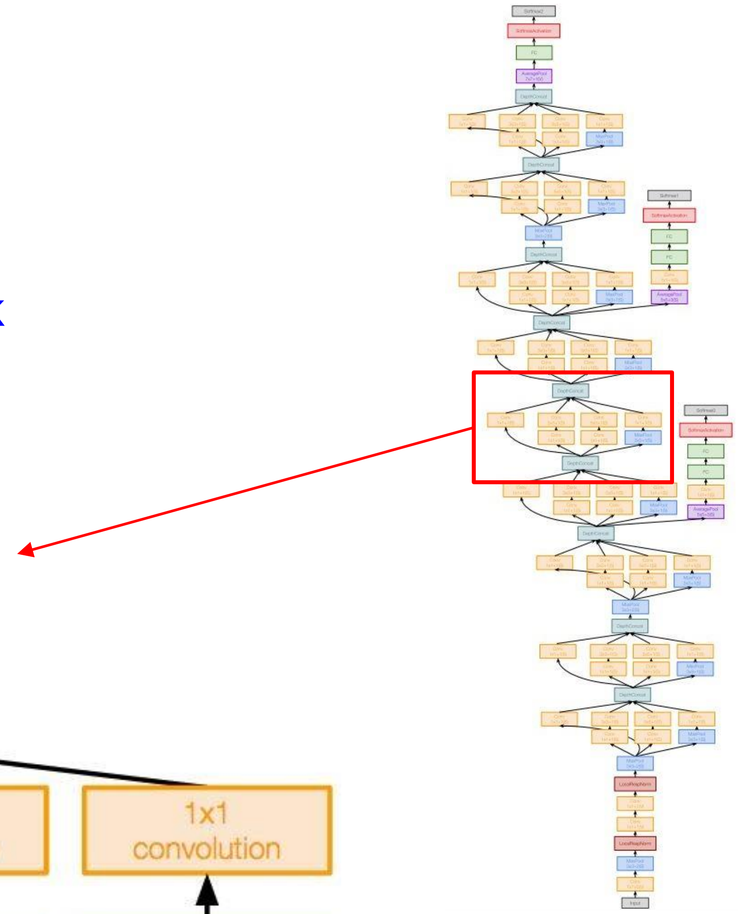
Inception module



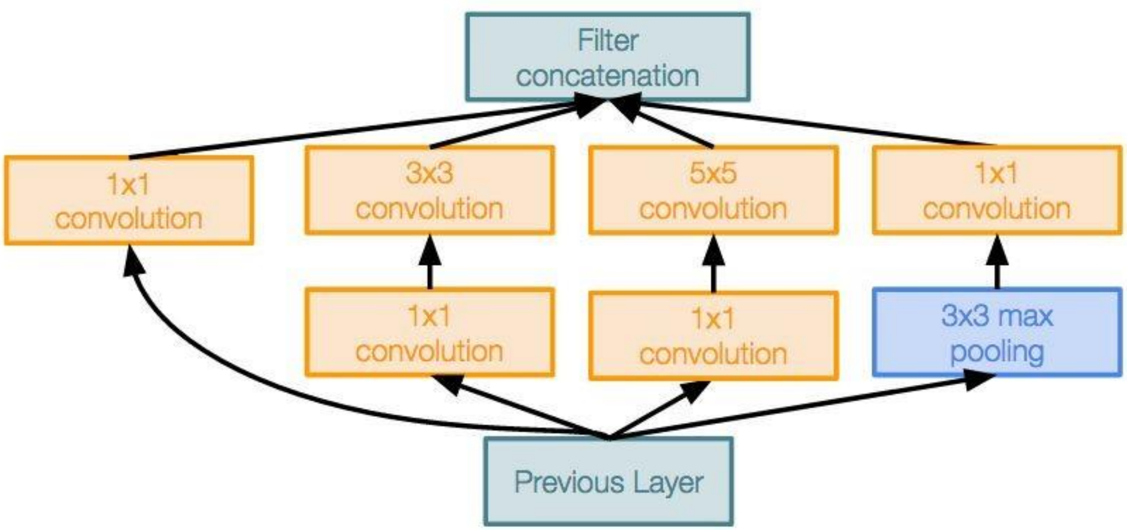
Case Study: GoogLeNet

[Szegedy et al., 2014]

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other

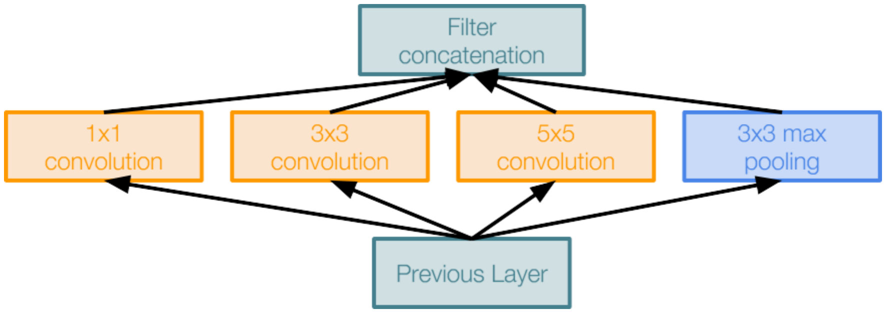


Inception module



Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

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

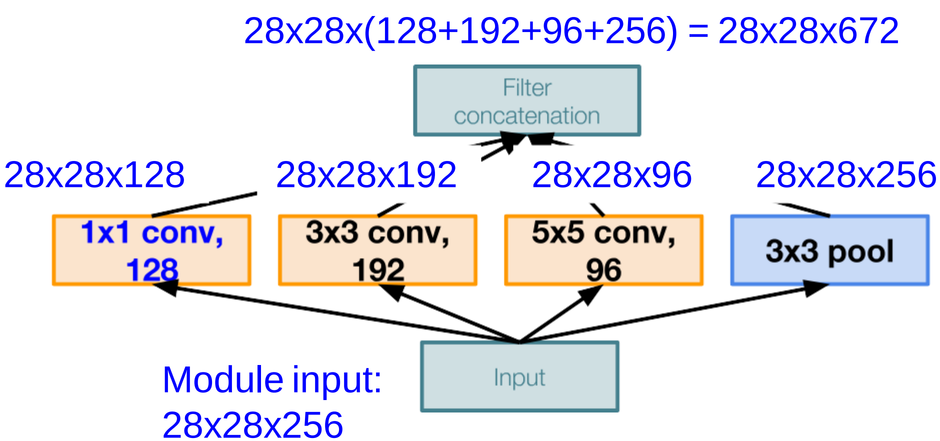
Concatenate all filter outputs together depth-wise

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size
after filter concatenation?



Naive Inception module

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

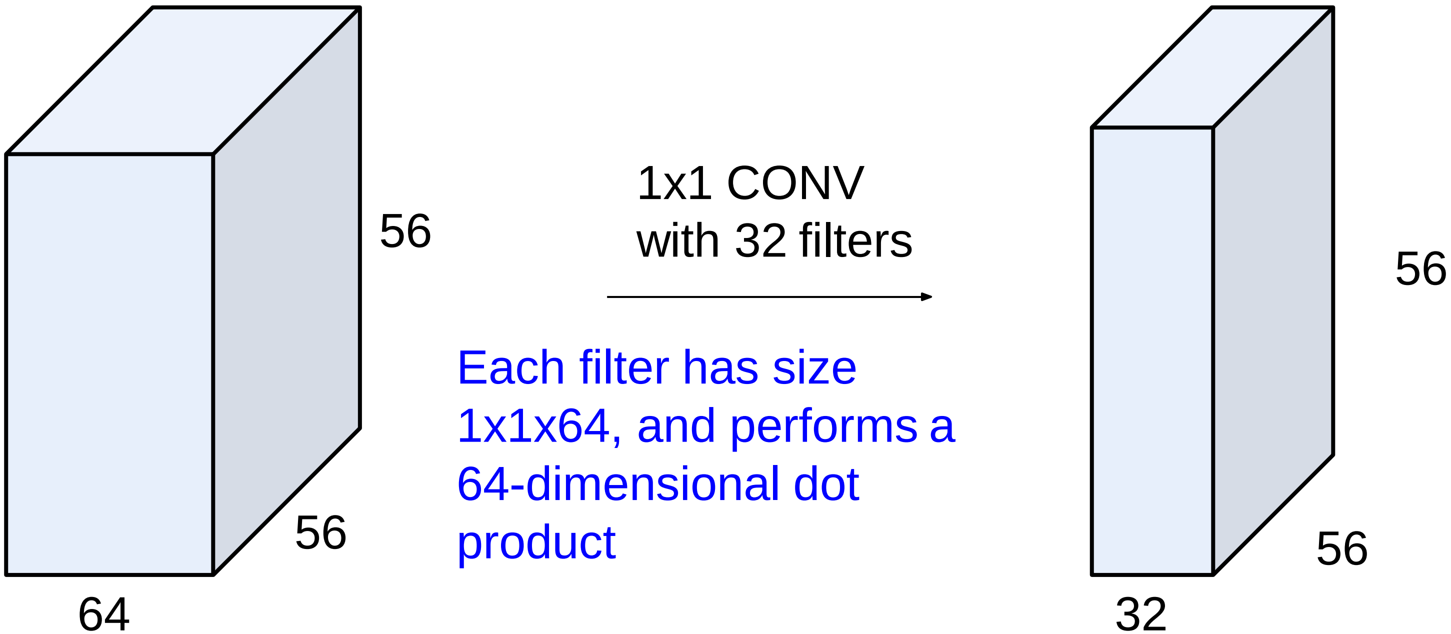
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

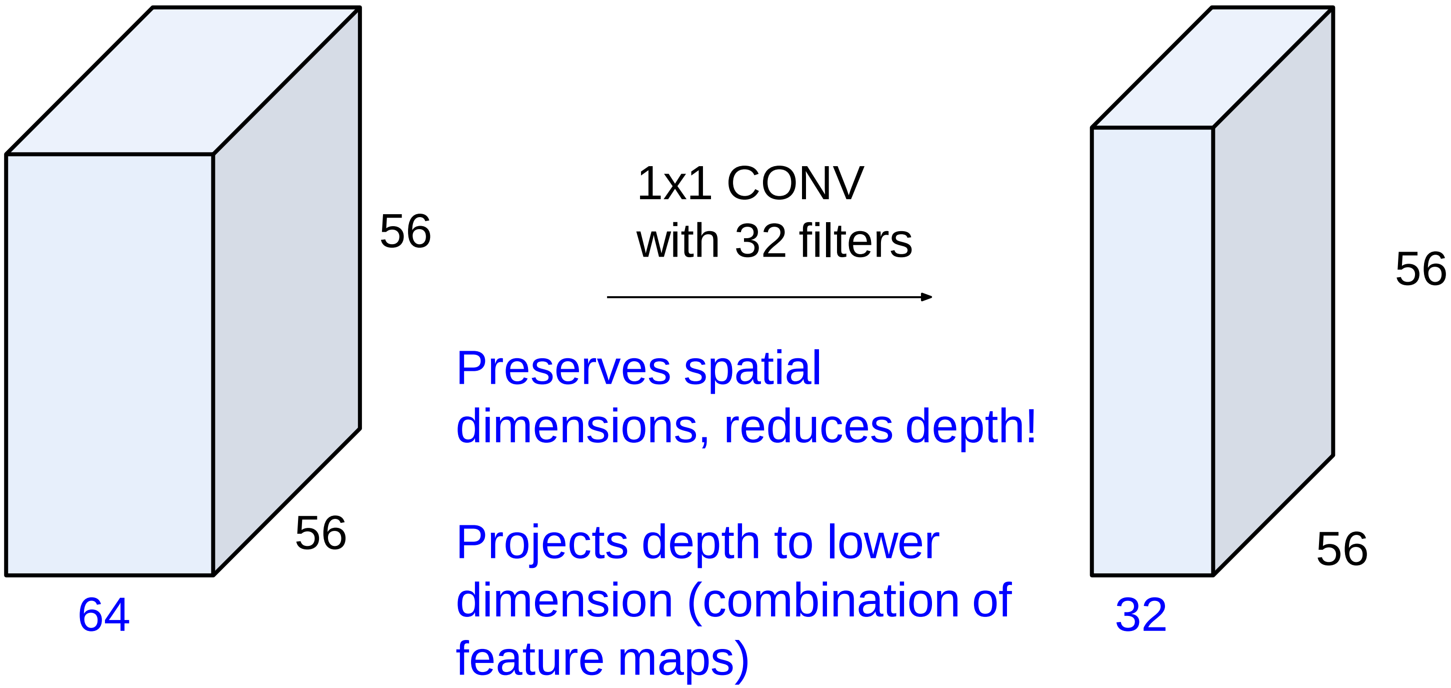
Pooling layer preserves feature depth, which means total depth after concatenation can only grow at every layer!

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

1x1 convolutions



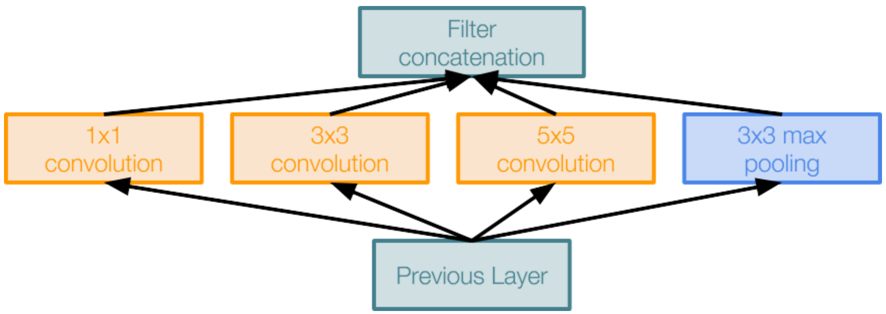
1x1 convolutions



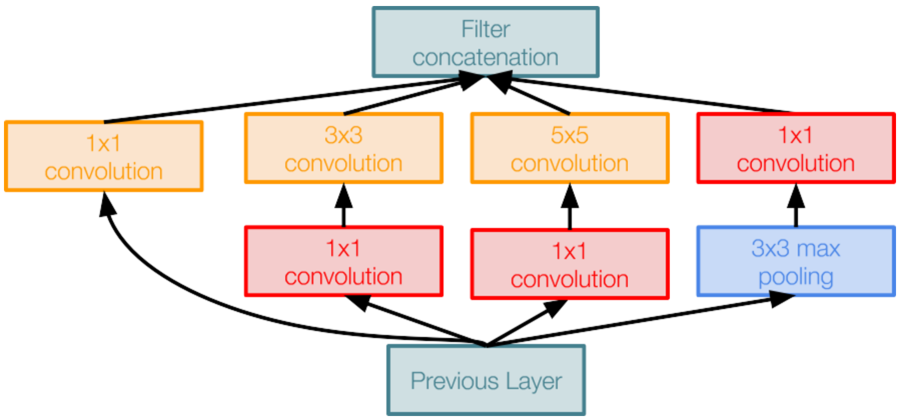
Case Study: GoogLeNet

[Szegedy et al., 2014]

1x1 conv "bottleneck" layers



Naive Inception module



Inception module with dimension reduction

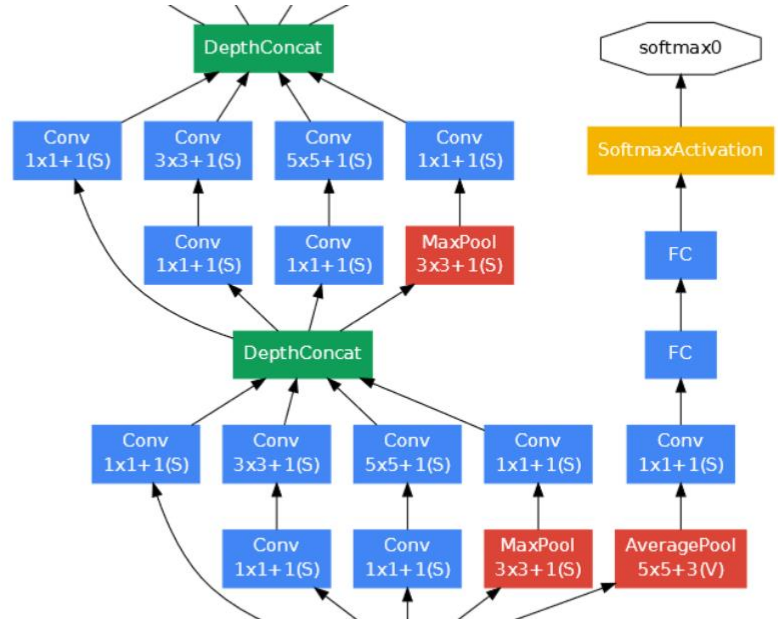
Total: 854M ops

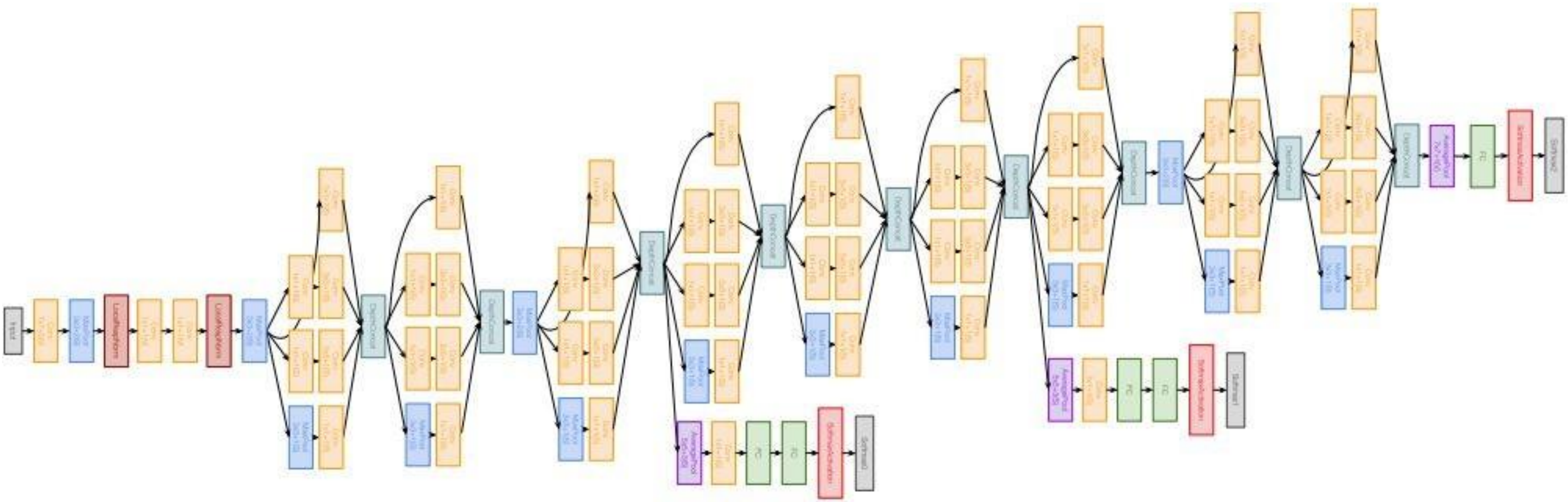
Total: 358M ops

Case Study: GoogLeNet

[Szegedy et al., 2014]

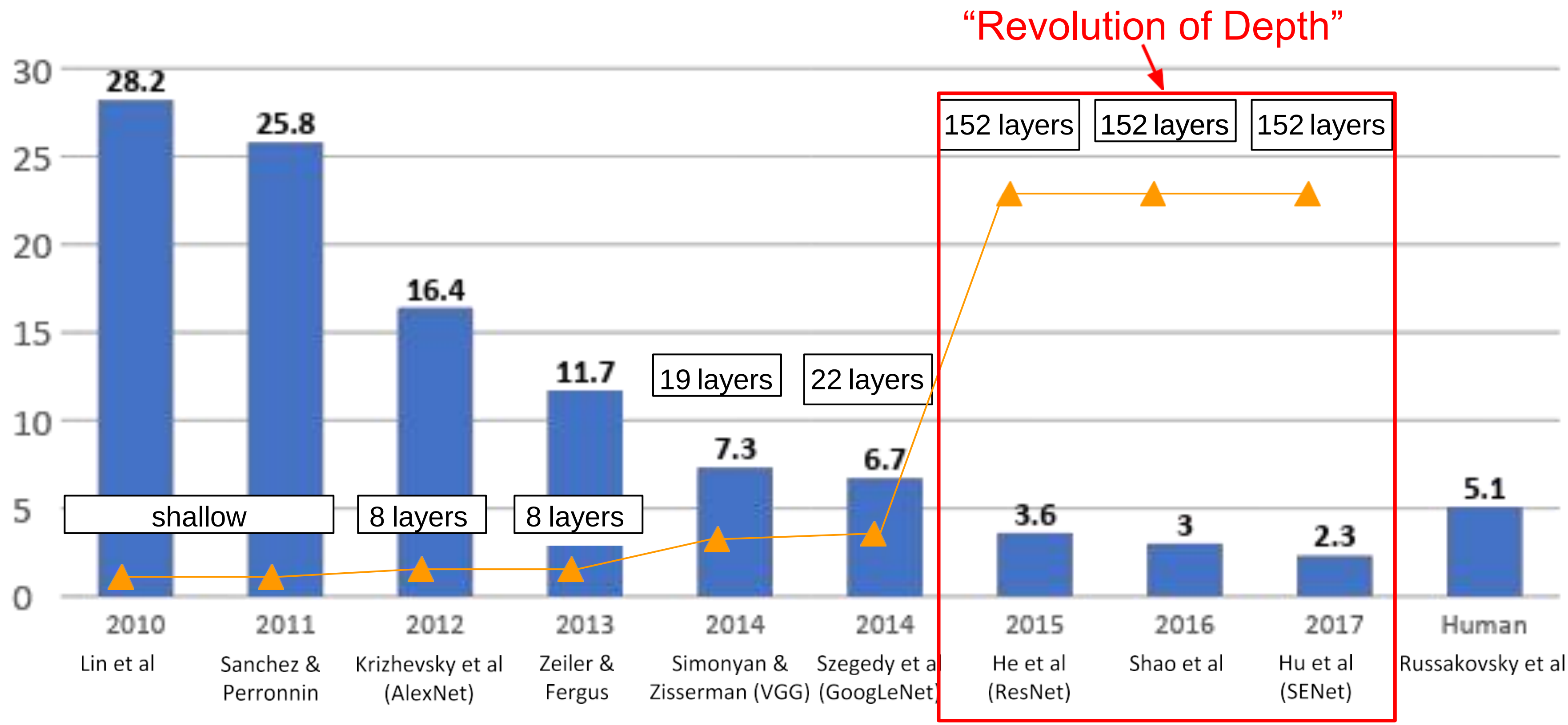
Full GoogLeNet architecture





 https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43022.pdf

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

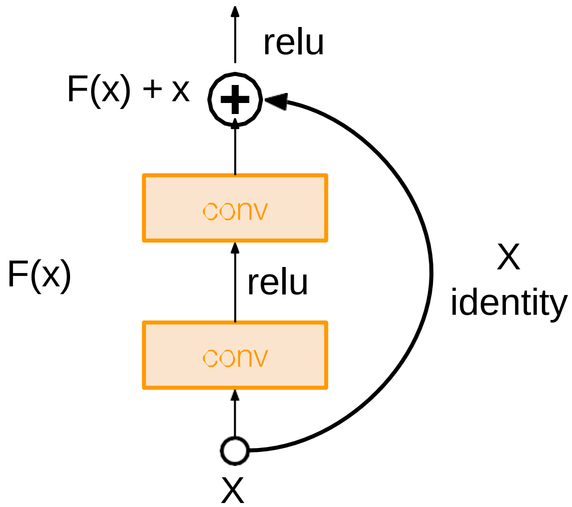


Case Study: ResNet

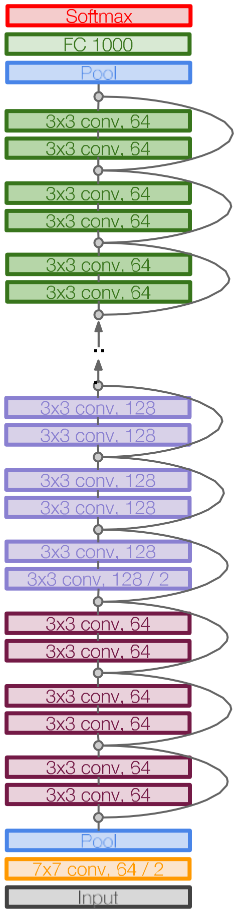
[He et al., 2016]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



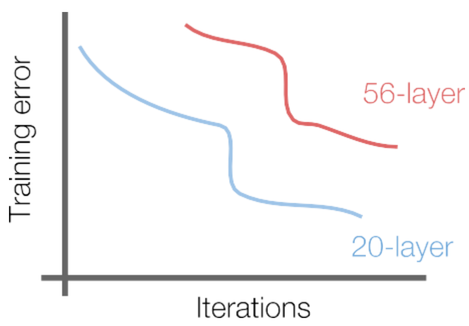
Residual block

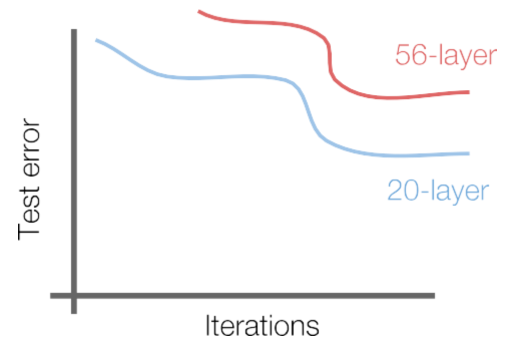


Case Study: ResNet

[He et al., 2016]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





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

56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Case Study: ResNet

[He et al., 2016]

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

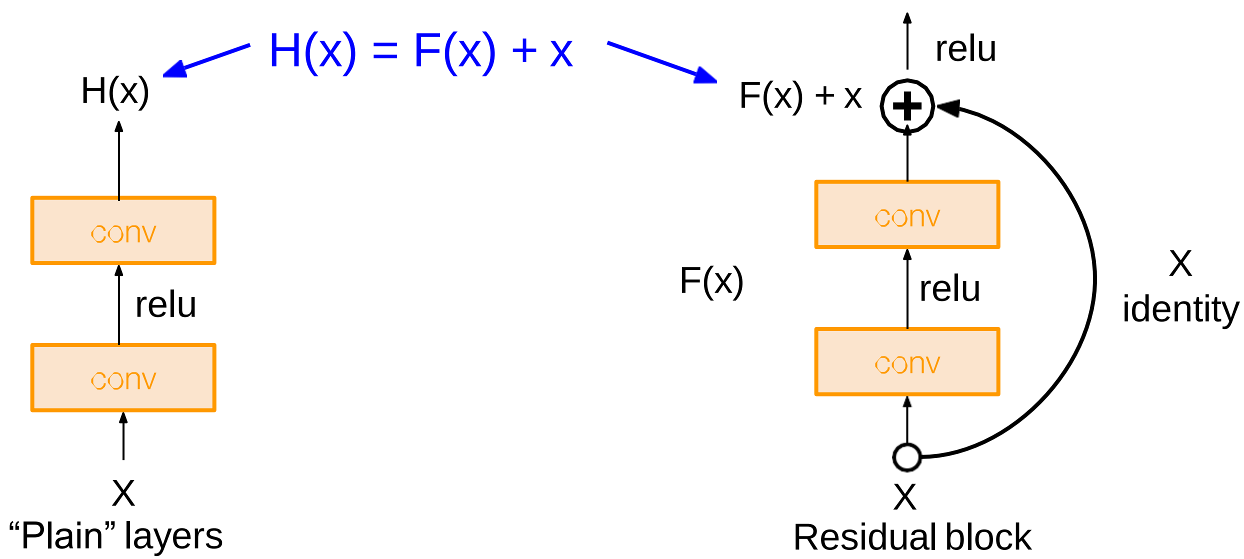
The deeper model should be able to perform at least as well as the shallower model.

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

Case Study: ResNet

[He et al., 2016]

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



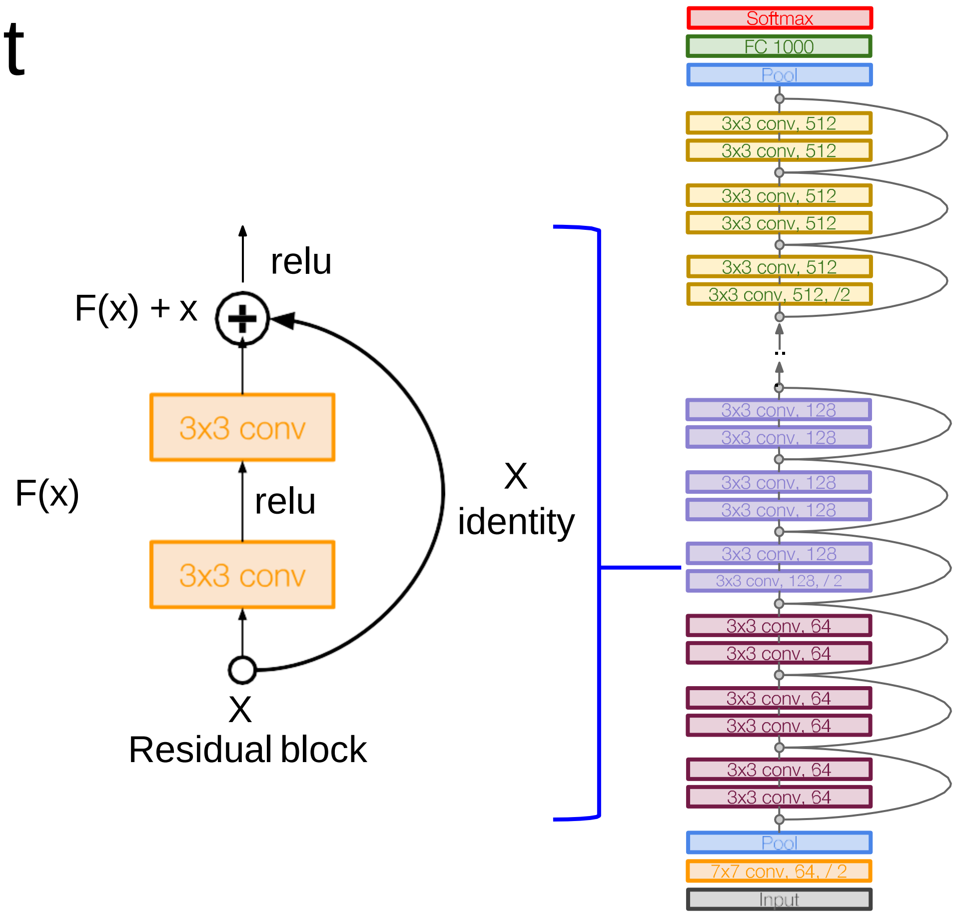
Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

Case Study: ResNet

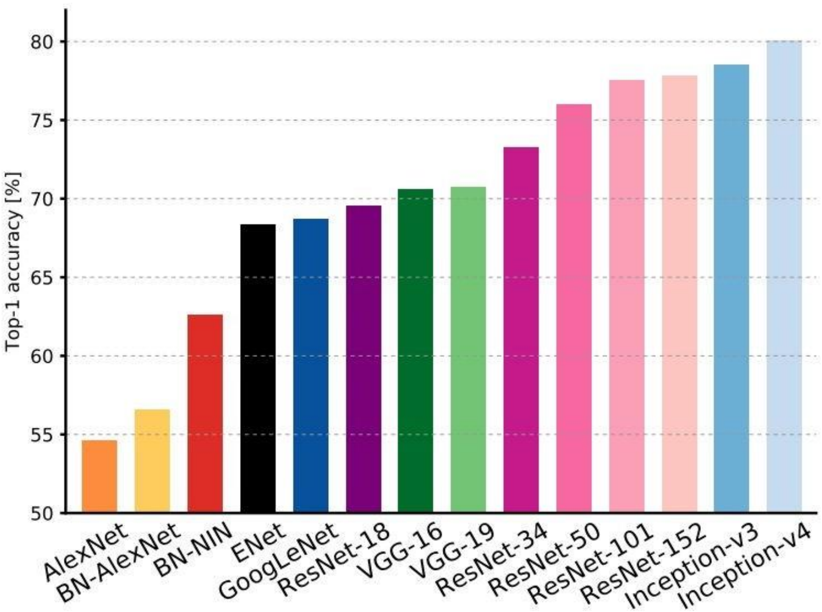
[He et al., 2016]

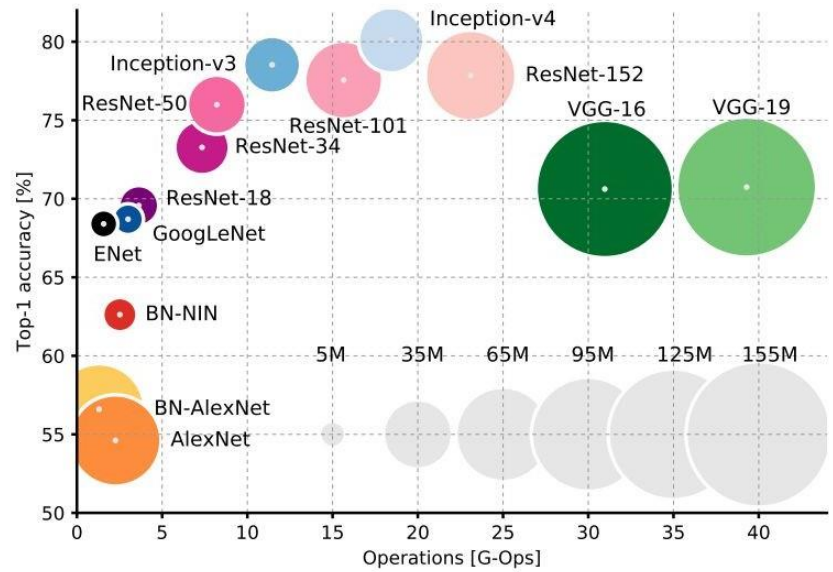
Full ResNet architecture:

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



Comparing complexity...





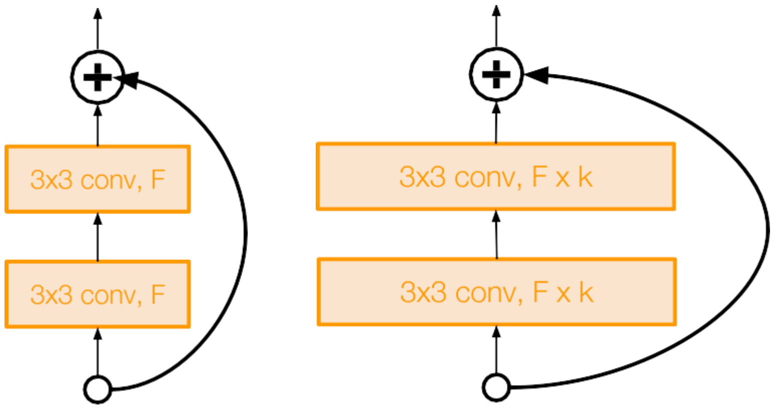
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Improving ResNets...

Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



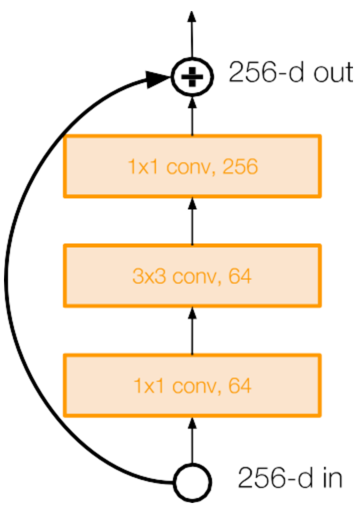
Basic residual block   Wide residual block

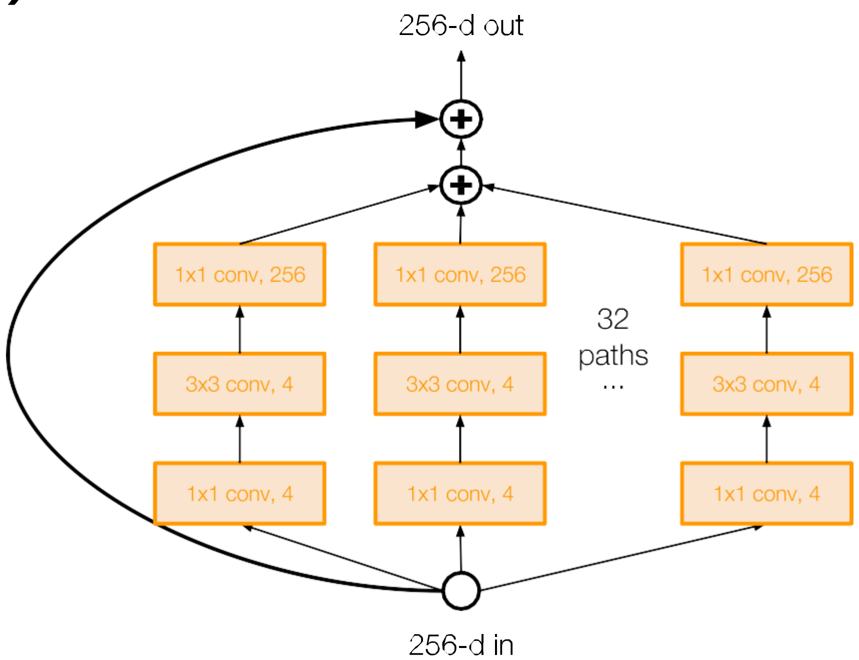
Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways ("cardinality")
- Parallel pathways similar in spirit to Inception module



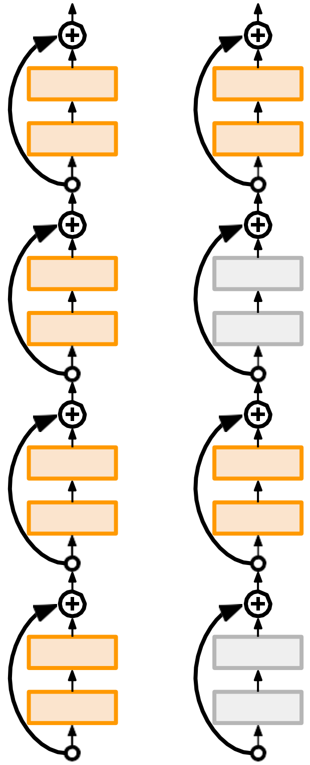


Improving ResNets...

Deep Networks with Stochastic Depth

[Huang et al. 2016]

- Motivation: reduce vanishing gradients and training time through short networks during training
- Randomly drop a subset of layers during each training pass
- Bypass with identity function
- Use full deep network at test time

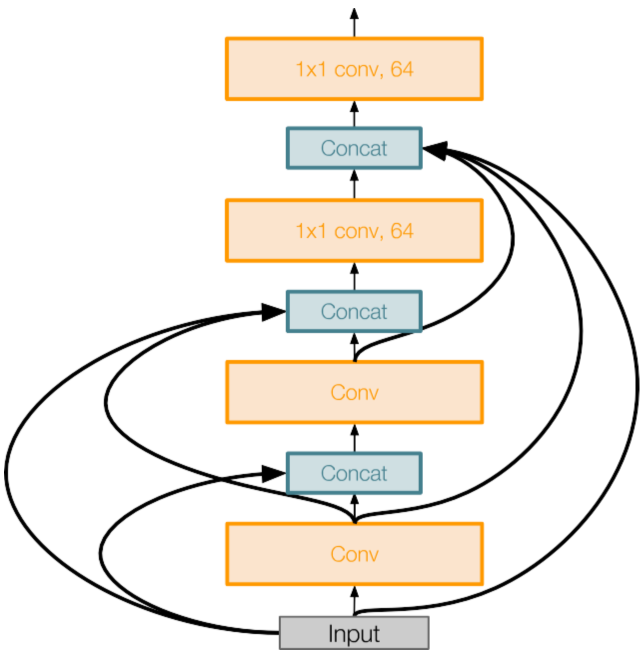


Beyond ResNets...

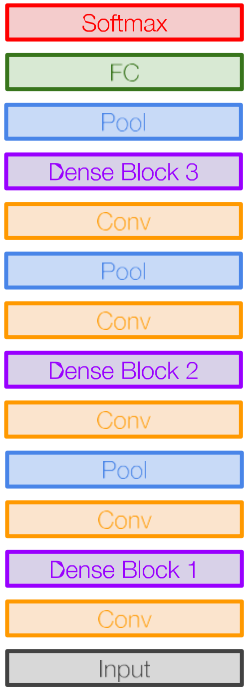
Densely Connected Convolutional Networks

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



Dense Block



Summary: CNN Architectures

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

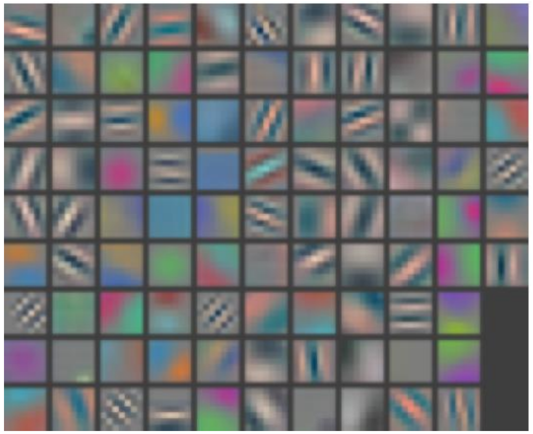
- Wide ResNet
- ResNeXT
- DenseNet

Summary: CNN Architectures

- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Efforts to investigate necessity of depth vs. width and residual connections
- Even more recent trend towards *meta-learning* (e.g. learning what architecture should be)

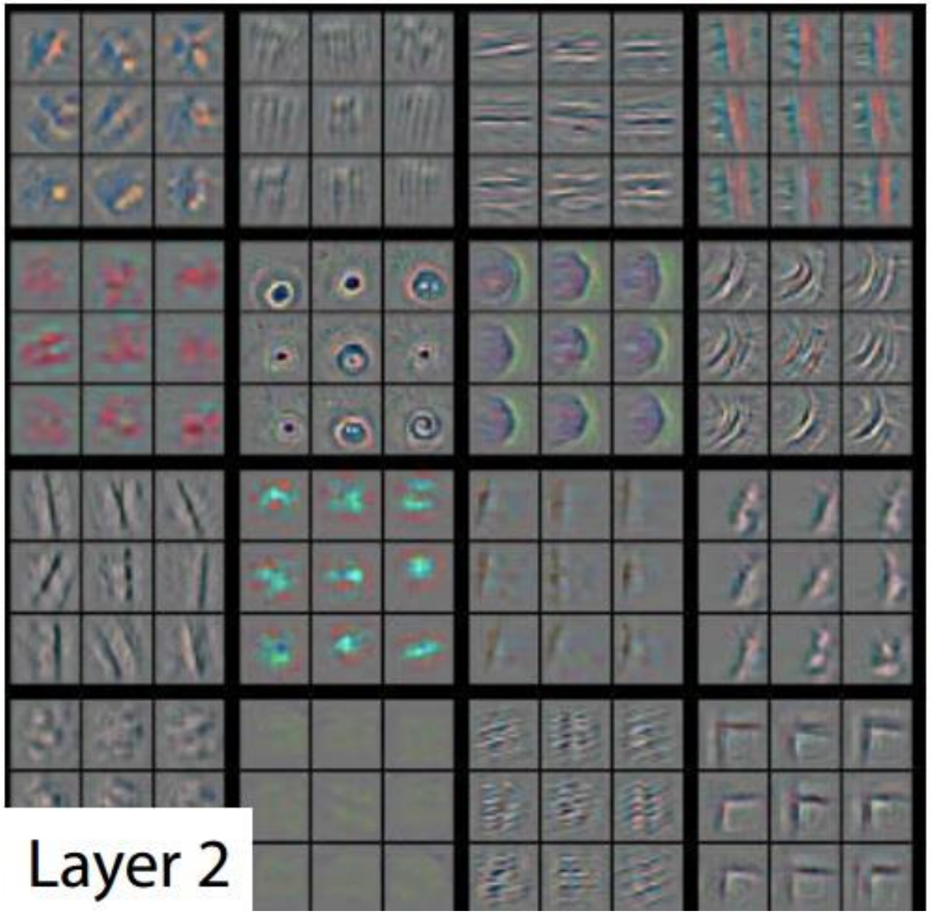
Understanding CNNs

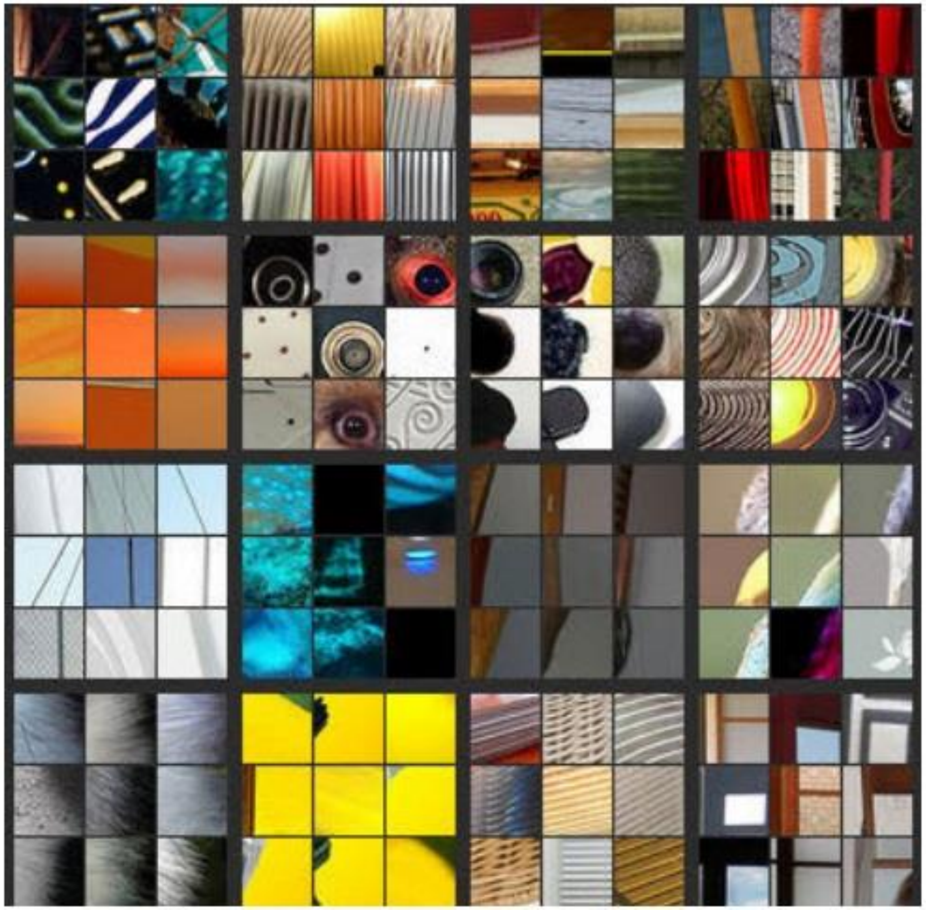
Layer 1



Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 2

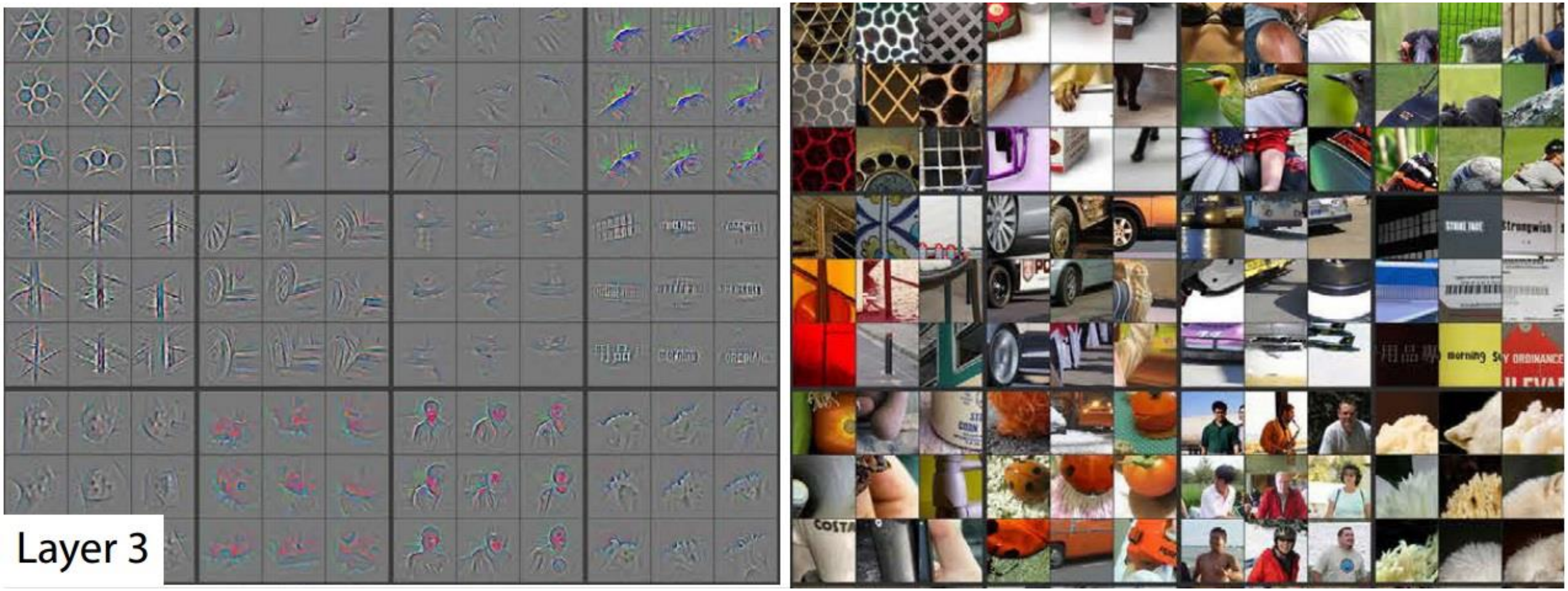




- Activations projected down to pixel level via decovolution
- Patches from validation images that give maximal activation of a given feature map

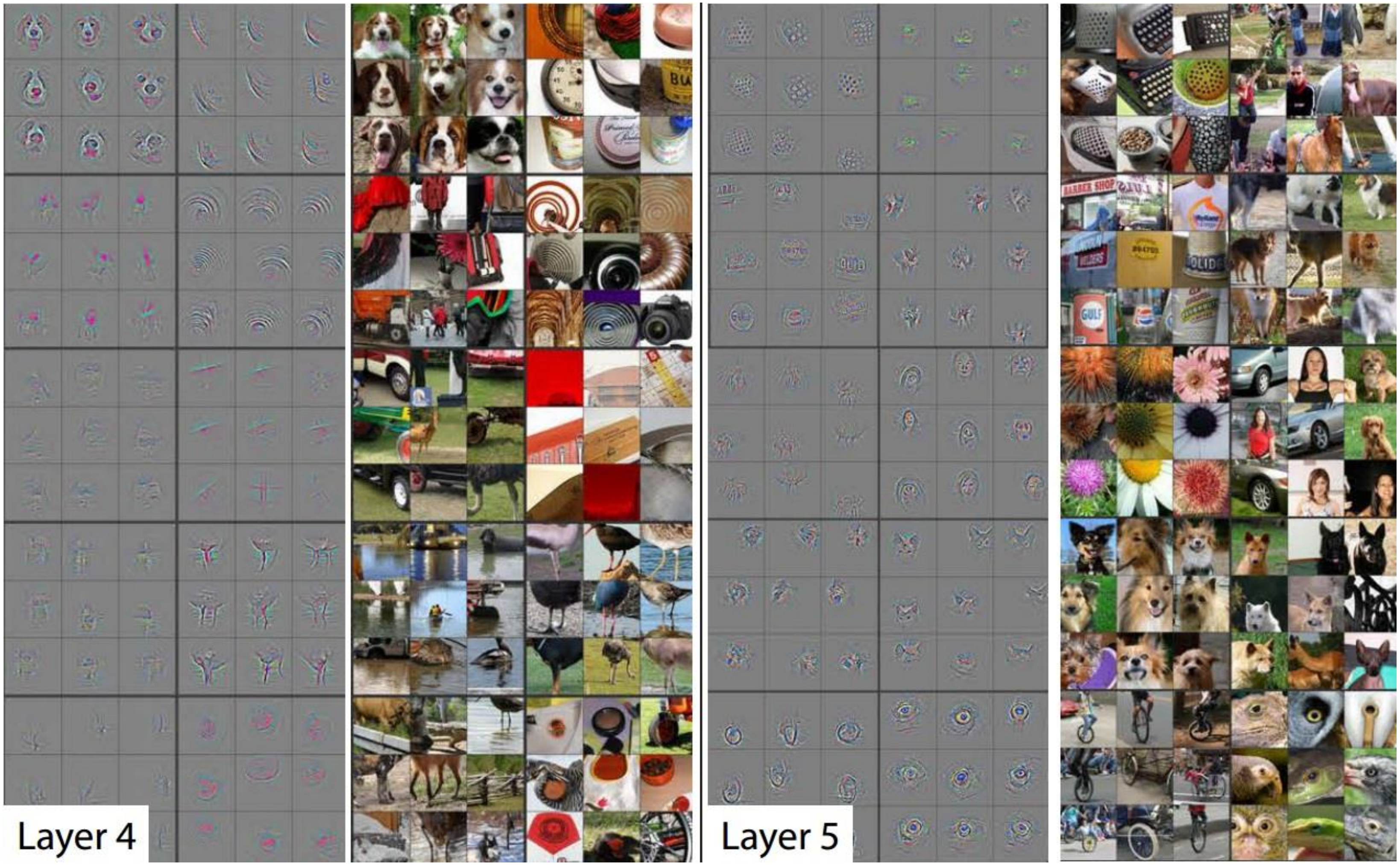
Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 3



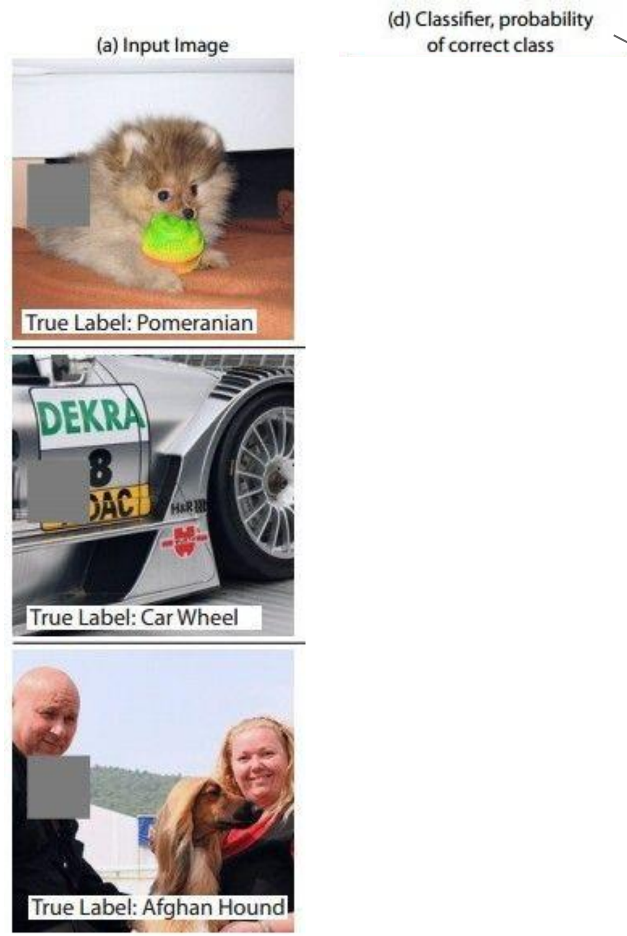
Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 4 and 5



Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

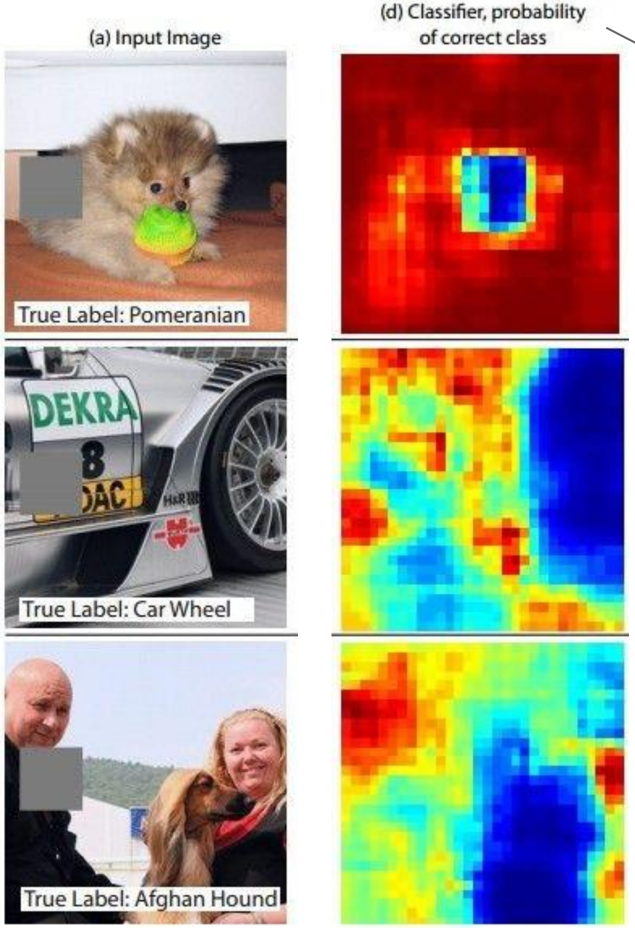
Occlusion experiments



(as a function of the position of the square of zeros in the original image)

[Zeiler & Fergus 2014]

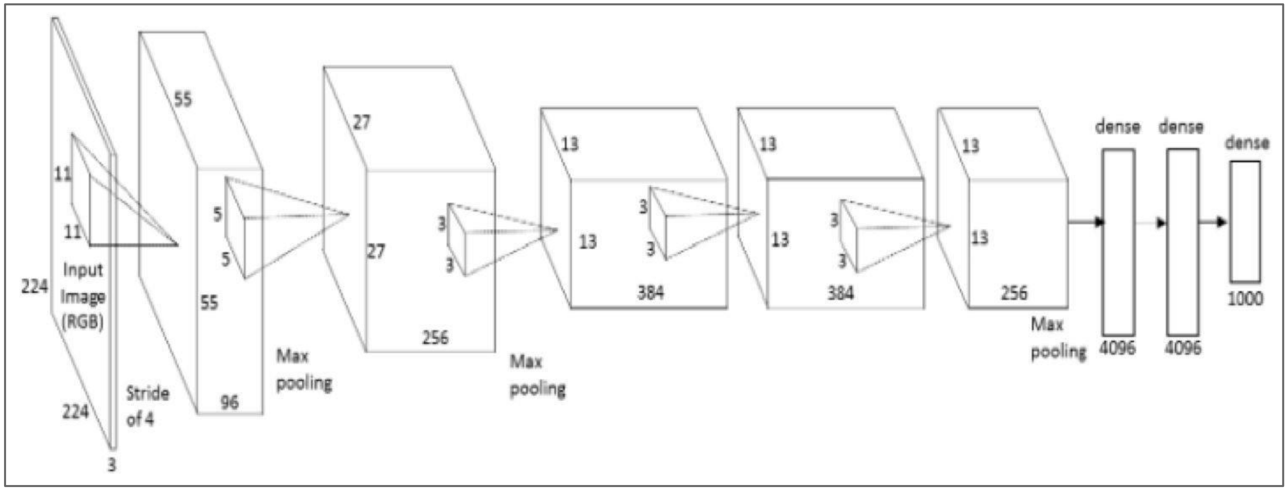
Occlusion experiments



(as a function of the position of the square of zeros in the original image)

[Zeiler & Fergus 2014]

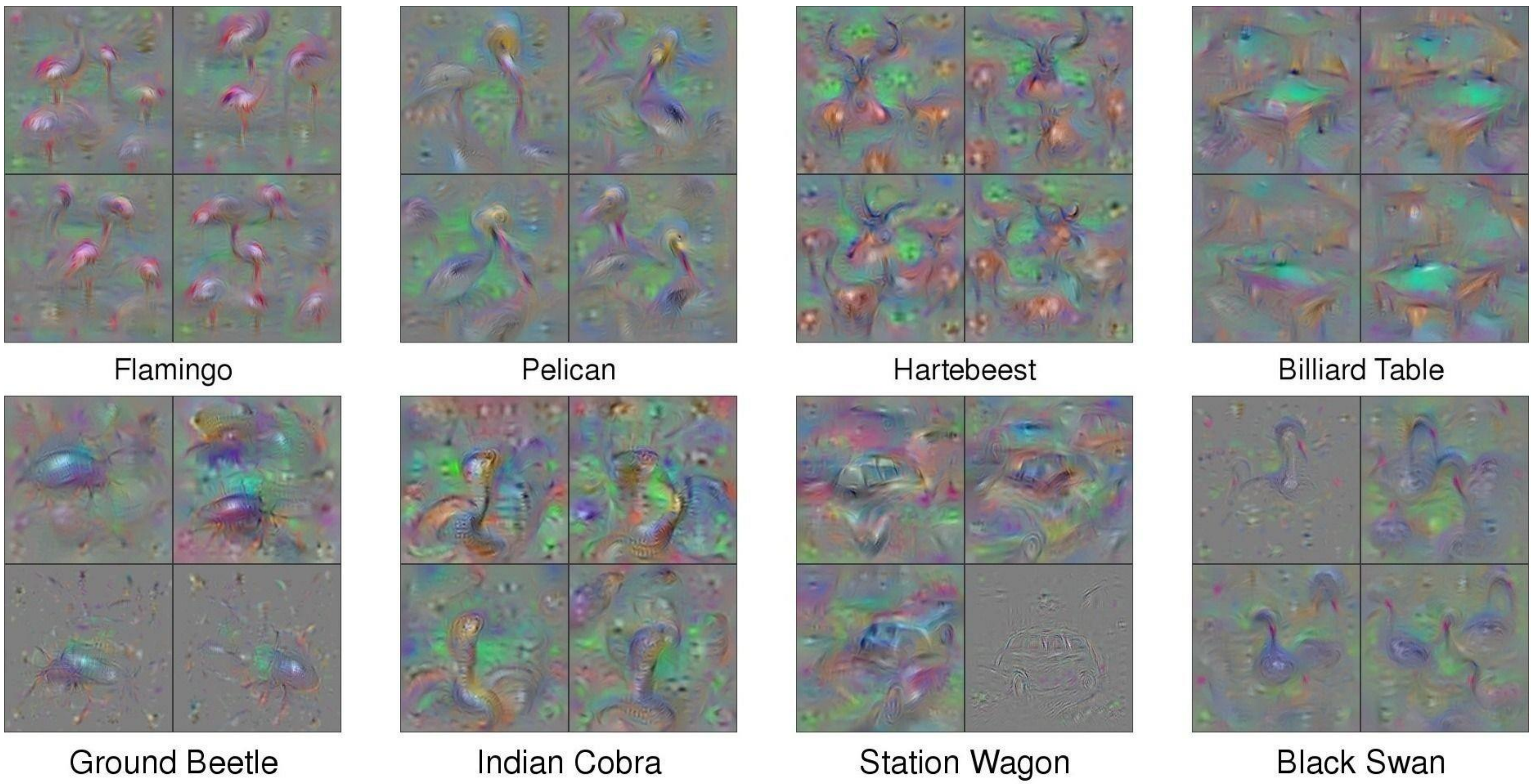
What image maximizes a class score?



Repeat:

1. Forward an image
2. Set activations in layer of interest to all zero, except for a 1.0 for a neuron of interest
3. Backprop to image
4. Do an “image update”

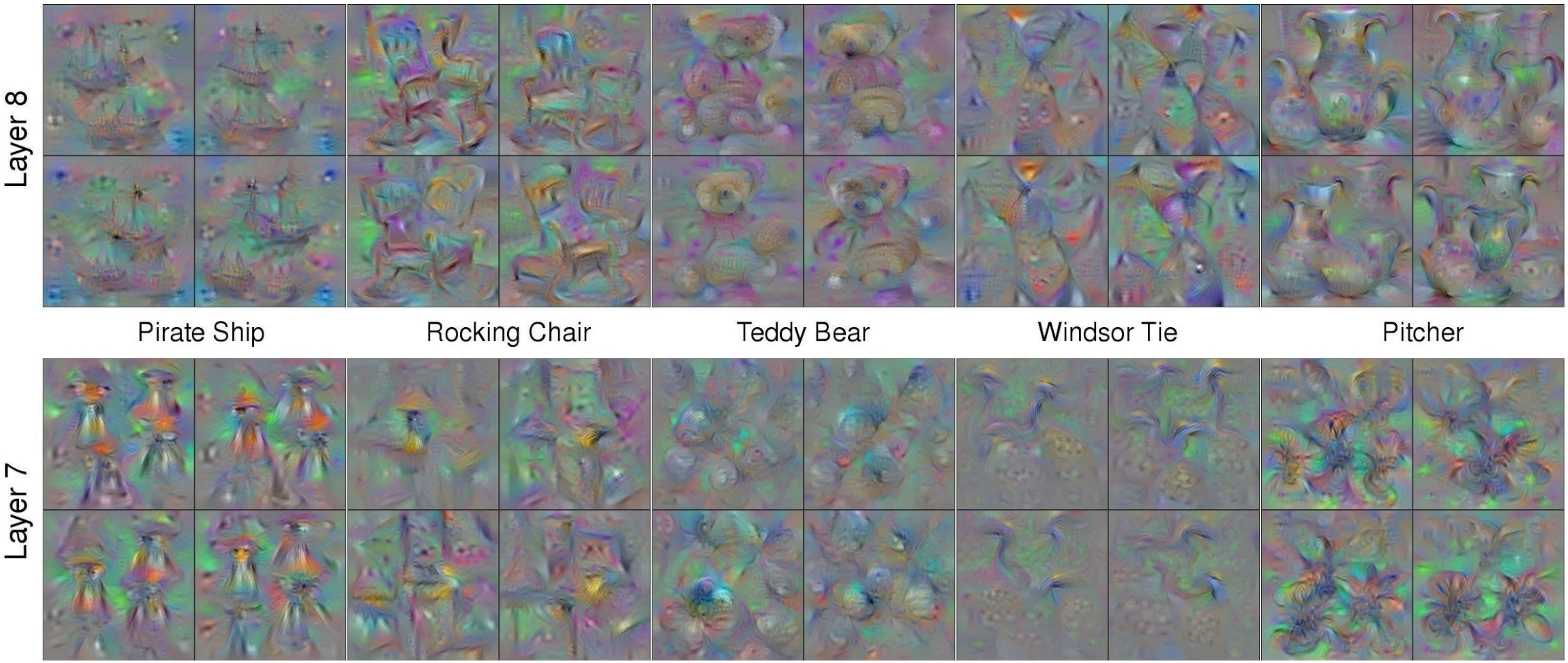
What image maximizes a class score?



[Understanding Neural Networks Through Deep Visualization, Yosinski et al. , 2015]

http://yosinski.com/deepvis

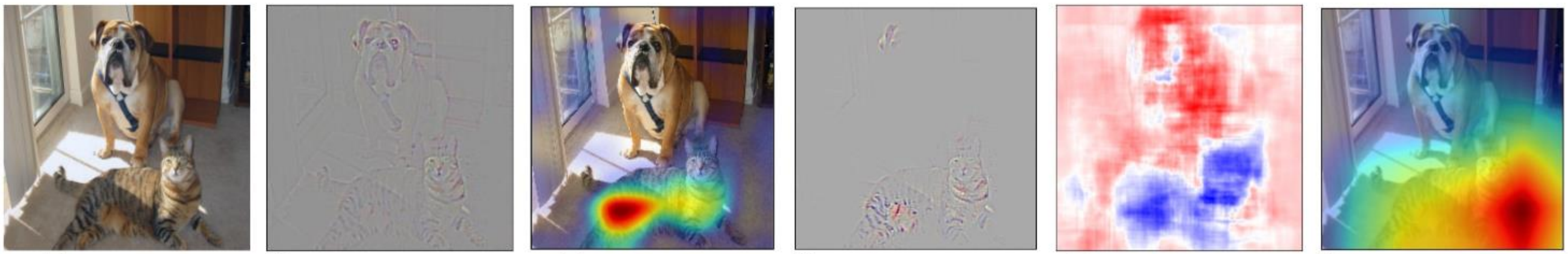
What image maximizes a class score?



Andrej Karpathy

GradCAM

Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization 3



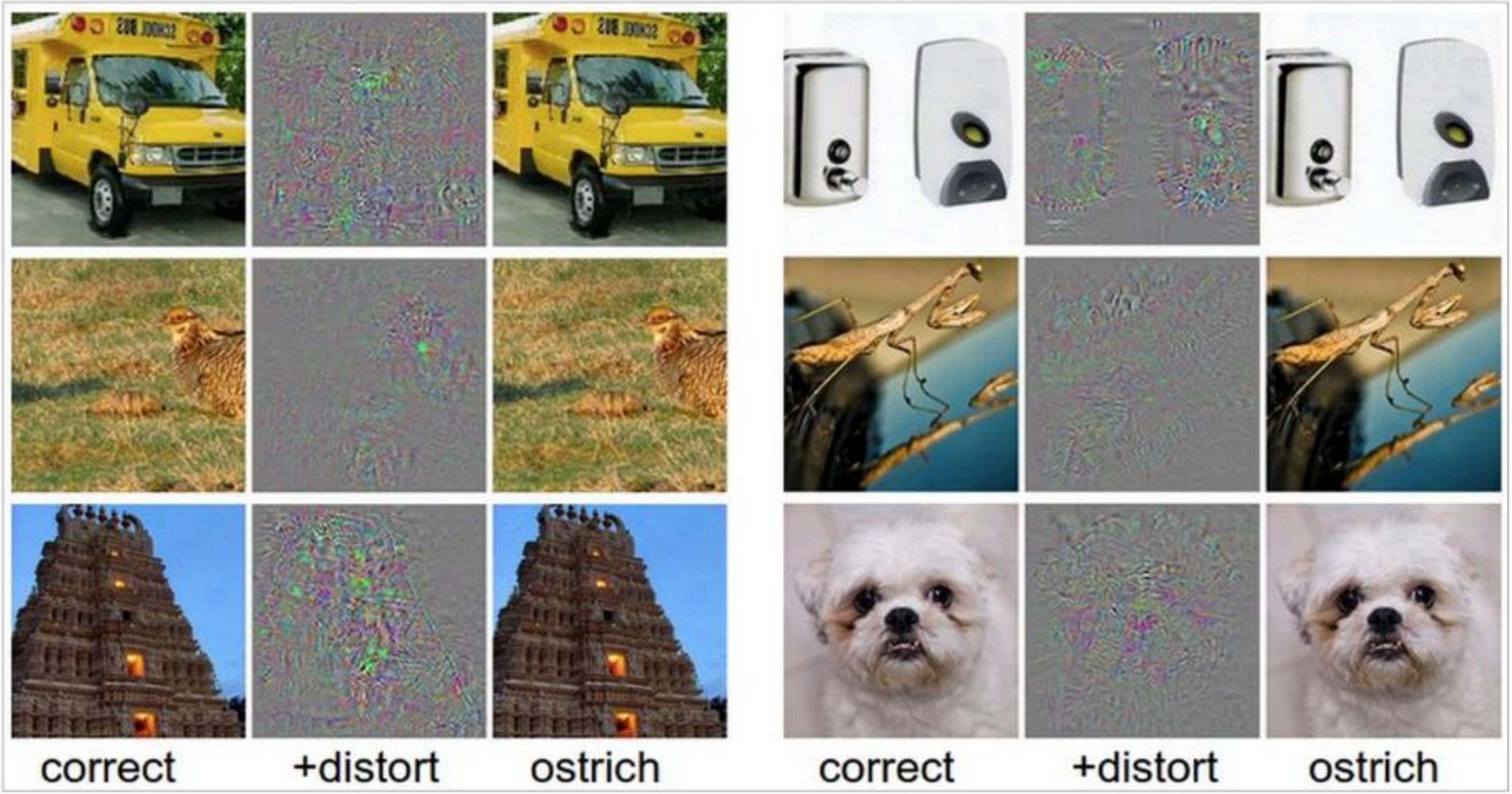
(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat' (d)Guided Grad-CAM 'Cat' (e) Occlusion map 'Cat' (f) ResNet Grad-CAM 'Cat'

(g) Original Image (h) Guided Backprop 'Dog' (i) Grad-CAM 'Dog' (j)Guided Grad-CAM 'Dog' (k) Occlusion map 'Dog' (l)ResNet Grad-CAM 'Dog'

Fig. 1: (a) Original image with a cat and a dog. (b-f) Support for the cat category according to various visualizations for VGG-16 and ResNet. (b) Guided Backpropagation [53]: highlights all contributing features. (c, f) Grad-CAM (Ours): localizes class-discriminative regions, (d) Combining (b) and (c) gives Guided Grad-CAM, which gives high-resolution class-discriminative visualizations. Interestingly, the localizations achieved by our Grad-CAM technique, (c) are very similar to results from occlusion sensitivity (e), while being orders of magnitude cheaper to compute. (f, l) are Grad-CAM visualizations for ResNet-18 layer. Note that in (c, f, i, l), red regions corresponds to high score for class, while in (e, k), blue corresponds to evidence for the class. Figure best viewed in color.

Selvaraju et al. ICCV 2017, IJCV 2019

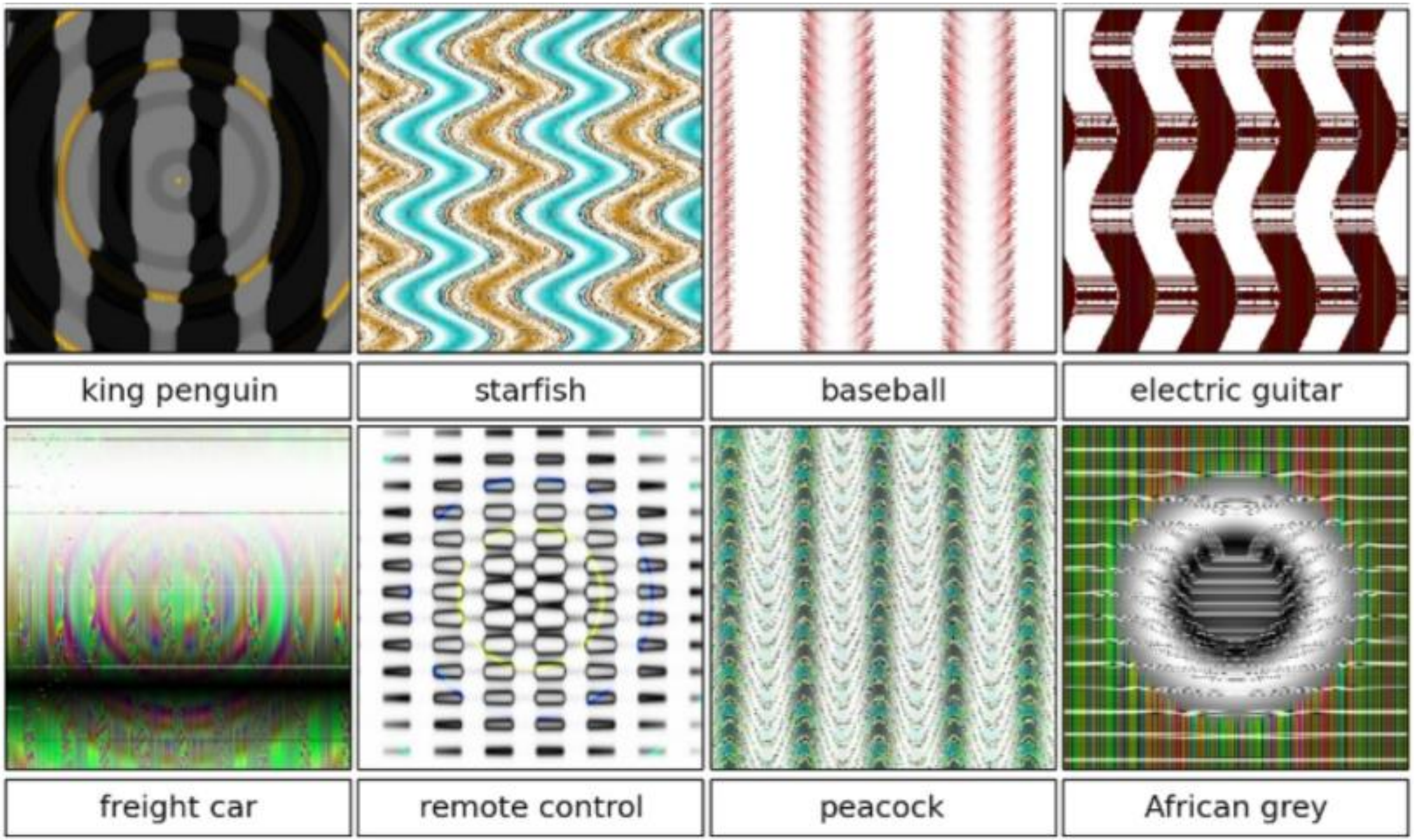
Breaking CNNs



Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

Intriguing properties of neural networks [Szegedy ICLR 2014]

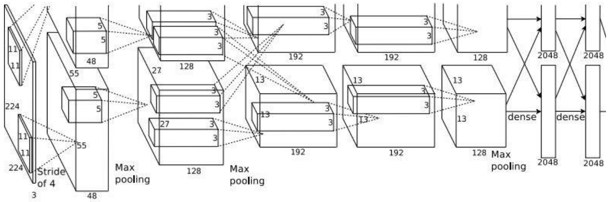
Breaking CNNs



Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images [Nguyen et al. CVPR 2015]

Applications in computer vision

Image Classification



Vector:
4096

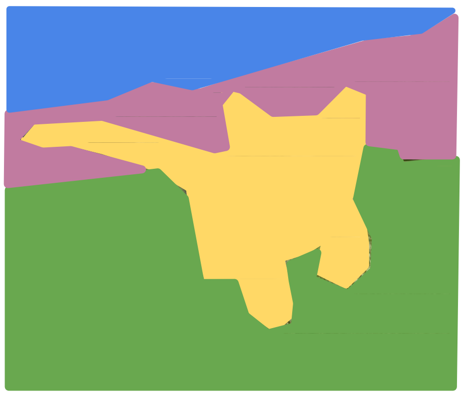
Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Slide by: Justin Johnson

Other Computer Vision Tasks

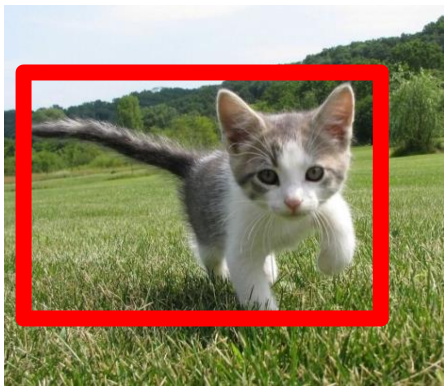
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

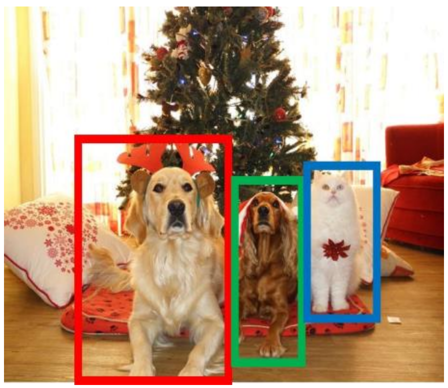
Classification + Localization



CAT

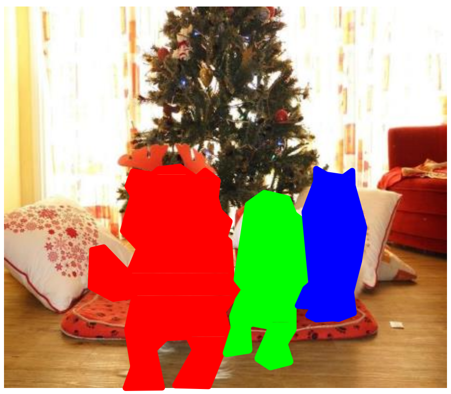
Single Object

Object Detection



DOG, DOG, CAT

Instance Segmentation

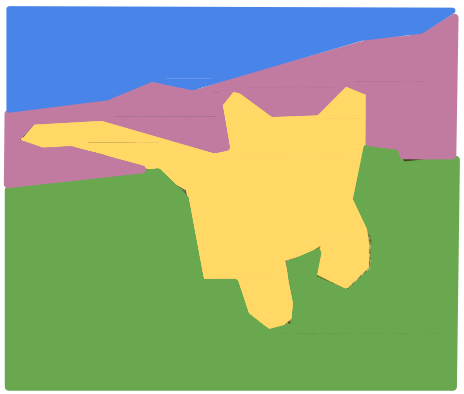


DOG, DOG, CAT

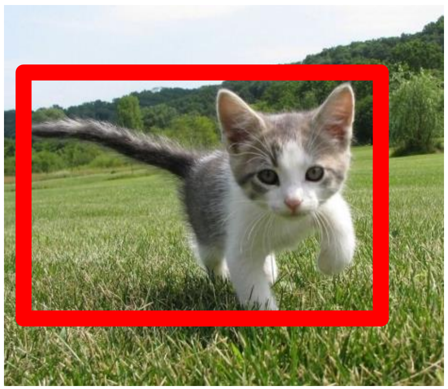
Multiple Object

Slide by: Justin Johnson

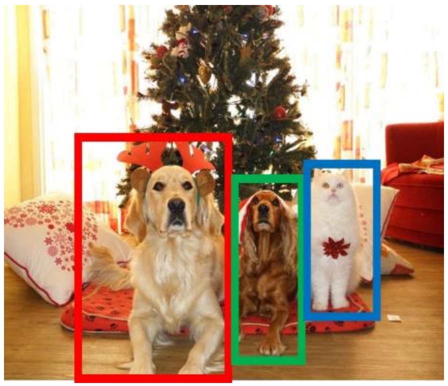
Classification + Localization



GRASS, CAT, TREE, SKY



CAT



DOG, DOG, CAT

DOG, DOG, CAT

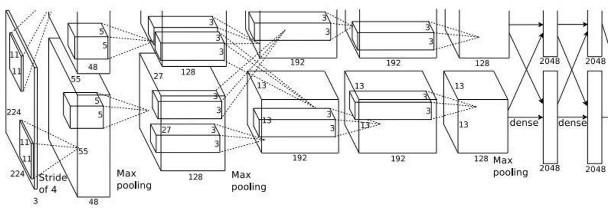
No objects, just pixels

Single Object

Multiple Object

Slide by: Justin Johnson

Classification + Localization



Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

Treat localization as a regression problem!

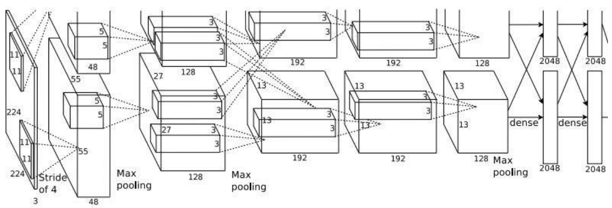
Classification + Localization

Correct label:
Cat

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Fully Connected:
4096 to 1000



Vector: 4096

Fully Connected:
4096 to 4

Box Coordinates
(x, y, w, h)

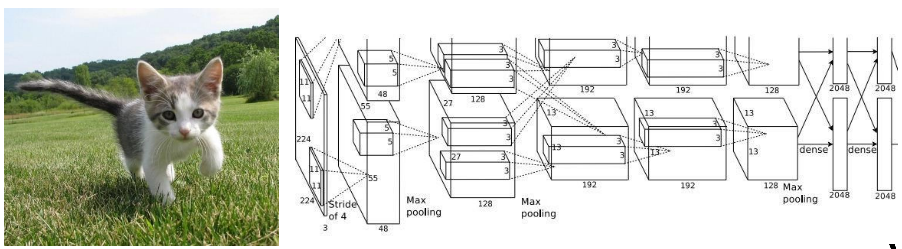
L2 Loss

Correct box:
(x’, y’, w’, h’)

Treat localization as a regression problem!

Slide by: Justin Johnson

Classification + Localization



Fully Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Correct label:
Cat

Softmax Loss

Vector: 4096

Multitask Loss

+ → **Loss**

Fully Connected:
4096 to 4

Box Coordinates
(x, y, w, h)

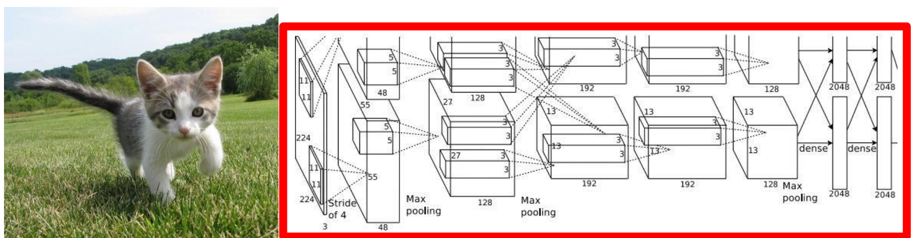
L2 Loss

Correct box:
(x', y', w', h')

Treat localization as a regression problem!

Slide by: Justin Johnson

Classification + Localization



Often pretrained on ImageNet (Transfer learning)

Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Correct label: Cat

Softmax Loss

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

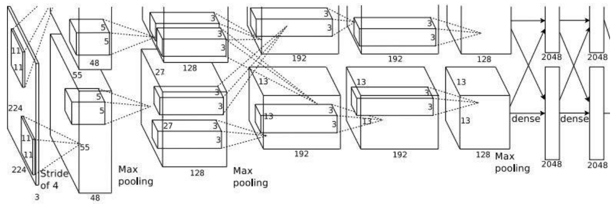
Correct box: (x', y', w', h')

+ → **Loss**

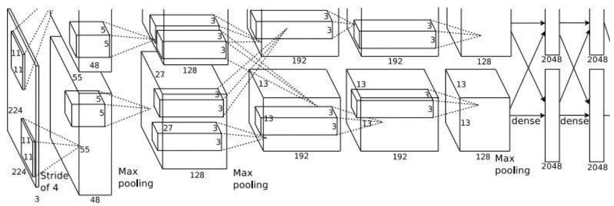
Treat localization as a regression problem!

Slide by: Justin Johnson

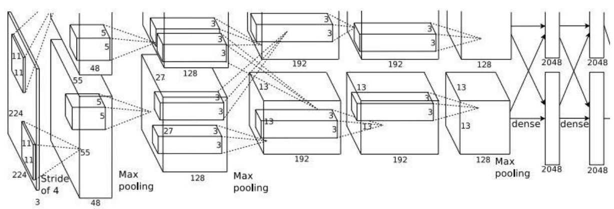
Object Detection as Regression?



CAT: (x, y, w, h)



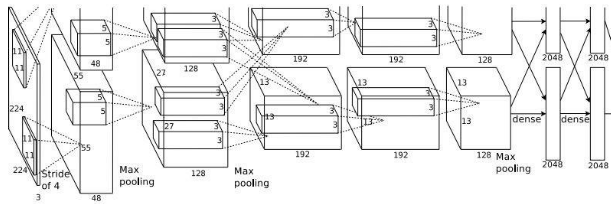
DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)



DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

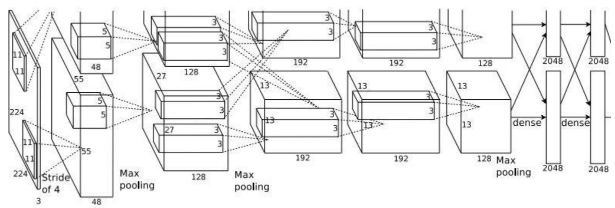
Slide by: Justin Johnson

Object Detection as Regression?

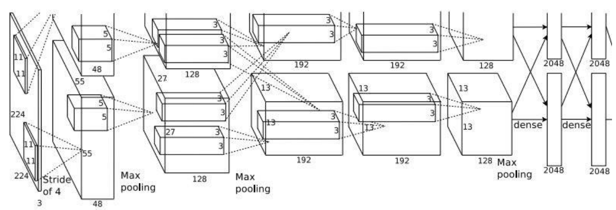
CAT: (x, y, w, h)

4 numbers

DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

16 numbers

DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

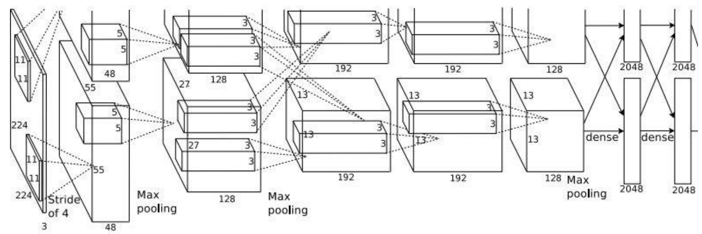
Many numbers!

Each image needs a different number of outputs!

Slide by: Justin Johnson

Object Detection as Classification: Sliding Window

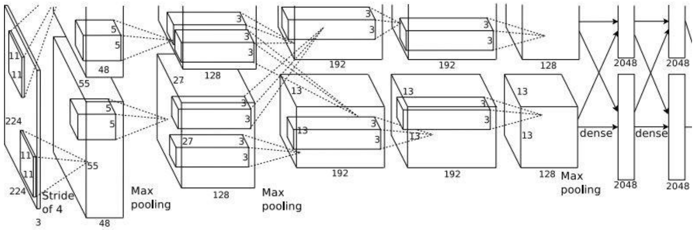
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? NO
Background? YES

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

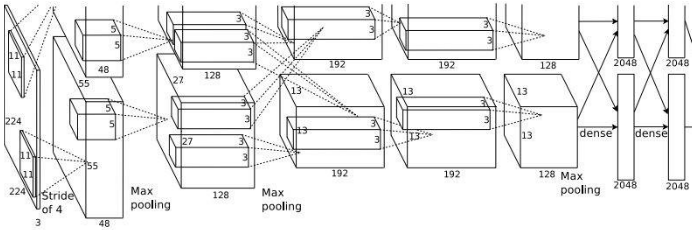


Dog? YES
Cat? NO
Background? NO

Slide by: Justin Johnson

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

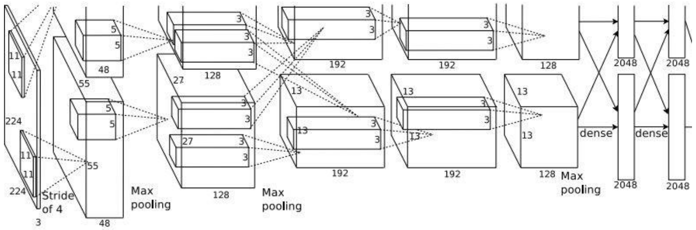


Dog? YES
Cat? NO
Background? NO

Slide by: Justin Johnson

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

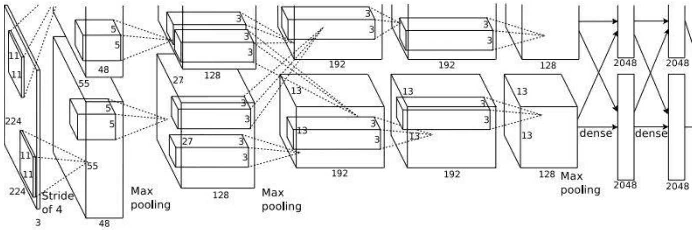


Dog? NO
Cat? YES
Background? NO

Slide by: Justin Johnson

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



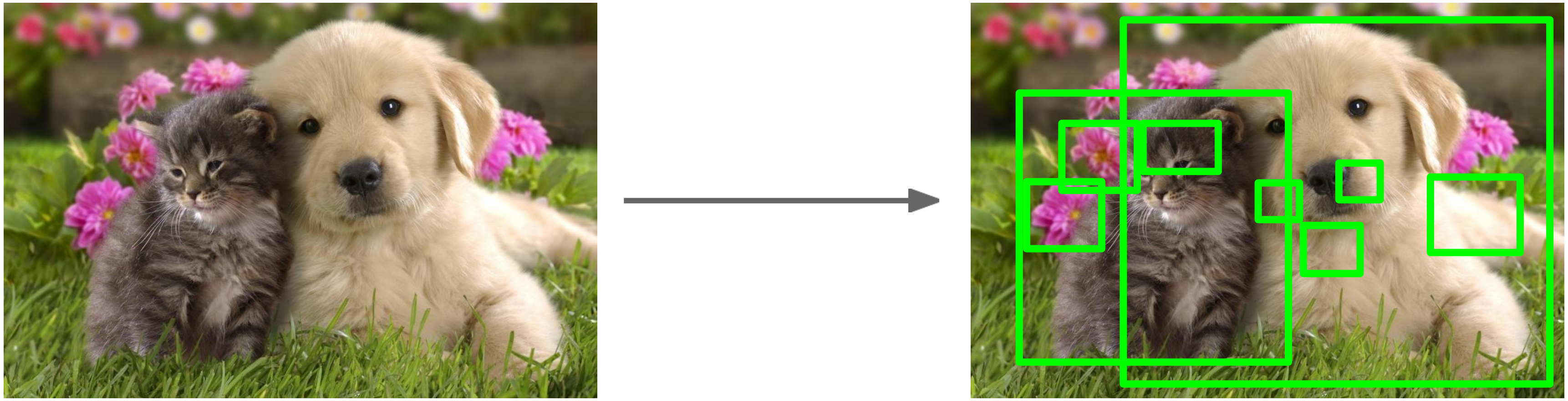
Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations and scales, very computationally expensive!

Slide by: Justin Johnson

Region Proposals

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 1000 region proposals in a few seconds on CPU



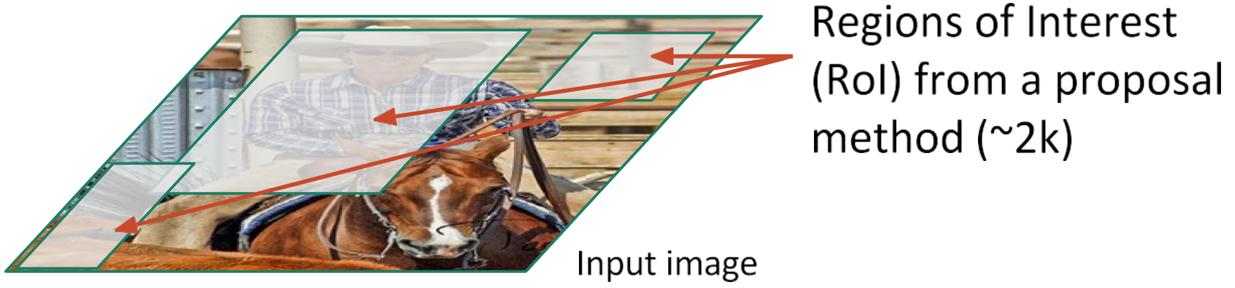
Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

R-CNN

Input image

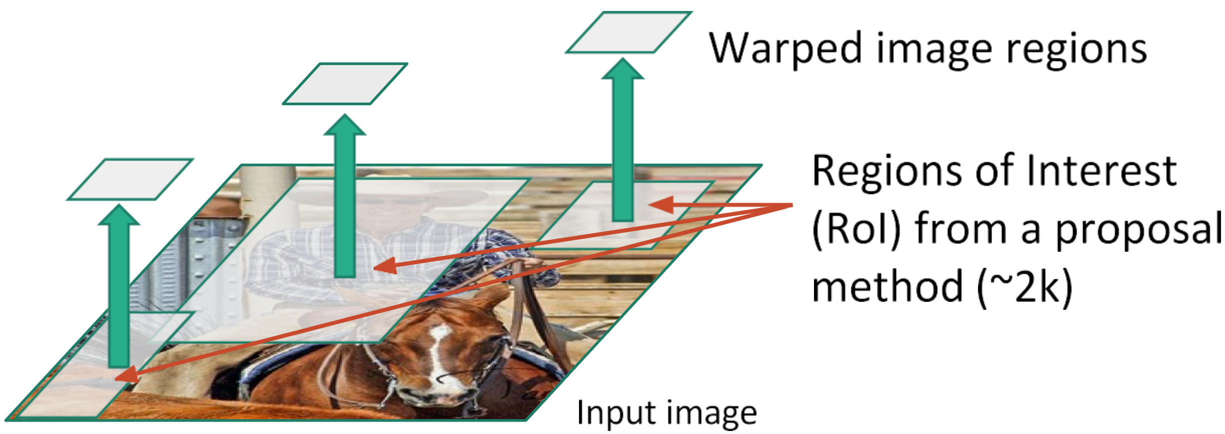
Girshick et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", CVPR 2014

R-CNN



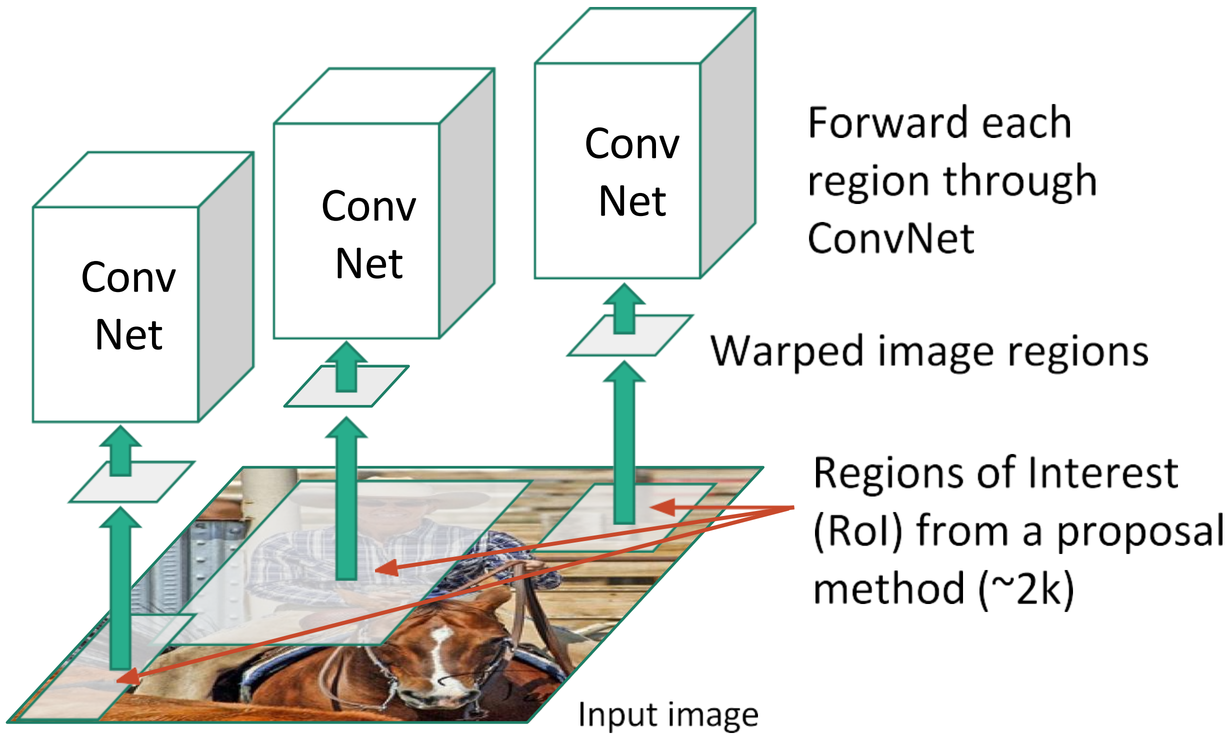
Girshick et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", CVPR 2014

R-CNN



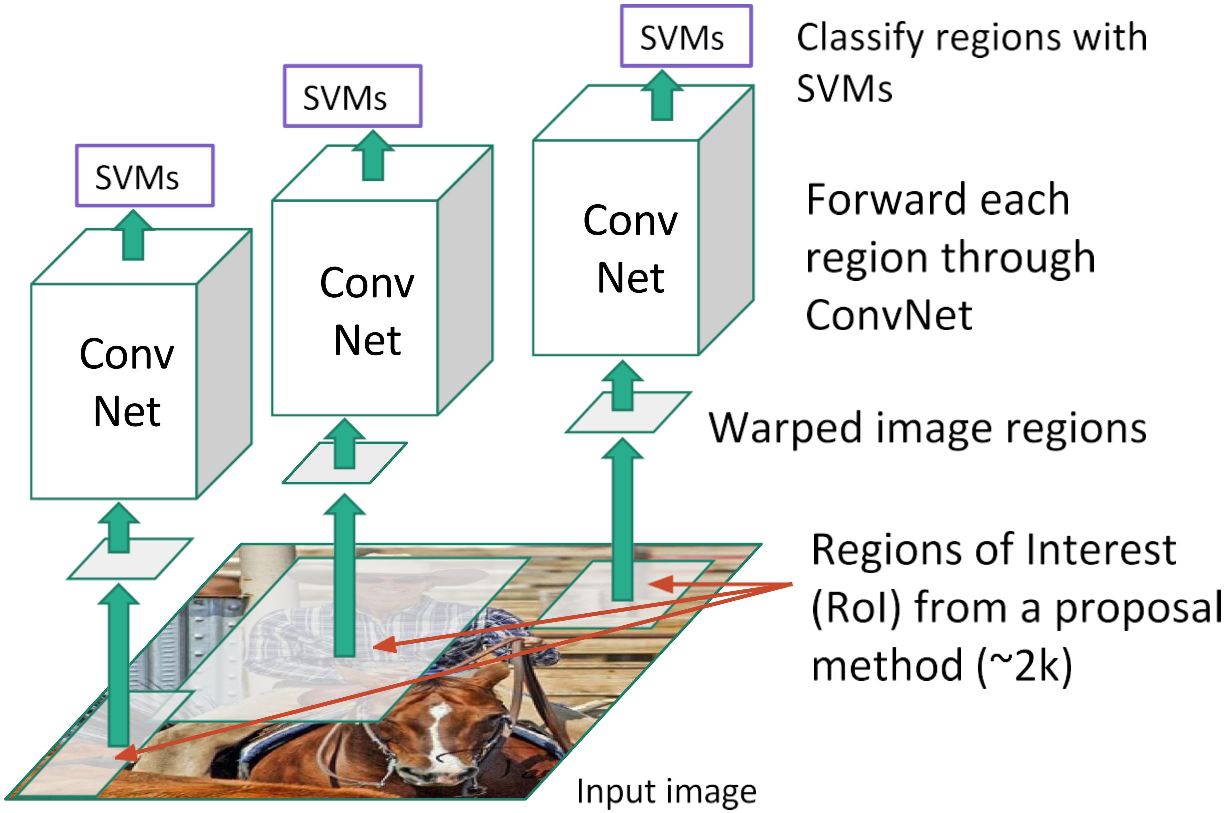
Girshick et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", CVPR 2014

R-CNN



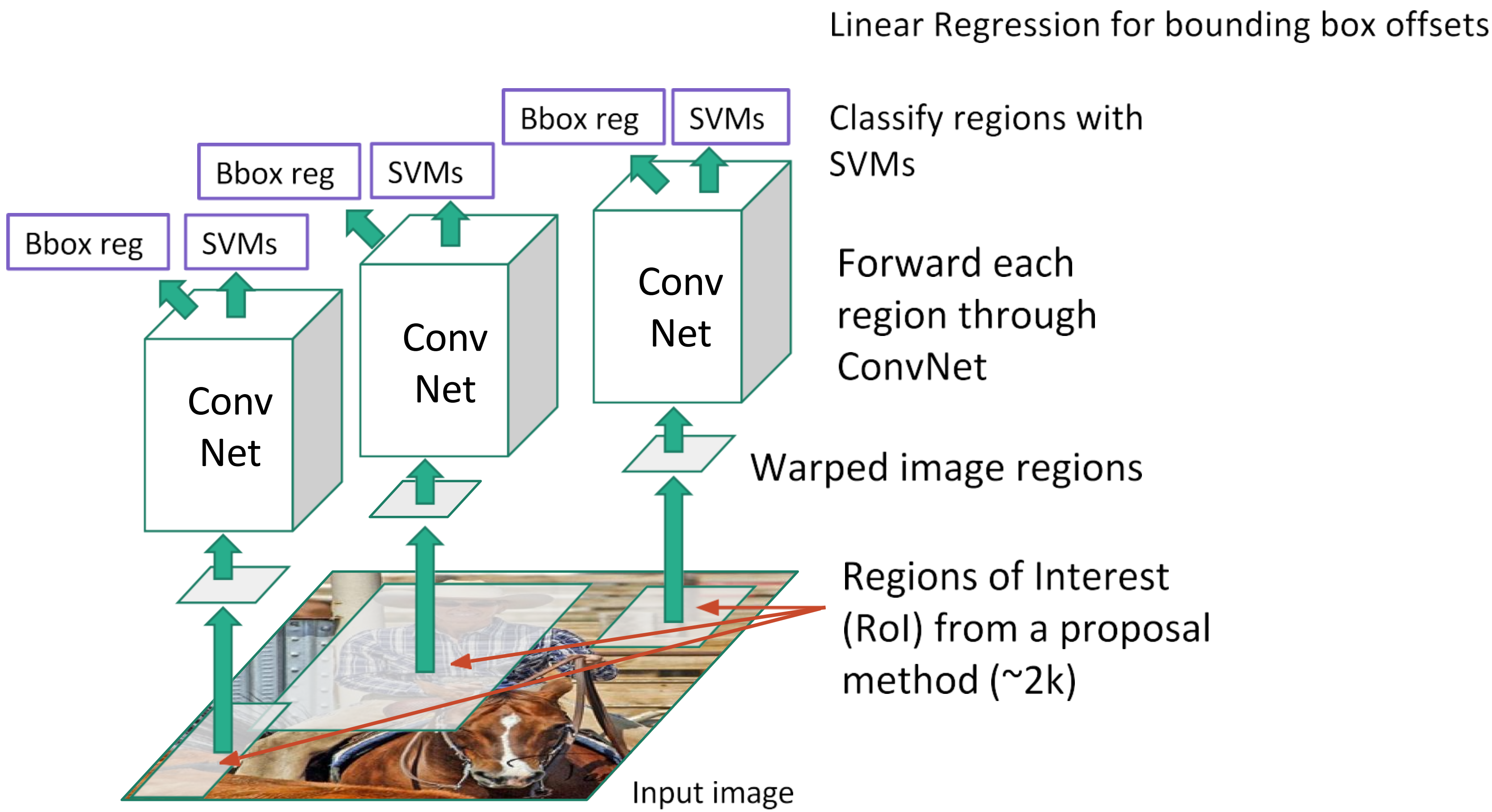
Girshick et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", CVPR 2014

R-CNN



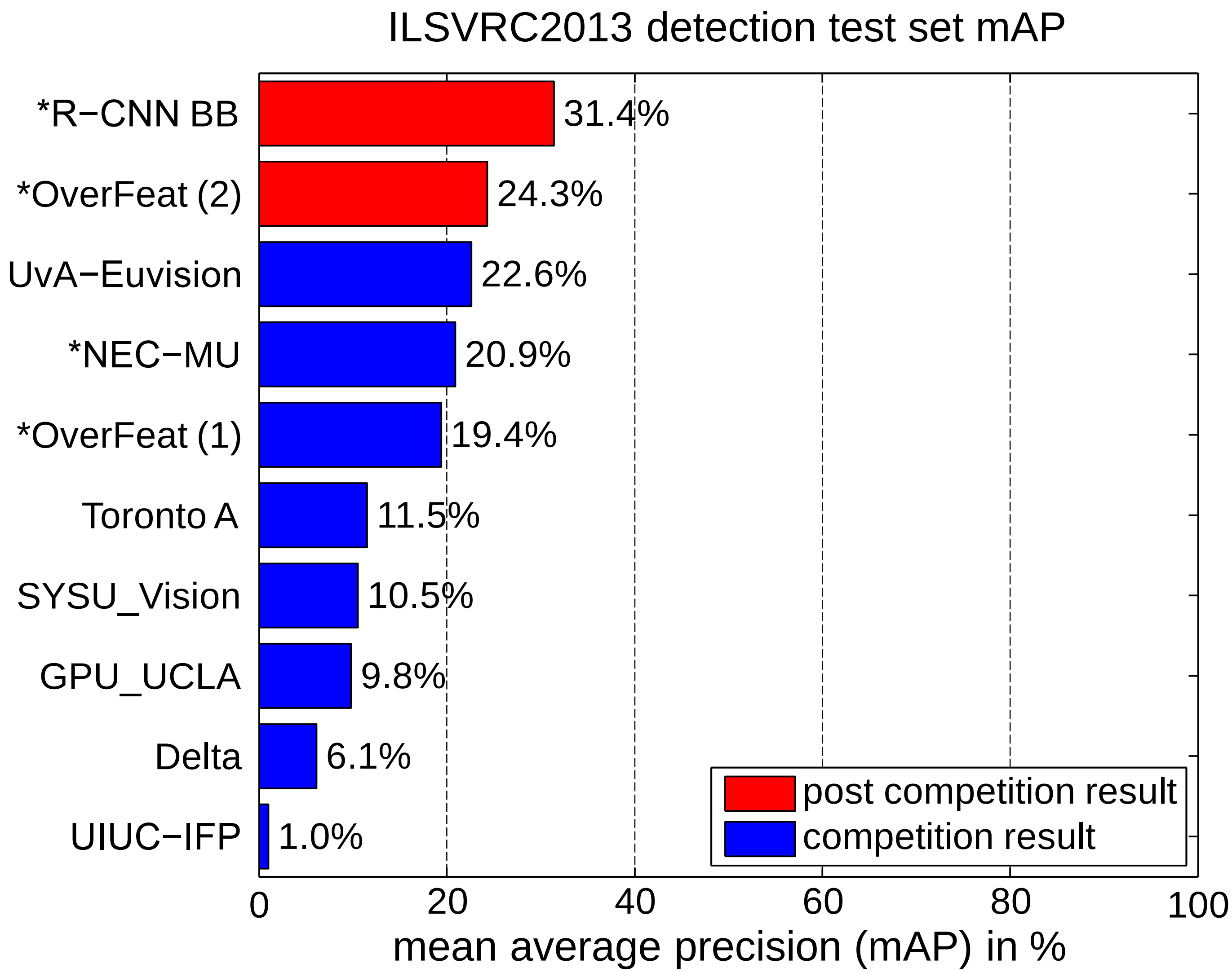
Girshick et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", CVPR 2014

R-CNN



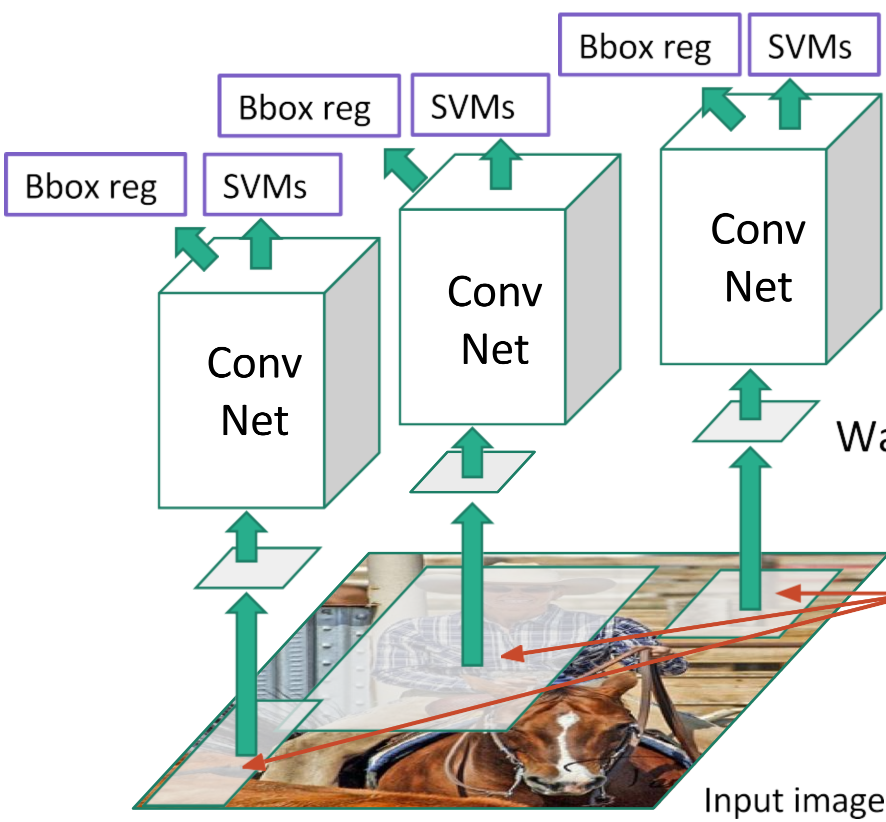
Girshick et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", CVPR 2014

R-CNN on ImageNet detection



Girshick et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", CVPR 2014

R-CNN

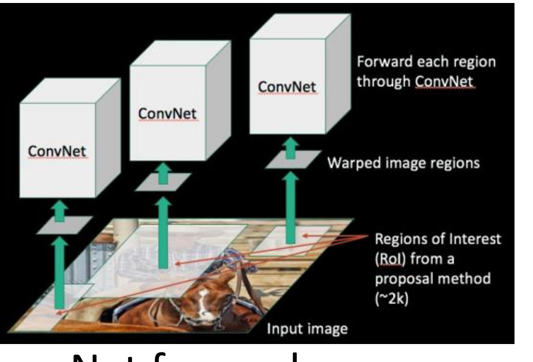


Post hoc component

Girshick et al., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", CVPR 2014

What's wrong with slow R-CNN?

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- Training is slow (84h), takes a lot of disk space
- Inference (detection) is slow
  - 47s / image with VGG16 [Simonyan & Zisserman, ICLR15]



~2000 ConvNet forward passes per image

Girshick, "Fast R-CNN", ICCV 2015

Fast R-CNN

- Fast test time
- One network, trained in one stage
- Higher mean average precision

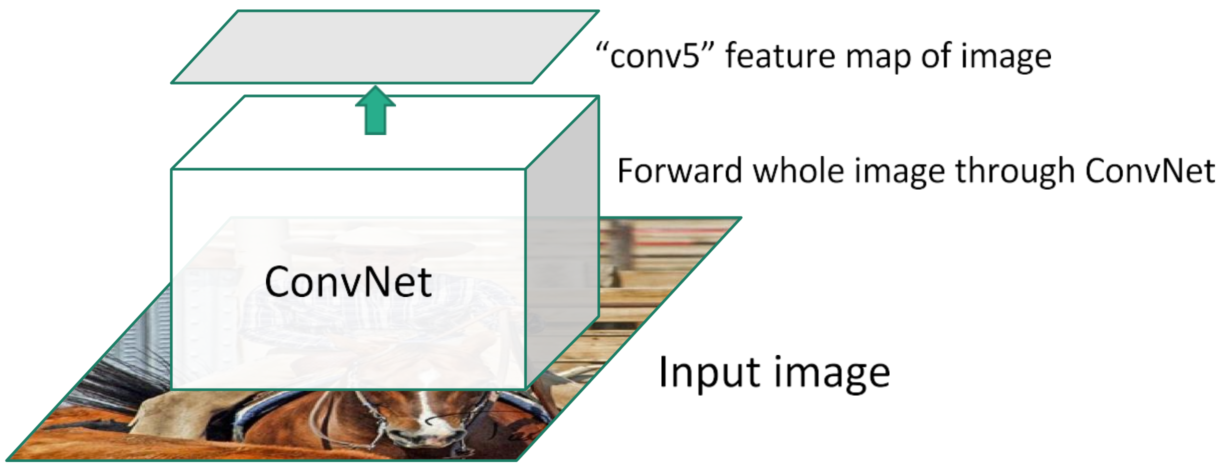
Girshick, "Fast R-CNN", ICCV 2015

Fast R-CNN

Input image

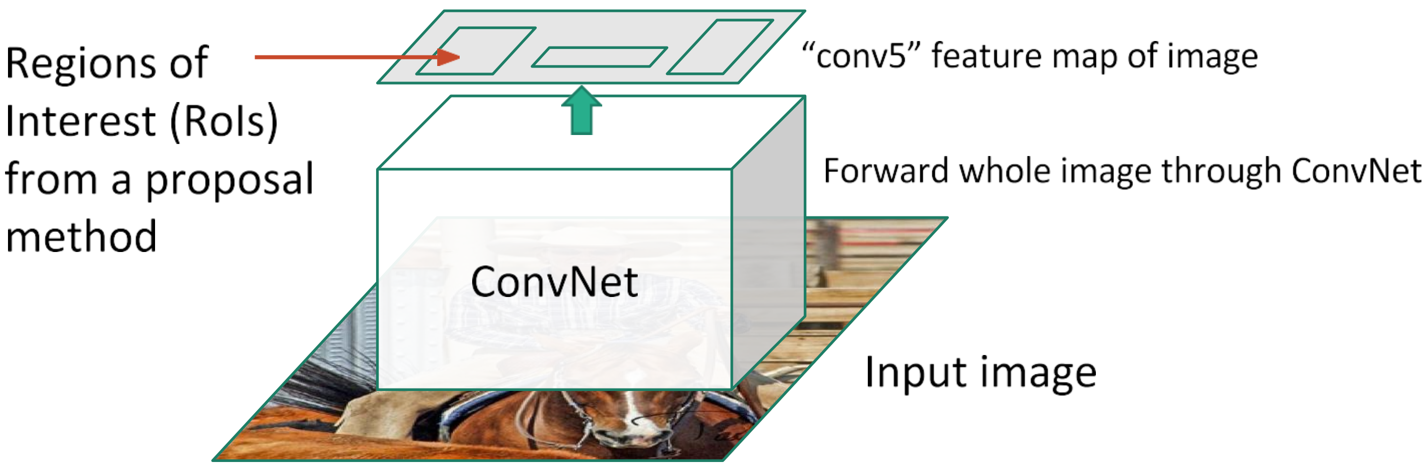
Girshick, "Fast R-CNN", ICCV 2015

Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015

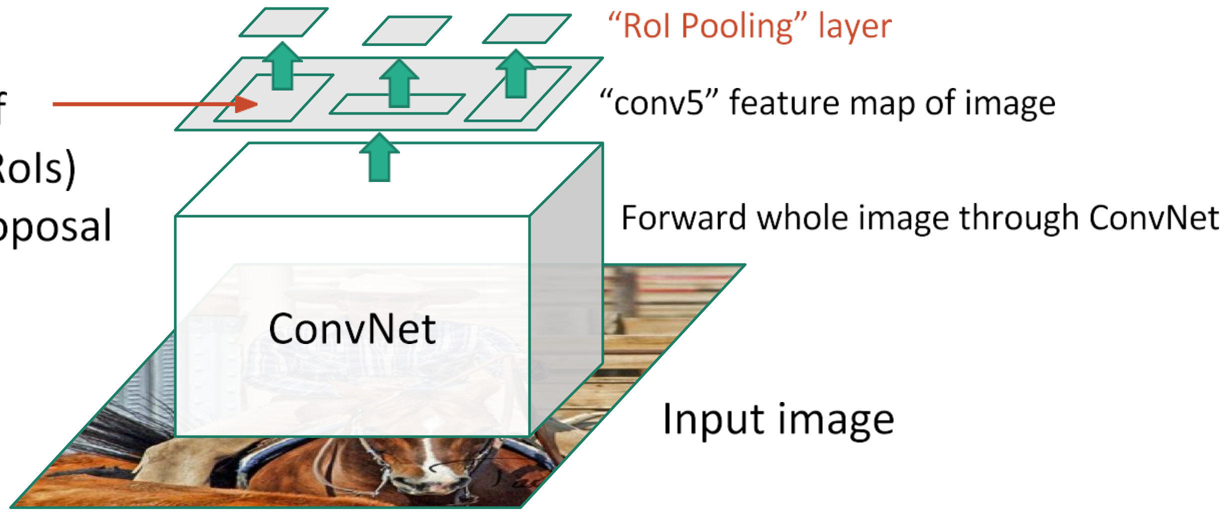
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015

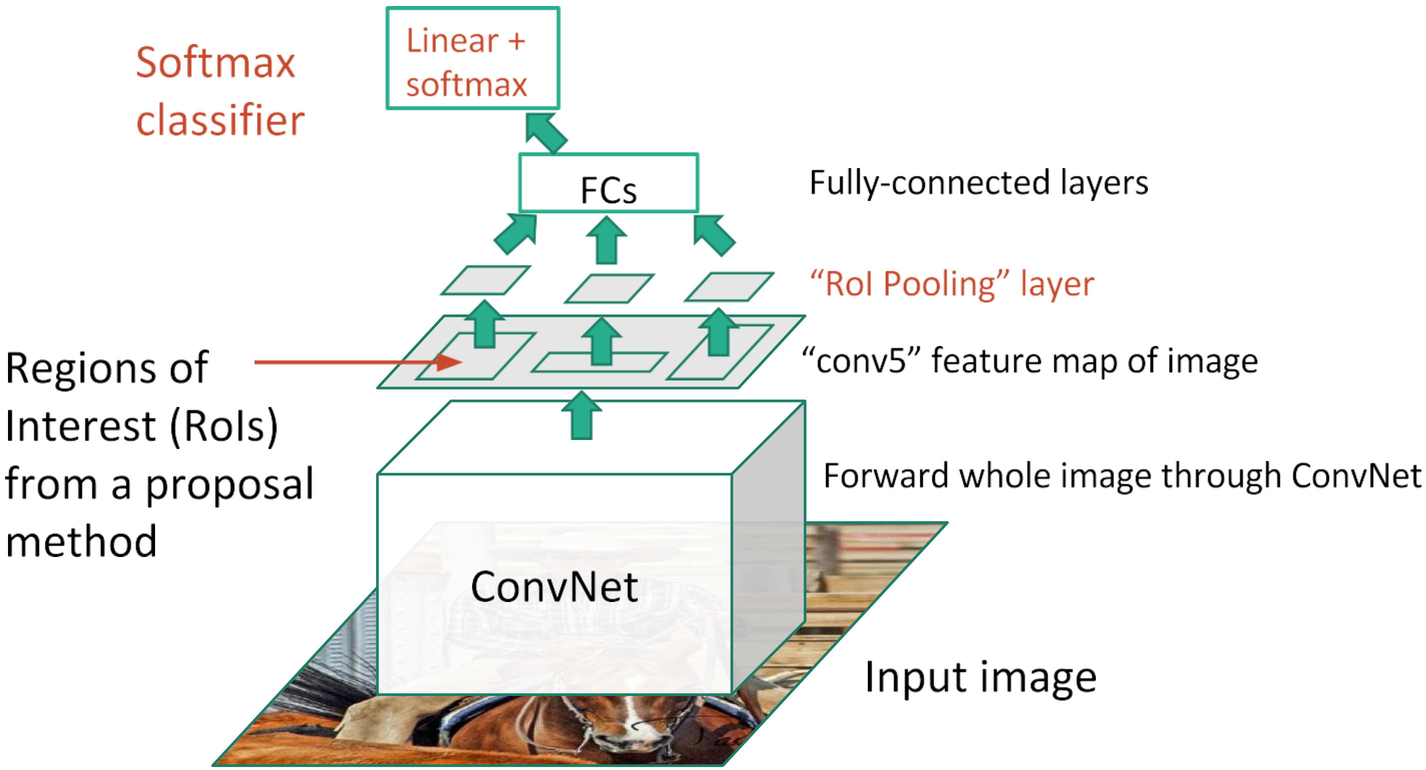
Fast R-CNN

Regions of Interest (RoIs) from a proposal method



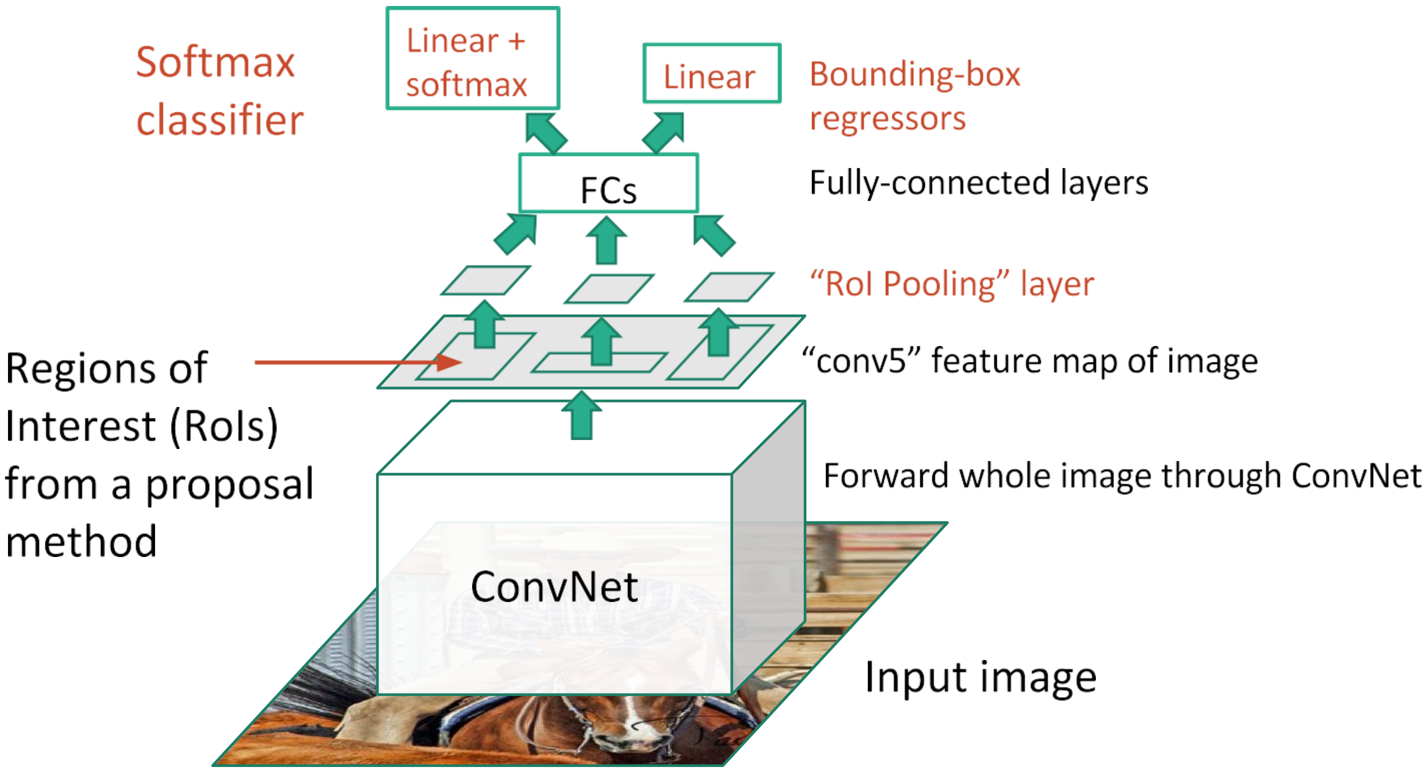
Girshick, “Fast R-CNN”, ICCV 2015

Fast R-CNN



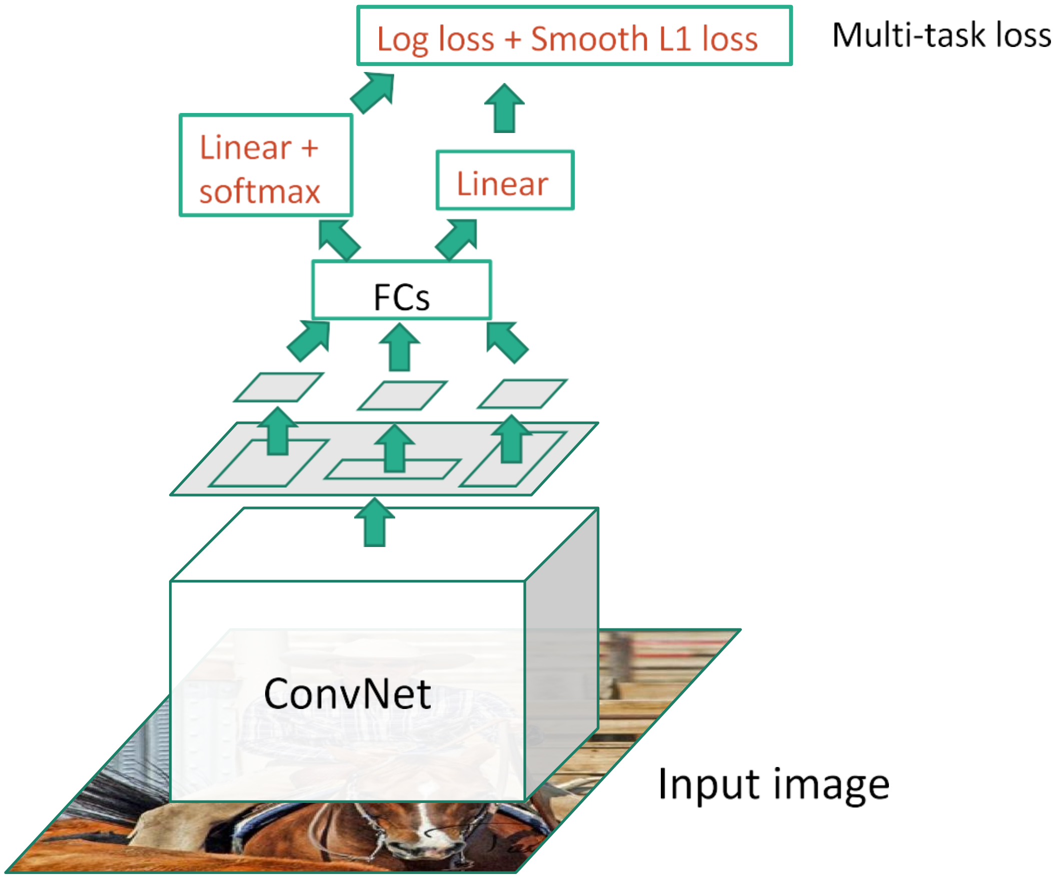
Girshick, "Fast R-CNN", ICCV 2015

Fast R-CNN



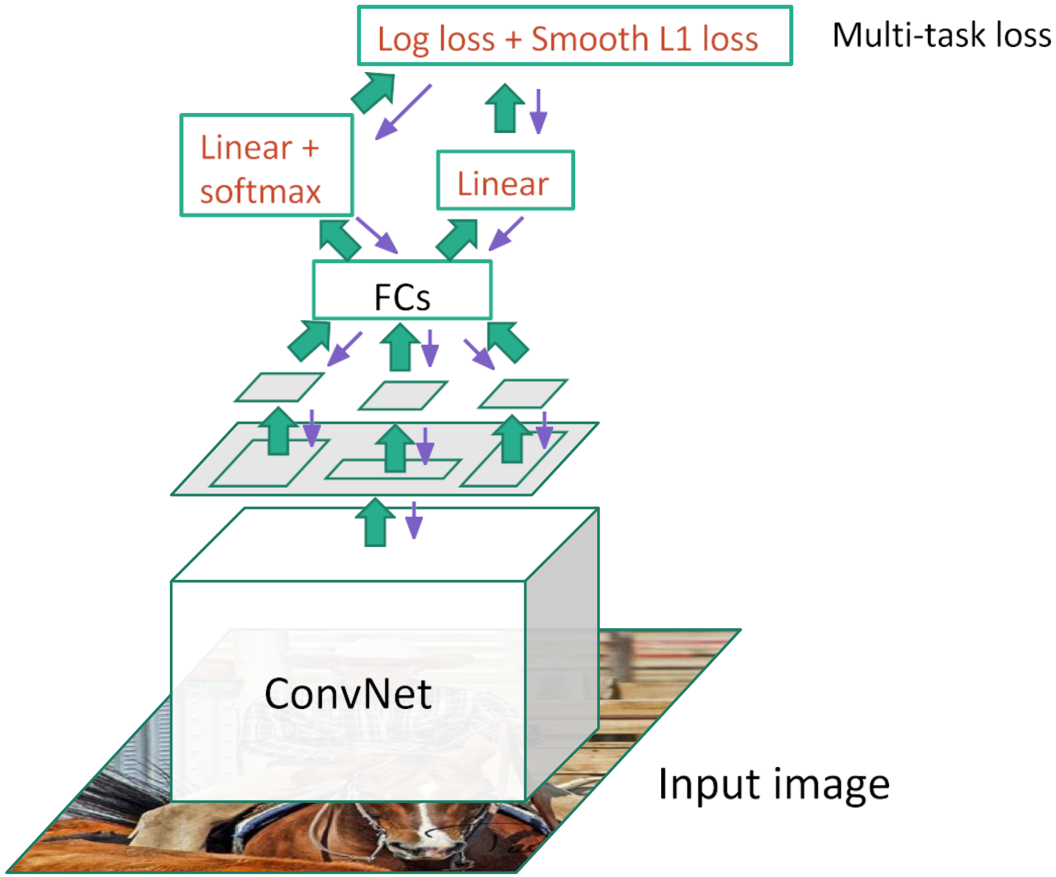
Girshick, "Fast R-CNN", ICCV 2015

Fast R-CNN (Training)



Girshick, "Fast R-CNN", ICCV 2015

Fast R-CNN (Training)



Girshick, "Fast R-CNN", ICCV 2015

Fast R-CNN vs R-CNN

	Fast R-CNN	R-CNN
Train time (h)	9.5	84
Speedup	8.8x	1x
Test time / image	0.32s	47.0s
Test speedup	146x	1x
mAP	66.9%	66.0%

Timings exclude object proposal time, which is equal for all methods.
All methods use VGG16 from Simonyan and Zisserman.

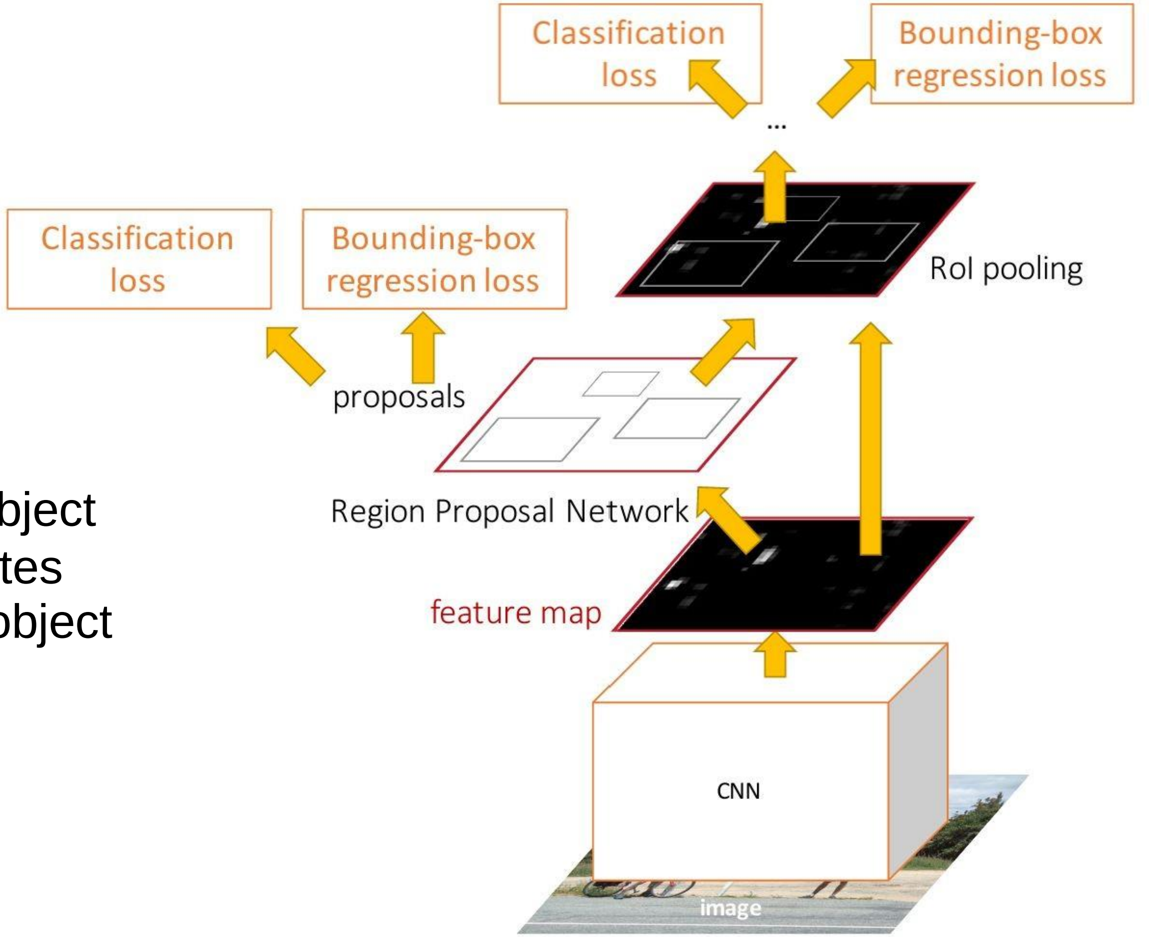
Girshick, "Fast R-CNN", ICCV 2015

Faster R-CNN

Make CNN do proposals!

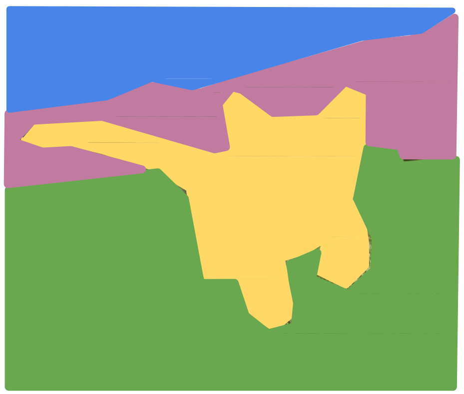
Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



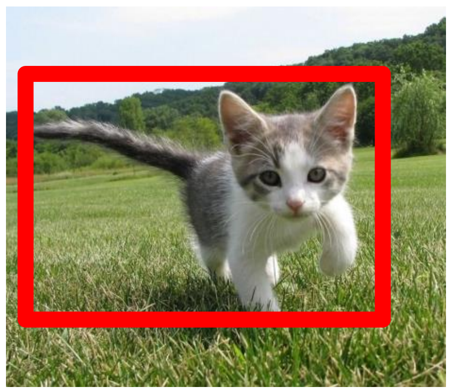
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Semantic Segmentation



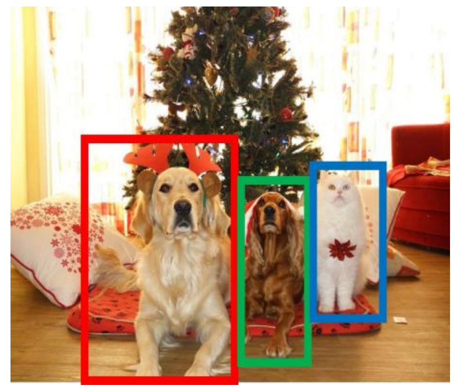
GRASS, CAT, TREE, SKY

No objects, just pixels



CAT

Single Object



DOG, DOG, CAT

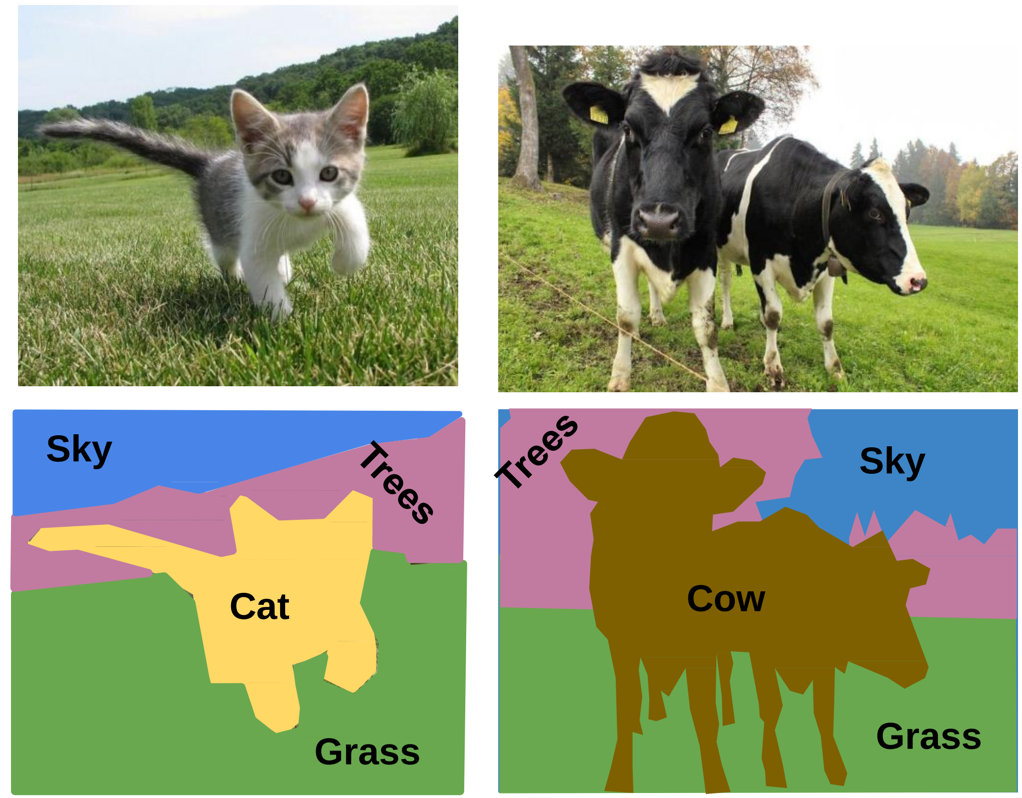
DOG, DOG, CAT

Multiple Object

Semantic Segmentation

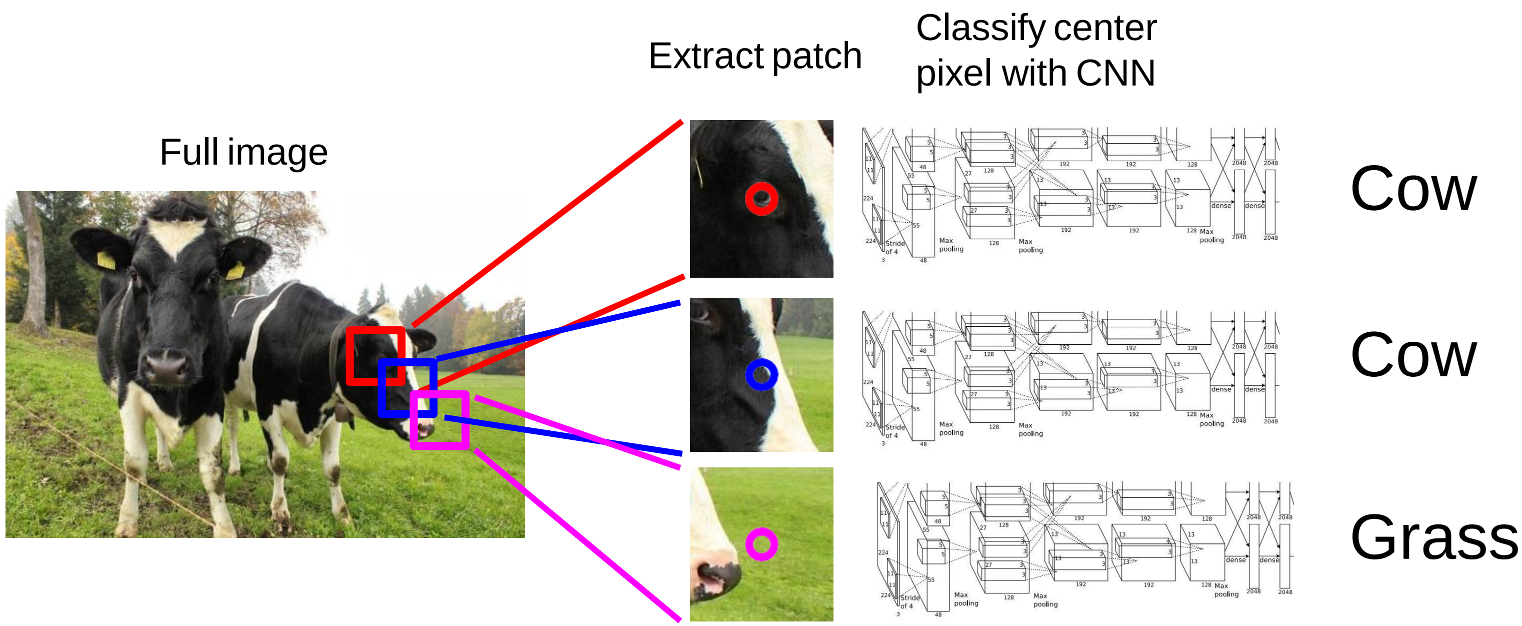
Label each pixel in the image with a category label

Don’t differentiate instances, only care about pixels



Slide by: Justin Johnson

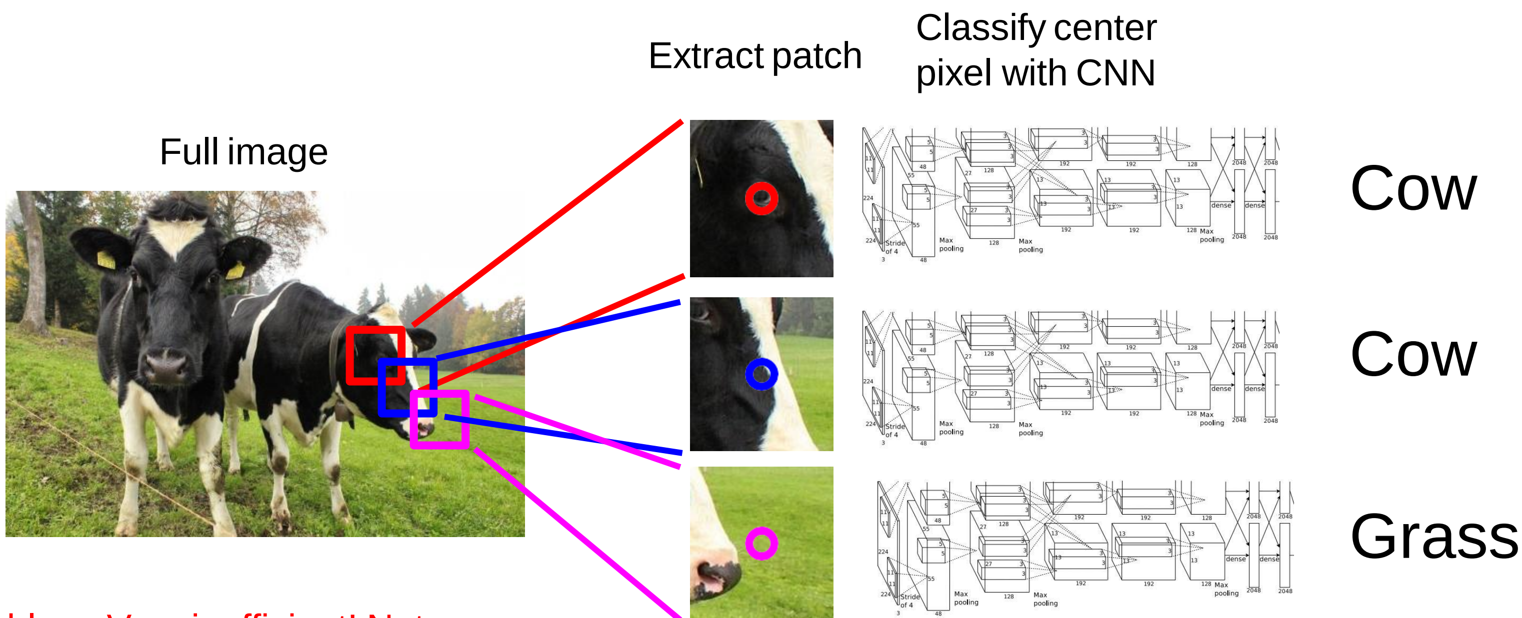
Semantic Segmentation Idea: Sliding Window



Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

Slide by: Justin Johnson

Semantic Segmentation Idea: Sliding Window

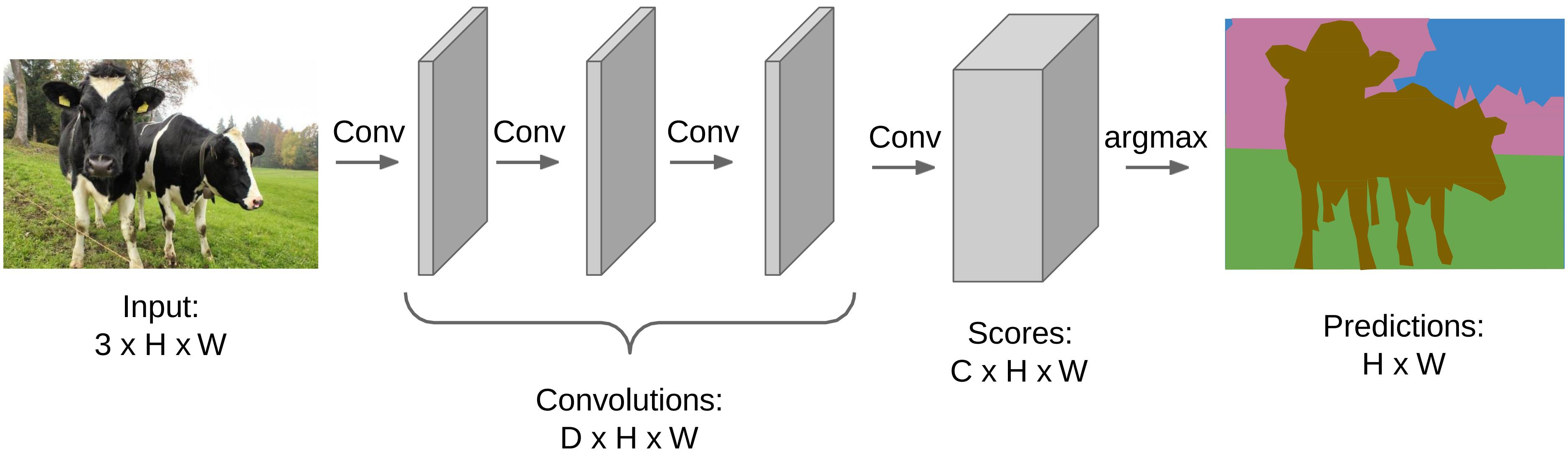


Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

Semantic Segmentation Idea: Fully Convolutional

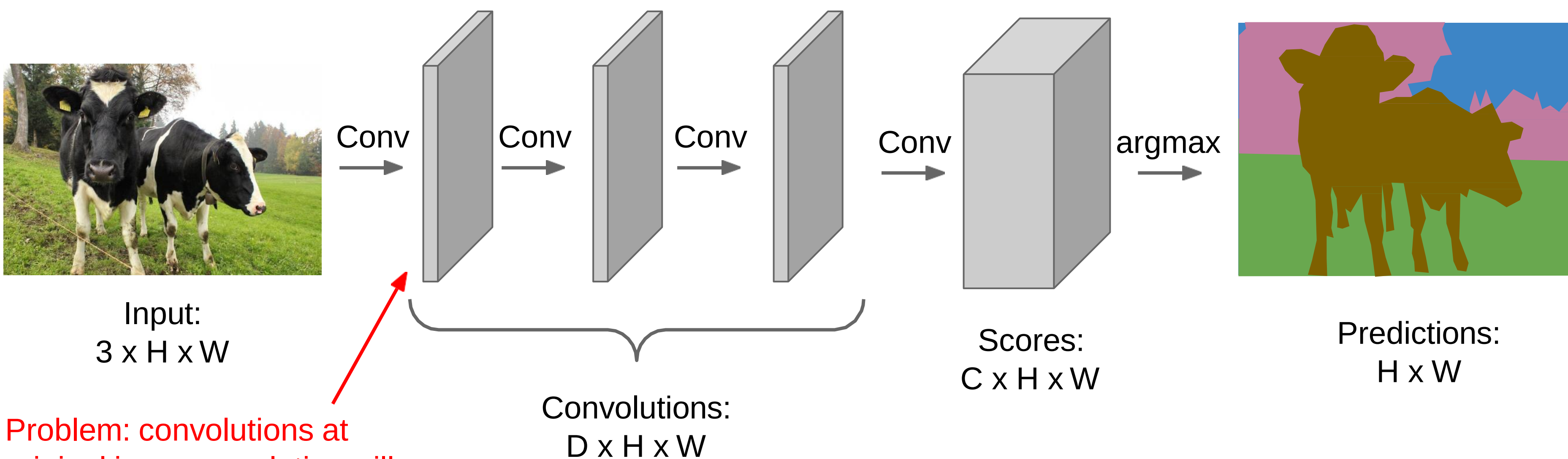
Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!



Slide by: Justin Johnson

Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!



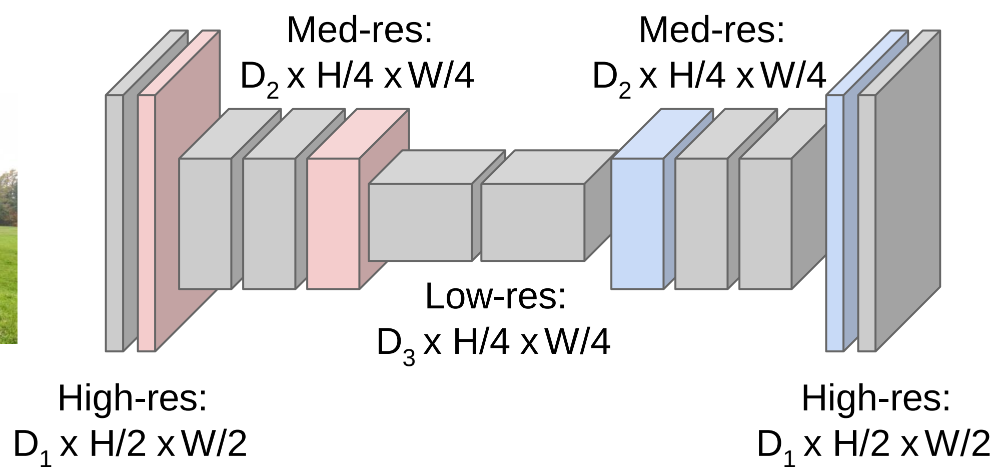
Problem: convolutions at
original image resolution will
be very expensive ...

Slide by: Justin Johnson

Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Slide by: Justin Johnson