Lectures 7 and 8: Convolutional Neural Networks and Spatial Localization and Detection

Thursday February 16, 2017

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Announcements!

- HW #2 due **next Friday Feb 24**
- Read **AlexNet paper** for next class
- Post paper summaries and discussion questions to class blog by **Mon Feb 20 11:59pm**
- These are easy points. Don't miss them.
- Final project teams will be posted to webpage this weekend.

Python/Numpy of the Day

- enumerate(<iterable object>)

- returns iterator not generator, but use case behavior is similar

- no 'yeild'

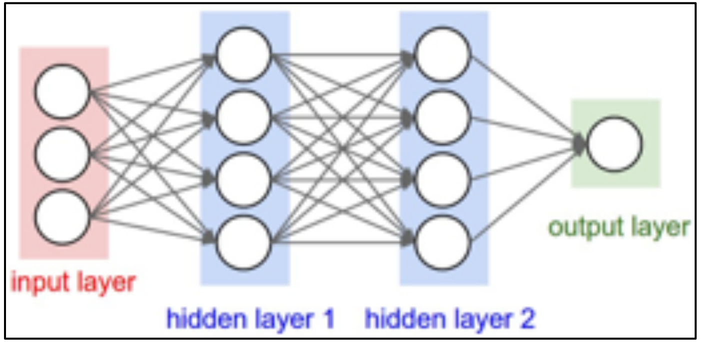
```
for ind, thing in enumerate(list_of_things):
    print 'index: {} item: {}'.format(ind, thing)

output:
index: 0 item: thing0
index: 1 item: thing1
…
```

- np.full(shape, fill_val) and np.full_like(ex_array, fill_val)

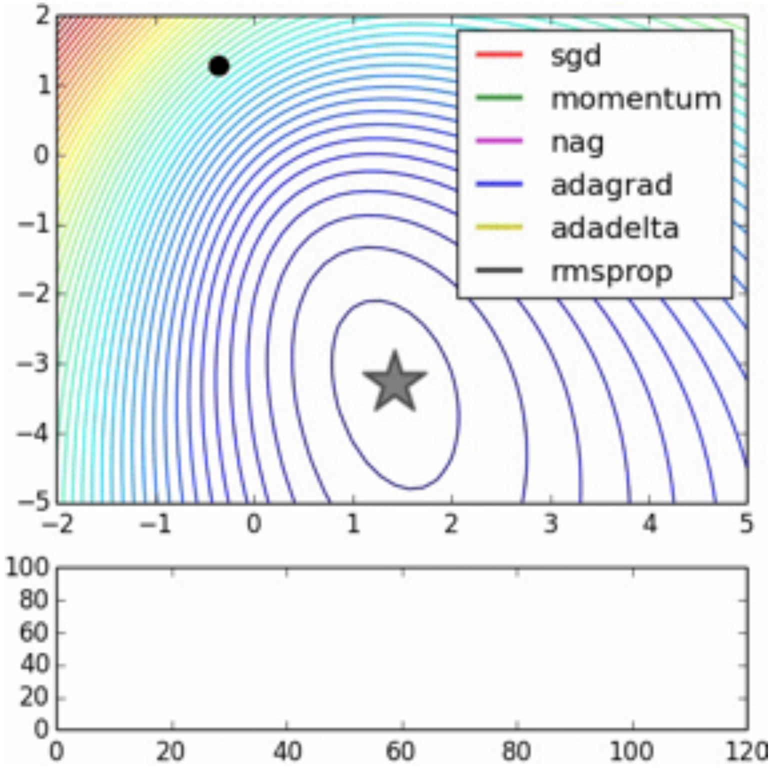
Mini-batch SGD

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient



Parameter updates

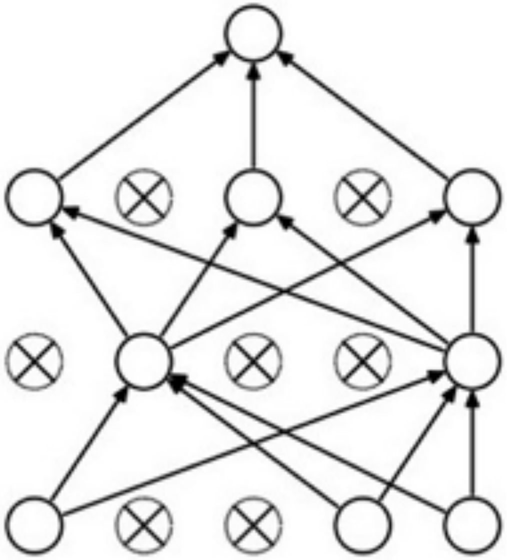


We covered:
sgd,
momentum,
nag,
adagrad,
rmsprop,
adam (not in this vis),

we did not cover adadelta

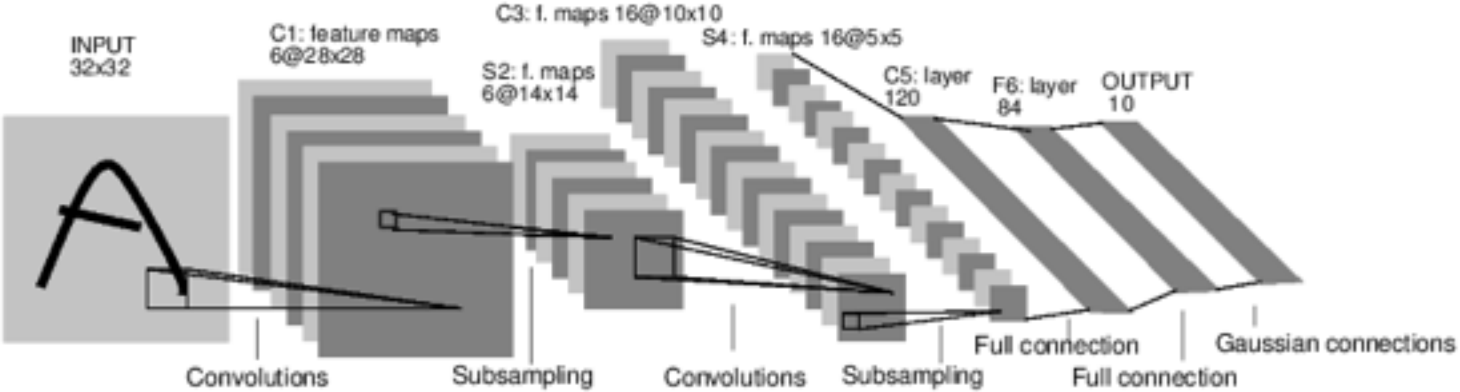
Image credits: Alec Radford

Dropout



Forces the network to have a redundant representation.

Convolutional Neural Networks



[LeNet-5, LeCun 1980]

Convolutional Neural Networks

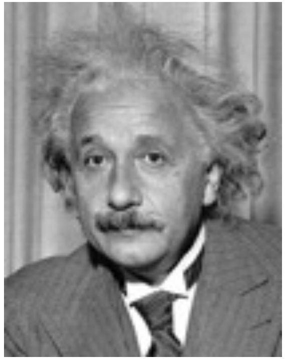
Review from linear filters

Original

Sharpening filter

- Accentuates differences with local average

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



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

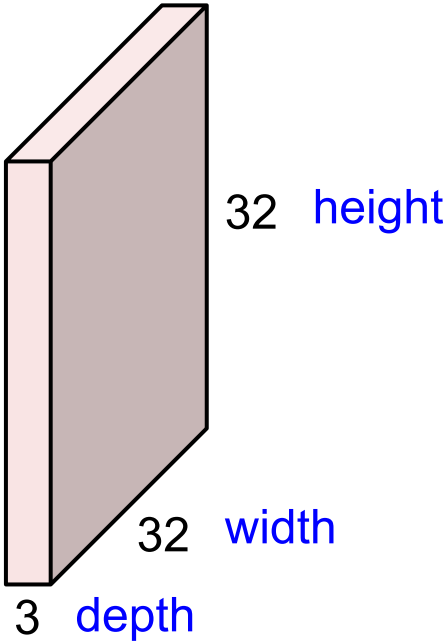
Sobel filter

- Vertical Edges

Source: D. Lowe

Convolution Layer

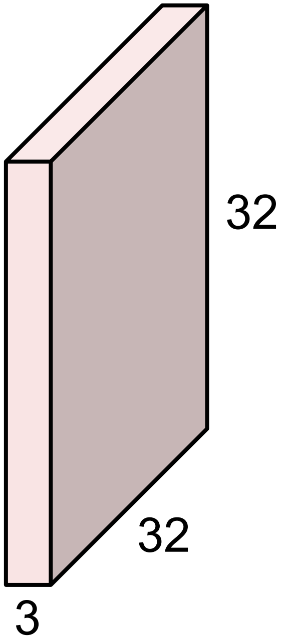
32x32x3 image



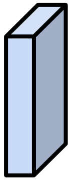
* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Convolution Layer

32x32x3 image



5x5x3 filter



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

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Convolution Layer - the convolution is in Fourier space

Let $\mathcal{F}$ denote the Fourier transform operator,

so $\mathcal{F}\{f\}$ and $\mathcal{F}\{g\}$ are the Fourier transforms of f and g, respectively.

Then

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

where $\cdot$ denotes point-wise multiplication. It also works the other way around:

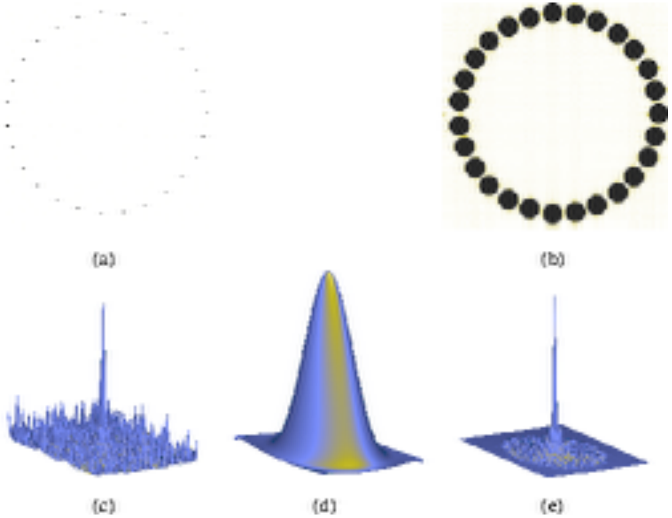
$$\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \mathcal{F}\{g\}$$

By applying the inverse Fourier transform $\mathcal{F}^{-1}$, we can write:

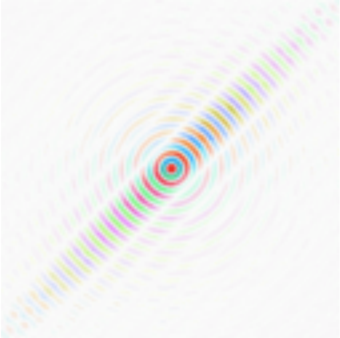
$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

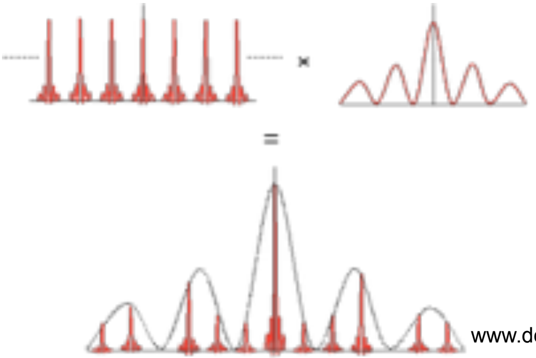
and:

$$f \cdot g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} * \mathcal{F}\{g\}\}$$



Credit: European Southern Observatory



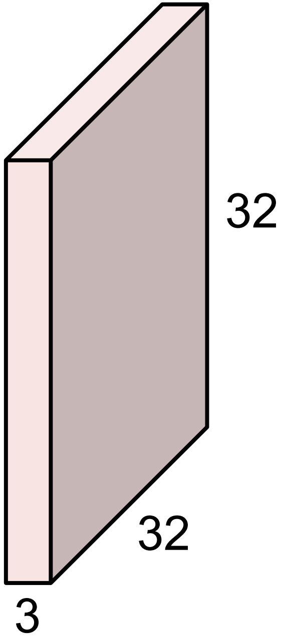


www.doitpoms.ac.uk

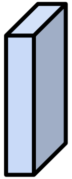
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

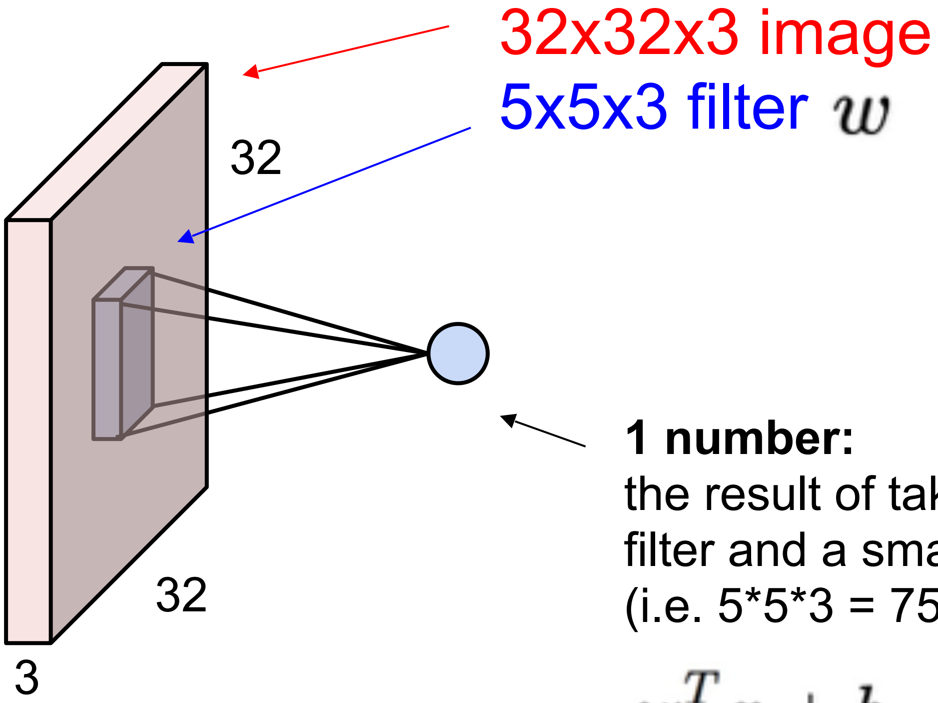


5x5x3 filter



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

Convolution Layer

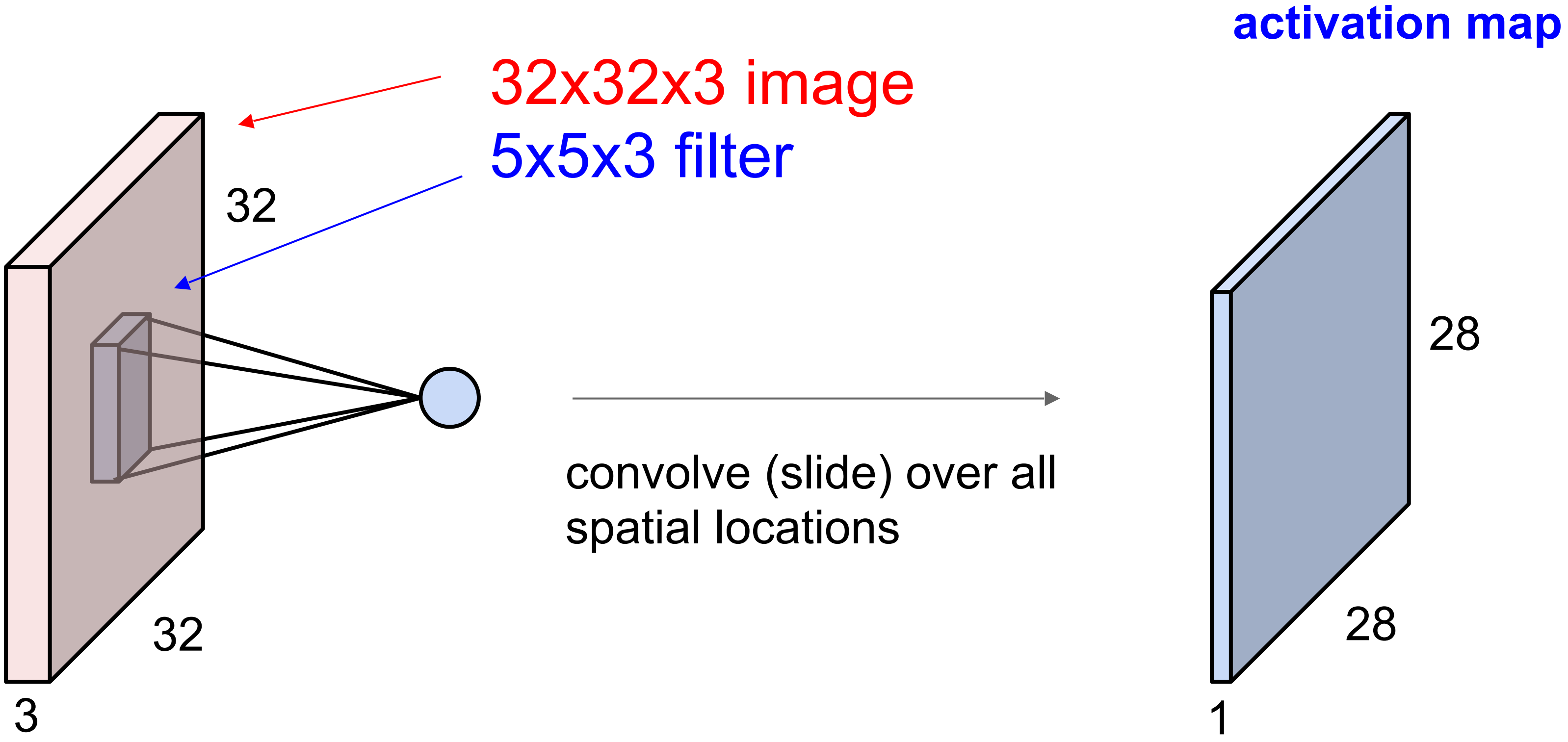


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

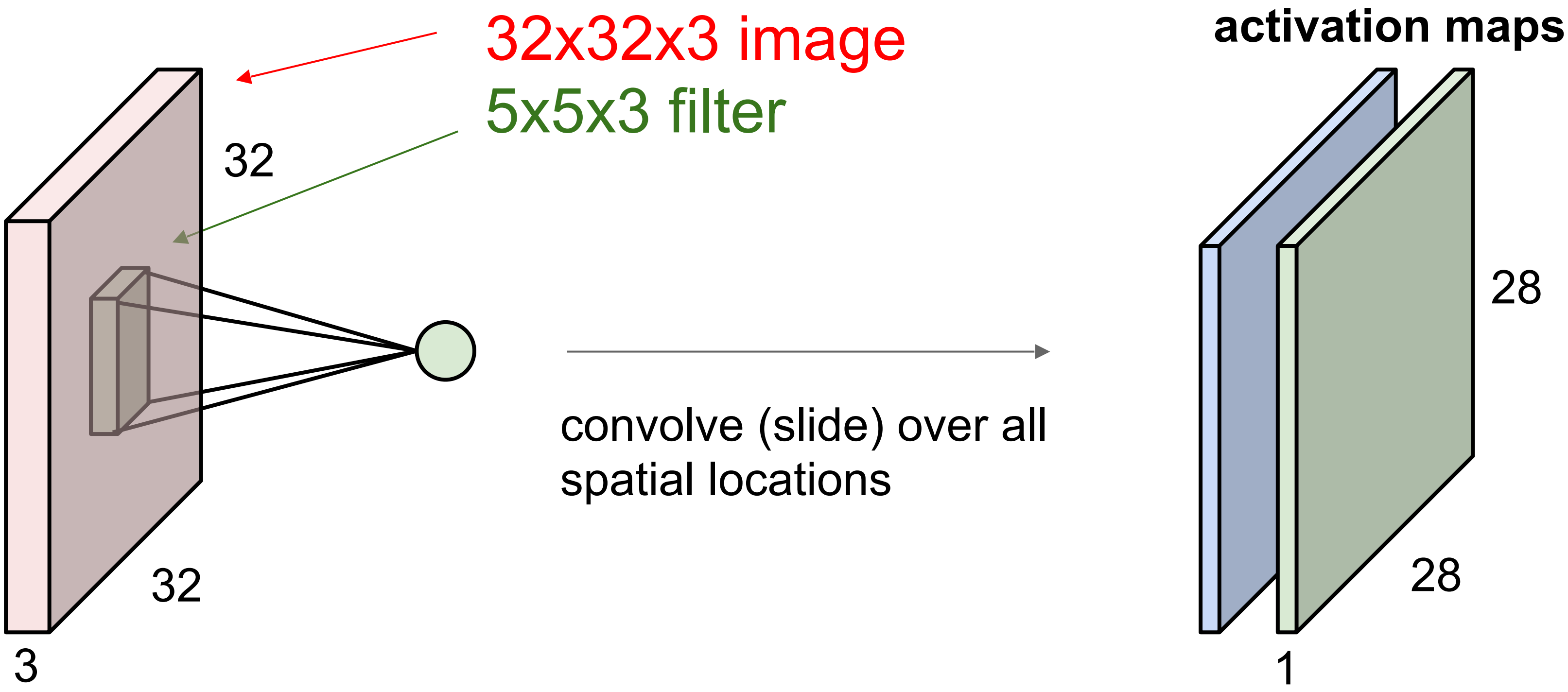
* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Convolution Layer

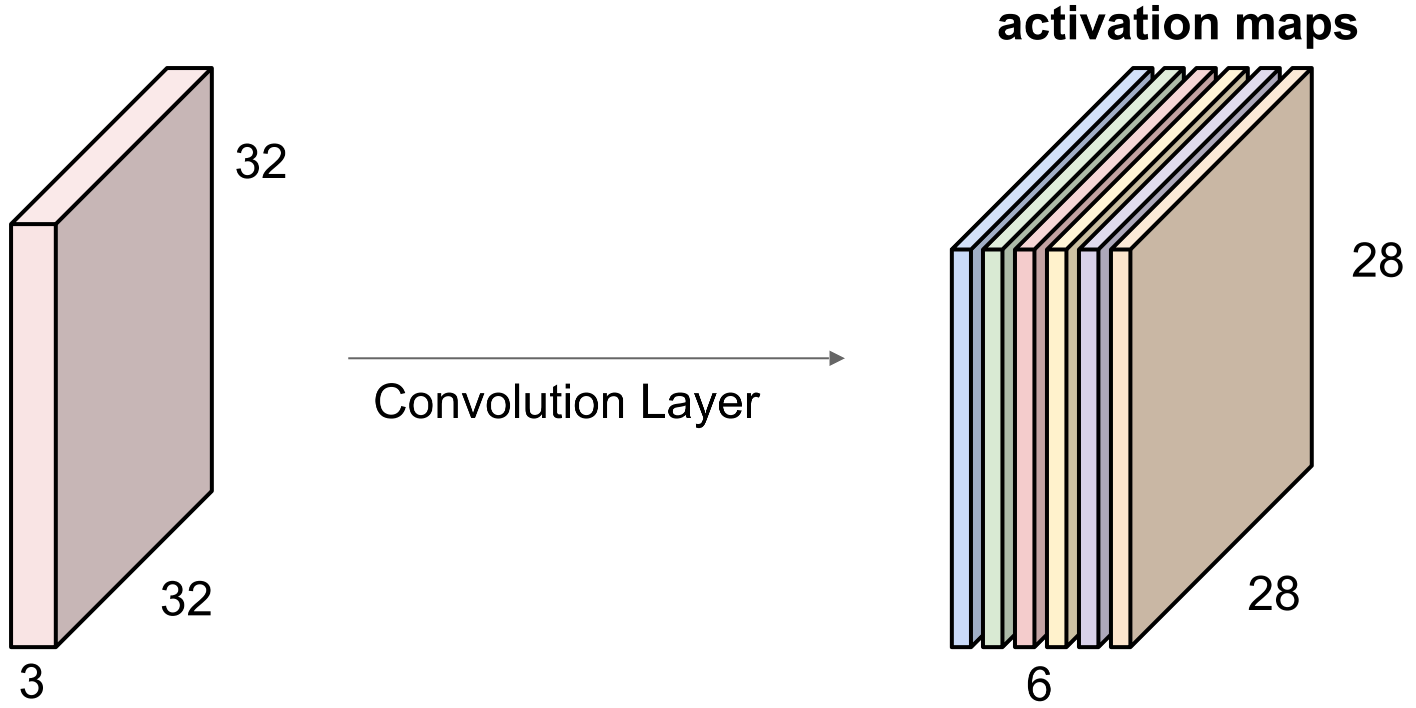


Convolution Layer

consider a second, green filter

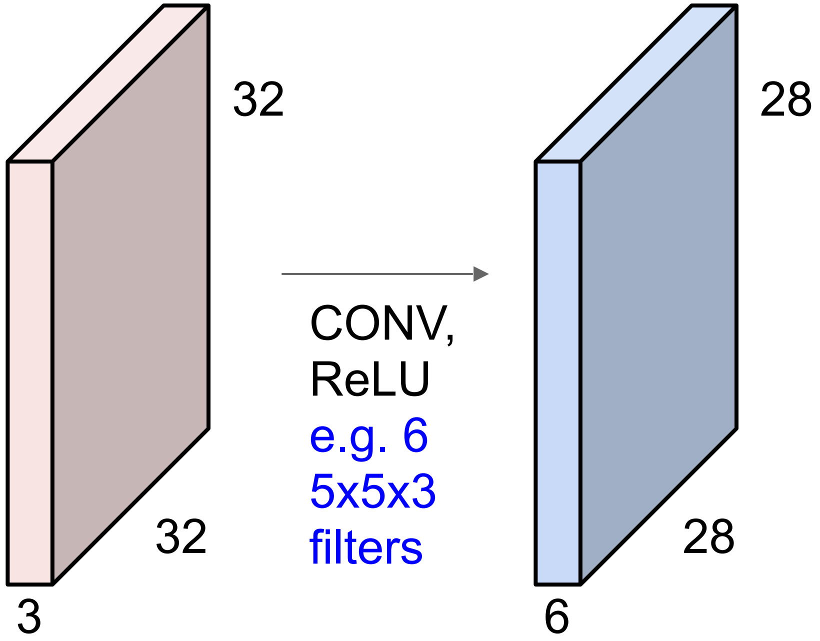


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

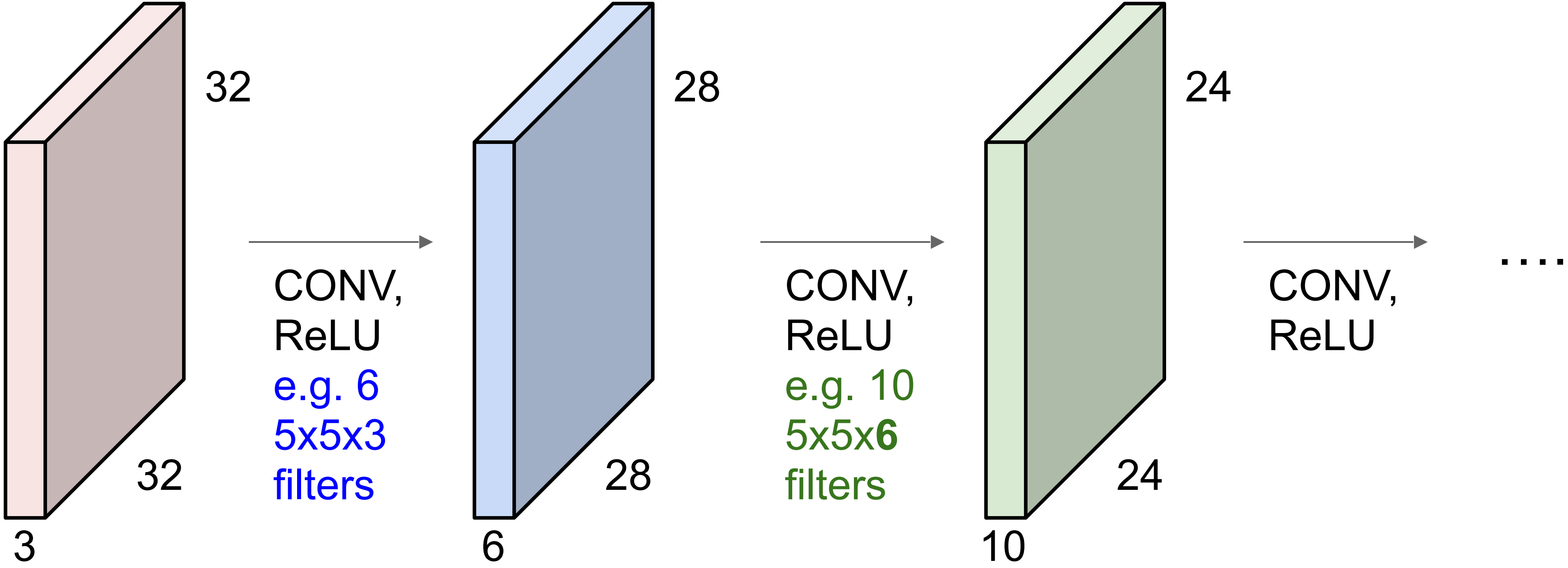


We stack these up to get a "new image" of size 28x28x6!

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

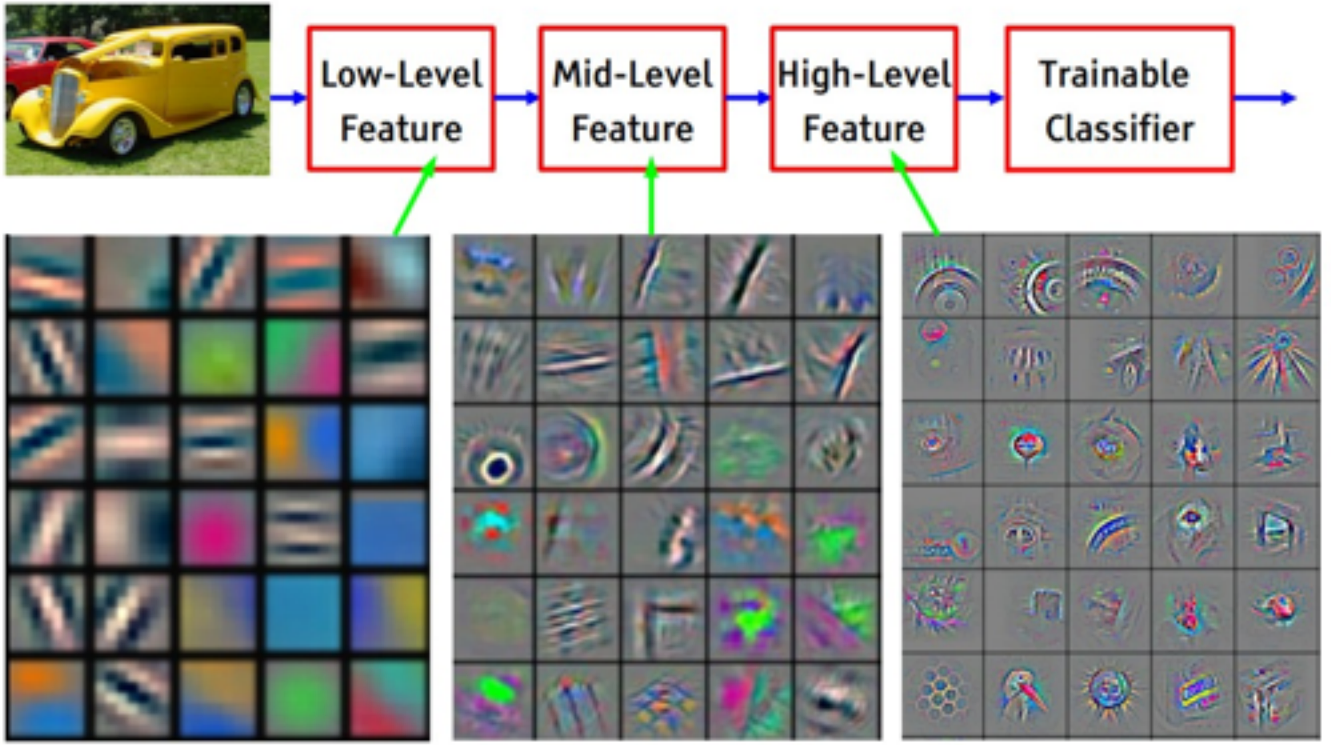


Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



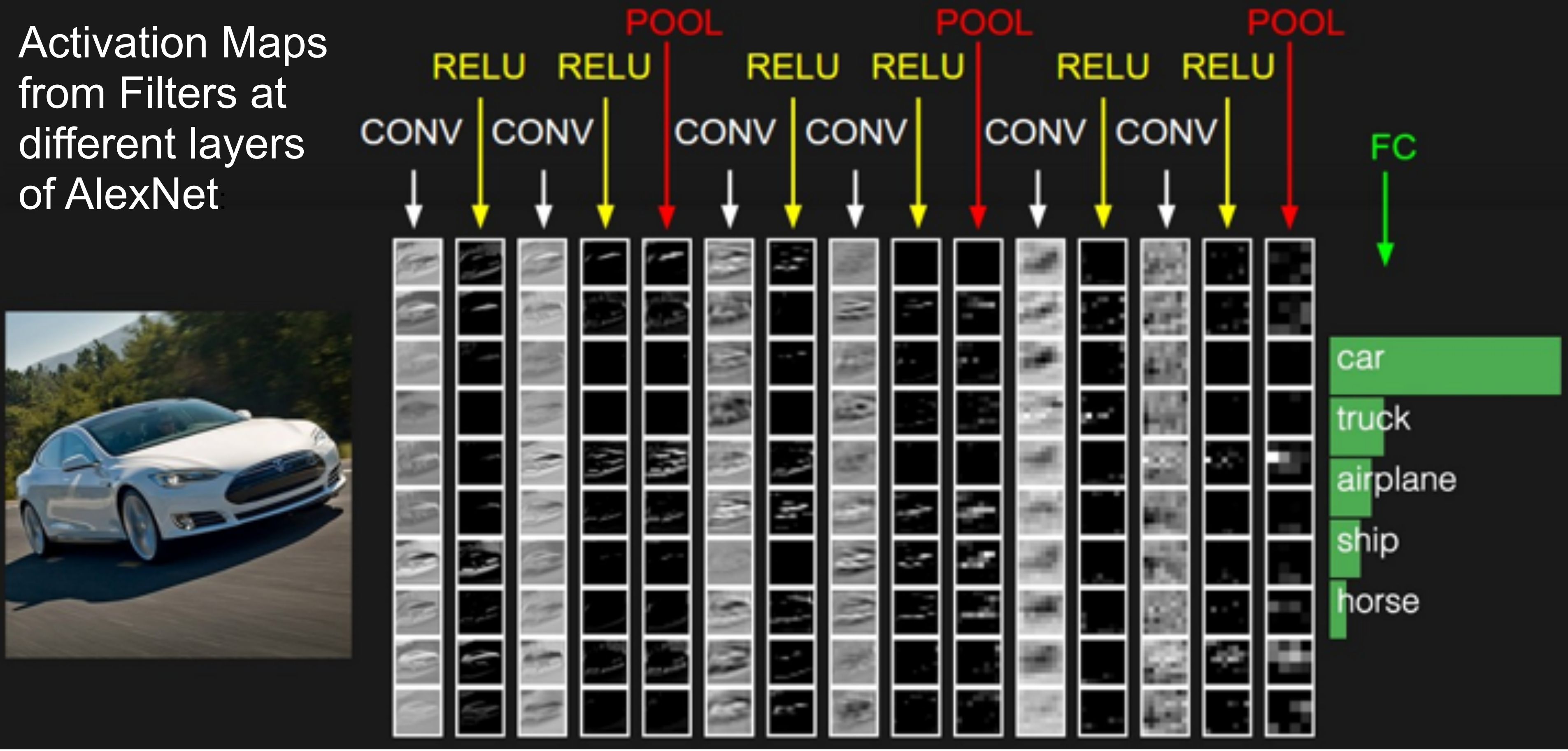
Preview

[From recent Yann LeCun slides]



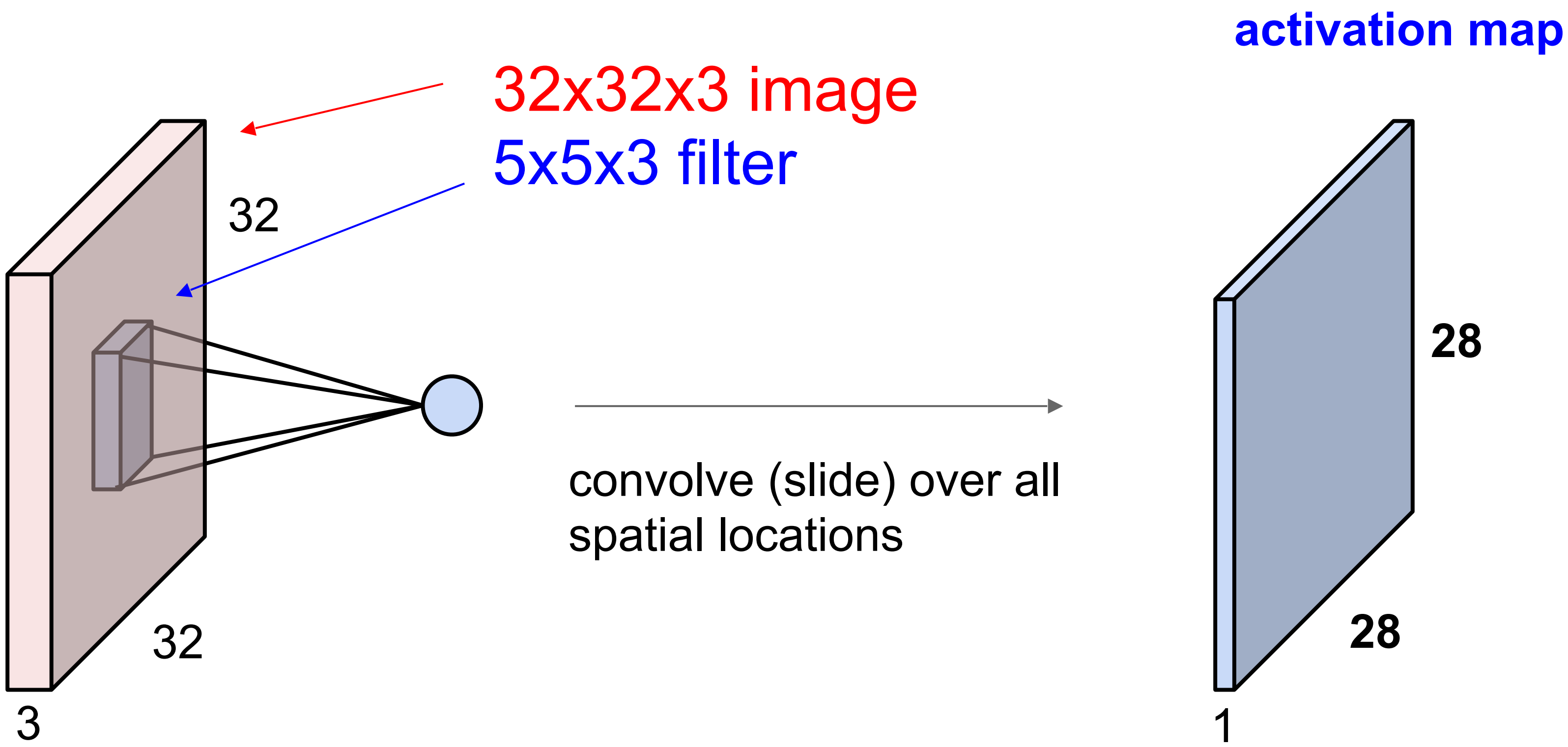
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Activation Maps from Filters at different layers of AlexNet:



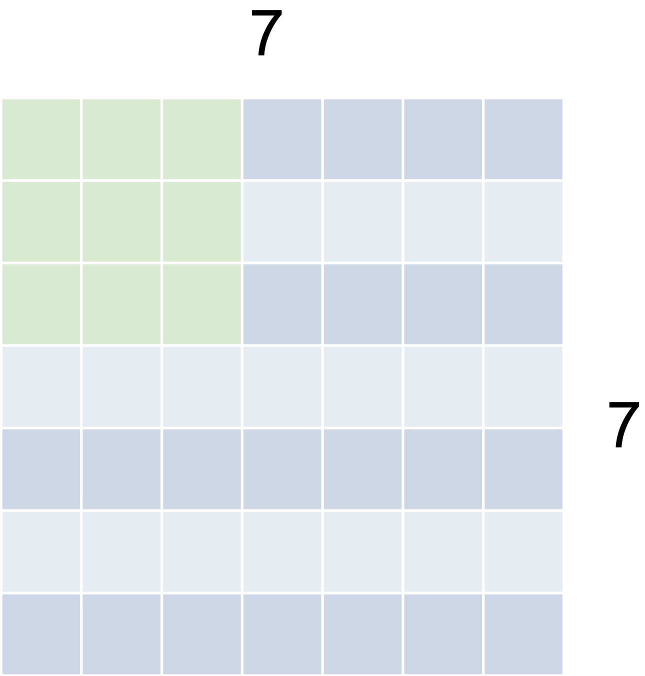
* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

A closer look at spatial dimensions:



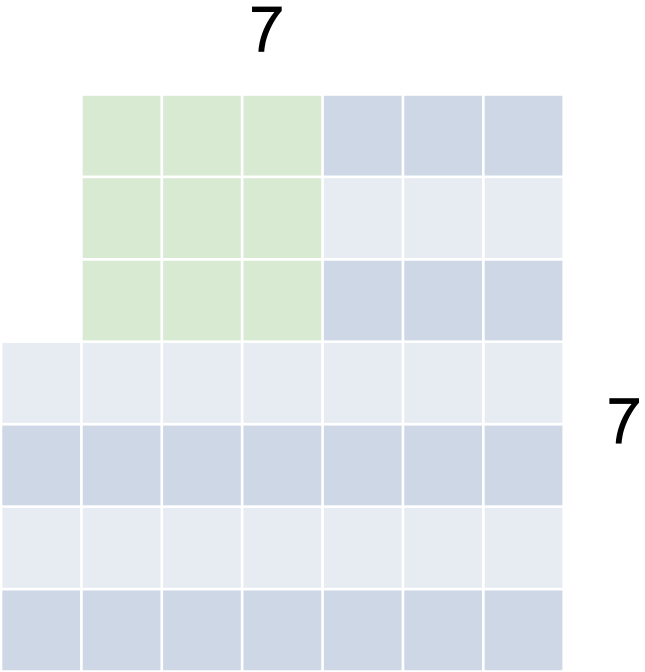
* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

A closer look at spatial dimensions:



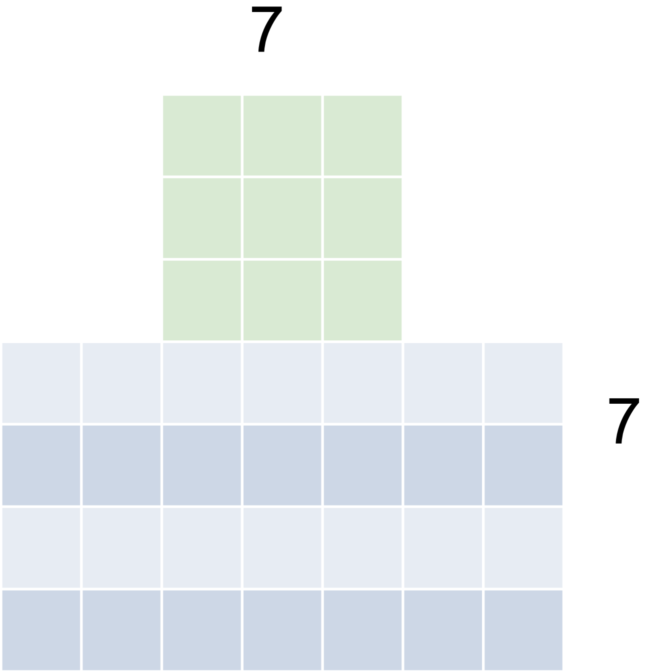
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

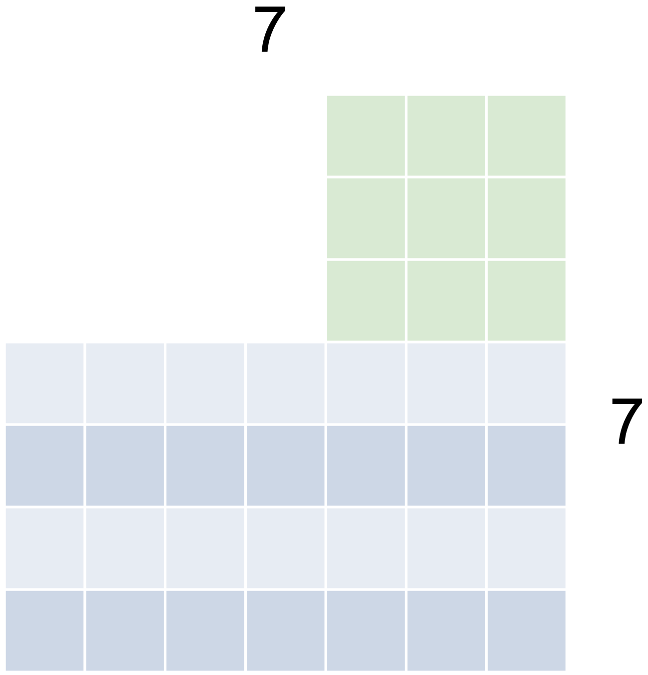


7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

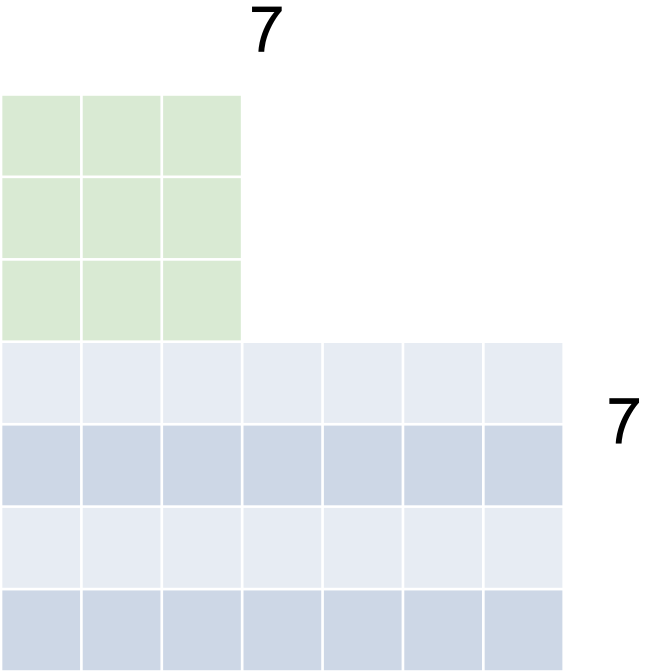
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

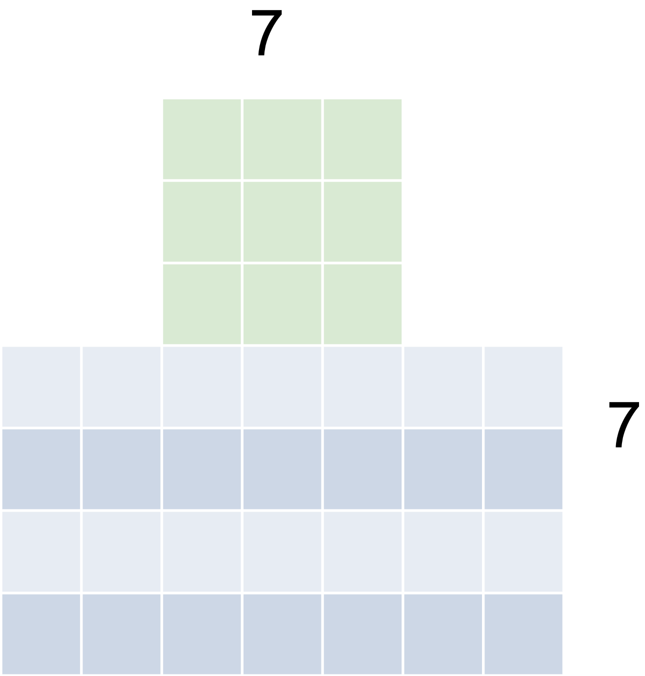
=> 5x5 output

A closer look at spatial dimensions:



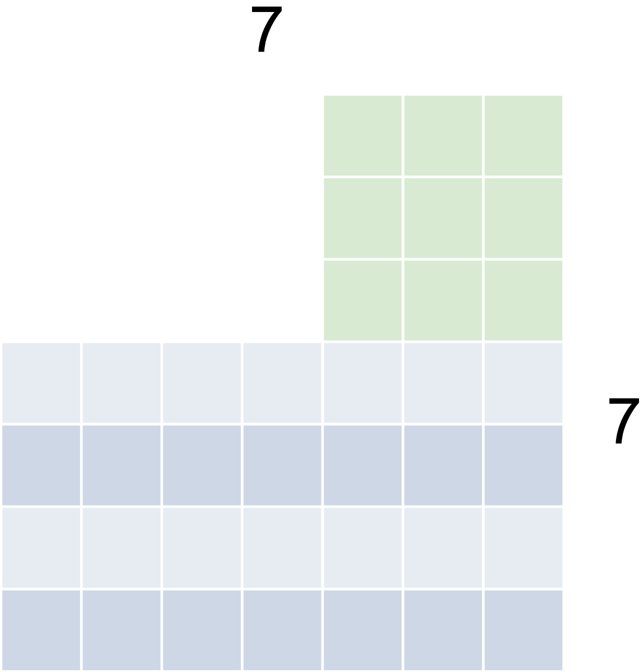
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



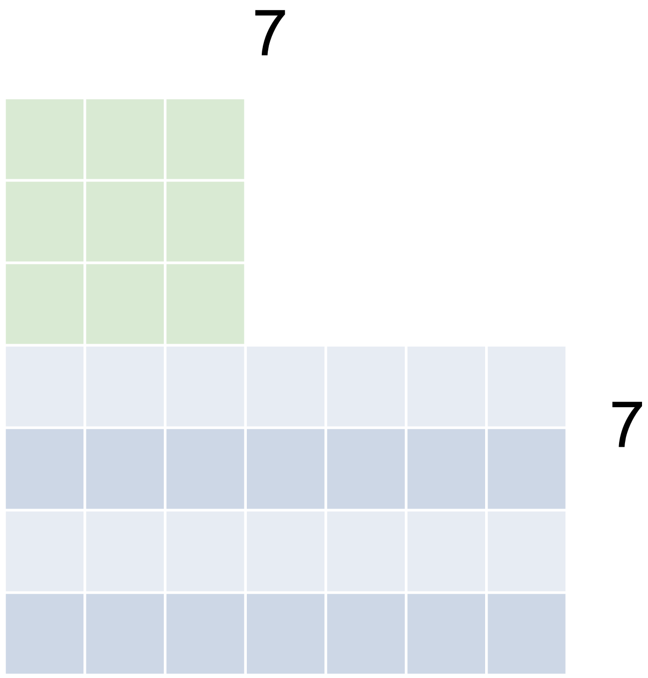
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



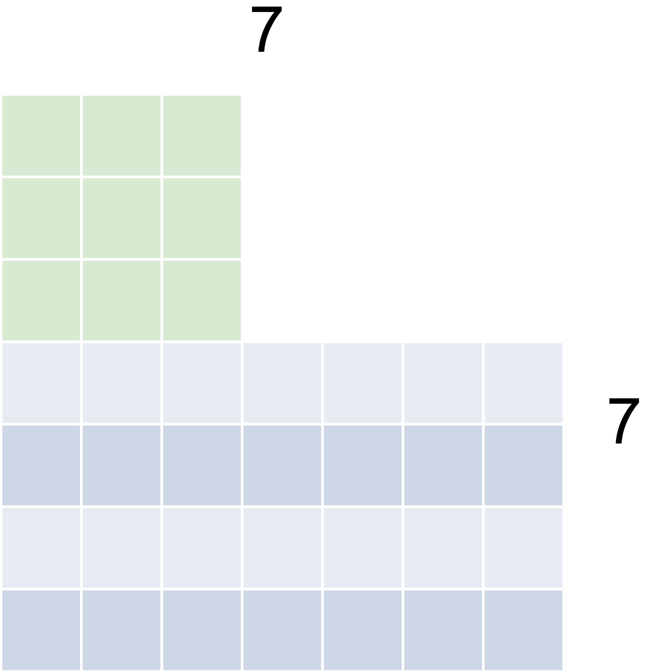
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



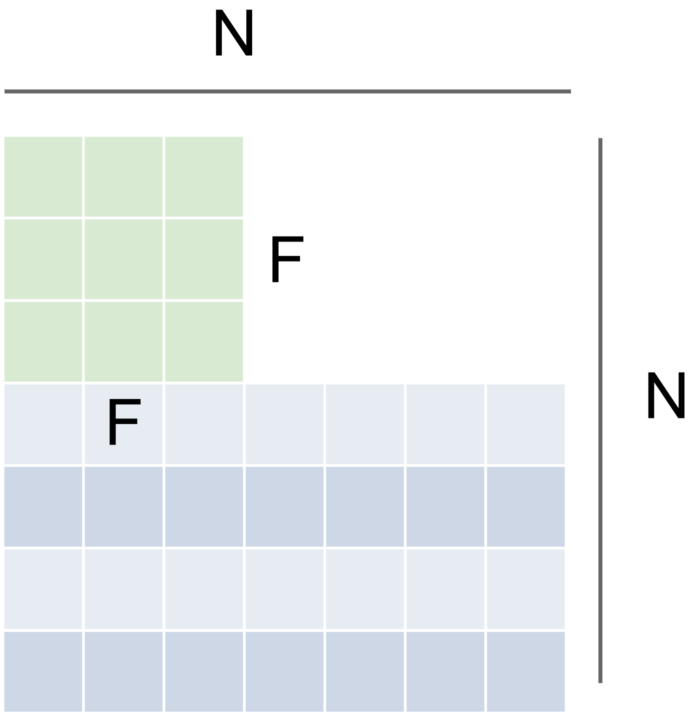
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

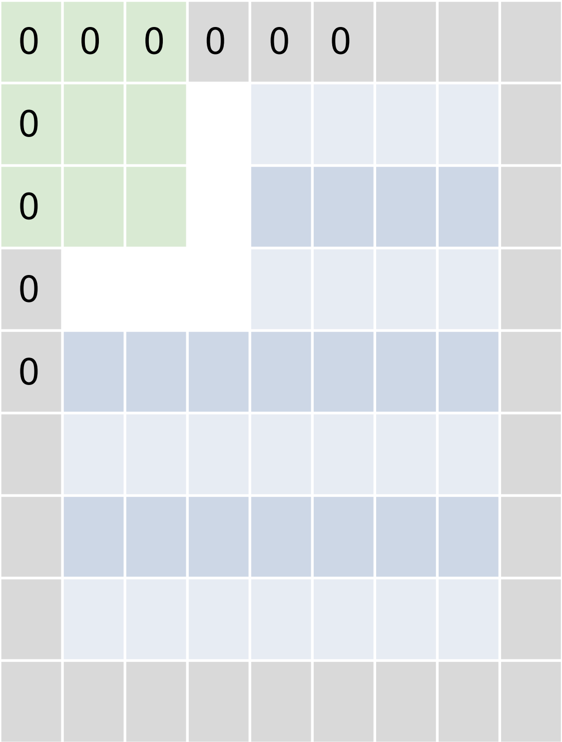
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 :\

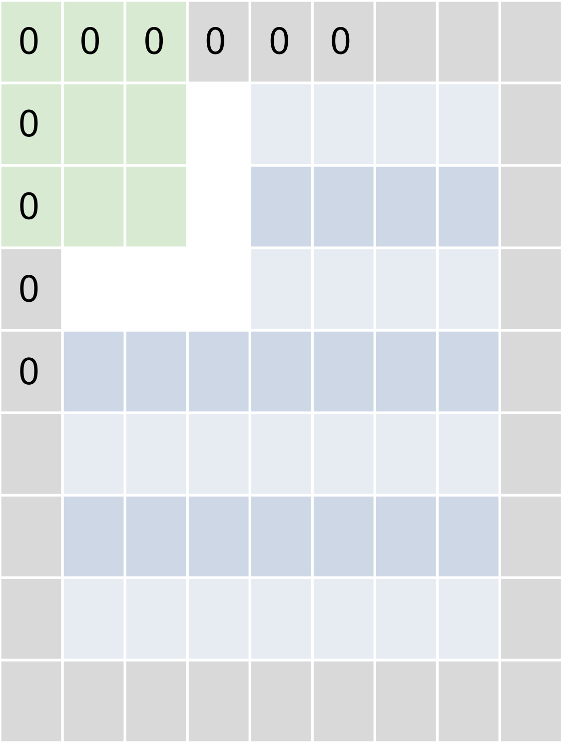
In practice: Common to zero pad the border



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

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

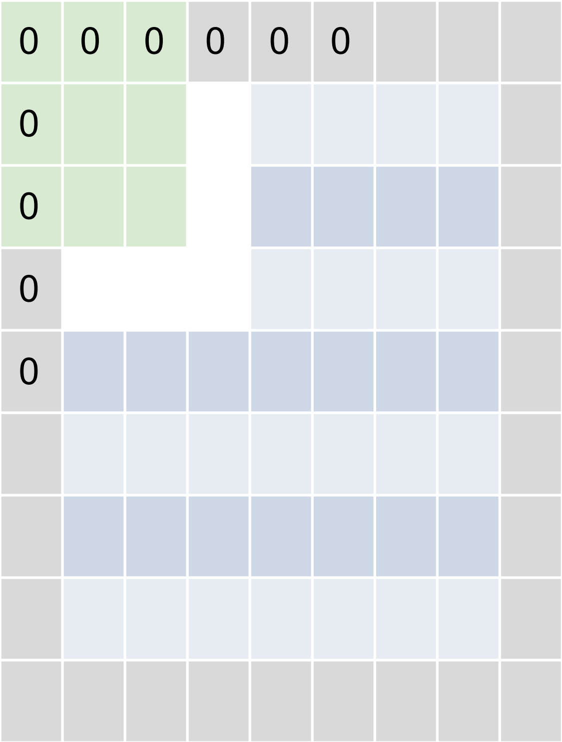
In practice: Common to zero pad the border



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

7x7 output!

In practice: Common to zero pad the border

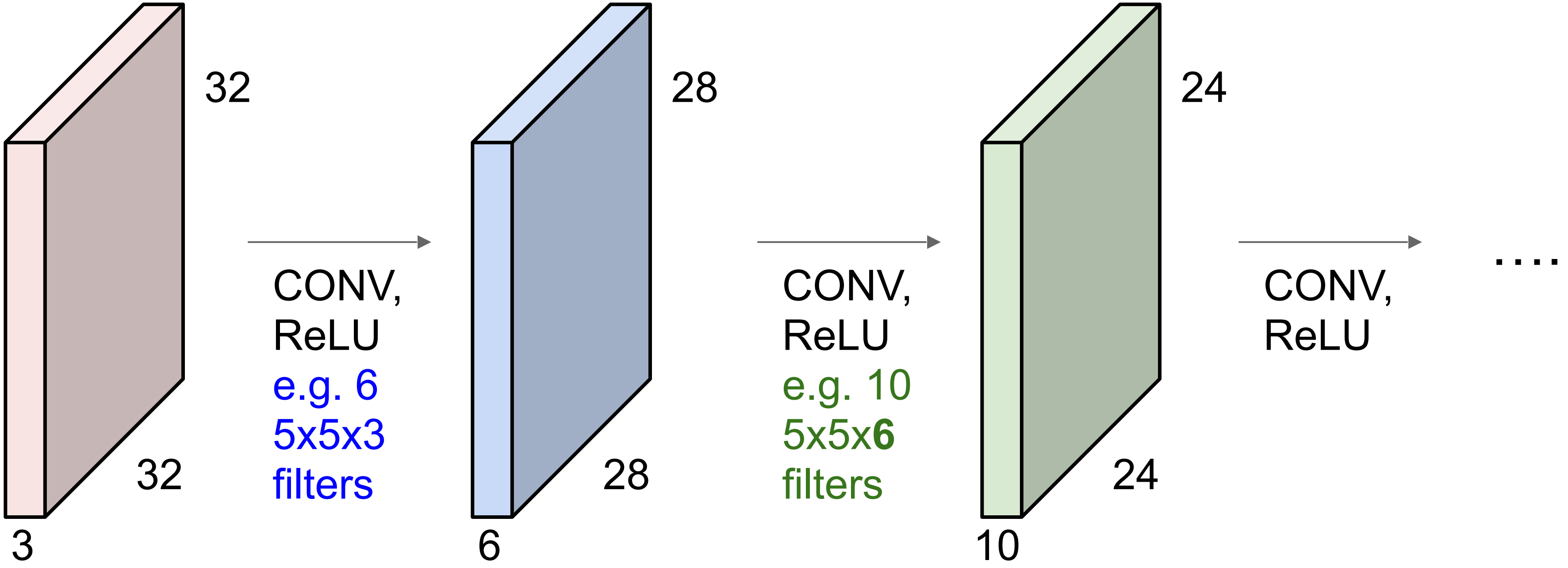


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

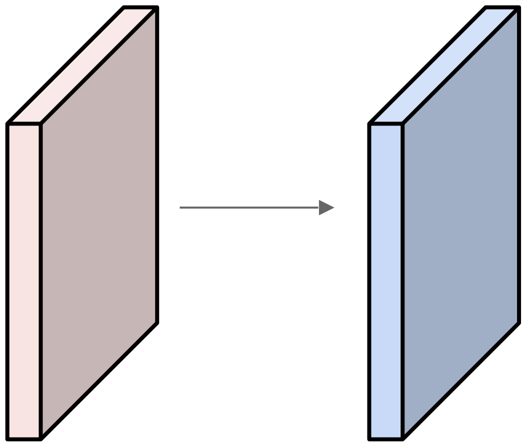
Remember back to…

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.



* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

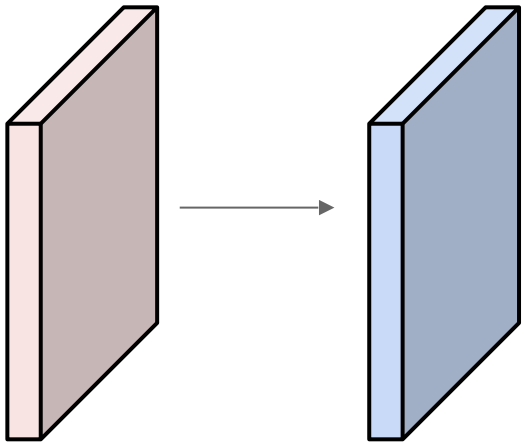
Examples time:



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

Output volume size: ?

Examples time:



Input volume: **32x32x3**

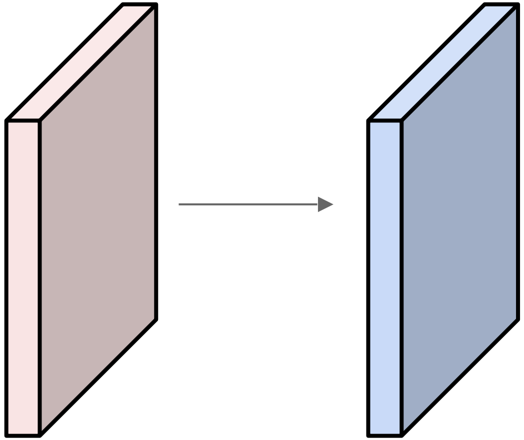
10 5x5 filters with stride 1, pad 2

Output volume size:

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

32x32x10

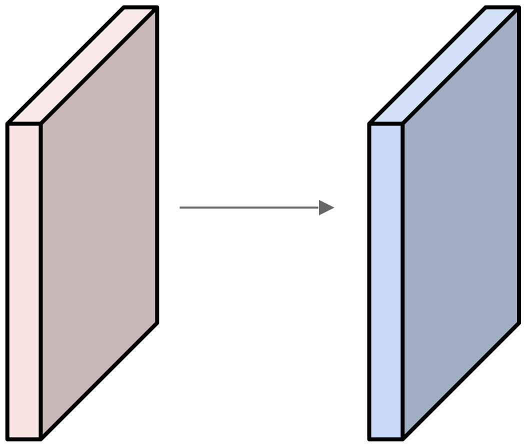
Examples time:



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

Number of parameters in this layer?

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**

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.

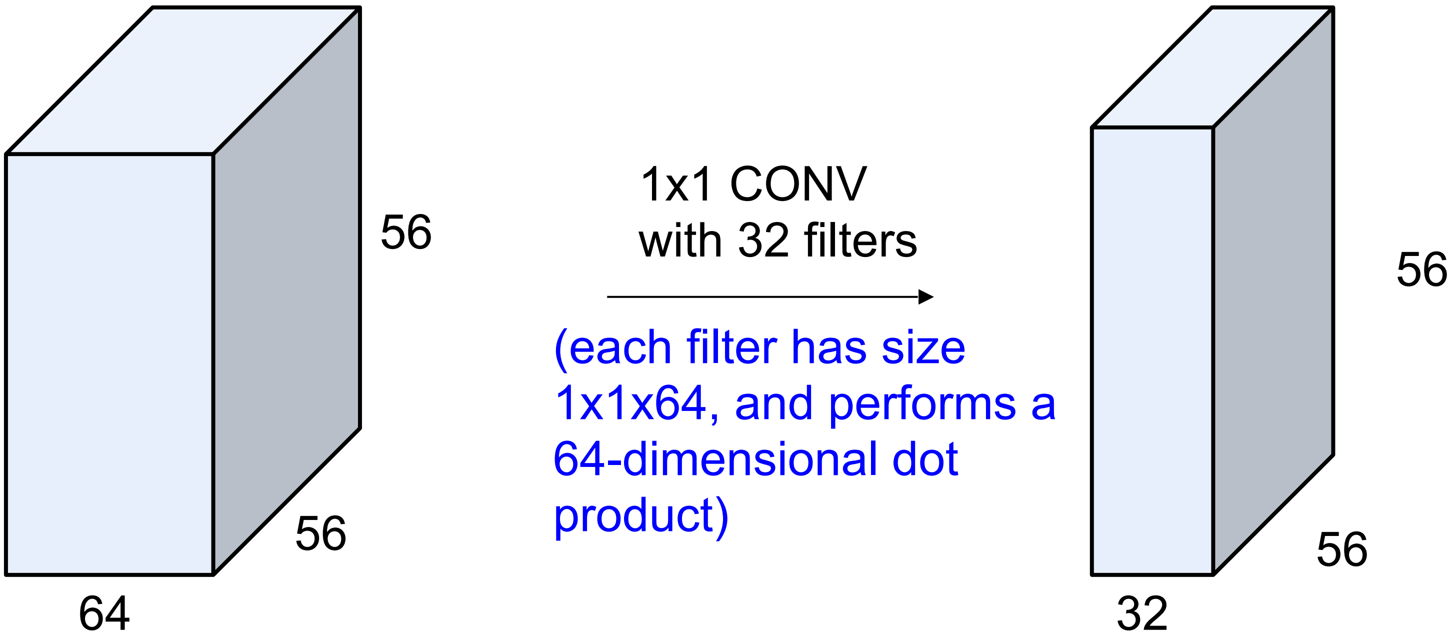
Common settings:

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)



Example: CONV layer in Torch

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.

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

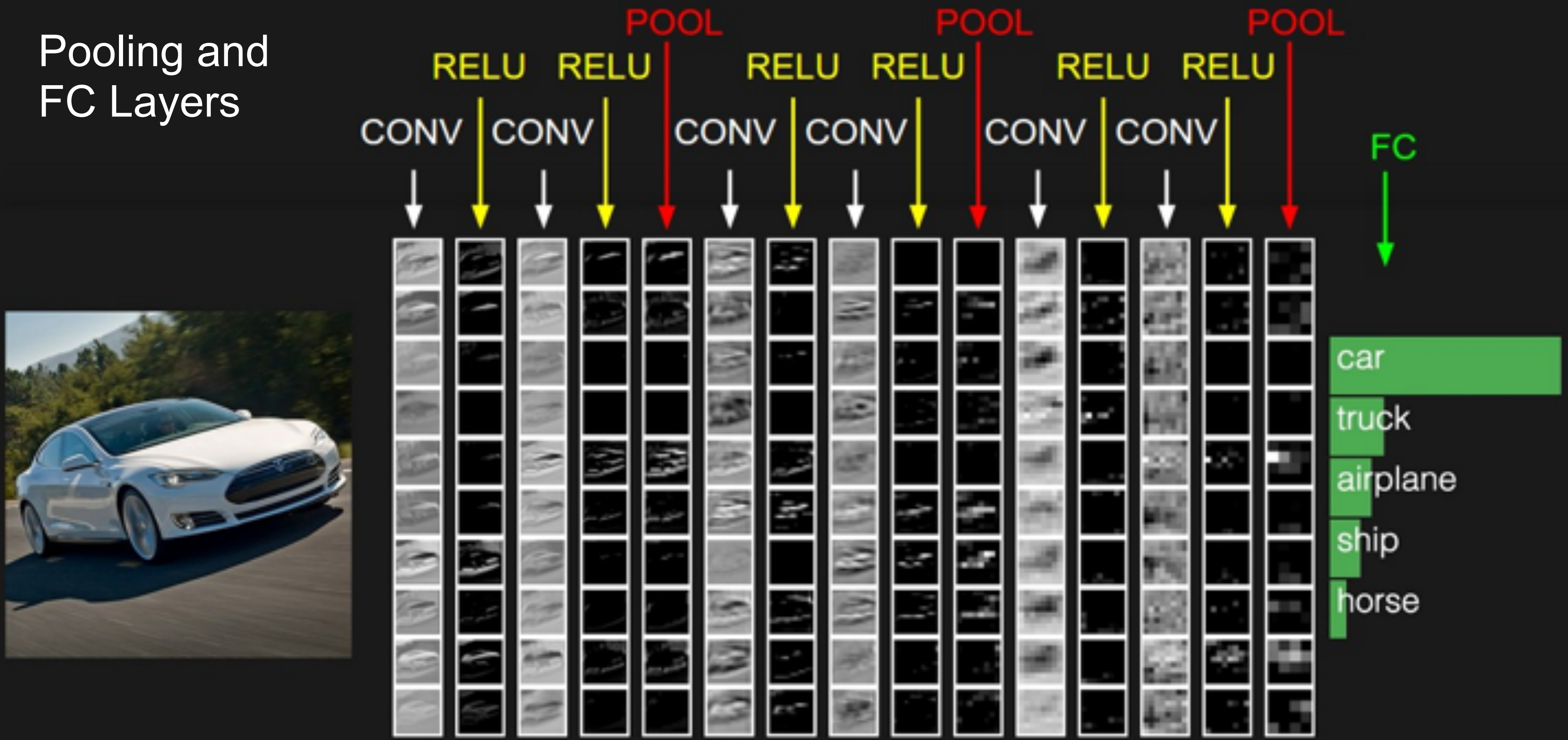
Example: CONV layer in Caffe

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Summary. To summarize, the Conv Layer:

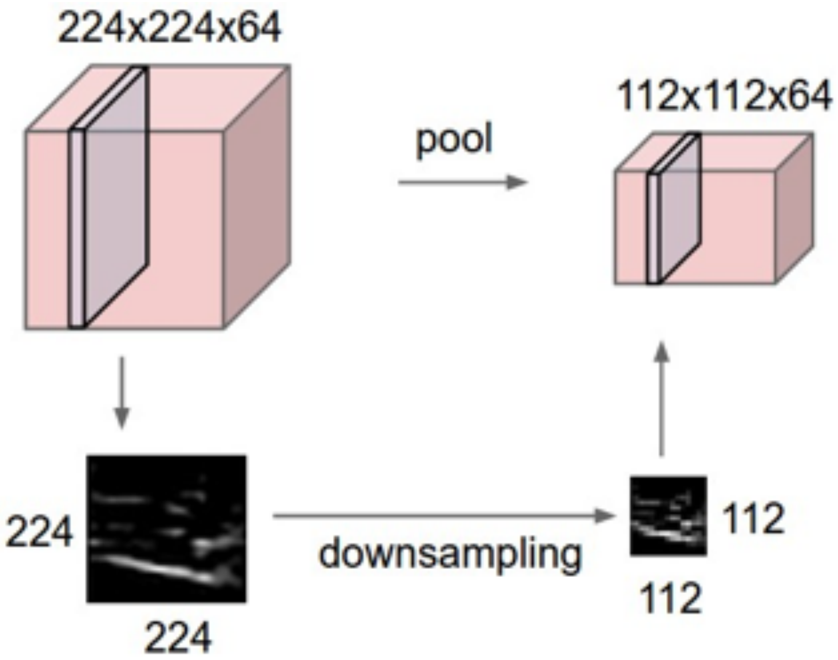
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

Pooling and FC Layers



Pooling layer

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



MAX POOLING

Single depth slice

x

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

y

max pool with 2x2 filters
and stride 2

6	8
3	4

Summary of Pooling Layer

- 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

Summary of Pooling Layer

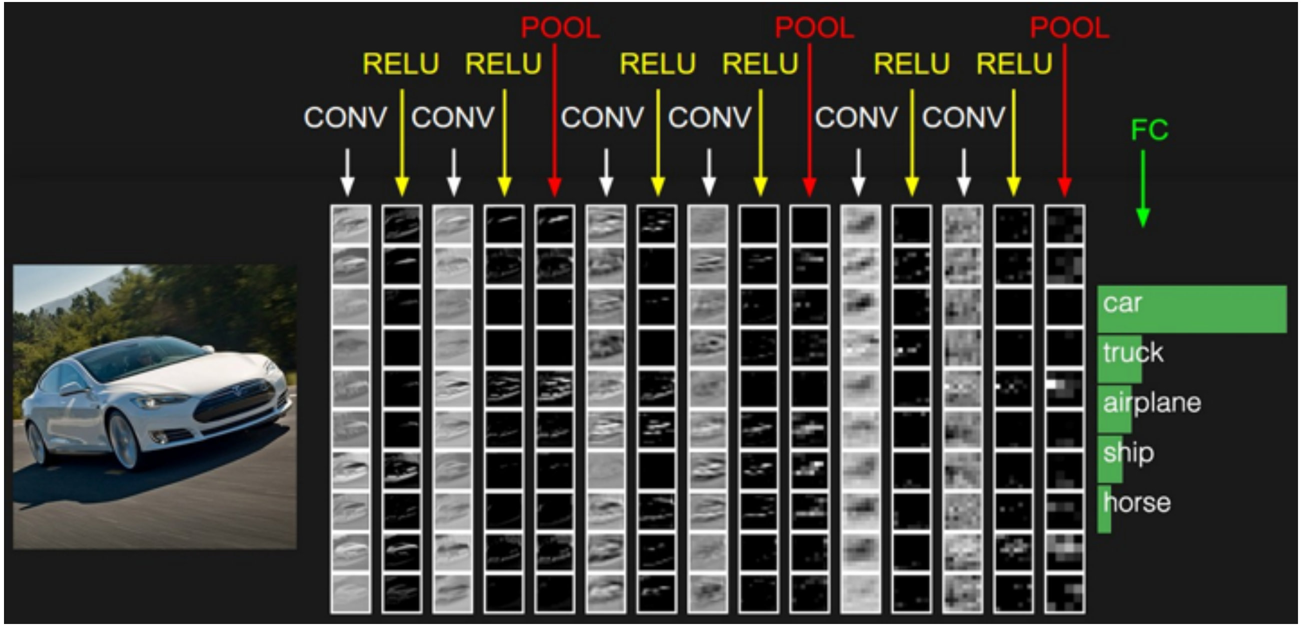
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since 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

Fully Connected Layer (FC layer)

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



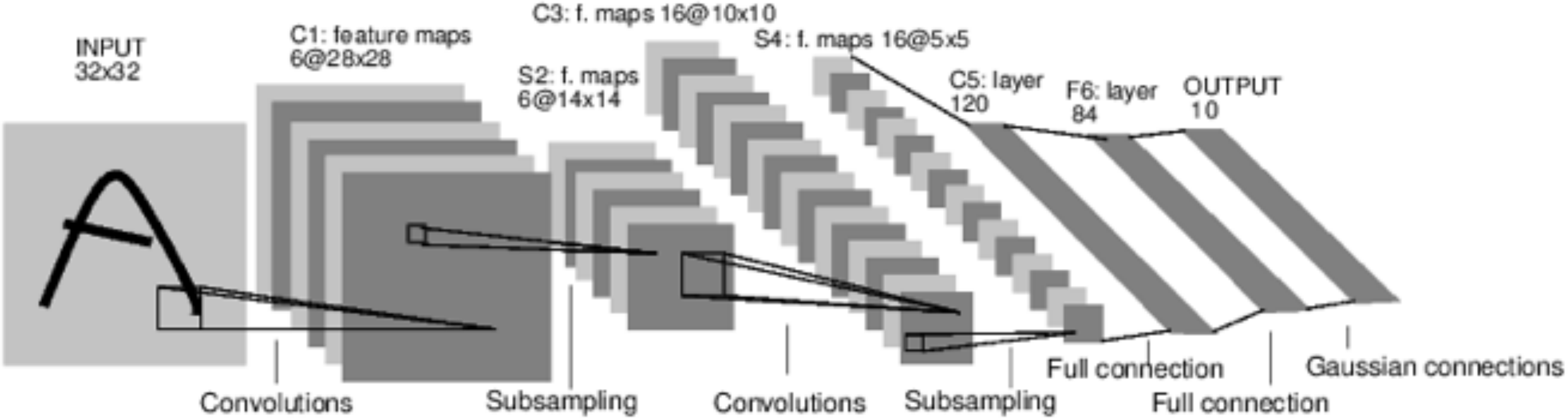
ConvNetJS demo: training on CIFAR-10

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

* Original slides borrowed from And and Li Fei-Fei, Stanford cs231n

Case Study: LeNet-5

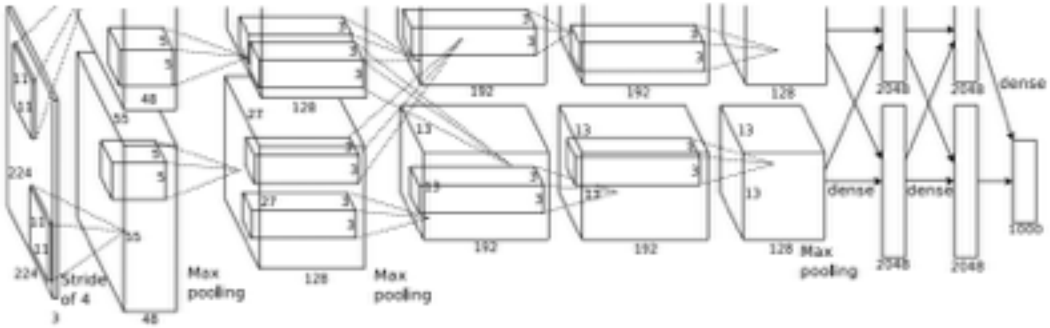
[LeCun et al., 1998]



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]

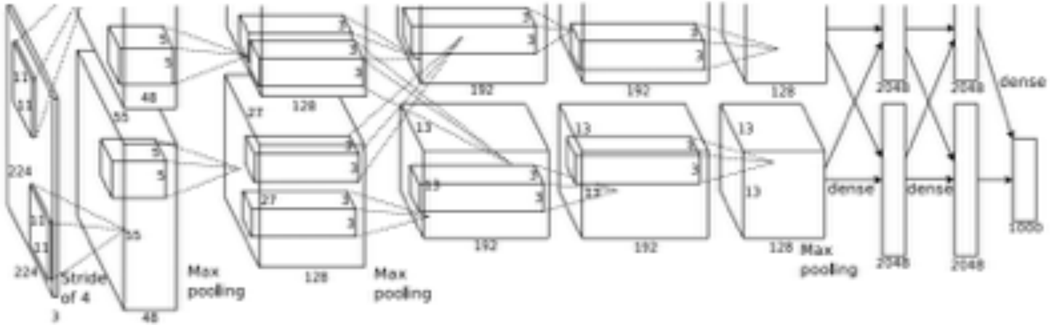


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet

[Krizhevsky et al. 2012]



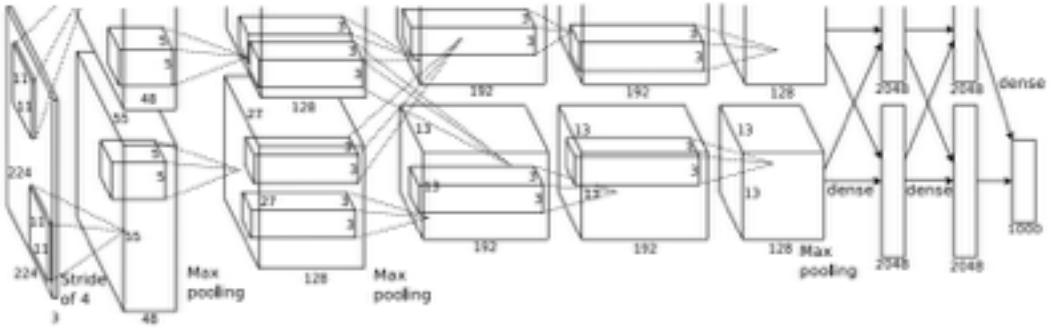
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

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

Case Study: AlexNet

[Krizhevsky et al. 2012]

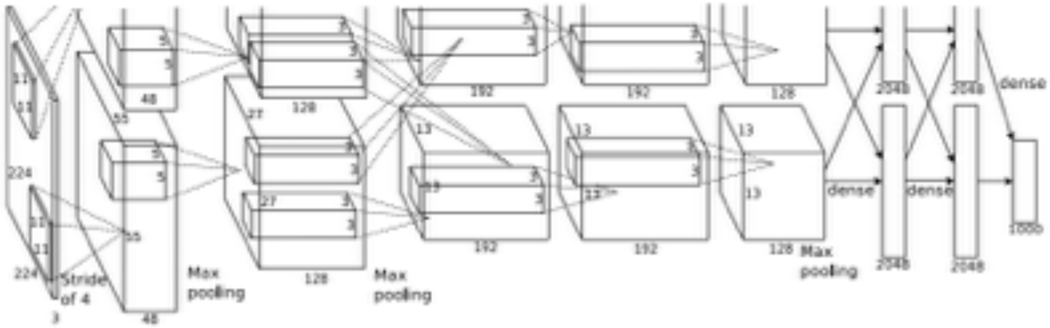


Input: 227x227x3 images

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]



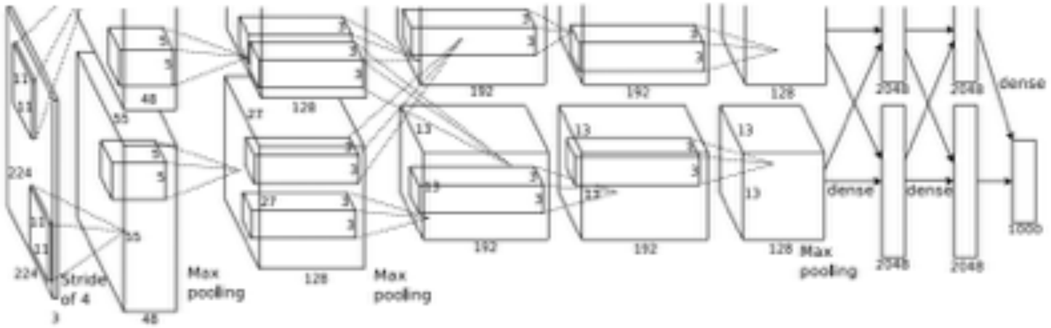
Input: 227x227x3 images
After CONV1: 55x55x96

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]



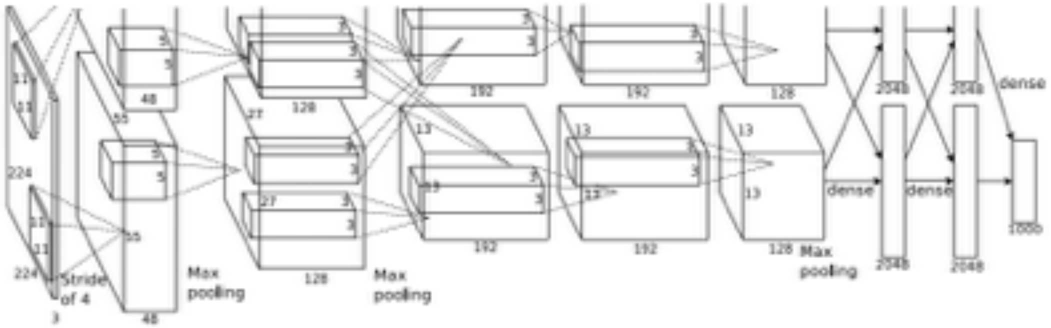
Input: 227x227x3 images
After CONV1: 55x55x96

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

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

Case Study: AlexNet

[Krizhevsky et al. 2012]

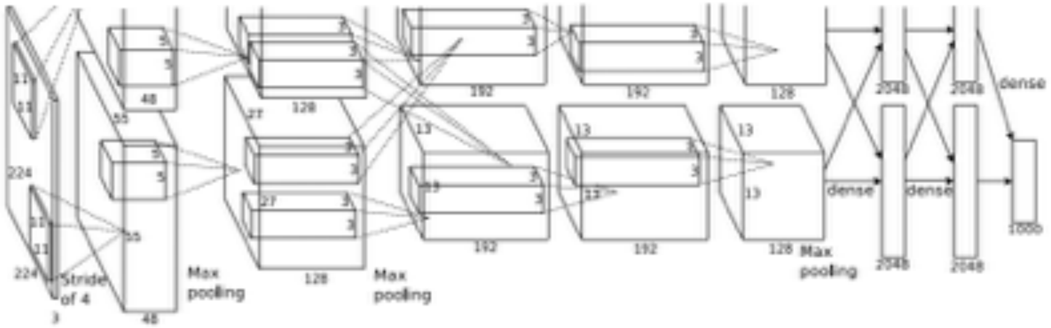


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Case Study: AlexNet

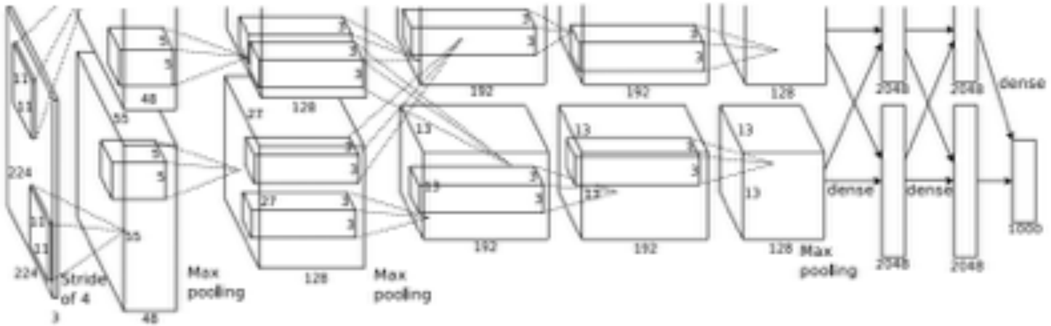
[Krizhevsky et al. 2012]



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

Case Study: AlexNet

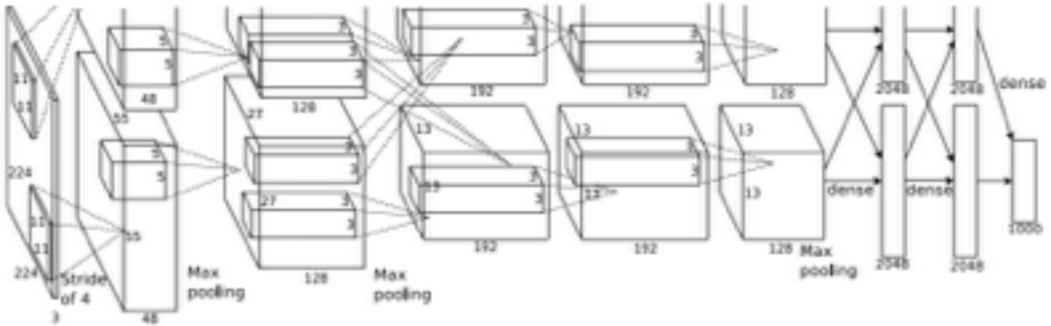
[Krizhevsky et al. 2012]



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: VGGNet

[Simonyan and Zisserman, 2014]

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

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

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

(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			

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
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 ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

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	
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			

(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:

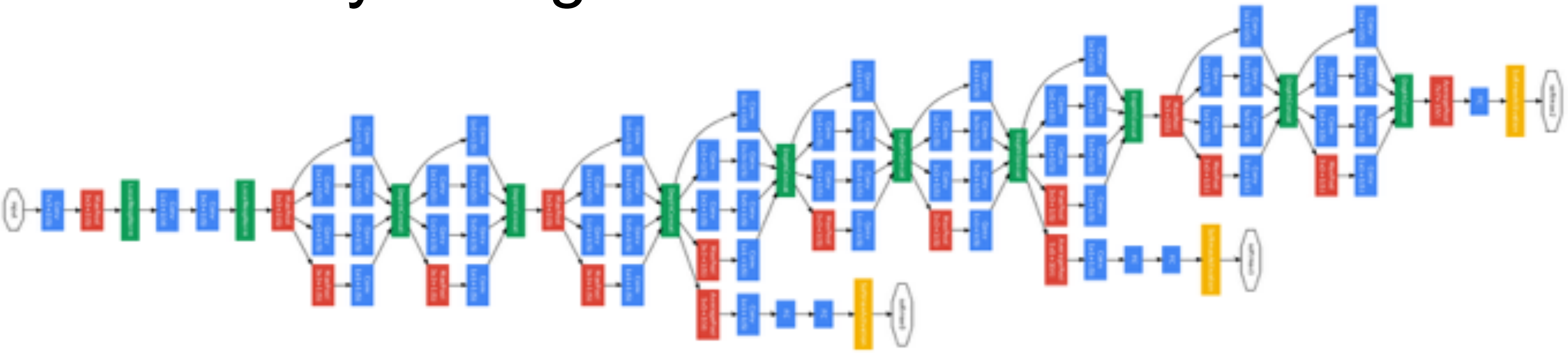
Most memory is in early CONV

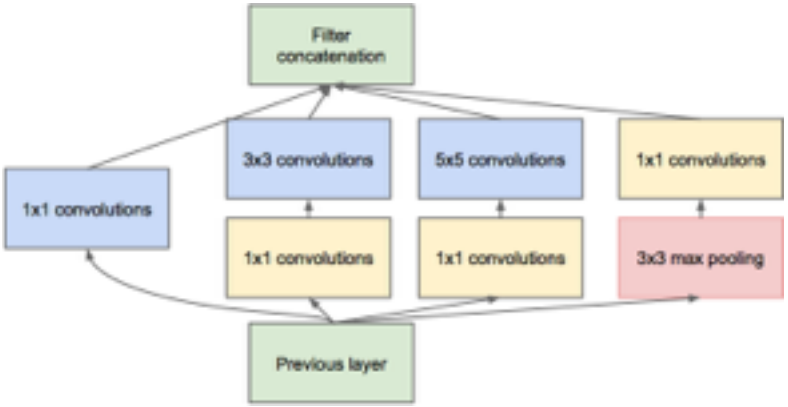
Most params are in late FC

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)

Case Study: GoogLeNet

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

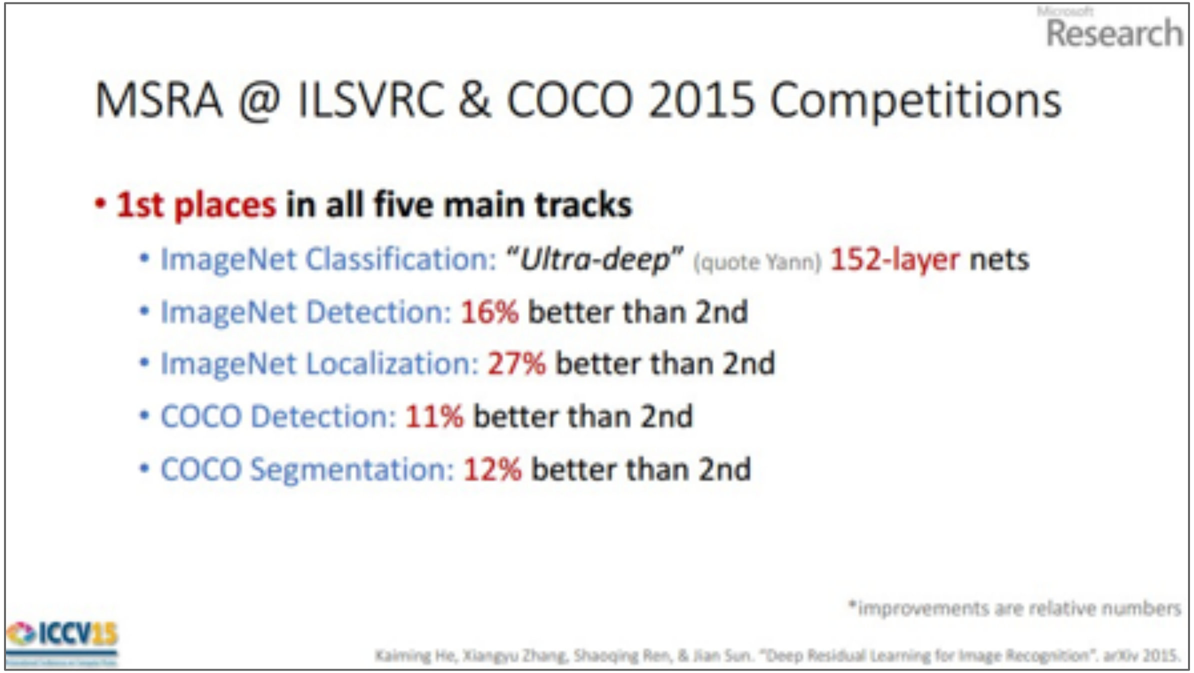
Fun features:

- Only 5 million params!
(Removes FC layers completely)

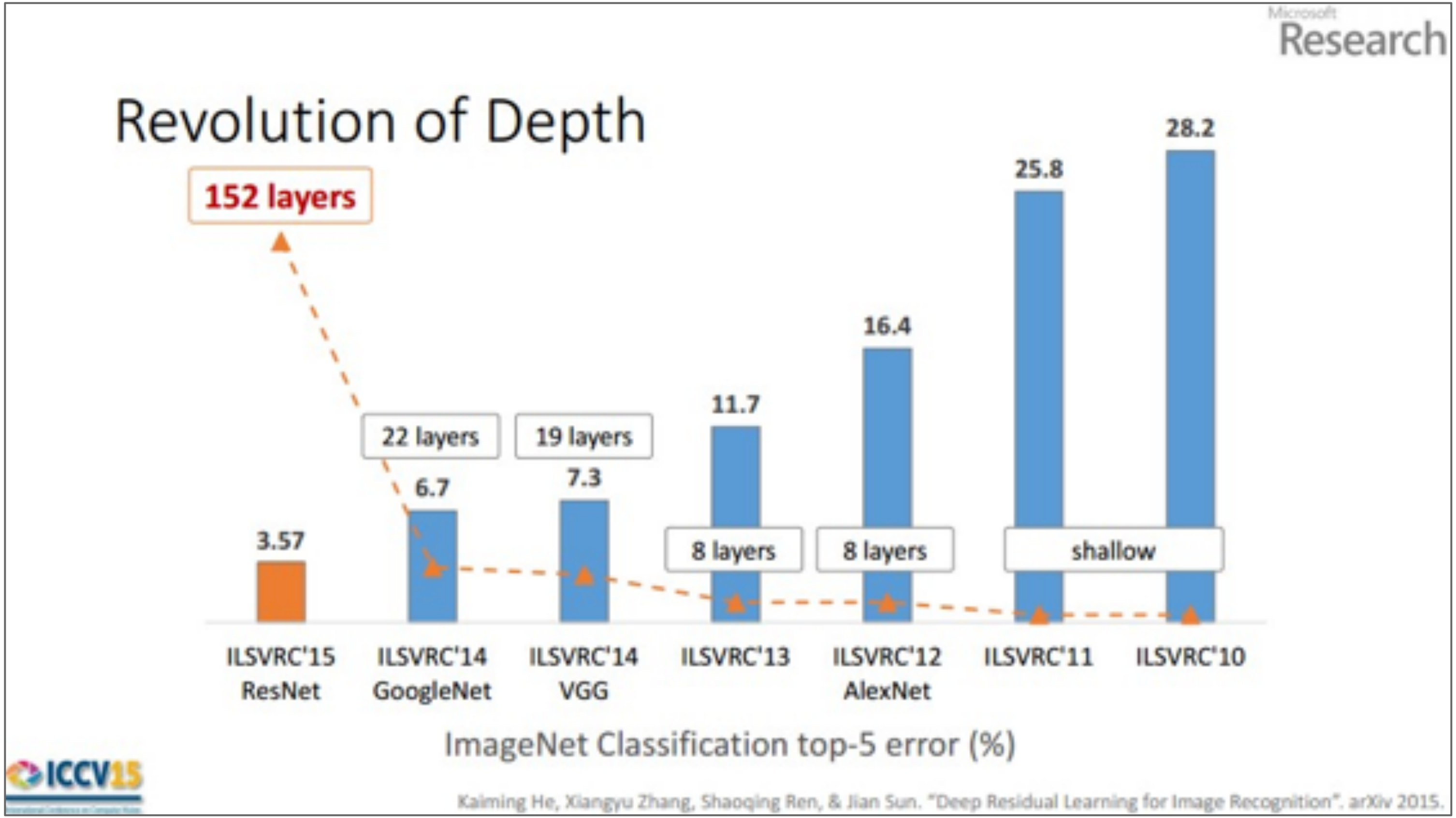
Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

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

ILSVRC 2015 winner (3.6% top 5 error)

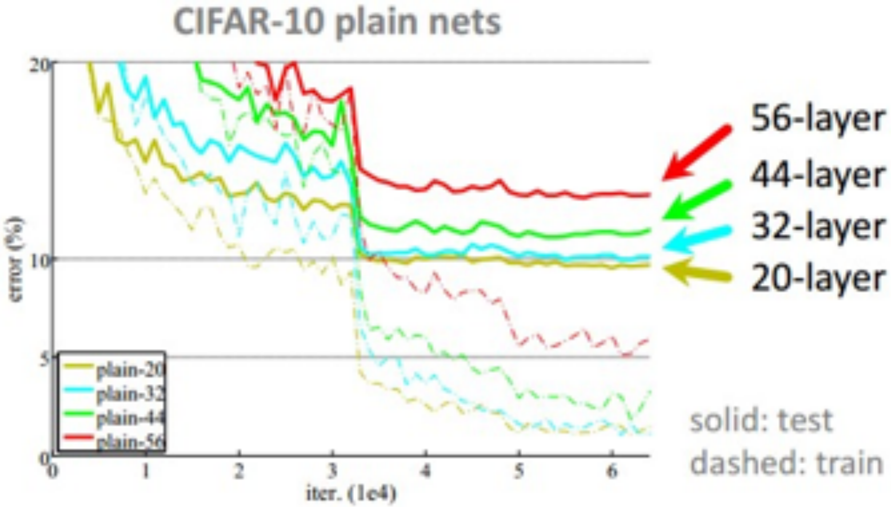


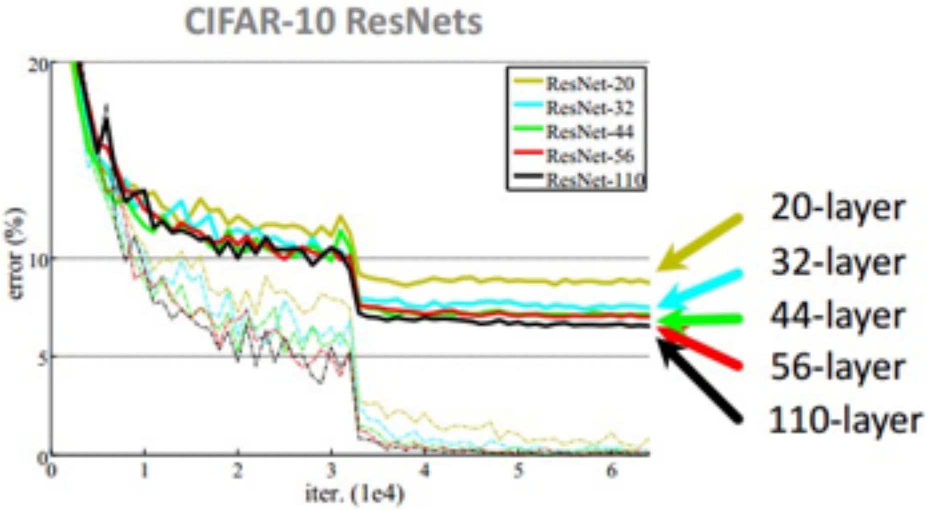
Slide from Kaiming He's ICCV 2015 presentation https://www.youtube.com/watch?v=1PGLj-uKT1w



(slide from Kaiming He's ICCV 2015 presentation)

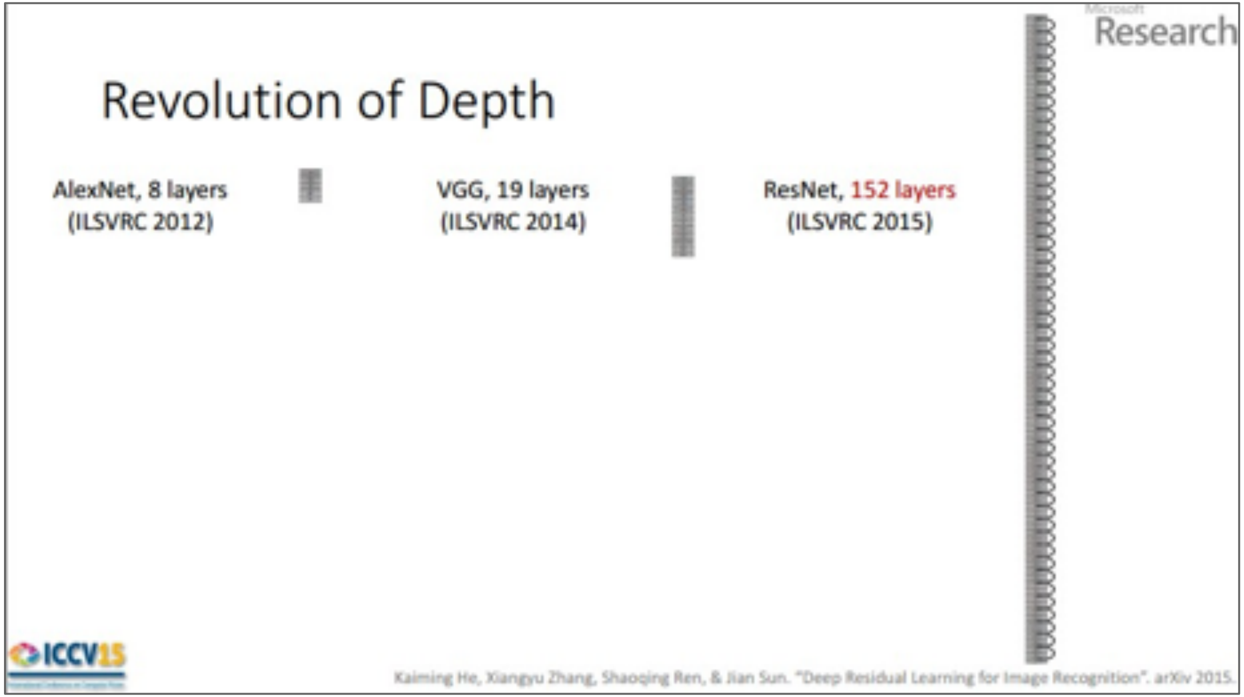
CIFAR-10 experiments





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

ILSVRC 2015 winner (3.6% top 5 error)



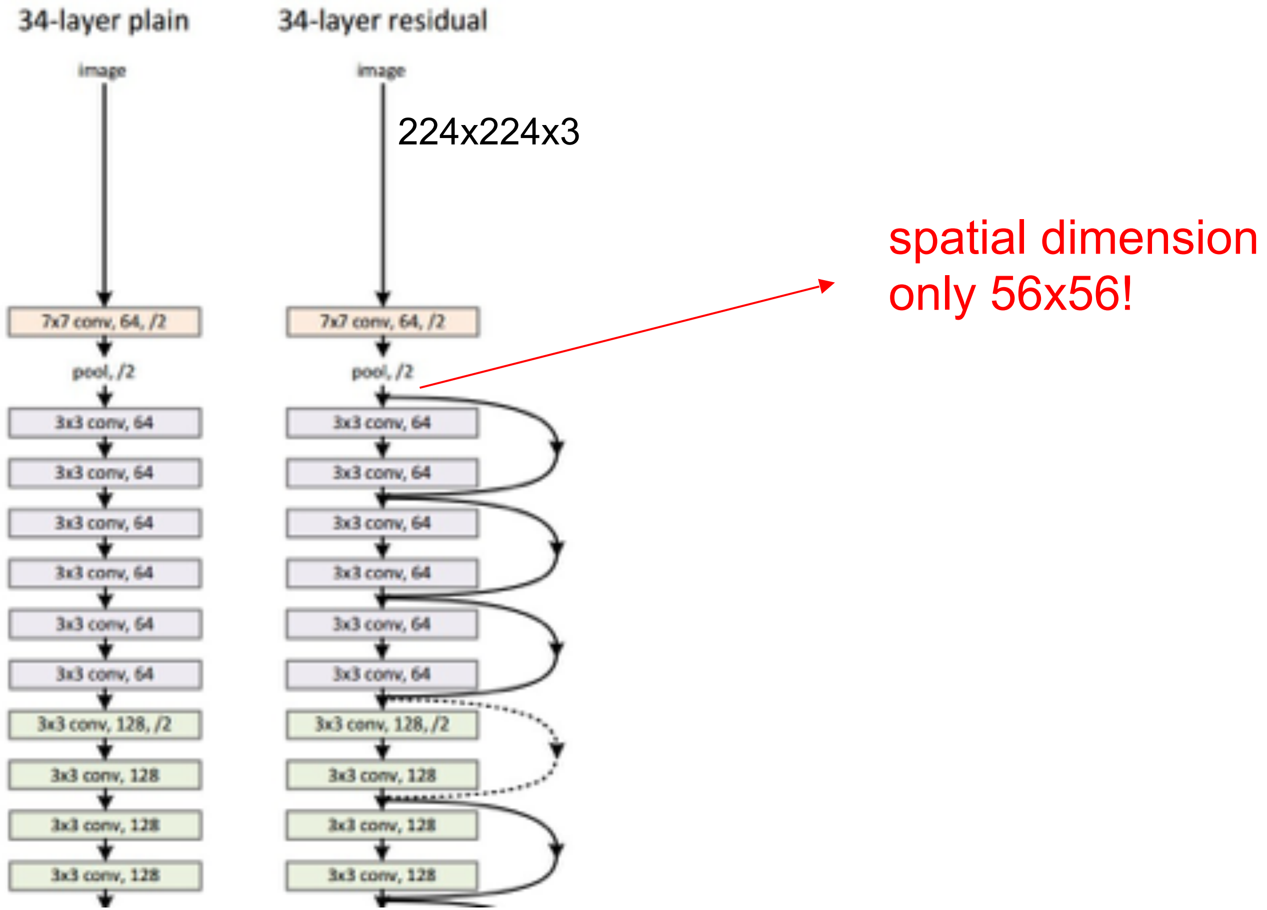
(slide from Kaiming He's ICCV 2015 presentation)

2-3 weeks of training on 8 GPU machine

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

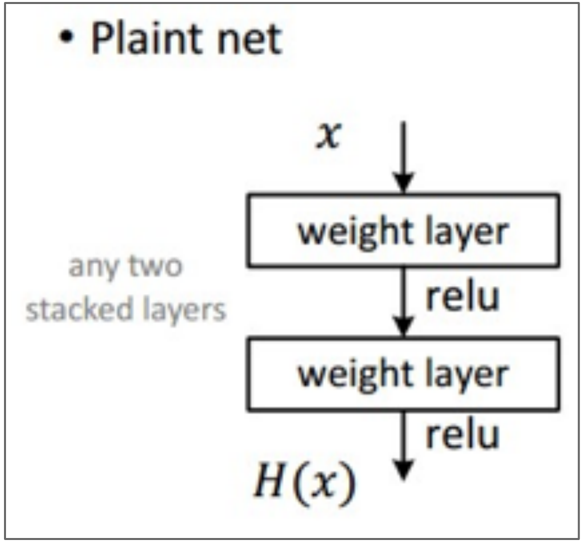
Case Study: ResNet

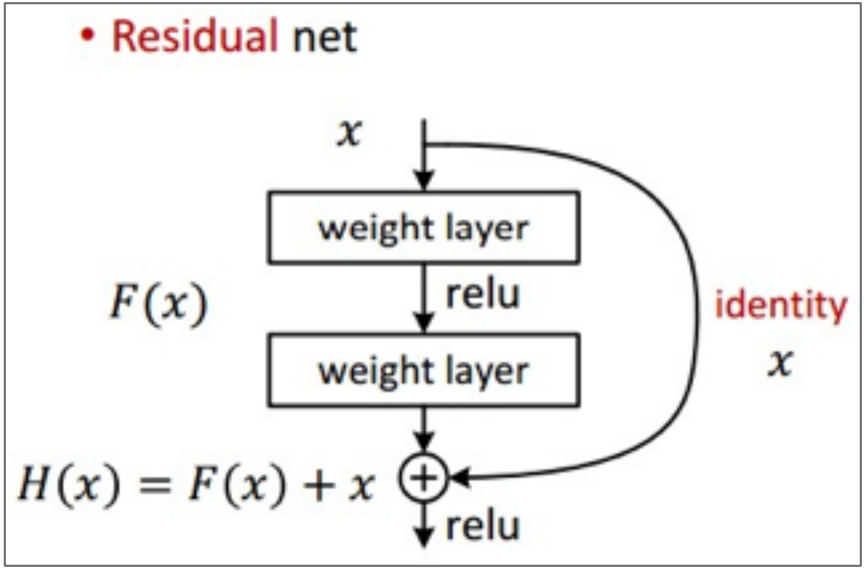
[He et al., 2015]



Case Study: ResNet

[He et al., 2015]





Case Study: ResNet

[He et al., 2015]

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used
- ResNet architecture can be thought of as large ensemble of relatively shallow networks. [Veit et al. NIPS 2016]

Intro to CNNs Summary

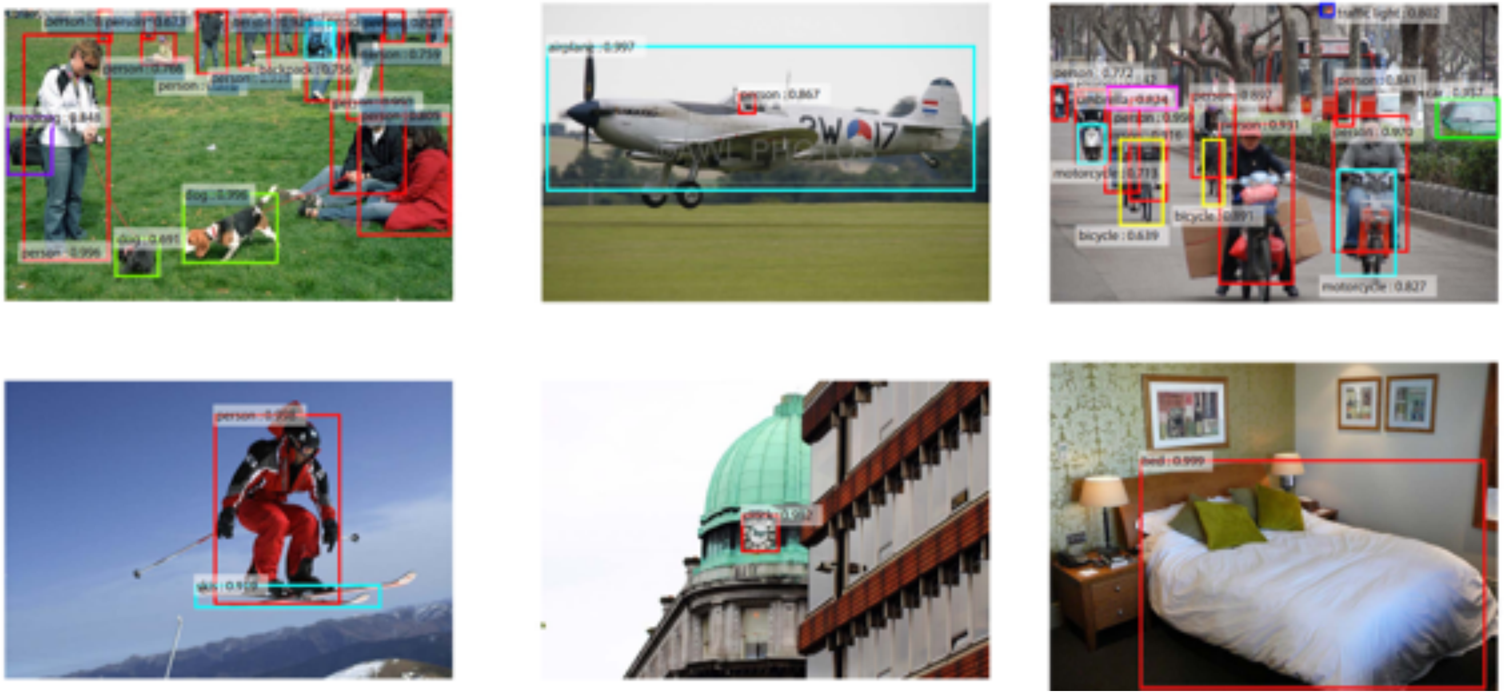
- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like

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

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

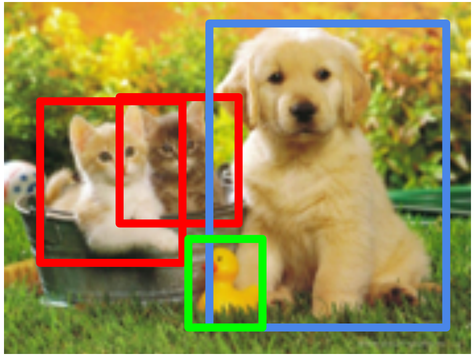
    - but recent advances such as ResNet/GoogLeNet challenge this paradigm

Spatial Localization and Detection



Results from Faster R-CNN, Ren et al 2015

Computer Vision Tasks

Classification	Classification + Localization	Object Detection	Instance Segmentation
			
CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK

Single object

Multiple objects

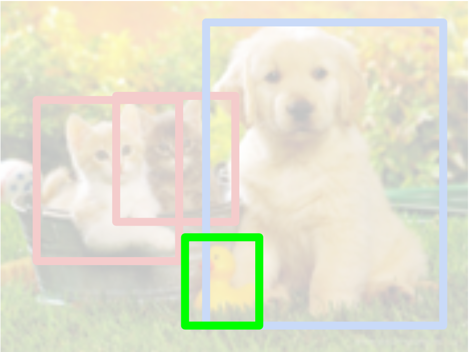
* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Computer Vision Tasks

Classification

Classification + Localization

Object Detection



Instance Segmentation

Classification + Localization: Task

Classification: C classes

Input: Image
Output: Class label
Evaluation metric: Accuracy

CAT

(x, y, w, h)

Localization:

Input: Image
Output: Box in the image (x, y, w, h)
Evaluation metric: Intersection over Union

Classification + Localization: Do both

Localization as Regression

Input: image

Neural Net

Output:
Box coordinates
(4 numbers)

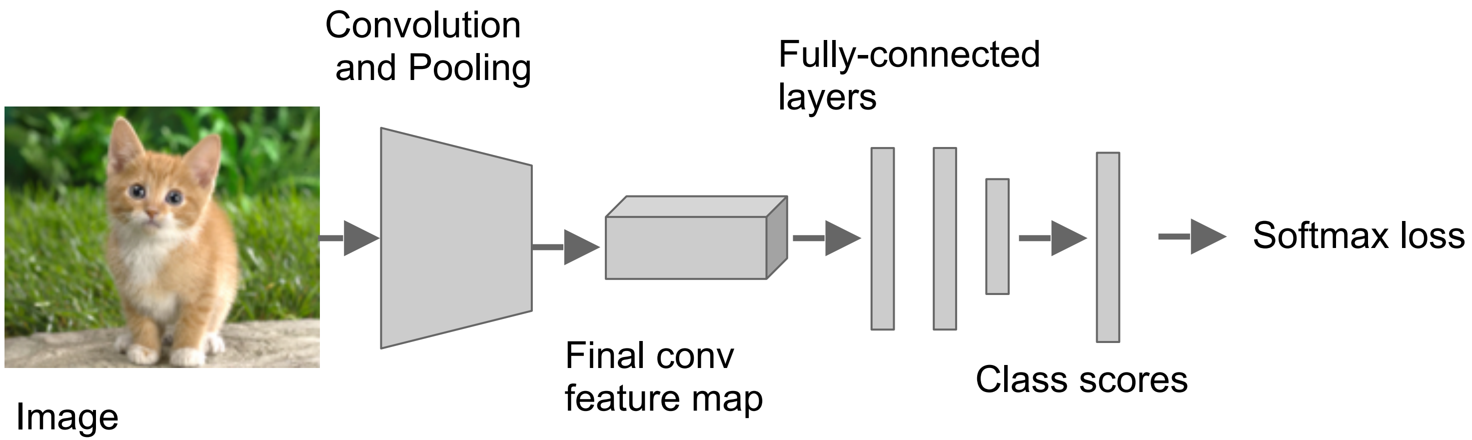
Correct output:
box coordinates
(4 numbers)

Loss:
L2 distance

Only one object,
simpler than detection

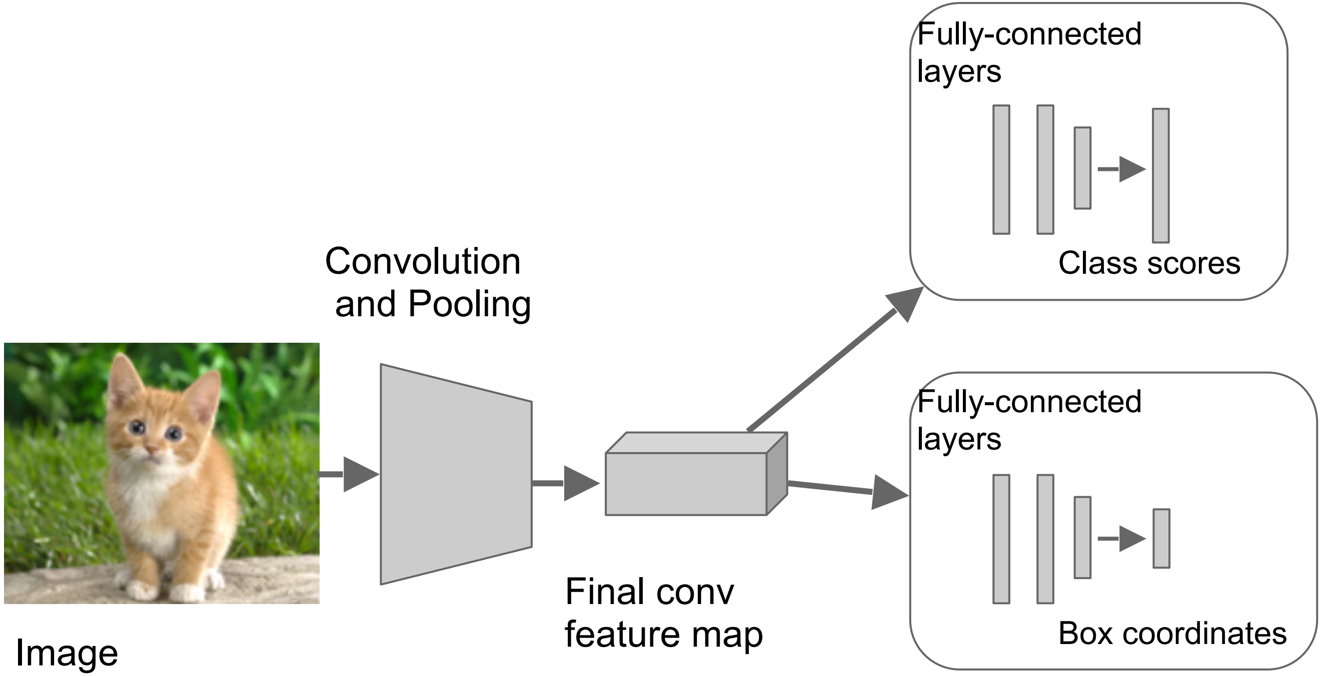
Simple Recipe for Classification + Localization

Step 1: Train (or download) a classification model (AlexNet, VGG, GoogLeNet)



Simple Recipe for Classification + Localization

Step 2: Attach new fully-connected "regression head" to the network

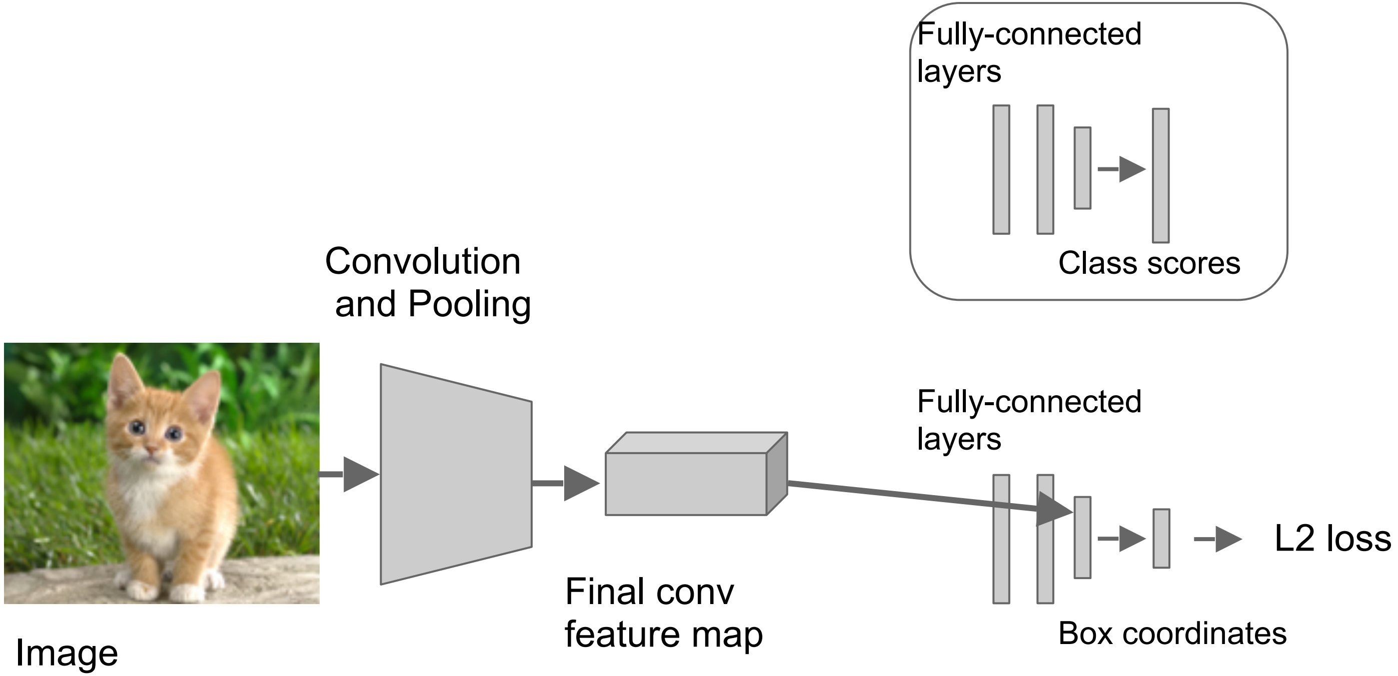


"Classification head"

"Regression head"

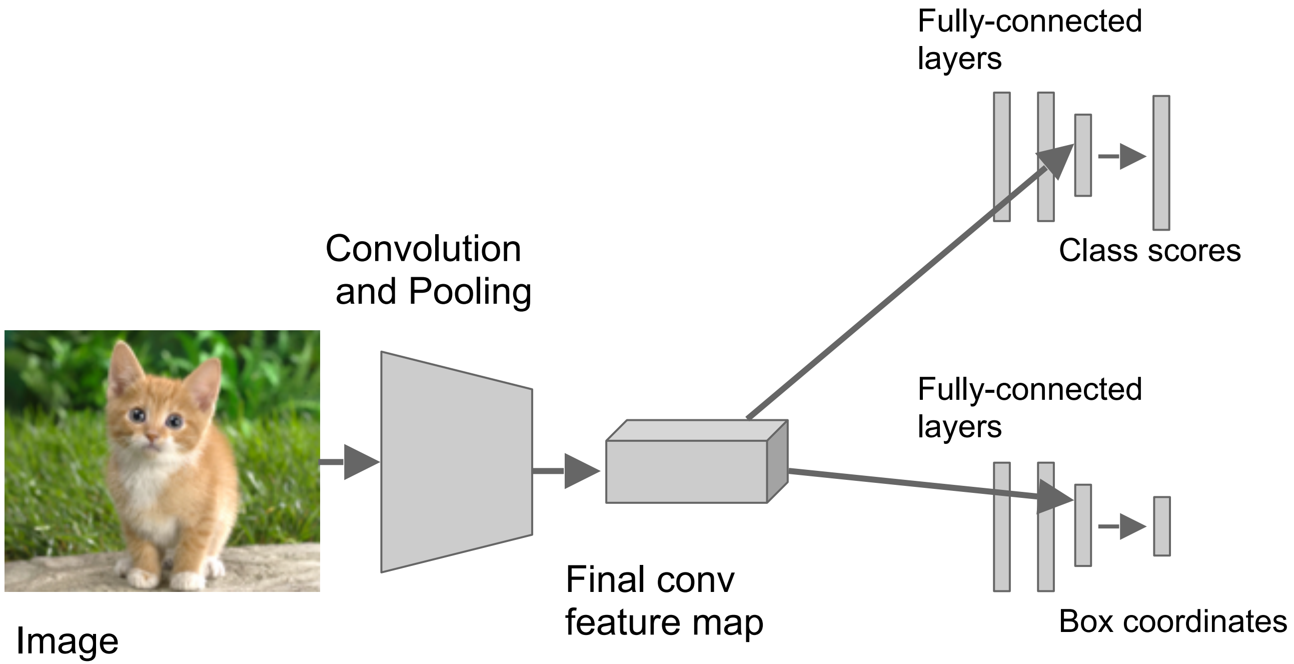
Simple Recipe for Classification + Localization

Step 3: Train the regression head only with SGD and L2 loss



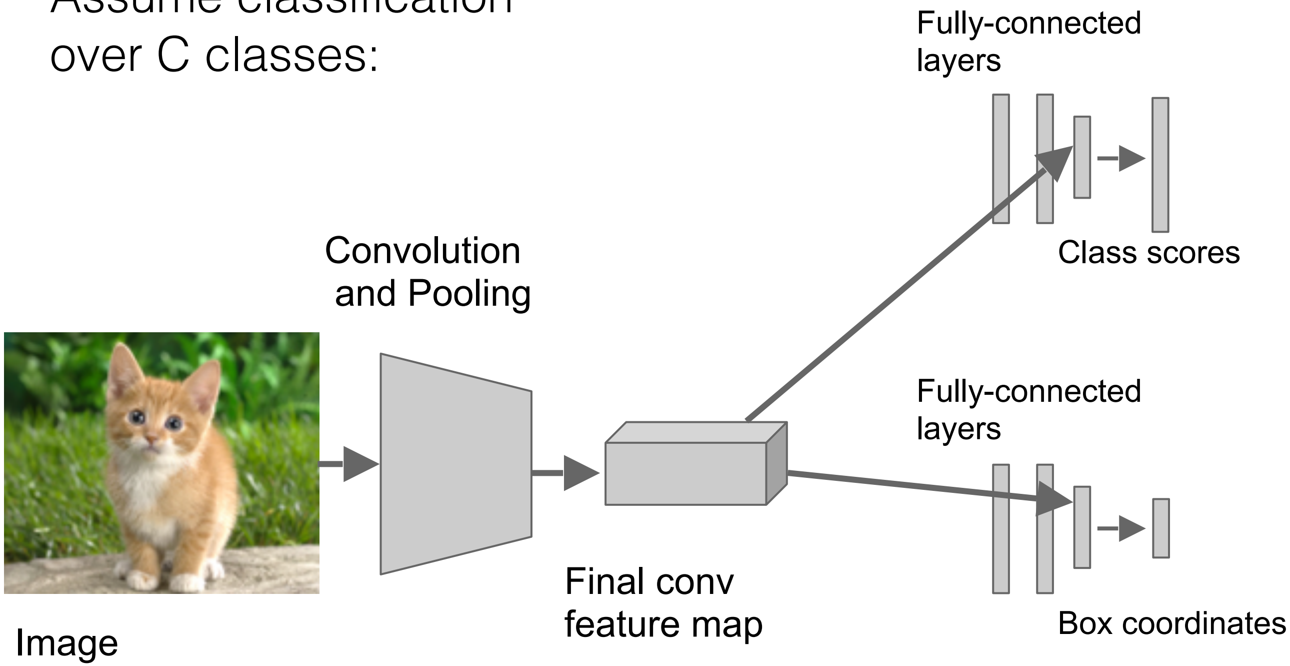
Simple Recipe for Classification + Localization

Step 4: At test time use both heads



Per-class vs class agnostic regression

Assume classification over C classes:



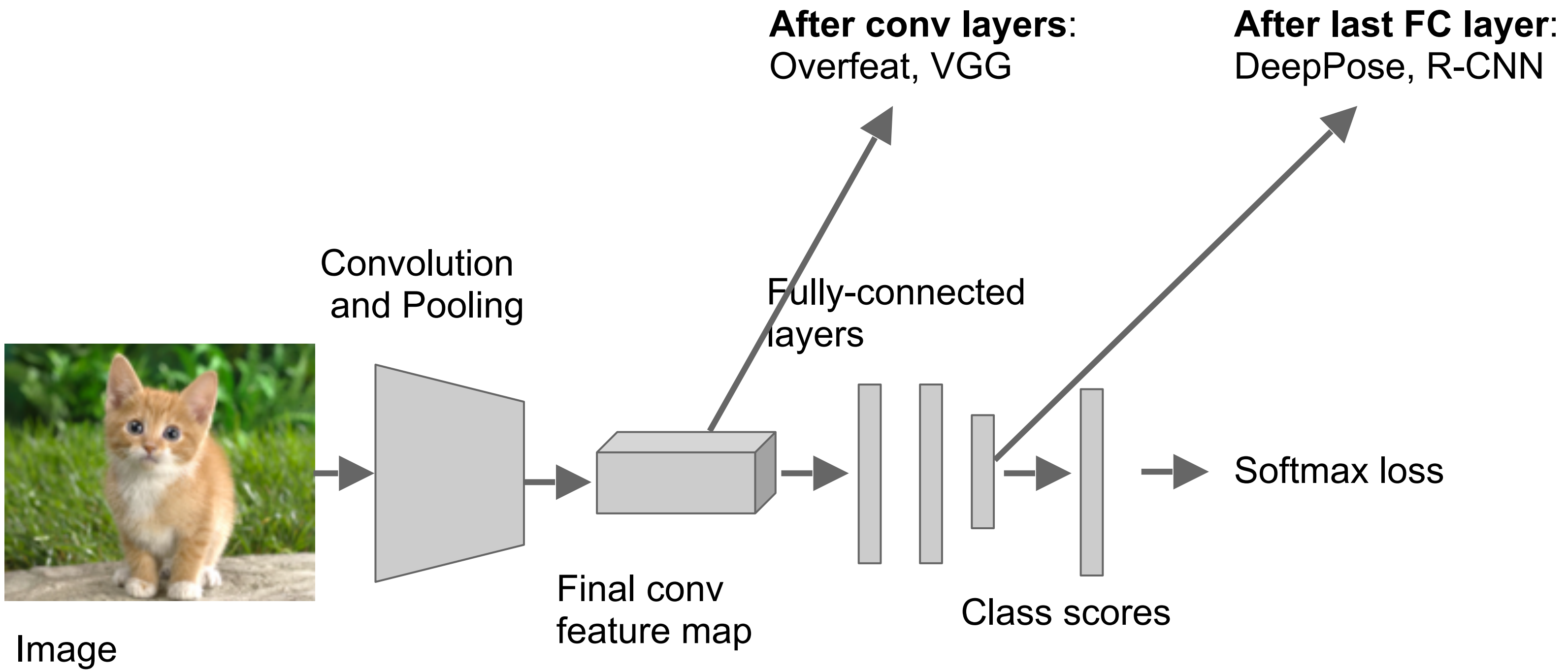
Classification head:
C numbers
(one per class)

Class agnostic:
4 numbers
(one box)

Class specific:
C x 4 numbers
(one box per class)

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

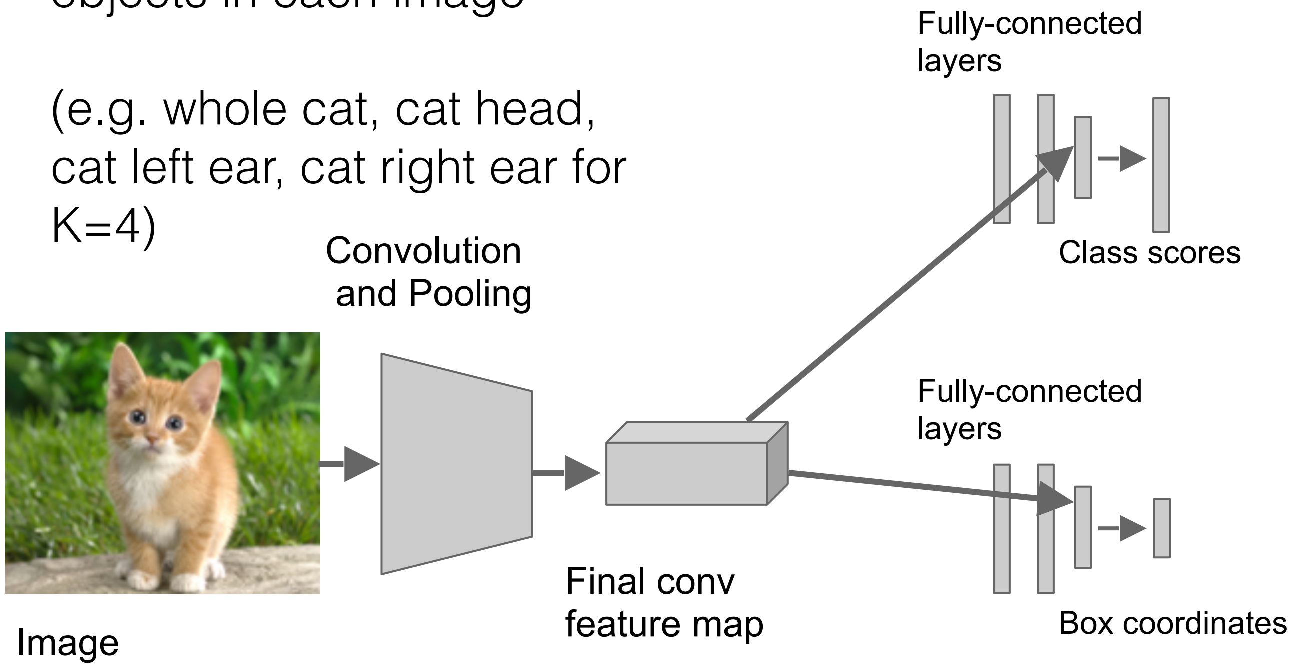
Where to attach the regression head?



Aside: Localizing multiple objects

Want to localize **exactly** K objects in each image

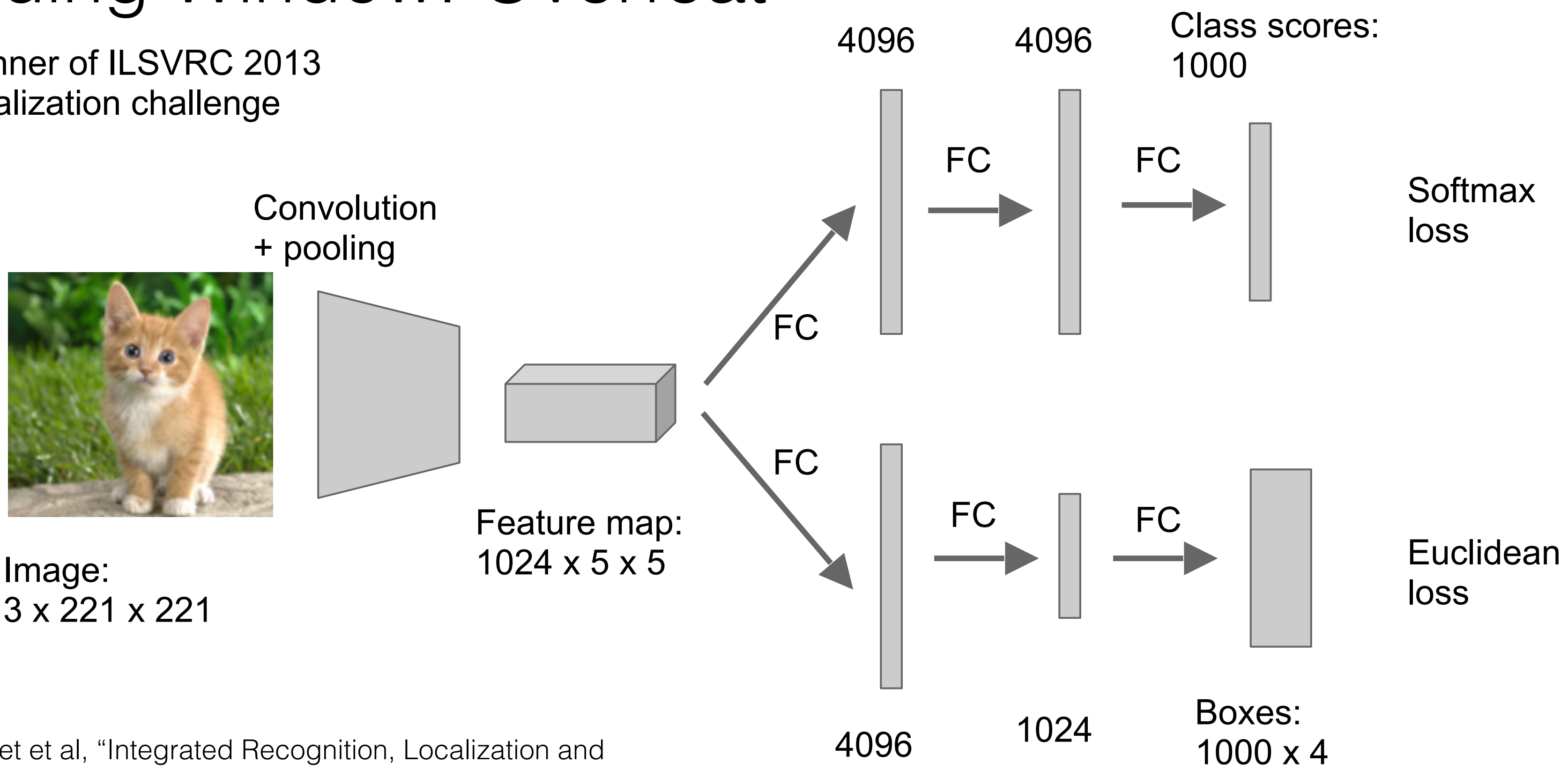
(e.g. whole cat, cat head, cat left ear, cat right ear for K=4)



K x 4 numbers
(one box per object)

Sliding Window: Overfeat

Winner of ILSVRC 2013
localization challenge



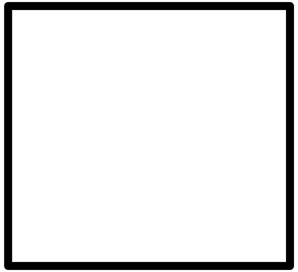
Sermanet et al, “Integrated Recognition, Localization and Detection using Convolutional Networks”, ICLR 2014

Sliding Window: Overfeat

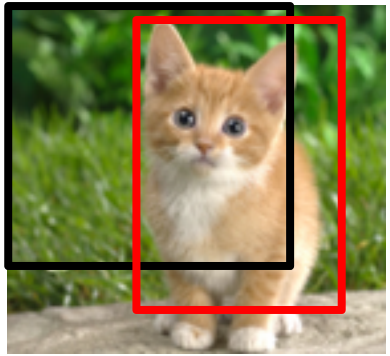
Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257

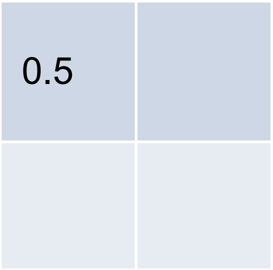
Sliding Window: Overfeat



Network input:
3 x 221 x 221

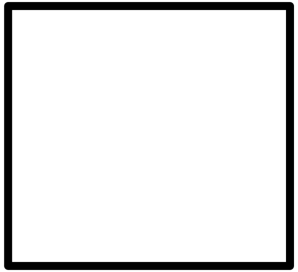


Larger image:
3 x 257 x 257

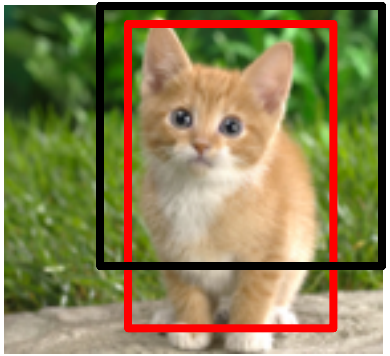


Classification scores:
P(cat)

Sliding Window: Overfeat



Network input:
3 x 221 x 221

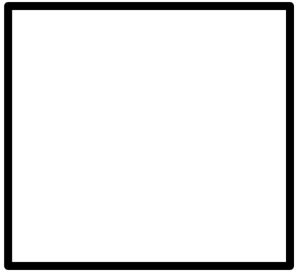


Larger image:
3 x 257 x 257

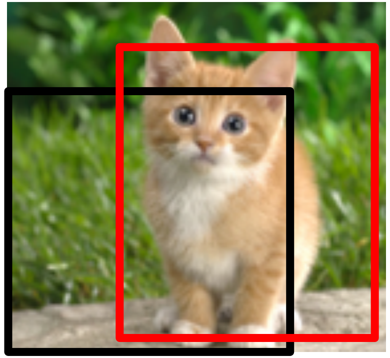
0.5	0.75

Classification scores:
P(cat)

Sliding Window: Overfeat



Network input:
3 x 221 x 221



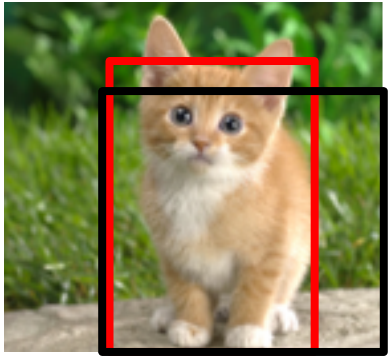
Larger image:
3 x 257 x 257

0.5	0.75
0.6	

Classification scores:
P(cat)

Sliding Window: Overfeat

Network input:
3 x 221 x 221



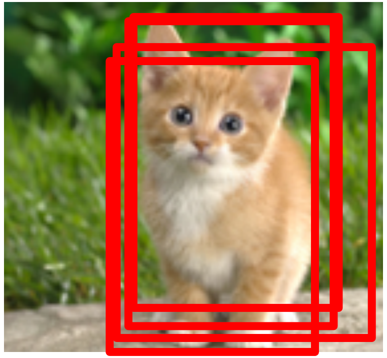
Larger image:
3 x 257 x 257

0.5	0.75
0.6	0.8

Classification scores:
P(cat)

Sliding Window: Overfeat

Network input:
3 x 221 x 221



Larger image:
3 x 257 x 257

0.5	0.75
0.6	0.8

Classification scores:
P(cat)

Sliding Window: Overfeat

Greedily merge boxes and scores (details in paper)

Network input:
3 x 221 x 221

Larger image:
3 x 257 x 257

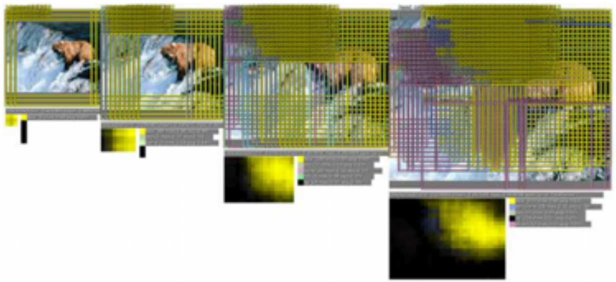
0.8

Classification score:
P(cat)

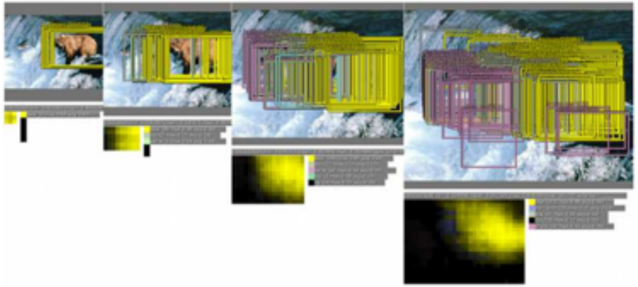
Sliding Window: Overfeat

In practice use many sliding window locations and multiple scales

Window positions + score maps



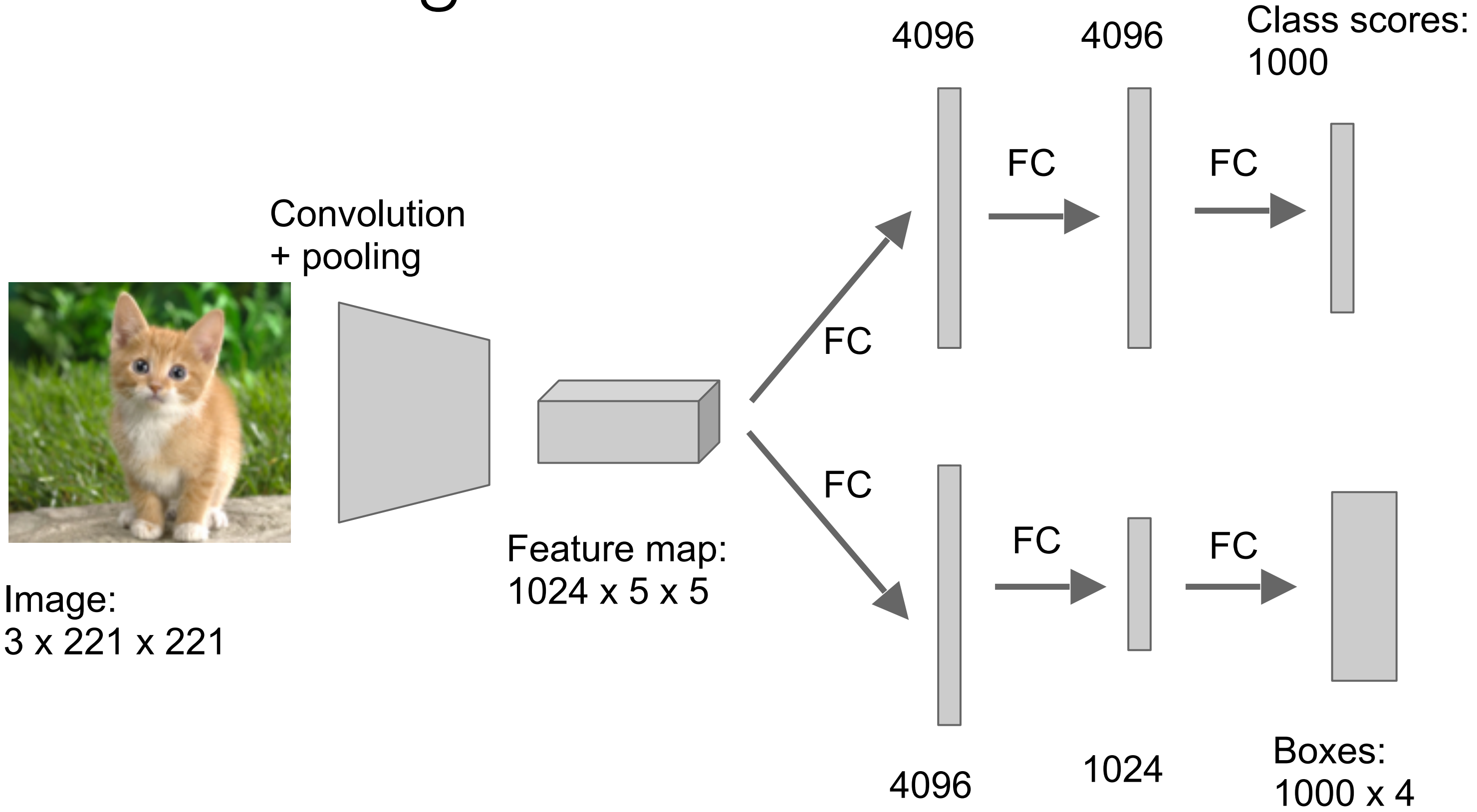
Box regression outputs



Final Predictions

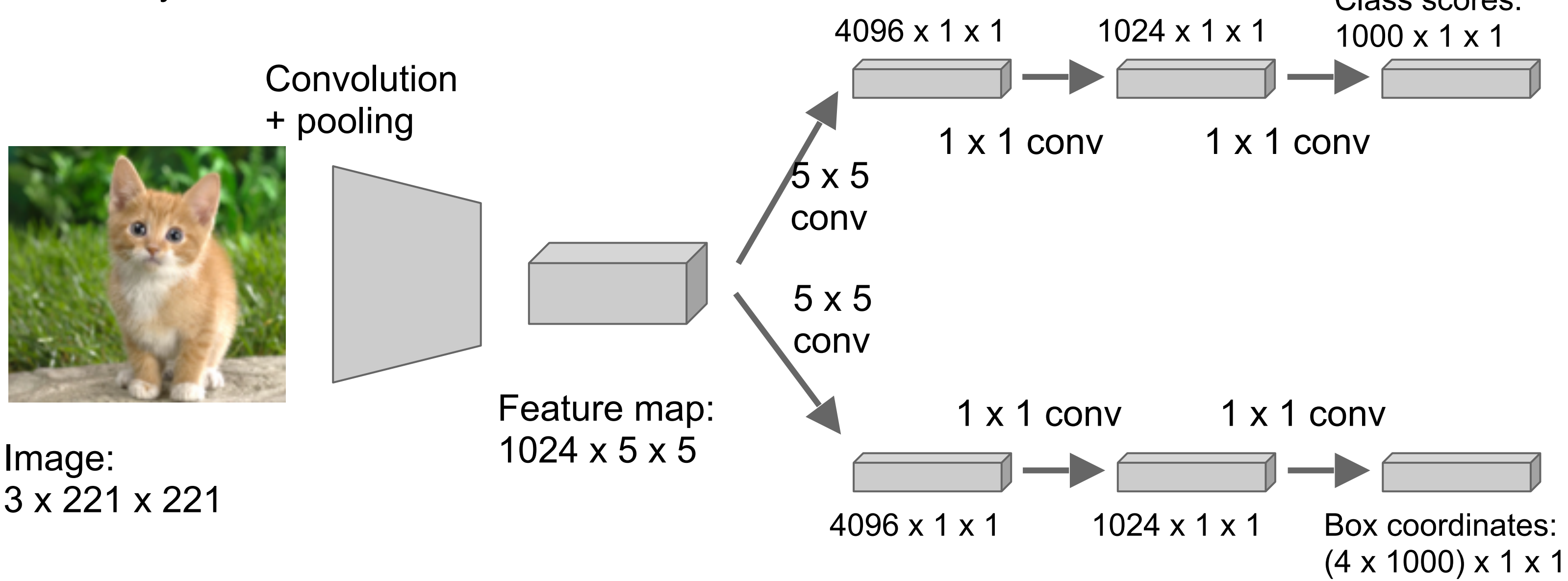
Sermanet et al, “Integrated Recognition, Localization and Detection using Convolutional Networks”, ICLR 2014

Efficient Sliding Window: Overfeat



Efficient Sliding Window: Overfeat

Efficient sliding window by converting fully-connected layers into convolutions



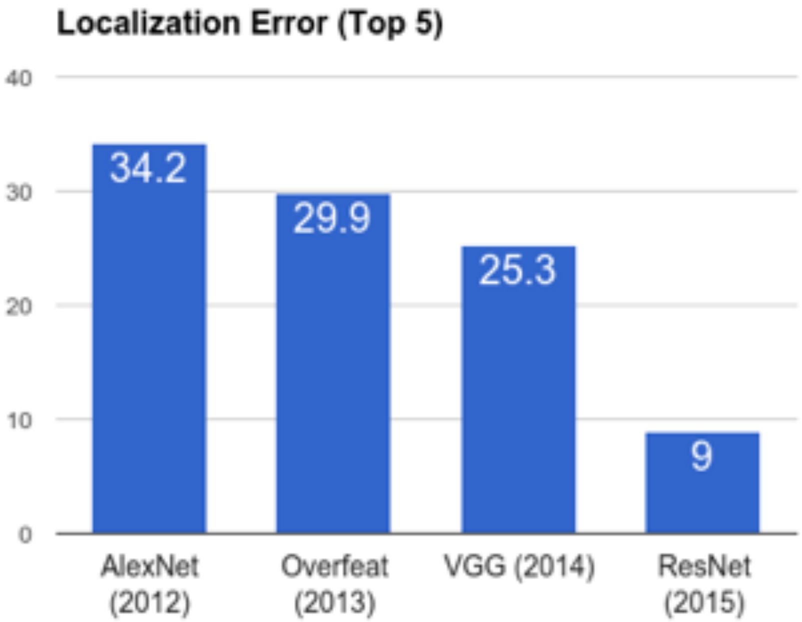
Summary: Sliding Window

- Run classification + regression network at multiple locations on a high-resolution image

- Convert fully-connected layers into convolutional layers for efficient computation

- Combine classifier and
  regressor predictions across all scales for final prediction

ImageNet Classification + Localization (1 object per image)



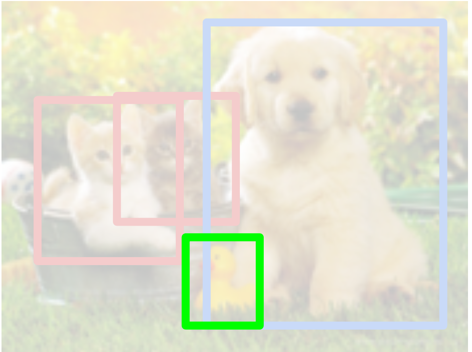
AlexNet: Localization method not published

Overfeat: Multiscale convolutional regression with box merging

VGG: Same as Overfeat, but fewer scales and locations; simpler method, gains all due to deeper features

ResNet: Different localization method (RPN) and much deeper features

Computer Vision Tasks

Classification	**Classification + Localization**	Object Detection	Instance Segmentation
			

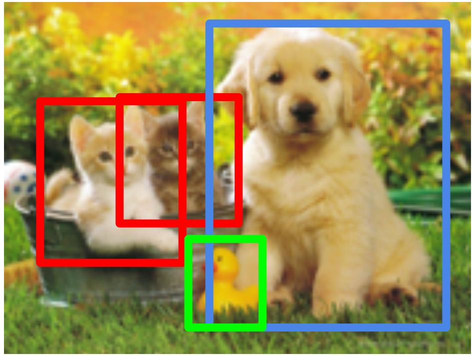
* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Computer Vision Tasks

Classification

Classification + Localization

Object Detection



Instance Segmentation

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Detection Metrics - COCO Challenge

Average Precision (AP):

AP	% AP at IoU=.50:.05:.95 **(determines challenge winner)**
$AP^{IoU=.50}$	% AP at IoU=.50 (PASCAL VOC metric)
$AP^{IoU=.75}$	% AP at IoU=.75 (strict metric)

AP Across Scales:

AP^{small}	% AP for small objects: area < 32^2
AP^{medium}	% AP for medium objects: 32^2 < area < 96^2
AP^{large}	% AP for large objects: area > 96^2

Average Recall (AR):

$AR^{max=1}$	% AR given 1 detection per image
$AR^{max=10}$	% AR given 10 detections per image
$AR^{max=100}$	% AR given 100 detections per image

AR Across Scales:

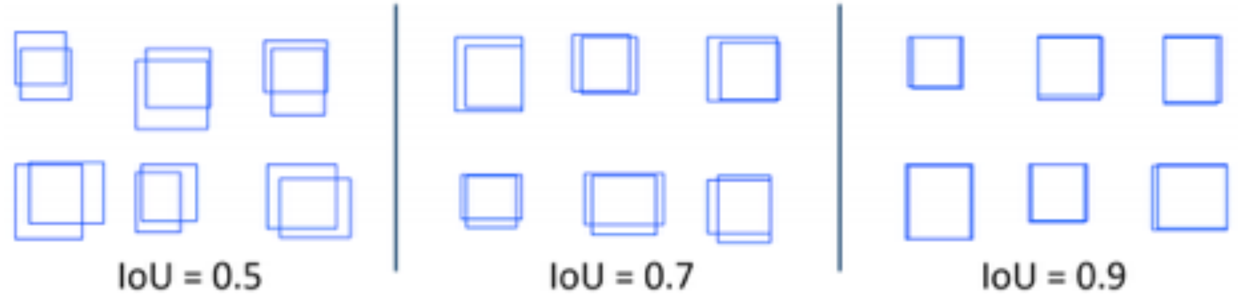
AR^{small}	% AR for small objects: area < 32^2
AR^{medium}	% AR for medium objects: 32^2 < area < 96^2
AR^{large}	% AR for large objects: area > 96^2

Detection Metrics

```
Average Precision (AP):
  AP              % AP at IoU=.50:.05:.95 (determines challenge winner)
  AP^{IoU=.50}    % AP at IoU=.50 (PASCAL VOC metric)
  AP^{IoU=.75}    % AP at IoU=.75 (strict metric)
```

Challenges Score: AP

- AP is averaged over multiple IoU values between 0.5 and 0.95.
- More comprehensive metric than the traditional AP at a fixed IoU value (0.5 for PASCAL).



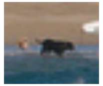
Detection Metrics

```
AP Across Scales:
   AP^small        % AP for small objects: area < 32²
   AP^medium       % AP for medium objects: 32² < area < 96²
   AP^large        % AP for large objects: area > 96²
```

Other Scores: Size AP

- AP is averaged over instance size:
  - small (A < 32 x 32)
  - medium (32x 32 < A < 96 x 96)
  - large (A > 96 x 96)

A < 32x32



32x32 < A < 96x96

A > 96x96

Detection Metrics

```
Average Recall (AR):
   AR^max=1                    % AR given 1 detection per image
   AR^max=10                   % AR given 10 detections per image
   AR^max=100                  % AR given 100 detections per image
AR Across Scales:
   AR^small                    % AR for small objects: area < 32^2
   AR^medium                   % AR for medium objects: 32^2 < area < 96^2
   AR^large                    % AR for large objects: area > 96^2
```

Other Scores: AR

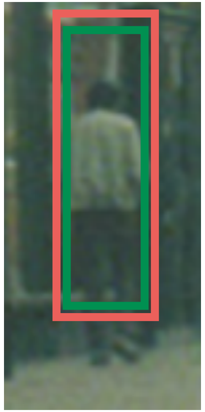
- Measures the maximum recall over a fixed number of detections allowed in the image of 1, 10, 100.
- AR is averaged over small (A < 32 x 32), medium (32x 32 < A < 96 x 96) and large (A > 96 x 96) instances of objects.

Detection Ambiguity

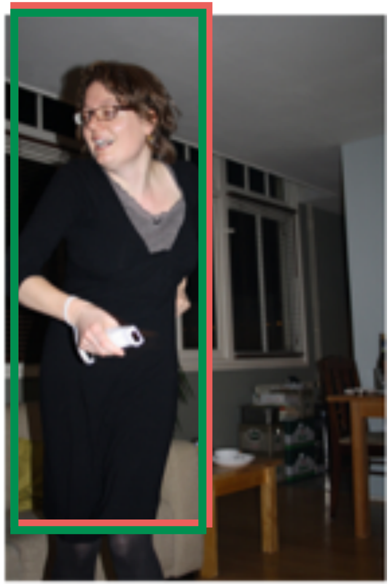
Which one is better?

IoU = 0.5

IoU = 0.7



IoU = 0.95



Ground-Truth BBox Detection BBox

Detection as Regression?

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

= 16 numbers

Detection as Regression?

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

= 8 numbers

Detection as Regression?

CAT, (x, y, w, h)
CAT, (x, y, w, h)
....
CAT (x, y, w, h)

= many numbers

Need variable sized outputs

Detection as Classification

CAT? NO

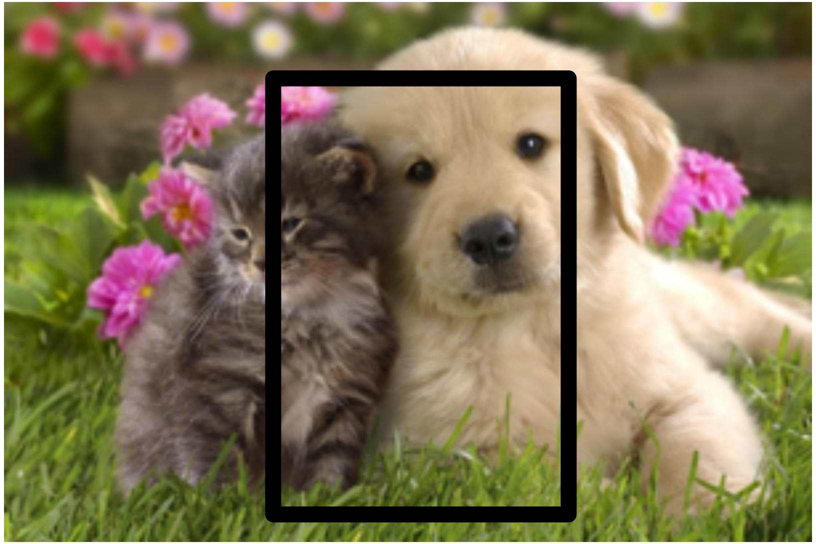
DOG? NO

Detection as Classification

CAT? YES!

DOG? NO

Detection as Classification



CAT? NO

DOG? NO

Detection as Classification

Problem: Need to test many positions and scales

Solution: If your classifier is fast enough, just do it

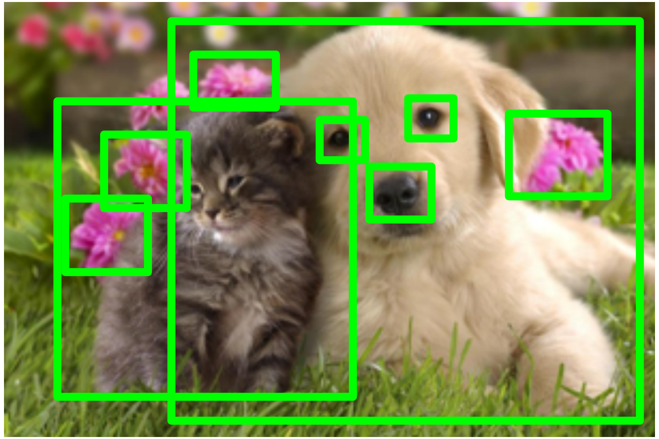
Detection as Classification

Problem: Need to test many positions and scales, and use a computationally demanding classifier (CNN)

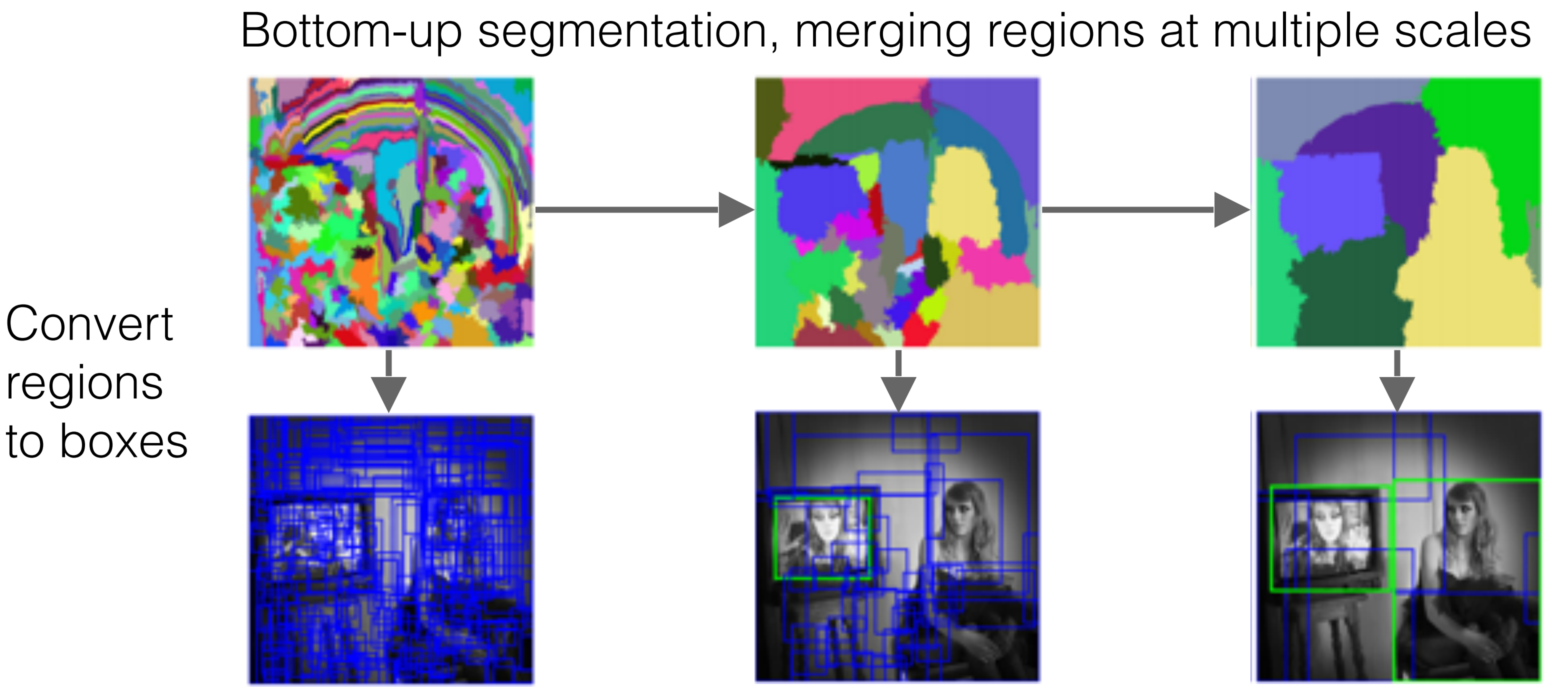
Solution: Only look at a tiny subset of possible positions

Region Proposals

- Find “blobby” image regions that are likely to contain objects
- “Class-agnostic” object detector
- Look for “blob-like” regions



Region Proposals: Selective Search



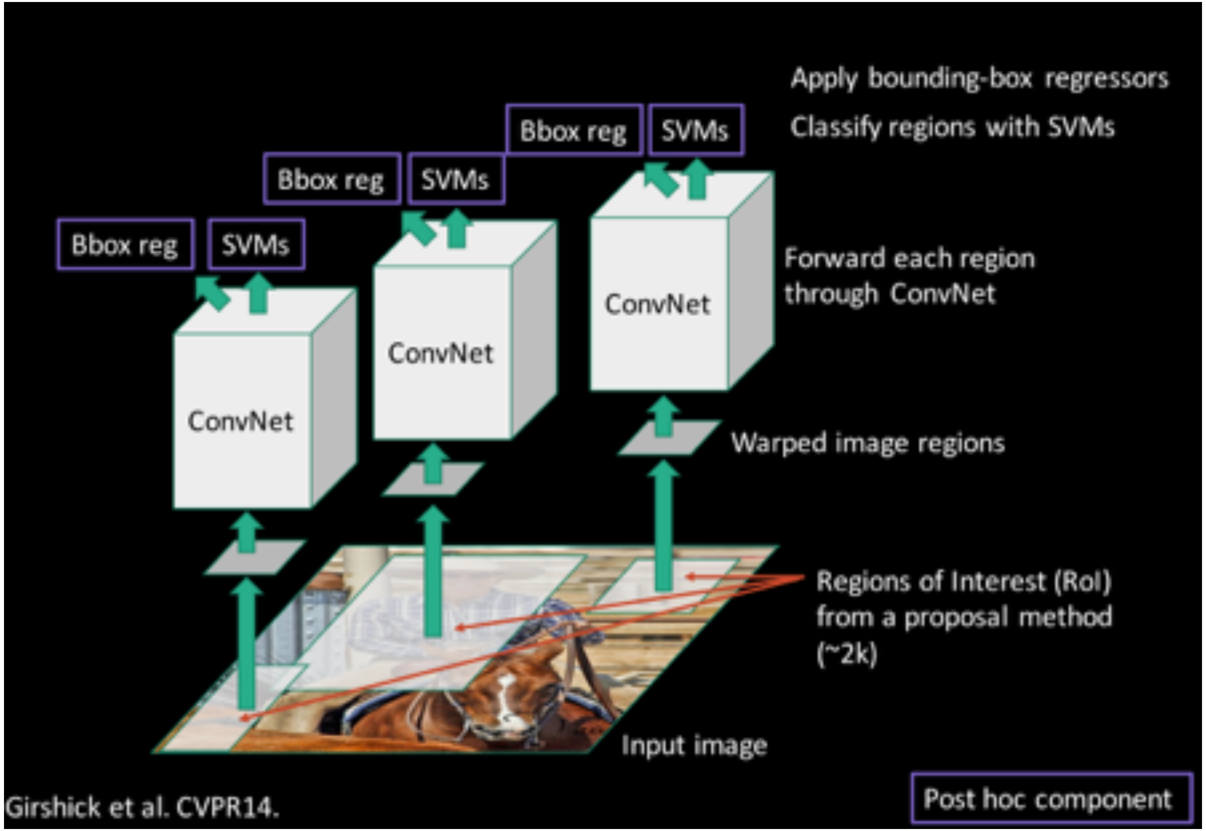
Uijlings et al, "Selective Search for Object Recognition", IJCV 2013

Region Proposals: Many other choices

Method	Approach	Outputs Segments	Outputs Score	Control #proposals	Time (sec.)	Repea-tability	Recall Results	Detection Results
Bing [18]	Window scoring		✓	✓	0.2	★★★	★	·
CPMC [19]	Grouping	✓	✓	✓	250	-	★★	★
EdgeBoxes [20]	Window scoring		✓	✓	0.3	★★	★★★	★★★
Endres [21]	Grouping	✓	✓	✓	100	-	★★★	★★
Geodesic [22]	Grouping	✓		✓	1	★	★★★	★★
MCG [23]	Grouping	✓	✓	✓	30	★	★★★	★★★
Objectness [24]	Window scoring		✓	✓	3	·	★	·
Rahtu [25]	Window scoring		✓	✓	3	·	·	★
RandomizedPrim's [26]	Grouping	✓		✓	1	★	★	★★
Rantalankila [27]	Grouping	✓		✓	10	★★	·	★★
Rigor [28]	Grouping	✓		✓	10	★	★★	★★
SelectiveSearch [29]	Grouping	✓	✓	✓	10	★★	★★★	★★★
Gaussian				✓	0	·	·	★
SlidingWindow				✓	0	★★★	·	·
Superpixels		✓			1	★	·	·
Uniform				✓	0	·	·	·

Hosang et al, “What makes for effective detection proposals?”, PAMI 2015

Putting it together: R-CNN

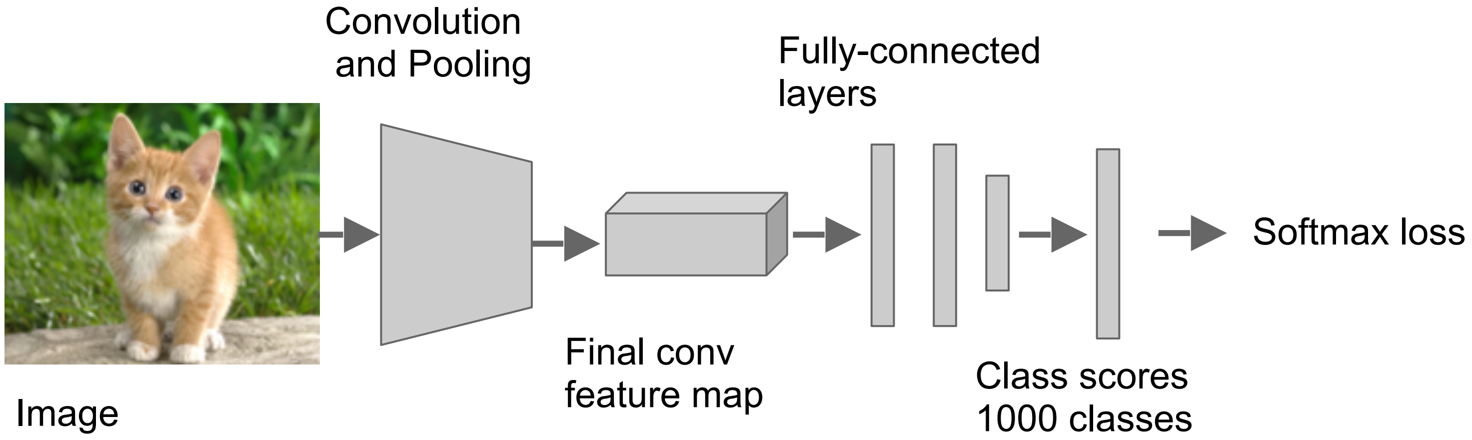


Girschick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014

Slide credit: Ross Girschick

R-CNN Training

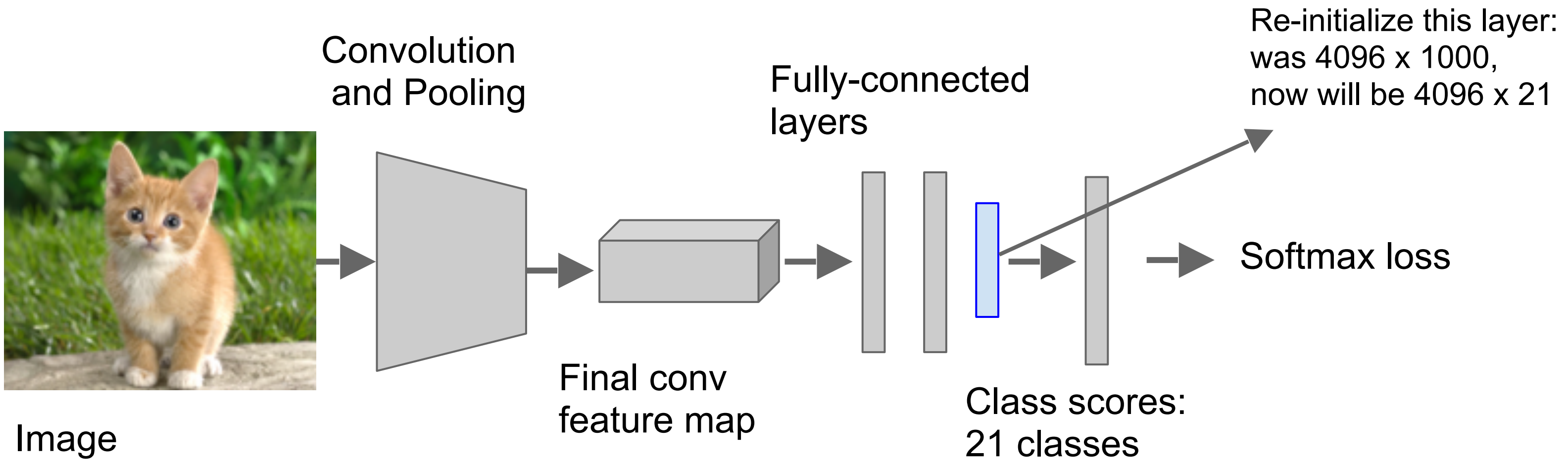
Step 1: Train (or download) a classification model for ImageNet (AlexNet)



R-CNN Training

Step 2: Fine-tune model for detection

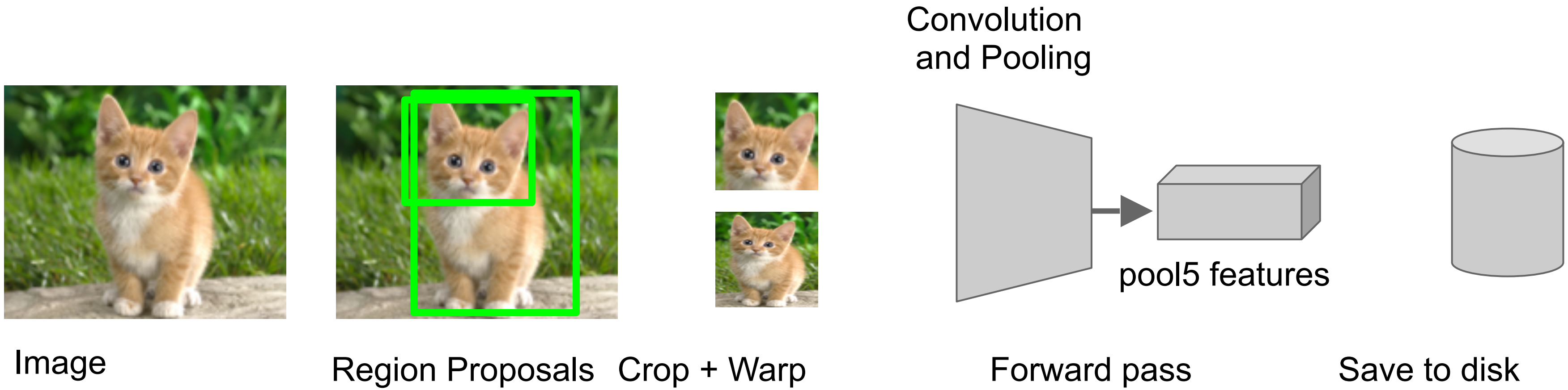
- Instead of 1000 ImageNet classes, want PASCAL 20 object classes + *background*
- Throw away final fully-connected layer, reinitialize from scratch
- Keep training model using positive / negative regions from detection images



R-CNN Training

Step 3: Extract features

- Extract region proposals for all images
- For each region: warp to CNN input size, run forward through CNN, save pool5 features to disk
- Have a big hard drive: features are ~200GB for PASCAL dataset!

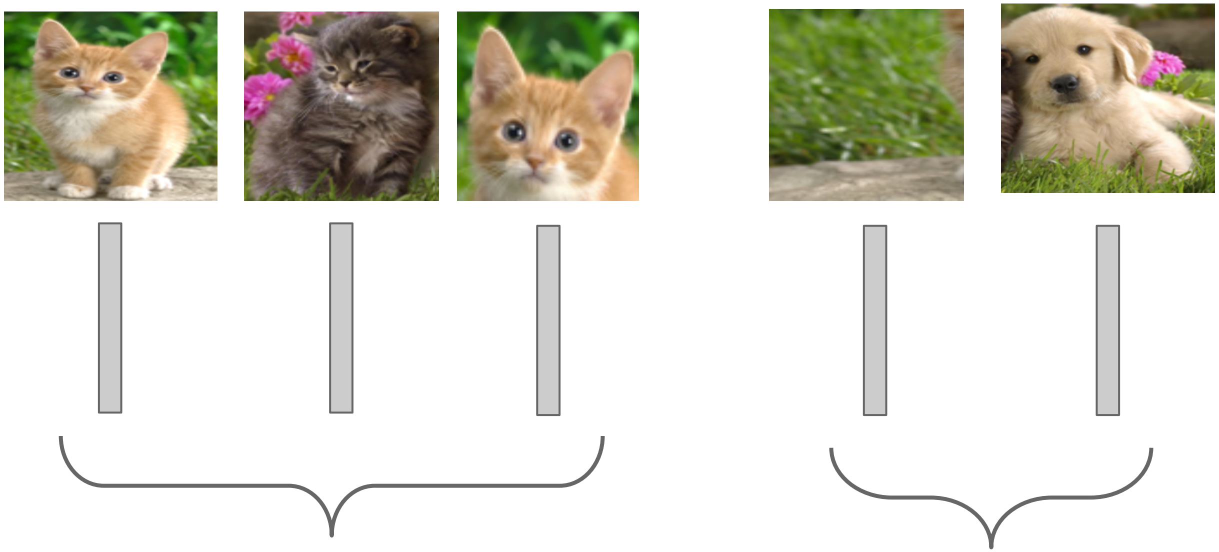


R-CNN Training

Step 4: Train one binary SVM per class to classify region features

Training image regions

Cached region features



Positive samples for cat SVM

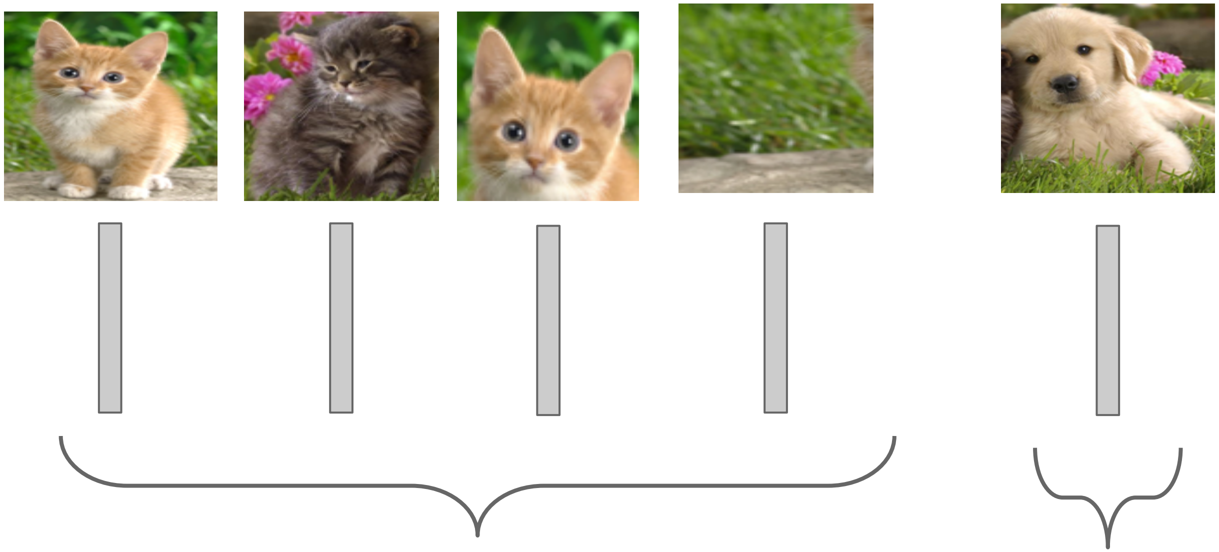
Negative samples for cat SVM

R-CNN Training

Step 4: Train one binary SVM per class to classify region features

Training image regions

Cached region features



Negative samples for dog SVM

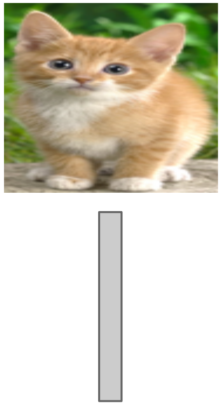
Positive samples for dog SVM

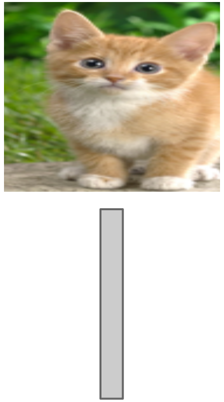
R-CNN Training

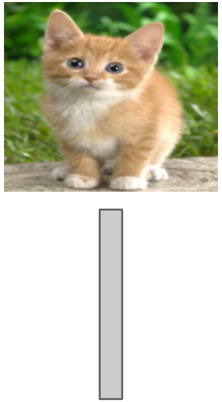
Step 5 (bbox regression): For each class, train a linear regression model to map from cached features to offsets to GT boxes to make up for “slightly wrong” proposals

Training image regions

Cached region features







Regression targets
(dx, dy, dw, dh)
Normalized coordinates

(0, 0, 0, 0)
Proposal is good

(.25, 0, 0, 0)
Proposal too far to left

(0, 0, -0.125, 0)
Proposal too wide

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

Object Detection: Datasets

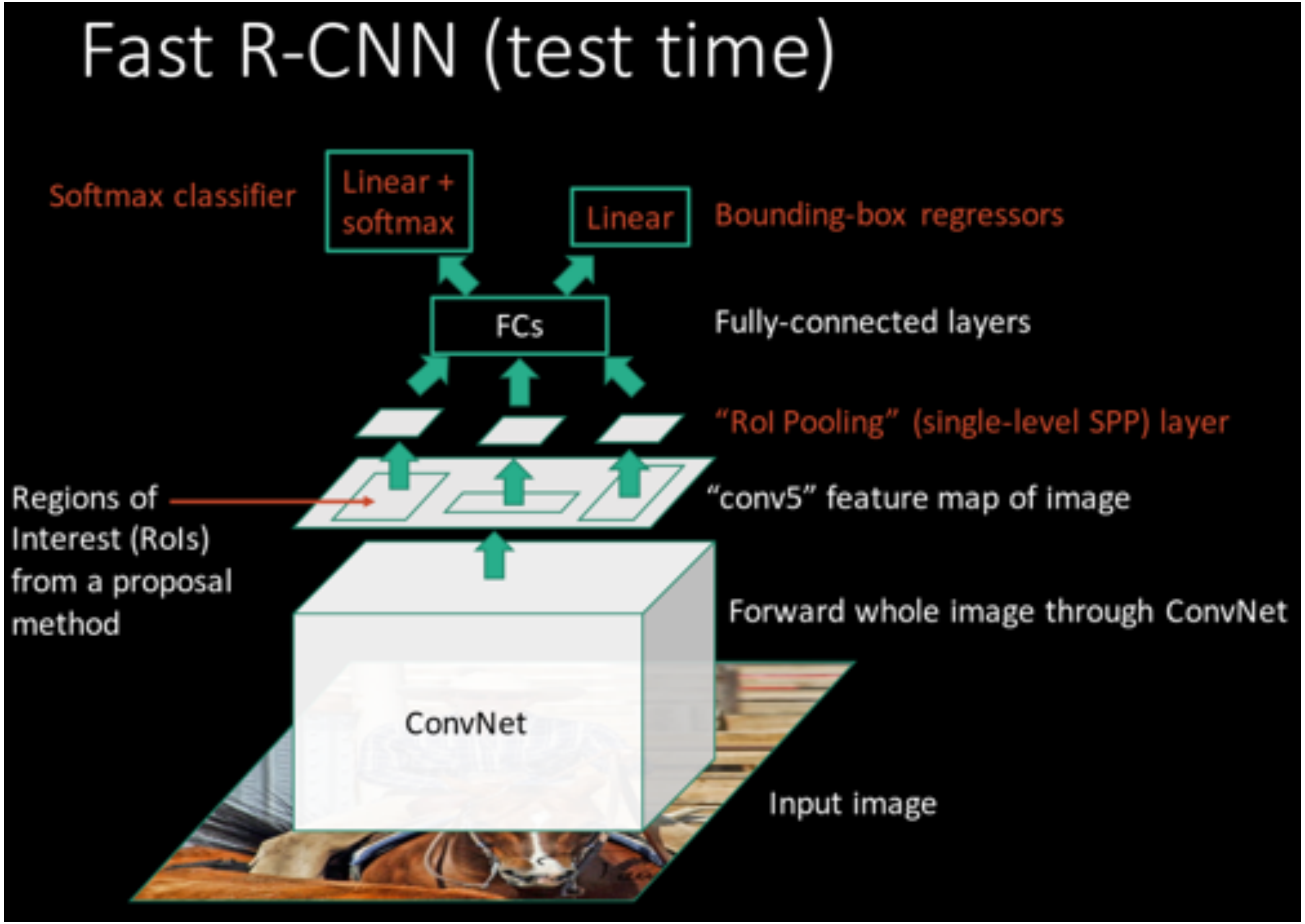
	PASCAL VOC (2010)	ImageNet Detection (ILSVRC 2014)	COCO (2014)
Number of classes	20	**200**	80
Number of images (train + val)	~20k	**~470k**	~120k
Mean objects per image	2.4	1.1	**7.2**

R-CNN Problems

1. Slow at test-time: need to run full forward pass of CNN for each region proposal

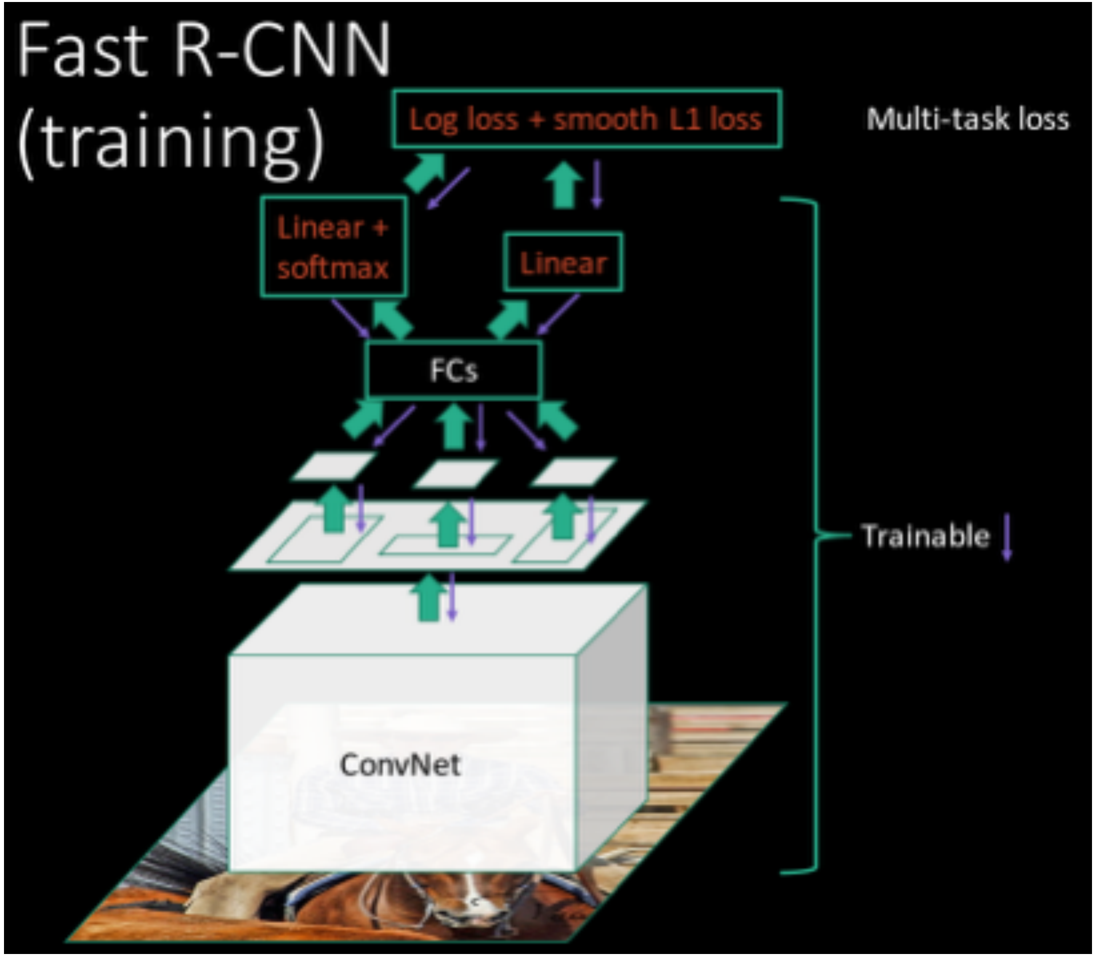
2. SVMs and regressors are post-hoc: CNN features not updated in response to SVMs and regressors

3. Complex multistage training pipeline



R-CNN Problem #1:
Slow at test-time due to independent forward passes of the CNN

Solution:
Share computation of convolutional layers between proposals for an image



R-CNN Problem #2:
Post-hoc training: CNN not updated in response to final classifiers and regressors

R-CNN Problem #3:
Complex training pipeline

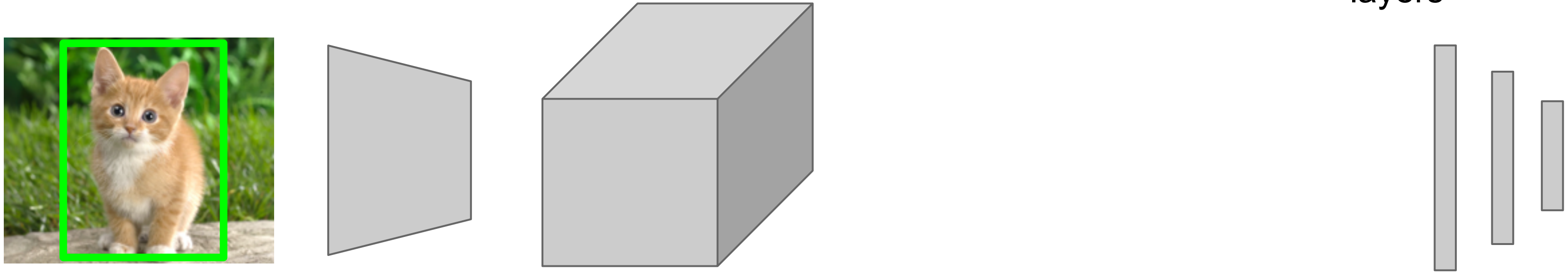
Solution:
Just train the whole system end-to-end all at once!

Slide credit: Ross Girschick

Fast R-CNN: Region of Interest Pooling

Convolution and Pooling

Fully-connected layers



Hi-res input image:
3 x 800 x 600
with region proposal

Hi-res conv features:
C x H x W
with region proposal

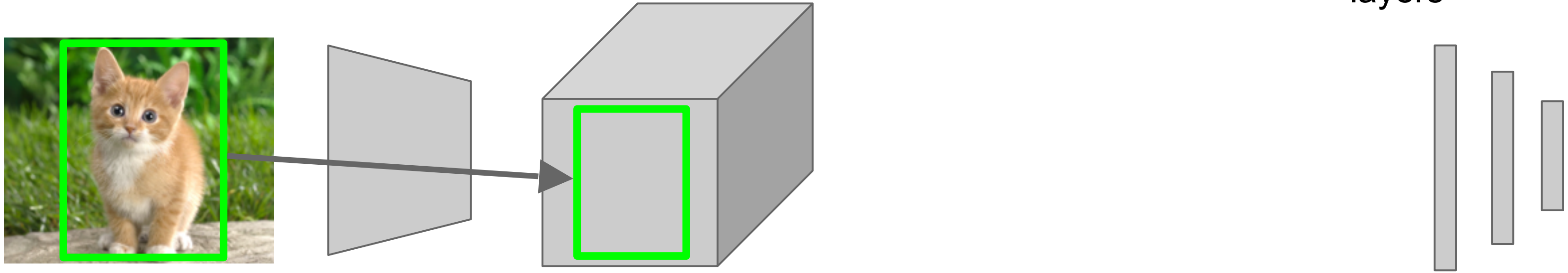
Problem: Fully-connected layers expect low-res conv features: C x h x w

Fast R-CNN: Region of Interest Pooling

Convolution
and Pooling

Project region proposal
onto conv feature map

Fully-connected
layers



Hi-res input image:
3 x 800 x 600
with region
proposal

Hi-res conv features:
C x H x W
with region proposal

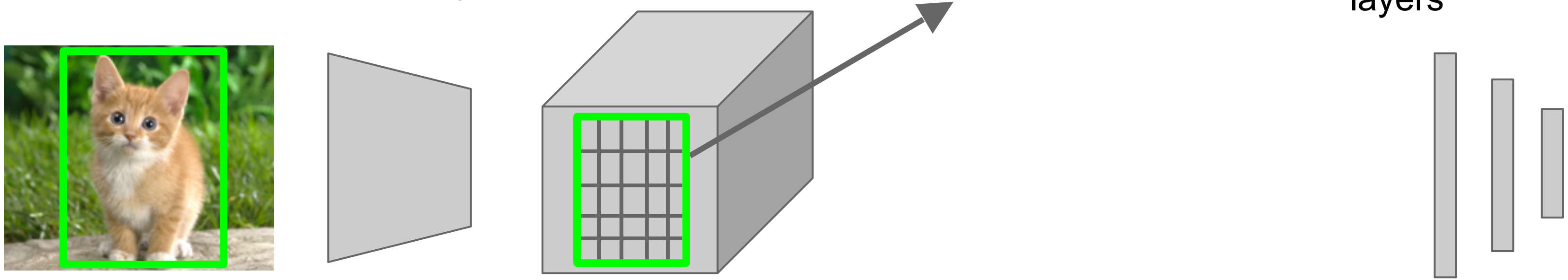
Problem: Fully-connected layers expect low-res conv features: C x h x w

Fast R-CNN: Region of Interest Pooling

Convolution and Pooling

Divide projected region into h x w grid

Fully-connected layers

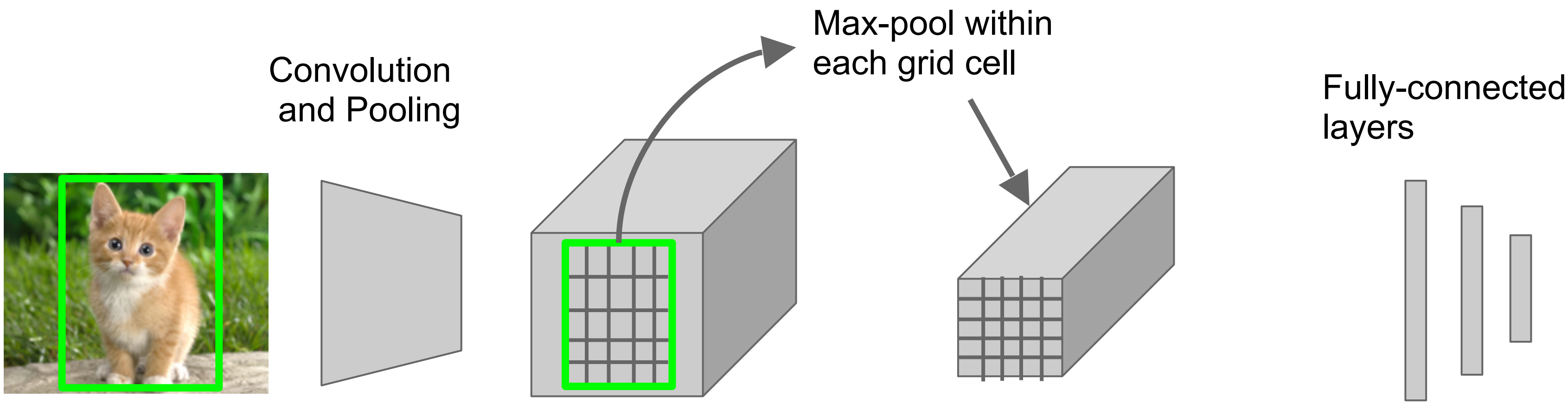


Hi-res input image: 3 x 800 x 600 with region proposal

Hi-res conv features: C x H x W with region proposal

Problem: Fully-connected layers expect low-res conv features: C x h x w

Fast R-CNN: Region of Interest Pooling



Hi-res input image:
3 x 800 x 600
with region
proposal

Hi-res conv features:
C x H x W
with region proposal

RoI conv features:
C x h x w
for region proposal

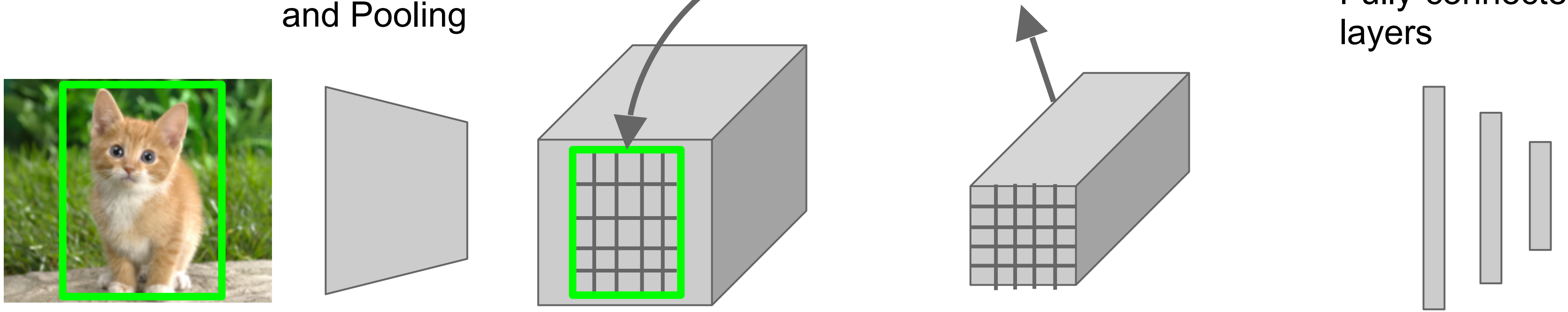
Fully-connected layers expect
low-res conv features:
C x h x w

Fast R-CNN: Region of Interest Pooling

Can back propagate
similar to max pooling

Convolution
and Pooling

Fully-connected
layers



Hi-res input image:
3 x 800 x 600
with region
proposal

Hi-res conv features:
C x H x W
with region proposal

RoI conv features:
C x h x w
for region proposal

Fully-connected layers expect
low-res conv features:
C x h x w

Fast R-CNN Results

Faster!

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**

Using VGG-16 CNN on Pascal VOC 2007 dataset

Fast R-CNN Results

		R-CNN	Fast R-CNN
Faster!	Training Time:	84 hours	**9.5 hours**
	(Speedup)	1x	**8.8x**
FASTER!	Test time per image	47 seconds	**0.32 seconds**
	(Speedup)	1x	**146x**

Using VGG-16 CNN on Pascal VOC 2007 dataset

Fast R-CNN Results

		R-CNN	Fast R-CNN
Faster!	Training Time:	84 hours	**9.5 hours**
	(Speedup)	1x	**8.8x**
FASTER!	Test time per image	47 seconds	**0.32 seconds**
	(Speedup)	1x	**146x**
Better!	mAP (VOC 2007)	66.0	**66.9**

Using VGG-16 CNN on Pascal VOC 2007 dataset

Fast R-CNN Problem:

Test-time speeds don't include region proposals

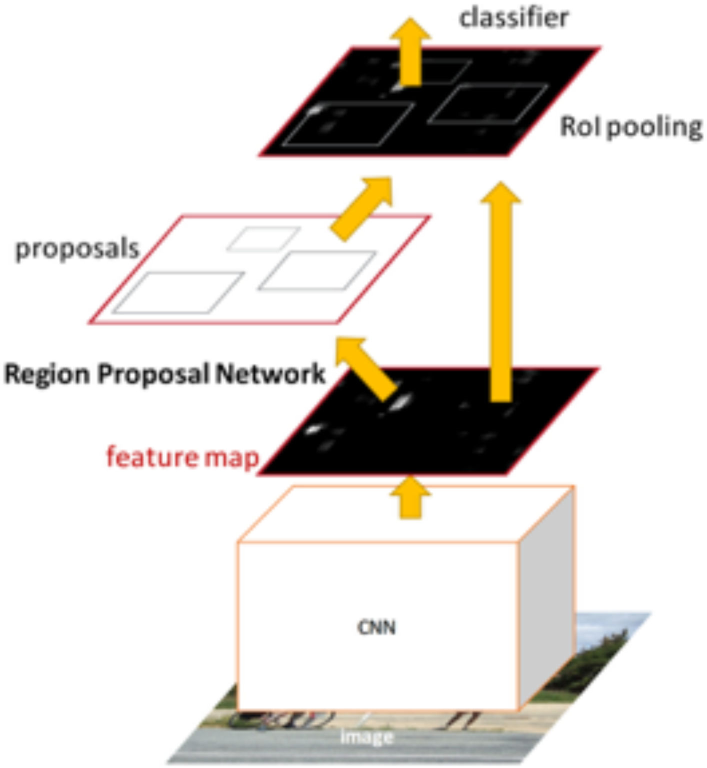
	R-CNN	**Fast R-CNN**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
Test time per image with Selective Search	50 seconds	**2 seconds**
(Speedup)	1x	**25x**

Fast R-CNN ~~Problem~~ Solution:

Test-time speeds don't include region proposals
Just make the CNN do region proposals too!

	R-CNN	Fast R-CNN
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
Test time per image with Selective Search	50 seconds	**2 seconds**
(Speedup)	1x	**25x**

Faster R-CNN:



Insert a **Region Proposal Network (RPN)** after the last convolutional layer

RPN trained to produce region proposals directly; no need for external region proposals!

After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN

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

Slide credit: Ross Girschick

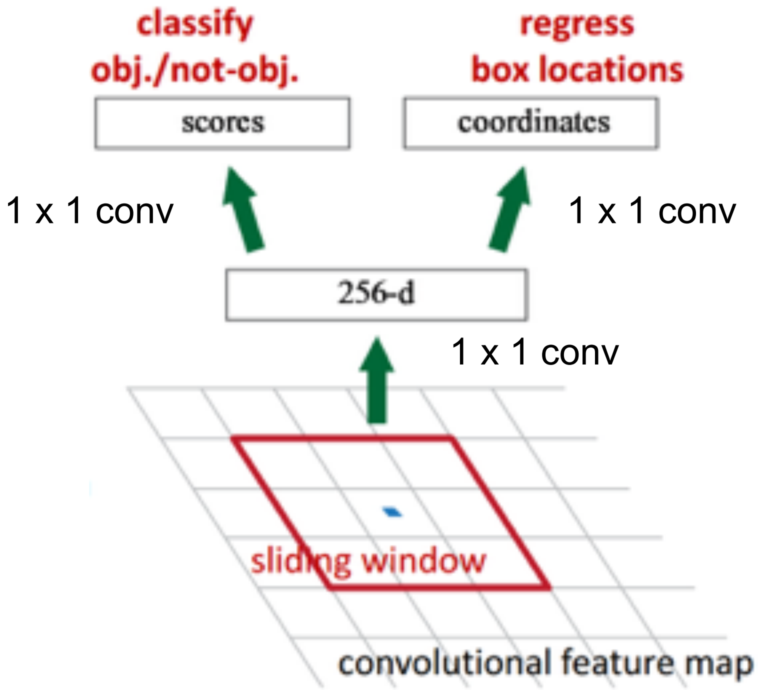
Faster R-CNN: Region Proposal Network

Slide a small window on the feature map

Build a small network for:
• classifying object or not-object, and
• regressing bbox locations

Position of the sliding window provides localization information with reference to the image

Box regression provides finer localization information with reference to this sliding window



Slide credit: Kaiming He

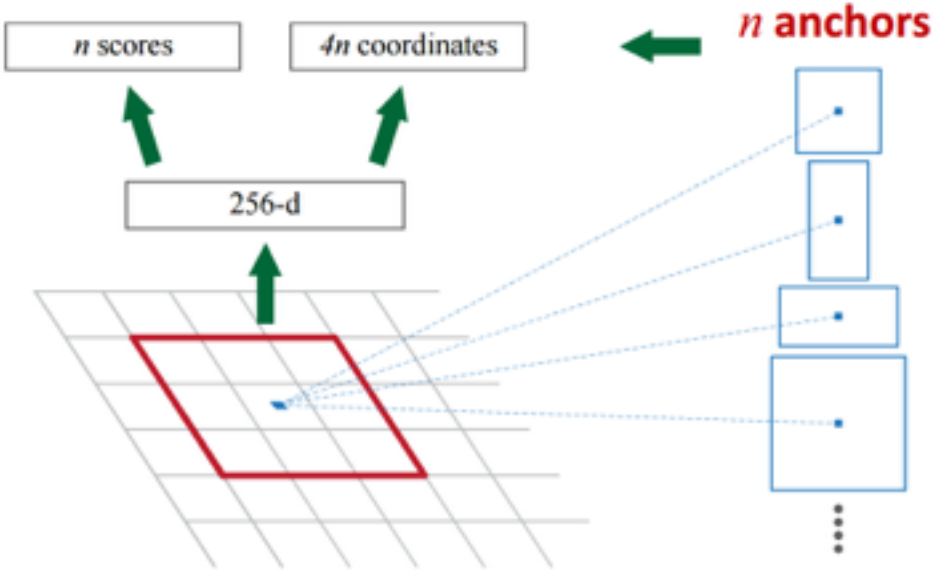
Faster R-CNN: Region Proposal Network

Use **N anchor boxes** at each location

Anchors are **translation invariant**: use the same ones at every location

Regression gives offsets from anchor boxes

Classification gives the probability that each (regressed) anchor shows an object



Faster R-CNN: Results

	R-CNN	**Fast R-CNN**	**Faster R-CNN**
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

Object Detection State-of-the-art: ResNet 101 + Faster R-CNN + some extras

training data	COCO train		COCO trainval	
test data	COCO val		COCO test-dev	
mAP	@.5	@[.5, .95]	@.5	@[.5, .95]
baseline Faster R-CNN (VGG-16)	41.5	21.2		
baseline Faster R-CNN (ResNet-101)	48.4	27.2		
+box refinement	49.9	29.9		
+context	51.1	30.0	53.3	32.2
+multi-scale testing	53.8	32.5	**55.7**	**34.9**
ensemble			**59.0**	**37.4**

He et. al, "Deep Residual Learning for Image Recognition", arXiv 2015

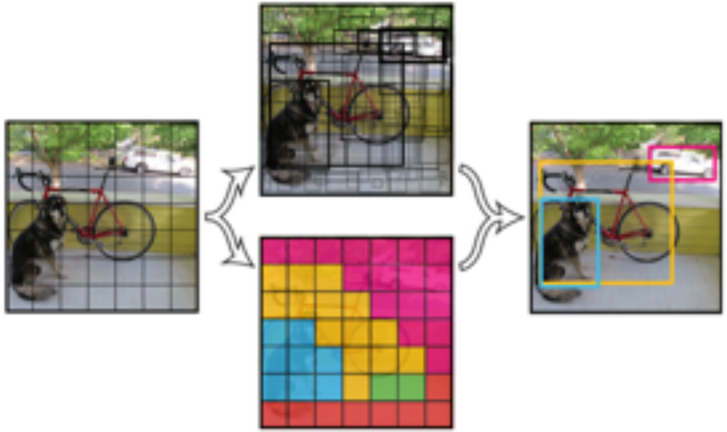
YOLO: You Only Look Once
Detection as Regression

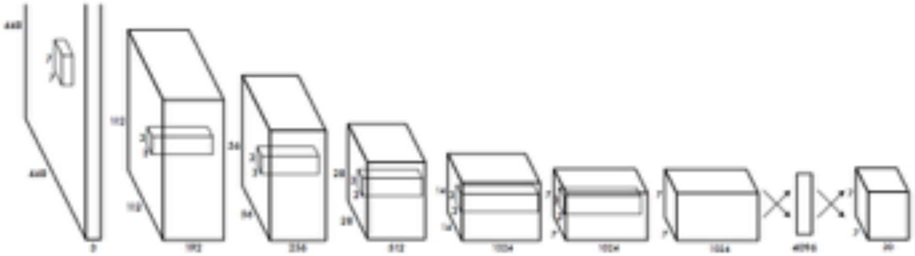
Divide image into S x S grid

Within each grid cell predict:
B Boxes: 4 coordinates + confidence
Class scores: C numbers

Regression from image to
7 x 7 x (5 * B + C) tensor

Direct prediction using a CNN





Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", arXiv 2015

Object Detection code links:

R-CNN
(Cafffe + MATLAB): https://github.com/rbgirshick/rcnn
Probably don't use this; too slow

Fast R-CNN
(Caffe + MATLAB): https://github.com/rbgirshick/fast-rcnn

Faster R-CNN
(Caffe + MATLAB): https://github.com/ShaoqingRen/faster_rcnn
(Caffe + Python): https://github.com/rbgirshick/py-faster-rcnn

YOLO
http://pjreddie.com/darknet/yolo/
(To be presented in class)