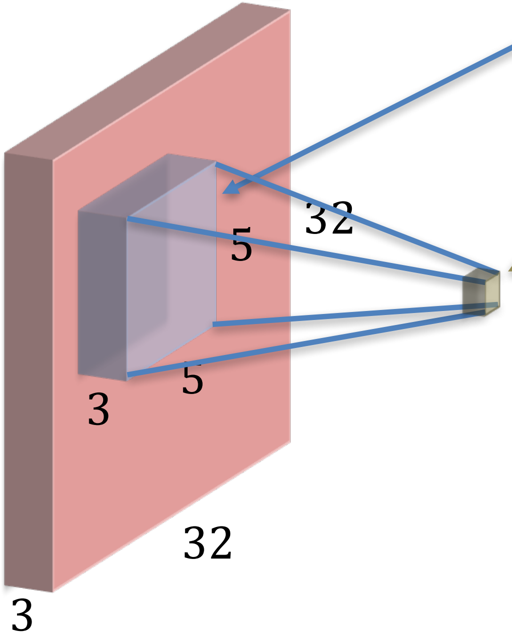
Lecture 7 Recap

Convolution Layers

$32\times32\times3$ image (pixels $\boldsymbol{x}$)

$5\times5\times3$ filter (weights $\boldsymbol{w}$)

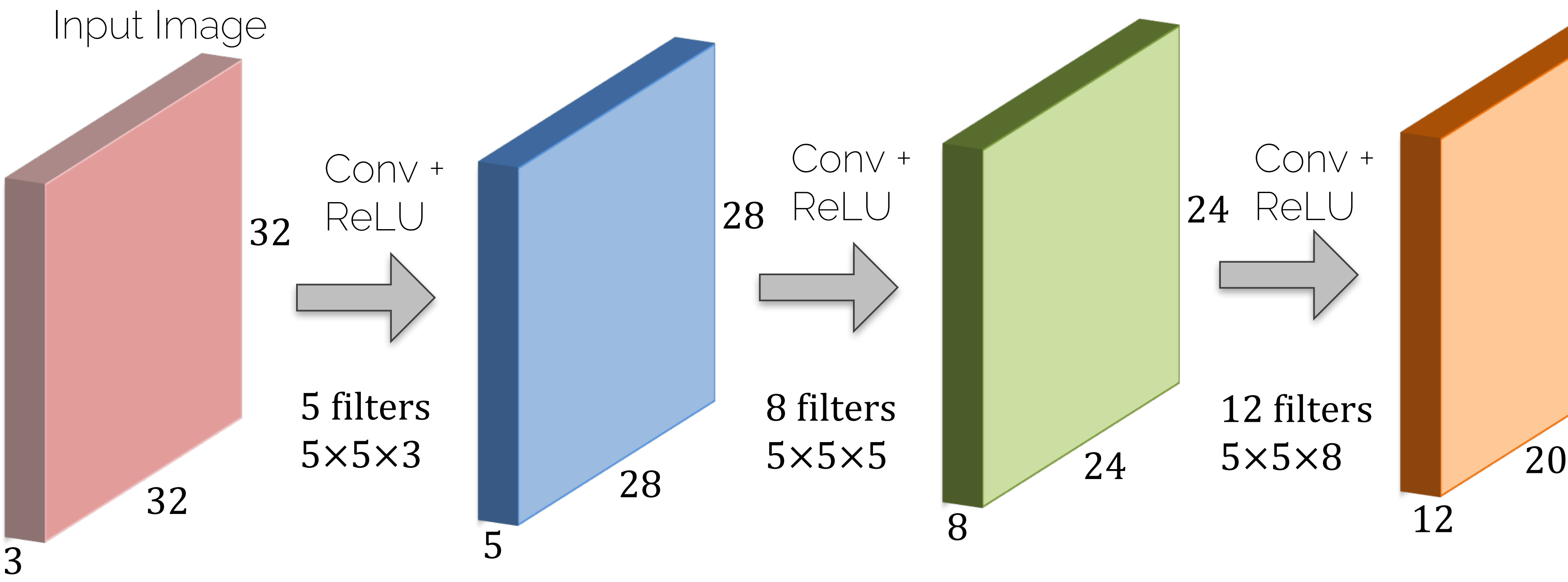


1 number:
dot product between filter weights $\boldsymbol{w}$
and $\boldsymbol{x_i - th}$ chunk of the image
Here: $5\cdot5\cdot3 = 75$-dim dot product + bias

$$\boxed{z_i = w^T x_i + b}$$

$5\times5\times3$ $\quad$ $5\times5\times3$ $\quad$ 1

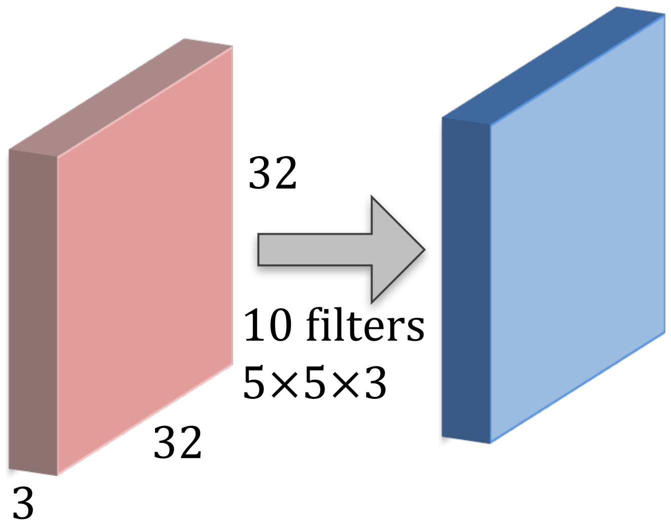
Convolution Layers: Dimensions



Convolution Layers: Dimensions

Example

Input image: $32 \times 32 \times 3$
10 filters 5×5
Stride 1
Pad 2



Output size is:

$$\frac{32 + 2 \cdot 2 - 5}{1} + 1 = 32$$

I.e., $32 \times 32 \times 10$

Remember

Output: $\left(\frac{N + 2 \cdot P - F}{S} + 1\right) \times \left(\frac{N + 2 \cdot P - F}{S} + 1\right)$

Convolution Layers: Dimensions

- Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
- Four hyperparameters
  - Number of filters K
  - Spatial filter extent F
  - Stride S
  - Zero padding P
- Output volume is of size $W_{out} \times H_{out} \times D_{out}$
  - $W_{out} = \frac{W_{in} - F + 2 \cdot P}{S} + 1$
  - $H_{out} = \frac{H_{in} - F + 2 \cdot P}{S} + 1$
  - $D_{out} = K$
- There are $F \cdot F \cdot D_{in}$ weights per filter; i.e., a total of $(F \cdot F \cdot D_{in}) \cdot K$ weights and K biases

- In the output volume, the D-th depth slice of size $(W_{out} \times H_{out})$ is the result of the convolution of the D-th over the input volume with a stride of S, and offset by its bias

Common settings:

$K =' \text{powers of } 2', \text{e.g.}, 32, 64, 128, 512$

$F = 3,\ S = 1,\ P = 1$

$F = 5,\ S = 1,\ P = 2$

$F = 5,\ S = 2,\ P = (whatever\ fits)$

$F = 1,\ S = 1,\ P = 0$

Slide by Li/Karpathy/Johnson

Pooling Layer: Max Pooling

Single depth slice of input

3	1	3	5
6	0	7	9
3	2	1	4
0	2	4	3

Max pool with
2×2 filters and stride 2

'Pooled' output

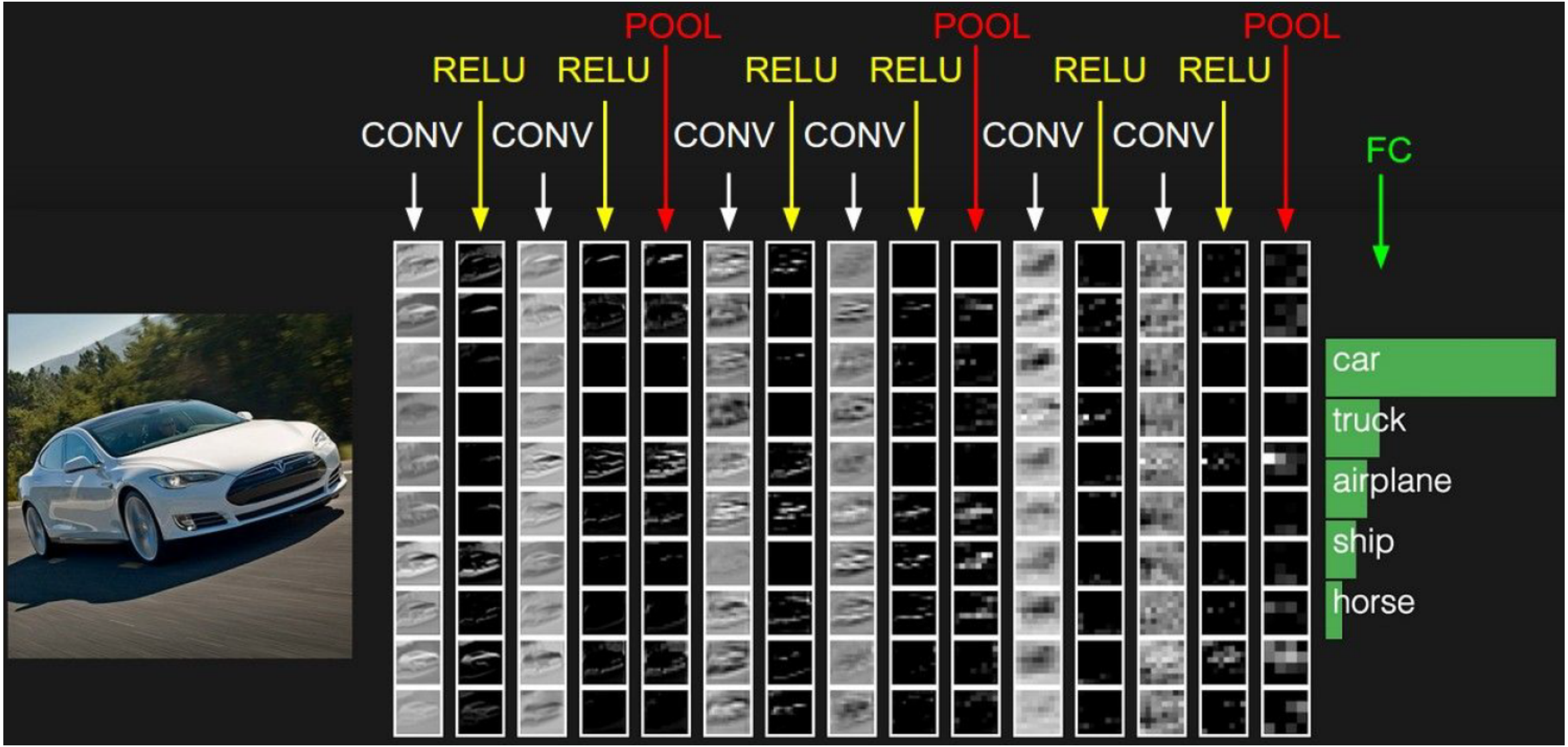
6	9
3	4

Pooling Layer

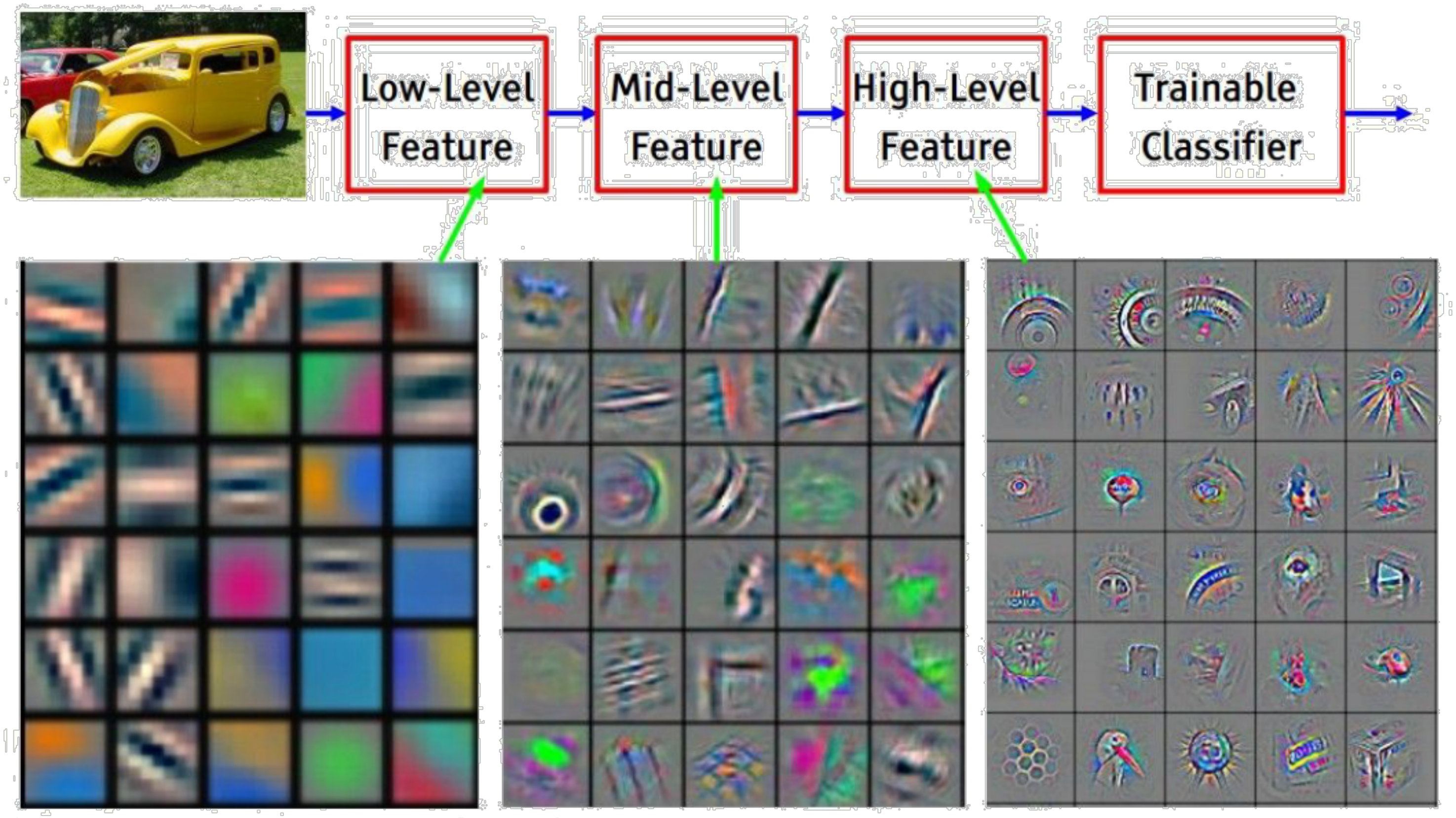
- Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
- Four hyperparameters
  - Spatial filter extent F
  - Stride S
- Output volume is of size $W_{out} \times H_{out} \times D_{out}$
  - $W_{out} = \frac{W_{in} - F}{S} + 1$
  - $H_{out} = \frac{H_{in} - F}{S} + 1$
  - $D_{out} = D_{in}$
- Does not contain parameters; e.g., its fixed function

Common settings:
$F = 2, S = 2$
$F = 3, S = 2$

Convolutional Neural Network



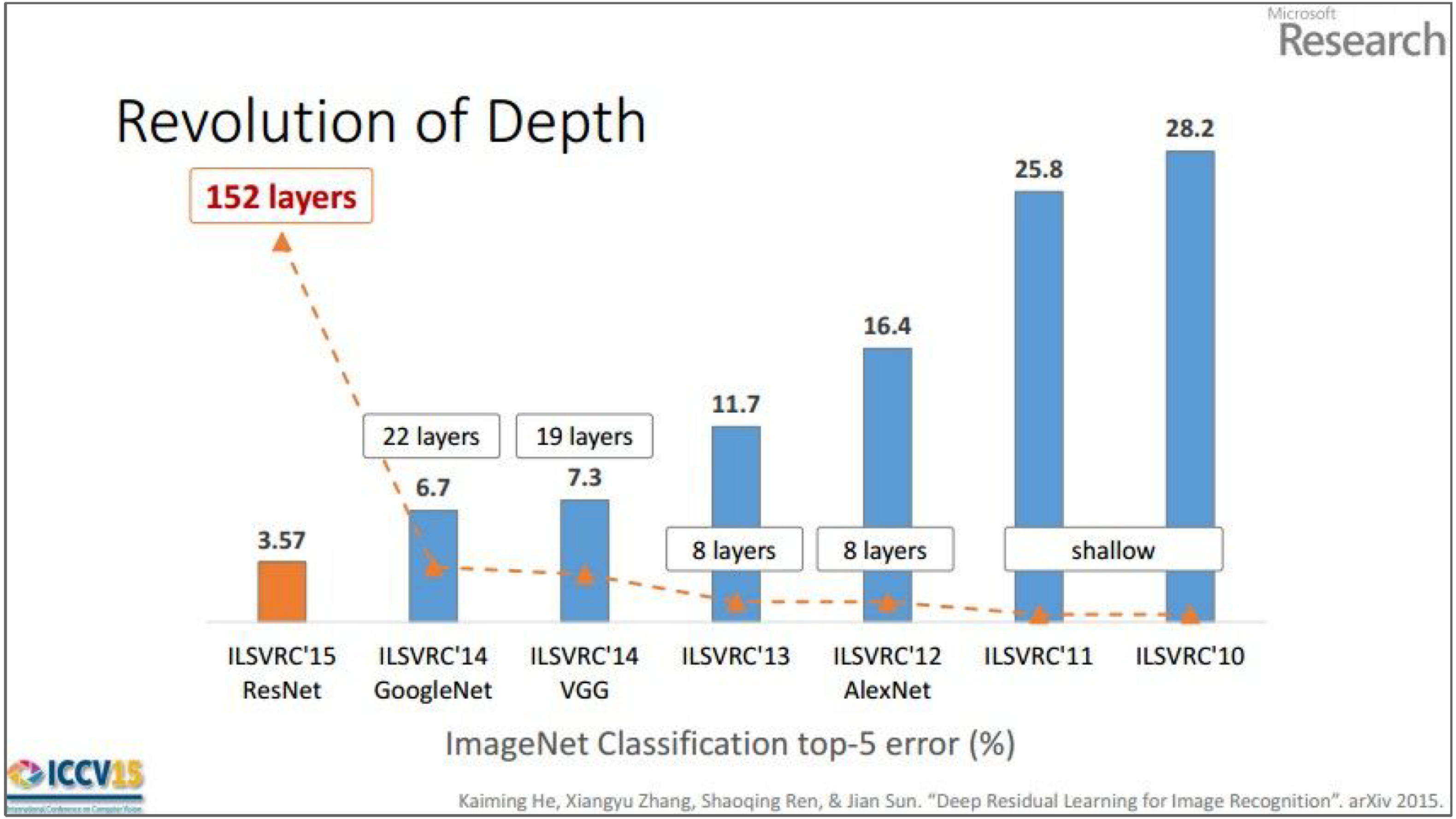
Convolutional Neural Network



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

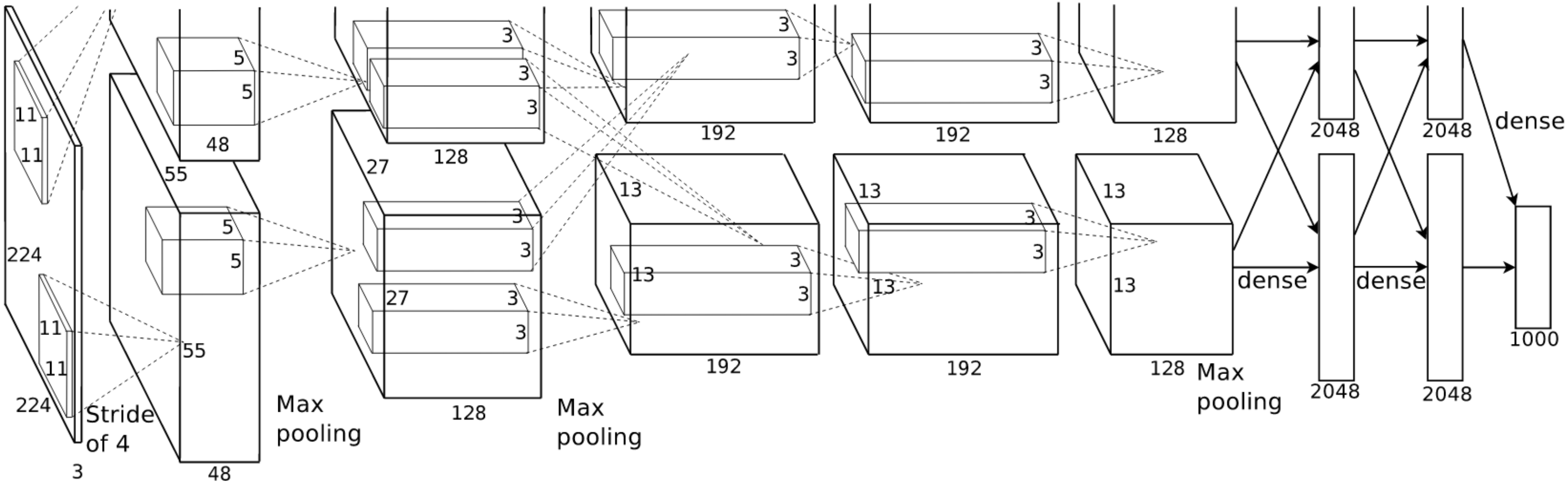
Slide by LeCun

CNN Architectures



CNN Architectures: AlexNet

[Krizhevskv et al. 2012]



Input: $227 \times 227 \times 3$ images

Conv1 -> MaxPool1 -> Norm1 -> Conv2 -> MaxPool2 -> Norm2 ->
-> Conv3 -> Conv4 -> Conv5 -> Maxpool3 -> FC6 -> FC7 -> FC8

First use of ReLU!

CNN Architectures: VGGNet

[Simonyan and Zisserman 2014]

Analyze different architectures!

Best model:

Ensemble
ImageNet top 5 error: 11.2% -> 7.3%

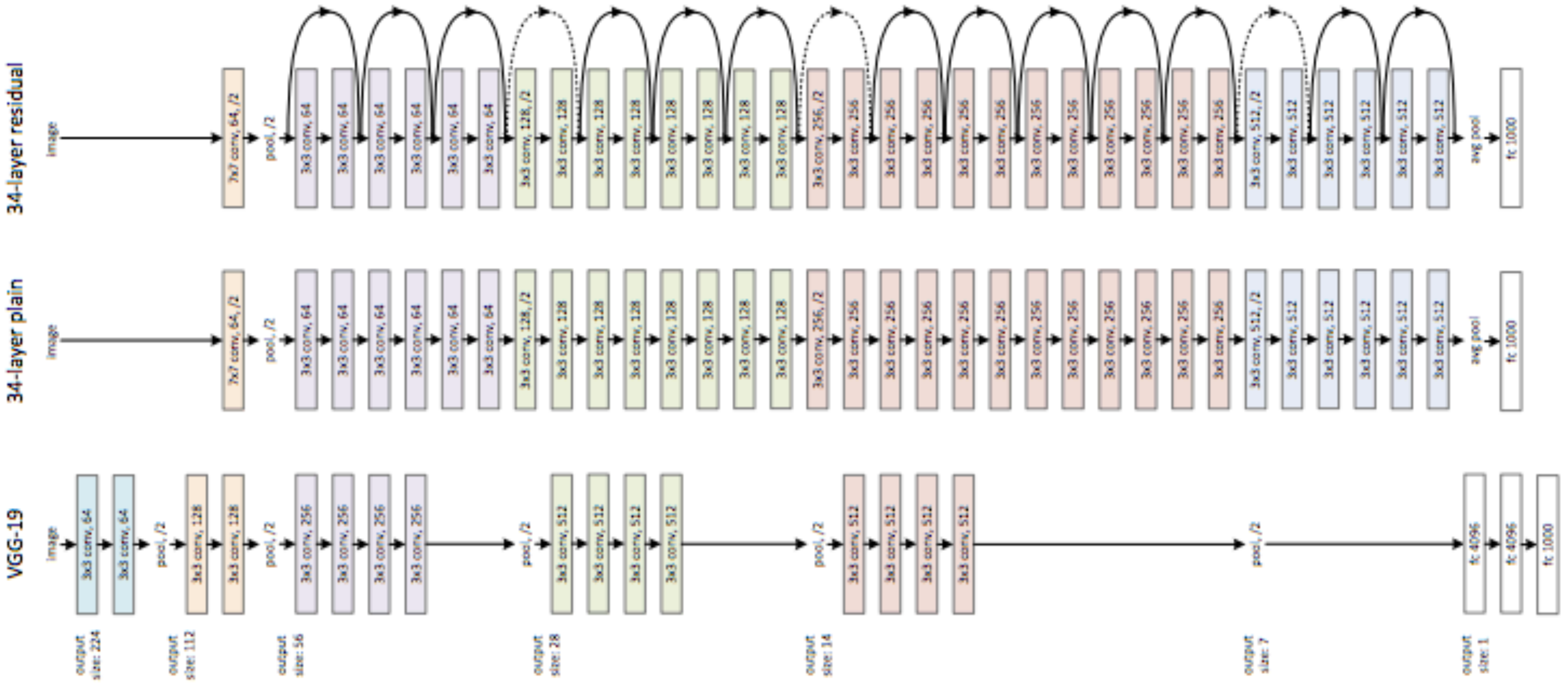
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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

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

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

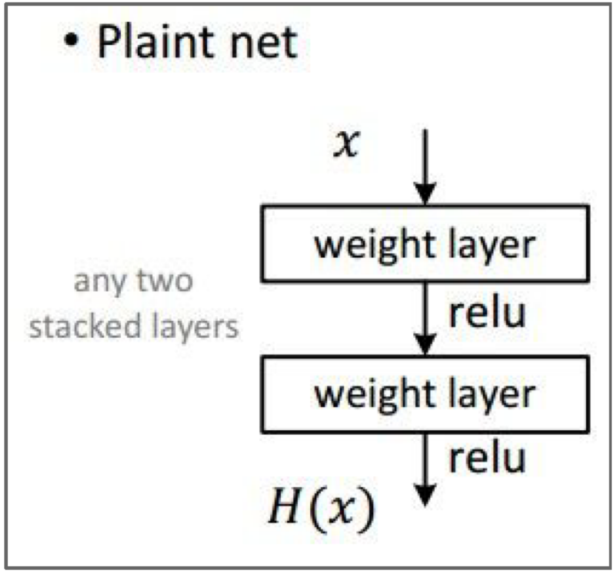
CNN Architectures: ResNet

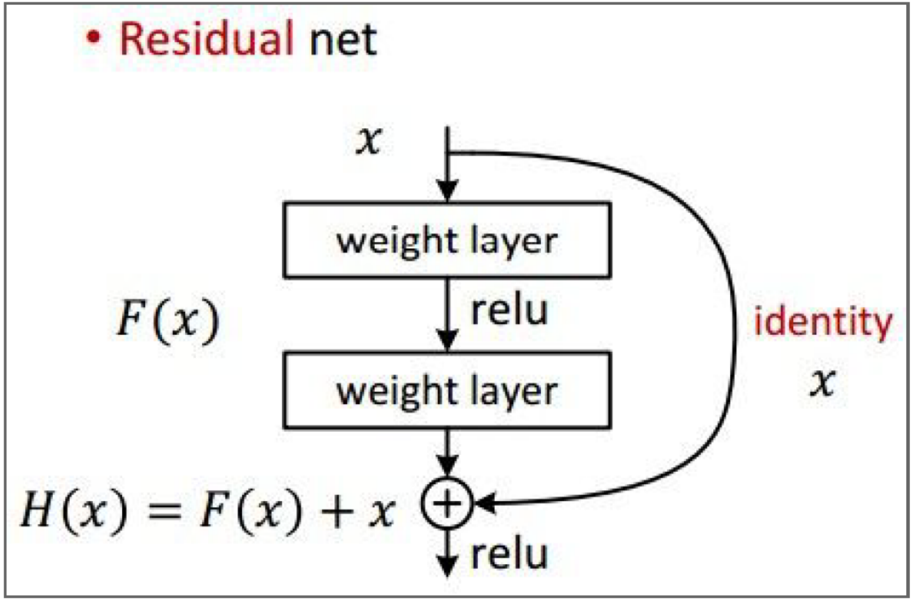
[He et al. 2015]



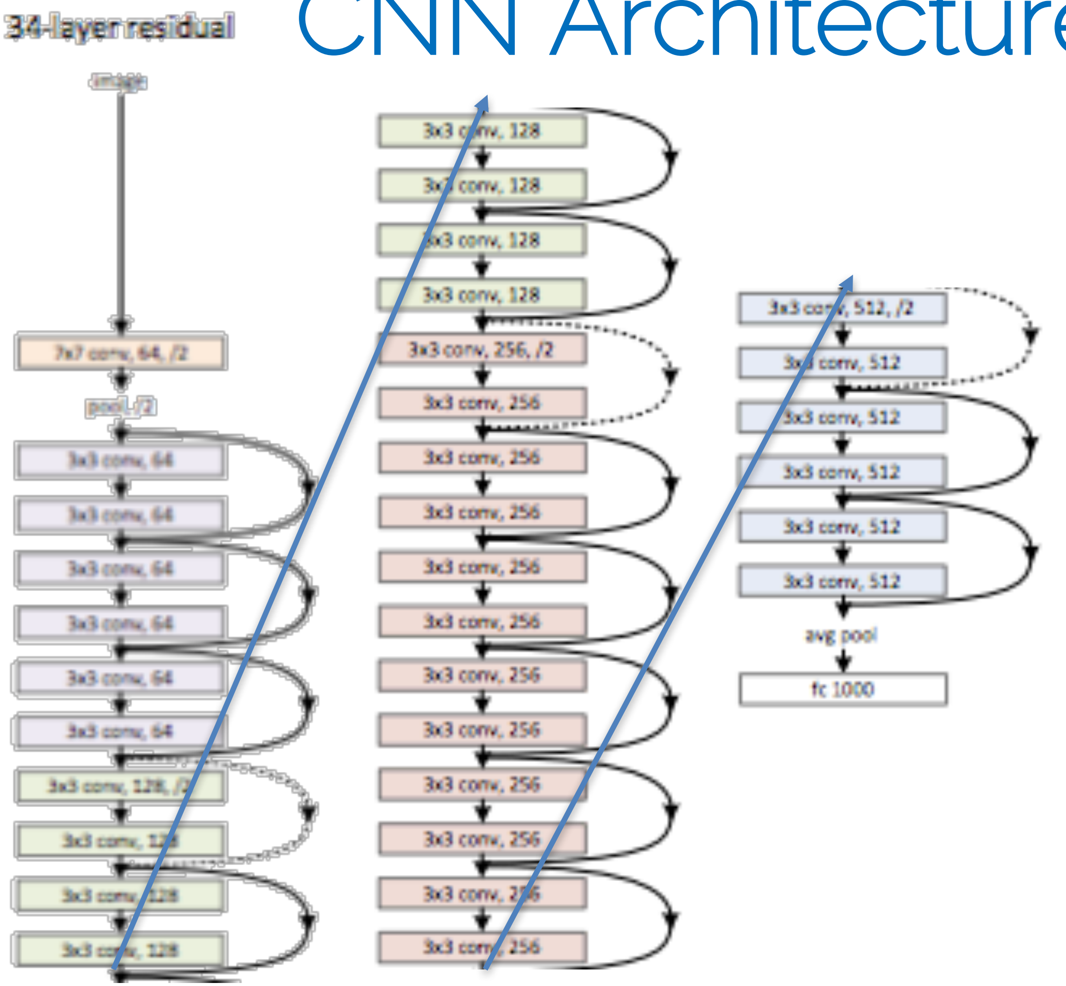
CNN Architectures: ResNet

[He et al. 2015]



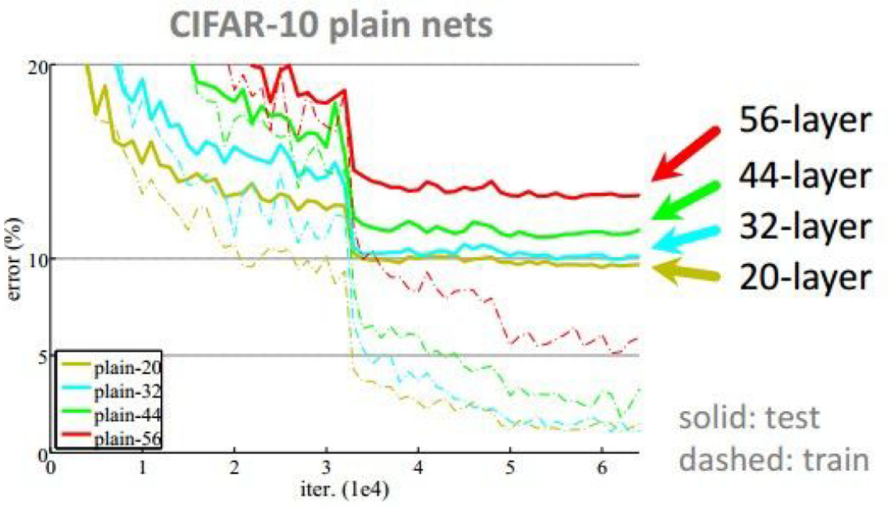


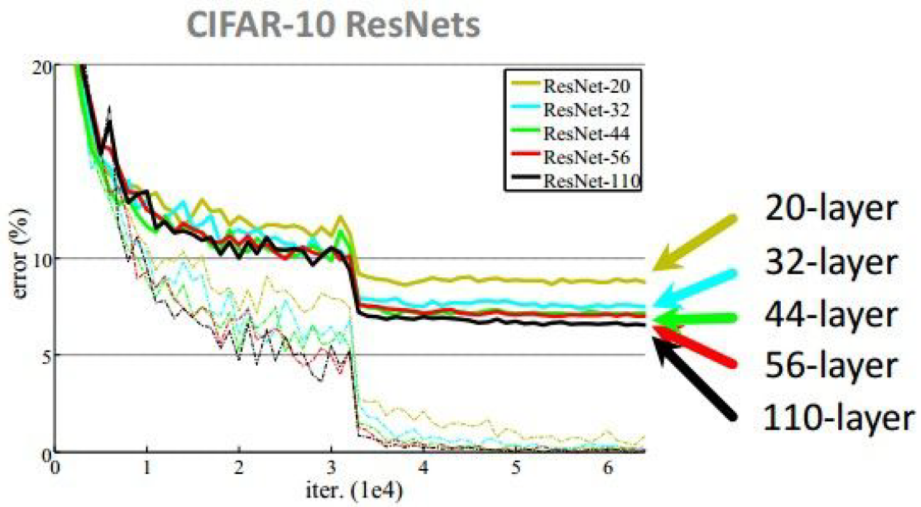
CNN Architectures: ResNet



- Batch norm after every Conv Layer
- Xavier/2 init by He et al.
- SGD + Momentum (0.9)
- Learning rate 0.1, divided by 10 when plateau
- Mini-batch size 256
- Weight decay of 1e-5
- No dopout!

CNN Architectures





Slide by Li/Karpathy/Johnson

CNN Architectures: ResNet

[He et al. 2015]

- What Conv Layers do spatially, ResNet and Inception models do across layers (kind of)

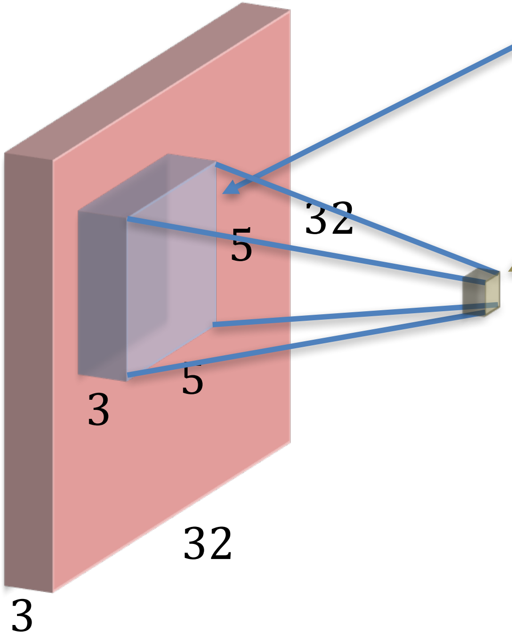
MSRA @ ILSVRC & COCO 2015 Competitions

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

Backprop through CNN Layers

$32{\times}32{\times}3$ image (pixels $\boldsymbol{x}$)

$5{\times}5{\times}3$ filter (weights $\boldsymbol{w}$)

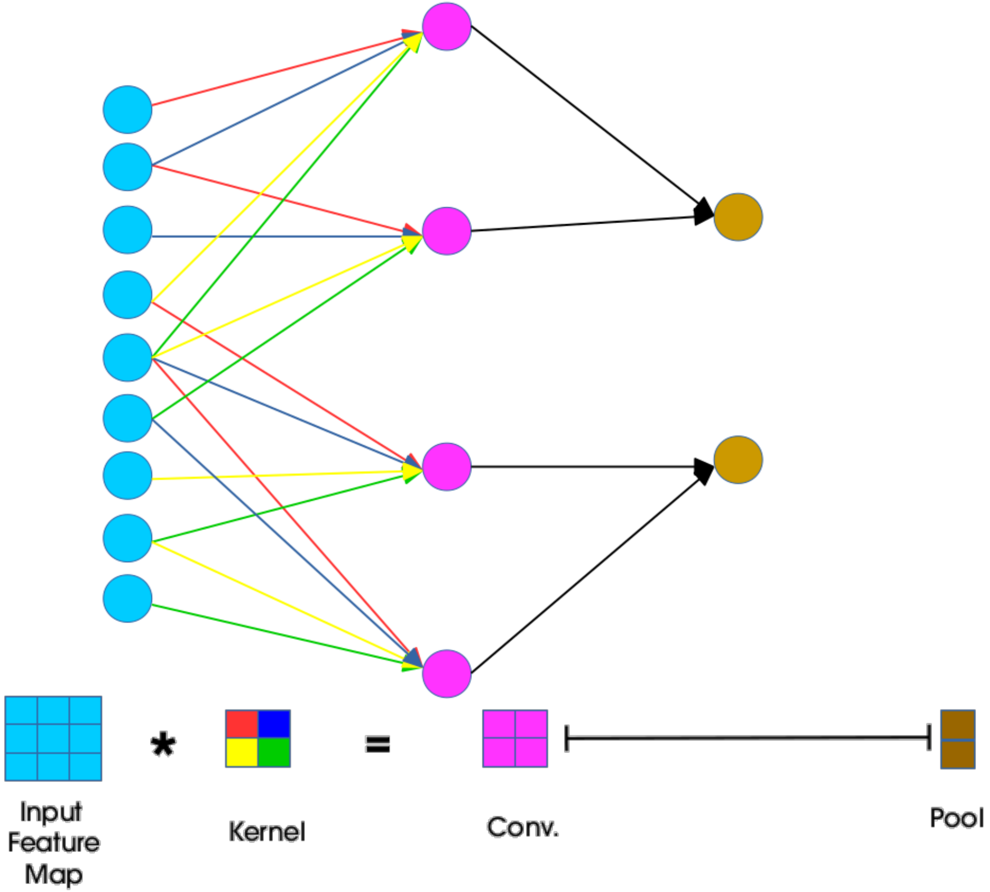


1 number:
dot product between filter weights $\boldsymbol{w}$
and $\boldsymbol{x_i - th}$ chunk of the image
Here: $5 \cdot 5 \cdot 3 = 75$-dim dot product + bias

$$\boxed{z_i = w^T x_i + b}$$

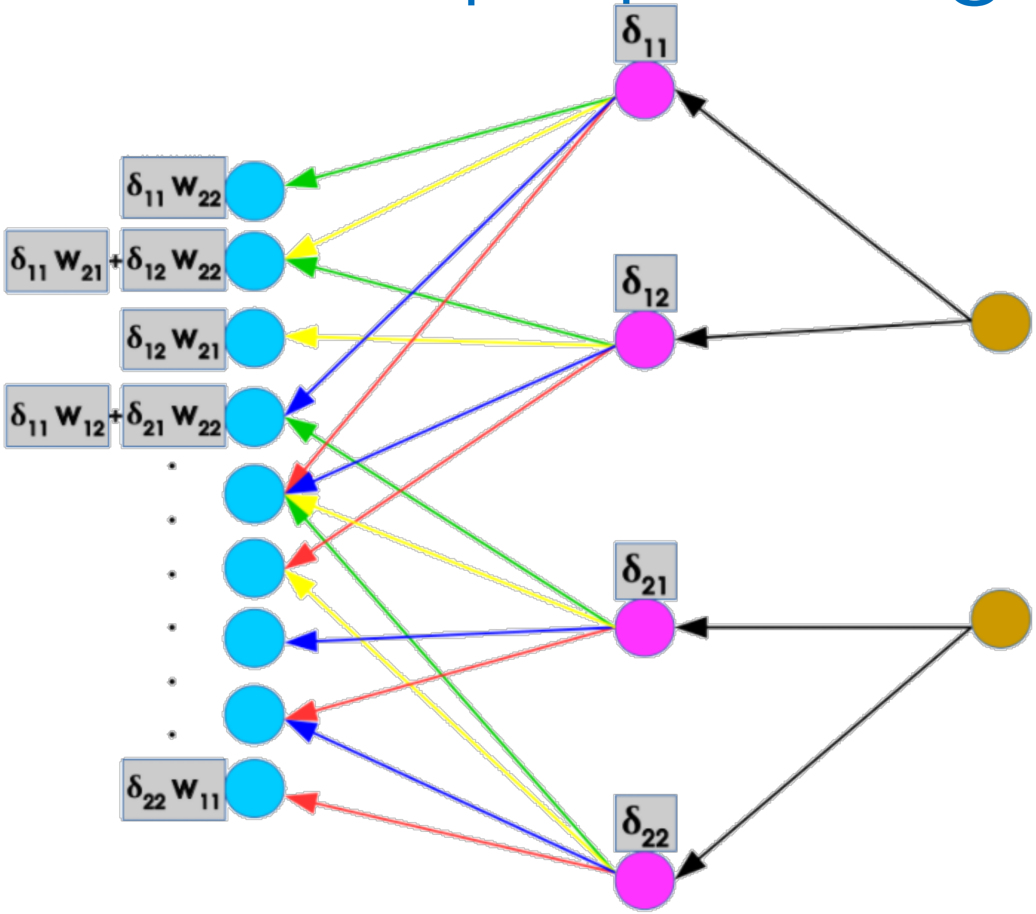
$5{\times}5{\times}3$ $5{\times}5{\times}3$ 1

Backprop through CNN Layers



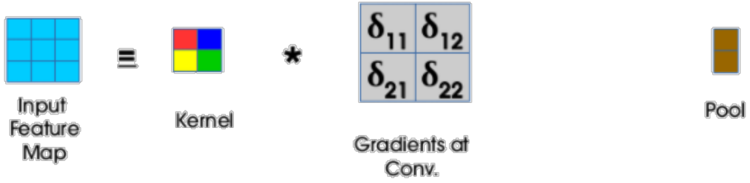
http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Backprop through CNN Layers



gradient

$$\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22}$$



http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Backprop through CNN Layers

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

Input: 16-dim vector
Output: 4-dim vector (will be re-shaped as 2 x 2 eventually)

Backward pass is simply multiplying with C^T

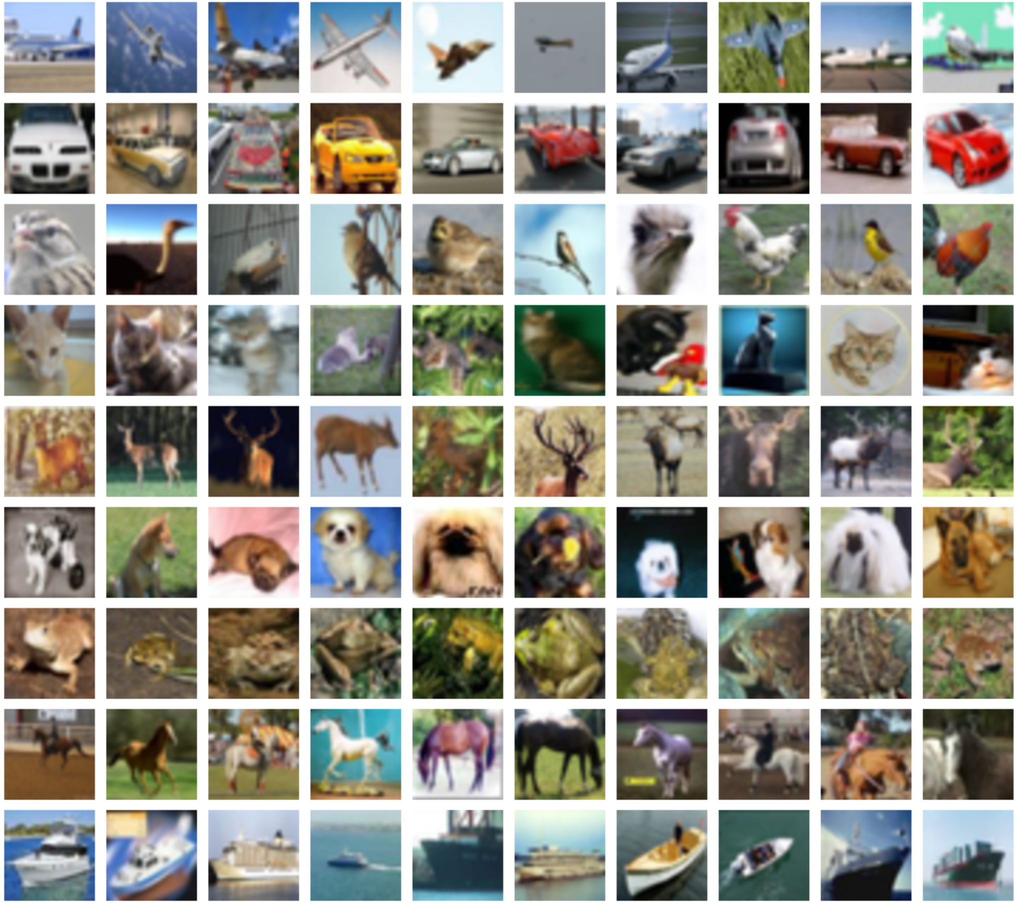
Task for at home: think it through on a piece of paper ☺

[Dumoulin et al. 16]

Using Convolutional Neural Networks

Classification on CIFAR

airplane
automobile
bird
cat
deer
dog
frog
horse
ship



60k 32 x 32 RGB images
6k images per class
50k training and 10k test

[Krizhevsky 09]

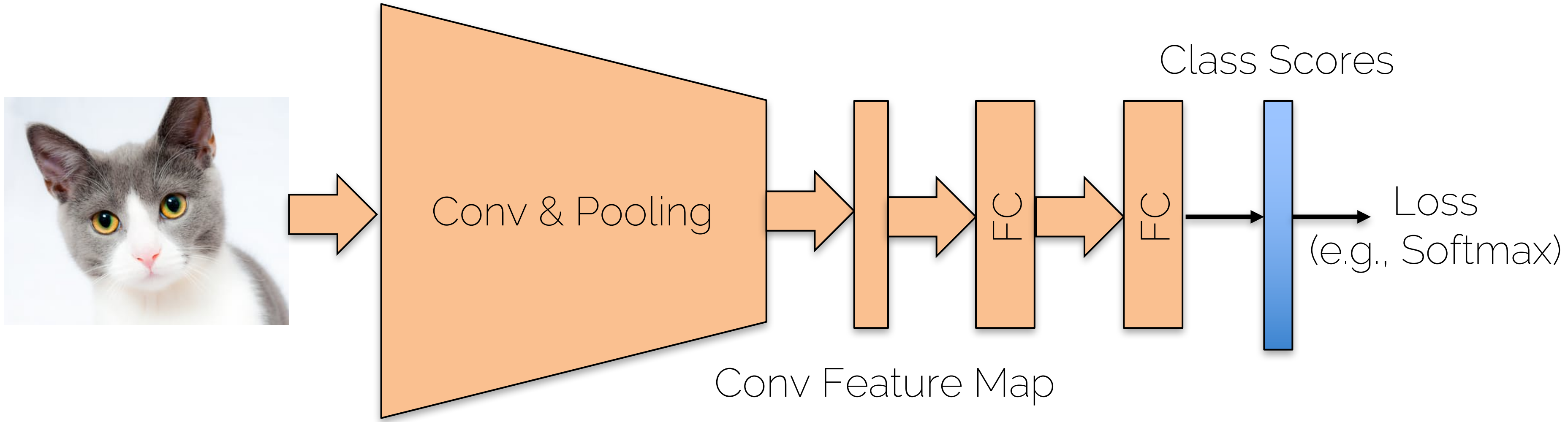
Classification on CIFAR

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

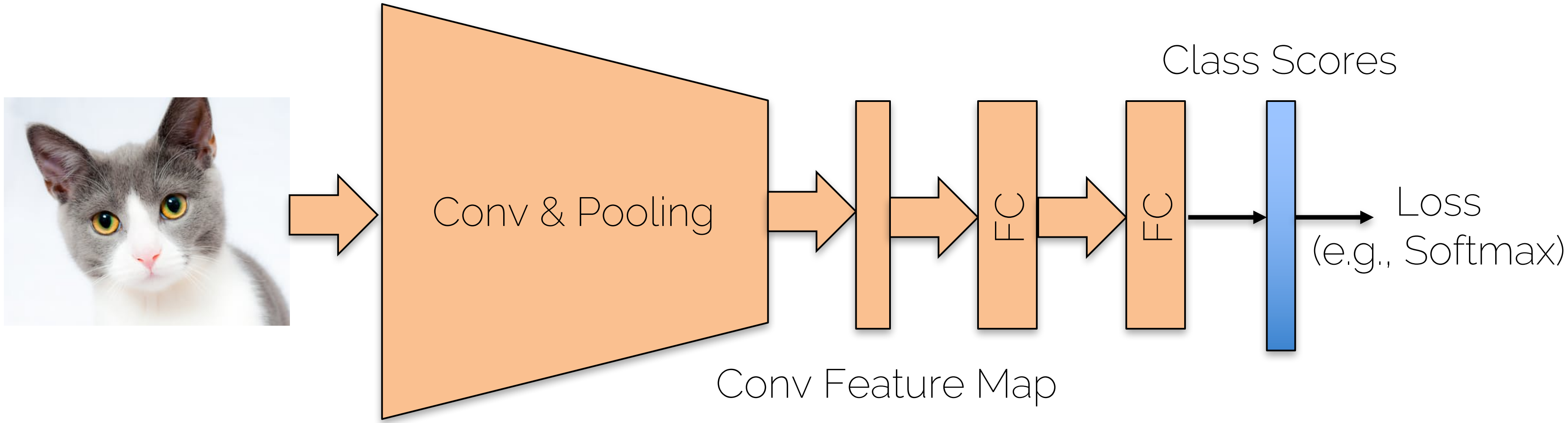
State of the art on CIFAR-10 is > 90%
It has isolated objects, so it's the 'straight-forward' applications of CNNs

[Karpathy]

How to Train in Practice?



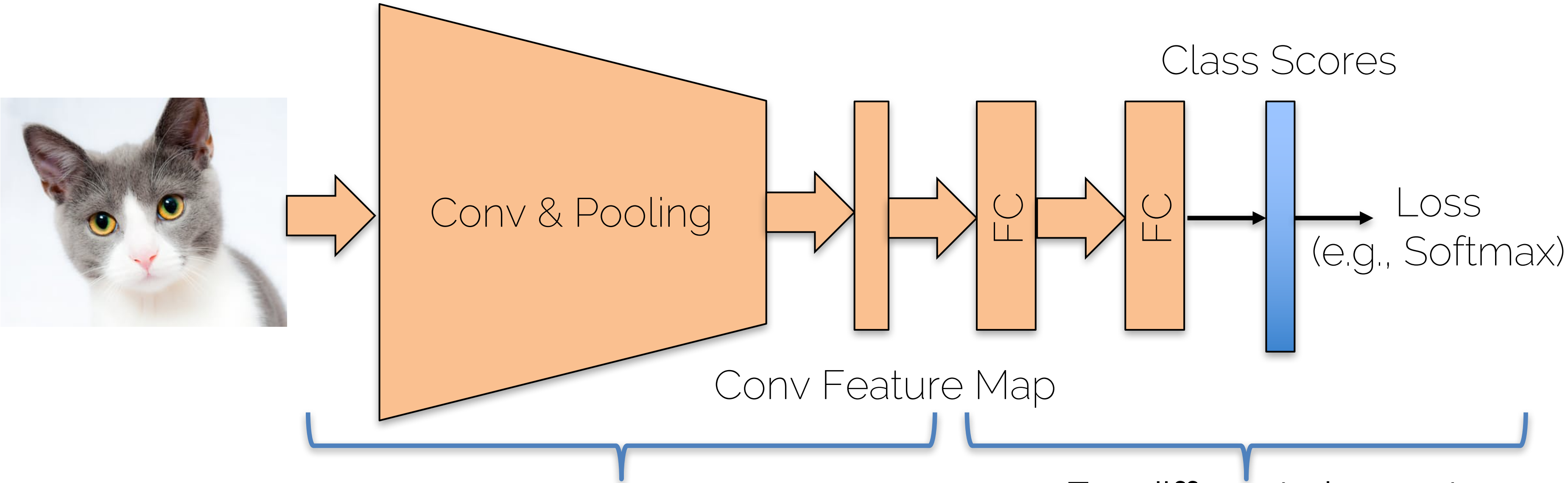
How to Train in Practice?



E.g. AlexNet, VGG, GoogLeNet

Train on ImageNet once (10 mio images) -> 1-2 weeks

How to Train in Practice?

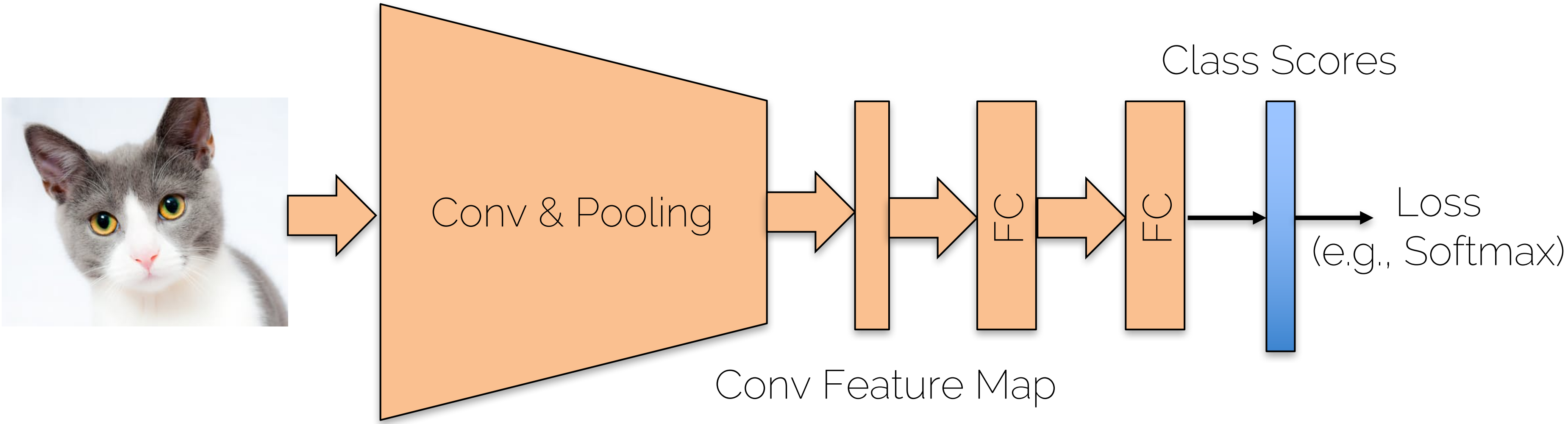


E.g. AlexNet, VGG, GoogLeNet

Use Pre-Trained Network (e.g., download model)
-> keep ConvLayers fixed

For different class set,
only train FCs
-> new class scores
-> less training data
-> faster training

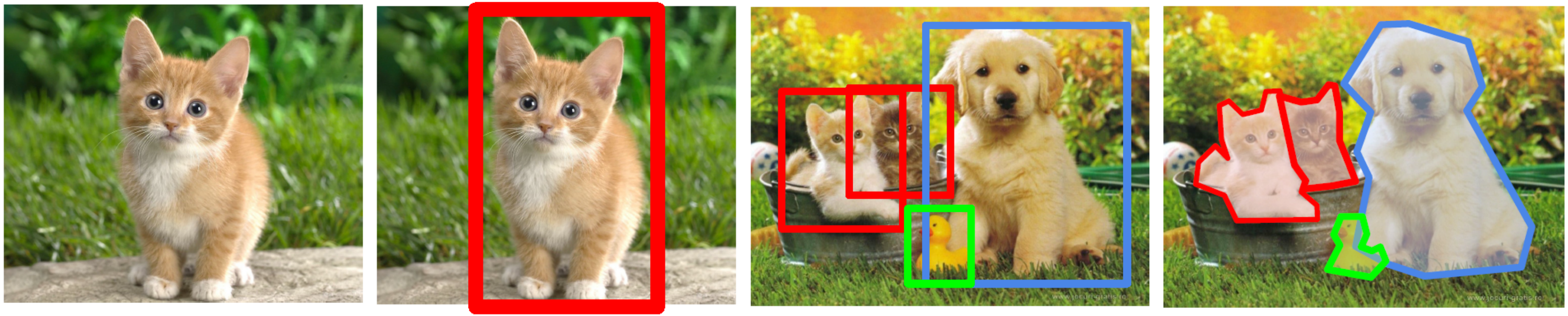
How to Train in Practice?



Always think about these strategies!
- Try to start with existing, pre-trained models
- In the assignments, don't try to train ImageNet model from scratch

Using CNNs in Computer Vision

Classification	Classification + Localization	Object Detection	Instance Segmentation



CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK

Single object — Multiple objects

Credit: Li/Karpathy/Johnson

Using CNNs in Computer Vision

Classification

Classification + Localization

Object Detection

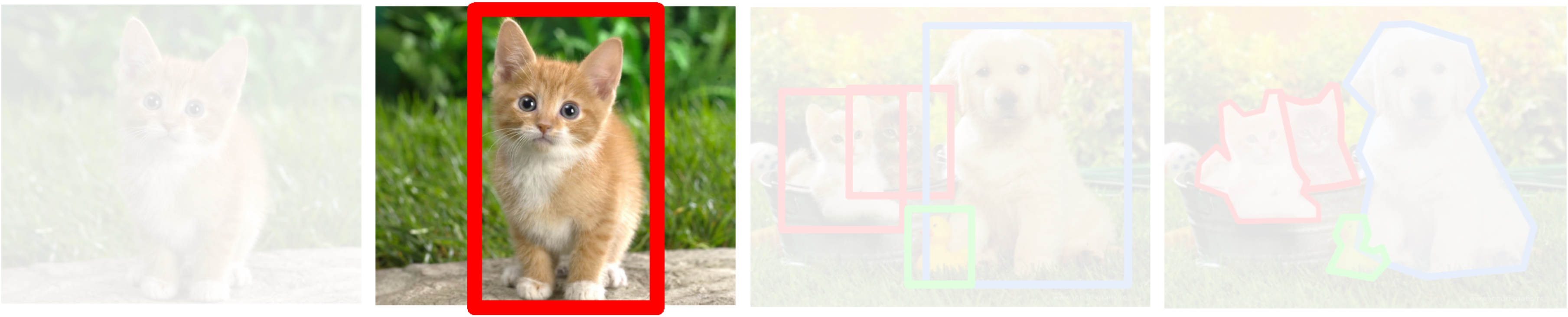
Instance Segmentation

CIFAR 10 +
"raw" CNN ☺

Credit: Li/Karpathy/Johnson

Using CNNs in Computer Vision

Classification	Classification + Localization	Object Detection	Instance Segmentation



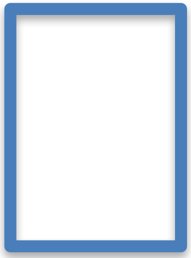
CIFAR 10 +
"raw" CNN ☺

Credit: Li/Karpathy/Johnson

Using CNNs in Computer Vision

- Classification:
  - Input: image
  - Output: class label
  - Loss of class accuracy
- Localization:
  - Input: image
  - Output: box in image (x, y, w, h)
  - Loss over IoU (intersection over union)
- Classification + Localization: combine both

class = cat



(x, y, w, h)

Localization as Regression

Input: input

(single object)

CNN

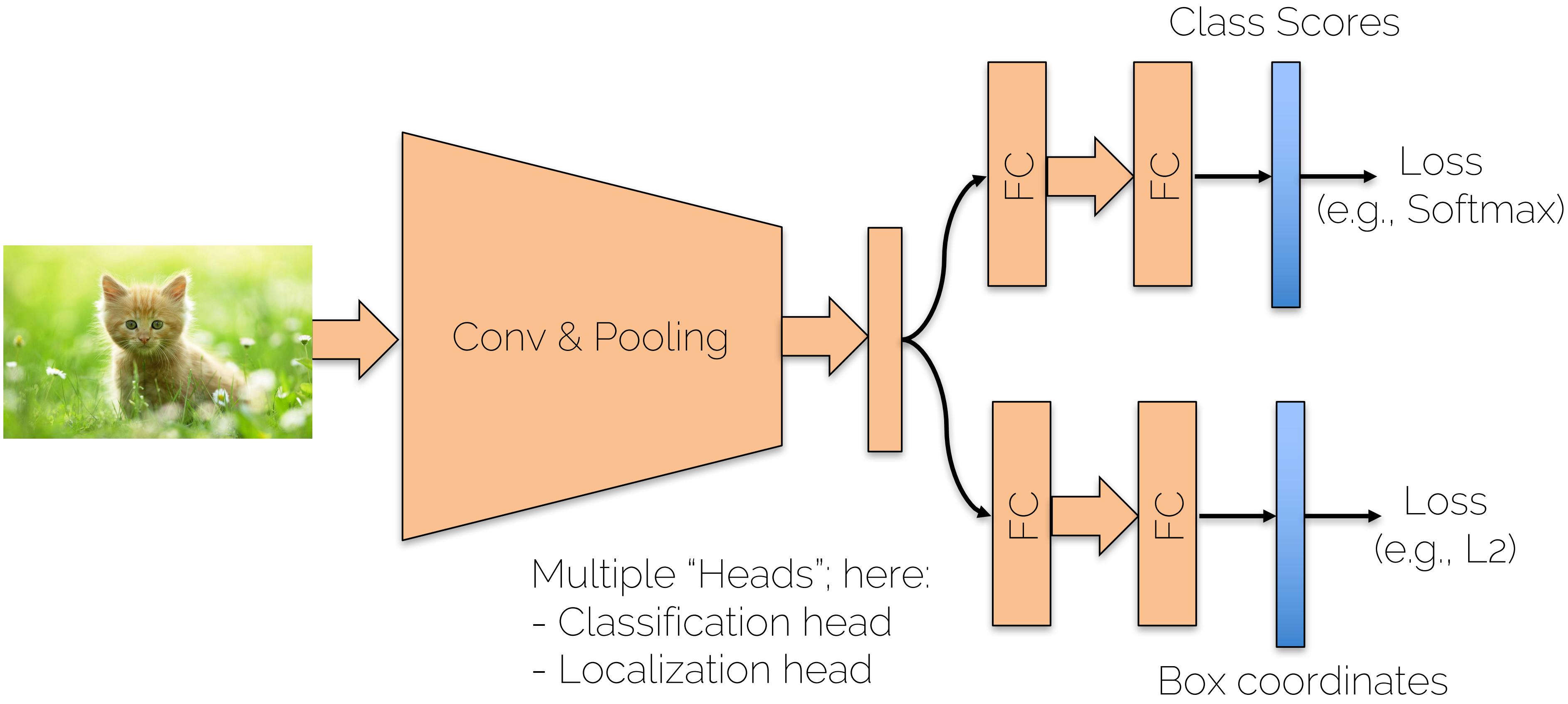
Output:
Box coordinates
(x, y, w, h)

Loss:
L2 distance

$$\sqrt{(x-x')^2+(y-y')^2+(w-w')^2+(h-h')^2}$$

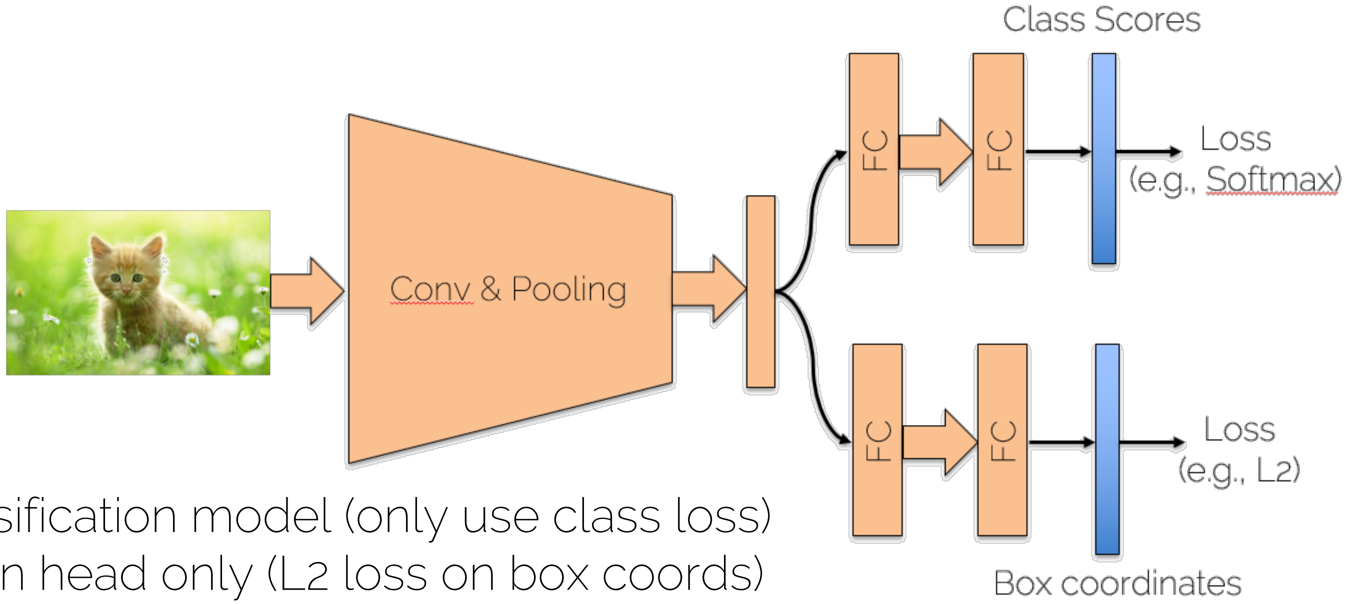
Ground Truth (from annotation):
Box coordinates
(x', y', w', h')

Classification + Localization: Regression



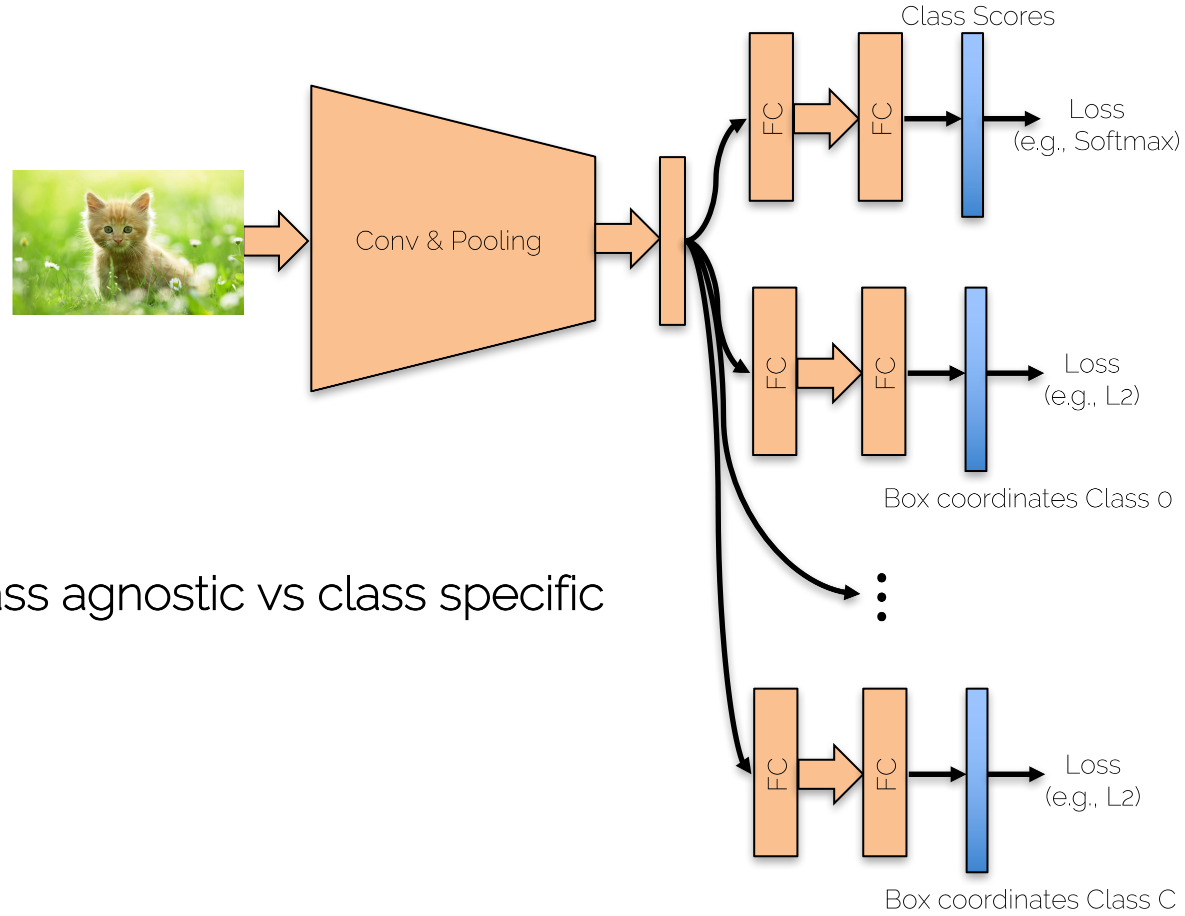
Multiple "Heads"; here:
- Classification head
- Localization head

Classification + Localization: Regression



1) Train only classification model (only use class loss)
2) Train regression head only (L2 loss on box coords)
3) At test time use both heads

Classification + Localization: Regression



Classification head
C classes

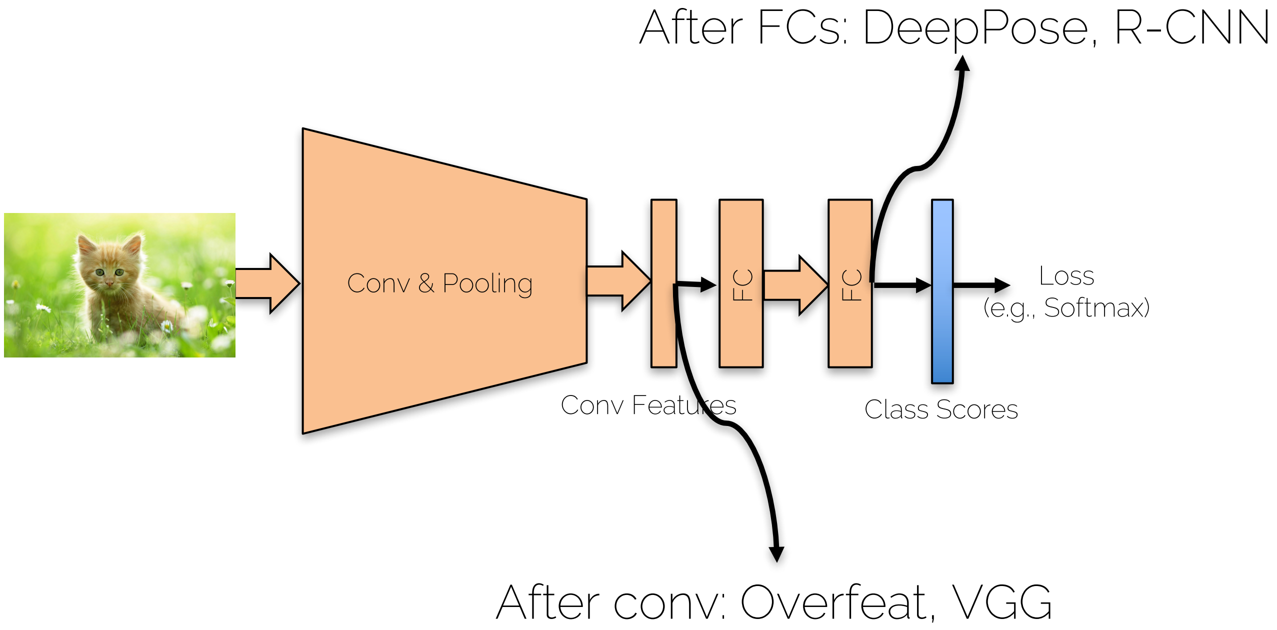
Class specific
localization heads

C x 4 numbers
(one per box)

class agnostic vs class specific

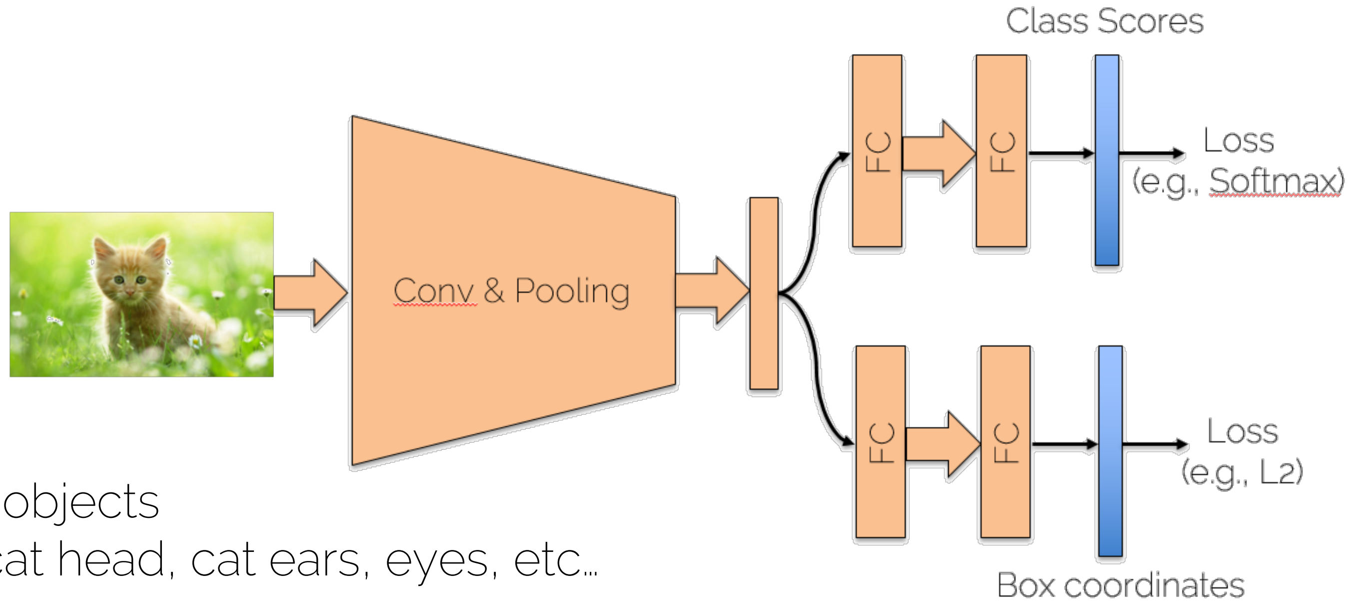
Classification + Localization: Regression

- Where to attach the regression head?



Classification + Localization: Regression

"Hack" for localization multiple, bit **fixed** number of objects

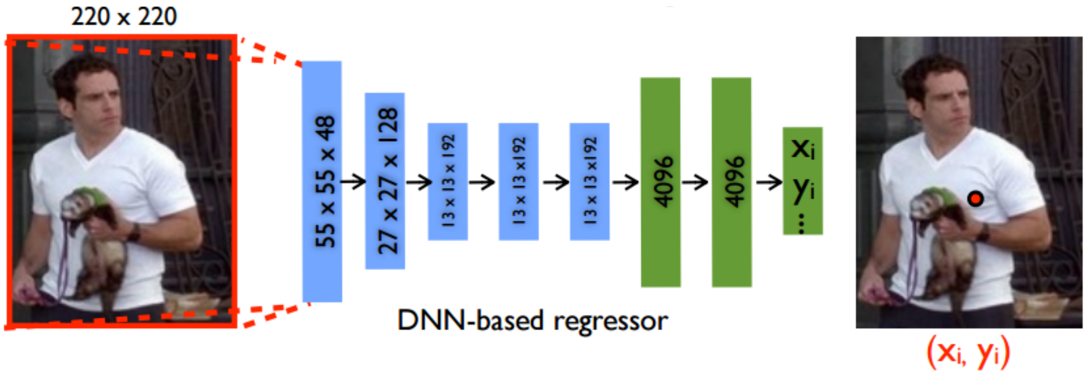


Exactly K objects
e.g., cat, cat head, cat ears, eyes, etc…

K x 4 numbers

Classification + Localization: Regression

- Human Pose Estimation
  - Person has K joints (similar to Kinect SDK)
  - Regress (x, y) for each joint from last FC of AlexNet
  - Post Refinment, use Normalized Device Coords



[Toshev and Szegedy: "DeepPose" 14]

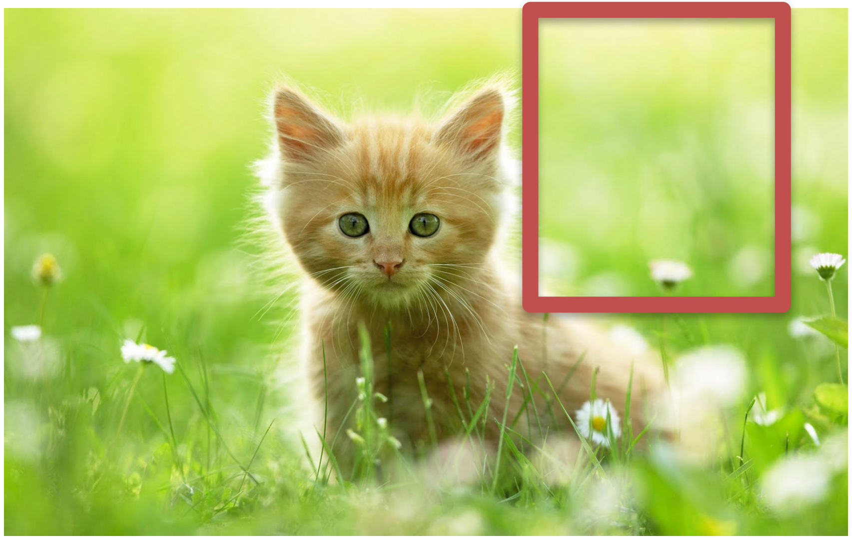
Classification + Localization: Regression

- Adding regression is very simple and efficient!
- Think about smart architecture design
- Can combine different Conv parts and "Heads"

Classification + Localization: Sliding Window

1. Train classification network on specific object(s)
2. Select random bounding box: check class score
3. Brute force testing: everywhere at every scale
4. Take location with highest class score

Classification + Localization: Sliding Window



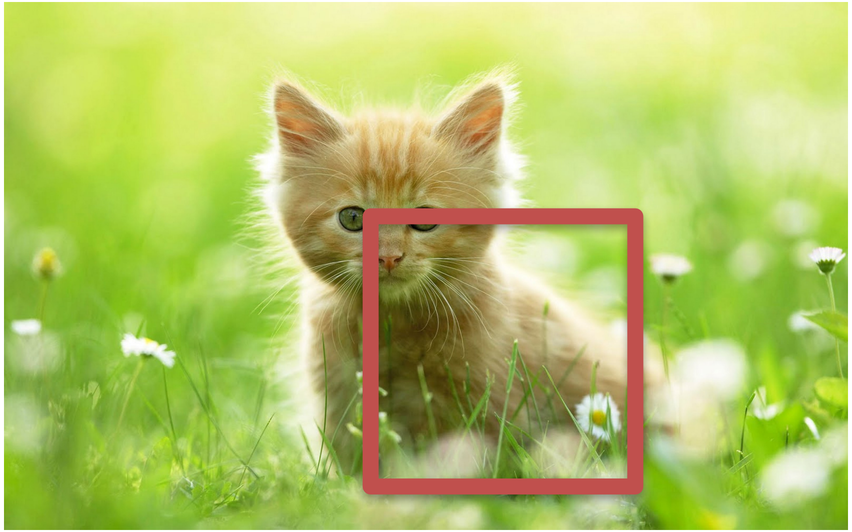
Class score (cat):

Box location 0 -> score 0.02

Classification + Localization: Sliding Window

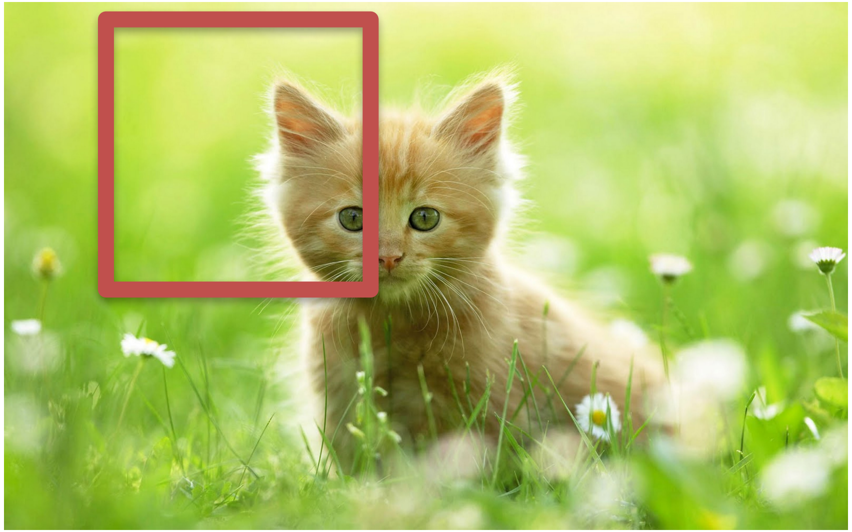
Class score (cat):
Box location 0 -> score 0.02
Box location 1 -> score 0.2

Classification + Localization: Sliding Window



Class score (cat):
Box location 0 -> score 0.02
Box location 1 -> score 0.2
Box location 2 -> score 0.42

Classification + Localization: Sliding Window



Class score (cat):

Box location 0 -> score 0.02

Box location 1 -> score 0.2

Box location 2 -> score 0.42

Box location 3 -> score 0.31

Classification + Localization: Sliding Window

Class score (cat):

Box location 0 -> score 0.02
Box location 1 -> score 0.2
Box location 2 -> score 0.42
Box location 3 -> score 0.31
Box location 4 -> score 0.8

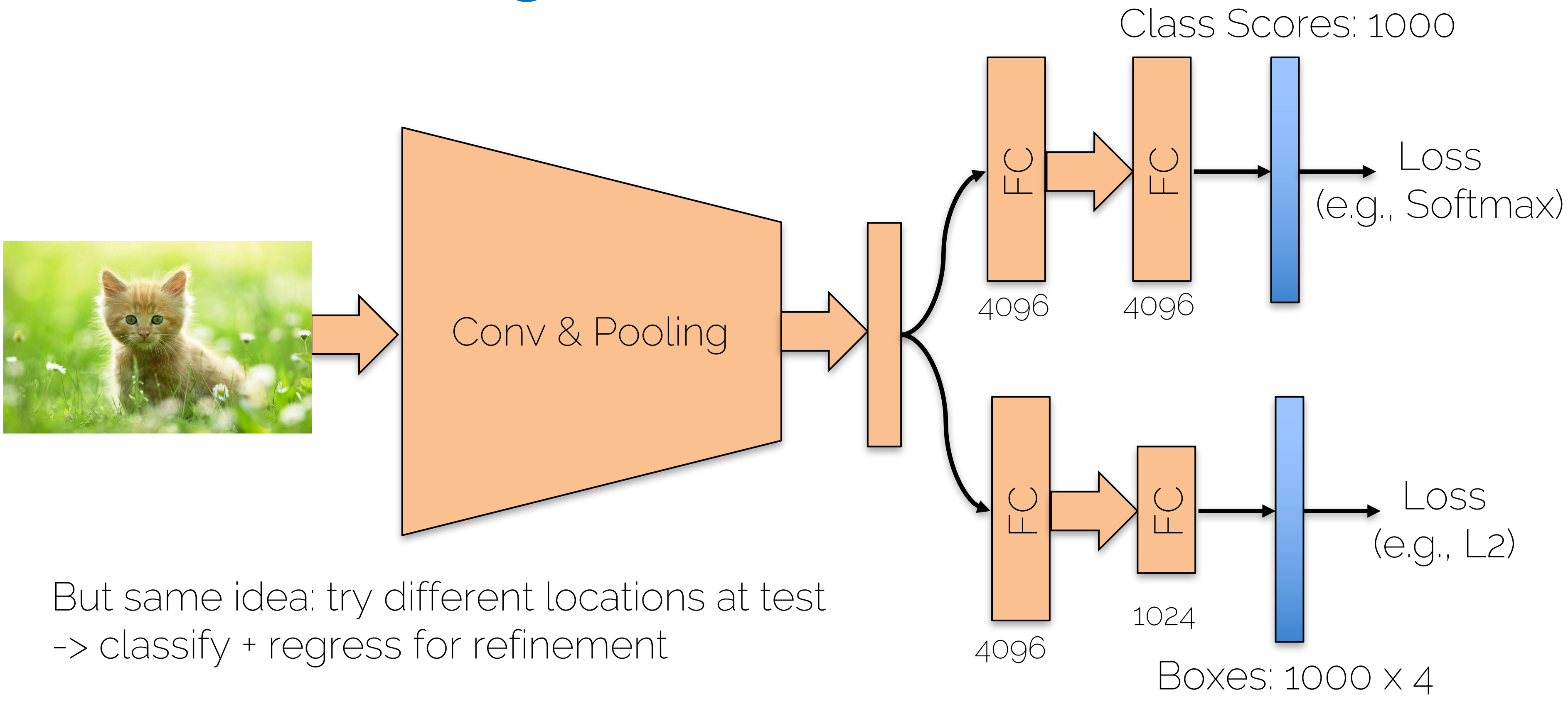
Take winning box location as result

Classification + Localization: Sliding Window

- Problem:
  - Slow testing, needs a lot of tests to find a good one.
  - Need to get *really* lucky to find the *exact* box
  - Harder to train, since classifier does not know about loc

- Idea:
  - Combine with regressor for refinenment
  - Train both

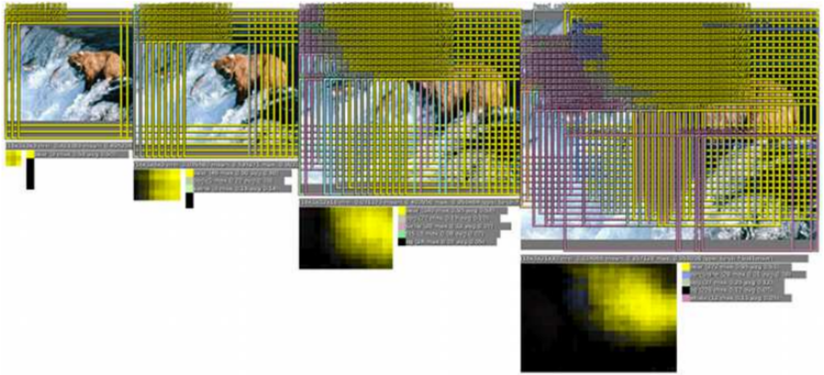
Sliding Window: Overfeat



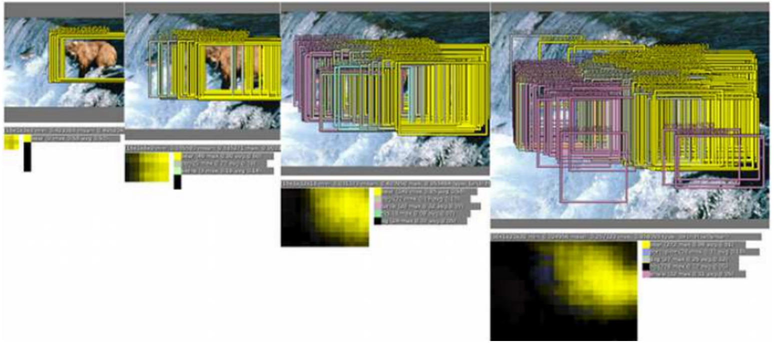
But same idea: try different locations at test
-> classify + regress for refinement

[Sarmenet et al.: Overfeat, 14]

Sliding Window: Overfeat



1) Window positions + score maps

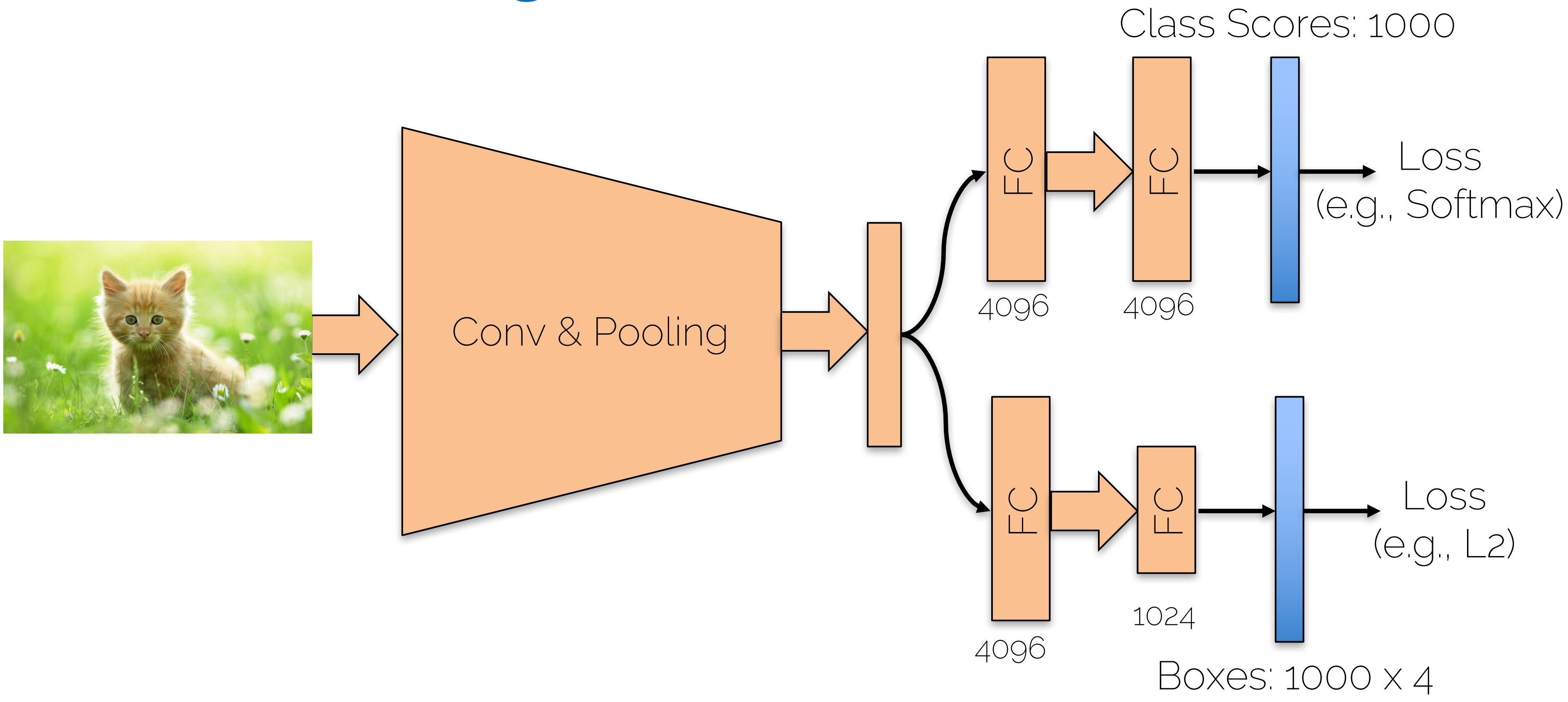


2) Box regression

3) Final bounding box prediction

[Sarmenet et al.: Overfeat, 14]

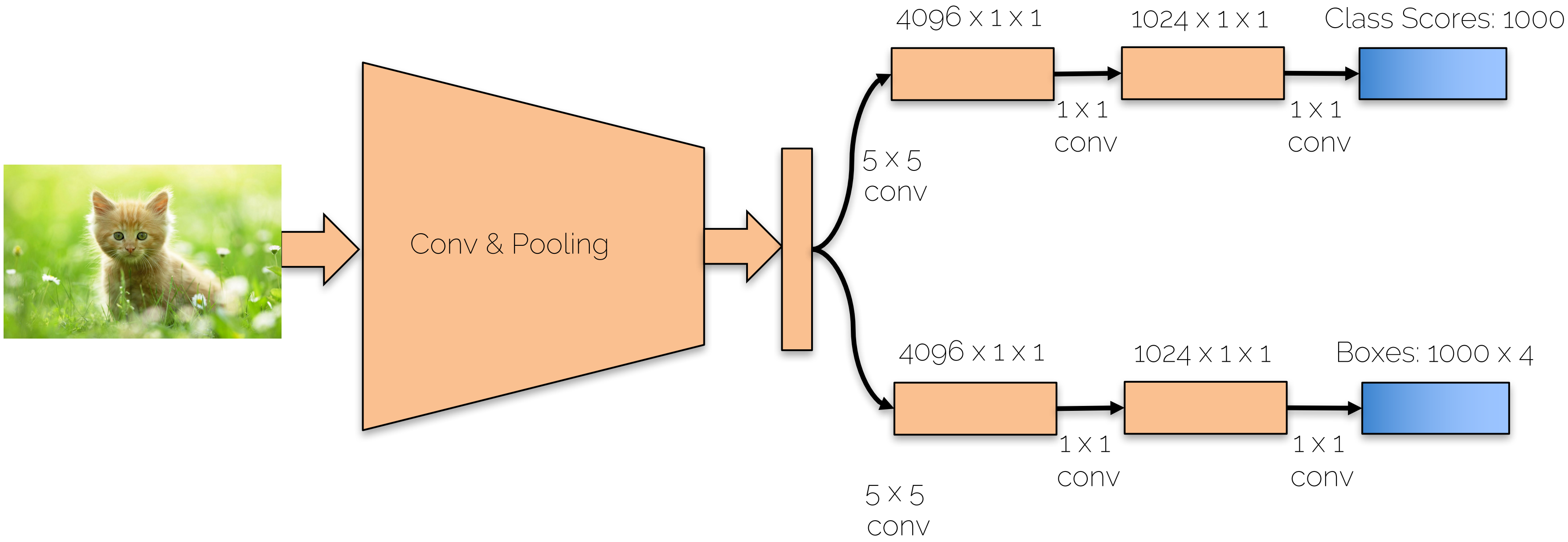
Sliding Window: Overfeat



[Sarmenet et al.: Overfeat, 14]

Sliding Window: Overfeat

Efficient sliding by converting FCs into convs



[Sarmenet et al.: Overfeat, 14]

Sliding Window: Overfeat

Convs are great in terms of compute (weight sharing!)

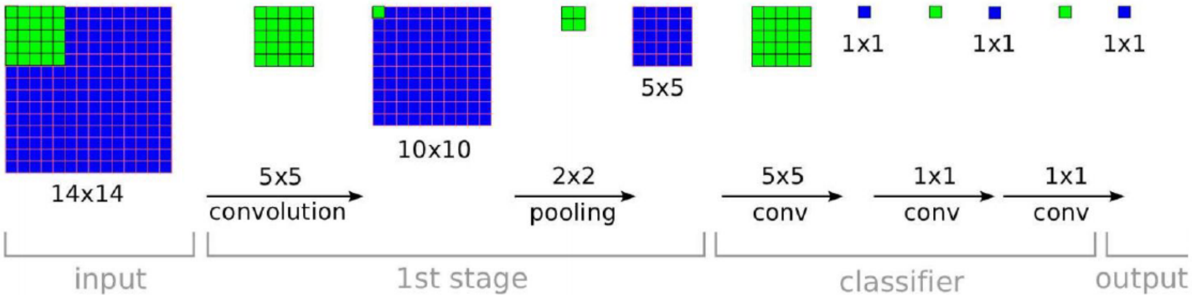
But what's the other main advantage?

[Sarmenet et al.: Overfeat, 14]

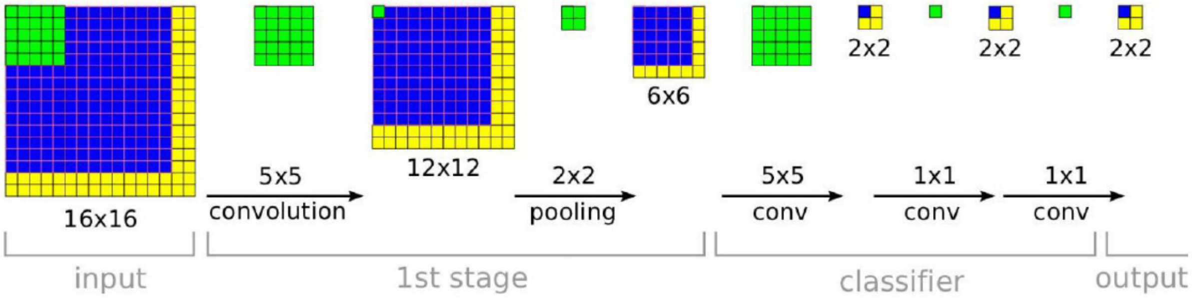
Sliding Window: Overfeat

Architecture is (somewhat) invariant to the image size

Training: 14x14 image
1 x 1 classifier output



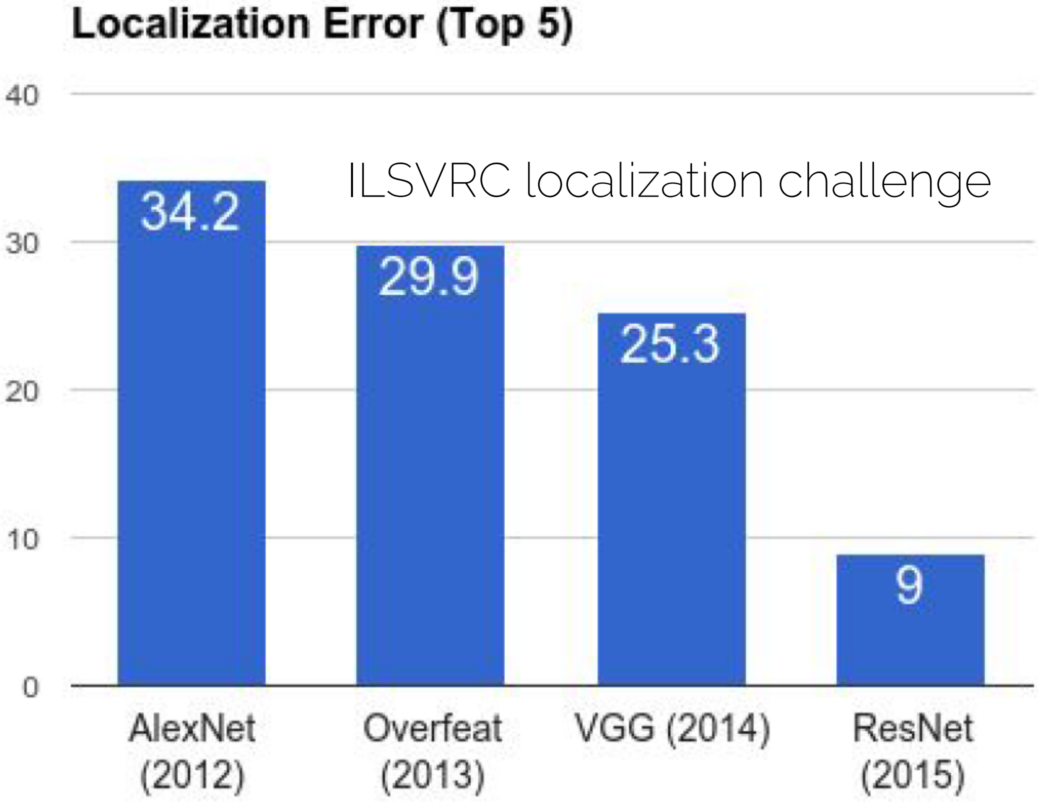
Testing: 2x2 image
2 x 2 classifier output



It needs to handle different box sizes!

[Sarmenet et al.: Overfeat, 14]

ImageNet Classification + Localization



Overfeat: Multiscale conv regression with box merging

VGG: Mostly the same, but better network (also fewer scales and location, gain by better features)

ResNet: Crazy network, and different localization method (region proposals, RPN)

Credit: Li/Karpathy/Johnson

Using CNNs in Computer Vision

Classification

CIFAR 10 +
"raw" CNN ☺

Classification
+ Localization

Regression and/or
sliding window

Object Detection

Instance
Segmentation

Credit: Li/Karpathy/Johnson

Using CNNs in Computer Vision

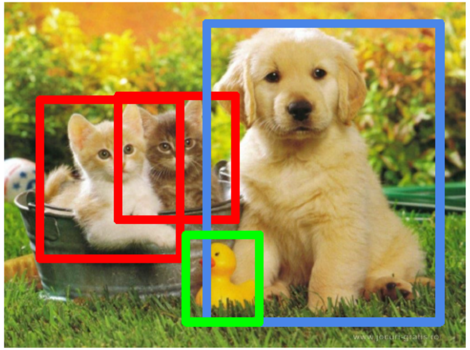
Classification

CIFAR 10 +
"raw" CNN ☺

Classification
+ Localization

Regression and/or
sliding window

Object Detection



Instance
Segmentation

Multiple objects!
(but we don't know how many)

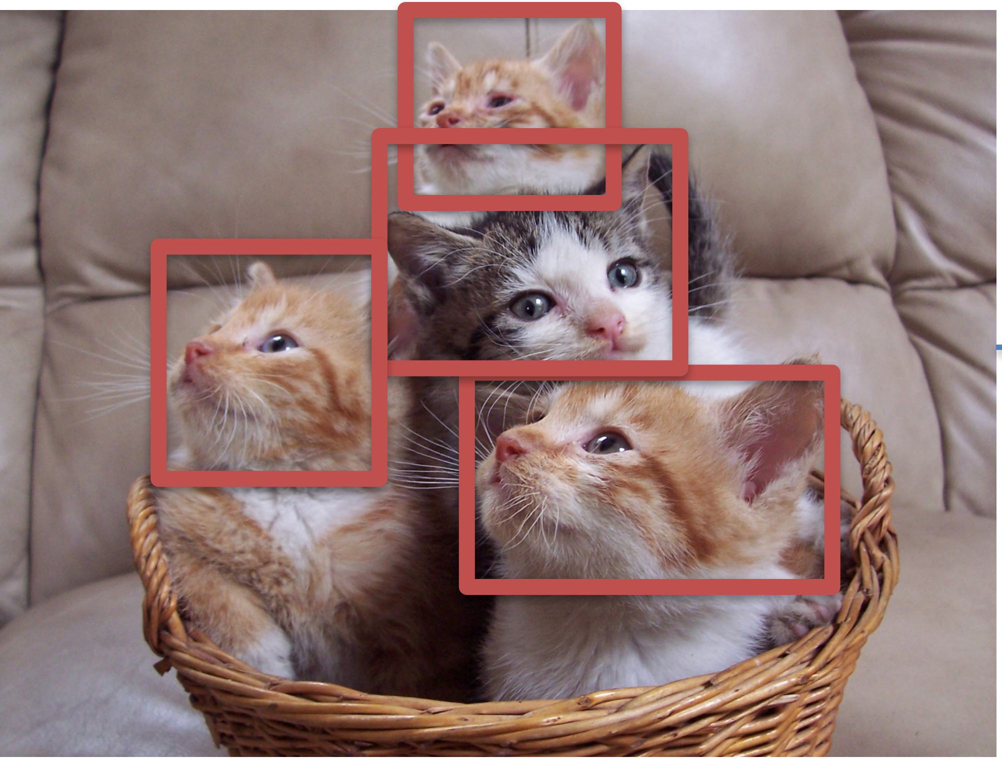
Credit: Li/Karpathy/Johnson

Object Detection as Regression?

Location (x, y, w, h) for car
Location (x, y, w, h) for motor bike

Regress 8 numbers
(distributed over 1 more multiple heads)

Object Detection as Regression?



Location (x, y, w, h) for cat 1
Location (x, y, w, h) for cat 2
Location (x, y, w, h) for cat 3
Location (x, y, w, h) for cat 4

Regress 16 numbers
(distributed over 1 more multiple heads)

Object Detection as Regression?

What now?

It is actually possible via regression (using RNNs -> more later)

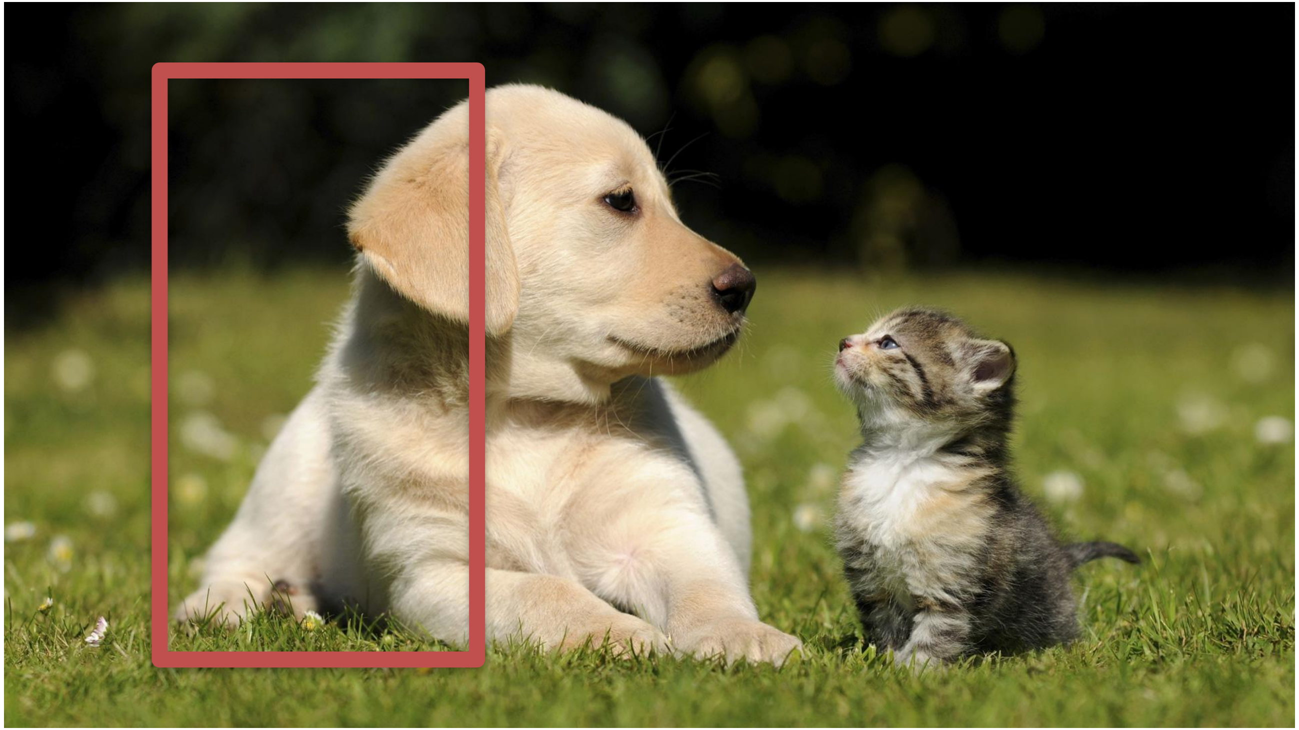
Object Detection as Classification

2 classes

Dog: no

Cat: no

Object Detection as Classification

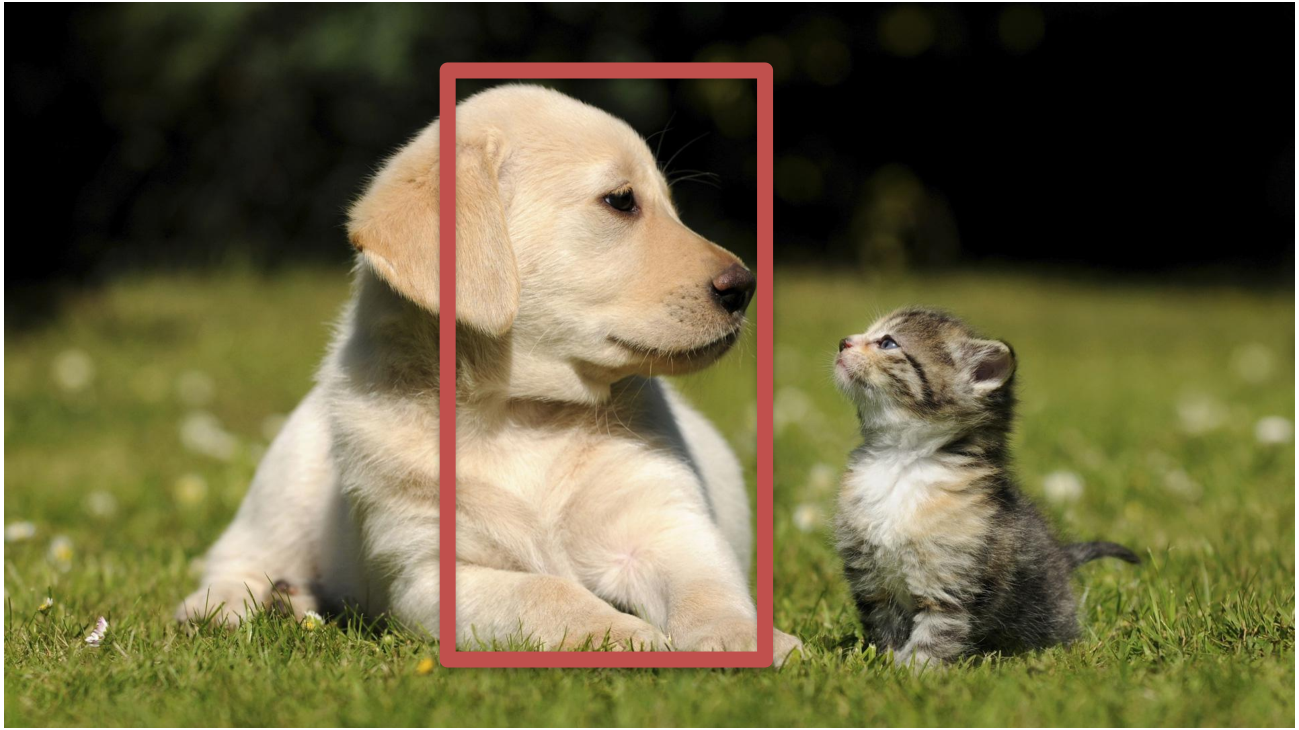


2 classes

Dog: maybe

Cat: no

Object Detection as Classification

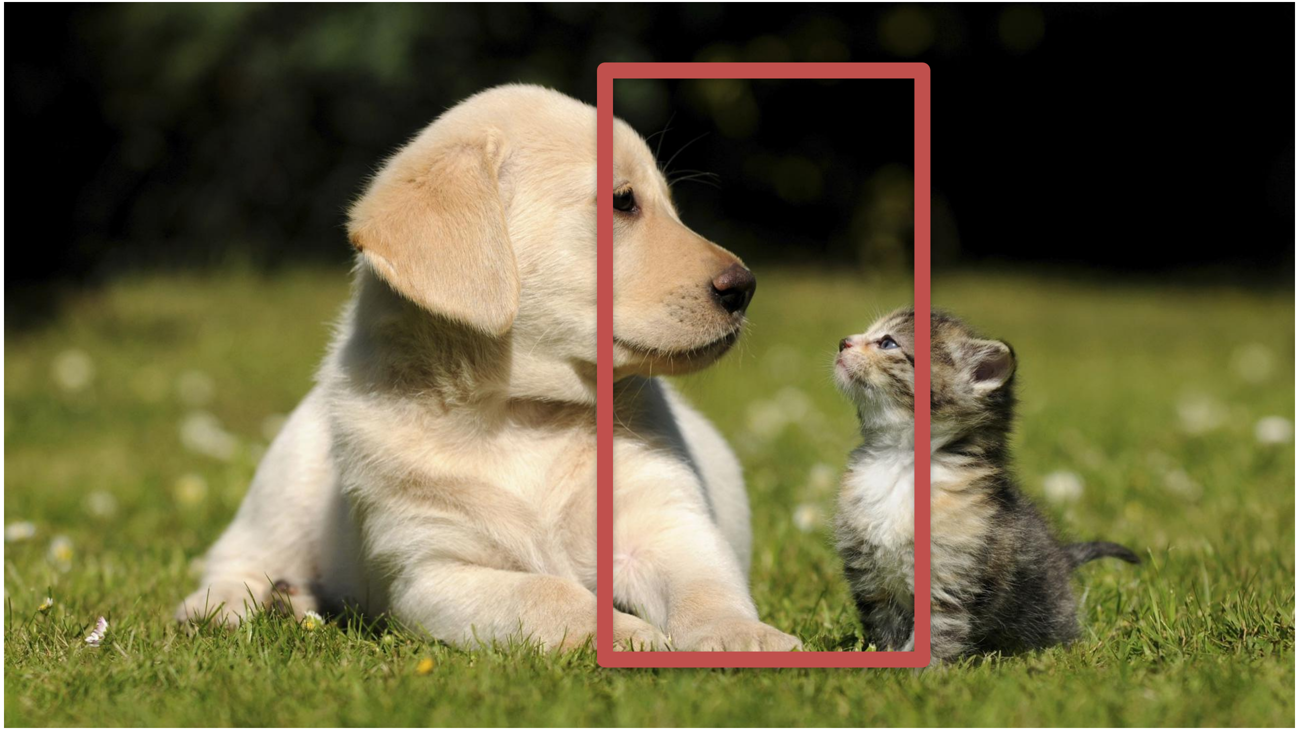


2 classes

Dog: yes

Cat: no

Object Detection as Classification

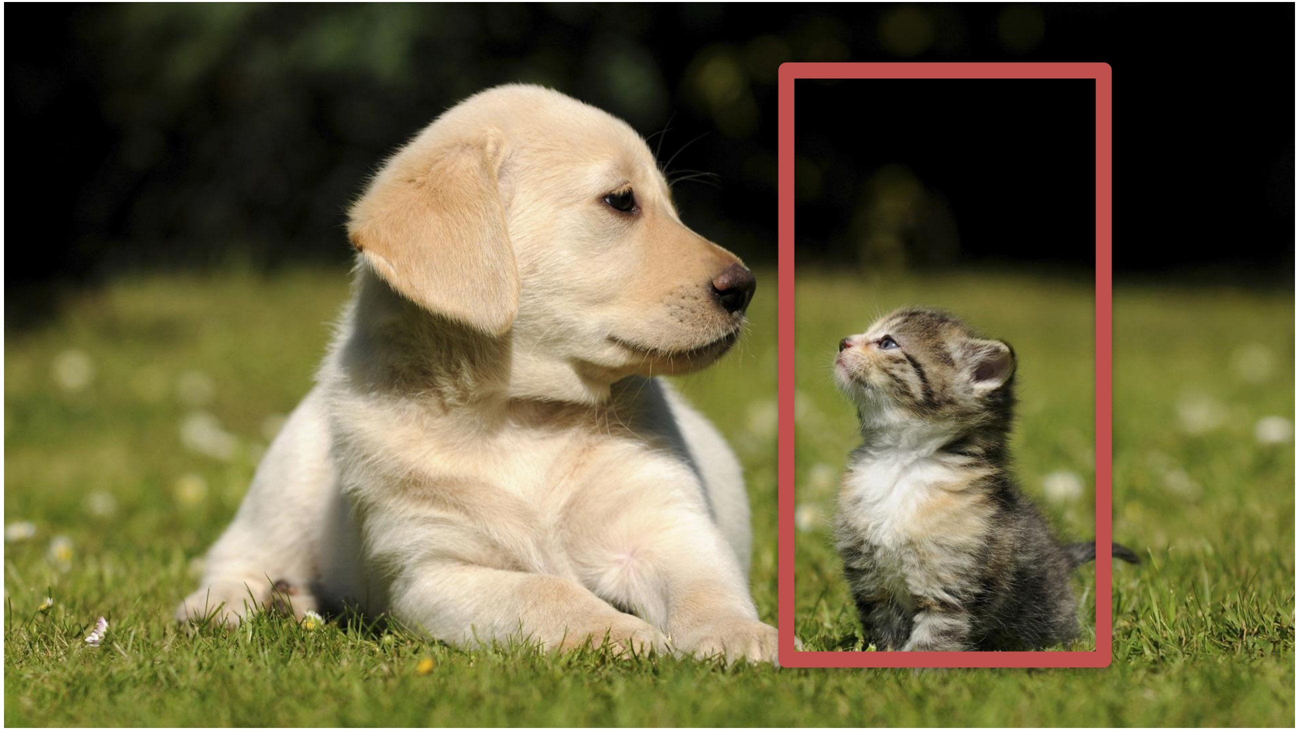


2 classes

Dog: maybe

Cat: maybe

Object Detection as Classification



2 classes

Dog: no

Cat: yes

Classification as Detection

- Problem: need to test at every position and scale
- Solutions
  - Just do it ☺ but it takes time at test
  - Smarter, but fewer, proposals
    - E.g., in videos you can use results from prev. frames
    - Train region proposals!

Region Proposals

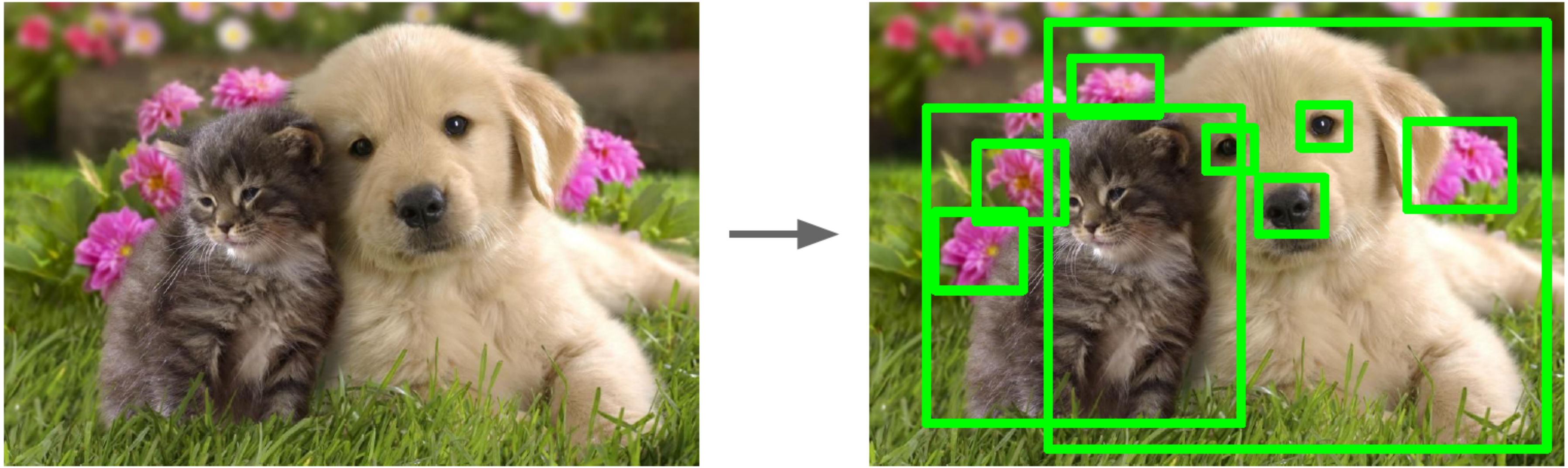
Main Idea:

- Running a CNN at every possible location is too costly

- Use a cheap proposal method

- Run ‘expensive’ CNN only at selected regions

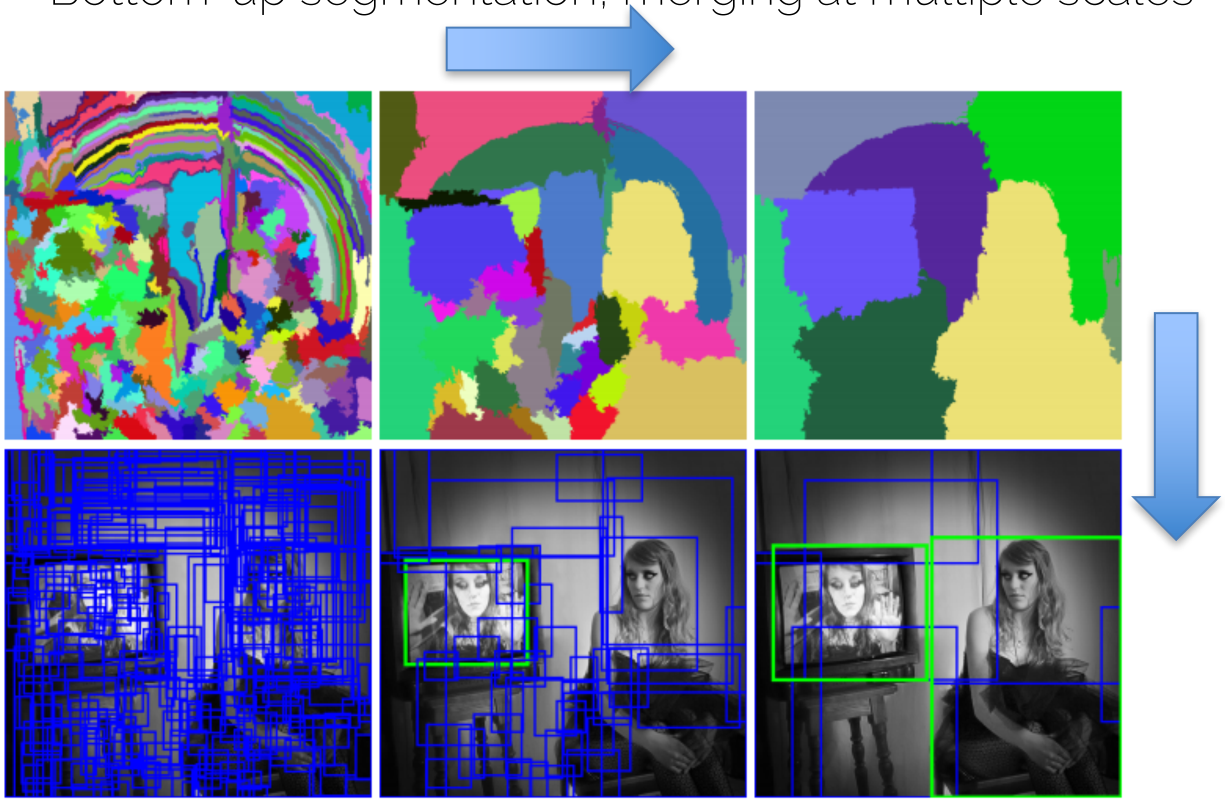
Region Proposals



Credit: Li/Karpathy/Johnson

Region Proposals: Selective Search

Bottom-up segmentation, merging at multiple scales



Convert regions to boxes

[Uijlings et al. 13, Selective Search for Object Recognition]

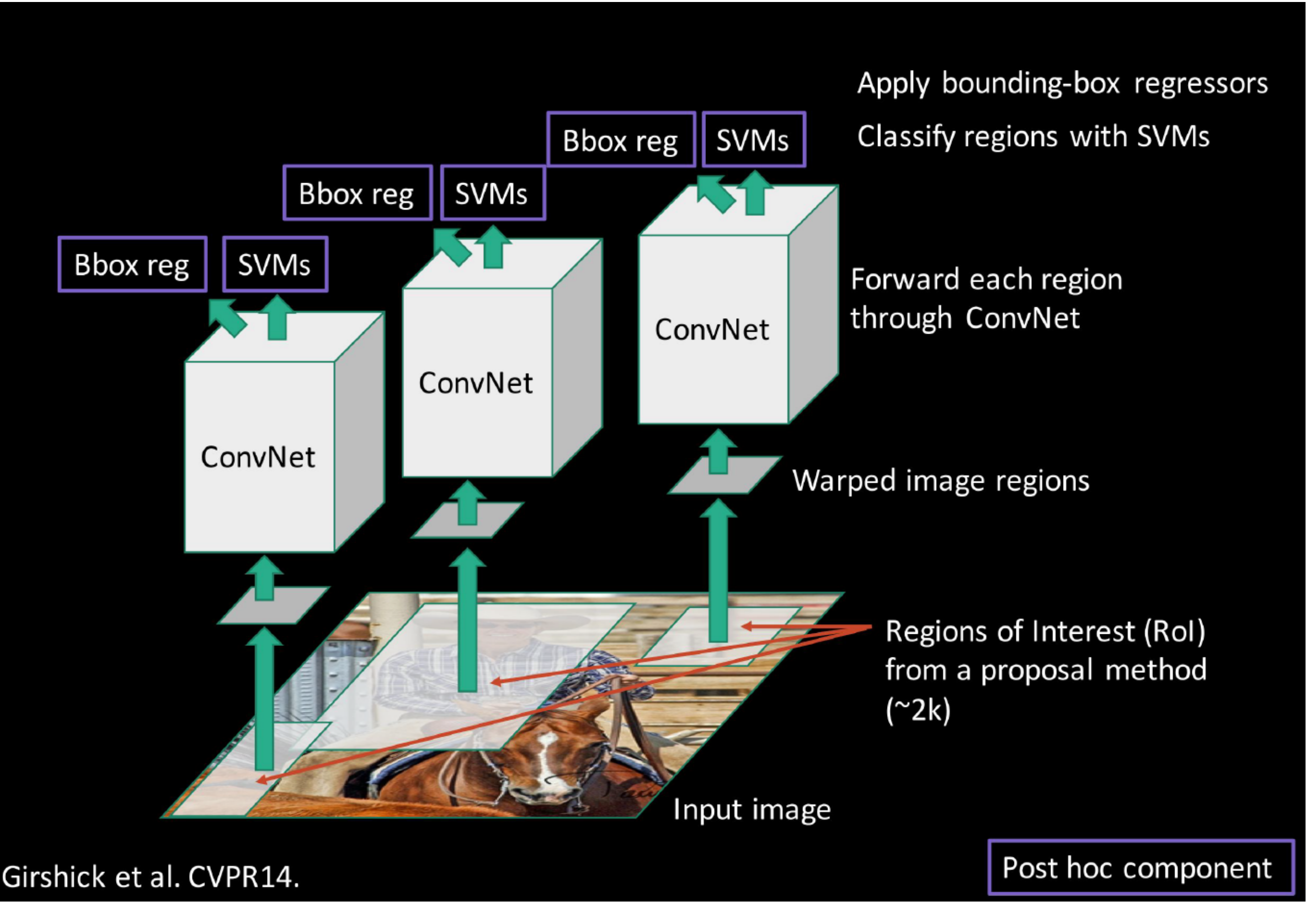
Region Proposals: Lots of Options

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	·	·	·

Most of them are not based on DL. Why ?

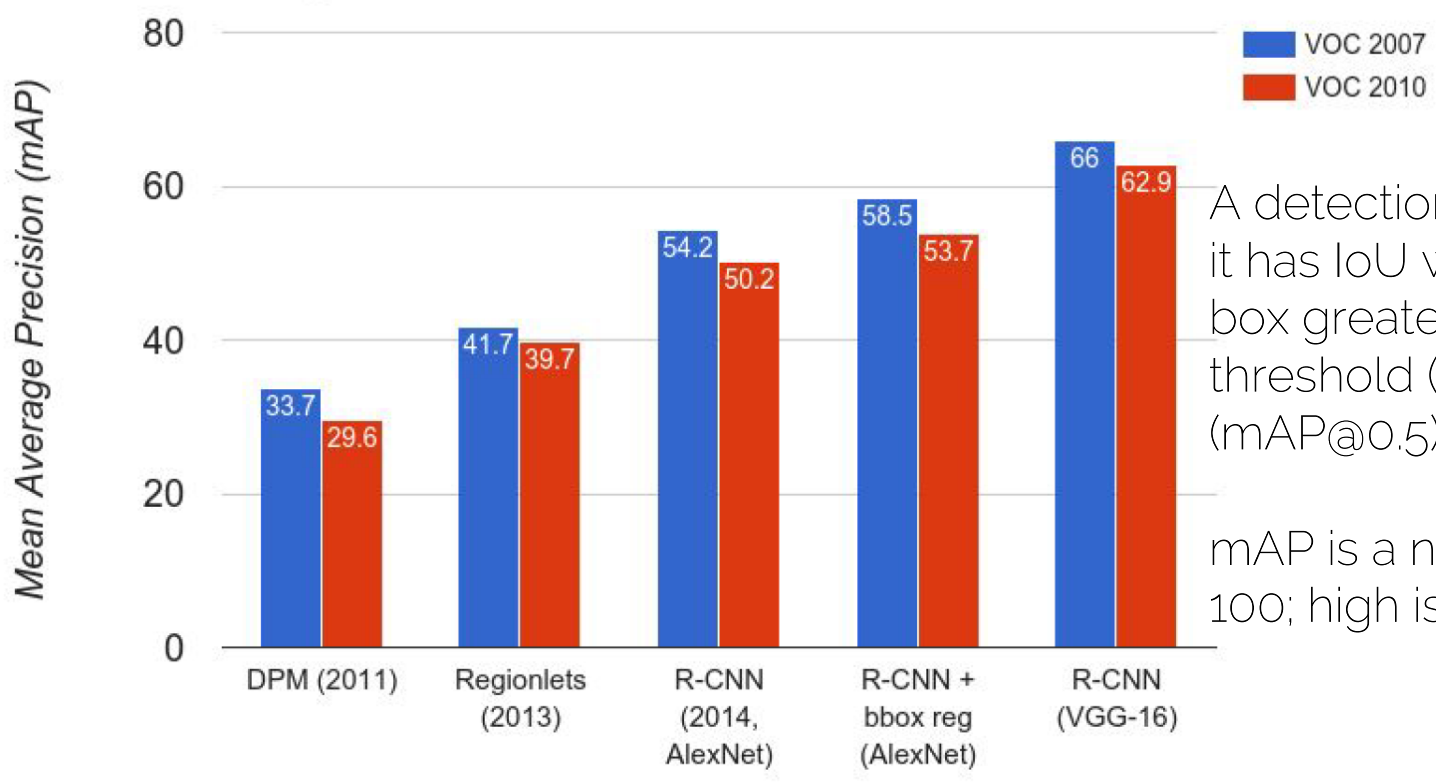
[Hosang et al. 15, Overview of object proposals]

Putting it Together: R-CNN



1) Run region proposal (e.g., selective search)
2) Warp (i.e., re-scale, re-size) to a fixed image size
3) This fixed output is fit into a CNN with class + regression head, which corrects for slightly off proposals

Putting it Together: R-CNN



A detection is a true positive if it has IoU with a ground-truth box greater than some threshold (usually 0.5) (mAP@0.5)

mAP is a number from 0 to 100; high is good

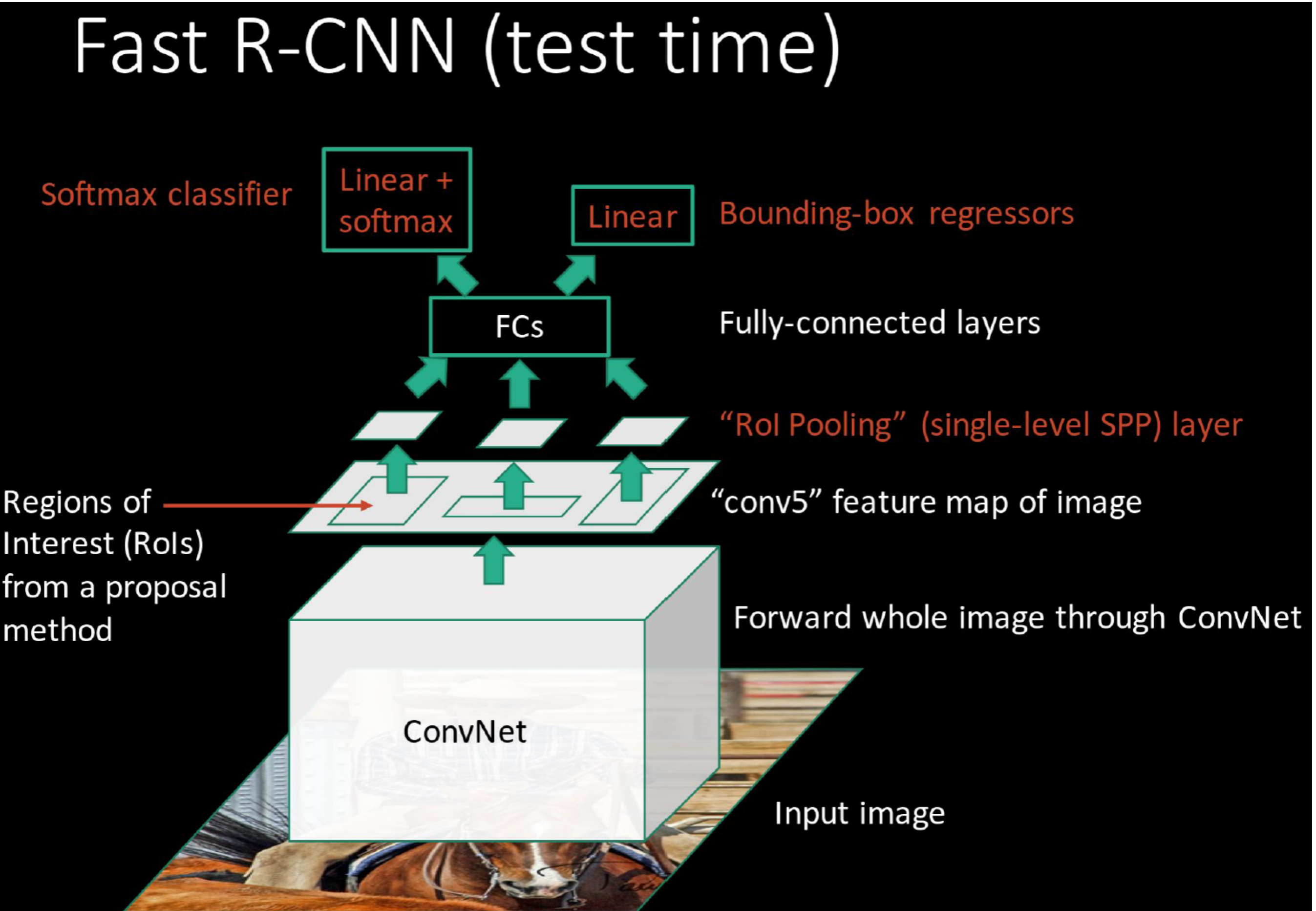
[Wang et al. 13, "Regionlets for Generic Object Detection"]

R-CNN: Training

- Unfortunately, training is fairly complex
  - Series of pre-training and fine-tuning tasks for classification, detection, etc.
  - Extraction of intermediate features that are cached that are pretty big for SVM classification (also space issue)
  - SVMs are not jointly trained with CNN
  - It's also extremely slow to train

[Girshick 14, R-CNN]

Fast R-CNN (testing)

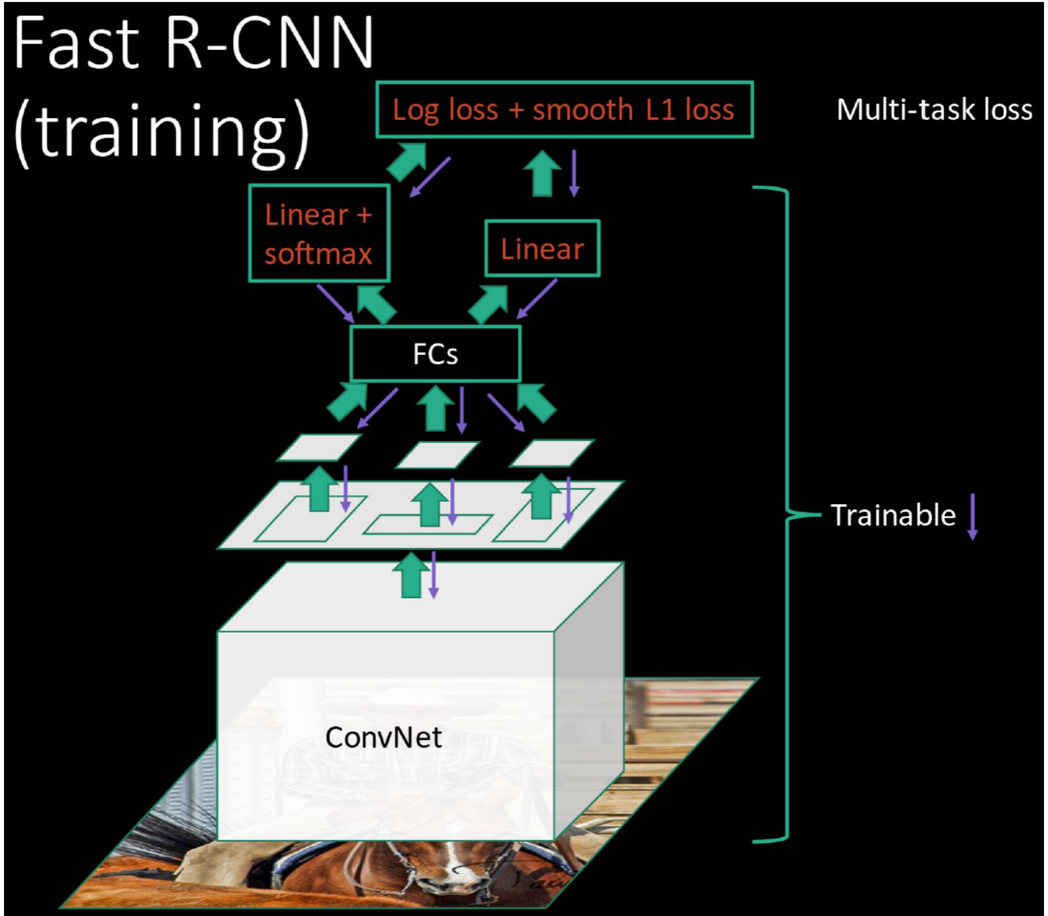


Solves test-time issue due to independent CNN forward passes

-> now one pass that shares computation of conv layers between proposals with in an image

[Girshick 15, Fast R-CNN]

Fast R-CNN (training)



Solves training time issue: 1) CNN not updated with SVM losses. 2) Complex training pipeline

-> Just train whole thing end-to-end

[Girshick 15, Fast R-CNN]

Fast R-CNN Results

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

Using VGG-16 CNN on Pascal VOC 2007 dataset

Credit: Li/Karpathy/Johnson

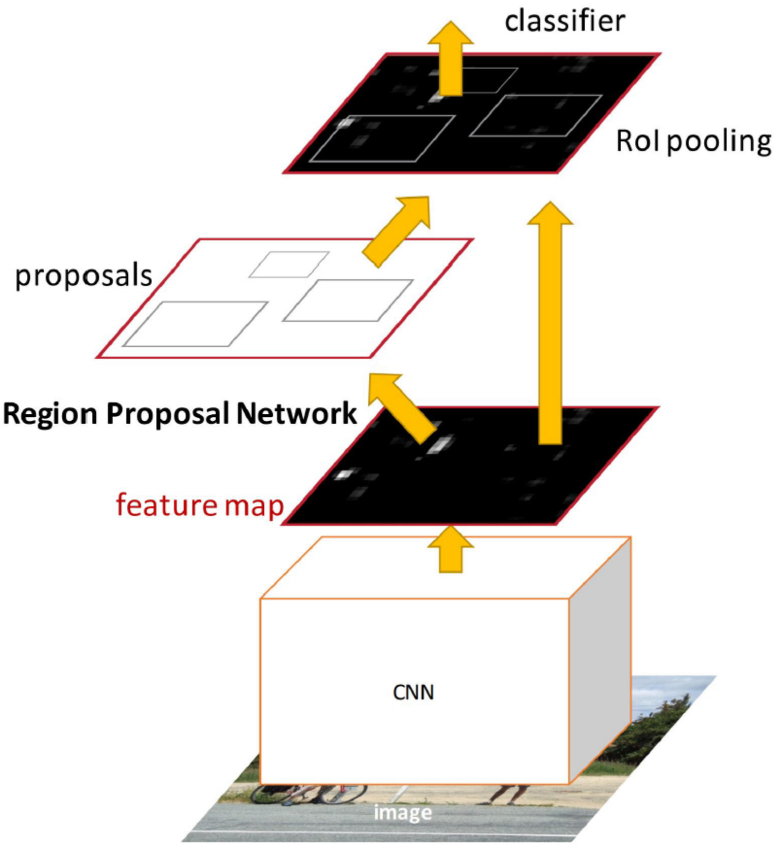
Fast R-CNN

- Issue: Test-time speeds don't include object 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**

Credit: Li/Karpathy/Johnson

Faster R-CNN



Solution: make the CNN also do region proposals!

Insert a Region Proposal Network (RPN) after last conv layer

RPN produces region proposals (one shot) -> no need for external proposals

After RPN, region of interest pooling, and use similar classifier and bbox regressor like Fast R-CNN

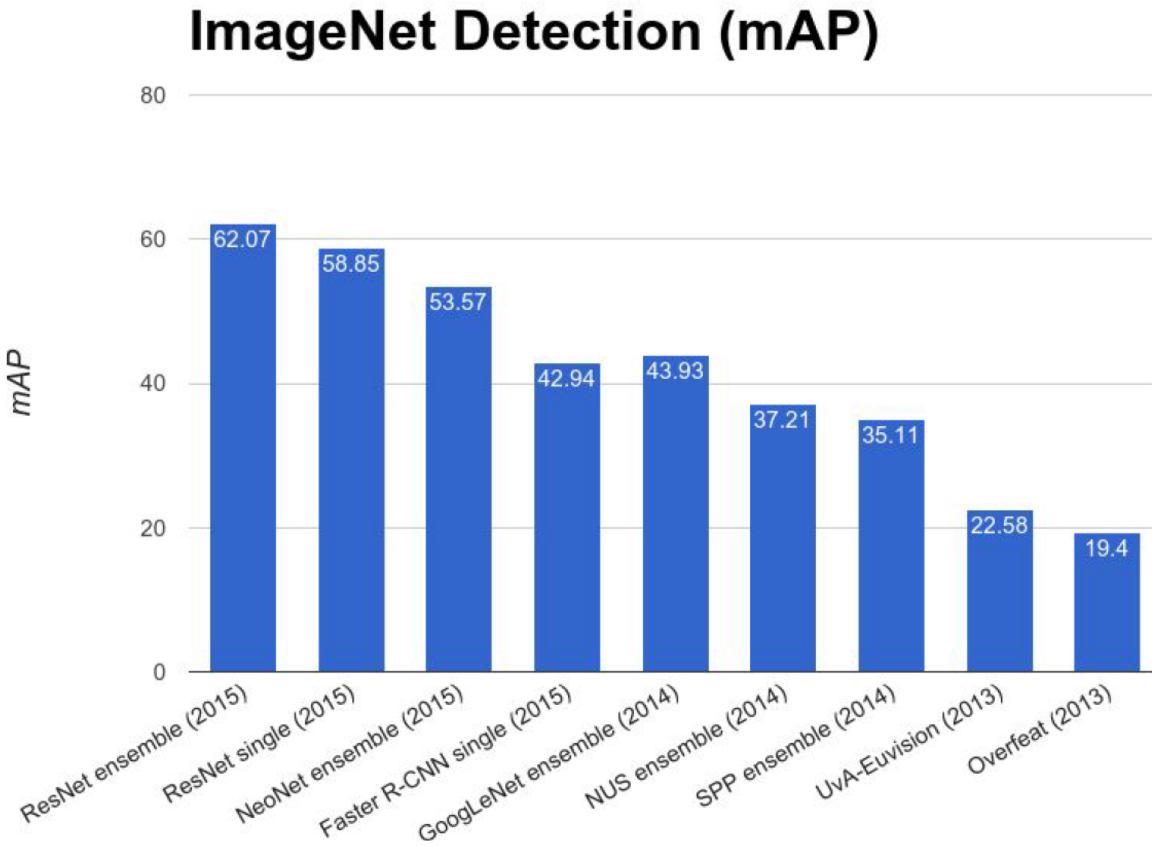
[Girshick 15, Faster R-CNN]

Faster R-CNN

	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**

Credit: Li/Karpathy/Johnson

ImageNet Detection 2013 - 2015



Credit: Li/Karpathy/Johnson

Using CNNs in Computer Vision

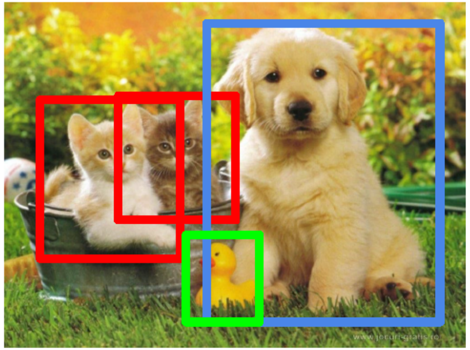
Classification

CIFAR 10 +
"raw" CNN ☺

Classification + Localization

Regression and/or
sliding window

Object Detection



Selective Search, RP
(Fast(er)) R-CNN

Instance Segmentation

Credit: Li/Karpathy/Johnson

Using CNNs in Computer Vision

Classification

CIFAR 10 +
"raw" CNN ☺

Classification + Localization

Regression and/or
sliding window

Object Detection

Selective Search, RP
(Fast(er)) R-CNN

Instance Segmentation

Next Lecture ☺

Credit: Li/Karpathy/Johnson

Using CNNs in Computer Vision

- We have CNNs (Convs, Pooling, FCs, Losses)
- We can employ them for classification
- We can employ them for regression

- Somewhat oversimplified: the “rest” is smart architectures and application of these tools
  - > of course it’s more complicated ☺

Important Datasets to Know

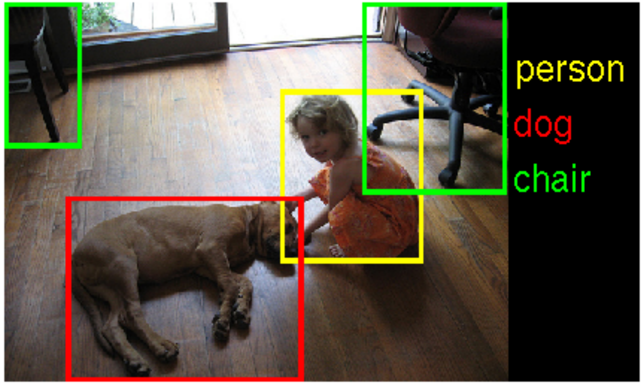
CIFAR-10: single object, centered, Krizhevsky et al.

MNIST: handwritten digits, LeCun et al.

Pascal VOC, 20 classes, 10k images, Everingham et al.

ImageNet: 10 mio images, Deng et al.

MSCoco, 300k images, Lin et al. 15



Administrative Things

- Thursday June 29th: More about CNN Architectures – lots of cool stuff: e.g., Dense Pixel Classification!
- Tomorrow: Project proposals start!
  - Example proposals
  - Guidelines: how much is doable?
- Dating pool: Finalize forming teams (e.g., on Moodle!)

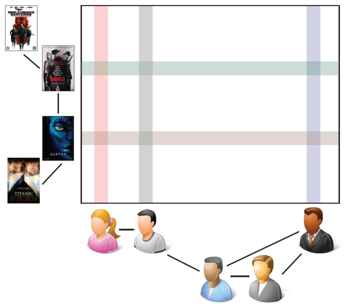
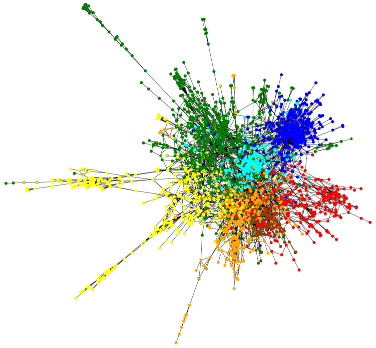
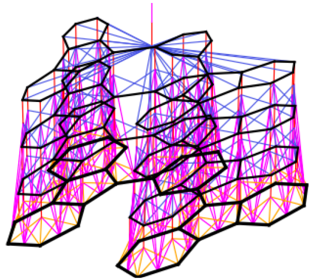
Special Course:

Geometric deep learning on graphs and manifolds

Going beyond Euclidean data

Michael Bronstein

USI Lugano / Tel Aviv University / Intel Perceptual Computing / TUM IAS

Preliminary: scheduled for Fri 30/6 and 7/7 (2pm to 4pm)
-> in our tutorial room