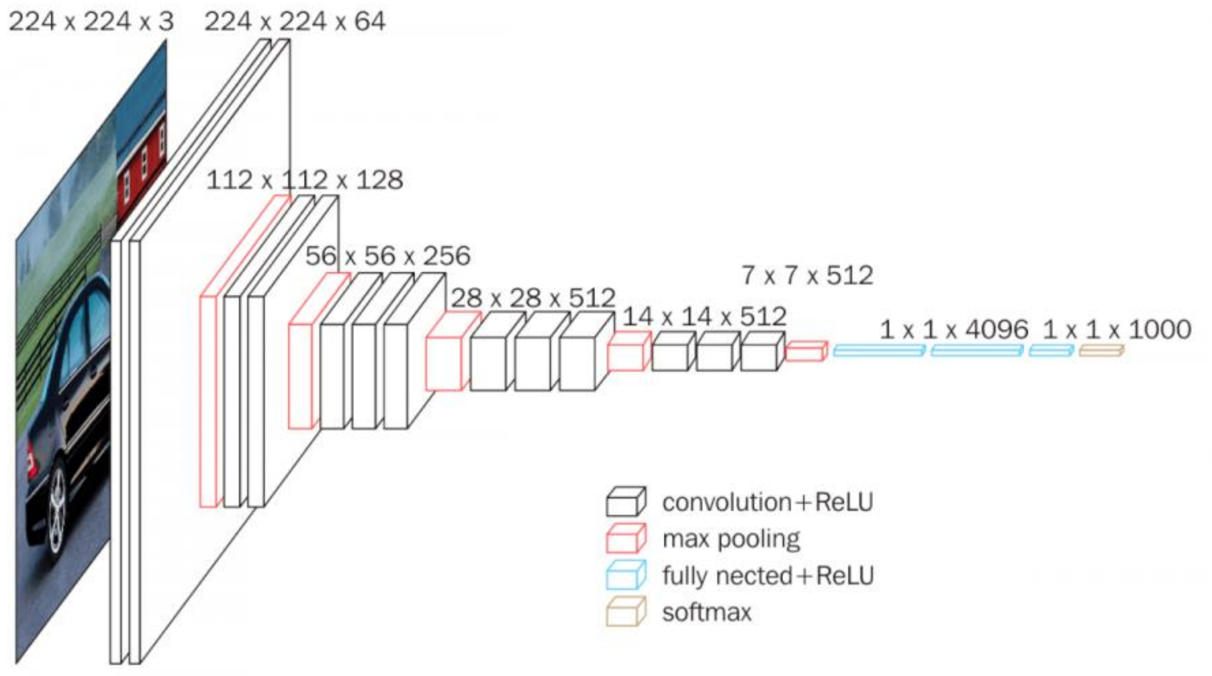
Computer Vision

Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley

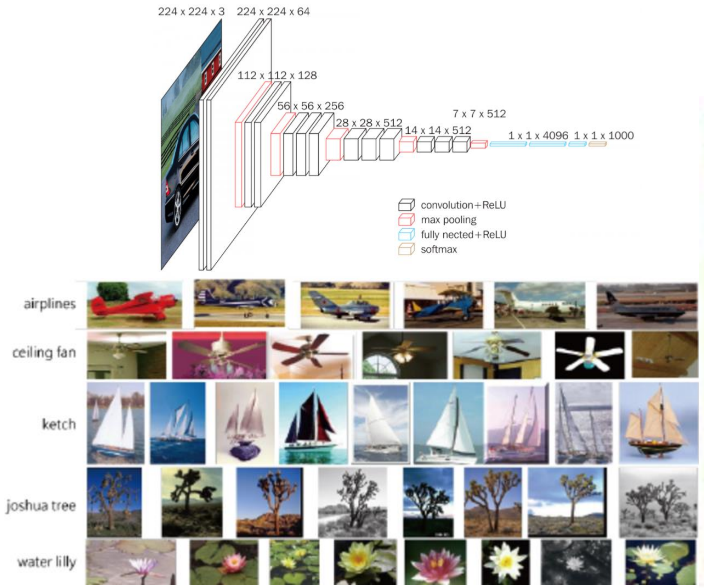
So far...



convolutional networks: map **image** to **output value**

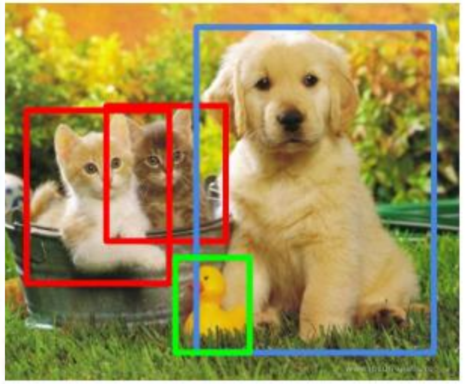
e.g., semantic category ("bicycle")

Standard computer vision problems

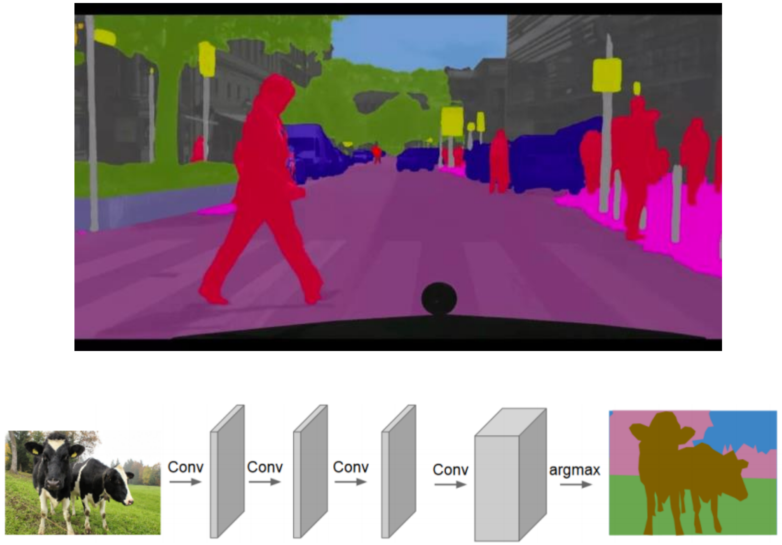


object classification

object localization



object detection



semantic segmentation
a.k.a. scene understanding

Object localization setup

Before: $\mathcal{D} = \{(x_i, y_i)\}$

image

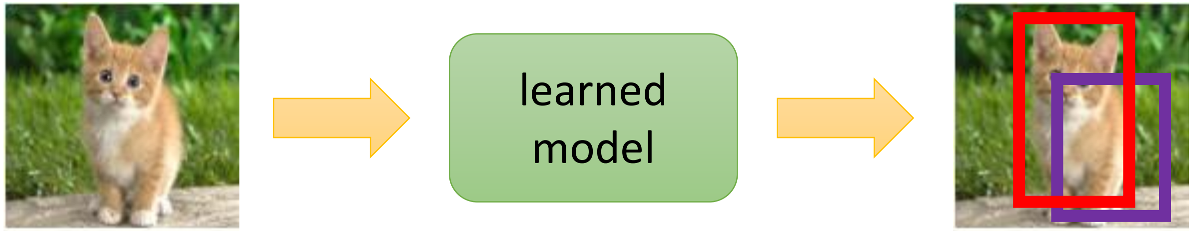
class label (categorical)

Now: $\mathcal{D} = \{(x_i, y_i)\}$

image

$y_i = (\ell_i, x_i, y_i, w_i, h_i)$

Measuring localization accuracy

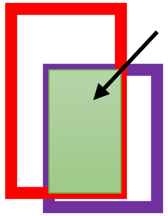


(x, y, w, h) ← predicted bounding box

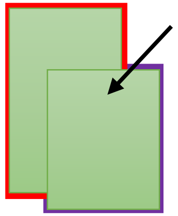
"cat": 0.64 ← prediction score (e.g., probability)

Did we get it right?

Intersection over Union (IoU)



intersection area (**I**)



union area (**U**)

IoU = I / U

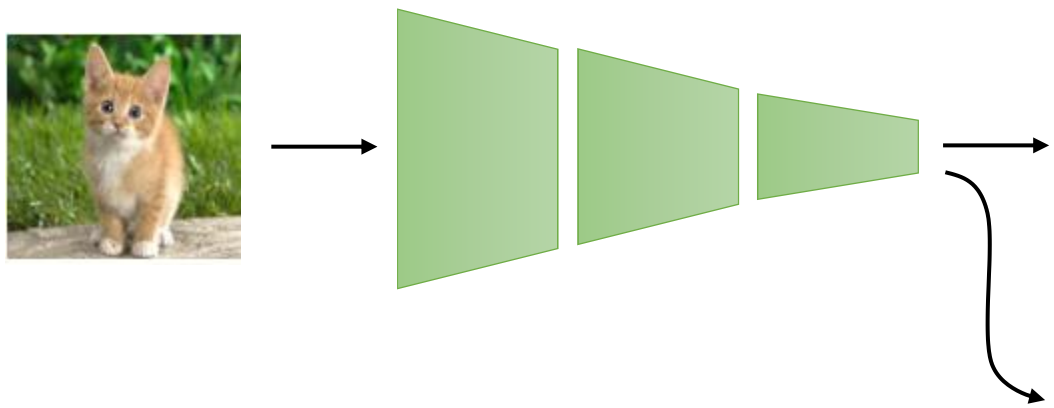
Different datasets have different protocols, but one reasonable one is: **correct if IoU > 0.5**

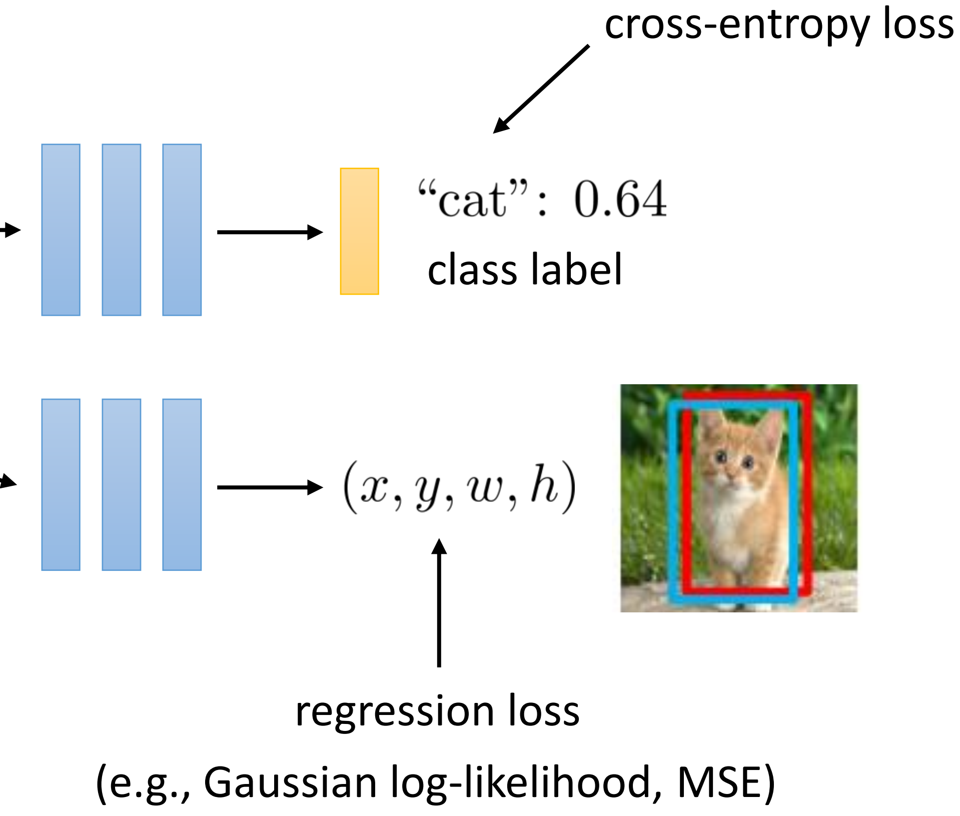
If also outputting class label (usually the case): **correct if IoU > 0.5 and class is correct**

This is **not** a loss function! Just an evaluation standard

Object localization as regression

$\mathcal{D} = \{(x_i, y_i)\}$ $\qquad y_i = (\ell_i, x_i, y_i, w_i, h_i)$

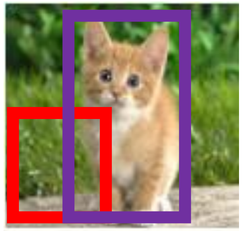




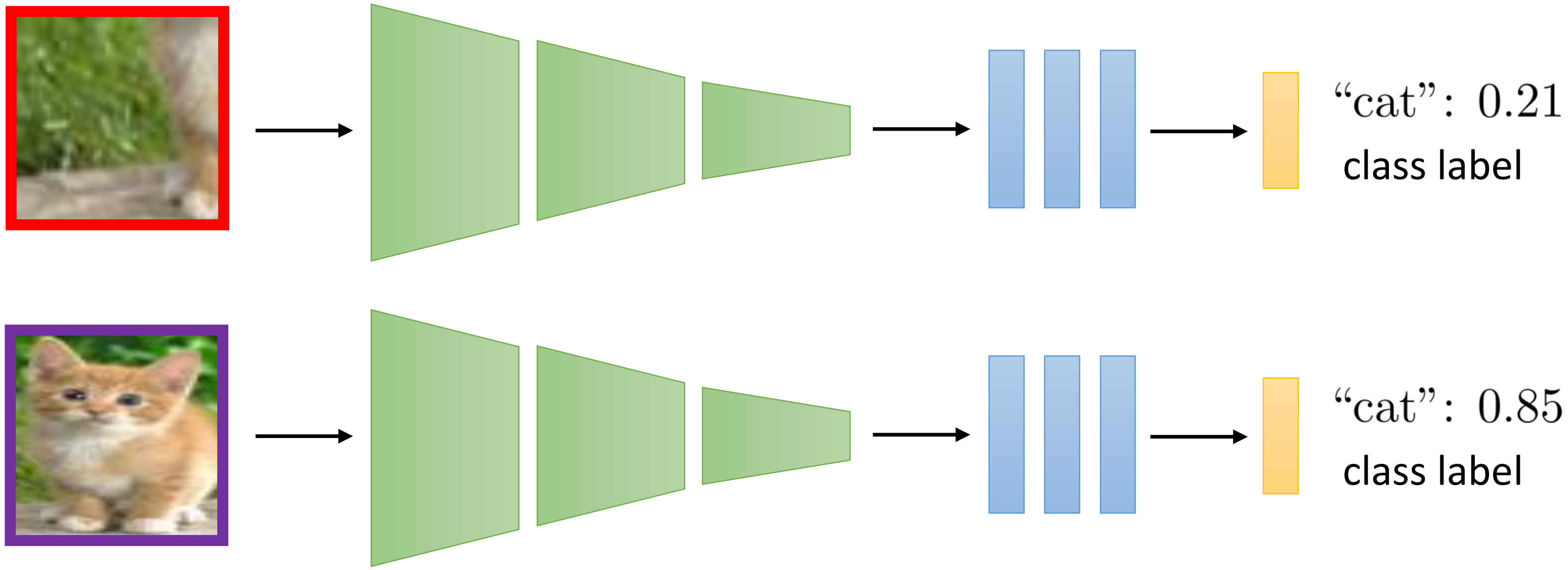
- Very simple design
- Can either train jointly (multi-task), or train with classification first, then train regression head
- More or less works OK
- By itself, this is **not** the way it’s usually done!
  - We’ll see why shortly

Sliding windows

$$\mathcal{D} = \{(x_i, y_i)\} \qquad y_i = (\ell_i, x_i, y_i, w_i, h_i)$$



What if we classify **every** patch in the image?

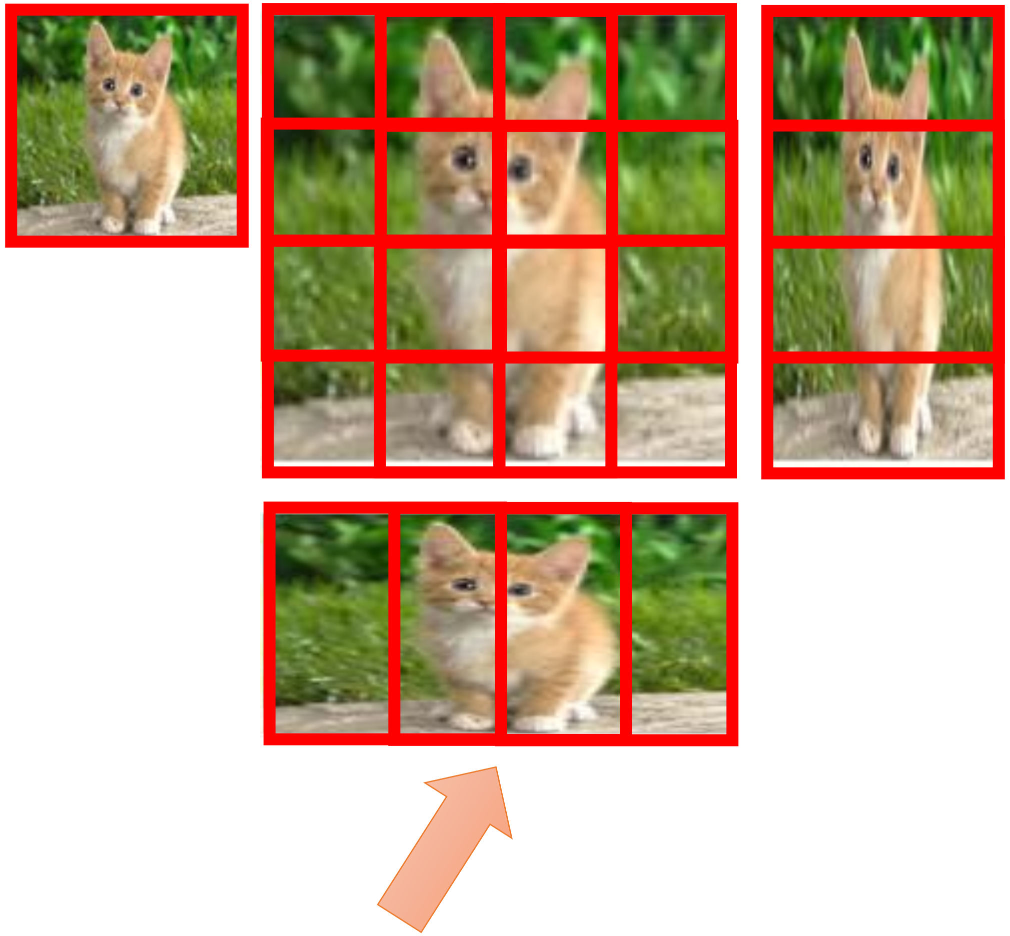


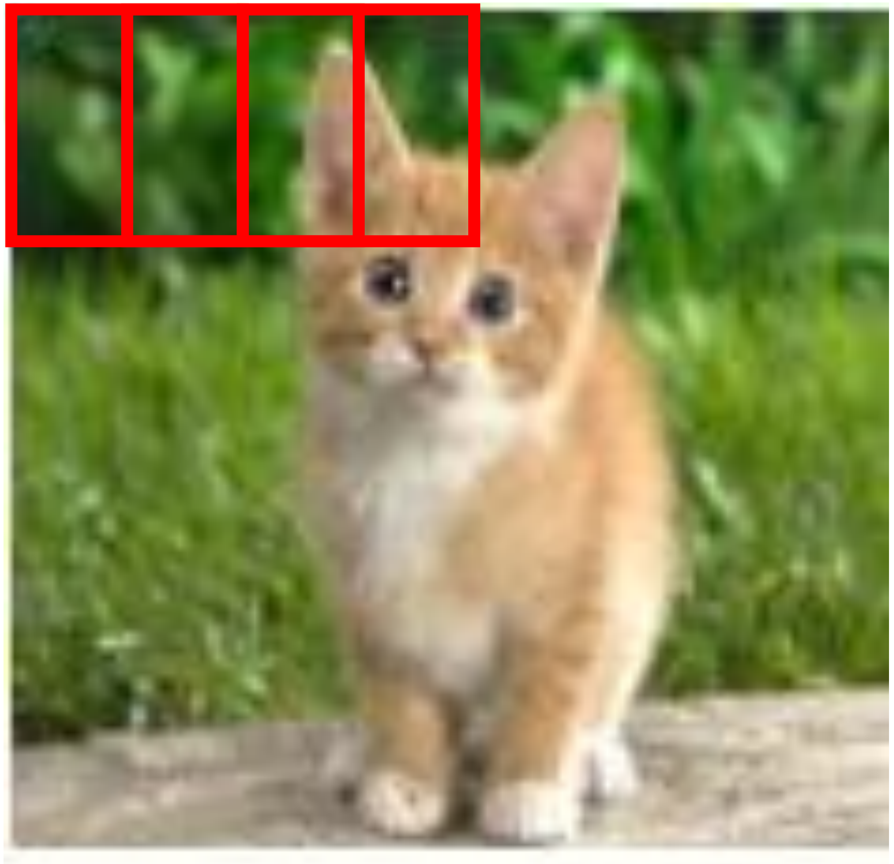
Sliding windows

$$\mathcal{D} = \{(x_i, y_i)\} \qquad y_i = (\ell_i, x_i, y_i, w_i, h_i)$$

could just take the box with the **highest** class probability

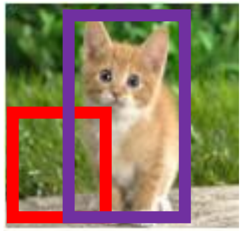
more generally: **non-maximal suppression**

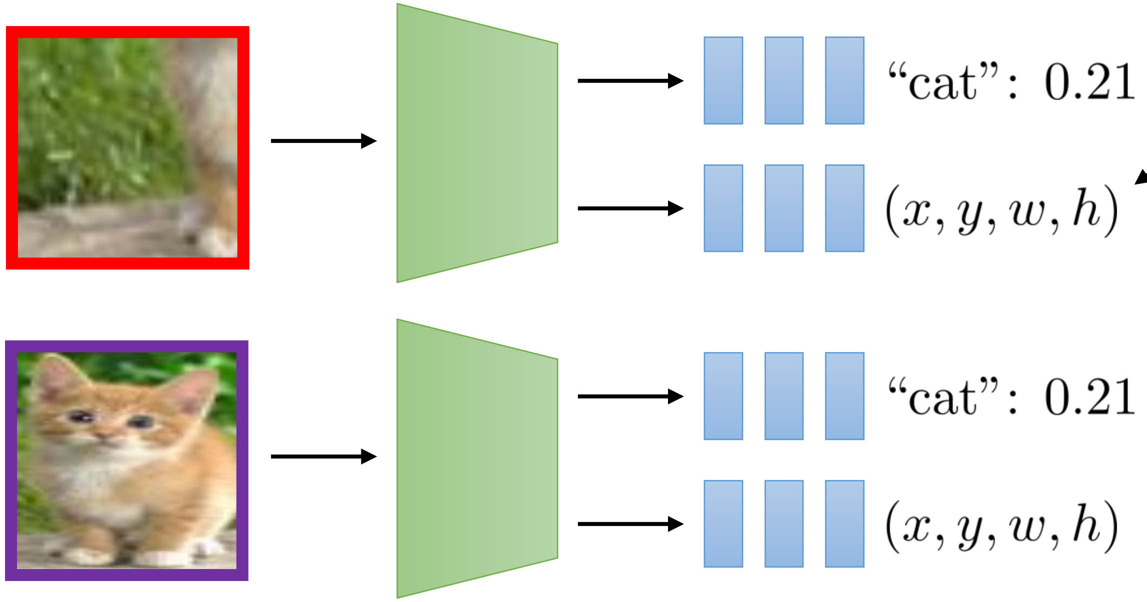




A practical approach: OverFeat

$$\mathcal{D} = \{(x_i, y_i)\} \qquad y_i = (\ell_i, x_i, y_i, w_i, h_i)$$



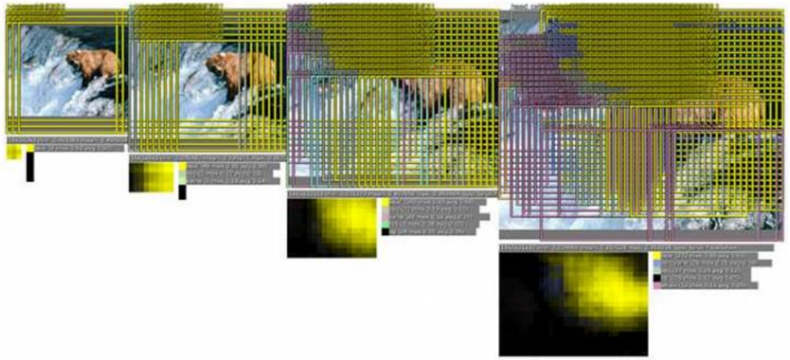


provides a little "correction" to sliding window

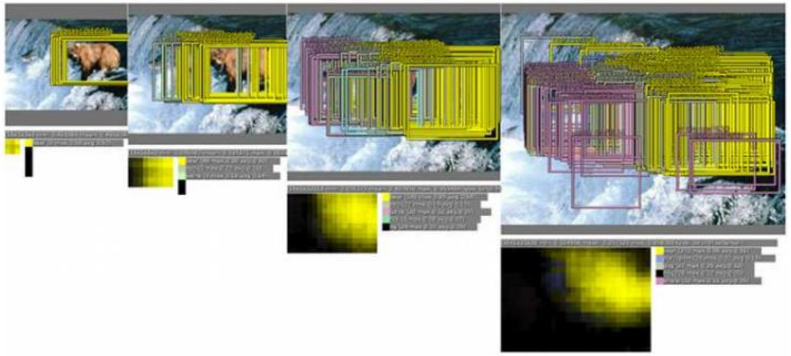
- Pretrain on **just** classification
- Train regression head on top of classification features
- Pass over different regions at different scales
- "Average" together the boxes to get a single answer

Sermanet et al. "**OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks.**" 2013

A practical approach: OverFeat



Sliding window **classification** outputs at each scale/position (yellow = bear)



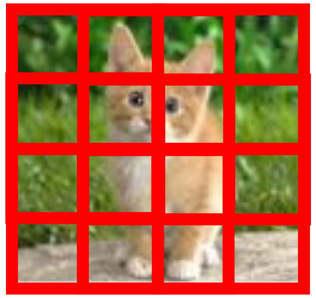
Predicted box x, y, w, h at each scale/position (yellow = bear)

Final combined bounding box prediction (yellow = bear)

Sermanet et al. "**OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks.**" 2013

Sliding windows & reusing calculations

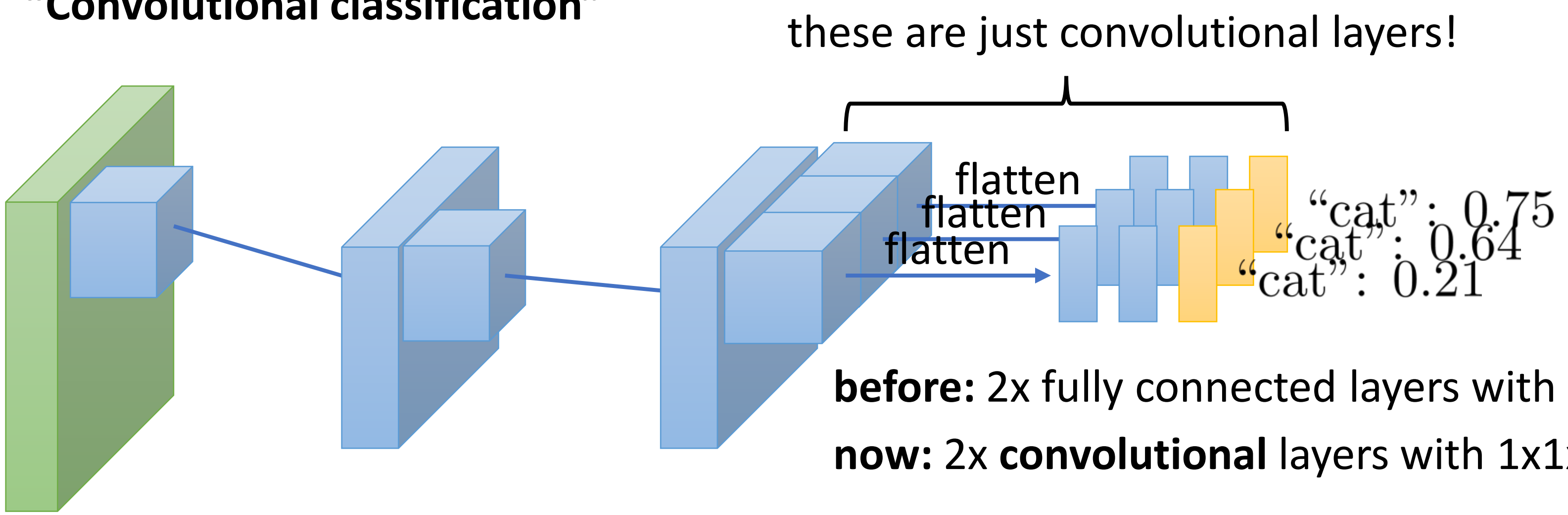
Problem: sliding window is very expensive! (36 windows = 36x the compute cost)



This looks **a lot** like convolution…

Can we just **reuse** calculations across windows?

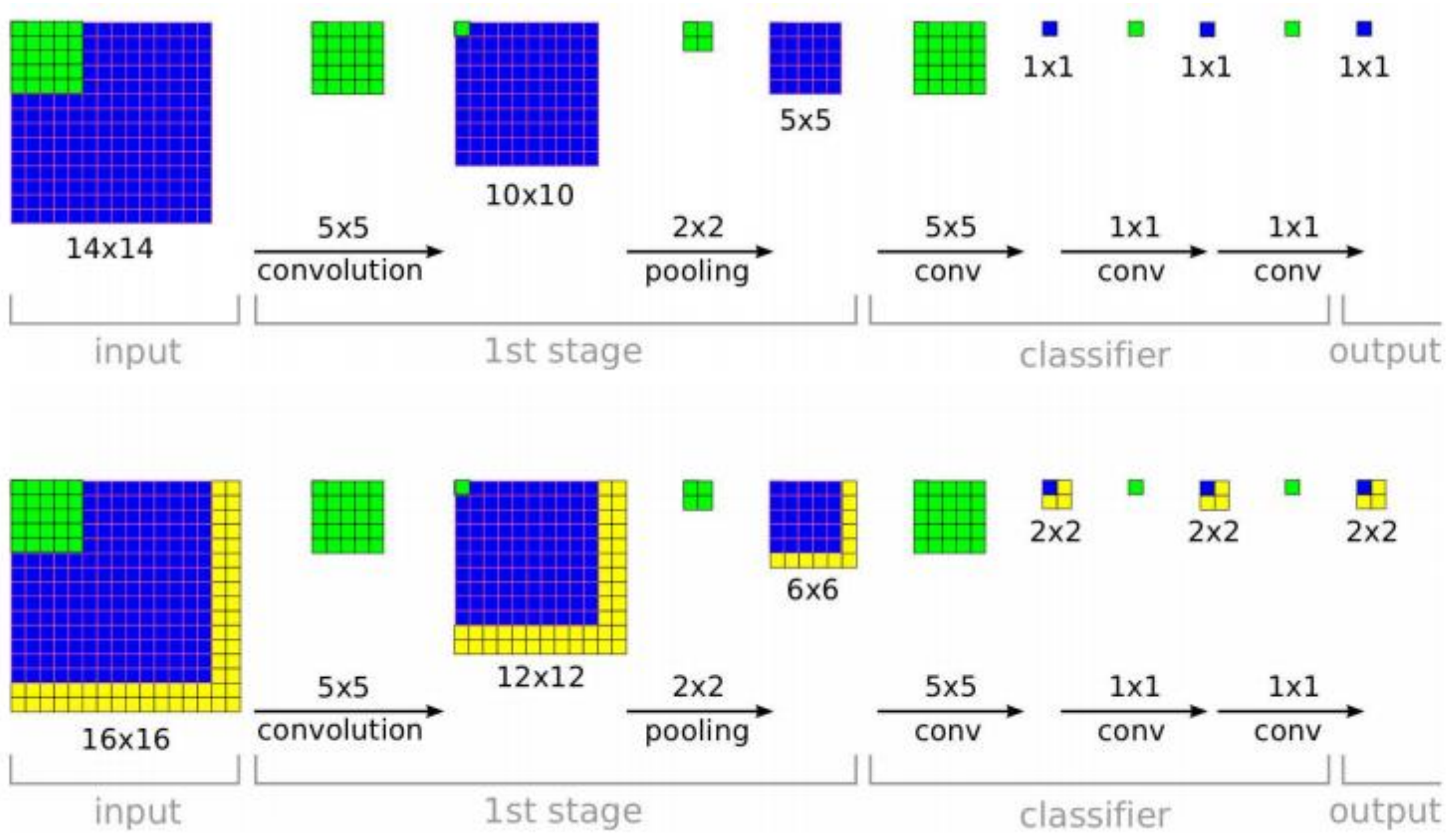
"Convolutional classification"



before: 2x fully connected layers with 4096 units

now: 2x **convolutional** layers with 1x1x4096 filters

Sliding windows & reusing calculations



This kind of calculation reuse is extremely powerful for localization problems with conv nets

We'll see variants of this idea in every method we'll cover today!

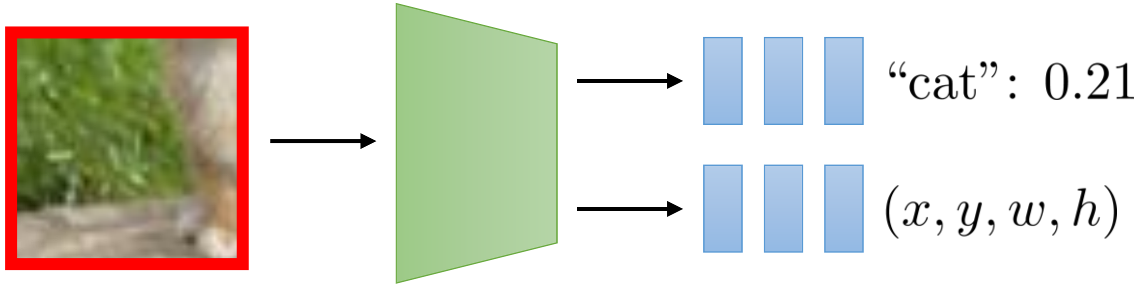
Sermanet et al. "**OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks.**" 2013

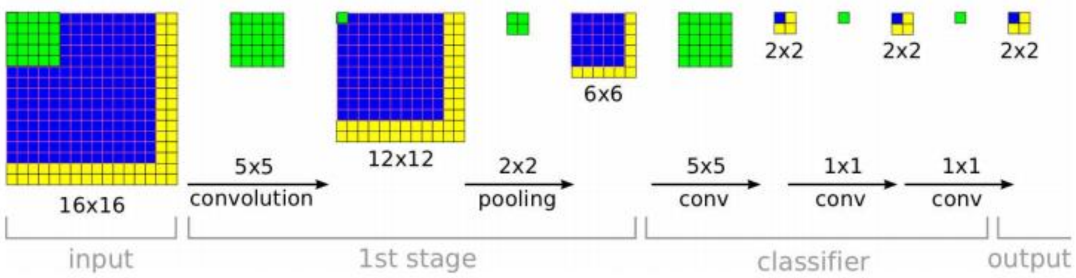
Summary

Building block: conv net that outputs class and bounding box coordinates

Evaluate this network at multiple scales and for many different crops, each one producing a probability and bounding box

Implement the sliding window as just another convolution, with 1x1 convolutions for the classifier/regressor at the end, to save on computation



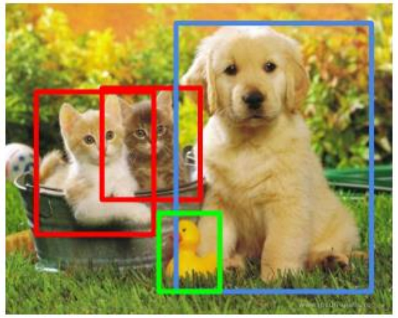


Object detection architectures

The problem setup

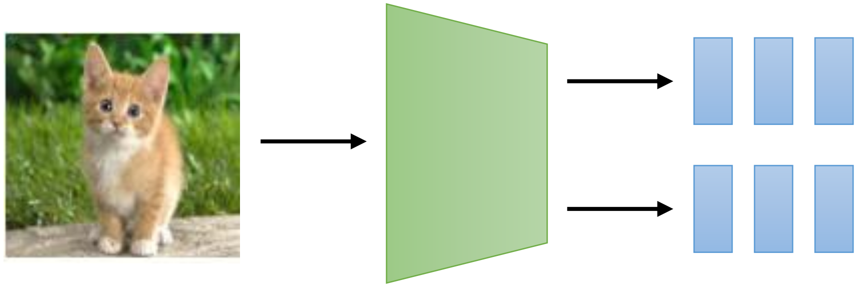
Before

Now



$(x_i,$

number of objects n_i different for each image x_i!



“cat”: 0.21

(x, y, w, h) **???**

How do we get multiple outputs?

Sliding window: each window can be a different object

Instead of selecting the window with the highest probability (or merging windows), just output an object in each window above some threshold

Big problem: a high-scoring window probably has **other** high-scoring windows nearby

> **Non-maximal suppression:** (informally) kill off any detections that have other higher-scoring detections of the same class nearby

non-maximal

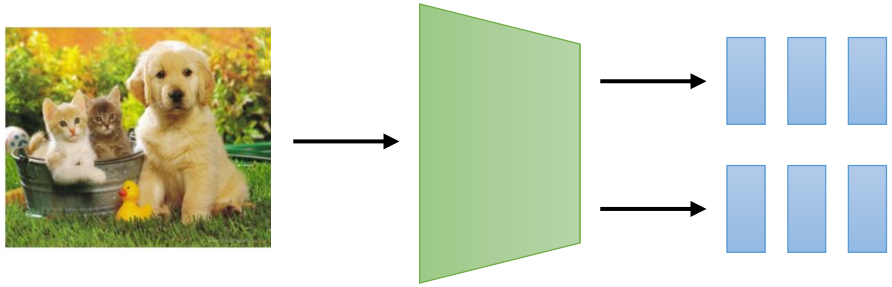
maximal

works great if combined

Actually output multiple things: output is a list of bounding boxes

Obvious problem: need to pick number, usually pretty small

not good by itself



$$\{ \text{“cat”}: 0.21, \text{“dog”}: 0.54 \}$$

$$(x_1, y_1, w_1, h_1, x_2, y_2, w_2, h_2)$$

Case study: you only ~~live~~ once (YOLO)

look

Actually, you look a few times (49 times to be exact…)

different output for each
of the 7x7 (49) grid cells
(a bit like sliding window)

for each cell, output:

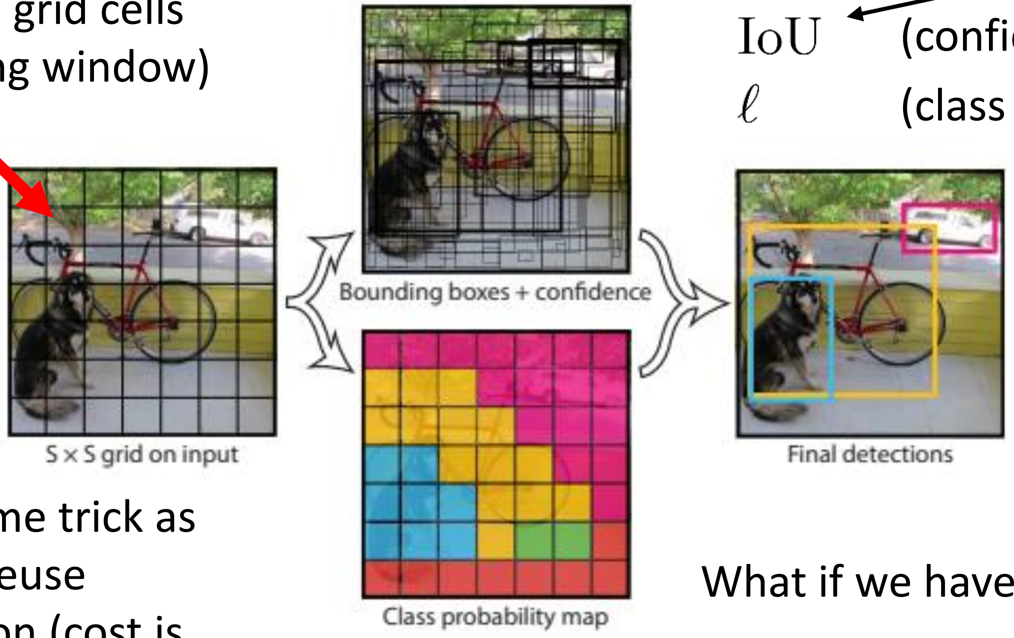
zero if no object

$$\left.\begin{array}{ll} (x, y, w, h) & \\ \mathrm{IoU} & \text{(confidence)} \\ \ell & \text{(class label)} \end{array}\right\} \text{output } B \text{ of these}$$

some training details:

need to assign which output is "responsible" for each true object during training

just use the "best-fit" object in that cell (i.e., the one with highest IoU)



use the same trick as
before to reuse
computation (cost is
not 49x higher!)

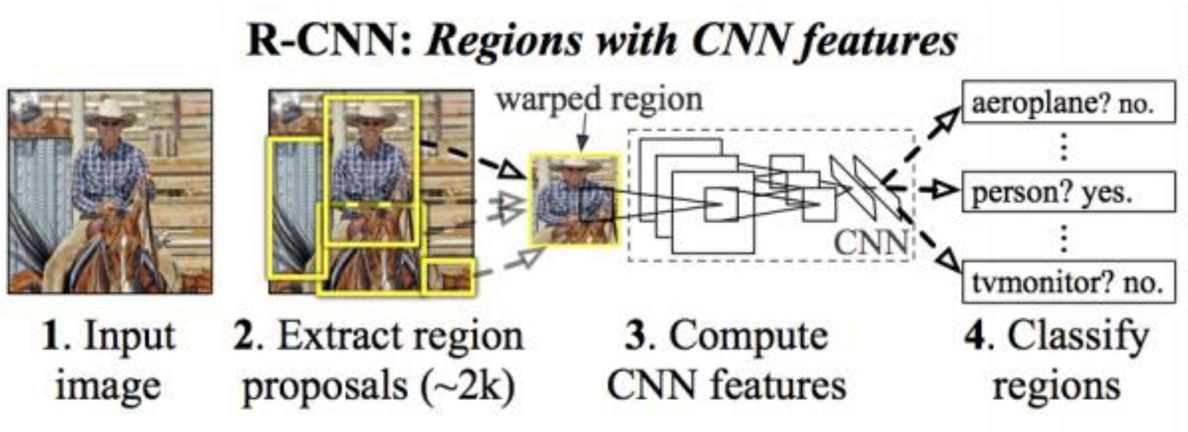
What if we have too many objects?

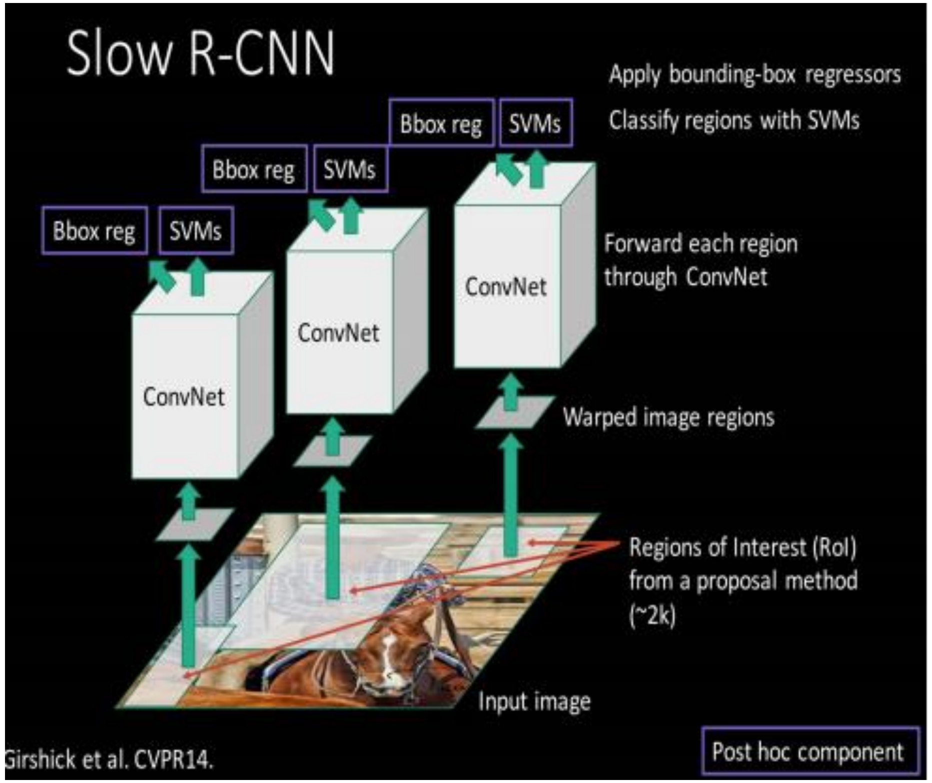
Well, nothing… we just miss them

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

CNNs + Region proposals

A smarter “sliding window”: region of interest proposals





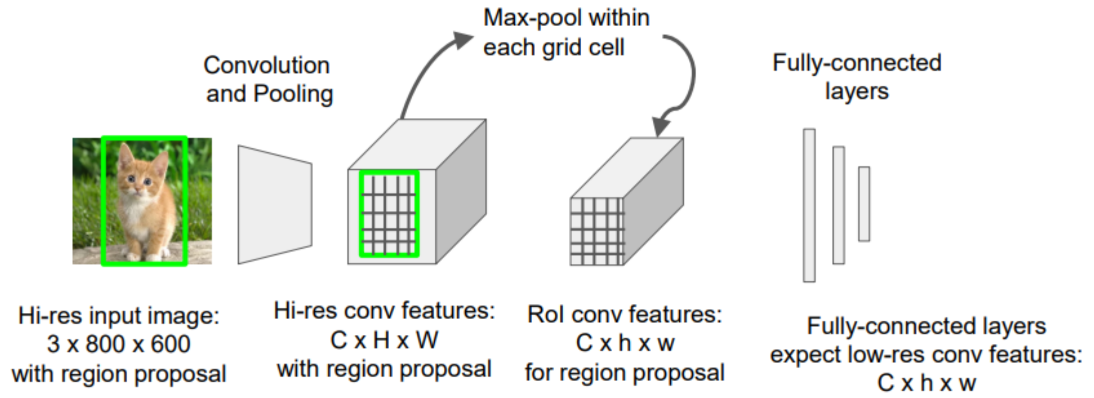
This is really slow

But we already know how to fix this!

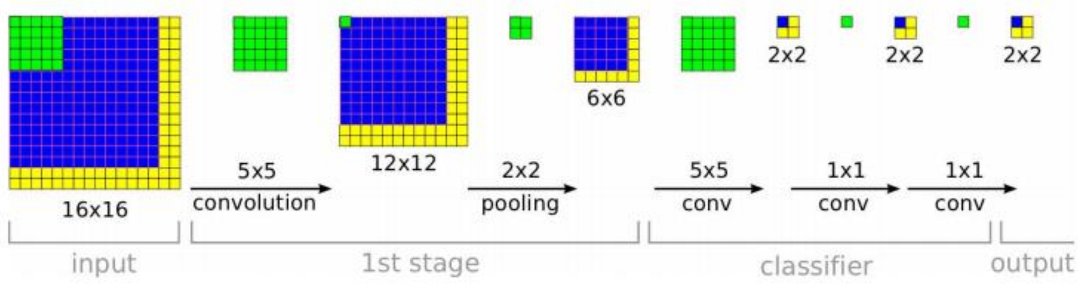
Girschick et al. “**Fast R-CNN.**” 2015

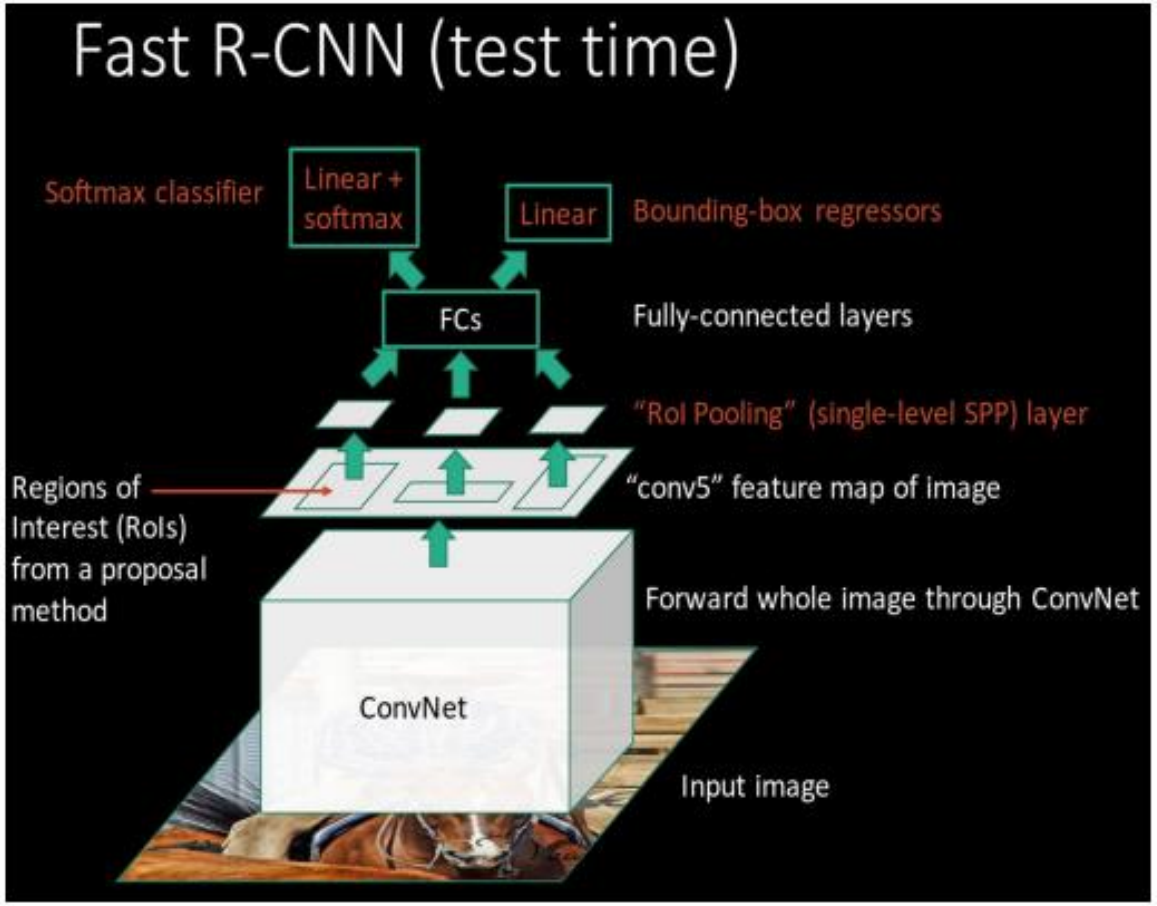
CNNs + Region proposals

A smarter "sliding window": region of interest proposals



Compare this to evaluating every location:



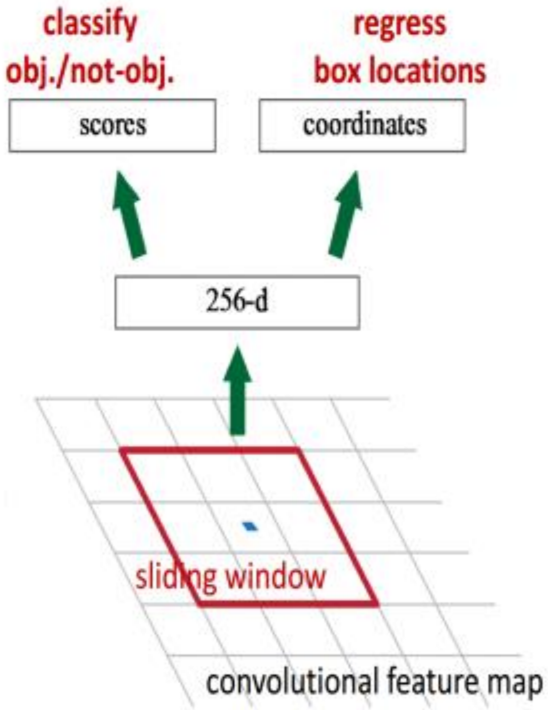


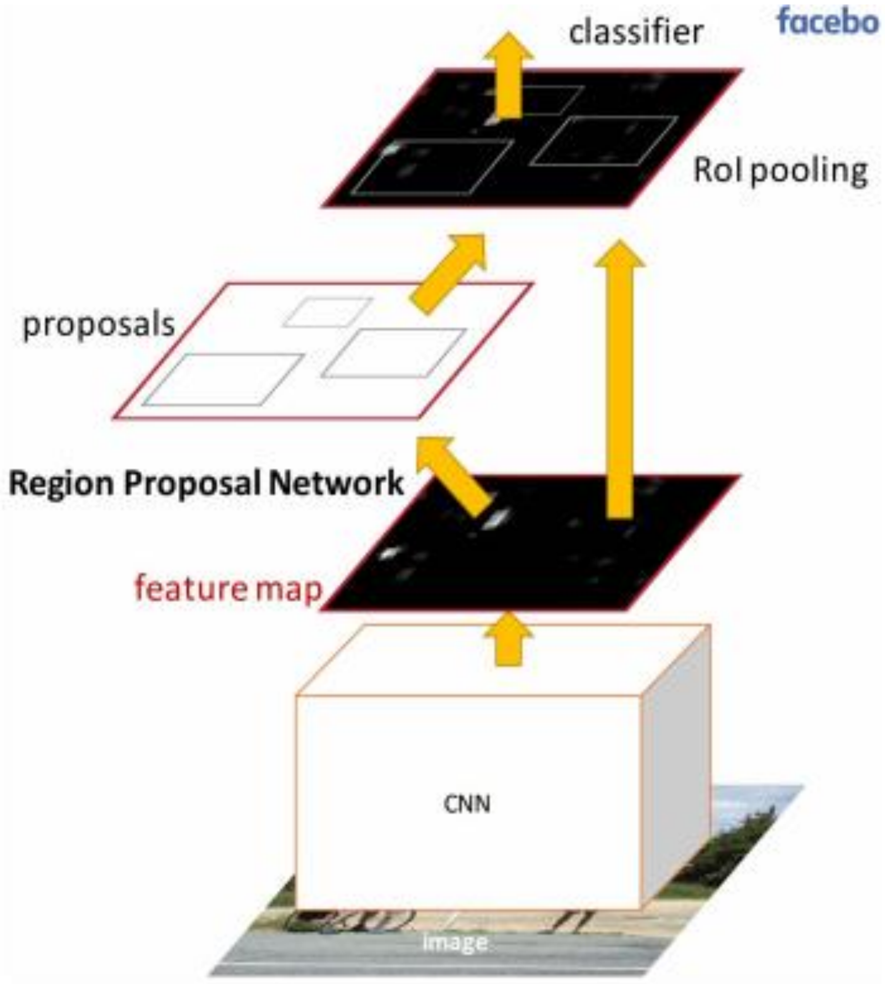
Girschick et al. "**Fast R-CNN.**" 2015

CNNs + Region proposals

How to train region of interest proposals?

Very similar design to what we saw before (e.g., OverFeat, YOLO), but now for predicting if **any** object is present around that location





Ren et al. “**Faster R-CNN.**” 2015

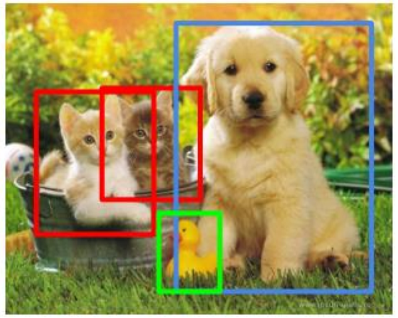
Suggested readings

- Redmon et al. "**You Only Look Once: Unified, Real-Time Object Detection.**" 2015
  - Just regress to different bounding boxes in each cell
  - A few follow-ups (e.g., YOLO v5) that work better
- Girschick et al. "**Fast R-CNN.**" 2015
  - Uses region of interest proposals instead of sliding window/convolution
- Ren et al. "**Faster R-CNN.**" 2015
  - Same as above with a few improvements, like region of interest proposal learning
- Liu et al. **SSD: Single Shot MultiBox Detector**. 2015
  - Directly "classifies" locations with class and bounding box shape

Segmentation architectures

The problem setup

Before



Now

Problem:

We want the output to have the same resolution as the input!

Label **every single** pixel with its class

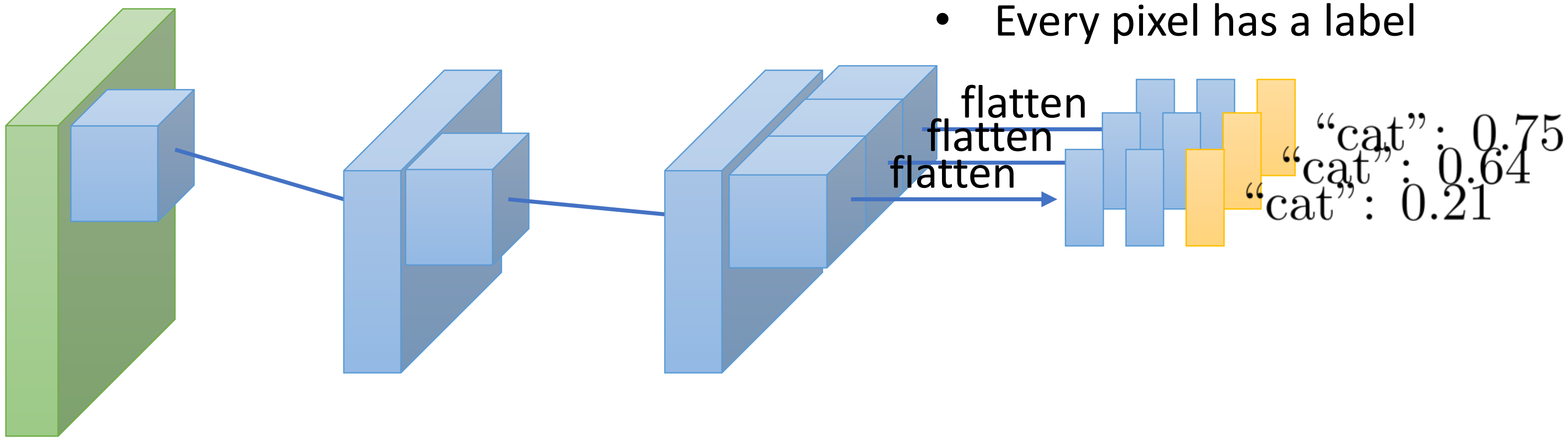
Actually simpler in some sense:

- No longer variable # of outputs
- Every pixel has a label

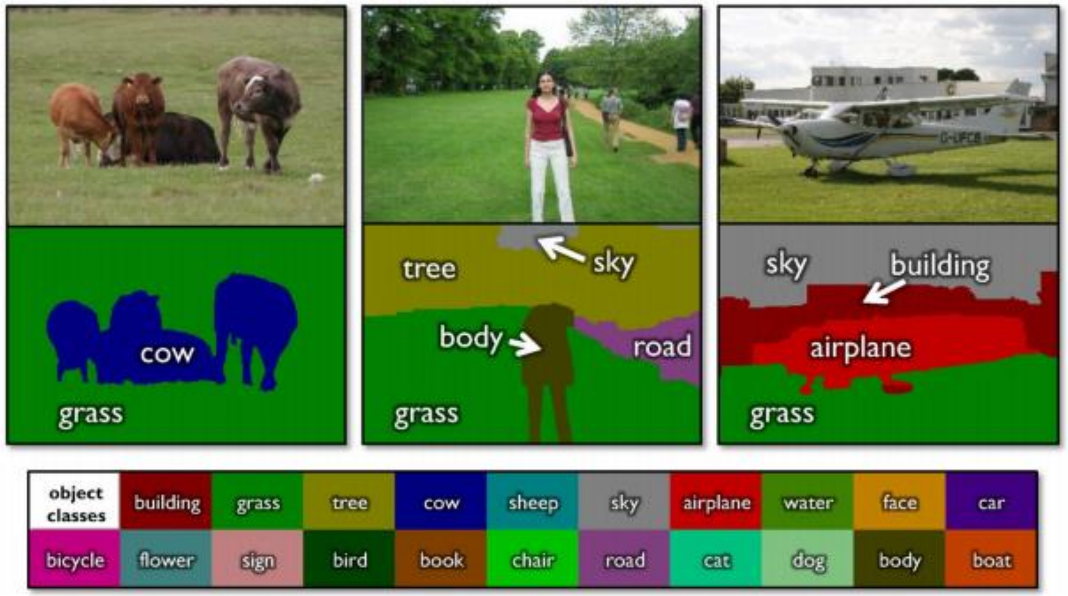
Not hard if we never downsample (i.e., zero padding, stride 1, no pooling), but that is very expensive

Simple solution:

“per pixel” classifier



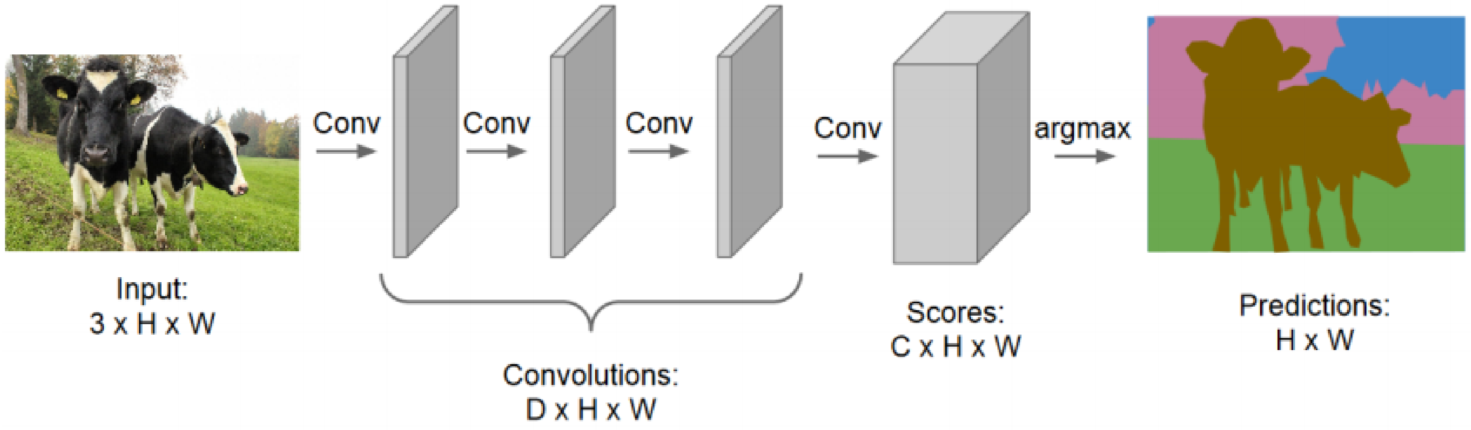
The problem setup



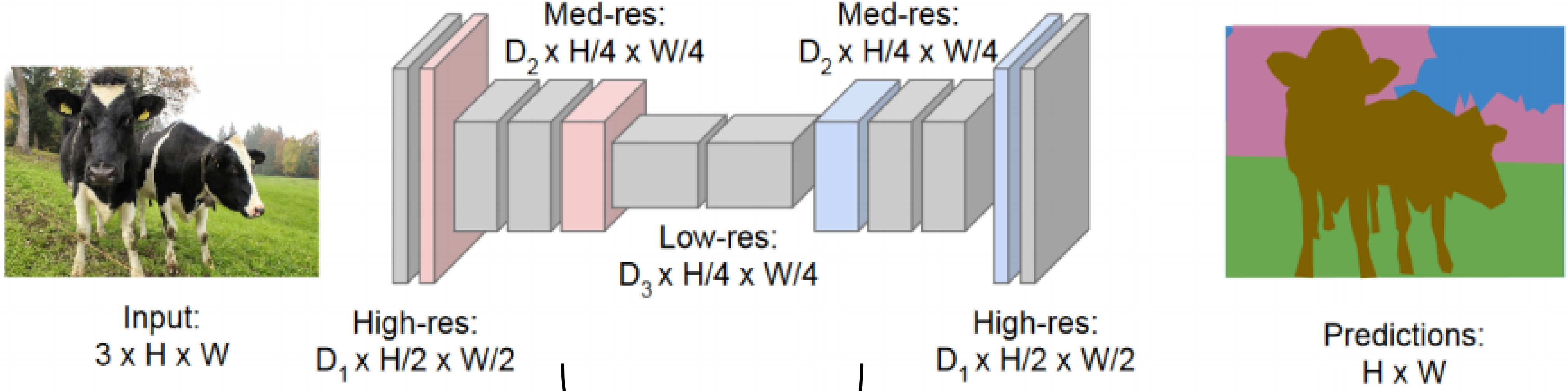
Classify every point with a class

Don’t worry for now about instances
(e.g., two adjacent cows are just one “cow blob,” and that’s OK for some reason)

The challenge: design a network architecture that makes this “per-pixel classification” problem computationally tractable



Fully convolutional networks



Low-res (but high-depth) processing in the middle integrates context from the entire image

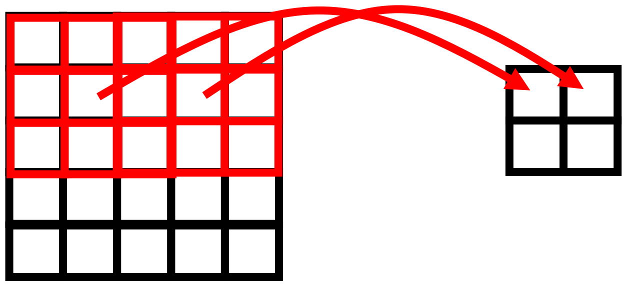
Up-sampling at the end turns these low-res feature vectors into high-res per-pixel predictions

Slide borrowed from Fei-Fei Li, Justin Johnson, Serena Yeung

Up-sampling/transpose convolution

Normal convolutions: reduce resolution with **stride**

Stride = 2



input: $H_f \times W_f \times C_{\text{in}}$

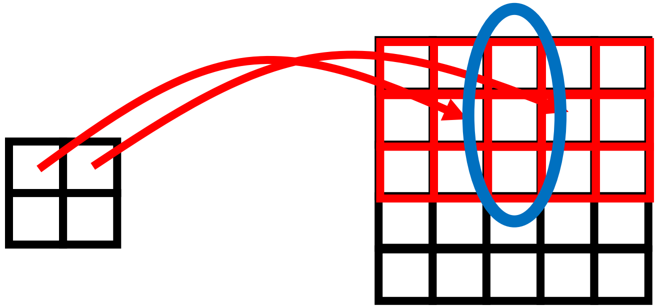
output: $1 \times 1 \times C_{\text{out}}$

filter: $H_f \times W_f \times C_{\text{in}} \times C_{\text{out}}$

Transpose convolutions: increase resolution with **fractional "stride"**

Stride = 1/2

we have two sets of values here! **just average them**

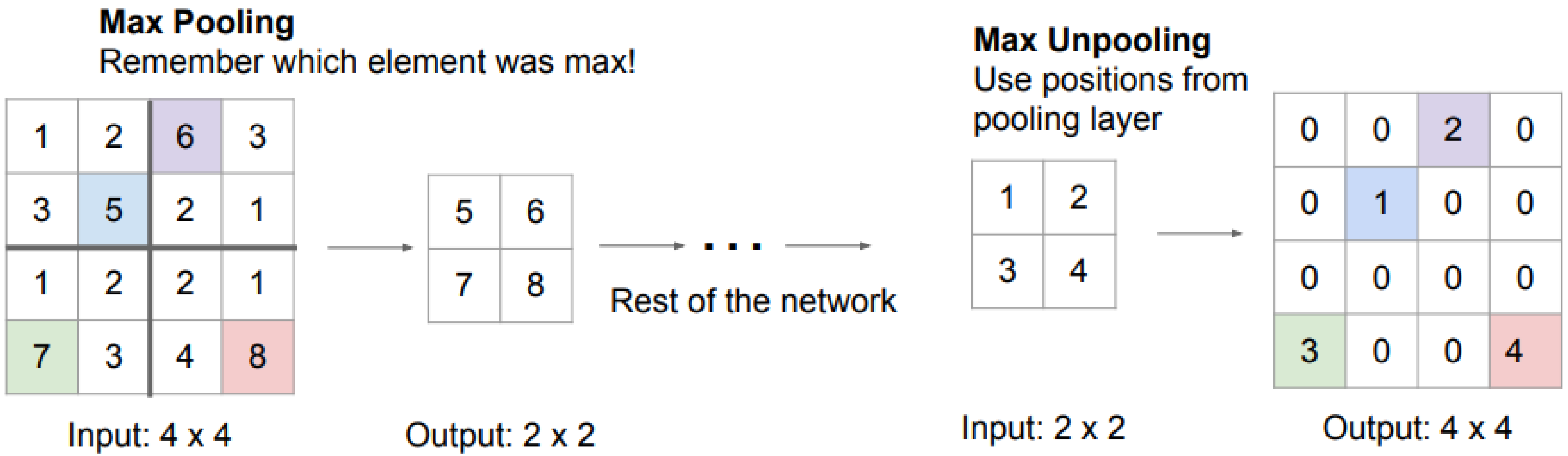


input: $1 \times 1 \times C_{\text{in}}$

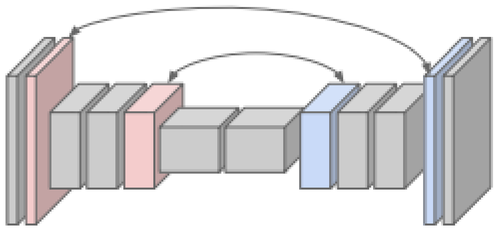
output: $H_f \times W_f \times C_{\text{out}}$

filter: $C_{\text{in}} \times H_f \times W_f \times C_{\text{out}}$

Un-pooling

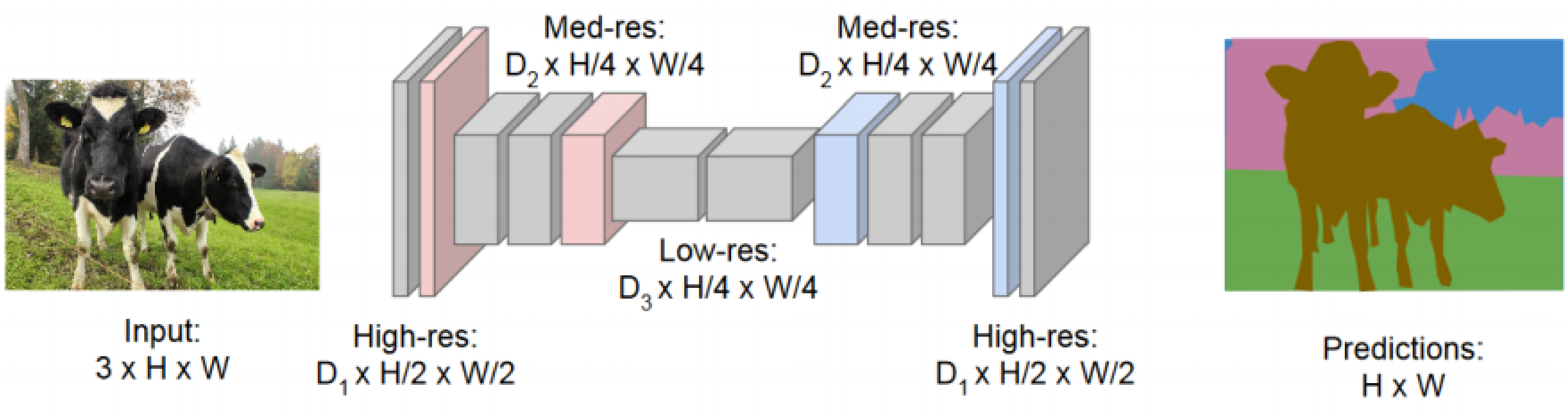


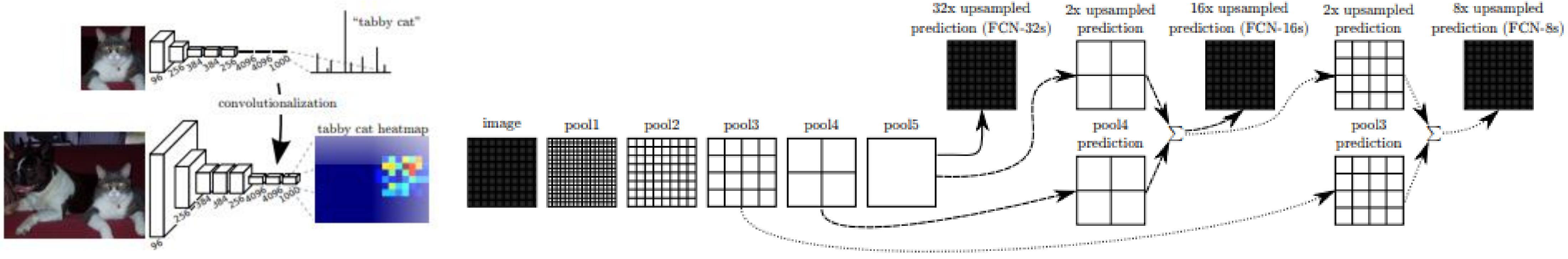
Corresponding pairs of
downsampling and
upsampling layers



Slide borrowed from Fei-Fei Li, Justin Johnson, Serena Yeung

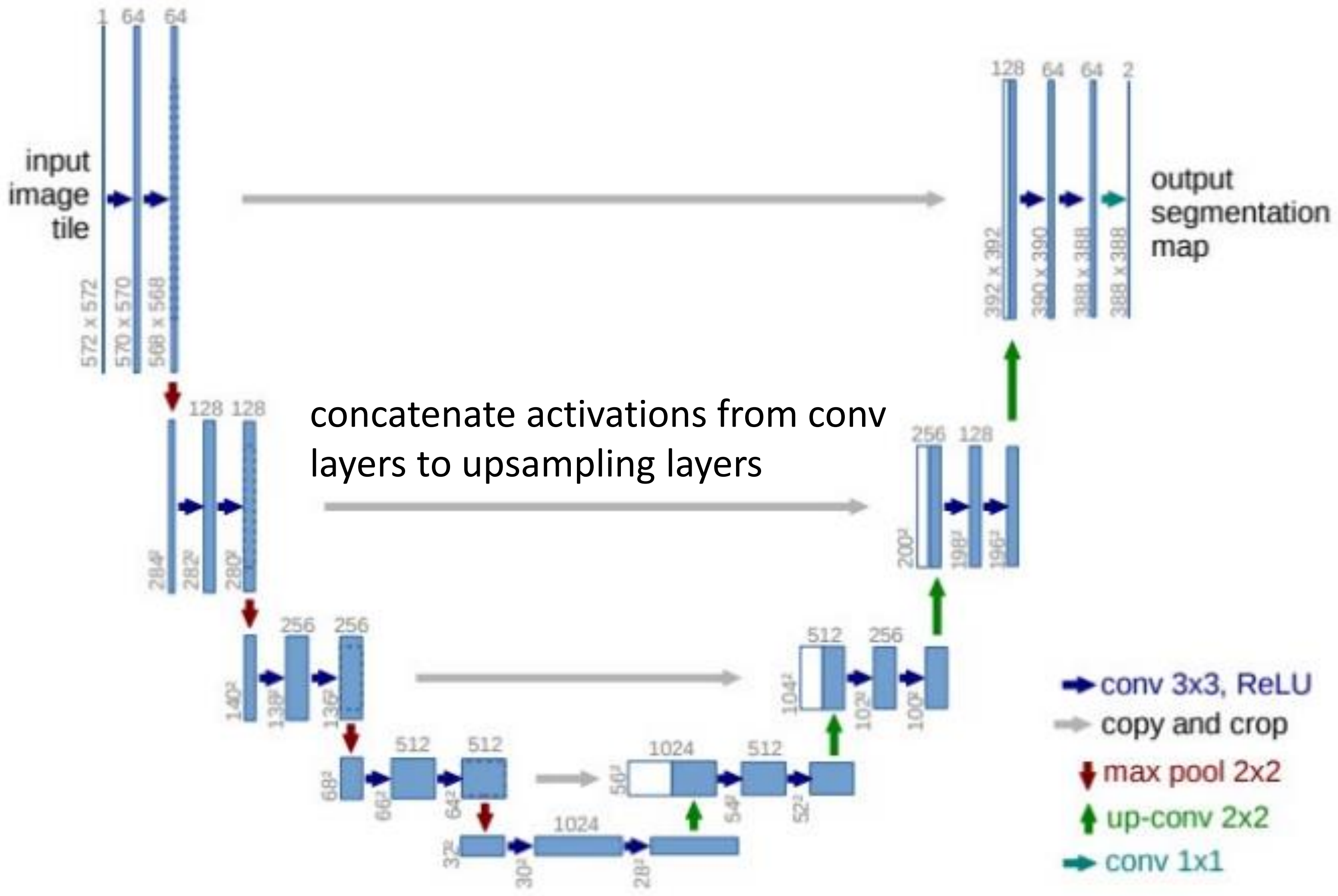
Bottleneck architecture





Long et al. "**Fully Convolutional Networks for Semantic Segmentation.**" 2014

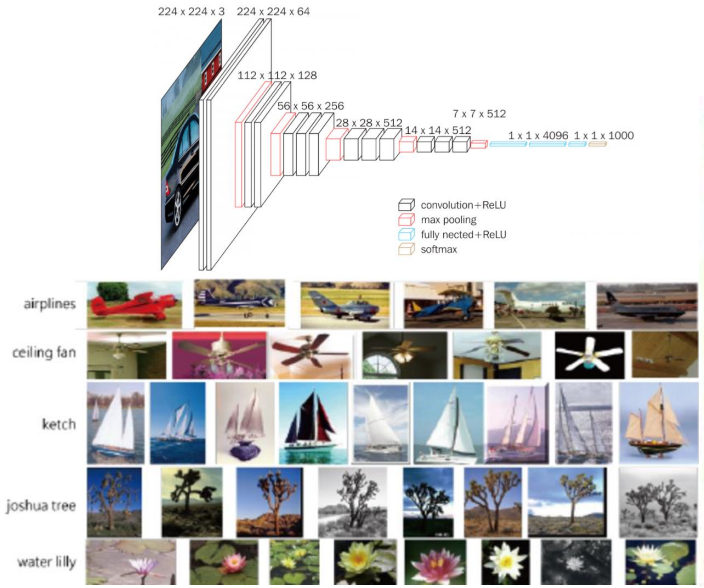
U-Net architecture



concatenate activations from conv layers to upsampling layers

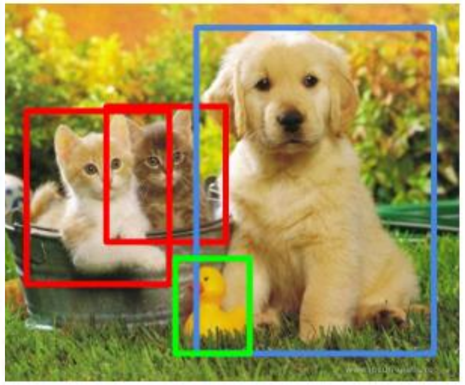
Ronneberger et al. **U-net: Convolutional networks for biomedical image segmentation**. 2015

Standard computer vision problems

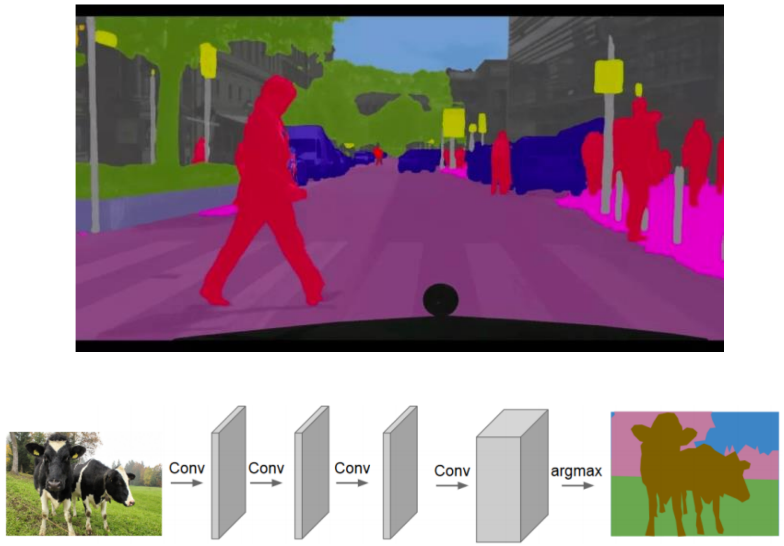


object classification

object localization



object detection



semantic segmentation
a.k.a. scene understanding