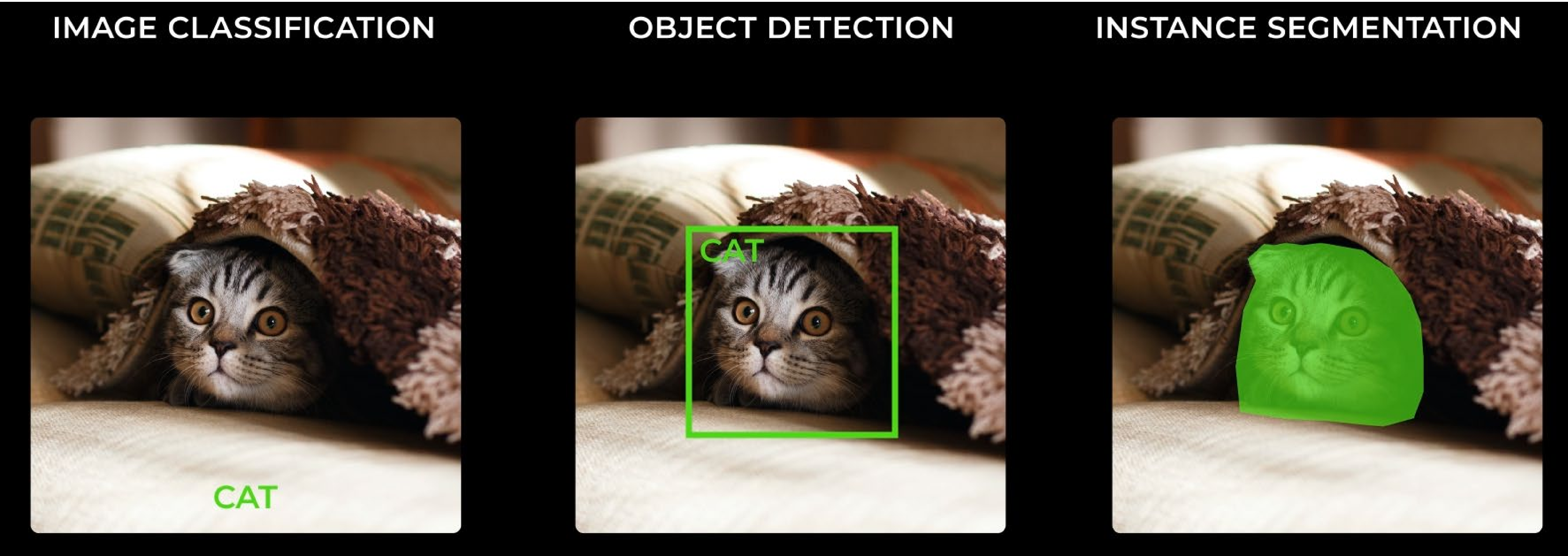
Contains slides from Owens, Isola

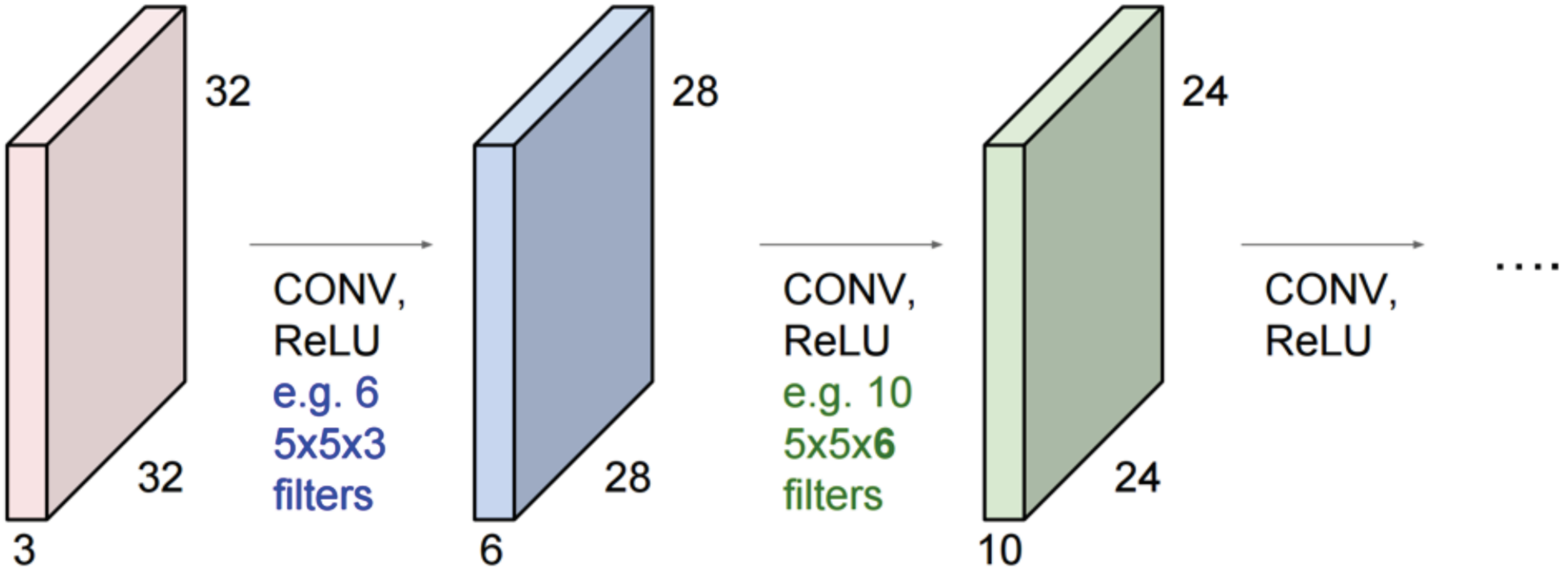
# Lecture 10

## Visual Recognition



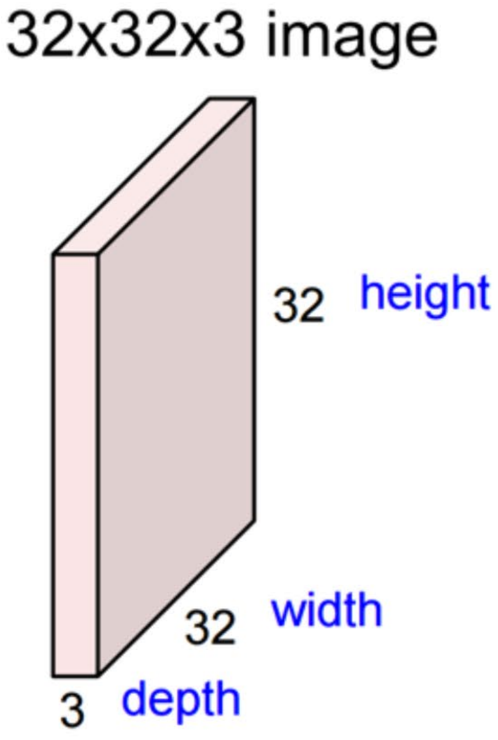
# (Recap)

A **ConvNet** is a sequence of convolutional layers, interspersed with activation functions (and possibly other layer types)



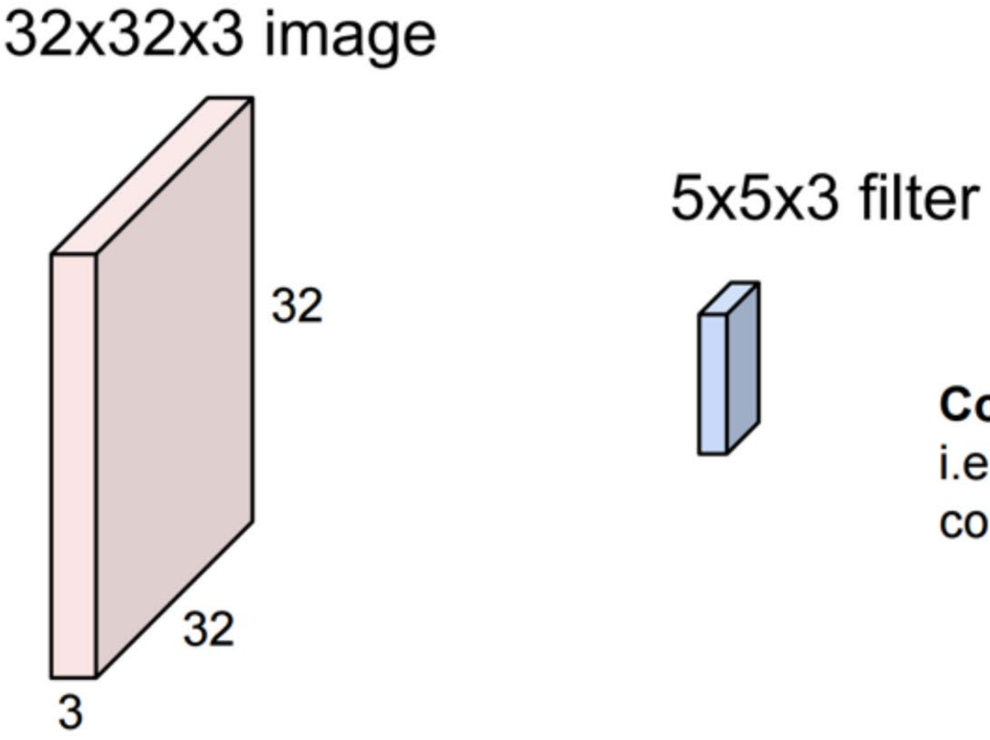
# (Recap)

## Convolution Layer



# (Recap)

## Convolution Layer



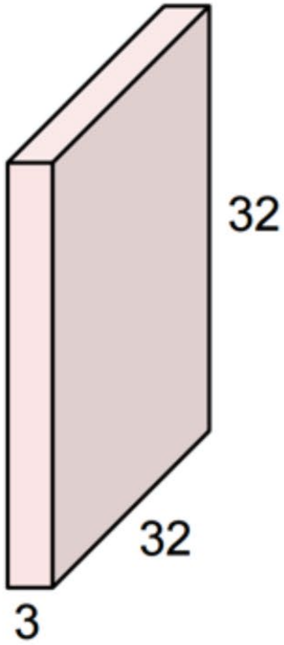

**Convolve** the filter with the image
i.e. “slide over the image spatially,
computing dot products”

# (Recap)

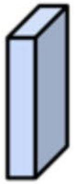
## Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



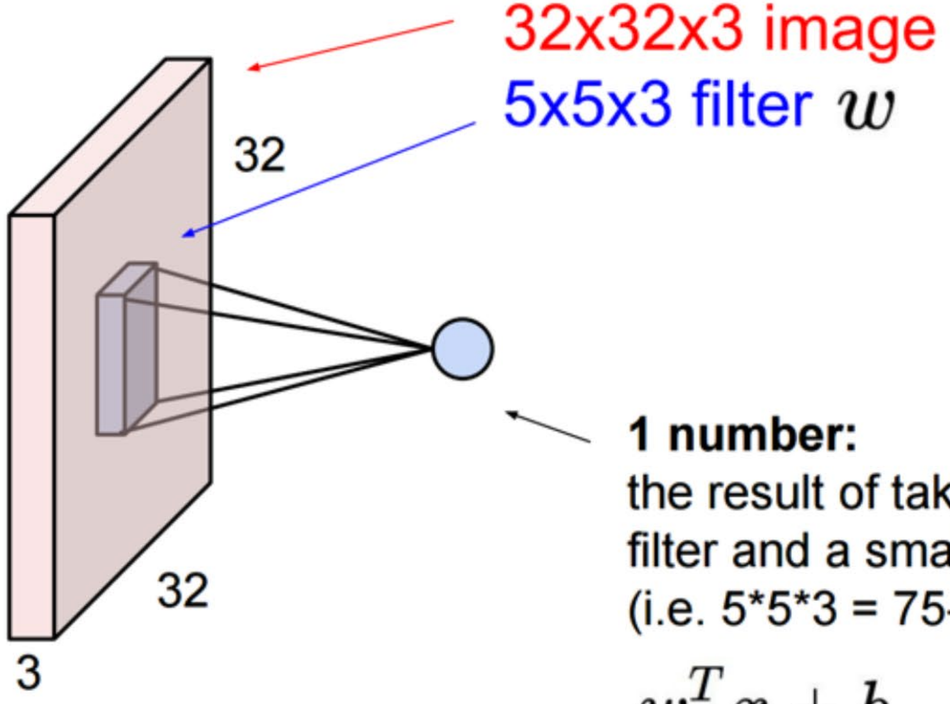

5x5x3 filter



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

# (Recap)

## Convolution Layer

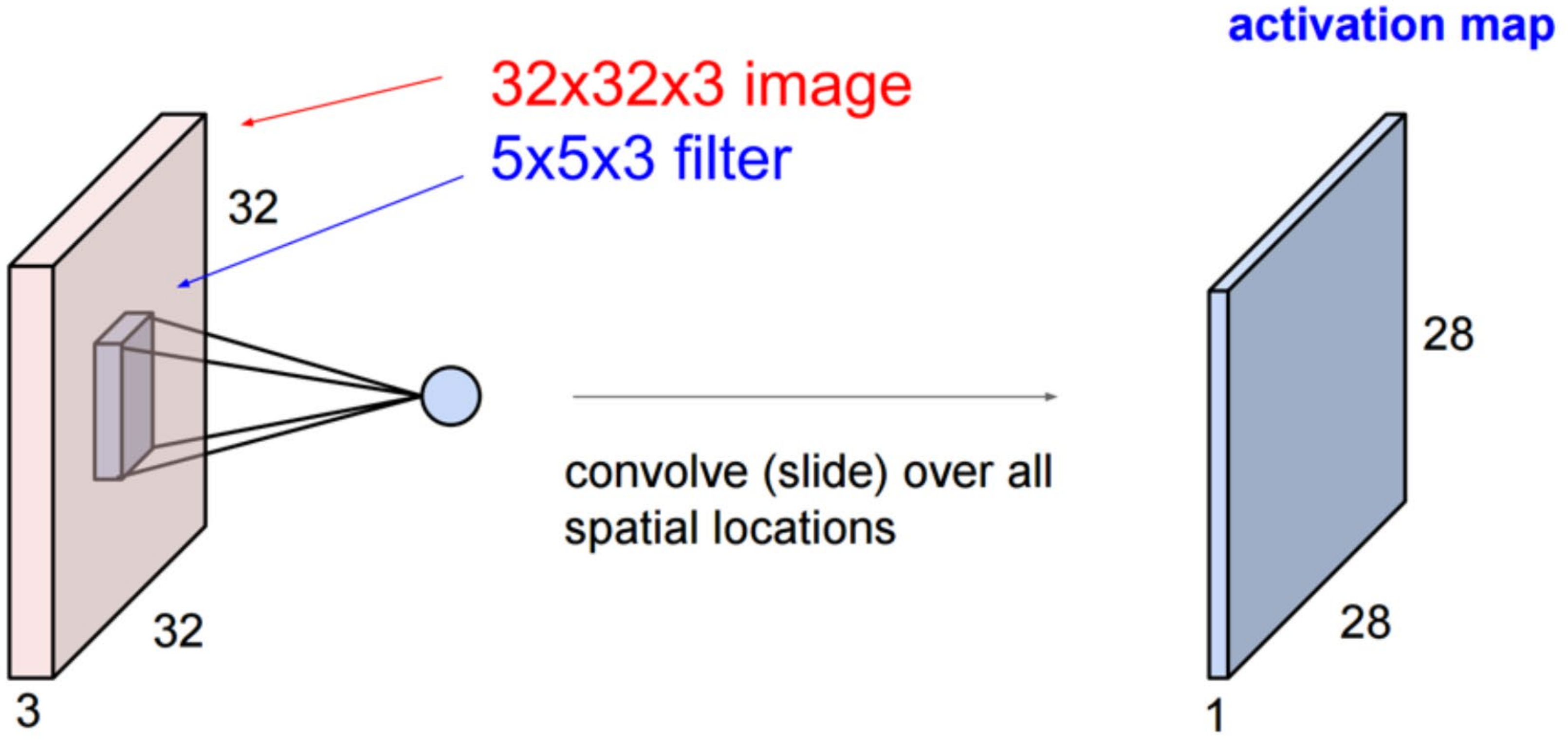



**1 number:**
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

# (Recap)

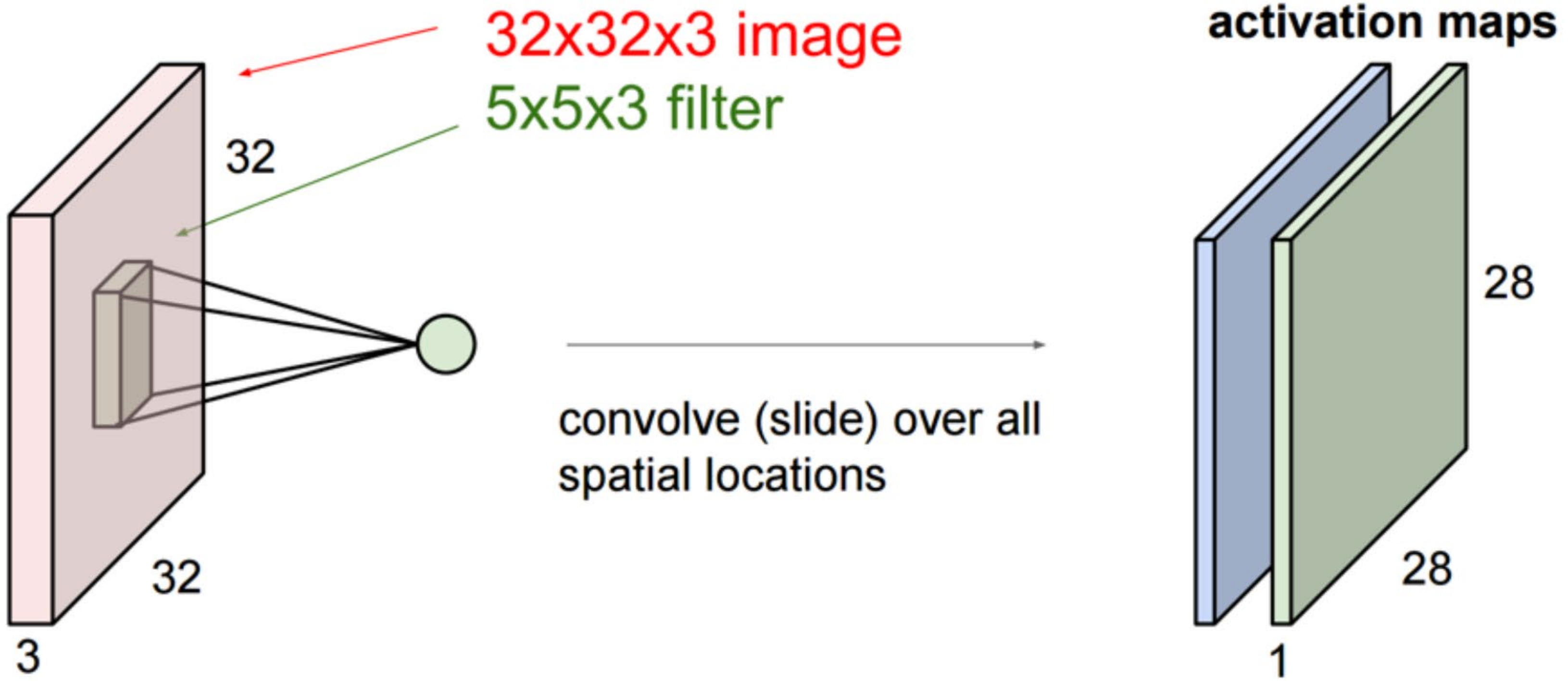
## Convolution Layer



# (Recap)

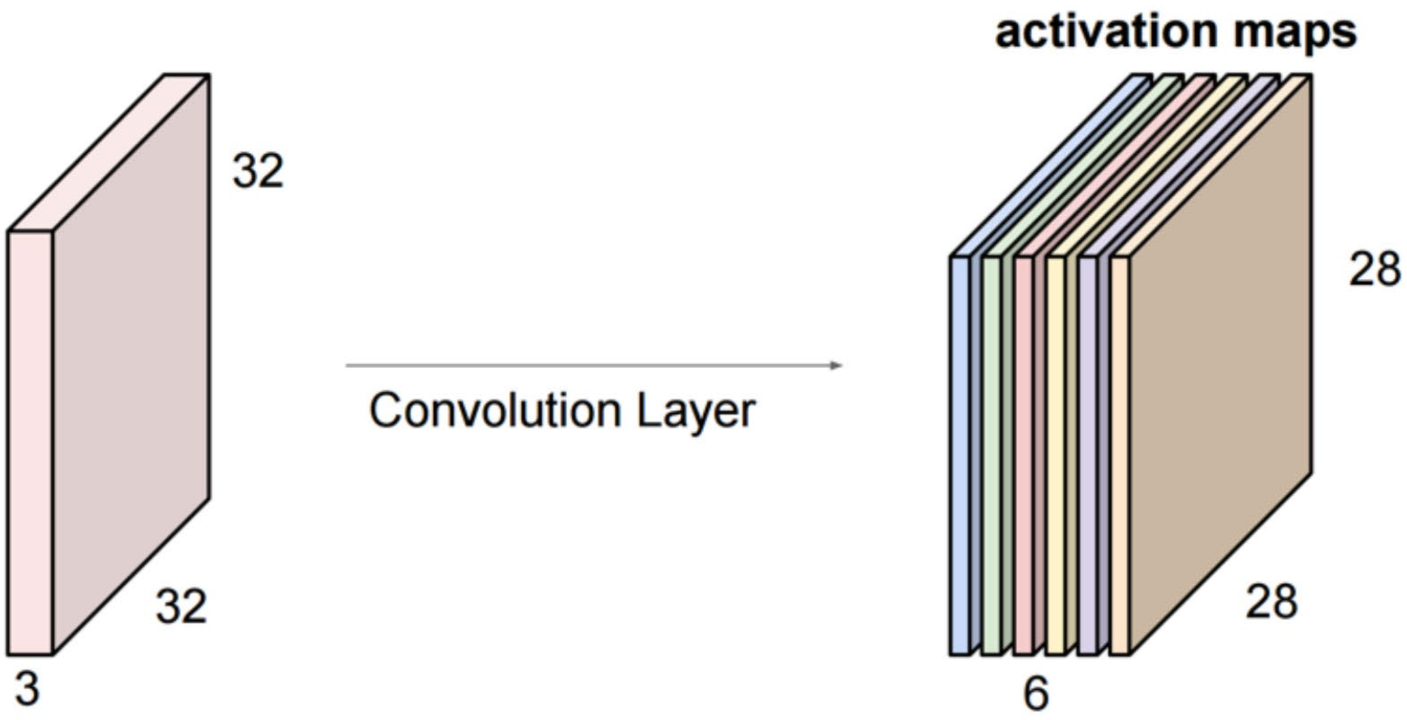
## Convolution Layer

consider a second, green filter



# (Recap)

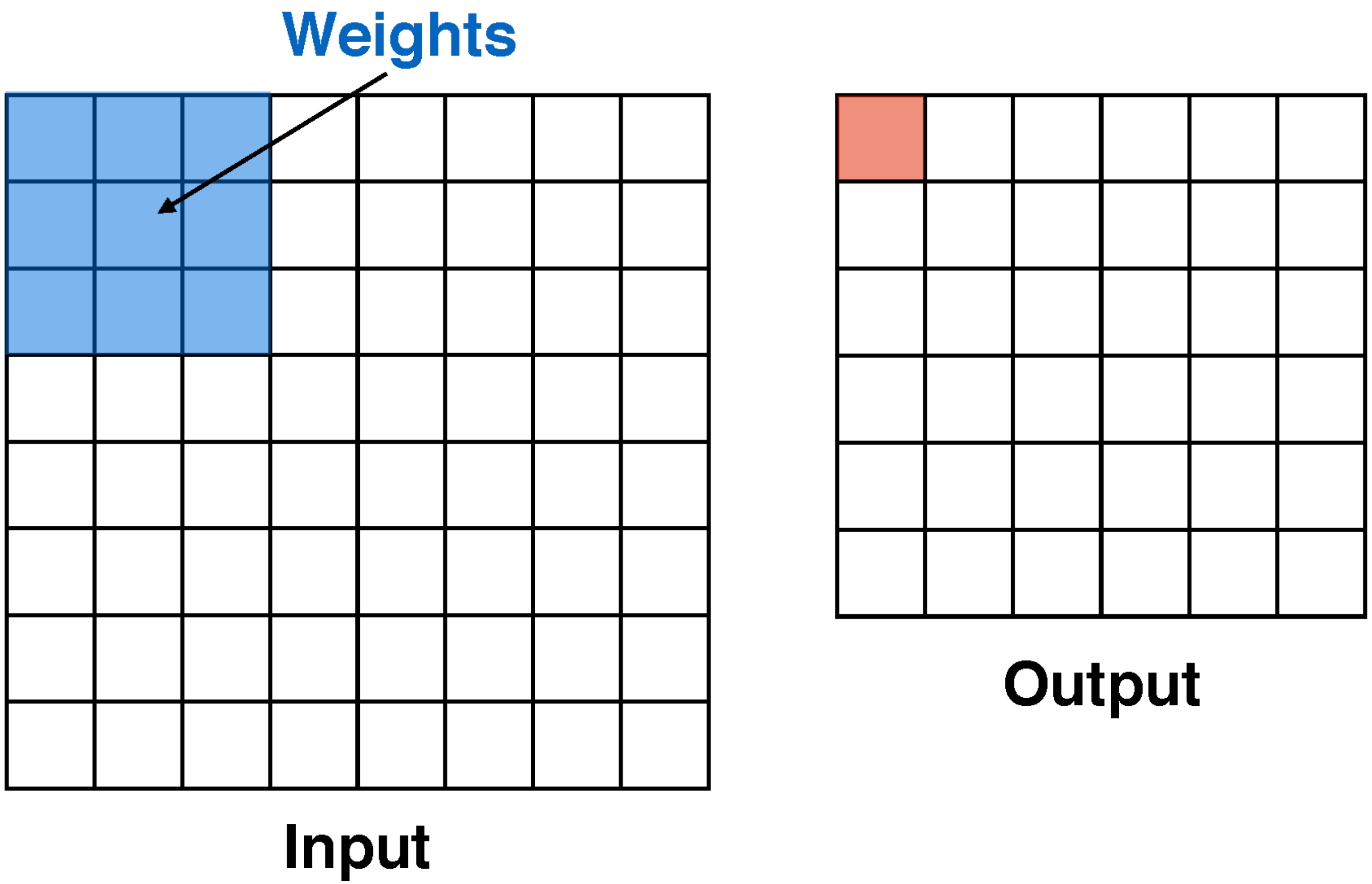
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:




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

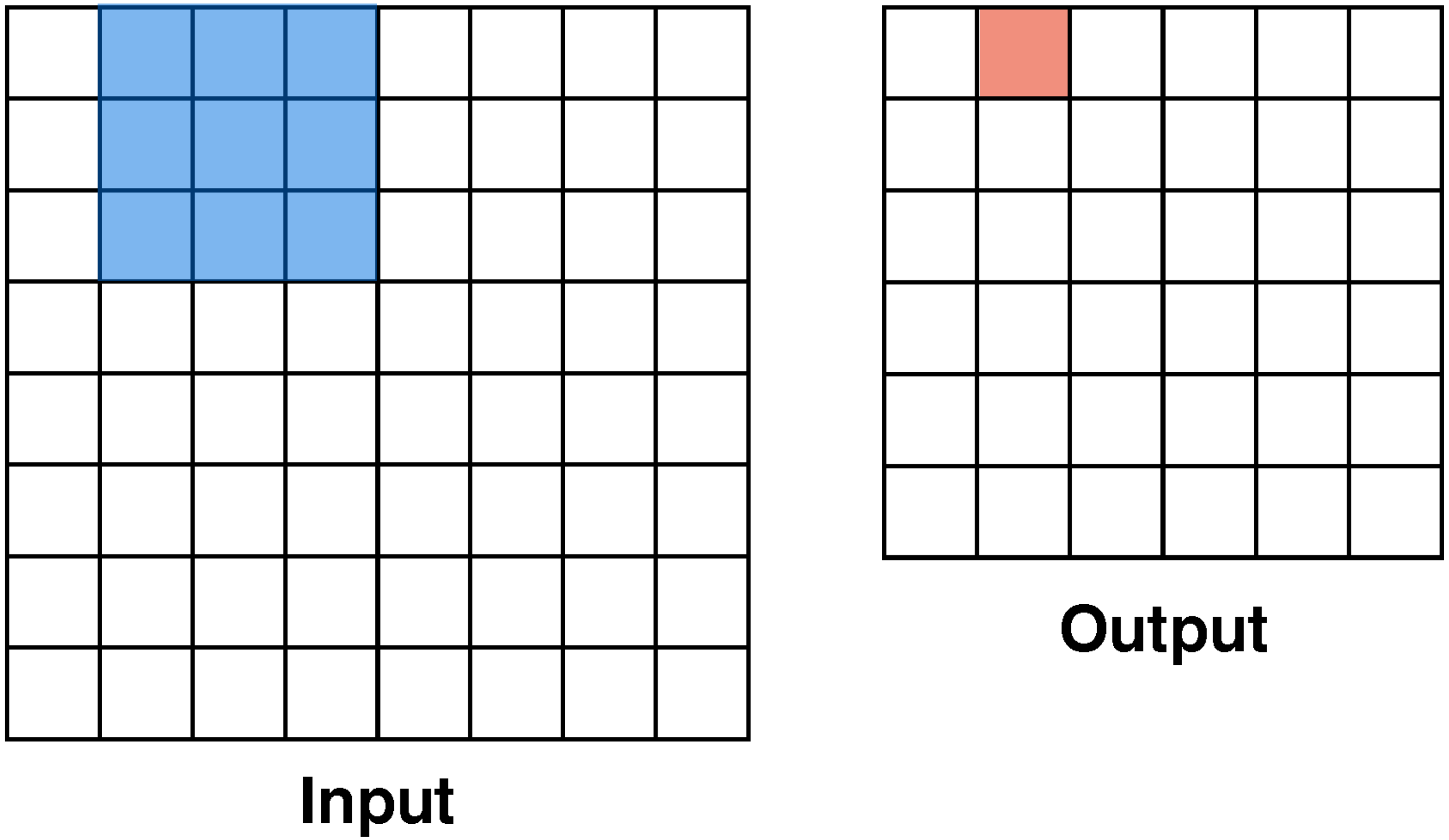
# Convolution: Stride

During convolution, the weights "slide" along the input to generate each output



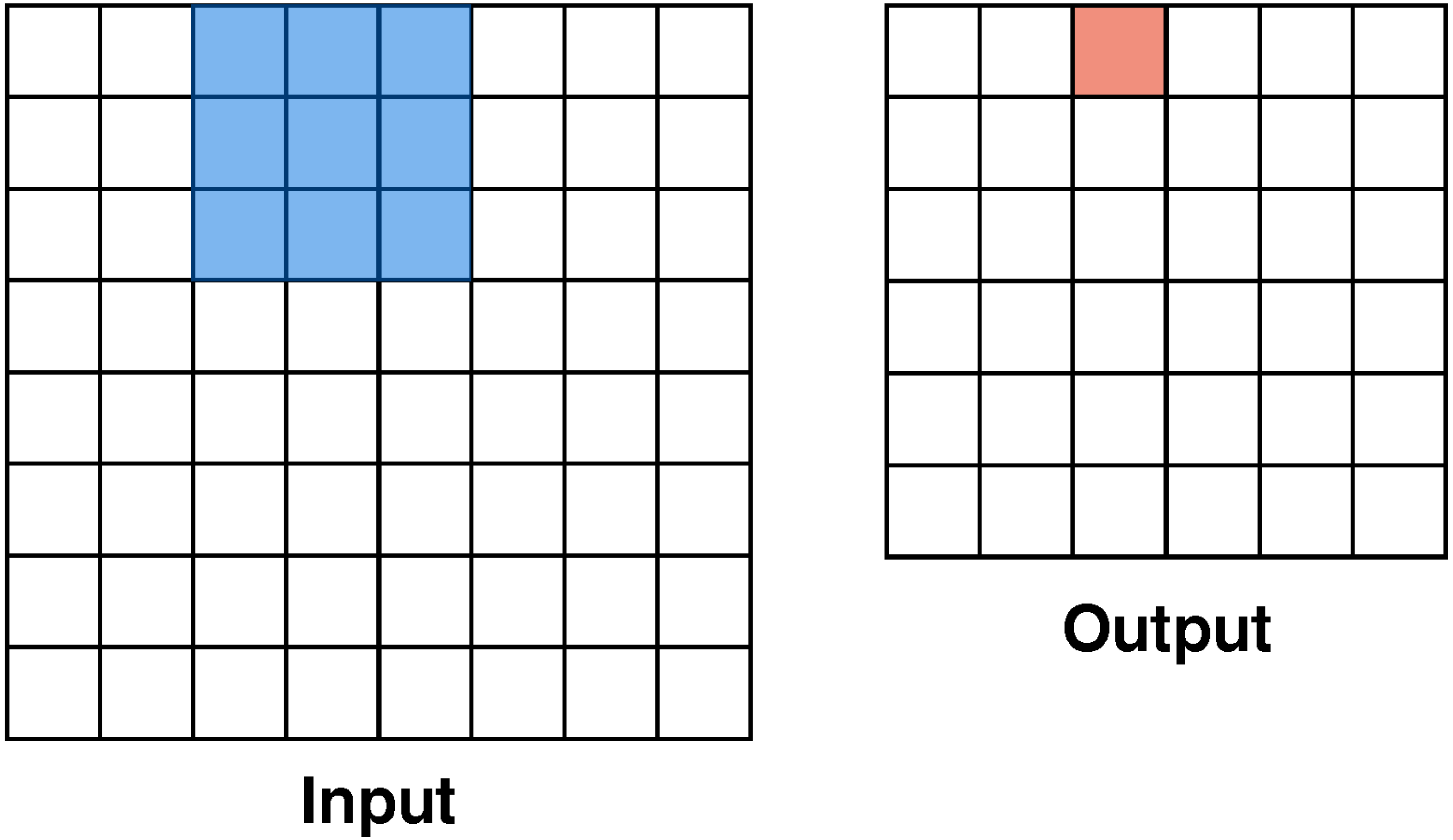
# Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



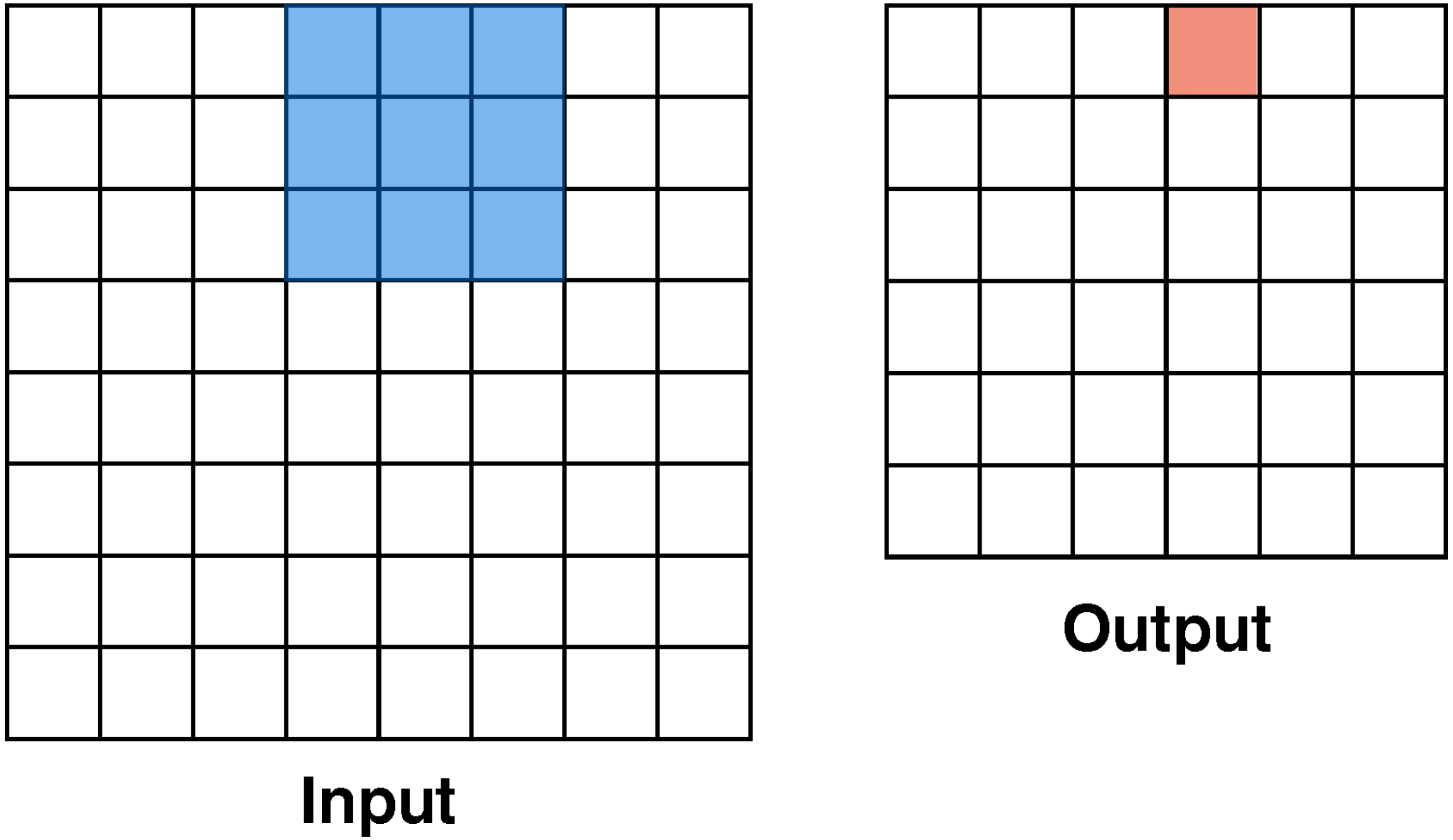
# Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



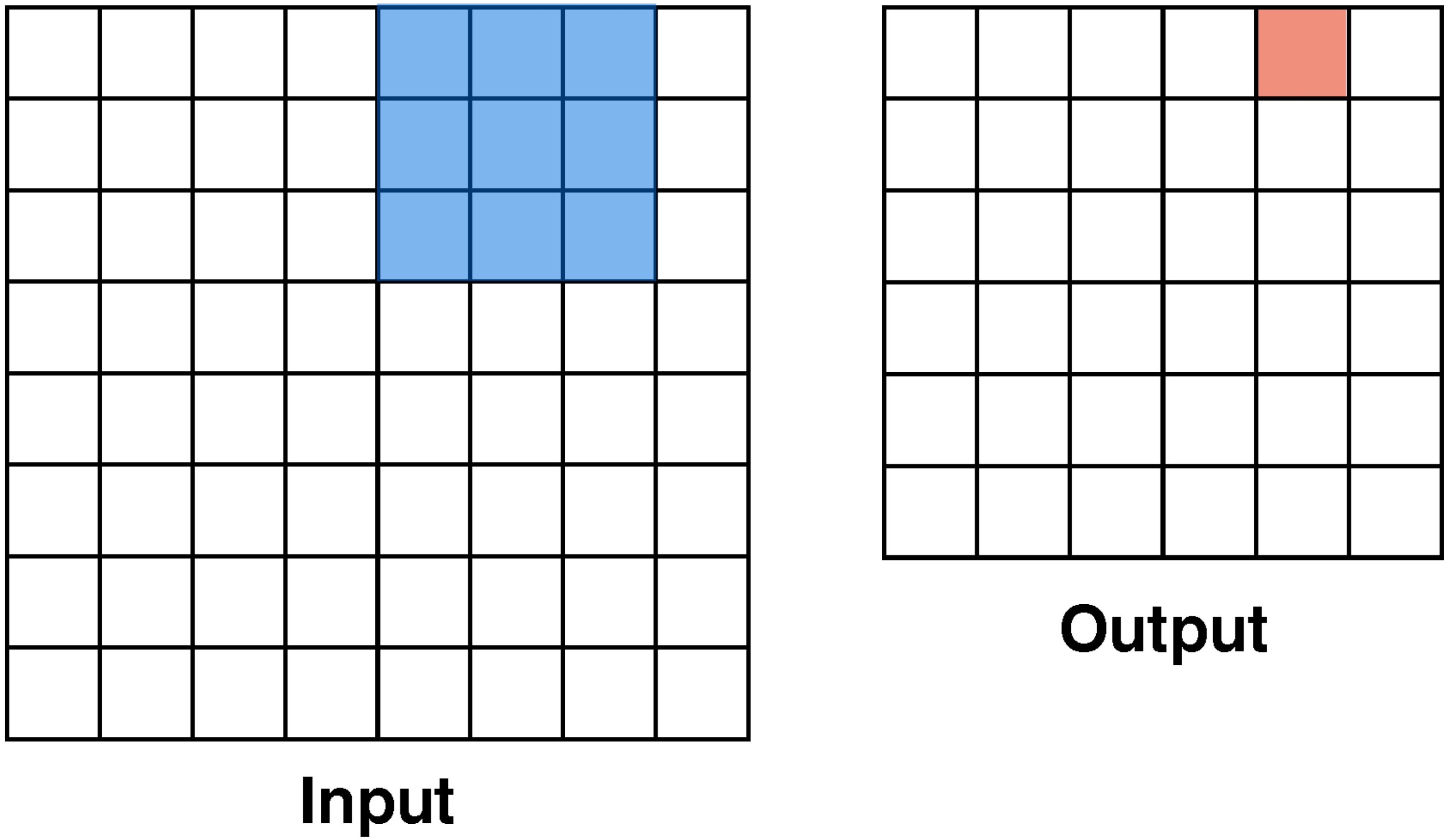
# Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



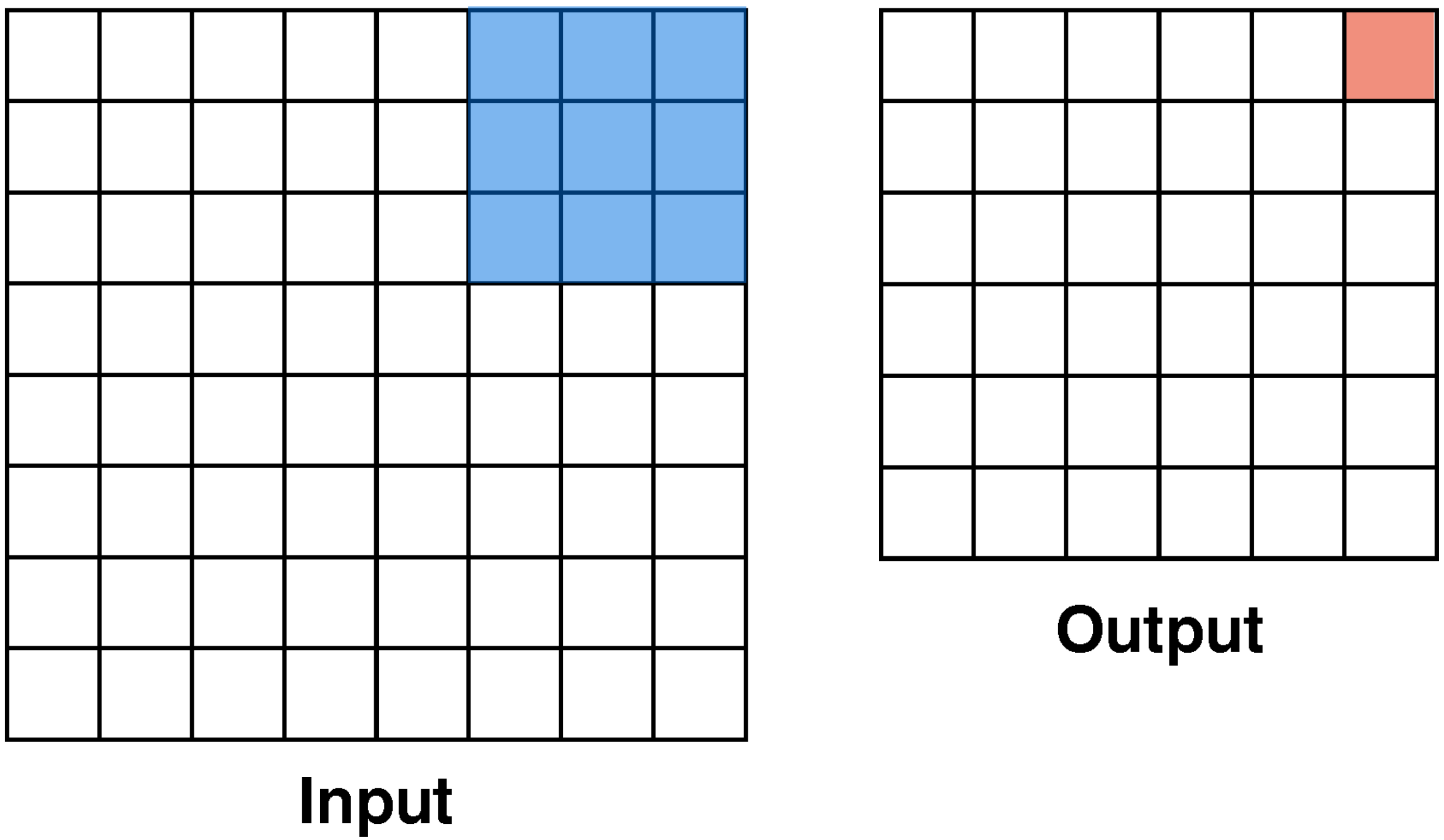
# Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



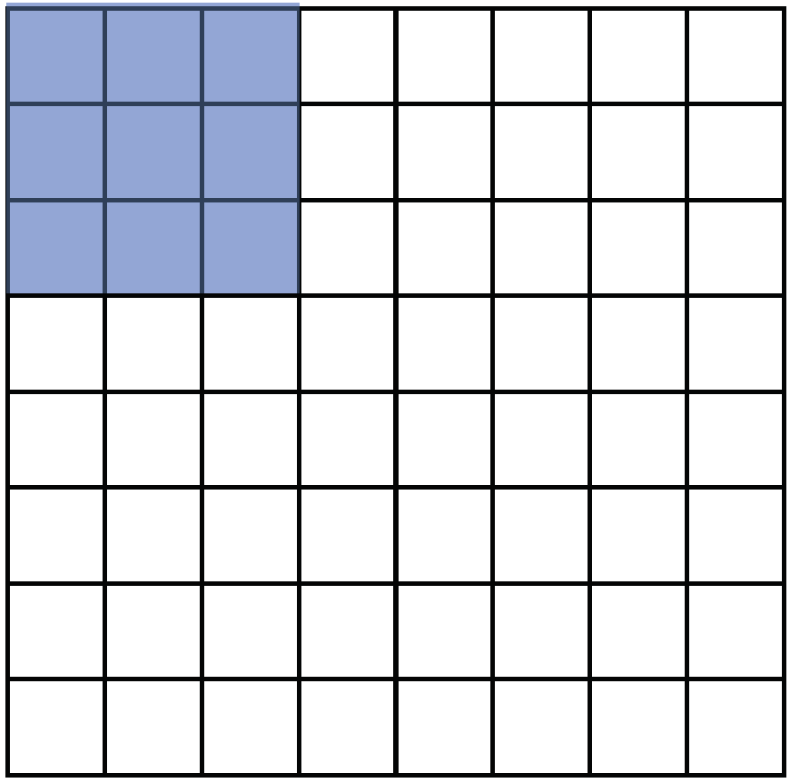
# Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



# Convolution: Stride

During convolution, the weights "slide" along the input to generate each output



**Input**

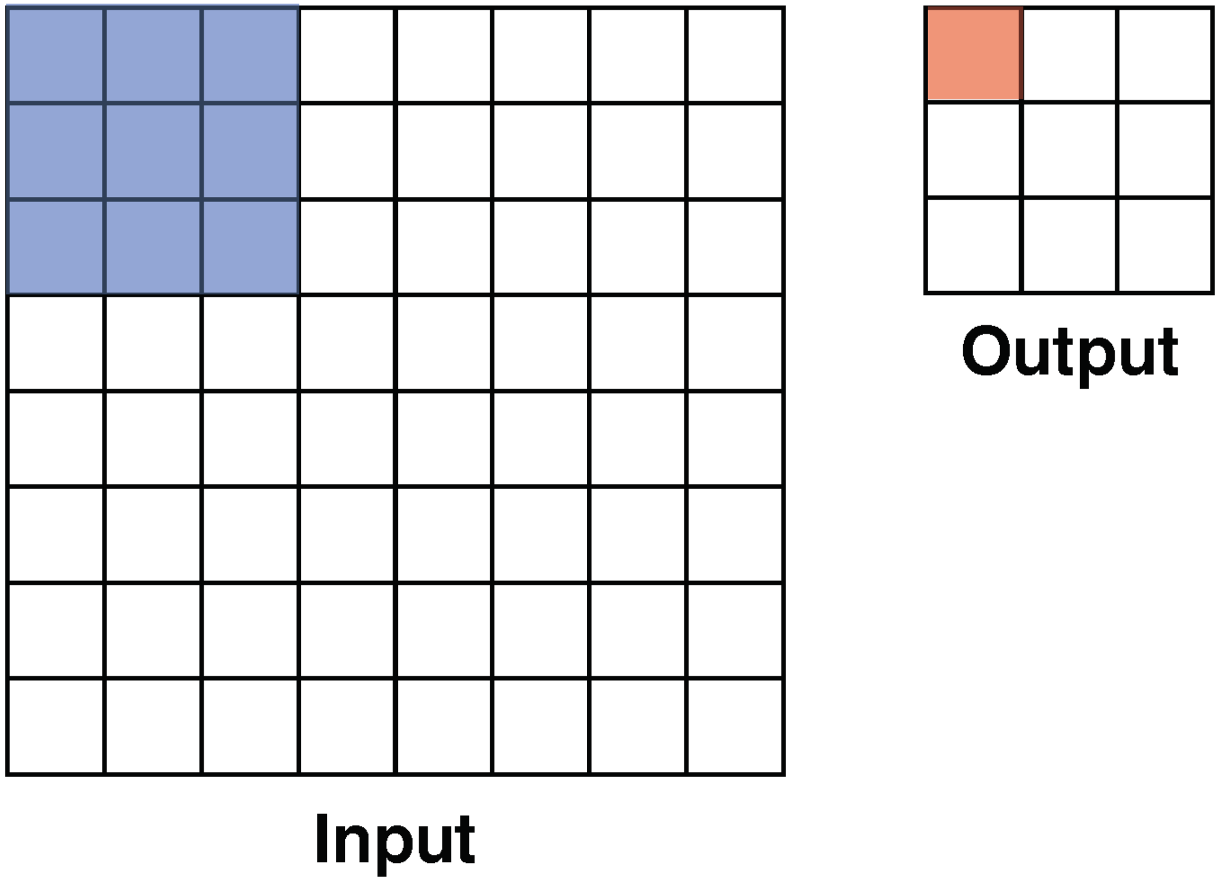
Recall that at each position, we are doing a **3D** sum:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

*(channel, row, column)*

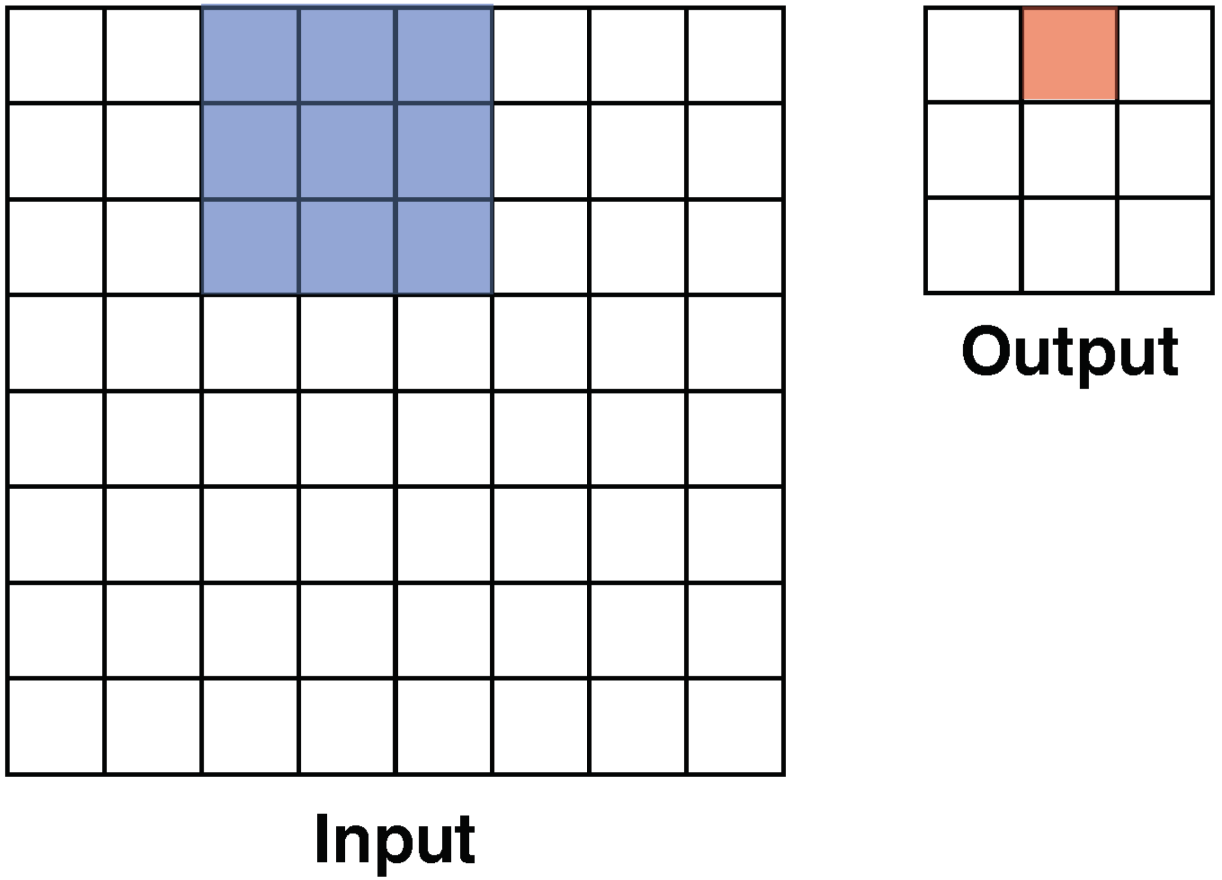
# Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



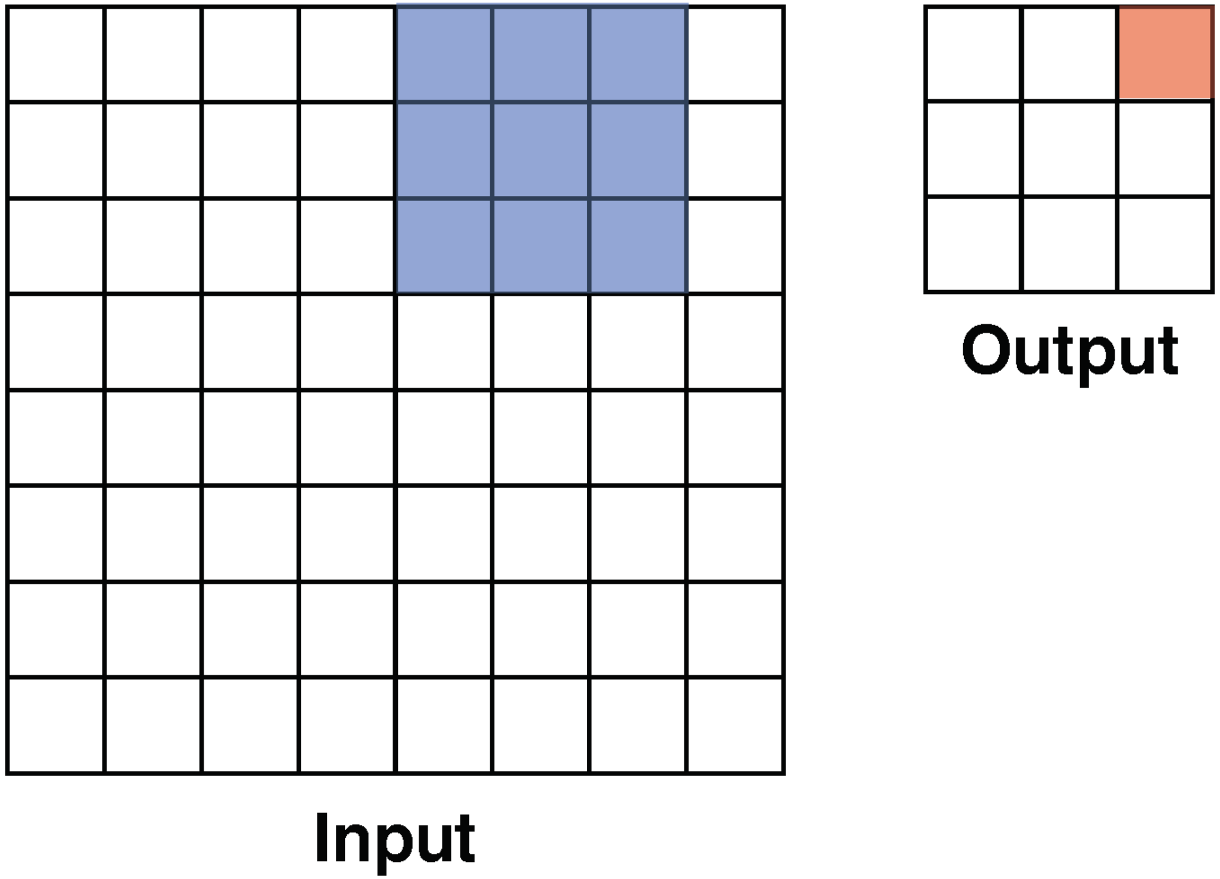
# Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



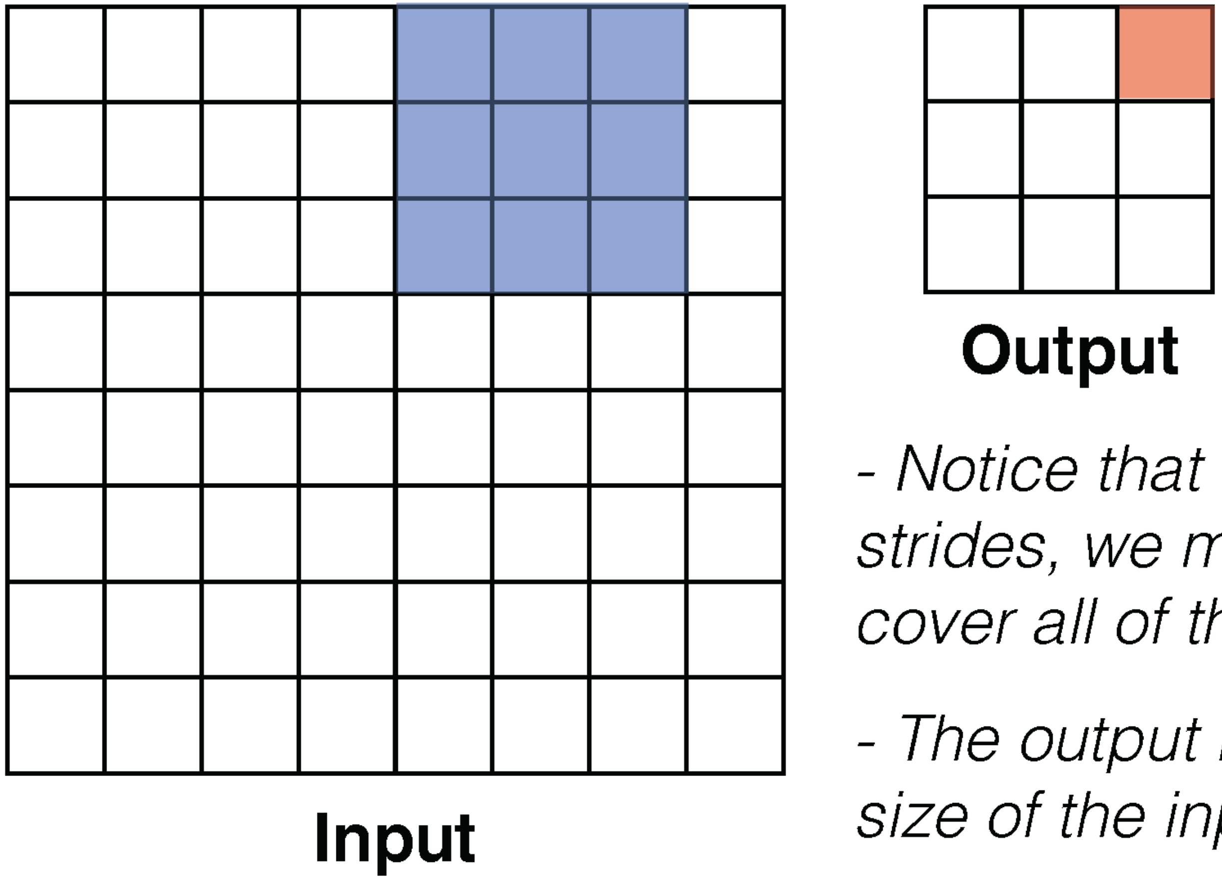
# Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



# Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2

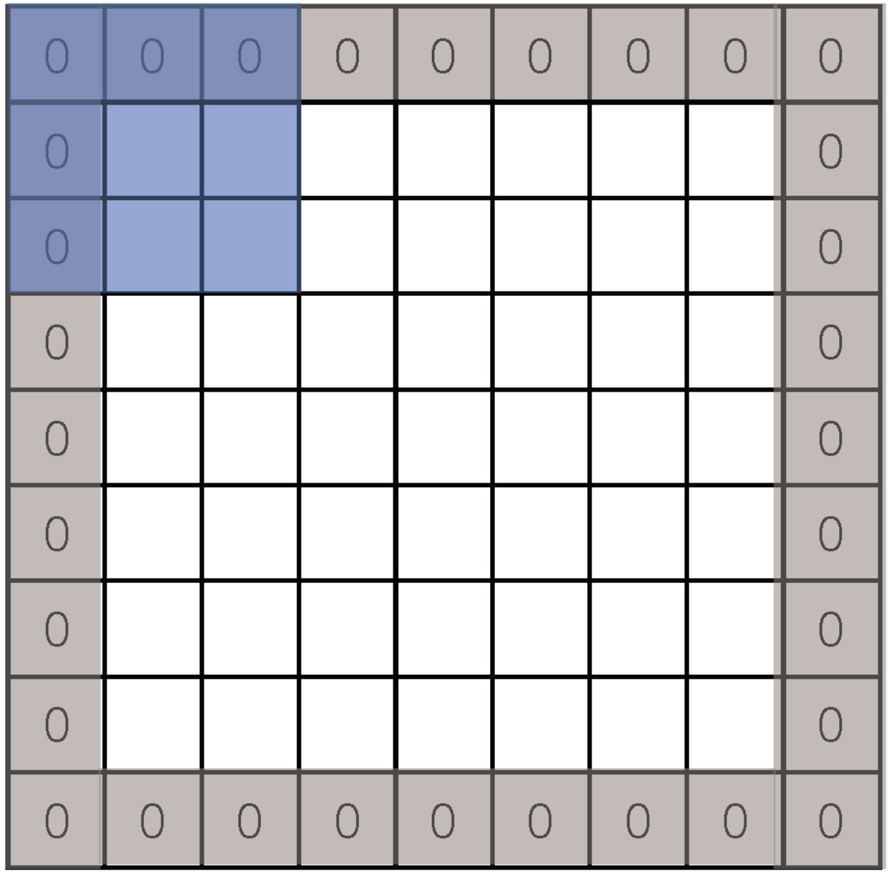



- *Notice that with certain strides, we may not be able to cover all of the input*

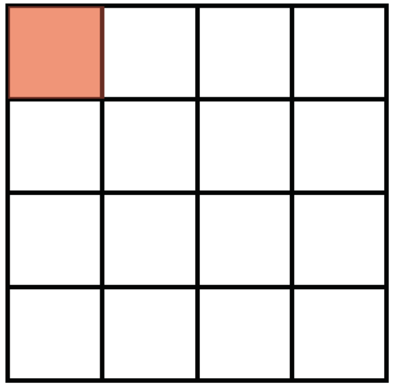
- *The output is also half the size of the input*

# Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



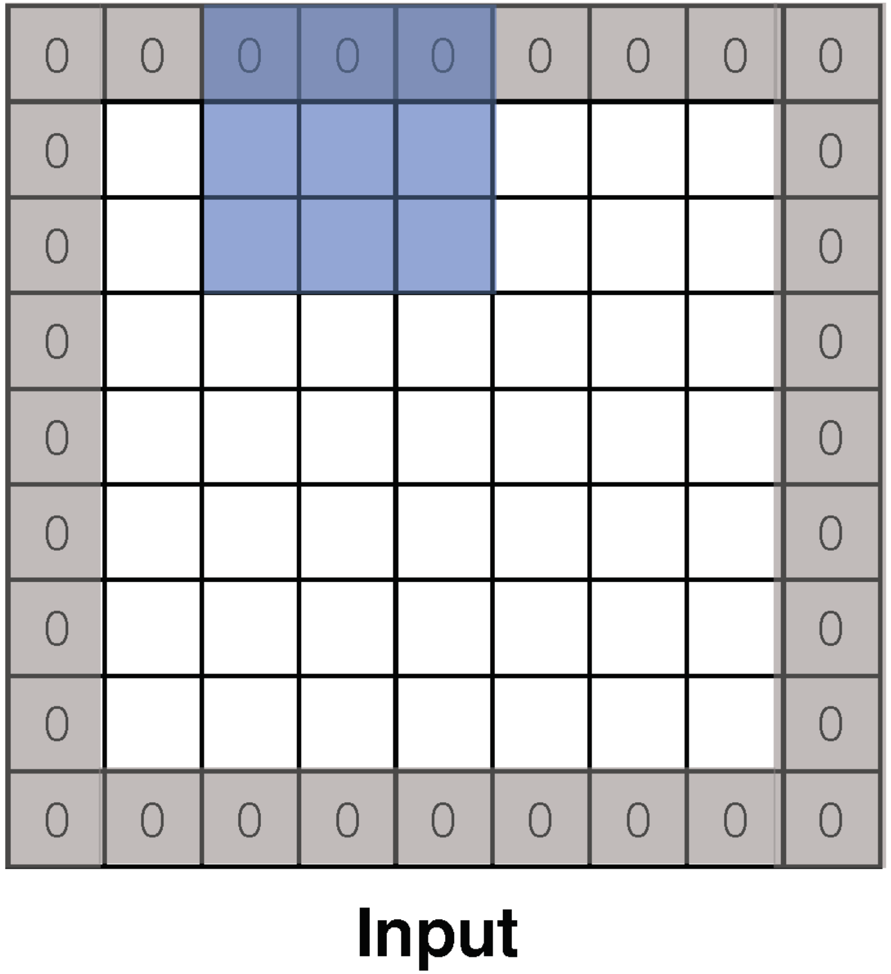

**Input**

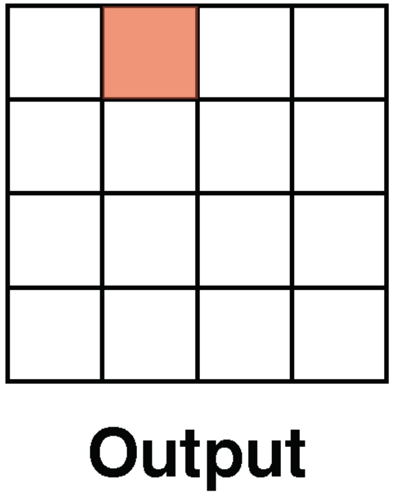


**Output**

# Convolution: Padding

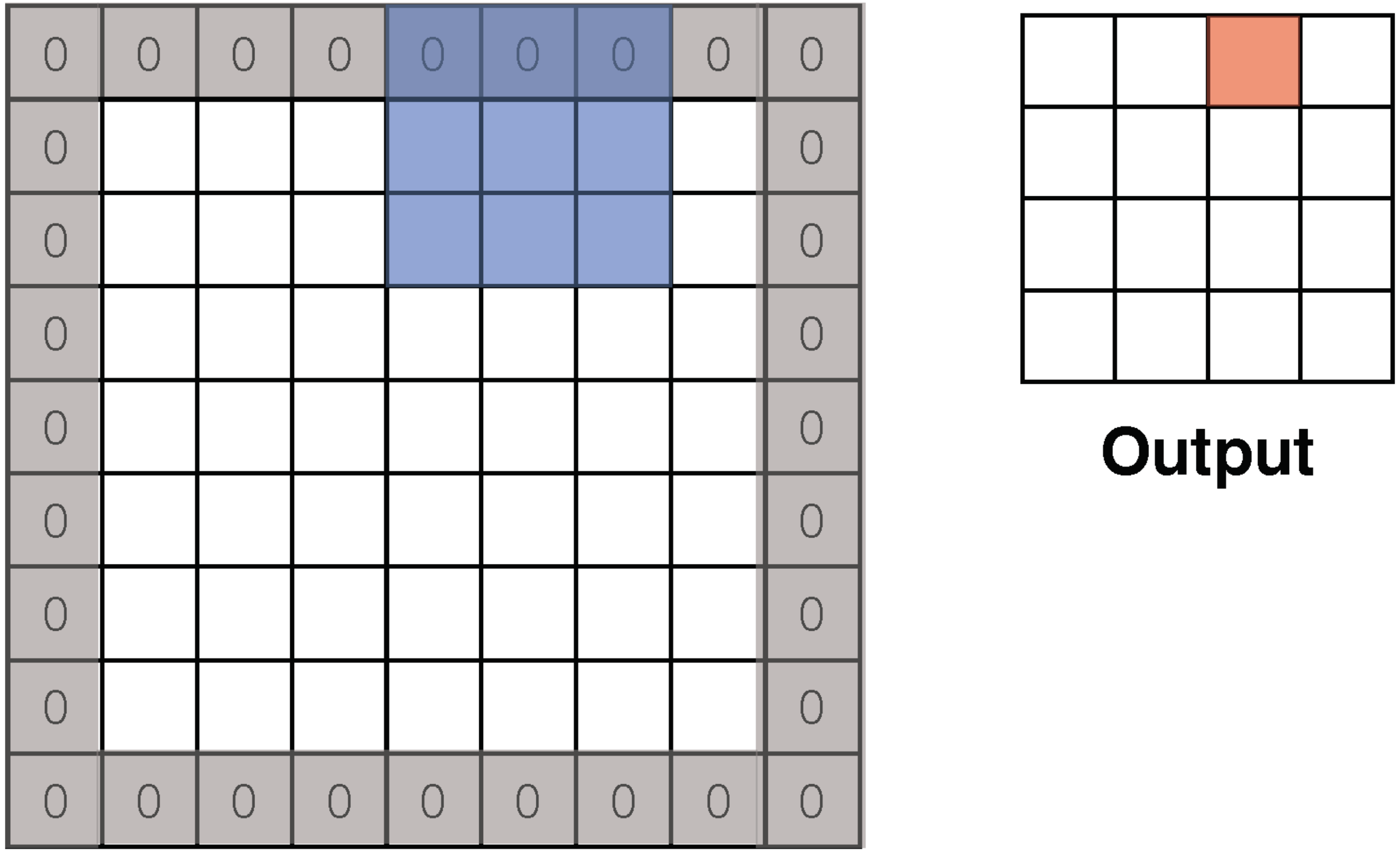
We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



# Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**

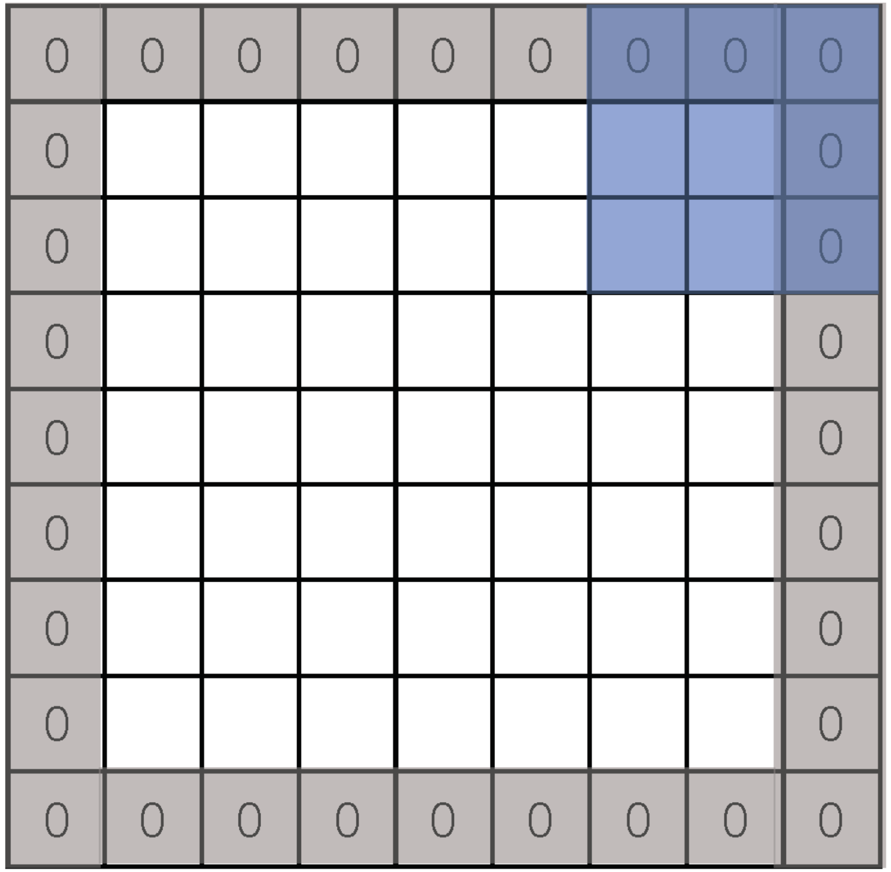


**Output**

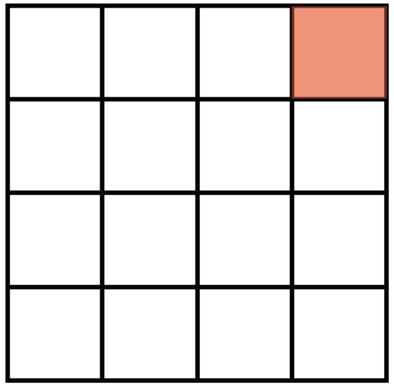
**Input**

# Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



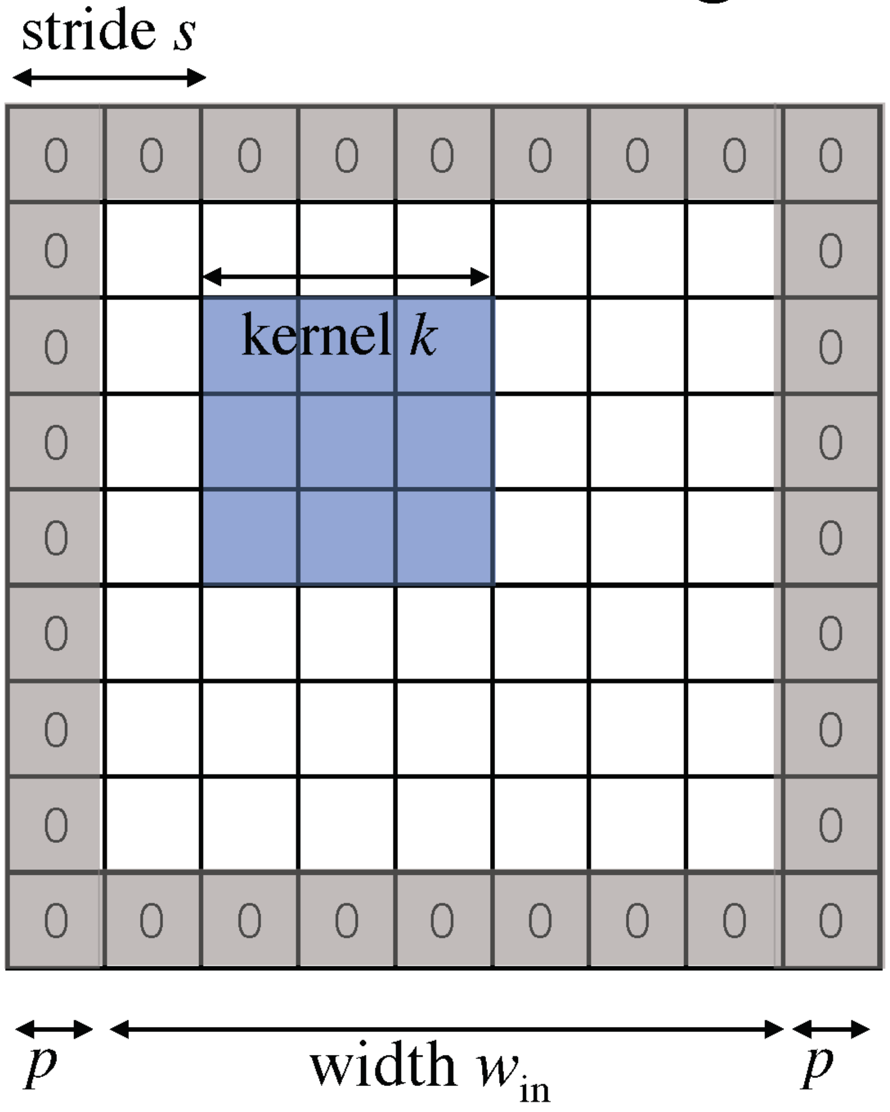

**Input**



**Output**

# Convolution:

## How big is the output?

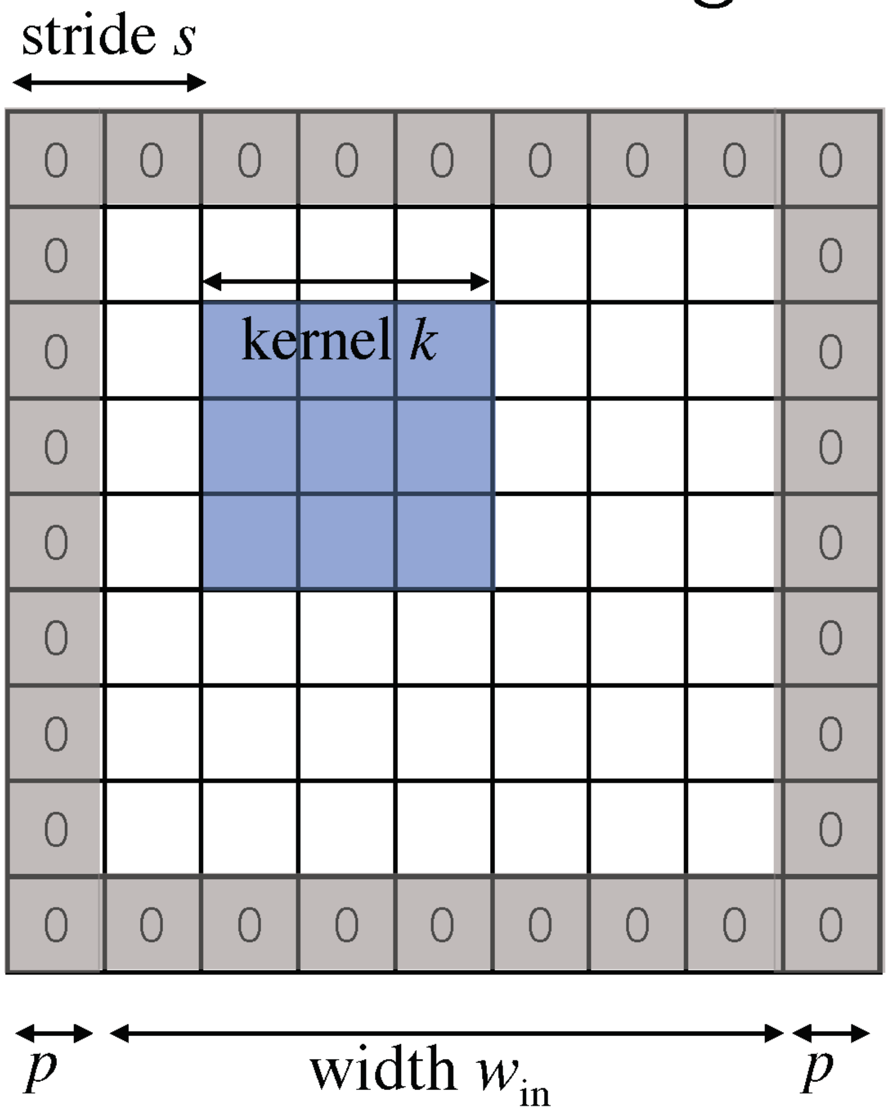



In general, the output has size:

$$w_{\text{out}} = \left\lfloor \frac{w_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

# Convolution:

## How big is the output?




**Example:** k=3, s=1, p=1

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

$$= \left\lfloor \frac{w_{in} + 2 - 3}{1} \right\rfloor + 1$$

$$= w_{in}$$

VGGNet [Simonyan 2014] uses filters of this shape

# Pooling

For most ConvNets, **convolution** is often followed by **pooling**:

- Creates a smaller representation while retaining the most important information
- The "max" operation is the most common
- Why might "avg" be a poor choice?

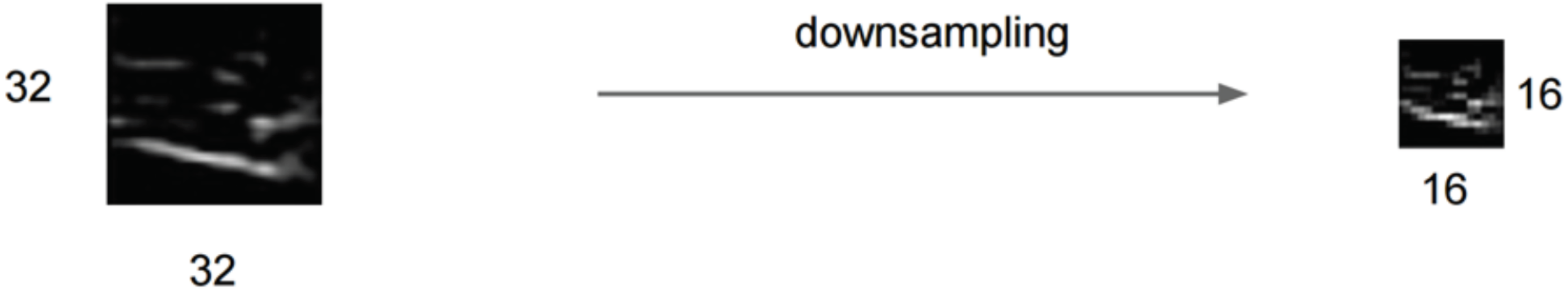
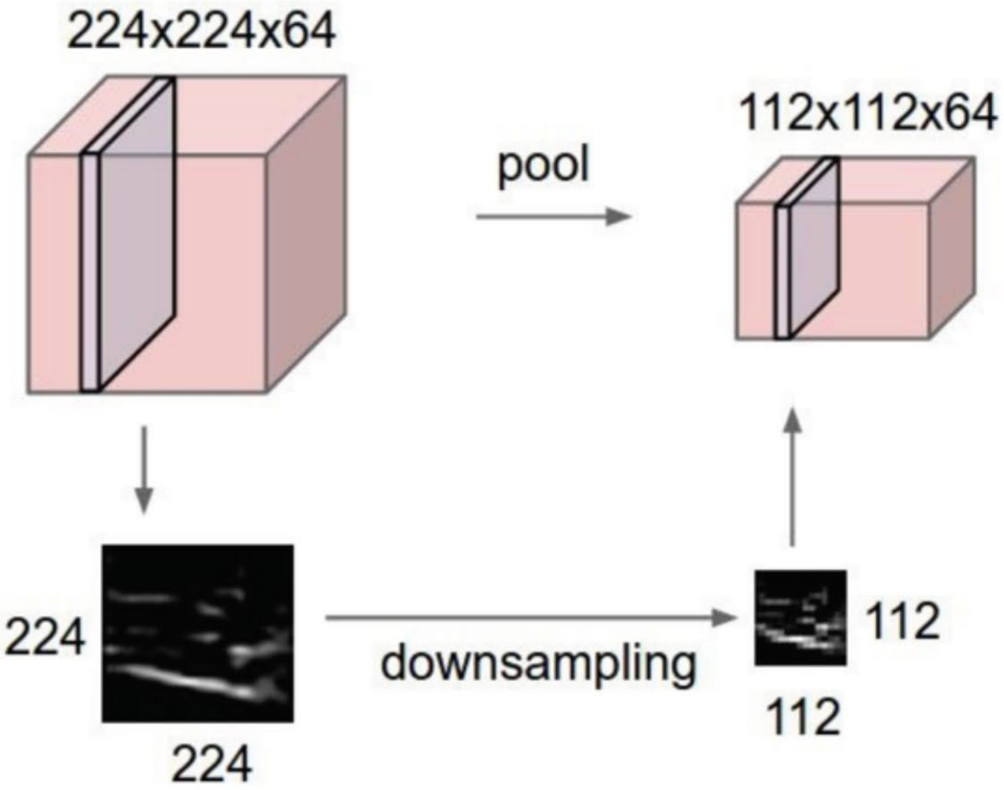


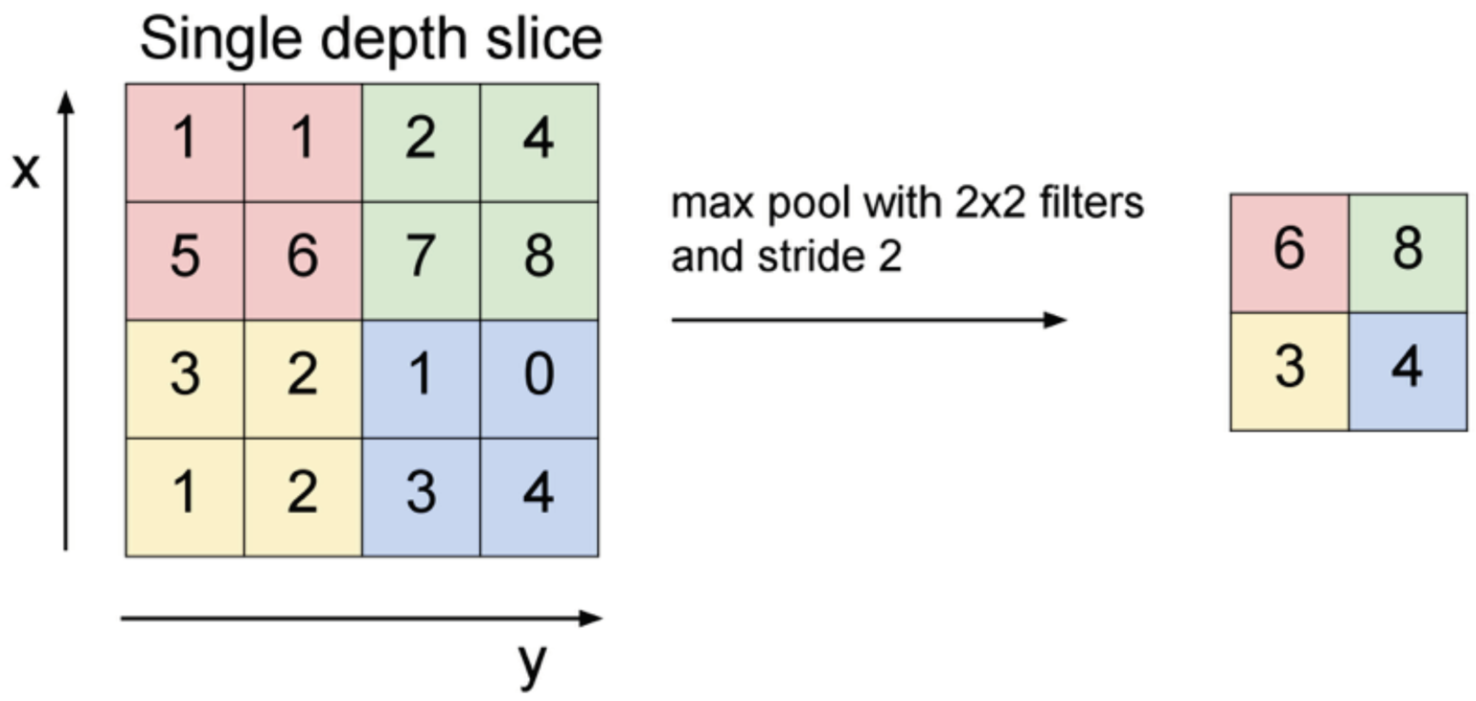

*Figure: Andrej Karpathy*

# Pooling

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



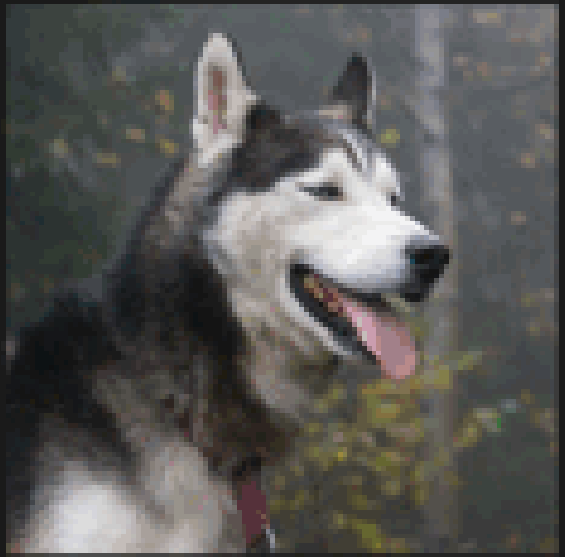
# Max Pooling

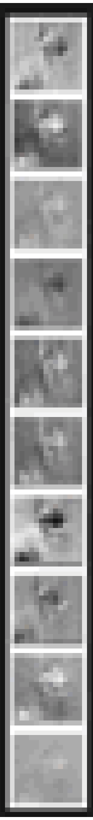



What’s the backprop rule for max pooling?

- In the forward pass, store the index that took the max
- The backprop gradient is the input gradient at that index

*Figure: Andrej Karpathy*





# Example ConvNet

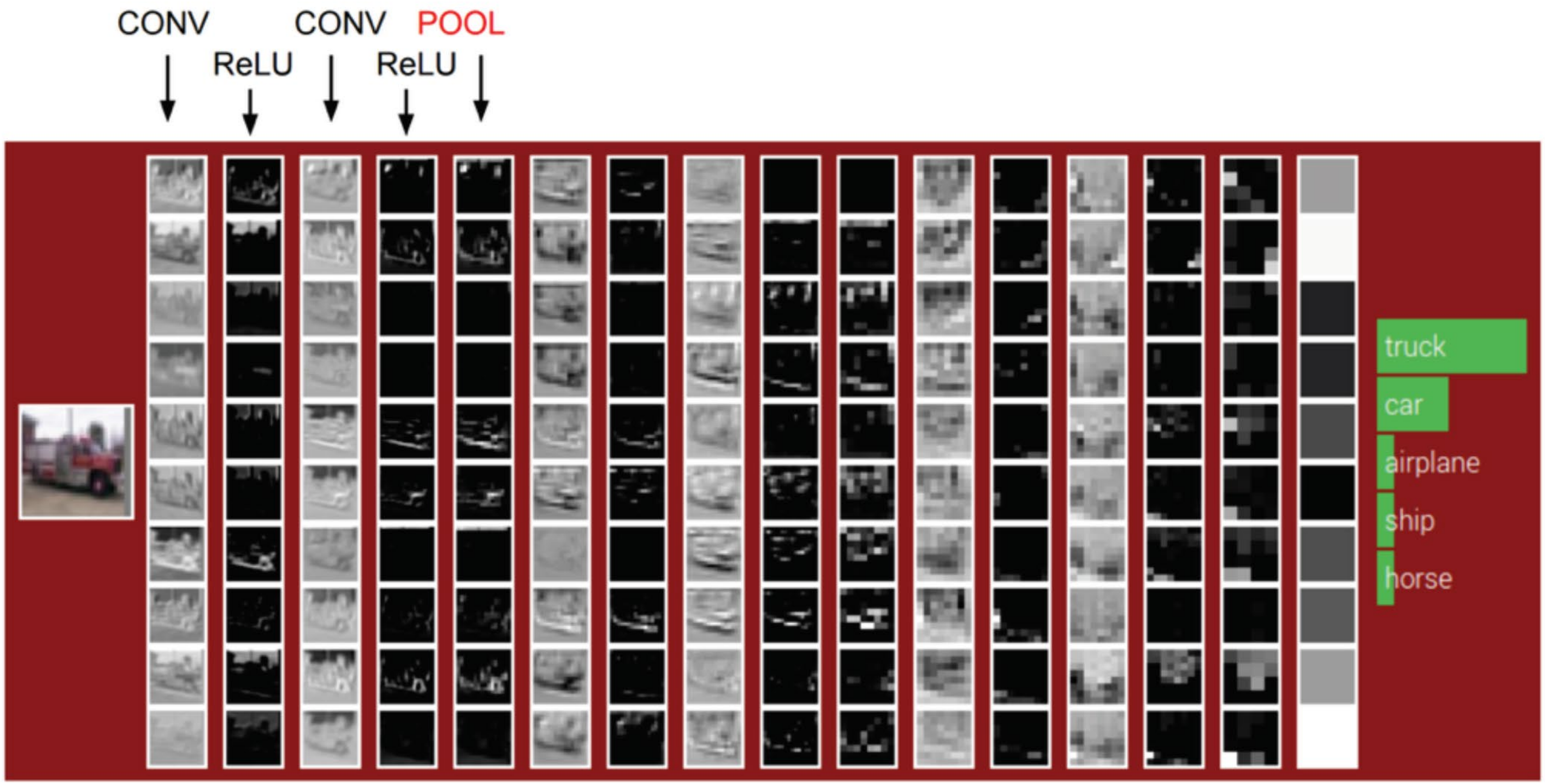



Figure: Andrej Karpathy

# Example ConvNet

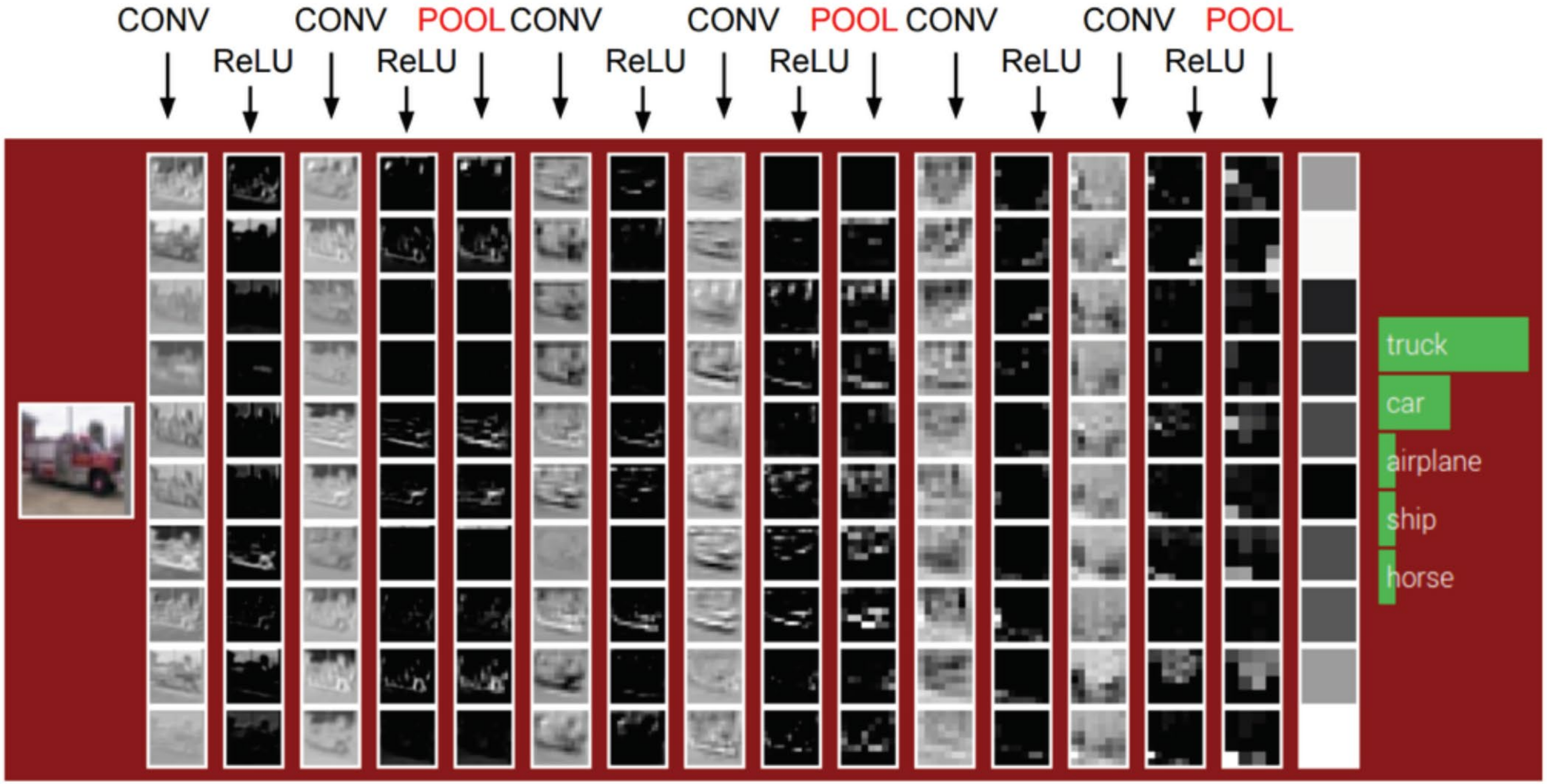



*Figure: Andrej Karpathy*

# Example ConvNet

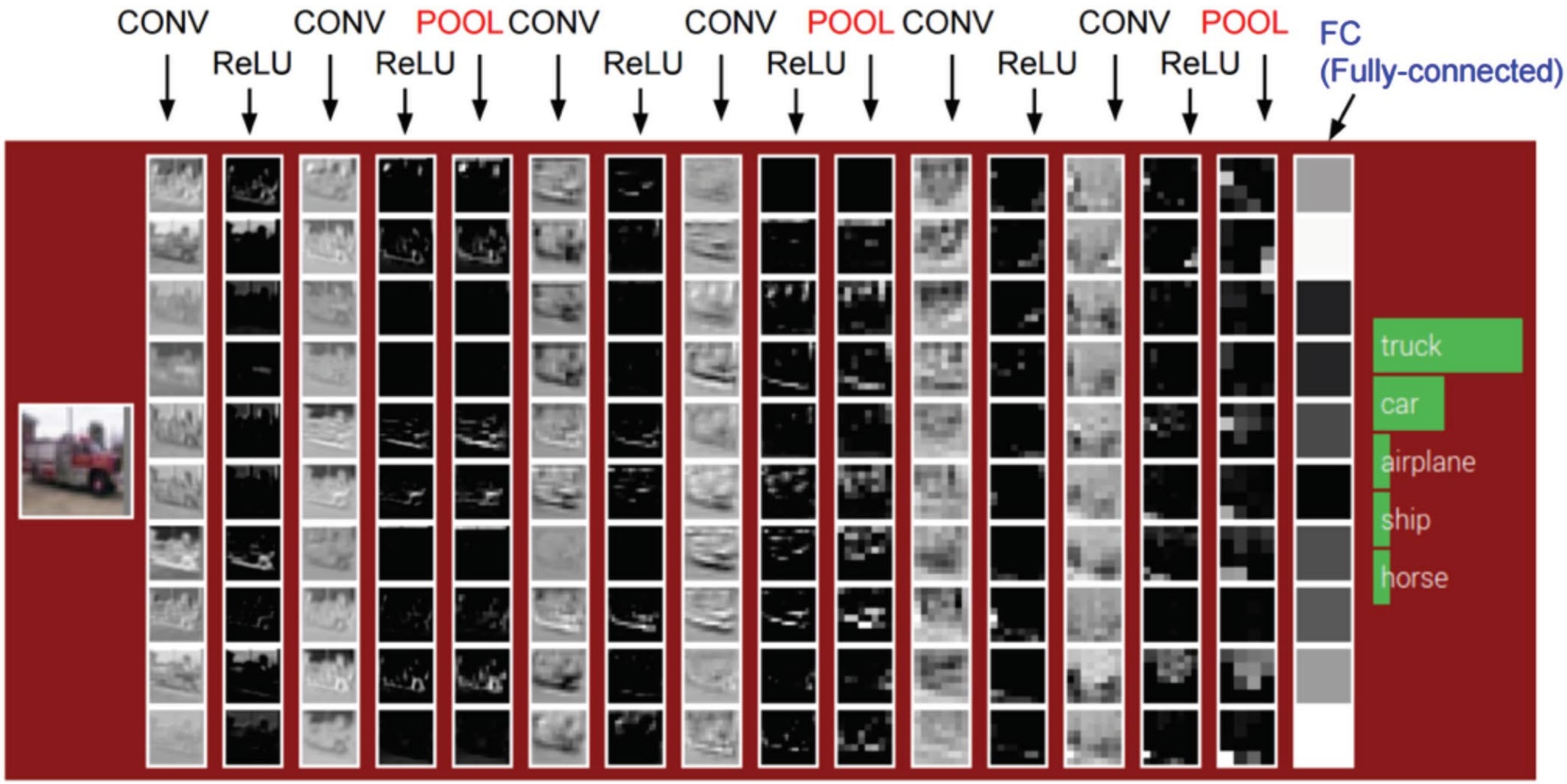
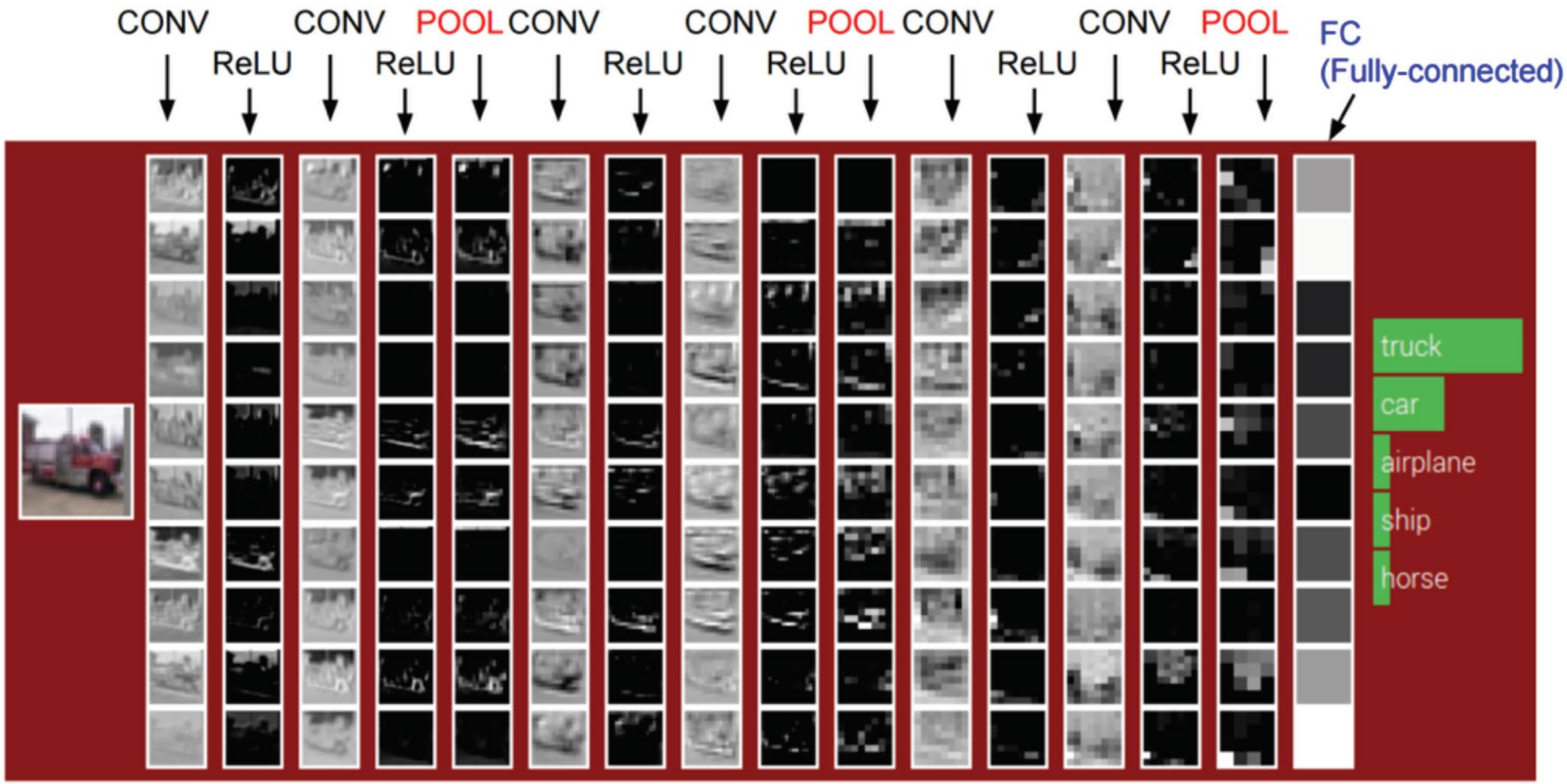



*Figure: Andrej Karpathy*

# Example ConvNet

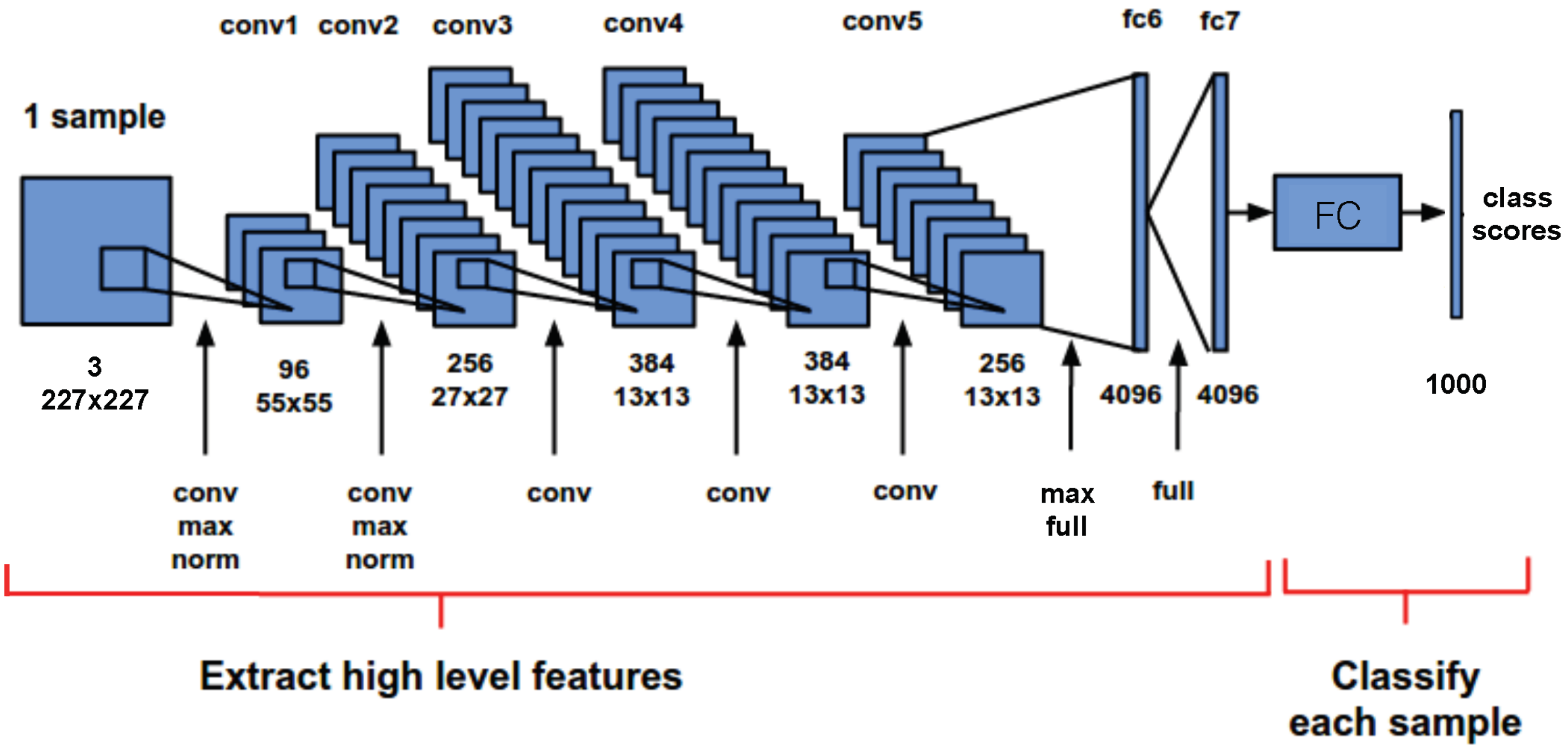



10x3x3 conv filters, stride 1, pad 1

2x2 pool filters, stride 2

*Figure: Andrej Karpathy*

# Example: AlexNet [Krizhevsky 2012]




“max”: max pooling
“norm”: local response normalization
“full”: fully connected

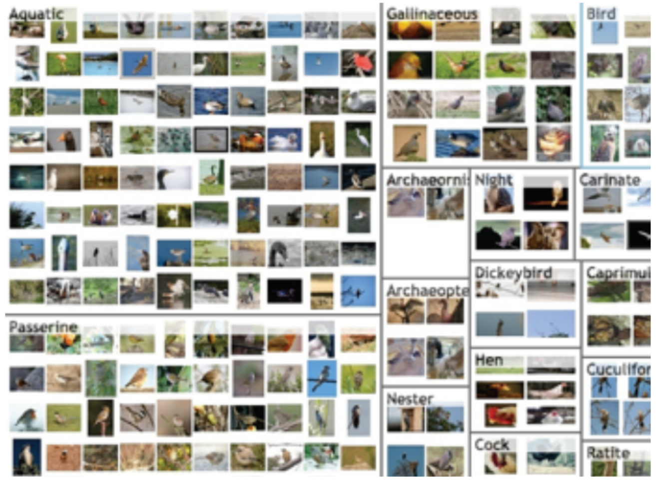
Figure: [Karnowski 2015] *(with corrections)*

# Training ConvNets

# How do you actually train these things?

**Roughly speaking:**

Gather labeled data




Find a ConvNet architecture

Minimize the loss

# Training a convolutional neural network

- Split and preprocess your data
- Choose your network architecture
- Initialize the weights
- Find a learning rate and regularization strength
- Minimize the loss and monitor progress
- Fiddle with knobs

# Mini-batch Gradient Descent

**Loop:**

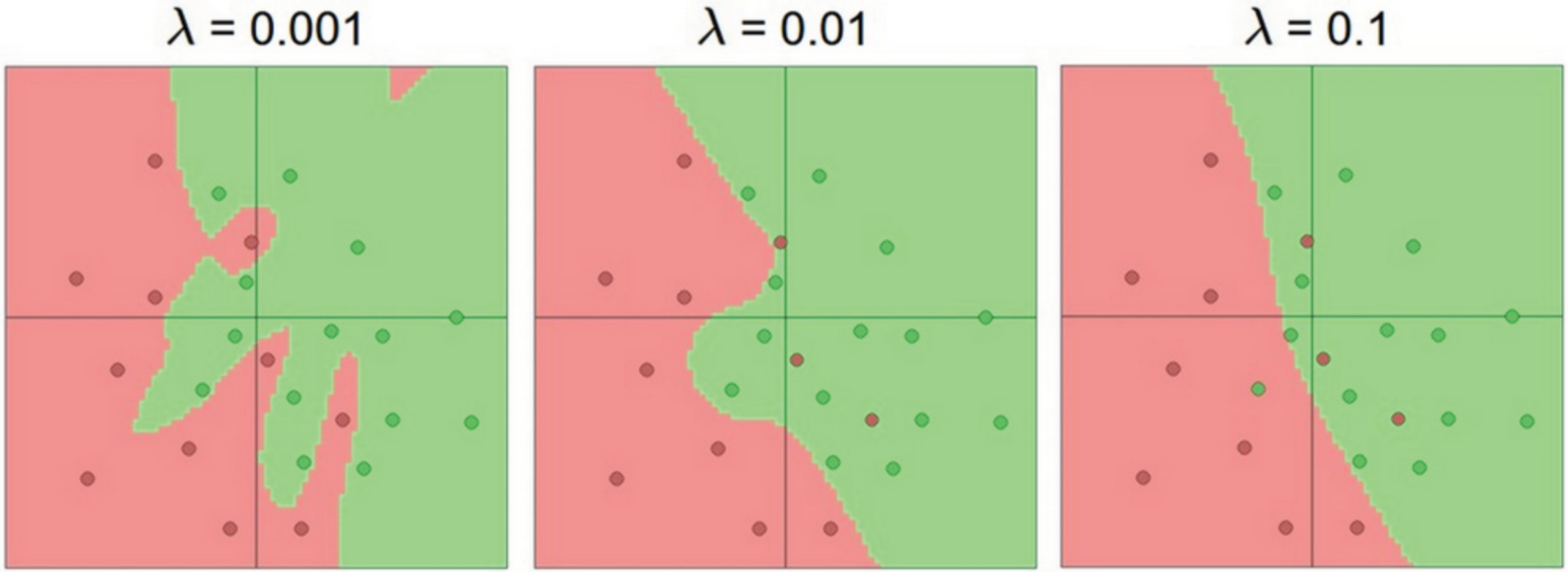
1. Sample a batch of training data (~100 images)
2. Forwards pass: compute loss (avg. over batch)
3. Backwards pass: compute gradient
4. Update all parameters

**Note:** usually called "stochastic gradient descent" even though SGD has a batch size of 1

# Regularization

**Regularization reduces overfitting:**

$$L = L_{\text{data}} + L_{\text{reg}} \qquad L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2$$



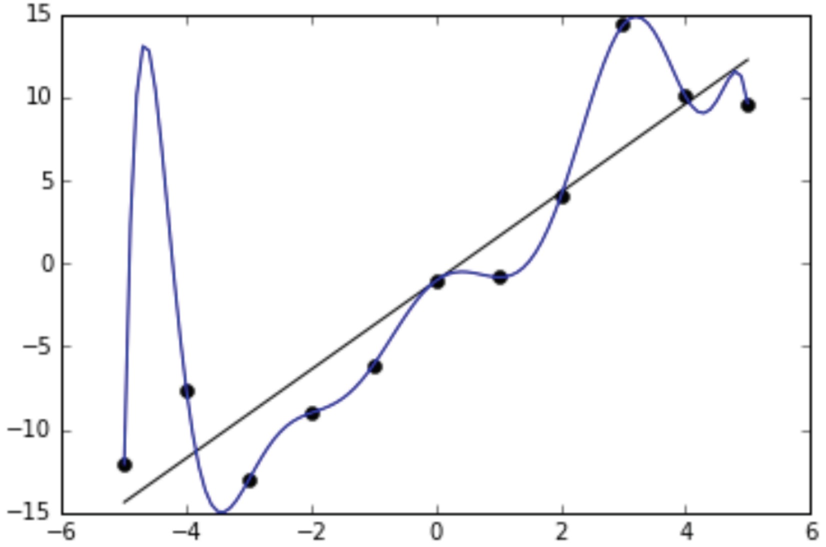

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

# Overfitting

**Overfitting:** modeling noise in the training set instead of the "true" underlying relationship

**Underfitting:** insufficiently modeling the relationship in the training set

**General rule:** models that are "bigger" or have more capacity are more likely to overfit




[Image: https://en.wikipedia.org/wiki/File:Overfitted_Data.png]

# Summary of things to fiddle

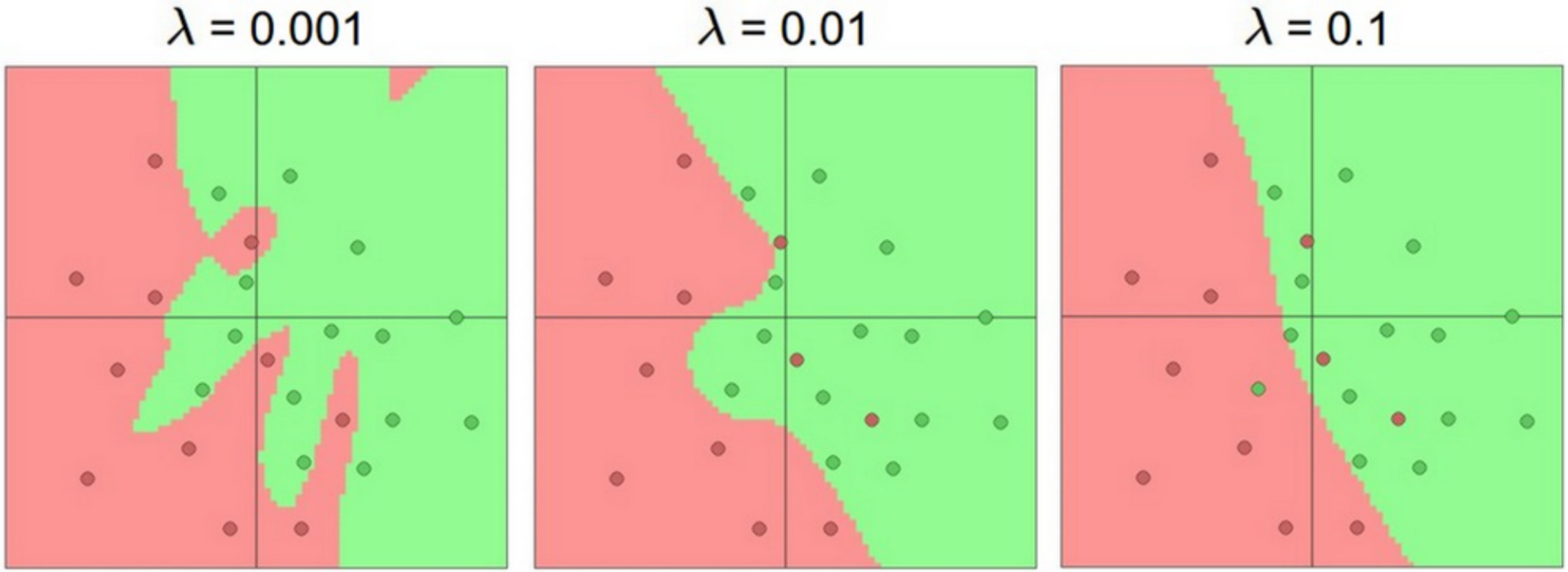
- Network architecture
- Learning rate, decay schedule, update type
- Regularization (L2, L1, maxnorm, dropout, …)
- Loss function (softmax, SVM, …)
- Weight initialization

Neural network parameters

# (Recall) Regularization reduces overfitting

$$L = L_{\text{data}} + L_{\text{reg}}$$

$$L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2$$




[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

# Example Regularizers

**L2 regularization**

$$L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2$$

(L2 regularization encourages small weights)

**L1 regularization**

$$L_{\text{reg}} = \lambda \|W\|_1 = \lambda \sum_{ij} |W_{ij}|$$

(L1 regularization encourages sparse weights:
weights are encouraged to reduce to exactly zero)

**“Elastic net”**

$$L_{\text{reg}} = \lambda_1 \|W\|_1 + \lambda_2 \|W\|_2^2$$

(combine L1 and L2 regularization)

**Max norm**

Clamp weights to some max norm

$$\|W\|_2^2 \le c$$

# "Weight decay"

**Regularization is also called "weight decay" because the weights "decay" each iteration:**

$$L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2 \longrightarrow \frac{\partial L}{\partial W} = \lambda W$$

Gradient descent step:

$$W \leftarrow W - \alpha \lambda W - \frac{\partial L_{\text{data}}}{\partial W}$$

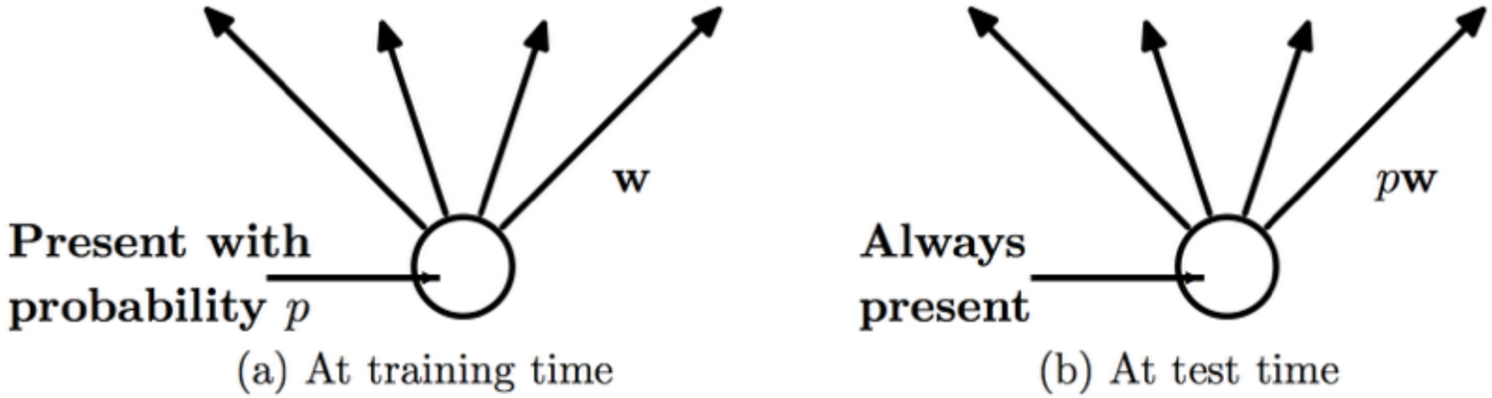
Weight decay: $\alpha\lambda$ (weights always decay by this amount)

**Note:** biases are sometimes excluded from regularization

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

# Dropout

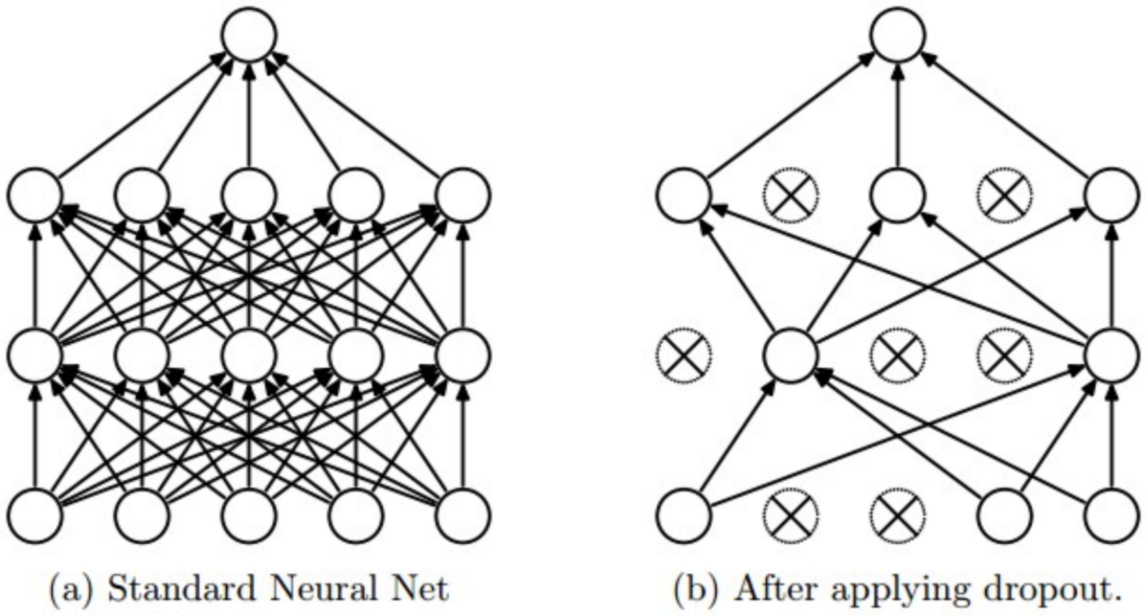
**Simple but powerful technique to reduce overfitting:**




[Srivasta et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014]

# Dropout

**Simple but powerful technique to reduce overfitting:**




**Note:** Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models

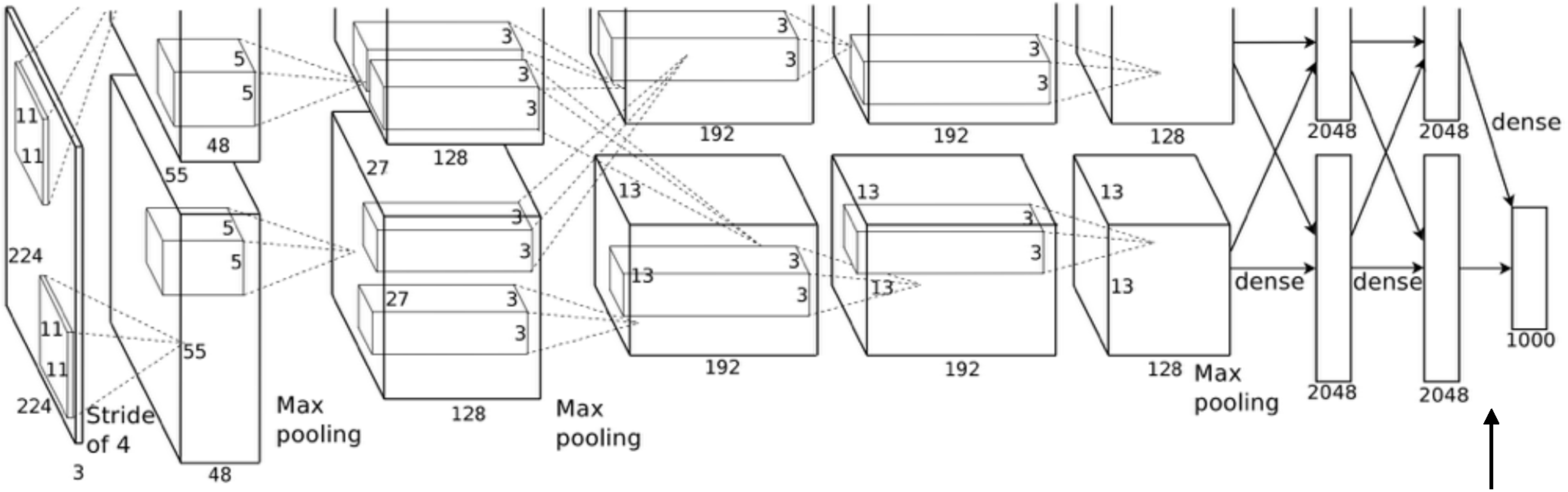
[Srivasta et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014]

# Dropout

**Case study: [Krizhevsky 2012]**

*"Without dropout, our network exhibits substantial overfitting."*

Dropout here




**But not here — why?**

[Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012]

# Summary

- Preprocess the data (subtract mean, sub-crops)
- Initialize weights carefully
- Use Dropout
- Use SGD + Momentum
- Fine-tune from ImageNet
- Babysit the network as it trains

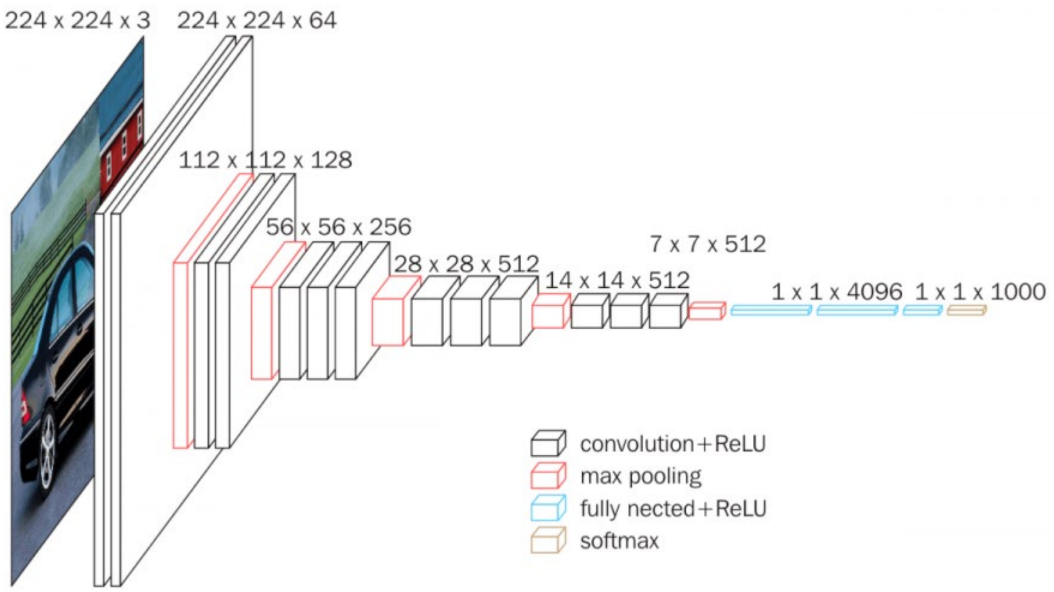
# Common Architectures

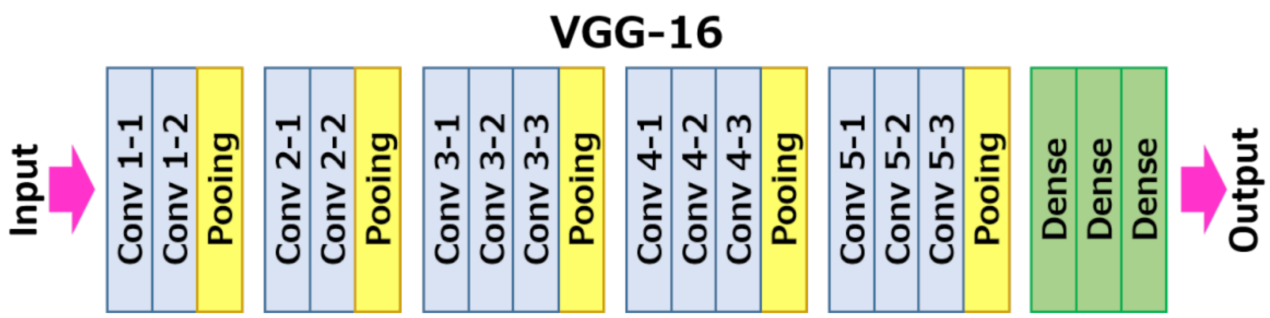
# VGG

- Introduced by K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”

- One of the most common CNN architectures used

- Also typically used for feature extraction



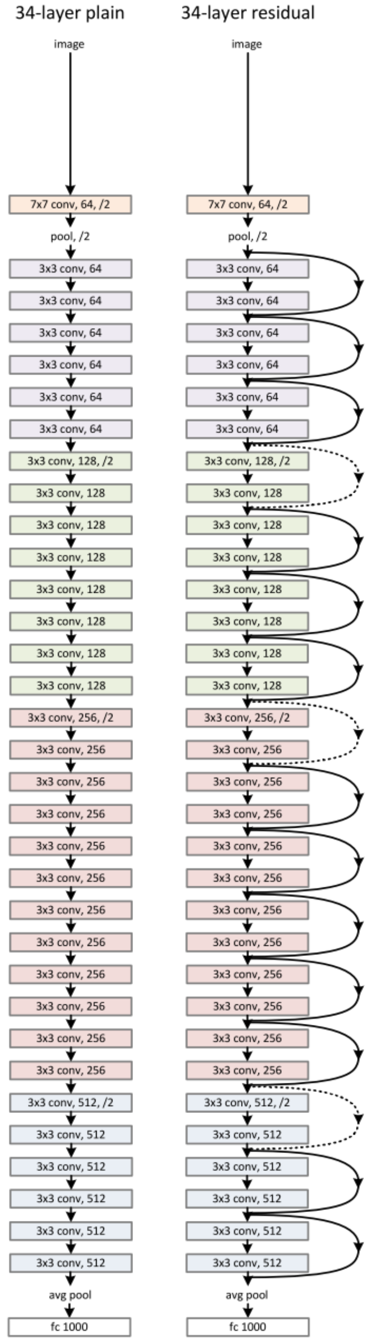
# ResNet

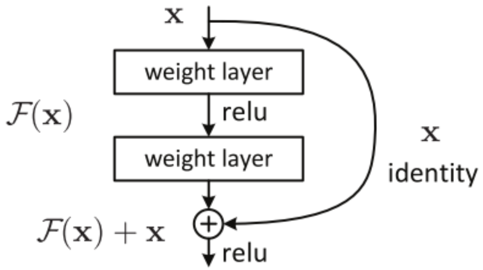
- He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2016). ["Deep Residual Learning for Image Recognition"](#) (PDF). Proc. Computer Vision and Pattern Recognition (CVPR), IEEE.

- Deep networks with more layers does not always mean better performance (vanishing gradient problem)

- Residual blocks = has skip connections

- Skipped layers train faster at the beginning, then later are filled out



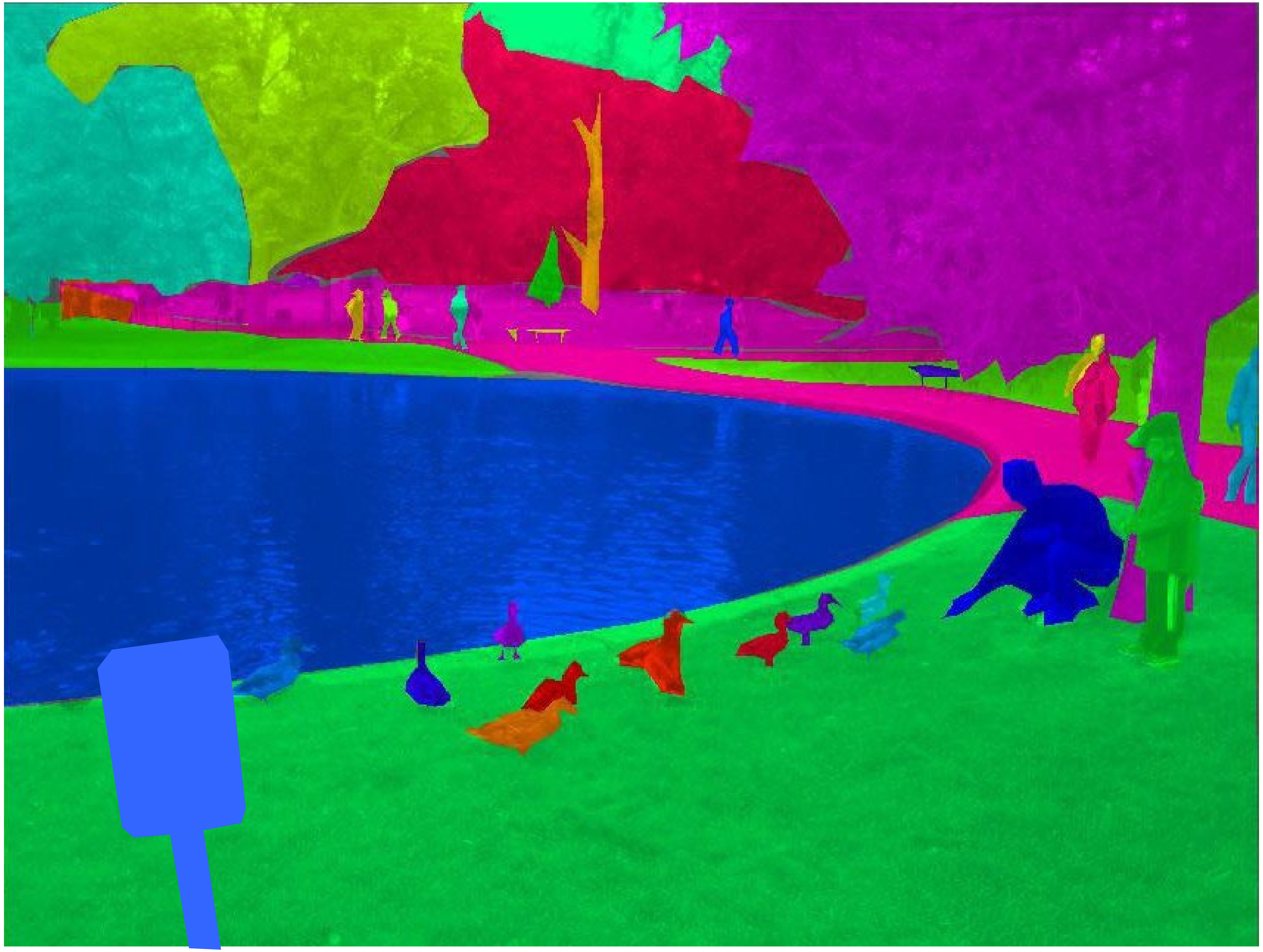



# Today

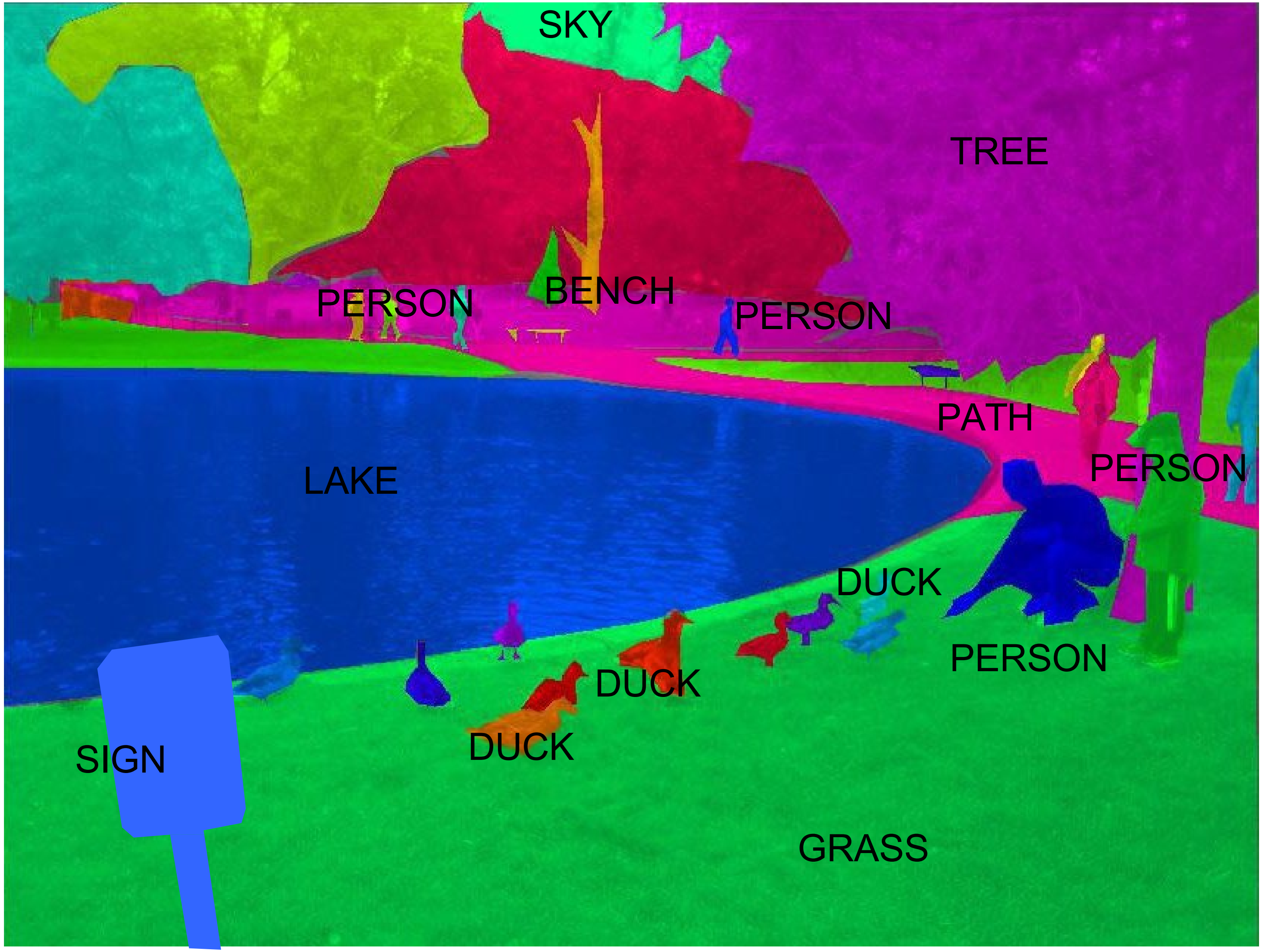
- **Introduction to scene understanding**
- Object detection models
- Evaluating object detectors
- Future challenges


Image contains Photoshopped sign

Source: Antonio Torralba



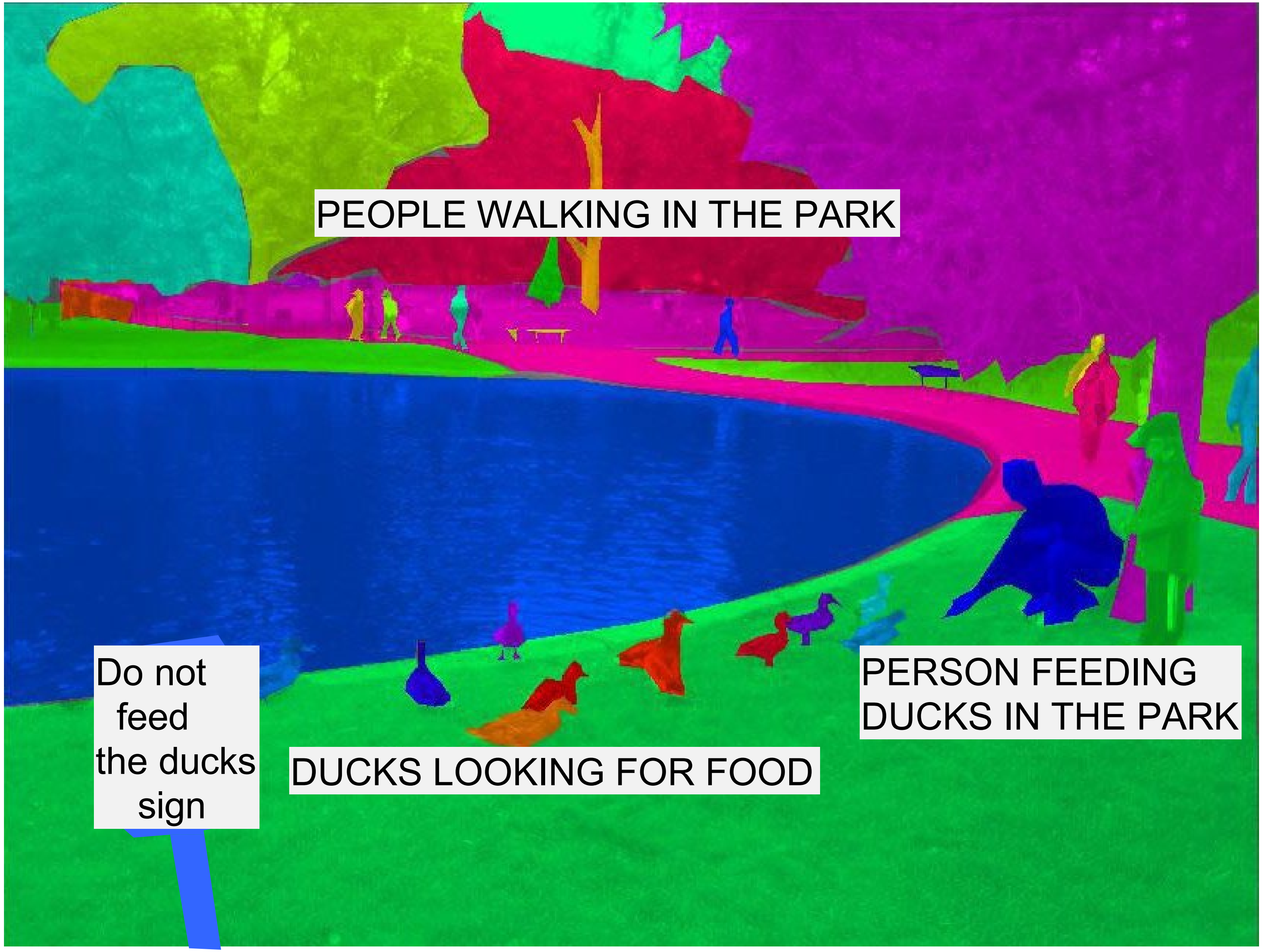
Label each pixel as a category. Each category has a unique color.




Label each pixel as a category. Each category has a unique color.


Scene-Level Classification:  This is a “PARK”


Image Captioning:  Describe the image in human language (e.g. English)



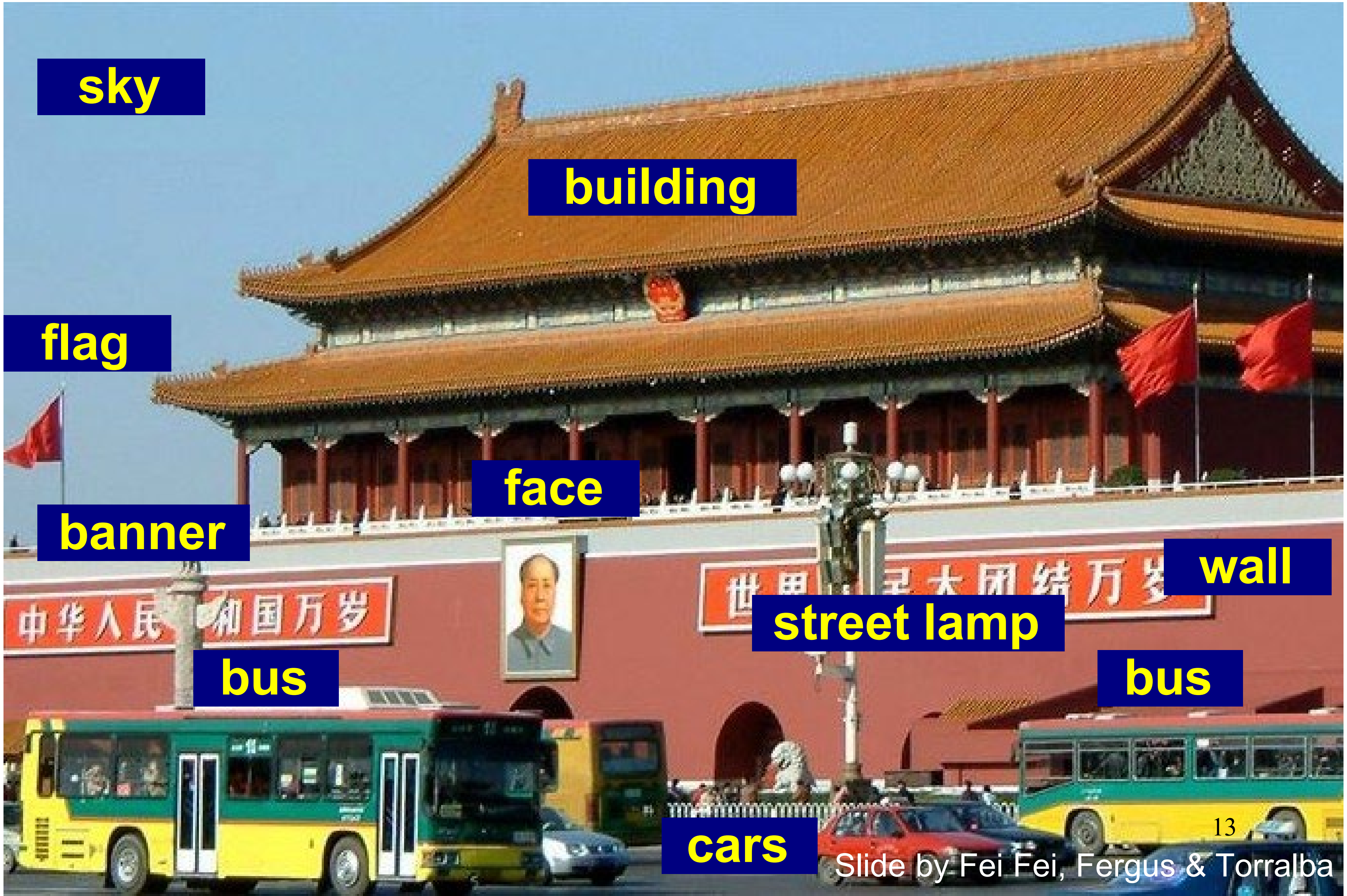

Dense Image Captioning: Describe several parts of the image

# What makes this challenging?

# Why do we care about recognition?

- The concept of **“categories”** encapsulates semantic information that humans use when communicating with each other.
- Categories are also linked with what can we do with those objects.

# Object categories aren’t everything




Slide by Fei Fei, Fergus & Torralba

# Object categories aren't everything

*A picture is worth a 1000 words…*

*Or just 10?*

sky

building

flag

face

banner

wall

street lamp

bus

bus

cars

Source: A. Efros

# How finegrained should categories be ?

A Beijing City Transit Bus #17, serial number 43253?

Source: A. Efros

# Need more general (useful) information

*What can we say the very first time we see this thing?*

**Functional:**

- A large vehicle that may be moving fast, probably to the right, and will hurt you if you stand in its way.
- However, at specified places, it will allow you to enter it and transport you quickly over large distances.

**Communicational:**

- bus, autobus, λεωφορείο, ônibus, автобус, 公共汽车, etc.

Source: A. Efros

# Visual challenges with categories

Chair



- A lot of categories are functional

car


- Categories are 3D, but images are 2D

- World is highly varied


train

Source: A. Efros

# Limits to direct perception


# Importance of context

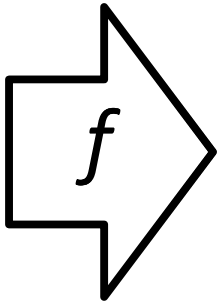
 

Source: Antonio Torralba

# Today

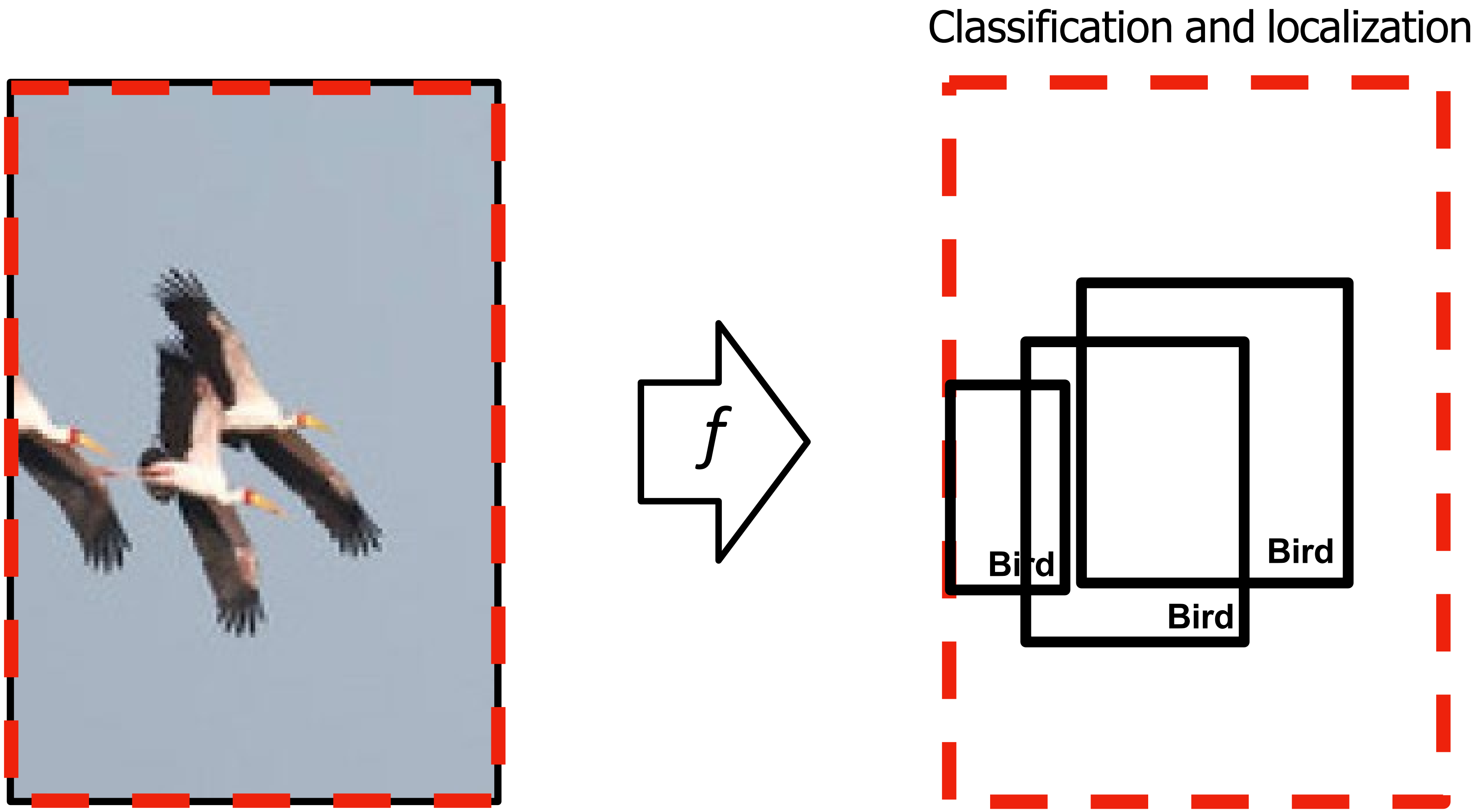
- Introduction to scene understanding
- **Object detection models**
- Evaluating object detectors
- Future challenges

# Image Classification




"Birds"

# Object detection

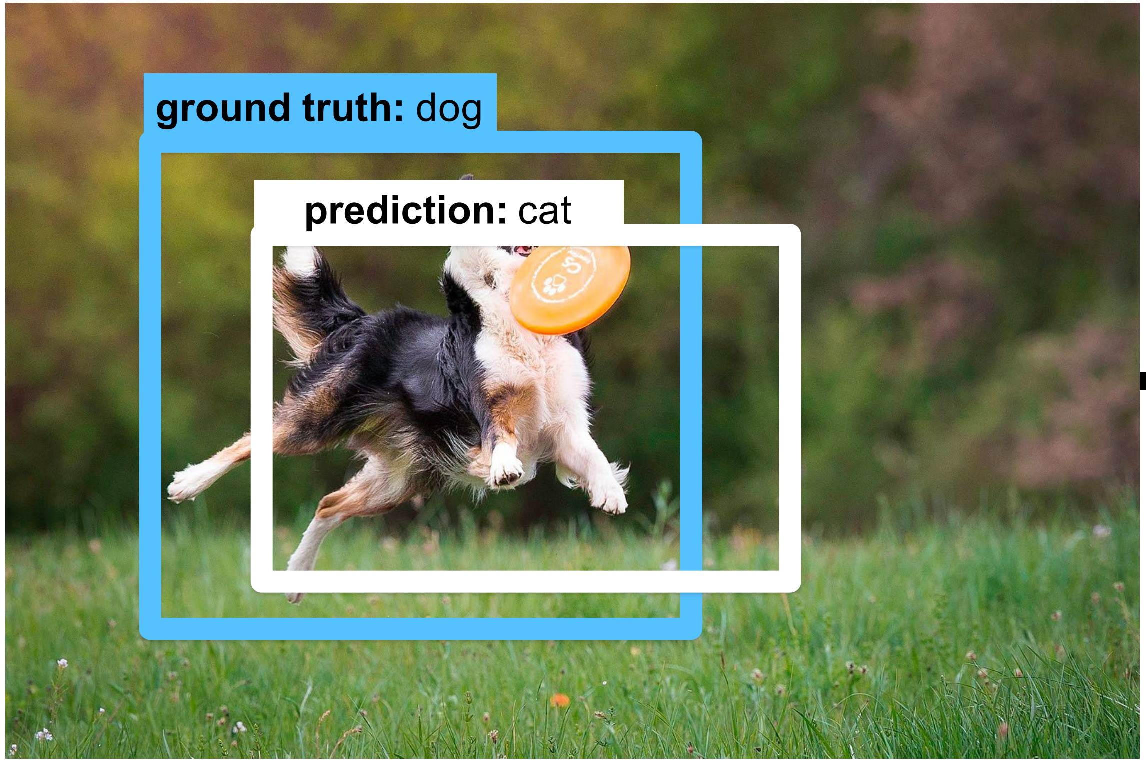


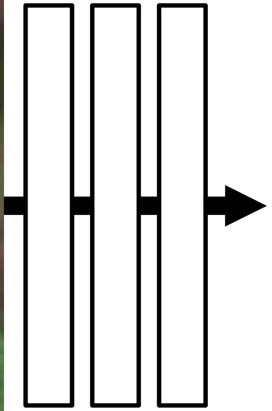

Each bounding box is:
[x,y,w,h]

Challenge: unbounded number of detections, possibly multiple detections per pixel

# Idea #1: regress bounding box

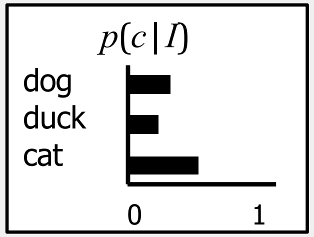
# Idea #1: regress bounding box



**Outputs**

**1. Class label**




**2. Box coords.**

$$(x, y, w, h)$$

**Losses**

**1. Cross entropy loss**

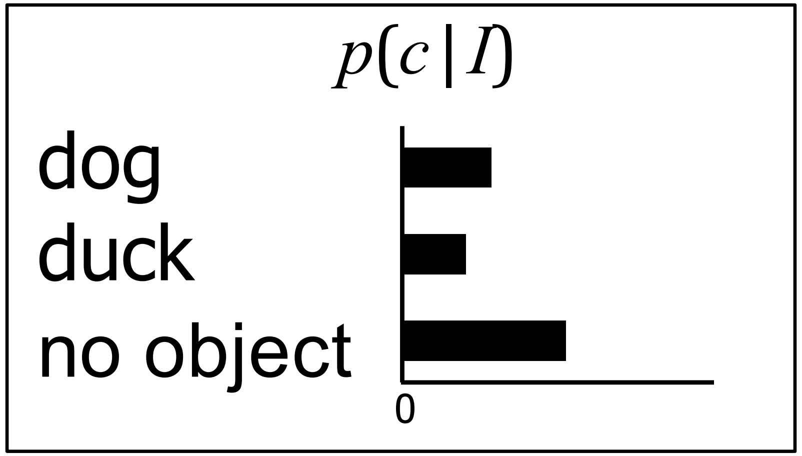
$$L_{cls} = -\log(p(y = \text{dog}))$$

**2. Squared distance**

$$L_{box} = \left\| \begin{bmatrix} x \\ y \\ w \\ h \end{bmatrix} - \begin{bmatrix} x_{gt} \\ y_{gt} \\ w_{gt} \\ h_{gt} \end{bmatrix} \right\|^2$$

**Doesn’t scale well to multiple objects.**

# Idea #2: sliding window

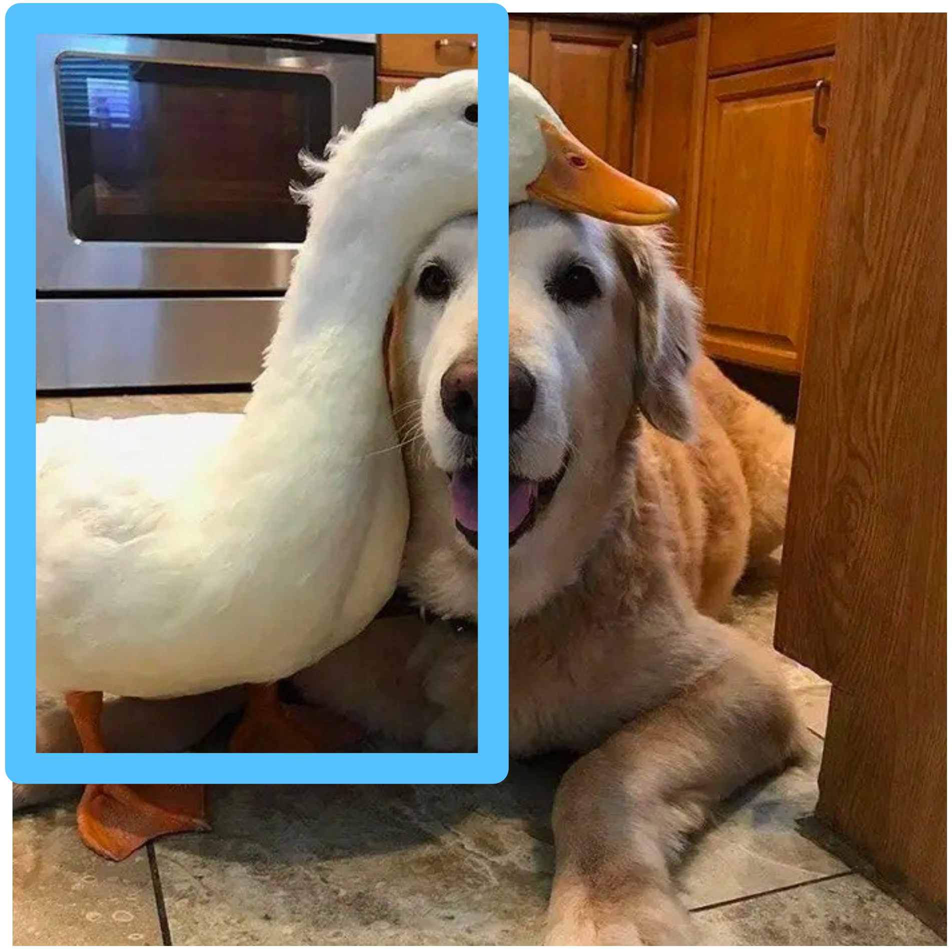


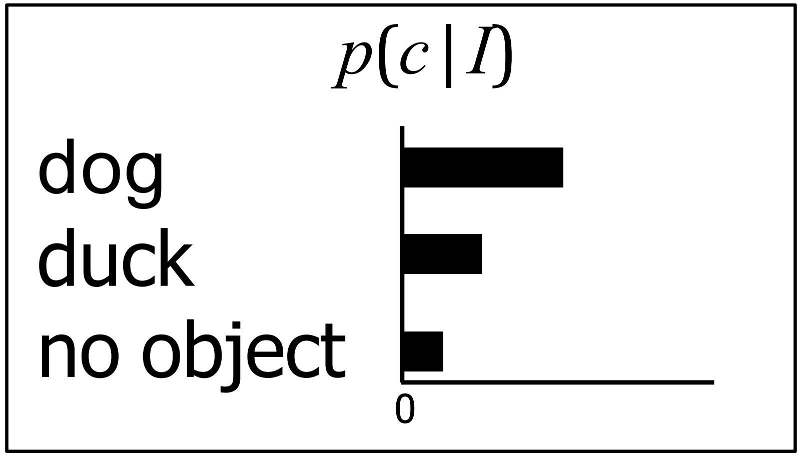

Bounding box

$(x, y, w, h)$

Need multiple scales and aspect ratios

# Idea #2: sliding window

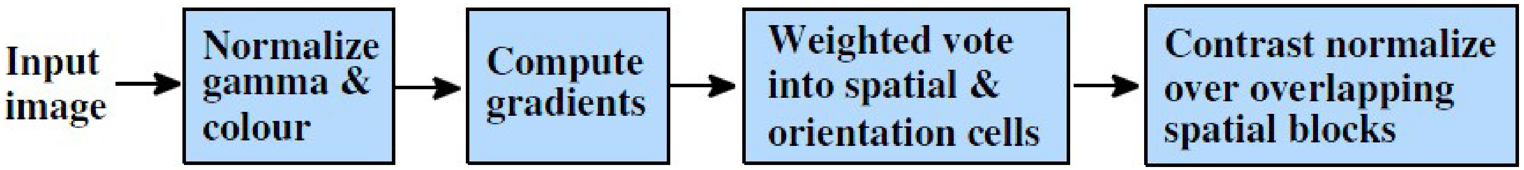





Bounding box

$(x, y, w, h)$

# Example: histograms of oriented gradients (HOG)



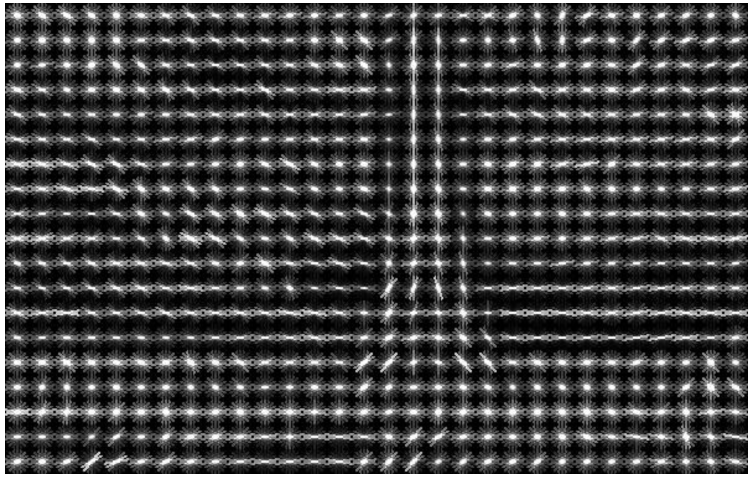



Image credit: N. Snavely

N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Source: S. Lazebnik

# Example: pedestrian detection with HOG

Train a pedestrian template using a linear classifier. Represent each window using HOG.

**positive training examples**

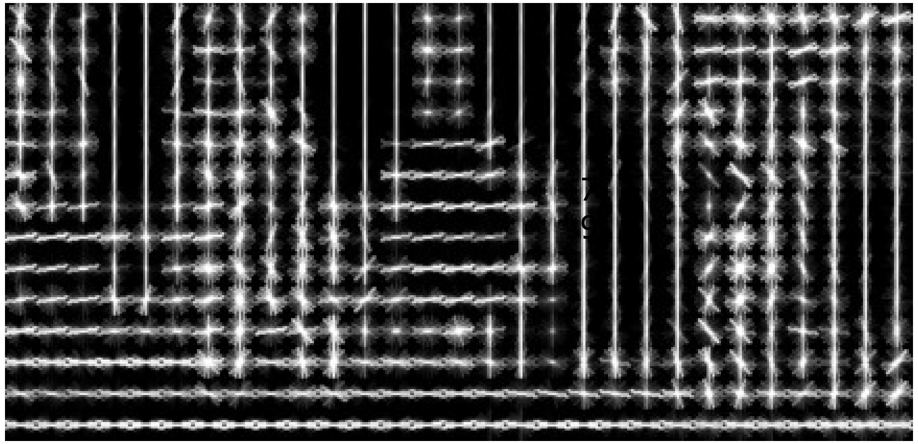
**negative training examples**

N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005 Source: S. Lazebnik

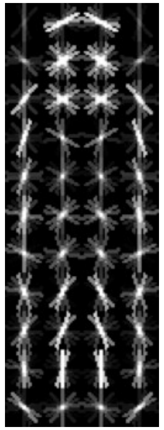
# Pedestrian detection with HOG

For multi-scale detection, repeat over multiple levels of a HOG pyramid

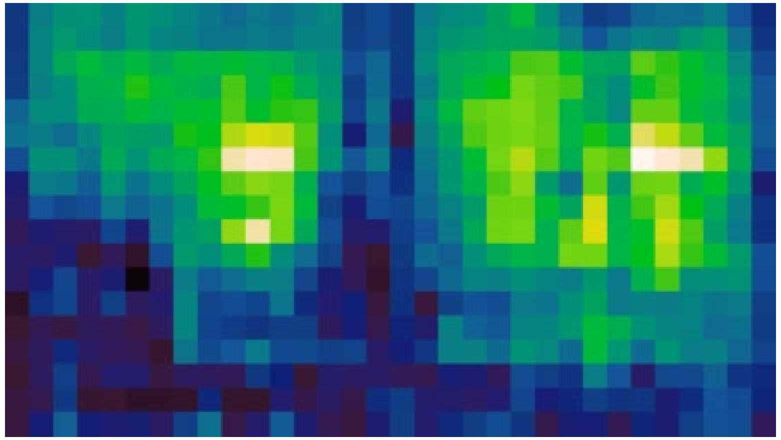
HOG feature map



Template

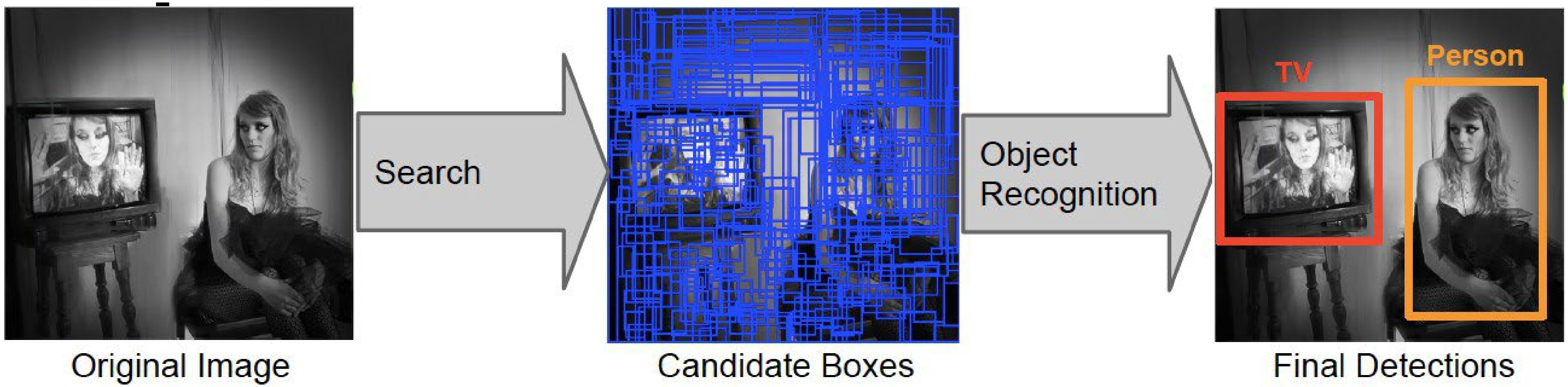


Detector response map



Source: S. Lazebnik

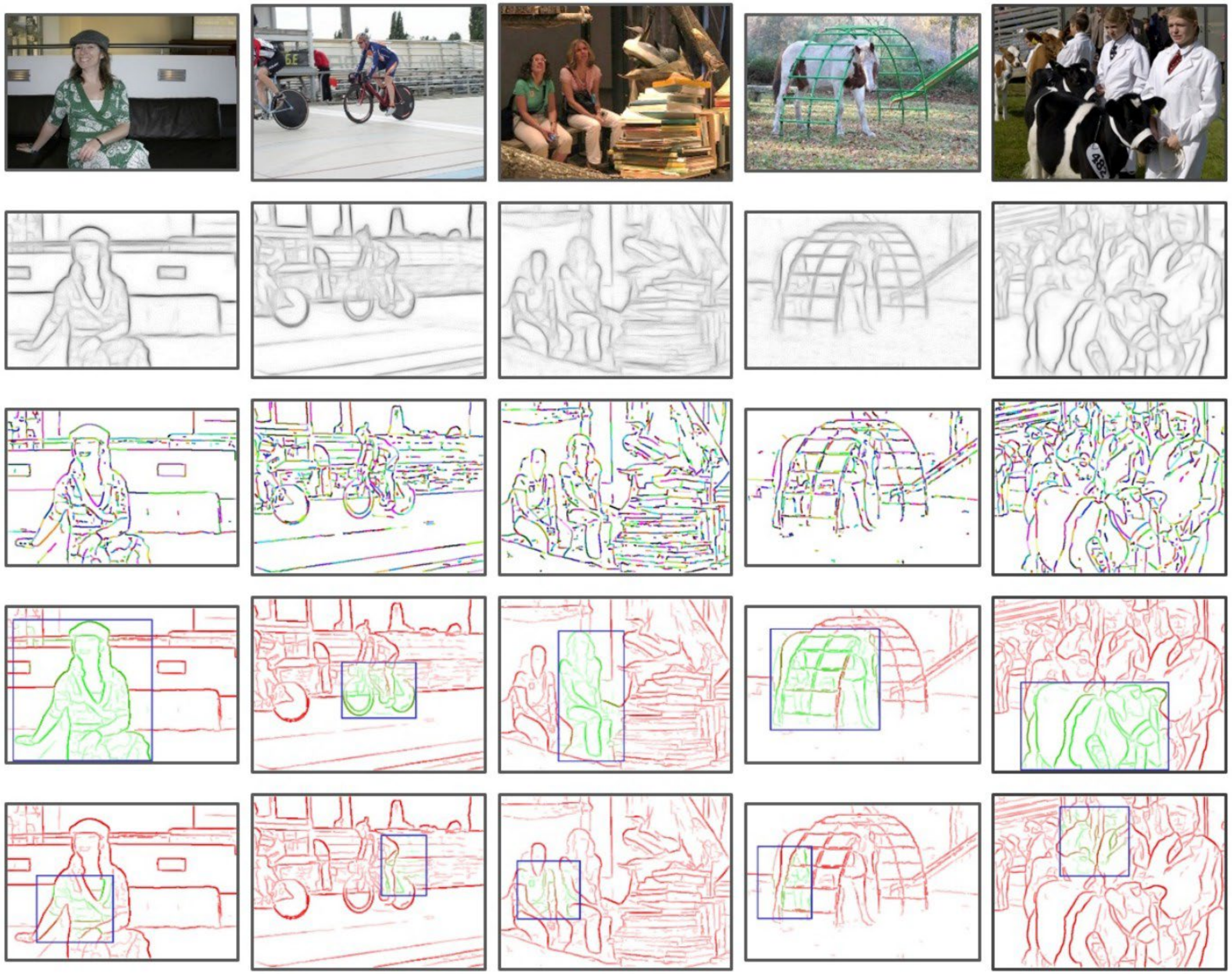
# Idea #3:




- Problem: evaluating a detector is very expensive
  - An image with pixels has $O(n^2)$ windows
- Only generate and evaluate a few hundred **region proposals** for regions that are "likely" to be an object of interest.

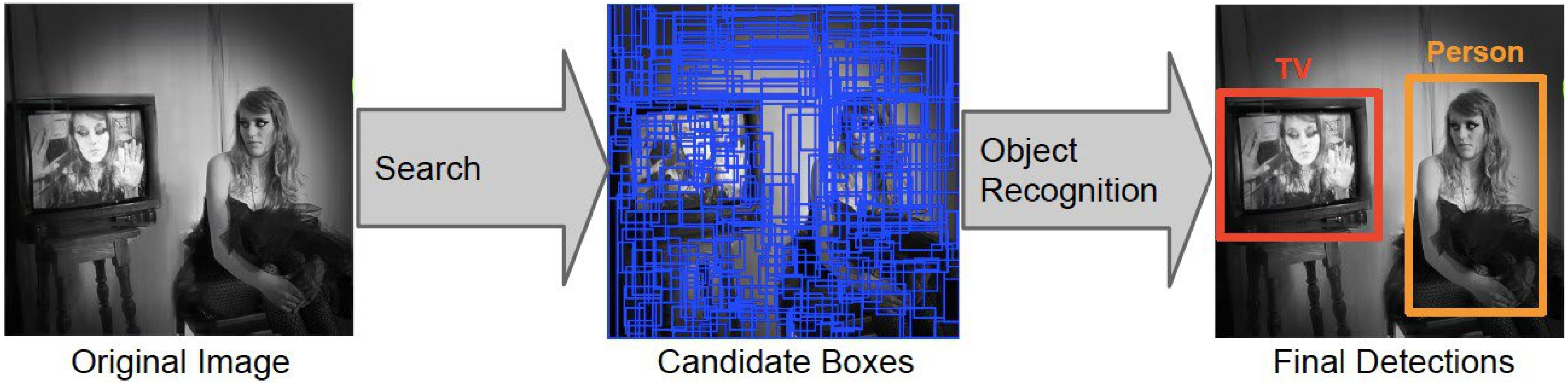
Source: S. Lazebnik

# Selective



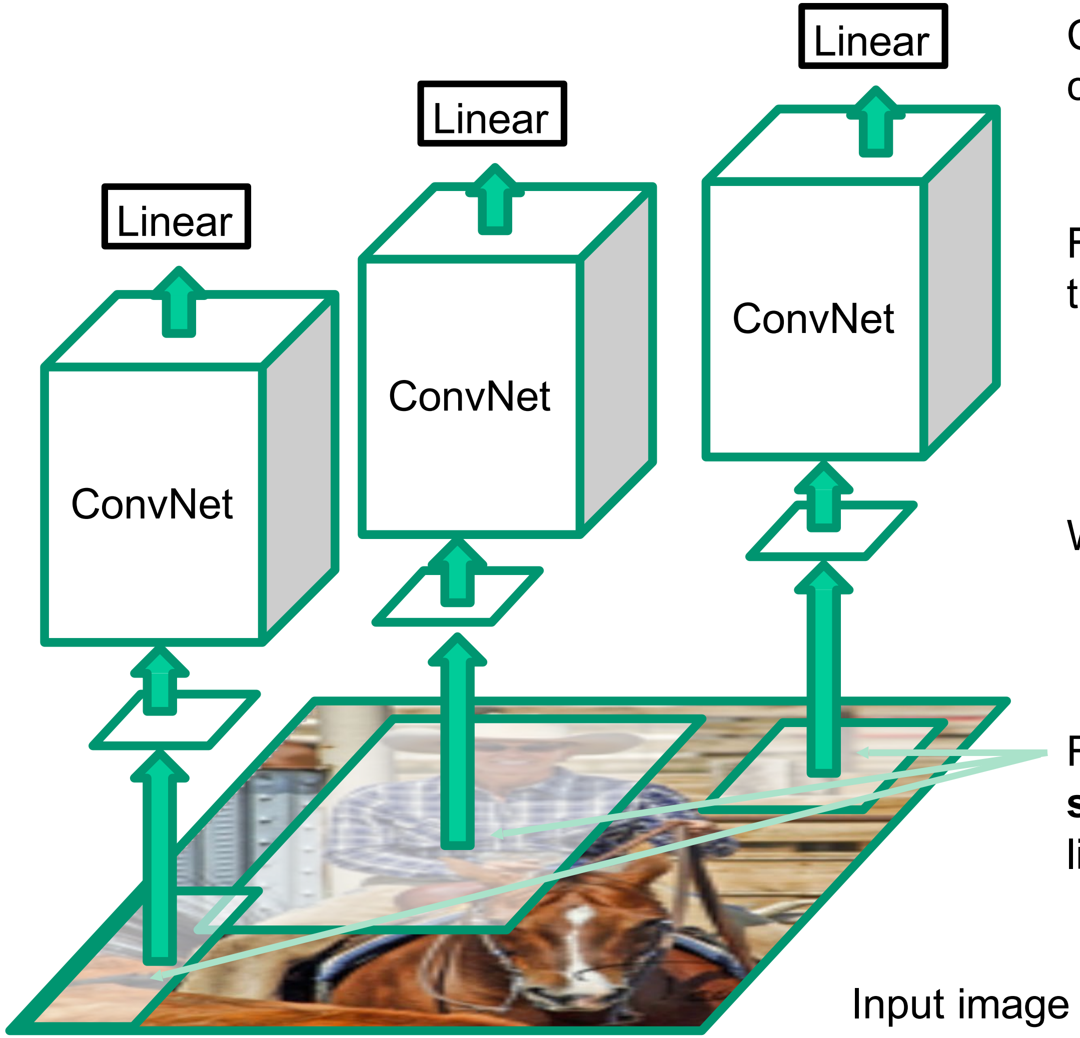
- Example: edge boxes [Zitnick & Dollar, 2014]
- Heuristic: detect edges, group them into contours
- Rank each window based on number of contours in window
- These are the only windows our detector will see

# Recall: idea #3: selective search




- Problem: evaluating a detector is very expensive
- An image with n pixels has $O(n^2)$ windows
- Only generate and evaluate a few hundred **region proposals** for regions that are "likely" to be an object of interest.

# R-CNN: Region proposals + CNN features




Classify regions with linear classifier

Forward each region through a CNN

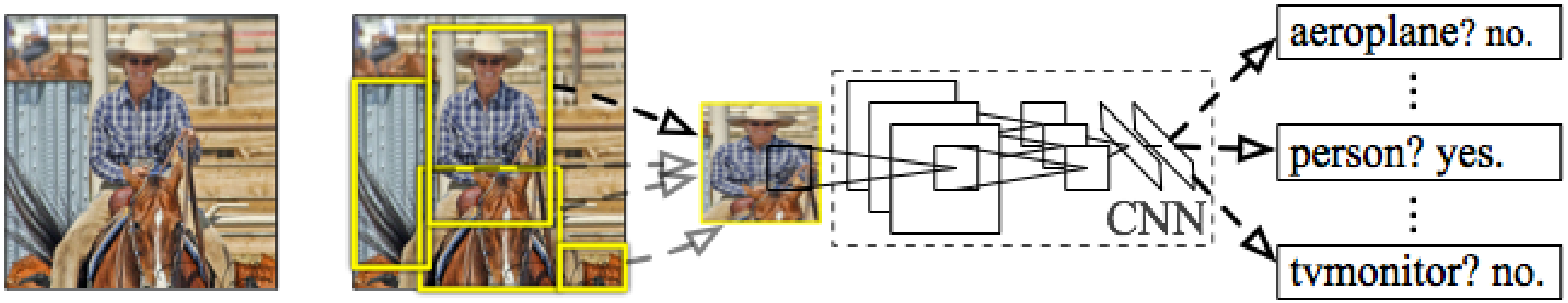
Warped image regions

Region proposals from **selective search** (~2K rectangles that are likely to contain objects)

R. Girshick, J. Donahue, T. Darrell, and J. Malik, [Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. CVPR 2014]()

Source: R. Girshick

# R-CNN at test time

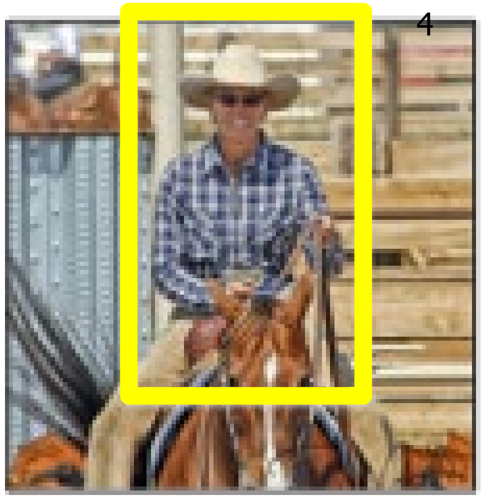


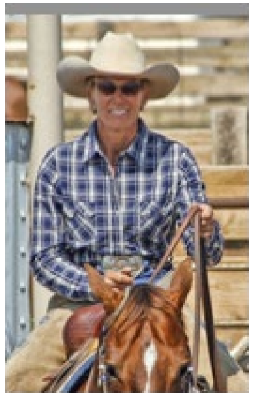

Input image

Extract region proposals (~2k / image)

→ Compute CNN features



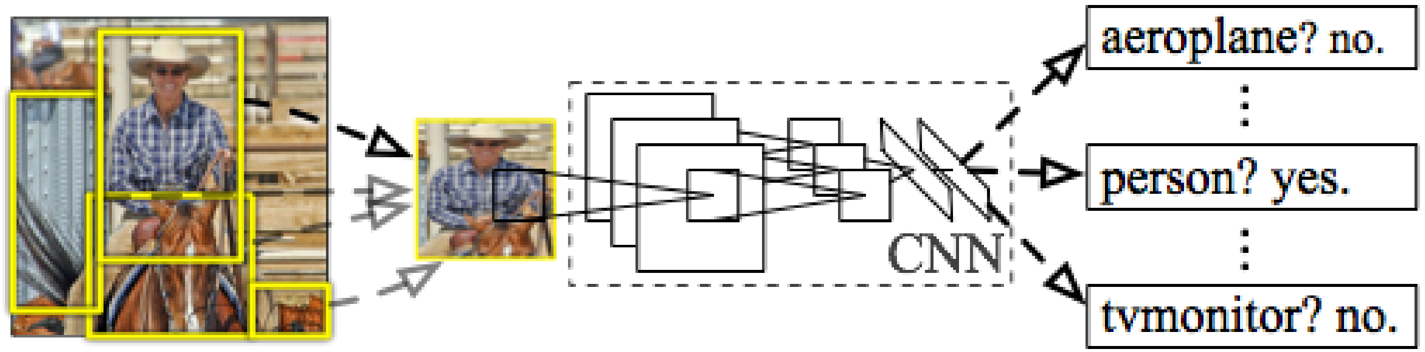




a. Crop

Source: R. Girshick

# R-CNN at test time

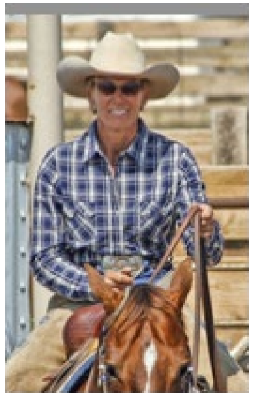



Input image

Extract region proposals (~2k / image)

Compute CNN features





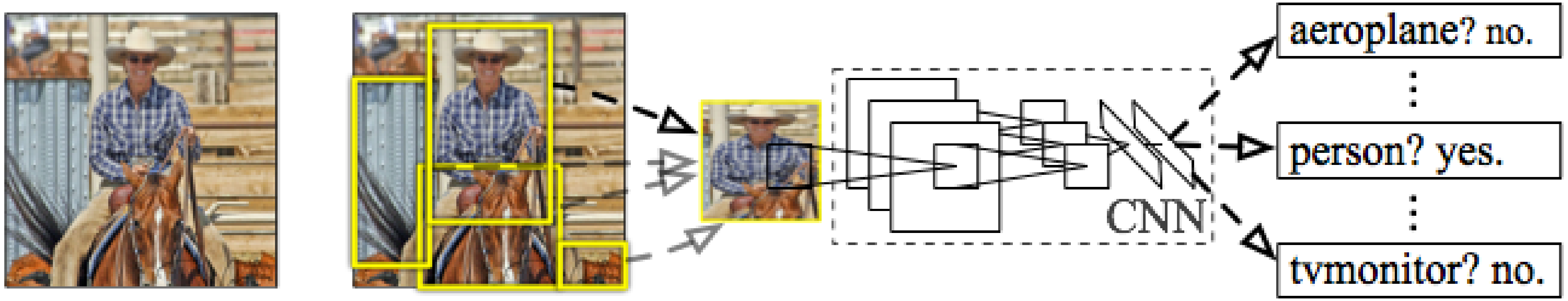
a. Crop

b. Scale

227 x 227

Source: R. Girshick

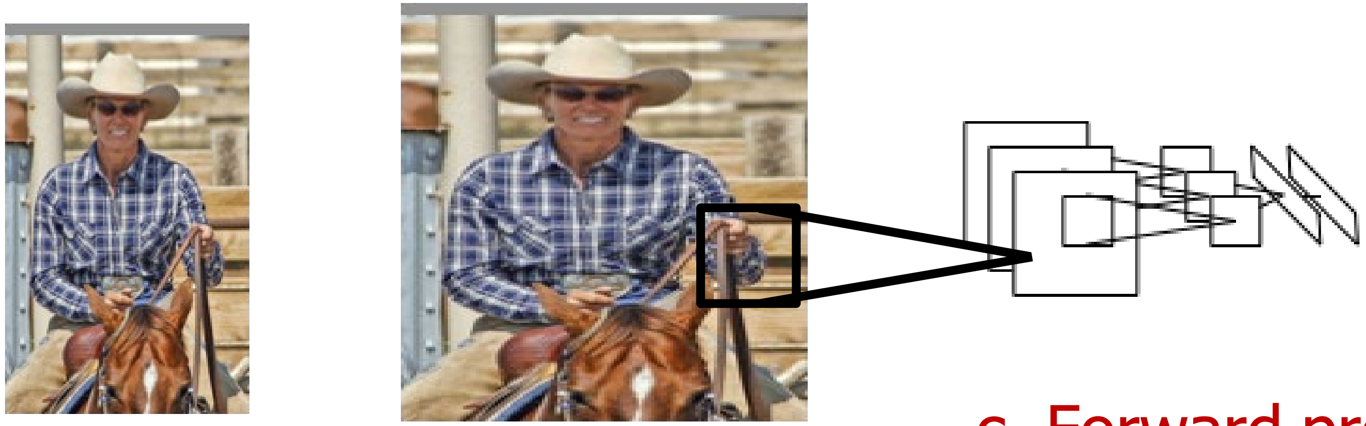
# R-CNN at test time




Input image

Extract region proposals (~2k / image)

Compute CNN features



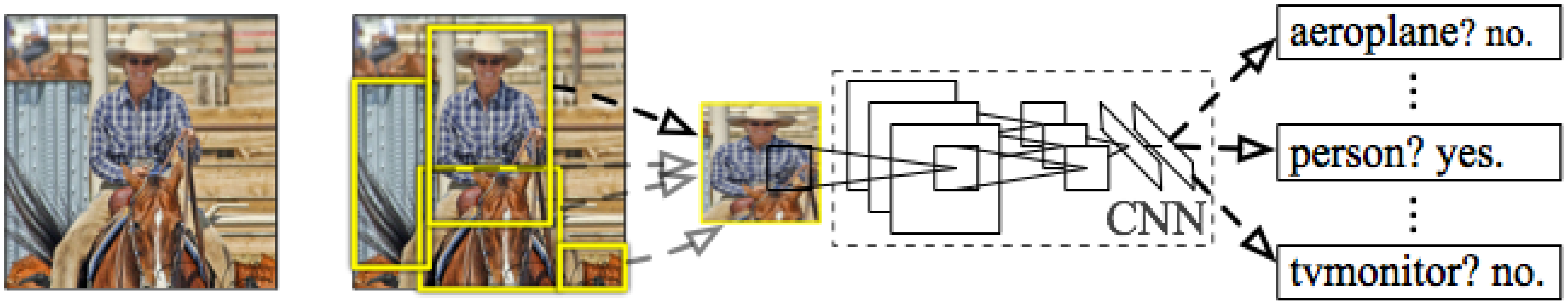
1. Crop

b. Scale

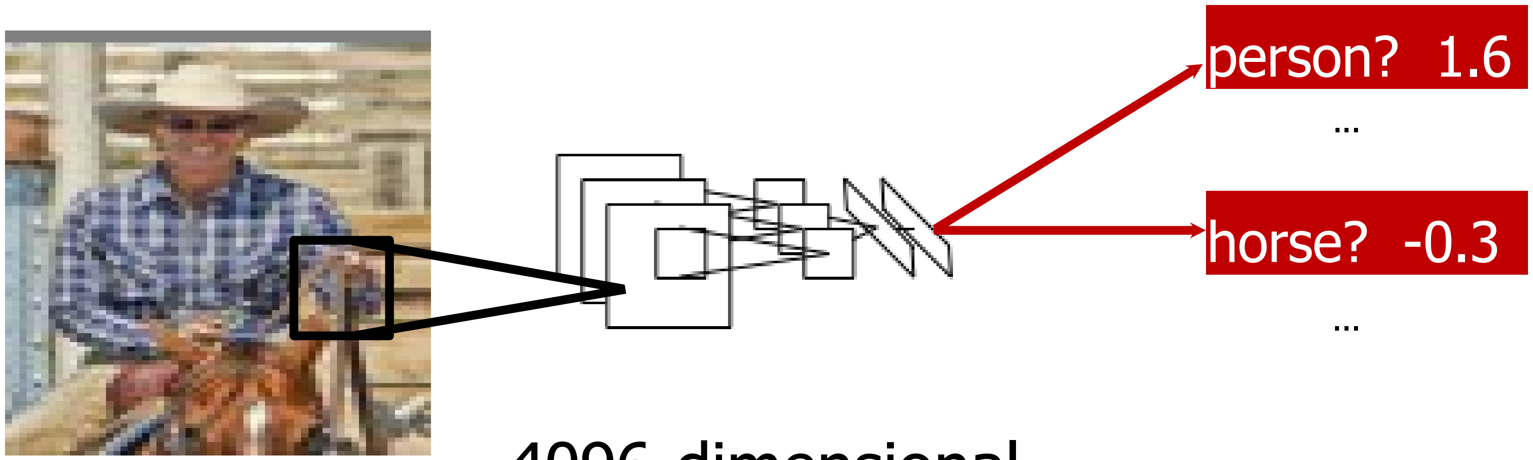
c. Forward propagate Output: "$fc_7$" features

Source: R. Girshick

# R-CNN at test time




Input image | Extract region proposals (~2k / image) | Compute CNN features → Classify regions




Warped proposal | 4096-dimensional fc$_7$ feature vector | linear classifier

Source: R. Girshick

# Proposal refinement

Linear regression

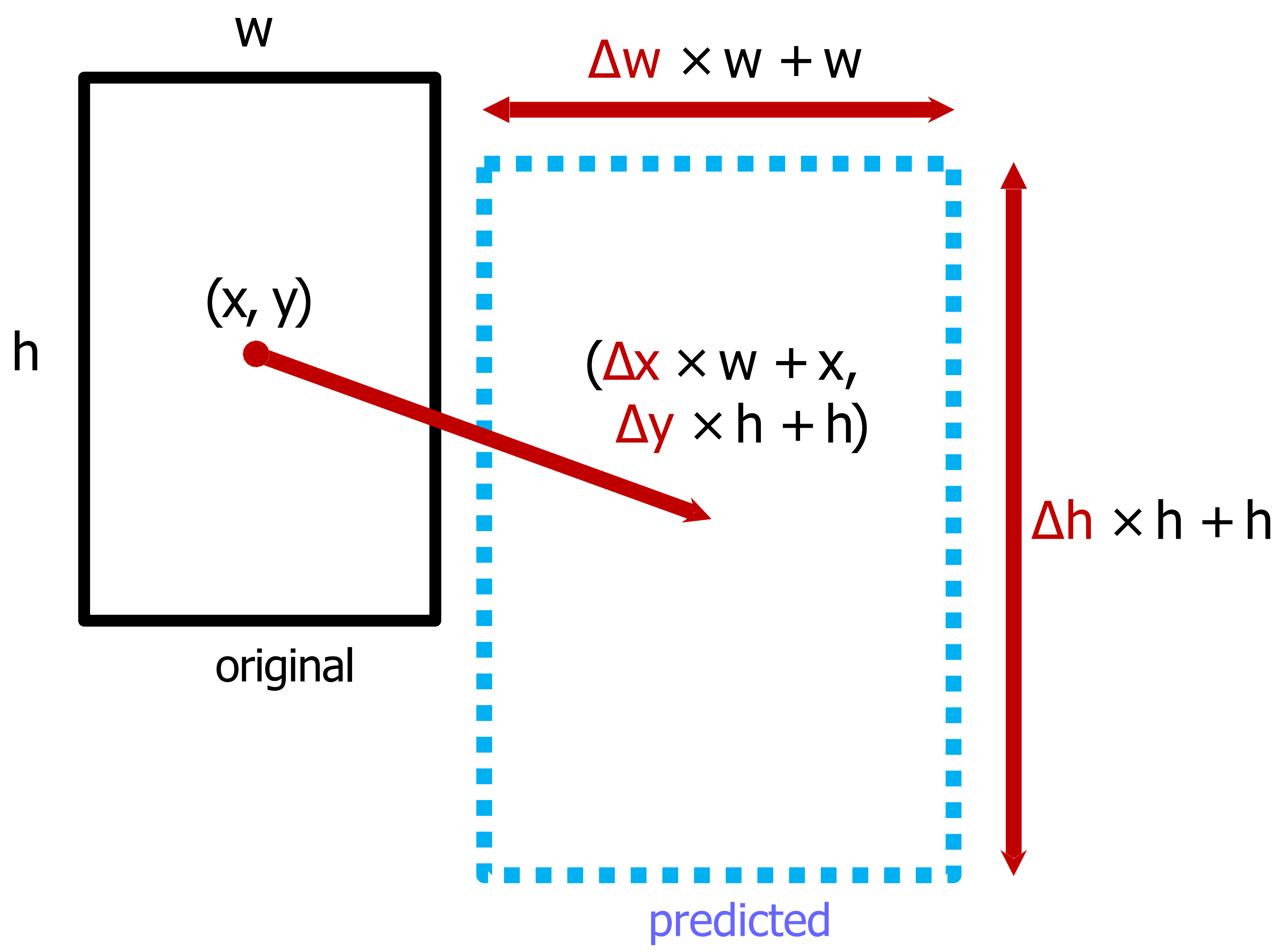
on CNN features

Original
proposal

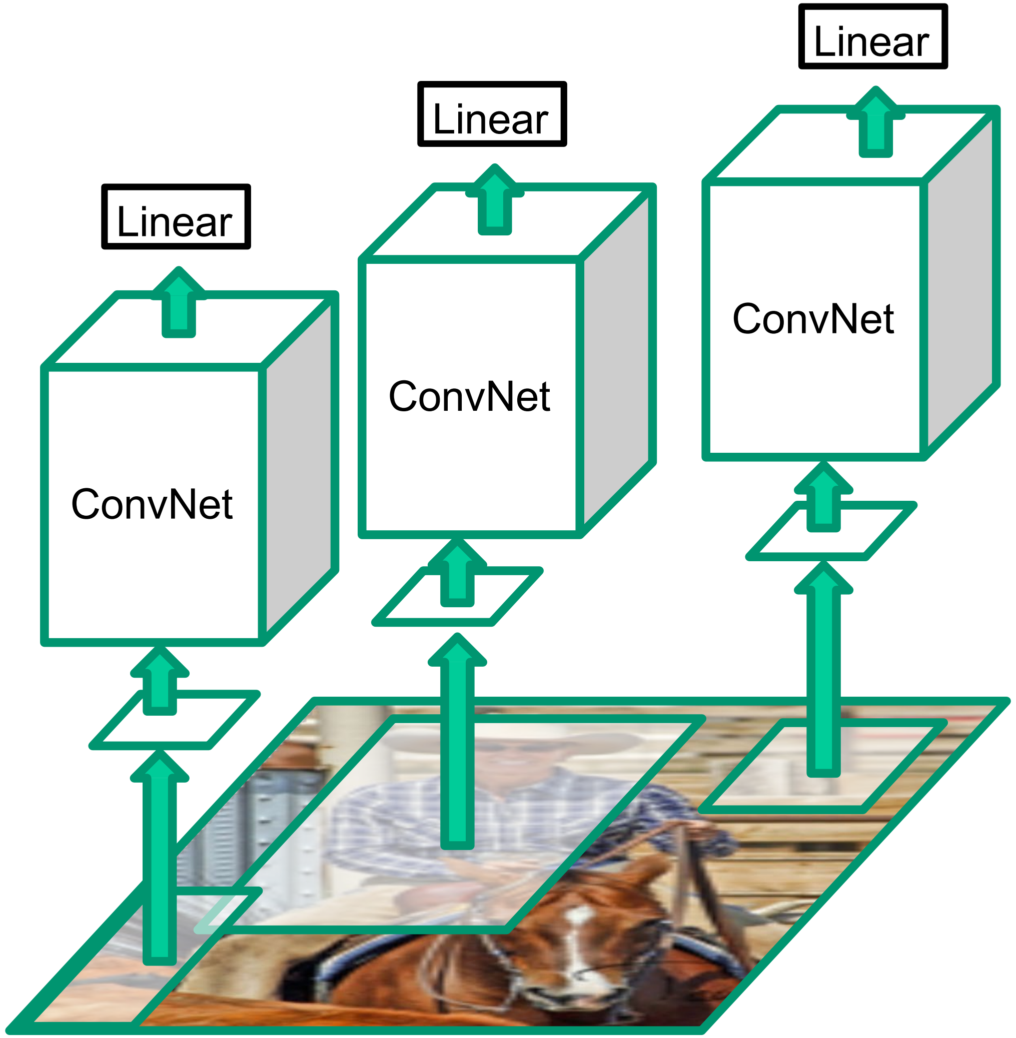
Predicted
object bounding box

Bounding-box regression

# Bounding-box regression

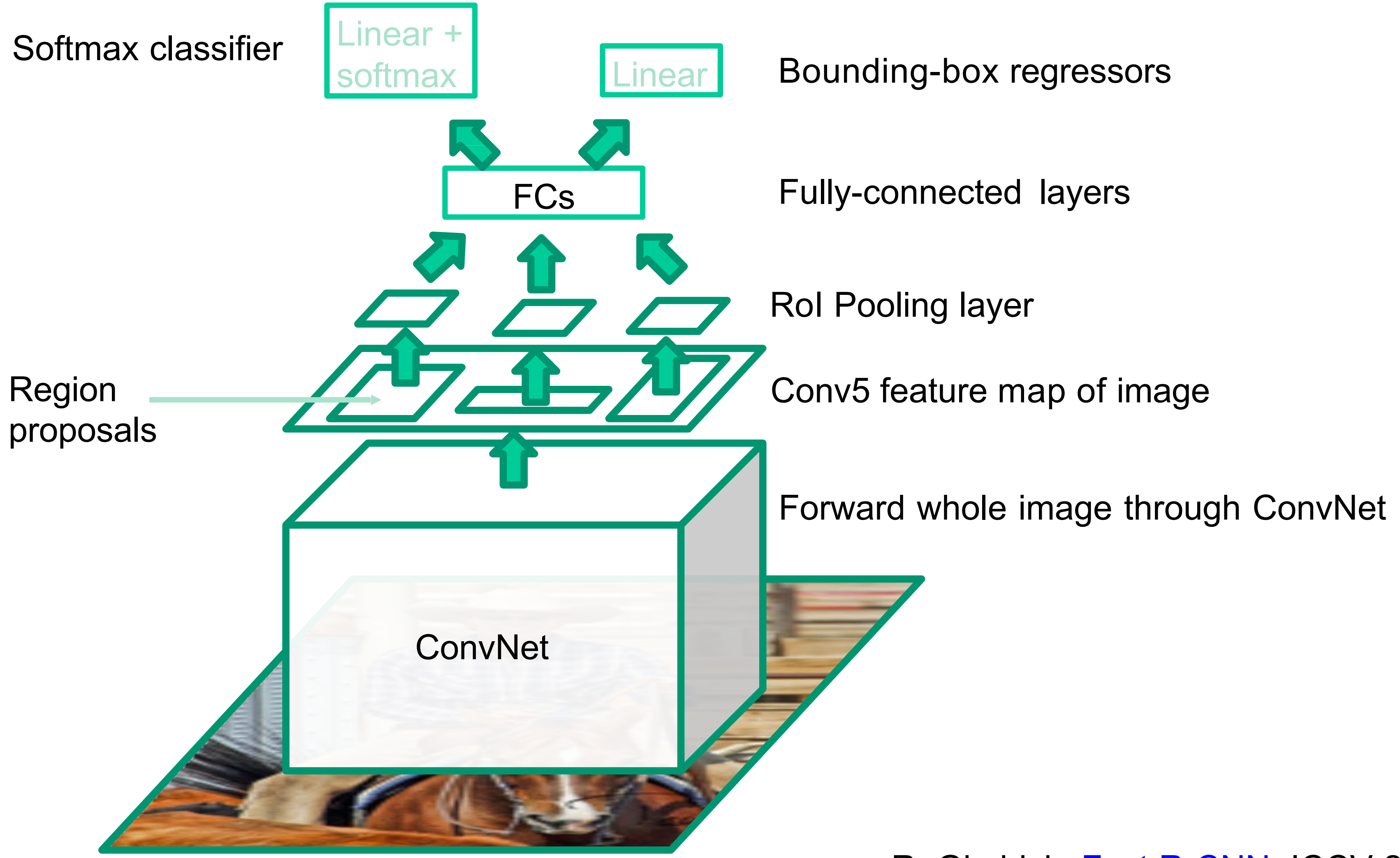


# Problems with R-CNN




1. Slow! Have to run CNN per window
2. Hand-crafted mechanism for region proposal might be suboptimal.

# "Fast" R-CNN: reuse features between proposals

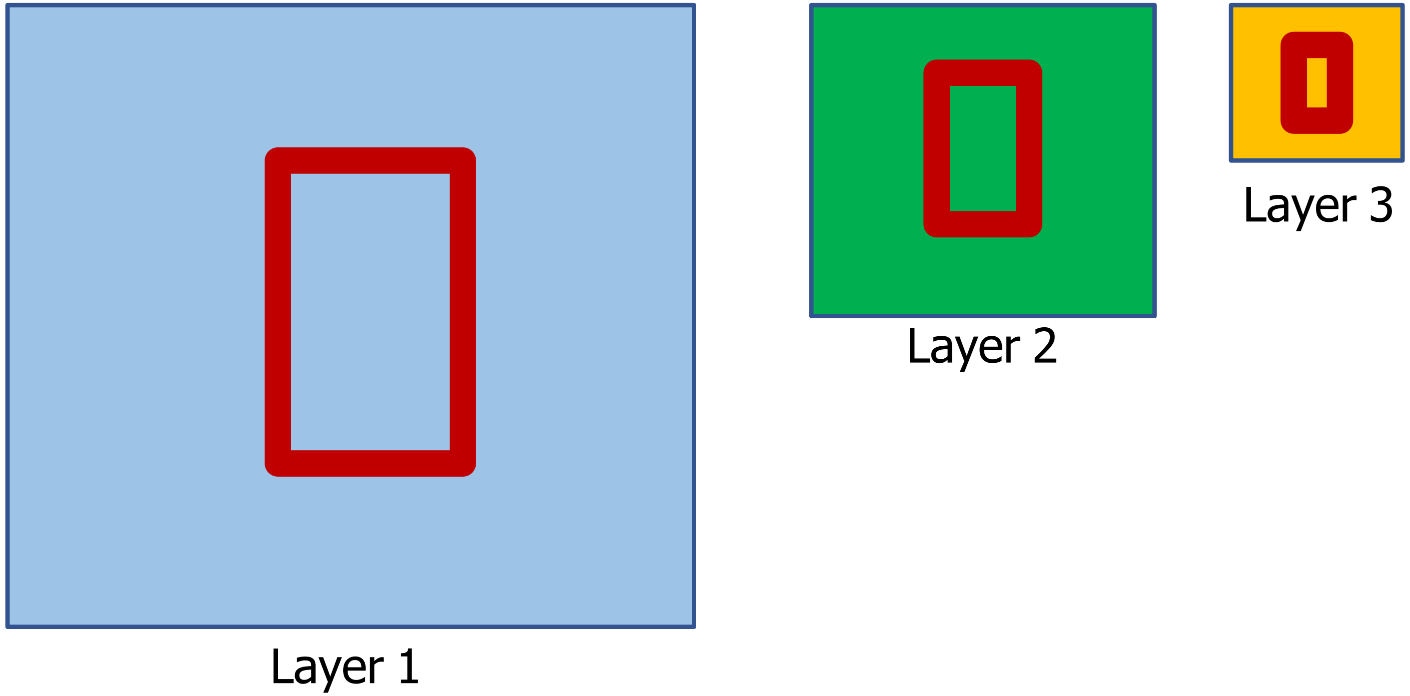



Source: R. Girshick

R. Girshick, Fast R-CNN, ICCV 2015

# ROI Pooling

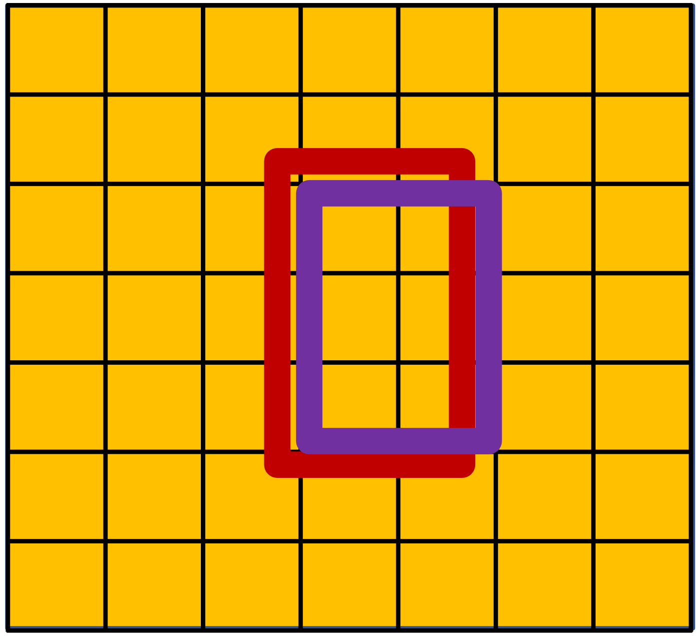
- How do we crop from a feature map?
- Step 1: Resize boxes to account for subsampling




Source: B. Hariharan

# ROI Pooling

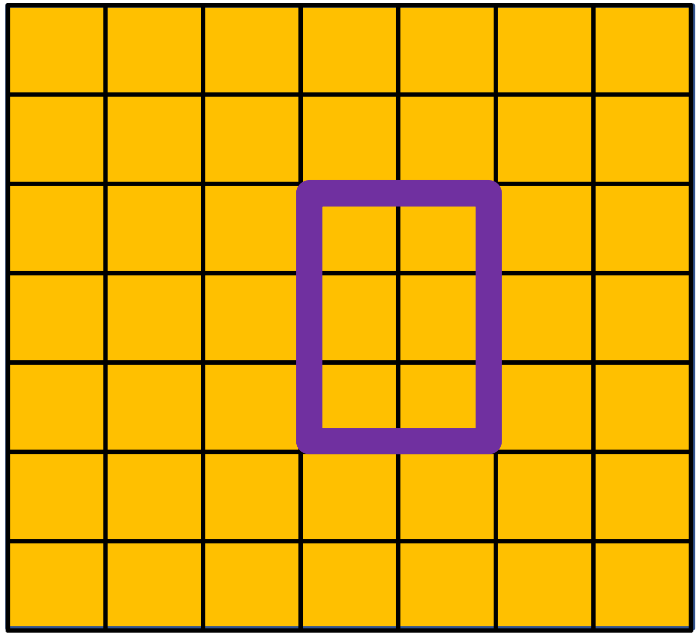
- How do we crop from a feature map?
- Step 2: Snap to feature map grid

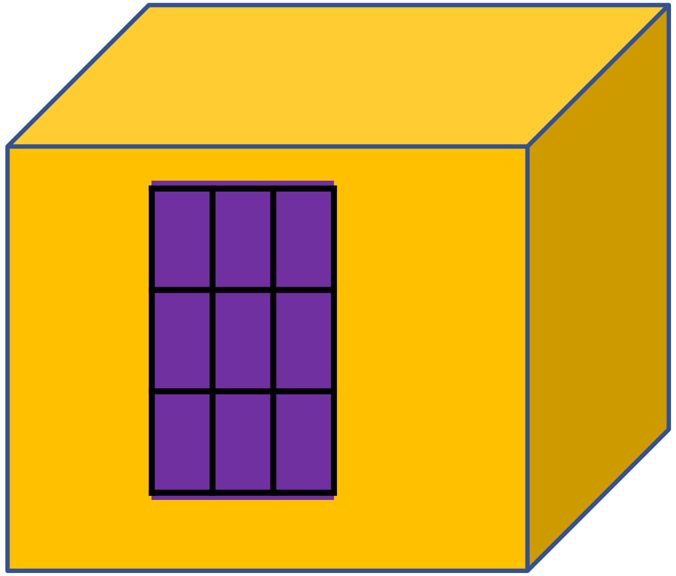


Source: B. Hariharan

# ROI Pooling

- How do we crop from a feature map?
- Step 3: Overlay a new grid of fixed size

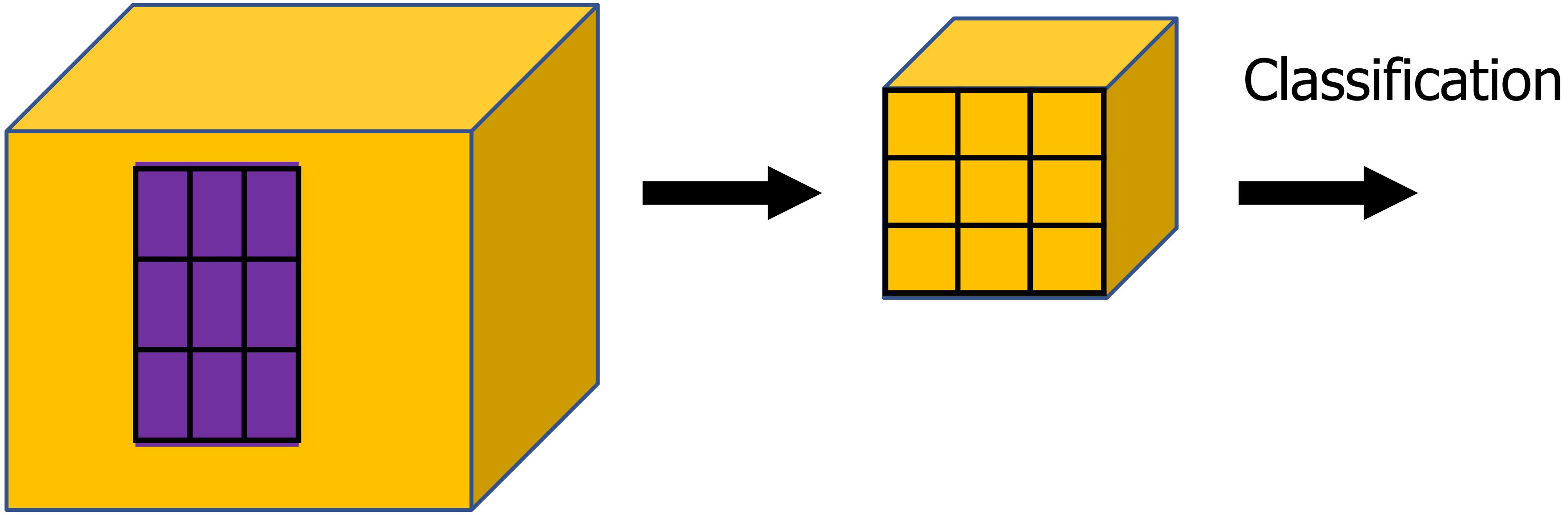




Source: B. Hariharan

# ROI Pooling

- How do we crop from a feature map?
- Step 4: Take max in each cell
- Can improve with bilinear sampling

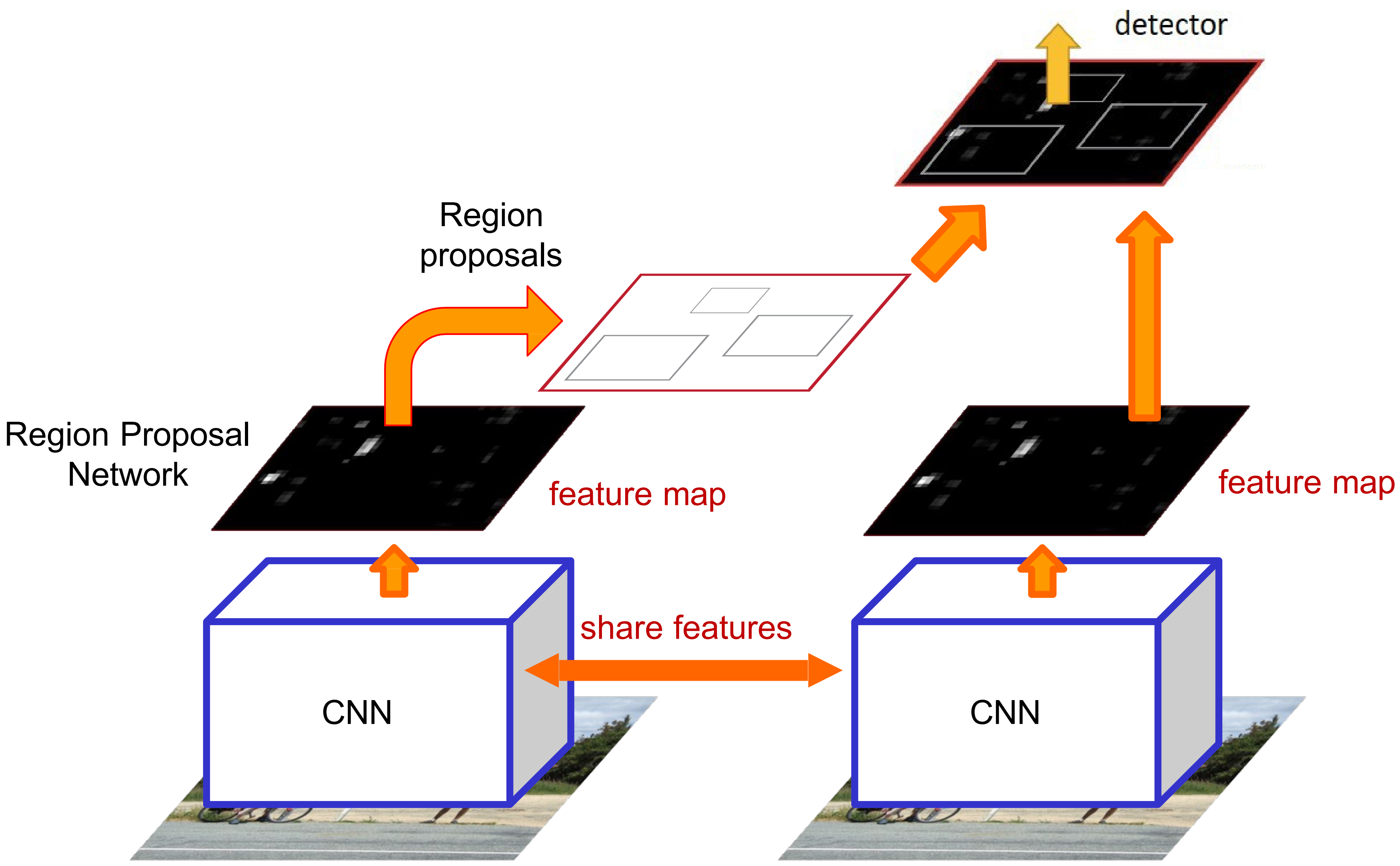



See more here: https://deepsense.ai/region-of-interest-pooling-explained/

Source: B. Hariharan

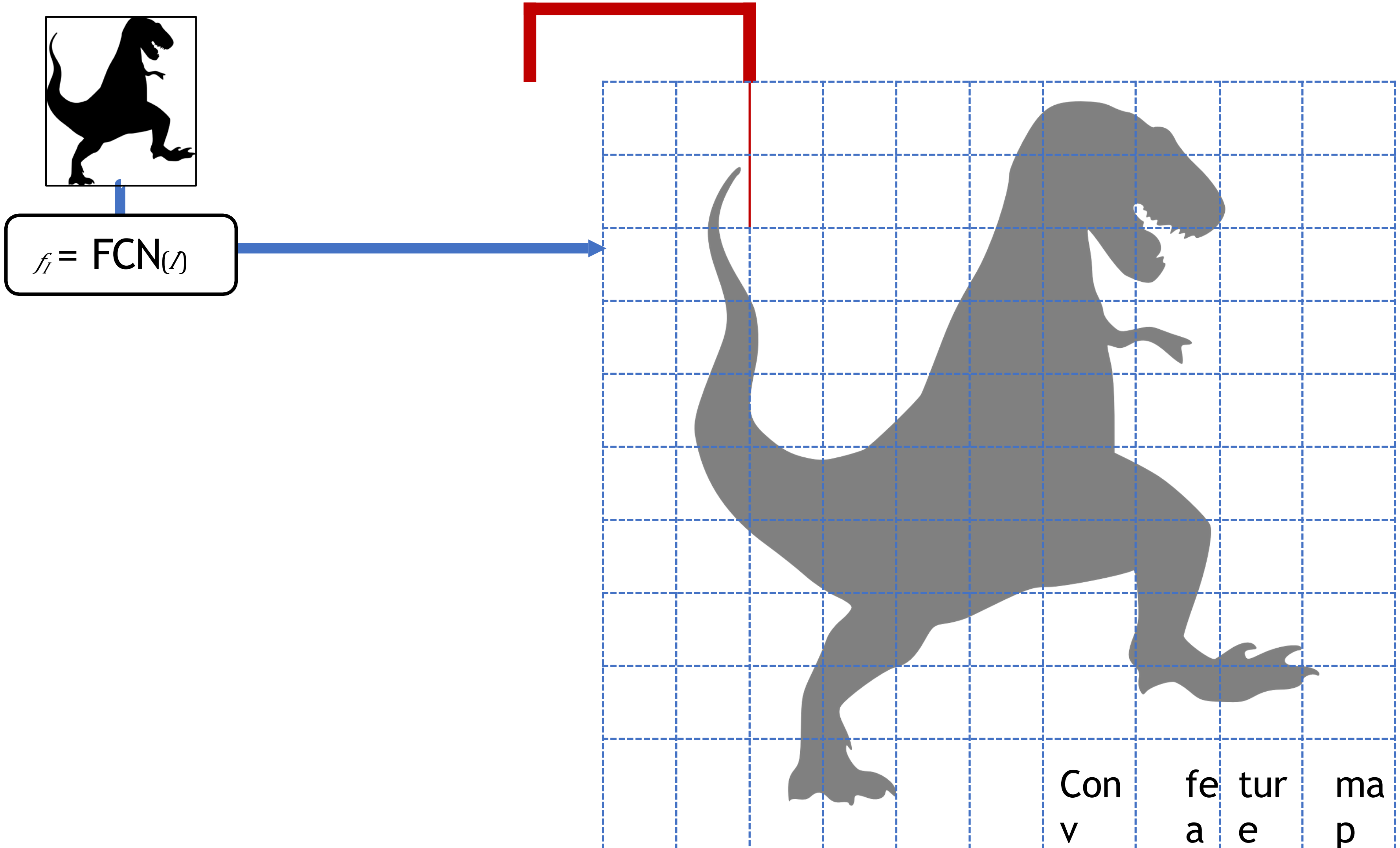
# 2024-03-13

# “Faster” R-CNN: learn region proposals



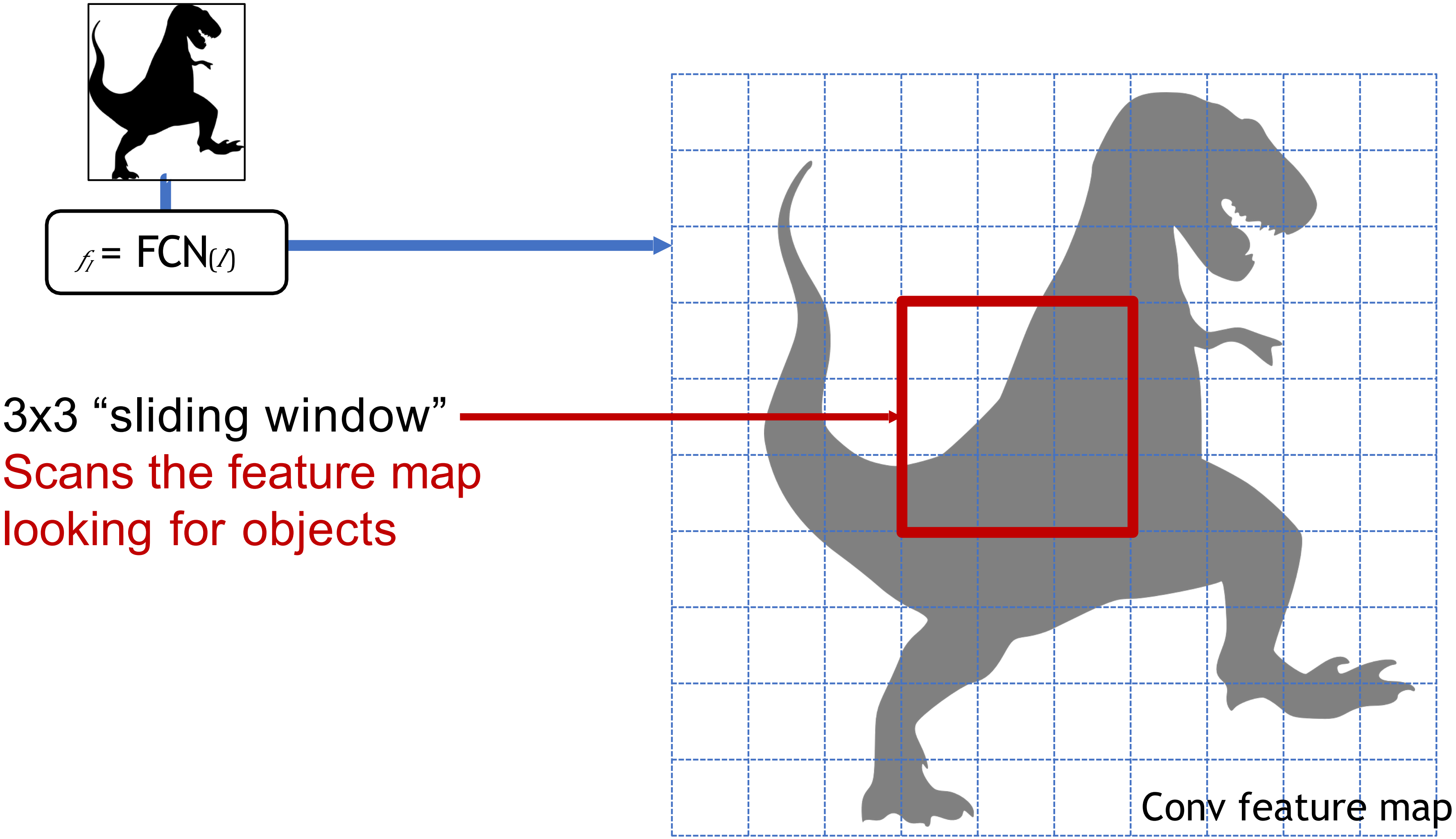

S. Ren, K. He, R. Girshick, and J. Sun, [Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks](), NIPS 2015

# RPN: Region Proposal Network




Source: R. Girshick

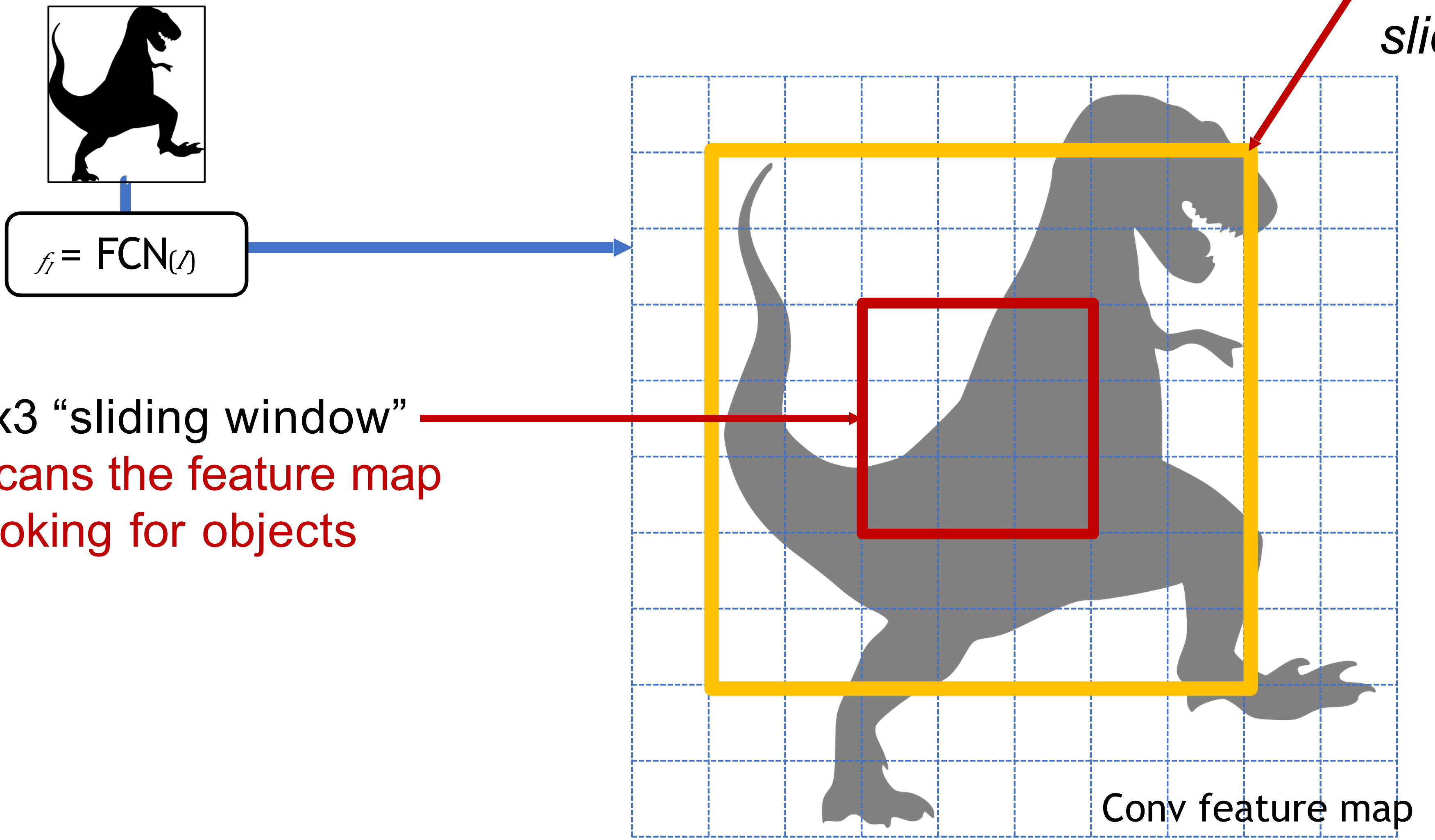
# RPN: Region Proposal Network




Source: R. Girshick

# RPN: Anchor Box

Anchor box: predictions are w.r.t. this box, *not the 3x3 sliding window*

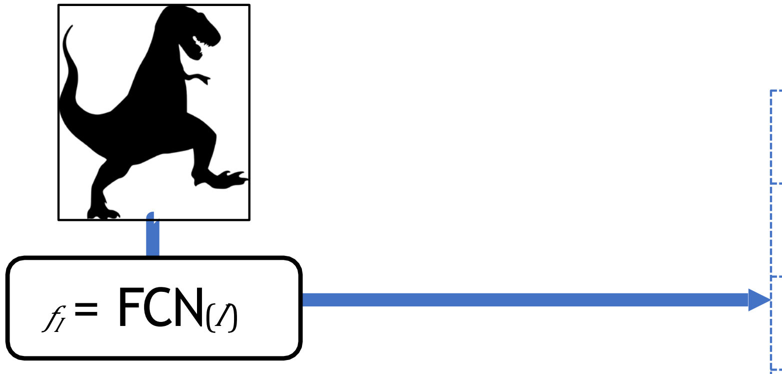



3x3 "sliding window"
Scans the feature map looking for objects

Source: R. Girshick

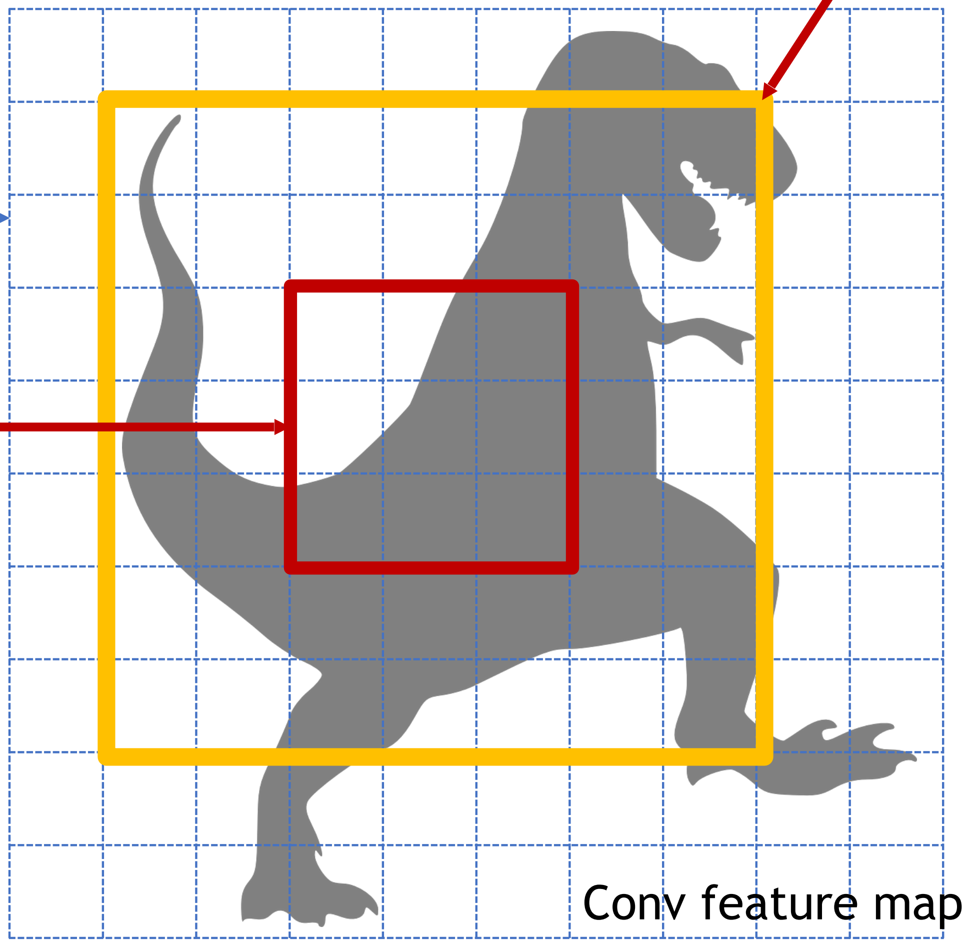
# RPN: Anchor Box

Anchor box: predictions are w.r.t. this box, *not the 3x3 sliding window*




3x3 “sliding window”

- ➢ Objectness classifier [0, 1]
- ➢ Box regressor predicting (dx, dy, dh, dw)




Source: R. Girshick

# RPN: Prediction (on object)

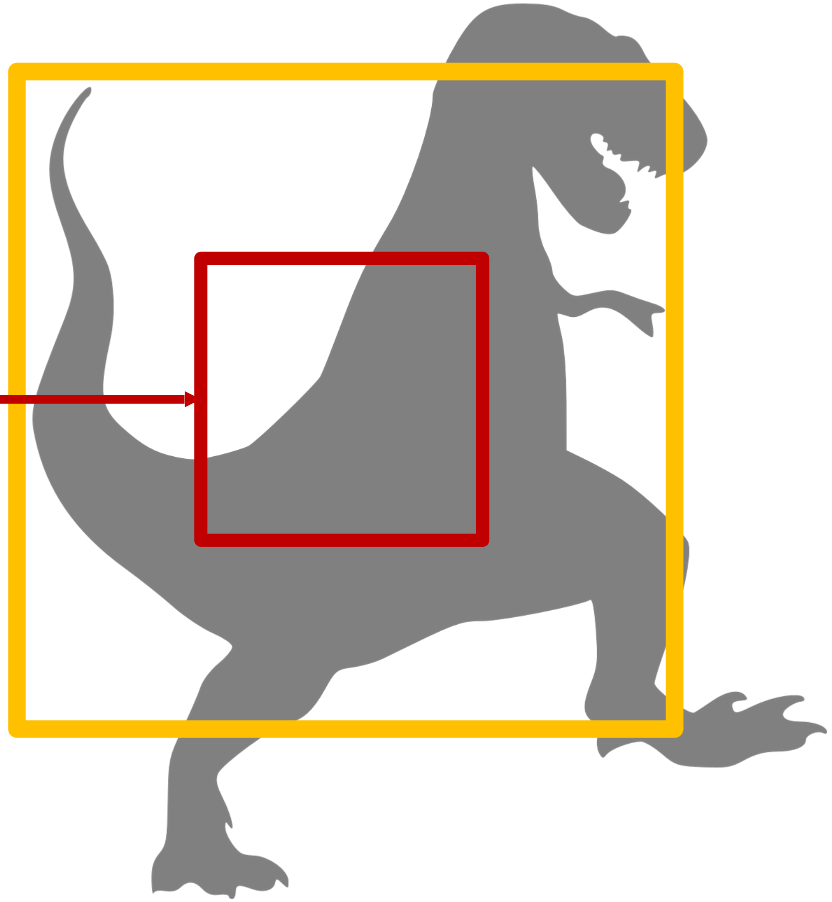
Objectness score

P(object) = 0.94

3x3 “sliding window”

- ➢ Objectness classifier [0, 1]

- ➢ Box regressor
  predicting (dx, dy, dh, dw)

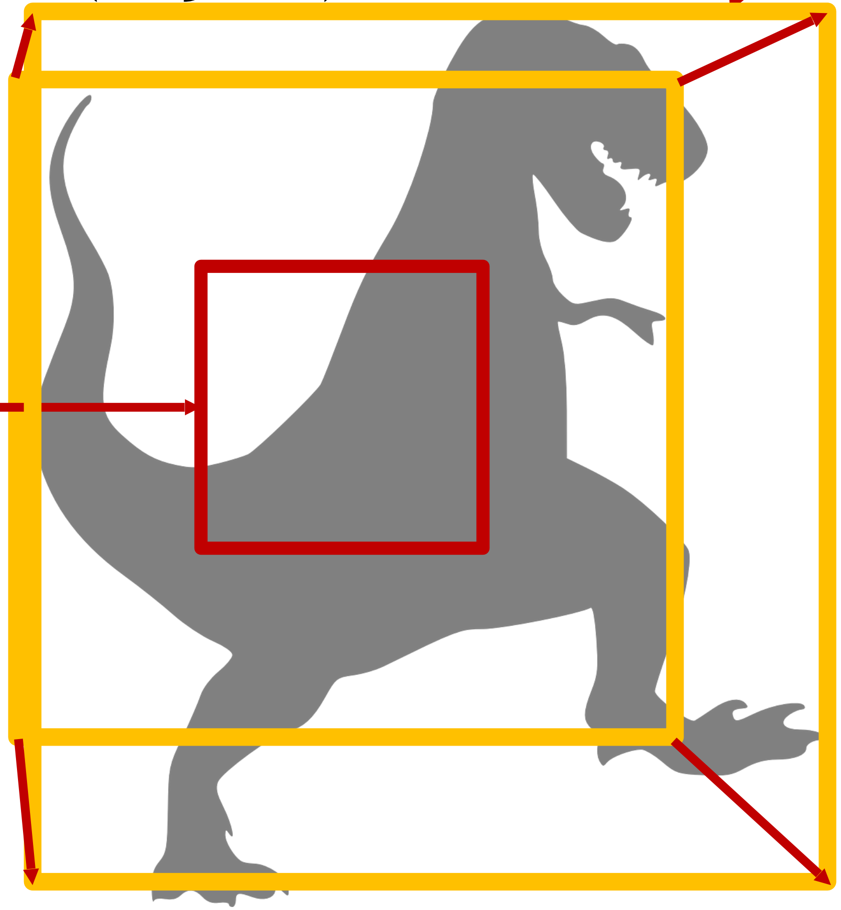


Source: R. Girshick

# RPN: Prediction (on object)

Anchor box: transformed by box regressor

P(object) = 0.94



3x3 "sliding window"

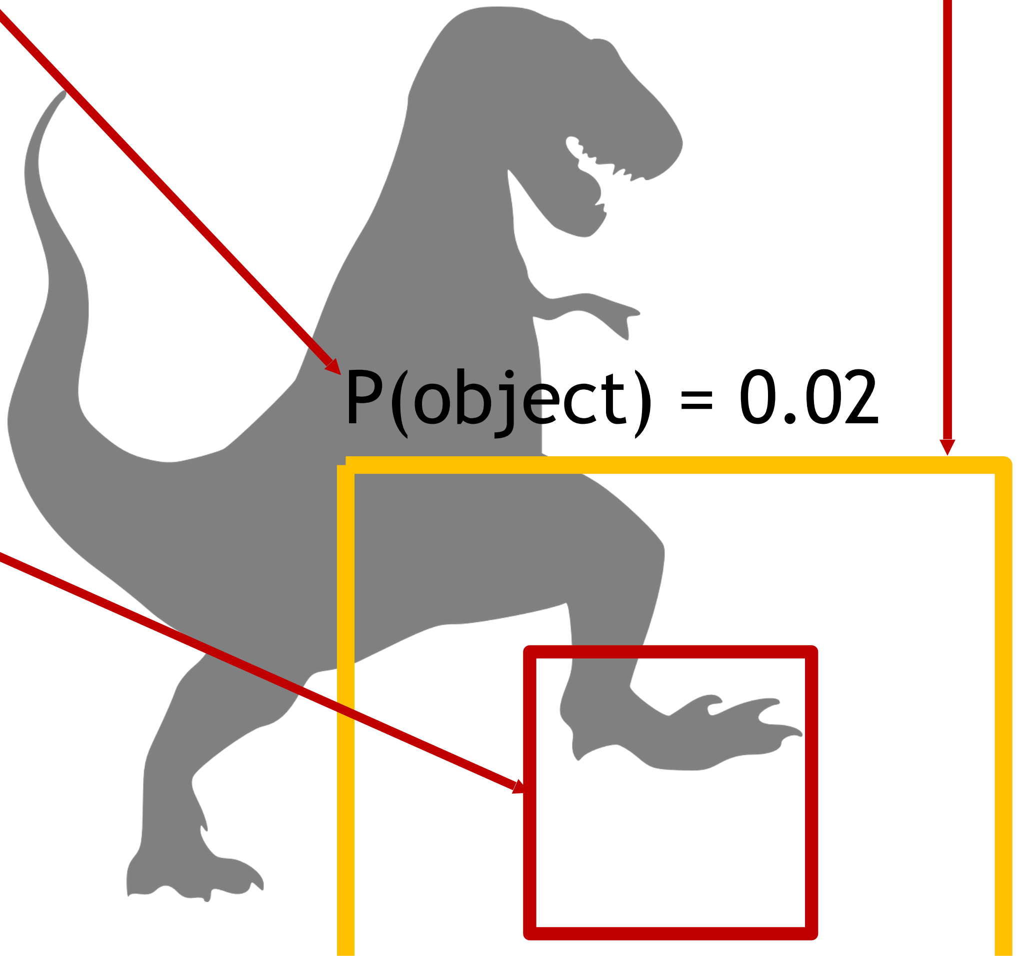
- ➢ Objectness classifier [0, 1]
- ➢ Box regressor predicting (dx, dy, dh, dw)

Source: R. Girshick

# RPN: Prediction (off object)

Anchor box: transformed by box regressor

Objectness score

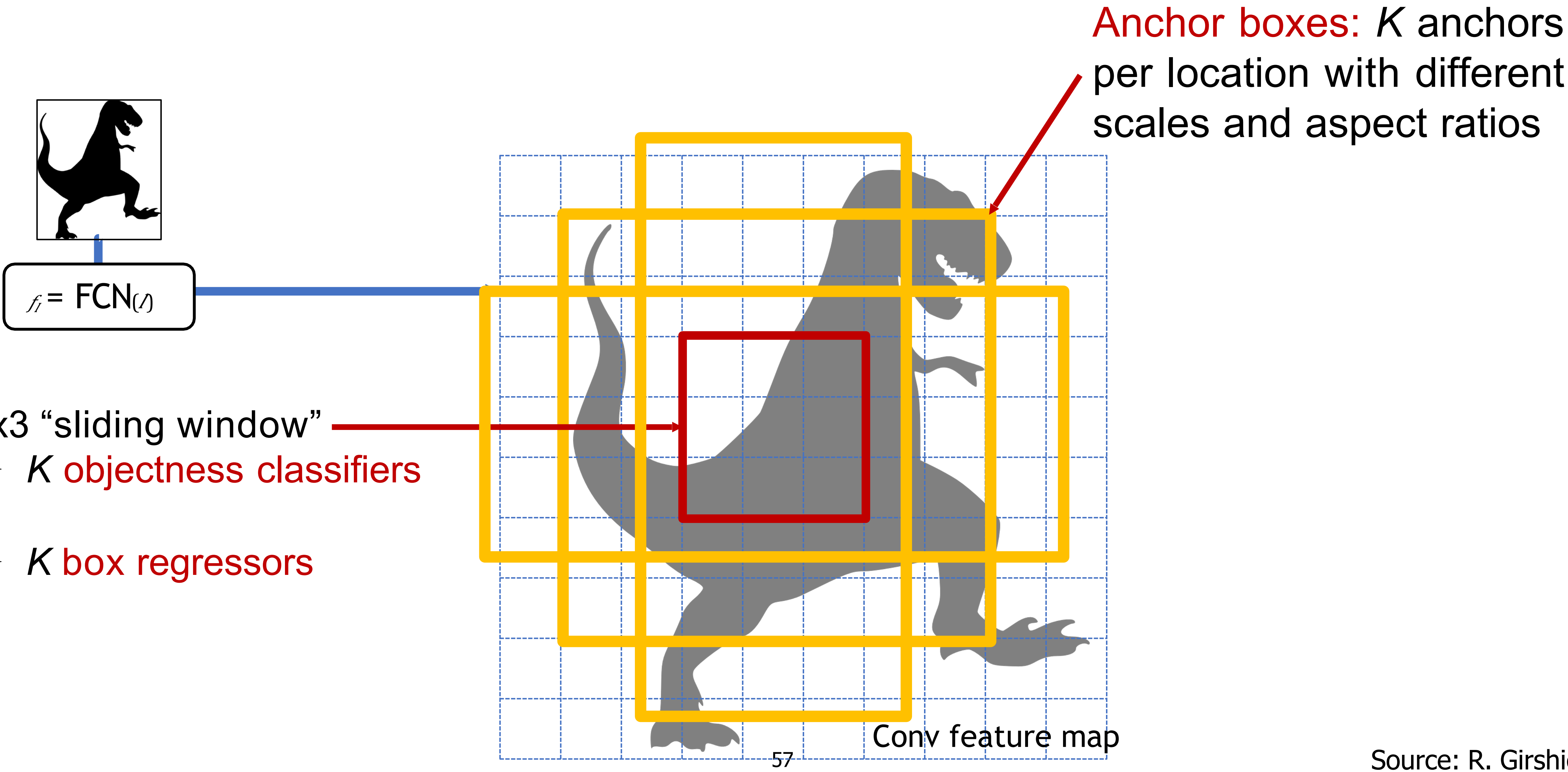



3x3 "sliding window"

- ➢ Objectness classifier
- ➢ Box regressor predicting (dx, dy, dh, dw)

Source: R. Girshick

# RPN: Multiple Anchors

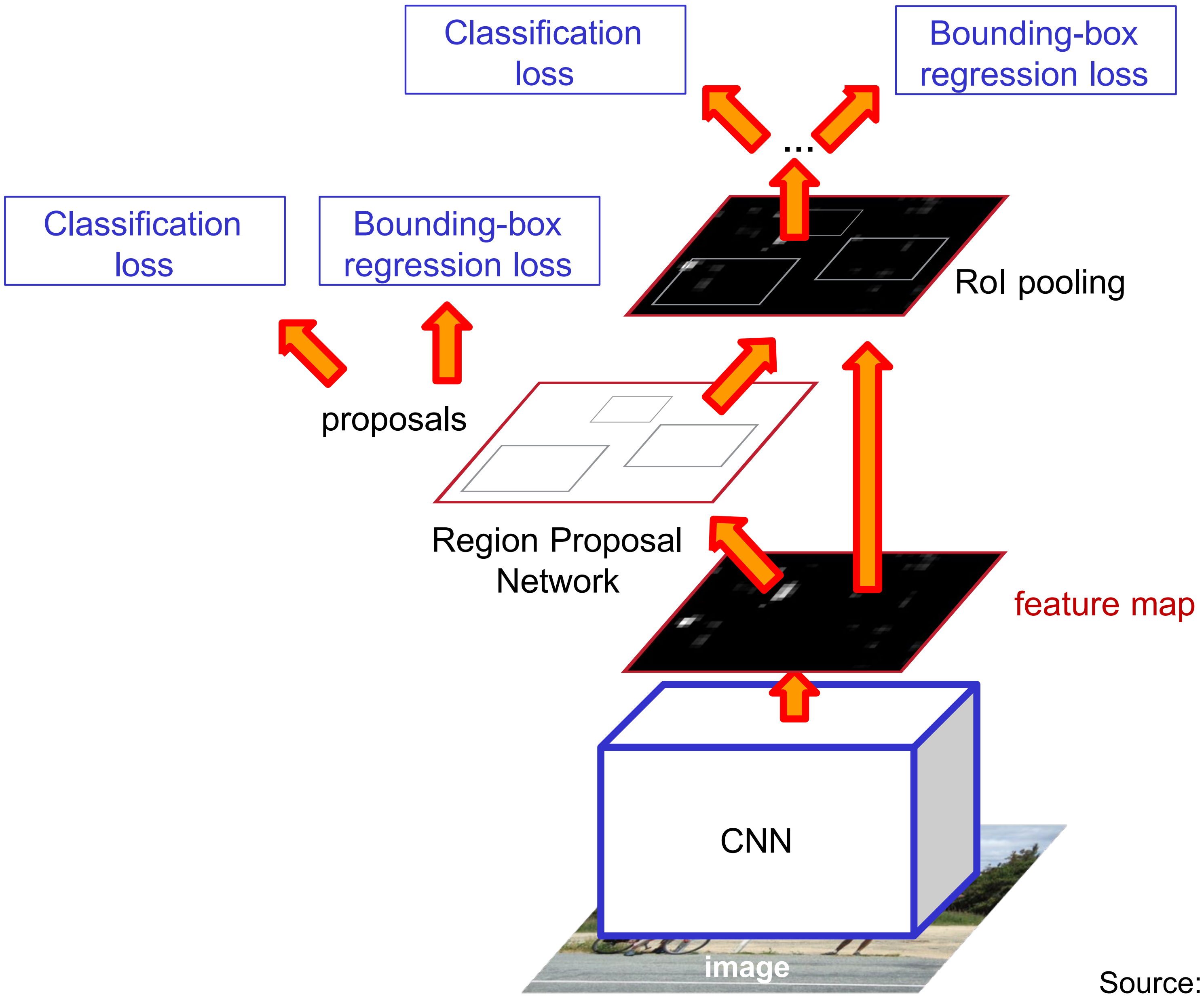



- $K$ objectness classifiers
- $K$ box regressors



Source: R. Girshick

# One network, four losses




Source: R. Girshick, K. He, S. Lazebnik

# Faster R-CNN results

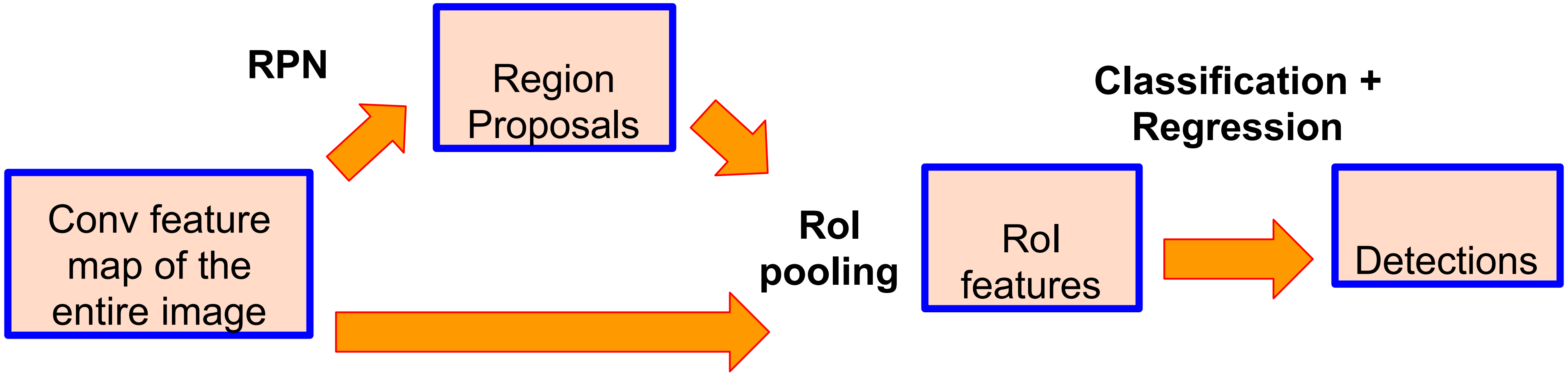
| system | time | 07 data | 07+12 data |
|---|---|---|---|
| R-CNN | ~50s | 66.0 | - |
| Fast R-CNN | ~2s | 66.9 | 70.0 |
| Faster R-CNN | 198ms | **69.9** | **73.2** |

detection mAP on PASCAL VOC 2007, with VGG-16 pre-trained on ImageNet

Source: S. Lazebnik

# Streamlined detection architectures

- The Faster R-CNN pipeline separates proposal generation and region classification:



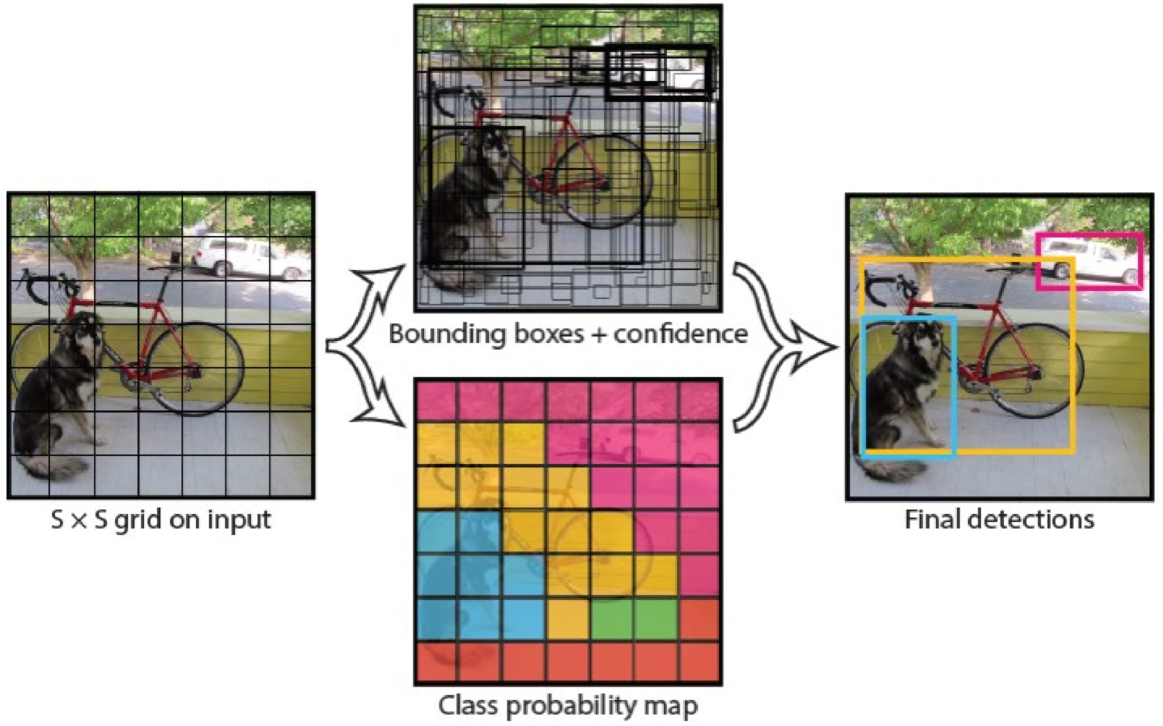

- Is it possible do detection in one shot?




Source: S. Lazebnik

# Single-stage object detector

- Divide the image into a coarse grid using a fully convolutional net
- Directly predict class label, confidence, and a few candidate boxes for each grid cell.






J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, [You Only Look Once: Unified, Real-Time Object Detection](), CVPR 2016

Source: S. Lazebnik

# YOLO detector

1. Take convolutional feature maps at 7x7 resolution
2. Predict, at each location, a score for each class and 2 bounding boxes (w/ confidence)

- E.g. for 20 classes, output is 7x7x30 (30 = 20 + 2*(4+1))
- 7x speedup over Faster R-CNN (45-155 FPS vs. 7-18 FPS) but less accurate (e.g. 65% vs. 72 mAP%)
- Extension: use anchor boxes in last layer to try a few possible aspect ratios


Bounding boxes + confidence


Class probability map

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, You Only Look Once: Unified, Real-Time Object Detection, CVPR 2016



Source: S. Lazebnik

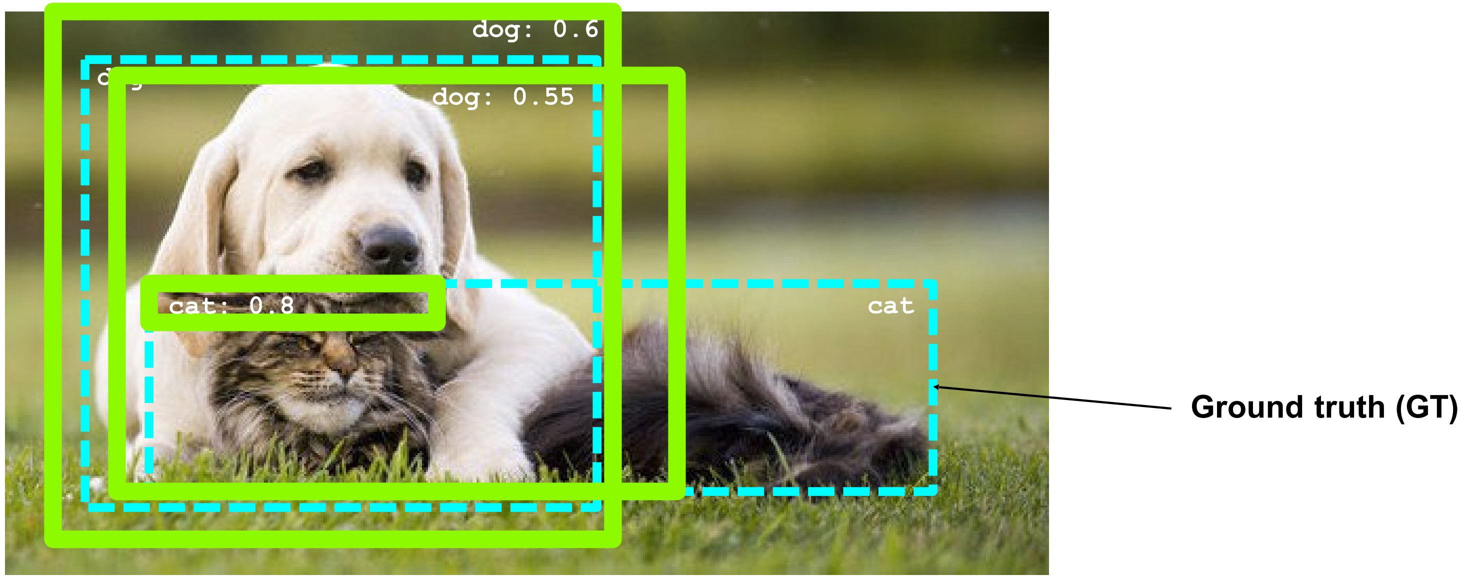
# Evaluating an object detector

# Evaluating an object detector

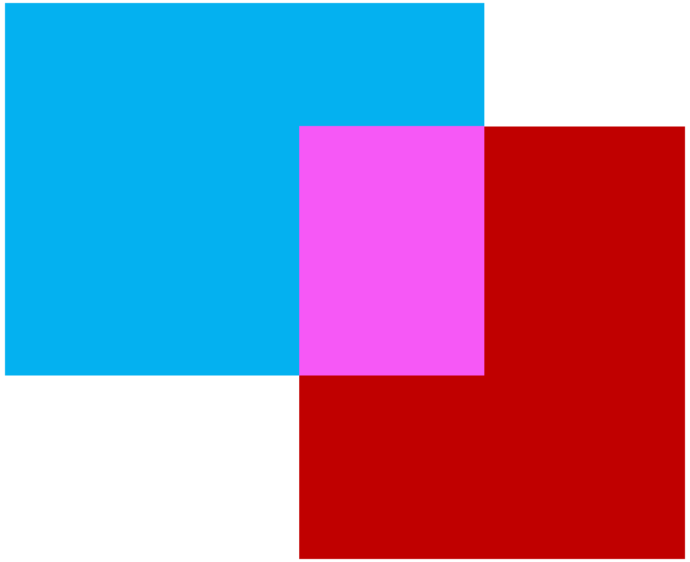
- At test time, predict bounding boxes, class labels, and confidence scores
- For each detection, determine whether it is a true or false positive

  **Intersection over union** (IoU):

  Area(GT ∩ Det) / Area(GT ∪ Det) > 0.5



# Evaluating an object detector



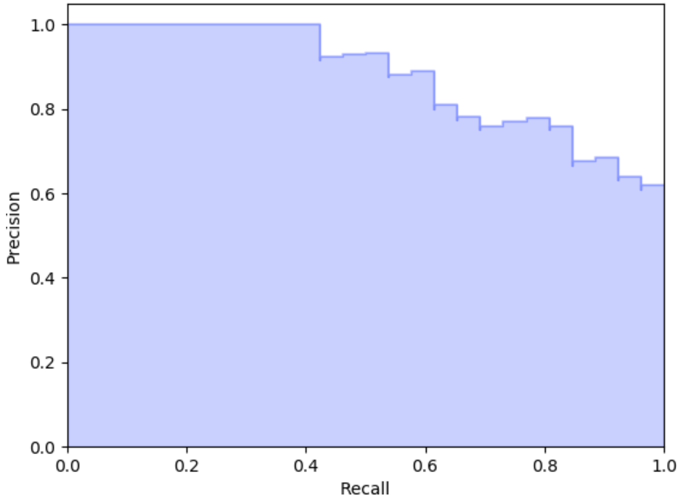
$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Intersection over union (also known as Jaccard similarity)

# Evaluating an object detector

- For each class, plot Precision-Recall curve and compute Average Precision (area under the curve)

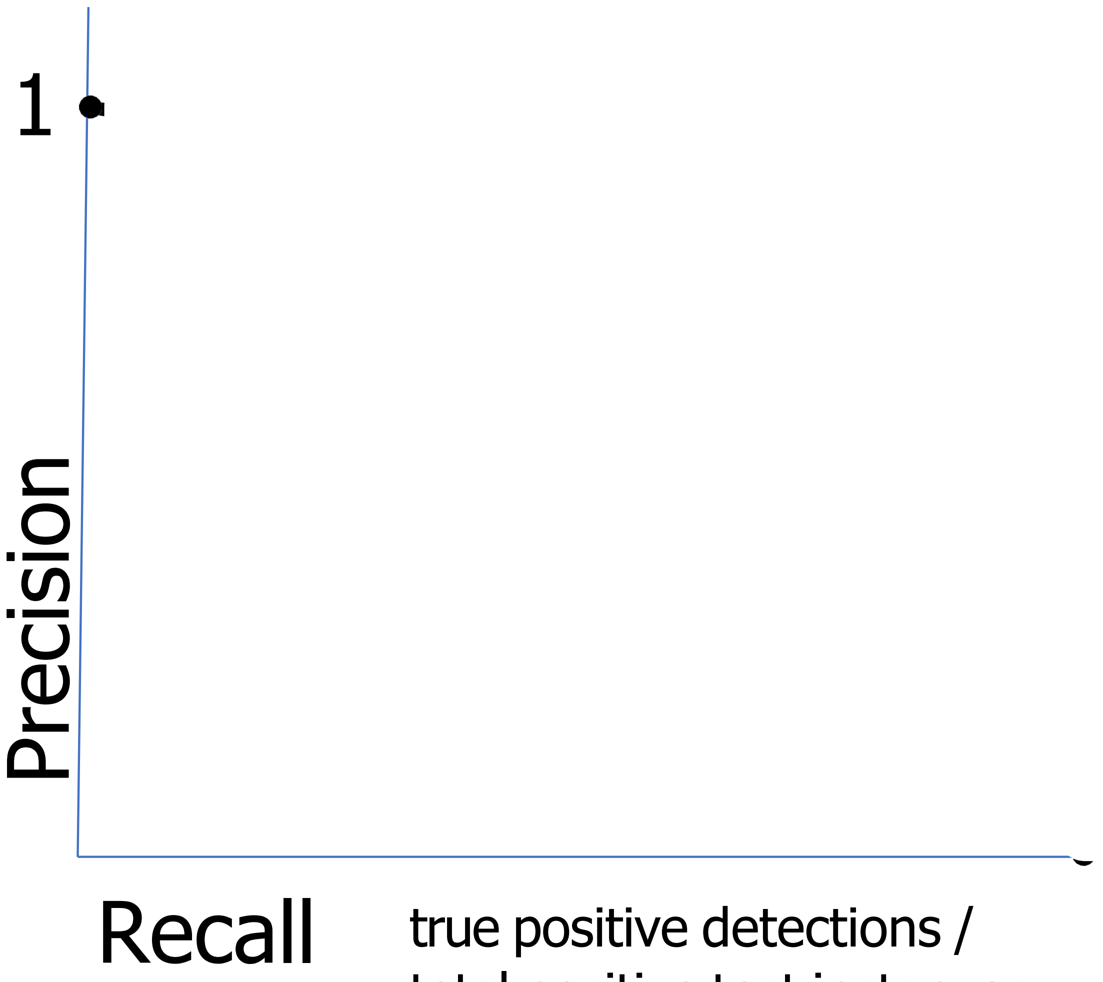
- Take mean of AP over classes to get mAP




**Precision:**
true positive detections /
total detections
**Recall:**
true positive detections /
total positive test instances

# Average precision

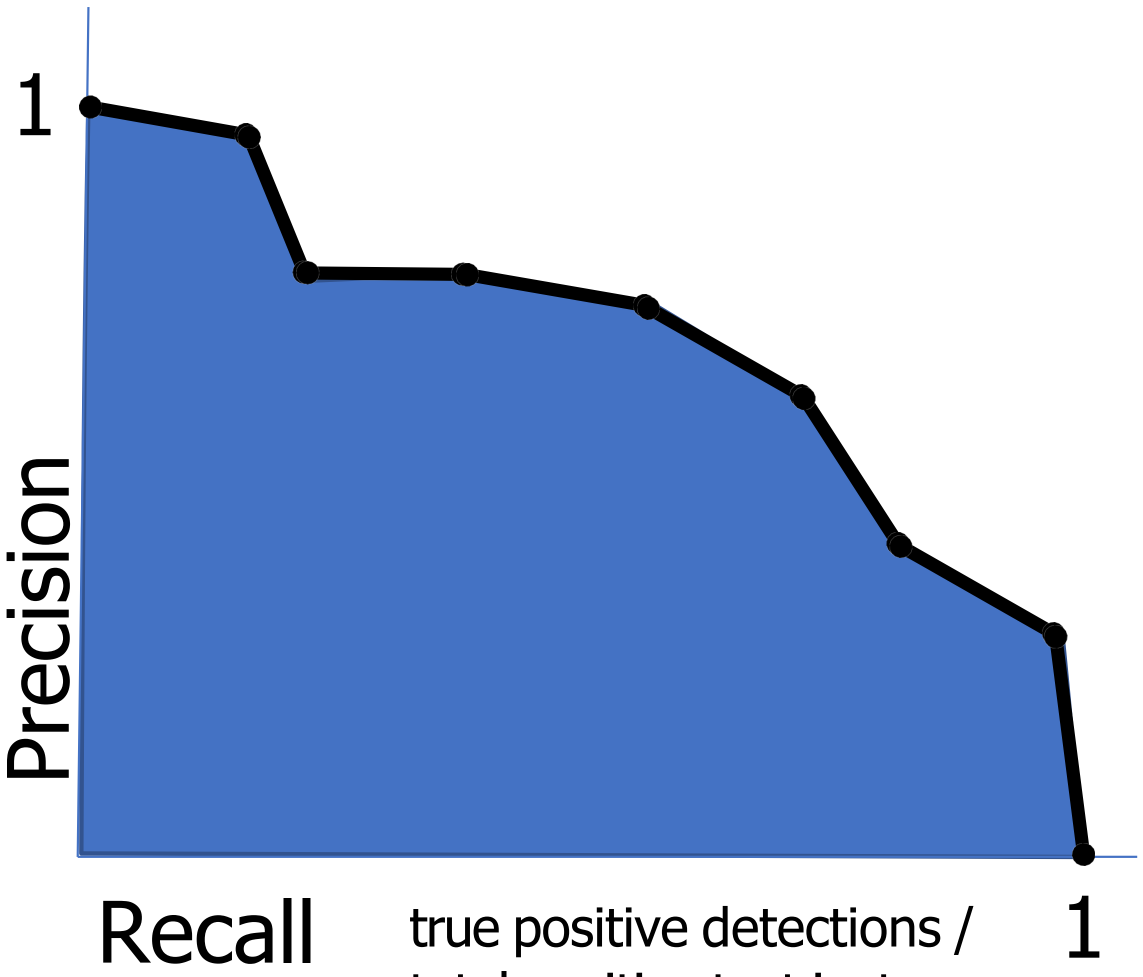
true pos. /
total detections




true positive detections /
total positive test instances

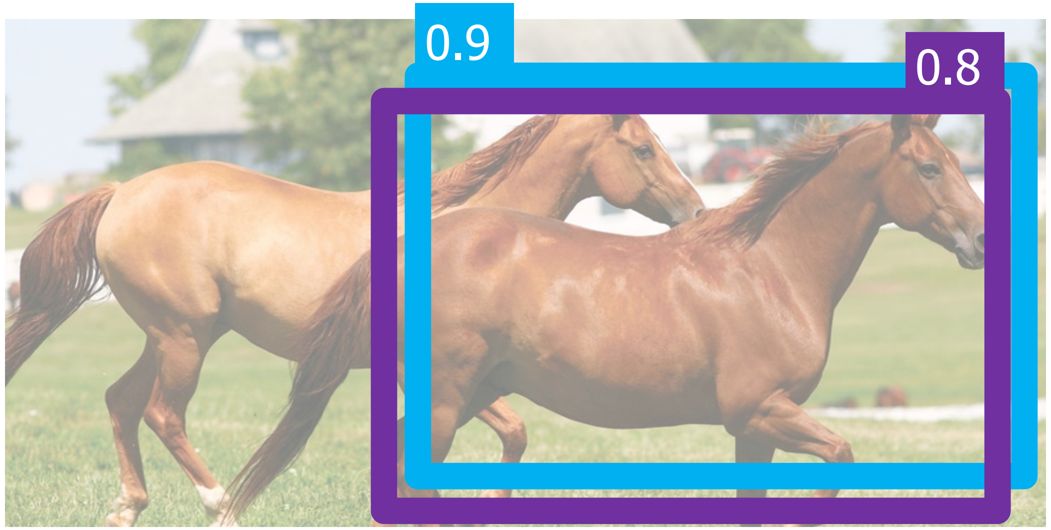
# Average precision

true pos. /
total detections




true positive detections /
total positive test instances

# Non-maximum suppression




- Subtlety: we predict a bounding box for every sliding window. Which ones should we keep?
- Keep only "peaks" in detector response.
- Discard low-prob boxes near high-prob ones
- Often use a simple greedy algorithm

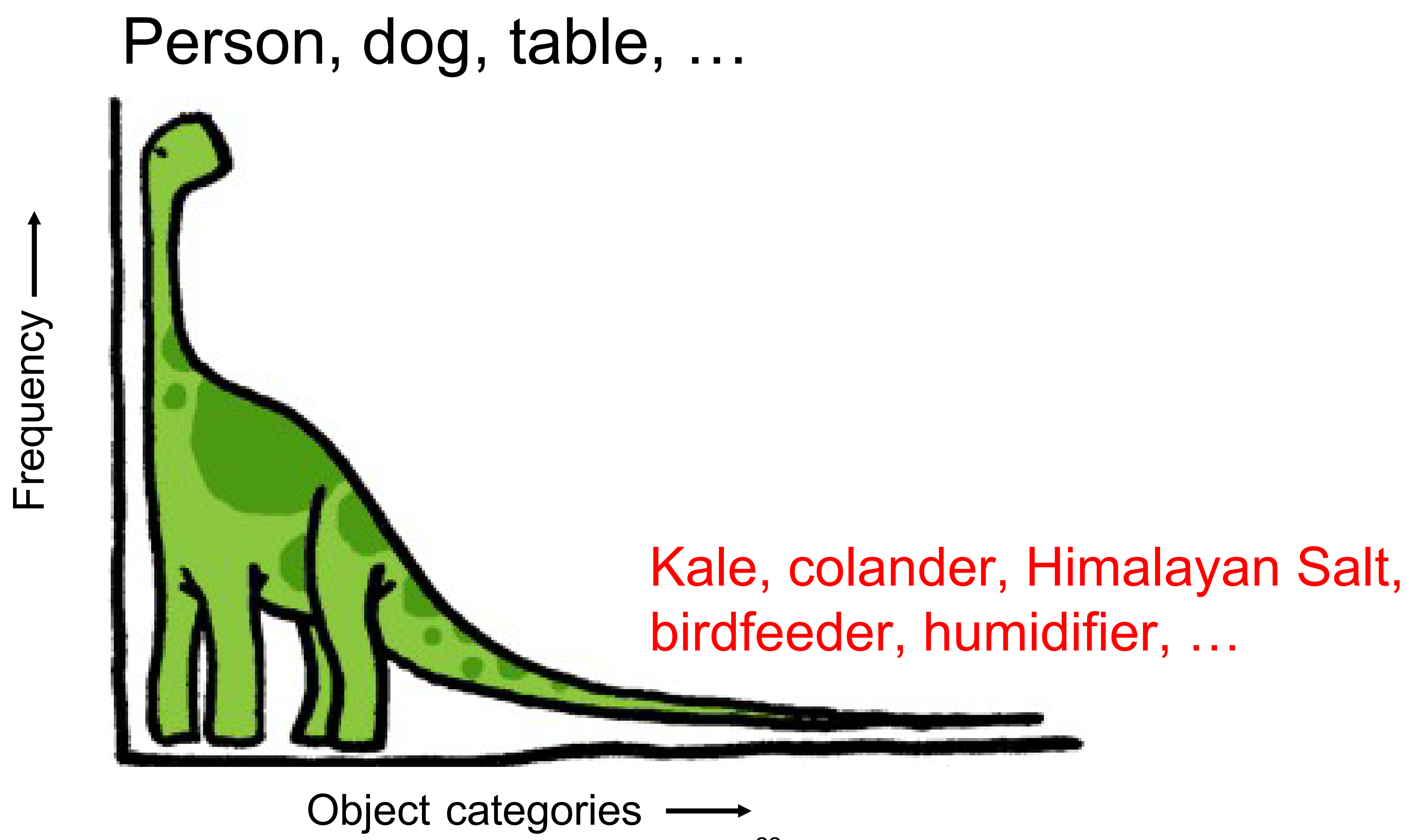
# Non-maximum suppression

Greedy algorithm, run on each class independently

let    be the set of all bounding boxes
let be the set of detections we'll keep, $D = \varnothing$
while $A \neq \varnothing$:
remove    the box with highest probability from
if    doesn't significantly overlap with an existing box in      (e.g. IoU > 0.5):
$D = D \cup \{x\}$
return

- Introduction to scene understanding
- Object detection models
- Evaluating object detectors
- **Challenges**

# Handle the long tail of the distribution



# Handle the “long tail” of the distribution


From COCO (80 categories)
[Lin et al., 2014]


LVIS dataset (1000+ categories)
“Few shot” (e.g. < 20 examples)
[Gupta et al., 2019]

Image source: R. Girshick

# OWL-ViT: Open-vocabulary object detection model

[Submitted on 12 May 2022 (v1), last revised 20 Jul 2022 (this version, v2)]

## Simple Open-Vocabulary Object Detection with Vision Transformers

Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, Xiao Wang, Xiaohua Zhai, Thomas Kipf, Neil Houlsby

Combining simple architectures with large-scale pre-training has led to massive improvements in image classification. For object detection, pre-training and scaling approaches are less well established, especially in the long-tailed and open-vocabulary setting, where training data is relatively scarce. In this paper, we propose a strong recipe for transferring image-text models to open-vocabulary object detection. We use a standard Vision Transformer architecture with minimal modifications, contrastive image-text pre-training, and end-to-end detection fine-tuning. Our analysis of the scaling properties of this setup shows that increasing image-level pre-training and model size yield consistent improvements on the downstream detection task. We provide the adaptation strategies and regularizations needed to attain very strong performance on zero-shot text-conditioned and one-shot image-conditioned object detection. Code and models are available on GitHub.

Comments: ECCV 2022 camera-ready version

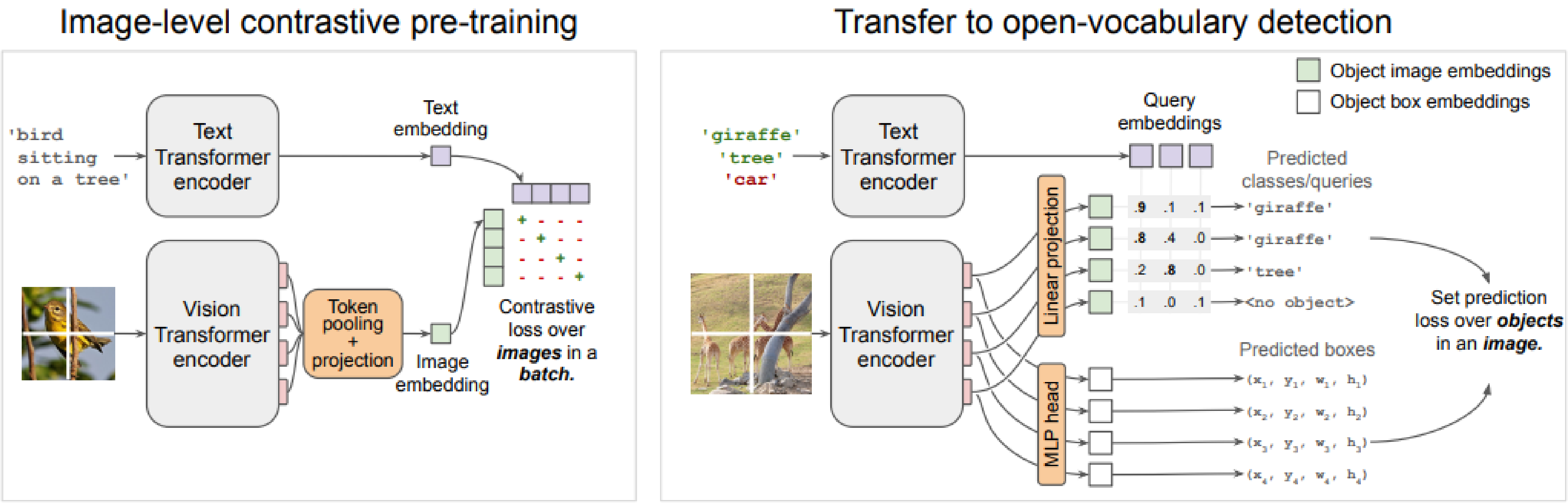



**Fig. 1.** Overview of our method. *Left:* We first pre-train an image and text encoder contrastively using image-text pairs, similar to CLIP [33], ALIGN [19], and LiT [44]. *Right:* We then transfer the pre-trained encoders to open-vocabulary object detection by removing token pooling and attaching light-weight object classification and localization heads directly to the image encoder output tokens. To achieve open-vocabulary detection, query strings are embedded with the text encoder and used for classification. The model is fine-tuned on standard detection datasets. At inference time, we can use text-derived embeddings for open-vocabulary detection, or image-derived embeddings for few-shot image-conditioned detection.

# OWL-ViT Demo

https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/zeroshot_object_detection_with_owlvit.ipynb

https://colab.research.google.com/drive/1evZkcsq4FTreFxGV6JXDmymnYcqWK43n