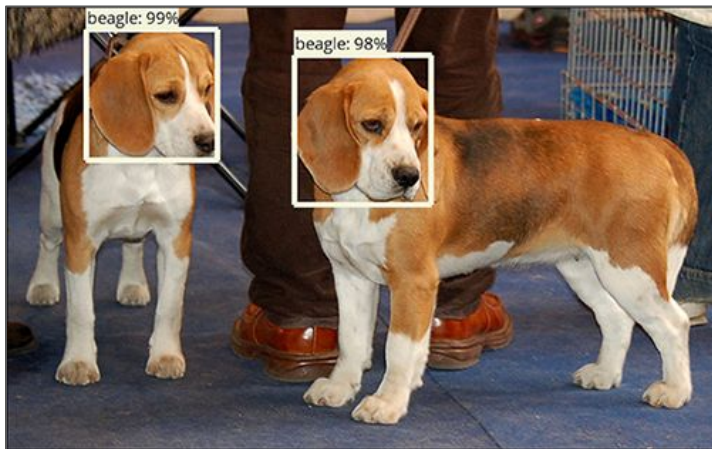


# CSEP 576: Object Detection



Jonathan Huang ([jonathanhuang@google.com](mailto:jonathanhuang@google.com))

University of Washington 17 May 2020

**Google Research**

# Lecture Outline

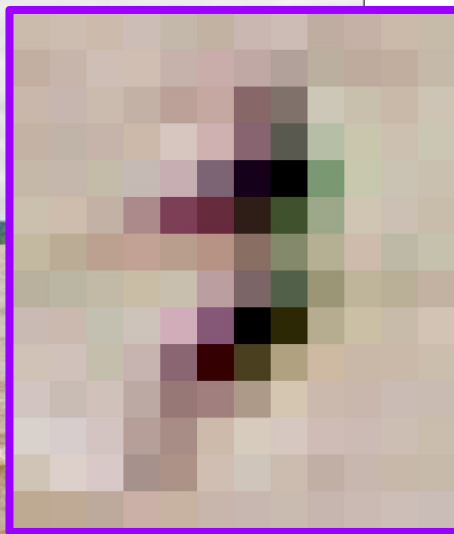
May 19

- Part 1: Advanced CNNs (Focusing on classification)
  - Reusable higher level building blocks of modern convnet architectures
    - Dropout, Batch Norm, Factorized Convolutions, Residual Connections, etc.
  - Tour through “popular” classification architectures
    - E.g., AlexNet, VGG, GoogLeNet, Resnet, MobileNet, SE-Net
- Part 2: Object Detection
  - Motivation, Applications
  - Anchor based detection methodology
  - Single stage and Two stage meta-architectures
  - Evaluation metrics
  - Practical Tips

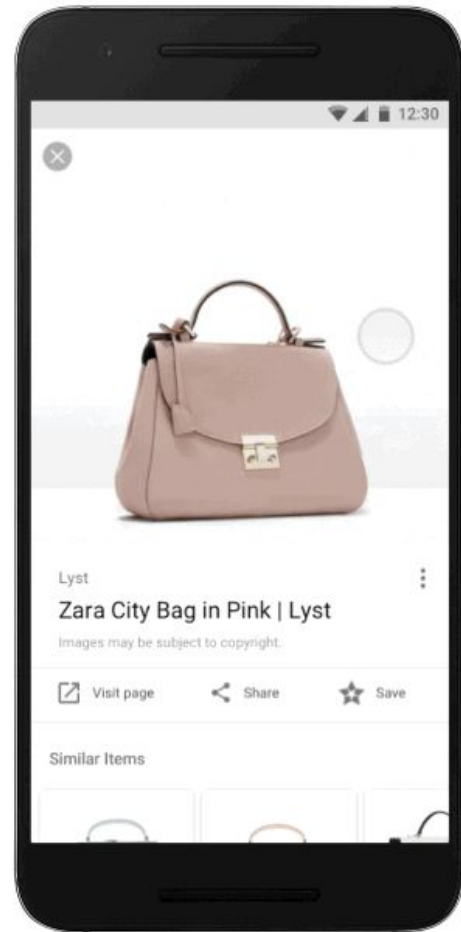
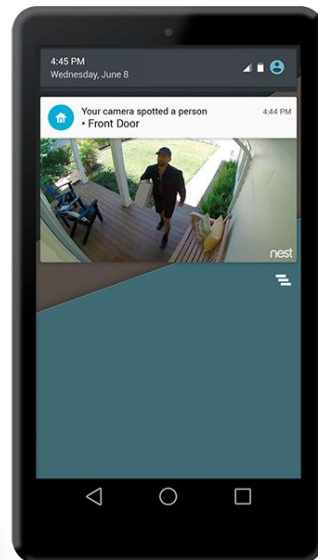
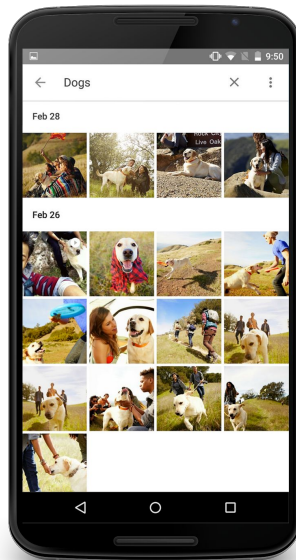
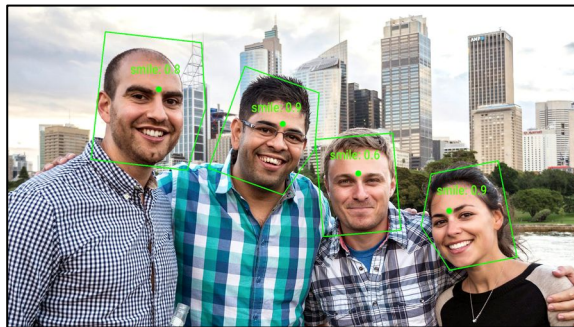
# From Classification to Detection

**Detection = Classification + Localization**

- Variable # outputs
  - *Need to classify based on much fewer pixels than in Imagenet setting; Requires context!*
- Usually need to operate on much larger images



# Object Detection Applications



# Object Detection Applications



# Object Detection Applications



# Object Detection Applications







# “Sliding Window” Detection

background



# “Sliding Window” Detection

background



# “Sliding Window” Detection

background



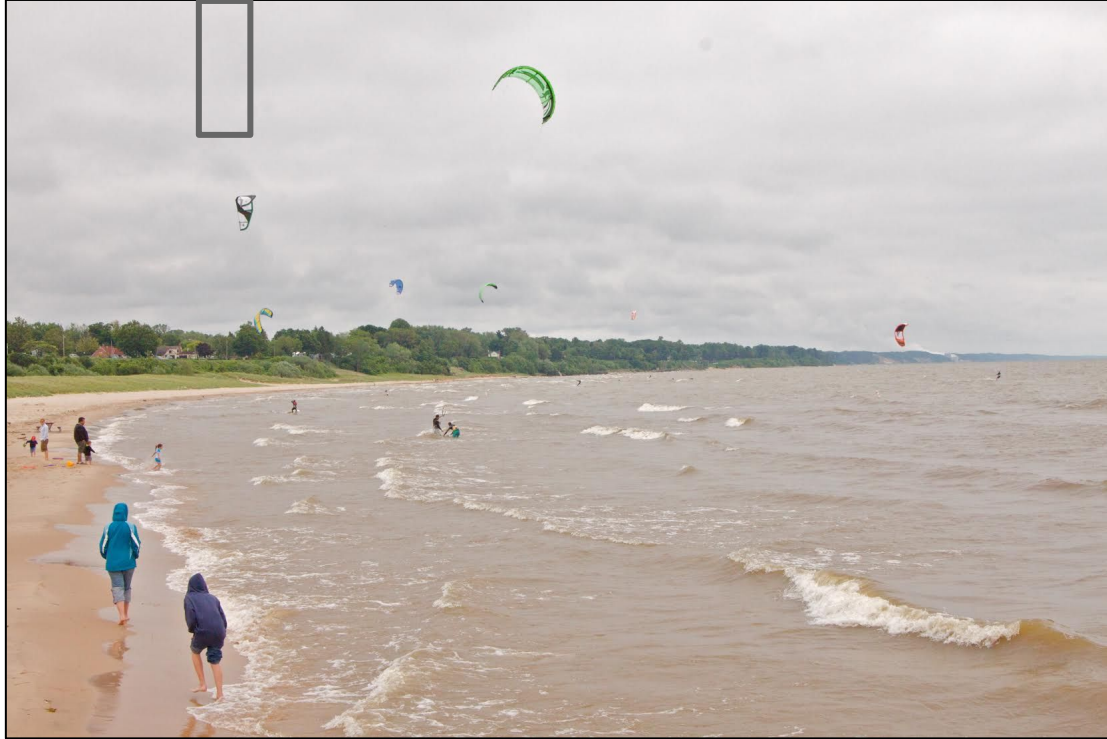
# “Sliding Window” Detection

background



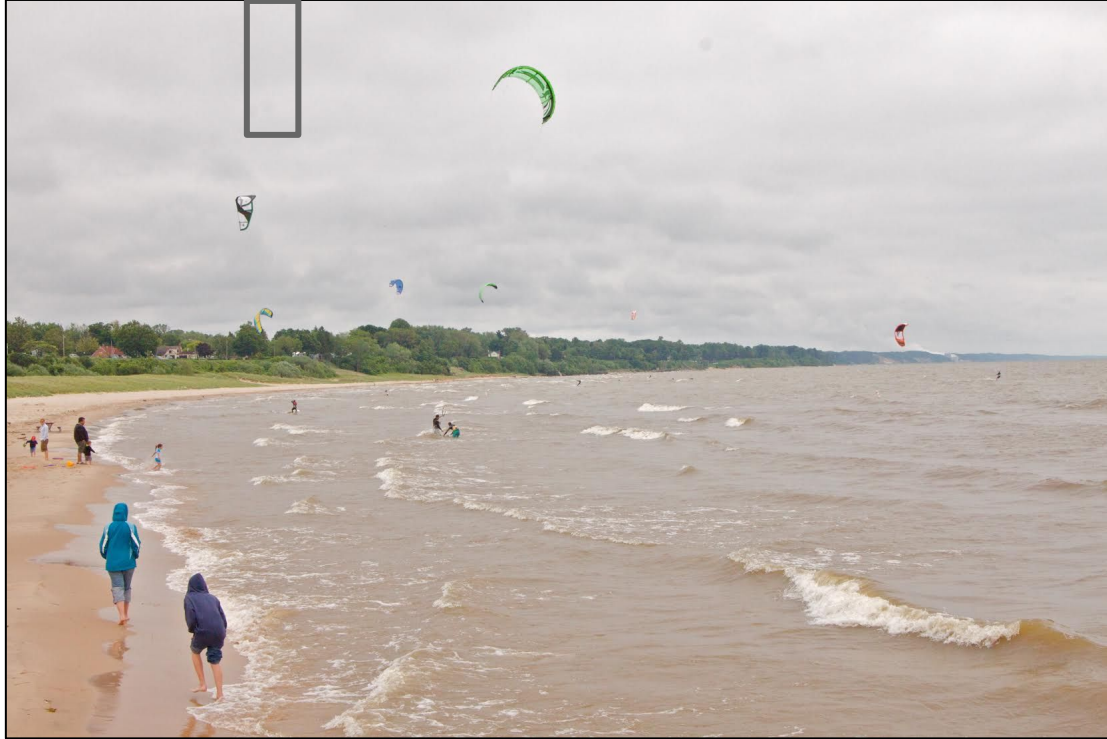
# “Sliding Window” Detection

background



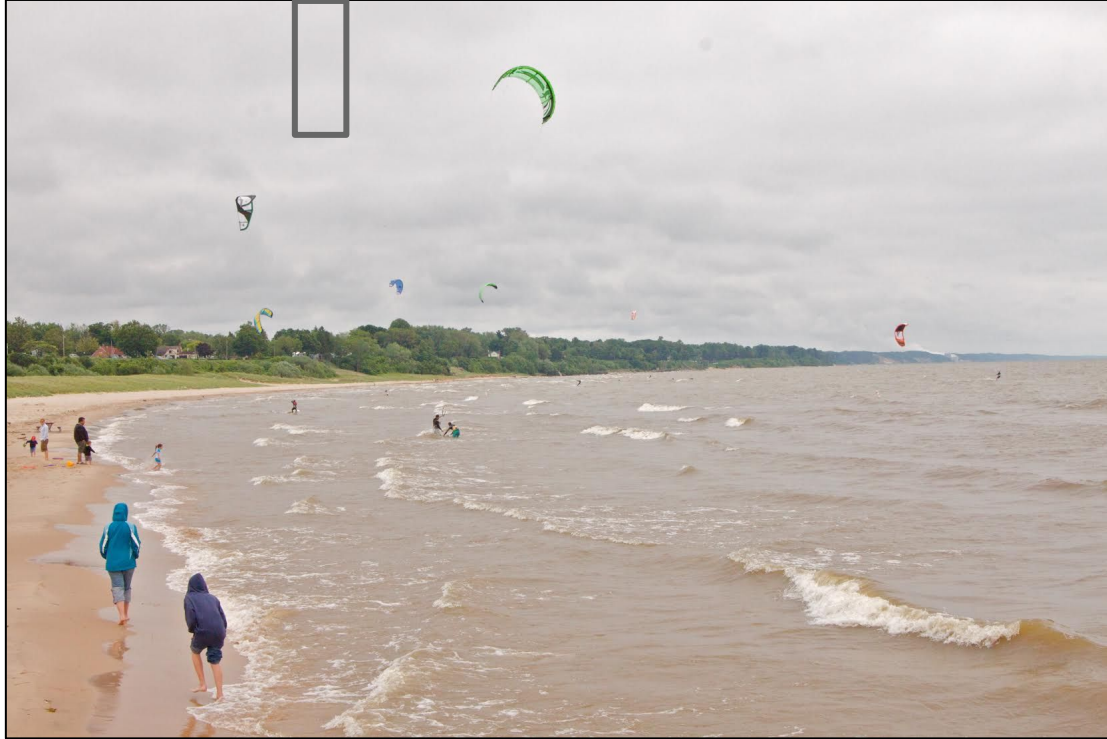
# “Sliding Window” Detection

background



# “Sliding Window” Detection

background



# “Sliding Window” Detection

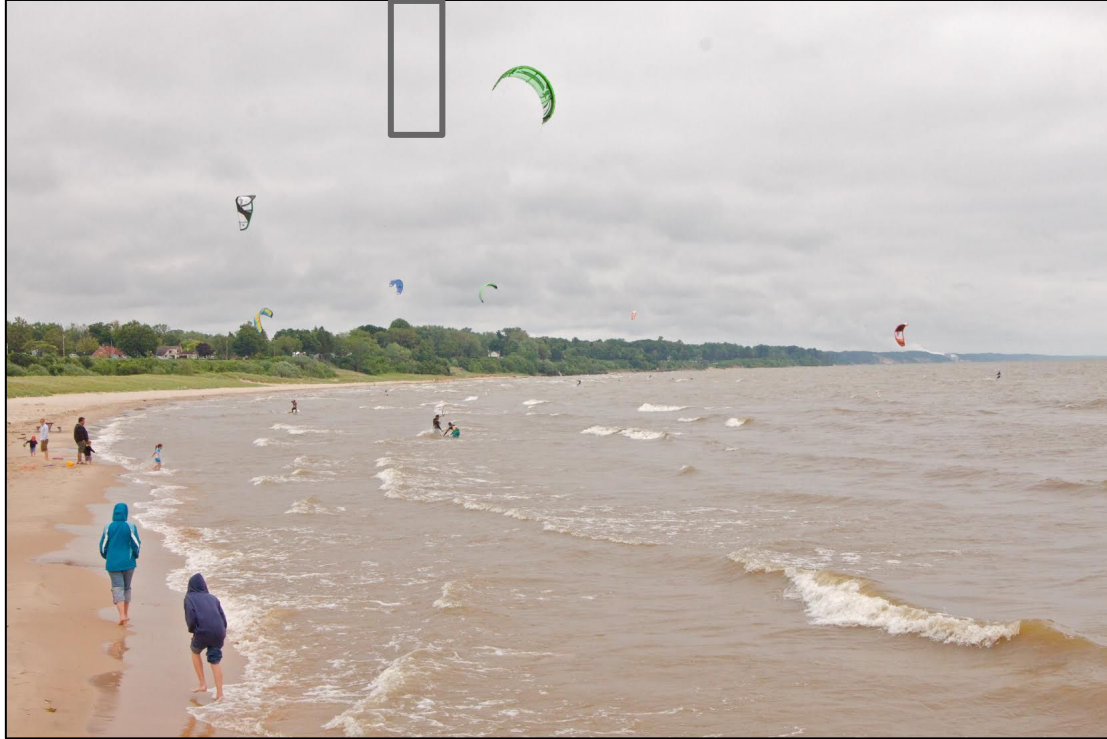
background





# “Sliding Window” Detection

background



# “Sliding Window” Detection



# “Sliding Window” Detection



# “Sliding Window” Detection



# “Sliding Window” Detection

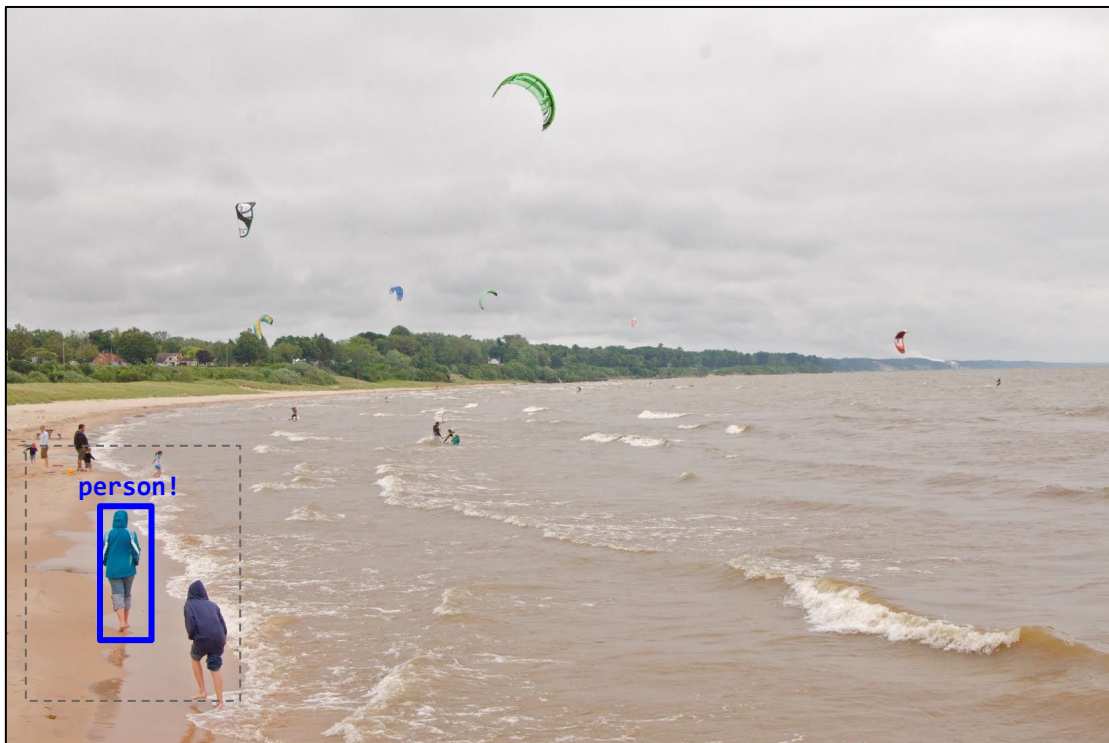


# “Sliding Window” Detection



Compute within-region features,  
then classify

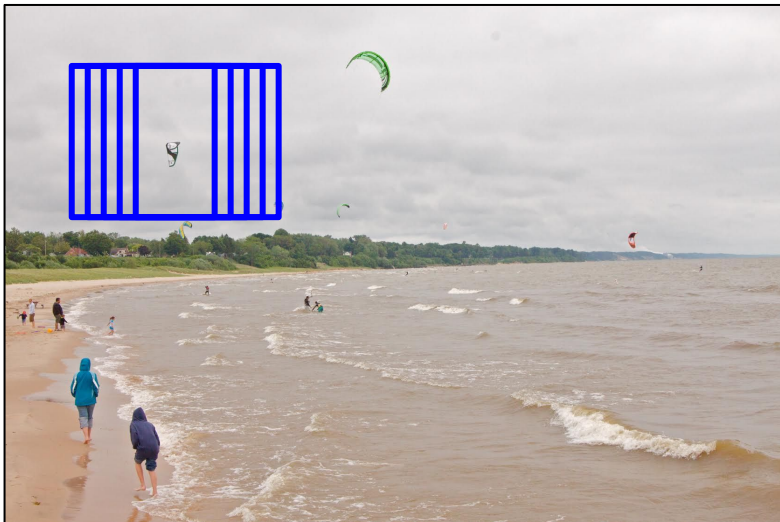
# “Sliding Window” Detection



Typical to enlarge region to include some “context”

# Sliding window placement

Slide over ***fine grid***  
in x, y, scale, aspect ratio



Slow and Accurate

Slide over ***coarse grid***  
in x, y, scale, aspect ratio



Fast and Not-so-accurate  
(... or can it be?)



# Bounding Box Regression

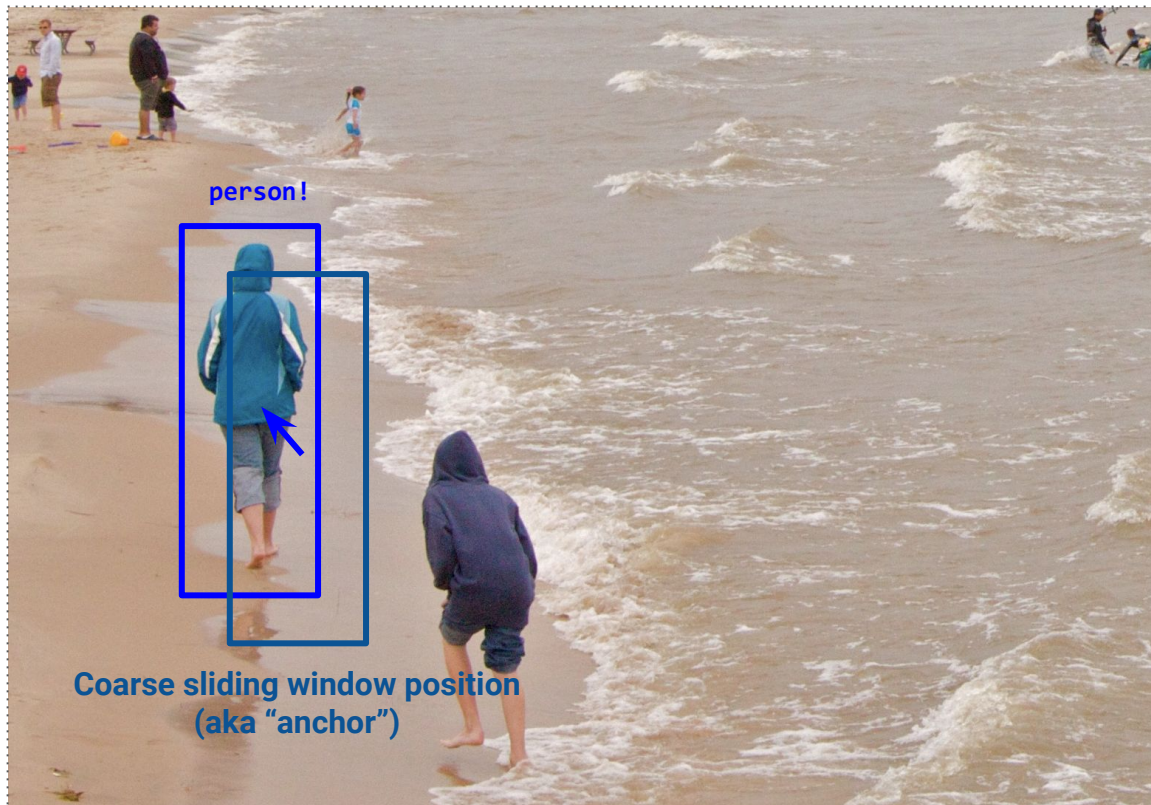


Coarse sliding window position  
(aka "anchor")

## Idea:

Also predict continuous offset from anchor to "snap" onto object

# Bounding Box Regression



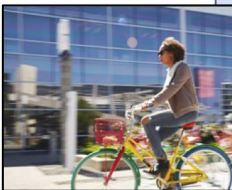
## Idea:

Also predict continuous offset from anchor to "snap" onto object

# Outline

- Sliding Window Detectors
- **Detection with Convolutional Networks**
- How to Evaluate a Detector
- Practical tips/tricks

# Using convolutional networks for detection



## Agenda for next few slides:

- Cover a simplified convnet approach for generating detections in detail;
- Touch on more modern architectures (all of which are based on the same concept)

Feature Extractor

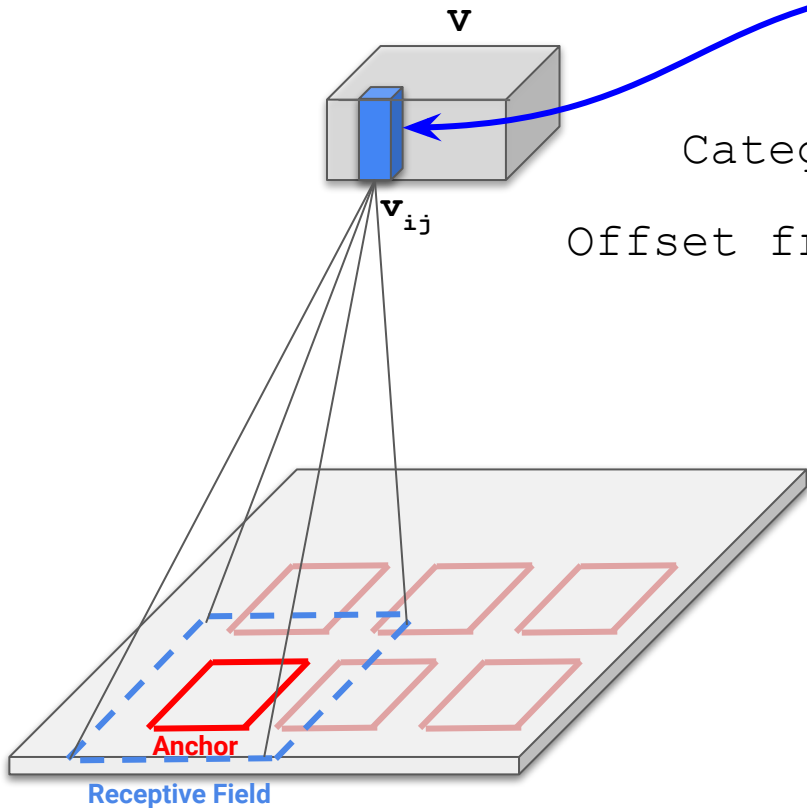
Detection Generator

Multiscale

- Extract features at sliding window positions via convolution
- Deep networks -> large receptive fields that can account for context

# A simplified convnet for detection

Think of each feature vector  $\mathbf{v}_{ij}$  as corresponding to a sliding window (anchor).

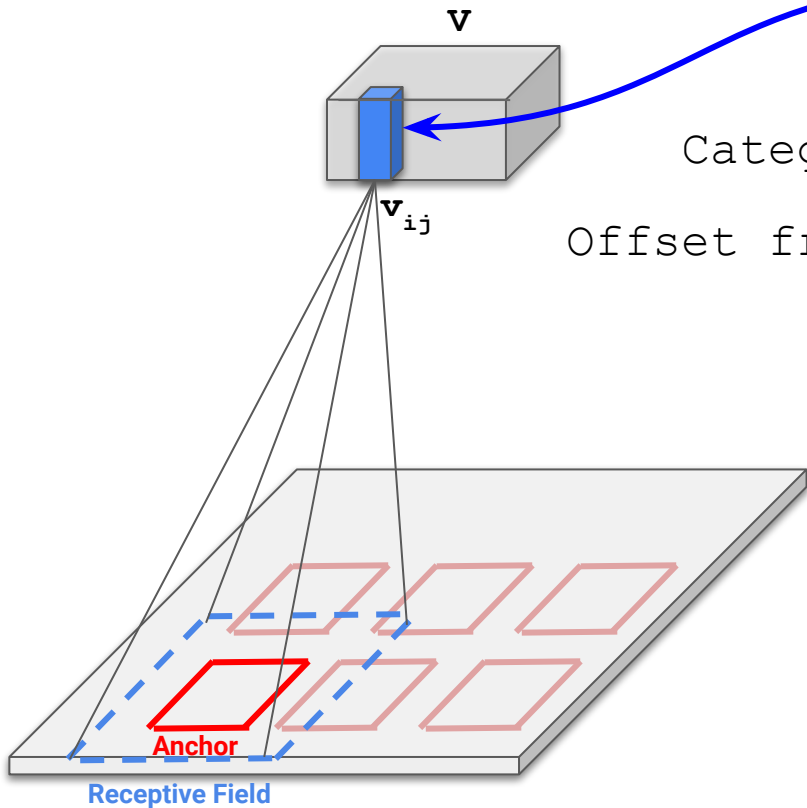


$$\text{Category score} = \text{SoftMax}(W^{\text{cls}} \cdot \mathbf{v}_{ij})$$

$$\text{Offset from anchor} = W^{\text{loc}} \cdot \mathbf{v}_{ij}$$

# A simplified convnet for detection

Think of each feature vector  $\mathbf{v}_{ij}$  as corresponding to a sliding window (anchor).



$$\text{Category score} = \text{SoftMax}(W^{\text{cls}} \cdot \mathbf{v}_{ij})$$

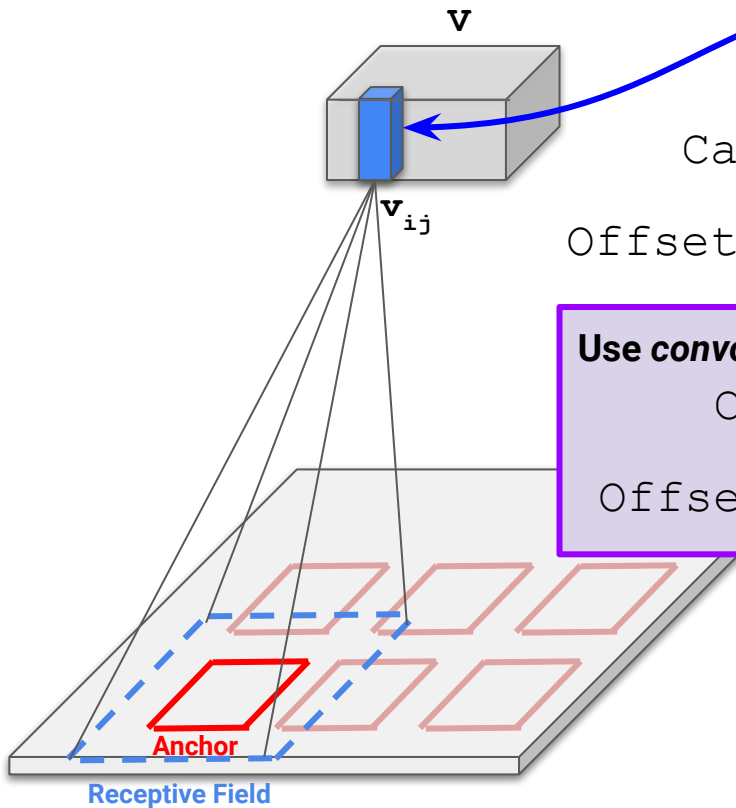
$$\text{Offset from anchor} = W^{\text{loc}} \cdot \mathbf{v}_{ij}$$

Use the same  $W^{\text{loc}}$  and  $W^{\text{cls}}$  for all  $i, j$  in anchor grid if anchors are:

- of the same shape, and
- contained and centered in receptive field

# A simplified convnet for detection

Think of each feature vector  $\mathbf{v}_{ij}$  as corresponding to a sliding window (anchor).



$$\text{Category score} = \text{SoftMax}(W^{\text{cls}} \cdot \mathbf{v}_{ij})$$

$$\text{Offset from anchor} = W^{\text{loc}} \cdot \mathbf{v}_{ij}$$

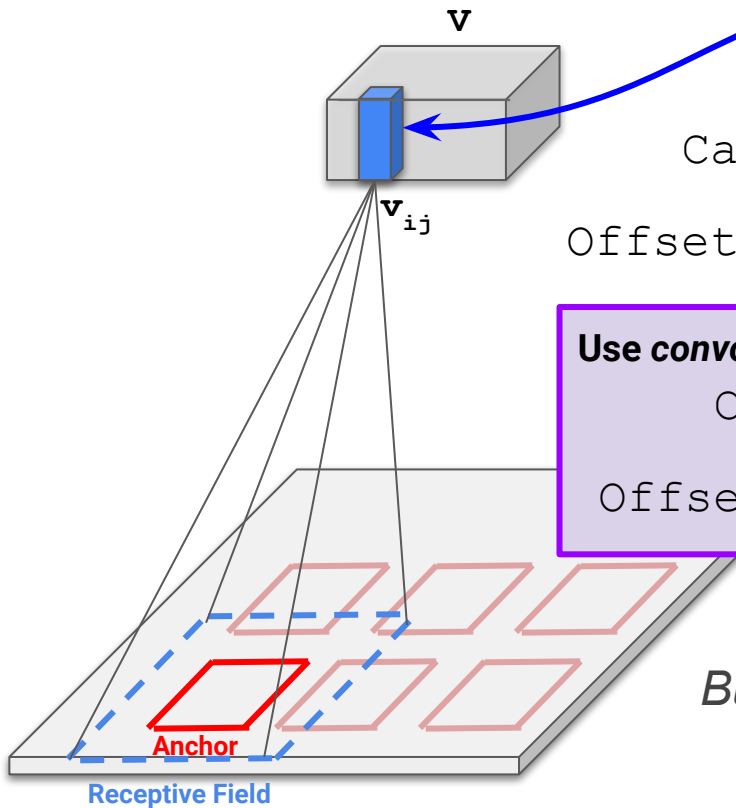
**Use *convolution* to do simultaneous prediction for all anchors:**

$$\text{Category score} = \text{SoftMax}(\text{Conv}(\mathbf{v}; W^{\text{cls}}))$$

$$\text{Offset from anchor} = \text{Conv}(\mathbf{v}; W^{\text{loc}})$$

# A simplified convnet for detection

Think of each feature vector  $\mathbf{v}_{ij}$  as corresponding to a sliding window (anchor).



$$\text{Category score} = \text{SoftMax}(W^{\text{cls}} \cdot \mathbf{v}_{ij})$$

$$\text{Offset from anchor} = W^{\text{loc}} \cdot \mathbf{v}_{ij}$$

**Use *convolution* to do simultaneous prediction for all anchors:**

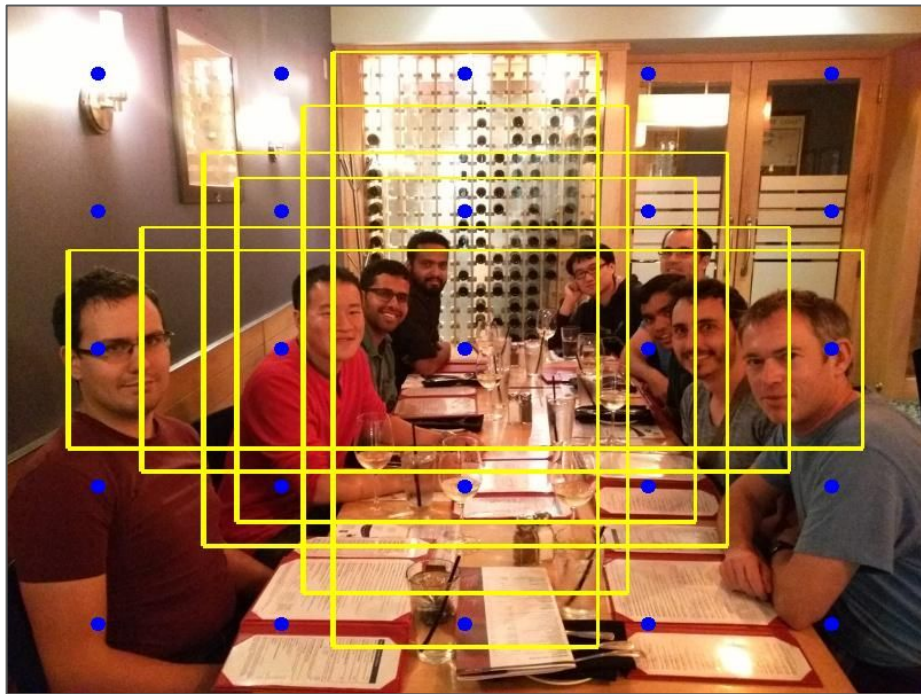
$$\text{Category score} = \text{SoftMax}(\text{Conv}(\mathbf{v}; W^{\text{cls}}))$$

$$\text{Offset from anchor} = \text{Conv}(\mathbf{v}; W^{\text{loc}})$$

*But... if anchors need to be the same shape, how do we handle different scales/aspect ratios?*



Solution: use multiple  $w^{loc}$  and  $w^{cls}$  (one for each aspect ratio/scale)



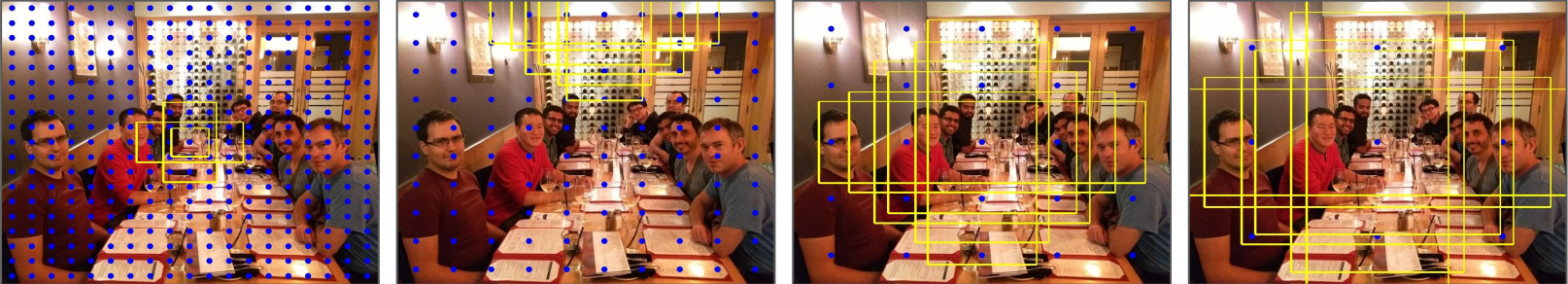
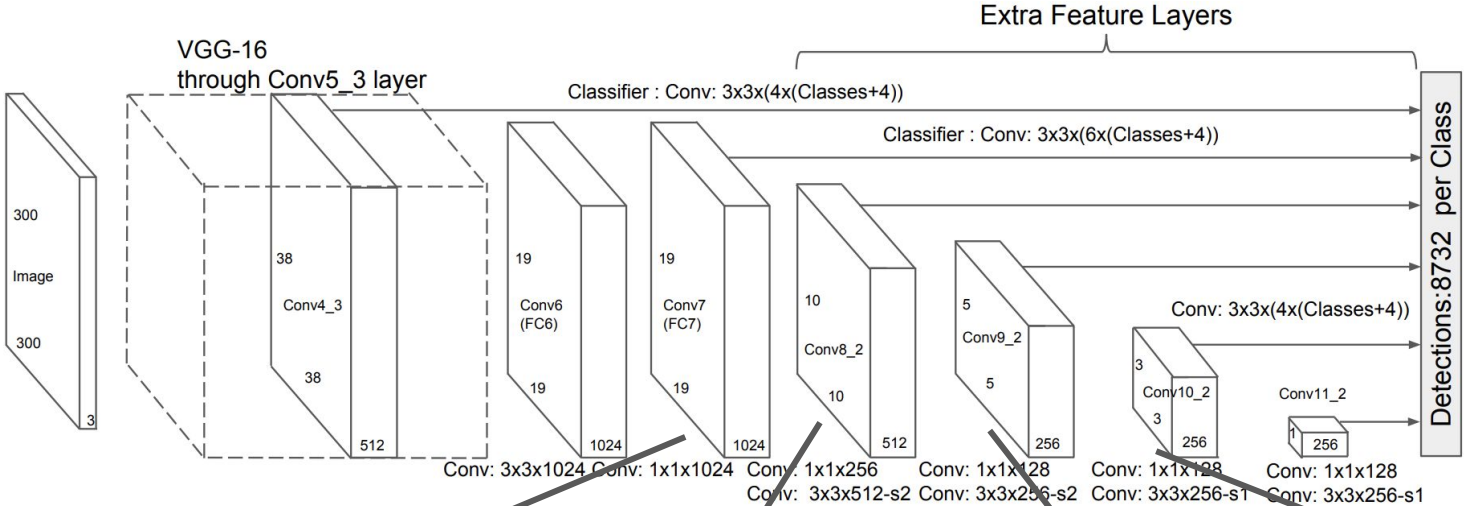
$$\text{SoftMax}(W^{cls, ar1} \cdot \mathbf{v}_{ij})$$
$$W^{loc, ar1} \cdot \mathbf{v}_{ij}$$

$$\text{SoftMax}(W^{cls, ar2} \cdot \mathbf{v}_{ij})$$
$$W^{loc, ar2} \cdot \mathbf{v}_{ij}$$

$$\text{SoftMax}(W^{cls, ar3} \cdot \mathbf{v}_{ij})$$
$$W^{loc, ar3} \cdot \mathbf{v}_{ij}$$

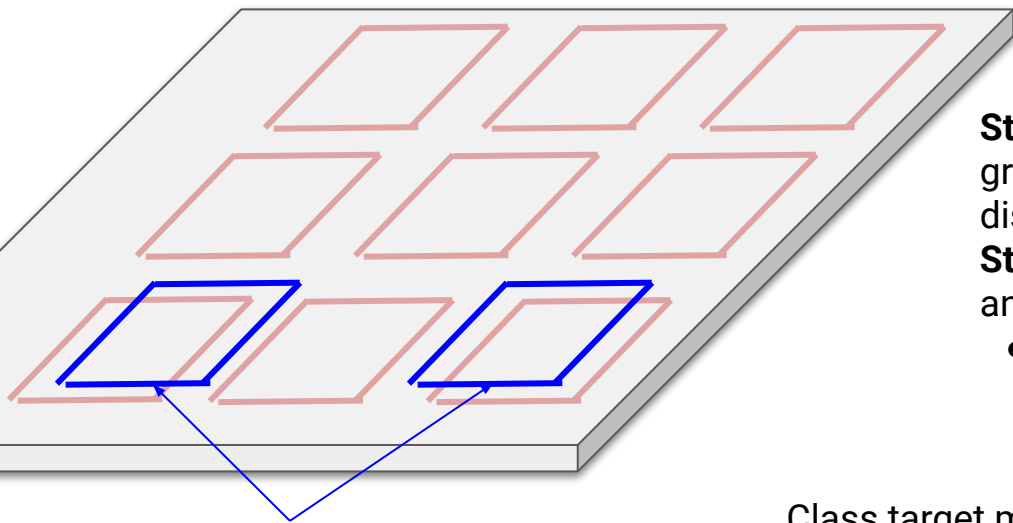
...

# Fancier Solution: use multiple anchor grid resolutions



[Liu et al 2016]

# Target Assignment



groundtruth boxes (person, class 2)

**Step 1:** Match anchor boxes to groundtruth boxes (based on Euclidean distance or overlap area)

**Step 2:** Give each anchor a classification and regression target

- If anchor has no matching groundtruth, it classifies as 0 and no regression target is given

Class target matrix  
(one entry per anchor)

0	0	0
0	0	0
2	0	2

Location targets  
(only for matched anchors)

$$\left( \begin{array}{l} \text{gt}_{x_{\min}} - \text{anchor}_{x_{\min}} \\ \text{gt}_{y_{\min}} - \text{anchor}_{y_{\min}} \\ \text{gt}_{x_{\max}} - \text{anchor}_{x_{\max}} \\ \text{gt}_{y_{\max}} - \text{anchor}_{y_{\max}} \end{array} \right)$$

# Typical Training Objective

## Per-anchor Loss:

$$L(\text{anchor } \mathbf{a}) = \alpha * \delta(\mathbf{a} \text{ has matching groundtruth}) * L_2(\mathbf{t}^{\text{loc}}, W^{\text{loc}} \cdot \mathbf{v}_{ij}) \\ + \beta * \text{SoftMaxCrossEntropy}(\mathbf{t}^{\text{cls}}, W^{\text{cls}} \cdot \mathbf{v}_{ij})$$

Common to use other location losses here...

**Total Loss:** Average per-anchor loss over anchors

Minimize w/SGD

# Localization Loss in Box Encoding Space

$$\text{Huber} \left( \begin{array}{c} \frac{x_c}{w_a} \\ \frac{y_c}{h_a} \\ \log w \\ \log h \end{array}, W^{\text{loc}} \cdot \mathbf{v}_i \right)_j$$

# Localization Loss in Box Encoding Space

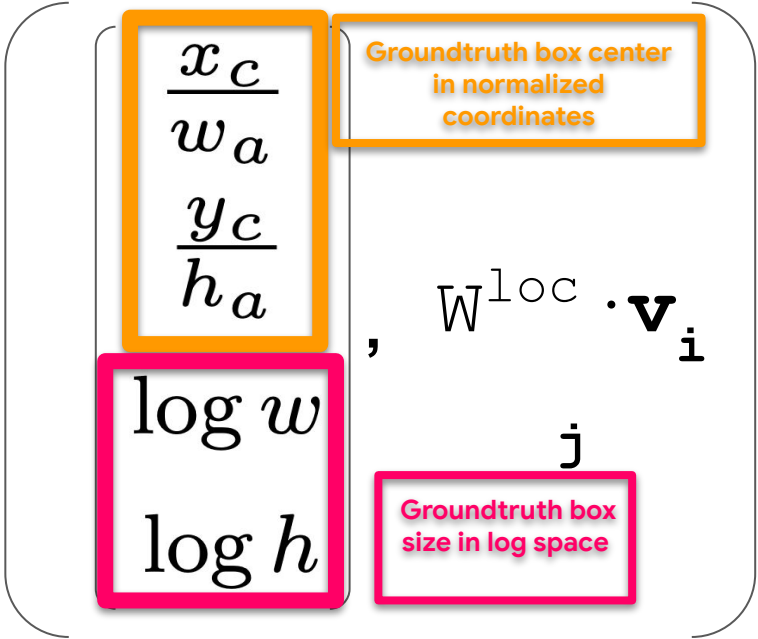
Huber

$$\left( \begin{array}{c} \left( \begin{array}{c} \frac{x_c}{w_a} \\ \frac{y_c}{h_a} \end{array} \right), W^{\text{loc}} \cdot \mathbf{v}_i \\ \log w \\ \log h \end{array} \right)$$

Groundtruth box center in normalized coordinates

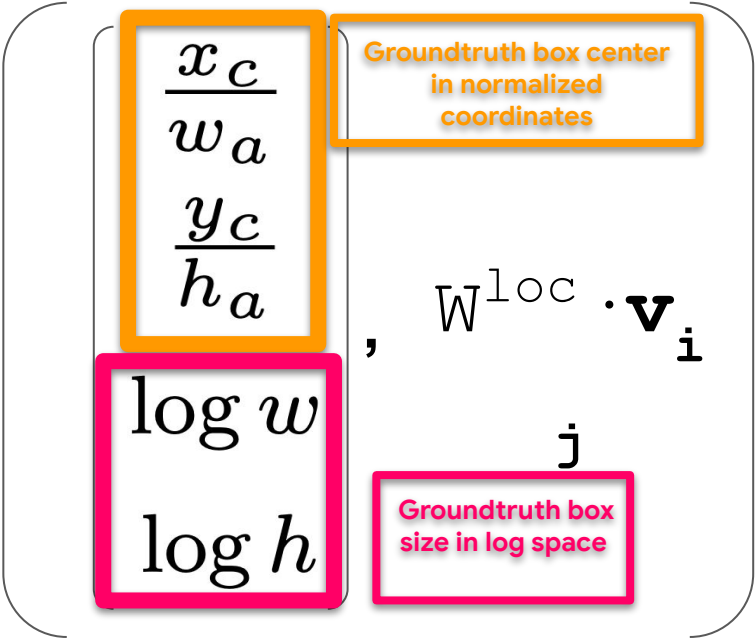
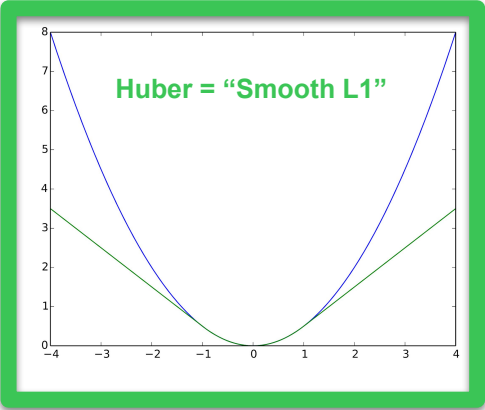
# Localization Loss in Box Encoding Space

Huber



# Localization Loss in Box Encoding Space

Huber





# Classification Loss: Dealing with Class Imbalance

## Problem:

# negative/background anchors  $\gg$  # positive/foreground anchors

## Typical solutions:

- Subsample negatives
- Downweight negatives
- Online hard mining (Srivastava et al.)
- Focal loss function (Lin et al.)



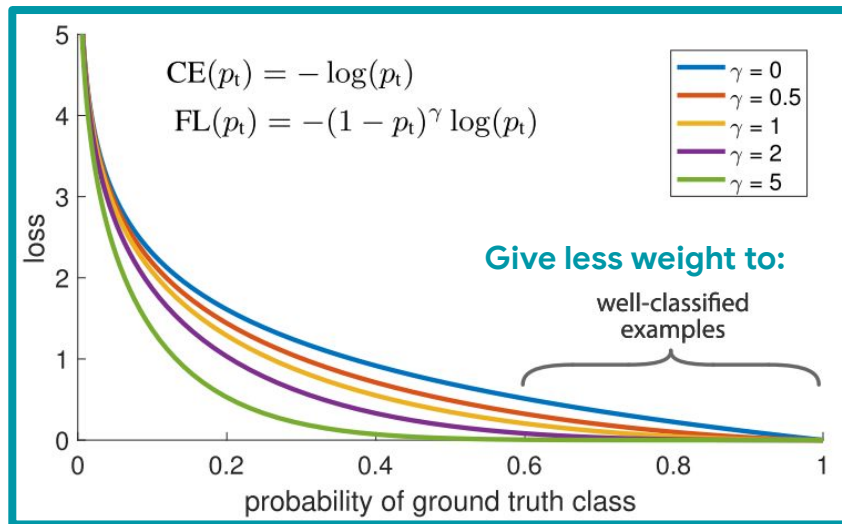
# Classification Loss: Dealing with Class Imbalance

## Problem:

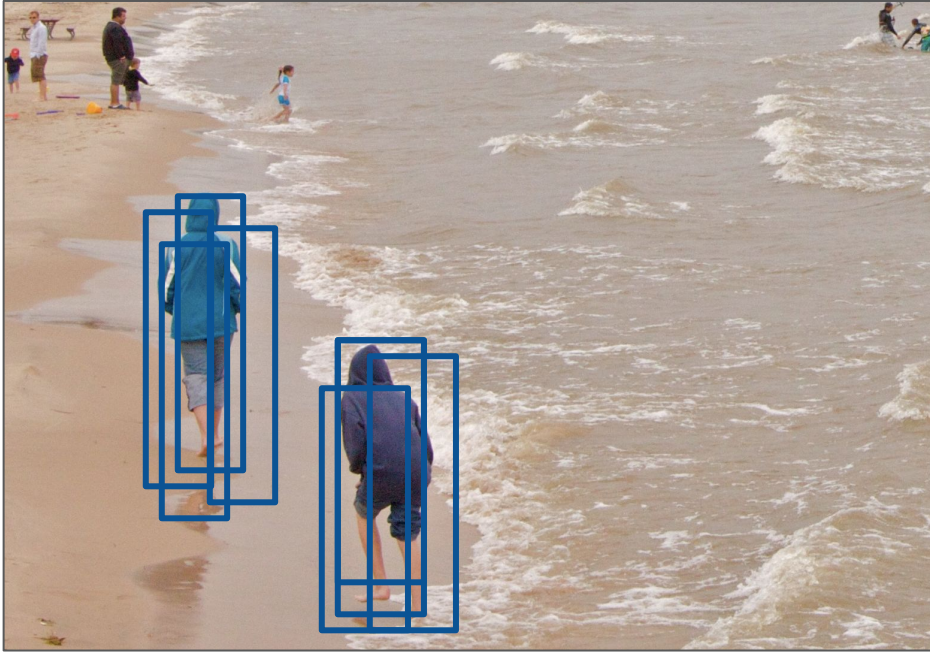
# negative/background anchors  $\gg$  # positive/foreground anchors

## Typical solutions:

- Subsample negatives
- Downweight negatives
- Online hard mining (Srivastava et al.)
- Focal loss function (Lin et al.)



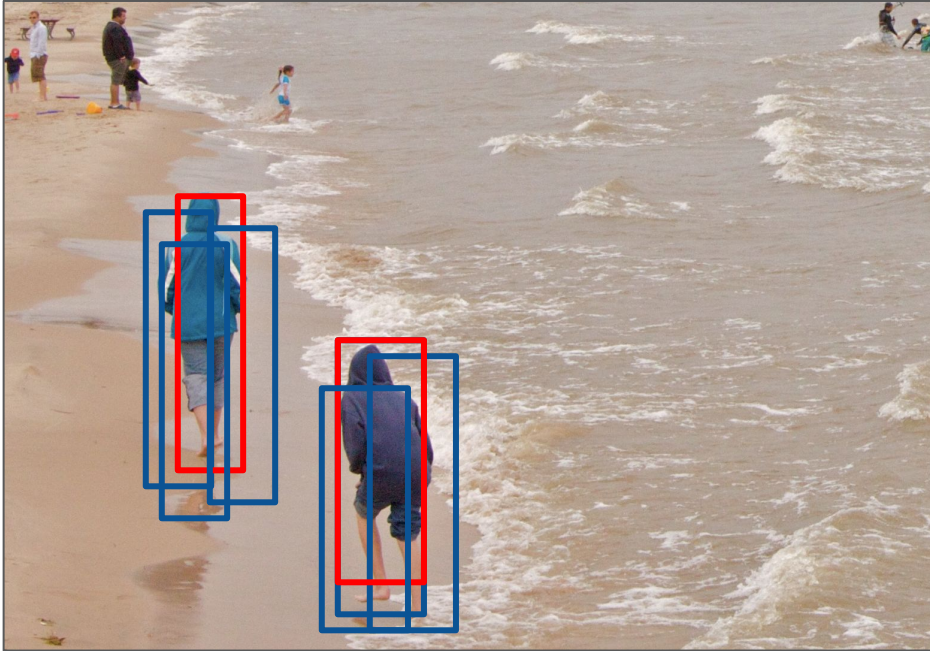
# Dealing with multiple detections of the same object



**Duplicate detection problem:** Typically many anchors will detect the same underlying object and give slightly different boxes, with slightly different scores.

**Solution:** remove detections if they overlap too much with another higher scoring detection.

# Non Max Suppression (NMS)



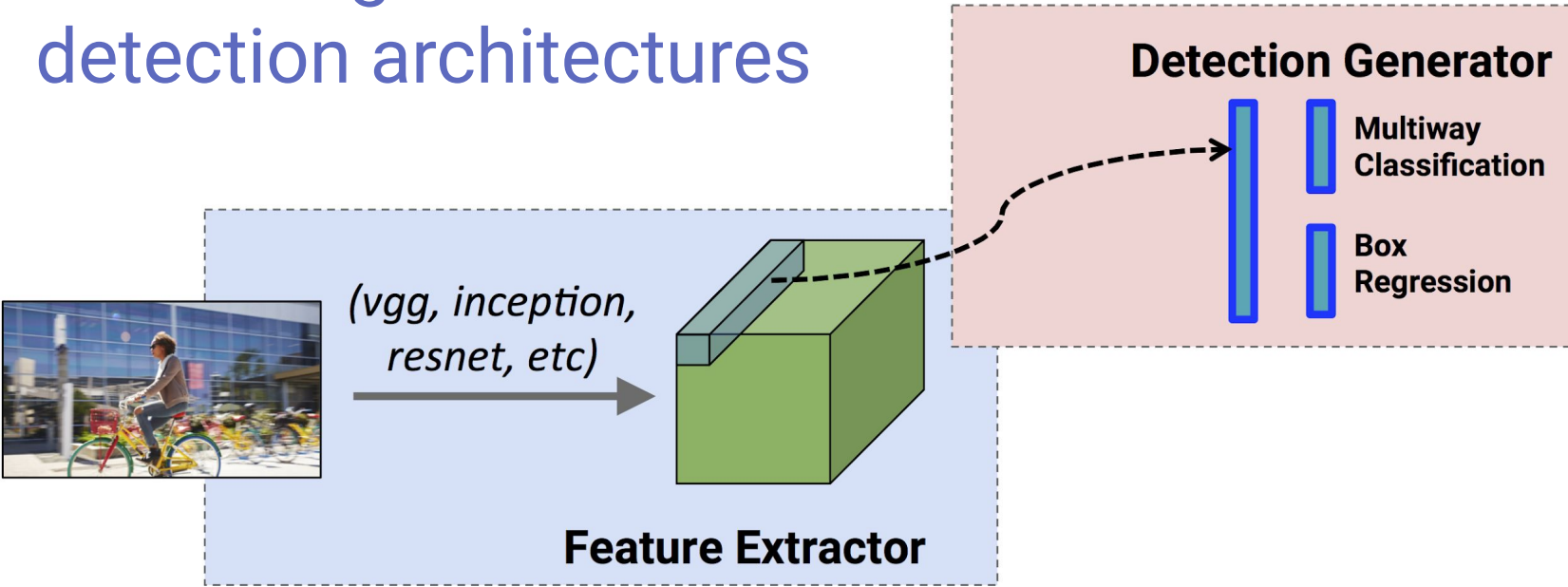
## Algorithm:

1. Sort detections in decreasing order with respect to score
2. Iterate through sorted detections:
  - a. Reject a detection if it overlaps with a previous (unrejected) detection with IOU greater than some threshold
3. Return all unrejected detections

## Some shortcomings of NMS to remember:

- Imposes a hard limitation on how close objects can be in order to be detected
- Similar classes do not suppress each other

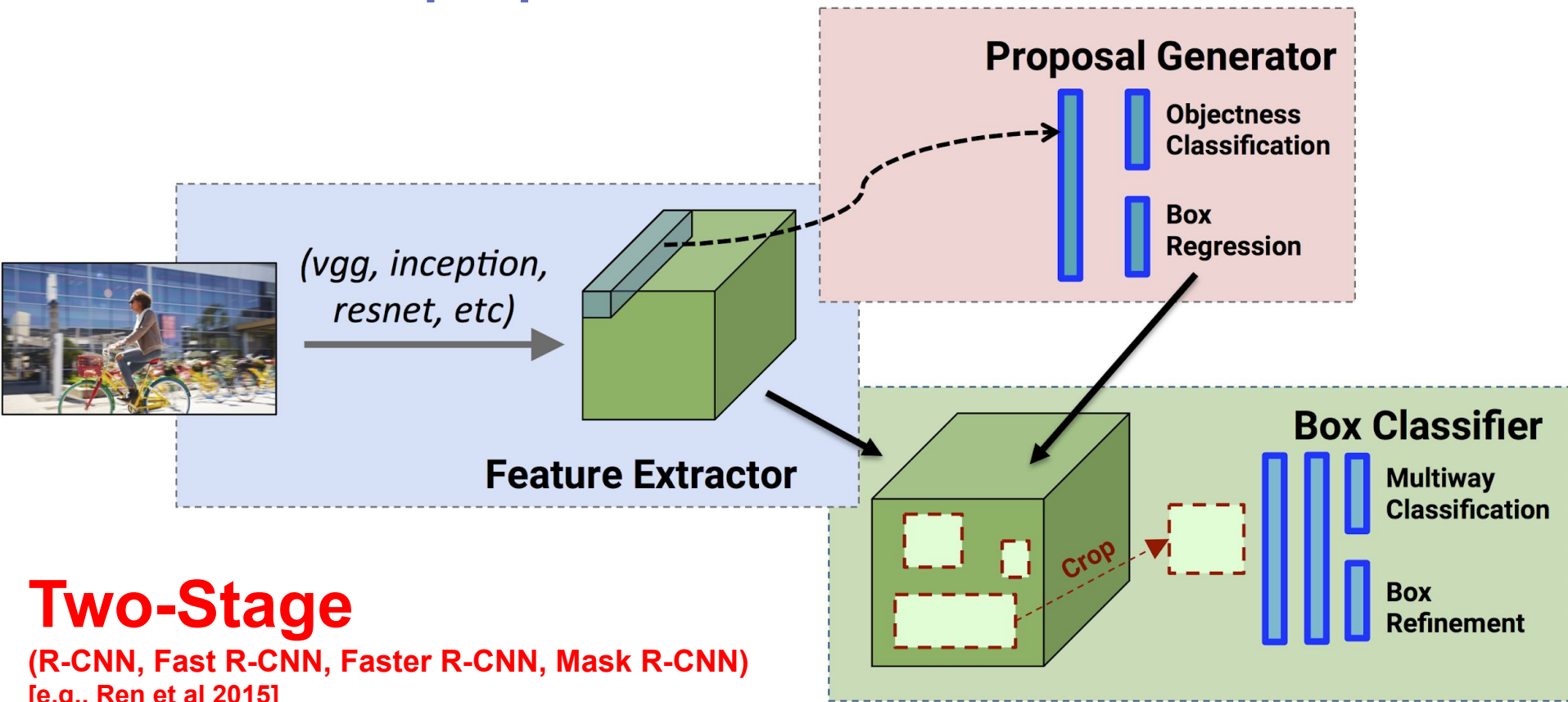
# Detection “*meta-architectures*” are a recipe for converting classification architectures into detection architectures



## Single Stage Models

(Encapsulates Multibox, SSD, YOLO, YOLO v2, RetinaNet)

# Another popular meta-architecture



# Single Stage Case Study: RetinaNet (Lin et al 2017)

## Key Ideas:

- **Multi-Resolution Feature Extractor:**

- **Resnet + FPN**

- **Focal Loss**

- **Smart initialization of classification**

- **bias for fast training:**

- **On Google TPUs, can train on**

- **COCO dataset in 3.5 hours**

arXiv:1612.03144v2 [cs.CV] 19 Apr 2017

### Feature Pyramid Networks for Object Detection

Tsung-Yi Lin<sup>1</sup>, Piotr Dollár<sup>1</sup>, Ross Girshick<sup>1</sup>, Kaiming He<sup>1</sup>, Bharath Hariharan<sup>1</sup>, and Serge Belongie<sup>2</sup>

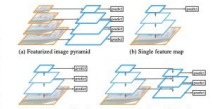
<sup>1</sup>Facebook AI Research (FAIR)  
<sup>2</sup>Cornell University and Cornell Tech

#### Abstract

Feature pyramids are a basic component in recognition systems for detecting objects at different scales. But recent deep learning object detectors have avoided pyramid representations, in part because they are compute and memory intensive. In this paper we explore the inherent multi-scale, pyramidal hierarchy of deep convolutional networks to construct feature pyramids with marginal extra cost. A top-down architecture with lateral connections is developed for building high-level semantic feature maps at all scales. This architecture, called a Feature Pyramid Network (FPN), shows significant improvement as a generic feature extractor in several applications. Using FPN in a basic Faster R-CNN system, our method achieves state-of-the-art single-model results on the COCO detection benchmark without bells and whistles, surpassing all existing single-model entries, including those from the COCO 2016 challenge winners. In addition, our method can run at 6 FPS on a GPU and thus is a practical and accurate solution to multi-scale object detection. Code will be made publicly available.

#### 1. Introduction

Recognizing objects at vastly different scales is a fundamental challenge in computer vision. Feature pyramids built upon image pyramids (for short we call these *featureized image pyramids*) form the basis of a standard solution [1] (Fig. 1(a)). These pyramids are scale-tolerant in the sense that an object's scale change is offset by shifting its level in the pyramid. Intuitively, this property enables a model to detect objects across a large range of scales by scanning the model over both positions and pyramid levels. Featureized image pyramids were heavily used in the era of hand-engineered features [5, 7, 2]. They were so critical that object detectors like SPM [1] required multi-scale sampling to achieve good results (i.e., 10 scales per octave). For recognition tasks, engineered features were



largely been replaced with features computed by deep convolutional networks (ConvNets) [3, 10]. Aside from being capable of representing high-level semantics, ConvNets are also more robust to variance in scale and thus facilitate recognition from features computed on a single input scale [15, 11, 10] (Fig. 1(b)). But even with this robustness, pyramids are still needed to get the most accurate results. All recent top entries in the ImageNet [13] and COCO [1] detection challenges use multi-scale testing on featureized image pyramids (e.g., [16, 10]). The principle advantage of featureizing each level of an image pyramid is that it produces a multi-scale feature representation in which all levels are semantically strong, including the high-resolution levels. Nevertheless, featureizing each level of an image pyramid has obvious limitations. Inference time increases considerably (e.g., by four times [11]), making this approach impractical for real applications. Moreover, training deep

arXiv:1708.02002v2 [cs.CV] 7 Feb 2018

### Focal Loss for Dense Object Detection

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, Facebook AI Research (FAIR)

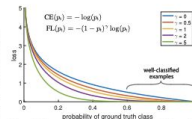


Figure 1: We propose a novel loss we term the *Focal Loss* that adds a factor  $(1 - p_t)^{\gamma}$  to the standard cross entropy criterion. Setting  $\gamma > 0$  reduces the relative loss for well-classified examples (e.g.  $\gamma = 4$ , putting more focus on hard, misclassified examples). As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

#### Abstract

The highest accuracy object detectors to date are based on a two-stage approach popularized by R-CNN, where a classifier is applied to a sparse set of candidate object locations. In contrast, one-stage detectors that are applied over a regular, dense sampling of possible object locations have the potential to be faster and simpler, but have traded the accuracy of two-stage detectors that fit. In this paper, we investigate why this is the case. We discover that the extreme foreground-background class imbalance encountered during training of dense detectors is the central cause. We propose to address this class imbalance by reshaping the standard cross entropy loss such that it downweights the loss assigned to well-classified examples. Our novel Focal Loss focuses training on a sparse set of hard examples and prevents the vast number of easy negatives from overwhelming the detector during training. To evaluate the effectiveness of our loss, we design and train a simple dense detector we call RetinaNet. Our results show that when trained with the focal loss, RetinaNet is able to match the speed of previous one-stage detectors while surpassing the accuracy of all existing state-of-the-art two-stage detectors. Code is at: [https://github.com/facebookresearch/focal\\_loss](https://github.com/facebookresearch/focal_loss).

#### 1. Introduction

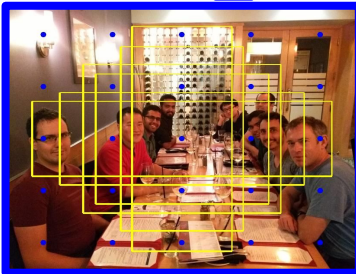
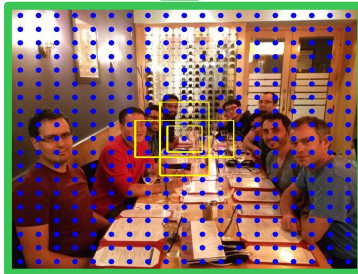
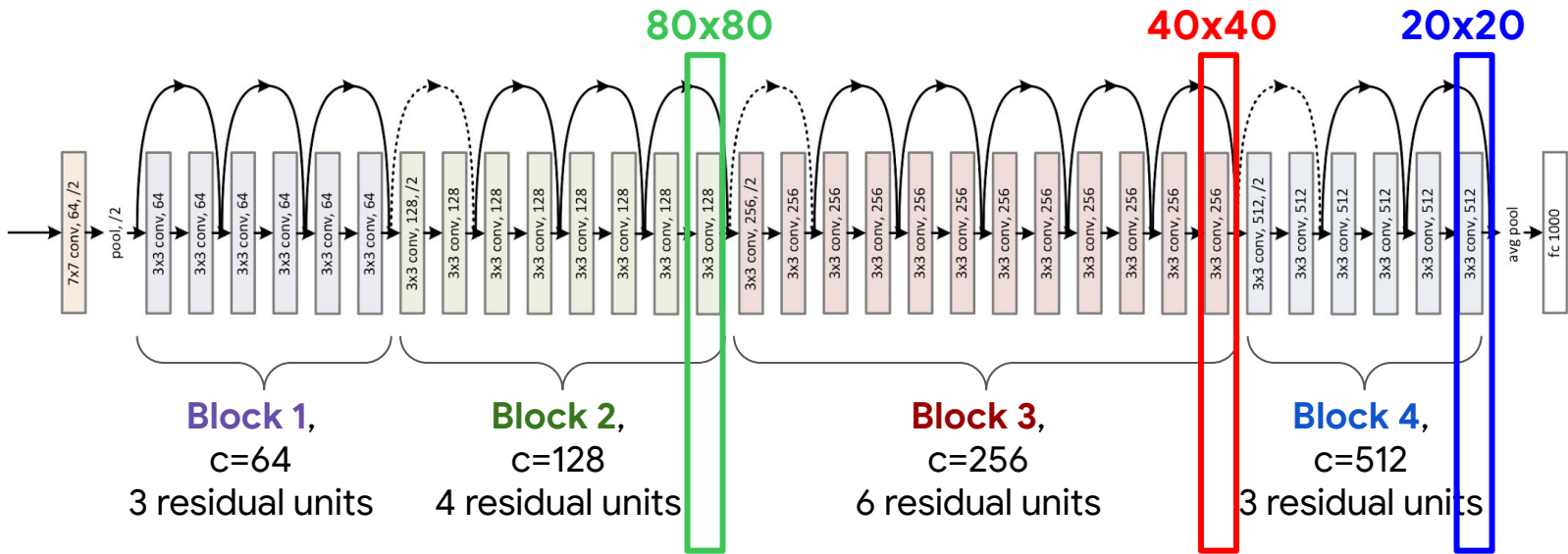
Current state-of-the-art object detectors are based on a two-stage, proposal-driven mechanism. As popularized in the R-CNN framework [11], the first stage generates a sparse set of candidate object locations and the second stage classifies each candidate location as one of the foreground classes or as background using a convolutional neural network. Through a sequence of advances [10, 25, 26, 16], this two-stage framework consistently achieves top accuracy on the challenging COCO benchmark [21].

Despite the success of two-stage detectors, a natural question to ask is: could a simple one-stage detector achieve similar accuracy? One-stage detectors are applied over a regular, dense sampling of object locations, scales, and aspect ratios. Recent work on one-stage detectors, such as YOLO [26, 27] and SSD [22, 25], demonstrates promising results, yielding faster detectors with accuracy within 10-40% relative to state-of-the-art two-stage methods.

This paper pushes the envelope further: we present a one-stage object detector that, for the first time, matches the state-of-the-art COCO AP of more complex two-stage det-

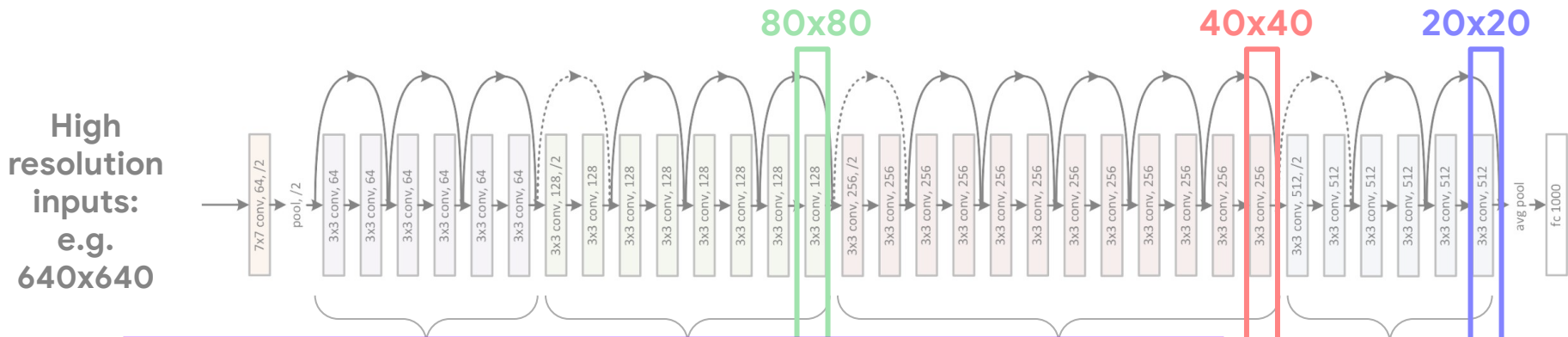
# One way to extracting multiresolution feature maps from Resnet

High resolution inputs:  
e.g.  
640x640





# One way to extracting multiresolution feature maps from Resnet



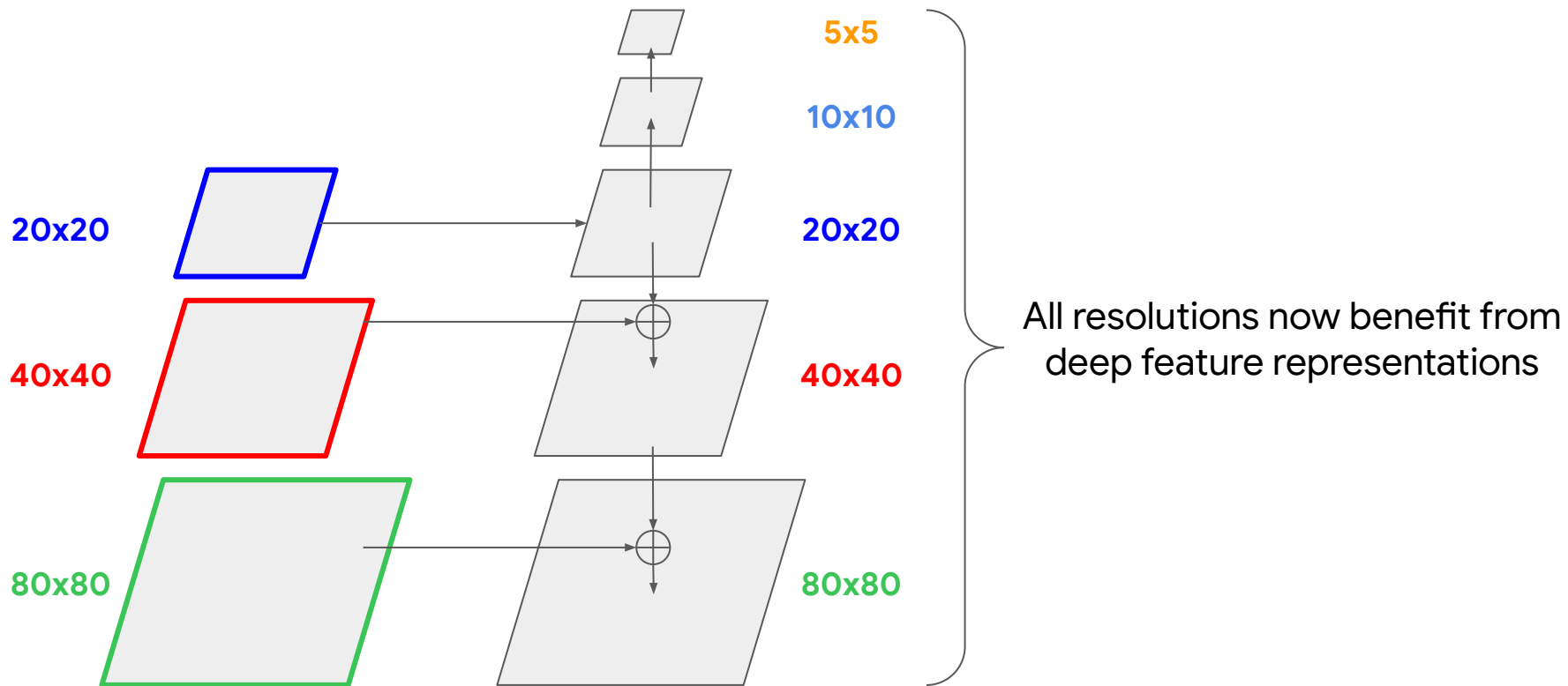
## Tradeoff:

- **Want:** high resolution feature maps to match high density anchor grids capturing small objects.
- **But:** high resolution feature maps live at earlier layers; *can't take advantage of network depth* :(

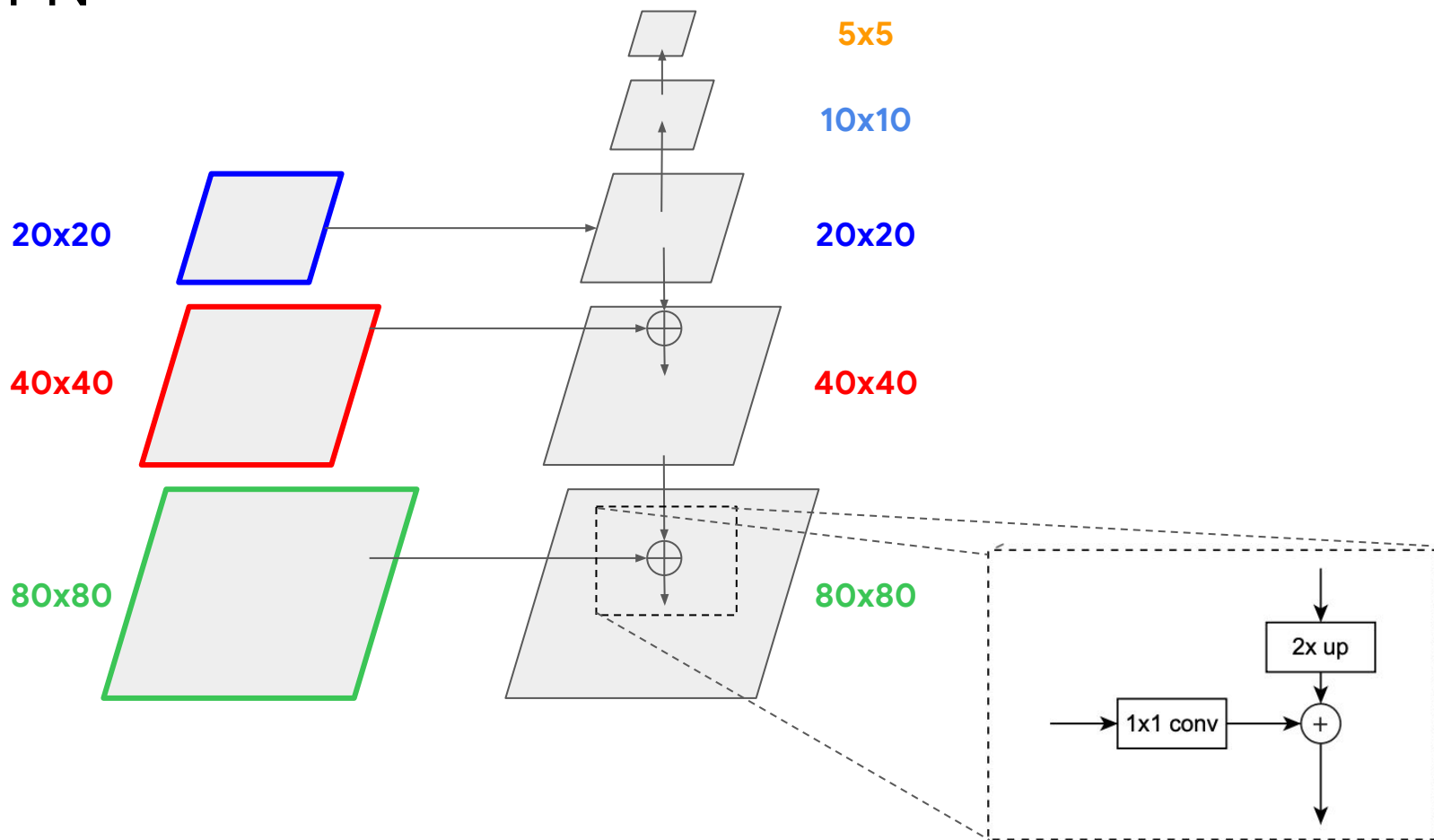
Block 4,  
c=512  
3 residual units



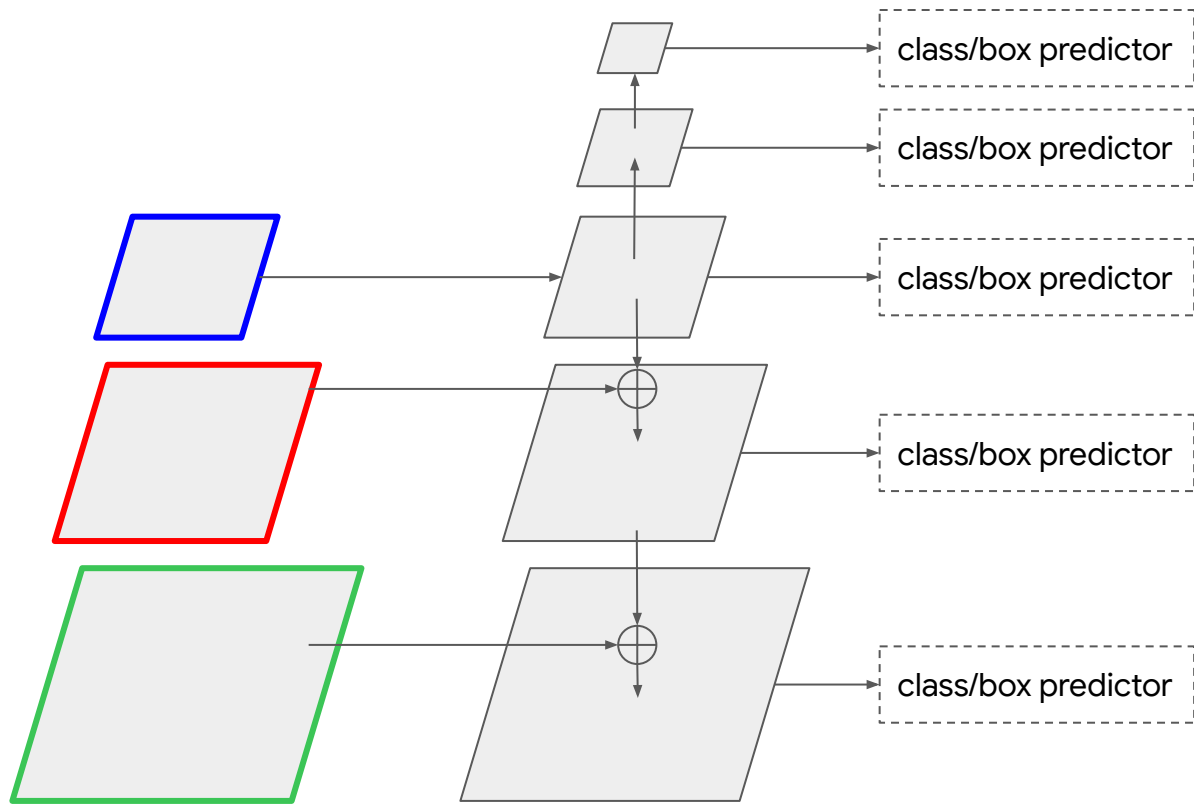
# Enter Feature Pyramid Networks (FPN)



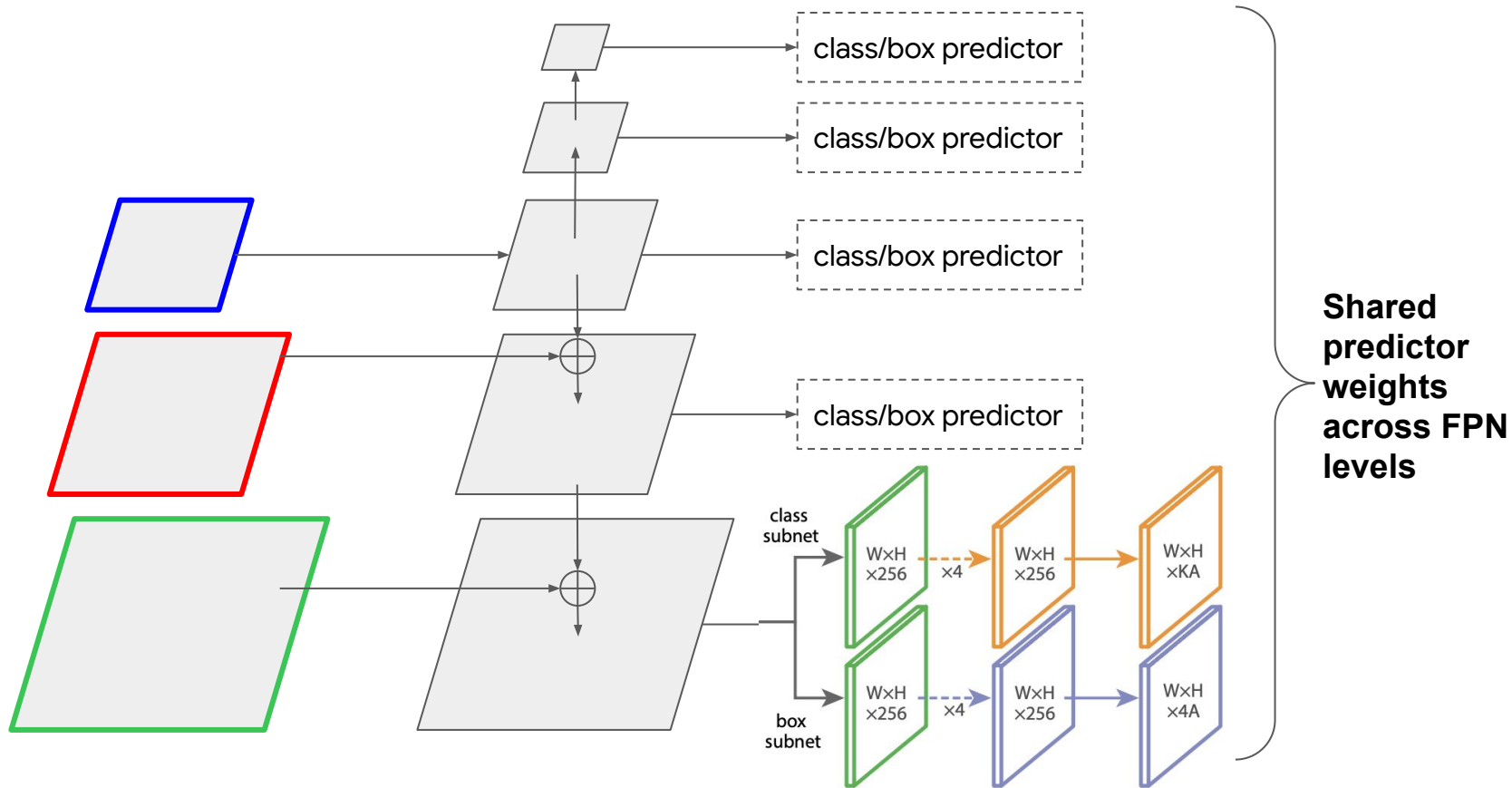
# FPN

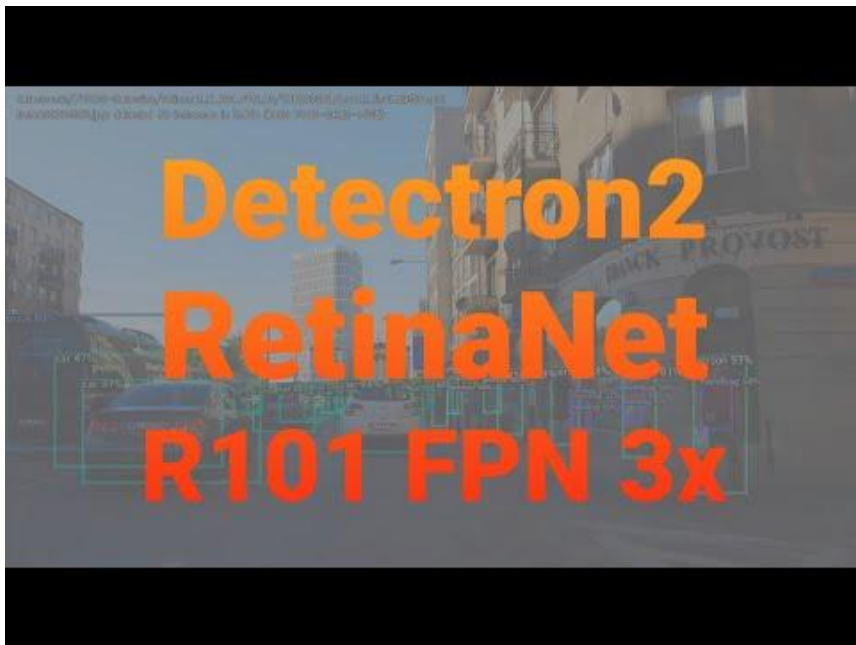


# FPN



# FPN





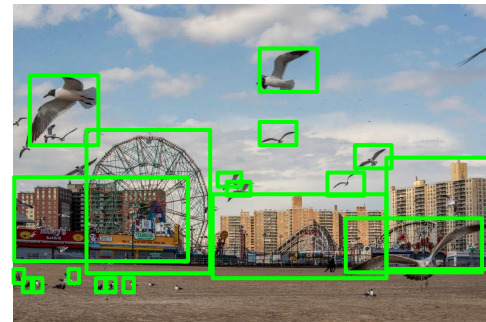
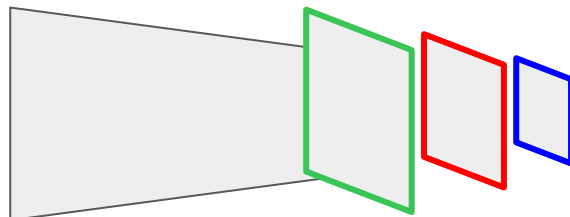
**RetinaNet with Resnet 101 with FPN Feature  
Extractor**



**“RetinaNet” with EfficientNet + biFPN  
Feature Extractor**

# Case Study: Faster R-CNN by Ren et al 2015

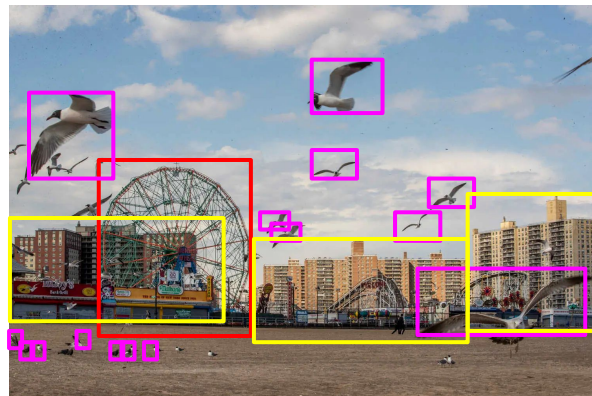
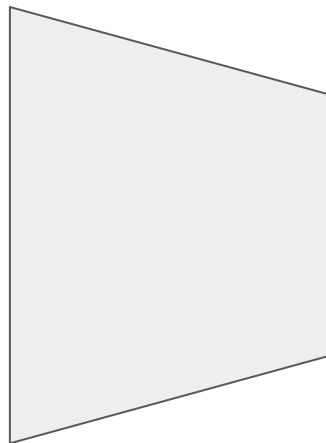
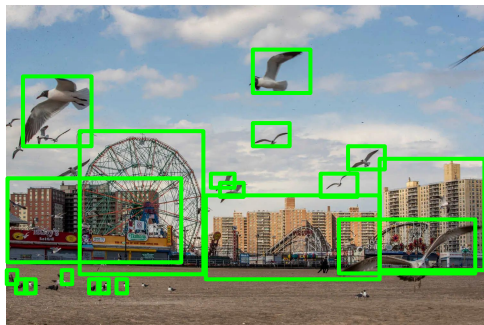
Stage 1: Use a first stage to (over)-predict class agnostic proposals



**Region Proposal Network (RPN)**  
(Think of this as your standard single stage model)

# Case Study: Faster R-CNN by Ren et al 2015

Stage 2: Crop + Resize; Second stage classification



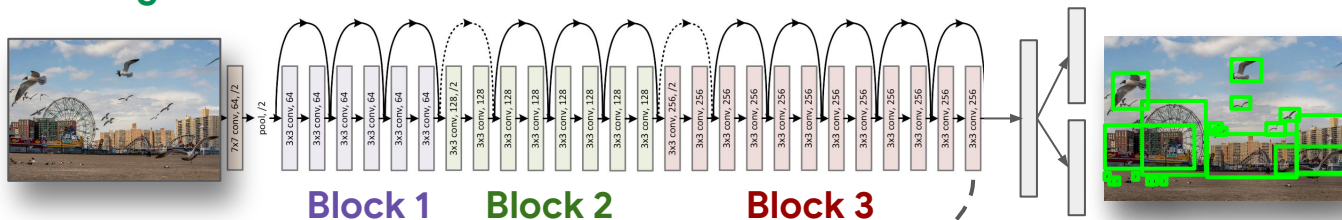
*But crop from RPN features  
instead of pixels!*



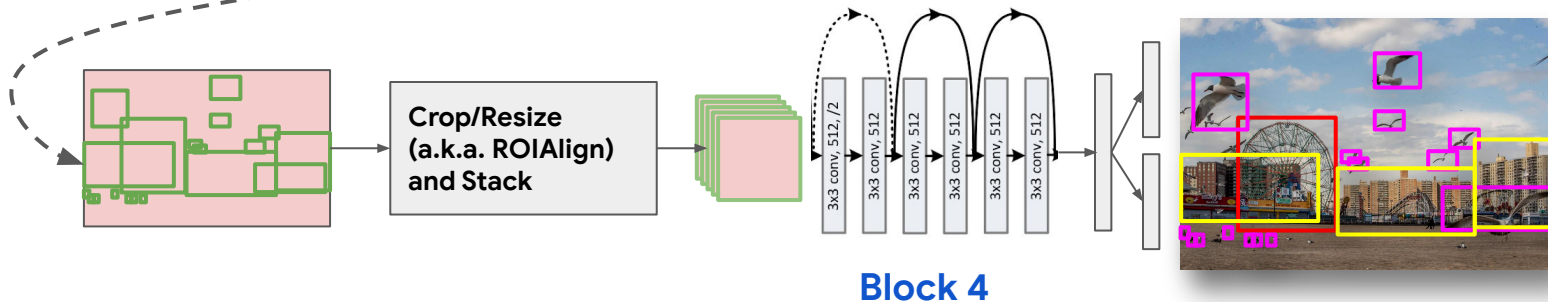
# Faster R-CNN with Resnet (He et al 2015)

*Winning architecture for COCO 2015 Challenge*

## Faster R-CNN Stage 1



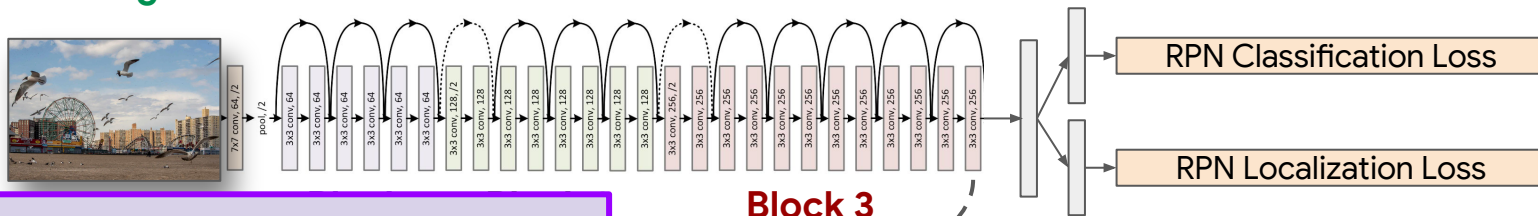
## Faster R-CNN Stage 2



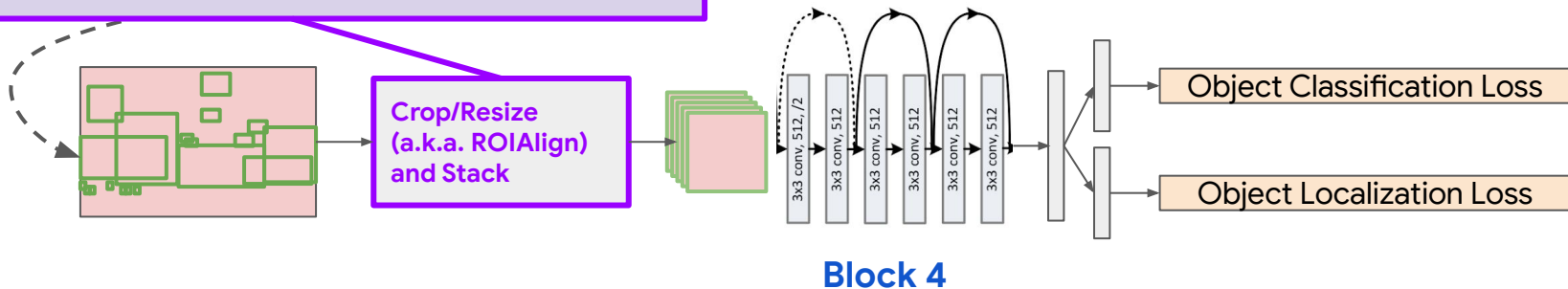
# Faster R-CNN with Resnet (He et al 2015)

Winning architecture for COCO 2015 Challenge

## Faster R-CNN Stage 1



**Differentiable Op!;** Entire model jointly trainable using sum of losses from first and second stage.



## Block 4

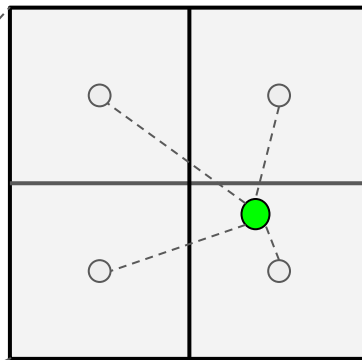
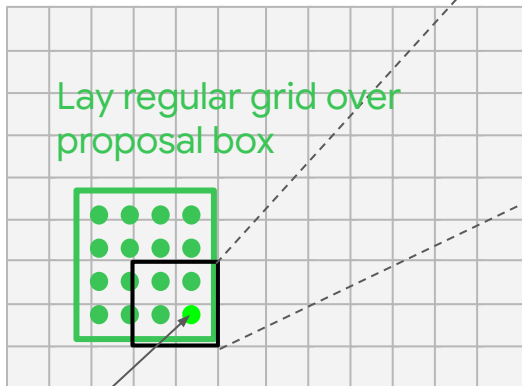
# ROI-Align / Crop-and-resize

PyTorch: torchvision.ops.RoIAlign

TF: tf.image.crop\_and\_resize



Samples typically won't line up nicely with RPN feature map grid

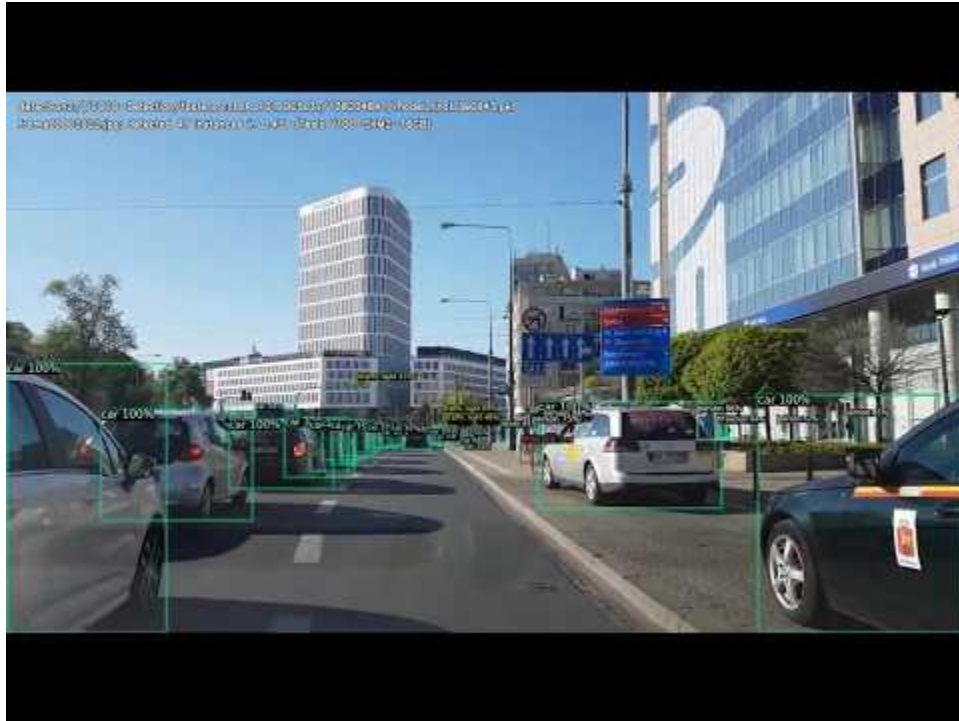


Use bilinear interpolation with 4 neighboring cells to compute feature@sample point

Sum over 4 neighbors

$$f_{x,y} = \sum f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|)$$

Differentiable with respect to RPN feature  $f_{i,j}$



## Faster R-CNN with Resnet 101 Feature Extractor

# One stage vs Two stage Models

- Somewhat-outdated understanding: one stage fast not as accurate, two stage slow, more accurate
- Today the divide is fuzzy
- One stage:
  - Tends to have a “simpler” architecture using only standard ops (Conv, BN, ReLU, Concat);
  - Fussier to “get right”
- Two stage:
  - Require NMS, ROIAlign at training time
  - Yields per-instance feature vectors - easier to stick Faster R-CNN together with other tasks (we will see this later)

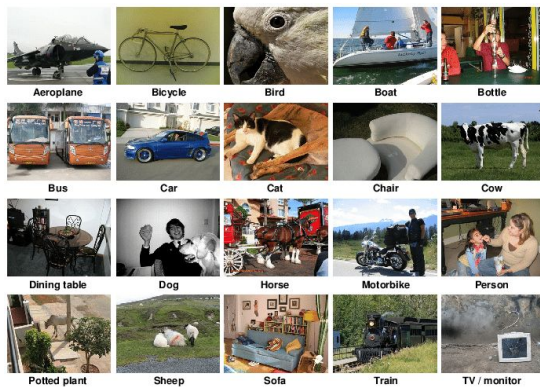
# You should know:

- How to do sliding window detection using ConvNets - aka anchor-based object detection
- Single stage and Two stage “meta-architectures” for detection

# Outline

- Sliding Window Detectors
- Detection with Convolutional Networks
- **How to Evaluate a Detector**
- Practical tips/tricks

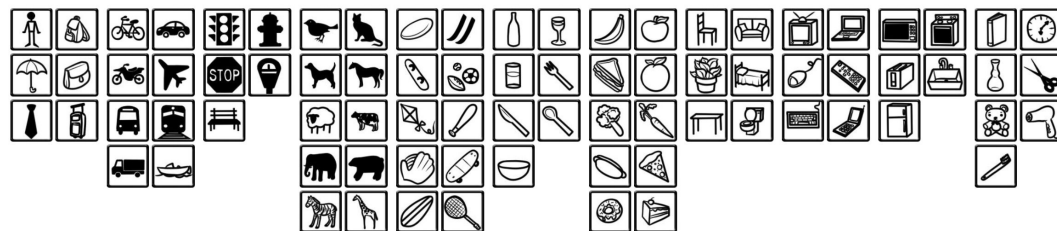
# First you need a dataset...



## Pascal VOC

20 classes, 5K images

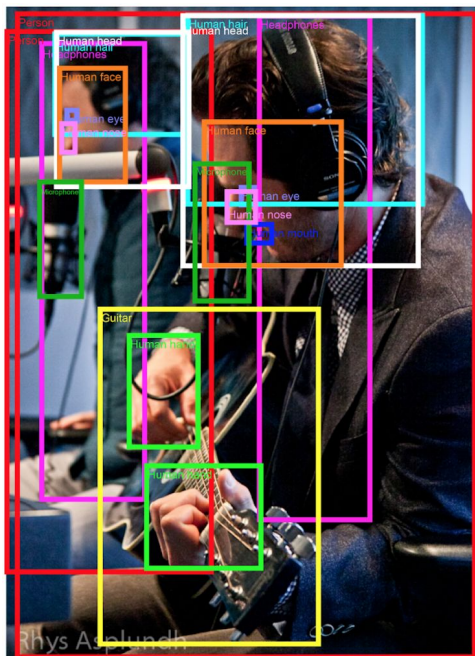
COCO 2017 train/val browser (123,287 images, 886,284 instances). Crowd labels not shown.



## COCO

80 classes, ~120K images



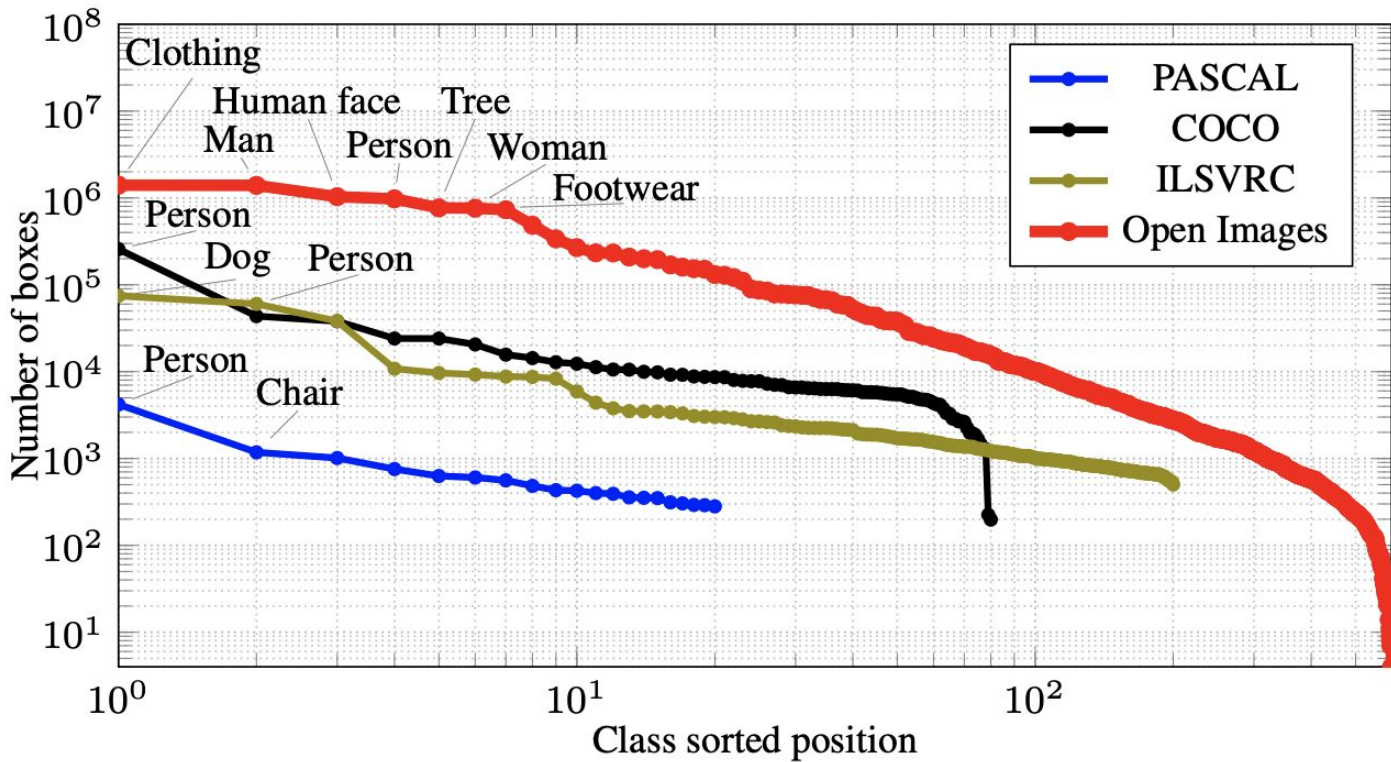


**Open Images**  
(600 classes, 1.7M images)



**LVIS v0.5**  
(1000 classes, 50K images)





# How do we know how good our model is?



↓  
cat ✓



↓  
dog ✓



↓  
cat ✗

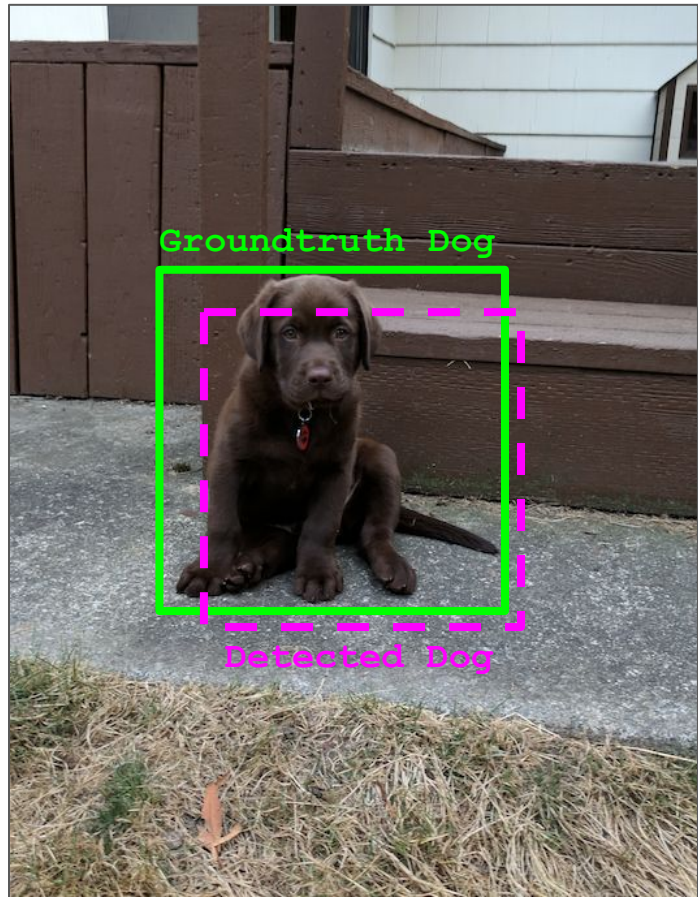


↓  
cat ✓

Accuracy: 75%

For image classification, life is easy :)

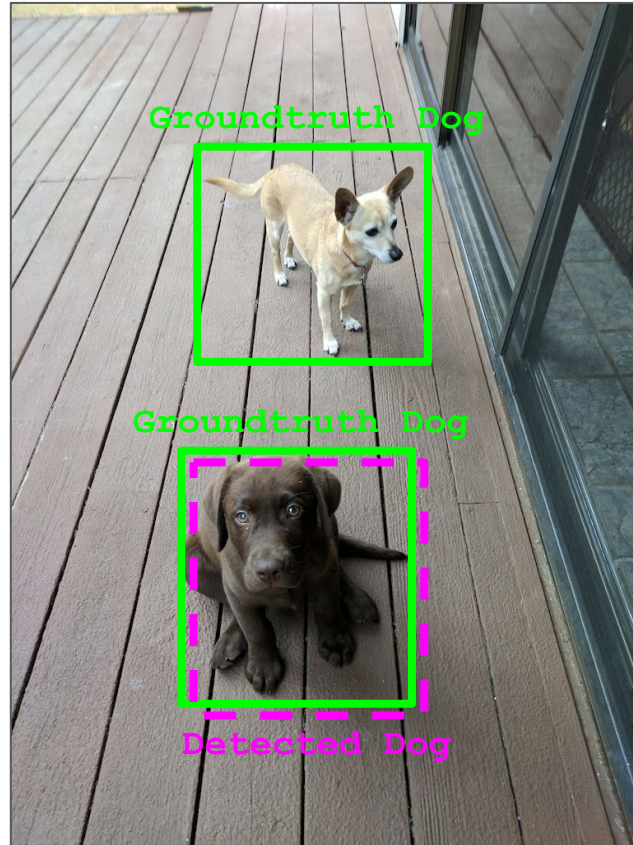
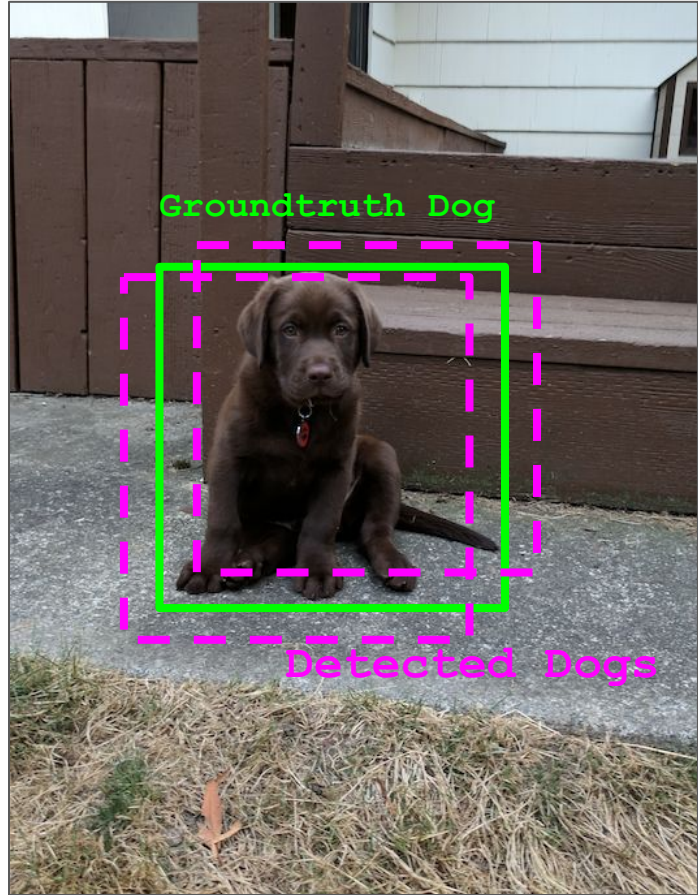
# Evaluating Detectors is harder :(



**Problem 1: Metrics must handle location errors**

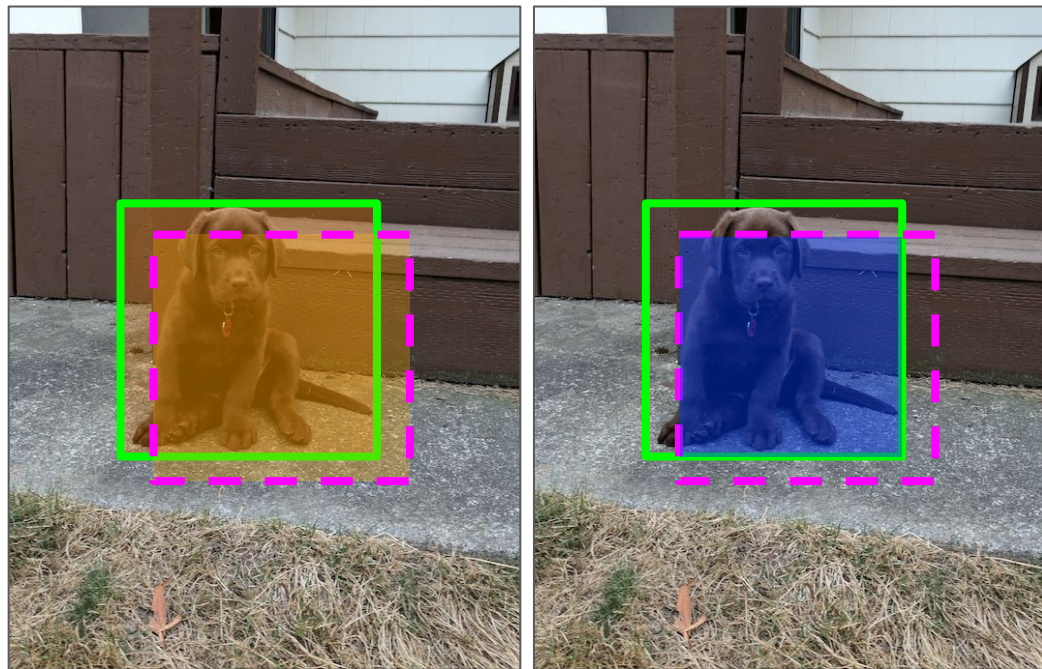
*Should we consider this detection to be correct?*

# Evaluating Detectors is harder :(

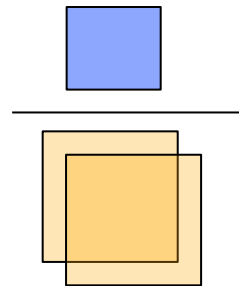


**Problem 2: Metrics must account for overprediction and underprediction**

# Intersection over Union (IOU)

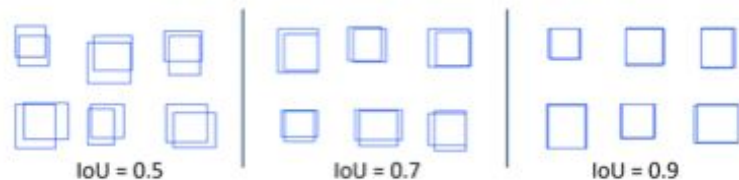


$$\text{IOU} = \frac{\text{Intersection}}{\text{Union}}$$



- Boxes are disjoint if and only if  $\text{IOU}=0$
- Boxes are identical if and only if  $\text{IOU}=1$

Detection is considered “correct” if  $\text{IOU} > \alpha$

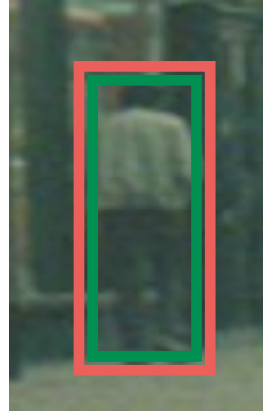


# Intersection over Union (IoU)

IoU = 0.5



IoU = 0.7



IoU = 0.95

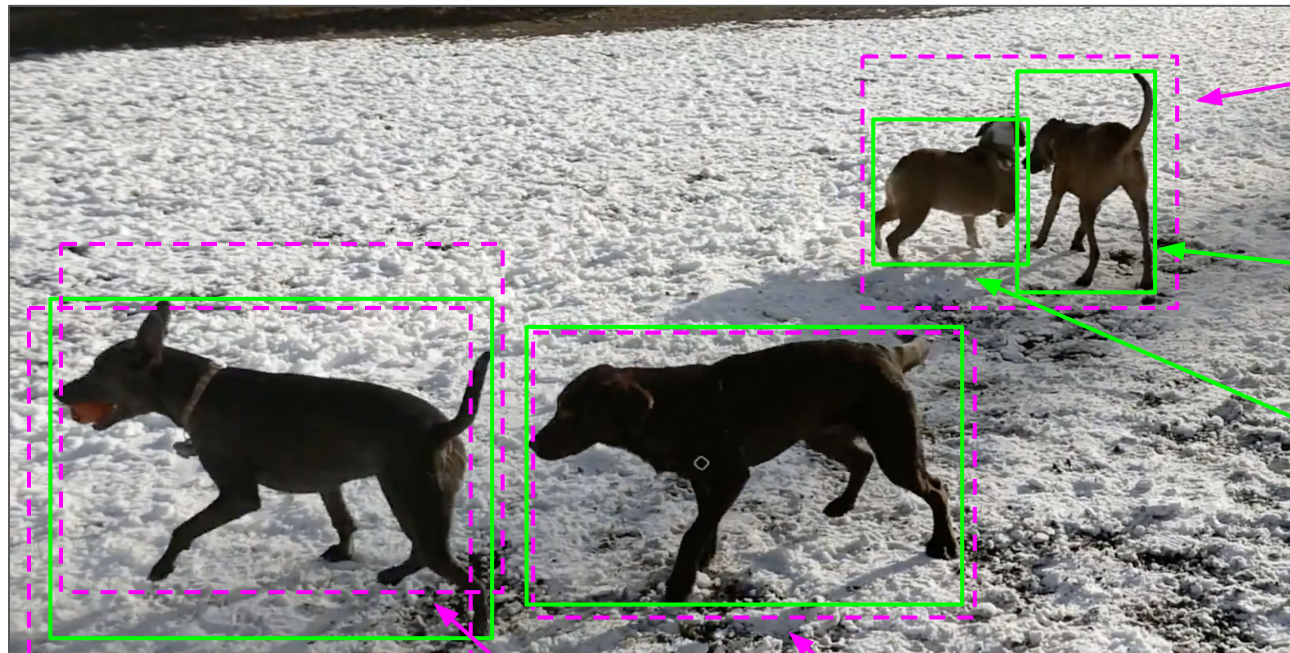


Ground-Truth BBox



Detection BBox

# True/False Positives and Missed Objects



True positive ✓

False positive ✗

True positive ✓

False positive ✗

Missed Object ✗

Missed Object ✗

- Match detections and groundtruth instances based on IOU
- Count missed groundtruth objects
- Mark detections as TP or FP based on whether  $\text{IOU} > \alpha$



## Summarizing Performance with Precision/Recall

**Precision:** Of the detections our model produced, how many were correct (i.e. True Positives)?

$$\text{Precision} = \frac{\#TP}{\#TP + \#FP}$$

**Recall:** Of the groundtruth instances in our data, what fraction of instances were correctly detected (i.e., not missed)?

$$\text{Recall} = \frac{\#TP}{\#Groundtruth\ Objects}$$

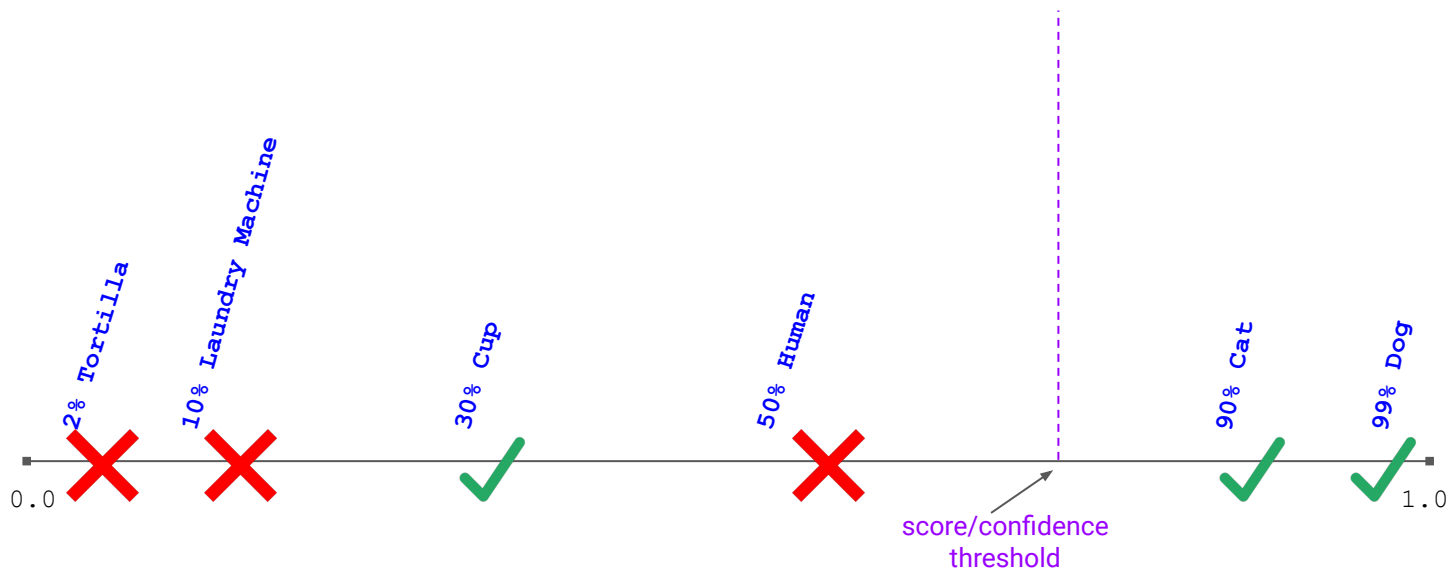


*Remember: Precision and Recall are in [0, 1] and higher is better.*

# Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

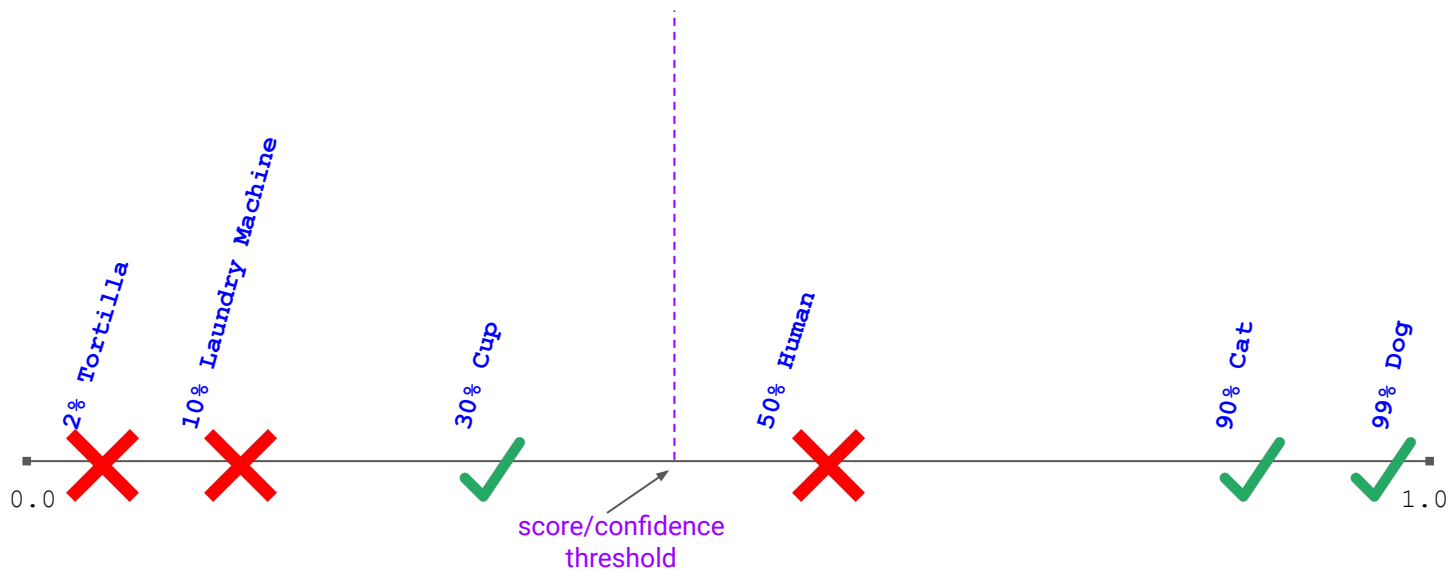
Last step of detection pipeline: *use score threshold to select final detections*



# Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

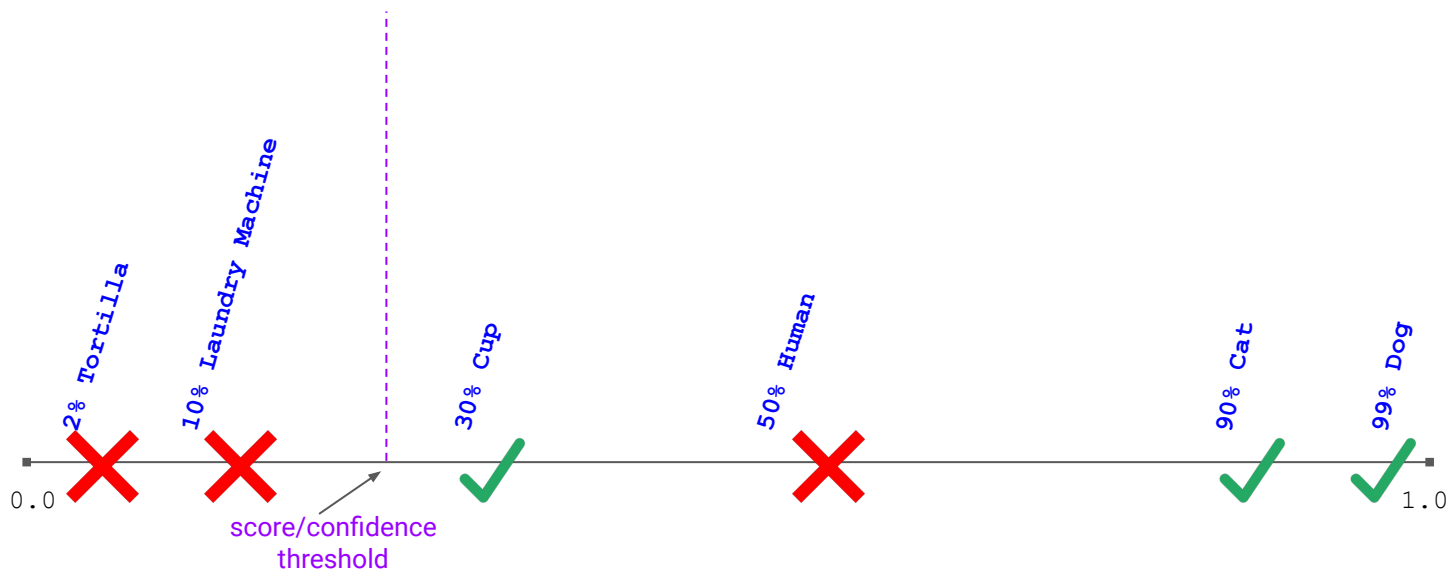
Last step of detection pipeline: *use score threshold to select final detections*



# Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

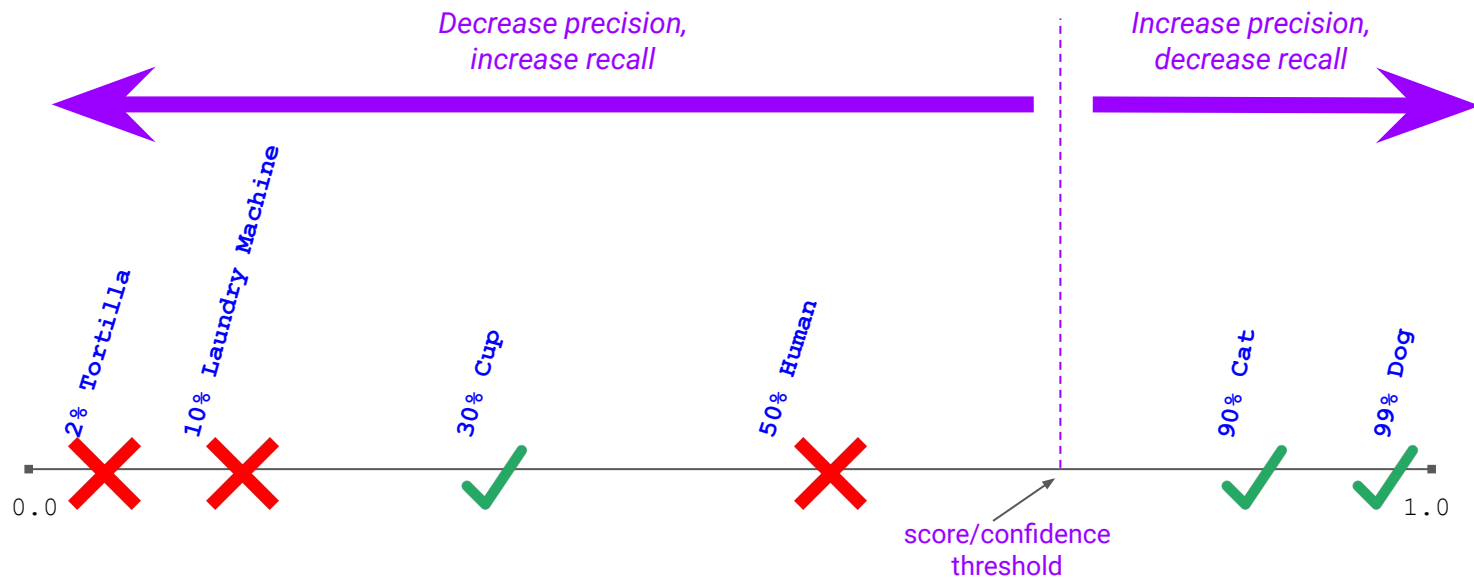
Last step of detection pipeline: *use score threshold to select final detections*



# Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

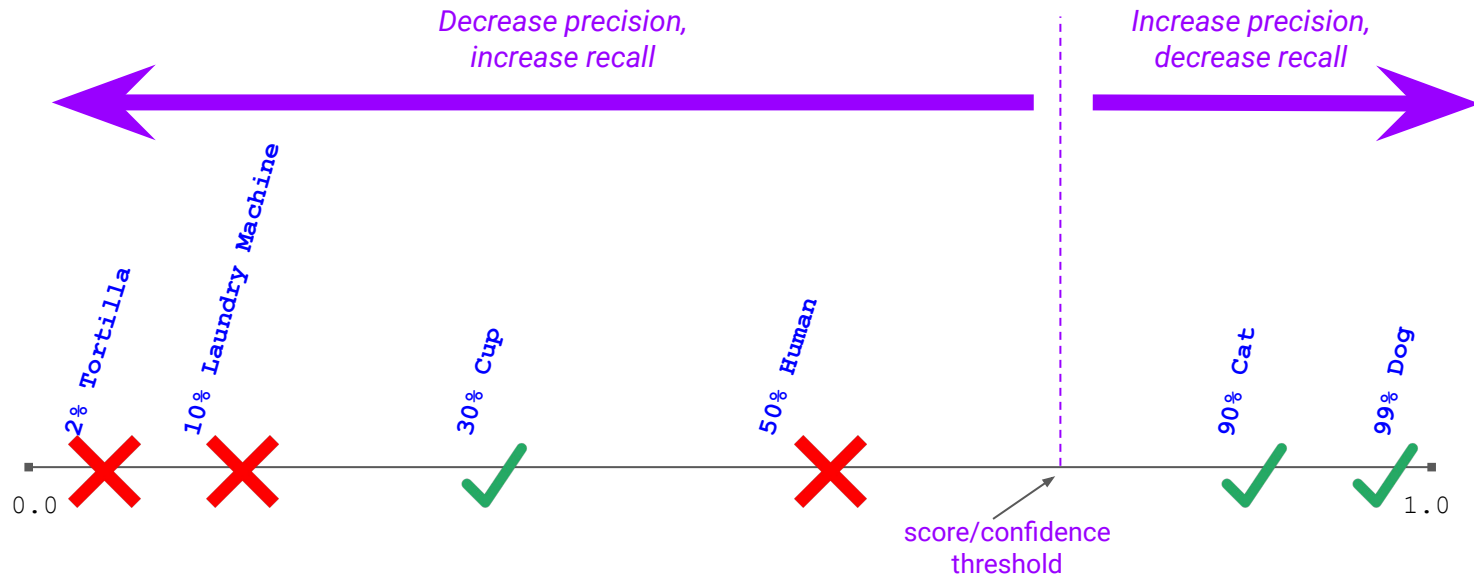
Last step of detection pipeline: *use score threshold to select final detections*



# Trading off between Precision and Recall

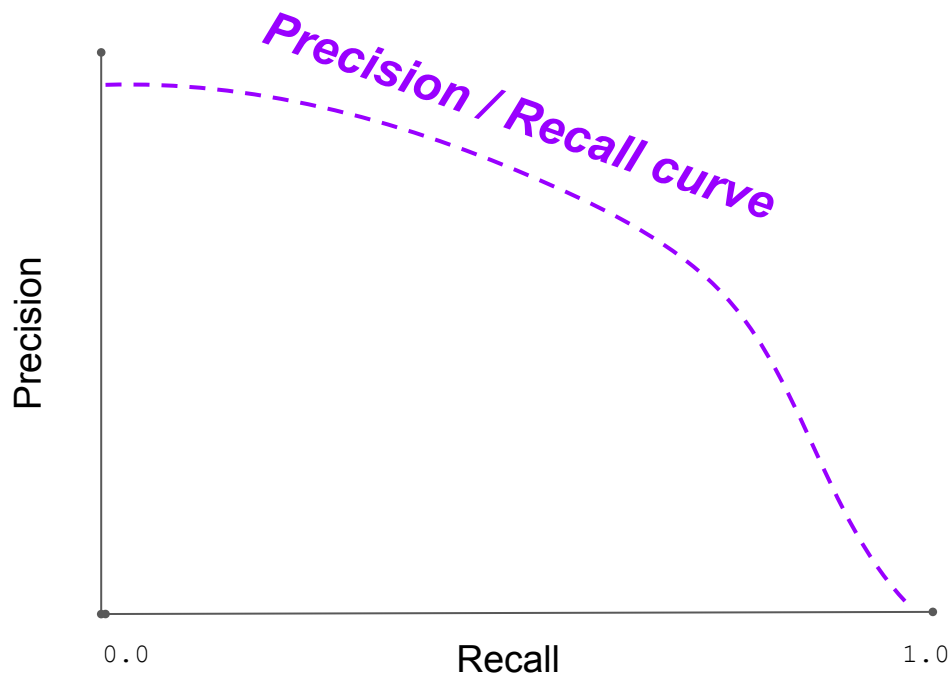
Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

Last step of detection pipeline: *use score threshold to select final detections*

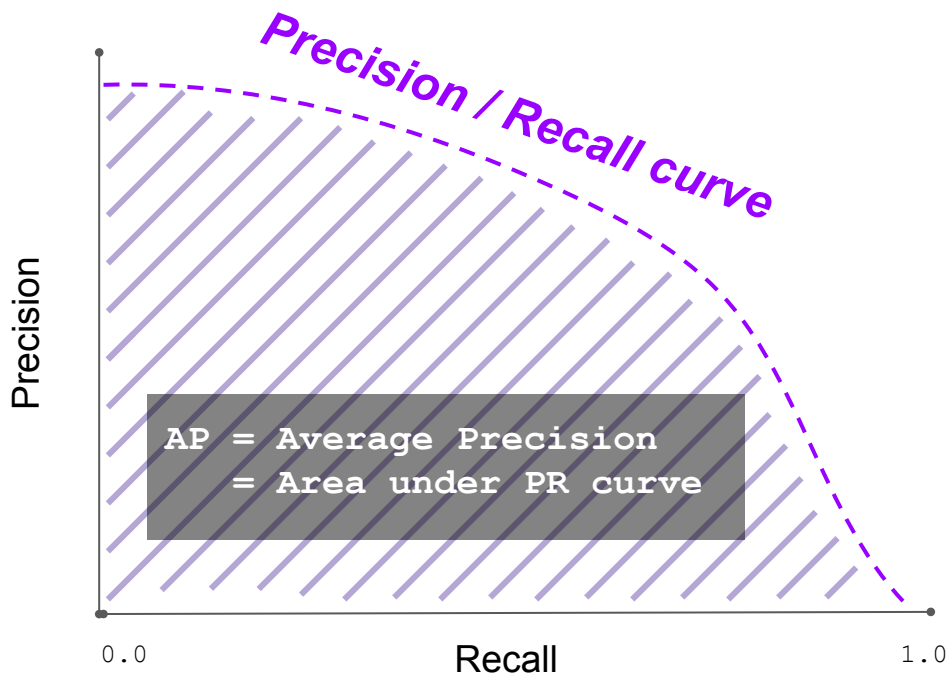


*When would it be better to be on one side of this spectrum than the other?*

# Precision/Recall Curves and AP (Average Precision)



# Precision/Recall Curves and AP (Average Precision)



Remember:



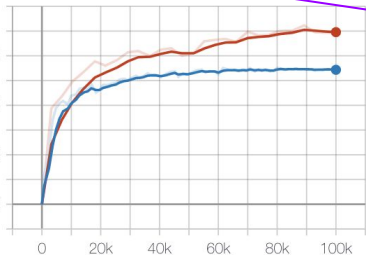
- *AP is always in  $[0, 1]$*
- *Higher AP is better*
- *Always relative to an IOU criterion, e.g., AP@.5 IOU, AP@.75 IOU, etc...*



# AP, mAP, “COCO (Integrated) mAP”

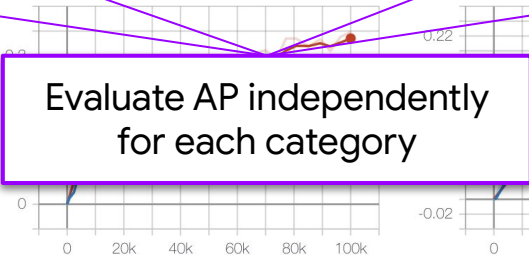
mAP/airplane

tag: DetectionBoxes\_PerformanceByCategory/mAP/airplane



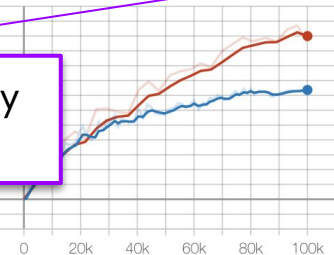
mAP/apple

tag: DetectionBoxes\_PerformanceByCategory/mAP/apple



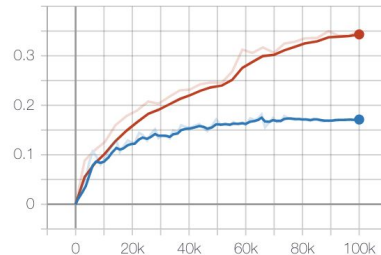
mAP/backpack

tag: DetectionBoxes\_PerformanceByCategory/mAP/backpack



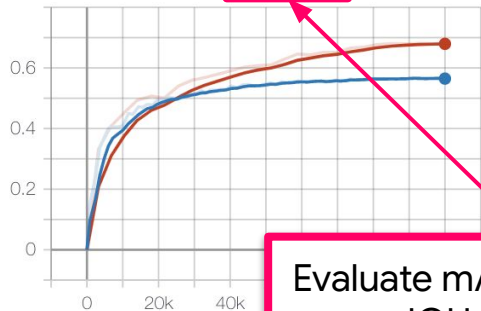
mAP/banana

tag: DetectionBoxes\_PerformanceByCategory/mAP/banana



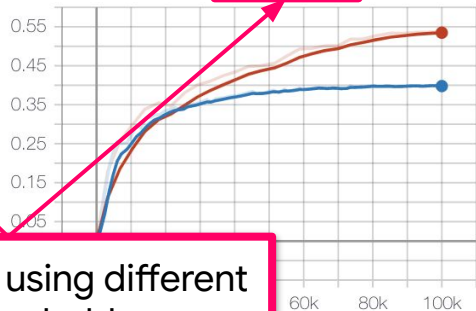
mAP@.50IOU

tag: DetectionBoxes\_Precision/mAP@.50IOU



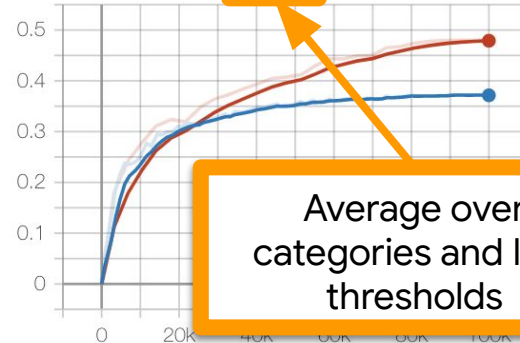
mAP@.75IOU

tag: DetectionBoxes\_Precision/mAP@.75IOU

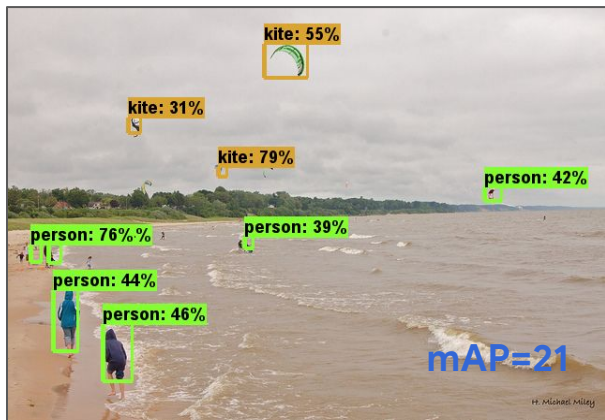


mAP

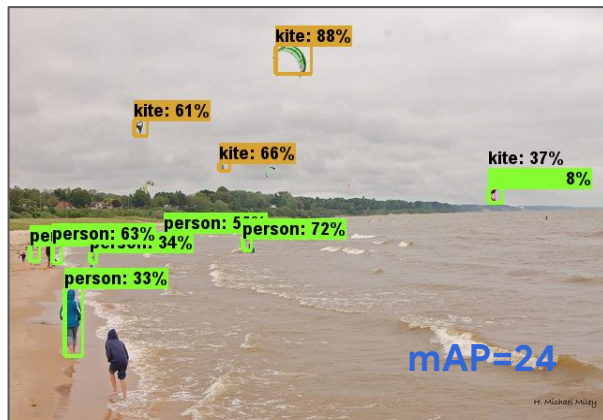
tag: DetectionBoxes\_Precision/mAP



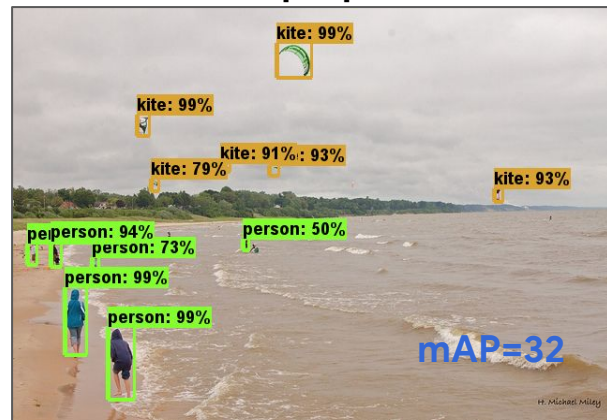
SSD w/MobileNet  
(Low Resolution)



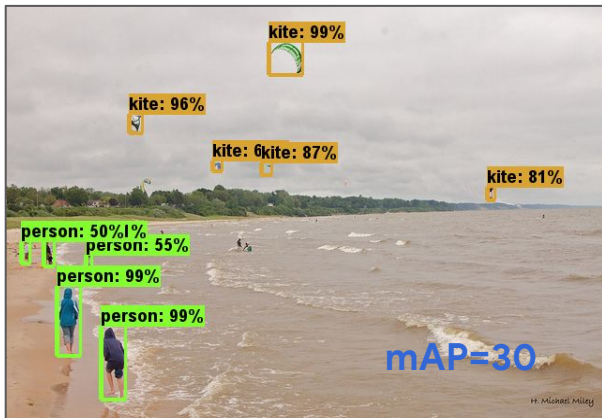
SSD w/Inception V2  
(Low Resolution)



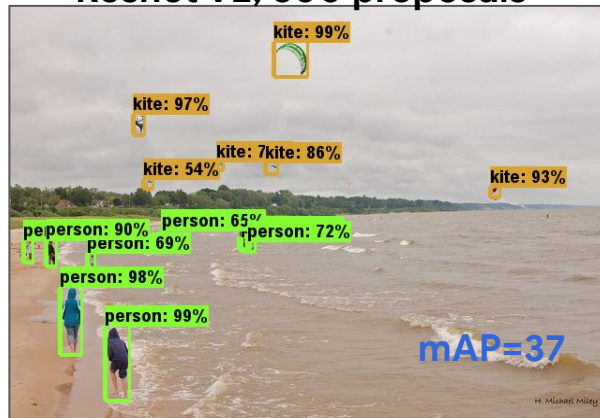
Faster R-CNN w/Resnet101,  
100 proposals



RFCN w/Resnet101, 300



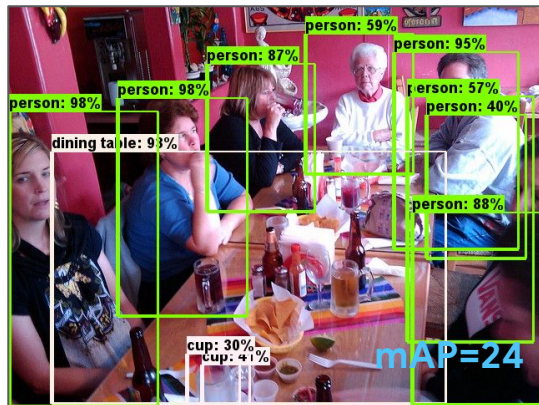
Faster R-CNN w/Inception  
Resnet V2, 300 proposals



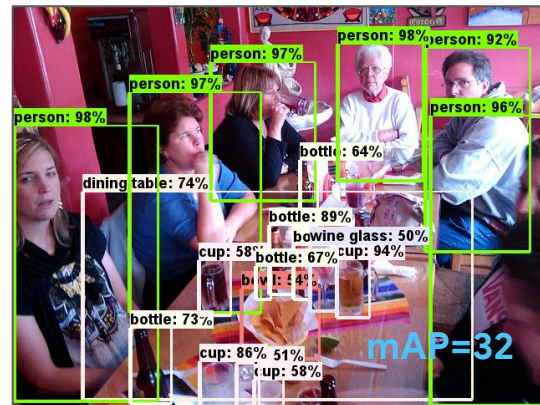
### SSD w/MobileNet (Low Resolution)



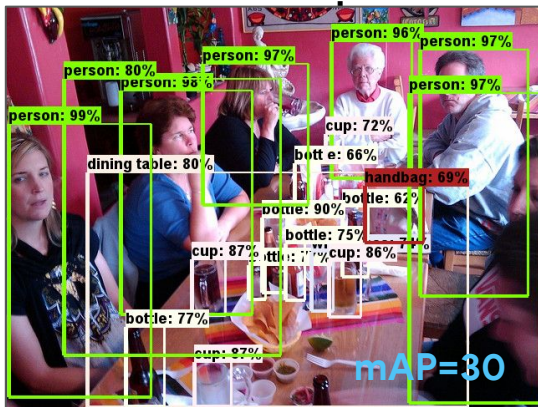
### SSD w/Inception V2 (Low Resolution)



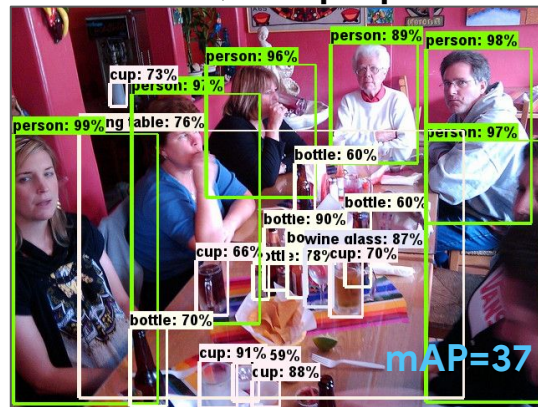
### Faster R-CNN w/Resnet101, AI 100 proposals



### RFCN w/Resnet101, 300



### Faster R-CNN w/Inception Resnet V2, 300 proposals



# You should know:

- How to mark detections as True or False positives based on IOU
- What *Precision* and *Recall* mean
- And have some vague idea about how P-R Curves and Average Precision are computed :)

# Outline

- Sliding Window Detectors
- Detection with Convolutional Networks
- How to Evaluate a Detector
- **Practical tips/tricks**

# How to select a model

## Decisions:

- Which meta-architecture?     {SSD, Faster R-CNN, R-FCN}
- Which feature extractor?     {VGG, Resnet, Inception v2, Inception v3, Mobilenet, etc}
- Which image size?             {300x300, 512x512, 600x1024, 800x1296}

**Things to consider:** sensor, device, latency constraints, memory constraints



Your laptop



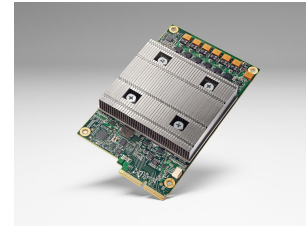
Datacenters



Mobile

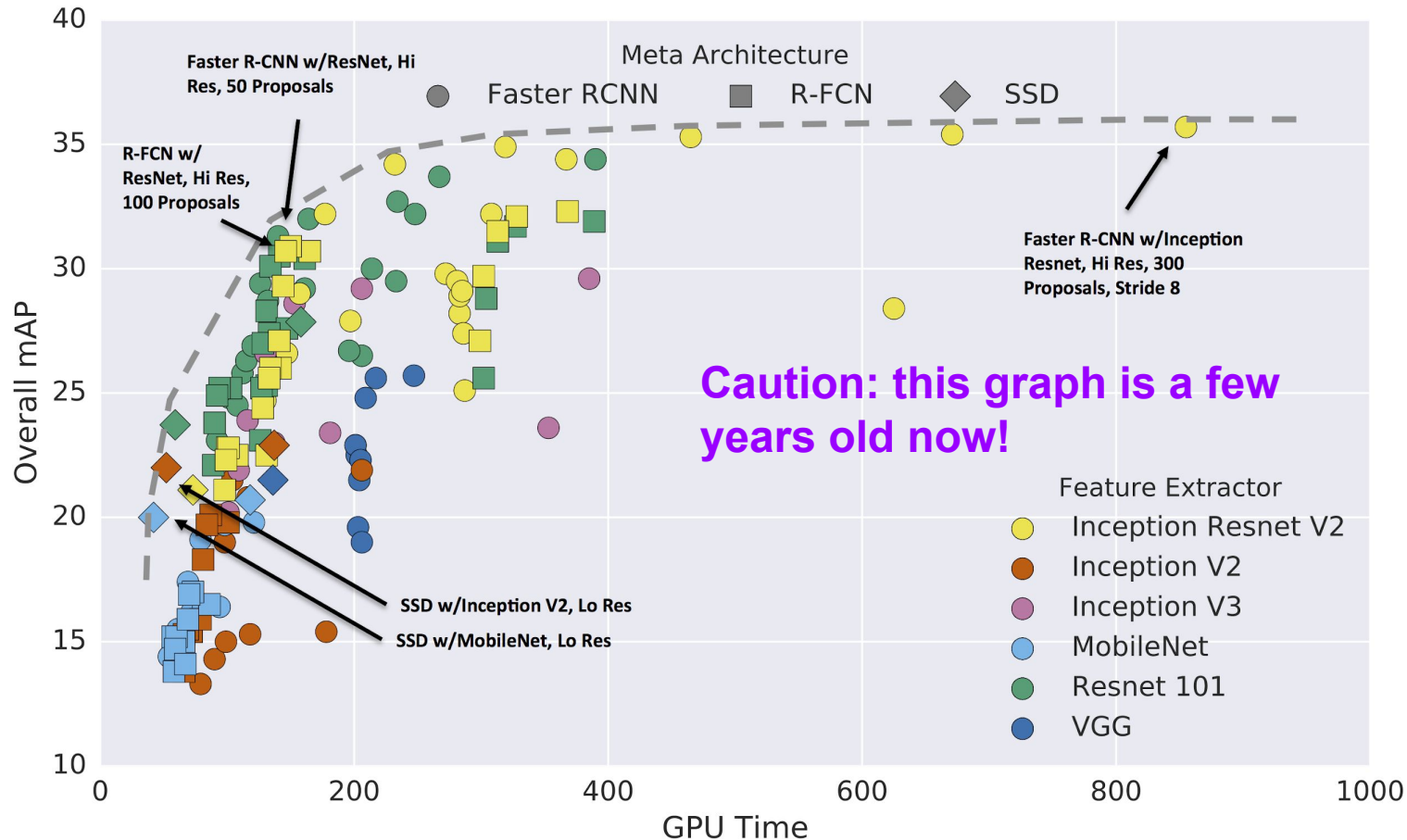


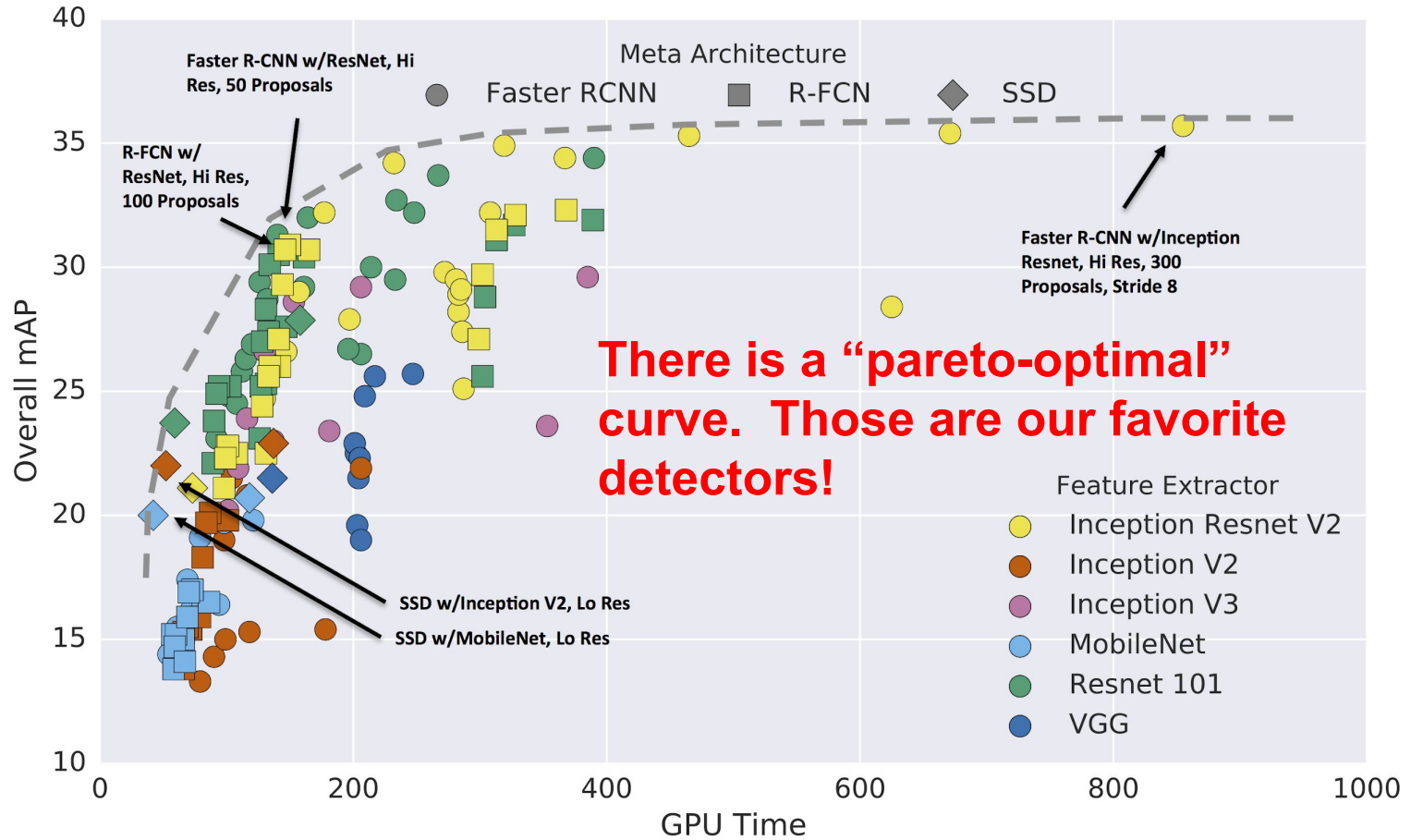
Raspberry Pi



Tensor Processing Unit

# Pick a point on the speed/accuracy tradeoff curve

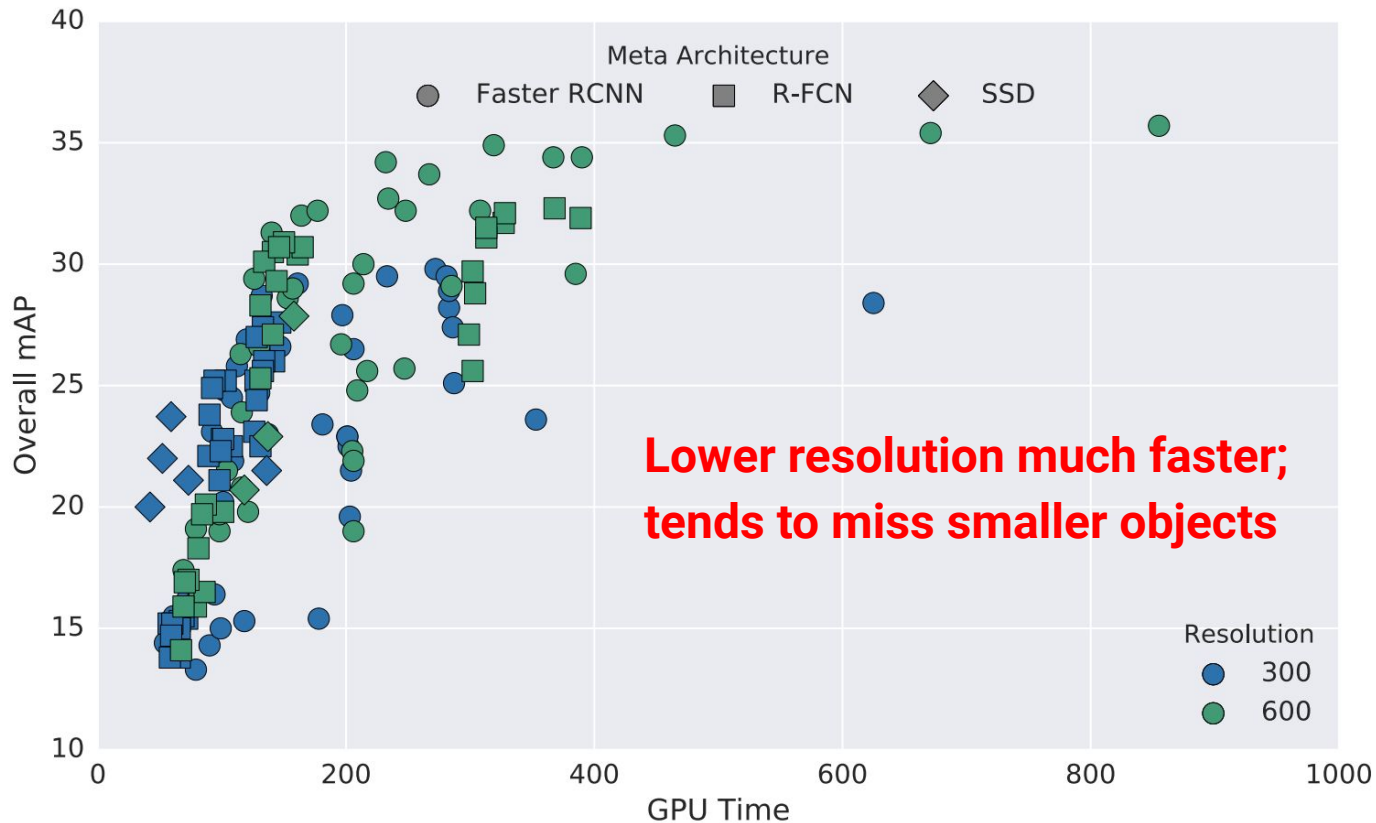




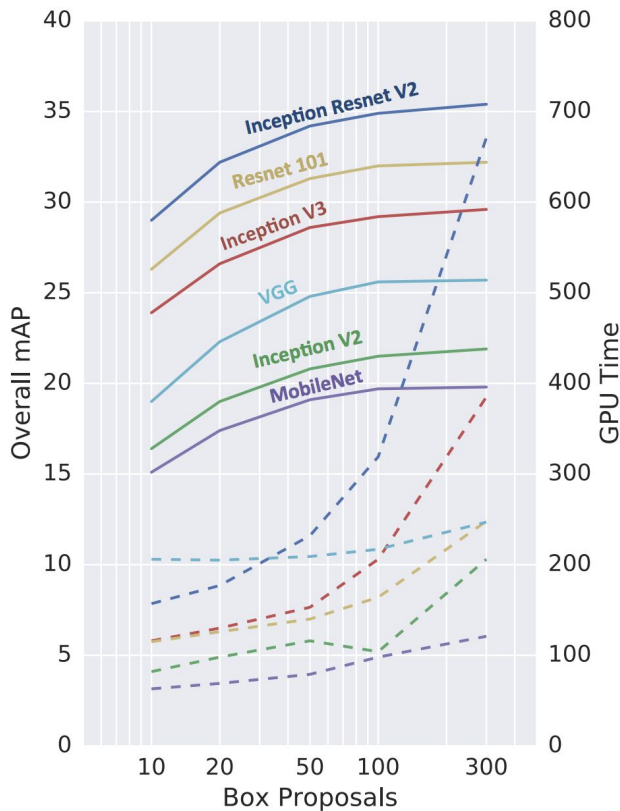




# Use lower resolution images for speed



# Use a small number of proposals for speed (for proposal based architectures)

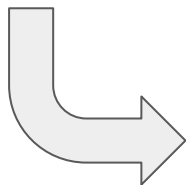


**Lower # of proposals much faster;  
sacrifices a bit of recall**

# Training with High Resolution Images

Dataset	Typical Training Resolutions
MNIST	28x28
CIFAR	32x32
ImageNet	112x112, 224x224, 299x299
COCO	640x640, 600x1024, 1024x1024, 800x1333,

} Often 10x more pixels!



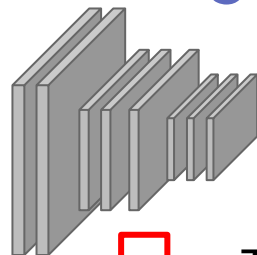
Larger images => Smaller batch sizes => Noisy batch norm statistics :(

## Common approaches:

- Freeze batch norm
- Use batch norm variant (e.g. GroupNorm)
- Train with multi GPU/TPU (even better, use Sync BN)

# Initialize from a model pre-trained to classify some other dataset (the larger the better)

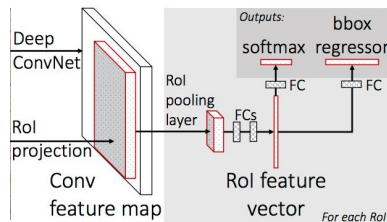
JFT 300M



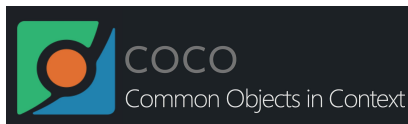
18K labels



Transfer weights



Detections



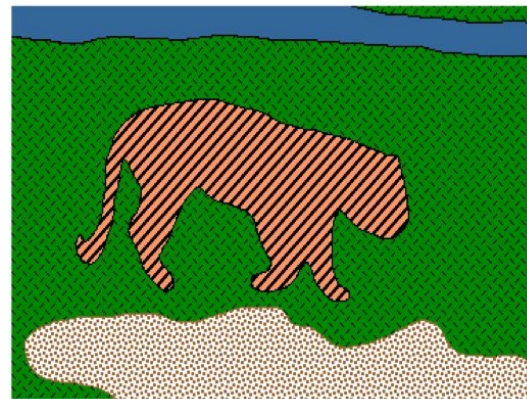
Method	mAP@0.5	mAP@[0.5,0.95]
He <i>et al.</i> [16]	53.3	32.2
ImageNet	53.6	34.3
300M	56.9	36.7
ImageNet+300M	<b>58.0</b>	<b>37.4</b>
Inception ResNet [37]	56.3	35.5

See “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era” [Sun et al 2017]

# You should know:

- Anchor based object detection methodology
- Examples of single stage and two stage models
- Evaluation concepts (IOU, Precision, Recall, mean AP)
- Practical Tips

# Next Time



## Segmentation

- Semantic Segmentation
- Dense Prediction: general
- Instance and Panoptic Segmentation
- Keypoint Estimation
- Object Detection II: Anchor free approaches