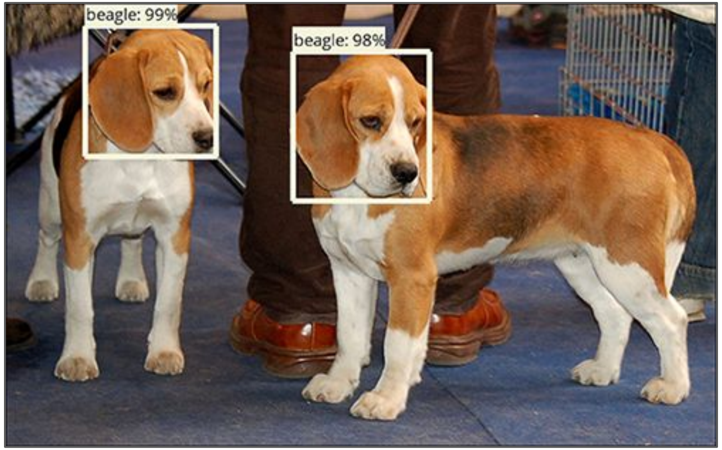
CSEP 576: Object Detection



Jonathan Huang (jonathanhuang@google.com)

University of Washington 17 May 2020

Google Research

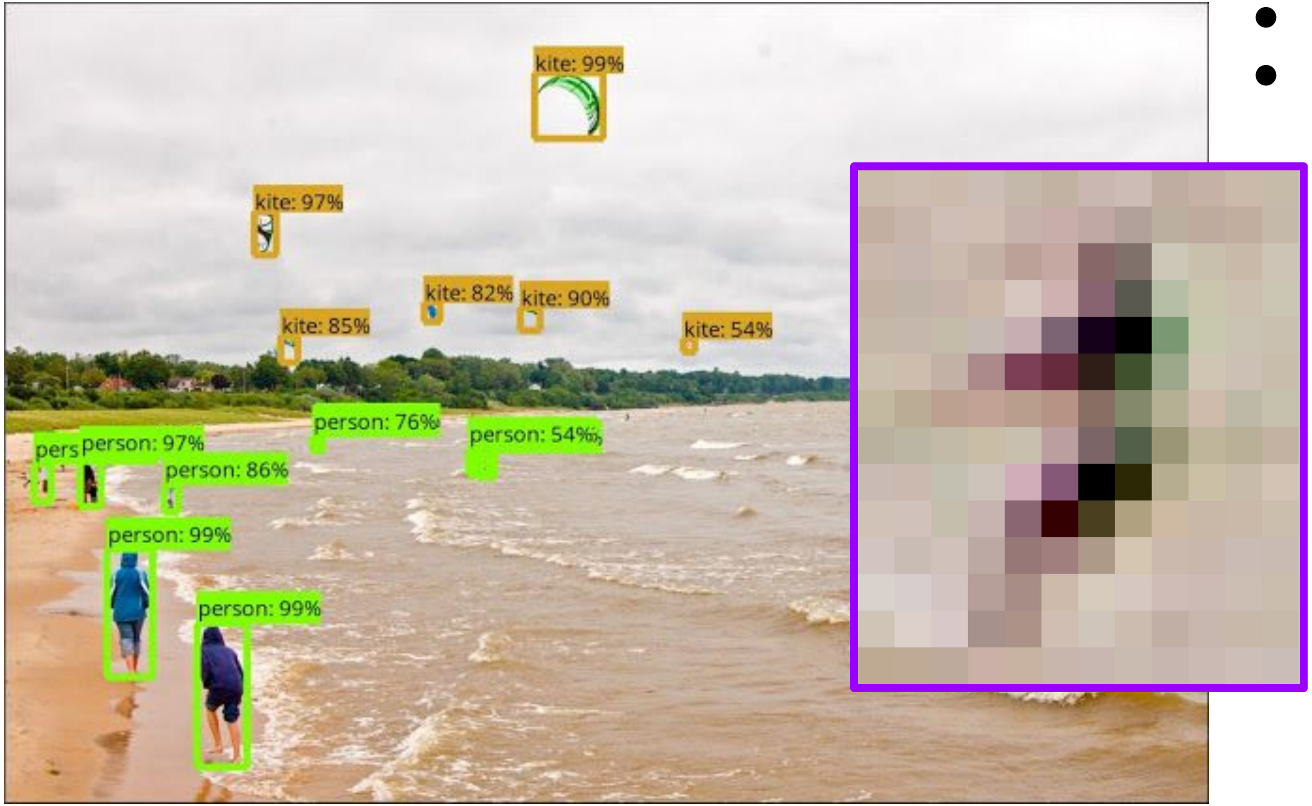
Lecture Outline

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

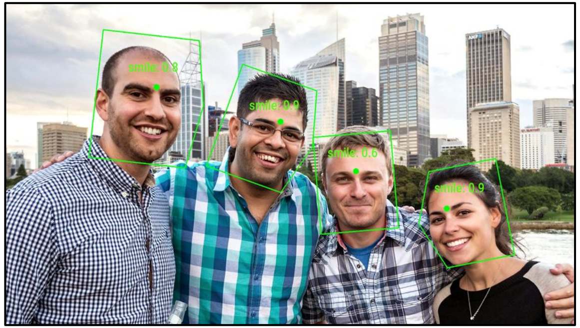
Detection = Classification + Localization

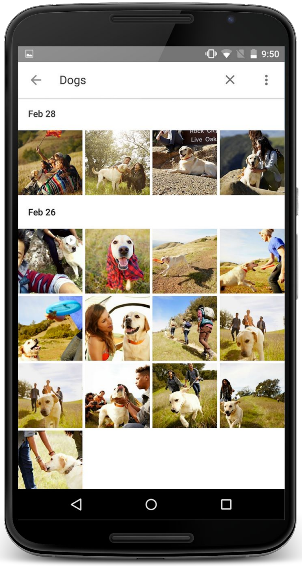


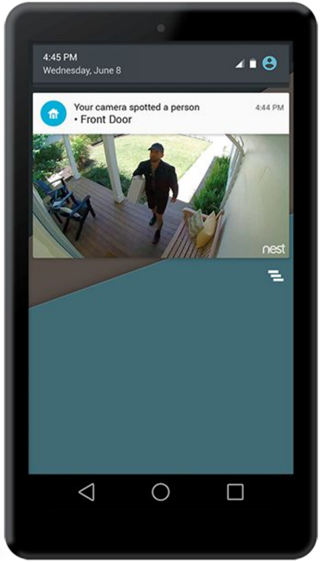
- Variable # outputs
- *Need to classify based on much fewer pixels than in Imagenet setting*; Requires context! Usually need to operate on much larger images

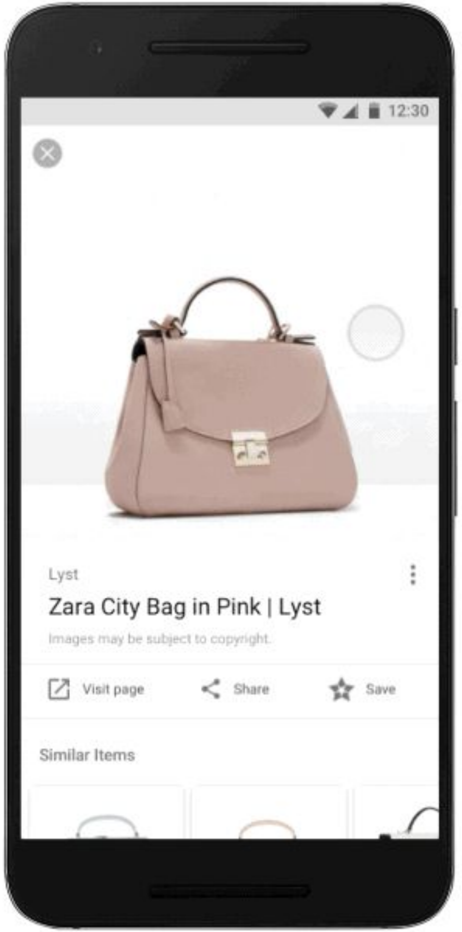
Photo credit: Michael Mina

Object Detection Applications

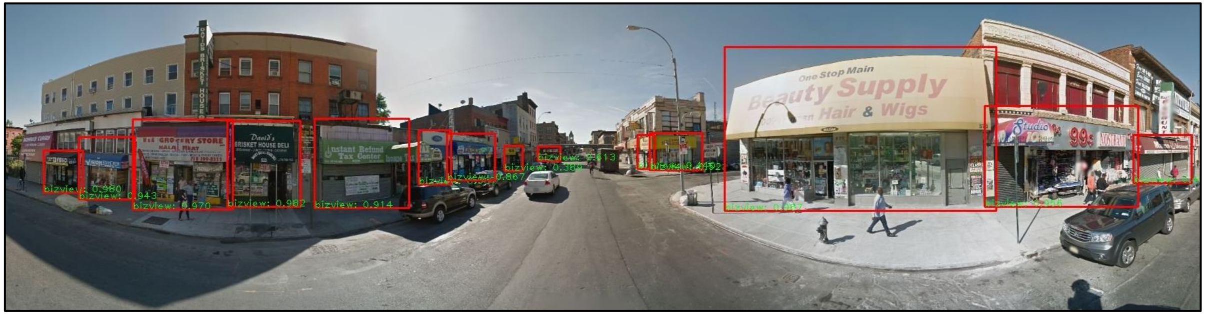








Object Detection Applications



Object Detection Applications

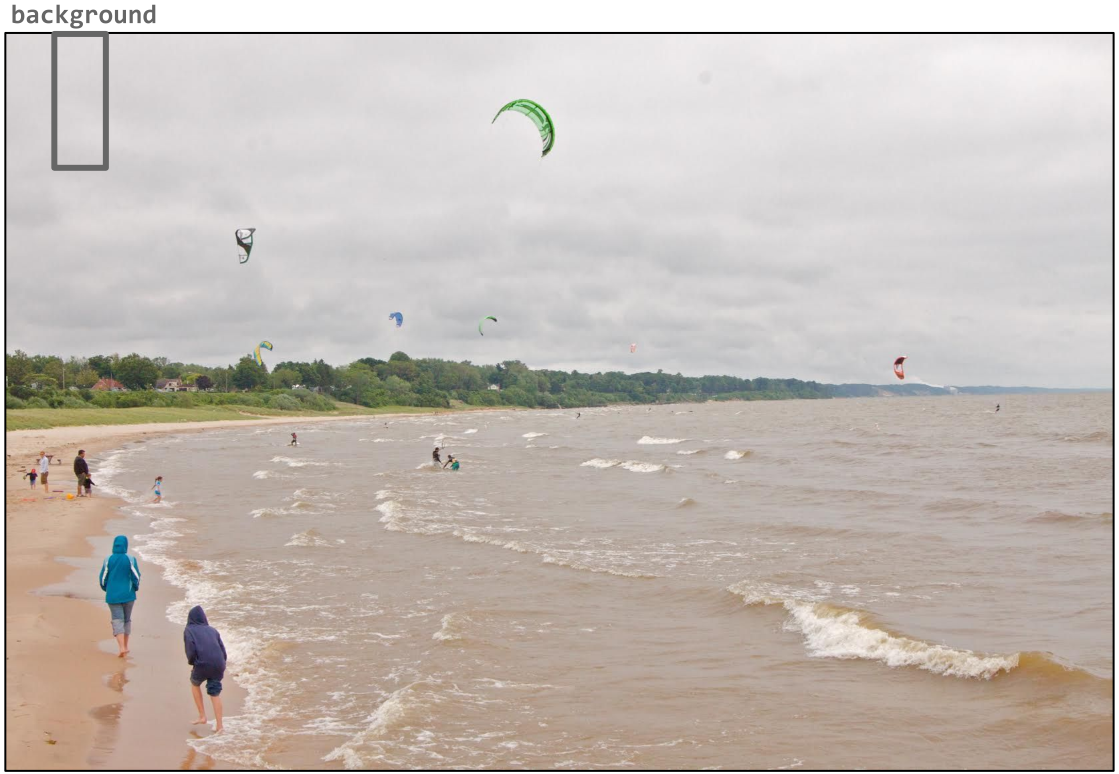
Object Detection Applications

Object Detection Applications

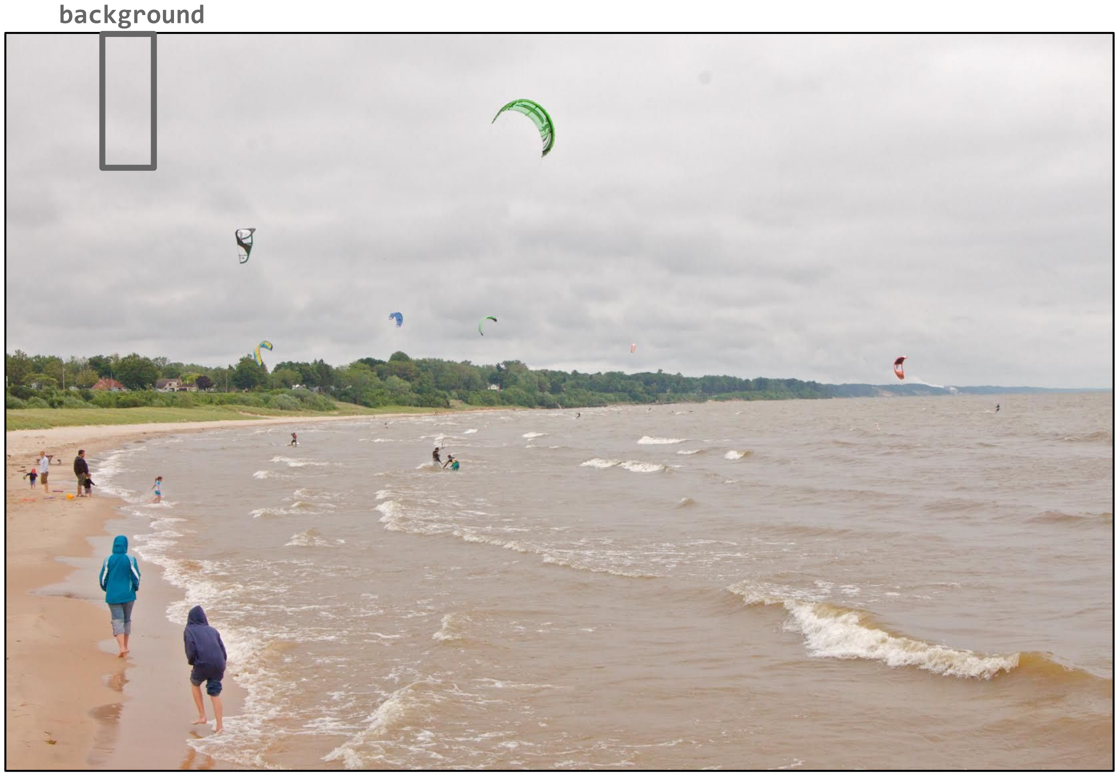
“Sliding Window” Detection

background

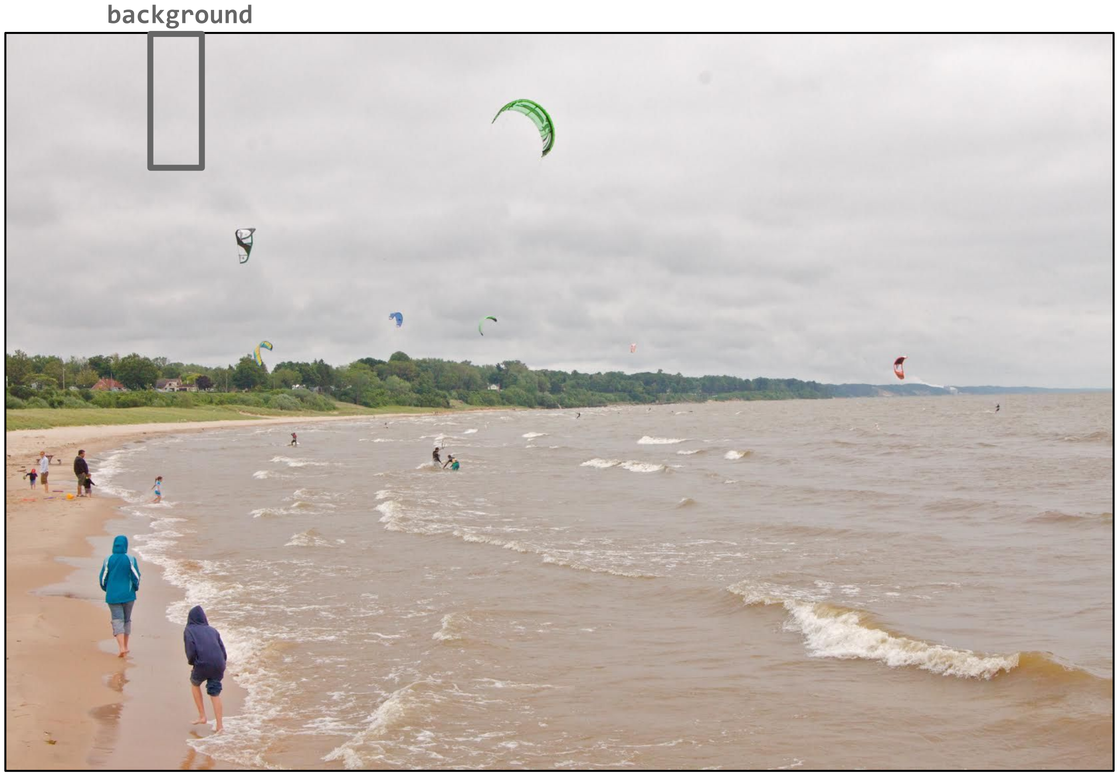
“Sliding Window” Detection



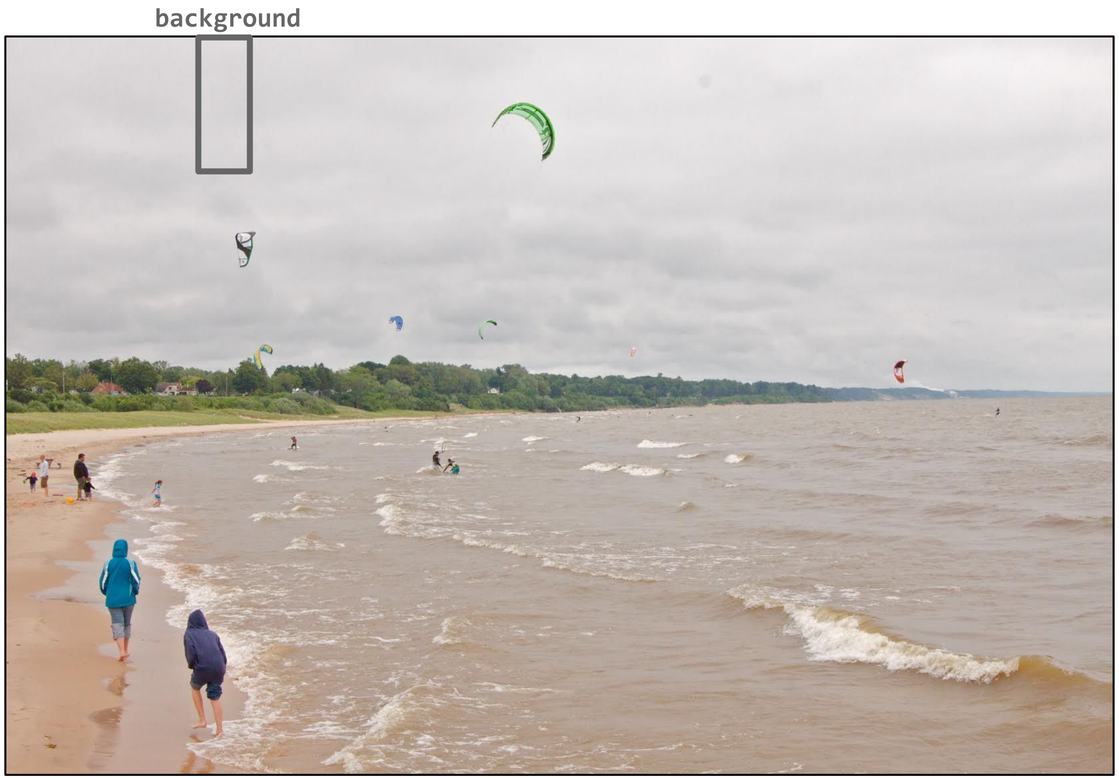
“Sliding Window” Detection



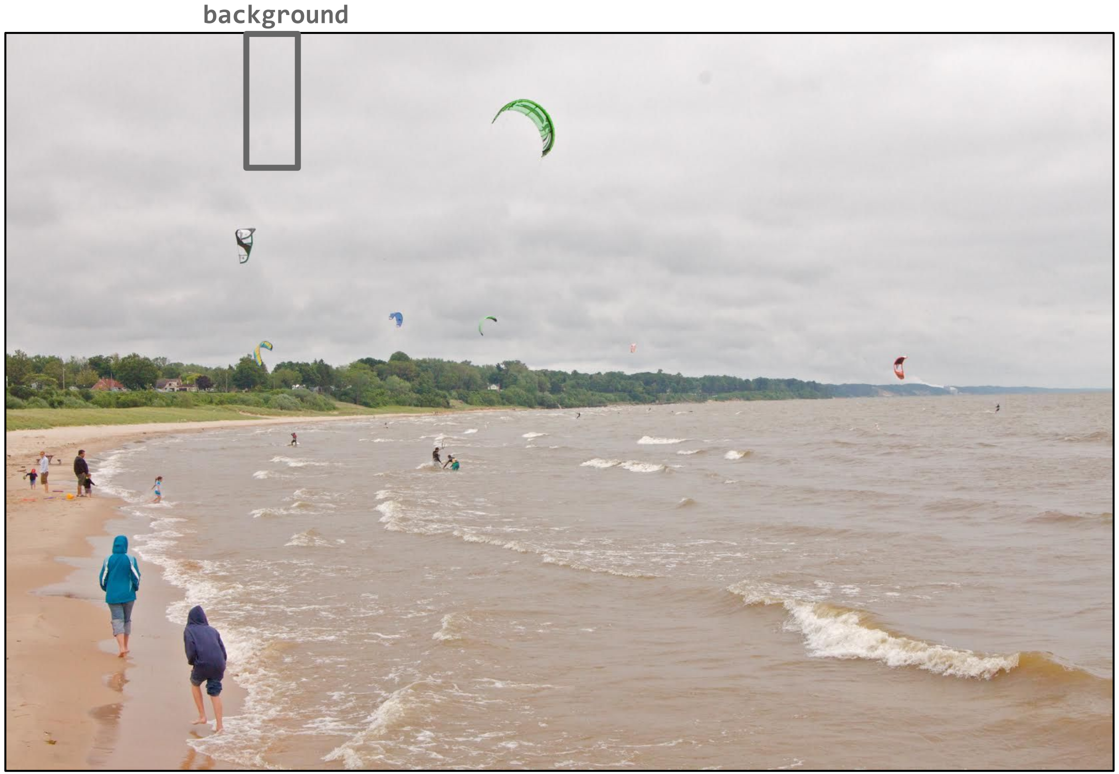
“Sliding Window” Detection



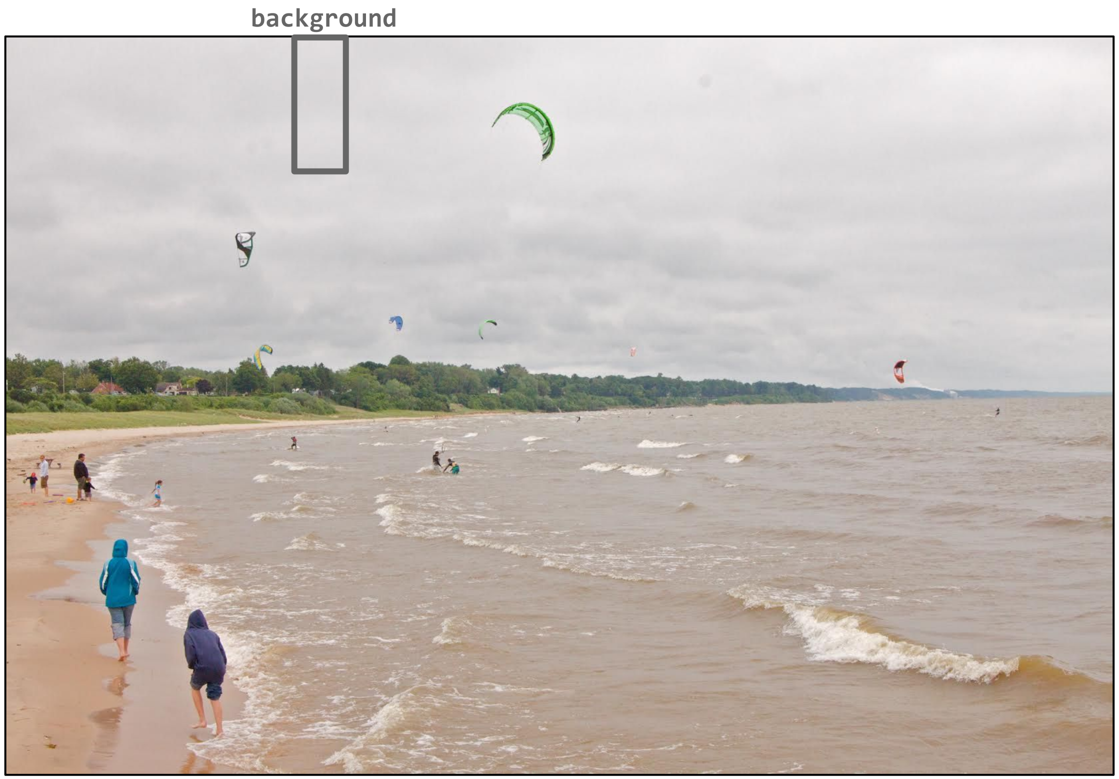
“Sliding Window” Detection



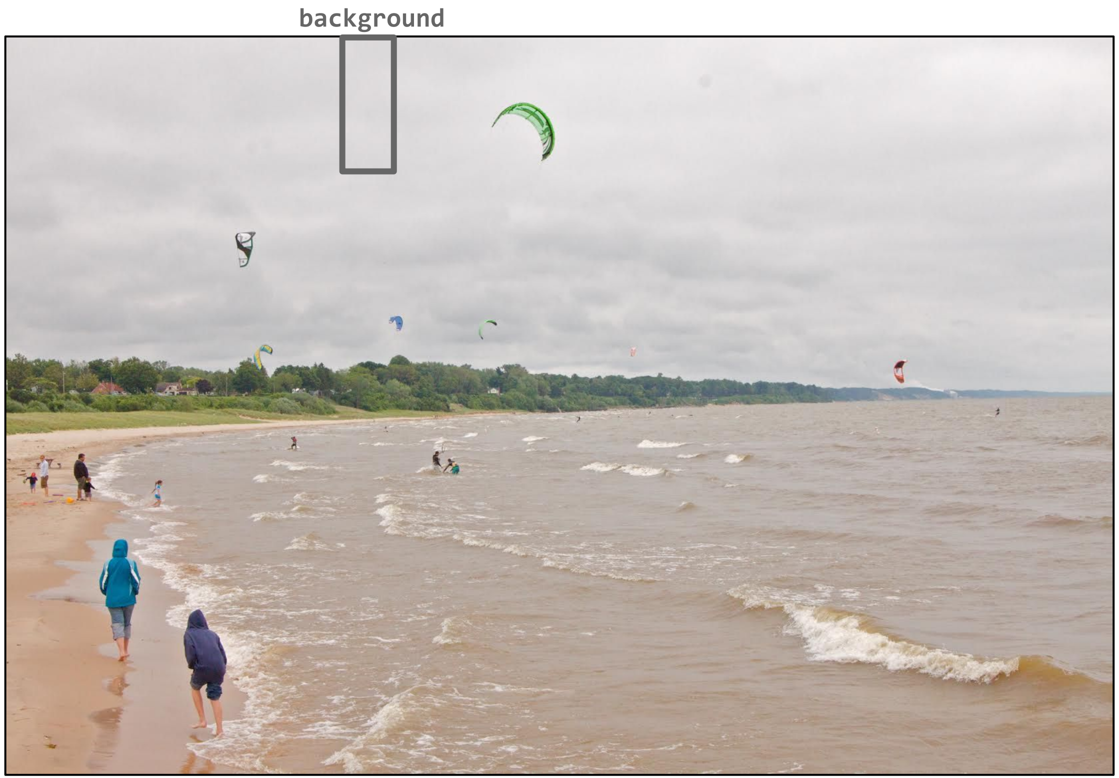
“Sliding Window” Detection



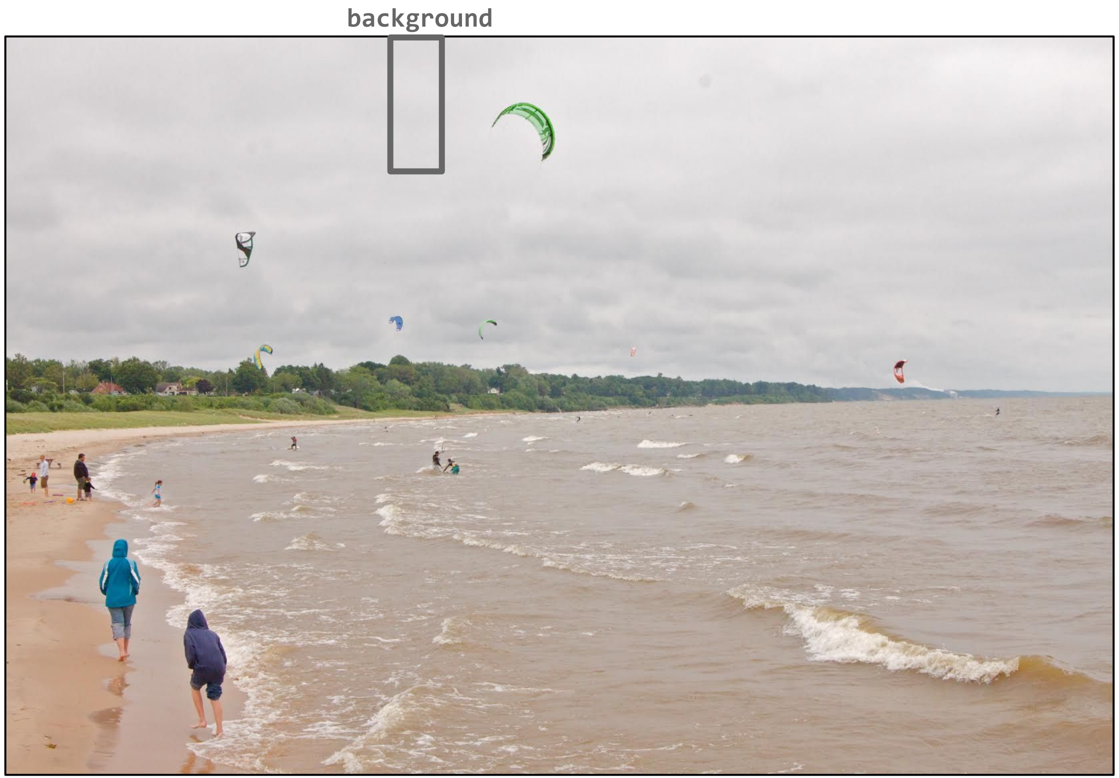
“Sliding Window” Detection



“Sliding Window” Detection

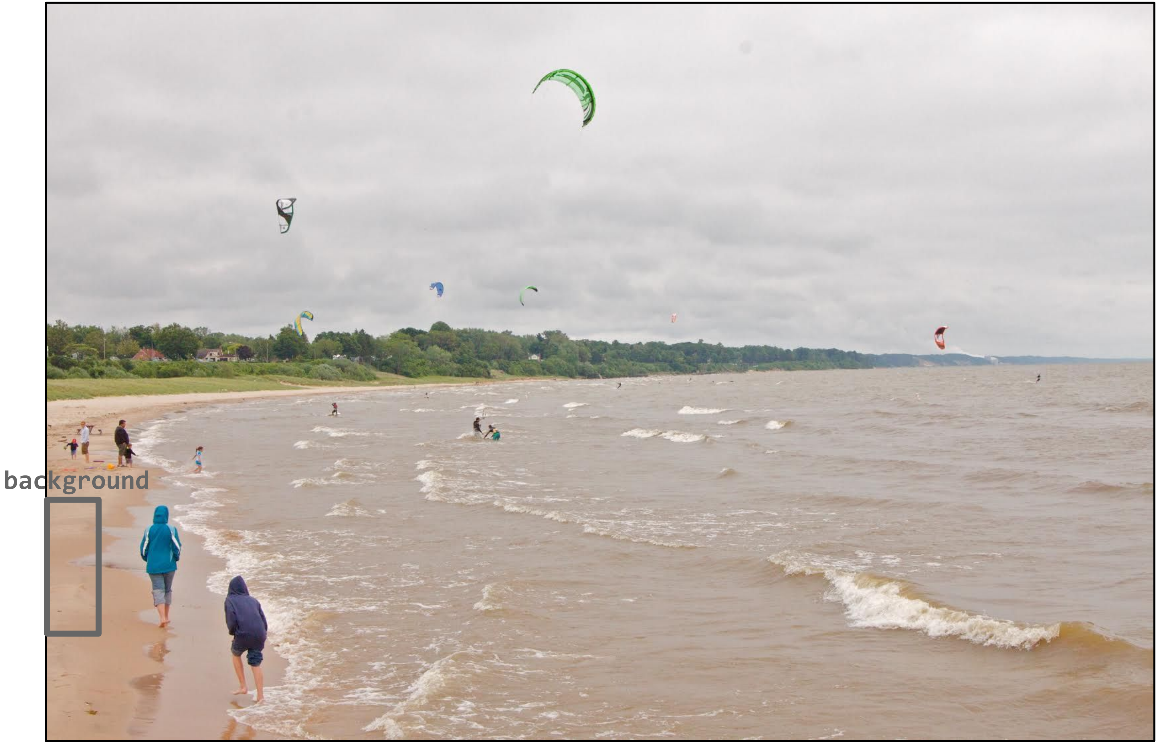


"Sliding Window" Detection



“Sliding Window” Detection

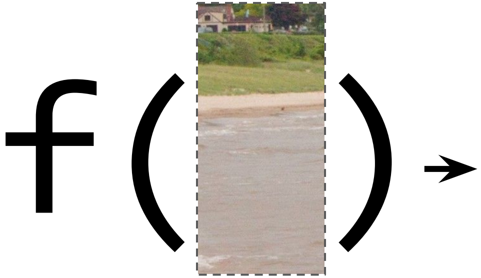
“Sliding Window” Detection



“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

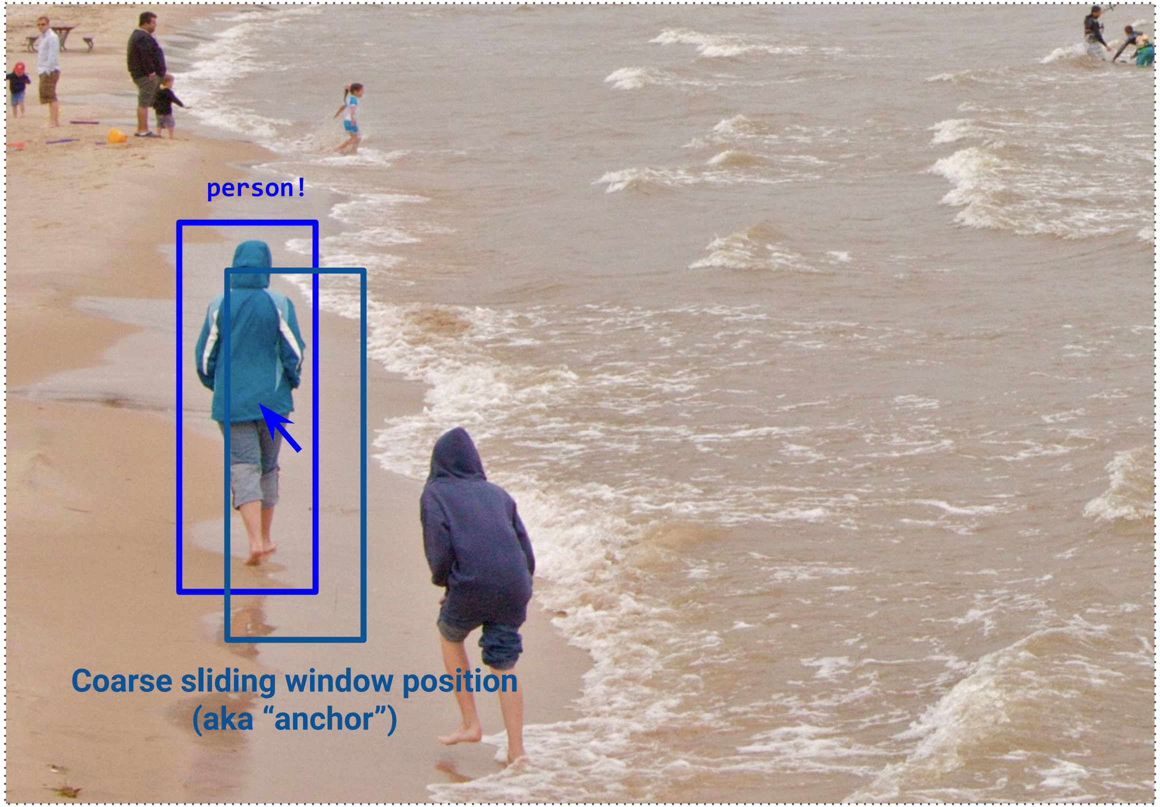
(... or can it be?)

Bounding Box Regression

Idea:

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

Bounding Box Regression



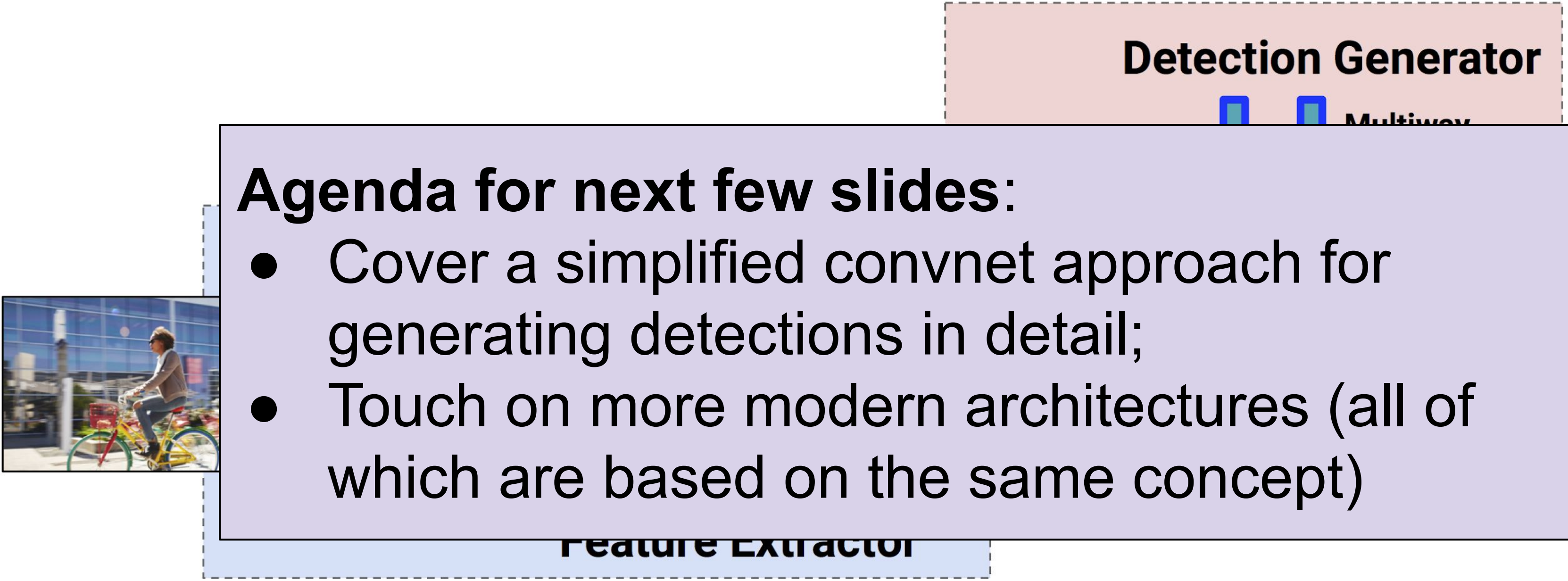
Idea:

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

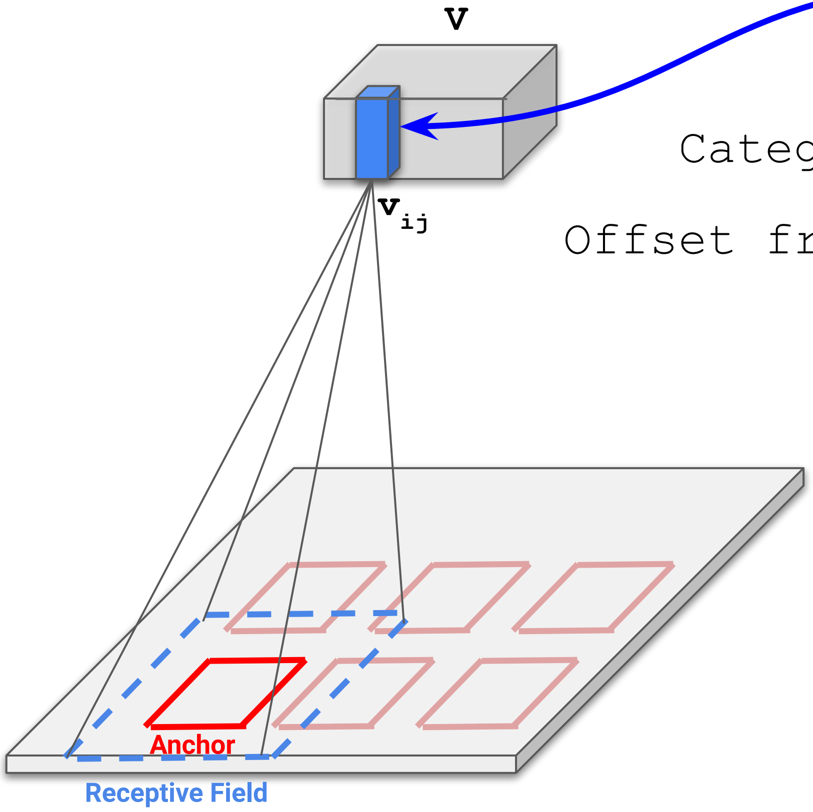


Agenda for next few slides:

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

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

A simplified convnet for detection

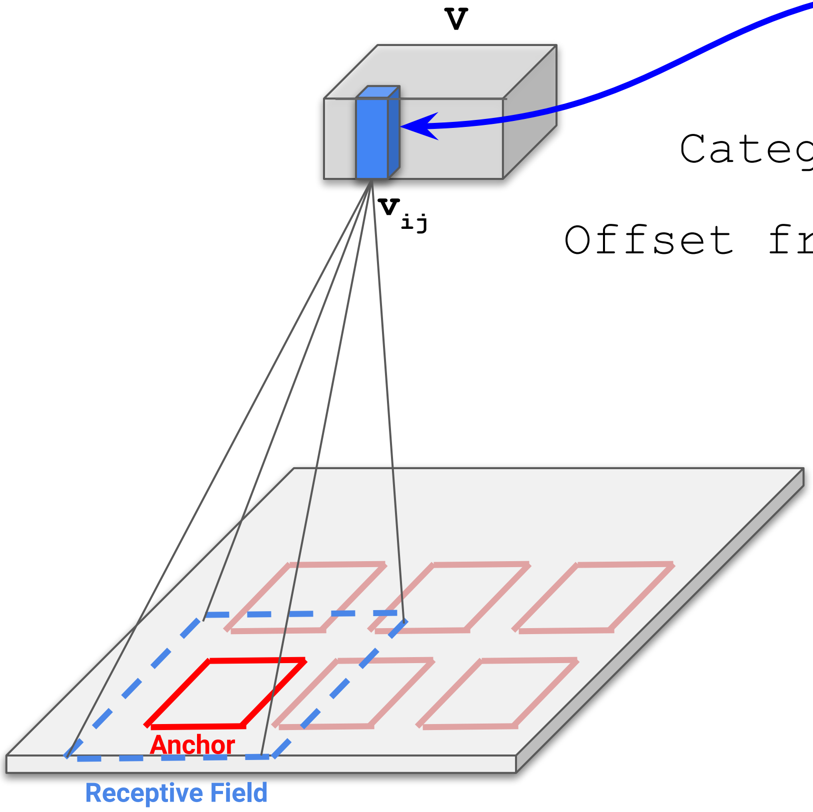


Think of each feature vector $\mathbf{v}_{ij}$ as corresponding to a sliding window (anchor).

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

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

A simplified convnet for detection



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

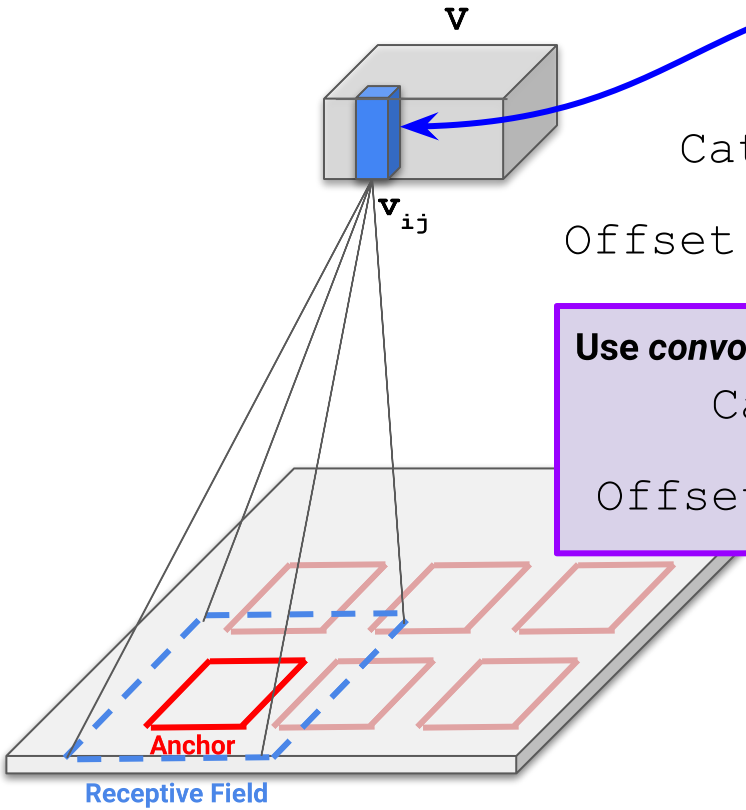
$$\texttt{Category score} = \texttt{SoftMax}(W^{cls} \cdot \mathbf{v}_{ij})$$

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

Use the same W^{loc} and W^{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).

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

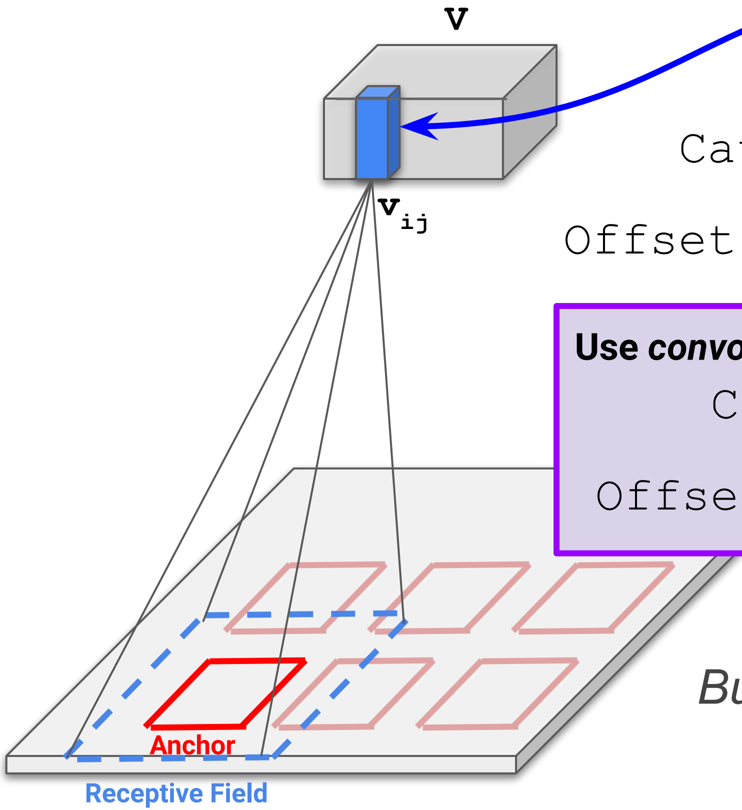
$$\text{Offset from anchor} = W^{loc} \cdot \mathbf{v}_{ij}$$

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

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

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

A simplified convnet for detection



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

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

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

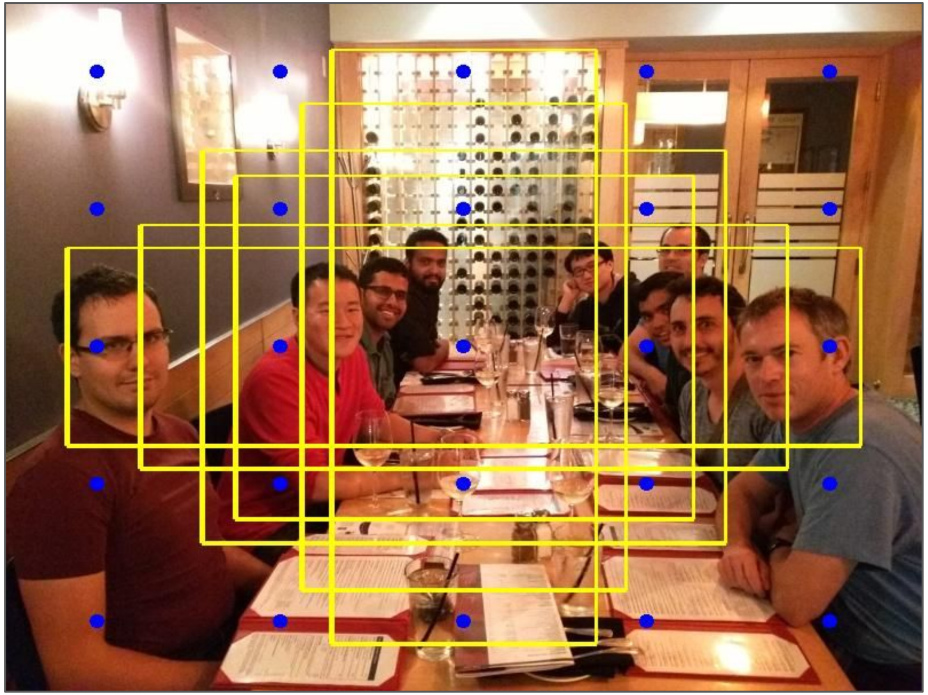
Use *convolution* to do simultaneous prediction for all anchors:

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

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

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)



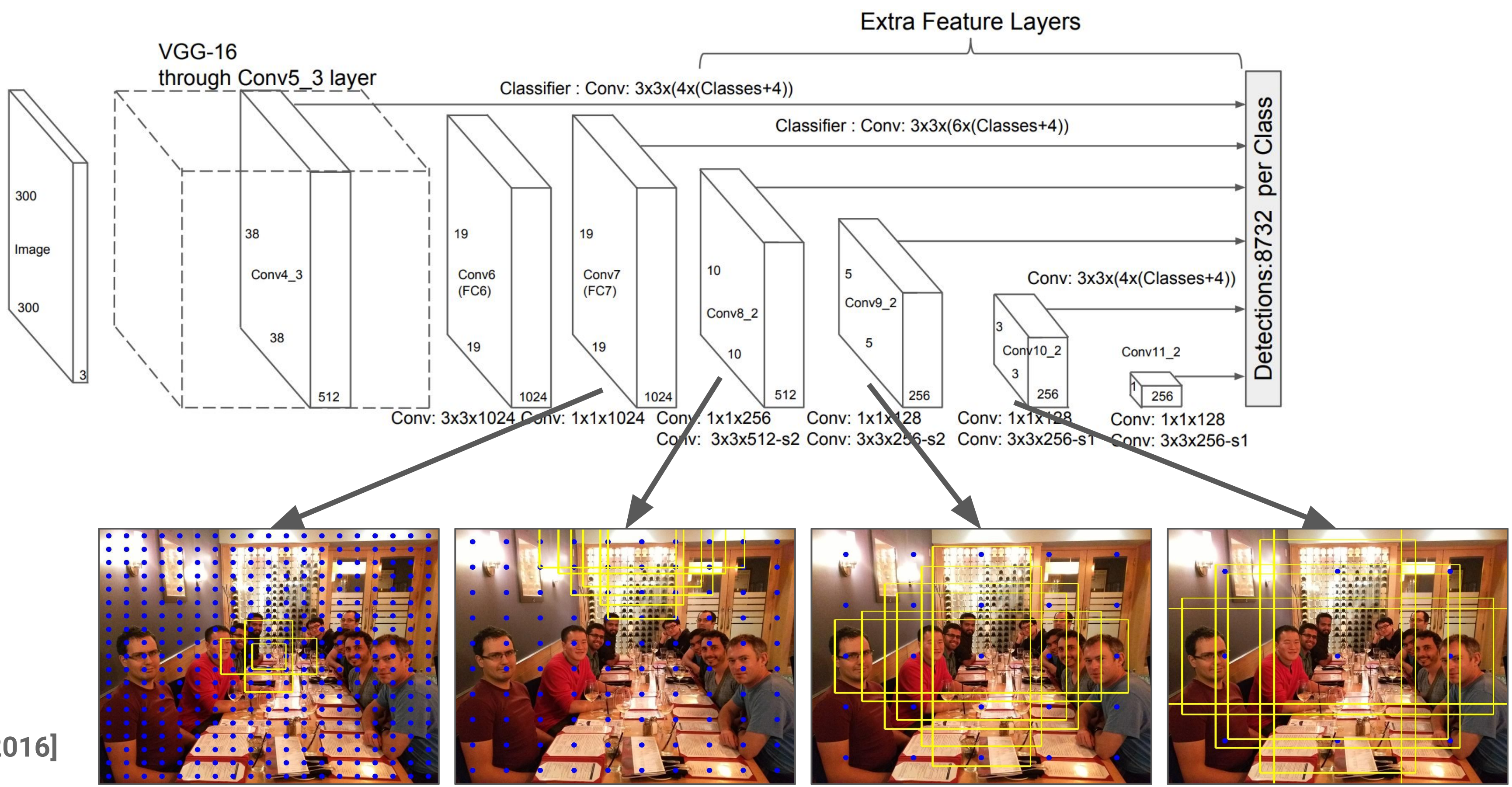
$\mathrm{SoftMax}(W^{cls,ar1} \cdot \mathbf{v}_{ij})$
$W^{loc,ar1} \cdot \mathbf{v}_{ij}$

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

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

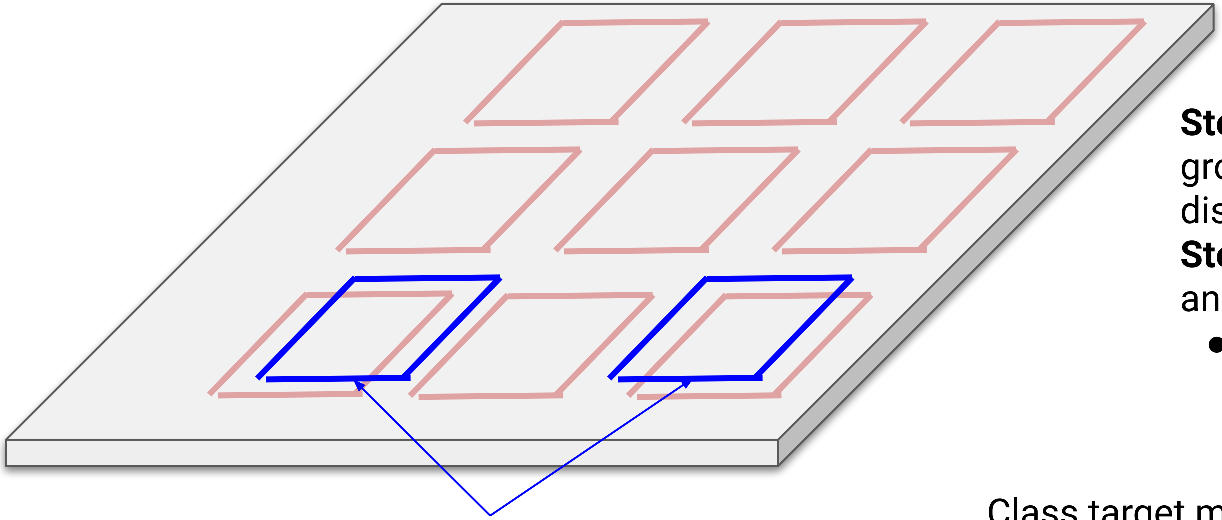
...

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)

$$\begin{pmatrix} gt_{xmin} - anchor_{xmin} \\ gt_{ymin} - anchor_{ymin} \\ gt_{xmax} - anchor_{xmax} \\ gt_{ymax} - anchor_{ymax} \end{pmatrix}$$

Typical Training Objective

Common to use other location losses here...

Per-anchor Loss:

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

$$+ \quad \beta * \text{SoftMaxCrossEntropy}(\mathbf{t}^{cls}, W^{cls} \cdot \mathbf{v}_{ij})$$

Total Loss: Average per-anchor loss over anchors

Minimize w/SGD

Localization Loss in Box Encoding Space

$$\textbf{Huber}\left(\begin{pmatrix} \dfrac{x_c}{w_a} \\ \dfrac{y_c}{h_a} \\ \log w \\ \log h \end{pmatrix}, \quad \mathtt{W}^{\mathtt{loc}}_{\mathbf{j}} \cdot \mathbf{v}_{\mathbf{i}} \right)$$

Localization Loss in Box Encoding Space

$$\textbf{Huber}\left(\begin{pmatrix}\boxed{\begin{matrix}\dfrac{x_c}{w_a}\\[6pt] \dfrac{y_c}{h_a}\end{matrix}}\\ \log w\\ \log h\end{pmatrix},\ \mathtt{W}^{\mathtt{loc}}_{\mathtt{j}} \cdot \mathbf{v}_{\mathtt{i}}\right)$$

Groundtruth box center in normalized coordinates

Localization Loss in Box Encoding Space

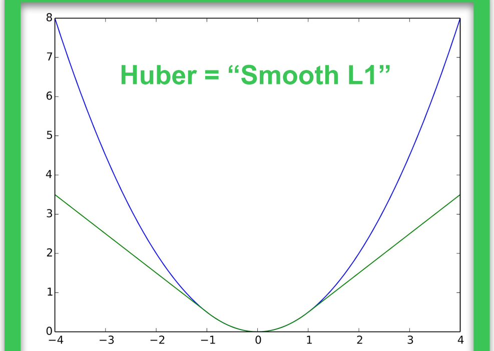
$$\textbf{Huber}\left(\begin{pmatrix} \boxed{\begin{matrix} \dfrac{x_c}{w_a} \\ \dfrac{y_c}{h_a} \end{matrix}} \\ \boxed{\begin{matrix} \log w \\ \log h \end{matrix}} \end{pmatrix}, \quad \mathtt{W}^{\mathtt{loc}}_{\mathtt{j}} \cdot \mathbf{v}_{\mathtt{i}} \right)$$

Groundtruth box center in normalized coordinates

Groundtruth box size in log space

Localization Loss in Box Encoding Space

Huber



$$\left(\begin{bmatrix} \frac{x_c}{w_a} \\ \frac{y_c}{h_a} \\ \log w \\ \log h \end{bmatrix}, \quad \mathtt{W}^{\mathtt{loc}}_{\mathtt{j}} \cdot \mathbf{v}_{\mathtt{i}} \right)$$

Groundtruth box center in normalized coordinates

Groundtruth box size in log space

Classification Loss: Dealing with Class Imbalance

Problem:

negative/background anchors >> # positive/foreground anchors

Typical solutions:

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

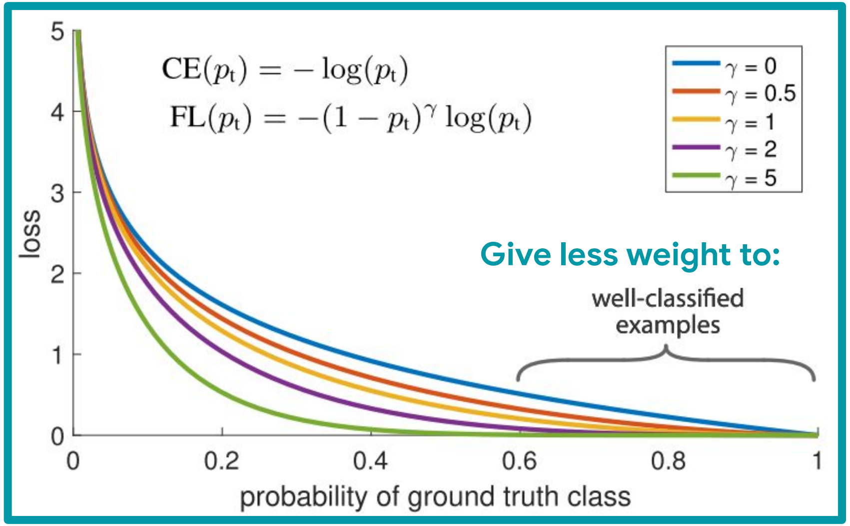
Classification Loss: Dealing with Class Imbalance

Problem:

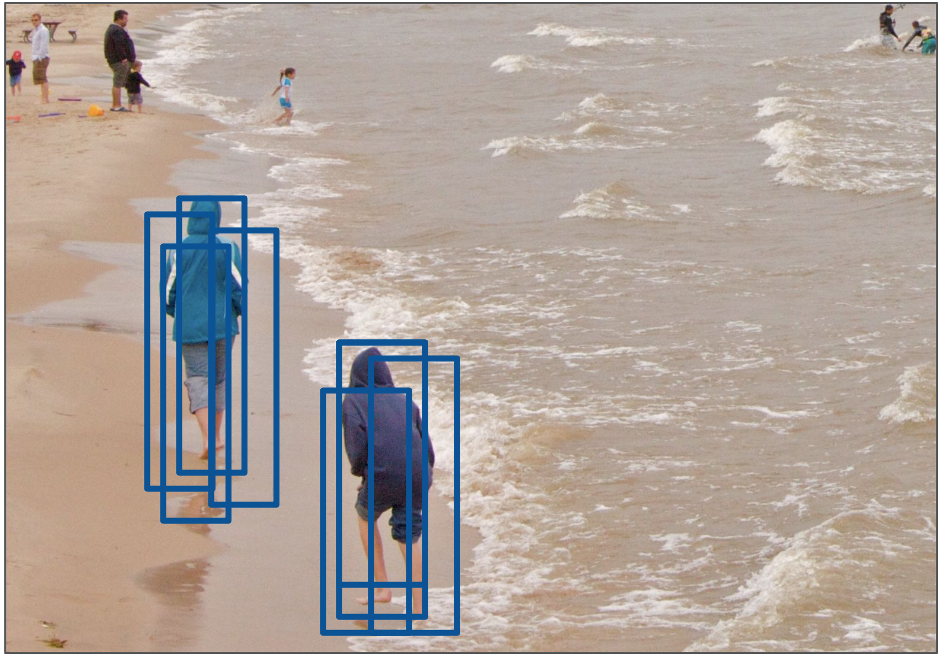
\# negative/background anchors >> # positive/foreground anchors

Typical solutions:

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



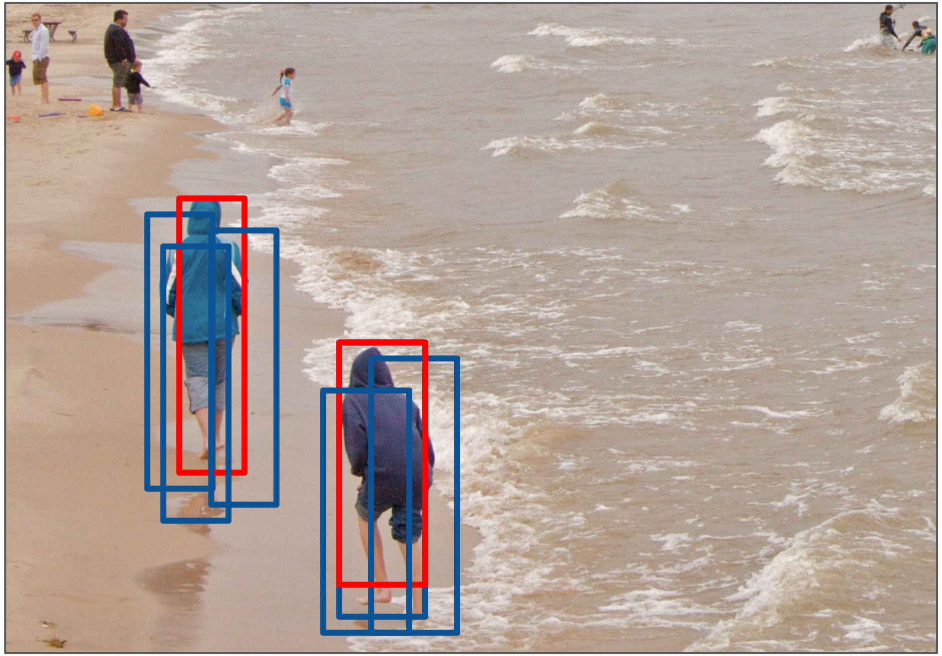
Dealing with multiple detections of the same object



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)



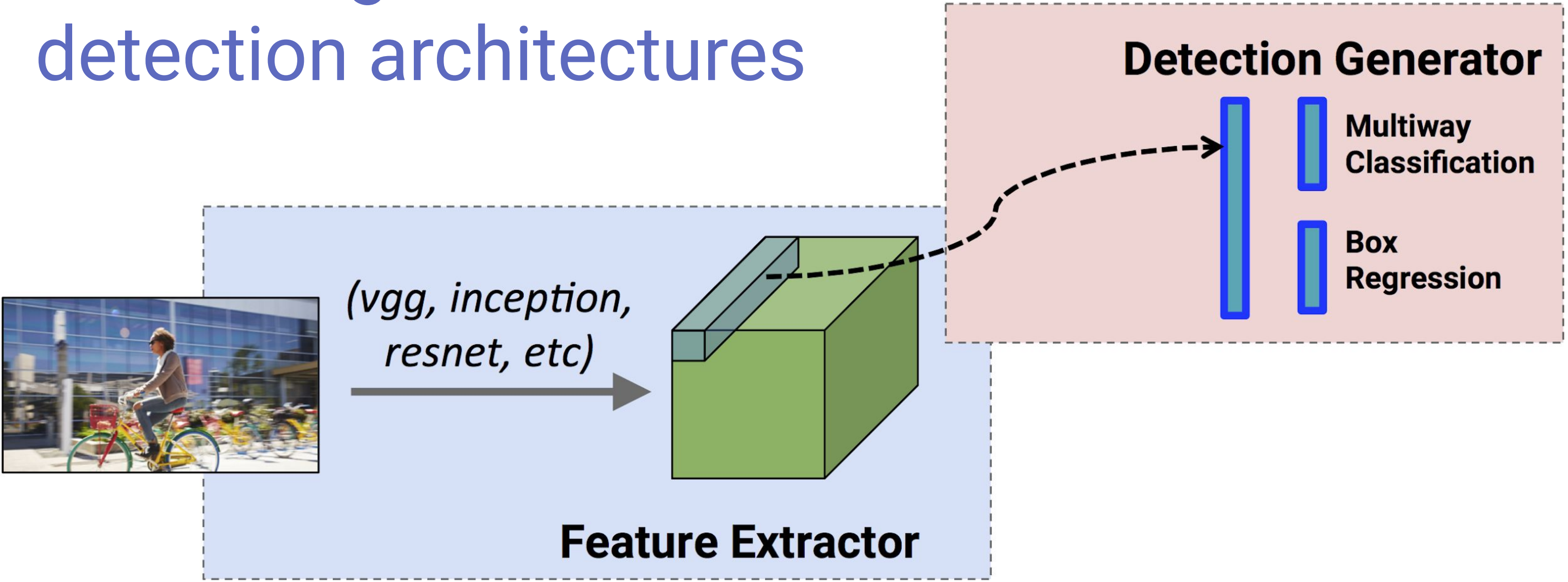
Algorithm:

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

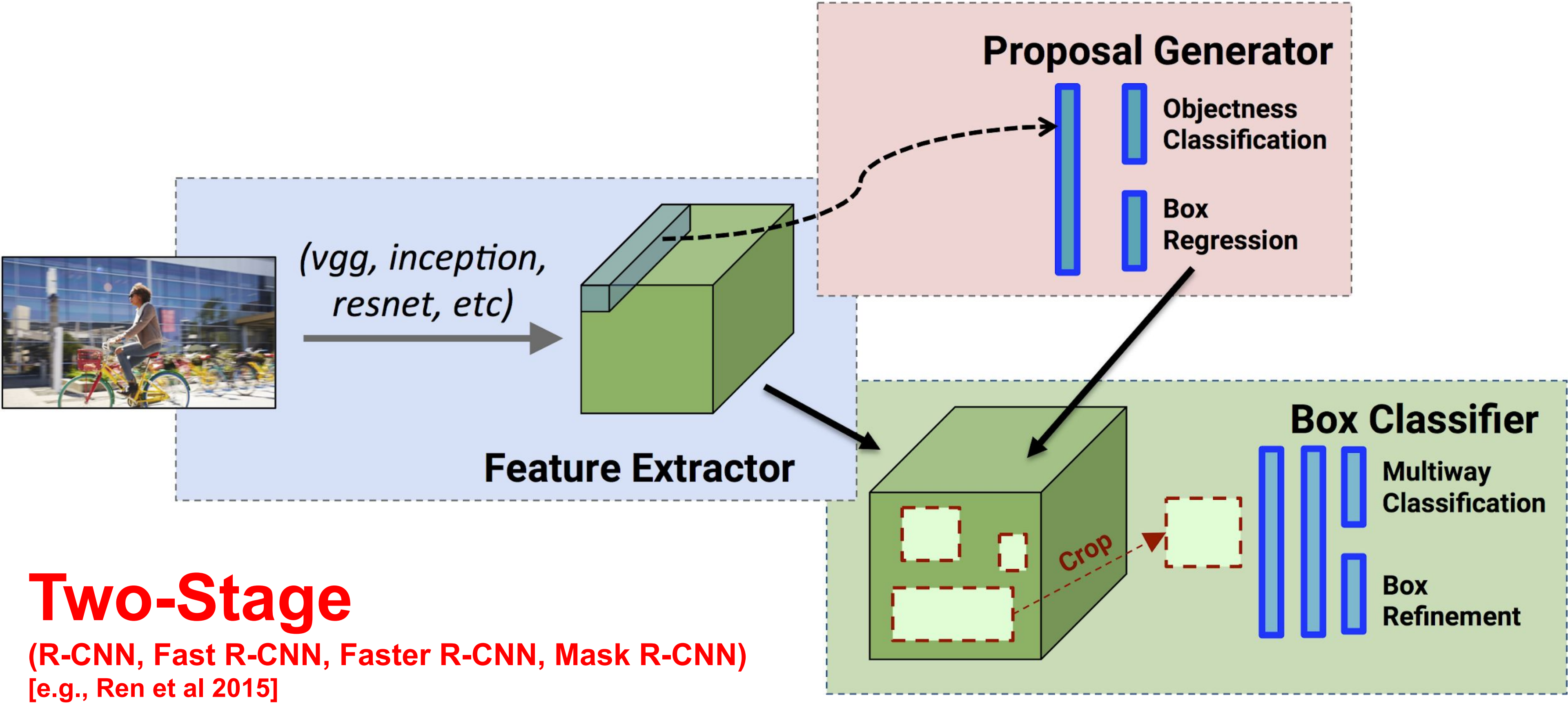
Detection "***meta-architectures***" are a recipe for converting classification architectures into detection architectures



Single Stage Models

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

Another popular meta-architecture



Two-Stage

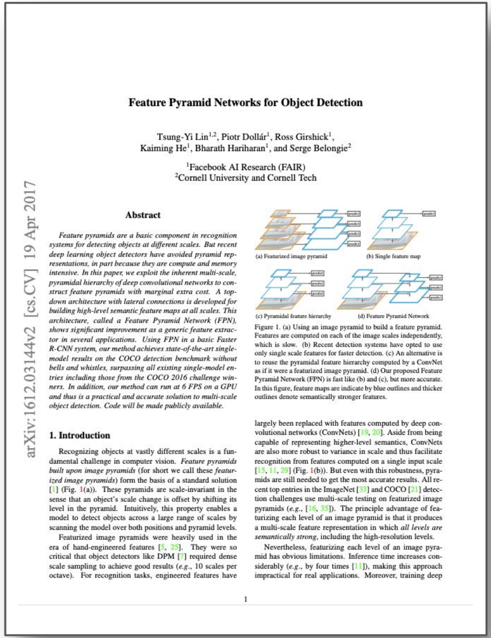
(R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN)

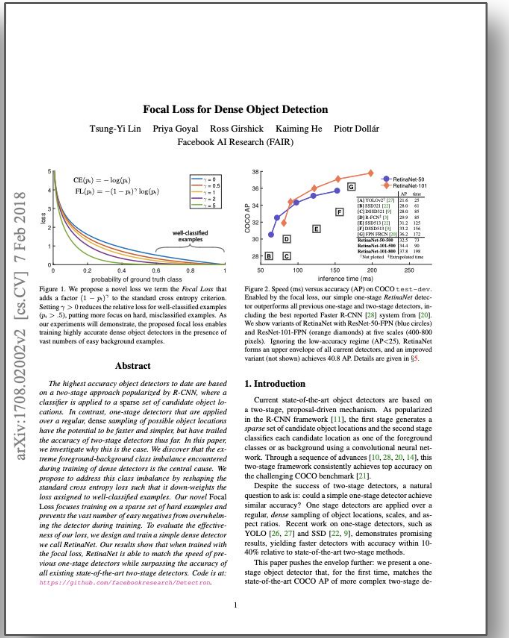
[e.g., Ren et al 2015]

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

Key Ideas:

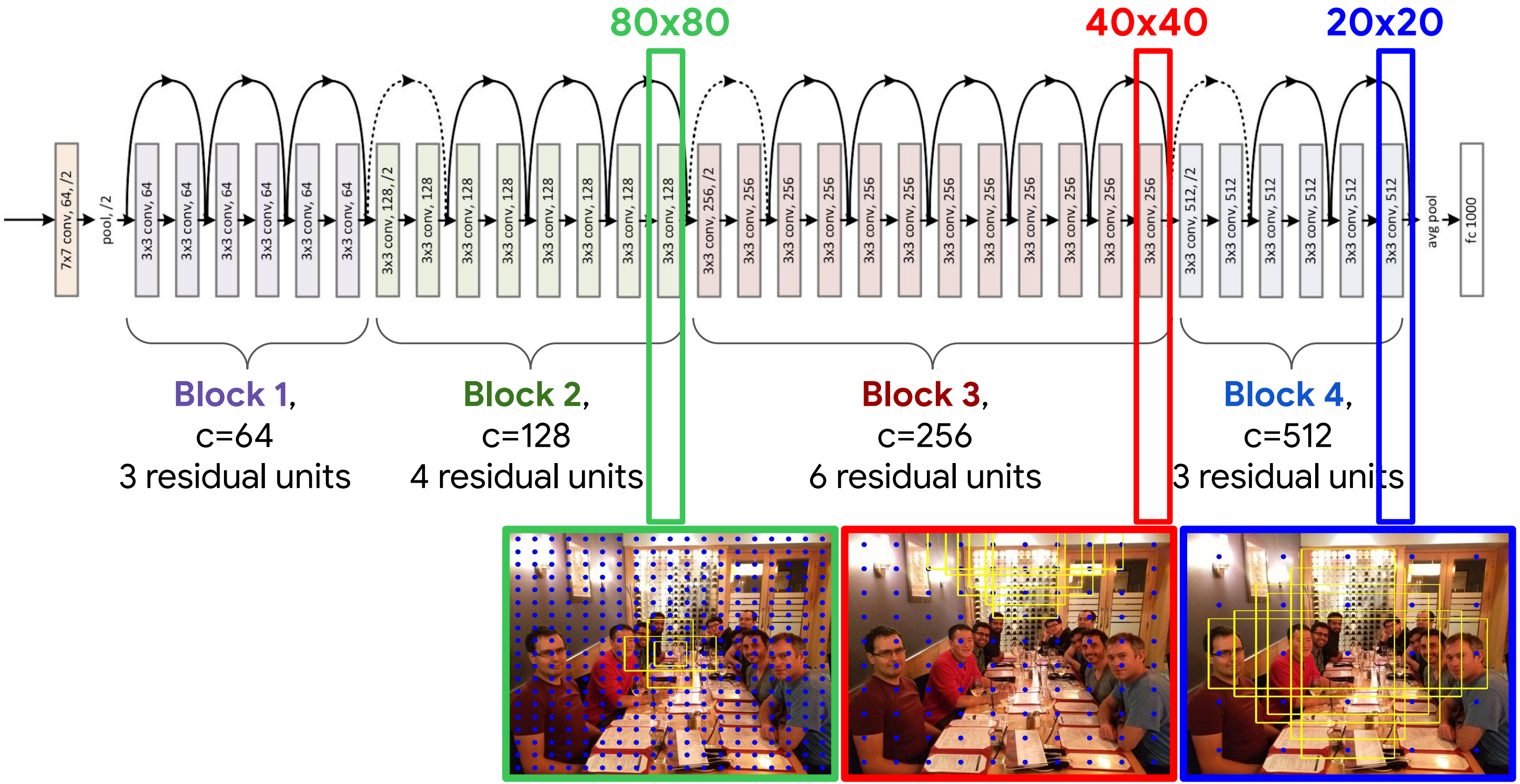
- **Multi-Resolution Feature Extractor:**
  - **Resnet + FPN**
- Focal Loss
- Smart initialization of classification bias for fast training:
  - On Google TPUs, can train on COCO dataset in 3.5 hours



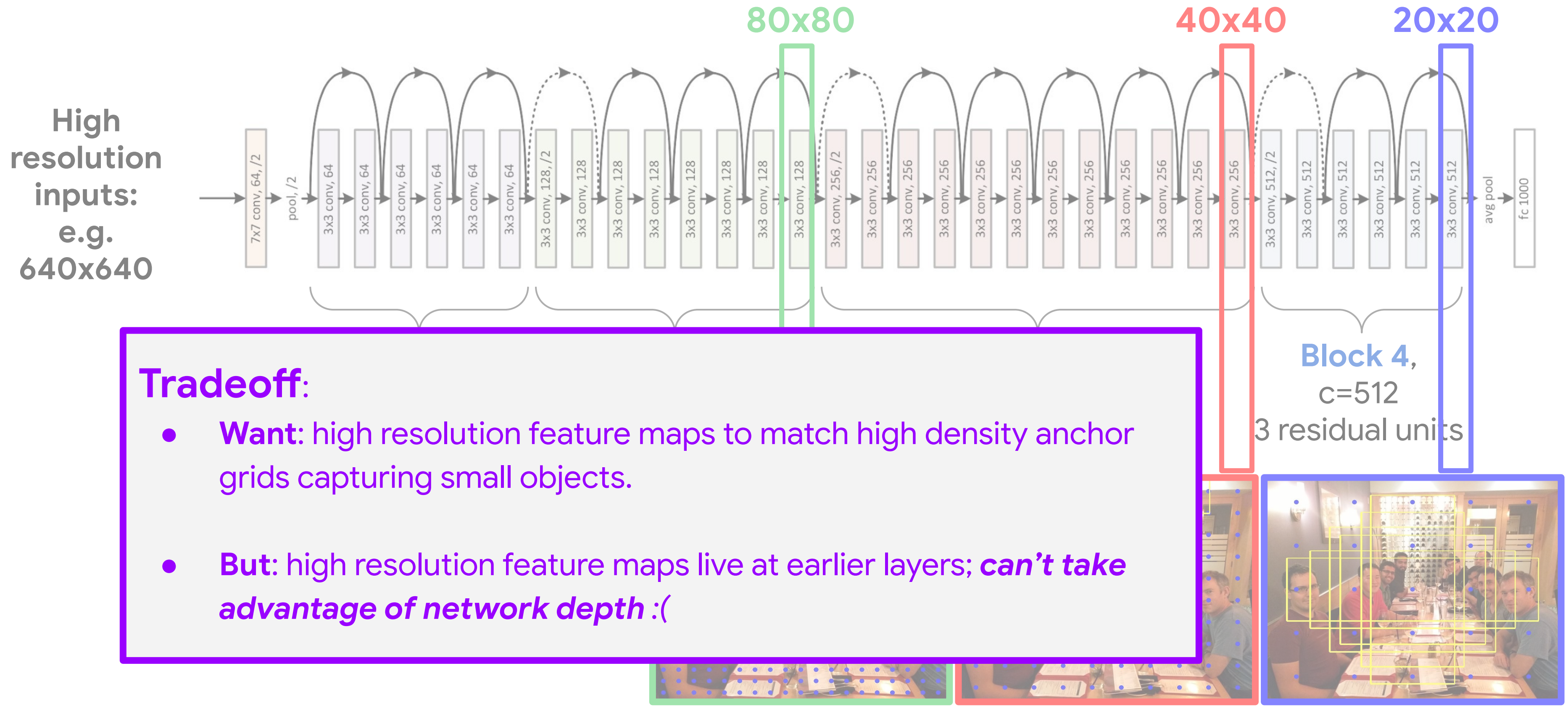


One way to extracting multiresolution feature maps from Resnet

High resolution inputs: e.g. 640x640



One way to extracting multiresolution feature maps from Resnet

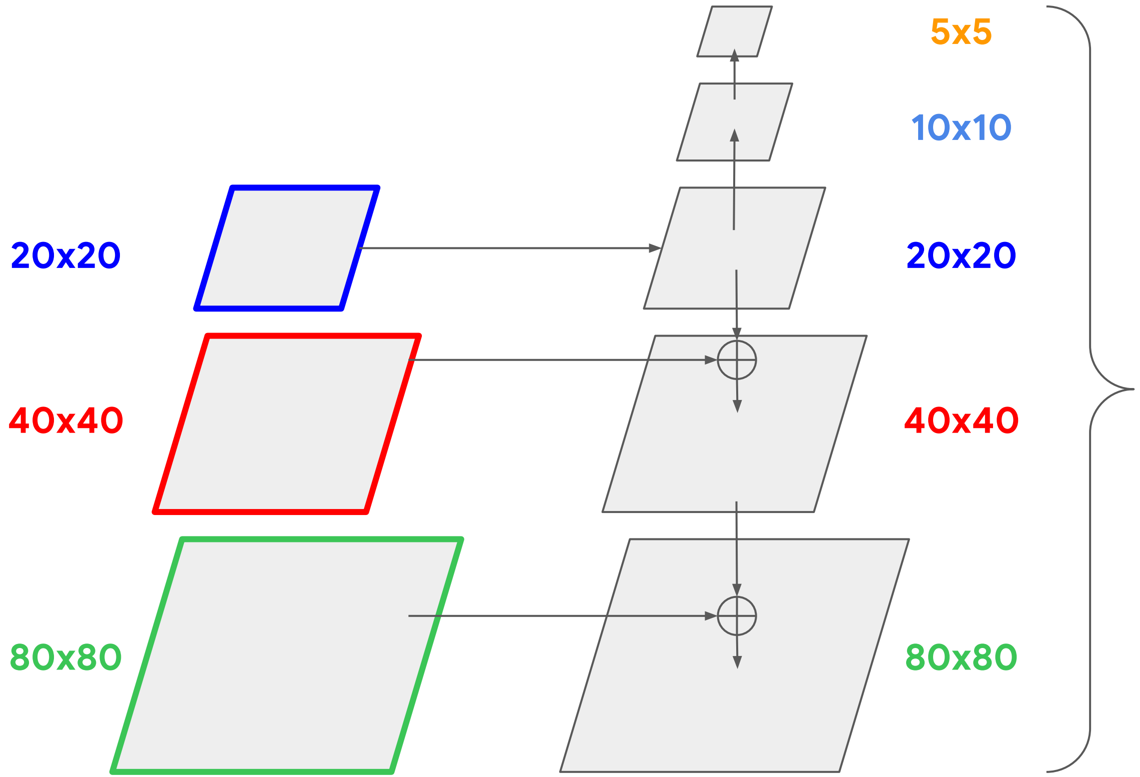


Tradeoff:

- **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*** :(

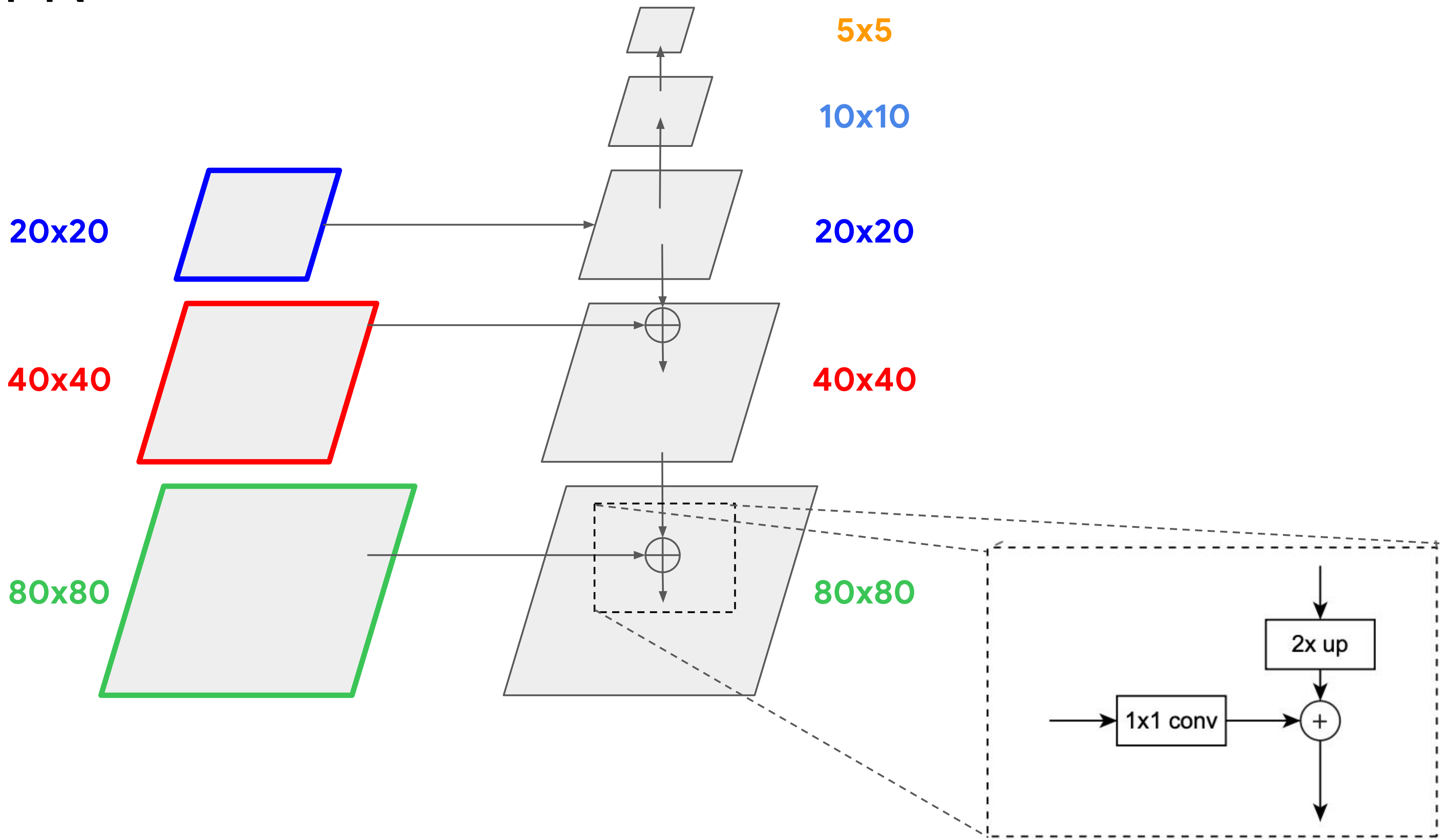
Enter Feature Pyramid Networks (FPN)



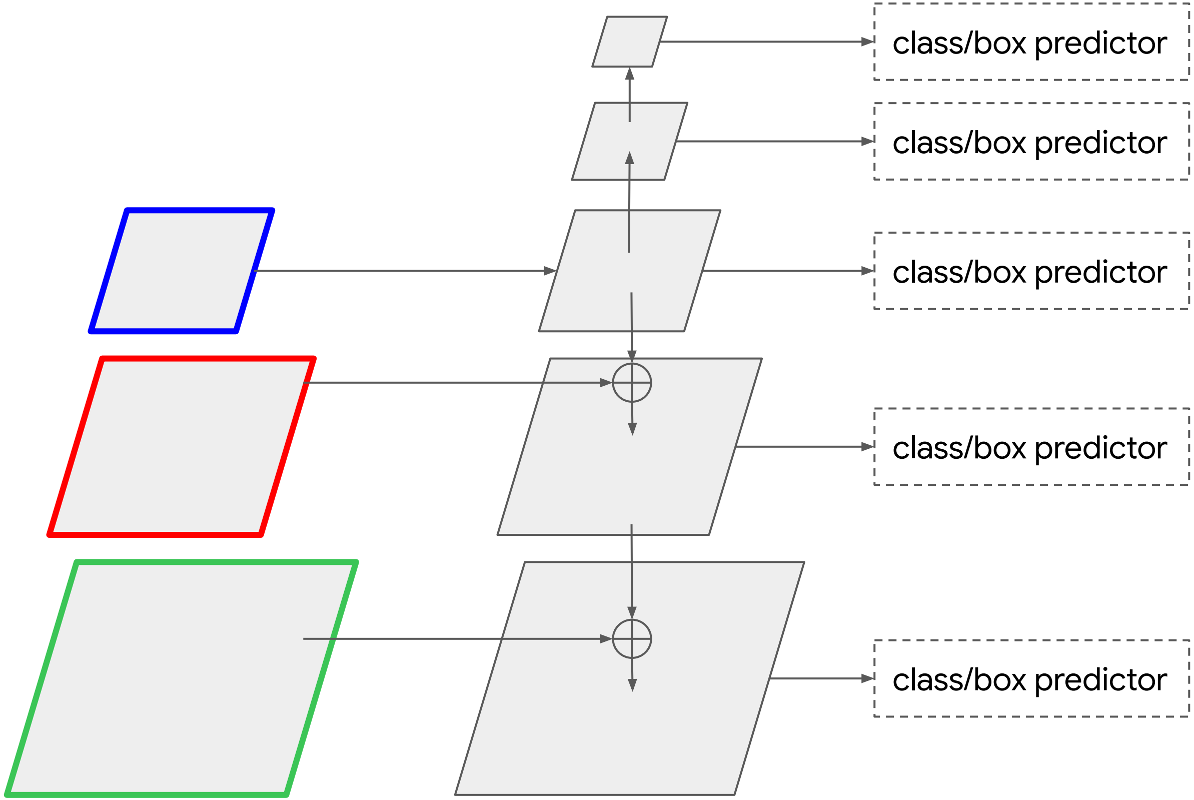
All resolutions now benefit from deep feature representations

Feature Pyramid Networks for Object Detection by Lin et al 2017
Focal Loss for Dense Object Detection by Lin et al 2018

FPN

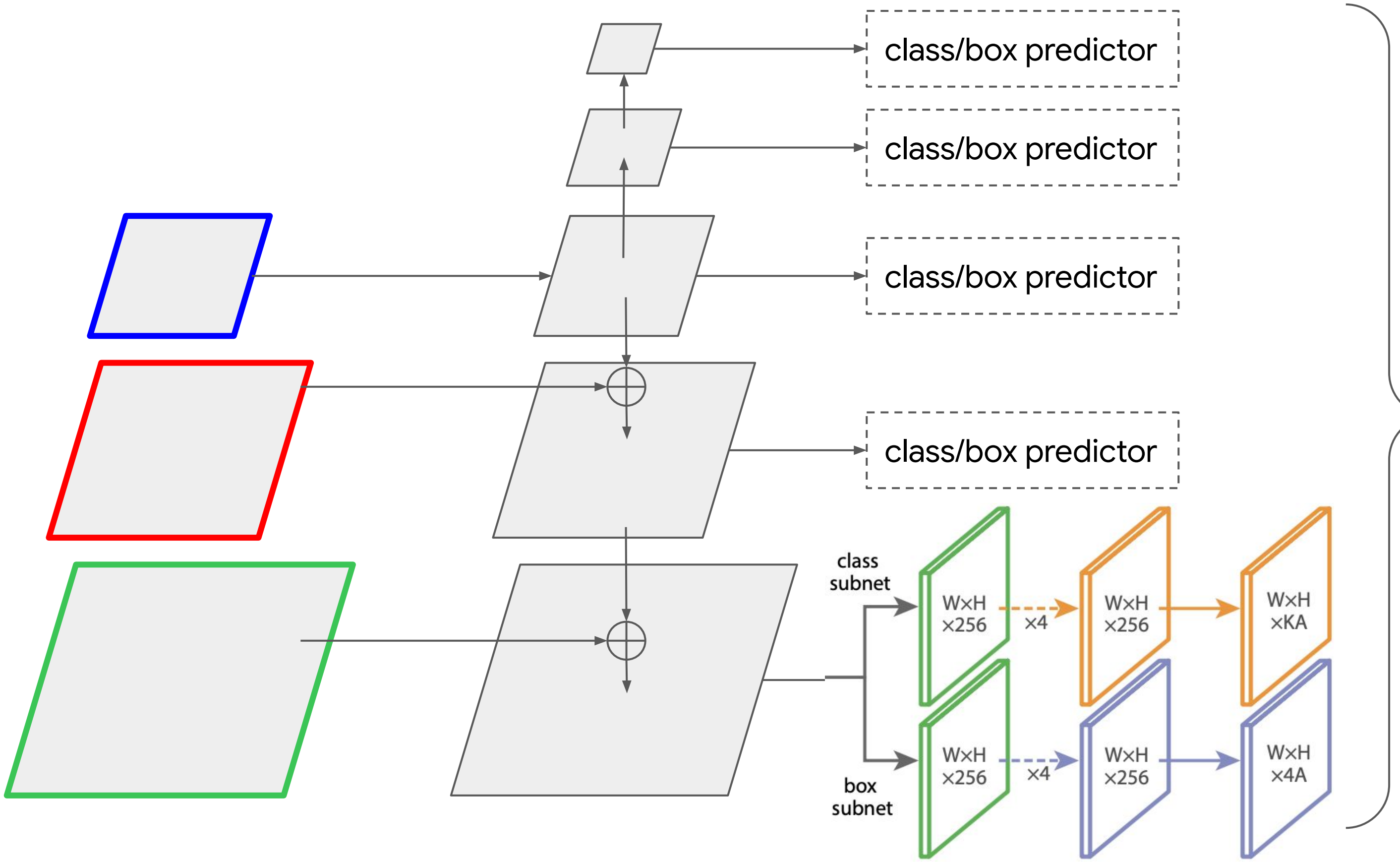


FPN



Feature Pyramid Networks for Object Detection by Lin et al 2017
Focal Loss for Dense Object Detection by Lin et al 2018

FPN



Shared predictor weights across FPN levels

Feature Pyramid Networks for Object Detection by Lin et al 2017
Focal Loss for Dense Object Detection by Lin et al 2018

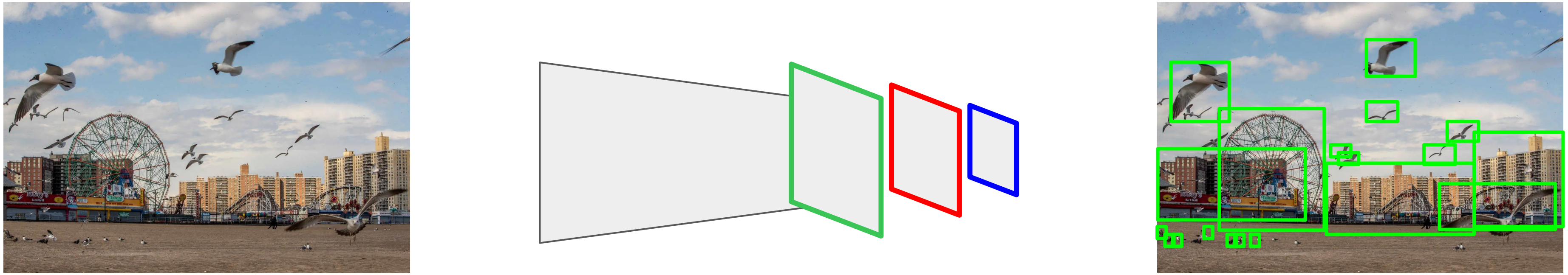
RetinaNet with Resnet 101 with FPN Feature Extractor

"RetinaNet" with EfficientNet + biFPN Feature Extractor

Videos by [Karol Majek]()

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

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

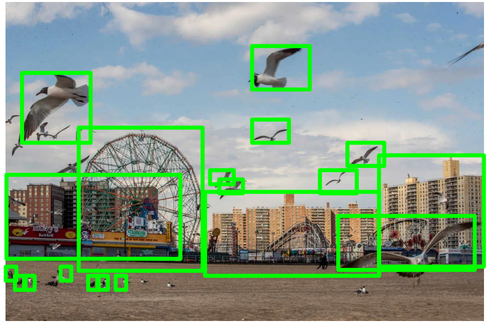


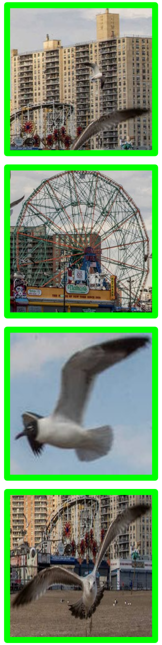
Region Proposal Network (RPN)
(Think of this as your standard single stage model)

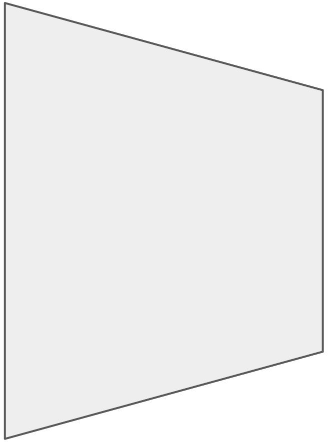
Image from [NYTimes]

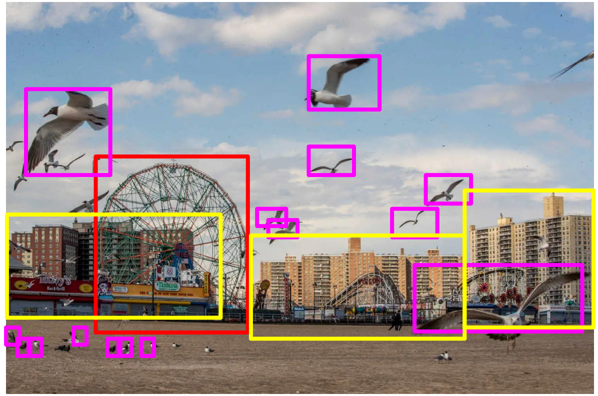
Case Study: Faster R-CNN by Ren et al 2015

Stage 2: Crop + Resize; Second stage classification







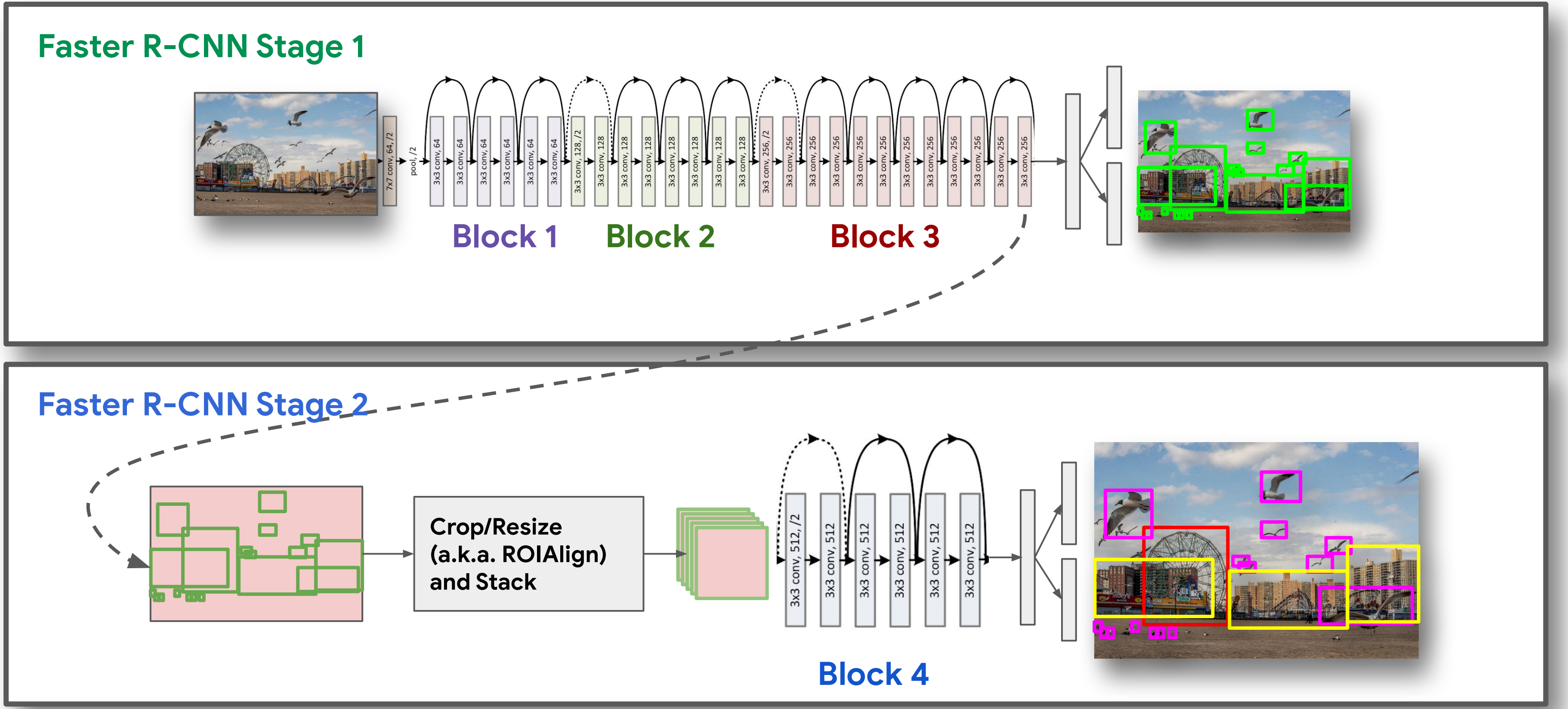


But crop from RPN features instead of pixels!

Image from [NYTimes]

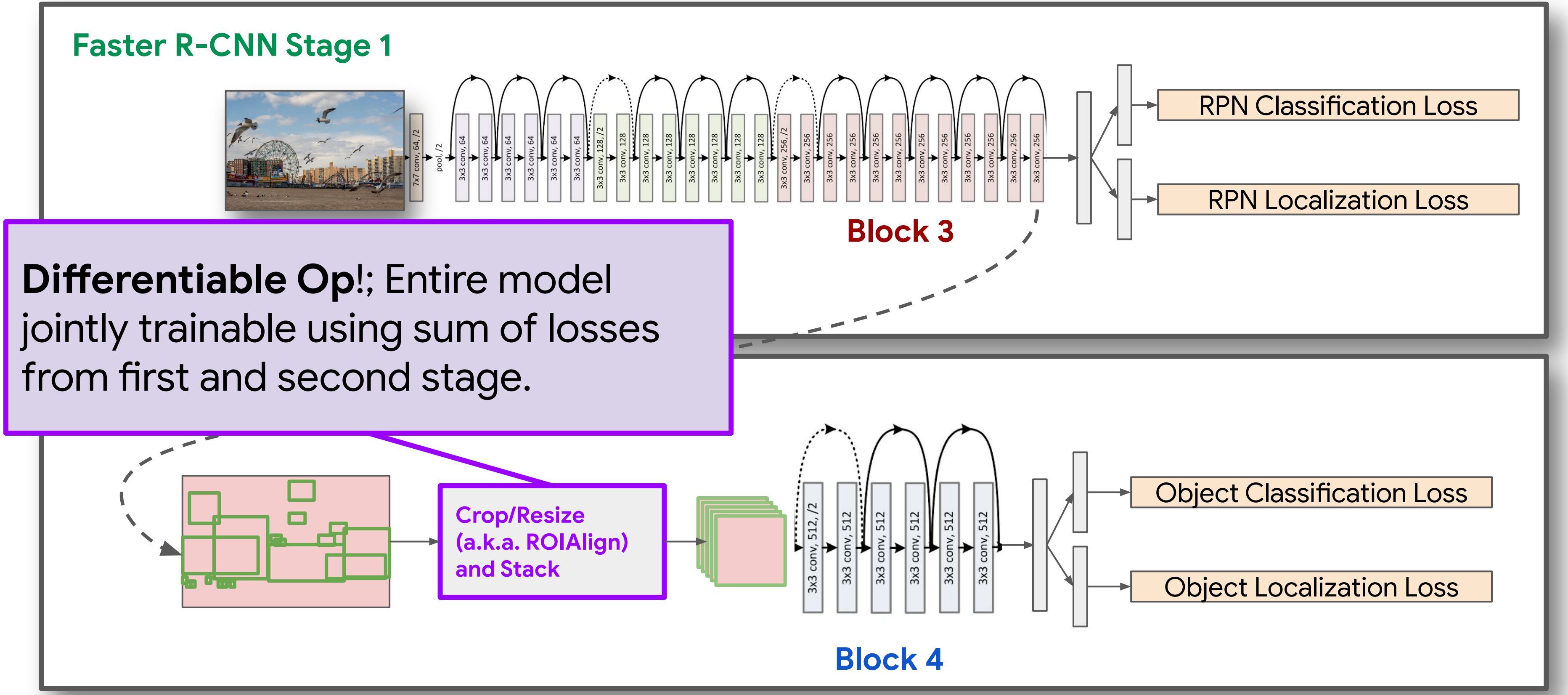
Faster R-CNN with Resnet (He et al 2015)

Winning architecture for COCO 2015 Challenge



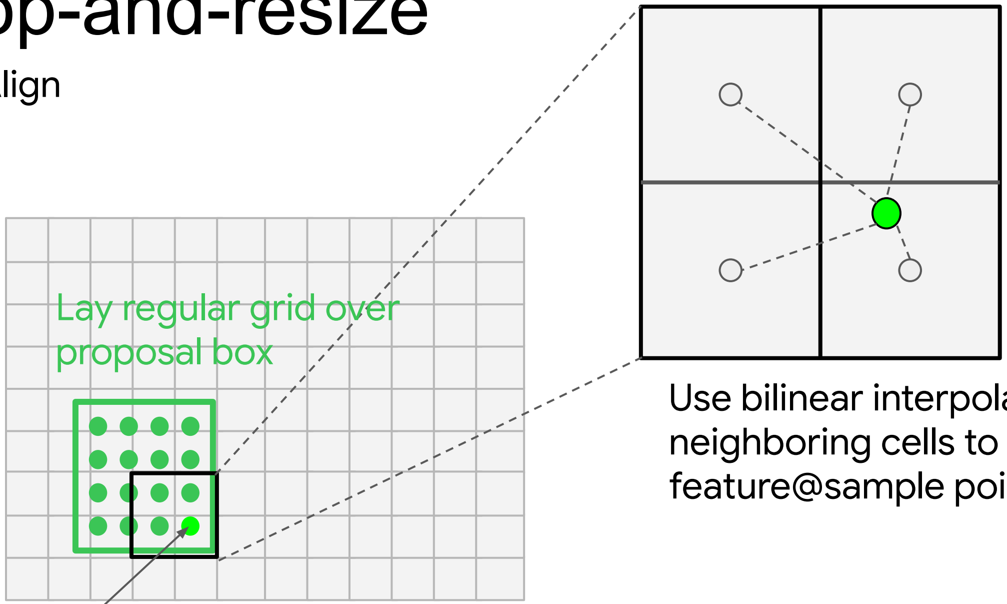
Faster R-CNN with Resnet (He et al 2015)

Winning architecture for COCO 2015 Challenge



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

Videos by [Karol Majek]

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)

Speed/accuracy trade-offs for modern convolutional object detectors by Huang et al

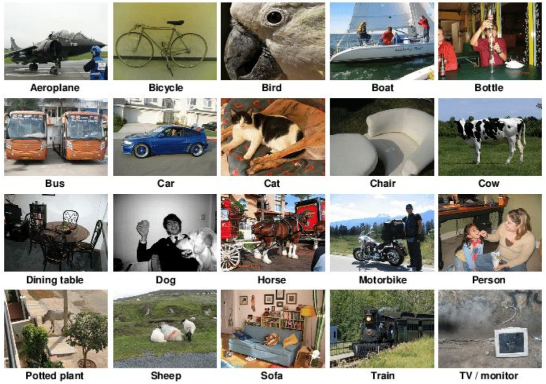
You should know:

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

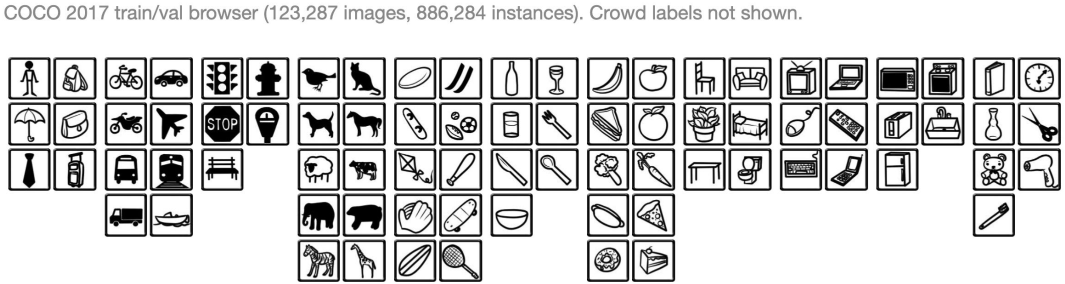
Outline

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

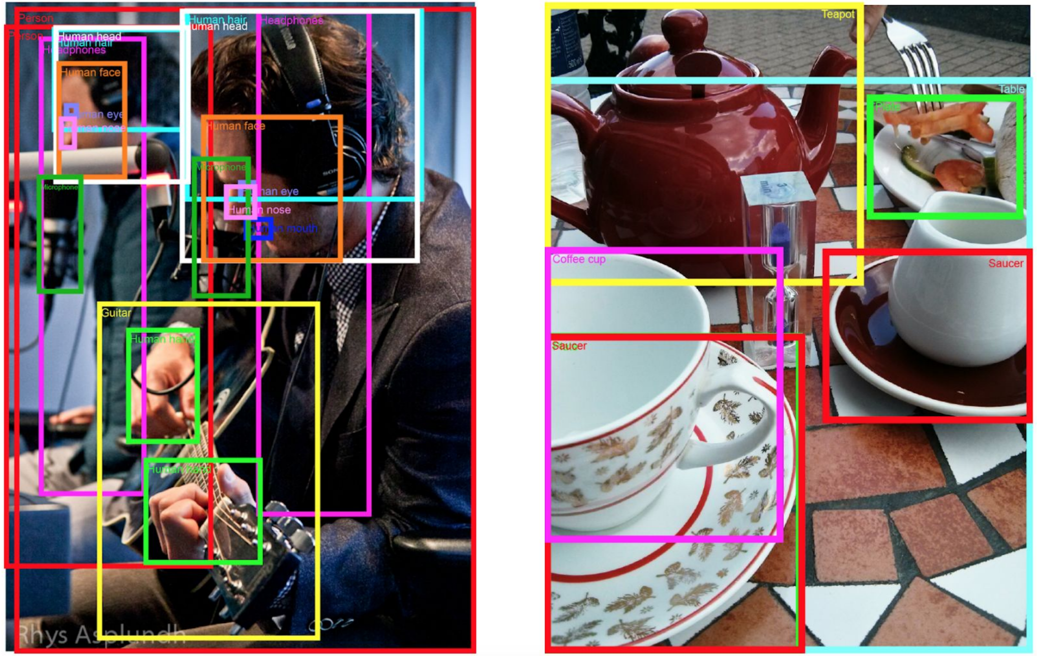
First you need a dataset...



Pascal VOC
20 classes, 5K images

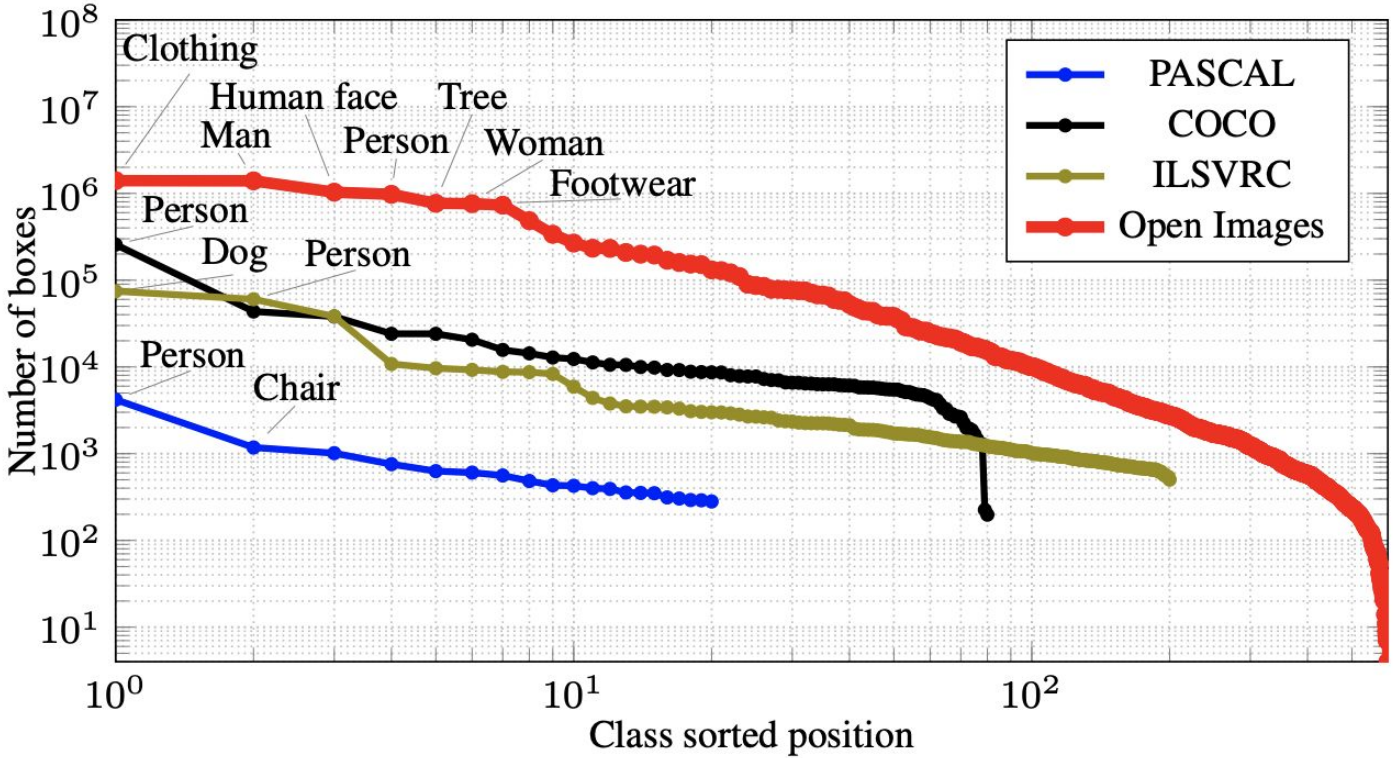


COCO
80 classes, ~120K images



Open Images
(600 classes, 1.7M images)

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



Clothing
Human face
Tree
Man
Person
Woman
Footwear
Person
Dog
Person
Person
Chair
PASCAL
COCO
ILSVRC
Open Images
Number of boxes
Class sorted position

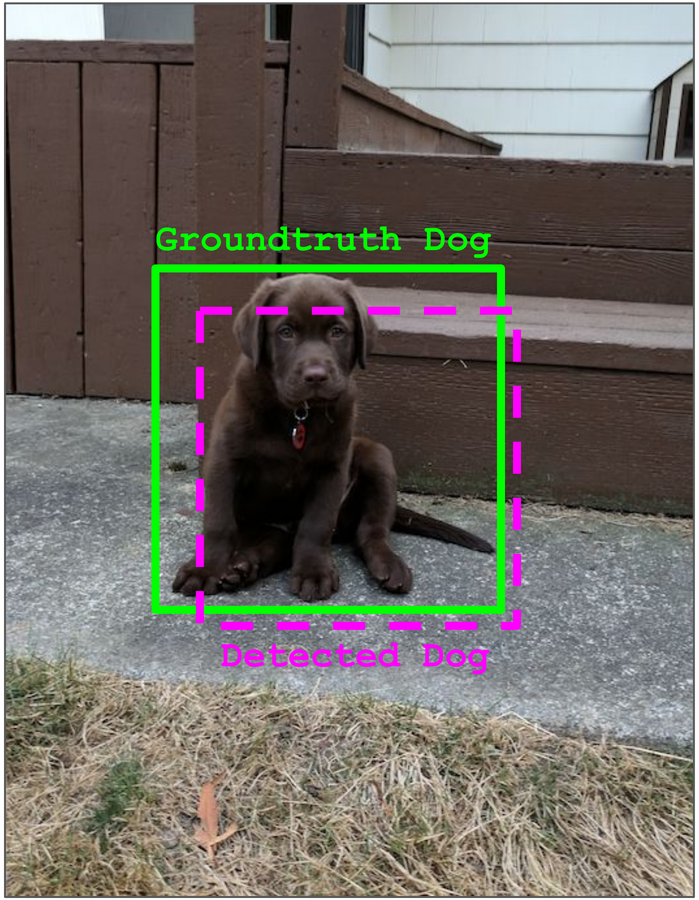
How do we know how good our model is?

cat ✓ dog ✓ cat ✗ cat ✓

Accuracy: 75%

For image classification, life is easy :)

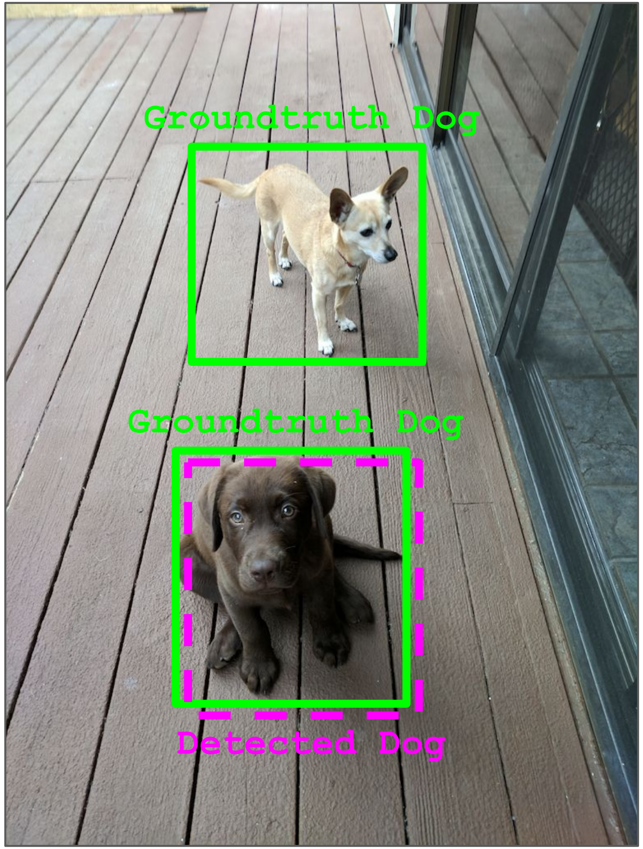
Evaluating Detectors is harder :(



Problem 1: Metrics must handle location errors

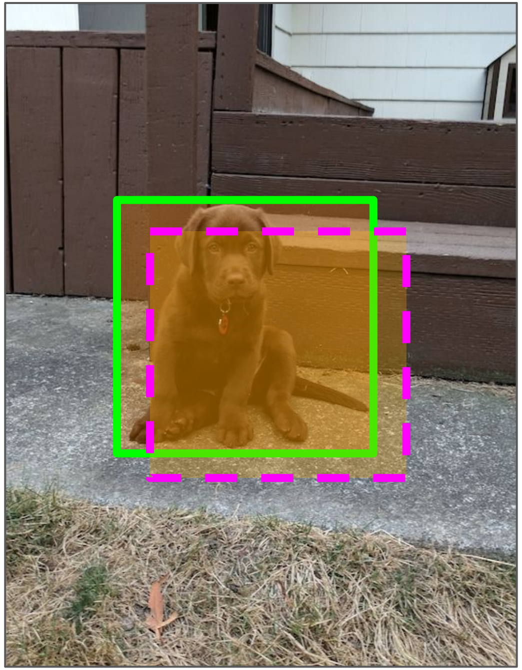
Should we consider this detection to be correct?

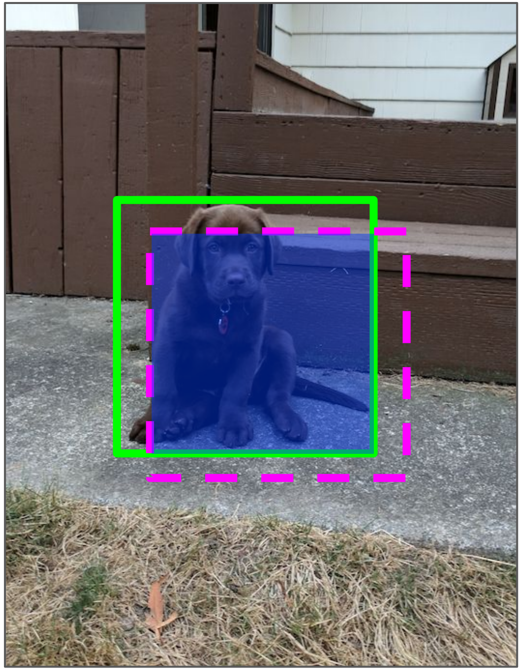
Evaluating Detectors is harder :(



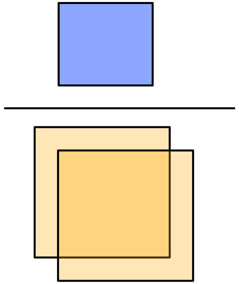
Problem 2: Metrics must account for overprediction and underprediction

Intersection over Union (IOU)



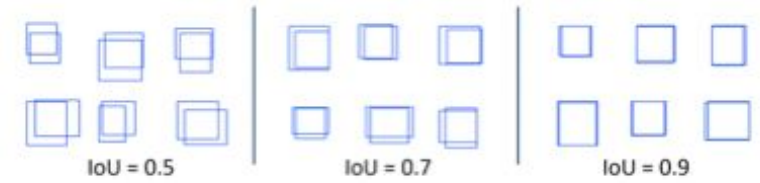


$$\mathtt{IOU} = \frac{\mathtt{Intersection}}{\mathtt{Union}}$$



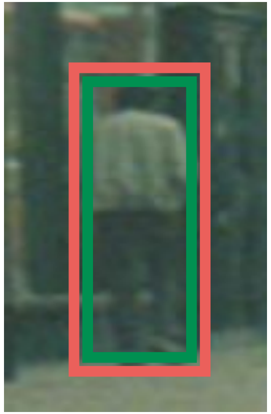
- Boxes are disjoint if and only if $\mathtt{IOU=0}$
- Boxes are identical if and only if $\mathtt{IOU=1}$

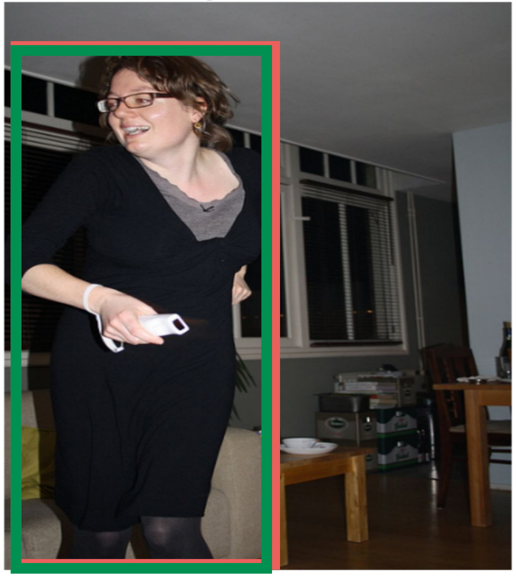
Detection is considered“correct” if $\mathtt{IOU} > \alpha$



Intersection over Union (IOU)

IoU = 0.5 **IoU = 0.7** **IoU = 0.95**

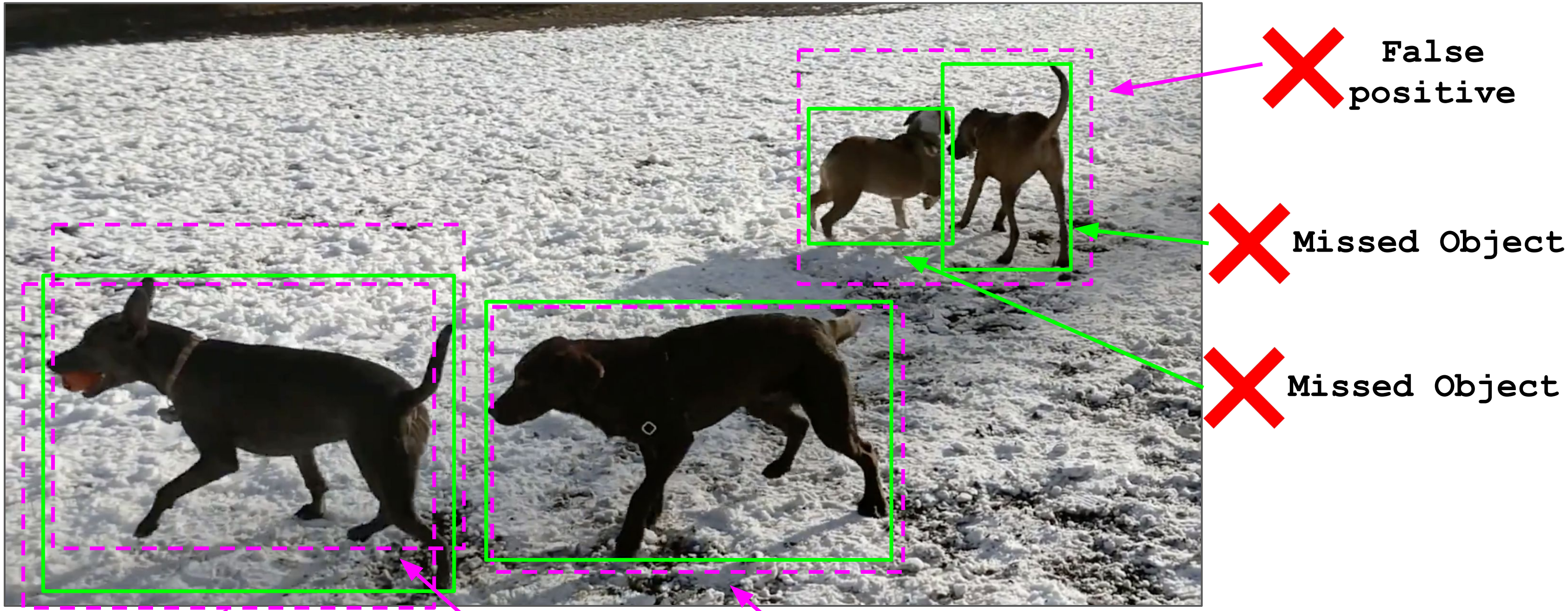




Ground-Truth BBox **Detection BBox**

Slide credit: http://image-net.org/challenges/talks/2016/ECCV2016_ilsvrc_coco_detection_segmentation.pdf

True/False Positives and Missed Objects



- Match detections and groundtruth instances based on IOU
- Count missed groundtruth objects
- Mark detections as TP or FP based on whether IOU>α

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)?

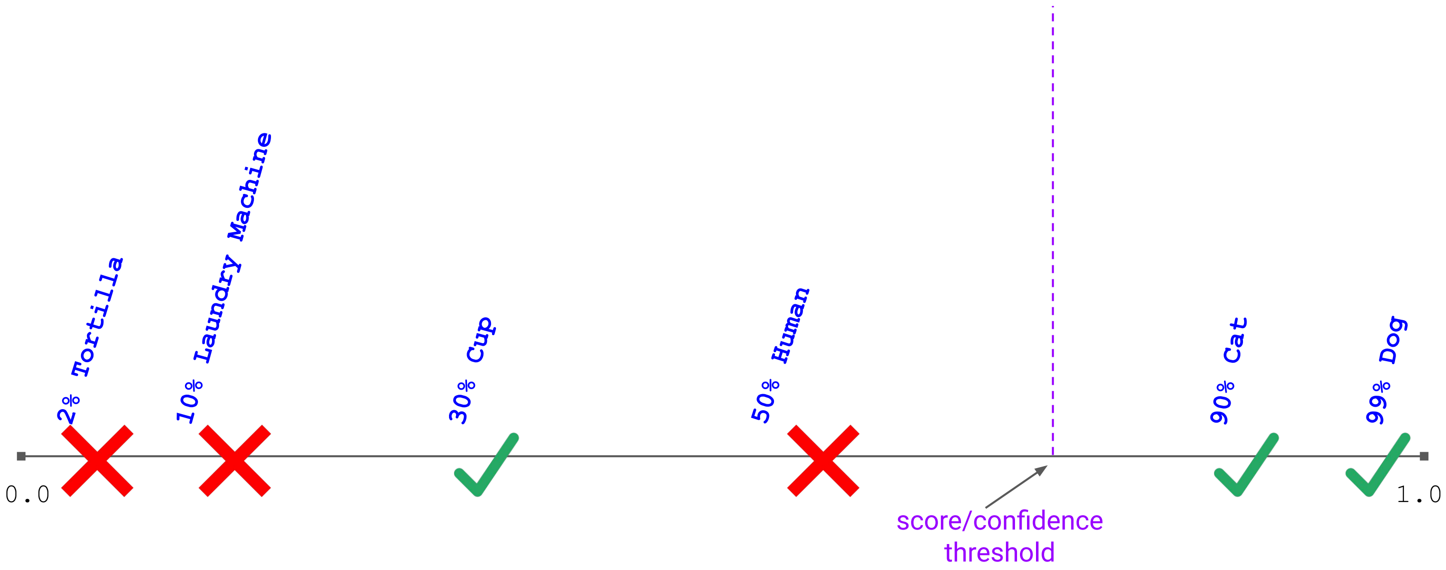
$$\text{Recall} = \frac{\#TP}{\#\text{Groundtruth Objects}}$$

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;

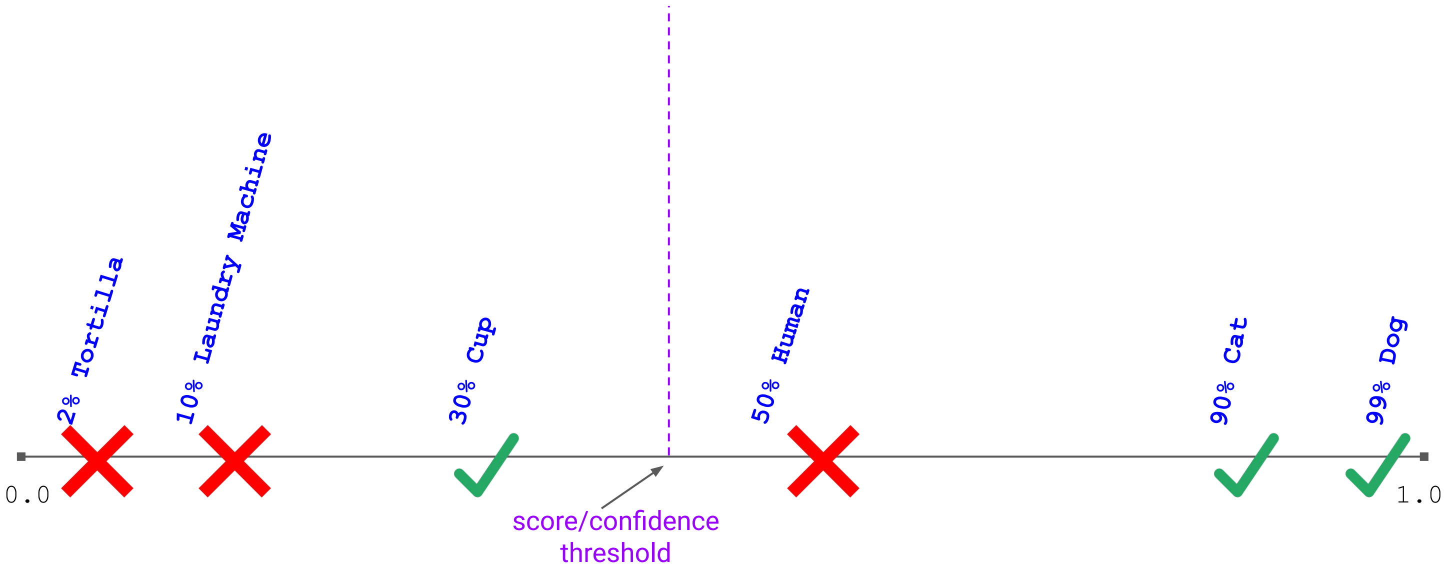
Last step of detection pipeline: *use score threshold to select final detections*



Trading off between Precision and Recall

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

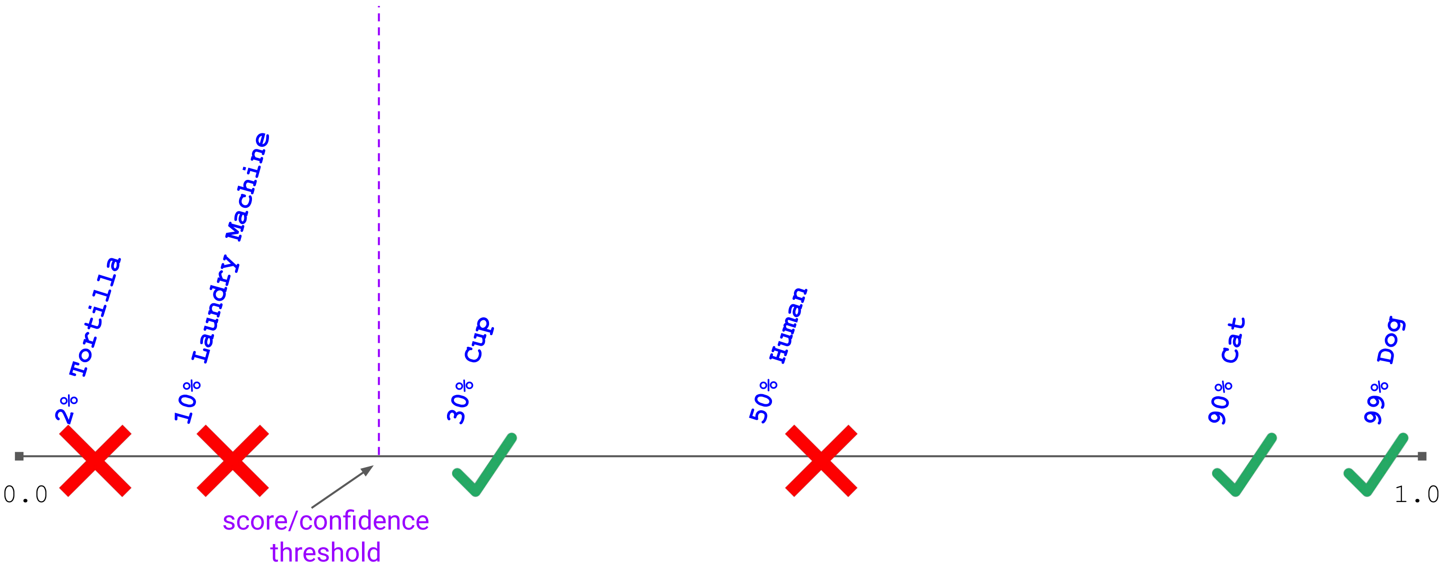
Last step of detection pipeline: *use score threshold to select final detections*



Trading off between Precision and Recall

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

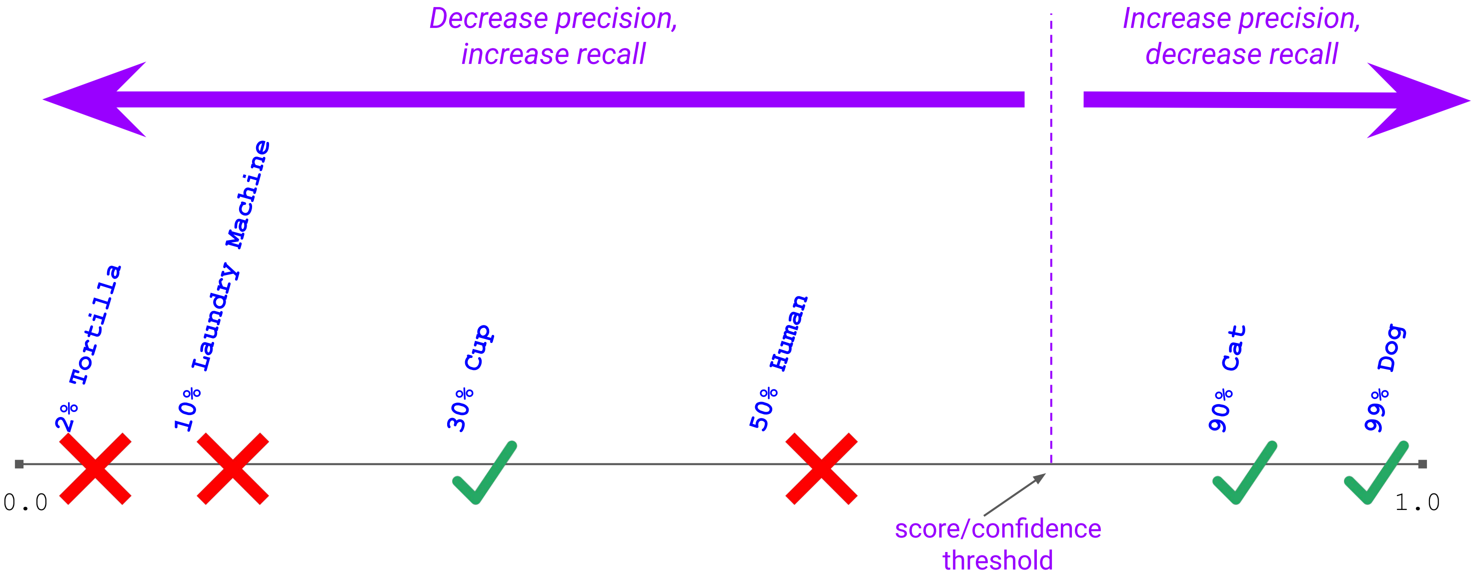
Last step of detection pipeline: *use score threshold to select final detections*



Trading off between Precision and Recall

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

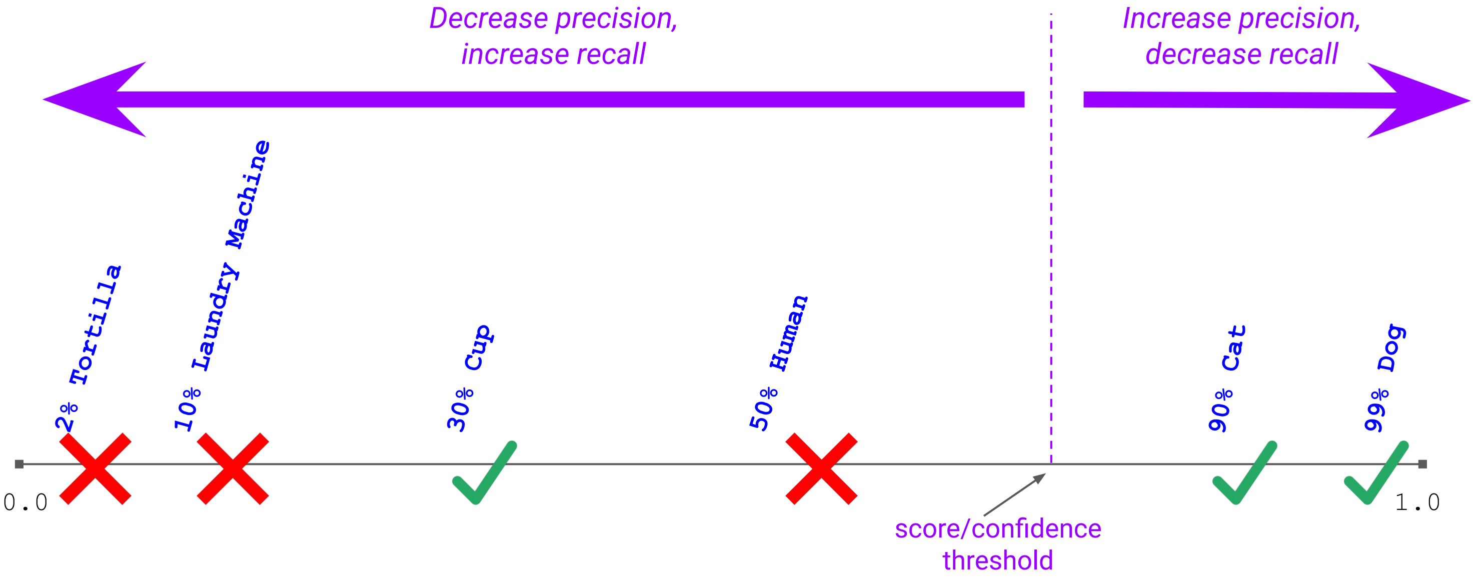
Last step of detection pipeline: *use score threshold to select final detections*



Trading off between Precision and Recall

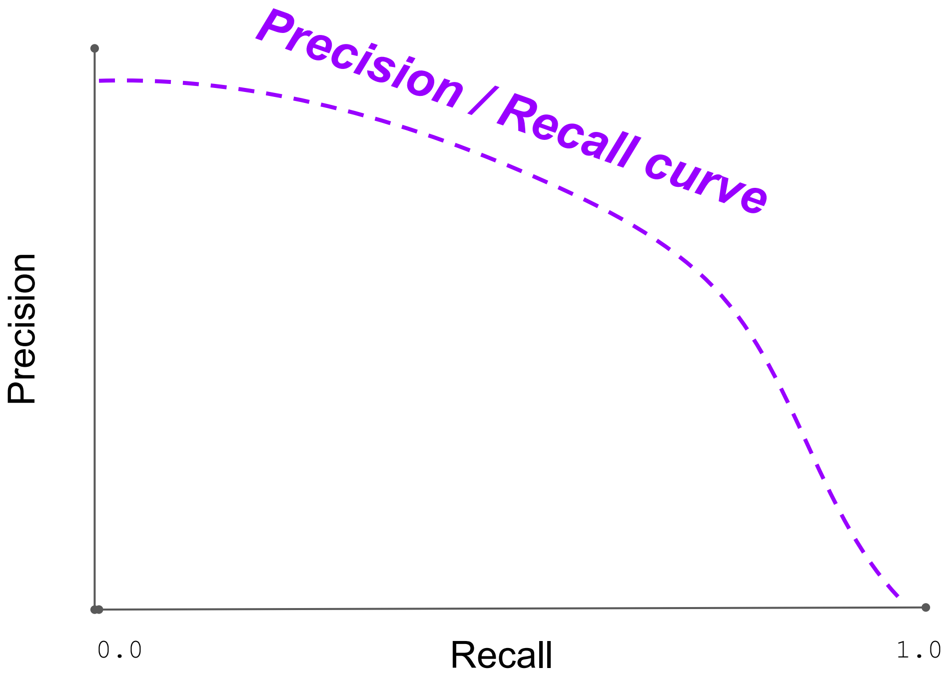
Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

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

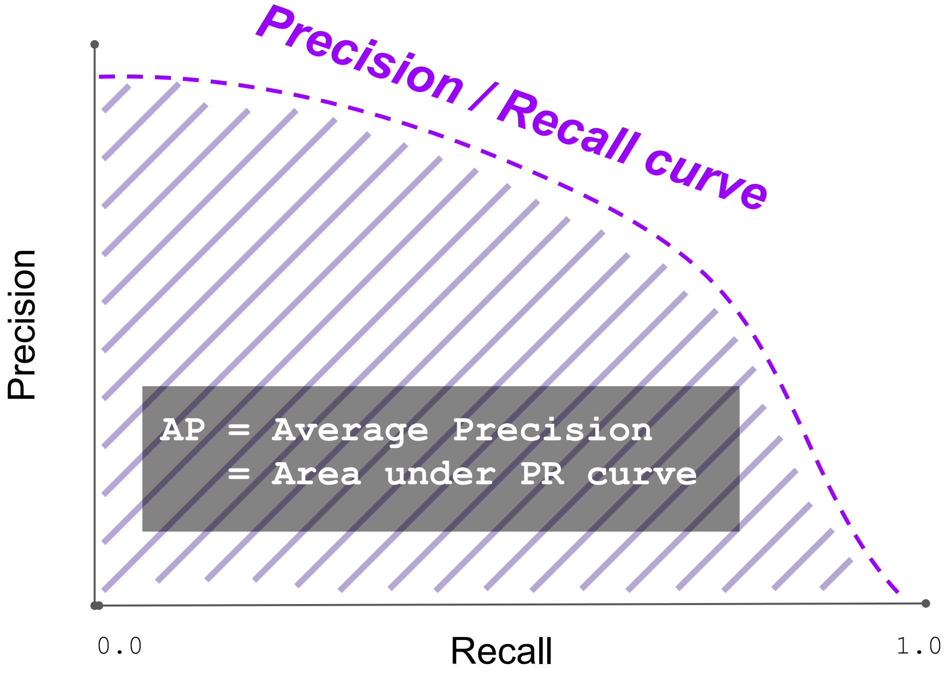


When would it be better to be on one side of this spectrum than the other?

Precision/Recall Curves and AP (Average Precision)



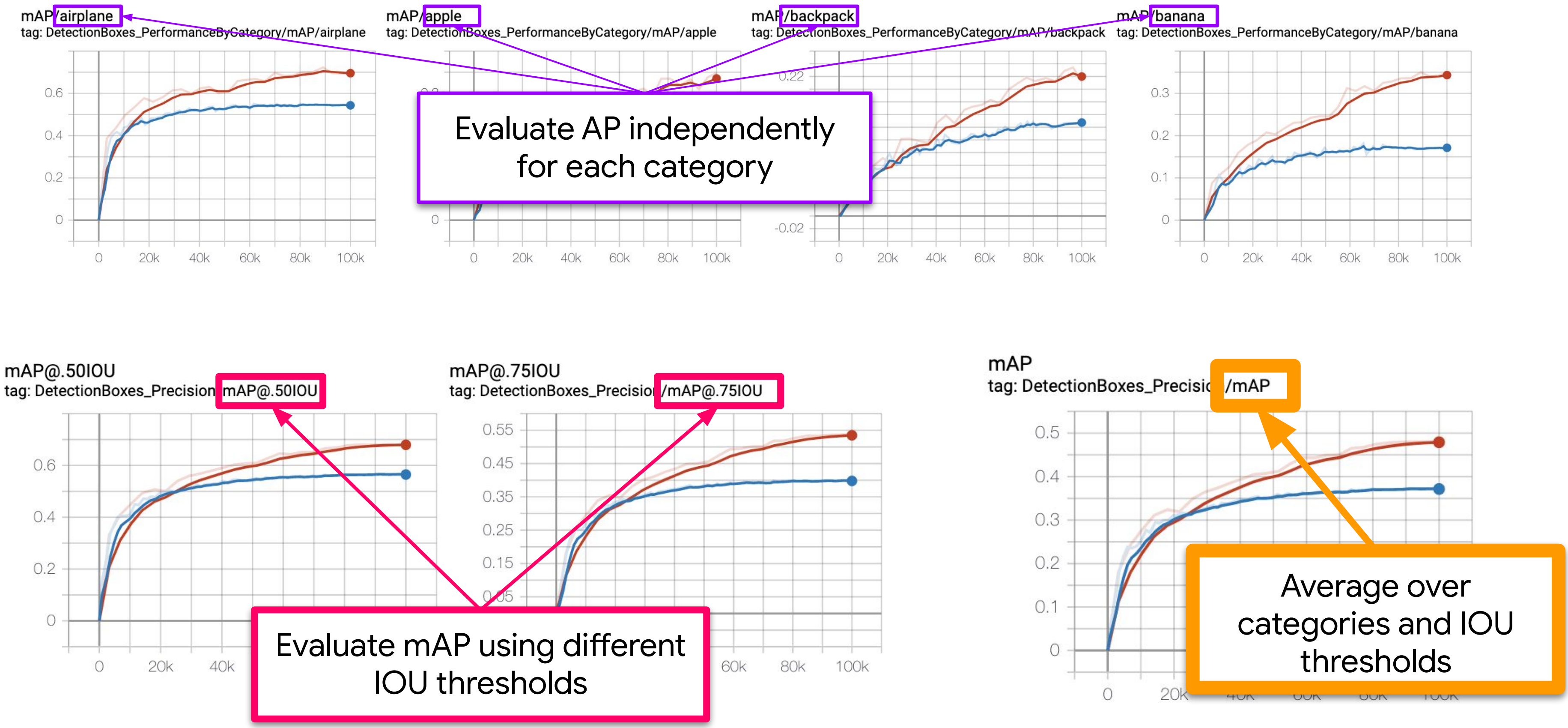
Precision/Recall Curves and AP (Average Precision)



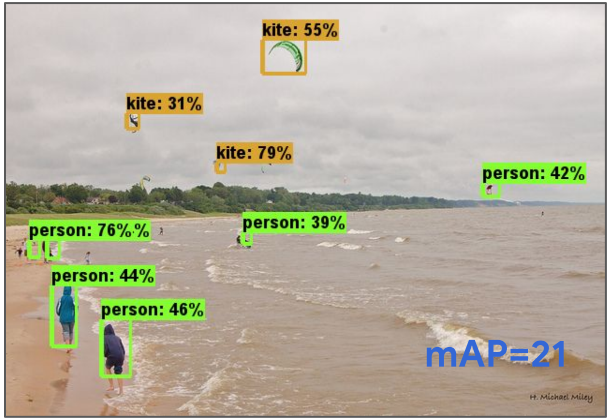
Remember:

- *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...*

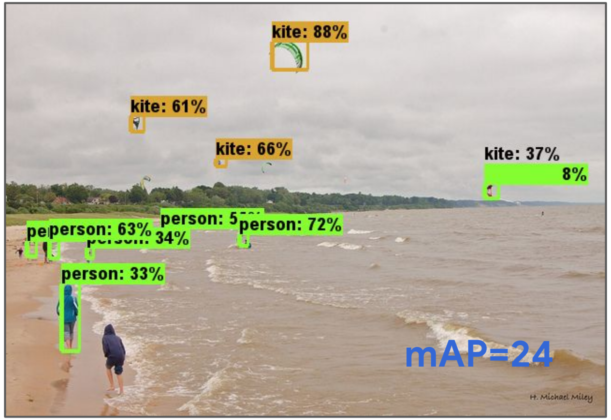
AP, mAP, “COCO (Integrated) mAP”



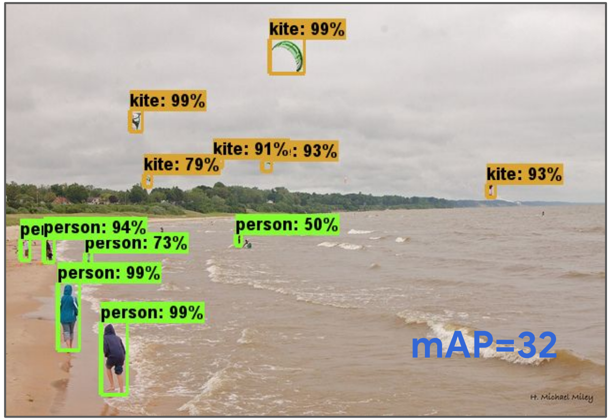
SSD w/MobileNet (Low Resolution)



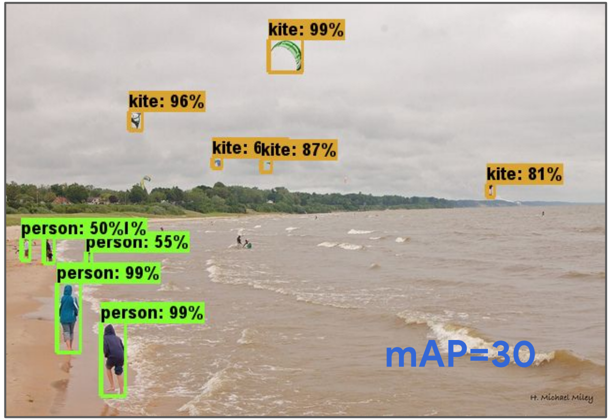
SSD w/Inception V2 (Low Resolution)



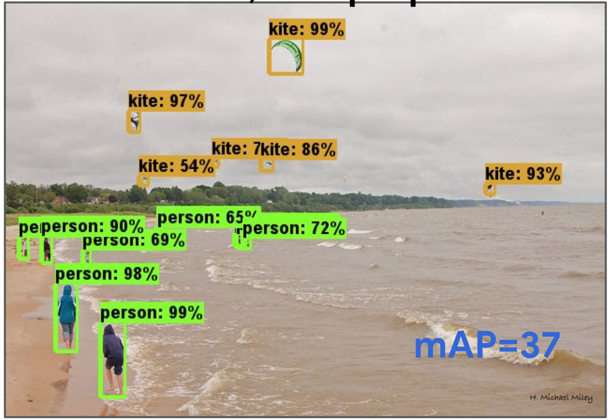
Faster R-CNN w/Resnet101, 100 proposals



RFCN w/Resnet101, 300

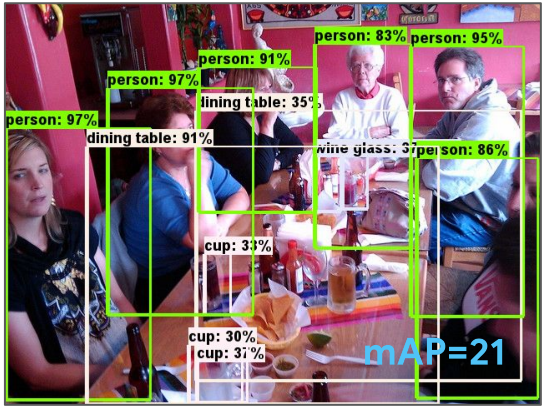


Faster R-CNN w/Inception Resnet V2, 300 proposals

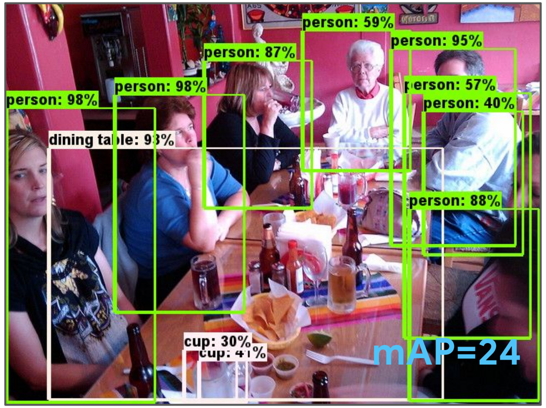


COCO

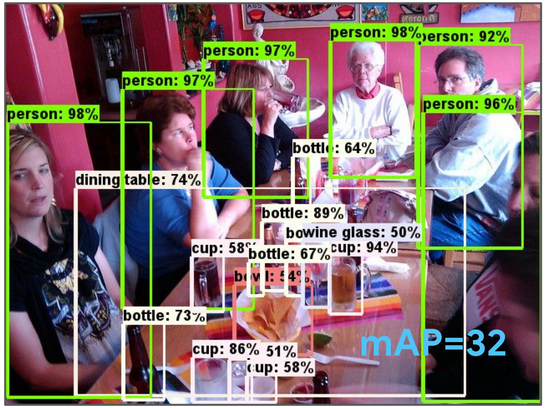
SSD w/MobileNet
(Low Resolution)



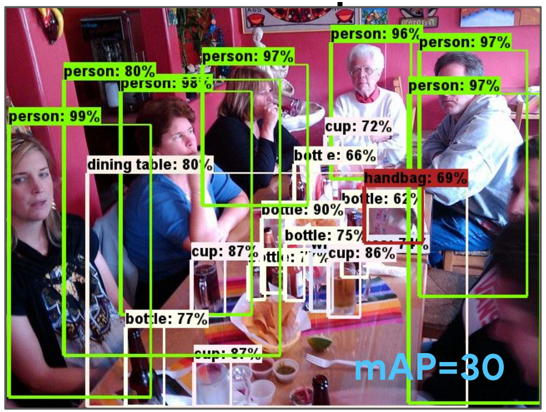
SSD w/Inception V2
(Low Resolution)



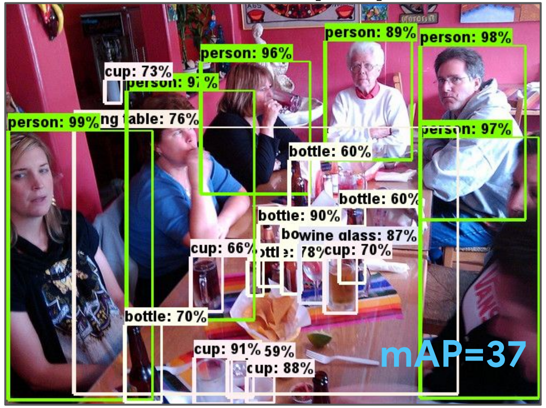
Faster R-CNN w/Resnet101,
100 proposals



RFCN w/Resnet101, 300



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



COCO

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**

Google AI

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}

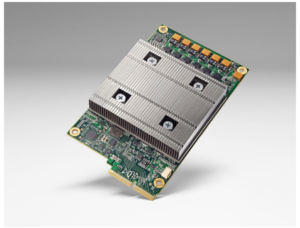
Things to consider: sensor, device, latency constraints, memory constraints

Your laptop

Datacenters

Mobile

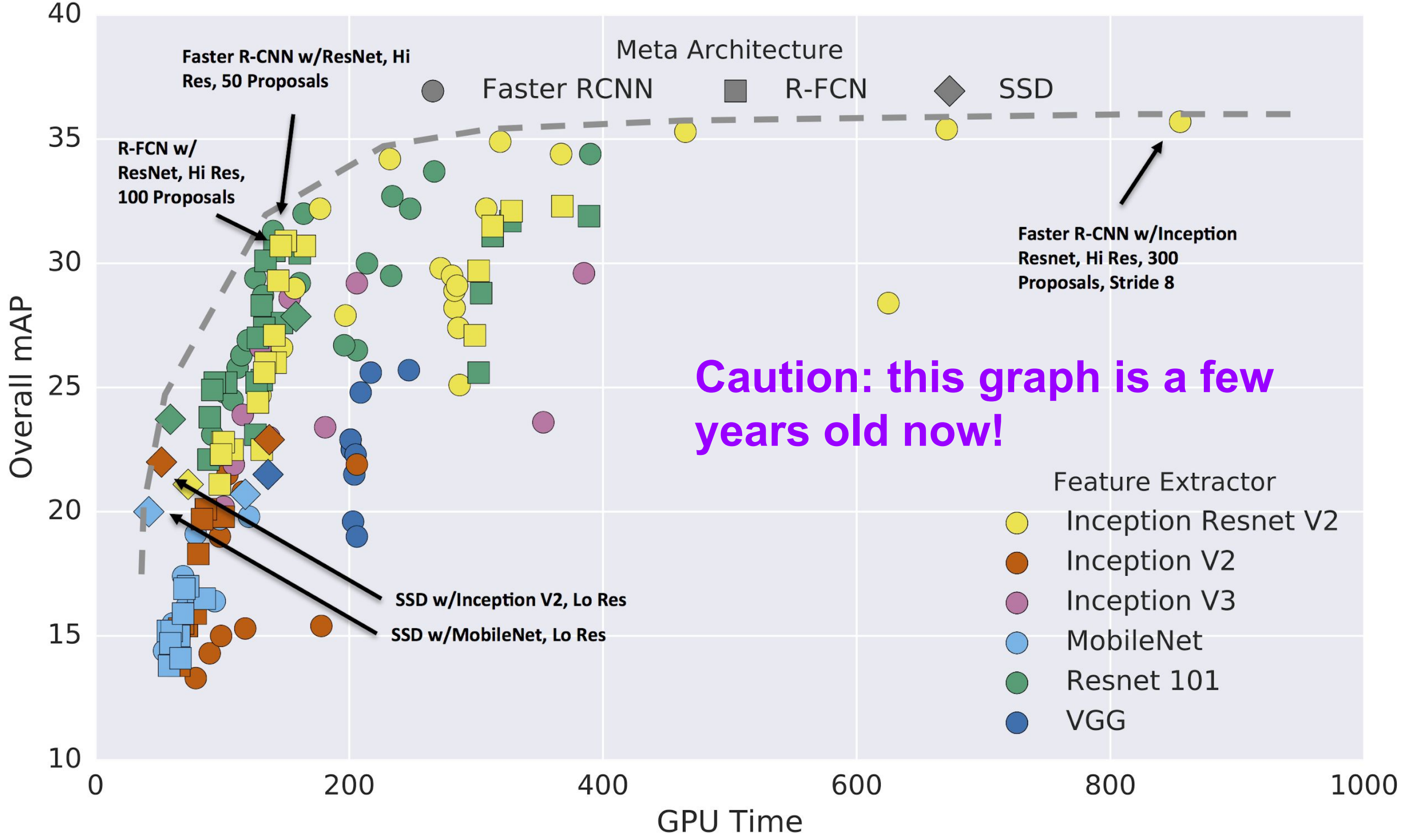
Raspberry Pi



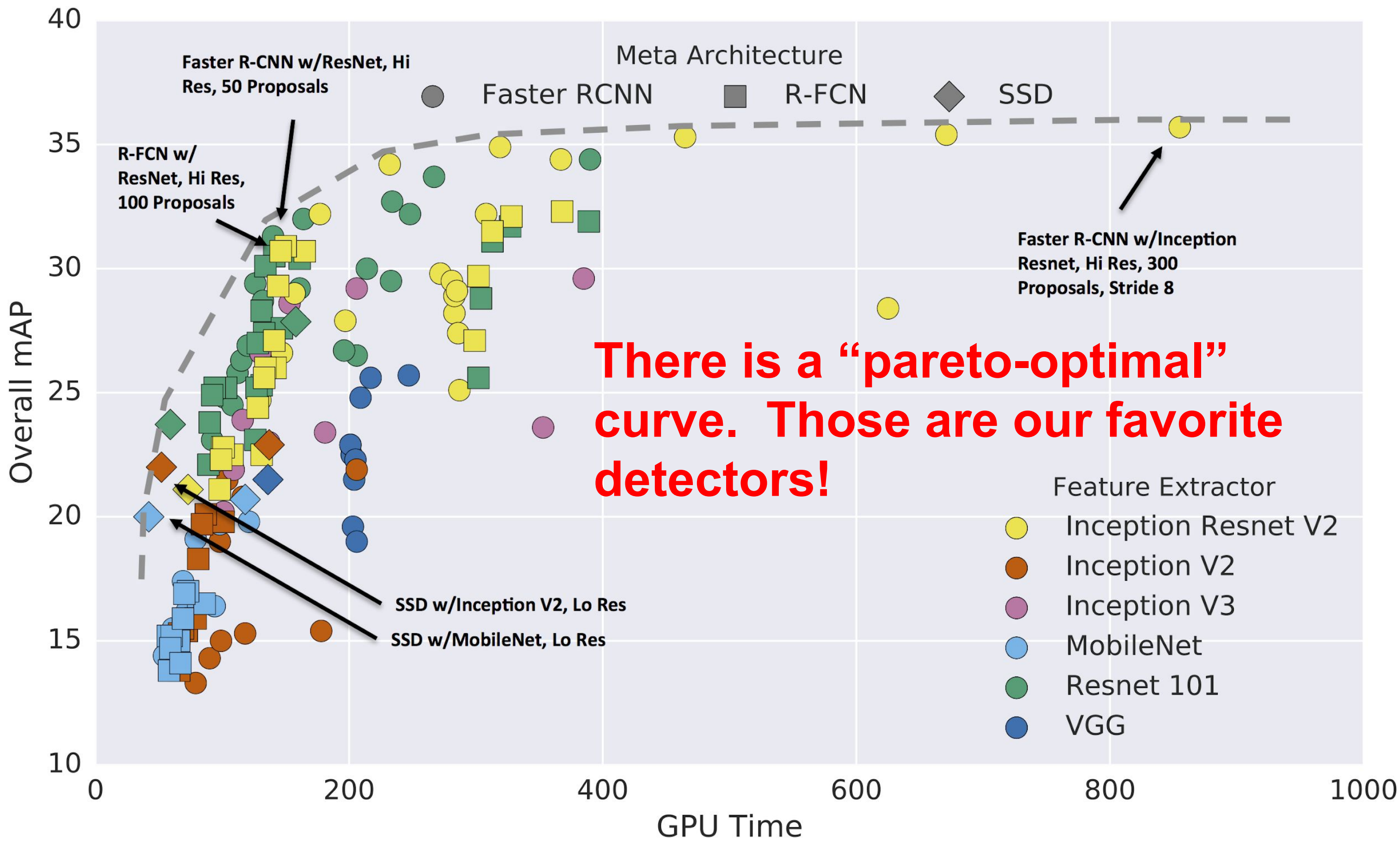
Tensor Processing Unit

Pick a point on the speed/accuracy tradeoff curve

Google AI



Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... Speed/accuracy trade-offs for modern convolutional object detectors. CVPR 2017



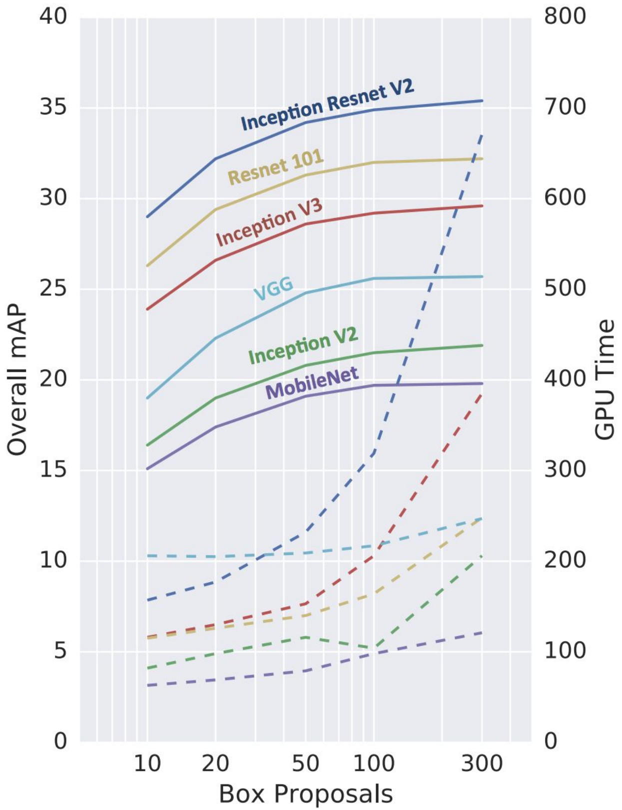
There is a "pareto-optimal" curve. Those are our favorite detectors!

Google AI

Use lower resolution images for speed

Lower resolution much faster; tends to miss smaller objects

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!

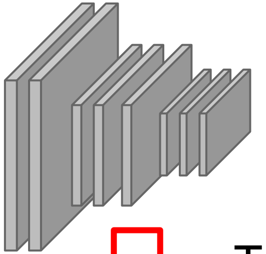
Larger images => Smaller batch sizes => Noisy batch norm statistics :(

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

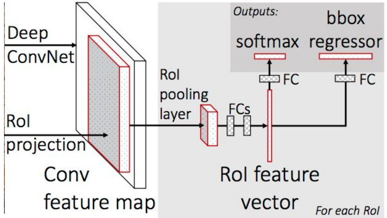


18K labels

Transfer weights

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

COCO
Common Objects in Context



Detections

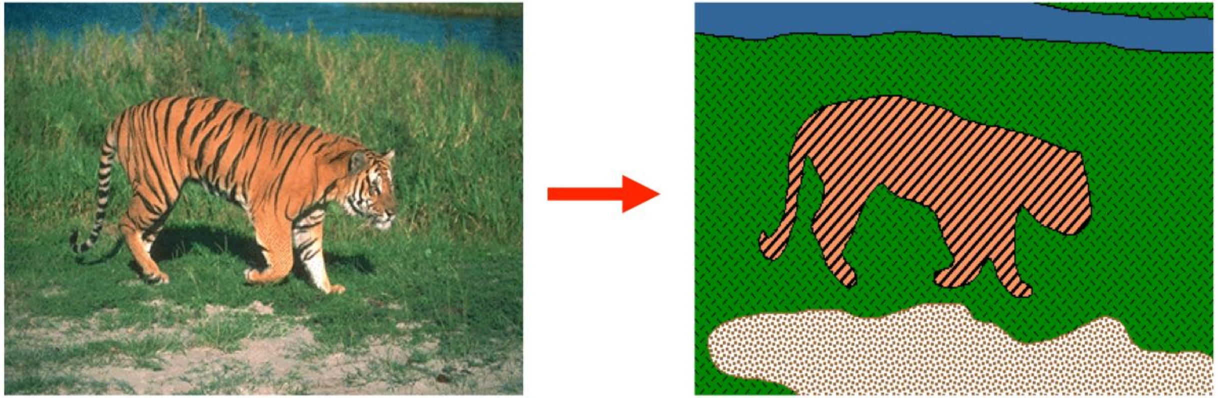
Method	mAP@0.5	mAP@[0.5,0.95]
He *et al.* [16]	53.3	32.2
ImageNet	53.6	34.3
300M	56.9	36.7
ImageNet+300M	**58.0**	**37.4**
Inception ResNet [37]	56.3	35.5

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

Image credit: https://ai.stanford.edu/~syyeung/cvweb/tutorial3.html