Real-time Object Detection

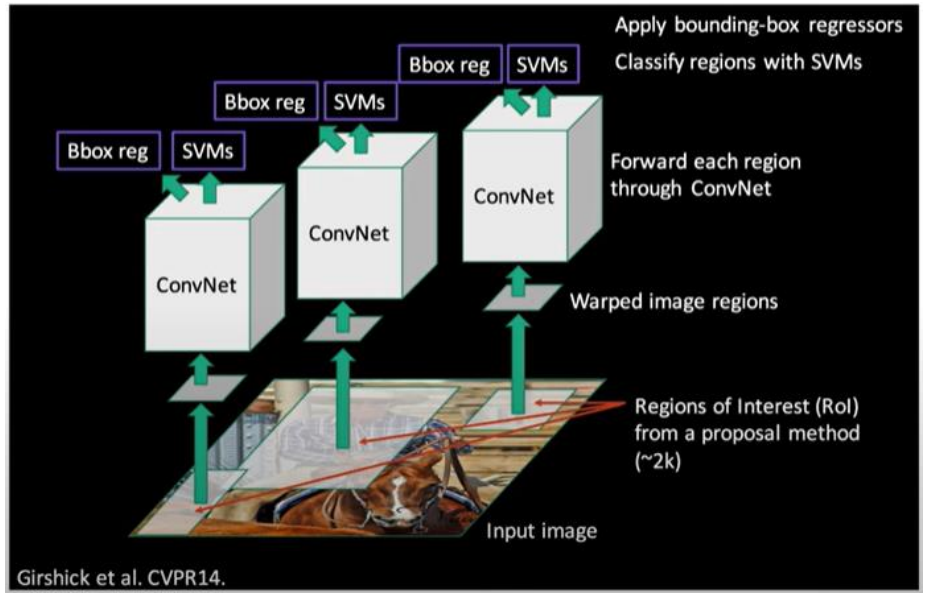
한양대학교

컴퓨터소프트웨어학부

박현준

Review

1. Bounding-box regression & SVM
   : **SVM classification → Bbox reg**
   To show Bbox reg and SVM is separate operation



2. Faster R-CNN Loss Function

$$L(\{P_i\},\{t_i\}) = \boxed{\frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*)} + \lambda \boxed{\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)}$$

<Classification> <Regression>

$$L_{RPN} = L_{cls}^{RPN} + L_{reg}^{RPN}$$

$$L_{FRCN} = L_{cls}^{FRCN} + L_{reg}^{FRCN}$$

$$\Rightarrow L = L_{RPN} + L_{FRCN}$$

Review

3. #Region Proposal of RPN

At each sliding-window location, we simultaneously predict multiple region proposals, where the number of maximum region proposals for each location is denoted as k

Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

train-time region proposals		**test-time region proposals**		
method	# boxes	method	# proposals	mAP (%)
SS	2000	SS	2000	58.7
EB	2000	EB	2000	58.6
RPN+ZF, shared	2000	RPN+ZF, shared	300	**59.9**
ablation experiments follow below				
RPN+ZF, unshared	2000	RPN+ZF, unshared	300	58.7
SS	2000	RPN+ZF	100	55.1
SS	2000	RPN+ZF	300	56.8
SS	2000	RPN+ZF	1000	56.3
SS	2000	RPN+ZF (no NMS)	6000	55.2
SS	2000	RPN+ZF (no *cls*)	100	44.6
SS	2000	RPN+ZF (no *cls*)	300	51.4
SS	2000	RPN+ZF (no *cls*)	1000	55.8
SS	2000	RPN+ZF (no *reg*)	300	52.1
SS	2000	RPN+ZF (no *reg*)	1000	51.3
SS	2000	RPN+VGG	300	59.2

Contents

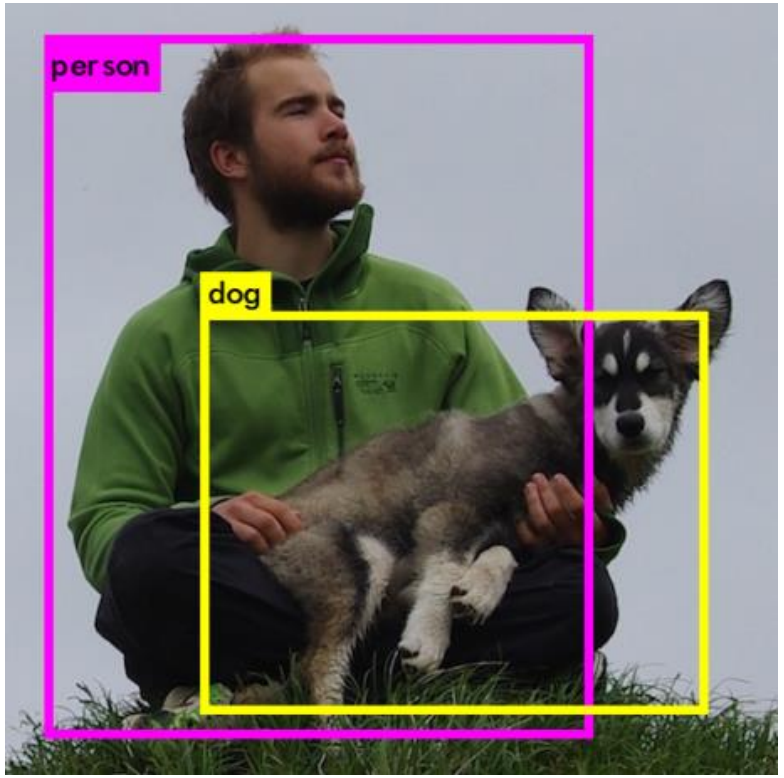
YOLOv1 : Unified Real-Time Object Detection

SSD : Single Shot MultiBox Detector

YOLOv2 : Better, Faster, Stronger

YOLOv3 : An Incremental Improvement

Why?



Joseph Redmond

Ali Farhadi

In YOLOv3…

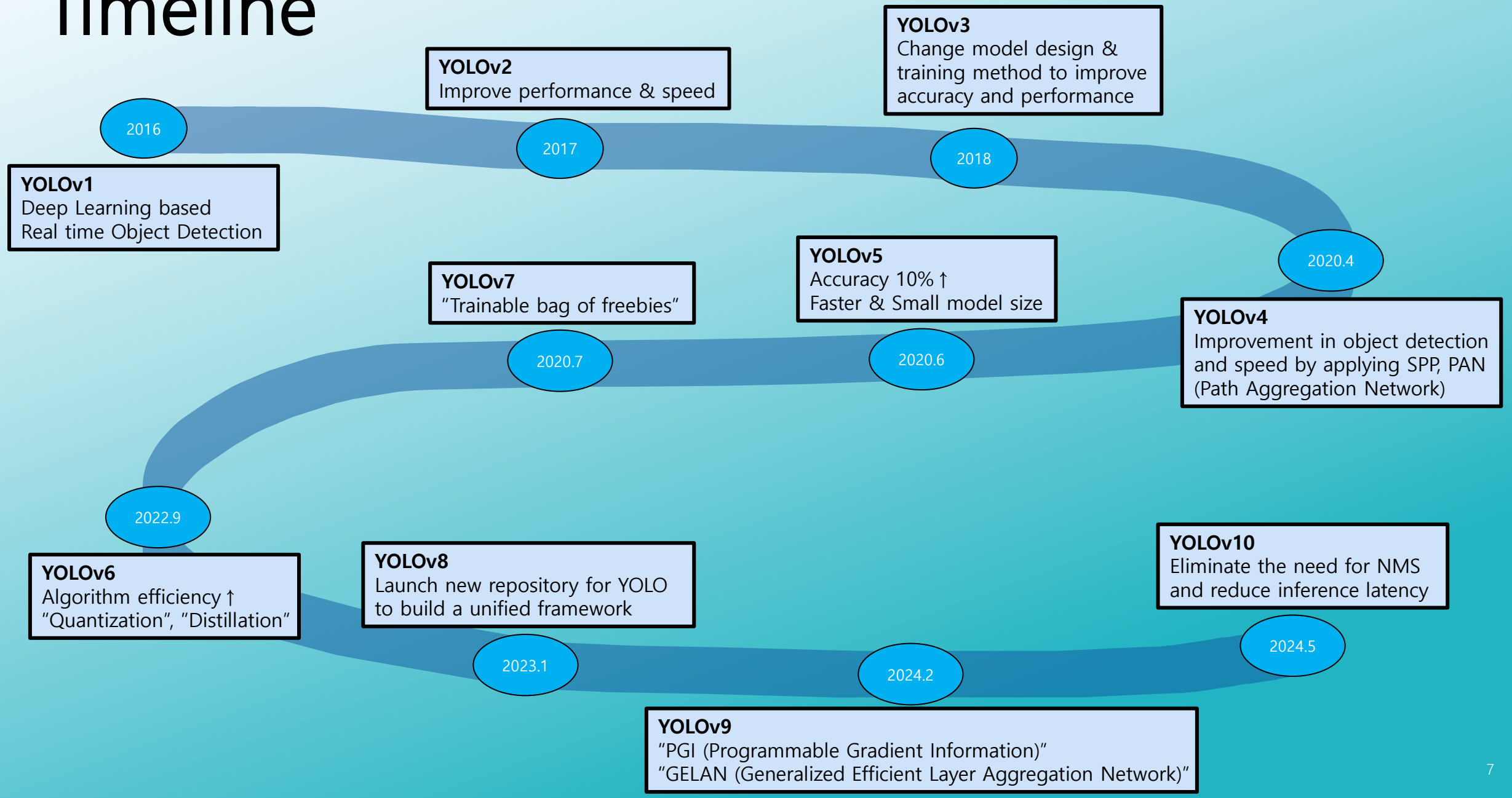
But maybe a better question is: "What are we going to do with these detectors now that we have them?" A lot of the people doing this research are at Google and Facebook. I guess at least we know the technology is in good hands and definitely won't be used to harvest your personal information and sell it to.... wait, you're saying that's exactly what it will be used for?? Oh.

Well the other people heavily funding vision research are the military and they've never done anything horrible like killing lots of people with new technology oh wait.....[1]

I have a lot of hope that most of the people using computer vision are just doing happy, good stuff with it, like counting the number of zebras in a national park [13], or tracking their cat as it wanders around their house [19]. But computer vision is already being put to questionable use and as researchers we have a responsibility to at least consider the harm our work might be doing and think of ways to mitigate it. We owe the world that much.

In closing, do not @ me. (Because I finally quit Twitter).

Timeline


2016
YOLOv1
Deep Learning based
Real time Object Detection
2017
YOLOv2
Improve performance & speed
2018
YOLOv3
Change model design &
training method to improve
accuracy and performance
2020.4
YOLOv4
Improvement in object detection
and speed by applying SPP, PAN
(Path Aggregation Network)
2020.6
YOLOv5
Accuracy 10% ↑
Faster & Small model size
2020.7
YOLOv7
"Trainable bag of freebies"
2022.9
YOLOv6
Algorithm efficiency ↑
"Quantization", "Distillation"
2023.1
YOLOv8
Launch new repository for YOLO
to build a unified framework
2024.2
YOLOv9
"PGI (Programmable Gradient Information)"
"GELAN (Generalized Efficient Layer Aggregation Network)"
2024.5
YOLOv10
Eliminate the need for NMS
and reduce inference latency

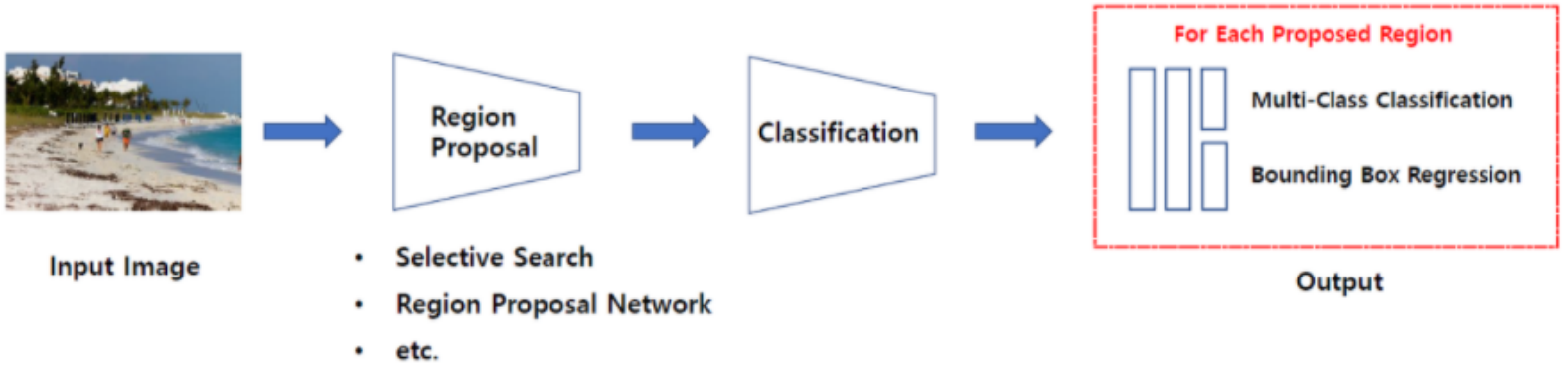
YOLOv1

: Unified, Real-Time Object Detection

2016.5.9 CVPR

Introduction

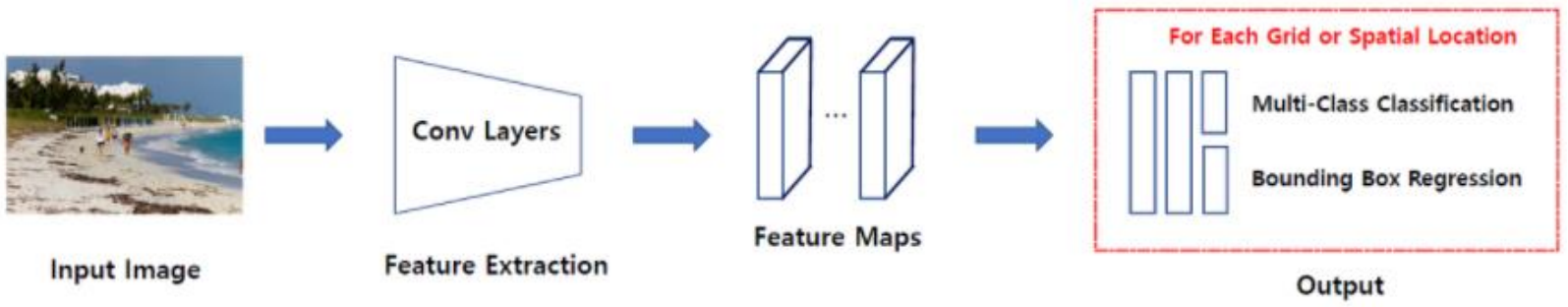
R-CNN



Detection ↑
Inference ↓

2-stage : Localization + Classification

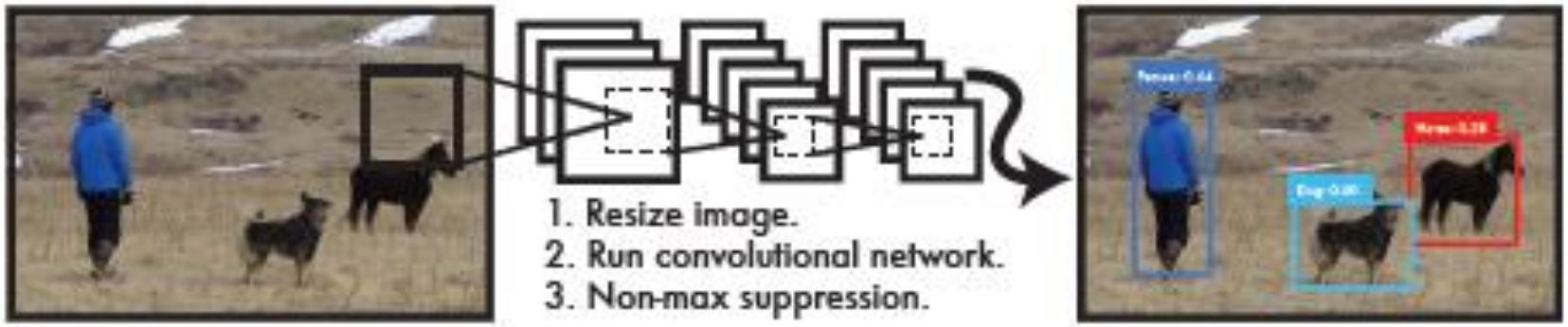
YOLO



Detection ↓
Inference ↑

1-stage : Feature Extraction

Introduction



"A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes"

Whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance while maintaining high average precision.

Unified Detection

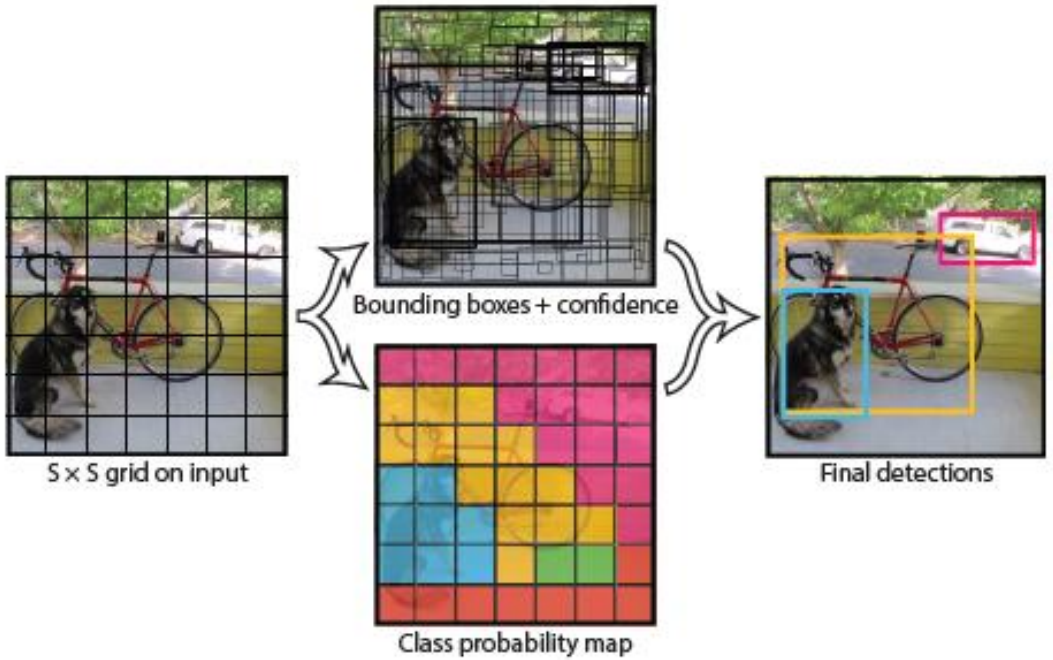


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Unified separate components into single neural network

Formally,

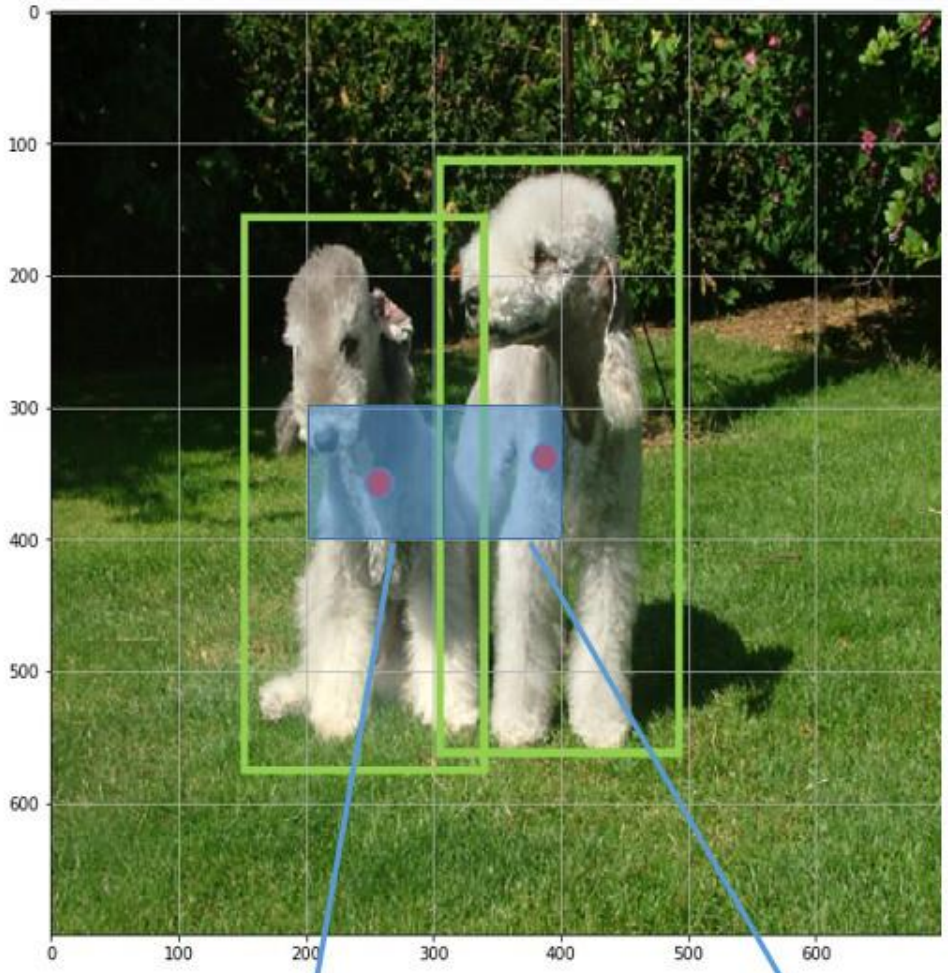
$$\text{Confidence} = \Pr(Object) * IoU_{Pred}^{Truth}$$

In YOLO,

$$\Pr(Class_i|Object) * \Pr(Object) * IoU_{Pred}^{Truth}$$

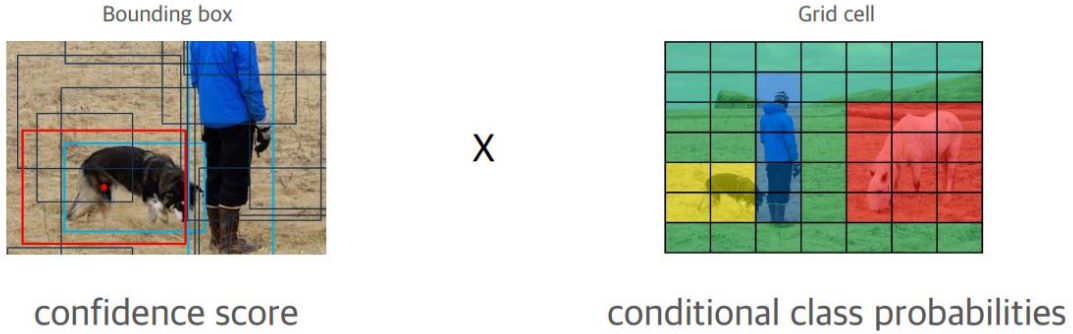
$$= \Pr(Class_i) * IoU_{Pred}^{Truth}$$

Unified Detection



(0.67, 0.6, 1.98, 4.01)

(0.43, 0.88, 1.90, 3.81)



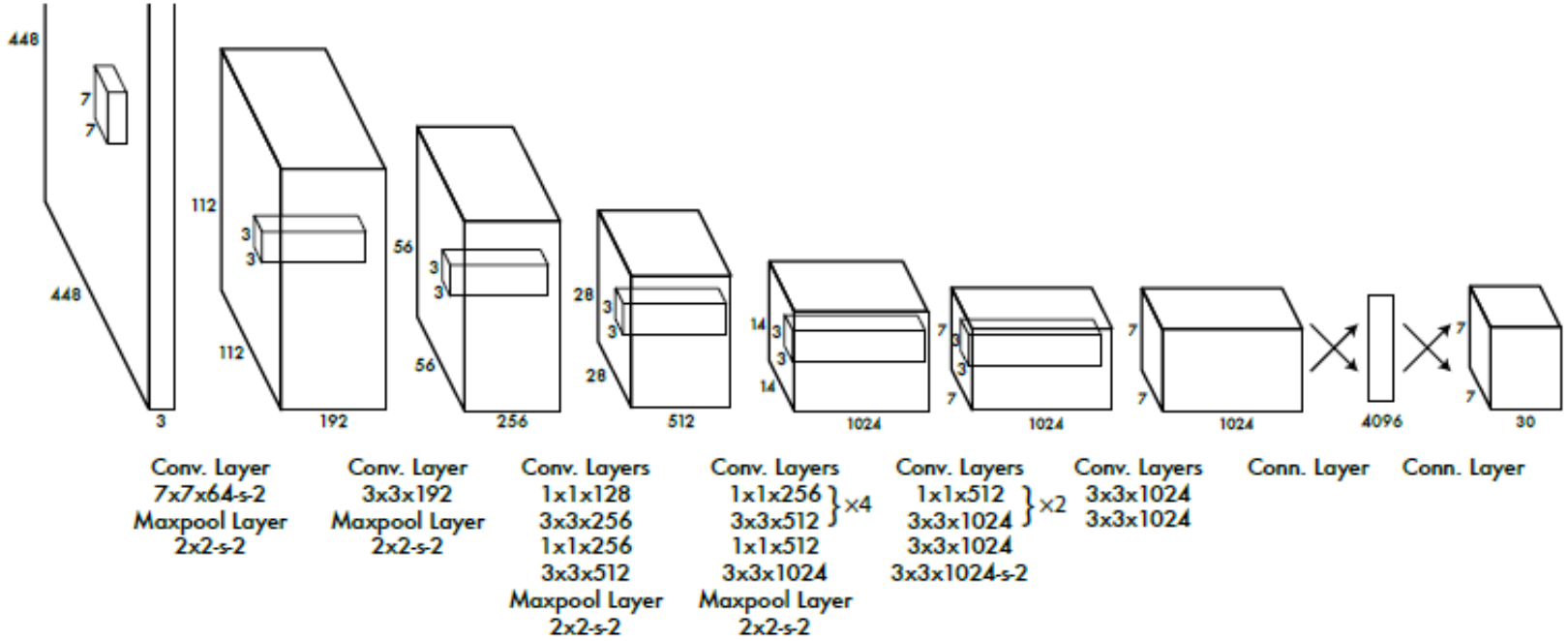
Predict $x,\ y,\ w,\ h,\ confidence$

PASCAL VOC : $S = 7,\ B = 2,\ C = 20$

$(S = grid\ size, B = \#bbox)$

$\Rightarrow 7 * 7 * 30$ tensor

Design (Darknet)



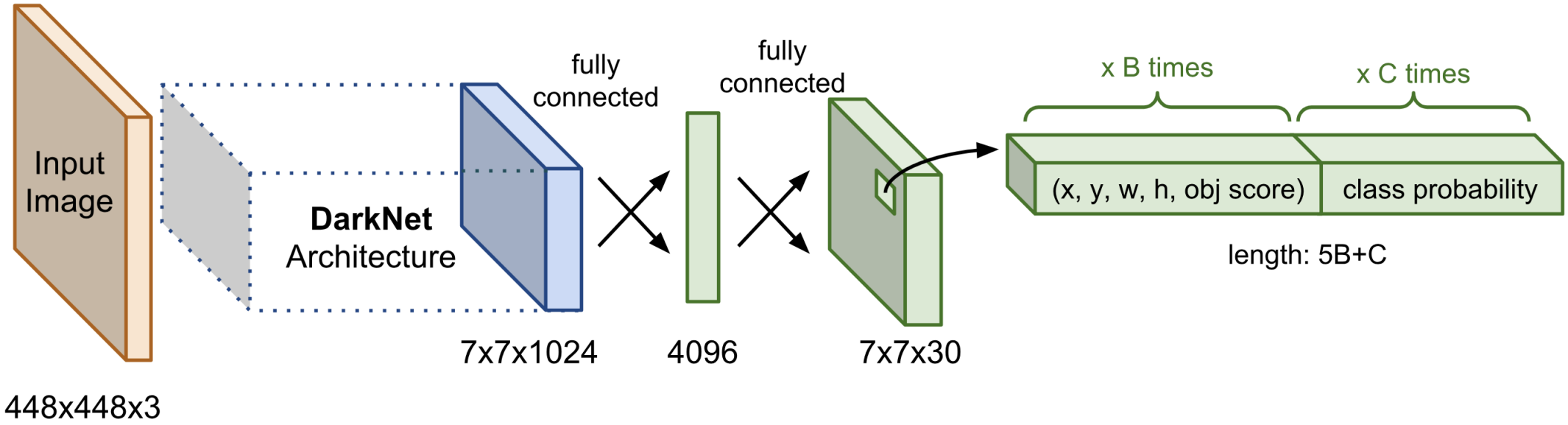
Inspired by **GoogLeNet** for image classification

- 24 Convolutional layers
- 2 fully-connected layers
- 1x1 reduction layers followed by 3x3 conv layers
- Leaky ReLU

- Leaky ReLU

$$\phi(x) = \begin{cases} x, & x > 0 \\ 0.1x, & otherwise \end{cases}$$

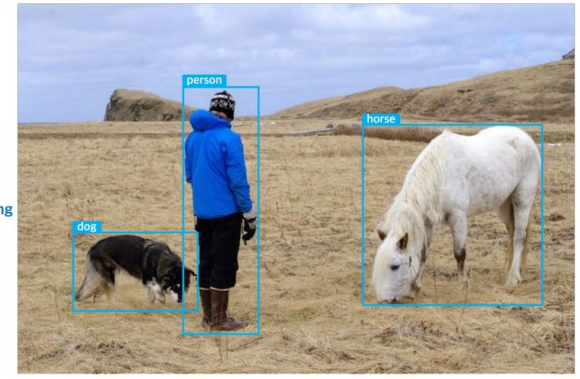
Design (Darknet)



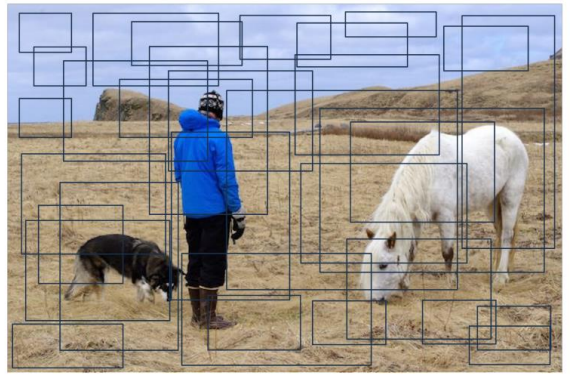
Procedure

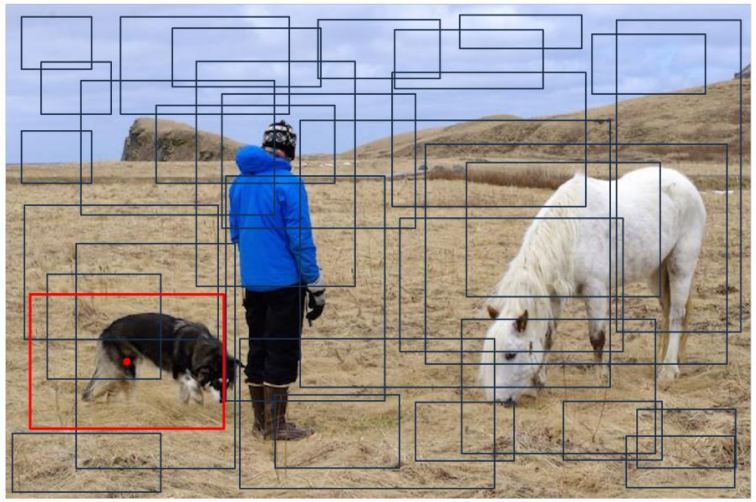
Image

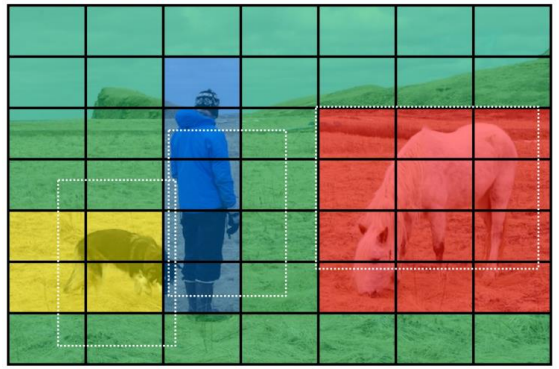
Labeling

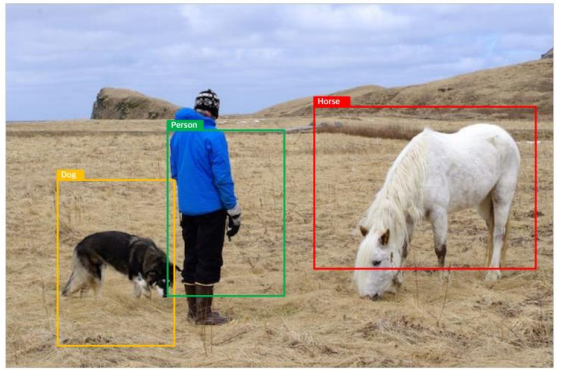


Ground Truth Box

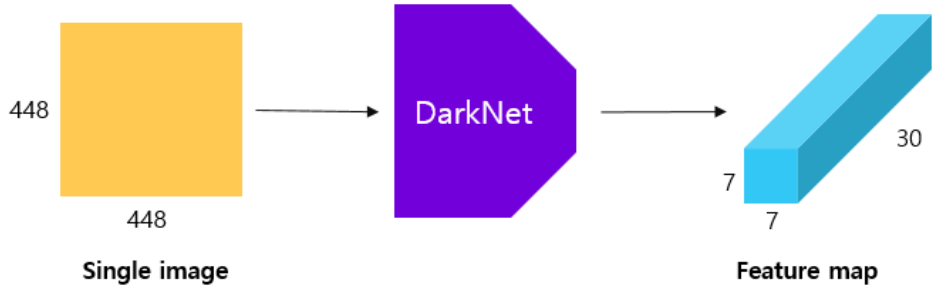








Training



- **Pre-training**
  : ImageNet 1000-class dataset
  (20 conv layers followed by average-pooling layer & fc layer)
- **Validation**
  : ImageNet 2012 validation-set → 88% accuracy
- **Framework** : **Darknet**

- Add 4 conv layers & 2 fc layers with randomly initialized weights for detection
- Input resolution : 224x224 → 448x448 (for fine-grained visual info)

Training

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Regression Loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2$$

$$+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2$$

Confidence Loss

$$+ \sum_{i=0}^{S^2} \mathbb{I}_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

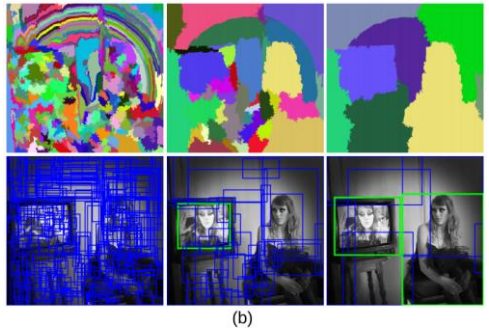
Classification Loss

$\lambda_{coord} = 5$

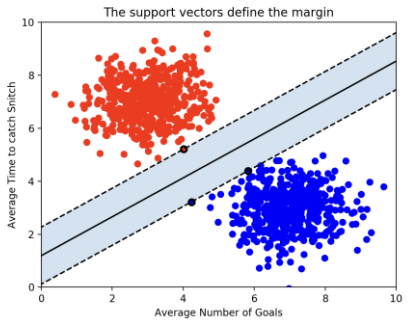
$\lambda_{noobj} = 0.5$

YOLO vs R-CNN

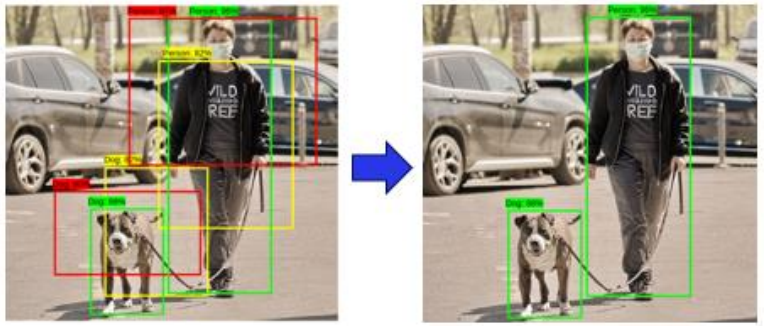
R-CNN



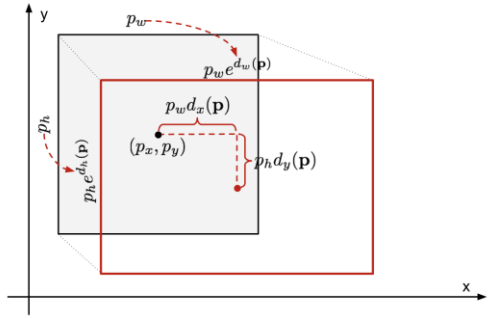
Selective Search



SVM



Non-Maximum Suppression



Bounding-box regression

YOLO

= Each grid cell proposes potential bboxes and scores those boxes using convolutional features

≠ Put spatial constraints on the grid cell proposals → **98** region proposals per image

YOLO vs Fast R-CNN

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Table 1: Real-Time Systems on PASCAL VOC 2007. Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

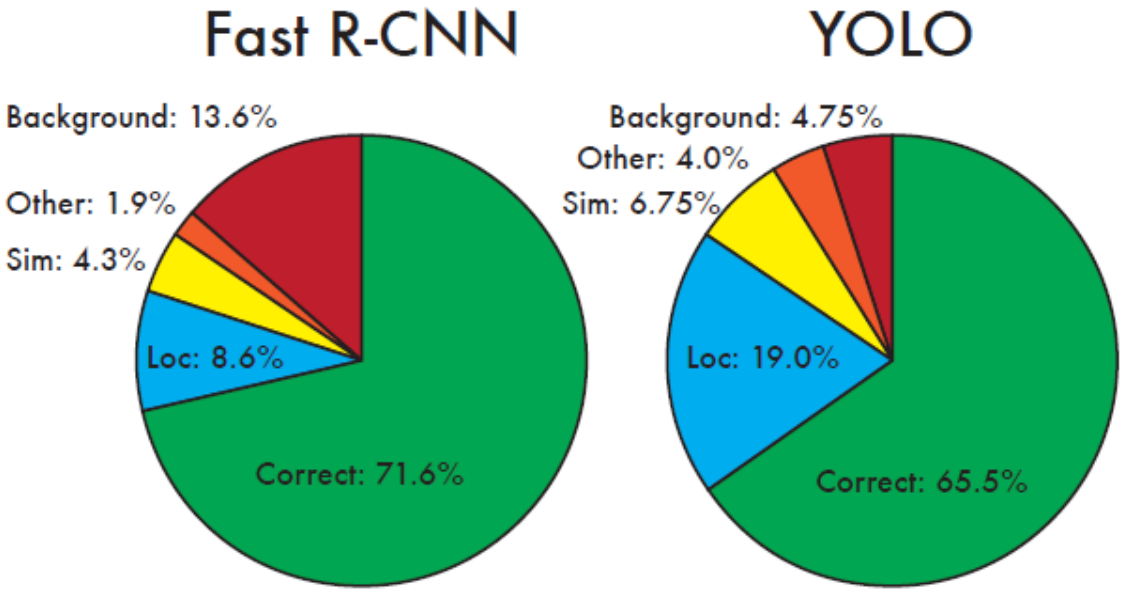


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

YOLO + Fast R-CNN

Background detection : **YOLO >>>> Fast R-CNN**

For every bounding box that R-CNN predicts → Check if YOLO predicts a similar box

	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	**66.9**	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	**75.0**	**3.2**

Table 2: Model combination experiments on VOC 2007. We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.

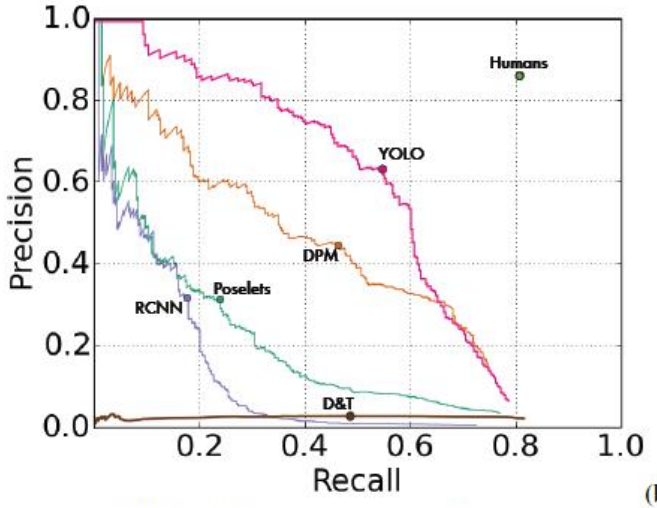
VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Table 3: PASCAL VOC 2012 Leaderboard. YOLO compared with the full comp4 (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.

YOLO + Fast R-CNN = No benefit from the speed of YOLO

∵ **Run model separately and combine the results**

Result



(a) Picasso Dataset precision-recall curves.

	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Figure 5: Generalization results on Picasso and People-Art datasets.

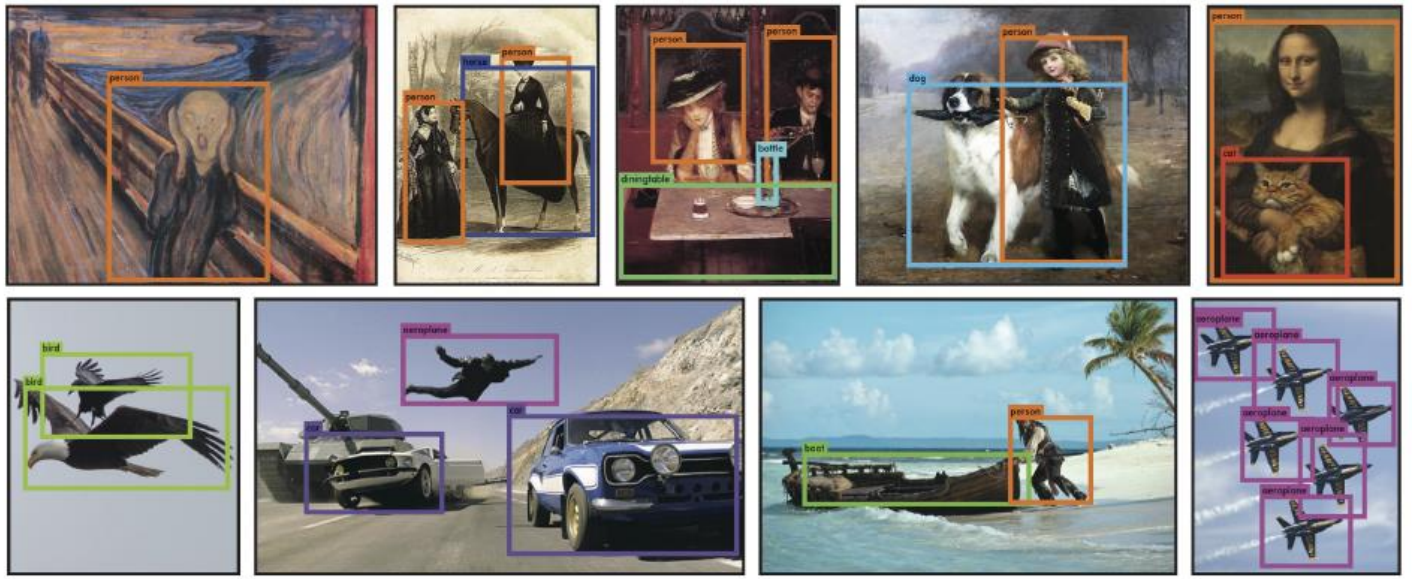


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

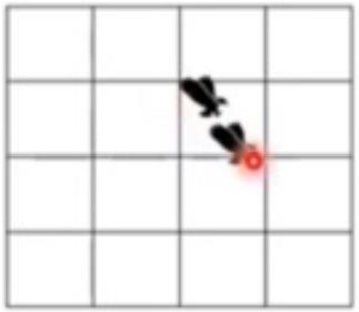
Conclusion

YOLO

Weakness

- Each grid cell predicts only one class.

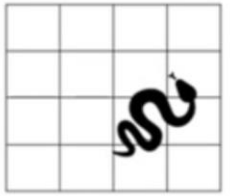
  If objects are overlapped, they cannot be predicted properly.



- Since the bounding box form is learned through data,

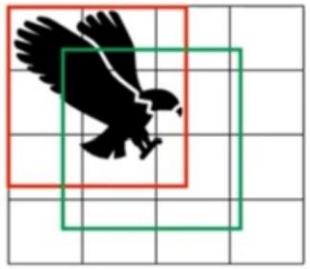
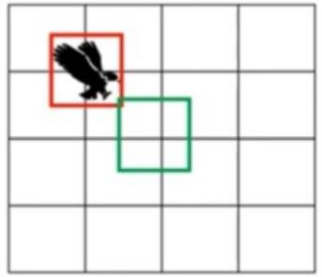
  it cannot be accurately predicted in the case of a

  new and unique type of bounding box.


- Loss function treats errors the same,

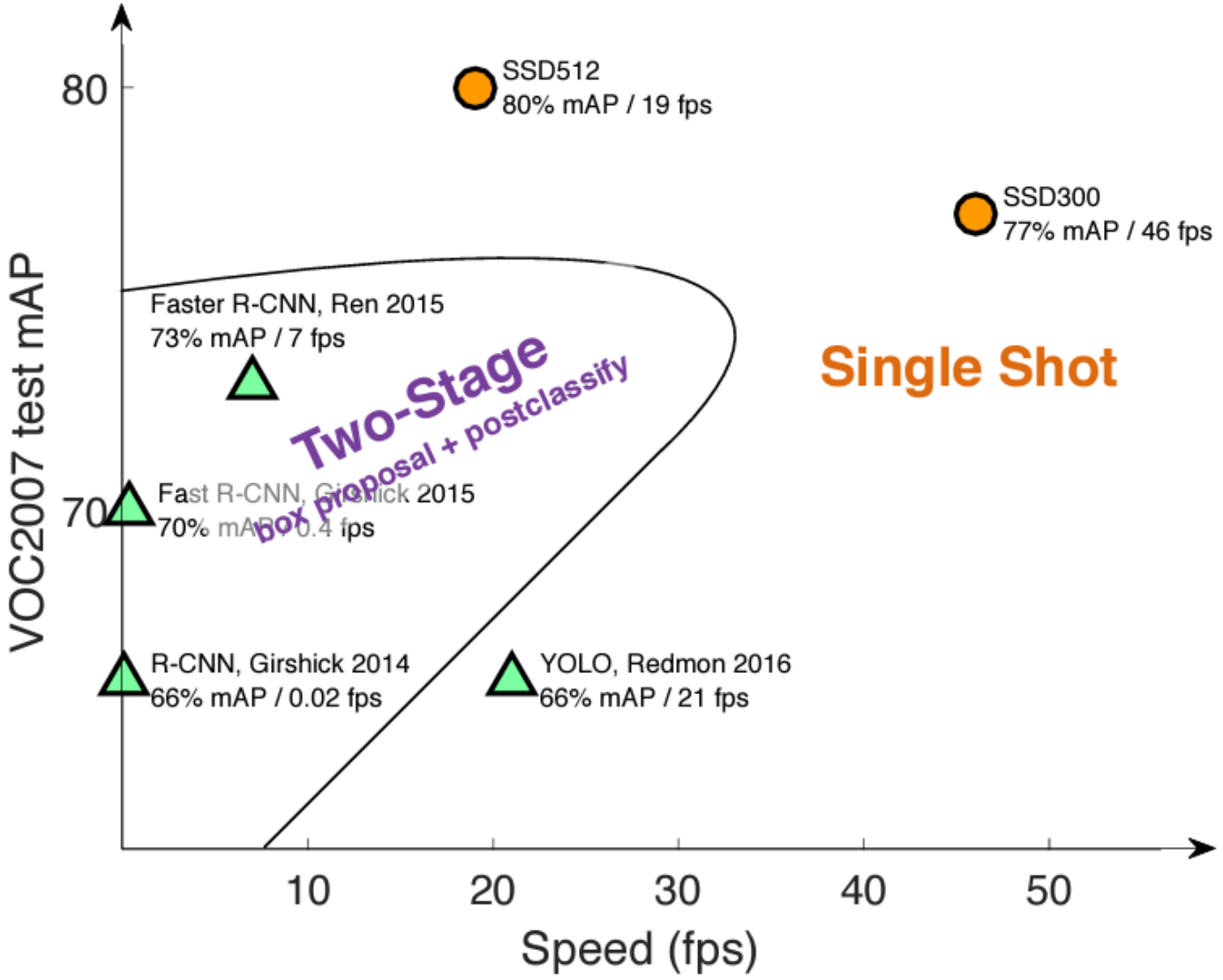
  regardless of the size of bounding boxes

SSD
: Single-Shot MultiBox Detector

2016.12.29 ECCV

Background


SSD512
80% mAP / 19 fps
SSD300
77% mAP / 46 fps
Faster R-CNN, Ren 2015
73% mAP / 7 fps
Two-Stage
box proposal + postclassify
Single Shot
Fast R-CNN, Girshick 2015
70% mAP / 0.4 fps
R-CNN, Girshick 2014
66% mAP / 0.02 fps
YOLO, Redmon 2016
66% mAP / 21 fps
VOC2007 test mAP
80
70
10
20
30
40
50
Speed (fps)

Introduction

SSD : Single-Shot MultiBox Detector

No object proposals (One forward pass)

Set of default boxes over different aspect ratios and scales

Contributions

1. Faster and more accurate than previous single shot detector (YOLO) and Faster R-CNN
2. Predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps
3. Produce predictions of different scales from feature maps of different scales to achieve high detection accuracy, and explicitly separate predictions by aspect ratio
4. Lead to simple end-to-end training and high accuracy, even on low resolution input images
5. Experiments include timing and accuracy analysis on models with varying input size evaluated on PASCAL VOC, COCO and ILSVRC are compared to a range of recent state-of-the-art approaches

Introduction

SSD : Single-Shot MultiBox Detector

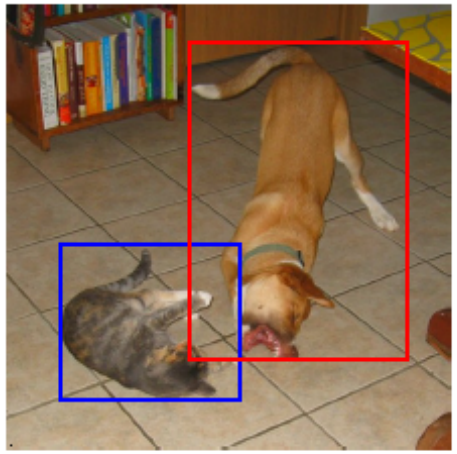
No object proposals (One forward pass)

Set of default boxes over different aspect ratios and scales

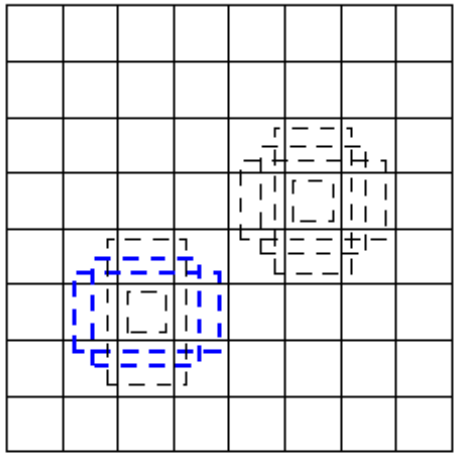
Contributions

1. Faster and more accurate than previous single shot detector (YOLO) and Faster R-CNN
2. Predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps
3. Produce predictions of different scales from feature maps of different scales to achieve high detection accuracy, and explicitly separate predictions by aspect ratio
4. Lead to simple end-to-end training and high accuracy, even on low resolution input images
5. Experiments include timing and accuracy analysis on models with varying input size evaluated on PASCAL VOC, COCO and ILSVRC are compared to a range of recent state-of-the-art approaches

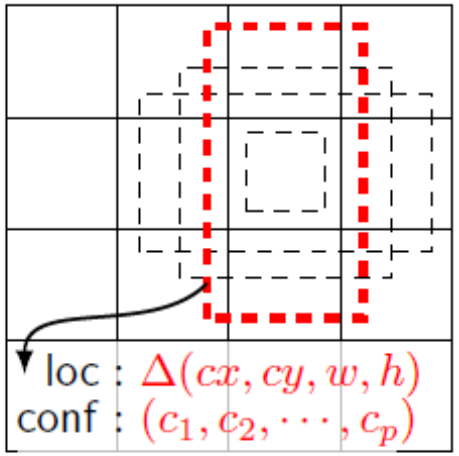
Single Shot Detector (SSD)



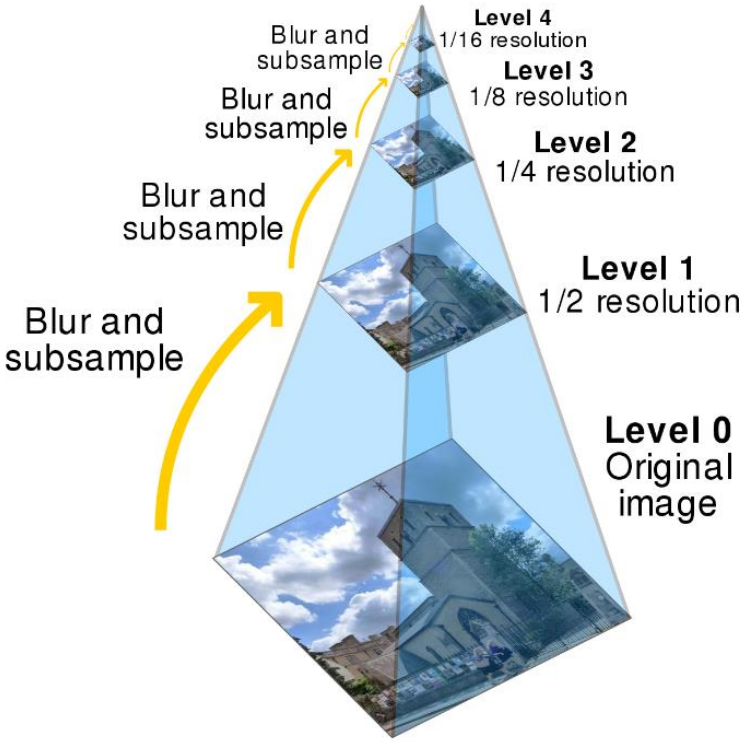
(a) Image with GT boxes



(b) 8×8 feature map

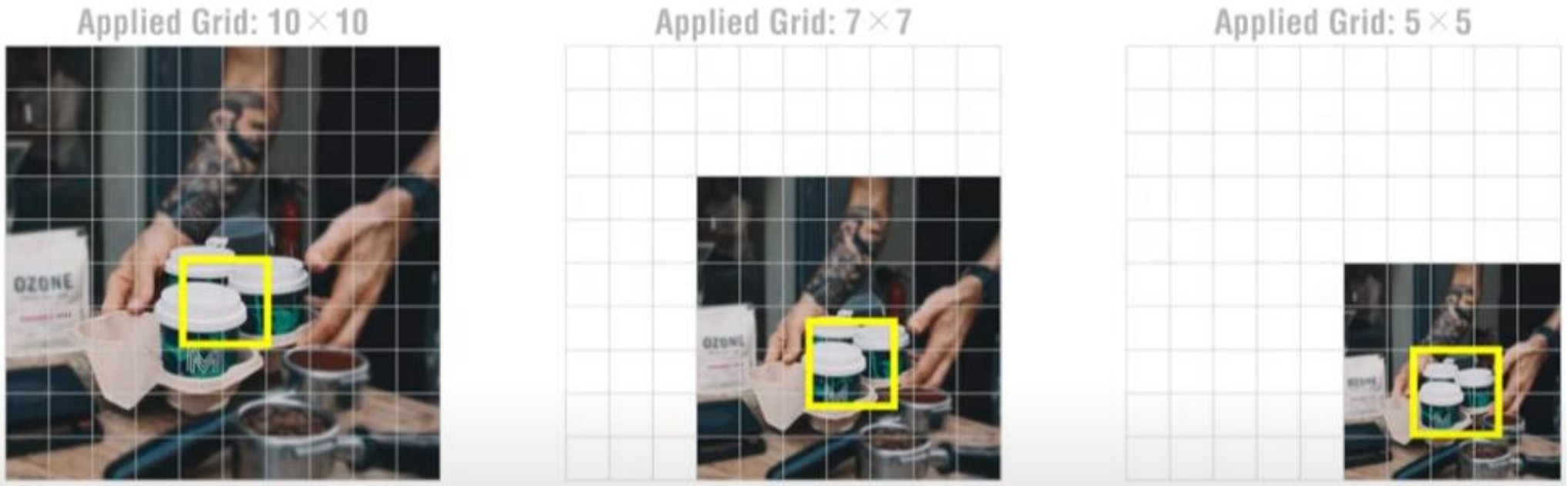


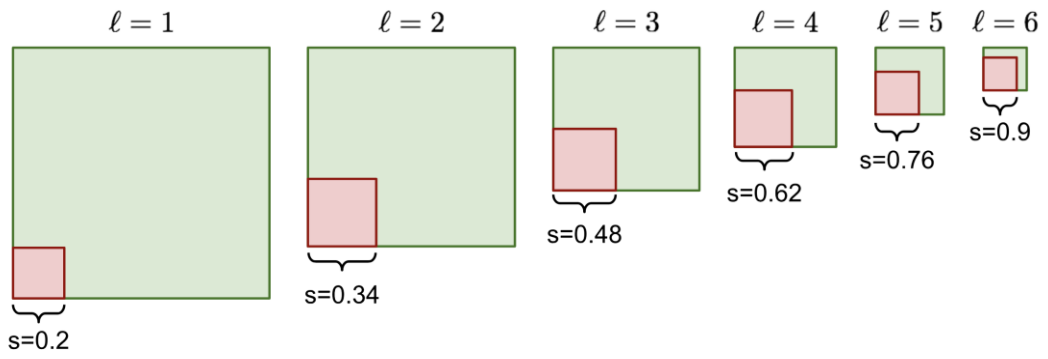
(c) 4×4 feature map



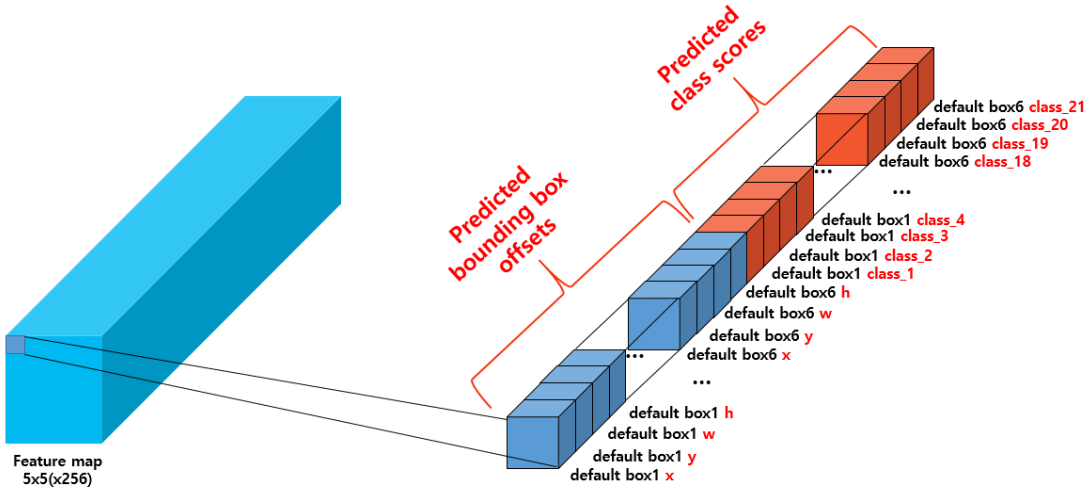
- Evaluate a default boxes of different aspect ratios at each location **in several feature maps with different scales**
- Predict shape offsets and the confidences for all object categories in each default box
- Match default boxes to the ground truth boxes

Single Shot Detector (SSD)





Default Boxes prediction



https://github.com/yeomko22/ssd_defaultbox_generator/blob/master/ssd_defaultbox_generator.ipynb

<Default box scale>

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m-1}(k-1), k \in [1, m]$$

$$s_{min} = 0.2, \qquad s_{max} = 0.9$$

$$a_r \in \{1,2,3,\frac{1}{2},\frac{1}{3}\}$$

$$\begin{cases} w_k^a = s_k\sqrt{a_r} \\ h_k^a = s_k/\sqrt{a_r} \end{cases}$$

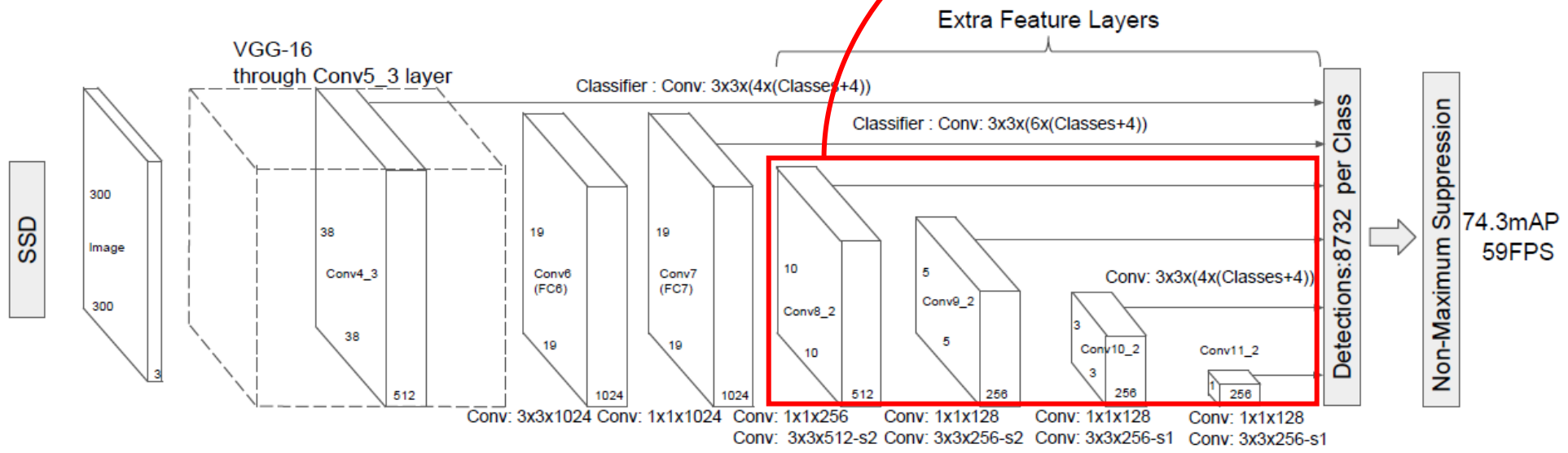
$$\left(\frac{i+0.5}{|f_k|}, \frac{j+0.5}{|f_k|}\right), \quad i,j \in [0, |f_k|)$$

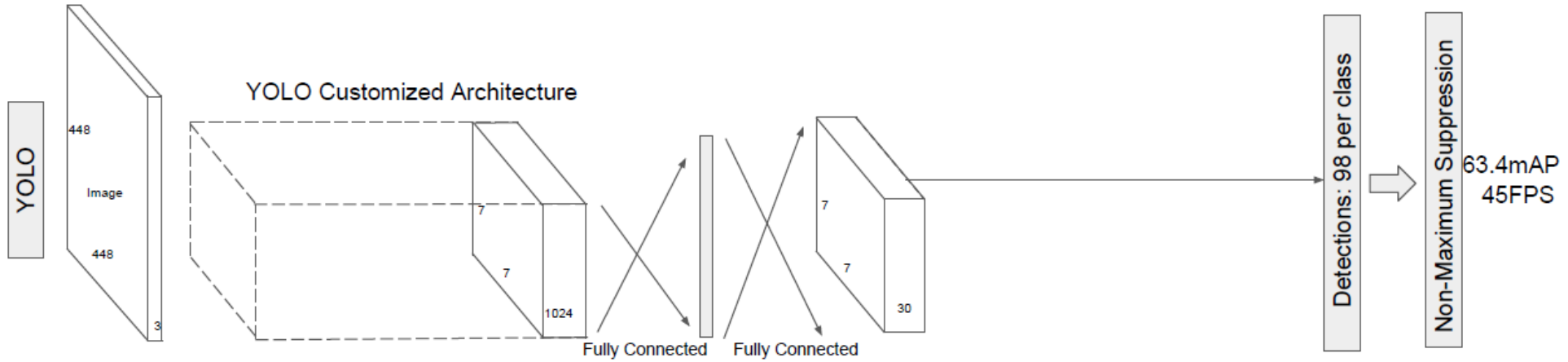
$$s_k' = \sqrt{s_k s_{k+1}} \text{ (if aspect ratio = 1:1)}$$

→ **6** default boxes per feature map location

Model

Add several feature layers to the end of a base network
→ Predict the offsets to default boxes of different scales and aspect ratios and their associated confidences





Training

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

- $L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \epsilon \{cx, cy, w, h\}} x_{ij}^{k} \, smooth_{L_1}(l_i^m - \hat{g}_j^m)$

Localization loss

  - $\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w$
  - $\hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$
  - $\hat{g}_j^w = \log(g_j^w / d_i^w)$
  - $\hat{g}_j^h = \log(g_j^h / d_i^h)$

Confidence loss

- $L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in N_{eg}} \log(\hat{c}_i^0)$

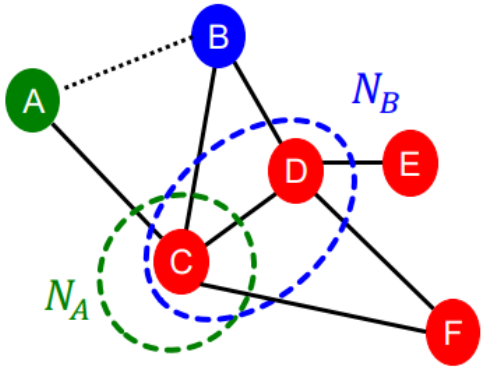
$$where \;\; \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

- $smooth_{L_1}(x) = \begin{cases} 0.5x^2 & if \; |x| < 1 \\ |x| - 0.5 & otherwise \end{cases}$
- $\alpha = 1$ (default)

Training

Matching Strategy

Default box = vary over location, aspect ratio, scale



Matching each GT box to the default box
with the best "**jaccard overlap**" $(threshold = 0.5)$

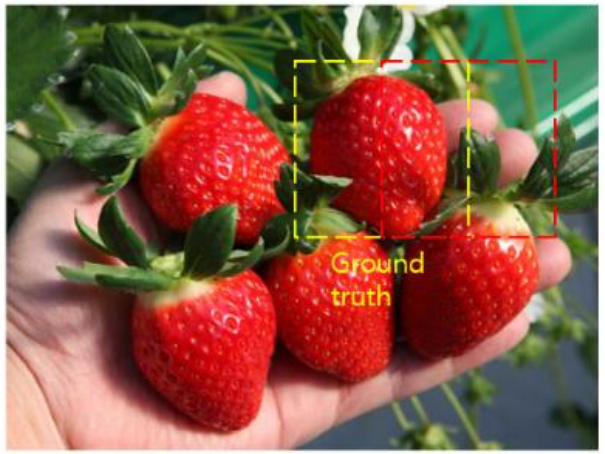
$$S_{Jaccard}[u, v] = \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|}$$

Hard negative mining

Most of the default boxes are negative

~~Using all the negative examples~~
→ sort them using the highest confidence loss
for each default box and pick the top one

Negative : Positive = 3 : 1 (at most)
⇒ **Faster Optimization & Stable training**



Result

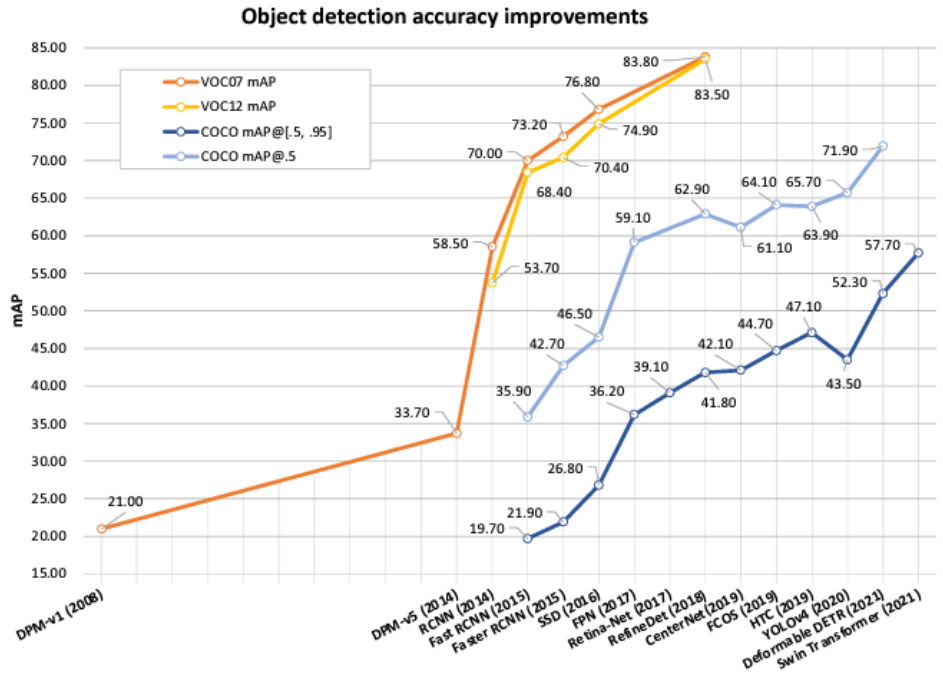
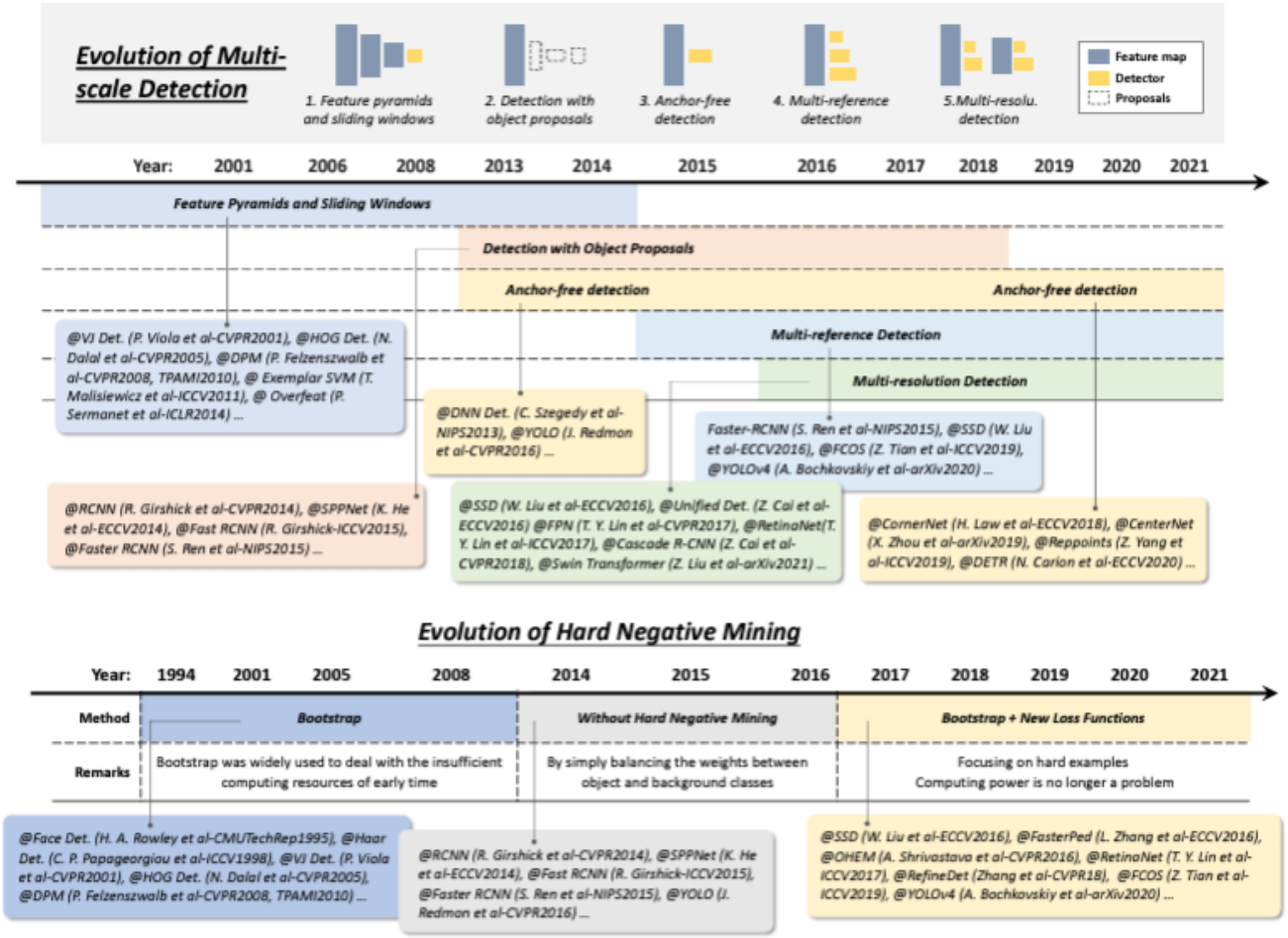


Fig. 3: Accuracy improvement of object detection on VOC07, VOC12 and MS-COCO datasets. Detectors in this figure: DPM-v1 [13], DPM-v5 [37], RCNN [16], SPPNet [17], Fast RCNN [18], Faster RCNN [19], SSD [23], FPN [24], Retina-Net [25], RefineDet [38], TridentNet [39] CenterNet [40], FCOS [41], HTC [42], YOLOv4 [22], Deformable DETR [43], Swin Transformer [44].



*Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, Jeiping Yi.「**Object Detection in 20 Years: A Survey**」, 2019*

Result

PASCAL VOC 2007

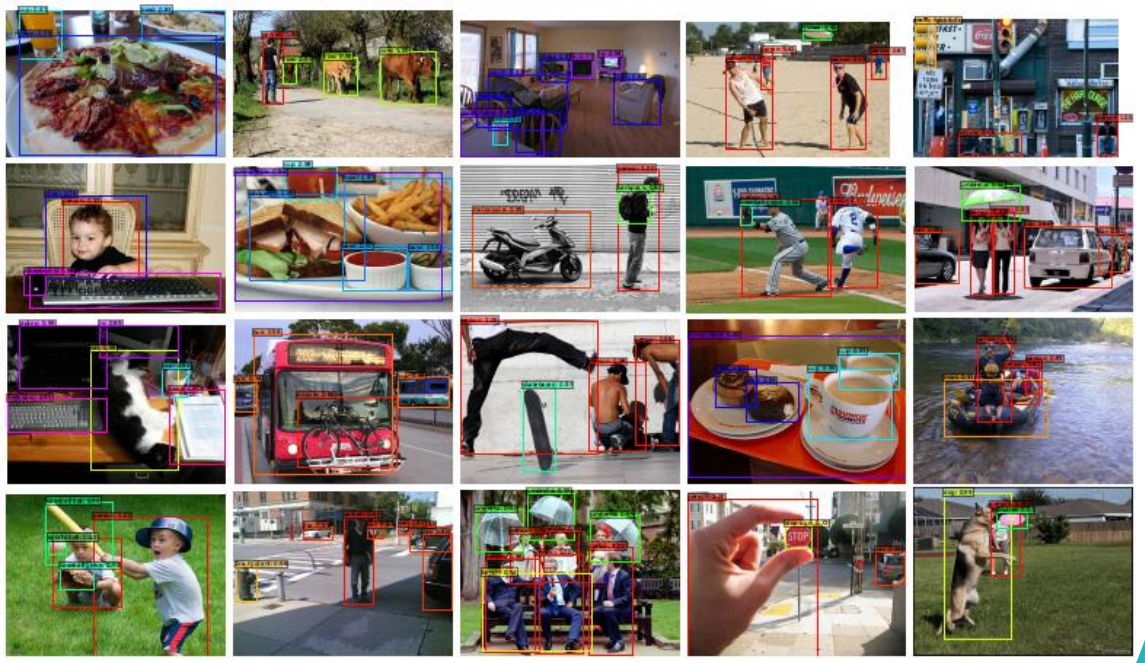
Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
Fast [6]	07+12	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
Faster [2]	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
Faster [2]	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
Faster [2]	07+12+COCO	78.8	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9
SSD300	07	68.0	73.4	77.5	64.1	59.0	38.9	75.2	80.8	78.5	46.0	67.8	69.2	76.6	82.1	77.0	72.5	41.2	64.2	69.1	78.0	68.5
SSD300	07+12	74.3	75.5	80.2	72.3	66.3	47.6	83.0	84.2	86.1	54.7	78.3	73.9	84.5	85.3	82.6	76.2	48.6	73.9	76.0	83.4	74.0
SSD300	07+12+COCO	79.6	80.9	86.3	79.0	**76.2**	57.6	87.3	88.2	88.6	60.5	85.4	**76.7**	**87.5**	**89.2**	84.5	81.4	55.0	81.9	**81.5**	85.9	78.9
SSD512	07	71.6	75.1	81.4	69.8	60.8	46.3	82.6	84.7	84.1	48.5	75.0	67.4	82.3	83.9	79.4	76.6	44.9	69.9	69.1	78.1	71.8
SSD512	07+12	76.8	82.4	84.7	78.4	73.8	53.2	86.2	87.5	86.0	57.8	83.1	70.2	84.9	85.2	83.9	79.7	50.3	77.9	73.9	82.5	75.3
SSD512	07+12+COCO	**81.6**	**86.6**	**88.3**	**82.4**	76.0	**66.3**	**88.6**	**88.9**	**89.1**	**65.1**	**88.4**	73.6	86.5	88.9	**85.3**	**84.6**	**59.1**	**85.0**	80.4	**87.4**	**81.2**

Table 1: **PASCAL VOC2007 `test` detection results.** Both Fast and Faster R-CNN use input images whose minimum dimension is 600. The two SSD models have exactly the same settings except that they have different input sizes (300×300 vs. 512×512). It is obvious that larger input size leads to better results, and more data always helps. Data: "07": VOC2007 `trainval`, "07+12": union of VOC2007 and VOC2012 `trainval`. "07+12+COCO": first train on COCO `trainval35k` then fine-tune on 07+12.

PASCAL VOC 2012

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast[6]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster[2]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
Faster[2]	07++12+COCO	75.9	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2
YOLO[5]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD300	07++12+COCO	77.5	90.2	83.3	76.3	63.0	53.6	83.8	82.8	92.0	59.7	82.7	63.5	89.3	87.6	85.9	84.3	52.6	82.5	**74.1**	**88.4**	74.2
SSD512	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
SSD512	07++12+COCO	**80.0**	**90.7**	**86.8**	**80.5**	**67.8**	**60.8**	**86.3**	**85.5**	**93.5**	**63.2**	**85.7**	**64.4**	**90.9**	**89.0**	**88.9**	**86.8**	**57.2**	**85.1**	72.8	**88.4**	**75.9**

Table 4: **PASCAL VOC2012 `test` detection results.** Fast and Faster R-CNN use images with minimum dimension 600, while the image size for YOLO is 448×448. data: "07++12": union of VOC2007 `trainval` and `test` and VOC2012 `trainval`. "07++12+COCO": first train on COCO `trainval35k` then fine-tune on 07++12.



Result

PASCAL VOC 2007

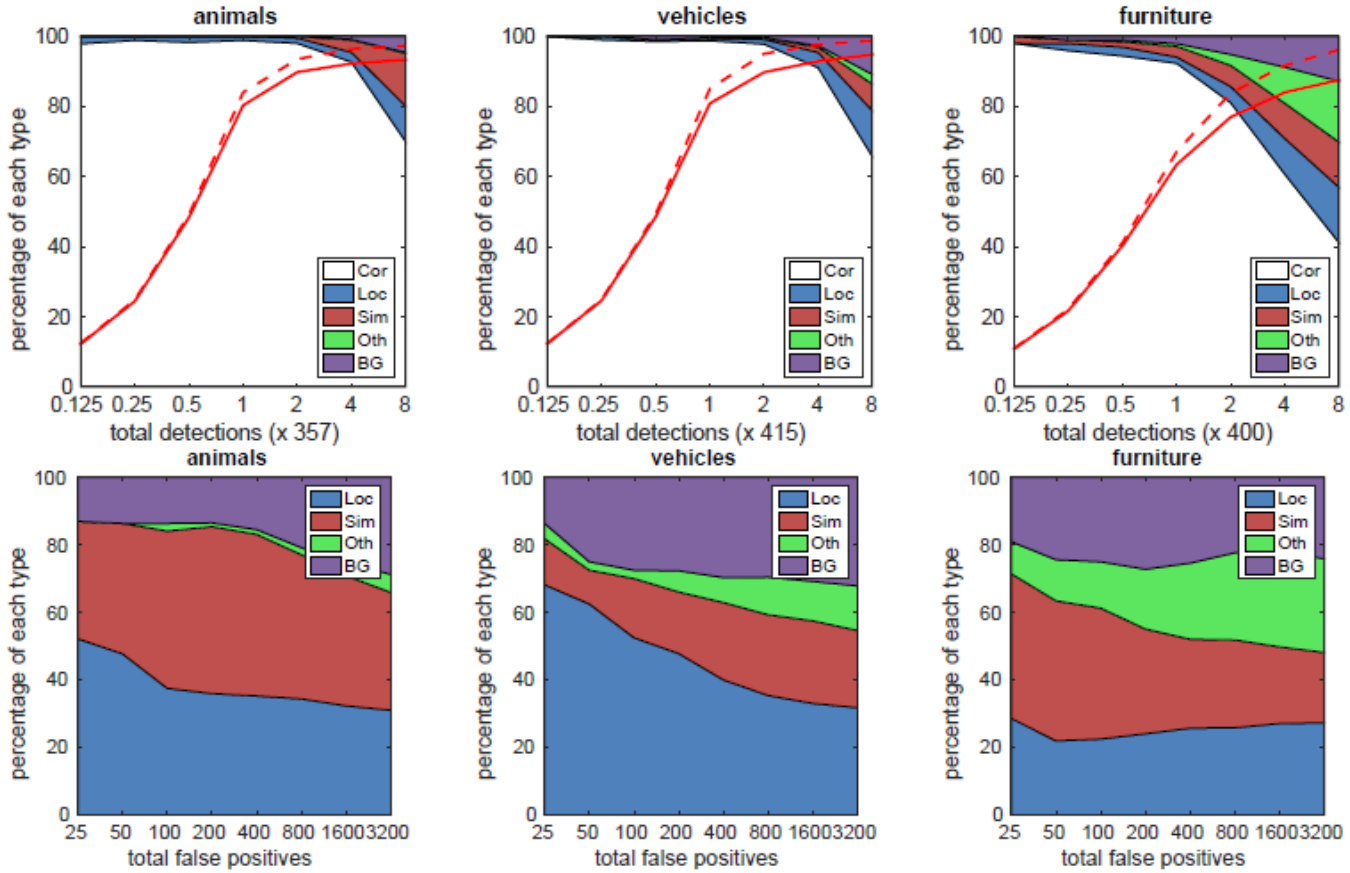


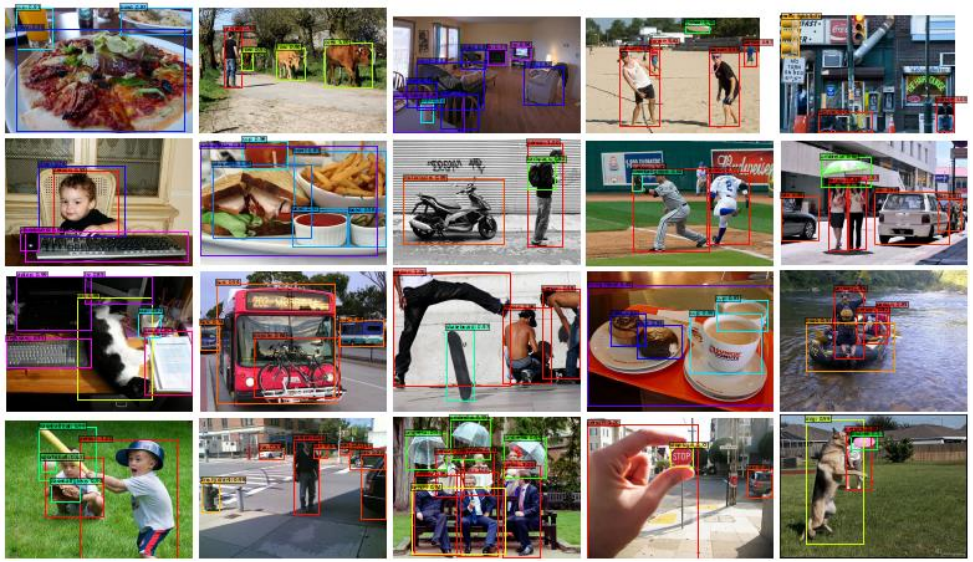
Fig. 3: **Visualization of performance for SSD512 on animals, vehicles, and furniture from VOC2007 `test`.** The top row shows the cumulative fraction of detections that are correct (Cor) or false positive due to poor localization (Loc), confusion with similar categories (Sim), with others (Oth), or with background (BG). The solid red line reflects the change of recall with strong criteria (0.5 jaccard overlap) as the number of detections increases. The dashed red line is using the weak criteria (0.1 jaccard overlap). The bottom row shows the distribution of top-ranked false positive types.

Result

MS COCO test-dev 2015

Method	data	Avg. Precision, IoU: 0.5:0.95	0.5	0.75	Avg. Precision, Area: S	M	L	Avg. Recall, #Dets: 1	10	100	Avg. Recall, Area: S	M	L
Fast [6]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast [24]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster [2]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [24]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster [25]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**

Table 5: **COCO `test-dev2015` detection results.**



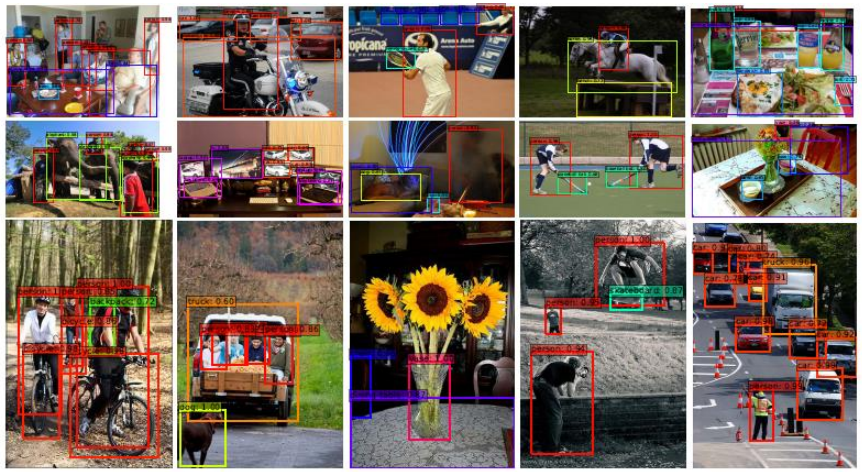


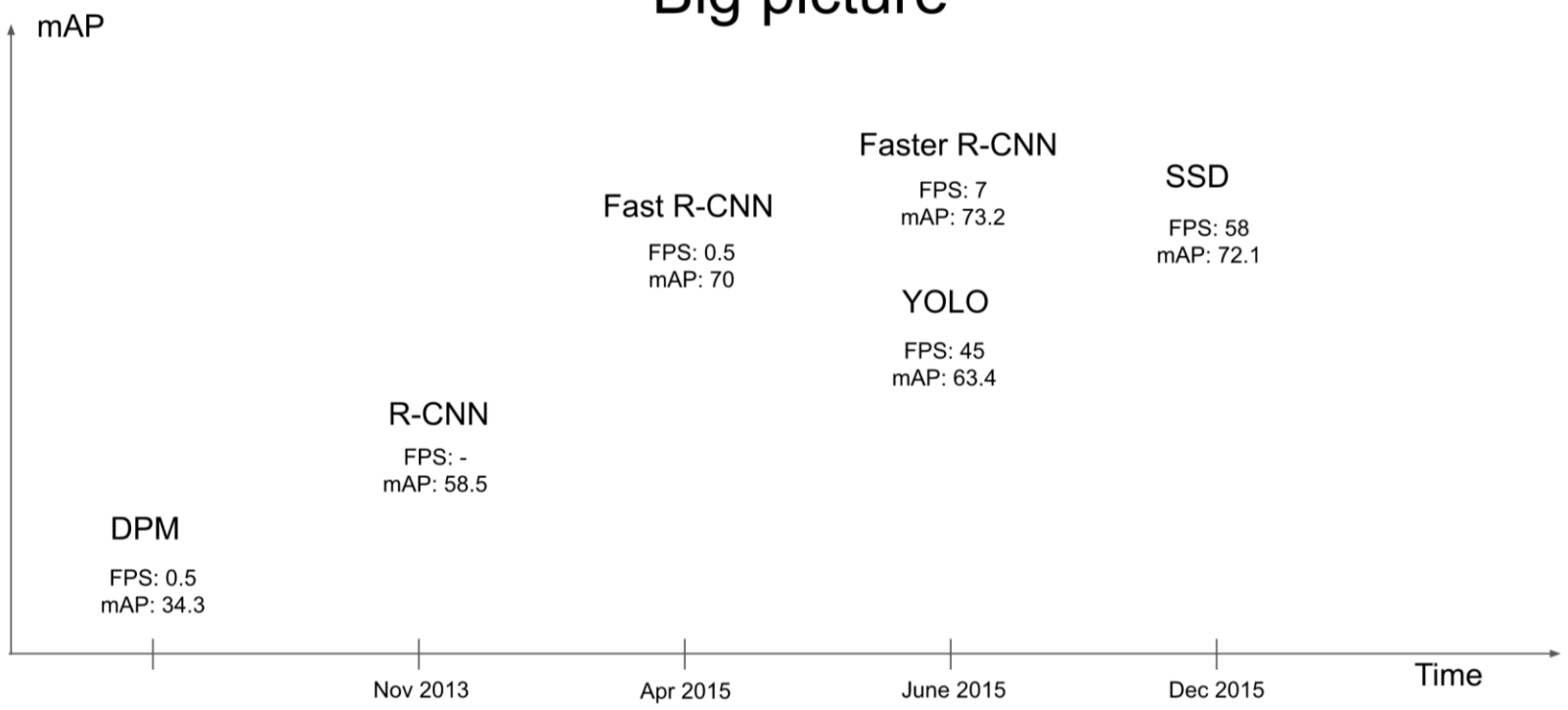
Fig. 5: **Detection examples on COCO `test-dev` with SSD512 model.** We show detections with scores higher than 0.6. Each color corresponds to an object category.

Result

Method	VOC2007 test		VOC2012 test		COCO test-dev2015		
	07+12	07+12+COCO	07++12	07++12+COCO	trainval35k		
	0.5	0.5	0.5	0.5	0.5:0.95	0.5	0.75
SSD300	74.3	79.6	72.4	77.5	23.2	41.2	23.4
SSD512	76.8	81.6	74.9	80.0	26.8	46.5	27.8
SSD300*	77.2	81.2	75.8	79.3	25.1	43.1	25.8
SSD512*	**79.8**	**83.2**	**78.5**	**82.2**	**28.8**	**48.5**	**30.3**

Table 6: **Results on multiple datasets when we add the image expansion data augmentation trick.** SSD300* and SSD512* are the models that are trained with the new data augmentation.

Big picture



Conclusion

SSD

Method	*mAP*	FPS	Test batch size	# Boxes
Faster R-CNN [2] (VGG16)	73.2	7	1	300
Faster R-CNN [2] (ZF)	62.1	17	1	300
YOLO [5]	63.4	45	1	98
Fast YOLO [5]	52.7	155	1	98
SSD300	74.3	46	1	8732
SSD512	76.8	19	1	24564
SSD300	74.3	59	8	8732
SSD512	76.8	22	8	24564

Strength

- Fast single-shot object detector for multiple categories
- Use of multi-scale convolutional bounding box outputs attached to multiple feature maps at the top of the network → Efficiently model the space of possible box shapes
- Use default bounding box instead of region proposal

	SSD300				
more data augmentation?		✔	✔	✔	✔
include $\{\frac{1}{2}, 2\}$ box?	✔		✔	✔	✔
include $\{\frac{1}{3}, 3\}$ box?	✔			✔	✔
use atrous?	✔	✔	✔		✔
VOC2007 `test` mAP	65.5	71.6	73.7	74.4	74.3

Weakness

- Cannot detect small objects properly because small objects performs detection by using feature maps which are created in the front layer
- Reliance on "**Data augmentation**" (for small objects)

YOLO9000

: Better, Faster, Stronger

2016.12.25 CVPR

Introduction

- State-of-the art on PASCAL VOC and MS COCO (78.6 mAP at 40 FPS on VOC 2007)
- Jointly train on object detection and classification → predict unlabeled detection data

<New Method>

- Harness the large amount of classification data that already have and use it to expand the scope of current detection systems

- **Joint training algorithm**
  : Leverages labeled detection images to learn to precisely localize objects while it uses classification images to increase its vocabulary and robustness

Better

YOLO vs Fast R-CNN

- Localization errors ↑

- YOLO has relatively low "Recall" compared to region proposal-based methods

⇒ **Focus mainly on improving "Recall" and localization while maintaining accuracy**

$$* \ Recall = \frac{TP}{TP+FN} = \frac{TP}{All\ Ground\ Truths}$$

Solution

: Better Performance ∝ Training large network & Ensembling multiple models

"More accurate detector that is still fast with simpler network"

Better

Batch Normalization

mAP 2% ↑ by adding BN to all conv layers

Can remove dropouts without overfitting

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

High Resolution Classifier

YOLOv1 : pre-train (224x224) → detection (448x448)

YOLOv2 : fine-tune classification network (448x448) x 10 epochs on ImageNet

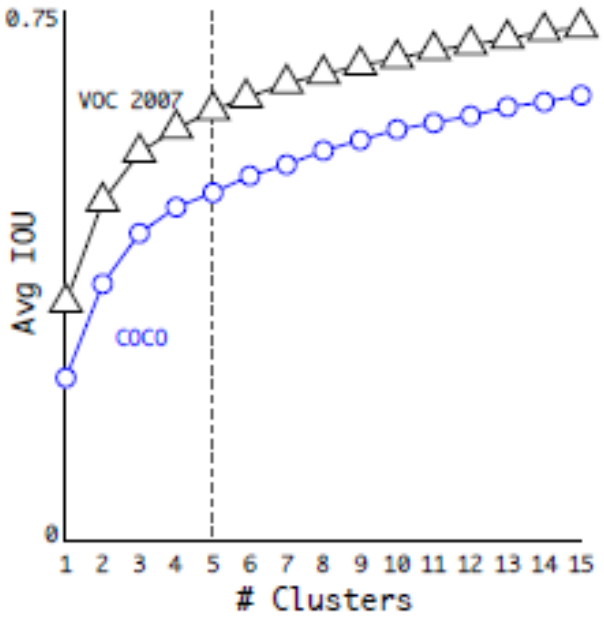
→ fine-tune again on detection ⇒ **mAP 4% ↑**

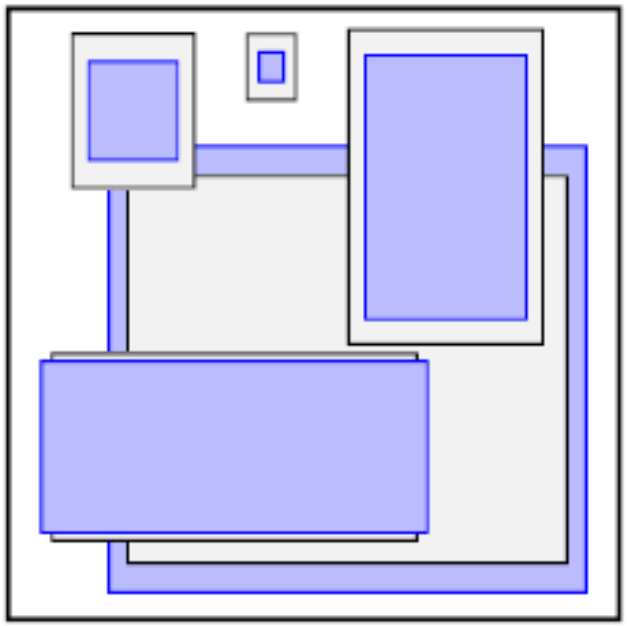
Convolutional with Anchor boxes

- Removed fully-connected layers and added anchor boxes (≒ RPN in Faster R-CNN)
- Input image size = 448x448 → 416x416
- Predict class and objectiveness for every anchor box
- Output feature map : 13x13

Better

Dimension Clusters





Box Generation	#	Avg IOU
Cluster SSE	5	58.7
Cluster IOU	5	61.0
Anchor Boxes [15]	9	60.9
Cluster IOU	9	67.2

Table 1: Average IOU of boxes to closest priors on VOC 2007. The average IOU of objects on VOC 2007 to their closest, unmodified prior using different generation methods. Clustering gives much better results than using hand-picked priors.

k ↑ =Recall ↑ (∵IoU ↑) but Complexity ↑

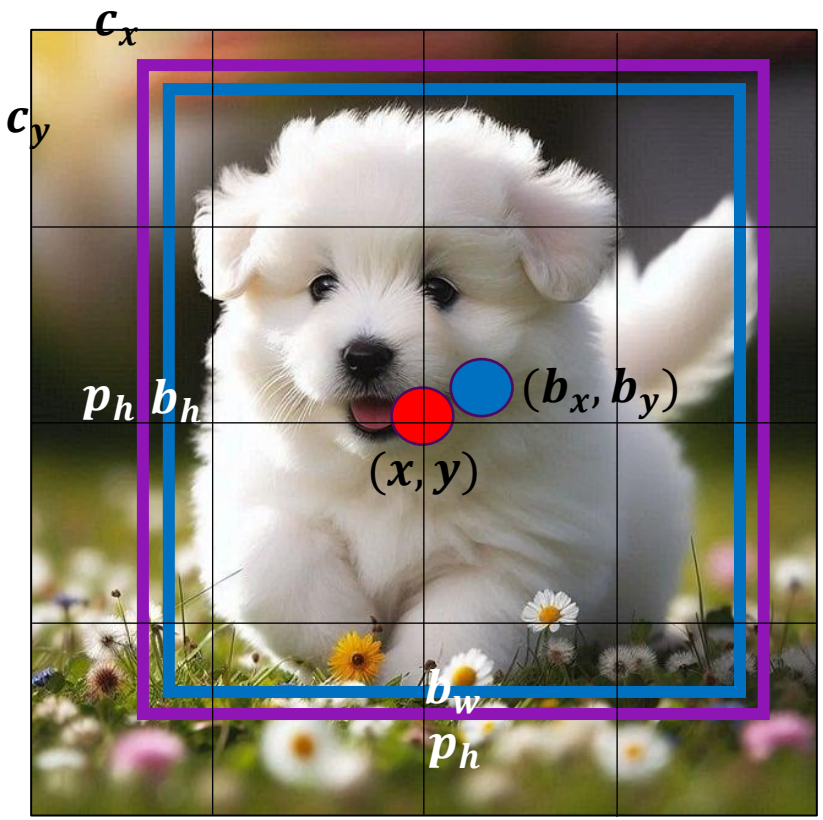
Box-dimension : by hand (Original) → K-means clustering (k=5)

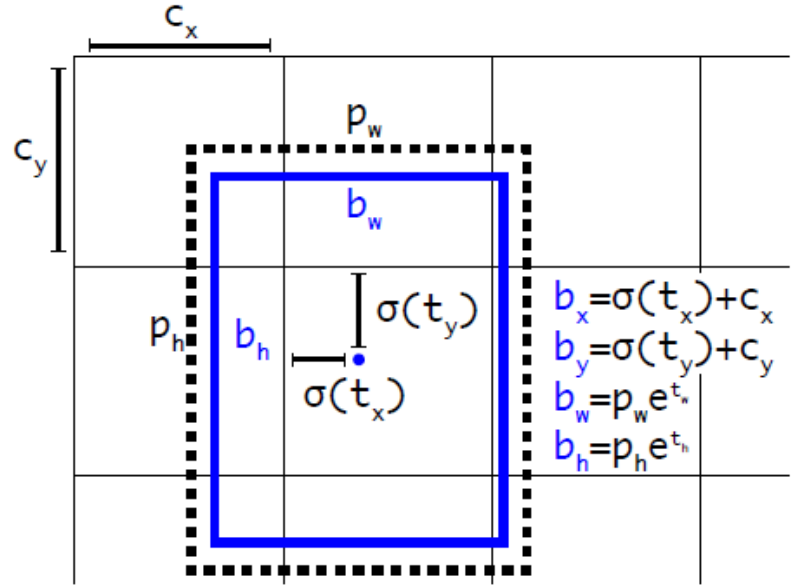
$$d(box, centroid) = 1 - IoU(box, centroid)$$

Better

Direct location prediction

Issue : **Model Instability** (iteration)
… from predicting (x, y) in the box





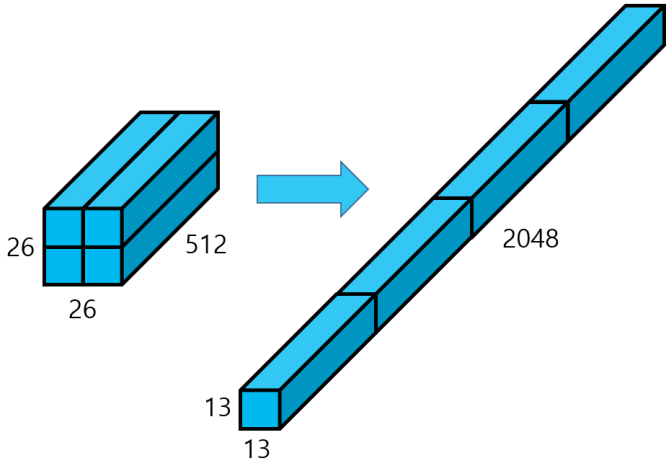
$$x = (t_x * w_a) - x_a$$
$$y = (t_y * h_a) - y_a$$

$$b_x = \sigma(t_x) + c_x \qquad (t_x = 1 : shift\ right)$$
$$b_y = \sigma(t_y) + c_y \qquad (t_x = -1 : shift\ left)$$
$$b_w = p_w e^{t_w} \qquad (p_w = prior)$$
$$b_h = p_h e^{t_h} \qquad (p_h = prior)$$

$$\Pr(object) * IoU(b, object) = \sigma(t_o)$$

Better

Fine-grained Features



YOLOv1 → weakness in detecting small objects

∴ adding "**passthrough layer**" (≒ Identity mapping in ResNet)
Benefit for localizing smaller objects

Multi-Scale Training

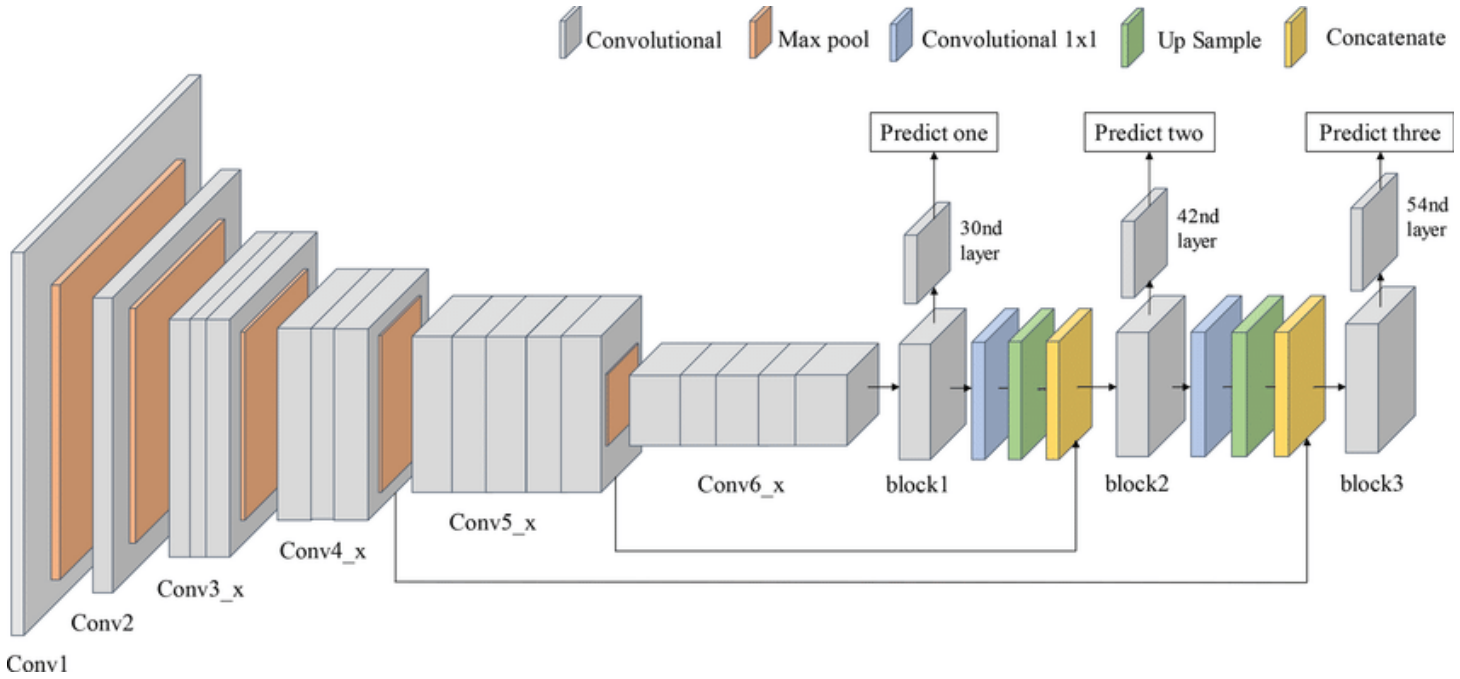
Robust to the different size of images
→ Randomly choose input size every 10 iters

320x320 ~ 608x608

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	78.6	40

Faster

Darknet-19



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Table 6: Darknet-19.

Darknet-19

19 Convolutional layers
5 Max-pooling layers

5.58 billion operations

[ImageNet]
72.9% top-1 accuracy
91.2% top-5 accuracy

Stronger

Jointly training on classification and detection data

→ Backpropagate based on YOLOv2 loss function

Problem

Dataset's label range : **Detection <<<<<<<<<< Classification**

⇒ **Need coherent ways to merge labels**

Animal

Dog = Dog

Breed

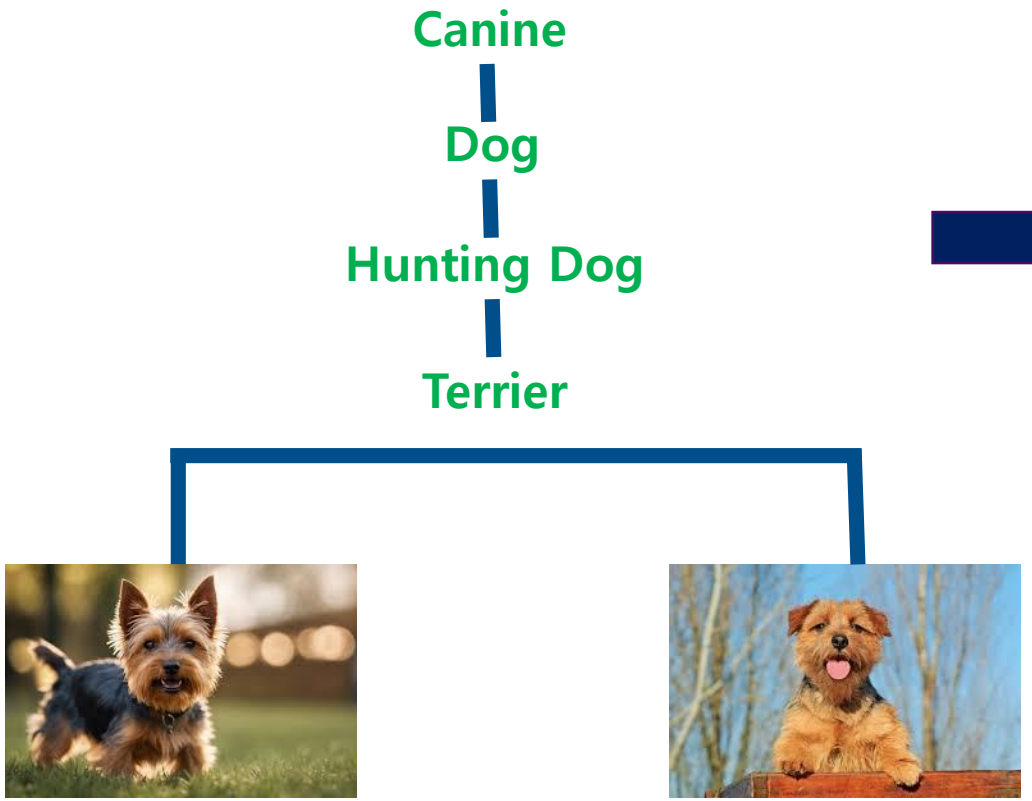
Yorkshire terrier ≠ Norfolk terrier

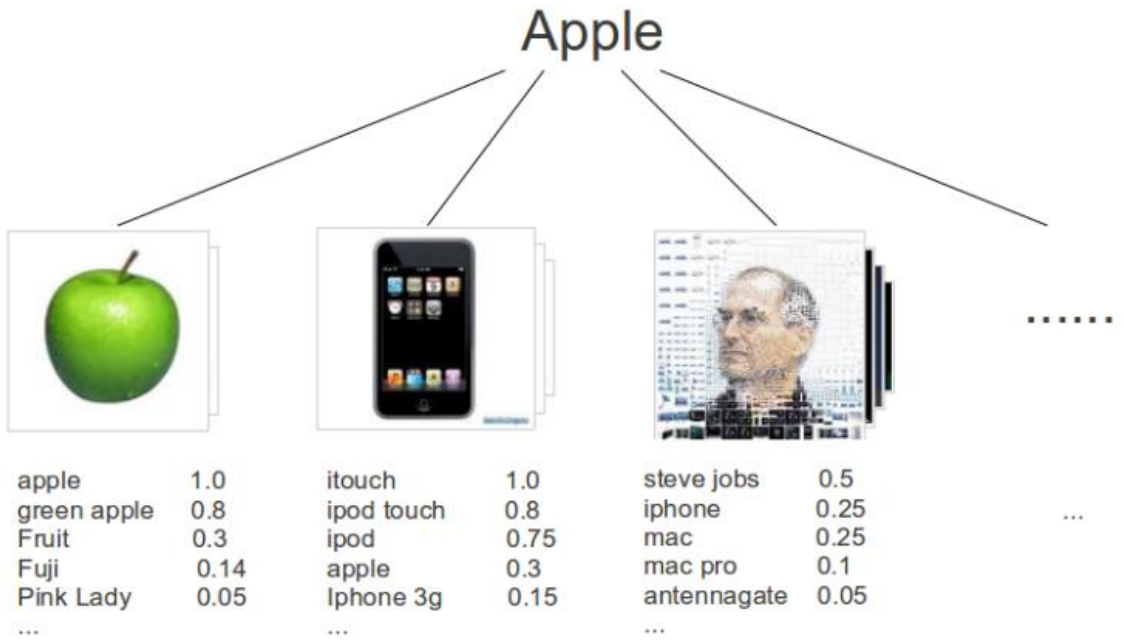
Use "**Multi-label model**" to combine datasets which does not assume mutual exclusion

Stronger

Solution - Hierarchical classification

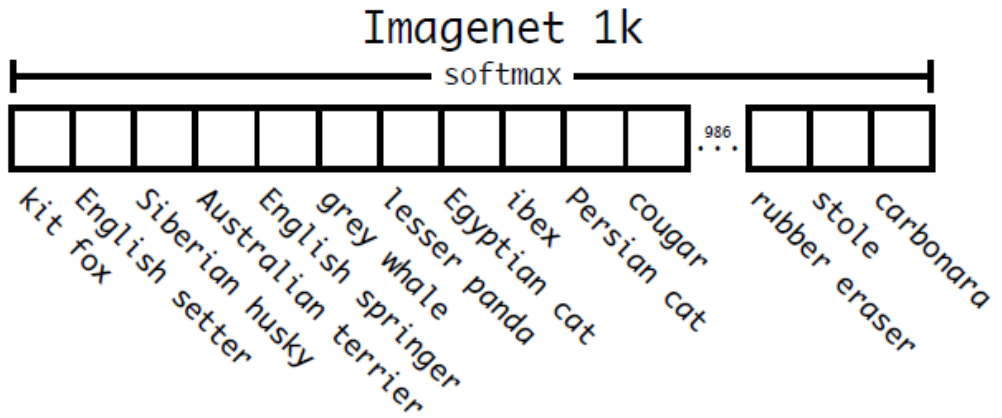
Directed Graph



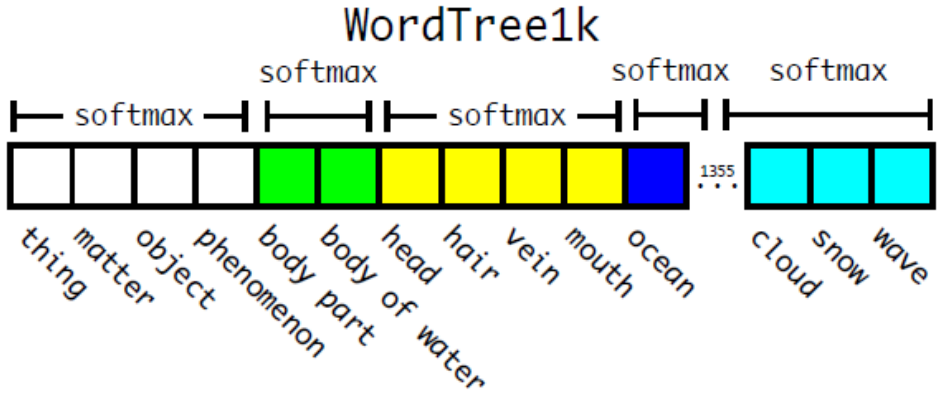


Stronger

Solution - Hierarchical classification



$$\Pr(\text{Norfolk terrier}|\text{terrier})$$
$$\Pr(\text{Yorkshire terrier}|\text{terrier})$$
$$\Pr(\text{Bedlington terrier}|\text{terrier})$$

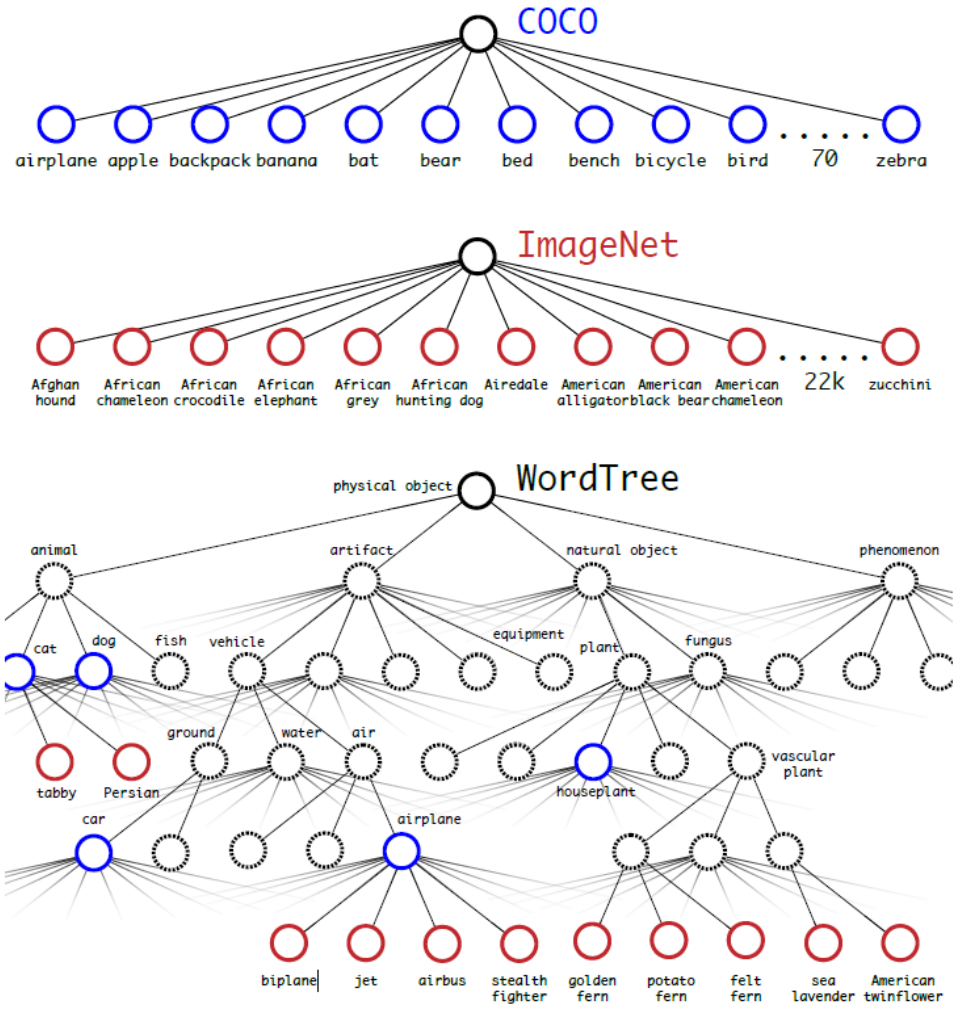


$$\Pr(\text{Norfolk terrier}) = \Pr(\text{Norfolk terrier}|\text{terrier}) * \Pr(\text{terrier}|\text{hunting dog}) * \cdots * \Pr(\text{mammal}|\text{animal}) * \Pr(\text{animal}|\text{physical object})$$

$$\text{Root} = \Pr(\text{physical object}) = 1$$

Stronger

Solution - Hierarchical classification



Offer more richer, more detailed output space for image classification

Training

$$Loss_{box} = \lambda_{obj}^{coord} \sum_{i}^{S^2} \sum_{j}^{B} \mathbb{I}_{ij}^{responsible_{obj}} \left(x_{ij}^{pred} - x_{ij}^{obj}\right)^2 + \left(y_{ij}^{pred} - y_{ij}^{pred}\right)^2 + \left(w_{ij}^{pred} - w_{ij}^{obj}\right)^2 + \left(h_{ij}^{pred} - h_{ij}^{obj}\right)^2$$

$$+\lambda_{noobj}^{coord} \sum_{i}^{S^2} \sum_{j}^{B} \mathbb{I}_{ij}^{no\_responsible\_object} \left(x_{ij}^{pred} - x_{ij}^{anchor\_center}\right)^2 + \left(y_{ij}^{pred} - y_{ij}^{anchor\_center}\right)^2$$

$$+\left(w_{ij}^{pred} - w_{ij}^{anchor\_default}\right)^2 + \left(h_{ij}^{pred} - h_{ij}^{anchor\_default}\right)^2$$

Localization Loss

$$+\lambda_{obj}^{conf} \sum_{i}^{S^2} \sum_{j}^{B} \mathbb{I}_{ij}^{responsible\_obj} \left\{conf_{ij}^{pred} - iou(box_{ij}^{pred}, box_{ij}^{truth})\right\}^2$$

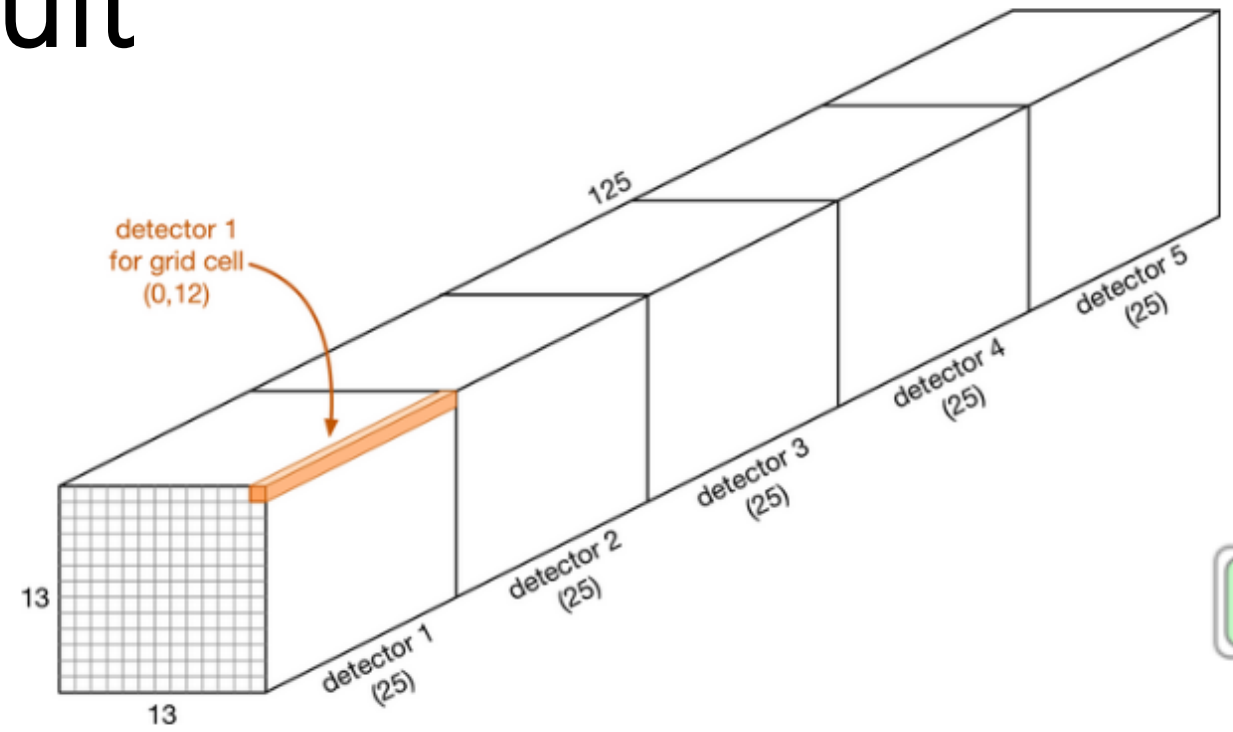
$$+\lambda_{noobj}^{conf} \sum_{i}^{S^2} \sum_{j}^{B} \mathbb{I}_{ij}^{no\_responsible\_obj} \left\{conf_{ij}^{pred} - 0\right\}^2$$

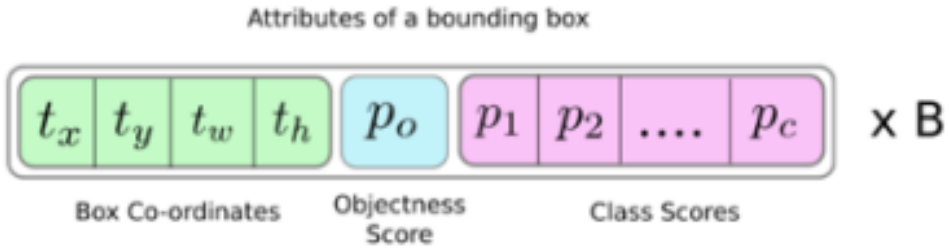
Confidence Loss

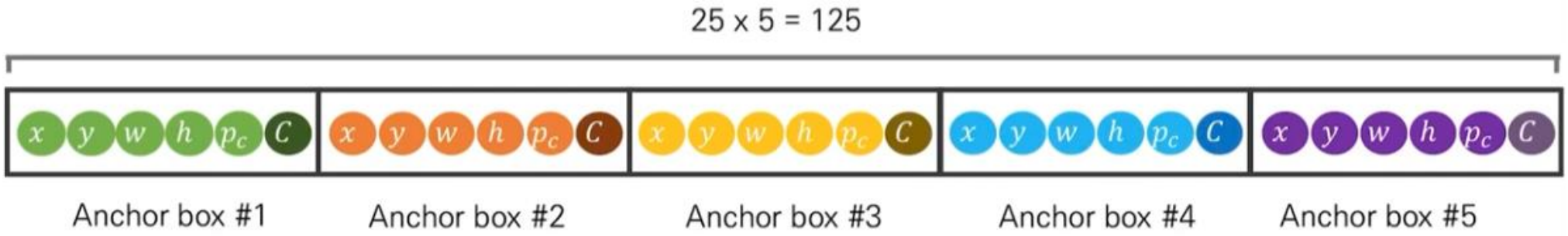
$$+\sum_{i}^{S^2} \sum_{j}^{B} \mathbb{I}_{ij}^{responsible\_obj} \left\{p_{ij}^{pred}(c) - p_{ij}^{truth}(c)\right\}^2$$

Classification Loss

Result


125
detector 1
for grid cell
(0,12)
13
13
detector 1
(25)
detector 2
(25)
detector 3
(25)
detector 4
(25)
detector 5
(25)


Attributes of a bounding box
t_x t_y t_w t_h p_o p_1 p_2 .... p_c
x B
Box Co-ordinates
Objectness Score
Class Scores


25 x 5 = 125
x y w h p_c C
Anchor box #1
Anchor box #2
Anchor box #3
Anchor box #4
Anchor box #5

Result

$$\text{Feature map size} = 13 * 13 * \big(5 * (\#coord + conf + \#class)\big)$$
$$= 13 * 13 * \big(5 * (4 + 1 + 20)\big) = 13 * 13 * 125$$

Result

YOLOv1 → YOLOv2

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

MS COCO

		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [5]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast R-CNN[1]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster R-CNN[15]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [1]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster R-CNN[10]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300 [11]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [11]	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**
YOLOv2 [11]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4

PASCAL VOC 2012

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Result

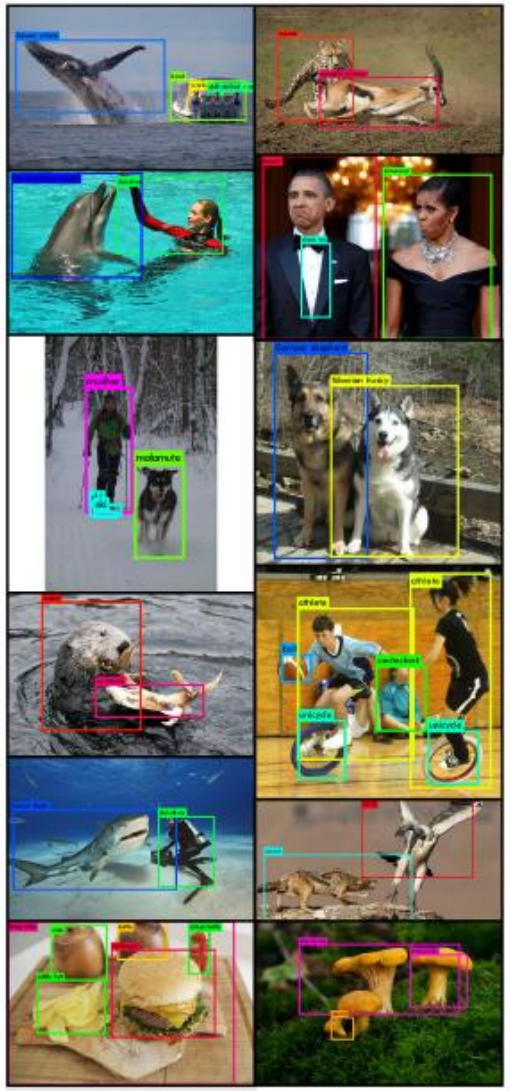
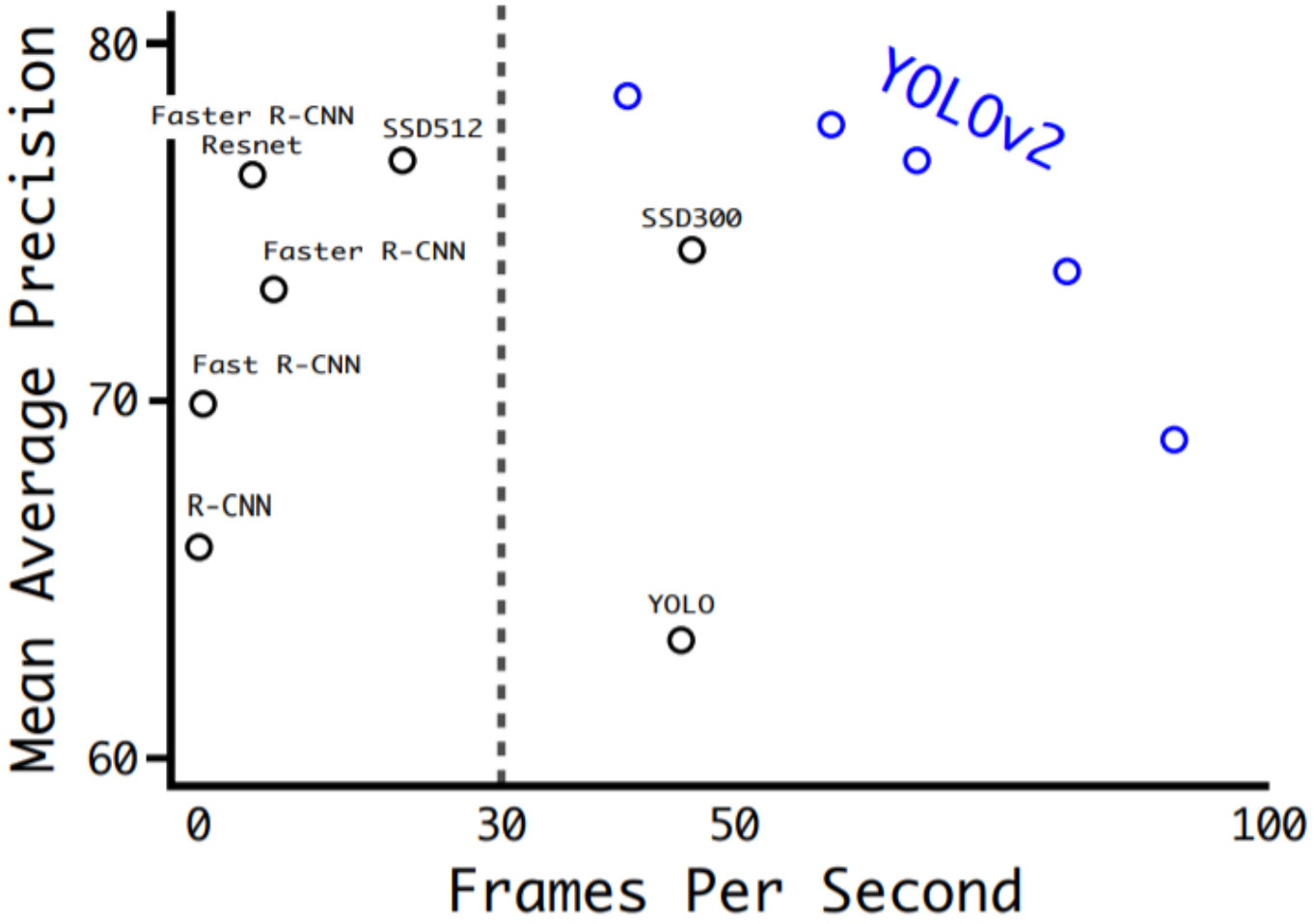


Figure 1: YOLO9000. YOLO9000 can detect a wide variety of object classes in real-time.



Conclusion

YOLO9000 (YOLOv2)

Strength

- Run at a variety of image sizes to provide a smooth tradeoff between speed and accuracy
- Use WordTree to combine data from various sources
- Train simultaneously on ImageNet and COCO

Weakness

- Learns good models for a variety of animals but struggles with new classes

diaper	0.0
horizontal bar	0.0
rubber eraser	0.0
sunglasses	0.0
swimming trunks	0.0
...	
red panda	50.7
fox	52.1
koala bear	54.3
tiger	61.0
armadillo	61.7

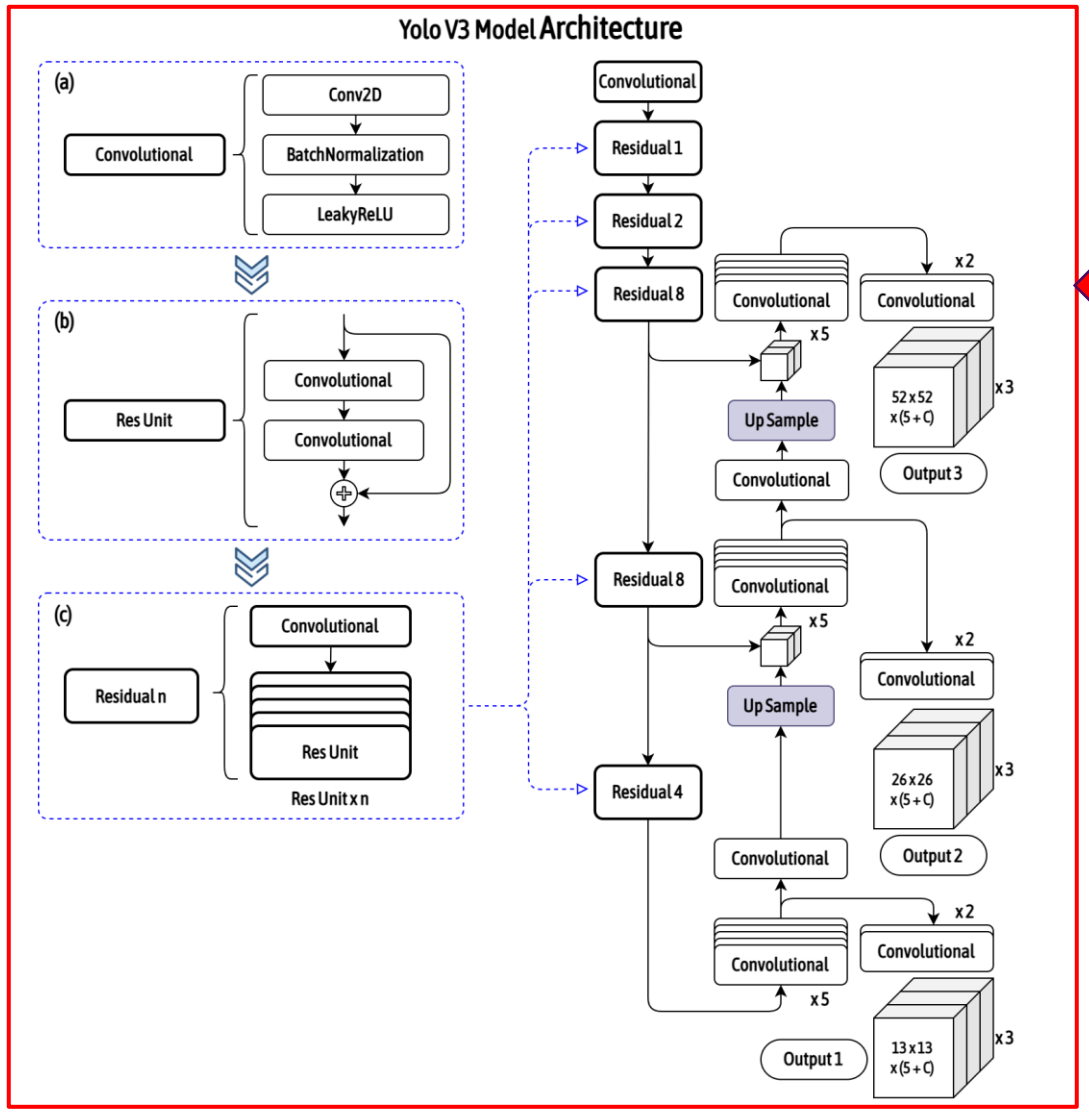
Table 7: YOLO9000 Best and Worst Classes on ImageNet. The classes with the highest and lowest AP from the 156 weakly supervised classes. YOLO9000 learns good models for a variety of animals but struggles with new classes like clothing or equipment.

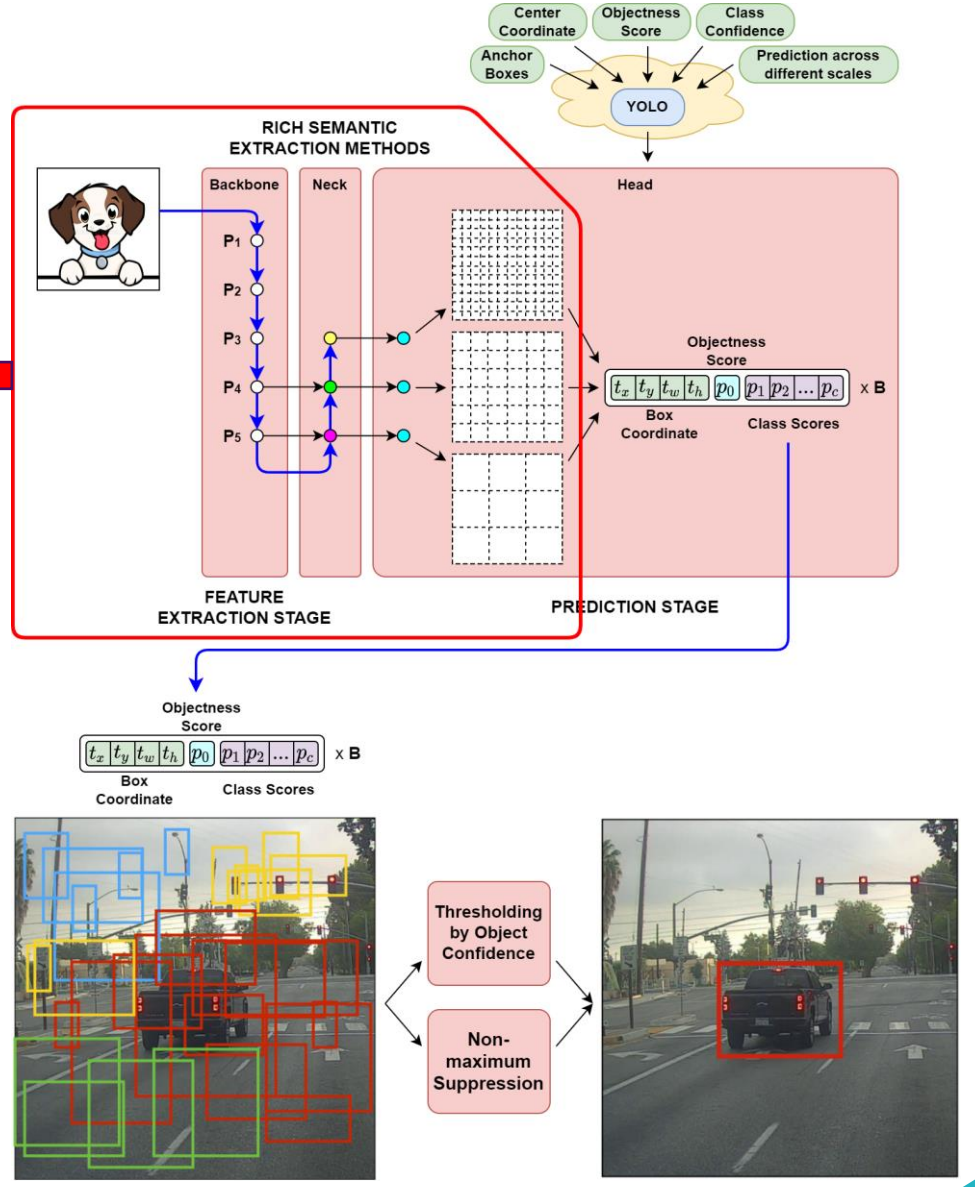
YOLOv3

: An Incremental Improvement

2018.4.8 CVPR

Design


Yolo V3 Model Architecture
(a)
Convolutional
Conv2D
BatchNormalization
LeakyReLU
(b)
Res Unit
Convolutional
Convolutional
(c)
Residual n
Convolutional
Res Unit
Res Unit x n
Convolutional
Residual 1
Residual 2
Residual 8
Residual 8
Residual 4
x2
x5
x3
Up Sample
52 x 52 x (5 + C)
Output 3
26 x 26 x (5 + C)
Output 2
13 x 13 x (5 + C)
Output 1


Center Coordinate
Objectness Score
Class Confidence
Anchor Boxes
Prediction across different scales
YOLO
RICH SEMANTIC EXTRACTION METHODS
Backbone
Neck
Head
P1
P2
P3
P4
P5
Objectness Score
Box Coordinate
Class Scores
x B
FEATURE EXTRACTION STAGE
PREDICTION STAGE
Thresholding by Object Confidence
Non-maximum Suppression

Design

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 2. **Comparison of backbones.** Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

$$FLOPS = cores * clock * \frac{FLOPs}{cycle}$$

: **FL**oating point **O**perations **P**er **S**econd

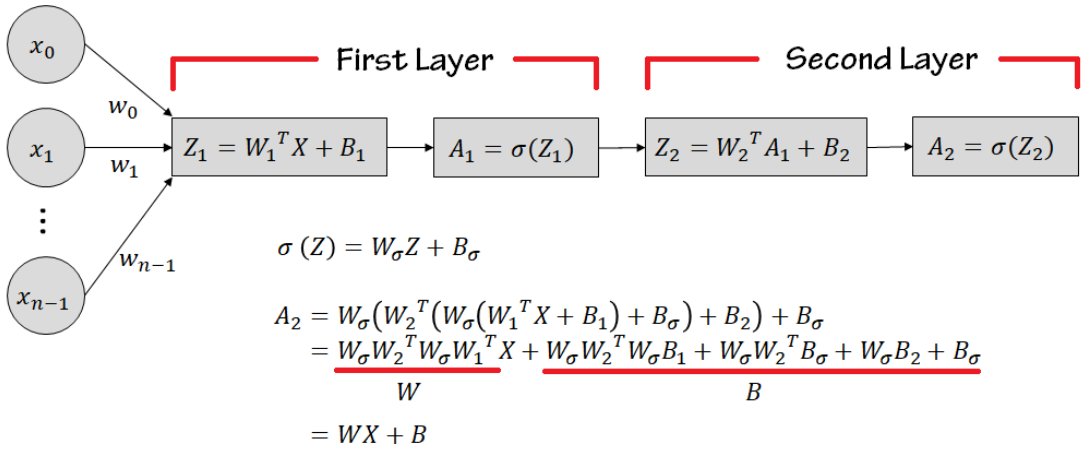
⇒ Better utilize GPU

Trial Failures

Anchor box x, y offset predictions

Normal anchor box prediction by using linear activation

⇒ **Decrease model stability**



Linear x, y predictions instead of logistics

Using linear activation to directly predict x, y offset instead of logistic activation

⇒ **Couple point drop in mAP**

Trial Failures

Focal loss

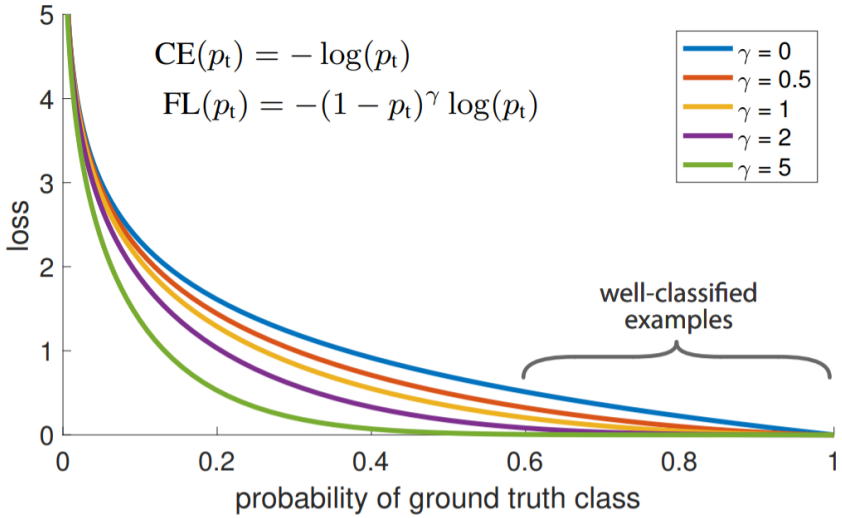
To solve class imbalance in cross-entropy loss,
May already be robust to the problem

⇒ **mAP 2% ↓**

$$CE(p_t) = -\alpha \log(p_t)$$
$$FL(p_t) = \underline{(1-p_t)^{\gamma}} \log(p_t)$$

Modulating factor

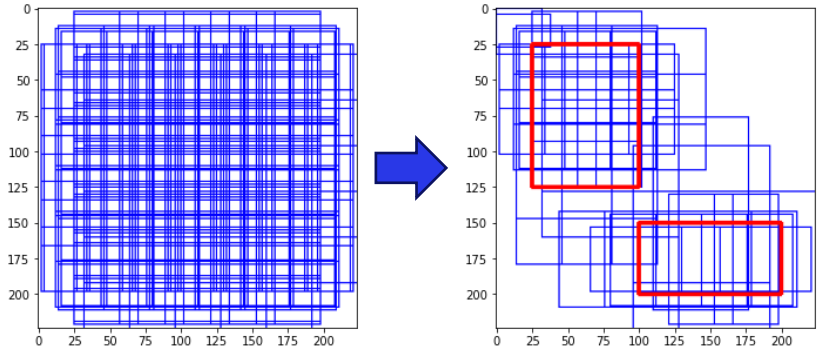
Tunable focusing parameter



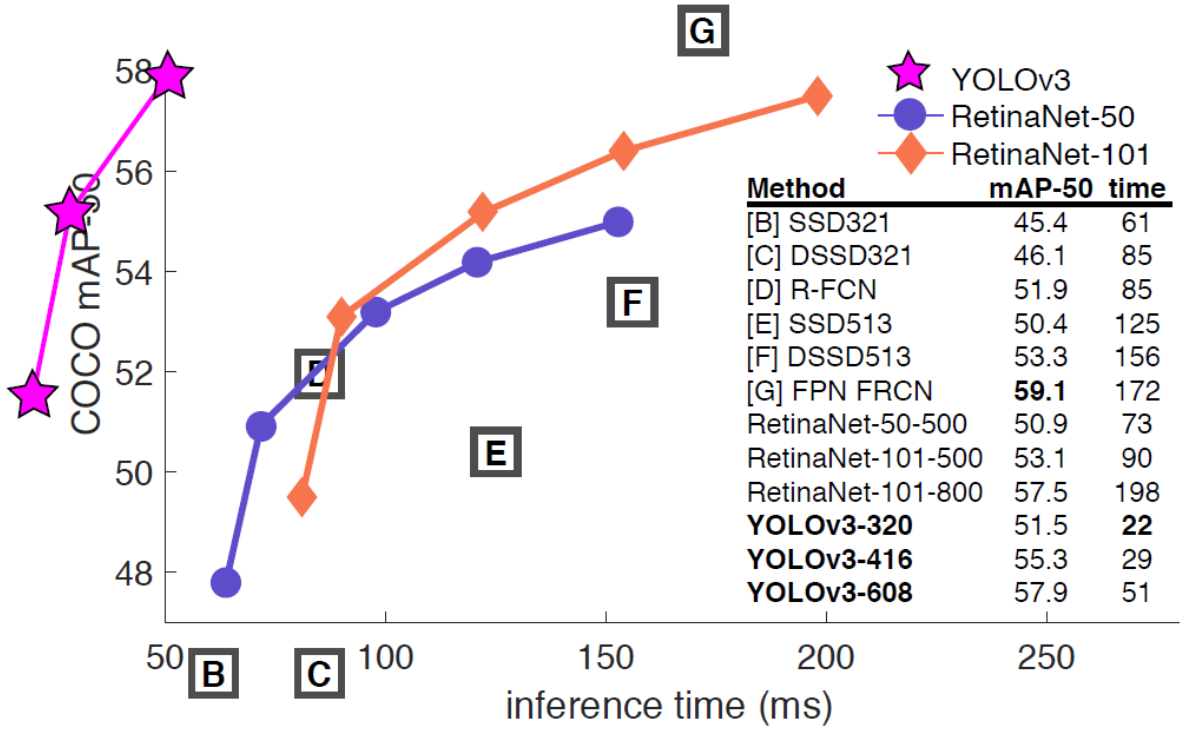
Dual IoU thresholds and truth assignment

Faster R-CNN uses 2 IoU thresholds during training.

$$\begin{cases} Positive\ (1) & if, IoU \geq 0.7 \\ Negative\ (0) & elif, IoU \leq 0.3 \\ No\ Contribution & O.T.W \end{cases}$$

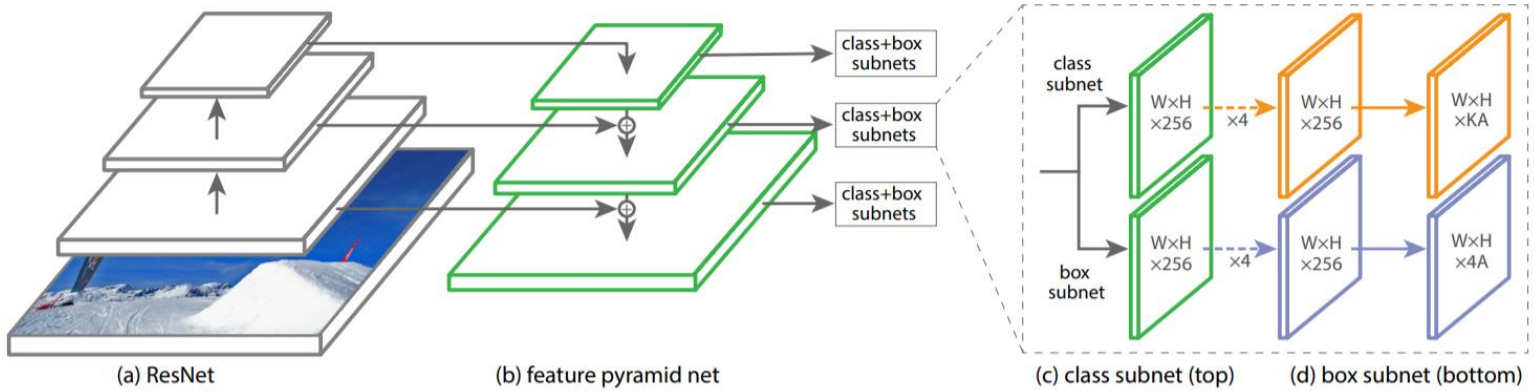


Result



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 3. Again adapted from the [9], this time displaying speed/accuracy tradeoff on the mAP at .5 IOU metric. You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try, this guy → [16]. Oh, I forgot, we also fix a data loading bug in YOLOv2, that helped by like 2 mAP. Just sneaking this in here to not throw off layout.



Result

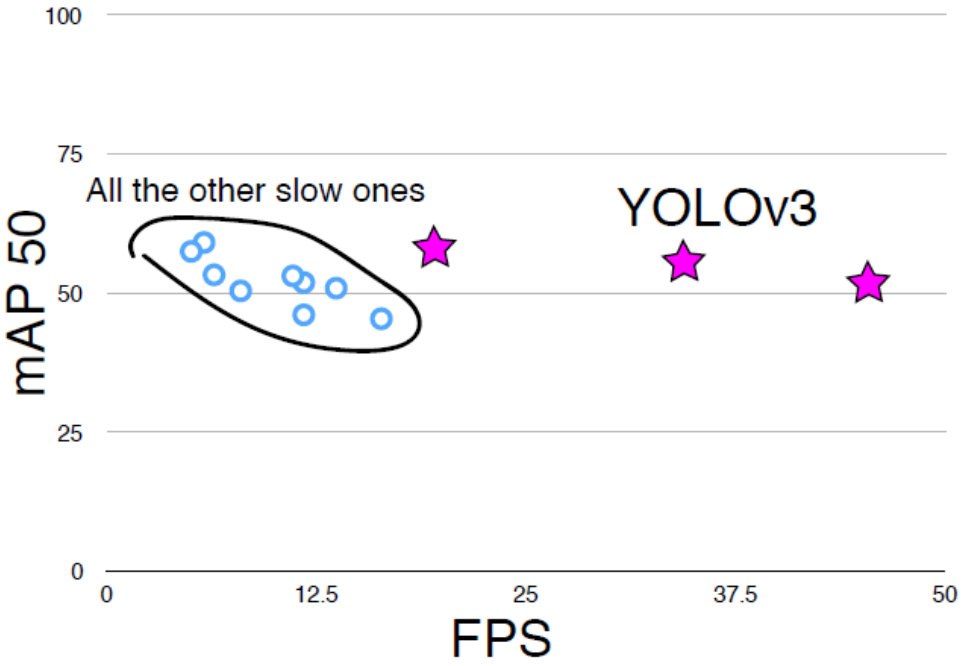


Figure 4. Zero-axis charts are probably more intellectually honest... and we can still screw with the variables to make ourselves look good!

Result

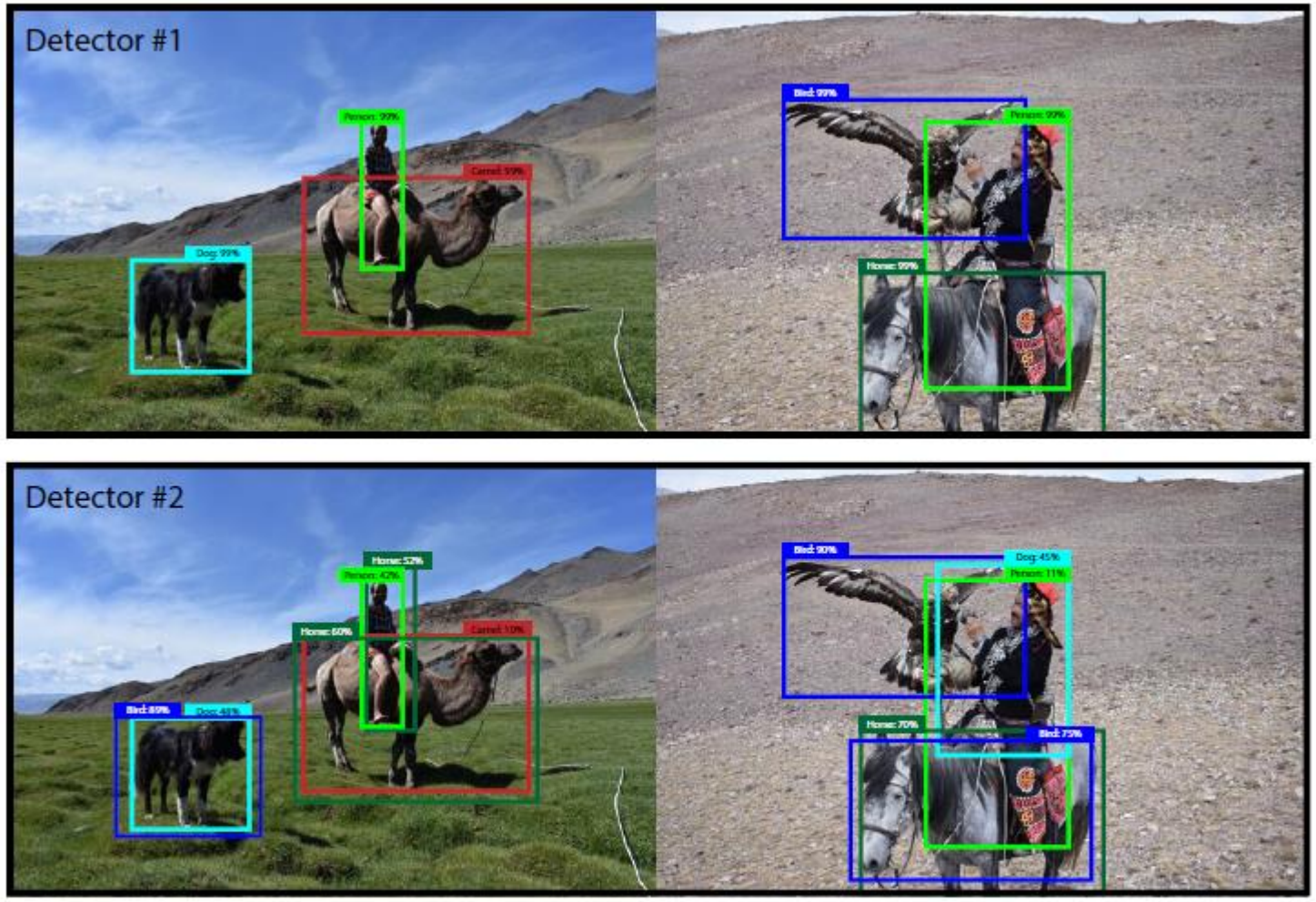


Figure 5. These two hypothetical detectors are perfect according to mAP over these two images. They are both perfect. Totally equal.

Result

Code (YOLOv3)

```
[1]  1 !git clone https://github.com/pjreddie/darknet

Cloning into 'darknet'...
remote: Enumerating objects: 5955, done.
remote: Total 5955 (delta 0), reused 0 (delta 0), pack-reused 5955
Receiving objects: 100% (5955/5955), 6.37 MiB | 9.44 MiB/s, done.
Resolving deltas: 100% (3932/3932), done.
```

```
 1 !make

mkdir -p obj
mkdir -p backup
mkdir -p results
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors
```

```
 1 !wget https://pjreddie.com/media/files/yolov3.weights

--2024-07-18 07:09:50--  https://pjreddie.com/media/files/yolov3.weights
Resolving pjreddie.com (pjreddie.com)... 162.0.215.52
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 248007048 (237M) [application/octet-stream]
Saving to: 'yolov3.weights'

yolov3.weights      100%[===================>] 236.52M  38.8MB/s    in 6.7s

2024-07-18 07:09:57 (35.3 MB/s) - 'yolov3.weights' saved [248007048/248007048]
```

Result

Code

(YOLOv3)

```
1 !./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

```
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x  32  0.639 BFLOPs
    1 conv     64  3 x 3 / 2   608 x 608 x  32   ->   304 x 304 x  64  3.407 BFLOPs
    2 conv     32  1 x 1 / 1   304 x 304 x  64   ->   304 x 304 x  32  0.379 BFLOPs
    3 conv     64  3 x 3 / 1   304 x 304 x  32   ->   304 x 304 x  64  3.407 BFLOPs
    4 res    1                 304 x 304 x  64   ->   304 x 304 x  64
    5 conv    128  3 x 3 / 2   304 x 304 x  64   ->   152 x 152 x 128  3.407 BFLOPs
    6 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64  0.379 BFLOPs
    7 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128  3.407 BFLOPs
    8 res    5                 152 x 152 x 128   ->   152 x 152 x 128
    9 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64  0.379 BFLOPs
   10 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128  3.407 BFLOPs
   11 res    8                 152 x 152 x 128   ->   152 x 152 x 128
   12 conv    256  3 x 3 / 2   152 x 152 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   13 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   14 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   15 res   12                  76 x  76 x 256   ->    76 x  76 x 256
   16 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   17 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   18 res   15                  76 x  76 x 256   ->    76 x  76 x 256
   19 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   20 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   21 res   18                  76 x  76 x 256   ->    76 x  76 x 256
   22 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   23 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   24 res   21                  76 x  76 x 256   ->    76 x  76 x 256
   25 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   26 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
```

Darknet 53

Darknet 19

```
   99 conv    128  1 x 1 / 1    76 x  76 x 384   ->    76 x  76 x 128  0.568 BFLOPs
  100 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  101 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
  102 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  103 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
  104 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
  105 conv    255  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 255  0.754 BFLOPs
  106 yolo
Loading weights from yolov3.weights...Done!
data/dog.jpg: Predicted in 19.976752 seconds.
dog: 100%
truck: 92%
bicycle: 99%
```

(YOLOv2)

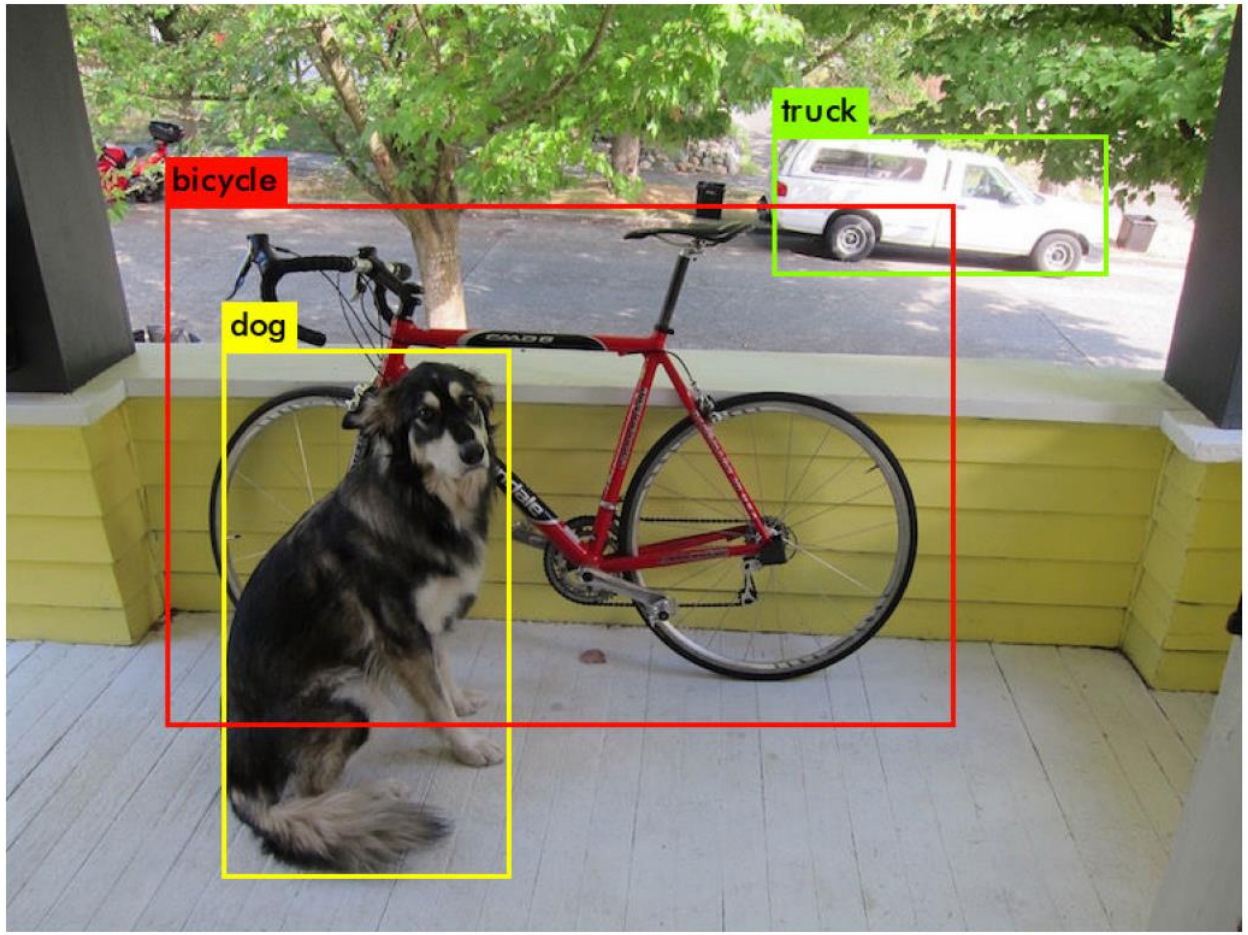
```
1 !./darknet detect cfg/yolo9000.cfg yolo9000.weights data/dog.jpg
```

```
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   544 x 544 x   3   ->   544 x 544 x  32  0.511 BFLOPs
    1 max          2 x 2 / 2   544 x 544 x  32   ->   272 x 272 x  32
    2 conv     64  3 x 3 / 1   272 x 272 x  32   ->   272 x 272 x  64  2.727 BFLOPs
    3 max          2 x 2 / 2   272 x 272 x  64   ->   136 x 136 x  64
    4 conv    128  3 x 3 / 1   136 x 136 x  64   ->   136 x 136 x 128  2.727 BFLOPs
    5 conv     64  1 x 1 / 1   136 x 136 x 128   ->   136 x 136 x  64  0.303 BFLOPs
    6 conv    128  3 x 3 / 1   136 x 136 x  64   ->   136 x 136 x 128  2.727 BFLOPs
    7 max          2 x 2 / 2   136 x 136 x 128   ->    68 x  68 x 128
    8 conv    256  3 x 3 / 1    68 x  68 x 128   ->    68 x  68 x 256  2.727 BFLOPs
    9 conv    128  1 x 1 / 1    68 x  68 x 256   ->    68 x  68 x 128  0.303 BFLOPs
   10 conv    256  3 x 3 / 1    68 x  68 x 128   ->    68 x  68 x 256  2.727 BFLOPs
   11 max          2 x 2 / 2    68 x  68 x 256   ->    34 x  34 x 256
   12 conv    512  3 x 3 / 1    34 x  34 x 256   ->    34 x  34 x 512  2.727 BFLOPs
   13 conv    256  1 x 1 / 1    34 x  34 x 512   ->    34 x  34 x 256  0.303 BFLOPs
   14 conv    512  3 x 3 / 1    34 x  34 x 256   ->    34 x  34 x 512  2.727 BFLOPs
   15 conv    256  1 x 1 / 1    34 x  34 x 512   ->    34 x  34 x 256  0.303 BFLOPs
   16 conv    512  3 x 3 / 1    34 x  34 x 256   ->    34 x  34 x 512  2.727 BFLOPs
   17 max          2 x 2 / 2    34 x  34 x 512   ->    17 x  17 x 512
   18 conv   1024  3 x 3 / 1    17 x  17 x 512   ->    17 x  17 x1024  2.727 BFLOPs
   19 conv    512  1 x 1 / 1    17 x  17 x1024   ->    17 x  17 x 512  0.303 BFLOPs
   20 conv   1024  3 x 3 / 1    17 x  17 x 512   ->    17 x  17 x1024  2.727 BFLOPs
   21 conv    512  1 x 1 / 1    17 x  17 x1024   ->    17 x  17 x 512  0.303 BFLOPs
   22 conv   1024  3 x 3 / 1    17 x  17 x 512   ->    17 x  17 x1024  2.727 BFLOPs
   23 conv  28269  1 x 1 / 1    17 x  17 x1024   ->    17 x  17 x28269  16.732 BFLOPs
   24 detection
mask_scale: Using default '1.000000'
Loading weights from yolo9000.weights...Done!
data/dog.jpg: Predicted in 7.976258 seconds.
```

Result

Code
(YOLOv3)

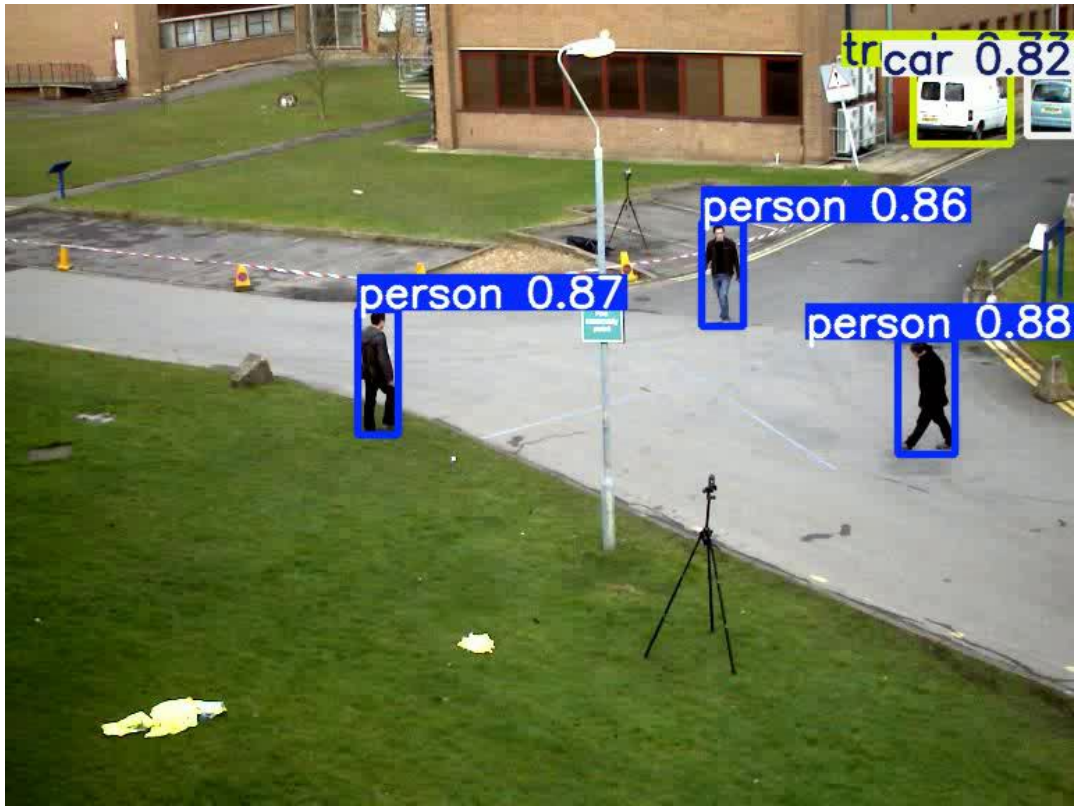
```
[8]  1 import cv2
     2 import matplotlib.pyplot as plt
     3
     4 def show(path):
     5   img = cv2.imread(path)
     6   fig = plt.gcf()
     7   fig.set_size_inches(18, 10)
     8   plt.axis("off")
     9   plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

     1 show('predictions.jpg')
```



Result

Video data

(YOLOv3)



Result

Video data

(YOLOv3)

Result

Image data
(YOLOv5)

```
[3]  1 import torch
     2
     3 model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
```

```
Using cache found in /root/.cache/torch/hub/ultralytics_yolov5_master
YOLOv5 🚀 2024-7-22 Python-3.10.12 torch-2.3.1+cu121 CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
Adding AutoShape...
```

```
[14]  1 import cv2
      2
      3 image_path = 'korea.jpg'
      4 image = cv2.imread(image_path)
      5
      6
      7 results = model(image)
      8 image = results.render()[0]
      9 cv2.imwrite('output_image.jpg', image)
```

```
True
```

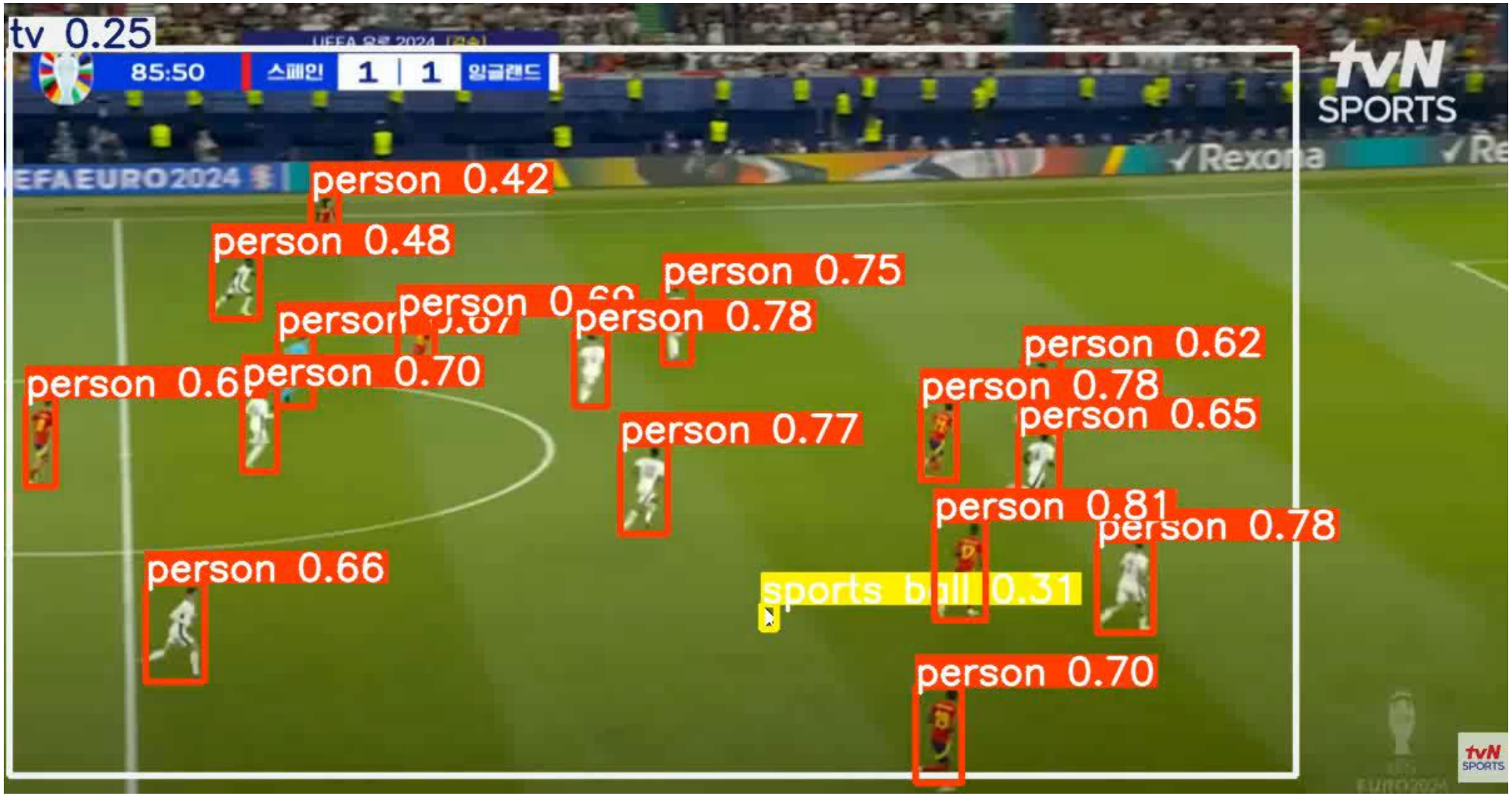
Result

Video data (YOLOv5)

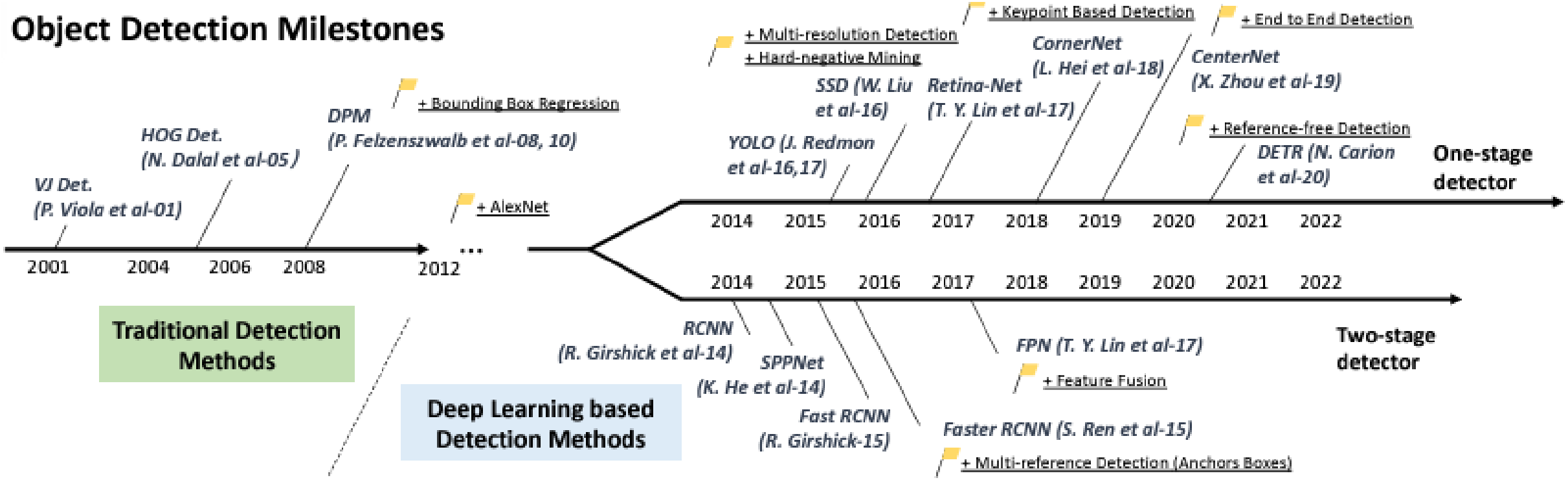
```
[8]
import cv2

video_path = 'EU2024_final.mp4'
cap = cv2.VideoCapture(video_path)

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output_video.mp4', fourcc, 30.0, (int(cap.get(3)), int(cap.get(4))))

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame)
    frame = results.render()[0]
    out.write(frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

Result

Video data



Summary


Object Detection Milestones
VJ Det.
(P. Viola et al-01)
HOG Det.
(N. Dalal et al-05)
DPM
(P. Felzenszwalb et al-08, 10)
+ Bounding Box Regression
+ AlexNet
2001
2004
2006
2008
2012
Traditional Detection Methods
Deep Learning based Detection Methods
+ Multi-resolution Detection
+ Hard-negative Mining
+ Keypoint Based Detection
+ End to End Detection
YOLO (J. Redmon et al-16,17)
SSD (W. Liu et al-16)
Retina-Net (T. Y. Lin et al-17)
CornerNet (L. Hei et al-18)
CenterNet (X. Zhou et al-19)
+ Reference-free Detection
DETR (N. Carion et al-20)
One-stage detector
2014
2015
2016
2017
2018
2019
2020
2021
2022
RCNN (R. Girshick et al-14)
SPPNet (K. He et al-14)
Fast RCNN (R. Girshick-15)
Faster RCNN (S. Ren et al-15)
FPN (T. Y. Lin et al-17)
+ Feature Fusion
+ Multi-reference Detection (Anchors Boxes)
Two-stage detector

Reference

- [YOLOv1] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.「**You Only Look Once: Unified, Real-Time Object Detection**」, 2015
- [YOLOv2] Joseph Redmon, Ali Farhadi.「**YOLO9000: Better, Faster, Stronger**」, 2016
- [YOLOv3] Joseph Redmon, Ali Farhadi.「**YOLOv3: An Incremental Improvement**」, 2018
- [SSD] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg.「**SSD: Single Shot MultiBox Detector**」, 2016
- Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun.「**Faster R-CNN: Toward Real-Time Object Detection with Region Proposal Networks**」**,** 2015
- [RetinaNet] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollar.「**Focal Loss for Dense Object Detection**」, 2018
- Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, Jeiping Yi.「**Object Detection in 20 Years: A Survey**」, 2019
- https://pjreddie.com/darknet/yolo/
- https://herbwood.tistory.com/13
- https://aigaeddo.tistory.com/47
- https://github.com/weiliu89/caffe/tree/ssd
- https://process-mining.tistory.com/155
- 컴퓨터비전-11. YOLO v1, v2, v3 개요와 실습
- https://qiita.com/ashitani/items/566cf9234682cb5f2d60

THANK YOU