Object detection

Valeriya Strizhkova
25.01.23

About myself

Valeriya Strizhkova

valeriya.strizhkova@inria.fr

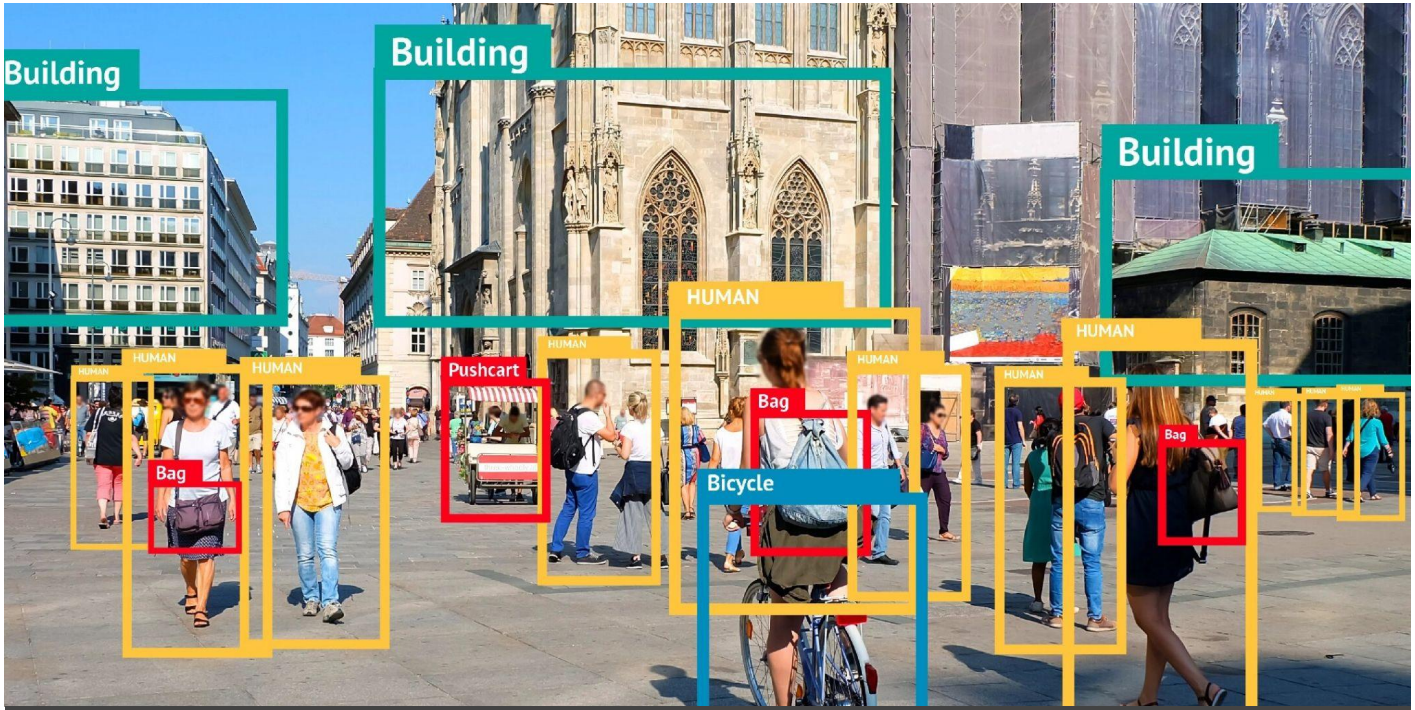
2nd year PhD student @ Inria & 3iA & UCA

Research topics: emotion detection, computer vision, deep learning

Overview

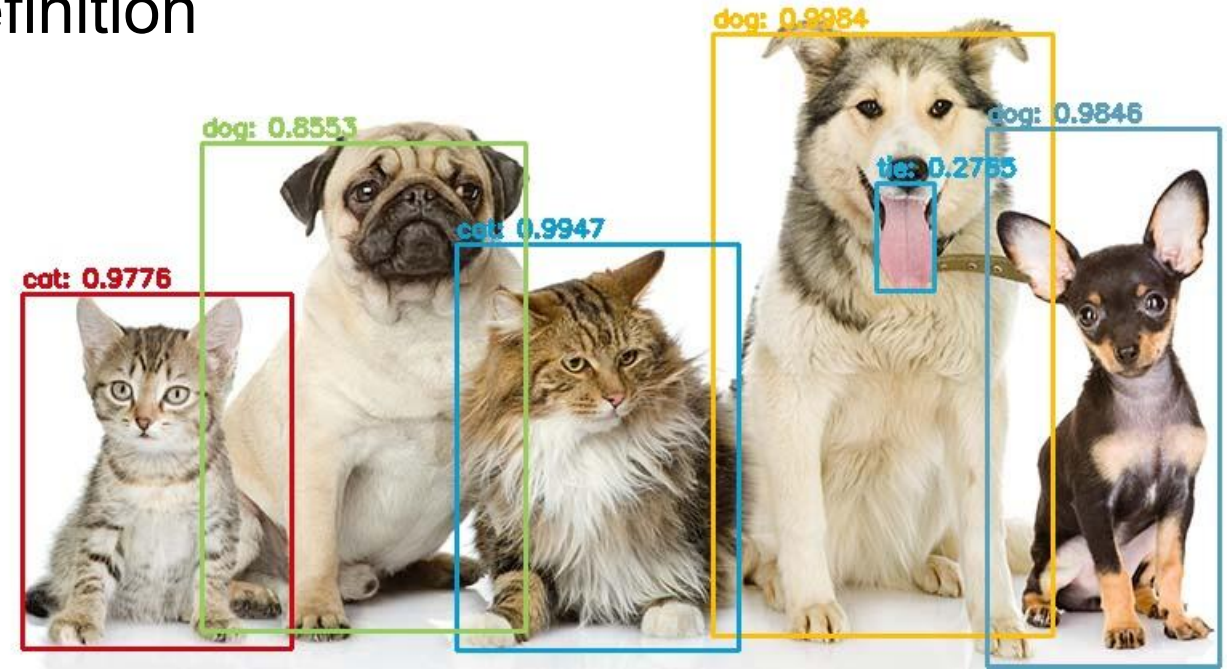
- **Introduction**
- Performance evaluation
- Two-stage (with proposal) object detectors
  - R-CNN
  - Fast R-CNN
  - Faster R-CNN
- One-stage (proposal-free) object detectors
  - YOLO
  - DETR

Object detection



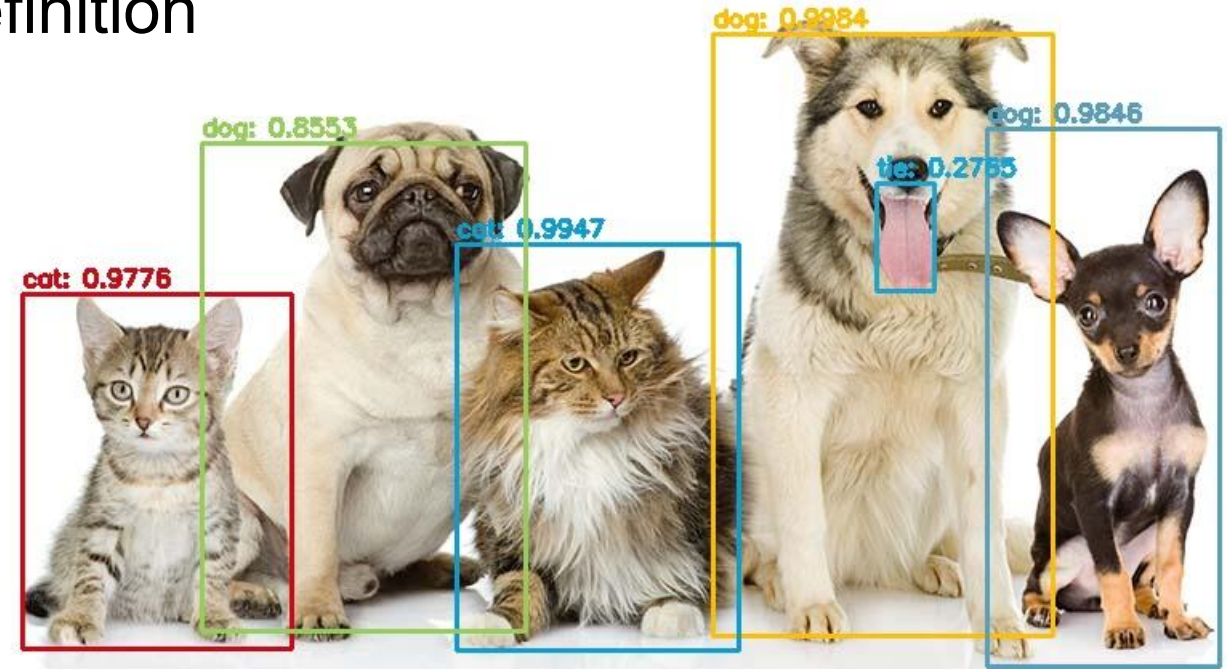
Goal: Localize (Bounding Box) and classify all objects in the image

Task definition



- Input: RGB image
- Output: set of bounding boxes with category label and confidence

Task definition

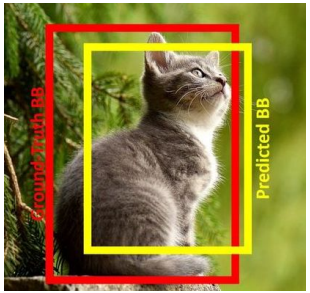


- **Input**: RGB image
- **Output**: set of bounding boxes with category label and confidence
- The number of objects is not known

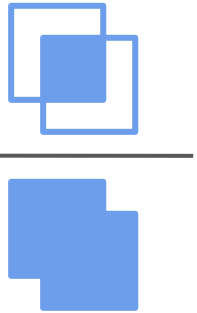
Overview

- Introduction
- **Performance evaluation**
- Two-stage (with proposal) object detectors
  - R-CNN
  - Fast R-CNN
  - Faster R-CNN
- One-stage (proposal-free) object detectors
  - YOLO
  - DETR

Performance evaluation. Intersection over Union (IoU)

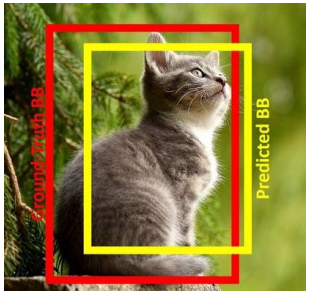


$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} =$$

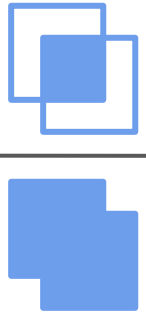


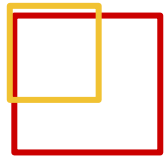
predicted bbox vs true bbox

Performance evaluation. Intersection over Union (IoU)



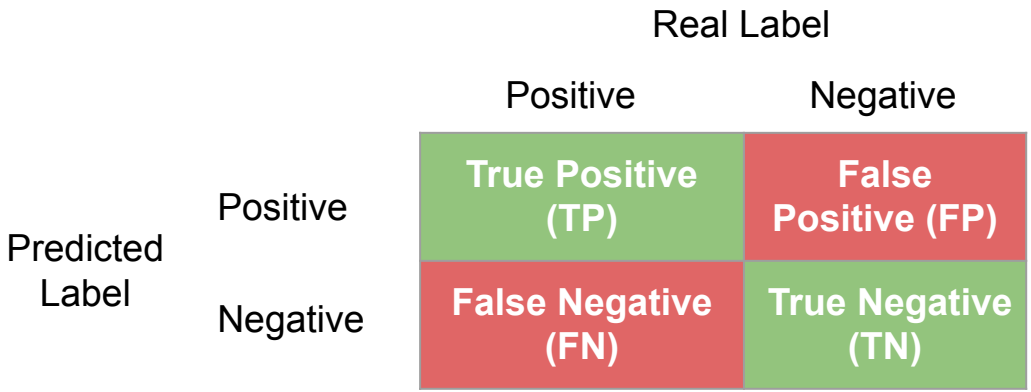
$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{\quad}{\quad}$$



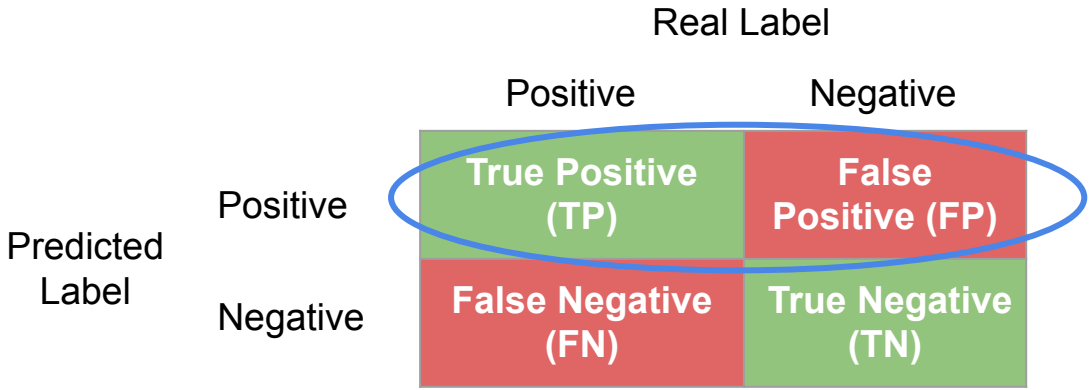


predicted bbox vs true bbox

Performance evaluation

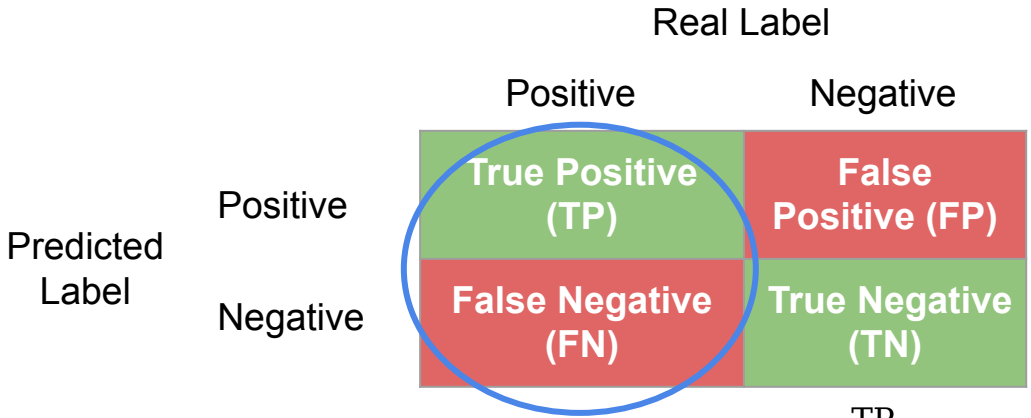


Performance evaluation



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

Performance evaluation



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

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Performance evaluation

		Real Label: Positive	Real Label: Negative
Predicted Label	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

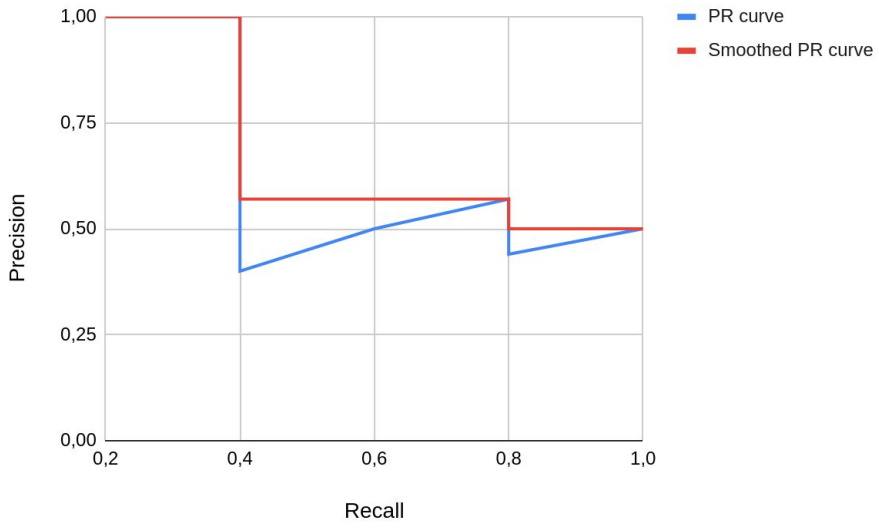
$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{total positive results}}$$

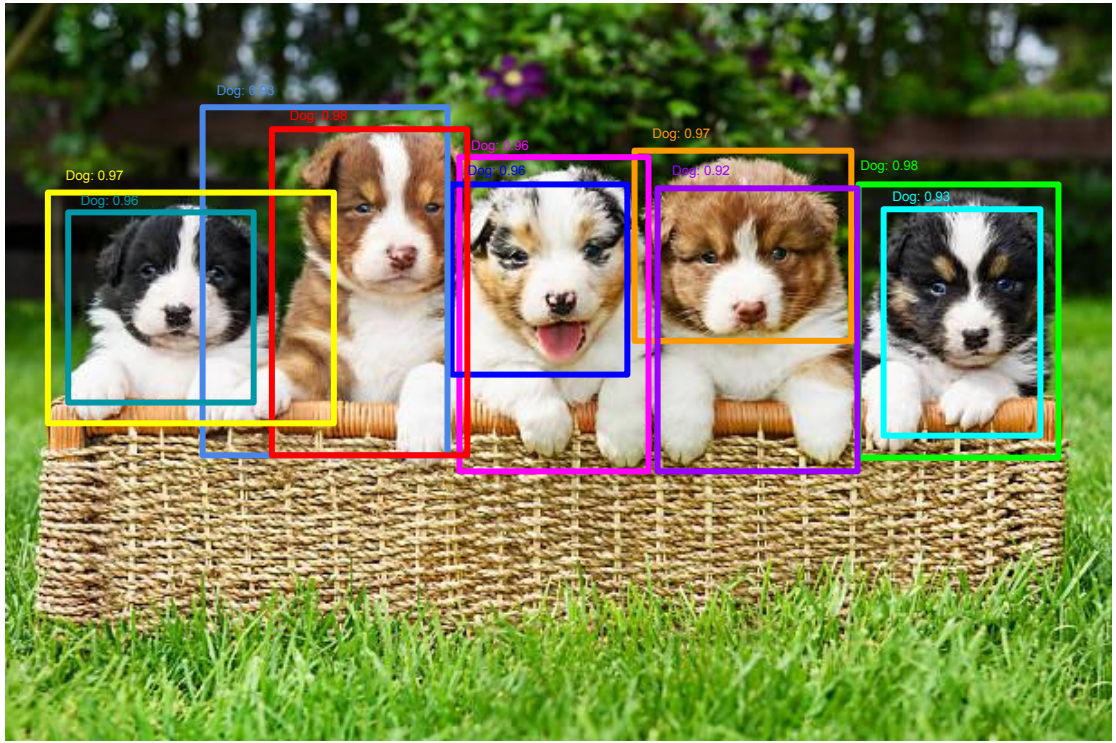
$$\text{Recall} = \frac{\text{TP}}{\text{total disease cases}}$$

Performance evaluation: Average Precision



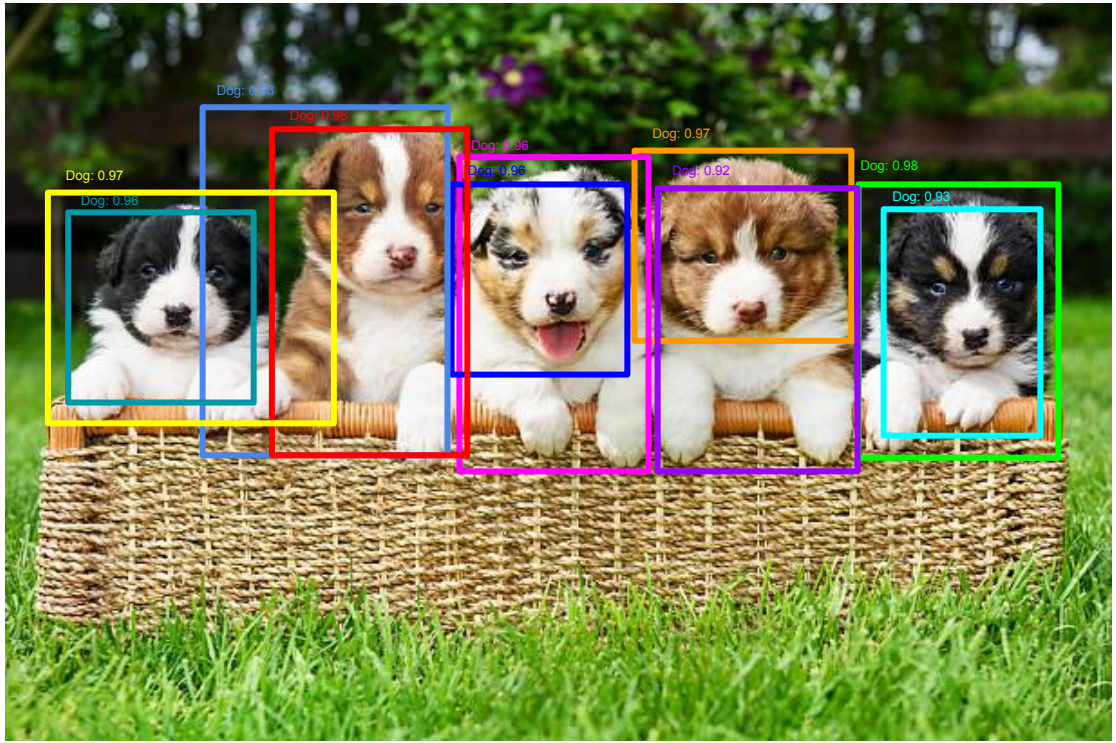
Performance evaluation: Average Precision

Performance evaluation: Average Precision



Performance evaluation: Average Precision

Rank	Correct?
1	True
2	True
3	False
4	False
5	False
6	True
7	True
8	False
9	False
10	True



Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} = \frac{2}{3} = 0.67$$

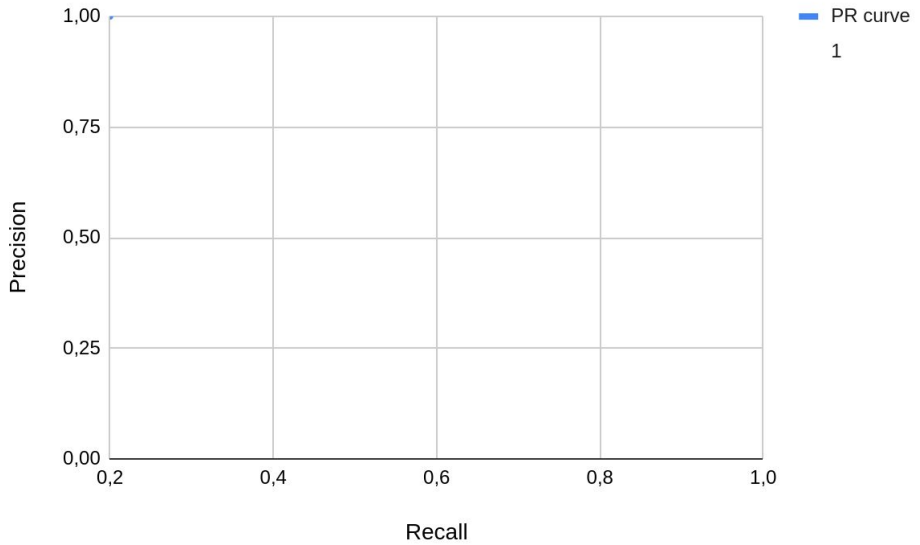
$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}} = \frac{2}{5} = 0.4$$

Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑

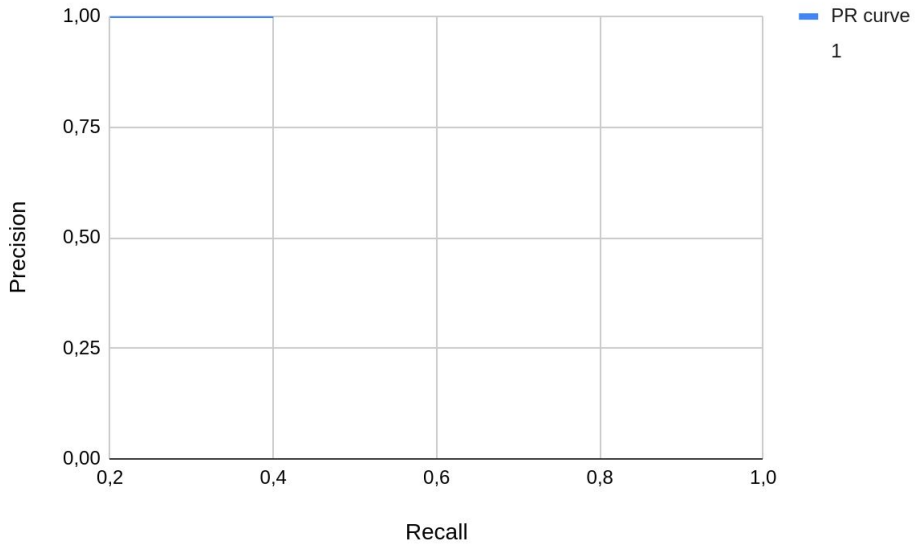
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	**True**	**1.0**	**0.2**
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



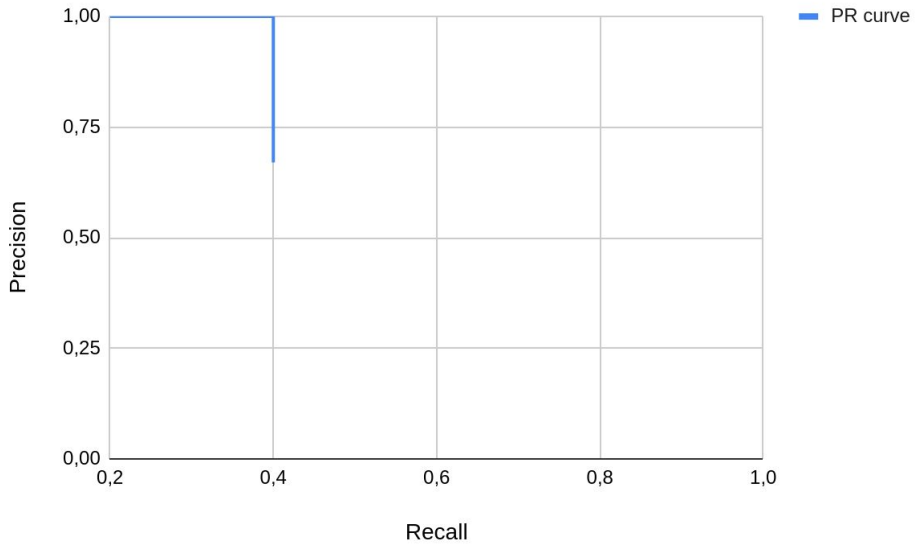
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	**True**	**1.0** —	**0.4** ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



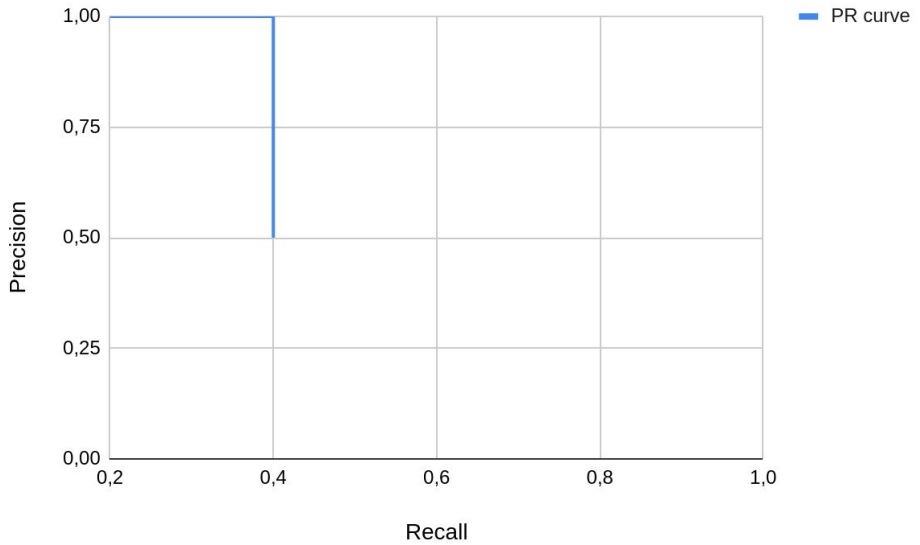
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	**False**	**0.67** ↓	**0.4** —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



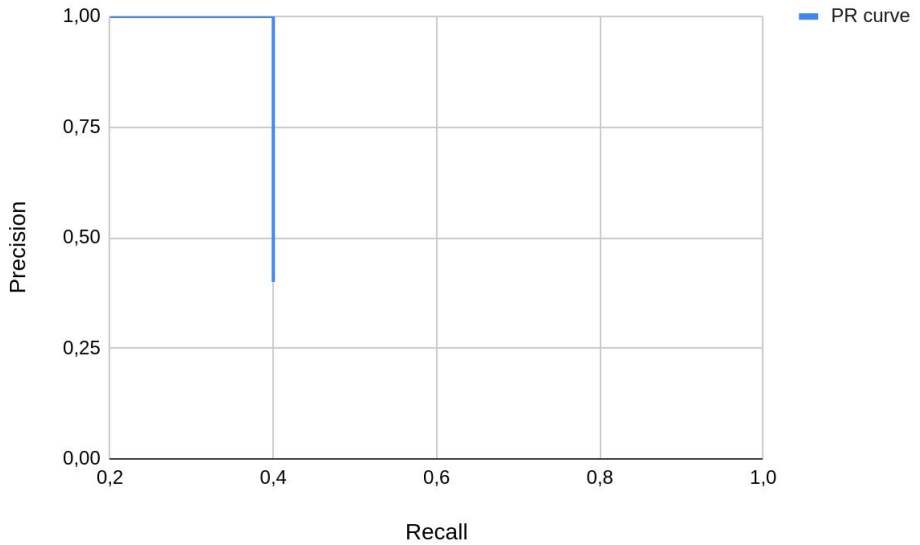
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	**False**	**0.5** ↓	**0.4** —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



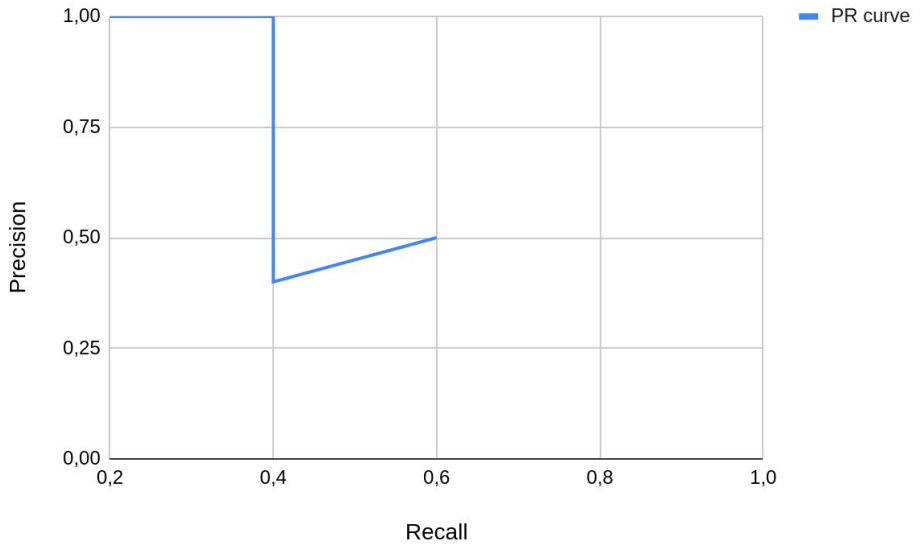
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	**False**	**0.4** ↓	**0.4** —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



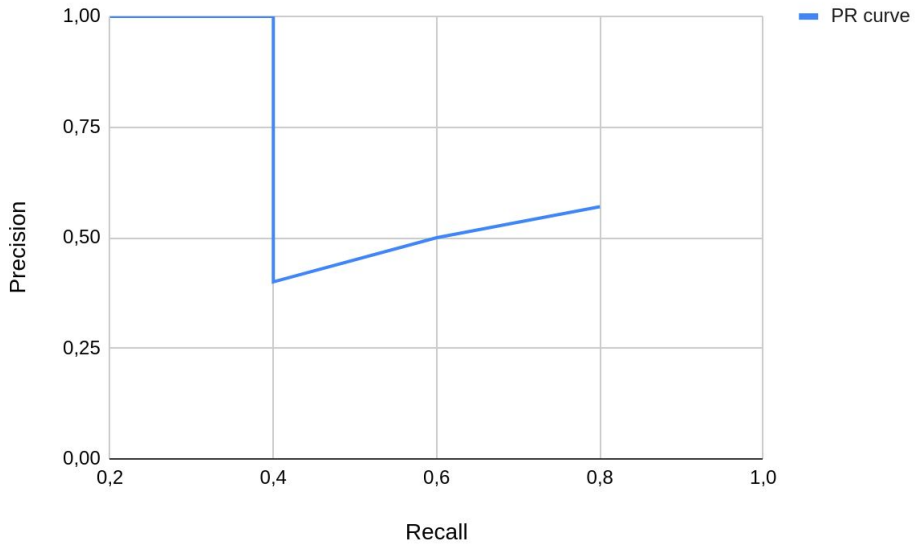
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	**True**	**0.5** ↑	**0.6** ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



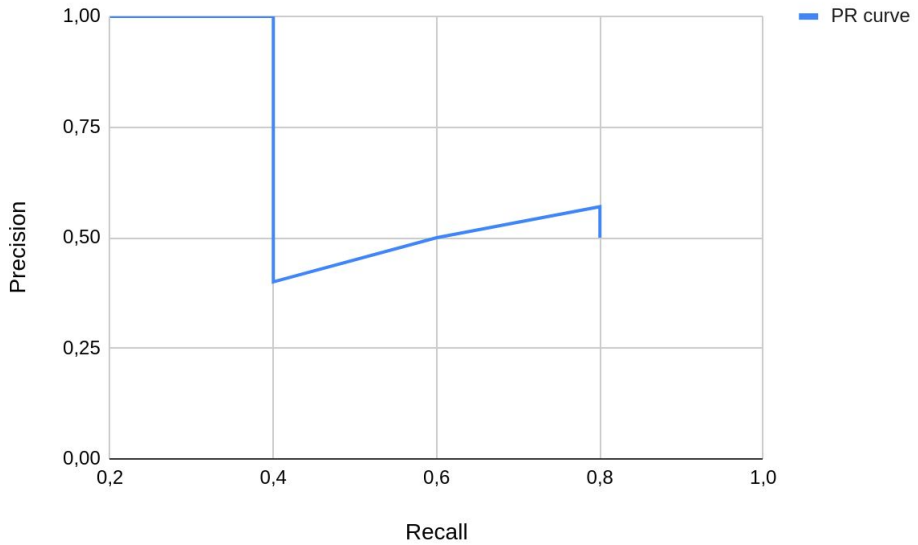
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	**True**	**0.57** ↑	**0.8** ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



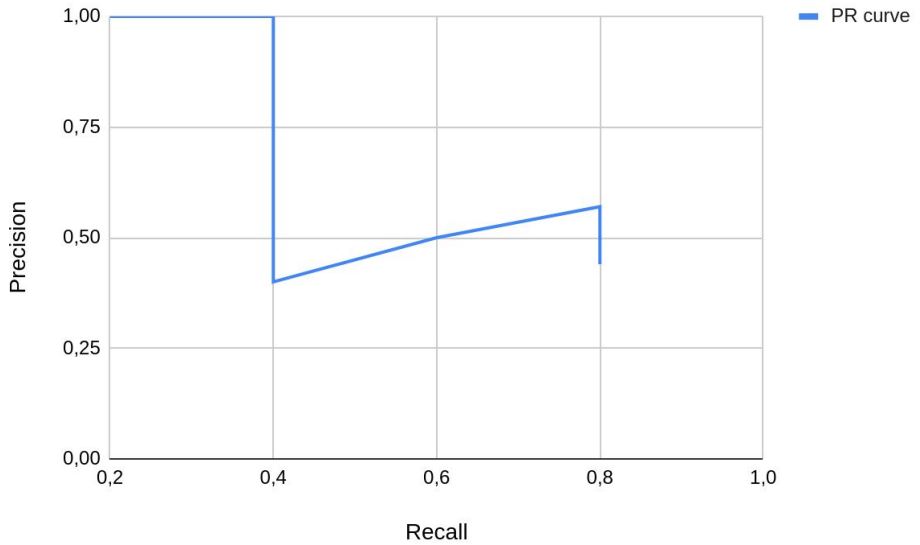
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	**False**	**0.5** ↓	**0.8** —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



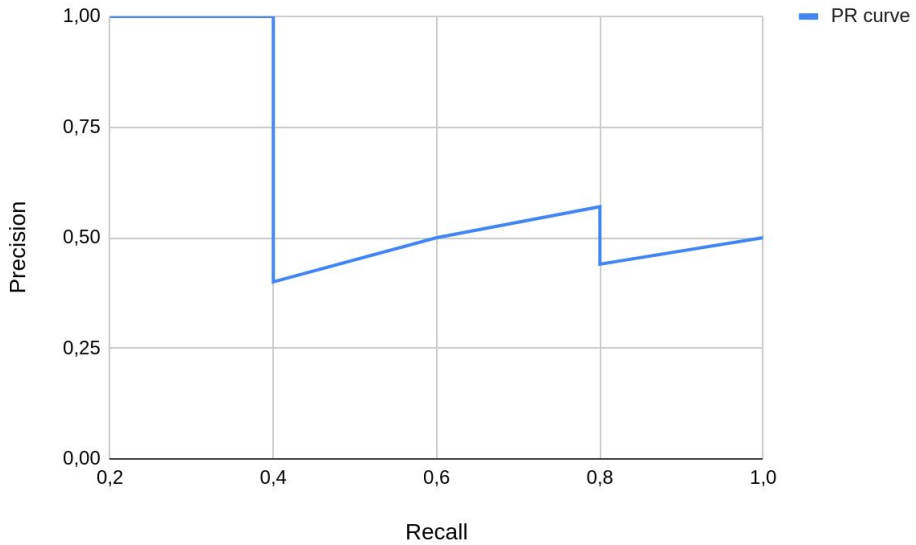
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	**False**	**0.44** ↓	**0.8** —
10	True	0.5 ↑	1.0 ↑



Performance evaluation: Average Precision

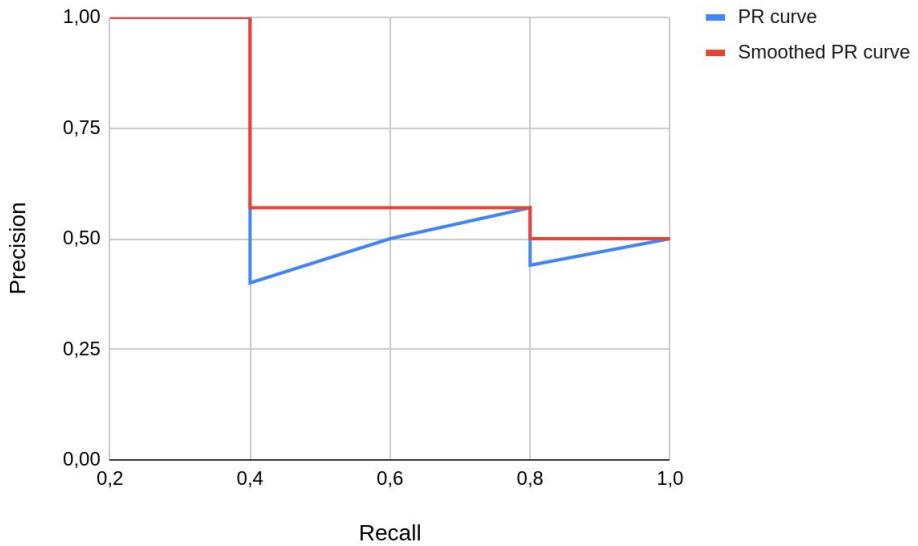
Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	**True**	**0.5** ↑	**1.0** ↑



$$AP = \int_0^1 p(r)dr$$

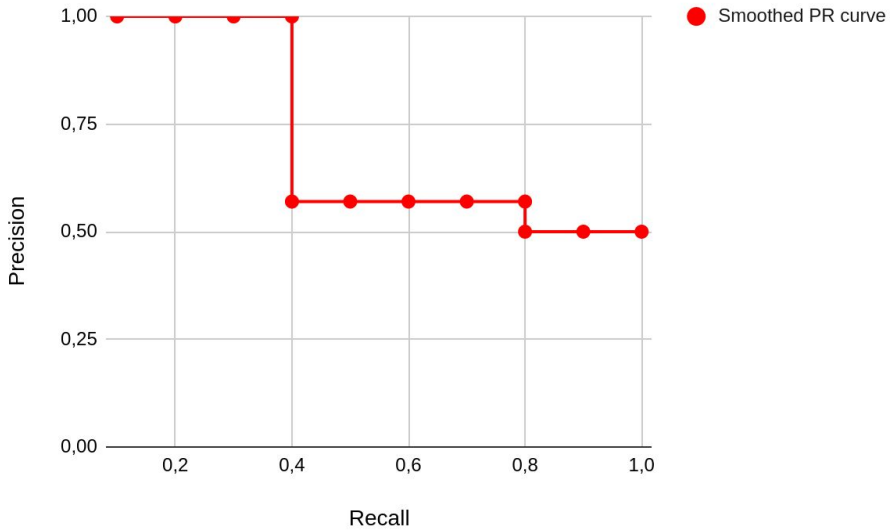
Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



Performance evaluation: Average Precision

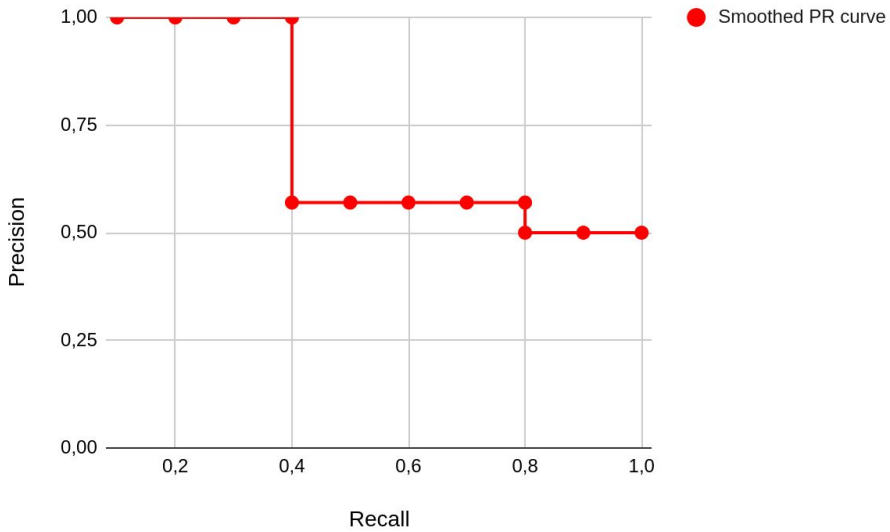
Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑



$$AP = \frac{1}{11} \times \sum_{r \in 0.0, 0.1, \ldots, 1.0} p(r) = \frac{1}{11} \times (p(0) + p(0.1) + \ldots + p(1.0))$$

Performance evaluation: Average Precision

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0 —	0.4 ↑
3	False	0.67 ↓	0.4 —
4	False	0.5 ↓	0.4 —
5	False	0.4 ↓	0.4 —
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑
8	False	0.5 ↓	0.8 —
9	False	0.44 ↓	0.8 —
10	True	0.5 ↑	1.0 ↑

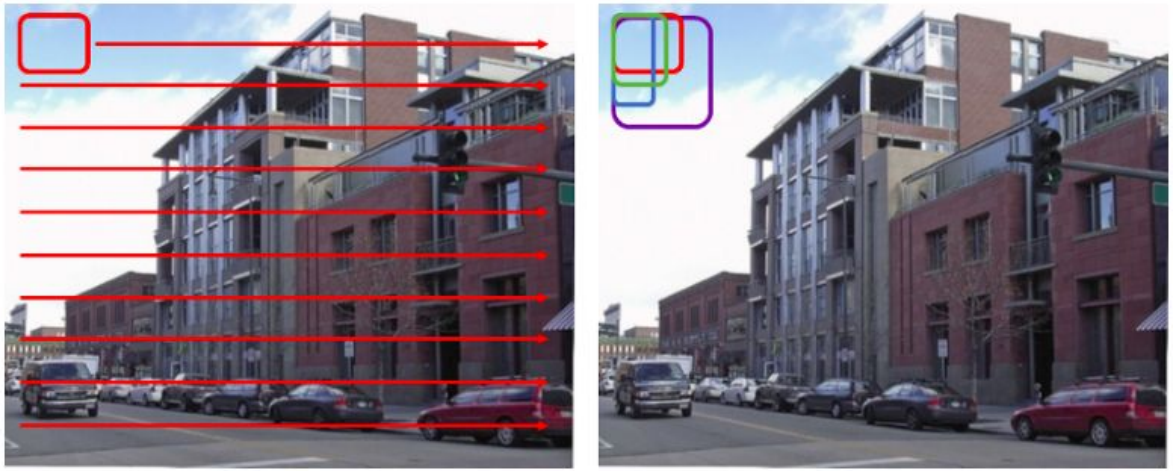


$$AP = \frac{1}{11} \times \sum_{r \in 0.0, 0.1, \ldots, 1.0} p(r) = \frac{1}{11} \times (p(0) + p(0.1) + \ldots + p(1.0))$$

$$AP = \frac{1}{11} \times (5 \times 1.0 + 4 \times 0.57 + 2 \times 0.5) = 0.75$$

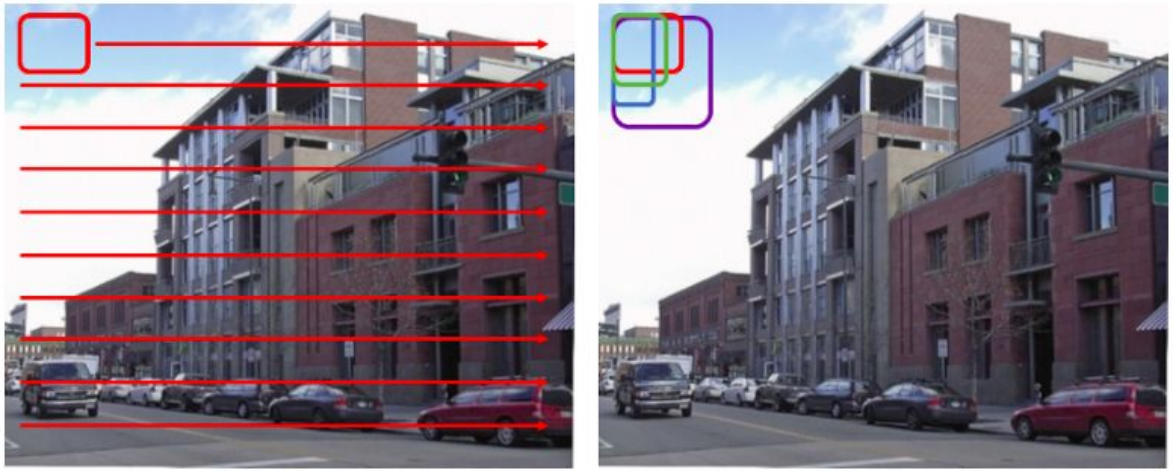
Sliding Window Object Detection

Sliding Window Object Detection



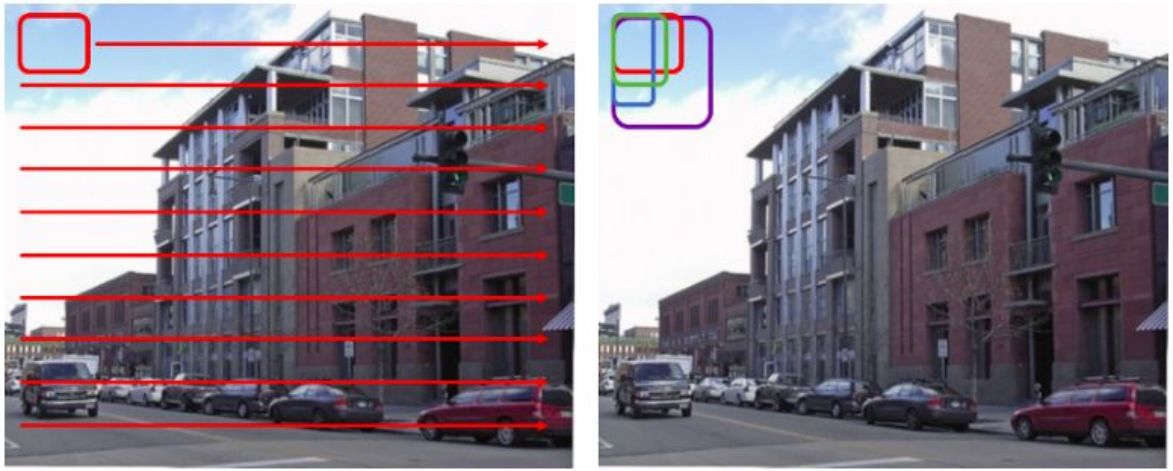
- Run **sliding window** of fixed size over the image and extract features for each image

Sliding Window Object Detection



- Run **sliding window** of fixed size over the image and extract features for each image
- **Classify each crop**

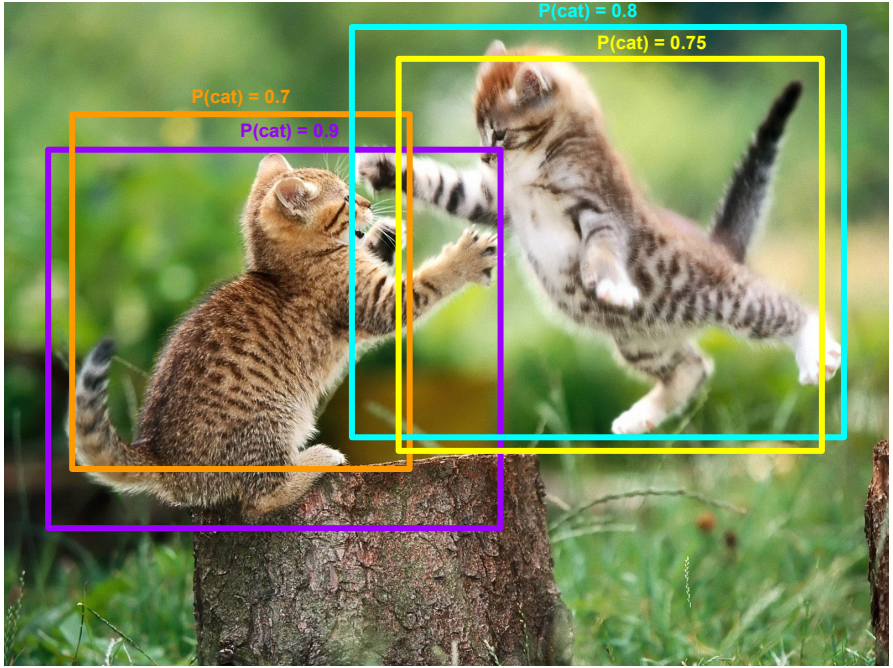
Sliding Window Object Detection



- Run **sliding window** of fixed size over the image and extract features for each image
- **Classify each crop**
- Iterate over multiple aspect ratios and scales

Overlapping Boxes: Non-Max Suppression (NMS)

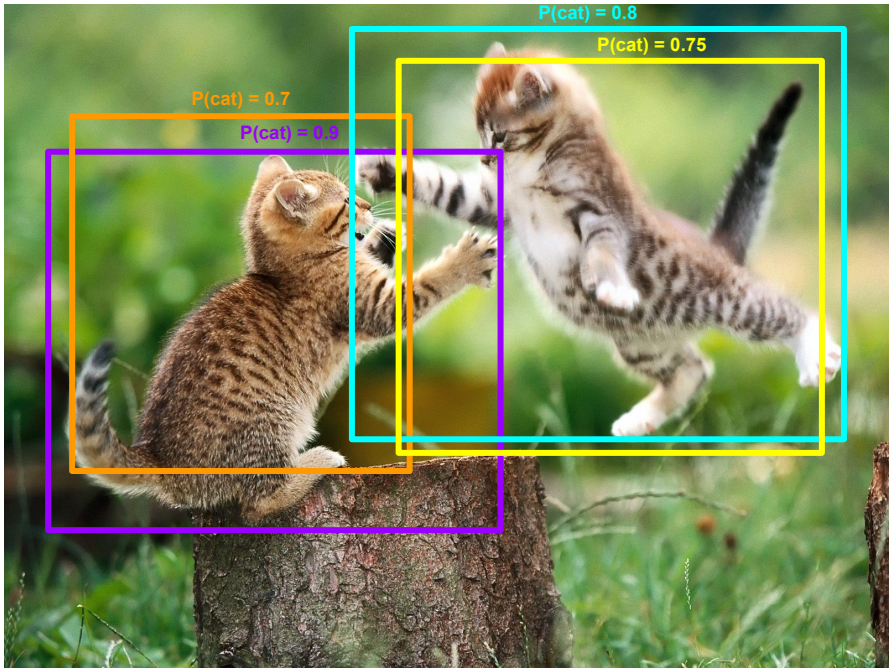
Problem: Object detectors often output many overlapping detections.



Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections.

Solution: Post-process raw detections using **Non-Max Suppression (NMS)**

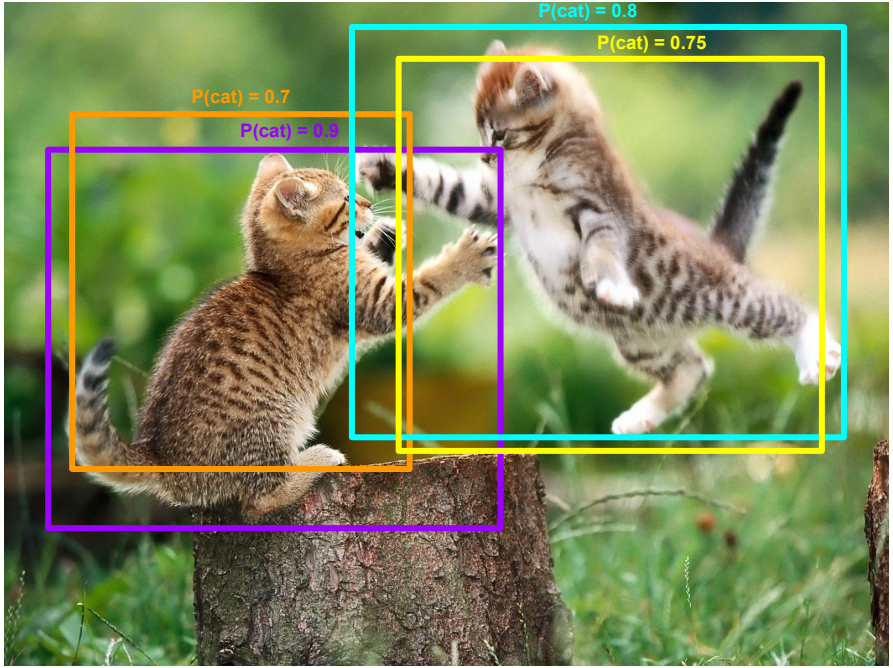


Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections.

Solution: Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1



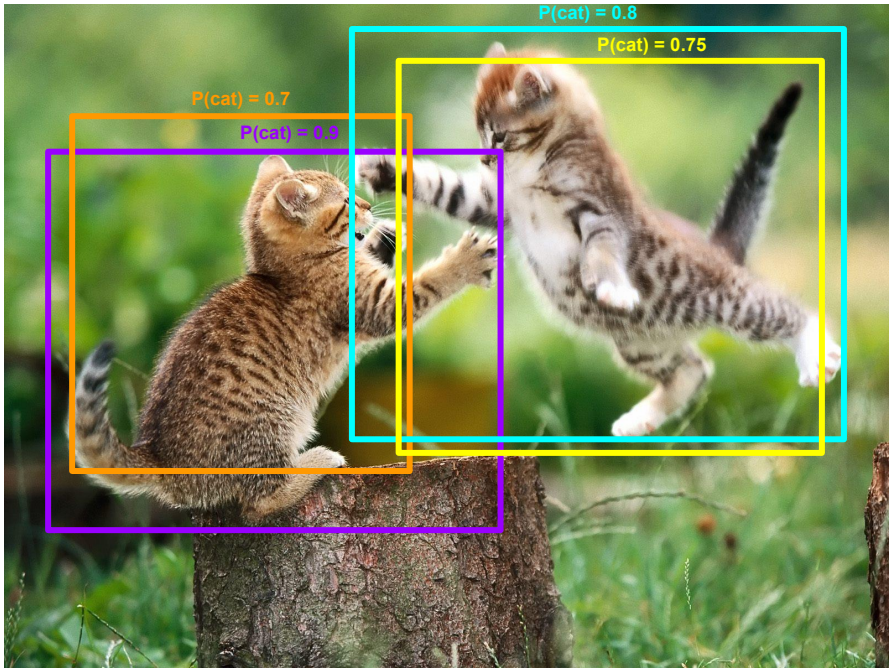
Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections.

Solution: Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(■, ■) = **0.8**
IoU(■, ■) = 0.08
IoU(■, ■) = 0.01



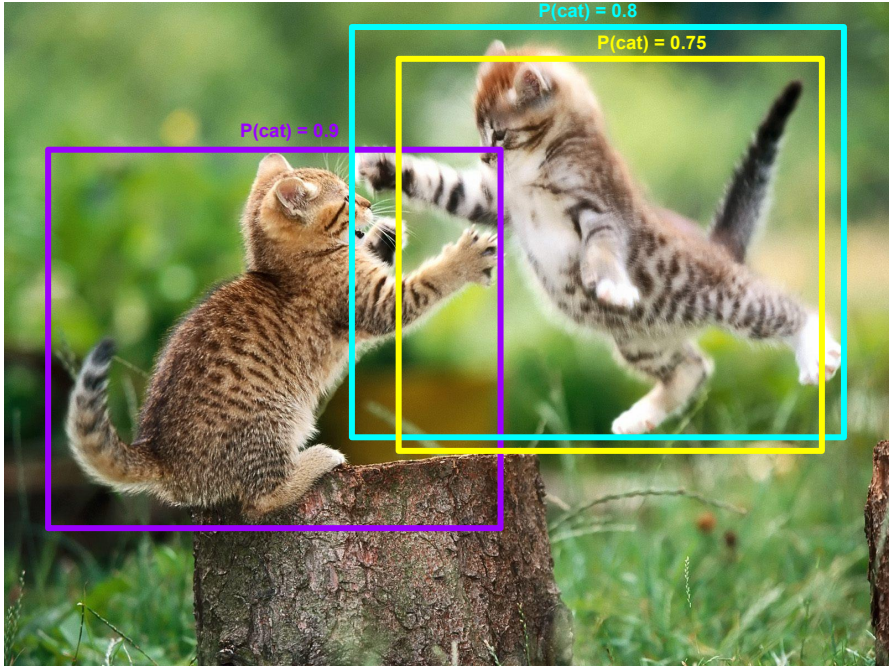
Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections.

Solution: Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(■, ■) = **0.95**



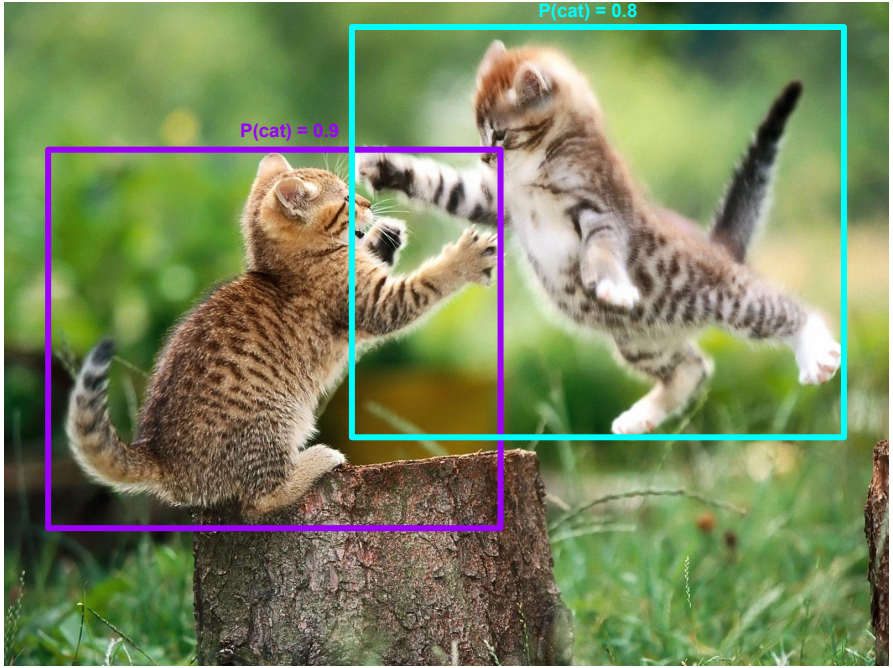
Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections.

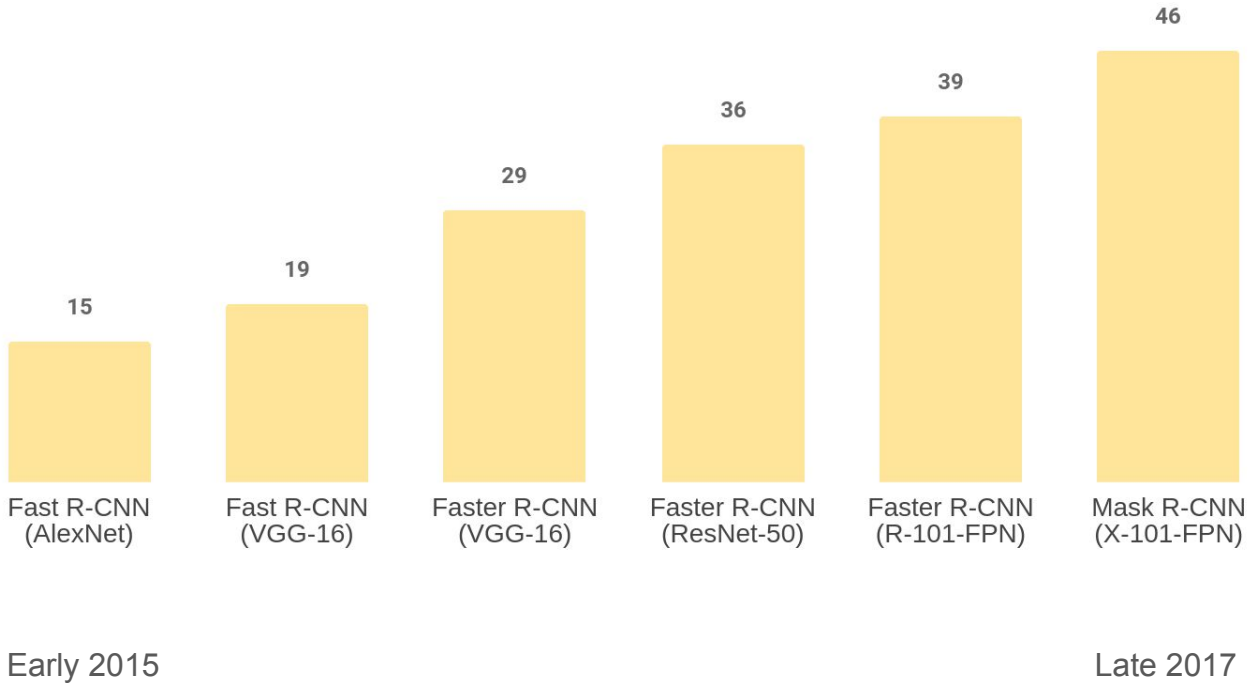
Solution: Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(,) = **0.95**



COCO Object Detection Average Precision (%)



Object Detection with Deep Neural Networks
Slide Credits: Ross Girshick

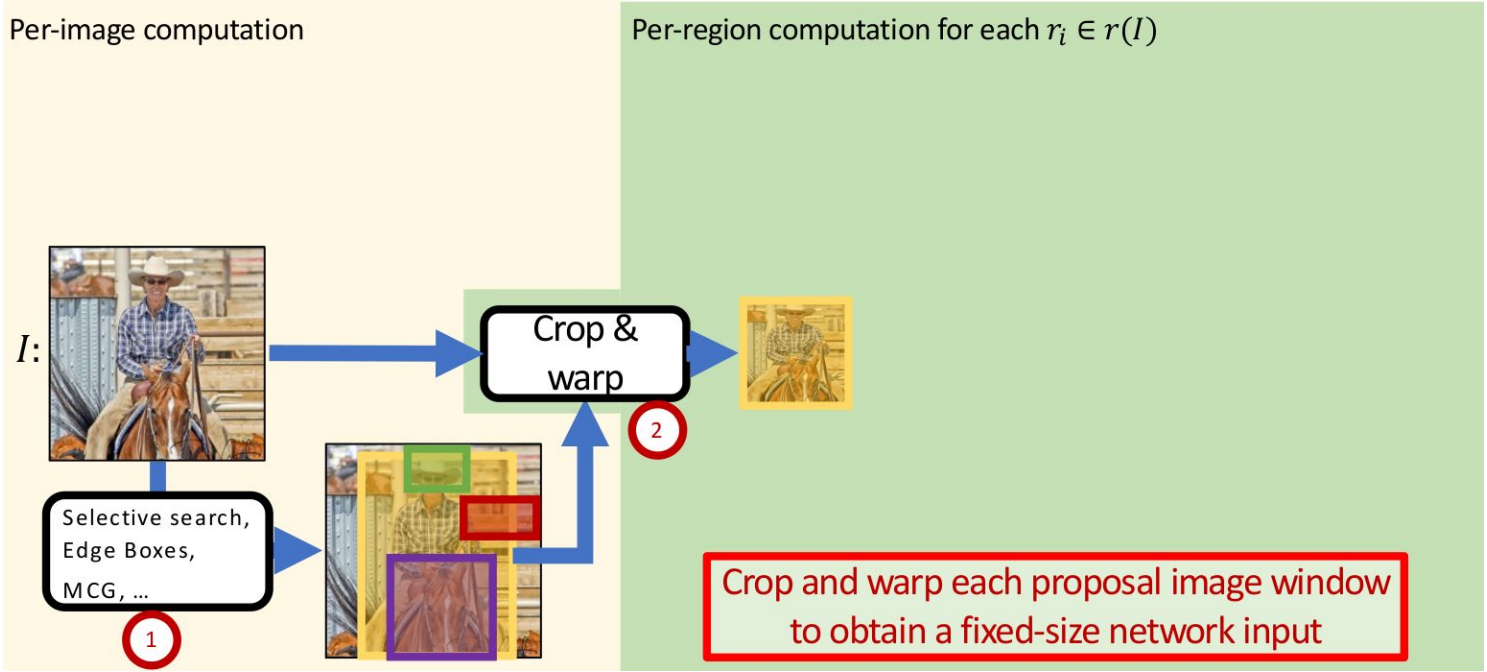
Overview

- Introduction
- Performance evaluation
- **Two-stage (with proposal) object detectors**
  - R-CNN
  - Fast R-CNN
  - Faster R-CNN
- One-stage (proposal-free) object detectors
  - YOLO
  - DETR

R-CNN [1]: Region-based Convolutional Neural Network

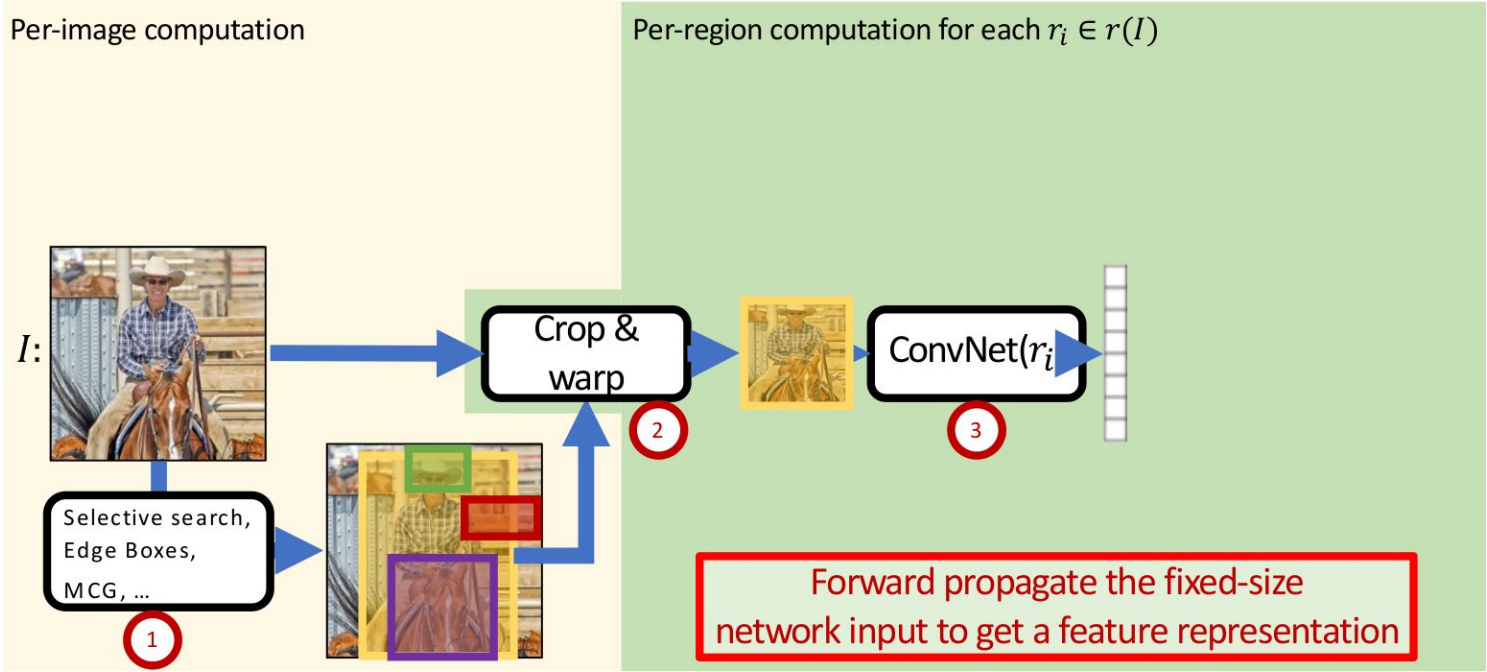
[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN [1]: Region-based Convolutional Neural Network



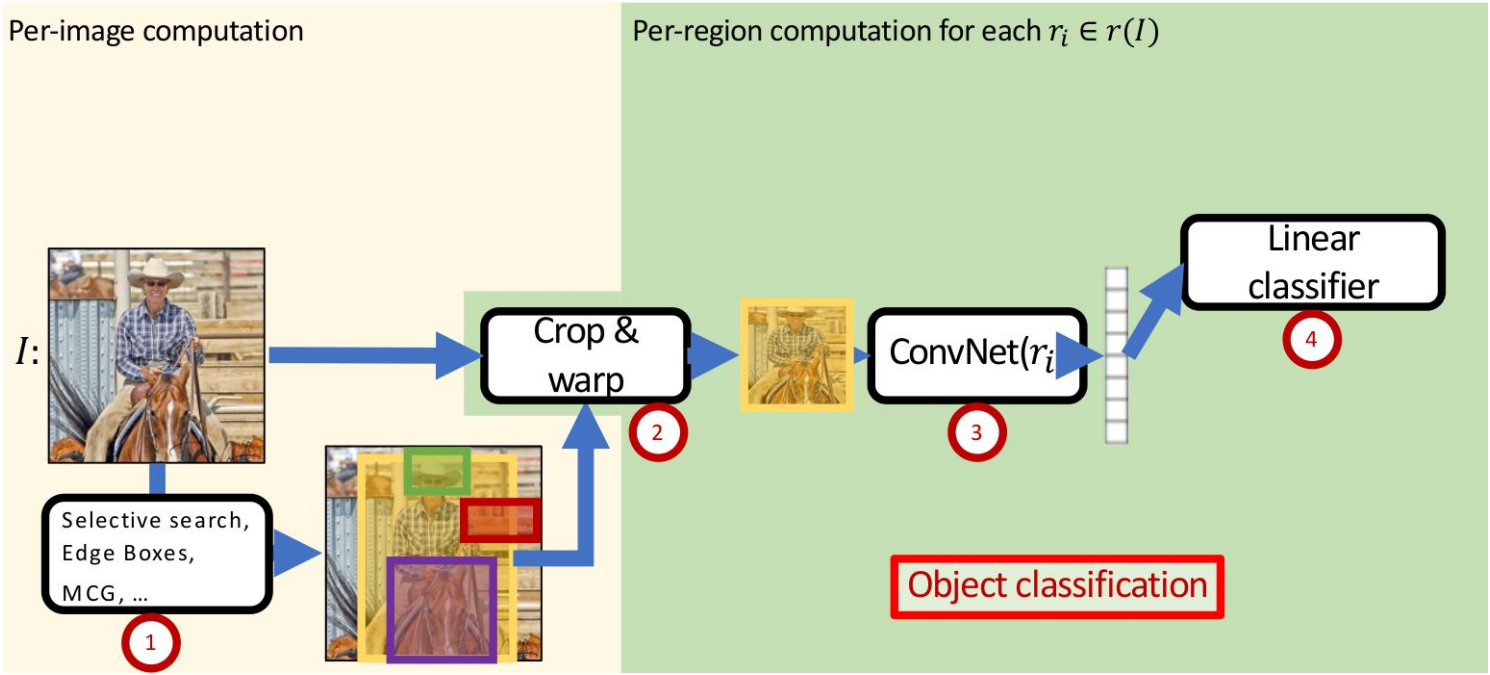
[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN [1]: Region-based Convolutional Neural Network



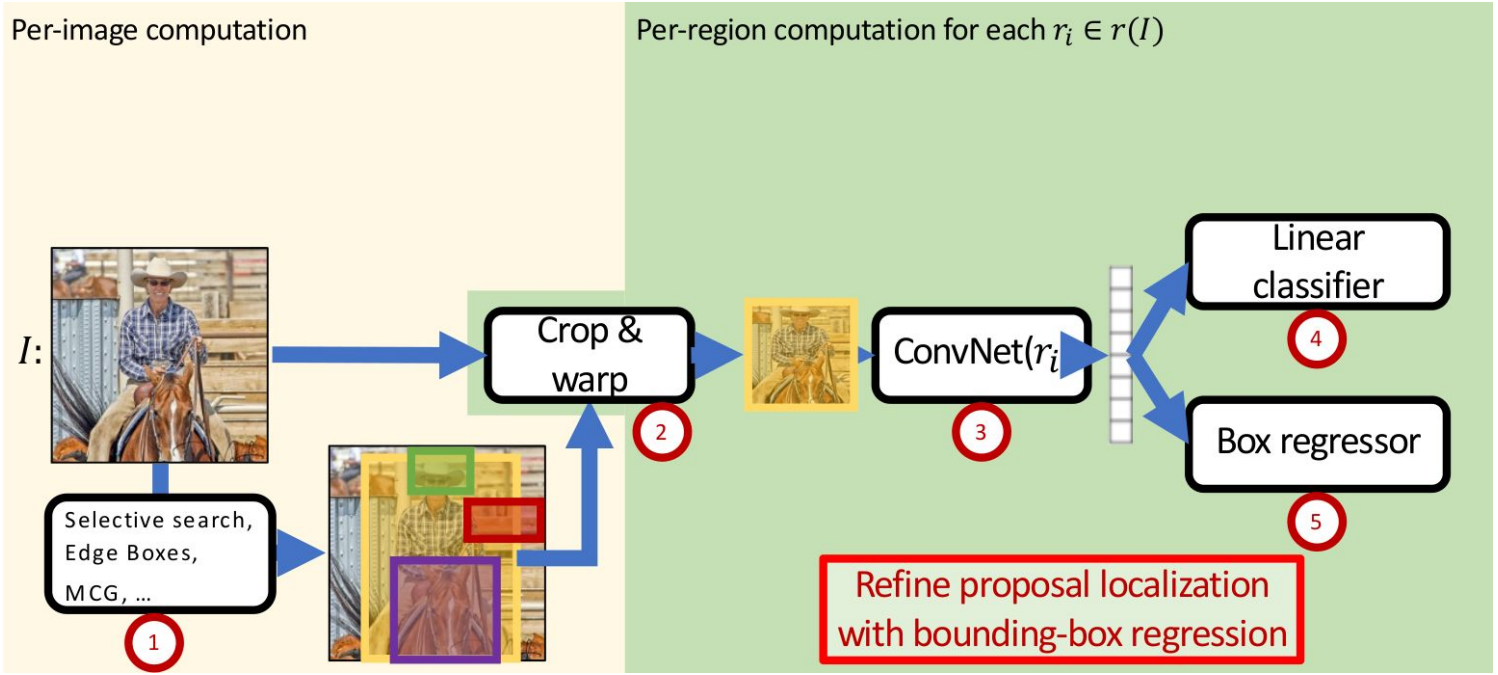
[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN [1]: Region-based Convolutional Neural Network



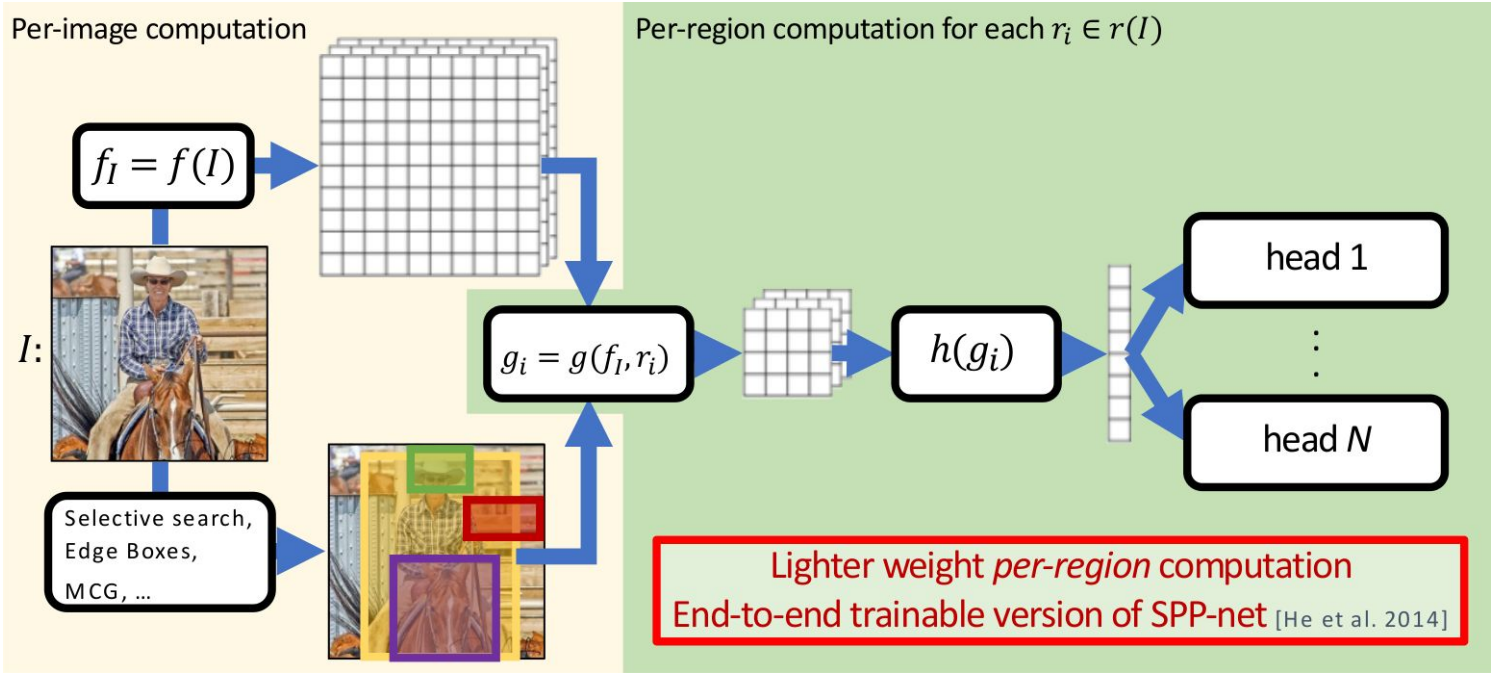
[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN [1]: Region-based Convolutional Neural Network

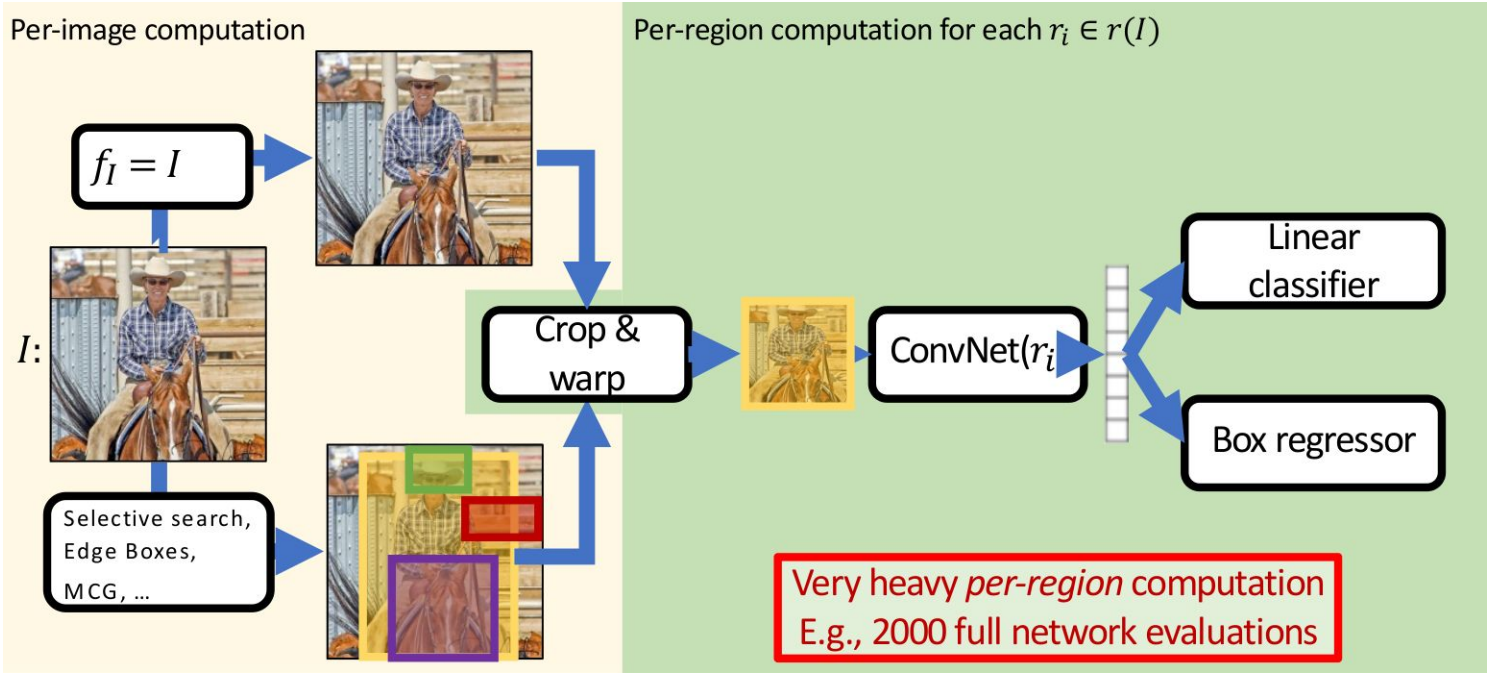


[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Generalized framework



R-CNN [1] in the Generalized Framework



[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN: Problems

- Heavy per-region computation (2000 full network evaluations)

[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN: Problems

- Heavy per-region computation (2000 full network evaluations)
- No computation / feature sharing

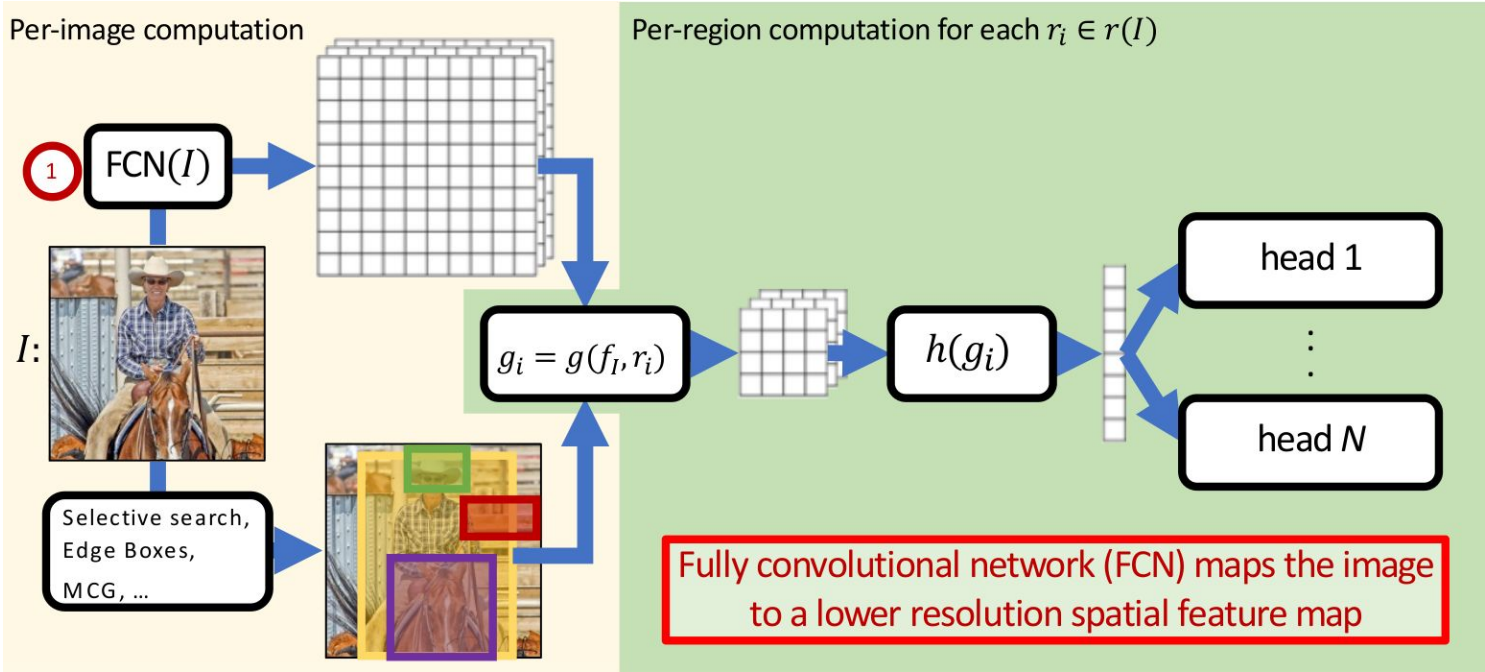
[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN: Problems

- Heavy per-region computation (2000 full network evaluations)
- No computation / feature sharing
- Slow region proposal method adds to runtime

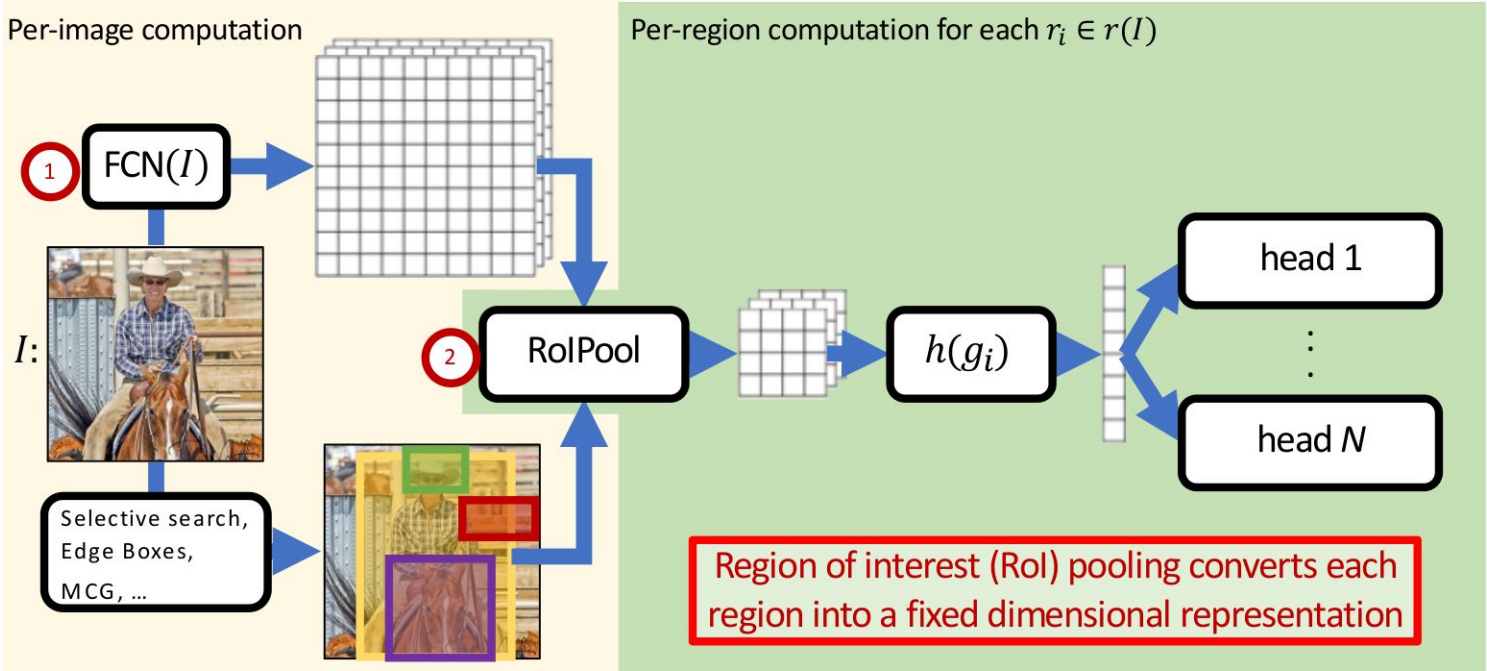
[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Fast R-CNN [1]



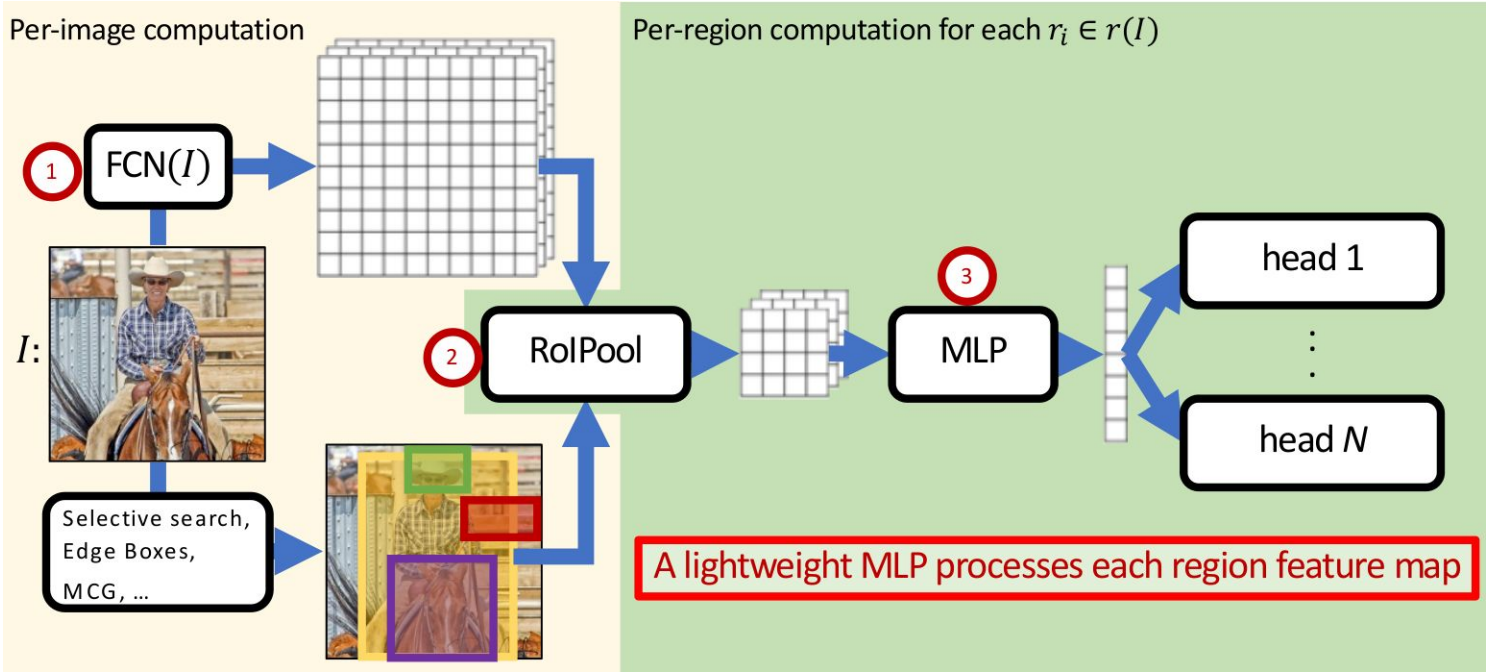
[1] Girshick. Fast R-CNN. ICCV, 2015.

Fast R-CNN



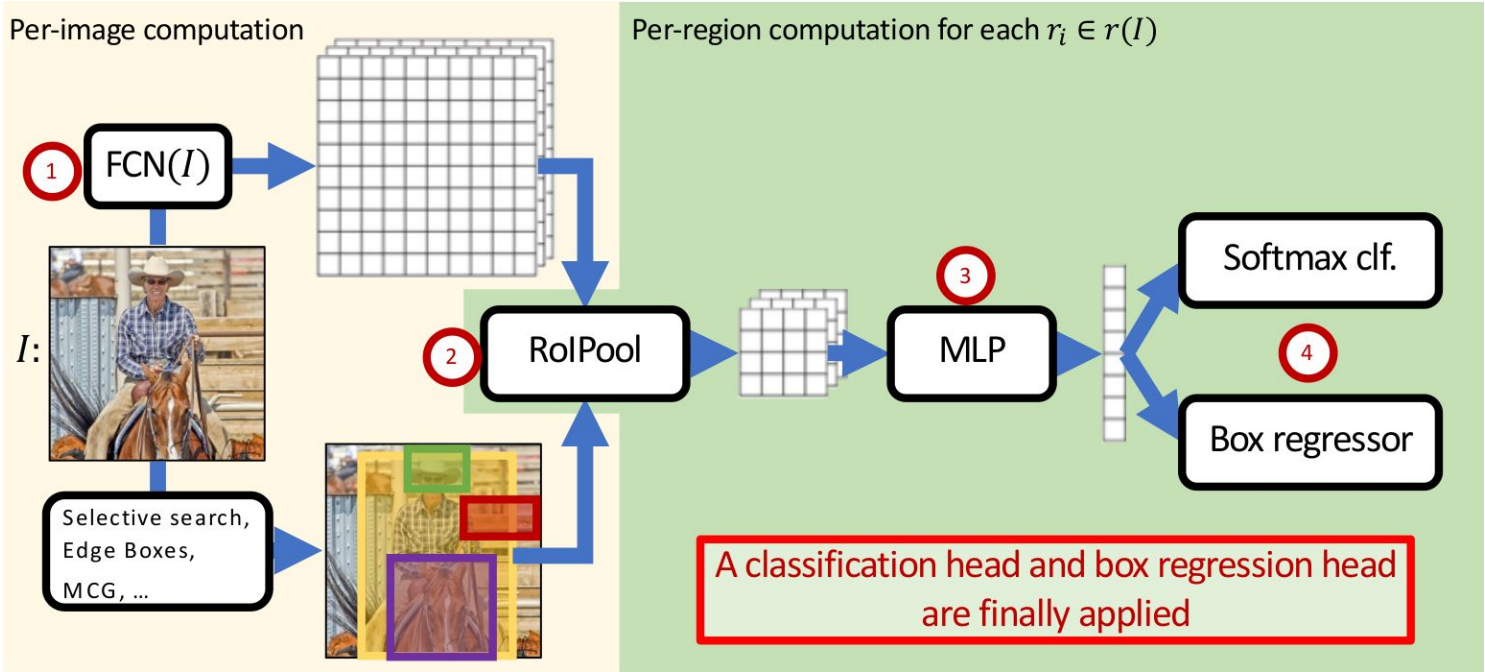
[1] Girshick. Fast R-CNN. ICCV, 2015.

Fast R-CNN



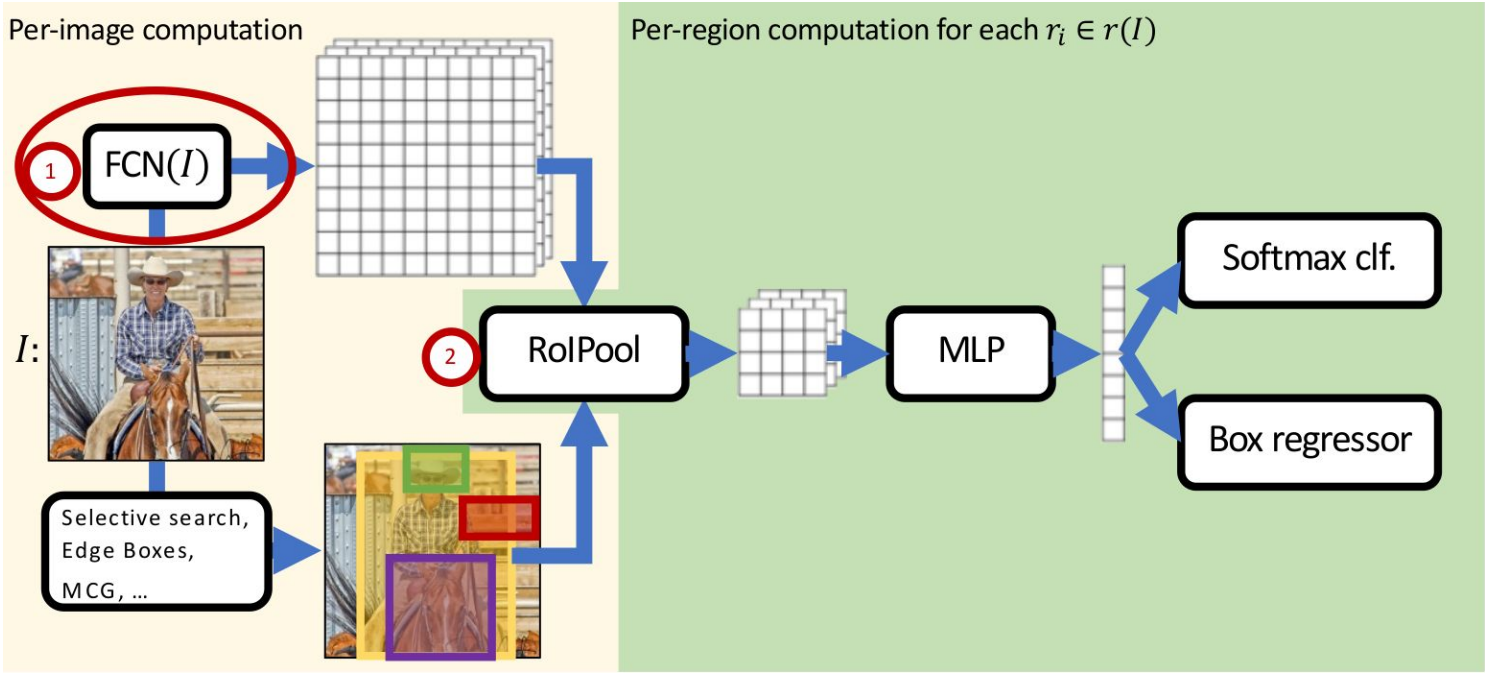
[1] Girshick. Fast R-CNN. ICCV, 2015.

Fast R-CNN



[1] Girshick. Fast R-CNN. ICCV, 2015.

Fast R-CNN



[1] Girshick. Fast R-CNN. ICCV, 2015.

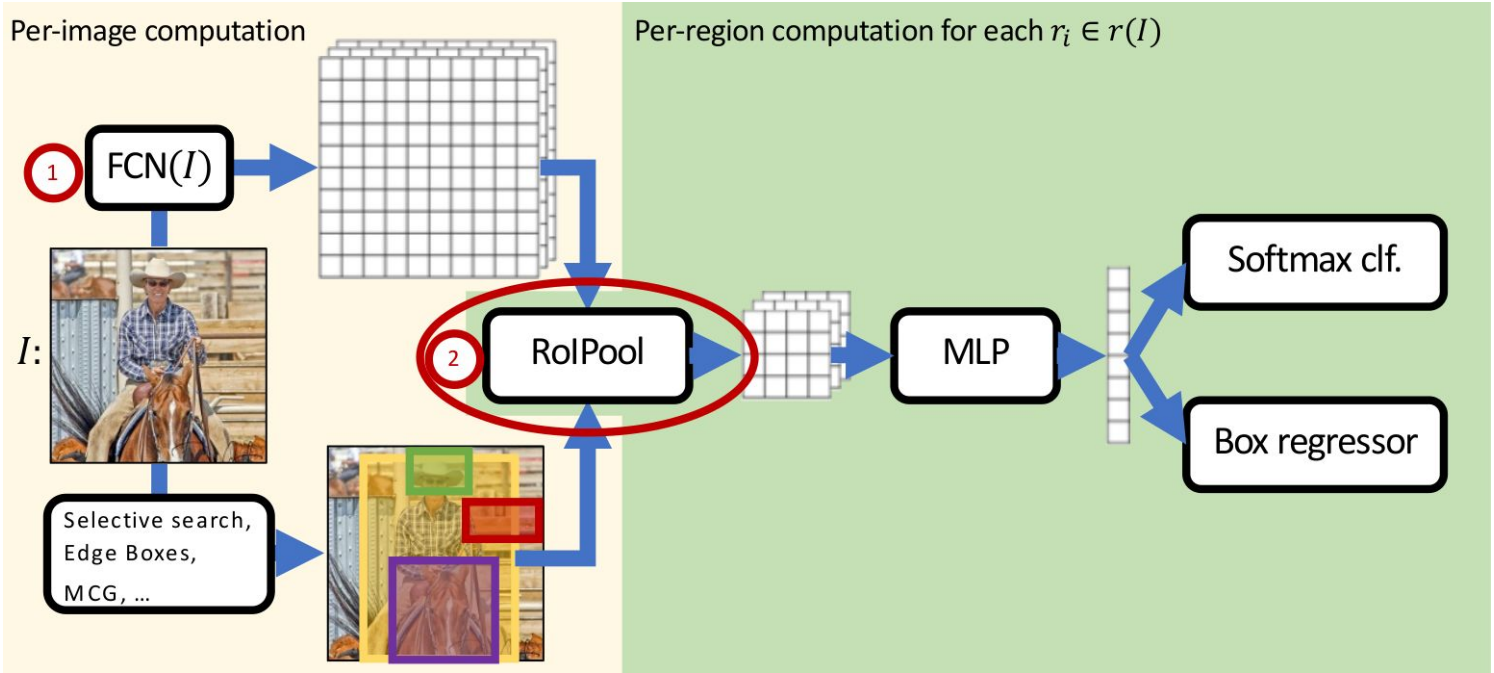
Fast R-CNN

Backbone:

- Use any standard convolutional network as “backbone” architecture, e.g. AlexNet, VGG, ResNet, Inception, ResNeXt, DenseNet, …
- Remove global pooling, so output spatial dimensions are proportional to input spatial dims
- A good network exploits the strongest recognition backbone (features matter!)

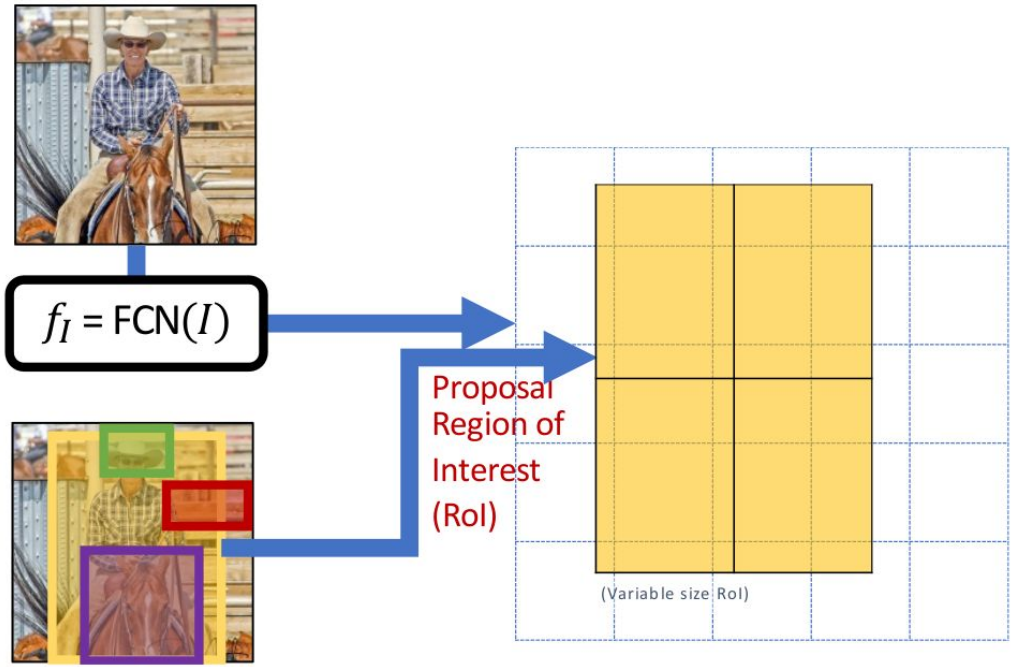
[1] Girshick. Fast R-CNN. ICCV, 2015.

Fast R-CNN



[1] Girshick. Fast R-CNN. ICCV, 2015.

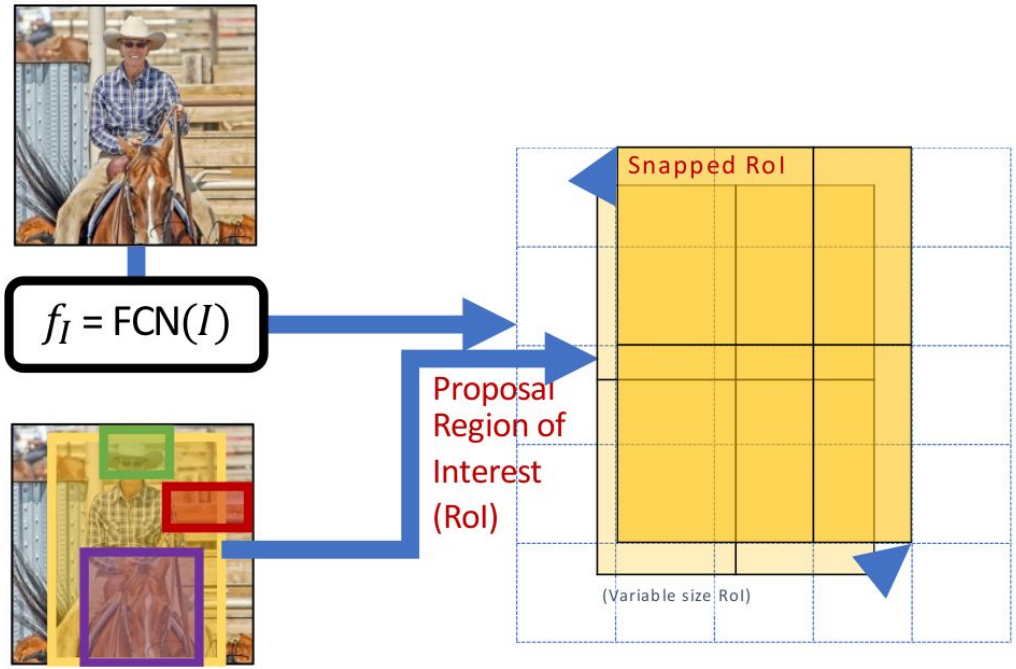
Fast R-CNN



Key innovation in SPP-net
[He et al. 2014]

[1] Girshick. Fast R-CNN. ICCV, 2015.

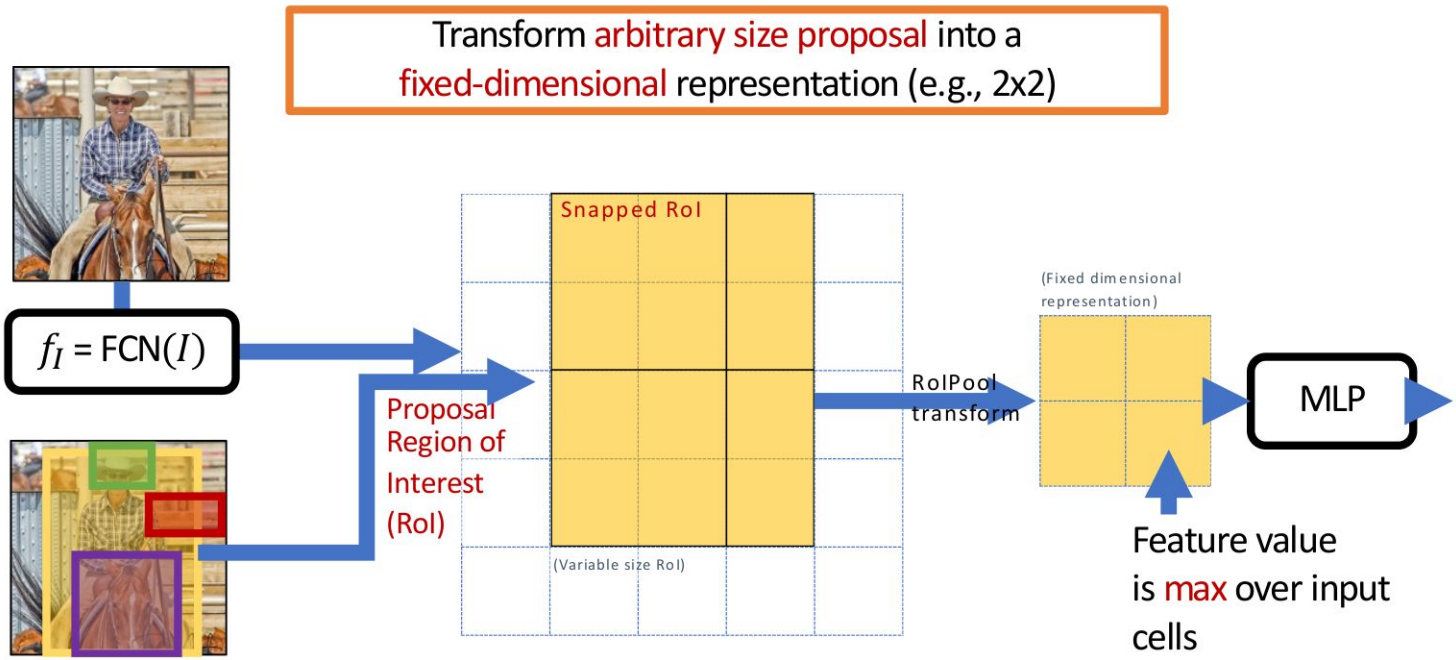
Fast R-CNN



Key innovation in SPP-net
[He et al. 2014]

[1] Girshick. Fast R-CNN. ICCV, 2015.

Fast R-CNN



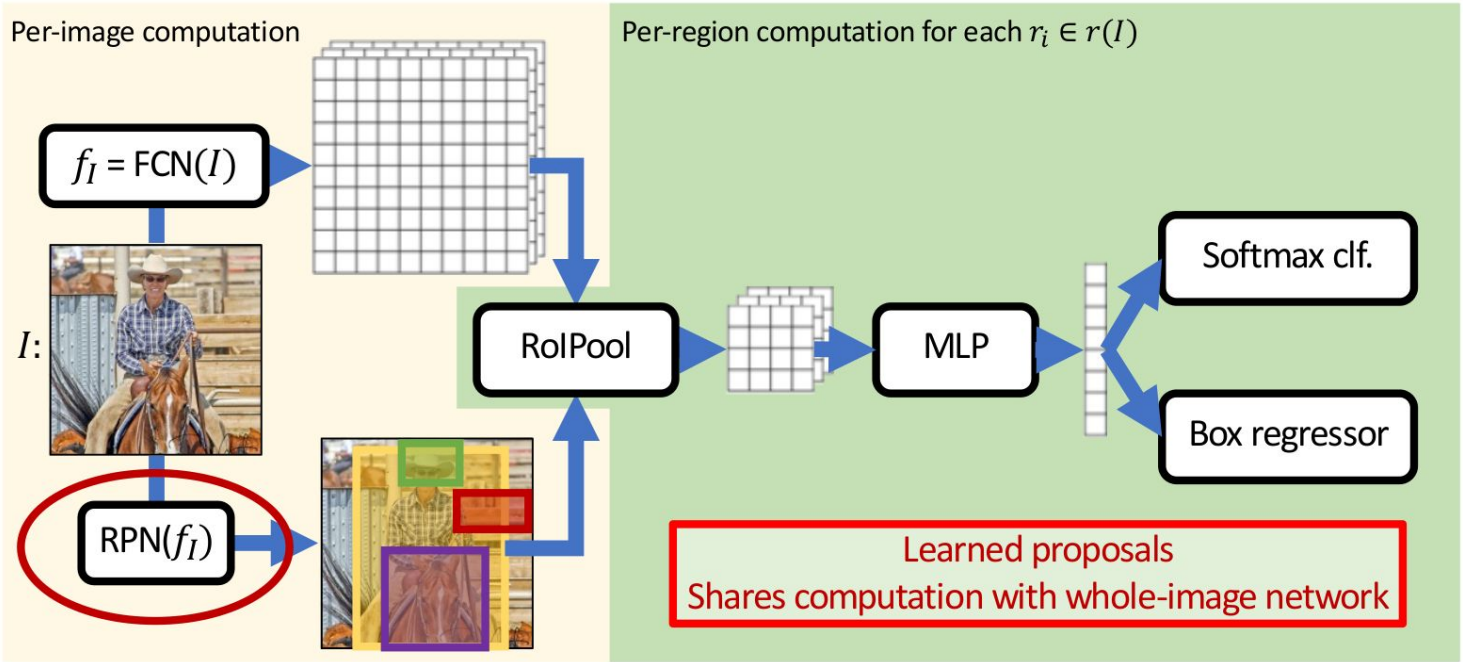
[1] Girshick. Fast R-CNN. ICCV, 2015.

Fast R-CNN problems

- ~~Heavy per-region computation (2000 full network evaluations)~~
- ~~No computation / feature sharing~~
- Slow region proposal method adds to runtime
- Generic proposal method has low recall

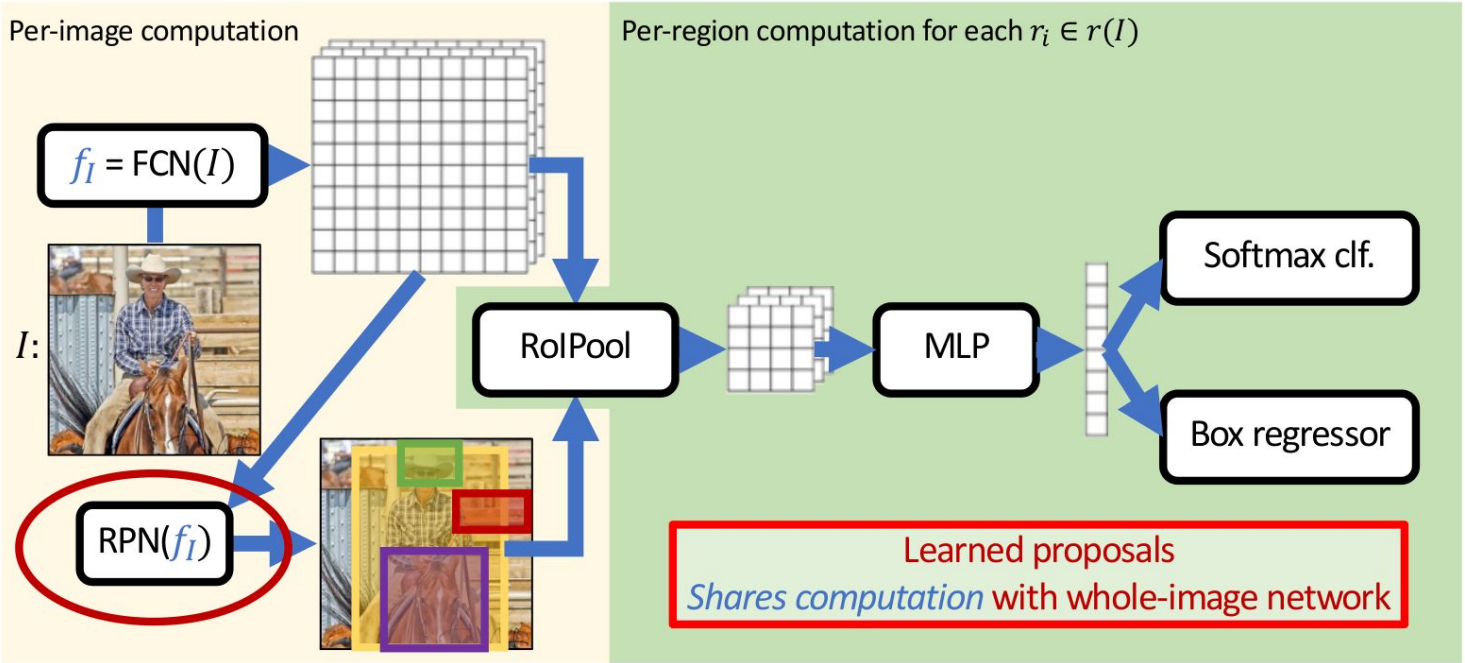
[1] Girshick, Donahue, Darrell and Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Faster R-CNN



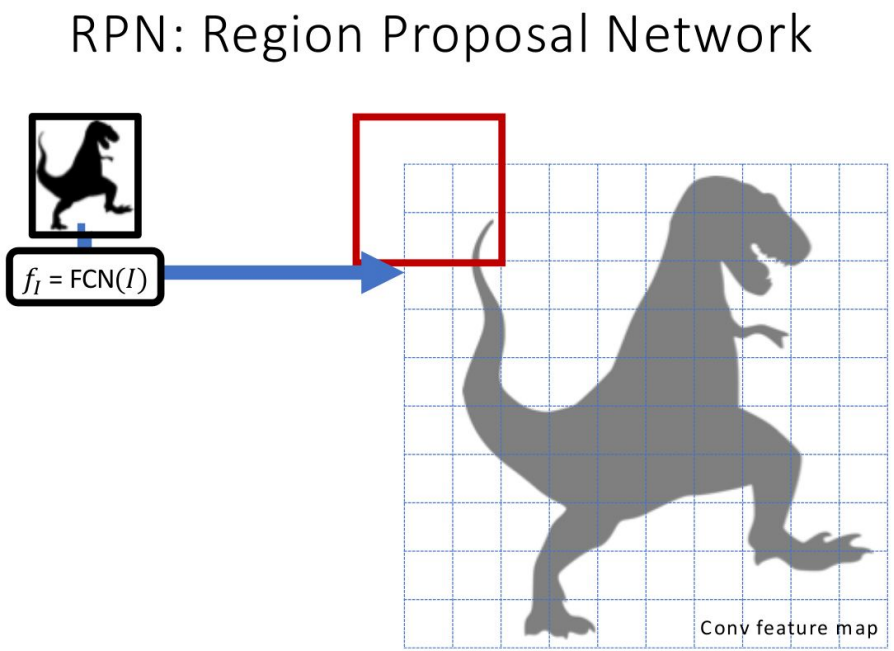
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN



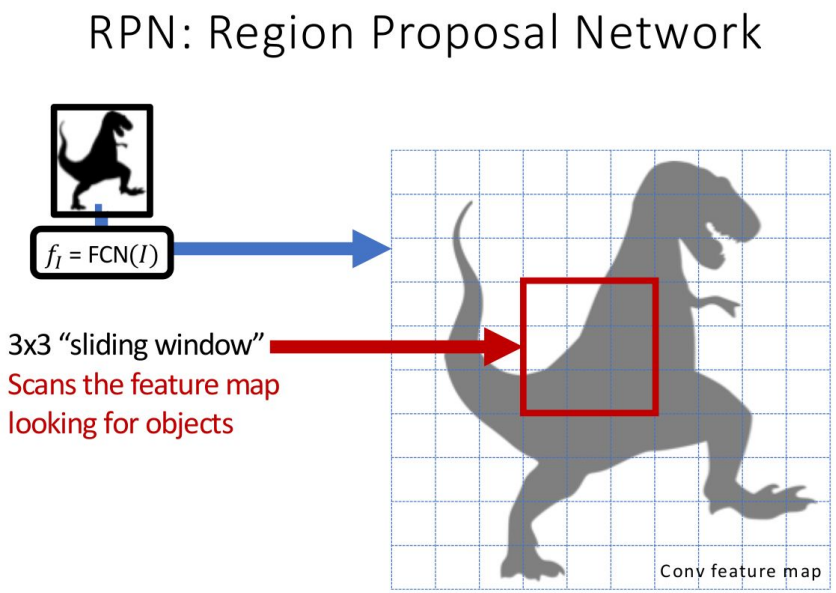
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



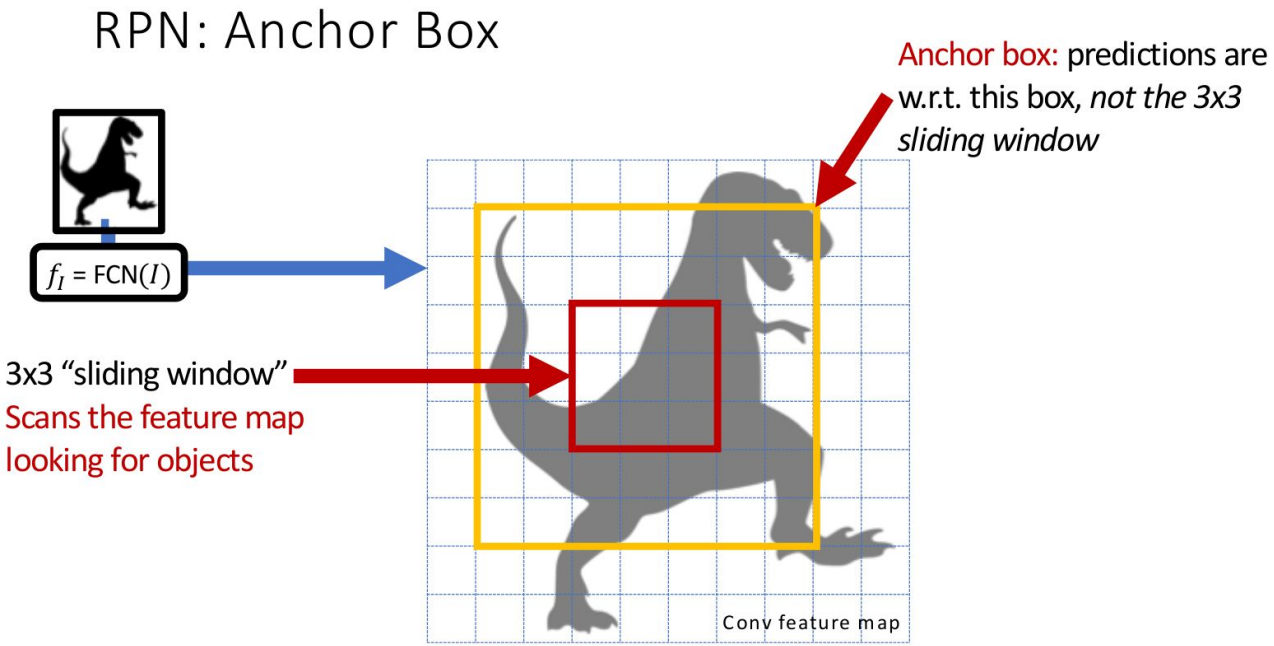
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



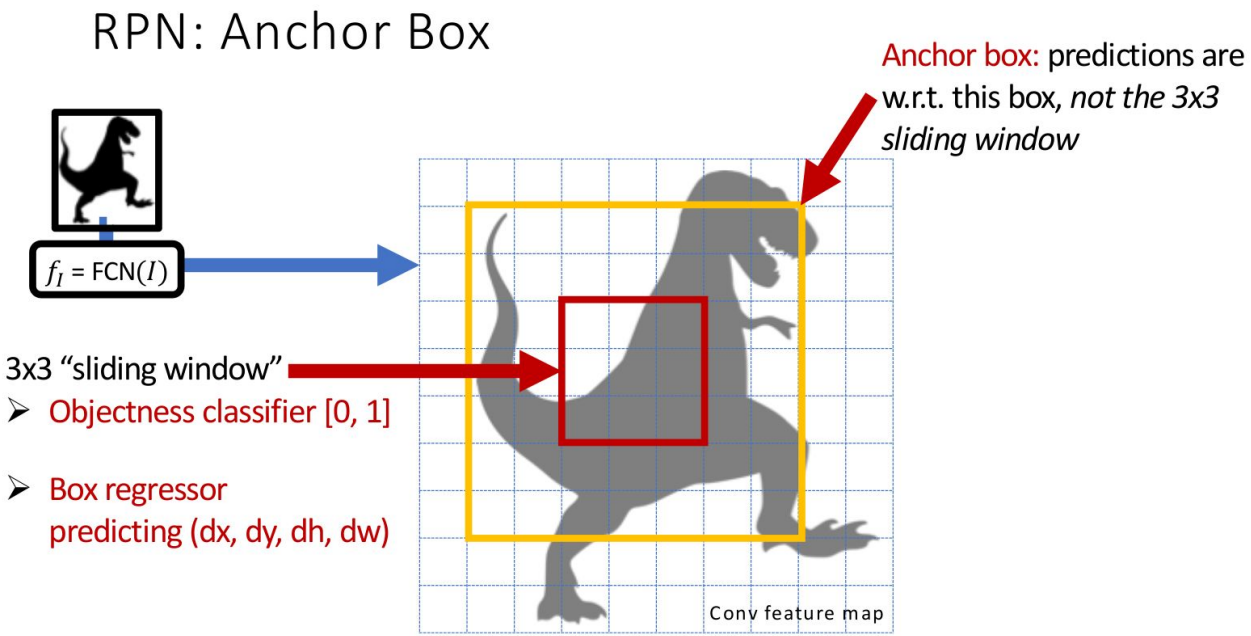
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



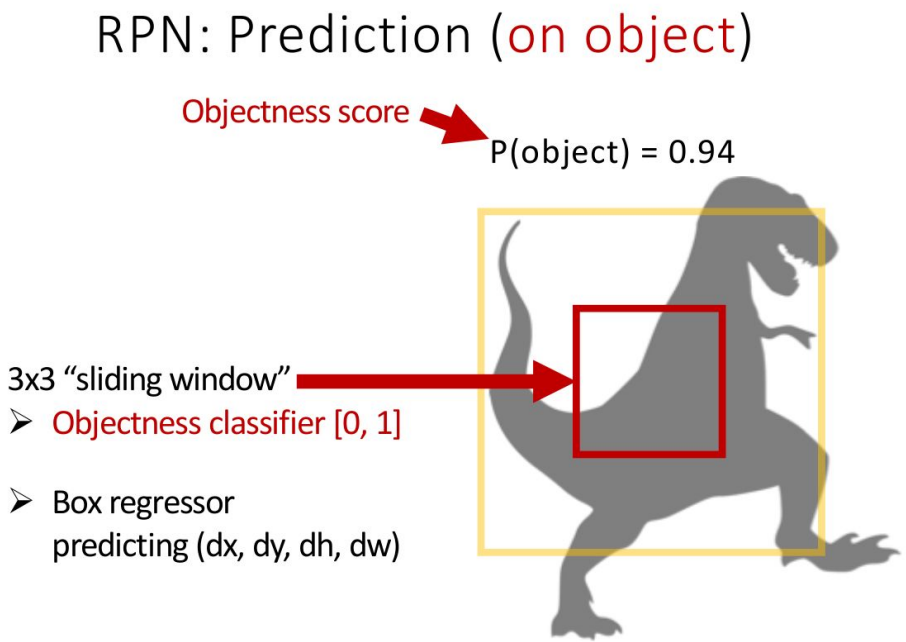
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



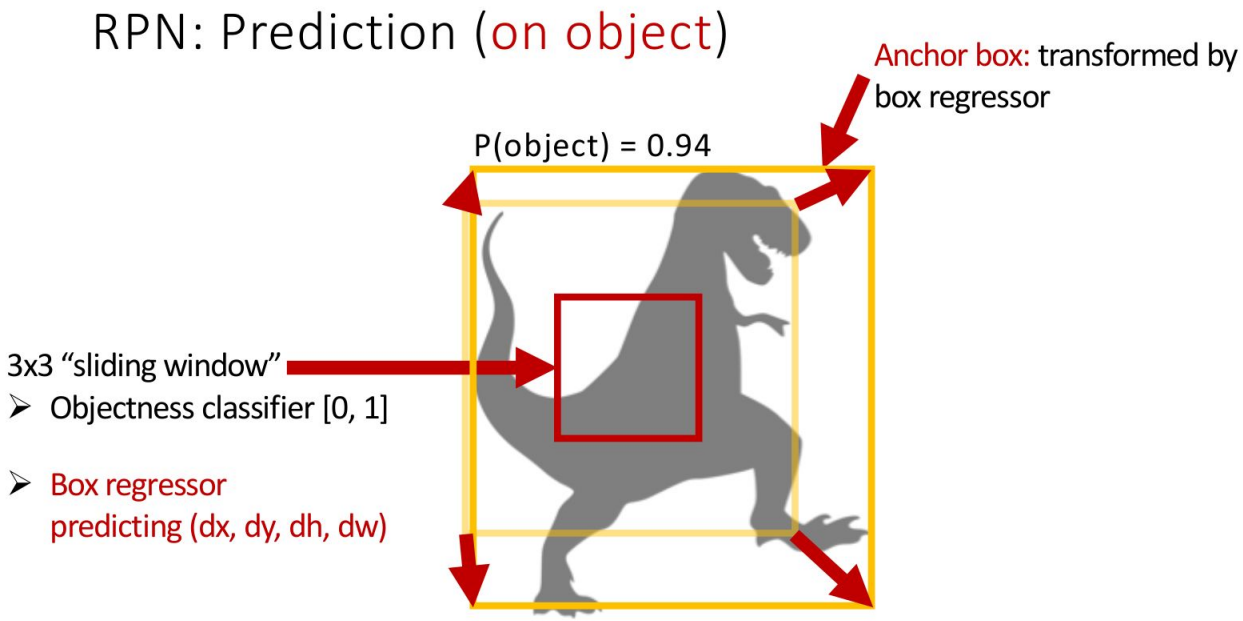
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



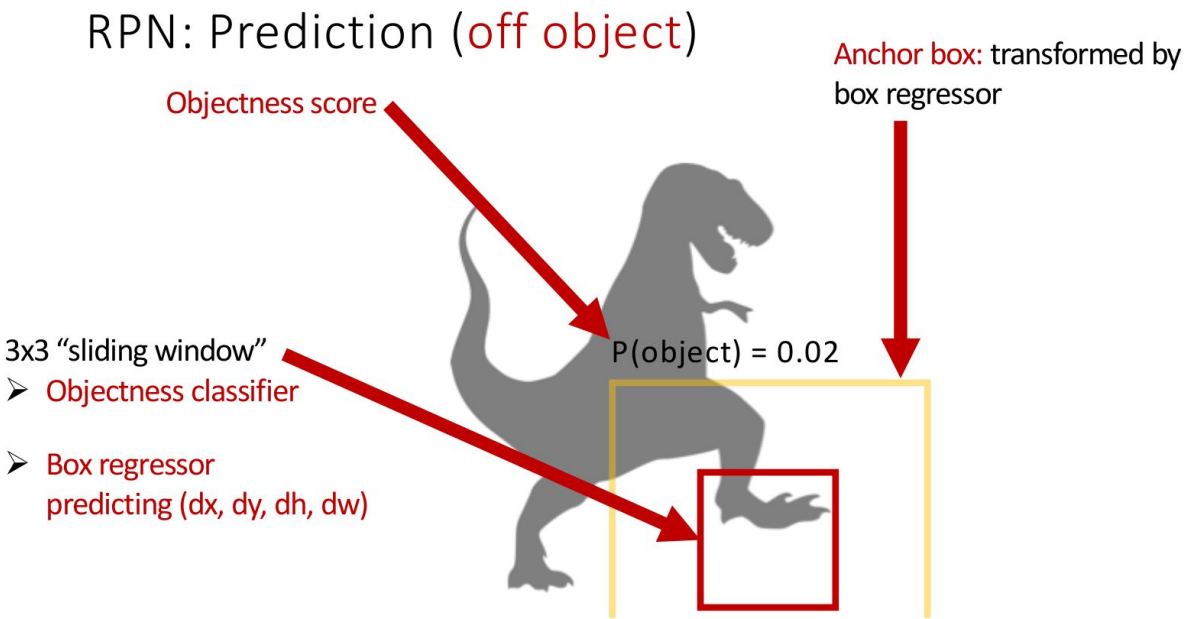
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



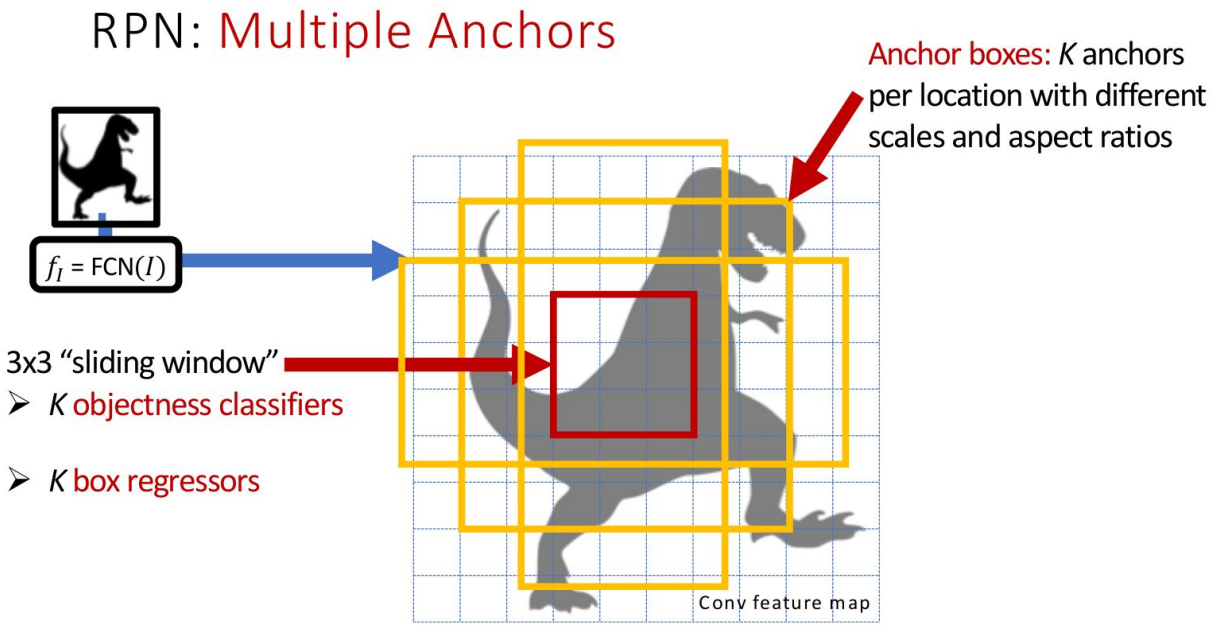
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



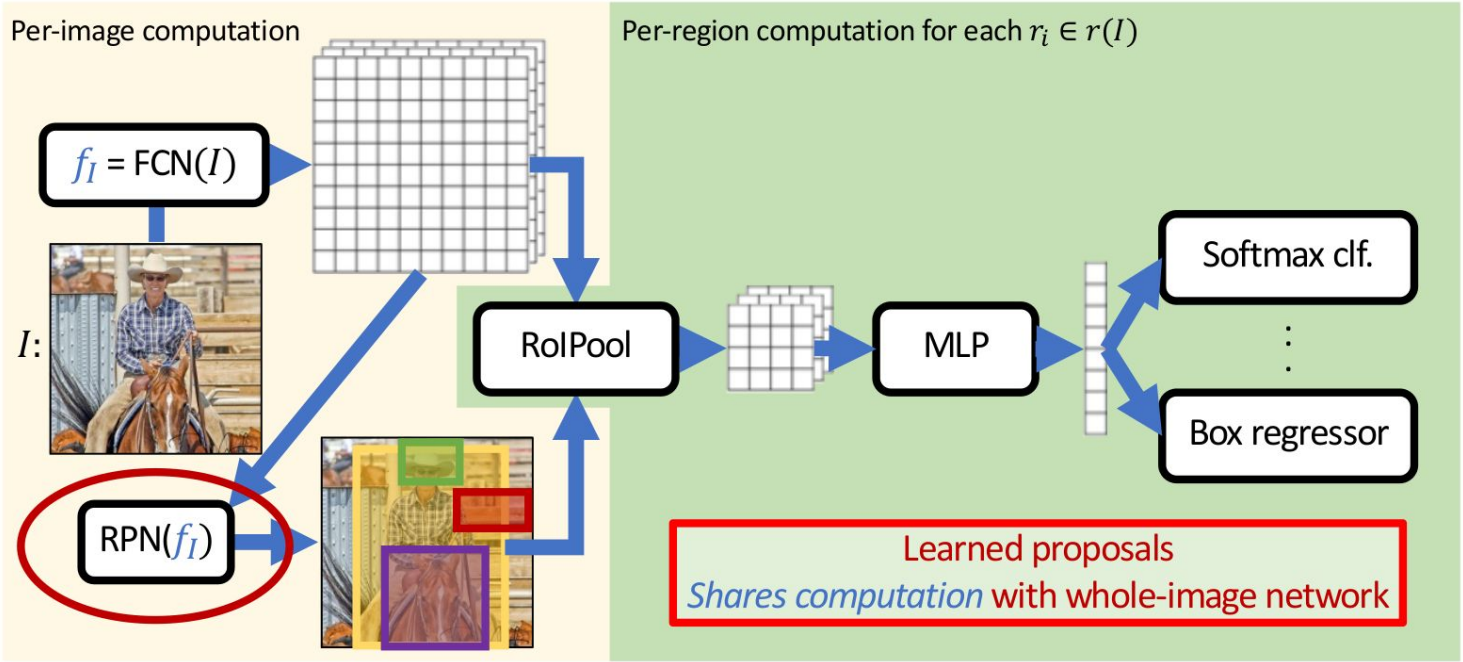
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



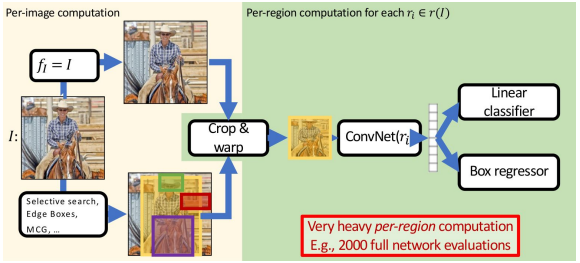
[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

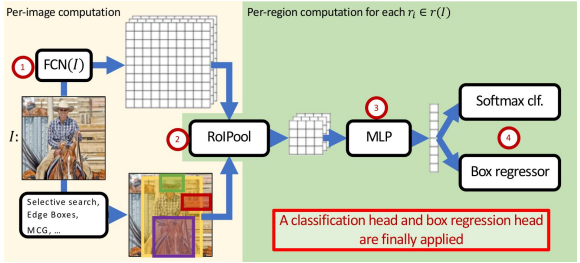
Faster R-CNN

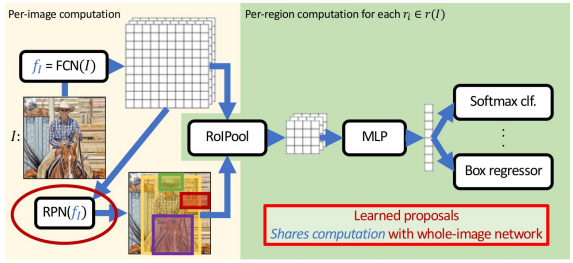


[1] Ren, He, Girshick and Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

Summary on two-stage object detectors







"Slow" R-CNN:

Run CNN independently for each region

Fast R-CNN:

Apply differentiable cropping to shared image features

Faster R-CNN:

Compute proposals with CNN

Overview

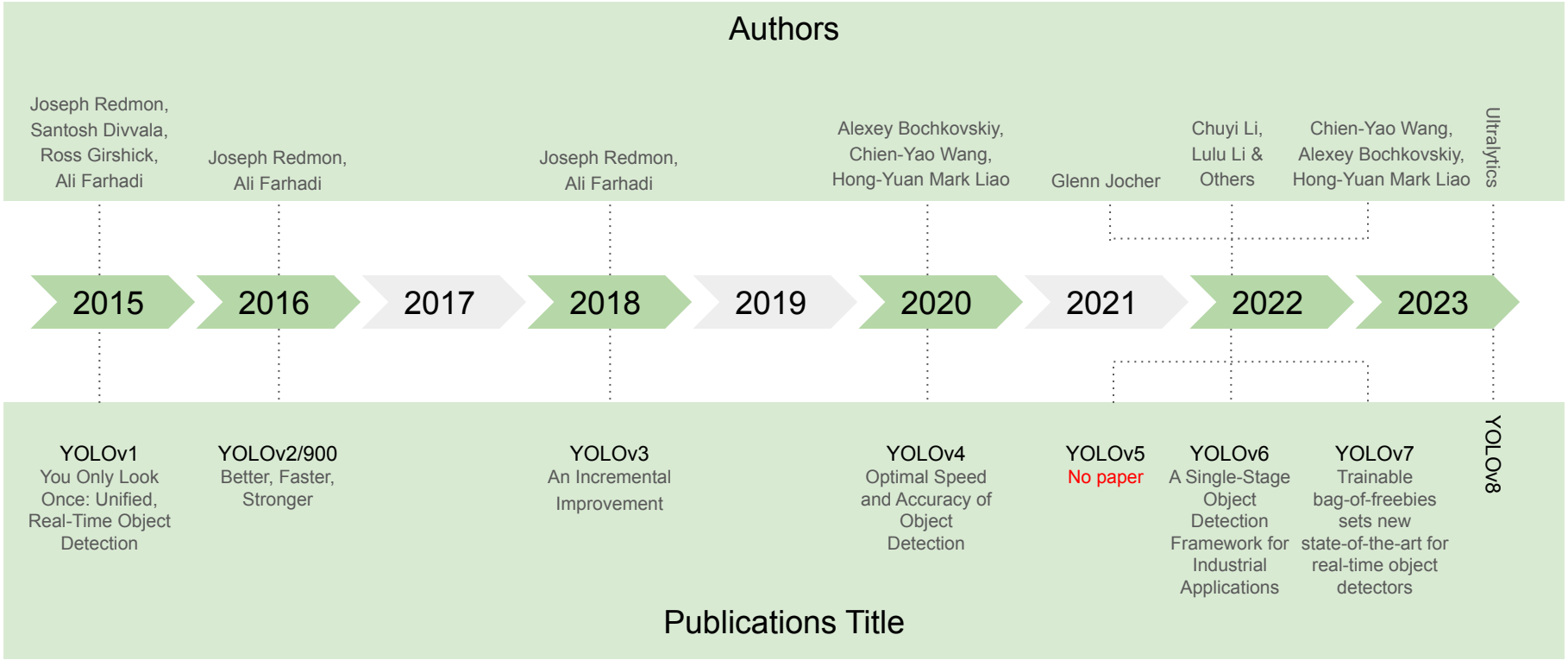
- Introduction
- Performance evaluation
- Two-stage (with proposal) object detectors
  - R-CNN
  - Fast R-CNN
  - Faster R-CNN
- **One-stage (proposal-free) object detectors**
  - YOLO
  - DETR

SSD

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg.

SSD: Single Shot MultiBox Detector. ECCV 2016

YOLO timeline from 2015 to 2022



YOLO v1

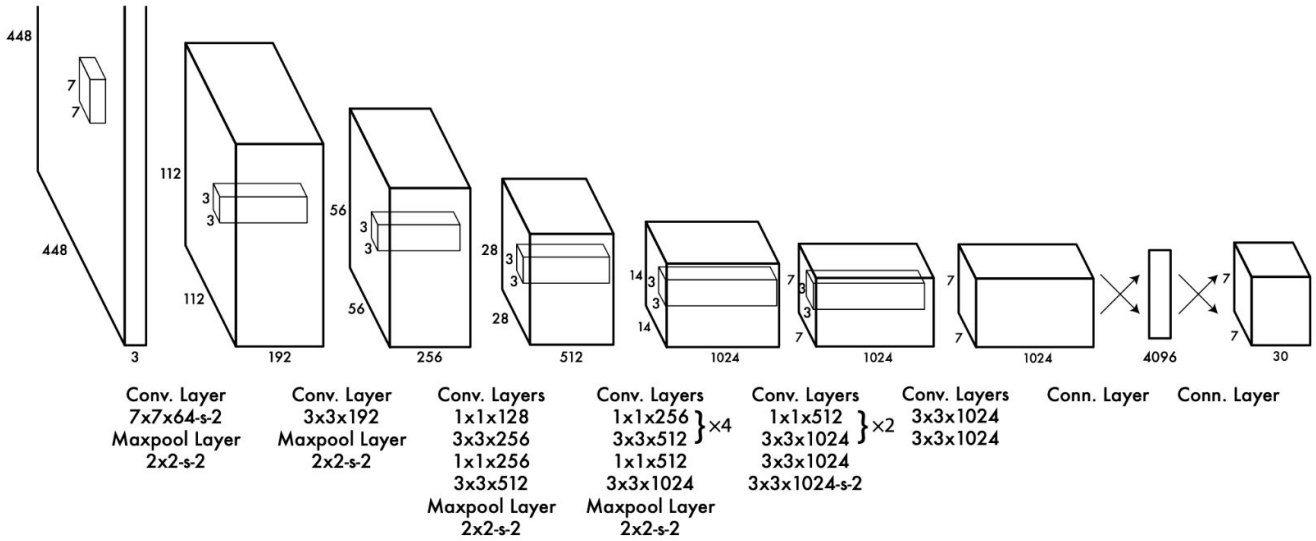


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.
You Only Look Once: Unified, Real-Time Object Detection. CVPR 2016

YOLO v1

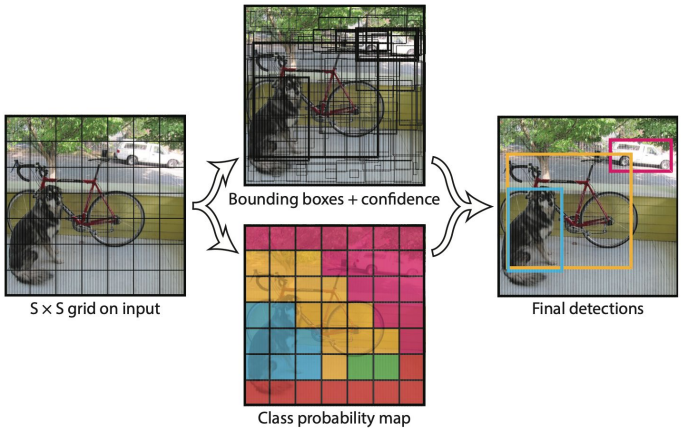


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

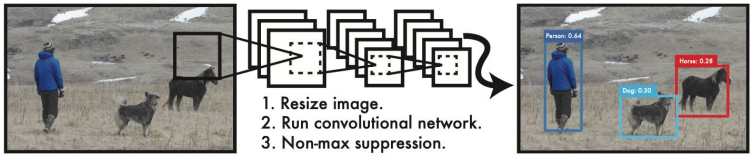


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.
You Only Look Once: Unified, Real-Time Object Detection. CVPR 2016

YOLO v2

	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**

Joseph Redmon, Ali Farhadi.
YOLO9000: Better, Faster, Stronger. CVPR 2017

YOLO v2

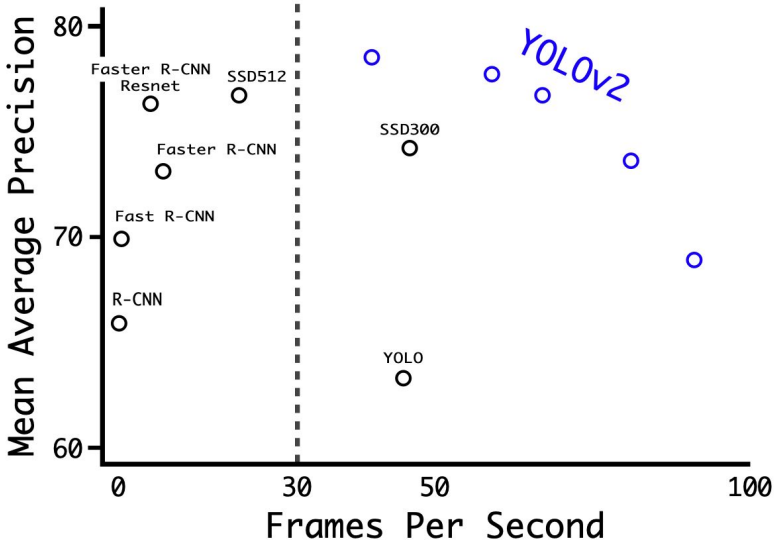


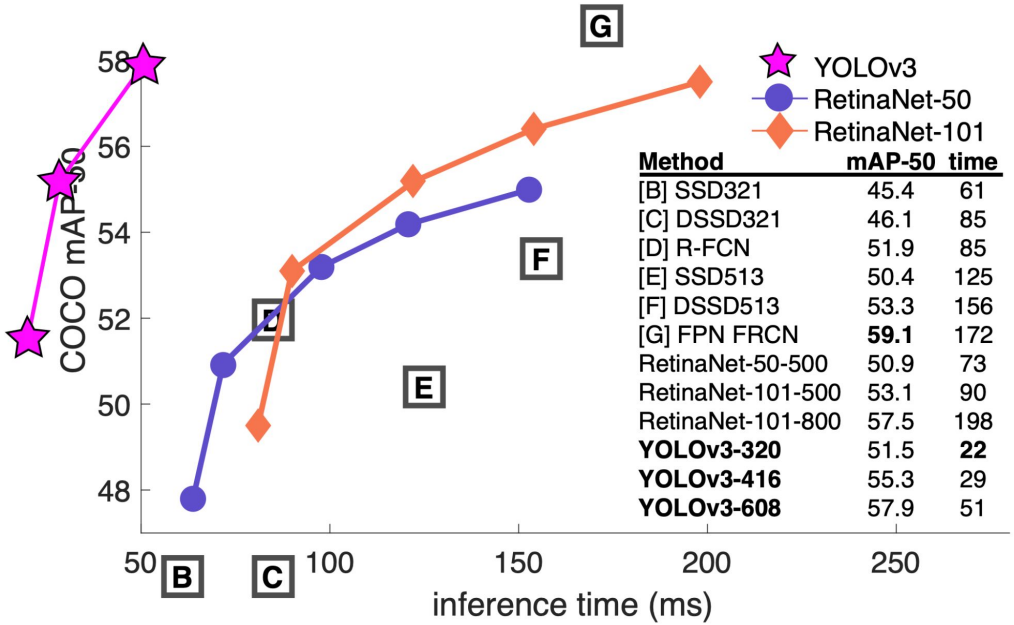
Figure 4: Accuracy and speed on VOC 2007.

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.

Joseph Redmon, Ali Farhadi.
YOLO9000: Better, Faster, Stronger. CVPR 2017

YOLO v3



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

	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.

Joseph Redmon, Ali Farhadi

YOLOv3: An Incremental Improvement

YOLO v4

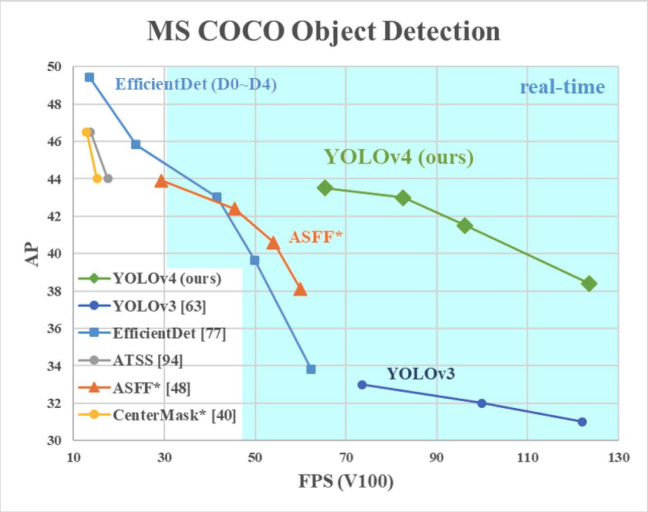
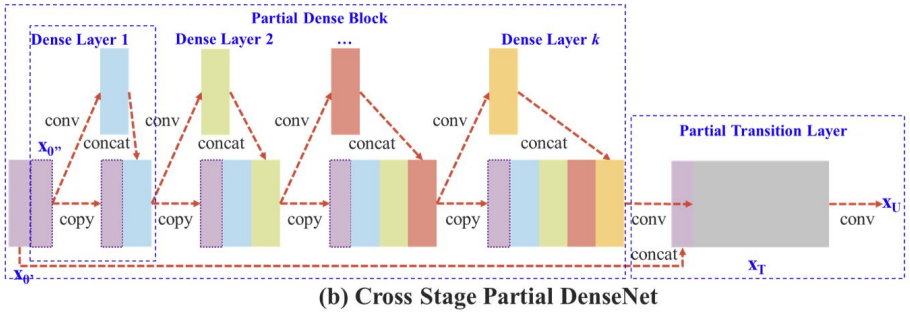


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.



(b) Cross Stage Partial DenseNet

Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao.
[YOLOv4: Optimal Speed and Accuracy of Object Detection]

YOLO v5

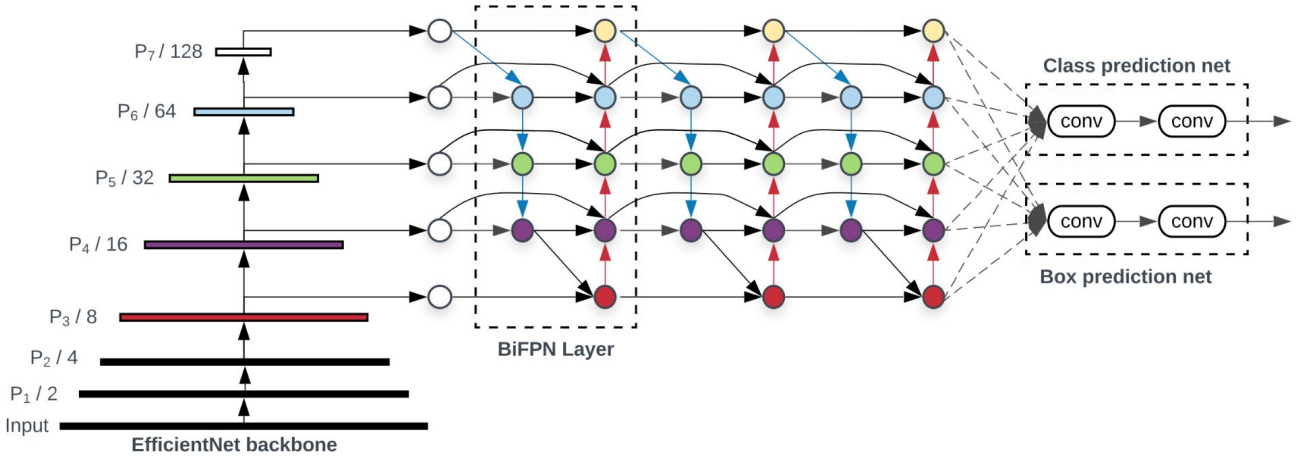


Figure 3: **EfficientDet architecture –** It employs EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

https://github.com/ultralytics/yolov5

YOLO v6

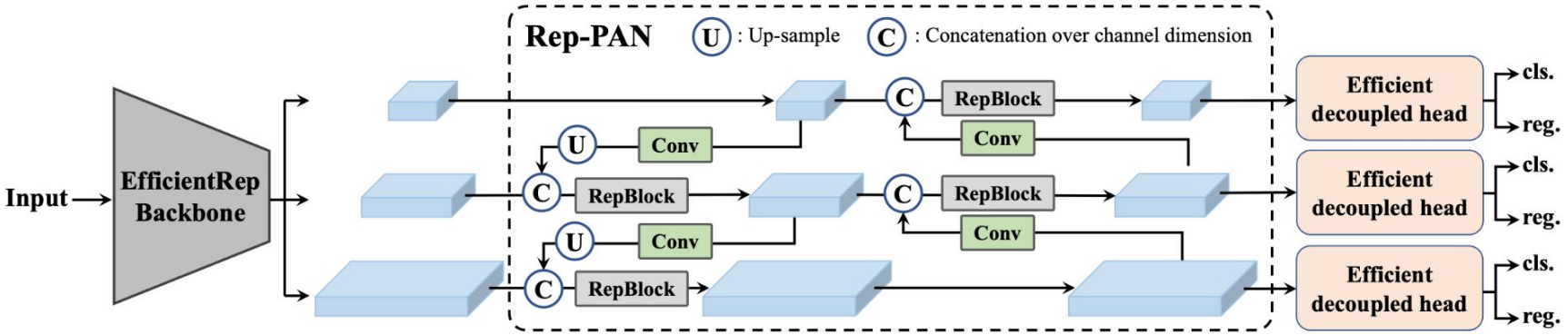


Figure 2: The YOLOv6 framework (N and S are shown). Note for M/L, RepBlocks is replaced with CSPStackRep.

Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, Xiaolin Wei.
YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications

YOLO v6

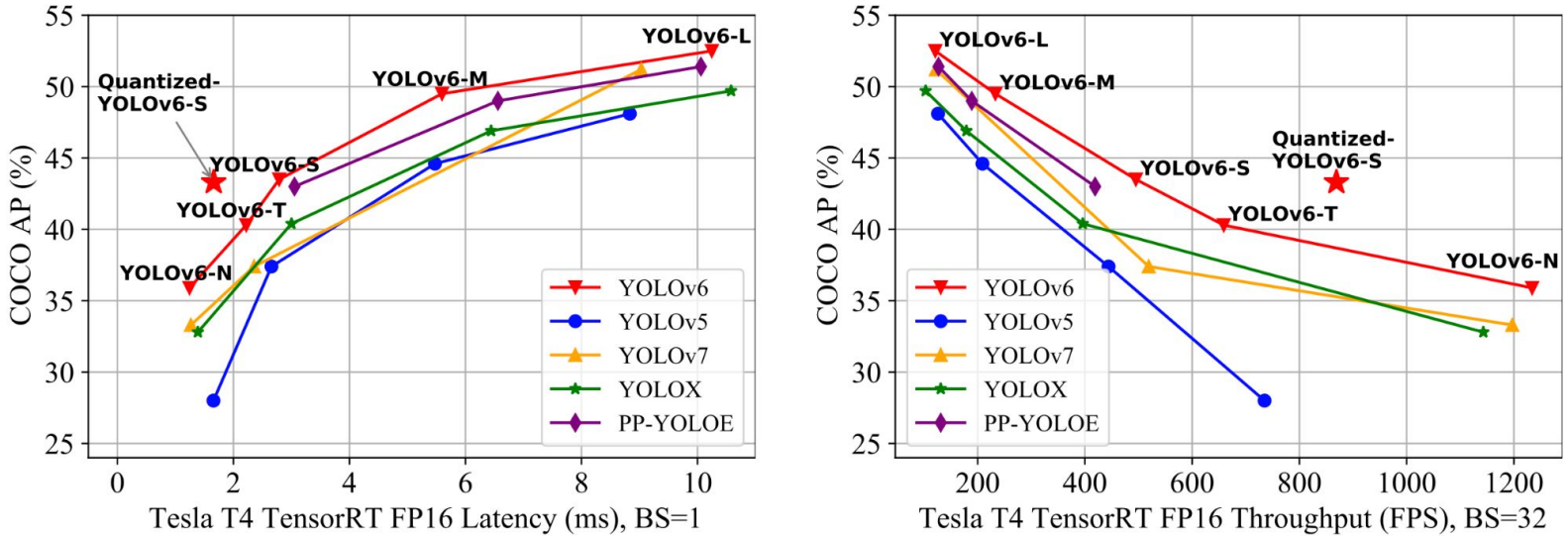


Figure 1: Comparison of state-of-the-art efficient object detectors. Both latency and throughput (at a batch size of 32) are given for a handy reference. All models are test with TensorRT 7 except that the quantized model is with TensorRT 8.

Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, Xiaolin Wei.
YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications

YOLO v7

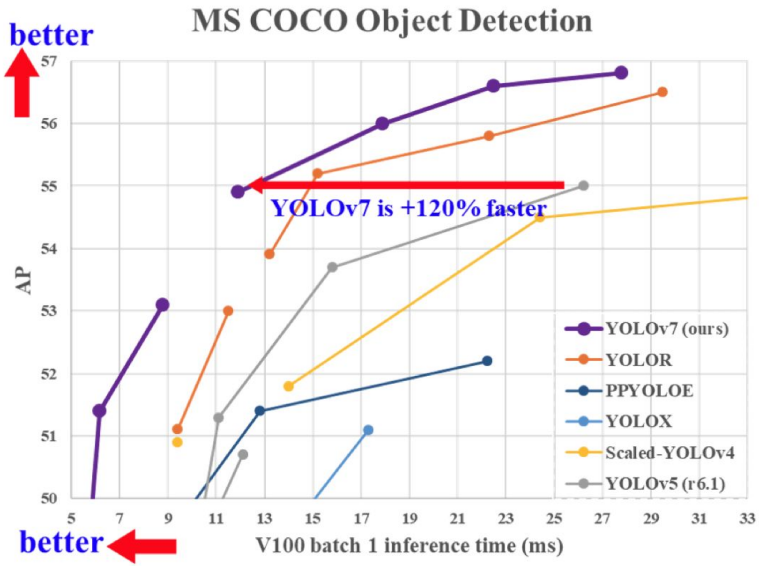


Figure 1: Comparison with other real-time object detectors, our proposed methods achieve state-of-the-arts performance.

- A key improvement in YOLO v7 is the use of a loss function called “**focal loss**”
- **Focal loss** down-weight the loss for well-classified examples and focus on the hard examples
- YOLO v7 also has a **higher resolution** than the previous versions, it processes images at a resolution of 608x608 pixels, which is higher than the 416x416 resolution used in YOLO v3
- YOLO v7 can process images at a rate of 155 frames per second (the original baseline YOLO model was capable of processing at a maximum rate of 45 frames per second)

Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao.
YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

YOLO v7

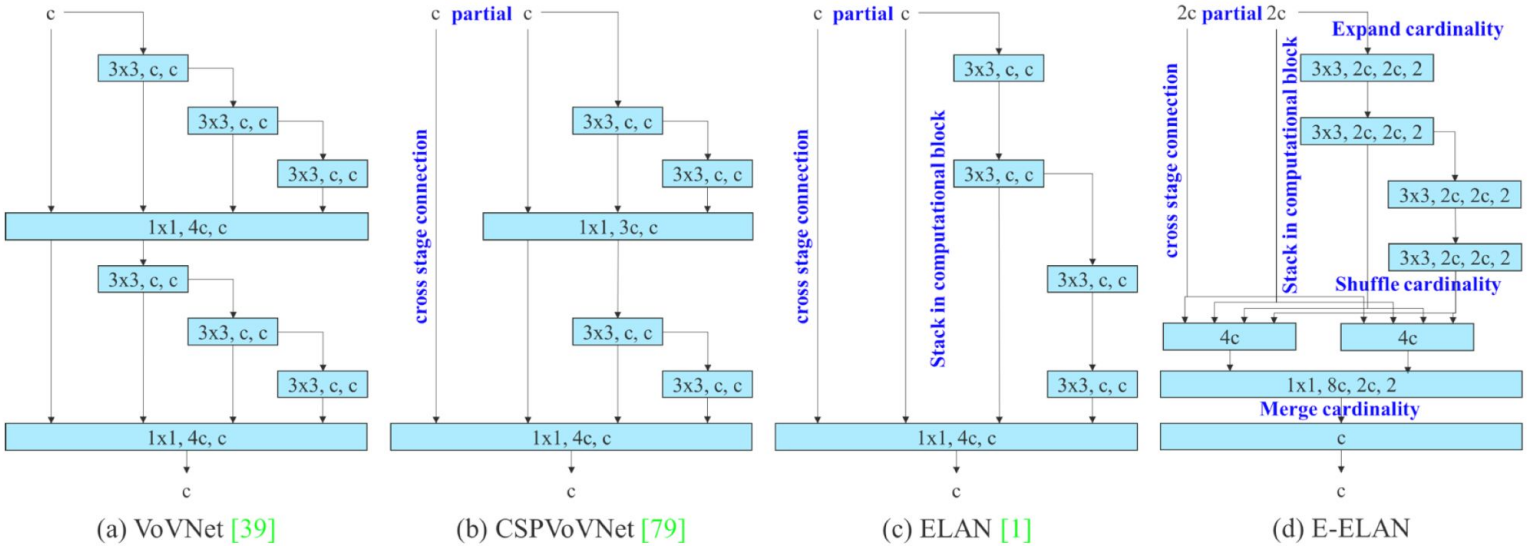
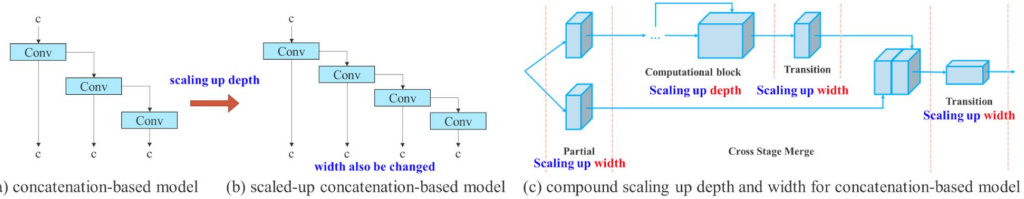


Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmission path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in a shuffle and merge cardinality manner. This way of operation can enhance the features learned by different feature maps and improve the use of parameters and calculations.

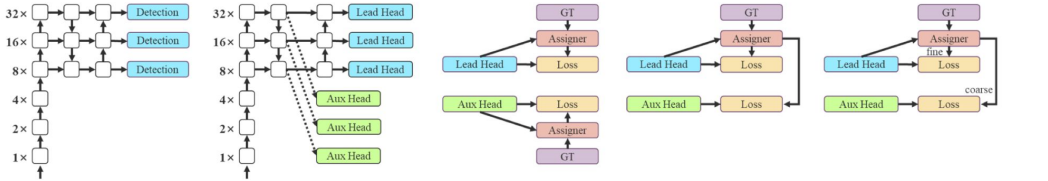
Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao.
YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

YOLO v7



(a) concatenation-based model (b) scaled-up concatenation-based model (c) compound scaling up depth and width for concatenation-based model

Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on concatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width of the subsequent transmission layer to increase. Therefore, we propose (c), that is, when performing model scaling on concatenation-based models, only the depth in a computational block needs to be scaled, and the remaining of transmission layer is performed with corresponding width scaling.



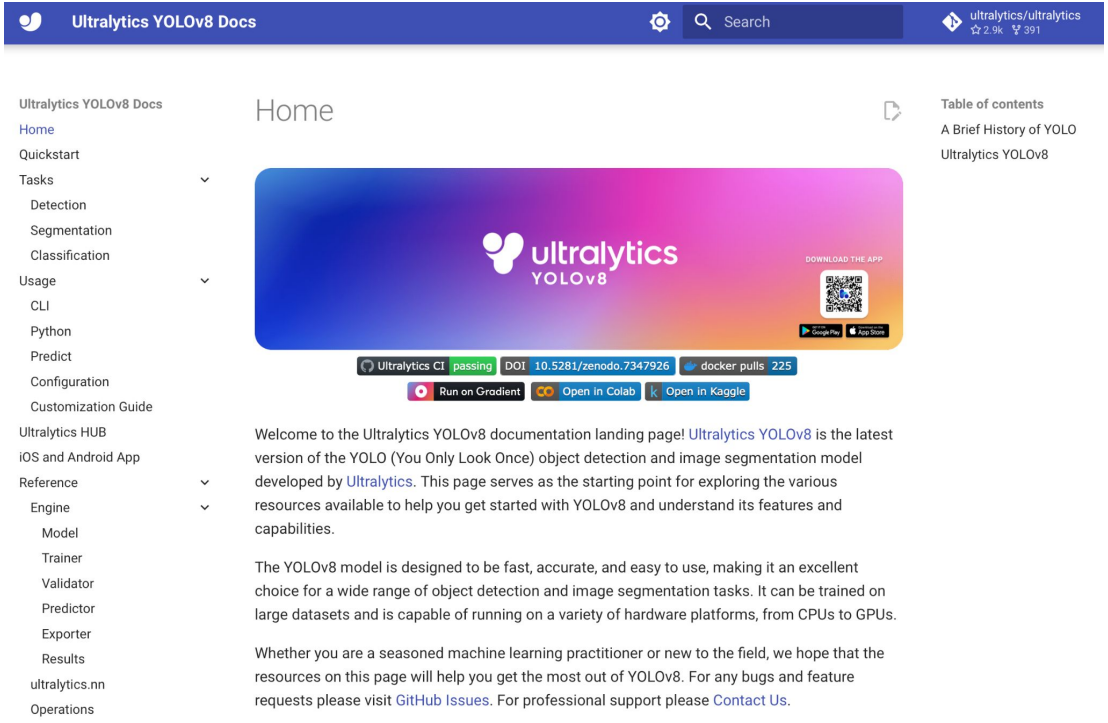
(a) Normal model (b) Model with auxiliary head (c) Independent assigner (d) Lead guided assigner (e) Coarse-to-fine lead guided assigner

Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be elaborated in Apendix.

Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao.

YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

YOLO v8



https://docs.ultralytics.com

DETR

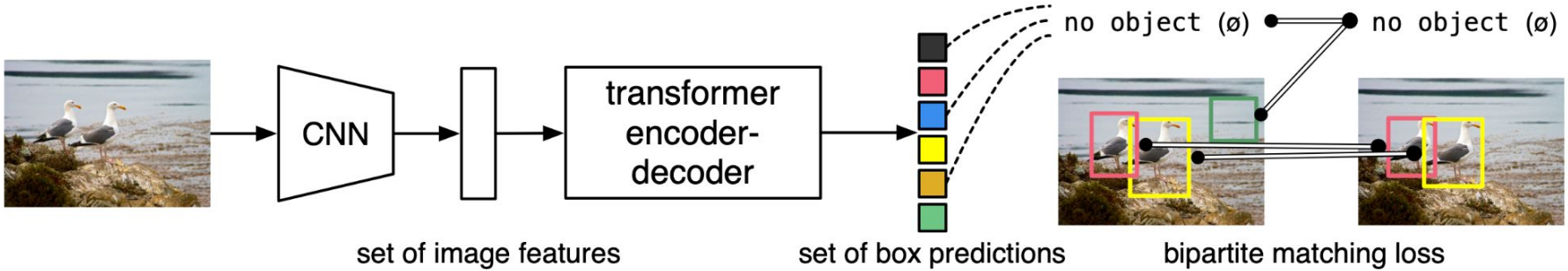


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, Sergey Zagoruyko. End-to-End Object Detection with Transformers. ECCV 2020

DETR

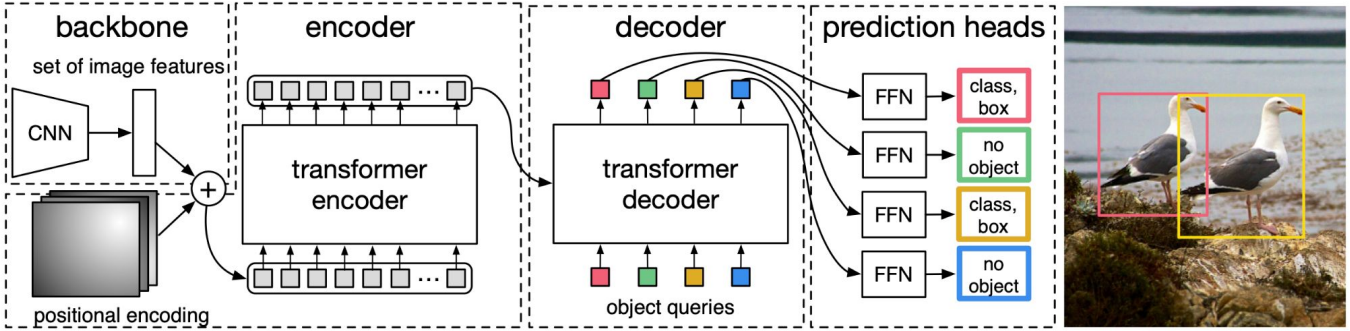


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, Sergey Zagoruyko. End-to-End Object Detection with Transformers. ECCV 2020

DETR

Table 1: Comparison with Faster R-CNN with a ResNet-50 and ResNet-101 backbones on the COCO validation set. The top section shows results for Faster R-CNN models in Detectron2 [50], the middle section shows results for Faster R-CNN models with GIoU [38], random crops train-time augmentation, and the long `9x` training schedule. DETR models achieve comparable results to heavily tuned Faster R-CNN baselines, having lower AP_S but greatly improved AP_L. We use torchscript Faster R-CNN and DETR models to measure FLOPS and FPS. Results without R101 in the name correspond to ResNet-50.

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, Sergey Zagoruyko. End-to-End Object Detection with Transformers. ECCV 2020

Q&A