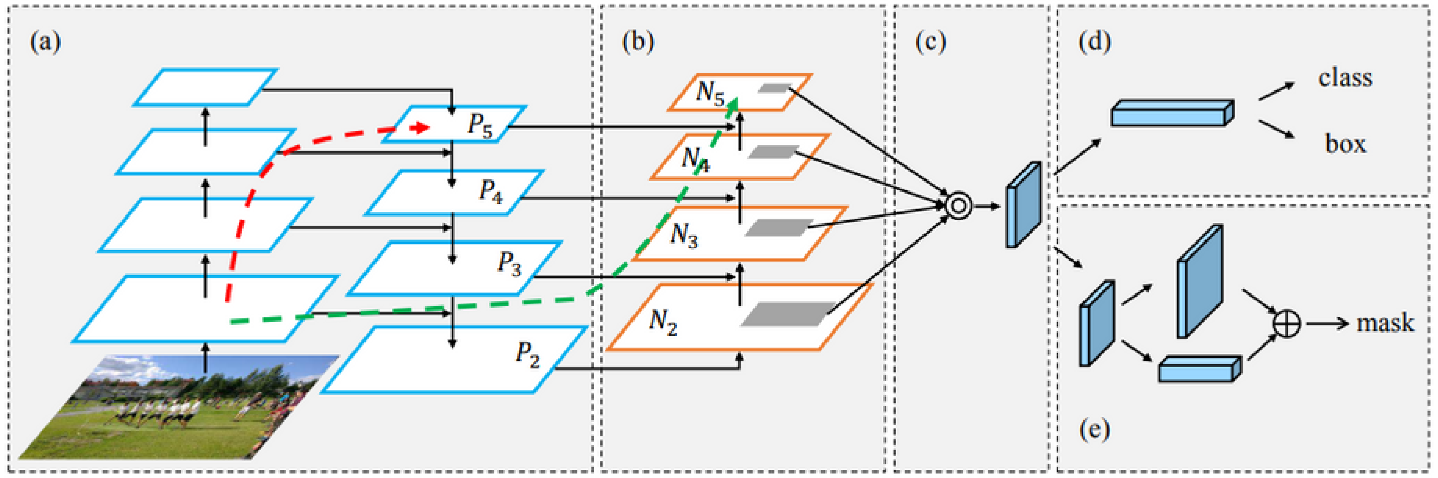
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Universidad de los Andes > FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

Detección de Objetos [YOLO, FPN, Focal Loss, PAN]



CLASE 7

José M. Saavedra R.
Profesor Asistente

jmsaavedrar@miuandes.cl

Ed. Ingeniería - Oficina 315

YOLO [You Only Look Once]

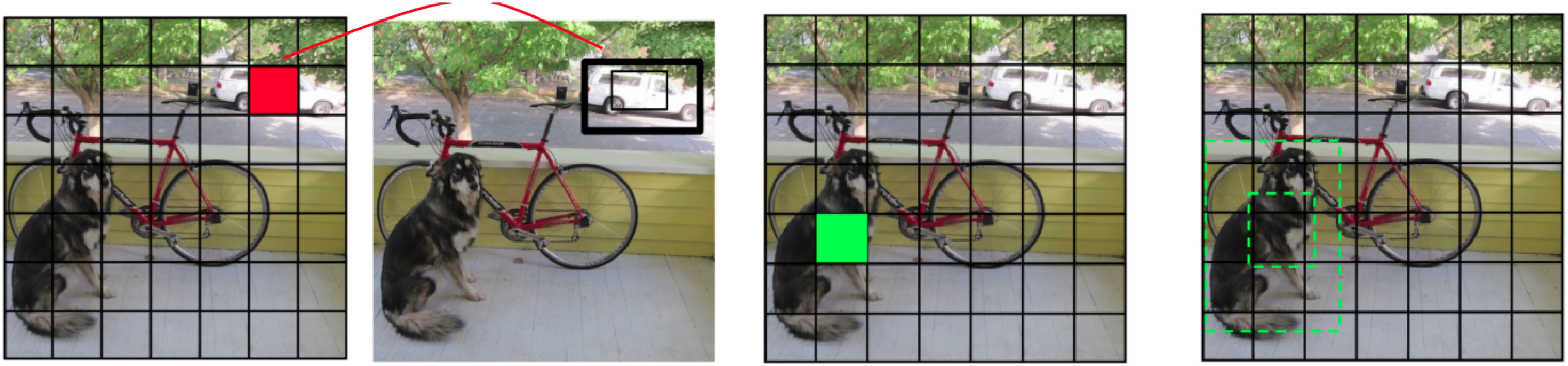
Unlike Faster R-CNN, YOLO is a one-stage model performing region proposal and **classification in a single step.**

https://arxiv.org/pdf/1506.02640.pdf

YOLO [You Only Look Once]

Predicted Boxes

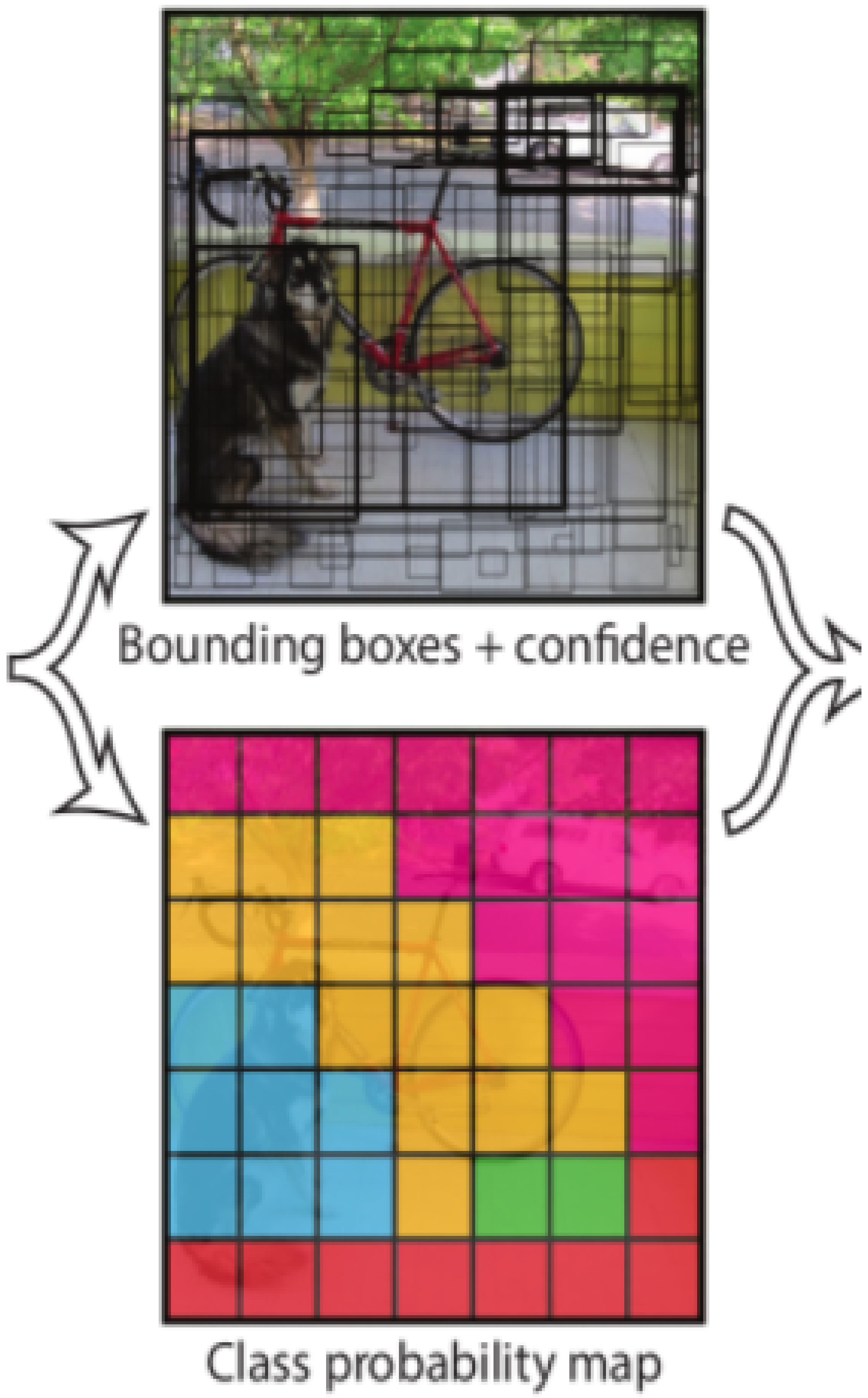
Each grid cell predicts B bounding boxes and confidence scores for those boxes under one-stage fashion.



YOLO [You Only Look Once]

Predicted Classes

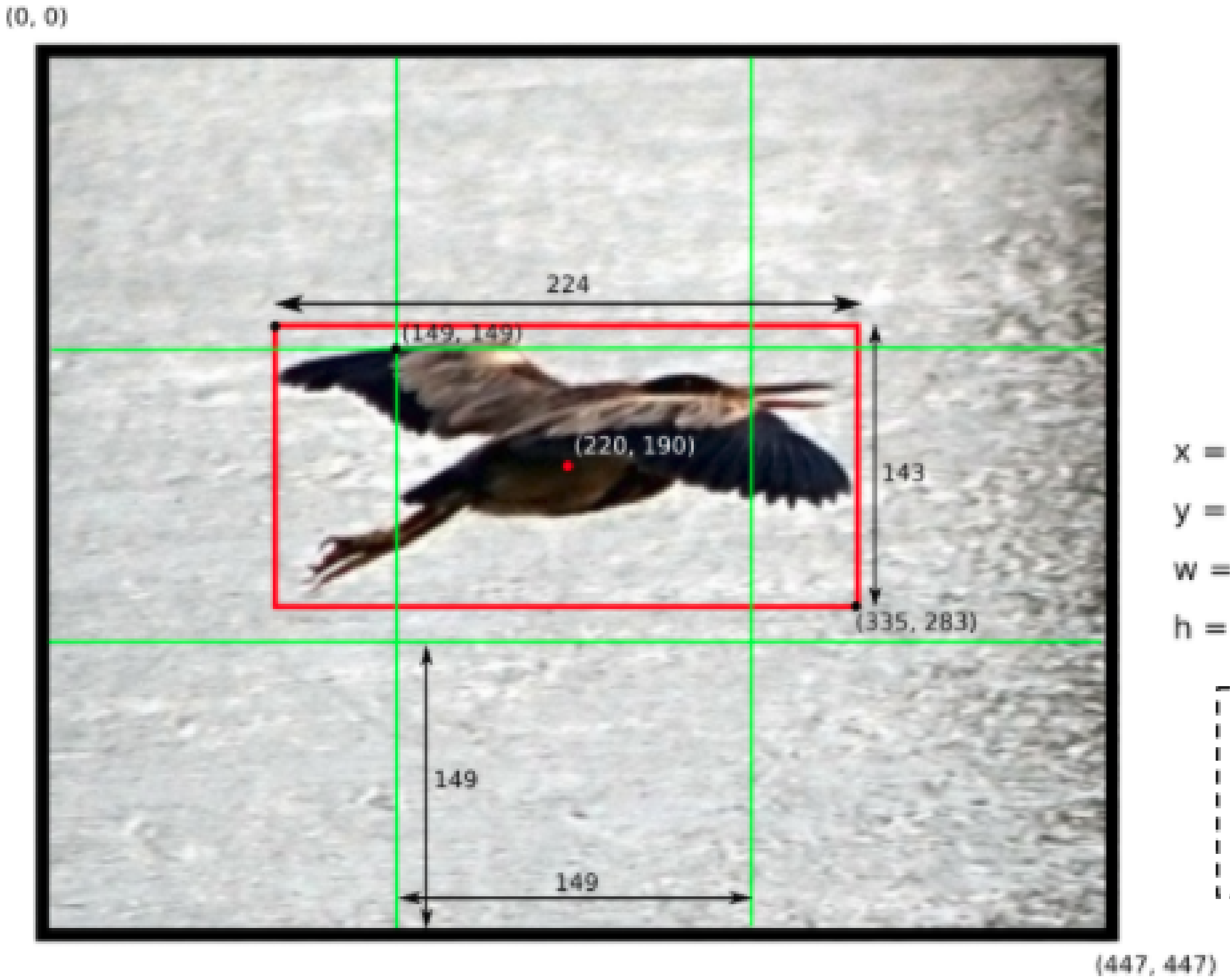
Each grid cell also predicts C conditional class probabilities, Pr(Class |Object). These probabilities are conditioned on the grid cell containing an object



YOLO [You Only Look Once]

- Coordinates of a box are relatives to the whole image, varying in [0,1].

- If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.



relative coordinates

x = (220-149) / 149 = 0.48

y = (190-149) / 149 = 0.28

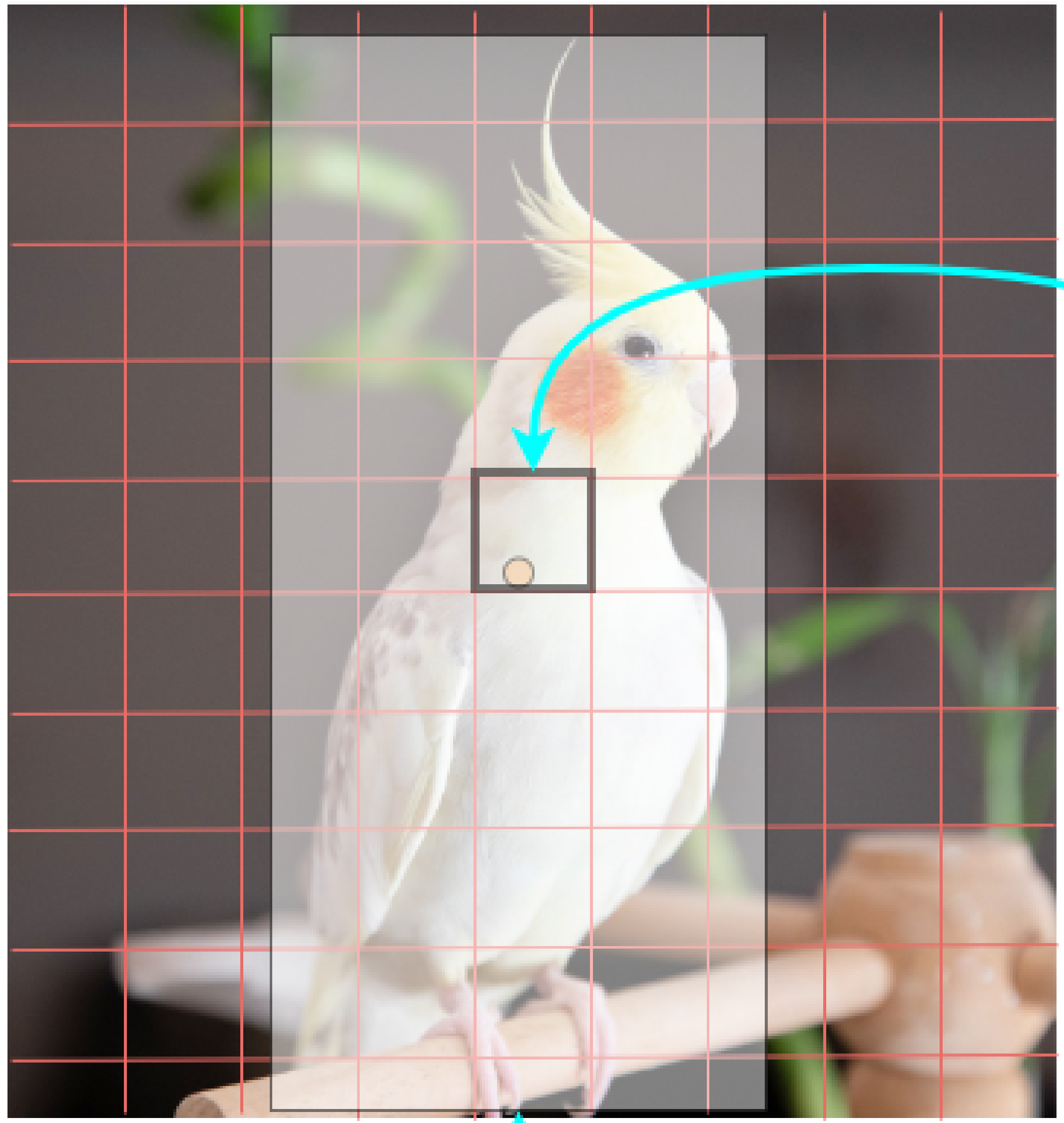
w = 224 / 448 = 0.50

h = 143 / 448 = 0.32

Only one cell is responsible for an object.

YOLO [You Only Look Once]

imagen :H x W



grilla
(feature map)
SxS (S=8)

$$GT = \underbrace{x^G, y^G}_{\text{center}}, w^G, h^G$$

$$i_x = \frac{x^G}{w} \cdot S$$

$$i_y = \frac{y^G}{H} \cdot S$$

x	y	w	h

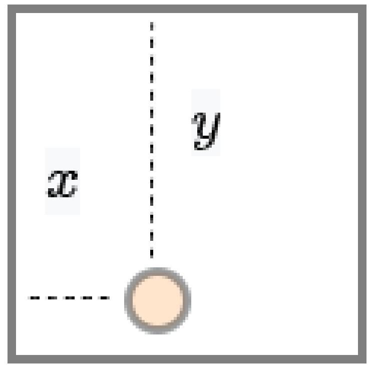
$$x = i_x - \lfloor i_x \rfloor$$

$$y = i_x - \lfloor i_y \rfloor$$

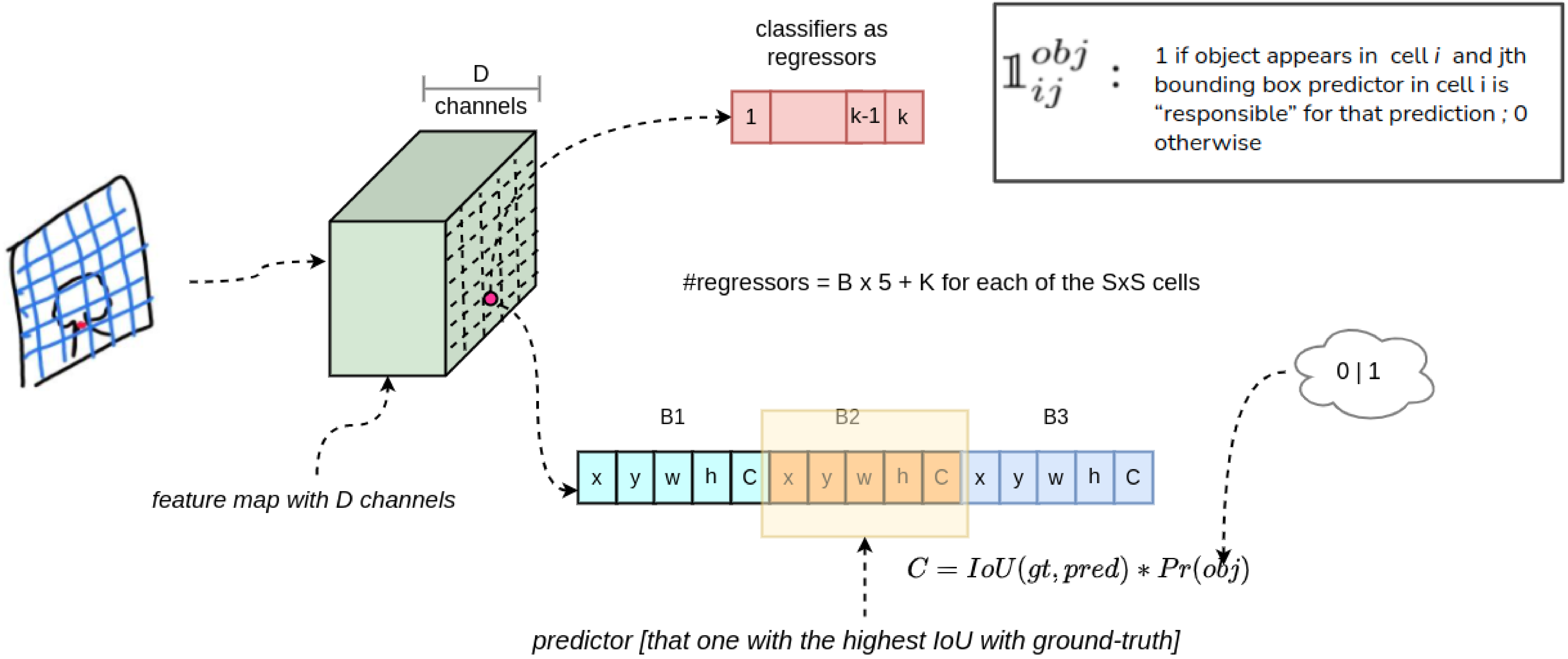
$$w = w^G / W$$

$$h = h^G / H$$

$(\lfloor i_x \rfloor, \lfloor i_y \rfloor)$



YOLO [You Only Look Once]



YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding
predictor to be responsible for each object. We assign one predictor to be "responsible" for predicti
object based on which prediction has the highest current IOU with the ground truth.

YOLO [You Only Look Once]

YOLO Loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

this term penalizes bad localization of center of cells

this term penalizes the bounding box with inaccurate height and width. The square root reflects that small deviations in large boxes matter less than in small boxes

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

this term tries to make the confidence score equal to the IOU between the object and the prediction when there is one object

this tries to make confidence score close to 0 when there is no object in the cell

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

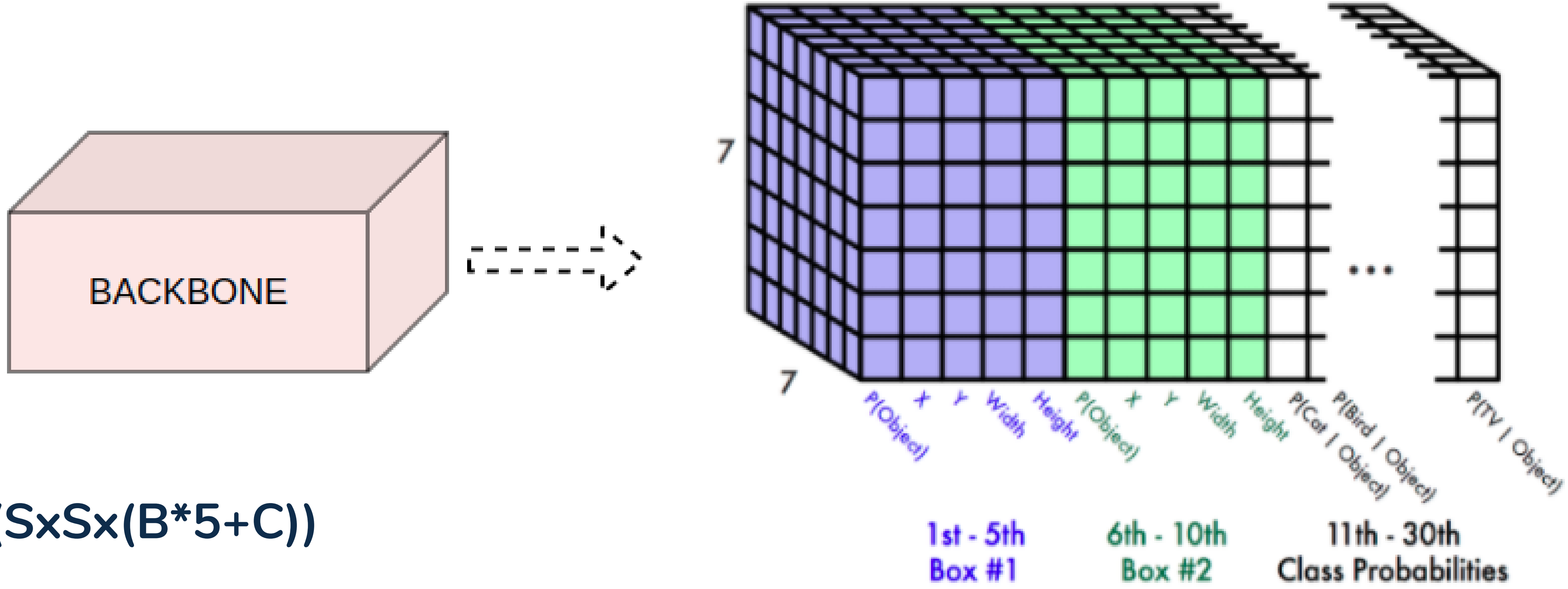
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

This is a simple classification loss

https://hackernoon.com/understanding-yolo-f5a74bbc7967

YOLO [You Only Look Once]

YOLO Model



Head: regressor de (SxSx(B*5+C))
S = 7
B = 2
C = número de clases, e.g. 20 en Pascal VOC

output → 7x7x30

https://hackernoon.com/understanding-yolo-f5a74bbc7967

YOLO [You Only Look Once]

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

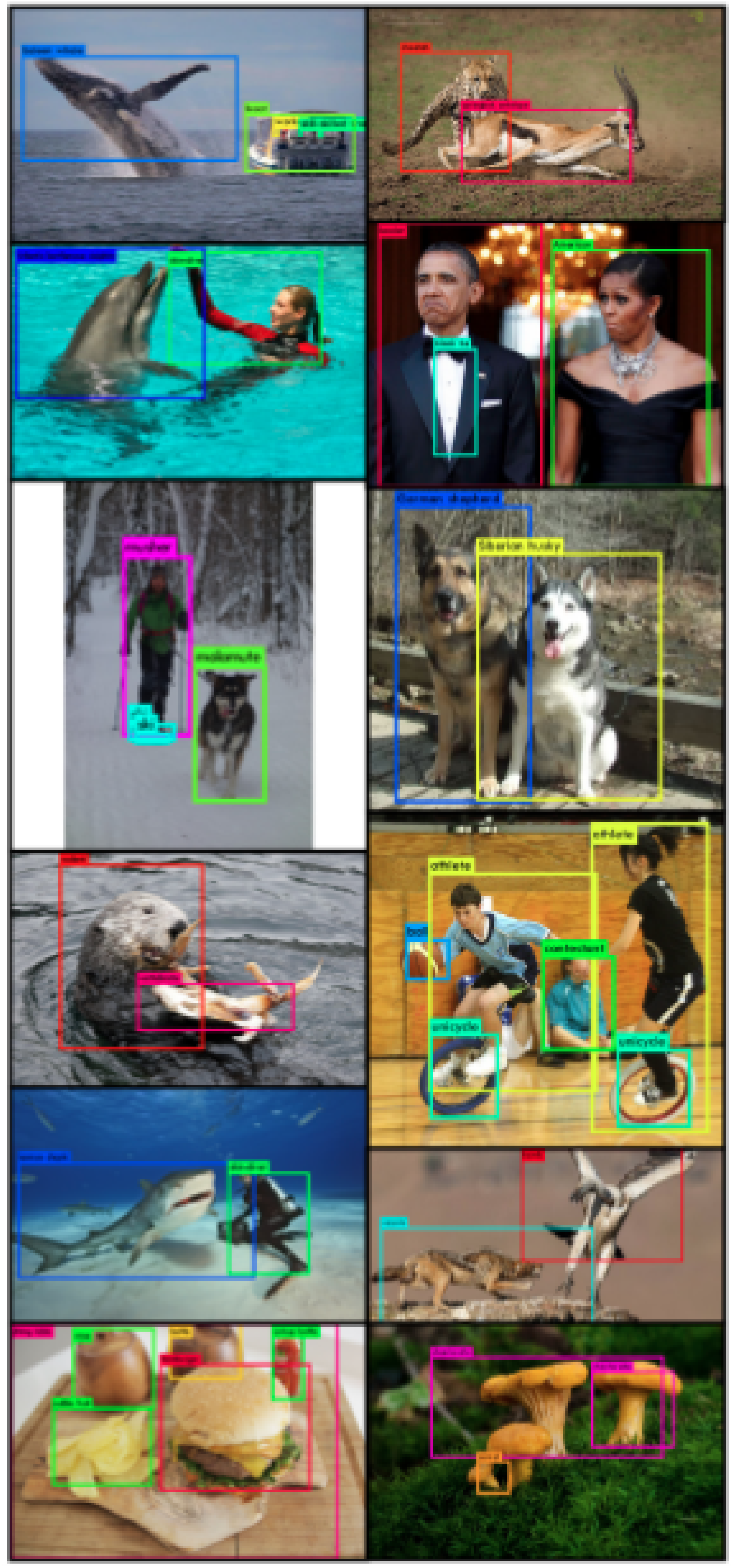
Table 1: Real-Time Systems on PASCAL VOC 2007. Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well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

YOLO [You Only Look Once]

YOLO 9000 (Yolo-v2)

An Improved version of YOLO (anchors are included)

https://arxiv.org/pdf/1612.08242.pdf



YOLOv2 [You Only Look Once]

Improvements

- Batch normalization
- A higher resolution classifier ($224 \to 448$, $S = 13$)
- Anchors for bounding boxes
- Number of anchors is estimated by clustering on training data.

YOLOv2 [You Only Look Once]

Anchors

Faster-RCNN: tx,ty are not limited, and the center of a predicted box can fall anywhere in the image producing instability.

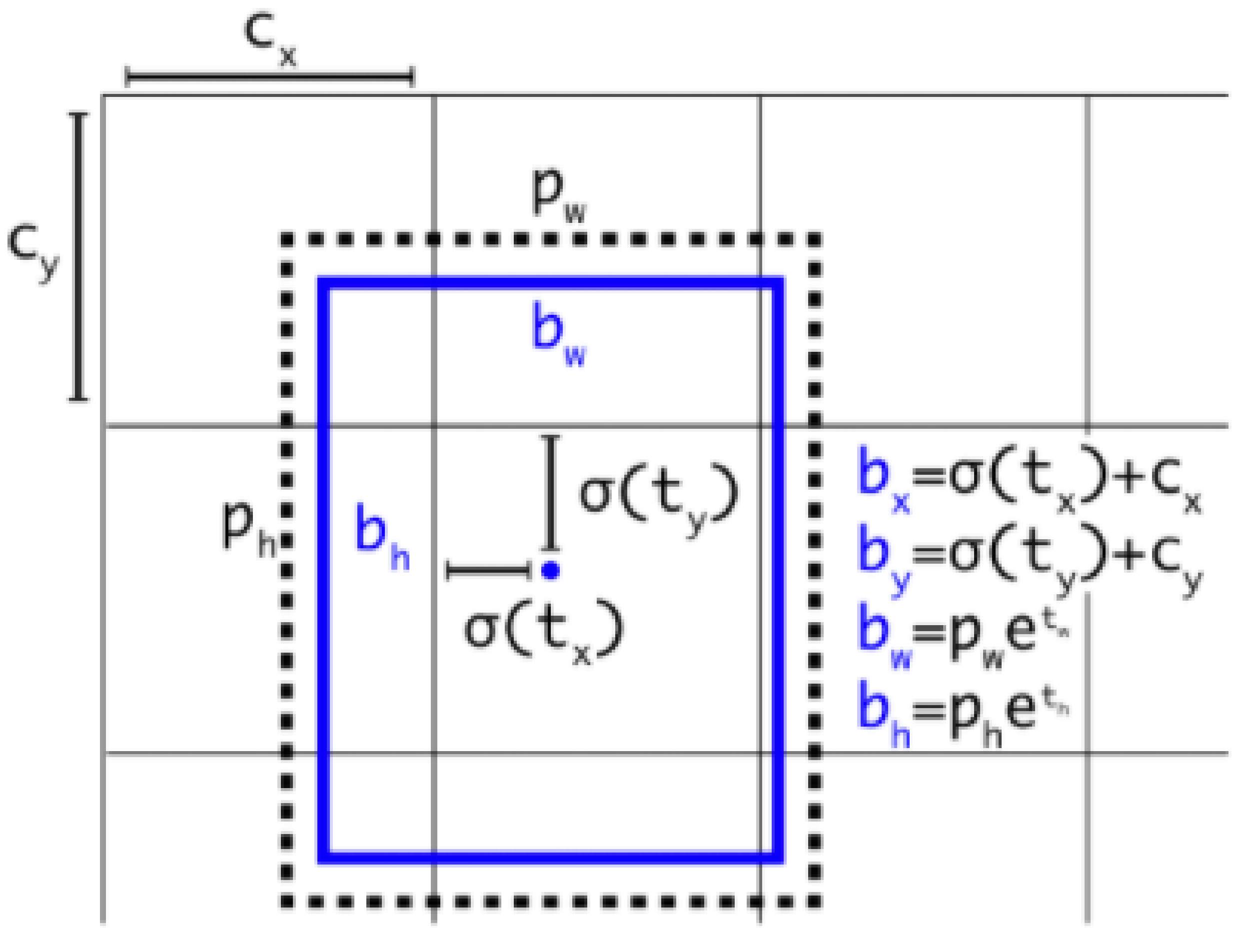
$$x = (t_x * w_a) + x_a$$

$$y = (t_y * h_a) + y_a$$

YOLOv2 [You Only Look Once]

Anchors

YOLOv2: x,y are constrained to fall within the corresponding cell



tx	ty	tw	th

sigma is the sigmoid function varying between 0 and 1

YOLOv2 [You Only Look Once]

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

YOLOv2 [You Only Look Once]

		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

Table 5: Results on COCO `test-dev2015`. Table adapted from [11]

COCO

https://cocodataset.org

COCO
Common Objects in Context

info@cocodataset.org

Home People Dataset Tasks Evaluate

News

- We are pleased to announce the LVIS 2021 Challenge and Workshop to be held at ICCV.
- Please note that there will not be a COCO 2021 Challenge, instead, we encourage people to participate in the LVIS 2021 Challenge.
- We have partnered with the team behind the open-source tool FiftyOne to make it easier to download, visualize, and evaluate COCO
- FiftyOne is an open-source tool facilitating visualization and access to COCO data resources and serves as an evaluation tool for model analysis on COCO.

What is COCO?

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- Object segmentation
- Recognition in context
- Superpixel stuff segmentation
- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image
- 250,000 people with keypoints

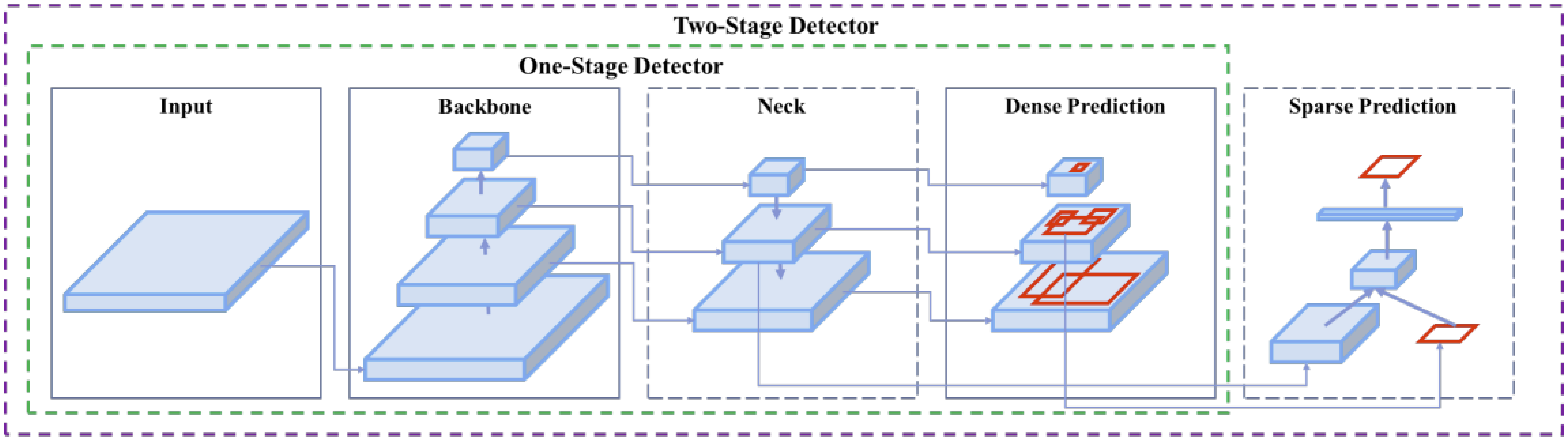
Collaborators

Tsung-Yi Lin Google Brain
Genevieve Patterson MSR, Trash TV
Matteo R. Ronchi Caltech
Yin Cui Google
Michael Maire TTI-Chicago
Serge Belongie Cornell Tech
Lubomir Bourdev WaveOne, Inc.
Ross Girshick FAIR
James Hays Georgia Tech
Pietro Perona Caltech
Deva Ramanan CMU
Larry Zitnick FAIR
Piotr Dollár FAIR

Sponsors

CVDF
Microsoft
facebook
Mighty Ai

Dataset examples



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

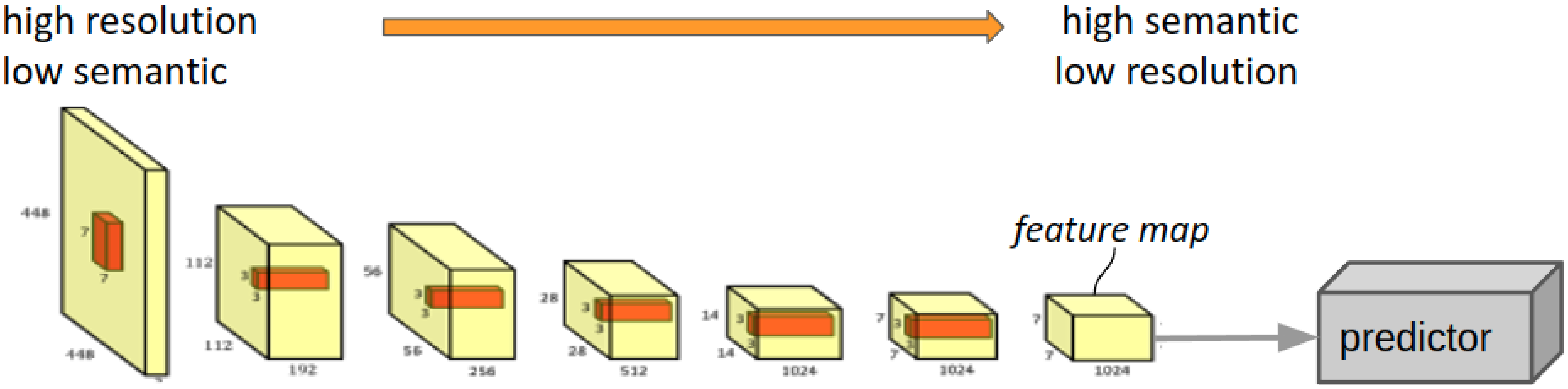
Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

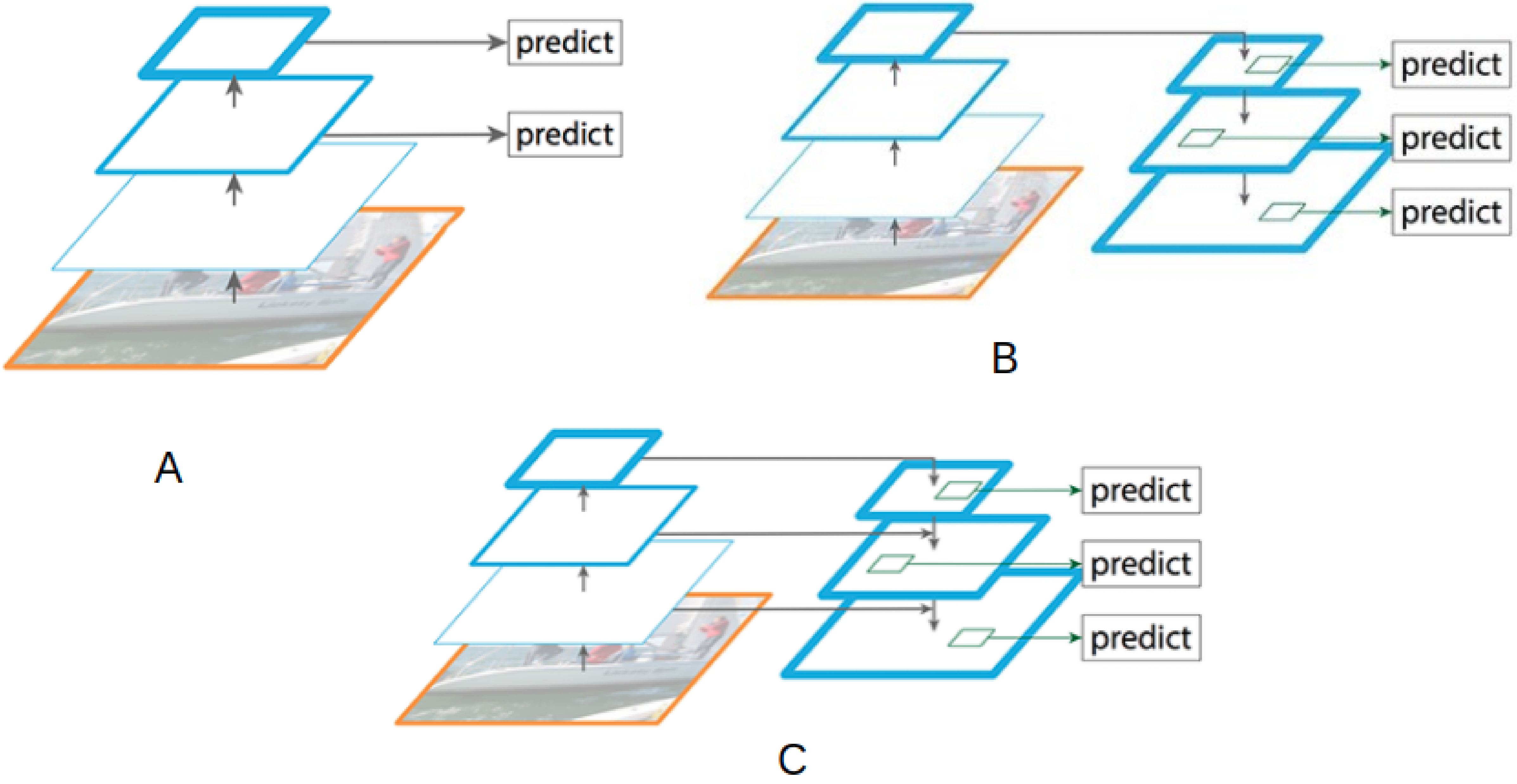
https://arxiv.org/pdf/2004.10934.pdf

A convnet [single-scale]



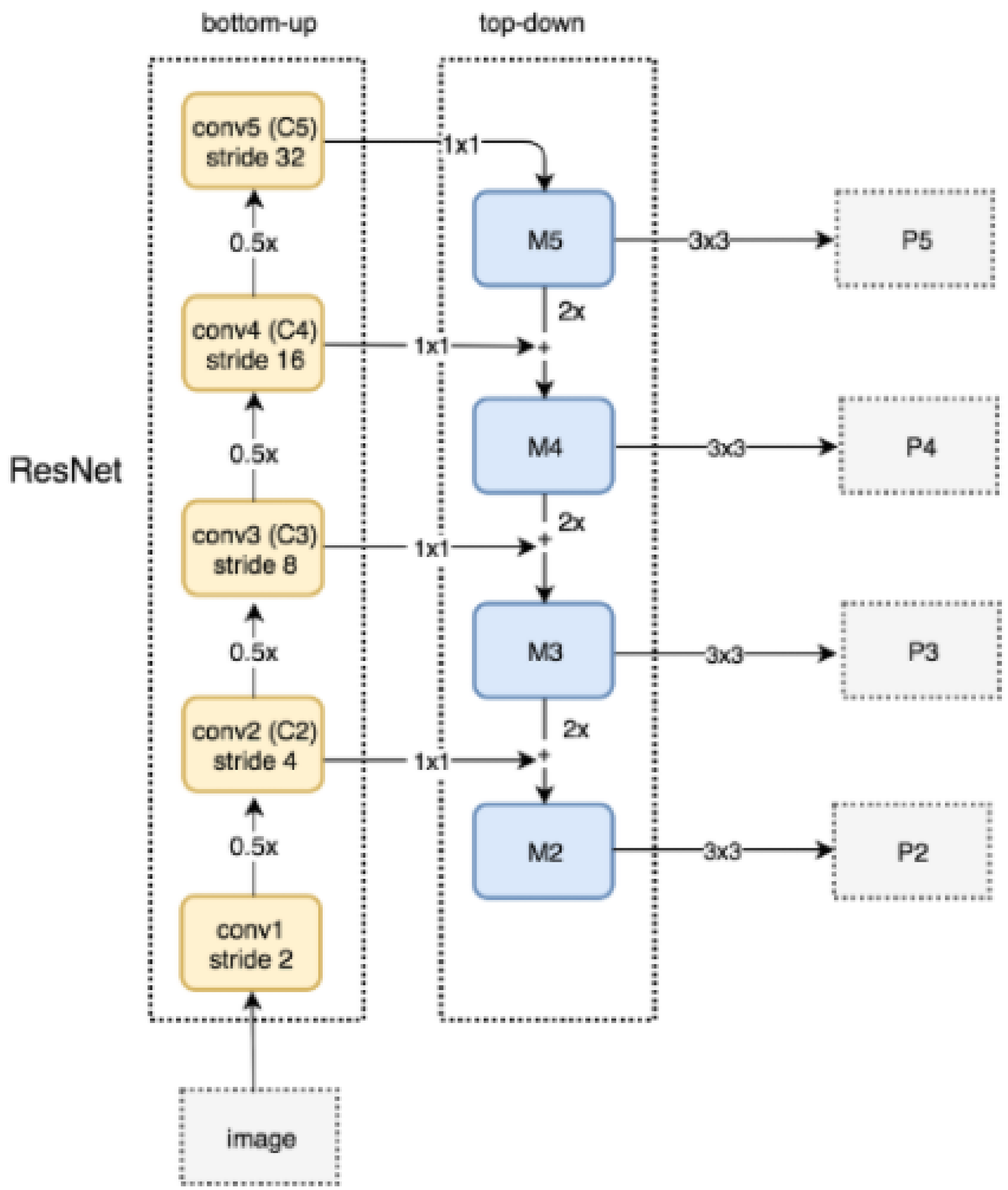
prediction head processes a single-scale feature map

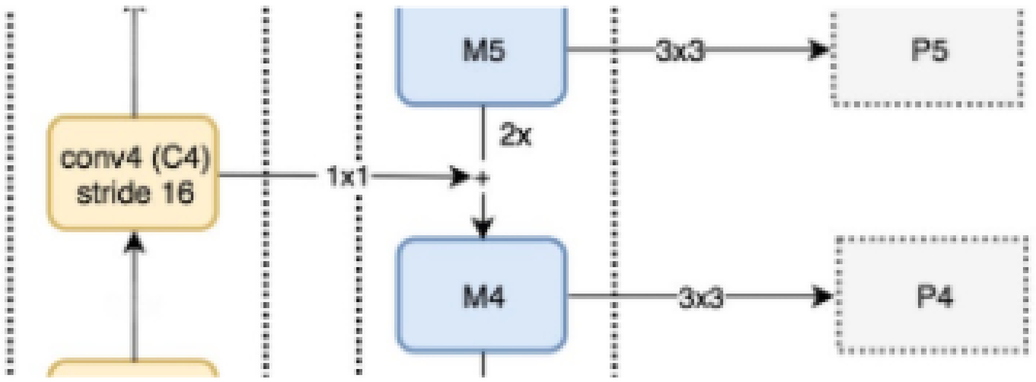
Feature Pyramid Network



https://arxiv.org/pdf/1612.03144.pdf

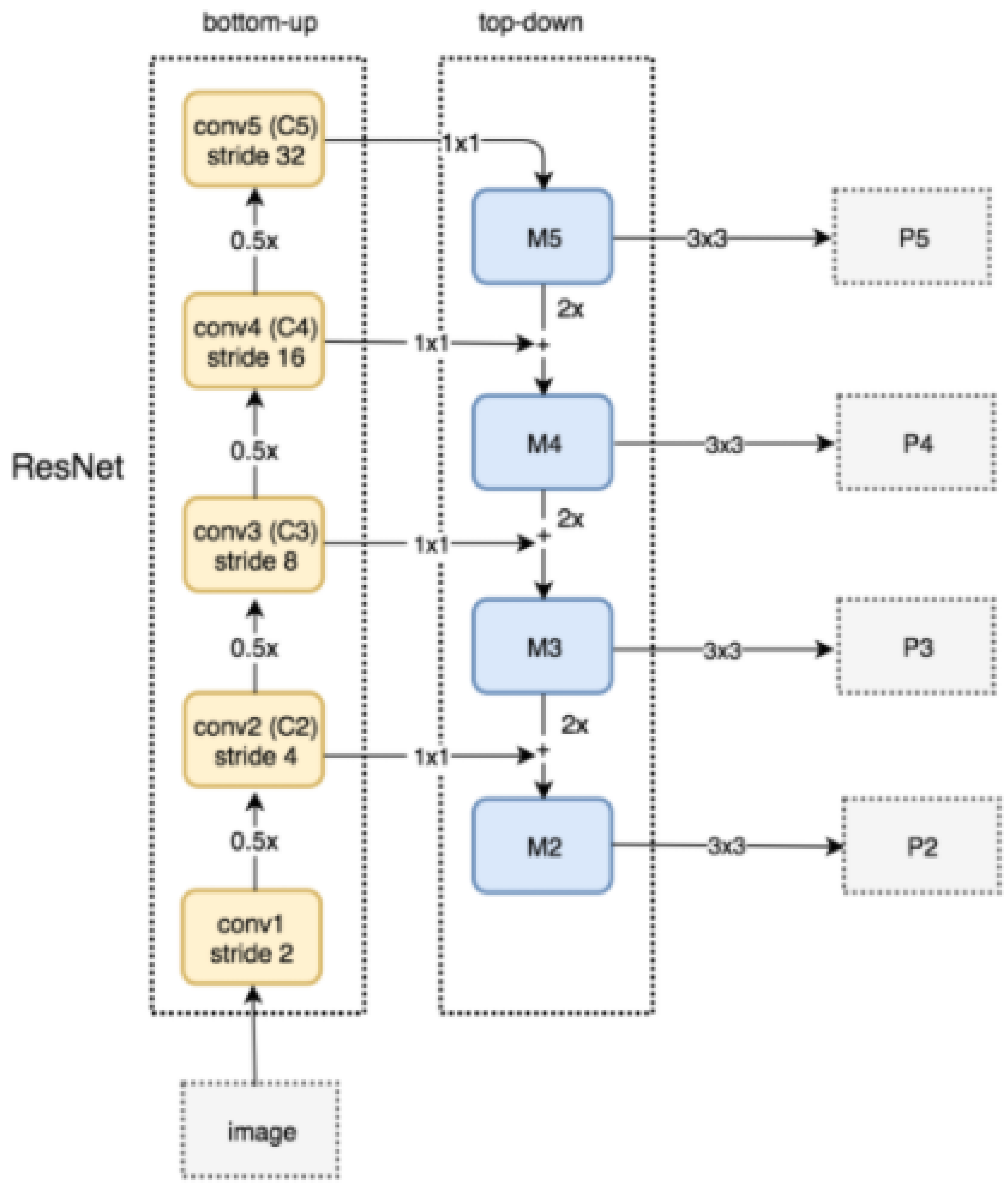
Feature Pyramid Network



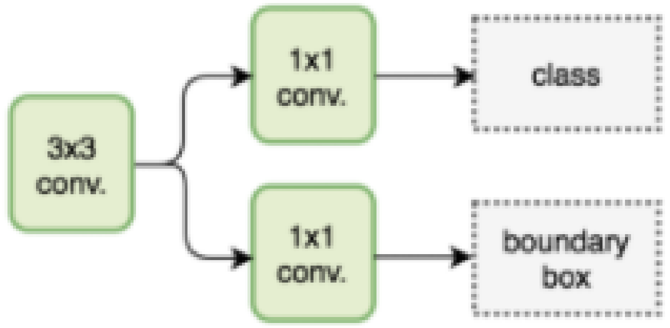


https://arxiv.org/pdf/1612.03144.pdf

Feature Pyramid Network



We note that the parameters of the heads are shared across all feature pyramid levels; we have also evaluated the alternative without sharing parameters and observed similar accuracy. The good performance of sharing parameters indicates that all levels of our pyramid share similar semantic levels. This advantage is analogous to that of using a featurized image pyramid, where a common head classifier can be applied to features computed at any image scale.

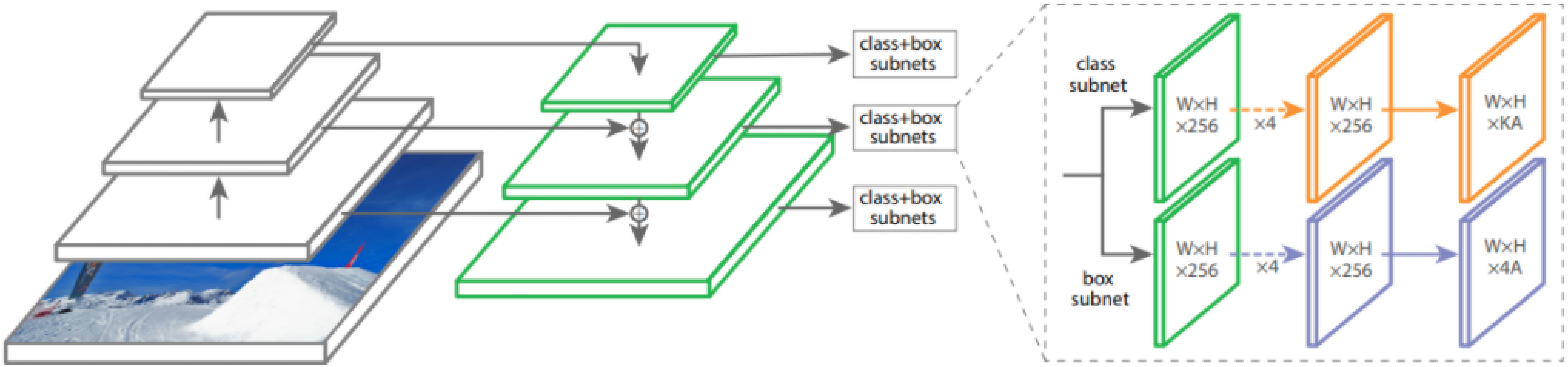


predictor head

shared weights

https://arxiv.org/pdf/1612.03144.pdf

Feature Pyramid Network

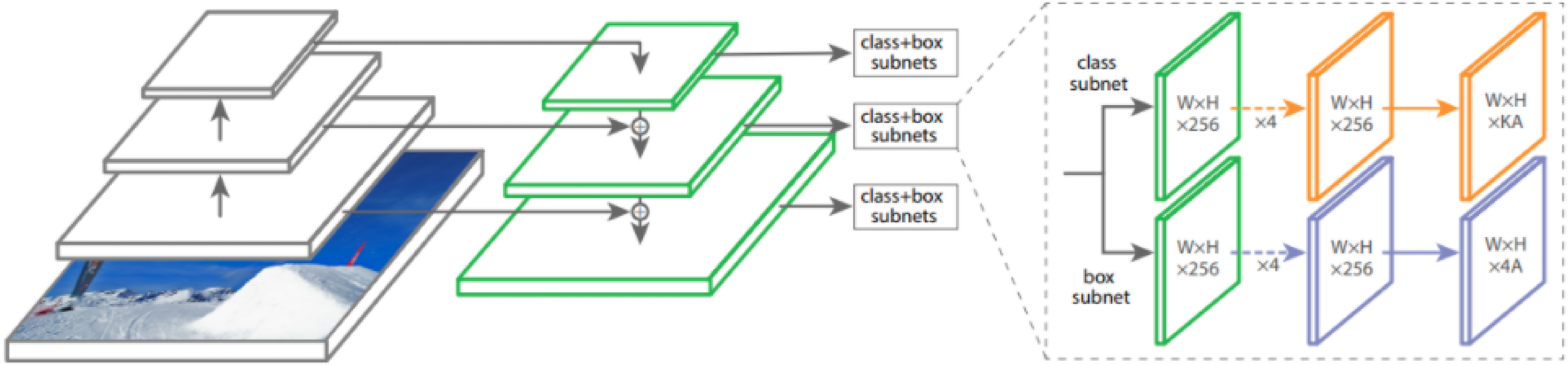


(a) ResNet (b) feature pyramid net (c) class subnet (top) (d) box subnet (bottom)

Because all levels of the pyramid use shared classifiers/regressors as in a traditional featurized image pyramid, we fix the feature dimension (numbers of channels, denoted as d) in all the feature maps. We set $d = 256$ in this paper and thus all extra convolutional layers have 256-channel outputs. There are no non-linearities in these extra layers, which we have empirically found to have minor impacts.

https://arxiv.org/pdf/1612.03144.pdf

Feature Pyramid Network



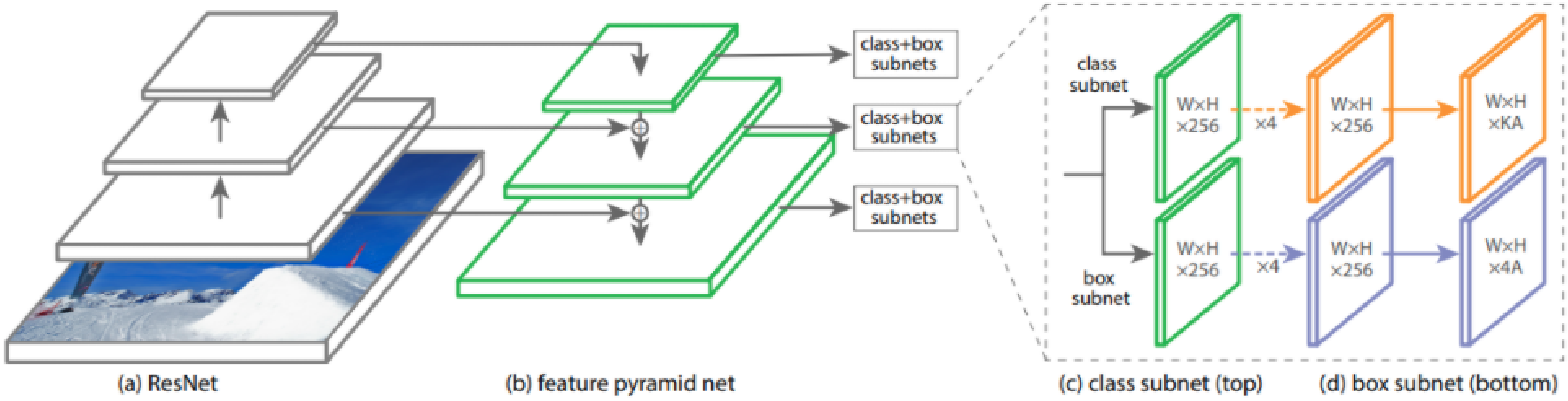
(a) ResNet (b) feature pyramid net (c) class subnet (top) (d) box subnet (bottom)

For region proposal (RPN)

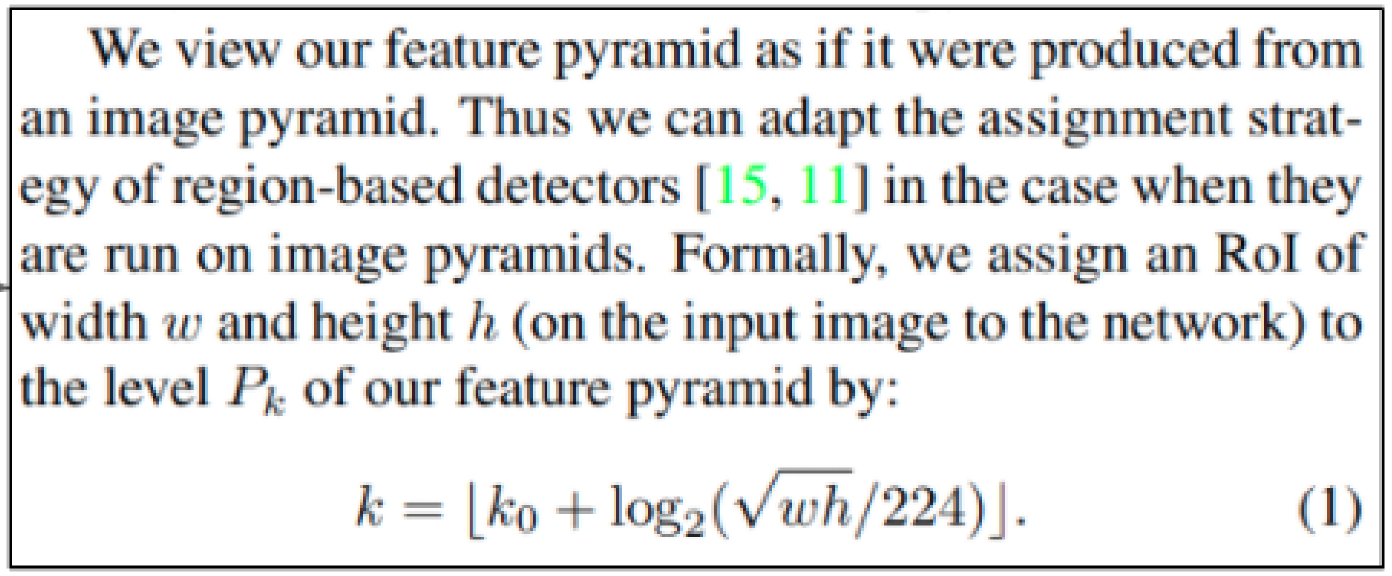
anchors on a specific level. Instead, we assign anchors of a single scale to each level. Formally, we define the anchors to have areas of $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ pixels on $\{P_2, P_3, P_4, P_5, P_6\}$ respectively.[1] As in [29] we also use anchors of multiple aspect ratios {1:2, 1:1, 2:1} at each level. So in total there are 15 anchors over the pyramid.

https://arxiv.org/pdf/1612.03144.pdf

Feature Pyramid Network



For classifier (Fast R-CNN)

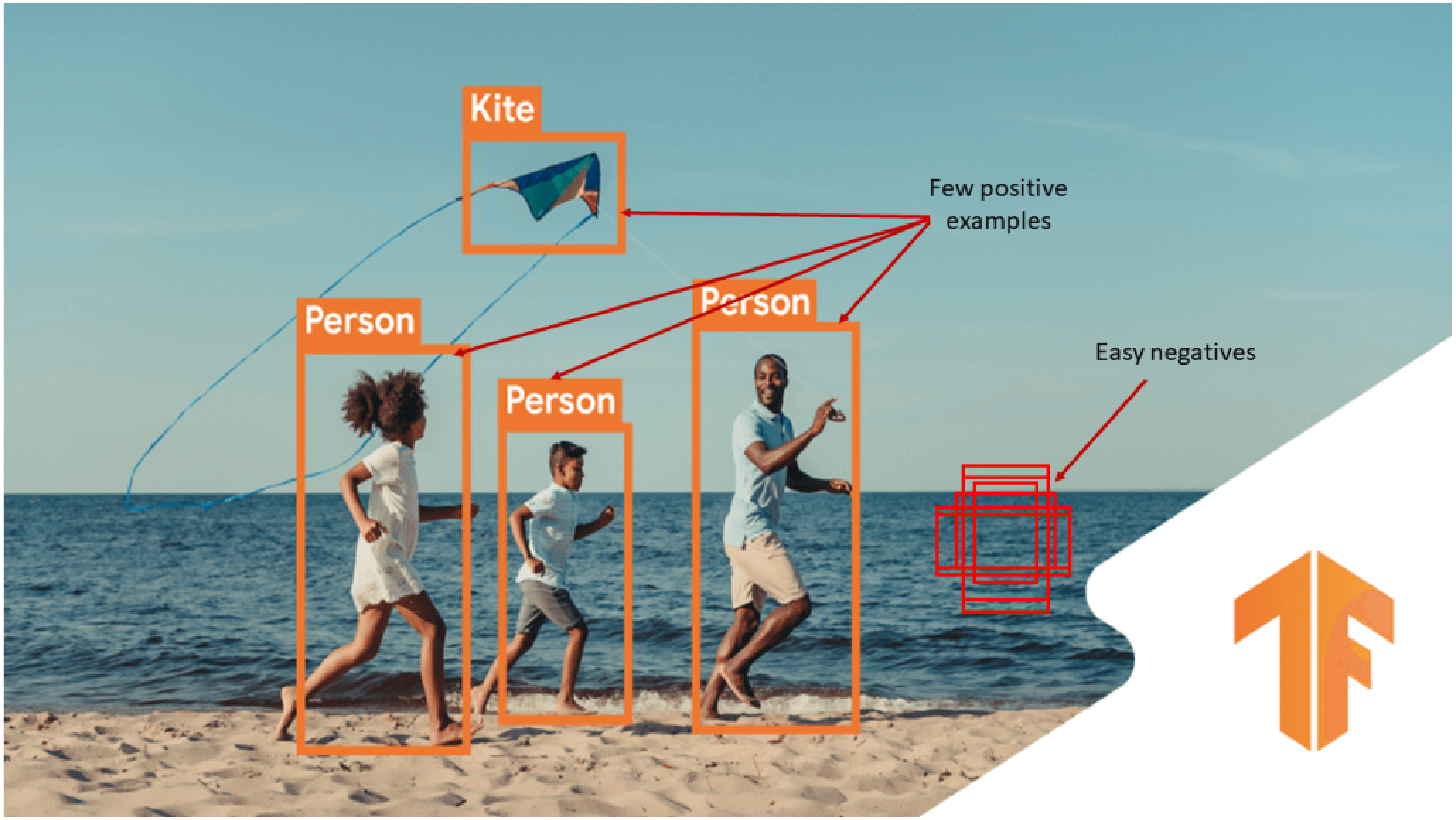


We view our feature pyramid as if it were produced from an image pyramid. Thus we can adapt the assignment strategy of region-based detectors [15, 11] in the case when they are run on image pyramids. Formally, we assign an RoI of width w and height h (on the input image to the network) to the level P_k of our feature pyramid by:

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor. \quad (1)$$

https://arxiv.org/pdf/1612.03144.pdf

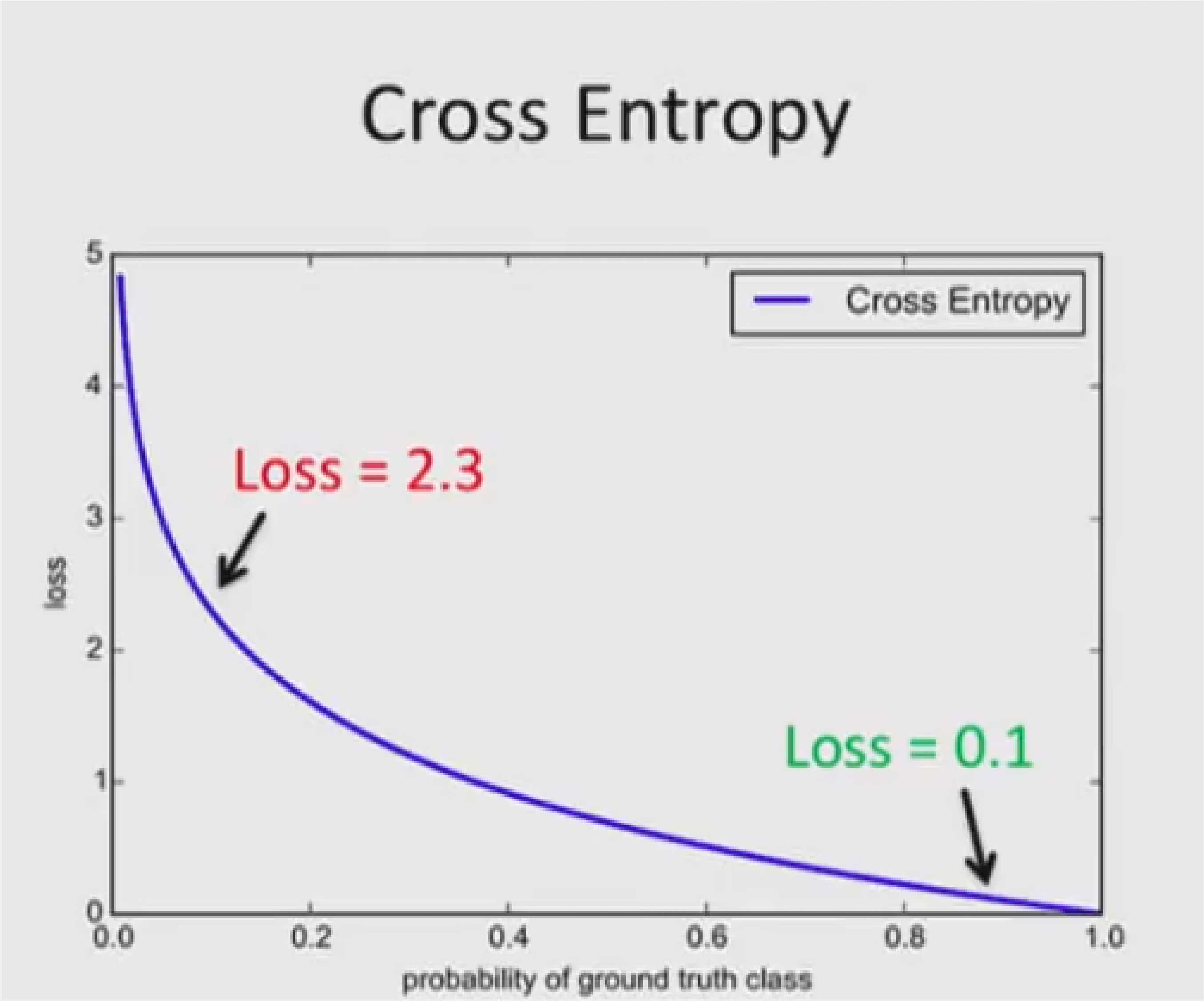
Focal Loss

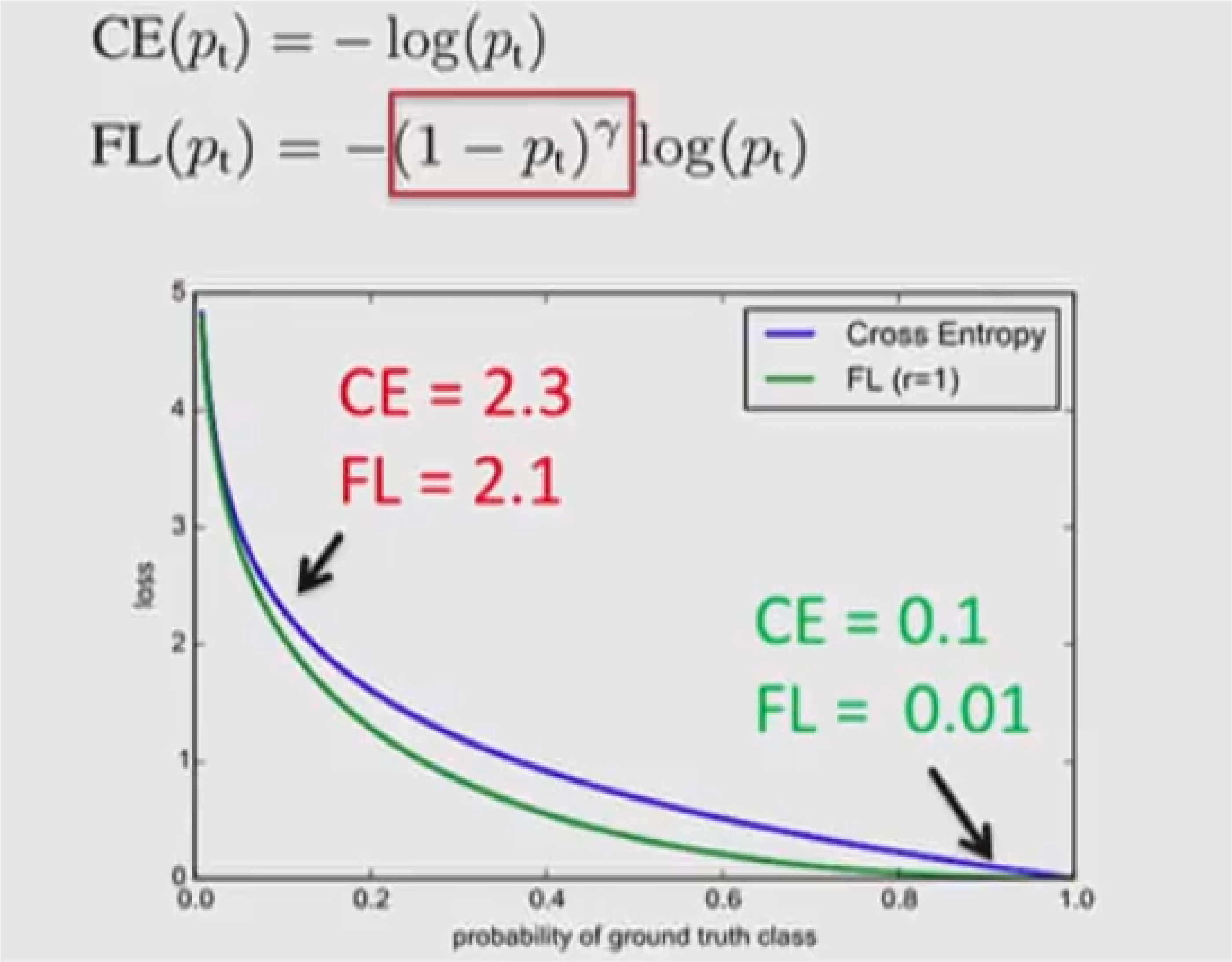


https://arxiv.org/pdf/1708.02002.pdf

Focal Loss

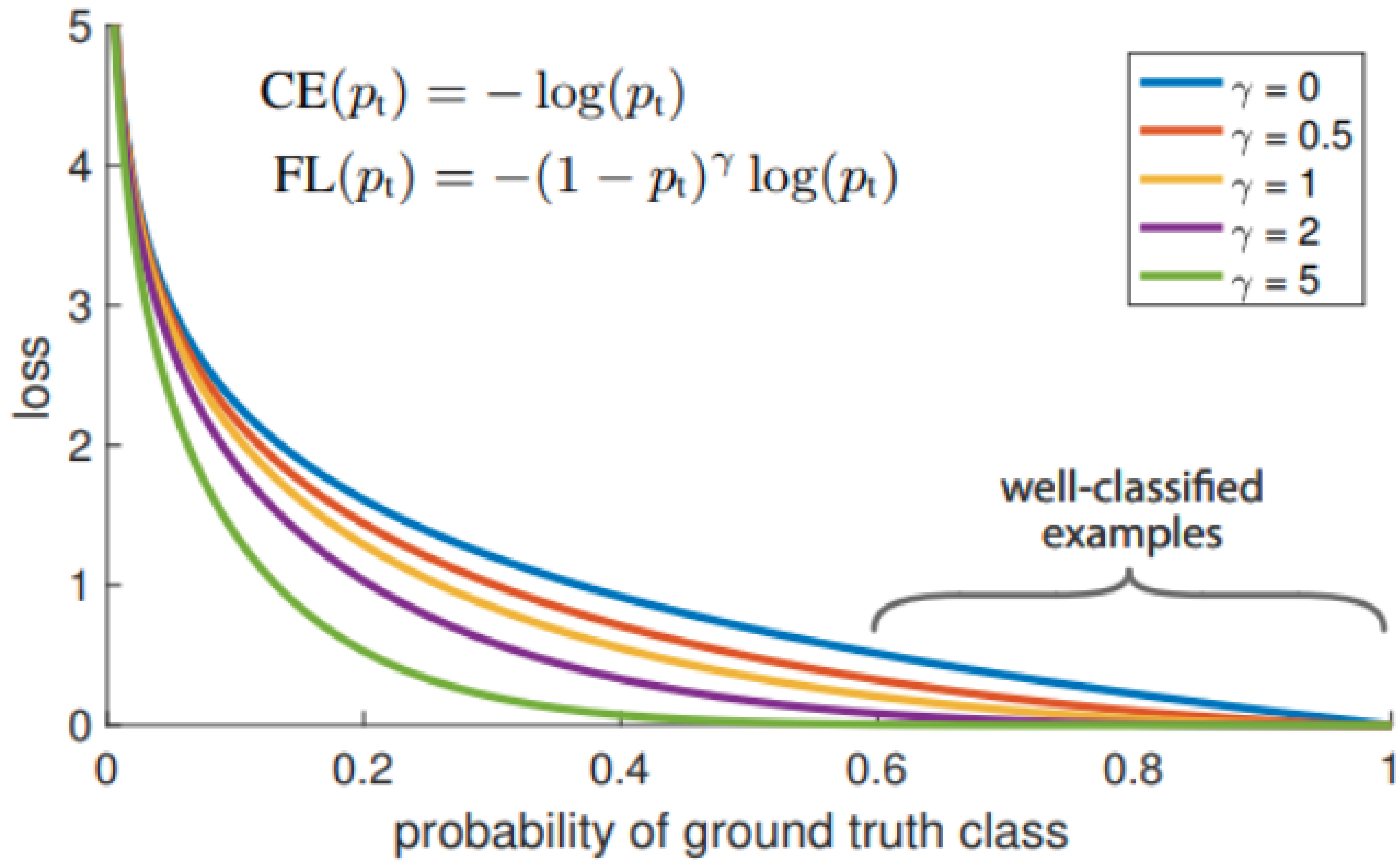
100000 easy : 100 hard examples





Loss: centrado en ejemplos fáciles

Focal Loss



$\mathrm{CE}(p_t) = -\log(p_t)$
$\mathrm{FL}(p_t) = -(1-p_t)^\gamma \log(p_t)$
$\gamma = 0$
$\gamma = 0.5$
$\gamma = 1$
$\gamma = 2$
$\gamma = 5$
well-classified examples
loss
probability of ground truth class
0
1
2
3
4
5
0
0.2
0.4
0.6
0.8
1

Focal Loss

- α-balanced Cross entropy

$$\mathrm{CE}(p_t) = -\alpha_t \log(p_t)$$

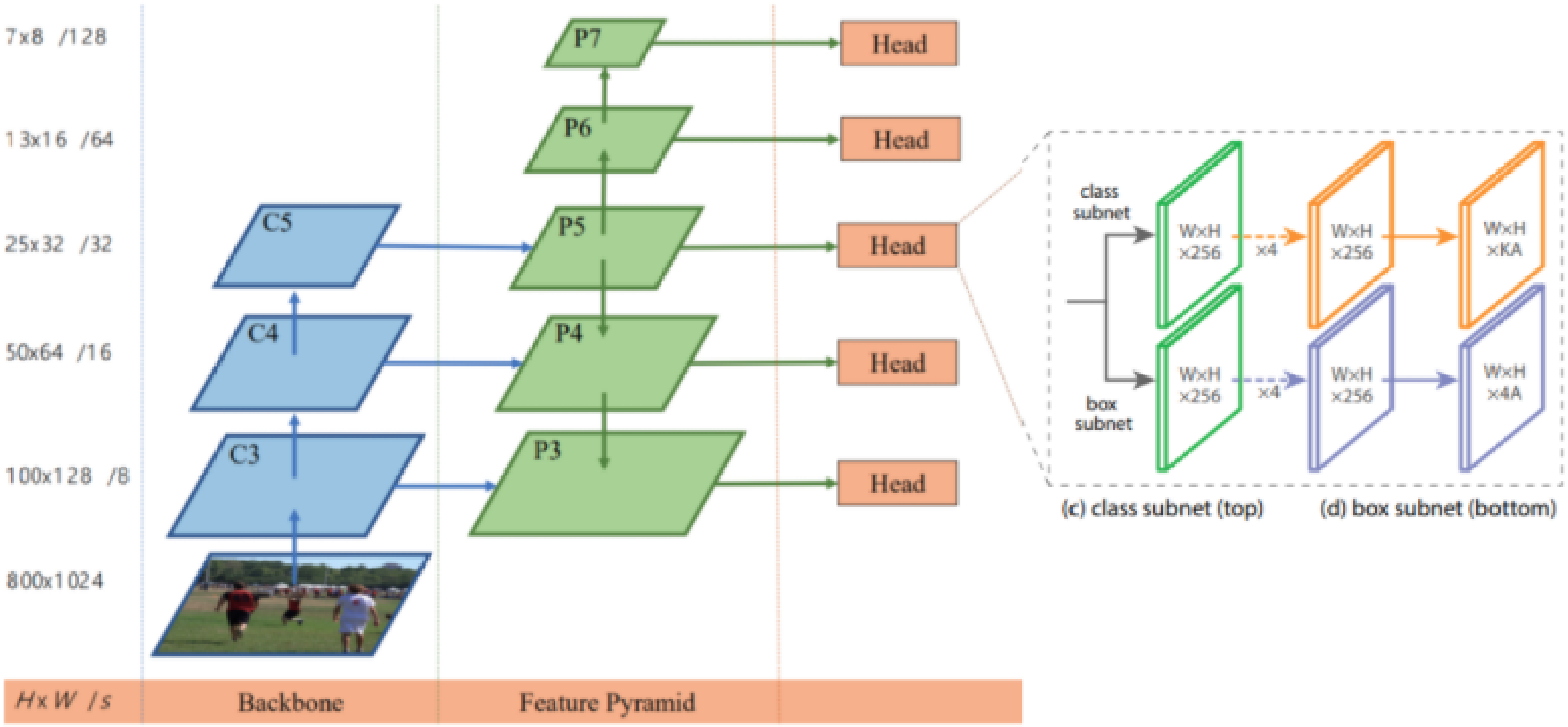
- α-balanced Focal Loss

$$\mathrm{FL}(p_t) = -\alpha_t (1 - p_t)^\gamma \log(p_t)$$

- γ: focus more on hard examples
- α: offset class imbalance of number of examples

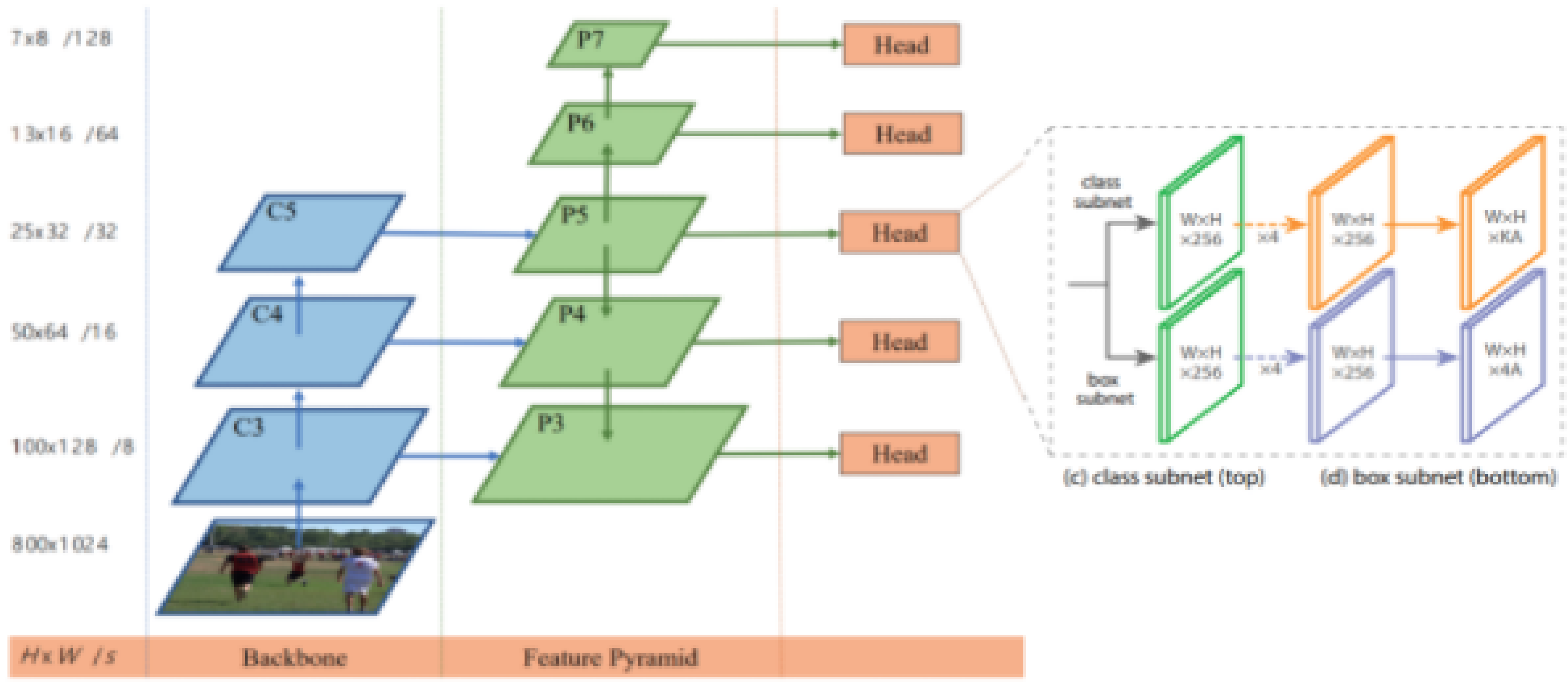
RetinaNet

One-stage object detector combining FPN + Focal Loss



RetinaNet

One-stage object detector combining FPN + Focal Loss



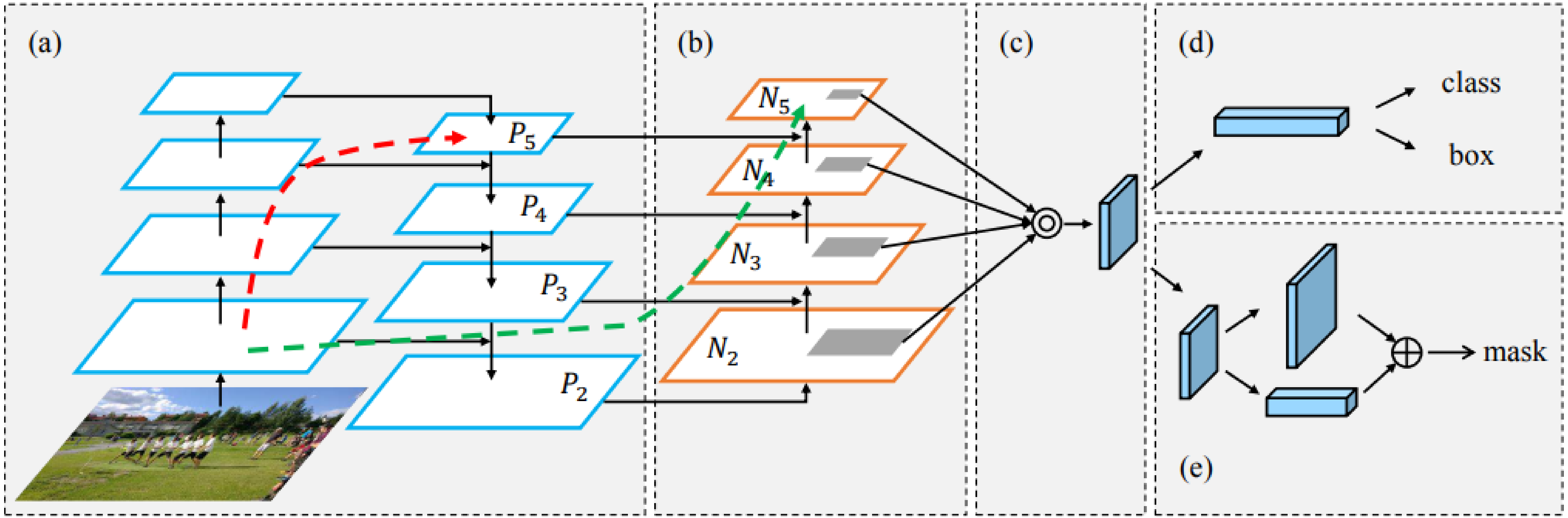
Anchors: We use translation-invariant anchor boxes similar to those in the RPN variant in [20]. The anchors have areas of 32^2 to 512^2 on pyramid levels P_3 to P_7, respectively. As in [20], at each pyramid level we use anchors at three aspect ratios $\{1{:}2, 1{:}1, 2{:}1\}$. For denser scale coverage than in [20], at each level we add anchors of sizes $\{2^0, 2^{1/3}, 2^{2/3}\}$ of the original set of 3 aspect ratio anchors. This improve AP in our setting. In total there are $A = 9$ anchors per level and across levels they cover the scale range 32 - 813 pixels with respect to the network's input image.

RetinaNet

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

Table 2. **Object detection** *single-model* results (bounding box AP), *vs.* state-of-the-art on COCO `test-dev`. We show results for our RetinaNet-101-800 model, trained with scale jitter and for 1.5× longer than the same model from Table 1e. Our model achieves top results, outperforming both one-stage and two-stage models. For a detailed breakdown of speed versus accuracy see Table 1e and Figure 2.

Path Aggregation Network (PAN)



Path Aggregation Network for Instance Segmentation

https://openaccess.thecvf.com/content_cvpr_2018/papers/Liu_Path_Aggregation_Network_CVPR_2018_paper.pdf

Path Aggregation Network

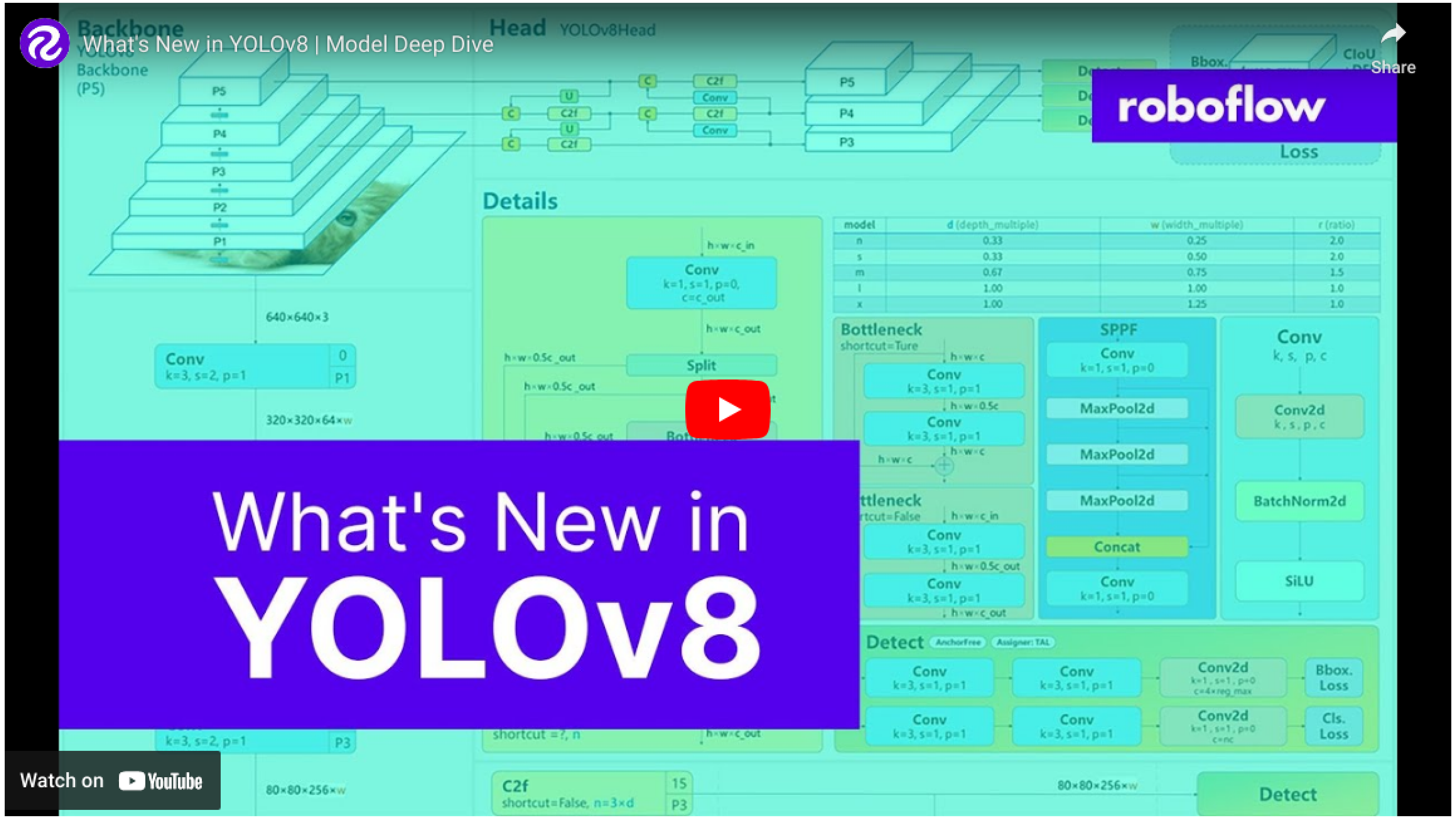
	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_{S}	AP^{bb}_{M}	AP^{bb}_{L}
Champion 2015 [23]	37.4	59.0	40.2	18.3	41.7	52.9
Champion 2016 [27]	41.6	62.3	45.6	24.0	43.9	55.2
Our Team 2017	**51.0**	**70.5**	**55.8**	**32.6**	**53.9**	**64.8**

Table 7. Box AP of COCO Object Detection Challenge in different years on *test-dev*.

Path Aggregation Network for Instance Segmentation

https://openaccess.thecvf.com/content_cvpr_2018/papers/Liu_Path_Aggregation_Network_CVPR_2018_paper.pdf

YOLO8



https://blog.roboflow.com/whats-new-in-yolov8/

YOLO: A Brief History

YOLO (You Only Look Once), a popular object detection and image segmentation model, was developed by Joseph Redmon and Ali Farhadi at the University of Washington. Launched in 2015, YOLO quickly gained popularity for its high speed and accuracy.

- YOLOv2, released in 2016, improved the original model by incorporating batch normalization, anchor boxes, and dimension clusters.
- YOLOv3, launched in 2018, further enhanced the model's performance using a more efficient backbone network, multiple anchors and spatial pyramid pooling.
- YOLOv4 was released in 2020, introducing innovations like Mosaic data augmentation, a new anchor-free detection head, and a new loss function.
- YOLOv5 further improved the model's performance and added new features such as hyperparameter optimization, integrated experiment tracking and automatic export to popular export formats.
- YOLOv6 was open-sourced by Meituan in 2022 and is in use in many of the company's autonomous delivery robots.
- YOLOv7 added additional tasks such as pose estimation on the COCO keypoints dataset.
- YOLOv8 is the latest version of YOLO by Ultralytics. As a cutting-edge, state-of-the-art (SOTA) model, YOLOv8 builds on the success of previous versions, introducing new features and improvements for enhanced performance, flexibility, and efficiency. YOLOv8 supports a full range of vision AI tasks, including detection, segmentation, pose estimation, tracking, and classification. This versatility allows users to leverage YOLOv8's capabilities across diverse applications and domains.

https://docs.ultralytics.com/

Muchas Gracias

José Manuel Saavedra Rondo
jmsaavedrar@miuandes.cl