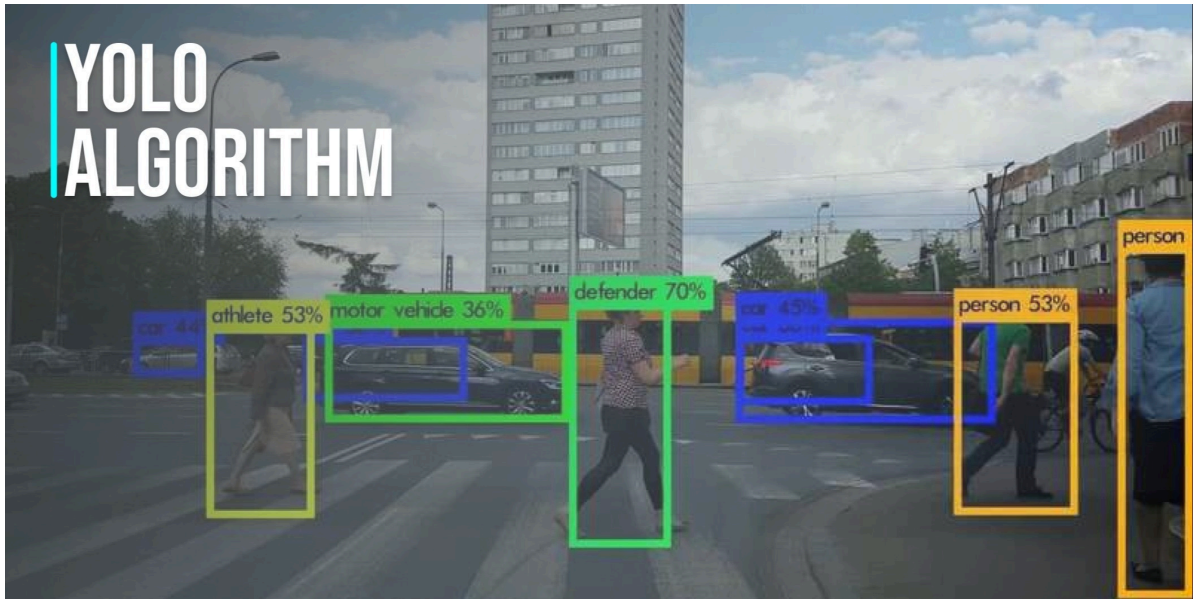
AI VIETNAM
All-in-One Course

Object Detection

(Part 4: YOLO and its variants algorithms)



Vinh Dinh Nguyen - PhD in Computer Science
Hung-An & Minh-Duc Bui - TA

Year 2023

Outline

- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

Outline

- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

Outline

- Object Detection Milestones: Transformer
- End-to-End Object Detection with Transformers
- Object Detection Milestones: YOLOs variants
- YOLO Motivations for OD
- YOLO-v1
- YOLO-v2

Object Detection Milestones

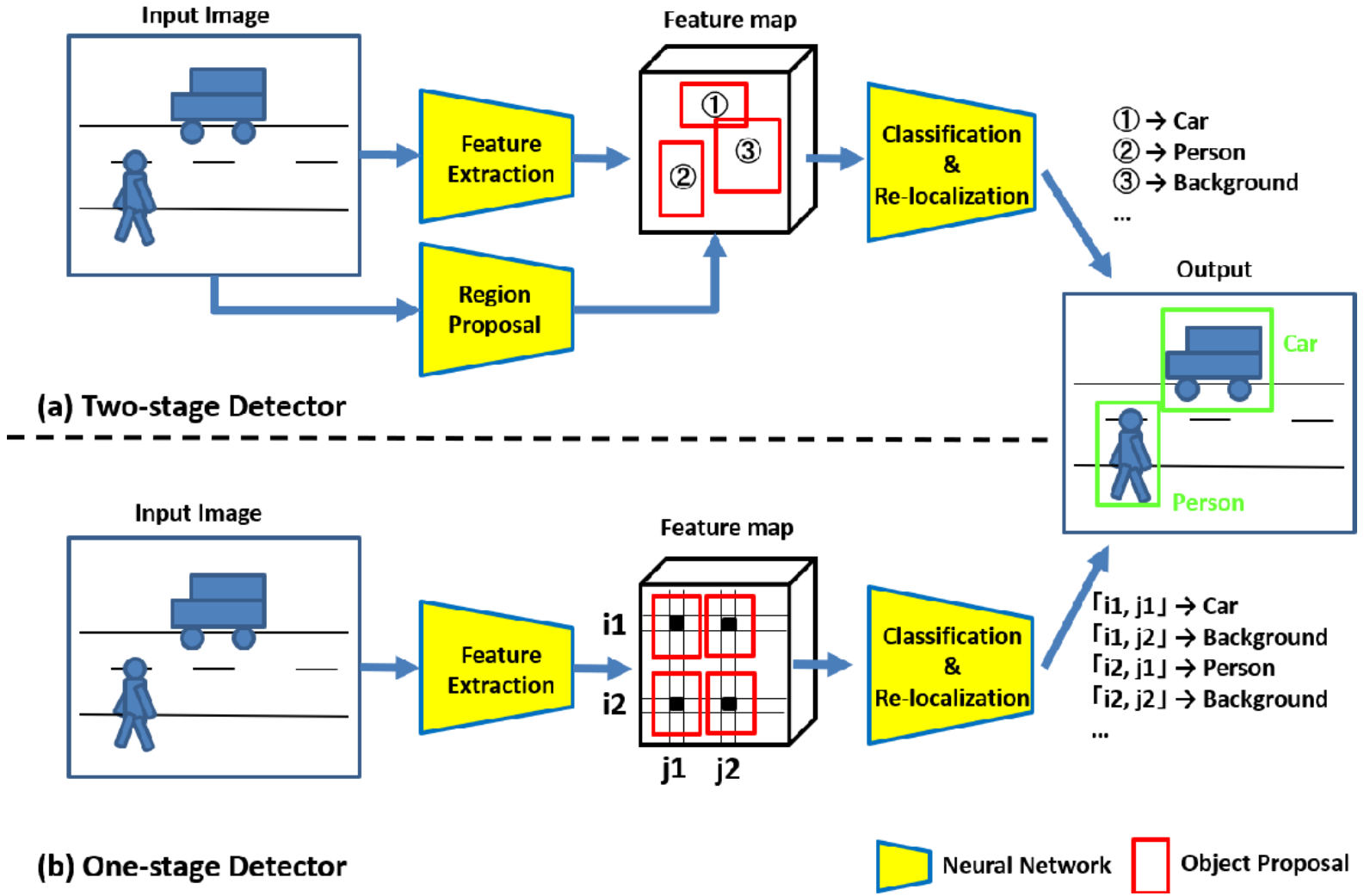
Milestones: Traditional Detectors
Viola Jones Detectors, SVM + HOG & DPM

Milestones: CNN based Two-stage Detectors
RCNN, SPPNet, Faster RCNN, Faster RCNN,..

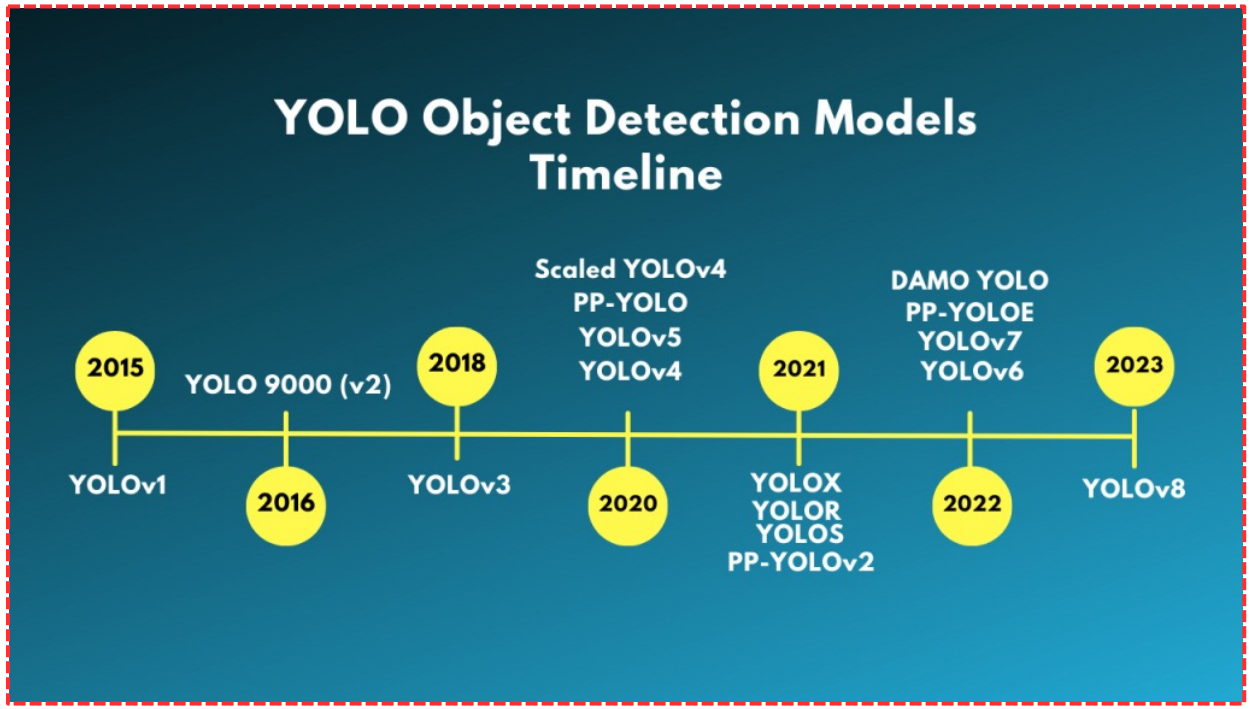
Milestones: CNN based One-stage Detectors
YOLO, SSD, RetinaNet, CornerNet, Center Net,..

Milestones: Transformer for OD
DETR, D-DETR, DINO,...

One-stage vs. Two-stage Detectors



Object Detection Milestones



R-CNN (2013.11) → OverFeat (ICLR' 14) → MultiBox (CVPR' 14) → SPP-Net (ECCV' 14) → MR-CNN (ICCV' 15) → DeepBox (ICCV' 15) → AttentionNet (ICCV' 15)

Fast R-CNN (ICCV' 15) → DeepProposal (ICCV' 15) → RPN (NIPS' 15) → Faster R-CNN (NIPS' 15) → YOLO v1 (CVPR' 16) → G-CNN (CVPR' 16) → AZNet (CVPR' 16)

Inside-OutsideNet(ION) (CVPR' 16) → HyperNet (CVPR' 16) → OHEM (CVPR' 16) → CRAFT (CVPR' 16) → MultiPathNet(MPN) (BMVC' 16) → SSD (ECCV' 16)

GBDNet (ECCV' 16) → CPF (ECCV' 16) → MS-CNN (ECCV' 16) → R-FCN (NIPS' 16) → PVANET (NIPSW' 16) → DeepID-Net (PAMI' 16) → NoC (TPAMI' 16) → DSSD (Arxiv' 17) → TDM (CVPR' 17)

Feature Pyramid Net(FPN) (CVPR' 17) → YOLO v2 (CVPR' 17) → RON (CVPR' 17) → DCN (ICCV' 17) → DeNet (ICCV' 17) → CoupleNet (ICCV' 17) → RetinaNet (ICCV' 17)

Mask R-CNN (ICCV' 17) → DSOD (ICCV' 17) → SMN (ICCV' 17) → YOLO v3 (Arxiv' 18) → SIN (CVPR' 18) → STDN (CVPR' 18) → RefineDet (CVPR' 18) → RFBNet (ECCV' 18) → ···

Z. Zou, K. Chen, Z. Shi, Y. Guo and J. Ye, "Object Detection in 20 Years: A Survey," in Proceedings of the IEEE, vol. 111, no. 3, pp. 257-276, March 2023, doi: 10.1109/JPROC.2023.3238524.

YOLOs vs DERTs

arXiv > cs > arXiv:2304.08069

Search... Help | Advanced

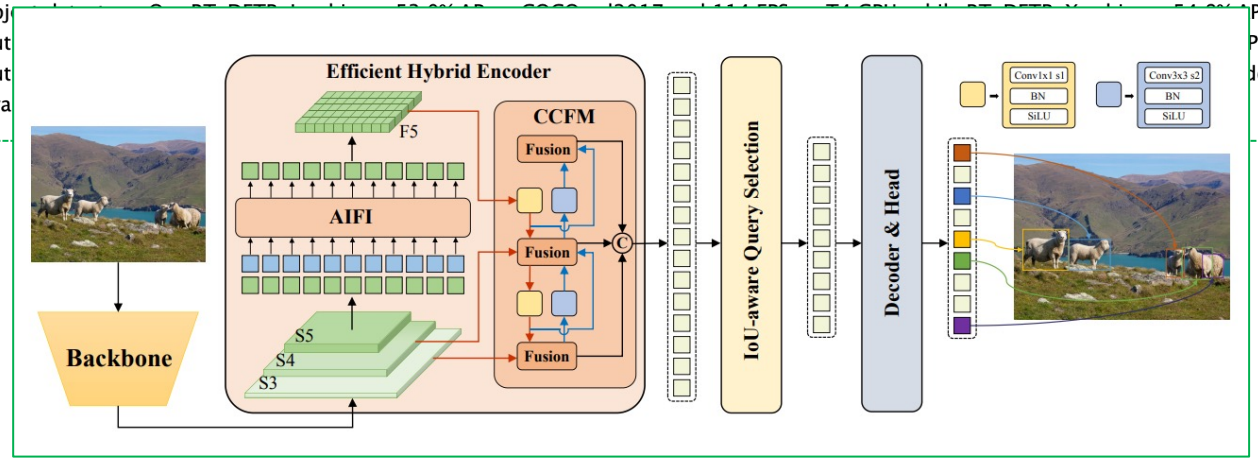
Computer Science > Computer Vision and Pattern Recognition

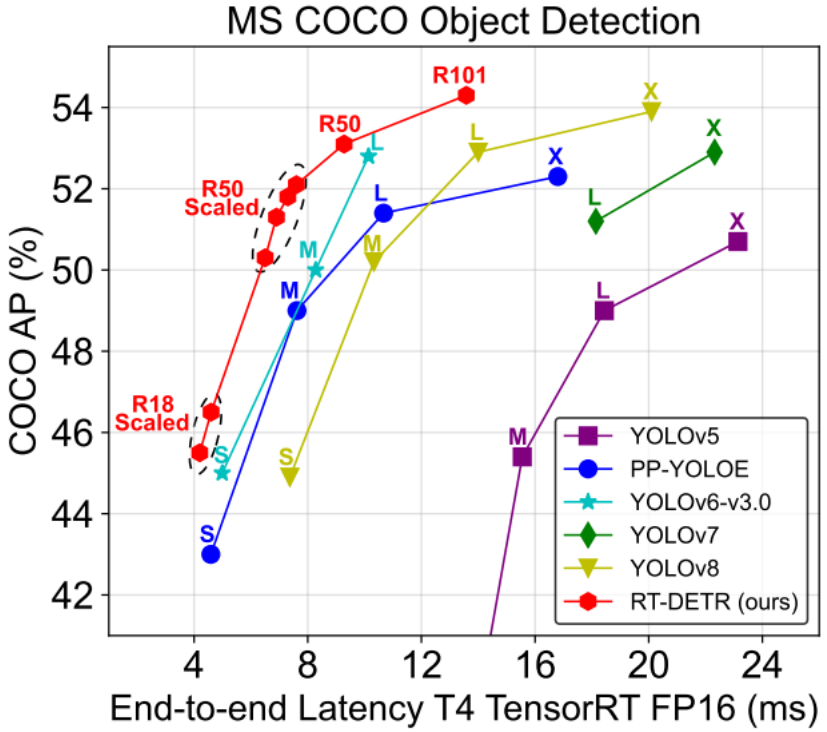
[Submitted on 17 Apr 2023 (v1), last revised 6 Jul 2023 (this version, v2)]

DETRs Beat YOLOs on Real-time Object Detection

Wenyu Lv, Yian Zhao, Shangliang Xu, Jinman Wei, Guanzhong Wang, Cheng Cui, Yuning Du, Qingqing Dang, Yi Liu

Recently, end-to-end transformer-based detectors~(DETRs) have achieved remarkable performance. However, the issue of the high computational cost of DETRs has not been effectively addressed, limiting their practical application and preventing them from fully exploiting the benefits of no post-processing, such as non-maximum suppression (NMS). In this paper, we first analyze the influence of NMS in modern real-time object detectors on inference speed, and establish an end-to-end speed benchmark. To avoid the inference delay caused by NMS, we propose a Real-Time DEtection TRansformer (RT-DETR), the first real-time end-to-end object detector to our best knowledge. Specifically, we design an efficient hybrid encoder to efficiently process multi-scale features by decoupling the intra-scale interaction and cross-scale fusion, and propose IoU-aware query selection to improve the initialization of object queries. In addition, our proposed detector supports flexibly adjustment of the inference speed by using different decoder layers without the need for retraining, which facilitates the practical application of real-time obj... P and 74 FPS, out... P and 108 FPS, out... dels are ava...





Why Study YOLO?

YOLO is an abbreviation for the term 'You Only Look Once'. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

Speed: This algorithm improves the speed of detection because it can predict objects in real-time.

High accuracy: YOLO is a predictive technique that provides accurate results with minimal background errors.

Learning capabilities: The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

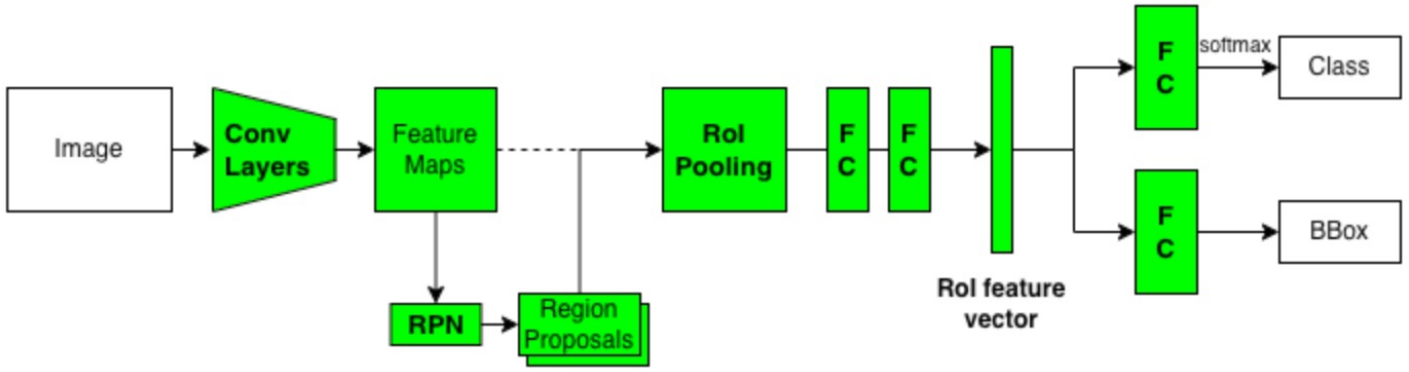
For small datasets and limited computational power, YOLOv8 might be a better choice as it has been optimized for speed and accuracy. However, if you have more significant datasets and more complex object detection tasks, DETR could be a better fit due to its ability to handle object detection without pre-defined anchor boxes

Outline

- Object Detection Milestones: Transformer
- End-to-End Object Detection with Transformers
- Object Detection Milestones: YOLOs variants
- YOLO Motivations for OD
- YOLO-v1
- YOLO-v2

YOLO: Motivations

In 2015, Joseph Redmon (University of Washington) developed YOLO. One of his co-authors, Ross Girshick (Microsoft Research), published a paper for Faster R-CNN around the same time. They probably shared common ideas in computer vision research as there are some similarities between YOLO and Faster R-CNN. For example, both models apply convolutional layers on input images to generate feature maps. However, Faster R-CNN uses a two-stage object detection pipeline, while YOLO has no separate region proposal step and is much faster than Faster R-CNN.



Faster R-CNN pipeline

YOLO pipeline

YOLO: Motivations

In 2015, Joseph Redmon (University of Washington) developed YOLO. One of his co-authors, Ross Girshick (Microsoft Research), published a paper for Faster R-CNN around the same time. They probably shared common ideas in computer vision research as there are some similarities between YOLO and Faster R-CNN. For example, both models apply convolutional layers on input images to generate feature maps. However, Faster R-CNN uses a two-stage object detection pipeline, while YOLO has no separate region proposal step and is much faster than Faster R-CNN.

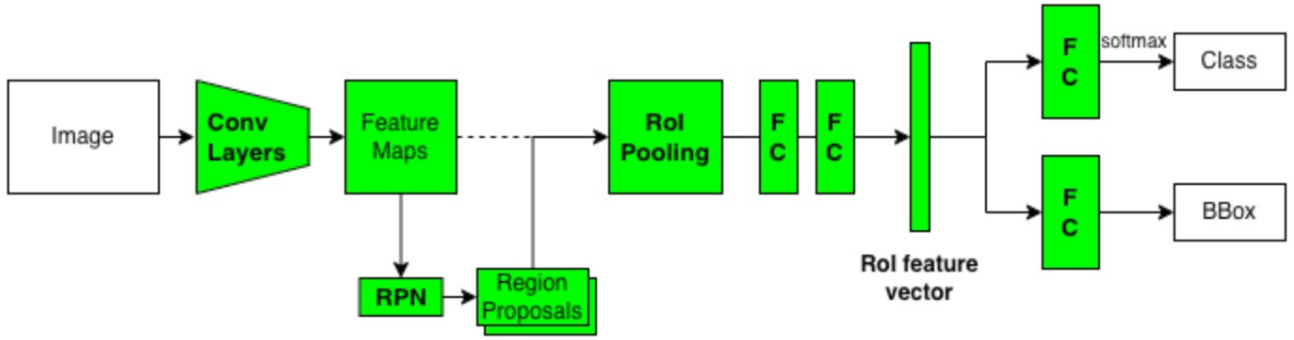
YOLO has many versions (variants). Joseph Redmon developed the first three versions of YOLO: YOLOv1, v2, and v3. Then, he quit.

After YOLOv3, different groups of people developed their versions of YOLO:

- YOLOv4 by Alexey Bochkovskiy, et al.
- YOLOv5 by Ultralytics
- YOLOv6 by Meituan
- YOLOX by Zheng Ge et al.
- YOLOv7 by Chien-Yao Wang et al. (The same people from YOLOv4)

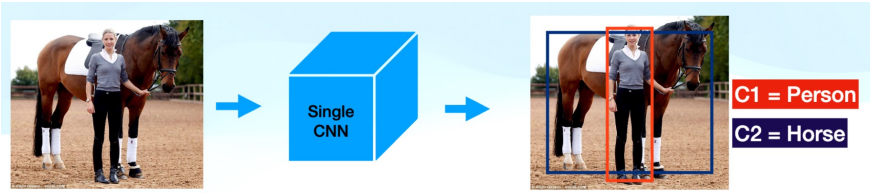
YOLO: Motivations

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)



Faster R-CNN pipeline

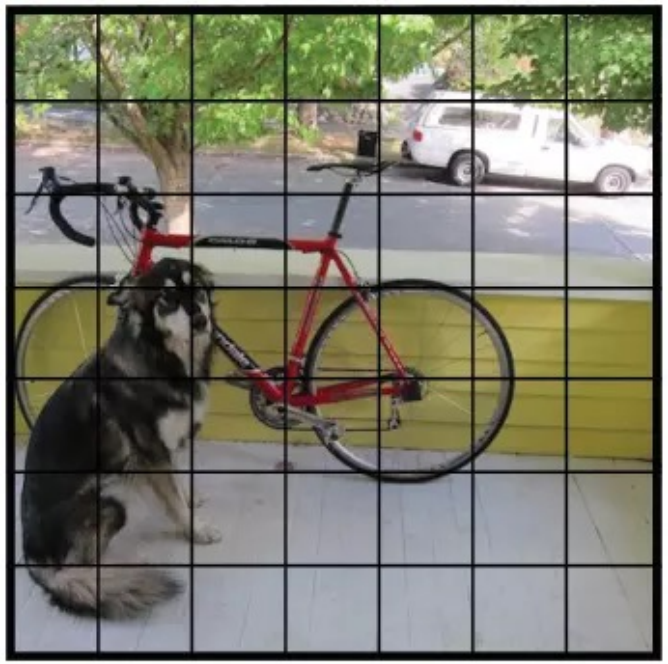
YOLO pipeline



YOLO: Motivations

Residual blocks

First, the image is divided into various grids. Each grid has a dimension of S x S. The following image shows how an input image is divided into grids.



S × S grid on input

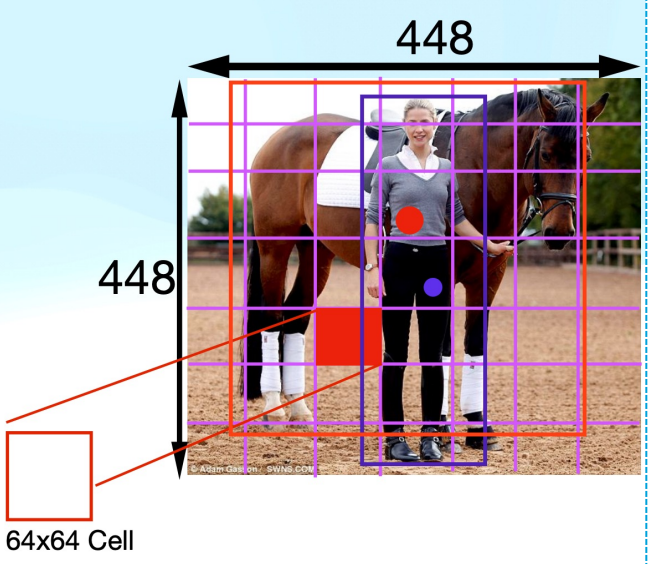
YOLO: Motivations

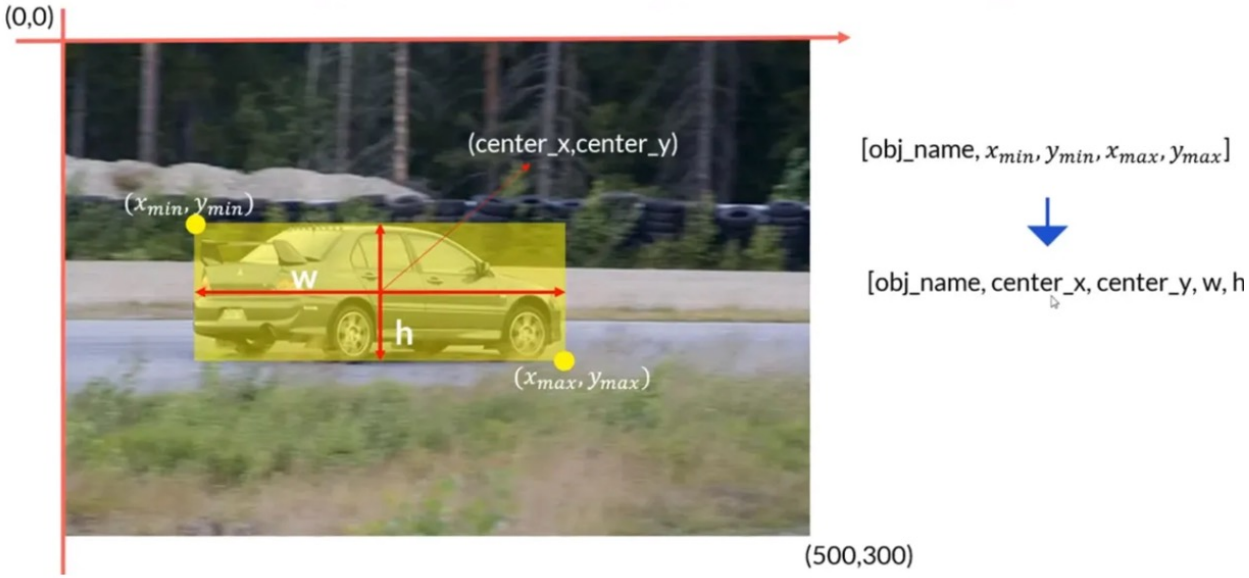
Residual blocks

First, the image is divided into various grids. Each grid has a dimension of S x S. The following image shows how an input image is divided into grids.

Steps in YOLO

- Take input image
- Resize to 448x448
- Divide into SxS Grid cells
- S=7 in paper
- Each cell is responsible for predicting one object
- Which cell is responsible?
- Where Center of object falls into

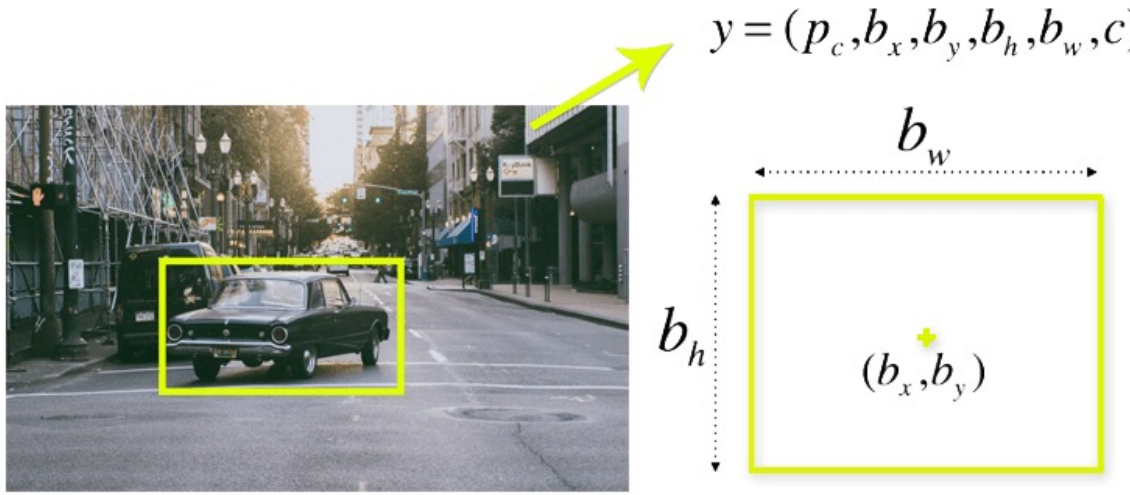




YOLO: Motivations

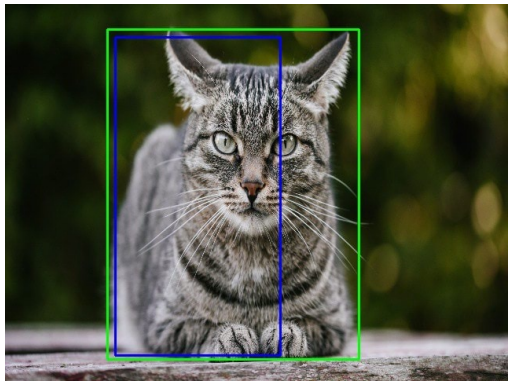
Bounding box regression

Every bounding box in the image consists of the following attributes: Width (bw), Height (bh), Class (for example, person, car, traffic light, etc.); This is represented by the letter c. Bounding box center (bx,by)

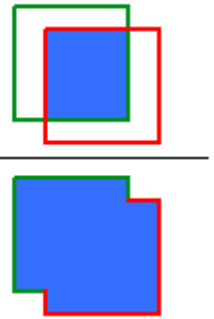


IoU

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.



$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$



YOLO: Motivations

- [x] Bounding box regression

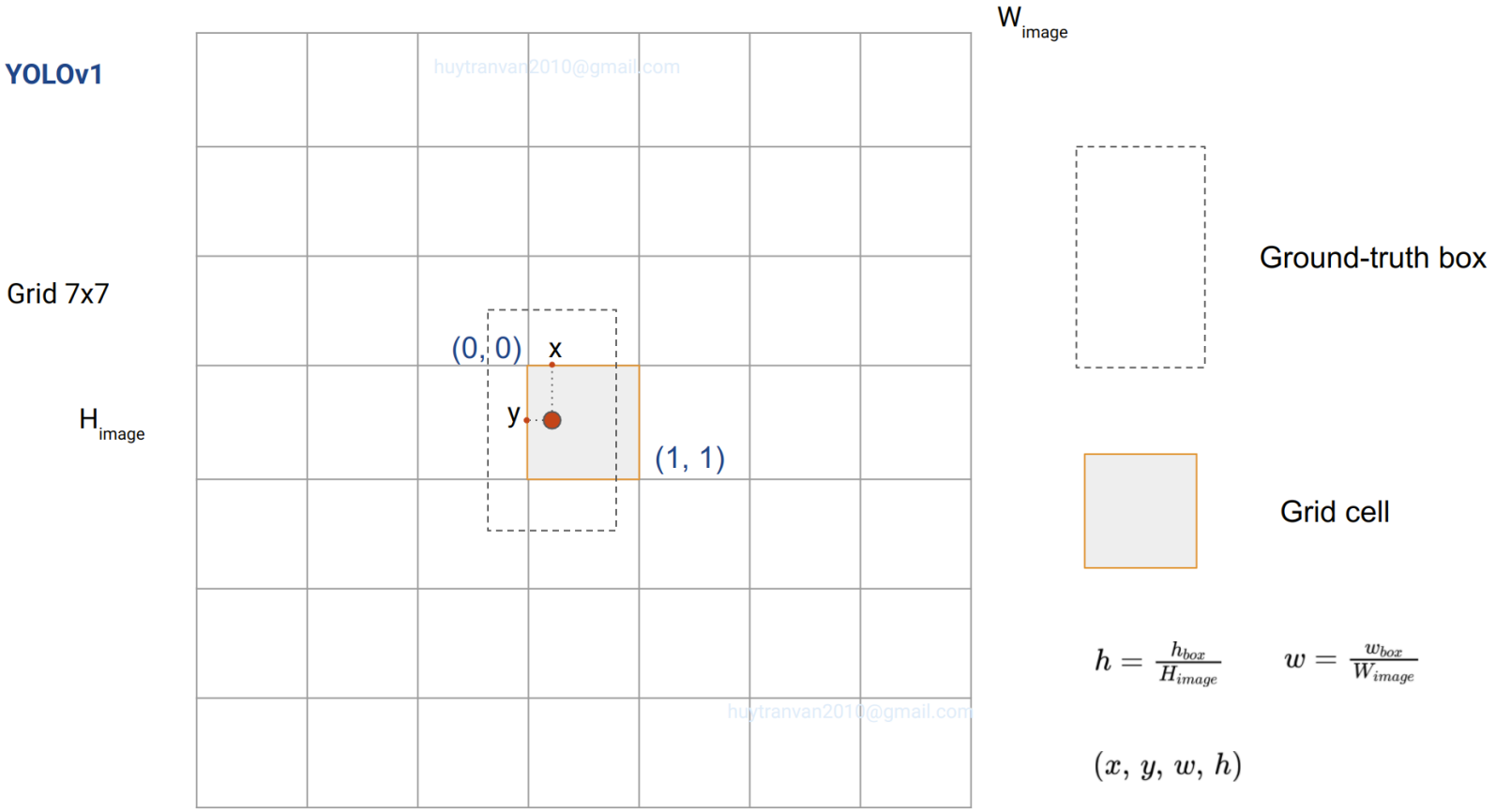


Image credits: https://huytranvan2010.github.io/YOLOv1-Core-Ideas/

YOLO: Motivations

Bounding boxes

- Center point (x,y): Relative to anchor that (x,y) falls into.

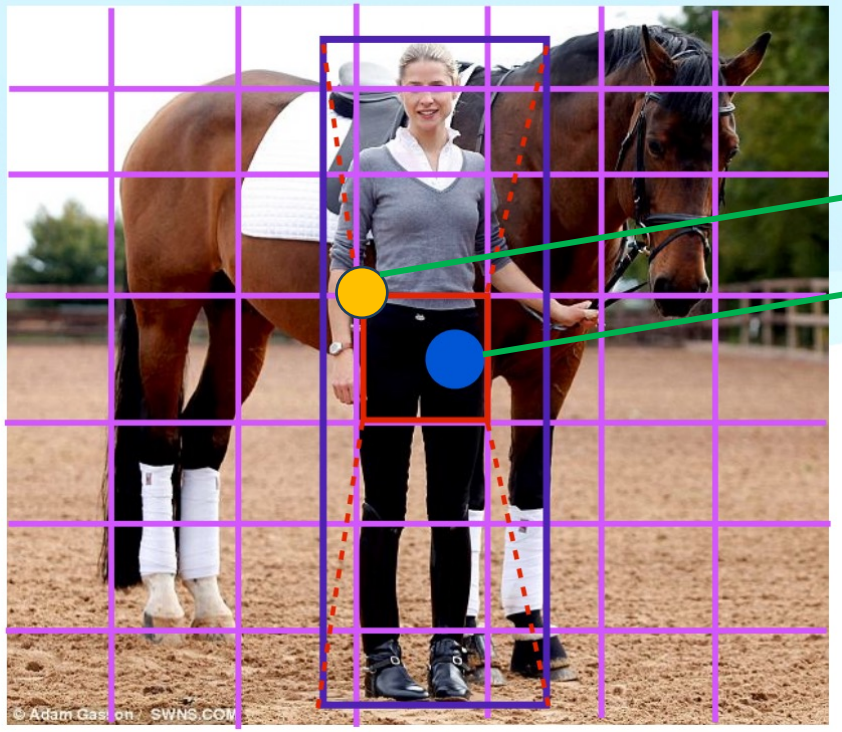
$$\Delta x = (x - x_a)/64$$
$$\Delta y = (y - y_a)/64$$

(x_a, y_a): the coordinate of left-top point

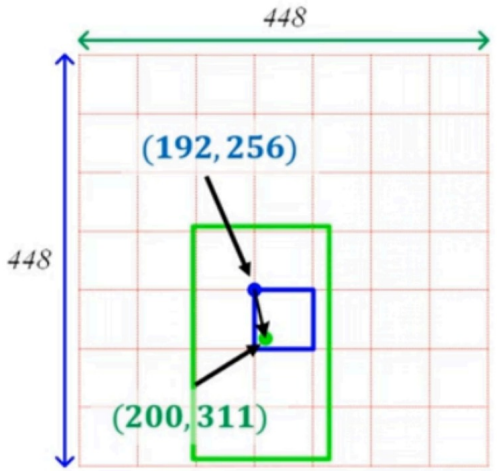
- Width/height (w,h): relative to the whole image

$$\Delta w = w/448$$
$$\Delta h = h/448$$

$(200, 311, 142, 250)$



YOLO: Motivations

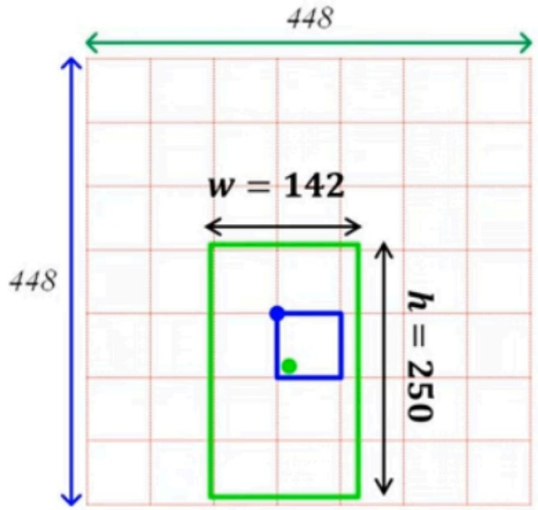


$$\Delta x = \frac{200 - 192}{64} \approx 0.13$$

$$\Delta y = \frac{311 - 256}{64} \approx 0.87$$

$$\Delta w = w/448$$

$$\Delta h = h/448$$



$$\Delta x = \frac{200 - 192}{64} \approx 0.13$$

$$\Delta y = \frac{311 - 256}{64} \approx 0.87$$

$$\Delta w = \frac{142}{448} \approx 0.31$$

$$\Delta h = \frac{250}{448} \approx 0.56$$

$$(200, 311, 142, 250) \Rightarrow (0.13,\ 0.87,\ 0.31,\ 0.56)$$

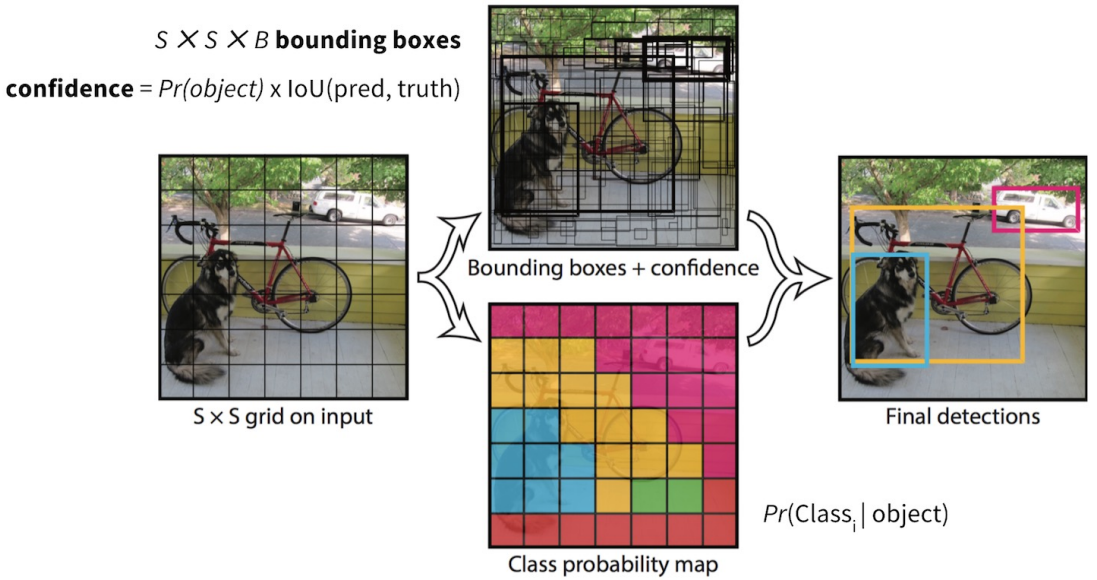
YOLO: Label Encoding

- For every Grid cell (Anchor box), we need to create targets/labels
- No Object - All zeros
- Object - Relative values w.r.t grid
- Classes - one-hot encoding
- Ex: (x,y,w,h,c) = (0.9,0.7,0.1,0.1,1.0)
  - Classes = (1.0, 0,0,…0) - 20 values

	$(\Delta\hat{x},\ \Delta\hat{y},\ \Delta\hat{w},\ \Delta\hat{h},\hat{c})$	$(\hat{p}_1,\ \hat{p}_2,\ \dots,\ \hat{p}_{20})$
A_1	(0 0 0 0 0)	(0 0 … 0)
A_2	(0 0 0 0 0)	(0 0 … 0)
	… … … … .	
A_{11}	(0.9 0.7 0.1 0.1 1.0)	(0 … 1.0 …)
	… … … … .	$\hat{p}_{14=\text{person}}$
A_{32}	(0.1 0.8 0.3 0.5 1.0)	(0 … 1.0 …)
	… … … … .	

YOLO: Motivations

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)



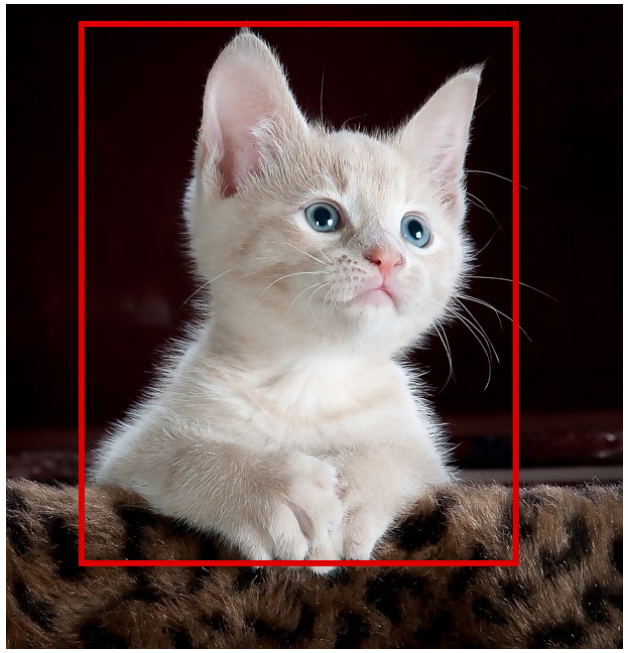
YOLO: Motivations

Image Classification

Is this a cat or a dog?

Neural Network Output:
Cat = 1
Dog = 0

Object Localization



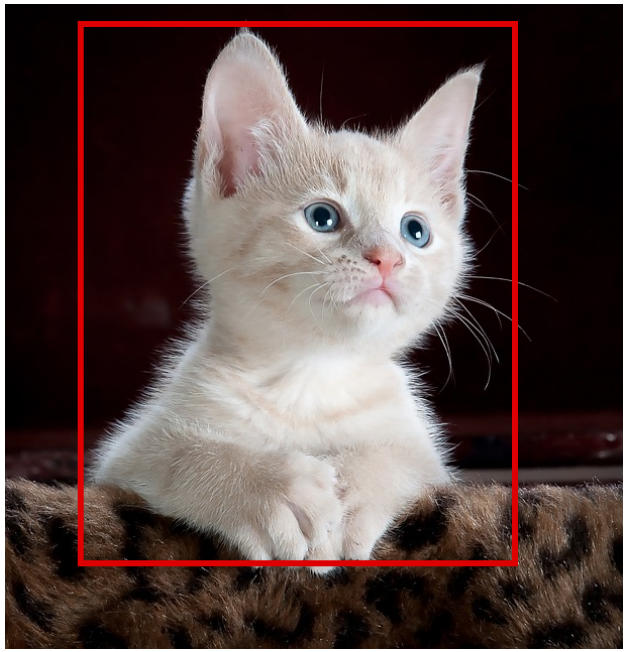
Where exactly is the cat in the image?

Neural Network Output:
Cat = 1
Dog = 0

Bounding box

YOLO: Motivations

Object Localization



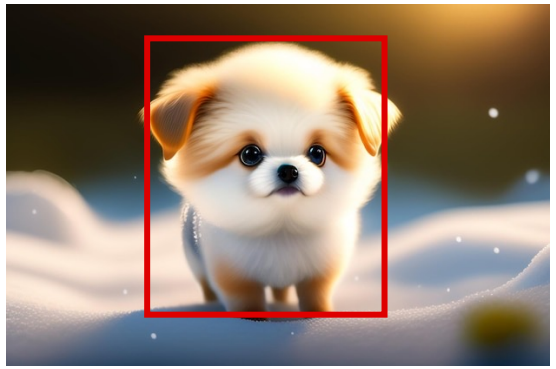
Where exactly is the cat in the image?

$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 50 \\ 70 \\ 60 \\ 70 \\ 1 \\ 0 \end{bmatrix}$$

C_1: Cat class
C_2: Dog class

Neural Network Output:
Cat = 1
Dog = 0

Bounding box

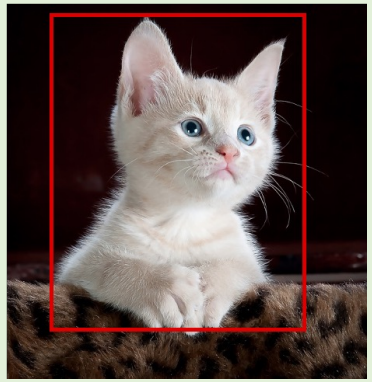


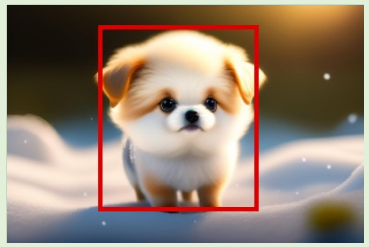
$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 30 \\ 28 \\ 28 \\ 82 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

YOLO: Motivations

X: Train



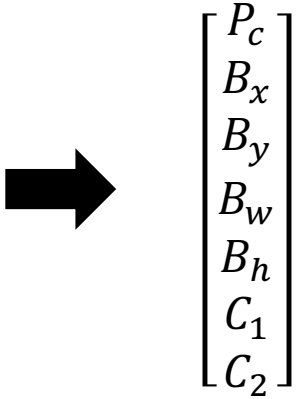


Y: Train

$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \begin{bmatrix} 1 \\ 50 \\ 70 \\ 60 \\ 70 \\ 1 \\ 0 \end{bmatrix}$$

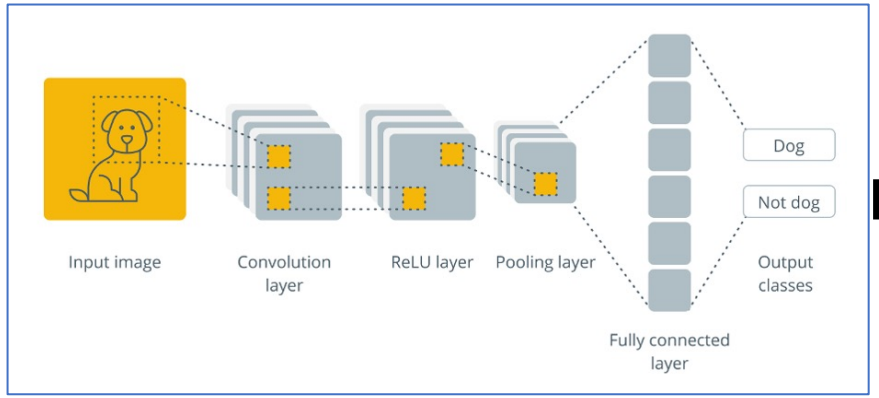
$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \begin{bmatrix} 1 \\ 30 \\ 28 \\ 28 \\ 82 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

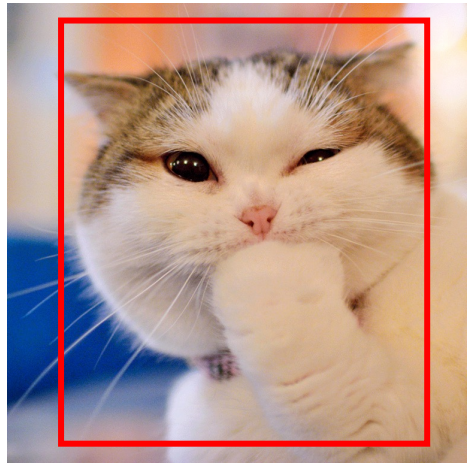


$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix}$$

YOLO: Motivations



$$\begin{bmatrix} 1 \\ 50 \\ 70 \\ 60 \\ 70 \\ 1 \\ 0 \end{bmatrix}$$

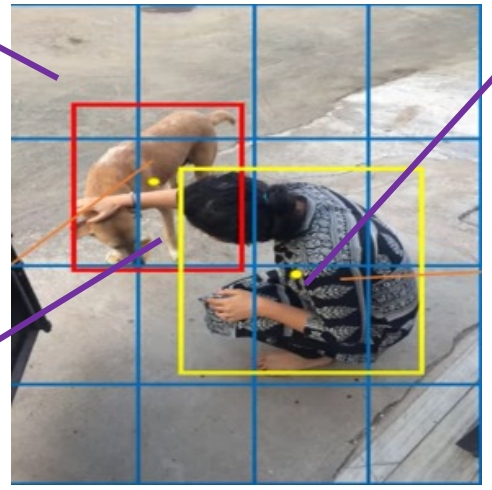


This works OK only for single object. What about multiple objects in an image

YOLO: Motivations

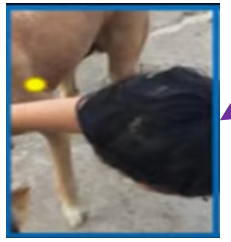
$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 0.32 \\ 0.02 \\ 3.0 \\ 2.0 \\ 0 \\ 1 \end{bmatrix}$$



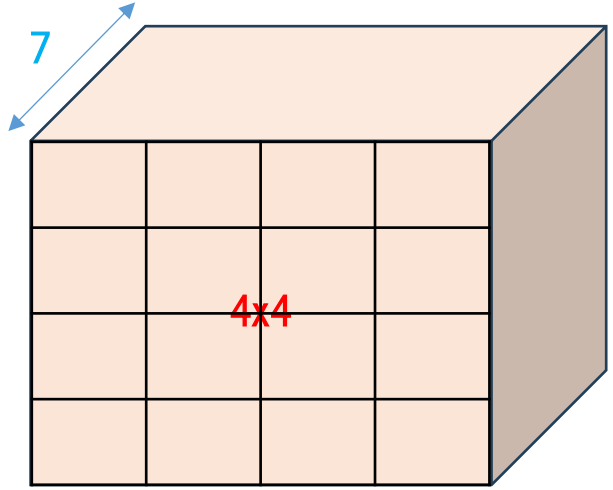
$$\begin{bmatrix} 1 \\ 0.05 \\ 0.3 \\ 2.0 \\ 1.3 \\ 1 \\ 0 \end{bmatrix}$$

(0,0)



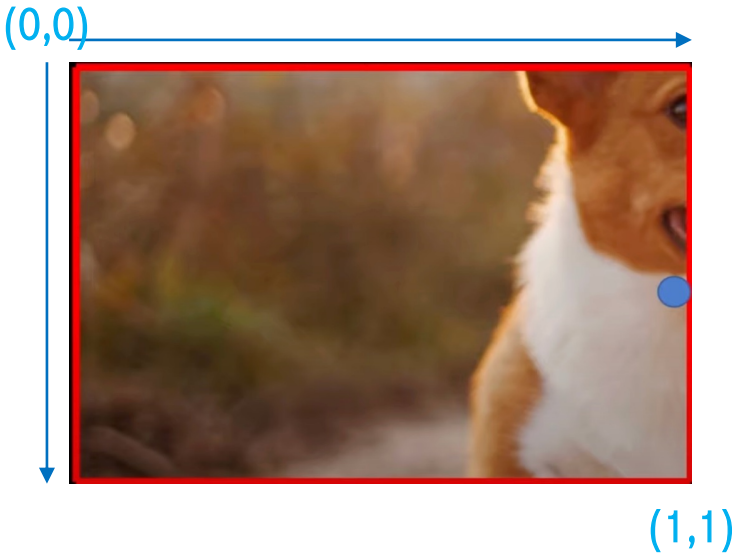
(1,1)

C_1: Dog class
C_2: Human class



YOLO: Motivations

How the labels look like?



Each output and label will be relative to the cell!
Each bounding box for each cell will have:
[x,y,w,h] = [0.95, 0.55, 0.5, 1.5]

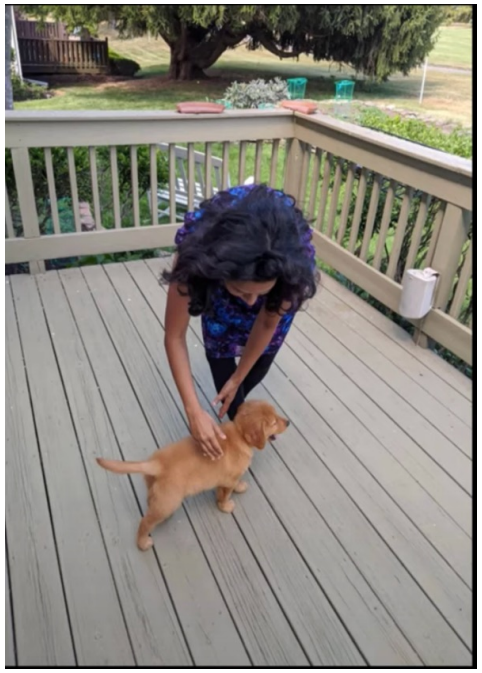
Bounding Box

$$\text{Label}_{\text{cell}} = [c_1, c_2, \ldots, c_{20}, p_c, x, y, w, h,]$$

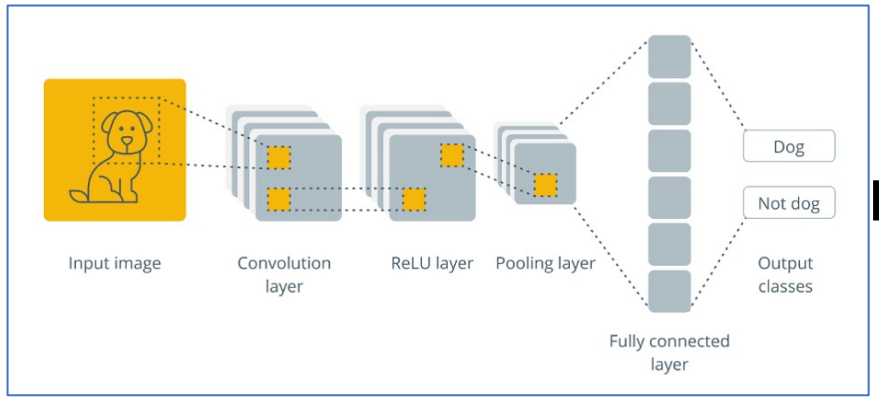
20 classes

Probability that there is an object (1 or 0)

YOLO: Motivations

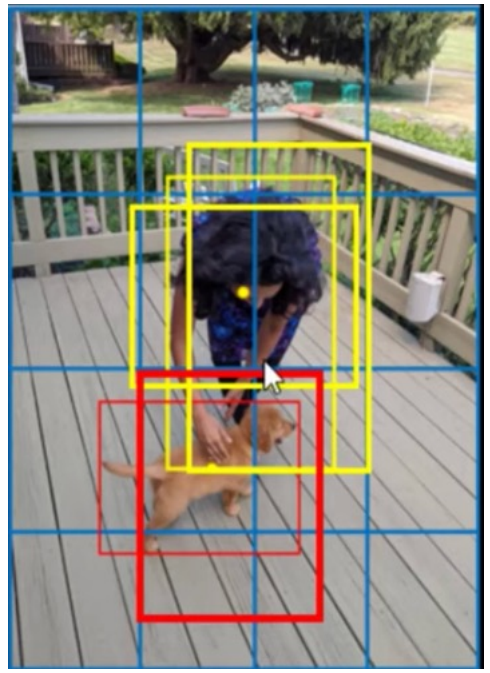


How the prediction look like?



$$\begin{bmatrix} P_c \\ B_x \\ B_y \\ B_w \\ B_h \\ C_1 \\ C_2 \end{bmatrix}$$

16 vectors



Multiple bounding boxes

Bounding Box

$$\text{Prediction}_{\text{cell}} = [c_1, c_2, \ldots, c_{20}, p_c, x, y, w, h]$$

20 classes

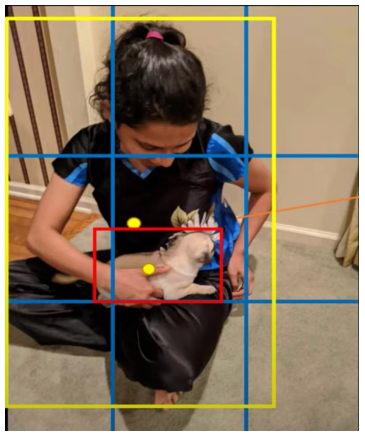
Probability that there is an object (1 or 0)

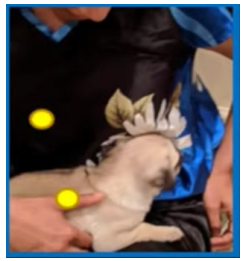
YOLO: Motivations

Non Maximum Suppression

Algorithm 1 Non-Max Suppression

procedure NMS(B,c)
  $B_{nms} \leftarrow \emptyset$ Initialize empty set
  for $b_i \in B$ **do** ⇒ Iterate over all the boxes
    $discard \leftarrow$ False Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
    for $b_j \in B$ **do** Start another loop to compare with b(i)
      if same$(b_i, b_j) > \boldsymbol{\lambda_{nms}}$ **then** If both boxes having same IOU
        if score$(c, b_j) >$ score(c, b_i) **then**
          $discard \leftarrow$ True Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
    if not $discard$ **then** Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So add it to the final list.
      $B_{nms} \leftarrow B_{nms} \cup b_i$
  return B_{nms} Do the same procedure for remaining boxes and return the final list



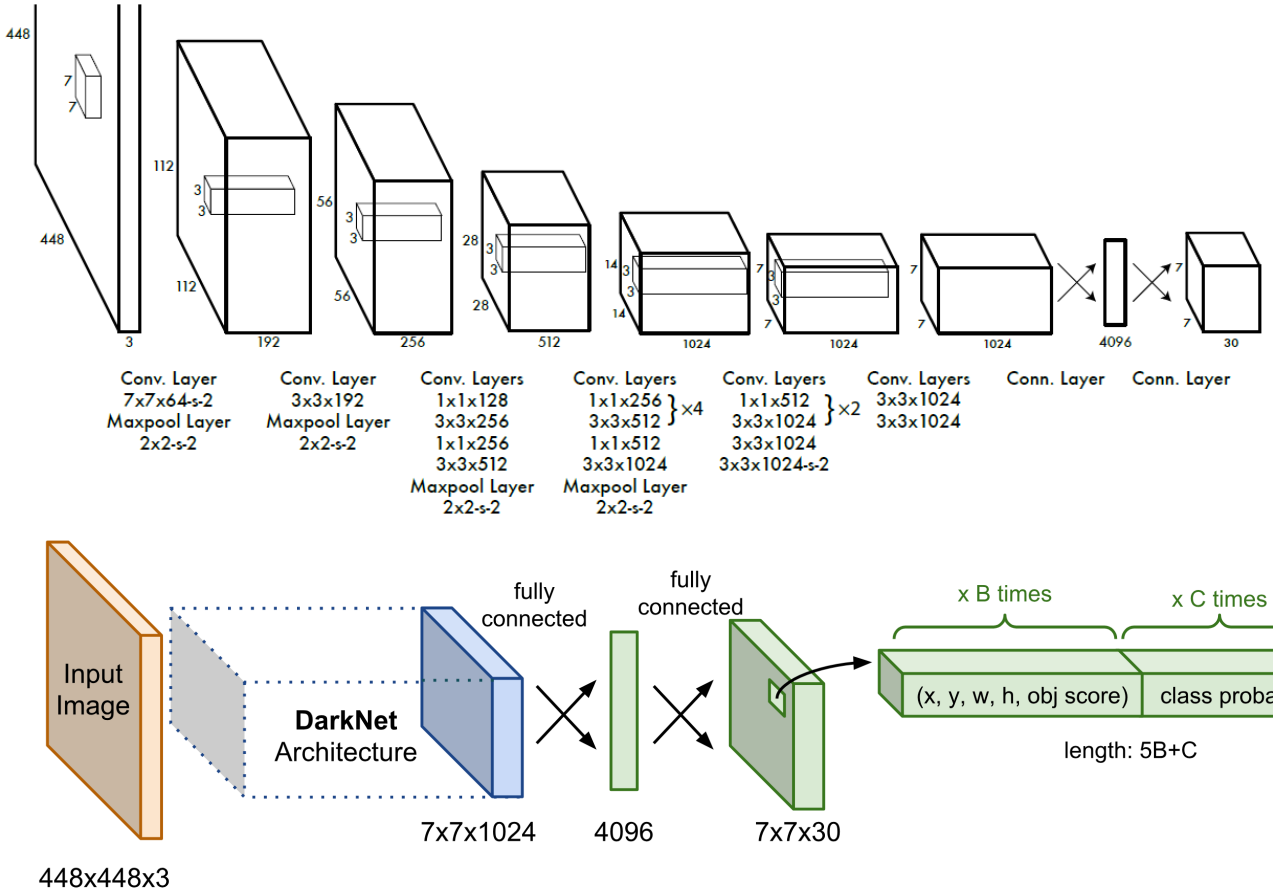


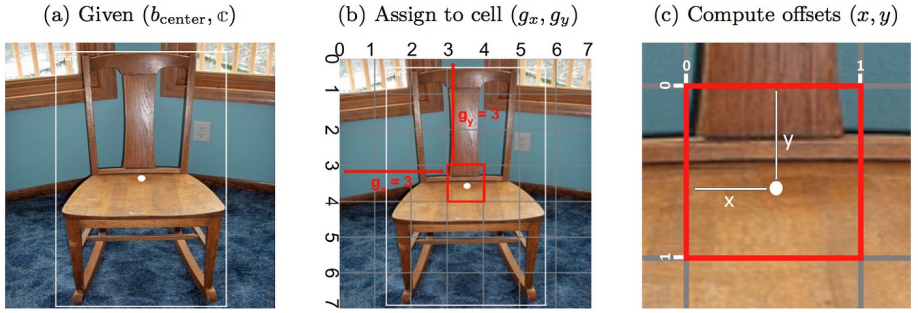
What if one grid cell has center of two objects

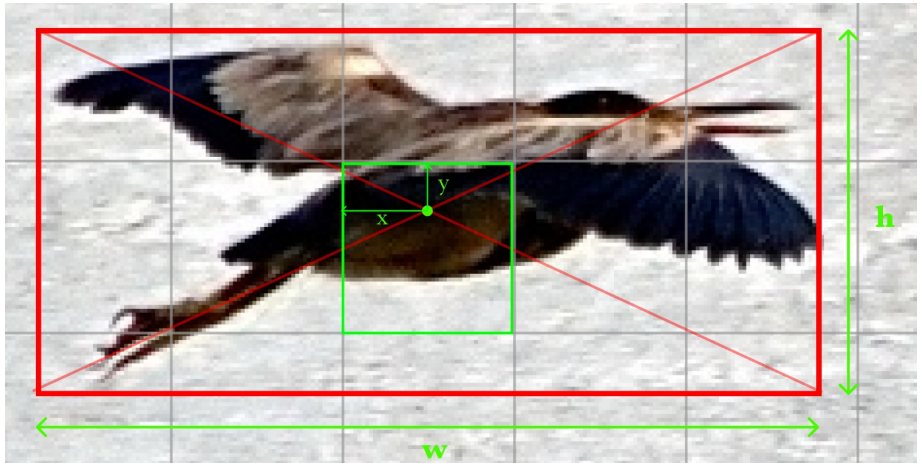
Outline

- Object Detection Milestones: Transformer
- End-to-End Object Detection with Transformers
- Object Detection Milestones: YOLOs variants
- YOLO Motivations for OD
- YOLO-v1
- YOLO-v2

YOLO-v1 Architecture



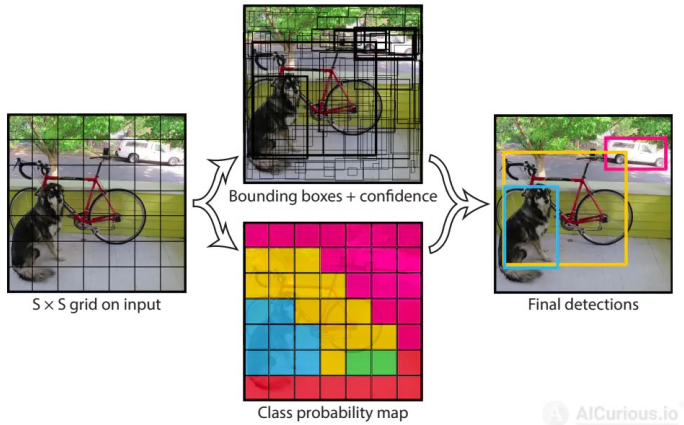


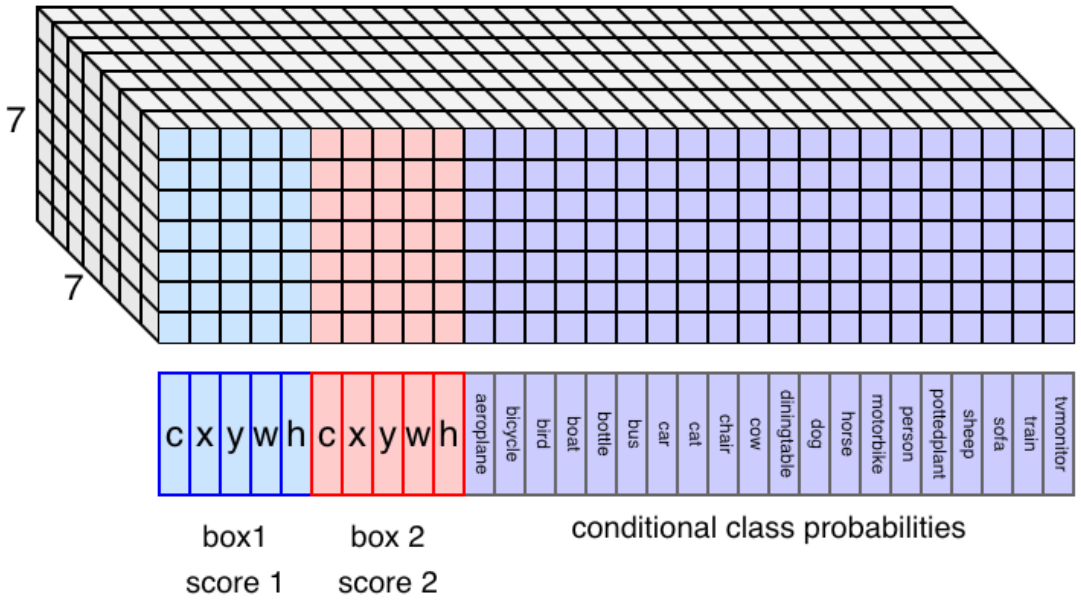


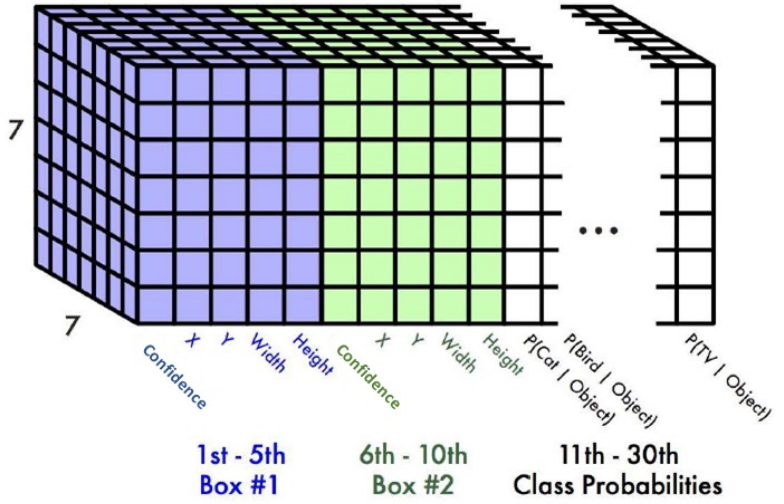
The YOLO is a network was "*inspired by*" GoogleNet. It has 24 convolutional layers working for feature extractors and 2 dense layers for doing the predictions. This architecture works upon is called Darknet. There is a fast version of YOLO called "Tiny-YOLO" which only has 9 convolution layers

YOLO-v1 Architecture

- The input image is divided into an S×S grid (S=7)
- Each grid cell predicts B bounding boxes (B=2) and confidence scores for those boxes
- Each bounding box consists of 5 predictions: x, y, w, h, and confidence
- Each grid cell also predicts conditional class probabilities, P(Class i | Object). (Total number of classes=20)







The output size : 7×7×(2×5+20)=1470

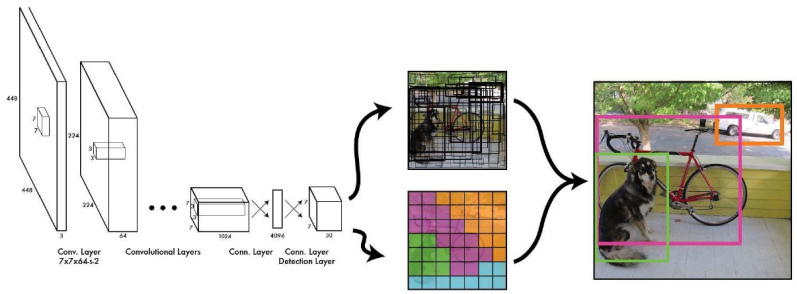
YOLO-v1 Architecture

Loss Function

Regression loss

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

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



Confidence loss

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

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

Classification loss

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

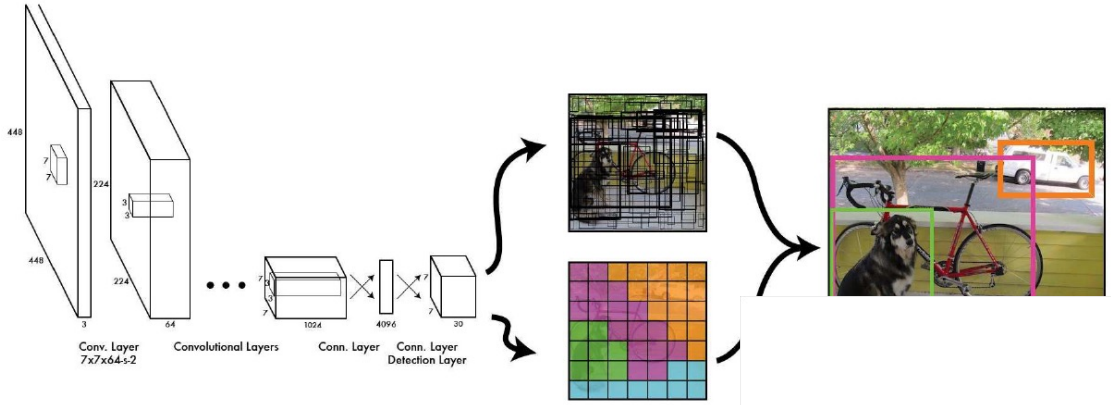
$$\mathcal{L} = \mathcal{L}_{\text{loc}} + \mathcal{L}_{\text{obj}} + \mathcal{L}_{\text{cls}}$$

$\mathcal{L}_{\text{loc}}$ - loss function cho dự đoán vị trí bounding box so với ground truth.

$\mathcal{L}_{\text{obj}}$ - loss function cho dự đoán trong cell có object hay không.

$\mathcal{L}_{\text{cls}}$ - loss function cho dự đoán phân phối xác suất cho từng class.

YOLO-v1 Architecture



$$\text{box confidence score} = Pr(object) \cdot IOU_{pred}^{truth}$$

$$\text{conditional class probabilities} = Pr(class_i|object)$$

$$\text{class confidence score} = Pr(class_i) \cdot IOU_{pred}^{truth}$$

Loss Function

Localization loss

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

Bounding box location (x, y) when there is object

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

Bounding box size (w, h) when there is object

Confidence loss

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

Confidence when there is object

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

Confidence when there is no object (in the box j of the cell i)

Classification loss

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

Class probabilities when there is object (in the cell i)

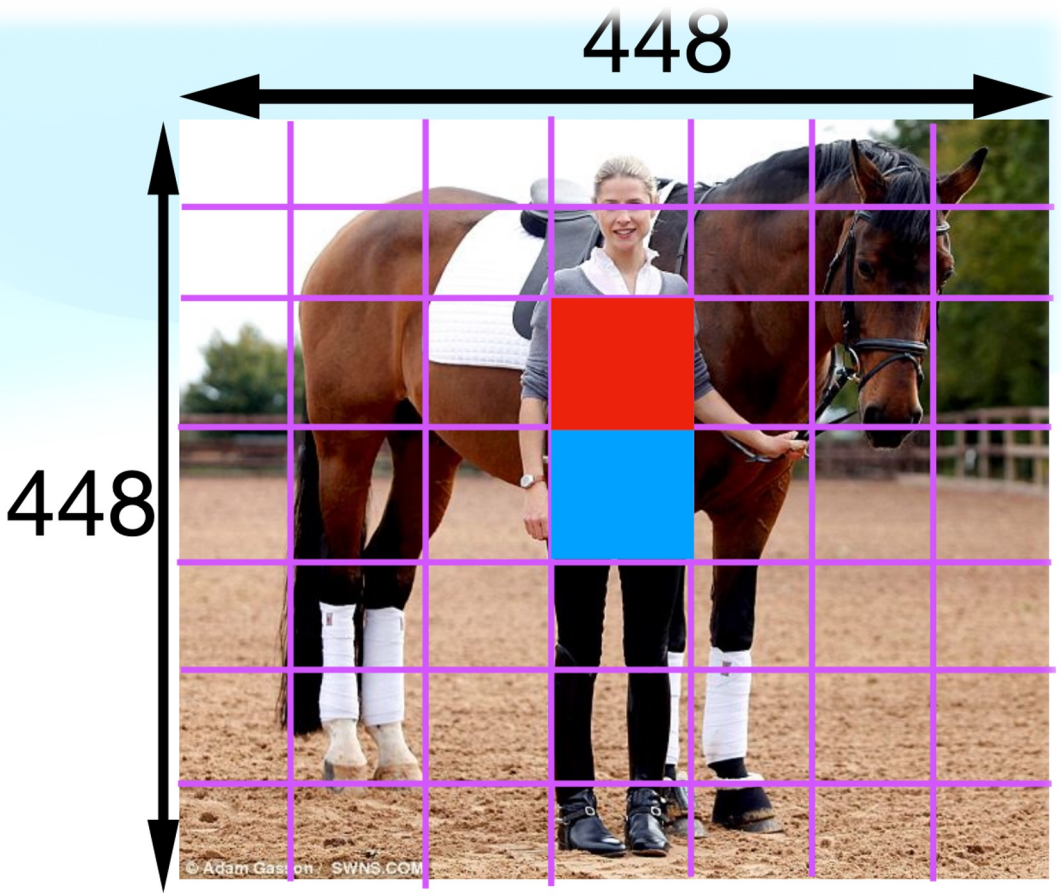
1 when there is object, 0 when there is no object (in the box j of the cell i)

1 when there is no object, 0 when there is object

Yolo-v1: Loss function

- Loss L is the sum of losses over all grid cells SxS.
- Put more importance on grid cells that contain objects
- Decrease the importance of grid cells having no objects
- Ex: 2 object cells, 47 no-object cells

$$L = \sum_{i=1}^{S^2} L_i$$



Yolo-v1: Loss function

- Loss L is the sum of losses over all grid cells SxS.
- Put more importance on grid cells that contain objects
- Decrease the importance of grid cells having no objects

$$L = \sum_{i=1}^{S^2} 1_i^{obj} \times L_{i,obj} + \lambda_{no\_obj} \sum_{i=1}^{S^2} 1_i^{no\_obj} \times L_{i,no\_obj}$$

$1_i^{obj} = 1$ if i^{th} grid is **object anchor**

$1_i^{no\_obj} = 1$ if i^{th} grid is **no-object anchor**

Reduce the importance of no object (in the paper they set to 0.5)

Yolo-v1: Loss function

Loss for object cells

- Loss = Objectness loss + classification loss + Box Regression loss
- Put more weightage on box parameters

$$L_{i,obj} = \lambda_{coord} \times L_{i,obj}^{box} + L_{i,obj}^{conf} + L_{i,obj}^{cls}$$

$$= 5$$

Yolo-v1: Loss function

Bounding box loss

- Sum of squared errors on predicted box parameters and ground truth labels

Why? Weighting factor

$$L_{i,obj}^{box} = \left(\Delta x_i^* - \Delta\hat{x}_i\right)^2 + \left(\Delta y_i^* - \Delta\hat{y}_i\right)^2 + \left(\sqrt{\Delta w_i^*} - \sqrt{\Delta\hat{w}_i}\right)^2 + \left(\sqrt{\Delta h_i^*} - \sqrt{\Delta h_i}\right)^2$$

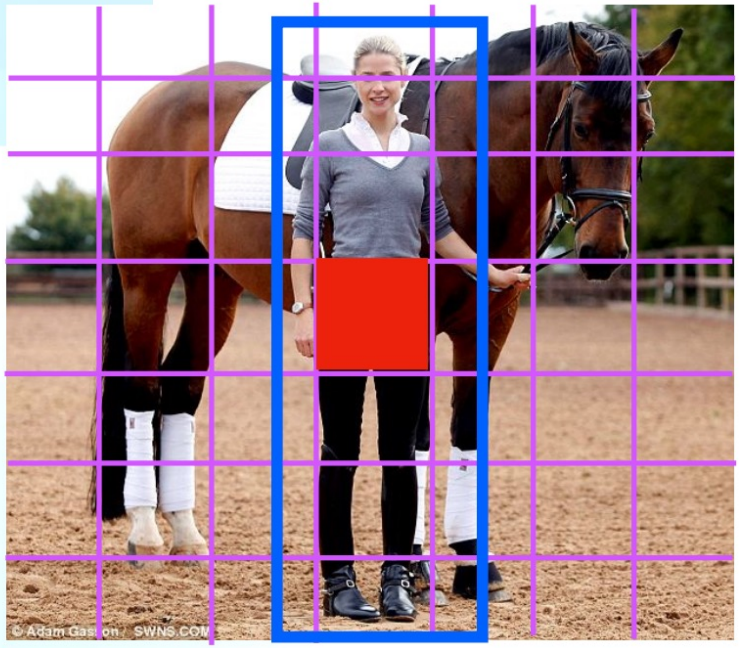
- $(\Delta\hat{x}_i, \Delta\hat{y}_i, \Delta\hat{w}_i, \hat{h}_i)$: ground-truth box
- $(\Delta x_i^*, \Delta y_i^*, \Delta w_i^*, \Delta h_i^*)$: **responsible** predicted box that has the largest IoU with ground-truth box

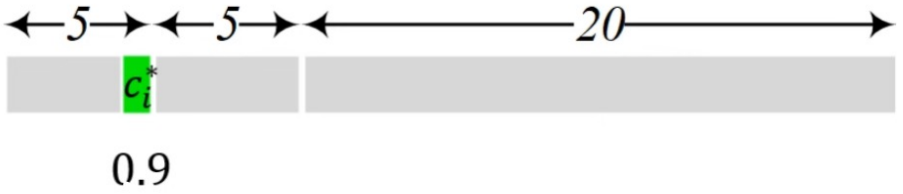
Yolo-v1: Loss function

Objectness Confidence Loss

- Squared error between the predicted confidence and encoded label confidence

$$L_{i,obj}^{conf} = (c_i^* - \hat{c}_i)^2 = (0.9 - 1.0)^2$$

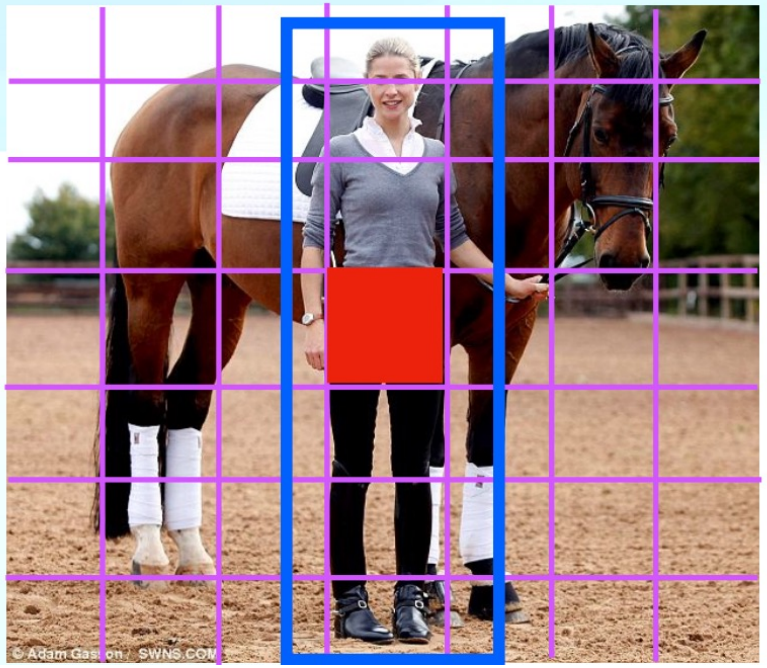




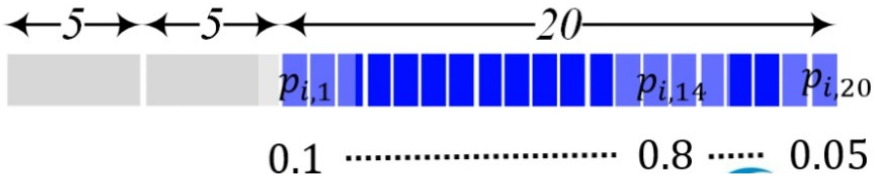
Yolo-v1: Loss function

Classification Loss

- Sum of squared errors over all class probabilities



$$L_{i,obj}^{cls} = (0.1 - 0.0)^2 + \cdots + (0.8 - \hat{p}_{i,14}(1.0))^2 + \cdots + (0.05 - 0.0)^2$$



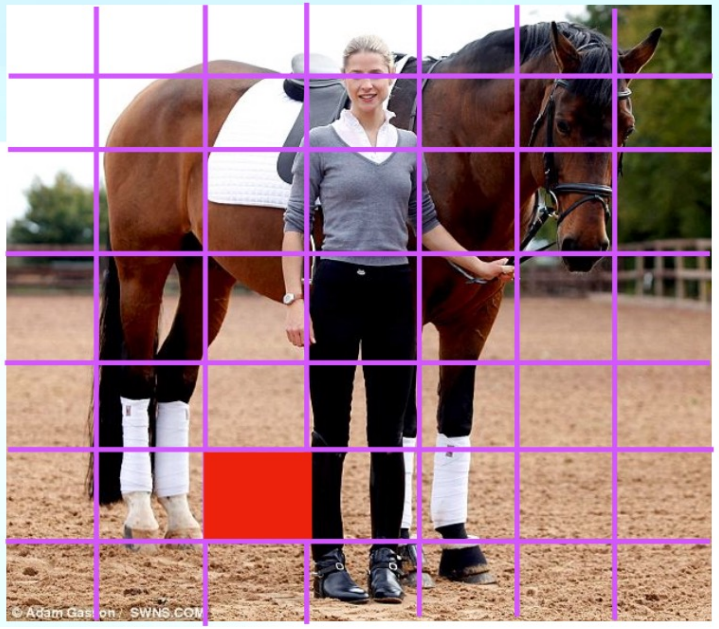
$$\hat{p}_{i,14=\text{person}} = 1.0$$

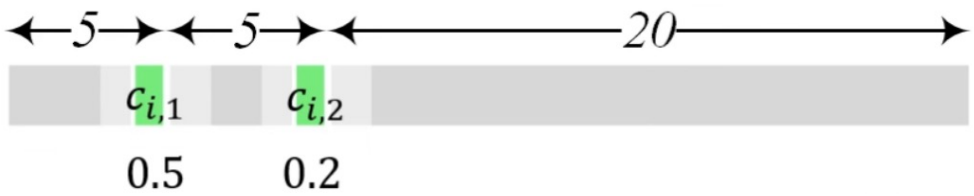
Yolo-v1: Loss function

Loss for no-object cells

Only calculates the objectness confidence score loss

$$L_{i,no\_obj} = (0.5 - \hat{c}_{i,1})^2 + (0.2 - \hat{c}_{i,2})^2 = 0.29$$

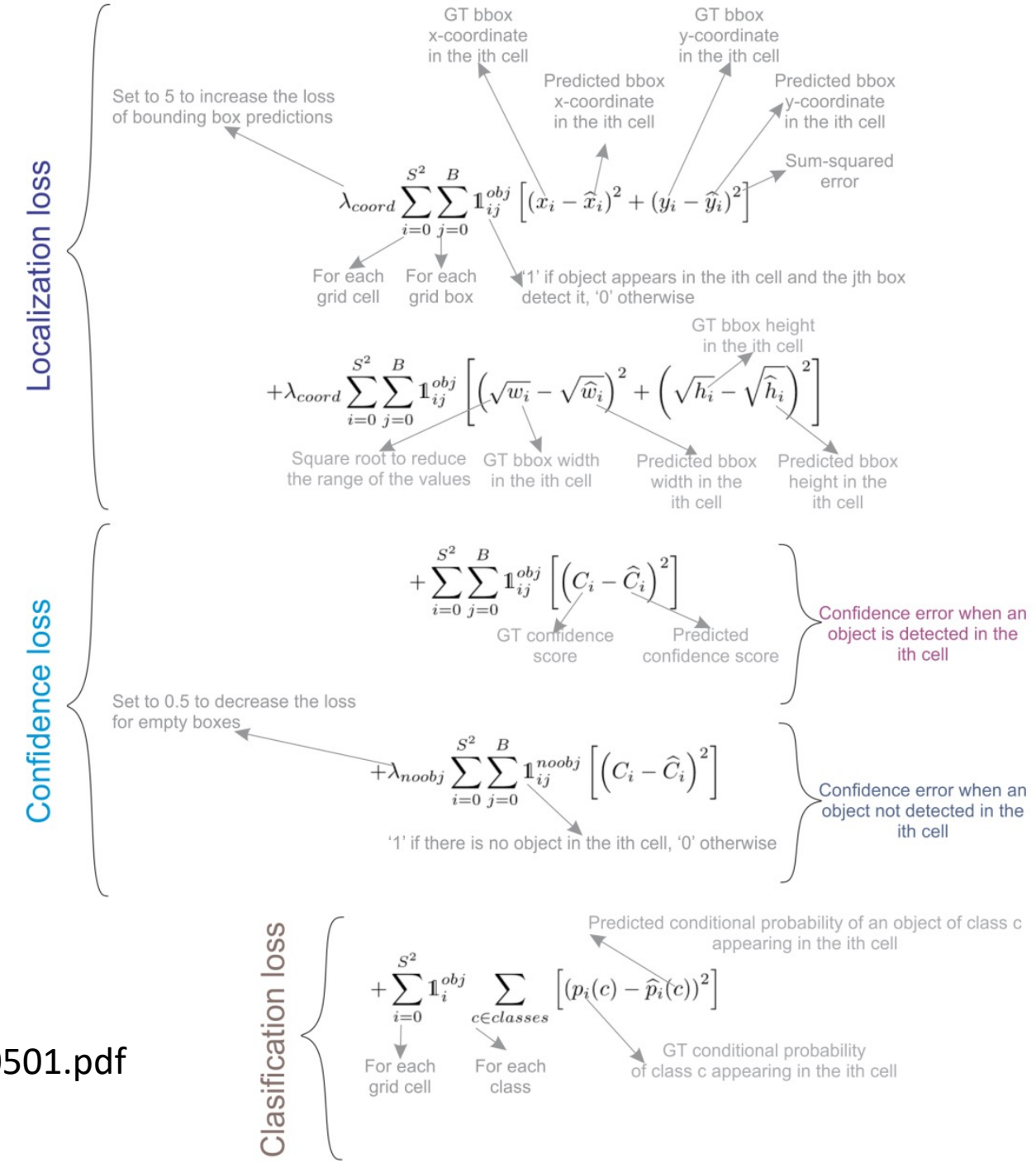




Why don't need to calculate bounding box loss and classfication loss

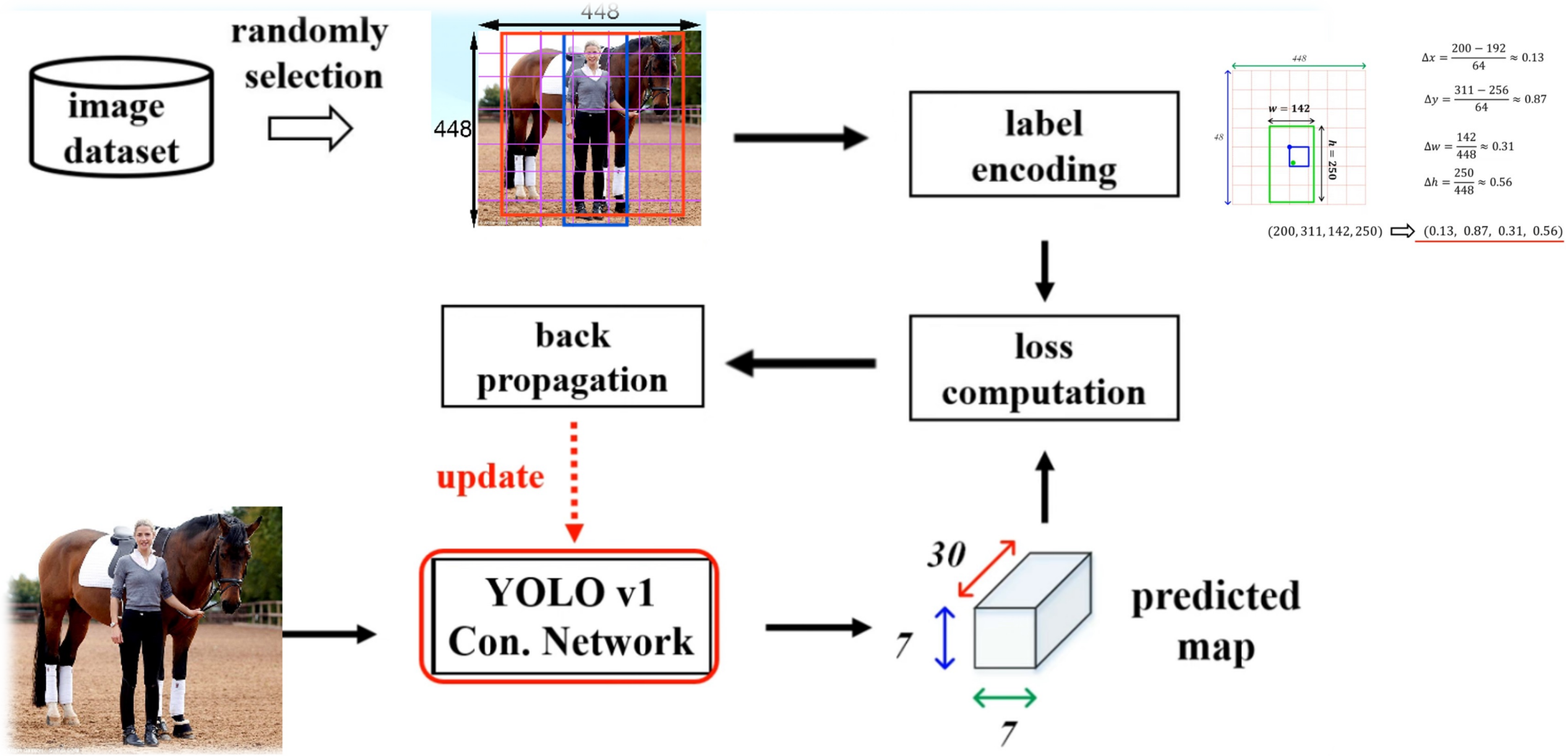
Yolo-v1: Loss function

$$
\begin{aligned}
\boldsymbol{L} = \lambda_{coord} \times \sum_{i=1}^{S^2} 1_i^{obj} \times \begin{pmatrix} (\Delta x_i^* - \Delta \hat{x}_i)^2 + (\Delta y_i^* - \Delta \hat{y}_i)^2 + \\ \left(\sqrt{\Delta w_i^*} - \sqrt{\Delta \hat{w}_i}\right)^2 + \left(\sqrt{\Delta h_i^*} - \sqrt{\Delta h_i}\right)^2 \end{pmatrix} \\
+ \sum_{i=1}^{S^2} 1_i^{obj} \times (c_i^* - \hat{c}_i)^2 + \sum_{i=1}^{S^2} 1_i^{obj} \times \sum_{c=1}^{20} \left(p_{i,c} - \hat{p}_{i,c}\right)^2 \\
+ \lambda_{no\_obj} \sum_{i=1}^{S^2} 1_i^{no\_obj} \times \sum_{j=1}^{B} \left(c_{i,j} - \hat{c}_{i,j}\right)^2
\end{aligned}
$$

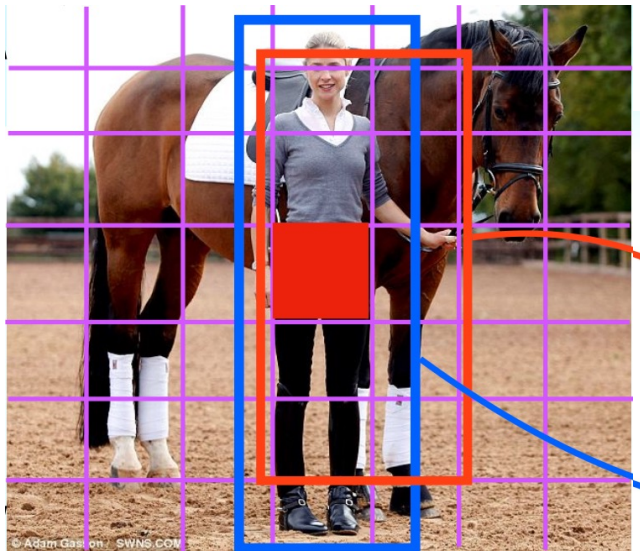


https://arxiv.org/pdf/2304.00501.pdf

Yolo-v1: Training Process



Yolo-v1: Output



$(\Delta x_1, \Delta y_1, \Delta w_1, \Delta h_1)$,C1 $(\Delta x_2, \Delta y_2, \Delta w_2, \Delta h_2)$,C2

$$\begin{cases} x_1 = \Delta x_1 \times 64 + x_a \\ y_1 = \Delta y_1 \times 64 + y_a \\ w_1 = \Delta w_1 \times 448 \\ h_1 = \Delta h_1 \times 448 \end{cases} \qquad \begin{cases} x_2 = \Delta x_2 \times 64 + x_a \\ y_2 = \Delta y_2 \times 64 + y_a \\ w_2 = \Delta w_2 \times 448 \\ h_2 = \Delta h_2 \times 448 \end{cases}$$

YOLO-v1 Limitations

Strong Spatial Constraints: Each cell only predicts two bounding boxes and can only have two class

YOLO struggles with small objects that appear in groups, like flocks of birds

Relative Coarse Features:

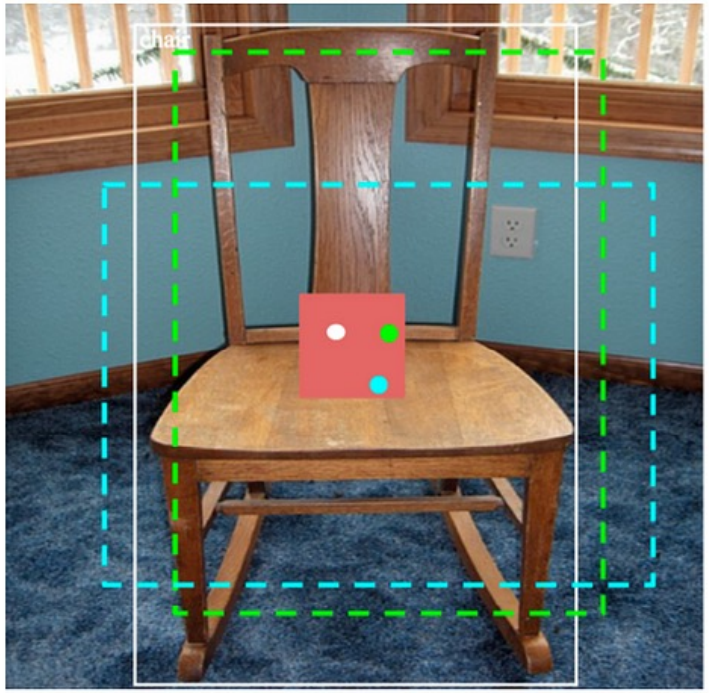
YOLO has many downsampling layers

It struggles to generalize to objects in new or unsual aspect ratios or configurations

Small-Object Localization

YOLO-v1 Limitations

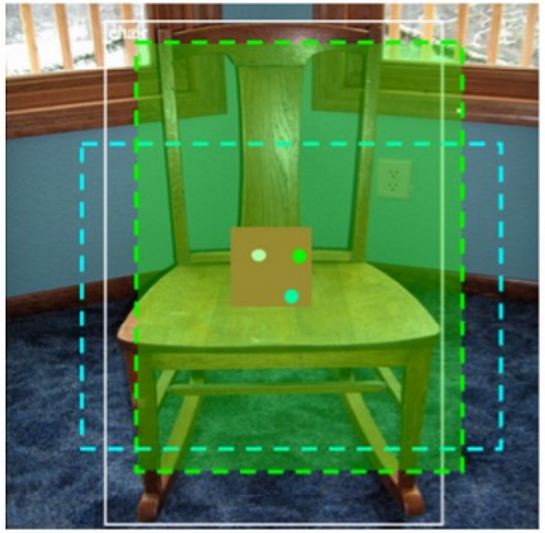
(a) Visualize bounding box predictions



$$\hat{y}_{g_x,g_y} = \begin{bmatrix} \hat{c}_1 \\ \hat{x}_1 \\ \hat{y}_1 \\ \hat{w}_1 \\ \hat{h}_1 \\ \hat{c}_2 \\ \hat{x}_2 \\ \hat{y}_2 \\ \hat{w}_2 \\ \hat{h}_2 \\ \hat{p}[1] \\ \vdots \\ \hat{p}[8] \\ \hat{p}[9] \\ \hat{p}[10] \\ \vdots \\ \hat{p}[20] \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \hat{b}_1 \\ \\ \\ \\ \\ \hat{b}_2 \\ \\ \\ \\ \\ \\ \hat{p} \\ \\ \end{matrix}$$

(b) Use IoUs to construct y_{g_x,g_y}

$$\hat{b}_1 = \underset{\hat{b} \in \{\hat{b}_1, \hat{b}_2\}}{\arg\max}\ IoU(b, \hat{b})$$



$$y_{g_x,g_y} = \begin{bmatrix} IoU(b,\hat{b}_1) \\ x \\ y \\ w \\ h \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ p[1] = 0 \\ \vdots \\ p[8] = 0 \\ p[9] = 1 \\ p[10] = 0 \\ \vdots \\ p[20] = 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ b \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ p \end{matrix}$$

Outline

- Object Detection Milestones: Transformer
- End-to-End Object Detection with Transformers
- Object Detection Milestones: YOLOs variants
- YOLO Motivations for OD
- YOLO-v1
- YOLO-v2

YOLO-v2 Motivations

YOLO v1 was faster than Faster R-CNN, but it was less accurate.

YOLO v1's weakness was the bounding box accuracy. It didn't predict object locations and sizes well, particularly bad at spotting small objects.

SSD, another single-stage detector, broke the record by being better (more accurate) than Faster R-CNN and even faster than YOLO v1.

Authors wanted to make their object detector to recognize a wide variety of objects. Pascal VOC object detection dataset contains only 20 classes. They wanted their model to recognize much more classes of objects

Changes in YOLO-v2

Batch Normalization

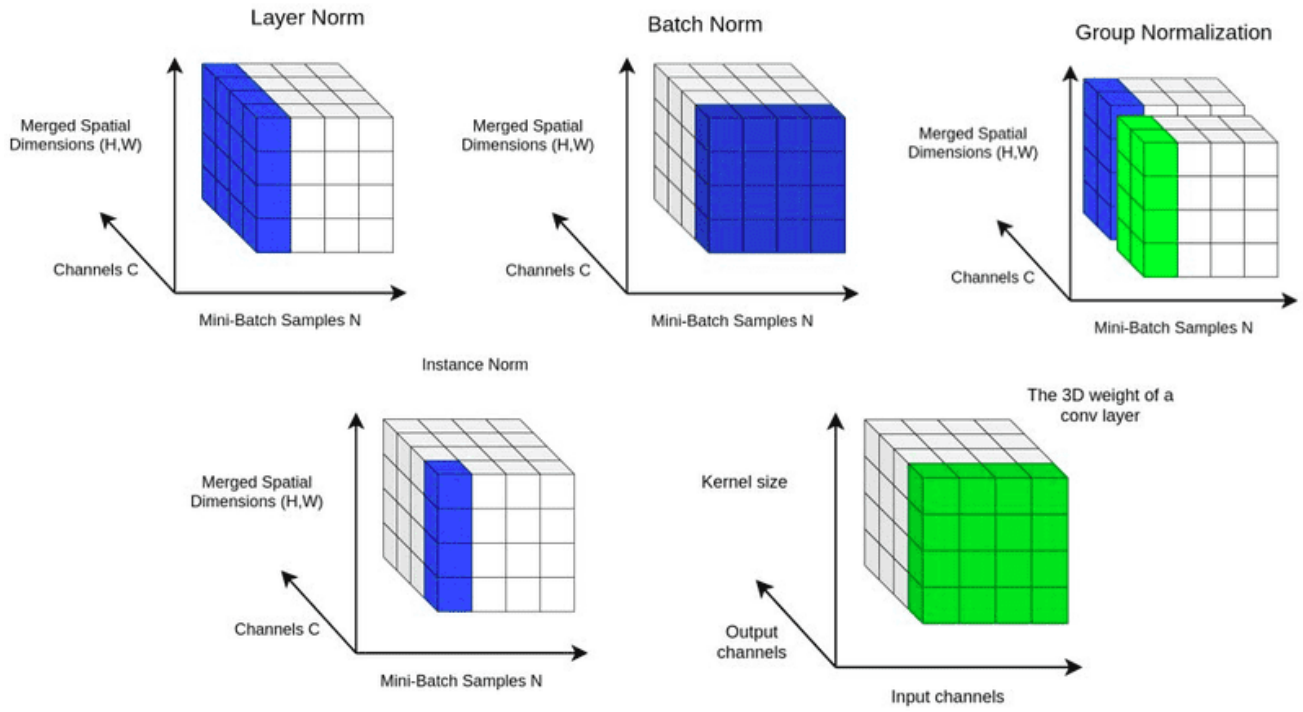
In YOLO v2, they added Batch Normalization to all convolutional layers
- It improved mAP by 2%
- It helped method model to avoid overtting

High-Resolution Classifier

- First, train the classifier on images of size 224 x 224
- Fine-tune the classifier on images of size 448 x 448 for 10 epochs on ImageNet
- Improve mAP by almost 4%

Changes in YOLO-v2

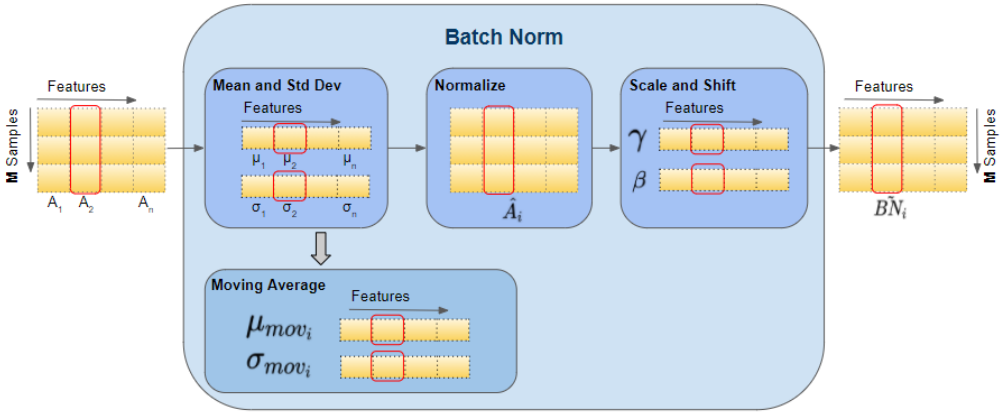
Batch Normalization



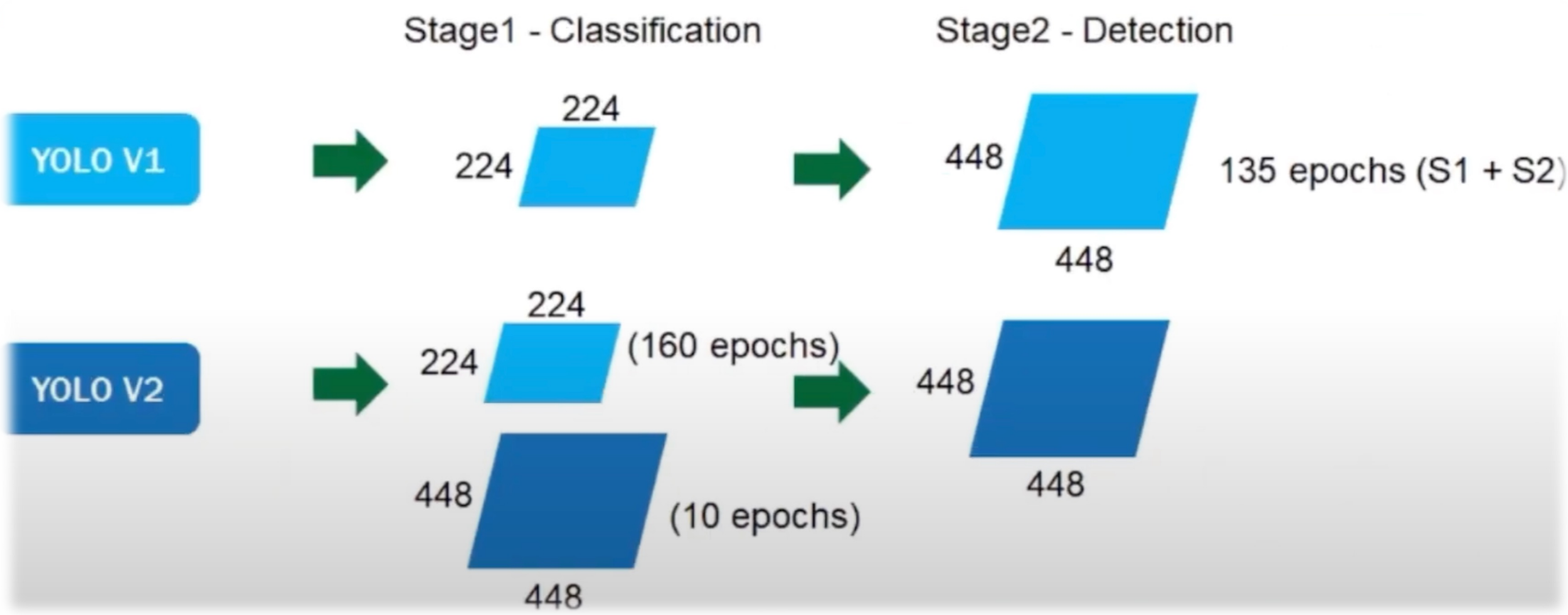
$$BN(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta$$

$$\mu_c(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_c(x))^2}$$

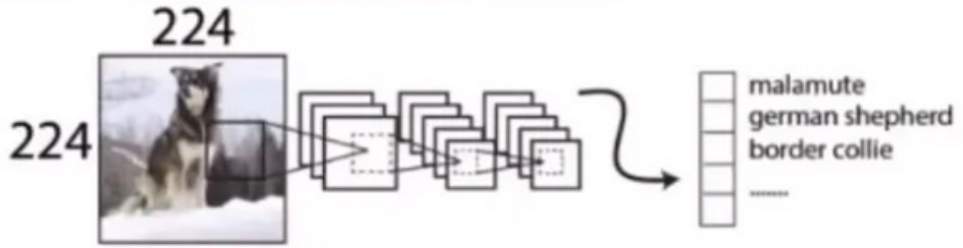


Changes in YOLO-v2

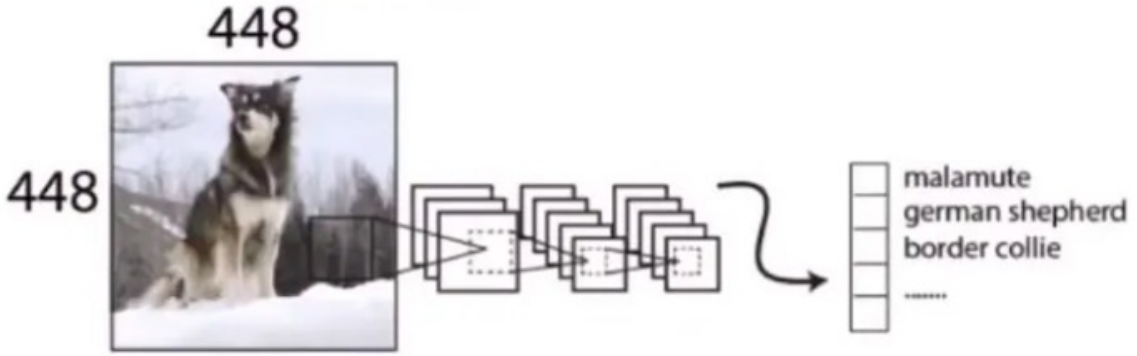


Changes in YOLO-v2

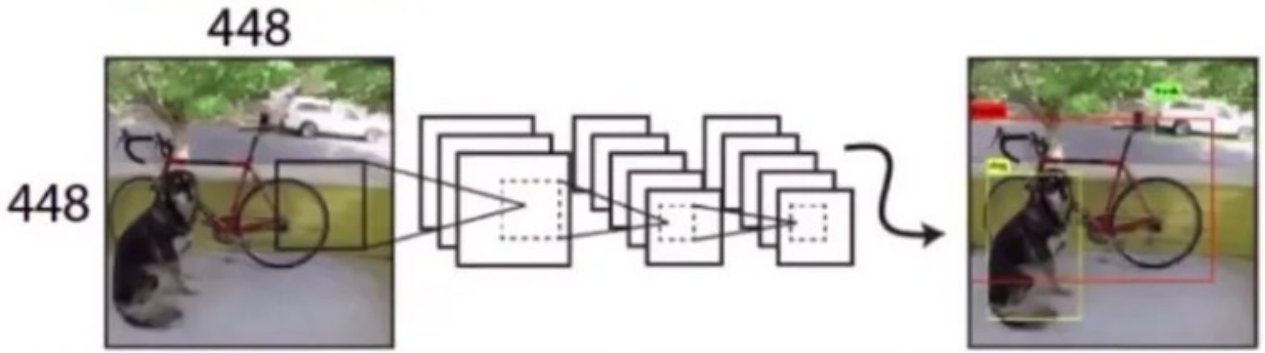
Train on ImageNet



Resize, fine-tune on ImageNet

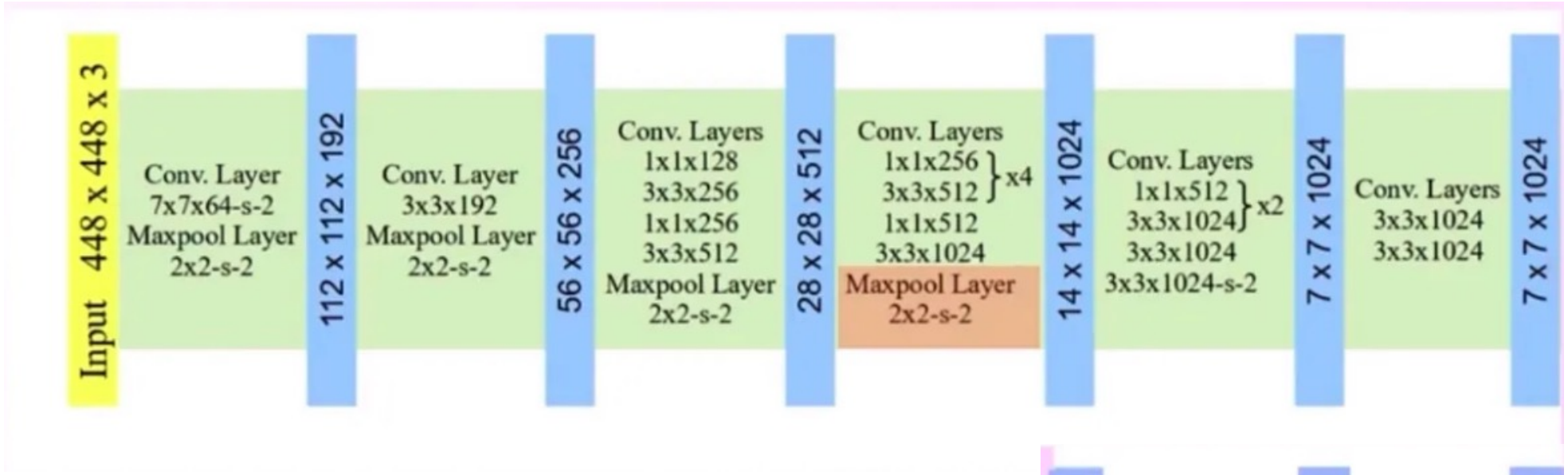


Fine-tune on detection

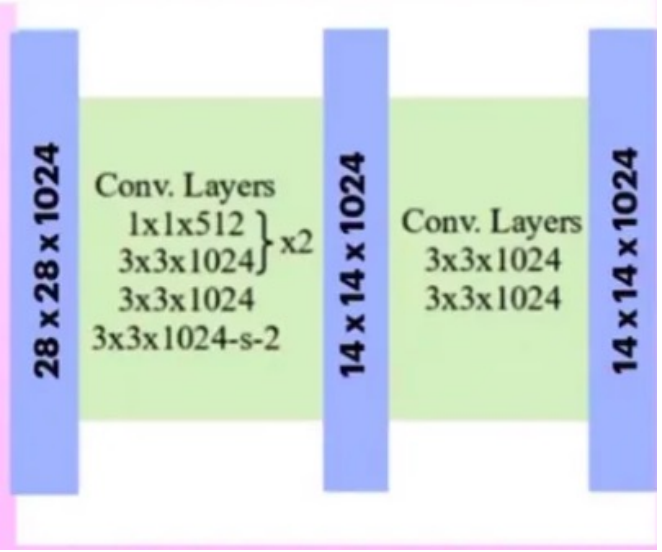


Changes in YOLO-v2

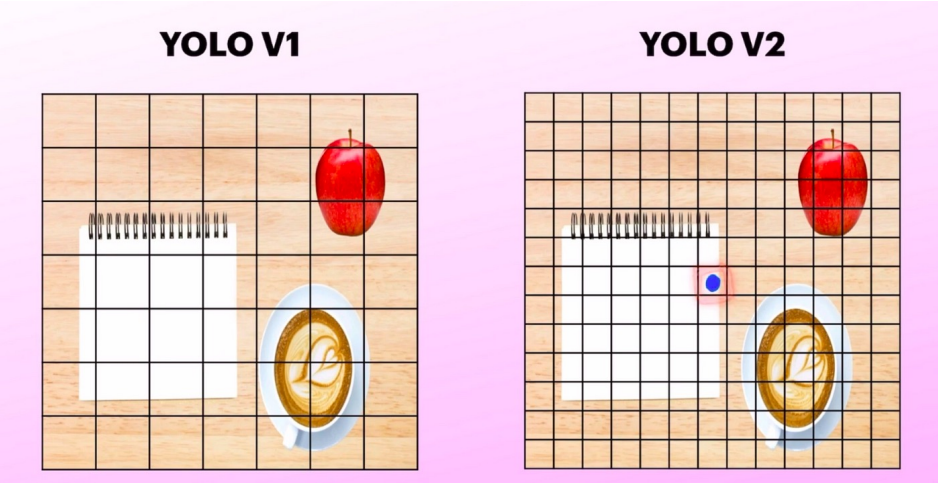
YOLOv1



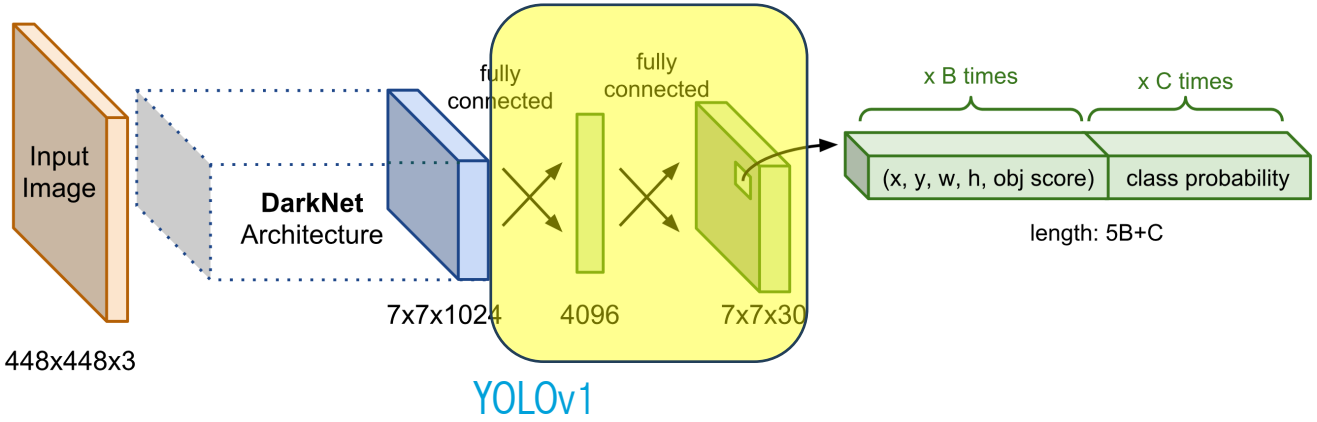
YOLOv2



Changes in YOLO-v2



YOLOv2 use 13x13 grids in stead of 14x14. Why?



YOLOv1

Problem of fully connected layer

Changes in YOLO-v2

Convolutional with Anchor Boxes

YOLO v1 suffered from a low recall rate

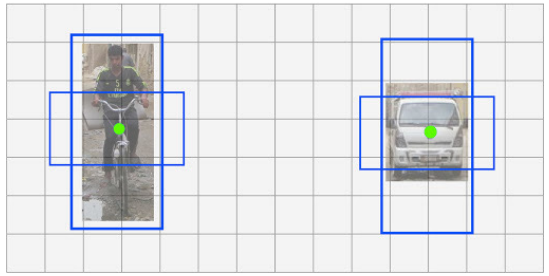
YOLOv2 removes all fully connected layers and uses anchor boxes to predict bounding boxes

YOLO v1 only predicted two bounding boxes per grid cell, which means a total of 98 (= 7 x 7 x 2) bounding boxes per image, much lower than Faster R-CNN.

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

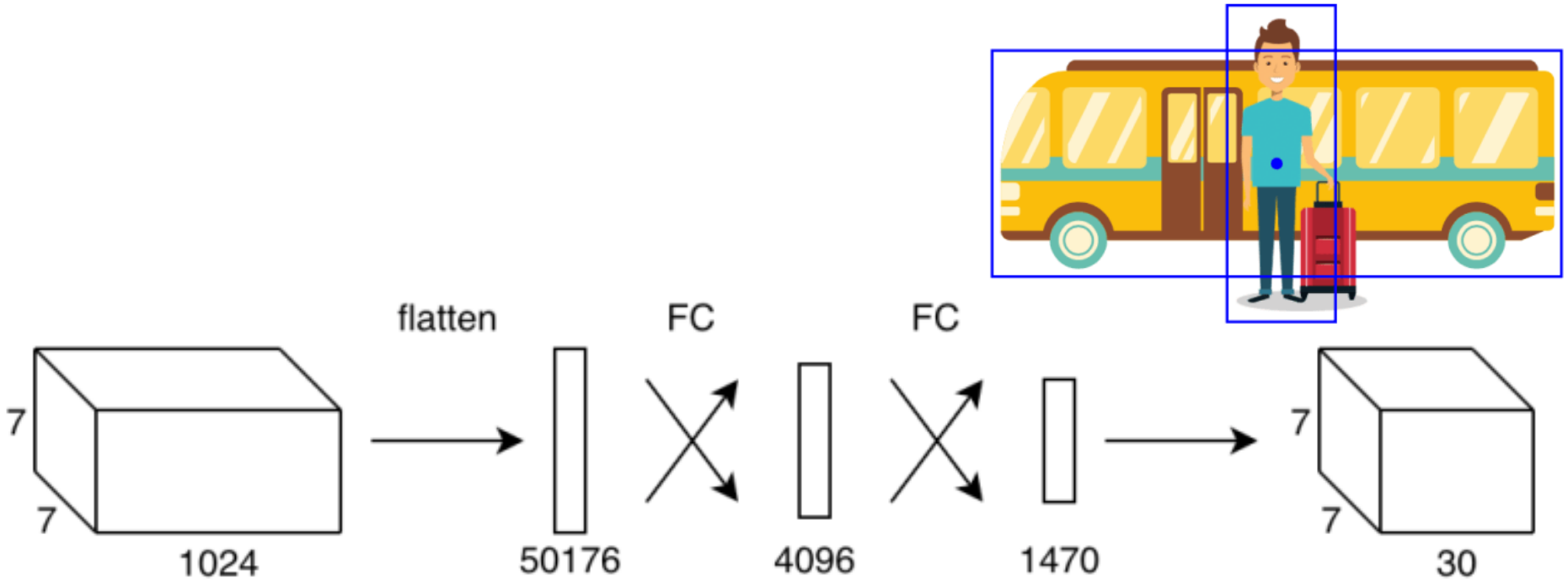
$$Precision = \frac{TP}{TP + FP}$$

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



Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

YOLO-v1 vs. YOLO-v2



Two bounding boxes had to share the same class probabilities. As such, increasing the number of bounding boxes would not benefit much. On the contrary, Faster R-CNN and SSD predicted class probabilities for each bounding box, making it easier to predict multiple classes sharing a similar center location.

YOLO-v1 vs. YOLO-v2

YOLOv1 was an anchor-free model that predicted the coordinates of B-boxes directly using fully connected layers in each grid cell.

Inspired by Faster-RCNN that predicts B-boxes using hand-picked priors known as anchor boxes, YOLOv2 also works on the same principle.

Unlike YOLOv1, wherein each grid cell, the model predicted one set of class probabilities per grid cell, ignoring the number of boxes B, YOLOv2 predicted class and objectness for every anchor box.

Changes in YOLO-v2

Dimension Clusters

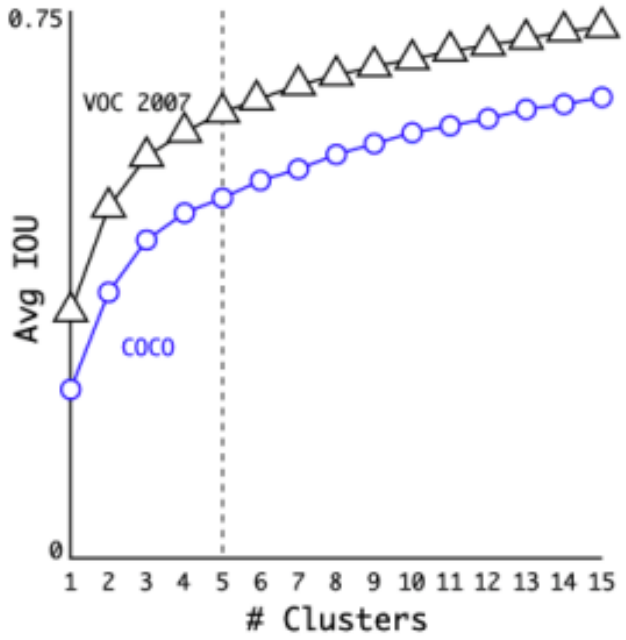
Dimension ClustersUnlike Faster-RCNN, which used hand-picked anchor boxes, YOLOv2 used a smart technique to find anchor boxes for the PASCAL VOC and MS COCO datasets.

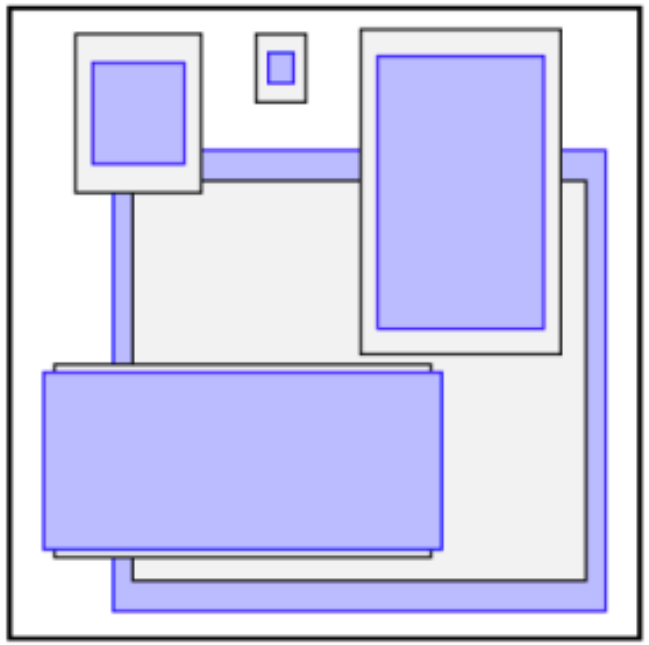
Redmon and Farhadi thought that instead of using hand-picked anchor boxes, we pick better priors that reflect the data more closely. It would be a great starting point for the network, and it would become much easier for the network to predict the detections and optimize faster.

Changes in YOLO-v2

Dimension Clusters

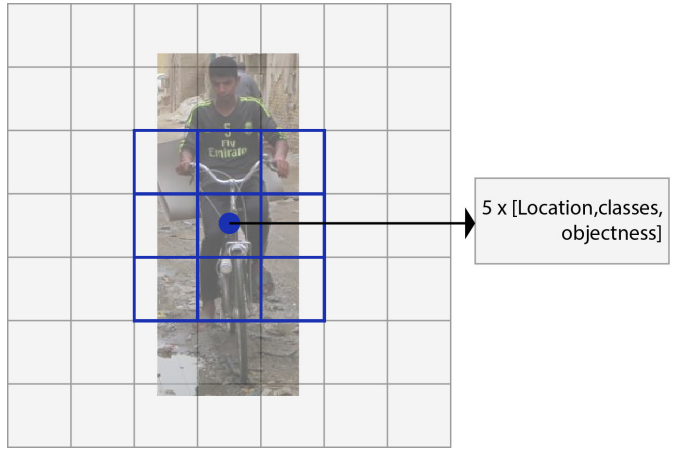
Using k-means clustering on the training set bounding boxes to find good anchor boxes or priors.





a) They picked the distance function as follows: d(box, centroid) = 1 − IOU(box, centroid).

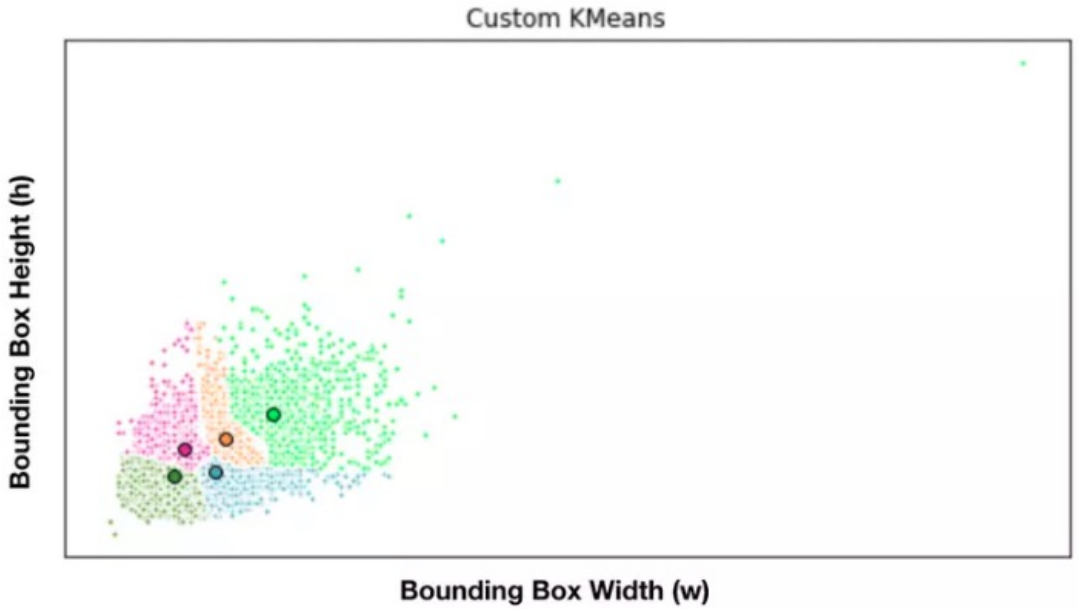
b) They ran K-Means with a various value of *k* and found out that k=5 gives a good tradeoff between between model complexity and high recall.

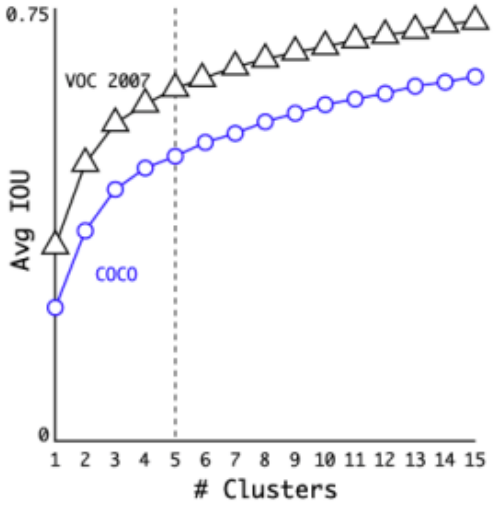


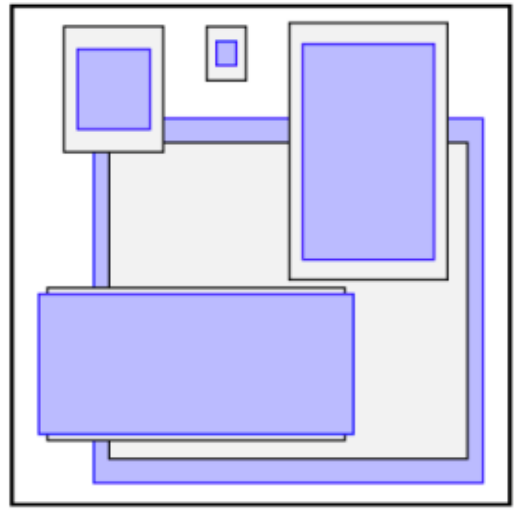
Changes in YOLO-v2

Dimension Clusters

Using k-means clustering on the training set bounding boxes to find good anchor boxes or priors.







Changes in YOLO-v2

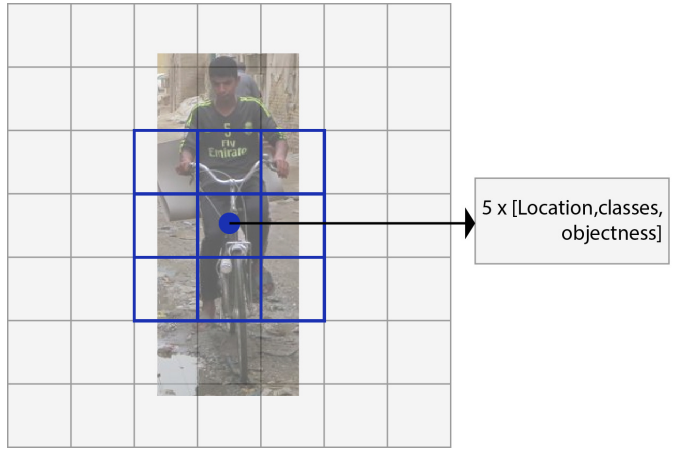
Dimension Clusters

Using k-means clustering on the training set bounding boxes to find good anchor boxes or priors.

Box Generation	Number of Anchors	Average IOU
Cluster Sum-Squared Distance	5	58.7
Cluster IOU	5	61.0
Anchor Boxes	9	60.9
Cluster IOU	9	67.2

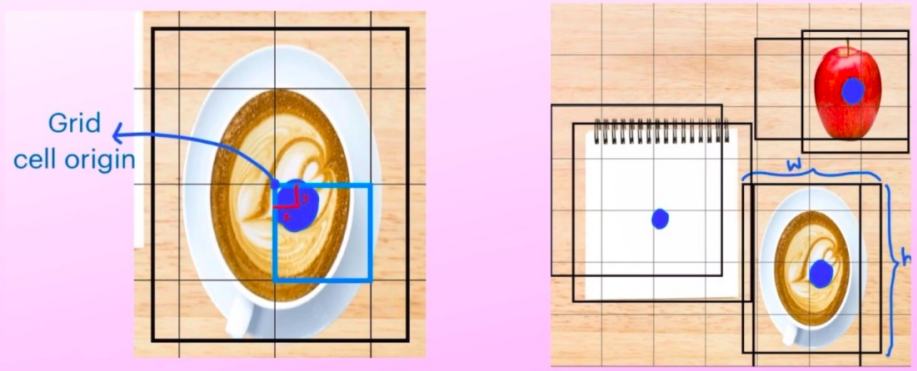
a) They picked the distance function as follows: d(box, centroid) = 1 − IOU(box, centroid).

b) They ran K-Means with a various value of *k* and found out that k=5 gives a good tradeoff between between model complexity and high recall.

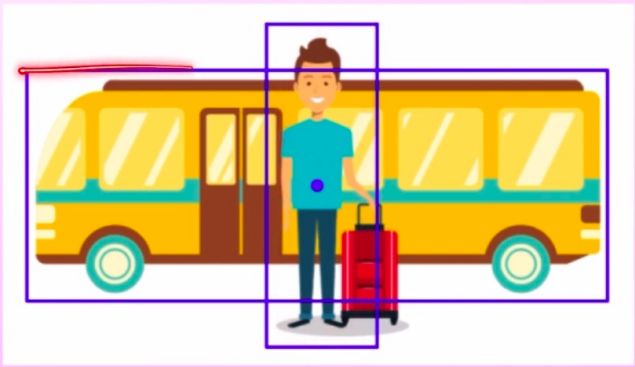


Poor Localization in YOLOv1

- Boxes are learnt relative to grid cell

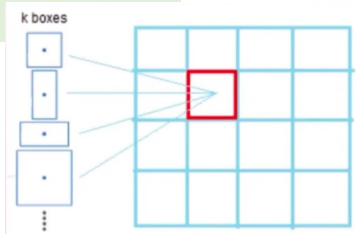


- Boxes are learnt relative to grid cell
- Objects can be of different shapes



Since bus is almost occupying the whole image, now predicting these kind of bigger objects from a single grid cell is very difficult because we don't know what is the shape of this object and how much range we have to look for in our predictions. So it can be difficult for the network to predict directly these boxes without any prior information. Now, what should we do in this case

Solution: Anchor boxes



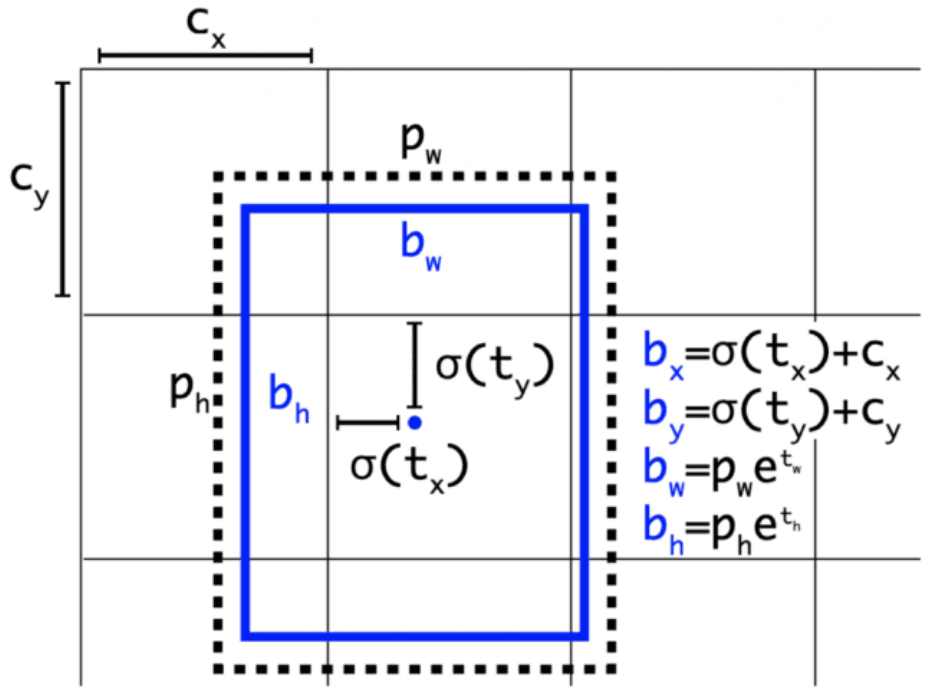
Problem of high Loss and and unstable training YOLOv1

- YOLO-V1 uses linear activation in the final layer - regression
- Predictions range - [-inf to +inf]
- Targets relative to grid cell - normalized to range of [0,1]
- Loss - Very high
- Unstable training
- Solution: Constraint the predictions

- Used logistic activation on location coordinates
- Range - [0,1]
- ***t_x = sigmoid(t_x), t_y = sigmoid(t_y)***
- Used exponential terms of width and height predictions
- Range - [0, +inf]
- ***t_w = exp(t_w), t_h = exp(t_h)***

Changes in YOLO-v2

Direct Location Prediction vs. Offset location predictions

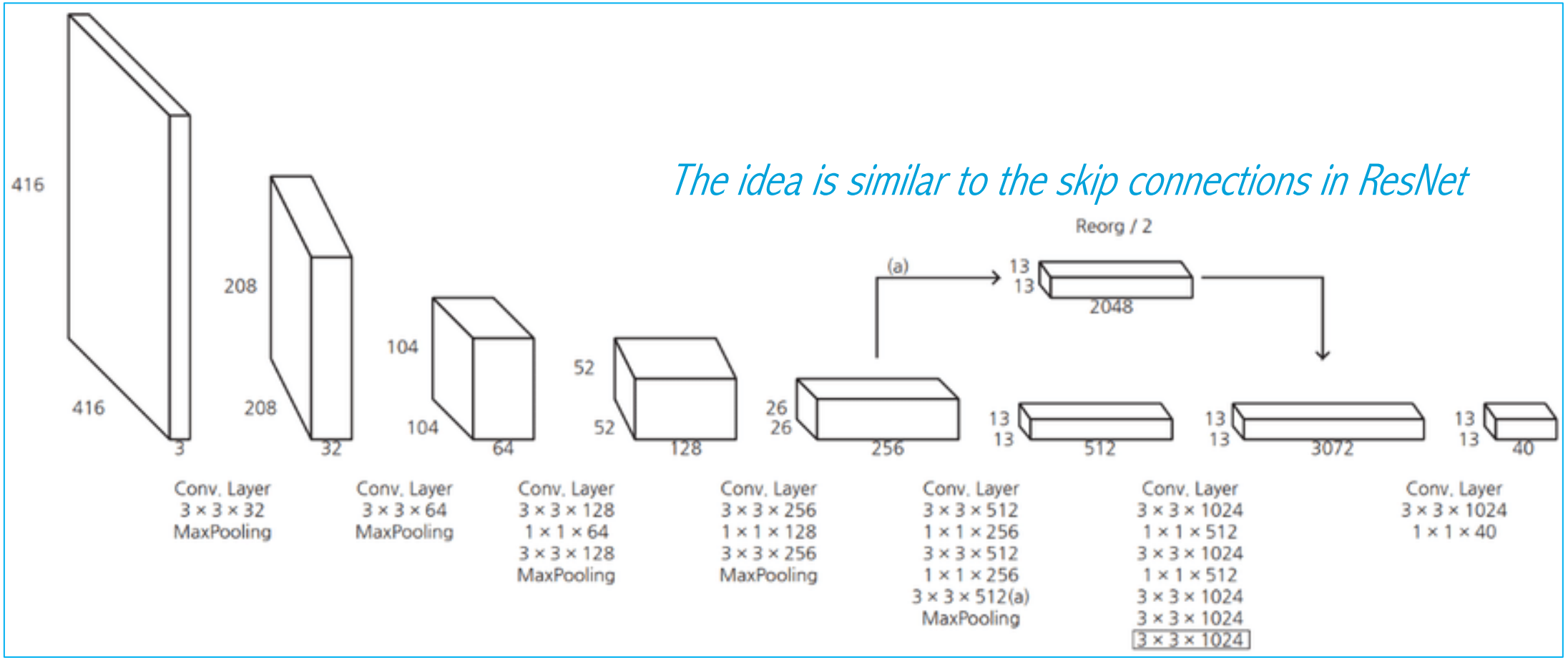


Instead of predicting the direct coordinates , they predict offsets to these bounding boxes during the training.

Here is the visualization. The blue box below is the predicted boundary box and the dotted rectangle is the anchor.

Changes in YOLO-v2

Add fine-grained feature



Changes in YOLO-v2

Multiple-Scale Training

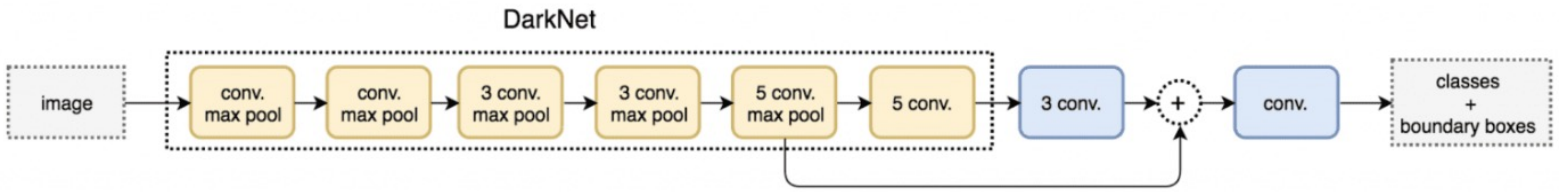
Detection Frameworks	Training Data	mAP	FPS
Fast R-CNN	2007+2012	70.0	0.5
Faster R-CNN with VGG-16 backbone	2007+2012	73.2	7.0
Faster R-CNN with ResNet backbone	2007+2012	76.4	5.0
YOLOv1	2007+2012	63.4	45.0
SSD300	2007+2012	74.3	46.0
SSD500	2007+2012	76.8	19.0
YOLOv2 with input size 288 x 288	2007+2012	69.0	**91.0**
YOLOv2 with input size 352 x 352	2007+2012	73.7	81.0
YOLOv2 with input size 416 x 416	2007+2012	76.8	67.0
YOLOv2 with input size 480 x 480	2007+2012	77.8	59.0
YOLOv2 with input size 544 x 544	2007+2012	**78.6**	40.0

Changes in YOLO-v2

Light-weight backbone

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 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 × 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			

Darknet-19 A fully convolutional model with 19 convolutional layers and five max-pooling layers was designed.



YOLO-v1 vs. YOLO-v2

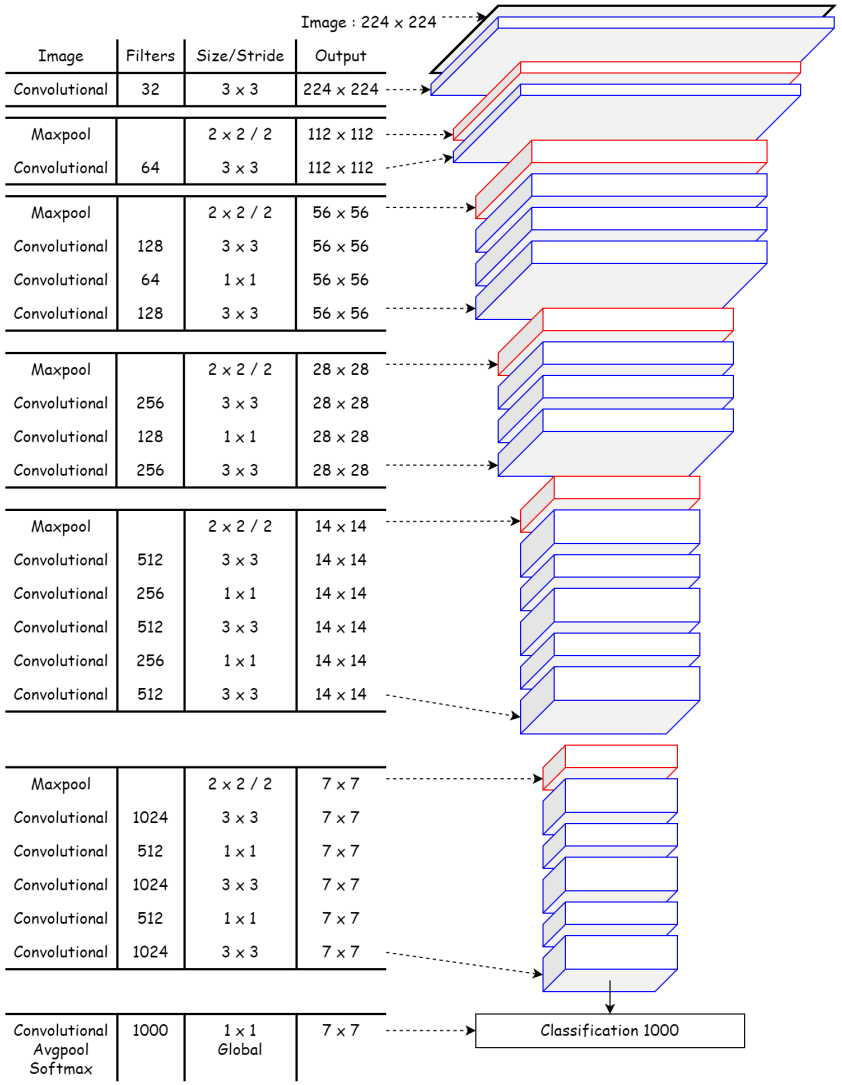
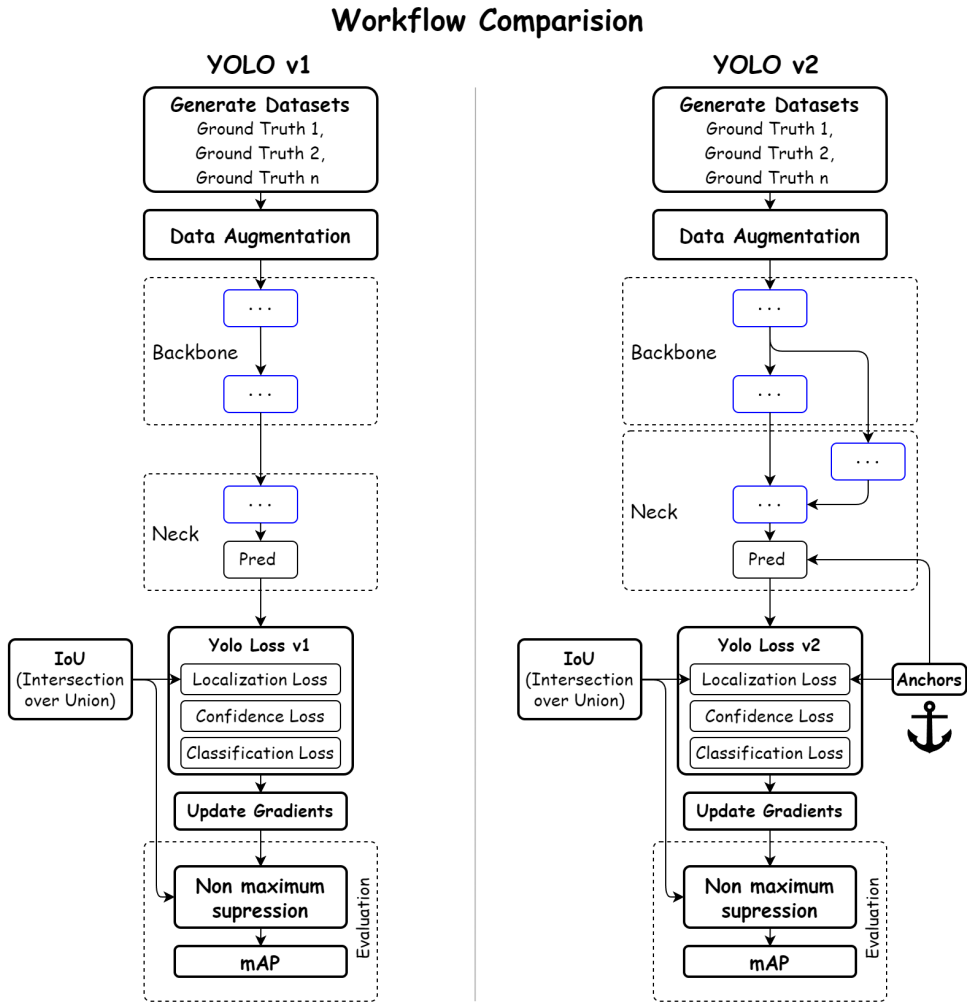


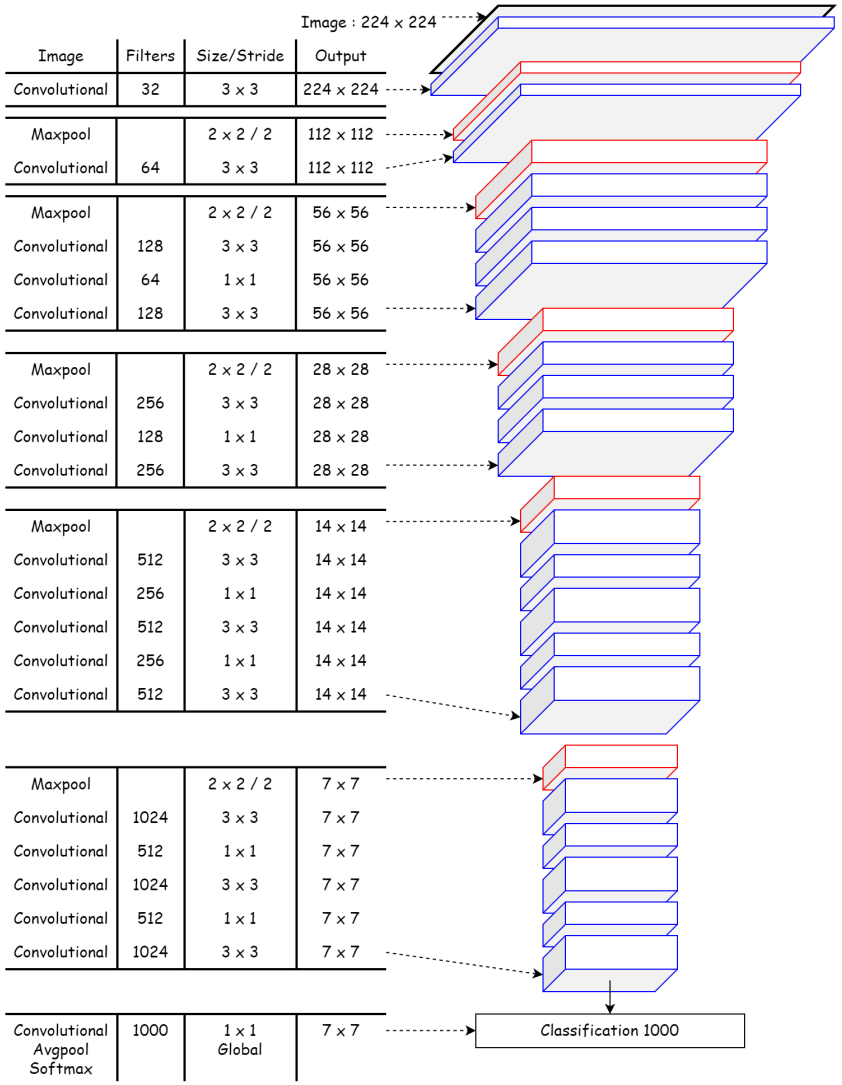
Image	Filters	Size/Stride	Output
Convolutional	32	3 x 3	224 x 224
Maxpool		2 x 2 / 2	112 x 112
Convolutional	64	3 x 3	112 x 112
Maxpool		2 x 2 / 2	56 x 56
Convolutional	128	3 x 3	56 x 56
Convolutional	64	1 x 1	56 x 56
Convolutional	128	3 x 3	56 x 56
Maxpool		2 x 2 / 2	28 x 28
Convolutional	256	3 x 3	28 x 28
Convolutional	128	1 x 1	28 x 28
Convolutional	256	3 x 3	28 x 28
Maxpool		2 x 2 / 2	14 x 14
Convolutional	512	3 x 3	14 x 14
Convolutional	256	1 x 1	14 x 14
Convolutional	512	3 x 3	14 x 14
Convolutional	256	1 x 1	14 x 14
Convolutional	512	3 x 3	14 x 14
Maxpool		2 x 2 / 2	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	512	1 x 1	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	512	1 x 1	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional Avgpool Softmax	1000	1 x 1 Global	7 x 7

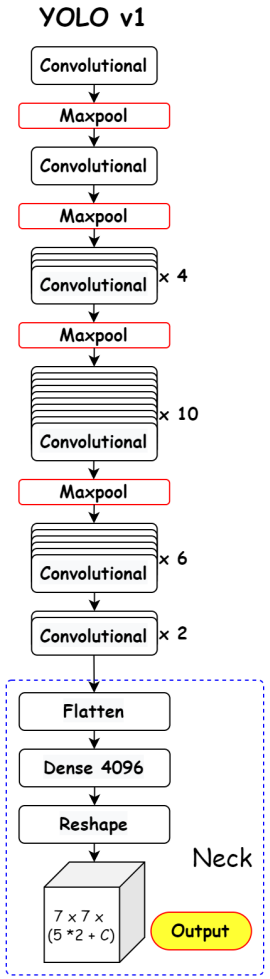


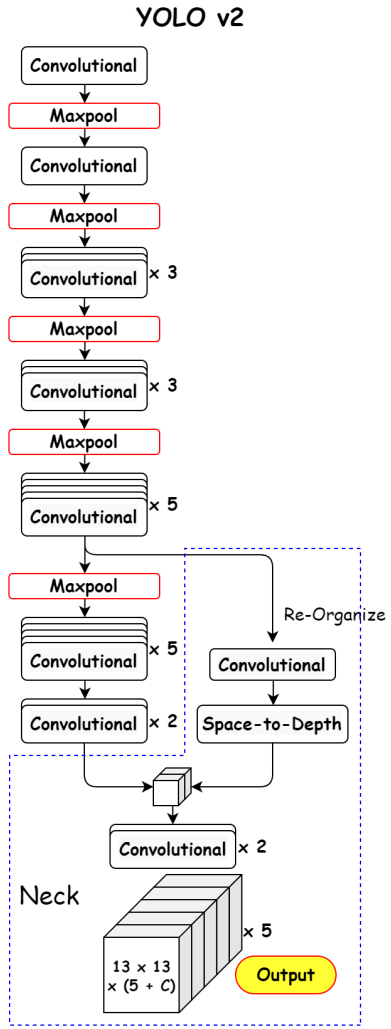
Evaluation will be explained in detail after the Yolo V3 Training.

https://wikidocs.net/167705

YOLO-v1 vs. YOLO-v2

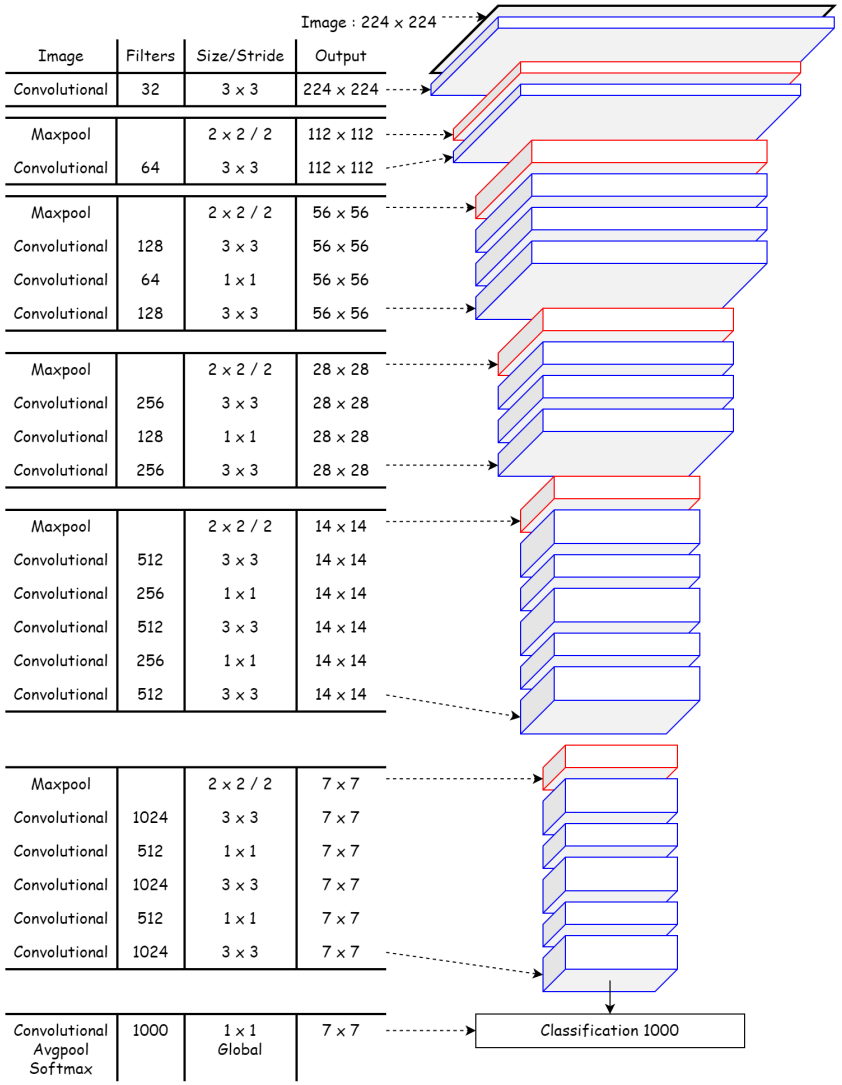




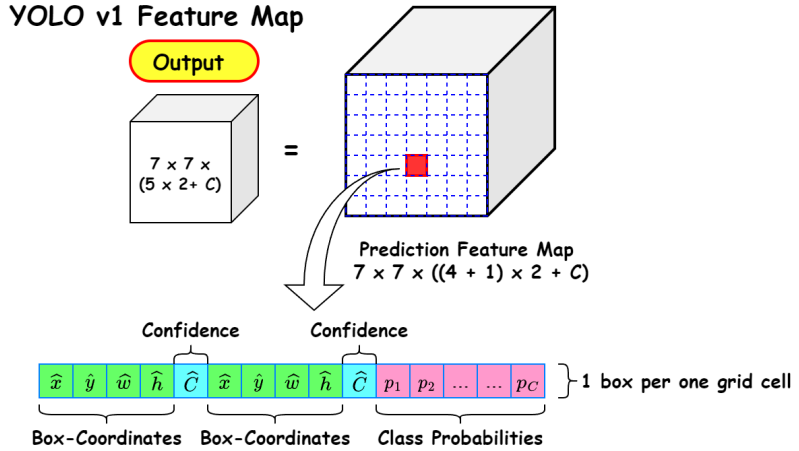


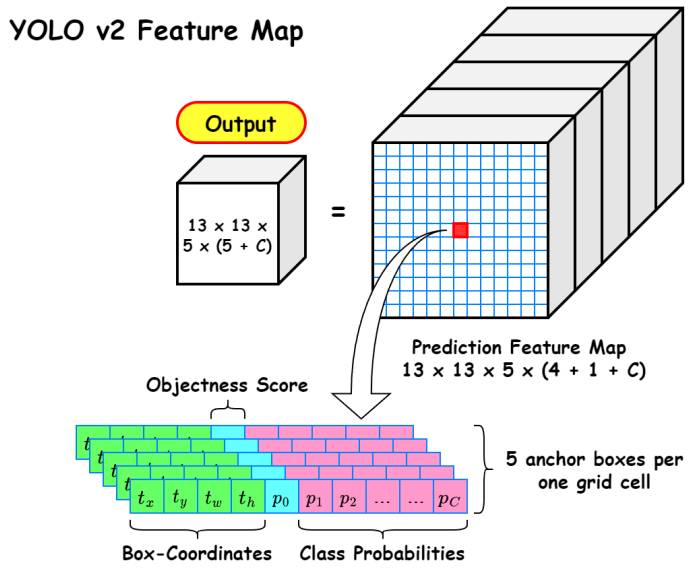
https://wikidocs.net/167705

YOLO-v1 vs. YOLO-v2



Head





https://wikidocs.net/167705

YOLO-v1 vs. YOLO-v2

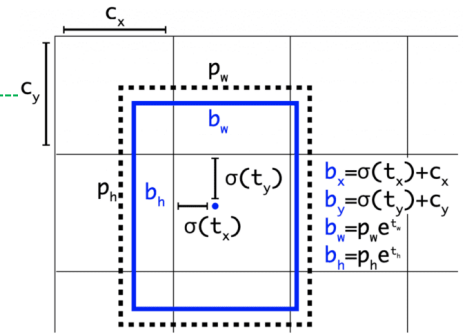
Loss function

YOLO v1 Loss

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]$$
$$+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]$$
$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$
$$+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2$$
$$+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{obj}\sum_{c\in\text{classes}}\left(p_i(c)-\hat{p}_i(c)\right)^2$$

YOLO v2 Loss

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[(x_i-b_{xi})^2+(y_i-b_{yi})^2\right]$$
$$+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{b_{wi}}\right)^2+\left(\sqrt{h_i}-\sqrt{b_{hi}}\right)^2\right]$$
$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left(C_i-\sigma(t_{oi})\right)^2$$
$$+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}\left(C_i-\sigma(t_{oi})\right)^2$$
$$+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{obj}\sum_{c\in\text{classes}}\left(p_i(c)-\hat{p}_i(c)\right)^2$$



$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w\, e^{t_w}$$
$$b_h = p_h\, e^{t_h}$$
$$\Pr(\text{object}) \star IOU(b, \text{object}) = \sigma(t_o)$$

where

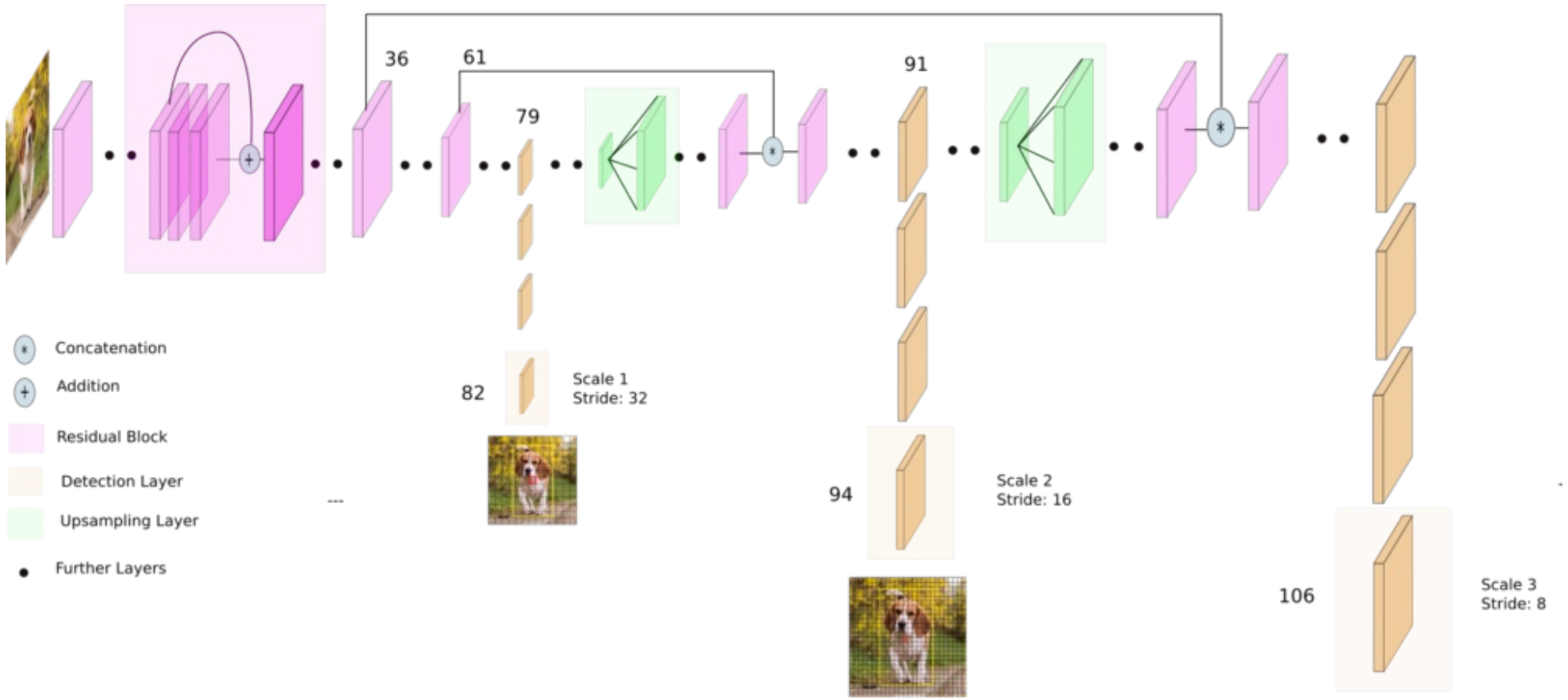
t_x, t_y, t_w, t_h	are predictions made by YOLO.
c_x, c_y	is the top left corner of the grid cell of the anchor.
p_w, p_h	are the width and height of the anchor.
c_x, c_y, p_w, p_h	are normalized by the image width and height.
b_x, b_y, b_w, b_h	are the predicted boundary box.
$\sigma(t_o)$	is the box confidence score.

https://wikidocs.net/167705

Outline

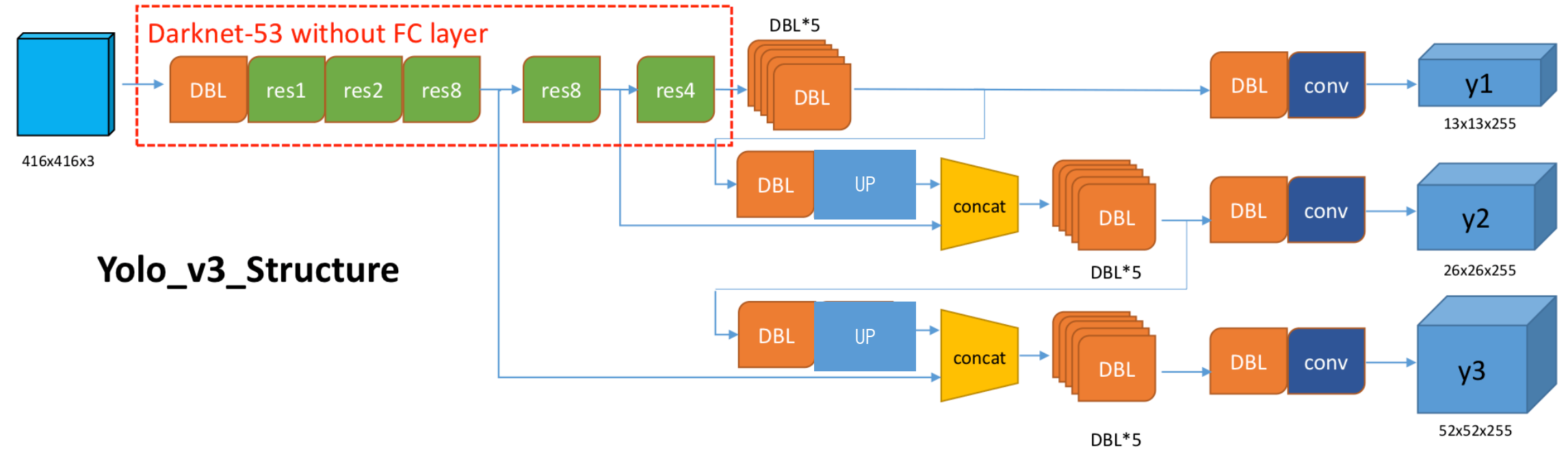
- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

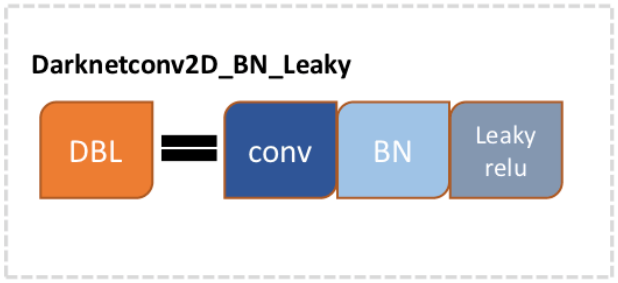
YOLO-v3 Architecture

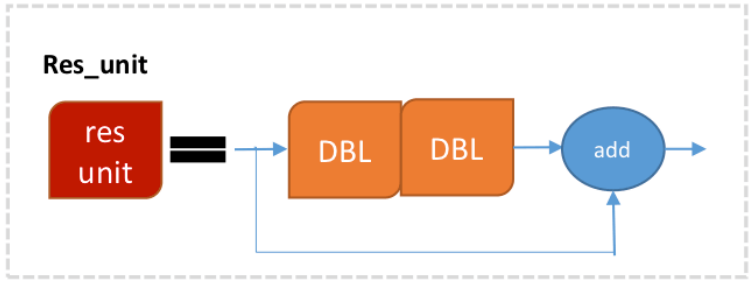


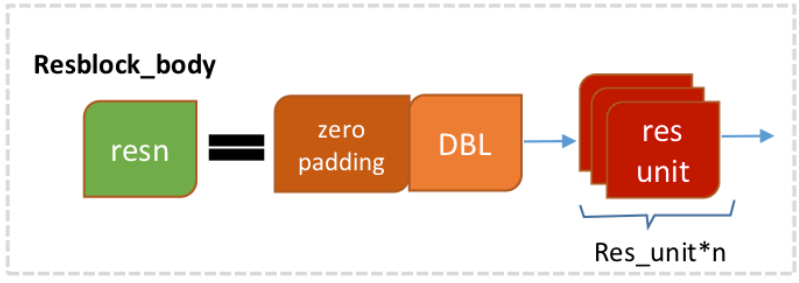
YOLO v3 uses a variant of Darknet, which originally has **53 layer network trained on ImageNet**. For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3. The detections are made at three layers **82nd, 94th and 106th layer.**

YOLO-v3 Architecture

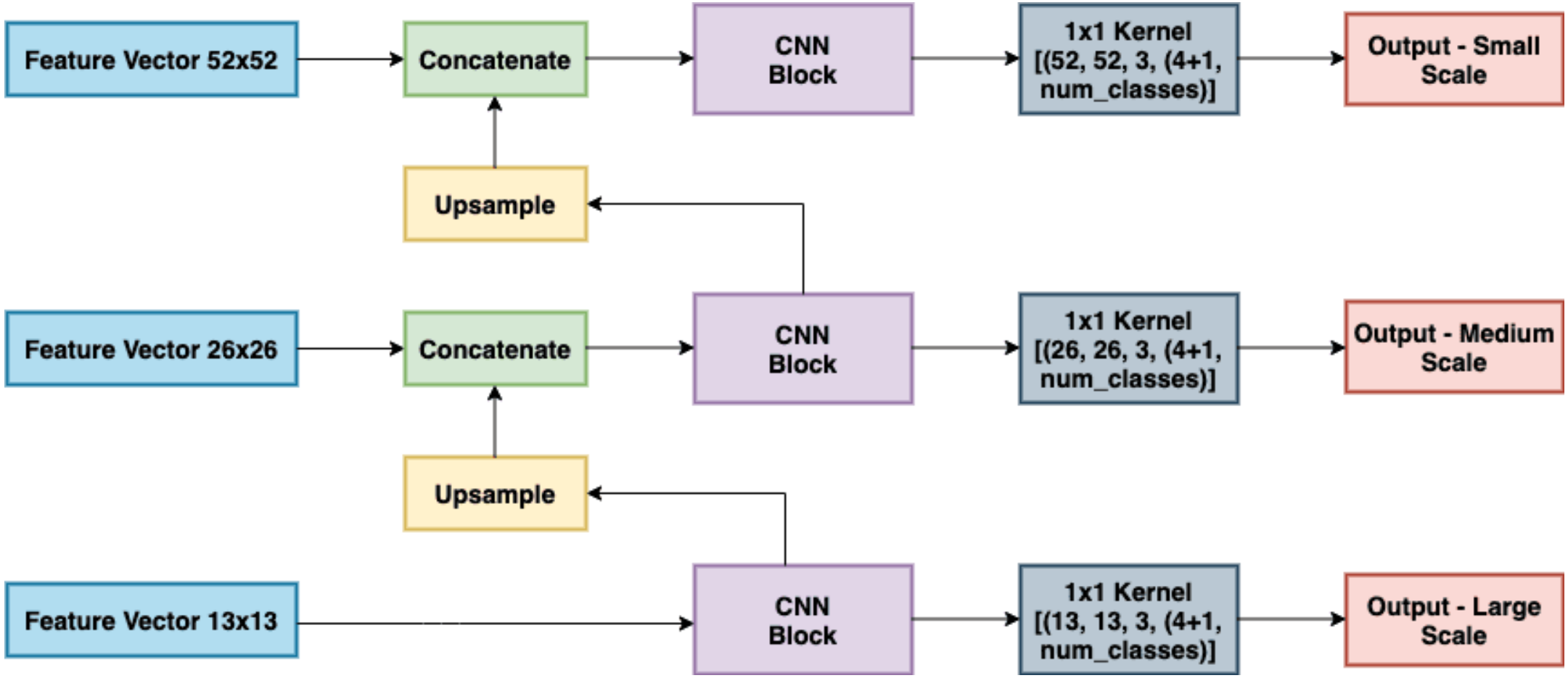








YOLO-v3 Architecture



YOLO-v3 Architecture

Darknet-19 (Input size up)

Image	Filters	Size/Stride	Output
Convolutional	32	3 x 3	416 x 416
Maxpool		2 x 2 / 2	208 x 208
Convolutional	64	3 x 3	208 x 208
Maxpool		2 x 2 / 2	104 x 104
Convolutional	128	3 x 3	104 x 104
Convolutional	64	1 x 1	104 x 104
Convolutional	128	3 x 3	104 x 104
Maxpool		2 x 2 / 2	52 x 52
Convolutional	256	3 x 3	52 x 52
Convolutional	128	1 x 1	52 x 52
Convolutional	256	3 x 3	52 x 52
Maxpool		2 x 2 / 2	26 x 26
Convolutional	512	3 x 3	26 x 26
Convolutional	256	1 x 1	26 x 26
Convolutional	512	3 x 3	26 x 26
Convolutional	256	1 x 1	26 x 26
Convolutional	512	3 x 3	26 x 26
Maxpool		2 x 2 / 2	13 x 13
Convolutional	1024	3 x 3	13 x 13
Convolutional	512	1 x 1	13 x 13
Convolutional	1024	3 x 3	13 x 13
Convolutional	512	1 x 1	13 x 13
Convolutional	1024	3 x 3	13 x 13

Neck

Darknet-53 (Input size up)

	Image	Filters	Size/Stride	Output
	Convolutional	32	3 x 3	416 x 416
	Convolutional	64	3 x 3 / 2	208 x 208
1 x	Convolutional	32	1 x 1	
	Convolutional	64	3 x 3	
	Residual			208 x 208
	Convolutional	128	3 x 3 / 2	104 x 104
2 x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			104 x 104
	Convolutional	256	3 x 3 / 2	52 x 52
8 x	Convolutional	128	1 x 1	
	Convolutional	256	3 x 3	
	Residual			52 x 52
	Convolutional	512	3 x 3 / 2	26 x 26
8 x	Convolutional	256	1 x 1	
	Convolutional	512	3 x 3	
	Residual			26 x 26
	Convolutional	1024	3 x 3 / 2	13 x 13
4 x	Convolutional	512	1 x 1	
	Convolutional	1024	3 x 3	
	Residual			13 x 13

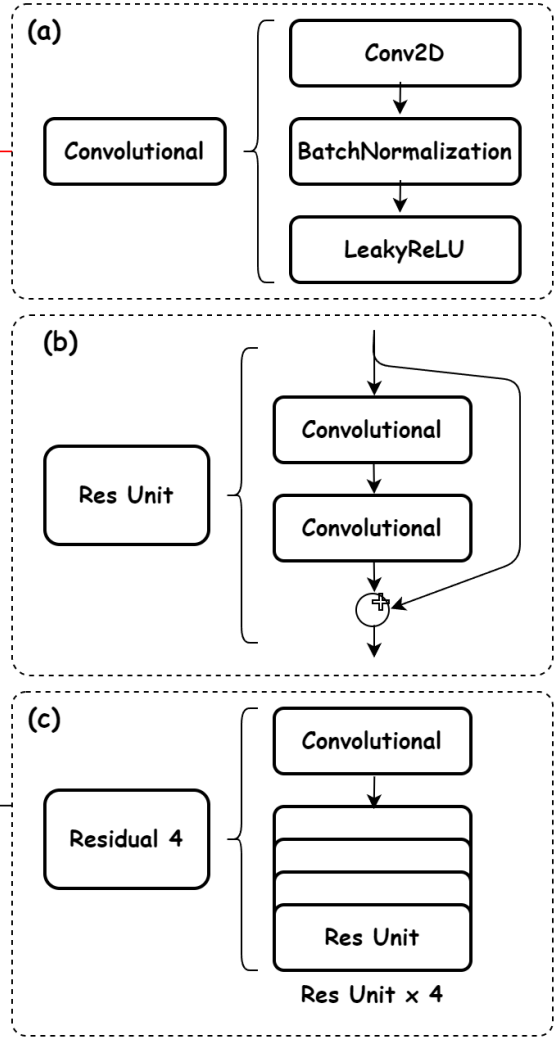
Neck(FPN)

https://wikidocs.net/177719

YOLO-v3 Architecture

Darknet-53 (in Paper)

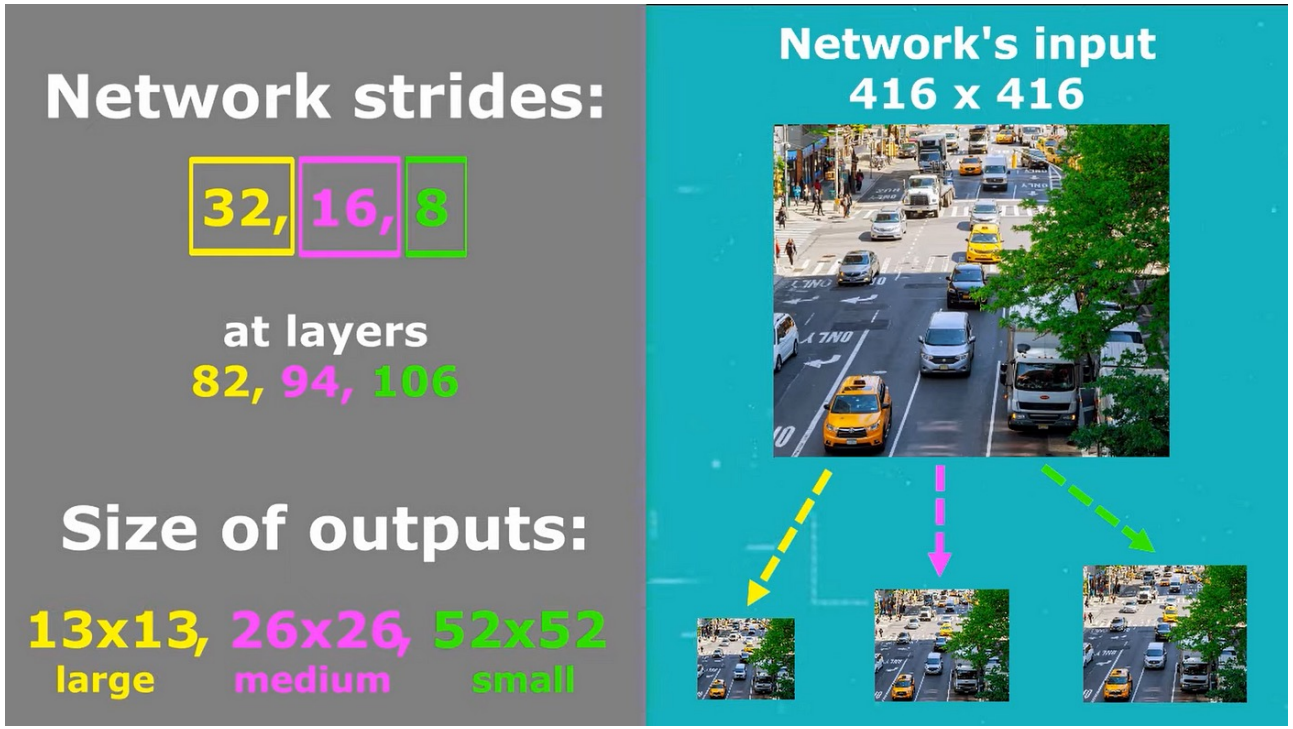
	Image	Filters	Size/Stride	Output
	Convolutional	32	3 x 3	256 x 256
	Convolutional	64	3 x 3 / 2	128 x 128
1 x	Convolutional	32	1 x 1	
	Convolutional	64	3 x 3	
	Residual			128 x 128
	Convolutional	128	3 x 3 / 2	64 x 64
2 x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			64 x 64
	Convolutional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

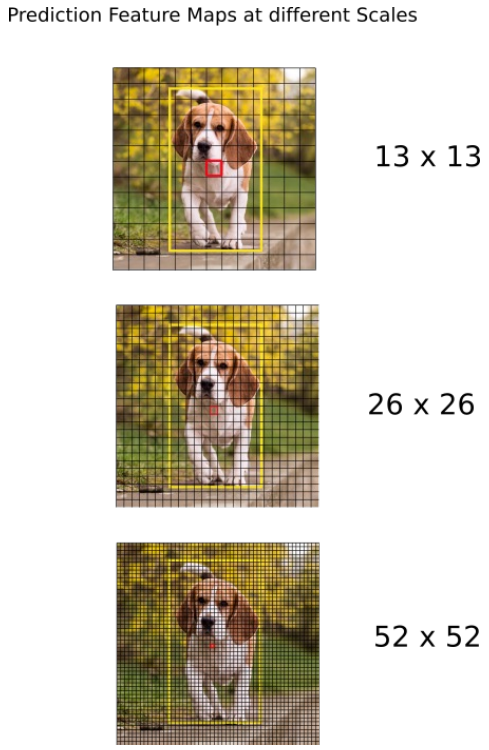


https://wikidocs.net/177719

YOLO-v3 Architecture

YOLOv3 makes detections at 3 different places in the network. These are layers 82nd, 94th and 106th layer. Network down samples input image by following factors **32, 16 and 8 at 82nd, 94th, 106th** layer accordingly, these numbers are called strides to the network and they show how the output at 3 places in the network are smaller than network input.



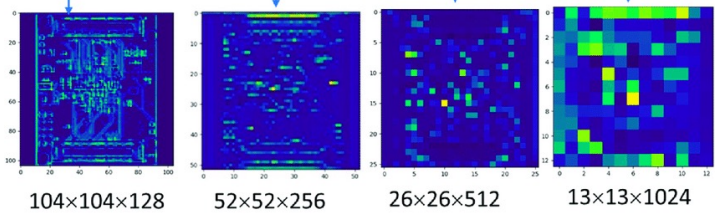


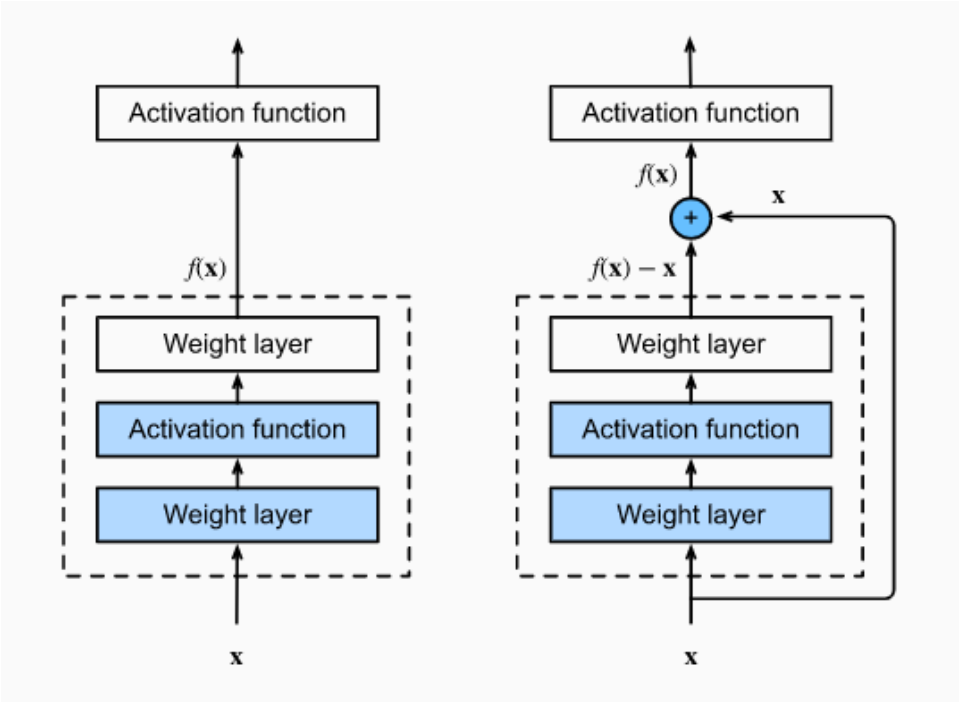
https://medium.com/analytics-vidhya/understanding-yolo-and-implementing-yolov3-for-object-detection-5f1f748cc63a

YOLO-v3 Architecture

Feature extraction: Yolov3 uses a larger network to perform feature extraction than yolov2. This model is known as DARKNET-53 which has 53 convolution layers with residual connections. We ignore the last three layers of darknet-53 (avgpool layer, fc layer, softmax layer) as these layers are mainly used for image classification. In this object detection task, we are using darknet-53 only to extract image features so these three layers will not be needed.

	Layer	filter	strides	output
	Convolutional	3×3×32	1	416×416×32
	Convolutional	3×3×64	2	208×208×64
1×	Convolutional	1×1×32	1	
	Convolutional	3×3×64	1	
	Residual			208×208×64
	Convolutional	3×3×128	2	104×104×128
2×	Convolutional	1×1×64	1	
	Convolutional	3×3×128	1	
	Residual			104×104×128
	Convolutional	3×3×256	2	52×52×256
8×	Convolutional	1×1×128	1	
	Convolutional	3×3×256	1	
	Residual			52×52×256
	Convolutional	3×3×512	2	26×26×512
8×	Convolutional	1×1×256	1	
	Convolutional	3×3×512	1	
	Residual			26×26×512
	Convolutional	3×3×1024	2	13×13×1024
4×	Convolutional	1×1×512	1	
	Convolutional	3×3×1024	1	
	Residual			13×13×1024



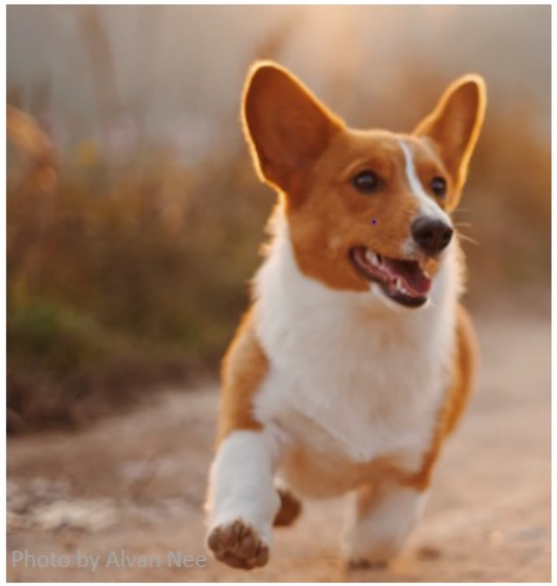


YOLO-

Feature extraction: Yolov3 uses a larger network to perform feature extraction than yolov2. This model is known as DARKNET-53 which has 53 convolution layers with residual connections. We ignore the last three layers of darknet-53 (avgpool layer, fc layer, softmax layer) as these layers are mainly used for image classification. In this object detection task, we are using darknet-53 only to extract image features so these three layers will not be needed.

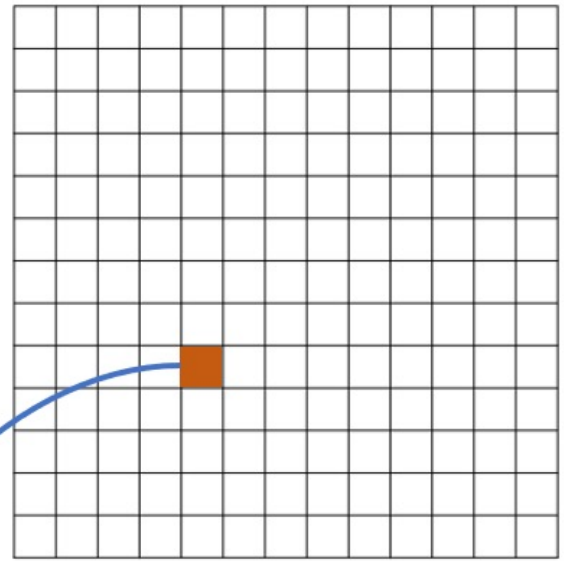
```
def residual_block(input_layer, input_channel, filter_num1, filter_num2):
    short_cut = input_layer
    conv = convolutional(input_layer, filters_shape=(1, 1, input_channel, filter_num1))
    conv = convolutional(conv       , filters_shape=(3, 3, filter_num1,   filter_num2))
    residual_output = short_cut + conv
    return residual_output

def darknet53(input_data, training=False): #python implementation of darknet-53
    input_data = convolutional(input_data, (3, 3,  3,  32))
    input_data = convolutional(input_data, (3, 3, 32,  64), downsample=True)

    for i in range(1):
        input_data = residual_block(input_data,  64,  32, 64)

    input_data = convolutional(input_data, (3, 3,  64, 128), downsample=True)

    for i in range(2):
        input_data = residual_block(input_data, 128,  64, 128)

    input_data = convolutional(input_data, (3, 3, 128, 256), downsample=True)

    for i in range(8):
        input_data = residual_block(input_data, 256, 128, 256)

    route_1 = input_data
    input_data = convolutional(input_data, (3, 3, 256, 512), downsample=True)

    for i in range(8):
        input_data = residual_block(input_data, 512, 256, 512)

    route_2 = input_data
    input_data = convolutional(input_data, (3, 3, 512, 1024), downsample=True)

    for i in range(4):
        input_data = residual_block(input_data, 1024, 512, 1024)

    return route_1, route_2, input_data
```

Output Layer Explanation



Pre-processing Image
(416, 416, 3)

YOLO v3
Network



Encoding*
(13, 13, **3**, (4+1+80))**

Objectness
score

3 anchor boxes per
one grid cell***

Bounding box
offset: x, y, w, h

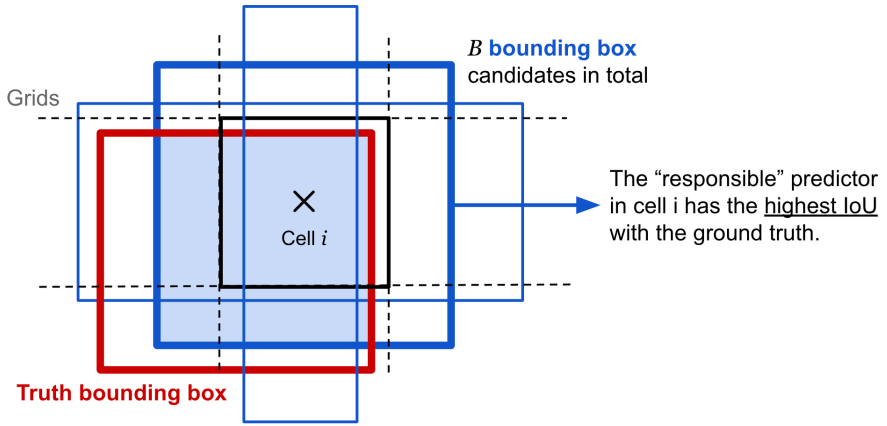
Class
probabilities

* In YOLO v3, the final output of detectors will be in shape 13x13, 26x26, 52x52. The example above is one of the output in 13x13 for illustration purposes only
** (13, 13, 3x(4+1+80)) are flatten into (13, 13, 255)
*** In YOLO v3, there are 3 anchor boxes per grid cell

YOLO-v3 Architecture

In total 9 anchor boxes are used, 3 anchor boxes for each scale, three biggest anchors for the first scale, the next three for the second scale, and the last three for the third. This means at each output layer every grid scale of feature map can predict 3 bounding boxes using 3 anchor boxes.

To calculate these anchors K-Means Clustering is applied in YOLOv3.
The width and height of anchors,
For, Scale 1: (116x90), (156x198), (373x326)
Scale 2: (30x61), (62x45), (59x119)
Scale 3: (10x13), (16x30), (33x23)
So, for, Scale 1: we have, 13x13x3 = 507 bounding box
Scale 2: we have, 26x26x3 = 2028 bounding box
Scale 3: we have, 52x52x3 = 8112 bounding box
In total, YOLOv3 predicts 10,847 boxes.



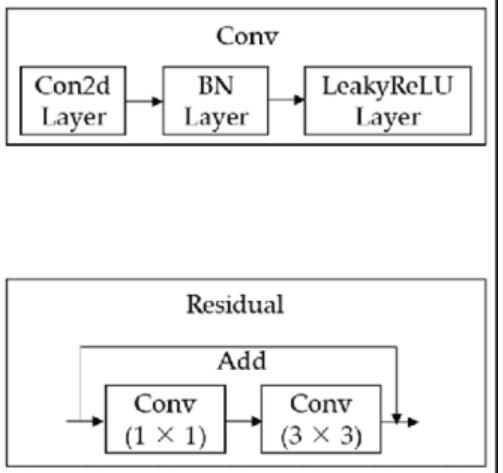
YOLO-v3 Architecture

YOLO v2 introduced darknet-19, which is 19-layer network supplemented with 11 more layers for object detection. However, having a 30-layer architecture, YOLO v2 often struggled at detecting small objects. Therefore, the authors introduce successive 3 × 3 and 1 × 1 convolutional layers followed by some shortcut connections allowing the network be much deeper. Thus, the authors introduce their Darknet-53, which is shown below.

	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			

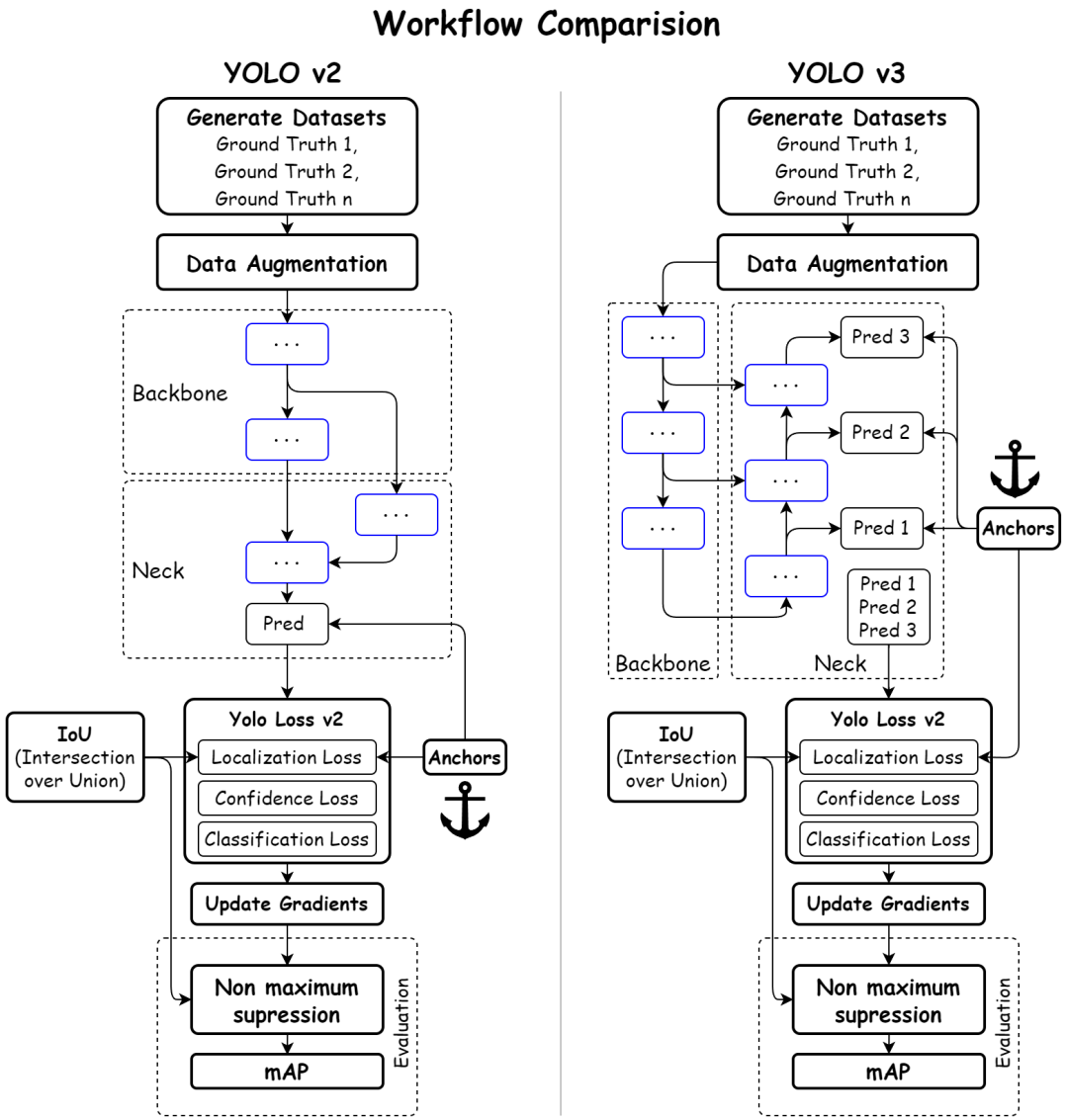
CNN architecture of Darknet-53

Layer	Filters size	Repeat	Output size
Image			416 × 416
Conv	32 3 × 3/1	1	416 × 416
Conv	64 3 × 3/2	1	208 × 208
Conv Conv Residual	32 1 × 1/1 64 3 × 3/1	Conv Conv Residual × 1	208 × 208 208 × 208 208 × 208
Conv	128 3 × 3/2	1	104 × 104
Conv Conv Residual	64 1 × 1/1 128 3 × 3/1	Conv Conv Residual × 2	104 × 104 104 × 104 104 × 104
Conv	256 3 × 3/2	1	52 × 52
Conv Conv Residual	128 1 × 1/1 256 3 × 3/1	Conv Conv Residual × 8	52 × 52 52 × 52 52 × 52
Conv	512 3 × 3/2	1	26 × 26
Conv Conv Residual	256 1 × 1/1 512 3 × 3/1	Conv Conv Residual × 8	26 × 26 26 × 26 26 × 26
Conv	1024 3 × 3/2	1	13 × 13
Conv Conv Residual	512 1 × 1/1 1024 3 × 3/1	Conv Conv Residual × 4	13 × 13 13 × 13 13 × 13

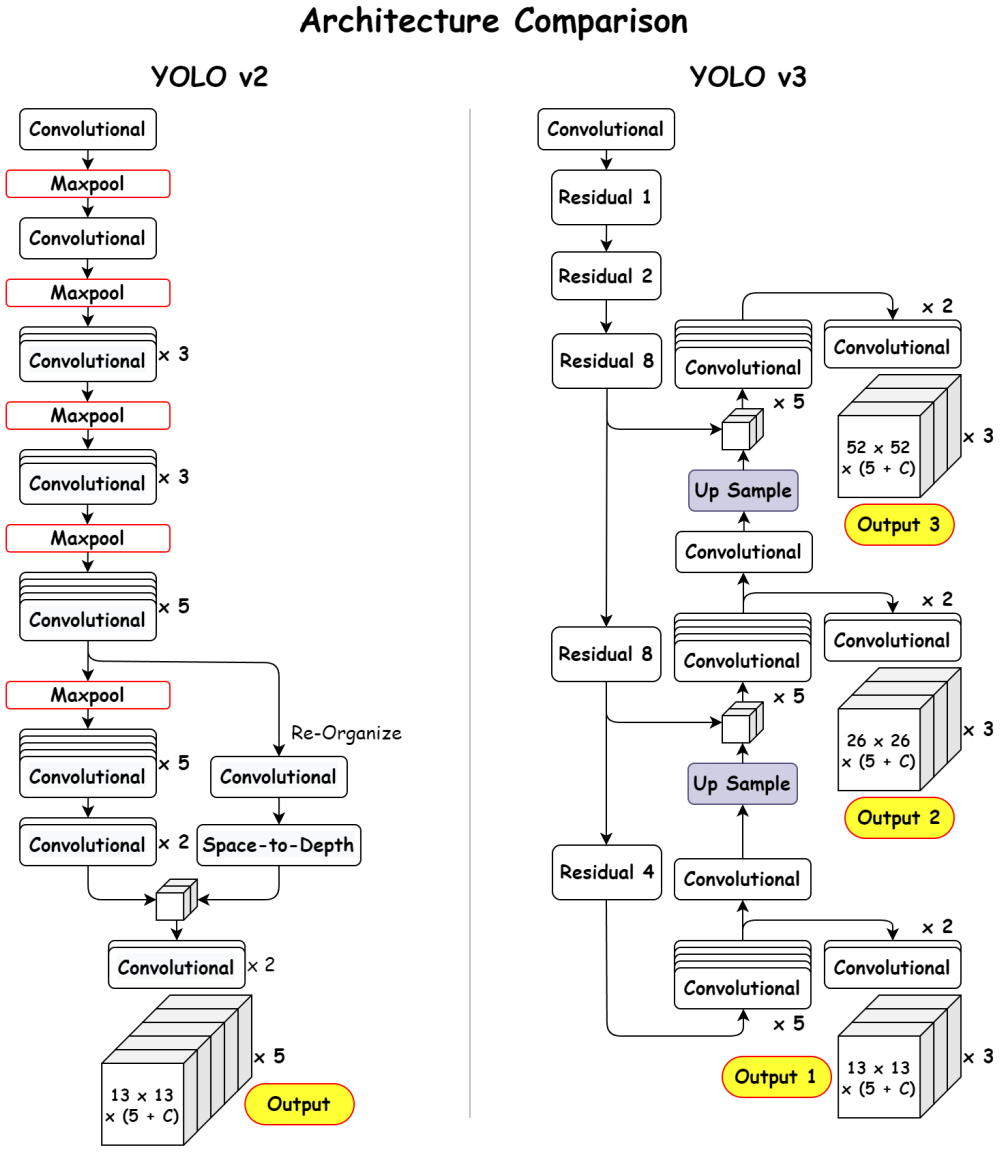


Architecture diagram of YOLOv3

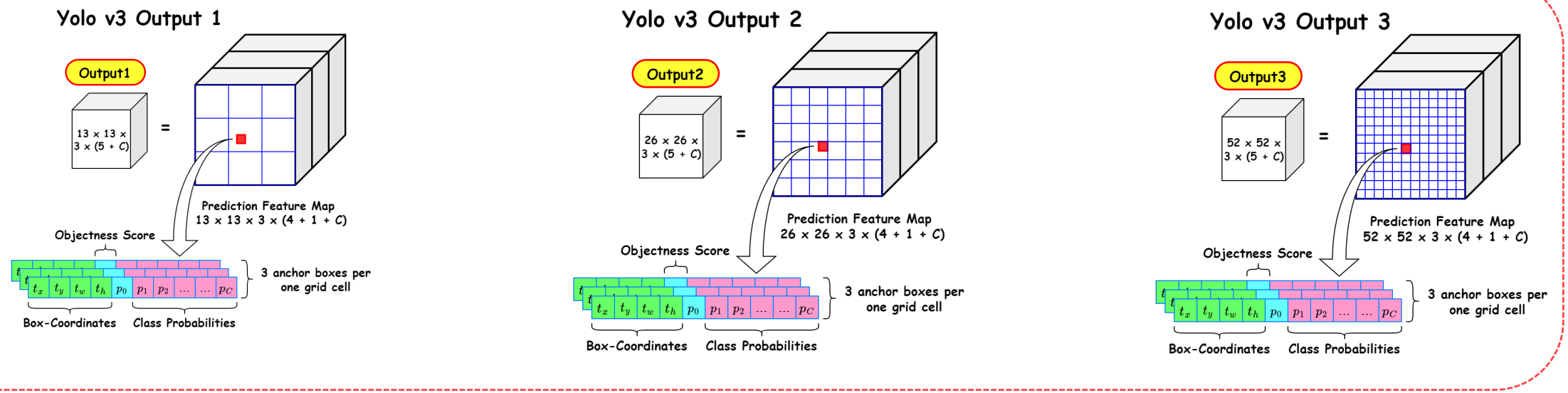
YOLO-v2 vs YOLO-v3: Work flow

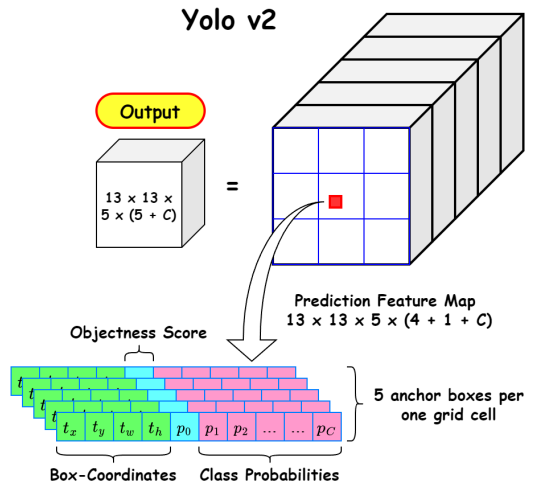


YOLO-v2 vs YOLO-v3: Architecture



YOLO-v2 vs YOLO-v3: Feature Map





Loss function in YOLOv3

Regression loss

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

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

Confidence loss

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

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

Classification loss

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

Square error function

Regression loss

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

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

Confidence loss

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i \right) \log\left(1 - C_i \right) \right] +$$

$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i \right) \log(1 - C_i) \right] +$$

Classification loss

$$\sum_{i=0}^{S^2} I_{ij}^{obj} \sum_{c \in classes} \left[\hat{p}_i(c) \log(p_i(c)) + \left(1 - \hat{p}_i(c) \right) \log(1 - p_i(c)) \right]$$

Entropy loss function

YOLO-v1,v2,v3 Summary

The Original YOLO was the first object detection network to combine the problem of drawing bounding boxes and identifying class labels in one end-to-end differentiable network.

YOLOv2 made a number of iterative improvements on top of YOLO including BatchNorm, higher resolution, and anchor boxes.

YOLOv3 built upon previous models by adding an objectness score to bounding box prediction, added connections to the backbone network layers, and made predictions at three separate levels of granularity to improve performance on smaller objects.

YOLO-v1,v2,v3 Summary

- YoloV1 predicts **98** boxes (7x7 grid cells, 2 boxes per cell @448x448)
- YoloV2 predicts **845** boxes (13x13 grid cells, 5 anchor boxes per grid cell @416x416)
- YoloV3 predicts **10647** boxes (13x13, 26x26,52x52 grid cells, 3 anchor boxes per grid cell @416x416)
- YoloV3 predicts more than **10x** the number of boxes predicted by YoloV2

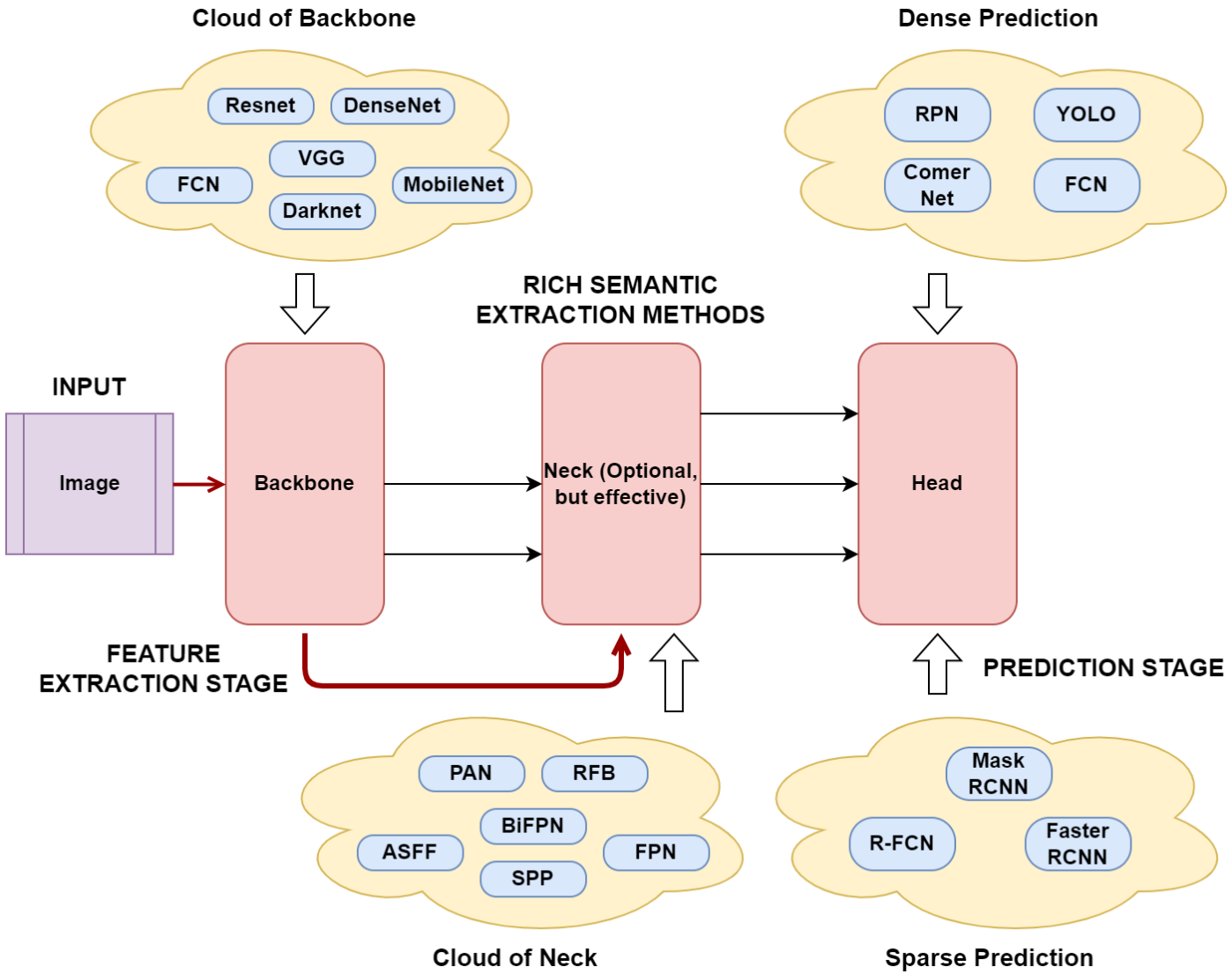
Example

```
from ultralytics import YOLO

# Load a model
model = YOLO('yolov3.pt')  # load an official model

# Predict with the model
results = model('/content/bus.jpg')

show_img(results)
```

Outline

- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

Object Detection: Main Flow



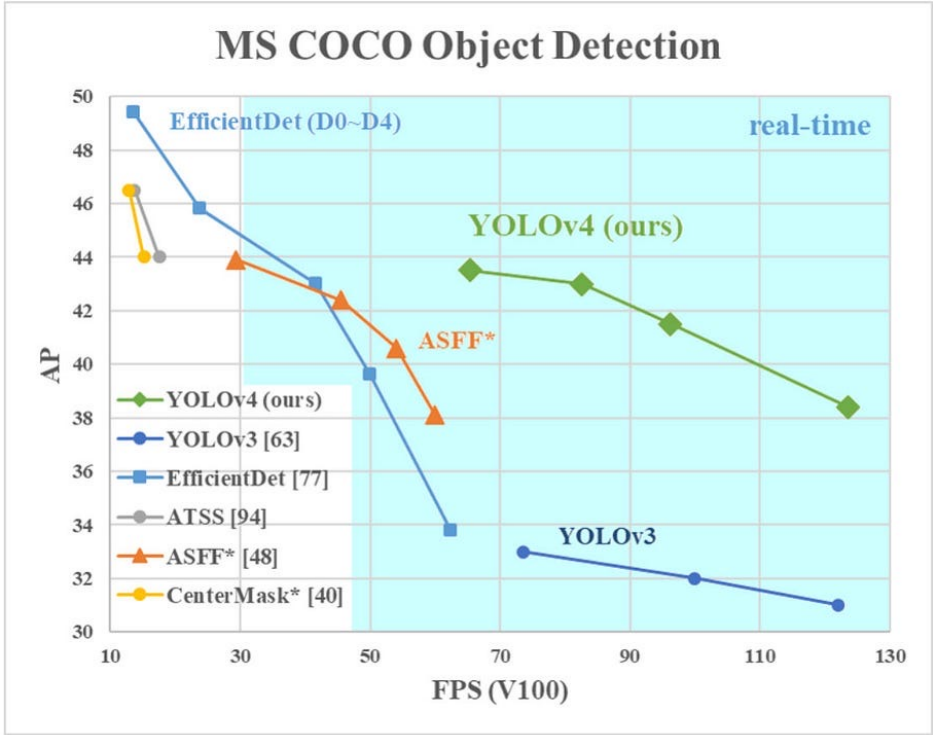
YOLO-v4

Kiến trúc của YOLOv3 nâng cấp lên YOLOv4 được tóm tắt như sau:

- Backbone: DarkNet53 --> CSPDarkNet53, áp dụng thêm DropBlock
- Neck: FPN --> SPP + PAN
- Head: Giữ nguyên từ YOLOv3

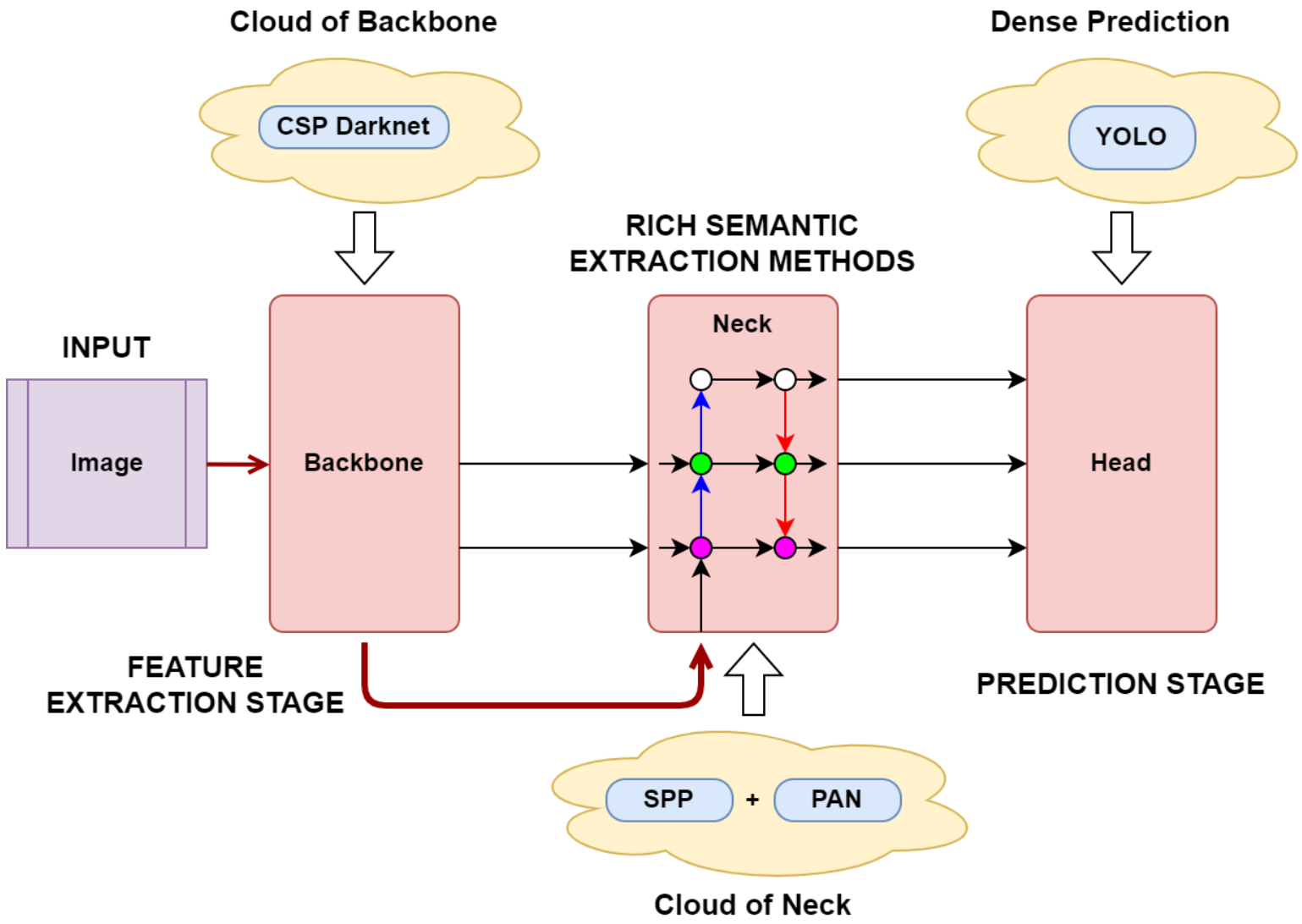
Các thay đổi khác trong YOLOv4 bao gồm:

- Data Augmentation: Mosaic Augmentation, Self-Adversarial Training
- Loss function: Regression Loss từ squared loss --> CIoU Loss
- Anchor Box: 1 anchor --> nhiều anchors
- Label Smoothing
- Loại bỏ Grid Sensitivity
- Cosine Learning Rate schedule



The YOLO v4 released in April 2020, but this release is not from the YOLO first author. *In Feb 2020, Joseph Redmon announced he was leaving the field of computer vision due to concerns regarding the possible negative impact of his works.*

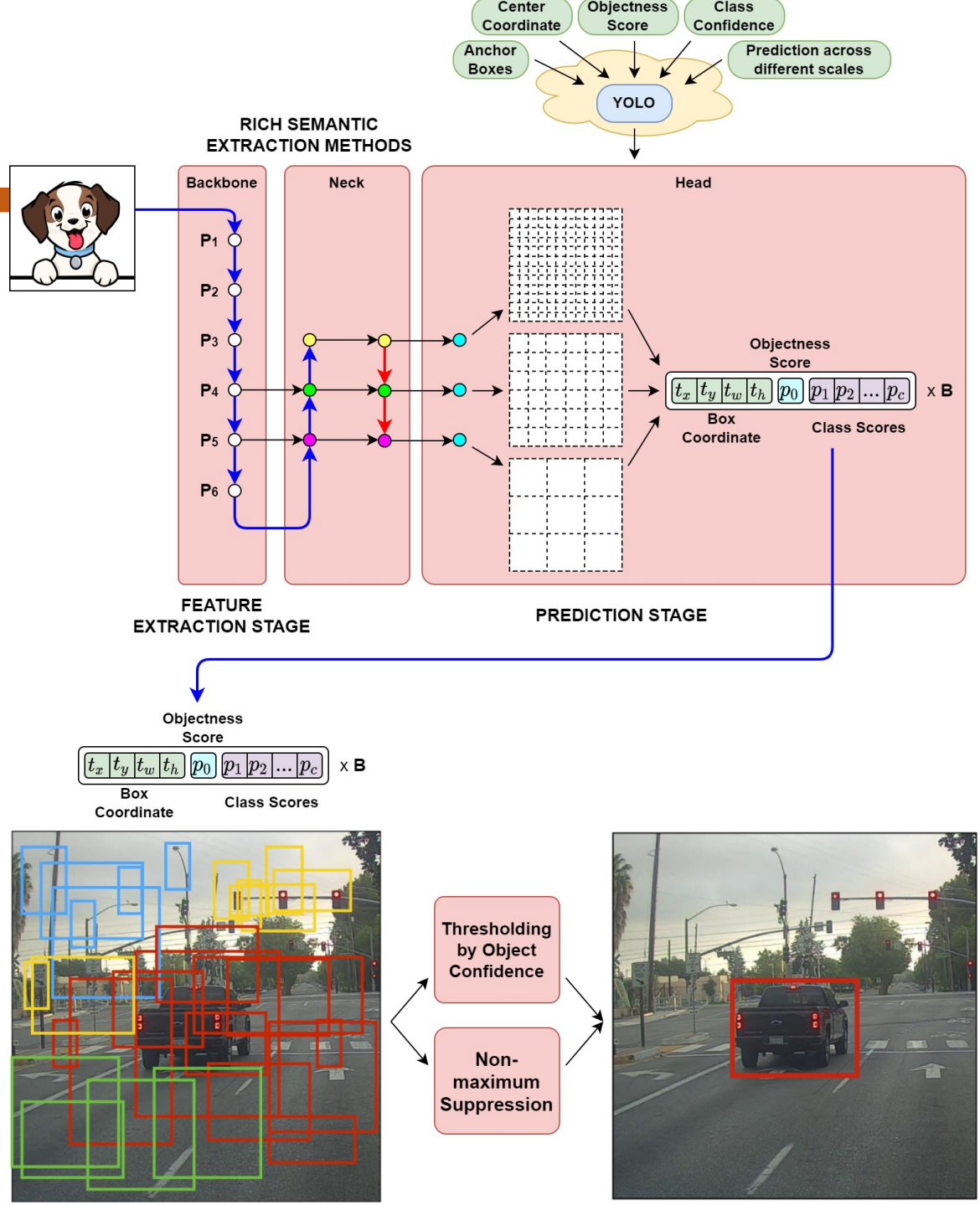
Simple YOLO v4 Architecture



AI VIETNAM
All-in-One Course

YOLOv4 Architecture

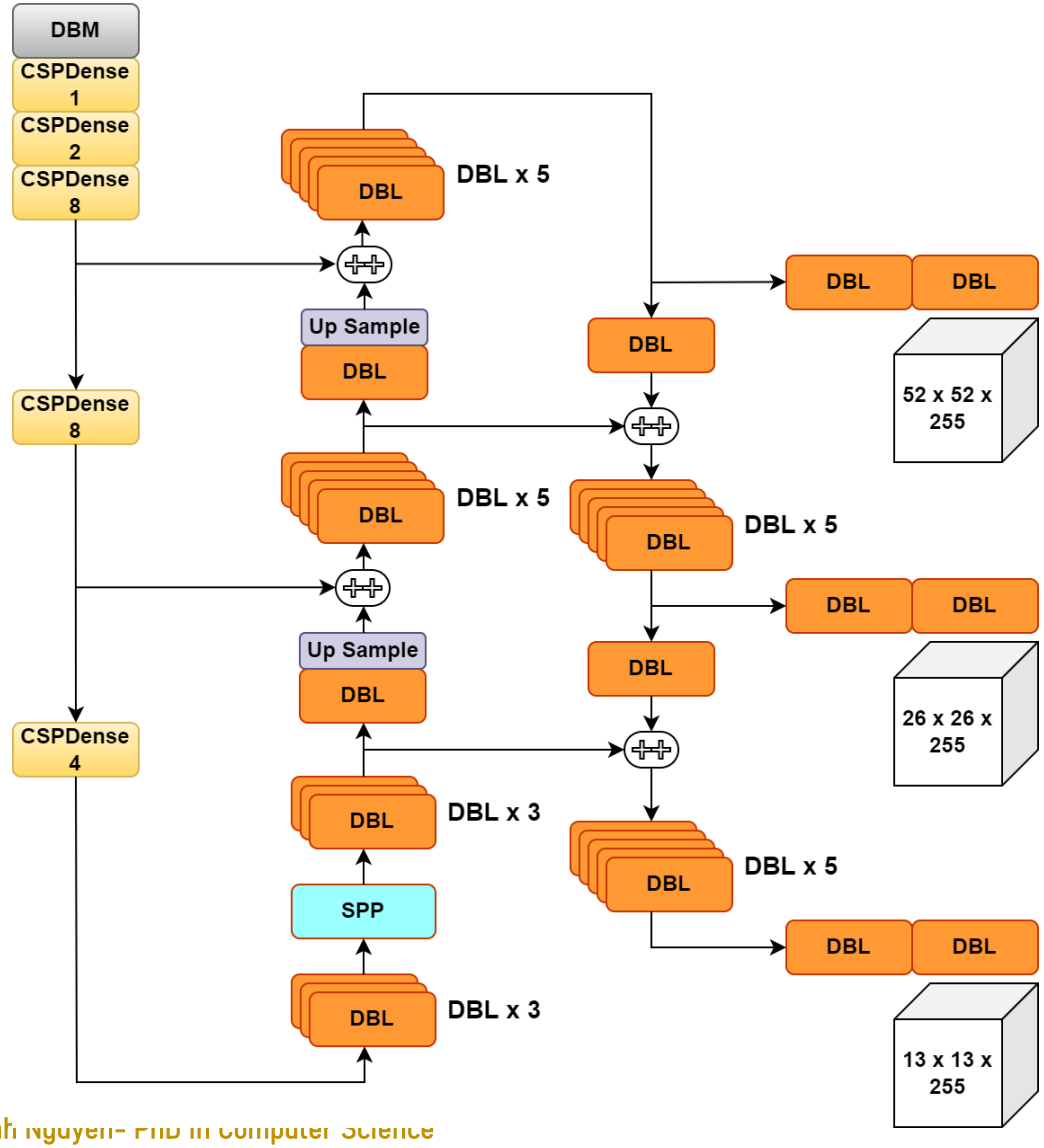
YOLOv4 is quite similar to YOLOv3. The only difference is that CSPDarknet is used in Backbone and SPP and PANet are used in Neck, and everything else is almost the same. If you go into it very densely, you can see a little difference, but overall, if you understand the above three things correctly, you can understand YOLOv4.

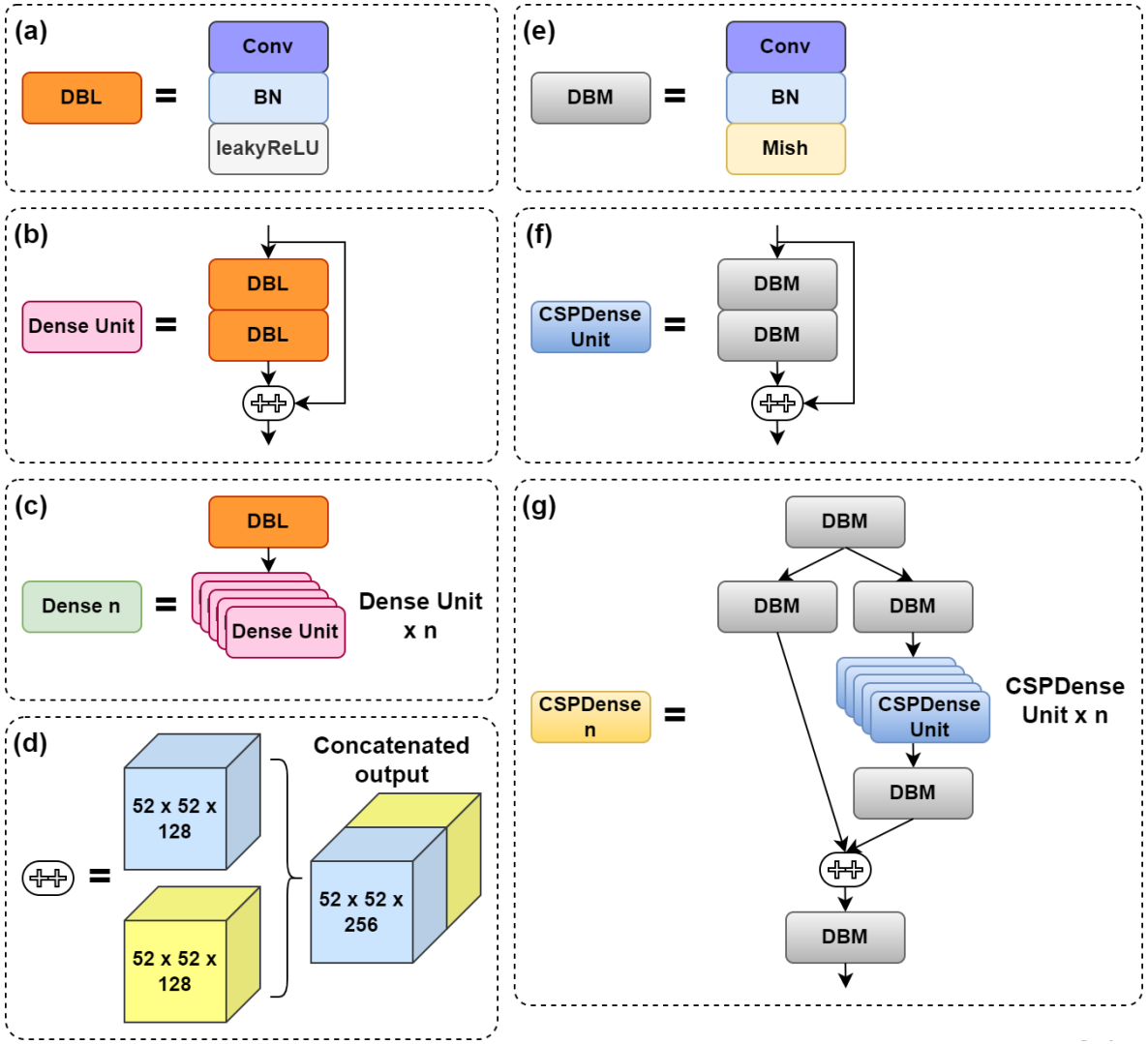


https://wikidocs.net/167833

AI VIETNAM
All-in-One Course

Simple YOLO v4 Architecture





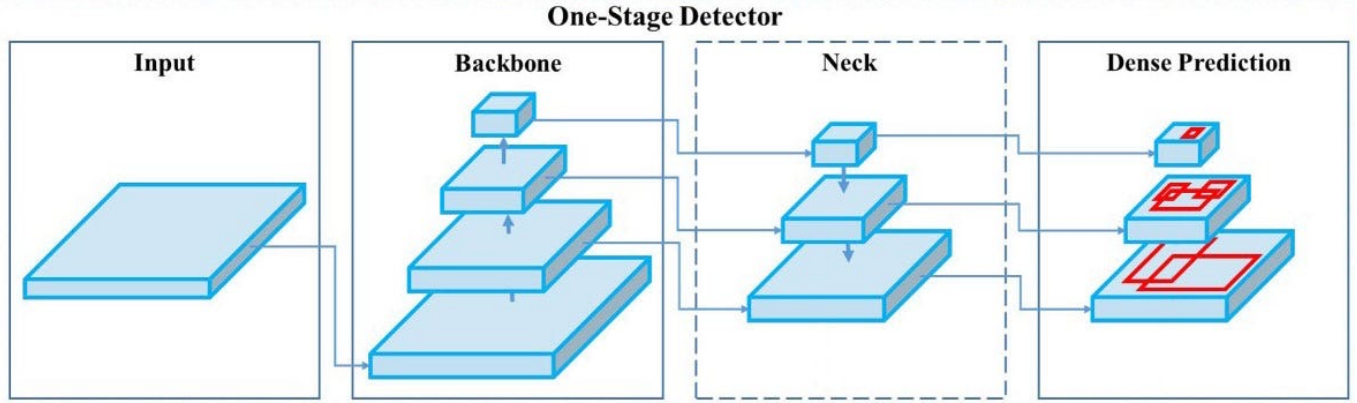
YOLO-v4 Architecture

Bag of freebies (Bof)

Improvements can be made in the training process (like data augmentation, class imbalance, cost function, soft labeling etc…) to advance accuracy. These improvements have no impact on inference speed and called "bag of freebies"

Bag of specials (BoS)

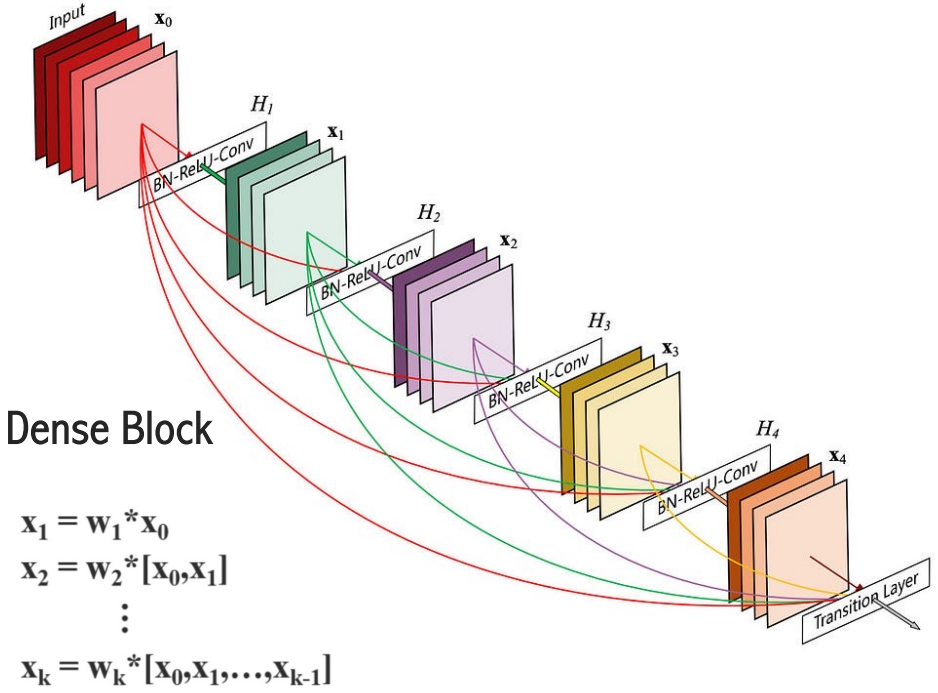
There are "bag of specials" which impacts the inference time slightly with a good return in performance. These improvements include the increase of the receptive field, the use of attention, feature integration like skip-connections & FPN, and post-processing like non-maximum suppression



YOLO-v4: Backbone

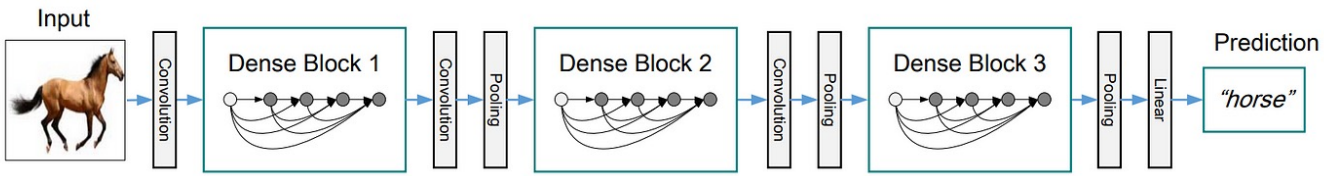
Dense Block and DenseNet

To improve accuracy, we can design a deeper network to extend the receptive field and to increase model complexity. And to ease the training difficulty, skip-connections can be applied.



Dense Block

$$x_1 = w_1 * x_0$$
$$x_2 = w_2 * [x_0, x_1]$$
$$\vdots$$
$$x_k = w_k * [x_0, x_1, \ldots, x_{k-1}]$$



DenseNet

YOLO-v4: Backbone

Dense Block and DenseNet

To improve accuracy, we can design a deeper network to extend the receptive field and to increase model complexity. And to ease the training difficulty, skip-connections can be applied.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

YOLO-v4: Backbone

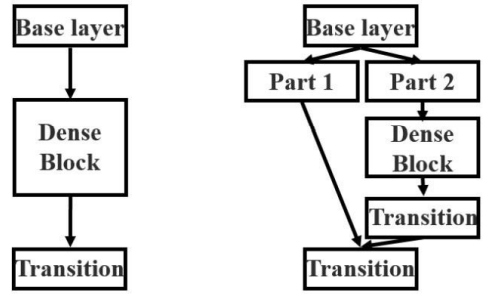
Cross-Stage-Partial-connections (CSP)

8.7.3. Transition Layers

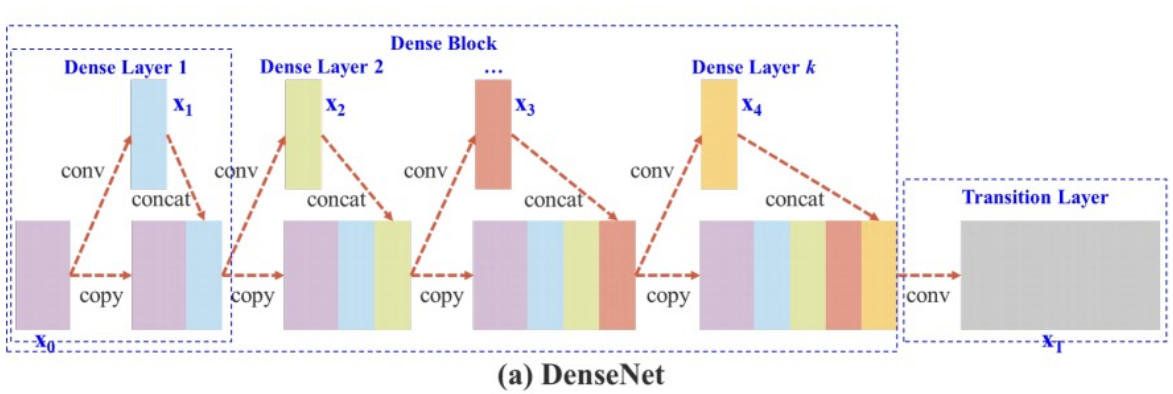
Since each dense block will increase the number of channels, adding too many of them will lead to an excessively complex model. A *transition layer* is used to control the complexity of the model. It reduces the number of channels by using a 1×1 convolution. Moreover, it halves the height and width via average pooling with a stride of 2.

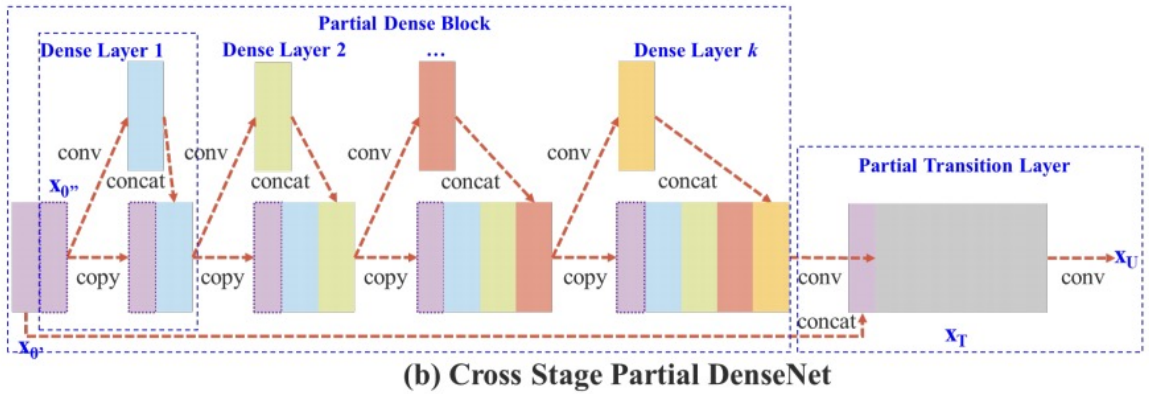
PYTORCH MXNET JAX TENSORFLOW

```
def transition_block(num_channels):
    return nn.Sequential(
        nn.LazyBatchNorm2d(), nn.ReLU(),
        nn.LazyConv2d(num_channels, kernel_size=1),
        nn.AvgPool2d(kernel_size=2, stride=2))
```



CSPNet separates the input feature maps of the DenseBlock into two parts. The first part x_0' bypasses the DenseBlock and becomes part of the input to the next transition layer. The second part x_0'' will go thought the Dense block as below.

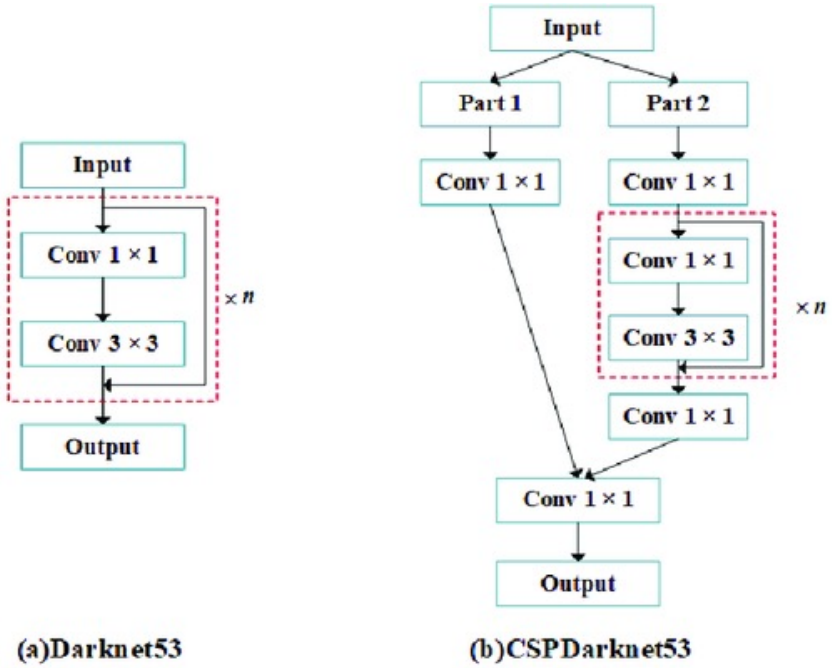




YOLO-v4:Backbone

CSPDarknet53

YOLOv4 utilizes the CSP connections above with the Darknet-53 below as the backbone in feature extraction.



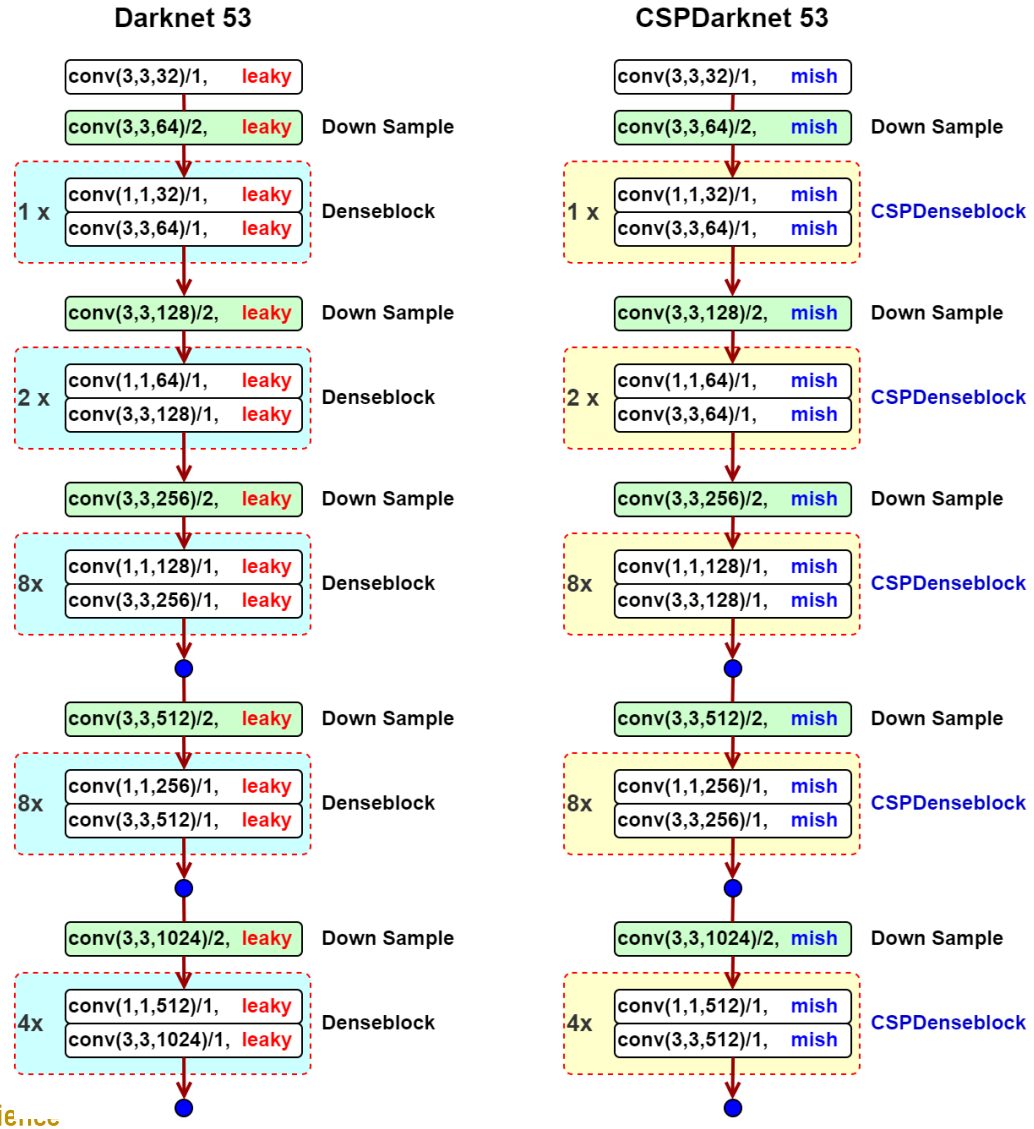
(a)Darknet53 (b)CSPDarknet53

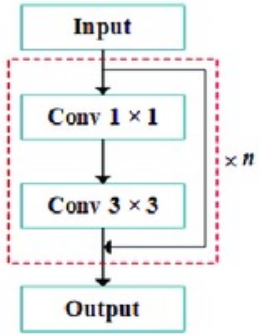
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 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.**

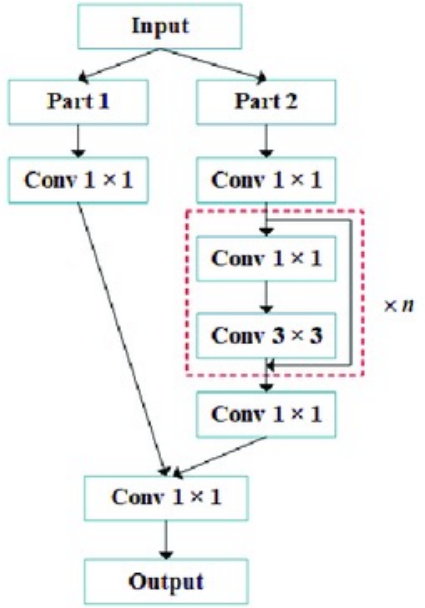
YOLO-v4:Backbone

CSPDarknet53





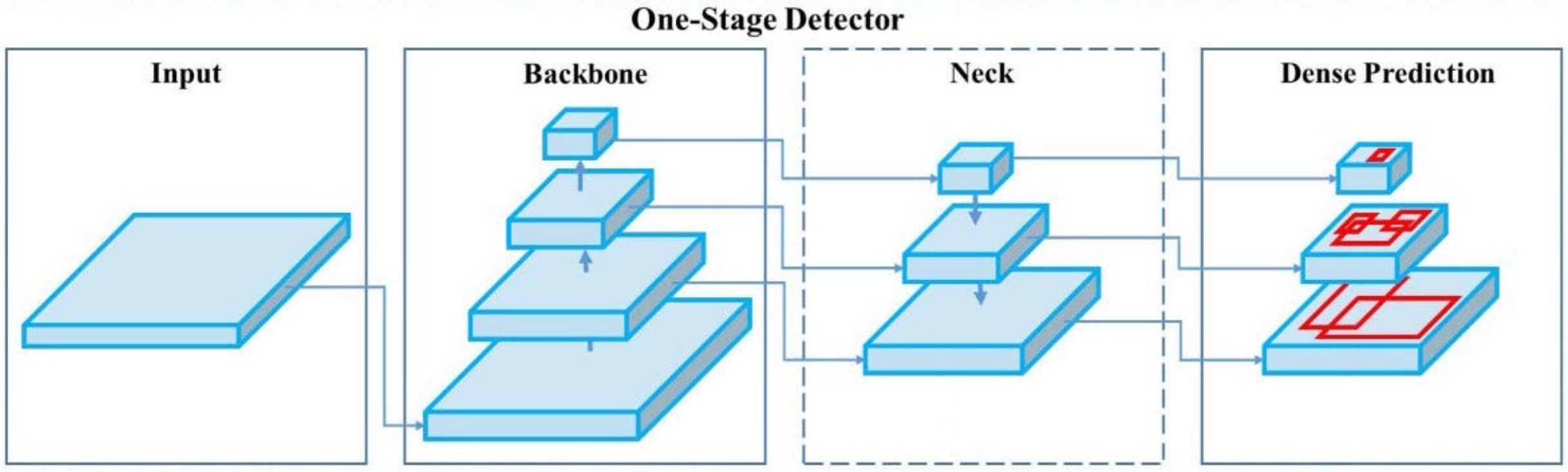
(a)Darknet53



(b)CSPDarknet53

YOLO-v4:Neck

Object detectors composed of a backbone in feature extraction and a head (the rightmost block below) for object detection. And to detect objects at different scales, a hierarchy structure is produced with the head probing feature maps at different spatial resolutions.

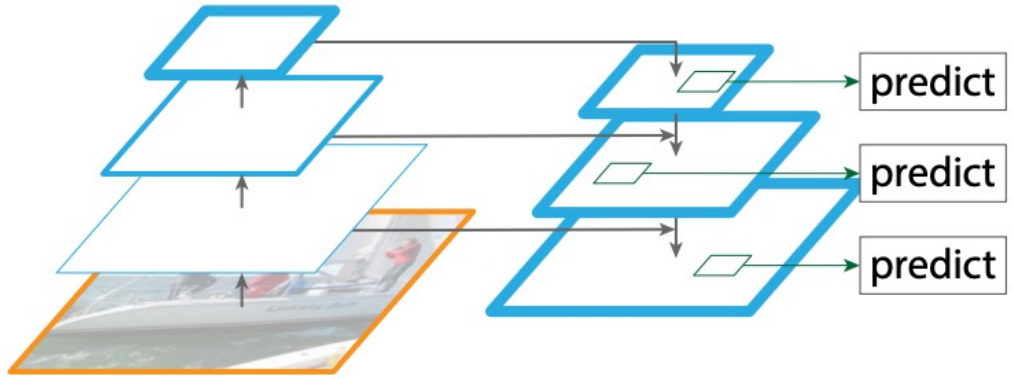


Therefore, the head's input will contain spatial rich information from the bottom-up stream and the semantic rich information from the top-down stream. This part of the system is called a neck. Let's get more details in its design.

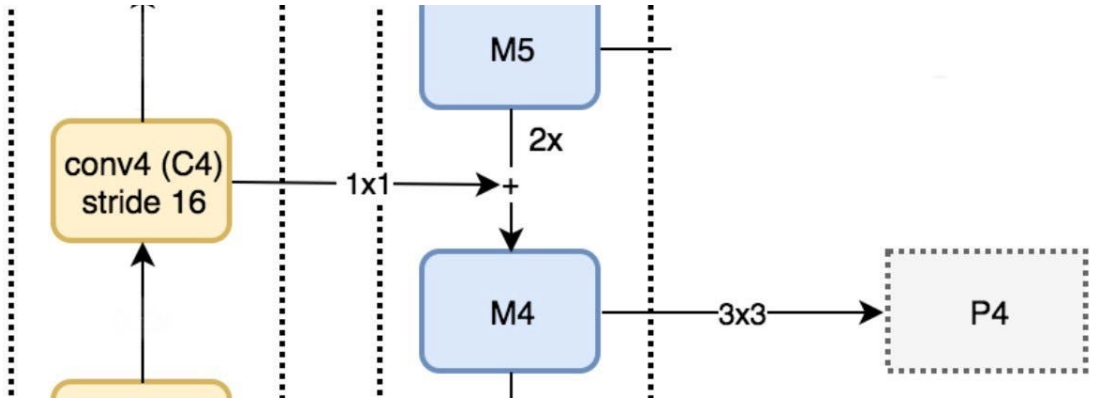
YOLO-v4:Neck

Feature Pyramid Networks (FPN)

YOLOv3 adapts a similar approach as FPN in making object detection predictions at different scale levels.



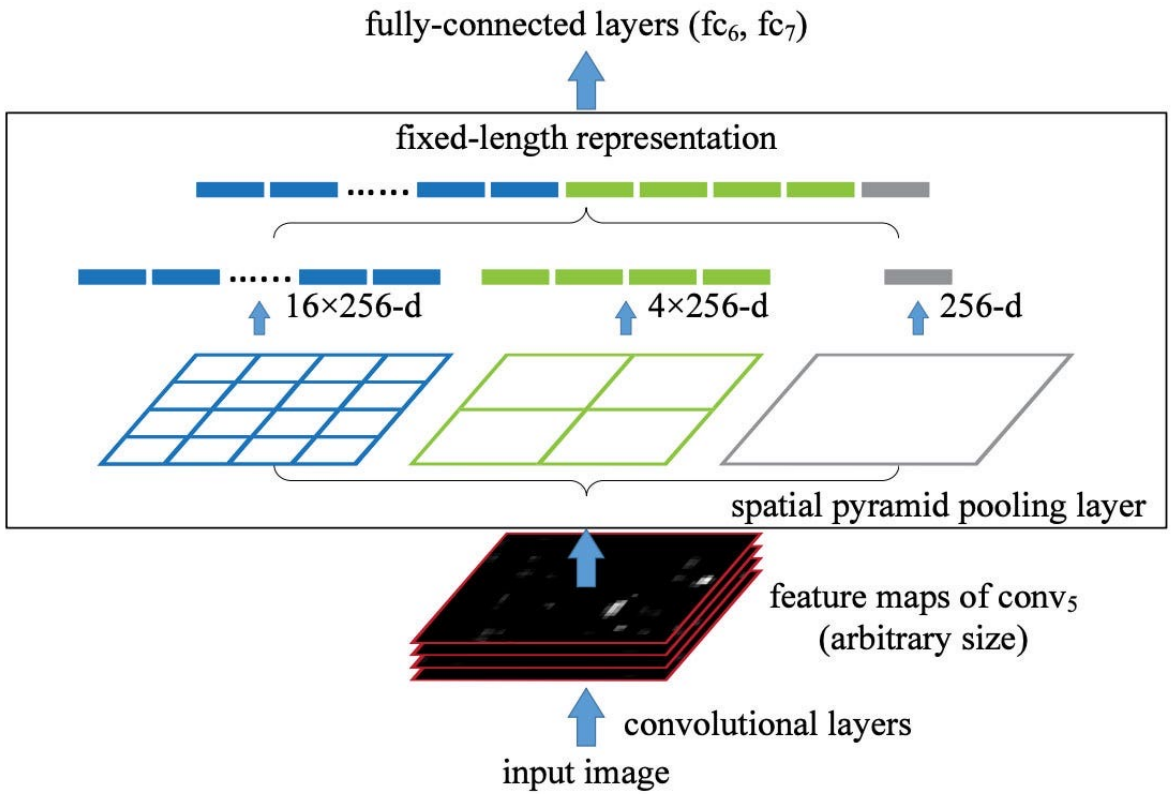
In making predictions for a particular scale, FPN upsamples (2×) the previous top-down stream and add it with the neighboring layer of the bottom-up stream (see the diagram below). The result is passed into a 3×3 convolution filter to reduce upsampling artifacts and create the feature maps P4 below for the head.



YOLO-v4:Neck

SPP (spatial pyramid pooling layer)

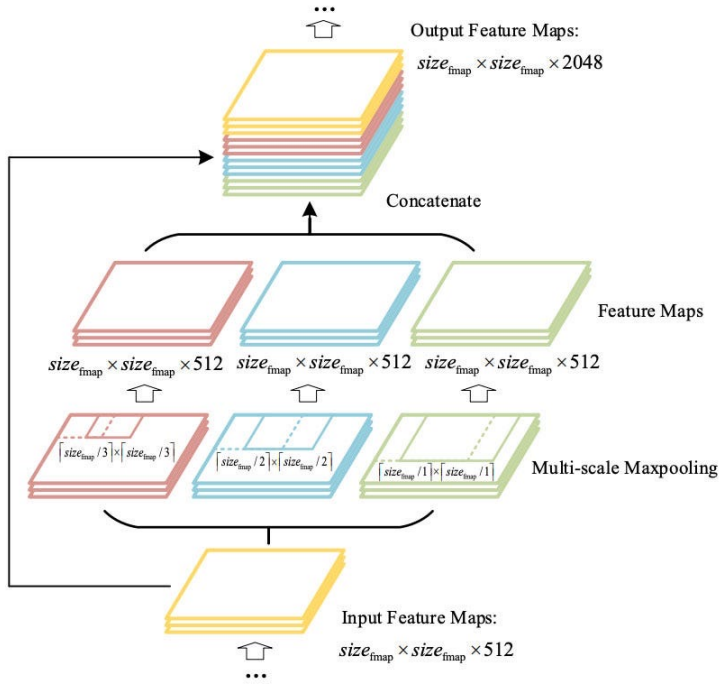
SPP applies a slightly different strategy in detecting objects of different scales. It replaces the last pooling layer (after the last convolutional layer) with a spatial pyramid pooling layer.



YOLO-v4:Neck

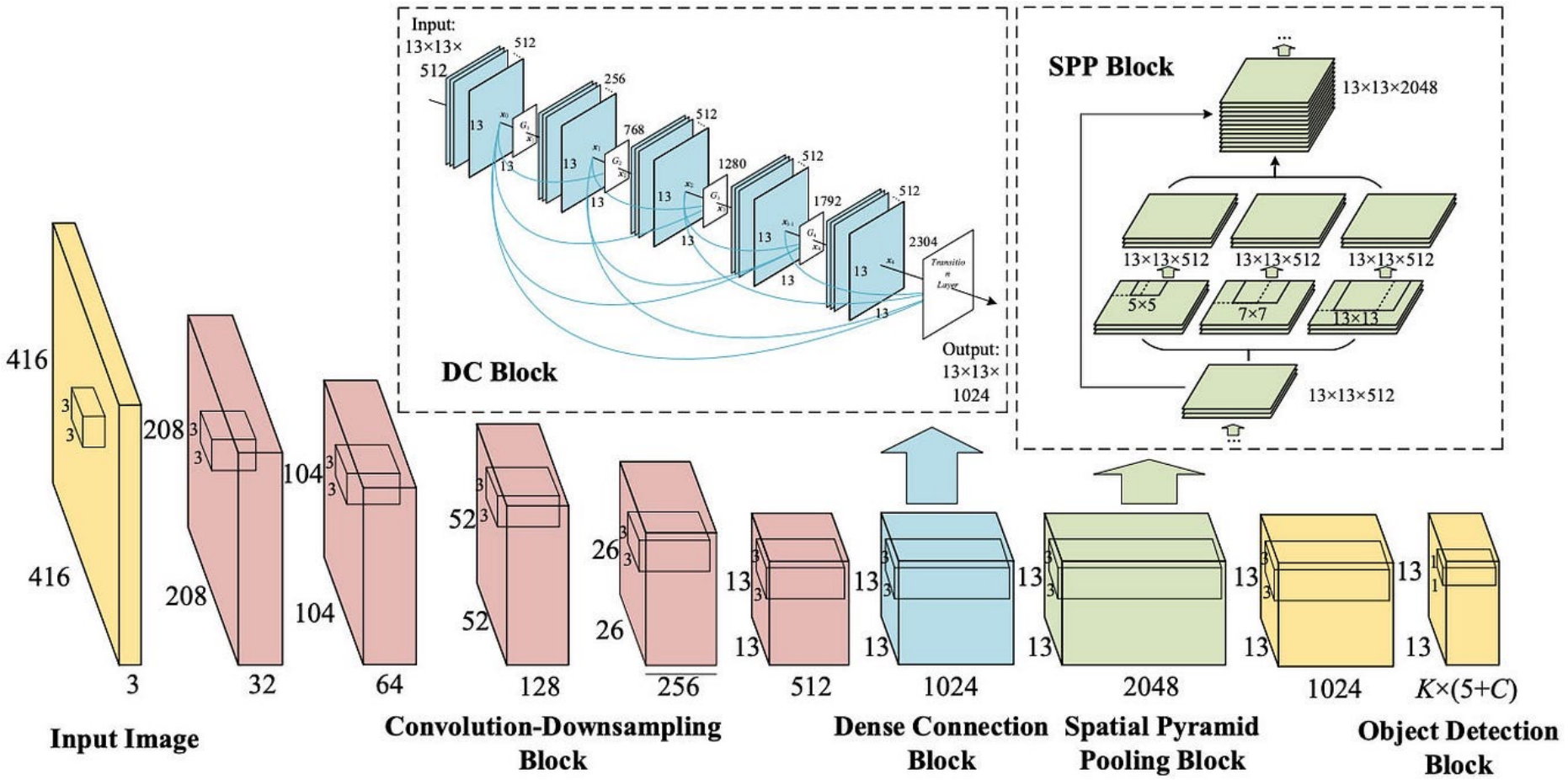
YOLO with SPP

In YOLO, the SPP is modified to retain the output spatial dimension. A maximu pool is applied to a sliding kernel of size say, 1×1, 5×5, 9×9, 13×13. Th spatial dimension is preserved. The features maps from different kernel size are then concatenated together as output.



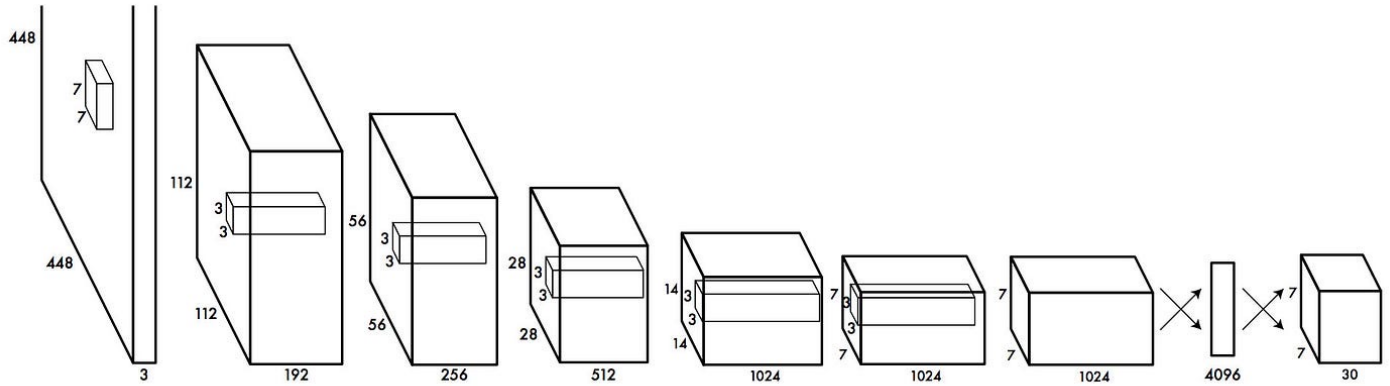
YOLO-v4:Neck

YOLO with SPP

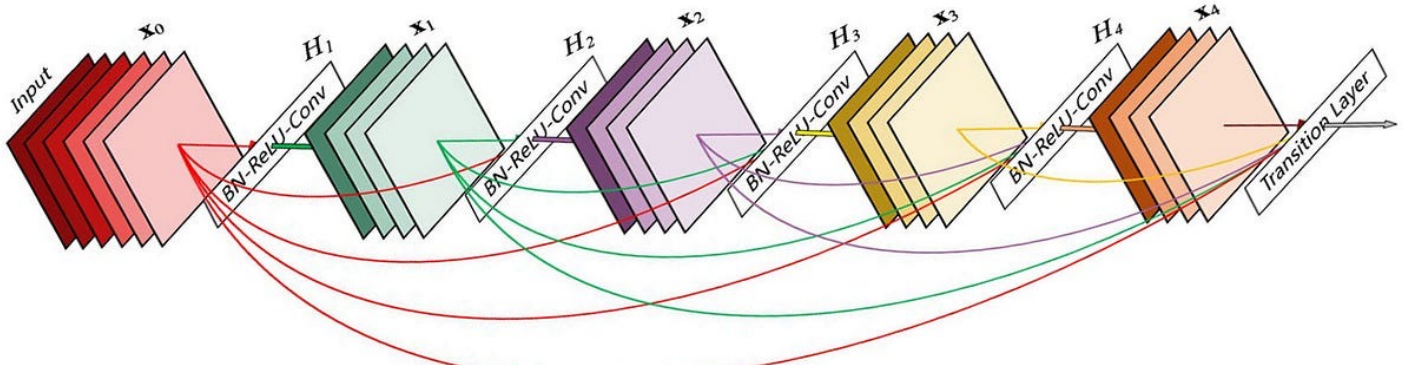


YOLO-v4:Neck

Path Aggregation Network (PAN)



Early Deep Learning Architecture

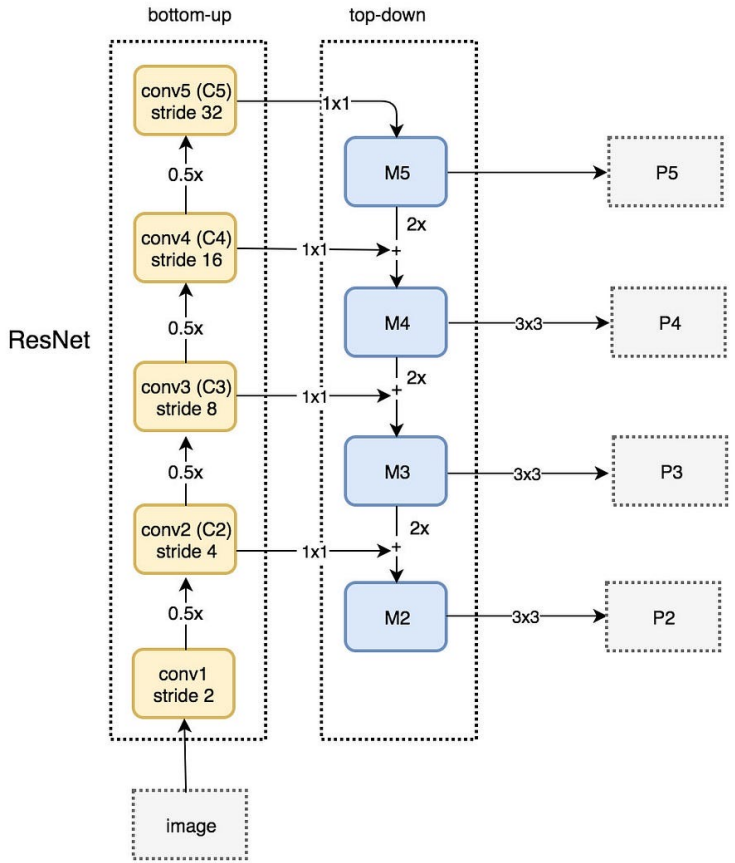


Later Deep Learning Architecture

YOLO-v4:Neck

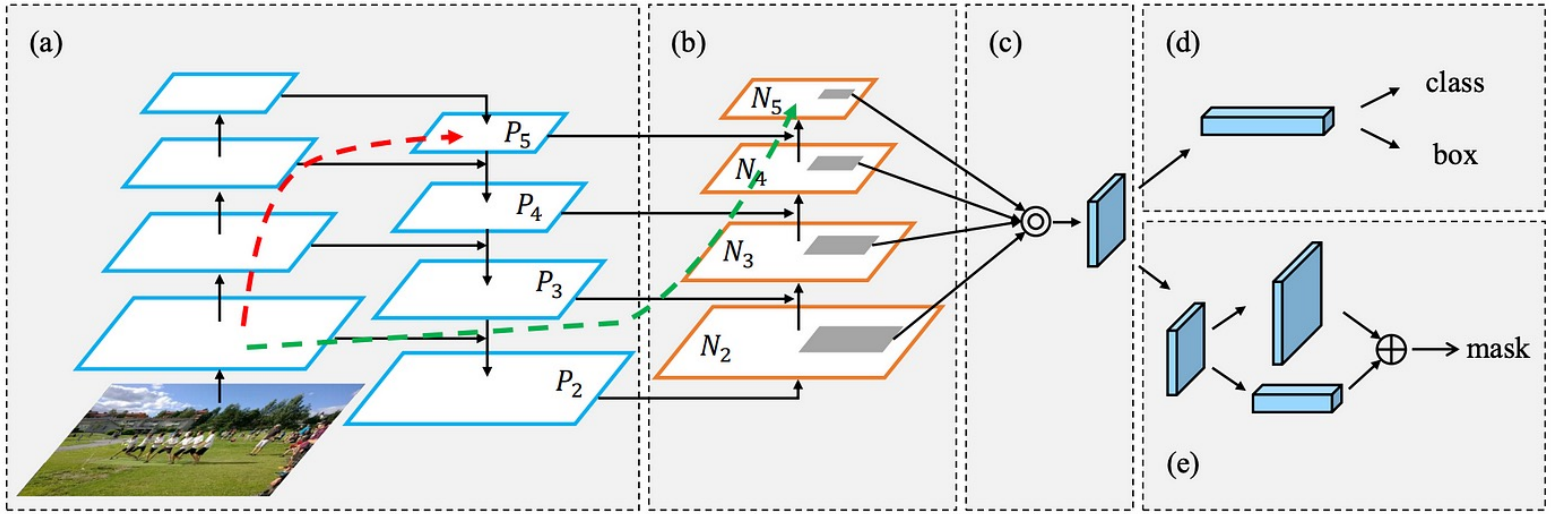
Path Aggregation Network (PAN)

In FPN, information is combined from neighboring layers in the bottom-up and top-down stream.

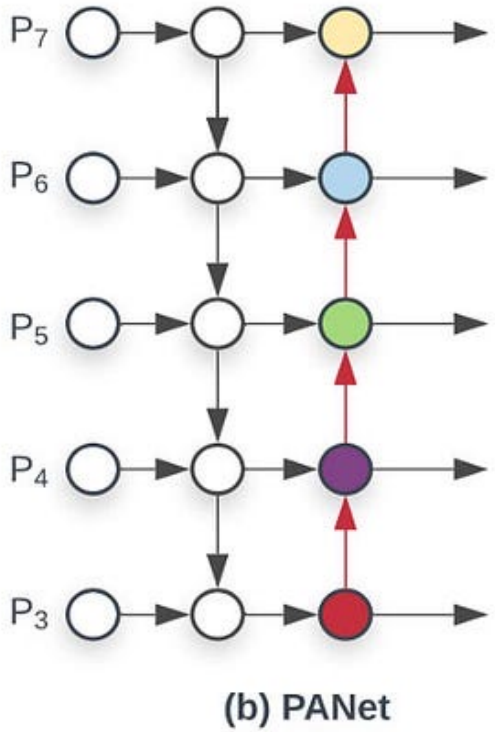


YOLO-v4:Neck

Path Aggregation Network (PAN)

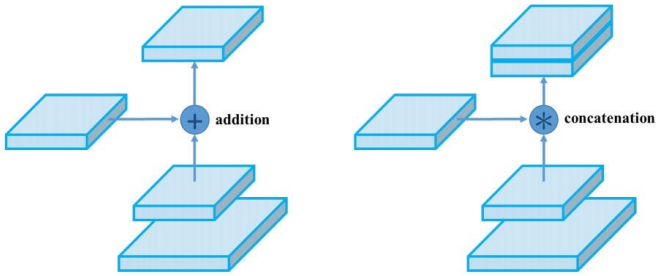


As a side note, the neck design can be visualized as the following



(b) PANet

However, instead of adding neighbor layers together, features maps are concatenated together in YOLOv4.

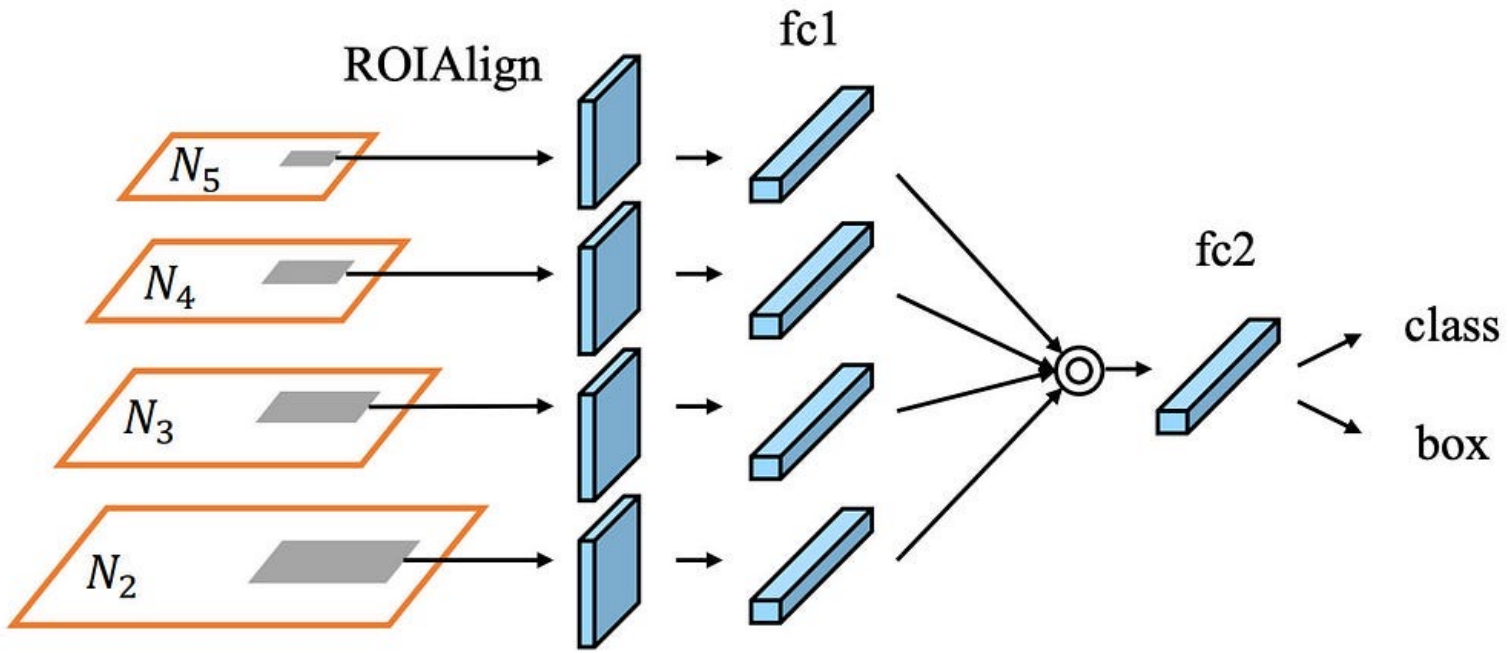


(a) PAN [49] (a) Our modified PAN

YOLO-v4:Neck

Path Aggregation Network (PAN)

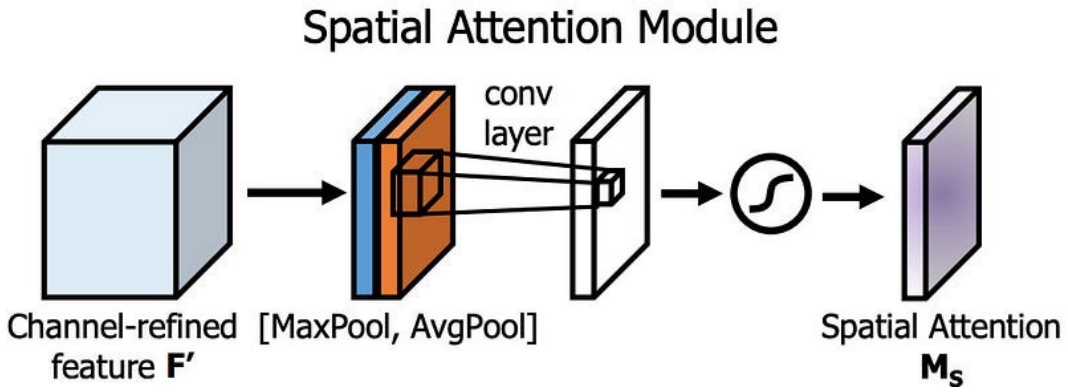
In FPN, objects are detected separately and independently at different scale levels. This may produce duplicated predictions and not utilize information from other feature maps. PAN fuses the information together from all layers first using element-wise max operation.

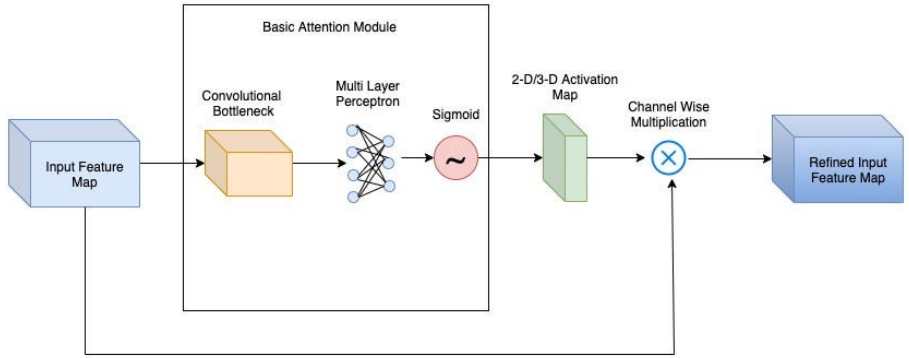


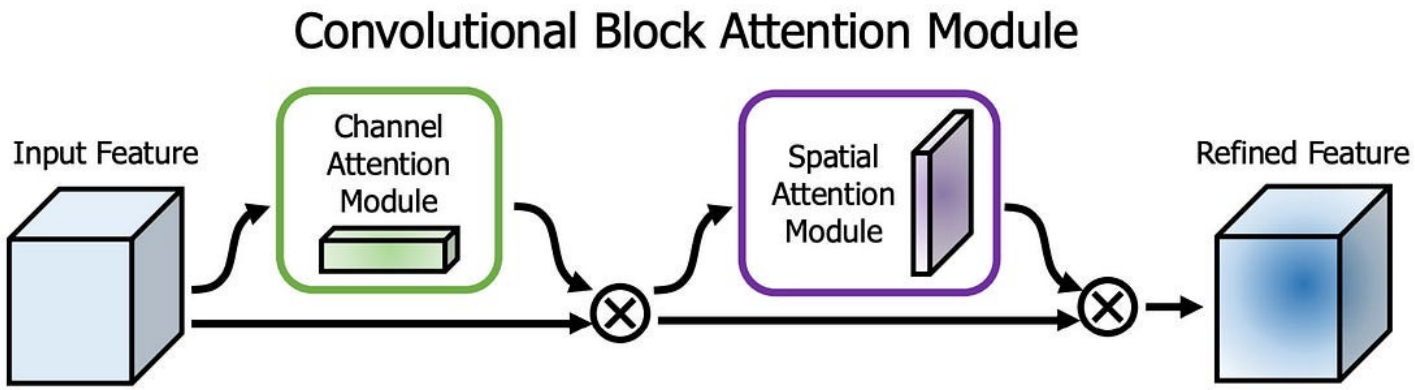
YOLO-v4:Neck

Spatial Attention Module (SAM)

Attention has widely adopted in DL designs. In SAM, maximum pool and average pool are applied separately to input feature maps to create two sets of feature maps. The results are feed into a convolution layer followed by a sigmoid function to create spatial attention.



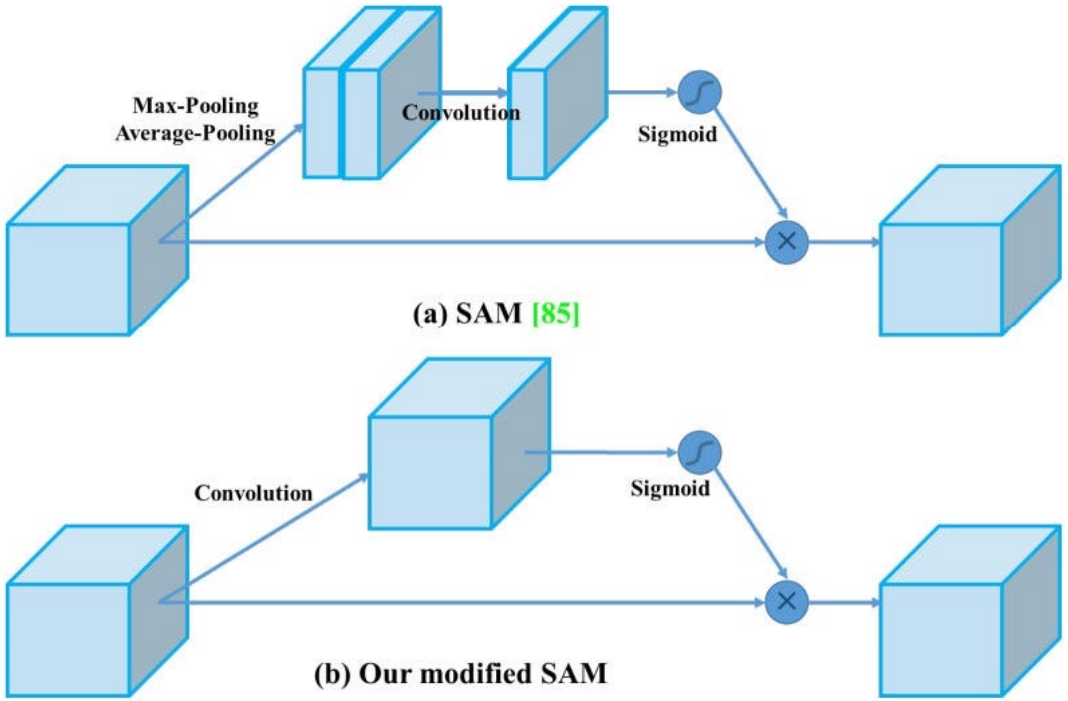




YOLO-v4:Neck

Spatial Attention Module (SAM)

In YOLOv4, a modified SAM is used without applying the maximum and average pooling.



YOLO-v4: Bag of Freebies (BoF) for backbone

CutMix data augmentation

	ResNet-50	Mixup [48]	Cutout [3]	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	**78.6** **(+2.3)**
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	**47.3** **(+1.0)**
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	**76.7** **(+1.1)**

YOLO-v4: Bag of Freebies (BoF) for backbone

Mosaic data augmentation

Mosaic is a data augmentation method that combines 4 training images into one for training (instead of 2 in CutMix)

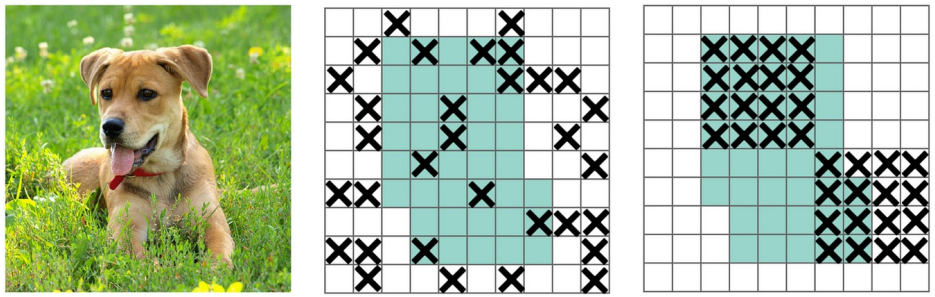
YOLO-v4: Bag of Freebies (BoF) for backbone

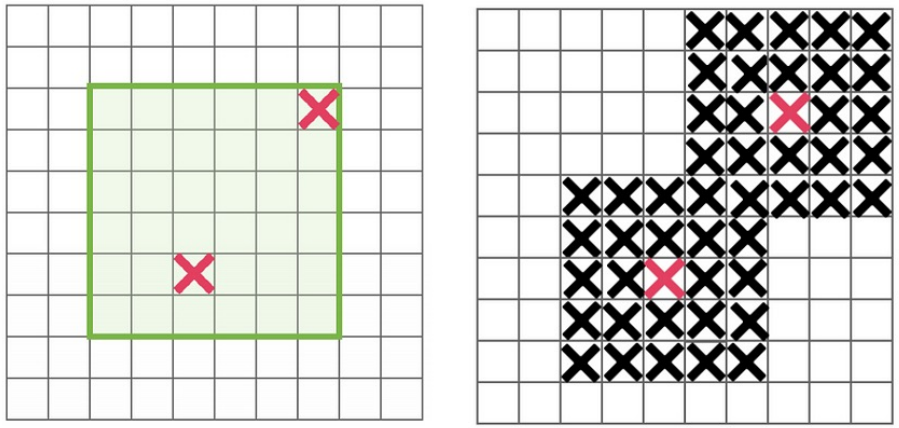
DropBlock regularization

Instead of dropping individual pixels, a block of block_size × block_size of pixels is dropped.

Algorithm 1 DropBlock

Input: output activations of a layer (A), $block\_size$, γ, $mode$
if $mode$ == *Inference* **then**
    return A
end if
Randomly sample mask M: $M_{i,j} \sim Bernoulli(\gamma)$
For each zero position $M_{i,j}$, create a spatial square mask with the center being $M_{i,j}$, the width, height being $block\_size$ and set all the values of M in the square to be zero (see Figure 2).
Apply the mask: $A = A \times M$
Normalize the features: $A = A \times \textbf{count}(M)/\textbf{count\_ones}(M)$





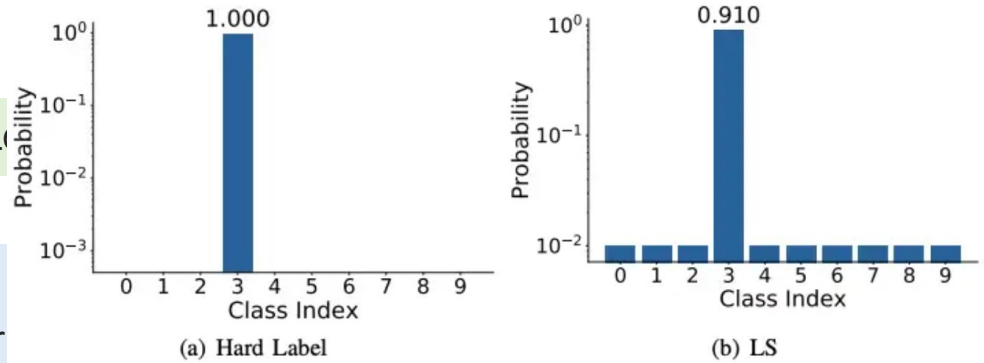
(a) (b)

YOLO-v4: Bag of Freebies (BoF) for backbone

Class label smoothing

Suppose we have $K = 3$ classes, and our label belongs to the 1st class. L

If we do not use label smoothing, the label vector is the one-hot encoded make a ≫ b and a ≫ c. For example, applying softmax to the logit vector decimal places.



source: Delving Deep into Label Smoothing

If we use label smoothing with $\alpha = 0.1$, the smoothed label vector ≈ [0.9333, 0.0333, 0.0333]. The logit vector [3.3322, 0, 0] approximates the smoothed label vector to 4 decimal places after softmax, and it has a smaller gap. This is why we call label smoothing a regularization technique as it restrains the largest logit from becoming much bigger than the rest.

$$y\_ls = (1 - \alpha) * y\_hot + \alpha / K$$

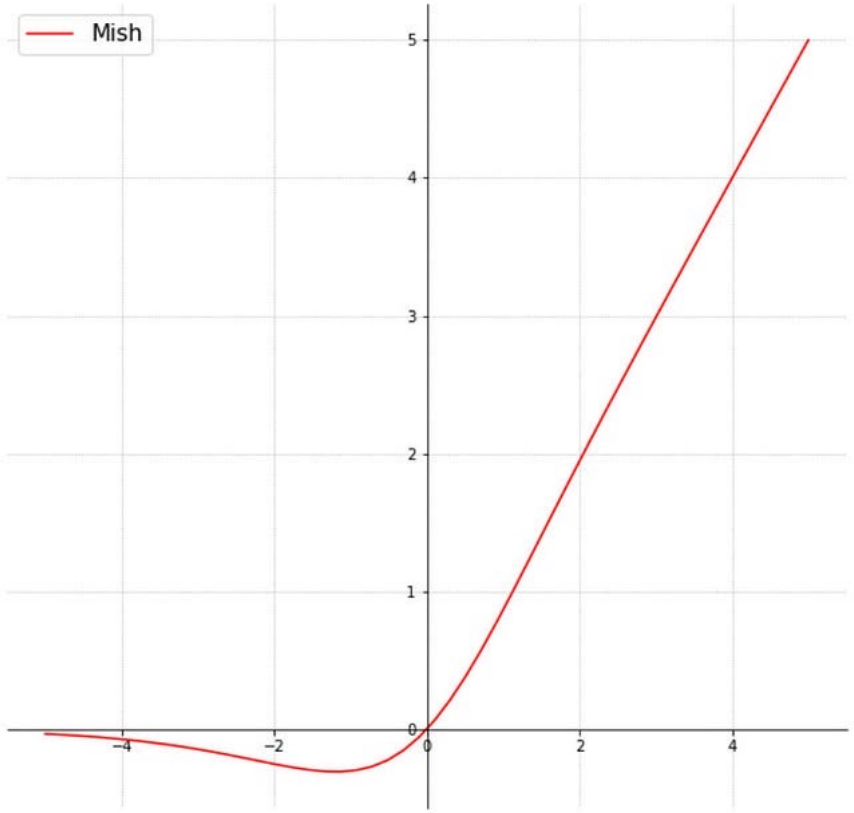
where K is the number of label classes, and α is a hyperparameter that determines the amount of smoothing. If $\alpha = 0$, we obtain the original one-hot encoded y_hot. If $\alpha = 1$, we get the uniform distribution.

YOLO-v4: Bag of Specials (BoS) for backbone

Mish activation

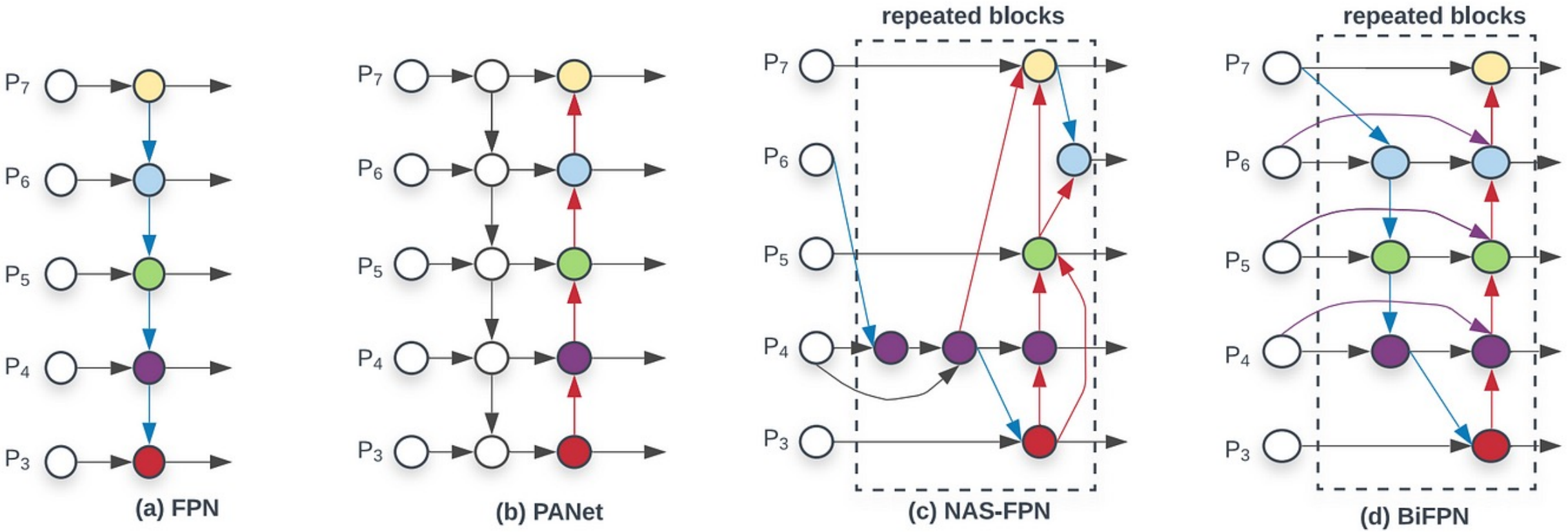
$f(x) = x \cdot tanh(\varsigma(x))$

where, $\varsigma(x) = \ln(1 + e^x)$ is the softplus activation function.



YOLO-v4: Bag of Specials (BoS) for backbone

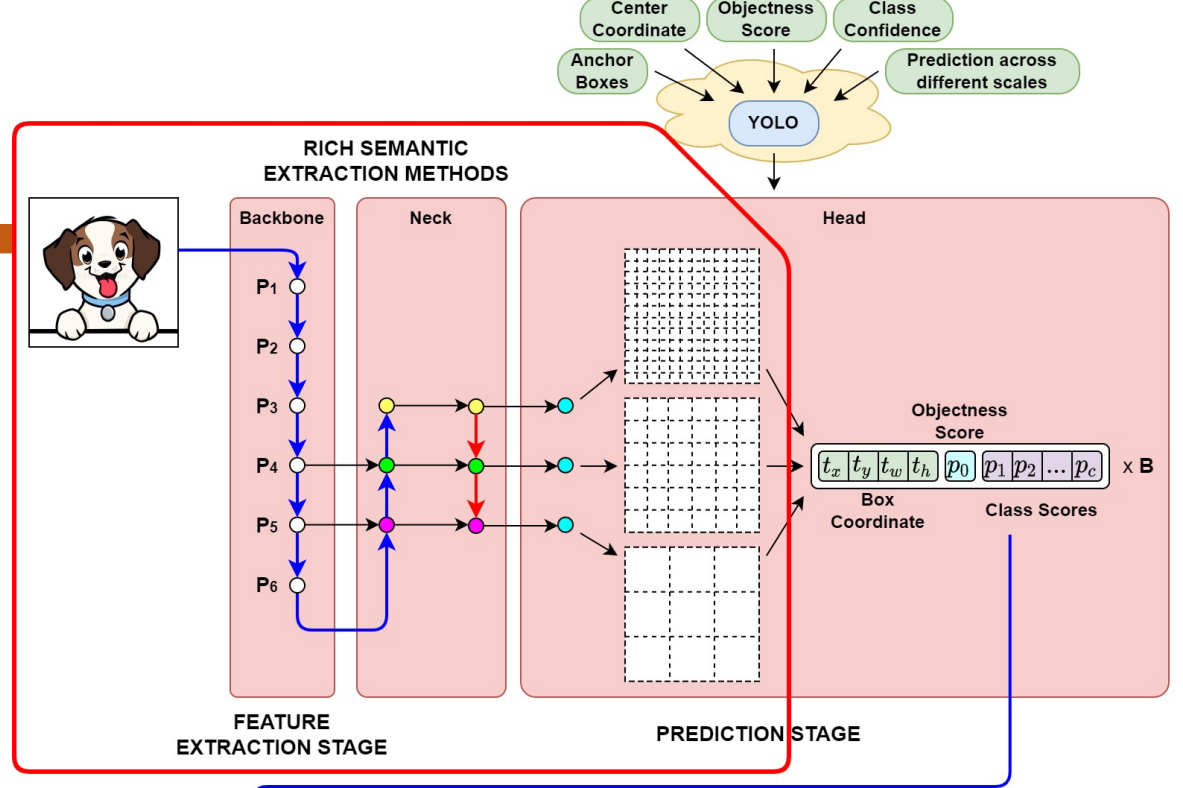
Multi-input weighted residual connections (MiWRC)

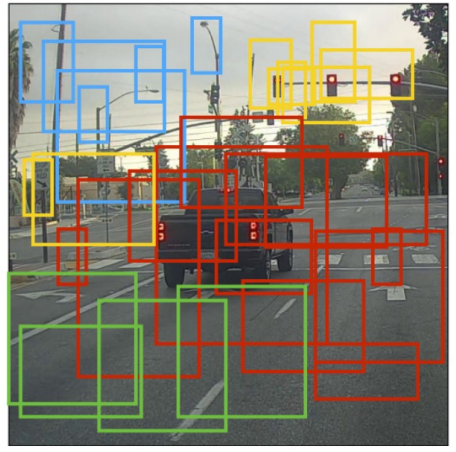


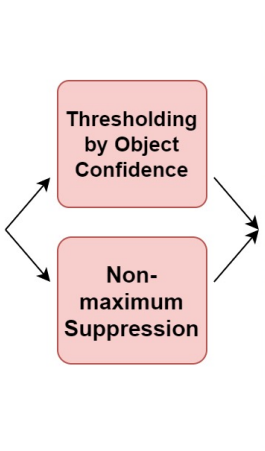
AI VIETNAM
All-in-One Course

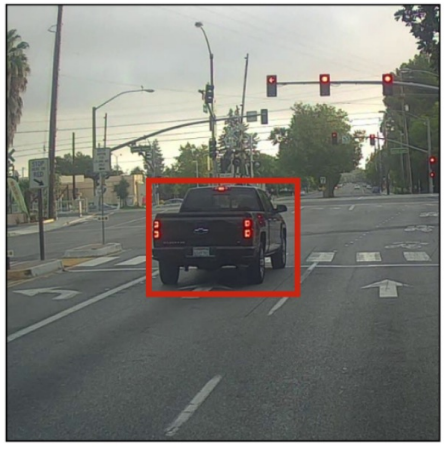
YOLOv4-Head

Many YOLOv4 articles cover the YOLOv4 Neck section. YOLOv4 Head is the same as YOLOv3. If you look at this in the Wokflow part, only the red inside below is covered,









YOLO-v4 Summary

- **Input**: Image, Patches, Image Pyramid
- **Backbones**: VGG16 [68], ResNet-50 [26], SpineNet [12], EfficientNet-B0/B7 [75], CSPResNeXt50 [81], CSPDarknet53 [81]
- **Neck**:
  - **Additional blocks**: SPP [25], ASPP [5], RFB [47], SAM [85]
  - **Path-aggregation blocks**: FPN [44], PAN [49], NAS-FPN [17], Fully-connected FPN, BiFPN [77], ASFF [48], SFAM [98]
- **Heads:**:
  - **Dense Prediction (one-stage)**:
    - RPN [64], SSD [50], YOLO [61], RetinaNet [45] (anchor based)
    - CornerNet [37], CenterNet [13], MatrixNet [60], FCOS [78] (anchor free)
  - **Sparse Prediction (two-stage)**:
    - Faster R-CNN [64], R-FCN [9], Mask R-CNN [23] (anchor based)
    - RepPoints [87] (anchor free)
- **Activations**: ReLU, leaky-ReLU, parametric-ReLU, ReLU6, SELU, Swish, or Mish
- **Bounding box regression loss**: MSE, IoU, GIoU, CIoU, DIoU
- **Data augmentation**: CutOut, MixUp, CutMix
- **Regularization method**: DropOut, DropPath [36], Spatial DropOut [79], or DropBlock
- **Normalization of the network activations by their mean and variance**: Batch Normalization (BN) [32], Cross-GPU Batch Normalization (CGBN or SyncBN) [93], Filter Response Normalization (FRN) [70], or Cross-Iteration Batch Normalization (CBN) [89]
- **Skip-connections**: Residual connections, Weighted residual connections, Multi-input weighted residual connections, or Cross stage partial connections (CSP)

Example

```
!cd /content
!rm -fr darknet
!git clone https://github.com/AlexeyAB/darknet/
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/g' Makefile
!sed -i 's/GPU=0/GPU=1/g' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/g' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/g' Makefile
!apt update
!apt-get install libopencv-dev

!make &> compile.log
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights
```

```
def predictImage(imageDir):
  os.system("cd /content/darknet && ./darknet detect cfg/yolov4.cfg yolov4.weights {}".fo
  image = cv2.imread("/content/darknet/predictions.jpg")
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)
  fig = plt.gcf()
  fig.set_size_inches(15, 8)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()

predictImage("/content/bus.jpg")
```

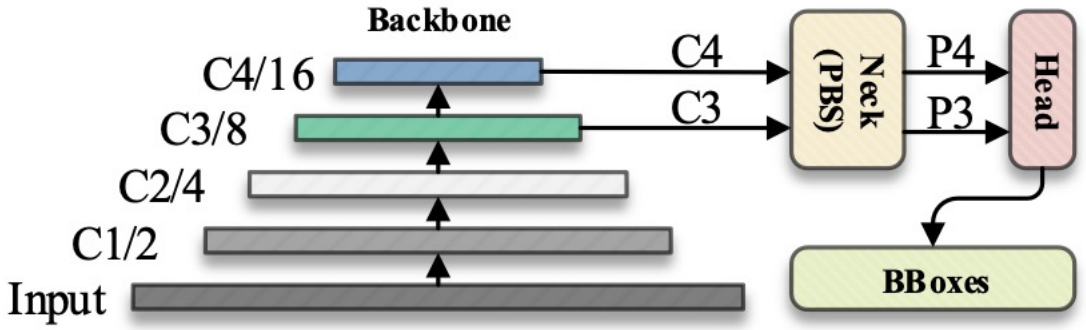
Outline

- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

YOLO-v5

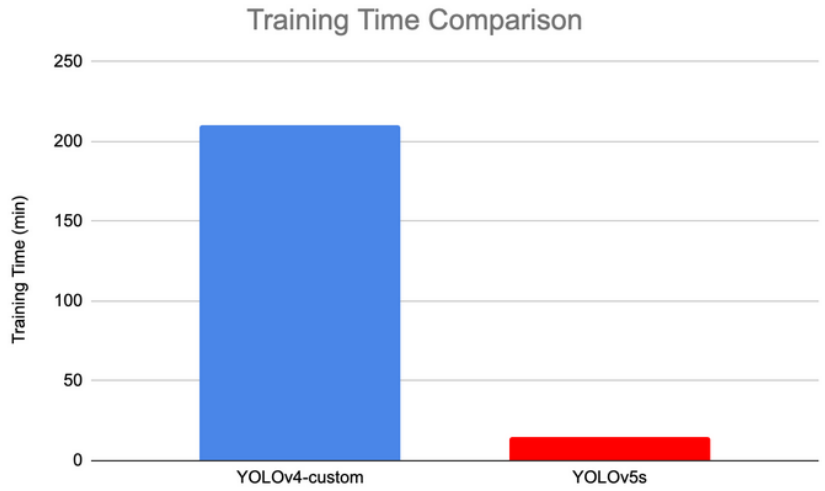
History of YOLO

- **Yolov1** (Jun 8th, 2015): You Only Look Once: Unified, Real-Time Object Detection
- **Yolov2** (Dec 25th, 2016): YOLO9000:Better, Faster, Stronger
- **Yolov3** (Apr 8th, 2018): YOLOv3: An Incremental Improvement
- **Yolov4** (Apr 23rd, 2020): YOLOv4: Optimal Speed and Accuracy of Object Detection
- **Yolov5** (May 18th, 2020): Github repo (there is no paper as of Aug 1st, 2021)



Advantages & Disadvantages of Yolo v5

- It is about 88% smaller than YOLOv4 (27 MB vs 244 MB)
- It is about 180% faster than YOLOv4 (140 FPS vs 50 FPS)
- It is roughly as accurate as YOLOv4 on the same task (0.895 mAP vs 0.892 mAP)
- But the main problem is that for YOLOv5 there is no official paper was released like other YOLO versions. Also, YOLO v5 is still under development and we receive frequent updates from **ultralytics**, developers may update some settings in the future.



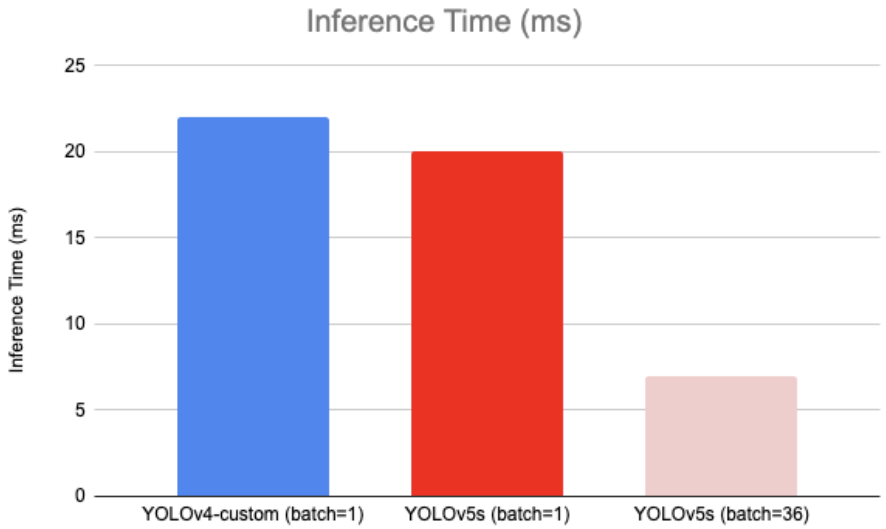
YOLO-v5

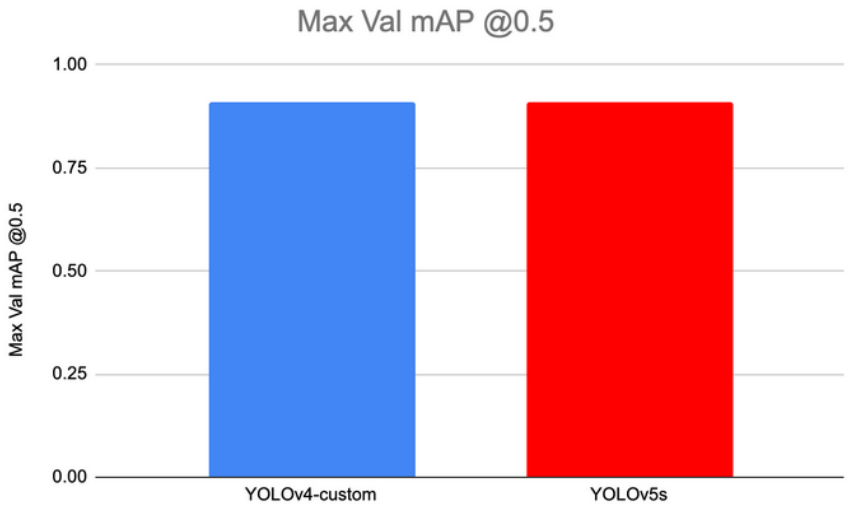
History of YOLO

- **Yolov1** (Jun 8th, 2015): You Only Look Once: Unified, Real-Time Object Detection
- **Yolov2** (Dec 25th, 2016): YOLO9000:Better, Faster, Stronger
- **Yolov3** (Apr 8th, 2018): YOLOv3: An Incremental Improvement
- **Yolov4** (Apr 23rd, 2020): YOLOv4: Optimal Speed and Accuracy of Object Detection
- **Yolov5** (May 18th, 2020): Github repo (there is no paper as of Aug 1st, 2021)

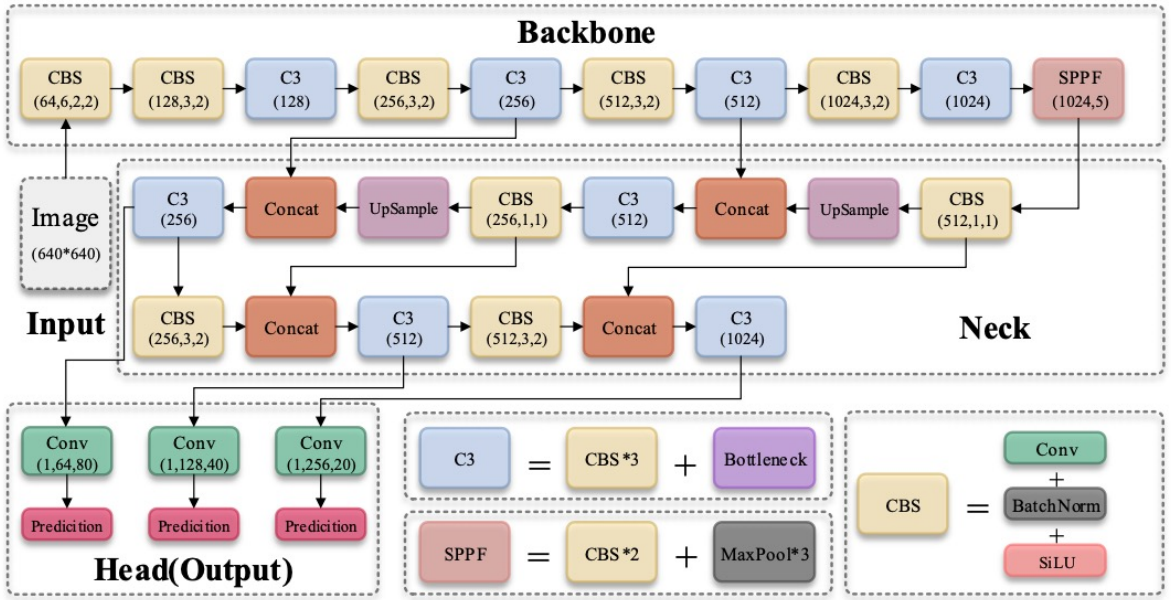
Advantages & Disadvantages of Yolo v5

- It is about 88% smaller than YOLOv4 (27 MB vs 244 MB)
- It is about 180% faster than YOLOv4 (140 FPS vs 50 FPS)
- It is roughly as accurate as YOLOv4 on the same task (0.895 mAP vs 0.892 mAP)
- But the main problem is that for YOLOv5 there is no official paper was released like other YOLO versions. Also, YOLO v5 is still under development and we receive frequent updates from **ultralytics**, developers may update some settings in the future.





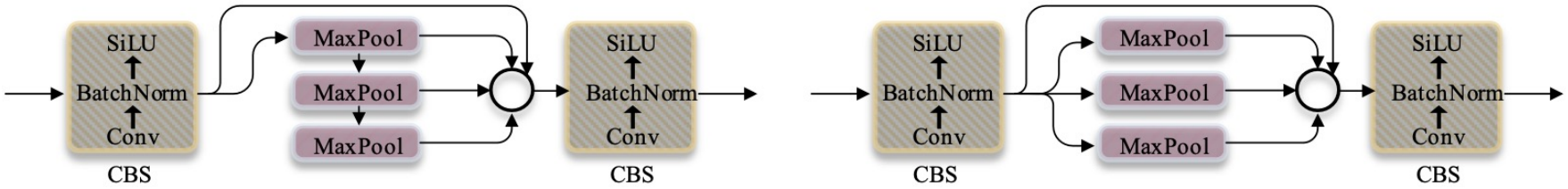
YOLO-v5 Backbone



The backbone is CSPDarknet53.

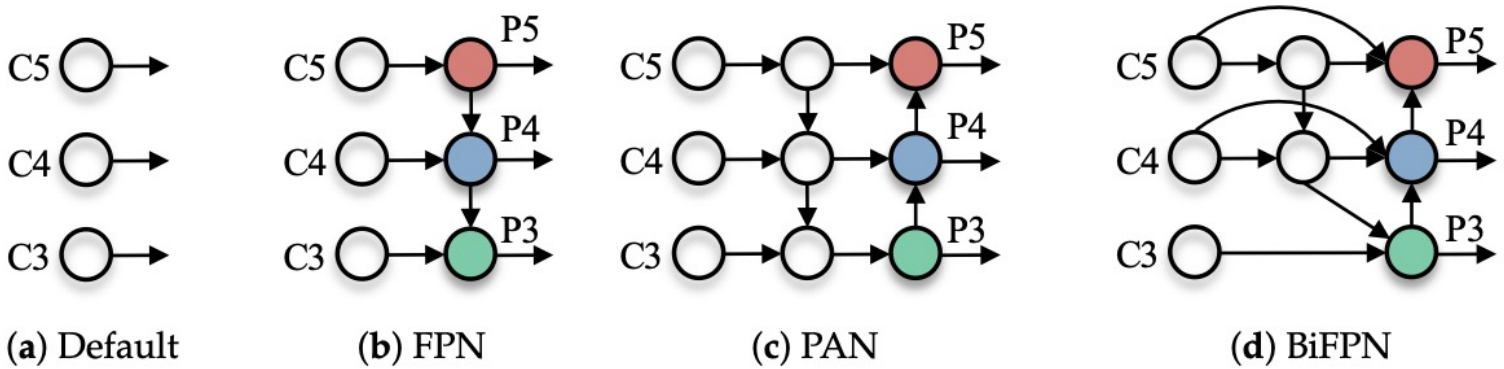
C	From	n	Params	Module	Arguments
0	−1	1	3520	CBS	[3, 32, 6, 2, 2]
1	−1	1	18,560	CBS	[32, 64, 3, 2]
2	−1	1	18,816	C3	[64, 64, 1]
3	−1	1	73,984	CBS	[64, 128, 3, 2]
4	−1	2	115,712	C3	[128, 128, 2]
5	−1	1	295,424	CBS	[128, 256, 3, 2]
6	−1	3	625,152	C3	[256, 256, 3]
7	−1	1	1,180,672	CBS	[256, 512, 3, 2]
8	−1	1	1,182,720	C3	[512, 512, 1]
9	−1	1	656,896	SPPF	[512, 512, 5]

Parameter of backbone in YOLOv5 network structure.



(a) SPPF (b) SPP

YOLO-v5 Neck

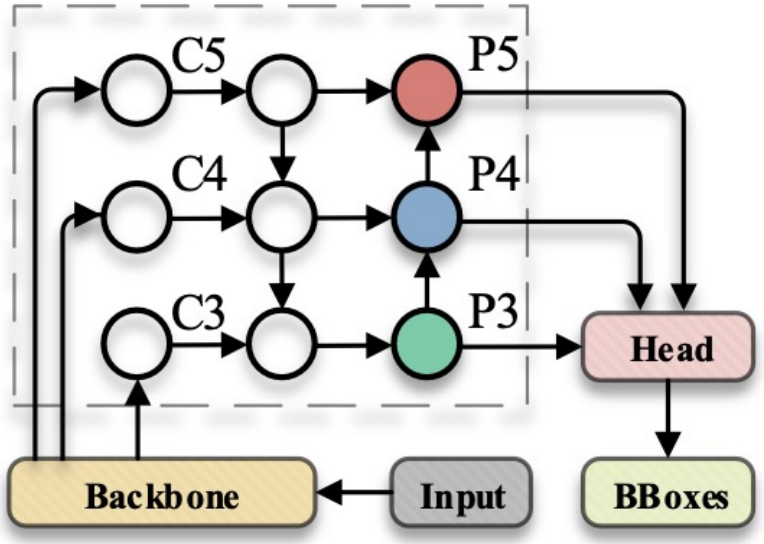


Common feature fusion paths

$$P_4^{td} = Conv\left(\frac{w_1 \cdot P_4^{in} + w_2 \cdot Resize\left(P_5^{in}\right)}{w_1 + w_2 + \varepsilon}\right)$$

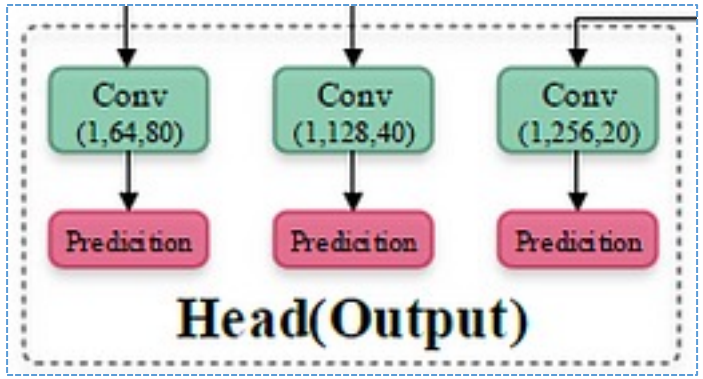
$$P_4^{out} = Conv\left(\frac{w_1' \cdot P_4^{in} + w_2' \cdot P_4^{td} + w_3' \cdot Resize\left(P_3^{out}\right)}{w_1' + w_2' + w_3' + \varepsilon}\right)$$

•The basic idea of FPN is to **up-sampling the output feature map (C3, C4, and C5) generated by multiple convolution down sampling operations from the feature extraction network to generate multiple new feature maps (P3, P4, and P5)** for detecting different scales targets.



YOLOv5 uses the methods of FPN and PAN.

YOLO-v5 Head



$$g_x = 2\sigma(s_x) - 0.5 + r_x$$
$$g_y = 2\sigma(s_y) - 0.5 + r_y$$
$$g_h = p_h(2\sigma(s_h))^2$$
$$g_w = p_w(2\sigma(s_w))^2$$

Bounding box Regression

The process of **adjusting the center coordinate** and **size of the preset prior anchor** to the center coordinate and size of the final prediction box

YOLOv5: Model Variants

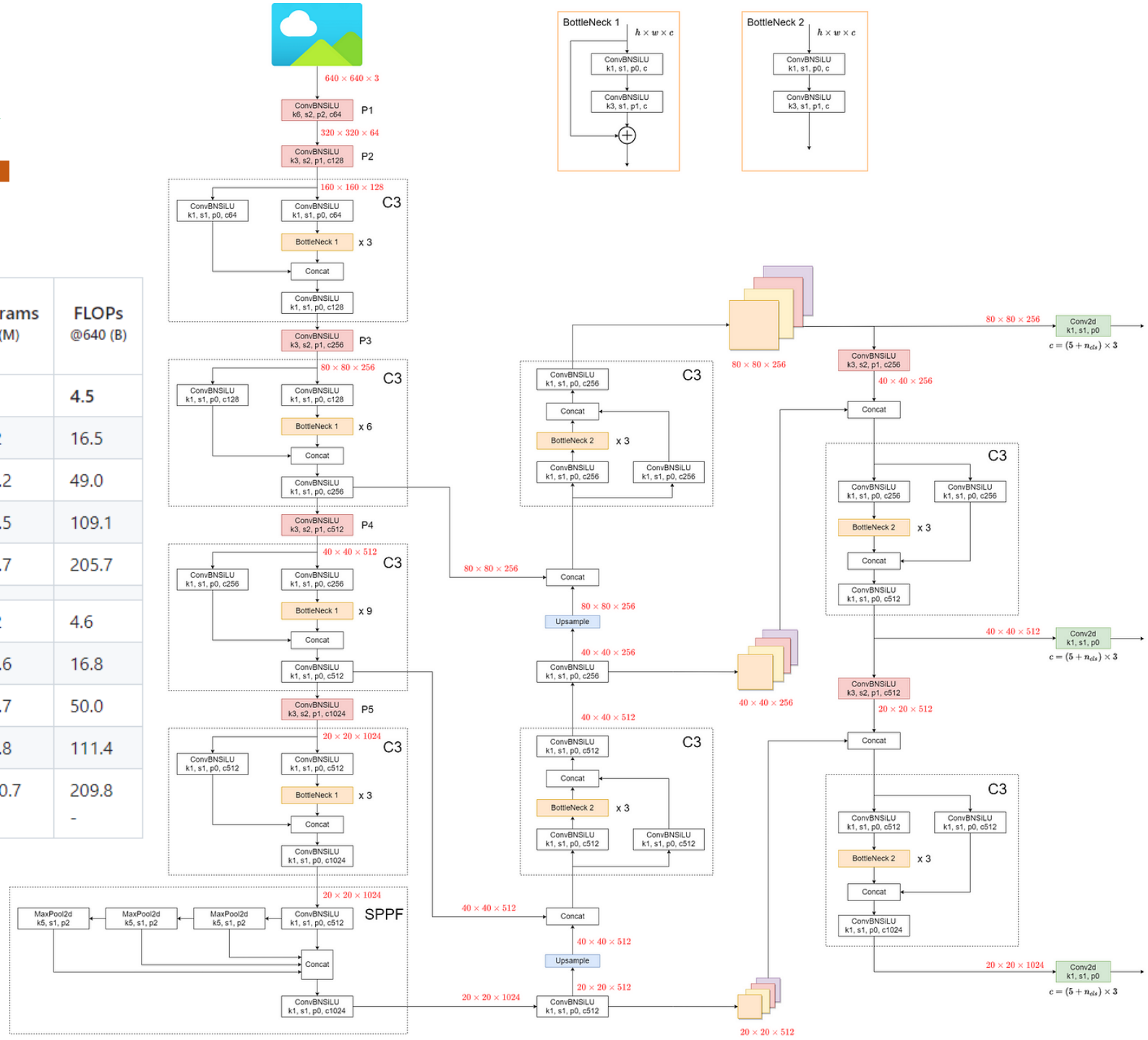
Model	Size (Pixels)	mAP @0.5:0.95	mAP @0.5	Time CPU b1 (ms)	Time V100 b1 (ms)	Time V100 b32 (ms)	Params (M)	FLOPS @640 (B)
YOLOv5n	640	28.0	45.7	45	6.6	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Dataset	Methods	Size	mAP@0.5	mAP@0.5:0.95	Parameters (M)	FLOPs (G)	Inference Time (ms)
TinyPerson	YOLOv5s	640	18.7	6.0	7.02	15.8	12.8
	SF-YOLOv5	640	20.0	6.5	2.23	13.8	11.0
	improvement	-	+1.3	+0.5	−68.2%	−12.7%	−14.1%
VisDrone	YOLOv5s	640	33.0	17.9	7.04	15.9	12.6
	SF-YOLOv5	640	34.3	18.2	2.24	13.8	11.5
	improvement	-	+1.3	+0.3	−68.2%	−13.2%	−8.7%
VOC2012	YOLOv5s	640	60.8	37.0	7.06	15.9	25.8
	SF-YOLOv5-P5	640	61.2	38.3	4.59	15.7	24.8
	improvement	-	+0.4	+1.3	−34.9%	−1.3%	−3.9%

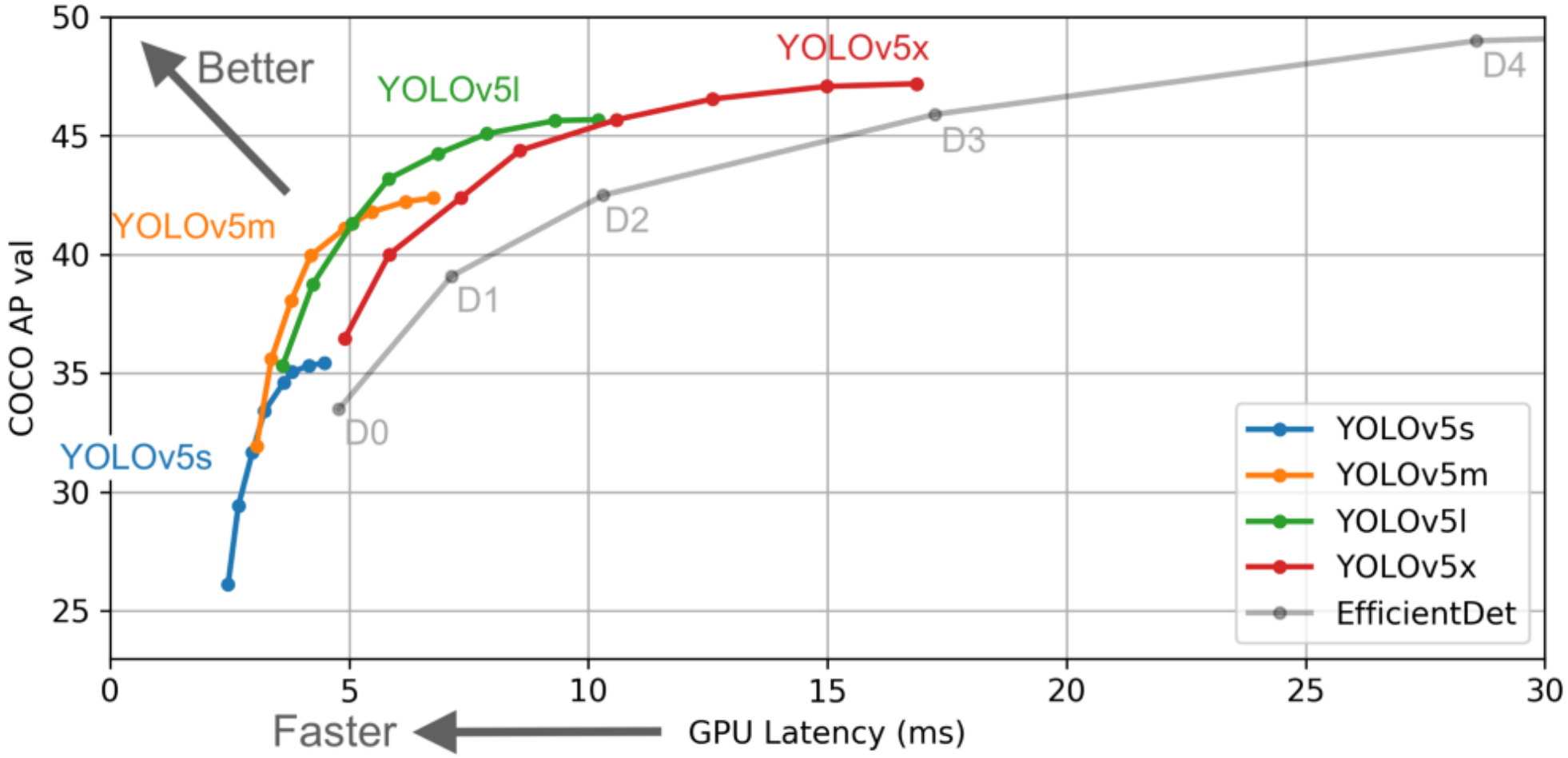
AI VIETNAM
All-in-One Course

YOLOv5l

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

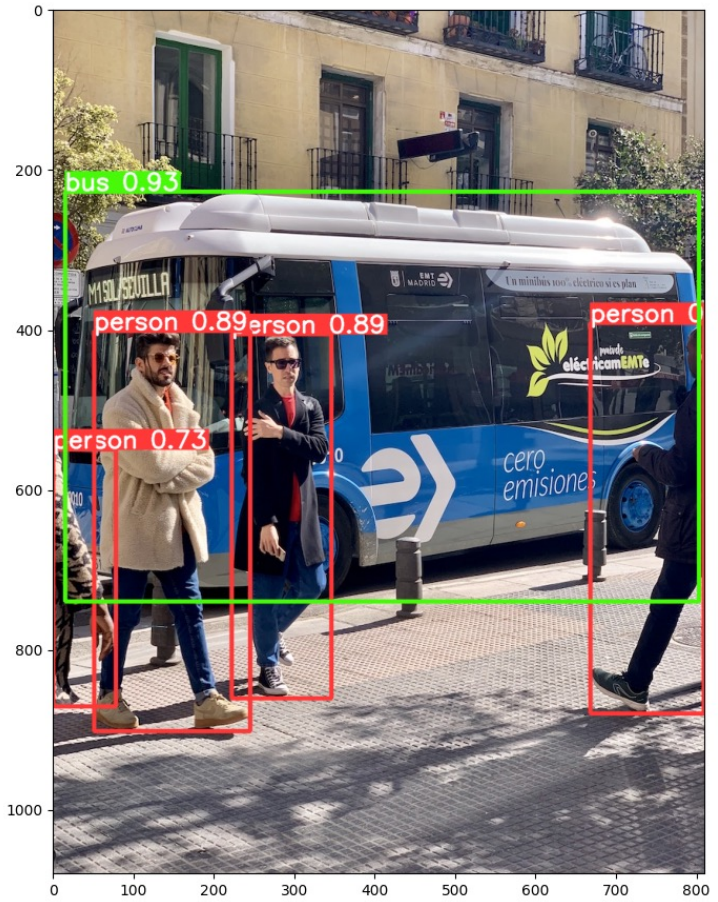


YOLOv5: Model Variants



Example

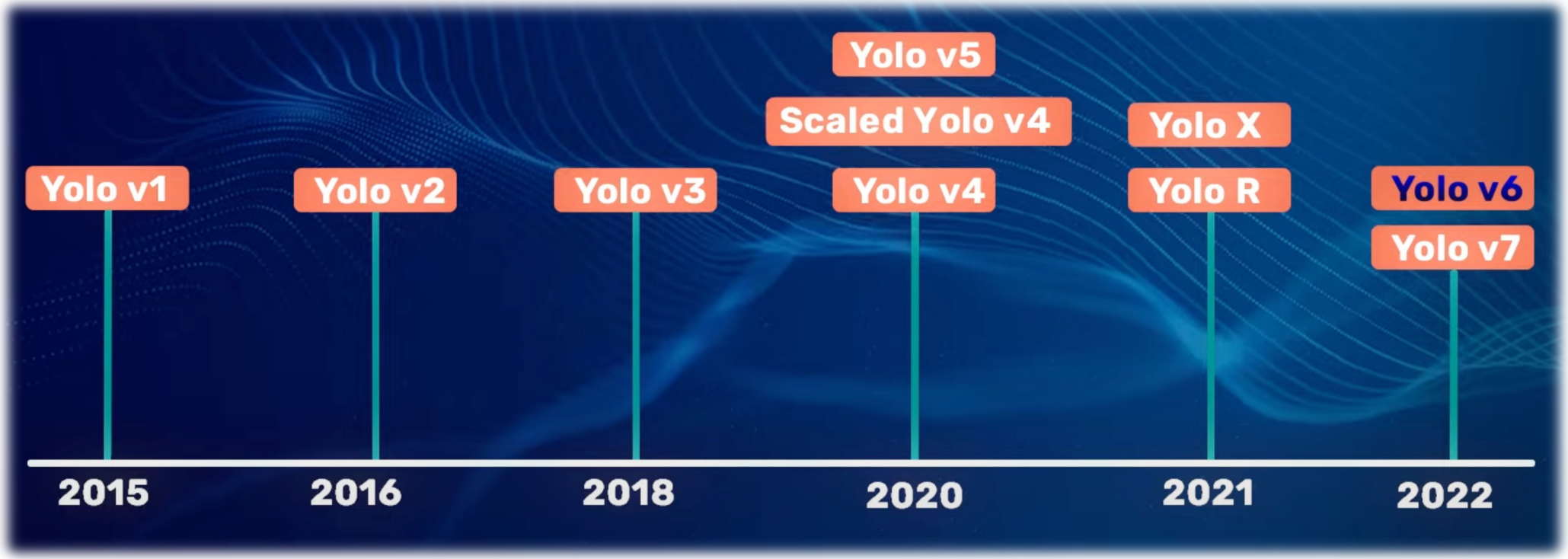
```
from ultralytics import YOLO

# Load a model
model = YOLO('yolov5mu.pt')  # load an official model

# Predict with the model
results = model('https://ultralytics.com/images/bus.jpg')

show_img(results)
```



Outline

- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

YOLO X Architecture



Released in July 2021, YOLOX has switched to the anchor free approach which is different from previous YOLO networks.
In short, salient features of YOLOX are,

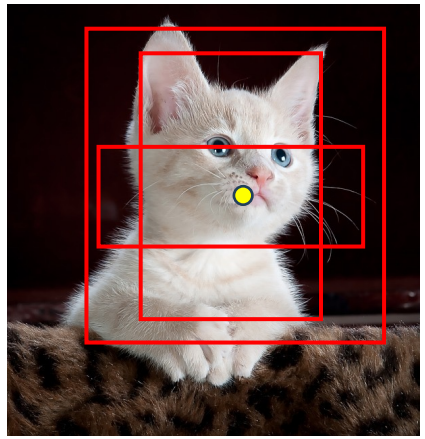
- Anchor free design
- Decoupled head
- simOTA label assignment strategy
- Advanced Augmentations: Mixup and Mosaic

YOLO X Architecture

Drawbacks of Anchor Based Approach

1.It needs a large set of anchor boxes. For example, it is more than 100k in RetinaNet.
2.The anchor boxes require a lot of hyperparameters and design tweaks. For example,

1. Number of anchors
2. Size of the anchors
3. The aspect ratio of the boxes
4. The number of sections the image should be divided into
5. IoU value, to decide labels (+ve or -ve), etc.

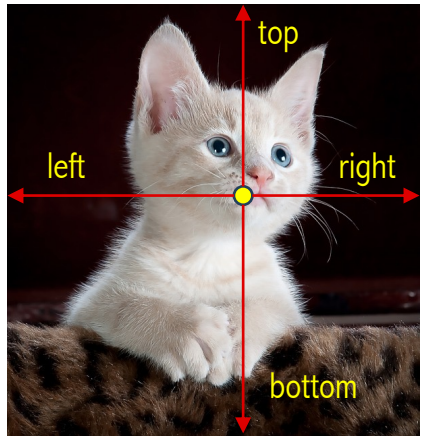


YOLOv1 detector was also anchor-free!

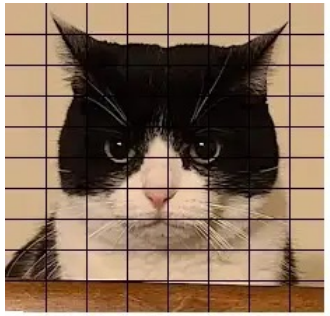
Anchor Free Object Detectors

Anchor-free methods try to localize the objects directly without using the boxes as proposals but using centers or key points

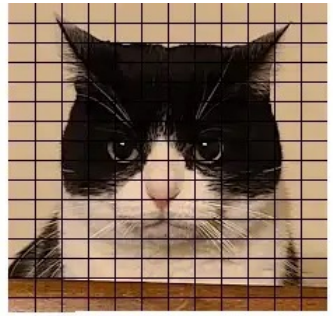
Over the years, anchor free detectors have evolved to be on par with or even better than anchor based detectors. Some examples of anchor free detectors are, CornerNet, CenterNet, and FCOS.



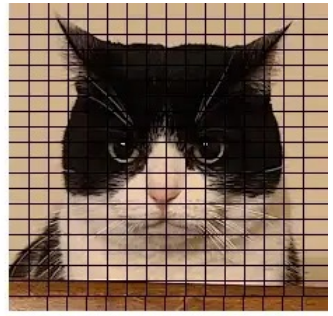
YOLO X Architecture



63 Predictions



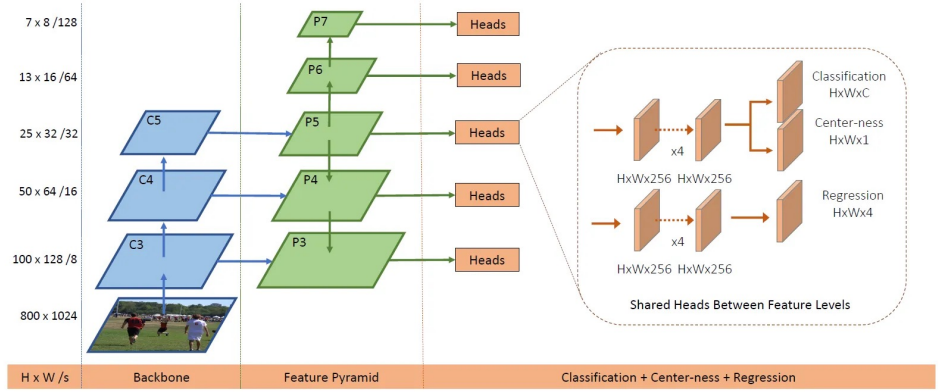
165 Predictions



285 Predictions

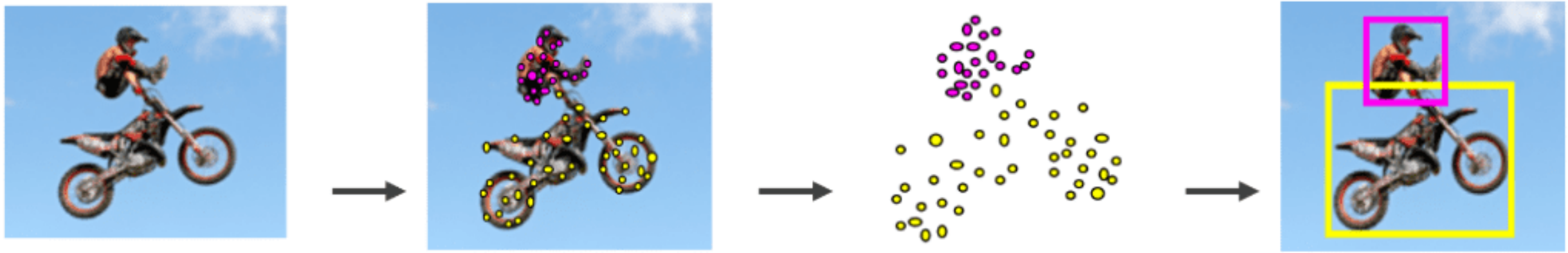
YOLOX was not just based on YOLOv3, it was also based on FCOS, which is another bounding box model

FCOS: Fully Convolutional One-Stage Object Detection

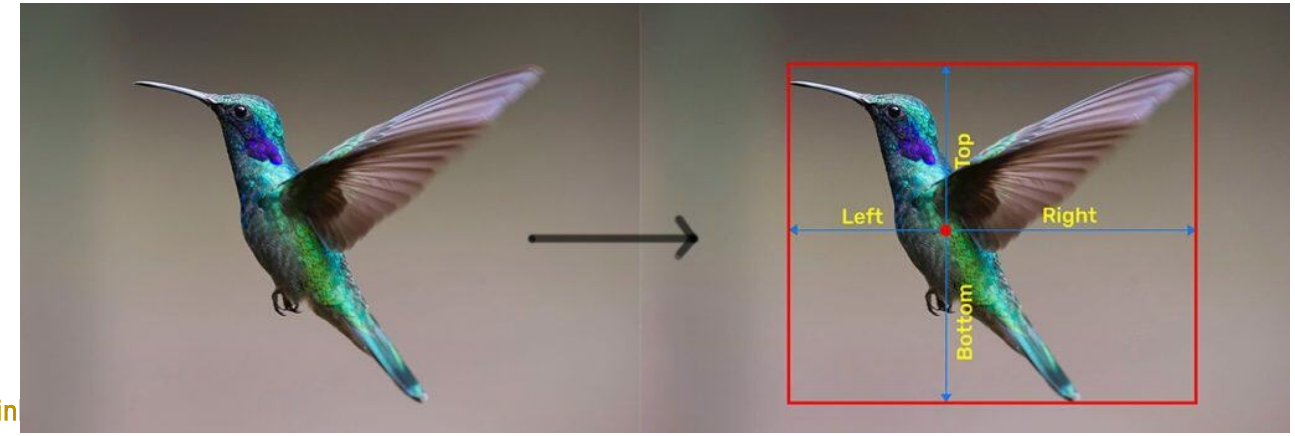


Anchor Free Detectors

Key-point based



Center-based

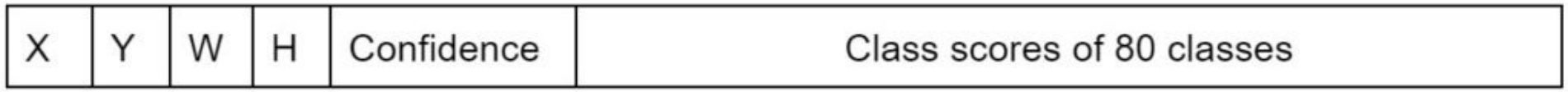


Anchor-based or anchor-free mechanisms are essentially based on how labels (positive, negative) are assigned to a sample. Anchor free does not mean the removal of bounding boxes. It is just a different design strategy where an anchor preset is not required.

Anchor Free YOLOX

YOLOX adopts the center-based approach which has a per-pixel detection mechanism. In anchor based detectors, the location of the input image acts as the center for multiple anchors.

YOLOv3 SPP outputs 3 predictions per location. Each prediction has an 85D vector with embeddings for the following values: Class score IoU score Bounding box coordinates (center x, center y, width, height)

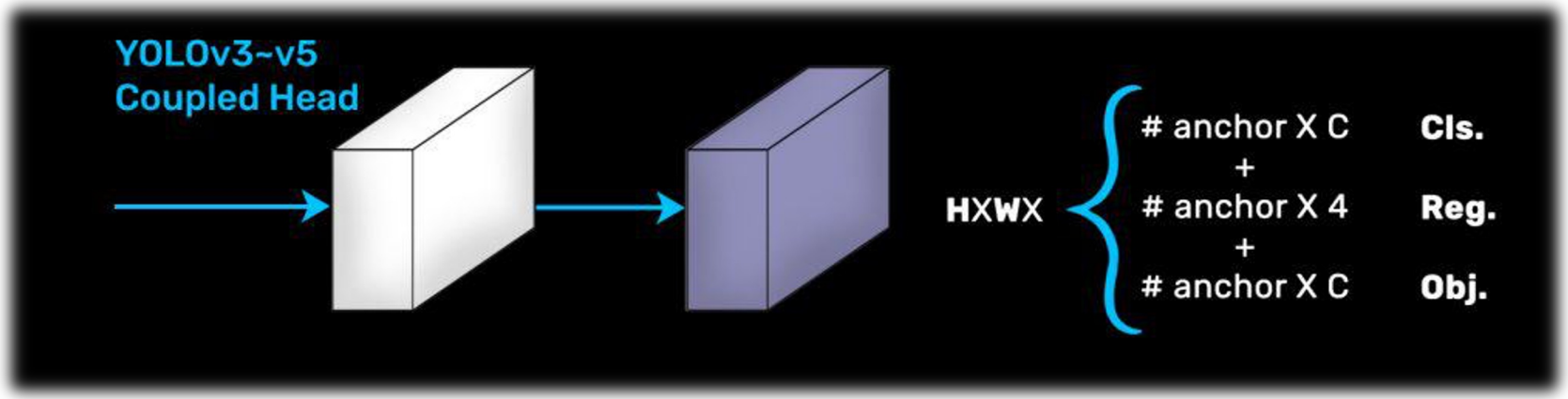


On the other hand, YOLOX reduced the predictions at each location (pixel) from 3 to 1. The prediction contains only a single 4D vector, encoding the location of the box at each foreground pixel.

T = {left, top, right, bottom}

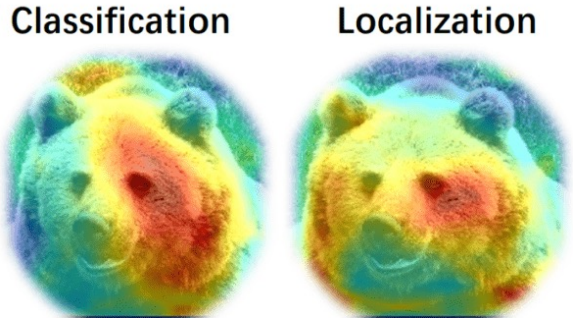
Decoupled Head in YOLOX

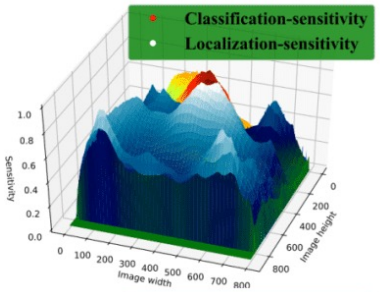
YOLO architecture consists of three parts. The backbone, neck, and head



The Need for Decoupling the YOLO Head

Coupled or shared head is the most commonly used architecture. It was doing fine until researchers at Microsoft pointed out the loopholes in 2020. The paper Rethinking Classification and Localization for Object Detection proved that there is a conflict between the regression (localization) and classification task. It happens due to the misalignment of features as shown below.

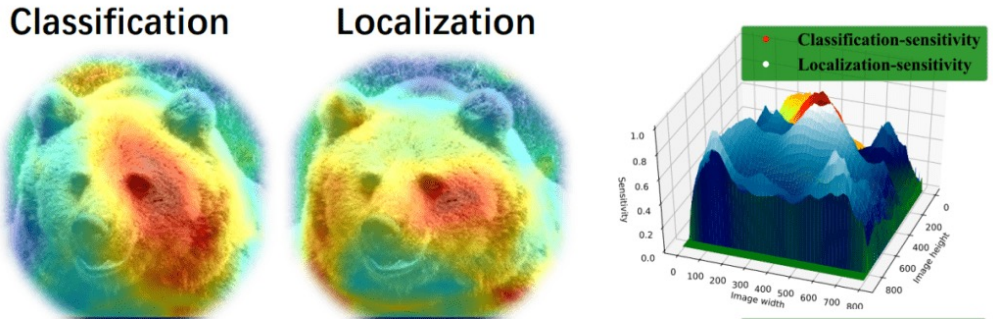


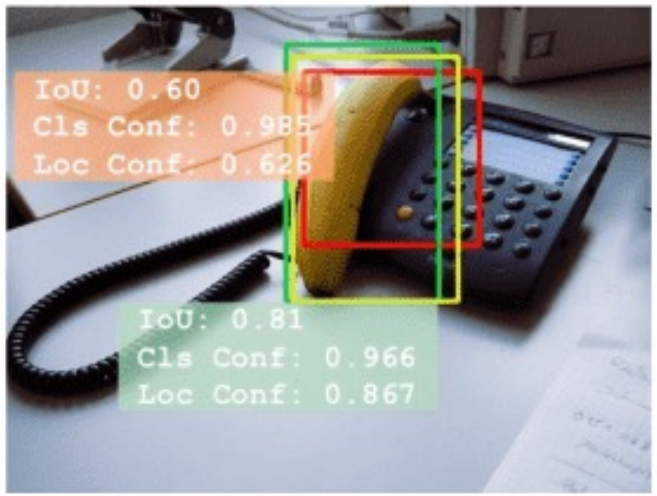


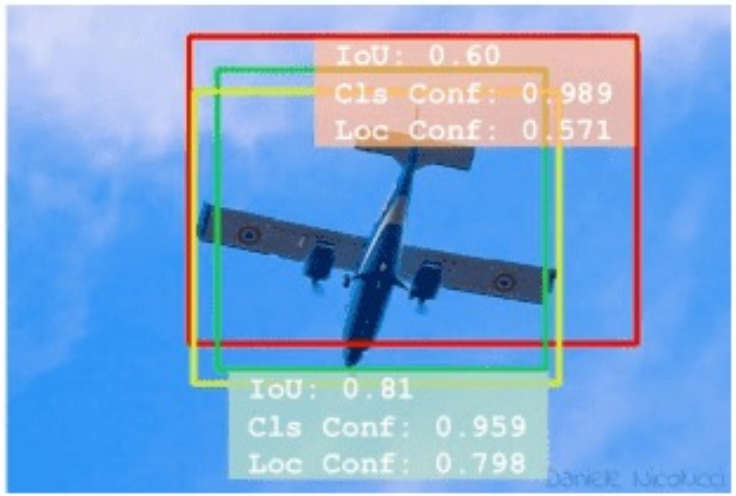
Decoupled Head in YOLOX

The Need for Decoupling the YOLO Head

Coupled or shared head is the most commonly used architecture. It was doing fine until researchers at Microsoft pointed out the loopholes in 2020. The paper Rethinking Classification and Localization for Object Detection proved that there is a conflict between the regression (localization) and classification task. It happens due to the misalignment of features as shown below.

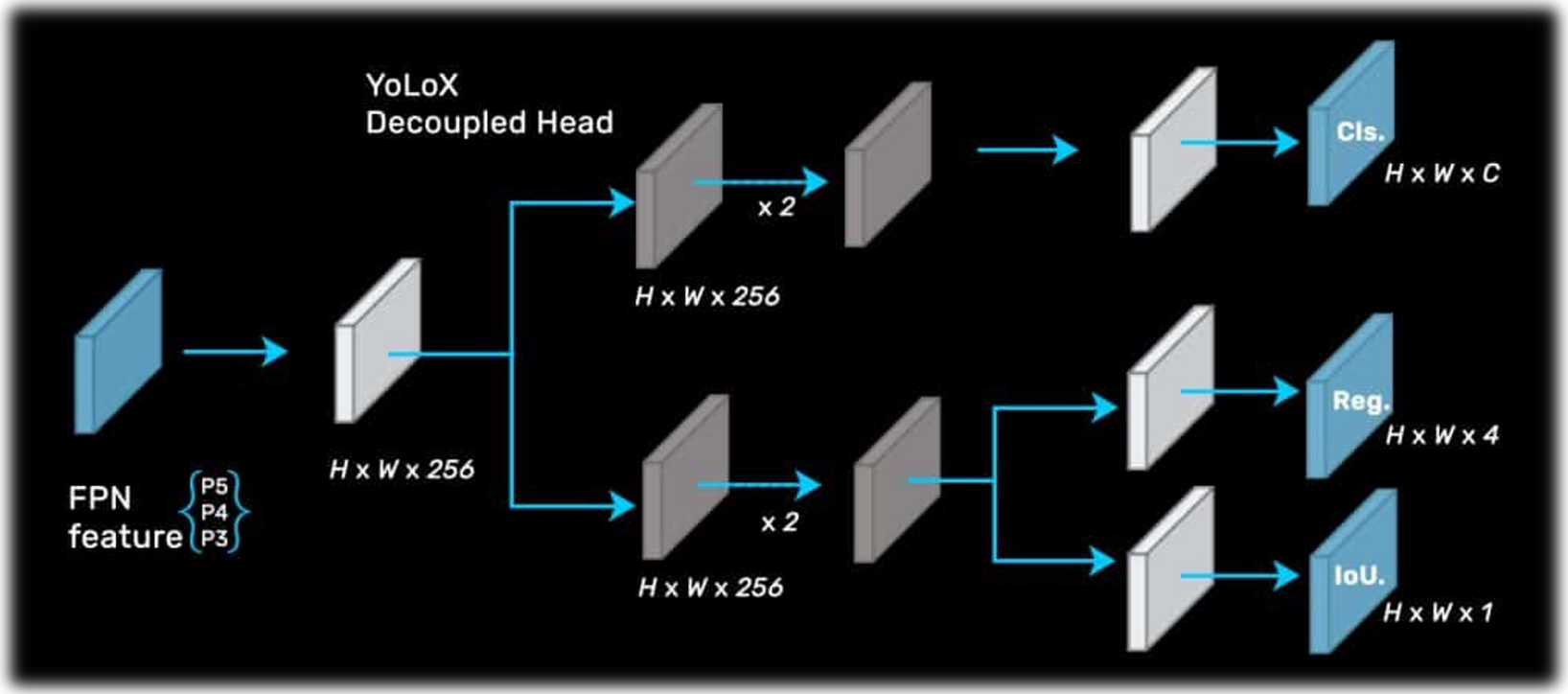






Decoupled Head in YOLOX

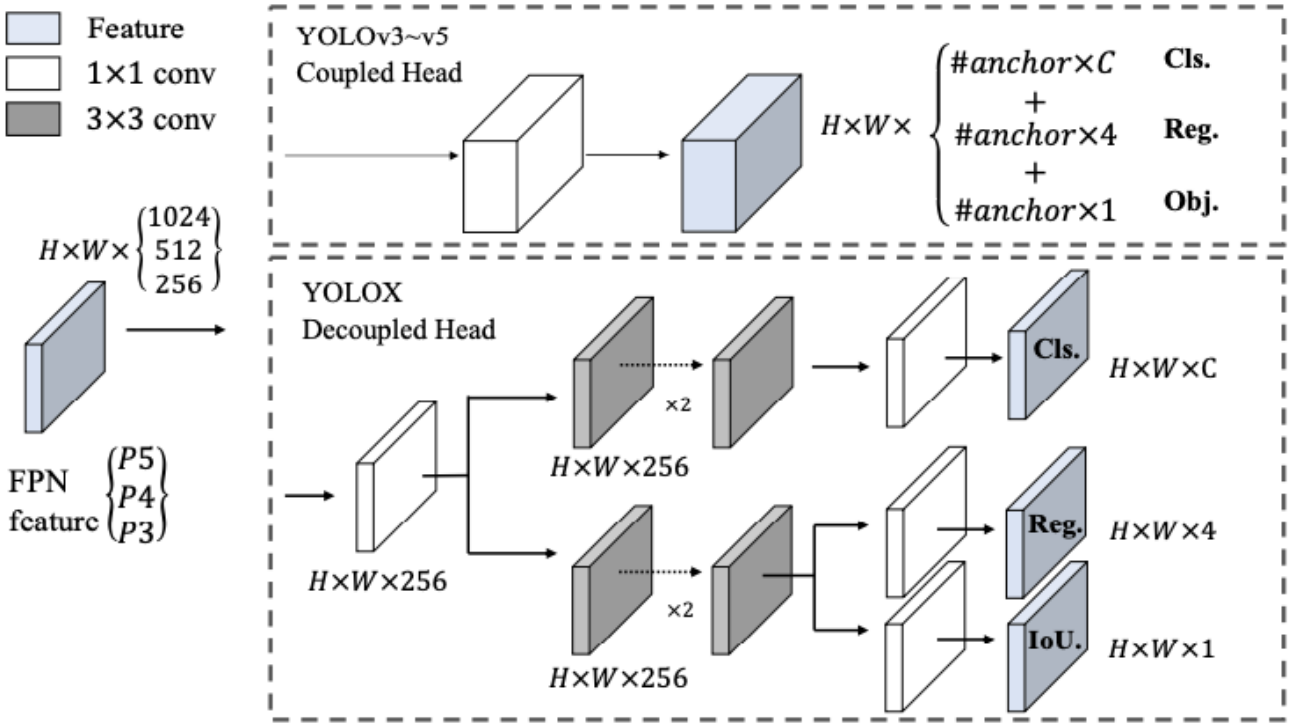
YOLOX implemented a decoupled head for classification and regression tasks. It contains a 1 × 1 **Conv** layer to reduce the channel dimension, followed by two parallel branches with two 3 × 3 **Conv** layers respectively.

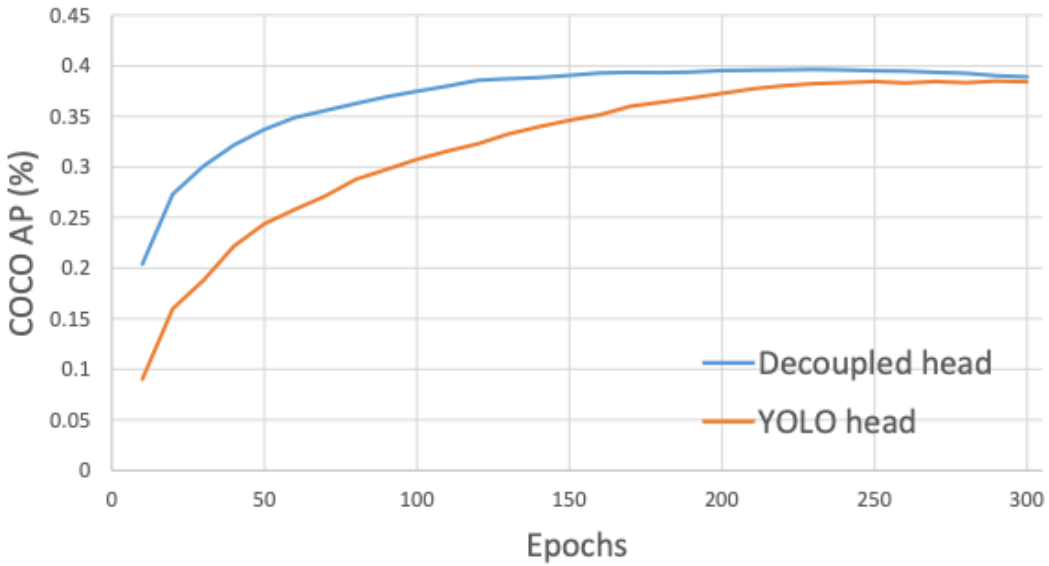


This approach resulted in better performance. Compared to a shared head, which converges in 300 epochs; YOLOX can converge in 200 epochs.

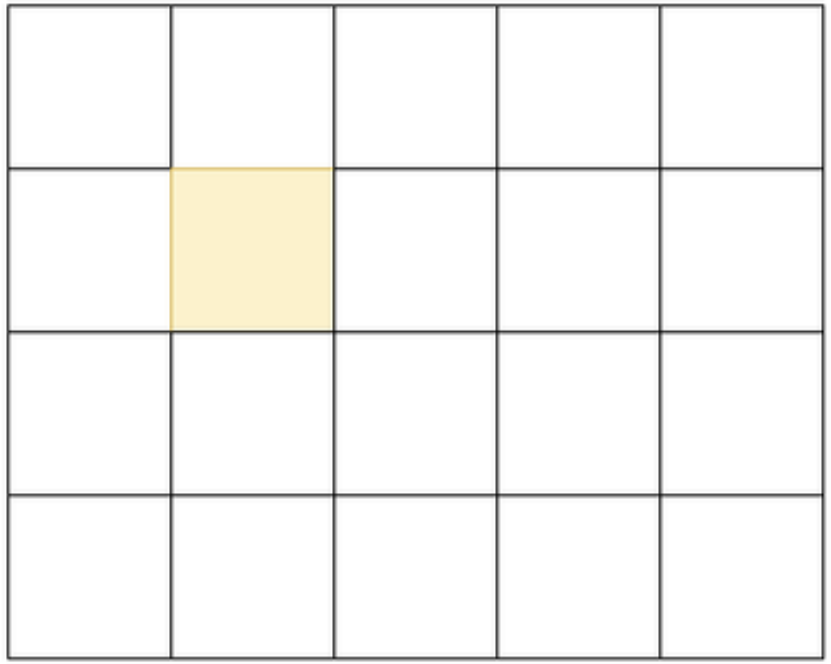
Decoupled Head in YOLOX

YOLOX implemented a decoupled head for classification and regression tasks. It contains a 1 × 1 **Conv** layer to reduce the channel dimension, followed by two parallel branches with two 3 × 3 **Conv** layers respectively.

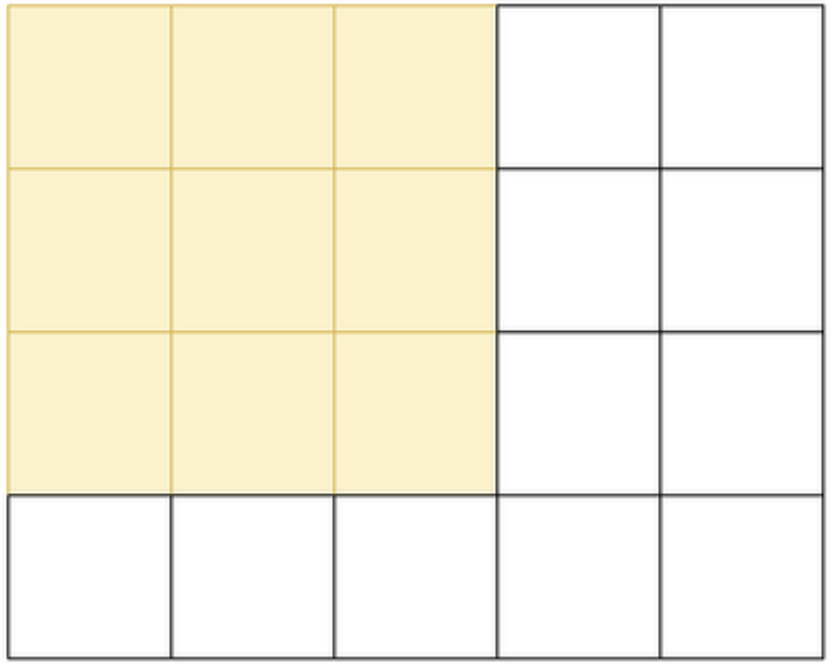




Multi-Positives in YOLOX to Improve the Recall

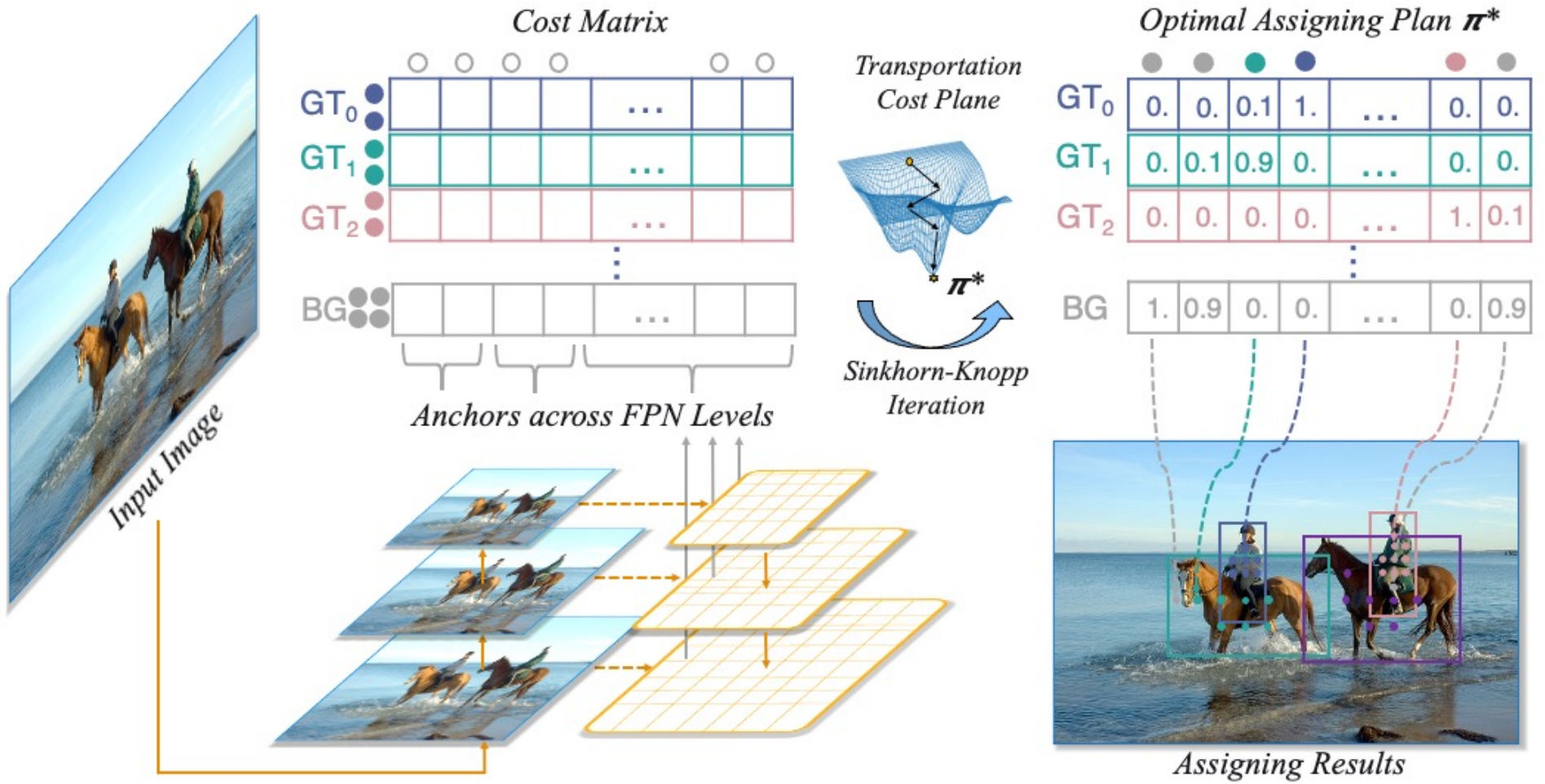


Single positive



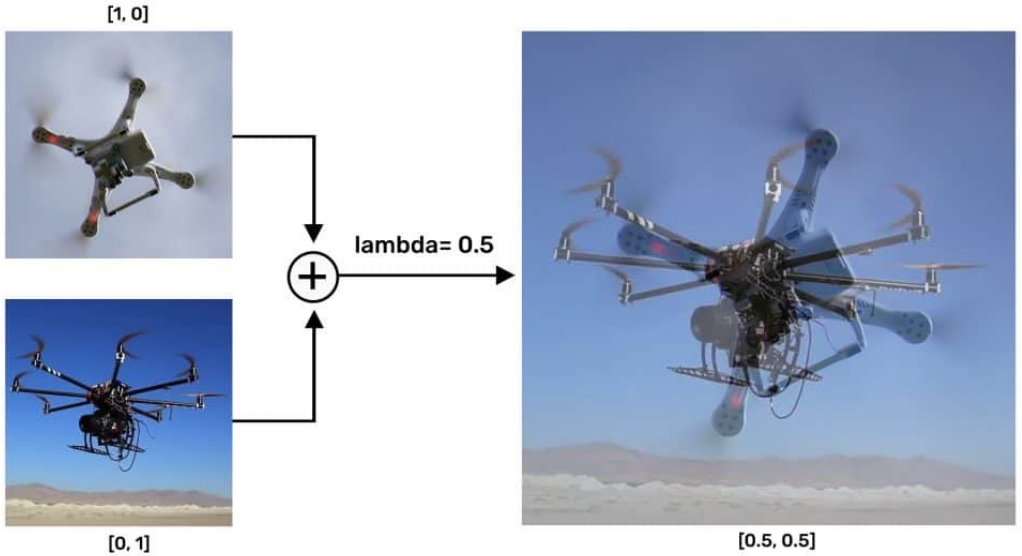
Multi positives

OTA: Optimal Transport Assignment for Object Detection

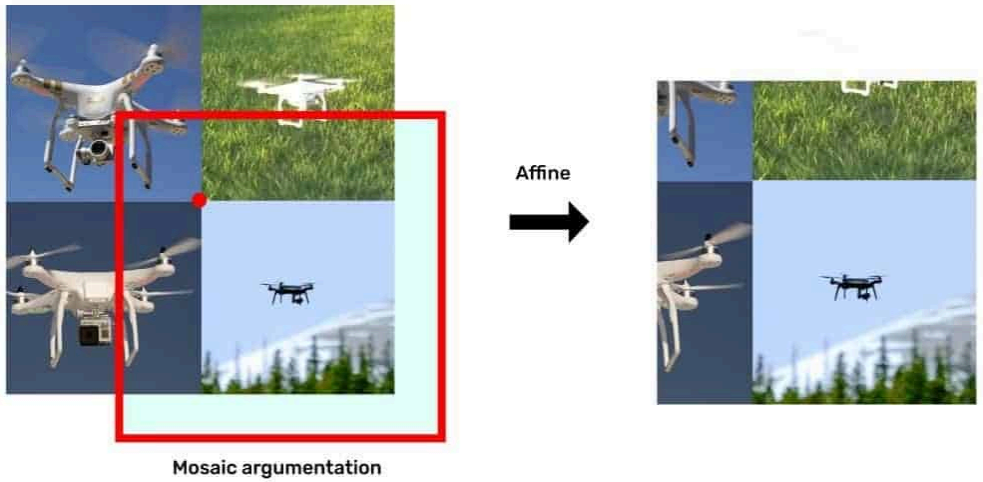


YOLOX: Strong Augmentation

Mixup Augmentation



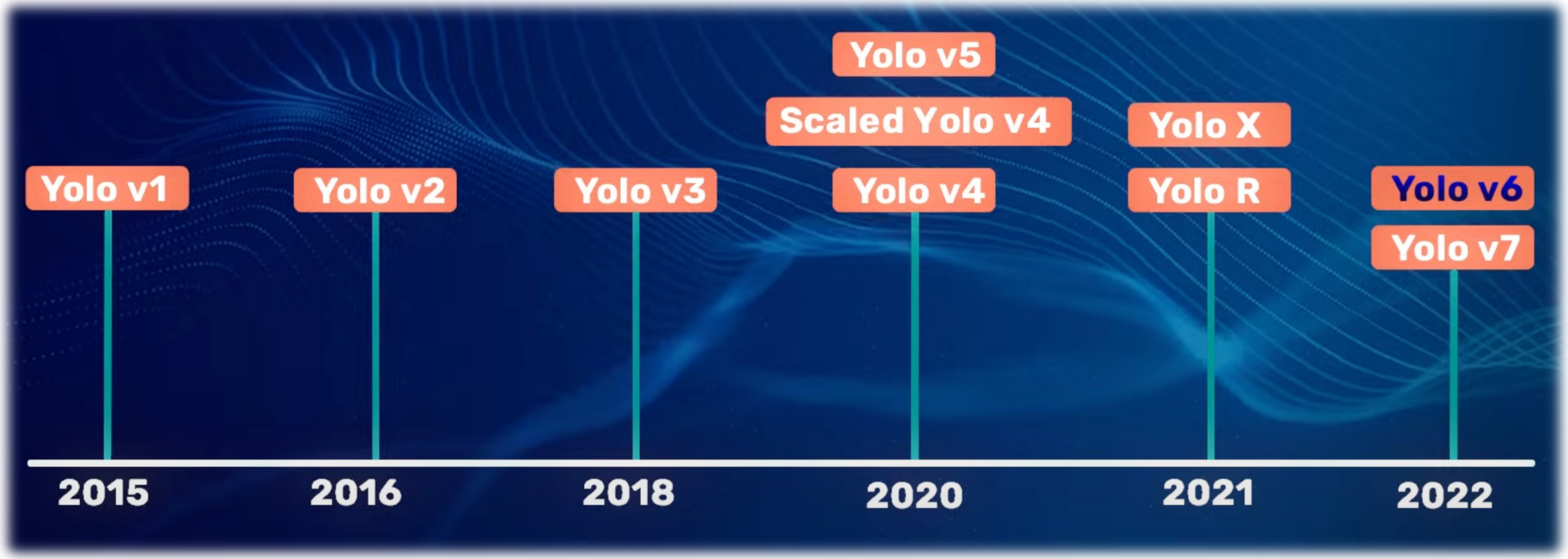
Mosaic Augmentation



Outline

- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

YOLO-v6 Architecture



Note that MT-YOLOv6 is not a part of the official YOLO series and has been called YOLOv6 as it was inspired by one-stage YOLO algorithms and hence is being pushed as the next version of the YOLO models by the authors.

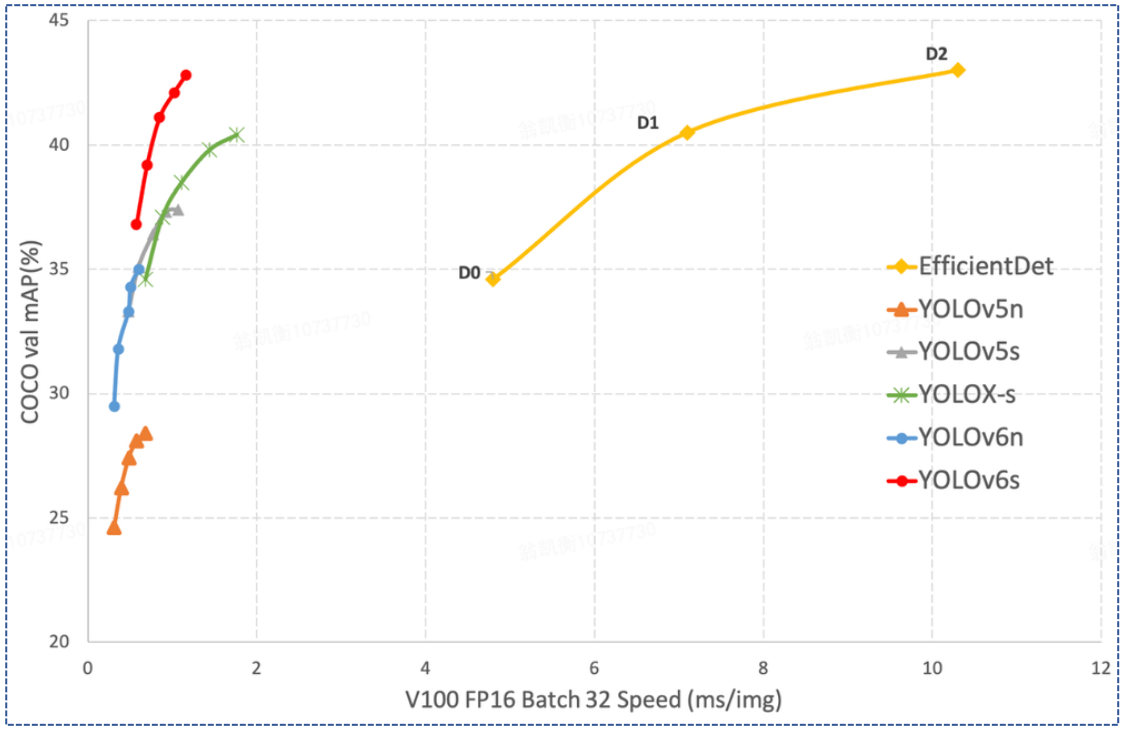
YOLO-v6 Architecture

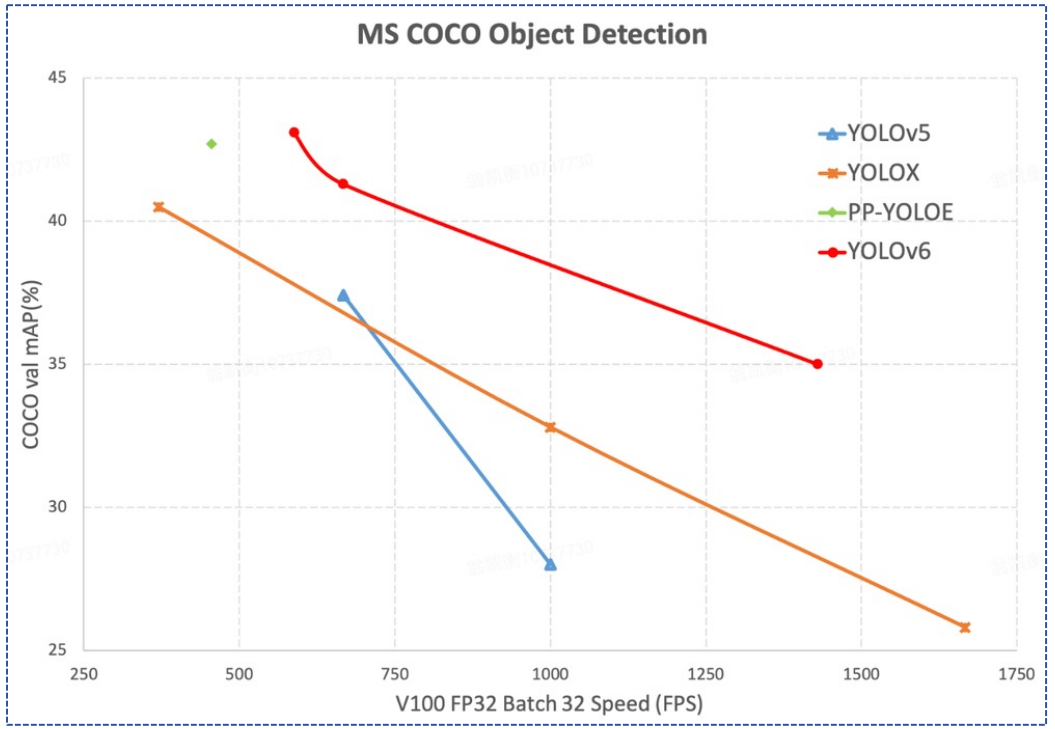
Layers	YOLOv3	YOLOv4	YOLOv5	YOLOv7	YOLOv6
Backbone	**Darknet53**	**CSPDarknet53** (CSPNet in Darknet)	**CSPDarknet53** Focus structure	**EELAN**	**RepVGG** and **CSPRepStack**
Neck	**FPN** (Feature Pyramid Network)	**SPP** (Spatial Pyramid Pooling) and **PANet** (Path Aggregation Network)	**PANet**	**PANet**	**RepPAN**
Head	**B x (5 + C)** output layer **B**: No. of bounding boxes **C**: Class score	Same as Yolo v3	Same as Yolo v3	**Lead Head** for final output, **Auxiliary Head** for middle layer outputs	**Decoupled Classification** and **Detection Head**
Loss Function	Binary Cross Entropy	Binary Cross Entropy	Binary Cross Entropy and Logit Loss Function	**BCE with Focal Loss** for Classification, **IoU loss** for Detection	**Varifocal Loss for Classification** and **Distribution Focal Loss for Detection**

YOLOv6 employs plenty of new approaches to achieve state-of-the-art results. These can be summarized into four points:

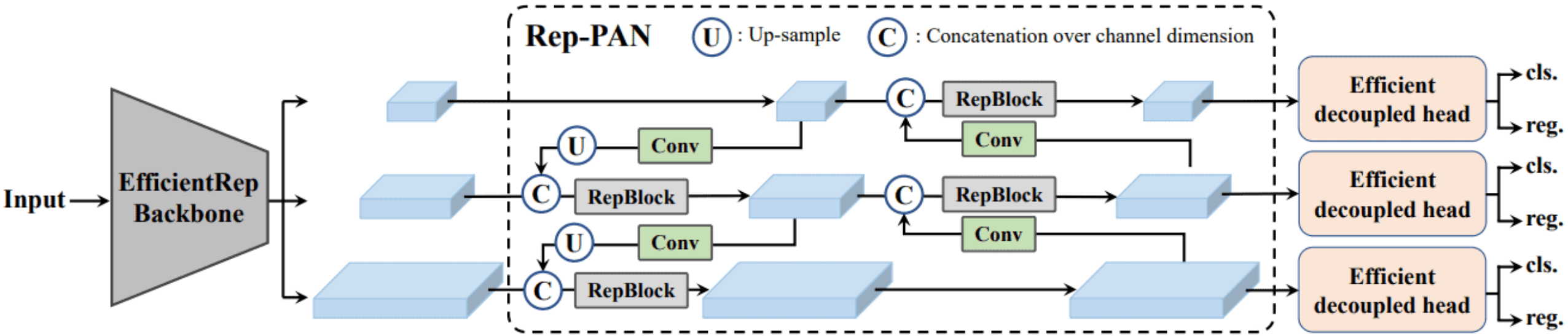
- **Anchor free:** Hence provides better generalizability and costs less time in post-processing.
- **The model architecture: YOLOv6 comes with a revised reparameterized backbone and neck.**
- **Loss functions: YOLOv6 used Varifocal loss (VFL) for classification and Distribution Focal loss (DFL) for detection.**
- **Industry handy improvements: Longer training epochs, quantization, and knowledge distillation** are some techniques that make YOLOv6 models best suited for real-time industrial applications.

YOLO-v6 Architecture



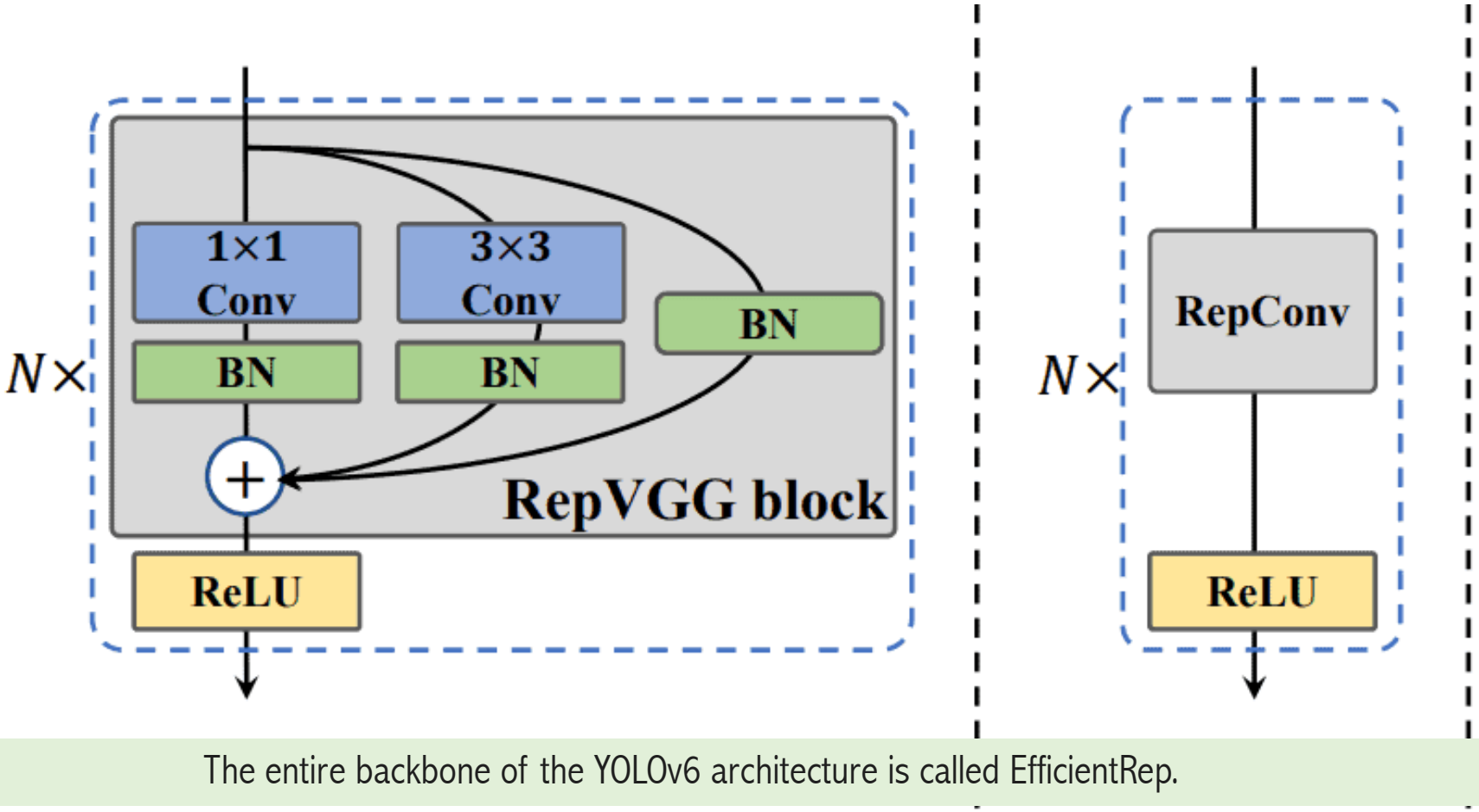


YOLO-v6 Architecture



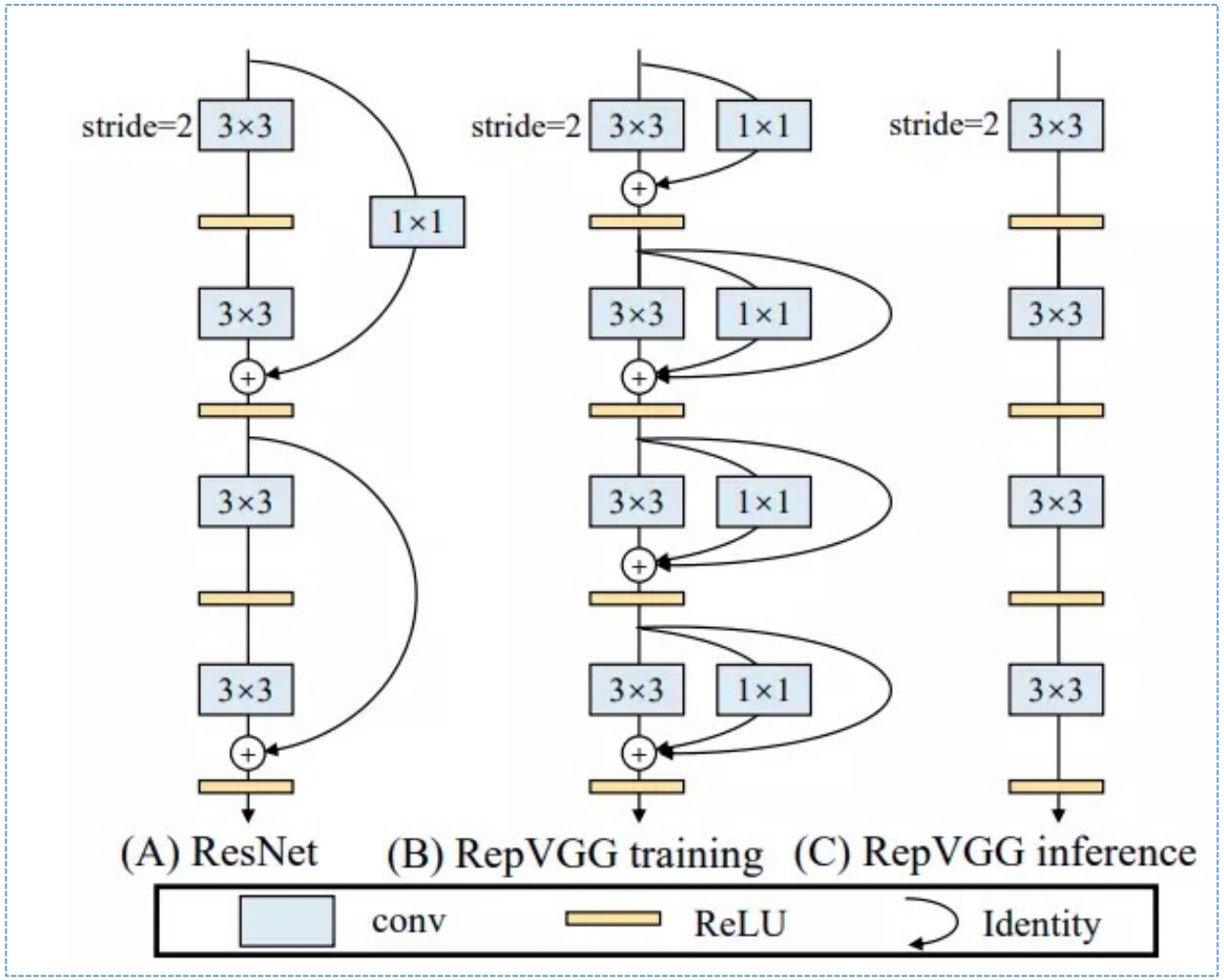
YOLOv6 Backbone Architecture

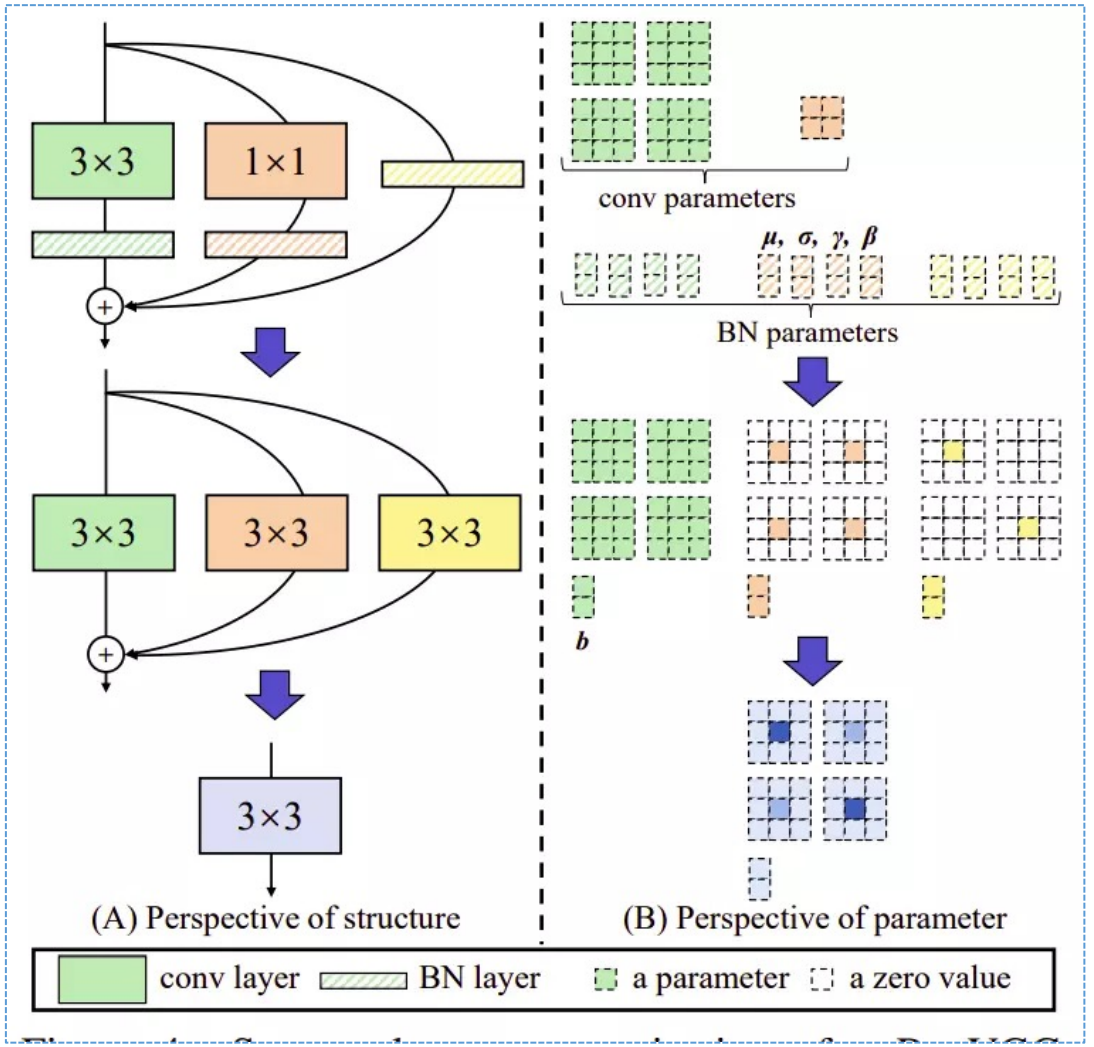
YOLOv6 Nano, Tiny, and Small architectures use reparameterized VGG networks



The entire backbone of the YOLOv6 architecture is called EfficientRep.

RepVGG: Making VGG-style ConvNets Great Again





Re-parameterization VGG

C1 = C2, H1 = H2, W1 = W2

- C_1 : input channel
- C_2: output channel
- $M^{(1)} \in R^{N \times C_1 \times H_1 \times W_1}$: input
- $M^{(2)} \in R^{N \times C_2 \times H_2 \times W_2}$: output
- $W^{(3)} \in R^{C_2 \times C_1 \times 3 \times 3}$: kernel của conv 3x3
- $W^{(1)} \in R^{C_2 \times C_1}$: kernel của conv 1x1
- $\mu^{(3)}, \sigma^{(3)}, \gamma^{(3)}, \beta^{(3)}$: mean, stanndard deviation, scalling factor và bias của lớp Batch Norm sau lớp conv 3x3
- $\mu^{(1)}, \sigma^{(1)}, \gamma^{(1)}, \beta^{(1)}$: mean, stanndard deviation, scalling factor và bias của lớp Batch Norm sau lớp conv 1x1
- $\mu^{(0)}, \sigma^{(0)}, \gamma^{(0)}, \beta^{(0)}$: mean, stanndard deviation, scalling factor và bias của lớp Batch Norm của nhánh identity
- $*$: phép tích chập
- bn: lớp batch normalization

Inference Time

$$\forall 1 \leq i \leq C_2$$

$$bn(M, \mu, \sigma, \gamma, \beta)_{:,i,:,:} = (M_{:,i,:,:} - \mu_i)\frac{\gamma_i}{\sigma_i} + \beta_i = M_{:,i,:,:}\frac{\gamma_i}{\sigma_i} + (-\mu_i\frac{\gamma_i}{\sigma_i} + \beta_i)$$

$$\mathrm{W}'_{i,:,:,:} = \frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i}\mathrm{W}_{i,:,:,:}, \quad \mathrm{b}'_i = -\frac{\boldsymbol{\mu}_i\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} + \boldsymbol{\beta}_i$$

$$M^{(2)} = bn(M * W, \mu, \sigma, \gamma, \beta)_{:,i,:,:} = (M_{:,i,:,:}W_{:,i,:,:} - \mu_i)\frac{\gamma_i}{\sigma_i} + \beta_i = (M^{(1)} * W')_{:,i,:,:} + b'_i = M^{(1)} * W' + b'$$

Khi huấn luyện: $M^{(2)} = M^{(1)} * W'^{(1+3+0)} + b'^{(1+3+0)}$

Khi inference: $M^{(2)} = M^{(1)} * W' + b'$

==> $M^{(1)} = W'^{(1+3+0)}$ và $b' = b'^{(1+3+0)}$

$$\mathrm{M}^{(2)} = \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(3)}, \boldsymbol{\mu}^{(3)}, \boldsymbol{\sigma}^{(3)}, \boldsymbol{\gamma}^{(3)}, \boldsymbol{\beta}^{(3)}) + \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(1)}, \boldsymbol{\mu}^{(1)}, \boldsymbol{\sigma}^{(1)}, \boldsymbol{\gamma}^{(1)}, \boldsymbol{\beta}^{(1)}) + \mathrm{bn}(\mathrm{M}^{(1)}, \boldsymbol{\mu}^{(0)}, \boldsymbol{\sigma}^{(0)}, \boldsymbol{\gamma}^{(0)}, \boldsymbol{\beta}^{(0)}).$$

$$M^{(2)} = (M^{(1)} * W'^{(3)})_{:,i,:,:} + b'^{(3)}_i + (M^{(1)} * W'^{(1)})_{:,i,:,:} + b'^{(1)}_i + (M^{(1)} * W'^{(0)})_{:,i,:,:} + b'^{(0)}_i$$

$$M^{(2)} = M^{(1)} * (W'^{(1)} + W'^{(3)} + W'^{(0)}) + (b'^{(0)} + b'^{(3)} + b'^{(1)})$$

Train Time

Re-parameterization VGG

- **weight** (*Tensor*) – the learnable weights of the module of shape $(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, \text{kernel\_size}[0], \text{kernel\_size}[1])$. The values of these weights are sampled from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{groups}{C_{\text{in}} * \prod_{i=0}^{1} \text{kernel\_size}[i]}$

- C_1 : input channel
- C_2: output channel
- $M^{(1)} \in R^{N \times C_1 \times H_1 \times W_1}$: input
- $M^{(2)} \in R^{N \times C_2 \times H_2 \times W_2}$: output
- $W^{(3)} \in R^{C_2 \times C_1 \times 3 \times 3}$: kernel của conv 3x3
- $W^{(1)} \in R^{C_2 \times C_1}$: kernel của conv 1x1
- $\mu^{(3)}, \sigma^{(3)}, \gamma^{(3)}, \beta^{(3)}$: mean, stanndard deviation, scalling factor và bias của lớp Batch Norm sau lớp conv 3x3
- $\mu^{(1)}, \sigma^{(1)}, \gamma^{(1)}, \beta^{(1)}$: mean, stanndard deviation, scalling factor và bias của lớp Batch Norm sau lớp conv 1x1
- $\mu^{(0)}, \sigma^{(0)}, \gamma^{(0)}, \beta^{(0)}$: mean, stanndard deviation, scalling factor và bias của lớp Batch Norm của nhánh identity
- $*$: phép tích chập
- bn: lớp batch normalization

C1 = C2, H1 = H2, W1 = W2

torch.nn.Conv2d()

Inference Time

$$\forall 1 \leq i \leq C_2$$

$$bn(M, \mu, \sigma, \gamma, \beta)_{:,i,:,:} = (M_{:,i,:,:} - \mu_i)\frac{\gamma_i}{\sigma_i} + \beta_i = M_{:,i,:,:}\frac{\gamma_i}{\sigma_i} + (-\mu_i\frac{\gamma_i}{\sigma_i} + \beta_i)$$

$$\mathrm{W}'_{i,:,:,:} = \frac{\gamma_i}{\sigma_i}\mathrm{W}_{i,:,:,:}, \quad \mathrm{b}'_i = -\frac{\mu_i\gamma_i}{\sigma_i} + \beta_i$$

$$M^{(2)} = bn(M * W, \mu, \sigma, \gamma, \beta)_{:,i,:,:} = (M_{:,i,:,:}W_{:,i,:,:} - \mu_i)\frac{\gamma_i}{\sigma_i} + \beta_i = (M^{(1)} * W')_{:,i,:,:} + b'_i = M^{(1)} * W' + b'$$

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$$

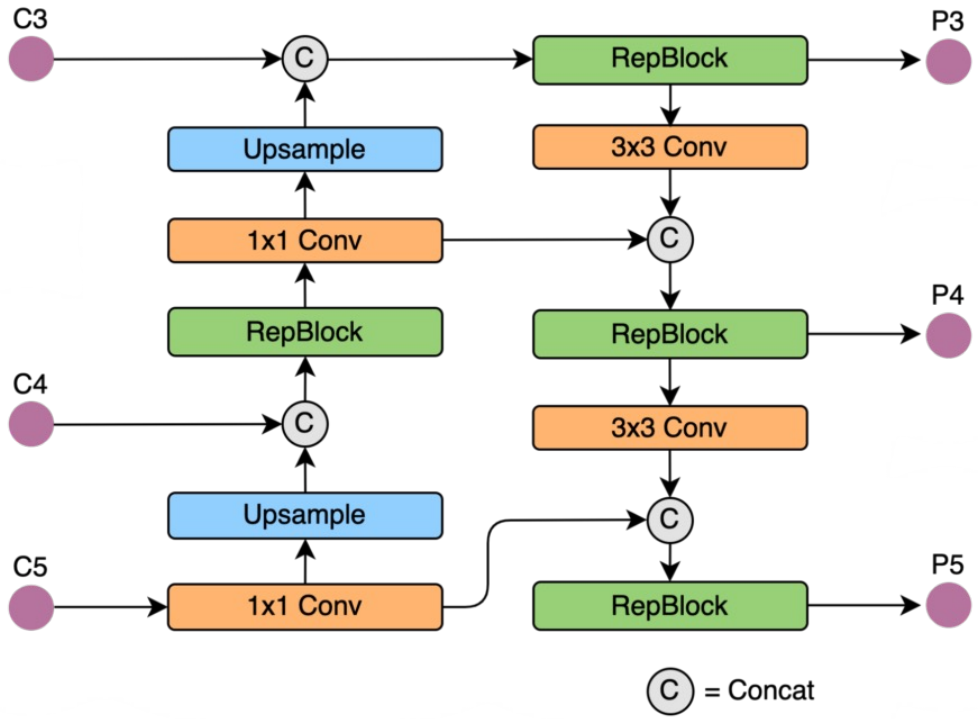
$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{(\sigma_B^{(k)})^2}}$$

Mean = 0
Variance = 1

$$y_i^{(k)} = \gamma_i^{(k)} \times \frac{x_i^{(k)} - \mu_B^{(k)}}{\sigma_B^{(k)}} + \beta_i^{(k)} \quad \Rightarrow \quad BN(M, \mu, \gamma, \sigma, \beta)_{[:,i,:,:]} = \frac{\gamma_i}{\sigma_i} \times M_{[:,i,:,:]} - \frac{\gamma_i \times \mu_i}{\sigma_i} + \beta_i$$

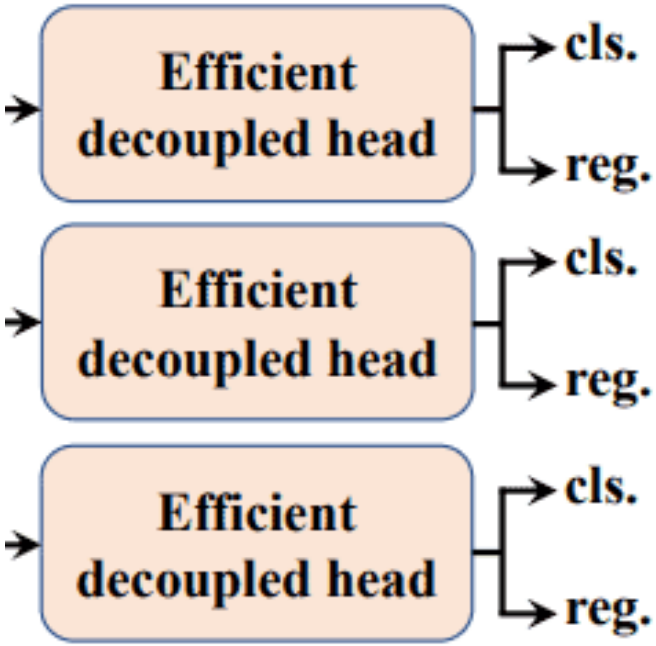
YOLOv6 Neck Architecture

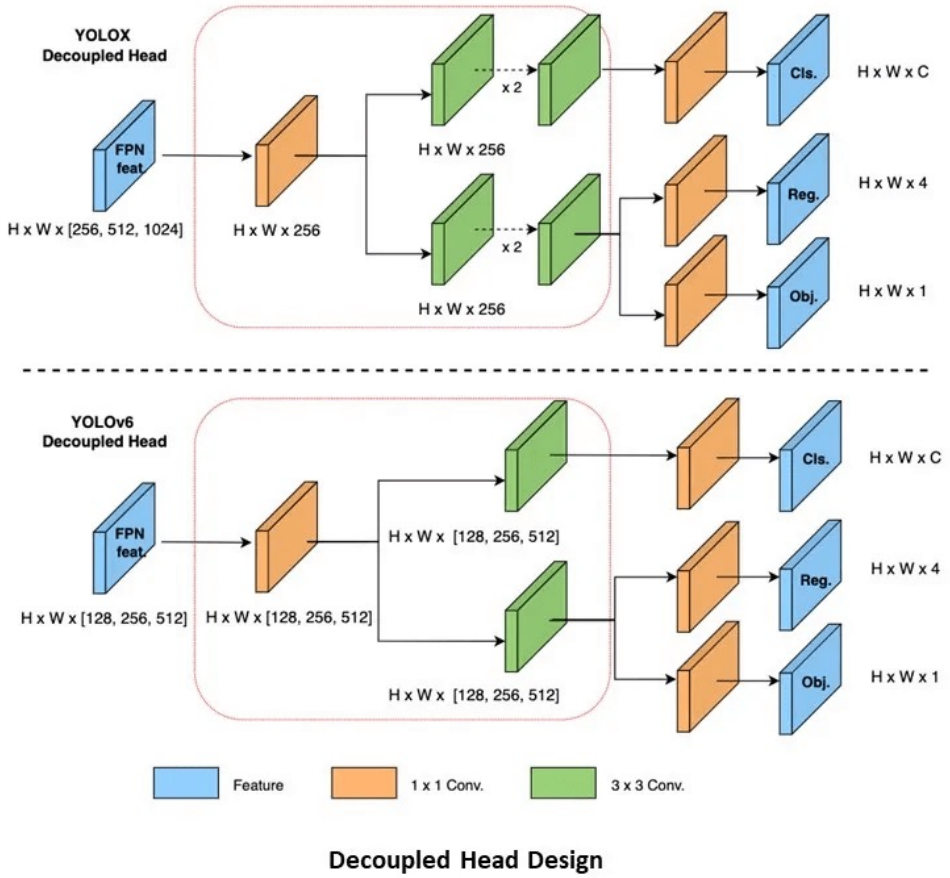
In most object detection models, the neck aggregates the multi-scale feature maps using **PAN (Path Aggregation Networks)**. This is not different in YOLOv6 and similar to what happens in YOLOv4 and YOLOv5.



YOLOv6 Detection Head

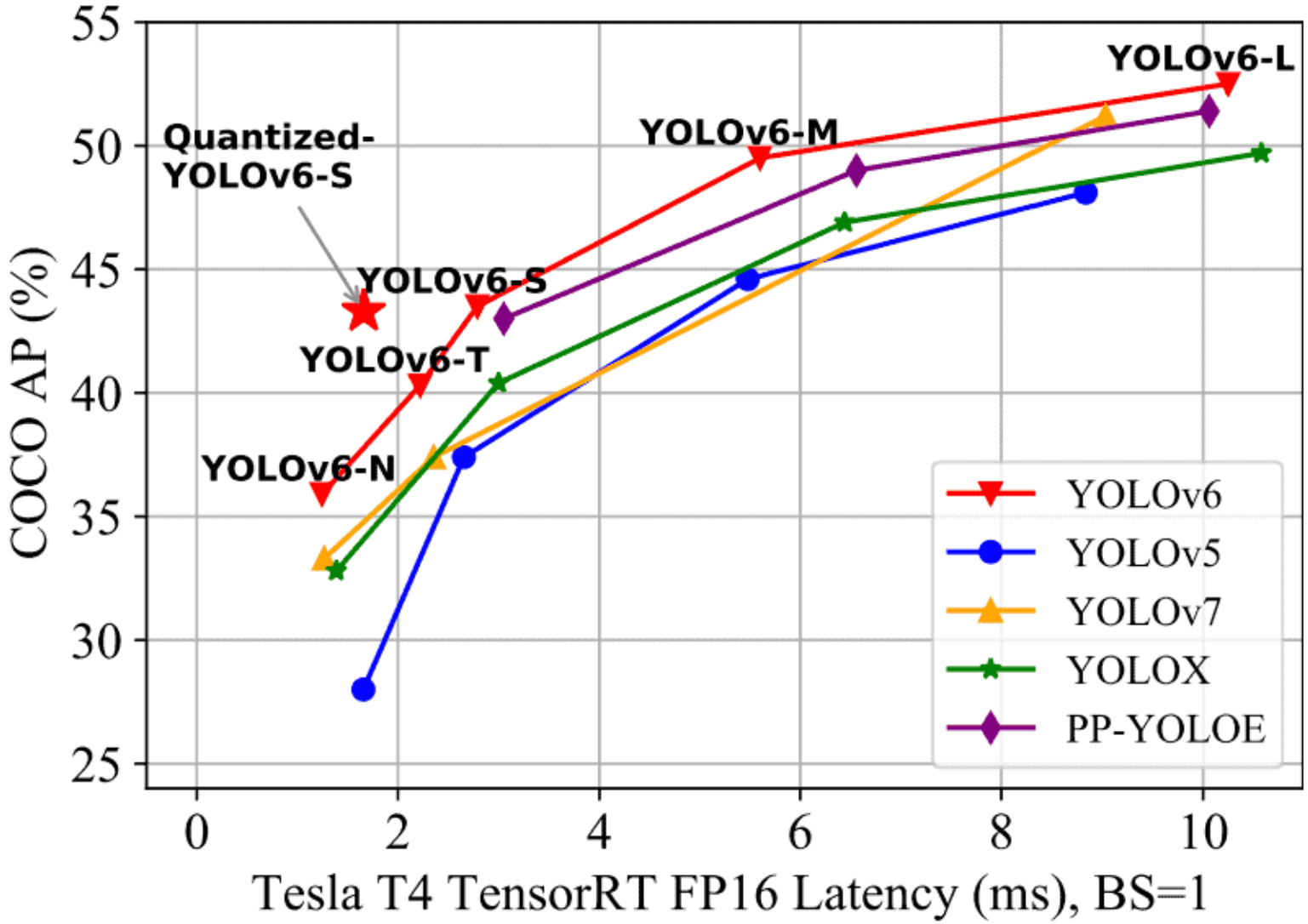
In most object detection models, the neck aggregates the multi-scale feature maps using **PAN (Path Aggregation Networks)**. This is not different in YOLOv6 and similar to what happens in YOLOv4 and YOLOv5.





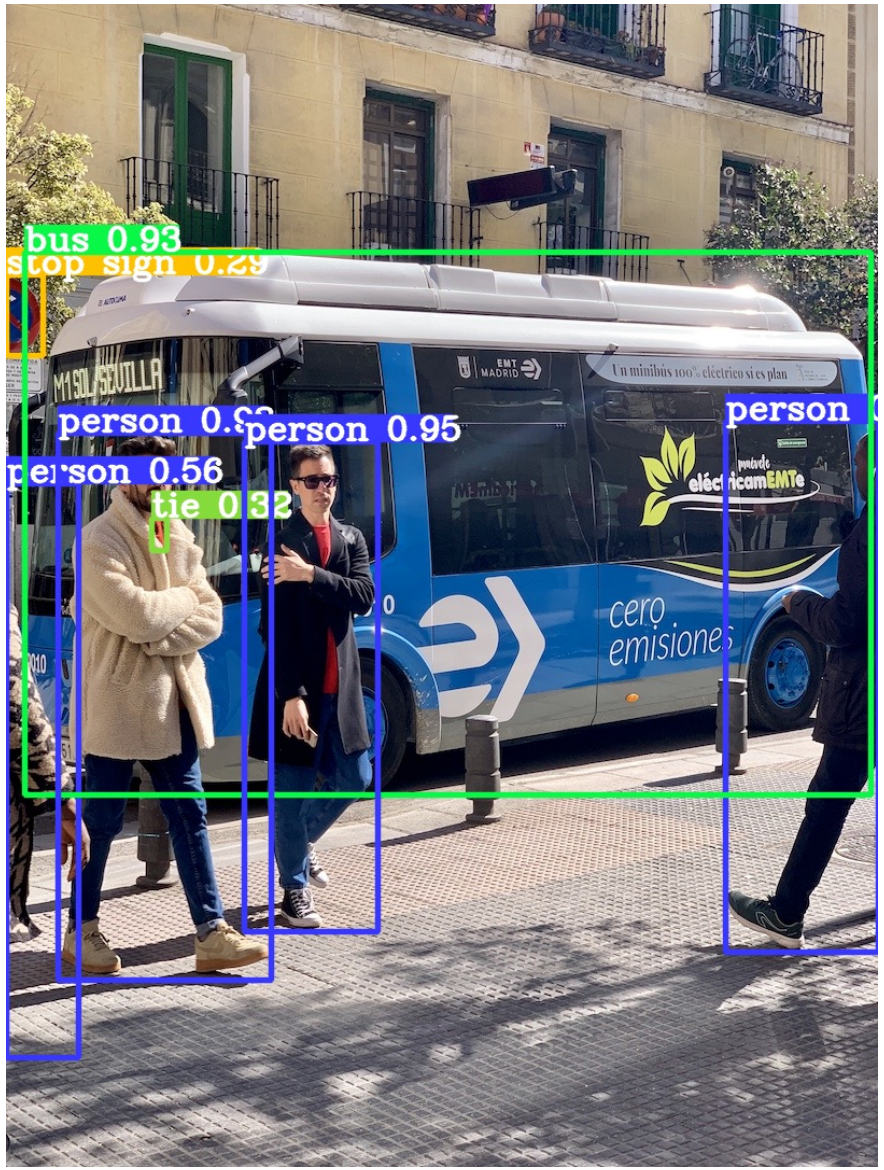
Decoupled Head Design

YOLOv6 Performance



Example

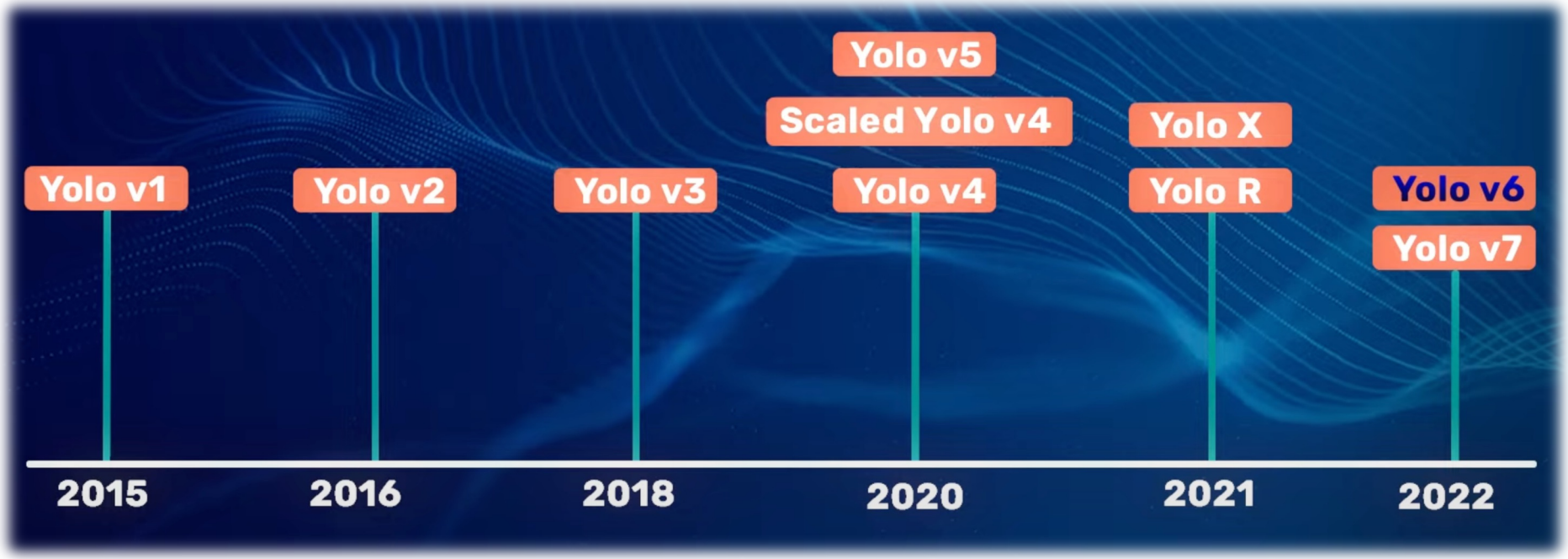
```
[ ]  %cd /content/

     # Clone and install MT-YOLOv6
     !git clone https://github.com/meituan/YOLOv6.git
     !cd YOLOv6
     !pip install -r requirements.txt
```



Outline

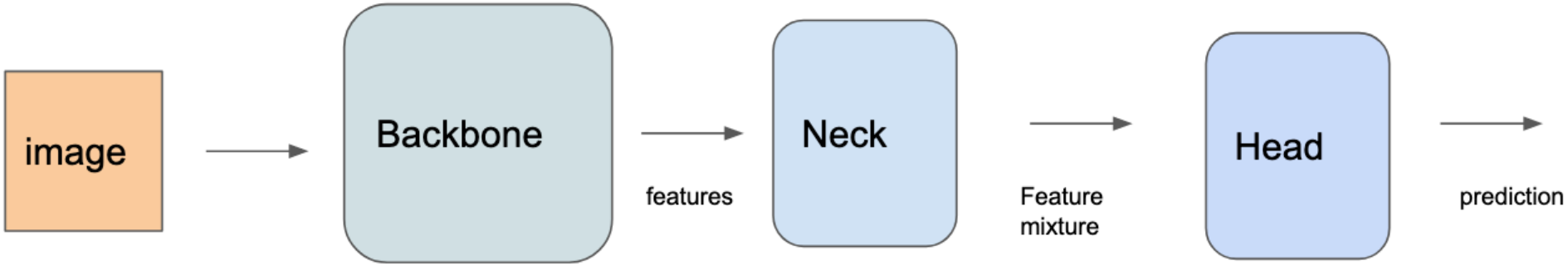
- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

YOLO-v7 Architecture



- **Backbone:** ELAN (YOLOv7-p5, YOLOv7-p6), E-ELAN (YOLOv7-E6E)
- **Neck:** SPPCSPC + (CSP-OSA)PANet (YOLOv7-p5, YOLOv7-p6) + RepConv
- **Head:** YOLOR + Auxiliary Head (YOLOv7-p6)
- The architecture is derived from YOLOv4, Scaled YOLOv4, and YOLO-R. Using these models as a base, further experiments were carried out to develop new and improved YOLOv7.

YOLO-v7 Architecture

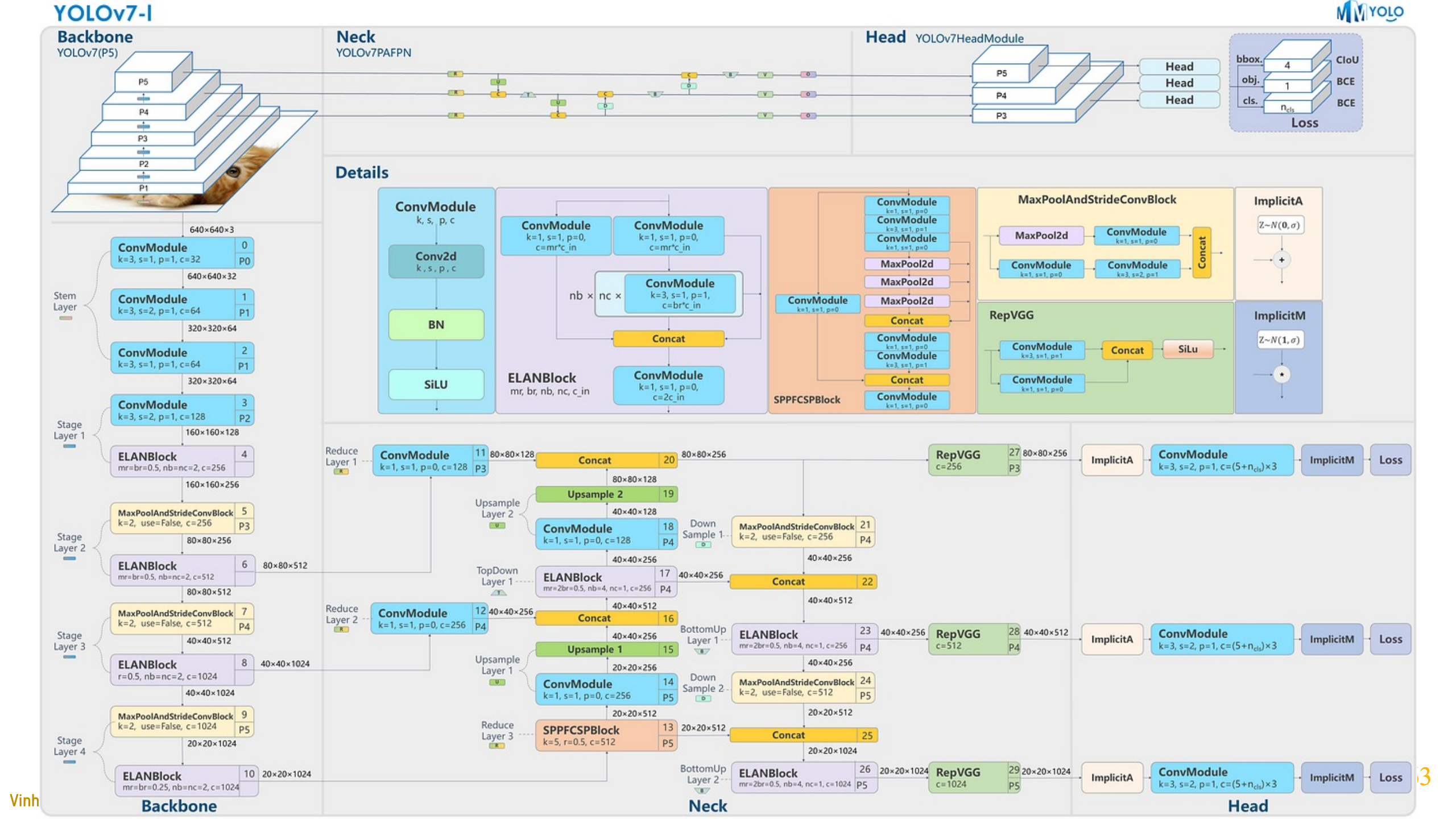


❑ Architectural Reforms

- E-ELAN (Extended Efficient Layer Aggregation Network) Model Scaling for Concatenation-based Models

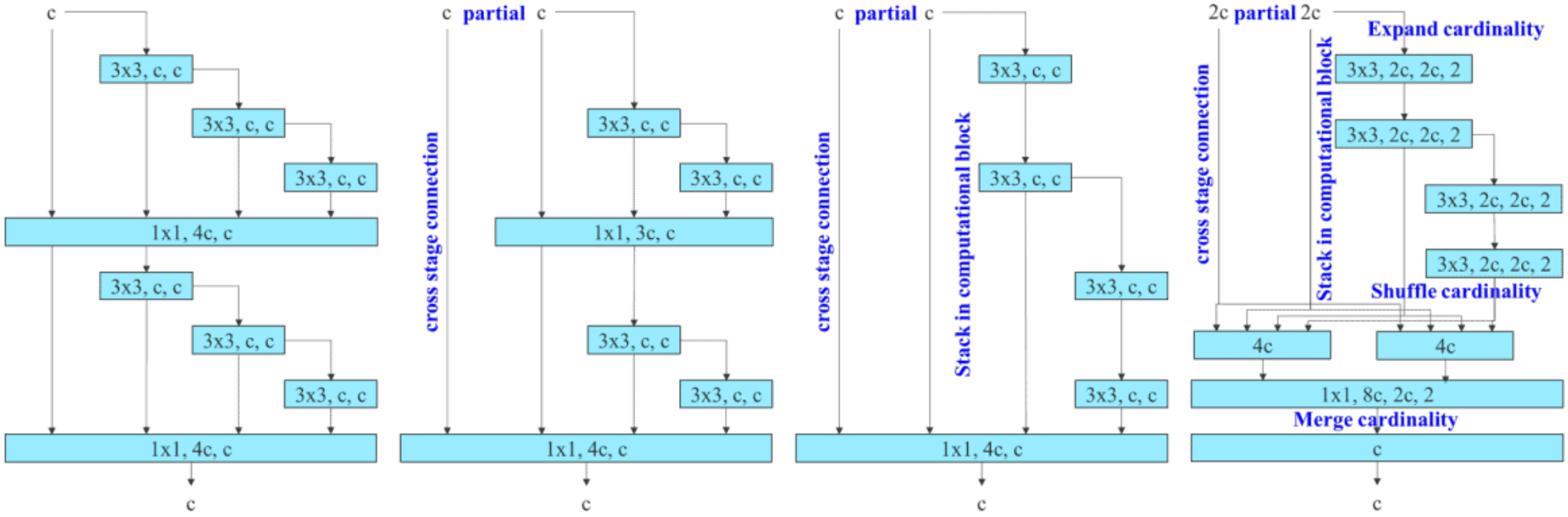
❑ Trainable BoF (Bag of Freebies)

- Planned re-parameterized convolution Coarse for auxiliary and Fine for lead loss


YOLOv7-l
Backbone
YOLOv7(P5)
Neck
YOLOv7PAFPN
Head
YOLOv7HeadModule
Head
Head
Head
bbox.
4
CIoU
obj.
1
BCE
cls.
BCE
Loss
Details
ConvModule
k, s, p, c
Conv2d
k , s , p , c
BN
SiLU
ELANBlock
mr, br, nb, nc, c_in
SPPFCSPBlock
MaxPoolAndStrideConvBlock
RepVGG
ImplicitA
ImplicitM
640×640×3
Stem Layer
Stage Layer 1
Stage Layer 2
Stage Layer 3
Stage Layer 4
Reduce Layer 1
Reduce Layer 2
Upsample Layer 2
Upsample Layer 1
TopDown Layer 1
Down Sample 1
Down Sample 2
BottomUp Layer 1
BottomUp Layer 2
Reduce Layer 3
Backbone
Neck
Head

YOLO-v7 Backbone

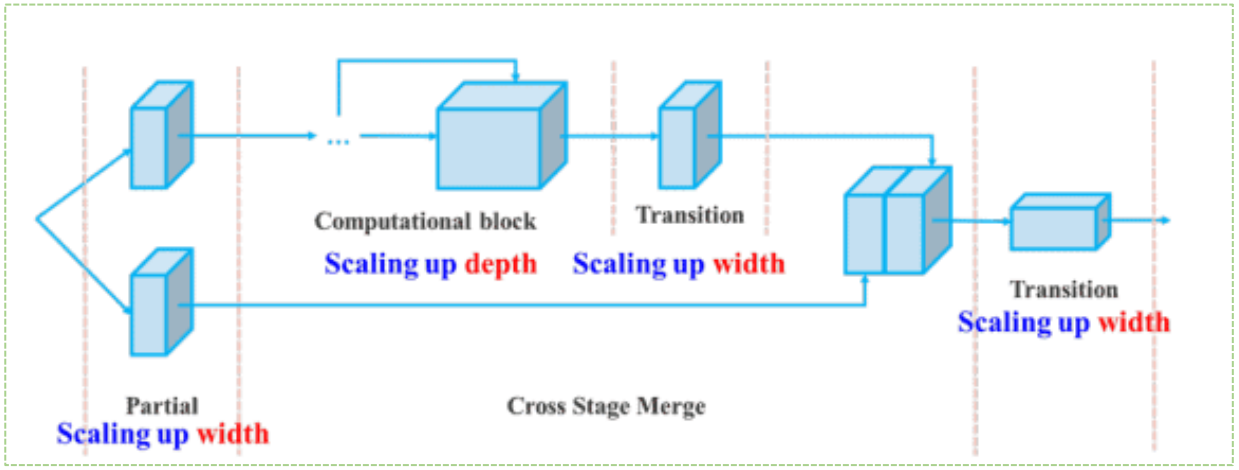
E-ELAN (Extended Efficient Layer Aggregation Network) in YOLOv7 paper

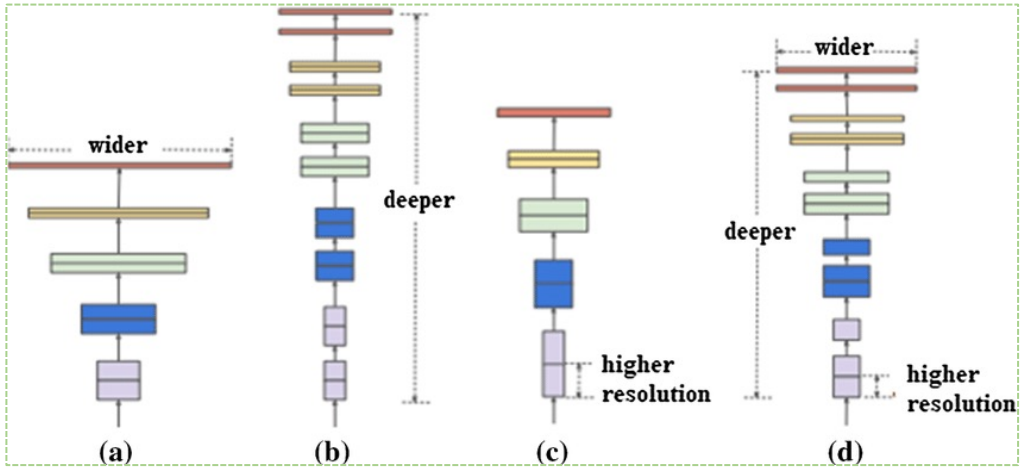


YOLOv7 : Compound Model Scaling

While scaling a model size, the following parameters are considered.

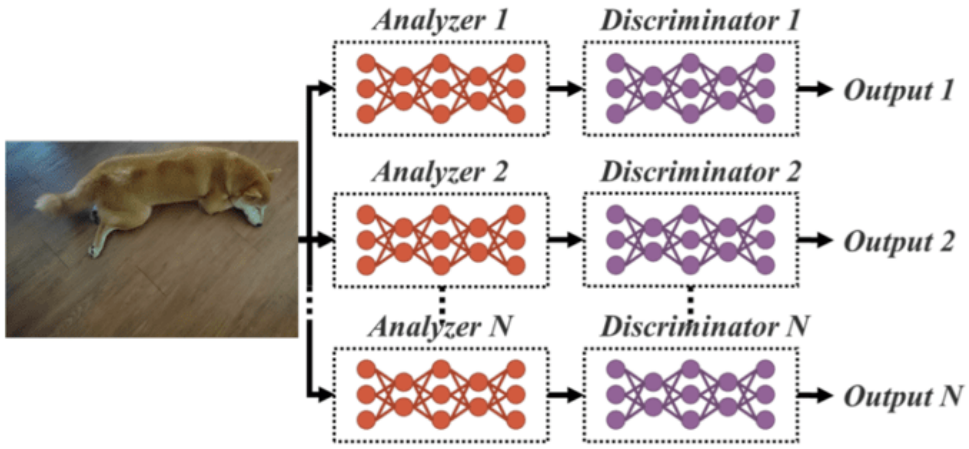
- ❑ Resolution (size of the input image)
- ❑ Width (number of channels)
- ❑ Depth (number of layers)
- ❑ Stage (number of feature pyramids)



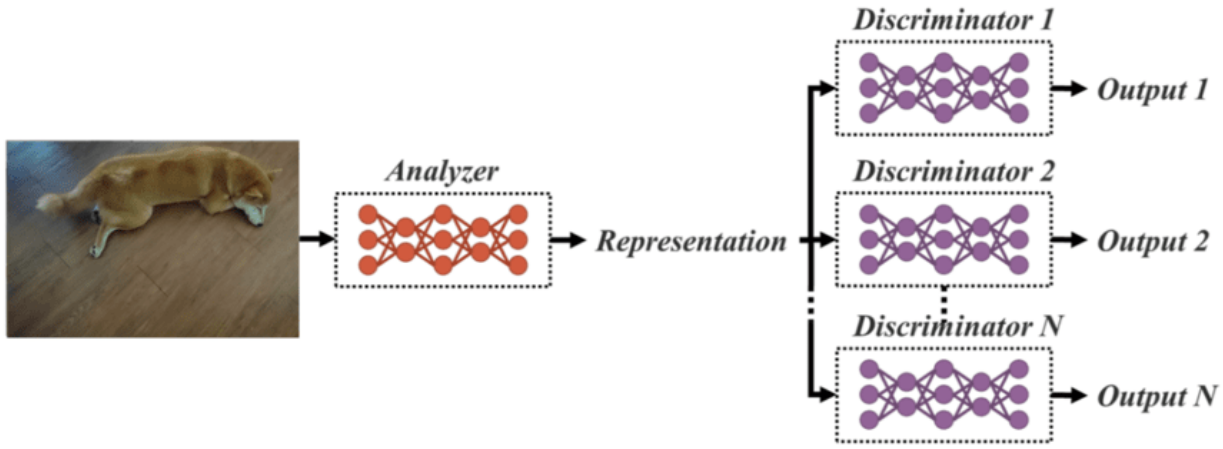


Compound scaling của EfficientNet

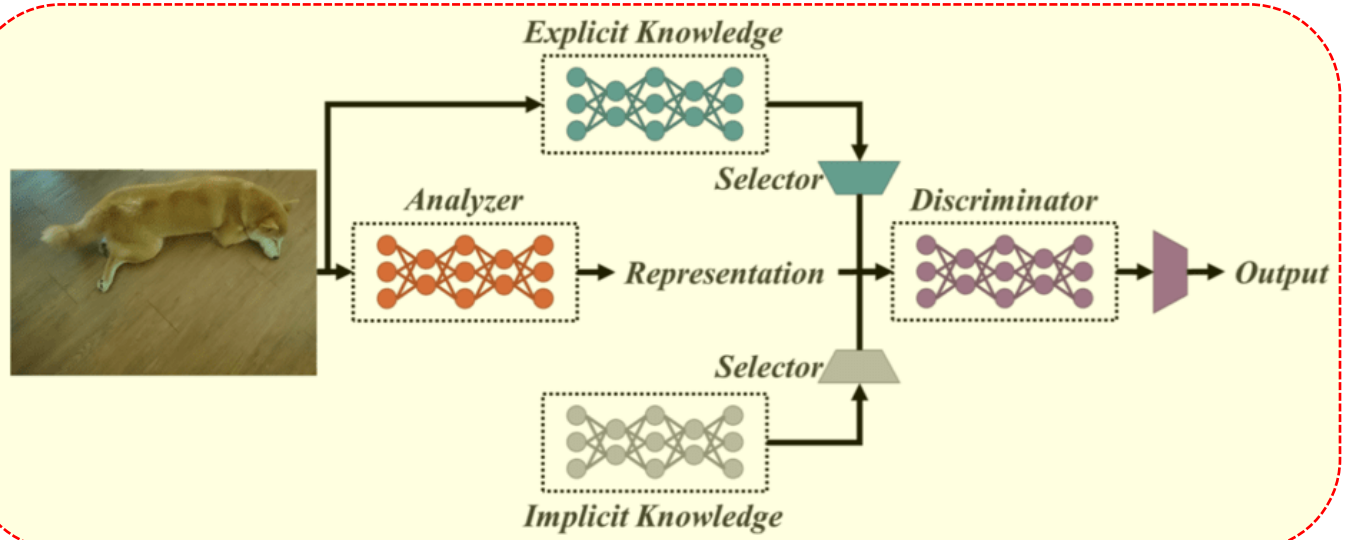
YOLOv7 Head : YOLO-R

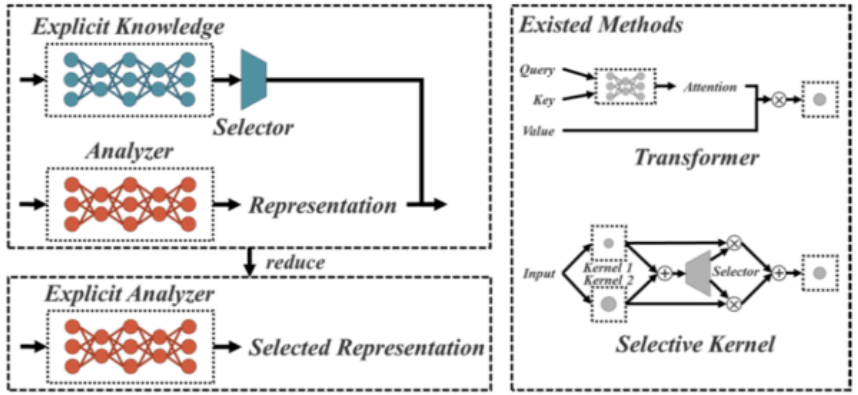


(a) Multi-purpose, multi-analyzer, multi-discriminator.



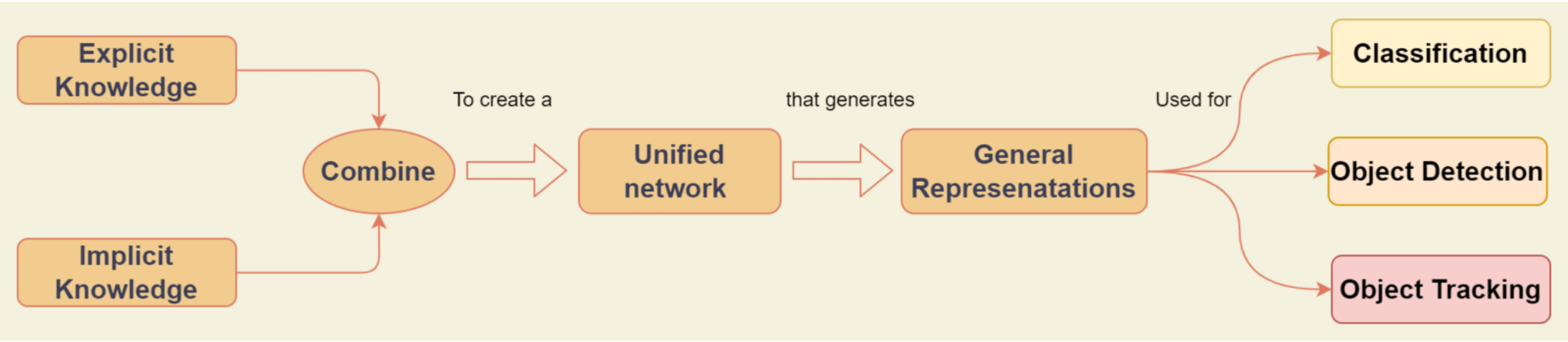
(b) Multi-purpose, single analyzer, multi-discriminator.





(c) Multi-purpose, single unified network.

YOLOv7 Head : YOLO-R



Combining explicit and implicit knowledge in YoloR for general feature generation

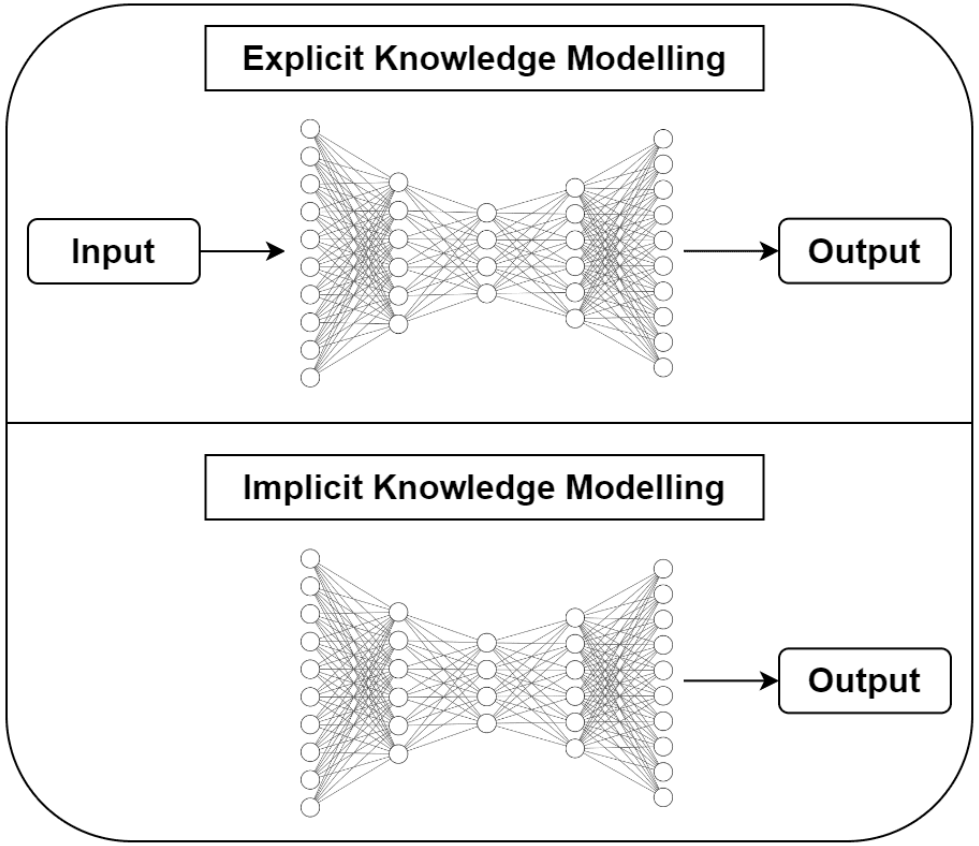
•**Explicit Knowledge:** The knowledge that is easy to articulate, write down, and share with others is known as explicit knowledge. It can be codified and digitized in books, documents, reports, memos, etc.

•**Implicit knowledge:** The knowledge gained from experience are more difficult to express and deeply embedded in the brain. For example, skills learned on a job that is transferable to different positions/jobs, i.e., knowledge acquired from experience.

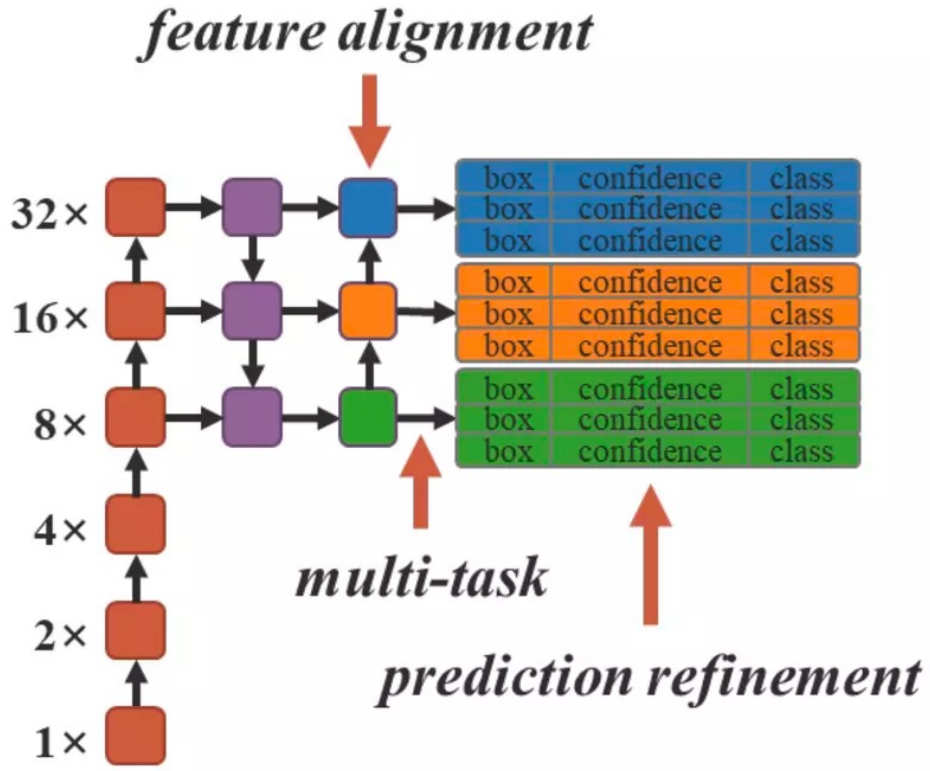
YOLOv7 Head : YOLO-R

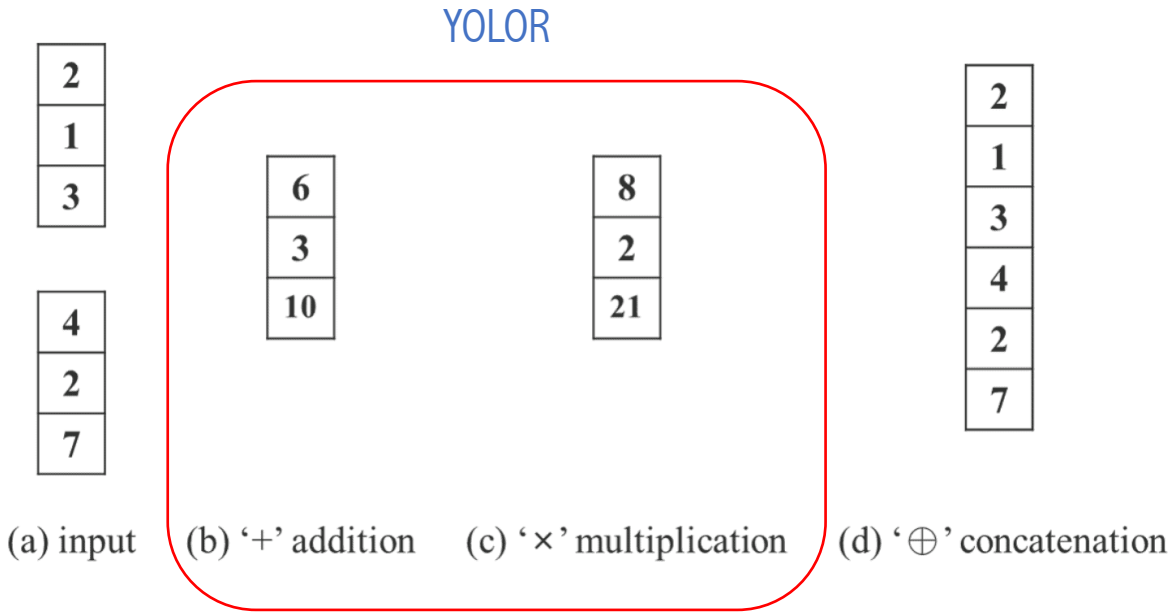
•**Explicit Knowledge:** The knowledge that is easy to articulate, write down, and share with others is known as explicit knowledge. It can be codified and digitized in books, documents, reports, memos, etc.

•**Implicit knowledge:** The knowledge gained from experience are more difficult to express and deeply embedded in the brain. For example, skills learned on a job that is transferable to different positions/jobs, i.e., knowledge acquired from experience.



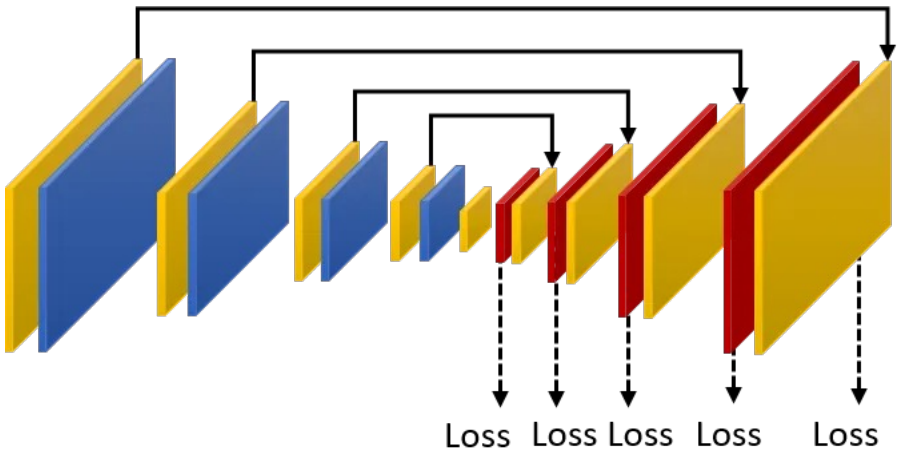
YOLOv7 Head : YOLO-R



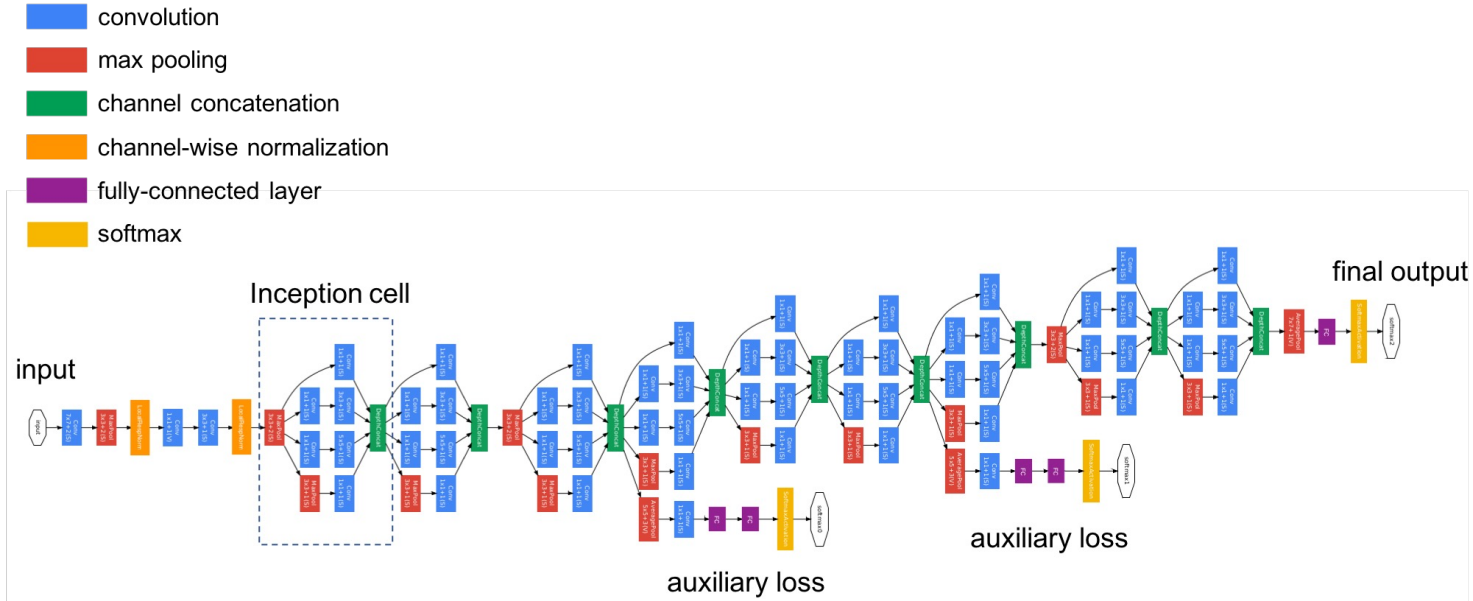


combine explicit and implicit knowledge

YOLOv7 Head : Auxiliary Head



Deep supervision: Auxiliary Head



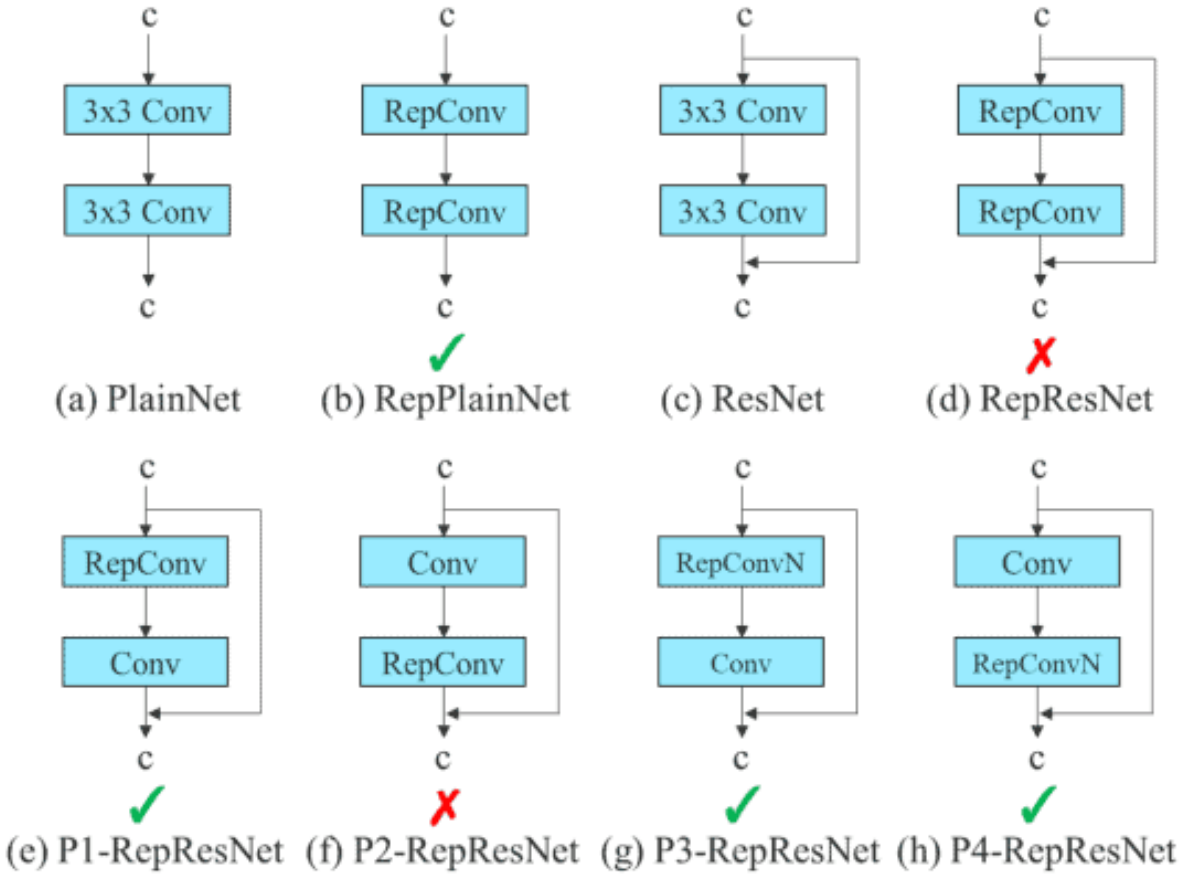

```
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

Google Inception Net

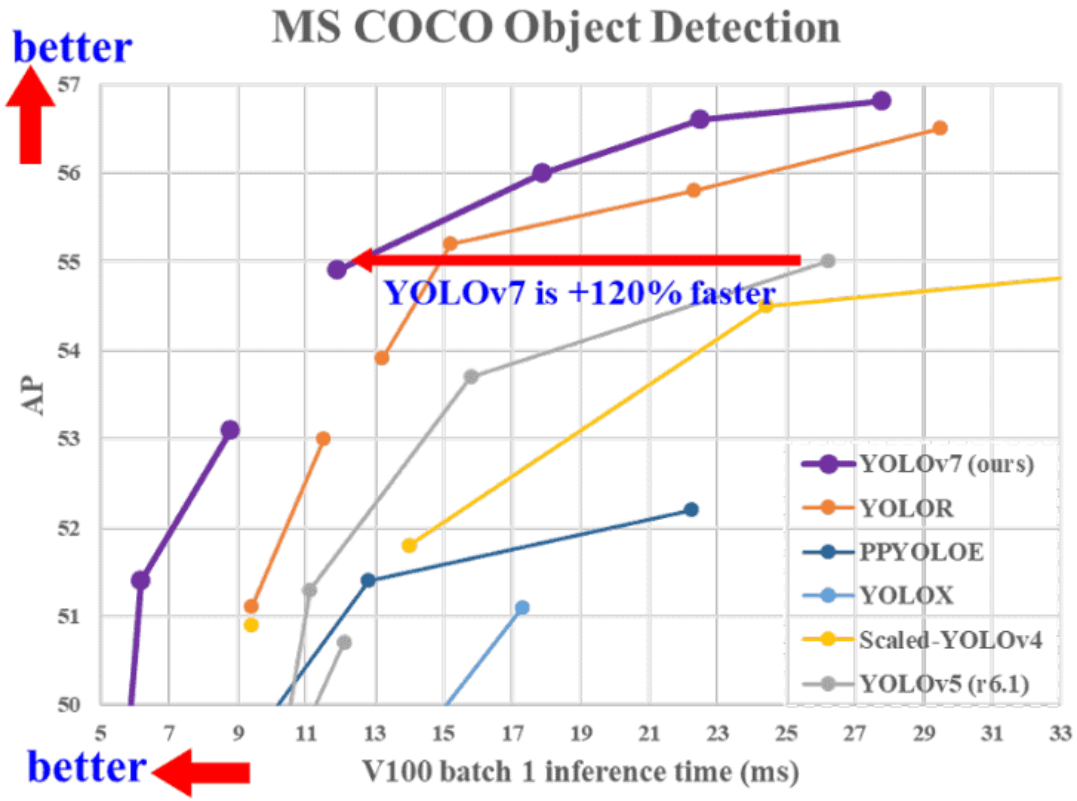
YOLOv7: BOF

Planned Re-parameterized Convolution

In YOLOv7, the architecture of planned re-parameterized convolution uses RepConv without identity connection (RepConvN). Read more at: https://viso.ai/deep-learning/yolov7-guide/



YOLOv7 Performance



Model	Parameters (million)	FPS	AP test (%)
YOLO7-Tiny	6.2	286	38.7
YOLOv7	36.9	161	51.4
YOLOv7-X	71.3	114	53.1
YOLOv7-W6	70.04	84	54.9
YOLOv7-E6	**97.2**	**56**	**56.0**
YOLOv7-D6	154.7	44	56.6
YOLOv7-E6E	151.7	36	56.8

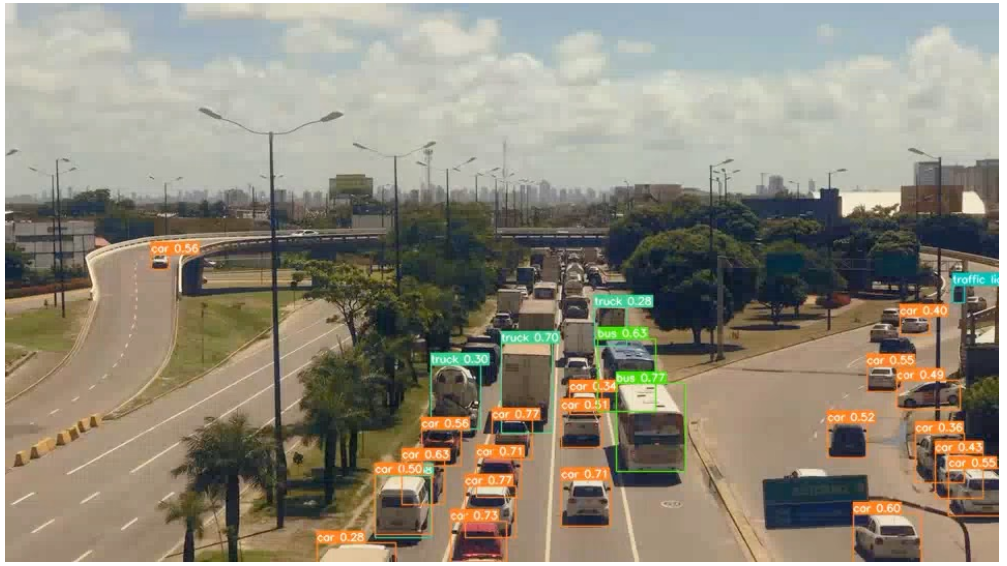
YOLOv7 Performance

Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
YOLOv4 [3]	64.4M	142.8G	640	49.7%	68.2%	54.3%	32.9%	54.8%	63.7%
YOLOR-u5 (r6.1) [81]	46.5M	109.1G	640	50.2%	68.7%	54.6%	33.2%	55.5%	63.7%
YOLOv4-CSP [79]	52.9M	120.4G	640	50.3%	68.6%	54.9%	34.2%	55.6%	65.1%
YOLOR-CSP [81]	52.9M	120.4G	640	50.8%	69.5%	55.3%	33.7%	56.0%	65.4%
YOLOv7	36.9M	104.7G	640	**51.2%**	**69.7%**	**55.5%**	**35.2%**	**56.0%**	**66.7%**
improvement	-43%	-15%	-	+0.4	+0.2	+0.2	+1.5	=	+1.3
YOLOR-CSP-X [81]	96.9M	226.8G	640	52.7%	**71.3%**	57.4%	36.3%	57.5%	68.3%
YOLOv7-X	71.3M	189.9G	640	**52.9%**	71.1%	**57.5%**	**36.9%**	**57.7%**	**68.6%**
improvement	-36%	-19%	-	+0.2	-0.2	+0.1	+0.6	+0.2	+0.3
YOLOv4-tiny [79]	6.1	6.9	416	24.9%	42.1%	25.7%	8.7%	28.4%	39.2%
YOLOv7-tiny	6.2	5.8	416	**35.2%**	**52.8%**	**37.3%**	**15.7%**	**38.0%**	**53.4%**
improvement	+2%	-19%	-	+10.3	+10.7	+11.6	+7.0	+9.6	+14.2
YOLOv4-tiny-3l [79]	8.7	5.2	320	30.8%	47.3%	32.2%	**10.9%**	31.9%	51.5%
YOLOv7-tiny	6.2	3.5	320	**30.8%**	**47.3%**	**32.2%**	10.0%	**31.9%**	**52.2%**
improvement	-39%	-49%	-	=	=	=	-0.9	=	+0.7
YOLOR-E6 [81]	115.8M	683.2G	1280	55.7%	73.2%	60.7%	40.1%	**60.4%**	69.2%
YOLOv7-E6	97.2M	515.2G	1280	**55.9%**	**73.5%**	**61.1%**	**40.6%**	60.3%	**70.0%**
improvement	-19%	-33%	-	+0.2	+0.3	+0.4	+0.5	-0.1	+0.8
YOLOR-D6 [81]	151.7M	935.6G	1280	56.1%	73.9%	61.2%	**42.4%**	60.5%	69.9%
YOLOv7-D6	154.7M	806.8G	1280	56.3%	73.8%	61.4%	41.3%	60.6%	70.1%
YOLOv7-E6E	151.7M	843.2G	1280	**56.8%**	**74.4%**	**62.1%**	40.8%	**62.1%**	**70.6%**
improvement	=	-11%	-	+0.7	+0.5	+0.9	-1.6	+1.6	+0.7

YOLOv7 Performance

Yolov5

Yolov4



Yolov7

Models	Video 1		Video 2		Video 3	
	Inference time(ms)	FPS	Inference time(ms)	FPS	Inference time(ms)	FPS
YOLOv4		14.8		14.8		14.5
YOLOv4 Tiny		153		115		96.6
YOLOv7	53	18.75	49.5	20.2	50	20
YOLOv7 Tiny	23.87	41.89	23.08	43.31	22.78	43.88
YOLOv5 Large	28.00	35.71	28.00	35.71	8.81	35.48
Yolov5 Tiny	7.00	142.85	7.90	126.58	8.20	121.95

https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/

Example

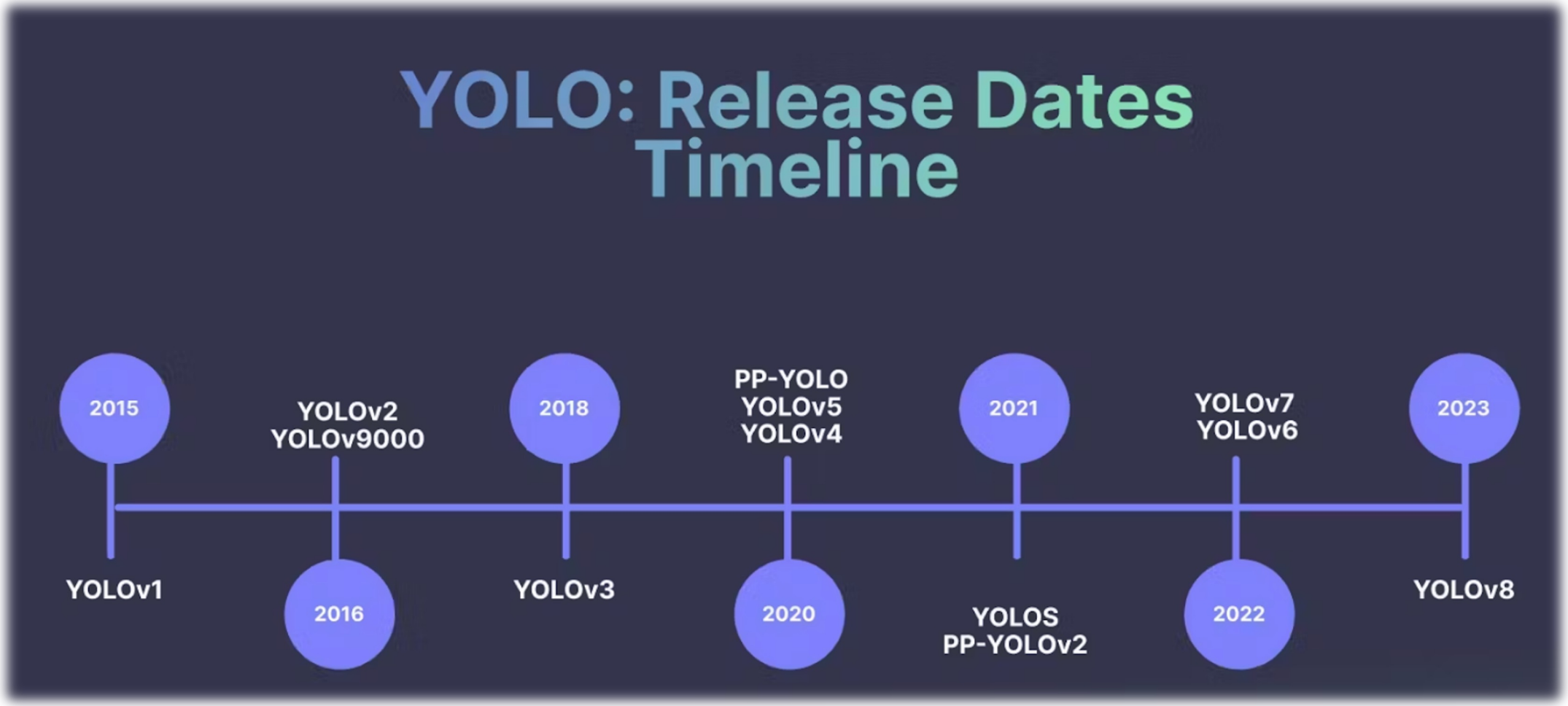
```
%cd /content/

# Clone the github
!git clone https://github.com/WongKinYiu/yolov7

# # Dowbload the sample
!wget https://ultralytics.com/images/bus.jpg

%cd yolov7

# Download trained weights
!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7-tiny.pt
```

```
[ ] !python detect.py --weights ./yolov7-tiny.pt --conf 0.25 --img-size 640 --source /content/bus.jpg
```

Outline

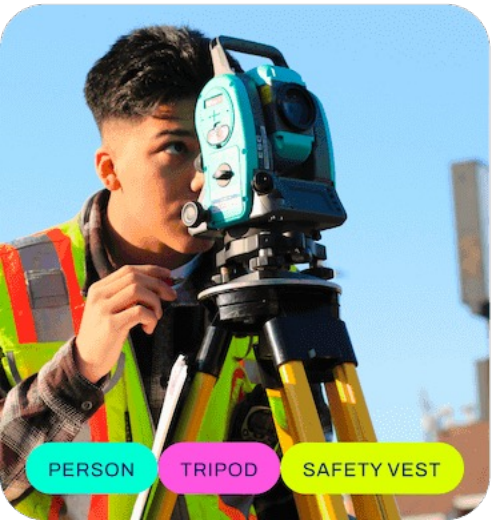
- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

YOLOv8

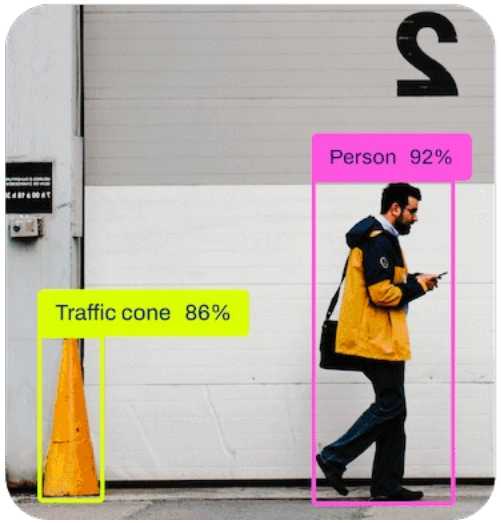

YOLO: Release Dates Timeline
2015
YOLOv1
YOLOv2
YOLOv9000
2016
2018
YOLOv3
PP-YOLO
YOLOv5
YOLOv4
2020
2021
YOLOS
PP-YOLOv2
YOLOv7
YOLOv6
2022
2023
YOLOv8

YOLOv8

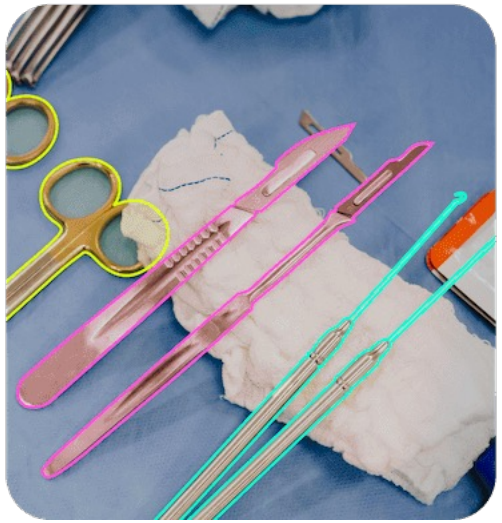
Classify



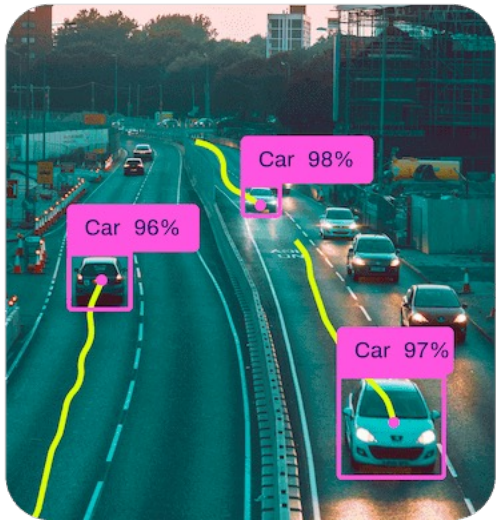
Detect



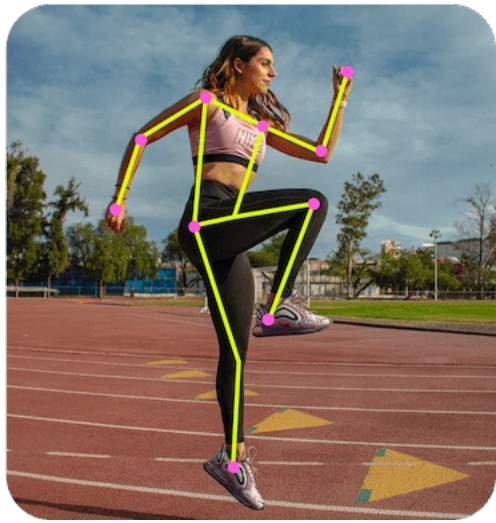
Segment



Track



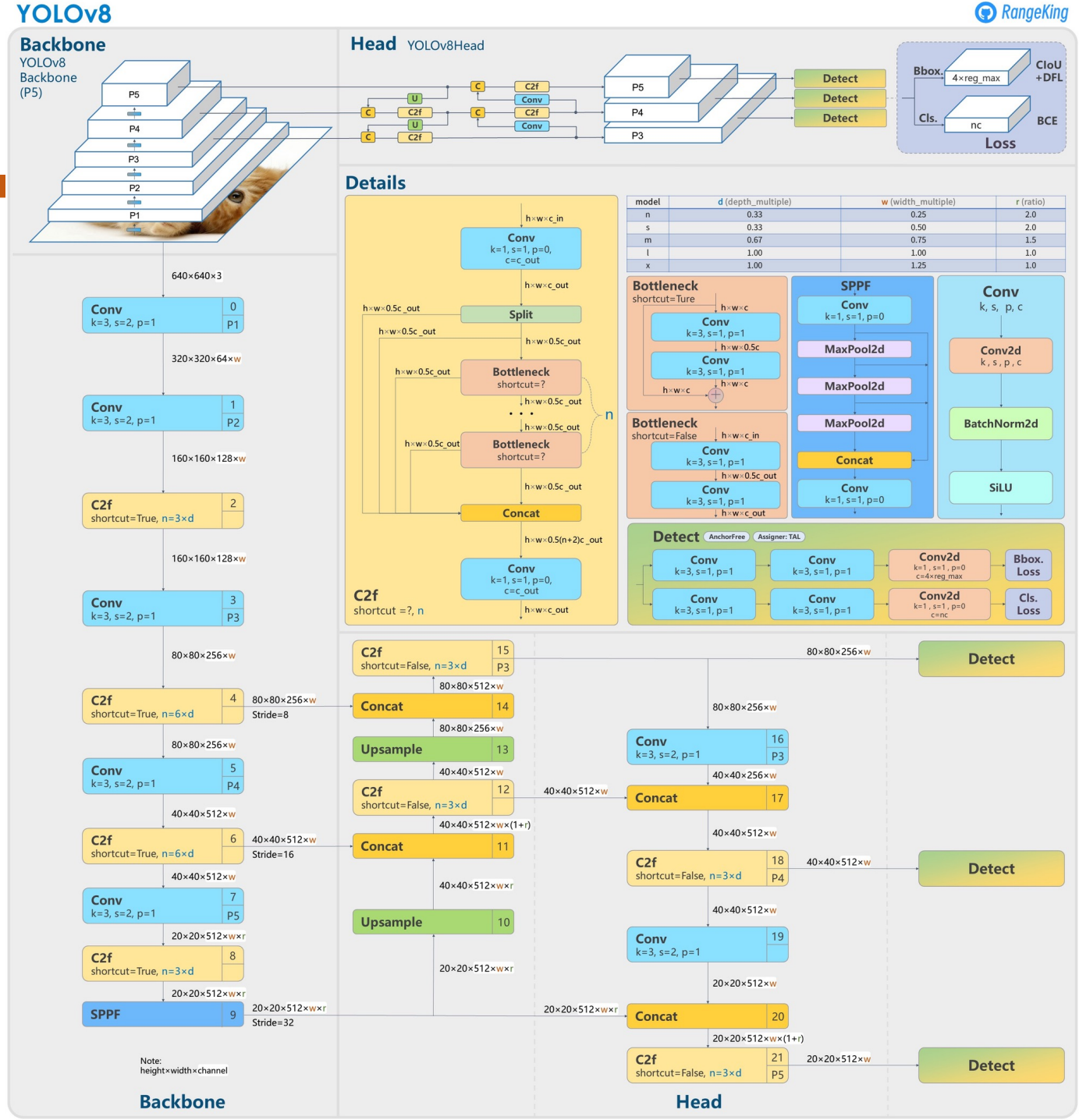
Pose



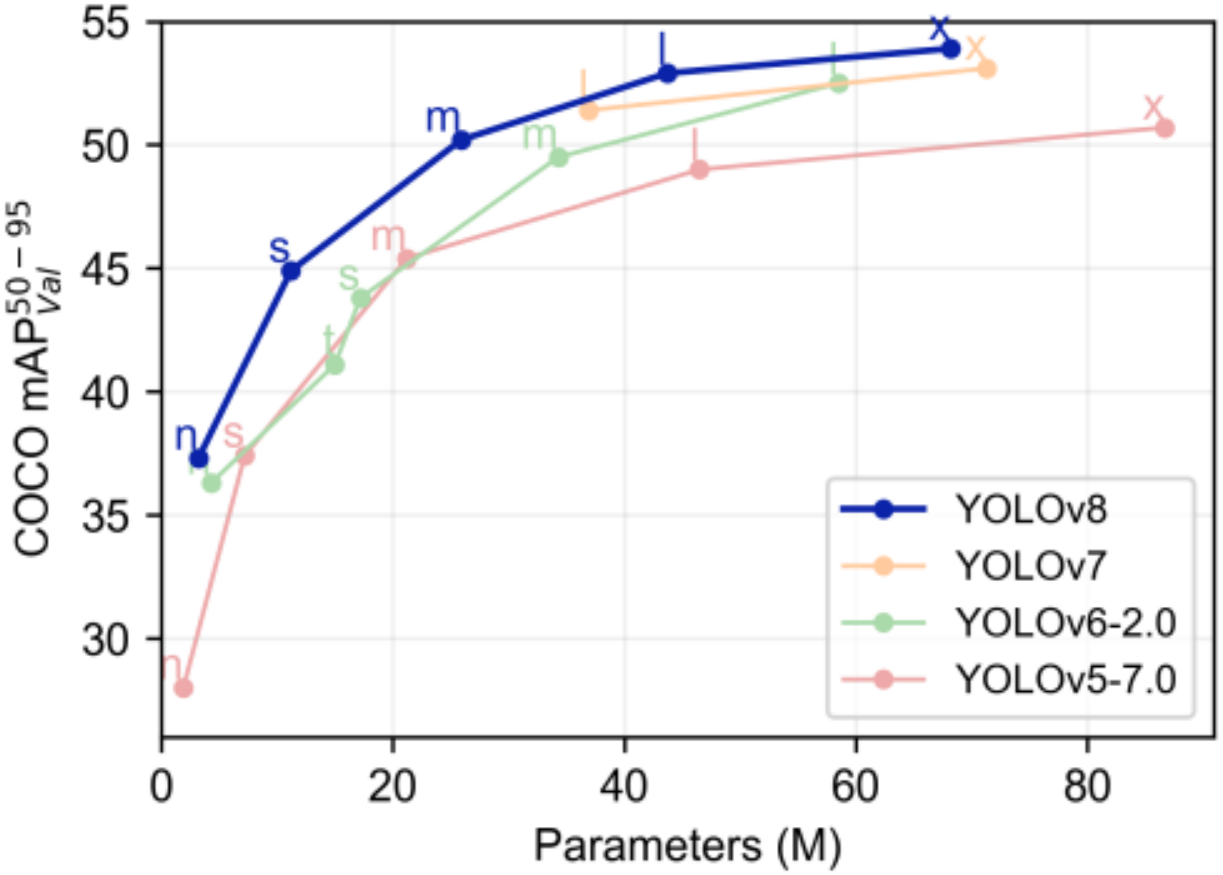
AI VIETNAM
All-in-One Course

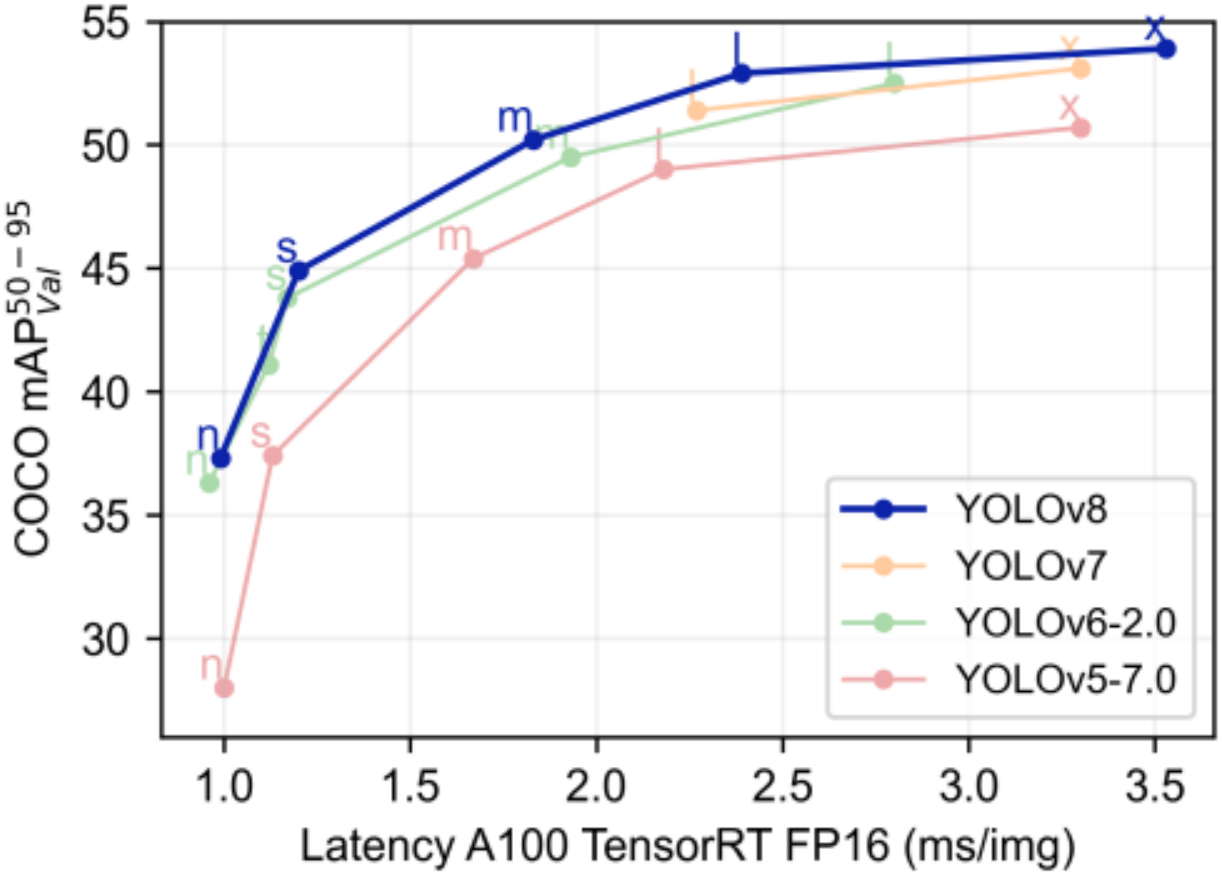
Yolov8

- The **C3 modules** have been replaced with **C2f modules**.
- The first **6×6 Conv** has been replaced with **3×3 Conv** in the **Backbone**.
- Usage of decoupled head and deletion of the **Objectness branch**.
- The first **1×1 Conv in Backbone** has been replaced with **3×3 Conv**.



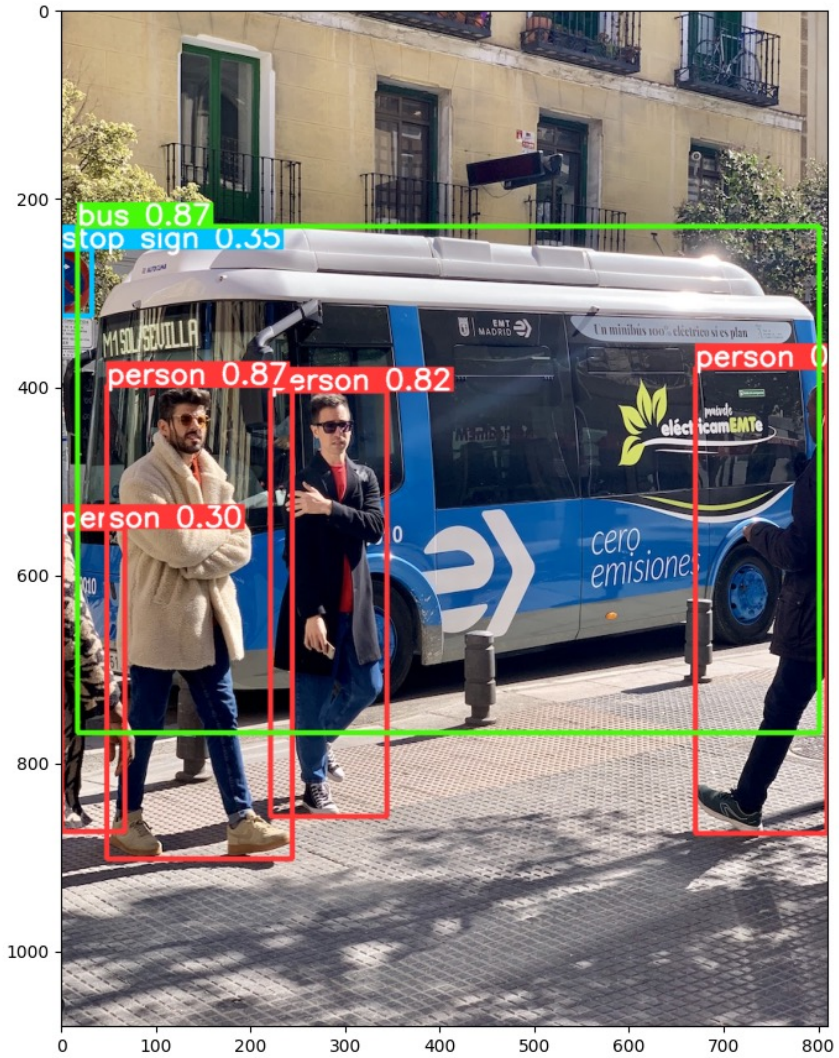
YOLO-v8 Performance





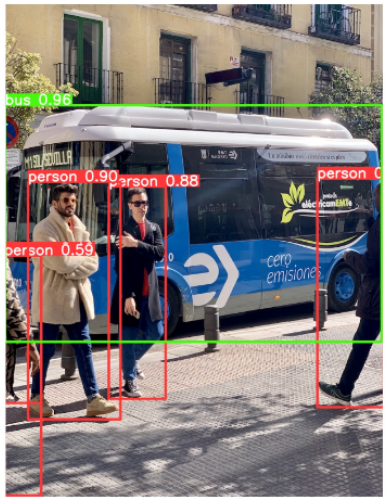
Example

```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load an official model

# Predict with the model
results = model('https://ultralytics.com/images/bus.jpg')

show_img(results)
```



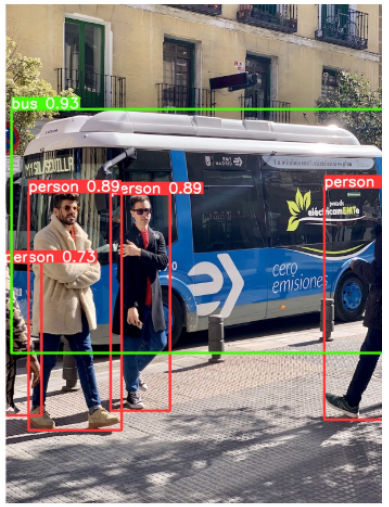
Performance Evaluation: YOLOs v3 - v8

YOLOv3

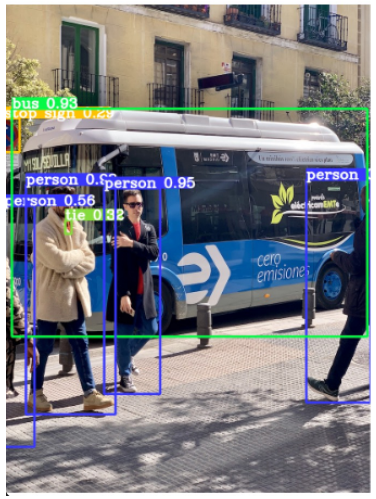


YOLOv4

YOLOv5

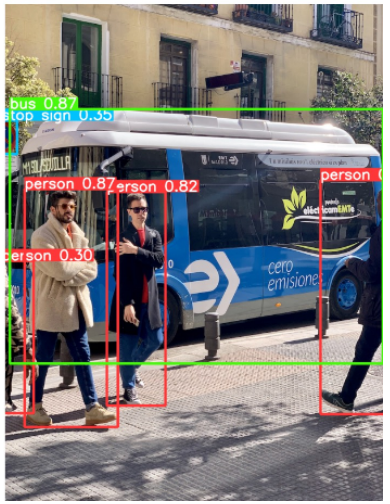


YOLOv6



YOLOv7

YOLOv8



Outline

- Object Detection Milestones: One-stage Detectors
- YOLOv1 and YOLOv2 Review
- YOLOv3
- YOLOv4
- YOLOv5
- YOLO X
- YOLOv6
- YOLOv7
- YOLOv8
- Further study

Group DETR: Fast DETR Training with Group-Wise One-to-Many Assignment

Qiang Chen[1*], Xiaokang Chen[2*], Jian Wang[1], Shan Zhang[3]
Kun Yao[1], Haocheng Feng[1], Junyu Han[1], Errui Ding[1], Gang Zeng[2], Jingdong Wang[1†]
[1]Baidu VIS
[2]Key Lab. of Machine Perception (MoE), School of IST, Peking University
[3]Australian National University
{chenqiang13,wangjian33}@baidu.com
{fenghaocheng,hanjunyu,dingerrui,wangjingdong}@baidu.com
{pkucxk,gang.zeng}@pku.edu.cn, shan.zhang@anu.edu.au

Abstract

Detection transformer (DETR) relies on one-to-one assignment, assigning one ground-truth object to one prediction, for end-to-end detection without NMS post-processing. It is known that one-to-many assignment, assigning one ground-truth object to multiple predictions, succeeds in detection methods such as Faster R-CNN and FCOS. While the naive one-to-many assignment does not work for DETR, and it remains challenging to apply one-to-many assignment for DETR training. In this paper, we introduce Group DETR, a simple yet efficient DETR training approach that introduces a group-wise way for one-to-many assignment. This approach involves using multiple groups of object queries, conducting one-to-one assignment within each group, and performing decoder self-attention separately. It resembles data augmentation with automatically-learned object query augmentation. It is also equivalent to simultaneously training parameter-sharing networks of the same architecture, introducing more supervision and thus improving DETR training. The inference process is the same as DETR trained normally and only needs one group of queries without any architecture modification. Group DETR is versatile and is applicable to various DETR variants. The experiments show that Group DETR significantly speeds up the training convergence and improves the performance of various DETR-based models. Code will be available at https://github.com/Atten4Vis/GroupDETR.

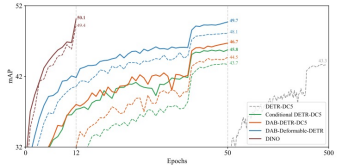


Figure 1. **Group DETR accelerates the training process for DETR variants.** The training convergence curves are obtained on COCO *val2017* [34] with ResNet-50 [22]. Dashed and bold curves correspond to the baseline models and the Group DETR counterparts. Best viewed in color.

crafted components, such as non-maximum suppression (NMS) [23] and anchor generation [44, 33, 43]. The architecture consists of a CNN [22] and transformer encoder [53], and a transformer decoder that consists of self-attention, cross-attention and FFNs, followed by class and box prediction FFNs. During training, one-to-one assignment, where one ground-truth object is assigned to one single prediction, is applied for learning to only promote the predictions assigned to ground-truth objects, and demote the duplicate predictions.

This work explores the solutions to accelerate the DETR training process. Previous solutions contain two main lines. The one line is to modify *cross-attention* so that informa-

Group DETR v2: Strong Object Detector with Encoder-Decoder Pretraining

Qiang Chen[1*], Jian Wang[1*], Chuchu Han[1*], Shan Zhang[2], Zexian Li[3], Xiaokang Chen[4], Jiahui Chen[3]
Xiaodi Wang[1], Shuming Han[1], Gang Zhang[1], Haocheng Feng[1], Kun Yao[1], Junyu Han[1], Errui Ding[1]
Jingdong Wang[1†]

[1]Baidu VIS [2]Australian National University [3]Beihang University [4]Peking University

Table 1. **Our method establishes a new SoTA on the COCO *test-dev* leaderboard.**

Method	#Params	Encoder Pretraining Data	Detector Pretraining Data	w/ Mask	mAP
Swin-L (HTC++) [16]	284M	IN-22K (14M)	n/a	✓	58.7
DyHead (Swin-L) [6]	213M	IN-22K (14M)	n/a	✓	60.6
Soft-Teacher (Swin-L) [25]	284M	IN-22K (14M)	COCO-unlabeled + O365	✓	61.3
GLIP (DyHead) [11]	≥284M	IN-22K (14M)	FourODs + GoldG + Cap24M	×	61.5
Florence (CoSwin-H) [29]	≥637M	FLD-900M (900M)	FLD-9M	×	62.4
GLIPv2 (CoSwin-H) [29]	≥637M	FLD-900M (900M)	FourODs + INBoxes + GoldG + CC15M + SBU	✓	62.4
SwinV2-G (HTC++) [15]	3.0B	IN-22K + ext-70M (84M)	O365	✓	63.1
DINO (Swin-L) [28]	218M	IN-22K (14M)	O365	×	63.3
BEIT-3 (ViTDet) [22]	1.9B	IN-22K + Image-Text (35M) + Text (160GB)	O365	✓	63.7
FD-SwinV2-G (HTC++) [23]	3.0B	IN-22K + IN-1K + ext-70M (85M)	O365	✓	64.2
FocalNet-H (DINO) [26]	746M	IN-22K (14M)	O365	×	64.3
Group DETR v2 (Our method)	629M	**IN-1K (1M)**	O365	×	**64.5**

All the results are achieved with test time augmentation. In the table, we follow the notations for various datasets used in DINO [28] and FocalNet [26]. 'w/ Mask' means using mask annotations when finetuning the detectors on COCO [13].

Abstract

*We present a strong object detector with encoder-decoder pretraining and finetuning. Our method, called Group DETR v2, is built upon a vision transformer encoder ViT-Huge [8], a DETR variant DINO [28], and an efficient DETR training method Group DETR [3]. The training process consists of self-supervised pretraining and finetuning a ViT-Huge encoder on ImageNet-1K, pretraining the detector on Object365, and finally finetuning it on COCO. Group DETR v2 achieves **64.5** mAP on COCO test-dev, and establishes a new SoTA on the COCO leaderboard*[1].

to achieve superior results on various vision tasks, including object detection. With supervised encoder-decoder pretraining on a large-scale dataset, Object365 [20], DINO [28] achieves a state-of-the-art result on COCO [13].

Our method, *Group DETR v2*, is built upon ViT-Huge, DINO, and Group DETR. We adopt an encoder-decoder pretraining and finetuning pipeline: pretraining and then finetuning a ViT-Huge encoder on ImageNet-1K [7], pretraining the detector, both the encoder and the decoder, on Object365, and finally finetuning it on COCO. Group DETR v2 achieves **64.5** mAP on COCO test-dev [13] (Table 1 and Table 2), setting a new record for COCO object

Detection Transformer with Stable Matching

Shilong Liu[1,2*†], **Tianhe Ren**[2*], **Jiayu Chen**[3*],
Zhaoyang Zeng[2], **Hao Zhang**[2,4], **Feng Li**[2,4], **Hongyang Li**[2,5],
Jun Huang[3], **Hang Su**[1], **Jun Zhu**[1‡], **Lei Zhang**[2‡]

[1] Dept. of Comp. Sci. and Tech., BNRist Center, State Key Lab for Intell. Tech. & Sys., Institute for AI, Tsinghua-Bosch Joint Center for ML, Tsinghua University
[2] International Digital Economy Academy (IDEA) [3] Platform of AI (PAI), Alibaba Group
[4] The Hong Kong University of Science and Technology
[5] South China University of Technology

liusl20@mails.tsinghua.edu.cn {rentianhe, zengzhaoyang}@idea.edu.cn {yunji.cjy, huangjun.hj}@alibaba-inc.com
{hzhangcx, fliay}@connect.ust.hk ftwangyeunglei@mail.scut.edu.cn {suhangss, dcszj}@mail.tsinghua.edu.cn leizhang@idea.edu.cn

Abstract

This paper is concerned with the matching stability problem across different decoder layers in DEtection TRansformers (DETR). We point out that the unstable matching in DETR is caused by a multi-optimization path *problem, which is highlighted by the one-to-one matching design in DETR. To address this problem, we show that the most important design is to* use and only use *positional metrics (like IOU) to supervise classification scores of positive examples. Under the principle, we propose two simple yet effective modifications by integrating positional metrics to DETR's classification loss and matching cost, named position-supervised loss and position-modulated cost. We verify our methods on several DETR variants. Our methods show consistent improvements over baselines. By integrating our methods with DINO, we achieve* 50.4 *and* 51.5 *AP on the COCO detection benchmark using ResNet-50 backbones under* 1× *(12 epochs) and* 2× *(24 epochs) training settings, achieving a new record under the same setting. We achieve **63.8** AP on COCO detection test-dev with a Swin-Large backbone. Our code will be made available at* https://github.com/IDEA-Research/Stable-DINO.

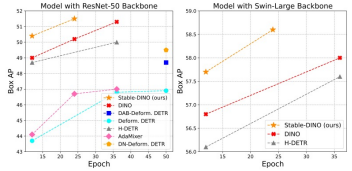


Figure 1: Comparison of our methods (named Stable-DINO in figures) and baselines. We compare models with ResNet-50 backbones in the left figure and models with Swin-Transformer Large backbones in the right figure. All models use a maximum 1/8 resolution feature map from a backbone, except AdaMixer uses a maximum 1/4 resolution feature map.

decades with the development of deep learning, especially the convolutional neural network (CNN) [36, 14, 16, 7].

Detection Transformer (DETR) [3] proposed a novel Transformer-based object detector, which attracted a lot of interest in the research community. It gets rid of the

Further study

DETRs with Collaborative Hybrid Assignments Training

Zhuofan Zong Guanglu Song Yu Liu*
SenseTime Research
{zongzhuofan,liuyuisanai}@gmail.com
songguanglu@sensetime.com

Abstract

In this paper, we provide the observation that too few queries assigned as positive samples in DETR with one-to-one set matching leads to sparse supervision on the encoder's output which considerably hurt the discriminative feature learning of the encoder and vice visa for attention learning in the decoder. To alleviate this, we present a novel collaborative hybrid assignments training scheme, namely Co-DETR, to learn more efficient and effective DETR-based detectors from versatile label assignment manners. This new training scheme can easily enhance the encoder's learning ability in end-to-end detectors by training the multiple parallel auxiliary heads supervised by one-to-many label assignments such as ATSS and Faster RCNN. In addition, we conduct extra customized positive queries by extracting the positive coordinates from these auxiliary heads to improve the training efficiency of positive samples in the decoder. In inference, these auxiliary heads are discarded and thus our method introduces no additional parameters and computational cost to the original detector while requiring no hand-crafted non-maximum suppression (NMS). We conduct extensive experiments to evaluate the effectiveness of the proposed approach on DETR variants, including DAB-DETR, Deformable-DETR, and DINO-Deformable-DETR. The state-of-the-art DINO-Deformable-DETR with Swin-L can be improved from 58.5% to 59.5% AP on COCO val. Surprisingly, incorporated with ViT-L backbone, we achieve 66.0% AP on COCO test-dev and 67.9% AP on LVIS val, outperforming previous methods by clear margins with much fewer model sizes. Codes are available at https://github.com/Sense-X/Co-DETR.

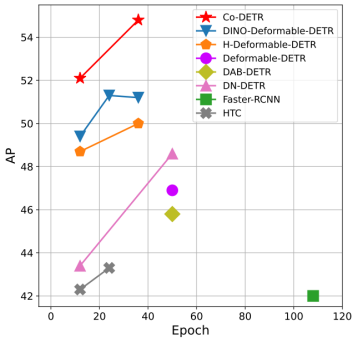


Figure 1. Performance of models with ResNet-50 on COCO `val`. *Co*-DETR outperforms other counterparts by a large margin.

a series of variants [31, 37, 44] such as ATSS [41], RetinaNet [21], FCOS [32], and PAA [17] lead to the significant breakthrough of object detection task. One-to-many label assignment is the core scheme of them, where each ground-truth box is assigned to multiple coordinates in the detector's output as the supervised target cooperated with proposals [11, 27], anchors [21] or window centers [32]. Despite their promising performance, these detectors heavily rely on many hand-designed components like a non-maximum suppression procedure or anchor generation [1]. To conduct a more flexible end-to-end detector, DEtection TRansformer (DETR) [1] is proposed to view the object detection as a set prediction problem and introduce the one-to-one set matching scheme based on a transformer encoder-decoder

SAP-DETR: Bridging the Gap between Salient Points and Queries-Based Transformer Detector for Fast Model Convergency

Yang Liu[1,3*] Yao Zhang[1,3*] Yixin Wang[2*] Yang Zhang[4] Jiang Tian[4]
Zhongchao Shi[4] Jianping Fan[4] Zhiqiang He[1,3,5†]
[1]Institute of Computing Technology (ICT), Chinese Academy of Sciences [2]Stanford University
[3]University of Chinese Academy of Sciences [4]AI Lab, Lenovo Research [5]Lenovo Ltd.
{liuyang20c,zhangyao215}@mails.ucas.ac.cn yxinwang@stanford.edu
{zhangyang20,tianjiang1,shizc2,jfan1,hezq}@lenovo.com

Abstract

*Recently, the dominant DETR-based approaches apply central-concept spatial prior to accelerating Transformer detector convergency. These methods gradually refine the reference points to the center of target objects and imbue object queries with the updated central reference information for spatially conditional attention. However, centralizing reference points may severely deteriorate queries' saliency and confuse detectors due to the indiscriminative spatial prior. To bridge the gap between the reference points of salient queries and Transformer detectors, we propose **SA**lient **P**oint-based **DETR (SAP-DETR)** by treating object detection as a transformation from salient points to instance objects. Concretely, we explicitly initialize a query-specific reference point for each object query, gradually aggregate them into an instance object, and then predict the distance from each side of the bounding box to these points. By rapidly attending to query-specific reference regions and the conditional box edges, SAP-DETR can effectively bridge the gap between the salient point and the query-based Transformer detector with a significant convergency speed. Experimentally, SAP-DETR achieves 1.4× convergency speed with competitive performance and stably promotes the SoTA approaches by ∼1.0 AP. Based on ResNet-DC-101, SAP-DETR*

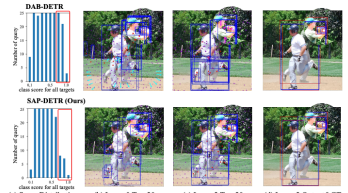


Figure 1. Comparison of SAP-DETR and DAB-DETR under 36 training epochs. (a) Statistics of the query count in different classification score intervals. (b) and (c) Distribution of reference points and the visualization of the query with top-20 classification score (blue proposal bounding boxes and red reference points) in different decoder layers. (d) Visualization of bounding boxes for positive queries (blue) and ground truth (red) during training process.

tors [6, 11, 14, 18, 20, 22] based on Convolutional Neural Networks (CNNs), have received widespread attention and made significant progress. Recently, Carion *et al.* [2] proposed a new end-to-end paradigm for object detection based on the Transformer [24], called DEtection TRansformer (DETR),

Further Study

Methods	Modifications				Publication	Highlights
	Bk	Pre	Attn	Qry		
DETR [7]	-	-	-	-	ECCV 2020	Transformer, Set-based prediction, bipartite matching
Deformable-DETR [9]			✔		ICLR 2021	Deformable-attention module
UP-DETR [10]		✔			CVPR 2021	Unsupervised pre-training, random query patch detection
Efficient-DETR [11]				✔	arXiv 2021	Refence point and top-k queries selection module
SMCA-DETR [12]			✔		ICCV 2021	Spatially-Modulated Co-attention module
TSP-DETR [28]			✔		ICCV 2021	TSP-FCOS and TSP-RCNN modules for cross attention
Conditional-DETR [14]				✔	ICCV 2021	Conditional spatial queries
WB-DETR [15]	✔				ICCV 2021	Encoder-decoder network without a backbone, LIE-T2T encoder module
PnP-DETR [16]			✔		ICCV 2021	PnP sampling module including pool sampler and poll sampler
Dynamc-DETR [17]			✔		ICCV 2021	Dynamic attention in the encoder-decoder network
YOLOS-DETR [18]		✔			NeurIPS 2021	Pretraining encoder network
Anchor-DETR [19]			✔	✔	AAAI 2022	Row and Column decoupled-attention, object queries as anchor points
Sparse-DETR [20]			✔		ICLR 2022	Cross-attention map predictor, deformable-attention module
D^2ETR [21]			✔		arXiv 2022	Fine fused features, cross-scale attention module
FP-DETR [22]	✔	✔			ICLR 2022	Multiscale tokenizer in place of CNN backbone, pretraining encoder network
CF-DETR [23]			✔		AAAI 2022	TEF module to capture spatial relationships, a coarse and a fine layer in the decoder network
DAB-DETR [29]				✔	ICLR 2022	Dynamic anchor boxes as object queries
DN-DETR [24]				✔	CVPR 2022	Positive noised object queries
AdaMixer [25]			✔		CVPR 2022	3D sampling module, Adaptive mixing module in the decoder
REGO [26]			✔		CVPR 2022	A multi-level recurrent mechanism and a glimpse-based decoder
DINO [27]				✔	arXiv 2022	Contrastive denoising module, positive and negative noised object queries


Thank you