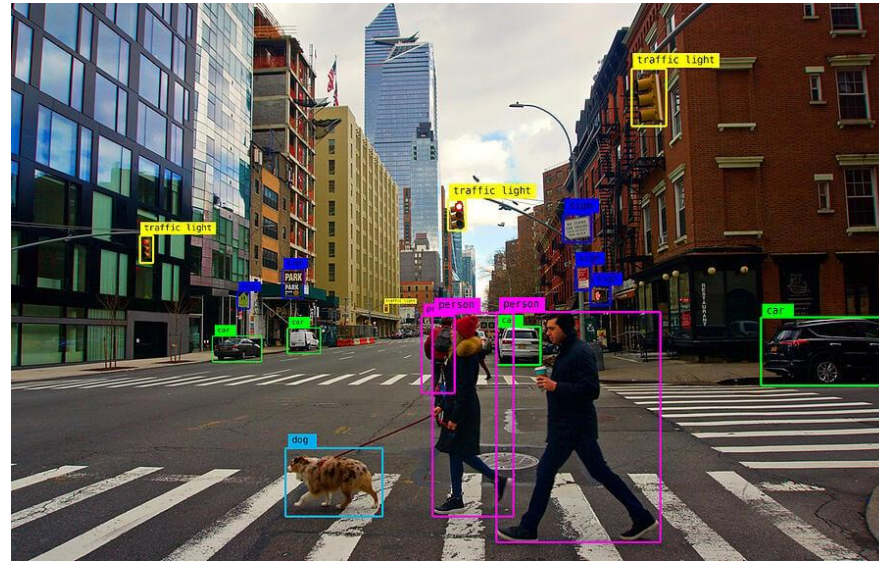


Object Detection

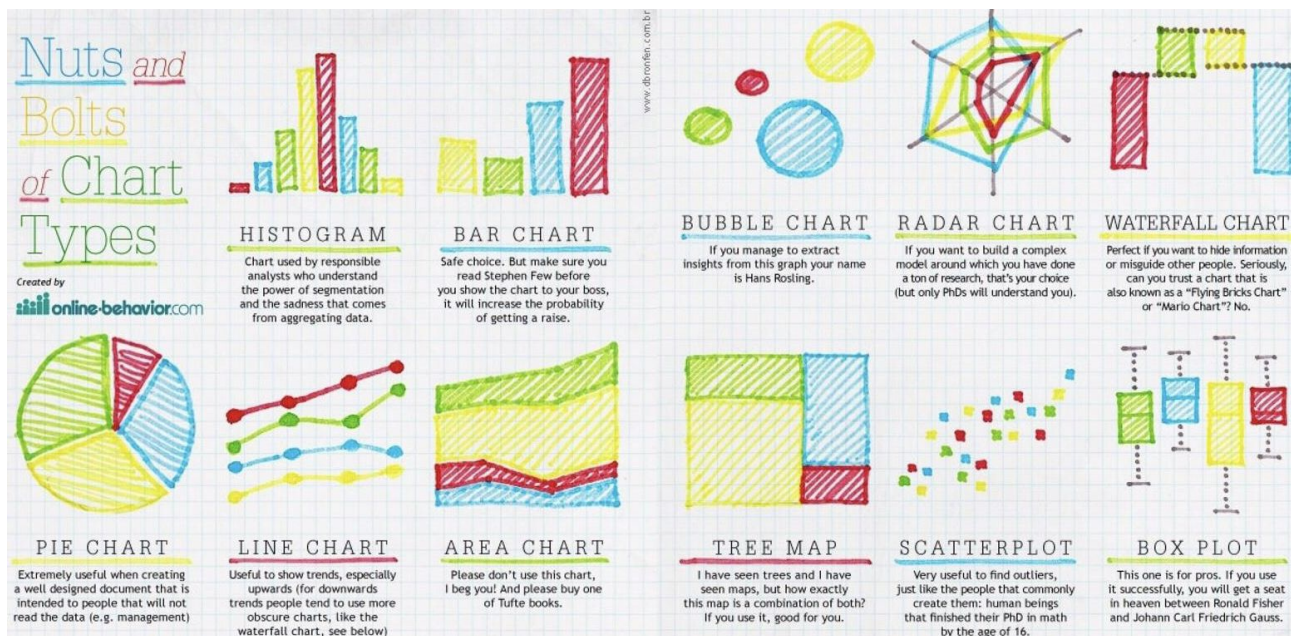


TA Hùng An

1. Một số thao tác EDA data cho bài toán Object Detection
2. Intersection Over Union
3. Non Maximum Suppression
4. Mean Average Precision
5. VOC2007 Dataset
6. YOLO-v1
7. Faster-RCNN

1 - Một số thao tác EDA data

Khi tiếp cận một bài toán AI nói chung hay bài toán Object Detection nói riêng thì việc tìm hiểu và nắm bắt được dữ liệu là một điều quan trọng để có được chiến lược tiếp cận bài toán và xây dựng model một cách hiệu quả



1 - Một số thao tác EDA data

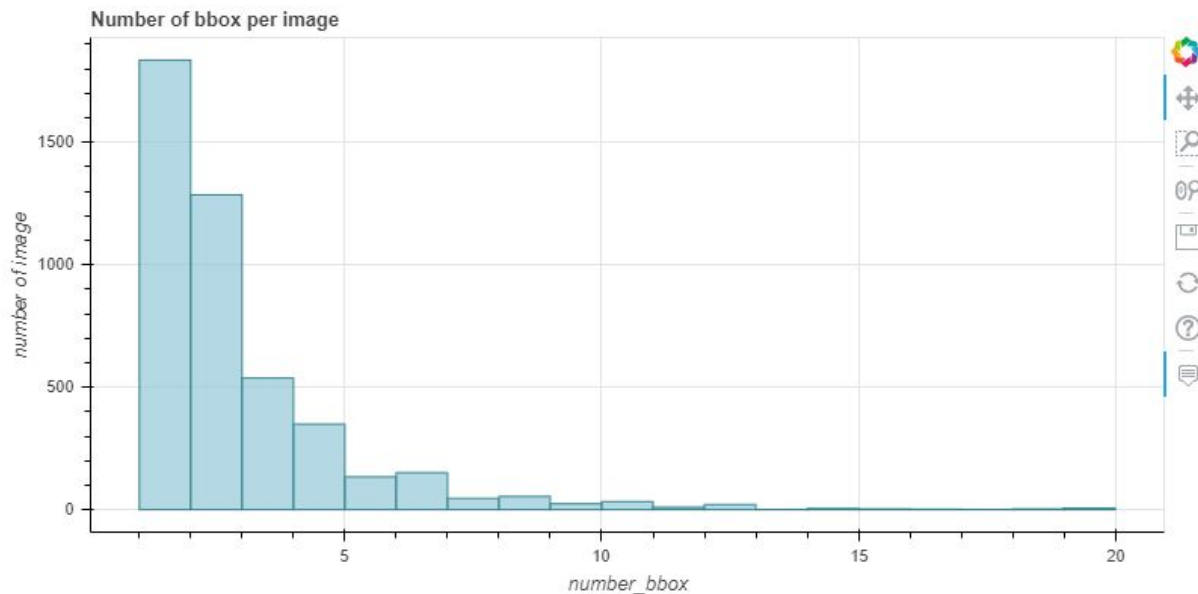
1 - Khảo sát thông tin của ảnh trong bộ dữ liệu

	file_name	width	height	number_bbox	image_id
0	3.png	1622	626	1	3
1	5.png	1622	626	2	5
2	6.png	1622	626	1	6
3	16.png	1622	626	1	16
4	17.png	1622	626	1	17

Một trong những thông tin cơ bản và cần ưu tiên nắm bắt đó là kích thước của ảnh.
Chúng ta cần biết được toàn bộ ảnh trong dataset có cùng chung kích thước hay không

1 - Một số thao tác EDA data

2 - Thống kê số lượng bounding box (đối tượng cần phát hiện) trong ảnh



Thông tin cần tìm hiểu trong dataset đó là số lượng bounding box trong một ảnh (mật độ xuất hiện). Đây là thông tin hữu ích để chúng ta biết được dữ liệu bài toán có phải là một dạng “Crowded Scenes” hay không?

1 - Một số thao tác EDA data

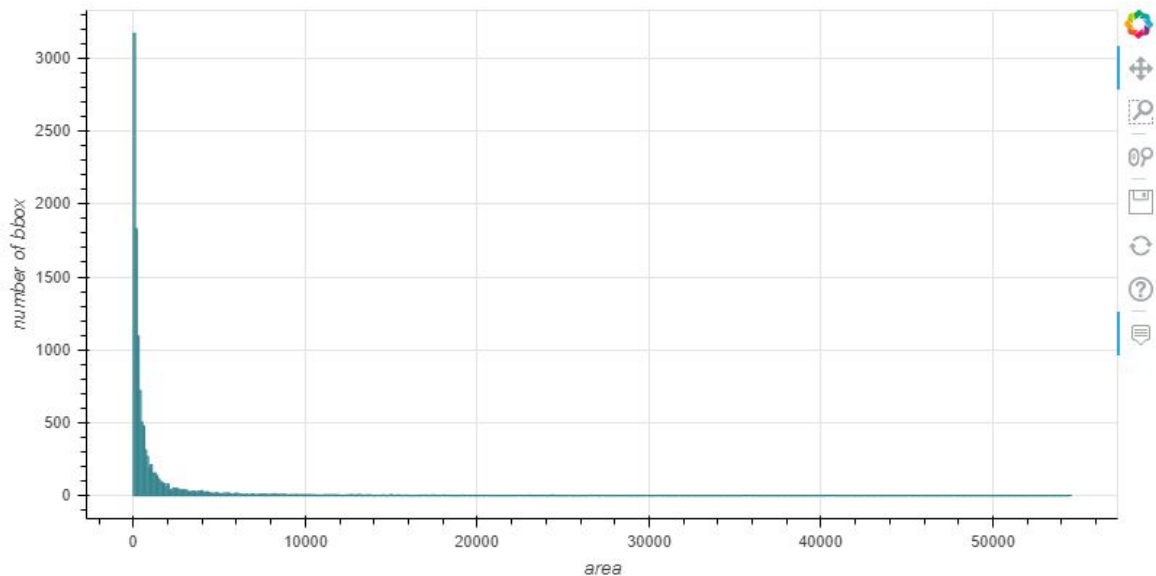
3 - Thống kê về kích thước của bounding box

	file_name	image_id	bbox	width_bbox	height_bbox	area	category_id	annotation_id
0	3.png	3	[880, 333, 19, 18]	19	18	342	2	0
1	5.png	5	[1069, 355, 83, 83]	83	83	6889	3	1
2	5.png	5	[768, 480, 9, 7]	9	7	63	2	2
3	6.png	6	[781, 337, 17, 15]	17	15	255	6	3
4	16.png	16	[733, 352, 7, 8]	7	8	56	2	4

Thông tin về kích thước của bounding box trong dataset giúp chúng ta thống kê được phân phối về kích thước của đối tượng trong ảnh. Với những số liệu này chúng ta có thể đưa ra được kết luận được việc xử lý “small object” hay không?

1 - Một số thao tác EDA data

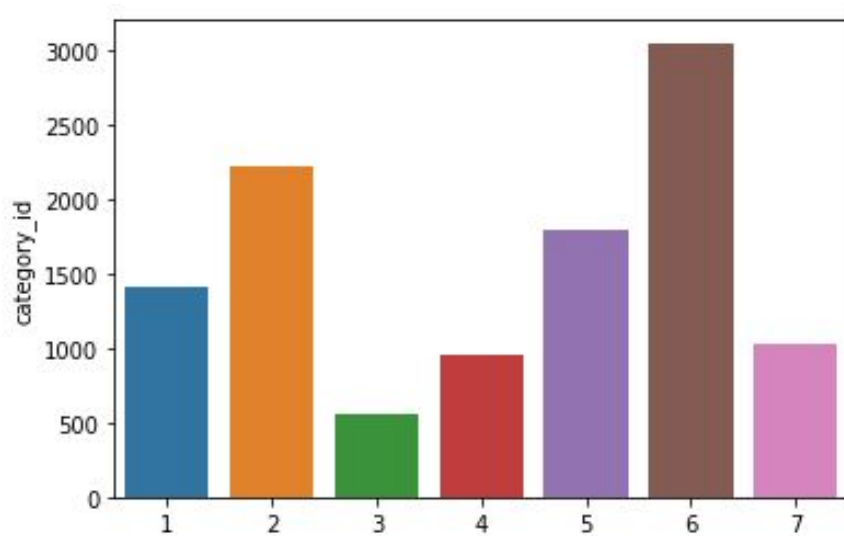
3 - Thống kê về kích thước của bounding box



Đối với việc thống kê thì chúng ta cần thể hiện số liệu dưới dạng chart để có được cái nhìn tổng quan hơn.

1 - Một số thao tác EDA data

4 - Thống kê số lượng Object trong từng class

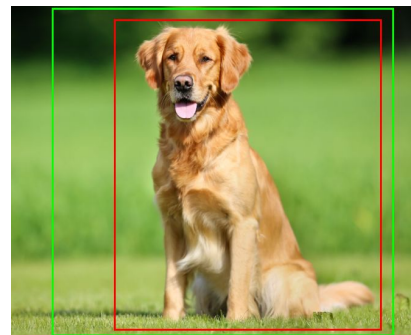
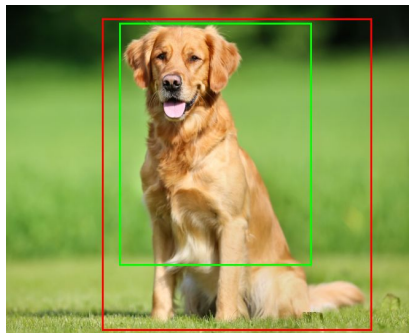
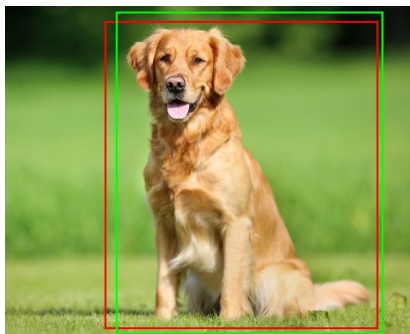


Việc xử lý dữ liệu mất cân bằng không chỉ là vấn đề của bài toán Classification mà cũng là vấn đề của bài toán Detection.

2 - Intersection Over Union

Với Bounding Box màu đỏ là Ground Truth và Bounding Box màu xanh là kết quả dự đoán của model.

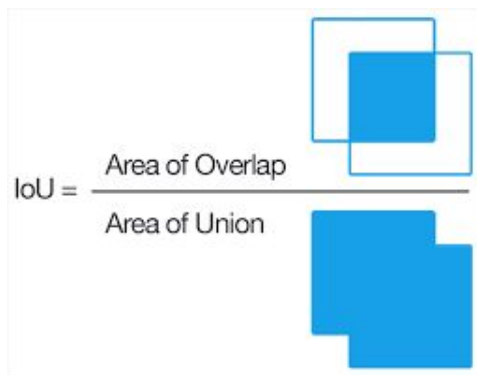
Chúng ta hãy nhận xét kết quả dự đoán của 3 model tương ứng với 3 ảnh bên dưới.

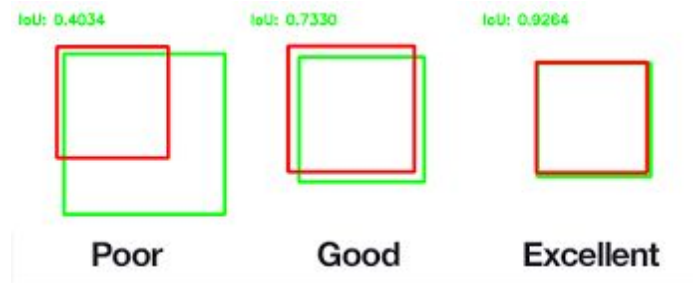


Cần một metric để có thể đánh giá được đâu là model tốt nhất trong 3 model trên.

2 - Intersection Over Union

Intersection Over Union (IOU) là chỉ số đánh giá độ chính xác của model Object Detection trên một bộ dữ liệu cụ thể. Được xác định bởi tỷ lệ giữa vùng “overlap” và vùng “combine” giữa kết quả dự đoán của model và Ground Truth.


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

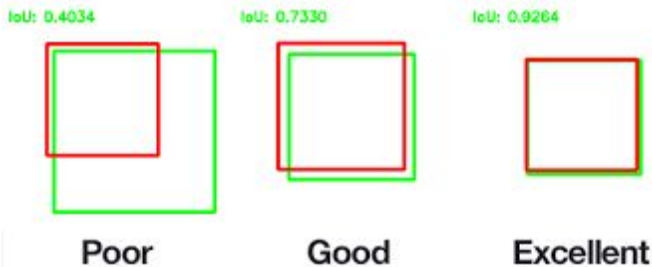
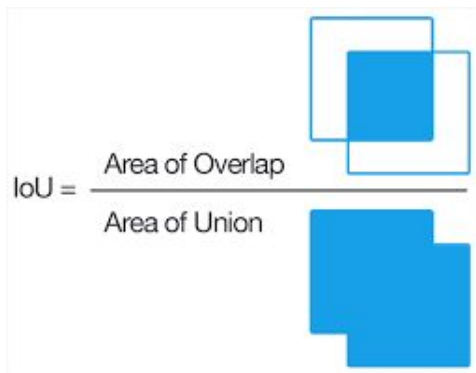


IOU là một metric phù hợp để đánh giá model khi:

- Dự đoán là sai khi model không dự đoán được đối tượng bên trong vùng Ground Truth.
- Kết quả dự đoán tràn (overflow) khỏi vùng Ground Truth.

2 - Intersection Over Union

Intersection Over Union (IOU) là chỉ số đánh giá độ chính xác của model Object Detection trên một bộ dữ liệu cụ thể. Được xác định bởi tỷ lệ giữa vùng “overlap” và vùng “combine” giữa kết quả dự đoán của model và Ground Truth



Tính toán tọa độ của hình chữ nhật giao nhau

```
x1 = torch.max(box1_x1, box2_x1)
```

```
y1 = torch.max(box1_y1, box2_y1)
```

```
x2 = torch.min(box1_x2, box2_x2)
```

```
y2 = torch.min(box1_y2, box2_y2)
```

.clamp(0) dùng để xử lý trường hợp chúng không giao nhau

```
intersection = (x2 - x1).clamp(0) * (y2 - y1).clamp(0)
```

Tính toán diện tích của hình chữ nhật giao nhau

```
box1_area = abs((box1_x2 - box1_x1) * (box1_y2 - box1_y1))
```

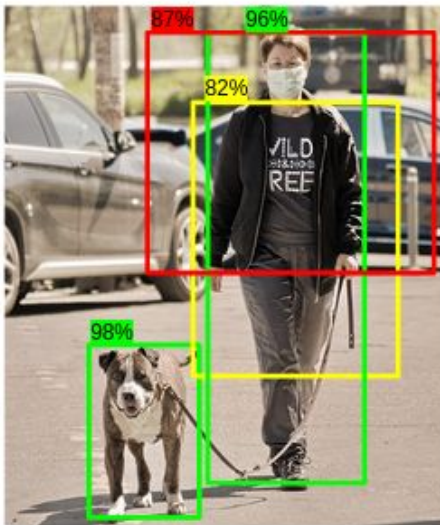
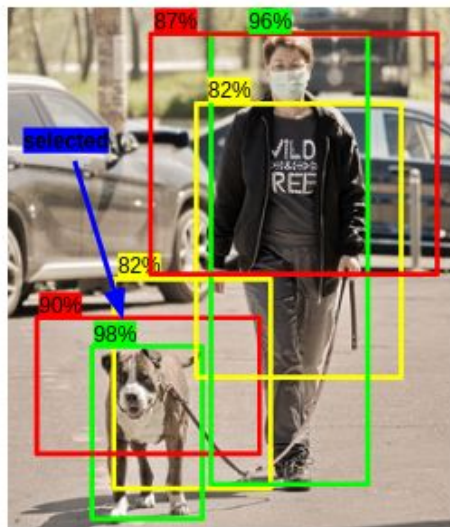
```
box2_area = abs((box2_x2 - box2_x1) * (box2_y2 - box2_y1))
```

Trả về tỷ lệ độ chồng chéo giữa các box, thêm epsilon để tránh chia cho 0

```
return intersection / (box1_area + box2_area - intersection + 1e-6)
```

3 - Non Maximum Suppression

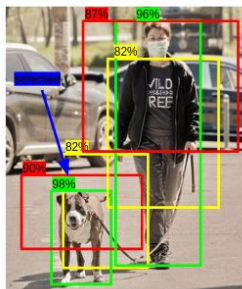
Non Maximum Suppression (NMS) là một bước hậu xử lý mà hầu hết các thuật toán Object Detection sau này đều sử dụng. Mục tiêu của NMS là lựa chọn một bounding box thích hợp nhất cho đối tượng.



3 - Non Maximum Suppression

NMS tính theo 2 tiêu chí:

- Objectiveness score được trả về bởi model
- Overlap hoặc IOU giữa các bounding box.



Đầu tiên NMS lựa chọn bounding box có scores cao nhất
(Lựa chọn bounding box màu xanh cho 2 đối tượng
cls-chó và cls-người)

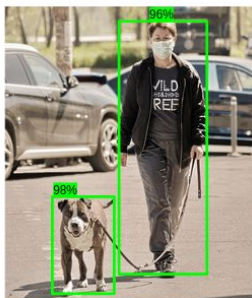
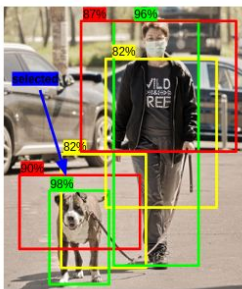


Sau đó loại bỏ những bounding box overlap với nó.
(Loại bỏ các bounding màu vàng và màu đỏ)

3 - Non Maximum Suppression

NMS tính theo 2 tiêu chí:

- Objectiveness score được trả về bởi model
- Overlap hoặc IOU giữa các bounding box.



```
# Lọc các box dự đoán dựa trên ngưỡng xác suất  
bboxes = [box for box in bboxes if box[1] > threshold]  
  
# Sắp xếp các box theo xác suất giảm dần  
bboxes = sorted(bboxes, key=lambda x: x[1], reverse=True)
```


3 - Non Maximum Suppression

Algorithm 1 Non-Max Suppression

```
1: procedure NMS( $B, c$ )
2:    $B_{nms} \leftarrow \emptyset$ 
3:   for  $b_i \in B$  do
4:      $discard \leftarrow \text{False}$ 
5:     for  $b_j \in B$  do
6:       if  $\text{same}(b_i, b_j) > \lambda_{nms}$  then
7:         if  $\text{score}(c, b_j) > \text{score}(c, b_i)$  then
8:            $discard \leftarrow \text{True}$ 
9:       if not  $discard$  then
10:         $B_{nms} \leftarrow B_{nms} \cup b_i$ 
11:   return  $B_{nms}$ 
```

Các bước xử lý của NMS:

1. Lựa chọn bounding box với score cao nhất
2. Sau đó so sánh IOU giữa bounding box đã chọn và các bounding còn lại
3. Loại bỏ các bounding box có IOU lớn hơn 50%
4. Tiếp tục với bounding box có score lớn hơn
5. Lặp lại các bước từ 2-4

3 - Non Maximum Suppression

Algorithm 1 Non-Max Suppression

```
1: procedure NMS( $B, c$ )
2:    $B_{nms} \leftarrow \emptyset$ 
3:   for  $b_i \in B$  do
4:      $discard \leftarrow \text{False}$ 
5:     for  $b_j \in B$  do
6:       if  $\text{same}(b_i, b_j) > \lambda_{nms}$  then
7:         if  $\text{score}(c, b_j) > \text{score}(c, b_i)$  then
8:            $discard \leftarrow \text{True}$ 
9:       if not  $discard$  then
10:         $B_{nms} \leftarrow B_{nms} \cup b_i$ 
11:   return  $B_{nms}$ 
```

```
# Thực hiện vòng lặp cho đến khi danh sách box rỗng
while bboxes:
    # Lấy box có xác suất cao nhất
    chosen_box = bboxes.pop(0)

    # Loại bỏ các box có IoU lớn hơn ngưỡng được chỉ định với box đã chọn
    bboxes = [
        box
        for box in bboxes
        if box[0] != chosen_box[0]
        or intersection_over_union(
            torch.tensor(chosen_box[2:]),
            torch.tensor(box[2:]),
            box_format=box_format,
        )
        < iou_threshold
    ]

# Thêm box đã chọn vào danh sách sau khi thực hiện NMS
bboxes_after_nms.append(chosen_box)
```


4 - Mean Average Precision

Average Precision - AP là giá trị trung bình trên nhiều IOU. $AP@[0.5:0.95]$ tương ứng cho AP trung bình cho IOU từ 0.5 đến 0.95 với step là 0.05. Đối với tập dữ liệu COCO AP là mức trung bình trên 9 mức IOU ($AP@[0.5:0.05:0.95]$) bắt đầu từ 0.05 đến 0.95 với step là 0.05

Average Precision (AP):

AP	% AP at $IoU=.50:.05:.95$ (primary challenge metric)
$AP^{IoU=.50}$	% AP at $IoU=.50$ (PASCAL VOC metric)
$AP^{IoU=.75}$	% AP at $IoU=.75$ (strict metric)

AP Across Scales:

AP^{small}	% AP for small objects: $area < 32^2$
AP^{medium}	% AP for medium objects: $32^2 < area < 96^2$
AP^{large}	% AP for large objects: $area > 96^2$

Average Recall (AR):

$AR^{max=1}$	% AR given 1 detection per image
$AR^{max=10}$	% AR given 10 detections per image
$AR^{max=100}$	% AR given 100 detections per image

AR Across Scales:

AR^{small}	% AR for small objects: $area < 32^2$
AR^{medium}	% AR for medium objects: $32^2 < area < 96^2$
AR^{large}	% AR for large objects: $area > 96^2$

4 - Mean Average Precision

Mean Average Precision - mAP là mức trung bình của AP. Được tính theo 2 bước:

1. Tính Average Precision cho mỗi class
2. Tính mAP

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i$$

```
1 81.852% = aeroplane AP
2 12.615% = apple AP
3 31.034% = backpack AP
4 27.652% = banana AP
5 60.591% = baseball-bat AP
6 52.699% = baseball-glove AP
7 91.895% = bear AP
8 73.863% = bed AP
9 43.250% = bench AP
10 52.618% = bicycle AP
11 47.743% = bird AP
12 42.745% = boat AP
13 23.295% = book AP
14 52.714% = bottle AP
15 55.977% = bowl AP
16 31.840% = broccoli AP
17 81.399% = bus AP
18 49.839% = cake AP
19 52.939% = car AP
20 2.849% = carrot AP
21 87.444% = cat AP
22 46.828% = cell-phone AP
23 52.618% = chair AP
24 74.931% = clock AP
25 57.715% = cow AP
26 58.847% = cup AP
27 47.931% = diningtable AP
28 79.716% = dog AP
29 24.626% = donut AP
30 80.199% = elephant AP
31 79.170% = fire-hydrant AP
32 46.861% = fork AP
```

```
33 82.098% = frisbee AP
34 80.181% = giraffe AP
35 4.545% = hair-drier AP
36 27.715% = handbag AP
37 79.503% = horse AP
38 18.734% = hot-dog AP
39 73.431% = keyboard AP
40 39.522% = kite AP
41 27.280% = knife AP
42 75.999% = laptop AP
43 69.728% = microwave AP
44 67.991% = motorbike AP
45 79.698% = mouse AP
46 4.920% = orange AP
47 58.388% = oven AP
48 81.872% = parking-meter AP
49 71.647% = person AP
50 33.640% = pizza AP
51 53.462% = pottedplant AP
52 72.852% = refrigerator AP
53 37.403% = remote AP
54 23.826% = sandwich AP
55 43.898% = scissors AP
56 62.098% = sheep AP
57 67.579% = sink AP
58 73.147% = skateboard AP
59 46.783% = skis AP
60 56.955% = snowboard AP
61 64.119% = sofa AP
62 27.465% = spoon AP
63 47.323% = sports-ball AP
64 68.481% = stop-sign AP
```

```
65 60.784% = suitcase AP
66 64.247% = surfboard AP
67 60.950% = teddy-bear AP
68 78.744% = tennis-racket AP
69 55.898% = tie AP
70 31.905% = toaster AP
71 83.340% = toilet AP
72 39.502% = toothbrush AP
73 44.792% = traffic-light AP
74 88.871% = train AP
75 48.374% = truck AP
76 73.030% = tvmonitor AP
77 67.164% = umbrella AP
78 59.283% = vase AP
79 54.945% = wine-glass AP
80 84.508% = zebra AP
81 mAP = 55.311%
```

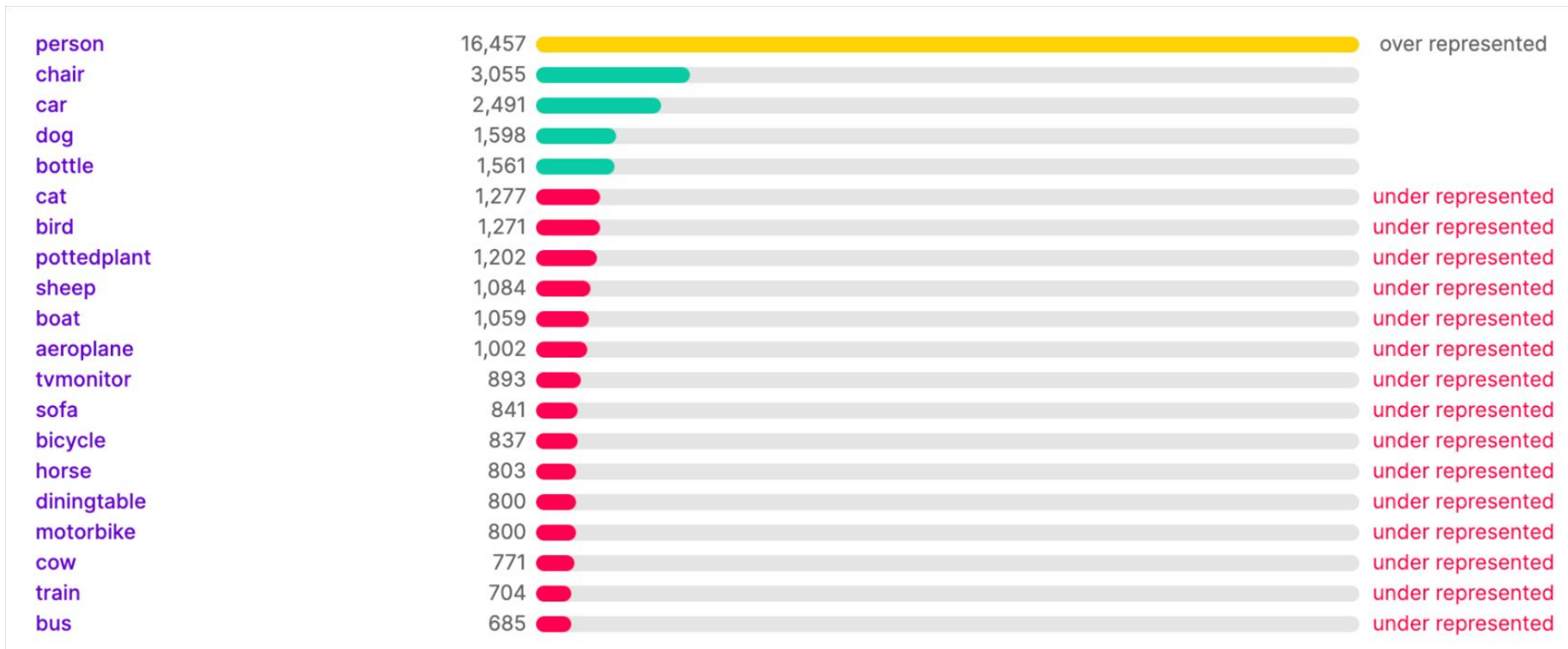
Yolo-V3 mAP

5 - VOC2007 Dataset



- ***Person:*** person
- ***Animal:*** bird, cat, cow, dog, horse, sheep
- ***Vehicle:*** aeroplane, bicycle, boat, bus, car, motorbike, train
- ***Indoor:*** bottle, chair, dining table, potted plant, sofa, tv/monitor

5 - VOC2007 Dataset



5 - VOC2007 Dataset

▼ images

- 000001.jpg
- 000002.jpg
- 000003.jpg
- 000004.jpg
- 000005.jpg
- ...

▼ labels

- 000001.txt
- 000002.txt
- 000003.txt
- 000004.txt
- 000005.txt
- ...

000001.txt (107 B)

class_id

```
11 0.34419263456090654 0.611 0.4164305949008499 0.262
14 0.509915014164306 0.51 0.9745042492917847 0.972
```

bounding_boxes

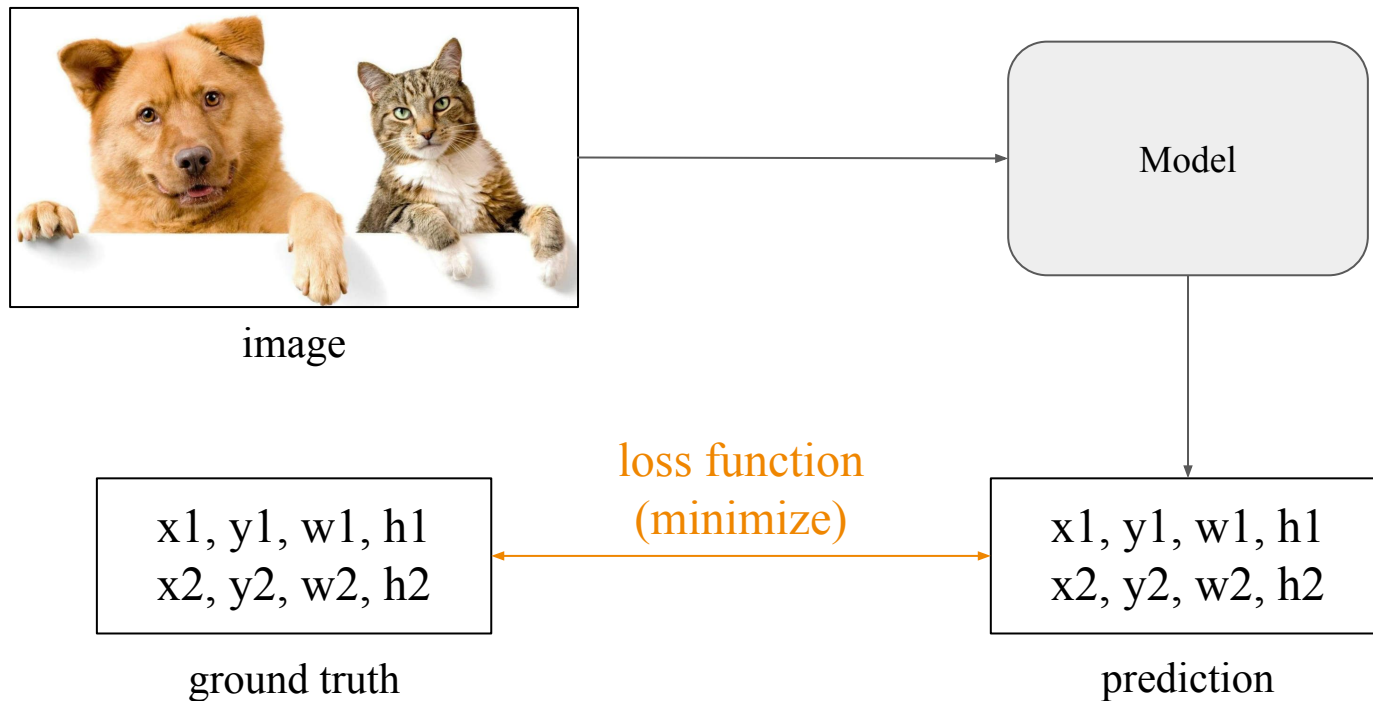
000002.txt (55 B)

class_id

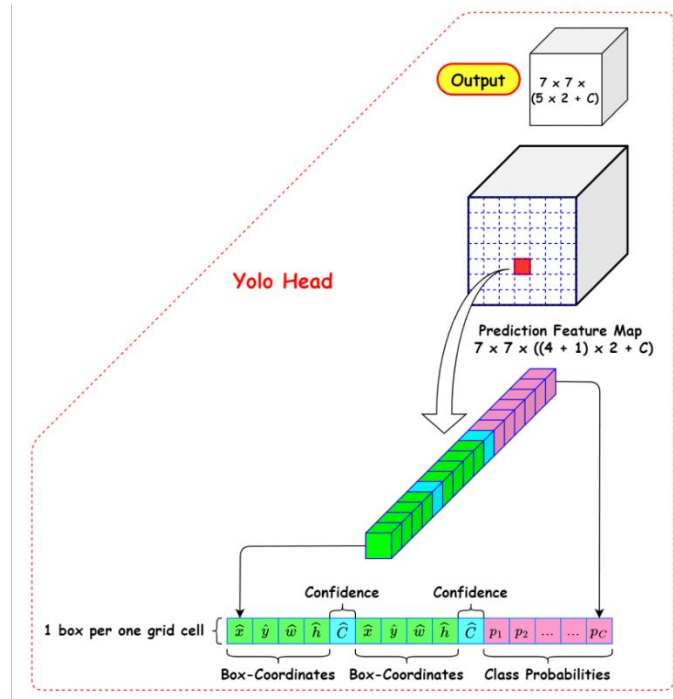
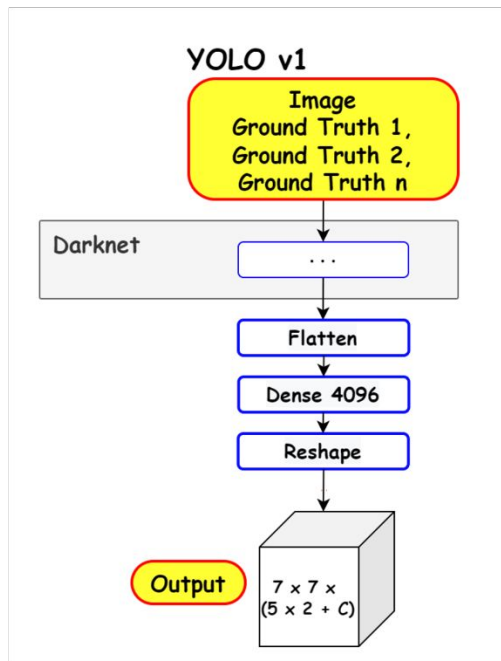
```
18 0.5164179104477612 0.501 0.20298507462686569 0.202
```

bounding_boxes

5 - VOC2007 Dataset



6 - Yolo v1

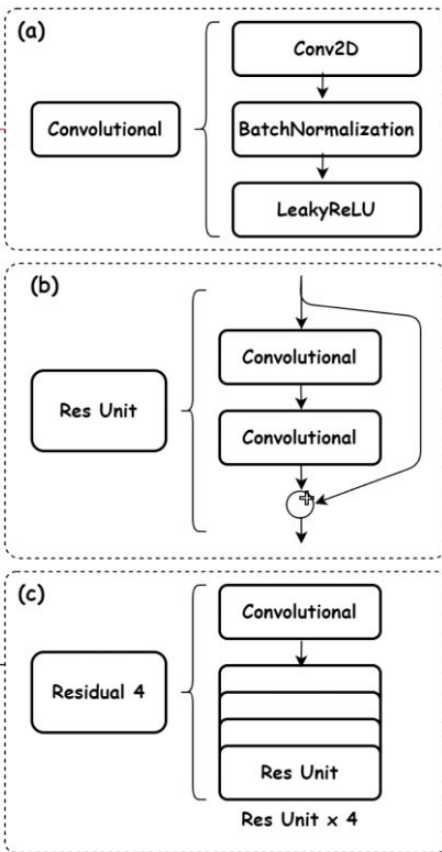


YOLO sử dụng một mạng neural duy nhất cho toàn bộ quá trình: dự đoán bounding box và xác suất của lớp trực tiếp từ hình ảnh đầy đủ trong một lần đánh giá.

6 - Yolo v1

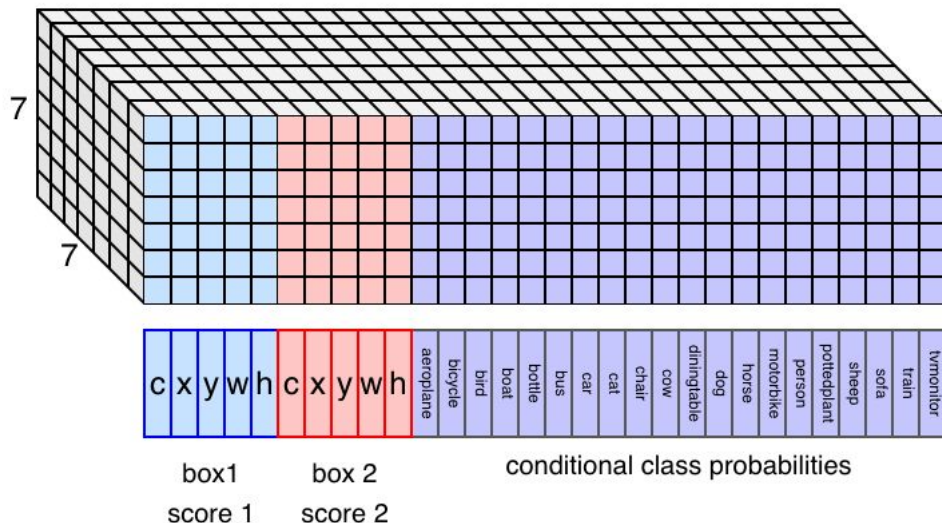
Darknet-53 (in Paper)

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



6 - Yolo v1

Output

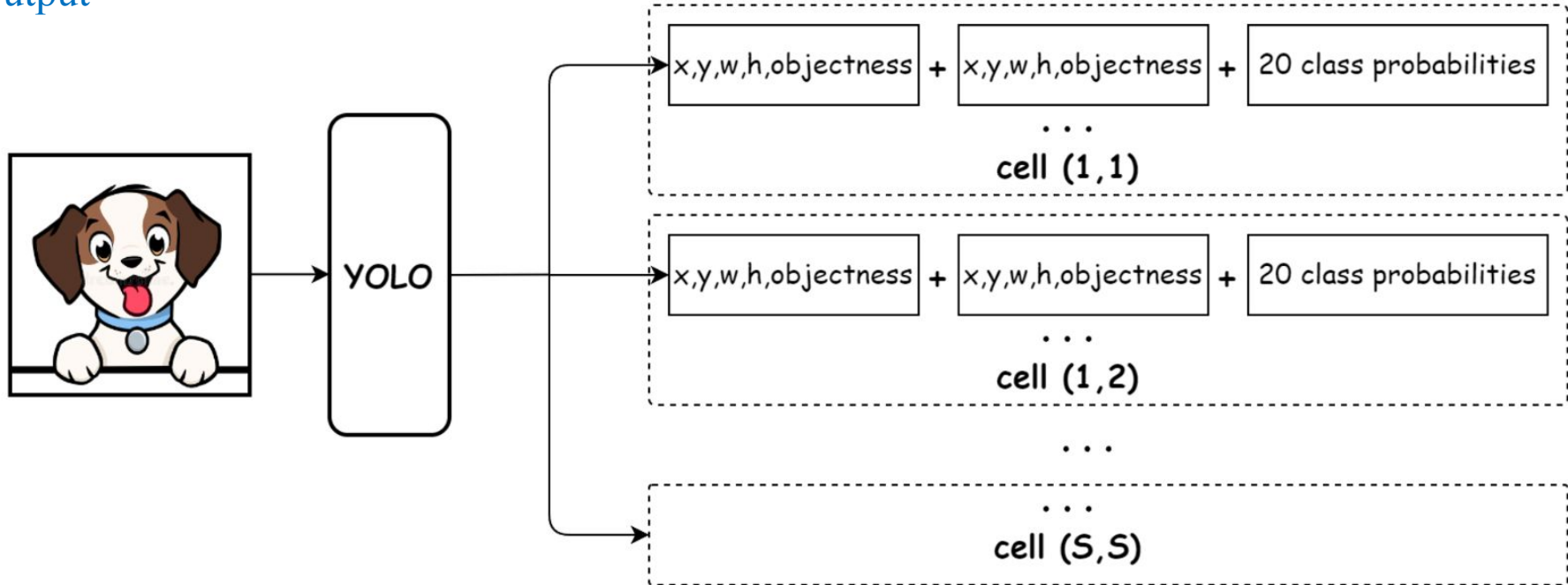


YOLOv1 output 7x7x30:

- 2 boxes for each grid cell
 - each box contain: x, y, w, h, and confidence score
- 20 classes of VOC dataset

6 - Yolo v1

Output



6 - Yolo v1

$$\mathcal{L}_{\text{loc}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + \\ + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

```
# ===== #  
#   FOR BOX COORDINATES   #  
# ===== #  
  
# Đặt các box không có đối tượng trong đó thành 0. Lấy ra một trong hai  
# dự đoán, là dự đoán có Iou cao nhất được tính toán trước đó.  
box_predictions = exists_box * (  
    (  
        bestbox * predictions[..., 26:30]  
        + (1 - bestbox) * predictions[..., 21:25]  
    )  
)  
  
box_targets = exists_box * target[..., 21:25]  
  
# Lấy sqrt của chiều rộng, chiều cao của box để đảm bảo  
box_predictions[..., 2:4] = torch.sign(box_predictions[..., 2:4]) *  
    torch.sqrt(torch.abs(box_predictions[..., 2:4] + 1e-6))  
)  
box_targets[..., 2:4] = torch.sqrt(box_targets[..., 2:4])  
  
box_loss = self.mse(  
    torch.flatten(box_predictions, end_dim=-2),  
    torch.flatten(box_targets, end_dim=-2),  
)
```

Yolo Loss

6 - Yolo v1

$$\sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} (C_{ij} - \hat{C}_{ij})^2$$

```
# ===== #  
#   FOR OBJECT LOSS   #  
# ===== #  
  
# pred_box là điểm tin cậy của bbox có IoU cao nhất  
pred_box = (  
    bestbox * predictions[..., 25:26] +  
    (1 - bestbox) * predictions[..., 20:21]  
)  
  
object_loss = self.mse(  
    torch.flatten(exists_box * pred_box),  
    torch.flatten(exists_box * target[..., 20:21]),  
)
```

Yolo Loss

6 - Yolo v1

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{noobj}} (C_{ij} - \hat{C}_{ij})^2$$

```
# ===== #  
#   FOR NO OBJECT LOSS   #  
# ===== #  
  
no_object_loss = self.mse(  
    torch.flatten((1 - exists_box) * predictions[..., 20:21], start_dim=1),  
    torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1),  
)  
  
no_object_loss += self.mse(  
    torch.flatten((1 - exists_box) * predictions[..., 25:26], start_dim=1),  
    torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1)  
)
```

Yolo Loss

6 - Yolo v1

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

```
# ===== #  
#   FOR CLASS LOSS   #  
# ===== #  
  
class_loss = self.mse(  
    torch.flatten(exists_box * predictions[..., :20], end_dim=-2),  
    torch.flatten(exists_box * target[..., :20], end_dim=-2),  
)  
  
loss = (  
    self.lambda_coord * box_loss # 2 hàng đầu  
    + object_loss # hàng thứ 3  
    + self.lambda_noobj * no_object_loss # hàng thứ 4  
    + class_loss # hàng thứ 5  
)
```

Yolo Loss

6 - Yolo v1

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

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

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

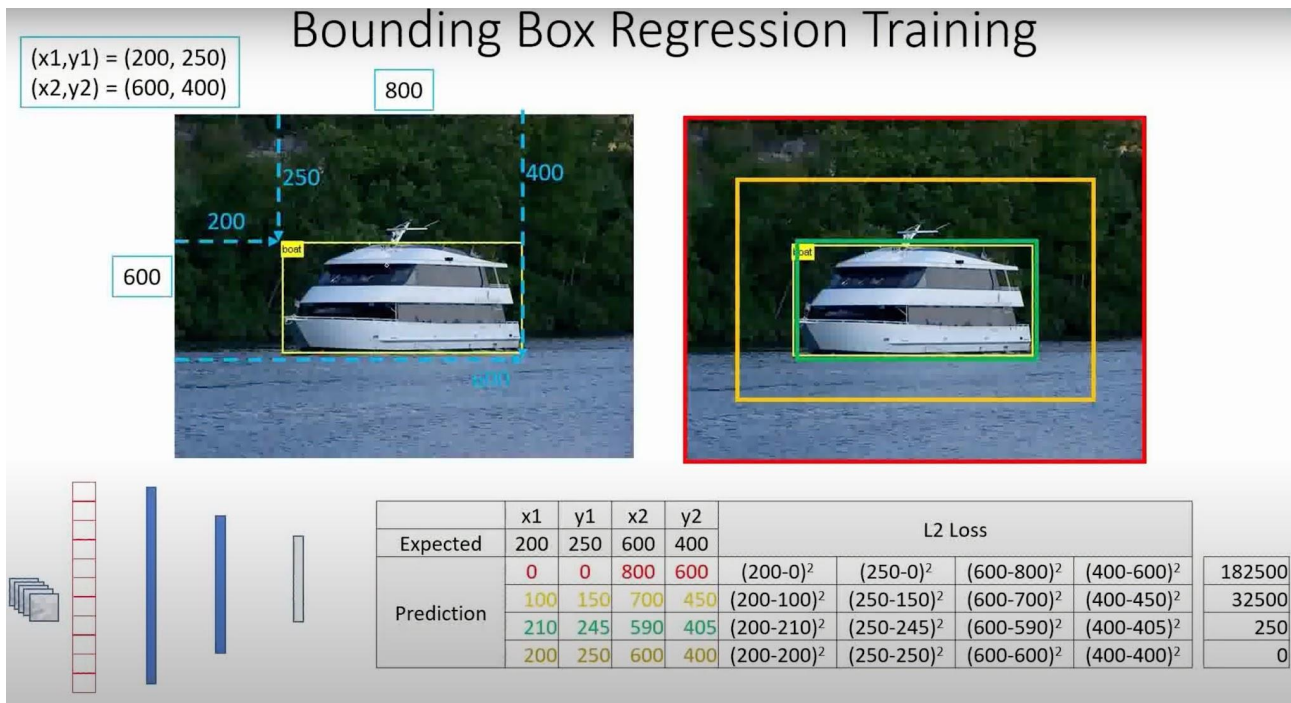
❸
$$\mathcal{L}_{\text{obj}} = \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} (C_{ij} - \hat{C}_{ij})^2$$

❹
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{noobj}} (C_{ij} - \hat{C}_{ij})^2$$

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

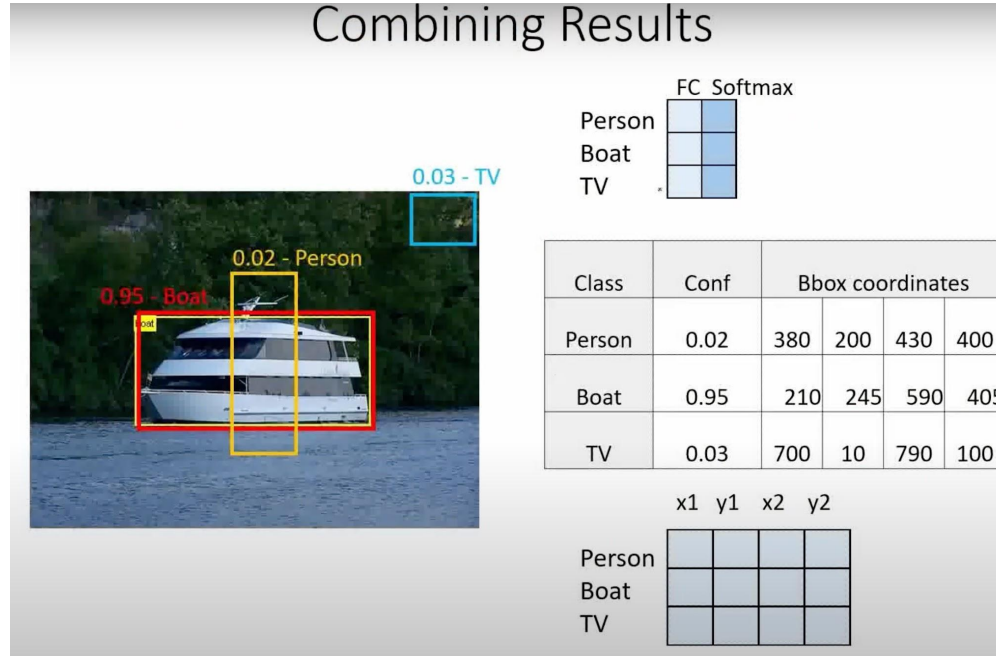
1. Phạt lỗi vị trí (x, y) của bounding box.
2. Phạt lỗi bounding box sai height và width.
3. Kéo giá trị confidence gần 1 khi có object trong cell đó.
4. Kéo giá trị confidence gần 0 khi không có object trong cell đó.
5. hàm loss classification đơn giản.

6 - Yolo v1



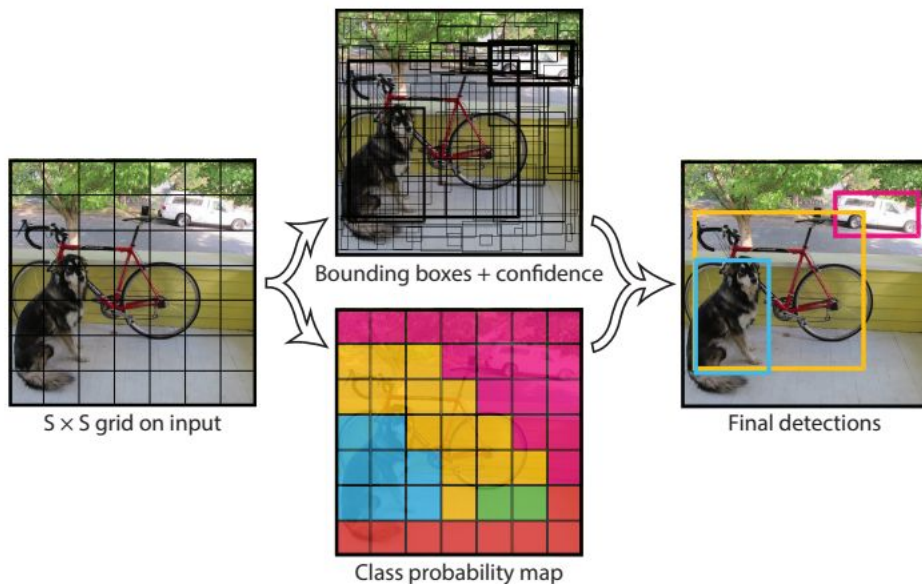
YOLO chuyển đổi việc phát hiện đối tượng thành một vấn đề hồi quy đơn lẻ, trực tiếp từ điểm ảnh hình ảnh đến tọa độ bounding box và xác suất của lớp.

6 - Yolo v1



Trong quá trình dự đoán, YOLO chỉ yêu cầu một lần đánh giá mạng trên mỗi hình ảnh.

6 - Yolo v1



Hạn chế về ràng buộc không gian: YOLO áp đặt ràng buộc không gian mạnh mẽ trên dự đoán bounding box, hạn chế số lượng đối tượng gần kề mà nó có thể dự đoán. Khó khăn với các đối tượng nhỏ, trong nhóm/cụm.

6 - Yolo v1

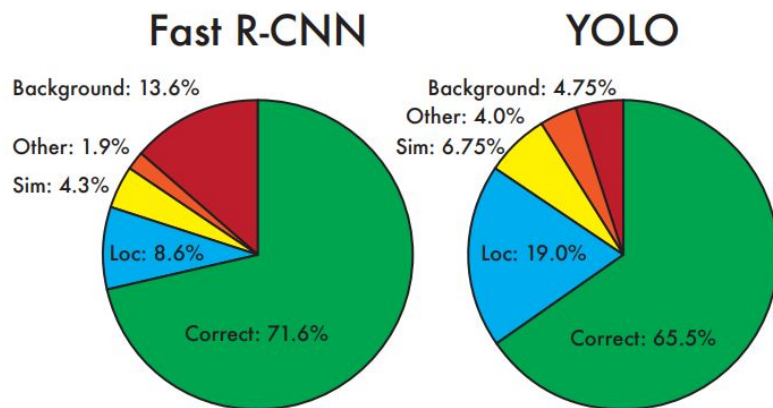
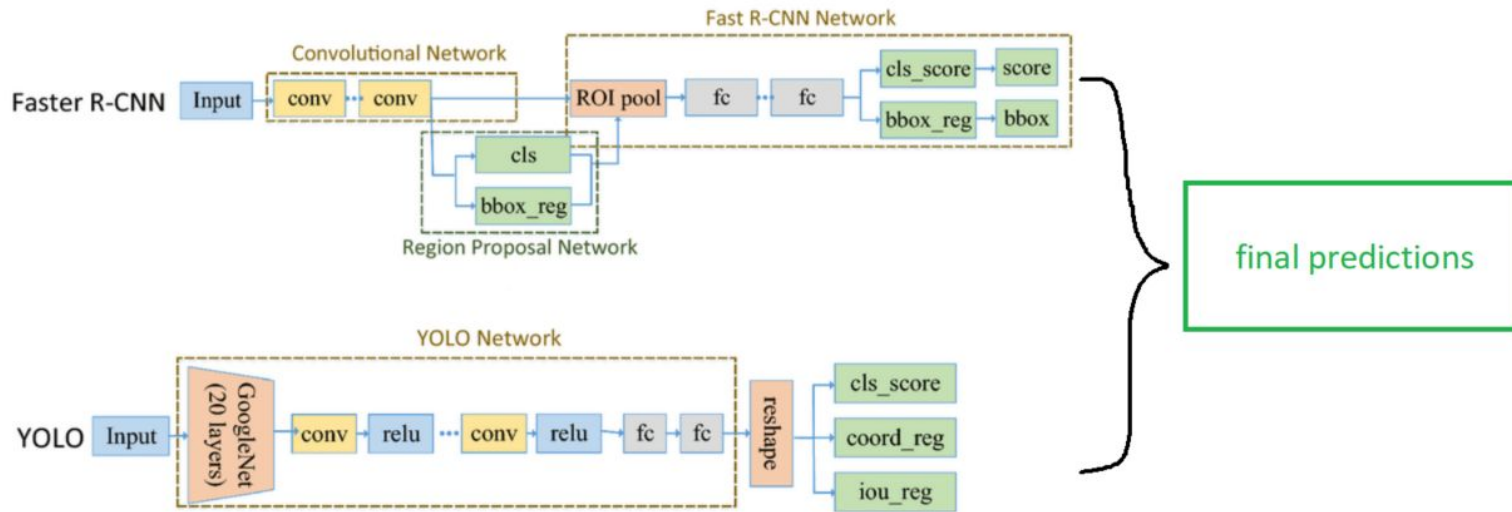


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories ($N = \#$ objects in that category).

- Lỗi vị trí xảy ra khi YOLO dự đoán bounding box không chính xác xung quanh đối tượng. Điều này có thể bao gồm việc bounding box, quá nhỏ, hoặc không đúng vị trí.
- YOLO sử dụng một phương pháp hồi quy trực tiếp từ điểm ảnh hình ảnh đến tọa độ bounding box và xác suất của lớp, điều này có thể làm tăng khả năng xuất hiện lỗi vị trí.
- Lỗi nền xảy ra khi một hệ thống phát hiện sai phần nền của hình ảnh là một đối tượng. YOLO có xu hướng gây ra ít kết quả false positives do lỗi nền, nhờ cách tiếp cận toàn cục khi xét toàn bộ hình ảnh.

6 - Yolo v1



YOLO còn được sử dụng để tăng cường phát hiện cho Fast R-CNN bằng cách loại bỏ phát hiện nền, có một sự tăng hiệu suất đáng kể. Cách tiếp cận kết hợp này cho thấy sức mạnh bổ sung của hai mô hình.

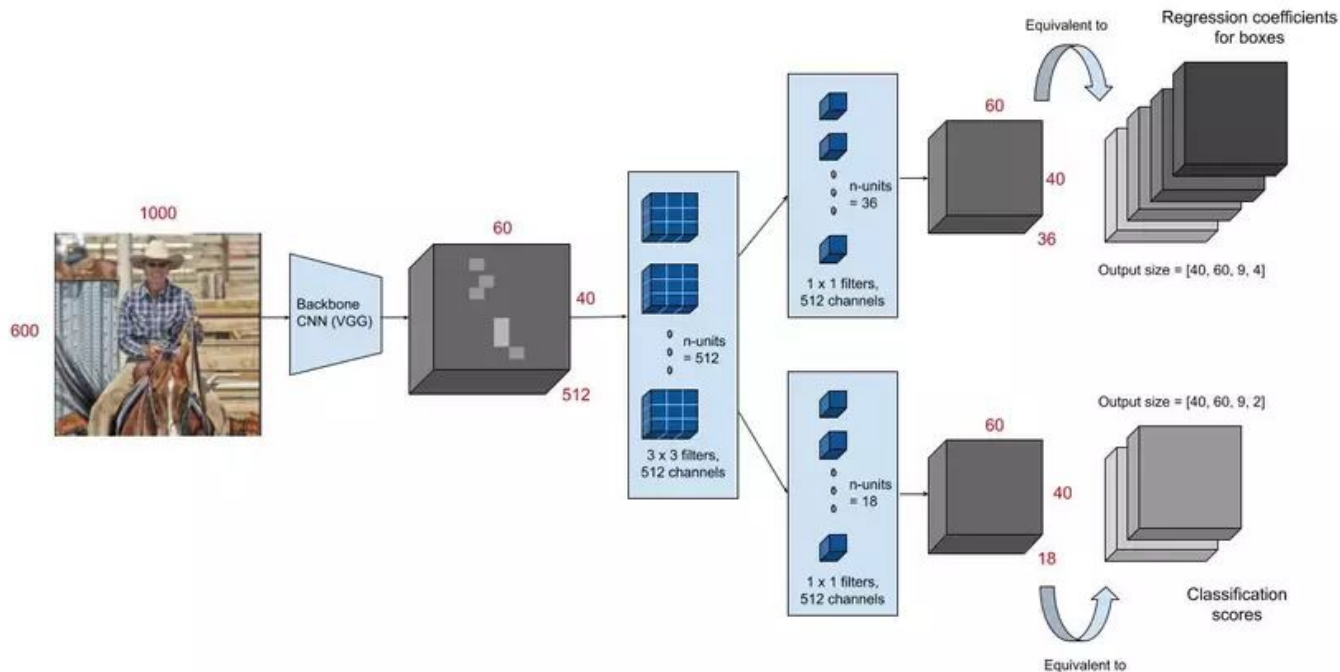
6 - Yolo v1

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

YOLO còn được sử dụng để tăng cường phát hiện cho Fast R-CNN bằng cách loại bỏ phát hiện nền, có một sự tăng hiệu suất đáng kể. Cách tiếp cận kết hợp này cho thấy sức mạnh bổ sung của hai mô hình.

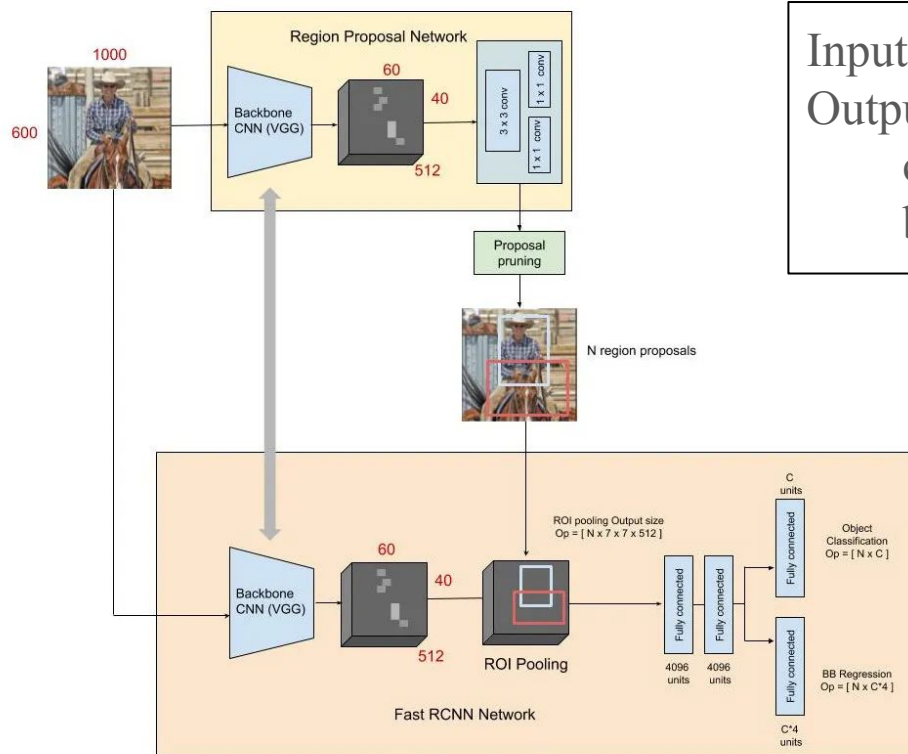
7 - Faster-RCNN

1. Region proposal network (RPN)



7 - Faster-RCNN

2. Faster R-CNN (RPN + Fast R-CNN)



Input: 600x1000x3

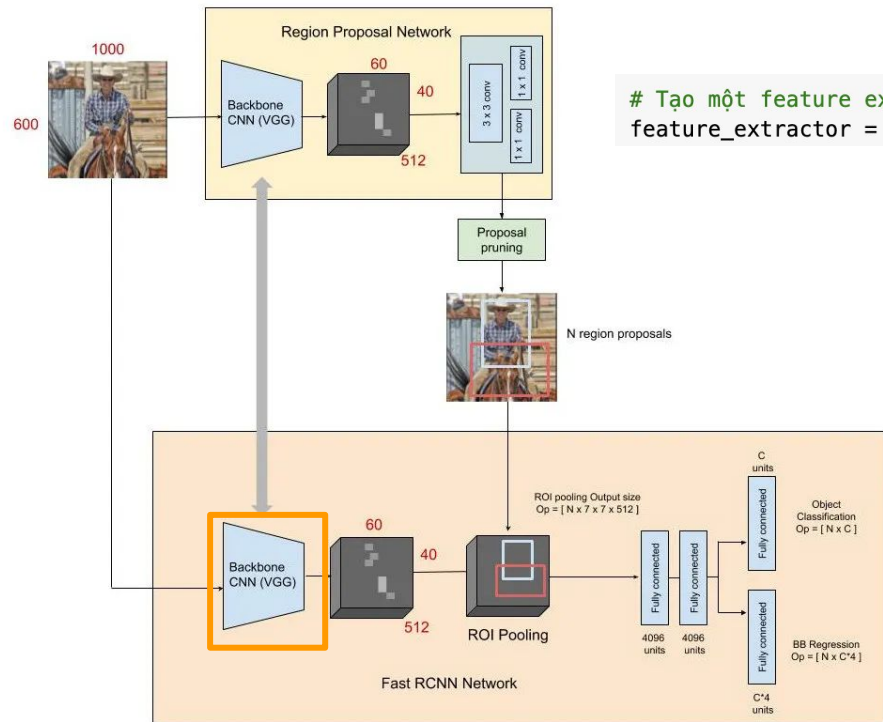
Output:

class: $N \times C$

bbox: $N \times C \times 4$

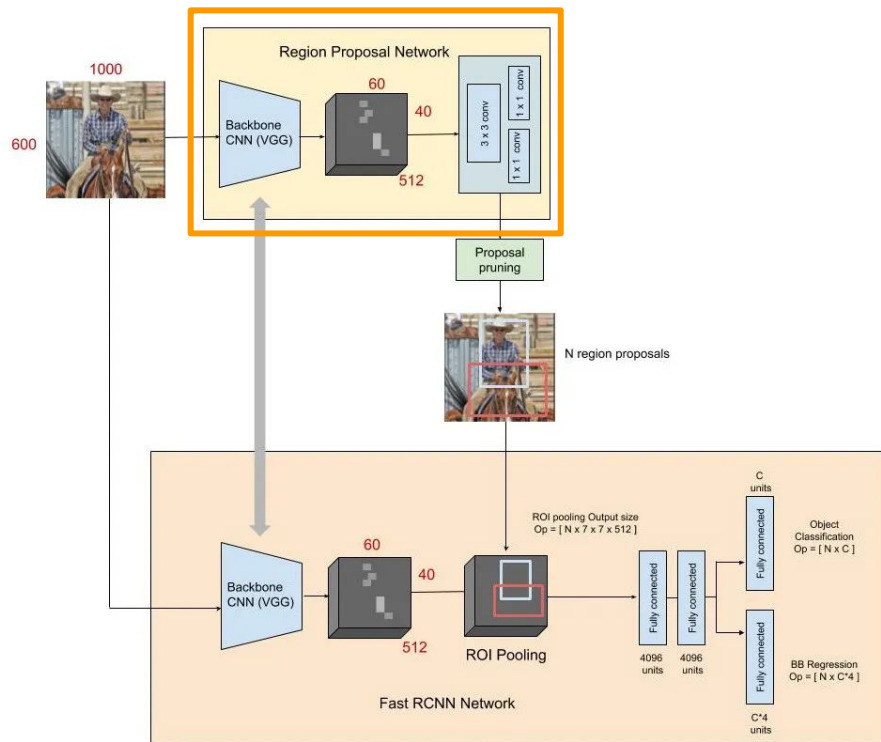
7 - Faster-RCNN

Feature Extractor



```
# Tạo một feature extractor từ mô hình VGG16 được pretrained, giữ lại các tầng cho đến conv5_3  
feature_extractor = torchvision.models.vgg16(pretrained=True).features[:30]
```


Region Proposal Network

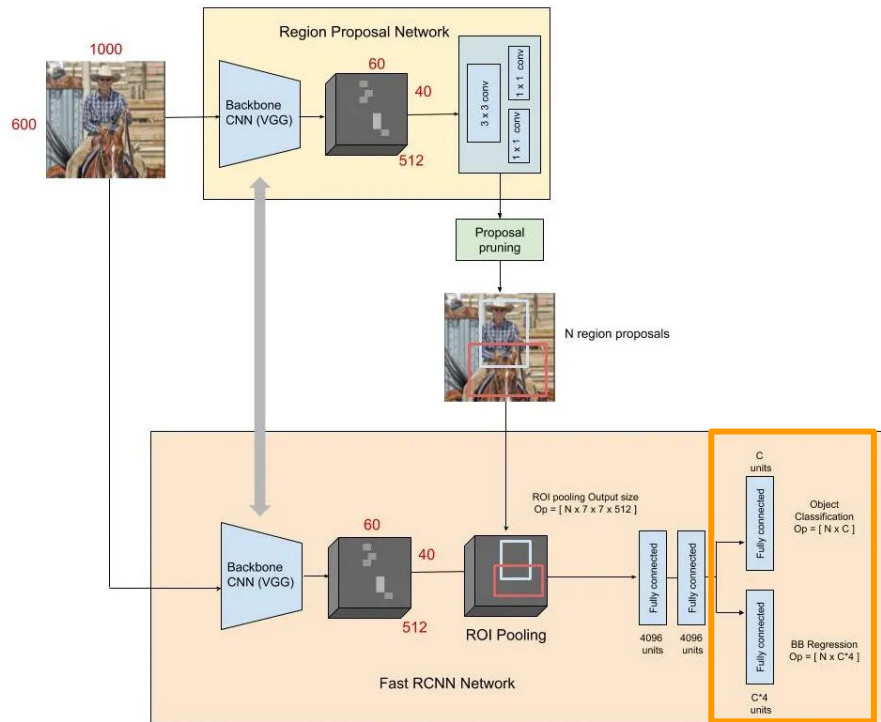


```
class RPN(nn.Module):
    def __init__(self, in_channels, num_anchors):
        super().__init__()
        # Định nghĩa một lớp convolutional 3x3 với 512 output channels
        self.conv = nn.Conv2d(in_channels, 512, 3, padding=1)
        # Định nghĩa một lớp convolutional 1x1 để dự đoán điểm số cho các anchor
        self.s1 = nn.Conv2d(512, num_anchors, 1)
        # Định nghĩa một lớp convolutional 1x1 để dự đoán offsets cho các anchor
        self.s2 = nn.Conv2d(512, num_anchors * 4, 1)
        # Định nghĩa hàm kích hoạt ReLU
        self.relu = nn.ReLU()

    def forward(self, x):
        # Áp dụng lớp convolutional và hàm kích hoạt ReLU
        x = self.relu(self.conv(x))
        # Dự đoán điểm số cho anchor sử dụng hàm kích hoạt sigmoid
        scores = torch.sigmoid(self.s1(x))
        # Dự đoán offsets cho anchor
        offsets = self.s2(x)
        return scores, offsets
```

7 - Faster-RCNN

Classifier



```
class Classifier(nn.Module):
    def __init__(self, in_channels, pool_size, num_classes):
        super().__init__()

        # Số chiều của tensor đầu vào khi được flat
        self.flat = in_channels * pool_size * pool_size

        # Linear layer 1 với đầu vào có số chiều là flat và đầu ra là 4096
        self.lin1 = nn.Linear(in_channels * pool_size * pool_size, 4096)

        # Linear layer 2 với đầu vào có số chiều là 4096 và đầu ra là 4096
        self.lin2 = nn.Linear(4096, 4096)

        # Hàm kích hoạt ReLU
        self.relu = nn.ReLU()

        # Linear layer cho dự đoán lớp, đầu ra có số chiều là num_classes+1
        self.score_layer = nn.Linear(4096, num_classes + 1)

        # Linear layer cho dự đoán hệ số regression, đầu ra có số chiều là 4*num_classes
        self.offset_layer = nn.Linear(4096, 4 * num_classes)

    def forward(self, x):
        # Flat tensor
        x = x.view(-1, self.flat)

        # Áp dụng Linear layer 1 và ReLU
        x = self.relu(self.lin1(x))

        # Áp dụng Linear layer 2 và ReLU
        x = self.relu(self.lin2(x))

        # Dự đoán điểm số cho từng lớp
        class_scores = self.score_layer(x)

        # Dự đoán hệ số regression
        offsets = self.offset_layer(x)

        # Trả về điểm số lớp và hệ số regression
        return class_scores, offsets
```

Loss function

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

Với i là index của anchor trong mini-batch và p_i là xác suất dự đoán của anchor i là một đối tượng. Giá trị nhãn ground-truth p_i^* là một nếu anchor là positive, và là không khi anchor là negative.

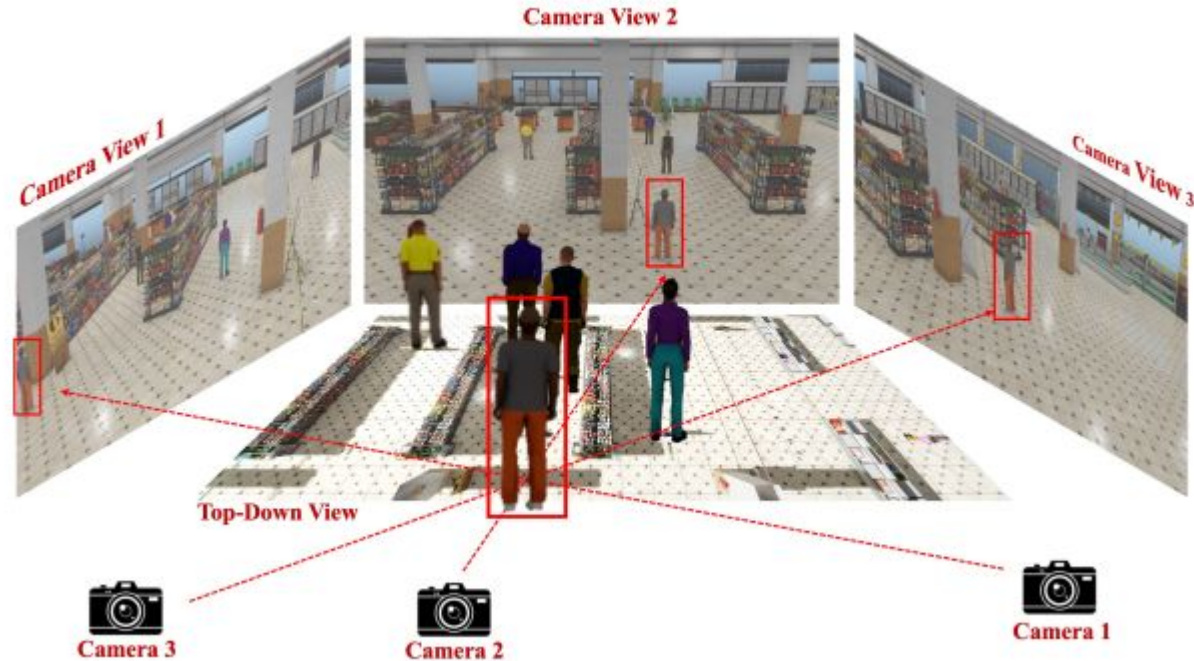
- t_i là một vector 4 chiều biểu diễn giá trị tọa độ của bounding box đã được dự đoán.
- t_i^* là vector 4 chiều biểu diễn giá trị tọa độ của ground-truth box tương ứng với positive anchor.
- L_{cls} là log loss của 2 class (object và non-object)
- L_{reg} dùng SmoothL1Loss



[HOME](#) [CHALLENGE](#) [WORKSHOP](#) [INFO](#) [PAST CHALLENGES](#)



Nội dung



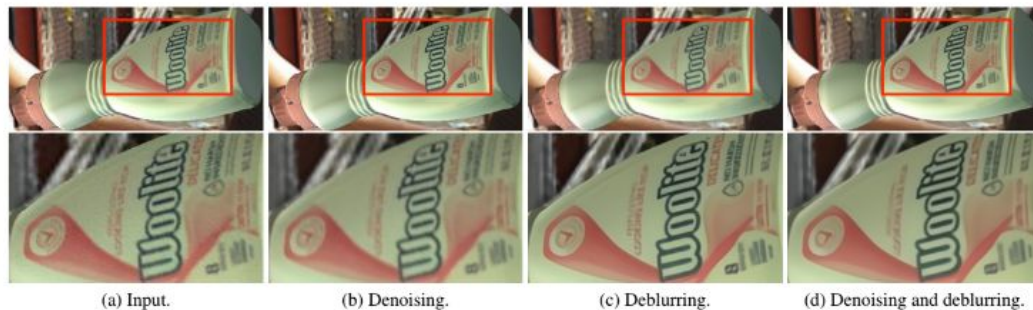


Figure 2. Illustration of denoising and deblurring on training images using NAFNet. Images are best viewed in color.

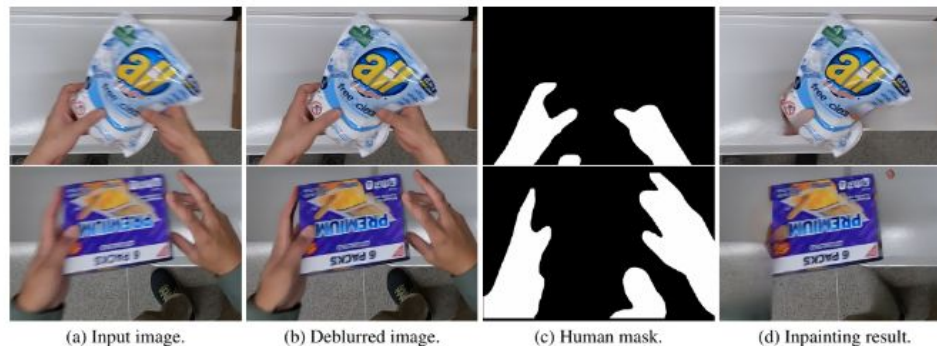


Figure 3. Illustration of motion deblurring and human inpainting on testing videos. Images are best viewed in color.