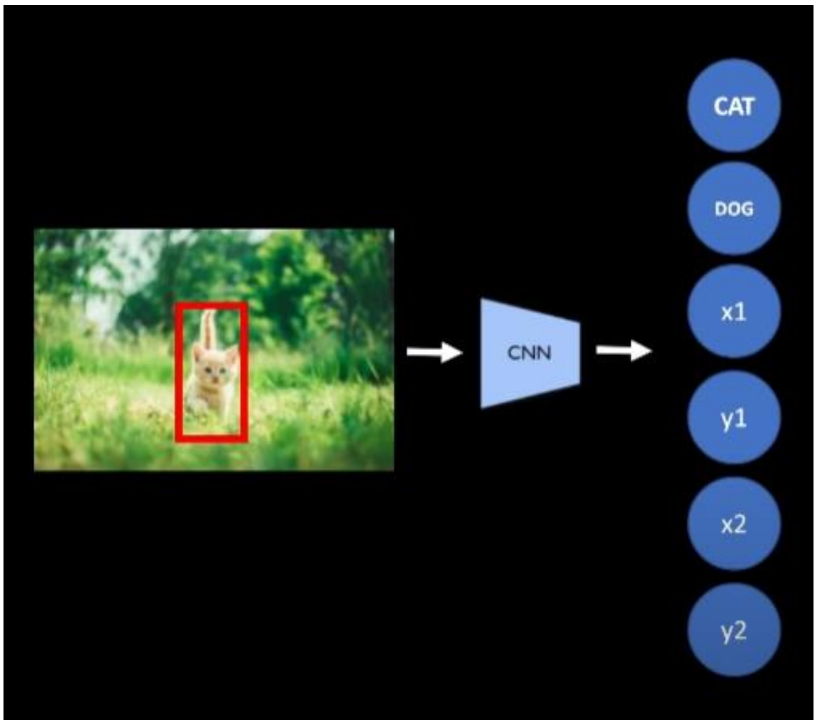
Introduction to Object Detection

Reference: https://www.youtube.com/playlist?list=PLhhyoLH6Ijfw0TpCTVTNk42NN08H6UvNq

Two-stage methods: RCNN series
One-stage methods: YOLO, SSD, RetinaNet

In the object detection task, we need to make model decide what the object is in the given image and a bounding box mark the object's location. Additionally, object detection task requires models to do object localization on multiple objects in a given image.

Output structure



Two BBOXES formats:
1. (x1, y1) for upper left corner point and (x2, y2) for bottom right corner point.
2. Two points for a corner point and the other two points for height and width.

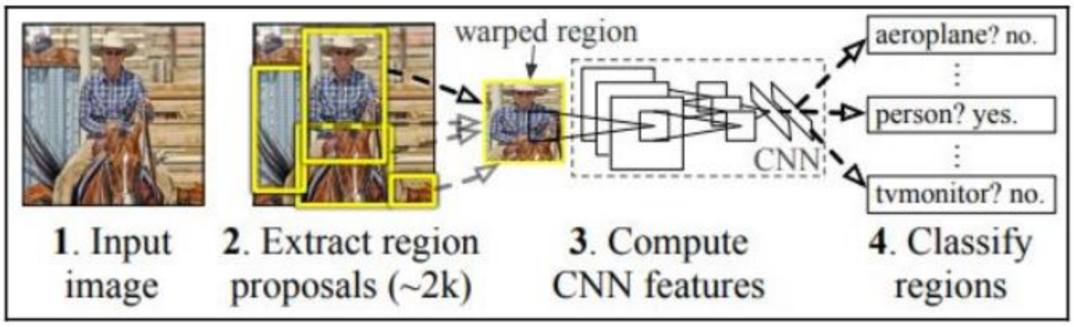
Sliding Windows

Having a predefined bounding box and slide it through the entire image and detect the object along the way. However, this approach requires a lot of computation, there will be many bounding boxes for an object.

Two-Stage Based Approach

Regional Based Networks

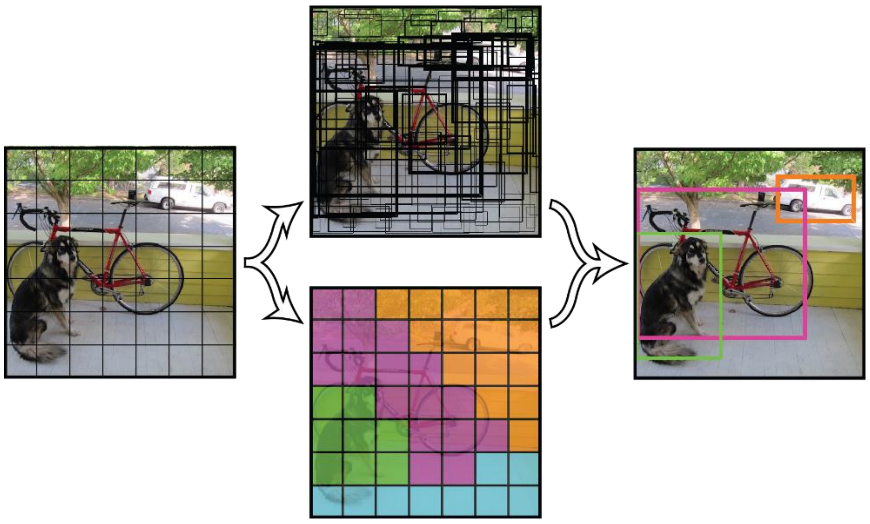
Regional based network uses a deterministic algorithm (selective search)



In the fourth step, the output includes the possible adjustment for the bounding box which is produced by the regional extraction algorithm. However, it's slow, cannot do it in real time and the implementation is complicated.

One-Stage Based Approach

YOLO (You Only Look Once)

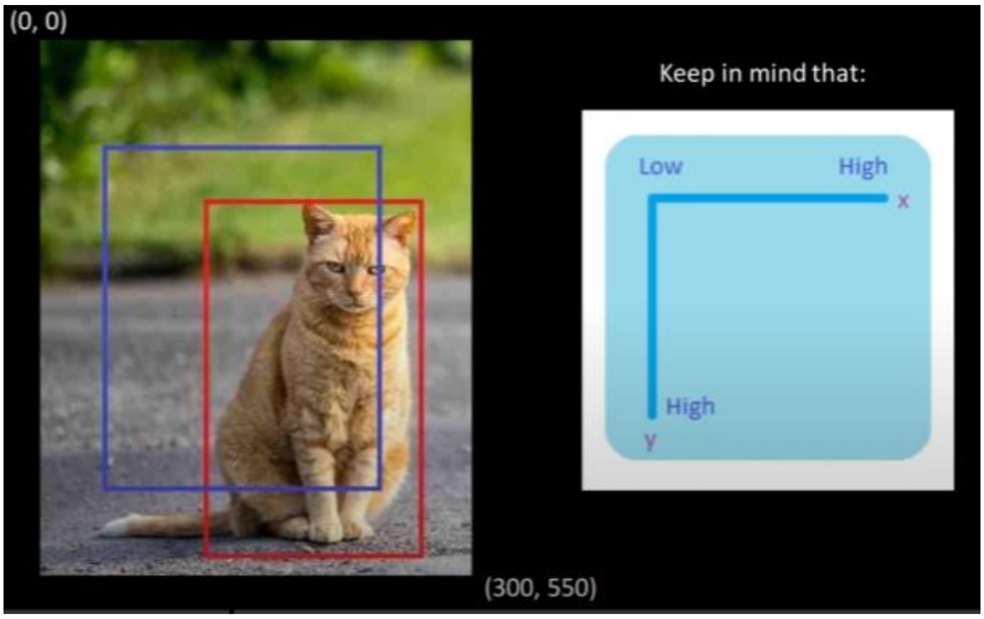


Divide the input image to S x S grid and for each grid is responsible to predict a bounding box of objects whose object center point is within that region. However, it's difficult for a grid to know which object it is responsible so the result will have many bounding boxes and pass through a Non-Max Suppression to clean all those bounding boxes.

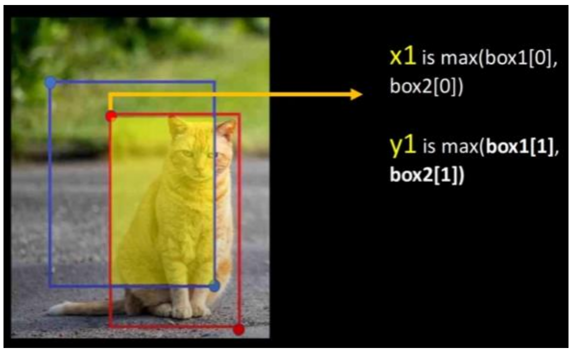
Intersection over Union

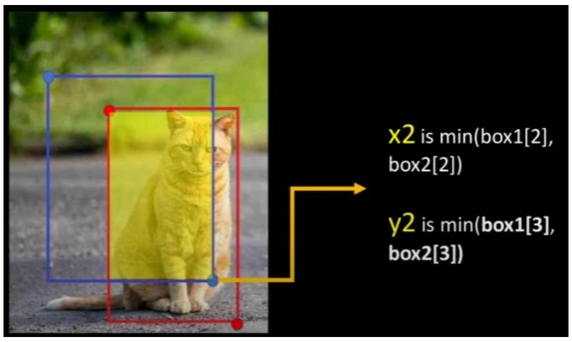
IoU is a way to quantify how good a predicted bounding box for an object.

IoU = Area of Intersection / Area Union



To calculate the corner points of the intersect region.





Implementation

The *boxes_preds* have shape (N, 4) where N stand for the number of the predicted bounding boxes and the second dimension is the four corresponding coordinates.

```
box1_x1 = boxes_preds[..., 0:1]

box1_y1 = boxes_preds[..., 1:2]

box1_x2 = boxes_preds[..., 2:3]

box1_y2 = boxes_preds[..., 3:4]

box2_x1 = boxes_labels[..., 0:1]

box2_y1 = boxes_labels[..., 1:2]

box2_x2 = boxes_labels[..., 2:3]

box2_y2 = boxes_labels[..., 3:4]


x1 = torch.max(box1_x1, box2_x1)

y1 = torch.max(box1_y1, box2_y1)

x2 = torch.max(box1_x2, box2_x2)

y2 = torch.max(box1_y2, box2_y2)


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

Use torch max and min to find the corner points of the intersect region. In this algorithm, the *x1* and *y1* should always bigger than *x2* and *y2*. Therefore, in case there is no intersection that *x1, y1* is smaller than *x2, y2*, *torch.clamp()* should be used so that the minimum value will be 0 and the intersection area is 0.

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

Finally, the area can be calculated and *1e-6* is used for prevent divided by zero.

Non-Max Suppression

This algorithm helps us clean up bounding boxes. From a list of bounding boxes having similar class prediction and location, we take the bounding box with highest confident and calculate the IoU with other bounding boxes. If a bounding box has an IoU higher than a certain threshold, delete or ignore the bounding box with lower confidence. Additionally, each class should be calculated independently.

```
while bboxes:
        chosen_box = bboxes.pop(0)

        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
        ]

        bboxes_after_nms.append(chosen_box)
```

The above code snippet shows the loop to apply NMS to a list of bounding boxes sorted by confidence. The *or* in the *if* statement will be checked only when the first statement is false.

Mean Average Precision

Commonly used to evaluate object detection models. Collect all bounding box predictions for a class on test set and set the IoU threshold as 0.5.

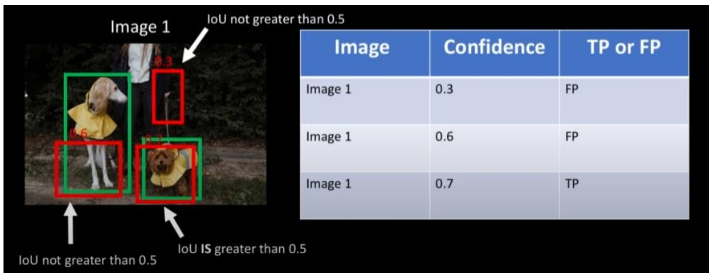


Image	Confidence	TP or FP
Image 1	0.3	FP
Image 1	0.6	FP
Image 1	0.7	TP

Sort by descending confidence score

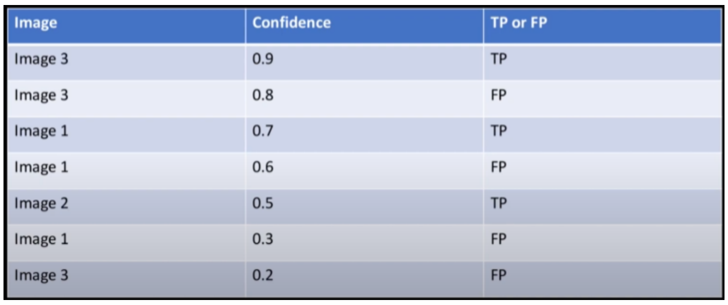


Image	Confidence	TP or FP
Image 3	0.9	TP
Image 3	0.8	FP
Image 1	0.7	TP
Image 1	0.6	FP
Image 2	0.5	TP
Image 1	0.3	FP
Image 3	0.2	FP

Precision *= true positive / true positive + false positive* (Of all bounding box predictions, what fraction was actually correct)

Recall *= true positive / true positive + false negative* (Of all target bounding boxes, what fraction did we correctly detect)

Calculate the precision and recall one by one from the sorted list

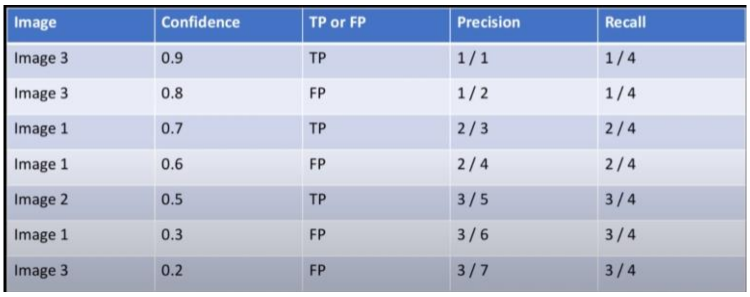
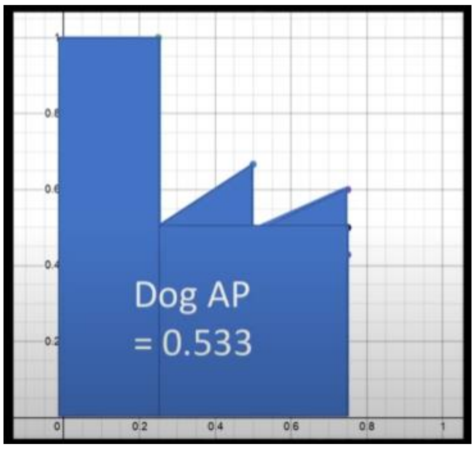


Image	Confidence	TP or FP	Precision	Recall
Image 3	0.9	TP	1 / 1	1 / 4
Image 3	0.8	FP	1 / 2	1 / 4
Image 1	0.7	TP	2 / 3	2 / 4
Image 1	0.6	FP	2 / 4	2 / 4
Image 2	0.5	TP	3 / 5	3 / 4
Image 1	0.3	FP	3 / 6	3 / 4
Image 3	0.2	FP	3 / 7	3 / 4

Plot the precision-recall graph and calculate the area under the line and it will be the Precision for the class



We need repeat all the steps for all the class and calculate the mean. Additionally, we need to calculate Precision for different IoU threshold and similarly calculate their average. For example, *mAP@0.5:0.05:0.95* (0.5, 0.55, 0.6, …) we need to calculate all the precision from 0.5 to 0.95 with step 0.05 and calculate their average.

Implementation

Parameter Formats

```
    """
    Calculates mean average precision
    Parameters:
        pred_boxes (list): list of lists containing all bboxes with each bboxes
        specified as [train_idx, class_prediction, prob_score, x1, y1, x2, y2]
        true_boxes (list): Similar as pred_boxes except all the correct ones
        iou_threshold (float): threshold where predicted bboxes is correct
        box_format (str): "midpoint" or "corners" used to specify bboxes
        num_classes (int): number of classes
    Returns:
        float: mAP value across all classes given a specific IoU threshold
    """
```

Collect all the predicted bounding boxes and ground-truth bounding boxes for a specific class (outer loop by all the classes)

```
        for detection in pred_boxes:
            if detection[1] == c:
                detections.append(detection)

        for true_box in true_boxes:
            if true_box[1] == c:
                ground_truths.append(true_box)
```

Building a dictionary-like data structure for ground-truth bounding boxes with training_idx (which image) as key and the number of bounding boxes as value.

```
        # find the amount of bboxes for each training example
        # Counter here finds how many ground truth bboxes we get
        # for each training example, so let's say img 0 has 3,
        # img 1 has 5 then we will obtain a dictionary with:
        # amount_bboxes = {0:3, 1:5}
        amount_bboxes = Counter([gt[0] for gt in ground_truths])
```

We need to keep track all the ground-truth bounding boxes we have covered. We cannot have multiple prediction bounding boxes for a ground-truth bounding box.

```
        # We then go through each key, val in this dictionary
        # and convert to the following (w.r.t same example):
        # ammount_bboxes = {0:torch.tensor[0,0,0], 1:torch.tensor[0,0,0,0,0]}
        for key, val in amount_bboxes.items():
            amount_bboxes[key] = torch.zeros(val)
```

Sort the prediction bounding boxes by their confidence and create data structure that keep track on the true positive and false positive.

```
        # sort by box probabilities which is index 2
        detections.sort(key=lambda x: x[2], reverse=True)
        TP = torch.zeros((len(detections)))
        FP = torch.zeros((len(detections)))
        total_true_bboxes = len(ground_truths)
```

For each predicted bounding box (for a specific class), calculate IoU with all the ground-truth bounding boxes and keep track the best one. Then if the best IoU is bigger than the threshold, mark the ground-truth bounding box taken and mark true positive for this specific predicted bounding box otherwise, mark the false positive.

```
            for idx, gt in enumerate(ground_truth_img):
                iou = intersection_over_union(
                    torch.tensor(detection[3:]),
                    torch.tensor(gt[3:]),
                    box_format=box_format,
                )
                if iou > best_iou:
                    best_iou = iou
                    best_gt_idx = idx
            if best_iou > iou_threshold:
                # only detect ground truth detection once
                if amount_bboxes[detection[0]][best_gt_idx] == 0:
                    # true positive and add this bounding box to seen
                    TP[detection_idx] = 1
                    amount_bboxes[detection[0]][best_gt_idx] = 1
                else:
                    FP[detection_idx] = 1
            # if IOU is lower then the detection is a false positive
            else:
                FP[detection_idx] = 1
```

Calculate the recall and precisions by using cumulative sum. Additionally, we need to add the first point in our diagram which is (1, 0)

```
TP_cumsum = torch.cumsum(TP, dim=0)
FP_cumsum = torch.cumsum(FP, dim=0)
recalls = TP_cumsum / (total_true_bboxes + epsilon)
precisions = TP_cumsum / (TP_cumsum + FP_cumsum + epsilon)
precisions = torch.cat((torch.tensor([1]), precisions))
recalls = torch.cat((torch.tensor([0]), recalls))
```

Φ *torch.trapz()* calculate the trapezoidal rule which is used for approximating the definite integral of a function by averaging its left and right Riemann sums.

Calculate the average precisions for the specific class by calculating the area under the plotted line which is plotted by the recall (*X*) and precisions (*Y*) and append to the list.

```
# torch.trapz for numerical integration

average_precisions.append(torch.trapz(precisions, recalls))
```

Finally, after calculate through all the classes, calculate the mean of the list containing average precisions.

```
return sum(average_precisions) / len(average_precisions)
```