Computer Vision

T5 Bootcamp by SDAIA

Agenda

- Object Detection
- Training Data for Object Detection
- Bounding Box
- Object Detection Models

Object Detection

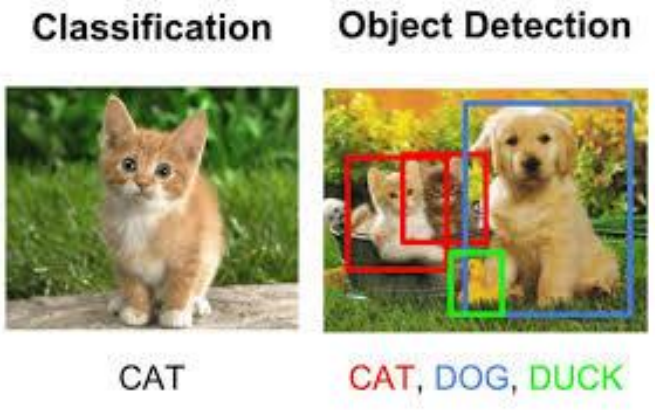
What is Object Detection?

Object Detection, within Computer Vision, involves identifying objects within images or videos. These algorithms commonly rely on machine learning or deep learning methods to generate valuable outcomes.

Object Detection task create a box around the dog that is present in the image and specify the **x** and **y** coordinates of this box

Image Classification vs Object Detection

- **Image Classification** focuses on categorizing entire images into predefined classes.
- **Object Detection** deals with identifying and localizing specific objects within images.



Both tasks require understanding the visual content of images and extracting meaningful features to make accurate predictions.

Image Classification vs Object Detection Tasks

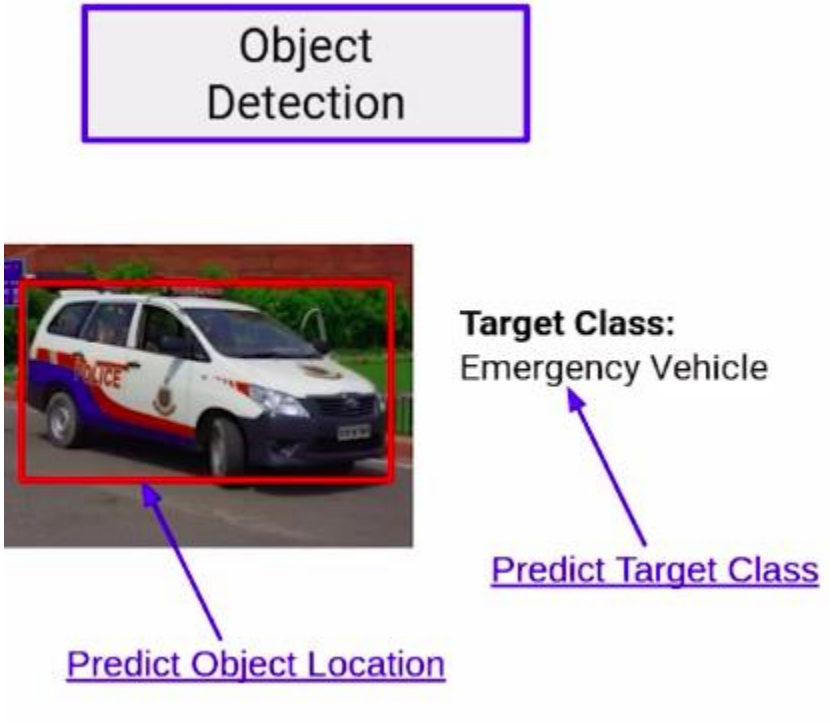
- **Image Classification** has only one task where we had to classify the objects in the image.
- **Object Detection** has two tasks, classify the objects in the image and also locate where these objects are present in the image.

Image Classification

Image Classification + Localization

Object Detection Task

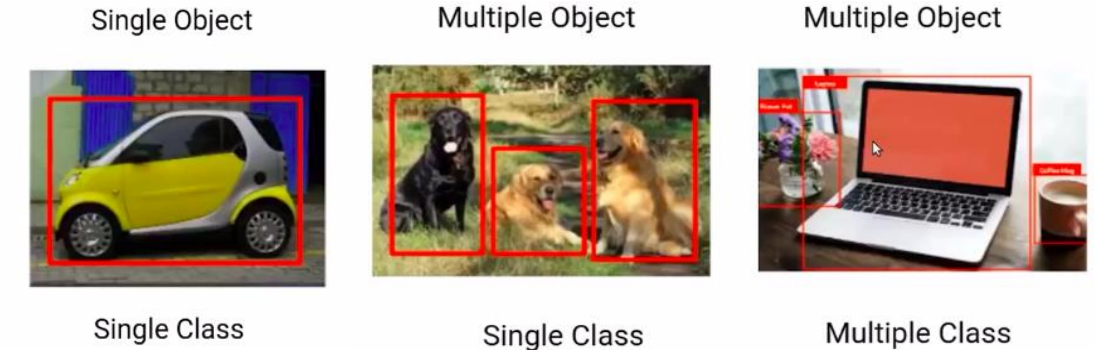
- ✓ Generally, There are three tasks for Object Detection problems:
  1. **Is there an object in the image?**
  2. **Where is this object located?**
  3. **What is this object?**



Object Detection Categories

Generally, The object detection problem can also be divided into multiple categories.

1. **Single Object - Single Class**: if you have multiple images the data set, and all these objects belong to a single class, then this will be an image localization problem.
2. **Multiple Object - Single Class:** if you have multiple images, and within each of these images, there are multiple objects, but all belong to the same class.
3. **Multiple Object - Multiple Class:** if you have multiple images, and within each of these images, there are multiple objects, but they can be of different classes.



Why does Object Detection matter?

- Object detection is a key technology behind **Advanced Driver Assistance Systems (ADAS)** that enable cars to detect driving lanes or perform pedestrian detection to improve road safety.
- Object detection is also useful in applications such as video surveillance or image retrieval systems

Using object detection to identify and locate vehicles.

How does Object Detection work?

Below are the basic steps used to perform an object detection task:

1. **Looking at the Picture:** Imagine a computer looking at a picture.
2. **Finding Clues**: The computer looks for clues like shapes, colors, and patterns in the picture.
3. **Guessing What's There**: Based on those clues, it makes guesses about what might be in the picture.
4. **Checking the Guesses**: It checks each guess by comparing it to things it already knows.
5. **Drawing Boxes:** If it's pretty sure about something, it draws a box around it to show where it thinks the object is.
6. **Making Sure:** Finally, it double-checks its guesses to make sure it got things right and fix any mistakes

Training Data for Object Detection

Training Data for Object Detection

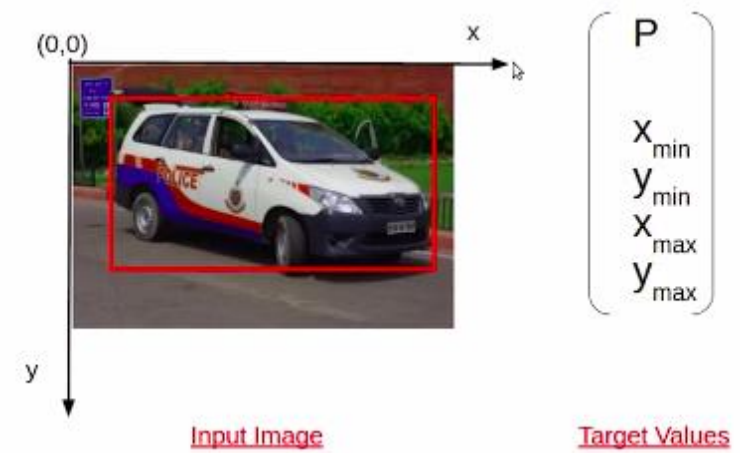
The object detection problem is different from a classification problem.

Input Image

Y = 1

Target Class

Classification Data



Input Image

Target Values

Object Detection Data

Object Detection Target Vector

The target variable has seven values the value **P** denotes the probability of an object whereas the four values **Xmin**, **Ymin**, **Xmax**, and **Ymax** denote the coordinates of the bounding box.

- **(P)** denotes the probability of an object being in the image
- **Xmin** and **Ymin** will be the top left corner of the bounding box
- **Xmax** and **Ymax** will be the bottom right corner of the bounding box.
- Two additional values **c1** and **c2** denoting which class does the object present in image.

Sample Input Images

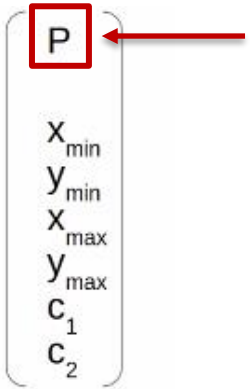
$$\begin{bmatrix} P \\ x_{min} \\ y_{min} \\ x_{max} \\ y_{max} \\ c_1 \\ c_2 \end{bmatrix}$$

Object Detection Target Variable *(cont'd)*

The target variable **(P)** answers only two questions?

1. **Is there an object present in the image?**
   - If an object is not present then p will be zero and when there is an object present in the image p will be one.
2. **If an object is present in the image where is the object located?**
   - You can find the object location using the coordinates of the bounding box.

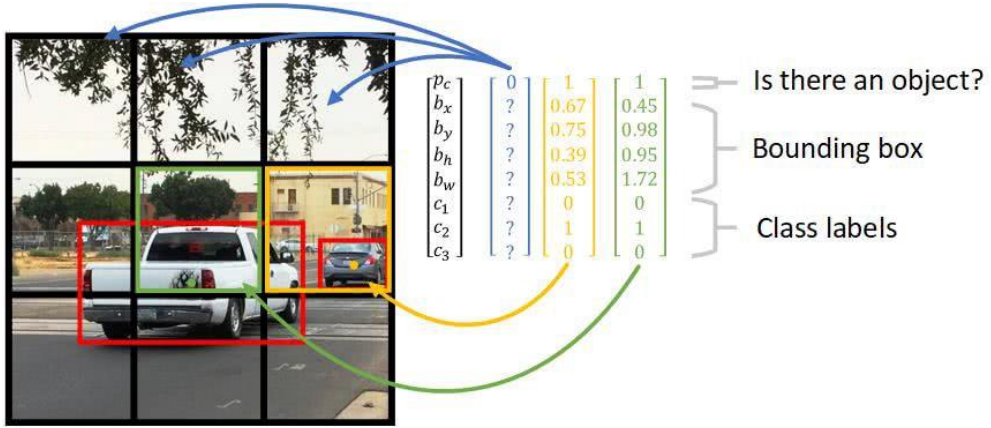
Sample Input Images



Object Detection Example

For example, In the below picture:

- The probability of an object (P) present in the image as 1.
- Xmin, Ymin, Xmax, and Ymax are the coordinates of the bounding box.
- There is also c2 is equal to 1 indicating cars while c1 (buildings) & c3 (pedestrian) would be 0.



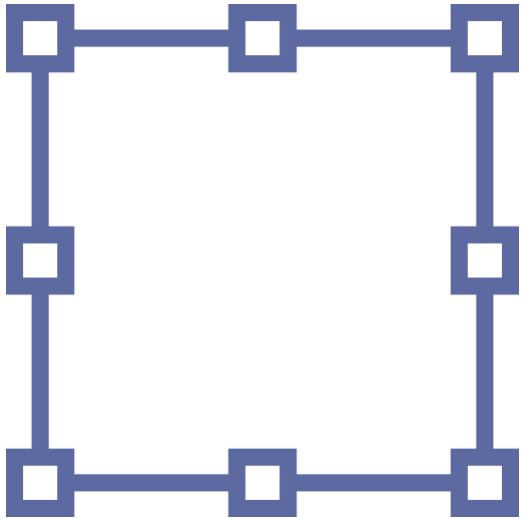
This is how the training data should look like in any Object Detection task

Object Detection Bounding Box

Object Detection Bounding Box

Before moving into depth, we need to know a few concepts regarding images such that:

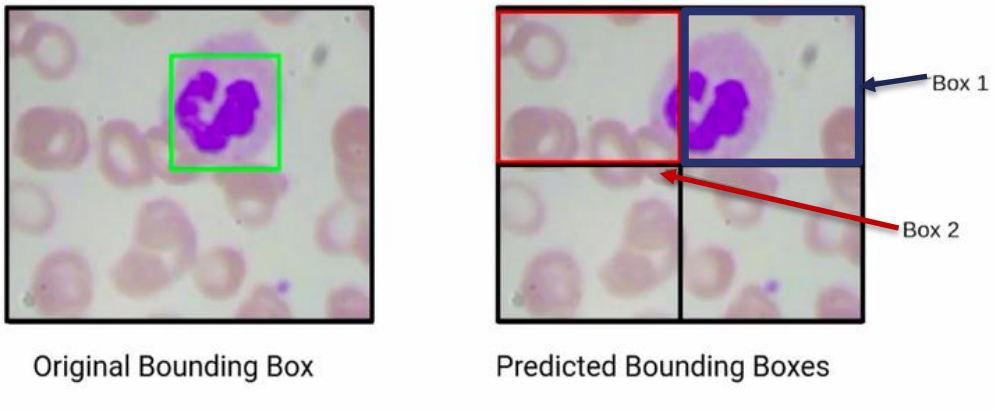
1. **How to do Bounding Box Evaluation?**
2. **How to calculate Intersection over Union?**
3. **Evaluation Metric - mean Average Precision**



Bounding Box Evaluation

Any model needs to be evaluated, so in Object Detection we need to assess model performance w.r.t to predicted target.

- Here we have two bounding boxes, box1 and box2. which of these two boxes is more accurate

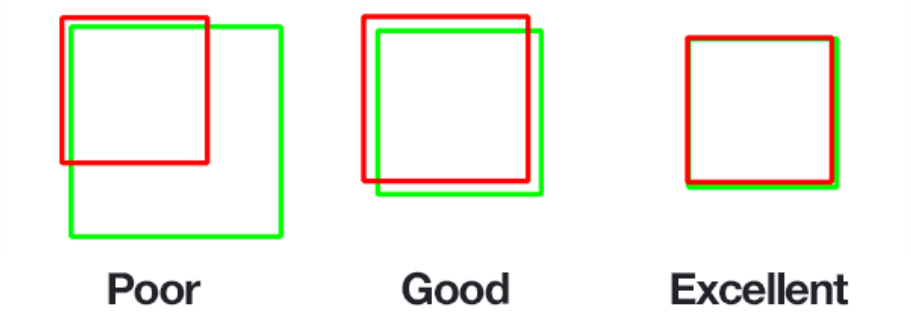


Which of these two boxes is more accurate?

Intersection over Union (IoU)

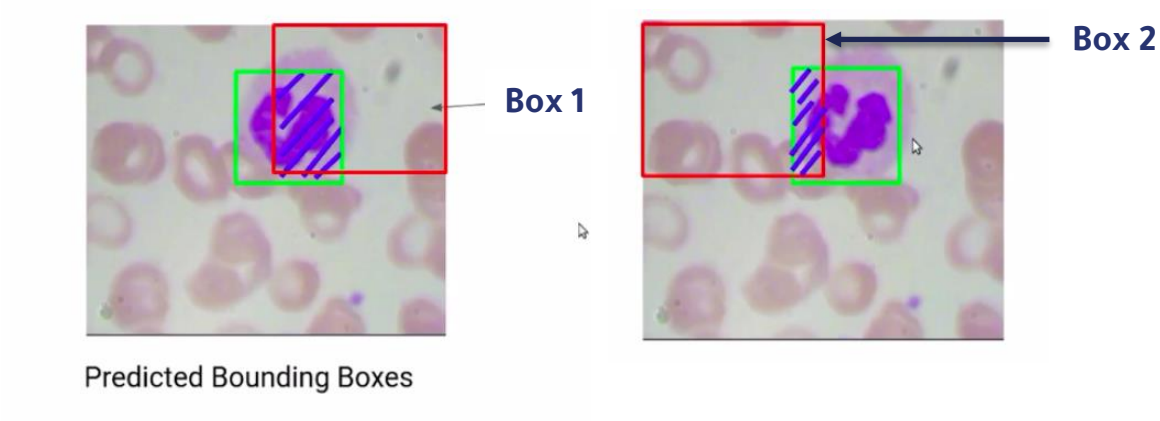
Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. It is used to compare the actual, and the predicted bounding boxes.

- It finds out the overlap of the actual, and the predicted bounding box, to be able to decide which bounding box is a better prediction.
- The bounding box that has a higher overlap with the actual bounding box is a better prediction.



Intersection over Union (IoU) *(cont'd)*

- The intersection of the left bounding box is certainly 100%.
- In the right image, the intersection of this predicted bounding box is just 70%.



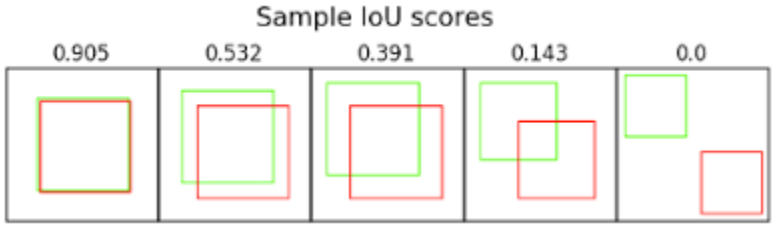
The bounding box on the left is a better prediction.

Intersection over Union (IoU) *(cont'd)*

Intersection over Union can be used as:

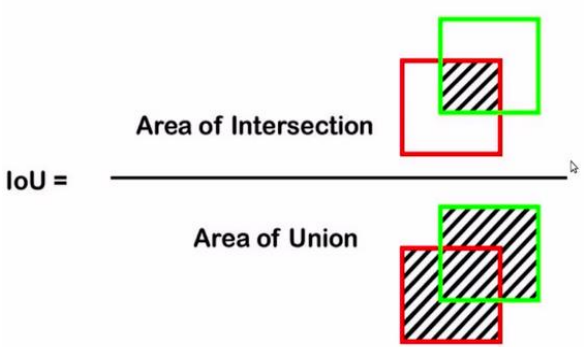
1. Selection criteria for the best bounding box
2. Evaluation Metric.

If the intersection over union is high, then the predicted bounding boxes are close to the actual bounding box, and we can say that the model is performing well.



Calculation Intersection over Union

The mathematical representation is:



Where,

- Area of Intersection = Common area shared by the two bounding boxes (Overlap)
- Area of Union = Total area covered by the two bounding boxes.

Intersection over Union: **Area of Intersection**

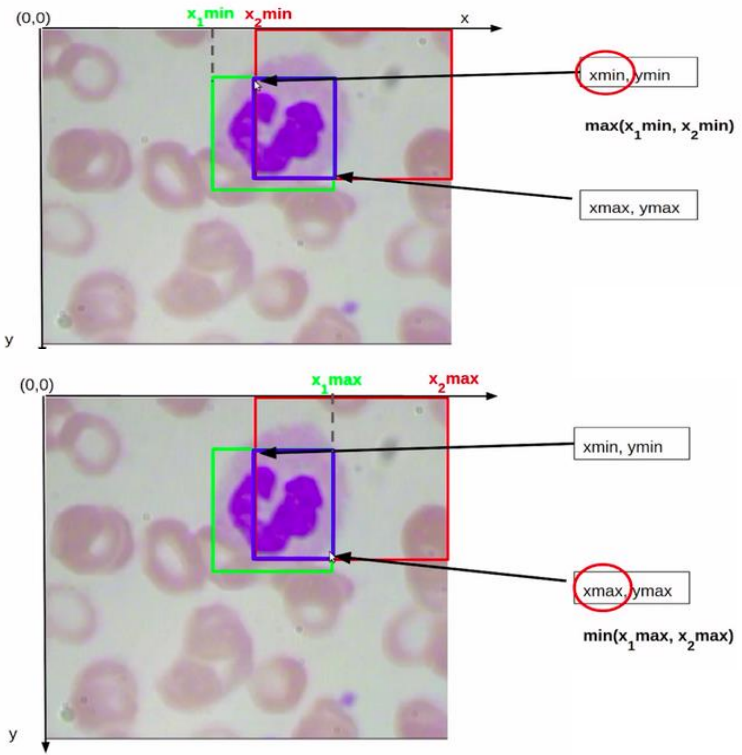
Coordinates: Utilize Xmin, Ymin, Xmax, and Ymax coordinates for bounding boxes.

Xmin Determination:

- Use Xmin values for both bounding boxes (X1min and X2min).
- Xmin for the intersection is the maximum value between X1min and X2min.

Xmax Determination:

- Compare X1max and X2max values.
- Xmax for the intersection is the minimum value between X1max and X2max.



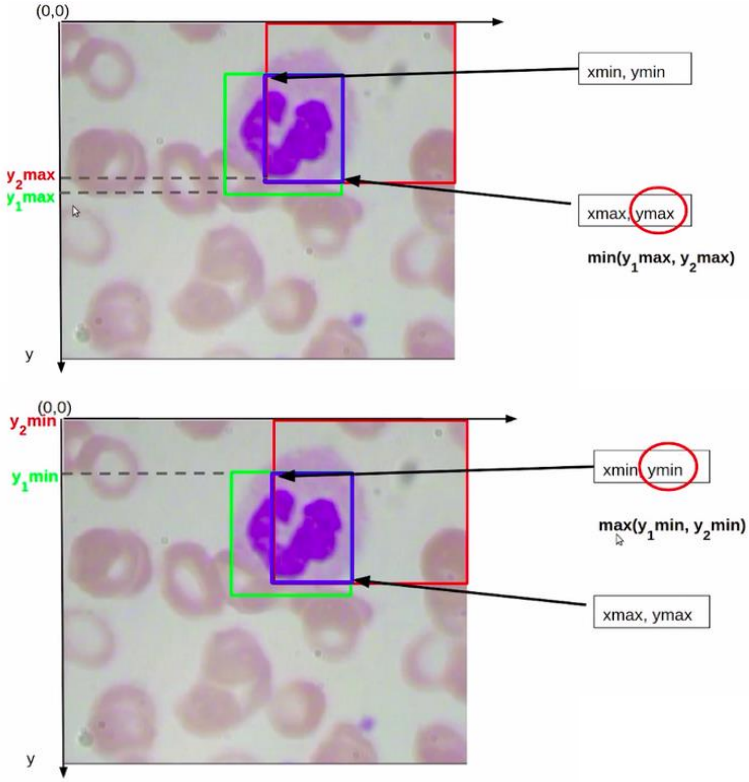
Intersection over Union: **Area of Intersection** *(cont'd)*

Ymin Determination:

- Use Ymin values for both bounding boxes (Y1min and Y2min).
- Ymin for the intersection is the maximum value between Y1min and Y2min.

Ymax Determination:

- Compare Y1max and Y2max values.
- Ymax for the intersection is the minimum value between Y1max and Y2max.



Intersection over Union: **Area of Intersection** *(cont'd)*

Once Xmin, Ymin, Xmax, and Ymax are determined:

Length of the intersection:

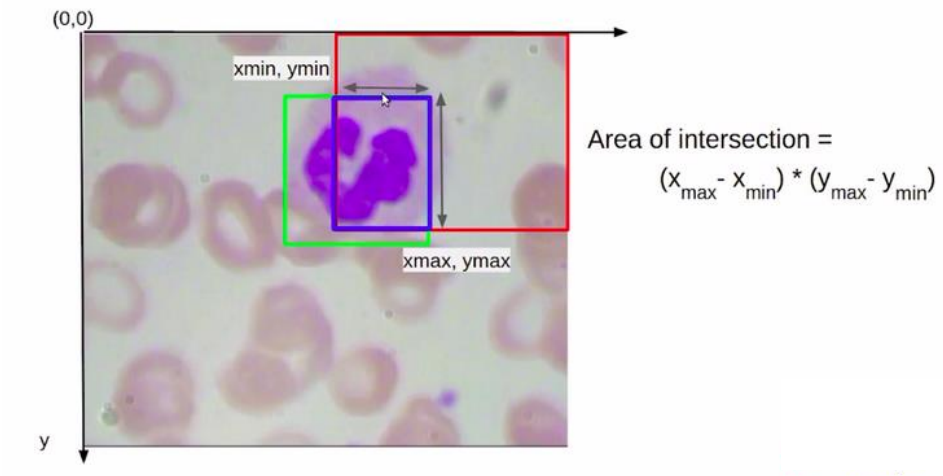
- Xmax - Xmin.

Width of the intersection:

- Ymax - Ymin.

Area of the intersection:

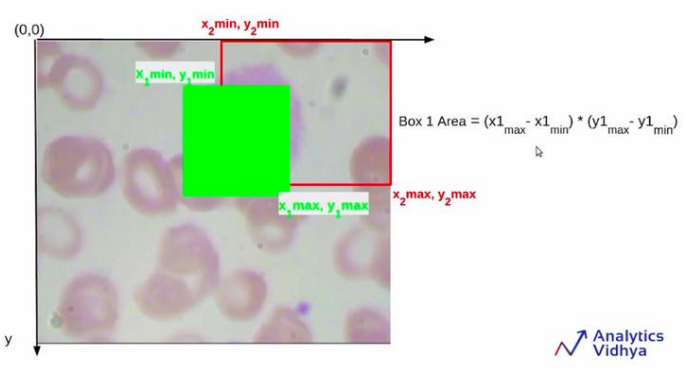
- Length x Width.

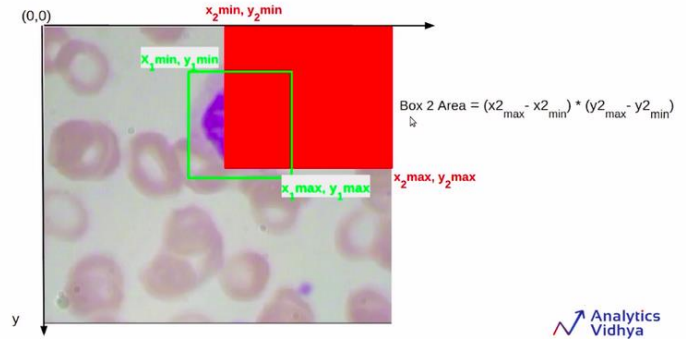


Intersection over Union: **Area of union**

Area of Union Calculation:

- Utilize the coordinates of both bounding boxes (green and red).
- Find the area of Box 1 (green bounding box):
  - ✓ Calculate length × width of the green shaded region.
- Note that the blue shaded region is counted twice in the areas of both Box 1 and Box 2 (green and red bounding boxes);
  - ✓ Subtract the area of intersection once from the total.
- The area of union is the sum of the areas of Box 1 and Box 2, minus the intersection area.

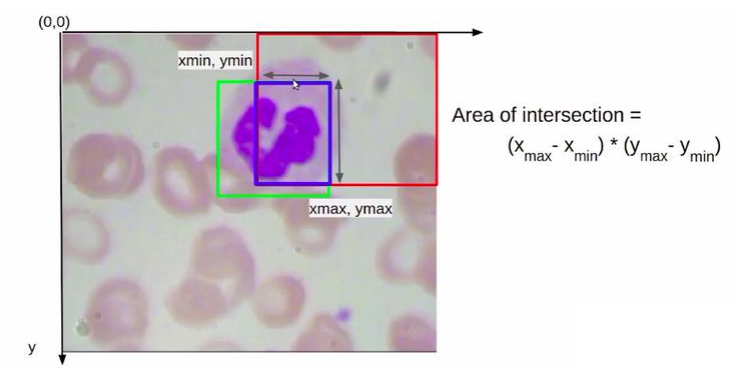


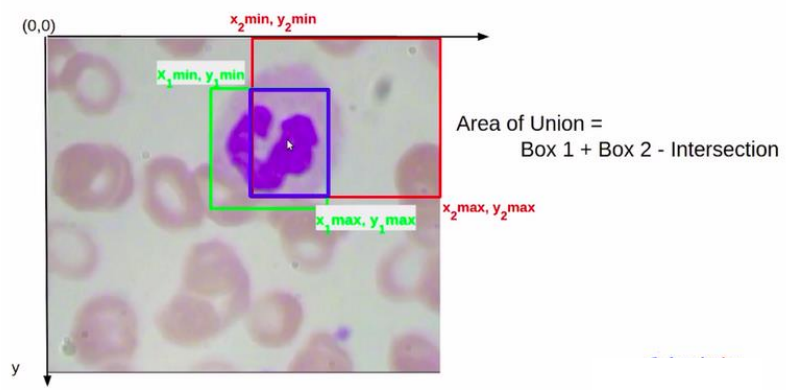


Intersection over Union: **Final Calculation**

Intersection Over Union (IoU):

- Calculate IoU as the area of intersection divided by the area of union.
- IoU = (Area of Intersection) / (Area of Union).





Intersection over Union: **Python Code**

```python
def calculate_iou(box1, box2):

    # Extract coordinates of the bounding boxes
    x1_box1, y1_box1, x2_box1, y2_box1 = box1
    x1_box2, y1_box2, x2_box2, y2_box2 = box2

    # Calculate the coordinates of the intersection rectangle
    x_left = max(x1_box1, x1_box2)
    y_top = max(y1_box1, y1_box2)
    x_right = min(x2_box1, x2_box2)
    y_bottom = min(y2_box1, y2_box2)

    # Calculate the area of intersection rectangle
    intersection_area = max(0, x_right - x_left + 1) * max(0, y_bottom - y_top + 1)

    # Calculate the area of both bounding boxes
    box1_area = (x2_box1 - x1_box1 + 1) * (y2_box1 - y1_box1 + 1)
    box2_area = (x2_box2 - x1_box2 + 1) * (y2_box2 - y1_box2 + 1)
```


Intersection over Union: **Python Code**

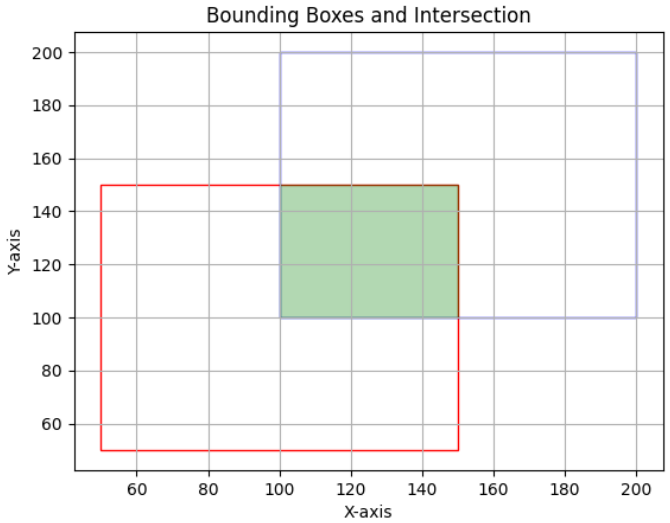
```python
    # Calculate the Union area by subtracting the intersection area
    # from the sum of areas of both bounding boxes
    union_area = box1_area + box2_area - intersection_area

    # Calculate IoU as intersection area divided by union area
    iou = intersection_area / union_area

    return iou

# Example usage:
box1 = (50, 50, 150, 150)  # Format: (x1, y1, x2, y2)
box2 = (100, 100, 200, 200)
iou_score = calculate_iou(box1, box2)
print("Intersection over Union (IoU):", iou_score)

Intersection over Union (IoU): 0.14611538677602381
```



Evaluation Metrics for Objection Detection

Evaluation Metrics for Objection Detection:

1. Intersection over Union(IoU)
2. Mean Average Precision(mAP)

We have previously discussed the intersection over the union (IoU) and How it can be used to evaluate the model performance by comparing the predicted bounding boxes with the actual bounding boxes.

Another popularly used metric is Mean Average Precision.

Mean Average Precision (mAP)

To understand the concept of mean average precision and its application in evaluation metrics for object detection, you need to use precision:

- Precision measures the proportion of true positives among all predicted positives.
- Calculated as true positives / (true positives + false positives).

Predicted	IoU	TP/ FP
Bounding Box 1	0.7	TP
Bounding Box 2	0.2	FP
Bounding Box 3	0.9	TP
Bounding Box 4	0.8	TP
Bounding Box 5	0.4	FP

Example of Precision Calculation:

✓ Given bounding box predictions and IoU scores, classify predictions as true positives or false positives based on a threshold (e.g., 0.5).

✓ Calculate precision based on true positives and false positives.

Mean Average Precision (mAP) *(cont'd)*

Average Precision:

- Average precision computes the mean of precision values across the dataset.

Example of Average Precision Calculation:

- Calculate precision for each bounding box individually by dividing the number of true positives by the sum of true positives and false positives.
- Average the precision values across all bounding boxes to obtain average precision.

Predicted	IoU	TP/ FP	$Precision_i$
Bounding Box 1	0.7	TP	1
Bounding Box 2	0.2	FP	1
Bounding Box 3	0.9	TP	0.66
Bounding Box 4	0.8	TP	0.75
Bounding Box 5	0.4	FP	0.75

$$AP = \frac{1}{N} \Sigma precision_i$$

$$= 1/5\ (4.16)$$

$$= 0.832$$

Mean Average Precision (mAP):

- mAP is calculated by averaging the average precision values across all classes.
- Provides a comprehensive evaluation metric for object detection performance across multiple classes.

Mean Average Precision (mAP): **Python Code**

```python
def calculate_average_precision(precision, recall):
    """
    Calculate the average precision from precision and recall values.

    """
    # Ensure the precision and recall lists have the same length
    if len(precision) != len(recall):
        raise ValueError("Precision and recall lists must have the same length.")

    # Calculate the area under the precision-recall curve (AP)
    ap = 0
    for i in range(1, len(precision)):
        ap += (recall[i] - recall[i - 1]) * precision[i]

    return ap
```


Mean Average Precision (mAP): **Python Code**

```python
def calculate_mean_average_precision(average_precisions):
    """
    Calculate the Mean Average Precision (mAP) from a list of average precision values.

    """
    # Calculate the mean of the average precision values
    mAP = sum(average_precisions) / len(average_precisions)

    return mAP

# Example usage:
precision = [0.8, 0.75, 0.7, 0.65, 0.6]  # Precision values for different recall levels
recall = [0.2, 0.4, 0.6, 0.8, 1.0]  # Recall values corresponding to the precision values
ap = calculate_average_precision(precision, recall)
print("Average Precision (AP):", ap)

# List of average precision values for different classes
average_precisions = [0.85, 0.78, 0.92, 0.65, 0.73]
mAP = calculate_mean_average_precision(average_precisions)
print("Mean Average Precision (mAP):", mAP)
```

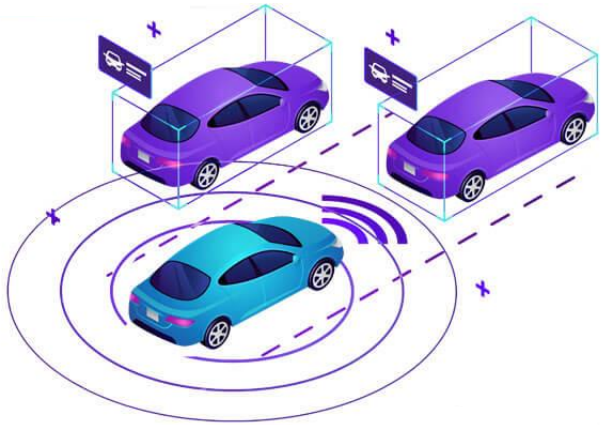

Object Detection Models

Object Detection Models

We'll explore some of the most influential architectures in the field. These models have significantly advanced the state-of-the-art in object detection tasks, offering varying levels of speed and accuracy.

Through our exploration of these models, we'll delve into their architectures, strengths, weaknesses, and real-world applications. We'll focus on four key models:

1. **R-CNN**
2. **Fast R-CNN**
3. **Faster R-CNN**
4. **YOLO**

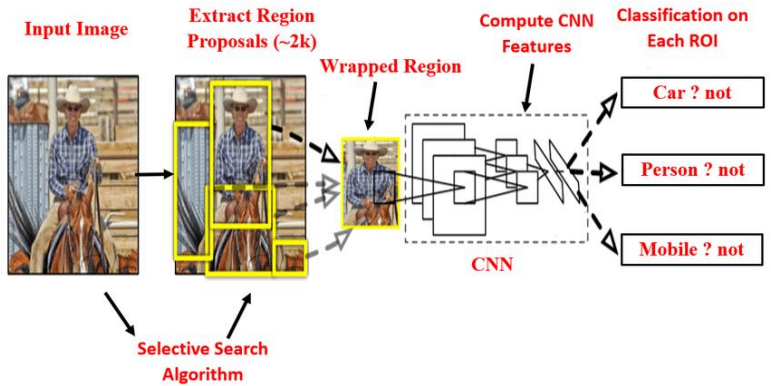


R-CNN

Region-based Convolutional Neural Network (R-CNN) is a type of deep learning architecture used for object detection in computer vision tasks.

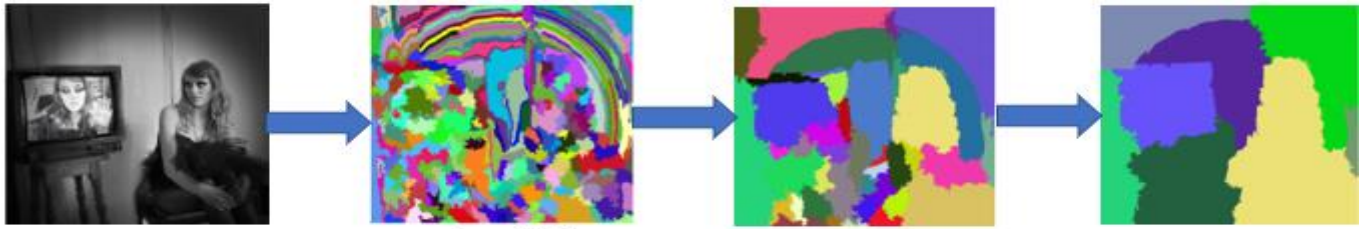
- R-CNN was one of the pioneering models that helped advance the object detection field by combining the power of convolutional neural networks and region-based approaches.
- R-CNN operates in a sequential manner, effectively combining:

1. Region Proposal
2. Feature Extraction
3. Object Classification
4. Bounding Box Regression
5. Non-Maximum Suppression (NMS)



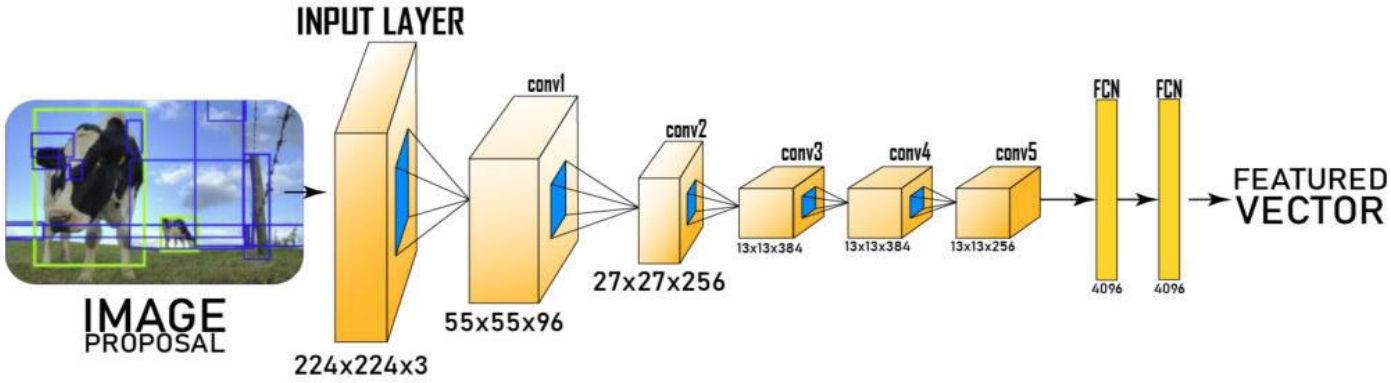
R-CNN: **Region Proposal**

- R-CNN starts by dividing the input image into multiple regions or subregions. These regions are referred to as "**Region proposals**" or "**Region candidates**."
- The region proposal step is responsible for generating a set of potential regions in the image that are likely to contain objects.
- R-CNN does not generate these proposals itself; instead, it relies on external methods like Selective Search or EdgeBoxes to generate region proposals.
- **Selective Search**, for example, operates by merging or splitting segments of the image based on various image cues like color, texture, and shape to create a diverse set of region proposals.



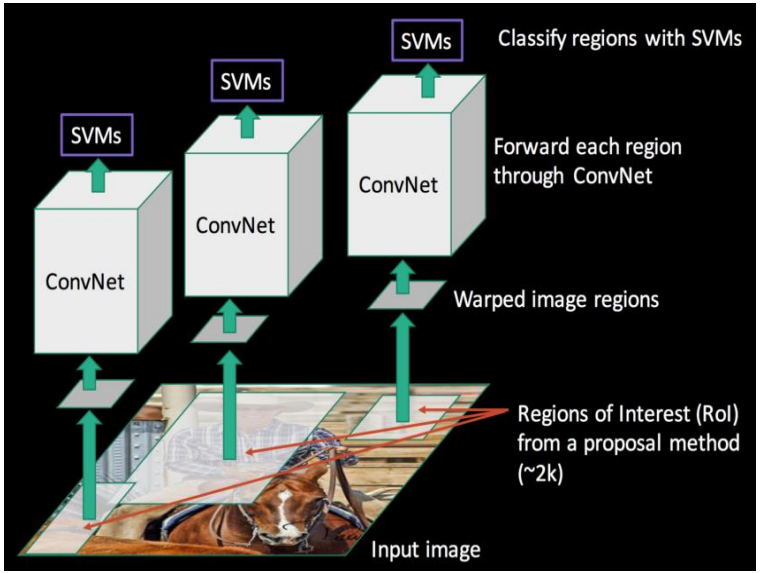
R-CNN: **CNN Architecture**

- Once the region proposals are generated, approximately 2,000 regions are extracted and warped to a consistent input size that the CNN expects *(e.g., 224x224 pixels)* and then it is passed through the CNN to extract features.
- The CNN model used here is a pre-trained AlexNet model, which is the state-of-the-art CNN model at that time for image classification.
- The last SoftMax layer is removed to get the (1, 4096) feature vector. We pass this feature vector into SVM and bounding box regressor.



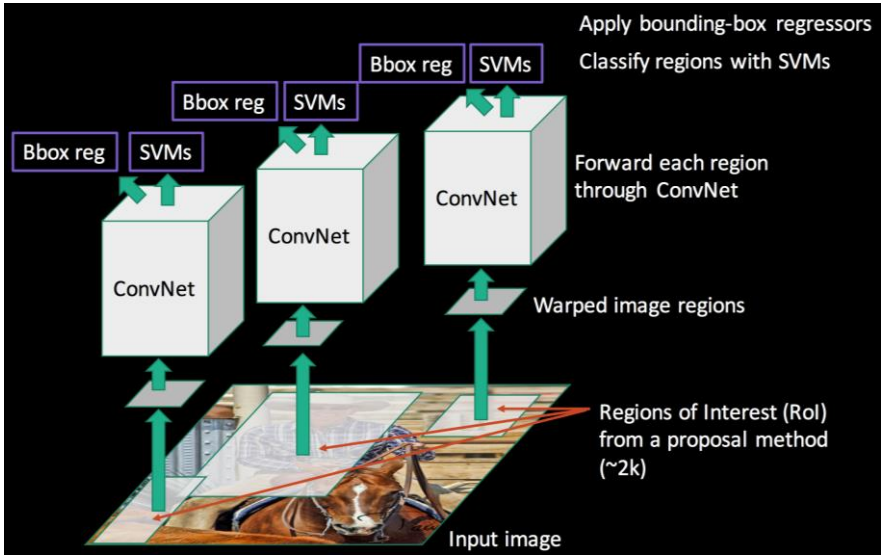
R-CNN: **SVM Classification**

- The extracted feature vectors from the region proposals are fed into a separate machine learning classifier for each object class of interest.
- R-CNN typically uses Support Vector Machines (SVMs) for classification. For each class, a unique SVM is trained to determine whether or not the region proposal contains an instance of that class.
- However, there is an issue with training with SVM is that we required AlexNet feature vectors for training the SVM class.
- So, we could not train AlexNet and SVM independently in paralleled manner.



R-CNN: **Bounding Box Regression**

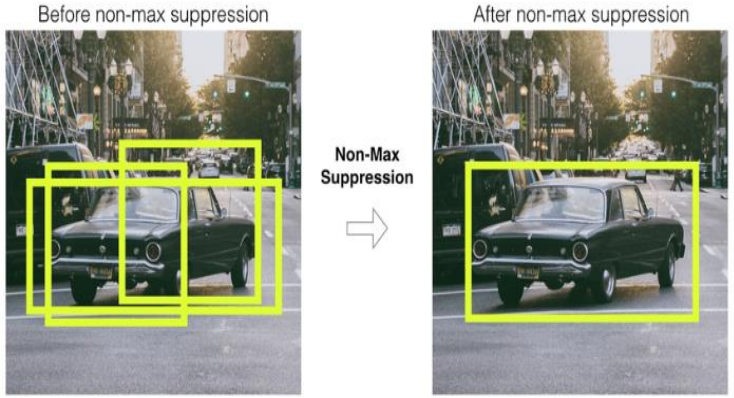
- In addition to classifying objects, R-CNN also performs bounding box regression.
- For each class, a separate regression model is trained to refine the location and size of the bounding box around the detected object.
- The bounding box regression helps improve the accuracy of object localization by adjusting the initially proposed bounding box to better fit the object's actual boundaries.



R-CNN: **Non-Maximum Suppression** (NMS)

After classifying and regressing bounding boxes for each region proposal, R-CNN applies non-maximum suppression to eliminate duplicate or highly overlapping bounding boxes.

Non-Maximum Suppression (NMS) is a crucial step in the R-CNN pipeline to refine object proposals and eliminate redundant detections. Here's how it works:



1. Select the proposal with the highest confidence score and add it to the final proposal list.
2. Compare this selected proposal with all other proposals and calculate the Intersection over Union (IOU) metric.
3. If the IOU of any proposal with the selected one exceeds a predefined threshold, remove that proposal from consideration.
4. Repeat the process by selecting the proposal with the next highest confidence score from the remaining proposals.
5. Continue until no more proposals are left in consideration.

Advantages of R-CNN

Below are a few of the key strengths of the R-CNN architecture.

- **Accurate Object Detection**: R-CNN provides accurate object detection by leveraging region-based convolutional features. It excels in scenarios where precise object localization and recognition are crucial.
- **Robustness to Object Variations:** R-CNN models can handle objects with different sizes, orientations, and scales, making them suitable for real-world scenarios with diverse objects and complex backgrounds.
- **Flexibility:** R-CNN is a versatile framework that can be adapted to various object detection tasks, including instance segmentation and object tracking. By modifying the final layers of the network, you can tailor R-CNN to suit your specific needs..

Problems with R-CNN

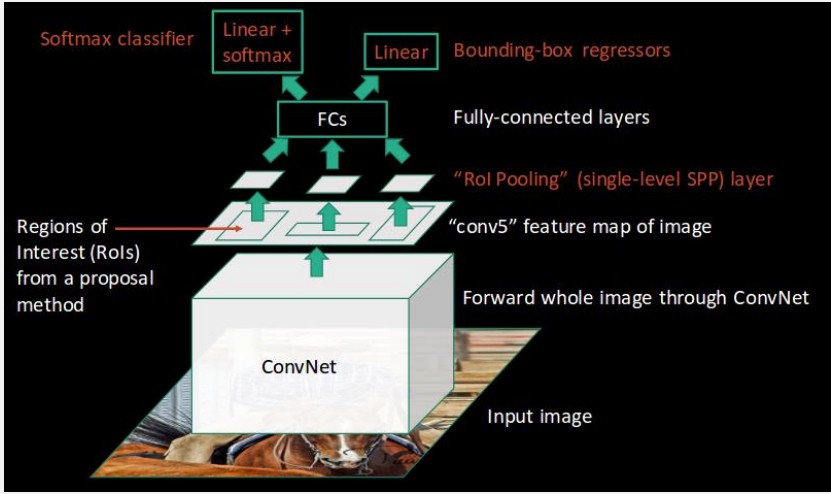
Below are a few disadvantages of the R-CNN architecture.

- **Computational Complexity:** R-CNN is computationally intensive. It involves extracting region proposals, applying a CNN to each proposal, and then running the extracted features through a classifier. It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
- **Slow Inference:** Due to its sequential processing of region proposals, R-CNN is relatively slow during inference. Real-time applications may find this latency unacceptable. It cannot be implemented real time as it takes around 47 seconds for each test image.
- **Overlapping Region Proposals:** R-CNN may generate multiple region proposals that overlap significantly, leading to redundant computation and potentially affecting detection performance.
- **R-CNN is Not End-to-End:** Unlike more modern object detection architectures like Faster R-CNN, R-CNN is not an end-to-end model. It involves separate modules for region proposal and classification, which can lead to suboptimal performance compared to models that optimize both tasks jointly.

Fast R-CNN

The same author of the previous paper(R-CNN) solved some of the drawbacks of R-CNN to build a faster object detection algorithm and it was called Fast R-CNN.

- Fast R-CNN, on the other hand, passes the entire image to ConvNet which generates regions of interest *(instead of passing the extracted regions from the image).*
- Also, instead of using three different models (*as we saw in R-CNN)*, it uses a single model which extracts features from the regions, classifies them into different classes, and returns the bounding boxes.

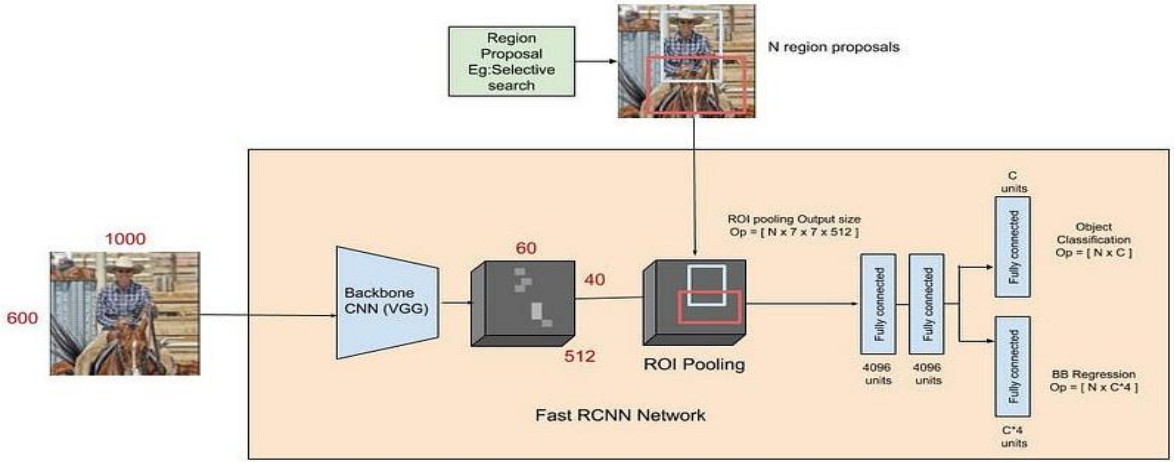


Fast R-CNN *(cont'd)*

- The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map.
- From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.
- From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.
- The reason "Fast R-CNN" is faster than R-CNN is because you don't have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it.

Fast R-CNN Architecture

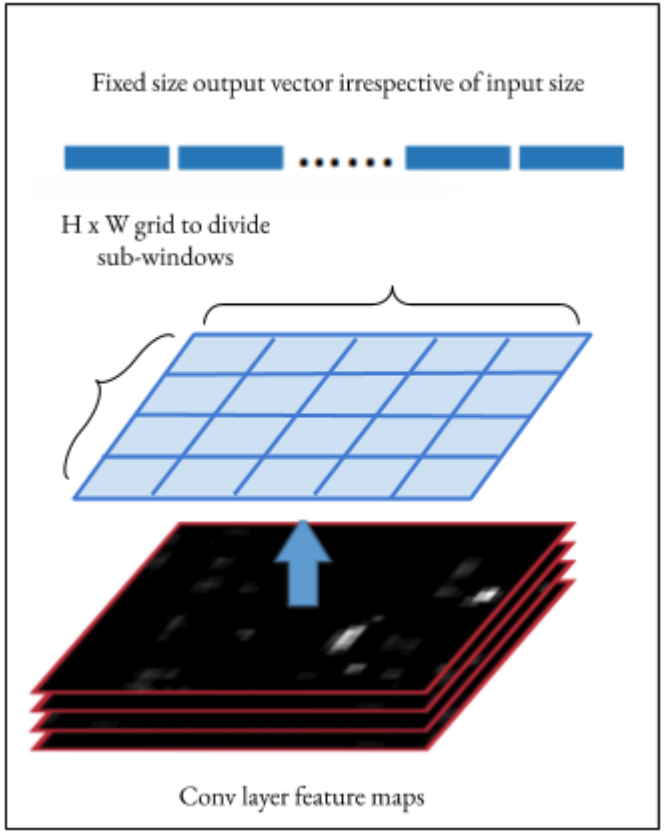
- The Fast R-CNN consists of a CNN *(usually pre-trained on the ImageNet classification task)* with its final pooling layer replaced by an "ROI pooling" layer and its final FC layer is replaced by two branches:
  - Category (K + 1) SoftMax layer branch
  - Category-specific bounding box regression branch.
- The entire image is fed into the backbone CNN and the features from the last convolution layer are obtained.



Fast R-CNN Architecture *(cont'd)*

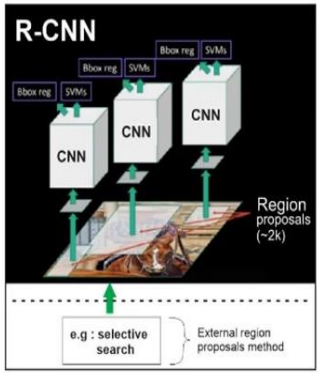
ROI pooling:

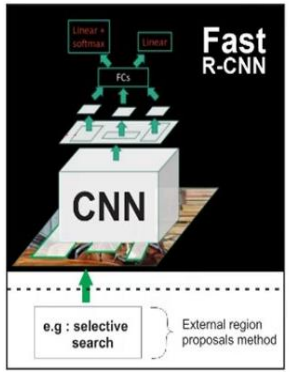
- The ROI pooling layer is a specialized version of the Spatial Pyramid Pooling (**SPP**) layer, with a single pyramid level.
- Features from selected proposal windows are divided into sub-windows of size h/H by w/W.
- Pooling operations are performed in each sub-window to generate fixed-size output features (H x W).
- Output size (H x W) is chosen to be compatible with the network's first fully-connected layer.
- In **ROI pooling, the pooling operation is conducted individually for each channel, mirroring the approach of regular pooling**.
- Output features (N x 7 x 7 x 512) are fed into successive fully connected (FC) layers, SoftMax classification branch, and bounding box regression branch.



Fast R-CNN vs R-CNN

- Fast R-CNN achieved state of the art performance at the time of its publication on VOC12, VOC10(with additional data) and VOC07.
- Fast R-CNN processes images 45x faster than R-CNN at test time and 9x faster at train time.
- On further using truncated SVD, the detection time of the network is reduced by more than 30% with just a 0.3 drop in mAP.





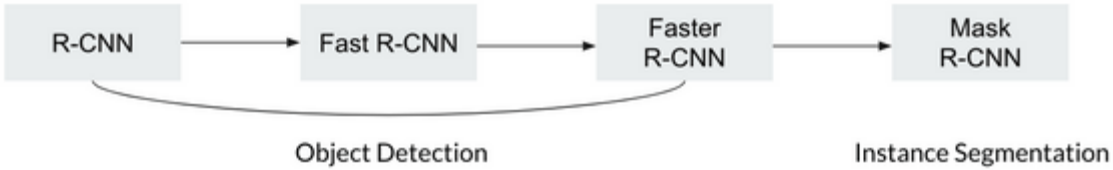
	R-CNN	Fast R-CNN
Test time per image	50 seconds	2 seconds
Speed-up	1x	25x
mAP (VOC 2007)	66.0%	66.9%

VOC12 refers to the PASCAL Visual Object Classes Challenge 2012, which is a benchmark dataset widely used in computer vision research for object detection, classification, and segmentation tasks.

Faster R-CNN

The most widely used state of the art version of the R-CNN family – **Faster R-CNN.** Faster R-CNN short for "Faster Region-Convolutional Neural Network" was introduced by Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun in 2015.

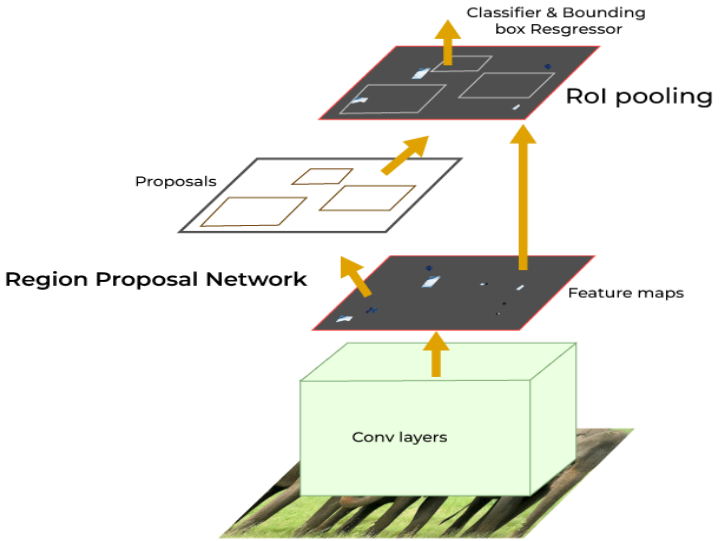
- In the R-CNN family of papers, the evolution between versions was usually in terms of computational efficiency reduction in test time, and improvement in performance (mAP).
- Both of the previous algorithms *(R-CNN & Fast R-CNN)* uses selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network.
- **The primary goal of the Faster R-CNN network is to develop a unified architecture that not only detects objects within an image but also locates the objects precisely in the image.**



Faster R-CNN Architecture

Faster R-CNN architecture consists of two components:

1. **Region Proposal Network (RPN):**
   - Responsible for generating region proposals in an image using convolutional-based networks, enhancing feature representation, and reducing proposal time.
2. **Fast R-CNN detector:**
   - Performs object detection within the generated region proposals by utilizing region of interest pooling, feature extraction, fully connected layers, object classification, and bounding box regression.

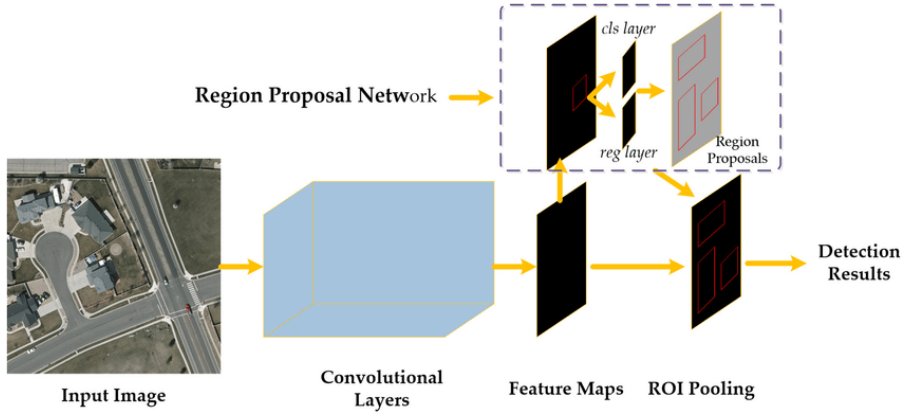


Faster R-CNN: **Region Proposal Network (RPN)**

- Faster R-CNN introduced the Region Proposal Network (RPN) to overcome the computational inefficiencies of traditional region proposal methods like Selective Search.
- RPN reduces proposal time per image significantly, from 2 seconds to just 10 milliseconds.

Purpose of RPN:

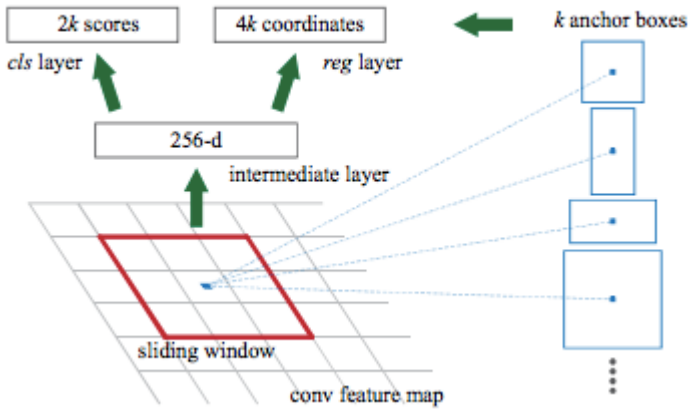
- RPN generates region proposals, guiding subsequent detection stages to focus on potential object locations within images.



Faster R-CNN: **Region Proposal Network (RPN)** *(cont'd)*

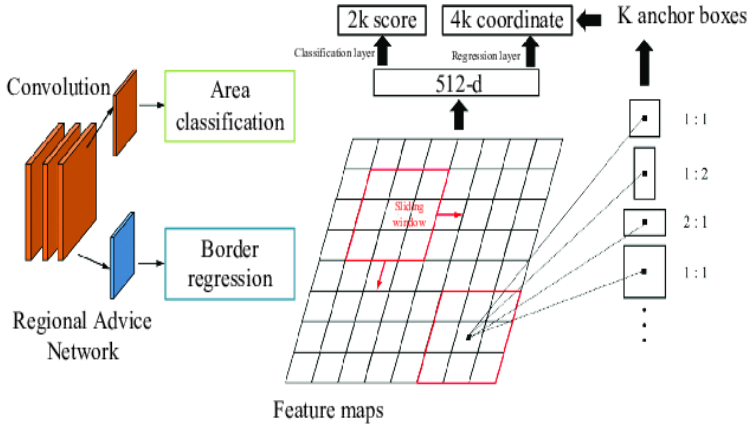
The key components of the Region Proposal Network are as follows:

1. **Anchor Boxes:**
   - Predefined boxes with various scales and aspect ratios used for generating region proposals.
   - Each anchor box has scale and aspect ratio parameters.
2. **Sliding Window Approach:**
   - RPN operates as a sliding window mechanism over feature maps.
   - Utilizes a small convolutional network to process features within the receptive field of the sliding window.



Faster R-CNN: **Region Proposal Network (RPN)** *(cont'd)*

3. **Objectness Score:**
   - Represents the probability that an anchor box contains an object of interest rather than background.
   - Predicted for each anchor during training.
4. **IoU (Intersection over Union):**
   - Metric measuring overlap between two bounding boxes.
   - Calculated as the ratio of area of intersection to area of union.
5. **Non-Maximum Suppression (NMS):**
   - Used to remove redundancy among overlapping proposals.
   - Selects top-N anchor boxes with the highest objectness scores.
   - Ensures final proposals are accurate and non-overlapping.

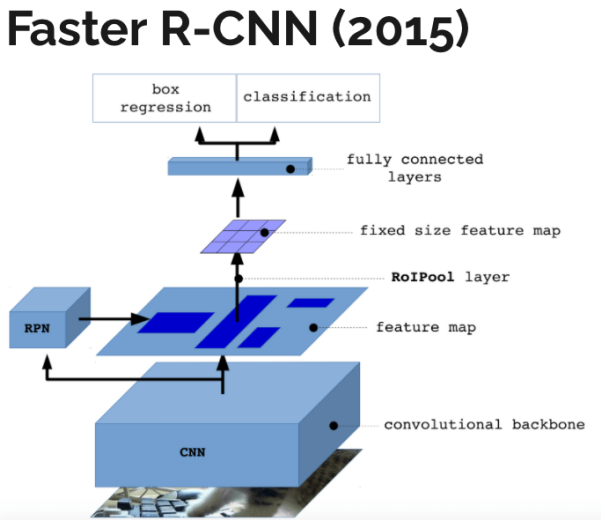


Improved Region Proposal with Faster R-CNN

Faster R-CNN addresses the bottleneck of region proposal generation in Fast R-CNN by introducing its own Region Proposal Network (RPN).

Advantages of RPN:

- RPN is faster and improves region proposal generation during training, replacing the slower selective search method.
- Faster R-CNN (VGG-16) achieves 0.2 seconds compared to 2.3 seconds in Fast R-CNN.
- Faster R-CNN (with RPN and VGG shared) achieves an mAP of 78.8% on COCO, VOC 2007, and VOC 2012 datasets, compared to 70% in Fast R-CNN on VOC 2007 test dataset.
- RPN contributes marginally to the improvement of mAP compared to selective search, further enhancing object detection performance.



Faster R-CNN: **Fast R-CNN Detector**

Fast R-CNN serves as an essential component within the Faster R-CNN architecture, playing a pivotal role in object detection within the region proposals suggested by the Region Proposal Network (RPN).

Operation of Fast R-CNN Detector:

- **Region of Interest (RoI) Pooling:** Transforms variable-sized region proposals into fixed-size feature maps through max pooling within equal-sized cells.
- **Feature Extraction:** Extracts meaningful features from RoI-pooled regions using the CNN backbone.
- **Fully Connected Layers:** Processes features through fully connected layers for object classification and bounding box regression tasks.
- **Object Classification:** Predicts class probabilities for each region proposal based on combined features.
- **Bounding Box Regression:** Refines bounding box positions and sizes for improved accuracy.
- **Multi-task Loss Function:** Combines classification and regression losses to optimize model parameters.
- **Post-Processing:** Refines final detection results using non-maximum suppression (NMS) to reduce redundancy and retain confident detections.

Exercise

Transitioning to Google-Colab for hands-on coding practice.

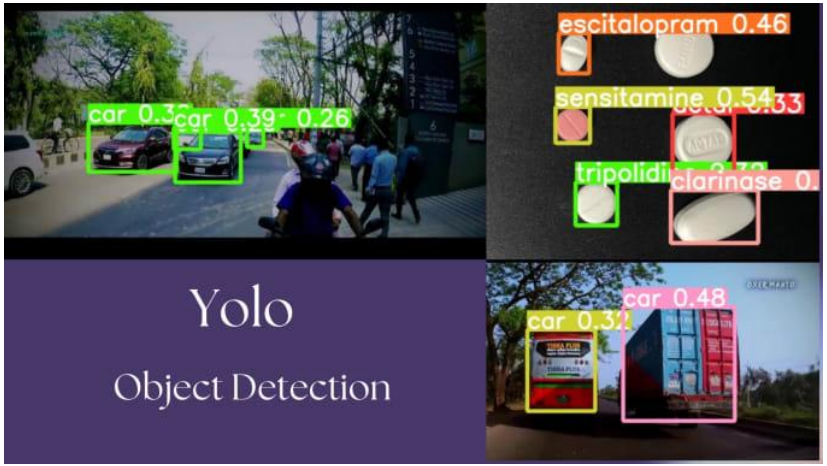
Notebook:

- TBD

YOLO – You Only Look Once

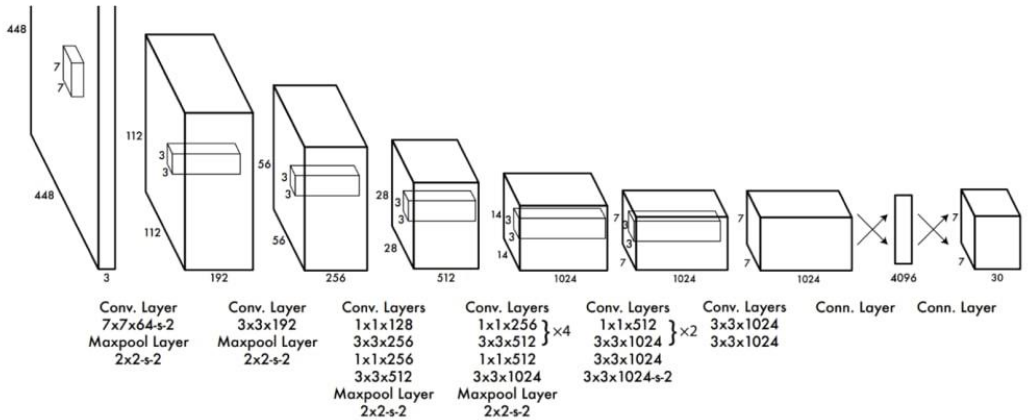
YOLO (You Only Look Once) is a popular object detection model known for its speed and accuracy. It was first introduced by Joseph Redmon et al. in 2016 and has since undergone several iterations, the latest being YOLO v8.

- All of the previous object detection algorithms use regions to localize the object within the image. The network does not look at the complete image. Instead, parts of the image which have high probabilities of containing the object.
- YOLO is an object detection algorithm much different from the region-based algorithms, it is a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.



YOLO Architecture

The YOLO algorithm takes an image as input and then uses a simple deep convolutional neural network to detect objects in the image. The architecture of the CNN model that forms the backbone of YOLO is shown below.



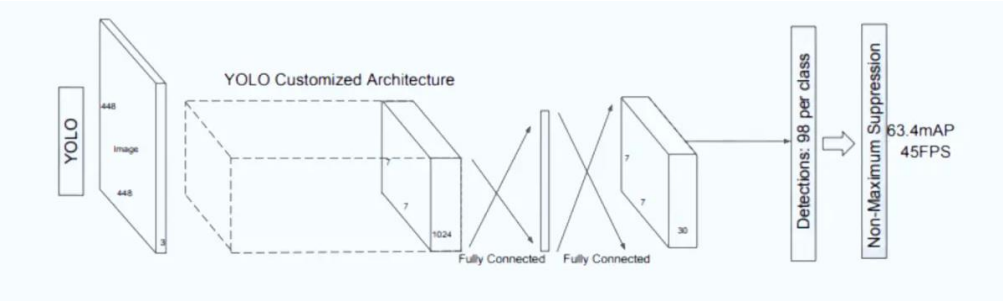
The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input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

YOLO Architecture from the original publication paper

YOLO Architecture

The architecture works as follows:

1. Resizes the input image into 448x448 before going through the convolutional network.
2. A 1x1 convolution is first applied to reduce the number of channels, which is then followed by a 3x3 convolution to generate a cuboidal output.
3. The activation function under the hood is ReLU, except for the final layer, which uses a linear activation function.
4. Some additional techniques, such as batch normalization and dropout, respectively regularize the model and prevent it from overfitting.

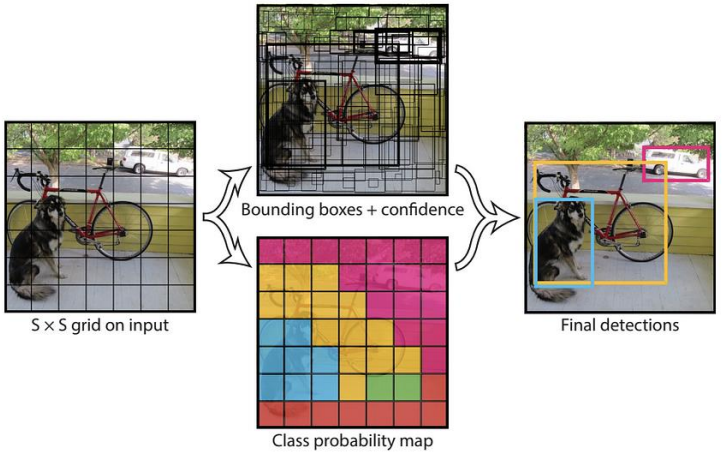


How does YOLO work?

The basic idea behind YOLO is to divide the input image into a grid of cells and, for each cell, predict the probability of the presence of an object and the bounding box coordinates of the object.

The process of YOLO can be broken down into several steps:

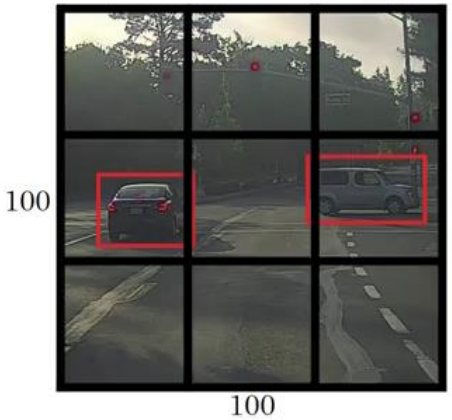
1. Input image is passed through a CNN to extract features from the image.
2. The features are then passed through a series of fully connected layers, which predict class probabilities and bounding box coordinates.
3. The image is divided into a grid of cells, and each cell is responsible for predicting a set of bounding boxes and class probabilities.
4. The output of the network is a set of bounding boxes and class probabilities for each cell.
5. The bounding boxes are then filtered using a post-processing algorithm called non-max suppression to remove overlapping boxes and choose the box with the highest probability.
6. The final output is a set of predicted bounding boxes and class labels for each object in the image.



YOLO's steps for Object Detection

1. YOLO first takes an input image:

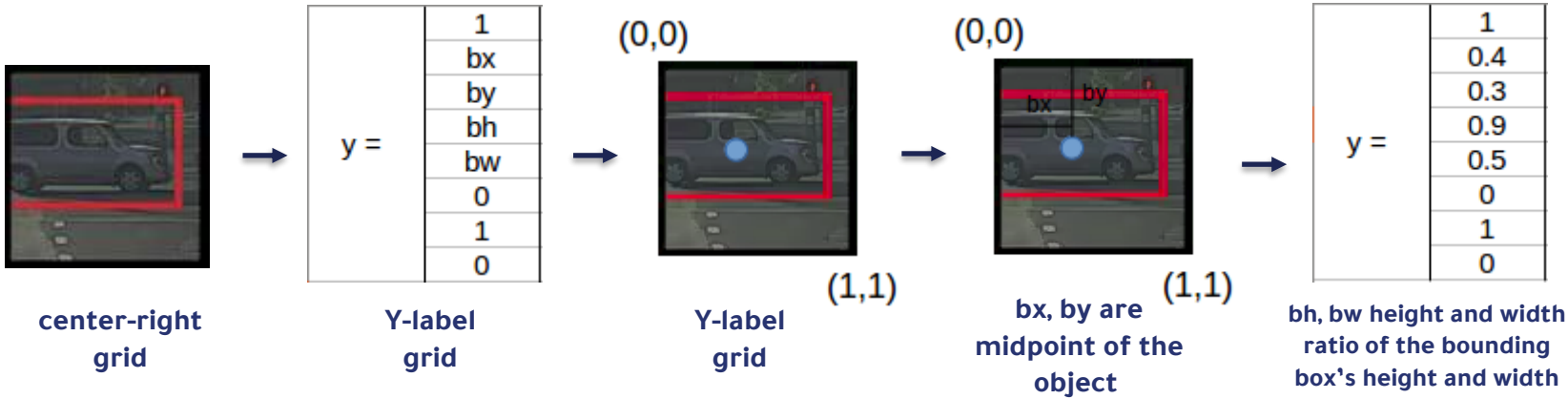
2. The framework then divides the input image into grids (say a 3 X 3 grid):



3. Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects (if any are found)

YOLO's steps for Object Detection *(cont'd)*

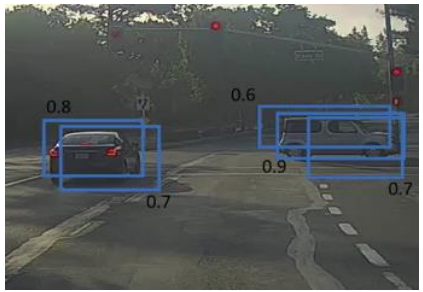
4. **To create the bounding boxes for YOLO model:**
   - Calculate bx and by relative to the grid cell containing the object's midpoint.
   - Compute bh as the ratio of the bounding box's height to the grid cell's height.
   - Determine bw as the ratio of the bounding box's width to the grid cell's width.
   - Ensure bx and by range between 0 and 1, while bh and bw can exceed 1 for larger bounding boxes.



YOLO's steps for Object Detection *(cont'd)*

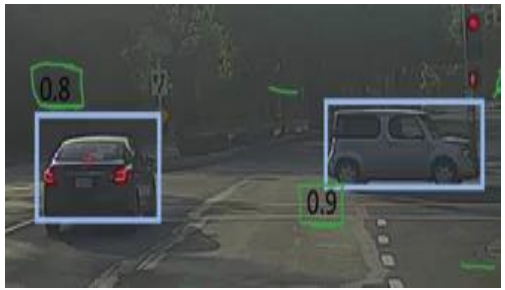
5. Evaluate the quality of predicted bounding boxes using Intersection over Union (IoU), which calculates the intersection area over the union of actual and predicted boxes.

6. Apply Non-Max Suppression to address duplicate detections, selecting the box with the highest probability and suppressing others with high IoU, iteratively refining predictions until a final set of bounding boxes is obtained.



Same object detected multiple times

Highest selected first then lower ones are discarded for each object

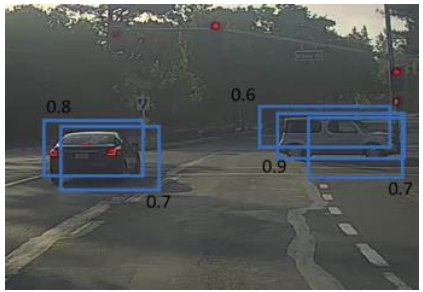


Highest probability boxes are selected for <u>each</u> object

YOLO's steps for Object Detection *(cont'd)*

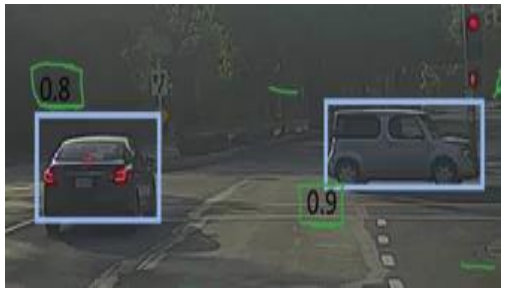
5. Evaluate the quality of predicted bounding boxes using Intersection over Union (IoU), which calculates the intersection area over the union of actual and predicted boxes.

6. Apply Non-Max Suppression to address duplicate detections, selecting the box with the highest probability and suppressing others with high IoU, iteratively refining predictions until a final set of bounding boxes is obtained.



Same object detected multiple times

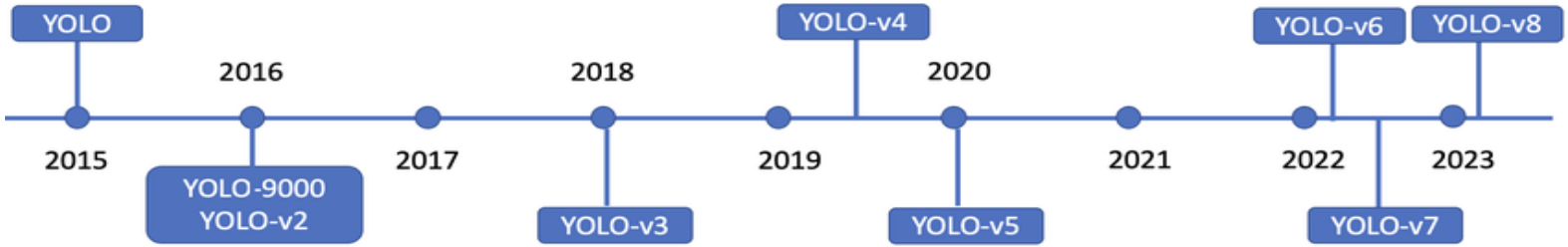
Highest selected first then lower ones are discarded for each object



Highest probability boxes are selected for <u>each</u> object

Evolution of YOLO in Object Detection Models

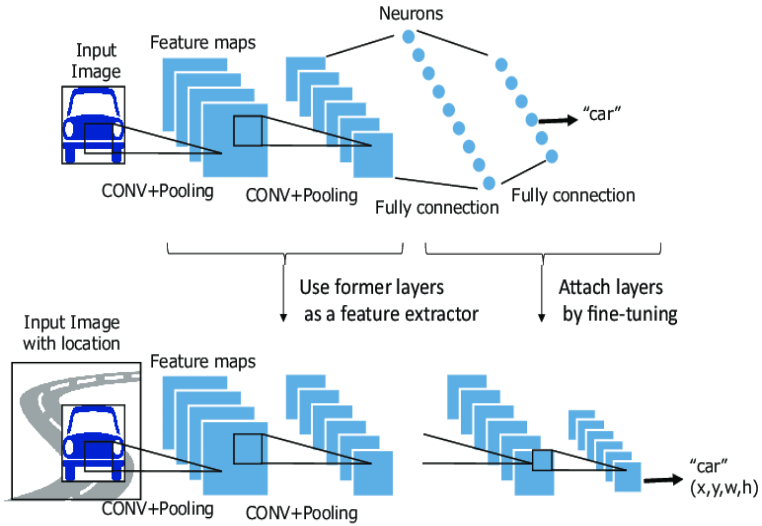
- ✓ **YOLOv1** introduced fast R-CNN capabilities, revolutionizing object detection with efficient recognition abilities but faced limitations in detecting smaller images within crowded scenes.
- ✓ **YOLOv2** refined the architecture with Darknet-19, batch normalization, and anchor boxes to address drawbacks and improve performance.
- ✓ **YOLOv3** continued enhancing performance with Darknet-53, logistic regression for bounding box prediction, and multiple predictions at different scales.
- ✓ **YOLOv4** optimized speed and accuracy with CSPDarknet53, Spatial Pyramid Pooling, PANet integration, and data augmentation techniques.
- ✓ **YOLOv5**, implemented in PyTorch, offered various model sizes and a focus layer for reduced complexity and improved speed.
- ✓ **YOLOv6** was tailored for industrial applications, featuring a hardware-friendly design and a more effective training strategy.



YOLO v7

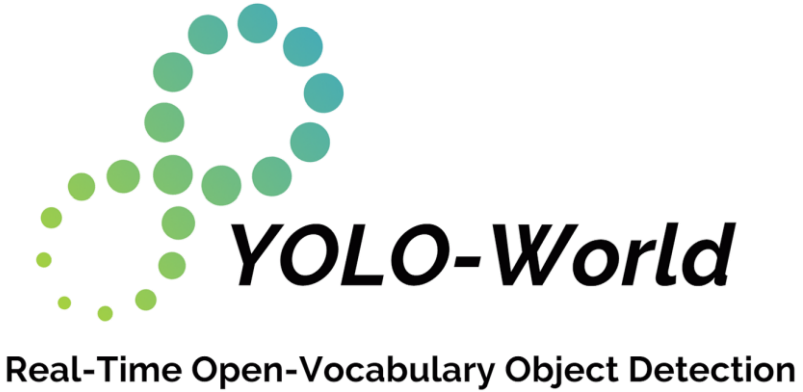
YOLO v7, on the most recent stable iterations of the YOLO algorithm. It boasts a number of enhancements compared to previous versions.

- ✓ A key enhancement is the implementation of anchor boxes. These anchor boxes, which come in various aspect ratios, are utilized to identify objects of various shapes.
- ✓ The use of nine anchor boxes in YOLO v7 enables it to detect a wider range of object shapes and sizes, leading to a decrease in false positives.
- ✓ In YOLO v7, a new loss function called "focal loss" is implemented to enhance performance.
- ✓ Unlike the standard cross-entropy loss function used in previous versions of YOLO, focal loss addresses the difficulty in detecting small objects by adjusting the weight of the loss on well-classified examples and placing more emphasis on challenging examples to detect.



YOLO Advantages

YOLO is widely used in real-world projects because of its accuracy and speed; its main powerful sides can be listed like the following:



- ✓ **Real-time object detection:** YOLO is able to detect objects in real-time, making it suitable for applications such as video surveillance or self-driving cars.
- ✓ **High accuracy:** YOLO achieves high accuracy by using a convolutional neural network (CNN) to predict both the class and location of objects in an image.
- ✓ **Single-shot detection:** YOLO can detect objects in an image with just one forward pass through the network, making it more efficient than other object detection methods that require multiple passes.
- ✓ **Good performance on small objects:** YOLO is able to detect small objects in an image because of its grid-based approach.
- ✓ **Ability to handle multiple scales:** YOLO uses anchor boxes, which allows the model to handle objects of different scales, thus allowing the model to detect objects of different sizes in the same image.

YOLO Limitations

Even though YOLO is a powerful object detection algorithm, it also has some limitations. Some of these limitations include:

- ✓ Limited to object detection: YOLO is primarily designed for object detection and may not perform as well on other tasks such as image segmentation or instance segmentation.
- ✓ Less accurate than some other methods: While YOLO is accurate, it may not be as accurate as two-shot object detection methods, such as RetinaNet or Mask R-CNN.
- ✓ Struggles with very small objects: YOLO's grid-based approach can make it difficult to detect tiny objects, especially if they are located close to other objects.
- ✓ No tracking capability: YOLO does not provide any tracking capability, so it may not be suitable for video surveillance applications that require tracking of objects over time...

Exercise

Transitioning to Google-Colab for hands-on coding practice.

Notebook:

- TBD

Thank You

SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority