Agenda

**Object Detection**
- Defining the object detection problem and a naive solution.

**YOLO Algorithm**
- YOLO algorithm steps
- Bounding boxes
- Measuring performance (UoI)
- Non-max suppression

**YOLO Implementations**
- Pretrained models with the COCO dataset.
- Custom trained models
Classification vs. Object Detection

Object Detection is the problem of locating and classifying objects in an image.

Classification
- Each image has one object
- Model predicts one label

Object Detection
- Each image may contain multiple objects
- Model classifies objects and identifies their location.
Naive Approach

1. Scan the image with a sliding window.

2. Feed the images into a classifier model to predict a label for that region.

- This approach is slow since it checks many windows that don’t contain anything -> Not good for real time uses.
- The Region-based Convolutional Neural Net (R-CNN) is an improved version that strategically selects regions that are likely to contain an object to run through the CNN.
YOLO Algorithm
YOLO "You Only Look Once"

- Instead of making predictions on many regions of an image, YOLO passes the entire image at once into a CNN that predicts the labels, bounding boxes, and confidence probabilities for objects in the image.
- YOLO runs much faster than region based algorithms quick because requires only a single pass through a CNN.
YOLO Steps

1. Divide the image into cells with an $S \times S$ grid.

2. Each cell predicts $B$ bounding boxes.

3. Return bounding boxes above confidence threshold.

A cell is responsible for detecting an object if the object’s bounding box falls within the cell. (Notice that each cell has 2 blue dots.)

All other bounding boxes have a confidence probability less than the threshold (say 0.90) so they are suppressed.

In practice, we would use large values ($S = 19$ and $B = 5$) to identify more objects.
How are bounding boxes encoded?

Let's use a simple example where there are 3x3 cells ($S=3$), each cell predicts 1 bounding box ($B=1$), and objects are either dog = 1 or human = 2. For each cell, the CNN predicts a vector $y$:

$$
\begin{align*}
( b_x, & b_y ) \\
\{ & b_w, b_h, b_x, b_y, b_w, b_h, c_1, c_2, p_c \} \\
\end{align*}
$$

- **$p_c$**: Probability the bounding box contains an object
- **$b_x$, $b_y$**: Coordinates of the bounding box’s center
- **$b_w$, $b_h$**: Width (height) of bounding box as a percent of the cell’s width or (height)
- **$c_1$, $c_2$**: Probability the cell contains an object that belongs to class 1 (or 2) given the cell contains an object

*There's a probability for each class so if there are 80 classes we would have $c_1, ..., c_{80}$*
What happens if we predict multiple bounding boxes per cell ($B > 1$)? We simply augment $y$.

Notice that $y$ has $5B+C$ elements ($C$ is the number of classes).
YOLO Overview

W × H × 3

W: Width of image in pixels
H: Height of image in pixels
3: Number of color channels in RGB

Series of convolutional and pooling layers.

S × S × (5B + C)

A tensor that specifies the bounding box locations and class probabilities.
Measuring Performance with UoI

- **Union over Intersection (UoI)** measures the overlap between two bounding boxes.
- During training, we calculate the UoI between a predicted bounding box and the ground truth (the prelabeled bounding box we aim to match).

![Diagram of UoI](https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)

**Union over Intersection** = \( \frac{\text{Area of Intersection}}{\text{Area of Union}} \)

- Poor
- Good
- Excellent
When predicting more than 2 bounding boxes per cell, sometimes the same object will be detected multiple times (overlapping boxes with the same label).

Non-max suppression solves multiple counting by removing the box with the lower confidence probability when the UoI between 2 boxes with the same label is above some threshold.

**Non-Max Suppression**

1. Identify the box with the highest confidence.
2. Calculate the UoI between the highest confidence box and each of the other boxes.
3. Suppress boxes with UoI above a selected threshold (usually 0.3).
Implementing YOLO
Pretrained Models

- Training a YOLO model requires images labeled with bounding boxes. These datasets may take time to label, so readily available prelabeled images are often used to train models.
- A common dataset for image classification/detection/segmentation is the COCO (Common Objects in Context), a database of images with 80 labelled classes.
- Popular pretrained YOLO models with COCO:
  - ImageAI (easy-to-use, lightweight YOLO implementation)
  - Darknet (trained by the author of YOLO)

Pineapples and cantaloupes are not in COCO so they are not recognized.
<table>
<thead>
<tr>
<th>Category</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>fire hydrant, elephant, skis, wine glass</td>
</tr>
<tr>
<td>bicycle</td>
<td>stop sign, bear, snowboard, cup</td>
</tr>
<tr>
<td>car</td>
<td>parking meter, zebra, sports ball, fork</td>
</tr>
<tr>
<td>motorbike</td>
<td>bench, giraffe, kite, knife, pizza</td>
</tr>
<tr>
<td>aeroplane</td>
<td>bird, backpack, baseball bat, spoon</td>
</tr>
<tr>
<td>bus</td>
<td>cat, umbrella, baseball glove, bowl</td>
</tr>
<tr>
<td>train</td>
<td>dog, handbag, skateboard, banana, chair</td>
</tr>
<tr>
<td>truck</td>
<td>horse, tie, surfboard, apple, sofa</td>
</tr>
<tr>
<td>boat</td>
<td>sheep, suitcase, tennis racket, sandwich</td>
</tr>
<tr>
<td>traffic light</td>
<td>cow, frisbee, bottle, orange, bed</td>
</tr>
</tbody>
</table>

Applications built with COCO trained models will only be able to identify these objects!
Custom Models

- If your use case only uses objects in COCO → you can use a pretrained model.
- Otherwise you will need to train your own YOLO model. This will require:

1. Finding images of the objects to recognize.
2. Label bounding boxes.
3. Train your YOLO model. There are 2 options:
   - a. Implement your own model using OpenCV, Tensorflow/Keras
   - b. Use ImageAI’s custom training methods.
References/Further Reading

- **YOLO**
  - [hackernoon.com/understanding-yolo-f5a74bbc7967](https://hackernoon.com/understanding-yolo-f5a74bbc7967)

- **R-CNN**

- **CNN**

- **Intersection Over Union**