Object Detection with YOLO
**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
Object Detection
Classification vs. Object Detection

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

Classification
- One object and label per image

Object Detection
- Multiple objects per image
- Determine objects' location

[Diagram showing examples of classification with a single object per image (e.g., Cat, Car, Dog) and object detection with multiple objects and bounding boxes (e.g., Cat, Dog, Car)]
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.

- Slow → Not good for real time uses
- Improved version: Region-based Convolutional Neural Net (R-CNN)
  - Strategically selects interesting regions to run through the classifier.

Label
Dog?
Person?
Nothing?
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 (much faster!)
- The CNN that predicts the **labels, bounding boxes, and confidence probabilities** for objects in the image.
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 $y$:

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

Example:

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_x$</td>
<td>$b_y$</td>
</tr>
<tr>
<td>$b_h$</td>
<td>$b_w$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

*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 \(y\).

The CNN will predict a \(y\) for each cell, so the size of the output tensor (multidimensional "matrix") should be: \(S \times S \times (5B+C)\)

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

Input

Convolutional Neural Net

Output

\[ W \times H \times 3 \]

- \( W \): Width of image in pixels
- \( H \): Height of image in pixels
- \( 3 \): Number of color channels in RGB

Series of convolutional and pooling layers.

\[ S \times S \times (5B+C) \]

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 Union over Intersection](https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)

### Performance Levels

- **Poor**: Small overlap between the predicted and ground truth bounding boxes.
- **Good**: Moderate overlap, indicating a reasonable match.
- **Excellent**: High overlap, indicating a very close match.

[Read more about Union over Intersection](https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)
**Double Counting Objects** *(Non-Max Suppression)*

- Sometimes the same object will be detected multiple times
- **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.

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)

<table>
<thead>
<tr>
<th>UoI</th>
<th>Dog: 0.95</th>
<th>Dog: 0.82</th>
<th>Dog: 0.41</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UoI: 0.62</td>
<td>UoI: 0.47</td>
<td></td>
</tr>
</tbody>
</table>
Implementing YOLO
Pretrained Models

- Images with bounding boxes prelabeled are often used to train object detection models
- **COCO (Common Objects in Context)** - a popular computer vision database of images with 80 labeled objects

Pineapples and cantaloupes are not in COCO so they are not recognized.
COCO Pretrained Labels

**Example labels:** person, car, motorbike, traffic light, dog, cat, sheep spoon, cup, sandwich, keyboard, chair, toaster, toothbrush, sports ball...

Image from [https://cocodataset.org/#home](https://cocodataset.org/#home)
## Custom Models

**Are the objects to detect in COCO?**

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pretrained Model</strong></td>
<td><strong>Train a Custom Model</strong></td>
</tr>
</tbody>
</table>
| 1. Download model. Some pretrained YOLO models:  
  - [ImageAI](https://imageai.readthedocs.io/en/latest/) (easy-to-use, lightweight YOLO implementation)  
  - [Darknet](https://darknet.faculty.ai/) (trained by the author of YOLO)  | 1. Finding images of the objects.  
2. Label bounding boxes.  
3. Train your YOLO model:  
  i. Implement your own model using OpenCV, Tensorflow/Keras  
  ii. Use a library (such as [ImageAI](https://imageai.readthedocs.io/en/latest/)'s custom training methods) |

---

*Berkeley SCET*
Applications built with COCO trained models will only be able to identify these objects!

<table>
<thead>
<tr>
<th>person</th>
<th>fire hydrant</th>
<th>elephant</th>
<th>skis</th>
<th>wine glass</th>
<th>broccoli</th>
<th>diningtable</th>
<th>toaster</th>
</tr>
</thead>
<tbody>
<tr>
<td>bicycle</td>
<td>stop sign</td>
<td>bear</td>
<td>snowboard</td>
<td>cup</td>
<td>carrot</td>
<td>toilet</td>
<td>sink</td>
</tr>
<tr>
<td>car</td>
<td>parking meter</td>
<td>zebra</td>
<td>sports ball</td>
<td>fork</td>
<td>hot dog</td>
<td>tvmonitor</td>
<td>refrigerator</td>
</tr>
<tr>
<td>motorbike</td>
<td>bench</td>
<td>giraffe</td>
<td>kite</td>
<td>knife</td>
<td>pizza</td>
<td>laptop</td>
<td>book</td>
</tr>
<tr>
<td>aeroplane</td>
<td>bird</td>
<td>backpack</td>
<td>baseball bat</td>
<td>spoon</td>
<td>donut</td>
<td>mouse</td>
<td>clock</td>
</tr>
<tr>
<td>bus</td>
<td>cat</td>
<td>umbrella</td>
<td>baseball glove</td>
<td>bowl</td>
<td>cake</td>
<td>remote</td>
<td>vase</td>
</tr>
<tr>
<td>train</td>
<td>dog</td>
<td>handbag</td>
<td>skateboard</td>
<td>banana</td>
<td>chair</td>
<td>keyboard</td>
<td>scissors</td>
</tr>
<tr>
<td>truck</td>
<td>horse</td>
<td>tie</td>
<td>surfboard</td>
<td>apple</td>
<td>sofa</td>
<td>cell phone</td>
<td>teddy bear</td>
</tr>
<tr>
<td>boat</td>
<td>sheep</td>
<td>suitcase</td>
<td>tennis racket</td>
<td>sandwich</td>
<td>potted plant</td>
<td>microwave</td>
<td>hair dryer</td>
</tr>
<tr>
<td>traffic light</td>
<td>cow</td>
<td>frisbee</td>
<td>bottle</td>
<td>orange</td>
<td>bed</td>
<td>oven</td>
<td>toothbrush</td>
</tr>
</tbody>
</table>
References/Further Reading

- **YOLO**
  - [Towards Data Science](https://towardsdatascience.com/you-only-look-once-yolo-implementing-yolo-in-less-than-30-lines-of-python-code-97fb9835bfd2)

- **R-CNN**
  - [Towards Data Science](https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e)

- **CNN**
  - [Coursera Lecture](https://www.coursera.org/lecture/convolutional-neural-networks/optional-region-proposals-aCYZv)

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

- **Intersection Over Union**
  - [PyImageSearch Blog](https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)