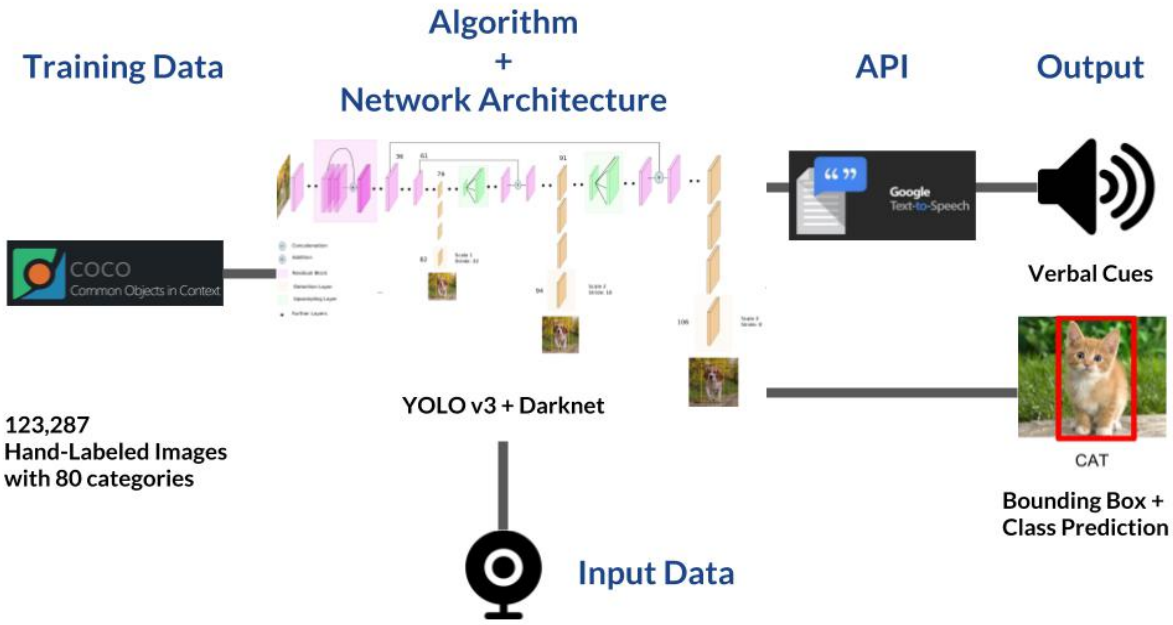
Object Detection With Real time Speech Control (YOLO+GTTS)

A very high-level overview :



1. **Training Data:** The model is trained with the Common Objects In Context (COCO) dataset. You can explore the images that they labeled in the link, it's pretty cool.

2. **Model:** The model here is the **You Only Look Once** (YOLO) algorithm that runs through a variation of an extremely complex Convolutional Neural Network architecture called the Darknet. Even though we are using a more enhanced and complex YOLO v3 model, I will explain the original YOLO algorithm. Also, the python **cv2** package has a method to setup Darknet from our configurations in the yolov3.cfg file.

I am more interested in getting something to work as soon as possible this time round so I will be using a pre-trained model. This means that COCO has already been trained on YOLO v3 by others and we have already obtained the weights stored in a 200+mb file.

If you are not sure what weights are, think of it as trying to find the Best Fit Line in Linear Regression. We need to find the right values of m and c in y=mx+c such that our line minimizes the error between all points. Now in our more complex prediction task, we have millions of Xs when we feed images into the complex network. These Xs will each have an m and these are the predicted weights stored in our yolov3.weights file. The **m**s have been constantly readjusted to minimize some loss function.

3. **Input Data:** We will be using our webcam to feed images at 30 frames-per-second to this trained model and we can set it to only process every other frame to speed things up.

4. **API:** The class prediction of the objects detected in every frame will be a string e.g. “cat”. We will also obtain the coordinates of the objects in the image and append the position “top”/“mid”/“bottom” & “left”/“center”/“right” to the class prediction “cat”. We can then send the text description to the Google Text-to-Speech API using the **gTTS** package.

5. **Output:** We will also obtain the coordinates of the bounding box of every object detected in our frames, overlay the boxes on the objects detected and return the stream of frames as a video playback. We will also schedule to get a voice feedback on the 1st frame of each second (instead of 30 fps) e.g. “bottom left cat” — meaning a cat was detected on the bottom-left of my camera view.

Understanding the YOLO algorithm:

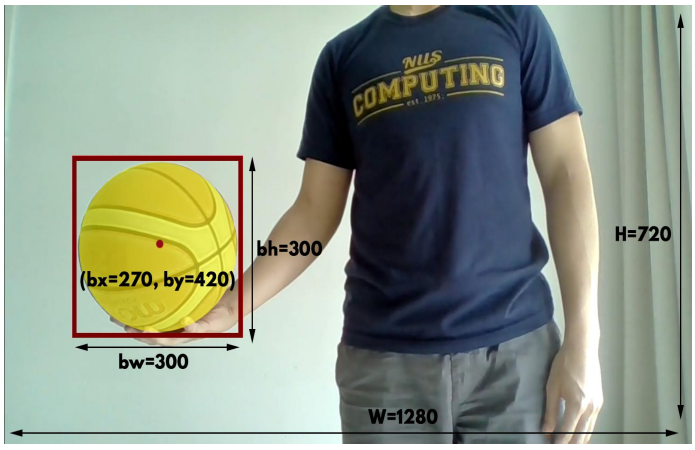
Previously, classification-based models were used to detect objects using localization, region-based classification or things such as the sliding window. Only high scoring regions of the image are considered as a detection and they could be very time-consuming

Instead, YOLO is **regression-based**. We’re predicting classes and bounding boxes for the whole image quickly in **one run** of the algorithm **(just one look of the image’s pixels)**, so that the predictions are informed by the global context in the image.

Training Time

```
29   suitcase
30   frisbee
31   skis
32   snowboard
33   sports ball
34   kite
35   baseball bat
36   baseball glove
37   skateboard
```

C represents the class index of the object we are trying to label. A sports ball would mean C=33. The training has already been done on COCO.



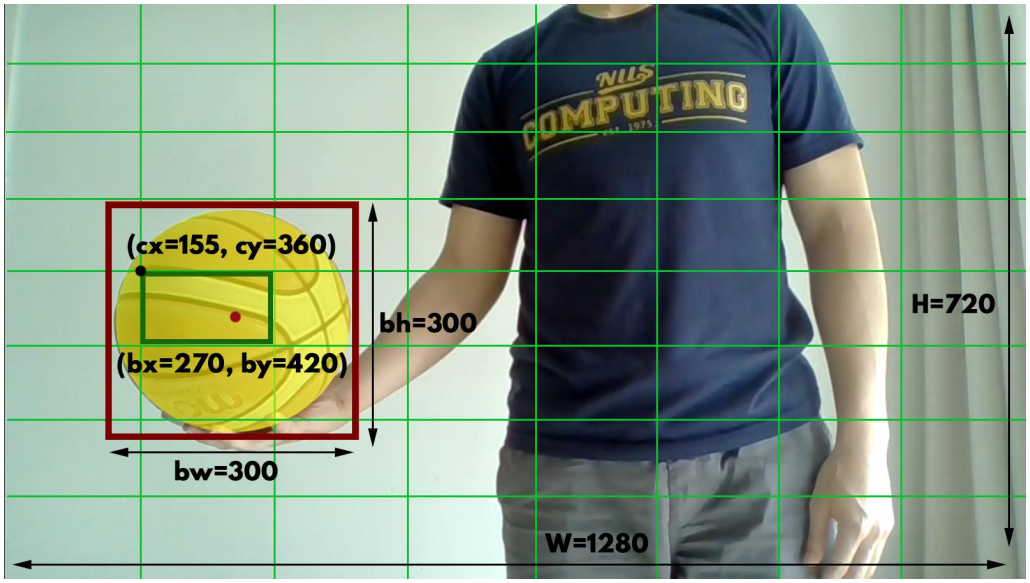
During training time, we have to manually label the following 5 values for each object in the image in this format `c bx by bw bh`. We will also normalize each of the 4 b values to lie between 0–1 by representing them as a proportion of W & H (1280 x 720 px).

Suppose we have 2 objects labeled in the frame — The sports ball and myself. This is represented by tensors: a general form of a vector, which will be fed into the model during training.

```
[33, 0.21, 0.58, 0.23, 0.42] - sports ball
[1, 0.67, 0.5, 0.5, 0.9] - myself
```

Prediction / Detection Time

Now we are feeding 1280 x 720 frames from our camera into YOLO at Prediction time. YOLO will automatically resize it to 416 x 234 and fit it into a popular standard-sized 416 x 416 network by padding the excess with 0s. YOLO divides each image into S x S cells each with a size of 32 x 32 (reduction factor=32). This creates 416/32 = 13 x 13 cells.



There are 5 values in a bounding box — (bx, by, bw, bh, BC)

If the center of an object (red dot) falls into a grid cell, only that grid cell (dark green cell) is responsible for detecting that object. Each bounding box has 5 values. The first 4 values **bx, by, bw, bh** represent the position of the box.

$$\frac{bx-cx}{cx}, \frac{by-cy}{cy}, \frac{bw}{W}, \frac{bh}{H}$$

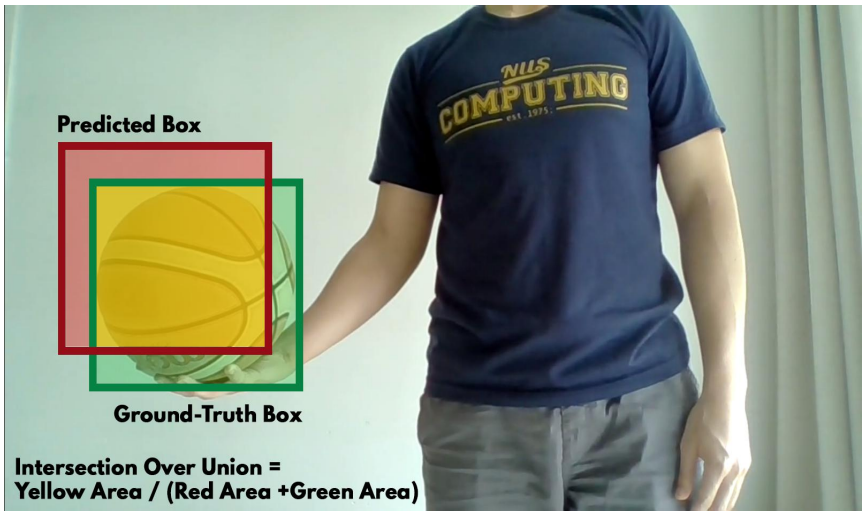
$$\frac{270-155}{155}, \frac{420-360}{360}, \frac{300}{1280}, \frac{300}{720}$$

1) Normalized using the coordinates of the top-left corner of the cell which contains the object's center. 2) using the dimensions of the entire image.

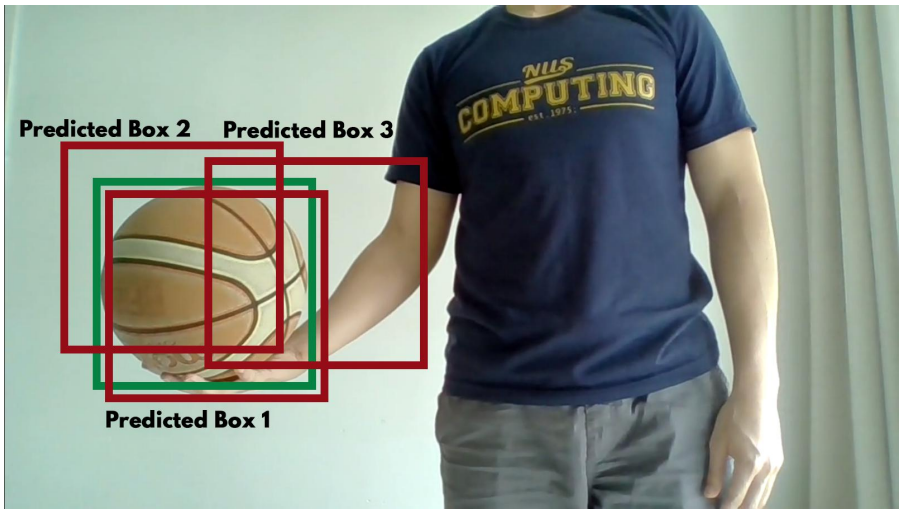
The 5th value is **BC**: the box confidence score.

BC = Pr(Object existing in box) * **IOU (**Intersection Over Union).

This measures how likely the box contains an object of *any class* and how accurate it is in predicting. BC=0 if no object exists in that box and we want BC=1 in predicting the ground truth.



There are B bounding boxes predicted in each grid cell



YOLO v3 makes B=3 bounding boxes in each cell to predict that one object in the cell.

There are also C conditional class probabilities in each grid cell

There are 80 conditional class probabilities — Pr(Class i | Object) per cell when we use COCO. It is the probability that the predicted object is of Class i given that there is an object in the cell.

```
1 person 0.01
2 bicycle 0.004
.
.
33 sports ball 0.9
.
.
80 toothbrush 0.02
```

In our example above, Class 33 has the highest probability and it will be used as our prediction of the object to be a sports ball.

To sum up the above

There are S x S cells and in each of these cells there are 2 things: 1) B bounding boxes each with 5 values (bx, by, bw, bh, BC), 2) C conditional class probabilities. The predictions are encoded as a S x S x (5 * B + C) tensor.

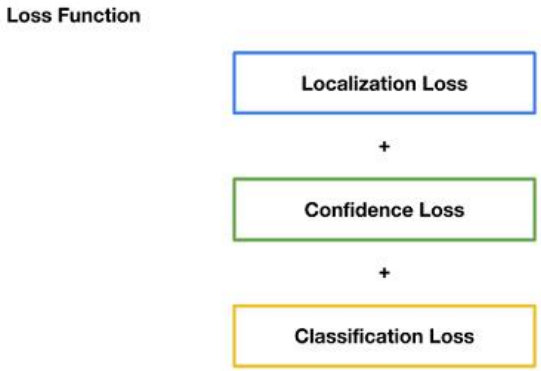
Non-Max Suppression

There are actually B duplicate detections for the grid cell that is the only one used to predict the object (dark green box). NMS suppresses detections with low box confidence scores: BC, so that we don't end up predicting 3 sports balls when there is only just one. An implementation of NMS could be:

1. Start with the bounding box that has the highest BC.
2. Remove any remaining bounding boxes that overlap it more than the given threshold amount = 0.5.
3. Go to step 1 until there are no more bounding boxes left.

Loss Function

While we already have our predictions, we want to understand how the weights were being adjusted such that our loss function for this model was minimized during Training Time. The function looks complex but it is actually very intuitive when broken down.



1. Difference between the ground-truth box vs the predicted boundary box.

2. Difference between 100% confidence that an object is in the box vs the box confidence.

3. Difference between C actual class probabilities (0, …, 1, …, 0) vs C predicted class probabilities (0.01, …, 0.8, …, 0.02)

Loss Function

Localization

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

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

Confidence

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Basically uses sum of squared-differences for each component. Most symbols are pretty self-explanatory.

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

The parameter λcoord is usually =5 to increase the weight of the localization loss to give it more importance.

The parameter λnoobj is usually =0.5 to decrease the weight of the confidence loss since boxes that do not contain objects have confidence scores BC=0. This makes the model more stable and easier to converge.

Voice Feedback

We can use bx & by relative to W & H to determine the position of the objects detected and send it as a text string to gTTS with this simple command.

```
tts = gTTS("mid left sports ball, lang='en')tts.save('tts.mp3')
```

I also used **pydub** and **ffmpeg** to manipulate the audio files generated.

Demo

I am unable to return the frames in real-time because it will make the video playback seem very jerky when it processes every 30th frame. I also explored multi-threading, which in theory should create 1 process for processing every 30th frame and another process for the video playback.

However, I am only able to produce the verbal description of objects detected in real-time on my webcam which is more important since the blind can't see bounding boxes anyway. For the video below, I recorded myself before passing it to create the bounding boxes and generate the verbal responses.