# Soccer ball detection using Learning

Rafael Oliveira Nº76525

Aveiro's University

**Abstract.** The main objective of this report is describing the process of training a neural network to detect certain objects crucial to the RoboCup games. YOLO algorithm version 3 is going to be used, as it has the best performance in real-time scenarios, of all object detection algorithms. A dataset for RoboCup soccer balls was provided but it needed to be changed to meet the objectives.

**Keywords:** Robot · Sensor fusion

## 1 Objectives

The initial objective was to detect arbitrary balls using a learning algorithm from a data set of images provided here.[1] A model was trained with this dataset of images but the results were really bad, so it was decided to make a dataset from scratch.

## 2 YOLO

### 2.1 Darknet

Darknet is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.


**Fig. 1.** Darknet logo.

### 2.2 How it works

YOLO(You Only Look Once) is a state-of-the-art, real-time object detection system. This algorithm rethinks the standard for object detection. The neural network is given the full image and gets object detection in a single past, i.e., the neural network only looks once at the image to perform the full detection pipeline thus the name.

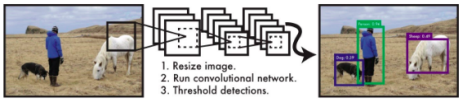


**Fig. 2.** How YOLO feeds the neural network.

YOLO has a different parameterization when comparing to other object detection algorithms.

Given an image, YOLO divides it into a grid and each cell is responsible for predicting some number of bounding boxes and some confidence values for each of the boxes, i.e., the probability that the box contains an object. So cells with boxes with high confidence will have objects and the ones with low confidence will not have any objects.

When all of the predictions are visualized together, the image will have a lot of boxes ranked by their confidence values.

**Fig. 3.** Bounding boxes ranked by confidence.

After this process, the algorithm knows where the objects are in the image but not what they are.

Each cell will predict the class probabilities and then multiply it by the confidence values to compute the bounding boxes weighted by their probabilities for containing that object.

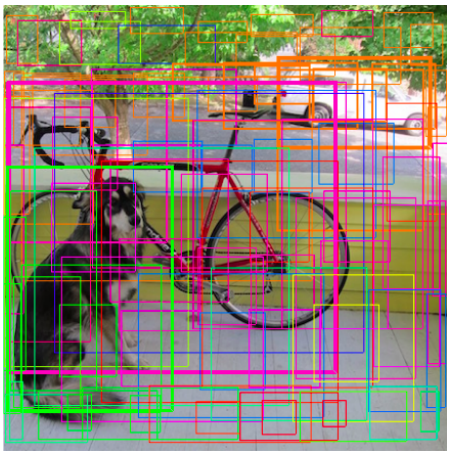

**Fig. 4.** Bounding boxes weighted by their probabilities for containing that object.

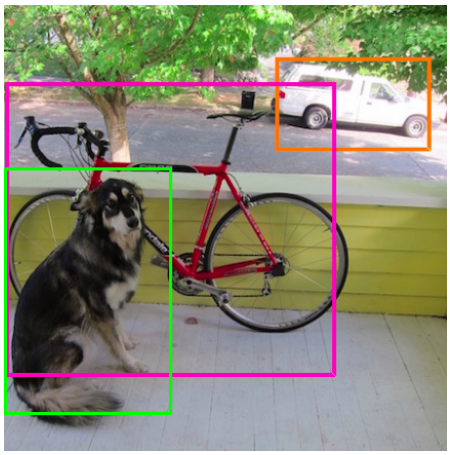

**Fig. 5.** Image after the threshold is applied.

As can be seen in **Fig.3**, there are a lot of boxes but the majority of these have very low confidence values. After applying a threshold to the predictions, the final detection image is done.

Because the image only passes through the network once the process is very fast comparing to other ways of object detection.[4]

### 2.3 YOLO9000 improvements

The first version of YOLO was really fast but not that good in terms of accuracy compared to other object detection algorithms.

The first good added feature to the second version was fine-tuning on ImageNet dataset, where the network is resized half the way through the training process.

The first version predict the coordinates directly for the objects. Other detection algorithms use anchor boxes which are pre-computed boxes to predict bounding boxes for the objects. It is easier to adjust these pre-computed boxes than to predict coordinates directly. Even though anchor boxes are a great improvement YOLO9000 has a better way to deal with this problem.

YOLO9000 looks at the training data and sees what the bounding boxes look like, then run k-means clustering in these boxes. K-means clustering uses the boxes that typically have objects and after running it, a cluster of boxes is created, called dimension clusters. These dimension clusters tell us how close is

the cluster to overlap the object in an image without modification, i.e., these process is used to place the boxes in the best staring place possible.

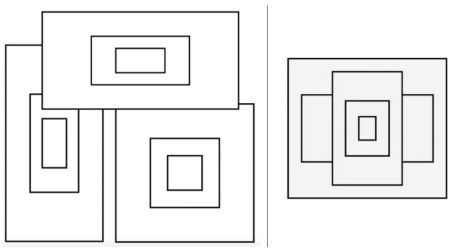

**Fig. 6.** Difference between anchor boxes(on the left) and dimension clusters(on the right).

The last feature added was multi-scale training. In the first version, the network was trained with a single aspect ratio, 48x48, resizing all the images to be that size.

In YOLO9000 the network is resized, randomly, through the training process to different scales. During training run images from 320x320 to 680x680, down sampling the images by a factor of 32. With this, images can be resized without affecting the network structure and without changing the weights.

This method can be seen as a way to do data augmentation and it avoids over fitting. Does not influence performance in a significant way because the neural network is the same the only thing changing is the input dimension. Because all the layers are convolution, the resize can be done without changing parameters and with minimal effect.[5]

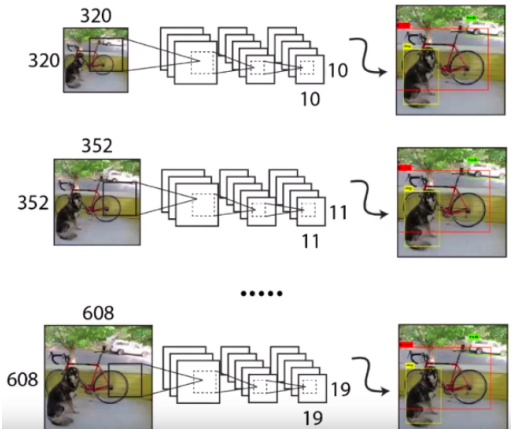


**Fig. 7.** How YOLO9000 performs multi-scaling training.

### 2.4 YOLOv3 improvements

YOLO9000 was great, but one of the bad sides of this version was the lack of detection of smaller objects. Thankfully YOLOv3 addressed this issue.

The previous version used multi-scale training to improve accuracy and YOLOv3 used multi-scale detection to achieve the same goal. These scales are given by downsampling the dimensions of the input image by 32, 16 and 8 respectively.

Detections at different layers helps address the issue of detecting small objects, a frequent complaint with YOLO9000.

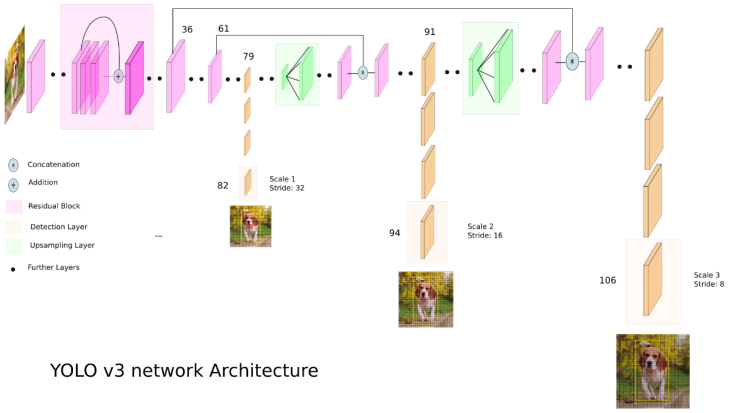


**Fig. 8.** How YOLOv3 performs multi-scaling detection.

Earlier, YOLO used to softmax the class scores and take the class with the maximum score to be the class of the object contained in the bounding box.

Softmaxing classes rest on the assumption that classes are mutually exclusive, i.e., if an object belongs to one class, then it cannot belong to the other. This works fine in COCO dataset, however when we have classes like Person and Women in a dataset, then the above assumption fails.

This is the reason why YOLOv3 uses logistic regression and a threshold to predict multiple labels for an object. Classes with scores higher than this threshold are assigned to the box.[6]

## 3 Training with the provided dataset

### 3.1 Extracting images from a hdf file

To train this model to detect a custom object, a dataset and its labels were provided inside an HDF5 file. HDF5 is a unique technology suite that makes possible the management of extremely large and complex data collections.

The description of how the data was structured was provided, helping with the extraction of the images and its labels.

**Listing 1.1.** Every image has a label that consists of 4 values:

```
* Is it a ball? (0 or 1)
* If it is a ball, x coordinate of the center
* If it is a ball, y coordinate of the center
* If it is a ball, radius of the seen ball
```

A package named h5py was used to extract the data from the file. This data was separated into two main types, the images and the labels, and each one into two sub-types, if the data represented a ball or not.

After this, NumPy and OpenCV were used to create a png file of the image and the labels were written in a txt file. Darknet needs that an association between the labels and the images, so it accomplishes this by taking an image and search for its name but with a txt extension, to search for its labels.

**Listing 1.2.** Code describing the conversion from HDF5 data to an image file

```
for i in img:
    data = np.array(i[:,:,:])
    file = data_type + '/ball'+str(c)+'.jpg'
    cv2.imwrite(file, data)
    c+=1
```

**Listing 1.3.** Code describing the conversion from HDF5 data to YOLO label format

```
for i in range(len(img)):
    f= open(data_type + '_labels/ball' + str(i) + '.txt',"w+")
    f.write("0 " +str(img[i][1]/32)+ " " +str(img[i][2]/32)+ " "\
    +str(2*img[i][3]/32)+ " " +str(2*img[i][3]/32))
    f.close()
```

YOLO only accepts a specific syntax of object labels. It was to be in the following format:

**Listing 1.4.** Labels format

```
<objectID> <center position in x> <center position in y>
<width of the box> <height of the box>
```

All these values of the position must not be in pixels but in percentage relative to the image size, i.e., if the center position in x and y is 0.5 it means that the center is in the middle of the image.

**Listing 1.5.** Labels format example

```
0 0.2331 0.214 0.412 0.1241
```

After converting all data to an image format and creating its labels, a program[3] was used to check if the labels were in the right place.

### 3.2 Train in Darknet

After moving all the images and labels to a folder inside Darknet, it was time to set up the model and training data to start training.

The model used was a YOLOv3 cfg. This model needed to be modified in some ways. The batch size and subdivisions were change to 64 and 16 respectively. This manly, change how much data will be processed at once, loading 64 images for iteration and subdivisions split the batch into 64/16=4 images per "minibatch". The subdivisions value can be lower if the GPU is powerful enough.

The *max_batches* value was changed to 2000 because it needed to be classes*2000 and the model was only training one class, *robot_cup_ball*. The *classes* value was changed from 80, 80 classes in the COCO dataset, to 1, and consequently, the *filters* values were changed to 18 because of filters=(classes + 5)x3.

After configuring the cfg file the classes names were added to a file obj.names, i.e., write *robot_cup_ball* in the file. Only one more file was created, which was obj.data were the following was written:

**Listing 1.6.** obj.data file

```
classes= 1
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = backup/
```

- **classes** value describes how many classes the model will train.
- **train.txt** has all the names of the images from the dataset.
- **test.txt** is used to test the performance of the model.
- **obj.names** describes the classes names.
- **backup** folder saves the weights from training every 100 iterations.

After this configuration, the model was trained in the Darknet framework.

### 3.3 Results

After 30 iterations or so, the framework stopped training and gave the weights for testing. Using these weights, the results were not good at all. Using a video from youtube of a RoboCup game, none of the balls were detected.

After looking at the dataset, it was realized that all images were black and white and all 32x32. Their size should not be a problem because, as said earlier, YOLOv3 trains and detects with multi-scaling, changing the image size during these processes. So the problem was probably in the colors of the dataset because the color of the image is very important for a convolutional network trying to train a model.

Realizing this the trained model was tested in a black and white video. This time the model detected the object, but in the crowd and not the balls so another

solution came up. Applying a sharpness filter to bigger images, because in their 32x32 size it did not make that big of a difference, and training the model again.

After doing the process all over again nothing came along. The balls were not detected even with these changes, so all hope was lost with this dataset.

## 4 Training with a custom dataset

After no balls detected by two trained models with the same dataset but different ways to handle the images, it was decided that a new dataset would be created.

Datasets need to express a lot of behaviors that the object have, for example in the case of this report, images of the balls moving, stationary, far away and close must be provided so the model recognizes all the different "states" of the object.

### 4.1 Custom dataset

Having this in mind, a set of videos were downloaded from the same domain that had the dataset and a dataset was created from there. Using OpenCV, images were extracted every 15 frames through a set of 10 videos, with different durations. The important part to take note is that not all the videos have the same camera angle, which means that the images will have balls with different sizes, i.e., far away and close.

In the end, 7700 images were extracted and then, the boring work. Using the program mentioned before[3], the one utilized to check if the labels were in the right place, every ball in every image was labeled to then train the model.

This program was really helpful because after selecting where the ball is in the image, it creates a file with the label format discussed earlier.

### 4.2 Results

YOLOv3 and YOLOv3 tiny models were trained with the same custom dataset to compare the two. Because the robots from RocoCup do not have much processing power, maybe the YOLOv3 tiny model would be more fitting for this case.

In the end, the two models detect balls in some angles but have difficulty if the ball is too far away thus being too small in the video. This scenario could be a limitation of the YOLO algorithm or the training needing more images.

Talking again about the model trained with the provided data, it was speculated that this could work with the "robot eyes", i.e., if a video of what the robot sees was provided it could probably detect the balls. This was concluded because, at first site, there is nothing wrong with the images provided, they simply do not detect anything with videos of angles showing the game.

## 5 Conclusion

After doing this project, it was easier to understand why convolution networks are so important for object detection and why YOLO is the state of the art on this topic.

In the end, the goal, training a model in YOLO with the dataset provided, was accomplished, even though the detection was not there.

I think the training with a custom dataset was more to prove that the training works and not to use in the robots themselves. I say this because, as speculated above, the robots will probably "see" the environment more like the dataset provided than the custom dataset.

## References

1. Server at sibylle.informatik.uni-bremen.de
   https://sibylle.informatik.uni-bremen.de/public/datasets/b-alls-2019/
2. Darknet project page
   https://pjreddie.com/darknet/
3. Program used to make and check labels
   https://github.com/ivangrov/YOLOv3-Series/tree/master/%5Bpart%204%5DOpenLabelling
4. YOLO paper
   https://pjreddie.com/media/files/papers/yolo$_1$.*pdf*
5. YOLO9000 paper
   https://pjreddie.com/media/files/papers/YOLO9000.pdf
6. YOLOv3 paper
   https://pjreddie.com/media/files/papers/YOLOv3.pdf