

# Training YOLO Model for Number Detection on Jerseys of Football Players

Ruiling Qian

April 29, 2020

*Index terms*— YOLOv3, Number Detector

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Background . . . . .	3
1.2	Aim . . . . .	4
1.3	Methodology . . . . .	4
<b>2</b>	<b>Literature Review</b>	<b>5</b>
<b>3</b>	<b>Data Preparation and Processing</b>	<b>6</b>
<b>4</b>	<b>Setting up environments</b>	<b>8</b>
<b>5</b>	<b>Training Process</b>	<b>9</b>
5.1	Labelling Data . . . . .	9
5.2	Integrating Data . . . . .	9
5.3	Modifying configuration . . . . .	10
5.4	Training . . . . .	11
5.5	Augmentation . . . . .	12
<b>6</b>	<b>Detection on images</b>	<b>13</b>

# Chapter 1

## Introduction

### 1.1 Background

Object detection is a highly concerned area of computer science since it can be applied on many occupations works, such as face detector, automatic driving. (Paul et al., 2001) To train a object detector, there are many built neural networks models for us to use, such as SSD321, R-FCN etc., YOLOv3 is one of these object detection system, it has better performance on the COCO dataset than other common detectors. (Redmon et al.,2018)

Football is one of the most popular sports around the world, and there are research about tracking and identification of player in the football video been done, they are motivated by practical usage for the team, such as match analysis, this can be achieved by using object detection on the jersey number of players. (Sebastian et al., 2016)

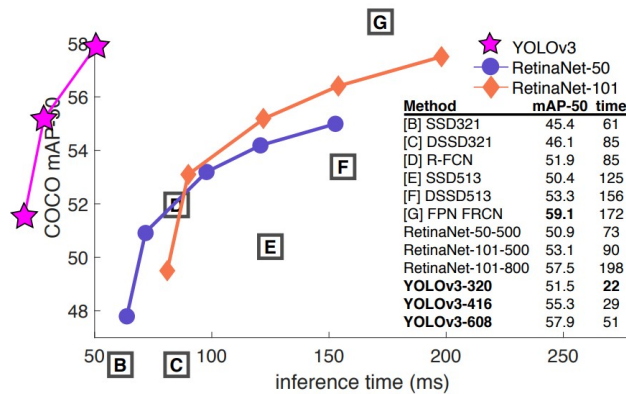


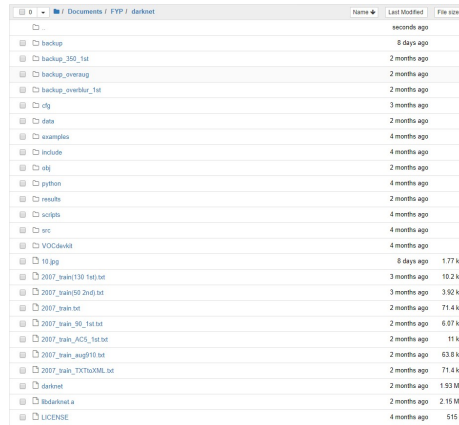
Figure 1.1: YOLO compared with other model on COCO dataset

## 1.2 Aim

The main aim of this project is to train an number detector model from scratch using provided football match video with YOLOv3 object detection package, and comparing models built using different dataset and augmentation methods.

## 1.3 Methodology

The main programming language used in this project is python, and OpenCV module is used frequently to process the dataset. Darknet framework is used to train the model, it is an open source neural network framework written in C and CUDA, it includes makefiles and configuration files for general detection and training with different format, VOC 2007 format is used in this project.



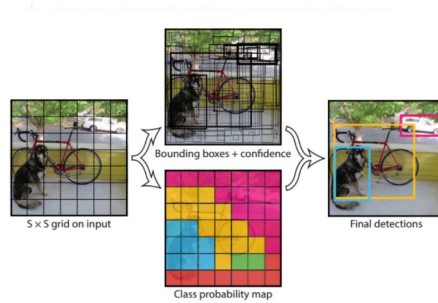
Name	Last Modified	File size
.	seconds ago	
backup	8 days ago	
backup_350.txt	2 months ago	
backup_overaug	2 months ago	
backup_overblur.txt	2 months ago	
cfg	3 months ago	
data	2 months ago	
examples	4 months ago	
include	4 months ago	
obj	2 months ago	
python	4 months ago	
results	2 months ago	
scripts	4 months ago	
src	4 months ago	
VOCdevkit	4 months ago	
10.jpg	8 days ago	1.77 kB
2007_train(150.txt).set	3 months ago	10.2 kB
2007_train(50.txt).set	3 months ago	3.92 kB
2007_train.txt	2 months ago	71.4 kB
2007_train_50.txt.set	2 months ago	6.07 kB
2007_train_ACS.txt.set	2 months ago	11 kB
2007_train_aug910.txt.set	2 months ago	63.8 kB
2007_train_TxtToXML.txt.set	2 months ago	71.4 kB
darknet	2 months ago	1.93 MB
Darknet.a	2 months ago	2.15 MB
LICENSE	4 months ago	515 B

Figure 1.2: darknet folder

## Chapter 2

# Literature Review

In YOLO framework, object detection is reframed as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Firstly, an image is resized and divided to multiple grids, then YOLO predicts multiple bounding boxes on those grids, which includes 4 coordinates and confidence for each bounding boxes, the confidence represents the IOU between the predicted box and any ground truth box. (Manish, 2017)



**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an  $S \times S$  grid and for each grid cell predicts  $B$  bounding boxes, confidence for those boxes, and  $C$  class probabilities. These predictions are encoded as an  $S \times S \times (B * 5 + C)$  tensor.

Figure 2.1: How YOLO works

## Chapter 3

# Data Preparation and Processing

At the beginning of this project, two match videos from Loughborough University database and players position in each frames are given for training, the first step is to clip out all the players from each frames to generate the dataset which contains only players with visible number on their jersey.



Figure 3.1: example frame

Using files like Fig 3.2 showed, a scripts is written for going thorough all the frames of the video and clip out the players showed in the files. With the clipped players image, 2 datasets are selected, one contains clear image with visible jersey number (1st class data), the other one contains blurry image with jersey number that is not quite visible by human eyes(2nd class data), both are used later for training and evaluating.

```

1 0,0,1039,730,20,56,1,1,1,1
2 0,1,1534,806,27,94,1,1,1,1
3 0,2,1716,751,23,79,1,1,1,1
4 0,3,1902,705,20,72,1,1,1,1
5 0,4,2019,677,13,59,1,1,1,1
6 0,5,2121,746,27,87,1,1,1,1
7 0,6,2148,726,34,76,1,1,1,1
8 0,7,2345,664,15,51,1,1,1,1
9 0,8,2527,915,32,149,1,1,1,1
10 0,9,2520,695,25,67,1,1,1,1
11 0,10,2576,800,40,105,1,1,1,1
12 0,11,4083,767,23,60,1,1,1,1
13 0,12,3268,733,24,77,1,1,1,1
14 0,13,3307,758,30,83,1,1,1,1
15 0,14,2925,679,21,62,1,1,1,1
16 0,15,3072,943,78,155,1,1,1,1
17 0,16,2892,751,36,89,1,1,1,1
18 0,17,2780,709,19,73,1,1,1,1
19 0,18,2738,1051,81,203,1,1,1,1
20 0,19,2528,647,20,46,1,1,1,1
21 0,20,2556,700,22,60,1,1,1,1
22 0,21,2520,733,25,81,1,1,1,1
23 1,0,1039,730,20,56,1,1,1,1
24 1,1,1534,806,27,94,1,1,1,1
25 1,2,1716,751,23,79,1,1,1,1
26 1,3,1902,705,20,72,1,1,1,1
27 1,4,2019,677,13,59,1,1,1,1
28 1,5,2121,746,27,87,1,1,1,1
29 1,6,2148,726,34,76,1,1,1,1
30 1,7,2345,664,15,51,1,1,1,1
31 1,8,2527,915,32,149,1,1,1,1
32 1,9,2520,695,25,67,1,1,1,1
33 1,10,2576,800,40,105,1,1,1,1
34 1,11,4083,767,23,60,1,1,1,1
35 1,12,3268,733,24,77,1,1,1,1
36 1,13,3307,758,30,83,1,1,1,1
37 1,14,2925,679,21,62,1,1,1,1
38 1,15,3072,943,78,155,1,1,1,1
39 1,16,2892,751,36,89,1,1,1,1
40 1,17,2780,709,19,73,1,1,1,1
41 1,18,2738,1051,81,203,1,1,1,1
42 1,19,2528,647,20,46,1,1,1,1
43 1,20,2556,700,22,60,1,1,1,1

```

Figure 3.2: players position in each frame

Figure 3.3: \*

each line represent a player and their position, as in: frame, id , x, y, w, h

```

1 import cv2
2 import os
3
4 file = open("acl_30fps.txt", "r")
5 lines = file.readlines()
6 file.close()
7
8 frame_detector_count = -1
9 count = 0
10
11 for line in lines:
12     count = count+1
13
14     parts = line.split(",") # split line into parts
15
16     frame=int(parts[0])
17
18     img = cv2.imread("acl_player_30fps/%d.jpg"%(frame))
19
20
21 if(frame != frame_detector_count and not
os.path.exists("/home/lunet/corq/Documents/Data/to_be_copied/acl/cropped321/frame%d"%(
frame))):
22     frame_detector_count = frame
23     os.mkdir("/cropped321/frame%d"%(frame))
24
25     id=int(parts[1])
26     x=int(parts[2])
27     y=int(parts[3])
28     w=int(parts[4])
29     h=int(parts[5])
30
31     if(y < 1000):
32         roi = img[y:y+h, x:x+w]
33
34 cv2.imwrite("/home/lunet/corq/Documents/Data/to_be_copied/acl/cropped321/frame%d/%d.jpg"%(
frame,count), roi)
35     else:
36         continue
37
38

```

Figure 3.4: clip code



Figure 3.5: clipped player



## Chapter 4

# Setting up environments

- Virtual environments for python:

A virtual environment is very useful for this project, since it provides isolated working copy of Python with its own files, therefore a virtual environment is created using anaconda platform. - Installing required python packages:

```
(base) corq@LNX-GRID-3:~$ source activate yolo  
(yolo) corq@LNX-GRID-3:~$
```

Figure 4.1: Activate virtual environment

There are lot of python modules are used in this projects, with anaconda platform, most of them can be installed easily with one simple command in the terminal such as openCV.(conda install -c conda-forge opencv)

- Installing darknet package:

By following this post, <https://pjreddie.com/darknet/yolo/>, (Redmon et al., 2018), Darknet platform is installed.

# Chapter 5

## Training Process

### 5.1 Labelling Data

Labellmg is used for labelling the data, it generates XML files in PASCAL VOC formats, simply use mouse to crop out the number of the jersey and label the number.



Figure 5.1: Labellmg interface

### 5.2 Integrating Data

After labelling data, an folder is created in VOC2007 data format inside the darknet package, which contains annotations, original images, labels translated for YOLO training, then training set and validation set is generated by scripts from darknet.

Once VOC dataset is ready, a scripts from darknet package was used to generate labels and a txt files which include the directories of the training data to feed to the darknet platform.

```

<annotation>
  <folder>AC5_1st</folder>
  <filename>98.jpg</filename>
  <path>/home/lunet/corq/Documents/FYP/cropped_data/AC5_1st/98.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>32</width>
    <height>87</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>4</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>17</xmin>
      <ymin>36</ymin>
      <xmax>25</xmax>
      <ymax>51</ymax>
    </bndbox>
  </object>
</annotation>

```

Figure 5.2: xml annotation

	name	Last modified	File size
0			
Annotations		seconds ago	
Annotations		2 months ago	
imageSets		4 months ago	
JPEGImages		2 months ago	
labels		2 months ago	
gray.py		3 months ago	1.05 kB
imageSets.py		4 months ago	613 B

Figure 5.3: dataset folder in VOC format

## 5.3 Modifying configuration

Several files in the darknet package regarding to the training were modified for training a number detector. Since VOC format dataset is used, following corresponding files are modified:

- cfg/voc.data: CFG file is a generic preference files, it stores class number, directories related to the training.

```

1 classes= 10
2 train = /home/lunet/corq/Documents/FYP/darknet/2007_train.txt
3 valid = /home/lunet/corq/Documents/FYP/darknet/2007_test.txt
4 names = data/voc.names
5 backup = backup

```

Figure 5.4: cfg/voc.data

- data/voc.names: this file record class names for the classification, in this case, they are 0 to 9.

- cfg/yolov3-voc.cfg: this file records the structure of the yolov3 network, training mode was switched by uncomment the training configuration in the scripts, then class number was changed to 10 for each yolo network layer.

```

1 0
2 1
3 2
4 3
5 4
6 5
7 6
8 7
9 8
10 9

```

Figure 5.5: data/voc.names

```

[net]
# Testing
# batch=1
# subdivisions=1
Training
batch=64
subdivisions=16
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

```

Figure 5.6: cfg/yolov3-voc.cfg: switch to training

```

[yolo]
mask = 6,7,8
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
classes=10
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1

```

Figure 5.7: cfg/yolov3-voc.cfg: change class number

## 5.4 Training

To save some time from training, a pre-trained yolov3 weights from darknet were used, it is downloaded from the official website, then training was started using the corresponding command showed in figure 5.8. During the training, darknet will print out relative information about the training such as average

IOU, obj detected, loss of each batch, these information can tell users how the training go. For instance, if the loss stop decreasing for many batches, then the training is going badly.

```
corq@INX-GRID-3:~/Documents/FYP/darknet$ ./darknet detector train cfg/voc.data cfg/yolov3-voc.cfg backup/yolov3-  
-voc 50000.weights
```

Figure 5.8: training command

```
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.000000, .5R: -nan, .75R: -nan,
Region 62 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.000000, .5R: -nan, .75R: -nan,
Region 94 Avg IOU: 0.00832, Class: 0.33417, Obj: 0.996770, No Obj: 0.003610, .5R: 1.000000,
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.000000, .5R: -nan, .75R: -nan,
Region 62 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.000000, .5R: -nan, .75R: -nan,
Region 94 Avg IOU: 0.853804, Class: 0.499365, Obj: 0.999395, No Obj: 0.002345, .5R: 1.000000,
count: 2
```

Figure 5.9: training

## 5.5 Augmentation

Augmentation on the dataset was done by following the project from <https://github.com/mukopikmin/bounding-box-augmentation>, methods include rotation, clip, adding noise, these augmented dataset was used to train for another weight.

## Chapter 6

# Detection on images

With command `./darknet detector test cfg/voc.data cfg/yolov3-voc.cfg backup/yolov3-voc-final.weights 01.jpg`, darknet framework will produce detection with corresponding weight on the image. There are 2 weights trained in this project, one is trained with 500 original image, the other one is trained with 500 original image with 900 additional images with different augmentation on the original images.

After comparing results of original dataset weights and augmented dataset weights, a conclusion that augmented dataset weights performance is not as good as the original dataset weight can be draw since the average accuracy dropped when the detector use the weights trained from augmented dataset, reason caused this could be the dataset was over augmented, for instance the rotation angle was too big thus the detector could not recognise data from the training set. (Jason et al., 2017)

There are two example results from applying detection on a image from the dataset.



Figure 6.1: original weights result



Figure 6.2: augmentation weights result

# Bibliography

- [1] Sebastian, G., Karsten, M., Ralf, S.(2016) Jersey Number Recognition using Convolutional Neural Networks. Retrieved 10th March from [http://www.vap.aau.dk/cvsports/wp-content/uploads/2016/01/ICCV\\_Jersey\\_Number\\_RalfSchaefer.pdf](http://www.vap.aau.dk/cvsports/wp-content/uploads/2016/01/ICCV_Jersey_Number_RalfSchaefer.pdf)
- [2] Redmon, Joseph and Farhadi, Ali.(2018) YOLOv3: An Incremental Improvement. Retrieved 25 February from <https://pjreddie.com/darknet/yolo/>
- [3] Joseph, R., Ali, F., (2018) YOLOv3: An Incremental Improvement
- [4] Manish, C., YOLO — You only look once, real time object detection explained. Retrieved 11th March from <https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006>
- [5] Joseph, R., Ali, F., (2018) Retrieved 13th March from <https://pjreddie.com/darknet/yolo/>
- [6] AlexeyAB, YOLOv4-Windows and Linux version of Darknet Neural Networks for object detection. Retrieved 28th March from <https://github.com/AlexeyAB/darknet-how-to-train-tiny-yolo-to-detect-your-custom-objects>
- [7] Zhxing, (2018) Retrieved 26th March from <https://www.jianshu.com/p/91eafe0f3719>
- [8] aleju, (2019) Image augmentation for machine learning experiments. Retrieved 18th March from <https://github.com/aleju/imgaug>
- [9] Jason, W., Luis,P.(2017) The Effectiveness of Data Augmentation in Image Classification using Deep Learning.



# List of Figures

1.1	YOLO compared with other model on COCO dataset . . . . .	3
1.2	darknet folder . . . . .	4
2.1	How YOLO works . . . . .	5
3.1	example frame . . . . .	6
3.2	players position in each frame . . . . .	7
3.3	* . . . . .	7
3.4	clip code . . . . .	7
3.5	clipped player . . . . .	7
4.1	Activate virtual environment . . . . .	8
5.1	LabelImg interface . . . . .	9
5.2	xml annotation . . . . .	10
5.3	dataset folder in VOC format . . . . .	10
5.4	cfg/voc.data . . . . .	10
5.5	data/voc.names . . . . .	11
5.6	cfg/yolov3-voc.cfg: switch to training . . . . .	11
5.7	cfg/yolov3-voc.cfg: change class number . . . . .	11
5.8	training command . . . . .	12
5.9	training . . . . .	12
6.1	original weights result . . . . .	14
6.2	augmentation weights result . . . . .	14