YOLOv3 as a clutter object detection program and further application

VUONG HO, NGHIA NIM, MINH NGUYEN*

MATH AND SCIENCE SUMMER PROGRAM SUMMER CAMP 2019

Monday, July 15th, 2019

Abstract

Our work here at MaSSP 2019 is COMPLETED and this paper will disclose exhaustively our process and results in making a program that uses YOLOv3 (a object detection algorithm) to detect specific clutter object in realtime through the camera. The purpose of this program is to help people find what is, supposedly, in front of their eyes, but is too hard to see due to low visibility or small sizes (or both). We also did some additional tweaking and testing out the YOLOv3 algorithm in detecting small stuff. Our code is all on our team's repository: `https://github.com/goodudetheboy/MaSSP-Team3`

I. INTRODUCTION

Howdy, partner! A fellow AI interest greets you. It is this time of the year again, is it not? People at my summer camp trying to pick up the pace on their own report and mentors busy checking all the stuff. But before we get into the ice-breaking parts, we want to let you know that the algorithm we will be discussing in this article is not ours, the YOLOv3, created by Joseph Redmon, but we are simply trying to use it to create a program. All credits go to this very cool AI guy: `https://pjreddie.com`.

In this article we will be explaining a little bit how YOLOv3 works, go through our program that use this algorithm and also look at some of our tweaks to make it more effective.

II. WHAT IS YOLOV3

YOLO is a clever neural network for doing object detection in real-time.[1] YOLOv3 is very much like a typical CNN, its structure is similar and could be confused with R-CNN. However, where CNN goes from regions to regions to try and extract basis functions from a given function space, Yolo does this simultaneously. That is, it employs a different method to extract information.
It follows these steps:

- First, it divides an image into smaller squares with equal area (13x13, 14x18 for example)

Image 2.1: Gridding the image

- These square simultaneously extract features (using PCA) and make predictions

*A thank you or further information

for the appropriate size of anchor boxes within. These boxes can be thought of as clusters of points in a vector space. Their size gets more and more fit with the object within (or sometimes bigger than) the grid.

Image 2.2: Creating anchor boxes

- After choosing the best boxes, each grid will make a prediction of what the item within the box may be. This is where we will be comparing our predictions with our ground truth (dataset).

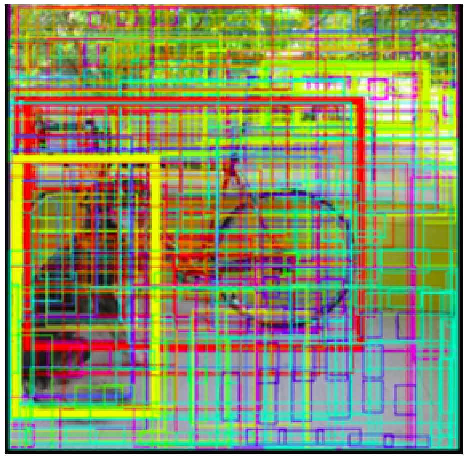
Image 2.3: Classification

- To get a geometric idea of what this looks like, imagine each prediction is a probability vector inside a vector space. This vector space contains clusters of points, and each of these clusters comprises our already defined and comparable ground truths. The algorithm will predict the dimension with the highest probability, meaning it is closest to that corresponding area (cluster).
- Finally, the **IoU** parameter comes into play. If there is no value higher than our **IoU** in that vector then that prediction will be discarded, leaving only ones that the program is sure of and those we have properly trained and defined.

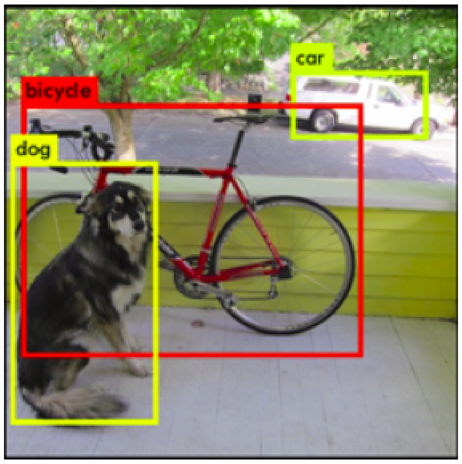

Image 2.4: Final result

Now that all of the lame AI algorithm stuff is finished, let's move on to the more interesting part, the program.

III. Our program

i. Our TEFPA

At the start of *any* Artificial Intelligence/Machine Learning project, we all have something called a TEFPA, which is an acronym for Task, Experience, Function Space, Performance and Algorithm. TEFPA is like a over-simplified blueprint that explains the 5W1H of an AI program. We have our program's TEFPA listed in the following.

i.1 T - Task

Our program is to aid users in finding small and partially hidden objects in clutter. For example, our program is to help deal with scenarios when you want to find keys or purses that are tucked away somewhere in the books .

i.2 E - Experience

Our training datasets come from `cocodataset.org`, *a large-scale object detection, segmentation, and captioning dataset* [2]. This dataset contains pictures which have clutters of objects, so we think this will be suitable for the training of YOLOv3.

i.3 F - Function Space

The program works with images extracted from realtime footage of the camera used.

i.4 P - Performance

The classification of boundary boxes was tested and trained using **Sum of squared errors**

$$SSE = \sum_{i=1}^{n} (x_i - x_m)^2$$

where n is the number of observations x_i is the value of the ith observation and x_m is the mean of all the observations [3].

i.5 A - Algorithm

YOLOv3

ii. How we trained our model

ii.1 Why train again?

Originally, we used the Yolov3-spp and the Yolov3-tiny weight set that are already pre-trained by Joseph Redmon. We tested on some of the images and found the result was quite opposite to what we have in mind: the small objects are being left out. I have put a table here listing the result. (We have chosen to use **IOU** number of 0.55 because we think that it is the best threshold to achieve the best accuracy for our program)

Table 1: *Detection results using Yolov3-spp weight set (IOU=0.55)*

Item[a]	Detected	Undetected	Total
Phone	5	155	160
Bottle	126	74	200
Key Chain	127	23	150
Book	107	14	121
Glasses	7	23	30

[a] All images contain only one specific item that our program has to find

Table 2: *Detection results using Yolov3-tiny weight set (**IOU**=0.55)*

Item[a]	Detected	Undetected	Total
Phone	7	154	161
Bottle	125	25	150
Key Chain	93	37	120
Book	56	64	120
Glasses	17	103	120

[a] All images contain only one specific item that our program has to find

As can be clearly observed in the above table, the program has a wide fluctuation in the percentage of detection. Some items, like bottle, key chain and book, can be detected pretty well using the weights (which can be explained by its size and its unique shape), when others, like phone and glasses, is not really detectable by the program (maybe because of the phone's shape being similar to other items like laptop).

ii.2 The training

So in order to cope with this discrepancies, we have decided to train our own weight using the above TEFPA. In order to save time, we used the good old Yolov3-spp weights (due to it being *slightly* better[1]), cut the SPP block in the

[1] in terms of or result of percentage of detection, not speed. Compared to the Yolov3-spp, the Yolov3-tiny is trained less and is optimized for speed [4], but the Yolov3-

Yolov3-spp (see image 3.2.1), use something our mentor called a "blocker" to train only specific data and voilà, our model.

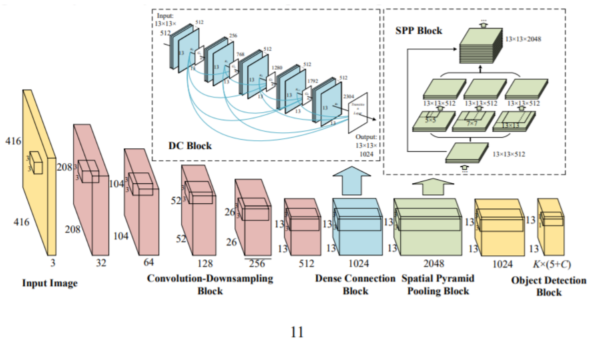

Image 3.2.1: A diagram showing the Yolov3-spp neural net. Our model is the same as this one, but without the SPP block.

With our model in mind, we moved on to the next part: hand-picking training data. We specifically targetted pictures that:

1. contains Phone, Bottle, Key Chain, Book, Glasses;
2. contains clutter of objects;
3. has our desired items small compared to its surrounding.

In order to maximize the speed of the training, we have also cut some convolutional layers, and drop the downscale for each pictures.

iii. Result

Here are our result with the weight set we get from training our own model.

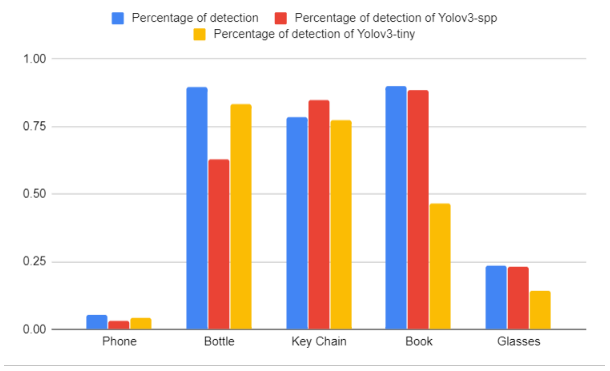

*Image 3.3.1: Our result compared to the other weight sets (the blue one is ours) with the **IOU** number of 0.55.*

The following table contains the percentage of detection for varying **IOU** value.

Table 3: *Detection results in varying **IOU***

IOU value / Item	0.25	0.35	0.45	0.65
Phone	51.4	29.6	15.33	2.1
Bottle	93.2	94.2	91.6	86.3
Key Chain	96.37	92.63	85.67	74.3
Book	89.9	95.3	92.55	85.36
Glasses	63.43	53.7	32.669	15.3

Our conclusion is that, this program works quite well. When compared with the our own result of other weight sets as well as outside result, our program pushes out a better than expected result. Regarding the feasibility of our program, it may not be available for widespread use because of the demanding nature of the Yolov3 model. To put this in context, a CPU of 3.3 GHz only output about 1 frame per second and the highest frame per second we have achieved with our program is to run the program using a GPU, which is 5-6 frames per second for a GTX 1050 Ti running cuDNN version 10.2. Considering the demanding nature of our program, we do not expect this to be in widespread use, such as as a mobile phone application.

spp has the SPP blocks (spatial pyramid pooling), which arguably "gets the best features in Max-Pooling layers" [5]

IV. Discussion

With our models and results, we expect our program to have way better uses than clutter object detection. What we are aiming at is the application of YOLOv3 in other field, such as medical (detection of cancer cells in a realtime surveillance of patient), chemistry (detection of rare chemicals in ore) and many more others.

References

[1] Hollemans, Matthijs. "Real-Time Object Detection with YOLO." *Real-Time Object Detection with YOLO*, 20 May 2017, https://machinethink.net/blog/object-detection-with-yolo/. Accessed Saturday, July 6th, 2019

[2] Common Objects in Context, COCO. "What is COCO?" *COCO dataset*, http://cocodataset.org/. Accessed Saturday, July 6th, 2019.

[3] Common Objects in Context, COCO. "Error Sum of Squares (SSE)." *Error Sum of Squares*, https://hlab.stanford.edu/brian/error_sum_of_squares.html. Accessed Saturday, July 6th, 2019.

[4] Redmon, Joseph. "Tiny Darknet" *Tiny Darknet*, https://pjreddie.com/darknet/tiny-darknet/. Accessed Saturday, July 6th, 2019.

[5] "YOLOv3 SPP and YOLOv3 difference?", https://stackoverflow.com/questions/54998225/yolov3-spp-and-yolov3-difference. Accessed Saturday, July 6th, 2019.