Bogazici University Computer Engineering Department
CMPE 591 Project Report

# Traffic Cone Finder using YOLO

Dilara Keküllüoğlu
<dilarahae@gmail.com>

**June 1, 2018**

# 1 Introduction

In this project, we are using object detection methods to detect traffic cones. This is a Group 3 project. However, we are helping Group 5 -TUBITAK Challenge Team to detect traffic cones for their vehicle. They provided us a 50sec video filmed with the specifications of the challenge; camera 50cm from ground, cones with 1m distance between, vehicle speed 30km/h. We annotated this video frame by frame. We also used dataset collected by DukeCone [3] and annotated some selected ones. In total our training dataset have 123 images from internet. Testing dataset has 10 images from internet and 20 frames from the video.

We used mAP(mean average precision) as our evaluation metric. It is used in Pascal VOC challenge and indicates the area under the precision-recall curve for a class. Since we have one class, mAP is equal with the average precision. For the project milestone, our training system had 48 images. After creating the testing dataset and applying the milestone results we achieved 48.96 mAP. Our best result after training the system with 123 images is 64.66.

# 2 Methodology

We used YOLO [4] for our project. It is 106-layer convolutional network that is fast and precise. We installed the YOLO to our Windows system with the instructions of Alexey AB's implementation [1]. They give detailed instructions for installing and using YOLO for Windows and Linux systems. They also show how to train the system for custom objects. We followed them to create our one-class (trafficcone) system and train the model.

We based our dataset to the one provided in DukeCone [3] system. We extracted traffic cones that are similar to ours and annotated the images. Annotation is done by the program provided by Alexey AB [2] by hand. Training dataset has 123 images collected from Google Images. Testing dataset has 10 images collected from Google Images and 20 images extracted from the video provided by Group 5. We also annotated these by hand.

We trained our system for 3500 epochs. Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005

Dataset can be found in our dropbox link :
`https://www.dropbox.com/sh/ay9wf7ii81q5zif/AADwIb9HkvpBmUDJvKpNl0Xna?dl=0`

# 3 Code and Usage

As we stated, we used the instructions of Alexey AB's [1] YOLO implementation. We adjusted the configuration files for our system. We have only one class, we modified the class and filter parameters according to the instructions.

We share our configuration files in our github page :
`https://github.com/dilara91/conefinder-yolo`
Also resulting weights can be found in our dropbox link :
`https://www.dropbox.com/sh/ay9wf7ii81q5zif/AADwIb9HkvpBmUDJvKpNl0Xna?dl=0`
To run the system for evaluation use the following line in terminal.

```
darknet.exe detector test data/obj.data yolo-obj.cfg yolo-obj_3200.weights img.jpg
```

This is for Windows. Linux code can be run with replacing darknet.exe with ./darknet. You need to compile darknet in your system with given instructions in Alexey AB's tutorial [1]. data folder and .cfg file can be found in our github page given above. This command will use the weights given to predict traffic cones in img.jpg and will write the results in predictions.png.

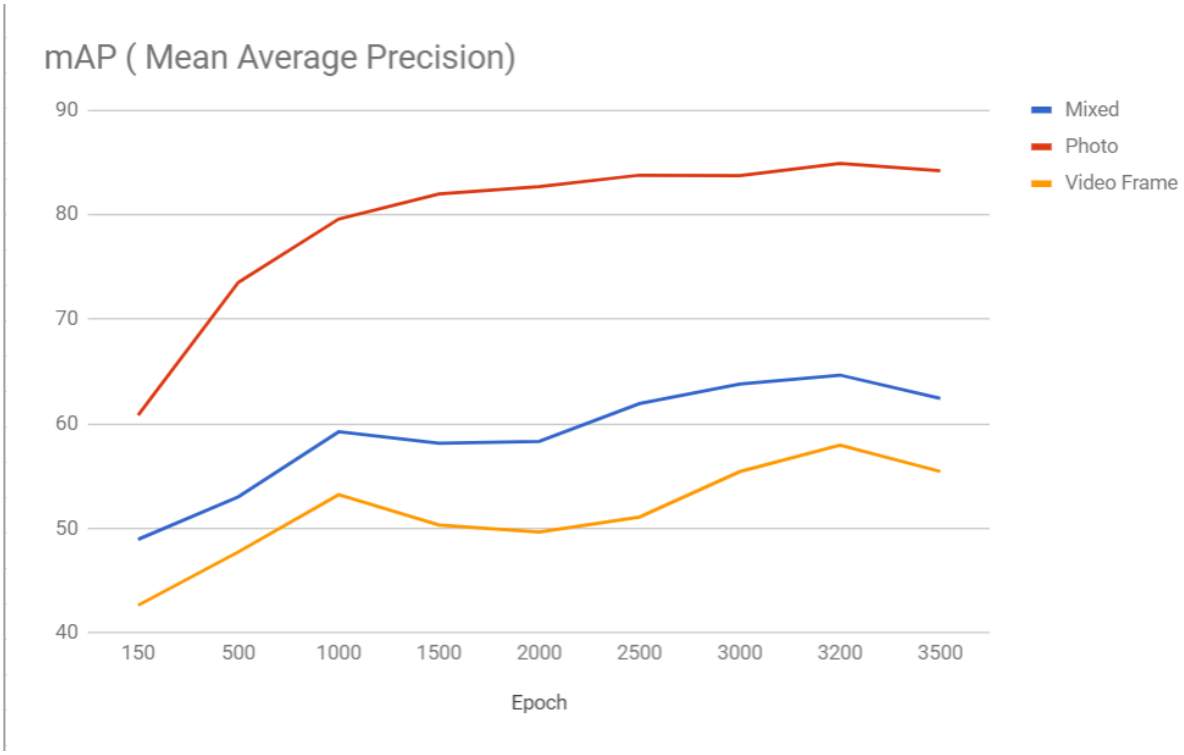


Figure 1: mAP Values with Different Epochs

## 4 Evaluation

Here are the results for the testing dataset. Mixed dataset consists of 10 Google Images and 20 video frames provided by Group 5. Photo is for only Google Images and Video Frame is for 20 video frames.

150 epoch one is the project milestone weights and best performing weights is created with 3200 epochs.

Our system best performs with the Google Images (Photo) as expected since we trained it with similar images. Still we achieve 57.97% mAP for the video frames. The middle line is the weighted average of both results. Our best results come with 3200 epochs.

As you can see our performance usually increases with the epoch number. This means the system did not overfit yet and still has room for improvement with more epochs.

You can find output videos created with the weights in the dropbox folder `https://www.dropbox.com/sh/ay9wf7ii81q5zif/AADwIb9HkvpBmUDJvKpNl0Xna?dl=0`.

There are two videos; one with the weights used for milestone presentation (150-epoch.mp4) and one for the best result we achieved in the end (end-result.mp4). You can also find the images used in this paper and others in the dropbox folder.

## 5 Improvement & Future Work

As we stated, we trained our system for 3500 epochs. However, there is still a room for improvement. System did not overfit and continued to improve. Hence, iterating for more epochs would be useful. We had to limit the epochs to 3500 because of time and hardware limitations.

Having more extensive dataset is always useful. Increasing the number of training images will improve the mAP values. In addition, testing dataset can be increased in size too.

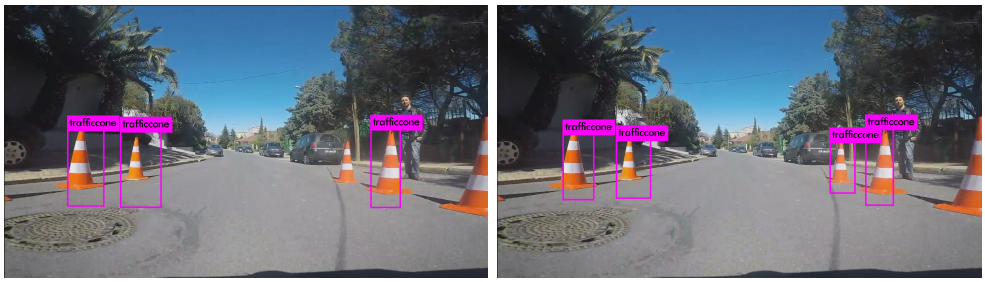


Figure 2: Video Frame with 150 and 500 epochs


Figure 3: Video Frame with 1000 and 1500 epochs

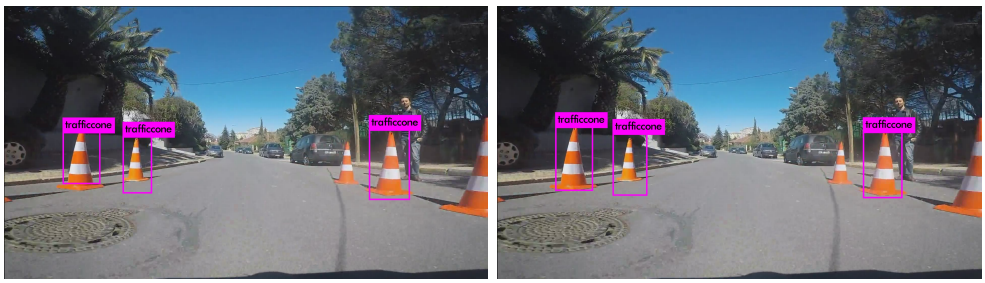


Figure 4: Video Frame with 2000 and 2500 epochs


Figure 5: Video Frame with 3000 and 3200 epochs

# Bibliography

[1] *Alexey AB's Windows Yolo*. accessed at June 2018.

[2] *Alexey AB's Yolo Mark*. accessed at June 2018.

[3] *DukeCone*. accessed at May 2018.

[4] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.