# CS 766: Computer Vision Midterm: Computer Vision in Self-driving Car

Sean Chung (cchung49) & Shang-Yen Yeh (syeh6)

## 1 Problem Statement

To make self-driving car feasible, the fundamental thing is to let the self-driving car understand the surroundings, and this is based on object detection system with Computer Vision.

## 2 Data

We obtained data resource from Udacity self-driving car project annotated by CrowdAI using a combination of machine learning and humans. The data (images) mainly consists of daylight sights in Mountain View, California. There are 9,420 images and 72,064 labels, which annotate objects in images. The labels are stored in a comma-separated values file and it has 7 headers (xmin, xmax, ymin, ymax, filename, class, URL). xmin, max, ymin, ymax are corresponding to two points (upper-left and lower-right corners) of an object; filename is corresponding to image; class is the label (i.e., car, truck, and pedestrian).

# 3 Current Progress and Results

## 3.1 Approaches

In our project proposal "current state-of-the-art", we give two kinds of approaches. In this stage, we choose to start with "Deep Learning approaches" and You Only Look Once (YOLO) specifically.

Different from other object detection algorithms use regions to localize the object within the image. YOLO look at the complete image. Also, YOLO is faster than other object detection algorithms in the case of video stream. All these reasons lead us to choose YOLO as our first approach.

## 3.2 Results

We use OpenCV DNN (Deep Neural Network) module as running inference on images/videos with pre-trained YOLO model "yolov3.weights" and "yolov3.cfg". How YOLO works is that we take an image and split it into an SxS grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network outputs a class probability and offset values for the bounding box. The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image.

With pre-trained YOLO model, we can detect almost every vehicle in the image from the Udacity self-driving car project data set. See below :

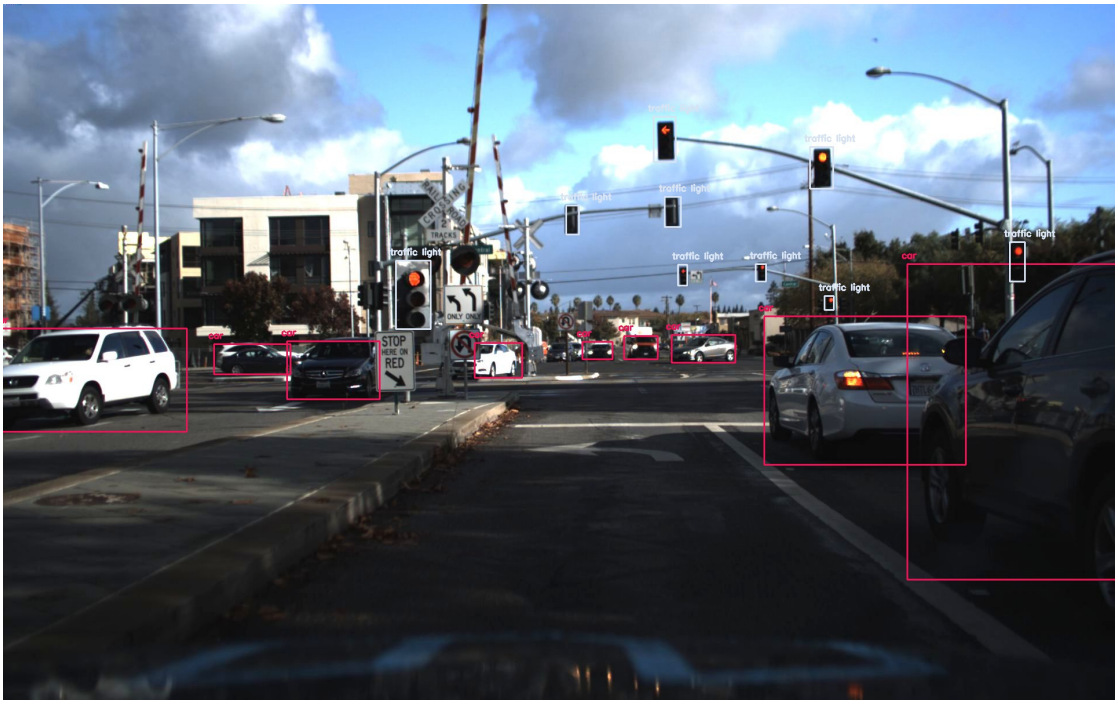


FIGURE 1 – Result of YOLO


FIGURE 2 – Result of YOLO

# 4 Difficulties during Implementation

There is a convenient API – Tensorflow Object Detection API, and it already has some pre-trained models available. However, these pre-trained models are not ideal in our project as they might be trained by some other types of data. In other words, they might be trained by some data which does not pertain to street sights. As a result, they may not be able to effectively identify vehicles, pedestrians, road signs, etc. In order to cope with this issue, we have to train our own models, and thereby difficulties arose : (1) How do we train models on this API and (2) Time cost.

## 4.1 API Implementation

We ran into many issues when implementing Tensorflow object Detection API, albeit there are several related tutorials. First of all, we need to convert labels file-type to TFRecord, which is an individual aggregated compact file summing up all the data. Next, using this TFRecord file to train models. We stuck at these two steps for a while since we are not familiar with the API, and there are some inconsistency in our data format and API's, etc.

## 4.2 Time Cost

A major issue for model training is time cost since image size is generally larger than text files. For instance, it takes over 60 hours to train the model. One problem is that our hardware is not powerful enough to run it on GPU ; namely, we could only train models on CPU. One way to improve this could be by applying early-stop and\or checkpoint to potentially prevent from running all data every time.

# 5 Proposal Revise

For our project proposal, we have fulfilled our midterm goal "being able to detect objects in the frame and draw rectangle for all objects". However, for our final project goal, we tend to train the YOLO model with our own data set,

instead of using the pre-trained one. Also, we prefer to apply the frame detection into video streaming. Thus, we decide to give up the lane drawing function.

# 6 Timeline

| Time | Goal |
|---|---|
| 4/6 | Train own YOLO model |
| 4/13 | Experiment second approach |
| 4/20 | Apply on video stream and measure precision & recall |
| 4/22 or later | Presentation |
| 5/6 | Project Webpage |

# 7 Reference


1. Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Commun. ACM, 60, 84-90.
2. Redmon, J., Divvala, S.K., Girshick, R.B., & Farhadi, A. (2016). You Only Look Once : Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788.
3. Szegedy, C., Ioffe, S., & Vanhoucke, V. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. AAAI.
4. Gavrila, D., & Philomin, V. (1999). Real-Time Object Detection for "Smart" Vehicles. ICCV.
5. Malisiewicz, T., Gupta, A., & Efros, A.A. (2011). Ensemble of exemplar-SVMs for object detection and beyond. 2011 International Conference on Computer Vision, 89-96.
6. Viola, P.A., Platt, J.C., & Zhang, C. (2005). Multiple Instance Boosting for Object Detection. NIPS.
7. Girshick, R.B., Donahue, J., Darrell, T., & Malik, J. (2016). Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38, 142-158.