Object Detection for Self-driving Cars

Mahmoud Dahmani
CMSC498L Introduction to Deep Learning
University of Maryland
College Park, MD 20740
dahmani@umd.edu

Aditya Khopkar
CMSC498L Introduction to Deep Learning
University of Maryland
College Park, MD
akhopkar@umd.edu

Abstract

The research and developments in Autonomous drive vehicles have grown over the years. With smarter algorithms and technological advancements coming into play, object detection for the autonomous vehicles still holds the position as one of its strongest pillars. Through this project, we aim to replicate the performance of an object detector for an autonomous vehicle, in real-time. It is to be noted that, when dealing with object detection for an autonomous vehicle, the performance of the model in terms of speed and time is one of the most critical criterion to gauge the suitability of the model for the application of autonomous drive. We tap the power of single stage detectors through this project.

1 Introduction

There are two broad categories of detectors heavily used, Single stage detectors and Multi stage detectors. While single stage detectors such as SSD, YOLO need to take just one shot of the image to learn the objects in the image, the multi-stage detectors such as R-CNNs, Faster R-CNNs, Mask R-CNNS are used to take more than one shots of the image.

In a real world scenario, Object detection could be deployed to aid the autonomous vehicle system to understand the scene of the environment around. From identifying vehicles, to identifying pedestrians, object detection could be responsible in identifying the dynamic environment the autonomous vehicle is in. This would thus, help the vehicle take better decisions to ensure safe traversing through the environment by employing sophisticated path planning algorithms. It is thus, imperative to perform this detection in real-time and thus, the speed of performance is a crucial parameter.

In this project, we implement this paradigm to test on a custom dataset, by using a pre-trained model on COCO dataset. Single-stage detectors are considered one of the most sophisticated detectors. Due to their superlative detection performance in terms of time complexity, single stage detectors could be used for real-time applications. Through this project, we intend to tap into the power of these detectors and obtain as accurate result as possible for object detection on our custom dataset. We provide a comparative study with different Single stage detectors employed.

2 Data

In this project, we wanted to leverage the power of deep neural networks that were trained on a large amount of data, and therefore we picked the single stage detection models that was pre-trained on the COCO dataset as it contains thousands of images of the objects that are relevant to driving such as bus, car, truck, traffic signs, pedestrians, etc.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

To validate the performance of the model on real-world data that is relevant to driving,

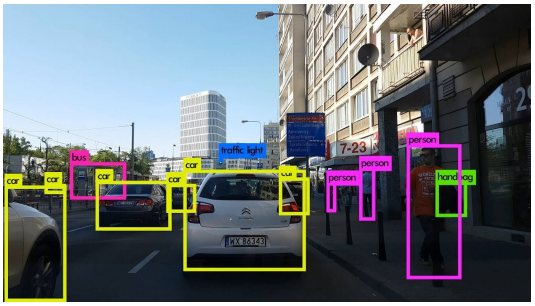


Figure 1: Object Detection in COCO

we used the Udacity driving dataset [3] which contains 65,000 labels across 9,423 frames collected from a Point Grey research cameras running at full resolution of 1920×1200 at 2Hz. We adopted mean average-precision as the evaluation metric of our object detector.

To further test the model, we have used our own custom dataset collected from our personal smartphone video recorder near our surrounding areas.

3 Related Work

When dealing with a real-world problem such as object detection for an autonomous vehicle, the detection process must be fast enough to be safely implemented in real-time. While R-CNNs are commonly used for object detection, instance segmentation etc., they aren't fast enough to be employed in a real-time application. A single stage object detector proves better in such scenarios.

We look into some of such Single Stage Detectors implemented and a brief rundown through their features for COCO dataset (refer fig.5):

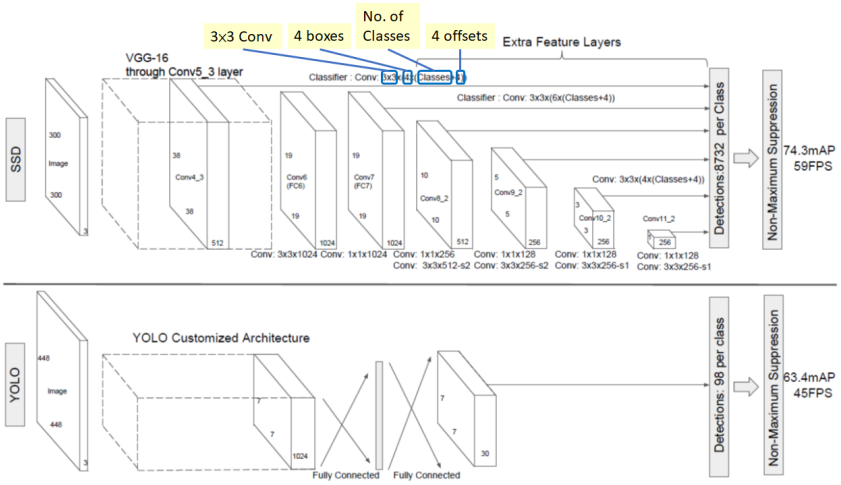


Figure 2: YOLO vs SSD architecture

1. **SSD300 and SSD500** : The entire architecture remains similar to other object detectors by employing Convolutional Neural Networks, we obtain feature map of $m \times n$ with p channels (refer fig.3). Each location we get k bounding boxes. The bounding boxes have different sizes and aspect ratios depending on the detection. For each of the bounding boxes, we compute c class scores and 4 offsets relative to the original default bounding box shape. SSD300 has achieved 41.2% mAP (mean average precision) at 46 FPS, and SSD500 has achieved 46.5% mAP at 19 FPS.

2. **YOLOv1** : Follows similar architecture as SSD as it follows the same logic of taking a single shot of the image to detect objects, what differs is the number of offsets taken into consideration. Unlike SSD, YOLO uses 5 offsets (Pc, xmin, ymin, xmax, ymax). Where, Pc is the objectness score and [xmin, ymin, xmax, ymax] are the bounding box coordinates. The version has reportedly reported 63.4% mAP at 45 FPS.

3. **YOLOv2** : YOLOv1 makes significant number of localization errors and has relatively low recall. To achieve improvements in recall and minimize localization errors while keeping the accuracy of the network intact, the developers of YOLOv2 implemented batch normalization, high resolution classifier, convolution with anchor boxes, which facilitates multiple object detection in the grid cell. YOLOv2 achieved 48.1% mAP at 40 FPS

4. **YOLOv3** : Minor improvements were done on the top YOLOv2 to develop YOLOv3, the latest version. YOLOv3 in training doesn't follow softmax instead follows an independent logistic regression for class prediction. YOLOv3 is generally a heavier model than the others but is more accurate. YOLOv3 has reported 55.43% mAP at 35 FPS.

4 Approach

Our goal is to implement a robust single-stage object detector that can achieve state-of-the-art results on the real-time detection problem for self-driving cars. We first wanted to leverage the power of pre-trained detection deep learning models and test them on a benchmark dataset as well as our own custom dataset to see if they meet the requirements of our system in terms of both accuracy and speed. Therefore, we chose models that were trained on the COCO dataset, as it already includes images of the objects found in a driving scene (i.e. cars, pedestrians, traffic signs, etc). YOLOv3, YOLOv3-tiny and MobileNet-SSD are the architectures we implemented for single-stage object detection. After evaluating the performance of each model, we found that the COCO YOLOv3-based detector meets our design objectives (i.e. accuracy and speed), and hence there was no need to do transfer learning or train the model from scratch. Our approach is thus divided into two sections, namely, detection pipeline and evaluation pipeline.

4.1 Detection Pipeline

We wanted to use Python to develop the whole framework, so we adopted OpenCV's implementation of the YOLOv3 models, since it is the only library that supports the latest official version of YOLO in Python. To fully build an accurate YOLO-based object detector, we needed to postprocess the network output in two stages by first filtering out predictions with low confidence scores (i.e. objectness probaility) and then applying a Non-Max Suppression algorithm to eliminate all the weaker detections (pertaining to the same object). As a result, our YOLO-based detector ended up having to hyperparameters, the minimum confidence threshold, and the minimum bounding boxes overlap (defined as the intersection over union measure). We also managed to to tradeoff between accuracy and speed of our YOLO detector without any training, by simply changing the size of the model. Essentially, We utilized YOLOv3-416 and YOLOv3-tiny. Moreover, we also implemented an equivalent SSD-based detection network which uses MobileNet as the base network. We chose this variant of SSD because MobileNet is a compact network mainly aimed at mobile platforms, making it suitable for real-time applications. Plus, it is pre-trained on COCO and then fine-tuned on Pascal VOC (which also includes all the classes relvant to our application).

4.2 Evaluation Pipeline

A crucial stage in deep-learning based systems is to evaluate their performance once a model is trained. We will do so by testing on the Udacity self-driving dataset [3] which contains 9,423 frames, and the mean average precision (mAP) will be used as the evaluation metric for our object detector. The mAP is defined as the mean of the classes' average precision; which corresponds to the area under the precision-recall curve. It is thus, imperative to understand the concept of precision and recall.

Precision is defined as the ratio of True positives (TP) over total number of predicted posi-

tives (Total predictions).

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall is defines as the ratio of true positives (TP) and total of the ground truth positives given by:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

In the evaluation paradigm, we are using a mAP module to produce the performance metrics of the

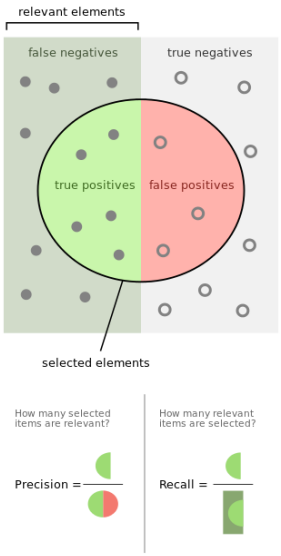


Figure 3: Precision vs Recall

object classes detected in the image frame. The module takes the ground truth values of the bounding boxes and the detection results of the bounding boxes.

- Ground Truth Input: The module take the bounding boxes values of the ground truth images stored in a .txt file of the following format:

  ```
  <label xmin ymin xmax ymax>
  ```

- Detection Results Input: This is obtained by the performing the detection of objects in the custom dataset. The input to the module is accepted in the following format:

  ```
  <label confidence xmin ymin xmax ymax>
  ```

  Where, label = class label, [xmin ymin xmax ymax] = bounding box parameters and, confidence is the confidence probability of the object detected.

The mAP is then computed by computing precision and recall based on Intersection over Union computed between the detection and the ground truth bounding boxes. A threshold parameter is used to find the precision. Once, precision and recall values are found, AP (average precision) is computed by finding the area beneath the precision-recall curve.

$$AP = \int_0^1 p(r) \tag{3}$$

Since, precision and recall values are always between 0 and 1, AP will be between 0 and 1. Once, AP is computed for each class, mean of AP for each class is computed. This results in finding mean average precision - the index of performance metric of the detector.

5 Experimentation and Results

In order to ensure the completeness of the implementation, we follow two paradigms as suggested in the approach section. The detection paradigm needs two hyper-parameter tuning: minimum confidence and minimum overlap, which were tuned to generate the required results. The setup for the experimentation was done on CPU of our systems. In order to establish a comparative analysis of the models, we examine YOLOv3, YOLO-Tiny and MobileNet-SSD.

Models	Minimum Confidence	Minimum Overlap	Inference Time
YOLOv3	0.5	0.3	1.001s
YOLO-Tiny	0.3	0.2	0.119s
MobileNet-SSD	0.3	-	0.095s

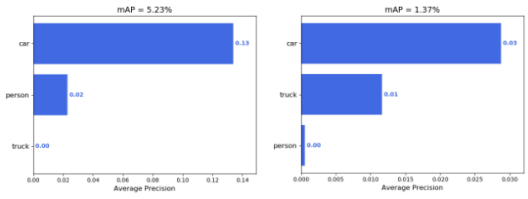
Figure 4: Model parameter comparison table

It can be seen that, MobileNet-SSD doesn't require the minimum overlap to achieve the results. The test output (ref. fig 5) was generated on the annotated dataset by the Udacity, as it wasn't possible for us to annotate our own custom dataset. It is to be noted that, for calculating the mAP for the

(a) (b) (c)

Figure 5: a: YOLOv3, b: MobileNet-SSD, c: YOLO-Tiny

evaluation of the model as suggested in Section 4.2, we need to have threshold set for minimum Intersection over Union achieved, which is set to 0.5 for the best evaluation of the model refer fig. 6. Thus, while YOLOv3 gave the most optimum results in detecting the objects with nearly mAP



a . b.

Figure 6: a. mAP visualization for MobileNet-SSD, b. mAP visualization for YOLO-Tiny

corresponding to 83% for about 80 classes, the model was found not to be efficient enough as it took an inference time of nearly more than 1 second, however can very well act as a real-time object detector given the computation is setup in GPU, as it is expected to give significant boost to the performance of the model. While, YOLO-tiny didn't produce the results as expected from an object detector in terms of precision, MobileNet SSD proved a bit better than YOLO-tiny in terms of speed and detection performance. It was observed that the two models detected the objects which were relatively closer than others more accurately. Thus, lowering the overall accuracy of the model.

The output on the custom dataset can be accessed from the following link: Video Outputs Link

6 Conclusion

Through this project, we achieved to obtain a comparative analysis of family of single-stage detectors such as: YOLOv3, MobileNet-SSD, YOLO-tiny. We compared the outputs from these models and evaluated the models using mean average-precision technique. During experimentation it was observed that the performance of YOLOv3 was extremely satisfying in terms of results and outputs. One interesting observation was that the object could be detected even beyond the scope of the image. The model extrapolated the bounding boxes to give an estimate of the object's presence in the image with utmost accuracy.

While YOLO-tiny and MobileNet-SSD models' advantage was that, they weren't heavy. However, they couldn't reproduce the detection with same performance metric as YOLOv3. Thus, YOLOv3 becomes a better choice for object detection in Self-driving cars due to its performance. The speed of the model can be accelerated by setting up the environment on GPU instead of CPU.

The presentation of this project can be accessed with this link: Presentation link

7 Future Work

While the project was initially designed to achieve outputs for YOLO, we achieved to have a comparative study with the family of single stage detectors. However, our future work plan goes as follows:

- 2 weeks plan: We would have implemented the YOLO object detector from scratch on PyTorch and trained the model on the same dataset used in this project - COCO. We could have had a comparative analysis with a pre-trained model and the model implemented from scratch.
- 2 months plan: We could have implemented a model with pipeline which would perform lane detection along with object detection and replace the detected 2D bounding boxes with 3D bounding boxes.

References

[1] Joseph Redmon & Ali Farhadi University of Washington *"YOLOv3: An Incremental Improvement"*

[2] Pre-trained model referred from *"https://pjreddie.com/darknet/yolo/"*

[3] Dataset from *"https://github.com/udacity/self-driving-car/tree/master/annotations"*

[4] Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko Weijun Wang Tobias Weyand Marco Andreetto & Hartwig Adam *"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications"*