A Comparative study on Real-time object detection in autonomous driving by using Google Tensorflow Object Detection API and YOLOv2

RUOGU LIU TONG LI
Ruogu Liu | ruogu@kth.se
Tong Li | tonli@kth.se

October 16, 2018

Abstract

Google Tensorflow object detection API is an open source object detector development system. Within the system, Google provided pre-trained detection models that have achieved state-of-the-art in object detection challenges and are widely used. On the other hand, YOLOv2, the state-of-the-art in object detection, can perform high accuracy while can still run very fast. Although these two detection systems are widely used in the industry, but hardly had they been used to train on a novel dataset and then compare the performance of new detectors in terms of accuracy and speed.

We train the KITTI dataset on these two object detection systems to get two new object detectors, and then compare the performance of these two detectors to infer which system is better in the context of autonomous driving. We find that Google Tensorflow Object Detection API performs better at accuracy while YOLOv2 performs better at speed.

Contents

1 Introduction

Object detection is a very exciting technology that is being used in autonomous driving. Detecting the object in real time and the accuracy of the detecting model[1] is very critical in some scenarios like detecting the pedestrian or traffic signs in autonomous driving. So, the model for the real-time object detection should be fast and accurate. Now with the advance of deep-learning and neural network framework like Caffe, YOLOv2[1], and Google's new Tensorflow object detection API [14], we can build the model for real-time object detection faster and even more accurate than before.

According to Google Research Blog[22], in 2016, Google's internal object detection system achieved state-of-the-art in the COCO detection challenge[15] and then was used in a sequence of research publications. The detection system has also been made into products, for example NestCam[16]. Now, with the release of the TensorFlow Object Detection API, this internal object detection system is released to the public. It's an open source object detection development system based on Google's Tensorflow. Building, training and deploying object detection models are getting more and more easy and fast with the help of this TensorFlow Object Detection API. Above all, the API contains high accuracy and speed pre-trained object detection models that pre-trained on COCO and the Open Images dataset.

While, YOLOv2[1] according to their website, is the updated version of the original YOLO and is state-of-the-art in PASCAL VOC [13] and COCO[15] detection challenges. "It can achieve a real-time object detection"[18]. In VOC2007 challenge, YOLOv2 achieved 76.8 mAP while performs a speed of 67 FPS. Faster and more accurate than Faster RCNN with ResNet and SSD, which are the top methods in object detection[1].

Although both of these two object detection systems are making build, train and deploy object detection models easier and faster, and they are both widely used in the industry. But, hardly had they been used to train on a novel dataset and then compare the performance of new trained object detectors in terms of accuracy and speed. Our research result will be very meaningful and helpful to people who are considering which object detection system should they choose.

1.1 Objective

Our objectives for this project are:

- Train the KITTI Object Detection Evaluation 2012[17] dataset on the pre-trained model provided by TensorFlow object detection API and YOLOv2 accordingly to get two new object detectors.
- Compare the performance of these two new object detectors on the KITTI test dataset, evaluate the result based on the speed and accuracy and infer which system is better in the context of autonomous driving.

1.2 Methodology

Our project use quantitative research approach, more specifically, the experimental research method, which we conduct the real-time object detection test on two different object detectors developed from two different systems accordingly. We collect the performance data and did a comparison study so that we can have some conclusions. It's more about a positivist deductive approach. We first train our object detectors on Tensorflow Object Detection API and YOLOv2 based on the KITTI 2D-object benchmark dataset to get two object detectors and then run the detector on corresponding platform to test the detector performance.

1.3 Document structure

This paper is structured as follows: The introduction section is about the objectives of this research project and the methodology for the research. In the background section, we give a brief introduction to objection detection and its networks, the public datasets, the measure metric for performance and the two frameworks we use. The experiment section goes into more detail about how the experiments were performed and

how to generate the result data. The result section will cover the result data comparison between YOLOv2 and Google TensorFlow Object Detection. In the end, the discussion section will discuss the result and the setting of hyperparameters. Finally, the conclusion section will summarize the research this paper presented and some suggestions for future work.

2 Background

In this section, we will give a brief introduction to objection detection and its networks, the public datasets, the measure metric for performance and the two object detection develop platform we use.

2.1 Object Detection

Object detection is subfield of computer vision. It can identify and locate the instance of defined classes in image or video. Objection detection is widely researched and applied in various fields such as face detection to determine the identity of a person and car detection in autonomous driving. For instance, in the Figure 1 we can see this street view from the image on the left and the object detector gets the result and we can see the bounding box around the detected objects.

Figure 1: The image with and without bounding box

2.1.1 Deep Learning in Object Detection

Deep learning is the state of art in object detection, in this section, we will go over the popular deep learning models applied to object detection in the sequence as they evolved.

- OverFeat [2]
  OverFeat was published in 2013 at NYU, when deep learning is used in object detection, OverFeat was adopted as one of the first models. OverFeat use multi-scale and sliding window over the CNN layers. OverFeat also introduced a new deep learning approach which is learn to predict object boundaries.

- R-CNN[3]
  R-CNN (Region-based Convolutional Neural Network) was published in Nov 2013 at UC Berkeley. It improves the mAP (mean average precision) by more than 30% compared to the previous best result on VOC 2012. The working procedure of R-CNN is: (1) Extract the possible objects using the selective search method, this is called the region proposal. (2) Convolute over the region proposals. (3) Use SVM to classify these CNN outputs.

- Fast R-CNN[4]
  Fast R-CNN was published in 2015 at Microsoft. One year after the publish of R-CNN, Ross Girshick

published the Fast R-CNN. Based on R-CNN, Fast R-CNN also uses the selective search method to extract possible objects, but Fast R-CNN improved its detection speed while increase the detection accuracy through several innovative improvements. Fast R-CNN convolute over the entire image, pool over the feature map using the Region of Interest (RoI).

- YOLO[5]
  YOLO was published short after Fast R-CNN by Joseph Redmon. YOLO is short for You Only Look Once, it proposed a simple Convolutional Neural Network which performs very well on both accuracy and speed. YOLO model can process the images at a speed of 45 f/s, which is a real-time speed. The smaller version of the model, Fast YOLO, can achieve 155 f/s while can still achieve double accuracy of other real-time object detectors.

- Faster R-CNN[6]
  Faster R-CNN is the 3rd generation of R-CNN. It introduced the Region Proposal Network (RPN) to replace the region proposal algorithm, which is the bottleneck in Fast R-CNN network, and end-to-end training can be realized. RPN is used to output object detections based on confidence score. Classification is realized by RoI Pooling and full connected layer.

- SSD [7]
  SSD (Single Shot Multi-box Detector) was published in Dec 2015. It achieved better accuracy and speed by discretizing the output space of bounding boxes and combining predictions from multiple sized convolutional feature maps. We will talk more detail about SSD at chapter 2.1.1.1.

- R-FCN [8]
  R-FCN (Region-based, Fully Convolutional Networks) was published in June 2016. The region-based detection of R-FCN is based on the convolutional of the entire image, in order to detect based on region, R-FCN proposes a novel concept of position-sensitive score maps. R-FCN takes the architecture of Faster R-CNN while use convolutional networks only.

- YOLOv2[1]
  YOLOv2 was published in Dec 2016. It was built upon the Darknet-19, a deep neural model. It outperforming many state of art methods like Faster RCNN with ResNet and SSD in detector accuracy and performing much faster. And it was well designed to tackle real-time object detection problems.

2.1.1.1 SSD

In this section,we will describe more detail about the SSD[7] framework as one of our object detector is trained based on SSD model.

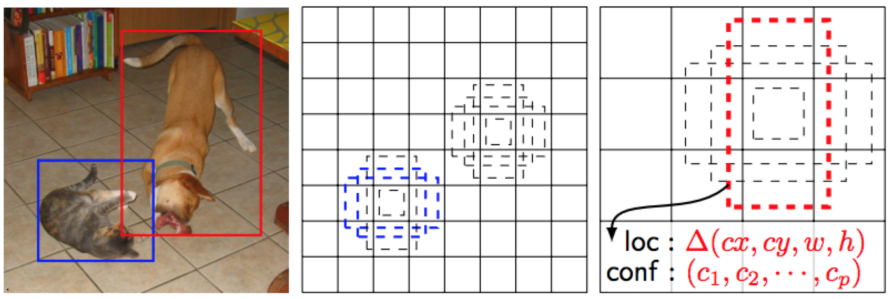

Figure 2: SSD as presented in[7]. Reprinted with permission from Wei Liu

To illustrate the framework of SSD, we need to know 2 concepts first, the feature map cell and the default box. Feature maps are of shape scale 8 x 8 or 4 x 4. In figure 2, (b) is a feature map of scale 8 x 8 and (c) is a feature map of scale 4 x 4. By dividing the feature map into 8 x 8 or 4 x 4 pieces, we got the

feature map cell. Each feature map cell is a square in figure 2 (b) and (c). Default boxes are a series of fixed size boxes located on the feature map cells, the dashed boxes in figure 2 (b) and (c) are default boxes.

SSD is based on the feed-forward CNN, which will produce a series of fixed-size bounding boxes and the possibility of each box containing a specific object instance, which we call it score. And then a Non-maximum suppression is used to finish the predictions. The first part of the SSD model, which is called base network, is a standard architecture for image classification. In addition to this, SSD added some more functional neuron network layers. These layers enhanced the object detection with the following functions:

- Multi-scale feature maps for detection
  Followed by the base network are a series of CNN layers, the size of these CNN layers is in descending order progressively, and can make predictions under multi scales.

- Convolutional predictors for detection
  For each extra feature layers, it can produce a series of fixed sized predictions by using a series of convolutional filters. As shown in figure 3, for a feature layer of size x * y and channel z, the prediction can be achieved by using a 3 * 3 * z convolutional filter, the predictions can be a confidence score of belongs to some object class or the shape offsets compared to the default box coordinate. This process is applied at every location of the x * y feature layer to produce the prediction.

- Default boxes and aspect ratios
  SSD predict a number of n boxes in the feature map. For each box, SSD will predict the confidence score of this object belonging to C classes, and will also predict the 4 shape offsets compared to the default box coordinate. We need (C+4) * n filters for each of the feature map cell in the feature map. For a feature map of size X * Y, we will have an output of X * Y * n * (C+4).

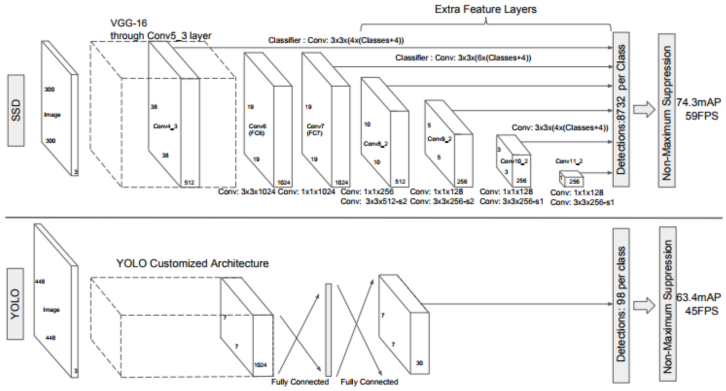

Figure 3: SSD vs YOLO, as presented in [7]. Reprinted with permission from Wei Liu

2.1.1.2 Pre-Trained Models

In deep CNN, to train the same CNN on different image datasets, the weights of the first few CNN layers are almost the same, even though this image dataset is used to train on CNN for distinguishing some specific categories, while the other image dataset is used to train on CNN for distinguishing another set of object categories[9]. Based on this intuition, when we want to train a CNN on a novel dataset, we don't have to train everything from scratch because it will take too much time. On the other hand, we can use some pre-trained models, in our case, we use pre-trained models provided by Google and YOLO.

Google provided many trainable object detection models[19], these models have already been trained on the COCO dataset and Open Images dataset [21]. Different models have different speed and accuracy trade-off[10]. In our case, in the context of autonomous driving, we choose model ssd-inception-v2-coco, as it can perform an adequate accuracy at a high speed.

For YOLOv2, we just use the original YOLOv2 pre-trained model[18] which was trained on the PASCAL VOC 2007 and 2012 datasets.

2.2 Public datasets

There are many public dataset available on the Internet for object detection. Since we choose the dataset for real-time object detection for autonomous driving, so we need to consider the object categories, that's the reason why we choose KITTI dataset. Along with the images, we need to have the full annotation for the training set and validation set within the dataset.

2.2.1 Ground Truth

Ground truth is the observed fact describing the position, size and class of objects in the image. It's used in object detection to compare with the result inferred by object detector, so that the deep CNN can learn the real fact. There are normally 5 values in one ground truth item, 1 value indicate the object class, and another 4 coordinate value to indicate a bounding box which surround the object.

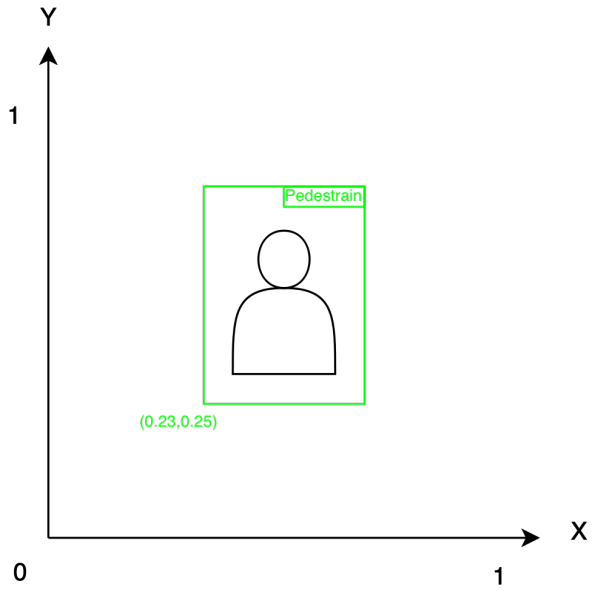

Figure 4: Representation of Bounding Box from Table 1

Class	X	Y	Width	Height
Pedestrian	0.23	0.25	0.2	0.23

Table 1: Table of an example of ground truth

2.2.2 KITTI Benchmark Dataset

"KITTI Vision Benchmark Suite were captured by driving around the mid-size city of Karlsruhe in Germany, in rural areas, and on highways. Up to 15 cars and 30 pedestrians are visible per image."[17] KITTI 2D objects dataset consists of 7481 training images and 7518 testing images with a resolution of 1240 x 375 pixels with 12GB in total. The numbers of classes of ground true in KITTI data set is 9. These classes are included in the following table 2.

DontCare	Pedestrian	Tram
person sitting	Car	Misc
Van	Truck	Cyclist

Table 2: Table of KITTI Dataset Class Categories

Figure 5: The samples from the KITTI benchmark dataset

2.3 Measuring Performance of Object Detection

Measuring metrics are needed for measuring the performance of two trained object detectors. The fps, which is short for frame per second, is introduced to measure the speed of object detector, it can reflect how many images the detector can process per second. To measure the accuracy, the measuring metric adapted in this paper is mean average precision(MAP), which is widely used in modern object detection. It is computed by the average precision(AP) on each class.

2.3.1 Calculate IOU

To understand MAP, we should know that is the true positive in object detection. A decision metric is needed to make the true positive decision, we decided to use Intersection- Over-Union (IOU), which is the most commonly used metric in object detection. The true positive is found when the overlap between the detector and the ground truth is higher than a threshold. A typical threshold value of IOU for the correct decision is 0.5.

$$IOU = \frac{Area - of - overlap}{Area - of - union} \tag{1}$$

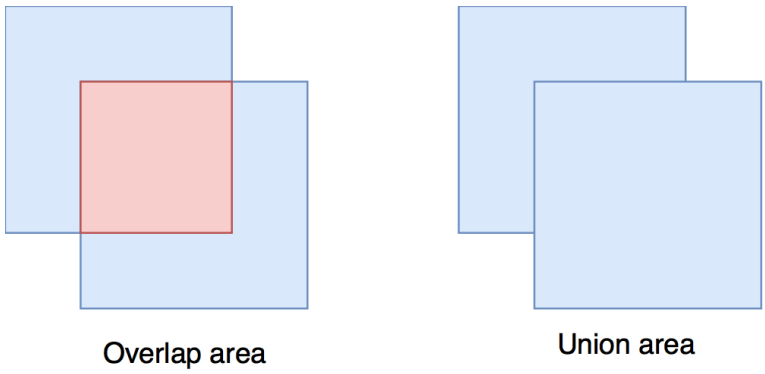

Figure 6: Visual representation of overlap and union areas between detector bounding box and ground truth.

2.3.2 Calculate MAP

A result is considered a true positive if it has IoU has the value higher than a threshold (usually 0.5). The detection result of all test images should be combined to draw a precision/recall curve for each class. ”AP is the area under the curve, computed via the Riemann sum” [11]:

$$AveragePrecision = \sum_{K=1}^{N} P(K)\Delta recall(K) \tag{2}$$

$$mAP = \frac{\sum_{K=1}^{N} AveragePrecision(q)}{Q} \tag{3}$$

The AP measure handles the different threshold values by computing precision and recall values for a high number of different thresholds, and averaging the precision score for each recall value. For Q, it's the number of classes. "N is the total number of the images. And delta r(k) is the change in recall that happened between cutoff k-1 and cutoff k."[20] For KITTI dataset, Q = 9.[13]

$$Precision = \frac{TP}{TP+TF} \tag{4}$$

"Precision is the proportion of all examples above that rank which are from the positive class". [12]

$$Recall = \frac{TP}{TP+FN} \tag{5}$$

"Recall is defined as the proportion of all positive examples ranked above a given rank."[12]

2.4 YOLOv2

YOLO (You Only Look Once), the real-time object detector, now has an improved version, YOLOv2[1]. According to YOLO's website[18] The performance of YOLOv2 improved a lot compared with old YOLO, it outperforms SSD300 on both accuracy and speed. "It is the state-of-the-art in object detection field"[1].

- Better
  YOLOv2 has better accuracy performance by introducing a variety of novel concepts which derived from their past work.

- Faster
  YOLOv2 proposes a new detection framework named Darknet-19, it makes YOLOv2 faster. For more detail description, please check chapter 2.4.1 Architecture.

- Stronger
  YOLOv2 proposes a very novel concept, which use WordTree to solve the mutually exclusive problem, this makes YOLOv2 even stronger.

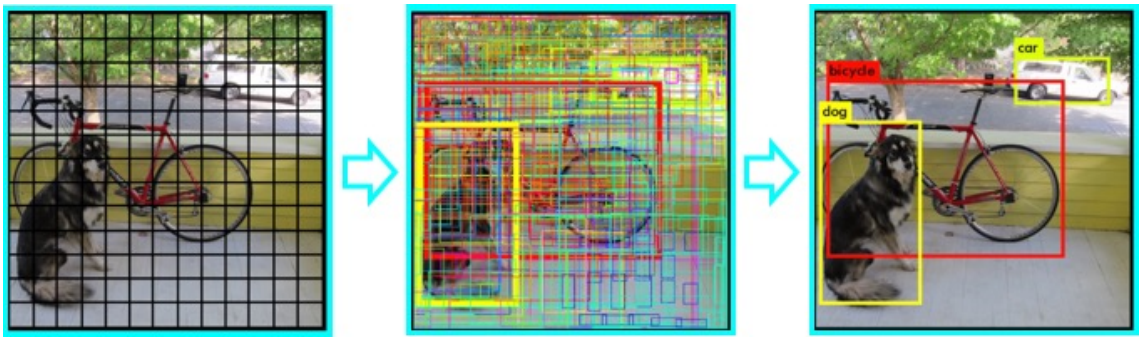

Figure 7: YOLO image processing overview[5]. Reprinted with permission from Joseph Redmon

2.4.1 Architecture

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 8: The architecture of Darknet19, which is the base of YOLOV2. “It has 19 convolutional layers and 5 maxpooling layers”[1]. Reprinted with permission from Joseph Redmon

2.4.2 Implementation

In our experiment, YOLO version 2 is built on Darknet, ”which is an open source neural network framework written in C and CUDA”[18]. It can be installed quickly and easy to be deployed on any platforms like Windows, Mac OS, Linux. Meanwhile, it can be deployed on either CPU or GPU.

2.4.3 YOLOv2 Pre-Trained Model

In order to simplify the process, save time used on training a new model, and also improve the output model’s accuracy and speed, YOLO provides the pre-trained models[18]. The model we adopted in the experiment was trained by YOLOv2[1] for a week with high-performance GPU. We select the original YOLOv2 pre-trained model, yolo-voc.weights (about 193 MB) provided on the YOLO’s website.

2.5 Google TensorFlow Object Detection API

As stated on the Google Tensorflow Object Detection API official website[14], the TensorFlow Object Detection API is an open source object detection development system based on Google’s Tensorflow. Building, training and deploying object detection models are getting more and more easy and fast with the help of this TensorFlow Object Detection API.

Within this object detection development system, Google provide the following toolkits[22]:

- Trainable pre-trained object detection models

Model name	Speed (ms)	COCO mAP
ssd-mobilenet-v1-coco	30	21
ssd-inception-v2-coco	42	24
rfcn-resnet101-coco	92	30
faster-rcnn-resnet101-coco	106	32
faster-rcnn-inception-resnet-v2-atrous-coco	620	37

Table 3: Table of pre-trained object detection models[19]

- Frozen weights
  For all the trainable pre-trained object detection models, Google also provides the frozen weights version, which we can use for off-the-shelf inference.

- Jupyter notebook for off-the-shelf inference.

- Scripts
  Google provides a series of scripts, which we can use for training pipeline configuration, evaluation, data parsing and so on.

Different models have different speed and accuracy trade-off[10]. In our case, in the context of autonomous driving, we choose SSD as our training deep CNN framework, we choose the ssd-inception-v2-coco as our pre-trained model, because we want an adequate accuracy at a high speed.

3 Experiments

3.1 Experiments Environment

We were able to use student edu email to register a 100 dollars AWS coupon. All of our experiments including training and evaluation were run on AWS EC2 p2.xlarge instance, which provide graphics card we need to boost the performance of deep learning algorithm. And we have to install cuDNN v6.0, CUDA® Toolkit 8.0 on the platform to support deep learning platform to use NVIDIA graphics card for training.

Cloud platform Instance type	Amazon EC2 p2.xlarge
OS	Ubuntu 16.04
SSD	30G
Dependencies for Tensorflow	cuDNN v6.0, Python 3.5.2, TensorFlow 1.4 CUDA® Toolkit 8.0
Dependencies for YOLOv2	YOLOv2, CUDA® Toolkit 8.0, cuDNN v6.0

Table 4: Table of experiments environment

3.2 KITTI Dataset

We use the original KITTI Object Detection Evaluation 2012 training dataset as our training and testing dataset, it has in total 7481 images. We divide the 7481 images into two parts, the training dataset and the testing dataset. The first 4981 images are for training. And the last 2500 images are for evaluation. Both of the training dataset and the evaluate dataset contains the KITTI images and labels (ground truth).

3.3 Data Parsing

We have to parse the original KITTI data into the format which YOLOv2 and Google TensorFlow Object Detection API can accept.

3.3.1 YOLOv2

The original label format of KITTI format needs to be converted to YOLOv2 label format which contains the category number and the relative position of the box for each object.You can find an example of the YOLOv2 label format in 2.2.1 the ground truth section.

3.3.2 Google TensorFlow Object Detection API

"Tensorflow Object Detection API reads data using the TFRecord file format."[14] According to the TensorFlow Python API[23], TFRecord store the information in CRC hash format, the example of one record is shown below:

```
uint64 length
uint32 masked_crc32_of_length
byte   data[length]
uint32 masked_crc32_of_data
```

A TFRecord file is formated by connecting all these records together in a specific sequence. We have to convert the original KITTI dataset into the TFRecord format.

3.4 Training

As mentioned before, we use AWS VM (virtual machine) to train the object detectors. We launched two VM, both of them have the same configuration as we stated in the experiment environment chapter. One of the VM is for YOLOv2 training, and another VM is used for the Google Tensorflow Object Detection API training. We use the same KITTI training dataset for training. For the deep CNN training hyperparameters, we also configure the same for both of these two object detection systems.

Batch Size	1
Learning Rate	0.0001
Training Steps	100,000
Use Dropout	FALSE
IOU Threshold	0.5

Table 5: Table of Hyperparameter Configuration

- We set the batch size to 1 because for each VM, we only have one NVIDIA K80 GPU, we will have "Out OF Memory" error if we configure the batch size larger than 1.
- As we use the pre-trained model, the CNN weights have already converged in some degree, so we set small value for the learning rate, 0.0001.
- Due to the limitation of our budget, we don't have enough coupon to run a too long training, 100K steps are the maximum steps we can train. It takes 12 hours to train 100K steps on the Google Tensorflow Object Detection system, and 8 hours to train on the YOLOv2 system.

3.5 Evaluation

We evaluate the two trained object detectors base on their performance of detection speed and detection accuracy on the same evaluation dataset.

3.5.1 Accuracy

To evaluate the detection accuracy, we use mAP. The method to calculate mAP is based on PASCAL Visual Object Classes (VOC) metric, which was mentioned in 2.3.2 calculate MAP section.

3.5.2 Speed

To evaluate the detection speed of the detector, we use frames per second (fps). We run our detectors over 1,000 images to calculate the average detecting time and calculate the speed of two detectors.

4 Results

The experiment results are shown in the table below.

Object Detection System	Google TensorFlow Object Detection API	YOLOv2
Accuracy(mAP)	56	13
Speed(fps)	2	58

Table 6: Table of Results

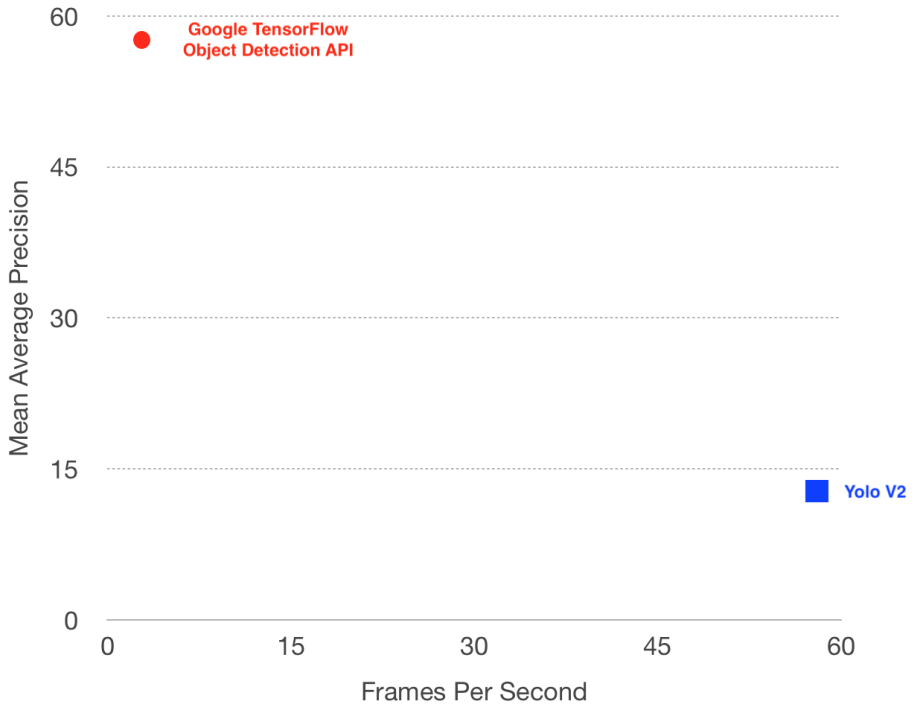

Figure 9: The comparison between YOLOv2 and Google TensorFlow Object Detection API

As you can see from the diagram above, it's very obvious that Google TensorFlow Object Detection API performs better at accuracy, while YOLOv2 performs better at speed. It seems that, choosing between these two object detection system is a trade-off between speed and accuracy.

- We think the reason why Google performs better at accuracy is because of the pre-trained model. In our experiment, we choose ssd-inception-v2-coco pre-trained model, which has been trained on the COCO dataset. The COCO dataset already covered a large variety of object instances and object

classes. While for YOLOv2, the pre-trained model is trained on the VOC2007+2012 dataset, which has a very limited object instances and object classes. So, in our case, Google will be able to perform better at accuracy even though we did't train it for too long.

- We think the reason why YOLOv2 performs better at speed is because of the architecture of YOLO and Darknet. The architecture of YOLO is simper than SSD and can performs faster than SSD.

4.1 Detection examples

Figure 10: Detection result example from Google TensorFlow Object Detection API

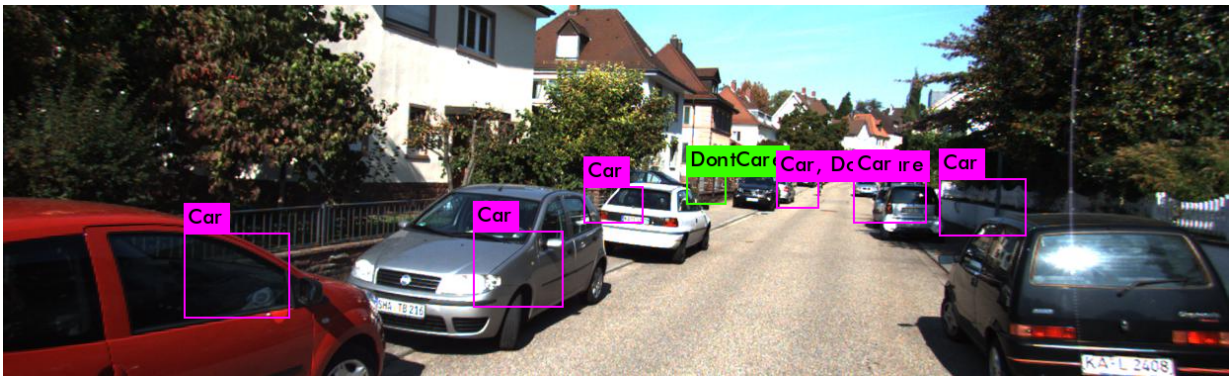

Figure 11: Detection result example from YOLOv2.

The bounding box of Google object detector can surround the object better than the bounding box of YOLOv2, which means Google performs a higher accuracy.

5 Conclusion

In our comparative study, with the specific hyperparameter setting and pre-trained model selection in our experiment, we found that, in the context of autonomous driving, where accuracy and speed are the two most important factors to measure the performance of object detectors:

- The Google TensorFlow Object Detection API developed object detector perform better at accuracy.
- The YOLOv2 developed object detector perform better at speed.

6 Discussion and Future work

Note that, for Deep Convolutional Neural Network, we train the dataset iteratively to find the global optima of the model. But, during the training process, the Deep CNN and global optima is sensitive to hyperparameters, example of hyperparameters are: training steps, pre-trained model selection, learning rate, drop-off, batch size, training dataset and so on. But, in our case, we set the hyperparameters to be the same for both of the object detection systems, and all the hyperparameter value are fixed. This unique hyperparameter setting will lead to unique result of the speed and accuracy performance, while the other combination of hyperparameters may lead to another unique result, which means, our conclusion for this comparative study may be totally different. We should use hyperparameter tuning methods like grid search and random search to find the best hyperparameters for two object detectors correspondingly. But, due to the fact that we use the p2-xlarge instance of AWS cloud services in our experiment, which is unaffordable for us to find the global optima of each object detectors.

In future work, we can try to find more GPU compute resources and perform hyperparameters tuning techniques to obtain a better model for autonomous driving and have the deeper understanding of the trade-off between accuracy and speed. Since Google TensorFlow Object Detection API developed object detector has the advantage of accuracy, and YOLOv2 developed object detector has the advantage of speed. We can try to develop a technique to combine these two models in the same object detection system to achieve a better result.

7 Acknowledgements

We thank Professor Gerald Q. ”Chip” Maguire Jr. for his invaluable help on this report.

References

[1] J. Redmon, A. Farhadi. *"YOLO9000: Better, Faster, Stronger"*. arXiv:1612.08242 [cs], Dec. 2016. [Online]. Available:https://arxiv.org/pdf/1612.08242.pdf.

[2] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. *"OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks"*. arXiv:1312.6229 [cs], Dec. 2013. [Online]. Available:https://arxiv.org/pdf/1312.6229.pdf.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. *"Rich feature hierarchies for accurate object detection and semantic segmentation"*. arXiv:1311.2524 [cs], Nov. 2013. [Online]. Available:https://arxiv.org/pdf/1311.2524.pdf.

[4] R. Girshick. *"Fast r-cnn"*. in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448. [Online]. Available:https://arxiv.org/pdf/1506.01497.pdf.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *"You Only Look Once: Unified, Real-Time Object Detection"*. arXiv:1506.02640 [cs], Jun. 2015. [Online]. https://arxiv.org/pdf/1506.02640.pdf.

[6] S. Ren, K. He, R. Girshick, and J. Sun, *"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"* . in Proceedings of the IEEE international conference on computer vision, Jun. 2015, pp. 1440–1448. [Online]. Available:https://arxiv.org/pdf/1506.01497.pdf.

[7] W. Liu et al. *"SSD: Single Shot MultiBox Detector"* . arXiv:1512.02325 [cs], vol. 9905, pp. 21–37, 2016. [Online]. Available:https://arxiv.org/pdf/1512.02325.pdf.

[8] J.Dai, Y. Li, K. He, and J. Sun. *"R-FCN: Object Detection via Region-based Fully Convolutional Networks"*. arXiv:1605.06409 [cs], May 2016. [Online]. Available:https://arxiv.org/pdf/1605.06409.pdf.

[9] S. Jensen and A. L. Selvik. *"Using 3D Graphics to Train Object Detection Systems,"*. Master's Thesis, NTNU, 2016. [Online]. Available:https://brage.bibsys.no/xmlui/handle/11250/2418432.

[10] J. Huang et al. *"Speed/accuracy trade-offs for modern convolutional object detectors"* . arXiv:1611.10012 [cs], Nov. 2016. [Online]. Available:https://arxiv.org/pdf/1611.10012.pdf.

[11] K. Buhler, J. Lambert, and M. Vilim. *"Real-time Object Tracking in Video"* . CS 229 Course Project, Departments of Computer Science and Electrical Engineering, Stanford University [Online]. Available:https://pdfs.semanticscholar.org/989c/7cdafa9b90ab2ea0a9d8fa60634cc698f174.pdf.

[12] Q. Li et al. *"How to calculate average precision."* . [Online]. Available: https://docs.microsoft.com/en-us/cognitive-toolkit/object-detection-using-fast-r-cnn [Accessed: 27- Oct- 2017]

[13] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. *"The pascal visual object classes challenge: A retrospective"* . International journal of computer vision, vol. 111, no. 1, pp. 98–136, 2015. [Online]. Available:https://pdfs.semanticscholar.org/cf9d/04c6c0ec9bde9da233eb352612f1bda643fe.pdf.

[14] J. Huang et al. *"Tensorflow Object Detection API"* . 2017. [Online]. Available:https://goo.gl/he1XoM [Accessed: 11- Oct- 2017]

[15] T.Y Lin. et al. *"Coco dateset challenge"* . 2017. [Online]. Available:http://cocodataset.org [Accessed: 11- Oct- 2017]

[16] J. Huang et al. *"Nestcam"* . 2017. [Online]. Available:https://nest.com/cameras/nest-aware/ https://nest.com/cameras/nest-aware/ [Accessed: 12- Oct- 2017]

[17] A. Geiger, P. Lenz, C. Stiller, R. Urtasun. *"KITTI dataset"* . [Online]. Available:http://www.cvlibs.net/datasets/kitti [Accessed: 20- Oct- 2017]

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *"Yolo official website"* . [Online]. Available:https://pjreddie.com/darknet/yolo/ [Accessed: 20- Oct- 2017]

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *"Tensorflow detection model zoo"* . [Online]. Available:https://goo.gl/GhRtps [Accessed: 20- Oct- 2017]

[20] *"How to calculate average precision."* . [Online]. Available:https://sanchom.wordpress.com/tag/average-precision [Accessed: 27- Oct- 2017]

[21] K. Ivan et al. *"Open Images Dataset V3"* . [Online]. Available:https://github.com/openimages/dataset [Accessed: 21- Oct- 2017]

[22] J.Huang, V. Rathod. *"Google research blog"* . [Online]. Available:https://goo.gl/Bnt1o4 [Accessed: 14- Oct- 2017]

[23] *"Tensorflow documentation"* . [Online]. Available:https://www.tensorflow.org/ [Accessed: 18- Oct- 2017]