Analysing Object Detection Algorithms in Auto Driving

Zixuan Pan Zhengyu Cai Yifeng He

Abstract

The idea of self-driving has arouse interest from the public for a long time. In order to reach this objective, many computer vision researches have been implemented to improve the perception of the vehicle. In this research, we start with analyse the architecture difference between Faster RCNN and YOLO. The design difference in this two models illustrates that one stage object detection algorithm have less processing time. This argument is validated during implementing YOLOv3 and Faster-RCNN with the auto-driving dataset - KITTI. We also find out that when limited training time provided, YOLOv3 outperforms Faster-RNN in mean of average precision in this scenario.

1 Introduction

Object detection has become an important aspect in computer vision domain. With a clear identification of the object in the image, machines can capture the information hidden behind the scene and develop more sophisticated functions like motion tracking or semantic synthesis. By implementing deep neural network as the backbone, researchers has obtained significant success in applying object detection methods in many scenarios. Autonomous driving, which has drawn a great amount of attention these day, is one of these scenarios that requires the support of advanced object detection algorithms. In this research, we are going to analyse two popular object detection method, YOLO [1] and Faster R-CNN [8], and evaluate the performance of these methods on identifying objects in traffic scenes. Our work are summarized as follows:

- Implement YOLOv3 and Faster-RCNN into the KITTI dataset. Our code for YOLOv3 is available at `https://github.com/MuMuPatrick/kitti_yolo` and the code for Faster-RCNN is at `https://github.com/jasonhaha25/Faster-RCNN-TensorFlow-Python3`
- Analyse the performance of these two architecture on KITTI by comparing mean of average precision and response time.

2 Related Work

Many researches have devoted in pushing performance of state-of-the-art in object detection. Inspired by the success of the AlexNet[4] in image classification, Sermanet et al.[9] proves the high efficiency of convolutional network (CNN) for object detection. In this approach, the author uses multi-scale sliding window on the image and derive the corresponding boundary box and classification for each window. These boundary boxes are combined to give the optimal solution.

In the same year, another approach from Girshick et al. [3] uses Selective Search algorithm to generate a region proposal and conduct convolution on the selected region. These proposed boundaries will be refined by regression after scoring each boundary box with class-specific detection Supported Vector Machine. Several improvements have been made to leverage the performance of this algorithm. Fast RCNN[2] improves the computation cost of the original RCNN by implement convolution on the input image and clipping the corresponding feature map of the selected regions. Faster RCNN[8] replaces the original Selective Search algorithm with a CNN named as Region Proposal Network. This network will be trained with the ground truth from the labelled picture to generate better region proposal.

Unlike those two-stage algorithms which generate region proposal before convolution, the well-known YOLO[5, 6, 7, 1] treats the operates in a one-stage manner which only implements one CNN in the architecture. It divides the input image into grid where each grid cell predicts multiple boundary boxes, the corresponding confidence level and the class probability. At test time, the class probability conditioned to the grid cell will be multiplied with the box confidence level to derive the class-specific confidence score for each boundary box.

3 Methodology

In this research we will analysis the performance of two object detection architectures, YOLO and Faster R-CNN. As two algorithms have outstanding performance in general-purpose object detection, it is valuable to find out whether these advancement are still preserved under self-driving scenario. Also, as depicted in section 2, the difference between these two architecture might affect the final performance on the new database. The comparison between YOLO and Faster R-CNN is worthwhile in understanding the mechanism behind these two models.

3.1 Faster RCNN

In object detection, the naive approach aims to select different regions of interest from the original image and use a CNN to classify them. R-CNN, short for region-based convolutional neural networks, aims to eliminate the issue of having too many regions of interest and save computation time. It uses so called Selective Search algorithm [10] to generate around 2000 region proposals. These are then warped and fed into a CNN network for feature extraction. Lastly, the results shall be passed into an SVM for final classification.

However, the network proposed in R-CNN still takes a long time to train since one image detection requires around 2000 region proposals. The same author later proposed "Fast R-CNN" which aims to solve some of the drawbacks of its predecessor. In his paper, he introduced a "ROI Pooling" layer that extracts feature vectors from all region proposals. Now instead of having to do the convolution step independently for every single region proposal, the feature extraction step is done all at once. Finally we are ready to predict the classes using a softmax layer.

Although Fast R-CNN is significantly faster than vanilla R-CNN, both approaches suffer due to heavy usage of Selective Search. The latter takes a very long time to compute and can not be customized which makes (Fast) R-CNN impossible for real time object detection and may be inaccurate in detecting all target objects. The author of Faster R-CNN introduced the Region Proposal Network in place of Selective Search. It aims to predict region proposals and is trained from the IoU score against the target boxes. Since this trainable network uses the same convolutional layer from Fast R-CNN, the drawbacks from Selective Search can be perfectly eliminated. Its customisable nature also allows the model to classify objects much better by having more accurate region proposals.

3.2 You Only Look Once (YOLO)

In contrast to R-CNN, the YOLO algorithm divides the input image into an SxS grid. Each grid cell will predict a number of bounding boxes along with a confidence score using a single neural network. It uses Darknet framework and is very popular in real time object detection. However, one of the major drawback of this approach is localization of boxes.

YOLO v2, the upgraded version, takes Darknet-19 as its backbone, combined with the use of batch normalization to stabilize the network, higher resolution classifier to improve mean of average precision. Another important upgrade is the use of anchor boxes instead of bounding boxes. By having this addition, one may predict all objects at once, without the need to scan the image with a sliding window, making real time object classification possible. Comparing with Faster-RCNN, this implementation in object detection will ensure YOLO and its variant perform with less processing time.

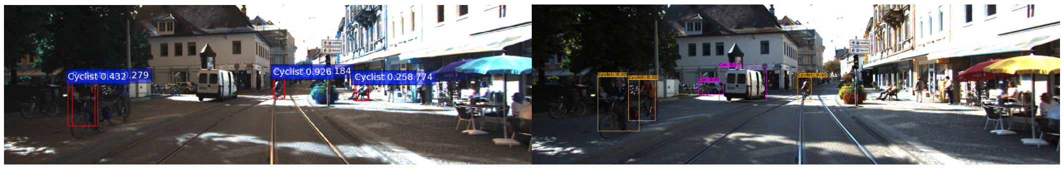

Figure 1: One prediction result that YOLO outperforms Faster-RCNN

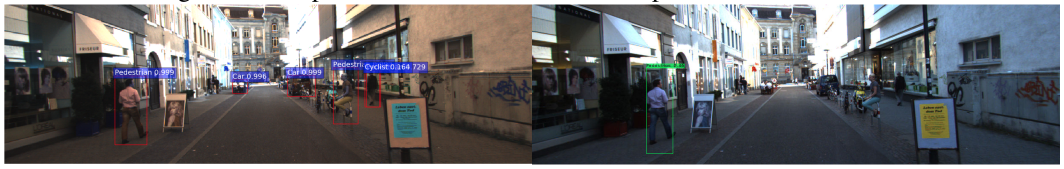

Figure 2: One prediction result that Faster-RCNN outperforms YOLO

The improved verision, YOLO v3, uses Darknet-53, meaning that it has significantly more (106) layers than its YOLO v2. It predicts more boxes and takes logistic regression (instead of softmax) as its loss function. This also means that YOLO v3 is able to handle multilabel classification. Moreover, Feature Pyramid Networks are introduced in order to have a better precision at different image scales.

4 Experiment

We implemented YOLOv3 and faster-RCNN into the KITTI 2D object detection benchmark. The original training dataset consists of 7481 images with 8 categories and the trained architecture is tested on the 7581 test images. Due to the limitation of time and compute resources, in this experiment, two architectures are trained on the first 1000 picture of the original training set. We also combined the original 8 categories into 3 main categories (Cars, Pedestrian and Cyclists) to narrow down the size of the model. To ensure a reasonable comparison, we limit the training process of both architecture to 10000 iterations with a batch size of 64. As the labels for the original test dataset are not accessible, 500 images are picked from the rest of the KITTI training dataset and these images are used as the test dataset. The YOLOv3 is initialized with the pre-trained backbone darknet53 and the Faster-RCNN is intilized with the pre-trained VGG16 network.

Regarding the evaluation, all the prediction above 0.7 threshold are treated as valid predictions. We compare both the mean of average precision (mAP) and per-class average precision between YOLOv3 and Faster R-CNN in test dataset to evaluate their object detection performance on the driving scenario. As the processing time is critical in auto-driving, We also calculate the average response time of these two architectures.

5 Result

It took us around 23 hours for YOLOv3 model, while only around 2 hours for Faster-RCNN to train 10000 iterations with batch size 64. From the loss/accuracy generated on the training set, YOLOv3 has an accuracy of around 80% and loss function doesn't change too much after 8000 iterations (toggles around 1). Faster-RCNN has a fairly low accuracy with rapid change on loss function, as we then tell it's not fully trained. Testing both models we trained, YOLOv3 performs better than Faster-RCNN with better mAP (see Table 1), meaning the trained YOLOv3 model identifies more positive cases than Faster-RCNN.

Figure 1 shows an test example of the trained model where YOLOv3 outperformed Faster-FCNN. Faster-RCNN failes to detect the car and other objects. It tends to predict a *Cyclist* + *Pedestrian* combination for one cyclist shown on image with low confidence. Through all test cases, we also noticed that it's hard for Fast-RCNN to detect back-viewed cars or busses with some blocking. With all combined, Faster-RCNN has very low mAP, both on Training and Testing data set. While YOLOv3 has a better performance, it is not sensitive to pedestrians or cyclists. As shown in Figure 2 YOLOv3 missed out many pedestrians in the image, but Faster RCNN is capable to identify most of them with

Model	mAP	Cars	Cyclist	Pedestrian	Avg Resp Time
YOLOv3-train	0.84	0.95	0.85	0.72	27ms
Faster RCNN (@0.7)	0.32	0.54	0.07	0.35	96ms
Faster RCNN (@0.1)	0.37	0.56	0.17	0.37	97ms
YOLOv3 (@0.7)	0.39	0.68	0.26	0.23	28 ms
YOLOv3 (@0.1)	0.59	0.86	0.47	0.44	28 ms

Table 1: Summary of results for customized KITTI dataset. Included are the mean of average precision (mAP), per-class average precision (Cars, Cyclists and Pedestrian) and the average response time (Avg Resp Time). Besides comparing the result with the 0.7 confidence threshold (@0.7), the results of the model under 0.1 confidence threshold (@0.1) are also compared

high confidence, which indeed Faster RCNN has a better performance identifying pedestrians given in Table 1.

In Table 1, another interesting inspection is obtained from the comparison between YOLO's train and test performance. The average precision in *Cyclist* and *Pedestrian* are nearly halved, but *Cars* only decrease 20 percent. We argue that this decrease is resulted from the limited train images in the training dataset. The lack of *Cyclist* and *Pedestrian* example in the train images restricts the generalization of the trained model in these two classes. On the other hand, *Cars* are the most common objects appear in normal driving scenes. The sufficient examples in this class stabilize the average precision. This imbalance of the train dataset also result in the noticeable change when the threshold increase from 0.1 to 0.7 as most confidence level for pedestrians/cyclists given by YOLOv3 are low.

Analysing *Avg Response Time*, Faster RCNN is almost 4 times slower than YOLOv3. Faster RCNN takes around 100ms per frame on average,making it impossible for Faster RCNN to create a 15fps real time analysis. YOLOv3 on the other hand has a significant time advantage, and is capable to use on real time analysis.

6 Conclusion

In this project, we implement two well-known object detection algorithms - YOLO and Faster-RCNN in the KITTI dataset. After 10000 iterations of training with 1000 street images, YOLOv3 performs better than Fast-RCNN in mean of average precision and average response time. The result in this experiment illustrates that when the architectures share the same early stage training with limited training resource, YOLOv3 is a faster learner comparing with Faster-RCNN and is preferred to be used to build the real-time street object detecting analysis.

7 Contribution

Zhengyu Cai researched relevant papers and gave critical insights which became the very backbone of this analysis. Zhengyu carefully reviewed each algorithm's implementation, and provided detailed mathematical background and reasoning that are presented in this paper. He drafted and lead in the field of methodology.

Zixuan Pan was in charge of implementing YOLOv3 on the KITTI dataset. He also conducted the performance analysis of the performance of YOLO in this dataset. Besides the work on development experiment, Zixuan Pan prepared the project proposal and drafted the abstract, introduction, related work, result and conclusion of the final project report.

YH was responsible of implementing Faster-RCNN on KITTI dataset. He also design the configurations of the comparing experiment, and analyzed the performance of Faster-RCNN. He drafted the results and the final conclusion of the project report.

References

[1] A. Bochkovskiy, C. Wang, and H. M. Liao. Yolov4: Optimal speed and accuracy of object detection. CoRR, abs/2004.10934, 2020.

[2] R. Girshick. Fast r-cnn. pages 1440–1448, 2015.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. Commun. ACM, 60(6):84–90, may 2017.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. 2016.

[6] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. CoRR, abs/1612.08242, 2016.

[7] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. CoRR, abs/1804.02767, 2018.

[8] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6):1137–1149, 2017.

[9] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2014.

[10] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. International Journal of Computer Vision, 2013.