Which CNN model is better for accuracy and/or speed when detecting objects for autonomous vehicles using a 2D camera?

Peter Geras, Minrui Li
New York University
psg2@nyu.edu, ml7136@nyu.edu

Background Introduction

Once only known in science fiction, autonomous vehicles are now being deployed onto the streets of the United States. Tesla has half a million vehicles on the road with Level 2 vehicle autonomy and Waymo has a fleet of taxis in Phoenix, Arizona with Level 4 vehicle autonomy. Autonomous vehicles rely on detecting objects such as other vehicles, pedestrians, traffic signs, etc. with cameras, radars, and LiDAR (except Tesla). The issue with autonomous vehicles is creating an excellent deep learning model that is fast at detecting objects but also accurate in real time. In many cases, this is a life-or-death scenario, particularly for Tesla. Tesla's Autopilot feature in their vehicles has become more scrutinized as there has been a few accidents and deaths when the Autopilot is turned on and the driver is not focused on the road.

Problem Statement

The ability to automatically identify objects is a vital task for autonomous driving. In addition to requiring high accuracy (like humans), object detection for autonomous driving also requires very short computation time. This ensures the vehicle can promptly react in a split second to keep passengers and pedestrians safe. Deep learning models must be trained on large datasets of 2D images and 3D data to improve the accuracy of object detection and compute time. Even with large datasets, autonomous vehicles are not ready for commercial use and researchers are still improving on their deep learning models. It could take years until the models are fully mature to be used on public roads.

Dataset

Waymo Open Source Dataset is an autonomous driving dataset, comprising of images recorded by multiple high-resolution cameras and sensor readings from LiDAR scanners. The data has been recorded with large geographical coverage within multiple cities in various weather conditions.

Since our project focuses on 2D detection, we will only use the camera images. Waymo dataset provides 2D bounding box labels (vehicles, pedestrians, cyclists) in the camera images. The camera labels are tight-fitting, axis-aligned 2D bounding boxes with globally unique tracking IDs. The label is encoded as (cx, cy, l, w) with a unique tracking ID, where cx and cy represent the center pixel of the box, l represents the length of the box along the horizontal (x) axis in the image frame, and w represents the width of the box along the vertical (y) axis in the image frame.

Below are the images from Waymo with and without bounding boxes:

Besides, we also found some other open datasets such as

KITTI and BDD (Berkley DeepDrive), we will utilize them if needed.

Faster R-CNN

Faster R-CNN (Ren et al. 2015) is a very popular model two-stage object detection field in recent years. It consists of the Region Proposal Network (RPN) as a region proposal algorithm and the Fast R-CNN as a detector network. Unlike R-CNN and Fast R-CNN, Faster R-CNN first goes through a convolutional network. Afterwards, another network predicts the proposes regions then compares the predicted regions with the Region of Interest (RoI) pooling. Using both the region proposal network and the RoI, the regions are then changed to classify the regions in the image and creates a bounding box.

Using the COCO dataset that we used in classes we can see that the larger the threshold the less and less the model can recognize objects. A low threshold means that model can recognized multiple object even with a low error rate. This is key when trying to recognize images in the distance to predict where a person or vehicle may go.

Figure 1: Threshold: 0.5

Figure 2: Threshold: 0.6

Figure 3: Threshold: 0.7

Figure 4: Threshold: 0.8

Figure 5: Threshold: 0.9

As the threshold is increases the less and less the

YOLOv5

YOLO (You Only Look Once) (Redmon et al. 2016) is a one-stage object detection algorithm much different from the region-based algorithms. In YOLOv5 a single convolutional network predicts the bounding boxes and the class probabilities for these boxes so it normally has higher speed, but the biggest drawback is that YOLOv5 always struggles with small objects within the image.

In this part we will use YOLOv5. Here is the architecture.

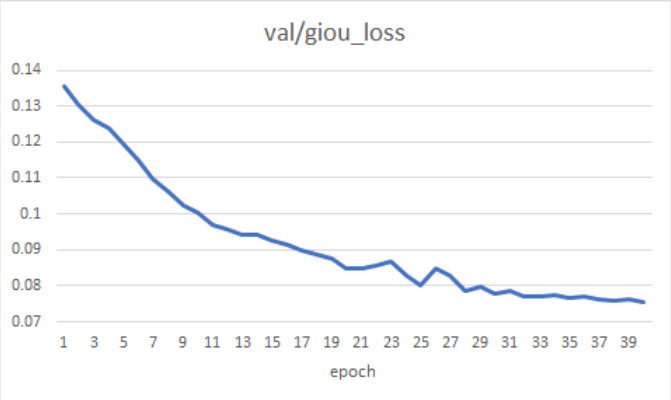
Data Preprocessing

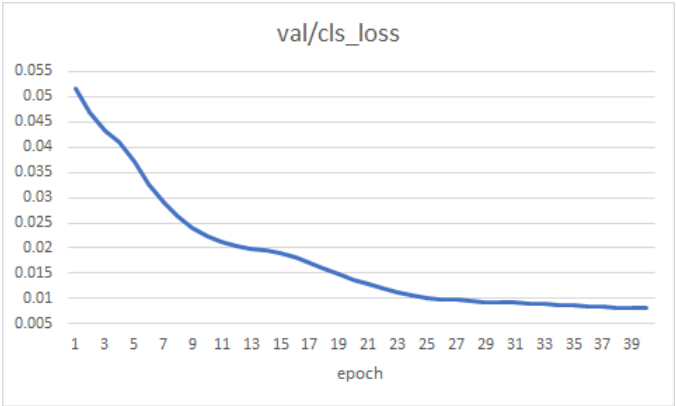
Before working on this part, we have realized that the Waymo dataset is too huge. It contains 25TB data in total, so it is not realistic for us to use the whole dataset. To begin, we decided to download one single frame and work on it first.

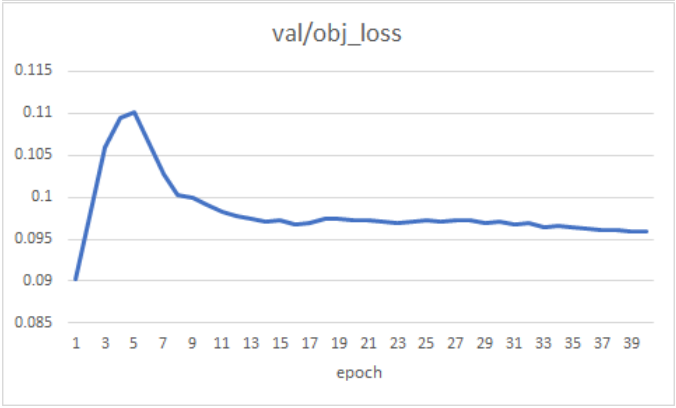
Besides, all the data was encoded into 'tfrecord' format, which can not be used by YOLOv5 directly. According to Waymo Open Dataset Toturial, I utilized and modified some code from Waymo_Kitti_Adapter to extract the data and convert them into '.jpg' images and '.txt' files.

Loss Function

The paper about YOLOv5 has not been published yet, so there is no official explanation of the loss function. From previous version of YOLO, we can see that giou_loss measures the errors in the predicted boundary box locations and sizes; cls_loss is classification loss; obj_loss basically depends on whether the object is detected in the box. The final loss function would be the sum of them. Below are the loss function curves of 40 epochs training:







Train on Custom Data

At this stage, we took YOLOv5s, the smallest one among them, as our pretrained model. It contains 270 layers (7033114 parameters). We set the batch size to be 16 and iterate 40 epochs.

Evaluate the Model

Here is what we got after 3, 13, 30 and 40 epochs of training, the threshold is 0.4.

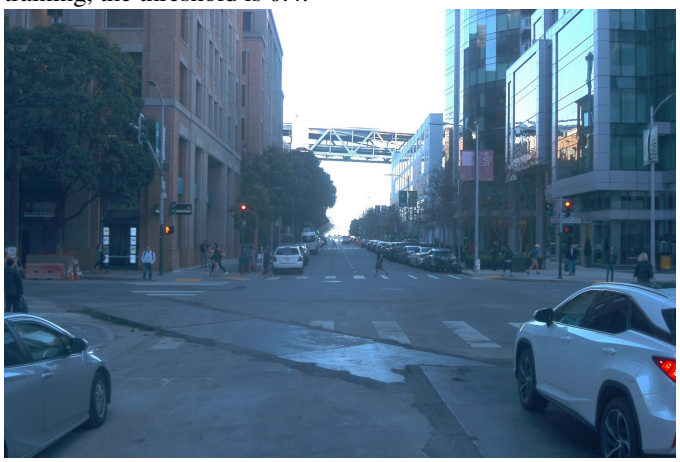


Figure 1: 3 epochs

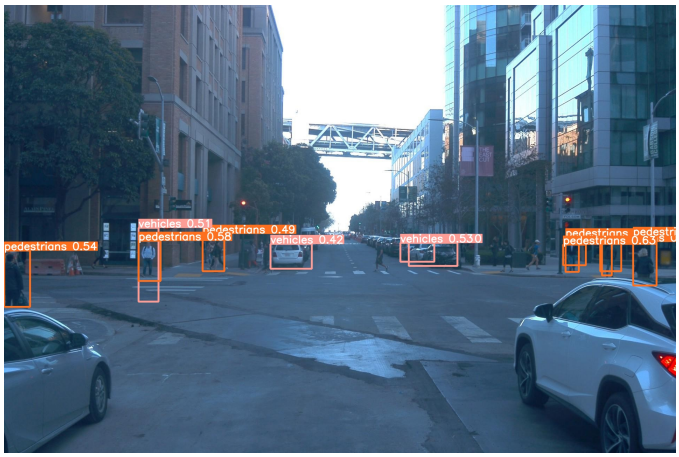


Figure 2: 13 epochs

Figure 3: 30 epochs

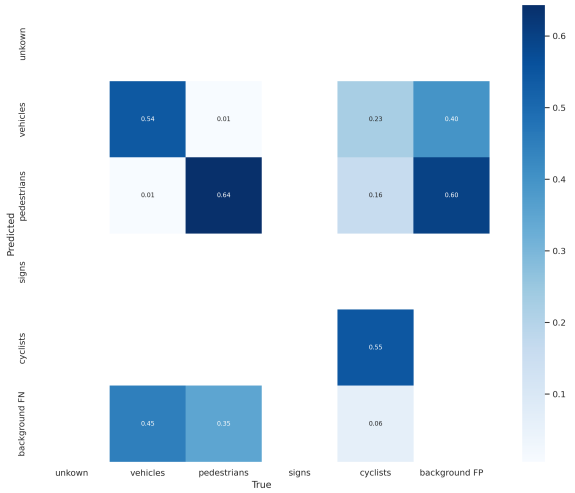
Figure 4: 40 epochs

From the result above, we can see that at the first few epochs our model can not detect any objects.

After about 10 epochs it can identify some pedestrians and vehicles. However, there are many duplicated boxes, and the performance on object nearby and far away is poor.

After 30 epochs training, we have been able to get a good model. even though there are still some duplicated boxes, we believe it will be solved after more training.

Below is the confusion matrix after 40 epochs training, Notice that in this project we will never use class ”signs”, it will be used in 3D motion prediction case.



Mask RCNN

Mask RCNN is a state-of-the-art Convolutional Neural Network in terms of image segmentation and instance segmentation. Theoretically, it will provide us higher accuracy.

In this part we will use MMDection as our library. Here is their official tutorial

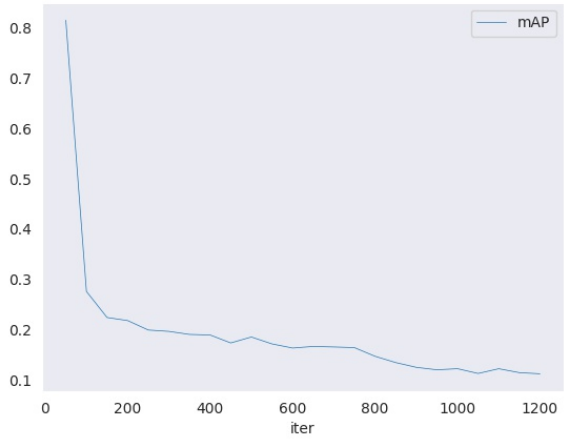
Data Preprocessing

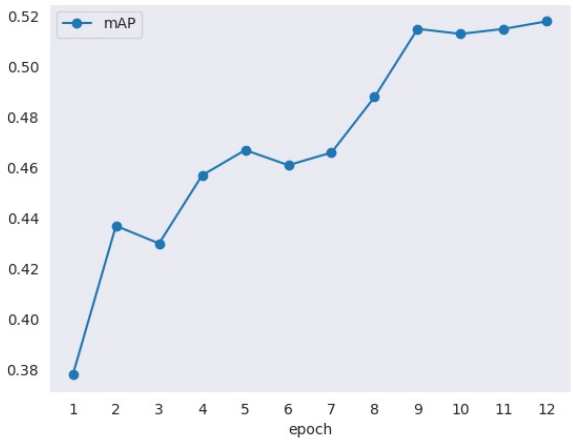
MMDection support COCO format or PASCAL format customize datasets. In the previous part, we have converted the ’.tfrecord’ format data into YOLOv5 format data. Now what we have to do first is to convert our YOLOv5 format data into COCO format. For the complete details of COCO format, please refer here.

We modified code from YOLOv5 to COCO, and saved the COCO format data into google cloud for further use.

Loss&mAP Curves

We plot the loss and mAP curves from our training log file:





Evaluate the Model

Because we used a pretrained model, we got a very good result after only a few hours of training:

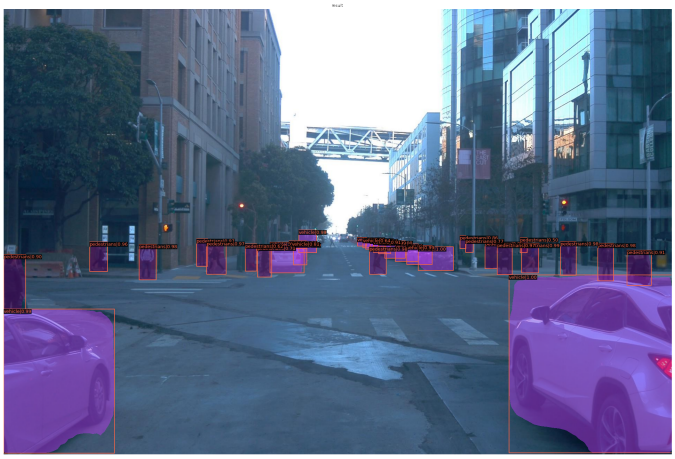


Figure: Mask-RCNN 2EPOCHS

Figure: Mask-RCNN 12EPOCHS

Compare with previous YOLOv5 result, Mask RCNN obviously give us a better prediction.

If you observe very carefully, you can see that the model after 12 epochs training performs slightly better then model after 2 epochs training, it can predict vary far objects very well.

Code and Demo

Here are our codes: Faster RCNN, YOLOv5, Mask RCNN

Here are the video demos of our 3 models: Faster RCNN demo, YOLOv5 demo, Mask RCNN demo.

Our model has a very good performance on recognizing vehicles, but cannot tell pedestrians and cyclists very well. This might because pedestrians and cyclists are more similar, we believe it can be solved by longer training time and using a deeper model.

Generally, RCNN has a better performance than YOLOv5 in terms of accuracy. However, YOLOv5 will still be a better choice, since the processing time of YOLOv5 is much less than RCNN.

Conclusion

We can see that there are many different types of CNNs that can be used to help autonomous vehicles detect objects. The more data and time to train a model dramatically changes how the model can accurately detect objects. As 2D images are one component of autonomous vehicles, the more accurate the detection the more useful the data can be to other components. One such component the data is useful towards is LiDAR. Using the technique bird eye view, LiDAR can detect and recognize objects in 3D space to better enhance the autonomous vehicle surroundings and predict vehicles or pedestrian may go.

As the images that we trained show a high accuracy on detecting object, they are still image and not the real world. According to Abhishek Gupta, in real world autonomous driving scenarios, the error rate "is still a concern." Faster R-CNN did not preform well in locating small vehicles from complex background in real world. YOLO is the faster of the three CNNs that are reported in this paper, however it does not preform as well as the others. "Processing real-time video streams to find the objects of interest is computationally intensive for self-driving scenarios." This means that it could take some time to have fully autonomous vehicles on the road. Either better CNNs and/or computing power is needed before we are ready for level 5 autonomy.

Challenges

The challenges I (Peter Geras) faced was trying to get Faster R-CNN working. I was using a popular GitHub repo made in 2019 that was using Pytorch, which was piggy-back from the original Pytorch repo that used Detetron 1. I followed other people's advice in the issue's page of the repo on how to fix some of the issues. I was changing lines of code, I even followed another tutorial of jwyang faster-rcnn.pytorch that was made in 2019 that tried to help get around the issues. And even then, I could not get it to work. There are still a lot of issues with the repo which the owner has stated he is too busy with his current job to try to fix. There are still people using the repo and running into the same issues.

The next option for myself was following a Detetron 2 tutorial on Faster R-CNN and other tutorials that were similar, to train the model on my own custom dataset from Waymo and pre-trained COCO model. I was given an error that I could not figure out and there were no other resources that showed how to fix the error. I then moved onto following the MMDetections tutorials. I got the idea from one of my partner's codes but this time I would try to train the COCO dataset. I paid for Google Colab Pro so I would be able to train the dataset (extra storage). The first attempt I left my computer for too long and then Google Colab stopped my training time. The second time I stayed the whole time. It trained for 4 and a half hours (out of 2 and a half days) until I stopped. I only trained 1 epoch and stated a low threshold, it could not detect any objects. My final option was a GitHub repo of Faster RCNN using TensorFlow using the vgg16 pretrained model. I wanted to train the first model with 3 epochs. After training the model a little over 5000 steps at epoch 1, the code gave me an error. I followed the GitHub instructions and somehow, I did not know why I was getting this error. As this was the fourth time trying to get something to work, I had to move on, and I felt that I had to present something to this report than nothing. I only posted the last 2 failures to my GitHub as they were the only ones I had. The other two I deleted in frustration.

References

Lewis, G. 2016. Object Detection for Autonomous Vehicles. Unpublished manuscript. Stanford University

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28, 91-99.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., and Berg, A. C. 2016. Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

Tümen, V., and Ergen, B. 2020. Intersections and crosswalk detection using deep learning and image processing techniques. Physica A: Statistical Mechanics and its Applications, 543, 123510.

Abhishek Gupta, Alagan Anpalagan, Ling Guan, Ahmed Shaharyar Khwaja, 2021. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues, source