Scene Understanding for Autonomous Driving

Jisoo Lee[1], Ayoung Kang[1], and Dayeon Oh[1]

[1]Oregon State University, Corvallis, OR, USA
{leejis, kanga2, ohda}@oregonstate.edu

Abstract

What would be the key components of self-driving technology? Detecting objects such as the likes of pedestrians or vehicles to avoid crashes or detecting lanes to follow the lanes to maintain orders on the roads would be the answer. Many related works previously done are focused on either object detection or lane detection, but not many of them have been done on both simultaneously. In this paper, our team suggests a new method that detects both objects and lanes on the roads. We will describe our implementation process and some example results from our method in the following sections.

Index Terms*—Deep learning, computer vision, OpenCV, object detection, real-time recognition*

1. Introduction

Autonomous driving technology is currently one of the most active research topics, which has been increasingly applied to the latest vehicle models rolled out onto the roads. Non-traditional auto companies such as Tesla, Cruise, Waymo, Voyage, etc have been leading the research and development of autonomous driving technologies so far, but traditional auto manufacturers with long history and manufacturing capacity have recently geared up their efforts not to lose their fair shares in this newly emerging market. Propelled by the intense competition between legacy manufacturers and new entrants, the market size of self-driving cars and trucks is projected to grow exponentially. [1] Then why is autonomous driving sought by so many? What are the benefits? The ultimate goal of autonomous driving is to make a safer and convenient driving environment. It attempts to navigate roadways without human intervention by sensing and reacting to the vehicle's immediate environment. In a study looking at critical reasons for car accidents, researchers found that 94% of car accidents were caused by human errors. [2] Automation in self-driving technology can help reduce the number of crashes on the roads that used to be made by drivers' behavioral errors. To avoid those crashes, cars need to 'detect' what is currently on the roads such as vehicles, pedestrians, traffic lights, and lanes. They need to recognize vehicles, pedestrians, traffic lights, and lanes to navigate the roads appropriately.

A YOLOv4 is a well-known computer vision library for object detection, and when we apply this library to road images, the existing YOLOv4 library only detects objects such as vehicles and pedestrians. Therefore, our team added the lane detection method to the YOLOv4 library to detect both objects and lanes. For object detection, we used YOLOv4 trained with Berkeley Driving Dataset with pre-trained weights from the COCO dataset. For lane detection, our team implemented the traditional OpenCV method. With our method, it is possible to detect both objects and lanes on the road in a video dataset. In addition, not just image data, our method is designed to apply video datasets to detect objects and lanes in real-time. In the following section, we will review related works, explain the development process of our detection method, show detection demonstration examples from road video and driving footage, and describe discussion items and future work.

2. Related Work

There are mainly two types of state-of-the-art object detectors. On one hand, we have two-stage detectors, such as R-CNN[3], Fast R-CNN[4] and Faster R-CNN[5] (Region-based Convolutional Neural Networks), that (i) use a Region Proposal Network to generate regions of interests in the first stage and (ii) send the region proposals down the pipeline for object classification and bounding-box regression. Such models reach the highest accuracy rates but are typically slower. On the other hand, we have single-stage detectors, such as YOLO (You Only Look Once)[6] and SSD (Single Shot MultiBox Detector)[7], that treat object detection as a simple regression problem by taking an input image and learning the class probabilities and bounding box coordinates. Such models reach lower accuracy rates but are

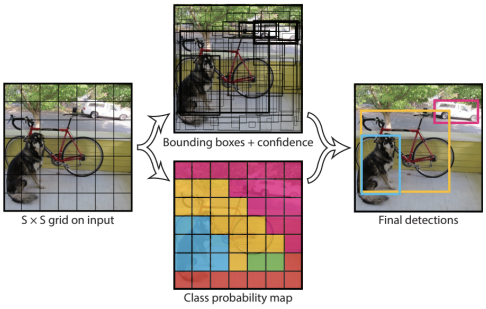

Figure 1: The detection pipeline of YOLO

much faster than two-stage object detectors. In this project, we focused on YOLO v4[8] because it was the most famous one for real-time detection. For lane detection, there are two options: deep learning approach and vision-based approach. With a deep learning model, they have already achieved an accuracy of around 97 percent. However, deep learning may not sometimes be the best option. A large number of parameters can slow down the computation speed. Therefore, we considered a computer vision approach for lane detection.

2.1. YOLO

The whole pipeline of YOLO is a single network as can be seen in Figure 1. It does not require convolution for every region of interest. YOLO splits the input image $n \times n$ into a grid and predicts bounding boxes and class scores for each cell. Then, it applies Non-Maximum Suppression (NMS) to only keep the best bounding box. It removes all the predicted bounding boxes whose detection probability is less than a given NMS threshold. It chooses the bounding boxes that have the highest detection probability and calculates the amount of overlap bounding boxes that predict the same class for different grid cells using Intersection of Union (IOU). All the bounding boxes whose IOU value is higher than a given IOU threshold will be eliminated. By repeating this until all of the non-maximum bounding boxes are removed for every class, it gets the final outputs: bounding boxes and their corresponding class. Yolo v4 has achieved 43.5% AP at a real-time speed of 65 FPS on Tesla V100.

2.2. Lane Detection

The state-of-the-art deep learning model for lane detection, FOLOLane[9] has achieved the accuracy of 96.920 and 100FPS. FOLOLane focuses on modeling local patterns and achieving the prediction of global structures in a bottom-up manner. RESA[10], another recent model for lane detection, takes advantage of strong shape priors of lanes and captures spatial relationships of pixels across rows and columns. This model achieved 36fps so this can be used for real-time applications. However, if the model detects both lanes and objects, FPS will drop. Also, it is not easy to train both object detection and lane detection. In order to train in a fully supervised learning method, the model needs the ground truth labels which contain bounding boxes and lanes. But most datasets don't provide the two kinds of labels together. This means that in order to do both tasks together, two types of datasets must be trained in a multi-task learning manner.

Without any datasets or deep learning strategy, we could detect lanes in a traditional computer vision manner. Canny edge detector[11] determines the potential edges from the image with two thresholds. After generating an edge map, apply Hough transform to get line candidates. Finally, we can get essential lines corresponding to lanes from the line candidates.

3. Implementation

This section describes how we implemented the object detection and lane detection algorithms. For experiments, Nvidia GeForce RTX 2080 and Tesla V100 were used.

3.1. Object Detection

For object detection, we adopted YOLOv4 and used pre-trained weights for the MS coco dataset without training the whole coco data. This is because it takes 10 hours per epoch to train the model. We referred to this Github repo as a base model. Based on the MS coco pre-trained model, we additionally trained and fine-tuned the model on the Berkeley Deepdrive dataset (BDD100K). BDD100K is one of the largest datasets for autonomous driving. BDD100K for object detection consists of 69,567 training images and 9,960 validation images. In addition, YOLOv4's data augmentation technique like mosaic can be combined to improve the model with a sufficient amount of new datasets.

Dataset preparation: Compared to the MS coco dataset which has 80 classes, BDD100K has 15 classes. We chose 10 classes from 15 classes: pedestrian, other person, rider, bicycle, car, motorcycle, bus, train, truck, and traffic light. Among the chosen classes, we merged some classes into one class to correspond to class names belonging to MS coco. Finally, we converted BDD100K dataset to MS coco format. After conversion, we convert it again to a data format for YOLO v4.

To train on BDD100K, we changed some hyperparameters and the class parameters of the YOLO layer and increased the stride from 32 to 64 in order to make the grid size smaller because the resolution of the BDD dataset is much larger than the MS coco dataset's resolution.

(a) Image with Object detection (b) Gray-Scaled image (c) Edge detection with Canny

(d) Line detection with Hough transform (e) Line detection result

Figure 2: Lane detection pipeline

3.2. Lane Detection

For lane detection, we implemented a vision-based lane detection algorithm. Firstly, after converting the RGB color image to grayscale and then we got an edge map. When processing the image using the Canny detector, we didn't choose the upper threshold and lower threshold manually. With image median value, our code automatically computed two thresholds. We set the bottom 1/3 of the video/images edge map as the region of interest(RoI) and got line candidates from that RoI. This is possible because the lanes are typically located at the bottom of the image due to the characteristic of the car dataset. From the several line candidates, we picked up two lines which are on the right side and left side of the image. We assumed that there is one lane on the left and one lane on the right by the driver. This work is added to the testing process.

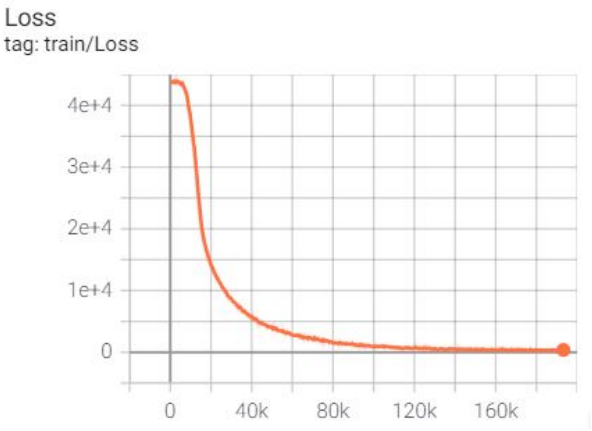

Figure 3: Training loss

4. Results

After training and testing with Berkeley Deep Drive data set with the pre-trained YOLOv4 model, we tested our detection model with custom data set. This section describes object and lane detection results from both image and video data.

4.1. Training Result

We trained the model using a subset of the BDD100K dataset (training size: 1,000, validation size: 300) for 780 epochs with a learning rate of 0.00261 and batch size 64. We measured average precision (AP), average recall (AR), and training loss. Regarding precision and accuracy, we have not achieved very high precision and accuracy since we used a subset of the BDD100K dataset and we were not able to fully train our model. Figure 3 shows the training loss for YOLO. The YOLO loss function composes of localization loss (loss_xy + loss_wh), confidence loss (the objectness of the box) (loss_obj), and classification loss(loss_cls). We observe that the total loss decreases as the training process goes on.

Figure 4: Detection results comparison (left: coco pretrained model, right: BDD100K fine-tuned model)

4.2. Detection Result

Before testing road detection with video data such as driving footage, we first tested with the static image file. We converted a video of the 1940s New York road [1] to a group of images by cutting the video in 1-second increments and tested the image group with the model. One of the detection results from the New York road video is shown in Figure 5. We can see that objects on the road such as buses, cars, and people are detected. After we figure out that our model is working well on the image file, we tested our model with a video file. We applied our model directly to the video file and created the detection demonstration file. For the testing video dataset, our group found driving footage from the familiar Corvallis downtown area[2] and used this as our demonstration file. Capture images from the demonstration video are shown in Figure 6. From Figure 6, we can see that lanes and objects are detected. Unlike Figure 5, there are more lanes and objects such as traffic lights are detected. For the video data, testing with GPU of Nvidia GeForce RTX 2080, 52MB video file with frame size 720*360, we

[1] https://www.youtube.com/watch?v=yHe_d2hUVnA

[2] https://www.youtube.com/watch?v=TYUBIAaV4HU

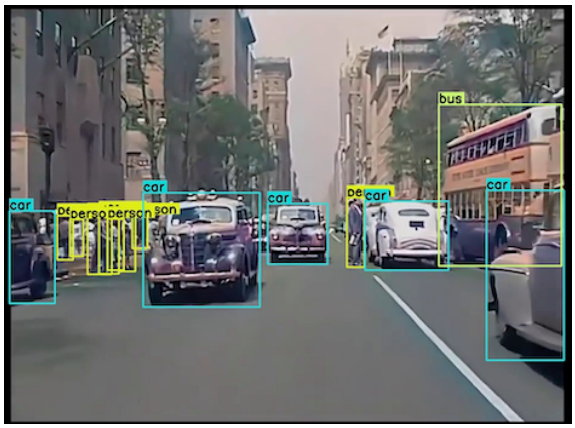

Figure 5: Object detection result from the 1940s New York street.

Figure 6: Capture frames from object and lane detection result video demonstration from Corvallis, Oregon downtown area.

achieved around 10 frames per second for our detection result. Considering testing with a less powerful GPU resulted in lower frames per second, we assume testing with a higher GPU or testing with a smaller file size would result in higher frames per second result. After a few tests run, we found that the model is working well with both image and video data types.

4.3. Detection Results Comparison

After 4 epochs of training on the whole BDD dataset are completed, we compared the detection results of the two models. We found out that, unlike MS coco, the model trained on BDD100K is applicable to diverse environmental domains. In Figure 4, the left column shows the results of using a pre-trained YOLO model and the right column shows the results of using a fine-tuned model trained on the BDD100K dataset. The top two rows show the detection results in night-time driving environments. We observe that the fine-tuned model detects small objects better than the pre-trained model does (Table 1). We can also see that the pre-trained model reports no objects (c) but the fine-tuned model detects a traffic light and cars (d). We think that this is because the BDD dataset contains driving sequences recorded in diverse locations around the United States, in a wide range of weather conditions and settings such as rainy, overcast, sunny, at night, and during the day. In the bottom row, we observe that the hood of the driving car is not detected as a car when using the fined tuned model.

Class	Pre-trained model	Fine-tuned model
person	0.965412	0.972265
person	0.956133	0.912019
person	0.728071	0.609608
person	0.673027	0.596342
person	0.601658	0.594183
person	0.478431	-
person	0.400903	-
car	0.990254	0.994744
car	0.988861	0.994270
car	0.969154	0.991415
car	0.955564	0.990689
car	0.785408	0.952090
car	-	0.739255
traffic light	0.727526	0.739762
traffic light	-	0.532760

Table 1: Comparison of accuracy scores for the top row of Figure 4

5. Limitation and Future work

Start early with a powerful GPU The dataset that our team used for training was the Berkeley Deep Drive data set, and its training data size was 3.77GB and validation size was 553MB. With Nvidia GeForce RTX 2080, training data took about 3 hours for each epoch. Also, since we were using a server from the university's engineering department, we were not able to use the whole GPU memory all the time. Therefore, our team was not able to finish the training with the epoch size that we first intended (300 epoch) which led us to get not enough value of loss and accuracy. For future work, we are hoping to use a more powerful GPU to train our model and test it again.

Applying dilated convolution to reduce the training time When using vanilla convolutions, we need to downsample to integrate global context. Dilated Convolutions are a type of convolution that inflate the kernel by inserting holes between the kernel elements [12]. By using dilated convolutions, we are able to capture global information from far away pixels, and we do not need to downsample all that much to capture the information. Therefore, we

believe that dilated convolutions will allow large-size inputs to be processed without slowing down.

Aiming for 30 frames per second Frames per second(FPS) is an important metric in evaluating real-time detection performance that indicates how many frames are processed per second. When it comes to real-time data like traffic monitoring or live streaming, 30 fps is known as best and our team is hoping to meet this goal in the future. Currently, the frames per second value that we achieved with the downtown area of Corvallis driving footage video was around 10, which is lower than our ideal goal. Adding a lane detection task in a computer vision manner dropped the FPS by 25%. Compared to the state-of-the-art deep learning-based approach, the performance of computer vision-based approach was not better. In future work, training the model in a multi-task learning strategy using two different datasets could be a solution. Also, the GPU spec or other development environment can be a reason for low fps performance, but for future work our team is hoping to increase the fps rate by changing the image size, using threading, or fine-tuning other hyperparameters.

Fine tune on KITTI dataset For our implementation process, we used pre-trained weights trained with COCO data and trained with Berkeley Deep Drive data. To achieve higher accuracy, we can fine-tune trained weights using other driving dataset like KITTI data. We are hoping we can test our custom data again with weights achieved from the training result with KITTI data.

References

[1] Statista Research Department. Size of the global autonomous car market 2019-2023. *Statista*, 2021. 1

[2] Bruce Brown. Evidence stacks up in favor of self-driving cars in 2016 nhtsa fatality report. *digitaltrends*, 2017. 1

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014. 1

[4] Ross Girshick. Fast r-cnn, 2015. 1

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. 1

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016. 1

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016. 1

[8] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020. 2

[9] Zhan Qu, Huan Jin, Yang Zhou, Zhen Yang, and Wei Zhang. Focus on local: Detecting lane marker from bottom up via key point, 2021. 2

[10] Tu Zheng, Hao Fang, Yi Zhang, Wenjian Tang, Zheng Yang, Haifeng Liu, and Deng Cai. Resa: Recurrent feature-shift aggregator for lane detection, 2021. 2

[11] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. 2

[12] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2016. 5