2023年自動駕駛實務

期末報告書

成員
朱福城 P76107085
徐子桓 E24084062
廖泓博 E24086234
廖家問 E24084729

說明：

此計劃包含兩部分的實作，基於 Waymo 資料集之 2D Object Detection 與基於 Lyft 資料集之 3D Object Detection。在接下來的章節將會詳細介紹兩者之間的差異性與使用場景。

1. Introduction

I. Object Detection Background
II. CNN in Object Detection
III. 2D Object Detection
IV. LIDAR and 3D Object Detection

2. Related Works

- 2D Object Detection
  I. SSD
  II. Faster RCNN
  III. YOLO
  IV. YOLO9000 / YOLOv2
  V. RetinaNet
  VI. EfficientDet

- 3D Object Detection
  VII. PointNet
  VIII. VoxelNet
  IX. SECOND
  X. Complex-YOLO
  XI. PointPillars

3. Datasets

I. Kaggle Competition: Lyft 3D Object Detection for Autonomous Vehicles
II. Waymo

4. Setup & Experiments

- Kaggle Competition, Lyft
  I. Hardware Spec. & Software Version
  II. Implementation Details
- Waymo

I. Hardware Spec. & Software Version
II. Implementation Details

5. Results

- Kaggle Competition, Lyft
- Waymo

6. Ablation Studies

7. Challenge

8. Conclusion

Introduction

In recent years, advancements in artificial intelligence (AI) and computer vision have paved the way for remarkable breakthroughs in autonomous applications. One of the key components that drives the success of these applications is object detection.

Object detection refers to the technology that enables machines to perceive and identify objects within their visual environment, empowering them to make informed decisions and interact with their surroundings autonomously. This revolutionary field holds immense potential for transforming various industries, including self-driving cars, robotics, surveillance systems, and more.

Object detection plays a critical role in enabling autonomous vehicles to perceive and understand their surroundings. By accurately identifying and tracking objects like pedestrians, vehicles, traffic signs, and traffic lights, self-driving cars can make informed decisions and navigate safely.

Object detection involves the process of recognizing and localizing objects of interest within an image or video stream. This technology equips autonomous systems with the ability to identify and track objects in real-time, providing crucial information for decision-making. Traditional computer vision methods relied on manual feature engineering, which often proved to be labour-intensive and limited in performance. However, with the advent of deep learning techniques, particularly convolutional neural networks (CNNs), object detection has experienced a significant leap forward.

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision, enabling highly accurate and efficient object detection. CNNs are designed to mimic the visual processing capabilities of the human brain, enabling machines to automatically learn and extract features from images. This ability to learn hierarchical representations of objects has dramatically enhanced the accuracy and speed of object detection algorithms.

2D Object Detection Techniques can be categorized into below three categories.

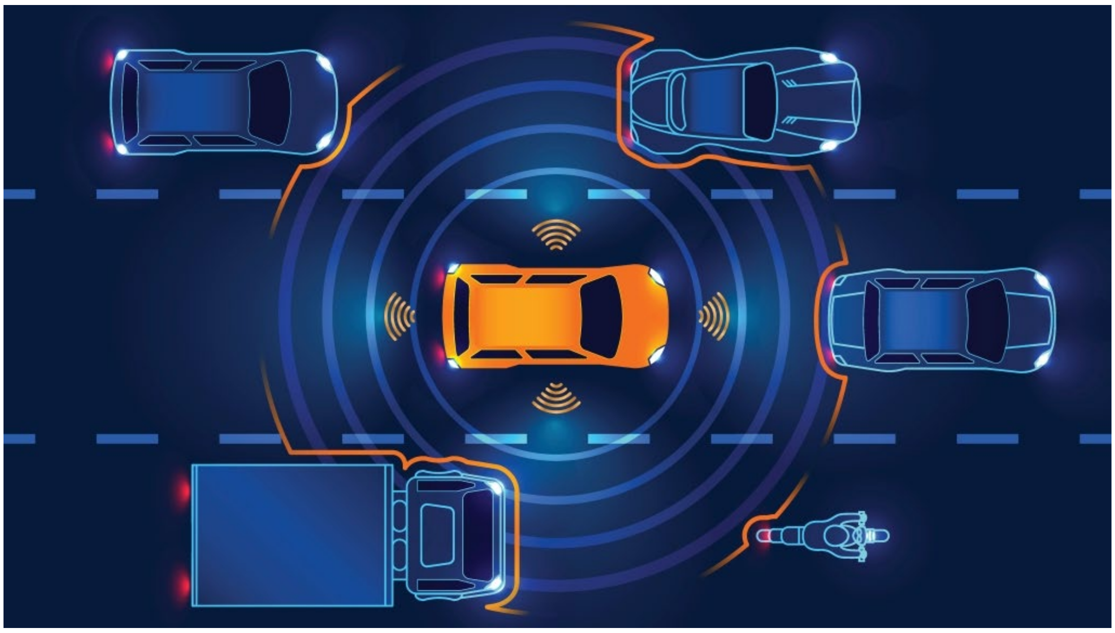
The first category is Single Shot Detectors (SSDs), a popular class of object detection algorithms that are renowned for their real-time performance. They employ a single neural network to simultaneously predict object classes and bounding box coordinates for multiple objects in an image. By leveraging feature maps at different scales, SSDs can effectively detect objects of varying sizes.

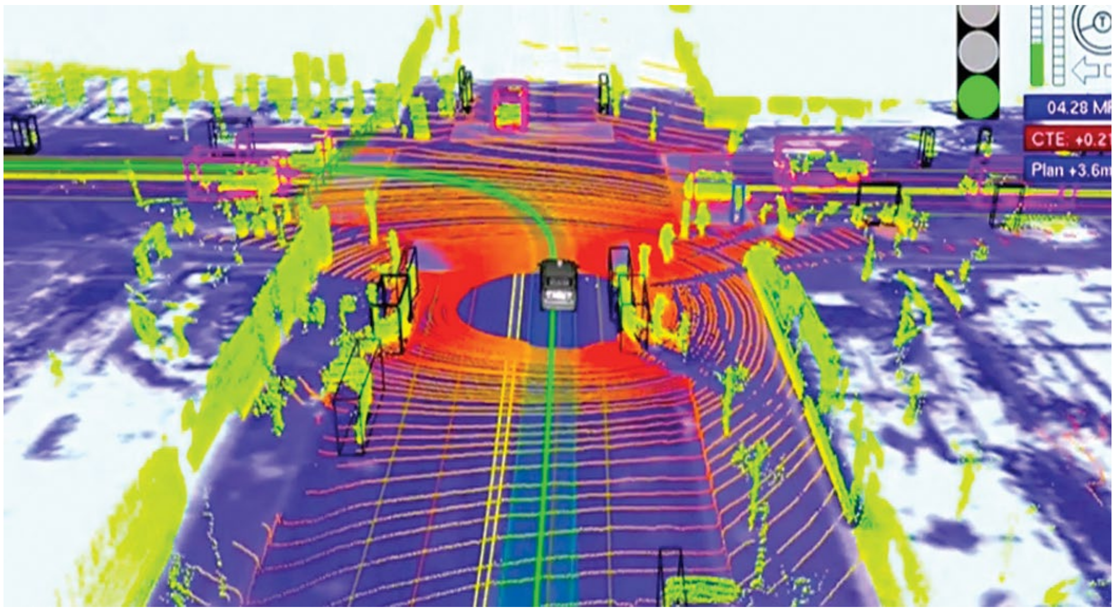
The next category is Region-Based Convolutional Neural Networks (R-CNNs). R-CNNs introduced the concept of region proposal algorithms to improve object detection accuracy. They generate a set of region proposals that are likely to contain objects, which are then classified and refined using CNNs. R-CNNs have been further improved with subsequent iterations, such as Fast R-CNN and Faster R-CNN, which offer enhanced speed and accuracy.

The third category is You Only Look Once (YOLO). YOLO is a pioneering real-time object detection algorithm that provides impressive speed and accuracy. It divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell. YOLO is known for its ability to achieve near real-time performance, making it well-suited for time-sensitive autonomous applications. For the YOLO series, it has now come to the 8th version.

Traditional 2D object detection methods operate on 2D images or video frames and focus on detecting and classifying objects in two dimensions. However, the real world is inherently three-dimensional, requiring autonomous vehicles to perceive and understand objects in 3D space. 3D object detection aims to extend the capabilities of traditional approaches by accurately localizing and classifying objects in 3D using sensor data from multiple sources.

Light Detection and Ranging (LiDAR) is a remote sensing technology that uses laser pulses to measure the distance to objects and create a detailed 3D point cloud representation of the surrounding environment. LiDAR sensors emit laser beams, and by measuring the time it takes for the light to bounce back after hitting an object, they can determine the distance to that object. By scanning the laser beams in different directions, LiDAR sensors generate a 3D point cloud, which provides precise spatial information about the objects in the environment.



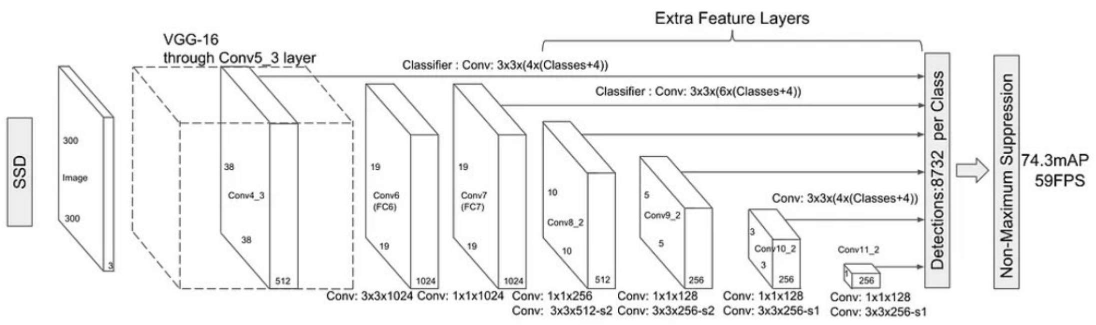


The fusion of LiDAR and 3D object detection techniques has become a cornerstone in enabling self-driving cars to accurately perceive and understand the surrounding world. LiDAR sensors provide crucial depth information, allowing for the precise localization of objects in 3D space. By combining LiDAR data with other sensor modalities such as cameras and radar, the perception system of a self-driving car can leverage the complementary strengths of each sensor to achieve robust and reliable object detection.

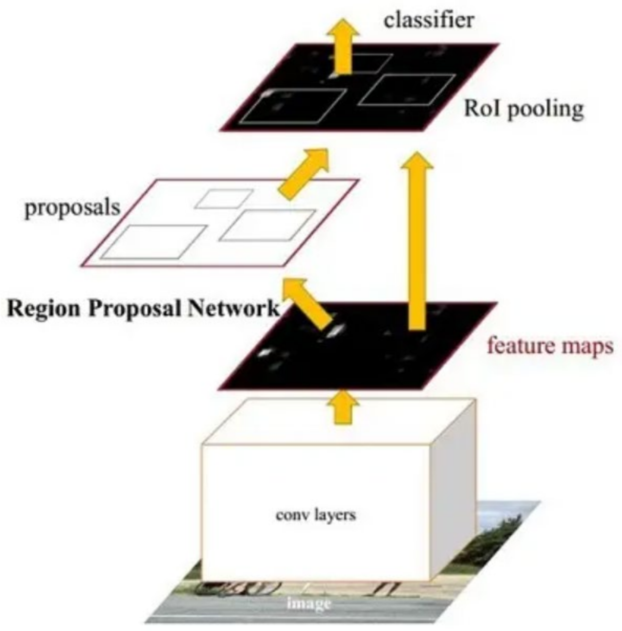
However, there are challenges associated with 3D object detection and LiDAR technology. LiDAR sensors can be costly and have limitations in certain weather conditions, such as heavy rain or fog, where their performance may be compromised. Furthermore, processing large-scale 3D point clouds in real-time requires significant computational resources and efficient algorithms to achieve the necessary speed for autonomous driving applications.

Related Works

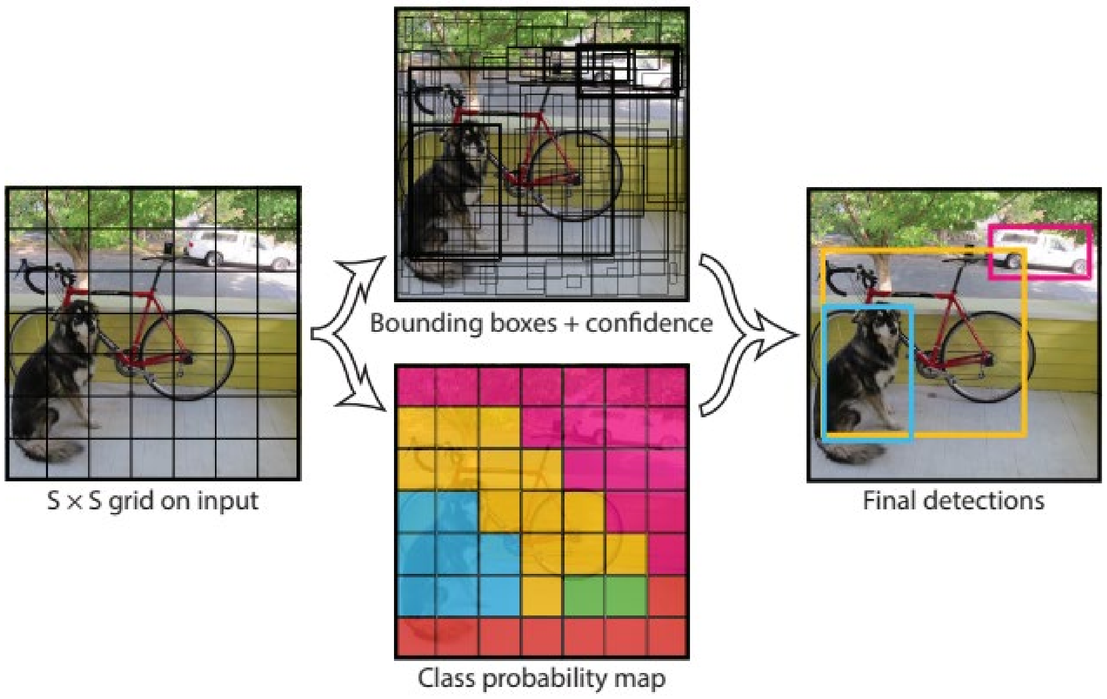
SSD. Wei Liu et al. introduced a single-shot object detection framework called Single Shot MultiBox Detector that achieved real-time performance without the need for region proposal networks. It is a type of object detection model that can detect and classify objects in an image or video in a single pass through the network. It utilized a series of convolutional layers at different scales to predict object bounding boxes and class probabilities directly from feature maps. SSD offered an excellent balance between accuracy and speed, making it well-suited for real-time applications in self-driving cars.



Faster R-CNN. ShaoQing Ren et al. proposed a region-based convolutional neural network architecture for object detection. It is an extension of the R-CNN framework that addresses its inefficiencies and improves the detection speed. It introduced the concept of region proposal networks (RPNs) to generate potential object regions for subsequent classification. The RPN generates a large number of region proposals, which are then refined by a region of interest (ROI) pooling layer. The ROI pooling layer extracts fixed-size feature maps from the convolutional feature map for each proposal. These features are subsequently fed into a classifier and a regressor for object classification and bounding box refinement, respectively. Faster R-CNN achieved significant improvements in detection accuracy and laid the foundation for subsequent works in the field.

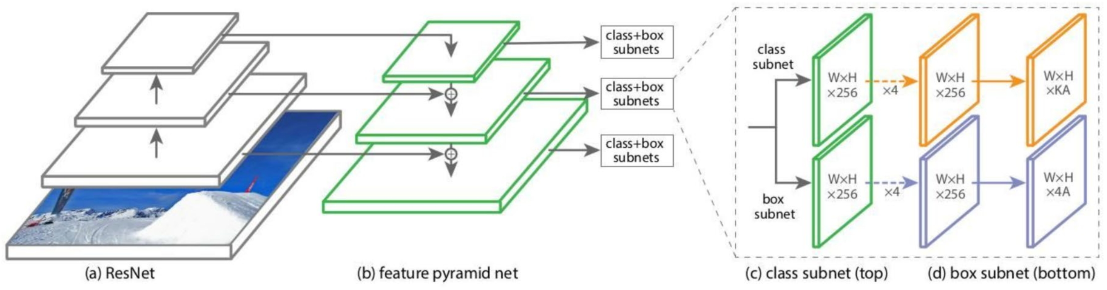


YOLO. Joseph Redmon et al. introduced the YOLO framework which revolutionized object detection in real-time applications. YOLO unified object detection and localization into a single end-to-end neural network, achieving impressive speed and accuracy. The model achieved state-of-the-art results on various benchmarks and paved the way for subsequent advancements in real-time object detection for self-driving cars.

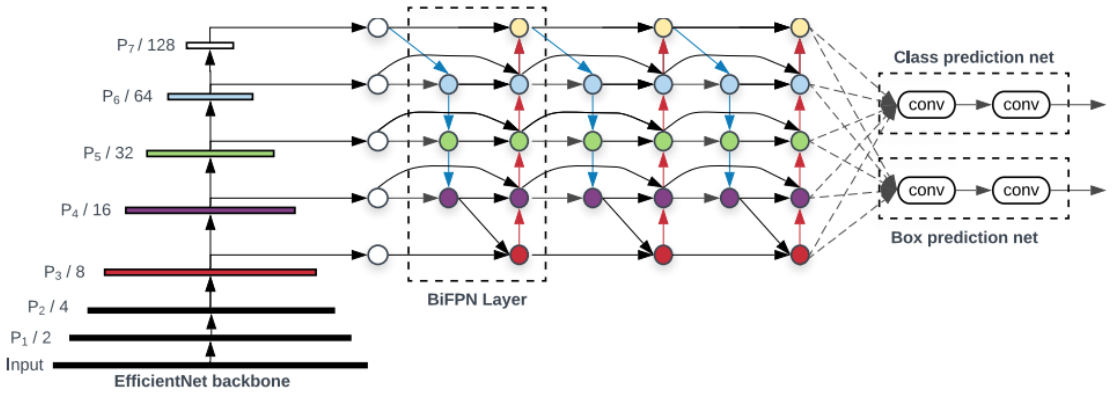


YOLOv2:YOLO9000. Building upon the success of the original YOLO, YOLOv2 introduced several key improvements to enhance detection performance. It introduced anchor boxes, which improved localization accuracy, and incorporated a hierarchical feature extraction network called Darknet-19. YOLOv2 also introduced the concept of joint training, enabling the model to learn from both COCO and ImageNet datasets, resulting in improved generalization and object detection capabilities.

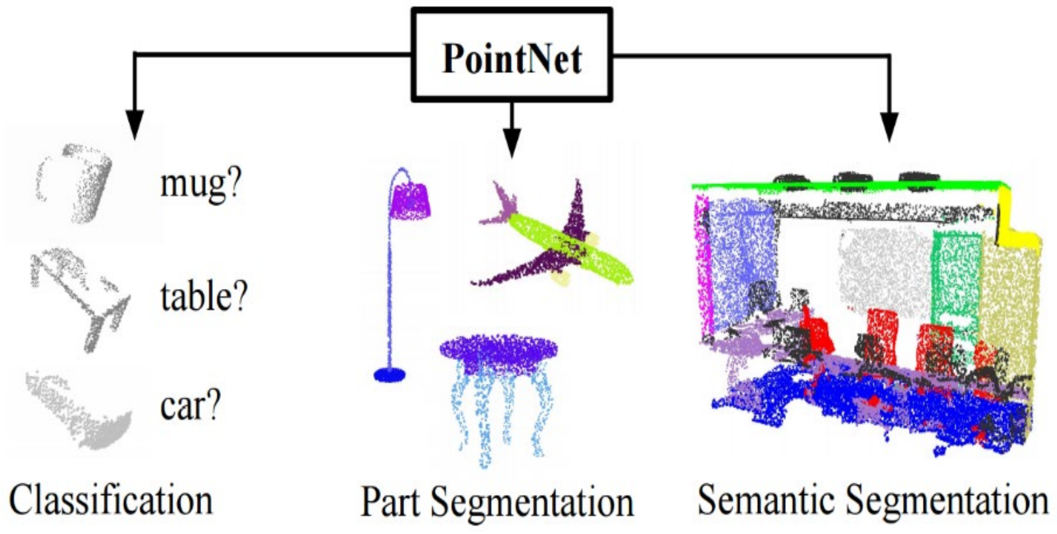
RetinaNet. Tsung-Yi Lin et al. introduced RetinaNet, a novel focal loss function to address the issue of class imbalance in object detection. The focal loss dynamically adjusts the loss weights during training to give more importance to hard examples and reduce the impact of easy examples. This is achieved by introducing a modulating factor, which downweighs the loss for well-classified examples and focuses more on the challenging examples. By emphasizing the training on hard examples, RetinaNet can effectively handle the class imbalance problem and improve the detection performance, especially for small objects. This approach significantly improved object detection accuracy, especially for small and challenging objects, making it highly relevant for self-driving car scenarios where accurate detection of pedestrians and other vulnerable road users is crucial.



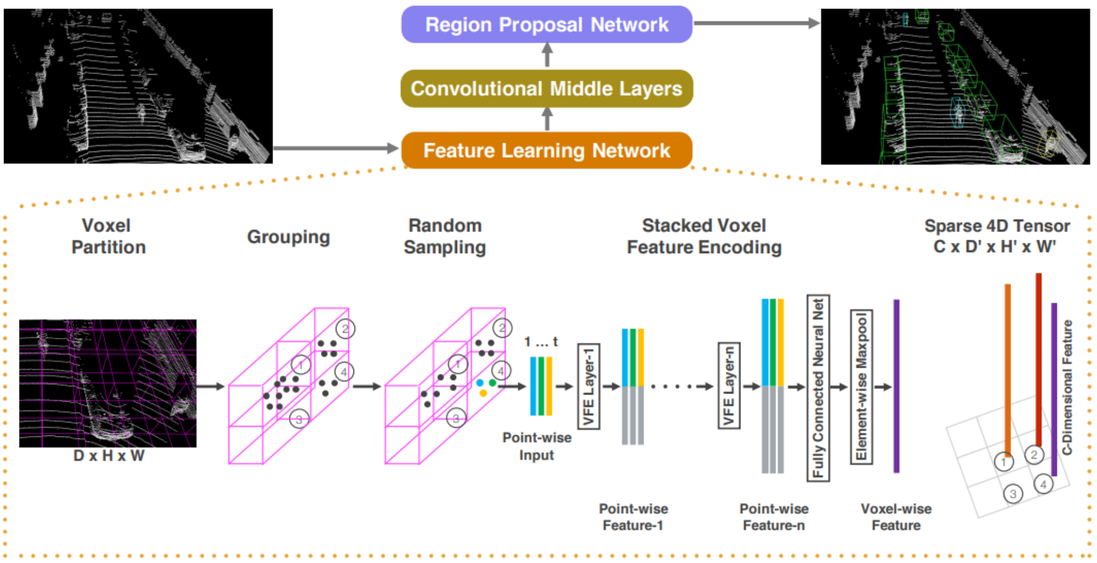
EfficientDet. Mingxing Tan et al. proposed a scalable and efficient object detection architecture that achieved state-of-the-art performance with significantly fewer computational resources. EfficientDet builds upon the success of previous object detection models, such as EfficientNet, which focused on efficient neural network architectures. The goal of EfficientDet is to design a scalable and efficient object detection framework that can achieve high accuracy even with limited computational resources. EfficientDet also introduces a novel compound scaling coefficient called "phi" that controls the overall model size and computational cost. By varying the value of phi, different versions of EfficientDet can be obtained, ranging from smaller and faster models to larger and more accurate ones. This flexibility allows EfficientDet to cater to a wide range of resource constraints and deployment scenarios.



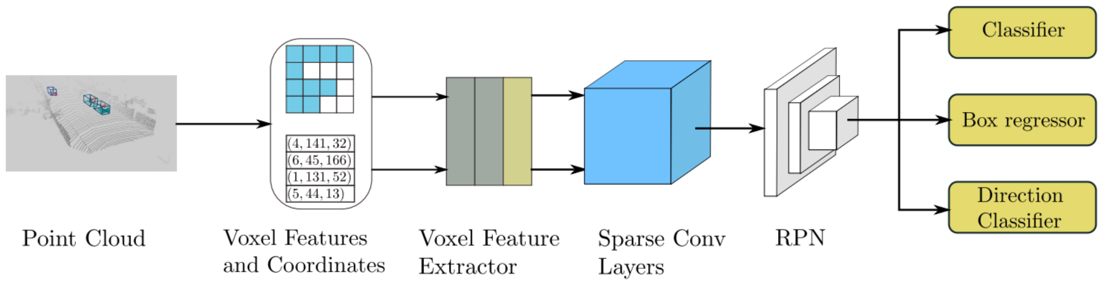
PointNet. Charles R. Qi et al. introduced Deep Learning on Point Sets for 3D Classification and Segmentation Network for processing unordered point cloud data. The architecture of PointNet consists of three main modules: the input transformation network, the feature extraction network, and the symmetric function. The input transformation network learns an affine transformation to normalize and align the input point cloud. The feature extraction network processes each point independently and captures local information using shared multi-layer perceptrons (MLPs). Finally, the symmetric function aggregates the features from all the points to generate a global feature vector that represents the entire point cloud.



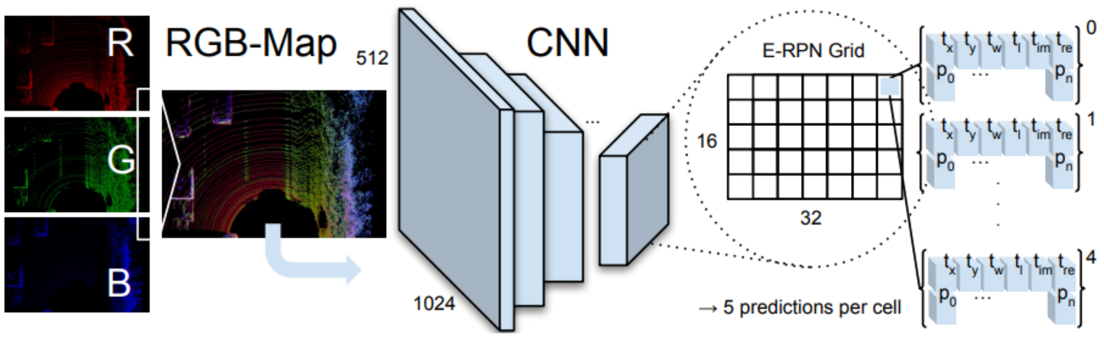
VoxelNet. Yin Zhou and Oncel Tuzel proposed a novel framework for 3D object detection using LiDAR point clouds. The main idea behind VoxelNet is to transform the point cloud data into a volumetric representation called a voxel grid. A voxel grid divides the 3D space into a grid of equally sized voxels, where each voxel represents a small volume in the scene. The occupancy state of each voxel is determined based on whether it contains any points from the input point cloud. VoxelNet consists of three main modules: the Voxel Feature Encoding (VFE) module, the 3D CNN module, and the Region Proposal Network (RPN). The VFE module encodes the point cloud features within each voxel, capturing local spatial information and intensity values. The 3D CNN module processes the encoded voxel features to learn higher-level representations, extracting semantic features from the volumetric data. The RPN generates 3D bounding box proposals for potential objects by predicting the object-ness score and regression offsets for each voxel.



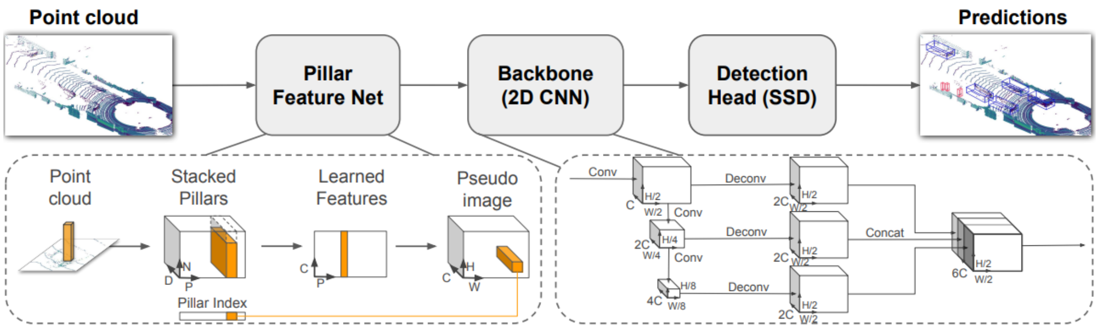
SECOND. Yan Yan et al. introduced Sparsely Embedded Convolutional Detection network for 3D object detection using LIDAR. The key motivation behind SECOND is to leverage the sparsity inherent in LIDAR point cloud data to develop an efficient and accurate 3D object detection system. Unlike dense voxel-based methods, SECOND operates on the sparse point cloud directly, reducing the computational complexity and memory requirements. The architecture of SECOND consists of two main components: the Voxel Feature Encoding (VFE) module and the Sparsely Embedded Convolutional (SEC) module. The VFE module processes the sparse point cloud by dividing it into fixed-height horizontal slices and encoding the points within each slice into a fixed-size voxel grid. This encoding captures local geometric and density information. The SEC module employs sparse convolutional layers to extract features from the encoded voxels, taking advantage of their sparsity and reducing computational costs.



Complex-YOLO. Matthias Müller et al. proposed Real-time 3D Object Detection on Point Clouds which combines the strengths of YOLO and PointNet. The architecture of Complex-YOLO consists of three main components: the Voxel Feature Encoding (VFE) module, the Complex-YOLO module, and the final prediction stage. The VFE module encodes the point cloud into a fixed-size voxel grid, capturing local spatial and geometric information. The Complex-YOLO module processes the encoded voxel features using a series of convolutional and fully connected layers to extract high-level semantic information. Finally, the prediction stage generates 3D bounding box predictions, including the object class, 3D location, dimensions, and orientation.



PointPillars. It was introduced in the research paper titled "PointPillars: Fast Encoders for Object Detection from Point Clouds" by Alex H. Lang et al. PointPillars addresses the challenge of efficiently processing large-scale point cloud data for object detection tasks. The main components of the network are a Pillar Feature Network, Backbone, and SSD Detection Head. The raw point cloud is converted to a stacked pillar tensor and pillar index tensor. The encoder uses the stacked pillars to learn a set of features that can be scattered back to a 2D pseudo-image for a convolutional neural network. The features from the backbone are used by the detection head to predict 3D bounding boxes for objects.

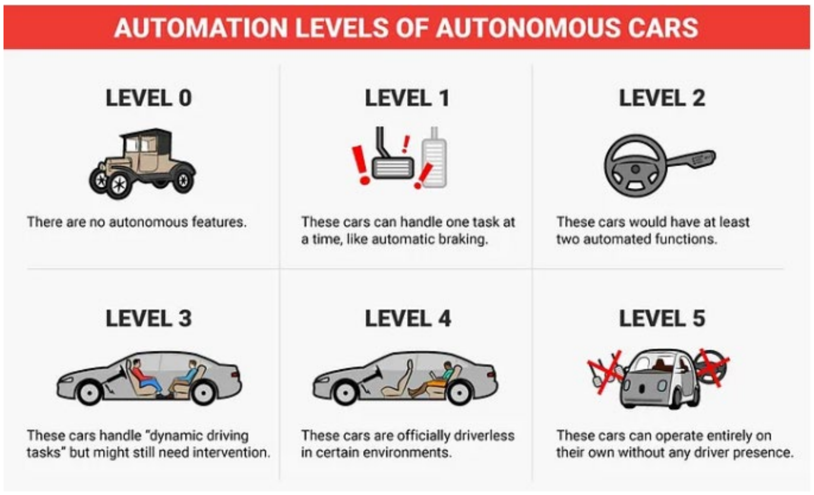


Datasets

- Kaggle Competition: Lyft 3D Object Detection for Autonomous Vehicles

Lyft is a transportation network company based in the United States. It operates a popular ride-hailing platform that connects passengers with drivers through a mobile app. Founded in 2012, Lyft offers on-demand transportation services, allowing users to request rides from nearby drivers. The company provides various ride options, including shared rides, standard rides, luxury rides, and more. Lyft's mission is to improve people's lives with the world's best transportation by providing safe, reliable, and affordable rides.

Lyft, whose mission is to improve people’s lives with the world’s best transportation, is investing in the future of self-driving vehicles. Level 5, their self-driving division, is working on a fleet of autonomous vehicles. Their goal is to democratize access to self-driving technology for hundreds of millions of Lyft passengers.

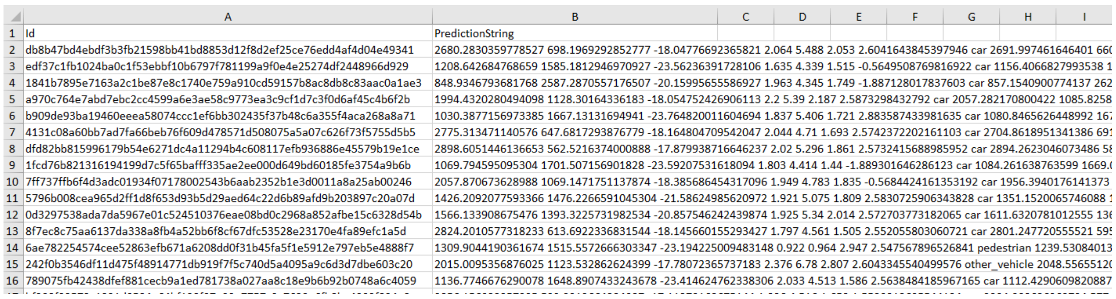

AUTOMATION LEVELS OF AUTONOMOUS CARS
LEVEL 0
There are no autonomous features.
LEVEL 1
These cars can handle one task at a time, like automatic braking.
LEVEL 2
These cars would have at least two automated functions.
LEVEL 3
These cars handle "dynamic driving tasks" but might still need intervention.
LEVEL 4
These cars are officially driverless in certain environments.
LEVEL 5
These cars can operate entirely on their own without any driver presence.

This dataset aims to foster innovation in higher-level autonomy functions for everyone, everywhere. By conducting a competition, we hope to encourage the research community to focus on hard problems in this space—namely, 3D object detection over semantic maps. This dataset features the raw sensor camera inputs as perceived by a fleet of multiple, high-end, autonomous vehicles in a restricted geographic area.

The dataset includes LIDAR, image, map and data files, mostly are in JSON format. The total size of the dataset is approximately 85GB in zip.

The data was captured by 10 host cars on the roads of Palo Alto, California. Each of the host cars has seven cameras and one LiDAR sensor on the roof, and 2 smaller sensors underneath the headlights of the vehicle.

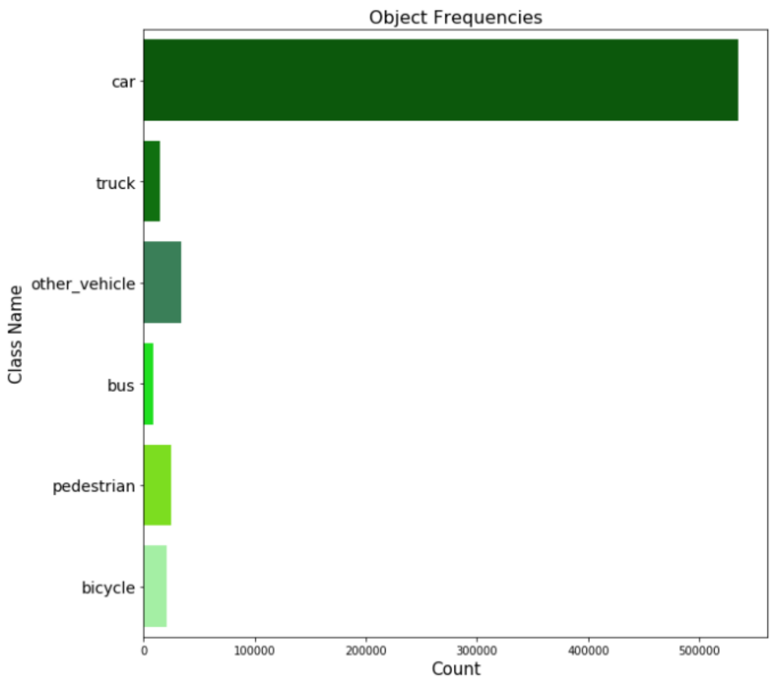
The annotation file "train.csv" has the following annotations in the format of: [center_x center_y center_z width length height yaw class_name]. The center_x, center_y and center_z are the world coordinates of the center of the 3D bounding volume. Whereas width, length and height are the dimensions of the volume. The yaw is the angle of the volume around the z axis (where y is forward/back, x is left/right, and z is up/down - making 'yaw' the direction the front of the vehicle / bounding box is pointing at while on the ground). And class_name is the type of object contained by the bounding volume.



	A	B
1	Id	PredictionString
2	db8b47bd4ebdf3b3fb21598bb41bd8853d12f8d2ef25ce76edd4af4d04e49341	2680.2830359778527 698.1969292852777 -18.04776692365821 2.064 5.488 2.053 2.6041643845397946 car 2691.997461646401 66
3	edf37c1fb1024ba0c1f53ebbf10b6797f781199a9f0e4e25274df2448966d929	1208.642684768659 1585.1812946970927 -23.56236391728106 1.635 4.339 1.515 -0.5649508769816922 car 1156.4066827993538 1
4	1841b7895e7163a2c1be87e8c1740e759a910cd59157b8ac8db8c83aac0a1ae3	848.9346793681768 2587.2870557176507 -20.15995655586927 1.963 4.345 1.749 -1.887128017837603 car 857.1540900774137 262
5	a970c764e7abd7ebc2cc4599a6e3ae58c9773ea3c9cf1d7c3f0d6af45c4b6f2b	1994.4320280494098 1128.30164336183 -18.054752426906113 2.2 5.39 2.187 2.5873298432792 car 2057.282170800422 1085.8258
6	b909de93ba19460eeea58074ccc1ef6bb302435f37b48c6a355f4aca268a8a71	1030.3877156973385 1667.13131694941 -23.764820011604694 1.837 5.406 1.721 2.883587433981635 car 1080.8465626448992 16
7	4131c08a60bb7ad7fa66beb76f609d478571d508075a5a07c626f73f5755d5b5	2775.313471140576 647.6817293876779 -18.164804709542047 2.044 4.71 1.693 2.5742372202161103 car 2704.8618951341386 69
8	dfd82bb815996179b54e6271dc4a11294b4c608117efb936886e45579b19e1ce	2898.6051446136653 562.5216374000888 -17.879938716646237 2.02 5.296 1.861 2.5732415688985952 car 2894.2623046073486 58
9	1fcd76b821316194199d7c5f65bafff335ae2ee000d649bd60185fe3754a9b6b	1069.794595095304 1701.507156901828 -23.59207531618094 1.803 4.414 1.44 -1.889301646286123 car 1084.261638763599 1669.
10	7ff737ffb6f4d3adc01934f07178002543b6aab2352b1e3d0011a8a25ab00246	2057.870673628988 1069.1471751137874 -18.385686454317096 1.949 4.783 1.835 -0.5684424161353192 car 1956.3940176141373
11	5796b008cea965d2ff1d8f653d93b5d29aed64c22d6b89afd9b203897c20a07d	1426.2092077593366 1476.2266591045304 -21.58624985620972 1.921 5.075 1.809 2.5830725906343828 car 1351.1520065746088 1
12	0d3297538ada7da5967e01c524510376eae08bd0c2968a852afbe15c6328d54b	1566.133908675476 1393.3225731982534 -20.857546242439874 1.925 5.34 2.014 2.572703773182065 car 1611.6320781012555 13
13	8f7ec8c75aa6137da338a8fb4a52bb6f8cf67dfc53528e23170e4fa89efc1a5d	2824.2010577318233 613.6922336831544 -18.145660155293427 1.797 4.561 1.505 2.552055803060721 car 2801.247720555521 59
14	6ae782254574cee52863efb671a6208dd0f31b45fa5f1e5912e797eb5e4888f7	1309.9044190361674 1515.5572666303347 -23.194225009483148 0.922 0.964 2.947 2.547567896526841 pedestrian 1239.53084013
15	242f0b3546df11d475f48914771db919f7f5c740d5a4095a9c6d3d7dbe603c20	2015.0095356876025 1123.532862624399 -17.78072365737183 2.376 6.78 2.807 2.6043345540499576 other_vehicle 2048.556551
16	789075fb42438dfef881cecb9a1ed781738a027aa8c18e9b6b92b0748a6c4059	1136.7746676290078 1648.8907433243678 -23.414624762338306 2.033 4.513 1.586 2.5638484185967165 car 1112.4290609820887

The following shows some examples captured by the host cars and will be used as our training and testing data. We have a total of 158,757 images for training data and 192,276 for testing data.

The frequency count of the classes is shown below.



The provided plot clearly indicates that a majority of the observed objects are cars. While this observation can align with the reality of more cars being present on the roads compared to other types of vehicles, it is essential to consider the potential limitations of the LiDAR sensor as well. LiDAR sensors are proficient at detecting and capturing information from solid objects, such as cars, due to their prominent reflective surfaces. However, they may face challenges in accurately perceiving certain objects that have different properties or shapes, such as bicycles or pedestrians. Hence, the dominance of cars in the plot might partially be attributed to the sensor's limitations in effectively capturing and recognizing other types of objects.

- Waymo

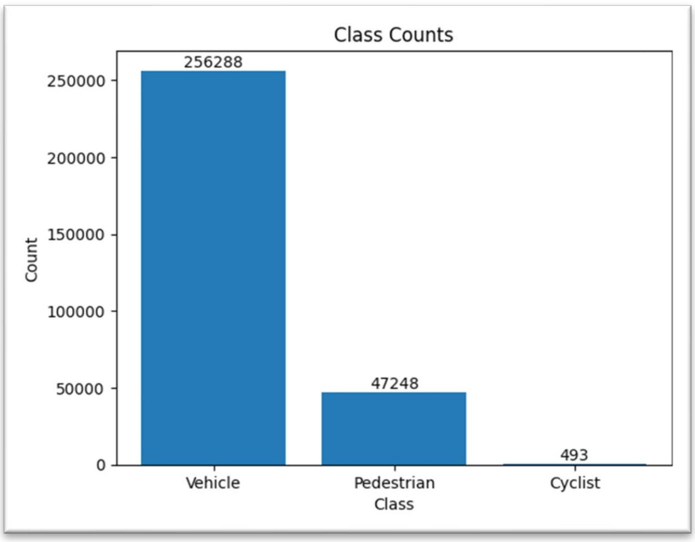
Waymo Open Data is a public dataset released by Waymo, a self-driving technology company owned by Alphabet Inc. The dataset is designed to facilitate research and development in the field of autonomous driving. It provides a large collection of high-resolution sensor data, including lidar and camera data, captured by Waymo's self-driving vehicles in various real-world driving scenarios.

The Waymo Open Data includes data from diverse urban and suburban environments, encompassing different weather and lighting conditions. It covers a wide range of driving scenarios, such as residential areas, busy city streets, highways, and complex intersections. The dataset offers a comprehensive representation of the challenges faced by autonomous vehicles in real-world driving scenarios.

The sensor data in the Waymo Open Data is carefully annotated and labeled, providing ground truth information about various objects and their attributes, including vehicles, pedestrians, cyclists, traffic signs, and road markings. These annotations enable researchers to develop and evaluate algorithms for object detection, tracking, and scene understanding.

In addition to sensor data and annotations, Waymo Open Data also provides detailed calibration information for the sensors, which is essential for accurately aligning and synchronizing the data from different sensors.

The frequency count of the classes using here is shown below.



Setup & Experiments

- Kaggle Competition: Lyft 3D Object Detection for Autonomous Vehicles
    a. Hardware Spec. & Software Version

System Platform	Linux
CPU	Intel i9-12900

GPU	Nvidia RTX 4090
Python	3.10.6
Pytorch	1.13.0+cu117
GCC	9.3
CuDNN	8.5
TorchVision	0.14.0+cu117
OpenCV	4.6.0

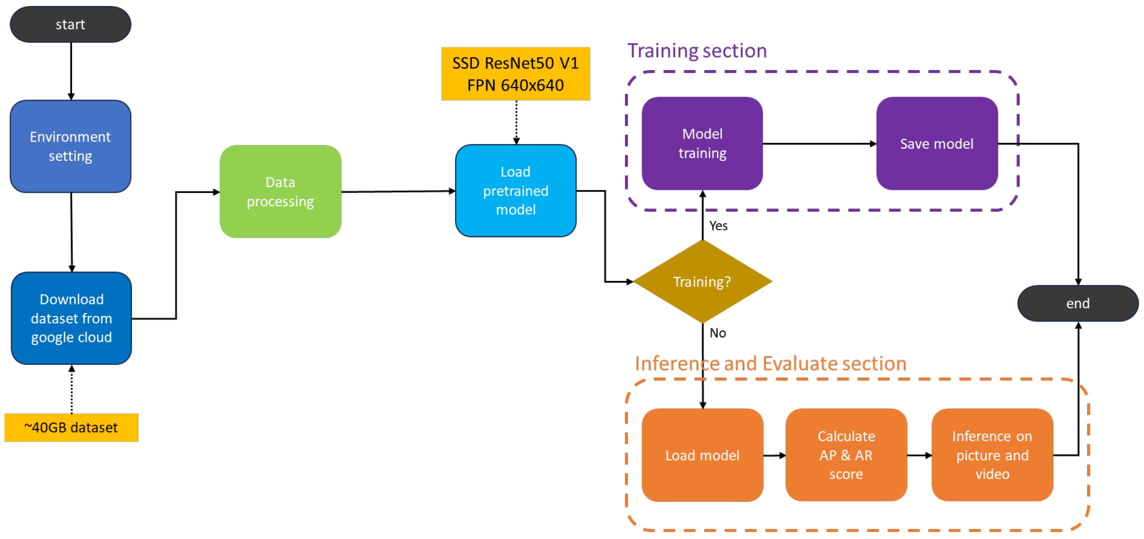
b. Implementation Details

- Waymo

a. Hardware Spec. & Software Version

System Platform	Linux
CPU	2*Intel(R) Xeon(R) CPU @ 2.20GHz
GPU	From colab
Python	3.10.11
tensorflow	2.12.0
CuDNN	11.8
waymo-open-dataset	tf-2.11.0==1.5.0

b. Implementation Details



2D object detection flowchart

Environment setting

This implementation uses Colab and utilizes the Waymo Open Dataset. Therefore, it was necessary to install the Waymo environment. Additionally, TensorFlow 2 Object Detection API provides the pretrained model. The COCO API was installed for evaluating the model's AP/AR performance.

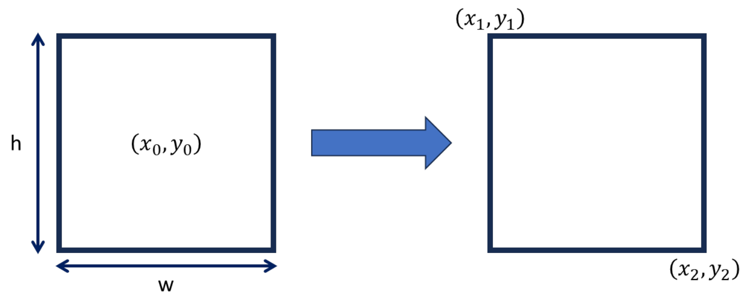
Data preprocessing

The input data is sourced from the Waymo dataset, which is stored in tfrecord files. Each tfrecord file contains captured images, Lidar information, class labels, bounding boxes (bb_box), and more. The data preprocessing steps are as follows:

In the class labels, there are five categories: Unknown, Vehicle, Pedestrian, Sign, and Cyclist. Since the goal of this implementation is on recognizing transportation, only the Vehicle, Pedestrian, and Cyclist classes are retained.

The Lidar information is not used in this implementation and is therefore discarded.

The bb_box format in the Waymo dataset is as bottom-left diagram, To facilitate model training, the format is converted to a format similar to how images are accessed in Python (as shown in the bottom-right diagram):



Train model

The model used in this project is SSD ResNet50 V1 FPN 640*640, which is a 2D object detection model. As the name implies, it combines the techniques of SSD, ResNet50, and utilizes the Feature Pyramid Network (FPN). ResNet50 is a variant of RetinaNet. And FPN is a feature pyramid network used for multi-scale object detection, which can detect and classifying objects at different scales. This model takes input data in the form of images with a size of 640*640.

The model was downloaded from the TensorFlow official model zoo, specifically from the object detection models. It is a pre-trained object detection model that comes with pre-trained weights. The Waymo dataset was then used as the input data to further train this model specifically for object recognition in self-driving cars.

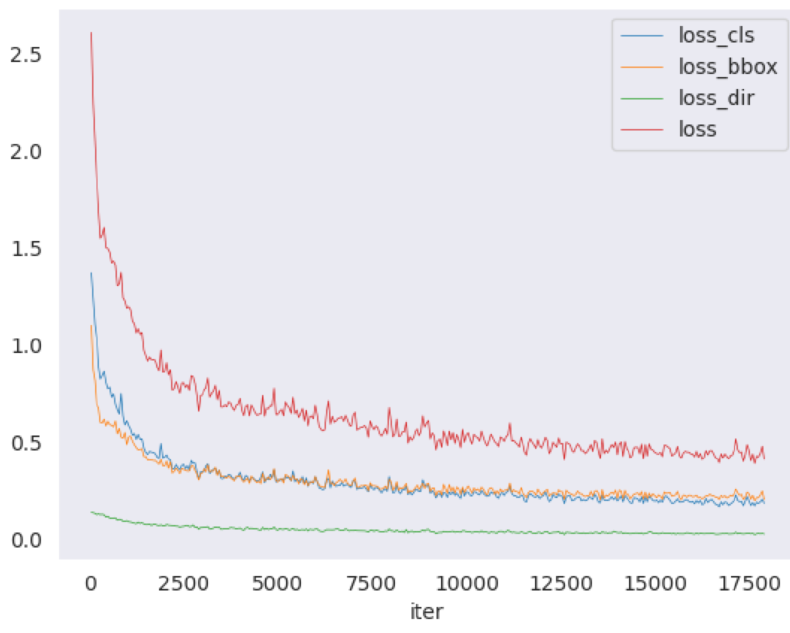
During training, the model was saved approximately per 6000 epochs, and its AP/AR performance was evaluated. Finally, the model was tested with actual images or videos, and if the results were not satisfactory, the model was continued to be trained until satisfactory performance was achieved.

At the end, the model was trained for 50,000 epochs, which took approximately 9 hours to complete the training process.

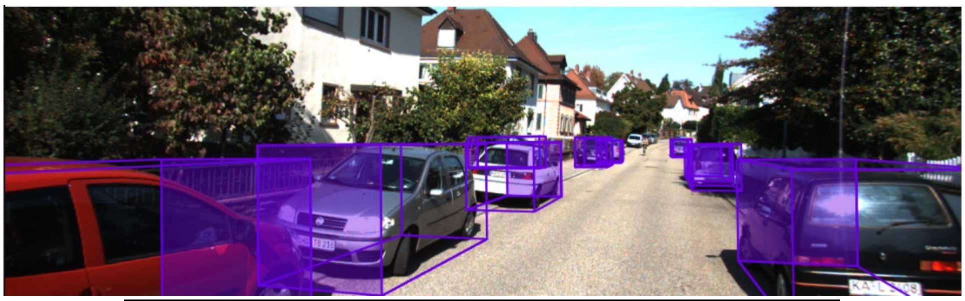
Results

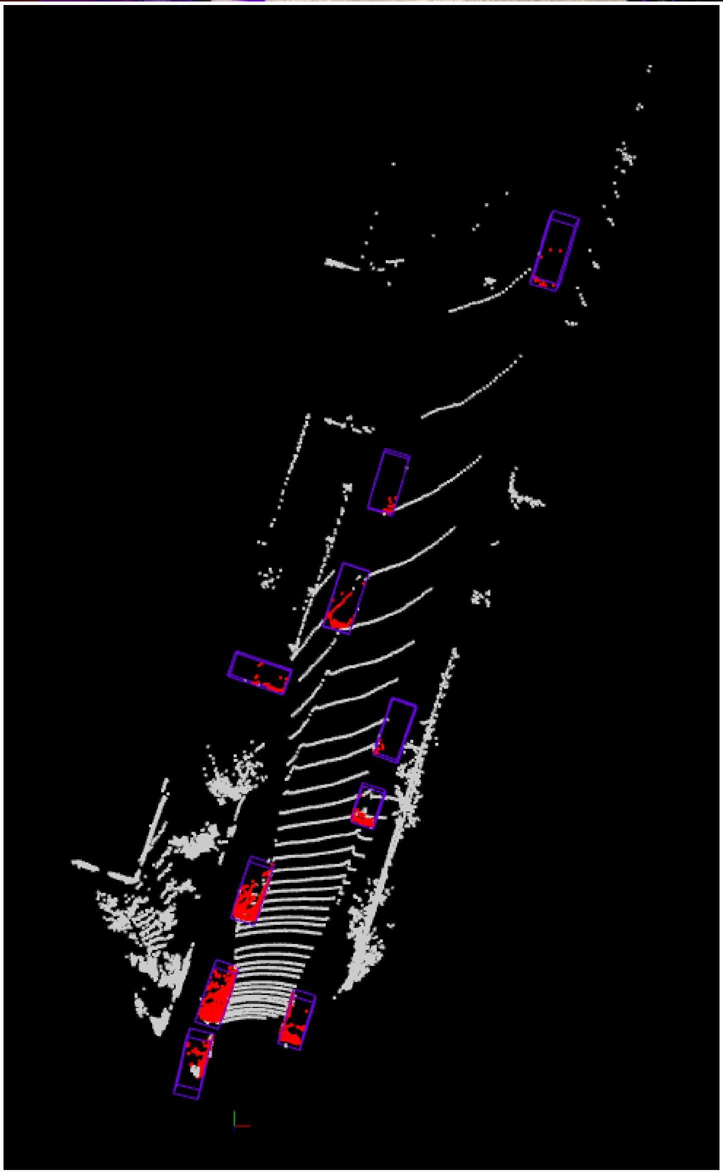
- Competition: Lyft 3D Object Detection for Autonomous Vehicles

Below image shows the training curve for 17500 iterations.



Inference results:

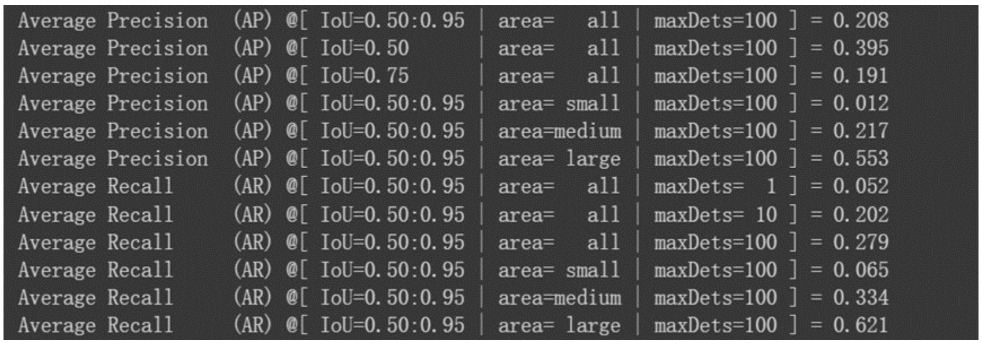




mAP 待補充---

- Waymo

Using Average Precision (AP) and Average Recall (AR) to evaluate：

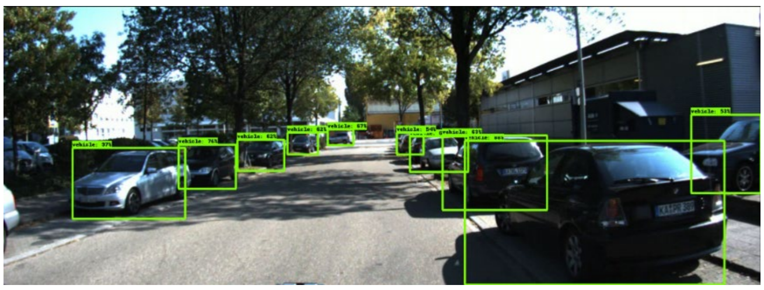


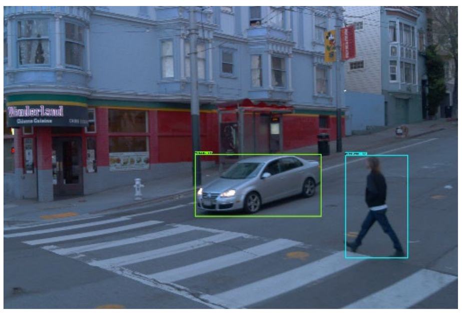
```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.208
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.395
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.191
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.012
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.217
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.553
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.052
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.202
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.279
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.065
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.334
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.621
```

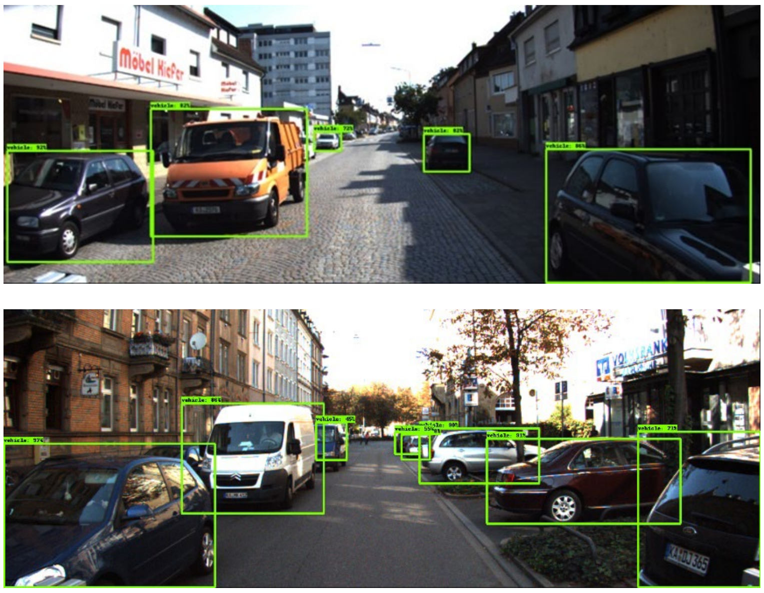
From the evaluation using COCOAPI shown in the figure above, it can be observed that the overall AP is not high, especially when the region is "small," where it can be as low as 0.25. This might be because in the context of self-driving cars, the objects to be recognized, such as vehicles and pedestrians, tend to have larger sizes.

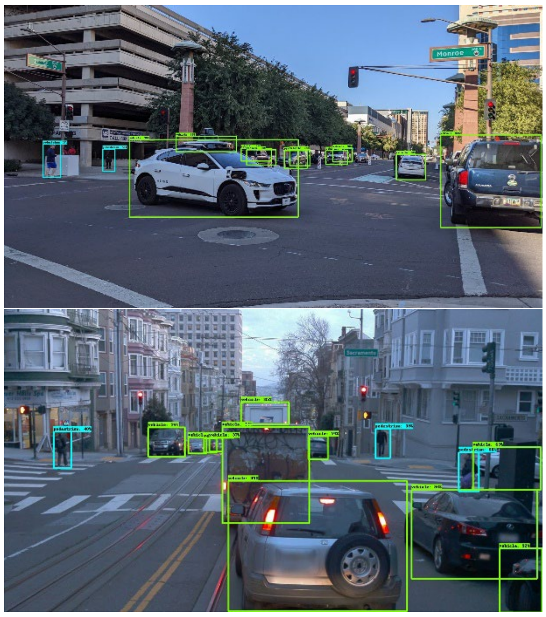
When filtering only "small" regions, most of the vehicles are naturally filtered out, resulting in a very small sample size for recognition. Consequently, this leads to a lower accuracy in predictions. Conversely, when the recognition region is "large," since all the cars in the image fall within this region and can be recognized with high probability, the vehicle features are effectively detected, resulting in higher accuracy, around 0.553.

The AR results are also similar. In our model, it struggles to recognize vehicles in "small" regions, resulting in an AR of only 0.052 for such regions. In other words, in a small region, the features of distant vehicles are condensed, resulting in poor recognition. This is also one of the reasons for the low AP. On the other hand, for objects in "large" regions, which are closer to the camera, their features can be well distinguished, leading to a higher AR of approximately 0.621.









Ablation Studies

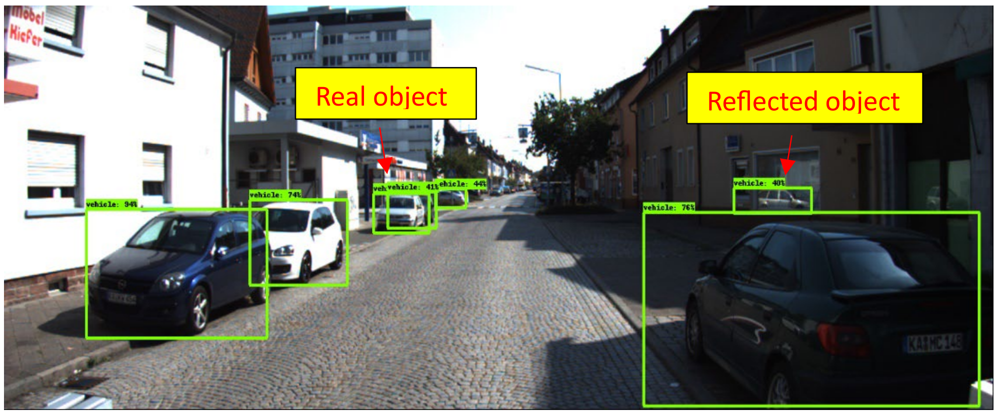
Challenge

- Waymo
    i. Our laptop does not have a GPU suitable for machine learning tasks.
    Solution: Conducting experiments in Colab

    ii. The dataset is too large to fit in Google Drive.
    Solution: Instead of uploading the data to the cloud, the data was directly uploaded to Colab. After preprocessing the data, the overall size has reduced, making it suitable for cloud storage.

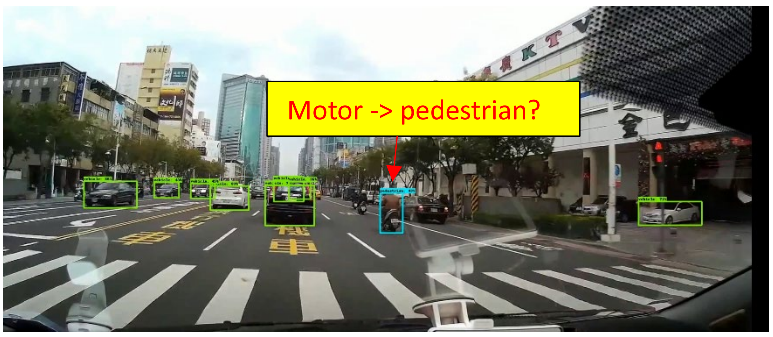
    iii. The model fails to detect the "cyclist" class during testing.
    Possible Solution: There are only 493 instances of cyclist data in our training dataset, making it difficult to train on. Trying to increase the training time or increase the number of instances in the cyclist class to improve the model's ability to recognize cyclist

iv. During testing, the model was evaluated in various environments and encountered a unique situation, as illustrated in the diagram below:



Though the vehicles were correctly identified, the glass windows of the buildings acted as mirrors under direct sunlight, causing the vehicles to be reflected and detected within the mirror images. This case is relatively rare, making it challenging for the model to learn for mirror reflections during training. Therefore, Extending the training time alone is ineffective in addressing this issue.
Possible Solution: Feed the model a large quantity of images containing vehicles reflected in mirrors. However, acquiring a substantial amount of relevant data for training purposes presents a significant challenge.

v. Since we are in Taiwan, we attempted to evaluate the model's performance in the Taiwanese environment, and the results are as follows:



It can be observed that motorcycles were identified as pedestrians. This could be attributed to the fact that our training data primarily consisted of the Waymo dataset, which was collected from foreign sources. Although the dataset includes vehicle labels, motorcycles are relatively rare in foreign environments and can resemble pedestrians in appearance. Consequently, they are classified as pedestrians.
Possible Solution: Train the model using other dataset that includes motorcycles class.

Conclusion

This project was built from scratch and aimed to be more realistic, which brought up many issues, especially related to environment setup: large amounts of data, incompatible package versions, and data preprocessing. Fortunately, all these issues were successfully resolved.

Since each person has different computer configurations, sharing processed data required conversions. Therefore, in the end, we decided to use Colab for the Waymo part to achieve better integration of the output results. This experience taught us the importance of having a unified approach in collaborative projects, which significantly improves work efficiency. In the future, when collaborating on projects, we will be better equipped to understand the situation and integrate smoothly.