

COMPARISON OF 3D OBJECT DETECTION METHODS FOR PEOPLE DETECTION IN
UNDERGROUND MINE

by

Yonas Dwiananta Yuwono

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mining and Earth Systems Engineering).

Golden, Colorado

Date _____

Signed: _____

Yonas Dwiananta Yuwono

Signed: _____

Dr. Sebnem H. Duzgun
Thesis Advisor

Golden, Colorado

Date _____

Signed: _____

Dr. M. Stephen Enders
Department Head
Department of Mining

ABSTRACT

Autonomous vehicles have received immense attention in the mining industry nowadays. However, there is limited research on 3D object detection in the underground mine. This thesis wants to compare the ability of 3D object detection models in the underground mine environment. Three state-of-the-art 3D object detections are analyzed to detect people in the underground mine. The author collects 1000 point cloud files from Edgar mine to train and test the algorithm performance. Data labeling and preprocessing methods are discussed to convert raw point clouds to the algorithm's format. Then, several training parameters such as the number of datasets and epochs are analyzed to obtain the maximum performance of object detection methods. PV-RCNN has the highest average precision for the study case in the underground mine. All datasets, source code, and train test split are accessible at <https://github.com/karana0103/EdgarObjDetection> for future use cases.

TABLE OF CONTENTS

ABSTRACT	iii
LIST OF FIGURES	vi
LIST OF TABLES	vii
ACKNOWLEDGMENTS	viii
CHAPTER 1 INTRODUCTION	1
1.1 3D Object Detection System	3
1.1.1 Point Cloud	3
1.1.2 3D Object Detection Algorithm	4
1.1.3 State-of-the-Art 3D Object Detection Algorithms	6
1.1.4 Transfer Learning	7
1.1.5 Evaluation Metrics	8
1.1.6 Object Detection, Autonomous Vehicles and Robots in Underground Mines	9
1.2 Research Goal	10
1.3 Research Questions	11
1.4 The Original Contribution of the Research	12
1.5 Structure of the Thesis	13
CHAPTER 2 METHODOLOGY	14
2.1 Selection of 3D Object Detection Algorithms	14
2.1.1 Part- A^2	15
2.1.2 PV-RCNN	17
2.1.3 Pointpillars	19
2.2 Data Collection	20
2.3 Data Preprocessing	20
2.4 Labeling	20
2.5 Transfer Learning	21

2.6	Epochs Analysis	21
2.7	Different Number of Dataset Analysis	22
2.8	Shuffling Analysis	22
2.9	Independent Dataset Testing	22
2.10	Assessment of the Result	23
CHAPTER 3 DESCRIPTION OF THE CASE STUDIES		24
3.1	Edgar Mine	24
3.2	Edgar Dataset	26
3.3	KITTI dataset	26
3.4	Comparison Between Edgar and KITTI Dataset	27
3.5	Preprocessing	28
CHAPTER 4 RESULTS AND DISCISSION		32
4.1	Results on Transfer Learning	32
4.2	Results on Epochs Analysis	32
4.3	Results on Different Numbers of Training Datasets	34
4.4	Results on Shuffling Analysis	35
4.5	Results on Independent Test	36
4.6	Best Case Results	38
4.7	Interpretation on Autonomous Vehicle Application	38
CHAPTER 5 CONCLUSION AND RECOMMENDATION		40
5.1	Conclusion	40
5.2	Recommendations for Future Research	41
REFERENCES		43
APPENDIX A PREPROCESSING.PY		46
APPENDIX B CONVERTER_LABEL.PY (ZHENGXUE ET AL., 2020)		47
APPENDIX C TRAINING AND TESTING ALGORRITHM (OPENPCDET DEVELOPMENT TEAM, 2020)		50
APPENDIX D PERMISSION TO REUSE FIGURES AND TABLE		70

LIST OF FIGURES

Figure 1.1	Edgar Mine point clouds	3
Figure 1.2	Taxonomy of 3D object detection	5
Figure 2.1	Steps of research methodology	14
Figure 2.2	PV-RCNN Architecture (source: Shi et al., 2020)	18
Figure 3.1	Edgar mine map with red circles indicating the place of Edgar datasets were taken	24
Figure 3.2	Husky robot for data collection in Edgar Mine	25
Figure 3.3	Point clouds of Edgar dataset	27
Figure 3.4	Point clouds of Edgar dataset	28
Figure 3.5	Various locations of KITTI dataset (source: Geiger et al., 2013)	29
Figure 3.6	(a) Total area histogram in KITTI training set (b) Total area histogram in KITTI testing set	29
Figure 3.7	(a) Total area histogram in Edgar training set (b) Total area histogram in Edgar testing set	30
Figure 3.8	(a) Point density histogram in KITTI training set (b) Point density histogram in KITTI testing set	30
Figure 3.9	(a) Point density histogram in Edgar training set (b) Point density histogram in Edgar testing set	30
Figure 3.10	(a) People histogram in KITTI dataset (b) People histogram in Edgar dataset	31
Figure 3.11	Comparison of the point clouds in (a) Edgar dataset and (b) KITTI dataset (source: Geiger et al, 2013)	31
Figure 4.1	Result of training and testing of Part- A^2 algorithm with Edgar dataset.	33
Figure 4.2	Result of training and testing of PV-RCNN algorithm with Edgar dataset.	33
Figure 4.3	Result of training and testing of PointPillars algorithm with Edgar dataset.	34
Figure 4.4	Training AP on different numbers of datasets.	34
Figure 4.5	Testing AP on different numbers of datasets.	35
Figure 4.6	Histogram of people’s height in Edgar dataset obtained in 2021.	37
Figure 4.7	Histogram of people’s height in Edgar dataset obtained in 2019.	37

LIST OF TABLES

Table 1.1	Level of automation for autonomous vehicles (source: Balasubramaniam et al., 2022)	1
Table 1.2	Definition of true positive, false positive, false negative, and true negative based on IoU threshold	9
Table 2.1	Comparison of open-source state-of-the-art 3D object detection	15
Table 2.2	Specification for training and testing models	22
Table 2.3	Training and testing split	22
Table 3.1	Specification of Husky unmanned ground vehicle	25
Table 3.2	Comparison of sensor for KITTI dataset (Velodyne) and Edgar dataset (Ouster)	26
Table 4.1	Comparison of testing result on the Edgar dataset (left) and KITTI dataset (right)	32
Table 4.2	Average precision of shuffled and non-shuffled dataset.	35
Table 4.3	Average precision with independent, mixed, and KITTI test set.	36
Table 4.4	Overall performance of three methods of object detection.	38

ACKNOWLEDGMENTS

I thank all who contributed to completing this thesis in one way or another. First, I thank God for his protection and ability to finish this research. I am so grateful to the PT Bukit Asam Tbk scholarship for making it possible for me to study here. I would like to express my sincere gratitude to my advisor, Dr. H. Sebnem Duzgun, for her wisdom, clear guidance, and endless support throughout this study. I would also like to thank the thesis committee members, Dr. Jurgen Brune and Dr. Andrew Petruska, for their invaluable criticism, suggestions, and guidance for my thesis study. Special thanks to my family for the encouragement that helped me complete this paper. I also thank My beloved wife, Erika, and my lovable child, Alora, who served as my inspiration to pursue this undertaking. Lastly, I want to thank my friends and colleagues, Jaime, Hakan, Sriram, Cagdas, and particularly those from Indonesia Students Association (PERMIAS), for every support, help, motivation, inspiration, and experience we shared.

CHAPTER 1

INTRODUCTION

Artificial intelligence (AI) has improved human life quality and created innovation across business sectors. Industries of all sizes, from the large-scale automotive industry to small-scale businesses, utilize artificial intelligence to make operations more efficient and effective. E-commerce, for example, uses AI-based personalized ads to get a higher sales percentage per user by detecting user behavior and predicting the item they need to buy. Meanwhile, the healthcare industry uses AI to detect diseases earlier and cure people better by leveraging AI's ability to detect specific disease features and predict if someone is infected or not.

One of the most important applications of artificial intelligence to date is its use in autonomous vehicles (AVs). SAE International, a global association of technical experts in the aerospace, automotive, and commercial-vehicle industries, categorized vehicle autonomy in six degrees, as given in Table 1.1. In levels 1 and 2, the vehicle provides features to support steering and acceleration for the driving experience; however, they still rely on the human driver to make decisions. Level 3 vehicles allow the autonomous system to operate in several conditions; however, particular conditions need human intervention to control acceleration and steer safely. Level 4 vehicles are designed to have a full self-driving mode in several specific conditions; however, they will stop if the conditions are not met. Level 5 vehicles can drive without human intervention in all conditions. From levels 3 to 5, an advanced driver assistance system (ADAS) should be installed. ADAS is a group of technologies that assist drivers in operating the vehicle. One key aspect of ADAS is perception technology, which enables the vehicle to detect nearby obstacles and respond accordingly. The main challenge of achieving excellent environmental perception is to detect objects using a software-based object detection algorithm.

Table 1.1 Level of automation for autonomous vehicles (source: Balasubramaniam et al., 2022)

SAE Level	Name	Driving Environment Monitor
0	No Automation	Human Driver
1	Driver Assistance	
2	Partial Driving Automation	
3	Conditional Driving Automation	ADAS System
4	High Driving Automation	
5	Full Driving Automation	

Object detection is a branch of computer vision tasks that detects objects such as pedestrians, traffic signs, or other vehicles. It is the main aspect of autonomous vehicle operation. Object detection is used as a means of perception to see and recognize the surrounding objects and the vehicle's path. It is also used for event detection and motion control. High-level tasks such as object tracking and collision prevention rely on the object detection quality.

The basic approach to creating object detection for autonomous vehicles is by using a large dataset to train the model so the model can learn and recognize the objects around the vehicle. Deep learning-based approaches are commonly used to extract the features and recognize the objects. In general, the algorithm consists of an encoder and decoder. An encoder takes the input and processes it through several layers and blocks to learn and extract its features. Then the output is passed to the decoder to predict bounding boxes and labels for the surrounding objects.

There are three common types of input for deep learning-based object detection, namely image-based, point cloud-based, and hybrid-based object detection. Image-based object detection uses pixels in the image as an input to be processed by the algorithm. Point cloud-based object detection uses points generated by 3D sensors such as LiDAR as an input. Hybrid or fusion-based object detection uses both image and point cloud as a method input.

Image-based object detection has several disadvantages for autonomous vehicles in the mining environment. Dust is inevitable and sometimes blocks the camera from producing good vision for the vehicle. Image-based object detection is also not good in dark environments, such as in underground mines or open-pit mines with poor luminance at night. Thus, this problem could be handled with point cloud-based object detection. Point cloud-based object detection is also beneficial as it gives the object's information in 3D, so the algorithm not only detects the object but also provides more information, such as the precise distance and size of the object.

Development of underground mining AVs and robots is an emerging area. The AVs are used to reduce cost and create a better mining operation. They are also used to mine in dangerous environments, preventing human injury and fatality. Underground robots can also be used for underground mine rescue in case a roof failure occurs in the mine (Tung et al., 2020).

Like in other areas of the AV space, object detection is important for the development of underground mine autonomous vehicles. The underground environment is unique and complex not only because of the limited lumination but also the limitation of GPS accuracy. This forces autonomous vehicle development to rely on the accuracy of the object detection algorithm as the main perception system.

The main challenge to developing autonomous vehicles for underground mines is the workers' safety. The AVs need to operate effectively to reduce operational costs and improve the safety of people in the

underground mines. Thus, in this thesis, the author has chosen to analyze object detection to detect people in underground mines. Several state-of-the-art algorithms are analyzed and used for the detection and identification of people.

1.1 3D Object Detection System

1.1.1 Point Cloud

There are several methods of 3D information acquisition. In the beginning, people used stereo vision consisting of two or more calibrated cameras to obtain the 3D representation. The development of RGB-D cameras and LiDAR has since made data acquisition in the 3D world more accessible and efficient. The common format to store the information from a 3D sensor is to store it in point clouds.

A point cloud is a set of data points in space. Each point consists of x , y , and z axes position coordinates, plus additional information such as color or the object's intensity in the area. Usually, hundreds to thousands of points are in a point cloud file. Unlike 2D images, which a regular grid can represent, 3D point clouds are dense in some areas and sparse in others. Point clouds are also irregular and unordered by nature. The combination of many points and those point cloud properties make 3D object detection more challenging than 2D object detection. Figure 1.1 shows an example of point clouds in Edgar Mine obtained using a LiDAR sensor.

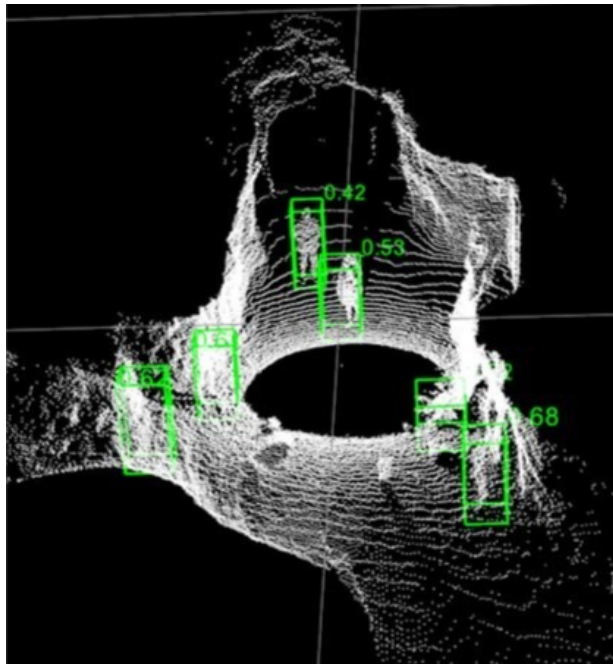


Figure 1.1 Edgar Mine point clouds

LiDAR is an abbreviation for Light and Detecting Range. It has a similar concept as radar and sonar. However, instead of radio or ultrasonic waves, LiDAR uses light as the medium to measure the range. LiDAR works based on emitting the light or laser and measuring the reflection time. Thus, from the reflection time, the distance can be measured. The basic measurement of the distance is given in the following equation:

$$r = \frac{c \times t}{2} \quad (1.1)$$

r = distance the pulse travelled

c = speed of light

t = time taken for the pulse of light returned with calibration of the angle encoders

1.1.2 3D Object Detection Algorithm

There are two categories in object detection algorithms: 2D and 3D object detection. 2D object detection is the common one, as there is much availability of 2D images from various sources. However, the 3D world we live in cannot be fully represented in two-dimensional images. Such images cannot fully project depth and position information precisely. Due to these factors, 3D object detection research has been emerging and rising significantly in recent years, especially in autonomous vehicle applications.

In the early years of 3D object detection development, people used traditional handcrafted methods to extract the information from the point clouds. For example, they used FPFH or SHOT as the feature descriptor (Shan et al., 2021). Then, they used SVM (Clement et al., 2011) or random forest (Nesrine et al., 2009) as the classifier. The handcrafted approach has several advantages; for instance, there is no need for vast training data, and it is fast. However, handcrafted methods are not as good as deep learning-based object detection methods.

Deep learning is a branch of artificial neural networks that can learn features in a way similar to the human brain. It uses multiple processing layers to learn and recognize patterns from the training set. However, it requires a huge amount of data to achieve high accuracy. As access to point cloud files is easy nowadays, deep-learning-based object detection systems are growing more popular.

Development of 3D object detection using deep learning started by imitating how 2D object detection works. As 2D image features can be extracted and recognized effectively using convolutional neural networks (CNNs), the early phase of 3D object detection utilized CNN in their architecture (Daniel et al., 2015). The convolution operation needs a structured grid, but point cloud data are normally unstructured. Researchers convert the point cloud data into a structured form to tackle that problem. However, the

recent research utilizes deep learning to directly learn features from point clouds without converting to a structured grid. The other approach is combining the two methods such as PV-RCNN algorithm. Thus, the author categorize 3D object detection methods into structured grid-based learning, direct raw point cloud learning and fusion-based learning as given in Figure 1.2.

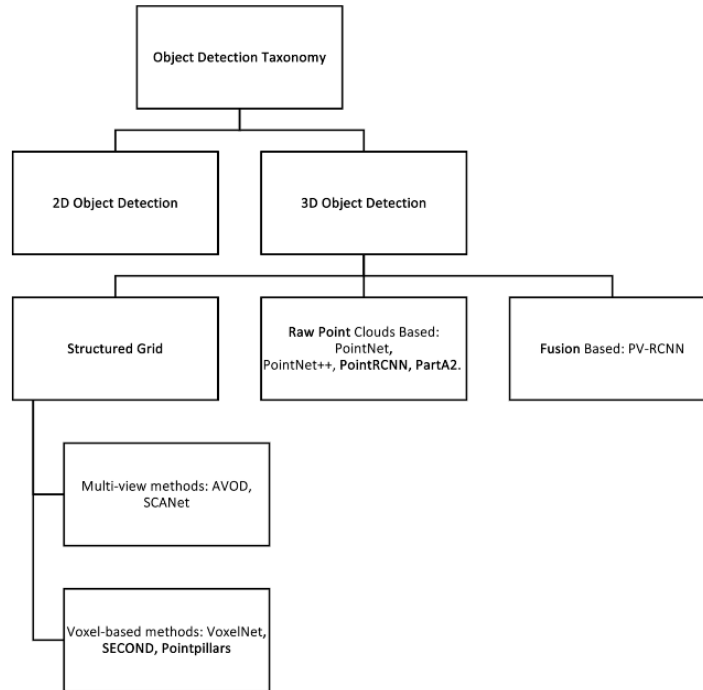


Figure 1.2 Taxonomy of 3D object detection

There are two approaches in structured grid-based 3D object detection: voxel-based and multi-view-based. In voxel-based learning, the points are converted to 3D voxel structures. The typical voxelization process scans the area and detects if there are points in it. It returns 0 if there is no point and 1 if points are detected. Then, the preprocessed point clouds are transferred to the various 3D convolutional, pooling, and fully connected layers for the learning stage.

However, the convolutional process is expensive computationally because of large amounts of data and sparsity of the point clouds. This is because the CNN layer should analyze areas with no points as well as dense areas. The researchers used a new approach called sparse convolution to tackle that problem. The sparse convolution method uses a similar approach to standard 3D CNN, but instead of scanning all areas, this method only focuses on the areas with points.

In multi-view-based learning, the algorithm converts the point cloud data to several projections of 2D images. It utilizes the mature 2D convolutional neural network for training and predicting the object.

Multi-view-based networks have a more efficient computational cost because they use efficient 2D techniques. They also perform better because they contain richer information than the sampled voxelization grid. However, the 2D projection images only capture the object's outline and do not handle occlusion very well. On some occasions, such as in the underground mine, occlusion of the roof is eminent and cannot be discarded.

The recent approach to 3D object detection uses raw points without any conversion to a structured grid. This approach recognizes localization information better than the fine-grained voxelization method. The idea became popularized after PointNet's publication was released. PointNet pioneered the raw points object detection method and became the basis for most other algorithms. Its architecture has two primary functions: multilayer perceptron (MLP) with learnable parameters and a maxpooling function. The MLPs are feature transformations that transform three-dimensional data to 1024-dimensional space. Then the maxpooling function is used to obtain one global 1024-dimensional feature vector. The feature vector can be used for recognition and segmentation tasks, as represents the feature descriptor of the input. PointNet then achieved state-of-the-art performance on ShapeNet part and Stanford 3D semantic parsing dataset [6]. Subsequently, many researchers developed the algorithm by capturing local structures. It represents the feature descriptor of the input.

1.1.3 State-of-the-Art 3D Object Detection Algorithms

Apart from 3D object detections that has been mentioned in previous section, there are several state of the art 3D object detection methods available. Shi et al. (2020) propose a graph neural network to detect object. The algorithm utilize a fixed radius near-neighbors graph to encode the point cloud directly without creating voxelization structures. This method rank as the highest average precision on Lidar-only based in KITTI benchmark in pedestrian category. This method recognize interconnectivity between points from the graph representation. They proposed an auto-registration mechanism to minimize translation variance. If more than one boxes generated in the same region, the method merge it to create a better scoring on the detection.

Wang et al. (2019) propose frustum-based methodology, named F-ConvNet, to divide 3D point clouds into frustums. A frustum is a portion of solid cone or pyramid that lies between one or two parallel planes cutting it. They used RGB image object detector to create point cloud regions to be used by 3D object detector models.

Chen et al. (2020) propose interesting method, named HotSpotNet, in 3D object detection as the method can handle occlusion and truncation object better than any other objects. They distinguish non empty voxels as spot and learn a small subset of spots in each object as Hotspots. Based on the object as

hotspot, a novel anchor-free detection head proposed, resulted a high precision on the pedestrian class in KITTI dataset. HotSpotNet is interesting method as in the mining environment, the objects are usually blocked by several object and the algorithm can handle it well. However, they do not provide an open-source code so it cannot be reproduced in the Edgar mine dataset.

Li et al. (2021) presented From voxel to point: a two-stage 3D object detector with a voxel-to-point decoder and Intersection over Union (IoU) guided algorithm. They used voxel-to-point decoder to extract the point features in addition to the map features from voxel based Region Proposal Network (RPN). They used 3D region of interest alignment to align the features with the proposal boxes. Then it aggregated with the corner geometry embeddings in the box refinement stage.

Ku et al. (2018) present AVOD-FPN, an Aggregate View Object Detection with Feature Pyramid Network. They used multimodal data as their training input. They used a region proposal network (RPN) and a second stage detector network. The RPN is able to perform fusion of the multimodal input on high resolution map. The proposal then transferred to the second stage detection network to generate the output.

Ku et al. (2019) also proposed object detection algorithm with virtual multi-view synthesis orientation estimation. The research focus on improving the orientation estimation results on pedestrian as it is important on autonomous vehicles application. The virtual multi-view synthesis module is used to improve orientation estimation. They put virtual cameras around each object to generate novel viewpoints. This methods greatly improves the orientation estimation on pedestrian.

Le et al. (2021) recently present PiFeNet, the pillar-feature network specialized design for pedestrian object detection. They used stackable Pillar Aware Attention (PAA) module for enhanced pillar feature that is naturally not too expressive. The module is also suppressing noises in the point clouds. Then they integrate multi-point-aware pooling, point-wise, channel-wise, and task-aware attention for feature detection. The methods generate a high precision score in KITTI dataset and rank 1st in easy difficulty pedestrian detection for BEV and 3D. However, they are not published the open-source code yet, so it cannot be used in the Edgar dataset.

Pang et al. (2022) used fusion based multimodal data as the input of Fast-CLOCs. Contrary to the common multimodal algorithm, Fast-CLOCs network can run high-accuracy in near real time. It is because Fast-CLOCs operates on the output candidates before Non-Maximum Suppression. A lightweight 3D detector-cued 2D image detector is also used to extract visual features from the image domain.

1.1.4 Transfer Learning

One characteristic of deep learning is the need for large amounts of training data. The training data should be well obtained and annotated. Moreover, the data should represent different case scenario

mimicking the real environment. Tan et al. (2018) offers deep transfer learning as the solution of limitation data. In transfer learning, the training data are not required as the model has been trained with another train set. It assumes that the pretrained model has learn features so well and can be applicable in a different test scenario. Transfer learning can solve the basic problem of insufficient training data.

With limitation of datasets in the underground mining environment, transfer learning approach becomes solution for object detection. If transfer learning can detect object in the underground mine as well in the common road environment, then the models can directly be used in underground environment. Lin et.al (2017) tried this approach to detect people in the underground coal mine. They used pretrained model YOLOv2 and assess the performance. Although the results are not good, it is naturally because the difficulty of the camera detecting people in low light condition. This thesis wants to try a similar approach by using pretrained model on detecting people in the underground mine.

1.1.5 Evaluation Metrics

Unified evaluation metrics are essential to make benchmarking easier. For 3D object detection, the most frequently used criterion is average precision (AP) (Mark et al., 2011). AP is obtained by calculating the area under the precision-recall curve. Precision is the ability of an algorithm to detect only the correct object. The equation of precision is given in the following equation, with TP as true positive and FP as false positive:

$$PRECISION = \frac{TP}{TP + FP} = \frac{TP}{ALL\ DETECTIONS} \quad (1.2)$$

Recall is the ability of an algorithm to find all the ground truths available in a case. The equation of recall is given in the following equation with FN as false negative:

$$RECALL = \frac{TP}{TP + FN} = \frac{TP}{ALL\ GROUNDTRUTHS} \quad (1.3)$$

True positive, false positive, true negative, and false negative can be obtained by calculating intersection over union (IoU). This measure evaluates the overlap between the predicted bounding box and the ground truth bounding box. In the people object detection case, the threshold for IoU is 50%. If the model can detect an object with an IoU of more than 50%, it counts as true positive, and vice versa. The definition is given in Table 1.2. The equation of IoU is given in the following equation:

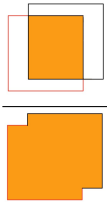
$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{img}}{\text{img}}$$

(1.4)

Table 1.2 Definition of true positive, false positive, false negative, and true negative based on IoU threshold

Terms	Description
True Positive	A correct detection ($IoU \geq 0.5$)
False Positive	A wrong detection ($IoU < 0.5$)
False Negative	A ground truth not detected
True Negative	Other area other than ground truth not detected

After knowing the precision and recall value, the ideal way to calculate average precision is by calculating the area under the precision recall curve. The equation is given below:

$$AP@{\alpha} = \int_0^1 p(r)dr$$
(1.5)

$p(r)$ = precision in a recall point.

However, KITTI benchmark (Andreas et al., 2013) and PASCAL VOC (Mark and John, 2011) use 11-point interpolated AP for easier calculation. The interpolated precision is calculated at each recall level by taking maximum precision on that recall point. The formula is given below:

$$p_{interp}(r) = \max_{r':r' \geq r} p(r')$$
(1.6)

Where $p(r')$ is the measured precision at recall r' .

After 8 Oct 2019, the method was changed to 40-point interpolation after critics from Simonelli et al (2019) which exposed 11-point interpolation flaws.

1.1.6 Object Detection, Autonomous Vehicles and Robots in Underground Mines

Object detection is the key aspect of autonomous vehicles and robots operation. In the mining environment, 2D image detection may be used in the proper illumination place. Lin et.al (2017) tried YOLOv2 architecture to be used as people detection in underground coal mine. The result is quite good

with 54.4 mAP in 43 fps. However, it can be done with training and testing dataset in the good illumination. The performance will drop if the algorithm meet low illumination and dusty places.

Li et al (2022) used improved YOLOv4 network to utilize infrared images for object detection. Infrared images are chosen because common camera might not perform well in large dust, uneven illumination environment in the underground mine. The images, firstly, preprocessed by super-resolution convolutional neural network to enhance their resolution. Then YOLOv4 as state-of-the-art algorithm in 2D images object detection is used for feature extraction. The results are good as it can achieve 96.25% precision in 48.2 fps. It is good and can be alternative on object detection in the underground mine. However, the information generated from 2D object detection is not as good as 3D object detection. The algorithm might has difficulties on accurately predict the location of the object. Thus, the application on autonomous vehicle in the underground mine is not recommended.

Object detection is vital for autonomous vehicles and robots perception systems. Mansouri et al. (2020) used two methods on micro aerial vehicle navigation purposes namely, 2D LiDAR vector geometry based and handcrafted convolutional neural network (CNN) of images. The algorithm then tested in the underground mine environment in Sweden. The results are that CNN based navigation is not as good as 2D LiDAR vector geometry algorithm. It is because the CNN relies on the image quality while in the underground mine, the illumination is very limited.

Dan et al. (2020) used fusion of LiDAR data and thermal vision for object search and exploration mission in underground mine. They used YOLO detector model for process the image and use 3D object localization algorithm to locate the detected object precisely. They are not using directly the 3D object detection and rely on the camera sensor for object detection perception. However, they used LiDAR sensor to measure the object location that is detected by the camera.

Bewley et al. (2016) used deep convolutional networks (DCN) to train the algorithm to detect people and vehicle from camera. The result however is not good. The algorithm perform poorly when presented with typical mining environment both in night and day time. The authors proposed several handcrafted improvement such as bayesian fusion framework between the output from DCN and background model training. The results slightly improve while the computational cost increasing dramatically.

1.2 Research Goal

The usage of 2D object detection based on images have some drawbacks at mining operation environment. Low lumination, heavy rain and dust are some condition that limit camera perception on surrounding objects. Therefore, 3D object detection can tackle those problems because the LiDAR sensor relatively not affected with low lumination and dust. However, studies about application of 3D object

detections are mainly discuss about its application in normal road condition. There is currently no comparison study of 3D object detection in the underground mine. This thesis compares state-of-the-art 3D object detections available with underground mine. So the development of autonomous vehicle in the underground mine can be advanced in the future.

To the best of the author’s knowledge, a published 3D people dataset on the underground mine environment is not yet available at the time of writing. This thesis provides more than 5,400 people objects in the underground mine. The datasets can be used for training and testing 3D object detection performance in the underground mine. This thesis also explains the format conversion from the raw dataset to the well-known dataset available so the algorithms can run the dataset.

1.3 Research Questions

The research questions of this thesis are listed below:

1. *How can we run a state-of-the-art 3D object detection algorithm with an underground mine dataset?*

There are several differences between the dataset from the open-space environment and the underground mine dataset. The underground mine dataset cannot be directly given to the algorithm built for the open-space environment. This thesis aims to answer the above question and explain the format transformation.

2. *Is transfer learning effective in detecting objects in the underground mine dataset?*

The pretrained model on KITTI datasets can detect pedestrians quite well. However, it is unclear if the model can detect people in the underground mine without any retraining, as while the size of people is relatively similar, the background and noises are very different between these two datasets.

3. *How many epochs are needed to generate high accuracy?*

When running a training process, the basic parameter is the number of epochs needed to generate a good amount of high accuracy. The need to identify the optimal number of epochs is crucial because large amounts of epochs consume more time and resources than smaller ones. Train and test accuracy are analyzed to prevent underfitting and overfitting in the trained model.

4. *What is the effect of increasing the dataset number?*

Deep learning needs a large dataset on which to train and recognize the pattern. However, the dataset number is linear with time consumption on labeling and preprocessing the data. The training time also depends on the number of training sets. However, a too-small dataset can cause the model to be overfitting or underfitting.

5. *What is the effect of shuffling the dataset to prevent underfitting or overfitting of the model?*

Deep learning algorithms process the information and create generalizations from the training data. If the datasets are in order, the generalization process will be biased and will not correctly learn the correct feature. Shuffling the training datasets becomes critical for a good model for non-biased object detection.

6. *Which algorithm is the best for 3D pedestrian detection in underground mines environment?*

From the three state-of-the-art 3D object detection methods used in the open space, it is interesting to discuss which one has the best performance in the underground mine dataset. The three object detection algorithms have different ways of processing the point clouds. Thus, we can analyze the best ways to extract the unique environment features in the underground mine.

7. *What is the performance of the algorithm with independent testing?*

The underground mine datasets were taken during two different time periods; one was taken in 2019 and the other in 2021. Training in the one set and testing it with the other independent dataset become critical to mimicking the actual use of the algorithm in the autonomous vehicle.

1.4 The Original Contribution of the Research

Deep learning performance relies on the quality and quantity of the dataset. If the dataset is not enough, a deep learning model cannot generalize the pattern and perform the way it should be. This thesis solves that problem by providing a significantly large labeled point cloud dataset for the underground mine environment. The dataset can be used for training and testing 3D objects in the underground mine and contribute to the development of autonomous vehicles technology in underground mines.

This thesis also contributes to developing autonomous vehicles in underground mines by comparing three state-of-the-art 3D object detection algorithms in the underground mine environment. There is no benchmarking tool or study about the performance of 3D object detection in underground mines yet. This comparison can become a reference for the further comparison and application of 3D object detection in underground mines.

State-of-the-art algorithms were built to use some datasets from a typical road or open space environment. The algorithm does not perform well in different datasets without any re-calibration and re-formatting of the custom dataset. Therefore, this thesis provides the code and guidance on preprocessing step to convert custom dataset point clouds to the KITTI dataset. More than four hundred 3D object detection algorithms use the KITTI dataset as their input. Thus, after preprocessing the custom dataset, the compatibility to use state-of-the-art object detection increases.

1.5 Structure of the Thesis

This thesis structure is divided into five chapters. Chapter two explains the methodology to finish this thesis. The methodology can be summarized as literature reviews, data collection and preprocessing, annotation of the dataset, and various training and testing methods to achieve the best results in object detection. Chapter three elaborates on the case study dataset as well the KITTI dataset as benchmarking tools for object detection performance. Chapter four analyzes the training and testing results of three state-of-the-art object detection methods in different scenarios. Chapter five presents the conclusion of the research and recommendation for future research.

CHAPTER 2

METHODOLOGY

This chapter explains the methodology of this thesis. Figure 2.1 illustrates the steps of the research methodology. Overall there are ten steps to complete in this thesis. First, is literature survey on 3D object detection algorithms. Then the authors collect data from the underground mine for testing and training dataset. The data need to be preprocessed and labeled to be used in the 3D object detection algorithm. After that, the process of evaluation start. The dataset is used for testing the performance of 3D object detection from transfer learning approach. Next, training process are conducted. An optimal number of epochs is analyzed to obtain performance to time efficiency in the training phase. Then, analysis about shuffling the training dataset is conducted to increase training convergence speed and overall precision number. The model then tested on the independent dataset, to review the performance on entirely new testing dataset. Lastly, the results are gathered and analyzed to be concluded.

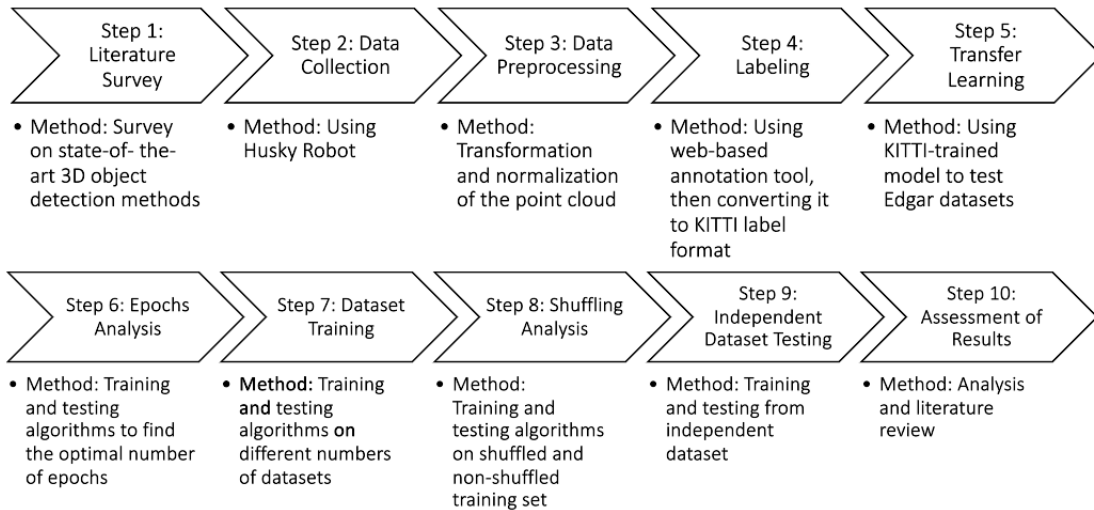


Figure 2.1 Steps of research methodology

2.1 Selection of 3D Object Detection Algorithms

Several methods of 3D object detection exist, as described in the introduction chapter. To choose the state-of-the-art algorithm to be used in this research, the author uses KITTI benchmark (Andreas et al., 2012) on 3D object detection as one of the best dataset provider and benchmarking tools for object detection algorithms. The author chooses the top seven algorithms that use LiDAR-only datasets and

provide open source code. The results are given in Table 2.1. Some open source code has not maintained well and the package library has grown outdated. Thus, the author is unable to reproduce the result even with the code modification. Point-GNN has maintained the code well, but several people have tried to reproduce the result with custom datasets and failed. The author of Point-GNN has not created guidance for modification on the code to be used in the custom dataset. On the other hand, the author of MMLab created a toolbox for 3D object detection and published it in <https://github.com/open-mmlab/OpenPCDet> (OpenPCDet Development Team, 2020). OpenPCDet is an open-source 3D object detection project that supports several models, namely Part- A^2 , PV-RCNN, PointPillars, SECOND, PointRCNN, and Voxel R-CNN. It is well maintained and provides guidance for running the model with a custom dataset. For that reason, this thesis analyzes Part- A^2 , PV-RCNN, and PointPillars for 3D object detection in the underground datasets.

Table 2.1 Comparison of open-source state-of-the-art 3D object detection

No	Method	Pedestrian AP (%)	Year Released	Code Maintenance	Custom Dataset Reproduction
1	Point-GNN	43.77	2020	Yes	No
2	F-ConvNet	43.38	2019	No	No
3	MMLab-Part- A^2	43.35	2020	Yes	Yes
4	MMLab PV-RCNN	43.29	2020	Yes	Yes
5	AVOD-FPN	42.27	2018	No	No
6	F-PointNet	42.15	2017	No	No
7	PointPillars	41.92	2019	Yes	Yes

2.1.1 Part- A^2

Part- A^2 net is a two-stage three dimensional object detection system. Stage-1 is the part-aware stage that it estimates precise intra-object part placements and learns point-wise characteristics. Stage-2 is a part aggregation parts that aggregates part data to raise predicted box quality. This approach produces three dimensional bounding boxes with the variables (x, y, z, h, w, l) , in which the center coordinates of the box are (x, y, z) and its height, width and length are (h, w, l) and l is the box orientation angle from a bird’s eye point of view. The network gains the ability to estimate intra-object parts and segment foreground coordinates in stage-I of part-awareness using segmentation masks and ground-truth object location labelling extracted directly from ground-truth three dimensional box labelling. It also does component estimation and foreground segmentation while providing 3D ideas from the raw point cloud.

Anchor-free and anchor-based methods are utilized for 3D proposal creation to handle a variety of situations. The anchor-free method uses memory more efficiently and is light-weight, whereas the anchor-based method provides greater recall rates at the cost of additional memory and computation costs.

For the anchor-free technique, 3D bounding box proposals are immediately created in a bottom-up manner by segmenting foreground coordinates and simultaneously producing three dimensional proposals from the estimated foreground coordinates. Because it doesn't need as many 3D anchor boxes in the full three dimensional region as other systems have, it saves a lot of memory. It creates 3D suggestions for the anchor-based approach using down sampled bird-view maps of features and hardcoded 3D anchor boxes at each point in space.

A newly designed RoI-aware point cloud pooling process is employed to reduce the uncertainty of the earlier point cloud pooling solutions. The Stage-II network gradually aggregates the pooled component characteristics of each three dimensional proposal for precise confidence estimation and refinement of the box, using sparse convolution and pooling operations.

For all three classes in the three dimensional object detection benchmark, the Part-A2 net surpasses all preceding LiDAR-only approaches on all levels of difficulty, as well as all earlier multi-sensor techniques on the critical "moderate" level of difficulty for both car and cyclist categories. For bird's view identification of car, pedestrian, and bike, this models outperform earlier sophisticated approaches by a wide margin on practically all stages of difficulty. On the KITTI three dimensional Object Detection Benchmark's three dimensional object detection leader board as of 15th of August 2019, the Part-A² -anchor net placed first among all approaches for the most significant automobile class, as well as first among all LiDAR-only techniques for the cyclist category.

In comparison to PointRCNN, the current approach Part-A² -anchor net greatly increases cyclist performance while achieving equivalent pedestrian results. The explanation for the slightly worse results on pedestrians could be that the pedestrian's orientation is difficult to discern from the sparse point cloud, which makes it difficult to forecast component placements in the Part-A² -anchor net. For recognizing small objects like pedestrians, multi-sensor approaches that incorporate RGB pictures might be advantageous.

Utilizing free intra-object position tags and foreground tags from ground-truth three dimensional box annotations, stage-I learns how to anticipate correct intra-object component placements. The second step of component aggregation can better gather the spatial data of object fragments, allowing for more exact scoring and location refining.

The new RoI-aware point cloud pooling module groups the expected intra-object part locations within the same proposal, resulting in a useful representation for encoding the geometry specific properties of each three dimensional proposal.

Unlike 2D image object detection, annotated 3D bounding boxes naturally and effectively separate 3D objects in autonomous driving scenes. Two 3D proposal generation techniques are presented to handle various situations. The anchor-free method conserves memory, but the anchor-based method increases

object recall. To address the ambiguity in existing point cloud region pooling techniques, a new RoI-aware point cloud region pooling procedure is used.

When compared to PointRCNN, Part-A2 -anchor net performs significantly worse on pedestrians. This could be because the pedestrian's direction is difficult to discern from the sparse point cloud, which makes it difficult to predict part placements in the Part-A2 -anchor net. For recognizing small objects like pedestrians, multi-sensor approaches that incorporate RGB pictures might be advantageous.

The most impressive performance On the KITTI validation dataset, the Part-A2 -anchor model records a number of false positives with varying score thresholds, which are caused by background confusion, poor localisation, and confusion with objects from other classes.

2.1.2 PV-RCNN

The proposed PV-RCNN algorithm works by integrating the voxel-based networks and also PointNet-based networks deeply for the 3D object to be detected using a two-step strategy to include the whole voxel to keypoint 3D scenario encoding and also the feature of keypoint to grid RoI featured abstraction. This framework helps in boosting the detection of 3D by utilizing the benefits of both features. It has the very principle of efficiently using the voxel-based methodology for encoding multi-scale feature descriptions. It can also contribute to generating quality 3D applications. On the other hand, PointNet based helps in preserving the location data by mending responsive fields. This method is also based on abstraction information which reduces the error and provides flexibility. The algorithm is aimed to achieve an even more accurate 3D detection system that can use point cloud data. The general understanding and depiction of the whole algorithm would be based on using the advantages of the two models and integrating them to provide an efficient proposal for 3D detection. The algorithm is also designed to utilize the RoI feature which features RoI grids and aggregates the scenario points to feature a confident prediction and refine the overall location of the target.

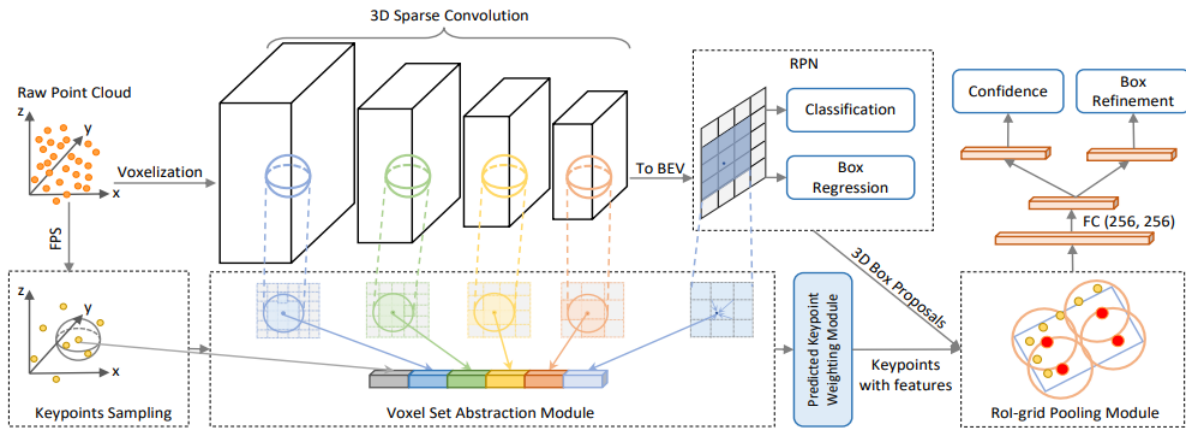


Figure 2.2 PV-RCNN Architecture (source: Shi et al., 2020)

The KITTI dataset has been known as a universally acclaimed dataset that has been used for 3D detection utilized for driving autonomously in self-driving cars. The PV-RCNN is then implemented and tested on the highly acknowledged KITTI dataset for results. The results of 3D detection by using the KITTI dataset outperformed the other methods by a very significant margin. The algorithm increased the mAP by more than 1% on all methods of easy, medium, and hard levels. Even for a very far method such as bird-view detection, it still showed significant margins on both easy and medium levels of difficulty while it dropped its performance slightly on the hard difficulty level. When the same bird-view detection method was used for a cyclist, then it completely outperformed the previous conventional LIDAR methods with a very high margin showing its grasp and capability on the field. While these results were only valid for the easy level, it proved the new concept to be more applicable and reliable. The algorithm has been ranked as to be the first among the conventional algorithms and has been proved to be more beneficial and reliable than the RGB and LIDAR combined and the LIDAR only methods.

The algorithm has been already accredited so far by the community. There are many aspects of this framework that make it unique and appropriate for the job. It clearly uses the advantages of two feature learning programs and integrates them to reduce the margin of error and increase the overall efficiency and accuracy. But yet it also carries out some of the disadvantages of both of the methods as well. The PV-RCNN boosts the recognition of the RGB spectrum and also utilizes the point cloud data to provide an accurate location. These methods have been optimized under the conditions of datasets such as KITTI etc. and have been trained on data sets trains which replicate only limited scenarios. But yet there still remains a significant amount of margin of error in the real world to be dealt with. There still have been some repercussions as the algorithm failed to satisfy the conditions of the experiment as it failed for hard difficulty self-driving and also failed on the medium and hard level for cyclists' bird-view 3D detection

phenomenon. These errors symbolize the cons of the algorithm and they should be assessed and rectified according to the requirement of the real world as that is the only way through which the program would get its introduction into real-life technology.

2.1.3 Pointpillars

In Pointpillars, researchers proposed an efficient point cloud encoder and a PointPillar network for complete end-to-end learning of a 3D object detection. Unlike PointNet which were heavily dependent upon 3D convolution, the PointPillars network uses 2D convolution. The PointPillars network operates in three phases.

In the first phase, the PointPillar network takes a point cloud and converts them into a pseudo-image for 2D convolutional process. The stacked pillars contains a set of pillars which are generated through the discretization of point clouds in x-y planes. Point clouds are sometimes scattered therefore the pillars appear to be empty. These scattered point clouds are utilized by applying a limitation on both empty and non-empty pillars where 'P' is denoted as a non-empty pillar set while the 'N' is denoted as point cloud per pillar. When the pillar contains less data to define the tensor size (D, P,N), then zero padding is applied but when it contains huge data points that are unable to fit inside tensor size then it is arbitrarily sampled. For an accurate computational result, PointNet is applied to generate the tensor output of size (C,P,N). After this encoding process, it returns back to the original position in order to form a 2D pseudo-image size (C,H,W) where H and W denotes the width and height of canvas. In the second phase called backbone, it has two further actions, one that generate object features in a very small pixels images while the other perform upsampling and concatenation of these features. For upsampling Batch Normalization and ReLU is applied. Moreover, the image also checked with the original pseudo image through transposed 2D convolution. Afterwards, this network uses a detection head similar to a single short detector (SSD) which is used to predict the 3D bounding boxes of the objects.

For the experimental result, researchers used the KITTI object detection dataset. All the samples contain lidar point clouds as well as images, but they used lidar points for training and later compared it with fusion methods that contained both lidar points and images as well. One network was trained for cars while the other network was trained for both cycles and pedestrians. The anchor for pedestrians has a width of 0.6, length of 0.8 and height of 1.73 meters whereas for the cyclist it has a width of 0.6, length of 1.76 and a height of 1.73 meters. Both having a z center of -0.6. These anchors were matched with the 2D IOU, such that it uses positive and negative thresholds of 0.5 and 0.35. With the official KITTI evaluation metrics such as: 3D, 2D, Bird's Eye View (BEV) and Average Orientation Similarity (AOS), it is found that the PointPillar network performed better than all the other methods. The PointPillar predicts the 3D

oriented boxes accurately, however it did not consider BEV and 3D metrics. In fact the 3D box is predicted through the images, where the 2D matching detection is conducted and then the orientation is estimated. The accurate classification for cycling and pedestrians was somehow difficult because the narrow shape vertical features like tree trunks were often classified as pedestrian. With the same inference point it was noted that the learned PointPillar provided better result, but compared to the other learned encoder like VoxelNet it was less stronger. However, the VoxelNet is quite slow such that for even a single point cloud it requires 225ms inference time (4.4Hz), whereas the PointPillar can get better speed at 62 Hz overall. Unlike VoxelNet, it also does not require a hand tuning for point cloud configuring. It can grasp the whole information provided by the cloud points directly.

2.2 Data Collection

The goal of the data collection is to obtain the point clouds data from the underground mine. The detailed steps on data collection are described in Chapter 3. The data collection was undertaken in the Edgar Mine, Colorado. A Husky Robot was used to take the 3D point clouds in several places in the underground mine. The point clouds were taken by LiDAR sensor with three to eight people as the objects. One thousand point cloud files were collected from two different time periods, 2019 and 2021, in the same location.

2.3 Data Preprocessing

The raw data obtained from the underground mine cannot be directly used with the algorithms because the data has a different format than the KITTI datasets. As we can see in Figure 3.10, the objects in KITTI dataset are located in front of the LiDAR sensor. Meanwhile, in Edgar dataset, the objects are surrounding the center. The data needs to be processed to obtain the same format as the KITTI datasets.

The author put the code of data preprocessing in <https://github.com/karana0103/EdgarObjDetection/tree/main/tools/preprocessing> and in Appendix A. The code shifts the point cloud data to KITTI like format so the object location is in front of the center axis and the height of axis origin is 1.73 meters above the ground. The code can be run by running `preprocessing.py` which will multiply the point clouds with a matrix translation and rotation transformation. The output from this step will be a set of point cloud data that can be used on the state-of-the-art object detection algorithm.

2.4 Labeling

In order to create a good model and benchmarking, every LiDAR file should be annotated precisely and accurately. Generally, annotating LiDAR files takes more time than images and some LiDAR annotation

tools charge a fee to use their service. Supervise.ly allows students and community members to use their LiDAR annotation tool free without any subscription fee. It is also quite easy to use and has intuitive operations. However, the annotation generated from supervise.ly should be processed to KITTI label format in order to work in KITTI-based 3D object detection. Supervise.ly generates json format labels and gathers all annotation in one file. Meanwhile, the KITTI label format uses one label file for one file. Understanding the difficulties of transforming the format, the author uses the label generation tools from (Zhengxue et al, 2021) and web-based LiDAR annotation tool supervise.ly. The paper (Zhengxue et al, 2021) provides open-source code to convert the data from supervise.ly to KITTI label format. It also provides a transformation matrix so we can easily calibrate our label to the KITTI format.

The conversion from supervise.ly label format to KITTI can be done using code given in: https://github.com/karana0103/EdgarObjDetection/tree/main/tools/label_extractor and Appendix B. It can be done by running `converterlabel.py` and place the supervise.ly results in the folder. The code contains data extractor from supervise.ly JSON file to KITTI label file. But it also contain matrix transformation to shift the label to the KITTI format label as described in preprocessing section before.

2.5 Transfer Learning

In this step, the pretrained models from the KITTI dataset is used to predict people in the Edgar dataset. If the models are good enough, then the further steps are no longer needed. The goal of this step is to know if the object detection models that were trained with the open-space dataset can also detect people in the underground mine. The author use 300 testing dataset to test the pretrained model. The testing dataset can be obtained from <https://github.com/karana0103/EdgarObjDetection>. This step answer research question number 1: *How can we run a state-of-the-art 3D object detection algorithm with an underground mine dataset?* and research question number 2: *Is transfer learning effective in detecting objects in the underground mine dataset?*

2.6 Epochs Analysis

In this step, each model is trained with the Edgar dataset. However, there is no guidance on the number of epochs needed to create a good model. A sufficient number of epochs will create optimal precision. Meanwhile, more epochs result in more training time. The optimal number of epochs is needed to balance precision performance and training time.

For this step, a thousand files of the Edgar dataset is divided into 700 files training set and 300 files testing set. A high-performance computer is used for training and testing. The specifications of the computer are given in Table 2.2. This step answer research question number 3: *How many epochs are*

needed to generate high accuracy?

Table 2.2 Specification for training and testing models

CPU	2× Intel Xeon E5-2680 v3 @ 2.50Ghz
GPU	4 NVIDIA Tesla K80
RAM	2133 MT/s, DUal Rank, ×4 Data Width RDIMM, (4.84 GB/core)
Interconnect	Omni-Path HFI (1/node)

2.7 Different Number of Dataset Analysis

As deep learning relies on the training dataset to learn, more training data will generate better precision of object detection. However, creating a large dataset is time-consuming and requires many resources. In this step, the author aims to find the optimal number of dataset files to generate a good precision number. This thesis uses a 70/30 training-testing split, with details given in the following table. This step answer research question number 4: *What is the effect of increasing the dataset number?*

Table 2.3 Training and testing split

Training Files	Testing Files
700	300
600	252
400	171
200	85

2.8 Shuffling Analysis

The training process for deep learning algorithms means finding the minimum value of loss. If the data gathered from the fields are in order, the model will have bias toward the training data. As discussed in [13] [14], shuffling the training dataset will prevent overfitting and boost the convergence rate. All of the data gathered from the Edgar Mine is naturally in order. This step will prove whether or not shuffling the training dataset will improve testing performance. This step answer research question number 5: *What is the effect of shuffling the dataset to prevent underfitting or overfitting of the model?*

2.9 Independent Dataset Testing

This step will prove the performance of object detection in a never-before-seen environment. The Edgar dataset was collected in two batches. The first batch was taken in 2019 and the second was taken in 2021.

Even though the location is similar, this step will analyze the improvement needed for the real use cases of the object detection, such as autonomous vehicles. This step answer research question number 7: *What is the performance of the algorithm with independent testing?*

2.10 Assessment of the Result

All of the results from the previous step will be collected and assessed in this step. In this last step, the author concludes the best method of 3D object detection in the underground mine environment. Several suggestions for future research are also discussed in this last step. This step answer research question number 6: *Which algorithm is the best for 3D pedestrian detection in underground mines environment?*

CHAPTER 3

DESCRIPTION OF THE CASE STUDIES

To compare performance of three object detection algorithms, this thesis used point clouds data from Edgar underground mine in Colorado. The object detection results is compared with KITTI datasets for benchmarking purposes. Details on both datasets is discussed in this chapter.

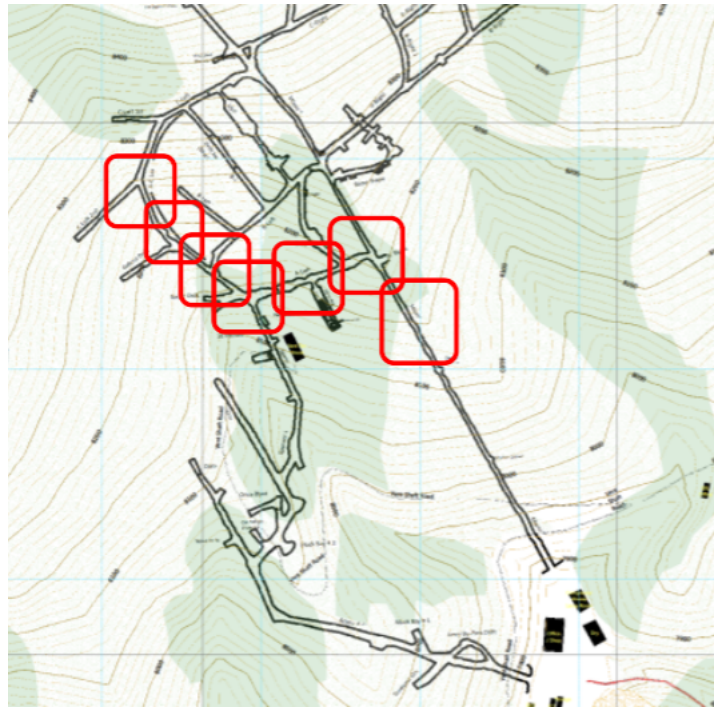


Figure 3.1 Edgar mine map with red circles indicating the place of Edgar datasets were taken

3.1 Edgar Mine

The Edgar Mine is an experimental mine located in Colorado and named after the Edgar mineral vein running above the hillside. It was acquired by lease from the Big Five Mining company, and since then, it has acquired more land and diversified its underground workings. Because the mine has been classified as being the best, with modern infrastructure and safe underground workings, many private and federal agencies have used the facility to undertake various projects such as tunnel detection, rockburst detection, and rock mechanics.

The mine covers almost two miles of underground workings and is divided into the Army and the Miami tunnel. Its underground workings consist of a horizontal maze that aggregates to a length of 9250

feet. The horizontal openings vary depending on size and distance from the surface; the minimum size is six feet wide and six feet tall, whereas the maximum size is 15 feet wide and 15 feet tall. Properly equipped with utility systems such as power, water, communication, ventilation, emergency exit, and ground support that allows safe workings underground, the mine is powered by an electric distribution system that distributes power throughout. The mine acquires its water from the Idaho Springs; the water is stored in the Miami tunnel reservoir and is pumped throughout the mine. A cast ventilation system supplies air underground. The mine is also equipped with three communication systems that use radios and feeders positioned strategically for quick access from any location in the mine. Due to the magnitude of the risk involved in working underground, the mine is fitted with emergency escape systems, including tunnels as primary escape and rooms fully fitted with a life-support facility.

Table 3.1 Specification of Husky unmanned ground vehicle

Dimension	990×670×390 mm
Weight	50 kg
Max speed	1.0 m/s (2.2 mph)
Run Time	3 hours
CPU	Intel i7 with 3.7 GHz
RAM	32 GB
GPU	Nvidia 1660Ti with 6 GB VRAM



Figure 3.2 Husky robot for data collection in Edgar Mine

3.2 Edgar Dataset

The Edgar dataset was taken in the Edgar Mine on two different occasions. Edgar dataset A was taken in 2019 and Edgar dataset B was taken in 2021. Five hundred files were chosen randomly from each occasion, resulting in a total of one thousand files generated. The data was gathered with the Husky unmanned ground vehicle robot (Figure 3.2) with specifications given in Table 3.1. An Ouster LiDAR sensor was installed in the Husky to obtain the point cloud in the underground mine area. The specification of the LiDAR is given in Table 3.2. The data was taken by running Husky Robot from the entrance of miami tunnel to the center of the mine as can be seen in Figure 3.1. Husky robot recorded the mine point clouds data for about 30 minutes for each data collection of Edgar dataset A and B.

The Edgar dataset is splitted into two parts: training and testing. To maintain the training and testing quality with limited number of datasets, 70/30 training/testing split will be used in this experiment. The Edgar dataset gathered from various locations in the underground mine such as straight entrance, intersections, and dead-end tunnel. Other than people, walls and roof, the datasets capture some tools in the underground mine such as toolbox, hose and piping, electrical equipment and other mining-related tools. Several examples of the Edgar datasets are given in Figure 3.3 and Figure 3.4 .

Table 3.2 Comparison of sensor for KITTI dataset (Velodyne) and Edgar dataset (Ouster)

	Velodyne	Ouster
Field of view		
<i>Horizontal</i>	360 deg	360 deg
<i>Vertical</i>	26.8 deg	45 deg
Vertical Resolution	64	64
Precision	2 cm	0.7 - 5 cm
Points per Second	1.3 million	1.3 million
Scanning Frequency	10 Hz	10 Hz or 20 Hz

3.3 KITTI dataset

The KITTI dataset is one of the most famous datasets in the object detection field, especially in autonomous vehicles. The dataset was obtained from five different locations: city, residential, road, campus, and person. It was taken from a car with a LiDAR sensor attached on top of it. They used a Velodyne-64 LiDAR sensor to collect point cloud from the surrounding objects. The specification of the Velodyne LiDAR is given in Table 5. The dataset contains 7,418 training files and 7,518 testing files. The KITTI dataset became famous not only because of the quality of the datasets, but also because it has been

used for benchmarking tools for many object detection algorithms. Karlsruhe Institute of Technology et al gathered the data and classified the objects into three classes: car, pedestrian, and cyclist. They not only provide the LiDAR data but also images that were captured at the same time with the point clouds.

3.4 Comparison Between Edgar and KITTI Dataset

The first distinguishing properties of the two objects is the location. The KITTI dataset is mainly used for development of autonomous vehicles for the road and common street area. The Edgar dataset was obtained in an underground mine, with the main purpose to detect people and mining equipment.

A good dataset should have a balanced proportion of training and testing set. In KITTI dataset, the total area between the training and testing set is quite similar, as visualized in Figure 3.6 a and Figure 3.6 b. The Edgar dataset was also handpicked to have a balanced proportion in training and testing set, as visualized in Figure 3.7 and Figure 3.9.

From Figure 3.6 and Figure 3.7 we can see that the total area in the KITTI dataset is relatively larger than the Edgar dataset. However, from Figure 3.8 and Figure 3.9 , we can see that the point density in the Edgar dataset is higher than the KITTI dataset. This is because in the Edgar dataset, the walls and roof are so eminent. It challenges the object detection methods to separate foreground objects from background objects.

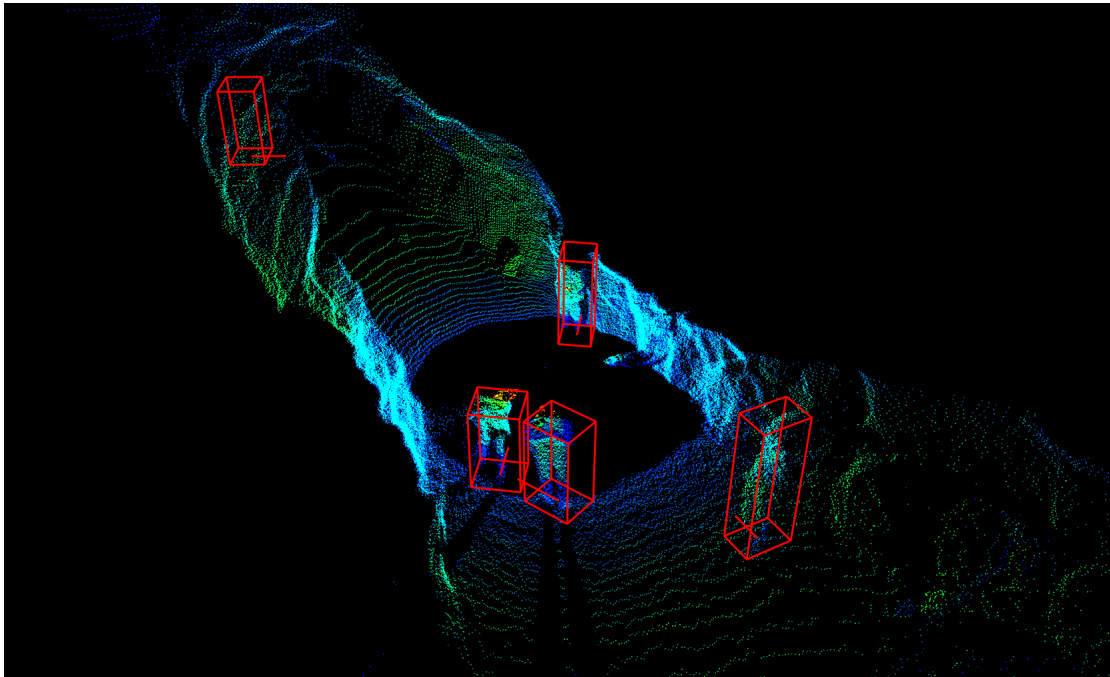


Figure 3.3 Point clouds of Edgar dataset

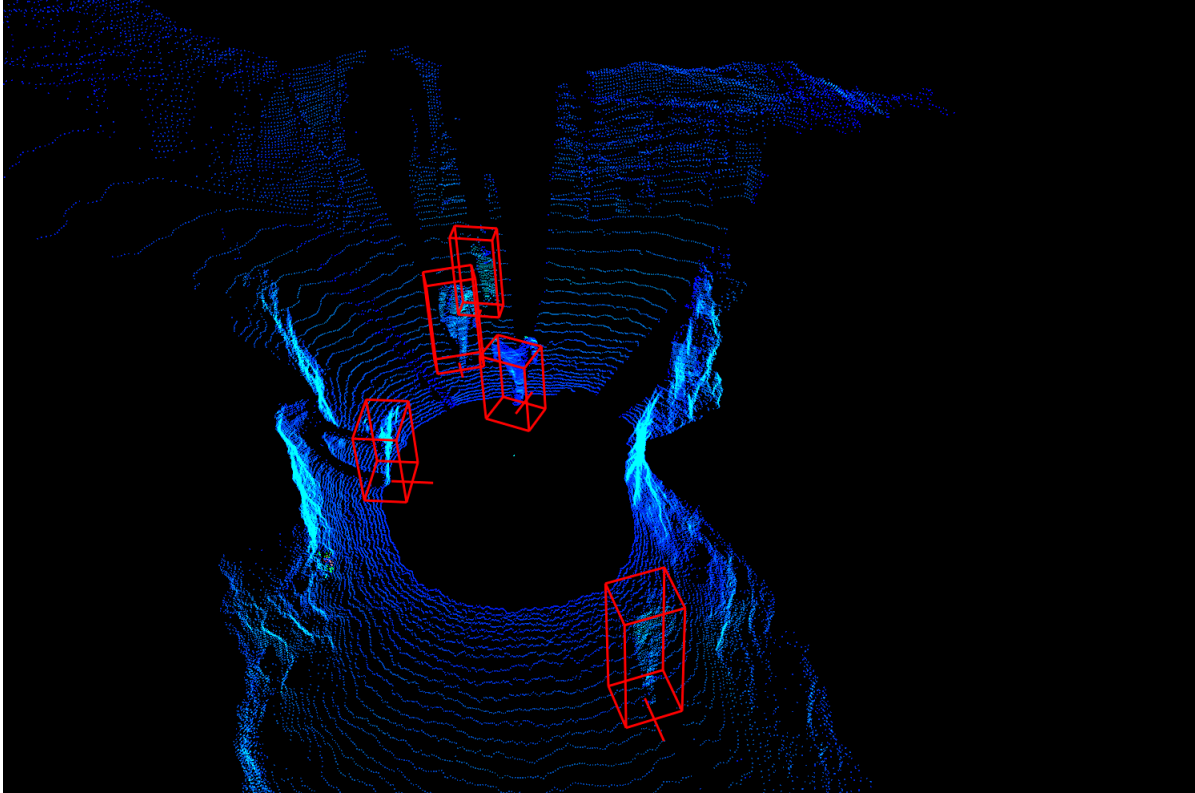


Figure 3.4 Point clouds of Edgar dataset

Although the KITTI dataset has a large number of training files, only 1,780 files have pedestrians, with total of 4,625 pedestrian objects. Meanwhile, Edgar dataset contains 5,145 people from 1,000 files. On average, there is only one person in a file of the KITTI dataset. In contrast, Edgar dataset contains five to six people in each file.

3.5 Preprocessing

As discussed in the previous section, the KITTI dataset was obtained from a car with a Velodyne sensor attached 1.73 meters above the ground. Meanwhile, the Edgar dataset was obtained from a Husky robot with LiDAR sensor attached 70 centimeters above the ground. Figure 3.11 also shows that the object of the KITTI dataset is in the front of the vehicle. It is different from the Edgar dataset, which has several objects around the vehicle. These position differences mean that the Edgar dataset cannot be directly used in the KITTI-based object detection algorithm. Preprocessing was conducted to shift the center axis in the Edgar dataset to around 1.73 meters above the ground and pointed at the object. This preprocessing was done by multiplication of the point clouds to the calibrated transformation matrix.



Figure 3.5 Various locations of KITTI dataset (source: Geiger et al., 2013)

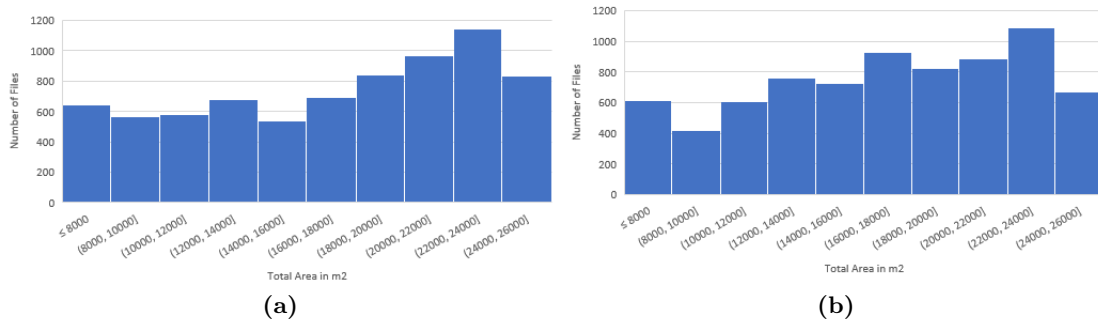
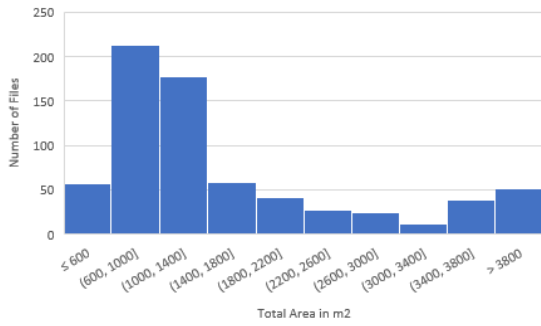
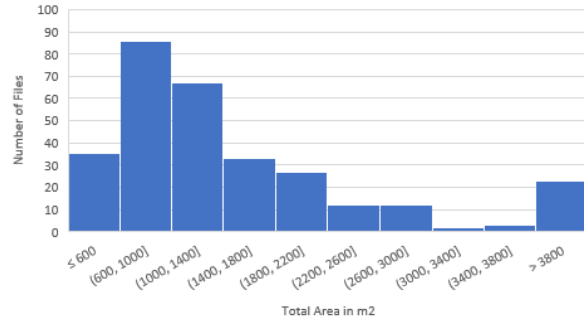


Figure 3.6 (a) Total area histogram in KITTI training set (b) Total area histogram in KITTI testing set

However, the position shifting is not enough to make the Edgar dataset’s format similar to that of the KITTI dataset. In the Edgar dataset and KITTI dataset, the point cloud stored four categories of information: position coordinates (from x, y, and z axes) and intensity. The range of intensity in the KITTI dataset is from 0 to 1, while in the Edgar dataset, the intensity ranges from 0 to 256. Thus, the Edgar dataset’s intensity should be normalized to 0–1 so the format is similar to that of the KITTI dataset. In conclusion, the Edgar dataset can be used in the KITTI dataset-based algorithm after the preprocessing stage.

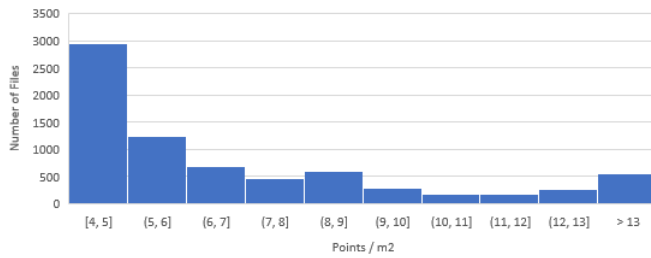


(a)

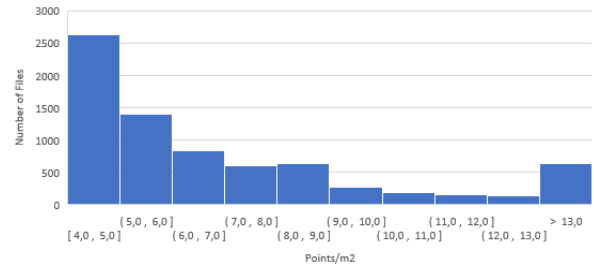


(b)

Figure 3.7 (a) Total area histogram in Edgar training set (b) Total area histogram in Edgar testing set

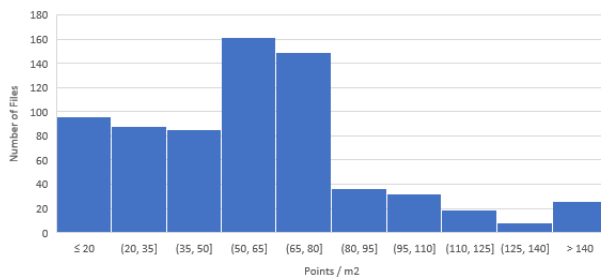


(a)

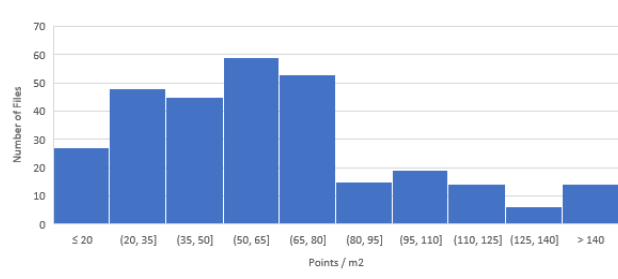


(b)

Figure 3.8 (a) Point density histogram in KITTI training set (b) Point density histogram in KITTI testing set



(a)



(b)

Figure 3.9 (a) Point density histogram in Edgar training set (b) Point density histogram in Edgar testing set

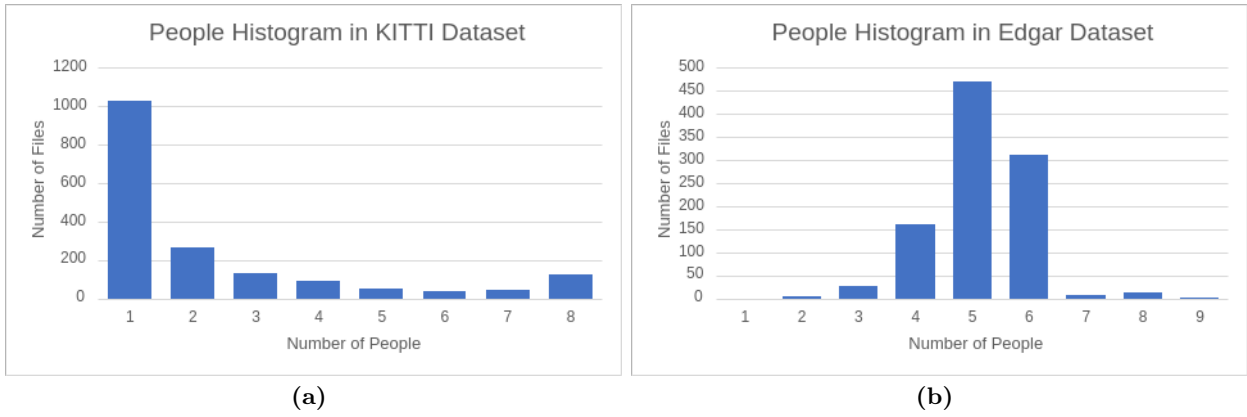


Figure 3.10 (a) People histogram in KITTI dataset (b) People histogram in Edgcar dataset

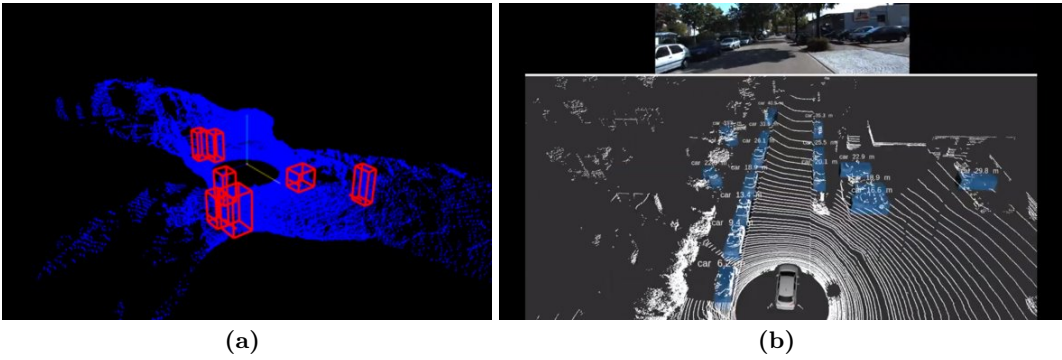


Figure 3.11 Comparison of the point clouds in (a) Edgcar dataset and (b) KITTI dataset (source: Geiger et al, 2013)

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Results on Transfer Learning

In this section, a transfer learning approach is adopted to test object detection performance in the Edgar Mine. The models have been trained using the KITTI dataset to detect people in the road and common street environments. It has an average precision of around 40% with the KITTI test set, as published on the KITTI Benchmark website. However, the object detection does not perform well in the Edgar test set. The comparison of testing results on the Edgar dataset and the KITTI dataset are given in Table 4.1.

Table 4.1 Comparison of testing result on the Edgar dataset (left) and KITTI dataset (right)

	Average Precision with Edgar test set (%)	Average Precision with KITTI test set (%)
Part- A^2	6.21	43.35
PV-RCNN	10.7	43.3
PointPillars	4.1	41.9

The Edgar test results were obtained from testing a 300-file dataset from Edgar Mine, while in the KITTI test set, 7,518 files were used to generate that result. PV-RCNN has the highest average precision, with 10.7% average precision in the Edgar test set, followed with Part- A^2 and PointPillars at 6.21% and 4.1% AP, respectively.

From the results, it can be concluded that transfer learning is not an effective method to detect people in the underground mining environment. Although the size of people is relatively similar, the algorithm has difficulties separating foreground and background objects. The wall and roof in the underground mine might act as noise to the object detection and prevent the algorithm from separating people from the surrounding objects. Occlusion in the underground mine is also visually high compared with the KITTI dataset, with the result that the pretrained model cannot detect people as well in non-occluded situations.

4.2 Results on Epochs Analysis

After determining that transfer learning is not effective due to different occlusion levels and noise levels, training on the Edgar dataset was performed. Seven hundred files were used for the training process, and the algorithm was tested with 300 different files. Batch size was chosen at four, following the

recommendation of OpenPCD publisher. At first, the algorithm was trained at 80 epochs, similar to the number that PV-RCNN used in their paper. In comparison, Part- A^2 used only 50 epochs with a batch size of six for the training process, as explained in the paper. The results are given in the following figures:

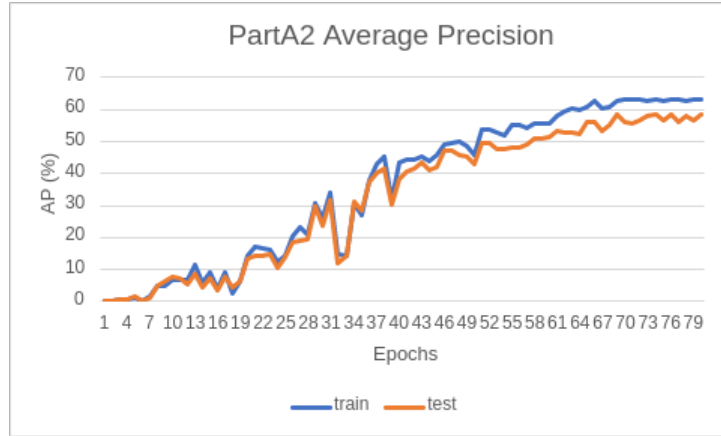


Figure 4.1 Result of training and testing of Part- A^2 algorithm with Edgar dataset.

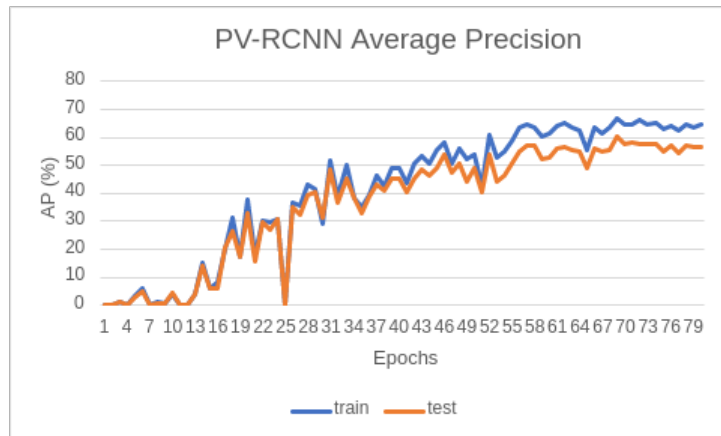


Figure 4.2 Result of training and testing of PV-RCNN algorithm with Edgar dataset.

From the figures, we can see that the peak of Part- A^2 , PV-RCNN, and PointPillars occurs at 69, 69, and 76 epochs, respectively. In Part- A^2 and PV-RCNN, there is no significant improvement on the AP after 67 epochs. The graphs are also relatively flat and fluctuate no more than 1%. In PointPillars, the graph is relatively flat after 70 epochs. There is no significant improvement in the following 10 epochs. From this graph, the author concludes that for time efficiency purposes, the user can choose 70 epochs for training the datasets. However, if time is not the main consideration, users can use 80 epochs to ensure the best performance of the algorithm.

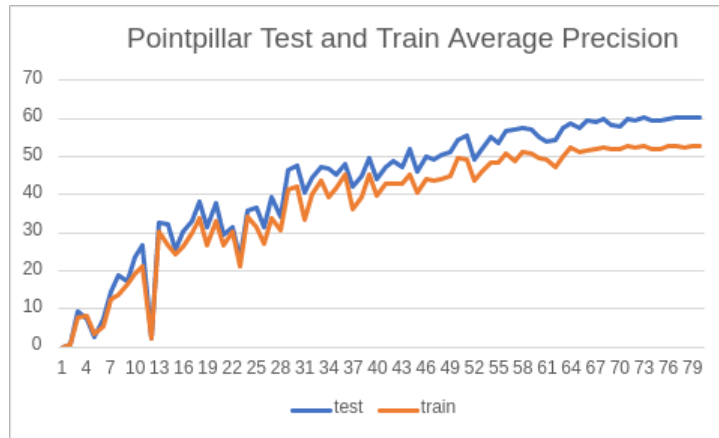


Figure 4.3 Result of training and testing of PointPillars algorithm with Edgar dataset.

4.3 Results on Different Numbers of Training Datasets

Next, the study aims to choose the optimal number of datasets needed to obtain a good enough precision. The training was conducted in four different scenarios, as described in Chapter 3. Performance measured at 70/30 training–testing split, and with 200, 400, 600, and 700 training files. The results are given in the following figures:

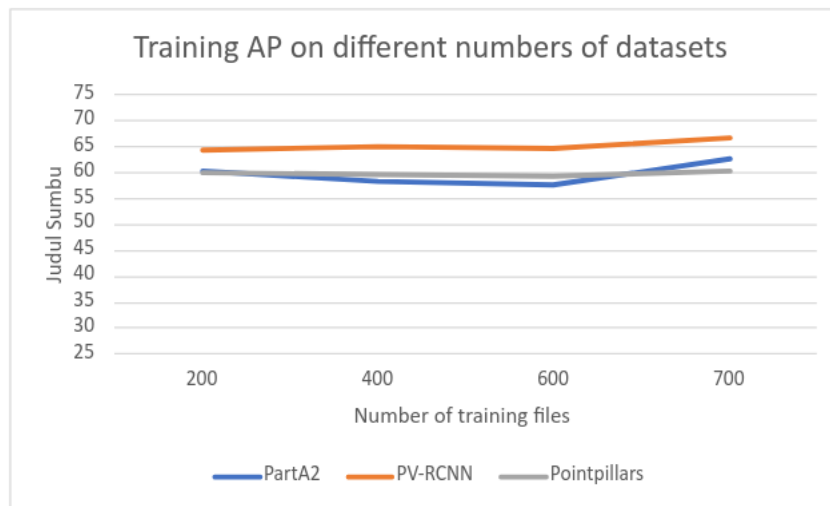


Figure 4.4 Training AP on different numbers of datasets.

It can be seen from Figure 4.4 that average precision is not so much changed with 200, 400, 600, or 700 training files. However, the testing AP changes dramatically as the number of training files increases. This means that with only 200 and 400 training files, the model is underfitting. It can perform well in the training set but cannot perform well in the new files. The changes of testing AP are not so significant

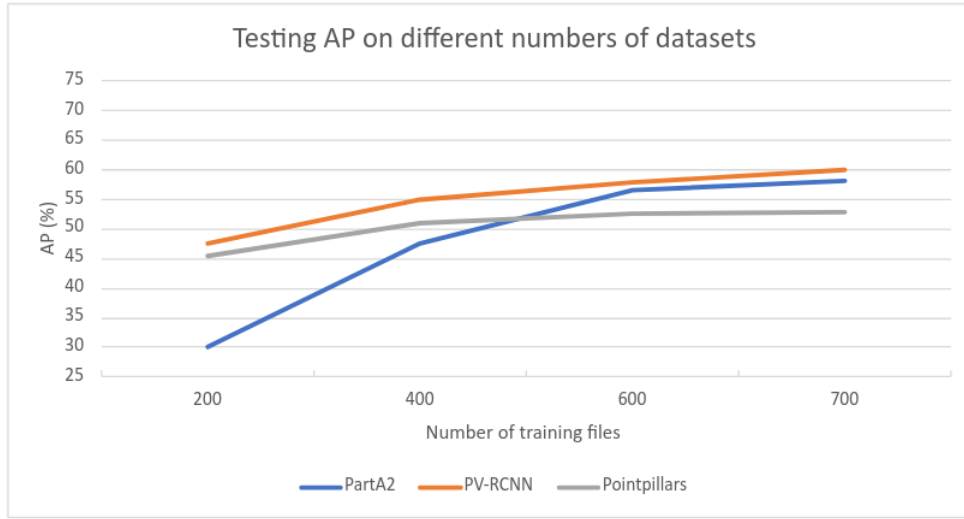


Figure 4.5 Testing AP on different numbers of datasets.

between 600 and 700 training files. Thus, the models no longer have the underfitting condition anymore. Rather, the models can detect the object in training as well as in testing files. Seven hundred files is optimal condition to be used for training, as increasing the number will not significantly improve the AP anymore.

4.4 Results on Shuffling Analysis

A comparison between the result on shuffled and non-shuffled training sets was performed. The training was performed with 700 files for training and 300 files for testing. It uses 80 epochs, as it is the optimal number according to previous analysis. Then the best number of AP is chosen for each model. The result is given in the following table:

Table 4.2 Average precision of shuffled and non-shuffled dataset.

		Average Precision (%)		
		Shuffled dataset	No-Shuffled dataset	Differences
Part-A²	Training	62.83	60.97	1.86
	Testing	58.27	57.50	0.77
PV-RCNN	Training	66.86	65.48	1.38
	Testing	60.13	58.95	1.18
PointPillars	Training	60.31	58.31	2.00
	Testing	52.76	52.25	0.51

From the table, we can conclude that the shuffled dataset performed better than the non-shuffled dataset in every model. The number of differences is varied, but on average, it can increase 0.5–1.2% in the

testing AP. This is because the shuffled data learn better and can recognize a different occasion with minimal bias.

Training with shuffled datasets can increase model performance and make training time faster because deep learning architecture relies on the quality of the datasets. By default, non-shuffled dataset files are chronologically in order from the entrance to the inside part of the mine. The differentiation between files is not much changed. People’s movement, for example, is only moving less than 1 meter per file. Walls and roofs condition are also not much changed. These conditions created a bias for the models in the non-shuffled dataset. The models generalized the features and resulted in overfitting condition. It resulted on high training precision but low testing precision. Moreover, the testing dataset is not chronologically in order because it was handpicked to create balance properties, as given in Figure 3.7 and Figure 3.9.

4.5 Results on Independent Test

An independent test was performed by recognizing the fact that the Edgar dataset was taken from two different time periods. The dataset contains 500 files taken in 2019 (Edgar dataset A) and 500 files taken in 2021 (Edgar dataset B), the training was performed with 500 files from Edgar dataset A, with 214 test files from Edgar dataset B to maintain a 70/30 split. To compare the performance, the normal mixed Edgar dataset A and B is also trained and tested. The results of independent test and normal mixed Edgar dataset A and B are given for comparison in the following table:

Table 4.3 Average precision with independent, mixed, and KITTI test set.

	Average Precision with Independent test set (%)	Average Precision with mixed Edgar dataset A and B test set (%)	Average Precision with KITTI test set (%)
Part-A^2	44.67	58.27	43.35
PV-RCNN	40.57	60.13	43.3
PointPillars	37.10	52.76	41.9

Overall, the result of the independent test set is not as good as the mixed test set while the dataset mixed from dataset A and B. This can be explained if we see the histogram of people’s height in Figure 4.6 and Figure 4.7. From the figures , the different patterns can be seen. The Edgar dataset A contains more people with heights less than 1.1 meters. People’s label with a height less than 1.1 meters means that there are people sitting in the mine. While there are hundreds of people sitting in the Edgar dataset A, the Edgar dataset B only has a few people sitting. Those differences make the model misclassify some rocks as people sitting. However, the Part- A^2 model can perform better than PV-RCNN in terms of detecting

people sitting. This is because Part- A^2 does not use the voxelization process that sometimes misclassifies rock as people. Part- A^2 algorithm uses a direct point cloud as input without any fine grinding method that creates information loss. This makes Part- A^2 perform better than the other methods and even better than the KITTI dataset performance in the testing dataset. From this experiment, we can conclude that the Part- A^2 can perform better in the new environment where there is limited similarity to the training set.

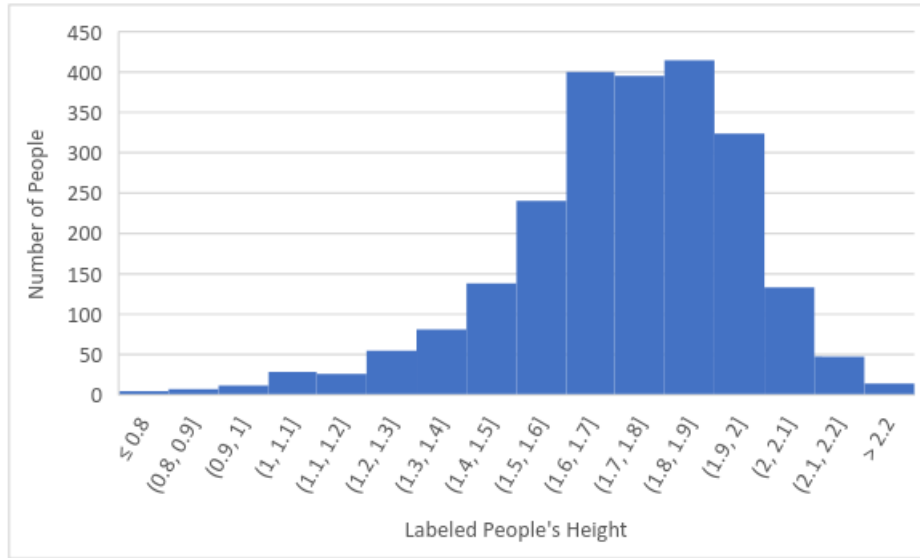


Figure 4.6 Histogram of people's height in Edgar dataset obtained in 2021.

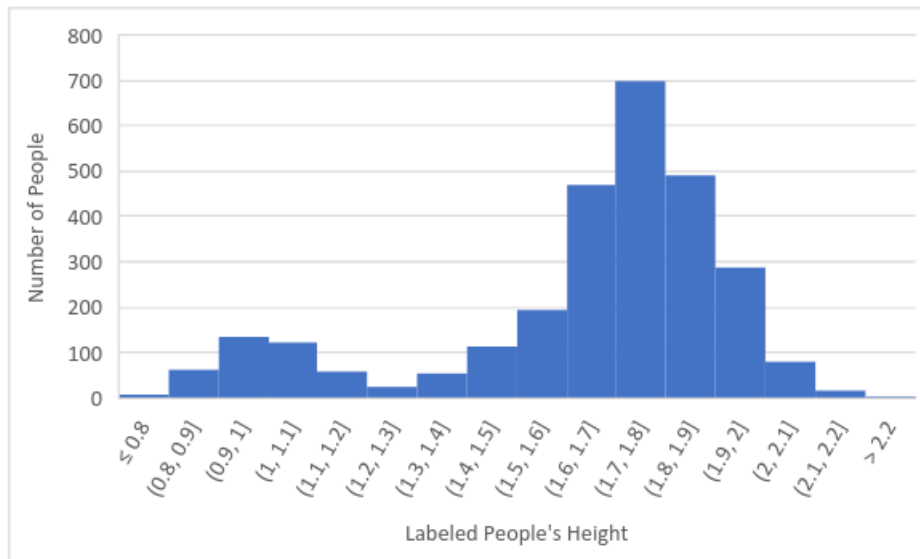


Figure 4.7 Histogram of people's height in Edgar dataset obtained in 2019.

4.6 Best Case Results

Using a combination of information gathered in the previous experiment, the final result of comparison of 3D object detection methods in an underground mine environment is given in the following table:

Table 4.4 Overall performance of three methods of object detection.

	Average Precision (%)	Training Time	Inference Time (in ms)	Inference Time (in Hz)
Part-A^2	58.27	2 hours 5 minutes	127 ms	7.8 Hz
PV-RCNN	60.13	3 hours 44 minutes	103 ms	9.7 Hz
PointPillars	52.76	1 hours 2 minutes	52.5 ms	19 Hz

From the average precision perspective of view, PV-RCNN is the best method for 3D object detection in the underground mine. Although Part- A^2 has better AP for the KITTI test benchmark, in the underground mine dataset, it only achieves 58.27%, 2% less than PV-RCNN. This is because of the significant differentiation between the road and underground mine environment. In the underground mine environment, a structured-based algorithm does not suit very well, as seen by the result of PointPillars. However, a raw point cloud-based algorithm such as Part- A^2 is not as effective as PV-RCNN, which combine the advantages of structured-based and raw point cloud-based algorithm. However, the ramification of using that combination is a high training time. In terms of inference time, the PV-RCNN is still more effective than Part- A^2 , even though it is not as good as PointPillars.

4.7 Interpretation on Autonomous Vehicle Application

From the specification of LiDAR in Table 3.2 we know that a typical velodyne LiDAR usually run at 10 Hz. For real time application in Autonomous Vehicle Application, only PointPillars can detect and run the algorithm without any delay on the detection. It is because PointPillars approach on stacking the point clouds into pillars has proven to improve the speed of detection without compromising the quality. However, as discussed by Le et al. (2021) Pointpillars approach has difficulty on detection pedestrian because the pedestrian has minimal pillar features to be used and extracted by the algorithm.

In case, if the LiDAR sensor speed can be downgraded to 9 hz, PV-RCNN can work well without any delay on the detection and having 10% higher precision on the object detection. PV-RCNN has longer time than Pointpillars because PV-RCNN combine two approaches, which are point net based and voxelization base, in its architecture. On one hand, it combine the advantages of object detection method. On the other hands, it combine also the disadvantages of the two approaches.

Part- A^2 has a special use case when the AV is deployed in a relatively new environment. It can achieve high precision, as explained in the independent dataset case study. The inference speed of Part- A^2 is the longest than the other methods. It is because Part- A^2 use raw point clouds data as the input for the algorithm without any fine-grained method or voxelization.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

State-of-the-art 3D object detection methods were built mainly for autonomous vehicle applications in common road and normal driving environments. Application of the methods in other use cases requires some preprocessing and modification. As the deep learning method relies on the quality of the training dataset, a transfer learning approach may not perform well in a different environment. The best way to use 3D object detection is by training the model for the real case of operation. This thesis conducts an analysis on people detection in the underground mine. It is interesting to be discussed because the underground mine environment has significant differences from the normal driving environment.

The transfer learning approach is conducted in this thesis with KITTI pre-trained models. The result is bad because of the significant differences between point clouds in the normal road and the underground mine. In the underground mine, the presence of roofs and walls is eminent. It makes KITTI pre-trained models unable to differentiate the object from the background and noise such as roofs. Transfer learning without any retraining process results in the bad performance of object detection in the underground mine.

To achieve good detection, the model needs to be trained with certain number of epochs. Generally, a high number of epochs will generate high-precision results. However, time for training becomes a consideration especially when dealing with thousands of files with limited computing capability. To solve that problem, evaluating the training and testing precision while running the training is recommended. If there are no significant changes in testing and training precision, the training process can be stopped at certain epochs.

3D object detection performance is also affected by the number of training data. A sufficient number of training data should be prepared to be trained by the model. To improve the quality of training and minimize the bias of the training set, shuffling the dataset is recommended. An independent test study is also important to know the performance of the object detection in a new and different environment.

Finally, in the case of object detection in underground mines, the study case in Edgar Mine reveals that PV-RCNN performs better than Part- A^2 and PointPillar. PV-RCNN utilizes the power of computation with voxelization but also recognizes localization accuracy by using raw point cloud as the input. Although PV-RCNN performs slightly worse than Part- A^2 in the KITTI benchmark test, a large number of point density and high occlusion characteristics in the underground mine allows PV-RCNN to predict the object better than Part- A^2 , which only utilizes part and aggregation of the objects.

5.2 Recommendations for Future Research

The object detection methods use a supervised deep learning approach to learn features of people in the underground mine. Thus, the quality of the training and testing datasets is really important to ensure that the machine learns the correct feature. For a real case study, people in the underground mine will not only stand and sit. Mineworkers might be operating the machine, holding tools, or laying. This thesis only provides a dataset with people sitting, walking, and standing positions. It is recommended to take a different object pose as varied as we can in the future data collection stage. The independent case study showed that the model will be biased if the train set always has people sitting when the test set does not have people sitting. It is better to annotate a different pose of people than the other class to visualize the evaluation well.

This thesis only analyzes people as the object. Using other objects such as machines, tools, or other relevant things in the underground mine is recommended. It is important because multi-class object detection behavior may be different from single-class object detection. In a real case situation, the underground mine contains many objects that should be recognized by the autonomous vehicle to operate safely and efficiently. This approach will evaluate the behavior of object detection for autonomous vehicles in underground mines more realistically.

Other evaluation metrics are also recommended for autonomous vehicle application in the underground mine. For example, instead of calculating average precision, the model is evaluated with the accuracy of detecting people position in the underground mine. A real case study using autonomous vehicle to detect people in the real time is also recommended.

This thesis provides one thousand annotated LiDAR files with KITTI format to be used in future research that can be accessed in <https://github.com/karana0103/EdgarObjDetection> . However, the files were gathered from the same location. As the tools for annotation and converting the files are publicly open, creating a new dataset with a different location will improve the quality of research for autonomous vehicles in underground mines. It would also be interesting to compare the performance of the object detection in open-pit mines.

For autonomous vehicles application, 60,13% average precision for 3D people detection might not be sufficient. As safety is essential in the underground mining industry, the object detection should be able to detect more precisely people surrounding the AVs. For comparison, Li et al. (2022) used thermal images camera and 2D YOLOv4 network and got over 90% average precision with 48.2 fps speed. However, 2D object detection cannot precisely detect the exact location and sizes of the objects. On the other hand, current 3D object detection has lower accuracy than 2D object detection. Thus, discussing multi-modal

object detection using LiDAR and thermal images is interesting in future research.

REFERENCES

- Balasubramaniam, Abhishek, & Pasricha, Sudeep. 2022. Object Detection in Autonomous Vehicles: Status and Open Challenges. *arXiv preprint arXiv:2201.07706*.
- Bengio, Yoshua. 2012. Practical recommendations for gradient-based training of deep architectures. *Pages 437–478 of: Neural networks: Tricks of the trade*. Springer.
- Bottou, Léon. 2009. Curiously fast convergence of some stochastic gradient descent algorithms. *Pages 2624–2633 of: Proceedings of the symposium on learning and data science, Paris*, vol. 8.
- Chehata, Nesrine, Guo, Li, & Mallet, Clément. 2009. Airborne lidar feature selection for urban classification using random forests. *In: Laserscanning*.
- Chen, Qi, Sun, Lin, Wang, Zhixin, Jia, Kui, & Yuille, Alan. 2020. Object as hotspots: An anchor-free 3d object detection approach via firing of hotspots. *Pages 68–84 of: European conference on computer vision*. Springer.
- committee, On-Road Automated Driving (ORAD). 2016 (sep). *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*.
- Dang, Tung, Mascarich, Frank, Khattak, Shehryar, Nguyen, Huan, Nguyen, Hai, Hirsh, Satchel, Reinhart, Russell, Papachristos, Christos, & Alexis, Kostas. 2020. Autonomous search for underground mine rescue using aerial robots. *Pages 1–8 of: 2020 IEEE Aerospace Conference*. IEEE.
- Everingham, Mark, & Winn, John. 2011. The pascal visual object classes challenge 2012 (voc2012) development kit. *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, **8**, 5.
- Geiger, Andreas, Lenz, Philip, & Urtasun, Raquel. 2012. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. *In: Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Geiger, Andreas, Lenz, Philip, Stiller, Christoph, & Urtasun, Raquel. 2013. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, **32**(11), 1231–1237.
- Guo, Yulan, Wang, Hanyun, Hu, Qingyong, Liu, Hao, Liu, Li, & Bennamoun, Mohammed. 2020. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, **43**(12), 4338–4364.
- Ku, Jason, Mozifian, Melissa, Lee, Jungwook, Harakeh, Ali, & Waslander, Steven L. 2018a. Joint 3d proposal generation and object detection from view aggregation. *Pages 1–8 of: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Ku, Jason, Mozifian, Melissa, Lee, Jungwook, Harakeh, Ali, & Waslander, Steven L. 2018b. Joint 3D Proposal Generation and Object Detection from View Aggregation. *Pages 1–8 of: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Ku, Jason, Pon, Alex D., Walsh, Sean, & Waslander, Steven L. 2019. Improving 3D Object Detection for Pedestrians with Virtual Multi-View Synthesis Orientation Estimation. *Pages 3459–3466 of: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

- Lang, Alex H, Vora, Sourabh, Caesar, Holger, Zhou, Lubing, Yang, Jiong, & Beijbom, Oscar. 2019. Pointpillars: Fast encoders for object detection from point clouds. *Pages 12697–12705 of: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*
- Le, Duy-Tho, Shi, Hengcan, Rezatofghi, Hamid, & Cai, Jianfei. 2021. PiFeNet: Pillar-Feature Network for Real-Time 3D Pedestrian Detection from Point Cloud. *CoRR*, **abs/2112.15458**.
- Li, Jiale, Dai, Hang, Shao, Ling, & Ding, Yong. 2021. From Voxel to Point: IoU-guided 3D Object Detection for Point Cloud with Voxel-to-Point Decoder. *CoRR*, **abs/2108.03648**.
- Li, Xiaoyu, Wang, Shuai, Liu, Bin, Chen, Wei, Fan, Weiqiang, & Tian, Zijian. 2022. Improved YOLOv4 network using infrared images for personnel detection in coal mines. *Journal of Electronic Imaging*, **31**(1), 1 – 25.
- Liu, Shan, Zhang, Min, Kadam, Pranav, & Kuo, Chung-Chieh Jay. 2021. *3D Point Cloud Analysis: Traditional, Deep Learning, and Explainable Machine Learning Methods*. Springer.
- Liu, Zhe, Zhao, Xin, Huang, Tengeng, Hu, Ruolan, Zhou, Yu, & Bai, Xiang. 2019. TANet: Robust 3D Object Detection from Point Clouds with Triple Attention. *CoRR*, **abs/1912.05163**.
- Lu, Haihua, Chen, Xuesong, Zhang, Guiying, Zhou, Qiu hao, Ma, Yanbo, & Zhao, Yong. 2019. SCANet: Spatial-channel attention network for 3D object detection. *Pages 1992–1996 of: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE.
- Mallet, Clément, Bretar, Frédéric, Roux, Michel, Soergel, Uwe, & Heipke, Christian. 2011. Relevance assessment of full-waveform lidar data for urban area classification. *ISPRS journal of photogrammetry and remote sensing*, **66**(6), S71–S84.
- Mansouri, Sina Sharif, Kanellakis, Christoforos, Kominiak, Dariusz, & Nikolakopoulos, George. 2020. Deploying MAVs for autonomous navigation in dark underground mine environments. *Robotics and Autonomous Systems*, **126**, 103472.
- Maturana, Daniel, & Scherer, Sebastian. 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. *Pages 922–928 of: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Qi, Charles R, Su, Hao, Mo, Kaichun, & Guibas, Leonidas J. 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Pages 652–660 of: Proceedings of the IEEE conference on computer vision and pattern recognition.*
- Qi, Charles R, Liu, Wei, Wu, Chenxia, Su, Hao, & Guibas, Leonidas J. 2018. Frustum pointnets for 3d object detection from rgb-d data. *Pages 918–927 of: Proceedings of the IEEE conference on computer vision and pattern recognition.*
- Qi, Charles Ruizhongtai, Yi, Li, Su, Hao, & Guibas, Leonidas J. 2017b. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, **30**.
- Shi, Shaoshuai, Wang, Xiaogang, & Li, Hongsheng. 2019. Pointcnn: 3d object proposal generation and detection from point cloud. *Pages 770–779 of: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.*
- Shi, Shaoshuai, Wang, Zhe, Shi, Jianping, Wang, Xiaogang, & Li, Hongsheng. 2020a. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *IEEE transactions on pattern analysis and machine intelligence*, **43**(8), 2647–2664.

- Shi, Shaoshuai, Guo, Chaoxu, Jiang, Li, Wang, Zhe, Shi, Jianping, Wang, Xiaogang, & Li, Hongsheng. 2020b. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. *Pages 10529–10538 of: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*
- Shi, Weijing, & Rajkumar, Raj. 2020. Point-gnn: Graph neural network for 3d object detection in a point cloud. *Pages 1711–1719 of: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.*
- Simonelli, Andrea, Bulo, Samuel Rota, Porzi, Lorenzo, López-Antequera, Manuel, & Kotschieder, Peter. 2019. Disentangling monocular 3d object detection. *Pages 1991–1999 of: Proceedings of the IEEE/CVF International Conference on Computer Vision.*
- Smith, Leslie N. 2018. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820.*
- Tan, Chuanqi, Sun, Fuchun, Kong, Tao, Zhang, Wenchang, Yang, Chao, & Liu, Chunfang. 2018. A survey on deep transfer learning. *Pages 270–279 of: International conference on artificial neural networks.* Springer.
- Team, OpenPCDet Development. 2020. *OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds.* <https://github.com/open-mmlab/OpenPCDet>.
- Wang, Chia-Hung, Chen, Hsueh-Wei, & Fu, Li-Chen. 2021. VPFNet: Voxel-Pixel Fusion Network for Multi-class 3D Object Detection. *CoRR*, **abs/2111.00966**.
- Wang, Lin, Li, Weishan, Zhang, Yuliang, & Wei, Chen. 2017. Pedestrian detection based on YOLOv2 with skip structure in underground coal mine. *Pages 1216–1220 of: 2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC).*
- Wang, Zhixin, & Jia, Kui. 2019. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. *Pages 1742–1749 of: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE.
- Yan, Yan, Mao, Yuxing, & Li, Bo. 2018. Second: Sparsely embedded convolutional detection. *Sensors*, **18**(10), 3337.
- Yang, Zetong, Sun, Yanan, Liu, Shu, & Jia, Jiaya. 2020. 3DSSD: Point-based 3D Single Stage Object Detector. *CoRR*, **abs/2002.10187**.
- Zhou, Yin, & Tuzel, Oncel. 2018. Voxelnet: End-to-end learning for point cloud based 3d object detection. *Pages 4490–4499 of: Proceedings of the IEEE conference on computer vision and pattern recognition.*
- Zhou, Zhengxue, Li, Leihui, Wang, Riwei, & Zhang, Xuping. 2021. Deep Learning on 3D Object Detection for Automatic Plug-in Charging Using a Mobile Manipulator. *Pages 4148–4154 of: 2021 IEEE International Conference on Robotics and Automation (ICRA).* IEEE.

APPENDIX A
PREPROCESSING.PY

Listing A.1: preprocessing.py

```
import os
import numpy as np
import sys

class kitti_object(object):
    '''Load and parse object data into a usable format.'''
    def __init__(self, root_dir, split=sys.argv[2]):
        '''root_dir contains training and testing folders'''
        self.root_dir = root_dir
        self.split = split
        self.split_dir = os.path.join(root_dir, split)
        self.lidar_dir = os.path.join('velodyne', self.split_dir)
    def get_lidar(self, idx):
        lidar_filename = os.path.join(self.lidar_dir, '%06d.bin'%(idx))
        return load_velo_scan(lidar_filename)

def load_velo_scan(velo_filename):
    scan = np.fromfile(velo_filename, dtype=np.float32)
    scan = scan.reshape((-1, 4))
    print(scan)
    return scan

#algorithm to edit from new dataset to shift x +20, z plus 1 and normalize the
#intensity
dataset = kitti_object('sys.argv[1]')
for x in range(sys.argv[3]):
    data_idx = x
    lidar_data = dataset.get_lidar(data_idx)
    lidar_data_edited=lidar_data
    lidar_data_edited[:,3]=lidar_data_edited[:,3]/np.max(lidar_data[:,3]) #normalize
    #intensity
    lidar_data_edited[:,0]=lidar_data_edited[:,0]+20 #shift the x of center axis to
    #make object in front
    lidar_data_edited[:,2]=lidar_data_edited[:,2]-1 #shift the height of center axis
    lidar_data_edited.tofile(str(x)+'.bin')
```

APPENDIX B
CONVERTER_LABEL.PY (ZHENGXUE ET AL., 2020)

Listing B.2: converter_label.py

```
import json
import copy
import argparse
import os
import collections
import numpy as np

arg = argparse.ArgumentParser("mylabel2KITTIlabel")

target_label = collections.OrderedDict()
target_label['type'] = 'Dontcare'
target_label['truncated'] = 0.0
target_label['occluded'] = 0 # used in image
target_label['alpha'] = 0
target_label['bbox'] = [0, 0, 50, 50] # put 0,0,50,50 as dummy pixel coordinate
target_label['dimensions'] = []
target_label['location'] = []
target_label['rotation_y'] = 0

transform_location = np.mat([
    [0, -1, 0],
    [0, 0, -1],
    [1, 0, 0],
])

def get_transformed_values(_list, m):
    r = m * np.mat(_list).T
    r = r.T
    return r.tolist()[0]

def get_dimensions(_list):
    x = _list[0]
    y = _list[1]
    z = _list[2]
    return list([z, x, y])

start_number = 0

if __name__ == '__main__':
    arg.add_argument("--input_folder", "-i", default="", type=str,
                    help="folder containing JSON file form https://3d.supervise.ly/
                    projects", required=True)
    arg.add_argument("--output_folder", "-o", default="", type=str, help="output
                    folder", required=True)
    args = arg.parse_args()
```

```

input_filename = args.input_folder
output_folder = args.output_folder
print('input_filename:\t', input_filename)
print('output_folder:\t', output_folder)

if not os.path.exists(output_folder):
    os.mkdir(output_folder)

label_list = [json_file for json_file in os.listdir(input_filename) if ".json"
               in json_file]
label_list.sort()
print(label_list)

for idx, data in enumerate(label_list):
    with open(input_filename + data) as f:
        annotation = json.load(f)
        KITTI_annotations = []
        # my_annotations = annotations['figures']
        data_name = data
        print('reading_{}'.format(data_name))

        # key -> name
        objects = {}
        for object_name in annotation['objects']:
            objects[object_name['key']] = object_name['classTitle']

        for label in annotation['figures']:
            # produce new annotations
            tmp_target_label = copy.deepcopy(target_label)
            tmp_target_label['type'] = objects[label['objectKey']]
            # print(list(annotation['geometry'] ['dimensions'].values()))
            tmp_target_label['dimensions'] = get_dimensions(
                list(label['geometry'] ['dimensions'].values()))
            # print('dimensions', tmp_target_label['dimensions'])
            position = list(label['geometry'] ['position'].values())
            position[2] = position[2]-1.7 # z (calibration from the center height)
            position[0] = position[0]-0.2+20 # x (-0.2 is interal calib and +10 is
                for shifted)
            tmp_target_label['location'] = get_transformed_values(position,
                transform_location)
            # print('location', tmp_target_label['location'])
            tmp_target_label['rotation_y'] = -float(label['geometry'] ['rotation'] ["z
                "]) -3.14
            KITTI_annotations.append(tmp_target_label)
            # exit()
            # save current annotation into
            save_filename = "{:06n}.txt".format(idx + start_number)
            save_path = output_folder + '/' + save_filename
            print('saving_{}'.format(save_path))

            with open(save_path, 'w') as f:
                for item in KITTI_annotations:
                    value_list = list(item.values())
                    for value in value_list:
                        if isinstance(value, list):
                            for v in value:
                                f.write(str(round(v, 2)) + '_')
                        else:

```

```
        if isinstance(value, str):
            f.write(value + ' ')
        else:
            f.write(str(round(value, 2)) + ' ')
f.write('\n')
```

APPENDIX C

TRAINING AND TESTING ALGORITHM (OPENPCDET DEVELOPMENT TEAM, 2020)

Listing C.3: train.py

```
import _init_path
import argparse
import datetime
import glob
import os
from pathlib import Path
from test import repeat_eval_ckpt

import torch
import torch.nn as nn
from tensorboardX import SummaryWriter

from pcdet.config import cfg, cfg_from_list, cfg_from_yaml_file, log_config_to_file
from pcdet.datasets import build_dataloader
from pcdet.models import build_network, model_fn_decorator
from pcdet.utils import common_utils
from train_utils.optimization import build_optimizer, build_scheduler
from train_utils.train_utils import train_model

def parse_config():
    parser = argparse.ArgumentParser(description='arg_parser')
    parser.add_argument('--cfg_file', type=str, default=None, help='specify the
        config_for_training')

    parser.add_argument('--batch_size', type=int, default=None, required=False, help
        ='batch_size_for_training')
    parser.add_argument('--epochs', type=int, default=None, required=False, help='
        number_of_epochs_to_train_for')
    parser.add_argument('--workers', type=int, default=4, help='number_of_workers_
        for_dataloader')
    parser.add_argument('--extra_tag', type=str, default='default', help='extra_tag_
        for_this_experiment')
    parser.add_argument('--ckpt', type=str, default=None, help='checkpoint_to_start_
        from')
    parser.add_argument('--pretrained_model', type=str, default=None, help='
        pretrained_model')
    parser.add_argument('--launcher', choices=['none', 'pytorch', 'slurm'], default=
        'none')
    parser.add_argument('--tcp_port', type=int, default=18888, help='tcp_port_for_
        distributed_training')
    parser.add_argument('--sync_bn', action='store_true', default=False, help='
        whether_to_use_sync_bn')
    parser.add_argument('--fix_random_seed', action='store_true', default=False,
        help='')
    parser.add_argument('--ckpt_save_interval', type=int, default=1, help='number_of
        _training_epochs')
    parser.add_argument('--local_rank', type=int, default=0, help='local_rank_for_
        distributed_training')
```

```

parser.add_argument('--max_ckpt_save_num', type=int, default=30, help='max_
number_of_saved_checkpoint')
parser.add_argument('--merge_all_iters_to_one_epoch', action='store_true',
default=False, help='')
parser.add_argument('--set', dest='set_cfgs', default=None, nargs=argparse.
REMAINDER,
help='set_extra_config_keys_if_needed')

parser.add_argument('--max_waiting_mins', type=int, default=0, help='max_waiting
minutes')
parser.add_argument('--start_epoch', type=int, default=0, help='')
parser.add_argument('--num_epochs_to_eval', type=int, default=0, help='number_of
checkpoints_to_be_evaluated')
parser.add_argument('--save_to_file', action='store_true', default=False, help='
')

args = parser.parse_args()

cfg_from_yaml_file(args.cfg_file, cfg)
cfg.TAG = Path(args.cfg_file).stem
cfg.EXP_GROUP_PATH = '/' + join(args.cfg_file.split('/')[1:-1]) # remove 'cfgs'
and 'xxx.yaml'

if args.set_cfgs is not None:
    cfg_from_list(args.set_cfgs, cfg)

return args, cfg

def main():
args, cfg = parse_config()
if args.launcher == 'none':
    dist_train = False
    total_gpus = 1
else:
    total_gpus, cfg.LOCAL_RANK = getattr(common_utils, 'init_dist_%s' % args.
launcher)(
args.tcp_port, args.local_rank, backend='nccl'
)
    dist_train = True

if args.batch_size is None:
args.batch_size = cfg.OPTIMIZATION.BATCH_SIZE_PER_GPU
else:
    assert args.batch_size % total_gpus == 0, 'Batch_size_should_match_the_
number_of_gpus'
    args.batch_size = args.batch_size // total_gpus

args.epochs = cfg.OPTIMIZATION.NUMEPOCHS if args.epochs is None else args.
epochs

if args.fix_random_seed:
    common_utils.set_random_seed(666)

output_dir = cfg.ROOT_DIR / 'output' / cfg.EXP_GROUP_PATH / cfg.TAG / args.
extra_tag
ckpt_dir = output_dir / 'ckpt'
output_dir.mkdir(parents=True, exist_ok=True)
ckpt_dir.mkdir(parents=True, exist_ok=True)

```

```

log_file = output_dir / ('log_train_%s.txt' % datetime.datetime.now().strftime('%Y%m%d-%H%M%S'))
logger = common_utils.create_logger(log_file, rank=cfg.LOCAL_RANK)

# log to file
logger.info('*****Start_logging*****')
gpu_list = os.environ['CUDA_VISIBLE_DEVICES'] if 'CUDA_VISIBLE_DEVICES' in os.
    environ.keys() else 'ALL'
logger.info('CUDA_VISIBLE_DEVICES=%s' % gpu_list)

if dist_train:
    logger.info('total_batch_size:%d' % (total_gpus * args.batch_size))
for key, val in vars(args).items():
    logger.info('{:16}{}'.format(key, val))
log_config_to_file(cfg, logger=logger)
if cfg.LOCAL_RANK == 0:
    os.system('cp %s %s' % (args.cfg_file, output_dir))

tb_log = SummaryWriter(log_dir=str(output_dir / 'tensorboard')) if cfg.
    LOCAL_RANK == 0 else None

# ----- create dataloader & network & optimizer

train_set, train_loader, train_sampler = build_dataloader(
    dataset_cfg=cfg.DATA_CONFIG,
    class_names=cfg.CLASS_NAMES,
    batch_size=args.batch_size,
    dist=dist_train, workers=args.workers,
    logger=logger,
    training=True,
    merge_all_iters_to_one_epoch=args.merge_all_iters_to_one_epoch,
    total_epochs=args.epochs
)

model = build_network(model_cfg=cfg.MODEL, num_class=len(cfg.CLASS_NAMES),
    dataset=train_set)
if args.sync_bn:
    model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model)
model.cuda()

optimizer = build_optimizer(model, cfg.OPTIMIZATION)

# load checkpoint if it is possible
start_epoch = it = 0
last_epoch = -1
if args.pretrained_model is not None:
    model.load_params_from_file(filename=args.pretrained_model, to_cpu=
        dist_train, logger=logger)

if args.ckpt is not None:
    it, start_epoch = model.load_params_with_optimizer(args.ckpt, to_cpu=
        dist_train, optimizer=optimizer, logger=logger)
    last_epoch = start_epoch + 1
else:
    ckpt_list = glob.glob(str(ckpt_dir / '*checkpoint_epoch_*.pth'))
    if len(ckpt_list) > 0:
        ckpt_list.sort(key=os.path.getmtime)
        it, start_epoch = model.load_params_with_optimizer(

```

```

        ckpt_list[-1], to_cpu=dist_train, optimizer=optimizer, logger=logger
    )
    last_epoch = start_epoch + 1

model.train() # before wrap to DistributedDataParallel to support fixed some
              # parameters
if dist_train:
    model = nn.parallel.DistributedDataParallel(model, device_ids=[cfg.
        LOCALRANK % torch.cuda.device_count()])
logger.info(model)

lr_scheduler, lr_warmup_scheduler = build_scheduler(
    optimizer, total_iters_each_epoch=len(train_loader), total_epochs=args.
        epochs,
    last_epoch=last_epoch, optim_cfg=cfg.OPTIMIZATION
)

# ----- start training -----
logger.info('*****Start_training_%s/%s(%s)
            *****'
            % (cfg.EXP_GROUP_PATH, cfg.TAG, args.extra_tag))
train_model(
    model,
    optimizer,
    train_loader,
    model_func=model_fn_decorator(),
    lr_scheduler=lr_scheduler,
    optim_cfg=cfg.OPTIMIZATION,
    start_epoch=start_epoch,
    total_epochs=args.epochs,
    start_iter=it,
    rank=cfg.LOCALRANK,
    tb_log=tb.log,
    ckpt_save_dir=ckpt_dir,
    train_sampler=train_sampler,
    lr_warmup_scheduler=lr_warmup_scheduler,
    ckpt_save_interval=args.ckpt_save_interval,
    max_ckpt_save_num=args.max_ckpt_save_num,
    merge_all_iters_to_one_epoch=args.merge_all_iters_to_one_epoch
)

if hasattr(train_set, 'use_shared_memory') and train_set.use_shared_memory:
    train_set.clean_shared_memory()

logger.info('*****End_training_%s/%s(%s)*****\n\n'
            % (cfg.EXP_GROUP_PATH, cfg.TAG, args.extra_tag))

logger.info('*****Start_evaluation_%s/%s(%s)
            *****' %
            (cfg.EXP_GROUP_PATH, cfg.TAG, args.extra_tag))
test_set, test_loader, sampler = build_dataloader(
    dataset_cfg=cfg.DATA_CONFIG,
    class_names=cfg.CLASS_NAMES,
    batch_size=args.batch_size,
    dist=dist_train, workers=args.workers, logger=logger, training=False
)
eval_output_dir = output_dir / 'eval' / 'eval_with_train'
eval_output_dir.mkdir(parents=True, exist_ok=True)

```

```

args.start_epoch = max(args.epochs - args.num_epochs_to_eval, 0) # Only
    evaluate the last args.num_epochs_to_eval epochs

repeat_eval_ckpt(
    model.module if dist_train else model,
    test_loader, args, eval_output_dir, logger, ckpt_dir,
    dist_test=dist_train
)
logger.info('*****End_evaluation_%%s/%%s(%%s)
    *****' %
    (cfg.EXP_GROUP_PATH, cfg.TAG, args.extra_tag))

if __name__ == '__main__':
    main()

```

Listing C.4: test.py

```

import _init_path
import argparse
import datetime
import glob
import os
import re
import time
from pathlib import Path

import numpy as np
import torch
from tensorboardX import SummaryWriter

from eval_utils import eval_utils
from pcdet.config import cfg, cfg_from_list, cfg_from_yaml_file, log_config_to_file
from pcdet.datasets import build_dataloader
from pcdet.models import build_network
from pcdet.utils import common_utils

def parse_config():
    parser = argparse.ArgumentParser(description='arg_parser')
    parser.add_argument('--cfg_file', type=str, default=None, help='specify the
        config_for_training')

    parser.add_argument('--batch_size', type=int, default=None, required=False, help
        ='batch_size_for_training')
    parser.add_argument('--workers', type=int, default=4, help='number_of_workers_
        for_dataloader')
    parser.add_argument('--extra_tag', type=str, default='default', help='extra_tag_
        for_this_experiment')
    parser.add_argument('--ckpt', type=str, default=None, help='checkpoint_to_start_
        from')
    parser.add_argument('--launcher', choices=['none', 'pytorch', 'slurm'], default=
        'none')
    parser.add_argument('--tcp_port', type=int, default=18888, help='tcp_port_for_
        distributed_training')
    parser.add_argument('--local_rank', type=int, default=0, help='local_rank_for_
        distributed_training')

```

```

parser.add_argument('--set', dest='set_cfgs', default=None, nargs=argparse.
    REMAINDER,
                    help='set_extra_config_keys_if_needed')

parser.add_argument('--max_waiting_mins', type=int, default=30, help='max_
    waiting_minutes')
parser.add_argument('--start_epoch', type=int, default=0, help='')
parser.add_argument('--eval_tag', type=str, default='default', help='eval_tag_
    for_this_experiment')
parser.add_argument('--eval_all', action='store_true', default=False, help='
    whether_to_evaluate_all_checkpoints')
parser.add_argument('--ckpt_dir', type=str, default=None, help='specify_a_ckpt_
    directory_to_be_evaluated_if_needed')
parser.add_argument('--save_to_file', action='store_true', default=False, help='
    ')

args = parser.parse_args()

cfg_from_yaml_file(args.cfg_file, cfg)
cfg.TAG = Path(args.cfg_file).stem
cfg.EXP_GROUP_PATH = '/'.join(args.cfg_file.split('/')[1:-1]) # remove 'cfgs'
    and 'xxx.yaml'

np.random.seed(1024)

if args.set_cfgs is not None:
    cfg_from_list(args.set_cfgs, cfg)

return args, cfg

def eval_single_ckpt(model, test_loader, args, eval_output_dir, logger, epoch_id,
    dist_test=False):
    # load checkpoint
    model.load_params_from_file(filename=args.ckpt, logger=logger, to_cpu=dist_test)
    model.cuda()

    # start evaluation
    eval_utils.eval_one_epoch(
        cfg, model, test_loader, epoch_id, logger, dist_test=dist_test,
        result_dir=eval_output_dir, save_to_file=args.save_to_file
    )

def get_no_evaluated_ckpt(ckpt_dir, ckpt_record_file, args):
    ckpt_list = glob.glob(os.path.join(ckpt_dir, '*checkpoint_epoch_*.pth'))
    ckpt_list.sort(key=os.path.getmtime)
    evaluated_ckpt_list = [float(x.strip()) for x in open(ckpt_record_file, 'r').
        readlines()]

    for cur_ckpt in ckpt_list:
        num_list = re.findall('checkpoint_epoch_(.*)\.pth', cur_ckpt)
        if num_list.__len__() == 0:
            continue

        epoch_id = num_list[-1]
        if 'optim' in epoch_id:
            continue

```

```

        if float(epoch_id) not in evaluated_ckpt_list and int(float(epoch_id)) >=
            args.start_epoch:
            return epoch_id, cur_ckpt
    return -1, None

def repeat_eval_ckpt(model, test_loader, args, eval_output_dir, logger, ckpt_dir,
                    dist_test=False):
    # evaluated ckpt record
    ckpt_record_file = eval_output_dir / ('eval_list_%s.txt' % cfg.DATA_CONFIG.
        DATA_SPLIT['test'])
    with open(ckpt_record_file, 'a'):
        pass

    # tensorboard log
    if cfg.LOCALRANK == 0:
        tb_log = SummaryWriter(log_dir=str(eval_output_dir / ('tensorboard_%s' % cfg
            .DATA_CONFIG.DATA_SPLIT['test'])))
        total_time = 0
        first_eval = True

    while True:
        # check whether there is checkpoint which is not evaluated
        cur_epoch_id, cur_ckpt = get_no_evaluated_ckpt(ckpt_dir, ckpt_record_file,
            args)
        if cur_epoch_id == -1 or int(float(cur_epoch_id)) < args.start_epoch:
            wait_second = 30
            if cfg.LOCALRANK == 0:
                print('Wait %s seconds for next check (progress: %1f / %d minutes):
                    %s\r'
                    % (wait_second, total_time * 1.0 / 60, args.max_waiting_mins,
                        ckpt_dir), end='', flush=True)
                time.sleep(wait_second)
                total_time += 30
            if total_time > args.max_waiting_mins * 60 and (first_eval is False):
                break
            continue

        total_time = 0
        first_eval = False

        model.load_params_from_file(filename=cur_ckpt, logger=logger, to_cpu=
            dist_test)
        model.cuda()

        # start evaluation
        cur_result_dir = eval_output_dir / ('epoch_%s' % cur_epoch_id) / cfg.
            DATA_CONFIG.DATA_SPLIT['test']
        tb_dict = eval_utils.eval_one_epoch(
            cfg, model, test_loader, cur_epoch_id, logger, dist_test=dist_test,
            result_dir=cur_result_dir, save_to_file=args.save_to_file
        )

        if cfg.LOCALRANK == 0:
            for key, val in tb_dict.items():
                tb_log.add_scalar(key, val, cur_epoch_id)

        # record this epoch which has been evaluated
        with open(ckpt_record_file, 'a') as f:

```

```

        print('%s' % cur_epoch_id, file=f)
    logger.info('Epoch_%s_has_been_evaluated' % cur_epoch_id)

def main():
    args, cfg = parse_config()
    if args.launcher == 'none':
        dist_test = False
        total_gpus = 1
    else:
        total_gpus, cfg.LOCAL_RANK = getattr(common_utils, 'init_dist_%s' % args.
            launcher)(
            args.tcp_port, args.local_rank, backend='nccl'
        )
        dist_test = True

    if args.batch_size is None:
        args.batch_size = cfg.OPTIMIZATION.BATCH_SIZE_PER_GPU
    else:
        assert args.batch_size % total_gpus == 0, 'Batch_size_should_match_the_
            number_of_gpus'
        args.batch_size = args.batch_size // total_gpus

    output_dir = cfg.ROOT_DIR / 'output' / cfg.EXP_GROUP_PATH / cfg.TAG / args.
        extra_tag
    output_dir.mkdir(parents=True, exist_ok=True)

    eval_output_dir = output_dir / 'eval'

    if not args.eval_all:
        num_list = re.findall(r'\d+', args.ckpt) if args.ckpt is not None else []
        epoch_id = num_list[-1] if num_list.__len__() > 0 else 'no_number'
        eval_output_dir = eval_output_dir / ('epoch_%s' % epoch_id) / cfg.
            DATA_CONFIG.DATA_SPLIT['test']
    else:
        eval_output_dir = eval_output_dir / 'eval_all_default'

    if args.eval_tag is not None:
        eval_output_dir = eval_output_dir / args.eval_tag

    eval_output_dir.mkdir(parents=True, exist_ok=True)
    log_file = eval_output_dir / ('log_eval_%s.txt' % datetime.datetime.now().
        strftime('%Y%m%d-%H%M%S'))
    logger = common_utils.create_logger(log_file, rank=cfg.LOCAL_RANK)

    # log to file
    logger.info('***** Start_logging*****')
    gpu_list = os.environ['CUDA_VISIBLE_DEVICES'] if 'CUDA_VISIBLE_DEVICES' in os.
        environ.keys() else 'ALL'
    logger.info('CUDA_VISIBLE_DEVICES=%s' % gpu_list)

    if dist_test:
        logger.info('total_batch_size:%d' % (total_gpus * args.batch_size))
    for key, val in vars(args).items():
        logger.info('{:16}{}'.format(key, val))
    log_config_to_file(cfg, logger=logger)

    ckpt_dir = args.ckpt_dir if args.ckpt_dir is not None else output_dir / 'ckpt'

```

```

test_set, test_loader, sampler = build_dataloader(
    dataset_cfg=cfg.DATA_CONFIG,
    class_names=cfg.CLASS_NAMES,
    batch_size=args.batch_size,
    dist=dist_test, workers=args.workers, logger=logger, training=False
)

model = build_network(model_cfg=cfg.MODEL, num_class=len(cfg.CLASS_NAMES),
    dataset=test_set)
with torch.no_grad():
    if args.eval_all:
        repeat_eval_ckpt(model, test_loader, args, eval_output_dir, logger,
            ckpt_dir, dist_test=dist_test)
    else:
        eval_single_ckpt(model, test_loader, args, eval_output_dir, logger,
            epoch_id, dist_test=dist_test)

if __name__ == '__main__':
    main()

```

Listing C.5: kitti_dataset.yaml

```

DATASET: 'KittiDataset'
DATA_PATH: '../data/kitti'

POINT_CLOUD_RANGE: [0, -40, -3, 70.4, 40, 1]

DATA_SPLIT: {
    'train': train,
    'test': val
}

INFO_PATH: {
    'train': [kitti_infos_train.pkl],
    'test': [kitti_infos_val.pkl],
}

GET_ITEM_LIST: ["points"]
FOV_POINTS_ONLY: True

DATA_AUGMENTOR:
    DISABLE_AUG_LIST: ['placeholder']
    AUG_CONFIG_LIST:
        - NAME: gt_sampling
          USE_ROAD_PLANE: True
          DB_INFO_PATH:
            - kitti_dbinfos_train.pkl
          PREPARE: {
              filter_by_min_points: ['Car:5', 'Pedestrian:5', 'Cyclist:5'],
              filter_by_difficulty: [-1],
          }

        SAMPLE_GROUPS: ['Car:20', 'Pedestrian:15', 'Cyclist:15']
        NUM_POINT_FEATURES: 4
        DATABASE_WITH_FAKELIDAR: False
        REMOVE_EXTRA_WIDTH: [0.0, 0.0, 0.0]
        LIMIT_WHOLE_SCENE: True

```

```

- NAME: random_world_flip
  ALONG_AXIS_LIST: ['x']

- NAME: random_world_rotation
  WORLD_ROT_ANGLE: [-0.78539816, 0.78539816]

- NAME: random_world_scaling
  WORLD_SCALE_RANGE: [0.95, 1.05]

POINT_FEATURE_ENCODING: {
  encoding_type: absolute_coordinates_encoding,
  used_feature_list: ['x', 'y', 'z', 'intensity'],
  src_feature_list: ['x', 'y', 'z', 'intensity'],
}

DATA_PROCESSOR:
- NAME: mask_points_and_boxes_outside_range
  REMOVE_OUTSIDE_BOXES: True

- NAME: shuffle_points
  SHUFFLE_ENABLED: {
    'train': True,
    'test': False
  }

- NAME: transform_points_to_voxels
  VOXEL_SIZE: [0.05, 0.05, 0.1]
  MAX_POINTS_PER_VOXEL: 5
  MAX_NUMBER_OF_VOXELS: {
    'train': 16000,
    'test': 40000
  }

```

Listing C.6: PartA2_{free}.yaml

```

CLASS_NAMES: ['Car', 'Pedestrian', 'Cyclist']

DATA_CONFIG:
  _BASE_CONFIG_: cfgs/dataset_configs/kitti_dataset.yaml

MODEL:
  NAME: PointRCNN

  VFE:
    NAME: MeanVFE

  BACKBONE_3D:
    NAME: UNetV2
    RETURN_ENCODED_TENSOR: False

  POINT_HEAD:
    NAME: PointIntraPartOffsetHead
    CLS_FC: [128, 128]
    PART_FC: [128, 128]

```

```

REG.FC: [128, 128]
CLASS_AGNOSTIC: False
USE_POINT_FEATURES_BEFORE_FUSION: False
TARGET_CONFIG:
  GT_EXTRA_WIDTH: [0.2, 0.2, 0.2]
  BOX_CODER: PointResidualCoder
  BOX_CODER_CONFIG: {
    'use_mean_size': True,
    'mean_size': [
      [3.9, 1.6, 1.56],
      [0.8, 0.6, 1.73],
      [1.76, 0.6, 1.73]
    ]
  }
}

LOSS_CONFIG:
  LOSS_REG: WeightedSmoothL1Loss
  LOSS_WEIGHTS: {
    'point_cls_weight': 1.0,
    'point_box_weight': 1.0,
    'point_part_weight': 1.0,
    'code_weights': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
  }
}

ROLHEAD:
  NAME: PartA2FCHead
  CLASS_AGNOSTIC: True

  SHARED_FC: [256, 256, 256]
  CLS_FC: [256, 256]
  REG_FC: [256, 256]
  DP_RATIO: 0.3
  DISABLE_PART: True
  SEG_MASK_SCORE_THRESH: 0.0

NMS_CONFIG:
  TRAIN:
    NMS_TYPE: nms_gpu
    MULTICLASSES_NMS: False
    NMS_PRE_MAXSIZE: 9000
    NMS_POST_MAXSIZE: 512
    NMS_THRESH: 0.8
  TEST:
    NMS_TYPE: nms_gpu
    MULTICLASSES_NMS: False
    NMS_PRE_MAXSIZE: 9000
    NMS_POST_MAXSIZE: 100
    NMS_THRESH: 0.85

ROLAWARE_POOL:
  POOL_SIZE: 12
  NUM_FEATURES: 128
  MAX_POINTS_PER_VOXEL: 128

TARGET_CONFIG:
  BOX_CODER: ResidualCoder
  ROL_PER_IMAGE: 128
  FG_RATIO: 0.5

```

```

SAMPLE_ROLBY_EACH_CLASS: True
CLS_SCORE_TYPE: roi_iou

CLS_FG_THRESH: 0.75
CLS_BG_THRESH: 0.25
CLS_BG_THRESH_LO: 0.1
HARD_BG_RATIO: 0.8

REG_FG_THRESH: 0.65

LOSS_CONFIG:
  CLS_LOSS: BinaryCrossEntropy
  REG_LOSS: smooth_l1
  CORNER_LOSS_REGULARIZATION: True
  LOSS_WEIGHTS: {
    'rcnn_cls_weight': 1.0,
    'rcnn_reg_weight': 1.0,
    'rcnn_corner_weight': 1.0,
    'code_weights': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
  }

POST_PROCESSING:
  RECALL_THRESH_LIST: [0.3, 0.5, 0.7]
  SCORE_THRESH: 0.1
  OUTPUT_RAW_SCORE: False

EVAL_METRIC: kitti

NMS_CONFIG:
  MULTICLASSES_NMS: False
  NMS_TYPE: nms_gpu
  NMS_THRESH: 0.1
  NMS_PRE_MAXSIZE: 4096
  NMS_POST_MAXSIZE: 500

OPTIMIZATION:
  BATCH_SIZE_PER_GPU: 4
  NUM_EPOCHS: 80

  OPTIMIZER: adam_onecycle
  LR: 0.003
  WEIGHT_DECAY: 0.01
  MOMENTUM: 0.9

  MOMS: [0.95, 0.85]
  PCT_START: 0.4
  DIV_FACTOR: 10
  DECAY_STEP_LIST: [35, 45]
  LR_DECAY: 0.1
  LR_CLIP: 0.0000001

  LR_WARMUP: False
  WARMUP_EPOCH: 1

  GRAD_NORM_CLIP: 10

```

Listing C.7: *pv_rnn.yaml*

```

CLASS_NAMES: ['Car', 'Pedestrian', 'Cyclist']

DATA_CONFIG:
  _BASE_CONFIG_: cfgs/dataset_configs/kitti_dataset.yaml
  DATA_AUGMENTOR:
    DISABLE_AUG_LIST: ['placeholder']
    AUG_CONFIG_LIST:
      - NAME: gt_sampling
        USE_ROAD_PLANE: True
        DB_INFO_PATH:
          - kitti_dbinfos_train.pkl
        PREPARE: {
          filter_by_min_points: ['Car:5', 'Pedestrian:5', 'Cyclist:5'],
          filter_by_difficulty: [-1],
        }

        SAMPLE_GROUPS: ['Car:15', 'Pedestrian:10', 'Cyclist:10']
        NUM_POINT_FEATURES: 4
        DATABASE_WITH_FAKELIDAR: False
        REMOVE_EXTRA_WIDTH: [0.0, 0.0, 0.0]
        LIMIT_WHOLE_SCENE: False

      - NAME: random_world_flip
        ALONG_AXIS_LIST: ['x']

      - NAME: random_world_rotation
        WORLD_ROT_ANGLE: [-0.78539816, 0.78539816]

      - NAME: random_world_scaling
        WORLD_SCALE_RANGE: [0.95, 1.05]

MODEL:
  NAME: PVRCNN

  VFE:
    NAME: MeanVFE

  BACKBONE_3D:
    NAME: VoxelBackBone8x

  MAP_TO_BEV:
    NAME: HeightCompression
    NUM_BEV_FEATURES: 256

  BACKBONE_2D:
    NAME: BaseBEVBackbone

    LAYER_NUMS: [5, 5]
    LAYER_STRIDES: [1, 2]
    NUM_FILTERS: [128, 256]
    UPSAMPLE_STRIDES: [1, 2]
    NUM_UPSAMPLE_FILTERS: [256, 256]

  DENSE_HEAD:
    NAME: AnchorHeadSingle
    CLASS_AGNOSTIC: False

    USE_DIRECTION_CLASSIFIER: True
    DIR_OFFSET: 0.78539

```

DIR_LIMIT_OFFSET: 0.0
NUM_DIR_BINS: 2

```
ANCHOR_GENERATOR_CONFIG: [  
  {  
    'class_name': 'Car',  
    'anchor_sizes': [[3.9, 1.6, 1.56]],  
    'anchor_rotations': [0, 1.57],  
    'anchor_bottom_heights': [-1.78],  
    'align_center': False,  
    'feature_map_stride': 8,  
    'matched_threshold': 0.6,  
    'unmatched_threshold': 0.45  
  },  
  {  
    'class_name': 'Pedestrian',  
    'anchor_sizes': [[0.8, 0.6, 1.73]],  
    'anchor_rotations': [0, 1.57],  
    'anchor_bottom_heights': [-0.6],  
    'align_center': False,  
    'feature_map_stride': 8,  
    'matched_threshold': 0.5,  
    'unmatched_threshold': 0.35  
  },  
  {  
    'class_name': 'Cyclist',  
    'anchor_sizes': [[1.76, 0.6, 1.73]],  
    'anchor_rotations': [0, 1.57],  
    'anchor_bottom_heights': [-0.6],  
    'align_center': False,  
    'feature_map_stride': 8,  
    'matched_threshold': 0.5,  
    'unmatched_threshold': 0.35  
  }  
]
```

TARGET_ASSIGNER_CONFIG:
NAME: AxisAlignedTargetAssigner
POS_FRACTION: -1.0
SAMPLE_SIZE: 512
NORM_BY_NUM_EXAMPLES: False
MATCH_HEIGHT: False
BOX_CODER: ResidualCoder

LOSS_CONFIG:
LOSS_WEIGHTS: {
 'cls_weight': 1.0,
 'loc_weight': 2.0,
 'dir_weight': 0.2,
 'code_weights': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
}

PFE:
NAME: VoxelSetAbstraction
POINT_SOURCE: raw_points
NUM_KEYPOINTS: 2048
NUM_OUTPUT_FEATURES: 128
SAMPLE_METHOD: FPS

```

FEATURES.SOURCE: ['bev', 'x_conv1', 'x_conv2', 'x_conv3', 'x_conv4', '
raw_points']
SALAYER:
  raw_points:
    MLPS: [[16, 16], [16, 16]]
    POOL_RADIUS: [0.4, 0.8]
    NSAMPLE: [16, 16]
  x_conv1:
    DOWNSAMPLEFACTOR: 1
    MLPS: [[16, 16], [16, 16]]
    POOL_RADIUS: [0.4, 0.8]
    NSAMPLE: [16, 16]
  x_conv2:
    DOWNSAMPLEFACTOR: 2
    MLPS: [[32, 32], [32, 32]]
    POOL_RADIUS: [0.8, 1.2]
    NSAMPLE: [16, 32]
  x_conv3:
    DOWNSAMPLEFACTOR: 4
    MLPS: [[64, 64], [64, 64]]
    POOL_RADIUS: [1.2, 2.4]
    NSAMPLE: [16, 32]
  x_conv4:
    DOWNSAMPLEFACTOR: 8
    MLPS: [[64, 64], [64, 64]]
    POOL_RADIUS: [2.4, 4.8]
    NSAMPLE: [16, 32]

```

POINT_HEAD:

```

NAME: PointHeadSimple
CLS_FC: [256, 256]
CLASS_AGNOSTIC: True
USE_POINT_FEATURES_BEFORE_FUSION: True
TARGET_CONFIG:
  GT_EXTRA_WIDTH: [0.2, 0.2, 0.2]
LOSS_CONFIG:
  LOSS_REG: smooth-l1
  LOSS_WEIGHTS: {
    'point_cls_weight': 1.0,
  }

```

ROLHEAD:

```

NAME: PVRCNNHead
CLASS_AGNOSTIC: True

```

```

SHARED_FC: [256, 256]
CLS_FC: [256, 256]
REG_FC: [256, 256]
DP_RATIO: 0.3

```

NMS_CONFIG:

```

TRAIN:
  NMS_TYPE: nms-gpu
  MULTI_CLASSES_NMS: False
  NMS_PRE_MAXSIZE: 9000
  NMS_POST_MAXSIZE: 512
  NMS_THRESH: 0.8
TEST:
  NMS_TYPE: nms-gpu

```

```

MULTI_CLASSES_NMS: False
NMS_PRE_MAXSIZE: 1024
NMS_POST_MAXSIZE: 100
NMS_THRESH: 0.7

ROI_GRID_POOL:
  GRID_SIZE: 6
  MLPS: [[64, 64], [64, 64]]
  POOL_RADIUS: [0.8, 1.6]
  NSAMPLE: [16, 16]
  POOL_METHOD: max_pool

TARGET_CONFIG:
  BOX_CODER: ResidualCoder
  ROI_PER_IMAGE: 128
  FG_RATIO: 0.5

  SAMPLE_ROI_BY_EACH_CLASS: True
  CLS_SCORE_TYPE: roi_iou

  CLS_FG_THRESH: 0.75
  CLS_BG_THRESH: 0.25
  CLS_BG_THRESH_LO: 0.1
  HARD_BG_RATIO: 0.8

  REG_FG_THRESH: 0.55

LOSS_CONFIG:
  CLS_LOSS: BinaryCrossEntropy
  REG_LOSS: smooth_l1
  CORNER_LOSS_REGULARIZATION: True
  LOSS_WEIGHTS: {
    'rcnn_cls_weight': 1.0,
    'rcnn_reg_weight': 1.0,
    'rcnn_corner_weight': 1.0,
    'code_weights': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
  }

POST_PROCESSING:
  RECALL_THRESH_LIST: [0.3, 0.5, 0.7]
  SCORE_THRESH: 0.1
  OUTPUT_RAW_SCORE: False

  EVAL_METRIC: kitti

NMS_CONFIG:
  MULTI_CLASSES_NMS: False
  NMS_TYPE: nms_gpu
  NMS_THRESH: 0.1
  NMS_PRE_MAXSIZE: 4096
  NMS_POST_MAXSIZE: 500

OPTIMIZATION:
  BATCH_SIZE_PER_GPU: 2
  NUM_EPOCHS: 80

  OPTIMIZER: adam_onecycle
  LR: 0.01

```

```

WEIGHT_DECAY: 0.01
MOMENTUM: 0.9

MOMS: [0.95, 0.85]
PCT_START: 0.4
DIV_FACTOR: 10
DECAY_STEP_LIST: [35, 45]
LR_DECAY: 0.1
LR_CLIP: 0.0000001

LR_WARMUP: False
WARMUP_EPOCH: 1

GRAD_NORM_CLIP: 10

```

Listing C.8: pointpillar.yaml

```

CLASS_NAMES: ['Car', 'Pedestrian', 'Cyclist']

DATA_CONFIG:
  _BASE_CONFIG_: cfgs/dataset_configs/kitti_dataset.yaml
  POINT_CLOUD_RANGE: [0, -39.68, -3, 69.12, 39.68, 1]
  DATA_PROCESSOR:
    - NAME: mask_points_and_boxes_outside_range
      REMOVE_OUTSIDE_BOXES: True

    - NAME: shuffle_points
      SHUFFLE_ENABLED: {
        'train': True,
        'test': False
      }

    - NAME: transform_points_to_voxels
      VOXEL_SIZE: [0.16, 0.16, 4]
      MAX_POINTS_PER_VOXEL: 32
      MAX_NUMBER_OF_VOXELS: {
        'train': 16000,
        'test': 40000
      }
  DATA_AUGMENTOR:
    DISABLE_AUG_LIST: ['placeholder']
    AUG_CONFIG_LIST:
      - NAME: gt_sampling
        USE_ROAD_PLANE: True
        DB_INFO_PATH:
          - kitti_dbinfos_train.pkl
        PREPARE: {
          filter_by_min_points: ['Car:5', 'Pedestrian:5', 'Cyclist:5'],
          filter_by_difficulty: [-1],
        }

      SAMPLE_GROUPS: ['Car:15', 'Pedestrian:15', 'Cyclist:15']
      NUM_POINT_FEATURES: 4
      DATABASE_WITH_FAKELIDAR: False
      REMOVE_EXTRA_WIDTH: [0.0, 0.0, 0.0]
      LIMIT_WHOLE_SCENE: False

```

- NAME: random_world_flip
ALONG_AXIS_LIST: ['x']
- NAME: random_world_rotation
WORLD_ROT_ANGLE: [-0.78539816, 0.78539816]
- NAME: random_world_scaling
WORLD_SCALE_RANGE: [0.95, 1.05]

MODEL:

NAME: PointPillar

VFE:

NAME: PillarVFE
WITH_DISTANCE: False
USE_ABSLOTE_XYZ: True
USE_NORM: True
NUM_FILTERS: [64]

MAP_TO_BEV:

NAME: PointPillarScatter
NUM_BEV_FEATURES: 64

BACKBONE_2D:

NAME: BaseBEVBackbone
LAYER_NUMS: [3, 5, 5]
LAYER_STRIDES: [2, 2, 2]
NUM_FILTERS: [64, 128, 256]
UPSAMPLE_STRIDES: [1, 2, 4]
NUM_UPSAMPLE_FILTERS: [128, 128, 128]

DENSE_HEAD:

NAME: AnchorHeadSingle
CLASS_AGNOSTIC: False

USE_DIRECTION_CLASSIFIER: True
DIR_OFFSET: 0.78539
DIR_LIMIT_OFFSET: 0.0
NUM_DIR_BINS: 2

ANCHOR_GENERATOR_CONFIG: [

```
{
  'class_name': 'Car',
  'anchor_sizes': [[3.9, 1.6, 1.56]],
  'anchor_rotations': [0, 1.57],
  'anchor_bottom_heights': [-1.78],
  'align_center': False,
  'feature_map_stride': 2,
  'matched_threshold': 0.6,
  'unmatched_threshold': 0.45
},
{
  'class_name': 'Pedestrian',
  'anchor_sizes': [[0.8, 0.6, 1.73]],
  'anchor_rotations': [0, 1.57],
  'anchor_bottom_heights': [-0.6],
  'align_center': False,
  'feature_map_stride': 2,
  'matched_threshold': 0.5,
```

```

        'unmatched_threshold': 0.35
    },
    {
        'class_name': 'Cyclist',
        'anchor_sizes': [[1.76, 0.6, 1.73]],
        'anchor_rotations': [0, 1.57],
        'anchor_bottom_heights': [-0.6],
        'align_center': False,
        'feature_map_stride': 2,
        'matched_threshold': 0.5,
        'unmatched_threshold': 0.35
    }
]

```

TARGET_ASSIGNER_CONFIG:

```

NAME: AxisAlignedTargetAssigner
POS_FRACTION: -1.0
SAMPLE_SIZE: 512
NORM_BY_NUM_EXAMPLES: False
MATCH_HEIGHT: False
BOX_CODER: ResidualCoder

```

LOSS_CONFIG:

```

LOSS_WEIGHTS: {
    'cls_weight': 1.0,
    'loc_weight': 2.0,
    'dir_weight': 0.2,
    'code_weights': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
}

```

POST_PROCESSING:

```

RECALL_THRESH_LIST: [0.3, 0.5, 0.7]
SCORE_THRESH: 0.1
OUTPUT_RAW_SCORE: False

```

EVAL_METRIC: kitti

NMS_CONFIG:

```

MULTI_CLASSES_NMS: False
NMS_TYPE: nms_gpu
NMS_THRESH: 0.01
NMS_PRE_MAX_SIZE: 4096
NMS_POST_MAX_SIZE: 500

```

OPTIMIZATION:

```

BATCH_SIZE_PER_GPU: 4
NUM_EPOCHS: 80

```

```

OPTIMIZER: adam_onecycle
LR: 0.003
WEIGHT_DECAY: 0.01
MOMENTUM: 0.9

```

```

MOMS: [0.95, 0.85]
PCT_START: 0.4
DIV_FACTOR: 10
DECAY_STEP_LIST: [35, 45]
LR_DECAY: 0.1

```

LR_CLIP: 0.0000001

LR_WARMUP: False

WARMUP_EPOCH: 1

GRAD_NORM_CLIP: 10

APPENDIX D

PERMISSION TO REUSE FIGURES AND TABLE

Permissions to include previously published figure (2.2) may be found in file Shi - Point-Voxel Feature Set Abstraction for 3D Object Detection - Reuse Permission.pdf.

Permissions to include previously published figures (3.5 and 3.11) may be found in file Geiger - Vision meets robotics: The KITTI dataset - Reuse Permission.pdf.

Permissions to include previously published table (1.1) may be found in file Balasubarianam - Object Detection in Autonomous Vehicles: Status and Open Challenges - Reuse Permission.pdf.