3D Objects Detection and Recognition from Colour and LiDAR Data

Lian Kang

Thesis submitted in partial fulfillment of the requirements for

Master of Applied Science

in

Electrical and Computer Engineering

School of Electrical Engineering and Computer Science

Faculty of Engineering

University of Ottawa

Abstract

In recent years, autonomous driving vehicles has become a hot spot in both commercial and scientific domains and has brought new challenges to the machine vision field. How to detect and recognize objects in a complex real-world road environment has become one of the most important problems facing autonomous vehicles and their ability to make decisions on the road. This requires the vehicle to be able to process 3D visual information fast and to recognize objects even in bad weather or under occlusion.

With the development of deep learning technology and the significant improvements made to computing performance, software and hardware support now exists for advanced 3D object detection and recognition tasks. In addition, LiDAR scanners can obtain high quality data under different lighting conditions and can provide high-range-high-precision spatial information. Designing a detector that can process data collected by a LiDAR scanner will bring new developments in the field of autonomous driving.

In this thesis, a 3D object detector is proposed, with focal loss and Euler angle regression to optimize the model's performance, using bird's-eye view (BEV) map generated from LiDAR point cloud and RGB images as input data. While preprocessing the point cloud data, height information is extracted using height thresholding and rescaled to enhance the detail on the range of interest and intensity information is extracted on an expanded BEV map that keeps the density information without an extra channel. Results show that the proposed 3D object detector has a speed over 46 FPS on a TESLA V100-PCIE GPU and an average precision over 90%.

In addition to the full-scale detector, a 3D mini detector is also proposed. The mini detector can process the same input data as the full-scale detector 3 times faster with slightly lower precision.

Acknowledgements

I would like to thank my thesis supervisor, professor Pierre Payeur, for supervising my research and providing me with motivation and academic advice.

I would also like to thank my friends Zhiming Hu, Yue Zhang, Cornelius Nesen and Lisa Chunn for their company and support during my life in Canada.

In addition, I would like to thank all my family members. Thank them for always keeping me motivated and loved and for helping me become a person I always wanted to be.

Finally, I would like to thank SMART lab and University of Ottawa for providing a safe environment during COVID. I am grateful for staying healthy and happy through these years.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

3D	three dimensions
Adam	adaptive moment estimation
AP	average precision
B-Box	bounding box
BEV	bird’s-eye view
BN	batch normalization
CBL	convolutional layer, batch normalization and Leaky ReLU activation
CNN	convolutional neural network
CONV	convolutional layer
CPU	central processing unit
E-RPN	Euler angle region proposal network
FN	false negative
FP	false positive
FPN	feature pyramid network
GIoU	generalized intersection over union
GPU	graphics processing unit
IoU	intersection over union
LiDAR	light detection and ranging

mAP	mean average precision
MAXPOOL	pooling layer using max pooling method
MSE	mean square error
P-R curve	precision - recall curve
RAM	random-access memory
R-CNN	region based convolutional neural network
ReLU	rectified linear activation unit
RGB	red, green and blue
RMSprop	root mean squared propagation
ROI	region of interest
RPN	region proposal network
SCA	spatial channel attention
SGD	stochastic gradient descent
SSD	single shot detector
SVM	support vector machine
TN	true negative
TP	true positive
YOLO	you only look once

Chapter 1 Introduction

1.1 Motivation and Research Problem

In recent years, with the rise of artificial intelligence, autonomous driving has become a new focus of investment for companies such as Google, Tesla and Waymo. One of the important tasks of autonomous driving is to detect passing pedestrians, vehicles, and other objects on the road, so that the vehicle can make corresponding driving decisions. These requirements lead to the development of object detection and recognition technologies.

Object detection and recognition has always been one of the most important and challenging tasks of machine vision. The development of deep learning methods has brought unprecedented progress to this task. In recent years, with the improvement of computers' calculation abilities, hardware support became available to solve this problem and more research have changed their focus from 2D to 3D object detection and recognition.

The object recognition technology based on monocular vision achieves satisfactory application performance at low cost. Research shows that bringing depth information into the detection model can provide reliable spatial information (3D position, posture), and therefore further increase the detector's performance.

There are different types of sensors that can obtain depth information, such as stereo cameras, depth cameras and light detection and ranging (LiDAR) scanners. Stereo cameras can provide spatial information, but they are computationally expensive and are limited by high occlusion environment. The vision-based perception system of depth cameras is not adaptive to different lightings, which makes their performance unstable under different road and weather conditions, hence depth cameras are more useful for indoor machine visions tasks and not suitable for autonomous driving. Compared to depth cameras, LiDAR scanners are more expensive but crucially their performance is not affected by lighting conditions and as a result they can provide consistent high-precision spatial information. Hence, they've become increasingly popular in 3D object detection tasks.

Unlike RGB images with dense pixels, the point clouds collected by LiDAR scanners are sparse and continuous and because of the wide detecting range of a LiDAR scanner, some of the data are redundant. Moreover, the point cloud can reflect the shape and posture information of the real-world objects, but it does not carry texture or colour information. Therefore, the point cloud collected by a LiDAR scanner often needs to be preprocessed and combined with RGB images so that the detection will be more efficient while not losing 3D location information, colour and texture information.

Aside from including 3D information to improve the detection accuracy, autonomous driving tasks also require high detection speed, which becomes a challenge for processing 3D information. Both the preprocessing method and the detection algorithm need to be modified and optimized. Furthermore, inspired by tiny-sized 2D detection models, a mini 3D detector is imperative for further research.

Considering that the ultimate goal of autonomous driving is to avoid collisions and increase safety, the proposed detection and recognition model works on single frame for the RGB images and LiDAR capture rather than on inter-frame object detection, motion, or tracking. That technological choice is motivated by the fact that if the system is going to wait for a few frames to determine whether an object is present or not (e.g., when a new pedestrian appears in front of a car), rather than act immediately at every frame it receives from the sensors, the reaction time may get longer and any extra delay at processing more information may lead to an incident.

1.2 Objectives

The main objectives of this thesis are:

- By introducing Euler angle's regression to DarkNet-53 [1] convolutional neural networks, propose a 3D object detector to detect and classify cars, pedestrians and cyclists from LiDAR point cloud and RGB images in real-world road scenes.
- To reduce calculation and memory usage during training and testing process of the proposed detector, convert a LiDAR point cloud into a bird eye view (BEV) map using coordinate system transformation and height thresholding.

- Train and test the recognition performance of the proposed method on real-life data provided by KITTI vision benchmark suite [2] and merge focal loss [3] and generalized intersection over union (GIoU) loss [4] to handle the biased data and optimize the proposed model.

1.3 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 introduces the technical background that helps understand the rest of the thesis, which includes an introduction of LiDAR technology and the basic components of convolutional neural networks. Meanwhile, Chapter 2 reviews the literature about object detection and recognition using deep learning methods. Additionally, it summarizes the classic deep learning-based object detection algorithms and models, the detectors using a LiDAR point cloud as input data and open-source LiDAR point cloud datasets.

Chapter 3 proposes a 3D object detector based on Darknet-53 [1], introduces the model's methodology, and presents experimental details and results analysis respectively. A modified preprocessing method that converts the point clouds into BEV maps is explained in detail. Euler angle regression is added to the detection model's algorithm and focal loss [3] and GIoU [4] are merged to solve problems caused by biased input data.

Chapter 4 proposes a mini detector based on the detector model introduced in Chapter 3. By training and testing both models in the same environment and with the same datasets, the results are compared and analyzed.

Chapter 5 summarizes the proposed method and experiments of this thesis and proposes directions for future research.

Chapter 2 Technical Background and Literature Review

2.1 Technical Background

The sections below provide an overview about the technical concepts that are directly leveraged in the proposed models in Chapters 3 and 4.

2.1.1 LiDAR

LiDAR is an acronym for “light detection and ranging” or "laser imaging, detection and ranging". It is a method for determining range (variable distance) by targeting an object with lasers and measuring the time for the reflected light to return to the receiver.

Basically, a LiDAR scanner collects measurements through a remote sensing process and uses this data to create 3D point clouds and intensity maps of objects and environments. There are many ways to classify LiDAR. One way is to classify based on the number of LiDAR scanning beams. LiDAR can be divided into single beam LiDAR and multi-beam LiDAR.

As the name suggests, a single-beam LiDAR scan produces only one scan line at a time and the data it acquires is 2D data [5], so 3D information about the target object cannot be collected. A multi-beam LiDAR scanner can generate multiple laser beams at a time. Currently, multi-beam products on the market include 4, 8, 16, 32 and 64 channels, which can be subdivided into 2.5D LiDAR and 3D LiDAR. The main difference between 2.5D LiDAR and 3D LiDAR is the vertical field of view of the LiDAR. The vertical field of view of the former generally does not exceed 10°, while the latter can reach 30° or even 40°.

Another way is to classify LiDAR according to whether it contains mechanical rotating parts [5], including mechanical LiDAR, hybrid solid-state LiDAR and solid-state LiDAR.

Mechanical LiDAR scanners continuously rotates the launch head to change the faster and more accurate laser beam from "line" to "surface" and arranges multiple laser beams (i.e., 32 or 64 channel radar) in the vertical direction to form multiple surfaces to achieve the purpose of dynamic 3D scanning. The internal structure of mechanical LiDAR is complex, mainly including lasers, scanners, photodetectors and position and navigation devices. Since the high-frequency precise rotation is achieved by a complex mechanical structure, the hardware cost is high, and it is difficult to maintain stable operation for more than 50,000 hours.

The hybrid solid-state LiDAR scanners do not have visible rotating parts in appearance, but there are some mechanical rotating parts inside for the 360 full-view angles. This set of mechanical rotating parts is very small and can be built in, such as using micro-electromechanical (MEMS) technology scanning mirror. In order to distinguish such a product concept from the traditional "solid state" concept, the term " hybrid solid-state" is used.

Solid-state LiDAR is mainly divided into two types: flash LiDAR and optical phased array LiDAR [5].

The flash technology is a relatively mature technology in solid-state LiDAR (similar in principle to a camera). It can directly emit a large area of laser covering the detection area in a short time and complete the mapping of the surrounding environment with a high-sensitivity receiver. The flash LiDAR scanners can quickly record the entire scene, avoiding the hassle of moving targets during scanning. But because each shot is spread out over the entire field of view, this means that only a small fraction of the laser hits certain points, making long-range detection difficult.

The optical phased array LiDAR uses multiple light sources to form an array. By controlling the light-emitting time difference (relative phase) of each light source, the main beam with a specific direction is synthesized and the scanning in different directions can be realized by controlling it. The optical phased array requires the unit size to be no greater than half a wavelength. Currently in the industrial field, the operating wavelength of LiDAR is about 1000 nm, so the size of the array unit must not be greater than 500 nm, which is difficult to produce.

The LiDAR scanner used to collect point clouds in the public datasets considered for this thesis is the Velodyne’s HDL-64E. It is a 64 channels multi-beam mechanical LiDAR scanner. It offers

360° horizontal and 26.9° vertical field of view [6]. Although Velodyne's HDL-64E is not the best quality LiDAR scanner on the market, it served to build a high-quality 3D point cloud of real driving scenarios considered for the experiments in this thesis.

Aside from the 3D position (x, y, z), the point cloud matrix collected by the Velodyne LiDAR scanner provides a fourth value, which is the intensity of the corresponding point. The intensity value that the LiDAR scanner returns can be calculated by [5]:

$$P_R = \frac{P_E D_R^2 \rho \cos \alpha}{4R^2} \eta_{sys} \eta_{atm} \qquad (2.1)$$

Where P_R is the returned value, also known as the received echo power and P_E represents the emitted echo power. R is the distance between the scanned point and the LiDAR scanner. D_R represents the aperture diameter of the receiver. ρ represents the reflection index of the surface of the scanned point. α is the incidence angle of the laser beam on the surface of the scanned point. η_{sys} and η_{atm} are the transmission index of the system and the atmosphere. As shown in equation (2.1), with the same LiDAR scanner working in the same environment, the intensity over different points varies in the square of the scanning distance to the surface of the object that is represented by the corresponding point. This number also varies with the composition of the scan angle, surface composition, roughness, and moisture content of object reflecting the laser beam. For these reasons, LiDAR intensity does not always lead to consistent results. It must be used as a relative measurement. An advantage is that unlike passive vision sensors (such as RGB cameras), it is not influenced by shadows.

To provide reliable navigation for autonomous vehicles, as well as vision support for safe driving and decision-making, on-board sensors are needed to provide highly accurate and informative data about the real-world environment to achieve environmental perception and precise positioning. Compared to using RGB cameras to take pictures, LiDAR is not as sensitive to changes in lighting conditions and can work day and night, even with glare and shadows. As shown in Table 2.1, although the data provided by LiDAR can provide accurate 3D location, it cannot provide colour images. Therefore, in this thesis, both colour images provided by an RGB camera and 3D point clouds provided by LiDAR are used.

Table 2.1 Comparison between RGB camera and LiDAR.

	RGB camera	LiDAR
Basic Principles	Convert optical signal into digital image signal	Calculate the reflection time of the laser beam
Image	2D RGB image	3D point cloud
Colour Distinguish	Can distinguish colours	Unable to distinguish colours
Measuring distance	Far, up to several hundred meters	Close, a few meters to two hundred meters
Hardware cost	Low, US$300 to US$600	High, US$20,000 to US$100,000

2.1.2 Important Components of Convolutional Neural Networks (CNN)

• Convolutional Layer and Convolution Operation

Convolutional neural networks (CNN) [7] operate on the principle of convolutional layers composed of a set of filters, which can be regarded as a set of matrices, also called convolution kernels. We can convolve the filter with the input image to produce the output image.

To describe the convolution operation in a mathematical formula:

$$(f * g)(t) \triangleq \int_{-\infty}^{+\infty} f(\tau) g(t-\tau)\ \tau' \qquad (2.2)$$

where $(f * g)(t)$ are the functions that are being convolved, normally the image and the filter, i.e., the convolutional kernel. The t is the real number variable of functions f and g. $g(\tau)$ is the convolution of the function $f(t)$. $\tau'$is the differential of τ.

- Batch Normalization

Batch normalization [8] is a method that normalizes the input of a layer by recentring and rescaling as proposed by Ioffe and Szegedy. This helps to reduce internal covariate shift and improves the flow of gradients through the network.

Batch normalization can be divided into two steps: first normalize the input, then perform rescale and offset. Batch normalization is the process of converting data into a distribution with a mean of 0 and a standard deviation of 1.

- Loss Function

The loss function is used to estimate the degree of inconsistency between the predicted value f(x) of the model and the true value Y. It is a non-negative real-valued function, usually represented by L (Y, f(x)). The smaller the loss, the better the robustness of the model.

For bounding box regression, there are a few commonly used loss functions, such as intersection over union (IoU), Smooth L1 loss, etc. Here in this section, IoU (see Algorithm 2.1 below) and GIoU [4] (see Algorithm 2.2) are explained to help understand the network that this thesis proposes in Chapter 3. From experiments conducted by [4], it has been determined that generalized intersection over union (GIoU) keeps the major properties of IoU. This can be used to replace IoU and solve a problem that it holds. That is, IoU cannot optimize when facing non-overlapping bounding boxes. Conversely, GIoU shows an increased localization accuracy on 2D object recognition tasks.

Algorithm 2.1 Intersection over union (IoU) loss.

input: B^p as the coordinates of predicted bounding box and B^g as the coordinates of ground truth bounding box: $B^p = (x_1^p, y_1^p, x_2^p, y_2^p),\ B^g = (x_1^g, y_1^g, x_2^g, y_2^g)$

output: IoU loss L_{IoU}

1. To predict bounding box B^p, we need to ensure $x_2^p > x_1^p$ and $y_2^p > y_1^p$ by computing:

$$\hat{x_1^p} = min(x_1^p, x_2^p),\ \hat{x_2^p} = max(x_1^p, x_2^p),$$

$$\hat{y_1^p} = min(y_1^p, y_2^p),\ \hat{y_2^p} = max(y_1^p, y_2^p)$$

2. calculate the area of predicted bounding box B^p: $A^p = (\hat{x_2^p} - \hat{x_1^p}) \times (\hat{y_2^p} - \hat{y_1^p})$.

3. calculate the area of ground truth bounding box B^g: $A^g = (x_2^g - x_1^g) \times (y_2^g - y_1^g)$.

4. the intersection I between B^p and B^g has:

$$x_1^I = max(\hat{x_1^p}, x_1^g),\ x_2^I = min(\hat{x_2^p}, x_2^g),$$

$$y_1^I = max(\hat{y_1^p}, y_1^g),\ y_2^I = min(\hat{y_2^p}, y_2^g)$$

5. calculate I: $I = \begin{cases} (x_2^I - x_1^I) \times (y_2^I - y_1^I),\ if\ \ x_2^I > x_1^I,\ y_2^I > y_1^I \\ 0,\ otherwise \end{cases}$

6. calculate intersection over union (IoU):

$$IoU = \frac{I}{U},$$

where $U = A^p + A^g - I$.

7. calculate the loss: $L_{IoU} = 1 - IoU$

Algorithm 2.2 Generalized intersection over union (GIoU) loss [4].

input: B^p as the coordinates of predicted bounding box and B^g as the coordinates of ground truth bounding box: $B^p = (x_1^p, y_1^p, x_2^p, y_2^p)$, $B^g = (x_1^g, y_1^g, x_2^g, y_2^g)$

output: GIoU loss L_{GIoU}

1. To predict bounding box B^p, we need to ensure $x_2^p > x_1^p$ and $y_2^p > y_1^p$ by computing:

$$\hat{x_1^p} = min(x_1^p, x_2^p),\ \hat{x_2^p} = max(x_1^p, x_2^p),$$

$$\hat{y_1^p} = min(y_1^p, y_2^p),\ \hat{y_2^p} = max(y_1^p, y_2^p)$$

2. calculate the area of predicted bounding box B^p: $A^p = (\hat{x_2^p} - \hat{x_1^p}) \times (\hat{y_2^p} - \hat{y_1^p})$.

3. calculate the area of ground truth bounding box B^g: $A^g = (x_2^g - x_1^g) \times (y_2^g - y_1^g)$.

4. the intersection I between B^p and B^g has:

$$x_1^I = max(\hat{x_1^p}, x_1^g),\ x_2^I = min(\hat{x_2^p}, x_2^g),$$

$$y_1^I = max(\hat{y_1^p}, y_1^g),\ y_2^I = min(\hat{y_2^p}, y_2^g)$$

5. calculate I: $I = \begin{cases} (x_2^I - x_1^I) \times (y_2^I - y_1^I),\ if\ x_2^I > x_1^I,\ y_2^I > y_1^I \\ 0,\ otherwise \end{cases}$

6. calculate the coordinates of smallest enclosing box B^c:

$$x_1^c = min(\hat{x_1^p}, x_1^g),\ x_2^c = max(\hat{x_2^p}, x_2^g),$$

$$y_1^c = min(\hat{y_1^p}, y_1^g),\ y_2^c = max(\hat{y_2^p}, y_2^g)$$

7. calculate the area of smallest enclosing box B^c: $A^c = (x_2^c - x_1^c) \times (y_2^c - y_1^c)$

8. calculate intersection over union (IoU):

$IoU = \frac{I}{U}$, where $U = A^p + A^g - I$.

9. calculate generalized intersection over union (GIoU): $GIoU = IoU - \frac{A^c - U}{A^c}$

10. calculate GIoU loss: $L_{GIoU} = 1 - GIoU = 1 - IoU + \frac{A^c - U}{A^c}$

• Activation Function: Leaky ReLU [9]

Leaky Rectified Linear Unit, or Leaky ReLU, represents an activation function. The Leaky ReLU function can be represented by:

$$y = f(x_i) = \begin{cases} x_i, & if\ x_i > 0 \\ a_i \cdot x_i, & if\ x_i \leq 0 \end{cases} \quad (2.3)$$

It is a type of activation function based on ReLU [10], with a small slope for negative values instead of a flat slope, as shown in Figure 2.1.The slope coefficient is determined before training, i.e., it is not learnt during training. Compared with ReLU, leak assigns a non-zero slope to all negative values, where the leak is a very small constant (we use 0.1 in our experiments), so that some negative axis values are retained, and all negative axis information will not be lost. It improves the zero gradients situation of ReLU, but also loses part of the sparsity and adds a hyperparameter.

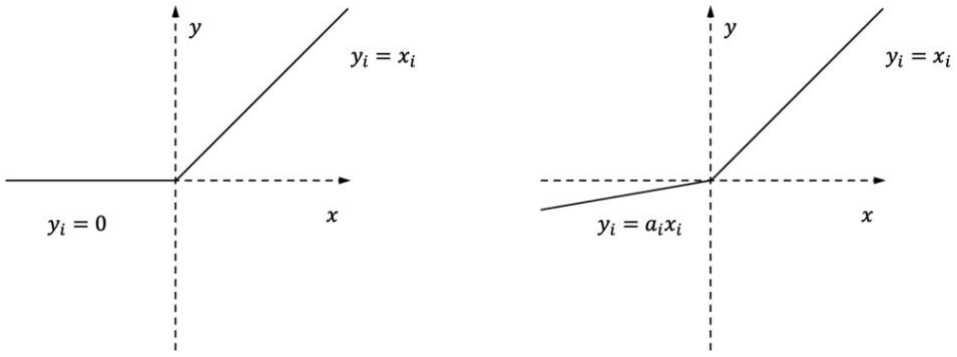


Figure 2.1 ReLU (left) and Leaky ReLU (right).

2.1.3 Pyramid Networks

When performing object detection, there may be multiple objects in an image and the objects may be large or small. Therefore, the object detection model must have the ability to detect objects of different sizes. In the feature map output by each layer of the actual convolutional neural network, the features detected by the convolutional layers at different depths vary. The

output of the feature map in relation to the shallow network undergoes fewer convolution operations and retains small sized detailed information such as object, colour, position, edge, etc. This is low-level and specific. As the depth of the network deepens, the output feature map has undergone more layers of convolution operations, including image information within a wider field of view. The information extracted by the feature map becomes abstract, such as the semantic information of the object (the category features of the object: pedestrian, cyclist, car, etc.). In response to this phenomenon, many models have begun to try and use different levels of feature maps for predictions. These are mainly following a mainstream method as shown in Figure 2.2.

- Featured Image Pyramid

Featured image pyramid method (Figure 2.2 a) is the most intuitive. Images of different scales are placed into the corresponding network for the detection of objects of different sizes. However, each level needs to be processed resulting in slow performance.

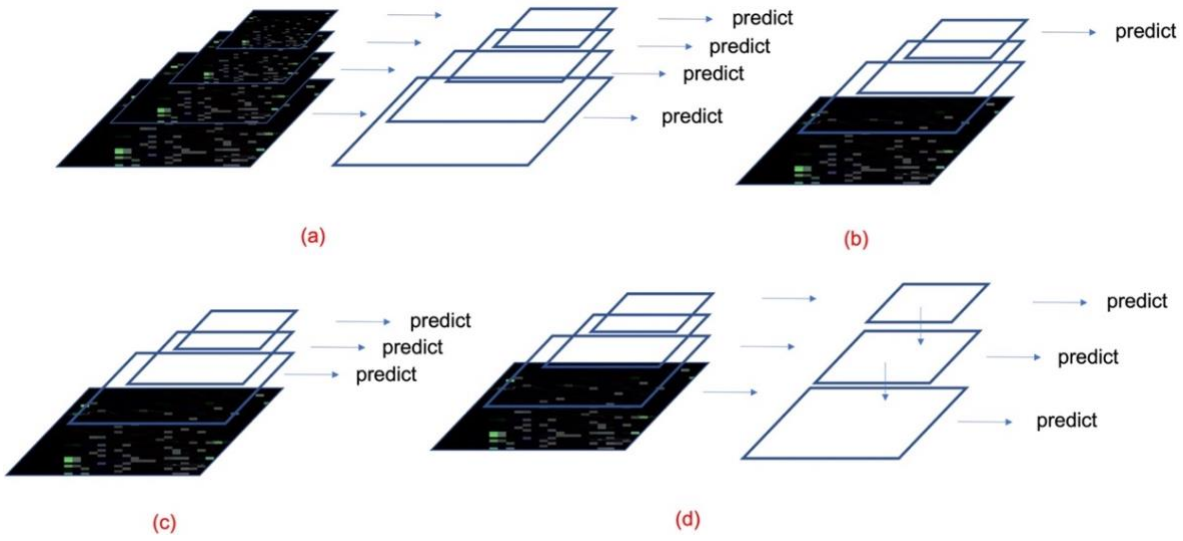


Figure 2.2 Pyramid network structures. (a) Featured image pyramid. (b) Single feature map. (c) Pyramidal feature hierarchy. (d) Feature pyramid network (adapted from [11]).

- Single Feature Map

Single feature map detection (Figure 2.2 b) is only performed in the last feature map stage. This structure cannot detect objects of different sizes [11], so it is not suitable for our experiments in the context of autonomous driving where vehicle or pedestrian instances are located at variable distance and therefore scales in the data.

- Pyramidal Feature Hierarchy

With pyramidal feature hierarchy (Figure 2.2 c), object detection is performed on feature maps of different depths. This is the structure used in the popular single shot detector (SSD) algorithm. The problem with this method is that the information obtained by each feature map only comes from the previous layer and the feature information of the subsequent layers cannot be obtained and used.

- Feature Pyramid Network

Feature pyramid network (FPN) [11] (Figure 2.2 d) is very similar to “pyramidal feature hierarchy” but one main difference is that the feature map of the current layer will be up sampled and used for the feature map of the future layer. This is a leapfrog design. As a result of this structure, the current feature map can obtain the information of the "future" layer. Due to this, the low-level features and the high-level features are organically integrated to improve the detection accuracy. This method is used in our model.

2.1.4 Optimization Methods for Deep Learning Models

Deep learning models are trained to learn features from input data and the process of learning consists of minimizing the model’s loss and improving the model’s performance. The process of minimizing (or maximizing) any mathematical expression is called optimization. With the development of deep learning algorithms, optimizers are used to provide a steady decrease of the loss.

The proposed model in this thesis uses Adam [12] for optimization. Adam and its foundation Stochastic Gradient Descent (SGD) [13], momentum [14] and RMSprop [15] will be explained in this section.

- Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is one of the most common gradient descent algorithms. The calculation process of the gradient descent method is to solve the minimum value along the direction of the gradient descent and the maximum value can also be solved along the direction of the gradient rise. Since SGD only needs to calculate the gradient of one sample for each parameter update, the training speed is very fast. When large sample sizes are used, it may only need a part of the samples to iterate to the optimal solution. Assume a sample (x^i, y^i) is selected to calculate its gradient, the parameter update equation is:

$$\theta_{i+1} = \theta_i - \alpha_t \cdot \nabla J_i(\theta_i,\ x^i,\ y^i) \qquad (2.4)$$

Where learning rate α_t determines the size of the steps that is taken to reach a local minimum. J_i is the object function that is minimized. Parameter θ_i is updated to θ_{i+1} by calculating the partial derivative of the object function on sample point (x^i, y^i) where parameter is θ_i, towards the overall optimization direction. The fluctuation of the gradient descent is very large, and it is easier to jump from one local optimum to another local optimum and the accuracy increases.

SGD also has shortcomings. For example, if the learning rate is too low, it will slow convergence while if the learning rate is too high, it will cause excessive fluctuations in convergence. Moreover, SGD requires that all parameters use the same learning rate, which does not apply to all models. SGD easily converges to a local optimum and in some cases may be trapped at the saddle point.

- Momentum

Momentum [14] is an optimization method that introduces the idea of momentum in physics to accelerate gradient descent. Momentum optimization methods have two algorithms: Momentum and Nesterov. When we roll a small ball down a mountain without resistance, its momentum will

become larger and larger, but if it encounters resistance, the speed will be reduced. The momentum optimization method uses this idea to make the gradient direction unchanged. In this dimension, the parameter updates become faster and when the gradient changes, the updated parameter becomes slower, which can speed up the convergence and reduce the turbulence.

The Momentum algorithm retains the direction of the updates previously made to the parameter to a certain extent. When the parameters are updated, a gradient of the current batch is used to fine-tune the final updated direction of its decrease. In short, it accelerates the current gradient by accumulating the previous momentum. When the gradient direction changes, momentum can reduce the parameter update speed, thereby reducing oscillation; when the gradient direction is the same, momentum can accelerate the parameter update, thereby accelerating convergence.

Suppose m_t represents the momentum at time t and μ represents the momentum factor, which is usually 0.9 or an approximate value. α is the learning rate and $\nabla_\theta J(\theta)$ represents partial derivative of the object function. If momentum is added based on SGD, the parameter update formula is as follows:

$$m_{t+1} = \mu \cdot m_t + \alpha \cdot \nabla_\theta J(\theta) \quad (2.5)$$

$$\theta_{t+1} = \theta_t - m_{t+1} \quad (2.6)$$

All in all, momentum can accelerate the convergence of SGD and suppress oscillations.

- Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) [12] can be seen as a combination of RMSprop [15] and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes the advantage of momentum by using moving averages of the gradient instead of gradient itself like SGD with momentum.

Algorithm 2.3 Adaptive moment estimation (Adam) [12].

1. Obtain the gradient with respect to the stochastic target at time step t: $g \leftarrow \nabla_\theta$
2. Update biased first moment estimate m_t: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$, where β_1 is the exponential decay rates of the first moment estimates
3. Update biased second raw moment estimate: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$, where β_2 is the exponential decay rates of the second moment estimates
4. Compute bias-corrected first moment estimate: $m^{\hat{t}} \leftarrow \frac{m_t}{1-\beta_1^t}$
5. Compute bias-corrected second moment estimate: $v_t \leftarrow \frac{v_t}{1-\beta_2^t}$
6. Update parameters $\theta_{t+1} = \theta_t - \frac{\delta}{\varepsilon+\sqrt{v^{\hat{t}}}} \cdot m^{\hat{t}}$, where ε represents a very small parameter (normally equals to 1e-08), in order to prevent the denominator from being 0 in the equation.

Momentum in Adam is directly incorporated into the estimation of the first moment of the gradient. When compared to RMSProp, where the second-order moment estimation may include a high bias at the start of training because of the lack of a correction factor – Adam, corrects all these problems. Adam includes a bias correction to fix the first-order momentum and second moments initialized from the origin.

Adam has several advantages:

- After the Adam gradient is biased and corrected, each iteration of the learning rate has a fixed range, which makes the parameters more stable.
- It combines the advantages of Adagrad, which is good at handling sparse gradients and RMSprop which is good at handling non-stationary targets.
- It calculates different adaptive learning rates for different parameters.
- It is also suitable for most non-convex optimization problems, which is suitable for large datasets and high-dimensional spaces.

2.1.5 Evaluation Metrics for Object Detection and Recognition Models

- Precision

Precision is defined by the amount of true positive among all detections:

$$precision = \frac{TP}{TP+FP} = \frac{TP}{all\ detections} \quad (2.7)$$

- Recall

Recall is defined as all the true positive over all ground truth:

$$recall = \frac{TP}{TP+FN} = \frac{TP}{all\ ground\ truths} \quad (2.8)$$

- AP

Take recall as the horizontal axis and precision as the vertical axis to get the Precision-Recall curve, also known as the P-R curve, which shows the trade-off between precision and recall under different thresholds. This curve helps to determine the best thresholds to maximize these two indicators, as the recall increases, the precision decreases. The reason is that when the number of positive samples increases (high recall), the precision of correct classification for each sample decreases (low precision). As the number of samples increases, the model is more likely to fail.

Since the curve is not complicated, we can use it to graphically determine the best values for precision and recall. The area enclosed by the P-R curve is Average-Precision, or AP for short, as shown below:

$$AP = \int_0^1 p(r)\, dr \quad (2.9)$$

Where $p(r)$ represents the P-R curve. Generally speaking, the better a classifier, the higher the AP.

- F1

When applying classification, it is necessary to comprehensively consider precision and recall. To determine a threshold, the *F-Measure*, a variable, is a common way to choose this threshold:

$$F_\beta = (1+\beta^2)\cdot\frac{precision\times recall}{\beta^2\cdot precision+recall} \qquad (2.10)$$

where β represents the weight of recall. If β is greater than 1, it means that the impact of recall is more important, and if β is less than 1, precision is more important. If β is equal to 1, it is equivalent to the harmonic averages of the two. Here is a commonly used indicator *F1* measure, which is an indicator of another evaluation model besides AP. This indicator is calculated according to:

$$F1 = \frac{2\cdot precision\times recall}{precision+recall} \qquad (2.11)$$

F1 measures the balance between precision and recall. When the value of F1 is high, it means that both precision and recall are high. A lower F1 score means a greater imbalance between precision and recall.

- mAP

mAP (mean average precision) is the average of AP over all classes considered.

2.1.6 Bilinear Interpolation

Bilinear interpolation is a method of estimating values between two points in a grid, based on a weighted average of the values at the four surrounding grid points [16]. This method is used in this thesis for preprocessing the RGB image.

Algorithm 2.4 shows the steps to perform bilinear interpolation:

Algorithm 2.4 Bilinear interpolation [16].

1. Define the four surrounding grid points and their values.
2. Calculate the relative position of the point to be estimated, with respect to the grid.
3. Calculate the weights for each of the four surrounding grid points, based on their distance from the point to be estimated.
4. Calculate the weighted average of the values at the four surrounding grid points, using the weights from step 3.

Bilinear interpolation is used to reduce the size of an image because it provides a more accurate representation of the image's original pixel values when compared to other interpolation methods like nearest-neighbor interpolation.

When reducing the size of an image, the number of pixels in the image is reduced, which means that some information from the original image is lost. Bilinear interpolation helps to mitigate this loss of information by taking into account the values of the surrounding pixels to determine the value of the new, reduced pixel.

In bilinear interpolation, the value of a new pixel is calculated as a weighted average of the surrounding original pixels, taking into account the distance of each original pixel from the new pixel. This allows for a smooth transition of pixel values as the size of the image is reduced, resulting in a reduced image that looks more similar to the original image.

Overall, bilinear interpolation is a useful tool for reducing the size of images while minimizing the loss of information and preserving the visual quality of the image as much as possible.

2.2 Overview of Object Detection and Recognition Algorithms

With the continuous improvement of traditional object detection and recognition algorithms, they gradually became saturated between 2010 and 2012. The development of new detection and recognition algorithms was slow and small progress was made mainly by building integrated systems and replacing variables [17]. Subsequently, a convolution neural network (CNN) [7] was proposed. CNN is based on the idea of local connectivity. For an image, local connectivity between pixels is stronger. This local connectivity ensures that the trained filter can have strong response to the local features, so that the neural network can extract the local features of the data. At the same time, the convolution kernel parameters used when computing neurons of the same depth are shared. The combination of these two features allows CNN to use a small number of parameters, but still have very good performance in image processing. OverFeat [18], R-CNN [20] and other deep learning methods based on CNN [7] are used in the field of object detection and recognition, which has brought tremendous development and is still used today.

Deep learning-based algorithms can be broken down into two separate categories, as shown in Figure 2.3. The first category takes classification as the main frame and focuses on accuracy. This can also be called “two stage detection”. For the second category, regression is the mainframe. It follows a “complete in one step” approach, thus can be defined as “single stage detection”.

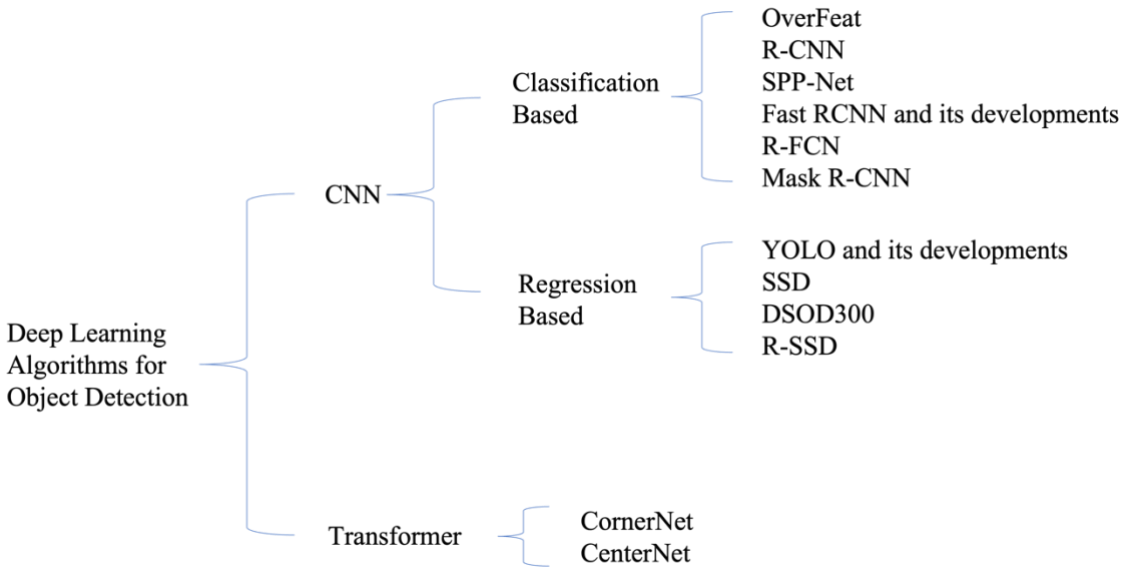


Figure 2.3 Deep learning algorithms for object detection.

Table 2.2 is a summary of the advantages and disadvantages popular deep learning models on object detection and recognition, which will be explained in detail in Section 2.2.1, 2.2.2 and 2.2.3.

Table 2.2 Summary of deep learning models.

Model	Advantage	Disadvantage
OverFeat [18]	The earliest use of CNN for feature extraction.	Image sliding window requires large memory and long training time.
R-CNN [20]	Using region proposal, CNN feature extraction and SVM classification. The performance is significantly improved over traditional algorithms.	Feature extraction for each region proposals, requires large memory and long training time.

SPP-Net [21]	Fast, SPP layer avoids the normalization of candidate regions.	Require large memory.
Fast R-CNN [22]	Training and testing at the same time.	Selective search generate regional proposal lowers the speed of the model.
Faster R-CNN [23]	End-to-end training and testing.	The model is complex and has poor results on small object detection.
R-FCN [24]	Higher accuracy.	Model is complex, too much computation.
Mask R-CNN [27]	Accurate instance segmentation, higher detection accuracy.	High cost of instance segmentation.
YOLO [28]	Simple network, high detection speed.	Low accuracy, poor results on small objects and multi-object detection.
YOLOv2 [29]	Allow users to choose between accuracy and speed.	Need pre-processing needed.
SSD [32]	Simple network, high detection accuracy.	Poor results on small objects.
DSOD300 [33]	No pre-processing required.	Low speed.
R-SSD [34]	Good accuracy for small objects.	Complicated model, low speed.
CornerNet [35]	Avoid the problem of a large number of anchors.	Accuracy is not as high as R-CNN.
CenterNet [36]	The structure of the model is relatively simple.	Not good at detecting overlapping objects.

Table 2.3 shows a comparison between the models summarized in Table 2.2 based on experimental testing results reported in the literature on commonly used datasets.

Table 2.3 Popular models, their backbones and reported testing results on ILSVRC 2013 and VOC 2007.

Model	Backbone	ILSVRC 2013 (%)	VOC2007 (%)	Speed (FPS)
OverFeat [18]	AlexNet [19]	24.3	—	—
R-CNN [20]	AlexNet	31.4	58.5	0.02
SPP-Net [21]	ZF-5	—	59.2	0.25
Fast R-CNN [22]	VGG 16 [26]	—	70.0	0.5
Faster R-CNN [23]	VGG 16	—	73.2	5
R-FCN [24]	ResNet-101	—	83.6	6
Mask R-CNN [27]	ResNetXt	—	63.3	5
YOLO [28]	DarkNet	—	63.4	45
YOLOv2 [29]	DarkNet-19	—	75.8	67
SSD [32]	VGG 16	—	73.2	59
DSOD300 [33]	DS/64	—	77.7	17.4
R-SSD [34]	VGG 16	—	80.8	16.6

2.2.1 Two Stages Models

- OverFeat [18]

OverFeat is one of the first algorithms to apply deep learning methodologies to object detection. OverFeat does not use the region proposal, but its idea is improved by the subsequent R-CNN

[20] series. The algorithm uses a multi-scale sliding window combined with AlexNet [19] as backbone to extract image features to complete the detection. The average accuracy rate on the ILSVRC 2013 dataset is 24.3% and the detection effect significantly improved after it is compared to the traditional algorithms.

- R-CNN [20]

Soon after OverFeat was published, Girshick et al. proposed Region Based Convolutional Neural Networks (R-CNN) [20], which made a huge breakthrough in object detection field. The R-CNN algorithm can be summarized as selecting a set of region proposals through selective search. Each region proposal can be rescaled into a uniform size and continue with using it as the input for a convolutional neural network. This allows the model to extract features and to use an SVM (support vector machine) linear classification to predict the objects in each region proposal to recognize the predicted score. The test on VOC2007 dataset shows that the performance of R-CNN far exceeds other algorithms at the time, the mAP reaches 0.58.

- SPP-Net [21]

R-CNN has a disadvantage that the input image needs to be cropped to a fixed size which may reduce the recognition accuracy. To solve this problem, He et al. proposed SPP-Net. In this paper, a spatial pyramid pooling layer [21] is added on top of the last convolutional layer, which removes the fixed-size constraint of the network.

- Fast R-CNN [22] and Faster R-CNN [23]

In 2015, Girshick proposed Fast R-CNN based on R-CNN [22], so that it can perform detector training and bounding box (B-Box) regression at the same time without changing the network configuration. The test on the VOC2007 dataset shows that, Fast R-CNN's mAP reaches 70.0%, far exceeding R-CNN.

Inspired by SPP-Net, Fast R-CNN proposed region of interest (ROI) pooling, which further improved the detection speed.

Ren et al. proposed Faster R-CNN in the same year. Faster R-CNN's speed is significantly improved, and it can detect almost in real time. Tests on the VOC2007 dataset show that its mAP reaches 73.2%. This means that the speed increased without lowering the detection accuracy.

- R-FCN [24]

Object detection and recognition includes two problems: the classification problem and the detection and localization problem. The former is translation invariant, and the latter is translation sensitive. Region-based Fully Convolutional Networks (R-FCN) uses the full convolutional network ResNet [25] instead of VGG16 [26] to improve the effect of feature extraction and classification. A full convolutional network is not suitable for translation sensitive application because the algorithm uses a specific convolutional layer to generate position sensitive information. This also makes it possible for the ROI pooling layer to not connect to the subsequent spatial position information in the distribution map, thus avoiding the extra computation.

The accuracy of R-FCN reaches 83% and its speed reaches 6 FPS, which is faster than Faster-RCNN. However, R-FCN needs to generate a channel number that linearly increases with the number of categories when obtaining the score map. While this process improves the accuracy of object detection, it slows down the detection speed.

- Mask R-CNN [27]

Mask R-CNN is an improved algorithm based on Faster R-CNN, which increases the focus on instance segmentation. In addition to classification and localization regression, the algorithm adds parallel branches on instance segmentation.

Instance segmentation requires the accuracy of instance localization to reach the pixel level and Faster-R-CNN introduces errors in the equal scaling process of the ROI Pooling layer, resulting in rough spatial quantization and inaccurate localization. Mask R-CNN proposes bilinear difference RoIAlign (Region of Interest Align) to obtain more accurate pixel information, which increases the mask accuracy rate by 10% to 50%. Mask R-CNN uses the ResNetXt as backbone and the detection speed on the COCO dataset is 5 FPS.

• Summary

From the algorithms previously examined, starting from R-CNN, researchers have focused on the problems of object detection in classification. Using the idea of "region proposal + CNN feature + SVM" as shown in Figure 2.4, the CNN network has greatly improved on accuracy of detection. Following this, SPP-Net, Fast-RCNN, Faster-RCNN all take note of this idea and improve on detection efficiency. However, Faster-RCNN can only reach 5 FPS – this is insufficient in terms of real-time performance. R-FCN has also improved but the effect remains unsatisfactory. Researchers have put forward another idea that directly converts the object detection to regression and uses an image to get both a bounding box and a category classification.

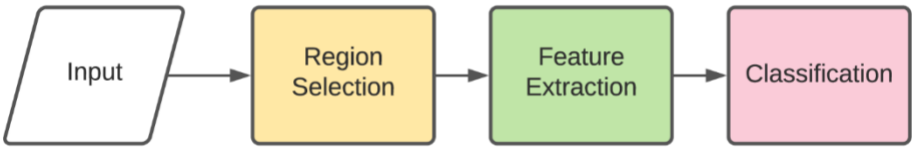


Figure 2.4 Commonly used two-stage algorithm architecture.

2.2.2 Single Stage Models

• YOLO [28] and its developments

From R-CNN to Faster-RCNN, object detection always follows the idea of "region proposal + classification". Training two models will inevitably lead to an increase in parameters and training volume, which will affect the speed of training and detection. As an alternative, YOLO proposed a "single-stage" approach.

YOLO (You Only Look Once) is a classic one-stage object recognition model. YOLO divides input images into S×S grid cells. Each grid cell is only responsible for detecting the target whose centre falls on it. Each grid cell needs to predict two scales of bounding boxes (B-Boxes) and

classification information. To complete the detection, it must predict the bounding box, target confidence and classification probability of the target contained in all regions at one time.

YOLO replaces the region proposal with a multi-scale area centred on the grid cell and discards some accuracy in exchange for a substantial increase in detection speed. The detection speed can reach 45 FPS, which is sufficient to meet real-time requirements; the detection accuracy is around 60%, which is slightly worse than Faster R-CNN's 70%.

YOLO greatly improves the detection speed but still has the following problems:

- Although each grid cell predicts 2 B-Boxes, in the end, only the B-Box with the highest intersection over union (IOU) is selected as the object detection output. That is, each grid cell can predict only one object. Therefore, when small objects that occupy a small proportion of the screen, and one grid cell contains multiple small objects, only one of them can be detected.

- YOLO's localization of the B-Boxes is slightly less accurate compared to using a sliding window method, thus its accuracy is not as good as Fast-RCNN.

YOLOv2 (also known as YOLO9000) [29] is the second version of the YOLO network. It improves the network structure of YOLO by adding a batch normalization layer and using residual network architecture. YOLOv2 first trains a high-resolution classification network (448x448) and then fine tune the resulting network on detection. By increasing the resolution of the input, the mAP has improved by 4%. YOLO uses a single grid cell to be spliced into a fully connected layer to complete the prediction of the frame, resulting in the loss of spatial information and inaccurate localization. Redmon, the author of both YOLO and YOLOv2 has optimized and improved in this version:

- Drawing on the anchor idea in Faster R-CNN, the author found in reality that using anchor-based detection instead of sliding window does not improve the accuracy. In the experiment, the author performed statistical analysis on the dataset of Pascal VOC and COCO (cluster analysis). The choice of anchor size has a significant impact on the performance of the model.

- The author found that the model convergence was unstable when using anchor boxes, especially in the early iterations. Most of the instability occurs in the optimization of the (x, y) coordinates of the prediction box. YOLO predicts the coordinates position relative to the grid cell. With the use of this logistic function, it normalizes the ground truth between 0 and 1. After the coordinate value has been normalized, the model optimization will be more stable.

The main difference between YOLOv3 [1] and YOLOv2 is the backbone. The backbone part of YOLOv3 evolves from Darknet-19 in the YOLOv2 to Darknet-53, which deepened the number of network layers and introduced the cross-layer sum operation from ResNet. The processing speed of Darknet-53 is 78 FPS, which is much slower than Darknet-19, but much faster than ResNet with the same precision. YOLOv3 still maintains high performance.

Same as the former YOLO algorithms, the loss function used in YOLOv3 is MSE (mean square error) and BCE (binary cross entropy) calculated based on the centre point coordinates and width and height information of the predicted bounding box (B-Box) and the ground truth.

Based on YOLOv3, YOLOv4 [30] adds a spatial pyramid pooling module, which is more effective in increasing the reception range of backbone features than simply using k*k maximum pooling, and significantly separates the most important contextual features. In terms of loss function, YOLOv5 [31] uses a combination of GIoU loss and weighted non maximum suppression, and its detection results are equally good as YOLOv4.

- SSD [32]

SSD is a single-shot detector, which improved YOLO to achieve a precision equivalent to that of the two-stage method (such as Faster R-CNN), while maintaining a fast speed. Compared with YOLO, SSD uses CNN to detect directly, instead of doing detection after the fully connected layer like YOLO. The main improvements of SSD are:

- Detection based on multi-scale feature images: Prediction on multi-scale convolutional feature maps to detect targets of different sizes, which improves the detection accuracy of small target objects to a certain extent.

- Drawing on the idea of anchor boxes in Faster R-CNN, sampling candidate regions on feature maps of different scales, which improves the recall rate of detection and the detection effect of small targets to a certain extent.

The mAP of SSD300 can reach 73%, which is basically the same as Faster R-CNN and the detection speed reaches 59 FPS, which is almost 6 times faster than Faster R-CNN. However, SSD has the following problems:

- The small target object corresponds to a small area in the feature map and cannot be fully trained. Therefore, the detection result of the SSD on the small target objects is still not ideal.

- Since there is no region proposal, the region regression is difficult, and it might affect the accuracy.

- The feature maps from different layers of the SSD are used as independent inputs to the classification network, resulting in the same object being detected by boxes of different sizes at the same time, hence exist duplicated computation.

• DSOD 300 [33]

Aimed at the problem of pre-training a model, the DSOD (Deeply Supervised Object Detector) [33] algorithm proposes a method of training a network from scratch. DSOD achieves an effect comparable to the fine-tune models such as RCNN. DSOD is based on the SSD (Single-Shot Detector) [32] model and introduces the DenseNet idea in the feature fusion part. The use of DenseNet greatly reduces the number of parameters. The mAP is nearly 75% which is equivalent to SSD300; however, the detection speed is only about 15 FPS.

• R-SSD [34]

Based on SSD, the R-SSD (Rainbow Single-Shot Detector) algorithm increases the association of feature maps between different layers to avoid the problem of repeating frames for the same object; at the same time, it increases the number of feature maps in the feature pyramid to improve the detection effect of small objects. The algorithm mAP is 80.8%, which is slightly

higher than that of SSD. However, the increase of the feature map leads to an increase in the amount of calculation and a decrease in the detection speed, which is less than 20 FPS.

2.2.3 Transformer Models

Bengio and his team proposed the attention mechanism [37] in 2014, and Ashish et al., based on this, proposed the transformer [38]. The transformer approach discards the traditional CNN and RNN, and the entire network structure is completely composed of the attention mechanism. More precisely, transformer consists of and only of self-attention and feed forward neural network. A transformer-based trainable neural network can be built in the form of stacking transformers and has been widely used in various fields of deep learning in recent years, including in computer vision.

Although the transformer architecture is not used in this thesis, considering it is a popular object detection and recognition method, this section introduces two examples of that architecture.

- CornerNet [35]

CornerNet is a deep learning-based object detection method proposed in 2018 [35]. It is designed to detect objects in an image using only corner keypoints, which is a novel approach to object detection in computer vision. Unlike traditional object detection methods that use bounding boxes or segmentation masks, CornerNet detects objects by identifying keypoints at the corners of an object's bounding box. These keypoints are then grouped into objects using a Corner Pooling layer, which maps the keypoints to the object bounding boxes.

The process of detecting corner keypoints involves training a deep neural network called Hourglass network. The Hourglass network is trained to predict the locations of the keypoints in an image. It takes an image as input and processes it through multiple convolutional and deconvolutional layers, producing a heatmap that represents the likelihood of each pixel being a corner keypoint. The keypoints are then extracted from the heatmap and used to generate the bounding boxes for the objects in the image.

CornerNet has several advantages over traditional object detection methods. It is more accurate and efficient, as it requires fewer parameters and computations. Additionally, it can handle objects with irregular shapes better, as it does not rely on predefined shapes or contours. This makes CornerNet a promising method for real-time object detection in a variety of applications.

- CenterNet [36]

CenterNet [36] is a transformer object detection method based on the idea of keypoint detection, similar to CornerNet [35]. However, instead of detecting corner keypoints, CenterNet detects the center of the object's bounding box. The center keypoint is then used to generate the bounding boxes for the objects in the image.

Like CornerNet, CenterNet uses Hourglass network to predict the center keypoints. The Hourglass network takes an image as input and processes it through multiple convolutional and deconvolutional layers, producing a heatmap that represents the likelihood of each pixel being the center of an object's bounding box. The center keypoints are then extracted from the heatmap and used to generate the bounding boxes for the objects in the image.

CenterNet requires fewer parameters and computations than CornerNet which makes it more efficient.

- Summary

Object detection based on deep learning has greatly improved the detection accuracy and detection speed compared with traditional methods, but still faces these problems:

  - For a small amount of training data, the current framework may not be able to get good results. Most of the current algorithms use transfer learning, that is, the existing large dataset is concentrated for training and then the trained "semi-finished products" are used for fine-tuning operations. If the target data is not in an evenly distributed dataset, the training effect depends on the degree of correlation between the target and the large dataset. Although the DSOD algorithm designed a network trained from scratch and achieved good results, its detection speed still needs to be improved.

- The use of GPU has improved the computing power of the computer, but many of the detection and recognition models are still too large. Explorations on how to simplify and reuse calculations while ensuring accuracy as much as possible may be an innovative point.

- Either R-CNN series or SSD and other single stage algorithms has not been able to obtain satisfactory results on small object detection problems. In order to improve the detection speed, the input image of the feature pyramid is usually downscaled to reduce the computational load of the processing. This will inevitably lead to insufficient training of small targets on the feature map. For example, R-SSD increases the number of feature maps but loses on detection speed.

2.3 3D Object Detectors using LiDAR Point Cloud

This section overviews neural network models using LiDAR point cloud as the training and testing input.

Compared with 2D, 3D object detection and recognition requires not only the traditional RGB or grayscale image dataset, but also depth information, that is, the position coordinates of each pixel in space. It is precisely because of this that 3D tasks usually require larger and more complex training and testing datasets, which lead to higher requirements in data processing and computing capabilities. At the same time, for outdoor 3D object detection and recognition tasks, as found in autonomous vehicles, the neural network model is as important as the dataset requirements. Generally, RGB-D cameras and stereo cameras cannot meet the requirements for outdoor depth image acquisition due to their technical limitations [39]. Therefore, for such tasks, the LiDAR point cloud is often chosen to provide input data to the detectors.

Based on convolutional neural network models such as CNN, RCNN, YOLO, SPP Net, SSD introduced in the previous section, a variety of neural network models are proposed in the literature to enhance performance in 3D vision tasks. In general, due to the complexity of LiDAR based data itself, compared to common 2D RGB images, it is more complex to design a model

for 3D object detection and recognition than for similar 2D tasks, but that does not stop researchers from learning from cutting-edge 2D models. Section 2.3 divides these models into two-stage methods and single-stage methods according to their corresponding algorithms and backbone structure of models for 2D object detection and recognition. Practically, based on the input data of the model, they can also be divided into discretization-based methods on camera/range view object detection methods, point/feature-based methods, BEV-based methods, and fusion feature-based methods. Table 2.4 summarizes the models that are surveyed in Sections 2.3.1, 2.3.2 and 2.3.3.

Table 2.4 LiDAR point cloud-based 3D object detection and recognition networks.

Method		Model
Two stages	Multi-view methods	MV3D-Net [40], AVOD [41], ContFuse [42], MMF [43], SCANet [44], RT3D [45]
	Segmentation-based methods	IPOD [46], Point-RCNN [47], PointRGCN [48], PointPainting [49], STD [50]
	Frustum-based methods	F-PointNet [51], SIFRNet [52], PointFusion [53], RoarNet [54], F-ConvNet [55], Patch Refinement [56]
	Point-based methods	Unified frame [57], Fast Point R-CNN [58], PV-RCNN [59], VoteNet [60], RGNet [61], ImVoteNet [62], Part-A^2 Net [63]
Single stage	BEV-based method	PIXOR [64], HDNET [65], BirdNet [66]
	Discretization-based methods	VeloFCN [67], 3D FCN [68], Vote3Deep [69], 3DBN [70], VoxelNet [71], SECOND [72], MVX-Net [73], PointPillars [74], SA-SSD [75]
	Point-based methods	3D-SSD [76]
Transformer	Discretization-based methods	VoTr-SSD, VoTr-TSD [77]

2.3.1 Two-Stage LiDAR Point Cloud-Based 3D Object Detectors

Inspired by two-stage 2D object detection and recognition algorithms enumerated in Section 2.2.1, the researchers modified and applied them to 3D object detection and recognition tasks. Using selective search to extract region proposals, combined with classification and regression methods, shows good results on 3D object detection. It is also popular to simplify the training process by using pre-trained parameters from 2D object detection and recognition models.

- MV3D-Net [40]

MV3D-Net is a paper published in 2017. It combines both RGB images and LiDAR point cloud as input. Unlike other voxel-based methods, it uses the BEV (bird's-eye view) and the front view generated from the point cloud. Using both the BEV and front view reduces the number of calculations performed thus reducing the information loss. Then applying 2D bounding boxes to generate 3D candidate regions, the method merges the features and candidate regions to output the final object detection frame. The idea from this paper of using both the point cloud and the RGB image to get different viewpoints is implemented in a different way in this thesis. One characteristic worth mentioning is that in MV3D-Net, the point cloud is divided equally into M slices to generate M height maps in the bird view part of the input, while in this thesis, a single cumulated height map is considered.

- AVOD [41]

AVOD and MV3D-Net hold many similarities. The main difference is that AVOD is an enhanced version of MV3D-Net by adding geometric constraints to the coding of the 3D B-Box (bounding box). AVOD uses a bottom surface and height to constrain the geometric shape of the 3D B-Box, which requires it to be a cuboid, while MV3D-Net only gives 8 vertices without any geometric constraints. In addition, 8 vertices in MV3D-Net require a 24-dimensional (3×8) vector representation, while AVOD only requires a 10-dimensional $(2 \times 4 + 1 + 1)$ vector, which achieves a good encoding dimensionality reduction.

- ContFuse [42]

The ContFuse network uses continuous convolution to fuse point cloud and RGB image features. The fusion process is completed under BEV. For each pixel on the BEV, the method first finds its K nearest points in the point cloud data and then maps these points in the 3D space to the image space to obtain the image characteristics of each point. The geometric feature of each point is considered as the XY offset from that point to the corresponding BEV pixel. The algorithm combines the image feature and the geometric feature as a point feature and then performs a weighted summation (the weight depends on the XY offset) according to the continuous convolution method to obtain the feature value at the corresponding BEV pixel. Similar processing is performed on each pixel of the BEV and a BEV feature map is obtained. This completes the conversion of image features to BEV views, which can then be easily merged with the BEV features from the point cloud. In ContFuse, the above-mentioned feature fusion is performed at multiple spatial resolutions to improve the detection ability of objects of different sizes.

- MMF [43]

Multi-task multi-sensor fusion network, MMF for short, was published by the same author as ContFuse [42]. The advantage of this work is that the multi-scale image features are merged into BEV and a variety of different methods are combined to improve the average accuracy of 3D detection. First, the point cloud is processed into a sparse depth map and is combined with RGB images to generate RGB-Depth (RGB-D) images. A CNN generates dense depth information to complement the LiDAR pseudo point cloud; secondly, the original point cloud maps road surface estimation to represent the BEV of point cloud data and uses it as input of the convolutional network. Then, the original point cloud and the depth-completed pseudo point cloud deep fusion (here is a pointwise fusion similar to ContFuse) is sent to the convolutional network and then performs a ROI-wise fusion similar to AVOD [41] is performed to initially estimate the 3D bounding box and project it to the RGB-D network at the same time.

- SCANet [44]

This paper introduces a spatial channel attention (SCA) component to apply multi-scale context information, which can obtain global and multi-scale information in the scene and identify useful features. They also propose an Extension-Spatial-Unsample (ESU) component to obtain high-dimensional features with rich spatial information by combining multi-scale low-dimensional features, so that reliable 3D proposal boxes can be generated.

- RT3D [45]

RT3D is famous for its “only takes 9 milliseconds per frame". The network is based on MV3D-Net [40] and uses pre-ROI (region of interest) pooling convolution to improve the efficiency of MV3D-Net, which speeds up the detection speed and reaches real-time. Specifically, they move most of the convolution operations before the ROI pooling module. Therefore, only one ROI pooling operation is performed for all objects. The overall speed is 5 times faster than MV3D-Net.

MV3D-Net [40], AVOD [41], ContFuse [42], MMF [43], SCANet [44], and RT3D [45] demonstrate that data fusion helps the object detection and recognition model to obtain information from different viewpoints, which inspired this thesis to use both the LiDAR point cloud and the RGB image as inputs.

- IPOD [46]

The authors of IPOD discover that some LiDAR based 3D detectors rely too much on the 2D detection result, hence cannot perform well under occlusion. To solve the occlusion problem, IPOD [46] proposes to use a 2D semantic segmentation to replace 2D object detection. First, the result of semantic segmentation on the image is used to remove the background points in the point cloud, which is done by projecting the point cloud into the 2D image space. Next, a candidate object frame is generated at each front spot in the scene. Finally, approximately 500 candidate frames are reserved for each frame of the point cloud. At the same time, a PointNet++ [78] grid is used for point feature extraction. With the candidate frame and point features, the last step uses a small-scale PointNet++ to predict the category and accurate object frame. IPOD

generates dense candidate object frames based on semantic segmentation, so the effect is better in scenes with multiple objects and mutual occlusion.

- Point-RCNN [47]

The Point-RCNN method combines point cloud processing with Faster RCNN, the pioneering work in the field of 2D object detection. First, PointNet++ [78] is used to extract point features. Point features are used for foreground segmentation to distinguish points on objects from background points. At the same time, each foreground pixel point will also output a 3D candidate B-Box. The next step is to perform further feature extraction on the points in the candidate B-Box, output the object category to which the B-Box belongs and refine its position and size. Point-RCNN only generates candidates on the foreground, which avoids the huge amount of calculation caused by generating dense candidate frames in 3D space. Nevertheless, as a two-stage detector, plus the large amount of calculation of PointNet++ itself, the operating efficiency of Point-RCNN is still low, only about 13 FPS (frames per second). Point-RCNN was later extended to Part-A^2 Net [63], with certain improvements in speed and accuracy.

- PointRGCN [48]

This model is a pioneering work on the use of graph convolutional networks (GCN) for three-dimensional object detection. It introduces two modules to use graph convolution to refine the object proposal box. The first module R-GCN uses all the points contained in the proposal box to extract the feature aggregation of each proposal box. The second module, C-GCN, merges the information of each frame from all the proposed frames and returns to the precise object frame by using the context.

- PointPainting [49]

PointPainting projects the point cloud into the result of image semantic segmentation, which is similar to the method in IPOD [46]. However, PointPainting does not use the result of semantic segmentation to separate the foreground pixel point, but directly attaches the information of semantic segmentation to the point cloud. The advantage of this is that the data after fusion is still a point cloud (but with richer semantic information), which can be processed by any point

cloud object detection network, such as Point-RCNN [47], VoxelNet [71], PointPillar [74] and so on.

What is attached to the point cloud in PointPainting is the semantic information of the 2D image, which is already highly abstracted information, while the original image features are discarded. From the perspective of fusion, the fusion of underlying features can retain information to a greater extent and use the complementarity between different features. Theoretically speaking, it is more likely to improve the effect of fusion.

- STD [50]

The sparse-to-dense detector has achieved many innovations. Like the models explained earlier in this section, the STD network is still divided into two-stage. The first stage is a bottom-up proposal generation network, which uses the original point cloud as input and generates an accurate proposal by seeding a new spherical anchor for each point. Compared with the previous work, this algorithm has a higher computational speed and a smaller computational load. Then, using a pooling layer to convert the internal point features of the proposal features from sparse expressions to compact representations to generate proposal features saves more calculation time. In the second stage of box prediction, this paper implements a parallel intersection and merge (IoU) branch to improve positioning accuracy, thereby further improving performance. Experiments were conducted on the KITTI [2] dataset and STD was evaluated from two aspects: three-dimensional object and bird's-eye view (BEV) detection. The method in this paper is superior to other technologies to a large extent and the recognition speed reaches over 10 FPS.

IPOD [46], Point-RCNN [47], PointRGCN [48], PointPainting [49], and STD [50] show that incorporating segmentation into 3D object detection helps increase the accuracy. However, the significant computational resources required by this method make it unsuitable for the autonomous driving challenges addressed in this thesis.

- F-PointNet [51]

F-PointNet generates 2D object bounding boxes from image data and then projects these bounding boxes into 3D space. Each 2D bounding box corresponds to a "frustum" in the 3D space and merges all the points that fall into the frustum as the feature of the bounding box.

Before generating the bounding box, it needs to go through a T-Net layer [51] and its function is to generate a translation. The reason for this step is that the centre of the object obtained in the previous step is not completely accurate, so in order to estimate the bounding box more accurately, the centre of mass of the object is further adjusted here. Given that this method is based on frustum, the disadvantage is that only one object to be detected can be processed in each frustum, which cannot meet the requirements for crowded scenes and small targets (such as pedestrians).

- SIFRNet [52]

SIFRNet proposes a Point-SENet component to predict a set of scale factors, which can be further used to adaptively extract useful features as well as suppress features with less information. In addition, they also use PointSIFT [52] in the network to obtain the orientation information of the point cloud. This method is very robust to the point cloud shape scaling.

- PointFusion [53]

This paper uses the 2D image area and the corresponding frustum points to accurately return to the 3D frame. To fuse the image features and the global features of the point cloud, they propose a global fusion network to directly return to the centre of the frame. In addition, they also propose a dense fusion network to predict the point-by-point offset from each centre point.

- RoarNet [54]

This network first estimates the 2D bounding box and 3D pose of an object from a 2D image and then extracts multiple geometrically feasible object candidate frames. These 3D candidate frames are sent as a frame regression network to predict the accurate 3D object frame.

- F-ConvNet [54]

Frustum convnet (F-ConvNet) generates a sequence of frustum along the frustum axis for each 2D area and applies PointNet to each frustum to extract features. Re-frustum-level features are considered to generate a 2D feature map and then it inputs the feature map into the full convolutional network for 3D bounding box estimation. This method achieves the most

advanced performance among the 2D image-based methods and ranks first in the official KITTI benchmarks.

- Patch Refinement [56]

Patch refinement uses its model to first obtain a preliminary detection result on the BEV map and then extract a small subset of points (also called patch) based on the BEV prediction. Then a local fine-tuning network is used to learn the local features of the patch to predict the high-precision three-dimensional bounding box.

- Unified frame [57]

This model comes from 3D IoU loss paper [57] and it is unified for both 2D and 3D. Due to the success of the axis-aligned IoU in image object detection, this paper integrates the IoU of two 3D rotating bounding boxes into several latest technical detectors such as PointR-CNN [47], SECOND [72] and PointPillars [74], in order to achieve performance improvement.

- Fast Point R-CNN [58]

Fast Point R-CNN proposes a two-stage network architecture that uses point cloud and voxel representation at the same time. First, the point cloud is voxelized and sent to a 3D R-CNN based network to generate initial detection results. Then, the boundary box is further fine-tuned using the characteristics of the internal points of the initial prediction. Although this design is relatively simple in concept, it achieves a performance that is competitive with Point-RCNN [47] while maintaining a rate of 16.7FPS.

- PV-RCNN [59]

After publishing Point-RCNN [47], Shi et al. then proposed PV-RCNN in 2020. The PV-RCNN model integrates multi-scale 3D voxel CNN features captured using PointNet with the voxel set abstraction layer proposed in this paper. And through the features of key points, the prediction frame and score generated by the one-stage detection are effectively optimized. Excellent results have been achieved on the KITTI and Waymo Open datasets. Compared with previous state-of-

the-art algorithms, the voxel-to-point and keypoint-to-grid structures proposed in this paper effectively and significantly improve the performance of 3D object detection.

- VoteNet [60]

VoteNet is inspired by a 2D object detector based on Hough voting and was proposed to directly vote from the point cloud to generate a virtual centre point and integrate voting features to generate a set of high-quality 3D object proposal boxes. This approach performs better than the ones only using geometric information. The disadvantage is that the predicted virtual centre point is not stable enough for objects that are partially occluded.

- RGNet [61]

RGNet (relation graph network) introduces a branch of direction vector to improve prediction accuracy of the virtual centre point and the three-dimensional candidate frame. In addition, the paper also establishes a 3D object-to-object relationship graph between candidate frames to emphasize features useful for accurate object detection.

- ImVoteNet [62]

ImVoteNet uses RGB images to provide geometric, semantic and texture information for the 3D voting process. First, it uses 2D detectors and PointNet++ [78] for RGB images and point clouds to obtain 2D boxes and seed points; second, it upgrades the geometric, semantic and pixel texture information extracted from RGB to 3D and attaches them to the seed points; finally, it generates the centre of the 3D object and then estimates the 3D box. By using a multi-model training method, ImVoteNet greatly improves the 3D detection performance of dealing with sparse or unfriendly distributed point clouds.

- Part-A^2 Net [63]

The Part-A^2 network consists of a partial perception stage and a partial aggregation stage. A U-Net-like network with sparse convolution and sparse deconvolution is used to learn the point-by-point features for prediction and roughly generates the position of the inner part of the object. In

the partial aggregation stage, ROI-aware pooling is used to aggregate the predicted partial positions for the scoring and position refinement of the border.

2.3.2 Single-Stage LiDAR Point Cloud-Based 3D Object Detectors

Not only two-stage algorithms are widely used in 3D LiDAR based detectors, but single-stage algorithms have also become popular. The single-stage algorithm introduced in YOLO [28] for 2D images offers advantages over two-stage algorithms in terms of computational complexity, while the detection accuracy can also meet the needs of real-world scenarios. As such, the concept was successfully transposed to 3D object detectors.

- PIXOR [64]

PIXOR discretizes the point cloud of a scene with equally spaced units and encodes the reflectivity in a similar manner to obtain a regular representation. Then, a full convolutional network (FCN) is used to estimate the position and heading angle of the target. This method is superior to most one-stage methods and its detection speed can reach 28.6 FPS.

- HDNET [65]

This model uses the geometric and semantic prior information provided by high-definition (HD) maps to improve the robustness and detection performance of the PIXOR [64] algorithm. Specifically, the authors obtain the coordinates of the ground point from the high-precision map and then replace the absolute distance in the BEV representation with the distance relative to the ground to remedy the translation variance caused by the road slope. In addition, they concatenate a binary road mask with the BEV representation along the channel dimension to focus on moving objects. Since high-definition maps are not available everywhere, they also propose an online map prediction module to estimate the map prior information of a single LiDAR point cloud. This map-aware method performs better than the PIXOR [64] methods on the TOR4D [78] and KITTI datasets [2]. But its generalization performance to point clouds of different densities is poor.

- BirdNet [66]

Given HDNET [65] poor performance on generalization, BirdNet was introduced to alleviate the problem. This model proposes a normalized map to consider the differences between different LiDAR sensors. The normalized map is a 2D grid with the same resolution as the BEV map, which encodes the maximum number of points contained in each cell. The results show that the normalized map significantly improves the generalization ability of BEV-based detectors.

Compared to two-stage methods, PIXOR [64], HDNET [65], and BirdNet [66] show that the single-stage detectors are more efficient. For time critical tasks such as autonomous driving, using single-stage structure brings more benefit to the detector.

- VeloFCN [67]

Vehicle detection from 3D LiDAR using a fully convolutional network (VeloFCN) presents the data in a 2D point map and uses a single 2D end-to-end fully convolutional network to simultaneously predict the object confidence and bounding box. By carefully coding the design for the bounding box, the complete 3D bounding box can be predicted even using a 2D convolutional network.

- 3D FCN [68]

3D fully convolutional network (3D FCN) converts a 2D full convolutional network FCN into a model that uses 3D point cloud as input data to achieve 3D object detection. It uses a square grid to discretize the point cloud. Discrete data can be represented by 4D arrays with length, width, height and channel size. For the simplest case, only one channel of value [0,1] is used to show whether any points are observed on the corresponding grid element. The 2D CNN can be naturally extended to the 3D grid.

- Vote3Deep [69]

The innovation of Vote3Deep is to perform a sparse convolution on the 3D representation, followed by a nonlinearity (ReLU) as activate function, which returns a new sparse 3D feature. This process can be repeated and superimposed like traditional CNN and the output layer

predicts the detection score. Prune duplicate detection with non-maximum suppression (NMS) allows the model to better handle objects behind each other because the overlap of 3D bounding boxes is less than their 2D projection.

- 3DBN [70]

The main contributions of 3DBN are two 3D backbones that improve detection accuracy. The first network structure uses a bottom-up residual module. The direct connection mode of sparse convolution used in the second network can be divided into four blocks, where each block generates a 2D feature map. The first network of 3DBN replaces these four blocks with a residual connection form. The second network integrates the FPN-like connection structure into the 3D convolution of 3D detection to fuse multi-scale features.

- VoxelNet [71]

The idea of VoxelNet is not complicated. First, the point cloud is quantified into a uniform 3D grid cell. A fixed number of points are randomly sampled in each grid (repeat if they are insufficient) and each point is represented by a 7-dimensional feature, including the X, Y and Z coordinates of the point, the intensity, R, and the point relative to the grid centroid (i.e., the mean value of all points in the grid) position difference ΔX, ΔY and ΔZ. The fully connected layer is used to extract the feature of the point and then the feature of each point is spliced with the feature average of all points in the grid to obtain a new point feature. The advantage of this feature is that it retains the characteristics of a single point and the characteristics of a small local area (grid) around the point at the same time. This Stacked Voxel Feature Encoding process can be repeated many times to enhance the feature description ability. Finally, all points in the grid are subjected to max pooling to obtain a fixed-length feature vector.

- SECOND [72]

This method uses the sparse convolutional network (SparsConv) to improve the efficiency of VoxelNet [71]. They also propose a sinusoidal error angle loss to solve the ambiguity problem of the rotation angle between 0-π.

- MVX-Net [73]

This method extends VoxelNet [71] by fusing an image and point cloud features at an early stage. Specifically, they project the non-empty voxels generated by VoxelNet onto the image and use a pre-trained network to extract the image features of each projected voxel. Then these image features are cascaded with voxel features to generate accurate 3D boxes. When compared to VoxelNet and SECOND [72], these methods can effectively use multi-modal information to reduce the number of false positives and false negatives. From the experimental results, the point fusion is slightly better than the voxel fusion result, which further shows that a lower fusion level may bring better results.

- PointPillars [74]

The idea of PointPillar is to convert 3D to 2D, that is, to quantify the point 3D cloud to a 2D XY plane grid. However, unlike the way that PIXOR [64] manually designs features, PointPillar directly stacks the points that fall into each grid and calls it a 'Pillar'. It then uses a similar methodology to PointNet to learn features and finally map the learned feature vector back to the grid coordinates to get data similar to the image. Doing so avoids the invalid calculation of 3D convolution and blank area in VoxelNet [71] and the problems of information loss and poor network adaptability caused by manual design features. The downside is that the learning of point features is limited to the grid and the contextual information of the neighbourhood cannot be extracted effectively.

- SA-SSD [75]

The core innovation structure of an aware single-stage detector (SA-SSD) is to apply the unique fine regression of the two-stage method to the first-stage detection method. For this reason, the authors use SECOND [72] as the backbone and adds two additional tasks to make the backbone have structure aware capabilities. As a result, the positioning is more accurate.

F-PointNet [51], SIFRNet [52], PointFusion [53], RoarNet [54], F-ConvNet [55], Patch Refinement [56], VeloFCN [67], 3D FCN [68], Vote3Deep [69], 3DBN [70], VoxelNet [71], SECOND [72], MVX-Net [73], PointPillars [74], and SA-SSD [75] all use voxel representation of the LiDAR point cloud. Compared to using a BEV map representation, voxel representation

creates more redundancy and requires more computational power, hence is not used in the proposed model in this thesis.

- 3D-SSD [76]

3D-SSD can be considered as an anchor-free single stage detector, which is also in line with the development trend of the entire object detection field and the speed of 3D-SSD can reach 25 FPS. Although the method proposed in this paper is not used in this thesis, it presents a different approach that is fast and accurate.

2.3.3 Transformer LiDAR Point Cloud-Based 3D Object Detectors

- VoTr-SSD and VoTr-TSD [77]

This article is among the first to use voxel-based transformer as a 3D backbone network for 3D object detection from point cloud data [77]. Due to their limited receptive field, traditional voxel-based 3D convolutional network detectors cannot effectively capture a large amount of environmental information. As an alternative, this paper introduces a transformer-based structure to find the relationship between voxels over a long-distance range through self-attention.

Considering the fact that non-empty voxels are sparse and numerous, it is not trivial to apply standard transformers directly to voxels. To this end, the authors propose a sparse voxel module and a submanifold voxel module, which can efficiently operate on empty and non-empty voxels.

In order to further increase the attention span while limiting the computational overhead corresponding to the convolutional detector, the authors also propose two multi-head attention mechanisms: local attention and dilated attention. At the same time, the authors further propose fast voxel query for speeding up voxel queries.

For testing and evaluation, VoTr-SSD uses SSD as backbone while VoTr-TSD uses R-CNN, which shows that the model is adapted to different backbones.

- Summary

In summary, LiDAR point cloud-based 3D object detection algorithms mainly vary in the data preprocessing method and detector structures. For the preprocessing method of the input data, compared with voxel-based methods and directly using point cloud, generating different 2D viewpoints from the original point cloud would be more efficient and more practical for experiments without high computing power hardware. Furthermore, fusing the LiDAR point cloud and RGB images, as explored in this thesis, can provide the detector with both the 3D information and colour feature, which can improve the detector's performance.

The structures of the LiDAR point cloud-based detection models are mostly elaborations from classical 2D detection models summarized in Section 2.2. The optimization methods of the models include but are not limited to the fusion of various models, improvement of the loss function, and adjustment of the model depth.

2.4 LiDAR Point Cloud Datasets

To train a neural network efficiently, reliable datasets are essential. Experiments conducted in this research not only use RGB images, but also leverage high-precision point cloud data collected by the LiDAR scanner for depth information, as reported in Section 2.1.1. However, a disadvantage of using LiDAR is the cost. Therefore, an open-source dataset is used in this research. Advantages of using an open-source dataset are also that it allows for easy comparison with other research methodologies, algorithms and benchmarks, while reducing the problems of dealing with incomprehensible results because of dataset differences.

As shown in

Table 2.5, this section summarizes the widely used and reliable LiDAR based open-source datasets and gives a brief introduction of their collection methods and characteristics. In the literature, these datasets are used for semantic segmentation, 3D object detection and recognition and they are widely exploited in many fields such as autonomous driving and remote sensing.

Table 2.5 LiDAR point cloud datasets.

Dataset	Format	Labels	Main Application	Points/Objects	Classes
Semantic 3D [79]	ASCII	X, Y, Z, Intensity, R, G, B	Segmentation	4 billion points	8
Oakland 3D [80]	ASCII	X, Y, Z, Class	Segmentation	1.6 million points	5
iQmulus [81]	PLY	X, Y, Z, Intensity, GPS time, Scan origin, Class	Segmentation	300 million points	22
Paris-Lille-3D [82]	PLY	X, Y, Z, Intensity, Class	Segmentation	143.1 million points	50
KITTI [2]	multiple	3D bounding boxes	2D/3D detection and recognition, segmentation, object tracking	80256 objects	8
Sydney Urban Objects Dataset [83]	ASCII	X, Y, Z, Range, Intensity	2D/3D detection and recognition	588 objects	14
ModelNet [84]	ASCII	X, Y, Z, Edges, Faces	2D/3D detection and recognition	4899 objects	40

- Semantic 3D [79]

Semantic 3D is a dataset designed to study the segmentation task. Using one of the largest outdoor dataset scenes, captured with LiDAR technology, it includes more than 4 billion points of data, covering an area of approximately 110,000 square meters. This dataset contains eight categories of labels and is divided into a training set and test set of almost equal size. These data are collected by static LiDAR with high measurement resolution and long measurement distance.

The challenges of using this dataset for research or application mainly stem from its massive point cloud, uneven distribution of point density and severe occlusion.

- Oakland 3D [80]

Oakland 3D Point Cloud Dataset is one of the earliest open-sourced LiDAR datasets. Oakland 3D's mobile platform equipped with LiDAR is used to scan the urban environment and generate approximately 1.6 million points and 100,000 of those points are divided into the validation set, leaving the rest for training. The entire dataset is labeled into five categories: wire, vegetation, ground, pole/tree-trunk and facade. The Oakland 3D dataset is small, so it is suitable for networks that are not too deep. In addition, this dataset can be used to test and adjust the network architecture before final training on other bigger datasets, without spending a lot of training time.

- TerraMobilita/iQmulus [81]

TerraMobilita/iQmulus dataset was acquired by a mobile LiDAR system in the Paris urban environment. There are more than 300 million points in this dataset, covering 10 kilometers of city streets. All the data is divided into 10 independent areas and marked with more than 20 sub-categories. However, this dataset also has severe occlusion, so it is rarely used in recent research.

- Paris-Lille-3D [82]

Paris-Lille-3D is also a common dataset used to perform segmentation tasks. Compared with Semantic 3D [79], Paris-Lille-3D contains fewer points (140 million points) and a smaller coverage area ($55{,}000\ m^2$). Another major difference is that Paris-Lille-3D was acquired by mobile LiDAR systems in two cities: Paris and Lille. When compared with Semantic 3D, the points in this dataset are sparse and the measurement resolution is relatively low. But this dataset is more like the LiDAR data obtained by autonomous driving vehicles. The entire dataset is fully annotated into 50 classes and unequally distributed in three scenes: Lille1, Lille2 and Paris. These 50 classes are combined into 10 coarse classes for scientific research.

- KITTI [2]

Different from the above LiDAR datasets which are specific to segmentation tasks, KITTI is acquired from an autonomous driving platform and records driving on city street using digital cameras, LiDAR and global positioning system/inertial measurement unit (GPS/IMU). It can be used for depth or stereo based 2D object detection, depth or stereo based 3D object detection and recognition and optical flow algorithm development. Geiger and his team used two high-resolution colour and grayscale video cameras situated on top of a car. They captured two sets of RGB images and two sets of grayscale images. They also used a Velodyne laser scanner to capture 3D information on the images and a GPS for a localization system. This dataset can serve multiple purposes. For 3D object detection benchmark, it consists of 7481 training images (and point clouds) and 7518 test images (and point clouds). It has eight classes (‘Car’, ‘Van’, ‘Pedestrian’, ‘Person (sitting)’, ‘Cyclist’, ‘Tram’ and ‘Misc’ (e.g., Trailers, Segways)) for detection tasks. There is also a non-functional class called ‘don’t care’ that captured bad data such as objects that were too far for the laser scanner to detect them. Despite the fact that they have labeled eight different classes, only the classes ‘Car’, 'Pedestrian' and ‘Cyclist’ are evaluated in our benchmark, as only those classes offer enough instances for a comprehensive evaluation.

- Sydney Urban Objects Dataset [83]

Sydney Urban Objects Dataset contains urban roads in the Sydney Central Business District, New South Wales, Australia and uses a set of LiDAR to scan and obtain point cloud data on the roads. This dataset includes a total of 588 marked objects, divided into 14 categories, including vehicles, pedestrians, signs and trees. The entire dataset is divided into four folders and used for training and testing respectively. Similar to other LiDAR datasets, the objects collected in this dataset are sparse and incomplete in shape. Although it is small and not ideal for classification tasks, it is still used for benchmark due to the limitations of the cumbersome labeling process.

- ModelNet [84]

ModelNet dataset is the largest existing 3D object recognition benchmark. This dataset consists of general objects with uniformly distributed point density and complete shapes in the CAD

model. There are approximately 130K marked models in a total of 660 categories (for example, cars, chairs and clocks). The most used benchmarks are ModelNet40, which contains 40 general objects and ModelNet10, which contains 10 general objects. These two datasets would be an ideal choice for deep architecture for 3D object recognition tasks if having access to computationally intensive hardware and more than enough time.

To solve the long-term autonomous driving tasks, a new autonomous driving dataset named Long-Term Autonomy [86] is published. They traversed 1,000 kilometers in the centre of Oxford, England and spent a total of one year collecting RGB images, LiDAR and GPS data. Therefore, the biggest highlight of this dataset is that they can capture the appearance of different scenes under various lighting, weather and seasons. Because the dataset was collected over a long period of time, it can provide insight into issues such as various road conditions and driving regulations in different weather that impede the development of autonomous vehicles at different times of the year.

Chapter 3 3D Object Detection and Recognition

Compared to other object detection and recognition tasks, autonomous driving applications impose higher requirements on the performance of detectors. The detector needs to be able to detect the objects quickly and accurately, which includes both the objects that are in the fully visible view and the objects that are partially or fully obstructed. That is, to ensure the vehicle gains the ability to deal with complex traffic conditions in the real-world environment.

From the literature review, it can be seen that the deep learning method not only achieves good results in 2D image processing and detection, but also can successfully perform 3D object detection and recognition tasks. In this chapter, an efficient deep learning-based 3D object detection and recognition model is proposed.

The proposed detector uses BEV maps that are encoded as 2-channel bidimensional grids where each pixel of a 2x2 map contains respectively a “cumulated height” parameter associated with 3D points mapped to the pixel, and a “cumulated intensity” estimate provided by a LiDAR sensor along with every 3D point measurement. The BEV maps converted from point cloud data collected by a LiDAR scanner are combined with front forward view images collected by an RGB camera as inputs. In order to distinguish it from the mini version that will be proposed in Chapter 4, the detector in this chapter will be referred as the “full detector”. The latter is based on DarkNet-53 [1], combined with feature pyramid network [11] and a classification head. To optimize the detector, a Euler angle encoding is added to the regression of the model, which adds orientation feature to the regression. Furthermore, to solve the bias introduced by KITTI dataset [2], focal loss [3] is used as the classification loss with GIoU [4].

In this chapter, the methodology for the proposed full detector will be explained. As shown in Figure 3.1, the bird’s-eye view (BEV) maps are converted from the LiDAR point clouds, to reduce the amount of computation without losing required features. Then the BEV maps and the front forward view RGB images are used as input data to extract features through a DarkNet-53 based convolutional neural network (CNN). Using the feature pyramid network (FPN) and Euler angle region proposal network (E-RPN), the full detector is trained and uses 3D bounding boxes

(B-Box) to mark the position of the target objects on the RGB images and 2D B-Box on the BEV maps.

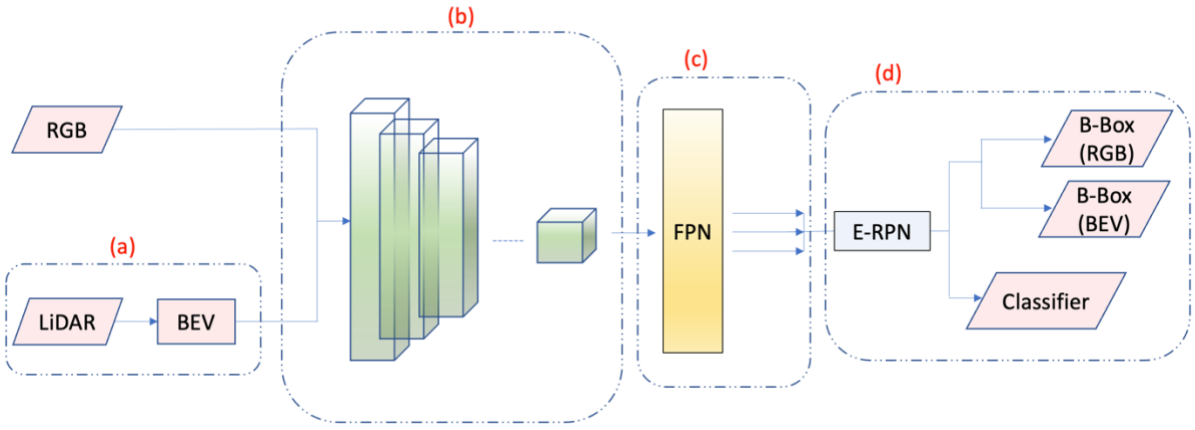


Figure 3.1 Architecture of the proposed model: (a) point cloud preprocessing converts LiDAR point cloud to BEV map, (b) backbone convolutional neural network to generate feature maps, (c) feature pyramid network (FPN) for integrating features, (d) detection head with Euler angle regression.

Following the methodology, the implementation details such as the training environment and some hyperparameters, and the analysis of experimental results are discussed. Aside from the general statistics on the detection and recognition results, the KITTI evaluation metrics is used to appraise the proposed full detector's adaptability to occlusion and support a comparison between similar advanced research in LiDAR point cloud-based 3D object detectors. To demonstrate the novelty of the proposed methodology, ablation studies are designed to evaluate the effect of adding Euler angle regression and focal loss [3].

3.1 Point Cloud Preprocessing

A 3D point cloud is the default representation of the data collected by a LiDAR scanner. The depth data used for this thesis is a set of point clouds collected by a Velodyne laser scanner, where each point cloud is represented as an array with N rows and four columns, with each row

corresponding to a point. There are over 120,000 points in each point cloud available in the KITTI dataset [2] considered here for experiments. Each point is represented by three values in the space domain (x, y, z), which are the first three columns in the point cloud array, and the fourth value represents the intensity, which is explained in Section 2.1.1 with Equation (2.1).

As per the literature review in Section 2.3, there are mainly two ways to process a 3D point cloud: one is directly processing a 3D matrix, another is to convert the 3D matrix into 2D, then processing it as a 2D matrix. In some papers using a LiDAR point cloud for object detection and recognition [71][72][75], researchers directly use the LiDAR point clouds to train their detectors. The convolution operations take more time and memory in 3D compared to when information is encoded in 2D. Moreover, the 3D point cloud being sparse leads to many empty anchors, which lowers the efficiency of the detector. For some models like MV3D-Net [40], the point cloud is converted into a front view and a BEV map and results show that this is more efficient than processing the 3D point cloud directly, while not lowering the accuracy.

The BEV map is a graphical representation of the point clouds from a bird's-eye view. It is obtained by projecting the discretized LiDAR point cloud on a plane perpendicular to the height direction (i.e., the LiDAR Z axis according to Figure 3.3). Therefore, a BEV map can be an equivalent representation of the 3D location information contained in the LiDAR point cloud with no overlapping location of objects in the front forward view direction (i.e., the LiDAR X axis according to Figure 3.3). Hence it is more efficient when a large amount of data is being processed. While a BEV map provides the 3D location information, the intensity can also be saved in a different channel of the BEV maps. The front forward view and colour information can be provided by corresponding RGB images. Therefore, all the required information can be obtained by combining the BEV maps converted from LiDAR point clouds and the RGB images.

This section explains how to preprocess the LiDAR point clouds and convert them into the BEV maps. As shown in Figure 3.2, the point clouds can be plotted in three dimensions and the colour of the points represents different intensity. First, the point cloud coordinate system is transformed based on the calibration information of the LiDAR scanner that is used to collect the data and its location relative to the RGB camera. Then based on the size of the objects and the scanning range of LiDAR scanner and RGB cameras, the region of interest (ROI) is set to extract

the points contained within the ROI from the complete LiDAR point cloud. After the ROI is set, the 3D points position is mapped to corresponding pixel position in the bird's-eye view, which converts the selected region from the point cloud into BEV map. Besides saving the depth information, the height and the intensity information are also saved to the BEV map. The height information, corresponding to the elevation or Z coordinate of any point in the LiDAR point cloud, is filtered through height thresholding (as will be detailed in Section 3.1.3) to focus on the details of target objects, and the intensity information from the raw point cloud is saved in another channel to the BEV map. The proposed point cloud preprocessing method is inspired by [40] [41] [43], combined with height thresholding to focus on the selected ROI. The generated BEV maps are used for the training of the proposed model as well as plotting the resulting bounding-boxes (B-Boxes), which will be shown in the following sections of this chapter.

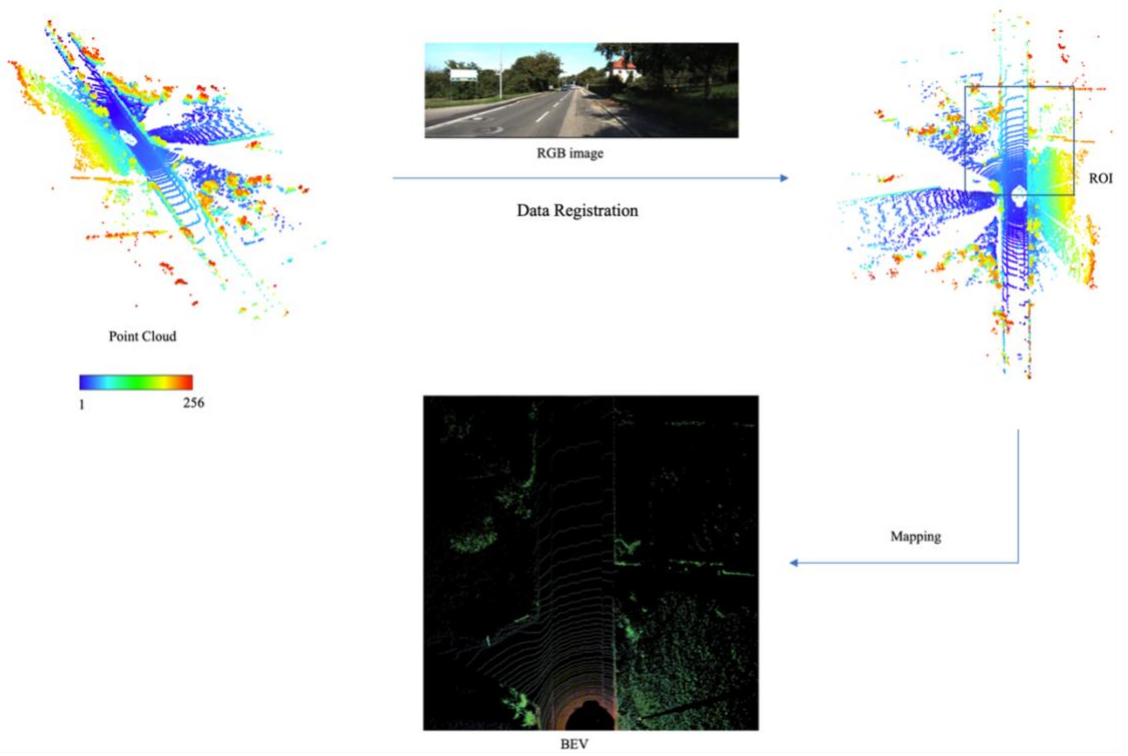


Figure 3.2 3D point cloud preprocessing: aligning the 3D point cloud and the RGB image into one coordinate system, estimating the region of interest (ROI), and mapping the 3D point cloud within the ROI into the bird's-eye view (BEV) map. The point cloud is coloured by the intensity value of its corresponding point, where blue represents the lowest intensity and red presents the highest.

3.1.1 Data Registration

The 3D point cloud and RGB images are obtained from different sensors and both the Velodyne LiDAR scanner, which provides 3D point cloud, and the RGB camera, which provides the RGB images, have their own coordinate system. Before the two sets of data are used together as input to the proposed model in this thesis, they need to be registered so that the points in the LiDAR point cloud can match with the corresponding points in the RGB image. Hence, coordinate system transformation is performed on LiDAR point cloud's coordinates to align its coordinate axes and origin to the RGB image coordinates.

Figure 3.3 (a) shows the RGB coordinate system on one of the RGB images used for the experiments in this thesis. To help understand the relative direction of the coordinates in the considered application for autonomous vehicles, Figure 3.3 (b) shows the original relative direction of the 3D point cloud and RGB images.

Figure 3.3 Relative coordinates of RGB images, LiDAR point cloud and BEV in (a) the front forward view RGB image, (b) 3D view of a car.

With the help of the calibration matrices provided by KITTI [2], the registration of the LiDAR point clouds and RGB images is performed using matrix transformation. First, the point cloud is calibrated using the calibration matrix of the Velodyne LiDAR scanner provided by KITTI:

$$C_c = TC_o \tag{3.1}$$

Where T is the calibration matrix, C_o represents the vector of a point in the original point cloud (first three values for the coordinate of a point and 4th value for the intensity), and C_c represents the calibrated point.

Then the calibrated vector is first rotated 90 degrees counterclockwise around the Y_{LiDAR} axis, then 90 degrees clockwise around the Z_{LiDAR} axis in order to align the LiDAR reference frame with the RGB frame. The corresponding matrix transformation formulation is as follows:

$$C = R_z R_y C_c \tag{3.2}$$

$$R_y = \begin{bmatrix} cos(-\frac{\pi}{2}) & 0 & -sin(-\frac{\pi}{2}) & 0 \\ 0 & 1 & 0 & 0 \\ sin(-\frac{\pi}{2}) & 0 & cos(-\frac{\pi}{2}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$R_z = \begin{bmatrix} cos\frac{\pi}{2} & -sin\frac{\pi}{2} & 0 & 0 \\ sin\frac{\pi}{2} & cos\frac{\pi}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Where C is a point vector in the point cloud after registration. R_y represent the rotation matrix around the Y_{LiDAR} axis and R_z represent the rotation matrix around the Z'_{LiDAR} axis where Z'_{LiDAR} is the transformed Z axis after the previous rotation, that is pointing toward the front of the vehicle in Figure 3.3.

3.1.2 Mapping 3D Points within Region of Interest to 2D Pixels

In the KITTI dataset, the point cloud of each scene takes 1.9 MB, which not only takes up lots of memory, but also highly increases computation for both training and testing and will reduce detection efficiency. Therefore, it is useful to focus on a region of interest (ROI) in the point cloud. To balance the model's efficiency while covering all the annotated target objects in the corresponding RGB image, an horizontal 2-dimensional ROI over the X'_{LiDAR}, Z'_{LiDAR} axes of the transformed LiDAR frame is manually set as a rectangle that spans 40m on either side of the LiDAR scanner, and 80m in front of it, as shown in Figure 3.4. The X'_{LiDAR}, Z'_{LiDAR} axes are the rotated X, Z axes of the initial LiDAR coordinate (detailed in Section 3.1.1). Thereafter, only 3D points contained within the defined horizontal ROI are preserved in the BEV representation, while measurements beyond that ROI are discarded.

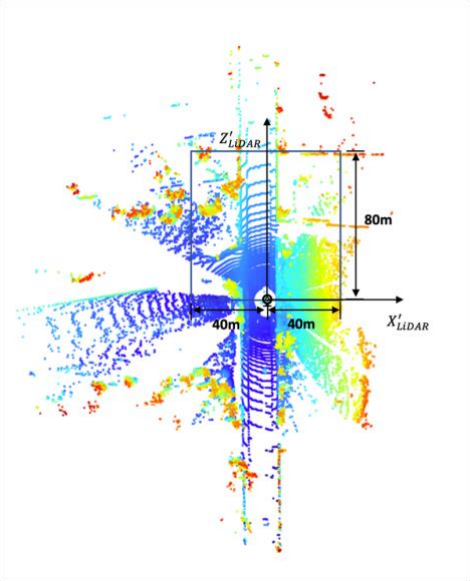


Figure 3.4 Manually selected ROI in the point cloud (as seen from above, along the X'_{LiDAR}, Z'_{LiDAR} axes).

The points extraction process is performed by comparing every point's coordinate with the boundaries that define the horizontal ROI:

$$extracted\ point\ cloud = \{C(x, y, z, i), where\ (x, z) \in ROI\} \qquad (3.5)$$

The parameters of the square ROI are tuned to retain 99% of the labeled target objects in the training and testing dataset. A square ROI shape is preferred over other shapes (round, triangular, etc.) as it is more efficient for the proposed CNN model to process a square distribution of points given that its shape will not change through convolution operations.

The point cloud data collected from the LiDAR scanner are 3D points with real (x, y, z) values that carry depth information. Up until this point, we registered the point cloud with its corresponding RGB image and filtered out the points outside the ROI. However, the registered points within the ROI carry real number values that need to be mapped into integer values that correspond to pixel location on the discretized bird's-eye view (BEV) map.

One way to map a 3D point position's value into BEV's pixel position is by converting all x and z values (corresponding to the point cloud data projection over the ground surface) into their rounded up or down approximation, however, resolution loss may occur. For the proposed detector, the model needs to capture fine details and recognize cars and even smaller things, such as pedestrians. By brutally converting the position value into integers will cause the loss of fine features that are crucial for the detector to extract.

Therefore, before typecasting point position into integers, we need to scale the data first. In the experiments conducted for this thesis, the real number values in the point cloud are rescaled to form a 608×608 size BEV map and carry features down to a 10 cm resolution. The scaling ratio can be manually adjusted. Generally speaking, the higher the scale goes, the finer features will be captured in the BEV map, and the more memory and computation power it will take for the detector to process the input data. The feature resolution should not exceed the range accuracy of the sensor that is used to collect the point cloud. The LiDAR scanner Velodyne HDL-64E [6] used for the experiment of this thesis has a range accuracy of 2 cm, which is compatible with the set resolution at 10 cm.

The rescaled point cloud values on the X'_{LiDAR}, Z'_{LiDAR}, axes in the current LiDAR coordinate system are then rounded up to integers that represent the index (pixel coordinate) on the BEV map, as follows:

$$(x_r, y, z_r, i) = (int(\frac{x}{r}), y, int(\frac{z}{r}), i\ ) \quad (3.6)$$

Where $\left(int(\frac{x}{r}), y, int(\frac{z}{r}), i\ \right)$ is the rescaled vector in the point cloud and x, z are the coordinates on the X'_{LiDAR}, Z'_{LiDAR} axes in the extracted point cloud C from the previous step. Ratio $r = \frac{10cm}{2cm} = 5$.

At this moment, when examining the data matrix of the point cloud, some of the points have negative x_r, z_r values and need to be converted to positive values in order to project the point cloud into the BEV map (since the x_r, z_r values will be the index of the BEV map). As shown in Equation (3.7), the point cloud is shifted to positive values to make sure that all values are equals or greater than 0. This step normalizes all the point clouds in the input dataset.

$$(x_s, y, z_s, i) = (x_r + (0 - min(x_r)), y, z_r + (0 - min(z_r))\ ,\ i) \quad (3.7)$$

3.1.3 Recording the Height and Intensity Information

For every 3D point from the original point cloud projected on the BEV map, the corresponding height information represented by the value on the Y'_{LiDAR} axis and the intensity information represented by the 4th values of each vector in the point cloud need to be extracted from the 3D LiDAR point cloud and saved as the height and intensity channels of the BEV map.

Inspired by PIXOR [64], a vertical ROI on the Y'_{LiDAR} axis in the transformed LiDAR coordinate system on C is selected and recorded in the height channel. Considering that the position of the LiDAR scanner used in the experiments for this thesis is at $y = 1.73$ m above the ground and that the height range of interest should cover all the target objects that the proposed model is to detect and recognize (namely vehicles, pedestrians, and cyclists), the height ROI is set to

$[-2m, 1.25m]$. The height of the vertical ROI is set based on two criteria: statistically, in the ground truth of the dataset used for this thesis, all the target objects are within the height range of [-1.63m, 0.78m]; and theoretically, all the target objects should be on the ground and not exceed 3m in height (considering the height of the cars, pedestrians, cyclists in real life), while the LiDAR scanner that collected the point cloud was set on top of a moving vehicle.

A height thresholding is therefore used to preserve data from the point cloud that are within the select vertical ROI. After that is being done, the height coordinates (on Y'_{LiDAR} axis on C) within the ROI are rescaled into the range]0, 255[, while the height coordinates outside the vertical ROI are cast to 0 or 255. Finally, the respective height values of the points from the point cloud that are mapped into the same pixel position on the BEV map are cumulated and recorded to the height channel (4^{th} value) of the BEV map representation.

Similarly, the intensity information from corresponding points in the point cloud contained within the selected vertical ROI and falling within the same pixel position of the BEV map are cumulated and recorded to the intensity channel (5^{th} value) of the BEV map.

Alternatives to height and intensity values cumulation were considered during the design on the proposed solution. As such, the height and intensity channels could be saved as the maximum height and intensity value of the corresponding 3D points falling within the same pixel position in the BEV map. However, it was observed that the information recorded in the BEV map is then more sensitive on the choice of the height threshold when compared to recording cumulated height and intensity values instead. Another method consists of slicing the point cloud into different height groups, then save as multiple height channels, as proposed by Chen et al. [40]. The downside of this method is that it highly increases the required computational load by introducing additional dimensions to the BEV map.

In summary, the point cloud data collected by the Velodyne LiDAR scanner is preprocessed before being used as part of the input data for the proposed 3D object detector. After applying coordinate system transformations to register the LiDAR point cloud to the corresponding RGB images, the point cloud is then scaled and mapped into the 2D pixel positions of the 608×608 BEV map. To increase the detection efficiency, the BEV map only covers a selected ROI along the three dimensions which is defined by the combination of a 2-dimension (X'_{LiDAR}, Z'_{LiDAR})

horizontal ROI and a 1-dimensional (Y'_{LiDAR}) vertical ROI that allows to focus on target objects with ground truth labels in the KITTI dataset [2]. The BEV map that emerges from the converted point cloud contains two channels: a cumulated height channel and a cumulated intensity channel.

3.2 Model Architecture

After converting the LiDAR point cloud into a BEV map, it is used along with the corresponding RGB image as the input to the proposed object detection model. The RGB images used in the experiments of this thesis are of original dimension 1242×375 pixels. Inspired by YOLOv3 [1], the RGB images are compressed to 608×183 while keeping their original scale using bilinear interpolation (detailed in Section 2.1.6) and then expanded to 608×608 pixels to match with the size of the BEV maps by filling the lower and upper parts with $[R, G, B] = [128, 128, 128]$ values, as shown in Figure 3.5(a).

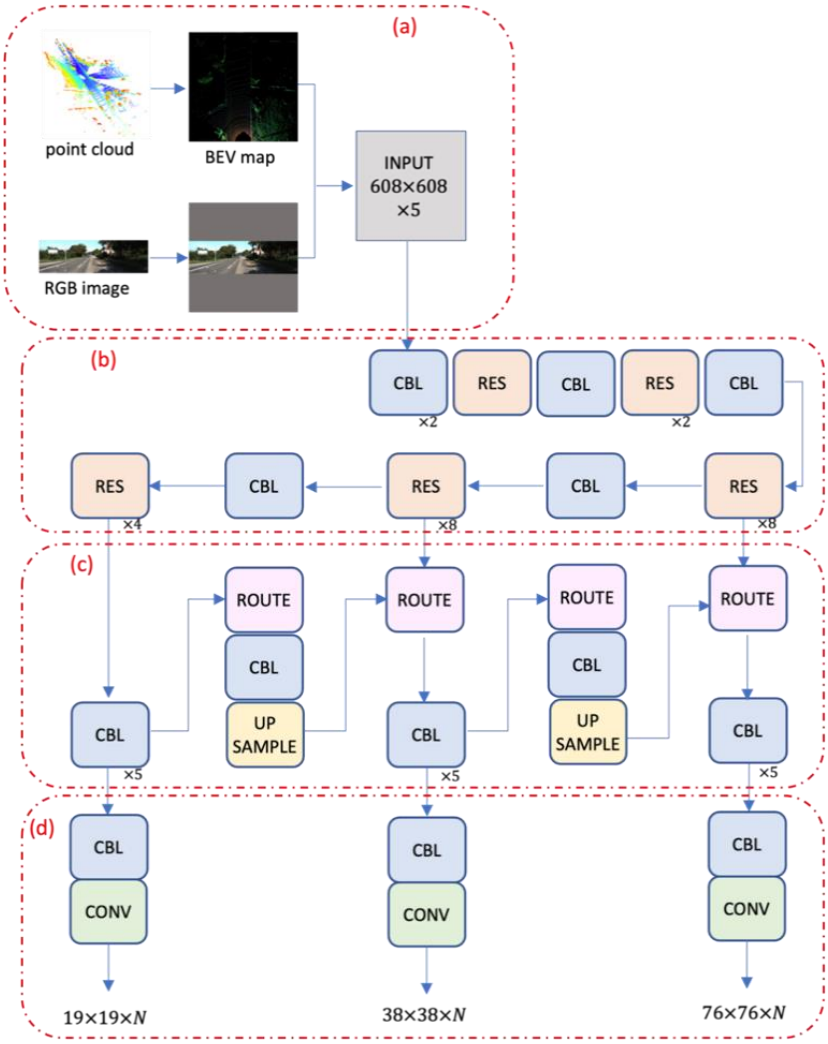


Figure 3.5 Proposed 3D object detection model structure: (a) image preprocessing converts point clouds into a 2-channel BEV maps and rescales the corresponding RGB image, (b) shows the structure of the backbone of the proposed model, (c) shows the structure of feature pyramid network (FPN), (d) shows the detection head which output prediction matrices in three different scales. CBL is the combination of convolutional layer, batch normalization and Leaky ReLU activation. Res unit contains CBL and shortcut layer, as shown in Figure 3.6.

A DarkNet-53 [1] based backbone is used to extract the features and a feature pyramid network (FPN) is used to process the feature maps with different levels of the features and scales. The proposed structure is shown in Figure 3.5. For the detection head, Euler angle is added to the

regression operation of the proposed network. Meanwhile, focal loss [3] and GIoU [4] are combined for classification and for reducing the impact of the imbalanced input data. Details of the proposed architecture will be explained in this section.

3.2.1 Backbone and feature pyramid network (FPN)

The proposed method combines BEV maps generated from a LiDAR point cloud and the associated RGB images as a single 5-channel (height, intensity, R, G, B) input to the 3D object detection model. Doing so preserves the 3D spatial distribution information collected by the LiDAR scanner and the colour information collected by the RGB camera.

Although this approach uses more computational resources compared to standard 2D image object detection, it does reduce the amount of computation required compared to using the BEV maps and RGB maps as separate inputs. Given the importance of quick decision making for autonomous driving tasks, any improvement in object detection speed while maintaining high object detection accuracy is valuable. With this in mind, a single stage detection model is proposed in this thesis.

Among the backbones summarized in Sections 2.2 and 2.3, compared to VGG-16, ResNet-50 and ResNeXt-101; the Darknet series are more efficient and commonly used for single stage detectors. The state-of-the-art CSPDarknet53 used in YOLOv4 and YOLOv5 shows better performance on small object detection, at the cost of a deeper model. Darknet19 is not as deep, but the extracted features do not support high accuracy detection. After comprehensive consideration, the backbone of the proposed object detection model is based on DarkNet-53 [1], modified to process the BEV maps and the RGB images. As shown in Figure 3.5(b), the structure of the backbone consists of CBL which means convolutional layer, batch normalization [8] and Leaky ReLU [9] activation, and of Res units, which contain CBL and shortcut layer. The structures of CBL and Res units are shown in Figure 3.6.

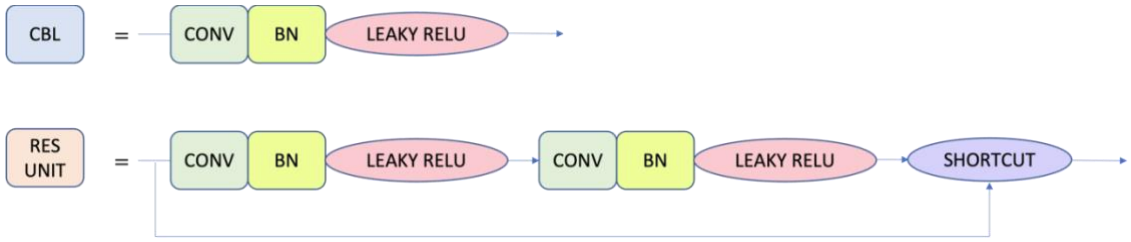


Figure 3.6 The structure of CBL (top) and res unit (bottom).

- CBL

The structure of CBL (convolutional layer, batch normalization and Leaky ReLU activation) is shown in the upper part of Figure 3.6. Each convolutional layer in a CBL unit consists of several convolutional units, which is introduced in Section 2.1.2. The input of the proposed detector contains RGB images scaled to 608×608 pixels with 3 channels (red, green and blue channels) and 608×608 BEV maps with 2 channels (height, intensity), different from the input size used for the original YOLOv3 model. Hence, the parameters of each convolutional layer are optimized to adapt to the scale of the input data.

Convolution operations can extract different features from the input, which makes the convolutional layers the most important part of the backbone. The low-level convolutional layer only extracts some low-level features such as edges, lines and corners. The deeper the network goes, more complex features are iteratively extracted from those low-level feature maps. Here in the proposed architecture shown in Figure 3.5, since pooling layers are omitted from the entire network, the convolutional layer also performs down sampling, which brings three major benefits to the model:

- It reduces the amount of memory and calculation. The smaller the feature maps, the less memory is used for further steps and the lower amount of calculation is needed.
- The down sampling operations enlarge the receptive field, so that with a convolution kernel of the same size, features can be extracted from a larger range. Large receptive

fields can better adapt to multi-classification tasks, such as the autonomous driving task and can obtain smoother feature boundaries.

- There are several branches with different levels of down sampling that can facilitate the fusion of multi-scale features, which makes the classification more accurate.

As shown in Figure 3.6, for CBL units, there is always a batch normalization (BN) operation that follows convolutional layers. As explained in Section 2.1.2, batch normalization recentres and rescales the output of the former convolutional layer so that it follows the distribution with a mean value of 0 and a standard deviation of 1. By shifting internal covariate and ensuring the input of every layer is distributed around the same mean and standard deviation, batch normalization smooths the loss function and improves the training speed of the model by optimizing model parameters.

The "pattern" that the proposed model learns and uses for detecting the objects and recognizing them is not a simple linear function, which means non-linear transformations need to be added into the model's backbone. The convolution operations are multiplying data matrix by weight matrix, which is a linear operation and so is the normalization operation. Therefore, a non-linear activation function is added after the linear operations in each CBL. Leaky ReLU is used as the activation function, and it is explained in detail in Section 2.1.2.

- Res unit

The Res unit is a combination of CBL and shortcut layer, as shown in the lower part of Figure 3.6. Shortcut layers are used as skip connections to combine feature maps from its previous CBL unit and the one before by simply performing an addition operation. Since the shortcut layer connects the feature maps of two connected CBL together, it can be considered as an empty layer that does not change any parameters.

- FPN

The output of convolutional layers are the feature maps and the focus or extracted features in each feature map varies depending on the level of the convolutional layer. For autonomous driving tasks, the detector needs to process objects at different scales and on different classes on

both RGB images and BEV maps. Therefore, feature maps of different levels need to be extracted and processed.

As explained in Section 2.1.3, the feature pyramid network (FPN) [11] is a useful tool to optimize the proposed model to process feature maps at different scales. The detailed structure used in the proposed model is shown in Figure 3.5(c). For FPN, the route layer is used to connect different feature maps, CBL for further extracting the features at different scales and upsampling operations to amplify the extracted high-level features.

The route layer serves two purposes. First, it can be used to output the feature maps of one specific layer by indexing the value of that layer within the route layer's attribute layer. Second, it can also be used to output the concatenated feature map of two or more layers by concatenating along their dimension set in the attribute layer.

The upsample layer performs an upsampling operation on the extracted features. The deconvolution operation is used for upsampling in the proposed network. In FPN, the current feature map can obtain the information of the "future" layer and because of that, the low-level features and the high-level features are combined to improve the detection accuracy.

3.2.2 Detection Head with Euler Angle Regression

The input of the proposed model contains RGB images and BEV maps. The RGB image part of the input represents the front forward view for autonomous driving, with three different colour channels. The BEV map part of the input represents the bird's-eye view over the detection range, with a height channel and an intensity channel. The output of the proposed model represents the detection and recognition result, which contains prediction matrices at three different scales. The prediction matrices then are used to draw B-Boxes around detected target objects and their respective classification.

Inspired by complex YOLO [87], the rotation angle uses Euler representation. The Euler angle was introduced by Leonhard Euler to describe the orientation of a rigid body with respect to a fixed coordinate system [88].

Euler angles are a set of three angles that can be used to describe the orientation of an object in 3D space. They are typically represented as pitch (ϕ), roll (φ), and yaw (θ). In the proposed method, the yaw of the target object, which represents its rotation around the vertical direction (i.e., over the surface of the ground in the BEV map that provides an observation point from above) is predicted and regressed in the BEV map.

Considering the compatibility, Euler angle is widely used in computer graphics and robotics, and there are many existing tools and libraries for working with them. This can make it easier to integrate Euler angle representations into existing systems and workflows compared to other representations such as fixed angles.

In this section, a Euler angle regression network is proposed and used as part of the detection head in the proposed model. The detection head is based on the YOLOv3 anchor regression method, modified by adding BEV variables and rotation angle regression.

- YOLOv3 based anchor regression

Through the backbone convolutional neural networks and the FPN, feature maps at three different scales are extracted from the input of the proposed model. As shown in Figure 3.1(d), a detection head then is used to generate the detection and recognition results based on these feature maps.

Within the detection head, each feature map will be divided into grid cells. For example, for a 32× down-sampled feature map, the feature map is in 19×19, which is divided into 361 grid cells. Each grid cell can be seen as a 1×1 rectangle. For each grid cell in the feature map, there are three anchors at different scales. Hence, as shown in Figure 3.7, the total number of prediction matrices for each input is $19 \times 19 \times 3 + 38 \times 38 \times 3 + 76 \times 76 \times 3 = 22743$.

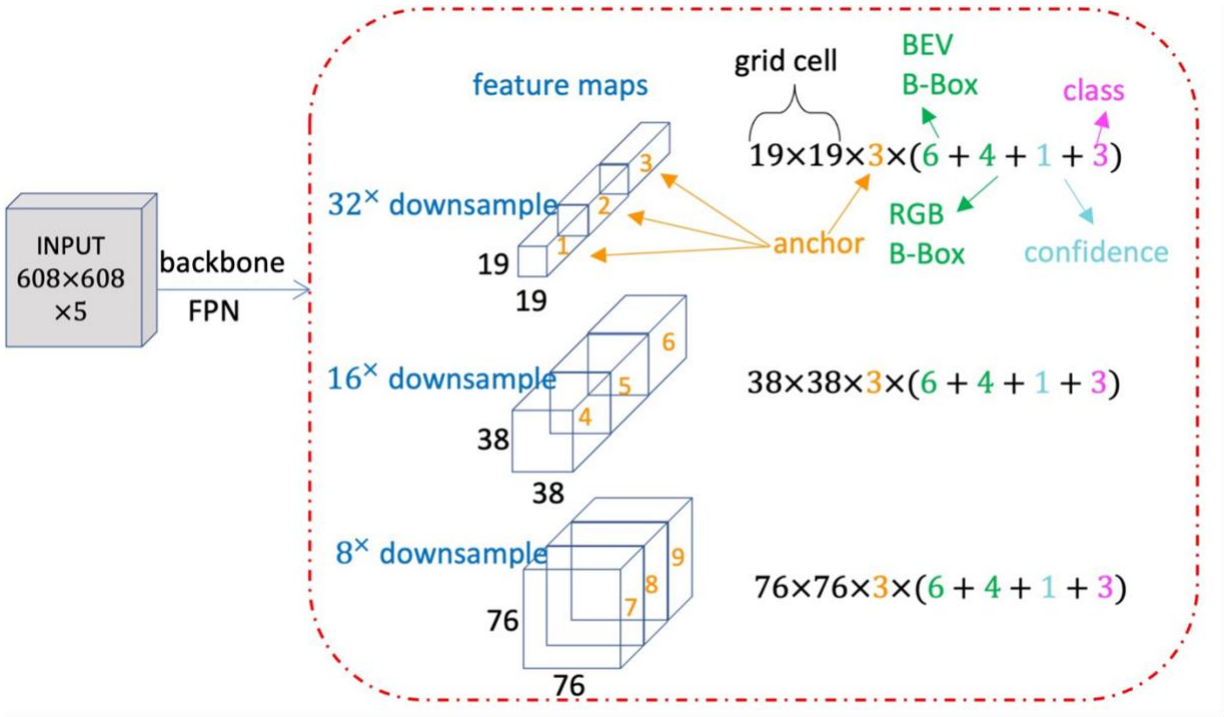


Figure 3.7 The detection head and its prediction matrix.

Anchors are priors for bounding boxes (B-Box). They are equivalent to a reference frame for the predicted B-Box. Based on this reference, the predicted B-Box generated by the detection head only needs to be fine-tuned based on this anchor. For the detector, fine-tuning anchors is easier to learn than generating B-Box from scratch [29].

The proposed detector learns how to fine tune the prior anchors by regression, which, as explained in Section 2.1.2, is a process of adjusting parameters to minimize the loss function. The loss functions used in the proposed detector will be explained in Section 3.2.3.

For every anchor, there is a prediction matrix that contains the parameters used for regression during training and the detection result. The output prediction matrix of the proposed detection head is shown in Figure 3.8. It contains the B-Box prediction matrix for both the RGB front forward view and the BEV, a confidence score, and classification scores over the 3 classes considered in this thesis.

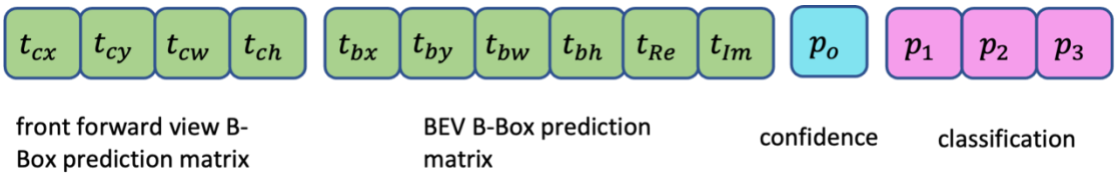


Figure 3.8 The prediction matrix.

The confidence score p_0 indicates the confidence that the predicted B-Box contains an object. If this predicted B-Box corresponds to the background, then this value should be 0.

The classification scores p_1, p_2, p_3 represent the possibility that the category of the predicted B-Box falls into 'car', 'pedestrian' or 'cyclist' respectively. For the final output, only the B-Box with p_0 higher than a detection threshold will be kept, and the classification shows max (p_1, p_2, p_3).

To adapt to the different viewpoints of the predicted input, the B-Box prediction matrix for the proposed detection head is modified. It is divided into two parts: one for the front forward view, another one for the BEV.

- The front forward view B-Box

In real world, cars, pedestrians, and cyclists stand or move on the ground. Therefore, from the front forward view of a moving vehicle, any target object should be parallel to the x-axis of the input RGB images. In other word, the front forward view B-Box can be represented simply by the centre coordinate and the width and length of the B-Box. For the regression of the proposed detector, the centre coordinate is represented by the front forward view B-Box prediction matrix with 4 variables: t_{cx} and t_{cy} are the offsets of the centre coordinate of the predicted B-Box, and t_{cw} and t_{ch} are the scaling of the width and height of the predicted B-Box relative to the anchor.

Therefore, as shown in Figure 3.9 (a), the predicted B-Box can be calculated by the following equations:

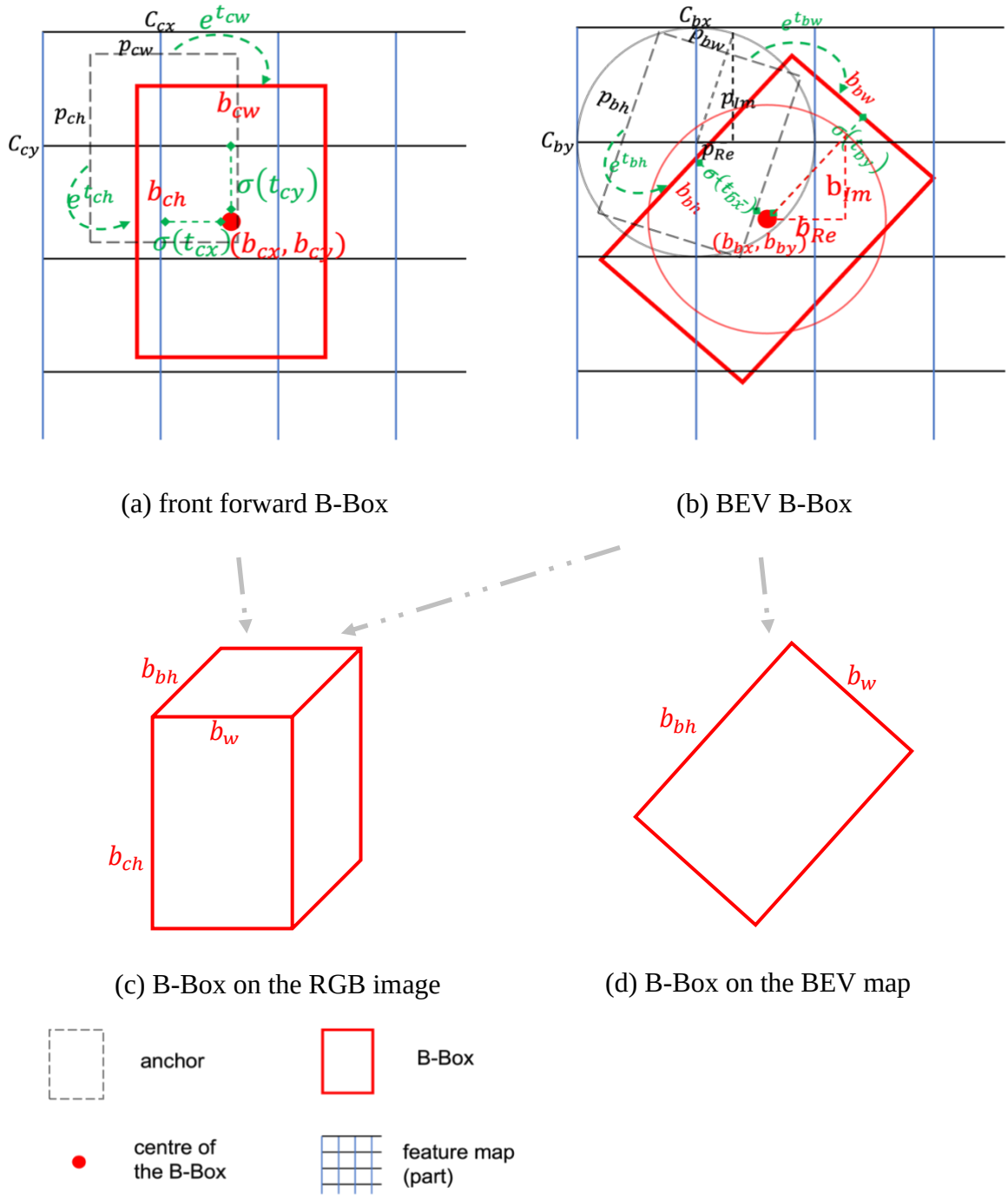
$$b_{cx} = \sigma(t_{cx}) + C_{cx} \quad (3.8)$$

$$b_{cy} = \sigma(t_{cy}) + C_{cy} \quad (3.9)$$

$$b_{cw} = p_{cw} e^{t_{cw}} \quad (3.10)$$

$$b_{ch} = p_{ch} e^{t_{ch}} \quad (3.11)$$

Where (b_{cx}, b_{cy}) represents the centre coordinates of the predicted front forward view B-Box, b_{cw}, b_{ch} represent the width and height of the predicted front forward view B-Box. (C_{cx}, C_{cy}) are the upper left corner coordinates of the grid cell in the feature map, and p_{cw}, p_{ch} are the width and height of the RGB anchor box in the feature map.



(a) front forward B-Box

(b) BEV B-Box

(c) B-Box on the RGB image

(d) B-Box on the BEV map

Figure 3.9 Converting the anchor-based prediction matrix into B-Box. (a) shows the front forward view B-Box matrix, (b) shows the BEV B-Box matrix, (c) shows the 3D B-Box on the output RGB image, (d) shows the BEV B-Box on the output BEV map.

- The BEV B-Box

Different from the B-Box on RGB images, the BEV B-Box might not be parallel to the BEV map's coordinate axes. To predict the rotation of the B-Box, inspired by complex YOLO [87], a Euler representation of the rotation angle is added to the prediction matrix of the proposed model.

Although the computer can directly predict the rotation angle [40], it may result in slight deviation, which will then grow bigger over iterations. Therefore, a linear expression of the rotation angle is more commonly used. Compared with the trigonometric function regression [64][65], Euler angle provides clearer information regarding the direction.

Hence, the BEV prediction matrix contains 6 variables: Aside from the offsets of the B-Box centre coordinate t_{bx}, t_{by}, and the scaling of the width and height t_{bw}, t_{bh}, there are also the scaling of the Euler representation of the rotation angle of the B-Box t_{Im} and t_{Re}.

As shown in Figure 3.9 (b), the predicted BEV B-Box can be calculated:

$$b_{bx} = \sigma(t_{bx}) + C_{bx} \quad (3.12)$$

$$b_{by} = \sigma(t_{by}) + C_{by} \quad (3.13)$$

$$b_{bw} = p_{bw} e^{t_{bw}} \quad (3.14)$$

$$b_{bh} = p_{bh} e^{t_{nh}} \quad (3.15)$$

$$b_{Re} = p_{Re} e^{\sigma(t_{Re})} \quad (3.16)$$

$$b_{Im} = p_{Im} e^{\sigma(t_{Im})} \quad (3.17)$$

Where (b_{bx}, b_{by}) represents the centre coordinates of the predicted BEV B-Box, and b_{bw}, b_{bh} represent the width and height of the predicted BEV B-Box. (C_{bx}, C_{by}) are the upper left corner coordinates of the grid cell in the feature map where the centre of the B-Box lands. p_{bw}, p_{bh} are the BEV width and height of the anchor box in the feature map, and p_{Re}, p_{Im} are the

Euler representation of the anchor rotation. b_{Re} and b_{Im} are the Euler representation of the rotation angle, which can be computed as:

$$\theta = arctan\ \frac{b_{Im}}{b_{Re}} \qquad (3.18)$$

The final detection output will be computed with the prediction matrix and will be shown as a 3D B-Box on RGB images, and as a 2D B-Box on the BEV maps. Examples will be presented in Section 3.4.

3.2.3 Loss Functions

The loss function used in the proposed model consists of a combination of classification loss, B-Box regression loss and confidence loss. Compared to YOLOv3 [1], the regression loss uses Generalized Intersection over Union (GIoU) [4] instead of mean square error (MSE). Moreover, to optimize the performance of the detector, the Euler angle is added to the B-Box regression loss. For the classification loss, focal loss [3] is added to address the imbalance problem in the training dataset.

- Regression Loss

To match with the Euler angle regression network proposed in Section 3.2.2, a combination of GIoU and Euler angle regression is used for the B-Box regression. Compared to the MSE loss function that is commonly used in LiDAR based detectors [40][41][64][73] and YOLOv3 [1], the proposed Euler GIoU loss generates better detection result, which will be discussed in detail in Section 3.4.3.

As explained in Section 2.1.2, the GIoU of the predicted B-Box and ground truth B-Box can be computed as:

$$GIoU = \frac{I}{B^g \cup B^p} - \frac{A^c - (B^g \cup B^p)}{A^c} \qquad (3.19)$$

Where B^g, B^p are the ground truth B-Box and the predicted B-Box respectively. I is the intersection of the ground truth and predicted B-Boxes, and $B^g \cup B^p$ is the union of the two B-

Boxes. A^c represents the smallest convex shape that encloses both B^g and B^p. $A^c - (B^g \cup B^p)$ represents the area that is inside A^c but outside $(B^g \cup B^p)$.

Since the B-Box prediction matrix contains a matrix with 4 variables for the front forward view and a matrix with 6 variables for the BEV. Therefore, the B-Box regression loss is divided into two parts: GIoU loss for the front forward view prediction matrix, and Euler GIoU for the BEV prediction matrix.

The GIoU loss used for the front forward view is covered in Algorithm 2.2 in Section 2.1.2. The coordinates of predicted bounding box can be calculated from the B-Box prediction matrix:

$$x_1^p = b_{cx} - \frac{1}{2} b_{cw} \quad (3.20)$$

$$x_2^p = b_{cx} + \frac{1}{2} b_{cw} \quad (3.21)$$

$$y_1^p = b_{cy} - \frac{1}{2} b_{ch} \quad (3.22)$$

$$y_2^p = b_{cy} + \frac{1}{2} b_{ch} \quad (3.23)$$

And the GIoU loss for the front forward view is represented by $L_{GIoU} = 1 - GIoU$.

As shown on the left side of Figure 3.10, the A^c represents the smallest enclosing convex object for the predicted B-Box and the ground truth B-Box. To simplify the computation and better regress the rotation angle, the A^c used for the Euler GIoU for the BEV prediction matrix is the smallest enclosing rectangle for the two B-Boxes with the rotation angle equals to the ground truth rotation angle. Therefore, it might not be the smallest enclosing convex object for the predicted B-Box and the ground truth B-Box in BEV. The Euler GIoU can be computed by the following Algorithm 3.1.

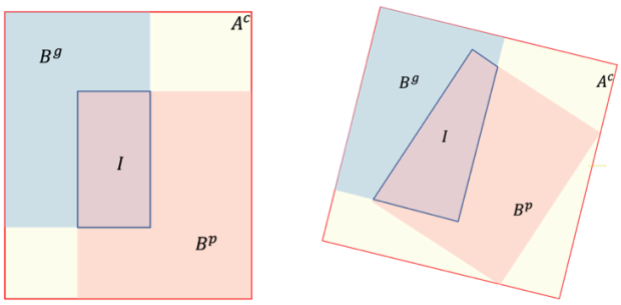


Figure 3.10 GIoU for front forward view (left) and Euler GIoU for the BEV (right).

Algorithm 3.1 Euler GIoU.

input: B^p as the coordinates and rotation angle's Euler representation of predicted B-Box, B^g as the coordinates and rotation angle's Euler representation of ground truth B-Box, where $B^p = (x_1^p, y_1^p, x_2^p, y_2^p, b_{Re}, b_{Im})$, $B^g = (x_1^g, y_1^g, x_2^g, y_2^g, g_{Re}, g_{Im})$

output: Euler GIoU loss L_{GIoU}^E

1. To predict bounding box B^p, we need to ensure $x_2^p > x_1^p$ and $y_2^p > y_1^p$ by computing:

$$\hat{x_1^p} = min(x_1^p, x_2^p),\ \hat{x_2^p} = max(x_1^p, x_2^p),$$
$$\hat{y_1^p} = min(y_1^p, y_2^p),\ \hat{y_2^p} = max(y_1^p, y_2^p)$$

2. calculate the area of predicted bounding box B^p: $A^p = (\hat{x_2^p} - \hat{x_1^p}) \times (\hat{y_2^p} - \hat{y_1^p})$.

3. calculate the area of ground truth bounding box B^g: $A^g = (x_2^g - x_1^g) \times (y_2^g - y_1^g)$.

4. calculate the intersection of B^p and B^g clockwise, get the intersection coordinate matrix:

$$[x^I, y^I] = \begin{bmatrix} x_1^I & y_1^I \\ \vdots & \vdots \\ x_n^I & y_n^I \end{bmatrix}$$

5. calculate the area of aggregated polygon: $I = \sum^{[x^I, y^I]}(x, y)$

6. calculate the area A^c of the enclosing rectangle with a rotation angle that fits:

$$\theta = arctan\frac{Im}{Re}$$

7. calculate intersection over union (IoU):

$$IoU = \frac{I}{U}, \text{ where } U = A^p + A^g - I.$$

8. calculate Euler generalized intersection over union (GIoU): $GIoU_E = IoU - \frac{A^c - U}{A^c}$

9. calculate Euler GIoU loss: $L_{GIoU}^E = 1 - GIoU_E = 1 - IoU + \frac{A^c - U}{A^c}$

• Classification Loss

In YOLOv3 [1], cross entropy loss [89] is used for classification. Ideally, a non-biased training dataset helps the model to learn the features for multi-class object detection and recognition, and cross entropy loss [89] would be suitable. However, the training data provided by KITTI [2] has more than half of the total objects in the class "car" and less than 20% of cyclist, which represents a bias. Inspired by [3], focal loss is used to replace cross entropy loss, which represents a first usage of focal loss on BEV maps to the best of our knowledge. As shown below, the focal loss is based on the cross-entropy loss, with the focal probability $(1 - p_i(c))^{\gamma}$ used as weight for each class. When the prediction is correct, the larger proportion of the predicted class, the smaller its weight gets. In a word, the category with less samples gets more "focus" from the loss function.

$$L_{cla}^{F} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{i,j}^{obj} \sum_{c \in classes} [\hat{p}_c (1 - p_c)^{\gamma} log(p_c) + (1 - \hat{p}_c) \hat{p}_c{}^{\gamma} log(1 - p_c)] \quad (3.24)$$

$$I_{i,j}^{obj} = \begin{cases} 1, & if \ \ object \in the \ j^{th} \ anchor \\ 0, & otherwise \end{cases} \quad (3.25)$$

Where γ is the relaxation parameter. The higher value of γ, the more "focus" will be given to misclassified examples, and the less loss will be propagated from examples. S^2 represents the number of grid cells, which is equal to the size of the feature map. In the proposed experiment, S^2 has three sizes: 19×19, 38×38, 76×76. B represents the B-Box. $I_{i,j}^{obj}$ is a binary value that indicates whether the j^{th} B-Box of the i^{th} grid cell's GIoU value is larger than the GIoU threshold. In other word, the detector considers it as a positive sample. p_c and $\hat{p}_c$ are the ground truth and the prediction classification score for class c.

• Confidence Loss

The confidence loss is used to measure the objectness of the B-Box. The proposed model uses focal loss in the confidence loss, as follows:

$$L_{con} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} \left[\hat{C}_i (1 - C_i)^{\gamma} log(C_i) + \left(1 - \hat{C}_i\right) \hat{C}_i^{\gamma} log(1 - C_i)\right] +$$

$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} \left[\hat{C}_i (1 - C_i)^{\gamma} log(C_i) + \left(1 - \hat{C}_i\right) \hat{C}_i^{\gamma} log(1 - C_i)\right] \quad (3.26)$$

$$I_{i,j}^{obj} = \begin{cases} 1, & if \ \ object \in the \ j^{th} \ anchor \\ 0, & otherwise \end{cases} \quad (3.27)$$

$$I_{i,j}^{noobj} = \begin{cases} 1, & if \ the \ j^{th} \ anchor \ only \ contains \ background \\ 0, & otherwise \end{cases} \quad (3.28)$$

$$\hat{C}_i = \hat{p}_i(c) \times (GIoU + GIoU_E) \quad (3.29)$$

$$C_i = \begin{cases} 1, if \ \ object \in the \ j^{th} \ anchor \\ 0, \ otherwise \end{cases} \quad (3.30)$$

If an object is detected in the B-Box, the confidence loss is $\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathrm{I}_{ij}^{obj} \left[\hat{C}_i (1 - C_i)^{\gamma} \log(C_i) + \left(1 - \hat{C}_i\right) \hat{C}_i^{\gamma} \log(1 - C_i)\right]$, and $\hat{C}_i$ is the confidence score of the j^{th} prediction B-box in i^{th} grid cell and C_i is the ground truth whether the B-Box contains an object.

In a real-life situation, most B-Box do not contain any object. This causes an imbalance problem where the background or negative samples are more frequently detected by the model than the objects or some positive samples. To fix this, the confidence loss is weighted down by a factor λ_{noobj} which if no object is detected in the box (detected background only), the confidence loss will be $\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathrm{I}_{ij}^{noobj} \left[\hat{C}_i (1 - C_i)^{\gamma} \log(C_i) + \left(1 - \hat{C}_i\right) \hat{C}_i^{\gamma} \log(1 - C_i)\right]$, where I_{ij}^{noobj} is the complement of I_{ij}^{obj} and λ_{noobj} weighs the loss down.

In summary, the loss function of the proposed detector combines the two GIoU regression losses (L_{GIoU} on the front forward view and the Euler angle loss L^{E}_{GIoU} on the BEV view), the focal classification loss (L^{F}_{cla}), and the confidence loss (L_{con}), all defined respectively in this chapter:

$$L = \alpha_1 L^{F}_{cla} + \alpha_2 L_{GIoU} + \alpha_3 L^{E}_{GIoU} + \alpha_4 L_{con} \quad (3.31)$$

Where α_1, α_2, α_3, α_4 are the weights for each part of the loss function, which are optimized based on the experiment results.

3.3 Implementation Details

To support reproduction and further testing of the proposed object detection model, the details of the implementation used for experimentation are listed in this section, including the training dataset considered and coding environment. The hyperparameters used in the model are also listed.

3.3.1 Training and Testing Data

The dataset used for the implementation is KITTI [2], and is divided into training: validation: testing = 6: 2: 2, as shown in Table 3.1.

The ROI of the point clouds is within the range of $[-2,1.25] \times [-40,40] \times [0,80]$ meters along the Z, Y, X axis of the LiDAR sensor respectively, as defined in Figure 3.3, and points outside of those boundaries are removed. The ROI is picked based on real life situation, and the statistic of the dataset shows that the ROI used in the experiment covers 99% of the target objects in both the RGB image and its corresponding LiDAR point cloud.

Table 3.1 Size of dataset.

Dataset	Size (pairs of RGB image and LiDAR point cloud)
Training	4481
Validation	1500
Testing	1500

3.3.2 Experimental Environment

The training efficacy may vary with different coding environments. These variations could be the result of hardware differences such as the GPU model and the memory size and software differences such as the version of the packages used for the code.

- Hardware Environment

The experiments run on a single GPU environment provided by Google Colab Pro [85] as detailed in Table 3.2.

Table 3.2 Hardware environment.

Number of CPU	1
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz
CPU core	1
CPU MHz	2199.998
GPU	NVIDIA Tesla V100
number of GPU	1
RAM	25.46GB

- Software Environment

The versions of the python packages used are listed in Table 3.3.

Table 3.3 Version of packages used for coding.

Packages	Version
opencv-python	4.5.1.48
easydict	1.9
matplotlib	3.3.4
mayavi	4.7.4
numpy	1.20.1
scikit_learn	1.1.1
scipy	1.6.2
Shapely	1.8.2
torch	1.11.0
torchsummary	1.5.1
torchvision	0.12.0
tqdm	4.59.0

3.3.3 Hyperparameters

Table 3.4 shows the hyperparameters used for the training and testing for the proposed full detector model. The hyperparameters are chosen based on the hardware environment provided in Section 3.3.2 and test results during the experiment. Note that not all the possible combinations of the following hyperparameters are tested, therefore the results can be further optimized.

Table 3.4 Hyperparameters for the training of the full model.

image width (RGB image and BEV map)	608 pixels
hflip probability	0.5
epochs (number of iterations)	300
activation function	leaky ReLu
batch size	4
cutout ratio	0.3
optimizer	Adam
learning rate type	cosin
learning rate	0.0013
minimum learning rate	1e-07
momentum	0.949
burn in	1000
steps	4000; 4500
weight decay	0.0005
α_1 (weight of classification loss)	37.4
α_2 (weight of GIoU loss)	3.54
α_3 (weight of Euler GIoU loss)	3.54
α_4 (weight of confidence loss)	64.3
λ_{noobj} (Equation (3.19))	100
number of trained parameters	63,959,226

3.4 Experimental Results

The detection and recognition results of the proposed model consist of the classification of the detected object and its 3D B-Box. To visualize the results, the 3D B-Boxes of the detected objects are plotted over the testing RGB images, and the corresponding 2D BEV B-Box are plotted over the BEV maps. The colour coding of the B-Boxes represents the class of the detected objects: yellow for ‘car’, red for ‘pedestrian’ and blue for ‘cyclist’. Examples are given in Figure 3.11.

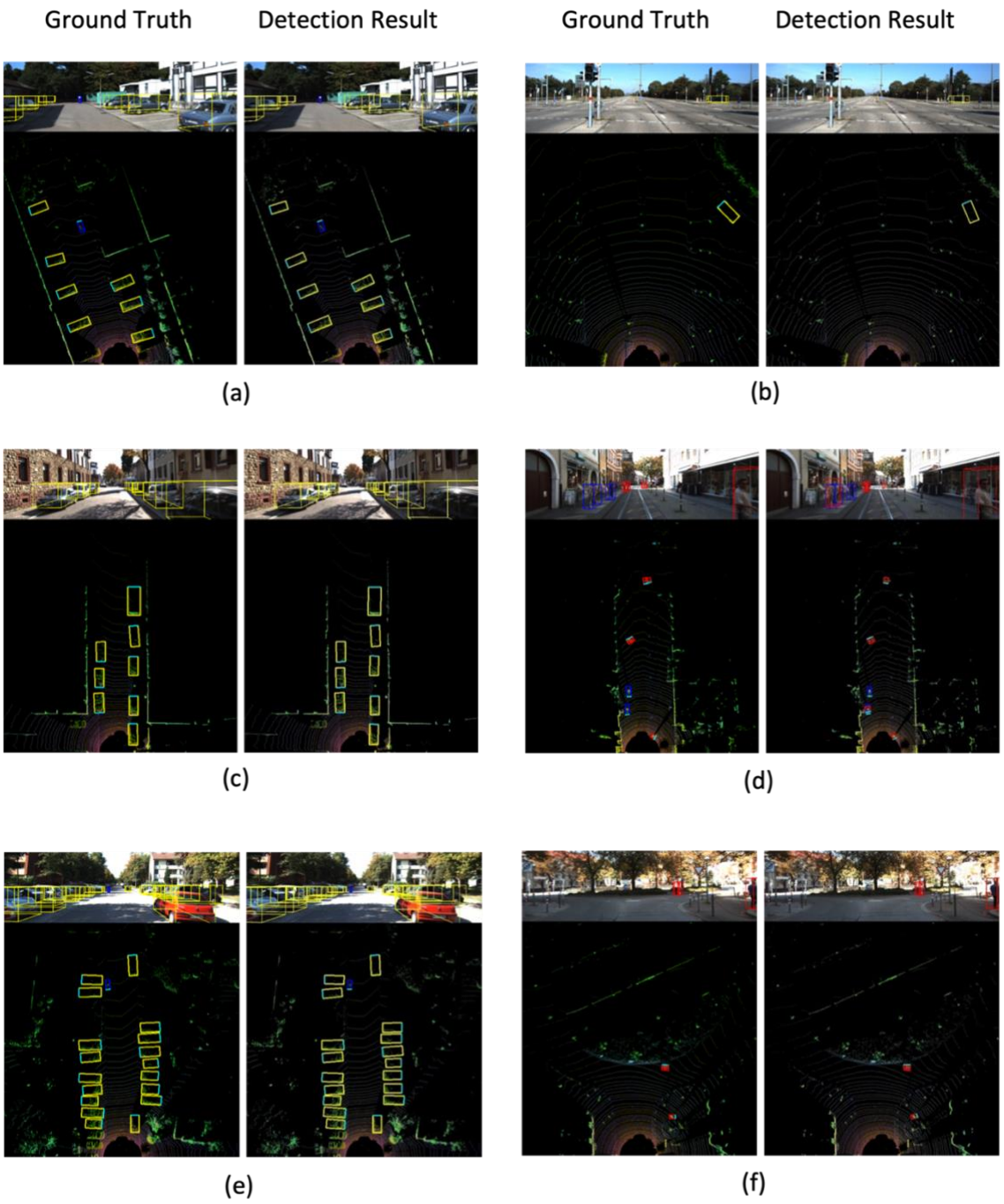


Figure 3.11 Samples of experimental results. Each sample shows the ground truth (left) and the detection result of the proposed model (right). Each output displays a 3D B-Box over the front forward view in the RGB image (upper part of each output) and a 2D B-Box over the BEV map (lower part of each output). The colour of the B-Box represents the different categories of the detected objects: yellow for “car”, red for “pedestrian” and blue for “cyclist”.

Table 3.5 shows the evaluation of the detection result. The metrics appearing in this table are explained in Section 2.1.5. The evaluation is obtained on the testing dataset with 1500 scenes (different pairs of RGB images and LiDAR point clouds provided by KITTI [2]) and the hardware and software environment used for the evaluation is detailed in Table 3.2 and Table 3.3 respectively.

Table 3.5 Detection results on testing dataset.

Class	Precision	Recall	AP	F1	mAP	Speed (FPS)
Car	0.9065	0.9868	0.9794	0.945	0.9026	46.4
Pedestrian	0.6389	0.9317	0.8272	0.758		
Cyclist	0.7951	0.9524	0.9013	0.8667		

The testing result shows that the proposed full model achieves 46.4 FPS detection speed using a single NVIDIA Tesla V100 GPU. The mAP of all target classes is up to 90.26%, with the AP of each class up to 97.94% for “car”, 82.72% for “pedestrian” and 90.13% for “cyclist”. The recall of all classes is higher than 93% and the precision of all classes vary from 63% to 90%, which indicates that among all the non-true positive detections, the number of false positive detection results are higher than the number of the false negative. In other words, compared with missing any target objects, which is a critical problem in autonomous vehicles, the proposed full detector makes more mistakes by generating false alarm. Figure 3.11(d) shows one false negative detection on the top, where a pedestrian (red B-Box) is not detected, and one misclassified B-Box on the left bottom, where a cyclist (blue B-Box) is recognized as a pedestrian (red B-Box). Figure 3.11(f) is an example of false positive results, with one additional pedestrian (red B-Box) being erroneously detected.

In terms of the orientation of the B-Boxes depicted on the BEV maps, most B-Boxes are correct or have invisible variation. In such cases, orientation has little effect on the GIoU (generalized intersection over union) of the predicted B-Boxes and their corresponding ground truth B-Boxes. There are a few cases with slight visible angular variations, as shown in Figure 3.11(b), and may result in false detections if the GIoU is lower than the threshold.

3.4.1 Adaptability to Occlusion and Performance Comparison with Other Research

In practical application scenarios of autonomous driving, the challenges for object detection often come from line-of-sight occlusion. To study the adaptability to occlusion of the proposed detector, we further evaluate the performance on KITTI evaluation metrics [2]. The metrics is provided by KITTI along with the dataset used in the experiments of this thesis and shows the average precision (AP) of the detection results under different occlusion levels.

The PASCAL criteria [90] is used to define the difficulty levels of the testing samples in the KITTI evaluation metrics. The definition of difficulty levels for the B-Box given by KITTI is detailed in Table 3.6. The "easy" detection objects are the ones that got almost no occlusion or truncation (truncation under 15%) and have a B-Box height over 40 pixels. The "moderate" level of difficulty requires the detected object to be at least partly visible and taller than 25 pixels in the front forward view RGB images. The "hard" objects can be truncated at most 30% and are difficult to see. Note that the occlusion level of the testing dataset is annotated by KITTI, therefore hard to describe with numbers or equations.

Table 3.6 The definition of difficulty levels under the PASCAL criteria.

	Min B-Box height (pixels)	Max occlusion level	Max truncation (%)
Easy	40	fully visible	15
Moderate	25	partly occluded	30
Hard	25	difficult to see	50

Aside from defined difficulty levels on the testing dataset, the KITTI evaluation metrics also uses different confidence thresholds for the detection output of different classes. The thresholds are defined as the IoU of the 3D B-Box, as shown in Table 3.7.

Table 3.7 The IoU threshold for the detected objects' classes under KITTI evaluation.

Class	Car	Pedestrian	Cyclist
IoU threshold	70%	50%	50%

The proposed full model is evaluated on the KITTI evaluation metrics, as shown in Table 3.8. The evaluation is compared with other 3D object detectors that use the KITTI evaluation metrics with KITTI LiDAR data as input: MV3D-Net [40], AVOD [41], F-PointNet [51], F-ConvNet [54], Fast Point R-CNN [58], MMF [43], STD [50], VoxelNet [71], SECOND [72], PointPillars [74], SA-SSD [75] and PIXOR [64].

Table 3.8 Comparative evaluation of some existing LiDAR point cloud-based 3D object detection and recognition models and the proposed full detector on KITTI dataset.

	Method	Data	Speed (FPS)	mAP		
				Easy	Moderate	Hard
Transformer	VoTr-SSD [77]	LiDAR	14.65	87.86	78.27	76.93
	VoTr-TSD [77]	LiDAR	7.17	89.04	84.04	78.68
Two stages	MV3D-Net [40]	LiDAR + RGB	2.7	86.49	78.98	72.23
	AVOD [41]	LiDAR + RGB	10	89.74	84.81	78.12
	F-PointNet [51]	LiDAR + RGB	5.7	91.16	84.61	74.77
	F-ConvNet [54]	LiDAR + RGB	1.9	91.44	85.84	76.11
	Fast Point R-CNN [58]	LiDAR	15.3	90.87	**87.71**	80.51
	MMF [43]	LiDAR + RGB	12.2	86.81	76.75	68.41
	STD [50]	LiDAR	10	89.93	86.20	79.42
Single stage	VoxelNet [71]	LiDAR	4.2	87.95	78.39	71.29
	SECOND [72]	LiDAR	19.7	89.33	82.87	78.51
	PointPillars [74]	LiDAR	41.9	90.07	86.56	**82.81**
	SA-SSD [75]	LiDAR	24.4	88.75	79.79	74.16
	PIXOR [64]	LiDAR + RGB	2.8	86.78	80.75	76.77
	Proposed: full detector	LiDAR + RGB	**46.4**	**94.71**	87.33	81.52

Among the experimental results listed in Table 3.8, MV3D-Net [40], AVOD [41], VoxelNet [71] PIXOR [64] and the proposed full detector are tested in the same hardware environment (as described in Section 3.3.2). The proposed full detector shows the highest detection speed among them (over 4 times faster than AVOD [41]), while maintaining a high detection accuracy.

The detection speed will be discussed further in Section 4.3 in relation with a more compact detector model that is also proposed in this thesis.

According to the statistics, on cases annotated as “easy”, the proposed full detector performs best among the listed detectors in Table 3.8, leading the second best by over 3%. As shown in Figure 3.12, the detection results of the proposed full detector on “easy” scenes are precise.

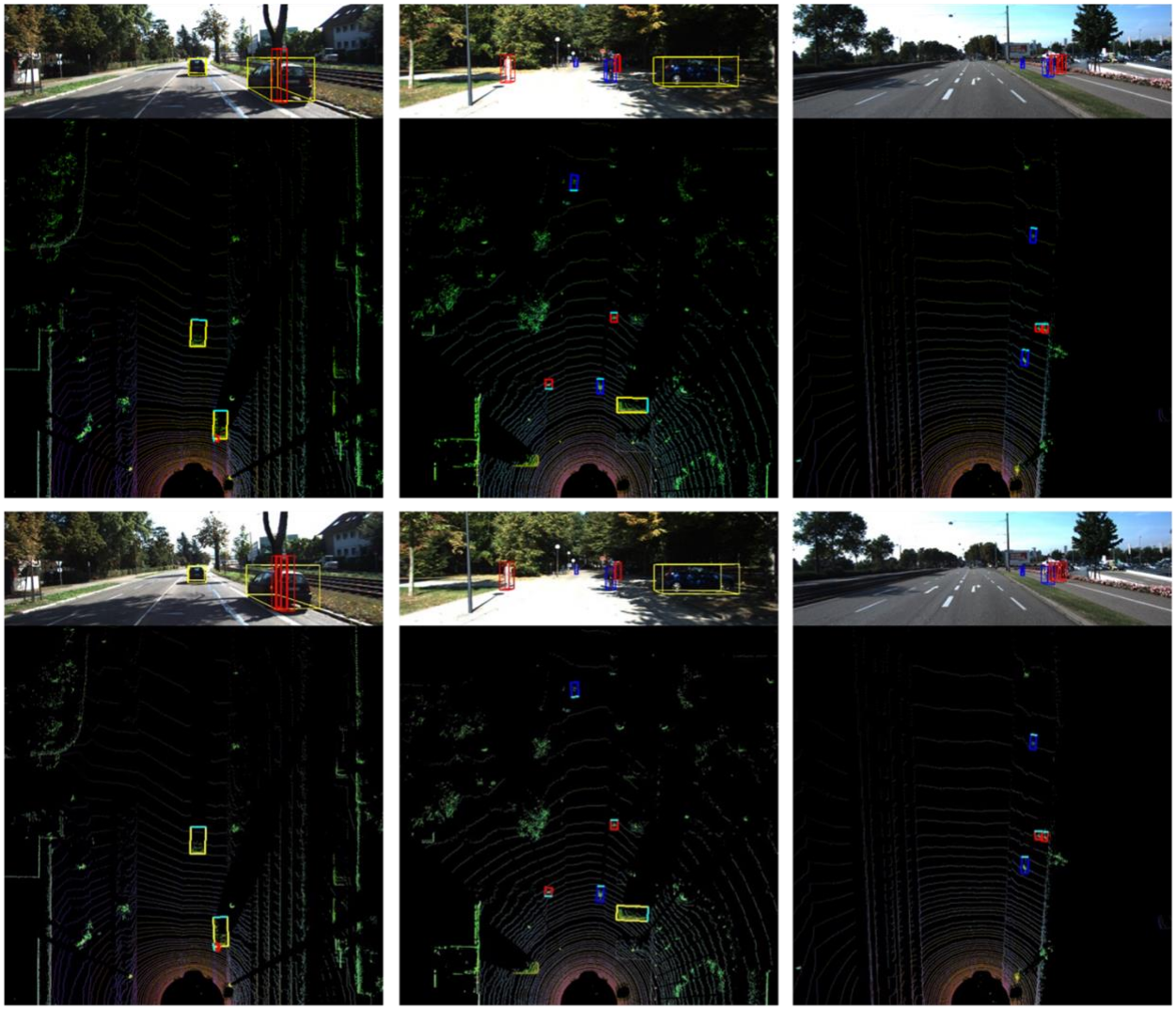


Figure 3.12 Samples of "easy" scenes in KITTI testing dataset. In each pair of samples, the top ones show ground truth, and the bottom ones show detection results of the proposed full model.

Compared with other detectors that use both the LiDAR point cloud and the RGB image, such as MV3D-Net [40], AVOD [41], MMF [43], F-PointNet [51], F-ConvNet [54] and PIXOR [64], the proposed full detector shows higher mAP on moderate and hard samples. As shown in Table 3.8, some models that use only a LiDAR point cloud as input show better detection result on hard tasks compared with models that combine a LiDAR point cloud and an RGB image. Although combining with RGB images weakens the model's ability to adapt to occlusion compared with using only a LiDAR point cloud as input, the proposed full model demonstrates good detection results on testing scenes with occlusion, as shown in Figure 3.13 and 3.14.

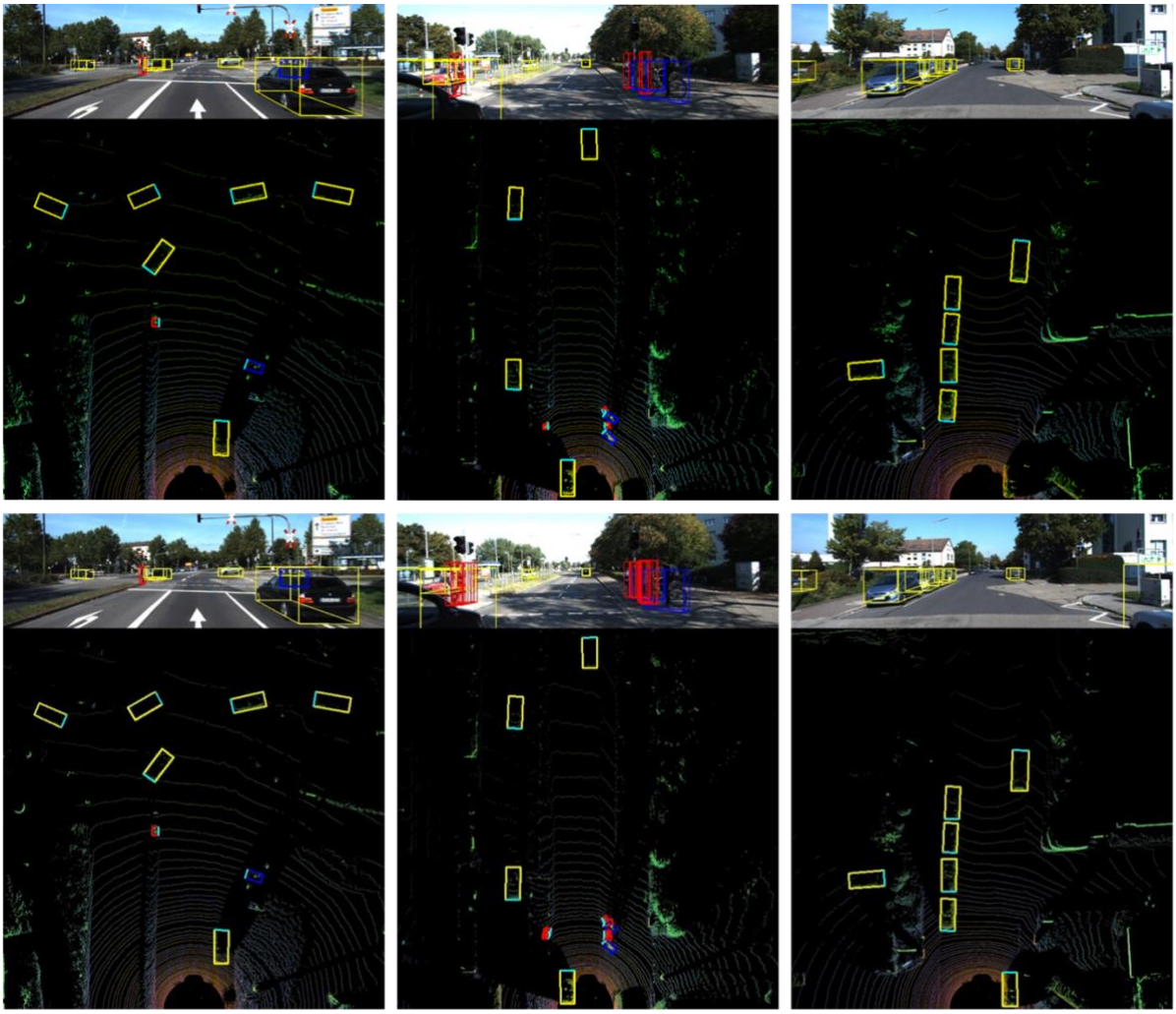


Figure 3.13 Samples of "moderate" scenes in KITTI testing dataset. In each pair of samples, the left one shows ground truth and the right one shows detection results of the proposed full model.

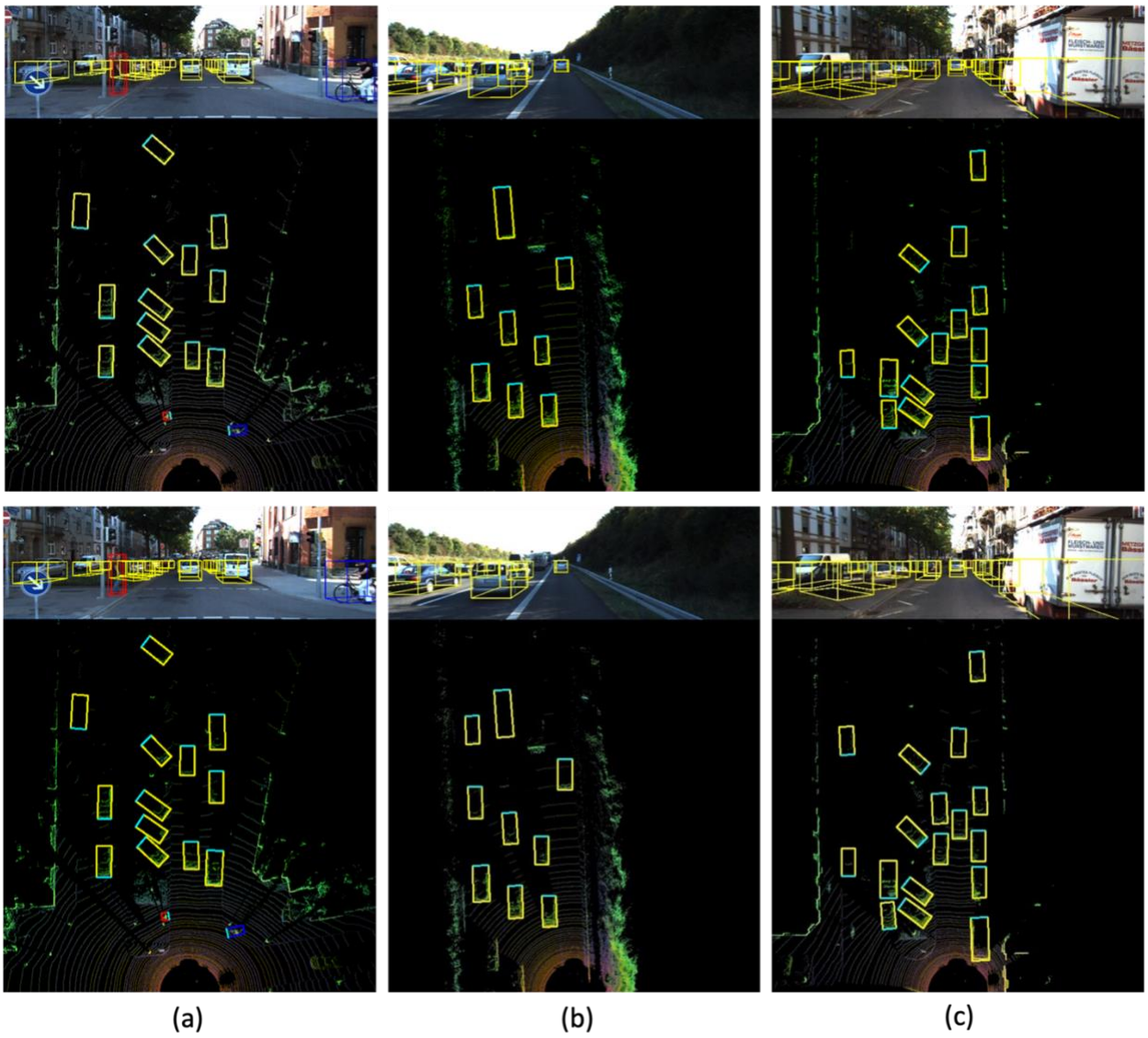


Figure 3.14 Samples of "hard" scenes in KITTI testing dataset. In each pair of samples, the left one shows ground truth and the right one shows detection results of the proposed full model.

Additional samples of experimental results with different difficulty levels are presented in Appendix A.

3.4.2 Model Optimization using Different Loss and Hyperparameters

• Performance Evaluation Comparison with Different GIoU Threshold

The GIoU threshold affects the calculation of true positive (TP): the higher GIoU threshold, the higher accuracy of the B-Box and the lower TP detection. Therefore, the GIoU affects the detection results. As shown in Figure 3.15, any GIoU threshold lower than 0.7 gives a good and stable result on “car” detection. For “pedestrian” and “cyclist” categories, when the GIoU threshold is higher than 0.5, the AP drops, and therefore the mAP over all three classes also drops.

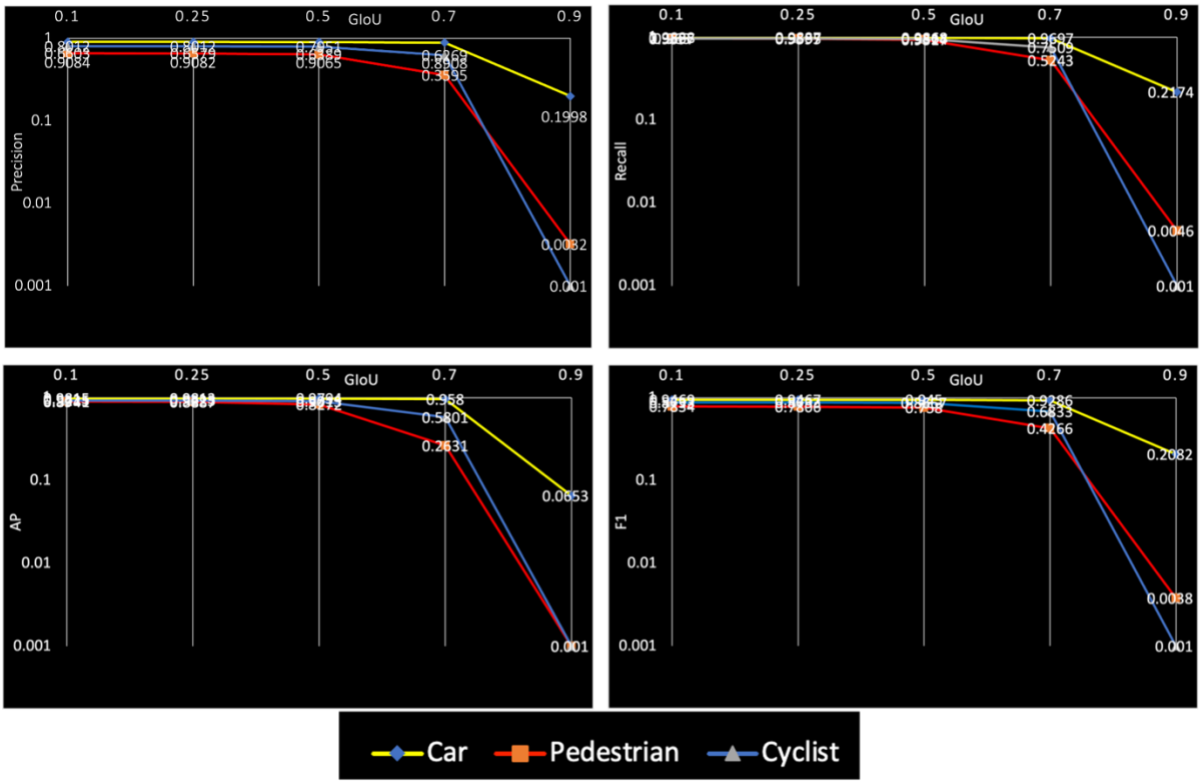


Figure 3.15 The precision, recall, AP, F1 evaluation of the proposed full detector model under different GIoU thresholds.

Compared to other 3D detectors, as shown in Figure 3.16, the proposed model offers relatively good robustness to variable threshold values. This indicates that the GIoU of B-Boxes predicted

by the proposed model is lower than that achieved by the rest of the listed detectors in Figure 3.16.

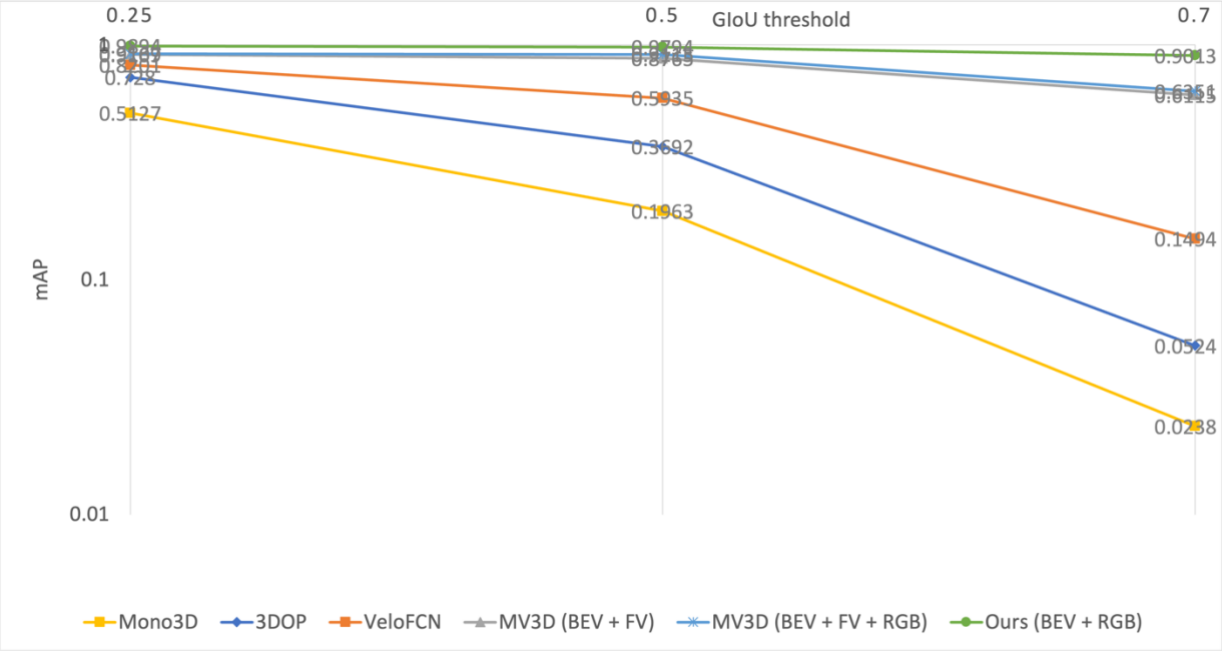


Figure 3.16 Comparison of mAP of different 3D detection and recognition models with different GIoU/IoU thresholds tested on KITTI dataset.

• Model Optimization using Different Regression Loss

The proposed detector merges a region proposal network (RPN) and Euler angle regression to the DarkNet-53 backbone to improve the detection accuracy and uses Euler angle regression and GIoU to optimize the training. To reduce the bias caused by the imbalance in the number of samples in each class of the dataset, focal loss [3] is also used as the classification loss and the confidence loss. To evaluate if these methods improve the performance of the proposed model, ablation studies are designed to test the performance of different regression loss and to verify that the proposed Euler angle regression does contribute to increase the detection accuracy of the proposed model. All three experiments are trained and tested in the same experimental environment as described in Section 3.3.2. The testing dataset contains 1500 pairs of the LiDAR point cloud and its corresponding RGB image. The results are listed in Table 3.9.

Table 3.9 The effect of different regression loss on the detection results.

Classification Loss	Regression Loss	Confidence Loss	AP			mAP
			Car	Pedestrian	Cyclist	
focal	MSE	focal	0.9511	0.5393	0.6258	0.7056
focal	GIoU	focal	0.9678	0.6461	0.8383	0.8174
focal	MSE + Euler	focal	0.9688	0.7848	0.9096	0.8877
focal	GIoU + Euler	focal	0.9794	0.8272	0.9013	0.9026

For both the experimental groups and the comparison control group, focal loss is used for classification and confidence, as explained in detail in Section 3.2.3. For the comparison control group, MSE is used as the regression loss on the proposed model without Euler angle regression network.

In the first experimental group, the GIoU loss is tested as the regression loss instead of MSE. Both groups are trained in the same software environment and tested on the same testing dataset. Results show that the GIoU regression loss achieves better performance on the proposed model, especially on the “cyclist” category, which has fewer samples in the training dataset compared to “car”.

In the second experimental group, the Euler angle regression detection head is added on the model used for the comparison control group. The MSE loss is used to regress all 10 variables used to calculate the prediction B-Box (4 for the front forward view RGB image and 6 for the BEV map). As shown in Table 3.9, the Euler angle regression network increases the average precision (AP) for all classes.

In the third experimental group, aside from adding the Euler angle to the regression network, GIoU loss is used instead of the MSE regression loss. As shown in Table 3.9, compared to the first experimental group which does not have the Euler angle regression network but uses GIoU loss for regression, adding the Euler angle regression network further increases the performance of the proposed full detector. Compared to the second experimental group which has the Euler angle regression network but uses the MSE regression loss, the combination of GIoU loss and Euler angle regression gives higher average precision in “car” and “pedestrian”, and similar AP

in “cyclist”. Compared with the comparison control group, the mAP of the third experimental group increased by 0.2.

Overall, the ablation studies with different regression loss show that the Euler angle regression network increases the detection accuracy of the proposed full object detector model. Compared to MSE, GIoU loss also shows better performance on the regression of the proposed model.

- Model Optimization using Different Classification Loss

Aside from the ablation studies designed to test the performance of different regression loss and to estimate how much the proposed Euler angle regression increases the detection accuracy of the proposed model, another ablation study is designed to test the focal loss’ ability to balance the results of different classes on the biased data. This experiment is conducted in the environment described in Section 3.3.2 and on the testing dataset containing 1500 pairs of the LiDAR point cloud and its corresponding RGB image.

As shown in Table 3.10, compared to using cross entropy as the classification loss, although focal loss decreases the average precision (AP) of “car”, it highly increases the AP of “pedestrian” and “cyclist”, hence it contributes to better balance the AP over all the target classes. As a result, the mAP achieved over all classes increases from 0.89 to 0.90.

Table 3.10 The effect of different focal loss on the detection results.

Classification Loss	Regression Loss	Confidence Loss	AP			mAP
			Car	Pedestrian	Cyclist	
cross entropy	GIoU + Euler	focal	0.9803	0.7991	0.8835	0.8876
focal	GIoU + Euler	focal	0.9794	0.8272	0.9013	0.9026

3.4.3 Masked BEV map

Another experiment conducted in the course of this thesis consisted in superimposing a mask (empty rectangle) over the BEV map to exclude some part of the information available to the detector. The experiment was designed for two purposes: one is to see whether the proposed model can handle damaged data, another is to see whether one part of the input is more "valuable" than the other (i.e., if the BEV map matters more than the RGB image to the detector).

For the first objective, experiments demonstrated that the proposed detector remains functional with damaged data, with a mAP being lowered less than 5%. For the second purpose, experiments did not lead to convincing conclusions because the training process was performed while providing the architecture with both parts of the input (BEV map and RGB image), which means the parameters were optimized for both parts instead of any one of the specific inputs. Full retraining of the model with either one of the inputs was not conducted due to resources and time constraints.

That being said, we can still observe that in most cases, if a target object is fully hidden or missing in data available from the BEV map, the proposed detector will miss the target object, as shown in Figure 3.17, reaching to the conclusion that the detector cannot operate reliably from only the RGB image input. Conversely, if a randomly placed mask partially covers a target object, as shown in Figure 3.18, it will in general not affect significantly the detection and recognition result, provided that partial information about the target object is available in the BEV map. Finally, as expected, when no target object is masked (i.e., the mask is placed over a background area), then the detection result will not be affected, as shown in Figure 3.19.

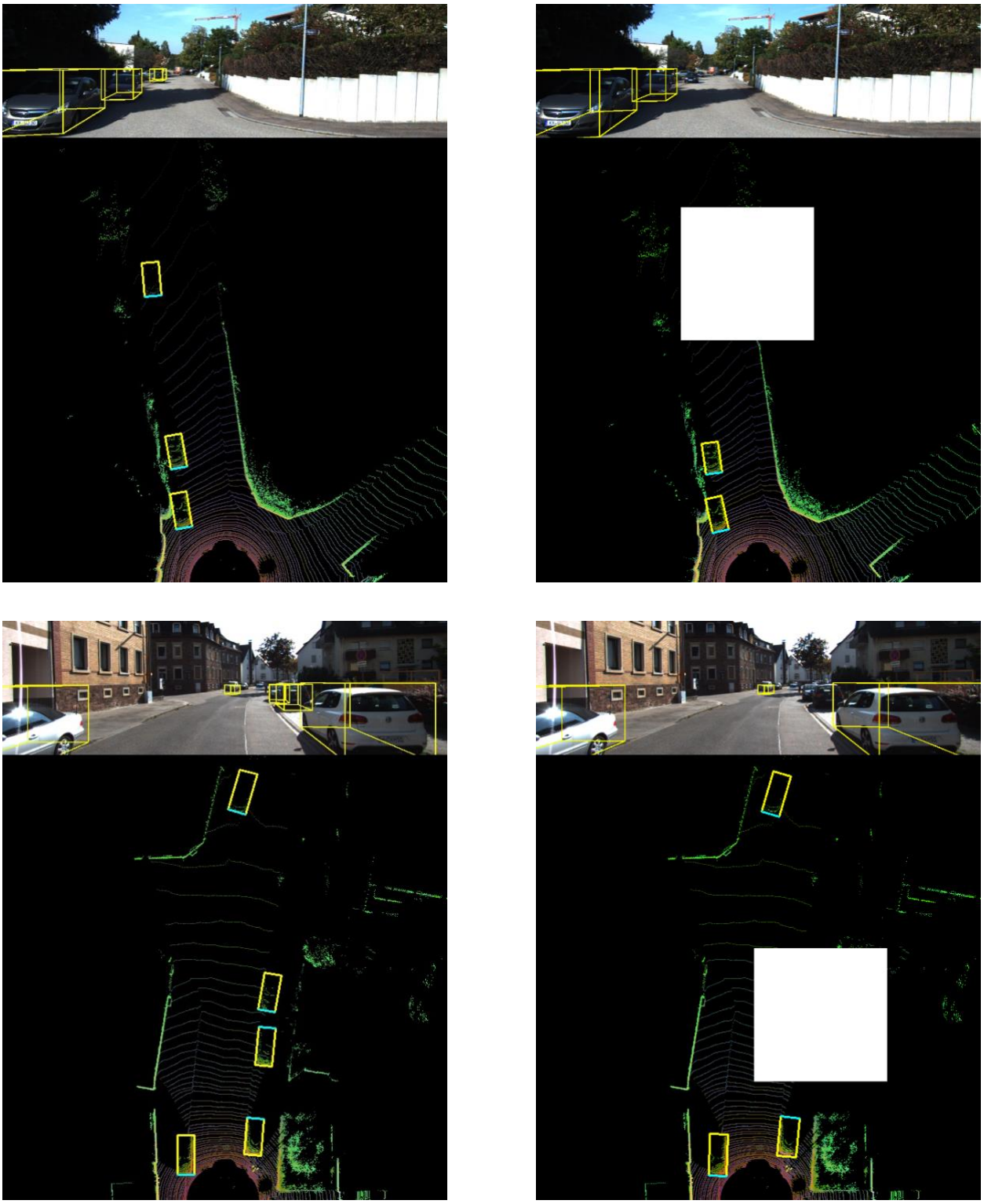


Figure 3.17 Samples with fully masked target objects (left: without mask; right: with mask).

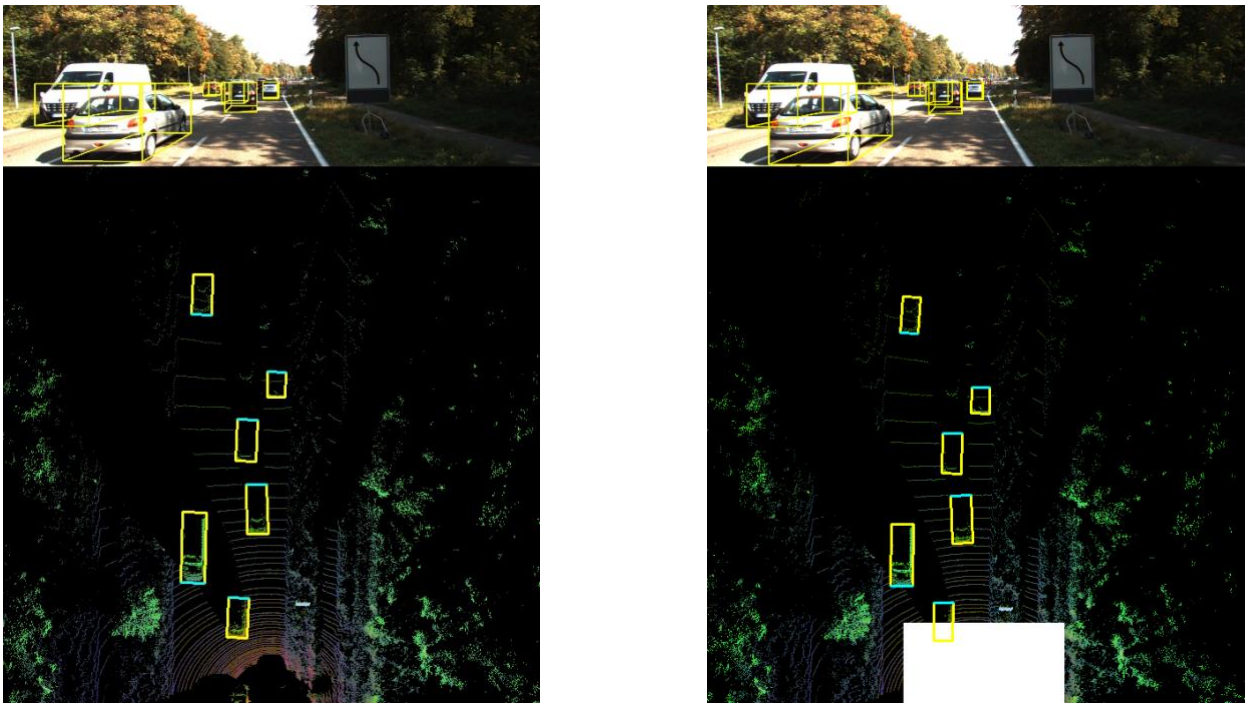


Figure 3.18 Sample with partially covered target objects (left: without mask; right: with mask).

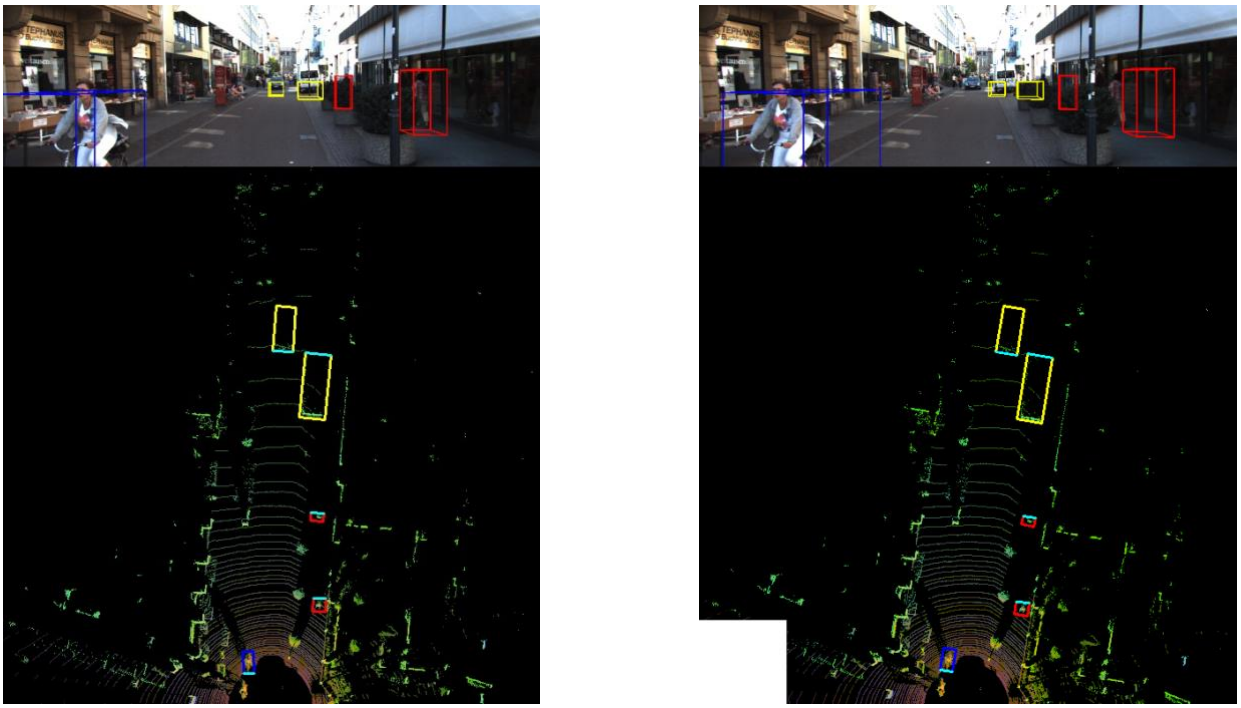


Figure 3.19 Sample with mask over the background (left: without mask; right: with mask).

3.5 Summary

This chapter presented the proposed methodology for combined LiDAR point cloud and RGB image-based 3D object detection and recognition and tested the performance of the proposed full detector model.

A pre-selected ROI over the LiDAR point cloud is preprocessed into a BEV map through coordinate system transformation. During the preprocessing, height thresholding is used to emphasize the height range of the target objects, which simplifies further the computation during the training of the proposed detector. The point cloud converted into a BEV map provides the bird's eye viewpoint of the area over which objects are to be detected, while the corresponding RGB image provides the front forward view of the same scenes to the proposed detector. The two sets of data are used as inputs for the experiments conducted in this thesis.

For the methodology of the proposed 3D detector, the DarkNet-53 architecture [1] is modified and used as the backbone. Euler angle regression loss is added to the detection head, which calculates the rotation angle of the B-Box. In addition, GIoU is used to replace the MSE loss for regression, and focal loss is used to balance the detection accuracy of different target classes caused by the biased training data.

The implementation details of the proposed detector are provided along with the experimental results. To further evaluate the performance, several ablation studies are designed and tested on the proposed detector. Results are compared to other related work and the proposed full detector shows stable results under different regression thresholds for the B-Box. In addition to that, the ablation studies with different regression loss demonstrates the advantage of the proposed improvements on the model, both from the inclusion of the Euler angle regression and the GIoU loss. Another ablation study shows that the focal loss increases the detection mAP by balancing the weight of different target classes from the biased training data. Overall, the proposed detector exhibits robustness to occlusions and achieves a mAP over 90% with a detection speed up to 46.4 FPS.

Chapter 4 Mini 3D Object Detector

In the previous chapter, a 3D detector (full detector) is proposed to perform 3D object detection and recognition tasks. The full detector uses RGB images and bird's eye view (BEV) maps generated from LiDAR point clouds as inputs, which are loaded into a Darknet-53 [1] based CNN for feature extraction and a detection head merged region proposal with Euler angle regression. The experiments demonstrate that the proposed full detector can perform well on 3D object detection and recognition tasks in the context of self-driving vehicles and offers some adaptability to real-life occlusion conditions.

With the continuous development of deep convolutional neural networks and the wide range of applications in this field, in addition to pursuing always higher accuracy, researchers also want to achieve higher detection speed. Ideally, a detector should be "compact", that is, the memory required, and the amount of calculation involved in the detection must be limited as the availability of advanced hardware support is low for applications such as autonomous vehicles. These constraints motivate the development of deep convolutional neural networks that are better suited for widespread deployment on embedded devices. Therefore, although the detection speed of the full detector proposed in Chapter 3 reaches up to 46.4 FPS, which is 15 times faster than PIXOR [64] and 3 times faster than MMF [43], we still want to explore the possibility of designing a lightweight network that uses fewer feature matrices to execute the same 3D object detection and recognition tasks.

This chapter proposes such a mini detector that merges the structure of tiny-YOLO [91] and the proposed full detector to generate feature maps at 2 different scales. The tiny-YOLO architecture is selected to substitute the DarkNet-53 backbone considered in the full detector, due its compactness and recognized performance. The mini detector's detection speed can reach up to over 3 times that of the proposed full detector, therefore is more adaptable on mobile devices, while finding a compromise on detection performance.

In Section 4.1, the motivation for designing the mini detector is briefly introduced. Section 4.2 details the architecture of the proposed mini detector and Section 4.3 lists the implementation

details and experimental results. A performance comparison between the two proposed 3D detectors is included in the results analysis.

4.1 Motivation

The need for high performance computing hardware poses a significant inconvenience for autonomous driving vehicles which need to carry and power these systems. Hence, it is beneficial to explore new approaches to make the detector more efficient, that is, use less computing power for the same detection speed, and/or be able to process more images with the same hardware. Converting the detection model into a lightweight version is increasingly popular for 2D object detection, but still lacks exploration and experimentation in the context of 3D object detection and recognition.

Inspired by lightweight convolutional neural network models such as SqueezeDet [92], Tiny SSD [93], Tiny-YOLO [91] which are designed based on commonly used backbones such as R-CNN [20], SSD [32] and DarkNet-53 [1], an idea for modifying the full detector proposed in Chapter 3 to make it more "compact" came up. The goal of the modification is to reduce the usage of memory and computation, while still being able to perform 3D object detection and recognition tasks using combined LiDAR point clouds and RGB images.

4.2 Model Architecture of the Mini Detector

The proposed mini detector follows a similar architecture as that of the full detector proposed in Chapter 3. The only differences are the backbone and FPN shown in Figure 4.1(b) and (c), which will be explained in detail.

The mini detector uses the same input and same training dataset as the proposed full detector. After converting the LiDAR point cloud into the BEV map and rescaling the corresponding RGB image as input, the mini backbone is used to extract feature.

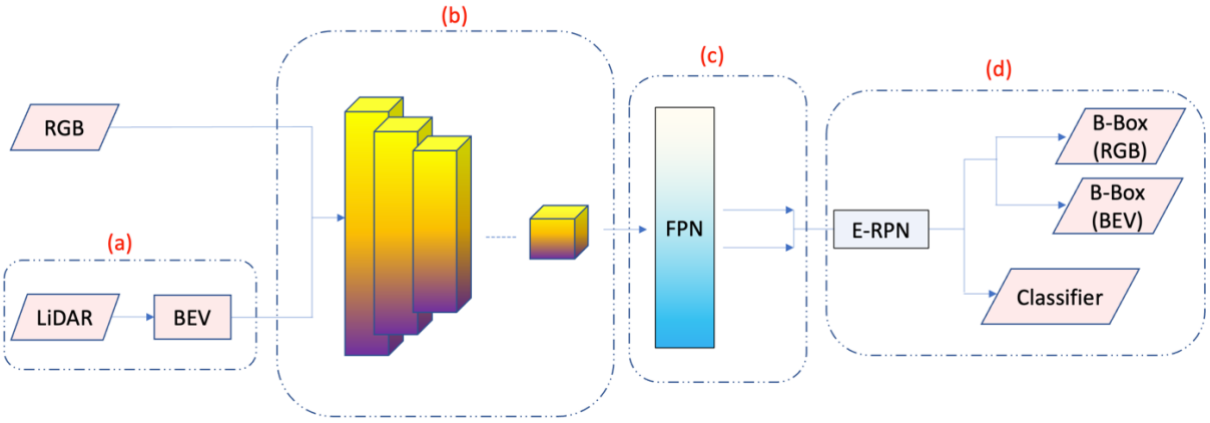


Figure 4.1 Architecture of the mini detector. (a) point cloud preprocessing converts LiDAR point cloud to BEV map, (b) mini backbone convolutional neural network to generate feature maps, (c) feature pyramid network (FPN) for integrating features with two different scales of feature maps generated from FPN, (d) detection head with Euler angle regression.

The backbone of the proposed mini detector is shown in Figure 4.2 (a). Compared to the 53-layer backbone of the proposed full detector, the mini detector's backbone only has 19 layers, that is about $\frac{1}{3}$ the depth of the full detector.

Same as for the proposed full detector, the backbone of the mini detector mainly relies on CBL units to extract the features of the input through convolution operation and enriches feature maps by multiple iterations and regressions. Since the goal for the mini detector is to be computationally lighter and to minimize the processing time, the use of convolutional layers is reduced. To compensate for the reduction in convolution operations, pooling layers are added to operate down sampling during the training and detection processes.

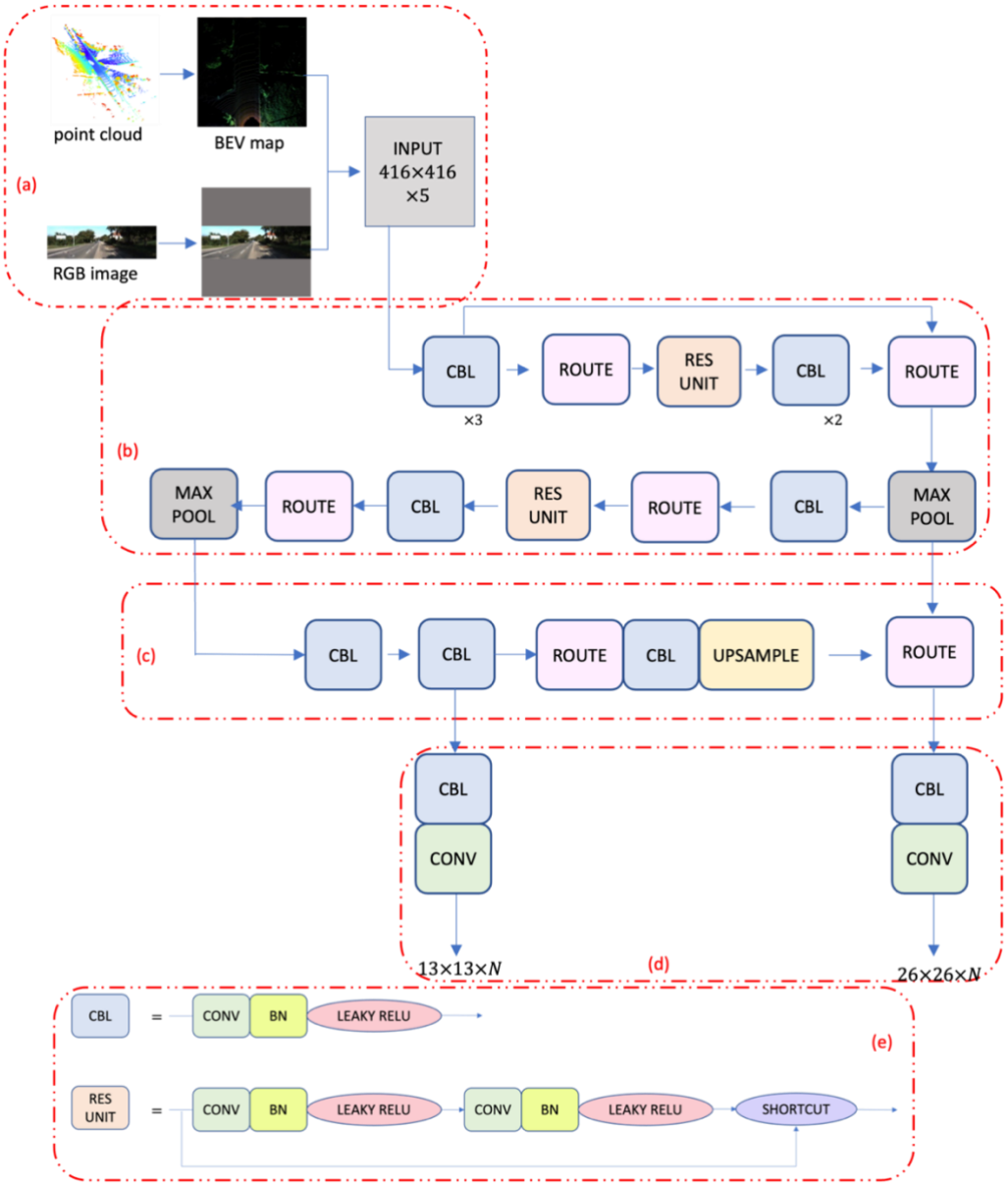


Figure 4.2 Model structure: (a) image preprocessing converts point clouds into BEV maps and rescales the corresponding RGB image, (b) shows the structure of the backbone of proposed mini detector, (c) shows the mini FPN, (d) shows the detection head, (e) shows the structure of CBL and Res Unit used in the structure of the mini detector.

Pooling results in reducing the number of features and parameters, but the benefit of pooling layers goes beyond this. By choosing the suitable method to proceed down sampling, pooling

layers can prevent over-fitting while retaining the main features. Pooling layers can also maintain a certain invariance, including to rotation, translation, expansion, etc., which is needed when designing the network based on the number of parameters and the size of feature maps. Here in the backbone of the proposed mini detector, max pooling layers are used to replace CBLs, therefore leading to a simplified model.

In the proposed full detector, FPN is used to generate feature maps at three different scales (see Figure 3.5). As shown in Figure 4.2 (c), for the mini detector only two different scales of feature maps are generated and passed to the detection head in Figure 4.2 (d) to reduce the calculation and minimize the depth of the mini detector model.

The detection head then converts the feature maps into prediction result. The detection head uses the same design as the proposed full model and the same loss function for training. However, the output of the mini detector only contains two scales instead of three in the full detector.

Note that since the detection head preserves the same structure, the prediction matrix has the same number of variables, therefore in Figure 4.2 (d), N = 14. Detail of the calculation is explained in Section 3.2.2.

4.3 Experimental Details and Results

Section 4.3.1 explains the modification made on the implementation details for the mini detector. Section 4.3.2 provides detailed results analysis and performance comparison between the full detector and the mini detector. The results will be discussed in Section 4.3.3. To better evaluate the performance of the two proposed detectors, testing experiments with different hardware support is also discussed in Section 4.3.4.

4.3.1 Implementation Details

The mini detector uses the same experimental environment as the full detector, detailed in Section 3.3. As for the hyperparameters, a few changes are made to improve the performance of the mini detector, as shown in Table 4.1.

Table 4.1 Hyperparameters of the mini detector.

image width (RGB image and BEV map)	416 pixels
hflip probability	0.5
epochs (number of iterations)	300
activation function	Leaky ReLU
batch size	4
cutout ratio	0.3
optimizer	Adam
learning rate type	cosin
learning rate	0.0026
minimum learning rate	1e-07
momentum	0.949
burn in	1000
steps	4000; 4500
weight decay	0.0005
α_1 (weight of classification loss)	37.4
α_2 (weight of GIoU loss)	3.54
α_3 (weight of Euler GIoU loss)	3.54
α_4 (weight of confidence loss)	64.3
λ_{noobj}	100
number of trained parameters	7,191,283

The main modification relates to the size of the input RGB image and BEV map, that are rescaled from 608×608 to 416×416 pixels, while keeping the aspect ratio unchanged. The purpose of reducing the sizes of the input BEV maps and RGB images is to reduce memory usage and computation, while fitting better with the scales of the feature maps generated from the FPN.

The number of trained parameters is decreased from 63,959,226 to 7,191,283 due to the changes in the model's structure.

4.3.2 Results and Analysis

For a fair comparison of performance of the two proposed detectors, the mini detector is trained and tested in the same software and hardware environment as the full detector. Moreover, the training and testing dataset remains the same.

Table 4.2 shows the detection and recognition results of both the mini detector and the full detector on 1500 pairs of the LiDAR point cloud and the corresponding RGB image. The evaluation metrics used in this table are explained in Section 2.1.5.

Table 4.2 Detection speed, precision, recall, AP and F1 estimated on each class and mAP of all three target classes for the mini detector compared with the full detector.

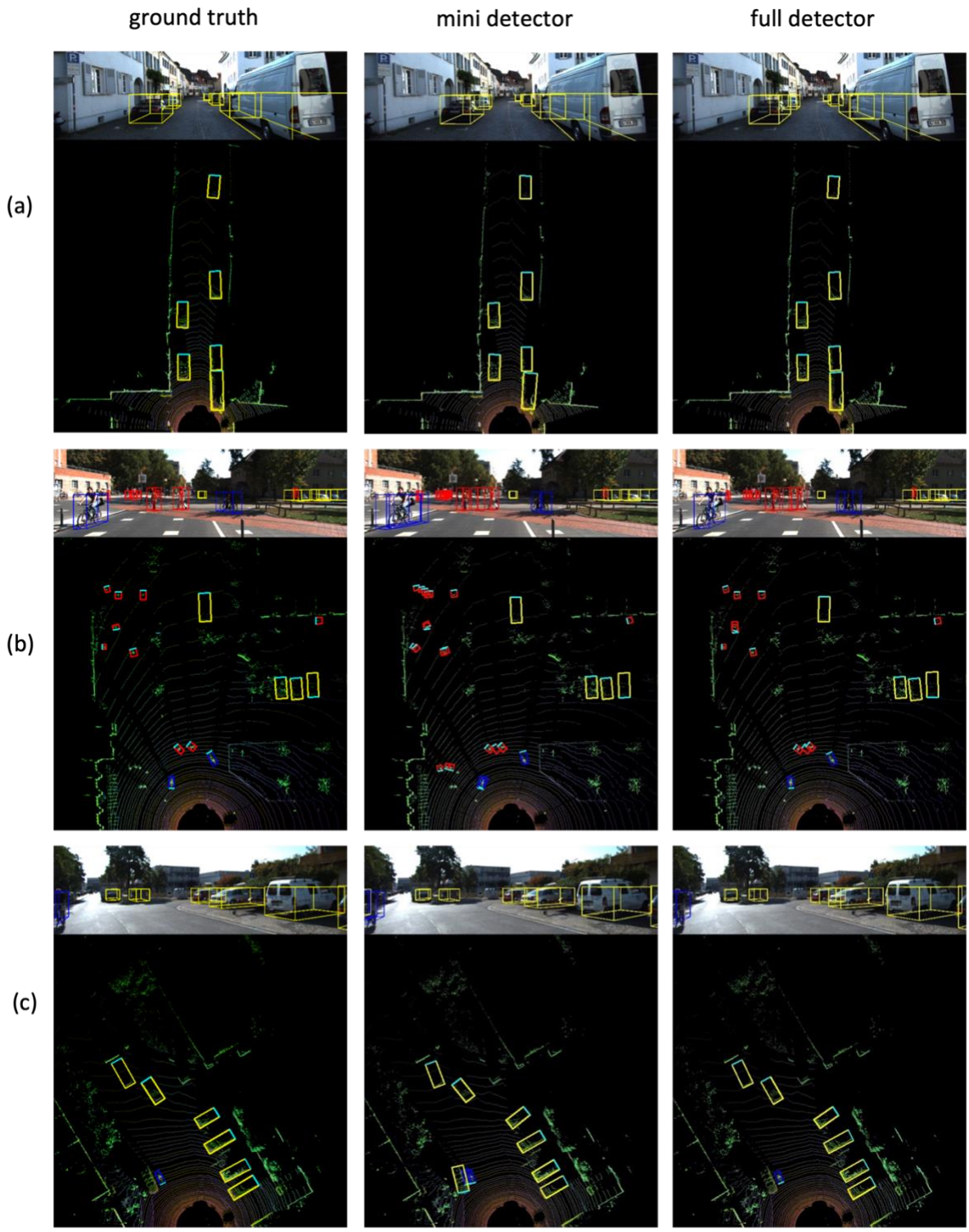
	Class	Mini detector	Full detector
Speed (FPS)		158.97	46.4
Precision	Car	0.8922	0.9065
	Pedestrian	0.4783	0.6389
	Cyclist	0.6874	0.7951
Recall	Car	0.9572	0.9868
	Pedestrian	0.6231	0.9317
	Cyclist	0.8772	0.9524
AP	Car	0.9371	0.9794
	Pedestrian	0.6908	0.8272
	Cyclist	0.8544	0.9013
F1	Car	0.9247	0.945
	Pedestrian	0.3838	0.758
	Cyclist	0.7832	0.8667
mAP		0.8274	0.9026

As shown in Table 4.2, the mini detector achieves a good performance on detecting cars with AP higher than 0.9, but relatively poor result on detecting “pedestrian” and “cyclist” targets. As mentioned in Chapter 3, the training dataset used for the proposed models is imbalanced, with no more than 20% of positive samples for the pedestrian and cyclist classes. Although FPN and focal loss [3] is used to reduce the impact of the data imbalance, with fewer layers and less features extracted in the mini model, testing results suffer more from the data imbalance than the full detector.

The speed of the mini detector is 3.4 times faster than the full detector when testing in the same environment, while a 7.5% reduction of the mAP is observed.

Figure 4.3 shows some testing samples with ground truth, detection and recognition results of the mini detector and compares with the results of the full detector. Each result image includes the

RGB front forward view (top) with 3D B-Boxes and the BEV map generated from the LiDAR point cloud with 2D bounding boxes. For the bounding boxes, yellow boxes represent cars, red boxes represent pedestrian and blue boxes represent cyclists.



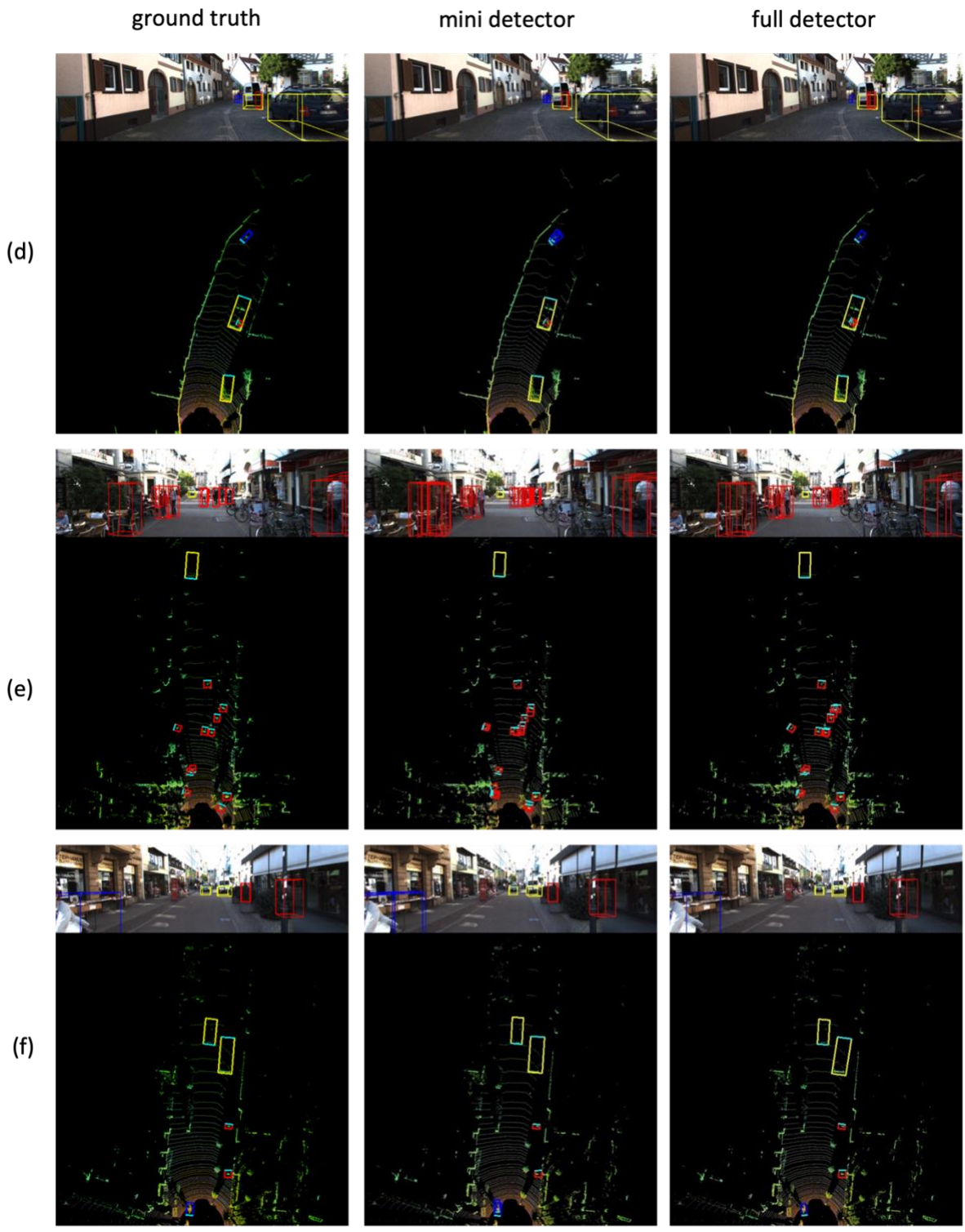


Figure 4.3 Comparison of ground truth (left), results of the mini detector (middle), and results of the full detector (right).

As observed in Figure 4.3 (b) and (e), with false positive “pedestrian” B-Boxes (in red), and in Figure 4.3 (d), a false positive “cyclist” B-Box (in blue), false positive detections in “pedestrian” and “cyclist” classes are the main cause of the mini detector’s relatively low mAP in comparison with the performance of the full detector. Another type of false positive detections exists in scenes such as Figure 4.3 (c), where a target object (yellow B-Box) appears only in the BEV map but not in the front forward view RGB image. In such a case, the mini detector will still recognize the target object, which indicates that the mini detector relies more on features in the BEV map compared to the full detector.

Although the orientation of the predicted B-Boxes of the mini detector is slightly worse than with the full detector, as shown in Figure 4.3(a) and (c), most of them are still acceptable.

Additional samples of experimental results using the proposed mini detector are presented in Appendix B.

Overall, the proposed mini detector performs fairly well in spite of its reduced size. The mini detector has a 3.4 times acceleration of the processing compared with the full detector, without exaggerated detrimental impact on performance. This experiment demonstrates that a trade-off is possible for applications where framerate is critical.

4.3.3 Efficacy Evaluation of the Mini Detector

As shown in Table 4.2, compared to the full detector, the mini detector’s detection speed reaches up to 158.97 FPS, more than three times faster than the full detector. Comparing with alternative compact implementations of objects detectors, as shown in Table 4.3, the proposed mini detector exhibits relatively high detection accuracy when compared to tiny YOLO [91] and tiny SSD [93], which are designed for 2D image-based object detection. Expanding the architecture to benefit from 3D information, the proposed mini detector remains competitive with the performance of similar scale detectors reported in the literature while achieving higher detection framerate.

Table 4.3 Comparison between lightweight detection models performance.

Model	Size	Number of trained parameters	Speed (FPS)	mAP (%)
Tiny YOLO [91]	60.5 MB	-	133	57.1
Tiny SSD [93]	2.3 MB	1.13 M	-	61.3
Mini detector (proposed)	44.9 MB	7.19 M	158.97	82.7

The size shows the size of the model and number of trained parameters indicates the size of the output. More trained parameters a model has, more memory is taken by the output making it harder to implement on embedded computing platforms. The weight and the number of trained parameters are two important factors to evaluate the complexity of a detector.

4.3.4 Speed Evaluation on Different Hardware Environments

One of the main goals of designing the mini detector is to reduce the requirements in computing power. Thus, additional experiments are designed to evaluate the efficiency of the two proposed detectors, both in training and testing. As shown in Figure 4.4, the training time of the full detector is about 1.5x that of the mini detector. Note that the same training and validation dataset are used for both the full detector and the mini detector, and the batch size and trained epochs are also the same. Limited by the hardware resources available to us, two different hardware environments are used to test the performance. Results show that when training in a single GPU environment, a Tesla V100-PCIE performs 3 to 4 times faster than a Tesla T4. The characteristics of the hardware therefore play an important role in the training performance, as expected. However, the needed training time remains quite variable, running between 35 and 207 hours depending on the network architecture considered and the hardware resources available.

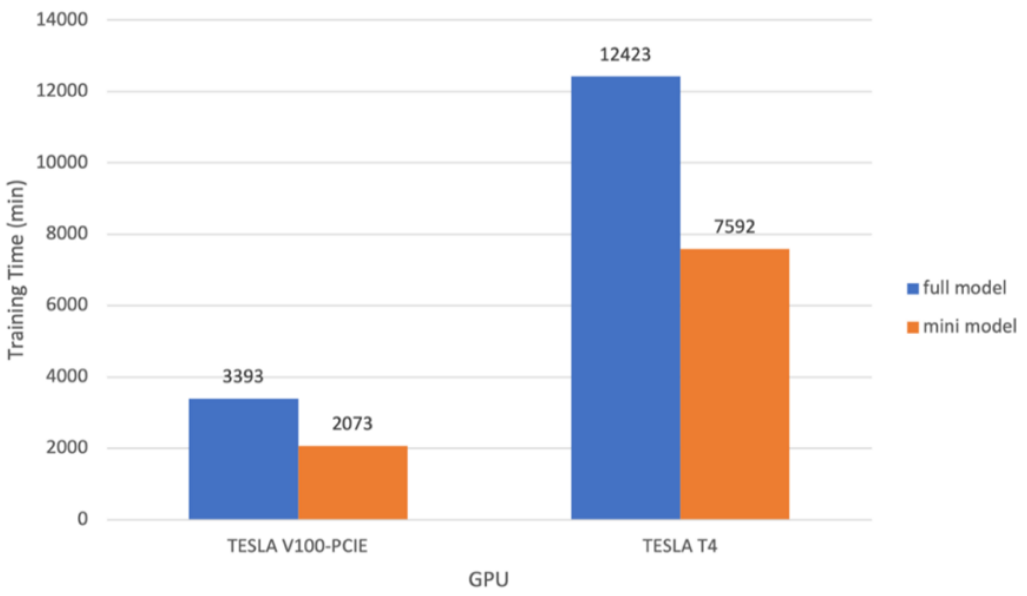


Figure 4.4 Training time of two proposed models on single Tesla V100-PCIE and single Tesla T4.

To evaluate the testing detection speed of the two proposed models on different hardware environments, tests are run on a single Tesla V100-PCIE, a single Tesla T4 and with no GPU, all combined with the same CPU and same memory size. As shown in Figure 4.5, while the full detector reaches 46.4 FPS on a single Tesla V100-PCIE, the mini detector's detection speed is over 3 times faster. While testing on a single Tesla T4, the overall framerate decreases but the gain in speed between the full and the mini detector reaches almost 7 times. The mini detector can even be used without a GPU, with a detection speed of 3.9 FPS, which is faster than most LIDAR based 3D detectors such as MV3D-Net [40] and PIXOR [64], and while the full detector version fails at providing real-time object detection and recognition.

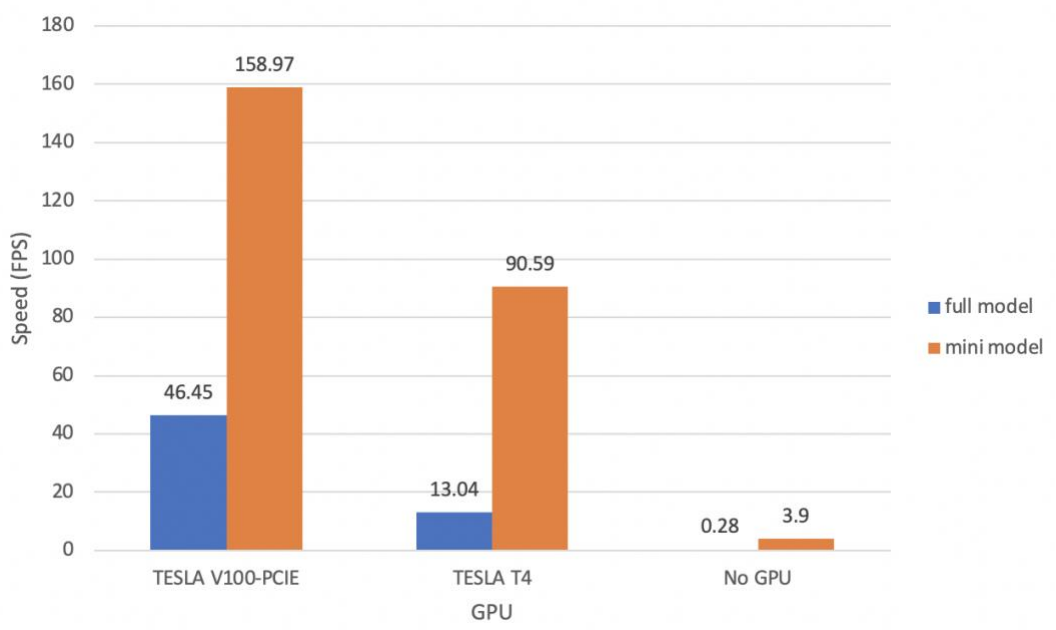


Figure 4.5 Detection speed, represented in frames per second, of the two proposed detector models running respectively on a single Tesla V100-PCIE, a single Tesla T4 and without GPU.

4.4 Summary

In this chapter, a mini detector model is proposed as a simplification of the full detector introduced in Chapter 3 and in combination with the tiny YOLO [91] architecture. By comparing the testing results achieved with the mini detector in comparison to the performance achieved with the full detector, it is concluded that by sacrificing some of the accuracy on the detection for some classes that suffer from a biased in the training dataset, the proposed mini detector can perform combined RGB image and LiDAR based 3D object detection and recognition task at least 3 times faster than the full detector while not dramatically reducing detection performance over classes of instances that are well represented in the training dataset.

Chapter 5 Conclusion

In this chapter, Section 5.1 summarizes the former chapters of this thesis. Section 5.2 discusses the contributions, and Section 5.3 proposes related research that can be pursued in the future.

5.1 Summary and Conclusion

Recent research shows an outstanding performance of convolutional neural networks in the field of computer vision and its potential for 3D image processing. Capitalizing on high-quality 3D point cloud collected by a LiDAR scanner and fusing the point cloud with RGB image, two deep learning architectures are proposed to further leverage the power of convolutional neural networks for 3D object recognition to assist autonomous driving.

To extract 3-dimensional location information from a LiDAR point cloud and reduce the computation and memory usage of the detection model, the point cloud is preprocessed into a BEV map. By using height thresholding, the height range of interest is emphasized on the height channel of the BEV map. Intensity values from the point cloud are accumulated and recorded to the corresponding pixel position as part of a resulting 5-dimensional BEV map.

The input data of the proposed detectors contains the BEV map converted from the LiDAR point cloud along with a registered and rescaled RGB image that provides a front forward view. In the design of the proposed full detector model, GIoU loss [4] and DarkNet-53 from a single stage detector YOLOv3 [1] are merged. In addition, Euler angle orientation is added to the detection head. Experiments demonstrate that the combination of Euler angle regression and GIoU losses provides better performance for the proposed detector in comparison to the MSE regression loss of the original YOLOv3 model.

To address the imbalance in detection results caused by some biased training data, focal loss [3] is used as the classification loss. Designed ablation studies show that focal loss can partially compensate for data imbalance in the proposed model and increases the mAP among classes.

The detection speed of the proposed full model reaches up to 46.4 FPS in our GPU-based implementation, with mAP over 90%. Experimental results also demonstrate that the model is adaptive to real-life autonomous driving environment conditions with different levels of occlusions.

In order to explore faster and lighter detection models better suited for real time and vehicle embedded implementations, a mini detector is also proposed. By introducing a lightweight deep learning detector into the 3D data processing field and combining it with key concepts proposed for the full detector, a compact 19-layer network model is implemented for 3D object detection and recognition, which reaches mAP over 80%. Experiments demonstrate that compared to the full detector, the mini detector takes about 2/3 of the training time and 1/3 of the testing time. A reasonable trade-off between processing time and accuracy can therefore be achieved for time-critical applications.

5.2 Contributions

This thesis makes the following original contributions to the fields of computer vision and machine learning with a focus on the application of combined RGB image and LiDAR data-based 3D object detection and recognition for autonomous driving:

- Introduction of Euler angle orientation to the regression of the detection model and modification of the corresponding loss function.
- Combination of focal loss [3], GIoU loss [4] and Euler angle in the loss function to tackle several problems in the training and improve detection performance.
- A merge of tiny YOLO [91] and proposed Euler angle regression network to form a lightweight architecture able to perform RGB image and LiDAR based 3D object detection and recognition tasks with higher framerate, which is of interest for time-critical applications.

5.3 Future Work

- Self-optimizing training model

From a theoretical point of view, the number and scale of feature maps generated during model training have variable effects on different training and testing images. Therefore, selecting appropriate parameters to generate appropriate feature maps has become one of the important steps that affect the detection performance. A stronger generalization ability is preferred for the detector, that is, to adapt to different environments (occlusion, low resolution, varying scene complexity), which requires better self-optimizing ability of the detection model. Existing networks can optimize parameters during training, but still involve manual setting of initial values. In future research, automating the adjustment of the training parameters and feature maps extracted from the training data will further improve the generalization ability of the detectors.

- Sensor-independent fusion framework

From an engineering point of view, the redundant design of autonomous vehicles is critical to their safety. Although the fusion of LiDAR scanner and cameras can improve perception performance, it also brings signal coupling issues. If a signal path fails during operation, the entire process may malfunction and affect downstream modules. This is unacceptable for autonomous vehicles operating in safety-critical environments. This problem can be solved by adding multiple fusion modules that can accept different sensor inputs, or asynchronous multi-modal data, multi-path fusion modules. But figuring out the best solution still involves further research.

- Fighting imbalance in dataset

From an experimental point of view, the imbalance of the dataset on which the model is trained has a great impact on the object detection accuracy. In the experiments conducted in this thesis, the imbalance of the training dataset causes the detection results of the proposed model to be biased for different classes. In this thesis, focal loss [3] is used to reduce the bias but the AP of “cyclist” and “pedestrian” remains lower than for the “car” category which is

over-represented in the dataset. For autonomous driving, detecting cyclists and pedestrians is as important as detecting cars, hence it is important to fight the imbalance in dataset. One way is to change the structure and parameters of the model to make it less sensitive to the number of the training samples. Another way is to adjust the training dataset to balance the number of samples from different classes, such as undersampling the majority class or oversampling the minority class. Future research and experiments on dealing with data imbalances are worth looking forward to.

References

[1] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," arXiv preprint arXiv:1804.02767, 2018.

[2] A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3354-3361, 2012.

[3] TY. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal loss for dense object detection," in Proceedings of the IEEE international conference on computer vision, pp. 2980-2988, 2017.

[4] H. Rezatofighi, N. Tsoi, J.Y. Gwak, A. Sadeghian, I. Reid and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 658-666, 2019.

[5] J. Christian and C. Scott, "A survey of LIDAR technology and its use in spacecraft relative navigation." In AIAA Guidance, Navigation, and Control (GNC) Conference, p. 4641. 2013.

[6] Velodyne, "HDL-64E," online, [Retrieved Jul. 15, 2020], retrieved from Internet, 2016.

[7] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," Nature 521, 436-444, 2015.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in International conference on machine learning, pp. 448-456, PMLR, 2015.

[9] A.K. Dubey and V. Jain, "Comparative study of convolution neural network's relu and leaky-relu activation functions," in Applications of Computing, Automation and Wireless Systems in Electrical Engineering, pp. 873-880, Springer, Singapore, 2019.

[10] A.F. Agarap, "Deep learning using rectified linear units (relu)," arXiv preprint arXiv:1803.08375, 2018.

[11] TY. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature pyramid networks for object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2117-2125, 2017.

[12] P.D. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[13] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in Proceedings of COMPSTAT'2010, pp. 177-186, Physica-Verlag HD, 2010.

[14] L.K.C. Chan, N. Jegadeesh and J.Lakonishok, "Momentum strategies," the Journal of Finance, 1681-1713, 1996.

[15] G. Hinton, N. Srivastava and K. Swersky, "Neural networks for Machine Learning," cs.toronto.edu, 2012.

[16] K.T.Gribbon and Donald G. Bailey. "A novel approach to real-time bilinear interpolation." In Proceedings. DELTA 2004. Second IEEE international workshop on electronic design, test and applications, pp. 126-131. IEEE, 2004.

[17] R.B. Girshick, From rigid templates to grammars: Object detection with structured models, The University of Chicago, 2012.

[18] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," arXiv preprint arXiv:1312.6229, 2013.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks." Communications of the ACM 60, no. 6 (2017): 84-90.

[20] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580-587, 2014.

[21] K. He, X. Zhang, S. Ren and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," IEEE transactions on pattern analysis and machine intelligence, 37, no. 9: 1904-1916, 2015.

[22] R. Girshick, "Fast r-cnn," in Proceedings of the IEEE international conference on computer vision, pp. 1440-1448, 2015.

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

[24] J. Dai, Y. Li, K. He and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," Advances in neural information processing systems, 2016.

[25] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[26] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[27] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask r-cnn," in Proceedings of the IEEE international conference on computer vision, pp. 2961-2969, 2017.

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

[29] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7263-7271, 2017.

[30] A. Bochkovskiy, C.Y. Wang, and H.Y. M. Liao. "Yolov4: Optimal speed and accuracy of object detection." arXiv preprint arXiv:2004.10934 (2020).

[31] G. Jocher, K. Nishimura, T. Mineeva, and R. Vilariño. "yolov5." Code repository (2020).

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

[33] Z. Shen, Z. Liu, J. Li, Y.G. Jiang, Y. Chen and X. Xue, "Dsod: Learning deeply supervised object detectors from scratch," in Proceedings of the IEEE international conference on computer vision, pp. 1919-1927, 2017.

[34] C. Yan, H. Zhang, X. Li and D. Yuan, "R-SSD: refined single shot multibox detector for pedestrian detection," Applied Intelligence: 1-18, 2022.

[35] H. Law, and J. Deng. "Cornernet: Detecting objects as paired keypoints." In Proceedings of the European conference on computer vision (ECCV), pp. 734-750. 2018.

[36] X. Zhou, D. Wang, and P. Krähenbühl. "Objects as points." arXiv preprint arXiv:1904.07850 (2019).

[37] D. Bahdanau, K. Cho and Y. Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser and I. Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[39] C. Kolhatkar and K. Wagle. "Review of SLAM algorithms for indoor mobile robot with LIDAR and RGB-D camera technology." Innovations in Electrical and Electronic Engineering: Proceedings of ICEEE 2020 (2021): 397-409.

[40] X. Chen, H. Ma, J. Wan, B. Li and T. Xia, "Multi-view 3d object detection network for autonomous driving," in Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 1907-1915, 2017.

[41] J. Ku, M. Mozifian, J. Lee, A. Harakeh and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1-8, IEEE, 2018.

[42] M. Liang, B. Yang, S. Wang and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," in Proceedings of the European conference on computer vision (ECCV), pp. 641-656, 2018.

[43] M. Liang, B. Yang, Y. Chen, R. Hu and R. Urtasun, "Multi-task multi-sensor fusion for 3d object detection," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7345-7353, 2019.

[44] H. Lu, X. Chen, G. Zhang, Q. Zhou, Y. Ma and Y. Zhao, "SCANet: Spatial-channel attention network for 3D object detection," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1992-1996, 2019.

[45] Y. Zeng, Y. Hu, S. Liu, J. Ye, Y. Han, X. Li and N. Sun, "Rt3d: Real-time 3-d vehicle detection in lidar point cloud for autonomous driving," IEEE Robotics and Automation Letters: 3434-3440, 2018.

[46] Z. Yang, Y. Sun, S. Liu, X. Shen and J. Jia, "Ipod: Intensive point-based object detector for point cloud," arXiv preprint arXiv:1812.05276, 2018.

[47] S. Shi, X. Wang and H. Li, "3d object proposal generation and detection from point cloud," in Proceedings of the IEEE conference on computer vision and pattern recognition, Long Beach, CA, USA, pp. 15-20, 2019.

[48] J. Zarzar, S. Giancola and B. Ghanem, "PointRGCN: Graph convolution networks for 3D vehicles detection refinement," arXiv preprint arXiv:1911.12236, 2019.

[49] S. Vora, A. H. Lang, B. Helou and O. Beijbom, “Pointpainting: Sequential fusion for 3d object detection," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 4604-4612, 2020.

[50] Z. Yang, Y. Sun, Shu Liu, X. Shen and J. Jia, “Std: Sparse-to-dense 3d object detector for point cloud,” in Proceedings of the IEEE/CVF international conference on computer vision, pp. 1951-1960, 2019.

[51] C. R. Qi, W. Liu, C. Wu, H. Su and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 918-927, 2018.

[52] X. Zhao, Z. Liu, R. Hu and K. Huang, “3D object detection using scale invariant and feature reweighting networks,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, pp. 9267-9274, 2019.

[53] D. Xu, D. Anguelov and A. Jain, “Pointfusion: Deep sensor fusion for 3d bounding box estimation,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 244-253, 2018.

[54] K. Shin, Y. P. Kwon and M. Tomizuka, "Roarnet: A robust 3d object detection based on region approximation refinement,” in Intelligent Vehicles Symposium (IV), pp. 2510-2515, IEEE, 2019.

[55] Z. Wang and K. Jia, "Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection,” in International Conference on Intelligent Robots and Systems (IROS), pp. 1742-1749, IEEE, 2019.

[56] J. Lehner, A. Mitterecker, T. Adler, M. Hofmarcher, B. Nessler and S. Hochreiter, "Patch Refinement--Localized 3D Object Detection,” arXiv preprint arXiv:1910.04093, 2019.

[57] D. Zhou, J. Fang, X. Song, C. Guan, J. Yin, Y. Dai and R. Yang, “Iou loss for 2d/3d object detection,” in International Conference on 3D Vision (3DV), pp. 85-94, IEEE, 2019.

[58] Y. Chen, S. Liu, X. Shen and J. Jia, "Fast point r-cnn,” in Proceedings of the IEEE/CVF international conference on computer vision, pp. 9775-9784. 2019.

[59] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang and H. Li, “Pv-rcnn: Point-voxel feature set abstraction for 3d object detection,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10529-10538, 2020.

[60] C. R. Qi, O. Litany, K. He and L. J. Guibas, “Deep hough voting for 3d object detection in point clouds,” in proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 9277-9286, 2019.

[61] M. Feng, S. Z. Gilani, Y. Wang, L. Zhang and A. Mian, “Relation graph network for 3D object detection in point clouds,” IEEE Transactions on Image Processing: 92-107, 2020.

[62] C. R. Qi, X. Chen, O. Litany and L. J. Guibas, “Imvotenet: Boosting 3d object detection in point clouds with image votes,” in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 4404-4413, 2020.

[63] S. Shi, Z. Wang, J. Shi, X. Wang and H. Li, “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: 2647-2664, 2020.

[64] B. Yang, W. Luo and R. Urtasun, “Pixor: Real-time 3d object detection from point clouds,” in Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 7652-7660, 2018.

[65] B. Yang, M. Liang and R. Urtasun, “Hdnet: Exploiting hd maps for 3d object detection,” in Conference on Robot Learning, pp. 146-155, PMLR, 2018.

[66] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. Garcia and A. D. L. Escalera, “Birdnet: a 3d object detection framework from lidar information,” in International Conference on Intelligent Transportation Systems (ITSC), pp. 3517-3523, IEEE, 2018.

[67] B. Li, T. Zhang and T. Xia, “Vehicle detection from 3d lidar using fully convolutional network,” arXiv preprint arXiv:1608.07916, 2016.

[68] B. Li, “3d fully convolutional network for vehicle detection in point cloud,” in International Conference on Intelligent Robots and Systems (IROS), pp. 1513-1518, IEEE, 2017.

[69] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong and I. Posner, “Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks,” in International Conference on Robotics and Automation (ICRA), pp. 1355-1361, IEEE, 2017.

[70] X. Li, J. Guivant, N. Kwok, Y. Xu, R. Li and H. Wu, “Three-dimensional backbone network for 3d object detection in traffic scenes,” arXiv preprint arXiv:1901.08373, 2019.

[71] Y. Zhou and O. Tuzel, “Voxelnet: End-to-end learning for point cloud based 3d object detection,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4490-4499, 2018.

[72] Y. Yan, Y. Mao and B. Li, “Second: Sparsely embedded convolutional detection,” Sensors: 3337, 2018.

[73] V. A. Sindagi, Y. Zhou and O. Tuzel, “Mvx-net: Multimodal voxelnet for 3d object detection,” in International Conference on Robotics and Automation (ICRA), pp. 7276-7282, IEEE, 2019.

[74] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang and O. Beijbom, “Pointpillars: Fast encoders for object detection from point clouds,” in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 12697-12705, 2019.

[75] C. He, H. Zeng, J. Huang, X. S. Hua and L. Zhang, “Structure aware single-stage 3d object detection from point cloud,” in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 11873-11882, 2020.

[76] Z. Yang, Y. Sun, S. Liu and J. Jia, “3dssd: Point-based 3d single stage object detector,” in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 11040-11048, 2020.

[77] J. Mao, Y. Xue, M. Niu, H. Bai, J. Feng, X. Liang, H. Xu and C. Xu. "Voxel transformer for 3d object detection." In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 3164-3173. 2021.

[78] W. Luo, B. Yang and R. Urtasun, “Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net,” in Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 3569-3577, 2018.

[79] R. B. Rusu, “Semantic 3D object maps for everyday manipulation in human living environments,” KI-Künstliche Intelligenz: 345-348, 2010.

[80] D. J. Munoz, A. Bagnell, N. Vandapel and M. Hebert, “Contextual classification with functional max-margin markov networks,” in Conference on Computer Vision and Pattern Recognition, pp. 975-982, IEEE, 2009.

[81] B. Vallet, M. Brédif, A. Serna, B. Marcotegui and N. Paparoditis, “TerraMobilita/iQmulus urban point cloud analysis benchmark,” Computers & Graphics: 126-133, 2015.

[82] X. Roynard, J. E. Deschaud and F. Goulette, “Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification,” the International Journal of Robotics Research: 545-557, SAGE Publications Sage UK: London, England, 2018.

[83] M. D. Deuge, A. Quadros, C. Hung and B. Douillard, “Unsupervised feature learning for classification of outdoor 3d scans,” in Australasian conference on robotics and automation, Kensington, Australia: University of New South Wales, 2013.

[84] K. V. Vishwanath, D. Gupta, A. Vahdat and K. Yocum, “Modelnet: Towards a datacenter emulation environment,” in Ninth International Conference on Peer-to-Peer Computing, pp. 81-82, IEEE, 2009.

[85] Google Colaboratory. [Online]. Available: https://research.google.com/colaboratory.

[86] N. Hawes, C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrova and J. Young, “The strands project: Long-term autonomy in everyday environments,” IEEE Robotics & Automation Magazine 24, 146-156, 2017.

[87] M. Simony, S. Milzy, K. Amendey and H. M. Gross, “Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds,” in Proceedings of the European Conference on Computer Vision (ECCV) Workshops, pp. 2018.

[88] J. Craig, Introduction to Robotics, Mechanics and Control, Pearson Prentice Hall, 3rd edition, section 2.8, 2005.

[89] Z. Zhang and M. Sabuncu, “Generalized cross entropy loss for training deep neural networks with noisy labels,” Advances in neural information processing systems, 2018.

[90] K. Jensen, and N. Wirth. PASCAL user manual and report: ISO PASCAL standard. Springer Science & Business Media, 2012.

[91] K. C. Saranya, A. Thangavelu, A. Chidambaram, S. Arumugam and S. Govindraj, “Cyclist detection using tiny yolo v2,” in Soft Computing for Problem Solving, pp. 969-979, Springer, Singapore, 2020.

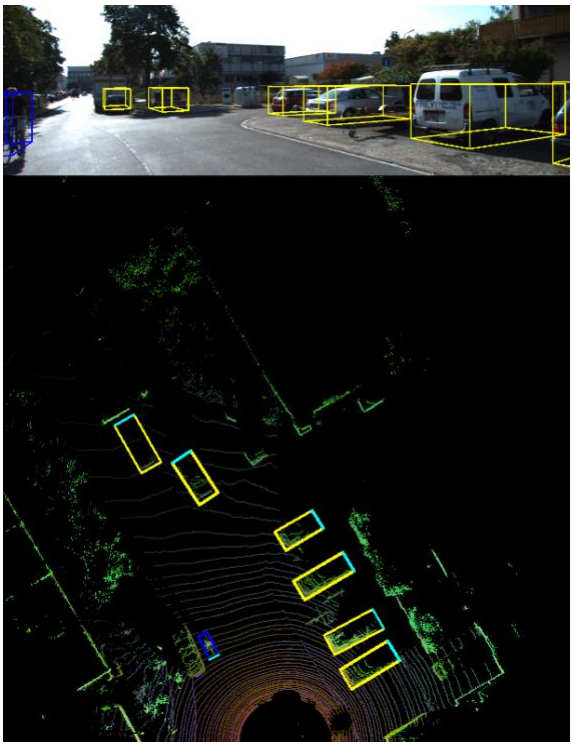
[92] B. Wu, F. Iandola, P. H. Jin and K. Keutzer, “Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving,” in Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp. 129-137. 2017.

[93] A. Womg, M. Javad Shafiee, F. Li and B. Chwyl, “Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection,” in Conference on Computer and Robot Vision (CRV), pp. 95-101, IEEE, 2018.

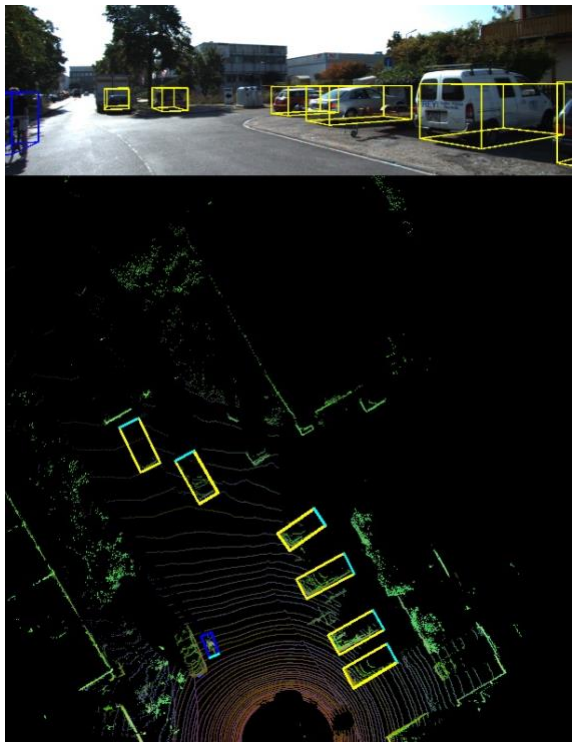
Appendix A: Detection Results with the Proposed Full-scale Detector on KITTI

Appendix A showcases further results of object detection experiments conducted on samples from the KITTI dataset, categorized into three difficulty levels as described in Section 3.4.1. For each sample, the original RGB image without bounding boxes is displayed above the results to aid in the visualization of target objects. The ground truth, as labeled in the KITTI dataset, is shown in the left column, while the right column displays the experimentally predicted bounding boxes around each object instance, color-coded by class (cars: yellow, cyclists: blue, and pedestrians: red) using the proposed full-scale detector. The top part of each image displays the front forward view RGB image, and the bottom part shows the corresponding bird-eye view map.

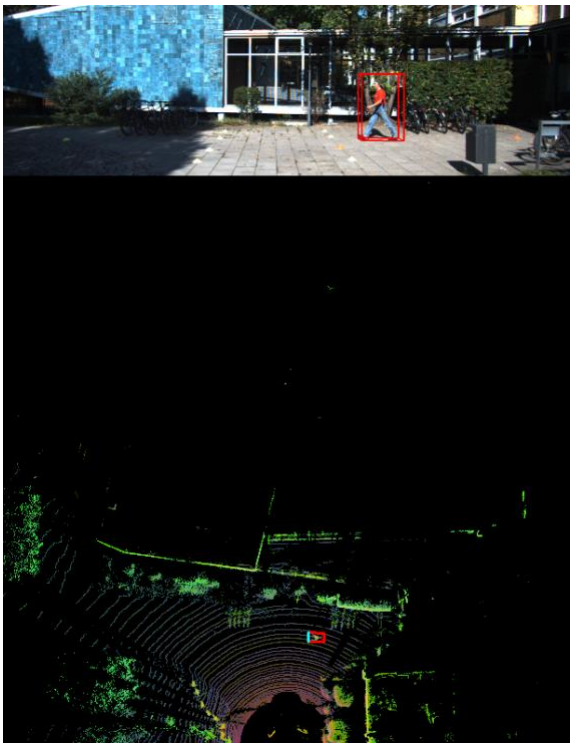
- A.1 Additional “easy” test case examples



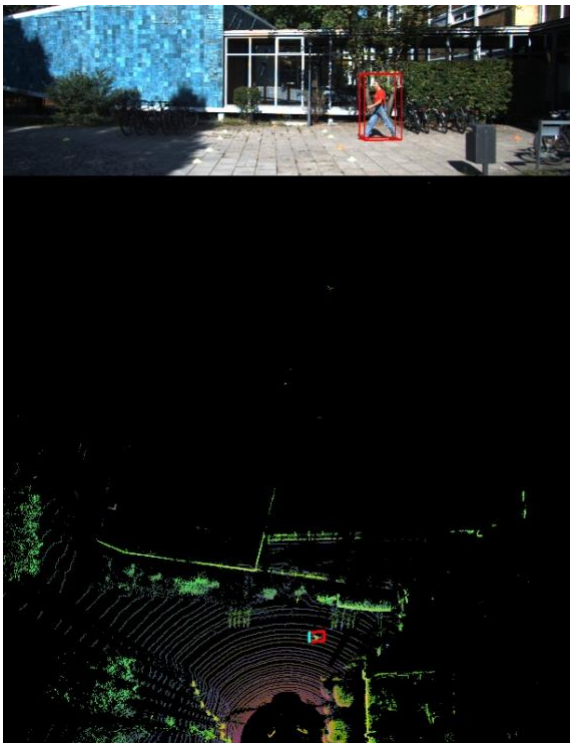
Ground truth



Result with full scale detector



Ground truth



Result with full scale detector

- A.2 Additional “moderate” test case examples

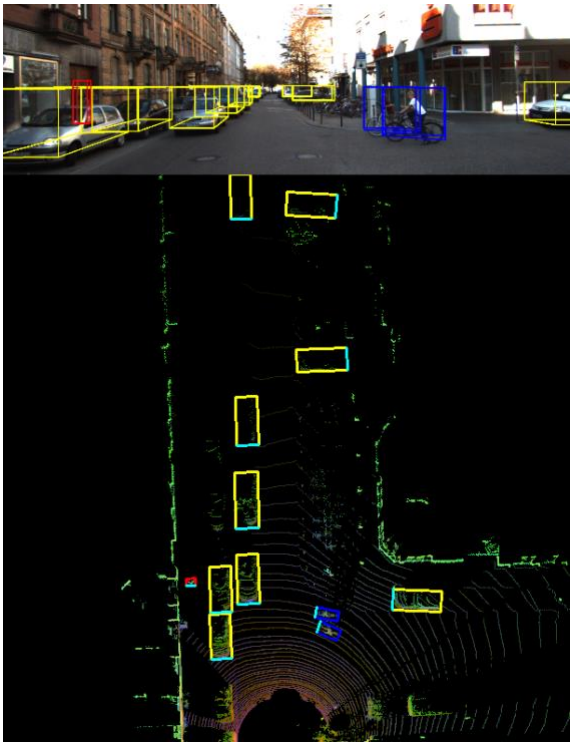
Ground truth

Result with full scale detector

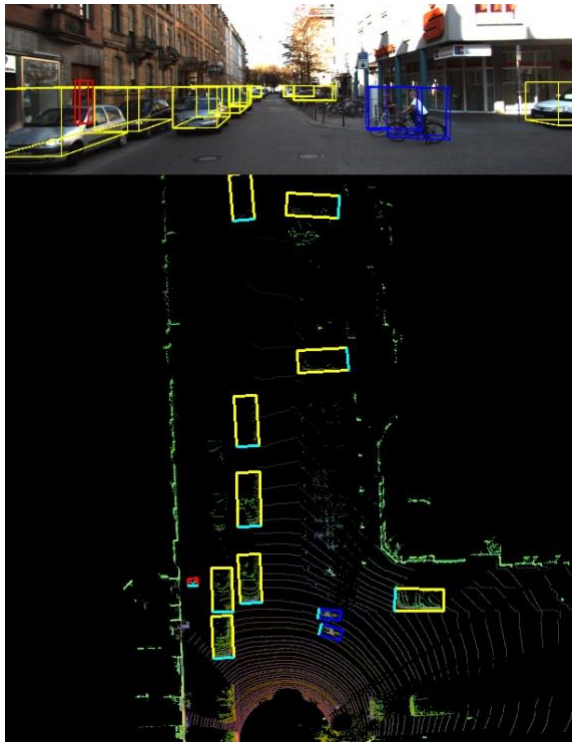
Ground truth

Result with full scale detector

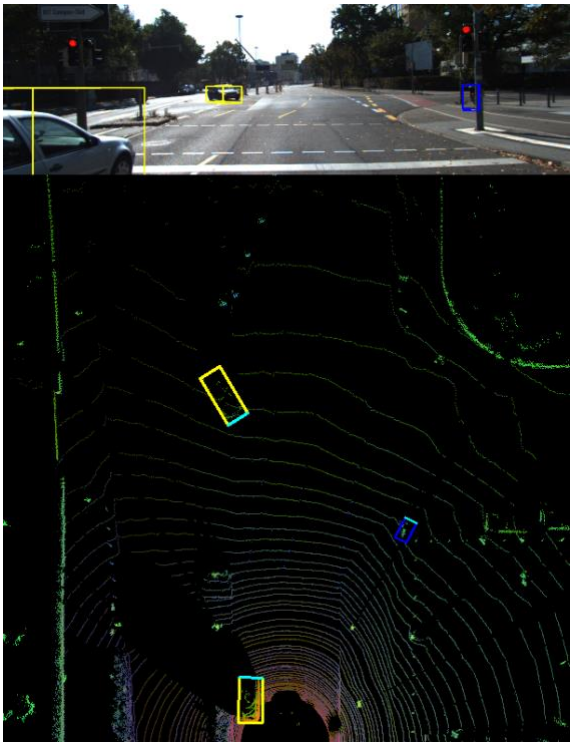
- A.3 Additional “hard” test case examples



Ground truth



Result with full scale detector

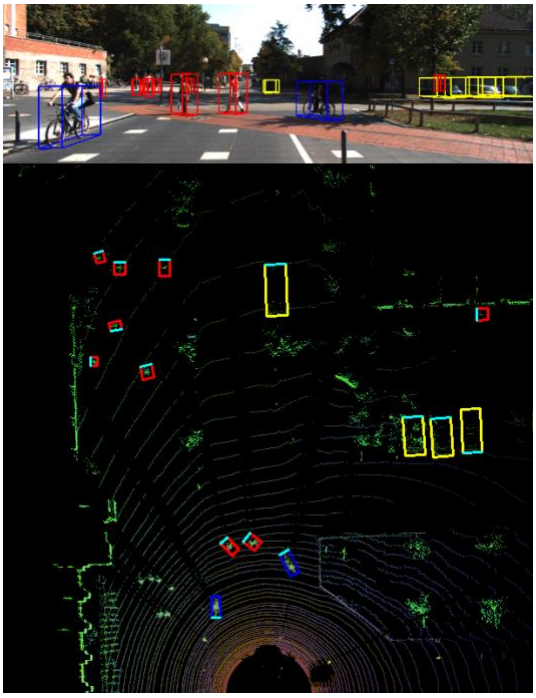


Ground truth

Result with full scale detector

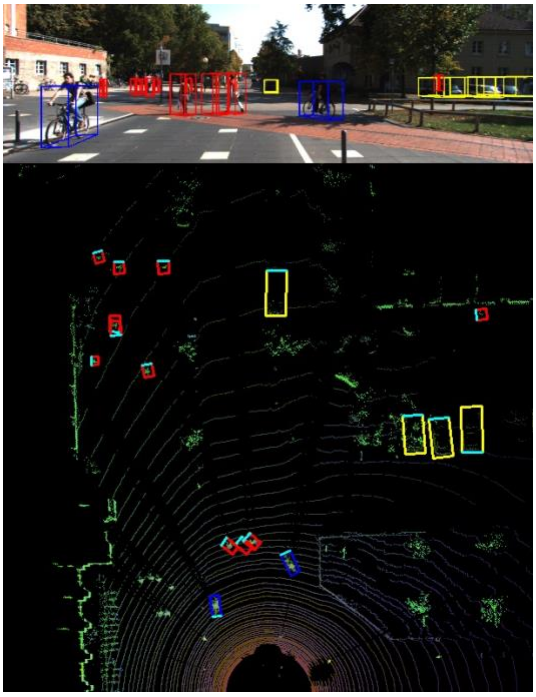
Appendix B: Detection Results with the Proposed Mini Detector

Appendix B compares the results of the proposed full detector and mini detector, as described in Section 4.3.2. For each sample, the ground truth with bounding boxes is on the top left of each figure. The original RGB image without bounding boxes is displayed on the top right of each figure for clear visualization of target objects. The bottom left shows the experimentally predicted bounding boxes around each object instance, color-coded by class (cars: yellow, cyclists: blue, and pedestrians: red) using the proposed full detector. The bottom right shows the experimentally predicted bounding boxes around each object using the proposed mini detector.

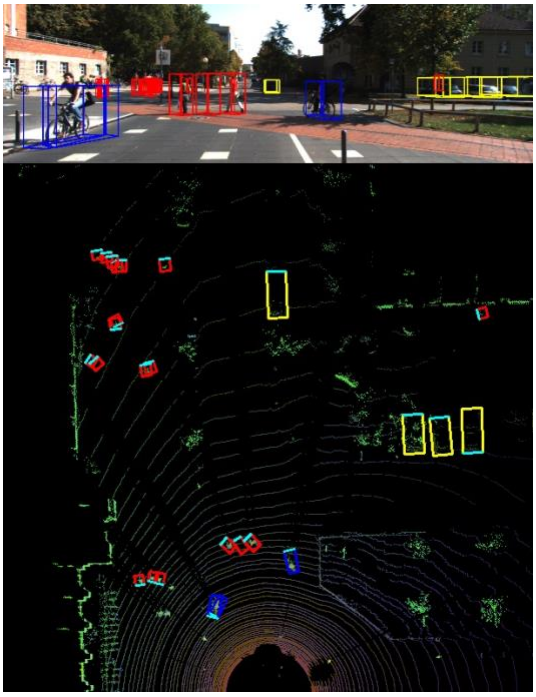


Ground truth

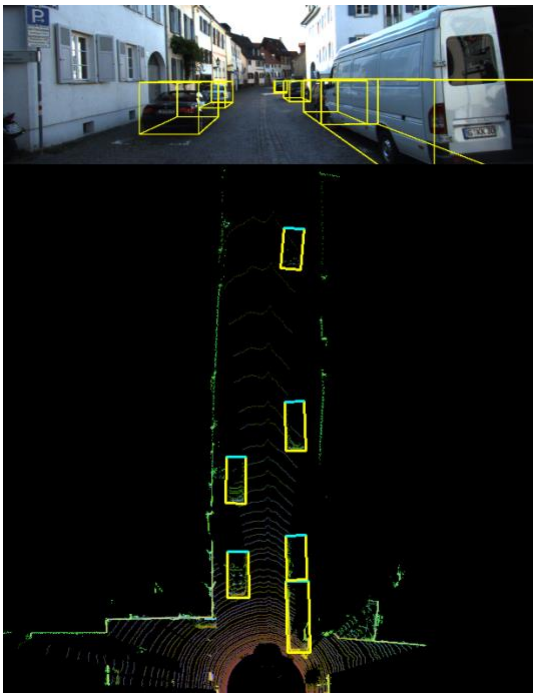
RGB image



Result with
full detector

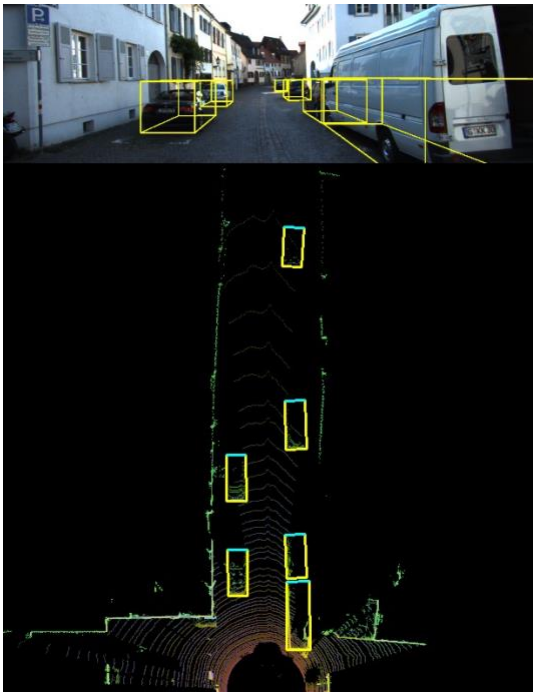


Result with
mini detector

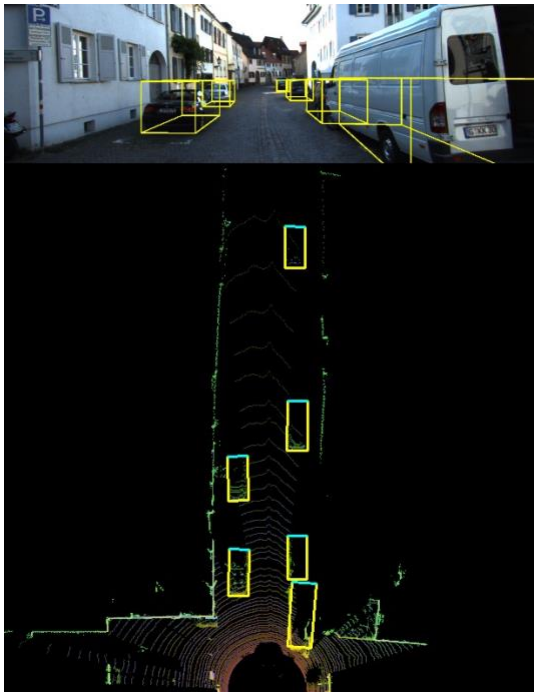


Ground truth

RGB image



Result with
full detector



Result with
mini detector