TUM School of Computation, Information and Technology
Technical University of Munich

Master's Thesis in Robotics, Cognition, Intelligence

Real-time Multi-view Road-side 3D Object Detection

Straßenseitige 3D-Objekterkennung in Echtzeit mit mehreren Ansichten

Supervisor Prof. Dr.-Ing. habil. Alois C. Knoll

Advisor M.Sc. Xingcheng Zhou, M.Sc. Walter Zimmer

Author Samyak Jain

Date July 15, 2024 in Munich

Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Munich, July 15, 2024

(Samyak Jain)

Acknowledgments

I extend my deepest gratitude to Prof. Knoll for giving me the opportunity to write my thesis under his supervision. I want to express my profound gratitude to Xingcheng Zhou and Walter Zimmer, my research supervisors, for their continuous support, guidance, and encouragement throughout this master's thesis journey. Their insightful feedback and unwavering support have been pivotal throughout this journey.

I am equally indebted to my family, whose love and trust have not gone unnoticed. Their endless encouragement and faith in my abilities gave me strength and resilience, enabling me to navigate and surmount the challenges of this academic endeavor. Their unyielding support has been a cornerstone of my success.

Abstract

Object perception from infrastructure perspective is increasingly gaining attention, yet vision-based 3D object detection from multiple view perspectives has not been thoroughly explored. Machine learning architectures based on cameras as sensor modalities have seen great advances in two dimensional object detection but rather perform poorly in three dimensional. This problem leads to the advent of BEVDepth which utilizes point cloud during its training to learn better depth estimation and infer solely using cameras. The decision of using point cloud only for training makes this model easier and cheaper to deploy in real world scenarios and increases its speed. In this thesis, we focus on expanding this model to multi-view scenarios. We utilize existing model as baseline, enhancing its performance while maintain its real-time speed. We employ the TUMTraffic-Intersection dataset and try to extend it to new views, and handle new objects.

Zusammenfassung

Die Objektwahrnehmung aus der Infrastrukturperspektive gewinnt zunehmend an Aufmerksamkeit, doch die visuelle 3D-Objekterkennung aus verschiedenen Blickwinkeln ist noch nicht gründlich erforscht worden. Architekturen des maschinellen Lernens, die auf Kameras als Sensormodalitäten basieren, haben bei der zweidimensionalen Objekterkennung große Fortschritte gemacht, schneiden aber bei der dreidimensionalen Erkennung eher schlecht ab. Dieses Problem führte zur Entwicklung von BEVDepth, das während des Trainings eine Punktwolke verwendet, um eine bessere Tiefenschätzung zu erlernen und nur mit Hilfe von Kameras zu schließen. Die Entscheidung, nur Punktwolken für das Training zu verwenden, macht es einfacher und billiger, dieses Modell in realen Szenarien einzusetzen und erhöht seine Geschwindigkeit. In dieser Arbeit konzentrieren wir uns auf die Erweiterung dieses Modells auf Szenarien mit mehreren Ansichten. Wir verwenden das bestehende Modell als Basis, verbessern seine Leistung und behalten dabei seine Echtzeitgeschwindigkeit bei. Wir verwenden den TUMTraffic-Intersection-Datensatz und versuchen, ihn auf neue Ansichten zu erweitern und neue Objekte zu verarbeiten.

Contents

Chapter 1

Introduction

1.1 Motivation

With the automobile industry currently transitioning from internal combustion engine cars to electrical vehicles, there is a ongoing boom to make the cars smarter. According to Statista [Pla19], the Vehicle to everything (V2X) market will grow to 6 billion US dollars in 2025 from a mere 860 million in 2018 as can be seen in Figure: 1.1. This parallel boom to have a autonomous driving car has led researchers to look at autonomous driving car more seriously. The term V2X is an umbrella term used to define the communication of ve-

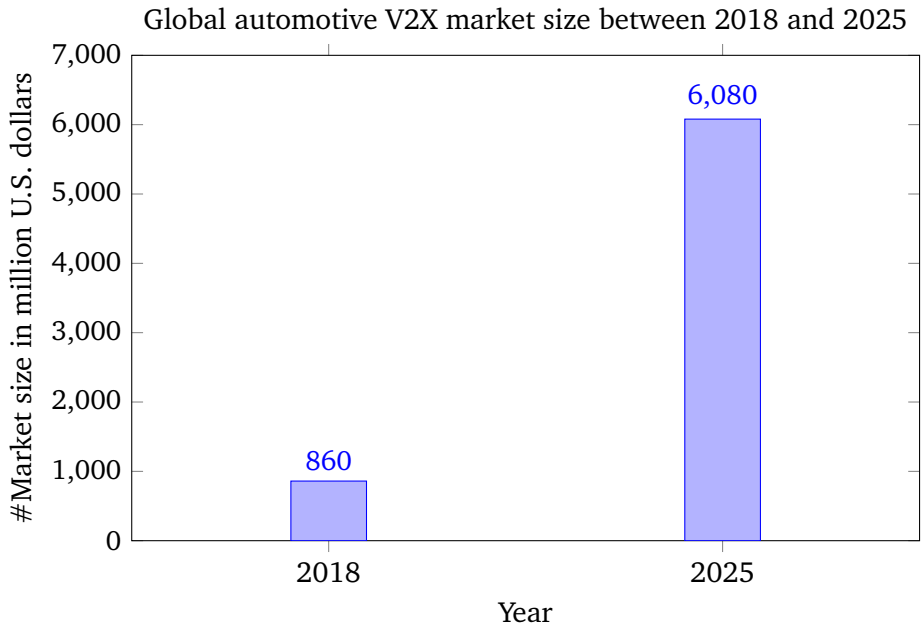
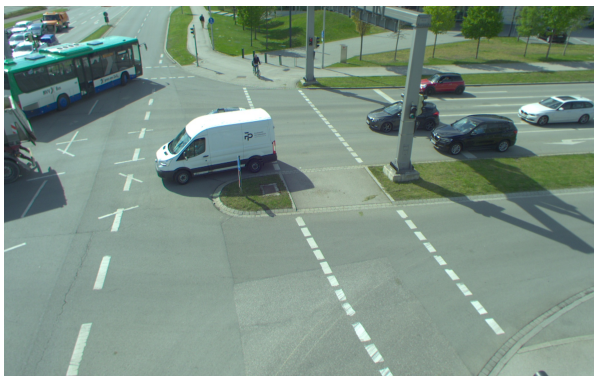


Figure 1.1: Statista estimate on increase in market value of V2X

hicle to other systems. In this work this other system is infrastructure, particularly a gantry with sensors embedded onto it with roadside computational unit available. The Vehicle-to-Infrastructure communication is necessary if autonomous driving is to be solved. This is the basis of TUMTraf-I dataset [Zim+23] which will be discussed in a later chapter but briefly Figure: 1.2 shows sample images from two different views in different day time condition.

As per the Society of Automotive Engineers (SAE) there are six levels of autonomy [Int21] (as of 2021):

- **Level 0: No Driving Automation** This state is already achieved with 0 driving assis-



(a) South 1 day time image

(b) South 2 day time image

(c) South 1 night time image

(d) South 2 night time image

Figure 1.2: Example images from TUMTraf-I dataset. Original image size 1920×1200

tance, all tasks are handled by the human driver.

- **Level 1: Driver Assistance** This too is already present in today's car, case in point cruise control systems. These system provides constant assistance but full attention of driver is required.
- **Level 2: Partial Driving Automation** The vehicle can change lane or perform parallel park, quite advanced features but still available in market. Full attention of driver still required.
- **Level 3: Conditional Driving Automation** First level where driver can sit back but has to take over if vehicle comes into a pickle. The semi-autonomous vehicle can overtake, brake and accelerate by judging the surrounding traffic.
- **Level 4: High Driving Automation** Vehicle is quite autonomous and takes over all the functionality from the driver.
- **Level 5: Full Driving Automation** The holy grail of autonomous driving where no help is required from human driver as the vehicle can drive better than it's host and can take care of issues which are encountered for the first time

Due to its limited long-range sensing capabilities and lack of global viewpoint, autonomous driving still has safety issues. It is generally acknowledged that collaboration between vehicles and infrastructure is necessary to reach Level 5 autonomy. There are several important benefits to using sensors from both infrastructure and vehicles, such as addressing blind spots and offering a global vision that extends well beyond the present horizon. This is where TUMTraf-I dataset can be utilized to train machine learning models in the infrastructure view and deploy it real time to get better understanding of the environment and share it with the vehicles on the road so that they make better informed decisions.

1.2 Birds-Eye-View representation

Learning powerful representations in bird's-eye-view (BEV) for perception tasks is rapidly gaining traction and attracting significant attention from both industry and academia. Traditionally, autonomous driving algorithms have performed detection, segmentation, and tracking in a front or perspective view. However, as sensor configurations become more complex, the integration of multi-source information from different sensors and the representation of features in a unified view have become critically important. BEV perception offers several advantages: it intuitively represents surrounding scenes, is fusion-friendly, and is highly desirable for subsequent modules such as planning and control. This makes BEV a perfect candidate for application in Infrastructure view.

BEVDepth [Li+23] is the model chosen in this work because of it's benefit of using only camera modality making it faster than other fusion based methods where point clouds needs to be processed. At the same time, there is the option to train to get a better depth estimate using the depth map generated from point cloud which in case of TUMTraf-I is available for each sample. Apart from that, it's performance on the both datasets Rope3D [Ye+22] and DAIR-V2X-I [Yu+22] placed it at the top just below BEVHeight [Yan+23] but during the start of this work, the code for it wasn't publicly available.

1.3 Goals

Goal of this work is to run BEVDepth real time using multiple views of image. The benchmark for acceptable real time speed of the model calculated as frames per second (FPS) is set to 10. BEVDepth trains on nuScenes [Cae+20] but this work is based on the TUMTraf-I dataset [Zim+23] which has a different format than nuScenes. Though a converter is available to convert TUMTraf-I dataset into nuScenes format, it could be improved and streamlined which it would be in this work. The performance on nuScenes dataset is 50.3% by training on both training and validation split and by using higher resolution image as model input simultaneously not reporting the model's speed and memory utilization. In this work the BEVDepth model would be trained only on training dataset split with validation split used as metric all the while reporting on the model's inference speed and memory utilization.

1.4 Outline

The following chapters are structured as follows: Chapter 2 explains other works related to this work in more detail. Chapter 3 delves into details of the TUMTraf-I dataset used in this work. Chapter 4 covers the implementation details of this work done. Chapter 5 covers the details of the experiments performed and qualitative and quantitative analysis performed along with ablation.Finally, Chapter 6 covers the conclusion reached by this work along with future works to improve.

Chapter 2

Related Work

This chapter provides an overview of all works that are related to this work. BEVDepth and the work it's built on is thoroughly discussed here. Along with other state-of-the-art Birds-Eye-View methods that serve as both inspiration, will be presented in the next sections.

2.1 Deep learning

Deep learning (DL) is a modeling technique inspired by neurons within the field of Machine learning (ML) which again comes under the umbrella of Artificial intelligence (AI) with prominent feature of finding pattern in both structured and unstructured datasets [GBC16]. One of the more evolved form of deep learning in dealing with images is convolution neural networks (CNN) [ON15] and very recently Transformers [Vas+17] has shown great potential.

With their characteristics of being able to find general pattern in data using multiple layers of neural networks, cutting edge research allows better and proficient models to do task which were generally limited only to humans e.g. Image recognition, object detection, autonomous driving, natural language processing, speech recognition, healthcare etc [BN06] [JZH21].

2.2 3D Object Detection

The application of deep learning in 2D detection is well known and studied but it fails to localise the objects in real world with real applications. This makes 3D object detection critical in various branches of work like autonomous driving, robotic manipulation, surveillance etc. Unlike 2D object detection where object detection concerns with bounding the object in a box, the degrees of freedom in 3D world is higher which translates to finding the length, width, height along with object's center (x, y, z) and orientation which can be represented as a unit Quaternion [Per11] $q = (w, x, y, z)$ in a global coordinate frame. State of the art deep learning models train on large datasets with multi view images and point cloud to learn about the general 3D features which are then used to generate bounding boxes and classification of the object in images and/or point cloud.

Since this work focuses on multi-view images, the related work focuses mostly on models with camera modality instead of point cloud. Though some models might use point cloud in certain other degree, they would still be strictly camera based meaning point cloud is only optional (but a nice to have).

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.1: different ResNet configs [He+16]

2.2.1 2D Backbones

Many of the state of the art models that would be discussed have their backbone and neck based on pretrained 2D models which have been trained on large images dataset like ImageNet [Rus+15]. This helps in extracting 2D features from images that could then be utilized for downstream task.

Often these backbones are CNNs like ResNet [He+16], SECOND FPN [YML18] etc are used with different variation of ResNet available. Residual Network (ResNet) was first introduced in 2016. The core idea that ResNet brought was the introduction of skip connection. In a case where there are two consecutive convolution layers, the skip connection allows the raw input of the first layer to be added to the output of the second layer. The reasoning to do such was that it helped solving the problem of vanishing gradients that deep neural networks were generally having.

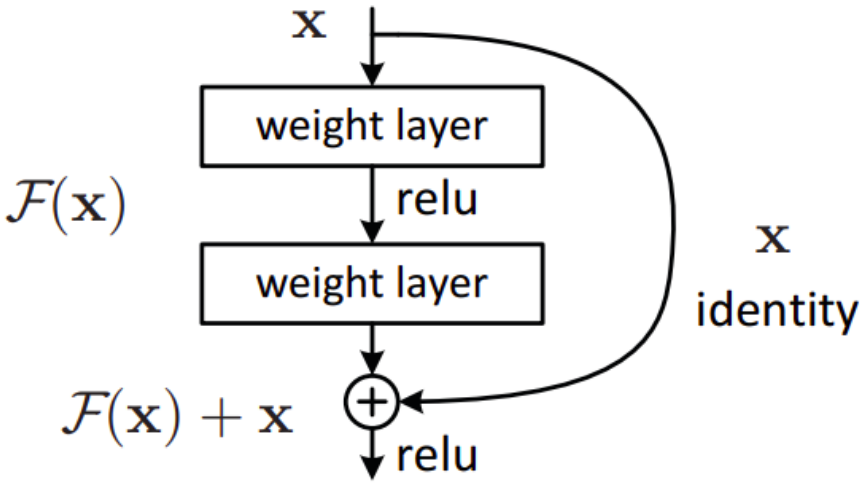


Figure 2.2: ResNet skip connection [He+16]

One of the most used ResNet configuration, as shown in Figure: 2.1, is ResNet50. It is special because it also uses skip connection with three consecutive layers. The three layers consist of an initial 1×1 layer, a subsequent 3×3 layer, and one more 1×1 layer. ReLU

serves as the activation function for each layer, which has varying depths based on its unique configuration. In case of ResNet50 it consists of sixteen such three-layer blocks in various configurations, an initial 7×7 convolution layer and a fully-connected layer, both common to all, that serves as a classifier in the end. Its name comes from having total of 50 layers. ResNet50 produces a output feature of $7 \times 7 \times 2048$.

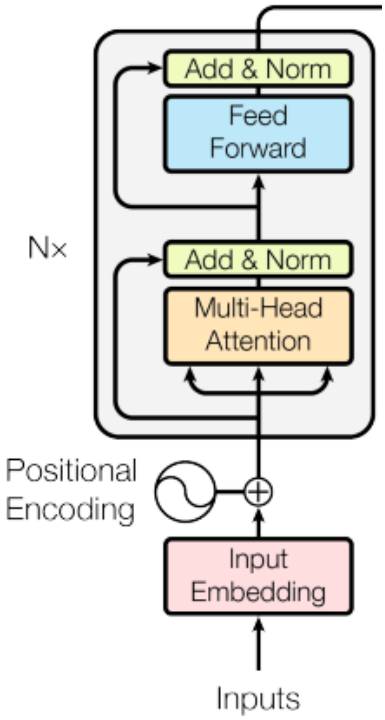


Figure 2.3: Transformer encoder [Vas+17]

For models that use transformers they mostly work with the encoder which gives it a worth to look at. Transformers were first introduced in 2017 [Vas+17] and since then there have been some promising models based on this architecture. The encoder architecture consists of a normalization layer, a Multi-Head Attention layer, another normalization layer, and finally a Multi Layer Perceptron (MLP). The part of interest here is the Multi-Head Attention layer. A Multi-Head Attention layer takes in three inputs: query, key and value with dimension of query and key sharing same dimension of d_k and values of dimension d_v. The attention function is computed on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V going into the equation for attention:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}}) \cdot V \tag{2.1}$$

2.2.2 Methods

This subsection discusses state of the art models along with some baseline models which are old but still relevant and using Birds-Eye-View representation.

Lift, Splat, Shoot [PF20] delves into the idea of using multi-view images with camera intrinsic and extrinsic available to project 2D image features into 3D. In lift, say n images $X_i \in \mathbb{R}^{3\times H\times W}$ with camera extrinsic $E \in \mathbb{R}^{3\times 4}$ and intrinsic $E \in \mathbb{R}^{3\times 3}$ and let p be image coordinate at (h,w) with d as depth coordinate defined in discrete steps D creating a large

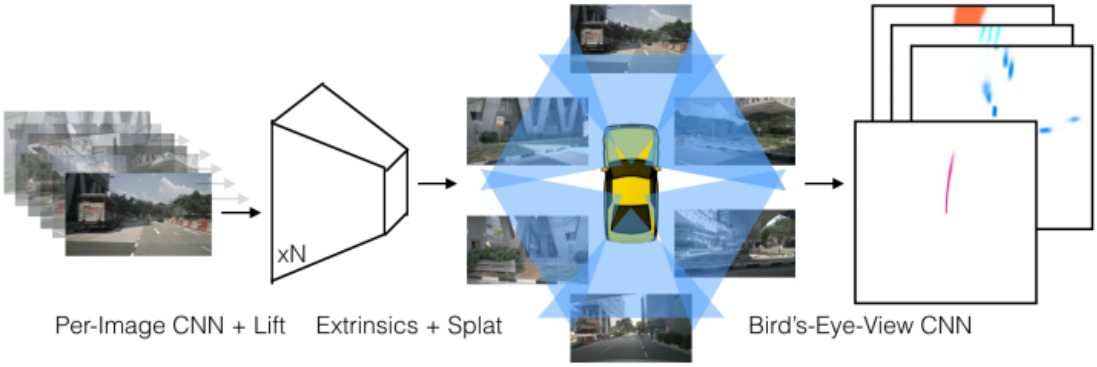


Figure 2.4: using multiple view images and camera intrinsic and extrinsic the 2D features are "lifted" and all the views frustums are "splat" for downstream task of BEV detection or planning [PF20]

point cloud of dimension $D \times H \times W$ for each image view (where H and W are image height and width). For each image coordinate p, the network predicts a context $\mathbf{c} \in \mathbb{R}^3$ and a distribution over depth $\alpha \in \Delta^{|D|-1}$. The feature $\mathbf{c}_d \in \mathbb{R}^C$ associated to point p_d is then defined as the context vector for pixel p scaled by α_d as seen in Figure: 2.5:

$$\mathbf{c}_d = \alpha_d \mathbf{c} \tag{2.2}$$

The network can predict a distribution depending on the confidence about the context. In case of ambiguous depth it can place the context over the entire depth or place it at a particular place in birds-eye-view representation. The splat is based on pointpillars [Lan+19] architecture where the different frustums created in previous process of "lift" can be "splat". Using infinite height "Pillars" as voxels, sum pooling is performed for every point to create $C \times H \times W$ tensor that can be treated by a CNN head for birds-eye-view inference as can be seen in Figure: 2.4.

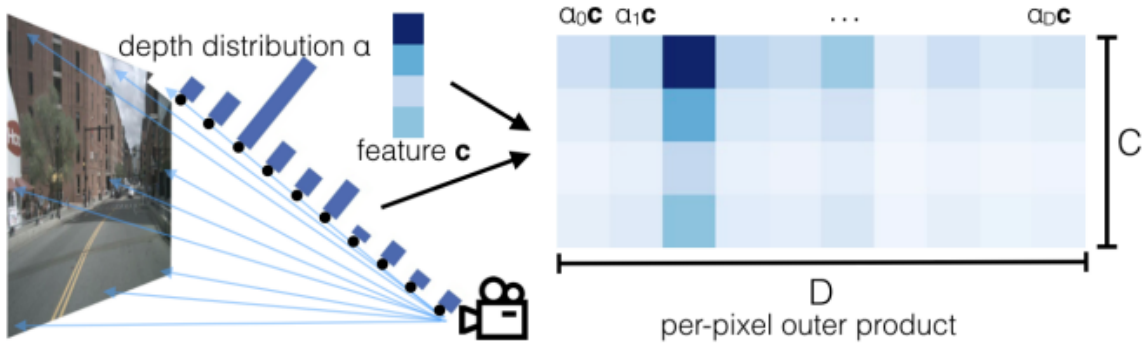


Figure 2.5: visualization of "lift" step in LSS [PF20]

BEVDepth [Li+23] builds up on LSS by adding explicit depth supervision using point cloud because they found that the implicit depth training was giving ambiguous depth. This point was verified by using constant random depth tensor while training and the resultant performance drop was 3.7% in mAP. To fix the depth, point clouds are used to explicitly train the depth module given the extrinsics of camera and LiDAR and intrinsic of camera.

To make the Depth Net more robust, as can be seen in Figure: 2.6, the camera intrinsic and extrinsic are also given to Depth Net to make it general to different field of views. The dimensions of camera are scaled up using Multi Layer Perceptron (MLP) and used to re-weight the image features with Squeeze and Excitation module [HSS18]. Finally, the camera

extrinsics are concatenated to intrinsics to give overall camera aware depth using image features. Depth Refinement module is new addition to aggregate features along the depth axis while the depth prediction confidence is low and performs rectification.

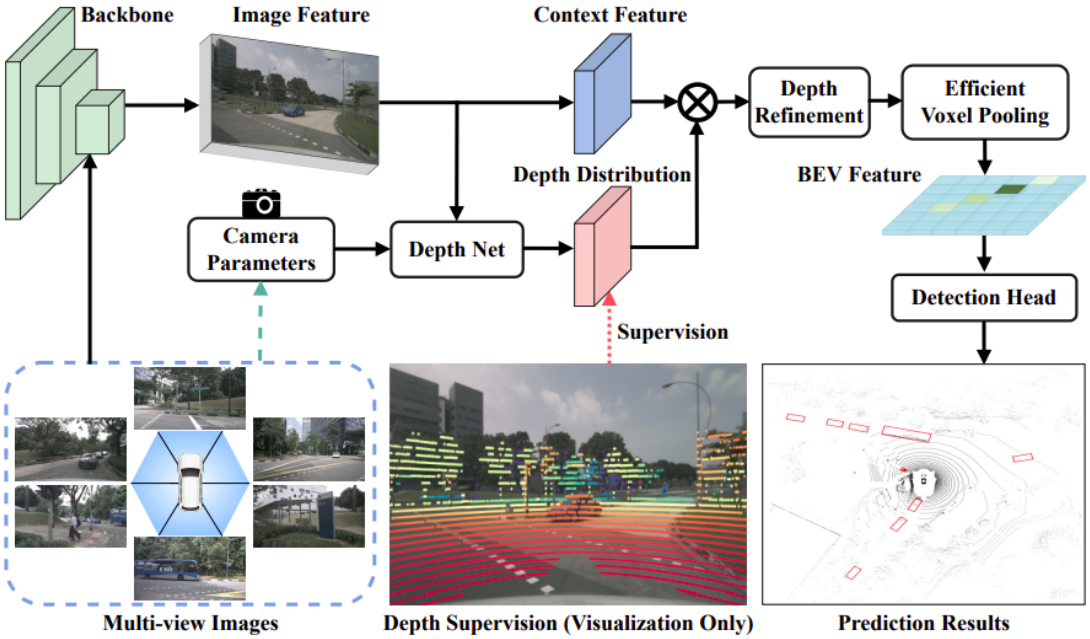


Figure 2.6: BEVDepth architecture [Li+23]

In comparison to LSS, BEVDepth performs "splat" using a more efficient voxel pooling by identifying the computationally inefficient "cumsum trick" which involves sorting and cumulative sum operations. By taking advantage of parallelism offered by CUDA, they are implemented by assigning each frustum a CUDA thread that adds features to its own BEV grid. This reduced the training time of model significantly in comparison to LSS.

The final BEV features are then fed to CenterPoint [YZK21] Head. CenterPoint follows the fact that 3D object detection is lot harder than 2D because with additional degrees of freedom in orientation, it makes box based models difficult to enumerate different orientations. To solve that problem, CenterPoint tracks objects rather as points than 3D box. It first detects the center of the object using keypoint detector and then regress the other outputs such as dimensions of box and it's orientation.

As can be seen in Figure: 2.7, the model gets map view features from 3D backbone but in the case of BEVDepth it's the BEV map view that is generated after "splat" operation of frustum. In the first stage, the it predicts centers and 3D boxes by having separate heads available to generate center heatmap and regression. The center heatmap plagues from sparse data since objects like cars are generally together leaving a lot of space empty and to counter that the ground truth heatmap have centers enlarged to get more supervision from nearby pixels.

Further second stage extracts additional point features from the map view features. These are the centers of faces for each predicted bounding box. Since top face, bottom face and box center have same center in BEV, only box center is considered making total 5 centers. These are concatenated and passed through MLP to get confidence score and bounding box refinement. Using these new methods BEVDepth is able to outperform LSS.

BEVFormer [Li+22] is another significant 3D perception model, building upon previous works that focus on multi-camera image processing and transformer-based models. It integrates both spatial and temporal information through a novel use of spatiotemporal trans-

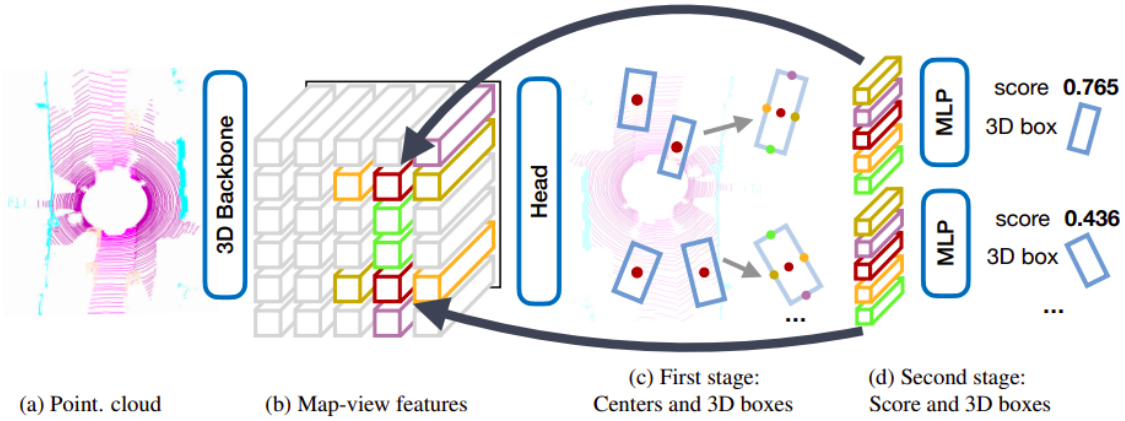


Figure 2.7: CenterPoint architecture with 2 stage detection [YZK21]

formers, allowing for the generation of accurate bird's-eye-view (BEV) representations crucial for tasks such as 3D object detection.

As seen in Figure: 2.8, input images are passed through backbone to get spatial features and BEV feature map from previous output is saved as temporal feature. The query Q is a grid shaped learnable parameter with $Q \in \mathbb{R}^{H \times W \times C}$ representing the BEV grid with ego coordinate at center. Point p at (h, w) pixel of this grid represents cell region in BEV plane.

Dealing with multiple views makes it computationally costly to use the vanilla multi head attention which was discussed in transformer encoder section (Figure: 2.3) so spatial cross attention is built based on deformable attention [Zhu+20] where the query Q_p will only interact with region of camera view images that are of interest. As seen in Figure: 2.8 (b), each query is lifted to be a pillar like query and N_{ref} number of points are sampled and projected onto the 2D views. Since the points may only be visible in some views so only those image views are considered with number of these images as V_{hit}. Now those 2D points are regarded as reference points for the original query Q_p and a weighted sum of the sampled features as the output of spatial cross-attention is performed as follows:

$$\text{SCA}(Q_p, F_t) = \frac{1}{|V_{\text{hit}}|} \sum_{i \in V_{\text{hit}}} \sum_{j=1}^{N_{\text{ref}}} \text{DeformAttn}(Q_p, \mathcal{P}(p, i, j), F_t^i), \tag{2.3}$$

where i indexes the camera view, j indexes the reference points, and N_{ref} is the total reference points for each BEV query. F_t^i is the features of the i-th camera view. For each BEV query Q_p,

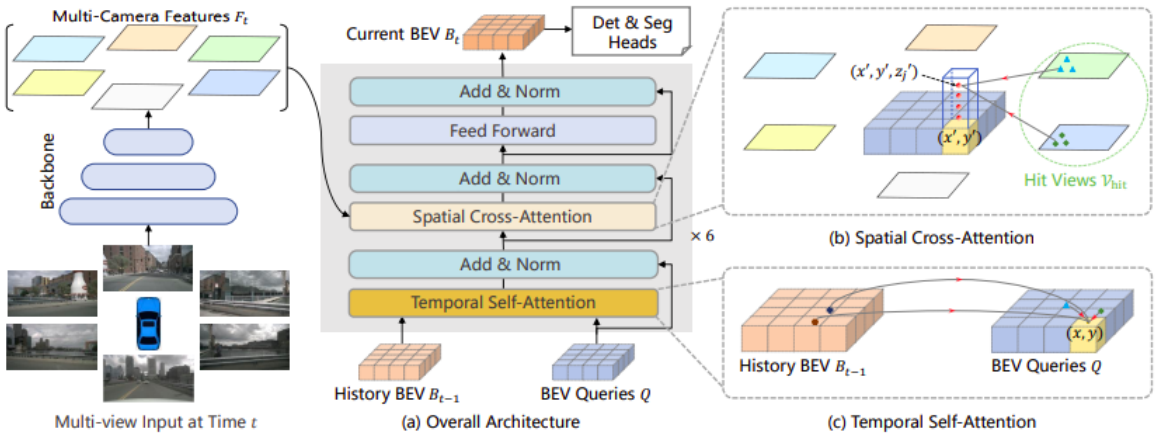


Figure 2.8: BEVFormer architecture with both spatial and temporal features [Li+22]

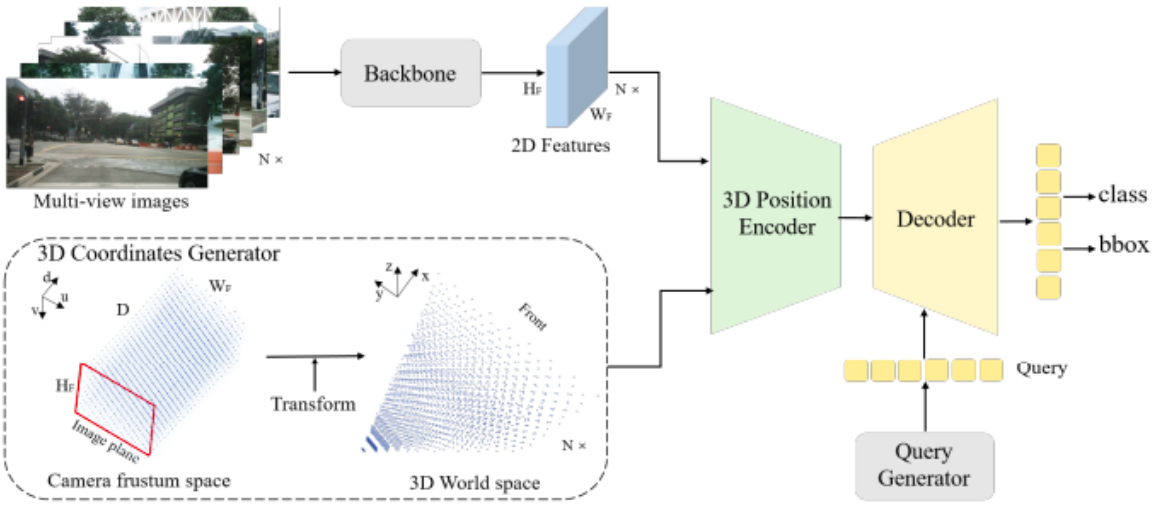


Figure 2.9: PETR architecture with 3D positional encoder and decoder [Liu+22]

we use a project function $\mathcal{P}(p,i,j)$ to get the j-th reference point on the i-th view image.

In addition to the spatial attention, temporal information is utilised to get better understanding of the environment. This is particularly useful in getting velocities going from one frame to another and helps with avoiding occlusions. Given the history BEV feature first it is aligned with the Query to make the features at the same grid correspond to same real world-location. However in real world, the objects that can travel have moved introducing offsets and so Deformation Attention layer is reused as follows:

$$\text{TSA}(Q_p, \{Q, B_{t-1}\}) = \sum_{V \in \{Q, B_{t-1}\}} \text{DeformAttn}(Q_p, p, V), \tag{2.4}$$

Where Q_p denotes the BEV query located at $p = (x, y)$ and previous BEV feature and Query are concatenated which is useful for starting sequence since there are no previous BEV features.

Another model which uses transformers style 3D object detection is "position embedding transformation (PETR) for multi-view 3D object detection" [Liu+22]. It comes as a improvement to DETR3D [Wan+22] with complains addressed by the author about how the predicted coordinates of reference point be not that accurate, making the sampled features out of the object region, since only the features at projected points are collected it fails to perform representation learning from a global view and complex feature sampling makes it out of range for practical applications.

As can be seen in Figure: 2.9, the multi-view images are passed through a backbone (ResNet50 [He+16] here) to extract 2D image features. Along that a 3D frustum from the camera's plane is discretized into a 3D meshgird. Since meshgrids are shared by multiple views, they are all brought to the global coordinate system using transformation matrices for transformation from camera to global coordinate frame and the global meshgrid is normalised using the maximum and minimum values for all three axis.

The 3D positional encoder architecture can be seen in Figure: 2.10.Given the 2D image features F^{2d} from backbone and 3D coordinates P^{3d}, the P^{3d} is first fed into a multi-layer perception (MLP) network and transformed to the 3D position embedding (PE). Then, the 2D features F^{2d} is transformed by a 1×1 convolution layer and added with the 3D PE to formulate the 3D position-aware features. Finally, the 3D position-aware features are flattened as the key component of transformer decoder.

The query generator that can be seen on Figure: 2.9 in this case is a set of learnable

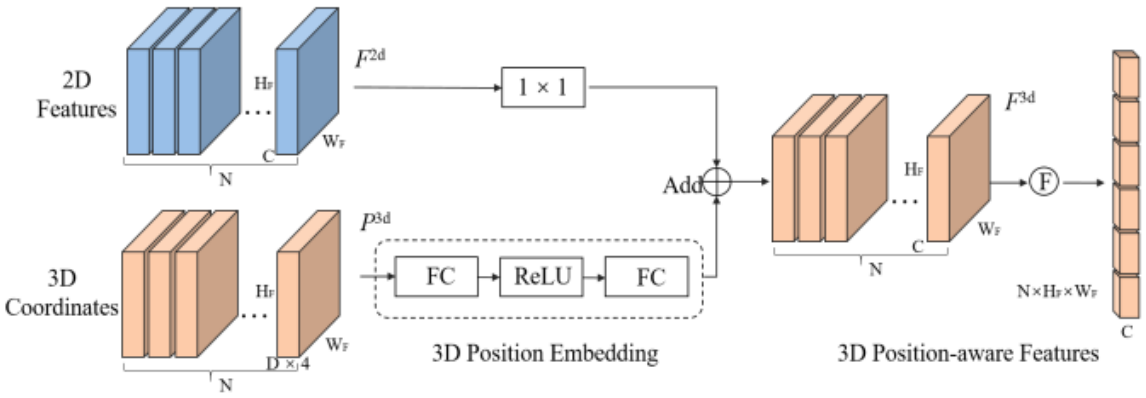


Figure 2.10: PETR 3D positional encoder [Liu+22]

anchor points in 3D world space with uniform distribution from 0 to 1. Then these anchor points are fed into MLP to generate the initial query used with decoder. The decoder used here comes from DETR [Zhu+20] with L decoder layers. The intial query is fed with 3D position-aware features generating next query. As it iterates, the updated query have high level representation that can be used for object detection.

Chapter 3

Dataset

This chapter deals with the TUMTraf Intersection Dataset [Zim+23] used in this work. Covered in following sections are surface details of the said dataset, different sensor modalities used and the breakdown of its use in BEVDepth. For more details on the dataset refer to the paper [Zim+23].

3.1 Location

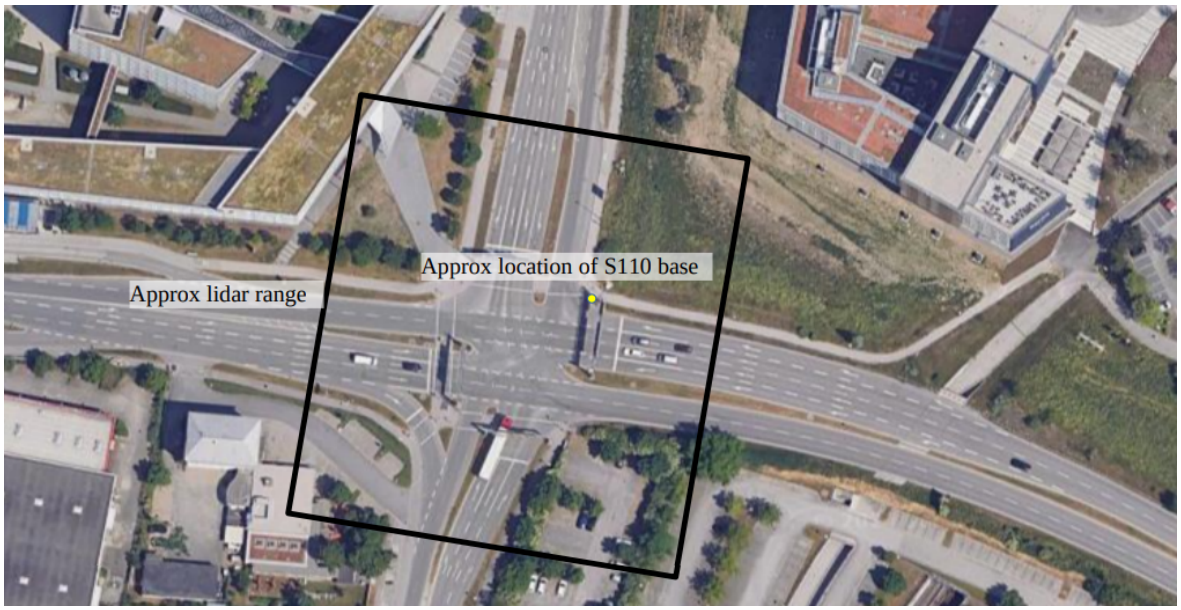


Figure 3.1: S110 base used as world co-ordinate system and approx LiDAR range. Imagery © 2024 GeoBasis-DE/BKG, Maxar Technologies, Map data © 2024

The dataset is based on the traffic that flows through the intersection S110 in Garching-Hochbrück, Germany. It is set up as Intelligent Transportation Systems (ITS) test bed which is part of a larger research project called Providentia++ [Krä+19] under the Chair of Robotics, Artificial Intelligence and Real-Time Systems at the TU Munich [CS20]. The testbed is set up on the gantry on Schleißheimer Straße (east-west direction) with intersection to Zeppelinstraße (north-south direction). Like a usual gantry, the direction boards and traffic light are present on it. On top of that there are cameras and LiDARs present on it getting live feed. There are 4800 LiDAR point clouds provided in this dataset with synchronized camera im-

ages with extrinsic and camera intrinsic calibrated data between the LiDAR and camera. The dataset used in this work is the 2nd release which uses data from two cameras and two LiDAR.

3.2 Roadside Setup

Figure 3.2: S110 gantry sensor setup. Image capture: Sep 2023 © 2024 Google

The S110 gantry is home to various different modalities sensors like RGB cameras, LiDARs, event based camera, radar etc. Like mentioned previously, for this work two RGB cameras , South 1 and South 2 respectively, are used with two LiDARs which are south LiDAR and north LiDAR. In Figure 3.2 the previously mentioned sensors are marked.

Since BEVDepth used nuScenes dataset, the point cloud for a sample is collected from top of the car. Similarly, TUMTraf Intersection dataset provides the opportunity to use single point cloud by merging the south and north LiDAR point clouds into one in the frame of south LiDAR.

The left sided base of S110 gantry (as can be seen in Figure 3.2) acts as the global and ego coordinate base in this work. The dataset provides the transformation between base to the LiDARs and further to the cameras. These transformations are used to change the coordinate system for ground truth bounding box, projection of LiDAR point cloud to a camera etc.

3.3 Sensors

With a wide array of sensors available in the market, the TUMTraf-I dataset relies mostly on the commonly used sensors in industry. This include LiDARs, RGB cameras, radar, GPS etc but for the scope of this work LiDARs and cameras are used.

3.3.1 Camera

Cameras have been the most reliable sensors since its invention. With advances in digital cameras, the camera technology has become one of the cheapest with respect to other sensors. This is one big reason why cameras play a key role in autonomous vehicles today. The quality and quantity of information that it brings along gives it the center stage in most of the computer vision research, as is the case here. A basic monocular camera provides an image with usually three channels of red, green and blue with a fixed size of 2 dimensional array. In TUMTraf-I dataset the images are of size 1920×1200.

Figure 3.3: Basler ace acA1920-50gc cameras [Bas24]

Since images represent the environment in viewing sense, they are incredible powerful in detecting objects in environment since humans too get most of their sensory input via eyes. Cameras comes with their biggest con which is the loss of depth information. Since the environment are in three dimensions, it would be nicer to have richer information about it also in three dimension but cameras work on the principle of pin-hole camera which projects the environment into a two dimensional image. Since camera relies on external light, night, fog and rain brings additional challenges since they either block the light from reaching the camera and in case of night there are not many light sources available.

The camera used in TUMTraf-I dataset is Basler ace U acA1920-50gc with Sony IMX174 sensor and 8mm lens. A prototype can be seen in Figure 3.3

3.3.2 LiDAR

LiDARs (Light Detection and Ranging) work on the principle of time of flight as seen in Figure 3.4. It produces a laser beam which goes into the environment and returns back to LiDAR. The time taken between the flight can be used to pin point the surface it is reflected from as the beam travels with the speed of light. The formula for time of flight is $d = ct/2$, where c is the speed of light, t is the time of flight and division by 2 is done because the path is travelled twice. Using LiDAR a point cloud can be generated which contains sparse points

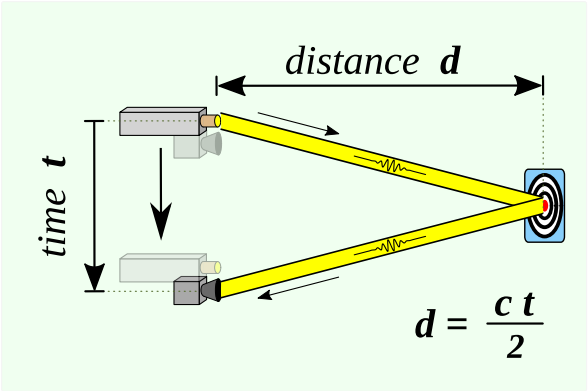


Figure 3.4: Time of flight

of the three dimensional environment as (x, y, z) coordinates and additionally intensity if needed. Since the information collected is three dimensional, it can be used to train models to predict good depthmap and is used as such in this work.

Figure 3.5: Ouster OS1-64 (gen. 2) [Ous24]

Reason for its low adoption can primarily come from its staggering cost. It is cheaper to have multiple cameras than single LiDAR. Also the big size makes it unwanted in autonomous driving scenarios. The laser beams requires that the surface it hits be Lambertian, meaning it reflects diffusely. A counter example to Lambertian surface is mirror which reflects all the light incident on it further concurrently and does not disperse it in all directions. This effect can also be seen in rain and fog where water droplets would either reflect it or let it pass causing the point cloud to not have accurate information in that area. Some of the drawbacks can be complemented by a camera and therefore together these two sensors are used in many works to get a better understanding of the three dimensional surroundings. The LiDARs used in TUMTraf-I dataset is Ouster OS1-64 (gen. 2) as NORTH LIDAR and SOUTH LIDAR.

3.4 Dataset composition

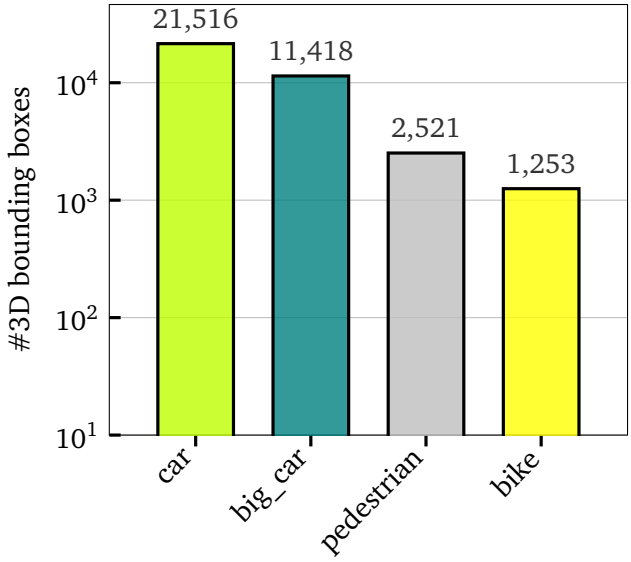


Figure 3.6: Different class distribution in TUMTraf-I Dataset

The TUM Traffic Intersection dataset [Zim+23] is the second release in line of various infrastructure view datasets. With total of 2400 samples each sample has two RGB images and two LiDARs (also available as one single fused point cloud) and a label file in *OpenLABEL* format. The data is collected using two Ouster OS1-64 (gen. 2) and two Basler ace acA1920-50gc cameras with Sony IMX174 sensor. These sensors are based on the S110 gantry at Garching-Hochbrück as part of Providentia++ testbed.

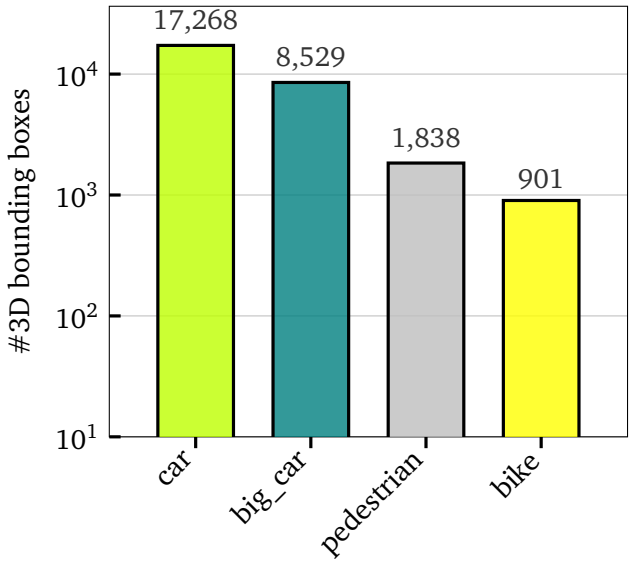


Figure 3.7: Different class distribution in TUMTraf-I training set

The dataset is split in 80-10-10 into training, validation and test set. For a total of 2400 samples, training set has 1920, validation set 240 and testing set 240 samples respectively. The dataset contains annotation of 10 classes which are CAR, VAN, BICYCLE, PEDESTRIAN, MOTORCYCLE,TRAILER, TRUCK, BUS, EMERGENCY_VEHICLE, and OTHER. For the pur-

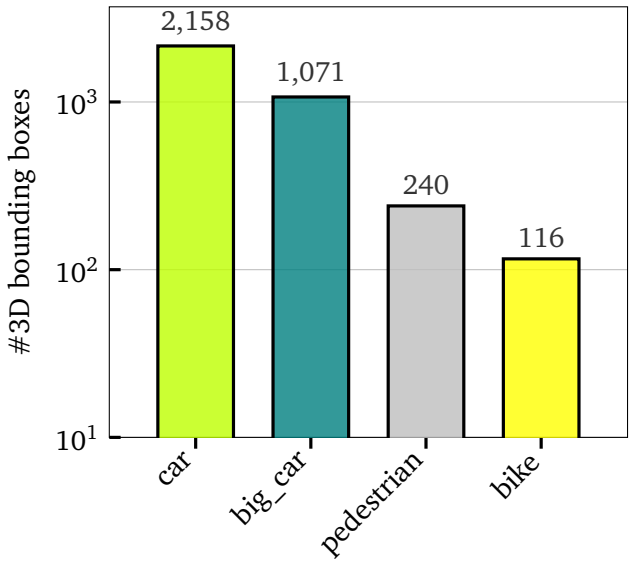


Figure 3.8: Different class distribution in TUMTraf-I validation set

pose of this work, similar objects were clubbed together to get more general model. CAR and OTHER becomes new class car where OTHER was found using custom created function *find_image* in *scripts/a9_explorer.py* to be a pickup truck as seen in Figure 3.10. Similarly, TRAILER,TRUCK,VAN,BUS and EMERGENCY_VEHICLE packaged together as big_car. MOTORCYCLE and BICYCLE were clubbed together into bike and remaining PEDESTRIAN as pedestrian. Using the function *list_categories* in *scripts/a9_explorer.py*, the statistics for the new class distribution was generated as seen in the Figure 3.6.

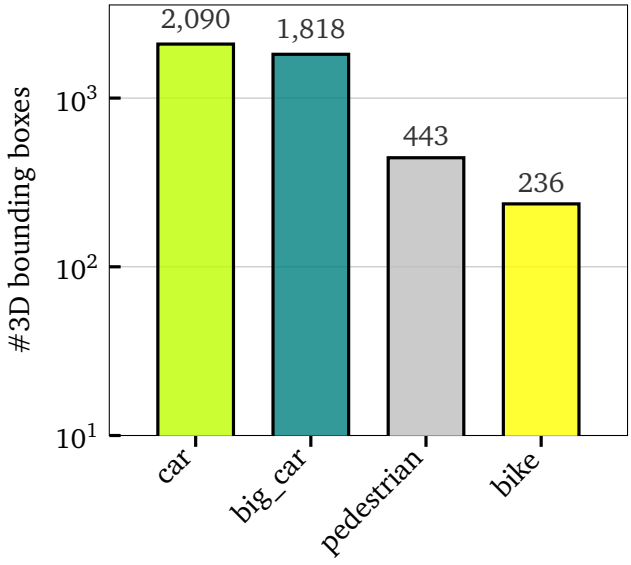


Figure 3.9: Different class distribution in TUMTraf-I test set

As can be seen from Figure: 3.7 to Figure: 3.9, the distribution of different classes are very consistent among the different splits. This is not true in the case of the original 10 classes of TUMTraf dataset. As seen in using the function *list_categories* in *scripts/a9_explorer.py*, the relative distribution in percentage can be seen in the Table 3.1.

Figure 3.10: OTHER object turns out to be American pickup truck

Split	car	big_car	pedestrian	bike
All	58.61%	31.10%	6.87%	3.41%
Train	60.51%	29.89%	6.44%	3.16%
Validation	60.20%	29.87%	6.69%	3.24%
Test	45.56%	39.63%	9.66%	5.14%

Table 3.1: Split among classes in percentage

Chapter 4

Implementation

This chapter delves into the implementation of methods to make BEVDepth work with TUMTraf-I dataset. BEVDepth is taken as base model with changes made to dataset, visualization , TUMTraf-I dataset converter etc. BEVDepth uses libraries like MMDETECTION, MMCV, MMDETECTION3D etc along with PYTORCH, so this work will continue with them in extension.

4.1 Data converter

BEVDepth works with Nuscenes [Cae+20] by taking out relevant information and pickling it into a .pkl file which PYTORCH dataloader could use as database for dataset. This brings the need for a converter because the dataset used in this work is TUMTraf-I which stores its sample in json file. The dataset follows the ASAM OpenLABEL [Hag20] format which is a push to make a more standardized approach in Autonomous Driving Systems (ADS). The decision to have the format in json is really good because there is good support available for it and minimum amount of work needs to be done to parse the information. Thankfully an OpenLABEL to nuScenes converter was already available by the master thesis work of an author [War23] but changes were needed to make it work with BEVDepth.

4.1.1 OpenLABEL format

```
{
    "openlabel": {
    "metadata": {
            "schema_version": "1.0.0"
        },
    "coordinate_systems": {
    "hd_map_origin": {...},
    "s110_base": {...},
    "s110_lidar_ouster_south": {
                "type": "sensor_cs",
                "parent": "s110_base",
                "children": [
                    "s110_camera_basler_south1_8mm",
                    "s110_camera_basler_south2_8mm"
                ],
```

```
                "pose_wrt_parent": {
                    "matrix4x4": [
                        0.99011438,
                        0.13828977,
                        0.02344061,
                        -2.36963586,
                        -0.13753536,
                        0.99000475,
                        -0.03121898,
                        16.19616412,
                        -0.02752358,
                        0.02768645,
                        0.99923767,
                        8.62,
                        0.0,
                        0.0,
                        0.0,
                        1.0
                    ]
                }
            },
    "s110_camera_basler_south1_8mm": {...},
    "s110_camera_basler_south2_8mm": {...}
        },
    "frames": {
        "559": {
        "frame_properties": {
            "timestamp": 1653330115.0101109,
            "point_cloud_file_name":
            "1653330115_010110958_s110_lidar_ouster_south.pcd",
            "weather_type": "RAINY",
            "time_of_day": "NIGHT",
            "transforms": {
                "s110_base_to_hd_map_origin": {
                    "src": "s110_base",
                    "dst": "hd_map_origin",
                    "transform_src_to_dst": {
                        "matrix4x4": [...]
                    }
                },
                "s110_lidar_ouster_south_to_s110_base": {
                    "src": "s110_lidar_ouster_south",
                    "dst": "s110_base",
                    "transform_src_to_dst": {
                        "matrix4x4": [...]
                    }
                }
            }
        },
    "object":{
```

```
    "ca9dd650-5f5b-45c9-95bb-c603f22de9e3": {
                "object_data": {
                    "name": "CAR_ca9dd650",
                    "type": "CAR",
                    "cuboid": {
                        "name": "shape3D",
                        "val": [
                            11.349679609591497,
                            30.00590974843884,
                            -7.513740620204429,
                            0.0,
                            0.0,
                            -0.7887201679394877,
                            0.6147523864821567,
                            4.20795,
                            1.9358,
                            1.50595
                        ],
                        "attributes": {
                            "text": [
                                {
                                    "name": "body_color",
                                    "val": "black"
                                },
                                {
                                    "name": "occlusion_level",
                                    "val": "NOT_OCCLUDED"
                                }
                            ],
                            "num": [
                                {
                                    "name": "number_of_trailers",
                                    "val": 0
                                },
                                {
                                    "name": "num_points",
                                    "val": 9
                                }
                            ]
                        }
                    }
                }
            },
            ...
        }
        }
        },
    "streams": {
            "s110_camera_basler_south1_8mm": {
                "description": "Basler RGB camera",
```

```
                "uri": "",
                "type": "camera",
                "stream_properties": {
                    "intrinsics_pinhole": {
                        "width_px": 1920,
                        "height_px": 1200,
                        "camera_matrix_3x4": [
                            [
                                1293.54,
                                0.0,
                                946.104,
                                0.0
                            ],
                            [
                                0.0,
                                1320.17,
                                621.989,
                                0.0
                            ],
                            [
                                0.0,
                                0.0,
                                1.0,
                                0.0
                            ]
                        ]
                    }
                }
            },
            "s110_camera_basler_south2_8mm": {...}
        }
    }
}
```


Listing 1: ASAM OpenLABEL format sample data

To understand the working of data converter, firstly the ASAM OpenLABEL format used in TUMTraf-I needs to be understood as shown in Listing 1. The format consists of 4 main branches of metadata, coordinate_systems, frames and streams where metadata is self explanatory, the other three are of interest here.

Branch coordinate_systems consists of the coordinate transformation between different sensors and coordinate frames as seen in Figure 4.1. It is to be noted that all the transformations are represented in as homogeneous matrices which means they contain information about the rotation and position to the next coordinate frame. Coordinate s110_base is placed at the corner of gantry as was discussed in chapter Dataset. Further, the coordinate 110_lidar_ouster_south is placed on the south LiDAR and since this work uses merged point cloud from both south and north, therefore only south LiDAR transformation is of value. From south LiDAR two transformations to s110_camera_basler_south1_8mm and s110_camera_basler_south2_8mm result in coordinate at the south camera one and two respectively.

Next branch frames consist of information about the current frame. This includes the timestamp of frame, weather type, time of day, point cloud name corresponding to this sam-

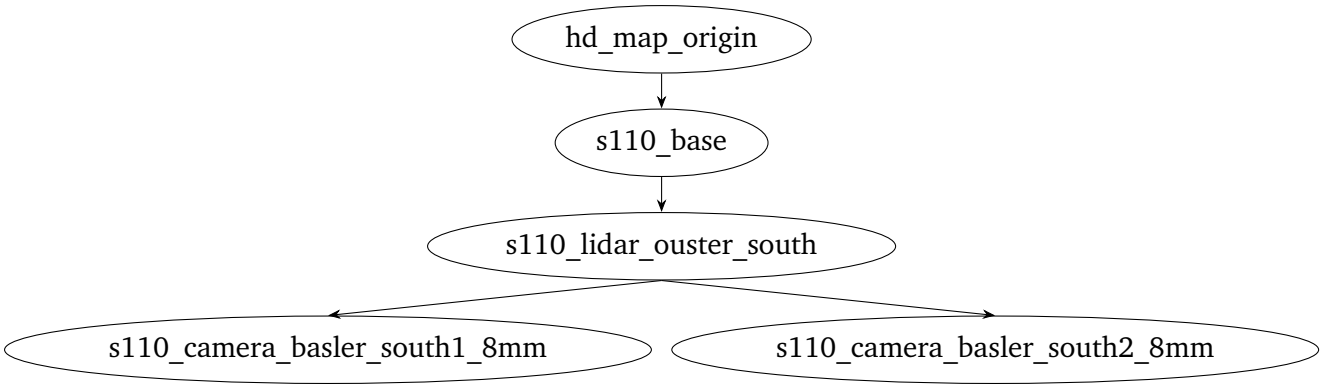


Figure 4.1: coordinate transformation tree

ple etc. Most important information in frames is that of the annotated objects. With a unique identifier provided to each instance of object in the sample, the information about the type of object is present which were discussed in Dataset chapter, the three dimensional information about the image is encoded as array sized ten where first three elements are the center of object, next four use quaternion to depict frame with respect to south LiDAR coordinate and last three gives the length, width and height of object. As can be seen in Figure 4.2, the bounding box with the center and orientation, height, width and length depicting the size of bounding box.

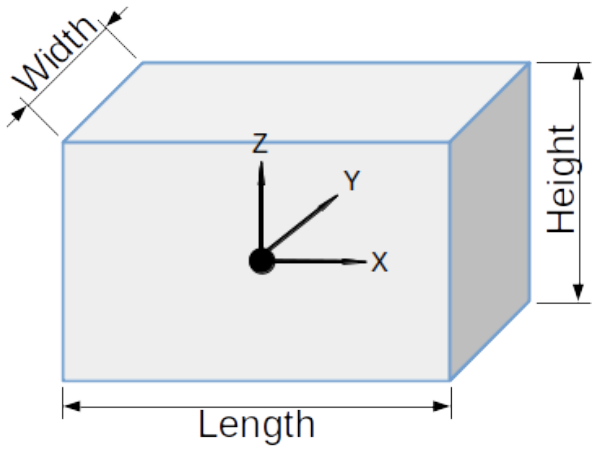


Figure 4.2: Basler ace acA1920-50gc cameras [Bas24]

Streams branch gives the camera intrinsic for both the south 1 and south 2 cameras along with the dimension of picture taken by them. The camera intrinsic is stored as a 3×4 matrix and is used to project three dimensional points to the image.

4.1.2 nuScenes format

The nuScenes format is quite complex with having to deal with different scenes, then different samples within the scene and various data of interest scattered around different classes as seen in Figure 4.3. NuScenes uses tokens to uniquely identify a sample and nuscenes-devkit [Inc19] provides access and manipulation of the dataset. From each sample the information about the image, it's size, image filename is required along with the camera intrinsic and extrinsic. Similarly with LiDAR point cloud filename along with it's extrinsic is taken.

Lastly, the annotation information for each instance of object such as it's translation, size and rotation are required.

4.1.3 OpenLABEL to nuScenes converter

As previously mentioned, a converter was already available but it further works were done to use it in this work. Firstly, since each sample in TUMTraf-I dataset doesn't have a unique identifier, timestamps are used to fill that gap. The converter can be found under *tools/create_traffix_data.py* which in turn uses *tools/data_converter/traffix_converter.py* that provides the class abstraction for the converter. User can specify which split of dataset they would want to convert along with the dataset root and path to save the converted dataset. Number of workers can be specified too but this wasn't implemented before and is thus a new addition. Multiprocessing is used to convert the point cloud from .pcd to .bin format. The function to convert was existing and using multiprocessing library in python it was extended to do conversion concurrently with multiple available cores.

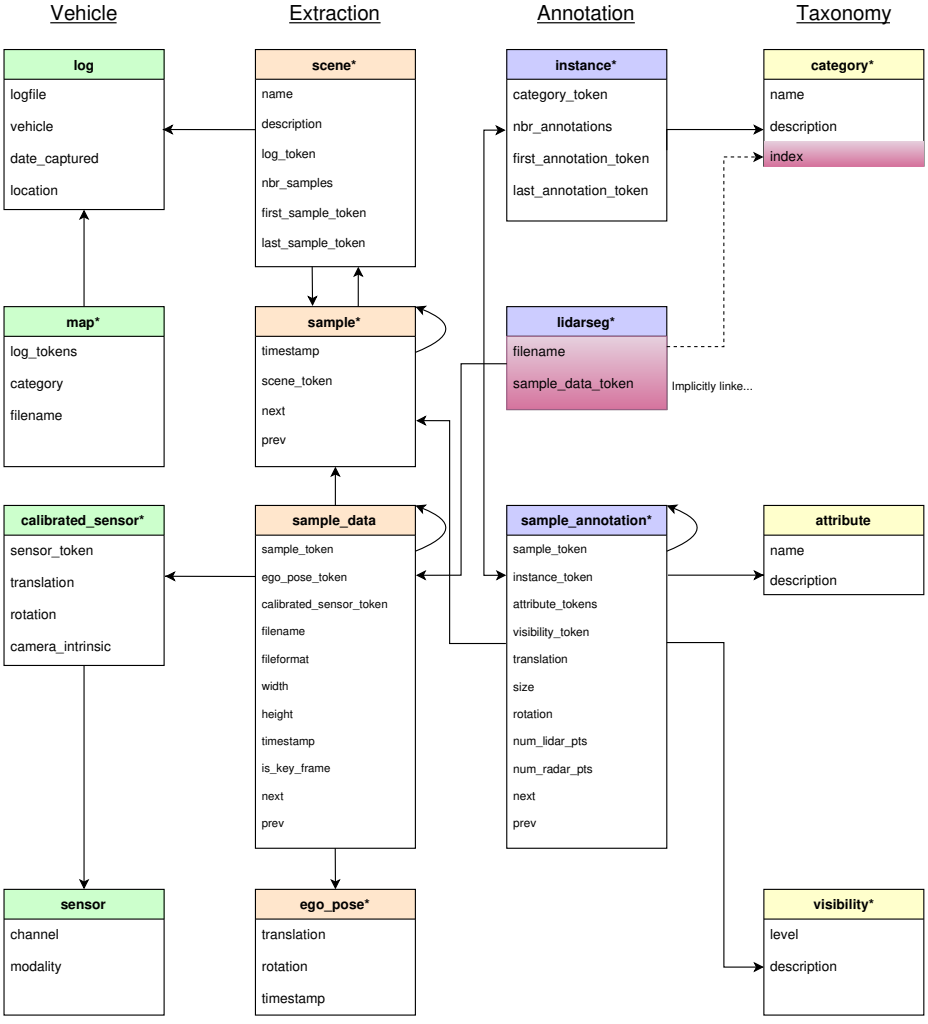


Figure 4.3: nuScenes schema

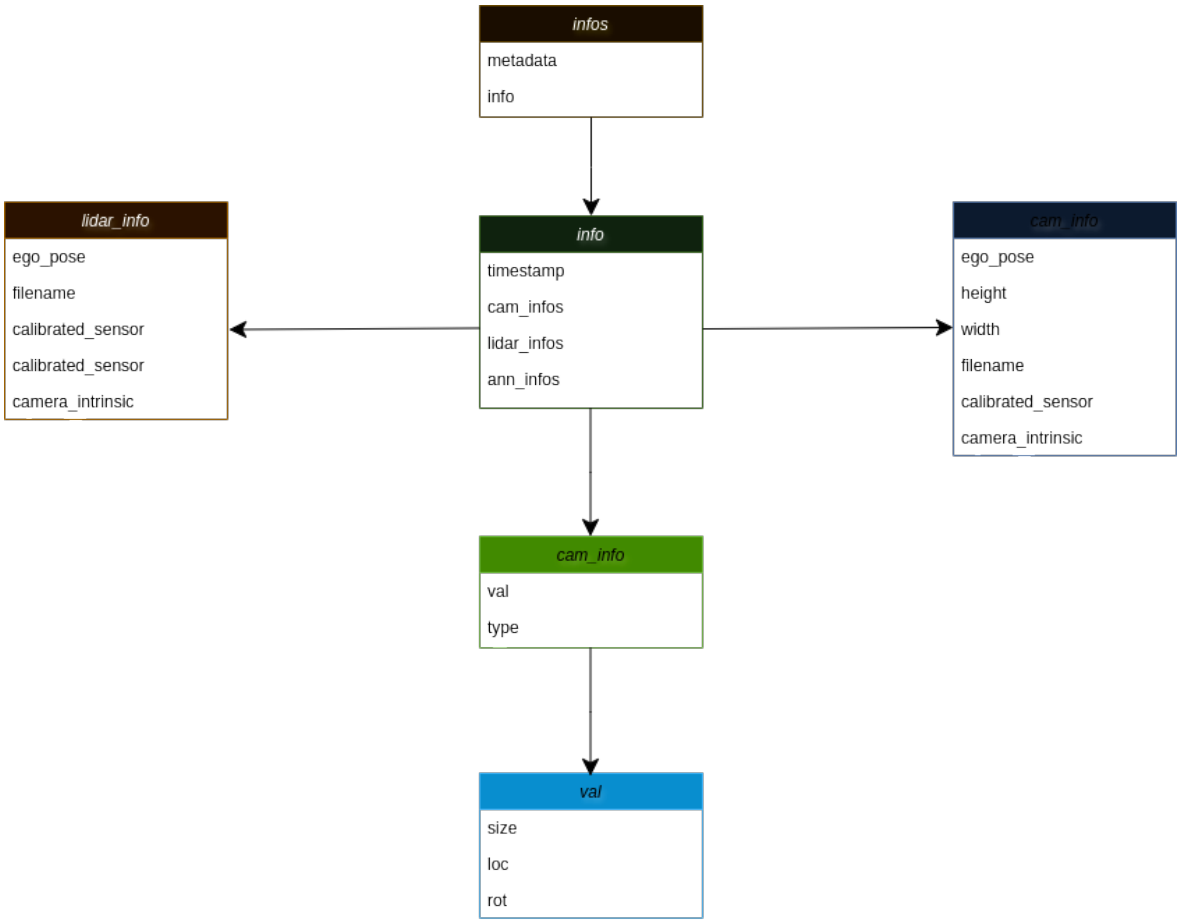


Figure 4.4: Converted TUMTraf-I data into nuScenes format

For each split, the converter makes a list of all the sample labels which contain all the necessary information about the sample. Along with it lists for different view images are created which would be then stored in current sample. A symbolic link to new dataset from old dataset is made for images as they remain the same and they do not need to be copied twice. The core work is done in the function *_fill_infos* where the json label file is read and data extracted. Since TUMTraf-I dataset is placed on a gantry, there isn't any movement between different sensors and coordinate bases. So all the transformations between the coordinate systems are static which means it doesn't change from one sample to another. Using this fact, *lidar2ego*, *south1intrinsics*, *south2intrinsics*, *south12lidar*, *south22lidar*, *south12ego*, *south22ego* and *ego2global* transformations are taken once from a sample and populated for all the samples where ego is set to S110 base and lidar is set to south LiDAR, south 1 and south 2 are self explanatory. The transformation *south12ego* is found in equation 4.1 where all of the matrices belong to $\mathbb{R}^{4\times4}$ and similar applies to *south22ego*.

$$south12ego = lidar2ego * south22lidar \tag{4.1}$$

All the data about camera and LiDAR are read from the json file for each sample. The annotation data is to be noted here since the data in annotation is present in the frame of reference of south LiDAR and since nuScenes and BEVDepth expects data in ego frame which in this case is S110 base. Therefore, both the center of annotated object and the coordinate frame of object at it's center needs to be rectified. Fixing the center is easy as shown in equation 4.2 but for the case of rotation first rotation matrix is made out of quaternion, its multiplied with *lidar2ego* to get new frame rotation, it is converted back to quaternion

and saved. The length, width and height are invariant to transformation so they remain unchanged. The new simplified schema can be seen in Fig: 4.4 and in comparison to nuScenes this is quite simplified since only relevant data is considered.

$$loc = lidar2ego * old\_loc \tag{4.2}$$

4.2 Dataset for BEVDepth

Idea for models like BEVDepth is to be independent of dataset but unfortunately this is not the case as the dataset for BEVDepth is heavily intertwined with nuScenes. This requires heavy alterations made to the dataset so a copy of the nuScenes dataset is used as base to make the file *a9_2_det_dataset.py*. First change to be made is to change the object classes that are considered by nuScenes to TUMTraf-I dataset classes. Like discussed in chapter of Dataset, the original classes CAR, TRUCK, BUS, TRAILER, VAN, and EMERGENCY_VEHICLE ,PEDESTRIAN, MOTORCYCLE, BICYCLE and OTHER are mapped to car, big_car , pedestrian and bike, this is implemented here.

In the *__getitem__* function of PyTorch dataset inherited class, most of the information comes from *get_image* function and therefore required maximum change. After loading all the frames for the split from .pkl file made in converter, variable fields like *lidar_path*, *sensor2ego_mats*,*sensor2sensors_mats*, *intrin_mats*, *point_depth*, *timestamps*, *ego2global_rotation* and *ego2global_translation*. The nuScenes dataset had ground truth disabled for testing which called the function *get_gt* where again the input for annotation labels and some transformations were changed to match new dataset.

4.3 Metrics

BEVDepth utilizes the nuScenes devkit [Inc19] to evaluate the performance of model but it couldn't be adapted to run TUMTraf-I dataset since the classes of objects between them are different and dataset converter could only cover the data structure to a small part, the TUMTraf-I devkit [Zim+23] is employed. Since the devkit wasn't extended to nuScenes style results and ground truth information, the functions *parse_input_nuscense*, *parse_input_nuscense_pred* were written to get the ground truth and predictions to match the format TUMTraf-I devkit required it to be.

Additionally, mAP over all the predicted classes over all the different difficulties is used as validation metric to maximise during training of model, the function *val_input_nuscense_pred* is written to be used during training at the end of each epoch.

Another metric that is required for this work is to find the speed at which the trained model is able to predict output, the metric frames per second (FPS) is used. Similarly the amount of space is used for computing, VRAM as a metric is employed. Both of the said metrics were not available in TUMTraf-I dataset devkit, so a script *scripts/benchmark.py* was created to address that.

Model with checkpoint are loaded in evaluation mode and run through the test dataset with time before forward pass and after it logged. The CUDA GPU threads are synchronized before and after forward pass to avoid any hanging threads. With time logged after a number of warm-up period that user can mention, the FPS is found as total iterations exceppt the warm-up period over total time differences over those periods. Similarly, *nvidia-smi* library is used to find the VRAM used during idle and during the run thus using mean of it as the memory employed during the operation.

Chapter 5

Evaluation

In this chapter the focus is on evaluating and analyzing the results of this work. So far, the implementation of converter was discussed. Since the baseline was based on nuScenes, new configurations are needed to make it work for TUMTraf-I dataset. Then training on the new dataset ablation studies are conducted to see how good the model performs and how fast since speed is important for real time application. Moreover, qualitative results would enrich the discussion of conclusions and further works that can be done to improve the method.

5.1 Experimental Setup

To evaluate the baseline model with its ablation, the experiments use almost the same configurations as follows:

- **Image Size:** The TUMTraf-I dataset and originally nuScenes dataset are both downsized to 256×704 from original 1200×1920 and 900×600 respectively. This configuration would be changed during ablation
- **Point Cloud Range:** Since the ego body frame in case of TUMTraf-I dataset is chosen as the gantry's S110 base, the range is chosen with respect to this coordinate frame as follows:

$$\text{X–Axis} \in [-150, 157.2], \quad \text{Y–Axis} \in [-95, 161], \quad \text{Z–Axis} \in [-3, 6]$$

- **Voxel and Grid Size:** For TUMTraf-I dataset, the voxel sizes are

$$V_x = 2.4, \quad V_y = 2, \quad V_z = 9$$

  with Grid size coming out as 128×128×1 for Birds-Eye-features as follows:

$$H = \left\lceil \frac{x_{max} - x_{min}}{V_x} \right\rceil, \quad W = \left\lceil \frac{y_{max} - y_{min}}{V_y} \right\rceil, \quad D = \left\lceil \frac{z_{max} - z_{min}}{V_z} \right\rceil$$

- **Image configuration:** Before loading the image to any model, the image is normalised using the this parameter which give the mean and standard deviation of RGB pixels throughout the images in dataset. This is found for TUMTraf-I dataset to be:

$$\mu = [99.95, 108.08, 100.44], \quad \sigma = [64.65, 64.20, 63.17]$$

- **Hardware:** For the TUMTraf-I dataset, the models are trained on Nvidia RTX 3090 GPU

- **Backbone:** Backbone used in BEVDepth is modified LSS [PF20]. The backbone in it comes from Resnet [He+16] and neck SECOND FPN [YML18] and depth net created using squeeze and excite module [HSS18].
- **Depth Refinement:** Designed in BEVDepth to aggregate feaures along depth axis
- **Efficient Voxel Pooling:** Based on existing pointpillars [Lan+19] used in LSS [PF20], implemented in CUDA with threads assigned to each frustum for BEV grid in BEVDepth
- **Detection Head:** BEVDepth replaces the detection head to CenterPoint [YZK21] for 3D detections with backbone of Resnet [He+16] and Second FPN [YML18] as neck. *CenterPointBBoxCoder* used as decoder with score threshold 0.5 and downsize factor 1. Train and test configurations hold similar values along with minimum radius for non maximum suppression (NMS) calculated as circle encircling the Birds-Eye-View 2D object with post processing maximum size of 20 per class. Loss for classification is *GaussianFocalLoss* and for bounding box *L1Loss*
- **Optimizer:** We utilize the AdamW optimizer [LH17], with a learning rate dependent on batch size and number of GPUs with base of 1e-4 /32, weight decay rate of 1e-7. Besides, batch size of 1 is set for both train and validation set. Multi-step learning rate is implemented with epochs defined to decay learning rate to 10%. This helps with reducing loss when it plateaus.
- **Training Procedure:** Most trainings are configured to last 20 epochs with mAP over all classes and difficulties on validation split as monitoring parameter to save the best checkpoints

5.2 Metrics

Main goal of this work is to be able to run camera based methods to run realtime and this comes at a cost of accuracy vs speed. As will be seen in ablation studies, the general trend of increasing accuracy with decrease in speed would be the case here. To showcase speed the frames per second (FPS) is chosen as a metric.

5.2.1 Precision and Recall

Thought this metric is not used directly, it is still very required to find mean Average Precision (mAP). In 3D object detection, precision calculates the proportion of correctly identified positive instances, called True Positives or TP, to all the instances that the model classifies as positive as shown in equation: 5.1 with True Positives (TP) + False Positives (FP). Similarly, recall measures the proportion of correctly identified positive instances (TP) to all of the ground-truth positive instances in the sample which is True Positives (TP) + False Negatives (FN) as seen in equation:5.2.

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

In this work, True Positives (TP) are instances where the model correctly detects a class object in the image whereas False Positives (FP) are instances in which the model detects an object in the image when there actually is none. Finally, False Negatives (FN) are instances where the model fails to detect a class object that exists in the image ground truth.

5.2.2 Mean Average Precision

Mean Average Precision [HF17] is a commonly used metric to measure performance in the field of computer vision in general. Due to it's popularity in 2D object detection, it is well used in 3D object detection as well. Reason for the popularity of this metric goes to the fact that it shows the performance (accuracy) of the evaluated model by including both precision and recall into it's consideration.

Mean Average Precision takes up mean over all the classes available in the dataset. In the case of TUMTraf-I dataset, since the classes were clustered into four classes, the mean is taken over car, big_car, pedestrian and bike. The precision-recall curve is plotted for each class over each threshold and difficulty which in this case are easy, moderate and hard. The difficulty levels are divided based on the distance from the S110 base coordinate system. Easy being distance 0-40m, moderate being 40-50m and remaining until 64 as hard. The area under the graph is average precision and to find the average precision for the class, weighted average is taken over difficulties with sample number as weights as shown in the equation: 5.3

$$AP_{class} = \frac{N_{easy} \cdot AP_{easy} + N_{moderate} \cdot AP_{moderate} + N_{hard} \cdot AP_{hard}}{N_{total}} \tag{5.3}$$

Similarly, mean average precision is found over all the classes. To put it formally, for N number of classes, the mean Average Precision is calculated as shown in equation: 5.4.

$$mAP_{overall} = \frac{1}{N} \sum_{k=1}^{N} AP_k \tag{5.4}$$

Since mean average precision takes in consideration over various difficulties, threshold and classes while striking a balance between precision and recall, this metric is of high importance in autonomous driving. As already discussed in previous chapter, the TUMTraf-I dataset devkit is utilized to find this metric.

5.3 Quantitative Results

This section covers the results of training BEVDepth on TUMTraf-I dataset with ablation studies to see how different components are important for performance. Originally, BEVDepth is trained on nuScenes dataset for approx 20-24 epochs without any monitored metric to save best checkpoints. In this work the models were allowed to train for many more epochs but it was found that the best trained model lies in the 15-25 epochs range, therefore most of the models are trained up to 25 with overall validation mAP metric monitored to save the best checkpoints.

As seen in table: 5.1, various model training routine with different configurations are employed to maximise the performance of the model. The depth refinement column refers to BEVDepth's module for aggregating depth and it's context as discussed in previous chapter. Weight regularisation is applied with optimizer with it's value given in previous section and

Table	depth refinement	regularization	image size
Table 5.2	✓	✓	256×704
Table 5.3	✓	-	256×704
Table 5.4	-	-	256×704
Table 5.5	-	-	512×1408
Table 5.6	-	-	1024×1920

Table 5.1: aggregated results

AP@50	Overall	0-40m	40-50m	50-64m
CAR	54.54	54.38	55.67	57.78
BIG_CAR	51.28	46.57	52.31	55.57
BIKE	36.91	41.81	61.93	0.0
PEDESTRIAN	17.88	47.45	3.98	10.67
mAP	40.15	47.55	43.47	31.0

Table 5.2: Result on test set for model with regularization and depth refinement, image size 256×704

image size is the input image size to the model. The resultant mAP are all given in percentage unless stated otherwise.

Results in table: 5.2 is of model with weight regularisation whereas in table: 5.3 are without any weight regularisation, both models working with input image of size 256×704 to show difference.

As seen in table: 5.3 the overall mAP for TUMTraf-I dataset comes out at **46.57** which is 16% higher than the result from table: 5.2 with mAP 40.15. This seems counter intuitive since the results on test set should go down as regularization is removed from model but that seems not to be the case here.

This goes to show how the model didn't reach the stage of over-fitting and was not generalised enough. Possible solutions would be discussed in upcoming chapters but for the rest of the work, regularization is disabled.

Comparing results in table: 5.3 and table: 5.4, the overall mAP drops by 35.5%. This shows the importance of depth refinement module and how it is able to aggregate features along the depth axis and thus improving the model's performance.

From the results in table: 5.4 to table: 5.6, the image size is changed while keeping all other configurations the same to see how better image features would affect the results. As the general trend would suggest, better image resolution would lead to model having better image features and thus better the result. This is put together to be seen in table: 5.7, it is to be noted that the experiment is conducted over the whole test split of TUMTraf-I dataset keeping batch size of one and two view per batch (South 1 and South 2 images).

The overall mAP increases slightly from image size 256×704 to 512×1408 but a bigger increase seen from size 512×1408 to 1024×1920. This greater increase in accuracy comes

AP@50	Overall	0-40m	40-50m	50-64m
CAR	62.84	61.73	58.53	66.99
BIG_CAR	55.21	50.81	56.71	57.52
BIKE	44.93	39.30	83.79	3.08
PEDESTRIAN	23.3	56.57	12.24	66.00
mAP	46.57	52.10	52.82	48.4

Table 5.3: Result on test set for model without regularization and with depth refinement, image size 256×704

at the cost of both speed and memory usage. Speed drops dramatically and barely keeps up to 10 FPS for image size 512×1408 whereas it goes down by 50% to 5 FPS in case of 1024×1920. Similarly, the memory usage increases by 31% and 44% for doubling image resolution each time.

AP@50	Overall	0-40m	40-50m	50-64m
CAR	47.99	47.62	42.77	58.31
BIG_CAR	35.18	33.82	35.83	37.8
BIKE	21.18	21.47	56.73	0.0
PEDESTRIAN	15.72	48.57	3.03	0.0
mAP	30.02	37.87	34.84	24.03

Table 5.4: Result on test set for model without regularization and depth refinement, image size 256×704

AP@50	Overall	0-40m	40-50m	50-64m
CAR	47.80	51.48	40.70	52.58
BIG_CAR	34.28	31.17	37.57	35.53
BIKE	26.05	23.36	61.61	0.0
PEDESTRIAN	16.96	51.37	4.49	0.0
mAP	31.27	39.34	36.09	22.03

Table 5.5: Result on test set for model without regularization and depth refinement, image size 512×1408

AP@50	Overall	0-40m	40-50m	50-64m
CAR	60.56	62.61	48.65	60.92
BIG_CAR	44.50	44.51	39.40	49.74
BIKE	29.35	33.13	49.52	0.0
PEDESTRIAN	17.82	52.28	4.32	32.0
mAP	38.06	48.13	35.47	35.67

Table 5.6: Result on test set for model without regularization and depth refinement, image size 1024×1920

5.4 Qualitative Results

In this section, a variety of qualitative outcomes are presented with prediction from BEVDepth in South 1 and South 2 images along with Birds-Eye-View prediction and features. As can be seen in Figure: 5.1, top two images display predictions on images from camera view South 1 and South 2. Below it lies two ground truth images to compare the result of predictions. It can be seen one car each in image South 1 and South 2 are False Negative. The ground truth also includes the depth found using the LiDAR point cloud projection onto the image, the same is used during explicit training of depth.

The Figure: 5.2 shows the Birds-Eye-View feature of the same image. It can be seen how two frustums are formed which protrudes outwards from camera which is exactly how it was made to. This is the process of lifting the 2D image features to 3D and projecting it in Birds-Eye-View using the efficient voxel pooling operation with or without depth refinement module.

Image dimension	mAP@50	FPS (Hz)	VRAM (MB)
256x704	30.02	33	1800
512x1408	31.27	10	2360
1024x1920	38.06	5	3400

Table 5.7: Ablation on image size with model's inference speed and space occupancy

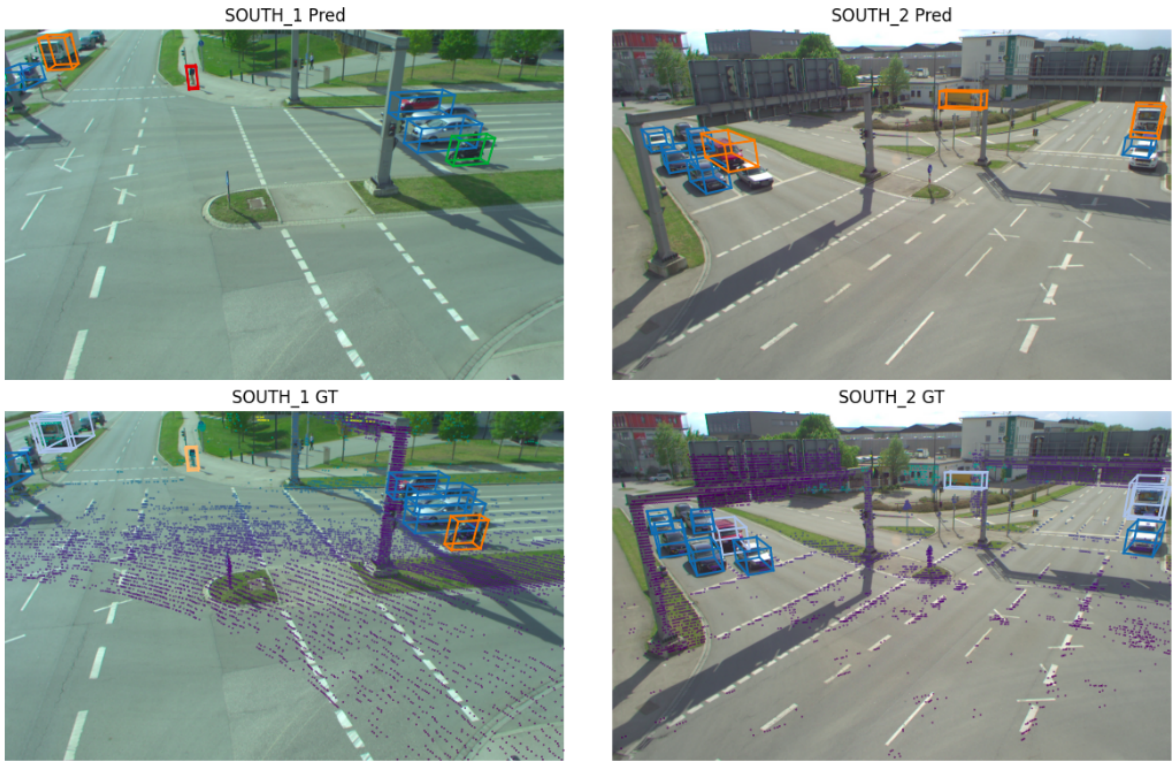


Figure 5.1: Example prediction

Finally, Figure: 5.3 shows the Birds-Eye-View image of object detection with ground truth in green and prediction in blue.

Another problem faced during training of the model is due to the missing ground truth label for object. As can be seen in Figure: 5.4, South 1 image contains a black car which has it's ground truth label missing. Needless to think, the model doesn't predict bounding box for it. The missing labels are particularly in region very close to gantry where LiDARs aren't able to measure perhaps giving the labeling tool no good measure for labeling.

Figure: 5.5 shows the BEV prediction image with missing car. It can also be noted, there is a truck with trailer shown in BEV image but since it is almost missing in the two images, there are no predictions available since BEVDepth is image based method and it does not have any point cloud information available .

As discussed in previous chapter, the classes of TRAILER,TRUCK,VAN,BUS and EMERGENCY_VEHICLE from TUMTraf-I dataset were clubbed together as big_car. This has been a problem for this class because of high variance of the dimensions among the original classes, the mean length and width calculated over all of them does not represent big_car correctly. This is because for NMS the radius r calculated might just be too big of too small which causes problem especially in case of truck with trailer as seen in Figure: 5.6. Since they are consecutively along each other all the time and many times NMS could through out perfectly good prediction. Even if both truck and trailer get predicted as in this case, the bounding box are quite wrong.

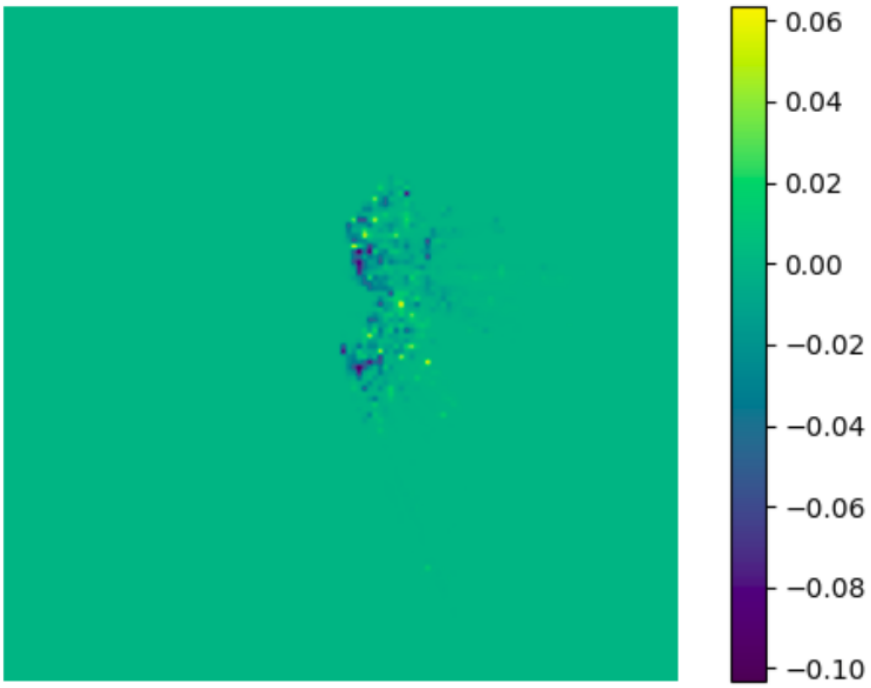


Figure 5.2: Example BEV features

As can be seen in the Birds-Eye-View prediction in Figure: 5.7, NMS in this case allowed three predictions with all of them giving wrong bounding box. Similarly are the cases where only one is predicted often in middle of both and covering none.

For the case of night, BEVDepth is unable to predict anything giving out all False Negative. This is seen in Figure: 5.8 with night time and raining on the street. If the BEV features are to be consulted in Figure 5.9, in comparison to day time features, the night time features are quite weak.

To get to the heart of the matter, the night images fed to the model can be seen in Figure: 5.10. The (a) and (b) sub-figure shows the two view images during night time after getting normalized to image configuration discussed in section: 5.1. sub-figure (c) is placed to show the difference in image features that daytime images have. In the whole dataset, the TUMTraf-I dataset is comprised of only 15% of night time samples but this alone isn't the only reason contributing to problem here. Since BEVDepth is strictly image based object detection model, having little to no features available during night time makes it hard for model to predict. Since mean and variance for whole dataset is skewed against the night time dataset, it could be probable to train another model only on night time data with mean and variance calculated only over the night time samples.

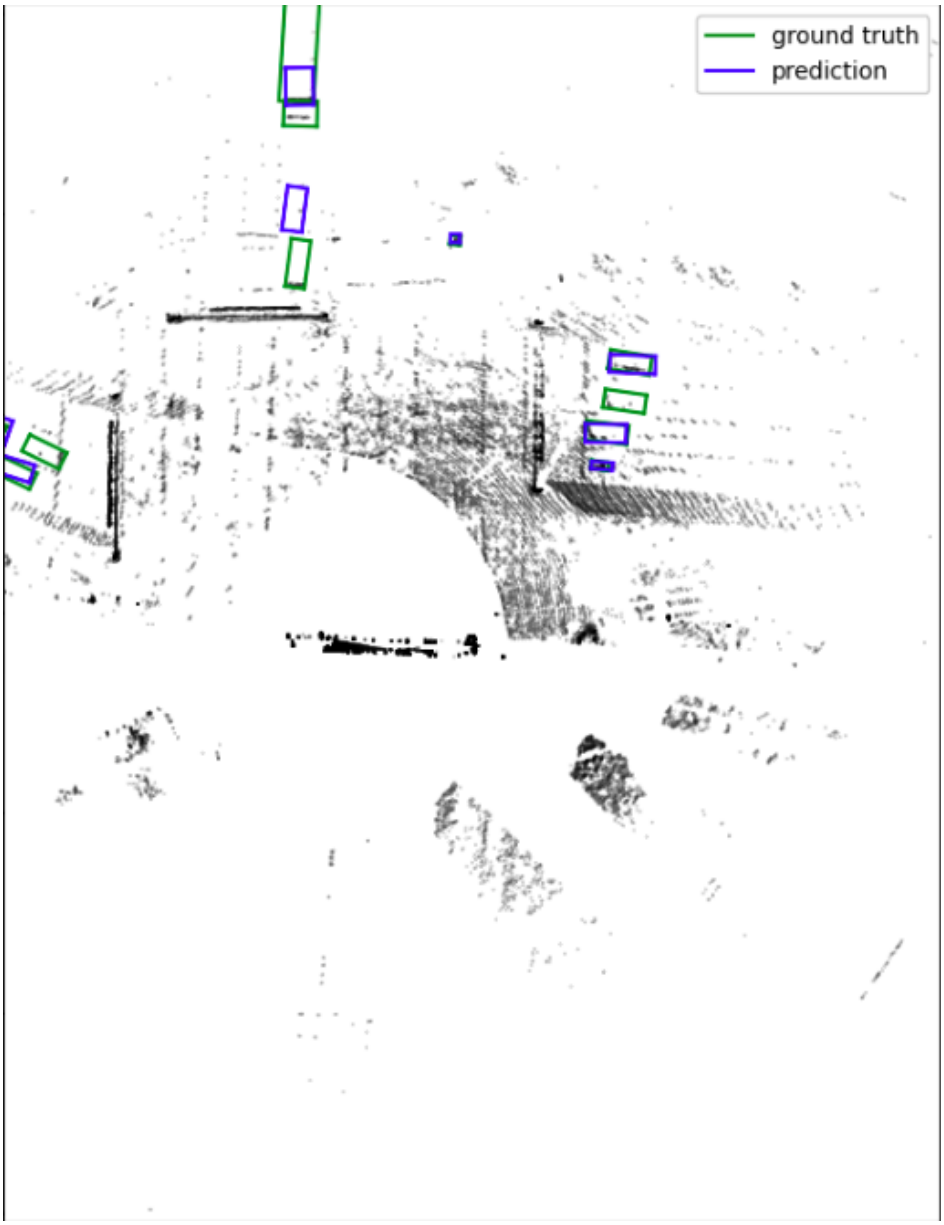


Figure 5.3: Example BEV prediction

Figure 5.4: Example of missing ground truth label

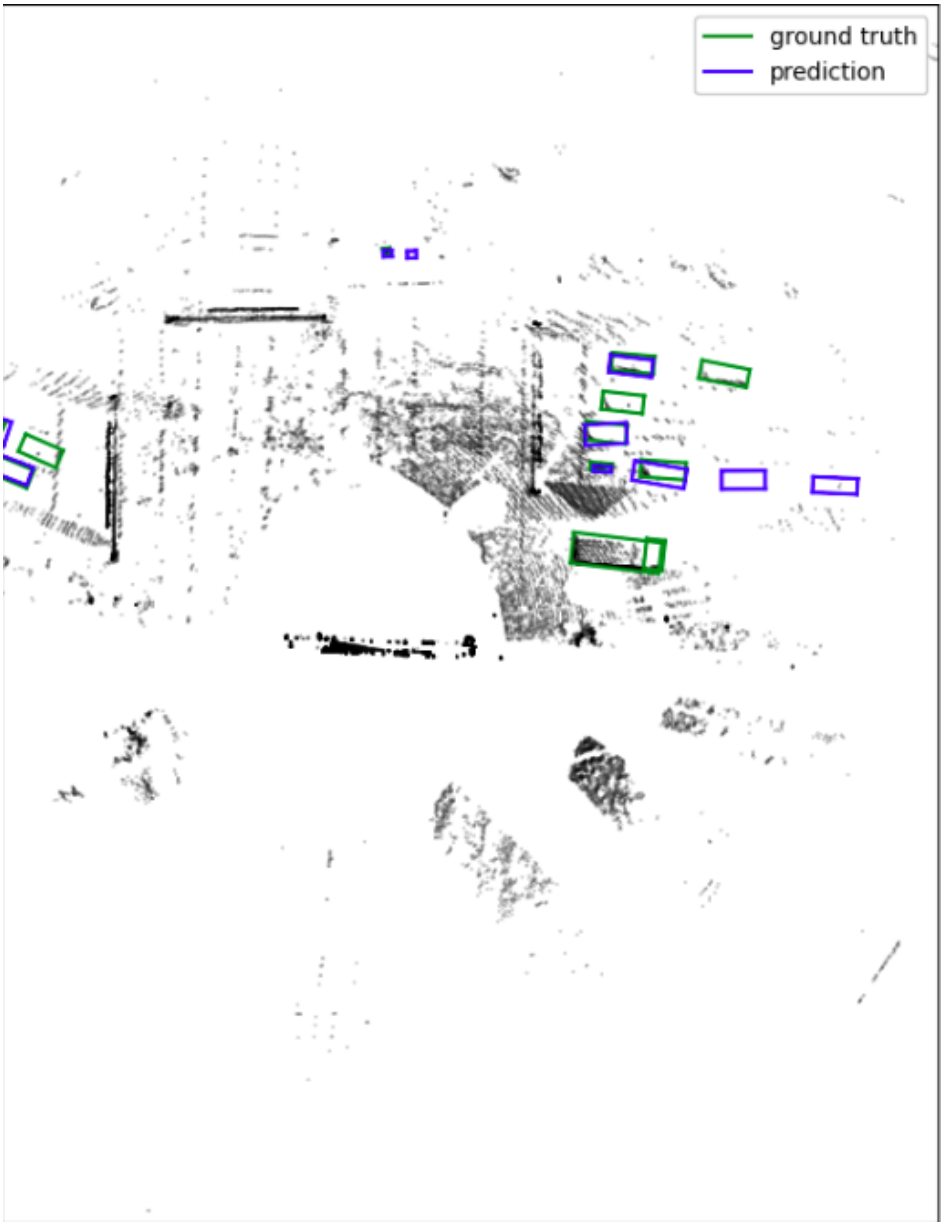


Figure 5.5: Example of missing ground truth label's BEV prediction

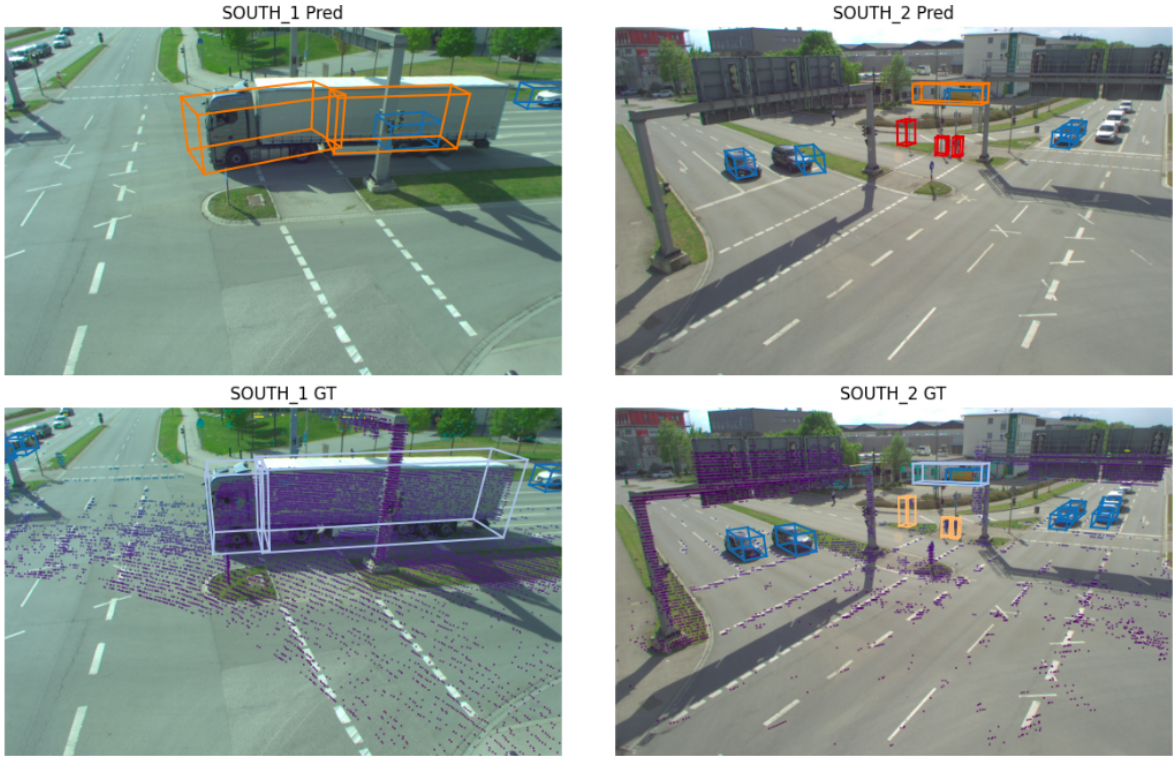


Figure 5.6: Example of truck vs trailer prediction

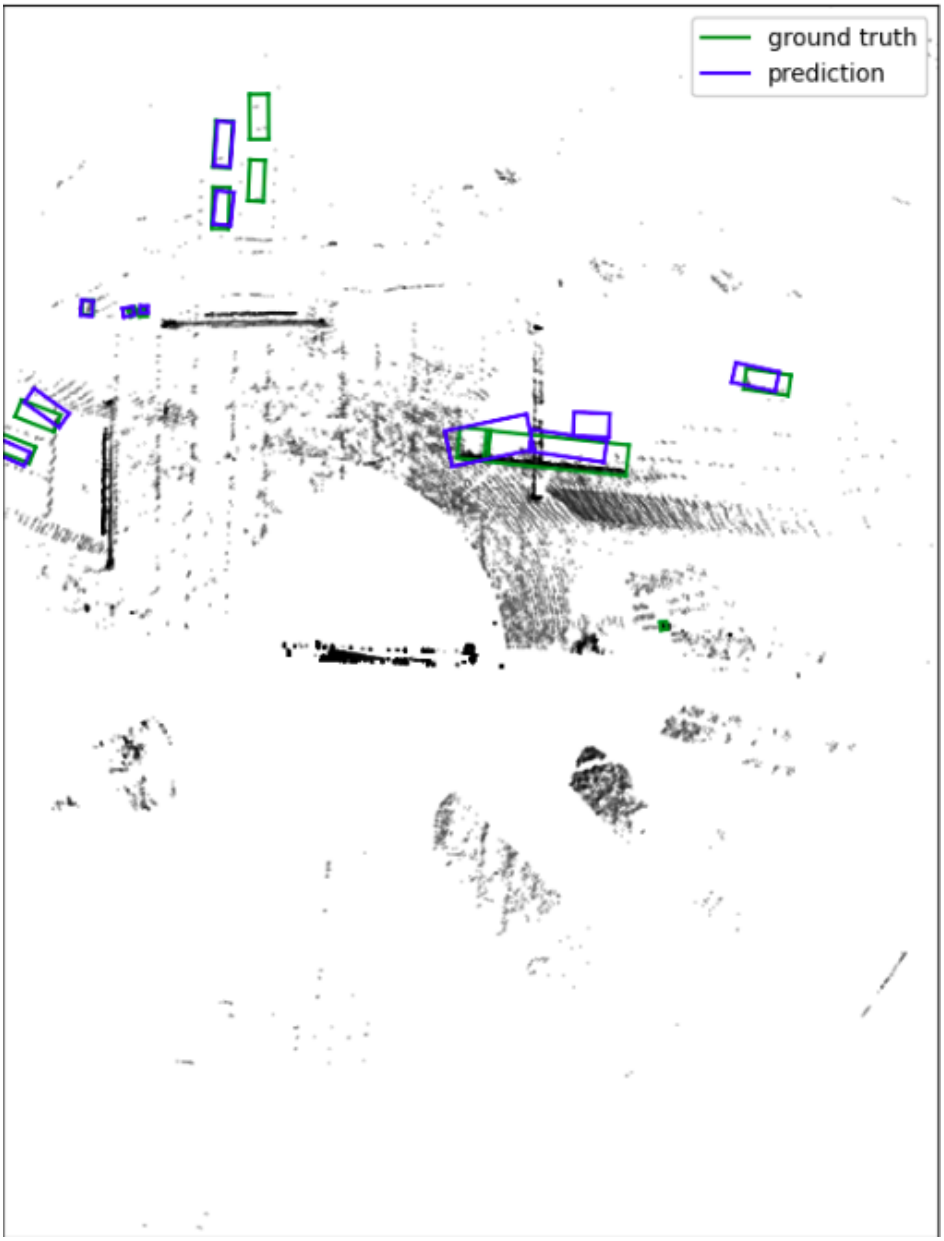


Figure 5.7: Example of truck vs trailer BEV prediction

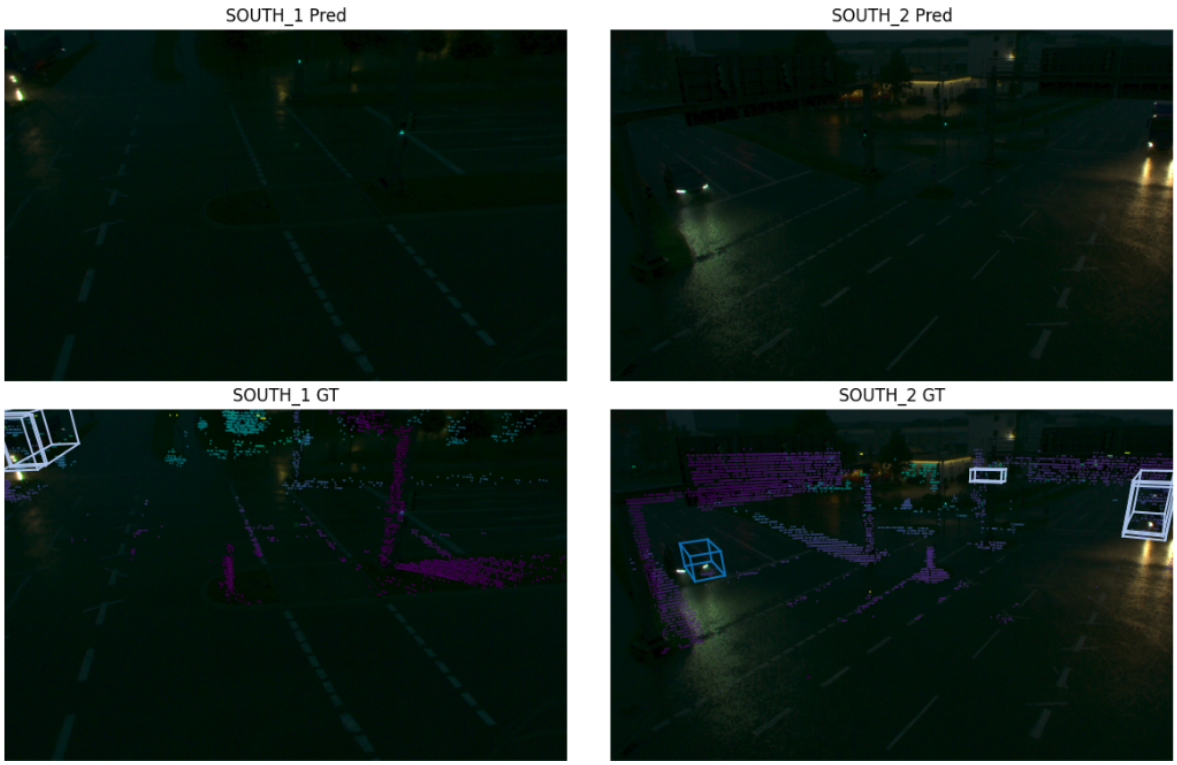


Figure 5.8: Example of night time

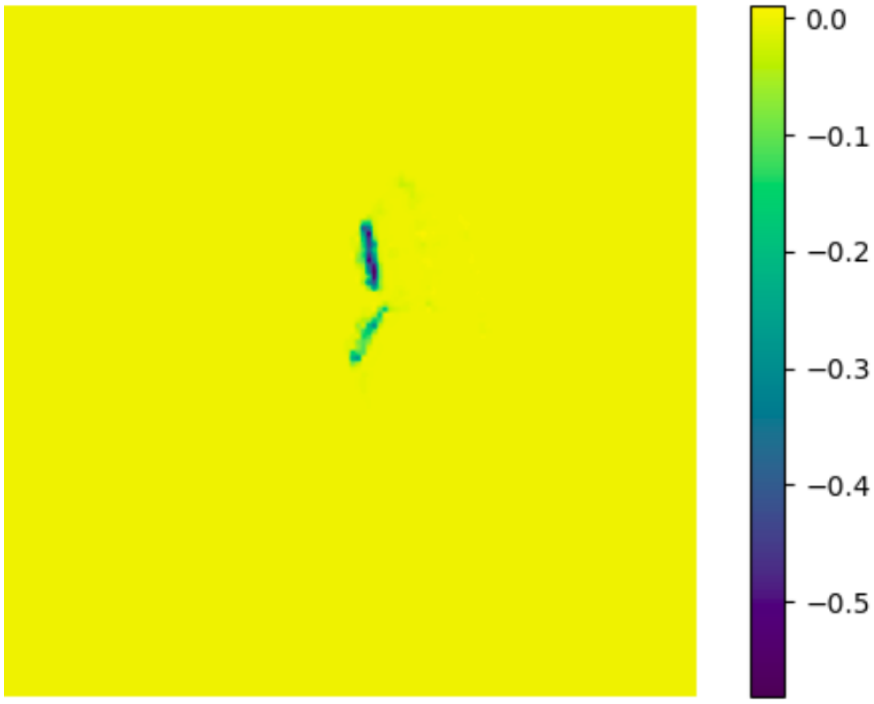
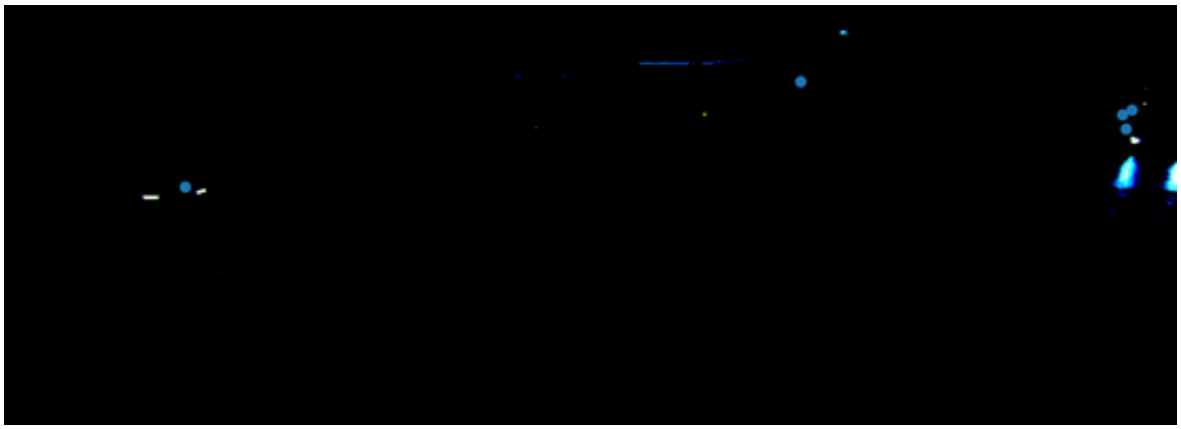


Figure 5.9: Example of night time BEV features

(a) South 1 night image



(b) South 2 night image

(c) Example South 1 daytime image

Figure 5.10: Example images fed to model by dataloader after normalizing. Blue dots added to see ground truth center of objects

Chapter 6

Conclusion

This thesis has successfully integrated state of the art Birds-Eye-View model and show it's potential in object detection in the Infrastructure View. With BEVDepth [Li+23], the real time run of model at **30+ FPS** while taking the benefit of point cloud without having to deal with its high processing time points to the benefit of using camera based object detection while incorporating benefits of LiDAR based methods.

The TUMTraf-I dataset [Zim+23] provides excellent ground for testing Infrastructure View object detection with multiple views of sample available along with point cloud making it perfect to test model like BEVDepth. NuScenes [Cae+20] is another dataset on which new state of the art models are tested on and most of the times it requires additional work to transfer dataset from one to another. This process was made more streamlined by using a dataset converter which was able to translate the TUMTraf-I dataset into nuScenes dataset format.

Ablation studies were conducted on BEVDepth to get an extend of performance it could give and for the case of high inference speed and low memory occupancy, mAP of **46.57%** was recorded for image image size 256×704. It was seen how increasing the resolution of image decreased the inference speed and increased the needed memory as does general consensus agrees. Peculiarity originated at the removal of weight regularization where basic machine learning teachings would predict the performance in object detection would deteriorate on test set but contrary to belief it actually improved. This points to the fact that model didn't generalize over the training dataset and this could be changed by increasing the dataset size.

Further improvements to the model could be suggested by using temporal information along with spatial as BEVDepth does. This would mean to include one or two last temporal samples with the reasoning that vehicles and pedestrian move within the confine of physical universe following laws of nature, so from one frame to another they would generally be in the vicinity of previously found location and thus using tracking algorithm the model can be made more robust.

Another direction this work can go along is the use of domain adaptation to solve the small dataset problem. The DAIR-V2X-I [Yu+22] dataset and Rope3D [Ye+22] dataset provide infrastructure view data similar to TUMTraf-I dataset and they could be leveraged to pre-train the model and apply modern domain adaptation to transfer the learning from their rich dataset to TUMTraf-I.

Finally, since the model is faster than 10 FPS, it could be deployed on the real system by exporting the model using TensorRT by NVIDIA. It has been shown that a mere 2GB of memory is sufficient to run the model on NVIDIA RTX 3090 GPU.

List of Figures

List of Tables

Bibliography

[Bas24] Basler. *Basler Camera*. 2024. URL: https://docs.baslerweb.com/aca1920-50gc (visited on 06/17/2024).

[BN06] Bishop, C. M. and Nasrabadi, N. M. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[Cae+20] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. "nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[CS20] Chair of Robotics, A. I. and Systems, R.-T. *Providentia++*. 2020. URL: https://innovation-mobility.com/ (visited on 06/10/2024).

[GBC16] Goodfellow, I. J., Bengio, Y., and Courville, A. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[Hag20] Hagedorn, N. *OpenLABEL Concept Paper*. 2020. URL: https://www.asam.net/project-detail/asam-openlabel-v100/ (visited on 06/13/2024).

[He+16] He, K., Zhang, X., Ren, S., and Sun, J. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[HF17] Henderson, P. and Ferrari, V. "End-to-end training of object class detectors for mean average precision". In: *Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part V 13*. Springer. 2017, pp. 198–213.

[HSS18] Hu, J., Shen, L., and Sun, G. "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.

[Inc19] Inc, M. *nuScenes Dataset Devkit*. 2019. URL: https://github.com/nutonomy/nuscenes-devkit (visited on 06/19/2024).

[Int21] International, S. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. 2021. URL: https://www.sae.org/standards/content/j3016_202104/ (visited on 07/13/2024).

[JZH21] Janiesch, C., Zschech, P., and Heinrich, K. "Machine learning and deep learning". In: *Electronic Markets* 31.3 (2021), pp. 685–695.

[Krä+19] Krämmer, A., Schöller, C., Gulati, D., Lakshminarasimhan, V., Kurz, F., Rosenbaum, D., Lenz, C., and Knoll, A. "Providentia–A Large-Scale Sensor System for the Assistance of Autonomous Vehicles and Its Evaluation". In: *arXiv preprint arXiv:1906.06789* (2019).

[Lan+19] Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. "Pointpillars: Fast encoders for object detection from point clouds". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 12697–12705.

[Li+23] Li, Y., Ge, Z., Yu, G., Yang, J., Wang, Z., Shi, Y., Sun, J., and Li, Z. "Bevdepth: Acquisition of reliable depth for multi-view 3d object detection". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 2. 2023, pp. 1477–1485.

[Li+22] Li, Z., Wang, W., Li, H., Xie, E., Sima, C., Lu, T., Qiao, Y., and Dai, J. "Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers". In: *European conference on computer vision*. Springer. 2022, pp. 1–18.

[Liu+22] Liu, Y., Wang, T., Zhang, X., and Sun, J. "Petr: Position embedding transformation for multi-view 3d object detection". In: *European Conference on Computer Vision*. Springer. 2022, pp. 531–548.

[LH17] Loshchilov, I. and Hutter, F. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[ON15] O'shea, K. and Nash, R. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[Ous24] Ouster. *Ouster LiDAR*. 2024. URL: https://ouster.com/products/hardware/os1-lidar-sensor (visited on 06/17/2024).

[Per11] Perumal, L. "Quaternion and Its Application in Rotation Using Sets of Regions". In: *International Journal of Engineering and Technology Innovation (IJETI)* 1 (Oct. 2011), pp. 35–52.

[PF20] Philion, J. and Fidler, S. "Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*. Springer. 2020, pp. 194–210.

[Pla19] Placek, M. *Global automotive V2X market size between 2018 and 2025*. 2019. URL: https://www.statista.com/statistics/565790/global-market-size-for-v2x-in-vehicles/ (visited on 07/14/2024).

[Rus+15] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115 (2015), pp. 211–252.

[Vas+17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[Wan+22] Wang, Y., Guizilini, V. C., Zhang, T., Wang, Y., Zhao, H., and Solomon, J. "Detr3d: 3d object detection from multi-view images via 3d-to-2d queries". In: *Conference on Robot Learning*. PMLR. 2022, pp. 180–191.

[War23] Wardana, G. A. "Deep Multi-Modal Cooperative 3D Object Detection of Traffic Participants Using Onboard and Roadside Sensors". MA thesis. Technical University of Munich, 2023.

[YML18] Yan, Y., Mao, Y., and Li, B. "Second: Sparsely embedded convolutional detection". In: *Sensors* 18.10 (2018), p. 3337.

[Yan+23] Yang, L., Yu, K., Tang, T., Li, J., Yuan, K., Wang, L., Zhang, X., and Chen, P. "Bevheight: A robust framework for vision-based roadside 3d object detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 21611–21620.

[Ye+22] Ye, X., Shu, M., Li, H., Shi, Y., Li, Y., Wang, G., Tan, X., and Ding, E. "Rope3d: The roadside perception dataset for autonomous driving and monocular 3d object detection task". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 21341–21350.

[YZK21] Yin, T., Zhou, X., and Krahenbuhl, P. "Center-based 3d object detection and tracking". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 11784–11793.

[Yu+22] Yu, H., Luo, Y., Shu, M., Huo, Y., Yang, Z., Shi, Y., Guo, Z., Li, H., Hu, X., Yuan, J., et al. "Dair-v2x: A large-scale dataset for vehicle-infrastructure cooperative 3d object detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 21361–21370.

[Zhu+20] Zhu, X., Su, W., Lu, L., Li, B., Wang, X., and Dai, J. "Deformable detr: Deformable transformers for end-to-end object detection". In: *arXiv preprint arXiv:2010.04159* (2020).

[Zim+23] Zimmer, W., Creß, C., Nguyen, H. T., and Knoll, A. C. "TUMTraf Intersection Dataset: All You Need for Urban 3D Camera-LiDAR Roadside Perception". In: *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2023, pp. 1030–1037.