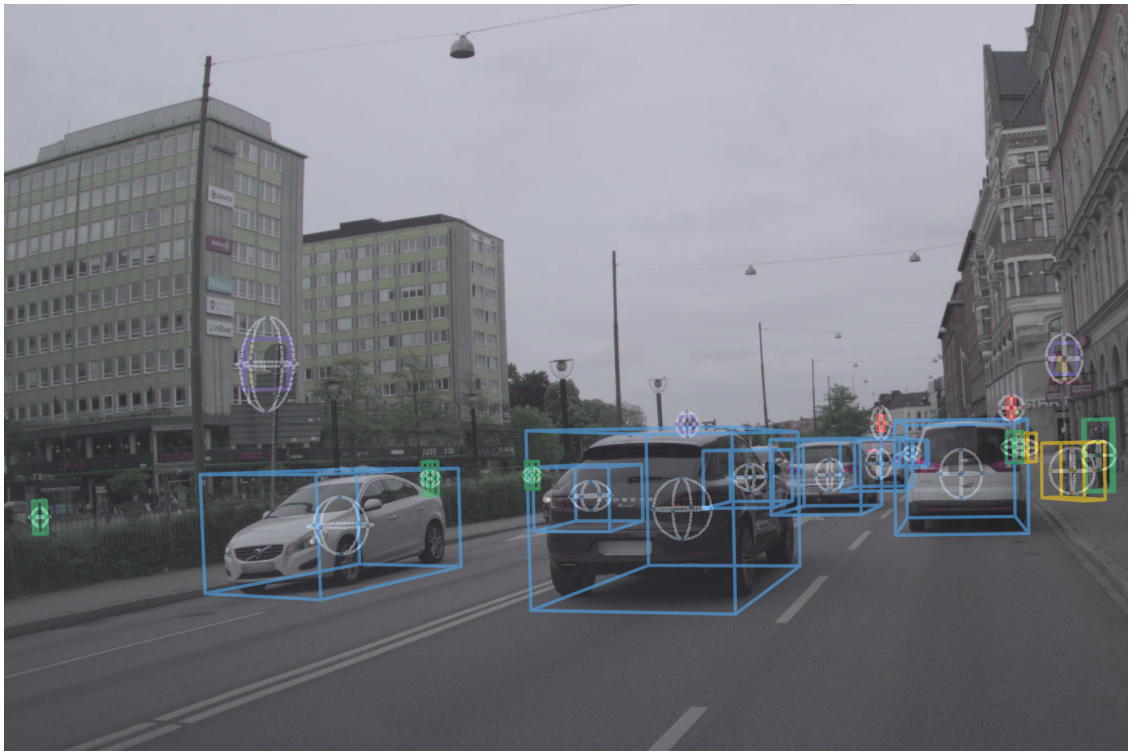
CHALMERS
UNIVERSITY OF TECHNOLOGY



Uncertainty estimation in multi-modal 3D object detection

Master's thesis in Data Science and AI

ANTON ROSÈN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

Master's thesis 2024

Uncertainty estimation in multi-modal 3D object detection

ANTON ROSÈN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Uncertainty estimation in multi-modal 3D object detection

ANTON ROSÈN

Supervisor: Lars Hammarstrand, Department of Electrical Engineering
Advisor: Joakim Johnander, Maryam Fatemi, Carl Lindström, Zenseact
Examiner: Lars Hammarstrand, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Output detections from a fusion model with visualized 95% confidence intervals of object center points on an example from ZOD[1][1].

Typeset in LaTeX
Gothenburg, Sweden 2024

[1]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details.

Uncertainty estimation in multi-modal 3D object detection

ANTON ROSÉN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Object detection is an important part of many autonomous driving systems, providing condensed information about the vehicle's surroundings. For good performance in different environmental conditions, multi-modal object detection is often used, where information from different sensors are fused. Due to factors such as sensor noise, occlusion, and adverse weather conditions, there is an inherent uncertainty in the object detection task. Most state-of-the-art approaches for multi-modal 3D object detection do not model these uncertainties for regression. Explicitly modeling and estimating the uncertainties leads to higher interpretability, allows analysis of difficult situations, and can improve the performance of downstream tasks.

In this work, we explore how uncertainty can be modeled and estimated in multi-modal 3D object detection. We show that directly modeling the uncertainties of bounding box parameters can provide meaningful uncertainty estimates without sacrificing neither predictive performance nor computational efficiency. We compare modeling the uncertainties both separately per detection, using normally distributed random vectors, and jointly per frame, using Poisson multi-Bernoulli random finite sets. Our results show that separate modeling enhances predictive performance, while joint modeling yields more accurate uncertainty estimates. Additionally, we demonstrate that these predicted uncertainties can identify unlabeled data where the model performs poorly, underscoring their importance for more interpretable and safe autonomous driving systems.

Keywords: 3D Object Detection, Sensor Fusion, Uncertainty Estimation.

Acknowledgements

I would like to thank Zenseact for allowing me the opportunity and resources to do this project. Special thanks go to my industrial supervisors: Joakim Johnander, Maryam Fatemi and Carl Lindström, who patiently answered all my questions and allowed me to bounce ideas. I'd also like to thank my supervisor and examiner Lars Hammarstrand, who provided the necessary guidance to complete the project. A special thank you is given to my friends and family who have supported me during the entirety of my education. A final thank you is owed to the Chalmers Formula Student team, always providing a sense of community and serving as a necessary distraction when bogged down in thesis work.

Anton Rosén, Gothenburg, 2024-06-27

Contents

List of Figures

List of Tables

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AP	**A**verage **P**recision
BEV	**B**ird's-**E**ye **V**iew
BNN	**B**ayesian **N**eural **N**etwork
CE	**C**alibration **E**rror
DoF	**D**egrees **o**f **F**reedom
ES	**E**nergy **S**core
LiDAR	**Li**ght **D**etection **A**nd **R**anging
LSS	**L**ift, **S**plat, **S**hoot
mAP	**m**ean **A**verage **P**recision
NDS	**N**uScenes **D**etection **S**core
NLL	**N**egative **L**og **L**ikelihood
OOD	**O**ut **O**f **D**istribution
PDF	**P**robability **D**ensity **F**unction
PMB	**P**oisson **M**ulti-**B**ernoulli
PMB-NLL	**P**oisson **M**ulti-**B**ernoulli **N**egative **L**og **L**ikelihood
PPP	**P**oisson **P**oint **P**rocess
RFS	**R**andom **F**inite **S**et
ZOD	**Z**enseact **O**pen **D**ataset

1

Introduction

Vehicles with autonomous capabilities have the opportunity to improve the transportation sector in several ways. Autonomous vehicles can reduce the number of traffic-related accidents, as the majority of accidents are a result of human error [2]. They can also revolutionize shared transportation. With driverless vehicles, the scope of public transport and other mobility services can be wider and offered at a lower cost, thereby reducing overall road congestion, decreasing parking demand, lowering energy usage, and cutting greenhouse emissions [3]–[6].

For vehicles to operate autonomously, several different systems work together. One of the key systems required is the perception system, which is responsible for identifying and analyzing the car's surrounding environment. Some typical perception tasks include detecting and tracking objects such as cars and pedestrians, classifying traffic signs, and finding lane markers. To perceive the world, autonomous vehicles utilize different sensor types, usually referred to as modalities. Some commonly used modalities are camera, LiDAR, and radar, all with respective advantages and disadvantages. Cameras provide high-resolution and semantically dense data. However, they typically perform poorly in low-light situations and do not provide depth information. LiDAR and radar both provide depth information and perform well under low-light situations, but the data is at a much lower resolution and does not contain color information. By combining these sensors effectively, it is possible to get the benefits of the each respective sensor type, enabling systems that perform well in a wider range of situations.

To effectively act on the information provided by perception systems, it is meaningful to consider not only the output of a model but also the certainty of that information. No matter how good a model is, uncertainty will always be present in any perception system. Examples of unavoidable sources of uncertainty include adverse weather conditions limiting visibility, the presence of new unseen objects in the scene, occluded objects, and inherent sensor noise. Knowing how much a detection can be trusted can provide vital information for the real-time decision-making systems of autonomous vehicles.

Perception uncertainties can be used to improve the safety of autonomous vehicles, by for instance propagating them through downstream tasks [7]. Many downstream tasks, such as target tracking [8], trajectory planning [9], and simultaneous localization and mapping [10], assume some measurement noise on the perception outputs. By estimating and outputting uncertainty in the perception model, a more repre-

sentative measurement noise can be used, potentially increasing the performance on the respective tasks. Separately, precautions such as slowing down, handing over control, or coming to a safe stop can be implemented in the presence of perception uncertainty.

Uncertainties can also enhance the interpretability of autonomous vehicles by providing deeper insights into their decision-making processes and predicted actions. Interpretability of autonomous driving systems are important on several levels. For passengers, model interpretability can increase the trust that the vehicle will drive them to their destination safely. Increased interpretability can also help diagnose issues with models, help validation against safety standards, and provide better accountability if something goes wrong [7].

Additionally, the predicted uncertainties could be used for active learning [11], by finding unlabeled instances where the model is uncertain. These instances can then be annotated and used to improve the model. By efficiently finding informative data points, both the annotation cost and training time can be decreased, as less overall data is required. It is thus desirable for a model to provide well-calibrated uncertainty estimates with its predictions.

Autonomous driving perception can be divided into different tasks. In this thesis, we focus on 3D object detection, where the task is to detect and classify objects, as well as regress their 3D pose. In the object detection literature, the vast majority of papers do not consider the uncertainty of predictions [12]. Of the papers that do, most papers only consider 2D object detection on images, e.g., [12]–[15]. A number of works consider 3D object detection with one modality, e.g., [16]–[18]. Very few works study uncertainty in 3D object detection with multiple modalities, however, one example is the work of Feng et al. [19].

In this thesis we investigate the estimation of uncertainties in camera-LiDAR 3D object detectors. Instead of only focusing on predictive performance, our main focus is on the uncertainty estimates: how they can be obtained, how to evaluate them, and how they can be used to create safer, more interpretable autonomous vehicles.

1.1 Thesis Objectives

The objective of this thesis is to identify and evaluate methods to estimate the uncertainty in 3D object detectors. The goal is to develop a model that provides useful uncertainty estimates that can be used in, for instance, downstream tasks or for active learning by highlighting cases where the model is uncertain. Modeling the uncertainties should ideally not sacrifice predictive performance or come at a significant computational cost, as that would limit the practical application of the model.

1.2 Limitations

As the focus of the thesis is to investigate uncertainty modeling, we will not design an object detector from scratch and instead use and modify the state-of-the-art model BEVFusion [20] to perform our experiments.

There are also a large amount of sensor modalities that could be used, for instance, camera, LiDAR, radar, and ultrasonic sensors. To keep the scope manageable, we will only consider the fusion between the LiDAR and camera modalities. This also enables us to use the Zenseact Open Dataset [1], which does not feature radar or ultrasonic data.

Additionally, there are a wide variety of ways to model and estimate uncertainty. Due to time limitations, we will not cover all the possible ways of estimating uncertainty. Most notably, we chose not to investigate Bayesian Neural Networks due to their poor predictive performance on 3D object detection [21]. We will also not investigate sampling based approaches such as Monte Carlo Dropout or Deep Ensembles, as they are computationally inefficient, requiring multiple forward-passes at test time [12].

1.3 Contributions and Main Results

The main results of this this thesis can be summarized in the following list:

- Direct modeling can effectively be applied to a general multi-modal 3D object detector to produce accurate uncertainty estimates without severely influencing inference time.
- Modeling the object detection problem as a set prediction problem, optimizing jointly over all predictions, produces the most accurate uncertainty estimates.
- The produced uncertainty estimates can be used to identify when the model performs poorly, help find out-of-distribution unlabeled cases for active learning, and lead to overall more interpretable object detectors.

2

Background

To model the uncertainty of object detectors, a few key prerequisites are described. Firstly, a general introduction to 3D object detection is given. Here, the problem is defined, common methods are illustrated, and the evaluation metrics are outlined. Secondly, a primer is given on random vectors and sets to be able to model the uncertainty. Finally, how the uncertainties are modeled and estimated is shown.

2.1 Object Detection

There are many ways to capture the information present in images or point clouds, all with respective advantages and disadvantages. By performing classification on an entire image or point cloud, one can obtain a high-level representation of the scene. Classifying every single pixel or point separately, as in segmentation tasks, a very detailed semantic understanding can be obtained. Object detection is instead a set prediction task, where the goal is to predict the set of all object in a scene. Normally, the set prediction problem is divided into one regression problem and one classification problem, where the goal is to regress – typically box-shaped – boundaries around all objects, and classify the object contained within each boundary. This creates an efficient representation of the environment and is typically used in autonomous driving applications to, for instance, detect vehicles, pedestrians, and obstacles.

One important distinction is between 2D and 3D object detection. In 2D object detection, the aim to find the boundary parameters of objects in the 2D image plane. The resulting object boundaries are thus in pixel coordinates (u, v), giving no notion of absolute scale, rotation, or position. In 3D object detection, the goal is instead to find the 3D locations of objects in a real-world reference frame. The resulting boundaries are thus in world coordinates (x, y, z), yielding a more useful representation for the task of autonomous driving.

The boundaries used in object detection are typically represented using boxes, as they serve as good approximations and are computationally efficient. These boxes are referred to as bounding boxes, and are defined as boxes that contain the objects. To gain the maximum information, it is desirable for the the bounding box to be as tight as possible to the object. More rigorously, the goal is to find the minimum bounding box of an object, i.e., the box of minimum volume that the object is fully

within.

For 2D object detectors, these bounding boxes are typically axis-aligned, meaning that they have no rotation with respect to the Cartesian coordinate axes. This yields a four degrees of freedom (DoF) parametrization of the bounding box, made up of either the coordinates of opposing corners or the coordinates of the center point together with the box size. When moving to 3D, rotation has a larger effect on the tightness of bounding boxes. In 3D there are 3 DoF for rotation. For autonomous driving perception, often only the rotation in the ground plane, yaw, is modeled. Figure 2.1 shows the difference between a 2D axis-aligned bounding box and a 3D bounding box with yaw rotation.

Figure 2.1: Difference between 2D (left) and 3D (right) bounding box representations on an example image from ZOD[1][1]. The 2D bounding box can not capture the depth and does not represent the location of the object with respect to the ego-vehicle.

Early object detectors used traditional machine vision algorithms, and relied on detecting hand-crafted features to find vehicles in images. Over the past decade, deep learning-based object detectors have become more and more capable, offering great detection performance and the capability of working in real-time on embedded hardware. Instead of relying on hand-crafted features, deep learning-based object detectors instead learn to identify what makes up objects by training on a large set of examples.

Generally, deep learning-based 3D object detectors first extract and encode features from the input. This process differs depending on the modality used. For camera-based detectors, features are often processed first in the 2D image plane, before the depth of features are estimated further processed in 3D. For LiDAR-based detectors, an initial transformation is often done to the unstructured, unordered point cloud, before 3D feature encoding is done. Finally, common for any modality, is the use of a detection head to perform bounding box regression and classification.

The detection head requires proposal bounding boxes to perform the regression and classification. There are two types of methods to generate proposals: anchor-free methods and methods based on anchor boxes. Anchor boxes are pre-determined bounding boxes of different sizes and at different locations. Generally, a large number of anchor boxes are required to cover the 3D space with sufficient density [22]. This can also create a large number of overlapping predictions. To deal with these

[1]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details

overlapping predictions post-processing steps like non-maximum suppression [23] can be utilized, which remove overlapping detections by thresholding on the amount of overlap and the classification confidence. Anchor-free methods instead generate proposals based on the features, often requiring significantly fewer proposals. An example of an anchor-free detector is CenterPoint [22], which is described in Section 2.1.2.

2.1.1 3D Object Detection Metrics

One of the most common performance metrics used for 3D object detectors is average precision (AP), which considers both classification and regression. AP is computed in two steps. First, the predictions from a 3D object detector are matched to the corresponding ground truth annotations. All predictions are sorted by confidence, and – starting with the most confident prediction – compared against all the annotations using some criterion. The criterion is typically the Euclidean distance between bounding boxes centers. If a prediction is within a specified distance threshold, the prediction is marked as a true positive and the corresponding annotation is not considered for the matching of future predictions.

After the matches have been computed, the second step is to, for each class, compute the average precision for different levels of recall. Using the matches, the detections are divided into false positives (FP), false negatives (FN), true positives (TP). Using these, the recall is given by

$$\text{Recall} = \frac{TP}{TP + FN} \ , \tag{2.1}$$

and the precision by

$$\text{Precision} = \frac{TP}{TP + FP} \ . \tag{2.2}$$

The precision and recall values can be used to construct the precision-recall curve, where precision is plotted against recall. The AP is then computed by integrating the precision-recall curve. This is done for each class separately and the AP is the mean over the classes. In the matching step, chosen threshold is a bit arbitrary. To counteract this, the mean average precision (mAP) can be used, where the mean of AP is taken for different thresholds. Typically, the thresholds are set at 0.5m, 1m, 2m, and 4m for the Euclidean distances between the centers of bounding boxes.

This formulation of mAP does not consider how well all the bounding box parameters of each match the ground truth, only considering the center location. To resolve this, some metrics extend mAP to additionally capture this information. One notable example of this is the nuScenes Detection Score (NDS) [24]. In NDS, the average translation error (ATE), scale error (ASE), and orientation error (AOE) are computed for true positive detections. In nuScenes, bounding boxes are additionally parameterized with velocity and class specific attributes, e.g. sitting and standing for pedestrians. These are captured with average velocity error (AVE) and attribute error (AAE). For exact definitions of each quantity, see [24]. The NDS is finally defined as the average of all these metrics, with mAP recieving the highest

weight,

$$\text{NDS} = \frac{1}{|\text{AE}| + 5}\left(5\,\text{mAP} + \sum_{\text{AE}}(1 - \min(1, \text{AE}))\right) \quad . \tag{2.3}$$

Here AE ∈ {ATE,ASE,AOE,AVE,AAE} are the additional metrics. If some quantities are not estimated, they can be omitted from AE. In this thesis we do not estimate velocity or class specific attributes and instead use AE ∈ {ATE,ASE,AOE}.

2.1.2 LiDAR-based 3D Object Detection

Due to the explicit 3D representation of LiDAR data, LiDAR-based 3D object detectors have become very capable. LiDAR stands for light detection and ranging and is an active sensor technology that transmits and receives waves of light. This makes LiDAR capable of performing well in low-light scenarios where passive technologies – like cameras that rely on the light in the environment – struggle. While LiDAR sensors are less common than cameras, due to recent improvements in resolution, cost, and volume, some car manufacturers have started integrating LiDAR sensors into their vehicles [25]–[27].

The sparse, unordered, point-based LiDAR data is not trivial to process. For LiDAR-based 3D object detectors, three types of approaches are common to transform and process the input: projection-based approaches, point-nets, and voxel-based approaches. The differences between these are illustrated in Figure 2.2.

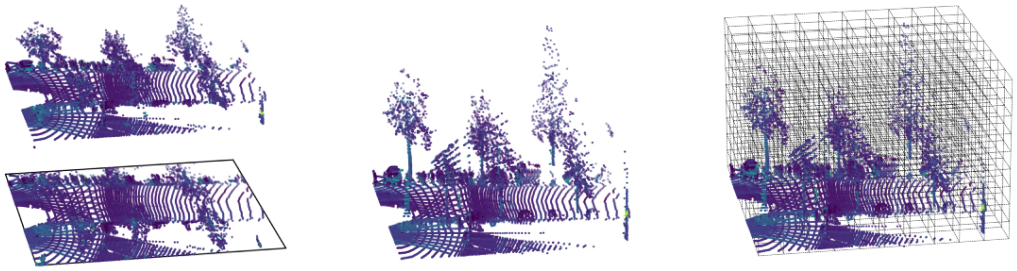


Figure 2.2: Different point cloud representations in 3D Object detection. Projection-based approaches (left) project the 3D points into a 2D image plane. Point-nets (middle) operate on the regular unordered set of points. Voxel-based approaches (right) divide the point cloud into structured voxels. Example point cloud from ZOD[2][1].

Projection-based approaches utilize the large amount of research done on 2D object detectors and project the point cloud into 2D. Projection-based approaches work in three steps. First, they generate a 2D image – either from the point of view of the LiDAR, or from a bird's-eye view (BEV) perspective – from the point cloud. Secondly, they perform 2D object detection on that image. Thirdly, they regress 3D bounding boxes using the depth information provided by the point cloud. This approach was used by for instance [28] and [29] to set a new state-of-the-art. Since

[2]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details

then, more powerful approaches have been proposed where the 3D representation is used.

Point-nets, instead utilize the sparse, unordered, point-based representation of point clouds by directly taking them as input. This method was popularised and named by the paper PointNet [30], where the layers operate on the individual points separately before transforming and aggregating them into a global feature vector. While this reduces the need for special transformations, and is invariant to the order of points, its computational complexity grows with the number of points in the point cloud and is sensitive to input perturbations.

The final type of LiDAR-based 3D object detectors are voxel-based representations. These methods partition space into voxels, which are the 3D generalization of pixels. VoxelNet [31], is one prominent voxel-based method. In VoxelNet, following voxelization, point-nets are applied to the points within each voxel to encode the shape information into a vector format. The spatial context is then aggregated across voxels using 3D convolutions, creating a feature map that is later fed into a detection head. However, 3D convolutions are computationally expensive, leading to different proposed improvements. SECOND [32], utilizes sparse convolutions to speed up the inference, and PointPillars [33], create a 2D grid of voxels with infinite height, enabling the use of the more efficient 2D convolutions.

Most of the previous methods used anchor-based detection heads. These either use axis-aligned anchor priors, that struggle to accurately capture the orientation of 3D objects, or use different anchors for each orientation, greatly increasing the computational burden. Therefore, anchor-free detection heads that do not rely on priors have become increasingly prominent. CenterPoint [22], is one such method. CenterPoint takes a feature map from any 3D backbone, such as the ones used in VoxelNet or PointPillars, and predicts a heatmap $\hat{Y} \in [0,1]^{W \times H \times C}$, where W, H are the number of width and height pixels in the heatmap, and C is the number of classes. A prediction of $\hat{Y}_{xyc} = 1$ corresponds to a detected object of class c at (x, y) and $\hat{Y}_{xyc} = 0$ corresponds to background, making the peaks of the heatmap the location of objects. An example heatmap from CenterPoint is shown in Figure 2.3. These heatmaps are learnt by projecting the annotated bounding boxes into a heatmap Y and performing pixelwise logistic regression with focal loss [34],

$$L_k = \frac{-1}{N} \sum_{xyc} \begin{cases} \left(1 - \hat{Y}_{xyc}\right)^{\alpha} \log\left(\hat{Y}_{xyc}\right) & \text{if } Y_{xyc} = 1 \\ \left(1 - Y_{xyc}\right)^{\beta} \left(\hat{Y}_{xyc}\right)^{\alpha} \log\left(1 - \hat{Y}_{xyc}\right) & \text{otherwise} \end{cases} . \tag{2.4}$$

Here, α and β are hyper-parameters of the focal loss and N are the number of ground truths. The top N peaks of the predicted heatmaps are then used as object proposals, with a class label assigned depending on which heatmap the object corresponds to. The precise bounding box parameters are parameterized as sub-voxel refinements, $(\delta x, \delta y)$; height over ground, z; size, (l, w, h); and $(\sin(\theta), \cos(\theta))$ for a wrap-around resilient representation of the yaw angle. These parameters are later regressed for each proposal using a feed-forward network. A confidence score for each prediction is also obtained, with higher heatmap peaks corresponding to higher confidences. This confidence score indicates the likelihood of an object existing and is useful

for ranking and thresholding detections; however, it is not a calibrated existence probability.

TransFusion [35], extends the CenterPoint head by using a transformer-based approach. In TransFusion, the top candidates from the heatmap are used as object queries, which are cross-attended with the feature maps to aggregate relevant context, and self-attended to reason about the relation between candidates. The queries are then independently decoded into boxes and class labels by a feed-forward network.

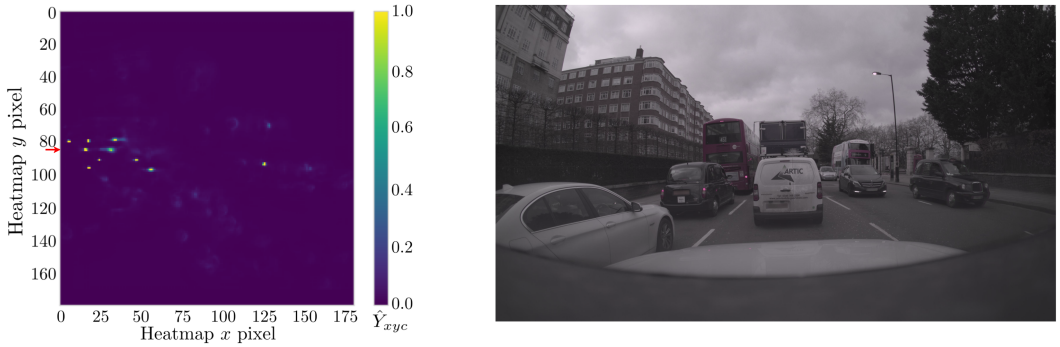


Figure 2.3: Example of a predicted heatmap on a frame from ZOD[3][1]. The peaks in the heatmap (left) represent the predicted object locations. In this example, all major peaks align with one of four horizontal lines, corresponding to the vehicles in the four lanes (right). The ego-vehicle is located at $(0, 85)$ in the heatmap, highlighted with a red arrow on the y-axis.

2.1.3 Camera-based 3D Object Detection

Camera-based 3D object detectors aim to localize and classify objects in 3D by only utilizing cameras. Compared to LiDAR-based 3D object detectors, vision-based detectors generally perform far worse on common benchmarks [21]. Despite this, using cameras for 3D object detection in autonomous vehicles is well studied [36]. Cameras, resembling human eyes in their data capture approach, are favored for their affordability, compact size, and ability to provide rich visual information. This makes them a preferred choice for 3D object detection tasks. The major problem with using cameras, is their lack of direct depth information. For each pixel in an image, there are an infinite amount of 3D points along its projective line that could correspond to it, making accurate 3D localization difficult.

Many ways have been proposed for estimating depth using cameras and the methods are divided into two categories: those based on monocular cameras and those based on stereo cameras. Utilizing stereo cameras, with known intrinsic and extrinsic calibrations, it is possible to estimate the depth by, for instance, matching features between the cameras. With features matched between the cameras, we can find the intersection of the projective lines, solving the depth ambiguity [37]. Stereo

[3]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details.

cameras, however, requires two, well calibrated, time synchronized, and spatially spread cameras for each view that you want to capture. This increases the cost, leads to higher data volumes and makes hardware integration into vehicles challenging. In contrast, while integration with monocular cameras is simpler, the geometry cannot be leveraged in the same way, requiring different methods.

Monocular depth estimation is the process of estimating depth from a single image. Here, instead of using pure geometry, the context of the image has to be utilized. One method to perform monocular depth estimation is to utilize ground truth depth information from a LiDAR and supervise a network to predict this depth [38]. Instead of relying on LiDAR, [39] uses data from stereo cameras and enforces consistency in the disparity between the left and the right images, producing accurate depth estimation without a ground truth depth. By assigning a depth to each pixel, a pseudo-LiDAR point cloud is obtained. These pseudo-LiDAR point clouds have been used in models like [40] and [41] to accurately regress 3D bounding boxes from 2D images.

A similar approach, popularised by Lift-Splat-Shoot (LSS) [42], is to implicitly unproject, or lift, pixels into 3D, generating a representation at multiple different depths for each pixel. For an image $\mathbf{X} \in \mathbb{R}^{3,H,W}$, we associate $|D|$ points $\{(u, v, d) \in \mathbb{R}^3 | \, d \in D\}$ to each pixel, where D is a set of discrete depths, and (u, v) are the image coordinates. From (u, v, d), the 3D coordinates (x, y, z) can then be obtained using the intrinsic and extrinsic matrices of the camera, together with a representative camera model. This creates a dense point cloud of size $D \cdot H \cdot W$, and for each pixel, the network predicts a distribution over the depth together with a context vector $\mathbf{c}$. The feature $\mathbf{c}_d$ associated with a point at a given depth is then defined as the context vector scaled by the depth probability α_d. This means that the network can choose between placing the context at a single point, if the depth is certain, and spreading the context across the ray, if the depth is uncertain. Figure 2.4 shows an example of lifting a 2D image into 3D using a pinhole camera model. In [43], this was applied to perform monocular 3D object detection by using a swin-transformer [44] image backbone, lifting into 3D, pooling the point cloud to BEV and using a CenterPoint [22] head.

2.1.4 Fusion-based 3D Object Detection

When it comes to fusing different sensor modalities, it is important to consider when the fusing is done. Many early methods for sensor fusion relied on late fusion, where separate networks were used for each modality to produce detections that were later merged together. This is a quite interpretable representation, as the individual outputs of the different sensors can be analysed. However, by fusing at the object level, some information that the sensors could use to help each other in the detection process is lost. For instance, if one sensor sees the front of an object and another sees the rear, a system using object-level fusion would likely be unable to capture the full object.

Many fusion-based object detectors instead choose to fuse the different modalities earlier, by e.g., fusing features between modalities or decorating the input of one

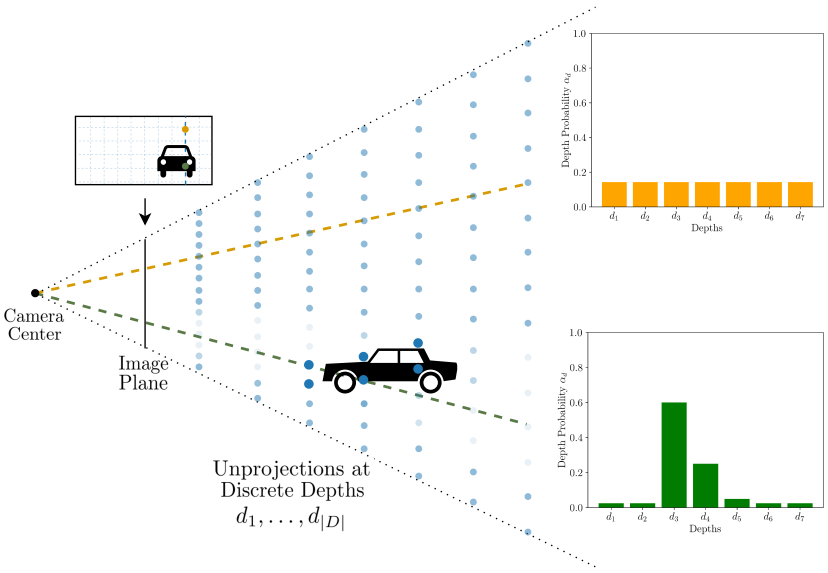


Figure 2.4: Example of the lifting step in Lift-Splat-Shoot [42]. The features are unprojected into 3D at some discrete depths, and for each depth the network predicts a depth probability α_d that together with the context vector encodes the feature in 3D. The depth probabilities allow the network to choose between assigning the feature to a few select points, if the network is certain of the depth (green line), or to spread it out, if the network is uncertain (orange line). Figure shows the unprojection of a vertical line in the image, with features from one channel, using the pinhole camera model. The opacity of points correspond to their probabilities.

modality with the other. Early multi-modal deep learning models opted for the second approach, either projecting a point cloud into a camera, e.g., [45], or projecting a camera into a point cloud, e.g., [46]. Both of these methods provided better results than their single-modality counterparts, however, there are some downsides. By projecting the camera information into a point cloud, the semantic density of the camera is lost, as a very small amount of pixels get a corresponding LiDAR point. And by projecting the LiDAR information into a camera, the geometric structure is lost, as neighbors in 2D space could be far away in 3D space.

Instead, most current state-of-the-art fusion-based 3D object detectors use separate backbones for each modality and then represent the features in a shared BEV representation. For LiDAR, this transformation can be done by flattening along the z direction, whereas for camera, a view transformation, for instance as in [42], can be used to lift the features into 3D before pooling into BEV. Among the first 3D object detectors to efficiently perform this was BEVFusion [20], which has been the foundation for many state-of-the-art 3D object detectors since, e.g., [47] [48].

The original BEVFusion architecture is made to work with a wide variety of backbones, necks and heads. An overview of the architecture can be seen in Figure 2.5. For the baseline implementation, the authors used the voxel-based backbone SECOND [32] to encode the LiDAR features, a swin-transformer [44] to encode the

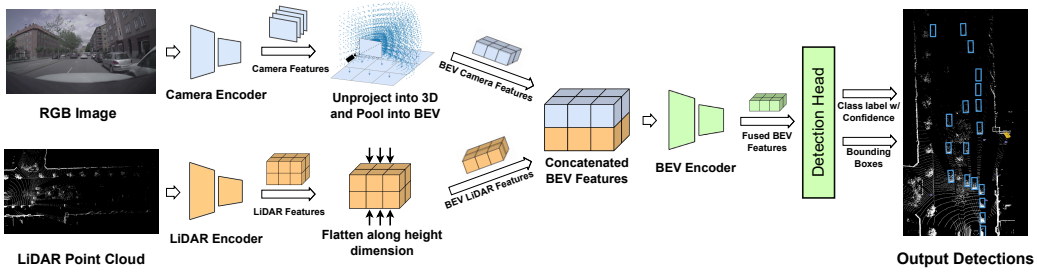


Figure 2.5: Overview of the BEVFusion architecture [20] using example inputs from ZOD[4][1]. For each modality, features are extracted and transformed into BEV. The BEV features are then concatenated and further processed before the detection head predicts the bounding box parameters and class labels with confidences.

camera features, a LSS-based view transformer [42] and a detection head based on TransFusion [35]. The main contribution of the BEVFusion paper was to make the pooling into BEV after the view-transformation more efficient, by implementing a specialized CUDA kernel. This allows the use of more accurate, computationally expensive methods, whilst maintaining real-time performance.

2.1.5 Datasets

Over the years, a huge amount of data has been collected and annotated for autonomous driving applications. While much of this data is proprietary, there are also several publicly accessible datasets. Notable examples include KITTI [49], nuScenes [24] and the Waymo Open Dataset [50]. These datasets feature diverse sensor configurations, annotations covering varying distances, and are captured across different geographical locations.

In this thesis, we leverage the recently released Zenseact Open Dataset (ZOD) [1]. ZOD contains 100,000 curated frames, where each frame consists of data captured by a high-resolution camera equipped with a wide-angle fish-eye lens, alongside inputs from three LiDAR sensors and other complementary sensors. The sensor setup and coordinate systems are illustrated in Figure 2.6, and a detailed description of relevant sensors is available in Table A.1. Each frame is annotated with objects visible in the image data, and contains 3D bounding boxes if the object is also present in the point cloud. The object classes in ZOD are divided into static – non-moving – objects, and dynamic – potentially moving – objects. The classes are split into general top-level classes, such as `Vehicle`, and further divided into sub-classes, such as `Car` and `Bus`. Additionally, each frame also contains metadata with sensor calibrations and additional information such as weather, time of day, and the occlusion ratio of an object. This extensive metadata enables, for instance, analyzing how the performance of a model changes in different environments and conditions.

[4]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details.

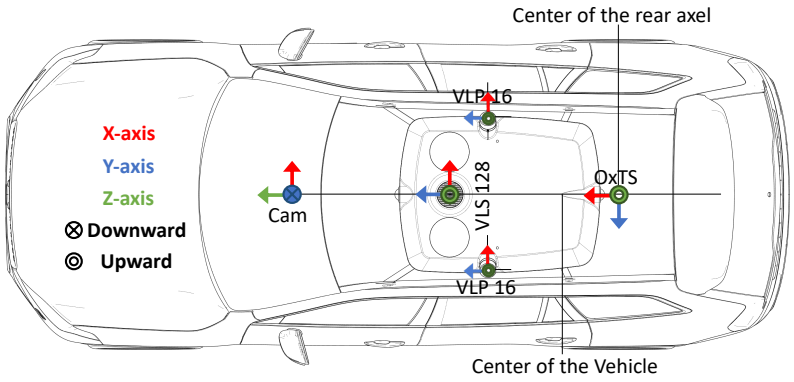


Figure 2.6: Sensor setup used in ZOD with corresponding coordinate systems [1]. VLP16 and VLS128 are the LiDARs, and OxTS is the GNSS/IMU defining the ego-vehicle frame. Figure reproduced with permission.

The camera used in ZOD is a wide-angle camera, meaning that there is a non-negligible optical distortion and the standard pinhole projection model is not applicable. Instead the cameras are modeled using the Kannala-Brant projection model [51], and the dataset includes fitted distortion and undistortion parameters for each frame. In the Kannala-Brant projection model, a 3D point, $(x, y, z) \in \mathbb{R}^3$, is projected into 2D, $(u, v) \in \mathbb{R}^2$, using

$$\begin{aligned} \gamma &= \sqrt{x^2 + y^2} \ , \\ \theta &= \text{atan2}(\gamma, z) \ , \\ \theta_d &= \theta + c_1\theta^3 + c_2\theta^5 + c_3\theta^7 + c_4\theta^9 \ , \\ u &= f_x \frac{\theta_d x}{\gamma} + c_x \ , \\ v &= f_y \frac{\theta_d y}{\gamma} + c_y \ , \end{aligned} \tag{2.5}$$

where c_1, c_2, c_3, c_4 are distortion coefficients, f_x, f_y are the focal lengths, (c_x, c_y) is the coordinate of the principal point, and $\text{atan2}(y, x)$ is the argument of the complex number $x + iy$. Note that the 3D point that is projected is with respect to the camera, meaning that additional transformations might be needed to, for instance, project 3D LiDAR points into the camera. Similarly, unprojecting a point from 2D to 3D is done using

$$\begin{aligned} \rho &= \sqrt{\left(\frac{u - c_x}{f_x}\right)^2 + \left(\frac{u - c_x}{f_x}\right)^2} \ , \\ \phi &= \text{atan2}(y, x) \ , \\ \theta &= \rho + k_1\rho^3 + k_2\rho^5 + k_3\rho^7 + k_4\rho^9 \ , \\ x &= \sin(\theta)\cos(\phi)d \ , \\ y &= \sin(\theta)\sin(\phi)d \ , \\ z &= \cos(\theta)d \ , \end{aligned} \tag{2.6}$$

where $k_1, \ldots, k_4$ are undistortion coefficients, and d is the depth of the unprojected point.

2.2 Random Vectors and Sets

There are different ways to model the uncertainty in object detectors. For instance, each detection can be viewed as a random vector. Random vectors extend the concept of a single random variable to multiple dimensions, where each element represents a distinct quantity subject to randomness. For each detection, the random vector contains both the variables that parameterize the bounding box, as well as the individual class probabilities. Typically the class probabilities are modeled using the discrete categorical distribution[5], and the bounding box parameters are modeled using the continuous normal distribution.

Furthermore, the number of detections in each frame is also unknown. To model this, random finite sets (RFSs) can be used. Random finite sets model an unknown number of unknown variables, variables which themselves could be random vectors. The object detection problem can thus be modeled using an RFS, where an unknown number of objects, each consisting of random vectors, are modeled. Section 2.2.2 provides a brief introduction to RFSs, with a focus on their application to object detection. For a complete rigorous treatment of RFSs, readers are refered to, e.g., [52]–[54].

2.2.1 Normally Distributed Multivariate Random Vectors

A normal distribution, also referred to as a Gaussian distribution, is a continuous probability distribution widely used to model random variables with unknown distributions. This section summarizes some important properties of normal distributions, more information is available in for instance [55]. One reason for the wide use of normal distributions is the central limit theorem. The central limit theorem states that for a sufficiently large random sample from a distribution of finite mean and variance, the random samples are approximately normally distributed. This holds true even if the source distribution is not normal, making the normal distribution appropriate to model the uncertainty of unknown distributions.

One-dimensional normally distributed random variables are denoted $x \sim \mathcal{N}(\mu, \sigma^2)$, where $\mu \in \mathbb{R}$ represents the mean and $\sigma^2 \in \mathbb{R}$ the variance. The probability density function (PDF) of a univariate normal distribution is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} . \tag{2.7}$$

The multivariate normal distribution generalizes the univariate normal to higher dimensions, modeling random vectors. An n-dimensional random vector that is normally distributed can be denoted as $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} \in \mathbb{R}^n$ denotes the

[5]See Appendix B.1 for a brief review of the categorical distribution.

mean vector and $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ the covariance matrix. The PDF of the multivariate normal distribution is

$$f(\mathbf{x}) = (2\pi)^{-n/2} \det(\mathbf{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \quad . \tag{2.8}$$

The covariance matrix of a multivariate normal distribution generalizes the variance to higher dimensional random variables. These matrices have a few important properties. For instance, all covariance matrices are symmetric, i.e., $\mathbf{\Sigma} = \mathbf{\Sigma}^T$, and positive semi-definite, i.e., $\mathbf{x}^T\mathbf{\Sigma}\mathbf{x} \geq 0 \ \forall \mathbf{x} \in \mathbb{R}^n$. Checking that a matrix is symmetric is trivial, however, checking that a matrix is positive semi-definite is slightly more difficult. One way, is to verify that all the eigenvalues are real and non-negative, another, is to utilize the property that all positive semi-definite matricies $\mathbf{A}$ can be Cholesky decomposed into the form

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T \quad , \tag{2.9}$$

where $\mathbf{L}$ is a lower triangular matrix with positive diagonal entries. Thus, if a matrix cannot be decomposed into $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^T$, then it cannot be a covariance matrix.

2.2.2 Random Finite Sets

A random finite set (RFS) is a random variable that takes values as unordered finite sets [52]. For an RFS the number of constituent elements, or cardinality of the set, is random and the elements themselves are random, distinct and unordered. This means that they can be used to model an unknown number of objects, where the objects themselves have unknown properties. Random finite sets are well used in the multi-object tracking literature, e.g., [56], [57], and have recently been used in the context of object detection [14].

In simple terms, an RFS can be specified using a distribution describing the cardinality, and distributions describing the elements, conditioned on the cardinality. The next subsection describes the distributions of some concrete RFSs applicable to object detection.

2.2.2.1 Poisson Multi-Bernoulli Random Finite Sets

The Bernoulli RFS is one of the simplest examples of an RFS. The Bernoulli distribution is probability distribution that can model the outcome of a single event with two possible states[6]. For the application of an object detector, this can be used to model the existence of one object, where the two states are the object existing or not existing. The probability distribution of the Bernoulli RFS is

$$f_{\mathrm{B}}(\mathbb{Y}) = \begin{cases} 1-r & \text{if } \mathbb{Y} = \emptyset \\ rp(y) & \text{if } \mathbb{Y} = \{y\} \\ 0 & \text{if } |\mathbb{Y}| > 1 \end{cases} \quad , \tag{2.10}$$

[6]See Appendix B.2 for a brief review of the Bernoulli distribution.

where $\mathbb{Y}$ is the set of objects assigned to the Bernoulli RFS, r is existence probability and $p(y)$ the single object density, describing the spatial and class densities of the object [56]. As the Bernoulli distribution is only capable of modeling the outcome of a single binary event, the Bernoulli RFS is only capable of modeling a single object, producing a probability of 0 for sets with more than one object.

To combat this, the multi-Bernoulli RFS (MB RFS) can be used. As the name suggests, the MB RFS utilizes multiple Bernoulli RFSs to be able to model several objects. More concretely, taking the union of n independent Bernoulli RFSs yields a multi-Bernoulli RFS with density

$$f_{\mathrm{MB}}(\mathbb{X}) = \sum_{\uplus_{i=1}^{n}\mathbb{X}_i=\mathbb{X}} \prod_{j=1}^{n} f_{\mathrm{B}_j}\left(\mathbb{X}_j\right) \quad , \tag{2.11}$$

where $\mathbb{X}$ is the MB RFS, $\mathbb{X}_1, \ldots, \mathbb{X}_n$ are the independent Bernoulli RFSs, and $\uplus_{i=1}^{n}\mathbb{X}_i = \mathbb{X}$ are all disjoint subsets whose union is $\mathbb{X}$ [14]. However, one problem with using MB RFSs to model arbitrary objects is that the number of independent Bernoulli RFS used to build the MB RFS needs to be larger or equal to the number of objects in a scene. Otherwise, at least one Bernoulli RFS will be assigned to multiple objects[7], leading to a likelihood of 0 for the MB RFS.

There are several ways to extend the MB RFS to handle arbitrary number of objects. One such way, used in model-based tracking, is to model the detected objects using a MB RFS, and model undetected objects using a Poisson point process (PPP) [58]. A Poisson point process is an RFS where all elements are independent and identically distributed, and the cardinality is Poisson distributed[8]. The density of the PPP for multiple objects is

$$f_{\mathrm{PPP}}(\mathbb{X}) = \exp\left(-\int \lambda(x')dx'\right) \prod_{x\in\mathbb{X}} \lambda(x) \quad , \tag{2.12}$$

where $\lambda(x)$ is the intensity function of the PPP, and integral represents the expected cardinality of the set [14]. Combining the multi-Bernoulli RFS to model detected objects with the PPP to model undetected objects we obtain the Poisson multi-Bernoulli RFS with density

$$f_{\mathrm{PMB}}(\mathbb{X}) = \sum_{\mathbb{X}^{\mathrm{U}}\uplus\mathbb{X}^{\mathrm{D}}=\mathbb{X}} f_{\mathrm{PPP}}\left(\mathbb{X}^{\mathrm{U}}\right) f_{\mathrm{MB}}\left(\mathbb{X}^{\mathrm{D}}\right) \quad , \tag{2.13}$$

where $\mathbb{X}^{\mathrm{U}}$ denotes the set of undetected objects, $\mathbb{X}^{\mathrm{D}}$ the set of detected objects, and $\mathbb{X}^{\mathrm{U}} \uplus \mathbb{X}^{\mathrm{D}} = \mathbb{X}$ all possible ways of partitioning $\mathbb{X}$ into these two disjoint sets [14].

[7]This is an example of the widely-known pigeonhole principle, which states that if there are more pigeons than pigeonholes, there must be at least one pigeonhole with at least two pigeons. Here, the pigeons are objects and the pigeonholes are Bernoulli RFSs.

[8]See Appendix B.3 for a brief review of the Poission distribution.

2.3 Uncertainty Modeling

There are many factors that introduce uncertainty in a model, from noisy sensor data to the concept defining what the goal of the model is. To try and formalize this [59] outlined seven different kinds of uncertainty for autonomous perception. These seven different kinds are: conceptual uncertainty, development situation and scenario coverage, situation or scenario uncertainty, sensor properties, labeling uncertainty, model uncertainty, and operational domain uncertainty. However, the authors did not suggest how these quantities should be measured. Instead uncertainty is often decomposed into epistemic and aleatoric uncertainty, which have more concrete mathematical definitions.

Epistemic uncertainty, also known as model uncertainty, measures how certain a model is in using its parameters to describe a dataset. Unknown objects or scenarios not present in the training data, so-called Out Of Distribution (OOD) data, will cause high epistemic uncertainty. Epistemic uncertainty can be generally reduced by increasing the amount of training data [60].

Aleatoric uncertainty, on the other hand, can generally not be reduced by just increasing the amount of training data [61]. Aleatoric uncertainty models the observation noise of data and can be seen as data uncertainty. In autonomous perception, it is likely to observe high aleatoric uncertainty when using RGB cameras at night or when there is heavy rain affecting the LiDAR.

Using the total law of variance, the decomposition of uncertainty into aleatoric and epistemic uncertainty can be viewed as

$$\underbrace{\mathrm{Var}(\mathbf{X})}_{\text{total uncertainty}} = \underbrace{\mathrm{Var}(\mathbb{E}[\mathbf{X} \mid \boldsymbol{\Theta}])}_{\text{epistemic uncertainty}} + \underbrace{\mathbb{E}[\mathrm{Var}(\mathbf{X} \mid \boldsymbol{\Theta})]}_{\text{aleatoric uncertainty}} , \tag{2.14}$$

for a random variable $\mathbf{X}$, and parameters $\boldsymbol{\Theta}$ [62], [63]. From (2.14) we can see how epistemic uncertainty can be reduced by using more data, as the posterior for $\boldsymbol{\Theta}$ will concentrate, reducing the variance of the expectation. In the same way the aleatoric uncertainty approaches the variance of $\mathbf{X}$, or variance independent on the parameters. In practice, completely decomposing uncertainty is challenging, and various models are employed to estimate aleatoric and epistemic uncertainties, with most models capturing both types to some degree.

Estimating these uncertainties in neural networks can be done in several ways. One way to get estimates of uncertainty is to use Bayesian Neural Networks (BNN). BNNs are neural network models that incorporate Bayesian principles, allowing them to express uncertainty in their predictions by representing weights and parameters as probability distributions rather than fixed values. This means that the uncertainties are directly modeled and optimized for in the network structure, making BNNs a compelling choice for estimating uncertainties. However, one main downside of BNNs is that they are generally harder to train and harder to perform inference with compared to non-Bayesian neural networks, especially for large datasets and models [64], [65]. Furthermore, empirical evidence suggests that BNNs often exhibit

inferior predictive performance compared to state-of-the-art non-Bayesian models, as observed in the nuScenes 3D object detection leaderboard [21].

2.3.1 Sampling-based Uncertainty Estimation

Instead of using BNNs, it is possible to extend non-Bayesian networks to model the uncertainty. In the literature, e.g. [12], the methods are typically divided into epistemic and aleatoric uncertainty estimation, however, all methods typically capture both to some extent. The approaches for epistemic uncertainty estimation are typically sample-based, where multiple predictions are generated and seen as samples from some underlying distribution. By assuming a specific distribution, the samples can then be used to estimate the parameters of that distribution.

Two common sample-based approaches to model uncertainty are Monte Carlo Dropout and Deep Ensembles. Monte Carlo dropout works by training a model using dropout [66] and, deviating from how dropout typically is used, keeping the dropout active during inference. By performing inference multiple times with the dropout active, a variation in predictions will be produced that can be used to approximate a distribution. For instance, by assuming that the parameters of a bounding box are normally distributed, one can associate predictions between samples, and for each detection estimate the mean and variance of each parameter. The downside of this is that multiple predictions have to be made at test time, with some suggesting up to 50 predictions [61] to get good results, leading to high latency and computational cost.

Deep ensembles work in a similar fashion, where an ensemble of networks of the same architecture with different initialization of parameters and training data are used to create a spread of predictions [67]. Similar to Monte Carlo dropout, deep ensembles require several predictions to be made at test time. Additionally, they require training separate networks, in contrast to Monte Carlo dropout where the same network can be used to generate each prediction. One benefit of deep ensembles is that they require significantly fewer predictions during inference, with [67] showing that only 5 networks can be sufficient. Common to both of these methods is that very little modification to the base network is required.

2.3.2 Direct Uncertainty Estimation

To model aleatoric uncertainty, the most common method is to use direct modeling. This is done by assuming that the observation noise depends on the input data, i.e. that uncertainty is heteroscedastic. In direct modeling the output is assumed to follow a certain distribution – e.g. assuming that the outputs making up the 3D bounding boxes follow a Gaussian distribution – and then adding extra output layers in the model to predict the parameters of the distribution [60]. This requires modifying the base network by changing the output layers and loss functions to efficiently learn the variance. However, this method does not directly imply a significant increase in computational load at test time.

A few different loss functions have been proposed to learn the distribution parame-

ters. In contrast to many other learning tasks, there is no ground truth which can be used to directly learn the parameters. This makes sense as it is impossible to annotate how uncertain a model is in a given scenario. Instead, we have to condition the models to output uncertainty estimates using proxies. The most common example of this is using the negative log likelihood (NLL) of the underlying distribution. The likelihood of a distribution is the probability of observing a certain set of data points given the parameters. By minimizing the NLL, we maximize the probability of observing the data, making the NLL a good candidate for a loss function. For a multivariate normal distribution, the NLL is

$$\text{NLL} = \frac{1}{2N} \sum_{n=1}^{N} (\boldsymbol{z}_n - \boldsymbol{\mu}_n)^T \boldsymbol{\Sigma}_n^{-1} (\boldsymbol{z}_n - \boldsymbol{\mu}_n) + \log \det \boldsymbol{\Sigma}_n \ , \tag{2.15}$$

where N is the size of the dataset, $\boldsymbol{z}_n$ are the ground truth parameters, $\boldsymbol{\mu}_n$ is the predicted mean, and $\boldsymbol{\Sigma}_n$ is the predicted covariance matrix. When using the NLL as a loss function, the model will be penalized for predicting a low variance if the error is high, due to the inverse of the covariance matrix magnifying the error. In a similar way, the model is punished for predicting a high variance if the error is low, due to the additive last term in (2.15).

One downside of the NLL as a loss function is that it penalizes underestimating the variance much more severely than overestimating it, leading to miscalibrated, underconfident predictors. This is due to the fact that, for a fixed error, the NLL grows with $1/x$ when the variance tends towards 0, but with $\log(x)$ when the variance tends to infinity. This has led to using alternative loss functions to estimate the distribution parameters. One such loss function is the energy score loss. The energy score can be written as

$$\text{ES} = \frac{1}{N} \sum_{n=1}^{N} \left(\frac{1}{M} \sum_{i=1}^{M} \|\mathbf{z}_{n,i} - \boldsymbol{z}_n\| - \frac{1}{2M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \|\mathbf{z}_{n,i} - \mathbf{z}_{n,j}\| \right), \tag{2.16}$$

where M is a fixed number of samples, and $\boldsymbol{z}_{n,i}$ is the i-th independent and identically distributed sample from $\mathcal{N}(\boldsymbol{\mu_n}, \boldsymbol{\Sigma_n})$ [13]. This can be efficiently approximated using the Monte-Carlo method [68], yielding

$$\text{ES} = \frac{1}{N} \sum_{n=1}^{N} \left(\frac{1}{M} \sum_{i=1}^{M} \|\mathbf{z}_{n,i} - \boldsymbol{z}_n\| - \frac{1}{2(M-1)} \sum_{i=1}^{M-1} \|\mathbf{z}_{n,i} - \mathbf{z}_{n,i+1}\| \right) \ , \tag{2.17}$$

significantly reducing the number of samples required.

By modeling object detection as a set prediction problem, and representing the detections as coming from a Poisson multi-Bernoulli RFS, it is possible to use the NLL of the PMB RFS density as a loss function. Naïvely, one can obtain the NLL by computing the likelihood of observing the ground truths, given the density (2.13), parameterized by the detections. However, this considers all possible assignments of objects to ground truths, which grows super-exponentially. Instead, [69] proposes to consider only the most likely assignments, arguing that the contribution of non-likely assignments are negligible. By constructing an optimal assignment problem,

and extracting the Q likeliest assignments $\boldsymbol{A}_1^*, \cdots, \boldsymbol{A}_Q^*$ using Murtys algorithm [70], the PMB-NLL can be approximated using

$$\text{PMB-NLL} \approx \int \lambda\left(y'\right) \mathrm{d}y' - \log\left(\sum_{q=1}^{Q} \prod_{y \in \mathbb{Y}^U\left(\boldsymbol{A}_q^*\right)} \lambda(y) \prod_{k=1}^{m} f_{\mathrm{B}_k}\left(\mathbb{Y}_k\left(\boldsymbol{A}_q^*\right)\right)\right) \quad , \tag{2.18}$$

where $\lambda(x)$ is the PPP intensity function, f_{B_k} is the density of a Bernoiulli RFS (2.10), $\mathbb{Y}_k\left(\boldsymbol{A}_q^*\right) = \left\{y_j \in \mathbb{Y} \mid \left[\boldsymbol{A}_q^*\right]_{k,j} = 1\right\}$, denoting the ground truths assigned to Bernoulli RFS k, and $\mathbb{Y}^U\left(\boldsymbol{A}_q^*\right) = \mathbb{Y} \backslash \cup_{i=1}^{m} \mathbb{Y}_i\left(\boldsymbol{A}_q^*\right)$, denoting the ground truth elements matched to the PPP.

2.3.3 Evaluating Uncertainty Estimates

By only evaluating probabilistic object detectors using regular object detection metrics, such as classification accuracy, mAP, or NDS, the uncertainty is not directly captured[9]. Instead, different metrics, ideally proper scoring rules, have to be used. A scoring rule is a function that maps from a predicted probability distribution and a ground truth value to a single scalar value, and a proper scoring rule is a scoring rule that is only minimized if the predicted probability distribution is the ground truth distribution [12]. One example of a proper scoring rule is the Negative Log Likelihood (NLL) and it has additionally been shown that mAP, and thus also NDS, are not proper scoring rules [13]. Therefore, by applying the NLL – using either the Poisson multi-Bernoulli representation in (2.18) or the more common representation in (2.15) – one obtains a more mathematically grounded score.

Another way to gain insights in the uncertainty estimates of a probabilistic object detector is to look at the calibration of the estimates. Calibration measures how accurate or biased the uncertainty estimates are. A calibrated model is a model that accurately estimates the probability of an event. In other words, the model is neither overconfident nor underconfident in its uncertainty estimates. This property can be visualized with a calibration plot, see Figure 2.7, where generated distributions are plotted with predictive probability on the x-axis and the empirical probability on the y-axis. If a network produces estimates such that the empirical probability is generally higher than the predictive, then the network is underconfident, and vice versa. To generate this calibration plot for regression, m confidence intervals are chosen, $0 \leq p_1 < p_2 < \cdots < p_m \leq 1$. For each threshold p_j, the empirical frequency is calculated

$$\hat{p}_j = \frac{|\{y_t | F_t(y_t) \leq p_j, t = 1, \ldots, T\}|}{T} \quad , \tag{2.19}$$

where T is the total number of detections made with the predictor, and F_t is the cumulative distribution function of the random variable corresponding to detection

[9] In mAP and NDS, the classification confidence is indirectly considered during the matching step. However, only the ranking of the detections is important; the actual confidence value itself is not captured.

t [71]. Furthermore, the calibration error can be obtained by

$$\mathrm{CE}_{\mathrm{reg}} = \sum_{j=1}^{m} (p_j - \hat{p_j})^2 \ . \tag{2.20}$$

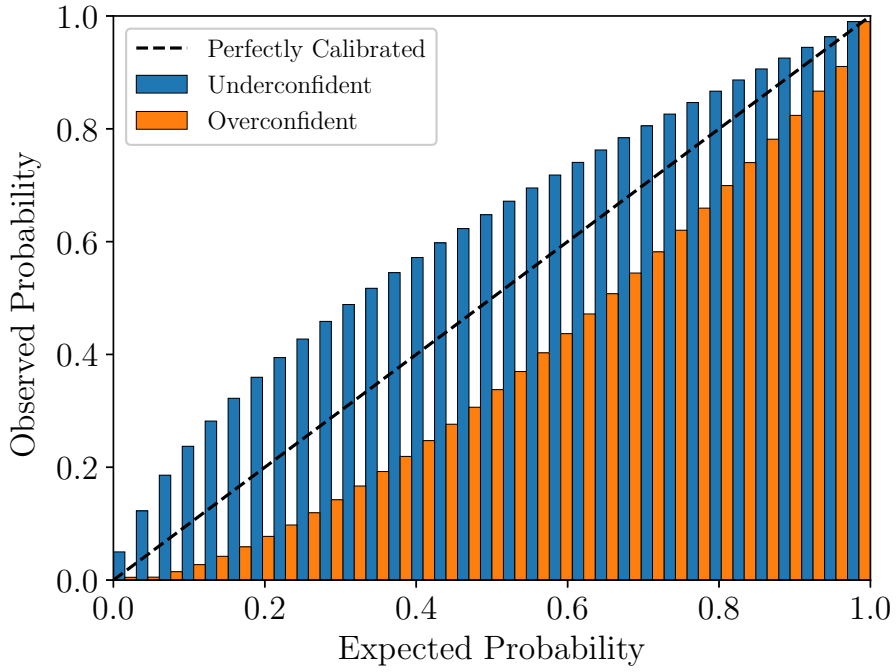


Figure 2.7: Example calibration plot. For the blue predictor the observed probability is higher than the expected probability, implying that it is underconfident. For the orange predictor the observed probability is lower than the expected probability, implying that it is overconfident. The black dashed line shows a perfectly calibrated predictor, where the observed probability is equal to the expected probability.

All previously mentioned methods require ground truth annotations. One way to reduce the uncertainty to a single number in the absence of annotations is to compute the entropy of the predicted random variables. The entropy captures the average level of uncertainty in the outcome, however, does not measure how good the estimate is. This means that it has limited application in evaluating detectors. Instead, the entropy can serve as an indicator of the overall uncertainty, even on data that is not annotated. Generally, the entropy of a discrete random variable $\mathbf{X}$ with probability mass function $p(x)$ is

$$H = -\sum p(x) \log p(x) \ , \tag{2.21}$$

where the sum is taken over all possible values of $\mathbf{X}$. The entropy can also be extended to continuous random variables. Commonly, the differential entropy is used

$$H = -\int f(x) \log f(x) dx \ , \tag{2.22}$$

where f the PDF of the random variable, and the integral is taken over the support of the PDF.

Taking the example of modeling object detection as random vectors, the entropy is divided into two parts. One classification part, to model the categorical class probabilities, and one regression part, to model the bounding box parameters. The entropy for the classification case is

$$H_{\text{cls}} = -\sum_{c \in \mathbb{C}} p(c) \log(p(c)) \ , \tag{2.23}$$

where $\mathbb{C}$ is the set of classes, and $p(c)$ is the class probability for class c. In this case, the entropy is maximized when all classes have the same probability, which is the case where the model is the most uncertain. For the bounding box parameters, assuming that they are normally distributed, the entropy is

$$H_{\text{reg}} = \frac{1}{2} \log\left((2\pi e)^N \det(\mathbf{\Sigma})\right) \ , \tag{2.24}$$

where $\mathbf{\Sigma} \in \mathbb{R}^{N \times N}$ is the covariance matrix of the random vector. Here, the entropy is only dependent on the determinant of the covariance matrix, capturing the overall regression uncertainty. When instead modeling object detection as random finite sets, the entropy can be computed by inserting the PDF (2.13) into (2.22). However, as for the NLL of the PMB RFS, the analytical solution for the entropy is intractable.

3

Methods

This chapter describes the methods used during the thesis to achieve the thesis objectives described in Section 1.1. We describe how the baseline models were modified and implemented. Additionally, the uncertainty modeling is detailed and the training procedures are described.

3.1 Baseline BEVFusion Modifications

To obtain a baseline for comparision, the state-of-the-art multi-modal 3D object detector BEVFusion [20] is implemented. The implementation is based on the open source repository[1] published by the authors of the BEVFusion paper, which utilizes the PyTorch-based 3D detection framework MMDetection3D [72], [73]. The original BEVFusion is implemented for the nuScenes dataset, and a few key changes are made for ZOD. These include modeling lens distortion, modifying class predictions, changing the bounding box parametrization, and adapting for a longer detection range.

3.1.1 Modeling Lens Distortion

The images used in nuScenes are rectlinear, meaning that the optical distortion is negligible and that the simple pinhole camera model can be used in the lifting step. In ZOD, the camera is fitted with a wide-angle lens and the distortion is non-negligible. One solution is to rectify the images by undistorting and cropping, the process of which is shown in Figure 3.1. However, this reduces the effective area covered by the camera, which results in reduced detection capability. Therefore, we instead modify the lifting step in BEVFusion – the original lifting step is illustrated in Figure 2.4 – to use a camera model that can capture the distortion, allowing the utilization of the full field of view of the camera. We here apply the Kannala-Brandt projection model [51], performing the unprojection from 2D to 3D using (2.6).

[1] https://github.com/mit-han-lab/bevfusion

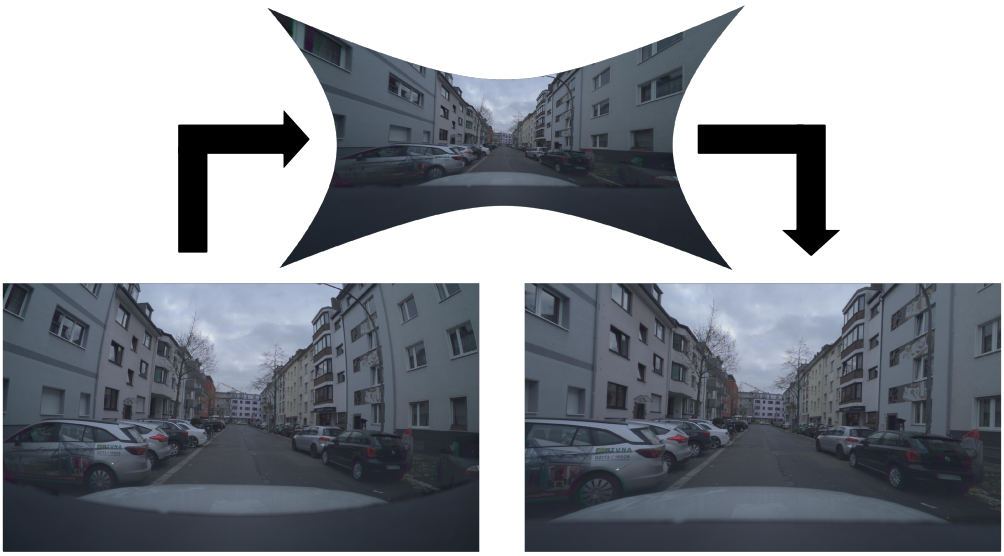


Figure 3.1: The image rectification process on an example image from ZOD[2][1]. The original distorted image (left) is undistorted (top) and cropped to produce the rectified image (right). The rectified image does not include the vehicle near the bottom right corner, and the car on the bottom left becomes more occluded.

3.1.2 Bounding Box Parametrization

Further modification is made to the parametrization of the bounding boxes. The original encoding of the bounding boxes is

$$[\delta x, \delta y, z, \log(l), \log(w), \log(h), \sin(\theta), \cos(\theta)] \ , \tag{3.1}$$

where $\delta x, \delta y$ are sub-voxel refinements of the bounding box center in the heatmaps, z is the height over the ground, $\log(l)$, $\log(w)$, $\log(h)$ are the logarithmized sizes, and θ is the yaw angle. This representation has some benefits, for instance, the logarithmized sizes mean that the decoded size of a bounding box is always non-negative. However, when modeling the variances as in Section 2.3.2, the variables representing the bounding boxes dictate what variance we are estimating. This means that we, for instance, would estimate the variance of $\log(l)$ instead of the variance of l. To address this, it is possible to scale the variances afterwards. The variance of the bounding box center can be analytically converted using that $\mathrm{Var}(a + b\mathbf{X}) = b^2\mathrm{Var}(\mathbf{X})$, for a random variable $\mathbf{X}$, and two scalars a, b. However, for the logarithm, no analytical solution exists, meaning that approximation errors would be introduced. Therefore, we instead change the bounding box representation to

$$[x, y, z, l, w, h, \sin(\theta/2), \cos(\theta/2)] \ , \tag{3.2}$$

where x, y, z denote the 3D center of the bounding box in the ego-vehicle frame. In addition, the size of the bounding box is described by l, w, h, and $\sin(\theta/2), \cos(\theta/2)$ represent the $\mathbf{q}_z$,$\mathbf{q}_w$ components of a quaternion where the axis of rotation is the

[2]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details.

z-axis. Doing this we obtain more interpretable variances, as they are in standard units.

3.1.3 Computing Existence Probabilities

In the original model, only a confidence score proportional to the heatmap peak is computed for each detection. This confidence score indicates the likelihood of there being an object, but is not a calibrated existence probability. Additionally, no class probabilities are obtained. To obtain these, we modify the detection head by adding a background class. The background objects are not part of the labeled dataset, instead the detections that are not matched to a ground truth get assigned to the background class. The model then learns to predict the background class for proposals that are not likely to be assigned to a ground truth.

By taking the softmax of the scores for each class, we obtain actual class probabilities, which is a more interpretable representation. Furthermore, the existence probability r can be computed by $r = 1 - p_{\text{background}}$, where $p_{\text{background}}$ is the class probability of the background class.

3.1.4 Training Procedure

The training is done using a 90:10 train-validation split and with mostly the same hyperparameters as the original paper. Some parameters were however changed due to the change in the sensor setup. The original models utilized multiple cameras, considering detections in a 102.4m$\times$102.4m square around the vehicle. In ZOD, only a single forward facing camera is available. We therefore consider only detections in the FOV of the camera, and limit the range to 128m. To deal with the increased range, and to compensate for the uncropped wide-angle camera, the image size was increased from 704×256 to 1200×672.

With BEVFusion modified to work on ZOD, we first train the camera-only BEVFusion model, based on BEVDet [43], and the LiDAR-only BEVFusion model, based on TransFusion [35]. The single-sensor models were trained for 25 epochs using the AdamW optimizer [74]. A cyclic learning rate policy was used [75], where the learning rate starts low, increases to a peak, then decreases back down during the training process.

To train the fusion model, the weights of a LiDAR-only model trained for 25 epochs are first loaded. The camera branch is then added – without any pre-training – and the weights of the camera-LiDAR model are jointly fine-tuned. The model is trained for 10 epochs using AdamW. Here, a cosine annealing learning rate policy [76] was used, starting with a high learning rate to allow large initial changes – to allow the model to incorporate the camera information – and decreasing smoothly like a cosine to fine-tune these changes as training progresses.

3.1.5 Obtaining Baseline Regression Uncertainties

The original BEVFusion models do not explicitly model regression uncertainty. However, to get a baseline for the uncertainty estimates, we compute the sample variances of the errors produced by the models over the entire validation set. These sample variances are then used by assuming homoscedasticity, i.e., that the uncertainty is the same for all detections. The sample variances are then used to compute the uncertainty metrics described in Section 2.3.3.

3.2 Direct Uncertainty Modeling

The approach investigated to estimate uncertainty is the direct modeling approach, where the parameters of some output distribution are regressed directly using a modified loss function. First, we determine the distribution with which we model the regression parameters.

3.2.1 Distribution of Regression Parameters

To determine which distribution should be used to model the bounding boxes, we compute and analyze the error between the ground truth bounding boxes and the associated predicted bounding boxes from the BEVFusion model. Figure 3.2 shows examples of errors with fitted distributions, showing that a normal distribution is a reasonable approximation to model the random variables with.

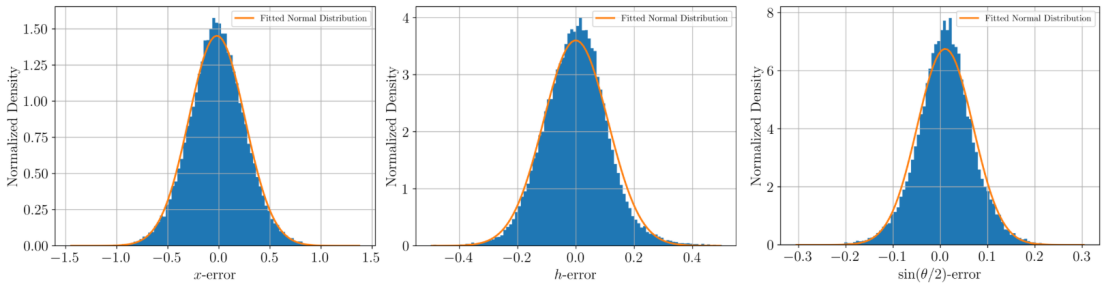


Figure 3.2: Histogram of bounding box parameter errors for x, h, and $\sin(\theta/2)$ with fitted univariate normal distributions. Errors are approximately normal.

To fully model the parameters using a multivariate normal distribution, we need to predict a 8×8 covariance matrix, consisting of 64 values. Utilizing the fact that covariance matricies are symmetric, the number drops to 36, which is still a significant number of extra variables to regress for each bounding box. If all bounding box parameters were linearly independent, one could reduce the number of parameters further, as the off-diagonals in the covariance matrix would be 0. Consequently, only the 8 diagonal elements of the covariance matrix would need modeling.

To analyze the linear dependence between the bounding box parameters we compute the correlation between bounding box parameter errors. Figure 3.3 shows the correlation between the bounding box parameters. We note that the correlation

is non-zero for the off-diagonal terms, meaning that the errors are not completely linearly independent. However, using a diagonal covariance matrix is still a fair approximation, as the magnitude of the correlations are generally low. Interestingly, there is nearly no correlation between $\sin(\theta/2)$ and $\cos(\theta/2)$, due to the fact that correlation does not capture non-linear dependence.

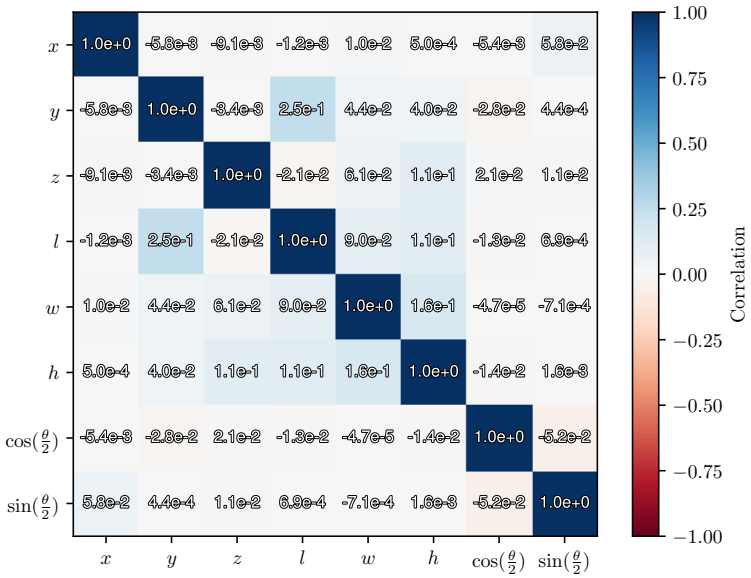


Figure 3.3: The correlation between bounding box parameter errors of true positive matches using a distance threshold of 2m. The correlations are generally small. However, some errors display a noteworthy correlation, e.g., y and l.

3.2.2 Learning Regression Uncertainties

With the errors analyzed, we first choose to model each bounding box as a normally distributed multivariate random vector, with a diagonal covariance matrix. This means that we need to extend the head of the BEVFusion model to output an additional 8 parameters for each bounding box.

To ensure that the predicted covariance matrix is positive semi-definite, we predict the elements of the lower triangular matrix $\mathbf{L}$ – where the exponential function is applied to the diagonal elements to ensure positivity – and obtain the covariance matrix using $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^T$, following (2.9). For a diagonal covariance matrix, the model output of a single parameter can thus be seen as the log of the standard deviation: $\log \sigma$.

To learn these parameters, we change the loss function used for the bounding boxes from the standard L1 loss and train models using the two probabilistic losses NLL (2.15) and ES (2.17). To train these models, we follow the same procedure as in Section 3.1.4.

Secondly, we model the output detections jointly as a Poisson multi-Bernoulli random finite set, as described in Section 2.2.2.1. We here employ the approximation of the PMB-NLL (2.18) as a loss function. To decide the number of assignments Q when computing the PMB-NLL, we investigated the effect on both PMB-NLL and computation time for different values of Q. The results can be seen in Table 3.1. With PMB-NLL decreasing slowly, and evaluation time growing quickly for increasing Q, a value of $Q = 1$ is chosen during training. To train this model we use the models trained with energy score as a base, and fine-tune for an additional 10 epochs using a cosine annealing learning rate policy.

Table 3.1: Mean PMB-NLL score and the corresponding time it took to evaluate on the ZOD validation set for differing number of assignments Q. Only the computation time of the PMB-NLL score considered.

Assignments (Q)	Mean PMB-NLL ↓	Evaluation Time (s) ↓
1	366.29	**313**
3	365.62	575
5	365.38	805
10	365.07	1413
15	364.95	2043
20	364.95	2638
25	364.88	3305
50	364.71	6430
100	**364.53**	12409

3.2.3 Out of Distribution Detection

A few different tests were employed to evaluate the use of the predicted uncertainties for detecting out of distribution cases. First, we compute the average classification entropy (2.23), and the average regression entropy (2.24), for each of the frames in the dataset using the best performing model. This is done to investigate and analyze the cases where the model is the most uncertain.

Second, we train a model where there is a shift in the distribution of the training and validation sets. Two different shifts are tested, one where there is a shift in location between the training and validation sets, and one where there is a class-shift. For the location shift, the model is trained using only data from Sweden, and then evaluated on data from both Sweden and Italy. For the class-shift, the model is trained on frames where only the subclass `Car` of the `Vehicle` class was present – meaning that the model never sees larger vehicles such as trucks or busses – and then evaluated on all frames.

Finally, we evaluate the applicability of using the entropies for active learning. We train a model using only 50% of the training data, and then compute the entropies for the remaining half. The training data is then extended using the frames with the highest entropies, and compared against randomly sampling from the remaining frames.

4

Results

This chapter contains the evaluation of the methods described in Section 3. First, the results of the baseline models are presented. These are evaluated using both traditional non-probabilistic object detection metrics, and probabilistic metrics by assuming a constant variance to be used for comparison. After this, the results of the models with explicit uncertainty modeling are presented. Finally, the out of distribution detection of the probabilistic models is evaluated by investigating the entropies produced when subjected to OOD data.

4.1 Baseline Results

The baseline camera-only, LiDAR-only and fusion models are evaluated using a mix of dynamic and static object classes. The chosen top-level classes are `Vehicle`, `VulnerableVehicle`, `Pedestrian`, `TrafficSign`, and `TrafficSignal`. Here, `Vehicle` contains protected vehicles such as cars and buses, whereas `VulnerableVehicle` contains unprotected vehicles such as motorcycles and strollers. Additionally, only objects within 128 meters from the ego-vehicle are considered for evaluation. Following [24], the different Euclidean matching thresholds for detections are chosen as 0.5m, 1m, 2m, and 4m. The results of evaluating the baseline models for the task of 3D object detection on the validation set, using the non-probabilistic metrics NDS and mAP, can be found in Table 4.1. We note that, as expected, the model performance of the camera-only model is significantly worse than the LiDAR and fusion models on all metrics. A qualitative comparison is shown in Figure 4.1, highlighting the strength of LiDAR-based detectors.

Table 4.1: Evaluation of baseline models with non-probabilistic metrics on ZOD. The camera-only and LiDAR-only models were trained for 25 epochs. The fusion model was fine-tuned for 10 epochs by loading the weights of the trained LiDAR-only model where applicable. Bolded values indicate the best scores across all modalities.

Model	NDS↑	mAP↑	mAP↑ `Vehicle`	mAP↑ `VulnerableVehicle`	mAP↑ `Pedestrian`	mAP↑ `TrafficSignal`	mAP↑ `TrafficSign`
Camera	0.476	0.225	0.372	0.198	0.178	0.231	0.147
LiDAR	0.750	0.628	0.871	0.454	0.680	0.567	0.566
Fusion	**0.772**	**0.663**	**0.889**	**0.490**	**0.710**	**0.644**	**0.583**

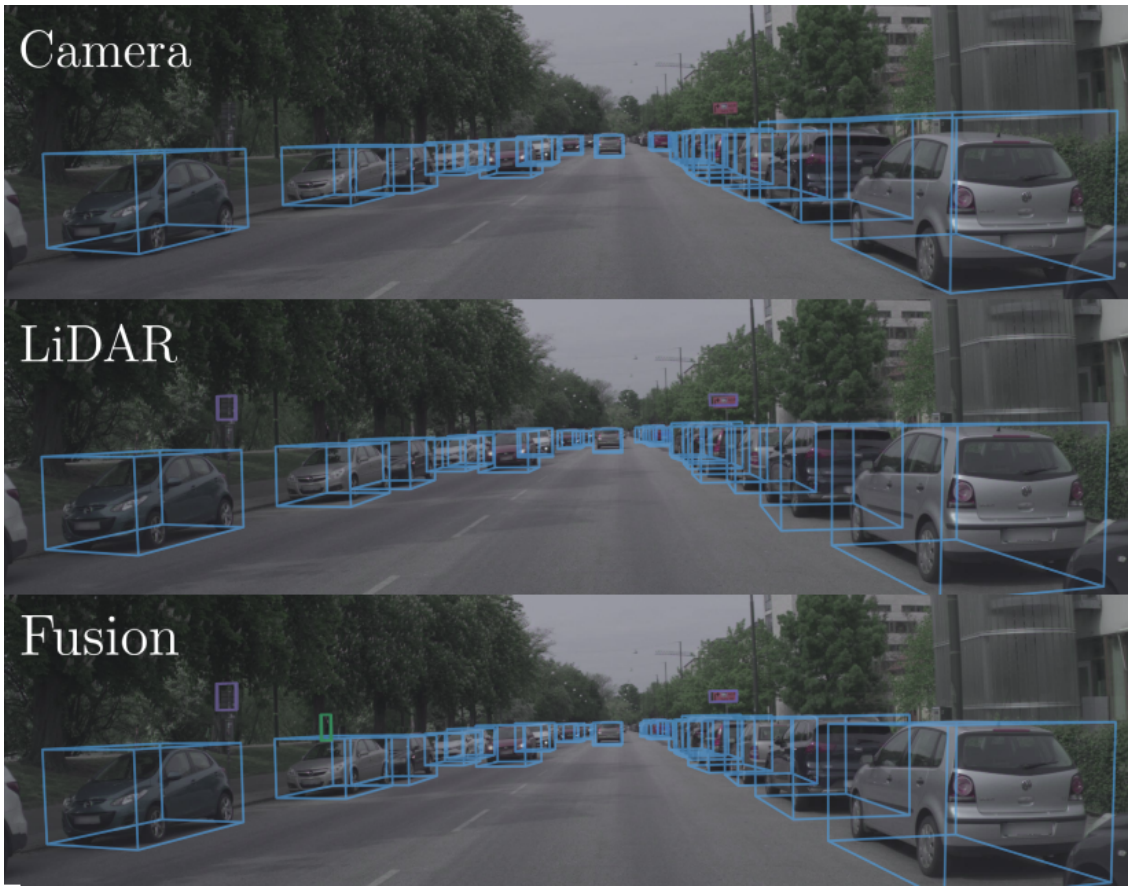


Figure 4.1: Qualitative comparison of baseline detectors on an example from ZOD[1][1]. Only detections with existence probability greater than 0.5 are shown. Blue bounding boxes represent predictions of class `Vehicle`, green the predictions of `Pedestrian`, and purple the predictions of `TrafficSign`. The camera-only model (top) misses the traffic signs and the occluded pedestrian. The LiDAR-only model (middle) detects the traffic signs and performs better at range. Finally, the fusion model (bottom) further detects the occluded pedestrian and produces more tightly fitting bounding boxes.

Additionally, we compute the sample variances of the errors and use them to model the uncertainties homoscedastically, i.e., assuming that the variance does not change with the data. The resulting models are evaluated using the calibration error (2.20), as well as the proper scoring rules NLL (2.15) and PMB-NLL (2.18). This selection of metrics provide the capability of measuring overall performance of the predictive models, through the NLL and PMB-NLL, as well as providing a quantification of the model bias, through the calibration error. The results of evaluating the homoscedastically modeled baselines can be found in Table 4.2. We see how the NLL-based metrics improve as the predictive performance improves, and that the more accurate models are slighly better calibrated.

[1]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details.

Table 4.2: Evaluation of baseline models with probabilistic metrics on ZOD. The uncertainty is modeled homoscedastically by using the sample variance of the dataset for each prediction. Bolded values indicate the best scores across all modalities.

Model	NLL↓	PMB-NLL↓	CE_{reg} ↓
Camera	1.12	9742.21	0.053
LiDAR	0.17	3535.72	0.032
Fusion	**0.04**	**3042.31**	**0.029**

4.2 Probabilistic Results

A similar procedure as in Section 4.1 is used to evaluate the models where the uncertainties were modeled heteroscedastically, i.e., where the variance changes for each prediction. The LiDAR-only and fusion models trained using NLL (2.15), energy score (2.17) and PMB-NLL (2.18) are here evaluated. First, the non-probabilistic metrics are computed and the resulting metrics can be seen in Table 4.3. We note that the highest predictive performance is achieved with the energy score models, closely followed by the PMB-NLL models. Compared to the baseline models, the energy score models performed similarly, whereas the NLL and PMB-NLL models had slightly worse performance.

Table 4.3: Evaluation of the models trained using probabilistic losses with non-probabilistic metrics on ZOD. NLL and ES LiDAR models trained for 25 epochs, PMB-NLL model finetuned for 10 epochs on the ES model. Fusion models finetuned for 10 epochs on their LiDAR counterparts. Bolded values indicate the best scores for each modality.

	Model	NDS↑	mAP↑	mAP↑ Vehicle	mAP↑ VulnerableVehicle	mAP↑ Pedestrian	mAP↑ TrafficSignal	mAP↑ TrafficSign
Camera	NLL	0.459	0.208	0.366	0.190	0.150	0.204	0.132
	ES	**0.477**	**0.224**	**0.371**	0.201	**0.166**	**0.230**	**0.151**
	PMB-NLL	0.468	0.218	0.369	**0.207**	0.163	0.209	0.142
LiDAR	NLL	0.724	0.587	0.857	0.376	0.653	0.521	0.531
	ES	**0.751**	**0.631**	0.874	**0.455**	**0.694**	0.562	0.568
	PMB-NLL	0.730	0.622	**0.875**	0.426	0.692	**0.577**	**0.578**
Fusion	NLL	0.761	0.650	0.880	0.475	0.699	0.575	0.621
	ES	**0.769**	**0.664**	0.888	**0.508**	**0.707**	**0.636**	**0.580**
	PMB-NLL	0.766	0.659	**0.907**	0.485	0.703	0.625	0.577

To compare the uncertainty estimates of the models, we further evaluate the models with the probabilistic metrics. Here, as opposed to the baseline, the predicted uncertainties are used to compute the scores. Table 4.4 shows the resulting metrics. We here note that the best results on the proper scoring rules are achieved by the models trained using PMB-NLL. However, the energy score models are slightly better calibrated. Qualitative comparisons of the different losses, using a LiDAR

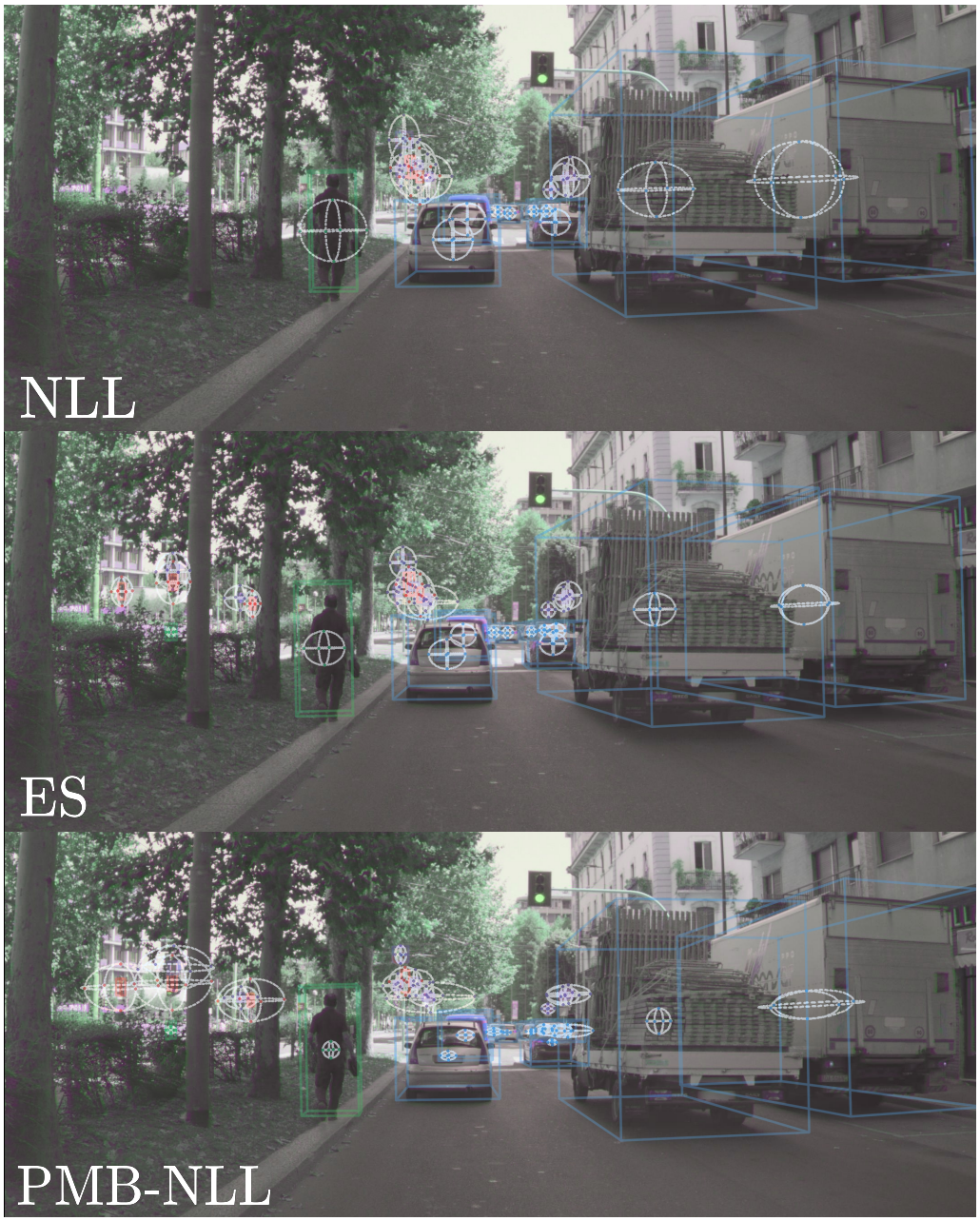


Figure 4.2: Qualitative comparison of LiDAR models trained with different losses on an example from ZOD[2][1]. Only detections with existence probability greater than 0.5 are shown. Blue bounding boxes represent predictions of class `Vehicle`, yellow the predictions of `VulnerableVehicle`, green the predictions of `Pedestrian`, purple the predictions of `TrafficSign`, and red the predictions of `TrafficSign`. The 95% confidence of the center point of each object is visualized using ellipses. The model trained using NLL (top) is less confident, whereas the energy score model (middle) and fusion model (bottom) perform predictions with higher confidences. Figure C.1 visualizes the same predictions in BEV.

[2]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details.

model, are shown in Figure 4.2 and Figure C.1. It can be seen that the uncertainties produced by the NLL model are larger in magnitude than the ES model, this is further highlighted in the calibration plot shown in Figure 4.3.

Table 4.4: Evaluation of the models trained using probabilistic losses with probabilistic metrics on ZOD. Bolded values indicate the best scores for each modality.

	Model	NLL↓	PMB-NLL↓	CE_{reg} ↓
Camera	NLL	-0.19	205.6	0.052
	ES	-0.43	180.4	**0.042**
	PMB-NLL	**-0.53**	**152.3**	0.045
LiDAR	NLL	-0.66	106.2	0.045
	ES	-1.09	84.5	**0.020**
	PMB-NLL	**-1.25**	**69.2**	0.031
Fusion	NLL	-1.17	61.2	0.039
	ES	-1.23	58.5	**0.019**
	PMB-NLL	**-1.32**	**49.7**	0.027

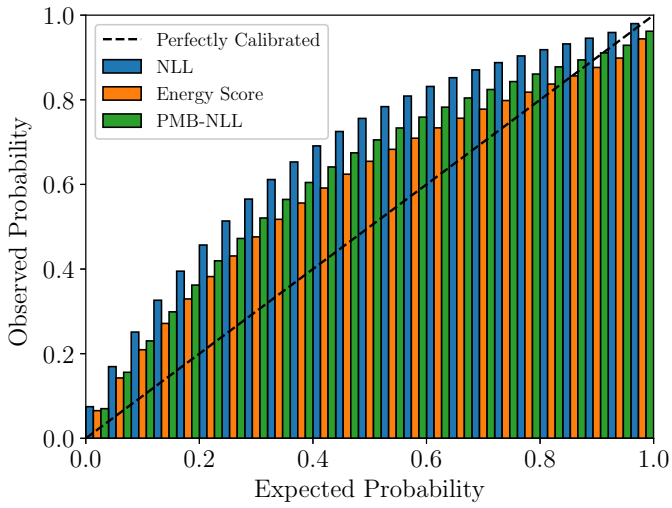


Figure 4.3: Calibration plot for the three fusion models trained using NLL (blue), energy score (orange), and PMB-NLL (green). All models are underconfident in their uncertainty estimates, with the NLL model showing the worst performance.

4.3 Training and Inference Time

One important metric for the real-time applicability of a model is its inference time. The inference time measures how long it takes for the model to perform its predictions when given input data. We measured this by performing inference 1000 times on the same frame with the different models, using an NVIDIA A100 GPU [77]. The results of this, together with the time it took to train each model, using four NVIDIA A100 GPUs, can be seen in Table 4.5. We see that the training time is

longer for the models where uncertainty is estimated, especially for the PMB-NLL models. The primary factor for the increased total training time of the PMB-NLL models is that they are finetuned on an already trained energy score model, meaning that the total number of epochs are increased. To compute the PMB-NLL a large optimal assignment problem additionally needs to be solved, further increasing the computational load. When it comes to inference times, however, we note that the times are very similar, with no model having a larger increase than 1% compared to their baseline.

Table 4.5: Inference time and training time for all evaluated models. Inference performed using a single NVIDIA A100 GPU and training using four NVIDIA A100 GPUs. For the fusion and PMB-NLL models that are finetuned, the full training time including base models are shown. Bolded values indicate the best times for each modality.

	Model	Inference Time (ms)↓	Training Time (h)↓
Camera	Baseline	**55.617**	**31.09**
	NLL	55.942	31.63
	ES	56.158	32.03
	PMB-NLL	56.198	46.92
LiDAR	Baseline	**89.726**	**26.35**
	NLL	90.285	27.43
	ES	90.256	28.69
	PMB-NLL	90.374	43.70
Fusion	Baseline	**135.398**	**42.68**
	NLL	135.786	43.85
	ES	135.814	44.91
	PMB-NLL	135.997	64.91

4.4 Out of Distribution Detection

To evaluate how well the models can detect out of distribution scenarios, we follow the methodology outlined in Section 3.2.3. First, to better understand the types of scenes with high entropy, we investigated the scenes with the highest regression and classification entropies. In total 20 frames were investigated, using the 10 highest entropies of both types. Among these frames, the majority – 18 of them – highlighted cases where the main LiDAR data was missing or heavily degraded. The remaining cases were made up of a scene with a weird blur and a scene with many occluded objects at night. Figure C.2 shows some examples of these frames.

Second, we investigate if the model can detect shifts in the data distribution. A LiDAR-only model was trained on only frames from Sweden, and then evaluated on frames from Sweden and Italy. Histograms showing the regression entropies on the frames from Sweden compared to the frames from Italy can be seen in Figure 4.4. We see that there is a minor difference in the mean entropies, where the Italian

frames show a higher mean. Additionally, we trained a model using frames where only the subclass `Car` of the `Vehicle` class is present. The model was then evaluated on all frames. Histograms showing the regression entropies of frames containing only the subclass car of the vehicle class and frames containing other subclasses can be seen in Figure 4.5. We note a major difference in the mean entropies and that a meaningful separation can be made between the cases.

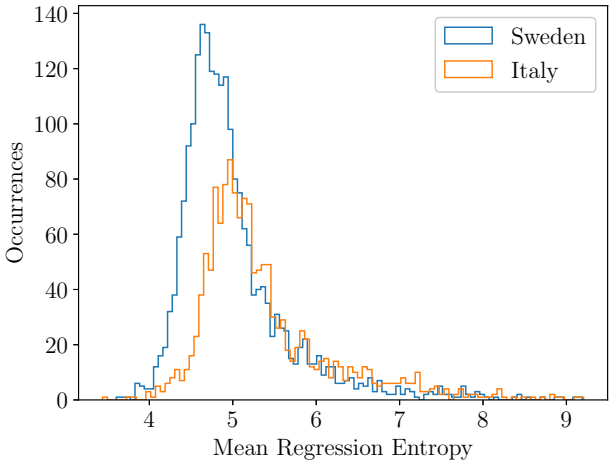


Figure 4.4: Histogram of regression entropies for location-based dataset shift. A LiDAR-only model was trained only on frames from Sweden and evaluated on frames from Sweden and Italy. The blue histogram shows the mean entropy when evaluating the model on Swedish scenes, and the orange histogram shows the mean entropy when evaluating on Italian scenes.

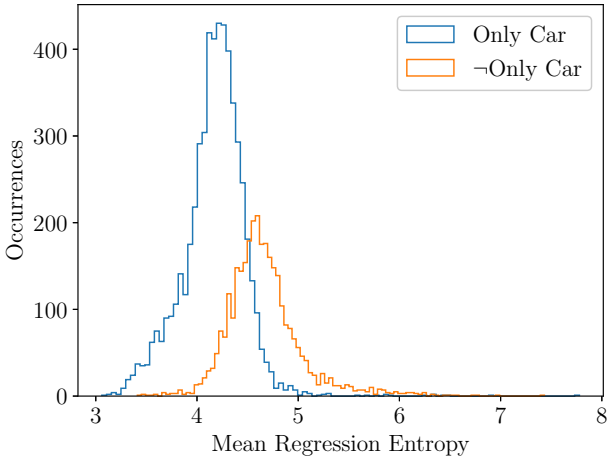


Figure 4.5: Histogram of regression entropies for class-based dataset shift. A model was trained seeing no instances of large, i.e., non-car, vehicles, and then evaluated on a dataset containing them. The blue histogram shows the entropy for frames where only the car subclass is present, and the orange histogram shows frames where large subclasses, e.g. busses and trucks, were present.

Finally, we also take a more general approach to determine the strength of using these entropies for active learning. We train a model using only half of the dataset,

and compute the entropies on the other half. We then compare training a model with a random extension of the dataset, with extending the dataset using the frames with the highest entropies. The results can be seen in Table 4.6 and we note that training using the high entropy set led to marginally improved results.

Table 4.6: Evaluation of active learning using computed entropies. A LiDAR-only model is trained using 50% of the dataset, 50% of the dataset together with a randomly sampled extension, and 50% of the dataset together with the largest entropies of the remaining set appended.

Dataset	mAP ↑	NLL ↓	PMB-NLL ↓
50%	0.615	-0.99	90.5
50% + Random	0.624	-1.04	87.5
50% + High Entropy	**0.629**	**-1.07**	**85.9**

5

Conclusion

In this chapter, the results presented in Chapter 4 are analyzed and discussed in relation to the thesis objectives. Finally, the conclusions of the thesis are drawn.

5.1 Discussion

The main research questions we wanted to answer were related to how the predictive performance is impacted when modeling uncertainties, how to estimate and evaluate the uncertainties, and how they can be utilized for safer autonomous vehicles. The following discussion aims to address these points as well as describe possible future work.

5.1.1 Evaluating Predictive Performance

An object detection model is not valuable if it cannot accurately detect objects, regardless of how good the uncertainty estimations are. We begin investigating the performance of the baseline. From Table 4.1 we note that the LiDAR and fusion-based methods vastly outperformed the camera-only model. Whilst direct comparison of results on different datasets is not possible, we sanity check our resulting mAPs on ZOD (0.225 for camera, 0.628 for LiDAR, and 0.663 for fusion) with those reported on nuScenes from the original paper (0.356 for camera, 0.647 for LiDAR, and 0.685 for fusion) [21]. Notably, our increase in performance by fine-tuning with the fusion model was on the same magnitude as in the original paper. However, the performance of the camera-only model was considerably lower. We attribute this discrepancy to the fact that nuScenes only considers detections within a range of 51.2m, with certain classes even more limited, while we consider detections up to 128m for all classes, a significantly more challenging task for a camera-only model. Given these considerations, we conclude that the baseline model is sufficiently robust to support the experiments in this thesis.

Comparing the predictive results from the baseline, Table 4.1, with those produced by the models that estimate uncertainty, Table 4.3, we note that the performance is overall similar. However, the models that represented the uncertainty as normally distributed random vectors and used NLL as a loss function performed noticeably worse. On the LiDAR-only model, the mAP was 0.587 when using NLL, compared with 0.628 for the baseline and 0.631 for the energy score model. One potential

reason for this disparity is the tendency for the NLL model to overestimate its uncertainties, something that is highlighted in Figure 4.3. When predicting larger uncertainties, the model assigns less weight to the bounding box error in the loss function. This can be seen in (2.15), where the error is weighted using the inverse of the covariance matrix. We further note that the energy score model outperformed the baseline, highlighting that it is possible to estimate uncertainty without sacrificing predictive performance.

5.1.2 Uncertainty Estimates

When evaluating the performance of uncertainty estimates, two types of metrics were used: proper and non-proper scoring rules. We recall that proper scoring rules are functions that are minimized when the predicted distribution equals the underlying distribution that the ground truth comes from. This makes proper scoring rules crucial in the evaluation of uncertainty estimates. A few key takeaways can be made by comparing the results of the models on the proper scoring rules NLL and PMB-NLL. First, the homoscedastic uncertainty estimates used for the baseline detectors performs poorly, as seen in Table 4.2, in comparison to the models where the uncertainty is modeled, seen in Table 4.4. This means that the uncertainty is data dependent, i.e., that the variance is not constant for each detection. It also means that modeling and estimating uncertainties, using direct modeling, improves the accuracy of uncertainty estimates. Second, we note that the scores are highly correlated to predictive performance. For instance, when evaluating the homoscedastic uncertainty estimates, the estimates of uncertainty were equally poor, however, the fusion model performed considerably better due to its increased predictive performance. This can be further seen by looking at the formulation of, e.g., the proper scoring rule NLL (2.15), where the prediction error is a key part. The fact that the predictive performance plays such a big role is important to consider when evaluating uncertainty estimates, as it says more than just how accurate the predicted covariances are. It also means that proper scoring rules are metrics that capture the performance of all parts of a detector, meaning that you can summarize the performance of the detector to one number, yielding easier comparisons.

For the different losses used when training models to estimate uncertainty, we note a pattern in the results in Table 4.4. Overall, the best results were obtained when using the PMB-NLL as a loss function, highlighting the strengths of this joint modeling. The models trained using NLL on random vectors performed consistently worse, and the energy score models performed somewhere in between. When considering the calibration of the detectors, as seen in Figure 4.3, we note that the energy score models are the best calibrated out-of-the-box, and that the calibration of the NLL is considerably worse. All models are underconfident in their predictions, meaning that the predicted covariances are larger than the expected covariances. Depending on the use-case, being underconfident is often preferred over being overconfident, however, being perfectly calibrated is of course the goal. Steps to recalibrate the predictors can be taken afterwards, e.g., as outlined in [71], however, it is still desirable to obtain well-calibrated models without recalibration.

5.1.3 Utilizing Uncertainties

For utilizing the uncertainties, we first note that the homoscedastic uncertainty estimates used for the baseline detectors say nothing about individual detections, severely limiting their application areas. They could still be used to model the constant measurement noise of the detector, which is a quantity used in downstream tasks such as simultaneous localization and mapping (SLAM) or model-based tracking. The heteroscedastic uncertainty estimates produced by the models are instead specific to each detection, significantly increasing the number of possible applications.

In Section 4.4 we saw how the entropy could be computed from the output uncertainties and used to find instances where the model is unexpectedly uncertain. In the evaluation set we found 18 frames where the main LiDAR was not present, causing significantly higher entropies. While figuring out that the LiDAR data is missing points is in many situations possible using other methods, the fact that the model was able to find these cases without explicitly looking for it speaks to the applicability of the model. In some cases, it might not be as obvious when there is a problem, either with the data being poor or the scene itself being difficult. By utilizing the uncertainties produced by the model, proper actions – e.g., reducing speed, coming to a safe stop, handing over responsibility to a driver etc. – can be taken to ensure safe behavior in these scenarios.

In addition, we saw how the entropies from a model could be used to find out-of-distribution cases, even on non-annotated data, and utilize these for active learning. By training a model where all of the frames containing large vehicles were held out, we observed that, during inference time, the entropies in the frames containing large vehicles were higher than the frames not containing large vehicles. This allows the extraction of frames which are out-of-distribution to be further annotated and trained with, improving the information density and reducing the number of annotated frames needed. This was further reinforced by training a model on half the dataset, computing the entropies of the remaining half, and then retraining by incorporating the high entropy frames compared with random sampling. Here, we saw that the high entropy frames led to increased predictive performance, signifying the applicability of the approach.

5.1.4 Future work

Some work is yet to be done to improve the approach further. One area is further investigating different ways to model the uncertainty. We modeled the uncertainty using multivariate normal distributions with diagonal covariance matricies and showed it was a decent approximation. By using the full covariance matrix, or using a different distribution it could be possible to better model the uncertainties. For instance, [14] used Laplacian distributions to model regression uncertainty in 2D object detection and got improved results compared to using normal distributions.

One important question is how well the uncertainty estimates can be used in downstream tasks. Recently, it was shown that predicting uncertainty in vectorized

map estimation and propagating the uncertainties to trajectory prediction led to increased prediction performance [78]. We hypothesize that the uncertainties from 3D object detection can also be applied to several downstream tasks with good results. However, further experiments are required to test this hypothesis and concretely measure the potential improvements.

Another main improvement would be to compute the entropy using the PMB RFS modeling. Currently, we compute the regression and classification entropies separately, and get entropies per detection. By instead computing the entropy of the PMB RFS, an entropy for the whole frame would be obtained, completely modeling the detection problem and obtaining more relevant results. Since the entropy of the PMB RFS is computationally intractable, some sampling-based approach would likely need to be used.

5.2 Conclusion

In conclusion, modeling the regression uncertainties for multi-modal 3D object detection is beneficial. Modeling regression uncertainties can be done at little extra computational cost and with similar or better performance on traditional 3D object detection metrics. For estimating the uncertainties, three main losses where compared: NLL, ES, and PMB-NLL, and we show that the ES performs best on the traditional metrics, whereas the PMB-NLL performs best on uncertainty-based metrics. Furthermore, we investigate practical applications of the uncertainty estimates. We show how uncertainty estimates can be used to identify cases where the model is extra uncertain, e.g., when the main LiDAR is unexpectedly missing. We show how uncertainty estimates can be used to detect out-of-distribution cases, and we show how uncertainty estimates can be used for active learning.

Bibliography

[1] M. Alibeigi, W. Ljungbergh, A. Tonderski, *et al.*, "Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 20 178–20 188.

[2] T. Stewart, "Overview of motor vehicle traffic crashes in 2021," Tech. Rep., 2023.

[3] D. Metz, "Developing policy for urban autonomous vehicles: Impact on congestion," *Urban Science*, vol. 2, no. 2, p. 33, 2018.

[4] Y. Kumakoshi, H. Hanabusa, and T. Oguchi, "Impacts of shared autonomous vehicles: Tradeoff between parking demand reduction and congestion increase," *Transportation Research Interdisciplinary Perspectives*, vol. 12, p. 100 482, 2021.

[5] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[6] M. H. Hasan and P. Van Hentenryck, "The benefits of autonomous vehicles for community-based trip sharing," *Transportation Research Part C: Emerging Technologies*, vol. 124, p. 102 929, 2021.

[7] R. T. McAllister, Y. Gal, A. Kendall, *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning," International Joint Conferences on Artificial Intelligence, Inc., 2017.

[8] X. Wang, T. Li, S. Sun, and J. M. Corchado, "A survey of recent advances in particle filters and remaining challenges for multitarget tracking," *Sensors*, vol. 17, no. 12, p. 2707, 2017.

[9] W. Xu, J. Pan, J. Wei, and J. M. Dolan, "Motion planning under uncertainty for on-road autonomous driving," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 2507–2512.

[10] C. Stachniss, J. J. Leonard, and S. Thrun, "Simultaneous localization and mapping," *Springer Handbook of Robotics*, pp. 1153–1176, 2016.

[11] B. Settles, "Active learning literature survey," 2009.

[12] D. Feng, A. Harakeh, S. L. Waslander, and K. Dietmayer, "A review and comparative study on probabilistic object detection in autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 9961–9980, 2021.

[13] A. Harakeh and S. L. Waslander, "Estimating and evaluating regression predictive uncertainty in deep object detectors," *arXiv preprint arXiv:2101.05036*, 2021.

[14] G. Hess, C. Petersson, and L. Svensson, "Object detection as probabilistic set prediction," in *European Conference on Computer Vision*, Springer, 2022, pp. 550–566.

[15] A. Harakeh, M. Smart, and S. L. Waslander, "Bayesod: A bayesian approach for uncertainty estimation in deep object detectors," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 87–93.

[16] D. Feng, L. Rosenbaum, and K. Dietmayer, "Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection," in *2018 21st international conference on intelligent transportation systems (ITSC)*, IEEE, 2018, pp. 3266–3273.

[17] M. Pitropov, C. Huang, V. Abdelzad, K. Czarnecki, and S. Waslander, "Lidar-mimo: Efficient uncertainty estimation for lidar-based 3d object detection," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2022, pp. 813–820.

[18] Z. Liu and Z. Han, "Efficient uncertainty estimation for monocular 3d object detection in autonomous driving," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2021, pp. 2711–2718.

[19] D. Feng, Y. Cao, L. Rosenbaum, F. Timm, and K. Dietmayer, "Leveraging uncertainties for deep multi-modal object detection in autonomous driving," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2020, pp. 877–884.

[20] Z. Liu, H. Tang, A. Amini, *et al.*, "Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation," in *2023 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2023, pp. 2774–2781.

[21] *nuScenes detection task leaderboard*, `https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Any`, Accessed: 2023-12-19.

[22] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 11 784–11 793.

[23] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *18th international conference on pattern recognition (ICPR'06)*, IEEE, vol. 3, 2006, pp. 850–855.

[24] H. Caesar, V. Bankiti, A. H. Lang, *et al.*, *nuScenes: A multimodal dataset for autonomous driving*, 2020. arXiv: `1903.11027 [cs.LG]`.

[25] *Volvo cars - ex90 electric*, `https://www.volvocars.com/se/cars/ex90-electric/`, Accessed: May 10, 2024.

[26] *Lotus eletre*, `https://www.lotuscars.com/sv-SE/eletre`, Accessed: May 10, 2024.

[27] *Nio et5 touring*, `https://www.nio.com/sv_SE/et5-touring`, Accessed: May 10, 2024.

[28] S.-L. Yu, T. Westfechtel, R. Hamada, K. Ohno, and S. Tadokoro, "Vehicle detection and localization on bird's eye view elevation images using convo-

lutional neural network," in *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE, 2017, pp. 102–109.

[29] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. Garcia, and A. De La Escalera, "Birdnet: A 3d object detection framework from lidar information," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 3517–3523.

[30] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[31] 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*, 2018, pp. 4490–4499.

[32] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[33] 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*, 2019, pp. 12 697–12 705.

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

[35] X. Bai, Z. Hu, X. Zhu, *et al.*, "Transfusion: Robust lidar-camera fusion for 3d object detection with transformers," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 1090–1099.

[36] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A survey on 3d object detection methods for autonomous driving applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3782–3795, 2019.

[37] S. T. Barnard and M. A. Fischler, "Computational stereo," *ACM Computing Surveys (CSUR)*, vol. 14, no. 4, pp. 553–572, 1982.

[38] D. Eigen, C. Puhrsch, and R. Fergus, *Depth map prediction from a single image using a multi-scale deep network*, 2014. arXiv: `1406.2283 [cs.CV]`.

[39] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 270–279.

[40] B. Xu and Z. Chen, "Multi-level fusion based 3d object detection from monocular images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2345–2353.

[41] X. Weng and K. Kitani, *Monocular 3d object detection with pseudo-lidar point cloud*, 2019. arXiv: `1903.09847 [cs.CV]`.

[42] J. Philion and S. Fidler, *Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d*, 2020. arXiv: `2008.05711 [cs.CV]`.

[43] J. Huang, G. Huang, Z. Zhu, Y. Ye, and D. Du, *Bevdet: High-performance multi-camera 3d object detection in bird-eye-view*, 2022. arXiv: `2112.11790 [cs.CV]`.

[44] Z. Liu, Y. Lin, Y. Cao, *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[45] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde, "Lidar–camera fusion for road detection using fully convolutional neural networks," *Robotics and Autonomous Systems*, vol. 111, pp. 125–131, 2019.

[46] 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*, 2020, pp. 4604–4612.

[47] H. Cai, Z. Zhang, Z. Zhou, Z. Li, W. Ding, and J. Zhao, *Bevfusion4d: Learning lidar-camera fusion under bird's-eye-view via cross-modality guidance and temporal aggregation*, 2023. arXiv: 2303.17099 [cs.CV].

[48] H. Hu, F. Wang, J. Su, *et al.*, *Ea-lss: Edge-aware lift-splat-shot framework for 3d bev object detection*, 2023. arXiv: 2303.17895 [cs.CV].

[49] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*, IEEE, 2012, pp. 3354–3361.

[50] P. Sun, H. Kretzschmar, X. Dotiwalla, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2446–2454.

[51] J. Kannala and S. S. Brandt, "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 8, pp. 1335–1340, 2006.

[52] B. T. Vo, "Random finite sets in multi-object filtering," 2008.

[53] A. Baddeley, I. Bárány, and R. Schneider, "Spatial point processes and their applications," *Stochastic Geometry: Lectures Given at the CIME Summer School Held in Martina Franca, Italy, September 13–18, 2004*, pp. 1–75, 2007.

[54] J. Kingman, "G. matheron, random sets and integral geometry," 1975.

[55] P. Olofsson and M. Andersson, *Probability, statistics, and stochastic processes*. John Wiley & Sons, 2012.

[56] K. Granström, M. Fatemi, and L. Svensson, "Poisson multi-bernoulli mixture conjugate prior for multiple extended target filtering," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 1, pp. 208–225, 2019.

[57] V. C. Ravindra, L. Svensson, L. Hammarstrand, and M. Morelande, "A cardinality preserving multitarget multi-bernoulli rfs tracker," in *2012 15th International Conference on Information Fusion*, IEEE, 2012, pp. 832–839.

[58] Á. F. García-Fernández, J. L. Williams, K. Granström, and L. Svensson, "Poisson multi-bernoulli mixture filter: Direct derivation and implementation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 4, pp. 1883–1901, 2018.

[59] K. Czarnecki and R. Salay, "Towards a framework to manage perceptual uncertainty for safe automated driving," in *Computer Safety, Reliability, and Security: SAFECOMP 2018 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Västerås, Sweden, September 18, 2018, Proceedings 37*, Springer, 2018, pp. 439–445.

[60] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" *Advances in neural information processing systems*, vol. 30, 2017.
[61] Y. Gal *et al.*, "Uncertainty in deep learning," 2016.
[62] S. Depeweg, J.-M. Hernandez-Lobato, F. Doshi-Velez, and S. Udluft, "Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning," in *International conference on machine learning*, PMLR, 2018, pp. 1184–1193.
[63] J. Moberg, "Uncertainty-aware models for deep reinforcement learning," 2019.
[64] J. Yao, W. Pan, S. Ghosh, and F. Doshi-Velez, "Quality of uncertainty quantification for bayesian neural network inference," *arXiv preprint arXiv:1906.09686*, 2019.
[65] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson, "A simple baseline for bayesian uncertainty in deep learning," *Advances in neural information processing systems*, vol. 32, 2019.
[66] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.
[67] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.
[68] T. Gneiting, L. I. Stanberry, E. P. Grimit, L. Held, and N. A. Johnson, "Assessing probabilistic forecasts of multivariate quantities, with an application to ensemble predictions of surface winds," *Test*, vol. 17, pp. 211–235, 2008.
[69] J. Pinto, Y. Xia, L. Svensson, and H. Wymeersch, "An uncertainty-aware performance measure for multi-object tracking," *IEEE Signal Processing Letters*, vol. 28, pp. 1689–1693, 2021.
[70] K. G. Murty, "An algorithm for ranking all the assignments in order of increasing cost," *Operations research*, vol. 16, no. 3, pp. 682–687, 1968.
[71] V. Kuleshov, N. Fenner, and S. Ermon, "Accurate uncertainties for deep learning using calibrated regression," in *International conference on machine learning*, PMLR, 2018, pp. 2796–2804.
[72] A. Paszke, S. Gross, S. Chintala, *et al.*, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.
[73] M. Contributors, *Mmdetection3d: Openmmlab next-generation platform for general 3d object detection*, 2020.
[74] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.
[75] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*, IEEE, 2017, pp. 464–472.
[76] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.
[77] *NVIDIA A100 Datasheet*, `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf`, Accessed: June 3, 2024.

[78] X. Gu, G. Song, I. Gilitschenski, M. Pavone, and B. Ivanovic, *Producing and leveraging online map uncertainty in trajectory prediction*, 2024. arXiv: 2403.16439 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2403.16439.

A

ZOD Details

A.1 License

The Zenseact Open Dataset [1] is the property of Zenseact AB (© 2022 Zenseact AB) is licensed under CC BY-SA 4.0[1] with the following notice:

> For this dataset, Zenseact AB has taken all reasonable measures to remove all personally identifiable information, including faces and license plates. To the extent that you like to request removal of specific images from the dataset, please contact privacy@zenseact.com.

A.2 Sensor Descriptions

A more detailed description of the relevant sensors used from the Zenseact Open Dataset are detailed in Table A.1.

Sensors	Description
Camera	One 120° Horizontal FOV, 67° Vertical FOV, 3848x2168 RGB camera facing forwards. Camera has a wide-angle fish-eye lens and captures images as 10Hz.
Main LiDAR	One 360° Horizontal FOV, 40° Vertical FOV Velodyne VLS128 with a vertical angular resolution down to 0.11 degrees (non-linear) and a horizontal angular resolution of 0.2 degrees. LiDAR is mounted on the roof of the vehicle and captures point clouds at ∼9Hz.
Side LiDARs	Two 360° Horizontal FOV, 30° Vertical FOV Velodyne VLP16s with a vertical angular resolution of 2 degrees and a horizontal angular resolution of 0.2 degrees. The LiDARs are mounted on the left and right side of the roof, rotated outwards in roll, and capture point clouds at ∼9Hz.

Table A.1: Relevant sensors used in the Zenseact Open Dataset.

[1] https://creativecommons.org/licenses/by-sa/4.0/

B

Additional Probability Distributions

Some additional basic probability distributions are mentioned but not defined in thesis. The additional distributions include the categorical distribution, the Bernoulli distribution and the Poisson distribution. This chapter provides short introductions to these and more information is available in for instance [55].

B.1 Categorical Distribution

The categorical distribution is a discrete probability distribution often used to model class probabilities in machine learning models. It models the possible outcomes of a random variable where each outcome belongs to one of K different, mutually exclusive categories. Each category i has a specified probability $p_i \in [0, 1]$, and the sum of probabilities over all categories is 1. The probability mass function is

$$f(k|\mathbf{p}) = p_k, \tag{B.1}$$

where k is the category for which to compute the probability, and $\mathbf{p} = (p_1, \ldots, p_K)$ are the probabilities for each category.

B.2 Bernoulli Distribution

The Bernoulli distribution is a discrete probability distribution that models the outcome of a single event with two mutually exclusive states. Without loss of generality, the two states can be denoted $k = 0$ and $k = 1$. The probability mass function is

$$f(k|p) = p^k(1-p)^{1-k}, \tag{B.2}$$

where p is the probability of the state $k = 1$ occurring.

B.3 Poisson Distribution

The Poisson distribution is a discrete probability distribution that models the number of times an event occurs within a fixed interval, such as time, area, or volume.

Additionally, the occurrences of the event must be independent from each other. The probability mass function of the Poisson distribution is given by

$$f(k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{B.3}$$

where λ is the mean number of occurrences of the event in the given interval and k is the number of occurrences for which to compute the probability.

C

Qualitative Results

C.1 BEV Visualization

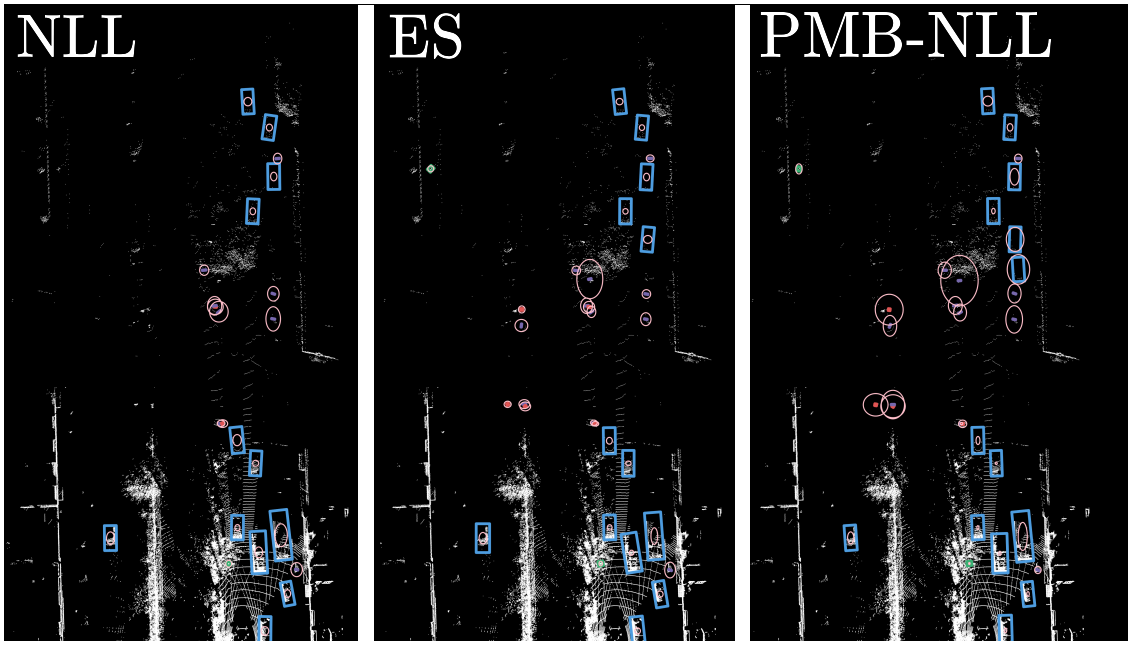


Figure C.1: Qualitative comparison of a LiDAR model using different losses visualized in BEV on an example from ZOD[1][1]. Only detections with existence probability greater than 0.5 are shown. Blue bounding boxes represent predictions of class `Vehicle`, yellow the prediction of `VulnerableVehicle`, green the prediction of `Pedestrian`, purple the prediction of `TrafficSign`, and red the preciction of `TrafficSign`. The 95% confidence of the center point of each object is visualized using pink ellipses. The model trained using NLL (left) is less confident, whereas the energy score model (middle) and PMB-NLL model (right) perform predictions with higher confidences. The corresponding camera view can be found in Figure 4.2.

[1]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details.

C.2 High Entropy Examples

Figure C.2: A selection of high entropy frames on ZOD[2][1]. The top image shows an image with many occluded pedestrians at night. The middle image shows an image with a weird blur. The bottom image shows a LiDAR point cloud where the data from the main Velodyne VLS128 LiDAR is missing.

[2]Licensed under CC BY-SA 4.0, see Appendix A.1 for license details.