Bachelor Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Ensemble Detection Models for LiDAR Point Clouds

Šimon Pokorný

Supervisor: Ing. Patrik Vacek
May 2021

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Pokorný Šimon**	Personal ID number:	**487004**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Ensemble Detection Models for LiDAR Point Clouds

Bachelor’s thesis title in Czech:

Sdružené detekční modely pro LiDARové mraky bodů

Guidelines:

This thesis will focus on neural networks and their usefulness for object detection in LiDAR point clouds. The main idea is to improve detection performance by adding other detection models and reasonably combine their outputs for more consistent and better detections. We will evaluate on publicly available benchmarks for LiDAR point clouds in autonomous driving scenes.
1) Survey publicly accessible neural network models for detection or segmentation of the point clouds.
2) Re-train selected neural networks on lidar data from the autonomous driving environment.
3) Create a method to combine outputs from all models for overall better predictions.
4) Design pipeline for learning from new predictions on unlabelled data from the proposed method from 3).
5) Discuss and compare the performance of the applied methods with using one model only.

Bibliography / sources:

[1] Joel Janai; Fatma Güney; Aseem Behl; Andreas Geiger, Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art, now, 2020
[2] Z. Wang, W. Zhan and M. Tomizuka, 'Fusing Bird’s Eye View LIDAR Point Cloud and Front View Camera Image for 3D Object Detection,' 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, 2018, pp. 1-6, DOI: 10.1109/IVS.2018.8500387
[3] R. Q. Charles, H. Su, M. Kaichun and L. J. Guibas, 'PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,' 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 77-85, doi: 10.1109/CVPR.2017.16.

Name and workplace of bachelor’s thesis supervisor:

Ing. Patrik Vacek, Vision for Robotics and Autonomous Systems, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **08.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Patrik Vacek	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my thesis advisor Ing. Patrik Vacek for his valuable guidance, patience, motivation, continuous support and also to prof. Ing. Tomáš Svoboda, Ph.D., both of them provided me with excellent expertise and consultations.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 17, 2021

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 17. května 2021

Abstract

We address the problem of the ensemble neural networks in the LiDAR pointclouds. Neural networks are sometimes saturated for specific situations meaning they perform worse on some scenarios due to capacity issues. The prediction results in the LiDAR point cloud domain are deteriorating with increasing distance due to the low density of the measurement in the remote areas. We are using a multi-view ensemble, which consists of detection models operating on a separate view and merging the transformation into one data representation. We are using frontview projection, which is the transformation of the canonical coordinates of the LiDAR point cloud to the spherical coordinates. The second view is the projection of scan points to xy plane called Bird's Eye View (BEV). In the both projections we merge models focusing on specific areas or distance range. We further exploit semi-supervised learning approach called *pseudo-labelling* in order to generate labels from the ensemble for baseline improvement. All methods are evaluated on semantic segmentation tasks in autonomous driving scenarios and achieve improvement in terms of IoU against the baseline architecture.

Keywords: neural networks, ensemble methods, LiDAR point clouds

Supervisor: Ing. Patrik Vacek
ČVUT v Praze
Fakult elektrotechnická
Katedra kybernetiky
Karlovo náměstí 13
121 35 Praha 2

Abstrakt

Řešíme problém z oblasti počítačového vidění, kde se zaměřujeme na sdružené detekční metody neuronových sítí pro LiDARové mraky bodů. Neuronové sítě jsou často naučené na konkrétní situace, což v některých případech výrazně zhoršuje jejich predikční schopnost. Úspěšnost predikce na LiDARových mračnech bodů se zhoršuje s narůstající vzdáleností naskenovaných bodů. Sdružujeme modely z více pohledů, kdy jednotlivé modely pracují v různých projekcích a spojení probíhá v jedná datové representaci. První pohled se nazývá *frontview*, kdy se jedná o transformaci kanonických souřadnic mračna LiDARových bodů do sférických souřadnic. Druhým pohledem je projekce naskenovaných bodů do roviny xy s názvem *Bird's Eye View* (BEV), neboli "ptačí pohled". V obou doménách pracují modely zaměřené na konkrétní oblast či rozsah vzdáleností. Dále využíváme *semi-supervised* učící techniku nazývanou *pseudo-labelling*, abychom vygenerovali *label* pro neanotovanou část datasetu pomocí celého sdruženého modelu a tím dosáhli lepších výsledků v průměrném Jaccardově indexu oproti *baseline* architektuře.

Klíčová slova: neurální sítě, sdružené metody, LiDARové mraky bodů

Překlad názvu: Sdružené detekční modely pro LiDARové mraky bodů

Contents

Figures

Tables

Chapter 1
Introduction

The development of autonomous vehicles is on the rise. For a safe autonomous driving in a city environment which is highly complex and dynamic, it is suitable to use LiDAR sensors, which tell us the correct distance of the individual points regardless of the weather or other external influences. Thus, LiDAR sensors are likely to become an integral part of the hardware equipment used on autonomous driven cars. In this Bachelor thesis, we are dealing with the object detection LiDAR pointcloud. Due to the time complexity of object detection neural networks, we focused on the semantic segmentation task on LiDAR pointcloud, which is a fundamental problem in computer vision. If the proposed methods work well in semantic segmentation tasks, they would might benefit even in object detection. The goal is to assign a label to each pixel or scan point. The semantic segmentation task is mostly used on RGB images, where it is very difficult due to the complexity of the scene, the complexity of object boundaries, and the occurrence of very small objects. The problem is much complicated in LiDAR pointcloud domain, where scan points have low density in remote areas, they are unordered, and it is very hard to set the boundaries of the objects, because the data are very sparse unlike RGB pictures, where pixels are in a more compact form. The bachelor thesis is focused on a multiview ensemble method of models and improvement of invidual neural networks for semantic segmentation in LiDAR pointclouds. The main idea is to improve the detection performance by adding other detection models and reasonably combine their outputs for more consistent and better detection. [13]

Semantic segmentation algorithms which consume directly pointcloud and work with them have not yet achieved the same results as well-established 2D CNN on RGB images. Thus, we transformed the original LiDAR pointclouds to a 2D space, where we are using two projections. On each of them, the individual models will be learned. First projection transforms the pointcloud to spherical coordinates, where two angles defining the location are discretize and put to a 2D grid. This projection is called Frontview insomuch as it is very similar to seeing a set of points with our own eyes. Second projection is called Bird's Eye View, which projects a set of scan points on the xy plane.

Detection models are sometimes saturated for specific situations, meaning they perform worse in some scenarios due to capacity issues. Part of the work is focusing on using multiple detectors altogether for detection, where we use models prediction on some specific distance ranges or areas and combine outputs from different projections. We force individual models to focus on different areas by modifying the loss. The final ensemble pipeline, hereinafter referred to as FEP, works with 6 models operating in Frontview projection and 5 models working in Bird's Eye View projection. FEP benefits from all models. The final output is produced in the Frontview projection, but it can be easily transformed to the pointcloud or Bird's Eye View projection.

The final part of the work deals with an effort to improve the performance of invidual models by a semi-supervised learning technique called pseudo-labelling. The FEP produce pseudo labels of the unanotated part of dataset, which is cheap and easy to get during data acquisition. These frames are added to the original training dataset. We used two approaches to boost the predictive ability of the model. We train the invidual models once again from scratch and we are apply a fine-tunning training phase to the trained model in order to increase the performance further with pseudo-labels.

Chapter 2
Approach

Majority of the progress of neural networks is focusing on deeper and deeper networks since 2012, when the AlexNet [14] was proposed. The most fundamental among the deep neural networks is ResNet [11], which is created from residual units. The research claims that individual residual units are not strongly depending on each others and the ResNet actually behaves as an exponential ensemble of relatively shallow networks, which are put in a row to make deep network [29]. Results on the ImageNet classification dataset [9] have shown that almost duplication of the number of layers, where we compare 152 and 269 layers ResNet, improves the mean IOU by 1.1%. Problem of these very deep neural networks is the vanishing gradient, where the problem appears even with the network uses shortcuts to propagate the gradient further. The wide ensemble of many residual units surpasses the results of deeper variants of ResNet network. This architecture achieves state-of-the-art on datasets including PASCAL VOC [10], and Cityscapes [8]. [29]

The reliable scene understanding will be an indispensable part of autonomous driving. The urban scenes are very complex and dynamics and that requires reliable predictions to avoid accidents. Therefore, object detection approaches have been proposed in variety modalities. The most used sensor is a video camera, where the visible spectrum is used for daytime prediction, whereas the infrared spectrum is used for night prediction. Relying on one single detector would be problematic with the view that every sensor have some error rate, different sensing due to weather changes, time of the day, or material of the object. For real use of autonomous driving will might be needed the multiple inputs ensemble neural network. To reach a maximal generalization of the neural networks and alleviate overfitting , there are commonly used techniques such as data generation, data augmentation, regularization, and ensemble methods of models. In this Bachelor thesis , we are focusing on the ensemble only from one input sensor, but the ensemble of multiple models in LiDAR point cloud domain all together to get the most out of one sensor for further merging with results from other sensors. [13]

Convolutional neural networks [CNN] are nonlinear operations, which brings us high flexibility in mapping very complex data. Thanks to this, CNN can be optimized to a different set of weights during the training, which produce a slightly different mapping of output to the same input. The ensemble methods suppress this side effect and bring us a pipeline with better predictive ability and generalization. [5]

2.1 Existing ensemble methods

2.1.1 Model Averaging

Frequently used approach to remove undesirable effects such as overfitting and increased prediction success is to apply averaging. This simple method trains n same models on the same dataset and during the prediction takes the average of the output of all these models.

Every model is initialised with slightly different parameters, thus all trained models are guaranteed to converge to different solutions, even though it sees the same training data. [19]

The functionality of this method is also supported by the results of a popular neural network called ImageNet, which achieved a top-5 error rate of 18.2% in "ILSVRC-2012" competition. Authors of the neural network tried to use an ensemble method called "averaging" of 5 similar CNNs. This ensemble model decreased top-5 error rate to 16.4%. [15]

2.1.2 Bootstrap Aggregating

It is a parallel method also known as "Bagging". We can separate this method in two steps, bootstraping and aggregating. The key idea behind bootstraping is to divide a big dataset into n subdatasets or if we are working with a small dataset to create subdatasets with samples, which are randomly chosen. These subdatasets are called bootstraps. Each bootstrap sample should have sufficient distribution of all classes and create independence between them thanks to a small overlap of frames in invidual bootstraps so that the samples are not too much correlated. Then use a process called aggregating, which creates n models with the same architecture and learns each of them with an assigned bootstrap. Outputs of all models are then averaged. Final model should be more robust, have better predictive ability and reduced variance. This method is often used in decision tree algorithms, where multiple trees are composed to a grouping called "random forest". [23]

2.1.3 Stacking

In this method, we can combine n models based on different or same architecture, called weak learners. Dataset is split into two parts. On the first subdataset are trained these weak learners. When the training is done, the models are evaluated on the second subdataset to create outputs from all frames. Weak-learners are combined by meta-model, which consumes outputs from them. The meta-model algorithm is trained on the second subdateset. [23]

2.1.4 Multiple inputs

Very often used approach is to work with multiple inputs from different sensors, which are ensembled by neural networks. This ensemble technique penetrated into the semantic segmentation task. One proposed way how to use this technique in the neural network based on U-net is to propagate each input in the encoder individually and the merging of featured maps takes place in the decoder. Authors use as inputs RGB images and infrared sensor for the semantic segmentation task. [12]

The next possible application of the multiple input ensemble method is to combine LiDAR pointcloud projected into Bird's Eye View and Frontview camera RGB images. Authors combine 2D RGB images captured by the camera and the LiDAR pointcloud projected into BEV grid. They proposed a sparse non-homogeneous pooling method to fuse Frontview RGB images and BEV pointcloud. The results on KITTI on the hardest predictive class pedestrians achieved the state-of-the-art. [30]

2.2 Proposed ensemble methods

2.2.1 Range Level Ensemble

Models does not have to yield enough parameters and structure for all types of detections. We can split it for different ranges and calculate loss only on specific areas. That can lead to better performance, since the parameters will be focused on specific features of lidar scan points as a function of range.

For multiclass classification is mostly used Cross Entropy loss (CE). We have the unbalanced dataset, thus we alleviate this problem with weight vector $\alpha \in \langle 0, 1\rangle$, which is an extension of cross entropy loss.

$$\mathrm{CE}(x, \alpha) = -\alpha_i \cdot \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right) \tag{2.1}$$

The equation 2.1 describe the computation of the loss, where x is pixel, where is the loss computed. Index i is actual class and index j denotes set of all classes.

We are further expanding the loss function to Focal Loss (FL) in the equation 2.2, which should support efforts to correct imbalances in the dataset. To the CE is add modulating factor $(1 - pt)^\gamma$, with tunable focusing parameter $\gamma \geq 0$. This factor should reduce the loss in pixels, where the neural network is confident and focus optimization on pixels, where is less certainty of determining the correct class. Variable pt in the equation 2.2 denotes probability. [18]

$$\mathrm{FL}(x, \alpha, \gamma) = (1 - pt)^\gamma \cdot \mathrm{CE} \tag{2.2}$$

We regularize the cross entropy loss and focal loss with a range function, which define at what distance will the neural network calculate the loss and fit to this distance range.

$$\text{range-CE}(d) = \text{range}(d) \cdot \text{CE} \tag{2.3}$$

$$\text{range-FL}(d, \gamma) = \text{range}(d) \cdot (1 - pt)^{\gamma} \cdot \text{CE} \tag{2.4}$$

The range function is masking points with weights based on the distance of the scan point. It will be examined more in the experiments section. In equations 2.3 and 2.4, d denotes distance, where the scan point was scanned.

2.2.2 Multiview ensemble method

Multiview ensemble method is not used in semantic segmentation, because most of these tasks are working with only RGB images, which have only one view to the scene.

We work with two projections, where each of them has some advantages over the other projection. We proposed *fusing model*, which is a neural network consuming outputs from both projections. It is trying to merge the outputs from these projections to get statistically the best out of each.

The *fusing model* should not have to yield enough parameters and structure, to prevent overfitting. Therefore, we tried several types of the model. These models are based on two types of methods.

We are proposing two methods how to combine outputs from both projection.

1. **Learnable scalar** It is a intuitive approach, where one of two projection is multiplied by scalar, that *fusing model* would begin to give priority to one of the projection. We can apply this method to each class from one projection, which would perform better than one number if each of the projection have better predictive ability in different classes.

2. **Smothing** This method is based on 2D convolutinal layers. Kernels should smooth the output, because it happens a lot that some object is classified as an object of one class, but in the middle of the object is a patch which is misclassified. Very often this occurs on buses, where part of the bus is classified as background.

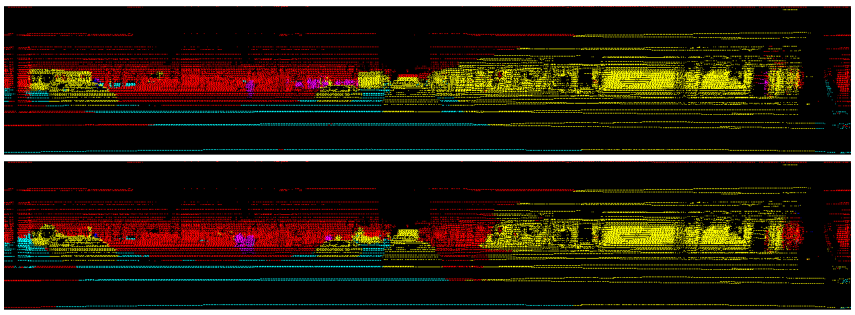

Figure 2.1: Example of misclassified patches on the cars, a upper fig is ground truth and bottom fig is prediction of the baseline in the frontview projection

As we can see in fig 2.1 second car from the left and the farthest car are only partly classified as a vehicle. The bus from the right of the fig 2.1 is predicted as a vehicle only of two thirds.

The smoothing approach would be able to smooth objects to more compact form and improve the prediction ability. It is important to have a model, which has not too many parameters and has a large enough receptive field, because the valid pixels are quite remote and misclassified patches can range from a few pixels to a big size.

2.3 Recurrent pipeline

Datasets have played a key role in the progress of many research fields by providing problem-specific examples with ground truth. For more complex tasks and better results are created datasets with hundreds or thousands frames, which helps the algorithm to learn the specific task and generalize the problem. Despite the great developments in the creation of datasets for computer vision tasks, the number and difference of frames is not sufficient very often. Moreover, the creating of the LiDAR pointcloud dataset is a very lengthy and expensive work, because all annotation is done manually with the help of advanced software which uses interpolation techniques to speed up the process. [13]

We use the proposed FEP to produce pseudo labels for non-annotated frames. These pseudolabels with the original dataset are used in the next training phase for semisupervised learning to improve the generalization of the invidual models. Thanks to the extension of the dataset with pseudo labels, we can boost each model of the pipeline to upgrade the prediction ability of the FEP.

Chapter 3
Implementation

3.1 Dataset

We used Argoverse dataset created by the company “Argo AI”, which headquarter is located in Pittsburgh , Pennsylvania [6]. Dataset is made up of 18137 pointclouds and each pointcloud has averaged captured around 107 000 points at 10 Hz. All data were captured in the city environment, specifically in Pittsburgh (86 km) and Miami (204 km).

Data were collected by a fleet of cars, which were equipped with two lidar sensors, seven ring-cameras and two-facing stereo cameras. Each lidar has 32 lines and they were placed on top of each other. With this placement was created an overlapping 40° vertical field of view. The maximal range of detectable rays is 200m.

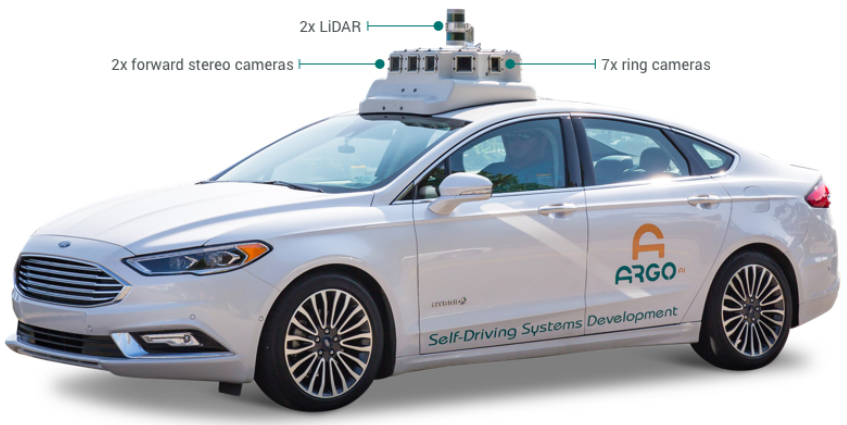

Figure 3.1: Ford Fusion Hybrid used for capturing data [6]

We work with 8245 labeled and 9892 unlabeled frames. Individual frames were captured in sequence, where the frames are very similar. On average in every sequence were 203 frames. We split the individual labeled frames in 3 groups, more precisely on training, validation and testing (5701, 1306 and 1238 frames). Frames from one sequence were put only in one group, in order to prevent subsequent depreciation of results. Unlabeled data were put to the unlabeled training dataset, which will be further used for semi-supervised learning.

number	name	color
0	buildings, sidewalk, trees ...	red
1	vehicles	yellow
2	pedestrians	purple
3	motorcycles, scooter, wheelchair users, animals, cyclists	blue
4	other moving objects	gray
5	roads	cyan
6	unreflected ray (added class)	black

Table 3.1: Overview table of individual classes

The dataset contains 6 classes, but classes 2 and 3 were put together based on similarities. Trained models were unable to recognize the difference between these two classes. Moreover, scan points density is very low on pedestrians and cyclists, so without knowledge of other outputs from sensors such as cameras, it is hard to decide about the classification.

Seventh class is artificially added to fill free space in 2D grid. This class appears only in 2D projections.

3.2 Data Representations

Argoverse dataset contains 8245 labeled point cloud frames from 40 different scenes and 9892 unlabeled point cloud from 49 scenes, which were used for semi-supervised learning. The LiDAR pointclouds are normally sparse, have variable density, and in addition non-uniform sampling of 3D space. When we want to deal with object classification or semantic segmentation task using neural networks on point clouds, we have to deal with these challenges. There are three domains how to represent point cloud and let the neural network learn on these representations.

3.2.1 Existing representations

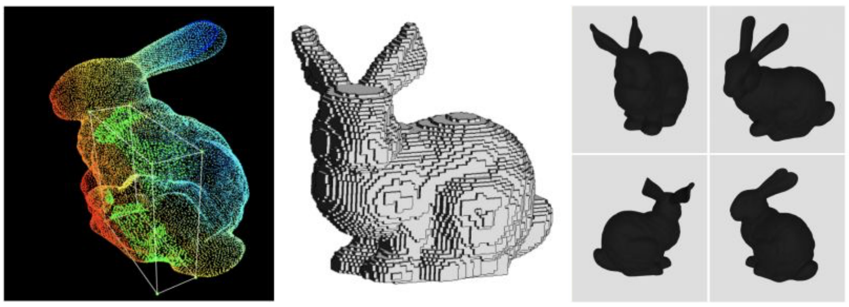

Figure 3.2: Example of representations, from the left: Point Cloud, Voxel grid, 2D representations. Source: Adapted from [20]

2D representation

We can project 3D space into 2D grid from different directions or use different transformations. Thus, we can use well-established 2D CNNs or pre-trained networks [17] as VGG [26], AlexNet [14], GoogleNet [28] or U-net [24] for semantic segmentation or object classification.

Projected point cloud to 2D grid has a disadvantage that lots of pixels in 2D grid are not occupied and we must add an extra class, which corresponds to an empty pixel. Moreover, size of the grid is hard to set on the optimal size, which would minimize the omission of the pixels and the grid has not big memory demand.

Geometric 3D representations of scenes or objects have far more information and details than just 2D projections of it. We are losing many points by projecting of point clouds to 2D grid, because one pixel in the grid very often corresponds to more than one pixel in the 3D space.

The well-developed 2D CNN architectures can better exploit the local and global information from projected 2D view scenes [27] and thanks to the reduction of one dimension, it reduces the computation cost and increases the speed of evaluating. However, most of today's classifiers are focused on robust 2D CNN [13].

Voxel grid

Normally, CNNs are used on data, which have some regular structure of grids. Thus, to apply CNN directly to an unordered point cloud we need to divide 3D space to a regular grid. Then we can smoothly use 3D convolutional layers and pooling layers. Coordinates of point cloud $X \in x_1, x_2, x_3, ..., x_n$ where $x_i \in \mathbb{R}^3$ needs to be discretized by process called Voxelization. It is a process, which discretizes 3D space of point cloud into 3D grid. Typically the size of the grid is related to the resolution of the data. Result of this process are voxels with coordinates $\hat{X} \in \hat{x}_1, \hat{x}_2, \hat{x}_3, ..., \hat{x}_n$, where $\hat{x}_i \in \mathbb{N}^3$. In 3D, we can visualise the voxel as a cube with the size of the length of sampling rate. Very often in the surroundings of the origin of the coordinate system, assuming that the origin is in the center of the lidar, each voxel corresponds to more than one point, because the grid has the same size in all 3D space, but the concentration of points in the surroundings is usually much higher.

Voxel based models.

- **VoxNet** is proposed by Maturana. It is an object classifier, which uses convolution layers, pooling layers and fully-connected layer and has less than 1 million parameters. The model uses three different occupancy grids, which are examined. The best one is chosen. The grids are binary occupancy grid, density grid, and hit grid. [32]

PointCloud

Learning directly on the LiDAR pointcloud is the latest approach. The LiDAR creates an unordered group of points. Thus, the neural network with the same group of N points, i.e. can be fed by N! permutations of points, which corresponds to the same frame. The neural network must be invariant to these permutations. A rigid transformation, such as rotation or translation can be applied to the LiDAR pointcloud. These transformations should not affect the performance of prediction. [7]

Voxel based models scan the local space by receptive fields, which have a fixed and constrained stride. On the other hand, the models consuming the point cloud decide the range of receptive fields, which have the biggest accuracy. [17]

PointCloud based models.

- **PointNet** is the first introduced algorithm, which consumes directly all points. This model was primarily developed for a classification task, but it can be used in the semantics segmentation task with modified architecture. PointNet takes only spatial coordinates. It can learn spatial features thanks to multilayer perceptron and then accumulating features by maxpooling layer. The learned spacial features are then assembled across all regions of the point cloud. Nevertheless, local points are grouped and then max-pooling layer is applied. As a result, the algorithm loses information about local structures. Therefore, PointNet is not robust to complex objects or scenes. [7]

- **PointNet++** is update of the mentioned PointNet. This update can consume spatial coordinates but also non-spatial features such as intensity, RGB, etc. Consuming of only spatial coordinates limits the ability to recognize different surfaces of objects and performance on complex scenes. [22]

  Nevertheless, the semantic segmentation performance on Semantic Kitti [4], which contains similarly urban scenes as Argoverse dataset is not the best. Pointnet achieved mIOU 14.6 and Pointnet++ achieved mIOU 20.1. [1]

We tried to use PointNet++ on the Argoverse dataset in the nonmodified architecture of the introduced model.

The model consumes all scan points, but the occurrence of invidual classes is unbalanced. Therefore, we used multiclass cross-entropy as the loss function with weight vector, which should deal with the problem.

class	0	1	2	3	4	5
Occurence [%]	82.22	10.92	0.31	0.01	0.06	6.47

Table 3.2: Occurrence of classes in the LiDAR pointcloud

Original point cloud is in [Features, Number of scan points], where the number of features is 4. Neural network will get information about coordinates x,y,z and intensity of reflected ray. We can see visualisation in fig 3.3.

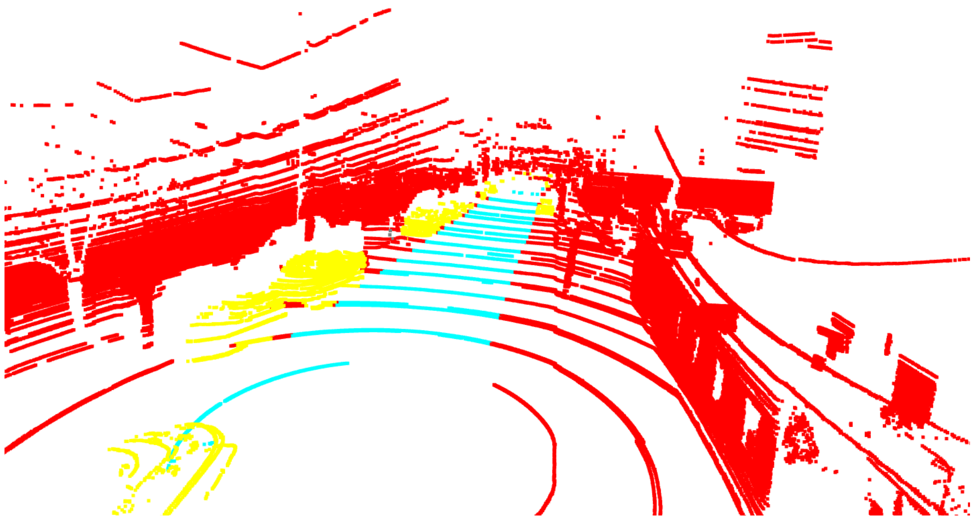

Figure 3.3: Visualisation of point cloud

The training was performed with batch size 6 and with the 150 epoch.

Results of Pointnet++			
Class	Precision	Recall	IOU
0 - background	**0.845**	**0.906**	**0.777**
1 - vehicles	0.815	0.706	0.609
2 - pedestrians	0.035	0.006	0.005
3 - cyclists	0.000	0.000	0.000
4 - moving objects	0.053	0.009	0.008
5 - road	0.598	0.733	0.491
Mean IOU: 0.315 Precision: 0.391 Recall: 0.393 use of points: 100 %			

Table 3.3: Results of PointNet++ on Argoverse dataset

Results have shown us that the model achieved better IOU than on Semantic KITTI, but the classes in Semantic KITTI are much more diverse. As we can see in table 3.3 the model is unable to learn small objects such as classes two, three, and four, which represent pedestrians, cyclists, and moving objects. Mean IOU on these classes is very close to zero.

We can see on the prediction of one frame in fig 3.4, which tells us that the model is able to learn the structure of the scenes such as location of the road, cars, buildings, etc.

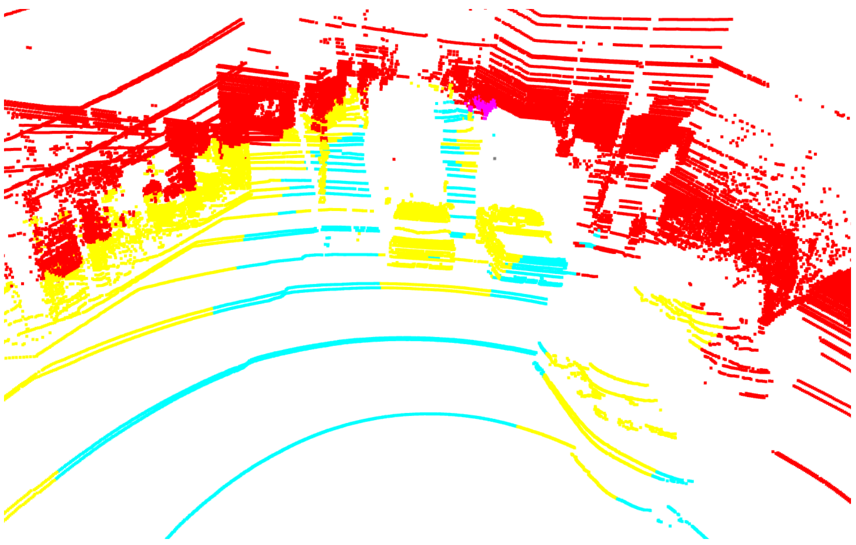

Figure 3.4: Prediction of PointNet++ on Argoverse dataset

Due to bad results on the Argoverse dataset, we decided to not use the PointNet++ in the ensemble methods.

3.2.2 Frontview

Frontview is an imitation of how a lidar captures a scene and it is closest to how we see the world with our eyes, with the difference that here the scene is not limited. Therefore, the scene is captured in a 360 degree view.

Original data of point cloud have information about coordinates $x,y,z \in \mathbb{R}$ and non-geometrical feature, which is *intensity* $\in \mathbb{R}^+$. The intensity value varies with the composition of the surface object reflecting the LiDAR ray, the angle of impact, surface roughness, and moisture content. A higher number corresponds to a higher reflectivity of the surface. [2]

The number of scan points is variable, but on average has around 107 000 points. First, we shifted the global origin to the center of the LiDAR, more precisely to the center of the two lidars. We have to do this operation, so that we can transform the canonical coordinate system to spherical without scene distortion. LiDAR's scanning lines form the shell of the cone using scanning rays. We transformed the canonical coordinates to the spherical representation using transformation relations from equations 3.1, 3.2, 3.3, where variables x,y,z denote canonical coordinates, φ is a elevation angle and θ denote azimuth in the spherical representation. Transformation can be seen in fig 3.5.

$$r = \sqrt{x^2 + y^2 + z^2} \tag{3.1}$$

$$\varphi = \arctan2(y, x) \tag{3.2}$$

$$\theta = \arccos\left(\frac{z}{r}\right) \tag{3.3}$$

Now we have coordinates $r \in (0; 150m\rangle, \theta \in (0; \pi\rangle, \varphi \in (0; 2\pi\rangle$. We set the threshold in distance (coordinate r) to 150m, because the farthest captured points are isolated and do not form any recognizable structure, therefore have not any valuable features for CNN.

We created 2D grid 2 x 244 x 2496 pixels, hereinafter referred to as grid [Features, Height, Width] or [N, H, W]. We discretized coordinates φ with $\varphi_{stepsize}$, where φ_{max} and φ_{min} are limits of the elevation angle computed in one frame. The range of φ is different in every frame due to the calibration of the data and the arrangement of the scene, therefore we set a condition that difference between φ_{max} and φ_{min} must be bigger then 50°. The condition prevents big vertical stretching of the frames, which were captured, for example, on a highway, where the upper rays did not return. If the difference of limits of the variable φ is smaller than 50°, we set φ_{max} as φ_{min} + 50°. All dataset was captured in the city, therefore the condition was never met.

Variables φ is in $\langle\varphi_{min}; \varphi_{max}\rangle$. In other words, pixels with coordinates from range $(0; \varphi_{stepsize})$ correspond to the same index in 2D grid. Thanks to this fact, in some cases there exists a possibility that more than one scan point corresponds to the same pixel in 2D grid. When this inconvenience occurs, we selected the nearest scan point of this subset.

$$\theta_{stepsize} = \frac{2 \cdot \pi}{2496} = 0.144° \tag{3.4}$$

$$\varphi_{stepsize} = \frac{\pi}{(\varphi_{max} - \varphi_{min})} \tag{3.5}$$

We also discretized θ with $\theta_{stepsize}$. Further, the discretized coordinates θ and φ will be marked as x,$y \in \mathbb{N}$. Coordinate r has been added to features.

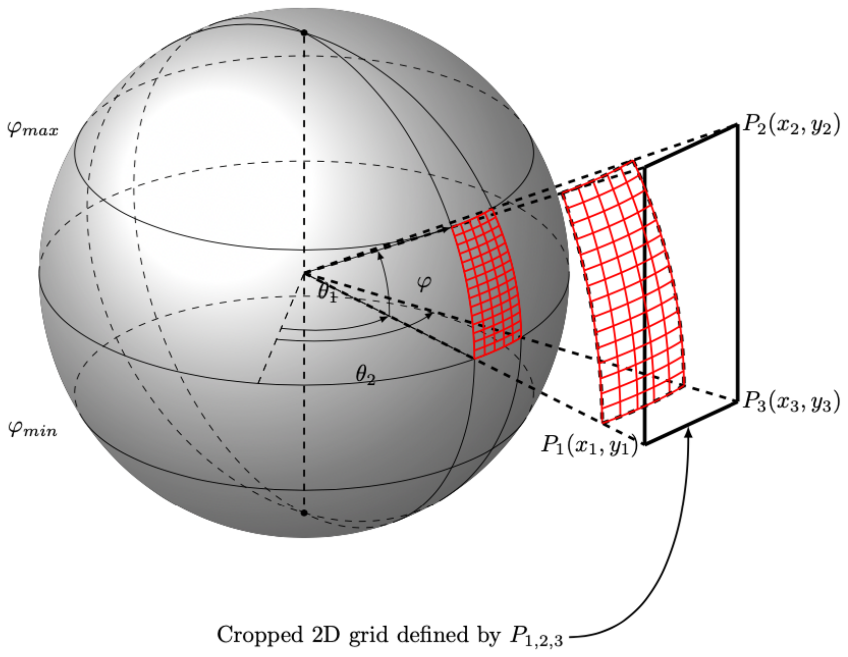

Figure 3.5: Example of transformation from spherical coordinates to cropped 5x15 2D grid (red)

Now we locate each pixel based on coordinates x,y and have 3 features (distance, intensity, T/F for scan points). Third feature is artificially created and tells if any scan point captured by LiDAR is corresponding to the 2D grid pixel, we put 1. In the other case that for the 2D grid pixel there is no LiDAR rays, which scan any object, we put in the pixel a value 0. After that, we normalize the data in features so that the distance is scaled in range $\langle -1; 1 \rangle$ (the furthest scan points have value 1) and intensity is also scaled in range$\langle -1; 1 \rangle$. During the transformation we lost a few points thanks to duplicity in pixels. We achieved the averaged use of original points 87.7%.

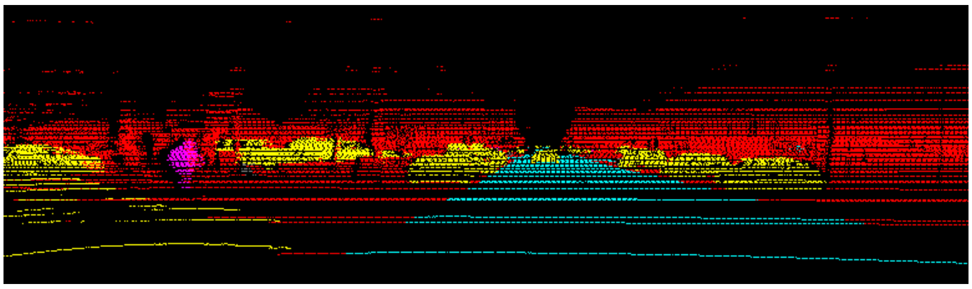

Figure 3.6: Visualisation of cropped 2D grid in Front View 775x224 pixels, that means 30° field of view

3.2.3 Bird's Eye View (BEV)

Another 2D map representation of the LiDAR pointcloud is BEV, which projects the set of scan points to xy plane. The most important information of the scene is the location, which can be mapped to 2D grid in BEV and height of objects to the feature space. The projection can smoothly help neural network to understand the scene, because contain well-encoded spatial locations, along with the overall context of the scene. The neural network can easily predict even small objects, which create a small subset of points with similar intensity and they are surrounded by class 6 (unreflected rays). The neural network has better results in BEV representation then Frontview, but BEV has a bigger memory demand to capture the same area as Frontview. Thanks to capturing of the scene in a representative manner, it is used for many tasks. [21]

The projection projects 3D space to xy plane and add coordinate z as another feature. LiDAR is capable of scanning points in the distance up to 200 m. The number of points more distant than 100m is very low. To achieve a quite good concentration of points in the grid, we decided to use an area about the size 20 x 40 m. This area creates a rectangular, where the longer side is parallel to side of the car. In other words, we are more focusing on the events in front of and behind the car that scans a scene using a lidar (hereinafter referred to as the *lidar car*).

We created 2D grid, which size is 512x1024 pixels. We also discretize cropped xy plane with *sampling rate*.

$$sampling\ rate = \frac{34}{512} = \frac{68}{1024} = 0.0066 \left[\frac{m}{pixel}\right] \tag{3.6}$$

In some cases there exists more than one scan point which corresponds to the same pixel, therefore we applied a decision logic that takes pixels in the same level (coordinate z) as the lidar car. If there are no scan points in the subset which correspond to the pixel in this range, we choose the closest pixel to the interval where the car is located. This projection can use on average 20.9% points from the original point cloud, with included duplicity and cropping. Duplicity of the pixel is elegantly visible in fig 3.7, where LiDAR scans wall of building, but in the projection is just a straight line.

We add 3 features to the 2D grid, coordinates z, intensity, and True/False for the existing scan point. The intensity was scaled to $\langle -1; 1 \rangle$ and we did the same with coordinate z, where -1 is the lowest scan point and 1 for the highest scan point. Third feature as in Front View is artificially created and tells the neural network, if this pixel have some corresponding scan point, we add 1. In the case that no scan point was captured in discrete location x,y, we add 0. Therefore, we created grid [3, 512, 1024] or [Number of features, Height, Width].

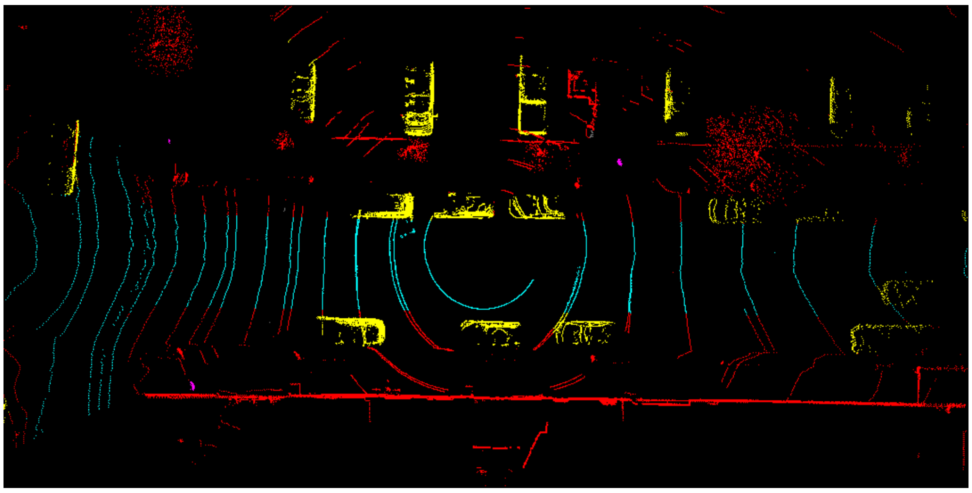
Figure 3.7: Visualisation of Bird's Eye View projection

3.3 2D Model

The model used for the segmentation task in the frontview and BEV data representation is a modified version of U-net [24].

Model can be divided into two parts, encoder and decoder. In the encoder, we used convolution layers with padding 1 and kernel size 3x3, batchnorms with momentum 0.1 and epsilon 1e-5. As an activation function, we decided to use relu. For reduction of dimensions a maxpool layer with kernel size 2x2 and stride 2 is used. From the maxpool layer, we get also indices (information in which pixel was maxpool applied) and put them to the similar layer in the decoder. Feature map from the last convolution layer from the subset of convolutional layers with the same size in the encoder is propagated to the decoder, where will be concatenated with the feature map with the same size.

The task is a multiclass segmentation problem and the occurrence of individual classes in the dataset is highly unbalanced, therefore we weighted the loss function with a weight vector, which was computed from the training dataset. For BEV and frontview is the vector different, because every representation takes different number of pixels. Weighted vector's dimension is 1x7, where each index corresponds to the sum of all pixels in the ground truth for each class in labelled training dataset, then the vector is rescaled to the sum of all indexes is equal 1. Thereafter, we raise every element to the power of minus one. Finally, we scaled the vector to one. For the basic configuration of the model was chosen the multiclass loss function Cross-entropy loss.

As an optimization method, we tried Stochastic Gradient Descent and Adam optimizer. Subsequently, we found out that Adam achieves better results during the training.

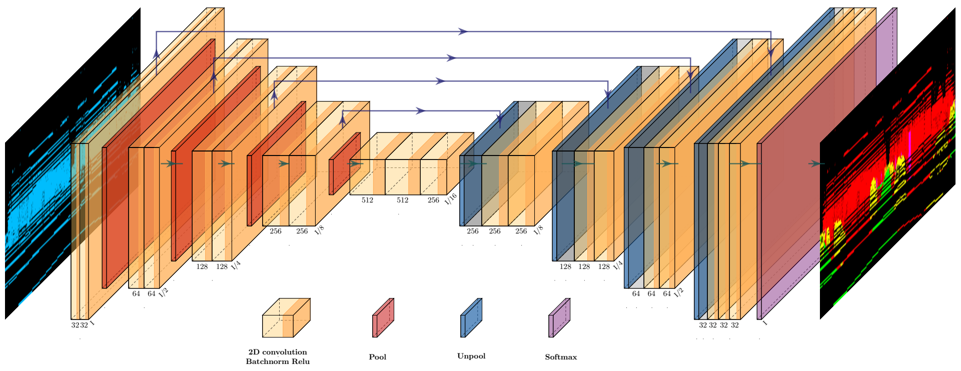

Figure 3.8: Architecture of our 2D model, which is inspired by U-net [24]

Architecture of the model is shown in the fig 3.8. Dark orange boxes correspond to max-pool layers, where dimensionality was reduced by coefficient 2. Light orange boxes represent a multilayer feature map with 2D convolutional layers, batchnorm layers and active function relu. Blue boxes are unpool layers, which takes indices from maxpool layers and concatinate the output with the feature maps from the encoder part. Number of channels is at the bottom of each box. Diagonally written numbers represent a reduction of size of the original grid.

3.3.1 Metrics

As a metric for semantic segmentation tasks is commonly used IoU (*Intersection over Union*), which defines the percent overlap between the target mask and the prediction output [17]. We also compare models based on their precision, recall and overall accuracy (OA), but we consider most the IoU and mean IoU ($\overline{IoU}$), because unlike precision and recall, the IoU considers FP and FN all at once. In table 3.4 we bring a brief overview of the metrics, which we are using in the evaluation of models.

Metric	Equation	Description
TP	-	True Positive
TF	-	True Negative
FP	-	False Positive
FN	-	False Negative
IoU	$\frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}$	Intersection over Union
$\overline{\text{IoU}}$	$\sum_{n=1}^{N} \frac{\text{IoU}_i}{N}$	Mean IOU
Precision	$\frac{\text{TP}}{\text{TP} + \text{FP}}$	Ratio of prediction which are relevant
Recall	$\frac{\text{TP}}{\text{TP} + \text{FN}}$	Ratio of correctly predicted pixels in ground truth
OA	$\frac{\sum_{n=1}^{N} c_i}{N}$	Overall accuracy, where c_i are correctly predicted pixels and N is number of all pixels.

Table 3.4: Definition of used metrics

3.3.2 Frontview baseline

This model will be used as a reference to compare with ensemble methods.

In the table 3.5 we can see that the dataset in frontview is very unbalanced. Classes two, three, and four (pedestrians, cyclist, and other moving objects) in particular are very underrepresented, which is reflected in the results. We tackle the balancing of the dataset in the loss function with weight vector.

class	0	1	2	3	4	5
training dataset [%]	81.53	11.25	0.31	0.02	0.06	6.81

Table 3.5: Occurrence of classes in Frontview

Inclusion of classes is very unbalanced, moreover in one frame is on average only 14.27 % of valuable pixels, the remaining 85.73 % are unreflected rays. Thus, convolutional kernels count worthless pixels until valuable pixels are not seen by the receptive field of pixels.

The baseline model was trained three times from scratch and we chose the best model based on the results of metrics mean IOU.

Baseline in frontview data representation			
Class	Precision	Recall	IOU
0 - background	**0.956**	**0.959**	**0.919**
1 - vehicles	0.884	0.909	0.812
2 - pedestrians	0.365	0.400	0.236
3 - cyclists	0.007	0.005	0.003
4 - moving object	0.520	0.112	0.102
5 - road	0.857	0.799	0.705
Mean IOU: 0.463 Precision: 0.598 Recall: 0.531 use of points: 87.7 %			

Table 3.6: Results of frontview baseline model on the test dataset

As we can see in table 3.6, the base model has relatively good predictive ability in classes 0, 1 and 5, which are mostly represented in frontview. Otherwise, classes 2, 3 and 4 do not reach even one percent representation even if we combine them into one class. Neural network has few frames to learn these classes and in addition, pedestrians, cyclists, or moving objects take pixels in the order of units, therefore the predictions are very difficult.

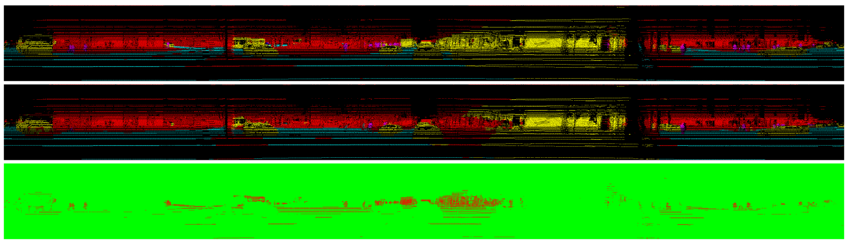

Figure 3.9: Worst prediction in fronview, from above (Ground truth; prediction; true/false, where green is for good prediction and red for wrong prediction)

3.3.3 Bird's eye view baseline

For reference in BEV domain was trained the BEV base model.

The representation of classes three, four and five is even worse in BEV representation, because pedestrians and cyclist takes fewer pixels. On these objects, BEV takes only the heads of people and a few more pixels.

class	0	1	2	3	4	5
training dataset [%]	67.31	17.30	0.34	0.03	0.00	14.93

Table 3.7: Occurrence of classes in Bird's Eye View

The number of valid pixels is on average 3.92 % from 2D grid. The remaining pixels are occupied by the sixth class, which represents the unreflected ray. This number is not surprising with respect to how one point cloud is captured. LiDAR scans horizontally, which gives a precondition for frontview transformation or projection on a cylinder, when the condition is to capture the most pixels.

Baseline in Bird's Eye View data representation			
Class	Precision	Recall	IOU
0 - background	**0.952**	**0.927**	**0.886**
1 - vehicles	0.835	0.918	0.777
2 - pedestrians	0.458	0.644	0.365
3 - cyclists	0.529	0.528	0.360
4 - moving object	0.567	0.453	0.336
5 - road	0.857	0.872	0.761
Mean IOU: 0.581			
Precision: 0.700			
Recall: 0.724			
use of points: 20.9 %			

Table 3.8: Results of bird's eye view baseline model on the test dataset

Despite the fact that the occurrence of classes 2, 3, and 4 is very low, the model is able to predict them better than the baseline model in frontview. As we can see on fig 3.10, object are in more compact form and they are surrounded by class 6. The neural network can easily put the boundaries of objects on the grid. On the other hand, the BEV base model confuses people with similar objects such as bushes.

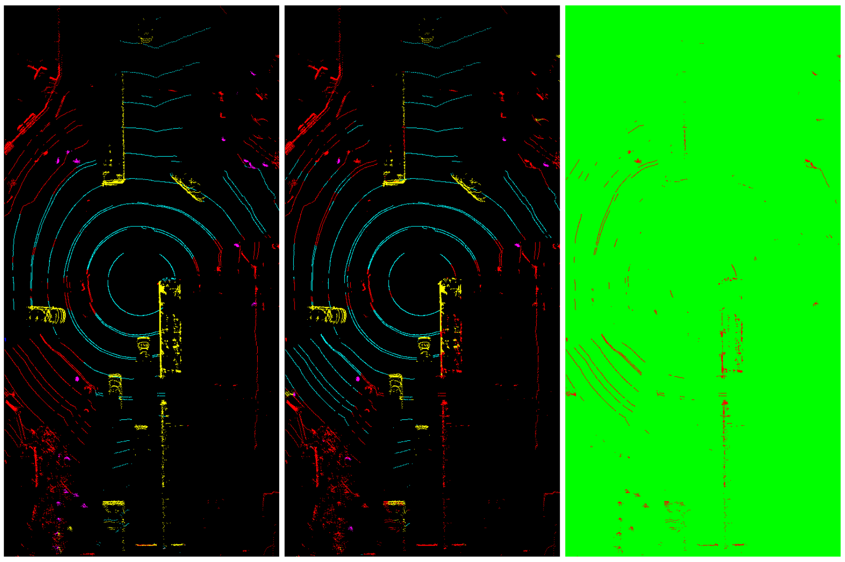

Figure 3.10: Worst prediction in bird's eye view, from left (ground truth; Prediction; True/False, where green is for good prediction and red for wrong prediction)

3.4 Range Level Ensemble

3.4.1 Frontview

We analyzed the performance of baseline model in frontview on different range levels. As we can see in table 3.9, neural networks have not the same predictive abilities in different distances. The reason of this inconsistency is the projection, because objects located closer to LIDAR sensors will take up more space in comparation with objects on the edge of the maximal range, where these objects take few pixels and they are very hard to predict. Our frontview projection takes only scan points closer than 150 metres.

Baseline model - IOUs							
Classes	Distance [m]						
	(0; 10)	(10; 20)	(20; 30)	(30; 40)	(40; 50)	(50; 60)	(60; 70)
0	0.905	0.920	0.918	0.927	**0.946**	0.932	0.944
1	**0.883**	0.688	0.672	0.574	0.575	0.446	0.362
2	0.223	**0.416**	0.199	0.058	0.003	0.017	0.016
3	**0.006**	0.003	0.000	0.000	0.000	0.000	0.000
4	**0.131**	0.110	0.112	0.046	0.036	0.042	0.080
5	**0.770**	0.707	0.667	0.601	0.545	0.493	0.421

Table 3.9: Analysis of baseline model performance in different ranges of distances

From the analysis of performance on different levels. We decided to train two types of models, where each of them will focus on a different range of distances. We choose the maximal distance to 50 m. Beyond this threshold, individual objects have not sufficient number of scan points for the neural network to decide about the class of scan points. Moreover, this dataset contains only the urban scene. Thus, we can assume that for autonomous driving is this distance sufficient with respect to the braking distance of cars driven in the city.

Success in individual distance range deals *PIXOR real time object detector from pointclouds*. The authors performed an analysis of several models in the distances 0-30, 30-50, 50-70. The results show that the performance in distance 30-50 drops by 30 percent on average and in the farthest interval the results are very weak. [31]

- Model focused on the surrounding area $\langle 0; 25\rangle [m]$
- Model focused on remote areas $\langle 25; 50\rangle [m]$

For both types of models, we tried several methods to include distance in the loss function. Best method will be selected and the models from both distance ranges will be work together as their final focus range of distances will be $\langle 0; 50\rangle [m]$. Each of the methods of models is trained for one time due to the time complexity.

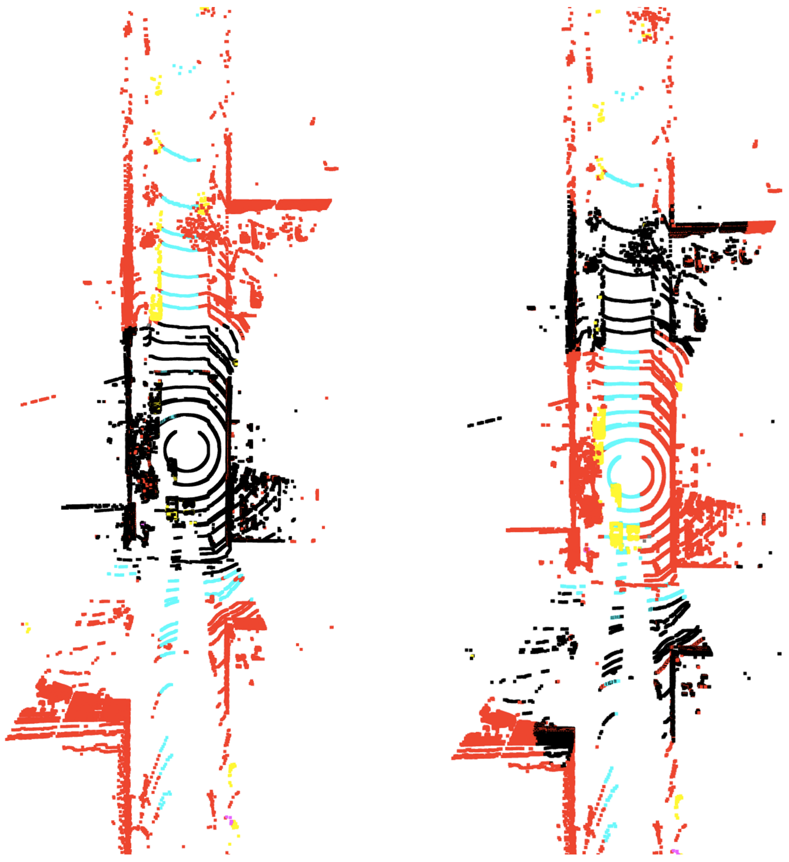
Figure 3.11: Example of focus area, where black pixels correspond into individual range functions; from left: surrounding area, remote area

As we can see on fig 3.11, the method focusing in the remote areas, which are in the distance from 25 up to 50 m, has very difficult work due to the sparsity of the data in these locations.

The model focused on the surrounding area

This model is specialized in the range of distances $\langle 0; 25\rangle [m]$. Model is concentrated with the range function. We tried three types of this function.

The range function is masking points with weights based on the distance of the scan point. As we can see in equation 3.7, range function multiplies the loss based on the distance of the scan point. Variable d denotes distance of the scan point.

$$\text{range-CE}(d) = \text{range}(d) \cdot \text{CE} \tag{3.7}$$

- **Baseline model** - This is the reference method, where we are using cross-entropy loss with weight vector. The model was described in section 3.3. The range function is 1 in all pixels, therefore there is no difference with normal CE.

$$\text{range}(d) = 1, d \in \mathbb{R} \tag{3.8}$$

- **Mode 1** - To this model entire frame is passed, thus neural network see every pixel, but loss is computed only in the surrounded area. We are using *range-Focal loss* with a weighted vector, which was computed for every pixel from the training dataset located closer than 25 metres.

$$\text{range}(d) = \begin{cases} 1, & d \in (0; 25\rangle [m] \\ 0, & d \in \langle 25; 150\rangle [m] \end{cases} \tag{3.9}$$

- **Mode 2** - This approach compute loss in every pixel, but in surrounded area is loss 10 times bigger then in others pixels. We used *range-Cross Entropy loss* with a weight vector, which was computed in all pixels in the training dataset.

$$\text{range}(d) = \begin{cases} 10/9, & d \in (0; 25\rangle [m] \\ 1/9, & d \in \langle 25; 150\rangle [m] \end{cases} \tag{3.10}$$

- **Mode 3** - This method computes the loss from range distance $\langle 0; 75\rangle [m]$. Maximal loss is applied to the area where this model will work, i.e. from 0 to 25 metres. Then based on the distance is a linear descent of the loss to 75 metres. For pixels located beyond 75 metres, no loss is computed, but the neural network propagates information from these pixels to pixels where the loss is computed through the receptive field of convolutional kernels. We use *range - Cross Entropy loss* with a weight vector from all pixels seen in the training dataset.

$$\text{range}(d) = \begin{cases} 1, & d \in (0; 25\rangle [m] \\ -\frac{1}{50}(x - 75), & d \in \langle 25; 75\rangle [m] \\ 0, & d \in (75; 150\rangle [m] \end{cases} \tag{3.11}$$

Every approach in the range function is for every frame rescaled, so that the sum of all values of the range function in all pixels is $2496 \cdot 244 = 609024$. This number is the sum of pixels in one frame.

$$\text{range}_{ij}(d) = \text{range}_{ij}(d) \cdot \frac{2496 \cdot 244}{\sum_x \sum_y \text{ranges}_{xy}(d)} \tag{3.12}$$

We can see in equation 3.12, that coefficient of rescale is different in every frame, because the sum of weights over the whole 2D grid will be different in every frame. Variable *i,j* denotes actual pixel and *x,y* are set of all coordinates in the 2D grid.

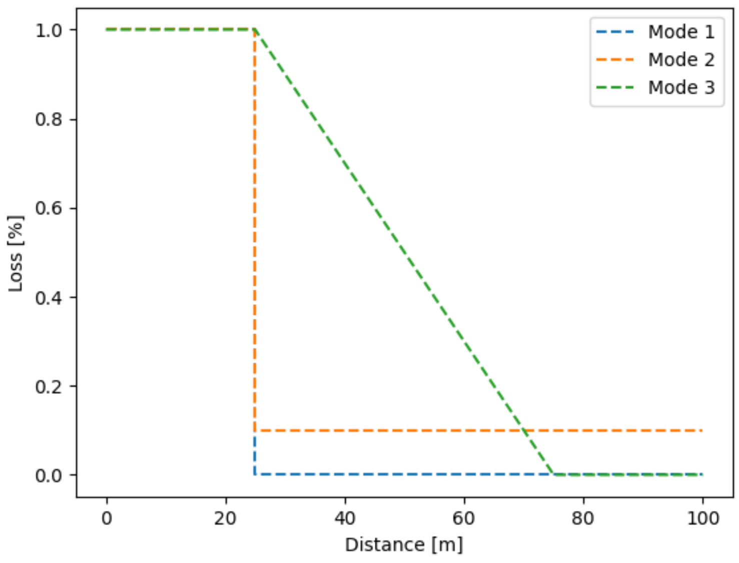

Figure 3.12: Calculation of losses based on individual methods in surrounded area

The model focused on remote areas

To complete ensemble we proposed a second model, which is focusing on the remote areas, i.e. predicting pixels, which are located from 25 to 50 metres. In this interval of distance, the data are relatively dense to recognize objects from them. In this approach, we also tried three types of the range function.

- **Mode 1** - In this method is used *range-CE* with the same weight vector as in the base model. The range function has on output 1 in pixels, which correspond to pixels from the focus area. Otherwise, the range functions have the value 0. Neural network consumes all pixels, but only on a few of them compute loss.

$$\text{range}(d) = \begin{cases} 1, & d \in \langle 25; 50\rangle [m] \\ 0, & d \in \langle 0; 25) \cup (50; 150\rangle [m] \end{cases} \tag{3.13}$$

- **Mode 2** - This approach has on an output 10/9, where the pixel is in the range from 25 to 50 metres. Otherwise, put 1/9 on output. Loss in the area it focuses on is 10 times bigger, then in others pixels. As the loss function is used *range-CE* with the same weight vector as in the baseline model.

$$\text{range}(d) = \begin{cases} 10/9, & d \in \langle 25; 50\rangle [m] \\ 1/9, & d \in \langle 0; 25) \cup (50; 150\rangle [m] \end{cases} \tag{3.14}$$

- **Mode 3** - This method computes loss from the distance range from 0 up to 75 meters. For the distance range at which it will operate, i.e. $\langle 25; 50\rangle [m]$, return 1 in the range function. Then applied linear descent on both sides of a interval as shown in fig 3.13. The method uses *range-CE* with the weight vector, which was computed on all pixels in the training dataset.

$$\text{range}(d) = \begin{cases} 1, & d \in \langle 25; 50\rangle [m] \\ \frac{1}{25} \cdot d, & d \in \langle 0; 25) [m] \\ -\frac{1}{25}(d - 75), & d \in (50; 75\rangle [m] \\ 0, & d \in (75; 150\rangle [m] \end{cases} \tag{3.15}$$

All three approaches apply a rescale of the range function on every frame. The result is that the sum of all outputs from the range function in one frame has the same value. This value is equal to the number of pixels in one frame. The rescalation can be seen in equation 3.12.

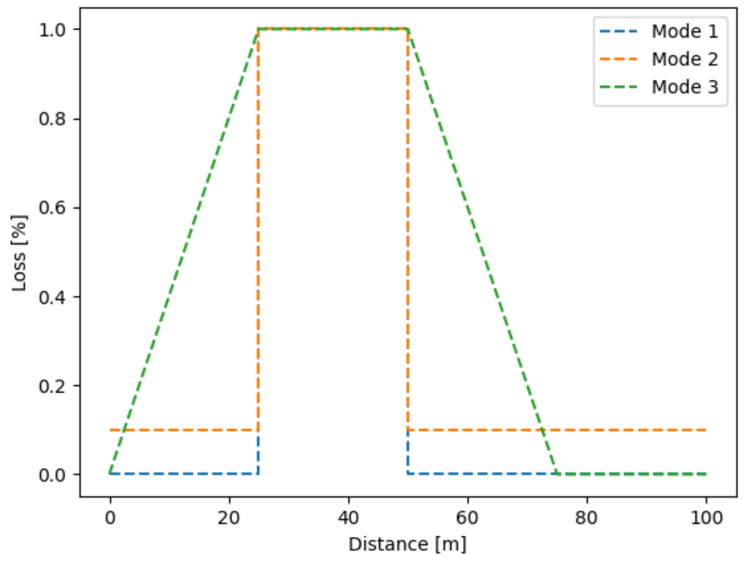

Figure 3.13: Calculation of losses based on individual methods in remote areas

3.4.2 Bird's Eye View

Input frames to the bird's eye view model captured an area with a capacity of 32 x 64 meters. Thus on the grid appears a captured ray, which was maximally at a distance 35.8 metres. Since the model takes only a few scan points, it could be used in areas where the frontview has not much data and its prediction ability is very low.

As the LiDAR will be used for detection around the car in the city environment, we tried to focus on the surrounding area of the car, which is crucial for safe driving of autonomous vehicles in the cities, given the fact that the environment around the car can be very dynamic and complex.

Surrounding of the car

2D grid has size 512x1024 pixels, which is 34x68 m in real units. We are focusing on surroundings of the car. This model focusing on a grid size 300x602 pixels or 20x40 m in real units, hereinafter as "Focus-Area". We proposed four methods how to apply distance in the loss function.

In the 2D grid in bird's eye view, the distances are incorporated directly into coordinates x,y in the grid, therefore we proposed for bird's eye view grid function *loss mask* istead of the *range* function. It has advantages such as every *loss mask* can be pre-computed and so it reduces the computational complexity.

Modified losses used bird's eye view such as *range-CE* or *range-FL*, compute with *loss mask* instead of *range function*.

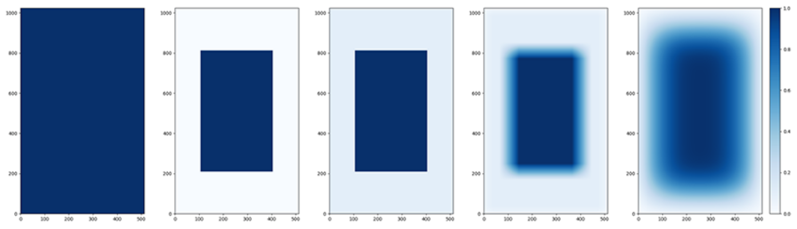

Figure 3.14: Loss: Normal, mode1, mode2, mode3, mode4

- **Base model** - It was described 3.3.3. The model use the *CE* loss with a weight vector from the training dataset in all pixels.

$$\text{loss mask}(x, y) = 1, x, y \in \mathbb{Z} \tag{3.16}$$

- **Mode 1** - The model use *range-CE* with a weight vector, which was computed in focus area in the training dataset. On the input of the model is all frame, but loss is computed from the *focus area*.

$$\text{loss mask}(x, y) = \begin{cases} 2.903, & x, y \in \text{"Focus-Area"} \\ 0, & x, y \in \mathbb{Z} \setminus \text{"Focus-Area"} \end{cases} \tag{3.17}$$

  On pixels, where the loss is computed, the *loss mask* puts on output 2.903, because we wanted to have the sum of all outputs from the range function equal to the sum of all pixels in one frame.

- **Mode 2** - This approach use *range-FL* with a weight vector from all pixels in the training dataset. Loss in focus area is 10 times bigger, then in others pixels.

$$\text{loss mask}(x,y) = \begin{cases} 2.439, & x,y \in \text{"Focus-Area"} \\ 0.2439, & x,y \in \mathbb{Z} \setminus \text{"Focus-Area"} \end{cases} \tag{3.18}$$

With the same purpose as in mode 1, the outputs from the *loss mask* are scaled.

- **Mode 3** - This method use *range-CE* with the same vector as the base model. The *loss mask* is created as a grid, where in focus area is put 1 and in others pixels 0.

$$\text{loss mask}(x,y) = \begin{cases} 1, & x,y \in \text{"Focus-Area"} \\ 0, & x,y \in \mathbb{Z} \setminus \text{"Focus-Area"} \end{cases} \tag{3.19}$$

We want a linear descent on the edge of these two areas. Thus, we applied the avg-pool2D function with kernel size 81x81 and padding 40 to keep the size of the loss mask. This operation applies a 2D average pooling over an input loss mask and creates a continuous transition between *Focus-Area* and the rest of the 2D grid.

$$\text{loss mask}_{new}(x,y) = \text{loss mask}(x,y) \cdot \frac{1024 \cdot 512}{\sum_i \sum_j \text{loss mask}(i,j)} \tag{3.20}$$

As the previous method, we rescaled every pixel in the *loss mask* as it is shown in equation 3.20, where variables *x,y* denotes actual pixel and $\sum_i \sum_j \text{loss mask}(i,j)$ is the sum of weights in the whole 2D grid.

- **Mode 4** - This method is very similar to mode 3, with the difference that the operation avg-pool2D were applied five times. The *loss mask* was also rescaled.

3.5 Projection ensemble method of range models

3.5.1 Frontview

As we can see at table 4.6 from the remote area, mode 2 has the best performance in comparison with all methods and the average of them. In the surrounding area, we have the best results on the average of all models. We fuse the models from these two areas, so that mode 2 predicts pixels in distance from 25 up to 50 metres. These predictions are added to the prediction from the average of models in the surrounding area, which compute pixels in distance from 0 to 25 metres.

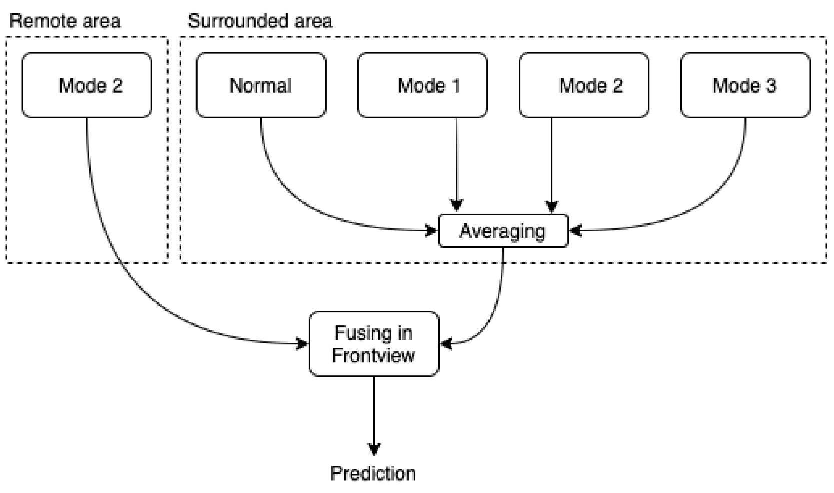

Figure 3.15: Ensemble of range models in frontview

3.5.2 Bird's eye view

In BEV, we tried the range loss function only on the surrounding area, where the predictions are crucial for the safe driving in the city. For these pixels located in this area, the average of all models proved to be the best.

The baseline model works for a more remote area. By combining the pipelines working on these two areas creates a model working on the distance from 0 up to 35.8 metres.

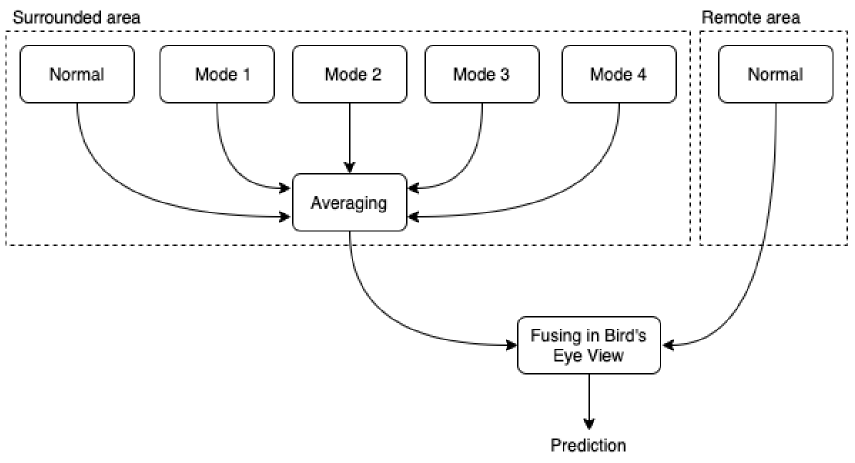

Figure 3.16: Fusing in Bird's Eye View

3.6 Fusing Front View and Bird's Eye View (FEP)

We want to apply the multiview ensemble method, which merge *Projection ensemble method of range models* from Frontview and BEV. To achieve this, we need to choose one data representation where the outputs from these two pipelines will work.

The use of a Frontview as the final data representation seems to be the most sensible for several reasons. Frontview can use 87.7% from original pointcloud, whereas BEV use on average 21%. We can also choose to work in 3D with the LiDAR pointcloud, where BEV and Frontview can be transformed. However, Pointnet++ did not achieve as good results as the 2D model, which we observe in BEV and Frontview.

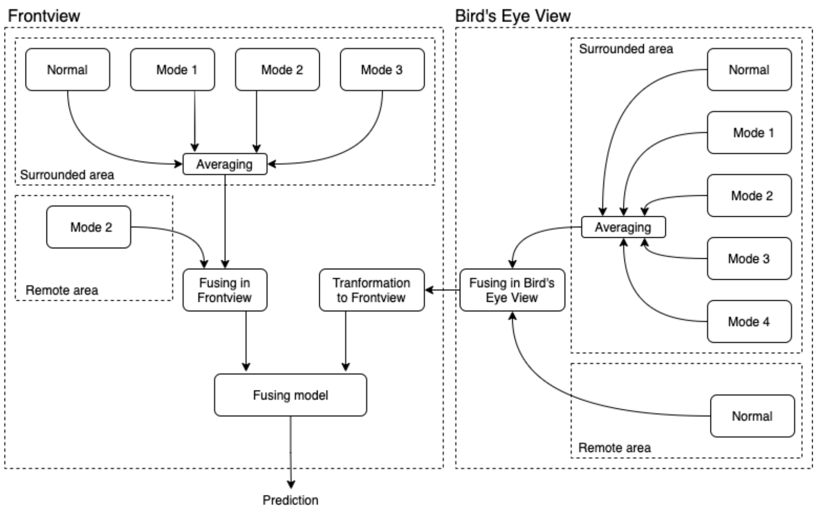

Figure 3.17: Visualisation of the FEP

To combine the outputs from both projections, we must transform BEV to Frontview. To reduce the computational complexity, we precomputed indices, which tell us the information where the individual points in the BEV data representation frames have to be mapped. The process is vectorized. The 95% of all points from BEV are used in the Frontview. Losses occur because BEV is able to map more points, which corresponds to the same pixel in Frontview. The pixels, which occurs in frontview, but BEV is unable to decide about the classification is added zero to all channels.

3.6.1 Fusing model

To improve the merge of outputs from BEV and Frontview. We want to learn *fusing model*, which consumes two outputs in shape [Batch Size, 224, 2496, Number of channels]. The number of channels is 7 and should represent the final features of invidual classes in pixels.

Scalar method

This method applies the softmax to both tensors, because the original outputs contain features in different ranges of values. The model has one or a few parameters which will multiply the output of BEV data representation.

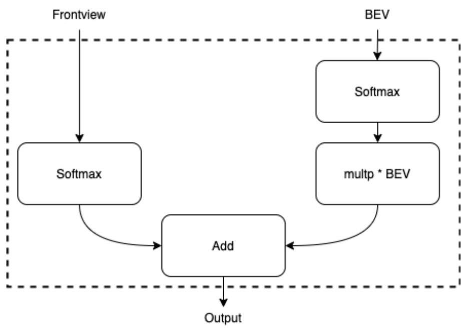

Figure 3.18: Visualisation of procces in fusing model, which is based on method *Scalar*

a. Learnable scalar for BEV. The simplest method is to use one learnable scalar, which multiplies the output from BEV and then adds the output from Frontview. Scalar *multp* tells us how much better or worse are the predictions in Bird's eye view then in the Frontview.

$$\text{Output} = \text{Output}_{frontview} + multp \cdot \text{Output}_{BEV} \tag{3.21}$$

The model contains one number, therefore we trained it only during 20 epoches. Initial value was set to one. Final value of the parameter is seen in the equation 3.22.

$$\text{multp} = 2.086 \tag{3.22}$$

The number corresponds to the results achieved by each pipeline, where BEV pipeline gets a better result on the test dataset. The achieved mean IOU is 0.6257.

b. Learnable scalar for each class. This is an extension of the previous method. Each channel from BEV is multiplied by a learnable scalar, which tells us on which projections we can rely on more in individual classes. It should perform better then the previous method if one projection has some worse and some better results in invidual classes.

$$\text{Output}_{frontview} + \text{Output}_{BEV} \cdot \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{bmatrix} \tag{3.23}$$

In equation 3.23, each parameter x_i corresponds to a scalar which will multiply the exact channel in the output from BEV pipeline. The model was trained with 20 epochs, which should ensure the convergence to optimal values. The initial value for every parameter was chosen number one, because at this point the model adds both projections with the same ratio. Loss was computed on pixels located closer than 50 metres, because the outputs from frontview and BEV pipelines are computed on these pixels.

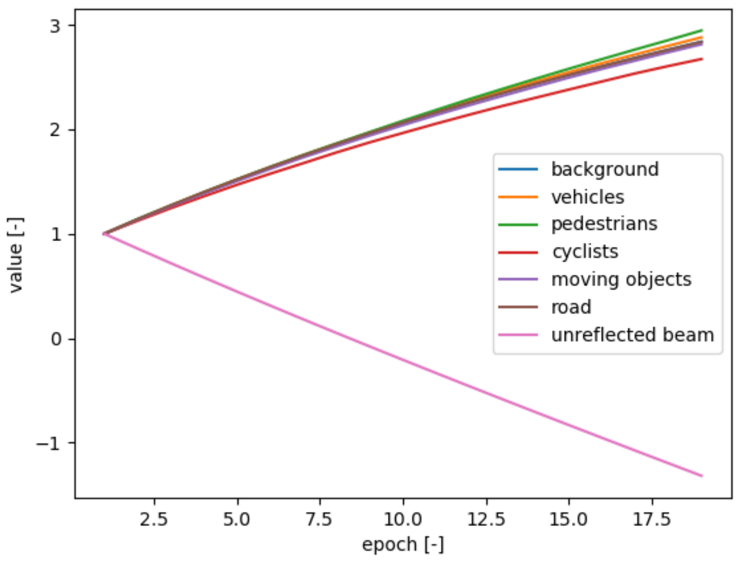

Figure 3.19: The development of values for invidual classes during training

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} \text{background} \\ \text{vehicles} \\ \text{pedestrians} \\ \text{bicycles} \\ \text{moving objects} \\ \text{road} \\ \text{unreflected ray} \end{bmatrix} = \begin{bmatrix} 2.063 \\ 2.074 \\ 2.082 \\ 1.965 \\ 2.040 \\ 2.065 \\ -0.207 \end{bmatrix} \tag{3.24}$$

The development of the values from equation 3.23 is shown in figure 3.19, where we can see that this model gives bigger weight to every class from BEV. This corresponds to the results from BEV and Frontview pipelines, where BEV has better predictive ability in every class. Updating of the model was stopped in the ninth epoch, where the accuracy in the validation dataset has reached the peak. An input to the model are probabilities, therefore larger values can no longer improve the results of the pipeline.

The final coefficients x_i are shown in equation 3.24, where every parameter is very close to the parameter from *method a*. Thus, both methods achieved similar or improvements.

Smoothing method

We tried multiple variants of kernel size, number of layers, and two approaches.

1. First approach consumes features in the original form. Outputs from BEV and Frontview are added and then the result is put to the input of the model. The model contains multiple convolution layers, batchnorm layer with momentum 0.1 and active function relu..

2. Second approach applies on both tensors softmax to rescale values to the same range. Thus, this type of models work with probabilities and we applied only convolutional layers, because relu and batchnorm would have not any effect.

Every model was trained for 50 epochs.

a. Two convolutional layers with kernel size 3 . The model contains a convolutional layer with kernel size 3 and padding 1. Then follows another convolutional layer with kernel size 3. We tried only approach two. This method has 882 learnable parameters and the receptive field is 5x5, which would perform worse than methods with bigger receptive field.

b. Two convolutional layers with kernel size 7 . On this setup was tried approach one and two. This model contains a convolutional layer with kernel size 7 and padding 3. Convolutional layers have in total 4802 parameters and the receptive field is 13x13, which would be enough large to capture even a big object.

d. Four convolutional layers with kernel size 3 . The deepest network contains four convolutional layers with kernel size 3 and padding 1 to keep sizes. On this method is tried approaches one and two. Convolutional layers have in total 1764 parameters and the receptive field is 13x13.

Combination of smoothing and learnable parameters (UNITE)

We choose a learnable scalar for each class and a smoothing model with kernel size 3 and depth of two layers. This smoothing model has the best performance on the test dataset.

Firstly, the output from BEV is multiple by a vector of parameters, then an output from Frontview is add. To this final tensor is applied a smoothing model.

3.7 Semi-Surervised training

We are using a naive semisupervised learning method called as pseudo-labelling. We generated a pseudo-labels of the non-annotated frames by FEP. Thus, we extended the original training dataset to 15593 frames. The ratio between annotated and pseudo-labelled frames is 5701:9892 or 1:1.735. The ratio shows as that the occurrence between annotated and non-annotated is not so high.

We tried two approaches. The primary method of application of pseudo labels is to use them only in the *fine tune training phase*. This method achieved the state-of-the-art on the MNIST dataset. [16]

We used the frontview baseline model trained on the training dataset, which has 5701 annotated frames. The parameters of the model were loaded and trained in *fine tune phase* for 30 epochs.

Second used approach uses pseudo-labels as hard labels. The authors [25] of this method also introduce an uncertainty weight for each sample loss, which is higher for samples that have distant k-nearest neighbors in the feature space. [3]

We treat the pseudo labels as it was ground truth. The method was examined on the original dataset plus pseudo labels and on the reduced training dataset numbering 2142 annotated frames in order to increase the ratio between original ground truth and pseudo-labels. Both models were trained from scratch on 100 epoch with weighted vector counted on the original training datasets. The probability distribution should not change, because pseudo labels were generated by FEP, which use the same weighted vector.

All implementation codes are available on: `https://gitlab.fel.cvut.cz/students/pokorny-simon`

Chapter 4

Experiments

4.1 Invidual models with range function

4.1.1 Frontview

The model focused on the surrounding area

Results with the reduced training dataset. We trained all proposed types of the range function on the reduced training dataset for one time. The reduced training dataset has 2142 frames from 22 scenes. Validation and testing datasets remain the same. We did this to compare the results with the small amount data and big amount of data. The difference between them is 2142:5701, or 1:2.66. In tab 4.1 we can see that none of the methods improved the prediction ability. Small amount of data and little diversification of scenes caused that the models have not learn. This is especially true for models with modes 2 and 3. The results show that the baseline model has the second best performance on a small amount of data. Mode 1 has achieved the best mean IOU, but the difference is small that we attribute it to the variance that neural networks have. On the other hand, the base model was chosen from 3 training cycles and the models with modes 1,2, and 3 were trained for 1 time, therefore modification of the loss function with the range function has the potential for improvement of predictive ability. We probably did not reach the maximal capacity of the model. Each of the models has sufficiently fit to the small training dataset, therefore we did not see a big difference.

Overall stats of IOU				
Class	Baseline	Mode 1	Mode 2	Mode 3
0 - background	0.856	**0.873**	0.856	0.811
1 - vehicles	0.699	**0.738**	0.702	0.679
2 - pedestrians	0.182	0.185	**0.233**	0.123
3 - cyclists	0.249	**0.279**	0.166	0.248
4 - moving object	0.080	0.088	**0.090**	0.062
5 - road	**0.699**	0.673	0.660	0.676
Mean IOU	0.460	**0.472**	0.451	0.433
Precision	0.594	0.618	0.575	**0.627**
Recall	0.600	0.590	**0.611**	0.521

Table 4.1: Results of individual methods trained on the reduced dataset. The evaluation took place on pixels located from 0 to 25 metres

Results with the original dataset. The whole labelled training dataset has 5701 frames, which provides a good diversity of the data. Prediction ability should be improved compared with results on the reduced training dataset. This is contradicted by the results, where the baseline model achieved approximately the same results as on the reduced dataset even that was also trained for 3 cycles and the best performing model was chosen. The baseline model is trying to describe the whole scene, therefore we probably reach the maximal descriptive capabilities of the frontview baseline model. Based on this every small improvement in models with range function we can take as confirmation of the functionality of this approach.

Models with range function were trained for 1 time. Mode 2 improved the mean IOU by 5.4% and mode 3 by 12.7% compared to the training on the reduced dataset. Results of mean IOU achieved by the baseline model have been increased by 5.4% using the range function with mode 3 and by 3.6% using mode 2.

It is hard to say which of the modes perform better, because each of the modes perform better on the bigger dataset or on the contrary. In any case, the range function helps the model to safe capacity to better describe the area on which is focused.

We tried to average all models and we get the best performance compared to all models separately. Thus, we suppressed the variance of the neural network.

Overall stats of IOU					
Class	Baseline	Mode 1	Mode 2	Mode 3	Averaging
0 - background	**0.919**	0.881	0.894	0.906	0.916
1 - vehicles	0.812	0.780	0.765	0.797	**0.832**
2 - pedestrians	0.236	0.244	0.332	**0.339**	0.333
3 - cyclists	0.003	**0.010**	0.004	0.002	0.002
4 - moving object	0.102	0.076	0.172	**0.152**	0.150
5 - road	0.705	0.693	0.711	0.732	**0.746**
Mean IOU	0.463	0.447	0.480	0.488	**0.497**
Precision	0.598	**0.620**	0.581	0.591	0.602
Recall	0.531	0.507	**0.601**	0.592	0.592

Table 4.2: Results of methods on pixels located from 0 to 25 metres

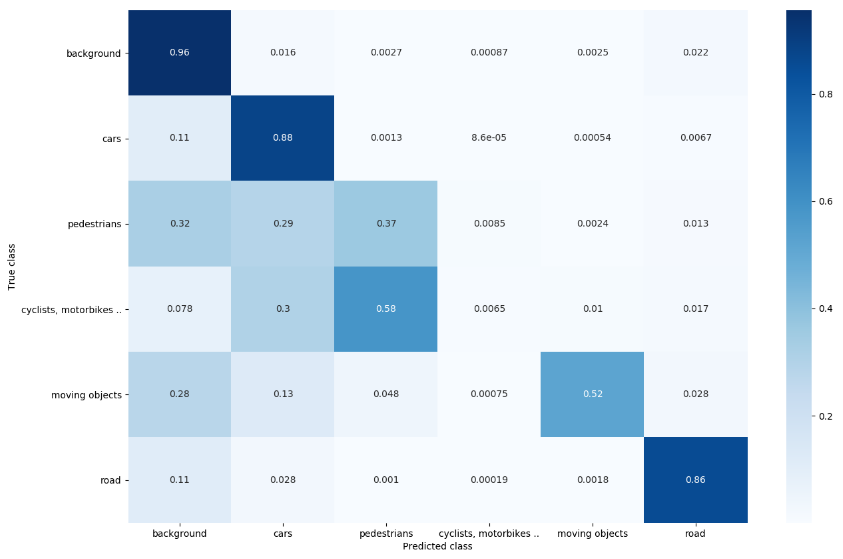

Figure 4.1: Confusion matrices of the baseline in frontview projection

Merge of classes. In fig 4.1 we can see that the baseline model has a problem to identify class number 3, which contains cyclists and confuse it with class number 2, which represent pedestrians. The models with modes 1,2, and 3 also confuse classes 2 and 3. Cyclists and pedestrians are very similar in sparse coverage of scan points by LiDAR, because the sensor captures mostly the human and on the bike are only a few pixels. Thus, we decided to merge these two classes.

Models were already trained to separate these classes, thus the model uses some of the capacity to divide the feature space. Assuming that the models are taught from scratch with just only 6 classes. They would probably achieve better results.

Overall stats of IOU						
Class	Baseline	Mode 1	Mode 2	Mode 3	Averaging	Mode 2+3
0	0.912	0.881	0.894	0.906	**0.916**	0.907
1	**0.835**	0.780	0.765	0.797	0.832	0.798
2+3	0.329	0.278	0.388	0.399	0.380	**0.404**
4	0.123	0.076	**0.172**	0.152	0.150	0.162
5	0.733	0.693	0.711	0.732	**0.746**	0.735
Mean IOU	0.586	0.542	0.586	0.597	**0.605**	0.601
Precision	0.738	**0.747**	0.703	0.715	0.729	0.714
Recall	0.675	0.613	0.730	0.733	0.726	**0.743**

Table 4.3: Results on pixels located from 0 to 25 metres and with merge of second and third class

Results with merge of classes. After unification classes 2 and 3, the differences between individual methods get smaller. Best results achieved the average of all models with the same coefficient. We tried to average the models with method 2 and method 3, but we get worse mean IOU than the average of all models.

The model focused on remote areas

Results with reduced training dataset. We trained the models with range function on the small amount of data as the models focused on the surrounding of the car. Classify pixels, which are located from 25 up to 50 metres is more complicated than in the closer pixels. Objects are covered with fewer points. Despite the fact that the baseline model see a few of the frames did not describe them as well as the models learned directly on them using the range function.

Each method achieved at least small improvements in comparison with the baseline model, because the baseline model has very weak predictive ability in this distance range in comparison with the surrounding area.

As we can see in tab 4.4, even the average did not achieve the same results as the model with the range function itself.

Overall stats IOU					
Class	Baseline	Mode 1	Mode 2	Mode 3	Averaging
0 - background	0.920	0.922	**0.924**	**0.924**	0.919
1 - vehicles	0.456	0.458	**0.465**	0.453	0.450
2 - pedestrians	0.037	0.076	0.063	**0.095**	0.038
3 - cyclists	0.000	0.000	0.000	0.000	0.000
4 - moving object	0.042	0.056	0.058	**0.077**	0.025
5 - road	**0.569**	0.527	0.550	0.566	0.568
Mean IOU	0.337	0.340	0.343	**0.353**	0.333
Precision	0.477	0.462	0.476	**0.488**	0.478
Recall	0.407	0.430	0.426	**0.438**	0.405

Table 4.4: Results of invidual methods trained on the reduced dataset. The evaluation took place on pixels located from 25 to 50 metres

Results with the original dataset. Bigger amount of data further deepened the difference between the baseline model and models using range function in the loss. This time it was the best model with mode 2. The best results was achieved on a different model than on a reduced dataset. Despite this fact, the range function helps the neural network aim to the areas, where it has better results. The worse the results are of the baseline in that area, the better the progress we can get using the range function.

Overall stats IOU					
Class	Baseline	Mode 1	Mode 2	Mode 3	Averaging
0 - background	0.934	0.930	**0.949**	0.940	0.943
1 - vehicles	0.611	0.560	**0.673**	0.560	0.633
2 - pedestrians	0.060	0.049	**0.089**	0.059	0.050
3 - cyclists	0.000	0.000	0.000	0.000	0.000
4 - moving object	0.057	0.071	**0.137**	0.131	0.090
5 - road	0.613	0.599	0.636	0.631	**0.641**
Mean IOU	0.379	0.368	**0.414**	0.387	0.393
Precision	**0.530**	0.513	0.525	0.498	0.521
Recall	0.435	0.430	**0.500**	0.480	0.460

Table 4.5: Results on pixels located from 25 to 50 metres

Results with merging of classes. Based on the same arguments, we merged classes 2 and 3. The evaluation was done with the same models as in *Results with the original dataset*, but we change the metrics. The difference between individual models is deepened.

Overall stats IOU					
Class	Baseline	Mode 1	Mode 2	Mode 3	Averaging
0	0.934	0.930	**0.949**	0.940	0.943
1	0.611	0.560	**0.673**	0.560	0.633
2+3	0.053	0.047	**0.088**	0.058	0.046
4	0.057	0.071	**0.137**	0.131	0.090
5	0.613	0.599	0.636	0.631	**0.641**
Mean IOU	0.453	0.441	**0.497**	0.464	0.470
Precision	**0.637**	0.617	0.630	0.597	0.625
Recall	0.518	0.514	**0.598**	0.574	0.548

Table 4.6: Results on pixels located from 25 to 50 metres with merging of classes 2 and 3

Bird's Eye View

Surrounding of the car

Results with merging of the classes. The baseline model was trained for 3 times and the best performing model was chosen. The models with individual modes were trained only for 1 cycle due to the time complexity. Training the baseline model has big variance in the results on the test dataset, therefore the results of models with modes can be distorted.

We merged classes 2 and 3 as in the frontview for the same reason. The models were trained for each class and we only merged the metrics in the evaluation. The results show us that the best performance has the average of all models. Despite the fact, that baseline model was trained for 3 times, the model with mode 4 achieved better results, particularly in classes 2+3 and 4, which are very hard to detect. Predictive ability on class five is similar in every approach. This also applies to the class 0.

The baseline model in BEV has relatively strong predictive ability in *the surrounding of the car*. Thus, models with range models do not improve the results significantly. A greater benefit has the average of more models. In our case of models with modes 1,2,3,4, and baseline.

Overall stats IOU						
Class	Baseline	Mode 1	Mode 2	Mode 3	Mode 4	Averaging
0	0.864	0.855	0.868	0.857	0.863	**0.877**
1	0.826	0.815	0.857	0.818	0.846	**0.859**
2+3	0.607	0.552	0.544	0.637	0.574	**0.639**
4	0.348	0.342	0.267	0.425	0.256	**0.451**
5	0.813	0.810	0.812	0.815	0.812	**0.824**
Mean IOU	0.691	0.675	0.670	0.710	0.670	**0.730**
Precision	0.806	0.785	0.828	0.814	**0.843**	0.824
Recall	0.799	0.797	0.750	0.827	0.749	**0.845**

Table 4.7: Results with merging classes 2 and 3

4.2 Ensemble methods

4.2.1 Frontview

This is the evaluation of fusing models focused on the surrounding of the car (0 up to 25 metres) and on the remote areas (25 up to 50 metres). The pipeline is visualise in fig 3.15. For remote areas we are using only the model with mode 2, because has the best results and for the closer area was chosen the average of all models. Outputs of these models are add. The evaluation on the test dataset shows that the ensemble of range models improved the mean IOU by 2.5 %, on top of that the ensemble has better ability in every class. Precision slightly decreased, but the difference is very small.

Overall stats IOU		
Class	Baseline	Ensemble of models
0 - background	0.917	**0.924**
1 - vehicles	0.819	**0.822**
2+3 - pedestrians + cyclists	0.271	**0.330**
4 - moving objects	0.108	**0.148**
5 - road	0.715	**0.731**
Mean IOU	0.566	**0.591**
Precision	**0.725**	0.716
Recall	0.648	**0.712**

Table 4.8: Results on the test dataset; evaluation was done only on pixels located from 0 up to 50 metres

4.2.2 Bird's Eye View

The process of the ensemble of models in BEV can be seen in fig 3.16. For *surrounding of the car* is used the averaged of all models. On the pixels where the average of the models is not focused on is chosen the baseline BEV model. Outputs from these areas are add-together.

The resulting pipeline achieved better results in IOU of all classes in comparison with BEV baseline model. Mean IOU, precision, and recall have been improved. The difference is not so noticeable, because we boost only the surrounding area, where the baseline has relatively good accuracy. We decided to be focused solely on the closest area, which is crucial for safe driving in the city environment.

Overall stats IOU		
Class	One model	Fusing pipeline
0 - background	0.886	**0.891**
1 - vehicles	0.777	**0.802**
2+3 - pedestrians + cyclists	0.398	**0.404**
4 - moving objects	0.336	**0.374**
5 - road	0.761	**0.768**
Mean IOU	0.632	**0.648**
Precision	0.741	**0.752**
Recall	0.768	**0.786**

Table 4.9: Results on the test dataset

4.3 Fusing Frontview and BEV

The final step to ensemble the FEP. In the section 3.6, we proposed several methods and types of *the fine tune model*, which will combine outputs from pipelines in BEV and Frontview.

We are trying to determine which of the methods will perform the best. Models based on *Learnable scalar* were trained for 20 epoches and models based on the *Smoothing* with approach 1 or 2 for 50 epoches, because contains many parameters. The *Unite*, which is a combination of both methods was trained for 50 epoches. As a reference (REF) we use a naive approach to add outputs from BEV and Frontview together.

Overall stats IOU									
		Learnable parametes		Approach 1			Approach 2		Unite
Class	REF	Multp BEV	Multp classes	2 layers, kernel 3x3	4 layers, kernel 3x3	2 layers, kernel 7x7	4 layers, kernel 3x3	2 layers, kernel 7x7	
0	0.928	**0.929**	**0.929**	0.926	0.926	0.926	0.927	0.927	0.926
1	0.826	**0.829**	**0.829**	0.819	0.822	0.823	0.825	0.828	0.824
2+3	0.346	0.366	0.366	0.336	**0.338**	0.339	0.352	0.361	0.346
4	0.199	**0.255**	**0.255**	0.195	0.179	0.166	0.170	0.167	0.157
5	0.747	0.750	**0.751**	0.742	0.737	0.737	0.735	0.731	0.735
Mean IOU	0.610	**0.626**	**0.626**	0.604	0.600	0.599	0.602	0.603	0.598
Precision	0.720	0.729	**0.730**	0.722	0.720	0.723	0.727	0.728	0.716
Recall	0.743	**0.761**	**0.761**	0.721	0.719	0.718	0.727	0.720	0.724

Table 4.10: Results of the fine tune model based on several methods

As we can see from tab 4.10 the naive approach *Learnable scalar achieved best results* achieved the best results. Smoothing approach did not work in any setup. I attribute these bad results to the fact that outputs from the pipeline are mostly generated by the averaging. Thus, the output of each pipeline is generalized and it is very hard to improve it. The *fine tune model* based on *the smoothing* reduces this generalizability and the final output a lot depends on its. Moreover, the data in frontview are scattered, because between valuable pixels are big gaps of pixels representing the unreflected ray. Thus, the smoothing method would work well on 2D RGB images, where we have only valuable pixels and the data are more continuous.

The method *multp classes* was chosen for generating the pseudo labels by FEP. However, the *multp BEV* would achieve the same effect, because the coefficients are very similar.

4.4 Semi-supervised learning

4.4.1 Original training dataset extended with pseudo labels

We tried two methods to use a naive semi-supervised technique called pseudo-labelling. The *fine tunning phase*, where we take the baseline model trained on 100 epoch and apply the fine tunning phase with the extended dataset on 30 epochs. The second approach trains the model from scratch on the extended dataset on 100 epoches. The ratio between annotated and pseudo-labelled frames is 5701:9892 or 1:1.735.

Overall mean IOU			
Class	Baseline	Fine tunning	From scratch
0 - background	0.917	0.928	**0.930**
1 - vehicles	0.819	0.851	**0.856**
2+3 - pedestrians + cyclists	0.271	0.348	**0.373**
4 - moving objects	0.108	**0.125**	**0.125**
5 - road	0.715	0.730	**0.732**
Mean IOU	0.566	0.596	**0.603**
Precision	0.725	**0.766**	0.760
Recall	0.648	0.662	**0.678**

Table 4.11: Semi-supervised learning results on pixels located from 0 up to 50 m

As we can see on tab 4.11, both approaches significantly boost the performance of the model. Training of the model from the scratch shows better results on every class in comparison with the first approach, but requires a big time complexity, because training on the dataset with pseudo labels is very slow.

4.4.2 Reduced dataset extended with pseudo labels

This approach is a bit unfair given that the FEP was trained on the training dataset with 5701 anotated frames, therefore pseudo-labels are much more accurately predicted. However, we had baseline in the frontview projection trained on 2142 annotated frames, thus we wanted to see use of the method pseudo-labelling, when we have a bigger ratio between pseudo-labels and annotated frames. In this case, we have the ratio 2142:9892 or 1:4.618.

The baseline on the reduced training dataset was trained for 3 time and the model under semi-supervised learning uses the same annotated frames plus pseudo labels and was trained for 1 time. The training of both models lasted 100 epochs.

Overall mean IOU		
Class	Baseline	From scratch
0 - background	0.872	**0.923**
1 - vehicles	0.681	**0.862**
2+3 - pedestrians + cyclists	0.151	**0.324**
4 - moving objects	0.069	**0.097**
5 - road	0.676	**0.710**
Mean IOU	0.490	**0.582**
Precision	0.649	**0.772**
Recall	0.579	**0.636**

Table 4.12: Semi-supervised learning results on pixels located from 0 up to 50 m with 2142 annotated frames

As we can see in tab 4.12, we achieved a big boost of performance of the model. It can be caused by the baseline model did not reach an optimal minimum to describe the smaller dataset or it has insufficient number of frames.

4.5 Discussion

It turns out that neural networks working with LiDAR point clouds have deteriorating prediction ability with increasing distance of points, because the current LiDAR sensors are not capable to capture remote areas with sufficient amount of points [31]. Our approach is to use a few models to predict pixels from the distance range 0 up to 25 metres, where the baseline have relatively strong ability to predict the data but in range 25 up to 50 its description of the data is weak, therefore a second group of the models is focused on this distance range.

The experiments in frontview show us that focusing a single model on some distance range can boost its performance in that area. The difference from the baseline is bigger when the model is focused on areas where the baseline model fails. In distance 25 up to 50 metres, we have made improvements in mean IOU by 5.4% in model using the range-CE loss with mode 3. We achieved even better results with averaging of all models, the mean IOU rose by 7.3%. We can see this approach as a wide neural network with the difference that each model have a clearly defined area which is predicted.

The baseline model in the Bird's Eye View projection has better results in comparison with frontview, but the projection itself can not absorb as many points as the frontview projection. Thus, the BEV is not a suitable candidate to represent the whole LiDAR point cloud, but the model can better work in that domain. We did not achieve such good results as in frontview by using range-CE or range-FL loss function. It is given by the fact that we focused on the surroundings of the car where the baseline itself has strong predictive ability. Thanks to achieving high mean IOU, the BEV is suitable for improving predictions in frontview after reprojection.

The naive methods *multp BEV* and *multp classes* turn out to be the best working in the fusing of two domains. The *smoothing* method failed to improve the predictive ability in any setup. It might be caused that the input to the *fusing model* is generalized thanks to using multiple models and applying another convolutional layer is not suitable in the final part of the pipeline. The final ensemble pipeline achieved the mean IOU 0.626, while the baseline model in frontview achieved the mean IOU 0.566.

The semi-supervised learning with a naive approach of pseudo-labels, which were generated by FEP, was able to significantly improve the baseline in the mean IOU by 6.5%. Further work could be focused on retraining the whole group of models in the FEP to improve each of them.

Chapter 5
Conclusion

The neural networks using data from LiDAR measurements have deteriorating prediction ability with increasing distance of the scan points. We proposed *range function*, which is masking points with weights based on the distance of the scan point. These weights are used in the loss function and the model is focusing on the specific areas. The experiments in the frontview projection have shown us that *range-function* boost predictive ability of the model inside the focused area. The improvement is more noticeable in the more distant areas, where the individual model fails due to the density of the data. In the distance range 25 metres up to 50 metres we achieved improvement by 5.4 % in the mean IOU. Usage of multiple models in the same area achieved even better results. It has improved the mean IOU by 7.3 %.

We are further fusing the outputs from frontview and bird's eye view pipelines, which are based on the ensemble of models focused on specific areas. The merge of outputs by a simple neural network called *fusing model* brings us better results in frontview. The experiments have shown us that the ensemble of the models is capable to reach better results in comparison with the frontview baseline model. The mean IOU has been improved by 10.6 %.

We also generate the pseudo-labels of the non-annotated part of the dataset to retrain the models under semi-supervised approach. The baseline trained from scratch on this extended dataset significantly improved the mean IOU by 6.5 %.

Every model in the final ensemble pipeline can be retrained by semi-supervised learning with pseudo-labelling approach for a few iterations to obtain a stronger predictor. The outputs of the final pipeline can be further merged with outputs from other sensors to improve quality of the detections.

Appendix A

Bibliography

[1] Papers with code - semantickitti benchmark (3d semantic segmentation). [online]. Available: `https://paperswithcode.com/sota/3d-semantic-segmentation-on-semantickitti`. Accessed May 06, 2021.

[2] Lidar intensity: What is it and what are it's applications?. [online]. Available: `https://geodetics.com/lidar-intensity-applications/`, Oct 2019. Accessed May 18, 2021.

[3] Eric Arazo, Diego Ortego, Paul Albert, Noel E O'Connor, and Kevin McGuinness. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020.

[4] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[5] Jason Brownlee. Ensemble learning methods for deep learning neural networks. [online]. Available: `https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/`, Aug 2019. Accessed May 28, 2021.

[6] Ming-Fang Chang, John W Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3d tracking and forecasting with rich maps. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[7] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and

Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[10] Mark Everingham, Luc Van Gool, C. K. I. Williams, J. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge, 2010.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[12] Christian Heipke and Franz Rottensteiner. Deep learning for geometric and semantic tasks in photogrammetry and remote sensing. *Geo-spatial Information Science*, 23(1):10–19, 2020.

[13] Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger, et al. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020.

[14] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[16] Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

[17] Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A. Chapman, Dongpu Cao, and Jonathan Li. Deep learning for lidar point clouds in autonomous driving: A review. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020.

[18] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.

[19] Dimitrios Marmanis, Jan D. Wegner, Silvano Galliani, K. Schindler, Mihai Datcu, and U. Stilla. Semantic segmentation of aerial images with an ensemble of cnss. In L. Halounova, K. Schindler, A. Limpouch, V. afá,

T. Pajdla, H. Mayer, S. Oude Elberink, C. Mallet, F. Rottensteiner, J. Skaloud, U. Stilla, and M. Brédif, editors, *ISPRS Congress*, volume III-3, pages 473–480. Copernicus Publications, 2016.

[20] Madali Nabil. Introduction to 3d deep learning. [online]. Available: `https://medium.com/@nabil.madali/introduction-to-3d-deep-learning-740c199b100c`, publisher=Medium, journal=Medium, year=2020, month=Aug. Accessed February 12, 2021.

[21] Mong H. Ng, Kaahan Radia, Jianfei Chen, Dequan Wang, Ionel Gog, and Joseph E. Gonzalez. Bev-seg: Bird's eye view semantic segmentation using geometry and semantic point cloud, 2020.

[22] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5105–5114, Red Hook, NY, USA, 2017. Curran Associates Inc.

[23] Joseph Rocca. Ensemble methods: bagging, boosting and stacking. [online]. Available: `https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205`, Mar 2021. Accessed May 28, 2021.

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, May 2015.

[25] Weiwei Shi, Yihong Gong, Chris Ding, Zhiheng MaXiaoyu Tao, and Nanning Zheng. Transductive semi-supervised deep learning using min-max features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[27] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 945–953, 2015.

[28] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[29] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Proceedings*

of the 30th International Conference on Neural Information Processing Systems, NIPS'16, page 550–558, Red Hook, NY, USA, 2016. Curran Associates Inc.

[30] Zining Wang, Wei Zhan, and Masayoshi Tomizuka. Fusing bird's eye view lidar point cloud and front view camera image for 3d object detection. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–6. IEEE, 2018.

[31] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[32] Yin Zhou and Oncel 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 (CVPR)*, June 2018.