Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 52 (2020) 50–55

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

5th International Conference on System-Integrated Intelligence

Self-Aware LiDAR Sensors in Autonomous Systems using a Convolutional Neural Network

Junyu Qu[a], David Barton[a,*], Philipp Gönnheimer[a], Florian Pinsker[b], Dominik Kufer[c], Jürgen Fleischer[a]

[a]*Karlsruhe Institute of Technology, wbk Institute of Production Science, Kaiserstraße 12, 76131 Karlsruhe, Germany*
[b]*T-Systems, Dachauer Straße 651, 80995 Munich, Germany*
[c]*IAV, Weimarer Straße 10, 80807 Munich, Germany*

* Corresponding author. Tel.: +49-1523-950-2565 ; fax: +49 721 608-45005. *E-mail address:* david.barton@kit.edu

Abstract

Autonomous systems, as found in autonomous driving and highly automated production systems, require an increased reliability in order to achieve their high economic potential. Self-aware sensors are a key component in highly reliable autonomous systems. In this paper we highlight a proof of concept (PoC) of a deep learning method that enables a LiDAR (Light detection and ranging) sensor to detect functional impairment. More specifically, a deep convolutional neural network (CNN) is developed and trained with labelled LiDAR data in the form of point clouds to classify the degree of impairment of its functionality. The results are statistically significant and can be regarded as a general classifier for objects within LiDAR data, applied to selected cases of sensor impairment. In detecting impairment and evaluating the correctness of the captured data, the sensor gains a basic form of self-awareness. The presented methods and insights pave the way for improved safety of autonomous systems by the means of more sophisticated "self-aware" neural networks.

Keywords: autonomous systems; machine learning; reliability

1. Introduction

Autonomous systems are in general not maintained by an operator and therefore self-awareness of functionality is not only a way to bridge the gap opened by the lack of testing functionality but also a further step towards self-aware robots, as more and more functions of the robot are checked by the robot itself. In animals, a simple form of mirror-self-recognition (MSR) is demonstrated when animals realise they have been marked when they see their reflection in a mirror [1]. In this contribution a similar level of self-awareness is implemented concerning the state of a robot's sensors.

3D sensor technology is currently being developed for driver assistance systems and autonomous driving. LiDARs are popular examples of 3D sensors with specific strengths such as their range (distances from centimetres up to hundreds of metres) and their high angular resolution compared to radar. Shortcomings of a single frequency laser scanning method are particularly resolution and effective colour-blindness compared to cameras, these can be partly overcome by using convolutional neural networks (CNN) for object detection and localization as well as combining LiDAR with other sensors such as cameras [2]. Further applications of LiDAR technology include assisting a visually impaired person [3], and detecting

10.1016/j.promfg.2020.11.010

archaeological objects [4]. This paper describes a CNN that is trained to determine whether a LiDAR sensor is impaired by dirt or other obstructions, and to which degree it is impaired, thereby enabling a higher level of autonomy and enhancing safety in LiDAR applications.

2. State of the art

In the following relevant methods from existing work are presented, focusing on preprocessing LiDAR data, classification using CNN and the detection of sensor impairment.

2.1. Preprocessing LiDAR data

The output data from LiDAR sensors consists of point clouds. The most common method for preprocessing these is by the means of voxel grids. Discretised point clouds in square grids and binary coding for 3D voxel grids are considered in [5]. The processed point cloud can be drawn as a 4D data array. The dimensions are length, height, width and the number of channels. A binary channel can be used to determine whether points are inside the grid. Then the mechanism is extended from 2D CNN to 3D for the considered point cloud.

[6] introduces a voxel coding layer that could learn the unified features directly from LIDAR point clouds. The first step is partition into voxels. The number of points in each voxel varies significantly, some voxels are empty. The contents of a voxel can be described as $V = \{p_i = [x_i, y_i, z_i, r_i]^T \in R^4\}_{i=1\dots t}$, where p_i consists of the coordinates x_i, y_i, z_i for a point and the received reflection r_i for this point. The voxel is considered non-empty if the number of points t exceeds a threshold T. First, the local mean value is calculated as the centre of gravity of all points in V and is designated as $(\vartheta_x, \vartheta_y, \vartheta_z)$. Then each point p_i is extended with the relative offset $V_{in} = \{\widehat{p_i} = [x_i,\ y_i,\ z_i,\ r_i,\ x_i - \vartheta_x,\ y_i - \vartheta_y,\ z_i - \vartheta_z]^T \in R^7\}_{i=1\dots t}$.

Next, each point is transformed by a Fully-Connected Network (FCN) into a feature space for later processes. [7, 8] encoded each non-empty voxel with 6 statistical quantities that derived from all points contained in the voxel. The points in the point cloud are defined by point positions and reflection values. First, the 3D space is divided into a grid with a fixed resolution, like an image, and each occupied cell is converted into a fixed-dimensional feature vector. The normal feature vector contains a binary occupancy value, the mean and variance of the reflection values, and three form factors. Thus the total voxel has 6 statistical quantities. Cells that are empty are not stored. The 3D voxel grids can be projected onto a plane to get a 2D point map [9].

2.2. Classification using CNN

Many image classification competitions have been won using deep learning architectures based on a CNN. [10] also confirms that a CNN is well suited to perform segmentation tasks with LiDAR data. This section aims to give an overview of some architectures that may provide the basis for classifying 3-dimensional data from LiDAR sensors for impairment detection.

AlexNet [11] achieved a leading result in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2010 (error rates in Top-1 and Top-5). It consists of 65,000 neurons and 60 million parameters. The AlexNet algorithm was originally written with CUDA and ran on GPUs. The whole process is performed using five convolutional layers and three fully connected layers. For down-sampling, max-pooling is used to reduce the number of neurons in the following layers. The image size of the input for AlexNet is $227 * 227 * 3$ and the size of the filter is $11 * 11 * 3$. With step size 4, the size of the output for this layer is $55 * 55 * 96$. To avoid overfitting, a drop-out within the first Fully-Connected layer was used in addition to the data expansion of the input data.

VGGNet [12] is also a very successful CNN architecture. Compared to AlexNet, VGGNet contains more layers and also delivered a good result in ILSVRC 2012 (error rates in Top-1 and Top-5). It consists of about 140 million parameters. There are 13 convolutional layers, 5 max-pooling layers and 3 fully connected layers. Apart from the depth, the other major difference to AlexNet is the structure of the convolutional layer. The size of the filter in the convolutional layer for VGGNet is $3 * 3 * 3$ instead of $11 * 11 * 3$ in AlexNet.

Another CNN architecture that has been successfully applied to image classification is ResNet [13]. The so-called Residual Neural Network (ResNet) is a new idea with "jump connections", shown successfully at ILSVRC 2015. In contrast to other architectures, the output for an input x is not $f(x)$ but $f(x) + x$, where $f(x)$ is the output of convolutional layers. First VGGNet is extended from 19 layers to 34 layers, then this structure is used to jump connections between the layers. This modification helps to solve the problem of vanishing gradients.

2.3. Detection of sensor impairment

In [14], the influence of rain and fog on the performance of data from LiDAR sensors is assessed. The analysis shows a significant impairment of object detection, with a significant reduction in the number of points per object. This confirms the need for an approach to detecting such adverse conditions. Additionally, support vector machines (SVM) and k-Nearest-Neighbor classifiers (kNN) are applied to classify the weather conditions, achieving a high accuracy. Thus an awareness of impairment due to weather conditions was implemented. However no work was found where an awareness of the condition of the sensor itself, due to dirt or other obstructions, is implemented.

3. Approach

This contribution describes a solution to detect sensor impairment in LiDAR. It includes an approach for

preprocessing LiDAR data and two classification tasks based on the preprocessed data. The first classification task consists in determining whether the sensor is impaired (binary classification). In the second task (multi-classification), the impaired condition is further divided into two levels of impairment.

3.1. Data collection and preparation

Point clouds were generated using the Velodyne VLP-16 LiDAR. To generate a data stream for training we recorded five distinctive scenes. Three scenes were recorded indoors at distinctive places. The other two scenes were recorded outdoors. Each scene was divided into three cases: clear vision, lightly impaired vision and heavily impaired vision. To simulate different states of the LiDAR, markers of two different sizes where placed on the sensor. Each point cloud dataset is irregular, and the number of points varies between frames. Based on these characteristics, we preprocess the point cloud datasets with regularization. The first step is to convert the whole spatial area of a point cloud into $n * n * n$ boxes (voxels), then the number of points in each voxel and the sum of the intensity of all points in each voxel are calculated. The $n * n * n$ voxels with one channel, containing the sum of the intensity of all points in each voxel, can be processed as a regular input by three-dimensional convolutional neural networks.

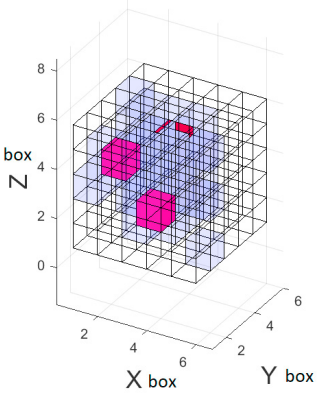

Fig. 1. Visualisation of preprocessed voxels.

An example of a point cloud converted to a resolution of $5 * 5 * 5$ voxels is visualised in Figure 1. The higher the intensity of points within a voxel, the more the colour is shifted towards red. Voxels without a point are transparent. In the following section we shortly present the architectures used for classification.

3.2. Network Architecture for binary classification

The CNN binary classification algorithm considered here consists of five layers: a convolutional layer, a mean pooling layer, a fully connected layer and two activation function layers. The arrangement is shown in Figure 2, with an input resolution of $5 * 5 * 5$ voxels. In this scenario the first layer contains a filter ω_{L1} with the size $2 * 2 * 2$ and bias b_{L1} with the size $4 * 4 * 4$ (the bias has the same size as the output). After the first layer the size of the output is $4 * 4 * 4$. The fourth layer contains a filter ω_{L4} with the size $2 * 2 * 2$ and bias b_{L4} with the size 1.

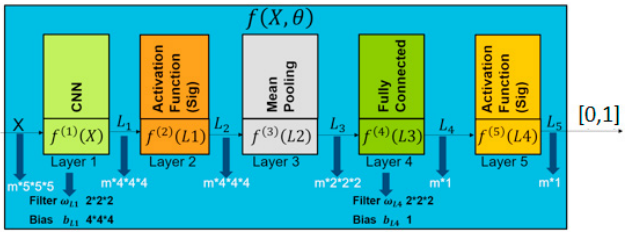

Fig. 2. Layer structure of CNN for binary classification.

The algorithm makes a final decision whether the point cloud was recorded by an impaired or clear LiDAR system. This is computed by the final sigmoid activation function as a probability in between 0 and 1. The vector θ includes optimisation parameters for the convolutional layer and fully connected layer. For the convolutional layer filters ω_{L1} and bias b_{L1} were used and for the fully connected layer filters ω_{L4} and bias b_{L4} were used.

3.3. Network Architecture for multi-classification

The algorithm for multi-classification consists of four layers which are a convolutional layer, an activation function layer, a mean pooling layer, and a softmax layer, as shown in Figure 3.

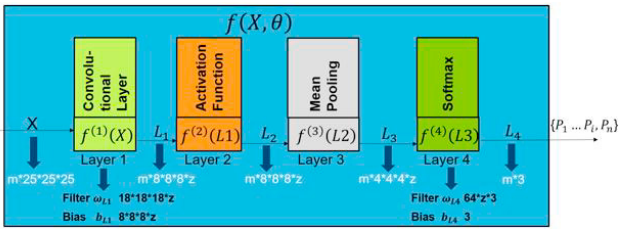

Fig. 3. Layer structure of CNN for multi classification.

In this particular example the input for the neural network has a resolution of $25 * 25 * 25$ voxels. In the first layer, ω_{L1} consists of z filters, which each have a size of $18 * 18 * 18$, and a bias b_{L1} with the size $8 * 8 * 8$. After the first layer the size of the output is $8 * 8 * 8 * z$. Through the pooling layer, the size is reduced to $4 * 4 * 4 * z$. The fourth layer has the filter ω_{L4} with the size $64 * z * 3$ and bias b_{L4} with the size 3. Then the output contains the probabilities for all three classes.

4. Result and discussion

In the following sections the results in both classification tasks are presented for different input formats (number of voxels). For the multi-classification task, the input format and other parameters are tuned to improve performance. Subsequently the achieved accuracies and computational cost are briefly discussed.

4.1. Validation tests for binary classification

The validation tests for binary classification are divided into two phases: testing the algorithm and improving the algorithm using the data. In the first phase a smaller set of data is used. Based on encouraging preliminary results, all available data from all scenes is used to improve the algorithm in the second phase.

In the first phase, the computational cost for preprocessing was minimised by considering only one scene. The point clouds were divided into two classes: "impaired" and "clear". 100 point clouds were randomly selected from each class. 160 point clouds were used for training and 40 point clouds for testing. The data was preprocessed for two different input resolutions, $5 * 5 * 5$ voxels and $15 * 15 * 15$ voxels, and separate models were trained with each format. The number of iterations here was 1000 for both cases. In the testing data, the average final output (probability of sensor impairment) was 0.9989 for the 20 "impaired" point clouds and 2.2182e-04 for the 20 "clear" point clouds. This comes very close to the ideal average outputs, respectively 1 and 0. Training lasted only approximately 1 minute for $5 * 5 * 5$ voxels and almost 10 minutes for $15 * 15 * 15$ voxels on a standard CPU. However, the higher resolution delivered better predictions.

For the second phase, preprocessing was automated using Python. In each of the 5 scenes, 10 point clouds were randomly selected from each of the three classes (clear, lightly impaired, heavily impaired), resulting in 150 point clouds in total. These were split into 120 point clouds for training (8 from each of the 15 cases) and 30 point clouds for testing (2 from each case). In this phase three different resolutions were used and compared: $5 * 5 * 5$ voxels, $9 * 9 * 9$ voxels and $15 * 15 * 15$ voxels. The number of iterations was set to 10000 for all the considered cases. Once again the intuition was confirmed that a higher resolution (higher number of voxels) leads to better predictions while increasing the time needed for training.

4.2. Validation tests and tuning for multi classification

Next we consider multi-classification of the degree of impairment, with three classes: clear, lightly impaired and heavily impaired. First different input resolutions were tested. For each of the 5 scenes, 100 point clouds were randomly selected from each class to obtain a total of 1500 point clouds. These were divided into 1200 point clouds for training and 300 point clouds for testing (respectively 80 and 20 from each case). We first considered five different input resolutions, as shown in Table 1. The number of mini-batches for the training is 12 and the number of epochs is set to 10. The average accuracies during training are compared, as shown in Figure 4.

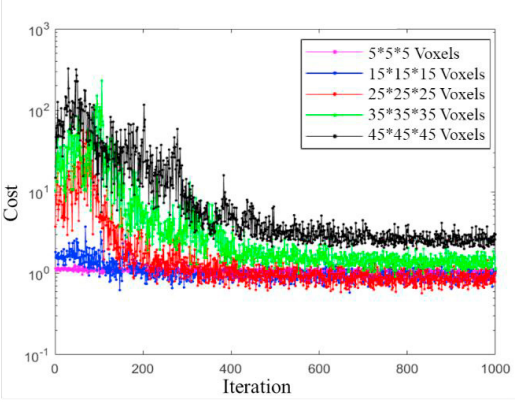

Fig. 4. Cost for different numbers of voxels.

Table 1. Accuracy rate for different numbers of input resolutions.

Voxels	5x5x5	15x15x15	25x25x25	35x35x35	45x45x45
Accuracy [%]	49.3	69.3	72.3	71.3	70.6

It becomes apparent that accuracy peaks at $25 * 25 * 25$ voxels. The training phase is especially costly with high resolutions and the low resolutions may not lead to the loss of too much information. Based on this result, an input resolution of $25 * 25 * 25$ is used for all subsequent tests, and all the available data (1000 point clouds for each class, in each scene) was preprocessed for this resolution, thus including a total of 15000 point clouds. The point clouds were split in the same proportions as above.

In the next test, all parameters except the minibatch size were kept the same as above. To determine the optimal minibatch size, training was performed with five minibatch sizes (60, 80, 100, 120 and 240), as shown in Figure 5 and Table 2.

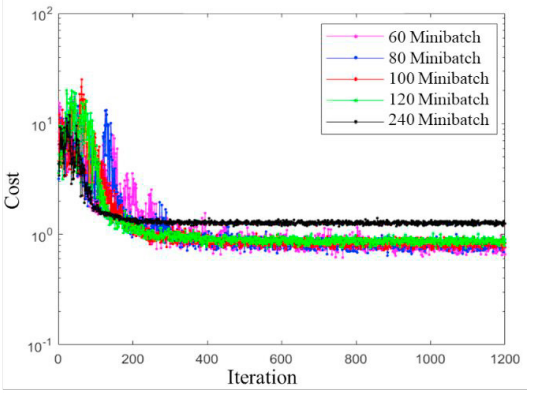

Fig. 5. Cost for different minibatch sizes.

Table 2. Accuracy rate for different minibatch sizes.

Minibatch size	60	80	100	120	240
Accuracy [%]	79.3	79.7	80.3	80.2	75.7

After selecting a minibatch size of 100, parameters within the model were tuned. The regularization method invoking the L2 norm, i.e. L2 Regularization, was used in our examples which has an additional parameter λ. To determine the adjusted λ for L2 Regularization, eight different λ were used in training, as shown in Figure 6 and Table 3. The best accuracy was obtained for $\lambda = 0.04$, therefore this setting was chosen.

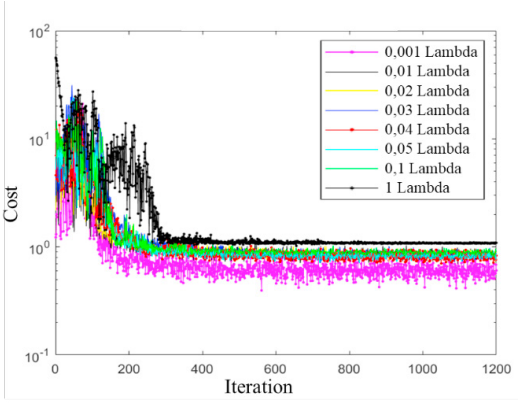

Fig. 6. Cost with different L2 regularization parameters.

Table 3. Accuracy rate with different L2 regularization parameters.

λ	0.001	0.01	0.02	0.03	0.04	0.05	0.1	1
Accuracy [%]	79.3	79.7	80.5	80.7	81.53	81.3	80.7	70.7

The learning rate, the size of the filter and the number of filters in the convolutional layer also influence accuracy. It was observed that the size of the filter and the number of filters have a significant effect on the accuracy of the prediction. Previous experiments were all performed with two $2 * 2 * 2$ filters. At first the number of filters was fixed as two and only the size of the filters was varied, as shown in Figure 7 and Table 4.

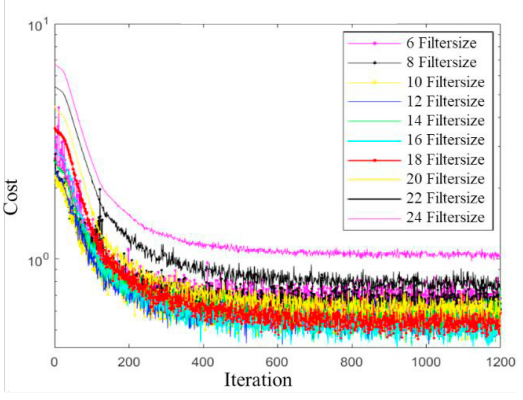

Fig. 7. Cost for different filter sizes.

Table 4. Accuracy rate for different filter sizes.

Filter size	6	8	10	12	14	16	18	20	22	24
Accuracy [%]	82.8	83.1	83.2	85.6	86.8	87.2	88	86.5	83.5	91.7

After the size of filters was set to 18, the number of filters was varied. To quickly determine the best number of filters, a sequence of increasing powers of two was used for the number of filters. Based on the results shown in Figure 8 and Table 5, the parameter was set to 64.

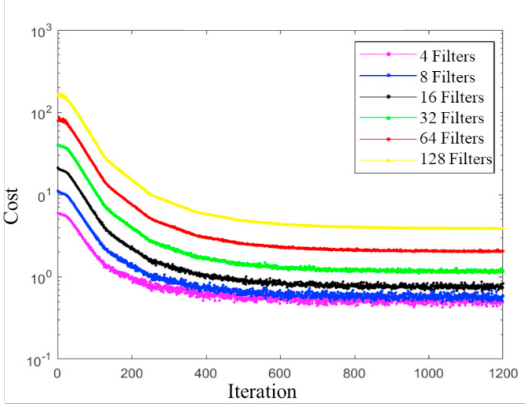

Fig. 8. Cost for different numbers of filters.

Table 5. Accuracy rate for different numbers of filters.

Number of filters	4	8	16	32	64	128
Accuracy [%]	89.9	90.3	90.7	91.1	91.7	91.1

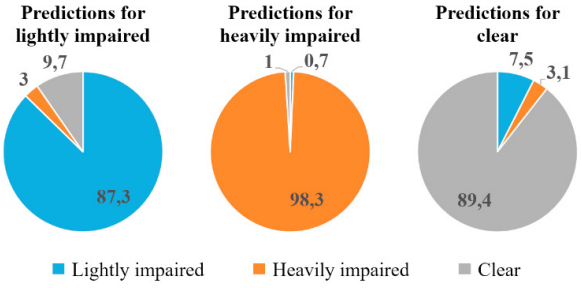

Fig. 9. Prediction for multi classification with 64 filters with size 18 (25*25*25 voxels) in percent

The average accuracy of this model for all point clouds of the test set is 91.67%. The detailed results for each class are shown in Figure 9. The highest accuracy is achieved when detecting heavy impairment, reaching 98.2%. A summary of the successive accuracy improvements through parameter tuning is given in Table 6.

Table 6. Summary of accuracy improvements through parameter tuning.

Tuning Method	Accuracy
Baseline (simple CNN with 25 voxels)	72.3%
Ten times more data and minibatch 100	80.33%
Regularization method L2 with $\lambda = 0.04$	81.53%
Filter size adjusted to $18 * 18 * 18$	88.03%
Number of filters increased to 64	91.67%

4.3. Discussion

The procedure of developing this algorithm can be divided in three parts which were recording LiDAR data, setting up the preprocessing of LiDAR data and finding a working deep learning algorithm. The obstruction of the LiDAR was created artificially to ensure a defined and reproducible state of impairment. This differs from real conditions where dirt is randomly distributed on the LIDAR system and the size of dirt is not known, however this made it much easier to label the data. The voxel based method used for preprocessing made it possible to feed the data to a CNN while limiting loss of information. In the classification model Cross-entropy Loss (CEL) was chosen as a loss function, which in combination with the sigmoid activation function simplified the calculation of gradients. The backpropagation algorithm was faster and more accurate than other methods we considered at the beginning. The developed model was simple enough to train on a regular office computer. The validation results of the algorithm for multi classification achieved a satisfactory accuracy of 91.7%, with a much higher rate of 98.2% for large obstructions. It was shown that the algorithm works with data collected from different scenes, both indoors and outdoors.

5. Conclusion

This contribution shows that a classifier based on a CNN can be used to detect obstructions on a LiDAR sensor, thus providing an example for a basic level of self-awareness in autonomous systems. This type of self-awareness is critical in reducing the need for monitoring quality of sensing either manually or with an additional sensor. Future work may consist in testing and improving the algorithm developed here for a wider range of conditions, including a greater variety of obstructions. Additionally the approach could be tested on other sensor types in harsh environments such as inside manufacturing equipment.

References

[1] Morrison R, Reiss D. Precocious development of self-awareness in dolphins. PLoS One 2018.

[2] Ku J, Mozifian M, Lee J, Harakeh A, Waslander SL. Joint 3d proposal generation and object detection from view aggregation. In: International Conference on Intelligent Robots and Systems 2018.

[3] Ton C, Omar A, Szedenko V, Tran V, Aftab A, Perla F, Bernstein M, Yang Y. LIDAR Assist spatial sensing for the visually impaired and performance analysis. In: IEEE Transactions on Neural Systems and Rehabilitation Engineering; 2018. p. 1727-1734.

[4] Verschoof-van der Vaart WB, Lambers K. Learning to Look at LiDAR: The use of R-CNN in the automated detection of archaeological objects in LiDAR data from the Netherlands. In: Journal of Computer Applications; 2019.

[5] Lagarias JC, Reeds JA, Wright MH, Wright PE. Convergence properties of the Nelder--Mead simplex method in low dimensions. In: SIAM Journal on optimization; 1998. p. 112-147.

[6] Ruder S. An overview of gradient descent optimization algorithms. arXiv preprint 2016.

[7] Li B. 3d fully convolutional network for vehicle detection in point cloud. In: International Conference on Intelligent Robots and Systems (IROS); 2017.

[8] Zhou Y, Oncel T. Voxelnet: End-to-end learning for point cloud based 3d object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2018.

[9] Engelcke M, Rao D, Wang DZ, Tong CH, Posner I. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In: IEEE International Conference on Robotics and Automation (ICRA); 2017.

[10] Velas M, Spanel M, Hradis M, Herout A. Cnn for very fast ground segmentation in velodyne lidar data. In: IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC); 2018.

[11] Wang DZ, Posner I. Voting for Voting in Online Point Cloud Object Detection. In: Robotics: Science and Systems; 2015.

[12] Li B, Zhang T, Xia T. Vehicle detection from 3d lidar using fully convolutional network. arXiv preprint 2016.

[13] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 2012.

[14] Heinzler R, Schindler P, Seekircher J, Ritter W, Stork W. Weather Influence and Classification with Automotive Lidar Sensors. In: IEEE Intelligent Vehicles Symposium; 2019.