Proceedings of the 10th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'24)
Barcelona, Spain - August 19-21, 2024
Paper No. MVML 117
DOI: 10.11159/mvml24.117

Paintedpillars: Efficient 3D Object Detection Using Only Statistical Processing To Compute Pillar Features with Explicit Class Probability Distributions

Masayuki Miyama[1]
[1]Kanazawa University
Kakumamachi, Kanazawa, Ishikawa, Japan
miyama@se.kanazawa-u.ac.jp

Abstract **-** In autonomous driving technology, high-speed and highly accurate three-dimensional object detection is necessary to accurately grasp the surrounding environment, and it is important to integrate information from multiple sensors. Conventional methods have performed 3D object detection by combining distance information from LiDAR (Light Detection And Ranging) and texture information from camera images, but there were issues with the amount of calculation and accuracy. Therefore, we propose a new method called PaintedPillars that explicitly assigns a class probability distribution to each pillar. Here, a pillar is a column-shaped space extending vertically into each division of a horizontal plane divided into a grid in a three-dimensional space. The proposed method, through statistical processing for each pillar, calculates class probability distributions from the semantic segmentation results of camera images, and obtains the spatial distribution of points from LiDAR data. In other words, the feature values for each pillar required for three-dimensional object detection, that is, object type discrimination and bounding box estimation, are obtained using only statistical processing. The amount of calculation is proportional to the product of the number of points and the number of input channels, and does not depend on the number of output channels, so it is less than the conventional method using neural networks. Furthermore, the problem of reduced accuracy due to padding does not occur. Compared to the conventional PointPainting, experiments showed that the detection accuracy of the proposed method was 1 % higher, and the execution time and GPU memory usage were reduced to 33 % and 60 %, respectively, demonstrating its effectiveness.

Keywords**:** autonomous driving, 3D object detection, sensor fusion, semantic segmentation, statistical processing

1. Introduction

Accurately recognizing the surrounding environment is extremely important in autonomous driving technology. Three-dimensional object detection is one of the advanced recognition technologies. 3D object detection in autonomous driving requires both accuracy and execution speed. For this reason, the development of high-speed and highly accurate 3D object detection models is actively underway.

Autonomous driving uses a variety of sensors, such as LiDAR and cameras, to understand the situation around the vehicle. There are limits to the information that can be obtained from a single sensor. LiDAR provides distance information, but not texture information. On the other hand, a camera captures texture information, but not distance information. In recent years, approaches that integrate sensor information have attracted attention to compensate for these limitations. This integration is expected to improve the recognition accuracy of self-driving cars by combining information obtained from multiple sensors, resulting in safer and more efficient driving.

As an efficient 3D object detection method, PointPainting combined with PointPillars is known [1,2]. In the conventional method, PointPainting is used to associate the semantic segmentation results of the camera image with the point cloud data obtained from LiDAR. Then, using PointPillars, the point cloud data is divided into pillars, a feature vector is calculated for each pillar, a feature map is created, and 3D object detection is performed. Here, a pillar is a columnar space extending in the z-axis direction in each section obtained by dividing the x-y plane into a grid in a three-dimensional space whose z-axis is the height direction. A neural network is used to calculate the feature vector for each pillar. The amount of calculation increases in proportion to the product of the number of points, input channels, and output channels. Additionally, in order to speed up calculations using GPUs, it is necessary to keep the number of points for each pillar constant. If the actual number of points in the pillar is small, padding is performed, and if there is a large number, the points are thinned out. Excessive padding or thinning has a negative effect on detection accuracy. Therefore, the conventional method has problems with the amount of calculation and detection accuracy.

Therefore, in this study, we perform 3D object detection by calculating a feature vector containing an explicit class probability distribution for each pillar using statistical processing. Figure 1 shows the processing flow of the proposed method. The flow of the first half is the same as PointPainting. The flow in the second half is the same as PointPillars. The difference from the conventional method is that the feature vector of each pillar is extracted only by statistical processing. The proposed method, through statistical processing for each pillar, calculates class probability distributions from the semantic segmentation results of camera images, and obtains the spatial point distribution from LiDAR data. This provides pillar feature vectors for 3D object detection, that is, object type discrimination and bounding box estimation. Statistical processing can be performed at high speed using GPU parallel tensor operations. The amount of calculation is proportional to the product of the number of points and input channels, and does not depend on the number of output channels, so it is less than the conventional neural network. Also, there are no padding problems. Experiments showed that the proposed method has 1 % higher detection accuracy than the conventional method, and the execution time and GPU memory usage are reduced to 33 % and 60 %, respectively, indicating high accuracy and efficiency.

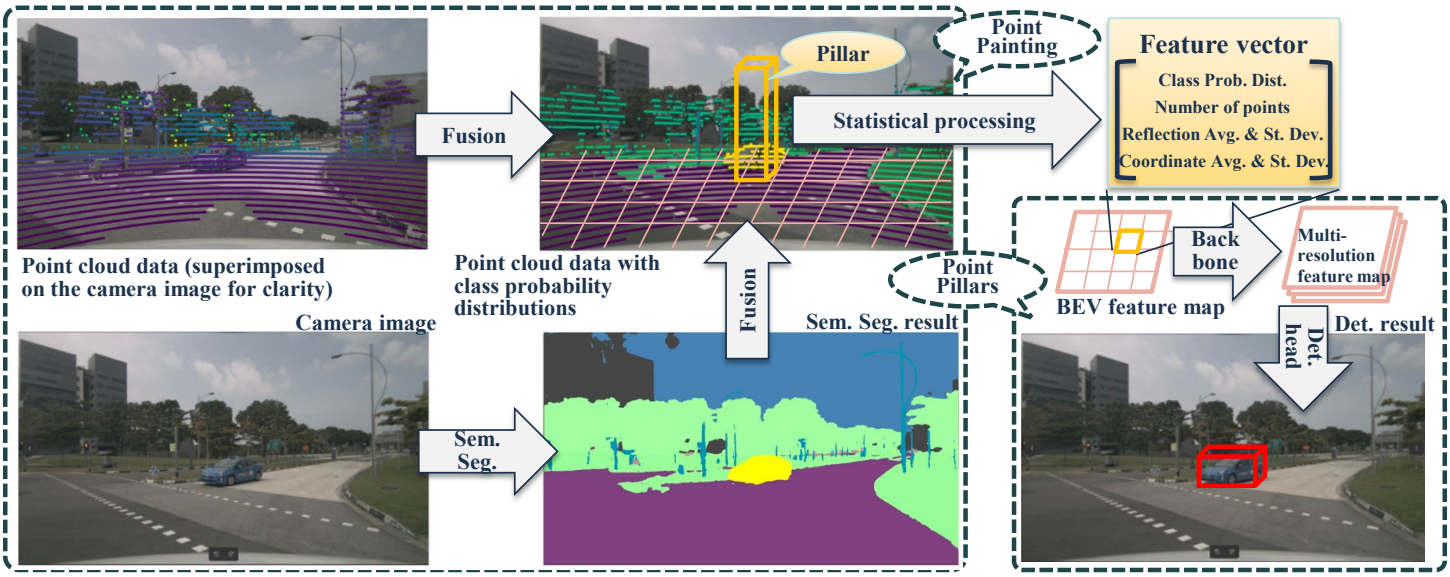

Fig. 1: Processing flow of the proposed PaintedPillars.

The remainder of this paper is structured as follows. Section 2 summarizes related research, Section 3 explains the proposed method, Section 4 discusses the experimental method and results, and Section 5 concludes the paper.

2. Related Work

Semantic segmentation is an AI task that assigns a class to each pixel in a camera image. DeepLabV3 is a semantic segmentation model that uses Atrous convolution, which performs convolution using a kernel with gaps between pixels [3]. This makes it possible to obtain a wide range of information while maintaining the resolution of the output feature map, improving accuracy.

Three-dimensional object detection is the task of surrounding cars and pedestrians included in point cloud data with a 3D rectangle called a bounding box, and assigning a class to each enclosed object [1,2,4]. Point cloud data is acquired using LiDAR. LiDAR is a sensor that measures the time it takes to detect the reflected light of an emitted laser and obtains the distance to an object.

Three-dimensional point cloud representations include range view and BEV (Bird Eye View). Range view is a two-dimensional map created by projecting the point where the laser hits to the intersection of a cylinder centered on the LiDAR and the reflected light [4]. This can accurately represent the distance and shape of the object, but the objects

may overlap on the map. BEV is a two-dimensional map that projects a three-dimensional point cloud vertically from the sky to the ground. This does not cause objects to overlap, but it is difficult to express the height of a point from the ground. Although both are used as input for 3D object feature extraction, non-overlapping BEVs are often used as input for the final detection head.

PointPillars is a 3D object detection model that processes point clouds by dividing them into pillars [2]. PointPillars divides a point cloud into pillars, then calculates a feature vector for each pillar using a neural network (PillarFeatureNetwork) rather than statistical processing, and arranges them on a plane to create a BEV-format feature map. PillarFeatureNetwork repeatedly performs point-by-point linear transformation, batch normalization, ReLU activation, and pooling. By concatenating the global features of the pillar obtained through pooling in the previous iteration to the features of each point, and performing the next iteration, each point acquires relationships with other points. The subsequent processing of PointPillars is the same as the process of obtaining detection results from the BEV feature map in the latter half of Figure 1. Two-dimensional convolution is performed on the BEV feature map using the backbone to create a multi-resolution feature map. The detection head then estimates the bounding box around the object by convolution.

Sensor fusion is the process of integrating information from multiple sensors, which improves recognition accuracy. For example, a pedestrian and a pole may look the same in point cloud data, but can be clearly distinguished in camera images. PointPainting is known as a sensor fusion method [1]. The process to obtain the point cloud data with the class probability distribution added in the first half of Figure 1 is the same as PointPainting. PointPainting first performs semantic segmentation on a camera image. Next, each point of the point cloud data is projected onto the camera image, a class probability distribution is obtained from the semantic segmentation result of the corresponding pixel, and a class probability distribution is assigned to that point. Then, three-dimensional object detection is performed using PointPillars, etc., using point cloud data in which the class probability distribution is added to the feature of each point. Class probability distributions are useful in determining object types, but PointPillars uses a neural network to extract features, so the feature vector of PointPainting in combination with PointPillars does not explicitly indicate the class probability distribution of the pillar.

3. Proposed Method

PointPillars uses GPU tensor parallel operations to quickly calculate feature vectors by keeping the number of points within a pillar constant. For pillars with more points than the upper limit, information is lost due to downsampling. For pillars with fewer points than the upper limit, padding reduces accuracy. Additionally, increasing the upper limit increases the amount of calculation and memory usage due to the increase in tensor size.

Figure 1 shows the processing flow of the proposed method. The first half of the process to obtain point cloud data with class probability distribution added is the same as PointPainting. The second half of the process from obtaining the detection results from the BEV feature map is the same as PointPillars. The difference between our method and PointPainting combined with PointPillars is that feature vectors are calculated using statistical processing rather than neural networks. The proposed method uses only statistical processing to calculate pillar feature vectors that are effective for detecting three-dimensional objects, that is, discriminating object types and estimating bounding boxes. The proposed method explicitly assigns a class probability distribution, which is useful for the discrimination, to a pillar by taking the average of all points within the pillar. Therefore, we call the proposed method PaintedPillars. To infer the position, size, and direction of the bounding box, we add the number of points and the average and standard deviation of point coordinates on the x, y, and z axes to the pillar feature vector. Furthermore, the average and standard deviation of the reflection intensity are also given to the vector as elements. It is difficult to extract features for determining object types using only statistical processing of three-dimensional coordinates and reflection intensity obtained from LiDAR point data. However, by combining this with the semantic segmentation results of camera images, it becomes possible to extract features for 3D object detection using only statistical processing.

Feature vector calculation using statistical processing is proportional to the product of the number of points and the number of input channels and does not depend on the number of output channels, so the amount of calculation and memory usage is smaller than that of conventional neural networks. The average and standard deviation can be calculated quickly using GPU parallel tensor operations. In addition, calculations can be made without being affected by padding for pillars with a small number of actual points, so there is no loss of accuracy.

4. Experiment

4.1. Setup

The proposed method PaintedPillars (abbreviated as papi) was compared with the conventional method. The conventional method was PointPillars (popi) and PointPainting combined with PointPillars (hereinafter simply (popa)). DeepLabV3 trained on Cityscapes was used for semantic segmentation of camera images [5]. The vertical and horizontal sizes of the pillars were both 20 cm. The upper limit of pillar points was set to 16/32/64/128. Pillars that did have the maximum number of points were zero padded.

We used nuScenes as the dataset. nuScenes is a large-scale public dataset for autonomous driving, consisting of LiDAR, camera, and radar sensor data [6]. The number of scenes used for learning, validation, and testing was 28130/ 4775/1244, respectively. The original validation data was used separately for validation and testing. There were four classes: car, bike, other_vehicle, and pedestrian. The detection range was set to 50 m front, back, left, and right.

The number of learning epochs was 35, the optimization method was Adam, the initial value of the learning rate was 0.001, and the learning rate was set to 1/10 every 10 epochs. As loss functions, we used Focal for the class probability distribution, SmoothL1 for the center coordinates and size of the bounding box, and Cross Entropy for the orientation of the bounding box. Each loss function was dynamically weighted using the method of reference [7], that uses the following equation weighted by the exponential function to calculate the loss followed by backpropagation to automatically calculate the parameter *w*.

$$loss = e^{-w_A}\, loss_A + e^{-w_B}\, loss_B + w_A + w_B \tag{1}$$

Open3D-MLwas used to implement the program [8]. PyTorch was used as a machine learning platform [9]. Statistical processing using tensor operations in PyTorch computed the pillar feature vectors. The CPU of the computer used was Intel Core i9-10920X CPU@3.50 GHz, the amount of memory was 128 GB, and the GPU was NVIDIA RTX A4000 (memory 16 GB).

mAP (mean average precision) was used as an evaluation index. mAP is expressed by the following formula.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{2}$$

$$AP = \sum_{i=1}^{M} (Recall_i - Recall_{i-1})\, Precision_i \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

Here, N is the total number of classes and M is the total number of intervals, TP is the number of True Positive samples, FP is the number of False Positive samples, and FN is the number of False Negative samples. The confidence threshold for determining that an object has been detected was set to 0.05.

IoU (Intersection over Union) was used to map correct answers and inferences. IoU is expressed by the following formula.

$$IoU = \frac{ground\ truth\ \cap inference}{ground\ truth\ \cup inference} \tag{6}$$

Here, *ground truth* is the area (volume) of the correct bounding box, and *inference* is the area (volume) of the inference bounding box. There are two methods for calculating IoU: BEV and 3D. BEV calculates the area of the bottom of the bounding box, and 3D calculates the volume. The IoU threshold was set to 0.5.

4.2. Results

Table 1 shows the mAP of each method when the maximum number of points in a pillar is 128. It can be seen that for both BEV and 3D, the proposed method PaintedPillars has a higher overall accuracy than the conventional method PointPainting by 1%, and more than 10% than PointPillars. Regarding pedestrians and bikes, PaintdPillars and PointPainting have higher accuracy than PointPillars, indicating that sensor fusion is effective for small objects.

Table 1: mAP (%) of each method when 128 points in a pillar (papi: PaintedPillars, popa: PointPainting, popi: PointPillars)..

	BEV			3D		
	papi	popa	popi	papi	popa	popi
pedestrian	30.56	**32.03**	25.52	**27.33**	27.28	11.42
bike	**64.21**	56.30	44.04	**52.89**	52.40	22.76
car	**88.79**	88.14	84.97	86.96	**87.23**	84.31
other_vehicle	61.11	**63.30**	51.38	**55.76**	51.18	28.16
overall	**61.17**	59.94	51.48	**55.73**	54.52	36.66

Table 2 shows the overall mAP when the maximum number of points in a pillar are 16/32/64/128. For BEV, the accuracy of PaintedPillars with 32 points was the highest value, and the accuracy with 128 points was almost equal to the highest value. In the case of 3D, PaintedPillars had the highest accuracy with 128 points. For both BEV and 3D, the accuracy of PaintedPillars with 128 points was higher than any of the other methods. The accuracy of PaintedPillars and PointPainting was at or near the best when the number of points was 128. In the method using sensor fusion, there is a tendency for the accuracy to increase as the number of points increases. The accuracy of PointPainting without sensor fusion decreased as the number of points increased, and was lowest at 128 points. This is considered to be the effect of excessive zero padding.

Table 2: mAP (%) of each method when 16/32/64/128) points in a pillar (papi: PaintedPillars, popa: PointPainting, popi: PointPillars).

	BEV			3D		
	papi	popa	popi	papi	popa	popi
16	59.58	**60.16**	55.72	48.65	49.67	**52.21**
32	**61.22**	60.76	55.23	**53.68**	50.00	52.10
64	56.74	**59.54**	53.86	49.87	46.95	**51.32**
128	**61.17**	59.94	51.48	**55.73**	54.52	36.66

Figures 2 represents the inference time and GPU memory usage of each method on the test dataset. In both graphs, the horizontal axis is the maximum number of points in a pillar. When the number of points was 128, the inference time of PaintedPillars was 33 % of PointPainting and 38 % of PointPillars. GPU memory usage was 60 % and 62 % of each conventional method. It can be seen that as the number of points increases, the inference time of PointPillars and PointPainting increases rapidly, and the amount of GPU memory used also increases rapidly. Compared to these, PaintedPillars' inference time and GPU memory usage increase slowly. This is thought to be due to the difference in the method of calculating feature vectors for pillars.

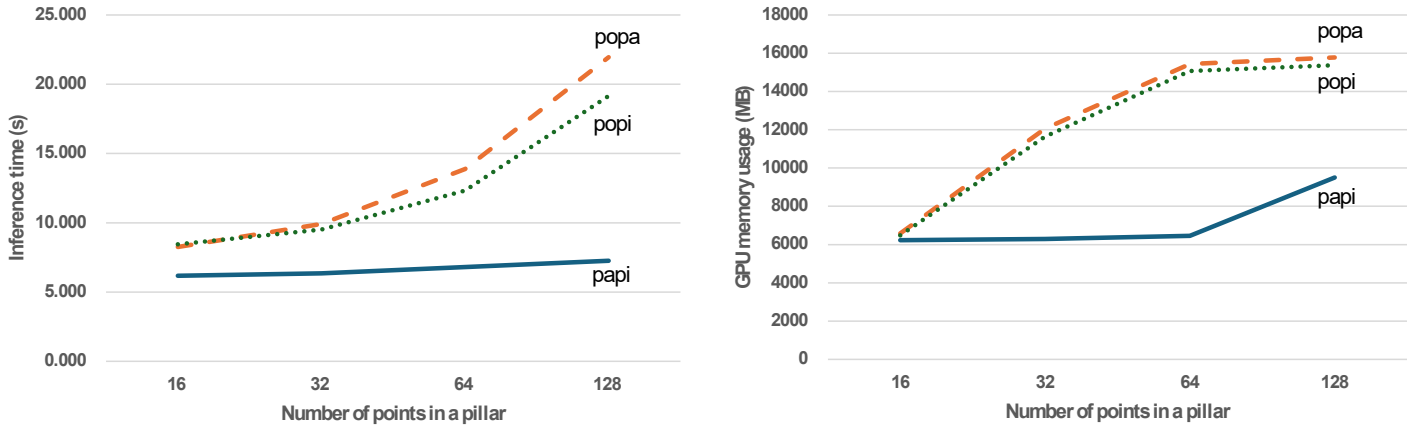

Fig. 2: Inference time and GPU memory usage of each method (papi: PaintedPillars, popa: PointPainting, popi: PointPillars).

5. Conclusion

In 3D object detection using BEV feature map as input using sensor fusion of camera and LiDAR, we newly proposed PaintedPillars, which calculates pillar feature vectors using only statistical processing and explicitly gives class probability distributions to pillars. As a result of the experiment, when the number of points in a pillar was limited to a maximum of 128, the proposed method was 1 % more accurate than the conventional method PointPainting and more than 10 % more accurate than PointPillars. The inference time was 33 % and 38 % of each conventional method, and the GPU memory usage was 60 % and 62 %. The proposed method was found to be more accurate, faster, less memory-intensive, and more efficient than the conventional method. Future challenges include real-time execution of the proposed method and its incorporation into automated driving.

Acknowledgements

I would like to thank Daiki Yamaguchi for his cooperation in this study.

References

[1] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "PointPainting: Sequential Fusion for 3D Object Detection," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 4604-4612.

[2] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast Encoders for Object Detection from Point Clouds," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12697-12705.

[3] L. C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 801-818.

[4] Y. Chai, P. Sun, J. Ngiam, W. Wang, B. Caine, V. Vasudevan, X. Zhang, and D. Anguelov, "To the Point: Efficient 3D Object Detection in the Range Image with Graph Convolution Kernels," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 16000-16009.

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[6] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Beijbom, "nuScenes: *A multimodal dataset for autonomous driving", Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11618-11628.

[7] https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo, Performance on ZCU104, 103 multi_task_v3.

[8] Q. Y. Zhou, J. Park, and V. Koltun, “Open3D: A Modern Library for 3D Data Processing,” *arXiv:1801.09847 [cs.CV]*, https://doi.org/10.48550/arXiv.1801.09847.
[9] https://pytorch.org/