PointMoSeg: Sparse Tensor-Based End-to-End Moving-Obstacle Segmentation in 3-D Lidar Point Clouds for Autonomous Driving

Yuxiang Sun, *Member, IEEE*, Weixun Zuo, Huaiyang Huang, *Graduate Student Member, IEEE*, Peide Cai, *Graduate Student Member, IEEE*, and Ming Liu, *Senior Member, IEEE*

***Abstract*—Moving-obstacle segmentation is an essential capability for autonomous driving. For example, it can serve as a fundamental component for motion planning in dynamic traffic environments. Most of the current 3-D Lidar-based methods use road segmentation to find obstacles, and then employ ego-motion compensation to distinguish the static or moving states of the obstacles. However, when there is a slope on a road, the widely-used flat-road assumption for road segmentation may be violated. Moreover, due to the signal attenuation, GPS-based ego-motion compensation is often unreliable in urban environments. To provide a solution to these issues, this letter proposes an end-to-end sparse tensor-based deep neural network for moving-obstacle segmentation without using GPS or the planar-road assumption. The input to our network are merely two consecutive (previous and current) point clouds, and the output is directly the point-wise mask for moving obstacles on the current frame. We train and evaluate our network on the public nuScenes dataset. The experimental results confirm the effectiveness of our network and the superiority over the baselines.**

***Index Terms*—3-D Lidar, autonomous driving, end-to-end, moving obstacle, point cloud, sparse tensor.**

I. Introduction

IN DYNAMIC traffic environments, especially in urban cities, moving obstacles, such as walking pedestrians and moving vehicles, are usually everywhere and unavoidable. On-line segmentation of moving obstacles in 3-D is an essential capability for autonomous driving. For example, it can be integrated with trajectory planning to find collision-free trajectories on-line when other road users bypass the ego-vehicle [1]. Moreover, it can also be used to improve simultaneous localization and mapping (SLAM) in dynamic traffic environments. Virtually all the current SLAM systems are built on the static-world assumption, moving objects could disturb the data associations in SLAM, and hence degrade the performance. Removing moving objects with motion segmentation has been validated as an effective solution to address this problem [2]–[5].

Most of the current 3-D Lidar-based moving-obstacle segmentation methods generally contain two steps [6]–[8]. The first step is to find obstacles, which is usually based on road segmentation, since it is intuitive to assume that the objects standing upright from the ground are obstacles. The second step is to distinguish whether an obstacle is moving or static. Note that we only consider on-line distinguishing here, which means that only the sensor measurements on or before the current time are used. This is important for causal systems (e.g., robots and self-driving cars), because it is often difficult to use future information for current decisions. Note that background subtraction methods can also detect motions on-line [9]. However, they are developed to work on static platforms (e.g., for video surveillance), while this work targets on moving platforms.

For the first step, based on the flat-road assumption, many methods use plane fitting algorithms to segment a road. However, the real road environments may be complex with potholes, uphill/downhill slopes and undulated surfaces, making these methods less generalizable. For the second step, most methods distinguish static or moving states using ego-motion compensation. Considering that if an ego-vehicle is static, the differences between two or more consecutive sensor measurements would be only caused by object motions. Moving objects could be easily identified through frame differencing with correct object correspondences (object tracking). However, if the ego-vehicle is moving, the differences would be caused by both the ego-vehicle motion and object motion. Compensating for the motion of the ego-vehicle allows adjacent frames to be captured as if from a static platform, so that moving objects can be identified. In autonomous driving, the motion of the ego-vehicle is usually obtained from GPS, or obtained from SLAM algorithms with camera, IMU, Lidar or sensor fusion. However, due to signal attenuation and multipath effects, the GPS performance may be degraded in urban environments, especially in downtown areas, and the SLAM performance could be degraded by moving objects as aforementioned.

The above issues motivate us to propose a deep-learning solution for 3-D moving-obstacle segmentation. So in this letter,

Manuscript received August 25, 2020; accepted December 6, 2020. Date of publication December 28, 2020; date of current version January 12, 2021. This letter was recommended for publication by Associate Editor S. Lee and Editor Y. Choi upon evaluation of the Reviewers' comments. This work was supported in part by the Young Scientists Fund of the National Natural Science Foundation of China under Grant 62003286, and in part by the Start-up Fund of The Hong Kong Polytechnic University under Grant P0034801. *(Corresponding author: Yuxiang Sun.)*

Yuxiang Sun is with the Department of Mechanical Engineering, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong (e-mail: yx.sun@polyu.edu.hk, sun.yuxiang@outlook.com).

Weixun Zuo is with the Shenzhen Unity Drive Innovation Technology Co. Ltd., Shenzhen, China (e-mail: zuoweixun@gmail.com).

Huaiyang Huang, Peide Cai, and Ming Liu are with The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: hhuangat@connect.ust.hk; pcaiaa@connect.ust.hk; eelium@ust.hk).

Digital Object Identifier 10.1109/LRA.2020.3047783

we develop an end-to-end point-cloud segmentation network, named PointMoSeg, to point-wisely label moving obstacles in 3-D point clouds. The input to our network are two consecutive point clouds (previous and current) from a 3-D Lidar, and the output is the point-wise moving-obstacle mask for the current frame. As we only use the sensor measurements on or before the current time during inference, our method is on-line.

Different from previous works, our method does not need road segmentation or ego-motion compensation, thereby alleviating the previous issues from the flat-road assumption and ego-motion estimation (e.g., the GPS issue). Note that although the road segmentation in previous works can be replaced with recent deep learning-based methods [10], which could also alleviate the flat-road assumption, the methods adopting the two-step pipeline could be bounded by the accuracy of the intermediate step of road segmentation. For example, misclassification of obstacles as part of the road could finally reduce the moving-obstacle segmentation accuracy, while our end-to-end paradigm could avoid such error propagation. To the best of our knowledge, this is the first end-to-end solution that segments moving obstacles on-line solely using 3-D point-cloud data information.

As 3-D Lidar point clouds are unstructured, they cannot be directly processed by convolutional neural networks (CNNs) that are designed for grid-like data, such as images or voxel grids. So one type of methods does not use CNNs, like PointNet [11], which processes point clouds mainly using fully-connected layers. To use CNNs, an intuitive idea is to impose structures on point clouds, such as projecting point clouds into front-view or bird-eye-view 2-D images. However, this could lead to the 3-D spatial information lost. Another idea is to voxelize point clouds into 3-D voxel grids, and then apply 3-D CNNs on the grids. As this method takes all the voxels including empty values into computation, it suffers from high-memory consumption and slow-speed computation. To alleviate this issue, sparse convolution based on sparse tensors [12] is proposed by taking the sparsity of voxel grids into consideration. Compared to 3-D CNNs, the sparse convolution is more efficient because sparse tensors are more compact and the convolutional output is only computed on pre-defined coordinates. In this letter, we build our network using the sparse convolution [13] based on sparse tensor. The contributions are summarized as follows:

1) We propose a novel sparse tensor-based moving-obstacle segmentation network merely using two frames of consecutive point clouds.
2) We develop a novel temporal module and a spatial module in our network to infer moving obstacles from the two-frame sequential information.

II. Related Work

A. Semantic Segmentation

Semantic segmentation aims to densely label each pixel or voxel/point in an image or point cloud into individual categories. Shelhamer *et al.* [14] proposed the first end-to-end image semantic segmentation network, Fully Convolutional Networks (FCN), by replacing the fully connected layers in image classification networks with convolution layers. Badrinarayanan *et al.* [15] proposed SegNet, in which the Encoder-Decoder architecture was firstly introduced. Chen *et al.* [16] developed Deeplab v3+, in which an atrous spatial pyramid pooling (ASPP) module was designed to capture the contextual information at multiple scales. For semantic point-cloud segmentation, Aijazi *et al.* [17] proposed a traditional pipeline that first clusters voxels into different objects, then semantically labels each object according to their properties. Wu *et al.* [18] proposed a deep learning-based method. They projected point clouds into range images, and then used CNN to segment the range images to get point-wise segmentation results. There also exist many object-detection networks, such as [19]–[21], from which the point-cloud processing methods and encoders can be borrowed for semantic segmentation.

B. 3-D Obstacle Segmentation

Many previous 3-D Lidar-based methods were proposed under the flat-road assumption. For example, the distance between two consecutive laser rings should be around a fixed value for a flat road. Based on this observation, Hata *et al.* [22] detected obstacles by thresholding the distance value. Asvadi *et al.* [6] represented the 3-D world in 2.5-D grids, and distinguished the ground and obstacles by thresholding the elevation variances of the 2.5-D grids. Since the assumption of flat roads is difficult to generalize to uneven roads, Asvadi *et al.* [7] alleviated this assumption by assuming that roads are piecewise planar, and used a plane fitting algorithm based on RANSAC to detect small road patches. Deep learning-based methods have become popular recently. They do not rely on the flat-road assumption. For example, Caltagirone *et al.* [10] projected a 3-D point cloud onto a front-view image, and then applied the upsampling procedure to produce a dense depth image. They developed a road segmentation network by cross-fusing the front-view image and the dense depth image. Yang *et al.* [23] designed a multi-view and multi-modal road segmentation network by taking as input the front-view image and Lidar depth image.

C. On-Line Distinguishing Static or Moving States

As aforementioned, ego-motion compensation was usually adopted for distinguishing moving/static states on-line. It can be generally divided into two categories: frame-to-map and frame-to-frame. For the first category, Azim *et al.* [24] built local maps by accumulating Lidar scans based on a sensor fusion-based odometry algorithm. They performed ego-motion compensation by registering a new scan to the local map. So moving objects could be detected from the inconsistencies between the new scan and the local map. With the similar idea, Asvadi *et al.* [7] built local maps in voxel grids using GPS and the iterative closest point (ICP) algorithm, then registered a new scan to the local map. Instead of on-line building local maps, Kiran *et al.* [8] registered a new scan to a pre-built map to perform the ego-motion compensation. For the second category, Sun *et al.* [4] compensated the camera ego-motion by warping the last frame to the current frame using the estimated 2-D homography, and then subtracted the warped frame with the current frame to find moving objects. There also exist some deep learning-based methods. Dewan *et al.* [25] used a semantic segmentation network to classify points to non-movable or movable, then used a Bayesian filter together with ego-motion to further infer the moving objects. Siam *et al.* [26] developed an end-to-end moving-object segmentation network by taking as input the current front-view image and an optical flow map.

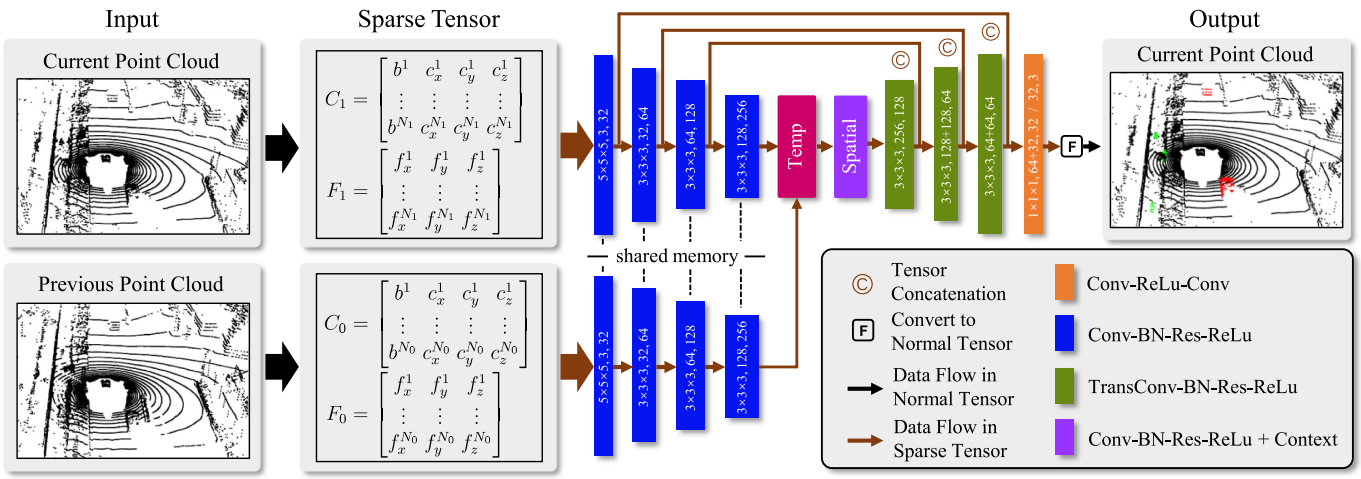

Fig. 1. Overall architecture of our network. The four blue rectangles constitute the encoder. The three green rectangles constitute the decoder. The kernel size, the numbers of input and output channels of the sparse convolutional and transposed convolutional layers (excluding the residual blocks) are respectively displayed on the rectangles. For the sparse convolutional layers in the residual blocks, the kernel size is $3 \times 3 \times 3$, and the number of input channels is equal to that of the output channels. The orange rectangle represents the output layer, in which the two sparse convolutional layers share the same kernel size $1 \times 1 \times 1$. C_0, F_0 and C_1, F_1 are the coordinate matrices and associated feature matrices for the two sparse tensors. N_0 and N_1 in the matrices represent the numbers of points after sparse quantization. *Temp*, *Spatial*, *Conv*, *TransConv*, *BN*, *Res* and *Context* represent the temporal module, the spatial module, convolution, transposed convolution, batch normalization, residual block and context block, respectively. The structure for the residual block can be found in Fig. 2. The red and green colors in the output represent moving vehicles and moving pedestrians, respectively. The figure is best viewed in color.

III. Proposed Approach

A. Sparse Tensor and Sparse Convolution

As we use the sparse tensor-based point-cloud processing method, our first task is to convert input point clouds into sparse tensors. A sparse tensor consists of a voxel coordinate matrix C and an associated feature matrix F (the sparse point cloud). The first step is to get C, which is realized by voxelizing a point cloud with a pre-defined voxel size. The second step is to find F, which is realized by removing redundant points in a same voxel (one voxel can only contain one point):

$$C = \begin{bmatrix} b^1 & c_x^1 & c_y^1 & c_z^1 \\ \vdots & \vdots & \vdots & \vdots \\ b^N & c_x^N & c_y^N & c_z^N \end{bmatrix}, \quad F = \begin{bmatrix} f_x^1 & f_y^1 & f_z^1 \\ \vdots & \vdots & \vdots \\ f_x^N & f_y^N & f_z^N \end{bmatrix}, \tag{1}$$

where b^i is the batch index for point i, $\{c_x^i, c_y^i, c_z^i\} \in \mathbb{Z}^3$ is the voxel-quantized integer-type coordinate, $\{f_x^i, f_y^i, f_z^i\} \in \mathbb{R}^3$ is the float-type coordinate generated by a 3-D Lidar, $i \in [1, N]$, N is the number of points after quantization, which is determined by the voxel size, $N \leq N_o$, where N_o is the original number of points before quantization.

The sparse convolution [13] takes as input a sparse tensor and also outputs a sparse tensor. Specifically, it first generates the coordinate matrix C^{out} for the output sparse tensor from the given input coordinate matrix (details are described in [13]). Then, it calculates the feature vector $\mathbf{f}_{\mathbf{c}}^{\text{out}}$ for an output coordinate $\mathbf{c}$ with the formula:

$$\mathbf{f}_{\mathbf{c}}^{\text{out}} = \sum_{\mathbf{s} \in \mathcal{N}(\mathbf{c}, K)} W_{\mathbf{s}} \mathbf{f}_{\mathbf{c}+\mathbf{s}}^{\text{in}}, \quad \mathbf{f}_{\mathbf{c}}^{\text{out}} \in F^{\text{out}}, \ \mathbf{c} \in C^{\text{out}}, \tag{2}$$

where $\mathbf{s}$ represents the offset to find the corresponding input coordinates, they are within the $\mathbf{c}$-centred neighbourhood covered by the kernel size K, which is denoted as $\mathcal{N}(\mathbf{c}, K)$, $\mathbf{f}_{\mathbf{c}+\mathbf{s}}^{\text{in}}$ represents the input feature vector at the input coordinate $\mathbf{c} + \mathbf{s}$, $W_{\mathbf{s}}$ represents the coefficient, which is to be learned through the training process. With the coordinate matrix C^{out} and feature matrix F^{out}, the output sparse tensor can be produced. Note that in the following text the feature matrix is also called feature vectors. The dimension of feature vectors is 3 for input sparse tensors, and changes with the convolutional operations.

B. Network Overview

Fig. 1 shows the overall architecture of our proposed sparse tensor-based network PointMoSeg. As we can see, we first convert the previous and current point clouds into sparse tensors. Then, we feed the sparse tensors into PointMoSeg to find the point-wise mask for moving obstacles on the current frame. The network mainly consists of an encoder, a temporal module, a spatial module, a decoder and an output layer. All of them are built with the sparse convolution [13]. The Encoder-Decoder architecture has been proven successful in CNN-based semantic segmentation. We adapt the encoder and decoder from ResUNet [27] into our network. The encoder is designed to extract features from the sparse tensors. During the encoding process, the number of feature vectors is gradually reduced, and the number of channels (dimension) is gradually increased. The *shared memory* in Fig. 1 means that we only have one encoder, which is shared by the two frames. The decoder is designed to reduce the number of channels and restore the number of feature vectors. After the output layer, the shape of the feature vector is $N_1 \times 3$, in which 3 means three classes: static background, moving vehicle and moving pedestrian. We finally convert the output sparse tensor to normal dense tensor by extracting the feature matrix F from the sparse tensor.

Between the encoder and decoder, we develop a temporal module and a spatial module. The former is designed to infer

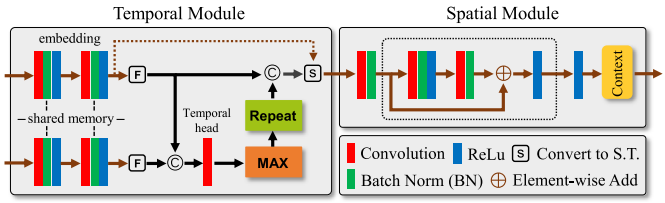

Fig. 2. Architectures for the proposed temporal and spatial modules. The inputs to the temporal module are the current (upper) and previous (lower) feature vectors. The dash line means passing the coordinate key and manager to construct the sparse tensor. *S.T.* represents sparse tensor. *MAX* means the maximum operation. The dash box shows the structure of the residual block. In the temporal module, the kernel sizes of the two sparse convolutional layers in the embedding block are set to $7 \times 7 \times 7$ and $3 \times 3 \times 3$, respectively. The temporal head is a 1-D normal dense convolutional layer, in which the kernel size is set to 3. The input and output channels for all the three convolutional layers is 256. In the spatial module, the kernel size for the first sparse convolutional layer is $3 \times 3 \times 3$, and the number of the input and output channels are both 512. The figure is best viewed in color.

the temporal information from the two frames and hence extract the features of moving obstacles. The latter is designed to restore the spatial information of the moving obstacles.

C. Temporal and Spatial Module

Fig. 2 shows the architectures for our temporal and spatial modules. In the temporal module, the sparse tensors from the current and previous frames are first fed into an embedding block to generate embedding vectors. The embedding block sequentially consists of two *Conv-BN-ReLu* blocks, which are shared by the two frames. Secondly, the embedding vectors are converted to normal dense tensors. They are with the shape of $M_0 \times CH$ and $M_1 \times CH$, where M_0 and M_1 are the numbers of the embedding vectors, CH is the number of channels. In this work, $CH = 256$. Generally, there exists $M_0 \neq M_1$ because the numbers of points for the two frames are different. We concatenate the two embedding vectors at the first dimension, so the concatenated embedding vector has the shape of $(M_0 + M_1) \times CH$. It is then sent to a temporal head (i.e., one-layer 1-D normal dense convolution) to compare the two vectors to produce the temporal feature. Thirdly, we apply the maximum operation on the first dimension of the temporal feature to extract the most prominent features in each channel. The prominent feature is with the shape of $1 \times CH$. Finally, we repeat the prominent feature to the shape of the current embedding vectors $M_1 \times CH$, and concatenate the current embedding vectors with the repeated prominent feature at the second dimension, so the final output is with the shape of $M_1 \times 2CH$. The output is converted to sparse tensor using the coordinate key and manager from the current embedding vector.

The output of the temporal module is fed into the spatial module, which sequentially consists of a *Conv-BN-Res-ReLu* block and a context block. We borrow the ASPP module (with the global average pooling branch removed) from Deeplab v3+ [28] as the context block and implement it using sparse convolution. The context block is expected to capture the context information of moving obstacles at multiple scales. It mainly consists of four parallel branches. Each branch consists of a *Conv-BN-ReLu* block. The dilation rates for the sparse convolutional layers in the four branches are set to 1, 6, 12, 18, respectively. We refer readers to [28] for more details.

IV. Experimental Results and Discussions

A. Dataset

We use the nuScene dataset [29] in this work, because it provides hand-labelled ground-truth of 3-D bounding boxes for moving obstacles (i.e., moving vehicle and moving pedestrian) in traffic environments. We derive our point-wise mask from the bounding boxes. Various sensors have been used to record the nuScene dataset, including one 32-beam mechanical spinning Lidar, 6×cameras, 5×radars and 1×IMU. In the dataset, a total of 1000 driving sequences were recorded in Boston and Singapore, where dense traffic and challenging driving situations are common. Each sequence lasts around 20 seconds. Ground truth is annotated at 2 Hz, so there are around 40 frames with ground truth in each sequence. Among the 1000 sequences, 850 sequences are publicly provided with ground truth. We randomly split the 850 sequences into *train* (425 sequences), *validation* (213 sequences) and *testing* (212 sequences), which account for around 50.00%, 25.06% and 24.94%. Note that we use an entire sequence for training, validation and testing. In other words, all the Lidar point clouds in a same sequence are used either for training or validation or testing. So the point clouds used for validation and testing are ensured not from the training set.

B. Training Details

We implement our PointMoSeg using PyTorch with the MinkowskiEngine sparse-tensor library (v0.4) [13], and train the network using the stochastic gradient descent optimizer. The initial learning rate, momentum and weight decay are set to 0.01, 0.9, and 0.001, respectively. The learning rate is exponentially decayed during training. The network is trained until the validation loss converges. The input sequences for training are randomly shuffled before each epoch. As we perform the shuffle operation on a whole set of two consecutive frames, the order of the two frames are ensured not being influenced.

For the loss functions, we adopt the cross entropy loss $\mathcal{L}_{ce}$ that is widely used in semantic segmentation for our point-wise segmentation. Since the number of points for the background and foreground are imbalanced (i.e., 98.03% for background, 1.83% for moving vehicle, 0.14% for moving pedestrian), we use the dice loss $\mathcal{L}_{dice}$ to tackle the imbalance problem [30]. So the total loss is:

$$\mathcal{L} = \mathcal{L}_{ce} + \mathcal{L}_{dice}. \tag{3}$$

For the dice loss, we employ the multi-class dice loss implementation from PyTorch Toolbelt [31].

C. Parameter Tuning

As aforementioned, we need to voxelize a point cloud to get a sparse tensor. The voxel size is a key parameter in the voxelization, which influences the granularity of a sparse tensor, and then influences the efficiency and performance of a network. A smaller voxel size is expected to provide better performance, but the efficiency would be reduced. We train and test our network with the voxel sizes of 0.1 m, 0.3 m, 0.5 m, 0.7 m and 0.9 m. The widely-used per-class accuracy (Acc) and

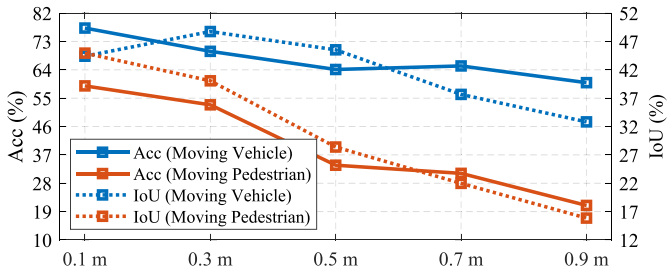

Fig. 3. Results for different voxel sizes. We use both the cross entropy loss and dice loss for all the experiments. We can see that the performance decreases as the voxel size increases. The figure is best viewed in color.

TABLE I
THE INFERENCE TIME FOR DIFFERENT VOXEL SIZES. MS MEANS MILLISECOND

	0.1 m	0.3 m	0.5 m	0.7 m	0.9 m
Time	205.54 ms	126.26 ms	101.96 ms	91.82 ms	84.72 ms

TABLE II
THE EXPERIMENTAL RESULTS USING DIFFERENT LOSS FUNCTIONS. BEST RESULTS ARE HIGHLIGHTED IN BOLD FONT

Loss Function	Moving Vehicle		Moving Pedestrian	
	Acc	IoU	Acc	IoU
$\mathcal{L}_{ce}$	64.19%	44.80%	40.62%	27.66%
$\mathcal{L}_{dice}$	67.24%	**48.97%**	44.85%	38.22%
$\mathcal{L}_{ce}+\mathcal{L}_{dice}$	**69.87%**	48.75%	**52.89%**	**40.06%**

intersection-over-union (IoU) metrics [32] are employed in this work to measure the segmentation performance. A larger value corresponds to better performance for both metrics. Fig. 3 and Table I display the segmentation results and the inference time tested on an NVIDIA RTX 2080 Ti card, respectively. We can see that, as the voxel size increases, the segmentation performance is degraded but the efficiency is increased, which is expected. By pair-wisely comparing the results from 0.1 m to 0.3 m, as well as those from 0.3 m to 0.5 m, we find that the performance degradation of the former is less than that of the latter. Since the training time cost for 0.1 m is much longer than that for 0.3 m, to trade-off the performance and efficiency, we use the voxel size of 0.3 m in the following experiments.

D. Loss Function Analysis

As aforementioned, we have two loss functions, one is the cross entropy loss $\mathcal{L}_{ce}$, and the other is the dice loss $\mathcal{L}_{dice}$. We respectively train our network using only $\mathcal{L}_{ce}$, only $\mathcal{L}_{dice}$, and both of them. The results are displayed in Table II. As we can see, using both the two loss functions generally gives the best performance, which shows that combining the two loss functions is a benefit here. So in the following experiments, we use both the two loss functions. Comparing the results of using only $\mathcal{L}_{ce}$ and only $\mathcal{L}_{dice}$, we find that using only $\mathcal{L}_{dice}$ gives better performance, especially for the IoU of the moving-pedestrian class. This demonstrates that using dice loss can effectively boost the performance for the class with very few points, and the dice loss is a useful solution to tackle the imbalance problem.

TABLE III
THE EXPERIMENTAL RESULTS FOR THE ABLATION STUDY. BEST RESULTS ARE HIGHLIGHTED IN BOLD FONT

Variants	Moving Vehicle		Moving Pedestrian	
	Acc	IoU	Acc	IoU
No-Temporal	60.65%	42.83%	41.37%	35.37%
No-Spatial	62.21%	45.93%	39.66%	34.41%
No-TempSpat	49.59%	35.61%	41.26%	34.01%
One-CBR	65.37%	**49.42%**	44.42%	38.06%
Minimum	67.00%	49.43%	46.06%	37.97%
Only-Context	67.75%	48.63%	43.77%	37.08%
No-Context	64.22%	47.10%	45.96%	37.75%
PointMoSeg (Ours)	**69.87%**	48.75%	**52.89%**	**40.06%**

E. Ablation Study

In this section, we compare our network with several variants to check whether the proposed temporal and spatial modules are beneficial to our network. The descriptions for the variants are listed as follows:

- **No-Temporal**: This variant has no the temporal module. The feature vectors from the encoder are simply concatenated at the first dimension, and then fed to the spatial module.
- **No-Spatial**: This variant has no the spatial module. The output from the temporal module is directly fed into the decoder.
- **No-TempSpat**: Based on the No-Temporal variant, this variant also deletes the spatial module. It can be seen as a combination of the above two variants.

The experimental results are displayed in Table III. Comparing the No-TempSpat results with ours, we find that our network is significantly better than No-TempSpat. In addition, the No-TempSpat variant generally presents the worst results among all the variants, especially for the moving-vehicle class. This shows that the proposed temporal and spatial modules are effective and beneficial to our network. We also find that the results of No-Temporal and No-Spatial are similar to each other, and none of them could give close results to ours. This indicates that only using one module is not enough. Combining the two modules together is necessary here.

To validate the effectiveness of our design for the temporal and spatial modules, we also create several variants:

- **One-CBR**: This variant modifies the embedding block in the temporal module to contain only one *Conv-BN-ReLu* block.
- **Minimum**: This variant replaces the maximum operation in the temporal module with the minimum operation.
- **Only-Context**: This variants deletes the *Conv-BN-Res-ReLu* block in the spatial module. So the spatial module only contains the context block.
- **No-Context**: This variants deletes the context block in the spatial module. So the spatial module only contains the *Conv-BN-Res-ReLu* block.

As we can see from Table III, our network generally presents the best performance among all the variants, which confirms the effectiveness of our design for the two modules. Comparing the

Minimum results with ours, we find that using the minimum operation causes performance degradation. So using the maximum operation would be more suitable to extract the prominent features here, especially for the moving-pedestrian class. For the One-CBR variant, we believe that its inferior performance is due to the insufficient embedding features, because there is only one *Conv-BN-ReLu* block in the embedding block of the temporal module. Comparing the results of Only-Context and No-Context, we believe that the context module plays a more important role in the spatial block, because the Only-Context generally presents better performance than No-Context.

F. Comparative Study

1) Baseline Methods: We create three baseline methods for the comparative study. The first two baselines are modified from our network, and the third one is built on PointNet [11]:

- **Two-Enc Baseline**: This baseline does not share the memory for the encoder in our network. In other words, the baseline has two identical encoders for the two consecutive point clouds. We name this baseline as Two-Enc, where Enc is the abbreviation for Encoder.
- **No-Seq Baseline**: This baseline does not take as input the point-cloud sequence (i.e., the two consecutive frames). It takes as input only the current frame, so there is no temporal information, and hence we remove the temporal module in our network. The features from the encoder are directly fed into the spatial module. We name this baseline as No-Seq, where Seq is the abbreviation for sequence.
- **PointNet Baseline**: This baseline is built on the PointNet re-implemented with the Minkowski sparse-tensor library [13], in which the fully-connected layers in PointNet are realized using $1 \times 1 \times 1$ sparse convolutions since there is no implementation for fully-connected layer in the library. Note that a convolution with kernel size of 1 is equivalent to fully connection [33]. We choose PointNet to build a baseline, because it is a typical point-cloud processing method that does not use CNNs. So this can be a representative contrast to our method. The architecture of the baseline is shown in Fig. 4. Similar to our method, the input point clouds are first converted to sparse tensors, and then the PointNet (with the last layer removed) is employed to extract features and generate predictions. The weights of the PointNet are shared between the two frames. To ensure fair comparison, we also integrate the proposed temporal and spatial modules in this baseline. At the end of the baseline, we employ a $1 \times 1 \times 1$ sparse convolutional layer to output the segmentation results. Since we have three classes and the prediction is made on the current frame, the output shape is $N_1 \times 3$.

We have also tried using 3-D CNN to realize our network, but it is not feasible. As 3-D CNN requires grid-like input data, the Lidar scan must be cropped. For example, cropping the x and y directions within -60 m ~ 60 m and the z direction within -2 m ~ 4 m leads to $400 \times 400 \times 20$ voxel grids given the 0.3 m voxel size. However, feeding such voxel grids into the 3-D CNN-based network causes the out of memory error (our GPU card has only 11 GB memory), thus making the network unable to train. Even we reduce the size of voxel grids, the training process costs much longer time than using sparse convolution, making it not feasible in practice. Moreover, the cropping operation may lose useful information. Compared to 3-D CNN, sparse convolution does not require grid-like data, so it could well utilize the sensory information without cropping. In addition, the computing efficiency is much higher, and the GPU memory cost is much lower, making the sparse convolution a promising solution for point-cloud processing.

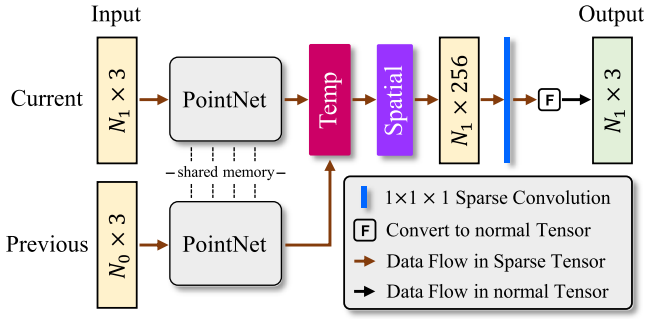

Fig. 4. Architecture for the PointNet baseline. To keep the figure concise, the process of converting input point clouds to sparse tensors is omitted. The yellow and green rectangles represent the sparse tensor and normal dense tensor, respectively. The shapes of them are indicated on the rectangles. N_0 and N_1 represent the number of points after sparse quantization for the two consecutive point clouds. The weight of PointNet is shared between the two frames. *Temp* and *Spatial* mean the proposed temporal and spatial modules, respectively. The figure is best viewed in color.

TABLE IV
THE QUANTITATIVE COMPARATIVE RESULTS WITH THE BASELINES. BEST RESULTS ARE HIGHLIGHTED IN BOLD FONT

Methods	Moving Vehicle		Moving Pedestrian		Time
	Acc	IoU	Acc	IoU	
PointNet Baseline	48.22%	33.81%	41.42%	33.47%	130.76 ms
No-Seq Baseline	65.11%	47.54%	43.67%	37.78%	**53.80 ms**
Two-Enc Baseline	68.27%	48.41%	48.03%	39.63%	126.45 ms
PointMoSeg (Ours)	**69.87%**	**48.75%**	**52.89%**	**40.06%**	126.26 ms

2) Quantitative Results: The quantitative results for the comparison are shown in Table IV. As we can see, our network outperforms the three baselines in terms of both the Acc and IoU metrics. We find that the PointNet baseline gives the worst performance, especially for the moving vehicle, which is significantly lower than the other methods. Comparing the Two-Enc baseline results with ours, we find that using shared memory for the encoder can provide better performance. The reason could be that the use of shared memory actually trains the encoder almost twice in each iteration, because there is usually a lot of overlap between two consecutive frames. More training leads to better features, and hence better performance. Comparing the No-Seq baseline to the Two-Enc baseline and our network, we find that the No-Seq baseline gives the worst performance. This is to be expected, because it is difficult to infer whether an object is moving or not with just one observation. In such a case, the No-Seq baseline simply treats our task as semantic point-cloud segmentation. Table IV also displays the inference time for each method tested on an NVIDIA RTX 2080 Ti card with the voxel

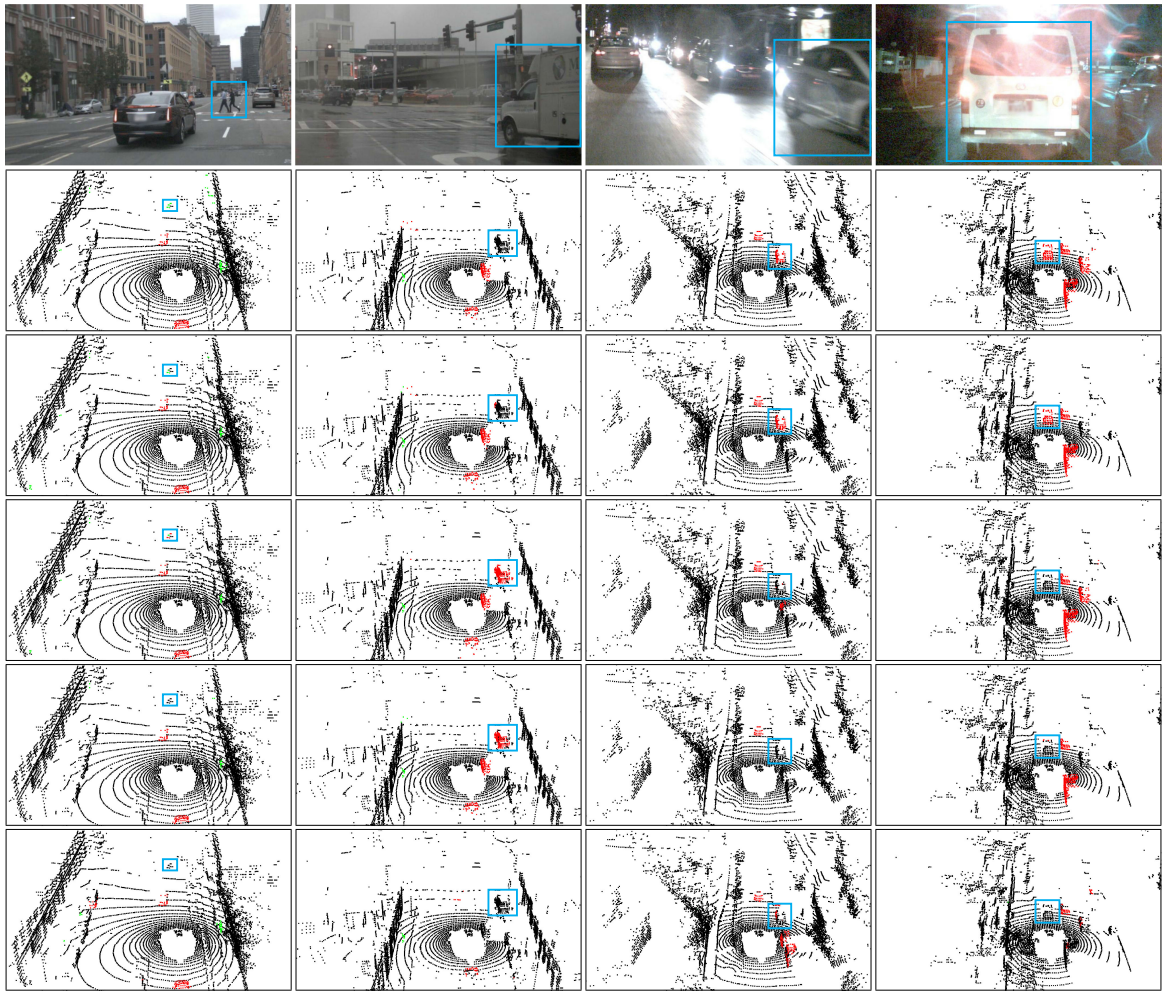

Fig. 5. Sample comparative demonstrations. The figures from the top row to the bottom row are front-view RGB images, ground truths, our PointMoSeg results, the No-Seq baseline results, the Two-Enc baseline results, the PointNet baseline results. Please zoom in for details. Note that the front-view RGB images are just for visualization. They are not used as input for the networks. The red and green colors in the results denote moving vehicle and moving pedestrian, respectively. The cyan boxes are just used for highlighting. The figure is best viewed in color.

size of 0.3 m. The No-Seq baseline achieves the fastest speed. This is because it deals with only one point-cloud frame at each inference, while the others deal with two frames. The time cost for the Two-Enc baseline is close to ours. This is because they both run the same encoder two times at each inference.

We also compare our method with the previous traditional method proposed by Asvadi *et al.* [7], which is also the most close work to ours. As [7] only provides binary segmentation results (i.e., static background and moving obstacle), we map the two moving classes to the one moving-obstacle class to generate the binary results for our network. In addition, as [7] uses ego-motion compensation to discriminate the motion states of obstacles, we feed the ground-truth Lidar pose provided by nuScenes to [7] as the ego-motion. Moreover, since [7] requires voxel grids as input, we crop the point cloud into the $400 \times 400 \times 20$ voxel grids with the voxel size of 0.3 m. We have also verified through a few experiments that increasing the size of the voxel grids could not further increase the performance, so we finally choose $400 \times 400 \times 20$ with a consideration on the running speed. We test the speed for [7] on a PC with a 3.60 GHz Intel i7-7820X CPU. Table V displays the comparative results.

TABLE V
THE QUANTITATIVE COMPARATIVE RESULTS WITH THE CLOSE-RELATED WORK [7]. BEST RESULTS ARE HIGHLIGHTED IN BOLD FONT

Methods	Moving Obstacle		Time
	Acc	IoU	
Asvadi *et, al.* [7]	23.50%	17.73%	898.92 ms
PointMoSeg (Ours)	**67.81%**	**47.89%**	**126.26 ms**

As we can see, our method significantly outperforms [7] in terms of both the accuracy and inference speed, which demonstrates the powerful capability of deep learning. Moreover, we find that good ego-motion could not ensure good results for the traditional method. The performance may also be subject to the ego-motion compensation strategy, road segmentation accuracy and data association algorithm (object tracking), etc.

3) Qualitative Demonstrations: Fig. 5 demonstrates sample comparative results. As we can see, our PointMoSeg generally gives the best results among all the methods. Specifically, in the first column, the Two-Enc baseline and PointNet baseline

totally fail to detect the walking pedestrians highlighted in the box. The No-Seq baseline produces ambiguous decisions. But our network gives correct and acceptable results. The second column shows a rainy daytime environment. We can see that the lens of the RGB camera is blurred. The truck in the box is waiting for the traffic light to go. The No-Seq baseline and Two-Enc baseline wrongly classify the truck as moving. Although the PointNet baseline correctly classifies it as static, it misses the moving vehicle on the left of the truck. The third column shows a nightime environment. We can see that, except our network, all the baselines fail to detect the on-coming moving vehicle. Moreover, the PointNet baseline also incorrectly labels the static vehicles behind the ego-vehicle as moving. The last column shows a rainy nightime environment. Severe glares caused by raindrops appear on the RGB image. Except our network, we can see that all the baselines fail to detect the front moving vehicle. Moreover, the PointNet baseline also wrongly labels the moving vehicle behind the ego-vehicle as static.

V. Conclusion

We proposed here a novel end-to-end solution for moving-obstacle segmentation. We solely use point-cloud information, no intermediate steps such as road segmentation and ego-motion compensation are needed, so limitations from previous methods could be alleviated. We adopted the sparse tensor-based point-cloud processing method, and built our network using the Minkowski sparse convolutional networks. Our network takes as input only two consecutive point clouds, and directly outputs the point-wise mask for the current frame. We train and test our network on the public nuScenes dataset. The ablation study and comparative experimental results demonstrate the effectiveness of our design and the superiority over the baselines. However, our network costs around 126 ms for each inference on an NVIDIA RTX 2080 Ti card with the voxel size of 0.3 m, making it not suitable for real-time applications. We consider this as a major limitation. In the future, we would use model pruning techniques to improve the efficiency.

References

[1] P. Cai, Y. Sun, H. Wang, and M. Liu, "VTGNet: A vision-based trajectory generation network for autonomous vehicles in urban environments," *IEEE Trans. Intell. Veh.*, pp. 1–1, 2020.

[2] Y. Sun, M. Liu, and M. Q. -H. Meng, "Motion removal for reliable RGB-D SLAM in dynamic environments," *Robot. Auton. Syst.*, vol. 108, pp. 115–128, 2018.

[3] F. Moosmann and C. Stiller, "Joint self-localization and tracking of generic objects in 3D range data," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2013, pp. 1146–1152.

[4] Y. Sun, M. Liu, and M. Q.-H. Meng, "Improving RGB-D SLAM in dynamic environments: A. motion removal approach," *Robot. Auton. Syst.*, vol. 89, pp. 110–122, 2017.

[5] F. Pomerleau, P. Krüsi, F. Colas, P. Furgale, and R. Siegwart, "Long-term 3D map maintenance in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 3712–3719.

[6] A. Asvadi, P. Peixoto, and U. Nunes, "Detection and tracking of moving objects using 2.5 d motion grids," in *Proc. IEEE 18th Int. Conf. Intell. Transp. Syst*., 2015, pp. 788–793.

[7] A. Asvadi, C. Premebida, P. Peixoto, and U. Nunes, "3D lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes," *Robot. Auton. Syst.*, vol. 83, pp. 299–311, 2016.

[8] B. Ravi Kiran *et al.*, "Real-time dynamic object detection for autonomous driving using prior 3D-maps," in *Proc. Eur. Conf. Comput. Vis.*, 2018.

[9] Y. Sun, M. Liu, and M. Q.-H. Meng, "Active perception for foreground segmentation: An RGB-D data-based background modeling method," *IEEE Trans. Automat. Sci. Eng.*, vol. 16, no. 4, pp. 1596–1609, Oct. 2019.

[10] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde, "Lidar-camera fusion for road detection using fully convolutional neural networks," *Robot. Auton. Syst.*, vol. 111, pp. 125–131, 2019.

[11] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 77–85.

[12] B. Graham, M. Engelcke, and L. Van Der Maaten, "3D semantic segmentation with submanifold sparse convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9224–9232.

[13] C. Choy, J. Gwak, and S. Savarese, "4D spatio-temporal ConvNets: Minkowski convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 3075–3084.

[14] E. Shelhamer and J. Long and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017.

[15] V. Badrinarayanan and A. Kendall and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[16] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 801–818.

[17] A. K. Aijazi, P. Checchin, and L. Trassoudaine, "Segmentation based classification of 3D urban point clouds: A super-voxel based approach with evaluation," *Remote Sens.*, vol. 5, no. 4, pp. 1624–1650, 2013.

[18] B. Wu, A. Wan, X. Yue, and K. Keutzer, "SqueezeSeg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3D lidar point cloud," in *Proc. IEEE Int. Conf. Robot. Automat*., Brisbane, QLD, 2018, pp. 1887–1893.

[19] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 12 697–12705.

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

[21] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "SBNet: Sparse blocks network for fast inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8711–8720.

[22] A. Y. Hata, F. S. Osorio, and D. F. Wolf, "Robust curb detection and vehicle localization in urban environments," in *Proc. IEEE Intell. Veh. Symp. Proc*, 2014, pp. 1257–1262.

[23] F. Yang, H. Wang, and Z. Jin, "A fusion network for road detection via spatial propagation and spatial transformation," *Pattern Recognit.*, vol. 100, 2020, Art. no. 107141.

[24] A. Azim and O. Aycard, "Detection, classification and tracking of moving objects in a 3D environment," in *Proc. IEEE Intell. Veh. Symp*., 2012, pp. 802–807.

[25] A. Dewan, G. L. Oliveira, and W. Burgard, "Deep semantic classification for 3D lidar data," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst*., 2017, pp. 3544–3549.

[26] M. Siam, H. Mahgoub, M. Zahran, S. Yogamani, M. Jagersand, and A. El-Sallab, "MODNet: Motion and appearance based moving object detection network for autonomous driving," in *Proc. 21st Int. Conf. Intell. Transp. Syst*., 2018, pp. 2859–2864.

[27] C. Choy, J. Park, and V. Koltun, "Fully convolutional geometric features," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 8958–8966.

[28] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proce. Eur. Conf. Comput. Vis.*, 2018, pp. 801–818.

[29] H. Caesar *et al.*, "nuScenes: A multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11 621–11 631.

[30] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-Net: Fully convolutional neural networks for volumetric medical image segmentation," in *Proc. 4th Int. Conf. 3D Vis*., 2016, pp. 565–571.

[31] E. Khvedchenya, "PyTorch Toolbelt," 2019. [Online]. Available: https://github.com/BloodAxe/pytorch-toolbelt

[32] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. G. Rodríguez, "Review on deep learning techniques applied to semantic segmentation," *CoRR*, vol. abs/1704.06857, 2017.

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proc. IEEE IRE*, vol. 86, no. 11, pp. 2278–2324, 1998.