MVLidarNet: Real-Time Multi-Class Scene Understanding for Autonomous Driving Using Multiple Views

Ke Chen*, Ryan Oldja*, Nikolai Smolyanskiy*, Stan Birchfield
Alexander Popov, David Wehr, Ibrahim Eden, Joachim Pehserl
NVIDIA

***Abstract*— Autonomous driving requires the inference of *actionable information* such as detecting and classifying objects, and determining the drivable space. To this end, we present a two-stage deep neural network (MVLidarNet) for multi-class object detection and drivable segmentation using multiple views of a single LiDAR point cloud. The first stage processes the point cloud projected onto a *perspective view* in order to semantically segment the scene. The second stage then processes the point cloud (along with semantic labels from the first stage) projected onto a *bird's eye view*, to detect and classify objects. Both stages are simple encoder-decoders. We show that our multi-view, multi-stage, multi-class approach is able to detect and classify objects while simultaneously determining the drivable space using a single LiDAR scan as input, in challenging scenes with more than one hundred vehicles and pedestrians at a time. The system operates efficiently at 150 fps on an embedded GPU designed for a self-driving car, including a postprocessing step to maintain identities over time. We show results on both KITTI and a much larger internal dataset, thus demonstrating the method's ability to scale by an order of magnitude.**

I. INTRODUCTION

Autonomous driving requires perception of *actionable information*, *i.e.*, data that can be directly consumed by the subsystem that controls the vehicle. Actionable information includes data such as the locations of the lanes, the curvature of the road, the color of the stop light, the presence of construction cones, whether the stop sign is facing the vehicle, the distance to the car in front, whether the pedestrian is crossing the road, and so forth. Such information is immediately useful in determining whether the vehicle should turn, accelerate, or brake.

Of these various types of actionable information, perhaps the most attention has been paid to the detection of nearby cars, cyclists, and pedestrians. To solve this problem, researchers have proposed methods using either RGB images, LiDAR data, or a fusion of the two modalities. While RGB object detection itself is relatively mature, lifting such results from image-space to world-space often yields significant geometric inaccuracies. LiDAR data solves this problem but introduces another: directly processing a 3D point cloud is not straightforward. One possibility is to process the point cloud as a 3D voxel grid [1], but this is computationally expensive and introduces quantization errors; another is to project the point cloud to a bird's eye view (BEV) as a height map or multi-channel representation [2], but this loses potentially valuable information, especially for pedestrians due to their small size.

In this paper we propose to overcome these limitations by projecting a LiDAR input into both perspective (ego-centric) and top-down (bird's eye) views. This provides the best of both worlds, since the former allows us to leverage shape information that is so crucial for detecting pedestrians, and the latter allows us to aggregate information in a format that is useful for autonomous driving. To facilitate the detection of pedestrians, we leverage semantic segmentation of the LiDAR points, which has only been recently been made possible with the introduction of the SemanticKITTI [3] dataset that contains pointwise ground truth segmantic segmentation.

Our approach leverages a two-stage multi-view network that performs semantic segmentation on a perspective-view projection of the point cloud, followed by object detection and classification on a bird's-eye projection. Unlike previous approaches, we focus on *multi-class* detection, in which the same network is used to detect multiple object classes simultaneously (that is, without training separate network weights for each class). As we show in the experimental results, this simple approach, leveraging two encoder-decoder stages, is able to achieve competitive results on multi-class KITTI object detection, while simultaneously determining the drivable space. The simple design yields efficient processing, enabling the system to process LiDAR point clouds at 150 fps with competitive accuracy, while maintaining identities over time via postprocessing.

The paper contains the following contributions:

- We present a novel *multi-class* system to detect vehicles and pedestrians while simultaneously computing drivable space, all with a single network processing a LiDAR input via two different projected views.
- Due to the simplicity of its design, our system operates faster than previous approaches, at 150 fps on an embedded GPU.
- Results of the system are shown on extremely challenging data with more than one hundred vehicles and pedestrians per frame, thus advancing state-of-the-art for autonomous driving LiDAR perception.

II. PREVIOUS WORK

There are several deep-learning-based approaches to object detection and classification for LiDAR point clouds. Some papers use 3D voxelized volumes as DNN input, which is

*Equal contribution.

computationally expensive. Other approaches project LiDAR point clouds as multi-channel bird's eye view (BEV) top-down representation and apply 2D convolutions, which is faster. Some papers use RGB perspective camera views in addition to LiDAR point clouds to improve detection rates.

PIXOR [2] uses 2D convolutions to detect objects in LiDAR point clouds projected as multi-channel BEV tensors. The DNN computes object detection confidence maps and regresses object's bounding box positions, sizes and orientations for each output pixel. Clustering is used to extract final bounding boxes. This method is fast, but can mis-detect challenging objects like pedestrians due to similarities to other objects in top-down BEV. In followup work, Fast and furious [1] attempts to solve several problems: detection, object tracking, motion forecasting. It uses an approach similar to PIXOR, but some of its versions use 3D convolutions applied to voxelized point cloud (4D tensor), which can be slow. Results are shown only for an internal dataset.

VoxelNet [4] voxelizes the entire 3D LiDAR point cloud, which is randomly subsampled for dense voxels. It uses a region proposal network (RPN) for 3D object detection and applies linear networks to voxels. It then converts voxels into dense 3D tensors for RPN. This approach is slower and voxelization leads to information loss. SECOND [5] converts point cloud to voxel features and coordinates. It then applies sparse CNN, followed by RPN. It extracts information from a vertical axis of a point cloud before downsampling 3D data as 2D-like representation. It runs at 20–40 fps, but voxelization method causes information loss.

MV3D [6] converts the point cloud to a multi-channel BEV representation that has several channels representing height map and intensity. The system uses LiDAR and RGB camera data to improve accuracy. It projects the point cloud onto BEV and front view (range map), combines it with RGB and uses region proposals for 3D detection. AVOD [7] uses LiDAR and RGB input. It introduced a novel feature extractor based on feature pyramid network (FPN) [8] and RPN for object detection.

PointNet [9] performs object classification and per-point semantic segmentation, operating directly on an unordered 3D point cloud. PointPillars [10] uses only LiDAR point cloud and achieves state-of-the-art accuracy and speed. It uses PointNets to learn to organize a point cloud into columns. DNN encodes points and then runs simplified version of PointNet, and finally run SSD [11] to detect objects. STD [12] uses two stages: PointNet++ for semantic segmentation of a point cloud and proposal generation network for classification and regression predictions.

The recently proposed Lidar DNN [13] also uses perspective and top-down BEV views of LiDAR point clouds. It uses bounding boxes as supervision targets and two parallel branches of DNN learn to extract features relevant to object detection from each view via 2D convolutions. These features, which are learned implicitly, are then used by the network trunk to detect objects. Like other methods, the results appear to show single class object detection. In contrast, our approach uses explicit features and representations for perspective and top-down view, which makes the system easy to train and debug; and we show results for multi-class object detection. Ours is also an order of magnitude faster.

RangeNet++ [14] processes LiDAR in a two-step process: 1) the point cloud is spherically projected onto a ego-centric range image, and a 2D semantic segmentation CNN is run on this range image, and 2) a fast, GPU-accelerated k-nearest neighbor (kNN) post-processing step is applied to the unprojected segmented point cloud to clean up the effects of bleeding between adjacent objects. The result is a real-time approach that can semantically label all points of the original point cloud, regardless of the discretization level of the CNN. But this DNN does not detect objects in the scene. The network is trained on SemanticKITTI, which is a version of the KITTI dataset [15], [16] where each LiDAR point is semantically labeled for 25 classes.

III. METHOD

The proposed system consists of a two-stage neural network, as shown in Fig. 1. The input to the system is a motion-compensated LiDAR point cloud capturing a 360° view of the scene, which is projected both perspectively (spherical projection) and top-down (orthographic projection).

The first stage extracts semantic information from the perspectively-projected LiDAR range scan. For the sensors used in this work, the resolution of each scan is $n_v \times n_s$, where $n_v = 64$ is the number of horizontal lines, and $n_s = 2048$ is the number of samples per line. Each sample contains the distance to the corresponding point as well as the intensity of the received signal. From these distances and intensities, the first stage segments the range scan into the following 7 classes: cars, trucks, pedestrians, cyclists, road surface, sidewalks, and unknown. (We experimented with more or fewer classes but found the best results were obtained with this choice.) Our motivation for using a perspective view is that pedestrians and cyclists are more easily discerned in this manner due to their characteristic shapes.

The architecture is similar to the feature pyramid network (FPN) [8] in its design. Processing is fast, since only 2D convolution/deconvolution layers are used. The encoder consists of three convolutional layers with 64, 64, and 128 3×3 filters at the input resolution, followed by three Inception blocks [17] at resolutions of $1/2$, $1/4$, and $1/8$ (using max pooling for downsampling). The three Inception blocks consist of two, two, and three Inception modules, respectively.[1] The decoder brings features back to the original resolution with 3 deconvolution blocks. Each block consists of a deconvolution layer followed by two convolution layers with 1×1 and 3×3 filters, respectively. The number of filters within these blocks (across all layers) is 256, 128, and 64, respectively. Skip connections are added at quarter and half resolution. The classification head consists of a 3×3 convolution layer with output feature size of 64, followed by a 1×1 layer that outputs a 7-element vector per pixel

[1] Specifically, we use Inception-v2 modules, as depicted in Fig. 5 of [17].

indicating the class probabilities. Every convolution layer is followed by batch normalization and ReLU activation. Input and output have the same spatial resolution, and cross-entropy is used as the loss in training. Note that this first stage directly outputs the drivable space.

The semantically labeled scan is reprojected onto a top-down view and combined with height information from the projected LiDAR data. That is, the second stage uses this representation to detect dynamic objects (vehicles and pedestrians) in the top-down view. The second stage takes as input a $w \times \ell \times d$ array, where w and ℓ are the dimensions of the array in the top-down view, and each cell consists of a d-dimensional vector of class probabilities concatenated with min height, max height, and mean intensity (i.e., $d = 10$). Using class probabilities (rather than the most likely class) enables the network to perform more complex reasoning about the data (e.g., a person on a bicycle); we experimented with both and found this approach to yield better results.

The second stage architecture is also an encoder-decoder with skip connections, with two heads for classification and bounding box regression. We use 3×3 2D convolution/deconvolution in all layers, which again yields fast processing times. The encoder starts with two parallel input blocks for semantics and height data. Each block has 4 layers with 16, 16, 32, and 32 filters, respectively. The last layer in each input block downsamples by 2 spatially, and the resulting tensors are concatenated along the depth dimension and fed to the encoder consisting of 3 blocks. Each block has 2 layers, where each second layer downsamples the input by 2 spatially. The blocks have 64, 128, 256 filters, respectively. The decoder consists of 2 blocks. Each block has a deconvolution (upsample by 2) followed by a convolution layer. Blocks have 128 and 64 filters, respectively. We add skip connections to each deconvolution from the corresponding encoder layer. The pixel classification head has 3 layers with 64, 32, and n_c number of filters, while the bounding box parameters regression head has 3 layers with 64, 32, and n_r number of filters (where n_c and n_r are described below). We use batch normalization followed by ReLU activation for each layer except the task head layers.

The final output consists of a 256×256 array of n_c-element vectors containing the class distribution ($n_c = 4$: car, pedestrian, cyclist, and unknown), along with another 256×256 array of n_r-element vectors ($n_r = 6$), each containing $[\delta_x, \delta_y, w_o, \ell_o, \sin\theta, \cos\theta]$, where (δ_x, δ_y) points toward the centroid of the corresponding object, $w_o \times \ell_o$ are the object dimensions, and θ is the orientation in the top-down view. By representing the object dimensions in this parameterized manner, our vehicle detection includes not only cars but also buses and trucks. The stage is trained with a loss that combines focal loss [18] for the classification head and L1 loss for the regression head, with corresponding weights.

A clustering algorithm is applied to the output of the second stage to detect individual instances of objects. For clustering, we use the DBSCAN algorithm applied to the regressed centroids using only the cells whose class confidence produced by the classification head exceeds a threshold. Individual bounding box coordinates, dimensions, and orientations are averaged within each cluster.

Note that the simplicity of the proposed approach makes it easy to implement and computationally efficient. The method leverages the best of both worlds: The perspective view captures rich shape information that allows semantic understanding of the scene, while the top-down view allows metric reasoning without the difficulties in handling occlusion. Both views are processed using only 2D convolutions, which are much faster than 3D convolutions that are commonly used with voxelized volumes.

The two stages can be trained either independently or together. We train them separately due to the non-overlapping nature of existing segmentation and object detection datasets (e.g., SemanticKITTI [3] and KITTI [15], [16]). If, however, appropriately labeled data were available, both stages could be trained together end-to-end.

IV. EXPERIMENTAL RESULTS

In this section we show that our simple approach, which consists of a single multi-class network trained on data from a single Velodyne HDL-64E, is able to achieve competitive results with an order of magnitude less computation than competing approaches. Moreover, we show results detecting vehicles and pedestrians in crowded scenes containing more than one hundred objects in each frame. (We do not show cyclist results, as the datasets do not contain enough cyclists to train without data augmentation.) As mentioned above, the two stages are trained independently due to differences in segmentation and object detection datasets (e.g. KITTI vs. SemanticKITTI). The first segmentation stage is trained to segment class masks only and uses segmentation labels either from labeled LiDAR point clouds directly or segmentation labels transferred from camera to LiDAR. The second stage is trained on LiDAR bounding box labels.

The input LiDAR data is motion compensated for training and inference. The input/output resolution of the first stage is set to 64×2048. The input resolution of the second stage is set to $w = 1024$ and $\ell = 1024$ as a compromise between spatial resolution, cell occupancy, and computational load. It covers an 80×80 m^2 area and yields a cell resolution of 7.8 cm per cell. The output of the second stage is set to 256×256, each output cell has a spatial resolution of 31.3 cm.

We use the Adam optimizer with initial learning rate set to 10^{-4}. For the second stage loss, we set the class and regression weights to 5.0 and 1.0, respectively. Both stages are trained for 40-50 epochs with batch size 4. The resulting model is exported to TensorRT for inference on the vehicle using an automotive Drive AGX embedded computer.

A. Semantic segmentation

Table I provides comparisons of our first stage DNN segmentation results with RangeNet LiDAR segmentation DNN on SemanticKITTI [3] dataset. Our network runs much faster while providing similar accuracy. At 64×2048

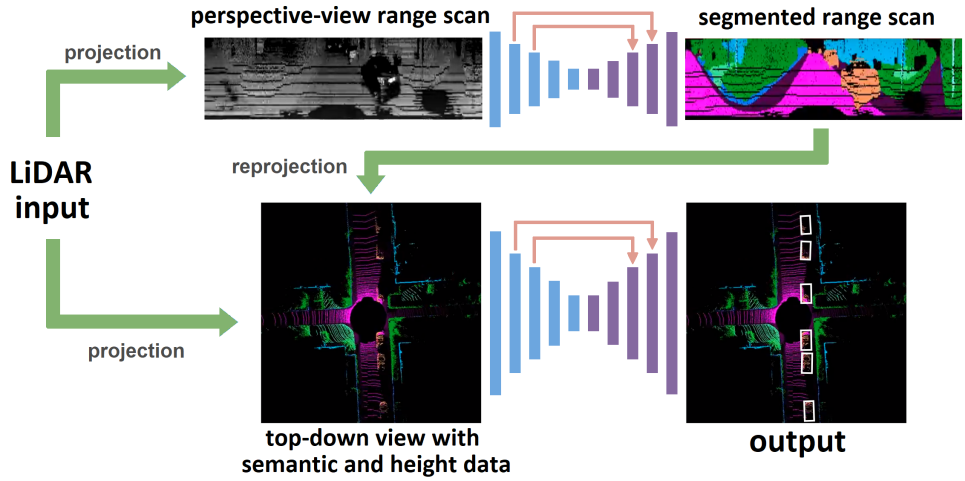

Fig. 1. Our proposed MVLidarNet is a neural network with two stages. The first stage performs semantic segmentation (including drivable space) on the LiDAR input after projecting to a perspective view. The second stage uses the output of the first stage reprojected to a top-down view, along with the LiDAR input height map, to detect dynamic objects. Both stages are feature pyramid networks (FPNs).

TABLE I

LiDAR segmentation results for SemanticKITTI [3] dataset, comparing our segmentation network with RangeNet++ [14]. Shown are the IoU scores for different classes and speed. The six rows use, respectively, input sizes of S, L, S, L, L, and L, respectively, where S means 64×1024 and L means 64×2048.

Method	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	mean IoU	speed (fps)
RangeNet53	84.6	20.0	25.3	24.8	17.3	27.5	27.7	7.1	90.4	51.8	72.1	22.8	80.4	50.0	75.1	46.0	62.7	33.4	43.4	45.4	25
RangeNet53	86.4	24.5	32.7	25.5	22.6	36.2	33.6	4.7	91.8	64.8	74.6	27.9	84.1	55.0	78.3	50.1	64.0	38.9	52.2	49.9	13
RangeNet53++	90.3	20.6	27.1	25.2	17.6	29.6	34.2	7.1	90.4	52.3	72.7	22.8	83.9	53.3	77.7	52.5	63.7	43.8	47.2	48.0	21
RangeNet53++	91.4	25.7	34.4	25.7	23.0	38.3	38.8	4.8	91.8	65.0	75.2	27.8	87.4	58.6	80.5	55.1	64.6	47.9	55.9	52.2	12
Ours	86.3	33.8	34.2	24.0	25.4	44.0	41.8	23.0	90.3	56.5	72.6	19.4	83.0	51.2	79.0	54.9	63.4	41.9	52.8	51.5	200
Ours++	87.1	34.9	32.9	23.7	24.9	44.5	44.3	23.1	90.3	56.7	73.0	19.1	85.6	53.0	80.9	59.4	63.9	49.9	51.1	52.5	92

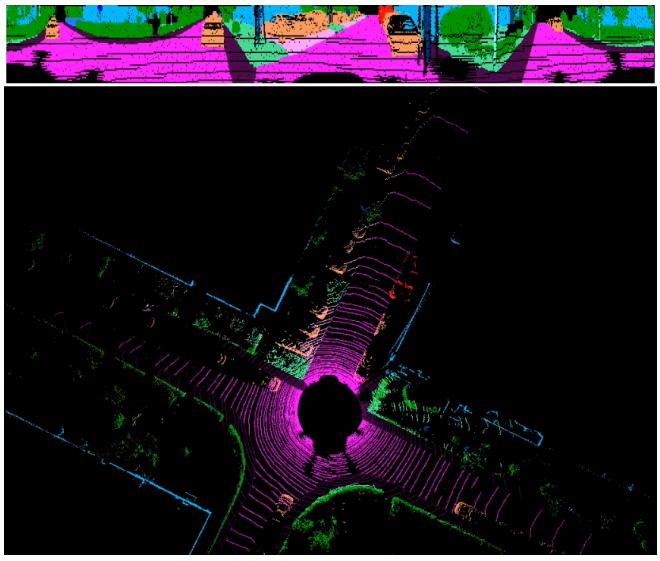

Fig. 2. Top: Output of the first stage segmentation DNN on a frame of the KITTI dataset. Bottom: BEV reprojection of the semantic segmentation and height data from the first stage, used as input to the second stage.

resolution, our end-to-end runtime is 2.1 ms in FP16 and 4.9 ms in FP32 on an embedded dGPU on NVIDIA Drive AGX computer. This is 200 frames per second, compared with RangeNet's best result of 13 frame per second at this resolution in FP32 mode. Our segmentation DNN runs at 480 frames per second in FP16 mode. We achieve this speed by using a simpler and shallower network structure based on the Inception architecture. The ++ addition to methods in Table I are for using extra post-processing as described in RangeNet++ [14].

B. Object detection

Table II provides comparisons of our full DNN output for bird's eye view (BEV) object detection on KITTI [15], [16] dataset for cars. Our network runs much faster than other DNNs while providing competitive accuracy. Our end-to-end (two stages combined) runtime is 6.8 ms per frame (corresponding to about 150 fps) on an embedded dGPU (NVIDIA Drive AGX computer).

Since the KITTI dataset does not have many pedestrian instances for training (4487 instances in the object detection set), PointPillars [10] introduced a set of augmentation techniques to improve AP scores on KITTI. Such techniques are

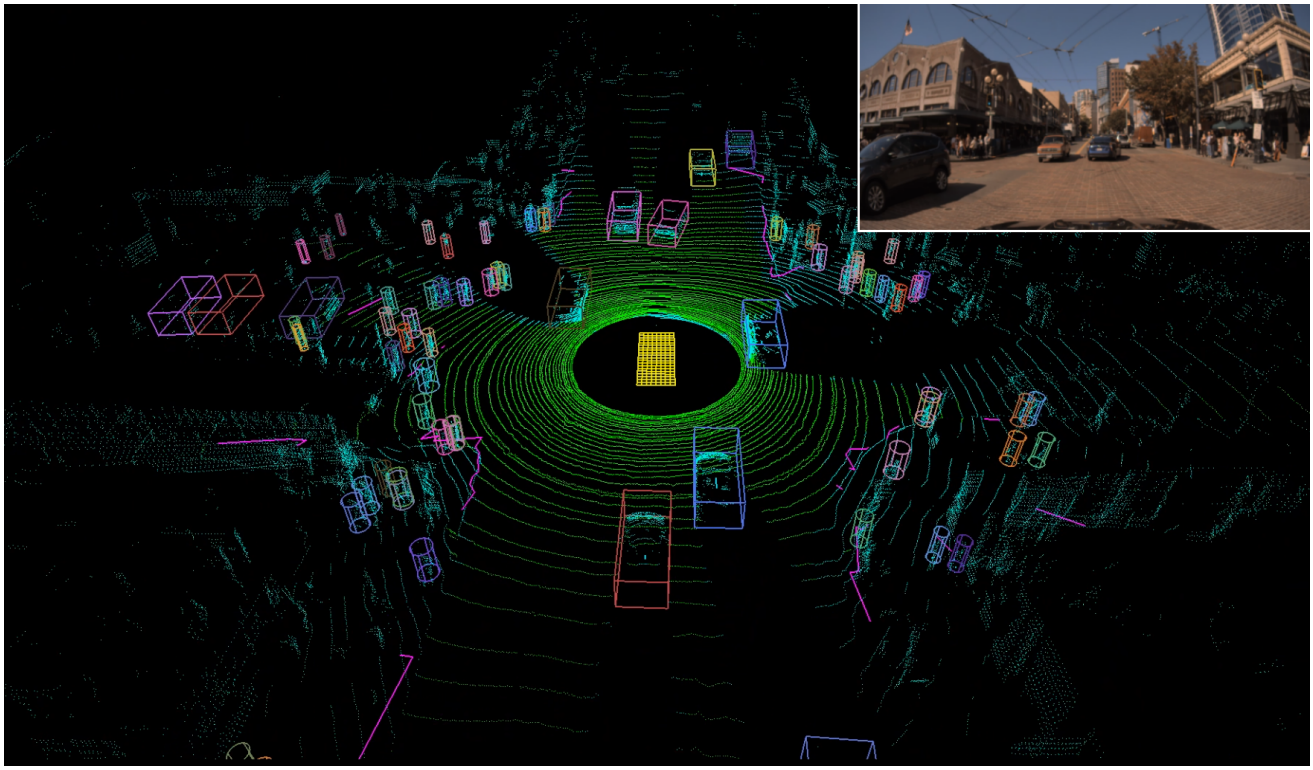

Fig. 3. End-to-end object detection and segmentation results for vehicles (cars, buses, and trucks), pedestrians, drivable space (green) on a crowded urban scene. 133 objects were detected in this frame: vehicles (boxes) and pedestrians (cylinders). The different colors indicate different instances, which are tracked over time for label consistency.

TABLE II

EVALUATION ON KITTI [15], [16] BEV OBJECT DETECTION BENCHMARK (TEST SPLIT). SHOWN ARE AP SCORES AT 0.7 IOU FOR CARS.

Method	Modality	Easy	Mod.	Hard	Speed (ms)
MV3D [6]	RGB + LiDAR	86.02	76.90	68.49	240
ContFuse [19]	RGB + LiDAR	88.81	85.83	77.33	60
AVOD-FPN [7]	RGB + LiDAR	88.53	83.79	77.90	100
F-PointNet [20]	RGB + LiDAR	88.70	84.00	75.33	170
MMF [21]	RGB + LiDAR	89.49	87.47	79.10	80
VoxelNet [4]	LiDAR only	89.35	79.26	77.39	500
SECOND [5]	LiDAR only	88.07	79.37	77.95	40
PointPillars [10]	LiDAR only	88.35	86.10	79.83	16
PointRCNN [22]	LiDAR only	89.47	85.68	79.10	100
Part-A^2 [23]	LiDAR only	89.52	84.76	81.47	80
STD [12]	LiDAR only	89.66	87.76	86.89	80
Ours	LiDAR only	89.27	80.59	70.90	7

TABLE III

EVALUATION ON OUR LARGE INTERNAL LIDAR OBJECT DETECTION DATASET. SHOWN ARE AP SCORES FOR BEV DETECTIONS AT VARIOUS RANGES FOR VEHICLES (CARS, TRUCKS, BUSES) AND PEDESTRIANS. WE COMPARE OUR TWO STAGE SEMANTIC SEGMENTATION + TOP-DOWN DETECTION WITH A TOP-DOWN APPROACH RELYING ON HEIGHT INFORMATION ALONE. OUR SEMANTICS+HEIGHT MVLIDARNET APPROACH ACHIEVES SIGNIFICANTLY BETTER RESULTS.

Method	Class	IoU	0–10 m	10–25 m	25–50 m
Top-down height only	Vehicles	0.7	96.45	91.52	77.48
Top-down height only	Pedestrians	0.5	51.75	48.20	28.67
Perspective semantics + top-down height	Vehicles	0.7	96.80	91.20	77.77
Perspective semantics + top-down height	Pedestrians	0.5	72.29	59.01	39.17
Perspective semantics + top-down height (VLS128)	Vehicles	0.7	97.48	95.39	86.77
Perspective semantics + top-down height (VLS128)	Pedestrians	0.5	88.89	61.27	48.34

not reflective of the actual data and therefore potentially bias the network since they do not capture real LiDAR geometry. As a result, we do not use these augmentation techniques. We only use horizontal flips and global rotations. Nevertheless, our approach still achieves competitive results due to the simple network architecture and semantic segmentation.

Our internal LiDAR dataset is much larger than KITTI (hundreds of thousands of scans) and includes more challenging scenes with crowds of pedestrians—nearly a hundred per frame in many cases (see Figure 3). In total the dataset has 123,195 pedestrians. Table III provides AP scores computed on this internal dataset for vehicles and pedestrians. Note that these scores are from a single multi-class network, *i.e.*, we do not train separately for vehicles and pedestrians but rather jointly. It also shows an ablation study that compares our full network (two stages that use semantic segmentation and top-down detection) with just a top-down network that relies only on height data (*i.e.*, the second stage was modified to take only height data as input). Using two stages with both semantics and height on pedestrian detection clearly shows significant improvement in the results. Adding semantic segmentation helps with object detection on LiDAR point clouds.

An example of our multi-class detection is shown on Figure 3. Note that in addition to detected dynamic objects, our system marks segmented drivable space (in green color vs. other LiDAR points in cyan). Currently, we only use drivable space, but with sufficient training data, other scene elements like sidewalks, trees, buildings, and poles provided by the first stage DNN could be added. Such semantic features are not possible with standard object detection DNNs.

V. CONCLUSION

We have presented a multi-class, multi-view DNN that simultaneously detects dynamic objects (vehicles and pedestrians) and segment the drivable space from LiDAR point cloud data. Our DNN consists of two stages: 1) one stage that semantically labels the points in the LiDAR range scan in a perspective view, and 2) another stage that detects objects using semantically segmented points reprojected onto a top-down bird's eye view (BEV), combined with height data from the LiDAR point cloud. The combination of these two stages allow improved detection of vulnerable road users (such as pedestrians) due to the easily discernable shape information present in the perspective view. We show that this simple architecture achieves results that are competitive with state-of-the-art despite not relying on complex data augmentation schemes used by previous techniques. Moreover, we show results on crowded scenes with unprecedented complexity, namely, more than a hundred objects (vehicles and pedestrians) in a single frame. Our system is very fast (6.8 ms per scan on an embedded dGPU) due to the use of shallow architectures and 2D convolutions. Note that this is faster than previous methods by at least 3X, even though our system performs multi-class detection on an embedded computer whereas previous systems perform single-class detection (*i.e.*, one at a time) on a desktop GPU. Future work includes training the two stages end-to-end with a combined segmentation and object detection dataset, experimenting with different height and semantics encodings, and extending the number of supported classes.

REFERENCES

[1] W. Luo, B. Yang, and R. Urtasun, "Fast and furious: Real time end-to-end 3D detection, tracking and motion forecasting with a single convolutional net," in *CVPR*, 2018.

[2] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D object detection from point clouds," in *CVPR*, 2018.

[3] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A dataset for semantic scene understanding of lidar sequences," in *ICCV*, 2019.

[4] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3D object detection," in *CVPR*, 2018.

[5] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, 2018.

[6] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *CVPR*, 2017.

[7] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander, "Joint 3D proposal generation and object detection from view aggregation," *IROS*, 2018.

[8] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017.

[9] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *CVPR*, 2017.

[10] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *CVPR*, 2019.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. Berg, "Ssd: Single shot multibox detector," in *ECCV*, 2016.

[12] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "STD: Sparse-to-dense 3D object detector for point cloud," in *arXiv:1907.10471*, 2019.

[13] Y. Zhou, P. Sun, Y. Zhang, D. Anguelov, J. Gao, T. Ouyang, J. Guo, J. Ngiam, and V. Vasudevan, "End-to-end multi-view fusion for 3d object detection in lidar point clouds," in *CoRL*, 2019.

[14] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "RangeNet++: Fast and accurate lidar semantic segmentation," in *IROS*, 2019.

[15] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *CVPR*, 2012.

[16] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *IJRR*, 2013.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.

[18] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *ICCV*, 2017.

[19] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3D object detection," in *ECCV*, 2018.

[20] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D object detection from RGB-D data," in *CVPR*, 2018.

[21] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-task multi-sensor fusion for 3D object detection," in *CVPR*, 2019.

[22] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *CVPR*, 2019.

[23] S. Shi, Z. Wang, X. Wang, and H. Li, "Part-A^2 Net: 3D part-aware and aggregation neural network for object detection from point cloud," in *arXiv:1907.03670*, 2019.