SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud

Bichen Wu, Alvin Wan, Xiangyu Yue and Kurt Keutzer
UC Berkeley
{bichen, alvinwan, xyyue, keutzer}@berkeley.edu

***Abstract*— In this paper, we address semantic segmentation of road-objects from 3D LiDAR point clouds. In particular, we wish to detect and categorize instances of interest, such as cars, pedestrians and cyclists. We formulate this problem as a point-wise classification problem, and propose an end-to-end pipeline called SqueezeSeg based on convolutional neural networks (CNN): the CNN takes a transformed LiDAR point cloud as input and directly outputs a point-wise label map, which is then refined by a conditional random field (CRF) implemented as a recurrent layer. Instance-level labels are then obtained by conventional clustering algorithms. Our CNN model is trained on LiDAR point clouds from the KITTI [1] dataset, and our point-wise segmentation labels are derived from 3D bounding boxes from KITTI. To obtain extra training data, we built a LiDAR simulator into *Grand Theft Auto V (GTA-V)*, a popular video game, to synthesize large amounts of realistic training data. Our experiments show that SqueezeSeg achieves high accuracy with astonishingly fast and stable runtime (8.7 ± 0.5 ms per frame), highly desirable for autonomous driving applications. Furthermore, additionally training on synthesized data boosts validation accuracy on real-world data. Our source code and synthesized data will be open-sourced.**

I. INTRODUCTION

Autonomous driving systems rely on accurate, real-time and robust perception of the environment. An autonomous vehicle needs to accurately categorize and locate "road-objects", which we define to be driving-related objects such as cars, pedestrians, cyclists, and other obstacles. Different autonomous driving solutions may have different combinations of sensors, but the 3D LiDAR scanner is one of the most prevalent components. LiDAR scanners directly produce distance measurements of the environment, which are then used by vehicle controllers and planners. Moreover, LiDAR scanners are robust under almost all lighting conditions, whether it be day or night, with or without glare and shadows. As a result, LiDAR based perception tasks have attracted significant research attention.

In this work, we focus on road-object segmentation using (Velodyne style) 3D LiDAR point clouds. Given point cloud output from a LiDAR scanner, the task aims to isolate objects of interest and predict their categories, as shown in Fig. 1. Previous approaches comprise or use parts of the following stages: Remove the ground, cluster the remaining points into instances, extract (hand-crafted) features from each cluster, and classify each cluster based on its features. This paradigm, despite its popularity [2], [3], [4], [5], has several disadvantages: a) Ground segmentation in the above pipeline usually relies on hand-crafted features or decision rules – some approaches rely on a scalar threshold [6] and others require more complicated features such as surface normals [7] or invariant descriptors [4], all of which may fail to generalize and the latter of which require significant preprocessing. b) Multi-stage pipelines see aggregate effects of compounded errors, and classification or clustering algorithms in the pipeline above are unable to leverage context, most importantly the immediate surroundings of an object. c) Many approaches for ground removal rely on iterative algorithms such as RANSAC (random sample consensus) [5], GP-INSAC (Gaussian Process Incremental Sample Consensus) [2], or agglomerative clustering [2]. The runtime and accuracy of these algorithmic components depend on the quality of random initializations and, therefore, can be unstable. This instability is not acceptable for many embedded applications such as autonomous driving. We take an alternative approach: use deep learning to extract features, develop a single-stage pipeline and thus sidestep iterative algorithms.

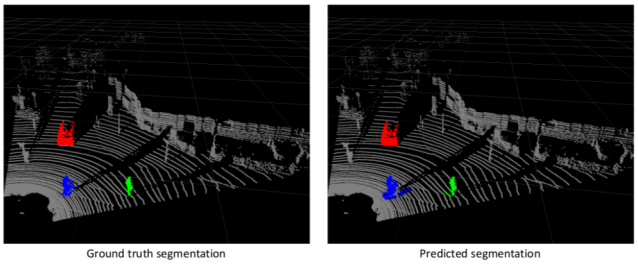

Fig. 1: An example of SqueezeSeg segmentation results. Our predicted result is on the right and the ground truth is on the left. Cars are annotated in red, pedestrians in green and cyclists in blue.

In this paper, we propose an end-to-end pipeline based on convolutional neural networks (CNN) and conditional random field (CRF). CNNs and CRFs have been successfully applied to segmentation tasks on 2D images [8], [9], [10], [11]. To apply CNNs to 3D LiDAR point clouds, we designed a CNN that accepts transformed LiDAR point clouds and outputs a point-wise map of labels, which is further refined by a CRF model. Instance-level labels are then obtained by applying conventional clustering algorithms (such as DBSCAN) on points within a category. To feed 3D point

clouds to a 2D CNN, we adopt a spherical projection to transform sparse, irregularly distributed 3D point clouds to dense, 2D grid representations. The proposed CNN model draws inspiration from SqueezeNet [12] and is carefully designed to reduce parameter size and computational complexity, with an aim to reduce memory requirements and achieve real-time inference speed for our target embedded applications. The CRF model is reformulated as a recurrent neural network (RNN) module as [11] and can be trained end-to-end together with the CNN model. Our model is trained on LiDAR point clouds from the KITTI dataset [1] and point-wise segmentation labels are converted from 3D bounding boxes in KITTI. To obtain even more training data, we leveraged *Grand Theft Auto V (GTA-V)* as a simulator to retrieve LiDAR point clouds and point-wise labels.

Experiments show that SqueezeSeg achieves high accuracy and is extremely fast and stable, making it suitable for autonomous driving applications. We additionally find that supplanting our dataset with artificial, noise-injected simulation data further boosts validation accuracy on real-world data.

II. Related Work

A. Semantic segmentation for 3D LiDAR point clouds

Previous work saw a wide range of granularity in LiDAR segmentation, handling anything from specific components to the whole pipeline. [7] proposed mesh based ground and object segmentation relying on local surface convexity conditions. [2] summarized several approaches based on iterative algorithms such as RANSAC (random sample consensus) and GP-INSAC (gaussian process incremental sample consensus) for ground removal. Recent work also focused on algorithmic efficiency. [5] proposed efficient algorithms for ground segmentation and clustering while [13] bypassed ground segmentation to directly extract foreground objects. [4] expanded its focus to the whole pipeline, including segmentation, clustering and classification. It proposed to directly classify point patches into background and foreground objects of different categories then use EMST-RANSAC [5] to further cluster instances.

B. CNN for 3D point clouds

CNN approaches consider LiDAR point clouds in either two or three dimensions. Work with two-dimensional data considers raw images with projections of LiDAR point clouds top-down [14] or from a number of other views [15]. Other work considers three-dimensional data itself, discretizing the space into voxels and engineering features such as disparity, mean, and saturation [16]. Regardless of data preparation, deep learning methods consider end-to-end models that leverage 2D convolutional [17] or 3D convolutional [18] neural networks.

C. Semantic Segmentation for Images

Both CNNs and CRFs have been applied to semantic segmentation tasks for images. [8] proposed transforming CNN models, trained for classification, to fully convolutional networks to predict pixel-wise labels. [9] proposed a CRF formulation for image segmentation and solved it approximately with the mean-field iteration algorithm. CNNs and CRFs are combined in [10], where the CNN is used to produce an initial probability map and the CRF is used to refine and restore details. In [11], mean-field iteration is reformulated as a recurrent neural network (RNN) module.

D. Data Collection through Simulation

Obtaining annotations, especially point-wise or pixel-wise annotations for computer vision tasks is usually very difficult. As a consequence, synthetic datasets have seen growing interest. In the autonomous driving community, the video game Grand Theft Auto has been used to retrieve data for object detection and segmentation [19], [20].

III. Method Description

A. Point Cloud Transformation

Conventional CNN models operate on images, which can be represented by 3-dimentional tensors of size $H \times W \times 3$. The first two dimensions encode spatial position, where H and W are the image height and width, respectively. The last dimension encodes features, most commonly RGB values. However, a 3D LiDAR point cloud is usually represented as a set of cartesian coordinates, (x, y, z). Extra features can also be included, such as intensity or RGB values. Unlike the distribution of image pixels, the distribution of LiDAR point clouds is usually sparse and irregular. Therefore, naively discretizing a 3D space into voxels results in excessively many empty voxels. Processing such sparse data is inefficient, wasting computation.

To obtain a more compact representation, we project the LiDAR point cloud onto a sphere for a dense, grid-based representation as

$$\begin{aligned} \theta &= \arcsin \frac{z}{\sqrt{x^2+y^2+z^2}}, \ \tilde{\theta} = \lfloor \theta / \triangle\theta \rfloor, \\ \phi &= \arcsin \frac{y}{\sqrt{x^2+y^2}}, \ \tilde{\phi} = \lfloor \phi / \triangle\phi \rfloor. \end{aligned} \tag{1}$$

ϕ and θ are *azimuth* and *zenith* angles, as shown in Fig. 2 (A). $\triangle\theta$ and $\triangle\phi$ are resolutions for discretization and $(\tilde{\theta}, \tilde{\phi})$ denotes the position of a point on a 2D spherical grid. Applying equation (1) to each point in the cloud, we can obtain a 3D tensor of size $H \times W \times C$. In this paper, we consider data collected from a Velodyne HDL-64E LiDAR with 64 vertical channels, so $H = 64$. Limited by data annotations from the KITTI dataset, we only consider the front view area of $90°$ and divide it into 512 grids so $W = 512$. C is the number of features for each point. In our experiments, we used 5 features for each point: 3 cartesian coordinates (x, y, z), an intensity measurement and range $r = \sqrt{x^2+y^2+z^2}$. An example of a projected point cloud can be found at Fig. 2 (B). As can be seen, such representation is dense and regularly distributed, resembling an ordinary image Fig. 2 (C). This featurization allows us to avoid hand-crafted features, bettering the odds that our representation generalizes.

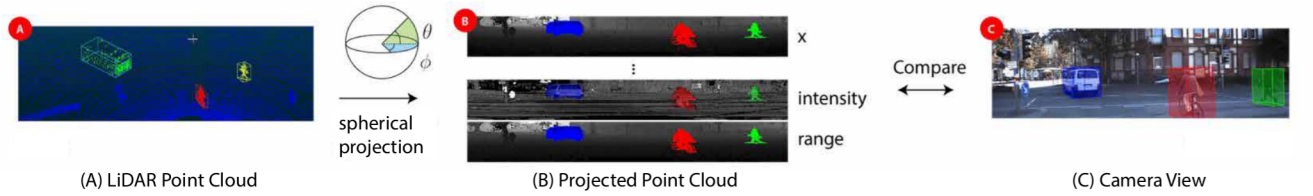

Fig. 2: LiDAR Projections. Note that each channel reflects structural information in the camera-view image.

B. Network structure

Our convolutional neural network structure is shown in Fig. 3. SqueezeSeg is derived from SqueezeNet [12], a light-weight CNN that achieved AlexNet [21] level accuracy with 50X fewer parameters.

The input to *SqueezeSeg* is a $64 \times 512 \times 5$ tensor as described in the previous section. We ported layers (*conv1a* to *fire9*) from SqueezeNet for feature extraction. SqueezeNet used *max-pooling* to down-sample intermediate feature maps in both width and height dimensions, but since our input tensor's height is much smaller than its width, we only down-sample the width. The output of *fire9* is a down-sampled feature map that encodes the semantics of the point cloud.

To obtain full resolution label predictions for each point, we used deconvolution modules (more precisely, "transposed convolutions") to up-sample feature maps in the width dimension. We used skip-connections to add up-sampled feature maps to lower-level feature maps of the same size, as shown in Fig. 3. The output probability map is generated by a convolutional layer (*conv14*) with *softmax* activation. The probability map is further refined by a recurrent CRF layer, which will be discussed in the next section.

In order to reduce the number of model parameters and computation, we replaced convolution and deconvolution layers with *fireModule*s [12] and *fireDeconv*s. The architecture of both modules are shown in Fig. 4. In a *fireModule*, the input tensor of size $H \times W \times C$ is first fed into a 1x1 convolution to reduce the channel size to $C/4$. Next, a 3x3 convolution is used to fuse spatial information. Together with a parallel 1x1 convolution, they recover the channel size of C. The input 1x1 convolution is called the *squeeze* layer and the parallel 1x1 and 3x3 convolution together is called the *expand* layer. Given matching input and output size, a 3x3 convolutional layer requires $9C^2$ parameters and $9HWC^2$ computations, while the *fireModule* only requires $\frac{3}{2}C^2$ parameters and $\frac{3}{2}HWC^2$ computations. In a *fireDeconv* module, the deconvolution layer used to up-sample the feature map is placed between *squeeze* and *expand* layers. To up-sample the width dimension by 2, a regular 1x4 deconvolution layer must contain $4C^2$ parameters and $4HWC^2$ computations. With the *fireDeconv* however, we only need $\frac{7}{4}C^2$ parameters and $\frac{7}{4}HWC^2$ computations.

C. Conditional Random Field

With image segmentation, label maps predicted by CNN models tend to have blurry boundaries. This is due to loss of low-level details in down-sampling operations such as max-pooling. Similar phenomena are also observed with SqueezeSeg.

Accurate point-wise label prediction requires understanding not only the high-level semantics of the object and scene but also low-level details. The latter are crucial for the consistency of label assignments. For example, if two points in the cloud are next to each other and have similar intensity measurements, it is likely that they belong to the same object and thus have the same label. Following [10], we used a conditional random field (CRF) to refine the label map generated by the CNN. For a given point cloud and a label prediction $\mathbf{c}$ where c_i denotes the predicted label of the i-th point, a CRF model employs the energy function

$$E(\mathbf{c}) = \sum_i u_i(c_i) + \sum_{i,j} b_{i,j}(c_i, c_j). \tag{2}$$

The unary potential term $u_i(c_i) = -\log P(c_i)$ considers the predicted probability $P(c_i)$ from the CNN classifier. The binary potential terms define the "penalty" for assigning different labels to a pair of similar points and is defined as $b_{i,j}(c_i, c_j) = \mu(c_i, c_j) \sum_{m=1}^{M} w_m k^m(\mathbf{f_i}, \mathbf{f_j})$ where $\mu(c_i, c_j) = 1$ if $c_i \neq c_j$ and 0 otherwise, k^m is the m-th Gaussian kernel that depends on features $\mathbf{f}$ of point i and j and w_m is the corresponding coefficient. In our work, we used two Gaussian kernels

$$\begin{aligned} w_1 \exp\left(-\frac{\|\mathbf{p_i} - \mathbf{p_j}\|^2}{2\sigma_\alpha^2} - \frac{\|\mathbf{x_i} - \mathbf{x_j}\|^2}{2\sigma_\beta^2}\right) \\ + w_2 \exp\left(-\frac{\|\mathbf{p_i} - \mathbf{p_j}\|^2}{2\sigma_\gamma^2}\right). \end{aligned} \tag{3}$$

The first term depends on both angular position $\mathbf{p}(\tilde{\theta}, \tilde{\phi})$ and cartesian coordinates $\mathbf{x}(x, y, z)$ of two points. The second term only depends on angular positions. σ_α, σ_β and σ_γ are three hyper parameters chosen empirically. Extra features such as intensity and RGB values can also be included.

Minimizing the above CRF energy function yields a refined label assignment. Exact minimization of equation (2) is intractable, but [9] proposed a mean-field iteration algorithm to solve it approximately and efficiently. [11] reformulated the mean-field iteration as a recurrent neural network (RNN). We refer readers to [9] and [11] for a detailed derivation of the mean-field iteration algorithm and its formulation as an RNN. Here, we provide just a brief description of our implementation of the mean-field iteration as an RNN module as shown in Fig. 5. The output of the CNN model is fed into the CRF module as the initial probability map. Next, we compute Gaussian kernels based

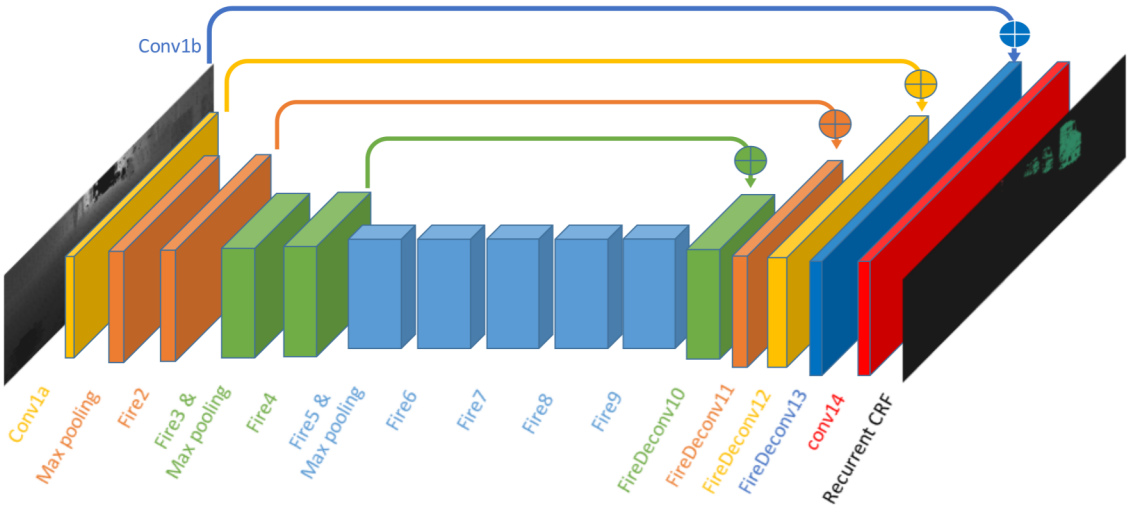

Fig. 3: Network structure of SqueezeSeg.

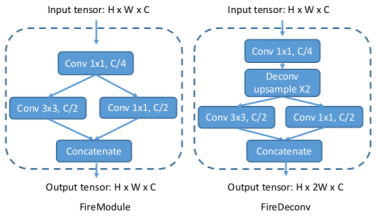

Fig. 4: Structure of a *FireModule* (left) and a *fireDeconv* (right).

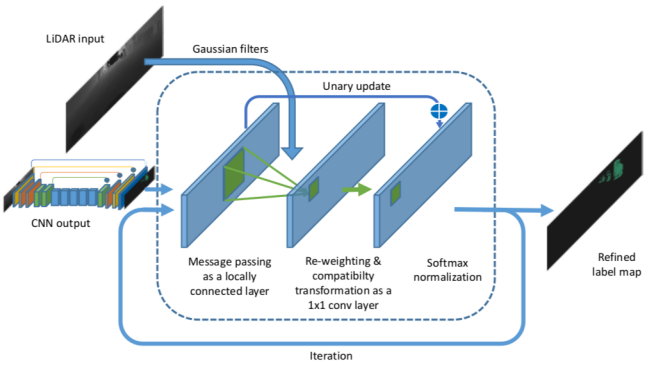

Fig. 5: Conditional Random Field (CRF) as an RNN layer.

on the input feature as equation (3). The value of above Gaussian kernels drop very fast as the distance (in the 3D cartesian space and the 2D angular space) between two points increases. Therefore, for each point, we limit the kernel size to a small region of 3×5 on the input tensor. Next, we filter the initial probability map using above Gaussian kernels. This step is also called message passing in [11] since it essentially aggregates probabilities of neighboring points. This step can be implemented as a locally connected layer with above Guassian kernels as parameters. Next, we re-weight the aggregated probability and use a "compatibilty transformation" to decide how much it changes each point's distribution. This step can be implemented as a 1x1 convolution whose parameters are learned during training. Next, we update the initial probability by adding it to the output of the 1x1 convolution and use *softmax* to normalize it. The output of the module is a refined probability map, which can be further refined by applying this procedure iteratively. In our experiment, we used 3 iterations to achieve an accurate label map. This recurrent CRF module together with the CNN model can be trained together end-to-end. With a single stage pipeline, we sidestep the thread of propagated errors present in multi-stage workflows and leverage contextual information accordingly.

D. Data collection

Our initial data is from the KITTI raw dataset, which provides images, LiDAR scans and 3D bounding boxes organized in sequences. Point-wise annotations are converted from 3D bounding boxes. All points within an object's 3D bounding box are considered part of the target object. We then assign the corresponding label to each point. An example of such a conversion can be found in Fig. 2 (A, B). Using this approach, we collected 10,848 images with point-wise labels.

In order to obtain more training samples (both point clouds and point-wise labels), we built a LiDAR simulator in GTA-V. The framework of the simulator is based on DeepGTAV[1], which uses Script Hook V[2] as a plugin.

We mounted a virtual LiDAR scanner atop an in-game car, which is then set to drive autonomously. The system collects both LiDAR point clouds and the game screen. In our setup, the virtual LiDAR and game camera are placed at the same position, which offers two advantages: First, we

[1]https://github.com/ai-tor/DeepGTAV
[2]http://www.dev-c.com/gtav/scripthookv/

can easily run sanity checks on the collected data, since the points and images need to be consistent. Second, the points and images can be exploited for other research projects, e.g. sensor fusion, etc.

We use ray casting to simulate each laser ray that LiDAR emits. The direction of each laser ray is based on several parameters of the LiDAR setup: vertical field of view (FOV), vertical resolution, pitch angle, and the index of the ray in the point cloud scan. Through a series of APIs, the following data associated with each ray can be obtained: a) the coordinates of the first point the ray hits, b) the class of the object hit, c) the instance ID of the object hit (which is useful for instance-wise segmentation, etc.), d) the center and bounding box of the object hit.

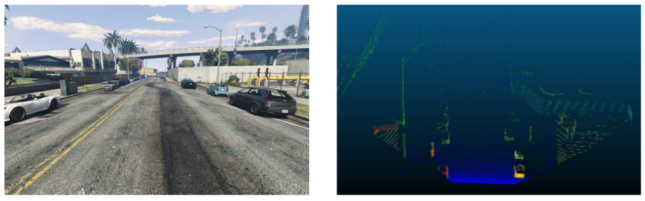

Fig. 6: Left: Image of game scene from GTA-V. Right: LiDAR point cloud corresponding to the game scene.

Using this simulator, we built a synthesized dataset with 8,585 samples, roughly doubling our training set size. To make the data more realistic, we further analyzed the distribution of noise across KITTI point clouds (Fig. 7). We took empirical frequencies of noise at each radial coordinate and normalized to obtain a valid probability distribution: 1) Let P_i be a 3D tensor in the format described earlier in Section III-A denoting the spherically projected "pixel values" of the i-th KITTI point cloud. For each of the n KITTI point clouds, consider whether or not the pixel at the $(\tilde{\theta}, \tilde{\phi})$ coordinate contains "noise." For simplicity, we consider "noise" to be missing data, where all pixel channels are zero. Then, the empirical frequency of noise at the $(\tilde{\theta}, \tilde{\phi})$ coordinate is

$$\epsilon(\tilde{\theta}, \tilde{\phi}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{P_i[\tilde{\theta}, \tilde{\phi}]=0\}}.$$

2) We can then augment the synthesized data using the distribution of noise in the KITTI data. For any point cloud in the synthetic dataset, at each $(\tilde{\theta}, \tilde{\phi})$ coordinate of the point cloud, we randomly add noise by setting all feature values to 0 with probability $\epsilon(\tilde{\theta}, \tilde{\phi})$.

It is worth noting that GTA-V used very simple physical models for pedestrians, often reducing people to cylinders. In addition, GTA-V does not encode a separate category for cyclists, instead marking people and vehicles separately on all accounts. For these reasons, we decided to focus on the "car" class for KITTI evaluation when training with our synthesized dataset.

IV. Experiments

A. Evaluation metrics

We evaluate our model's performance on both class-level and instance-level segmentation tasks. For class-level segmentation, we compare predicted with ground-truth labels, point-wise, and evaluate precision, recall and IoU (intersection-over-union) scores, which are defined as follows:

$$Pr_c = \frac{|\mathcal{P}_c \cap \mathcal{G}_c|}{|\mathcal{P}_c|}, recall_c = \frac{|\mathcal{P}_c \cap \mathcal{G}_c|}{|\mathcal{G}_c|}, IoU_c = \frac{|\mathcal{P}_c \cap \mathcal{G}_c|}{|\mathcal{P}_c \cup \mathcal{G}_c|},$$

where $\mathcal{P}_c$ and $\mathcal{G}_c$ respectively denote the predicted and ground-truth point sets that belong to class-c. $|\cdot|$ denotes the cardinality of a set. IoU score is used as the primary accuracy metric in our experiments.

For instance-level segmentation, we first match each predicted instance-i with a ground truth instance. This index matching procedure can be denoted as $\mathcal{M}(i) = j$ where $i \in \{1, \cdots, N\}$ is the predicted instance index and $j \in \{\emptyset, 1, \cdots, M\}$ is the ground truth index. If no ground truth is matched to instance-i, then we set $\mathcal{M}(i)$ to $\emptyset$. The matching procedure $\mathcal{M}(\cdot)$ 1) sorts ground-truth instances by number of points and 2) for each ground-truth instance, finds the predicted instance with the largest IoU. The evaluation script will be released together with the source code.

For each class-c, we compute instance-level precision, recall, and IoU scores as

$$Pr_c = \frac{\sum_i |\mathcal{P}_{i,c} \cap \mathcal{G}_{\mathcal{M}(i),c}|}{|\mathcal{P}_c|},$$
$$recall_c = \frac{\sum_i |\mathcal{P}_{i,c} \cap \mathcal{G}_{\mathcal{M}(i),c}|}{|\mathcal{G}_c|},$$
$$IoU_c = \frac{\sum_i |\mathcal{P}_{i,c} \cap \mathcal{G}_{\mathcal{M}(i),c}|}{|\mathcal{P}_c \cup \mathcal{G}_c|}.$$

$\mathcal{P}_{i,c}$ denotes the i-th predicted instance that belongs to class-c. Different instance sets are mutually exclusive, thus $\sum_i |\mathcal{P}_{i,c}| = |\mathcal{P}_c|$. Likewise for $\mathcal{G}_{\mathcal{M}(i),c}$. If no ground truth instance is matched with prediction-i, then $\mathcal{G}_{\mathcal{M}(i),c}$ is an empty set.

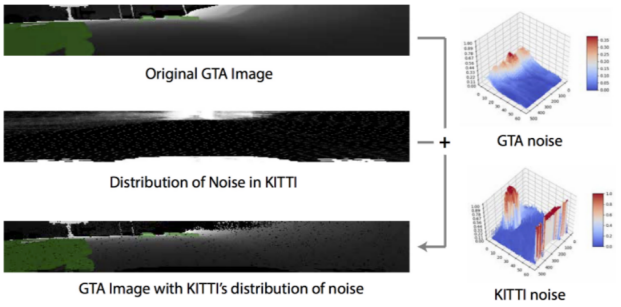

Fig. 7: Fixing distribution of noise in synthesized data

B. Experimental Setup

Our primary dataset is the converted KITTI dataset described above. We split the publicly available raw dataset into a training set with 8,057 frames and a validation set with 2,791 frames. Note that KITTI LiDAR scans can be temporally correlated if they are from the same sequence. In our split, we ensured that frames in the training set do

not appear in validation sequences. Our training/validation split will be released as well. We developed our model in Tensorflow [22] and used NVIDIA TITAN X GPUs for our experiments. Since the KITTI dataset only provides reliable 3D bounding boxes for front-view LiDAR scans, we limit our horizontal field of view to the forward-facing $90°$. Details of our model training protocols will be released in our source code.

C. Experimental Results

Segmentation accuracy of SqueezeSeg is summarized in Table.I. We compared two variations of SqueezeSeg, one with the recurrent CRF layer and one without. Although our proposed metric is very challenging–as a high IoU requires point-wise correctness–SqueezeSeg still achieved high IoU scores, especially for the car category. Note that both class-level and instance-level recalls for the car category are higher than 90%, which is desirable for autonomous driving, as false negatives are more likely to lead to accidents than false positives. We attribute lower performance on pedestrian and cyclist categories to two reasons: 1) there are many fewer instances of pedestrian and cyclist in the dataset. 2) Pedestrian and cyclist instances are much smaller in size and have much finer details, making it more difficult to segment.

By combining our CNN with a CRF, we increased accuracy (IoU) for the car category significantly. The performance boost mainly comes from improvement in precision since CRF better filters mis-classified points on the borders. At the same time, we also noticed that the CRF resulted in slightly worse performance for pedestrian and cyclist segmentation tasks. This may be due to lack of CRF hyperparameter tuning for pedestrians and cyclists.

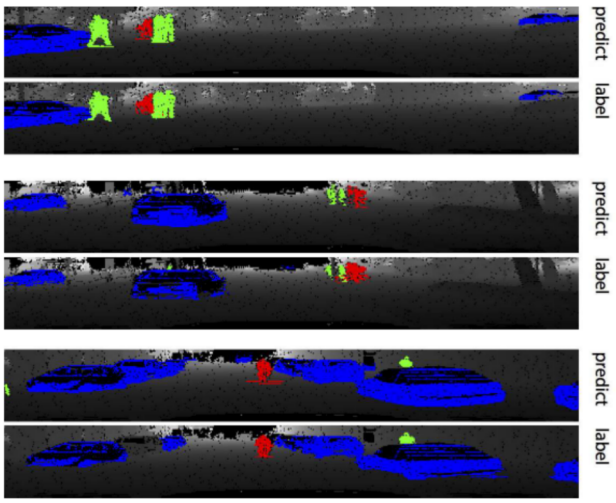

Fig. 8: Visualization of SqueezeSeg's prediction on a projected LiDAR depth map. For comparison, visualization of the ground-truth labels are plotted below the predicted ones. Notice that SqueezeSeg additionally and accurately segments objects that are unlabeled in ground truth.

TABLE I: Segmentation Performance of SqueezeSeg

		Class-level			Instance-level		
		P	R	**IoU**	P	R	**IoU**
car	w/ CRF	66.7	95.4	64.6	63.4	90.7	59.5
	w/o CRF	62.7	95.5	60.9	60.0	91.3	56.7
pedestrian	w/ CRF	45.2	29.7	21.8	43.5	28.6	20.8
	w/o CRF	52.9	28.6	22.8	50.8	27.5	21.7
cyclist	w/ CRF	35.7	45.8	25.1	30.4	39.0	20.6
	w/o CRF	35.2	51.1	26.4	30.1	43.7	21.7

Summary of SqueezeSeg's segmentation performance. *P, R, IoU* in the header row respectively represent precision, recall and intersection-over-union. IoU is used as the primary accuracy metric. All the values in this table are in percentages.

Runtime of two SqueezeSeg models are summarized in Table.II. On a TITAN X GPU, SqueezeSeg without CRF only takes 8.7 ms to process one LiDAR point cloud frame. Combined with a CRF layer, the model takes 13.5 ms each frame. This is much faster than the sampling rate of most LiDAR scanners today. The maximum rotation rate for Velodyne HDL-64E LiDAR, for example, is 20Hz. On vehicle embedded processors, where computational resources are more limited, SqueezeSeg comfortably allows trade-offs between speed and other practical concerns such as energy efficiency or processor cost. Also, note that the standard deviation of runtime for both SqueezeSeg models is very small, which is crucial for the stability of the entire autonomous driving system. However, our instance-wise segmentation currently relies on conventional clustering algorithms such as DBSCAN[3], which in comparison takes much longer and has much larger variance. A more efficient and stable clustering implementation is necessary, but it is out of the scope of this paper.

TABLE II: Runtime Performance of SqueezeSeg Pipeline

	Average runtime (ms)	Standard deviation (ms)
SqueezeSeg w/o CRF	8.7	0.5
SqueezeSeg	13.5	0.8
DBSCAN clustering	27.3	45.8

We tested our model's accuracy on KITTI data, when trained on GTA simulated data–the results of which are summarized in Table.III. Our GTA simulator is currently still limited in its ability to provide realistic labels for pedestrians and cyclists, so we consider only segmentation performance for cars. Additionally, our simulated point cloud does not contain intensity measurements; we therefore excluded intensity as an input feature to the network. To quantify the effects of training on synthesized data, we trained a SqueezeSeg model on the KITTI training set, without using intensity measurements, and validated on the KITTI validation set. The model's performance is shown in the first row of the table. Compared with Table.I, the IoU score is worse, due to the loss of the intensity channel. If we train the model completely on GTA simulated data, we see significantly

[3]We used the implementation from `http://scikit-learn.org/0.15/modules/generated/sklearn.cluster.DBSCAN.html`

worse performance. However, combining the KITTI training set with our GTA-simulated dataset, we see significantly increased accuracy that is even better than Table.I.

A visualization of the segmentation result by SqueezeSeg *vs.* ground truth labels can be found in Fig.8. For most of the objects, the predicted result is almost identical to the ground-truth, save for the ground beneath target objects. Also notice SqueezeSeg additionally and accurately segments objects that are unlabeled in ground truth. These objects may be obscured or too small, therefore placed in the "Don't Care" category for the KITTI benchmark.

TABLE III: Segmentation Performance on the Car Category with Simulated Data

	Class-level			Instance-level		
	P	R	**IoU**	P	R	**IoU**
KITTI	58.9	95.0	57.1	56.1	90.5	53.0
GTA	30.4	86.6	29.0	29.7	84.6	28.2
KITTI + GTA	69.6	92.8	66.0	66.6	88.8	61.4

V. CONCLUSIONS

We propose SqueezeSeg, an accurate, fast and stable end-to-end approach for road-object segmentation from LiDAR point clouds. Addressing the deficiencies of previous approaches that were discussed in the Introduction, our deep learning approach 1) does not rely on hand-crafted features, but utilizes convolutional filters learned through training; 2) uses a deep neural network and therefore has no reliance on iterative algorithms such as RANSAC, GP-INSAC, and agglomerative clustering; and 3) reduces the pipeline to a single stage, sidestepping the issue of propagated errors and allowing the model to fully leverage object context. The model accomplishes very high accuracy at faster-than-real-time inference speeds with small variance, as required for applications such as autonomous driving. Additionally, we synthesize large quantities of simulated data, then demonstrate a significant boost in performance when training with synthesized data and validating on real-world data. We use select classes as a proof-of-concept, granting synthesized data a potential role in self-driving datasets of the future.

ACKNOWLEDGEMENT

This work was partially supported by the DARPA PERFECT program, Award HR0011-12-2-0016, together with ASPIRE Lab sponsor Intel, as well as lab affiliates HP, Huawei, Nvidia, and SK Hynix. This work has also been partially sponsored by individual gifts from BMW, Intel, and the Samsung Global Research Organization.

REFERENCES

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3354–3361.

[2] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, "On the segmentation of 3d lidar point clouds," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2798–2805.

[3] M. Himmelsbach, A. Mueller, T. Lüttel, and H.-J. Wünsche, "Lidar-based 3d object perception," in *Proceedings of 1st international workshop on cognition for technical systems*, vol. 1, 2008.

[4] D. Z. Wang, I. Posner, and P. Newman, "What could move? finding cars, pedestrians and bicyclists in 3d laser data," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4038–4044.

[5] D. Zermas, I. Izzat, and N. Papanikolopoulos, "Fast segmentation of 3d point clouds: A paradigm on lidar data for autonomous vehicle applications," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 5067–5073.

[6] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[7] F. Moosmann, O. Pink, and C. Stiller, "Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion," in *Intelligent Vehicles Symposium, 2009 IEEE*. IEEE, 2009, pp. 215–220.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.

[9] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *Advances in neural information processing systems*, 2011, pp. 109–117.

[10] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv preprint arXiv:1606.00915*, 2016.

[11] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr, "Conditional random fields as recurrent neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1529–1537.

[12] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *arXiv:1602.07360*, 2016.

[13] M.-O. Shin, G.-M. Oh, S.-W. Kim, and S.-W. Seo, "Real-time and accurate segmentation of 3-d point clouds based on gaussian process regression," *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[14] L. Caltagirone, S. Scheidegger, L. Svensson, and M. Wahde, "Fast lidar-based road detection using fully convolutional neural networks." in *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, 2017, pp. 1019–1024.

[15] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," *arXiv preprint arXiv:1611.07759*, 2016.

[16] J. Schlosser, C. K. Chow, and Z. Kira, "Fusing lidar and images for pedestrian detection using convolutional neural networks," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2198–2205.

[17] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *arXiv preprint arXiv:1608.07916*, 2016.

[18] D. Maturana and S. Scherer, "3d convolutional neural networks for landing zone detection from lidar," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3471–3478.

[19] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European Conference on Computer Vision (ECCV)*, ser. LNCS, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9906. Springer International Publishing, 2016, pp. 102–118.

[20] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, and R. Vasudevan, "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?" *CoRR*, vol. abs/1610.01983, 2016. [Online]. Available: http://arxiv.org/abs/1610.01983

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," *Google Technical Report*, 2015.