

Hierarchical Recurrent Attention Networks for Structured Online Maps

Namdar Homayounfar^{1,2} Wei-Chiu Ma^{1,3}, Shrinidhi Kowshika Lakshminikanth¹, Raquel Urtasun^{1,2}
¹Uber Advanced Technologies Group ²University of Toronto ³ MIT

namdar.homayounfar@mail.utoronto.ca, weichium@mit.edu, kowshika@cs.cmu.edu
urtasun@cs.toronto.edu

Abstract

In this paper, we tackle the problem of online road network extraction from **sparse 3D point clouds**. Our method is inspired by how an annotator builds a lane graph, by first identifying how many lanes there are and then drawing each one in turn. We develop a hierarchical recurrent network that attends to initial regions of a lane boundary and traces them out completely by **outputting a structured polyline**. We also propose a novel differentiable loss function that measures the deviation of the edges of the ground truth **polylines** and their predictions. This is more suitable than distances on vertices, as there exists many ways to draw equivalent polylines. We demonstrate the effectiveness of our method on a 90 km stretch of highway, and show that we can recover the right topology 92% of the time.

1. Introduction

A self driving car software stack is typically composed of three main components: perception, prediction and motion planning. *Perception* consists of estimating where everyone is in the scene in 3D given data from the sensors (e.g., LiDAR, cameras, radar, ultrasonic). *Prediction* is concerned with predicting the future action of the dynamic objects that have been identified by perception. Finally, the outputs of perception and prediction are used by *motion planning* in order to decide the trajectory of the ego-car.

Despite several decades of research, these three tasks remain an open problem. To facilitate these tasks, most self-driving car teams rely on high definition maps, commonly referred to as *HD maps*, which contain both geometric and semantic information about the static environment. For example, planning where to go is easier if we know the geometry of the road (i.e., lane composition). This information is also very useful to determine the future motion of other traffic participants. Furthermore, false positives in vehicle detection can be reduced if we know where the road is.

To create these maps, most self-driving car programs rely on offline processes where semantic components such

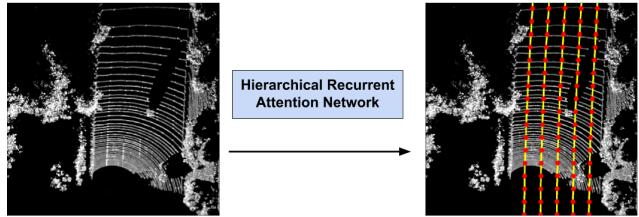


Figure 1. Our hierarchical recurrent model takes a sparse point cloud sweep of the road (**right**) and outputs (**left**) a structured representation of the road network where each lane boundary instance is retrieved.

as lanes are extracted with the help of a user in the loop. Furthermore, they typically require multiple passes over the same environment in order to build accurate geometric representations. This is very expensive and requires a dedicated fleet for mapping. It has been estimated that mapping the US only once will cost over 2 Billion dollars. This approach is not scalable globally and thus it is of fundamental importance to design online mapping algorithms that do not require a user in the loop or in the least minimize their involvement to correction tasks.

The most basic information that is required for driving is to be able to extract the location of the lanes in 3D space (mapping), and their relationship to the ego-car (localization to the map). In the context of maps, these lanes are structured objects and are typically represented as a set of polylines, one per lane boundary. We refer the reader to Fig. 1 for a representation of a lane graph.

Existing automatic approaches to lane extraction have two major drawbacks. First, they treat the problem as semantic segmentation [26, 27] or lane marking detection [20, 11, 14]. As a consequence they produce solutions that are not topologically correct, e.g., a lane might have holes due to occlusion. This is problematic as most motion planners can only handle lane graphs that are structured and represent the right topology. Second, these methods attempt to extract lane graphs from camera images [43]. Unfortunately, a good image based lane estimation is not equivalent

to an accurate 3D lane extraction. Due to perspective projection, pixels in image space have different physical widths in 3D. This largely limits their applications in real world.

With these challenges in mind, we present a novel approach to online mapping that extracts structured lane boundaries directly from a single LIDAR sweep. To be more specific, we propose a hierarchical recurrent neural network that is able to both count the number of lanes and draw them out. The model takes as input sparse LIDAR point clouds, which is the natural 3D space for detecting lane boundaries, and outputs a structured representation of the road network that is topologically correct and hence consumable by existing motion planners. As there exists many ways to draw equivalent polylines, we further develop a novel differentiable loss function that directly minimizes the distance between two polylines rather than penalizing the deviation of GT and predicted vertices using cross-entropy or regression loss [5]. The objective helps our network focus on learning the important lane graph structure, rather than the irrelevant vertices coordinates. Our model can be trained in an end-to-end manner without heuristics or post-processing steps such as curve fitting. The overall process mimics how humans annotate maps and thus is amenable to a guided user in the loop.

We demonstrate the effectiveness of our approach on highway sequences captured over a range of 90 km. Our approach determines the right topology 92% of the time. Furthermore, we recover the correct road network with an average of 92% precision and 91% recall at a maximum distance of 20 cm away from the lane boundaries .

2. Related Work

Road and Lane Detection Finding the drivable path in front of an autonomous vehicle and the lane boundaries is of outmost importance. [29, 21, 42] apply graphical models on manually generated annotations to estimate the free space and the road. Some other methods [36, 22, 31, 3, 16, 7, 39, 1, 18] use unsupervised or self-supervised methods based on color, texture or geometric priors to detect the road. The authors in [19, 38, 13, 2, 28] develop road detection algorithms either by automatically or manully generating labels from maps or using them as priors. The authors in [8] extend a small subset of the KITTI [9] dataset and provide a benchmark for detecting polygons that define the free space of the road and the ego-lane. Recently, the resurgence of deep learning methods [17, 35] has provided tremendous success in many different computer vision tasks. For lane detection, [20] train a neural network that detects land and road markings by leveraging vanishing points.

Semantic segmentation of aerial images/ road network extraction Aerial imagery can be used for road network extraction and segmentation. Although, aerial imagery can

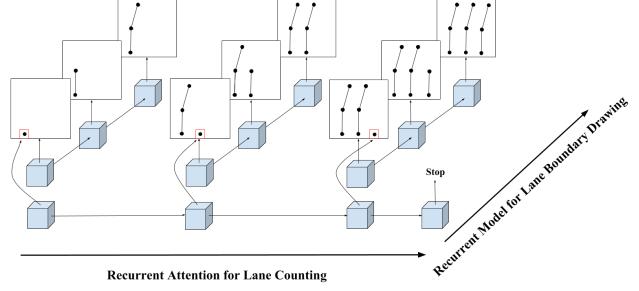


Figure 2. The overall structure of our model where a convolutional recurrent neural network sequentially attends to the initial regions of the lane boundaries while a another convolutional LSTM traces them out fully. This process iterates until the first RNN signals a stop.

cover a huge portion of the world, they operate on a lower resolution and thus cannot be used for fine grained map creation. [26, 27] enhance freely available maps using aerial imagery by fine grained semantic segmentation and inference in an MRF. In other work [40, 41, 30] extract the road network from aerial images using a CRF model. [25] use an end-to-end fully convolutional network to segment high resolution aerial images. [25] presents an end-to-end semantic segmentation deep learning approach of very high resolution aerial images. Recently, The Torontocity dataset [37] provides a benchmark for extracting road curbs and center-lines from bird's eye view maps.

Other Our work is inspired by [5] in which the authors develop a semi-automatic annotation tool of object instances by directly predicting the vertices of the polygon outlining the object's segmentation mask. They use the cross-entropy loss to learn the position of the vertices. We note that a loss function on the location of the vertices is not appropriate as there are many ways to draw the same polygon. As such, we design a novel loss function that penalizes directly in a differentiable manner the deviation of the edges of the predicted polylines from their ground truth. In [33, 34] the a recurrent network iteratively attends to object instances and segments them. We use a similar idea by attending to lane boundaries and drawing them out.

3. Hierarchical Recurrent Attention Model for Lane Graph Extraction

Our goal is to extract a structured representation of the road network, which we will henceforth refer to as a *lane graph*. A lane graph is defined as a set of polylines, each of which defines a lane boundary. Towards this goal, we exploit a LIDAR sweep in the form of a point cloud projected onto bird's eye view (BEV) as our input $x \in \mathbb{R}^{H \times W \times 1}$. The point cloud contains LIDAR intensity for each point,

a cue that allows us to exploit the reflective properties of paint. This provides us with a sparse representation of the 3D world. We refer the reader to Fig. 1 for an example of our point cloud input and the predicted lane graph.

Our approach is inspired by how humans create lane graphs when building maps. In particular, annotators are presented with a bird’s eye view of the world, and sequentially draw one lane boundary at a time. To do so, they typically start from the bottom corner of the left most lane and draw each lane by first choosing an initial vertex on the lane boundary and tracing the lane boundary by a sequence of further clicks.

When the lane boundary is fully specified in the form of a polyline, the annotator moves on to its closest neighbour and repeats the process until no more lane boundaries are visible.

We design a structured neural network that closely mimics this process as demonstrated in Fig. 2. Our hierarchical recurrent network sequentially produces a distribution over the initial regions of the lane boundaries, attends to them and then draws a polyline over the chosen lane boundary by outputting a sequence of vertices.

Our network iterates this process until it decides that no more lane boundaries are present and it is time to stop.

In the following, we explain in detail the main components of our model. In particular, an encoder network is shared by a recurrent attention module that attends to the initial regions of the lane boundaries (Section 3.2) and a decoder network that feeds into a conv-lstm that draws each lane boundary given the initial region (Section 3.3).

3.1. Encoder-Decoder Backbone

Our model is based upon the feature pyramid networks of [23, 24, 6], where a residual [10] encoder-decoder architecture with lateral additive connections is used to build features at different scales. The features of the encoder are shared by both the recurrent attention module 3.2 and the Polyline-RNN 3.3 and capture information about the location of the lane boundaries at different scales. The decoder is composed of multiple convolution and bilinear upsampling modules that build a feature map used by only the Polyline-RNN module. We use batch norm [12] and ReLU non-linearity throughout the network. The exact architecture is outlined in the supplementary material.

3.2. Recurrent Attention for Lane Counting

We design a network that is able to decide how many lane boundaries exist and attends to the region in the image where the lane boundary begins. We have deliberately simplified the output of the net to be a region rather than the exact coordinates of a vertex. This way, if run in interactive mode, an annotator is only required to provide a coarse starting region for a lane boundary to be drawn. The

task of predicting the actual vertex coordinates falls upon the Polyline-RNN which we shall describe in detail in section 3.3. These regions correspond to non-overlapping bins that are obtained by dividing the input x into K segments along each spatial dimension as demonstrated in Fig. 4.

For the network to predict the starting regions of the lane boundaries, we deploy a similar strategy as [32, 5] and concatenate the feature maps of the encoder network so that the net has clues at different granularities. We use convolution layers with large non-overlapping receptive fields to downsample the larger feature maps and use bilinear upsampling for the smaller feature maps to bring all of them to the same resolution. Next, this feature map is fed to two residual blocks in order to obtain a final feature map f of smaller resolution than the point cloud input x to the network. We reduce the resolution since we care only about the regions where a lane boundary starts rather than its exact starting coordinate.

Next, a vanilla convolutional RNN is iteratively applied to f with the task of attending to regions of x and outputting a starting region of the lane boundary.

In order to be able to stop, this RNN also outputs a binary variable denoting whether we have already counted all the lanes or not. In particular, at each timestep t , the conv-RNN outputs on one hand the probability h_t of halting while the other output is a softmax s_t of dimension $K \times K \times 1$ over the region of the starting vertex of the next lane boundary. At inference time, we replace the softmax with an argmax and threshold the probability of halting.

3.3. Drawing Lane Boundaries with Polyline-RNN

We use a convolutional LSTM to iteratively draw a polyline as a sequence of vertices. In particular, the recurrent attention mechanism of the previous section provides us with a region which contains the first vertex of the lane boundary. A section of dimension $H_c \times W_c$ around this region is cropped from the output feature map of the decoder of section 3.1 and fed into the conv-LSTM. The conv-LSTM then produces a softmax over the position of the next vertex p_1^t on the lane boundary. The vertex p_1^t is then used to crop out the next region and the process continues until the lane boundary is fully traced or we reach the boundary of the image. As such we obtain our polyline P_t at timestep t .

4. Learning

To facilitate learning, we derive a **multitask objective** that provides supervision for each component of our hierarchical recurrent attention model. Our output loss function computes the difference between **two polylines**, the ground truth and our prediction. Note that a loss function on the location of the vertices is not appropriate as there are many ways to draw the same polyline which will have very different location of vertices.

Sparse Point Cloud

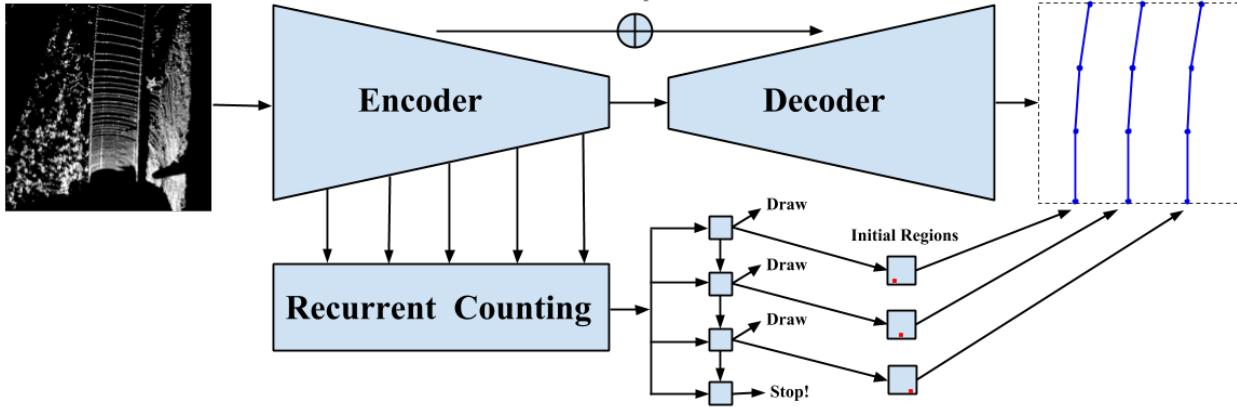


Figure 3. Our Hierarchical Recurrent Attention Network: An encoder network is shared by the recurrent attention module for counting and attending to the initial regions of the lane boundaries as well as a decoder that provides features for the Polyline-RNN module that draws the lane boundaries of the sparse point cloud.

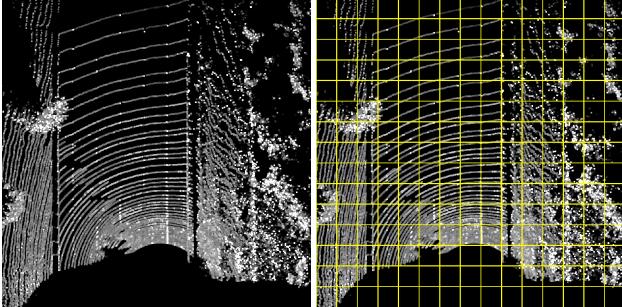


Figure 4. The input point cloud (**right**) is discretized into K bins along each dimension (**left**). The recurrent attention module for counting, as well as an annotator, need only to focus on an initial region of the lane boundary rather than an exact vertex.

Instead, we directly penalize the deviations of the two curves. We further provide supervision at the level of our attention mechanism over regions which contain a starting vertex of a polyline. We also define a loss function that teaches the network when to stop counting the polylines.

4.1. Polyline Loss

We encourage the edges of a prediction P to superimpose perfectly on those of a ground truth Q . In particular, we define:

$$\begin{aligned} L(P, Q) &= L_{P \rightarrow Q} + L_{Q \rightarrow P} \\ &= \sum_i \min_{q \in Q} \|p_i - q\|_2 + \sum_j \min_{p \in P} \|p - q_j\|_2 \quad (1) \end{aligned}$$

Note that although our predictions are vertices of P , in the above equation we sum over the coordinates of all the edge pixels of P and Q rather than solely their vertices.

We note that the two terms of the loss function are symmetric. Intuitively, the first term $L_{P \rightarrow Q}$ encourages the predicted polyline P to lie on the ground truth Q by summing and penalizing the deviation of all the edge pixels of P from those of Q . While necessary, this loss is not sufficient for P to cover Q completely since it ignores those superimposing polylines P that are shorter than Q . To overcome this, the second loss $L_{Q \rightarrow P}$ instead penalizes the deviations of the ground truth from the predicted polyline. In particular, if a segment of Q is not covered by P , all the edge pixels of that segment would incur a loss.

As noted, the loss function in Eq. (1) is defined w.r.t. to all the edge pixel coordinates on P whereas the Polyline-RNN network predicts only a set of vertices. As such, for every two consecutive vertices p_j and p_{j+1} on P , we obtain the coordinates of all the edge pixel points lying in-between by taking their convex combination. This makes the gradient flow from the loss functions to the network through every edge point.

In practice, both loss functions can be obtained by computing the pairwise distances, and then taking a min-pool and finally summing. We illustrate the two terms $L_{P \rightarrow Q}$ and $L_{Q \rightarrow P}$ in Fig. 5(a) and show their effect through a toy example in Fig. 5(b).

Comparison against Polygon-RNN [5] While our work is inspired by [5], there exists a critical difference — our loss functions are defined w.r.t. the edges rather than the vertices. As shown in Fig. 6(a), there exist many ways to draw equivalent polylines. It is thus more suitable to consider the distance between polylines than the deviation of the vertices. Fig. 6(b) shows the caveats of [5]. The prediction can be superimposed perfectly with the ground truth, yet Polygon-RNN still penalizes the model. Since polygons are simply special cases of polylines, our polyline loss can

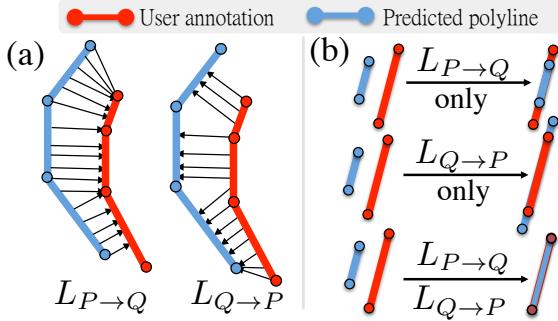


Figure 5. (a) Illustration of the two objectives $L_{P \rightarrow Q}$ and $L_{Q \rightarrow P}$. (b) The effect of the objectives on a toy example. $L_{P \rightarrow Q}$ and $L_{Q \rightarrow P}$ both have blind spots. By combining both, the model can learn to superimpose perfectly.

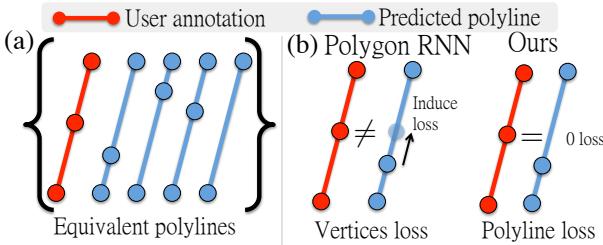


Figure 6. (a) A subset of polylines that are equivalent. (b) Comparison to Polygon-RNN [5]. Our loss function encourages the model to learn to draw equivalent polylines, rather than output the exact vertex coordinate.

be directly plugged into [5]. It can also be applied to other tasks that require learning boundaries, such as boundary detection [44]. We leave this for future study.

4.2. Attention Loss

To train the recurrent attention network for polyline counting, we apply a cross entropy loss on the region softmax output s_t and a binary cross entropy loss on the halting probability h_t . The ground truth for the regions are the bins in x where the initial vertex of the lane boundaries falls upon. We present the ground truth bins to the loss function in order from the left of the image to the right similar to how an annotator picks the initial regions. For the binary cross entropy, the ground truth is equal to one for each lane boundary and zero when it is time to stop counting.

4.3. Training Procedure

We train our model in two stages. At first, the encoder-decoder model with only the polyline-RNN is trained with the ground truth initial regions. We clip the gradients of the conv-lstm to the range of $[-10, 10]$ to remedy the exploding/vanishing gradient problem. For training the conv-lstm, we crop the next region using the predicted previous vertex. The conv-lstm iterates until the next region falls outside the

image boundaries or a maximum of image height divided by crop height plus 3. We let the size of the crop to be 60×60 pixels. We train using SGD [4] with initial learning rate of 0.001, weight decay 0.0005 and momentum 0.9 for one epoch with a minibatch size of 1.

Next, we freeze the weights of the encoder and train only the parameters of the recurrent attention module for counting for one epoch. We train the conv-rnn that predicts the number of lane boundaries using the Adam optimizer [15] with an initial learning rate of 0.0005 and weight decay of 0.0005 with a minibatch size of 20. The training criteria were determined based on the results on the validation set. The model is trained on one Titan XP GPU for close to 24 hours with the majority of the training time devoted to the Conv-LSTM module.

5. Experimental Evaluation

Dataset: We curated a dataset on highways and mapped a stretch of 90 km and geofenced to rotating consecutive stretches of 10 km for each of training, validation and the test set. Our autonomous vehicle uses a mounted Lidar that captures point clouds at 10 fps. We sampled uniformly 50,000 frames for the training set and 10,000 frames for each of the validation and the test sets from the corresponding regions. Our data contains both night and day scenes.

For each frame, we project the 3D point cloud and the ground truth lane graph to BEV such that the autonomous vehicle is positioned on the bottom center of the image looking up. We rasterize the lidar point cloud such that each pixel corresponds to 5 cm. We use images of size 960×960 pixels corresponding to 48 m in front and 24 meters on each side of the autonomous vehicle.

Baselines: Since there are no existing baselines in the literature for this task, we developed a strong baseline to evaluate and motivate our method. In particular, we take the encoder and decoder modules of our architecture, remove the lane counting and Polyline-Rnn modules, and output a sigmoid function corresponding to a dense 20 pixel wide region around each lane boundary. In other words, we aim to detect a dense representation of the lane boundaries. We used the exact same architecture of our hierarchical network and trained the network using binary cross-entropy for three epochs. We use the Adam optimizer [15] with an initial learning rate of 0.001 and weight decay of 0.0001 determined from the validation set. We have visualized some lane detection results in Fig. 5.

Note that while the output of our hierarchical model is a structured representation where each lane boundary instance is predicted by the network, the baseline only outputs a dense representation of the lane boundaries and further post processing steps are required to obtain each individual instance. Thus, we proceed as follows: First, we threshold the sigmoid output of the baseline for different values

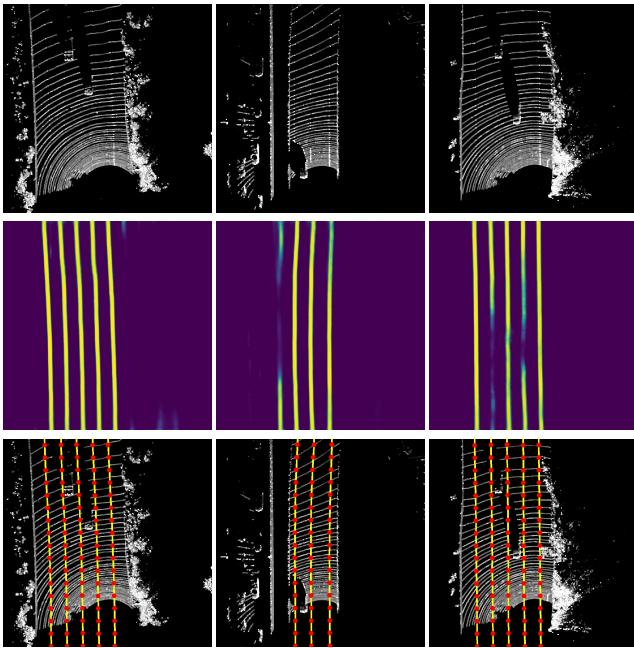


Figure 7. **Top Row:** Point cloud sweep of the road. **Middle Row:** The sigmoid output of the baseline corresponding to a 20 pixels wide region around the lane boundaries. **Bottom Row:** The lane boundary instances outputs of our network.

of 0.3, 0.5, 0.7 and 0.9 to remove spurious detections. Each threshold is considered as a baseline and we refer to them as CE at 0.3 to Ce at 0. Next, separately for each baseline, we skeletonize the result and finally obtain each individual lane boundary instance using connected components.

Topology: Our first metric focuses on evaluating whether the correct number of lanes is estimated. In Fig. 8, we demonstrate the cumulative distribution of the absolute value of the difference between the ground truth number of lanes and the predicted number of lanes by our network as well as the baselines. Our model estimates the correct number of lanes 92% of the time while deviating by one or less lane boundaries almost 99% of the time. We note that our method outperforms the strongest baseline (in terms of topology) that retrieves the correct number of lanes only 46% of the time while being one away or less 61% of the time. We highlight that our model is specifically designed to output a structured representation of the lane boundaries by learning to count and draw polylines. On the other hand, the ad-hoc post processing steps applied to the baseline introduce holes in places where the lane detections do not fire and as such topology deteriorates. Hence the reason some of the baselines predict more than 10 lane boundaries. Moreover, our structured representation enables the annotator to easily correct a mistake by either specifying the initial

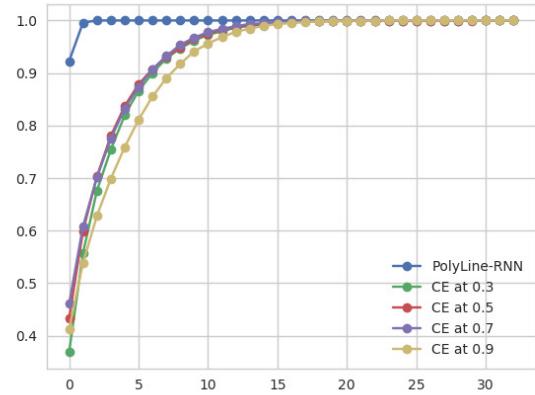


Figure 8. The cumulative distribution of the absolute value of the difference between the ground truth number of lanes and the predicted number of lanes by our network as well as the baselines.

region of a lane boundary (if it's missed) or by deleting it altogether. This benefit is not extendible to the baselines. We will present an experiment later on that corroborates this aspect of our model.

Precision/Recall: We use the precision and recall metric of [37]. We define precision as the number of predicted polyline points within a threshold distance of a ground truth lane boundary divided by the total number of points on the predicted polylines. Recall is defined in a symmetric fashion as with respect to a thresholded distance of the predicted polylines. We set the thresholds at the intervals of 5, 10, 15, and 20 cm corresponding to a maximum deviation of 4 pixels from a polyline. Note that bigger thresholds are not interesting as the autonomous vehicle should be able to localize itself with high accuracy.

From Table. 5 one can note that the best performing baseline in terms of precision, i.e. CE at 0.9, performs the worst in comparison to CE at 0.5 that achieves the best recall. This alludes to the fact that when only a small number of points are retained for a higher threshold, precision goes up while the opposite holds for recall.

Our model beats the baselines in both precision and recall. Although the values are within 1-2 percentage points of each other. We remind the reader that our method beats the best performing baseline with a high margin when it comes to retrieving the correct topology.

Annotator In the Loop: The output of our model is a structured representation of the lane boundary instances and as such is easily amenable to bring an annotator in the loop. To demonstrate this, we perform an experiment on the set of 687 examples where we predict the wrong topology. In particular, the annotator either clicks on the starting region

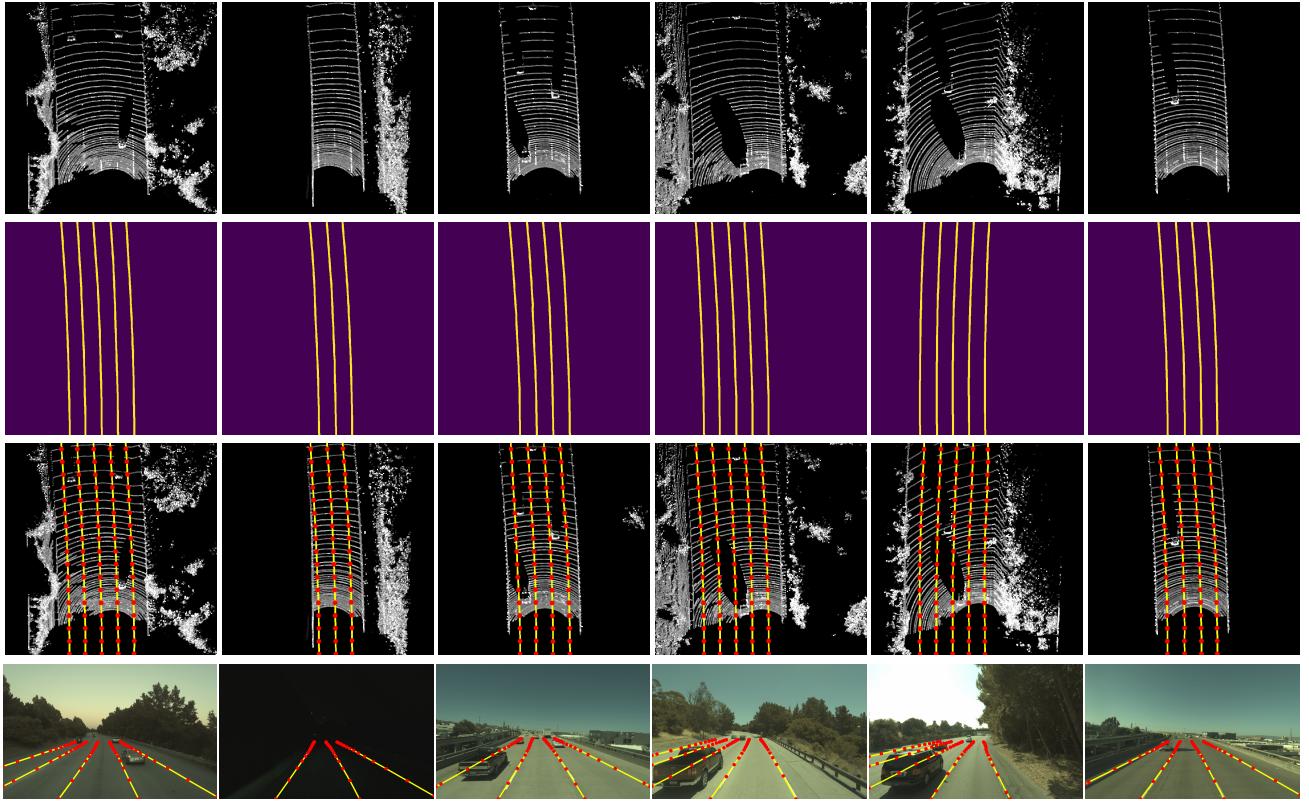


Figure 9. Qualitative Examples. **Top Row:** Point cloud sweep of the road. **Second Row:** The ground truth lane graph. **Third Row:** The lane boundary instances outputs of our network. **Bottom Row:** The predicted lane boundaries projected onto the image.

	Precision at (cm)				Recall at (cm)			
	5	10	15	20	5	10	15	20
CE at 0.3	0.200	0.550	0.773	0.879	0.203	0.560	0.788	0.896
CE at 0.5	0.211	0.575	0.796	0.896	0.209	0.574	0.799	0.894
CE at 0.7	0.212	0.577	0.801	0.903	0.207	0.566	0.787	0.887
CE at 0.9	0.212	0.580	0.810	0.917	0.198	0.546	0.762	0.861
Ours	0.226	0.609	0.827	0.92	0.223	0.6	0.816	0.908

Table 1. Comparison of our proposed model vs. the cross entropy baseline in terms of precision and recall for distances of 5 to 20 cm from the lane boundaries.

of a lane boundary or removes one by just a click when it is either missed or hallucinated respectively. In Table. 5, we observe that among these failure cases, precision suffers after correction by a maximum of 2 % for different distances to the lane boundary while recall increases by at least 1% and maximum 10% at a 20 cm distance to the lane boundary. This is expected since there is usually low evidence for a lane boundary in failure cases and adding just the starting region would improve only the recall but have an adverse effect on the overall precision. Importantly, the annotator takes on average 1.07 clicks to fix these issues.

We highlight that the annotator only needs to specify a coarse starting region of the lane boundary, e.g. Fig. 4, rather than an exact initial vertex. This facilitates the task;

One can see from the Lidar images of Fig 5 that initial vertex of the lane boundaries are not visible while it is easy to guess the region where they begin.

Qualitative Examples: In Fig. 5, we demonstrate the abilities of our model to make high precision and recall predictions with perfect topology. Our model is able to deal with occlusions due to other vehicles and most importantly has learned to extrapolate the lane boundaries to parts of the image where no evidence exists. Moreover, our model performs well at night where camera based models might have difficulty. We also depict the projection of the predicted lane graph onto the frontal camera view of the road for visualization purposes only.

	Precision at (cm)				Recall at (cm)			
	5	10	15	20	5	10	15	20
Ours Before Correction	0.195	0.534	0.748	0.851	0.169	0.461	0.647	0.736
Ours After Correction	0.188	0.515	0.726	0.833	0.189	0.519	0.731	0.841

Table 2. Evaluating the annotator in the loop by comparing precision and recall for before and after images with the wrong topology are corrected. On average an annotator takes 1.07 clicks to fix these mistakes.

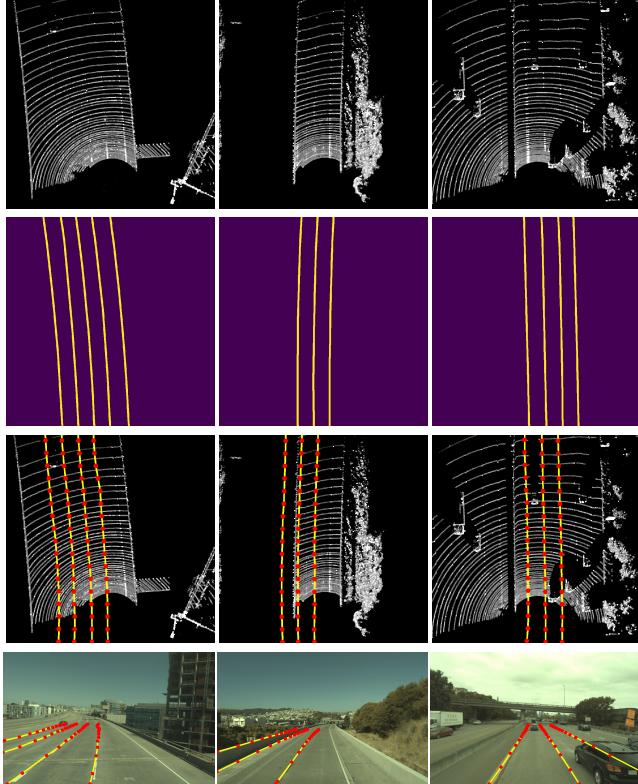


Figure 10. Failure Cases. **Top Row:** Point cloud sweep of the road. **Second Row:** The ground truth lane graph. **Third Row:** The lane boundary instances outputs of our network. **Bottom Row:** The predicted lane boundaries projected onto the image.

Failure Modes: In Fig. 10 we visualize a few failure cases. In columns 1 and 3, we observe that the topology is wrong due to an extra predicted lane boundary. In the second column, the road separator is mistaken for a lane boundary. However, note that an annotator can pass through these images and fix the issues with only one click.

Inference Time: Our model takes on average 175 ms for one forward pass timed on a Titan XP. While the encoder-decoder module takes only 15 ms, the majority of the inference time is spent on the convolutional LSTM cells.

Learned Features: In Fig. 11 we visualize three channels of the last feature map of the decoder network before feeding it is fed into the conv-lstm for lane boundary drawing. The input to the network is a sparse point cloud and the

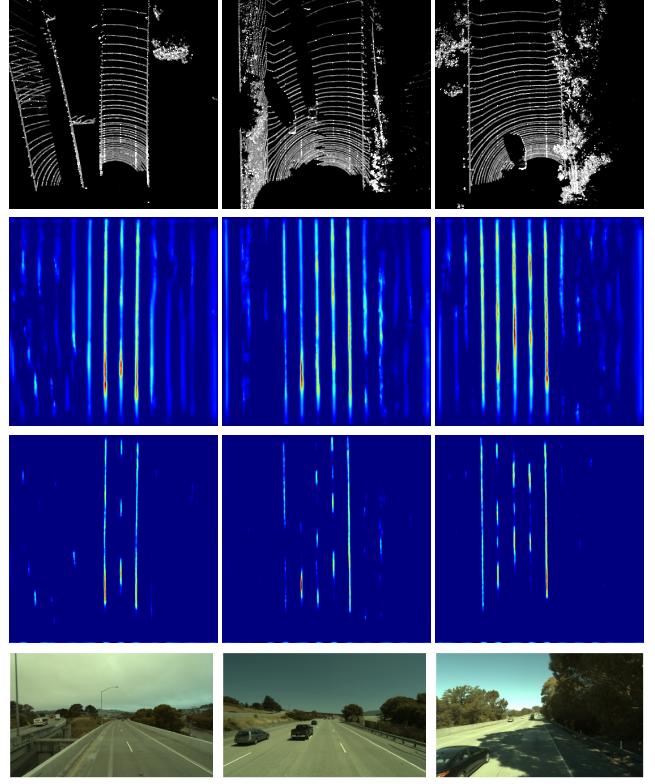


Figure 11. Learned Features. **Top Row:** Point cloud sweep of the road. **2-4th Rows:** Three feature map channels of the decoder network. **Bottom Row:** The camera view of the vehicle.

output is a structured lane graph. We observe that in order to learn this mapping, the network has learned to pick out lane markings and extrapolate them to fine the boundaries.

6. Conclusion

In this paper, we proposed a hierarchical recurrent attention network that mimics how an annotator creates a map of the road network. In particular, given a sparse lidar sweep of the road, a recurrent attention module attends to the initial regions of the lane boundaries while a convolutional LSTM draws them out completely. We developed a novel loss function that penalizes the deviation of the edges of the ground truth and predicted polylines rather than their vertices. We demonstrated the effectiveness of our method by extensive experiments on a 90 km of highway.

References

- [1] J. M. Alvarez, T. Gevers, Y. LeCun, and A. M. Lopez. Road scene segmentation from a single image. In *ECCV*, 2012. [2](#)
- [2] J. M. Álvarez, A. M. Lopez, T. Gevers, and F. Llumbreras. Combining priors, appearance, and context for road detection. *IEEE Transactions on Intelligent Transportation Systems*, 2014. [2](#)
- [3] J. M. Á. Alvarez and A. M. Lopez. Road detection based on illuminant invariance. *IEEE Transactions on Intelligent Transportation Systems*, 2011. [2](#)
- [4] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. 2010. [5](#)
- [5] L. Castrejon, K. Kundu, R. Urtasun, and S. Fidler. Annotating object instances with a polygon-rnn. In *CVPR*, 2017. [2](#), [3](#), [4](#), [5](#)
- [6] A. Chaurasia and E. Culurciello. LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation. *arXiv*, 2017. [3](#)
- [7] H.-Y. Cheng, B.-S. Jeng, P.-T. Tseng, and K.-C. Fan. Lane detection with moving vehicles in the traffic scenes. *IEEE Transactions on intelligent transportation systems*, 2006. [2](#)
- [8] J. Fritsch, T. Kühnl, and A. Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, 2013. [2](#)
- [9] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. [2](#)
- [10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016. [3](#)
- [11] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv*, 2015. [1](#)
- [12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015. [3](#)
- [13] K. Irie and M. Tomono. Road recognition from a single image using prior information. In *IROS*, 2013. [2](#)
- [14] J. Kim and M. Lee. Robust lane detection based on convolutional neural network and random sample consensus. In *International Conference on Neural Information Processing*, 2014. [1](#)
- [15] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. [5](#)
- [16] H. Kong, J.-Y. Audibert, and J. Ponce. General road detection from a single image. *TIP*, 2010. [2](#)
- [17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. [2](#)
- [18] T. Kühnl, F. Kummert, and J. Fritsch. Spatial ray features for real-time ego-lane extraction. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, 2012. [2](#)
- [19] A. Laddha, M. K. Kocamaz, L. E. Navarro-Serment, and M. Hebert. Map-supervised road detection. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, 2016. [2](#)
- [20] S. Lee, J. Kim, J. S. Yoon, S. Shin, O. Bailo, N. Kim, T.-H. Lee, H. S. Hong, S.-H. Han, and I. S. Kweon. Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. *arXiv*, 2017. [1](#), [2](#)
- [21] D. Levi, N. Garnett, E. Fetaya, and I. Herzlyia. Stixelnet: A deep convolutional network for obstacle detection and road segmentation. In *BMVC*, 2015. [2](#)
- [22] D. Lieb, A. Lookingbill, and S. Thrun. Adaptive road following using self-supervised learning and reverse optical flow. In *RSS*, 2005. [2](#)
- [23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. *arXiv*, 2016. [3](#)
- [24] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv*, 2017. [3](#)
- [25] D. Marmanis, J. D. Wegner, S. Galliani, K. Schindler, M. Datcu, and U. Stilla. Semantic segmentation of aerial images with an ensemble of cnss. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2016, 2016. [2](#)
- [26] G. Matyus, S. Wang, S. Fidler, and R. Urtasun. Enhancing road maps by parsing aerial images around the world. In *ICCV*, 2015. [1](#), [2](#)
- [27] G. Matyus, S. Wang, S. Fidler, and R. Urtasun. Hd maps: Fine-grained road segmentation by parsing ground and aerial images. In *CVPR*, 2016. [1](#), [2](#)
- [28] V. Mnih and G. E. Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 2012. [2](#)
- [29] R. Mohan. Deep deconvolutional networks for scene parsing. *arXiv*, 2014. [2](#)
- [30] J. A. Montoya-Zegarra, J. D. Wegner, L. Ladicky, and K. Schindler. Mind the gap: modeling local and global context in (road) networks. In *German Conference on Pattern Recognition*, 2014. [2](#)
- [31] L. M. Paz, P. Piniés, and P. Newman. A variational approach to online road and path segmentation with monocular vision. In *ICRA*, 2015. [2](#)
- [32] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to refine object segments. In *ECCV*, 2016. [3](#)
- [33] M. Ren and R. S. Zemel. End-to-end instance segmentation with recurrent attention. In *CVPR*, 2017. [2](#)
- [34] B. Romera-Paredes and P. H. S. Torr. Recurrent instance segmentation. In *ECCV*, 2016. [2](#)
- [35] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 2015. [2](#)
- [36] C. Tan, T. Hong, T. Chang, and M. Shneier. Color model-based real-time learning for road following. In *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*, 2006. [2](#)
- [37] S. Wang, M. Bai, G. Matyus, H. Chu, W. Luo, B. Yang, J. Liang, J. Cheverie, S. Fidler, and R. Urtasun. TorontoCity: Seeing the world with a million eyes. In *ICCV*, 2017. [2](#), [6](#)

- [38] S. Wang, S. Fidler, and R. Urtasun. Holistic 3d scene understanding from a single geo-tagged image. In *CVPR*, 2015. 2
- [39] A. Wedel, H. Badino, C. Rabe, H. Loose, U. Franke, and D. Cremers. B-spline modeling of road surfaces with an application to free-space estimation. *IEEE Transactions on Intelligent Transportation Systems*, 2009. 2
- [40] J. D. Wegner, J. A. Montoya-Zegarra, and K. Schindler. A higher-order crf model for road network extraction. In *CVPR*, 2013. 2
- [41] J. D. Wegner, J. A. Montoya-Zegarra, and K. Schindler. Road networks as collections of minimum cost paths. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2015. 2
- [42] J. Yao, S. Ramalingam, Y. Taguchi, Y. Miki, and R. Urtasun. Estimating drivable collision-free space from monocular video. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, 2015. 2
- [43] J. H. Yoo, S.-W. Lee, S.-K. Park, and D. H. Kim. A robust lane detection method based on vanishing point estimation using the relevance of line segments. *ITS*, 2017. 1
- [44] Z. Yu, C. Feng, M.-Y. Liu, and S. Ramalingam. Casenet: Deep category-aware semantic edge detection. 5