Learning Hierarchical Semantic Segmentations of LIDAR Data

David Dohan, Brian Matejek, and Thomas Funkhouser
Princeton University, Princeton, NJ USA
{ddohan, bmatejek, funk}@cs.princeton.edu

Abstract

This paper investigates a method for semantic segmentation of small objects in terrestrial LIDAR scans in urban environments. The core research contribution is a hierarchical segmentation algorithm where potential merges between segments are prioritized by a learned affinity function and constrained to occur only if they achieve a significantly high object classification probability. This approach provides a way to integrate a learned shape-prior (the object classifier) into a search for the best semantic segmentation in a fast and practical algorithm. Experiments with LIDAR scans collected by Google Street View cars throughout ∼100 city blocks of New York City show that the algorithm provides better segmentations and classifications than simple alternatives for cars, vans, traffic lights, and street lights.

1. Introduction

Applications in mapping, virtual reality, and city management have motivated a remarkable explosion in worldwide effort to acquire 3D data in urban environments. Now, several companies (e.g., Google, Nokia, Apple, etc.) are driving cars mounted with cameras and LIDAR scanners systematically up and down streets of urban areas throughout the world to capture 3D point cloud data. For example, Google Street View cars have three LIDAR scanners in addition to their cameras, two of which point directly left and right and capture vertical stripes of 3D point samples at 1 degree increments over a 180 degree range approximately 75 times per second. These scanners yield a set of 3D points on both sides of the street at approximately 20 centimeter resolution on nearby building facades, as shown in Figure 1. Although this data is directly useful for visualization (e.g., in Google Street View), it lacks the semantic information required for most 3D applications, like urban planning, traffic simulation, safety analysis, etc.

A semantic segmentation of a point cloud, which associates each point with a semantic class label (such as car, tree, etc.) and a segment identifier is an ideal starting place for many of these applications. However, automatic segmentation and classification of urban point cloud data is complicated by a number of data characteristics. First, data acquired from terrestrial vehicle-mounted scanners is limited – there is limited training data available online. Second, the point clouds are also often sparse, with only a few points representing small objects, and they contain data only from the street side of every object. Missing data is also common on reflective objects, such as windows or the sides of cars. As a result, previous methods for analyzing LIDAR scans of cities have had difficulties with semantic segmentation of street-side objects.

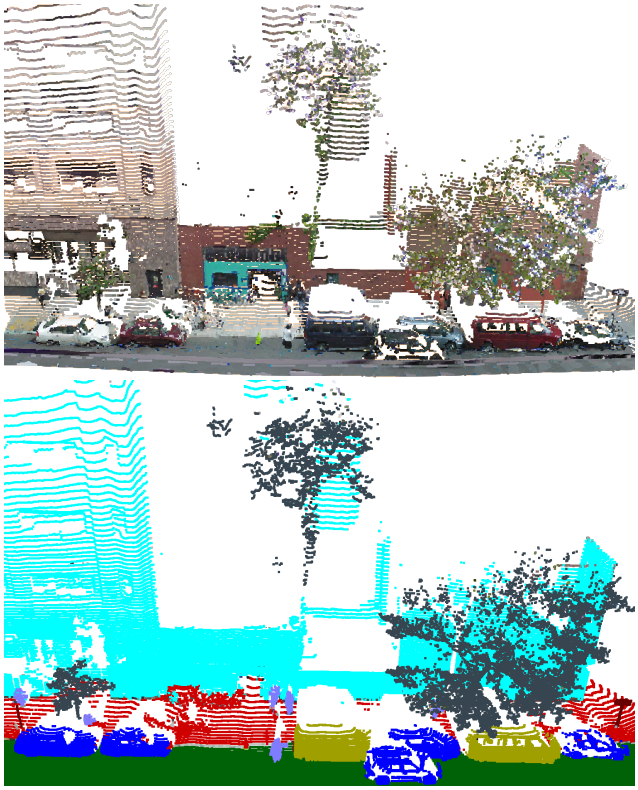

Figure 1. LIDAR point cloud captured by a Google Street View car in New York City (top image) and an example ground truth semantic labeling (bottom image).

We present a supervised algorithm that performs se-

mantic segmentations of terrestrial LIDAR point clouds by searching for the decompositions of points into segments that provide the highest category classification performance. We start from an oversegmentation of points into segments (like superpixels) and perform a hierarchical clustering to increase the confidence of category-level classifications. At the core of our method are learned classifiers that predict whether a pair of nearby segments are part of the same semantic object and whether merging the segments produces a segment typical of a given object class. We learn those classifiers from examples in a supervised training set and leverage them when ordering and selecting segment merges in a hierarchical clustering algorithm.

The main novel contribution is to integrate an object classifier into a hierarchical segmentation algorithm. This guides the algorithm to produce segments with properties (shape, context, etc.) consistent with example objects in a training set and is thus more likely to produce the correct semantic segmentation than alternative approaches that produce candidate segments without a class-specific shape prior. Experiments on large-scale Google Street View scans in New York City suggest this algorithm outperforms comparable alternatives.

2. Related Work

There is an extensive literature on segmenting and classifying objects in image data (e.g., [9, 26, 29, 33]). The following section highlights common methods, focusing on ones previously applied to recognizing small objects in LIDAR scans of cities [1, 8, 19, 32, 34, 35].

The simplest approach is to predict the semantic label for each point based on features of its local neighborhood [5, 7, 14, 16, 23]. Markov random fields (MRF) or conditional random fields (CRF) can be used to combine predictions based on pairwise affinities [2, 31]. Or, implicit shape models can be used to vote for object detections based on recognition of local features [15, 24, 30]. These methods do not explicitly consider the overall shapes of the resulting object predictions and are not as accurate as methods that classify objects based on complete segmentations.

Another simple approach is to perform segmentation and classification of a point cloud separately in a sequence of operations [6, 22]. Golovinskiy et al. [8] developed an end-to-end pipeline for recognizing small objects in urban point clouds, first removing the ground plane, then clustering points to identify possible object centers, using these as seeds for object segmentation, and finally classifying the segments based on segmented shapes. This approach can leverage a training set of correctly segmented objects to learn shape priors, and thus is effective for object classes that can be segmented robustly (e.g., stand-alone poles) [35]. However, it provides no feedback from object classification to segmentation, and therefore performs poorly for data where segmentations are ambiguous.

More recently, several researchers have considered using CAD models and/or templates as shape priors for object detection [11]. For example, [18, 17] use objects from Google's 3D Warehouse to train an object detection system for 3D point clouds collected by robots navigating through urban and indoor environments. [28] uses 3D CAD models as templates for a sliding window search. Shen et al. [25] use a database of segmented models and assembly-based modeling algorithms to reconstruct novel 3D models. Nan et al. [20] integrate a region growing segmentation algorithm with object recognition for cluttered indoor scene understanding. Most work with 3D CAD models has been demonstrated on only a few small scenes for objects with small shape variations (e.g., chairs). They would be difficult to apply to trees, traffic lights, or other objects classes with large intra-class shape variation.

Even more recently, there has been a strong trend towards deep learning for object detection and semantic segmentation. For example, Gupta et al. [10] propose methods to learn convolutional neural networks that score object proposals generated for region proposals and then use a random forest to perform foreground segmentation for each detection. In work more related to ours, Socher trains a recursive neural network on labeled images to score potential hierarchical merges in an image to produce a semantic scene segmentation [27]. These methods are very effective for object detection in images, but need massive amounts of labeled training examples and co-aligned RGB and depth image data, neither of which is available for our data sets (Google Street View).

Most closely related to our work are previous methods for learning affinity functions for hierarchical clustering [21]. For example, Jain et al. [13] train a convolutional neural network to directly minimize the RAND error, a measure of segmentation quality, on segmentations of neuron images. The CNN is used to cluster superpixels to produce a segmentation of new images. Our work extends this approach to work for LIDAR data and to include not only learned classifiers for evaluating potential merges between clusters, but also to evaluate the resulting properties of the merged clusters, in our case to validate whether the resulting segment is likely to be an object of a target semantic class.

3. System Overview

In this paper, we address the problems of segmenting and classifying 3D points acquired from terrestrial LIDAR data in urban environments.

The inputs to our system are: 1) a target semantic object class, 2) a test set captured with LIDAR scanners mounted on R5 Google Street View cars, and 3) a training set of such LIDAR data augmented with semantic segmentations. The

output of our system is a semantic segmentation of the test set – i.e., a semantic class label and instance label for each LIDAR point.

Our system preprocesses each LIDAR point cloud by: 1) detecting and removing major structural elements of urban environments (e.g., roads, curbs, sidewalks, building facades, etc.); 2) computing a feature vector of properties for each remaining LIDAR point (e.g., distance to the ground, planarity of its neighborhood, etc.); and 3) partitioning the LIDAR points into small segments akin to superpixels (e.g., clusters of points expected to be part of the same semantic object). The result is a set of small segments S, each containing a cluster of LIDAR points augmented with feature vectors (see details in Section 4.1).

During the training phase, our system uses the preprocessed training set to learn two functions: one that estimates the probability $P(C|A)$ that a segment A resides within an instance of the target object class C, and a second that estimates the probability $M(A, B)$ that two nearby segments A and B are part of the same semantic object.

Finally, during the testing phase, our system uses the learned functions to produce a semantic segmentation of the preprocessed test set $\hat{S}$. Ideally, our system would find the semantic segmentation S^* (labeled segments) that maximizes the a posteriori joint probability of the two learned functions. However, finding that solution is NP-Hard (by reduction to set partition). Therefore, we use a hierarchical clustering strategy to iteratively merge segments in the order dictated by $M(.,.)$ as long as the classification probability $P(C|...)$ of the merged segment is significantly high.

The main advantage of this approach is that it balances high-quality semantic segmentations with efficient execution. It achieves high quality semantic segmentations by utilizing a shape prior (a segment classifier $P(C|A)$) to guide the search over potential segmentations, producing segments that not only have high objectness but also match the geometric properties of the target object class (note that this use of $P(C|A)$ for merged sets of segments is a significant difference from standard MRF methods). We achieve efficient execution by incorporating the shape prior into the inner-loop of a greedy hierarchical segmentation algorithm, which runs in time roughly linear in the number of segments output by the preprocessing steps. These two features make it practical to run on large-scale data sets spanning entire cities, as required by the Google Street View dataset.

4. Algorithmic Details

This section provides details for the main three steps of the hierarchical semantic segmentation algorithm.

4.1. LIDAR Preprocessing

For every given LIDAR scan (during both the training and test phases), we first apply a sequence of preprocessing steps to extract structures and features from the raw point cloud. Only one of the preprocessing operations (curb detection) is particularly novel—the others are fairly standard. They are described here as context for the following two subsections, which describe the main algorithmic contributions of the paper.

The input data is a set of scans of street sides as seen from either the left or right side of a Google Street View car (Figure 1). Every scan is composed of a set of vertical scanlines, each composed of 180 depth samples spaced along 1 degree increments across 180 degrees ranging from straight down (0 degrees) to straight up (180 degrees), with parallel to the road appearing somewhere near the middle (90 degrees). Estimates for the scanner position and orientation are provided by Google, and so every depth sample can be represented by a 3D point in space.

The preprocessing for each 3D point begins by estimating its surface normal. It then estimates an RGB color by reprojection into the RGB image collected by the same Street View car with the closest timestamp. Finally, it computes a set of geometric features for each point, including its depth from the scanner, the cosine of the angle between its normal and the scanning direction, and differences between the eigenvalues of the covariance matrix of points within its one-meter neighborhood, $(\lambda 2 - \lambda 1)$ and $(\lambda 3 - \lambda 2)$, which estimate the "planarity" and "linearity" of the scanned surface surrounding the point. These color and geometric properties form a feature vector (with two more features described in the next paragraph) used later by a classifier to predict the semantic class of each point.

In addition to these basic point processing steps, we also execute an algorithm to partition points into several coarse semantic classes: road, sidewalk, building facade, small object, and other. This algorithm is somewhat novel in the way it leverages the structure of the LIDAR point acquisition process to discover boundaries between roads, sidewalks, and building facades. Noting that every scan is a sequence of vertical scanlines (a pushbroom depth image), and every vertical scanline contains some samples on a road, then possibly some on a sidewalk, and then possibly some on a building facade when considered from the bottom (looking straight down) to the top (looking straight up). Therefore, detecting these structures simply requires detecting boundaries between them (e.g., curbs) and then partitioning each vertical scanline according to the detections. Unfortunately, the LIDAR data has noise, occlusion, and poor sampling density, so detections of curbs, for example, based on local shape and color features within each scanline individually are not very robust. To address this issue, we formulate the problem as a graph cut: given a function representing the cost of labeling a point as a boundary (e.g., curb), (which was a simple template in our case but could be learned from examples), we aim for a solution that minimizes the sum

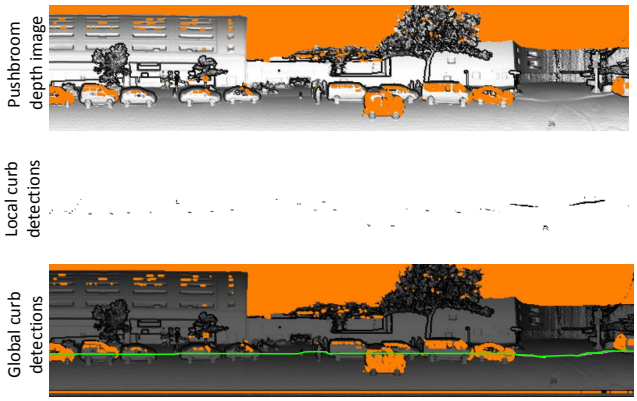

Figure 2. Curb detection. The top image shows a pushbroom image (sequence of vertical scanlines) showing 3D points acquired along one street with Google Street View. The next shows the output of a curb detector based on local image features (darkness is proportional to strength of prediction). The bottom image shows the output of the dynamic programming algorithm: a globally optimal curb detection (green curve).

of costs of a cut separating the bottom scanlines from the top. This formulation simultaneously tries to minimize the length of the boundary curve while aligning it with likely boundary points. An optimal solution can also be found with a dynamic program similar to ones proposed for seam carving [3] (Figure 2). We run this algorithm once to find road boundaries (curbs) and again to find building boundaries.

We label large planar horizontal regions directly below the curbs as roads and directly above the curbs as sidewalks, and we label large vertical planar regions within the building boundaries as facades. Then we eliminate the points associated with these large structures from further processing. Finally, we add two more properties to the feature vector for each remaining point representing the vertical distance to the closest road and the signed horizontal distance to the closest curb. These features are helpful because they provide semantic contextual cues.

The remaining points are candidates for small object segmentation and classification. They are partitioned into an initial segmentation using a clustering algorithm similar to the hierarchical one in Golovinskiy et al [8], but tuned so that segments are likely to be subsets of semantic objects, rather than vice-versa. The result is an (over)segmentation of the points akin to superpixels. For each segment, we compute a feature vector containing statistics of the property distributions for its member points (median, average, variance, and maximum difference), geometric properties of its 3D bounding box (volume, length of its vertical extent, area of its horizontal cross section, and ratio of the longer horizontal edge to the shorter one), geometric properties of its point cluster (z component of the best fitting plane normal), and contextual properties of the segment (length and depth extent relative to the curb). In all, there are 96 properties comprising the feature vector for each segment.

4.2. Learning Hierarchical Segmentations

During a training phase, our system takes as input a set of preprocessed and semantically segmented example LIDAR scans and outputs a Segment and a Merge Classifier.

Segment Classifier: the first classifier, $P(C|A)$, estimates the probability that a given segment A is part of an object of the target class C.

Learning this classifier would be straight-forward if we were only going to use it to classify segments of the initial segmentation: each initial segment is represented by a precomputed feature vector and has a ground truth label and thus provides exactly the type of data required to learn a classifier. However, we aim to build a classifier that works not only for the initial segments, but also for merged combinations of them that might be considered during hierarchical clustering. That is, if there are three initial segments $\{$A,B,C$\}$, then we need a classifier that can predict whether any combination of them $\{$A,B,C,AB,AC,BC,ABC$\}$ is part of the same object instance. This is difficult because the number of possible merge combinations is exponential in the number of initial segments, and few of those produce positive examples. So, we must be clever about how to create the training data.

We begin by extracting the ground truth label and segment id for each segment in the initial preprocessed segmentation: we assign a segment to a particular object in the ground truth data if the majority of its points overlap with the objects truth segmentation. Then, we generate training examples by considering merges involving at least one segment associated with a ground truth object (we don't pollute the training set with combinations of background segments). For each such candidate merge, the two segments can be (a) within the same object, (b) within different labeled objects, or (c) between a segment in a labeled object and an unlabeled background segment. We count (a) as a positive example and (b) and (c) as negative examples.

Even if we consider only segments associated with ground truth objects and their nearby neighbors, there still are far too many combinations to consider training a classifier on all of them. As a result, we must limit the training set to include only combinations of segments similar to ones that can be created by the hierarchical segmentation algorithm in the testing phase. To achieve this, the simplest approach would be to run the hierarchical segmentation on the training set and then add positive and negative examples as they are encountered by the algorithm. However, that approach is not possible because the hierarchical algorithm depends on the classifiers learned during the training phase. A chicken and egg problem.

We address this by iterating between learning the classifiers and generating example hierarchical segmentations. The process starts with classifiers trained on sampled combinations of nearby segments (within 1m). We then perform the hierarchical merging process (described in next section) to generate new candidate training examples. The resulting segments are labeled as positive and negative examples based on the truth data, and classifiers are then re-trained for the next iteration. We repeat this process until convergence (or at most 10 iterations). We refer to this process as *hierarchical example mining*.

Within each iteration, we execute merges between unlabeled segments (to encourage a variety of background segment examples - these are not, however, added to the training set) and within the same truth segment. Whenever a merge is executed, we consider all merge candidates created between the resulting segment and neighbors. If a merge is a positive example but has low probability according to the current classifier, or is a negative example with a high probability, then we add it to the training set. To keep the candidate segment set a manageable size and to add diversity through randomization, we randomly sample a subset of $10K$ examples from the training set before re-training the classifiers.

We then train a random forest [4] from this training set using the Shark Machine Learning Library [12] with default parameters. During experiments, we found that it provides good classification rates in comparison to alternatives and use it for all classification problems in this paper.

Merge Classifier: the second classifier estimates the probability $M(A, B)$ that two segments A and B are part of the same semantic object.

To learn this classifier, we must address three issues. The first is the same problem encountered with the Segment Classifier: exponential combinations of possible segment merges to train on. Our solution to this problem is the same as described for the Segment Classifier – i.e., we integrate learning the Merge Classifier with the Segment Classifier by using the same sets of positive and negative training examples. However, in this case, the training examples are pairs of segments to merge rather than the merged segments themselves. We say that a merge is correct if the resulting segment lies wholly within a ground truth object.

The second issue is to create a discriminating feature vector representing each merge between two segments, A and B, to be used by the Merge Classifier. Our solution is to concatenate four feature vectors: 1-2) the feature vectors for both segments, 3) the differences between the feature vectors for the two segments, and 4) the feature vector computed from the merged segments. We also include other merge specific features related to the relative alignment of the two merge candidates, including: (1) the amount of overlap along each dimension (if A's bounding box is contained inside B's bounding box along one dimension, then the overlap is the A's size along that dimension), and (2) the distance between the two candidates centroids along each dimension and in 3D. Together, this descriptor allows the Merge Classifier to consider the shapes of the two candidate segments, the differences between their shapes, the shape of the combined segment, and the relative placements of the candidate segments. The latter features are especially important because they indicate whether candidate segments are stacked vertically or one projects from the other (as is often the case in street lights).

The third issue is class-specificity. For the Segment Classifier, we train on examples from each object class separately. This makes sense for that classifier because the shapes and spatial contexts of each object class are largely different. However, for the Merge Classifier, it is possible to consider a class-independent classifier that is learned from examples of all classes together. This approach takes advantage of similarities in how parts merge into whole objects across all classes, and provides the classifier with significantly more data to train on. During experiments, we found that this class-independent approach to training the Merge Classifier performs better than a class-dependent one, and so we use it in all results of this paper.

4.3. Searching for Hierarchical Segmentations

During a testing phase, our system takes as input a novel preprocessed LIDAR scan, the two classifiers learned in the training phase, and a target object class C. It outputs a semantic segmentation of the novel scan, where each output point is associated with a segment and given a probability that it is part of an object in class C.

The hierarchical segmentation algorithm starts with segments output by the preprocessing phase. It builds a graph, where nodes represent segments and edges represent potential merges between them. An edge is added between two segments if they have two points within some fixed distance in 3D (1 meter in our experiments). The algorithm then iteratively collapses edges of the graph (merges segments) in priority order (as estimated by the Merge Classifier) as long as: 1) the probability the merge is correct is greater than 0.5 (as estimated by the Merge Classifier), and 2) the probability that the merged segment is part of an object of class C is greater than a threshold (0.01 in our experiments). The hierarchical agglomeration process continues until no further merges are possible meeting these criteria.

Each segment in the final set of segments is classified with the Segment Classifier to produce final probabilities for object class C. Those probabilities are assigned to all points within the segment in cases where pointwise predictions are required (as in our classification experiment in the next section).

5. Experimental Results

In this section, we report results of experiments aimed at investigating how well our algorithms are able to perform semantic segmentation of objects in street view LIDAR scans of cities.

Input Data Set: Our experiments were run on 3D point cloud data acquired by the side-facing LIDAR scans of the R5 Google Street View (GSV) cars driven through a $3.5km^2$ region of lower Manhattan. The data provided by Google is partitioned into 20 "runs," each collected during a different time a GSV car went out to capture data. In total, all the runs cover approximately 100 city blocks and contain 2.9M vertical scanlines and 390M LIDAR points.

Ground-Truth Annotation: We generated ground-truth semantic segmentations for 6 of the largest runs manually using an interactive tool that allows a person to merge, split, and label segments of LIDAR points through key-clicks in an interactive viewer. The viewer starts with the oversegmentation output from the preprocessing steps described in Section 4.1 and then asks a person to split segments that span more than one object, merge segments that contain parts of the same object, and select a semantic label for all objects of certain target object classes (trees, cars, vans, trucks, street lights, traffic lights, traffic signs, etc.). The result is a ground-truth annotation with all instances of the target classes segmented and labeled (statistics for some runs and classes appear in Table 5). In total, the ground truth contains 5,633 labeled objects across all runs.

We focus on vehicle and lighting classes in our experiments because they are well sampled in our dataset. Additionally, each shares similar parts with other objects in the dataset. Vans and cars appear in similar contexts, but the van is clearly differentiated by its size difference. Traffic lights and street lights are a particularly good test of whether improving the segmentation improves classification accuracy because sections of a traffic light are often only differentiated from traffic signs or other poles when the top section (which stretches over the street) is considered.

Experiment Methodology: All experiments were run with a leave-one-out cross validation, where splits coincide with non-overlapping GSV runs. We train on the ground-truth data for all but one run using the algorithms described in Section 4.2 and then test on the remaining run using the algorithms described in Section 4.3. Once all splits are processed, we average results across all runs to produce final statistics.

Segmentation Results: We first present results that investigate whether our hierarchical semantic segmentation algorithm produces better segmentations than the inital baseline ones provided by the preprocessing step.

Run	Car	Van	Truck	Tree	Street Light	Traffic Light	Stop Sign
NYC 0	291	36	71	109	54	41	16
NYC 0, side 2	194	44	17	91	34	28	38
NYC 11	50	2	12	67	21	14	5
NYC 11, side 2	35	6	5	161	26	28	2
NYC 12	324	52	40	131	61	55	12
NYC 14	82	12	4	107	17	15	6
Total	976	152	149	666	213	181	79

Table 1. Number of labeled objects for the target object classes.

To address this question quantitatively, we measure the quality of the highest overlap segment with each instance of the class in the ground truth data using a Jaccard Index. That is, for each object in the ground truth, we find the segment with highest overlap from the predicted segmentation and then calculate $\frac{\text{\# intersecting points}}{\text{\# points in union}}$. We then plot the cumulative distribution of the Jaccard indices – i.e., where each point on a curve indicates the fraction of ground truth objects (y-axis) with Jaccard index below a threshold (x-axis).

Figure 3 shows the results for several object classes (cars, vans, street lights, and traffic lights). Note that Jaccard indices achieved with our hierarchical semantic segmentation algorithm (solid blue curve) are better than those achieved with the baseline segmentation algorithm (dotted green curve). This indicates that the hierarchical algorithm was able to learn classifiers that provide useful merge ordering and termination criteria.

Representative segmentation results are shown in Figure 4 (see the supplemental materials for more examples). The segmentation results of the baseline preprocessing step are shown directly to the left of the corresponding results of the hierarchical merging algorithm. Note that preprocessing segmentations, which are based solely on objectness using algorithms from Golovinskiy et al. [8], can separate objects into multiple parts due to noise, undersampling, and occlusion in the data, resulting in segments that provide low probabilities for the learned object classifier (shown as gray values). The hierarchical algorithm merges these parts to form segments with higher classification probabilities. Of course, the algorithm is not perfect – the example in the top-right of Figure 4 shows a case where the end of the traffic light was not merged with its pole, probably because it was already highly likely to be part of a traffic light and merging did not increase its probability in this case.

Classification Results: We next present results that investigate the classification performance of our hierarchical semantic segmentation algorithm.

For this experiment, we aim to evaluate results independent of the segmentations. So, rather than count correctly labeled objects, we count correctly labeled LIDAR points. That is, we predict a class label for every LIDAR point in the test data set and then produce precision-recall plots depicting how the points' predicted class labels agree with the

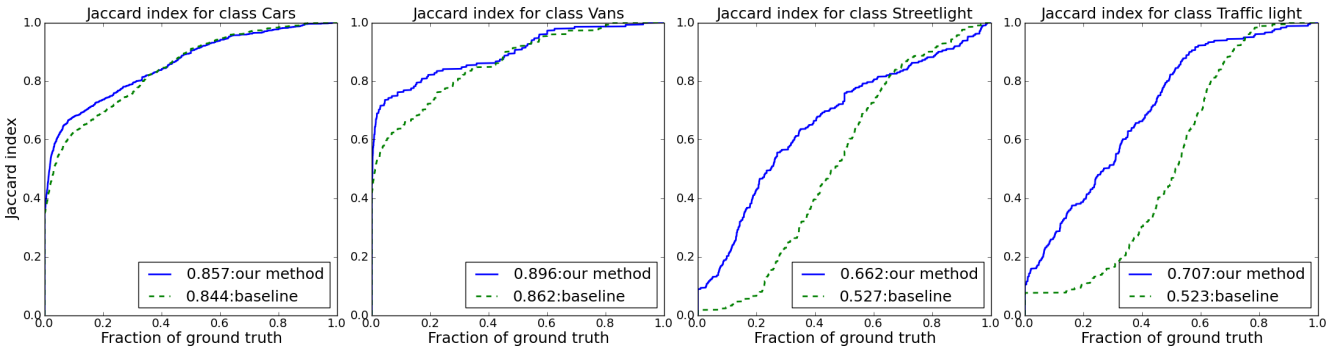

Figure 3. Cumulative distribution of Jaccard indices for cars, vans, street lights, and traffic lights using our hierarchical semantic segmentation algorithm (solid blue curve) versus the baseline segmentation (dotted green curve).

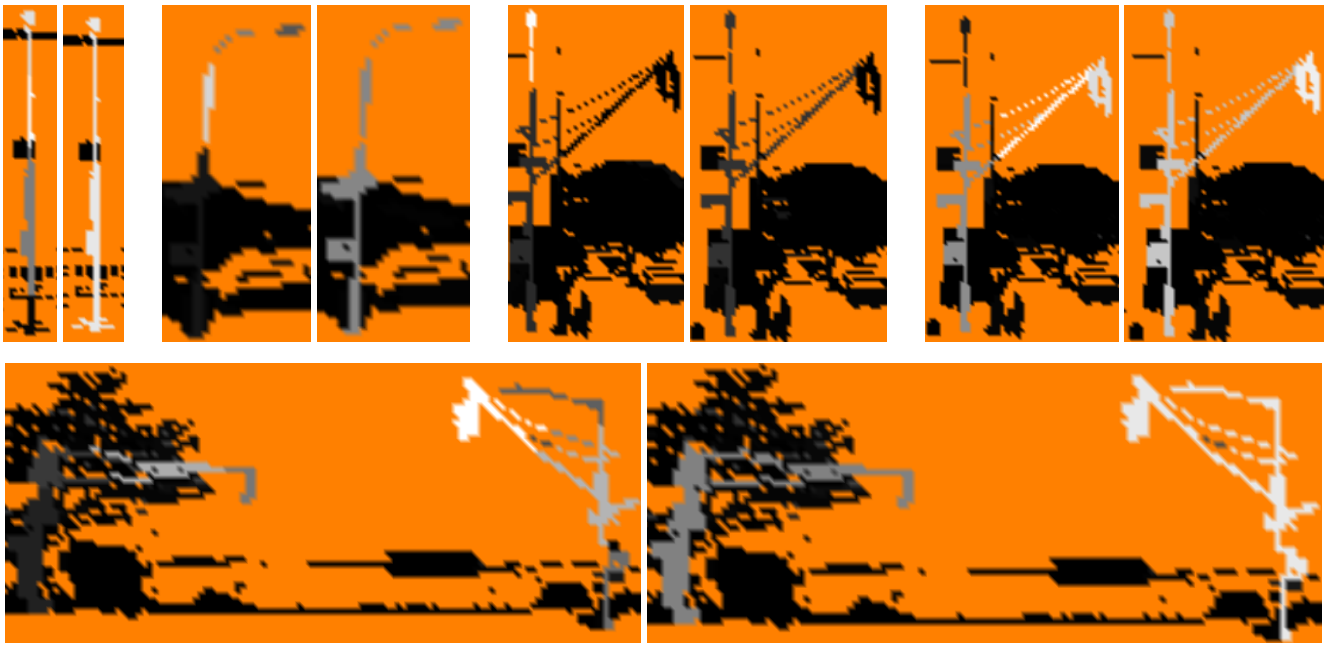

Figure 4. Comparisons of segmentations produced by the initial baseline algorithm (left) and ones produced by our hierarchical merging algorithm (right). Gray-levels are proportional to the probability that each segment is a member of the target object class according to our learned classifier (street light for the first three examples, and traffic light for the last two). Orange represents no LIDAR return. Please note that the quality of segmentation and predicted classification probability is generally increased in the hierarchical segmentation results.

ground truth. This evaluation method has the disadvantage that larger objects get more weight in the results, but has the advantage that direct comparison of classification results is possible across a range of segmentation algorithms.

We compare our algorithm to two alternatives: 1) one that applies a Point Classifier to every point independently (i.e., a classifier that predicts the probability $p(C|a)$ that a given point a is part of an object of class C based on its feature vector alone, and 2) one that applies the Segment classifier $P(C|A)$ to every initial segment A output by the preprocessing phase. These baseline algorithms are representative of simple approaches taken by previous work [8].

Figure 5 shows precision-recall curves for our method (blue curves) and the two alternatives for four different object classes. In all cases, the classification results with our algorithm are at least as good and sometimes better than the alternative approaches. For Cars, the *Segment* classifier is very good, even without hierarchical merging, because features such as "signed distance to the closest curb" and "height off the ground" already provide great cues for detecting Cars. However, for the other more difficult objects, in particular Traffic Lights, which come in a variety of positions and shapes, the hierarchical segmentation algorithm provides a significant improvement in classification rates in comparison to both alternative approaches.

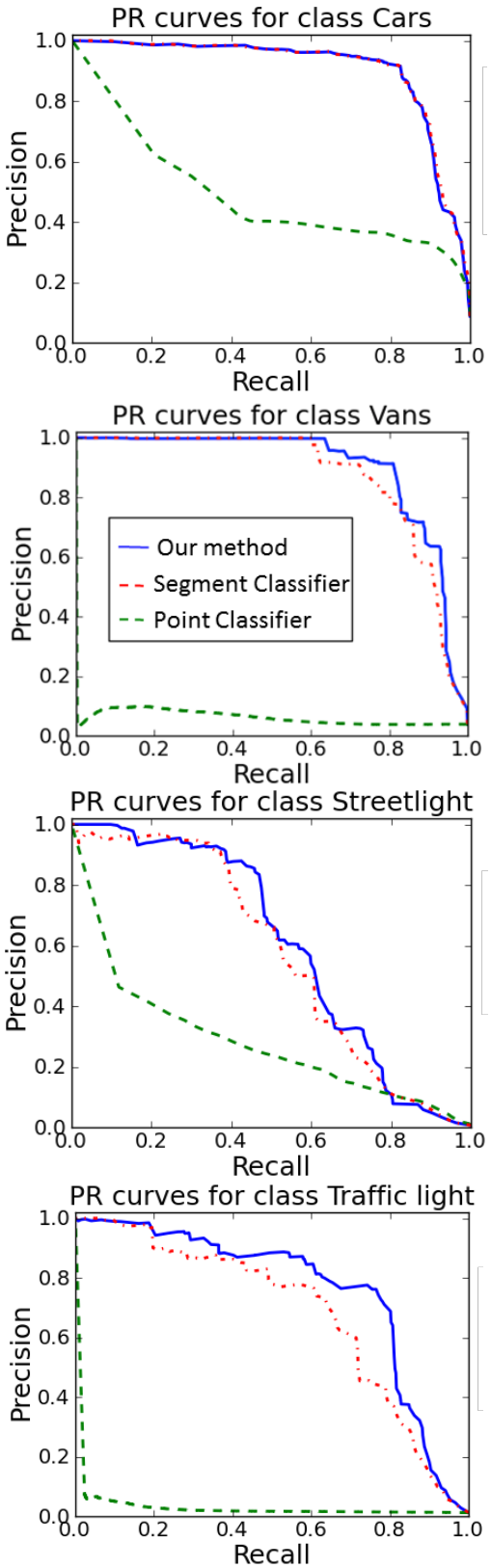

Figure 5. Precision-recall curves showing object classification performance for classifiers trained on individual points (dotted green curve), initial baseline segments (dotted red curve), and hierarchically merged segments – i.e., our method (blue curve).

6. Conclusion and Future Work

This paper investigates whether it is possible to learn classifiers that guide a hierarchical agglomeration algorithm to produce semantic segmentations for a target object class. We learn classifiers that predict the probability that two segments should be merged and the probability that a segment represents an object of the target class. These learned classifiers are used to guide a hierarchical search over possible segmentations towards ones that produce high probability semantic segmentations. Experiments on large-scale Google Street View scans of New York City suggest this algorithm performs at least as well as alternatives and sometimes better.

This work suggests several avenues for future work. First, the segmentation algorithm used during the preprocessing phase could be improved as it sometimes undersegments scans. Testing the methods with a finer oversegmentation (potentially even learning to merge individual points), or learning to merge across multiple oversegmentations of the same point cloud could improve results. Second, it would be informative to apply techniques such as CRFs to point and segment classifications to provide additional points for comparison, potentially using the affinity learning methods presented here to learn the pairwise potentials between segments. Third, it will be interesting to compare and combine object recognition results obtained from the LIDAR data (as in this project) with ones obtained from image data available with Google Street View. Finally, it will be interesting to consider how the proposed methods could be integrated into a semi-automatic semantic segmentation tool – e.g., when the user selects a region to segment, our system can present predicted hypotheses for the user to choose between. These and other extensions of the proposed method should be fruitful topics for further research.

This paper would not have been possible without the generosity of Google, Intel, and NSF. All three provided funding, and Google provided data (special thanks to Tilman Reinhardt and Aleksey Golovinskiy). We also thank Amy Ousterhout, Fisher Yu, and Aleksey Boyko, who laid some of the early ground work for the project, and Robert Matejek, who provided ground truth labelings.

References

[1] A. K. Aijazi, P. Checchin, and L. Trassoudaine. Segmentation based classification of 3d urban point clouds: A supervoxel based approach with evaluation. *Remote Sensing*, 5(4):1624–1650, 2013. 2

[2] D. Anguelov, B. Taskarf, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. Discriminative learning of markov random fields for segmentation of 3d scan data. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2005. 2

[3] S. Avidan and A. Shamir. Seam carving for content-aware image resizing. *ACM Transactions on Graphics (SIGGRAPH)*, 26(3), 2007. 4

[4] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 5

[5] M. De Deuge, F. Robotics, A. Quadros, C. Hung, and B. Douillard. Unsupervised feature learning for classification of outdoor 3d scans. *Australasian Conference on Robitics and Automation*. 2

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

[7] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik. Recognizing objects in range data using regional point descriptors. In *ECCV*, pages 224–237. 2004. 2

[8] A. Golovinskiy, V. G. Kim, and T. Funkhouser. Shape-based recognition of 3d point clouds in urban environments. In *International Conference on Computer Vision (ICCV)*, pages 2154–2161, 2009. 2, 4, 6, 7

[9] C. Gu, J. J. Lim, P. Arbeláez, and J. Malik. Recognition using regions. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2009. 2

[10] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik. Indoor scene understanding with rgb-d images: Bottom-up segmentation, object detection and semantic segmentation. *International journal of Computer Vision (IJCV)*, 2014. 2

[11] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik. Aligning 3d models to rgb-d images of cluttered scenes. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[12] C. Igel, V. Heidrich-Meisner, and T. Glasmachers. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008. 5

[13] V. Jain, S. C. Turaga, K. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung. Learning to agglomerate superpixel hierarchies. In *Advances in Neural Information Processing Systems*, pages 648–656, 2011. 2

[14] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 21(5):433–449, 1999. 2

[15] J. Knopp, M. Prasad, G. Willems, R. Timofte, and L. Van Gool. Hough transform and 3d surf for robust three dimensional classification. In *ECCV*. 2010. 2

[16] K. Lai, L. Bo, and D. Fox. Unsupervised feature learning for 3d scene labeling. In *IEEE Robotics and Automation (ICRA)*, 2014. 2

[17] K. Lai, L. Bo, X. Ren, and D. Fox. Detection-based object labeling in 3d scenes. In *IEEE Robotics and Automation (ICRA)*, pages 1330–1337, 2012. 2

[18] K. Lai and D. Fox. 3d laser scan classification using web data and domain adaptation. In *Robotics: Science and Systems (RSS)*, 2009. 2

[19] F. Moosmann and M. Sauerland. Unsupervised discovery of object classes in 3d outdoor scenarios. In *International Conference on Computer Vision (ICCV)*, pages 1038–1044, 2011. 2

[20] L. Nan, K. Xie, and A. Sharf. A search-classify approach for cluttered indoor scene understanding. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 31(6), 2012. 2

[21] J. Nunez-Iglesias, R. Kennedy, T. Parag, J. Shi, and D. B. Chklovskii. Machine learning of hierarchical clustering to segment n-dimensional images. *CoRR*, abs/1303.6163, 2013. 2

[22] S. Pu, M. Rutzinger, G. Vosselman, and S. O. Elberink. Recognizing basic structures from mobile laser scanning data for road inventory studies. *Journal of Photogrammetry and Remote Sensing (ISPRS)*, 66(6, Supplement):S28 – S39, 2011. 2

[23] X. Ren, L. Bo, and D. Fox. Rgb-(d) scene labeling: Features and algorithms. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 2759–2766, 2012. 2

[24] A. Serna and B. Marcotegui. Detection, segmentation and classification of 3d urban objects using mathematical morphology and supervised learning. *Journal of Photogrammetry and Remote Sensing (ISPRS)*, 93:243–255, 2014. 2

[25] C.-H. Shen, H. Fu, K. Chen, and S.-M. Hu. Structure recovery by part assembly. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 31(6):180:1–180:11, 2012. 2

[26] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2008. 2

[27] R. Socher, C. C. Lin, A. Y. Ng, and C. D. Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *International Conference on Machine Learning (ICML)*, 2011. 2

[28] S. Song and J. Xiao. Sliding shapes for 3d object detection in depth images. In *ECCV*, pages 634–651. Springer, 2014. 2

[29] J. Tighe and S. Lazebnik. Finding things: Image parsing with regions and per-exemplar detectors. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2013. 2

[30] A. Velizhev, R. Shapovalov, and K. Schindler. Implicit shape models for object detection in 3d point clouds. In *International Society of Photogrammetry and Remote Sensing Congress*, 2012. 2

[31] X. Xiong, D. Munoz, J. A. Bagnell, and M. Hebert. 3-d scene analysis via sequenced predictions over points and regions. In *IEEE Robotics and Automation (ICRA)*, pages 2609–2616, 2011. 2

[32] B. Yang, Z. Dong, G. Zhao, and W. Dai. Hierarchical extraction of urban objects from mobile laser scanning data. *Journal of Photogrammetry and Remote Sensing (ISPRS)*, 99(0):45 – 57, 2015. 2

[33] J. Yao, S. Fidler, and R. Urtasun. Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2012. 2

[34] A. Zelener, P. Mordohai, and I. Stamos. Classification of vehicle parts in unstructured 3d point clouds. *International Conference on 3D Vision (3DV)*, 2014. 2

[35] X. Zhu, H. Zhao, Y. Liu, Y. Zhao, and H. Zha. Segmentation and classification of range image from an intelligent vehicle in urban environment. 2010. 2