CS229 Final Project - Improving LiDAR Point Cloud Classification of Urban Objects

Peggy (Yuchun) Wang
Department of Computer Science
Stanford University
Stanford, CA
peggy.yuchun.wang@cs.stanford.edu

Tristan Gosakti
Department of Computer Science
Stanford University
Stanford, CA
tgosakti@stanford.edu

Abstract

LiDAR point cloud classification is hugely important for autonomous driving applications. In this study, we attempt to improve classifications of the Sydney Urban Objects Dataset by experimenting with data augmentation methods and modifying architectures for VoxNet, a deep learning model which previously had good performance on this dataset. We were able to improve the F1 score by 3.32% by removing outlier points from the raw point cloud and improve the F1 score by 4.75% after adding an additional Max Pool layer to the original VoxNet architecture.

1 Introduction

With the rise in popularity and prevalence of autonomous driving, LiDAR (short for Light Detection and Ranging) has become a popular way for cars to glean insight into its surroundings. Using the sparse point clouds generated from LiDARs, self-driving cars are able to estimate distances to and classify different objects for use in autonomous navigation, localization, and obstacle avoidance. Common applications in autonomous driving after point cloud classification include swerving around bikers, identifying landmarks, and modeling the type of vehicle around the agent. Therefore, urban object classification from sparse point clouds is essential for autonomous driving perception systems.

The input to our models is the 32x32x32 voxelized occupancy grid of this point cloud data (described in the Dataset and Features section below). The output is the predicted object classification of the urban object. We used a 3D convolutional neural network (CNN) based off of the VoxNet architecture [4] as our model. We tried different network architectures with different layers, amount of max-pools, data preprocessing, and regularization.

2 Related Work

Classical machine learning methods in point cloud detection include Support Vector Machines (SVM) and Random Forest (RF), and have typically relied on a range of hand crafted shape descriptors [3]. With recent breakthroughs in deep learning, 3D CNNs were used in point cloud classifications. One of the earliest deep learning models done on classifying point cloud objects with LiDARs was done with VoxNet [4], which had a very good performance on the Sydney Urban Objects dataset with a F1 score of 71%. VoxNet converts the raw point cloud data into an voxel occupancy grid, represented by a 32 x 32 x 32 array. Each element in the array is a probabilistic estimate of its occupancy as a function of incoming sensor data and prior knowledge. The array is then passed through a series of 3D convolutional layers and max pool layers to output a predicted class. We base our approach heavily on VoxNet.

More recently, studies have classified point clouds by using the raw point cloud (x, y, z) coordinates directly as input rather than transforming the point cloud to a voxelized grid. PointNet [5] uses multi-layer perceptrons (MLPs) to transform the N x 3 point cloud array directly to a predicted output class by learning features and significant points. PointNet++ [6] builds on top of the PointNet approach by using a hierarchical neural network structure that applies PointNet recursively on a nested partitioning of the input point set, which is then able to learn local features. However, both PointNet and PointNet++ were trained on ShapeNet [1], which contains dense and sampled point cloud datasets based on 3D CAD models of objects. The ShapeNet point clouds therefore differs from LiDAR point clouds, which are typically sparse and noisy.

VoxelNet [9] also performs end-to-end learning on LiDAR point clouds directly. However, the input to VoxelNet is a LiDAR point cloud of many objects, and VoxelNet then uses a sliding window method and converts points to regional voxels. They then use a fully connected layer to convert each voxel region into a feature vector and uses 3D convolutional layers to detect objects. However, VoxelNet includes many objects in a single point cloud and only detects three objects in the original paper. Due to those limitations, we believe VoxNet, which performs LiDAR point cloud classification on a single object that is already segmented, is the best architecture for our purposes.

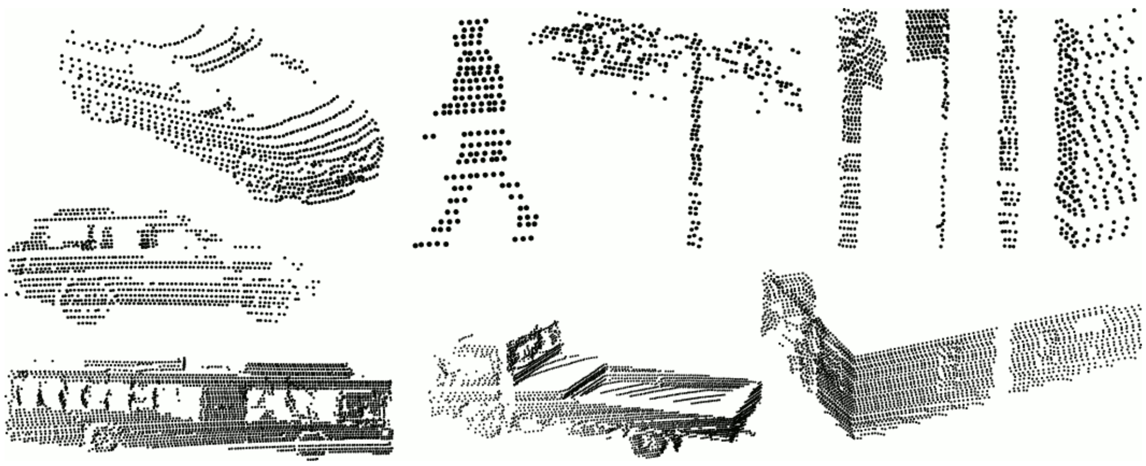


Figure 1: SUOD Dataset Point Cloud Examples [7]

3 Dataset and Features

3.1 Dataset

We used the Sydney Urban Objects Dataset (SUOD) [7], which contains a variety of common urban road objects scanned with a Velodyne HDL-64E LIDAR. Visual examples of this dataset is shown in Figure 1. This dataset was used for the original VoxNet paper [4] and has realistic examples of LiDAR point clouds. In SUOD, each object contains several feature categories – in addition to the standard (x, y, z) coordinate representation of each point in the point cloud, there is also information on azimuth angles, laser return intensity, laser id, and several more obscure features. For point cloud classifications, we will only need to use the (x, y, z) point coordinates. Each object appeared in each point cloud at most one time, so there won't be instances of an object containing multiple cars, for example.

The dataset contains 631 Velodyne LiDAR point clouds with 26 unique classes (entire distribution in the Appendix). There are several classes where the number of instances that shows up in the dataset is so few (less than 5 instances per category) that it wouldn't be feasible to try classifying and testing unless more data is available. For that reason, we only pick the 14 classes that appear more than 10 times, which results in 588 objects. Interestingly, there are classes that are very prone to overlap, such as different types of road vehicles (4wd, bus, car, ute, van), which makes this classification problem more challenging.

3.2 Features

The (x, y, z) points of each object is then transformed to a N x 3 matrix, where N is the number of points in the point cloud of a specific object. N differs for each point cloud. Following the original VoxNet approach [4], we chose to represent this data through voxel occupancy grids, where we turn the point cloud data into a volumetric grid representing our estimate of spatial occupancy. These occupancy grids represent the state of the environment as a 3D lattice of random variables (each corresponding to a voxel) and maintain a probabilistic estimate of their occupancy. There are many advantages to representing our input like this – one of them is that they can be stored and manipulated by efficient data structures.

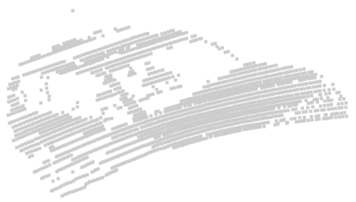
3.3 Data Augmentation

Since the dataset has a small number of examples (588 objects among 14 classes), following the original VoxNet approach, we perform data augmentation in order to generate a bigger training set. The original dataset of 588 objects is split into 4 folds (3 training folds and 1 test fold), where each fold contains at least one instance of each class. There is a total of 433 objects in the training set and 155 objects in the testing set. Each object was then augmented by being uniformly randomly rotated and translated 12 times, for a total of 13 objects (12 augmented + 1 original) per one original object. We now have a total of 5629 objects in our training set and 2015 objects in our test set.

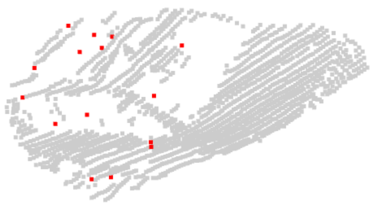
4 Methods

4.1 VoxNet Baseline

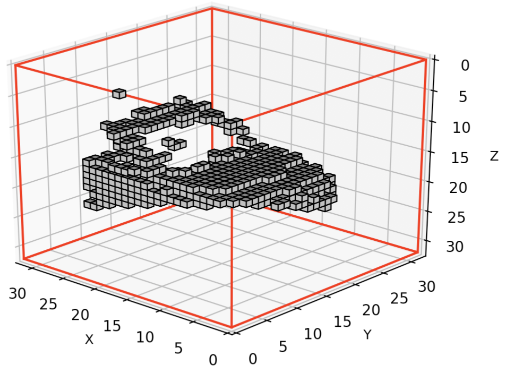
For the baseline point cloud classification, we used the VoxNet structure (Figures 3) with hyperparameters given in Table 1. We modified the VoxNet code implemented in Tensorflow from [2].



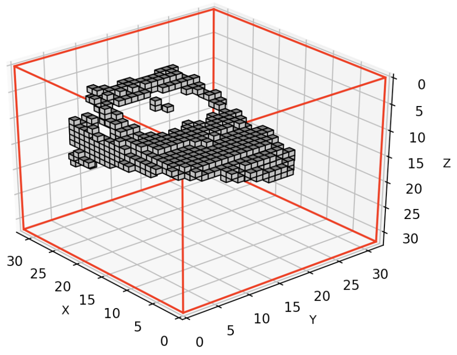
(a) Original Point Cloud



(b) Remove Outliers (red points)



(c) Voxelization



(d) Voxelization After Removing Outliers

Figure 2: 4WD Point Cloud Example Visualizations. (a) shows a visualization of the original point cloud of a Four Wheel Drive (label 4WD). (b) shows the red outliers that are removed in the original point cloud. (c) and (d) are comparisons of the point cloud voxel before and after removing the outlier points, respectively.

4.2 Remove Outliers

We attempted to see if we can improve the results using data preprocessing by removing outlier points from the point cloud, since LiDAR point clouds are noisy. Before the data augmentation step, we removed points that are further away from their 3 nearest neighbors compared to the average +/- 3.0 standard deviation for the point cloud. We used the *open3d.remove_statistical_outlier(nb_neighbors=3, std_ratio=3.0)* method from the Open3d library [8] to do so. A visualization of the outlier points to be removed is shown in Figure 2b. We then trained our preprocessed dataset using the VoxNet architecture with the same hyperparameters.

4.3 VoxNet Architecture Changes

We tried several different architectures on the original augmented dataset without outlier removal, with the same hyperparameters as Table 1. The better-performing ones include the VoxNet-DMP (double max pool) and VoxNet-Conv (another double max pool but with the same double initial convolutions as the baseline), outlined in Figure 4.

The VoxNet-DMP model is made of two layers of 3D Convolution + Max Pool and two fully connected layers, with standard kernel sizes and stride lengths (outlined in figure 4). The VoxNet-Conv model also has two Max Pools but retains the double convolutions of different kernel sizes at the beginning that the VoxNet baseline has. We decided to attempt a more complex model because we thought that a single Max Pool layer wouldn't be enough to capture the complexities of a 3D model. Thus, we decided to add another Max Pool layer. We kept the starting kernel with stride of 5 because as we've seen from the occupancy grid visualizations, objects that are both labelled cars doesn't necessarily have similar shapes (a car isn't identical to the next), so we wanted a big enough kernel to generalize.

4.4 VoxNet Regularization

We also tried regularization in an attempt to prevent overfitting. We added a kernel regularizer on the two convolutional layers in the VoxNet baseline with lambda of 0.01 and 0.1. We kept the other hyperparameters the same as the ones given in Table 1.

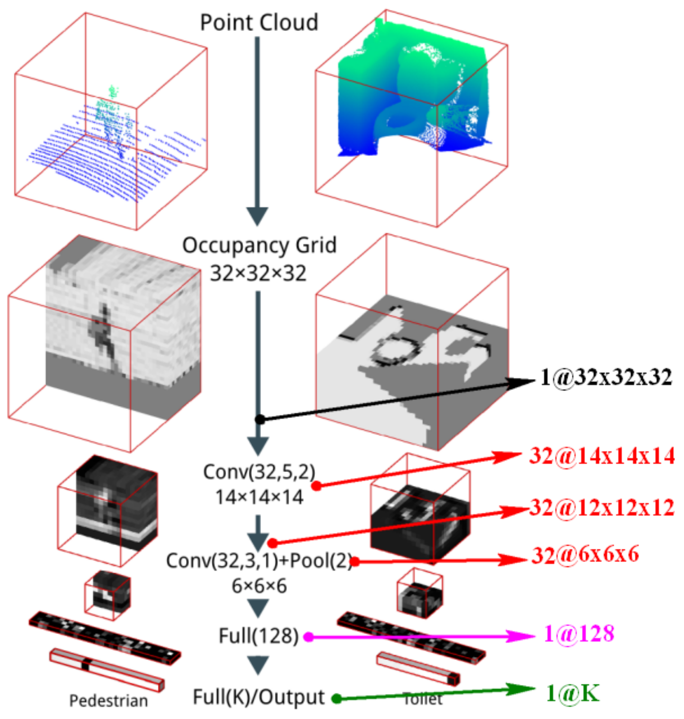


Figure 3: A Visualization of the VoxNet Architecture [4]

Name	VoxNet	VoxNet-DMP	VoxNet-Conv
Layer 1	3D Conv - 32F-5K-S2	3D Conv - 32F-5K-S2	3D Conv - 32F-5K-S2
Layer 2	3D Conv - 32F-3K-S1	Max Pool 2x2 - S2	3D Conv - 32F-3K-S1
Layer 3	Max Pool 2x2 - S2	3D Conv - 32F-3K-S1	Max Pool 2x2 - S2
Layer 4	FC-128	Max Pool 2x2 - S2	3D Conv - 32F-3K-S1
Layer 5	FC-14	FC-128	Max Pool 2x2 - S2
Layer 6	-	FC-14	FC-128
Layer 7	-	-	FC-14

Figure 4: Network Architectures of VoxNet, VoxNet-DMP, and VoxNet-Conv. In the 3D Conv and Max Pool Layers, 32F refers to filters = 32, 5K refers to kernels = 5, and S2 refers to stride = 2. FC-128 refers to the fully connected layer with 128 units, and FC-14 refers to the fully connected layer with 14 outputs.

5 Results and Discussion

Our results are shown in Table 3. The original VoxNet paper had an F1 score of 71% [4], which we were unable to replicate with our baseline VoxNet (62% F1 score). However, we were able to make substantial improvements over the baseline model.

We found that simply by performing preprocessing by removing outliers from the raw point cloud, we were able to improve the F1 score by 3.32%. This does not come as a surprise because raw LiDAR data tend to be very noisy. Removing outliers also changes the voxelization representation, as shown in Figures 2c and 2d, where an outlier voxel that was in the top left corner of 2c was removed in 2d.

Our best performing architecture, VoxNet-DMP, improved the F1 score by 4.75% after adding an additional Max Pool layer to the original VoxNet architecture. This result was with the original non-preprocessed point cloud. We believe that the result was because of the additional Max Pool layer after the first convolutional layer, which effectively acts as a non-linear activation function for convolutional layers.

While we made improvements over the baseline model, we noticed that our top F1 score is still below 70%. We compared our test accuracy with our train accuracy and saw that our train accuracy was much higher (92% training accuracy for the baseline versus 64%

Table 1: Hyperparameters

Hyperparameter	Value
Total Steps	1381
Number of Classes	14
Voxel Size	24x24x24
Voxel Zero Padding Size	32x32x32
Conv Activation Function	Leaky RELU
FC Activation Function	RELU
Loss Function	Softmax Cross Entropy
Number of Epochs	8
Learning rate	0.001
Optimizer	Adam
Batch Size	32

testing accuracy), suggesting that the model was overfitting. To explore if this were the case, we tried several regularization techniques on the baseline.

We tried using a kernel regularizer and discovered several things. First of all, for all regularization rates, the training time dropped significantly (average of 41 seconds per 100 steps versus an average of 430 seconds per 100 steps for without regularization). We found that the regularization with lambda 0.01 was too small and didn't have any meaningful changes in training or testing accuracy. On the other hand, a lambda of 0.1 resulted in lower train and test accuracies, suggesting that the model was too regularized to learn the proper weights. Other attempts of lambdas (0.05, 0.07) yielded in lower testing accuracy as well.

Our end models had the same hyperparameters as the original baseline VoxNet (Table 1). We found that these hyperparameters were best – when we tried tweaking them in different models, they reduced accuracy. Letting the model train for more total steps tended to increase the training accuracy but reduce the testing accuracy, suggesting this resulted in overfitting. The loss function plateaus out at around 800 steps. We used an Adam optimizer to speed up the learning rates and decided that number of epochs 8 and batch size of 32 is a reasonable number to keep because it balances learning the weights with training time well on our tests.

We think that it is reasonable to assume that the dataset is the cause for many of these limitations. First of all, our dataset is very small with 588 objects. With 13x data augmentation, we still only have a total of 7644 objects for 14 classes (average about 546 objects per class). Further, many objects to classify have very common overlapping features: bus, car, trailer, truck, trunk, ute, van. This overlap substantially increases the chances of misclassifications. Lastly, the data shown is virtually always occluded and only has a small portion of it represented in the occupancy grid (it would be difficult to classify whether an object is a car versus a 4x4 when only the windshield and front portion of the vehicle isn't occluded – a real example we encountered). Our most common misclassifications were amongst these objects.

Table 3: Results

Model	Train Acc	Test Acc	Test Precision	Test Recall	Test F1	Test Loss
VoxNet (Baseline)	0.9231	0.6454	0.6168	0.6454	0.6237	2.7060
VoxNet-Outliers	0.9750	0.6653	0.6660	0.6653	0.6563	3.3907
VoxNet-DMP	0.9563	0.6953	0.6558	0.6952	0.6712	2.0662
VoxNet-Conv	0.8995	0.6429	0.6001	0.6429	0.6170	1.9999
VoxNet-Reg (λ: 0.01)	0.9218	0.6536	0.6208	0.6546	0.6306	2.5903
VoxNet-Reg (λ: 0.1)	0.7670	0.5849	0.5121	0.5386	0.5386	2.7933

Figure 5: Results of experiments. VoxNet (Baseline) refers to vanilla VoxNet as described in [4], with the architecture described in Figure 4. VoxNet-Outliers refers to the VoxNet architecture with preprocessing by removing outlier points. VoxNet-DMP and VoxNet-Conv refers to the corresponding network architectures described in in Figure 4. VoxNet-Reg refers adding a regularization to convolution layers with the corresponding λ values to the VoxNet (Baseline) architecture.

6 Conclusion and Future Work

We manage to reach a testing F1 score of 67%, which is a substantial improvement of 5% over our baseline of 62%, a feat that shouldn't be understated considering the difficulty of the data we were working with. That being said, there is future work that we think is worth trying. For example, we might try further augmenting the data by adding random noise for points in addition to the translations and rotations. Scaling the voxelizations slightly may also help (to account for the different potential shapes of cars, for example).

We think that while there is a long way to go until we reach 90%+ testing accuracy, we are excited with the increase in testing accuracy compared to the original paper's result of 71% accuracy [4].

A worthwhile attempt also includes building off of PointNet [5], a novel type of neural network which does not use voxelization and instead learns directly from the point cloud itself. However, since the PointNet paper used densely sampled point clouds from 3D models rather than LiDAR data, we were unsure about how well this architecture will perform on noisy and sparse LiDAR data, which we will try in the future.

Contributions

Peggy Wang: Implemented VoxNet Code, trained VoxNet on dataset, created images and figures. Ran experiments with outlier removal and implemented accuracy and F1 score metrics. Made poster. (Wrote most of abstract, introduction, results, related works sections in paper.)

Tristan Gosakti: Implemented, trained and tested VoxNet-DMP and VoxNet-Conv models. Tuned VoxNet with varying regularizers. (Wrote most of dataset and features, methods, results, and discussion sections in paper.)

Link to Code

`https://drive.google.com/drive/folders/1YLKJrta3JIAlenGOW-fmQPAJRbGzxreo?usp=sharing`

References

[1] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015. URL `http://arxiv.org/abs/1512.03012`.

[2] Durant35. Durant35/voxnet, Apr 2018. URL `https://github.com/Durant35/VoxNet`.

[3] D. Griffiths and J. Boehm. A review on deep learning techniques for 3d sensed data classification. *CoRR*, abs/1907.04444, 2019. URL `http://arxiv.org/abs/1907.04444`.

[4] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017. URL `http://arxiv.org/abs/1706.02413`.

[7] A. Quadros, J. Underwood, and B. Douillard. Sydney urban objects dataset, 2013. URL `http://www.acfr.usyd.edu.au/papers/SydneyUrbanObjectsDataset.shtml`.

[8] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[9] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017. URL `http://arxiv.org/abs/1711.06396`.

Appendix

Classes and their counts in the dataset: 4wd: 21, bench: 2, bicycle: 8, biker, 4 building: 20, bus: 16, car: 88, cyclist: 3, excavator: 1, pedestrian: 152, pillar: 20, pole: 21, post: 8, scooter: 1, ticket machine: 3, traffic lights: 47, traffic sign: 51, trailer: 1, trash: 5, tree: 34, truck: 12, trunk: 55, umbrella: 5, ute: 16, van: 35, vegetation: 2