VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition

Daniel Maturana and Sebastian Scherer

***Abstract*— Robust object recognition is a crucial skill for robots operating autonomously in real world environments. Range sensors such as LiDAR and RGBD cameras are increasingly found in modern robotic systems, providing a rich source of 3D information that can aid in this task. However, many current systems do not fully utilize this information and have trouble efficiently dealing with large amounts of point cloud data. In this paper, we propose *VoxNet*, an architecture to tackle this problem by integrating a volumetric Occupancy Grid representation with a supervised 3D Convolutional Neural Network (3D CNN). We evaluate our approach on publicly available benchmarks using LiDAR, RGBD, and CAD data. *VoxNet* achieves accuracy beyond the state of the art while labeling hundreds of instances per second.**

I. INTRODUCTION

Semantic object recognition is an important capability for autonomous robots operating in unstructured, real-world environments. Meanwhile, active range sensors such as LiDAR and RGBD cameras are an increasingly common choice of sensor for modern autonomous vehicles, including cars [1], quadrotors [2] and helicopters [3]. While these sensors are heavily used for obstacle avoidance and mapping, their potential for semantic understanding of the environment is still relatively unexplored. We wish to take full advantage of this kind of data for object recognition.

In this paper, we address the problem of predicting an object class label given a 3D point cloud segment, which may include background clutter. Most of the current state of the art in this problem follows a traditional pipeline, consisting of extraction and aggregation of hand-engineered features, which are then fed into an off-the-shelf classifier such as SVMs. Until recently, this was also the state of the art in image-based object recognition and similar tasks in computer vision. However, this kind of approach has been largely superseded by approaches based on Deep Learning [6], where the features and the classifiers are jointly learned from the data. In particular, the state of the art for image object recognition has been dramatically improved by Convolutional Neural Networks (CNNs) [7]. CNNs have since shown their effectiveness at various other tasks [8].

While it is conceptually simple to extend the basic approach to volumetric data, it is not obvious what architectures and data representations, if any, will yield good performance. Moreover, volumetric representations can easily become computationally intractable; perhaps for these reasons, 3D convolutional nets have been described as a "nightmare" [9].

Authors are with the Robotics Institute, Carnegie Mellon University, Forbes Ave 5000, Pittsburgh PA 15201 USA { dimatura, basti } at cmu.edu

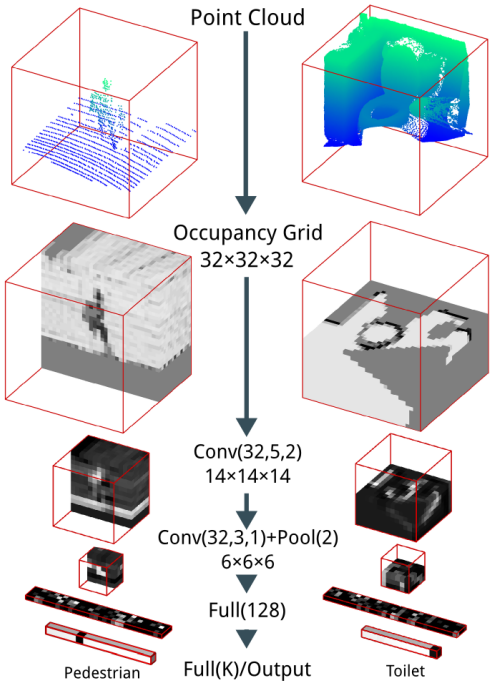

Fig. 1. The VoxNet Architecture. $Conv(f, d, s)$ indicates f filters of size d and at stride s, $Pool(m)$ indicates pooling with area m, and $Full(n)$ indicates fully connected layer with n outputs. We show inputs, example feature maps, and predicted outputs for two instances from our experiments. The point cloud on the left is from LiDAR and is part of the Sydney Urban Objects dataset [4]. The point cloud on the right is from RGBD and is part of NYUv2 [5]. We use cross sections for visualization purposes.

The key contribution of this paper is *VoxNet*, a basic 3D CNN architecture that can be applied to create fast and accurate object class detectors for 3D point cloud data. As we show in the experiments, this architecture achieves state-of-the-art accuracy in object recognition tasks with three different sources of 3D data: LiDAR point clouds, RGBD point clouds, and CAD models.

II. RELATED WORK

A. Object Recognition with Point Cloud Data

There is a large body of work on object recognition using 3D point clouds from LiDAR and RGBD sensors. Most of this work uses a pipeline combining various hand-crafted features or descriptors with a machine learning classifier ([10], [11], [12], [13]). The situation is similar for semantic segmentation, with structured output classifiers instead of single output classifiers ([14], [15], [16]). Unlike these approaches, our architecture learns to extract features and classify objects

from the raw volumetric data. Our volumetric representation is also richer than point clouds, as it distinguishes free space from unknown space. In addition, features based on point clouds often require spatial neighborhood queries, which can quickly become intractable with large numbers of points.

B. 2.5D Convolutional Neural Networks

Following the success of CNNs on tasks using RGB images, several authors have extended their use to RGBD data ([17], [18], [19], [20]). These approaches simply treat the depth channel as an additional channel, along with the RGB channels. While straightforward, this approach does not make full use of the geometric information in the data and makes it difficult to integrate information across viewpoints.

For LiDAR, [4] propose a feature that locally describes scans with a 2.5D representation, and [21] studies this approach in combination with a form of unsupervised feature learning. [22] propose an encoding that makes better use of the 3D information in the depth, but is still 2D-centric. Our work differs from these in that we employ a fully volumetric representation, resulting in a richer and more discriminative representation of the environment.

C. 3D Convolutional Neural Networks

Architectures with volumetric (i.e., spatially 3D) convolutions have been successfully used in video analysis ([23], [24]). In this case, time acts as the third dimension. Algorithmically, these architectures work the same as ours, but the nature of the data is very different.

In the RGBD domain, [25] uses an unsupervised volumetric feature learning approach as part of a pipeline to detect indoor objects. This approach is based on sparse coding, which is generally slower than convolutional models. In concurrent work, [26] propose a generative 3D convolutional model of shape and apply it to RGBD object recognition, among other tasks. We compare this approach to ours in the experiments.

In the LiDAR domain, [27] is an early work that studies a 3D CNN for use with LiDAR data with a binary classification task. There is also our own previous work [28], which introduced 3D CNNs for landing zone detection in UAVs. Compared to this work, we tackle a more general object recognition task with 3D data from different modalities. We also study different representations of occupancy and propose techniques to improve performance when the data varies significantly in scale and orientation.

III. APPROACH

The input to our algorithm is a point cloud segment, which can originate from segmentation methods such as [12], [29], or a "sliding box" if performing detection. The segment is usually given by the intersection of a point cloud with a bounding box and may include background clutter. Our task is to predict an object class label for the segment. Our system for this task has two main components: a volumetric grid representing our estimate of spatial occupancy, and a 3D CNN that predicts a class label directly from the occupancy grid. We describe each component below.

A. Volumetric Occupancy Grid

Occupancy grids ([30], [31]) 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 as a function of incoming sensor data and prior knowledge.

There are two main reasons we use occupancy grids. First, they allow us to efficiently estimate free, occupied and unknown space from range measurements, even for measurements coming from different viewpoints and time instants. This representation is richer than those which only consider occupied space versus free space such as point clouds, as the distinction between free and unknown space can potentially be a valuable shape cue. Second, they can be stored and manipulated with simple and efficient data structures. In this work, we use dense arrays to perform all our CNN processing, as we use small volumes (32^3 voxels) and GPUs work best with dense data. To keep larger spatial extents in memory we use hierarchical data structures and copy specific segments to dense arrays as needed. Theoretically this allows us to store a potentially unbounded volume while using small occupancy grids for CNN processing.

B. Reference frame and resolution

In our volumetric representation, each point (x, y, z) is mapped to discrete voxel coordinates (i, j, k). The mapping is a uniform discretization but depends on the origin, orientation and resolution of the voxel grid in space. The appearance of the voxelized objects depends heavily on these parameters.

For the origin, we assume it is given as an input, e.g. obtained by a segmentation algorithm or given by a sliding box.

For the orientation, we assume that the z axis of the grid frame is approximately aligned with the direction of gravity. This can be achieved with an IMU or simply keeping the sensor upright. This still leaves a degree of freedom, the rotation around the z axis (yaw). If we defined a canonical orientation for each object and were capable of detecting this orientation automatically, it would be reasonable to always align the grid to this orientation. However, it is often non-trivial in practice to detect this orientation from sparse and noisy point clouds. In this paper we propose a simple alternative based on data augmentation, discussed in III-F.

For the resolution, we adopt two strategies, depending on the dataset. For our LiDAR dataset, we use a fixed spatial resolution, e.g. a voxels of $(0.1\,\mathrm{m})^3$. For the other datasets, the resolution is chosen so the object of interest occupies a subvolume of $24 \times 24 \times 24$ voxels. In all experiments we use a fixed occupancy grid of size $32 \times 32 \times 32$ voxels. The tradeoff between these two strategies is that in the first case, we maintain the information given by the relative scale of objects (e.g., cars and persons tend to have a consistent physical size); in the second case, we avoid loss of shape information when the voxels are too small (so that the object is larger than the grid) or when the voxels are too large (so that details are lost by aliasing).

C. Occupancy models

Let $\{z^t\}_{t=1}^T$ be a sequence of range measurements that either hit ($z^t = 1$) or pass through ($z^t = 0$) a given voxel with coordinates (i, j, k). Assuming an ideal beam sensor model, we use 3D ray tracing [32] to calculate the number of hits and pass-throughs for each voxel. Given this information, we consider three different occupancy grid models to estimate occupancy:

Binary occupancy grid. In this model, each voxel is assumed to have a binary state, occupied or unoccupied. The probabilistic estimate of occupancy for each voxel is computed with log odds for numerical stability. Using the formulation from [31], we update each voxel traversed by the beam as

$$l_{ijk}^t = l_{ijk}^{t-1} + z^t l_{\text{occ}} + (1 - z^t) l_{\text{free}} \tag{1}$$

where l_{occ} and l_{free} are the log odds of the cell being occupied or free given that the measurement hit or missed the cell, respectively. We set these to the values suggested in [33], $l_{free} = -1.38$ and $l_{occ} = 1.38$ and clamp the log odds to $(-4, 4)$ to avoid numerical issues. Empirically we found that within reasonable ranges these parameters had little effect on the final outcome. The initial probability of occupancy is set to 0.5, or $l_{ijk}^0 = 0$. In this case, the network acts on the log odd values l_{ijk}.

Density grid. In this model each voxel is assumed to have a continuous density, corresponding to the probability the voxel would block a sensor beam. We use the formulation from [34], where we track the Beta parameters α_{ijk}^t and β_{ijk}^t, with a uniform prior $\alpha_{ijk}^0 = \beta_{ijk}^0 = 1$ for all (i, j, k). The update for each voxel affected by the measurement z_t is

$$\alpha_{ijk}^t = \alpha_{ijk}^{t-1} + z^t$$
$$\beta_{ijk}^t = \beta_{ijk}^{t-1} + (1 - z^t)$$

and the posterior mean for the cell at (i, j, k) is

$$\mu_{ijk}^t = \frac{\alpha_{ijk}^t}{\alpha_{ijk}^t + \beta_{ijk}^t} \tag{2}$$

In this case we use μ_{ijk} as input to the network.

Hit grid. This model only consider hits, and ignores the difference between unknown and free space. Each voxel has an initial value $h_{ijk}^0 = 0$ and is updated as

$$h_{ijk}^t = \min(h_{ijk}^{t-1} + z^t, 1) \tag{3}$$

While this model discards some potentially valuable information, in our experiments it performs surprisingly well. Moreover, it does not require raytracing, which is useful in computationally constrained situations.

D. 3D Convolutional Network Layers

There are three main reasons CNNs are an attractive option for our task. First, they can explicitly make use of the spatial structure of our problem. In particular, they can learn local spatial filters useful to the classification task. In our case, we expect the filters at the input level to encode spatial structures such as planes and corners at different orientations. Second, by stacking multiple layers the network can construct a hierarchy of more complex features representing larger regions of space, eventually leading to a global label for the input occupancy grid. Finally, inference is purely feed-foward and can be performed efficiently with commodity graphics hardware.

In this paper, we consider CNNs consisting of the following types of layers, illustrated in Figure 1. Each layer type is denoted a shorthand description in the format $Name(hyperparameter)$.

Input Layer. This layer accepts a fixed-size grid of $I \times J \times K$ voxels. In this work, we use $I = J = K = 32$. Depending on the occupancy model, each value for each grid cell is updated from Equation 1, Equation 2 or Equation 3. In all three cases we subtract 0.5 and multiply by 2, so the input is in the $(-1, 1)$ range; no further preprocessing is done. While this work only considers scalar-valued inputs, our implementation can trivially accept additional values per cell, such as LiDAR intensity values or RGB information from cameras.

Convolutional Layers $C(f, d, s)$. These layers accept four-dimensional input volumes in which three of the dimensions are spatial, and the fourth contains the feature maps. The layer creates f feature maps by convolving the input with f learned filters of shape $d \times d \times d \times f'$, where d are the spatial dimensions and f' is the number of input feature maps. Convolution can also be applied at a spatial stride s. The output is passed through a leaky rectified nonlinearity unit (ReLU) [35] with parameter 0.1.

Pooling Layers $P(m)$. These layers downsample the input volume by a factor of by m along the spatial dimensions by replacing each $m \times m \times m$ non-overlapping block of voxels with their maximum.

Fully Connected Layer $FC(n)$. Fully connected layers have n output neurons. The output of each neuron is a learned linear combination of all the outputs from the previous layer, passed through a nonlinearity. We use ReLUs save for the final output layer, where the number of outputs corresponds to the number of class labels and a softmax nonlinearity is used to provide a probabilistic output.

E. Proposed architecture

Given these layers and their hyperparameters, there are countless possible architectures. To explore this space, in our previous work [28] we performed extensive stochastic search over hundreds of 3D CNN architectures on a simple classification task on simulated LiDAR data. Several of the best-performing networks had a small number of parameters in comparison to state of the art networks used for image data; [7] has around 60 million parameters, while the majority of our best models used less than 2 million.

While it is difficult to compare these numbers meaningfully, given the vast differences in tasks and datasets, we speculate that volumetric classification for point clouds is in some sense a simpler task, as many of the factors of variation in image data (perspective, illumination, viewpoint effects) are diminished or not present.

Guided by this precedent, our base model, *VoxNet*, is $C(32, 5, 2) - C(32, 3, 1) - P(2) - FC(128) - FC(K)$, where

K is number of classes (Figure 1). VoxNet is essentially a simpler version of the two-stage model reported in [28]. The changes aimed to reduce the number of parameters and increase computational efficiency, making the network easier and faster to learn. The model has 921736 parameters, most of them from inputs to the first dense layer.

F. Rotation Augmentation and Voting

As discussed in subsection III-B, it is nontrivial to maintain a consistent orientation of objects around their z axis. To counter this problem, many features for point clouds are designed to be rotationally invariant (e.g. [36], [37]). Our representation has no built-in invariance to large rotations; we propose a simple but effective approach to deal with this problem.

At *training* time, we augment the dataset with by creating n copies of each input instance, each rotated $360°/n$ intervals around the z axis. At *testing* time, we pool the activations of the output layer over all n copies. In this paper, n is 12 or 18. This can be seen as a voting approach, similar to how networks such as [7] average predictions over random crops and flips of the input image; however, it is performed over an exhaustive sampling of rotations, not a random selection.

This approach is inspired by the interpretation of convolution as weight sharing across translations; implicitly, we are sharing weights across rotations. Initial versions of this approach were implemented by max-pooling or mean-pooling the dense layers of the network during training in the same way as during test time. However, we found that the approach described above yielded comparable results while converging noticeably faster.

G. Multiresolution Input

Visual inspection of the LiDAR dataset suggested a $(0.2\,\mathrm{m}^3)$ resolution preserves all necessary information for the classification, while allowing sufficient spatial context for most larger objects such as trucks and trees. However, we hypothesized that a finer resolution would help in discriminating other classes such as traffic signs and traffic lights, especially for sparser data. Therefore, we implemented a multiresolution VoxNet, inspired by the "foveal" architecture of [24] for video analysis. In this model we use two networks with an identical VoxNet architectures, each receiving occupancy grids at different resolutions: $(0.1\,\mathrm{m})^3$ and $(0.2\,\mathrm{m})^3$. Both inputs are centered on the same location, but the coarser network covers a larger area at low resolution while the finer network covers a smaller area at high resolution. To fuse the information from both networks, we concatenate the outputs of their respective $FC(128)$ layers and connect them to a softmax output layer.

H. Network training details

Training of the network parameters is performed by Stochastic Gradient Descent (SGD) with momentum. The objective is the multinomial negative log-likelihood plus 0.001 times the L_2 weight norm for regularization. SGD is initialized with a learning rate of 0.01 for the LiDAR dataset and with 0.001 in the the other datasets. The momentum parameter was 0.9. Batch size is 32. The learning rate was decreased by a factor of 10 each 8000 batches for the LiDAR dataset and each 40000 batches in the other datasets.

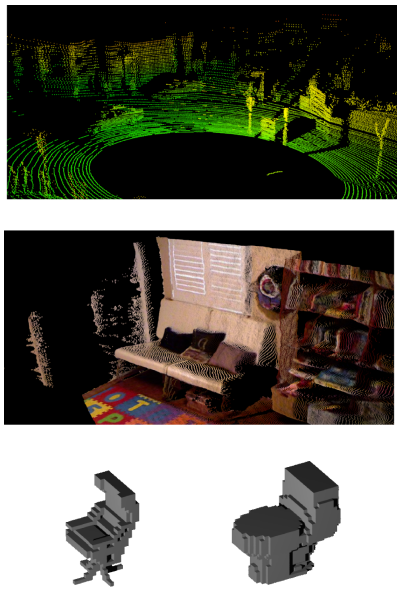

Fig. 2. From top to bottom, a point cloud from the Sydney Objects Dataset, a point cloud from NYUv2, and two voxelized models from ModelNet40.

Dropout regularization is added after the output of each layer. Convolutional layers were initialized with the method proposed by [38], whereas dense layers were initialized from a zero-mean Gaussian with $\sigma = 0.01$.

Following common practices for CNN training, we augment the data by adding randomly perturbed copies of each instance. The perturbed copies are generated dynamically during training and consist of randomly mirrored and shifted instances. Mirroring is done by along the x and y axes; shifting is done between -2 to 2 voxels along all axes.

Our implementation uses a combination of `C++` and Python. The Lasagne[1] library was used to compute gradients and accelerate computations on the GPU. The training process takes around 6 to 12 hours on our K40 GPU, depending on the complexity of the network.

IV. EXPERIMENTS

To evaluate VoxNet we consider benchmarks with data from three different domains: LiDAR point clouds, RGBD point clouds and CAD models. Figure 2 shows examples from each.

1) LiDAR data - Sydney Urban Objects: Our first set of experiments was conducted on the Sydney Urban Objects Dataset[2], which contains labeled Velodyne LiDAR scans of 631 urban objects in 26 categories. We chose this dataset for evaluation as it provides labeled object instances and the LiDAR viewpoint, which is used to compute occupancy. When voxelizing the point cloud we use all points in a bounding

[1] https://github.com/Lasagne/Lasagne

[2] http://www.acfr.usyd.edu.au/papers/SydneyUrbanObjectsDataset.shtml

box around the object, including background clutter. To make our results comparable to published work, we follow the protocol employed by the dataset authors. We report the average F_1 score, weighted by class support, for a subset of 14 classes over four standard training/testing splits. For this dataset we perform augmentation and voting with 18 rotations per instance.

2) CAD data - ModelNet: The ModelNet datasets were introduced by Wu et al. [26] to evaluate 3D shape classifiers. ModelNet40 has 151,128 3D models classified into 40 object categories, and ModelNet10 is a subset based on classes that are found frequently in the NYUv2 dataset [5]. The authors provide the 3D models as well as voxelized versions, which have been augmented by 12 rotations. We use the provided voxelizations and train/test splits for evaluation. In these voxelizations the objects have been scaled to fit a $30 \times 30 \times 30$ grid; therefore, we don't expect to benefit from a multiresolution approach, and we use the single-resolution VoxNet. For comparison of performance we report the accuracy averaged per class.

3) RGBD data - NYUv2: Wu et al also evaluate their approach on RGBD point clouds obtained from the NYUv2 dataset [5]. We use the train/test split provided by the authors, which uses 538 images from the RMRC challenge[3] for training, and the rest for testing. After selecting the boxes sharing a label with ModelNet10, we obtain 1386 testing boxes and 1422 training ground truth boxes. Wu et al report results on a subset of these boxes with high depth quality[4], whereas we report results using all the boxes, possibly making the task more difficult. We will make the split available to facilitate comparison.

For this dataset, we compute our own occupancy grids. However, to make results comparable to Wu et al we do not use a fixed voxel size; instead, we crop and scale the object bounding boxes to $24 \times 24 \times 24$, with 4 voxels of margin; likewise, we use 12 rotations instead of 18. As in the Sydney Objects dataset, we keep all points in a bounding box around the object; unlike Wu et al, we do not use a per-pixel object mask to remove outlying depth measurements from the voxelization.

A. Qualitative results

Learned filters. Figure 3 depicts cross sections of some learned filters from the input layer and corresponding feature maps learned from the input in the Sydney Objects dataset. The filters in this layer seem to encode primitives such as edges, corners, and "blobs". Figure 4 shows filters learned in the NYUv2 and ModelNet40 datasets. The filters are similar across datasets, similar to what occurs for image data.

Rotational invariance. A natural question is whether the network learns some degree of rotational invariance. Figure 5 is an example supporting this hypothesis, where the two fully connected layers show a highly (but not completely) invariant response across 12 rotations of the input.

[3]http://ttic.uchicago.edu/~rurtasun/rmrc/

[4]Personal communication.

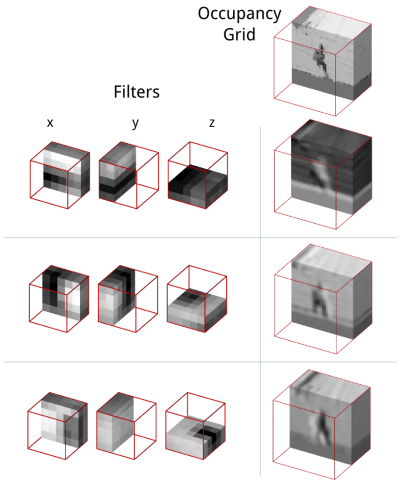

Fig. 3. Cross sections of three $5 \times 5 \times 5$ filters from the first layer of VoxNet in the Sydney Objects Database, with corresponding feature map on the right.

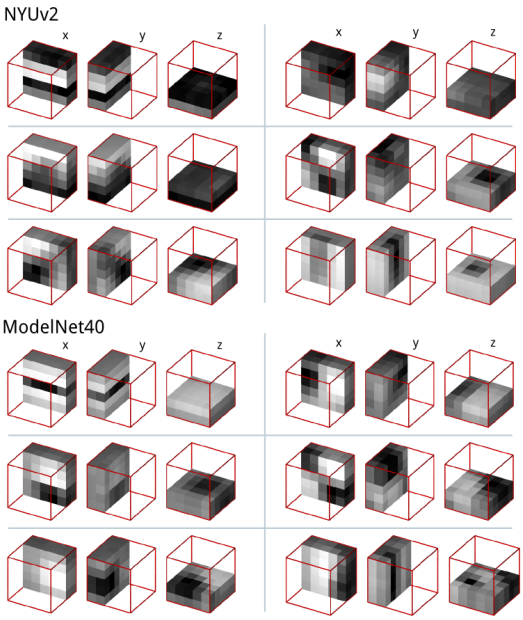

Fig. 4. Cross sections along the x, y and z axes of selected first layer filters learned in the ModelNet40 and NYUv2 datasets.

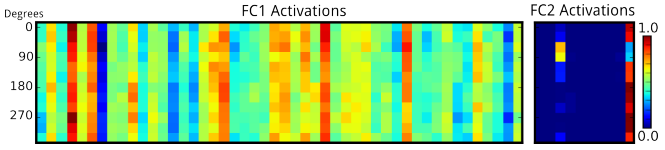

Fig. 5. Neuron activations for the two fully connected layers of VoxNet when using the point cloud from Fig. 1 (right) as input in 12 different orientations. For the first fully connected layer only 48 features are shown. Each row corresponds to a rotation around z and each column corresponds to a neuron. The activations show a approximate rotational invariance. The neurons in the right correspond to output classes. The last column, for *toilet*, is the correct response. Near 90°, the object becomes confused with a *chair* (third column); by voting across all orientations we obtain the correct answer.

TABLE I

EFFECTS OF ROTATION AUGMENTATION AND VOTING

Training Augm.	Test Voting	Sydney F1	ModelNet40 Acc
Yes	Yes	**0.72**	**0.83**
Yes	No	0.71	0.82
No	Yes	0.69	0.69
No	No	0.69	0.61

TABLE II

EFFECT OF OCCUPANCY GRIDS

Occupancy	Sydney F1	NYUv2 Acc
Density grid	**0.72**	**0.71**
Binary grid	0.71	0.69
Hit grid	0.70	0.70

B. VoxNet variations

Rotation Augmentation. We study four different cases for Rotation Augmentation, depending on whether it is applied or not at train time (as augmentation) and test time (as voting) for the Sydney Objects and ModelNet40 datasets. For the cases in which no voting is performed at test time, a random orientation is applied on the test instances, and the average over four runs is reported. For the cases in which no training time augmentation is performed, there are two cases. In ModelNet40, we select the object in a canonical pose as the training instance. For Sydney Objects, this information is not available, and we use the unmodified orientation from the data. Table I shows the results. They indicate that training time augmentation is more important. As suggested by the qualitative example above, the network learns some degree of rotational invariance, even if not explicitly enforced. However, voting at training time voting still gives a small boost. For ModelNet40, we see a large degradation of performance when we train on canonical poses but test on an arbitrary poses, as expected. For Sydney Objects there is no such mismatch, and there is no clear effect. Since rotation augmentation seems consistently beneficial, in the rest of the results section we use VoxNet with rotation augmentation at both test time and run time.

Occupancy grids. We also study the effect of the Occupancy Grid representation in Table II. We found VoxNet to be quite robust to the different representations. Against expectations, we found the Hit grid to perform comparably or better than the other approaches, though the differences are small. This is possibly because any advantage provided by differentiating between free space and unknown space is negated by the extra viewpoint-induced variability of Density and Binary grids relative to Hit grids. By default, we will use Density grids in the experiments below.

Resolution. For the Sydney Object Dataset we evaluated VoxNet with voxels of size $0.1\,\mathrm{m}$ and $0.2\,\mathrm{m}$. We found them to perform almost indistinguishably, with an F_1 score of 0.72. On the other hand, fusing both with the multiresolution approach described in subsection III-G slightly outperformed both with a score of 0.73.

1) Comparison to other approaches: Here we compare VoxNet against publicly available results in the literature.

TABLE III

COMPARISON WITH OTHER METHODS IN SYDNEY OBJECT DATASET

Method	Avg F1
UFL+SVM[21]	0.67
GFH+SVM[37]	0.71
Multi Resolution VoxNet	**0.73**

TABLE IV

COMPARISONS WITH SHAPENET IN MODELNET (AVG ACC)

Dataset	ShapeNet	VoxNet
ModelNet10	0.84	**0.92**
ModelNet40	0.77	**0.83**

TABLE V

COMPARISON WITH SHAPENET IN NYUV2 (AVG ACC)

Dataset	ShapeNet	VoxNet	VoxNet Hit
NYU	0.58	**0.71**	0.70
ModelNet10→NYU	**0.44**	0.34	0.25

Table III shows our best VoxNet against the best approach from [21], which combines an unsupervised form of Deep Learning with SVM classifiers, and [37], which designs a rotationally invariant descriptor and classifies it with a nonlinear SVM. We show a small increase in accuracy relative to these approaches. Moreover, we expect our approach to be much faster than approaches based on nonlinear SVMs, as these do not scale well to large datasets.

Finally, we compare against the Shapenet architecture proposed by Wu et al [26] in the task of classification for ModelNet10, ModelNet40, and in the NYUv2 datasets, as well as in the task of classifying the NYUv2 dataset with a model trained on ModelNet10. Shapenet is also a volumetric convolutional architecture. It is trained generatively with discriminative fine tuning, and also employs rotation augmentation for training. ShapeNet is a relatively large architecture, with over 12.4 million parameters, while VoxNet has less than 1 million. We do not use pretraining for NYUv2, but instead train from scratch. Table IV shows results in the ModelNet datasets and Table V shows results with density grids (VoxNet) and hit grids (VoxNet Hit) for the two tasks involving the NYU dataset.

Despite the fact we use a more adverse testing set, VoxNet outperforms in ShapeNet in all tasks except the cross-domain task (second row). We are unsure what causes the difference. Both models perform rotation augmentation at training time; VoxNet also votes over rotations at test time, but this only accounts for 1-2% improvement. The simpler architecture of VoxNet may result in better generalization when using purely discriminative training. On the other hand, the worse performance in the cross-domain task may be because the discriminative training is less capable of dealing with the domain shift, or because we did not use masks to select points.

C. Timing

We use a Tesla K40 GPU in our experiments. Our slowest configuration, the multiresolution VoxNet with rotational

voting, takes around $6\,\mathrm{ms}$ when classified individually, and around $1\,\mathrm{ms}$ when averaged over a batch of size 32. Depending on the number of points, raytracing may also be a bottleneck; our implementation takes around two milliseconds for around 2000 points (typical for LiDAR) but up to half a second for $200k$ points, as may happen with RGBD. For this situation, one can use Hit Grids, or use one of several raytracing optimization strategies in the literature.

V. CONCLUSIONS

In this work, we presented VoxNet, a 3D CNN architecture for for efficient and accurate object detection from LiDAR and RGBD point clouds, and studied the effect of various design choices on its performance. Our best system outperforms the state of the art in various benchmarks while performing classification in real time.

In the future we are interested in the integration of data from other modalities (e.g., cameras), and the application of this method to other tasks such as semantic segmentation.

ACKNOWLEDGMENTS

This research was sponsored under a fellowship by United Technologies Research Center. The Tesla K40 used for this research was donated by the NVIDIA Corporation. We thank the reviewers for their feedback.

REFERENCES

[1] C. Urmson, J. Anhalt, H. Bae, J. A. D. Bagnell, C. R. Baker, R. E. Bittner, T. Brown, M. N. Clark, M. Darms, D. Demitrish, J. M. Dolan, D. Duggins, D. Ferguson , T. Galatali, C. M. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, S. Kolski, M. Likhachev , B. Litkouhi, A. Kelly , M. McNaughton, N. Miller, J. Nickolaou, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, V. Sadekar, B. Salesky, Y.-W. Seo, S. Singh, J. M. Snider, J. C. Struble, A. T. Stentz, M. Taylor , W. R. L. Whittaker, Z. Wolkowicki, W. Zhang, and J. Ziglar, "Autonomous driving in urban environments: Boss and the urban challenge," *JFR*, vol. 25, no. 8, pp. 425–466, June 2008.

[2] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *ISRR*, Flagstaff, Arizona, USA, Aug. 2011.

[3] S. Choudhury, S. Arora, and S. Scherer, "The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety," in *AHS*, May 2014.

[4] A. Quadros, J. Underwood, and B. Douillard, "An occlusion-aware feature for range images," in *ICRA*, May 14-18 2012.

[5] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[8] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: an astounding baseline for recognition," *CoRR*, vol. abs/1403.6382, 2014.

[9] G. Hinton, "Does the brain do inverse graphics?" Brain and Cognitive Sciences Fall Colloquium.

[10] A. Frome, D. Huber, and R. Kolluri, "Recognizing objects in range data using regional point descriptors," *ECCV*, vol. 1, pp. 1–14, 2004.

[11] J. Behley, V. Steinhage, and A. B. Cremers, "Performance of histogram descriptors for the classification of 3D laser range data in urban environments," in *ICRA*, 2012, pp. 4391–4398.

[12] A. Teichman, J. Levinson, and S. Thrun, "Towards 3D object recognition via classification of arbitrary object tracks," in *ICRA*, 2011, pp. 4034–4041.

[13] A. Golovinskiy, V. G. Kim, and T. Funkhouser, "Shape-based recognition of 3D point clouds in urban environments," *ICCV*, 2009.

[14] D. Munoz, N. Vandapel, and M. Hebert, "Onboard contextual classification of 3-D point clouds with learned high-order markov random fields," in *ICRA*, 2009.

[15] H. Koppula, "Semantic labeling of 3D point clouds for indoor scenes," *NIPS*, 2011.

[16] X. Ren, L. Bo, and D. Fox, "RGB-(D) scene labeling: Features and algorithms," in *CVPR*, 2012.

[17] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," in *RSS*, 2013.

[18] Richard Socher and Brody Huval and Bharath Bhat and Christopher D. Manning and Andrew Y. Ng, "Convolutional-Recursive Deep Learning for 3D Object Classification," in *NIPS*, 2012.

[19] L. A. Alexandre, "3D object recognition using convolutional neural networks with transfer learning between input channels," in *IAS*, vol. 301, 2014.

[20] N. Höft, H. Schulz, and S. Behnke, "Fast semantic segmentation of RGBD scenes with gpu-accelerated deep neural networks," in *37th Annual German Conference on AI*, 2014, pp. 80–85.

[21] M. De Deuge, A. Quadros, C. Hung, and B. Douillard, "Unsupervised feature learning for classification of outdoor 3d scans," in *ACRA*, 2013.

[22] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from RGB-D images for object detection and segmentation," in *ECCV*, 2014.

[23] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE TPAMI*, vol. 35, no. 1, pp. 221–231, 2013.

[24] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *CVPR*, 2014.

[25] K. Lai, L. Bo, and D. Fox, "Unsupervised feature learning for 3D scene labeling," in *ICRA*, 2014.

[26] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shape modeling," in *CVPR*, 2015.

[27] D. Prokhorov, "A convolutional learning system for object classification in 3-D lidar data," *IEEE TNN*, vol. 21, no. 5, pp. 858–863, May 2010.

[28] D. Maturana and S. Scherer, "3D convolutional neural networks for landing zone detection from lidar," in *ICRA*, 2015.

[29] B. Douillard, J. Underwood, V. Vlaskine, A. Quadros, and S. Singh, "A pipeline for the segmentation and classification of 3D point clouds," in *ISER*, 2010.

[30] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *ICRA*, 1985.

[31] S. Thrun, "Learning occupancy grid maps with forward sensor models," *Auton. Robots*, vol. 15, no. 2, pp. 111–127, 2003.

[32] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *Eurographics '87*, Aug. 1987, pp. 3–10.

[33] D. Hähnel, D. Schulz, and W. Burgard, "Map building with mobile robots in populated environments," in *IROS*, 2002.

[34] G. D. Tipaldi and K. O. Arras, "FLIRT - interest regions for 2D range data," in *ICRA*, 2010.

[35] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*, vol. 30, 2013.

[36] A. Johnson, "Spin-images: A representation for 3-D surface matching," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1997.

[37] T. Chen, B. Dai, D. Liu, and J. Song, "Performance of global descriptors for velodyne-based urban object recognition," in *IV*, June 2014, pp. 667–673.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.