arXiv:1604.03351v2 [cs.CV] 19 Oct 2017

Orientation-boosted Voxel Nets for 3D Object Recognition

Nima Sedaghat
nima@cs.uni-freiburg.de

Mohammadreza Zolfaghari
zolfagha@cs.uni-freiburg.de

Ehsan Amiri
amirie@cs.uni-freiburg.de

Thomas Brox
brox@cs.uni-freiburg.de

Computer Vision Group
University of Freiburg
Germany

Abstract

Recent work has shown good recognition results in 3D object recognition using 3D convolutional networks. In this paper, we show that the object orientation plays an important role in 3D recognition. More specifically, we argue that objects induce different features in the network under rotation. Thus, we approach the category-level classification task as a multi-task problem, in which the network is trained to predict the pose of the object in addition to the class label as a parallel task. We show that this yields significant improvements in the classification results. We test our suggested architecture on several datasets representing various 3D data sources: LiDAR data, CAD models, and RGB-D images. We report state-of-the-art results on classification as well as significant improvements in precision and speed over the baseline on 3D detection.

1 Introduction

Various devices producing 3D point clouds have become widely applicable in recent years, e.g., range sensors in cars and robots or depth cameras like the Kinect. Structure from motion and SLAM approaches have become quite mature and generate reasonable point clouds, too. With the rising popularity of deep learning, features for recognition are no longer designed manually but learned by the network. Thus, moving from 2D to 3D recognition requires only small conceptual changes in the network architecture [21, 38].

In this work, we elaborate on 3D recognition using 3D convolutional networks, where we focus on the aspect of auxiliary task learning. Usually, a deep network is directly trained on the task of interest, i.e., if we care about the class labels, the network is trained to produce correct class labels. There is nothing wrong with this approach. However, it requires the network to learn the underlying concepts, such as object pose, that generalize to variations in the data. Oftentimes, the network does not learn the full underlying concept but some representation that only partially generalizes to new data.
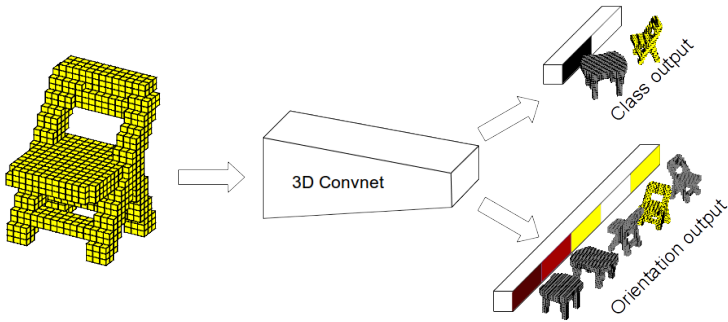
Figure 1: Adding orientation classification as an auxiliary task to a 3D classification network improves its category-level classification accuracy.

In the present paper, we focus on the concept of object orientation. The actual task only cares about the object label, not its orientation. However, to produce the correct class label, at least some part of the network representation must be invariant to the orientation of the object, which is not trivial in 3D. Effectively, to be successful on the classification task, the network must also solve the orientation estimation task, but the loss function does not give any direct indication that solving this auxiliary task is important. We show that forcing the network to produce the correct orientation during training increases its classification accuracy significantly – Figure 1.

We introduce a network architecture that implements this idea and evaluate it on 4 different datasets representing the large variety of acquisition methods for point clouds: laser range scanners, RGB-D images, and CAD models. The input to the network is an object candidate obtained from any of these data sources, which is fed to the network as an occupancy grid. We compare the baseline without orientation information to our orientation-boosted version and obtain improved results in all experiments. We also compare to the existing 3D classification methods and achieve state-of-the-art results using our *shalow* orientation-boosted networks in most of the experiments. In the scope of our experiments, we extended the Modelnet40 dataset, which consists of more than 12k objects, with per-class alignments by using some automated alignment procedure [28]. We will provide the additional annotation.

We also applied the classifier in a 3D detection scenario using a simple 3D sliding box approach. In this context, the orientation estimation is no longer just an auxiliary task but also determines the orientation of the box, which largely reduces the runtime of the 3D detector.

2   Related Work

Most previous works on 3D object recognition rely on handcrafted feature descriptors, such as Point Feature Histograms [26, 27], 3D Shape Context [19], or Spin Images [18]. Descriptors based on surface normals have been very popular, too [14, 23]. Yulanguo *et al.* [40] gives an extensive survey on such descriptors.

Feature learning for 3D recognition has first appeared in the context of RGB-D images, where depth is treated as an additional input channel [1, 6, 8]. Thus, the approaches are

conceptually very similar to feature learning in images. Gupta *et al.* [12] fits and projects 3D synthetic models into the image plane.

3D convolutional neural networks (CNNs) have appeared in the context of videos. Tran *et al.* [35] use video frame stacks as a 3D signal to approach multiple video classification tasks using their 3D CNN, called C3D. 3D CNNs are not limited to videos, but can be applied also to other three-dimensional inputs, such as point clouds, as in our work.

The most closely related works are by Wu *et al.* [38] and Maturana & Sherer [21], namely 3D ShapeNets and VoxNet. Wu *et al.* use a Deep Belief Network to represent geometric 3D shapes as a probability distribution of binary variables on a 3D voxel grid. They use their method for shape completion from depth maps, too. The ModelNet dataset was introduced along with their work. The VoxNet [21] is composed of a simple but effective CNN, accepting as input voxel grids similar to Wu *et al.* [38]. In both of these works the training data is augmented by rotating the object to make the network learn a feature representation that is invariant to rotation. However, in contrast to the networks proposed in this paper, the network is not enforced to output the object orientation, but only its class label. While in principle, the loss on the class label alone should be sufficient motivation for the network to learn an invariant representation, our experiments show that an explicit loss on the orientation helps the network to learn such representation.

Su et al. [34] take advantage of the object pose explicitly by rendering the 3D objects from multiple viewpoints and using the projected images in a combined architecture of 2D CNNs to extract features. However, this method still relies on the appearance of the objects in images, which only works well for dense surfaces that can be rendered. For sparse and potentially incomplete point clouds, the approach is not applicable. Song et al. [32] focus on 3D object detection in RGB-D scenes. They utilize a 3D CNN for 3D object bounding box suggestion. For the recognition part, they combine geometric features in 3D and color features in 2D.

Several 3D datasets have become available recently. Sydney Urban Objects [7] includes point-clouds obtained from range-scanners. SUN-RGBD [33] and SUN-3D [39] gather some reconstructed point-clouds and RGB-D datasets in one place and in some cases they also add extra annotations to the original datasets. We use the annotations provided for NYU-Depth V2 dataset [30] by SUN-RGBD in this work. ModelNet is a dataset consisting of synthetic 3D object models [38]. Sedaghat & Brox [28] created a dataset of annotated 3D point-clouds of cars from monocular videos using structure from motion and some assumptions about the scene structure.

Quite recently and parallel to this work, there have been several works utilizing 2D or 3D CNNs merely on 3D CAD models of ModelNet [38] in supervised [2, 9, 13], semi-supervised [24] and unsupervised [37] fashions. Although our main architecture is rather shallow compared to most of them, and we use a rather low resolution compared to methods relying on high-resolution input images, we still provide state-of-the-art results on the aligned ModelNet10 subset, and on-par results on a roughly and automatically aligned version of the ModelNet40 subset.

Most of the published works on detection try to detect objects directly in the 2D space of the image. Wang & Posner [36] were among the first ones to utilize point clouds to obtain object proposals. In another line of work Gonzalez *et al.* [11] and Chen *et al.* [5] mix both 2D and 3D data for detection. Many authors, including Li *et al.* [20] and Huang *et al.* [15] approach the task with a multi-tasking method.
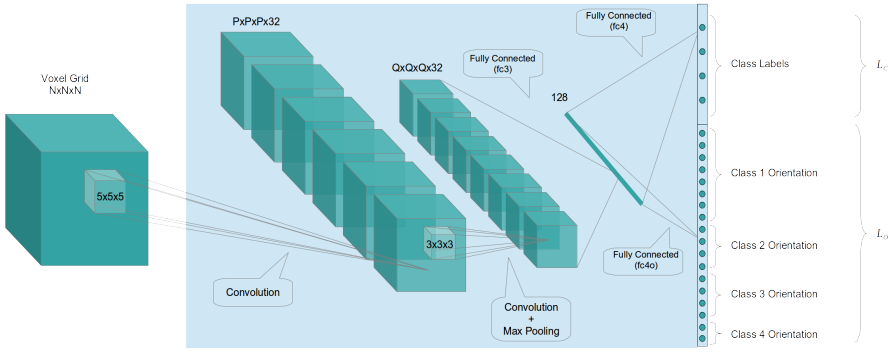
Figure 2: Basic Orientation Boosting. Class labels and orientation labels are two separate outputs. The number of orientation labels assigned to each class can be different to the others. Both outputs contribute to the training equally – with the same weight.

3   Method

The core network architecture is based on VoxNet [21] and is illustrated in Fig. 2. It takes a 3D voxel grid as input and contains two convolutional layers with 3D filters followed by two fully connected layers. Although this choice may not be optimal, we keep it to be able to directly compare our modifications to VoxNet. In addition, we experimented with a slightly deeper network that has four convolutional layers.

Point clouds and CAD models are converted to voxel grids (occupancy grids). For the NYUv2 dataset we used the provided tools for the conversion; for the other datasets we implemented our own version. We tried both binary-valued and continuous-valued occupancy grids. In the end, the difference in the results was negligible and thus we only report the results of the former one.

Multi-task learning   We modify the baseline architecture by adding orientation estimation as an auxiliary parallel task. We call the resulting architecture the ORIentation-boosted vOxel Net – ORION. Without loss of generality, we only consider rotation around the z-axis (azimuth) as the most varying component of orientation in practical applications. Throughout this paper we use the term 'orientation' to refer to this component.

Orientation is a continuous variable and the network could be trained to provide such an output. However, the idea is to treat different orientations of an object differently, and therefore we cast the orientation estimation as a classification problem. This also serves as a relaxation on dataset constraints, as a rough alignment of data obviates the need for strict orientation annotations. The network has output nodes for the product label space of classes and orientations and learns the mapping

$$x_i \mapsto (c_i, o_i) \tag{1}$$

where x_i are the input instances and c_i, o_i are their object class and *orientation class*, respectively.

We do not put the same orientations from different object classes into the same orientation class, because we do not seek to extract any information from the absolute pose of the objects. Sharing the orientation output for all classes would make the network learn features

shared among classes to determine the orientation, which is the opposite of what we want: leveraging the orientation estimation as an auxiliary task to improve on object classification. For example, a table from the 45° orientations class is not expected to share any useful information with a car of the same orientation.

We choose multinomial cross-entropy losses [25] for both tasks, so we can combine them by summing them up:

$$\mathcal{L} = (1-\gamma)\mathcal{L}_C + \gamma\mathcal{L}_O \qquad (2)$$

where $\mathcal{L}_C$ and $\mathcal{L}_O$ indicate losses for object classification and orientation estimation tasks respectively. We used equal loss weights ($\gamma = 0.5$) and found in our classification experiments that the results do not depend on the exact choice of the weight γ around this value. However, in one of the detection experiments, where the orientation estimation is not an auxiliary task anymore, we used a higher weight for the orientation output to improve its accuracy.

Due to various object symmetries, the number of orientation labels differs per object class – Figure 2. The idea is that we do not want the network to try to differentiate between, e.g., a table and its 180° rotated counterpart. For the same reason, to rotationally symmetric objects, such as poles, or rotationally *neutral* ones, such as trees to which no meaningful azimuth label can be assigned, we dedicate only a single node. This is decided upon manually in the smaller datasets. However, during the auto-alignment of the bigger Modelnet40 dataset, the number of orientations are also automatically assigned to different classes. Details are given in the supplementary material.

Voting   Object orientations can be leveraged at the entry of the network, too. During the test phase we feed multiple rotations of the test object to the network and obtain a final consensus on the class label based on the votes we obtain from each inference pass, as follows:

$$c_{final} = \arg\max_k \sum_r S_k(x_r) \qquad (3)$$

where S_k is the score the network assigns to the object at its kth node of the main (object category) output layer. x_r is the test input with the rotation index r.

4   Datasets

We train and test our networks on four datasets, three of which are illustrated in Figure 3. We have chosen the datasets such that they represent different data sources.

Sydney Urban Objects - LiDAR/Pointcloud   This dataset consists of LiDAR scans of 631 objects in 26 categories. The objects' point-clouds in this dataset are always incomplete, as they are only seen by the LiDAR sensor from a single viewpoint. Therefore the quality of the objects are by no means comparable to synthetic 3D objects, making classification a challenging task, even for human eyes; see Figure 3. This dataset is also of special interest in our category-level classification, as it provides a tough categorization of vehicles: *4wd*, *bus*, *car*, *truck*, *ute* and *van* are all distinct categories. We use the same settings as [21] to make our results comparable to theirs. The point clouds are converted to voxel-grids of size 32x32x32, in which the object occupies a 28x28x28 space. Zero-paddings of size 2 is used on each side to enable displacement augmentation during training. We also annotated the orientations to make the data suitable for our method. These we will provide to the public.
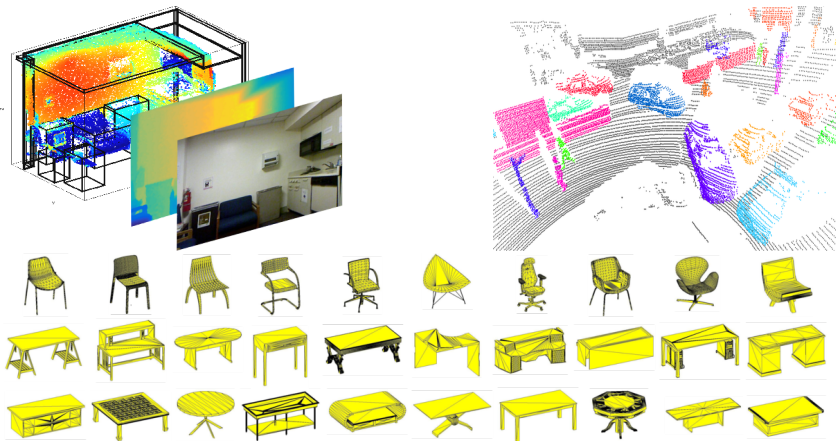
Figure 3: Examples from the various 3D datasets we used in experiments. On top, two exemplar scenes from the NYUv2 [30] & Sydney [7] datasets are depicted. On the bottom samples from the Modelnet dataset are displayed. The KITTI dataset is similar to the Sydney dataset and is not shown here.

NYUv2 - Kinect/RGBD   This dataset consists of an overall number of 2808 RGBD images, corresponding to 10 object classes. The class types are shared with the ModelNet10 dataset. We used voxel grids of size 32x32x32, which contain the main object in the size of 28x28x28. The rest includes the context of the object and each object has a maximum number of 12 rotations. The dataset does not provide orientation annotations and therefore we used the annotations provided by the SUN-RGBD benchmark [33].

ModelNet - Synthetic/CAD   This dataset is composed of synthetic CAD models. The ModelNet10 subset consists of uniformly aligned objects of the same classes as in the NYUv2 dataset. The object meshes in this dataset are converted to voxel grids of size 28x28x28, similar to the NYUv2 setting. The ModelNet40 subset does not come with alignments (or orientation annotations). Thus, we provided manual annotation of orientation that we will make publicly available. In addition, we ran an unsupervised automated procedure to align the samples of ModelNet40. Please refer to supplemental material for details.

KITTI - LiDAR/Pointcloud   The KITTI dataset [10] contains 7481 training images and 7518 test images in its object detection task. Each image represents a scene which also comes with a corresponding Velodyne point cloud. 2D and 3D bounding box annotations are provided in the images. Using the provided camera calibration parameters they can be converted into the coordinates of the Velodyne scanner. We use this dataset only in the detection experiment. To be able to report and analyze the effects of our method at multiple levels, we split the publicly available training set to 80% and 20% subsets for training and testing, respectively.

Method↓				Dataset		
		# Conv	# param	Sydney	NYUv2	ModelNet10
Hand-crafted feat.	Recursive D [31]	-	-	-	37.6	-
	Recursive D+C [31]	-	-	-	44.8	-
	Triangle+SVM [7]	-	-	67.1	-	-
	GFH+SVM [4]	-	-	71.0	-	-
Deep Network	FusionNet [13]		118M	-	-	93.1
	VRN† [2]	43	18M	-	-	93.6
Shallow Network	ShapeNet [33]	3	-	-	57.9	83.5
	DeepPano [29]	4	-	-	-	85.5
	VoxNet [21] (baseline)	2	890K	72	71	92
	ORION (Ours)	2	910K	**77.8**	**75.4**	**93.8**
		4	4M	77.5	**75.5**	**93.9**

Table 1: Classification results and comparison to the state-of-the-art on three datasets. We report the overall classification accuracy, except for the Sydney dataset where we report the weighted average over F1 score. The auxiliary task on orientation estimation clearly improves the classification accuracy on all datasets.† We report the single-network results for this method.

5 Experiments and Results

5.1 Classification

The classification results on all datasets are shown in Table 1. For the Sydney Urban Objects dataset, we report the average F1 score weighted by class support as in [33] to be able to compare to their work. This weighted average takes into account that the classes in this dataset are unbalanced. For the other datasets we report the average accuracy. The Sydney dataset provides 4 folds/subsets to be used for cross-validation; in each experiment three folds are for training and one for testing. Also due to the small size of this dataset, we run each experiment three times with different random seeds, and report the average over all the 12 results.

We achieve clear improvements over the baseline and report state-of-the-art results in all the three datasets, with a far shallower architecture compared to previous state-of-the-art (2 vs. 43 conv. layers) and a big saving on number of parameters (1M vs.18M).

We also experimented with a slightly deeper network (last row of Table 1), but found that the network starts to overfit on the smaller datasets. Details of this extended architecture can be found in the supplementary material.

5.1.1 Non-aligned Dataset

Since the Modelnet40 dataset does not come with alignments we manually annotated the orientation. As an alternative, we also aligned the objects, class by class, in an unsupervised fashion using the method introduced in Sedaghat & Brox [28]. Details of the steps of this process can be found in the supplementary material. Table 2 shows the large improvement obtained by using the extra annotation during training. Interestingly, the automatic alignment
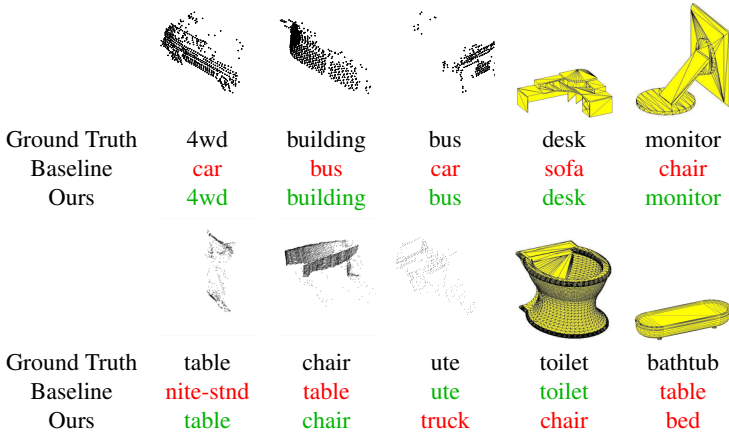
Ground Truth	4wd	building	bus	desk	monitor
Baseline	car	bus	car	sofa	chair
Ours	4wd	building	bus	desk	monitor

Ground Truth	table	chair	ute	toilet	bathtub
Baseline	nite-stnd	table	ute	toilet	table
Ours	table	chair	truck	chair	bed

Figure 4: Some exemplar classification results. We show examples on which the outputs of the two networks differ.

Method	Conv. Layers	Batch Norm.	Accuracy (%)		
			No Alignment	Rough, Automatic Alignment	Perfect, Manual Alignment
VoxNet [21] (baseline)	2	×	83	-	-
ORION (Ours)	2	×	-	88.1	87.5
	2	✓	-	88.6	88.2
	4	✓	-	89.4	**89.7**

Table 2: Classification accuracy on Modelnet40. Orientation information during training clearly helps boost the classification accuracy even when orientation labels are obtained by unsupervised alignment [28]. In fact, manually assigned labels do not yield any significant improvement. Batch normalization and two additional convolutional layers improve results.

is almost as good as the tedious manual orientation labeling. This shows that the network can benefit even from coarse annotations.

Since the number of training samples is large, the deeper network with four convolutional layers performed even better than its counterpart with only two convolutional layers.

Batch normalization (BN) is known to help with problems during network training [16]. Adding batch normalization to the convolutional layers yielded consistent improvements in the results; e.g. see Table 2. We conjecture that batch normalization causes the error from the second task to propagate deeper back into the network.

5.2 Detection

We tested the performance of our suggested method in a detection scenario, where the orientation sensitive network is used as a binary object classifier to assign a score to 3D bounding box proposals in a sliding window fashion. We tested the 3D detector to detect cars in the KITTI dataset.

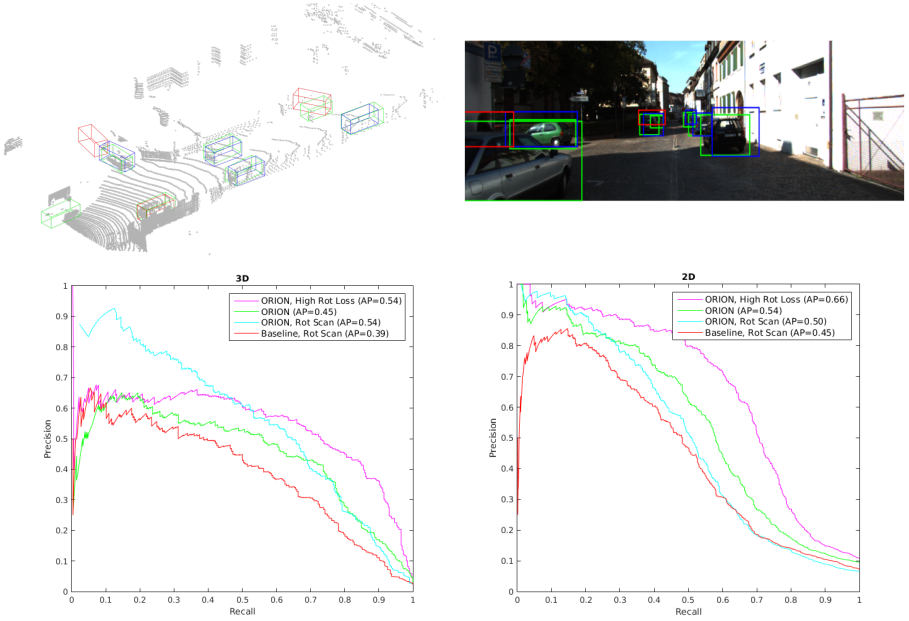Figure 5 quantifies the improvements of the ORION architecture in such an exemplar

Figure 5: On the top left, detected boxes of an exemplar scene are displayed in its 3D point-cloud representation. 3D boxes are then projected to the 2D image plane – top right. Green boxes are the ground truth cars. Blue and red show true and false positives respectively. The bottom row illustrates the Precision-Recall curves for multiple detection experiments.

detection scenario. The mere use of our architecture as a binary classifier significantly pulls up the PR curve and increases the mean average precision. In this case, we only relied on the object classification output of the network and performed an exhaustive search over the rotations – 18 rotation steps to cover 360 degrees. The main benefit is achieved when we also leveraged the orientation output of the network to directly predict the orientation of the object. This resulted in an $18\times$ run time improvement. We also noticed that by increasing the loss weight of the orientation output, thus emphasizing on the orientation, the detection results improved further.

It is worth noting that in contrast to most of the detectors which run detection in the RGB image of the KITTI dataset, we do not use the RGB image but only use the 3D point cloud. We also limited the search in the scale and aspect-ratio spaces by obtaining statistical measures of the car sizes in the training set.

6 Analysis

To analyze the behavior of the orientation-boosted network, we compare it to its corresponding baseline network. To find a correspondence, we first train the baseline network long enough, so that it reaches a stable state. Then we use the trained net to initialize the weights of ORION, and continue training with low learning rate. We found that some filters tend to become more sensitive to orientation-specific features of the objects. We also found that in the baseline network some filters behave as the dominant ones for all the possible rotations of

the objects in a class, while ORION has managed to spread the contributions over different filters for different orientations. Details of these experiments and visualizations are given in the supplementary material.

7 Conclusions

We showed for the task of 3D object classification that learning of certain concepts, such as invariance to object orientation, can be supported by adding the concept as an auxiliary task during training. By forcing the network to produce also the object orientation during training, it achieved better classification results at test time. This finding was consistent on all datasets and enabled us to establish state-of-the-art results on most of them. The approach is also applicable to 3D detection in a simple sliding 3D box fashion. In this case, the orientation output of the network avoids the exhaustive search over the object rotation.

Acknowledgements

References

[1] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Unsupervised feature learning for RGB-D based object recognition. In *Experimental Robotics*, pages 387–402. Springer, 2013.

[2] Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. Generative and Discriminative Voxel Modeling with Convolutional Neural Networks. *arXiv:1608.04236 [cs, stat]*, August 2016.

[3] Fatih Calakli and Gabriel Taubin. Ssd: Smooth signed distance surface reconstruction. In *Computer Graphics Forum*, volume 30, pages 1993–2002. Wiley Online Library, 2011.

[4] Tongtong Chen, Bin Dai, Daxue Liu, and Jinze Song. Performance of global descriptors for velodyne-based urban object recognition. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 667–673. IEEE, 2014.

[5] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-View 3D Object Detection Network for Autonomous Driving. *arXiv:1611.07759 [cs]*, November 2016.

[6] Camille Couprie, Clément Farabet, Laurent Najman, and Yann LeCun. Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*, 2013.

[7] Mark De Deuge, Field Robotics, Alastair Quadros, Calvin Hung, and Bertrand Douillard. Unsupervised Feature Learning for Classification of Outdoor 3D Scans. 2013.

[8] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929, 2013.

[9] A. Garcia-Garcia, F. Gomez-Donoso, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla, and J. Azorin-Lopez. PointNet: A 3D Convolutional Neural Network for real-time object class recognition. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1584, July 2016. doi: 10.1109/IJCNN.2016.7727386.

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[11] A. González, G. Villalonga, J. Xu, D. Vázquez, J. Amores, and A. M. López. Multiview random forest of local experts combining RGB and LIDAR data for pedestrian detection. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 356–361. doi: 10.1109/IVS.2015.7225711.

[12] Saurabh Gupta, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Aligning 3D Models to RGB-D Images of Cluttered Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4731–4740, 2015.

[13] Vishakh Hegde and Reza Zadeh. FusionNet: 3D Object Classification Using Multiple Data Representations. *arXiv:1607.05695 [cs]*, July 2016.

[14] Berthold Klaus Paul Horn. Extended gaussian images. *Proceedings of the IEEE*, 72 (12):1671–1686, 1984.

[15] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. DenseBox: Unifying Landmark Localization with End to End Object Detection. *arXiv:1509.04874 [cs]*, September 2015.

[16] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456. JMLR Workshop and Conference Proceedings, 2015.

[17] Alec Jacobson et al. gptoolbox: Geometry processing toolbox, 2016. http://github.com/alecjacobson/gptoolbox.

[18] Andrew E Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(5):433–449, 1999.

[19] Marcel Körtgen, Gil-Joo Park, Marcin Novotni, and Reinhard Klein. 3D shape matching with 3D shape contexts. In *The 7th central European seminar on computer graphics*, volume 3, pages 5–17, 2003.

[20] Chi Li, M. Zeeshan Zia, Quoc-Huy Tran, Xiang Yu, Gregory D. Hager, and Manmohan Chandraker. Deep Supervision with Shape Concepts for Occlusion-Aware 3D Object Parsing. *arXiv:1612.02699 [cs]*, December 2016.

[21] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[22] Gavin Miller. Efficient Algorithms for Local and Global Accessibility Shading. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 319–326, New York, NY, USA, 1994. ACM. ISBN 978-0-89791-667-7. doi: 10.1145/192161.192244. URL http://doi.acm.org/10.1145/192161.192244.

[23] Alexander Patterson IV, Philippos Mordohai, and Kostas Daniilidis. Object detection from large-scale 3d datasets using bottom-up and top-down descriptors. In *Computer Vision–ECCV 2008*, pages 553–566. Springer, 2008. 00021.

[24] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep Learning with Sets and Point Clouds. *arXiv:1611.04500 [cs, stat]*, November 2016.

[25] Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer Science & Business Media. ISBN 978-1-4757-4321-0. Google-Books-ID: 8KgACAAAQBAJ.

[26] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning informative point classes for the acquisition of object model maps. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 643–650. IEEE, 2008.

[27] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009. 00311.

[28] Nima Sedaghat and Thomas Brox. Unsupervised Generation of a Viewpoint Annotated Car Dataset from Videos. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[29] Baoguang Shi, Song Bai, Zhichao Zhou, and Xiang Bai. DeepPano: Deep Panoramic Representation for 3-D Shape Recognition. *IEEE Signal Processing Letters*, 22(12): 2339–2343, December 2015. ISSN 1070-9908, 1558-2361. doi: 10.1109/LSP.2015. 2480802.

[30] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. In *Computer Vision–ECCV 2012*, pages 746–760. Springer, 2012.

[31] Richard Socher, Brody Huval, Bharath Bath, Christopher D. Manning, and Andrew Y. Ng. Convolutional-recursive deep learning for 3d object classification. In *Advances in Neural Information Processing Systems*, pages 665–673, 2012.

[32] Shuran Song and Jianxiong Xiao. Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images. *arXiv preprint arXiv:1511.02300*, 2015.

[33] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 567–576, 2015.

[34] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 945–953, 2015.

[35] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3D: Generic Features for Video Analysis. *arXiv preprint arXiv:1412.0767*, 2014.

[36] Dominic Zeng Wang and Ingmar Posner. Voting for Voting in Online Point Cloud Object Detection. In *Robotics: Science and Systems*.

[37] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[38] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

[39] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. SUN3D: A database of big spaces reconstructed using sfm and object labels. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1625–1632. IEEE, 2013.

[40] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, and Jianwei Wan. 3D Object Recognition in Cluttered Scenes with Local Surface Features: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2270–2287, November 2014. ISSN 0162-8828, 2160-9292. doi: 10.1109/TPAMI.2014.2316828.

Supplementary Material

8   Auto-Alignment of the Modelnet40 dataset

Modelnet40 [38] consists of more than 12000 *non-aligned* objects in 40 classes. We used the method of Sedaghat & Brox [23] to automatically align the objects class by class.

Mesh to Point-Cloud Conversion   The auto-alignment method of [23] uses point-cloud representations of objects as input. Thus we converted the 3D mesh grids of Modelnet40 to point-clouds by assigning uniformly distributed points to object faces.

Hidden faces in the mesh grids needed to be removed, as the so called Hierarchical Orientation Histogram (HOH) of [23] mainly relies on the exterior surfaces of the objects. We tackled this issue using the Jacobson's implementation [17] of the "ambient occlusion" method [22].

We tried to distribute the points roughly with the same density across different faces, regardless of their shape and size, to avoid a bias towards bigger/wider ones. Our basic point-clouds consist of around 50000 points per object, which are then converted to lighter models using the Smooth Signed Distance surface reconstruction method (SSD) [3] as used in [23].

Auto-Alignment   We first created a "reference-set" in each class, consisting of a random subset of its objects, with an initial size of 100. This number was then decreased, as the low-quality objects were automatically removed from the reference set, according to [23]. This reference set was then used to align the remaining objects of the class one by one.

For the HOH descriptor, we used 32 and 8 divisions in ϕ and θ dimensions respectively, for the root component. We also used 8 child components with 16 divisions for ϕ and 4 for θ – see [23].

Automatic Assignment of Number of Orientation Classes   As pointed out in the main paper, we do not use the same number of orientation classes for all the object categories. We implemented the auto-alignment procedure in a way that this parameter is automatically decided upon for each category: During generation of the reference-set in each class, the alignment procedure was run with 3 different configurations, for which the search space spanned over 360, 180 and 90 degrees of rotations respectively. Each run resulted in an error measure representing the overall quality of the models selected as the reference-set, and we designated respectively 12, 6 and 3 orientation levels to each category, whenever possible. When none of these worked, e.g. for the 'flower_pot' class, we assigned 1 orientation class which is equivalent to discarding the orientation information.

9   Analysis

To analyze the behavior of the orientation-boosted network, we compare it to its corresponding baseline network. We would like to know the differences between corresponding filters in the two networks. To find this correspondence, we first train a baseline network, without orientations outputs, for long enough so that it reaches a stable state. Then we use this trained net to initialize the weights of the ORION network, and continue training with a low learning rate. This way we can monitor how the learned features change in the transition from the baseline to the orientation-aware network.

In Figure 6 transition of a single exemplar filter is depicted, and its responses to different rotations of an input object are illustrated. It turns out that the filter tends to become more sensitive to the orientation-specific features of the input object. Additionally some parts of the object, such as the table legs, show stronger response to the filter in the orientation-aware network.

With such an observation, we tried to analyze the overall behavior of the network for specific object classes with different orientations. To this end we introduce the "dominant signal-flow path" of the network. The idea is that, although all the nodes and connections of the network contribute to the formation of the output, in some cases there may exist a set of nodes, which have a significantly higher effect in this process for an specific type of object/orientation. To test this, we take this step-by-step approach: First in a forward pass, the class, c, of the object is found. Then we seek to find the highest contributing node of the last hidden layer:

$$l^{n-1} = \arg\max_{k}\{w_{k,c}^{n-1} a_k^{n-1}\} \qquad (4)$$

where n is the number of layers, a_k^{n-1} are the activations of layer $n-1$, and $w_{k,c}^{n-1}$ is the weight connecting a_k^{n-1} to the cth node of layer n. This way we naively assume there is a significant maximum in the *contributions* and assign its index to l^{n-1}. Later we will see that this assumption proves to be true in many of our observations. We continue "back-tracing" the signal, to the previous layers. Extension of (4) to the convolutional layers is straightforward, as we are just interested in finding the index of the node/filter in each layer. In the end, letting $l^n = c$, gives us the vector l with length equal to the number of network layers, keeping the best contributors' indices in it. Now to depict the "dominant signal-flow path" for a group of objects, we simply obtain l for every member of the group, and plot the histogram of the l^is as a column. Figure 7(a) shows such an illustration for a specific class-rotation of the objects. It is clearly visible that for many objects of that group, specific nodes have been dominant.

In Figure 7(b), the dominant paths of the baseline and ORION networks for some sample object categories of the Modelnet10 dataset are illustrated. It can be seen that in the baseline network, the dominant paths among various rotations of a class mostly share a specific set of nodes. This is mostly visible in the convolutional layers – e.g. see the red boxes. On the contrary, the dominant paths in the ORION network rarely follow this rule and have more distributed path nodes. We interpret this as one of the results of orientation-boosting, and a helping factor in better classification abilities of the network.

←—— Object Orientations ——→

baseline

iterations

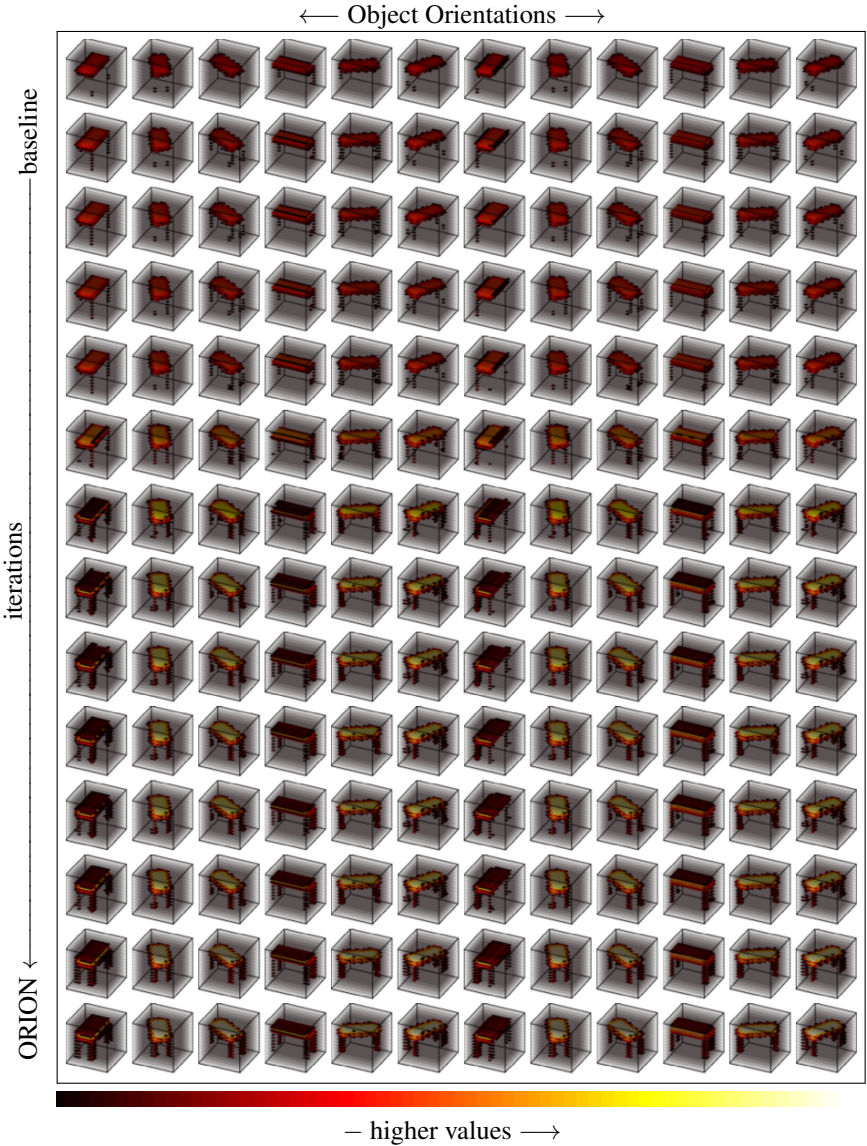ORION ↙

— higher values —→

Figure 6: The picture illustrates the activations of one of the nodes of the first layer, while the network transitions from a baseline network to ORION. The input is always the same object, which has been fed to the network in its possible discretized rotations (columns) at each step (row). We simulated this transition by first training the baseline network and then fine-tuning our orientation-aware architecture on top of the learned weights. To be able to depict the 3D feature maps, we had to cut out values below a specific threshold. It can be seen that the encoded filter detects more orientation-specific aspects of the object, as it moves forward in learning the orientations. In addition, it seems that the filter is becoming more sensitive to a *table* rather than only a horizontal surface – notice the table legs appearing in the below rows.
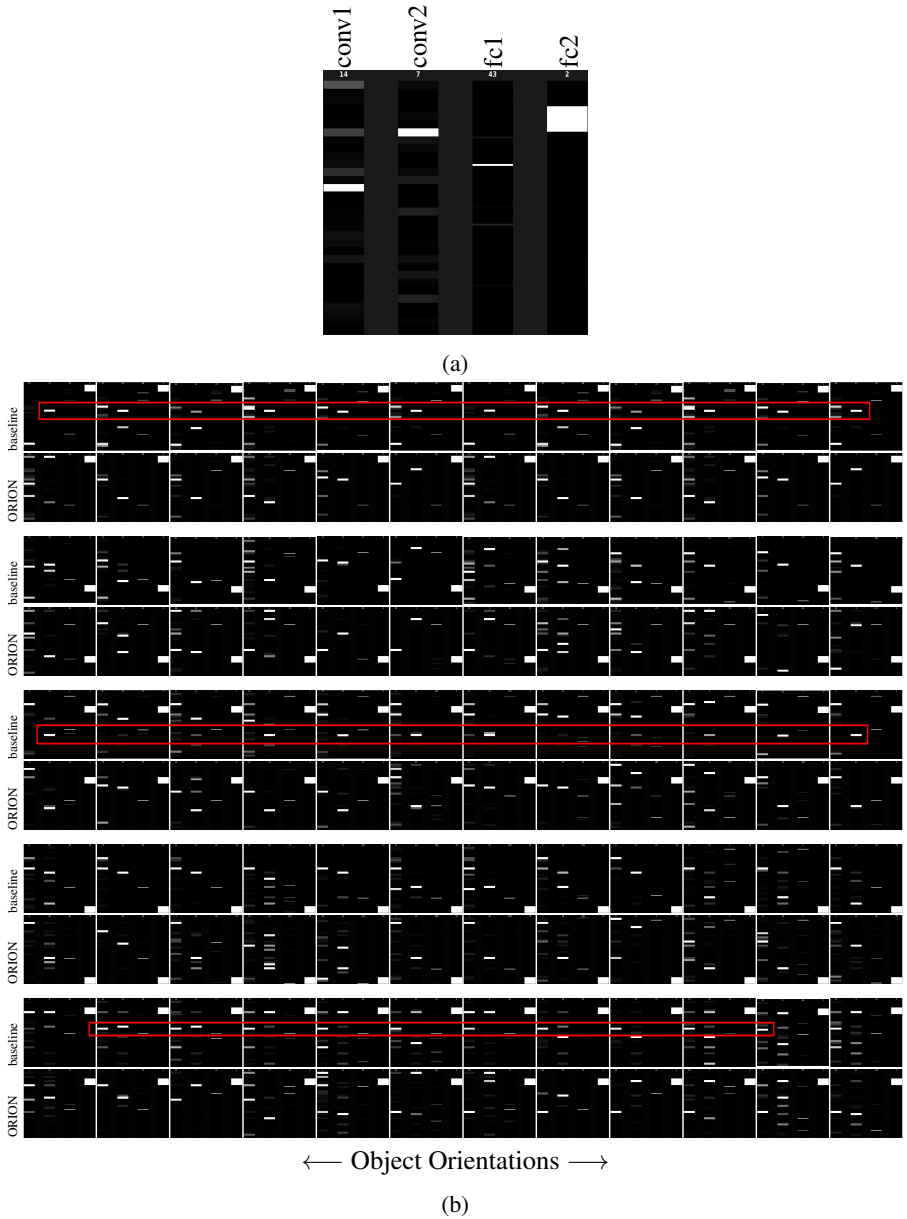
(a)

⟵ Object Orientations ⟶

(b)

Figure 7: (a) shows the "dominant signal-flow path" of the network, for an exemplar object category-orientation. Each column contains the activations of one layer's nodes. Obviously the columns are of different sizes. Higher intensities show dominant nodes for the specific group of objects. Details of the steps taken to form such an illustration are explained in the text. In (b), rows represent object classes, while in different columns we show rotations of the objects. So each cell is a specific rotation of a specific object category. It can be seen that in the baseline network, many of the rotations of a class, share nodes in their dominant path (e.g. see the red boxes), whereas, in the ORION network the paths are more distributed over all the nodes.

10   Extended Architecture

	Conv1	Conv2	Conv3	Conv4	Pool4
# of filters	32	64	128	256	-
kernel size	3x3x3	3x3x3	3x3x3	3x3x3	2x2x2
stride	2	1	1	1	2
padding	0	0	0	0	-
dropout ratio	0.2	0.3	0.4	0.6	-
batch normalization	✓	✓	✓	✓	-

	fc1	fc2:class	fc2:orientation
# of outputs	128	10/40	variable[†]
dropout ratio	0.4	-	-
batch normalization	×	×	×

Table 3: Details of the extended architecture introduced in Tables 1 & 2 of the main article.
[†] The number of nodes dedicated to the orientation output varies in different experiments.

11   Orientation Estimation Results

Although the orientation estimation was used merely as an auxiliary task, here in Table 4 we report the accuracies of the estimated orientation classes. Note that getting better results on orientation estimation would be possible by emphasizing on this task – e.g. see the detection experiment in the main article.

	Accuracy %			
	Sydney	NYUv2	ModelNet10	ModelNet40
ORION	71.5	51.9	89.0	86.5
ORION – Extended	70.1	54.5	89.3	87.6

Table 4: Orientation estimation accuracies on different datasets. The extended architecture of the second row, is the one introduced in the main article and detailed in Table 3 of this document.