CHESS RECOGNITION FROM A SINGLE DEPTH IMAGE

Yu-An Wei, Tzu-Wei Huang, Hwann-Tzong Chen

National Tsing Hua University, Taiwan

JenChi Liu

Industrial Technology Research Institute

ABSTRACT

This paper presents a learning-based method for recognizing chess pieces from depth information. The proposed method is integrated in a recreational robotic system that is designed to play games of chess against humans. The robot has two arms and an Ensenso N35 Stereo 3D camera. Our goal is to provide the robot visual intelligence so that it can identify the chess pieces on the chessboard using the depth information captured by the 3D camera. We build a convolutional neural network to solve this 3D object recognition problem. While training neural networks for 3D object recognition becomes popular these days, collecting enough training data is still a time-consuming task. We demonstrate that it is much more convenient and effective to generate the required training data from 3D CAD models. The neural network trained using the rendered data performs well on real inputs during testing. More specifically, the experimental results show that using the training data rendered from the CAD models under various conditions enhances the recognition accuracy significantly. When further evaluations are done on real data captured by the 3D camera, our method achieves 90.3% accuracy.

***Index Terms*—** 3D object recognition, volumetric representation, convolutional neural networks

1. INTRODUCTION

Building a robot that can play games of chess against a human opponent has been a way to showcase machine intelligence since the 18th century. Recently we just finish a project of constructing a chess-playing robot having the ability to recognize the chess pieces and to grip a chess piece for making a planned move. The robot, as shown in Fig. 1, has two arms with grippers and an Ensenso N35 Stereo 3D camera for capturing depth information. The implementation of this robotics AI system requires great effort to integrate robotics, mechanics, visual processing, and machine learning. In this paper, we focus on addressing the problem of recognizing the chess pieces from the depth information.

Object recognition is a fundamental problem in computer vision. Researchers have been working on classifying 2D objects from RGB images. Current trends in the designs of object recognition methods indicate that machine learning based approaches can generally outperform handcrafted systems. Many datasets that consist of color images of objects are available for training and for evaluating various learning-based models. The success of convolutional neural networks in solving object recognition is also owing to the availability of large-scale datasets with ground-truth annotations.

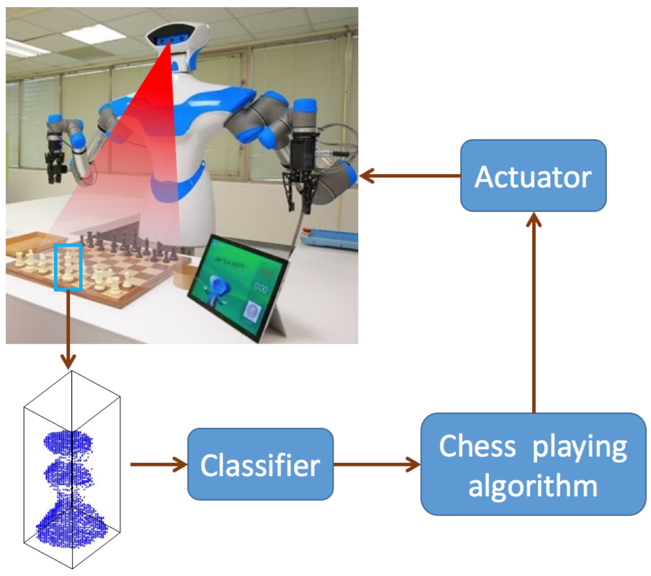

Fig. 1. An overview of our system. The depth camera is mounted on the robot's head to sense the chess pieces on the chessboard. Each individual chess piece is segmented from the depth image according to the homography derived from the chessboard. The point cloud of a chess piece is computed using the depth information and is fed into the convolutional neural network for recognition. The recognition results are forwarded to the chess-playing AI algorithm to decide a move and then to control the robot arm to complete the move.

While the study of 2D object recognition makes great progress in recent years, researchers also seek to develop learning-based algorithms for 3D object recognition using the information captured by stereo cameras or depth sensors. With the help of depth information, 3D object recognition should be able to resolve some situations that cannot be well handled in 2D object recognition, such as textureless objects or cluttered backgrounds. However, annotated

3D object datasets [1, 2] are not as plentiful as 2D datasets. For instance, the biggest RGBD dataset NYUv2 [3] only contains 1,449 labeled RGBD images. Such a size of dataset is much smaller than the typical 2D image datasets, which usually contain more than ten thousand or even one million images. Some recent works have employed 3D *computer aided design* (CAD) models to make up the shortage of 3D data, using open-source CAD models from SketchUp 3D Warehouse, TurboSquid and Shapeways.

Given the CAD models, a key issue of generating practical training data is choosing an effective representation for 3D data. Intuitively, we can set a virtual camera and render depth images from different viewpoints. We extract the 3D surface mesh of a CAD model, and represent the mesh as a point cloud. We then convert the point cloud into a binary volumetric cuboid. A binary volumetric cuboid is comprised of binary voxels. Each binary voxel indicates whether any 3D points are present at the corresponding location of the voxel. Therefore, we are able to represent the original point cloud in a quantized format. Furthermore, to cover possible variations that might occur in real situations, we may add noises of different levels to the binary volumetric cuboid. This also highlights an advantage of our choice of using binary representation in that we do have to simulate complex rendering conditions with the 3D CAD models but just need to handle Bernoulli noise. Other types of transformations can also be easily included to increase the diversities of training data.

Problem Definition

We assume that after preprocessing we can obtain the 3D bounding box of each chess piece on the chessboard. The point cloud enclosed by the bounding box is converted into the binary volumetric representation as is described in the previous section. We then formulate a 3D object recognition problem that takes an input of binary volumetric cuboid and produces an output of class label. For the problem of chess recognition, we consider six classes of chess pieces: `king`, `queen`, `bishop`, `knight`, `rook`, and `pawn`. We ignore the color because it is rather trivial to differentiate beforehand.

Our Approach

To solve the problem of predicting the classes of chess pieces from the inputs of point clouds, we train an end-to-end 3D convolutional neural network (CNN) using the training data generated by CAD models. We represent both the synthetic training data and the real test data as binary volumetric cuboids. We focus on generating various kinds of data with different settings of variations such as adding noise or removing points in models to simulate missing information. Because point clouds captured in real conditions are noisy in general, taking account of different variations will be useful for improving the CNN's performance when testing on real data. The CNN is able to model the contour, height, and depth information even when part of the point cloud is missing. With the simulated training varieties, the CNN can achieve an accuracy rate of 90% on real test data.

2. RELATED WORK

CNNs on 2D Images

Recent breakthroughs on object recognition in 2D domain have been made owing to the development of convolutional neural networks (CNNs) [4]. State-of-the-art methods on RGB image datasets, (*e.g.* ImageNet [5]), are able to achieve impressive performance. Besides object recognition, there are also significant improvements on 2D object detection, for example, RCNN [6], Fast-RCNN [7], Faster-RCNN [8], YOLO [9], and SSD [10]. These methods need to pinpoint possible object locations and recognize the object class inside the bounding box. For 3D objects, it could be easier to locate the objects using the available information of depth and reference plane. We can therefore focus on the problem of recognizing the object inside the bounding box.

CNNs on Depth and 3D Data

With the popularity of 3D sensors and applications such as virtual reality and augmented reality, more efforts are made on 3D objects recognition. Socher et al. [11] treat depth as an additional channel and feed it into an architecture consisting of CNN and RNN. Gupta et al. [12] use the HHA feature to encode depth information. HHA represents *height above ground*, *horizontal disparity*, and *angle between gravity and pixel normal*. The Sliding Shapes method [13] runs a 3D sliding window and directly recognizes each 3D window on depth images with the help of 3D CAD models.

To fully utilize 3D information, ShapeNet [14] builds a 3D CNN and considers 3D binary voxel grid as input. The Princeton ModelNet dataset [15] is also useful for object recognition. Other methods such as [16] and [17] use volumetric representations and also further enhance the performance. Apart from volumetric representations, MV-CNN [18] renders 3D models in different views and extracts features from each viewpoint. Qi et al. [19] have tried to analyze the effectiveness of different representations. FusionNet [20] uses multiple representations with significantly less parameters compared to standard CNNs using RGB images. Ren and Sudderth [21] present COG to encode 3D features learned by calculating 3D gradients.

3. VOLUMETRIC CONVOLUTIONAL NEURAL NETWORK

3D convolutional neural networks are widely used recently. The most common types for representing 3D shapes are mul-

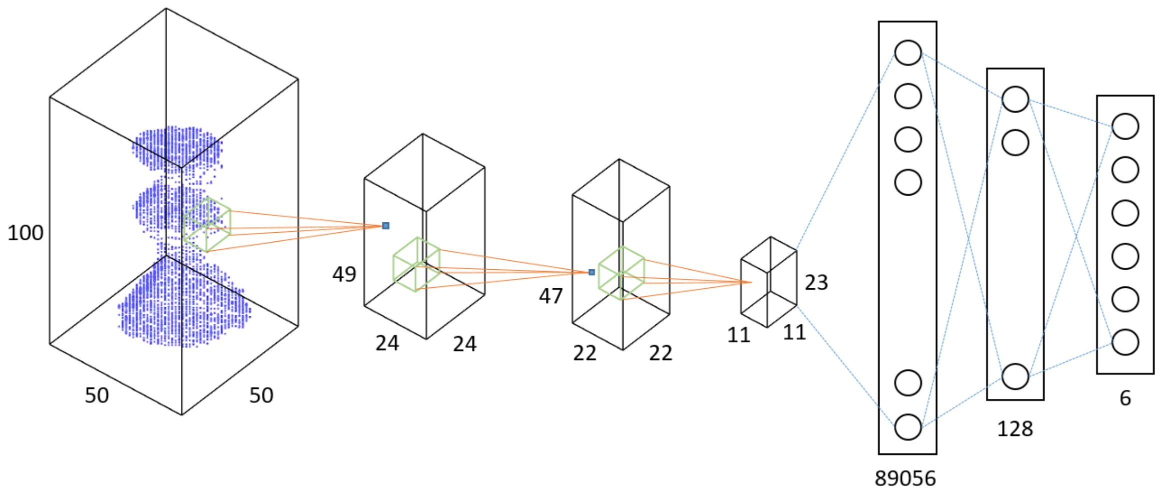

Fig. 2. An overview of the volumetric convolution neural network. The input is a cuboid of size 50 x 50 x 100. Each input will be processed by two 3D convolution layers with filter size five and three respectively, followed by one max pooling layer and two fully connected layers. Note that the stride is set to 2 and padding is 1 in the first convolutional layer.

tiview representations and volumetric representations. Multiview representations mean that a 3D model is rendered under multiple viewpoints and the respective features are extracted from different views. On the other hand, volumetric representations encode a 3D model directly from its shape. In this work, we choose volumetric representations since our test data, unlike in previous methods, are captured by real depth camera rather than 3D CAD models. The test data are noisy and often miss the information of the other side of a whole chess piece due to self-occlusion. Our volumetric CNN architecture is illustrated in Fig. 2.

3.1. Network Architecture

We train a volumetric convolutional neural network to classify the six classes of chess pieces. The input of the network is a tensor of size $50 \times 50 \times 100$. The input layer is followed by two convolutional layers and one max-pooling layer. The 3D filter size of the convolutional layers is $5 \times 5 \times 5$ and the pooling size of the max-pooling layer is $2 \times 2 \times 2$. After pooling, we reshape the pooling result and append two fully connected (FC) layers. The first FC layer shrinks the dimension to 128 and the second FC layer shrinks the dimension to six. Note that there are ReLU and dropout layers between the first three layers.

3.2. Training the Network

The weight of the network is initialized with standard Gaussian distributions. We use stochastic gradient decent with learning rate 0.01, weight decay 0.0001, and batch size 32 to train the network. The network converges in about one hour on an Nvidia TitanX GPU with four data loading threads. The network is implemented in Torch [22].

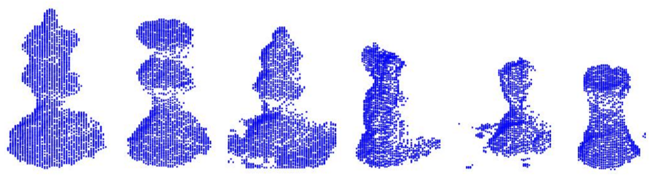

Fig. 3. Examples of real test data. From left to right: `king`, `queen`, `bishop`, `knight`, `pawn`, `rook`. The captured chess pieces differ in height and shape. Every point cloud may contain noisy points.

4. DATA AUGMENTATION

4.1. Gathering Test Data

Our input data are captured by an Ensonso N35 3D camera. The output of the camera is a point cloud of the whole scene with resolution 0.1 mm. Assume that a human player places the chess pieces moderately. Since the camera is calibrated, the point cloud of each chess piece on the chessboard can be easily extracted. After extraction, we locate the center p of a chess piece's bottom plane by fitting a circle on that plane. Starting from p, we quantize the point cloud along the x, y, and z directions with bin size of 1 mm to form a cuboid, and the extracted length is 50 mm, 50 mm, 100 mm respectively. The dimension of the cuboid is $50 \times 50 \times 100$. Also, p is

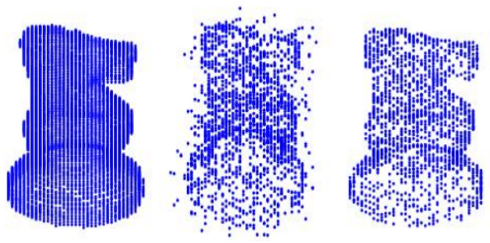

Fig. 4. An example of variation of `knight`. From left to right: the original chess piece, slightly shifting 50% points and removing 10% points, and randomly removing 10% points.

aligned with the center of the cuboid's bottom. We call each 1 mm $\times$ 1 mm $\times$ 1 mm cell a voxel. As mentioned earlier, we use a binary volumetric representation. The value of each voxel is set to zero if the voxel is empty. Otherwise, it is set to one if the voxel contains points. Some examples of the preprocessed test data are shown in Fig. 3.

4.2. Synthesizing Training Data

Our volumetric convolutional neural network needs to classify the six classes of chess pieces, which are `bishop`, `king`, `knight`, `pawn`, `queen`, `rook`. Each CAD model is rotated by every 6 degrees to generate data that cover different orientations of a chess piece. We further augment the data by varying its height and shape so that the network can learn to recognize imperfect inputs.

We consider five kinds of variations **Nor50**, **Jitter**, **Removal**, **Noise**, **Whole** to augment the data.

- **Nor50**: All cuboids are compressed from 50 $\times$ 50 $\times$ 100 to 50 $\times$ 50 $\times$ 50.
- **Jitter**: The value in each voxel is shifted to nearby voxels around the center of cuboid, within a range of ± 5 voxels in each direction.
- **Removal**: The value in each voxel is randomly set to zero with probability from 0% to 90%.
- **Noise**: Randomly choose 0% to 90% voxels and shift the value of each chosen voxel by ± 5 voxels in each direction from the original position.
- **Whole**: We may choose to keep the whole set of point cloud without take the visibility into consideration. Or, we may set a virtual camera direction and remove invisible points, i.e., the points whose normal directions are at an angle larger than 90 degree to the camera direction.

Table 1. Experimental results with different setting of variations.

Variations	Accuracy
nor50	30%
jitter	60%
jitter+removal	64%
jitter+whole+removal	84.40%
jitter+whole+removal+noise	**90.30%**

We use five parameters for the five aforementioned variations to generate our training data. Each combination generates 100K to 300K training data depending on what parameters are chosen. Note that the training data can be applied with more than one variation. For example, we can remove 10% points in the cuboid, and then shift values away from the center of cuboid.

4.2.1. Variations in Height

We set our input size of the binary cuboid as 50 $\times$ 50 $\times$ 100 (width $\times$ depth $\times$ height) in both training and test data. Specifically, we do not normalize every model into the same size. We put every model in the cuboid with their original scale. In this way, we can prevent squeezing the shape of tall models such as `king` and `queen`, and can also retain the height cue for each model.

4.2.2. Variations in Shape

Missing information at the boundary of the point cloud may make the shape of the model become difficult to recognize. It often happens in real scenarios due to the sensing conditions of the depth camera. To solve this problem, we remove 0% to 90% of points in original CAD models randomly. Also, we pretend that there is a virtual camera pointing at the model at different heights. A point should be removed if the angle between the camera's direction and the normal of that point is larger than 90 angle, since such a point is not visible in real situation.

Noisy points are often caused by incorrectly estimated bounding boxes. Some points that are not part of a real chess piece might be included in the extracted point cloud. The noisy points would further cause error in the step of fitting and finding the center of the chess piece. The chess piece is therefore moved off-center. To simulate such phenomenon, we move the center of each model in the cuboid to add variations in training data. We also randomly shift 0% to 90% of the points in original CAD models. Fig. 4 shows some variations of training data.

Table 2. Confusion matrix of the classification results.

	bishop	king	knight	pawn	queen	rook
bishop	0.96	0	0.032	0.008	0	0
king	0	0.568	0.01	0	0.422	0
knight	0	0	0.953	0.005	0	0.042
pawn	0	0	0	0.946	0	0.054
queen	0	0	0.001	0	0.999	0
rook	0	0	0	0.053	0	0.947

5. EXPERIMENTS

We use the synthetic training data with different settings of variations to train our network, and evaluate the performance using the same test data for comparison. The total number of real test data is 5000. The results are shown in Table 1.

5.1. Importance of Height Variations

The experimental results show that the height information is critical. As we can see in Table 1, if the training and test data are normalized to 50 x 50 x 50 cuboid, the accuracy drastically drops to 30%. It is important to keep the height variations of different chess pieces.

5.2. Analysis on Shape Variations

Overall, the combination of shifting the chess piece in cuboid, randomly removing points, adding noises, and preserving all points regardless of camera directions yields the best result. It seems that preserving all points regardless of camera directions is the most critical factor to improve the performance. Based on this outcome, we may argue that no matter the points are occluded or not, it would be more informative to provide a more complete point cloud during training as if the model can be examined in all directions. In other words, imagining that we discover some unknown object, we will look around at it to get a more holistic impression of the shape rather than only see half of the object and ignore the other side. Another important parameter is adding slight shifts to the points in three directions. Adding noise is a good way to imitate real test data. It helps to improve the accuracy to 90% at last.

Our method performs well on most classes (above 90%) except the `king`, see Fig. 2. We think the reason is that the `king` looks almost the same as the `queen` everywhere except the crown. In real data, the situation of missing crown is usually caused by placing the depth camera too high so that the chess piece is almost viewed from the top. Also, noise and insufficient resolution of the depth camera could make the crown unrecognizable. Sometimes a knight might be misclassified as a pawn or a rook when the knight is not facing the depth camera. We may use heuristics and multiple predictions to resolve these mis-classifications in real game playing. A few examples of false classifications are shown in Fig. 5.

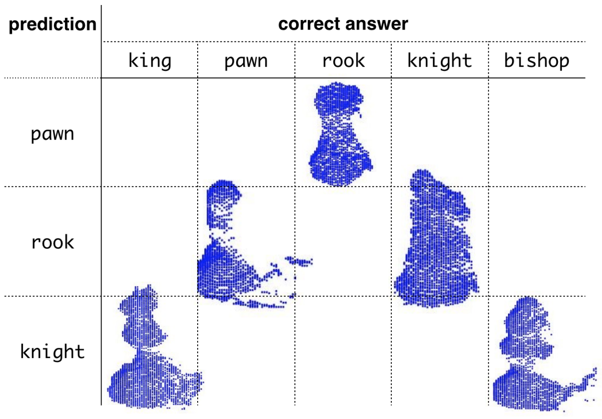

Fig. 5. Wrong classifications.

6. CONCLUSION

We train a 3D convolutional neural network to classify the point clouds captured by a depth camera. With the help of CAD models, we can generate a large number of synthetic data to train the network. We conduct several experiments such as adding noises, shifting models, or removing points to simulate the captured information in real scenarios. We show that including different variations is useful for improving the recognition accuracy on real test data. The trained network is integrated into a chess-playing robot for interactive demonstration.

7. ACKNOWLEDGE

This research was supported in part by ITRI grant F101WD2400 and MOST grant 103-2218-E-007-017-MY3.

8. REFERENCES

[1] Allison Janoch, Sergey Karayev, Yangqing Jia, Jonathan T. Barron, Mario Fritz, Kate Saenko, and Trevor Darrell, "A category-level 3-d object dataset: Putting the kinect to work," in *IEEE International Conference on Computer Vision Workshops, Barcelona, Spain, November 6-13, 2011*, 2011, pp. 1168–1174.

[2] Jianxiong Xiao, Andrew Owens, and Antonio Torralba, "SUN3D: A database of big spaces reconstructed using sfm and object labels," in *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, 2013, pp. 1625–1632.

[3] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus, "Indoor segmentation and support inference from RGBD images," in *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision,*

Florence, Italy, October 7-13, 2012, Proceedings, Part V, 2012, pp. 746–760.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Intelligent Signal Processing*. 2001, pp. 306–351, IEEE Press.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, 2009, pp. 248–255.

[6] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, 2014, pp. 580–587.

[7] Ross B. Girshick, "Fast R-CNN," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 1440–1448.

[8] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 91–99.

[9] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 779–788.

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg, "SSD: single shot multibox detector," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, 2016, pp. 21–37.

[11] Richard Socher, Brody Huval, Bharath Putta Bath, Christopher D. Manning, and Andrew Y. Ng, "Convolutional-recursive deep learning for 3d object classification," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 665–673.

[12] Saurabh Gupta, Ross B. Girshick, Pablo Andrés Arbeláez, and Jitendra Malik, "Learning rich features from RGB-D images for object detection and segmentation," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII*, 2014, pp. 345–360.

[13] Shuran Song and Jianxiong Xiao, "Sliding shapes for 3d object detection in depth images," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI*, 2014, pp. 634–651.

[14] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu, "ShapeNet: An Information-Rich 3D Model Repository," Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

[15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao, "3d shapenets: A deep representation for volumetric shapes," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 1912–1920.

[16] André Brock, Theodore Lim, James M. Ritchie, and Nick Weston, "Generative and discriminative voxel modeling with convolutional neural networks," *CoRR*, vol. abs/1608.04236, 2016.

[17] Daniel Maturana and Sebastian Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, 2015, pp. 922–928.

[18] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 945–953.

[19] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," *CoRR*, 2016.

[20] Vishakh Hegde and Reza Zadeh, "Fusionnet: 3d object classification using multiple data representations," *CoRR*, vol. abs/1607.05695, 2016.

[21] Zhile Ren and Erik B. Sudderth, "Three-dimensional object detection and layout prediction using clouds of oriented gradients," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[22] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.