

A Short Survey of Deep Learning Methods for 3-D Geometric Models

Shayan Hoshyari, Zicong Fan, Dylan Green

1 Deep Learning for 3-D Geometry

Three-dimensional geometric models are widely used in many fields such as computer graphics, computer-aided design (CAD), visualization, and multimedia. These models are represented in various forms, the most common of which include surface meshes, point clouds, voxels, and implicit functions (shown in Figure 1). Naturally, algorithms for processing such data are of high demand. Following the great success of deep learning methods for image processing, there has been a recent trend to generalize such methods to 3-D data. Figure 2 shows some of the applications where deep learning methods have been most successful. In this document, we will review some of the recent papers that devise deep learning architectures to reason about 3-D geometric models.

1.1 Multi-View Based Methods

We start with the work of Su et al. [2015], which is based on a simple observation: a 3-D model can be rendered as one or multiple 2-D images, which can, in turn, be processed by conventional convolutional neural networks (CNN). This approach is often referred to as multi-view based and can be used for applications such as object classification and model retrieval. The networks are referred to as a multi-view convolutional neural network (MVCNN).

Figure 3 depicts a schematic of the MVCNN workflow. First, the input which is 3-D surface mesh is rendered as 2-D images from various angles. These images are fed to a conventional CNN (tied across all renders). The per-image outputs of the first CNN are then pooled (using the max function) to produce a single set of hidden units and are then passed to a second CNN. The second CNN will then output per-class probabilities, or its hidden units at a particular layer can be used to define a metric for object retrieval.

The method is evaluated on the ModelNet40 [Wu et al., 2015] dataset for classification and object retrieval tasks. It achieves an accuracy of 90% for classification, which has not been beaten (at least by a considerable margin) even to this date (Table 1 shows comparisons against other methods). A strength of this method is that it leverages existing tools in the image processing community, and it can use pre-trained networks on natural images. On the other hand, the method has many parameters such as the location and angle of the camera for the rendered images, and the parameters of two large CNNs for image classification. Furthermore, it is not clear how the method can be extended for applications that need per point labels (e.g., segmentation or normal estimation).

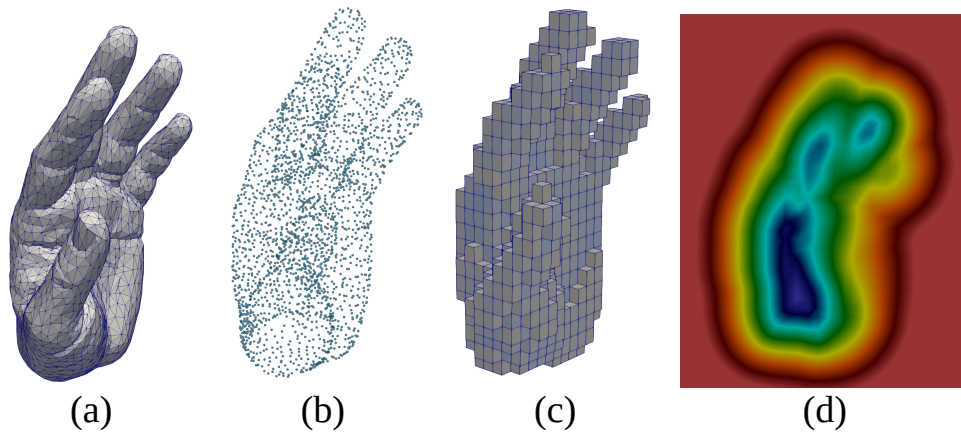


Figure 1: Examples of 3-D geometry representations: a) surface mesh, b) point cloud, c) voxels, and d) implicit function. Note that implicit functions are often, but not necessarily, stored on a spatial grid. Furthermore, voxels are a special case of piecewise constant binary implicit functions that are true in the grid cells that lie inside the shape and are false otherwise.

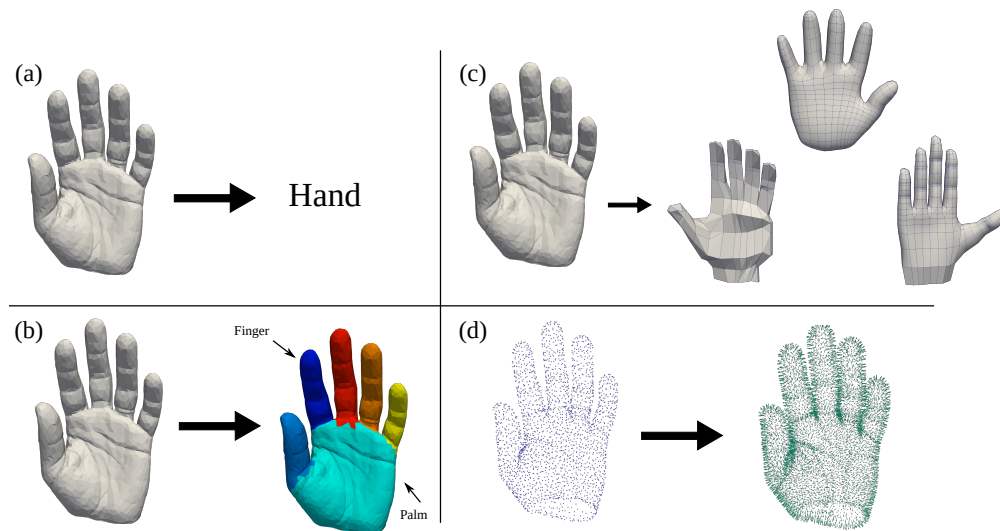


Figure 2: Examples of deep learning applications on 3-D data: a) classification, b) semantic segmentation, c) closest object retrieval, and d) point-cloud normal estimation. Note that in the retrieval task considered here is not concerned with *generating* similar examples, instead the goal is to *find* similar examples in the training data using a metric defined by the networks hidden units. Furthermore, the point cloud normal estimation task is trivial unless the pointcloud is relatively sparse.

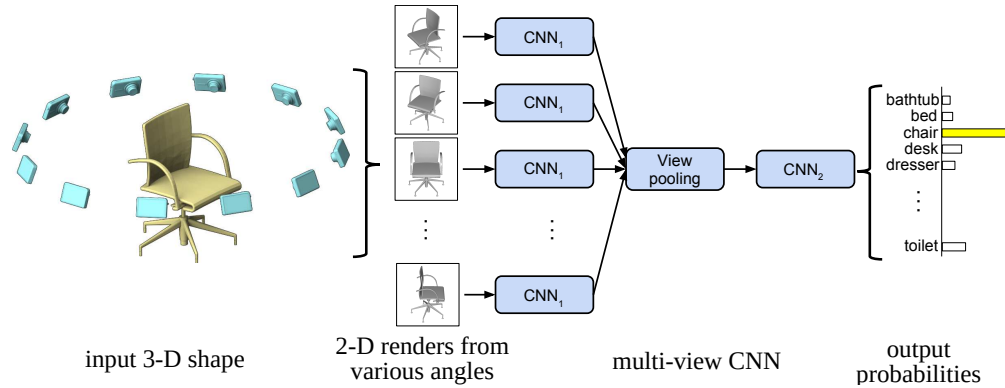


Figure 3: A schematic of the MVCNN architecture. Image is courtesy of Su et al. [2015].

1.2 Voxel Based Methods

Another obvious extension of CNNs to 3-D data is to represent geometric models as a field over a spatial grid, where convolution operations are well defined, and optimized implementations are available. Common choices include a binary inside-outside function, the surface normal field, and distance from the surface (possibly multiplied by a Gaussian filter).

In this front, Wu et al. [2015] use a binary function on a uniform $30 \times 30 \times 30$ grid as their input data representation. Instead of a discriminative model, they build a generative, convolutional deep belief network (CDBN). This allows them to support a wide range of applications such as classification, simultaneous shape recognition and completion from depth images, next best view prediction, object retrieval, and even object generation. The paper introduces the ModelNet dataset and uses it to validate the performance of the network. The main weaknesses of the method are that training a generative model is quite difficult and requires a lot of heuristic machinery. Moreover, an input resolution of $30 \times 30 \times 30$ (a restriction imposed by memory and computation cost) might not be enough to capture the inputs well enough, which can partly explain the loss of accuracy in classification (Table 1).

Wang et al. [2017] and Riegler et al. [2017] observe that the change in the input field is only abrupt near the model surface, and use it to alleviate the high memory cost of a uniform spatial grid. Instead of a uniform grid, they use an octree (one among many approaches to build a non-uniform grid) to represent the input field and implement pooling and convolution operations for this representation. The main difference between their methods is different algorithmic choices for efficient octree representation and convolution operations on the GPU. Also, Wang et al. [2017] use the normal field as the input, while Riegler et al. [2017] use the binary inside-outside function. The methods are evaluated on the ModelNet [Wu et al., 2015], [Riemenschneider et al., 2014], and [Löhner et al., 2016] datasets for applications including classification, segmentation, and retrieval (see Table 1 for comparisons of classification accuracy). Perhaps the main downside of this two papers is that they cannot be trivially implemented for execution on the GPU using conventional neural network libraries (e.g., TensorFlow [Abadi et al., 2016], PyTorch [Paszke et al., 2017]) and need hand-tailored GPU code. Moreover, the approach of Wang et al. [2017] requires the surface normals, which might not be readily available, or cannot be confidently computed for complicated sparse point clouds (and it obviously cannot do normal estimation). Finally, in some cases, it is not clear which improvements are due to the novel convolution operations, and which are due to the different network architecture, or training procedures.

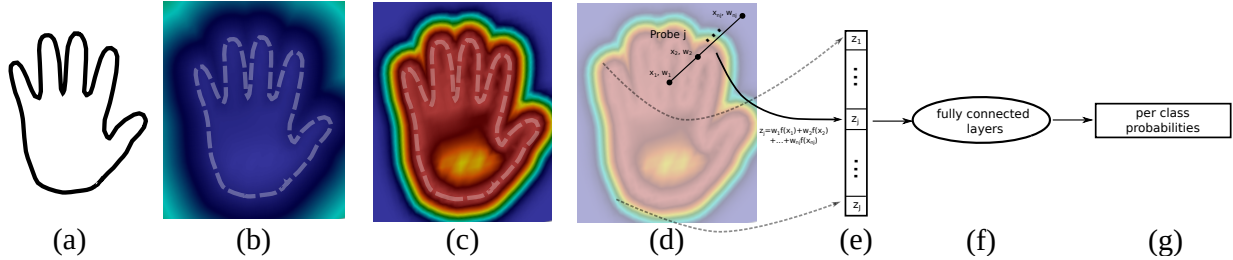


Figure 4: A schematic of the field probing neural network (FPNN) architecture for classification. The input is a geometric model (a), the distance from surface field of the model, f , is calculated (b), the distance is multiplied by a Gaussian filter to increase the attention to the model surface (c), each probe which consists of a set of positions, x_i , and weights, w_i (d) outputs a hidden unit $z = \sum_i w_i f(x_i)$ (e). The hidden units z are then passed through a fully connected layer (f) to obtain the final per-class probabilities (g).

1.3 Departing From Voxels

Several works have explored the alternative to voxel-based convolutions. Li et al. [2016] propose a field probing scheme (Figure 4). They first convert the input shape into a distance field filtered by a Gaussian to increase the attention to the model surface. They would then sample this field at several positions via a set of probes. Note that each probe is simply a set of positions and weights which act as shown in Figure 4. The positions and weights of each filter are learned during training, while the number of filters and the number of points/weights per filter are hyper-parameters. The output units of the probes are then passed to a fully connected network to predict per-class probabilities. The method only supports the task of classification, and is validated on the ModelNet40 [Wu et al., 2015] dataset (see Table 1 for accuracy). The proposed method also has many hyper-parameters (number of probes and their positions and weights) and is very sensitive to initialization. On the other hand, the method is much more memory efficient than uniform grid based convolutions, while it is much simpler to implement than octree based convolutions.

Qi et al. [2017a] introduce a network architecture that can directly process point clouds. The input to the network is the 3-D position (and optionally additional information such as normals or colors) of a variable number of points, n . The main challenge, in comparison to a conventional convolutional neural network (CNN), is that the output should be invariant under rigid body transformations or permutations of the input points. They introduce the vanilla pointnet architecture (shown in Figure 5) which satisfies the order invariance property and theoretically prove that it can approximate any order invariant function, given enough hidden units. With the addition of further heuristics, they attempt to improve the networks invariance under rigid body transformation and allow for part segmentations. We do not go into the detail of these heuristics as they were beaten by the authors' next extension of pointnet [Qi et al., 2017b]. The method is validated by considering tasks such as object recognition, segmentation of single objects, and segmentation of scenes on multiple 3-D shape datasets (ModelNet40 [Wu et al., 2015], Shapenet Part [Chang et al., 2015], and Stanford 3D semantic parsing [Armeni et al., 2017] datasets). The proposed network architecture achieves results on par with the state of the art methods that use convolutions on voxels (see Table 1 for classification accuracy). Since sampling points from the surface of a 3-D model leads to much less data compared to multi-view renders or uniform volumetric sampling. Thus, the proposed network is more efficient ($\sim 10 - 100\times$) and has fewer parameters ($\sim 10\times$) compared to its grid-based and multi-view-based predecessors. The sampling of point clouds can also be done adaptively for further optimization. Nevertheless, the neighborhood of each point is completely ignored in the network, until the very last max-pooling layer. As this strategy is ignoring valuable information from the neighborhood (e.g., curvature), the proposed network might need a substantial number of training examples and can be



Figure 5: A schematic of the vanilla pointnet architecture for classification. The input is a $n \times 3$ matrix, where each row is the coordinates of a point in the point-cloud. First, each point is passed through a fully connected neural network with $(3, 64, 128, 1024)$ units tied across all points. The per-point outputs are then pooled using the max function, and passed to a second fully-connected neural network with $(1024, 512, 256, k)$ units, where k is the number of classes.

very sensitive. The authors’ next work [Qi et al., 2017b] attempts to fix this issue.

1.4 Generalized Convolution and Pooling Operations

Some papers introduce novel convolution and pooling like the transformation that are not based on voxel grids. Klovov and Lempitsky [2017] introduce such an operation based on k -d trees. Their framework for classification is shown in Figure 6. Given an input point cloud, a k -d tree of a certain depth is constructed based on the point cloud’s density. The point cloud is then subsampled such that a single point exists per tree leaf which constitutes the input of the network. For each tree level and each stump direction, a matrix, W , and a bias vector, b , is defined. Then, at each layer of the network, the depth of the tree is reduced by one, and the hidden unit of each new leaf is found as a function of its children, and the W matrix and b vector of the corresponding stump. For example, in Figure 6,

$$z_4 = \sigma \left(W_{\text{green}} \begin{bmatrix} z_8 \\ z_9 \end{bmatrix} + b_{\text{green}} \right),$$

where σ is the activation function. The W matrices and b vectors are the parameters of the network and are determined by training. The approach is validated on the MNIST, ShapeNet Core [Chang et al., 2015], and ModelNet [Wu et al., 2015] datasets for classification and part segmentation (see Table 1 for classification accuracy). Overall, the method seems to be very accurate in classification compared the concurrent work. For segmentation, the method seems to be inferior to pointnet [Qi et al., 2017a]. It would be interesting to see the performance of the method on more complicated datasets, and more challenging tasks such as scene segmentation. Also, the k -d tree construction task does not seem to be very GPU friendly and constitutes a large portion of the training/testing time.

Similarly, Qi et al. [2017b] introduce a novel convolution and pooling like transformation. Instead of a k -d tree, they use the vanilla pointnet [Qi et al., 2017a] network to build this operation. A simplified version of their network for classification is shown in Figure 7. Given an input point cloud, they sample a series of furthest points and then apply tied pointnet architectures to all the points in a certain neighborhood of the sampled ones. They also introduce an approach to combine multiple neighborhood sizes of each sample, to increase the network’s robustness to nonuniform point cloud densities. The process is repeated for multiple layers until a single final unit (but with a large dimension) is arrived at. This unit is finally passed to a fully connected neural network that generates per-class probabilities. A slightly more complicated hourglass network with skip connections is also proposed for shape segmentation, where interpolation is used as the deconvolution operator. The proposed network is validated on the MNIST, ModelNet40 [Wu et al., 2015], SHREC16 [Löhner et al., 2016], and ScanNet [Dai et al., 2017] datasets for classification and segmentation (see Table 1 for classification accuracy). The method outperforms ablations studies are also performed to investigate the effect of missing data and non-uniform point cloud densities, which the method successfully

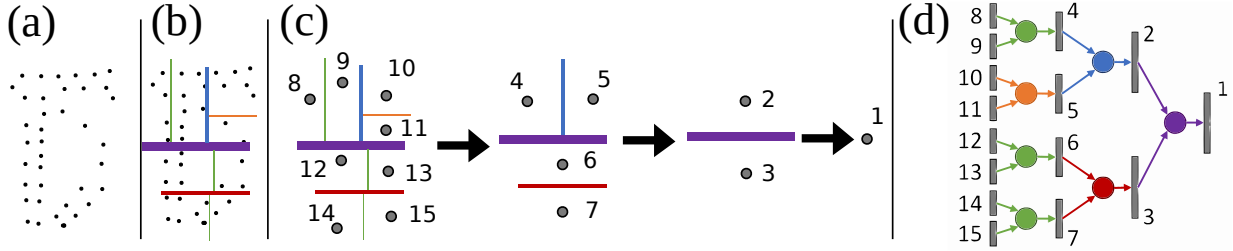


Figure 6: A schematic of the classification architecture of Klokov and Lempitsky [2017]. a) Input shape as a point cloud. b) A k -d tree is constructed based on the density of this point-cloud. c) The point-cloud is subsampled, such that there is only a single point per leaf. These points are then transformed to arrive at a single output vector. d) The order at which the input and hidden unit points are combined is shown in more detail. Circles with the same colors denote parameter tying. The figure is partly adapted from [Klokov and Lempitsky, 2017].

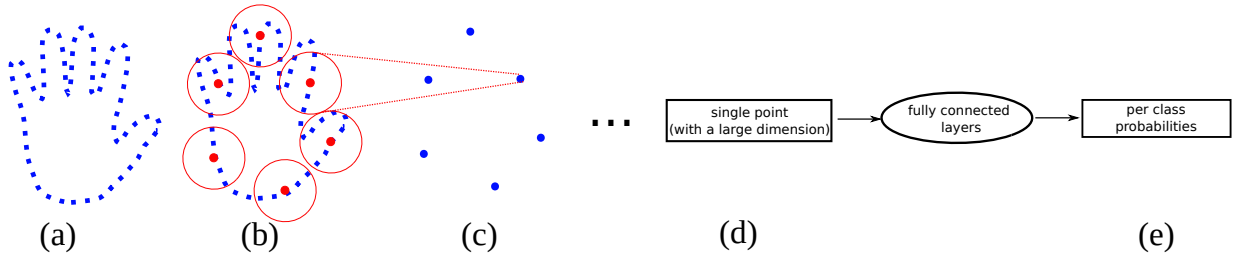


Figure 7: A schematic of the classification architecture of pointnet++ [Qi et al., 2017b]. a) Input shape as a point cloud (in blue). b) A series of farthest distance points are sampled from the point cloud (in red). c) A tied vanilla pointnet network [Qi et al., 2017a] is applied at a certain neighborhood of each point to arrive at the next layers hidden units. d) This process is repeated until we arrive at a single unit with a possibly large dimension. e) The final unit is passed through a fully connected neural network that outputs per-class probabilities.

handles. Pointnet++ seems to be a strong architecture, at the cost of many parameters, and non-GPU friendly operations (e.g., furthest point sampling, and finding points within a certain distance).

Atzmon et al. [2018] define convolution and pooling operations on pointclouds by leveraging radial basis functions (RBF). Their convolution operation is shown in Figure 8. This operation first extends per-point information in a point-cloud to a continuous function in the ambient volume using RBF basis functions. Then, a volume convolution is applied to this smooth function, where the convolution itself is also constructed using RBFs. Finally, the volumetric function is restricted back to the point cloud by sampling. In a similar manner, pooling and restriction (see original paper for more details) are defined and used to construct classification and segmentation networks. The networks are validated on the ModelNet40 [Wu et al., 2015] and Shapenet Part [Chang et al., 2015] datasets for classification and segmentation, respectively (see Table 1 for classification accuracy). One plus side of this approach is perhaps its elegance and unifying nature. Also, it is theoretically justified to be robust to noisy and sparse point-clouds. On the other hand, the extension/restriction operation heavily rely on sparse matrix operations and are difficult to optimize.

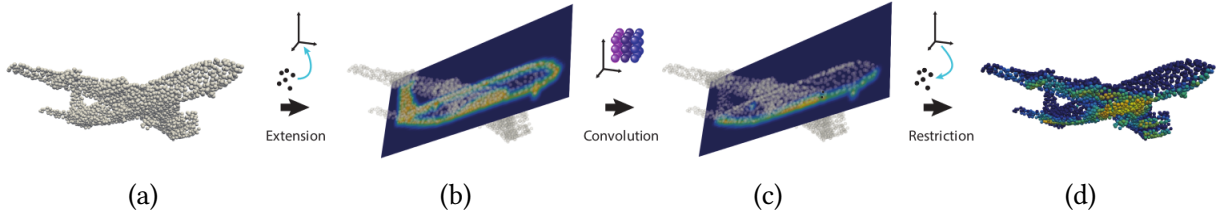


Figure 8: Schematic of the convolution operation of Atzmon et al. [2018]. A function defined on a point-cloud (a) is first extended to the ambient volume using RBF basis functions (b), to which a volume convolution is then applied (c). Finally, the volumetric function is restricted back to the point cloud by sampling (d). The Figure is courtesy of Atzmon et al. [2018].

Table 1: Classification accuracy of the reviewed methods on the ModelNet40 dataset.

method	accuracy	method	accuracy
MVCNN ⁺ [Su et al., 2015]	90.1%	3-D Shapenets [Wu et al., 2015]	77.3%
OCNN* [Wang et al., 2017]	90.6%	OCTNET[Riegler et al., 2017]	86.5%
FPNN [Li et al., 2016]	88.4%	pointnet [Qi et al., 2017a]	89.2%
escape-from-cells [Klokov and Lempitsky, 2017]	90.6%	pointnet++ [Qi et al., 2017b]	90.7%
RBF based convolution [Atzmon et al., 2018]	92.3%		

⁺: uses mesh structure

*: uses normals

1.5 Graph Based Methods

As mentioned above, pointnet achieved promising results by applying feed-forward operations on individual points and aggregating them with an unordered aggregating operation. Since pointnet does not sufficiently exploit local structure, pointnet++ was introduced to address this issue by partitioning points into smaller clusters and perform pointnet operations on each cluster. As a result, pointnet++ provides an improvement in performance but suffers from computing time (also a slight increase in the number of parameters compared to vanilla pointnet). Due to the trending attention of graph-based neural network in the deep learning community [Kipf and Welling, 2016; Yang et al., 2018; Teney et al., 2017] (semi-supervised document classification, image scene graph generation and visual question answering), there is also attention for exploiting graph structure for point clouds. For example, KCNet was introduced by Shen et al. [2018] to improve the computational efficiency of pointnet++. Specifically, they provide two operations for local point cloud structure. One operation is to define a convolution kernel for point cloud to capture the local geometric structure of point cloud. Another emphasizes on feature aggregation on a k-nearest-neighbor graph based on 3d position. As a result, KCNet achieves comparable accuracy in point cloud classification and parts segmentation tasks while maintaining the number of parameters and inference speed similar to that of the vanilla pointnet (see Table 2).

Table 2: Model size and inference time. “M” stands for million. Networks were tested on a PC with a single NVIDIA GTX 1080 GPU and an Intel i7-8700@3.2 GHz 12 cores CPU (caption provided in the original paper).

Method	#params (M)	Fwd. time (ms)
PointNet (vanilla)	0.8	11.6
PointNet++	1.0	163.2
KCNet (M=16)	0.9	18.5
KCNet (M=3)	0.9	12.0

References

- Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. *ICCV*, 2015.
- Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D Shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015.
- Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM TOG (SIGGRAPH Proc.)*, 2017.
- Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3D representations at high resolutions. In *CVPR*, 2017.
- Hayko Riemenschneider, András Bódis-Szomorú, Julien Weissenberg, and Luc Van Gool. Learning where to classify in multi-view semantic segmentation. In *ECCV*. Springer, 2014.
- Z. Löhner, E. Rodolà, M. M. Bronstein, D. Cremers, O. Burghard, L. Cosmo, A. Dieckmann, R. Klein, and Y. Sahillioglu. Shrec’16: Matching of deformable shapes with topological noise. In *Proc. of Eurographics Workshop on 3D Object Retrieval (3DOR)*, 2016.
- Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016.
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- Yangyan Li, Soeren Pirk, Hao Su, Charles R Qi, and Leonidas J Guibas. FPNN: Field probing neural networks for 3D data. In *NeurIPS*, 2016.
- Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. *CVPR*, 2017a.
- Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017b.
- 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. Technical report, Stanford University – Princeton University – Toyota Technological Institute at Chicago, 2015.

- I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, 2017.
- Roman Klokov and Victor Lempitsky. Escape from cells: deep kd-networks for the recognition of 3D point cloud models. In *ICCV*, 2017.
- Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017.
- Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM TOG (SIGGRAPH Proc.)*, 2018.
- Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *NeurIPS*, 2016.
- Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. In *ECCV*, 2018.
- Damien Teney, Lingqiao Liu, and Anton van den Hengel. Graph-structured representations for visual question answering. In *CVPR*, 2017.
- Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *CVPR*, 2018.

A Explicit List of Reviewed Papers

- Methods based on multi-view images
 - (1) Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. *ICCV*, 2015
- Methods based on grids
 - (2) Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D Shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015
 - (3) Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3D representations at high resolutions. In *CVPR*, 2017
 - (4) Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM TOG (SIGGRAPH Proc.)*, 2017
- New ideas for pointclouds
 - (5) Yangyan Li, Soeren Pirk, Hao Su, Charles R Qi, and Leonidas J Guibas. FPNN: Field probing neural networks for 3D data. In *NeurIPS*, 2016
 - (6) Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. *CVPR*, 2017a
 - (7) Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017b
 - (8) Roman Klokov and Victor Lempitsky. Escape from cells: deep kd-networks for the recognition of 3D point cloud models. In *ICCV*, 2017
 - (9) Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM TOG (SIGGRAPH Proc.)*, 2018
- Graph neural network for point clouds
 - (10) Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *CVPR*, 2018