Point Cloud Semantic Segmentation using Graph Convolutional Network

Wentao Yuan
Robotics Institute
Carnegie Mellon University
`wyuan1@cs.cmu.edu`

1 Introduction

With the development of 3D sensors, there is an increasing interest in understanding 3D data using deep learning techniques. In particular, there has been many attempts on extending the convolutional architecture which has proven to be very successful in 2D image understanding to 3D data. However, the output of 3D sensors is usually a collection of 3D points called a point cloud. Unlike 2D images, point clouds do not lie on a regular grid and thus do not possess the translational-invariant structure that enables convolution. To leverage the advantage of convolutional networks such as weight sharing and local connectivity, raw point clouds are usually converted to some other representation, such as 3D occupancy grids or rendered 2D images. However, these conversions are often irreversible, and destroy the local geometric structure of the original point cloud.

Here, we explore a new way of converting point clouds to a representation suitable for deep learning, without destroying any geometric information. Specifically, we connect neighbouring points in a point cloud to form an undirected graph. Although graphs lack the translational-invariant structure just as point clouds, there has been a line of work [1, 2] that extends CNNs to graphs by defining convolution in the spectral domain. The aim of this project is to investigate the effectiveness of these spectral CNNs on the task of point cloud semantic segmentation.

2 Related Work

Point Cloud Segmentation Most previous works on point cloud segmentation rely on geometric cues. For example, 3D Hough Transform [3] decomposes an indoor scene by detecting planes; Locally Convex Connected Patches [4] and Constrained Planar Cuts [5] detects object boundaries via local convexity; Voxel Cloud Connectivity Segmentation [6] produces “supervoxels" using spatial connectivity. These methods are unsupervised and can group points into geometrically meaningful regions, but the segmentation result does not necessarily carry semantic meanings.

On the other hand, learning based methods such as [7] use semantically labeled point clouds to train a model that assigns a semantic class to each point. Recently, some works [8, 9] start to explore deep neural networks as models for point cloud segmentation.

Deep Learning on 3D Data The pioneering approach in deep learning on 3D data is Volumetric CNN [10, 11], which generalizes 2D CNN by applying 3D convolutions to voxelized data. With careful engineering, these networks can achieve state-of-the-art results in object classification on standard data set such as ModelNet [12]. However, the volumetric representation is constrained by its resolution because of the sparsity of 3D data and computational cost of 3D convolutions.

An alternative to Volumetric CNN is Multi-view CNN [13, 11], which classifies 3D point clouds by applying 2D CNN on multiple rendered images of the point clouds. With efficient 2D CNNs, these networks can operate on data of much higher resolution. However, it is unclear how to propagate things learned by the 2D CNNs back to 3D to perform tasks like point cloud segmentation.

10707 Class Project (2017). Code available at `https://github.com/TonythePlaneswalker/pcd_seg`.

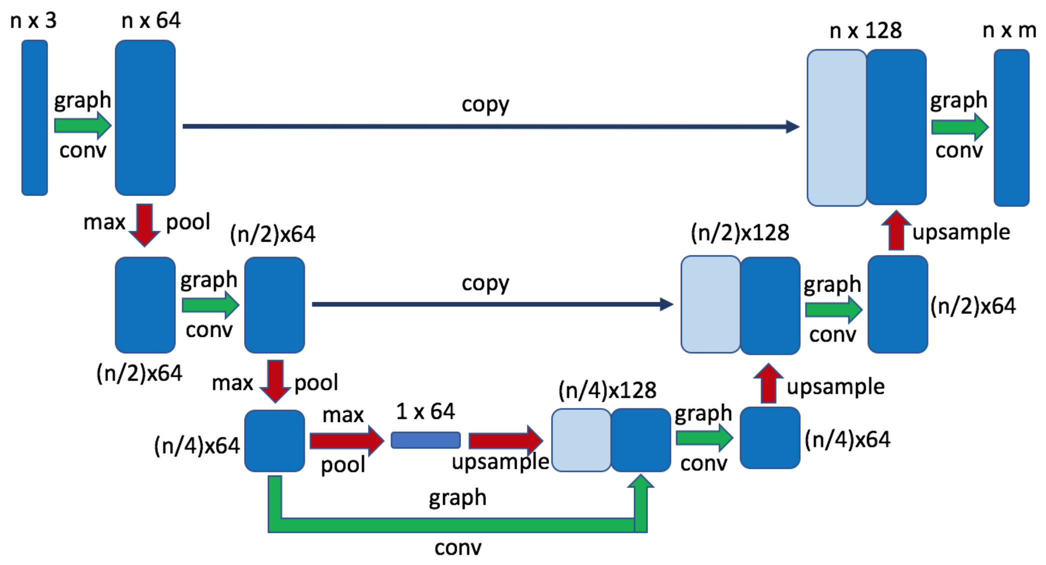


Figure 1: Network architecture with two levels of graph pooling. Low-level features are combined with concatenated high-level features via skip connections. The bottleneck at the bottom ensures each point gets access to some global information about the shape.

More recently, other types of architectures have been proposed. For instance, PointNet [9, 14] is a very recent architecture that operates directly on raw point clouds. The architecture can be applied to point cloud classification as well as segmentation by combining global and local features. Our approach is most similar to [8], which applies graph convolutional networks to neighbourhood graphs constructed from the point clouds.

Graph Convolutional Network Graph Convolutional Network, or Spectral CNN, is first proposed by [1] and extended in [15]. These works use ideas from graph signal processing [16] to define a localized filter on irregular data structure represented as a graph. However, these filters operate in the spectral domain, which is not suited for non-linear activation functions. Consequently, the signal needs to be transformed back and forth between spectral and spatial domains using graph Fourier transform, which can be expensive. Recently, [2, 17] proposed a new type of graph CNN that solves this problem by approximating the spectral convolution with Chebyshev polynomials, which will be introduced in more detail in the next section.

3 Proposed Approach

Given a point cloud represented as a set of points $\{p_i | i = 1, \dots, n\}$, our task is to predict m scores for each of the n points, where each score indicates the probability that the point belongs to a particular semantic part of the object. Here, each point p_i is a vector (x, y, z) of its coordinates. Additional features, such as color, surface normal etc., can also be added to each vector as input.

To transfer the problem on point clouds to a problem on graphs, we first build a weighted undirected graph $G = (V, E, W)$, where we connect each point with its k nearest neighbours. The edge weight matrix W is constructed by setting $W_{ij} = \exp(-\alpha d_{ij}^2)$ if p_i, p_j are connected and 0 otherwise, where d_{ij} is the Euclidean distance between p_i, p_j and α is some constant (e.g. inverse of the variance of distances among neighbouring points).

Then, we train a neural network to predict the $n \times m$ output signal from the $n \times 3$ input signal. The network architecture is shown in Fig. 1. Below, we will explain the key components of the network – graph convolution and pooling.

3.1 Spectral Graph Convolution

Convolution is a localized and translational-invariant operation. On irregular data structure like graphs, the structure of a local neighbourhood varies across different locations, so it is difficult to define a filter that is invariant to spatial translations. Alternatively, convolutions can be defined as linear operators that diagonalize the Fourier basis [18]. To be specific, given two signals x and y, the convolutional operator can be defined as $x * y = U((U^T x) \odot (U^T y))$, where U is the matrix whose columns are the Fourier basis. Thus, we can define a filter f in the spectral domain as a vector of Fourier coefficients, and the resulting signal of x filtered by f is simply $y = U \operatorname{diag}(f) U^T x$.

However, a non-parametric filter like the one above is not localized spatially, and its dimension is equal to the dimension of $U^T x$, the number of basis in U, which can be quite large if we want to keep the high frequency information. Therefore, we wish to find a parametrization that reduces the dimension of f.

First, we introduce an essential operator in spectral graph theory, the graph Laplacian L. We use the normalized definition $L = I - D^{-\frac{1}{2}} W D^{\frac{1}{2}}$, where D is the diagonal degree matrix with $D_{ii} = \sum_j W_{ij}$. The graph Laplacian is diagonalized by the Fourier basis, i.e. $L = U \Lambda U^T$ where $\Lambda = \operatorname{diag}([\lambda_0, \ldots, \lambda_{n-1}])$ is the diagonal matrix of eigenvalues. Note that any linear transformation of L induces the same linear transformation on Λ, and vice versa. Moreover, since U is orthonormal, we have $L^k = U \Lambda^k U^T$. It follows that if $P(L)$ is a polynomial in L, then $P(L) = U P(\Lambda) U^T$.

Now, observe that f can be written as a function of the eigenvalues of L. Thus, we can use a set of basis functions to parametrize f. Specifically, we use the Chebyshev polynomials $T_k(x)$ [19], which forms an orthogonal basis for $L^2([-1,1], d\frac{y}{\sqrt{1-y^2}})$, the space of square integrable functions with respect to the measure $d\frac{y}{\sqrt{1-y^2}}$ defined on $[-1,1]$. The measure for integration is irrelevant to our application, but we do need to normalize the eigenvalues between -1 and 1. This can be done by letting $\tilde{\Lambda} = \frac{2}{\lambda_{max}} \Lambda - I$, and $\tilde{L} = U \tilde{\Lambda} U^T$. Then, a filter can be parametrized as the truncated Cheybshev expansion of order K:

$$f_\theta(\tilde{\Lambda}) = \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda}) \tag{1}$$

It follows that a convolution of x with f can be computed as

$$x * f = U \left(\sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda}) U^T \right) x = \sum_{k=0}^{K} \theta_k (U T_k(\tilde{\Lambda}) U^T) x = \sum_{k=0}^{K} \theta_k T_k(\tilde{L}) x \tag{2}$$

An advantage of this parametrization is the Chebyshev polynomials can be pre-computed by the recurrence relation $T_k(\tilde{L}) = 2\tilde{L} T_{k-1}(\tilde{L}) - T_{k-2}(\tilde{L})$ with $T_0 = I$ and $T_1 = \tilde{L}$. Because $\tilde{L}$ is sparse, this computation only costs $O(K|E|)$ operations. When k is small, the resulting matrices are also sparse. In addition, because the parametrized filter is a K-th order polynomial of the Laplacian, it is K-localized, i.e. it depends only on nodes that are at maximum K steps away [17], which fulfills our desire for a spatially localized filter.

Based on this parametrization, the operation of a graph convolution layer is implemented as

$$Y = \sigma \left(\sum_{k=0}^{K} T_k(\tilde{L}) X \Theta_k \right) \tag{3}$$

where each Θ_k is a $M \times N$ weight matrix that mapas M input channels to N output channels, and σ is a non-linear activation function.

3.2 Pooling of Graph Signals

In addition to convolution, CNNs also use pooling operations to increase the receptive field of filters in order to learn high level features. To enable pooling on graphs, [2] employs a greedy graph coarsening scheme proposed by [20] which approximately divides the number of nodes by half via matching and merging pairs of neighbouring nodes.

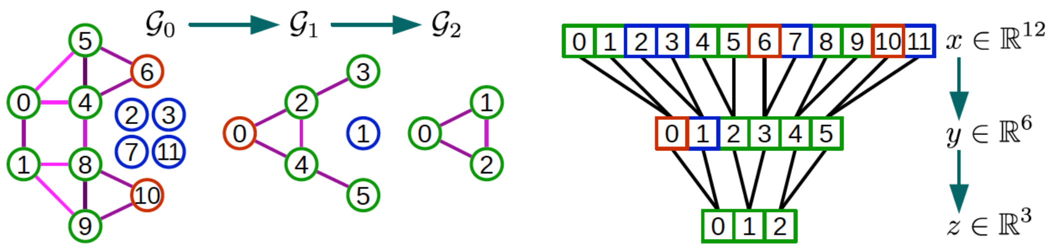


Figure 2: Example of graph coarsening and pooling [2]. The left figure shows two coarsening levels of a graph with 8 vertices. The right figure shows the rearrangement of vertices with added fake nodes, and how a pooling operation is performed. Green boundaries indicate matched vertices. Red boundaries indicate unmatched vertices. And blue boundaries indicate fake nodes.

The graph coarsening scheme naturally induces a binary tree structure on the vertices, as shown in Fig. 2. To deal with singleton nodes, we add a fake node as its sibling. If a fake node is added at level $l > 0$, then we also add two fake nodes as its children at level $l - 1$. By rearranging the vertices in the same order as the leaves of the induced binary tree, a pooling operation on the graph can be implemented as an 1D pooling of stride 2.

4 Data Processing

We use the data from ICCV 2017 part level segmentation challenge to train and test our network. The data set contains about 17,000 models from 16 shape categories in ShapeNet [21]. The exact number of models for each category is shown in Tab. 1. The shapes are represented as point clouds uniformly sampled from 3D surfaces and annotated with 2 to 6 parts as point labels. A 70%/10%/20% training/validation/test split is provided. A detailed break-down of the number of samples is shown in Table.

As input to our graph convolutional network, a k-NN graph with $k = 6$, as well as the coarsened graphs up to level 4, is constructed from each point cloud. Additional fake nodes are added to the graph at each level so that the top level graph has 4096 vertices. The fake nodes all have coordinates $[0, 0, 0]$ and are disconnected from the graph. They are also excluded from loss and accuracy calculations via masking. Fig. 4 shows the coarsened graphs of an airplane model.

We also pre-compute the Chebyshev polynomials $T_k(\tilde{L})$ up to for each of the graphs $k = 3$ to facilitate training. Although the matrices are quite large (4096×4096 for the top level graph), they are very sparse and thus can be stored efficiently.

5 Experiments

5.1 Semantic Part Segmentation

We trained a graph convolutional network on each of the 16 categories, and evaluated the mean intersection-over-union (IOU) over all parts on the provided test split. The results are shown in Tab. 1. A visualization of some samples are shown in Fig. 5.

The network uses 2 levels of graph coarsening and first-order Chebyshev expansion to parametrize filters. It is trained using batch size 16 and learning rate 0.0001 that decays by 0.5 every 10000 steps. Smaller batch sizes are used for categories that do not have enough validation models (bag, cap, earphone and rocket). We did not did not apply any regularization to our model.

5.2 Level of Graph Coarsening

In this experiment, we investigate the effect of the number of graph coarsening levels on the performance of the network. Specifically, we trained our network with 1, 2, 3 and 4 levels of graph coarsening on the airplane category, fixing the order of Chebyshev polynomials used at 1. The results are shown in Fig. 3. As can be seen, more levels of coarsening lead to improvements in mean IOU.

category	# models	# parts	SyncSpecCNN [8]	PointNet [9]	PointNet++ [14]	Ours
mean	1055	-	84.7	83.7	**85.1**	80.3
airplane	2690	4	81.6	**83.4**	82.4	75.8
bag	76	2	**81.7**	78.7	79.0	73.2
cap	55	2	81.9	82.5	**87.7**	72.9
car	898	4	75.2	74.9	**77.3**	69.7
chair	3758	4	90.2	89.6	**90.8**	86.0
earphone	69	3	**74.9**	73.0	71.8	60.6
guitar	787	3	**93.0**	91.5	91.0	88.3
knife	392	2	**86.1**	85.9	85.9	80.4
lamp	1547	4	**84.7**	80.8	83.7	78.8
laptop	451	2	**95.6**	95.3	95.3	94.2
motorbike	202	6	66.7	65.2	**71.6**	64.1
mug	184	2	92.7	93.0	**94.1**	93.2
pistol	283	3	**81.6**	81.2	81.3	80.4
rocket	66	3	**60.6**	57.9	58.7	50.7
skateboard	152	3	**82.9**	72.8	76.4	69.8
table	5271	3	82.1	80.6	**82.6**	78.7

Table 1: Mean IOU of semantic part segmentation on ShapeNet

Is this improvement purely caused by the increase in network depth, or by the graph pooling operations which aggregate local information? To answer this question, we trained another four networks with exactly the same architecture, except that the graph pooling and upsampling operations were replaced by identity. Note that because the filter sizes depend only on the number of input and output channels, this replacement does not change the number of learnable parameters in the network. The comparison is shown in Fig. 3. We can see that networks with graph pooling shows more improvements with increasing depths, while networks without graph pooling struggle do not improve as much and start showing overfitting when they become deeper.

5.3 Order of Chebyshev Polynomials

In this experiment, we investigate how the network's performance is affected by the order of Chebyshev polynomial used to parametrize the filter. Specifically, we trained a 2-level network on the airplane category using 0, 1, 2 and 3 order Chebyshev polynomials. The results are shown in Fig. 3. It can be seen that higher polynomial order, which increases the network's width, does help the network perform better, but the gain is diminishing quickly. Moreover, the coefficient matrices $T_k(\tilde{L})$ become less sparse as k increases. For instance, the percentage of non-zero elements is around 0.1% when $k = 1$, but becomes about 1.1% when $k = 3$. This significantly increases the time and space required to train the network.

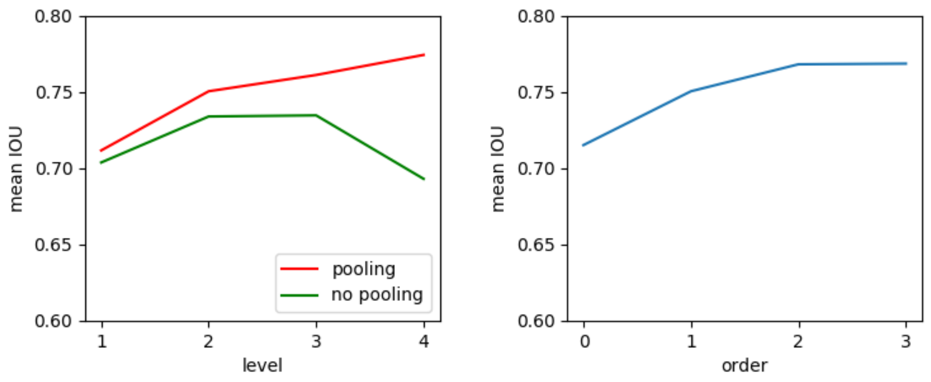
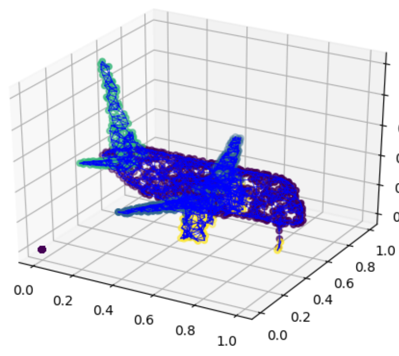
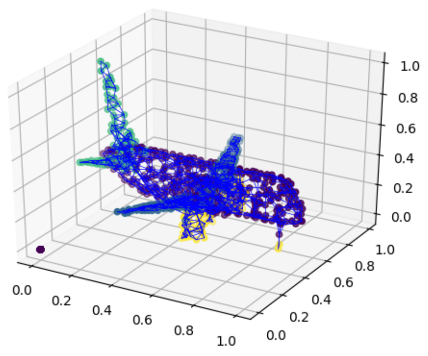
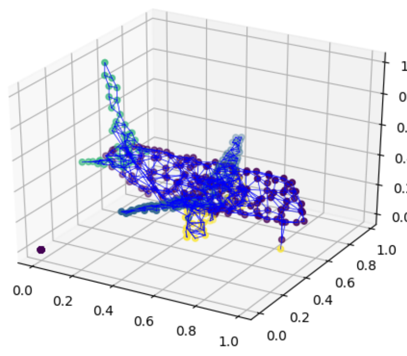


Figure 3: Performance of networks with different coarsening levels and Chebyshev polynomial order


(a) Level 1


(b) Level 2


(c) Level 3

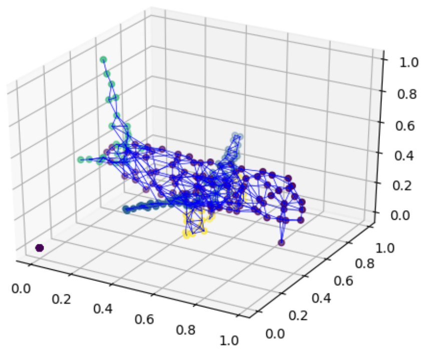

(d) Level 4

Figure 4: Graph representation of an airplane model at different coarsening levels

6 Discussion

Multiplying a graph signal X by a Chebyshev polynomial coefficient $T_k(\tilde{L})$ is equivalent to computing a weighted average of X in the k-neighbourhood around each point p and store the result at p. Therefore, the operations carried out by the graph convolution layer can be viewed as first aggregating information in a local neighbourhood and then applying a point-wise 1D convolution as in [9]. This is in some sense similar to PointNet++ [14]. It is possible to also learn the local aggregation weights, but this will significantly increase the number of parameters (e.g. there are 4096×6 non-zero elements in $T_1(\tilde{L})$ in the bottom level). Some sort of weight sharing may be necessary.

A flaw of our approach is that the graph convolutional network proposed by [2] is designed to process signals on a single, fixed graph. However, in our case each shape is represented by a different graph. Thus, we might need to align the shapes in the spectral domain, as addressed by [8].

Another thing worth noting is the fake nodes introduced by the coarsening algorithm. A fake node added in a high level introduces exponentially many fake nodes in the lower levels. Since they are masked out in the loss calculation, propagating information through them is quite wasteful. It would be nice to have a way to reduce the number of fake nodes and avoid wasting computation on them.

References

[1] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” *arXiv preprint arXiv:1312.6203*, 2013.

[2] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[3] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, “The 3d hough transform for plane detection in point clouds: A review and a new accumulator design,” *3D Research*, vol. 2, no. 2,

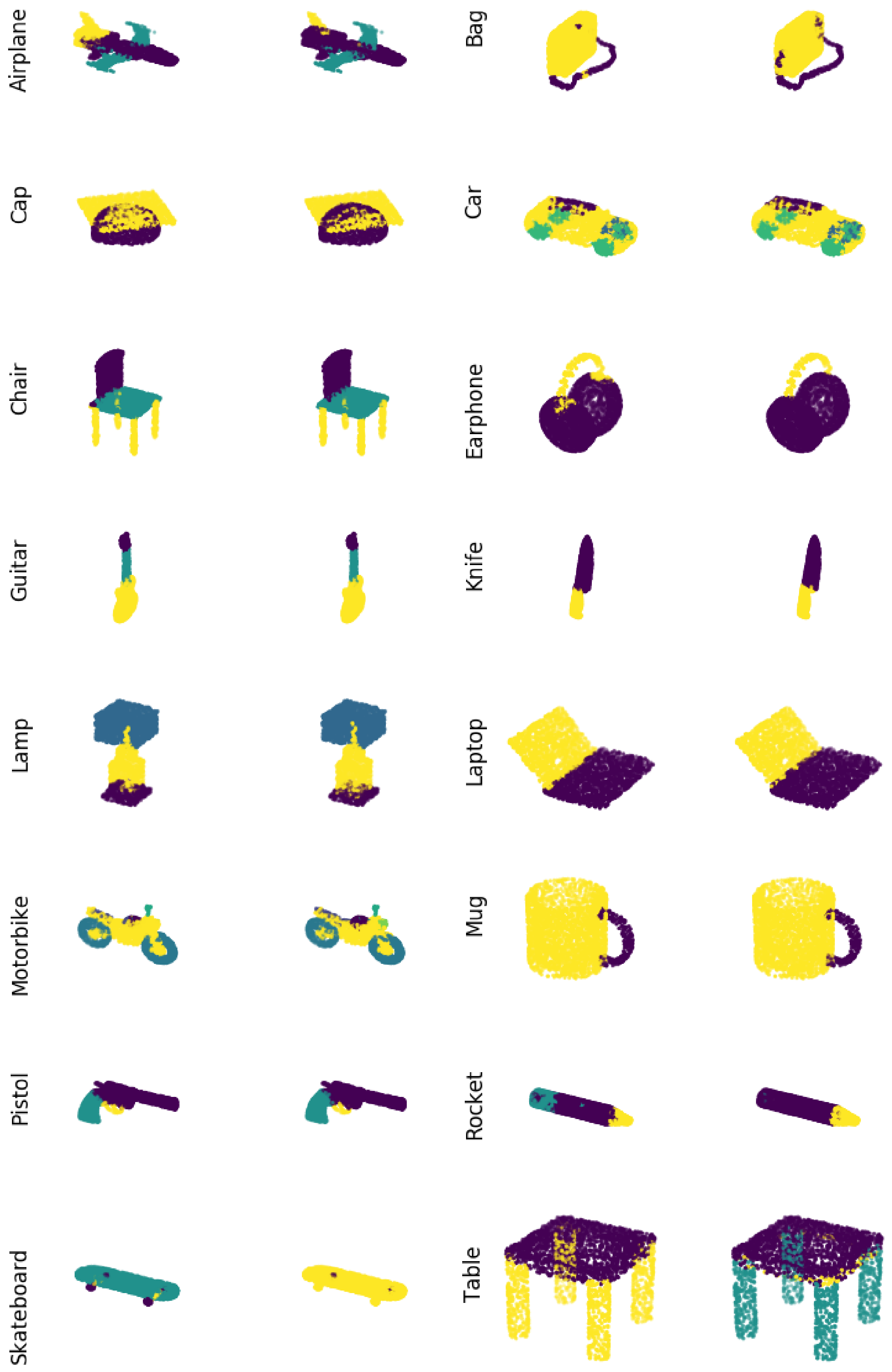


Figure 5: Semantic segmentation samples. The network's predictions are shown on the left and the ground truths are shown on the right.

p. 3, 2011.

[4] S. Christoph Stein, M. Schoeler, J. Papon, and F. Worgotter, "Object partitioning using local convexity," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 304–311.

[5] M. Schoeler, J. Papon, and F. Worgotter, "Constrained planar cuts-object partitioning for point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5207–5215.

[6] J. Papon, A. Abramov, M. Schoeler, and F. Worgotter, "Voxel cloud connectivity segmentation-supervoxels for point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2027–2034.

[7] D. Anguelov, B. Taskarf, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng, "Discriminative learning of markov random fields for segmentation of 3d scan data," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2. IEEE, 2005, pp. 169–176.

[8] L. Yi, H. Su, X. Guo, and L. Guibas, "Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation," *arXiv preprint arXiv:1612.00606*, 2016.

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

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

[11] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5648–5656.

[12] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[13] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[14] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[15] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[16] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[18] S. Mallat, *A wavelet tour of signal processing*. Academic press, 1999.

[19] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[20] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, 2007.

[21] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.