A GRAPH-CNN FOR 3D POINT CLOUD CLASSIFICATION

Yingxue Zhang and Michael Rabbat

McGill University
Montreal, Canada

ABSTRACT

Graph convolutional neural networks (Graph-CNNs) extend traditional CNNs to handle data that is supported on a graph. Major challenges when working with data on graphs are that the support set (the vertices of the graph) do not typically have a natural ordering, and in general, the topology of the graph is not regular (i.e., vertices do not all have the same number of neighbors). In this paper we develop a Graph-CNN for classifying 3D point cloud data which has been obtained from sampling a manifold. We propose an architecture combines localized graph convolutions with two types of graph downsampling operations (also known as pooling). The proposed architecture achieves competitive performance on the 3D object classification benchmark ModelNet, and our architecture is more stable than competing schemes.

Index Terms— Graph convolutional neural networks, graph signal processing, 3D point cloud data, supervised learning

1. INTRODUCTION

With the advent of very large datasets and improved computational capabilities, methods using *convolutional neural networks* (CNNs) now achieve state-of-the-art performance on a variety of tasks, including speech recognition and image classification. Many emerging applications give rise to data that may be viewed as being supported on the vertices of a graph, and field of *graph signal processing* (GSP) has developed filtering and other operations on graph signals [1, 2]. Data may either be naturally sampled on the vertices or edges of a graph (e.g., flows on a transportation network), or the data may simply be unstructured and a graph is imposed to capture the manifold structure underlying the data (e.g., the 3D point clouds considered in this paper). Unlike the domains encountered in more traditional signal processing (e.g., 1D time-series, 2D images), general graph topologies do not have the same regularity or symmetries, and so there is not a unique, well-defined notion of convolution on a graph. This has motivated researchers to develop a variety of approaches to convolutions on graphs, which can then be applied in graph-CNNs and other graph-based signal processing architectures.

Bruna et al. [3, 4] first proposed the idea of using a graph convolution defined in the graph spectral domain together with a graph multiresolution clustering approach to achieve pooling/downsampling. Defferrard et al. [5] propose a fast localized convolution operation by leveraging the recursive form of Chebyshev polynomials to both avoid explicitly calculating the Fourier graph basis and to allow the number of learnable filter coefficients to be independent of the graph size. Atwood and Towsley [6] use a similar localized filtering idea but define the convolution process directly in the spatial domain by searching the receptive filed at different scales using random walk.

Graph kernels have also been widely used to achieve graph classification related task [7, 8] which aims at capturing topological by designing feature extraction scheme on graph instead of learning what features to look for as graph-CNNs. Besides, Graph kernels suffer from quadratic training complexity in the number of graphs [8], which is computational heavy for large scale dataset.

The formulation of Graph-CNNs opens up a range of applications. Defferrard et al.[5] validate their model on an image classification task and demonstrate the effectiveness of Graph-CNNs. Kipf and Welling [9] study the application of the Graph-CNNs to semi-supervised learning. Most application of Graph-CNNs so far focus on cases where the graph structure is homogeneous, and only the graph signal varies.

GSP techniques have also been applied to process 3D point cloud data, such as that obtained by *light detection and ranging* (LiDAR) sensors. Rather than binning point clouds into voxels, graph-based approaches fit a graph with one vertex for each point and edges between nearby points, and then operate on the graph. The effectiveness of GSP for processing 3D point cloud data has been demonstrated in applications such as data visualization, in-painting, and compression [10, 11, 12].

In this work we propose a Graph-CNN architecture for classifying graphs. Unlike most previous Graph-CNNs, in this setting both the signals and the graph structure vary from input to input. The proposed architecture uses existing graph convolution together with graph multi-resolution pooling. The architecture learns a latent signature summarizing each point cloud which is invariant to 3D rotations. We achieve an average classification accuracy comparable to the state-of-the-art on the ModelNet benchmark, and the variance of the proposed approach is substantially lower than existing point-based classification methods.

2. PROBLEM STATEMENT

We consider a classification problem where we are given m labeled training instances $\{(X_j, y_j)\}$, each composed of an input $X_j \in \mathcal{X}$ and an output $y_j \in \mathcal{Y}$. Our goal is produce a function $y = f(X)$ to predict the output y associated with a new, unseen input X. We consider the case where the output space $\mathcal{Y}$ is finite (the classes), and each input X_j is a set of n points, $\{x_{j,1}, \ldots, x_{j,n}\} \subset \mathbb{R}^3$.

Previous work has approached this problem from different perspectives [13, 14, 15, 16, 17], including rendering and processing a collection of 2D images (projections of the points onto an image plane from different perspectives), or binning the points into voxels. We instead take a graph-based approach, fitting a graph and learning a mapping from the space of graphs and its corresponding graph signal to classes.

3. METHODOLOGY

Given a set of points $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^3$, we first fit a (symmetrized) k-nearest neighbor graph to the points and then weight each edge using a Gaussian kernel: for $i, j = 1, \ldots, n$,

$$W_{i,j} = \begin{cases} \exp(-\|x_i - x_j\|^2/\sigma^2) & \text{if } j \in \mathcal{A}_k(i) \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathcal{A}_k(i)$ is the set of k nearest neighbors of vertex i.

We will also use the coordinates as graph signals. Let $x_i = [x_i^{(1)}, x_i^{(2)}, x_i^{(3)}]^T$, and let $\boldsymbol{x}^{(c)} \in \mathbb{R}^n$ denote the vector with ith entry equal to $x_i^{(c)}$. Then $x^{(c)}$ can be seen as a signal on the graph (one value per vertex). The associated coordinate vectors $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)} \in \mathbf{R}^{n \times 3}$ will serve as graph signals.

3.1. Graph Signal Processing

We briefly review notions of graph filtering and convolution before describing the proposed architecture. Given the symmetric weighted adjacency matrix $W \in \mathbb{R}^{n \times n}$ of a graph, let $L = I_n - D^{-1/2} W D^{-1/2}$ denote the normalized Laplacian matrix. A linear vertex-domain graph filter with coefficients $\alpha_0, \ldots, \alpha_K$ transforms one graph signal, $\boldsymbol{x}$, to another, $\boldsymbol{y}$, via $\boldsymbol{y} = h_\alpha(L)\boldsymbol{x} = \sum_{k=0}^{K} \alpha_k L^k \boldsymbol{x}$. It can also be convenient to represent or approximate filters in terms of Chebyshev polynomials of L [18],

$$\boldsymbol{y} = g_\theta(L)\boldsymbol{x} = \sum_{k=0}^{K} \theta_k T_k(L)\boldsymbol{x}, \tag{2}$$

defined recursively via $T_0(L) = I$, $T_1(L) = L$, and for $k \geq 2$,

$$T_k(L) = 2LT_{k-1}(L) - T_{k-2}(L).$$

Graph-CNNs involve multiple such filters where the coefficients $\{\alpha_k\}$ or θ_k are learned from data.

Graph filters have a spectral interpretation, in terms of the eigendecomposition $L = U\Lambda U^T$ of the Laplacian. We have

$$\boldsymbol{y} = h_\alpha(L)\boldsymbol{x} = Uh_\alpha(\Lambda)U^T\boldsymbol{x} = Uh_\alpha(\Lambda)\widehat{\boldsymbol{x}},$$

so the eigenvectors U^T serve as a graph spectral basis, $\widehat{\boldsymbol{x}}$ is the vector of graph spectral coefficients of $\boldsymbol{x}$, and the filter $h_\alpha(\cdot)$ acts entrywise on the eigenvalues Λ (with an identical interpretation possible in terms of g_θ). The eigenvectors of the Laplacian are known to correspond to low- or high-variation over the graph proportional to the corresponding eigenvalue [1].

As an illustration on point cloud data, we apply the filtering and sampling approach described in Chen et al. [19, 10] for designing high-pass and low-pass graph filters. Figure 1 shows the vertices with the largest signal norm after applying high-pass and low-pass filtering operations for three example point clouds. Clearly different types of filters emphasize different structural characteristics. In this work we take the approach of learning different filters in order to obtain features which can be used to classify point clouds.

3.2. Proposed Graph-CNN Architecture

Next we discuss our Graph-CNN architecture for 3D point cloud classification. Similar to typical CNNs and other Graph-CNN architectures, our architecture combines three main types of layers: convolutional, pooling, and fully-connected. Because the convolutional and pooling layers are particular to the graph setting, we describe these in detail.

Convolutional layer. During the training process, our goal is to train a set of graph filter coefficients that can translate the input signal to latent feature maps that capture relevant structure information

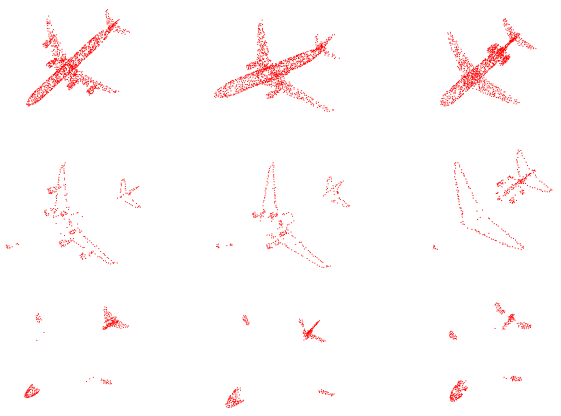


Fig. 1. *Top:* Three point clouds from the airplane class. *Middle:* 300 points with highest l2 norm after Haar-like high-pass graph filtering [19]; *Bottom:* 300 points with highest l2 norm after low-pass graph filtering with first 50 graph frequency components

to discriminate between object classes. Because we will apply the architecture to different graphs, and the Laplacian spectra of different graphs have different ranges, we first perform a normalization: we use the rescaled Laplacian $\tilde{L} = 2L/\lambda_{\max} - I_n$, where $\lambda_{\max}$ is the largest Laplacian eigenvalue, so that all eigenvalues of $\tilde{L}$ are in the interval $[-1, 1]$. We have found that this improves stability of the network during learning.

As illustrated above, each filter of order K has K learnable parameters (the filter coefficients). Previous researchers have investigated the usage of the Chebyshev graph filtering approximation in distributed signal processing [20]. Graph-CNNs (e.g., ChebyNet [5]) and Defferrard et al. [5] demonstrate the effectiveness of this convolution block in homogeneous graph prediction tasks such as image classification, where each images can be regarded as a grid graph (hence, having the same graph structure, assuming the image size is fixed in advance).

We adapt a similar scheme to deal with heterogeneous graphs. Specifically, to obtain one level of feature transformation, we apply Chebyshev polynomial filters (2) and use the coordinate vectors $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(3)}$ as input vectors for first convolution layer. With a layer of m filters, each of order K, and d input signal dimension, we have dmK learnable parameters and mn outputs. After each convolutional layer we apply a rectified linear unit (ReLu) nonlinear activation function. Fig. 2 shows an example with two layers.

Pooling layer. The feature maps output by the convolutional layer are a point-wise latent representation. We implement two forms of pooling operations to aggregate information from these representations. One form of pooling computes global statistics across all output points, while the other form of pooling acts locally within the point cloud, leading to multi-resolution pooling similar to graph coarsening methods employed in previous work [3, 5].

Global pooling. As discussed in Sec. 3.2, the output of graph filters can emphasize meaningful structural information about a point cloud. We seek representations that are invariant to rotations (to deal with the case when different point clouds have not been oriented/registered). To this end we use 1-max-pooling (i.e, taking the max of all filter outputs) and variance pooling. Max pooling highlights the most distinctive points, whereas variance pooling quantifies spread of the outputs afer filtering, and our experiments suggest

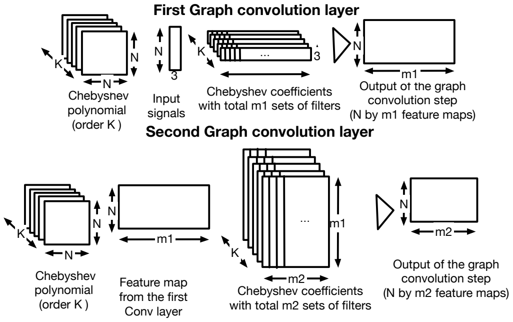

First Graph convolution layer
K
N
N
Chebyshev polynomial (order K)
N
3
Input signals
K
...
3
Chebyshev coefficients with total m1 sets of filters
N
m1
Output ot the graph convolution step (N by m1 feature maps)
Second Graph convolution layer
K
N
N
Chebyshev polynomial (order K)
N
m1
Feature map from the first Conv layer
K
...
m1
m2
Chebyshev coefficients with total m2 sets of filters
N
m2
Output of the graph convolution step (N by m2 feature maps)

Fig. 2. Convolution layer

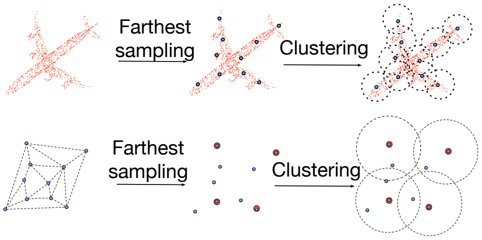


Fig. 3. Multi-resolution pooling in point set

that both provide useful information for classification. When using multiple convolutional layers, we compute these statistics for each layer and concatenate them as inputs to the final layer for computing the final class probabilities.

Multi-resolution pooling. Graph clustering algorithms are computationally demanding, and we seek a light-weight method for local pooling of graph signals. In the special case of point cloud data, we can leverage the geometry inherent to the data to avoid explicitly calculating a graph coarsening or clustering. Instead, we realize multi-resolution pooling by sub-sampling a set of points that are most scattered from each other. This is done by first sampling a random point and adding it to the sampling set. The next sampling point is taken to be the one furthest from the initial sampling point, and subsequent sampling points are those that are mutually furthest from all other sampling points. After reaching the desired number of sampling points (resolution at the next level) we associate each non-sampled point with the nearest sampled point and aggregate among these groups. After the pair-wise distances have been computed once, they can be cached and easily reused at subsequent sampling steps. (Recall that the pair-wise distances are also calculated to form the k-nearest neighbor graph and to calculate the non-zero entries of W.) An example of applying this multi-resolution pooling scheme is shown in Fig. 3.

Final architecture. The final architecture we use in the experiments below is composed of two convolutional layers. Each layer involves graph convolution with order $K = 3$, followed by ReLu activation and one form pooling (we report the results of different combinations below). The first layer has 1000 filters and the second layer has 1200 filters. When using multi-resolution pooling we downsample to 35 points for the second layer where clusters are formed by 50 nearest neighbors of every centroid point. The second pooling layer contains 6 centroid points and each cluster containing 10 nearest neighbors, followed by one final global pooling step. The intermediate representations produced by the convolutional layers are flattened into a vector and passed through a final fully-connected linear layer with softmax activation, where the number of outputs is the number of classes. Thus, the final output corresponds to a vector with entries giving the predicted probability of the input belonging to each of the possible classes.

Training details. The training is done using Adam optimizer with mini-batch training fashion with batch size 28 to achieve fast convergence and memory-efficient. To prevent the model from overfitting, two methods have been used. One is adding dropout after both convolutional layers and fully connected layer with 0.5 dropout rate, and the other one is adding regularization term with $\alpha = 2 \times 10^{-4}$ on the weights to avoid learning complicated model.

4. PERFORMANCE EVALUATION

Dataset description and preparation. We evaluate our algorithm on the ModelNet40 and ModelNet10 datasets for 3D object recognition [13]. ModelNet10 contains 4,899 CAD models from 10 categories, split into 3,991 for training and 908 for testing. ModelNet40 contains 12,311 CAD models from 40 categories, split into 9,843 for training and 2,468 for testing. We use the same data format as PointNet [17], the state-of-the-art point-based 3D object classification approach, where the data are uniformly sampled on the CAD object mesh face to obtain $n = 2048$ points. Both datasets have unbalanced class distribution, which poses a challenge in model training. In ModelNet10 all models are oriented, and in ModelNet40 they are not oriented.

All the point clouds are initially normalized into a unit sphere. To further reduce the size of each object for fast computation, we preprocess the data to 1024 points per object by uniform subsampling. We experimented with other preprocessing schemes, not reported here due to space limitations, such as contour-enhanced subsampling [19], but they didn't lead to any improvement in performance. However, using the contour-enhanced subsampling do lead to better resistance against data corruption, when noise being added to the point clouds.

Performance comparison. 3D data have mainly three types of representations. Different representations of 3D objects lead to different approaches to solve the problem. We compare our algorithm with state-of-art methods using volume as input (i.e., binning into voxels) [13, 14, 15], using images from different views as input [16], and using point sets [17] as input respectively. Since the dataset has class imbalance, we consider two performance metrics, mean instance accuracy and mean category accuracy, to evaluate the performance. A model can be optimized for either metric. To optimize for class accuracy, weighted gradient descent is applied, where the weighting provides uniform sampling across classes at each step. This imposes a greater cost on the model for making classification mistakes on minority classes during training. These extra penalties can bias the model to pay more attention to the classes which are less well-represented.

From the results in Table 1, we can see that there is still a gap between our method and the state-of-arts 3D object classification approach MVCNN [16], which uses ImageNet1K to pre-train and uses an extensive data augmentation scheme. In contrast, we use no pre-training or data augmentation. On ModelNet 40, we slightly improve the performance compared to the state-of-the-art point-based method PointNet [17] in terms of mean instance accuracy.

Fluctuation of the performance during the training (i.e., sensitivity to initial seeds) is one of the difficulties when training on ModelNet40, since it has significant class imbalance. This phenomenon is especially severe for PointNet [17]. Also, as shown in Figure 4 and

Algorithm	Input format	ModelNet 10 Prediction Accuracy (avg. class)	ModelNet 10 Prediction Accuracy (overall)	ModelNet 40 Prediction Accuracy (avg. class)	ModelNet 40 Prediction Accuracy (overall)
3D ShapeNets[13]	volume (1 view)	83.5%	-	77%	84.7%
VoxNet [14]	volume (12 views)	92%	-	83%	85.9%
SSCN [15]	volume (20 views)	-	-	88.2%	-
MVCNN [16]	image (80 views)	-	-	90.1%	-
PointNet* [17]	point (1024 points per object)	91.33%	91.59%	85.40%	88.35%
Proposed algorithm gloabl pooling	point (1024 points per object)	90.86%	91.31%	84.61%	89.18 %
Proposed algorithm gloabl pooling (weighted)	point (1024 points per object)	91.64%	90.81%	85.41%	88.68 %
Proposed algorithm multi-resolution pooling	point (1024 points per object)	90.64%	91.26%	84.46%	89.00 %

Table 1. Results comparison with state-of-art methods on ModelNet [13]. *Means the result is reproduced by the code provided by the paper author. Otherwise, the baseline results presented here is provided by the original paper. All the reproduced or proposed method result reported above is the average results of 50 trails for each scenario.

	Mean instance accuracy std.	Mean class accuracy std.
PointNet	0.2449%	0.4177%
Proposed method (global pooling)	**0.1084%**	**0.1761%**
Proposed method (multi-resolution pooling)	0.2109%	0.3946%

Table 2. Performance standard deviation from 50 trails in PointNet and our proposed methods in ModelNet 40

Table 2, we improve the reliability of the model performance, especially when using the global pooling approach. In our model, since we introduce the structure of the data by providing the local interconnection between points, we narrow the search space, which guarantees more stability performance of the model. The instability of the model performance when using multi-resolution pooling mainly comes from the fact that in every iteration, the centroid points in the pooling step is chosen randomly.

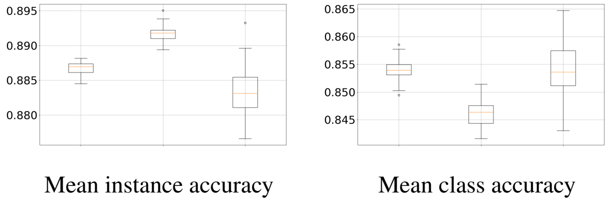
Visualizing the effect of max-pooling. One of the key operations in the proposed architecture is the global operation max pooling, which aims to pick the unique pattern points and summarize the global signature of each object. We call the points that have the maximum values among each feature map the active points. We visualize an example of these active points for feature maps coming from both graph convolutional layers in Fig. 5. Active points at the first layer appear to emphasize local patterns, while those at the second layer emphasize more global structural information. We experimented with using additional graph convolutional layers and didn't find any benefit on the ModelNet benchmarks. In our experiments we also found that using high-pass sampling rather than uniform sampling for preprocessing led to slightly inferior performance. However, it is apparent from these illustrations that the network nevertheless learns some high-pass filters. However, the model learns the contour and keypoint information from data rather than from predefined features, which explains the improved performance.

5. CONCLUSION

In this paper, we propose a Graph-CNN model for 3D point cloud classification. The model has two fast localized graph convolutional layers and point cloud data specific designed pooling layer using global pooling or multi-resolution pooling, our proposed approach is demonstrated to be competitive on the 3D point-cloud classification benchmark dataset ModelNet [13].

The proposed method has a number of interesting properties. First, from the learning complexity perspective by leveraging geometric information encoded in the graph structure we reduce the model complexity (our model has 8.6% fewer learnable parameters than PointNet), which makes the model easier to train and also improves the robustness (as illustrated via the standard deviation of the model performance). Second, one of the biggest problems for point-based classification problem methods is how to achieve invariance to point orientation. In the proposed algorithm, the features we learn are spatially localized by design since filters of order K combine information from K-hop neighbors and rotations preserve nearest neighbors. Moreover, both pooling schemes guarantee order invariance, so the overall architecture preserves this invariance.

Because the proposed approach operates on graphs which are symmetric by design, the resulting filters (defined in terms of the Laplacian) are isotropic and do not capture any notion of directionality along the manifold from which the points were sampled.



Mean instance accuracy

Mean class accuracy

Fig. 4. Detailed comparison with PointNet on ModelNet40 dataset. We fit each model 50 times by running the Adam optimizer 5 times each from 10 different initial seeds, and we examine the resulting accuracies. Within each subplot, the *left box* is for the proposed method with class weighting scheme, the *middle box* is for the proposed method without weighting scheme, and the *right box* is for PointNet.

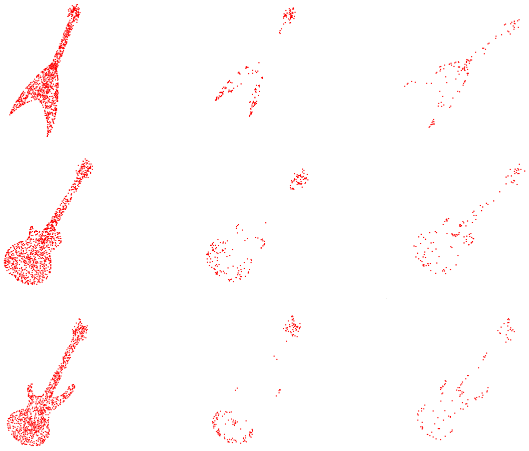


Fig. 5. Max pooling contribution points from different layers (Guitar); *Left:* Original point cloud; *Middle:* Layer one active points; *Right:* Layer two active points

6. REFERENCES

[1] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, 2013.

[2] A. Sandryhaila and J.M.F. Moura, "Big data analysis with signal processing on graphs," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 80–90, 2014.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, "Spectral networks and locally connected networks on graphs," *CoRR*, vol. abs/1312.6203, 2013.

[4] Mikael Henaff, Joan Bruna, and Yann LeCun, "Deep convolutional networks on graph-structured data," *CoRR*, vol. abs/1506.05163, 2015.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016, pp. 3837–3845.

[6] James Atwood and Don Towsley, "Diffusion-convolutional neural networks," in *NIPS*, 2016, pp. 1993–2001.

[7] Risi Kondor and John D. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," in *ICML*. 2002, pp. 315–322, Morgan Kaufmann.

[8] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[9] Thomas N. Kipf and Max Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[10] Siheng Chen, Dong Tian, Chen Feng, Anthony Vetro, and Jelena Kovacevic, "Fast resampling of 3d point clouds via graphs," *CoRR*, vol. abs/1702.06397, 2017.

[11] Francois Lozes, Abderrahim Elmoataz, and Olivier Lezoray, "Pde-based graph signal processing for 3-d color point clouds : Opportunities for cultural herihe arts and found promising," *IEEE Signal Process. Mag.*, vol. 32, no. 4, pp. 103–111, 2015.

[12] Dorina Thanou, Philip A. Chou, and Pascal Frossard, "Graph-based compression of dynamic 3d point cloud sequences," *IEEE Trans. Image Processing*, vol. 25, no. 4, pp. 1765–1778, 2016.

[13] 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, pp. 1912–1920, IEEE Computer Society.

[14] Daniel Maturana and Sebastian Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *IROS*. 2015, pp. 922–928, IEEE.

[15] Benjamin Graham and Laurens van der Maaten, "Submanifold sparse convolutional networks," *CoRR*, vol. abs/1706.01307, 2017.

[16] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *ICCV*. 2015, pp. 945–953, IEEE Computer Society.

[17] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *CoRR*, vol. abs/1612.00593, 2016.

[18] 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.

[19] Siheng Chen, Dong Tian, Chen Feng, Anthony Vetro, and Jelena Kovacevic, "Contour-enhanced resampling of 3d point clouds via graphs," in *ICASSP*. 2017, pp. 2941–2945, IEEE.

[20] D. I Shuman, P. Vandergheynst, and P. Frossard, "Chebyshev polynomial approximation for distributed signal processing," in *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems*, Barcelona, Spain, June 2011.