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). Thus, Graph-CNNs have huge potential to deal with 3D point cloud data which has been obtained from sampling a manifold. In this paper we develop a Graph-CNN for classifying 3D point cloud data, called *PointGCN*[1]. The architecture combines localized graph convolutions with two types of graph downsampling operations (also known as pooling). By the effective exploration of the point cloud local structure using the Graph-CNN, the proposed architecture achieves competitive performance on the 3D object classification benchmark ModelNet, and our architecture is more stable than competing schemes.

支撑集：图中的顶点

将局部图形卷积和
两种下采样的方法
结合起来

***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 applied to the graph classification task [7, 8] which aims to classify a graph based on its topology, as opposed to classifying or otherwise processing signals on a graph. However, Graph kernels suffer from quadratic training complexity in the number of graphs [8].

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. In this paper, we explore the application of the Graph-CNNs in 3D point cloud data.

GSP techniques have 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, 13].

In this work, we propose a Graph-CNN architecture called PointGCN for classifying 3D point cloud data by exploring its local structure encoded in the constructed graph. 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 operation together with two types of specifically designed pooling layers for point cloud data. The architecture learns a latent signature summarizing each point cloud at different receptive fields.

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. For point-based 3D classification problem, 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 taken different approaches to classifying 3D point clouds [14, 15, 16, 17, 18, 19, 20, 21], including rendering and processing a collection of 2D images (projections of the points onto an image plane from different perspectives), or binning the points

[1]Code is available at `https://github.com/maggie0106/Graph-CNN-in-3D-Point-Cloud-Classification`

into voxels. However, the former involves extensive data augmentation, preprocessing and heavy computation, and the latter introduces discretization error as well as use a relatively sparse format to represent the 3D data. Directly using the point cloud format to tackle the 3D classification problem is an area of research interest. PointNet [18] is the pioneer in this field, which uses spatial transformation network and symmetric functions to learn the global representation of each point cloud object. Deep Sets [21] constructs similar universal approximators to achieve invariance to permutation within each point cloud. However, the lack of local structure exploration leads to the loss of local context of each point. The following work, PointNet++ [20], tries to address this problem, but each point is still treated individually in the constructed local clusters. We instead take a graph-based approach, fitting a graph and learning a mapping from the space of graphs and its corresponding graphs signal to classes. By doing so, the local geometric details are encoded in the pairwise distance between center point and its neighbors.

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 $i, j = 1, \dots, n$,

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

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 $\boldsymbol{x}^{(c)}$ can be seen as a signal on the graph (one value per vertex). The associated coordinate vectors $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)} \in \mathbb{R}^{n\times 3}$ 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 $L = I_n - D^{-1/2}WD^{-1/2}$ denote the normalized Laplacian matrix. A linear vertex-domain graph filter with coefficients $\alpha_0, \dots, \alpha_K$ 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 [22],

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

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].

Since different types of filters emphasize different structural characteristics, in this work we take the approach of learning graph filter bank in order to obtain features which can optimize the downstream point cloud classification task.

3.2. Proposed Graph-CNN Architecture for Point Cloud Classification

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. The overall architecture is shown in Figure 1.

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 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 as well as the performance of the network during learning.

As discussed in Sec. 3.1, each filter of order K has K learnable parameters (the filter coefficients), each parameter control the learned latent representation from different receptive field (from 1-hop neighbors to K-hops neighbors). Previous researchers have investigated the usage of the Chebyshev graph filtering approximation in distributed signal processing [23], Graph-CNNs (e.g., ChebyNet [5]). 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 $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}$ as input vectors for first convolution layer.

After each convolutional layer we apply a rectified linear unit (ReLu) nonlinear activation function. One of the advantages of the Chebyshev polynomial filtering is the learned feature maps will be localized within K-hops neighbors of each point. Besides, every time we add one more graph convolutional layer, the receptive field is enlarge by K hops.

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 point order permutation and 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 (i.e, calculating the variance of each filter output across the n points).

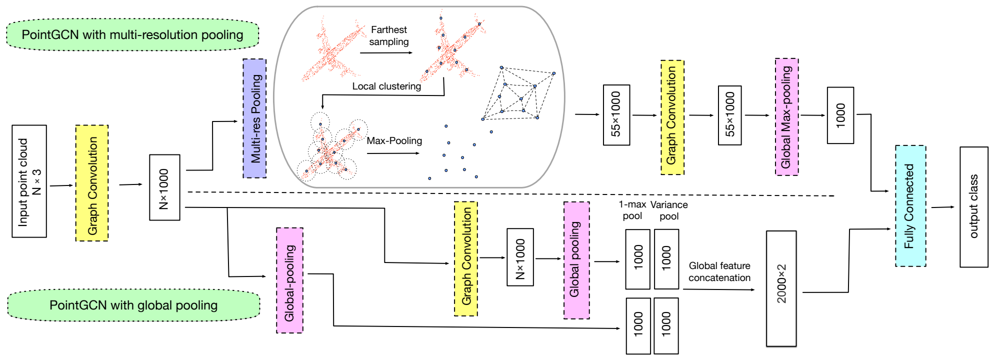

Fig. 1. Overall architecture for graph-CNN based point cloud classification model PointGCN. The top branch is the model architecture using multi-resolution pooling and the bottom branch is the model architecture using merely the global pooling layer.

Max pooling highlights the most distinctive points, whereas variance pooling quantifies spread of the outputs after filtering. 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 by max-pooling.

Final architecture. The final architecture we use in the experiments below is composed of two graph convolutional layers. We construct a 40-nearest neighbor graph for each point cloud object to perform graph convolution. 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). Both first and second convolutional layer has 1000 filters. When using multi-resolution pooling we downsample to 55 points for the second layer where clusters are formed by 50 nearest neighbors of every centroid point. The second convolutional layer is 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 (batch size of 28). To prevent the model from overfitting, two methods have been used. One is adding dropout after both convolutional layers and fully connected layer with 0.9 and 0.5 dropout rate respectively, and the other one is adding a ℓ_2 regularization term (so-called weight decay) with coefficient 2×10^{-4} 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 [14]. 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 [18], 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 farthest subsampling. We experimented with other preprocessing schemes, not reported here due to space limitations, such as contour-enhanced subsampling [10], but they didn't lead to any improvement in performance. However, using the contour-enhanced subsampling do lead to better resistance against data corruption.

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-the-art methods using volume as input (i.e., binning into voxels) [14, 15, 16], using images from different views as input [17], and using point sets [18, 19, 21] 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. To address the class imbalance issue, we use the standard approach of weighting the loss associated with each training instance inversely proportional to the frequency that the corresponding class of the training instance appears in the training set, penalizing mistakes on less common classes more heavily than on more common classes.

The results are shown in Table 1, where we can see that there is still a gap between our method and the state-of-the-arts 3D object classification approach MVCNN [17], which uses ImageNet1K to pre-train and uses an extensive data augmentation scheme (observe 3D objects from 80 views) to cope with the data where the orientation is not aligned. In contrast, we use no pre-training and only use input data from one view. Comparing to other point-based methods, on ModelNet 10 and ModelNet 40, we improve the performance compared to PointNet [18] and ECC [19] in both performance met-

Algorithm	Input format	ModelNet 10 Accuracy (avg. class)	ModelNet 10 Accuracy (overall)	ModelNet 40 Accuracy (avg. class)	ModelNet 40 Accuracy (overall)
3D ShapeNets	volume (1 view)	83.5%	-	77%	84.7%
VoxNet	volume (12 views)	92.0%	-	83%	85.9%
SSCN	volume (20 views)	-	-	88.2%	-
MVCNN	image (80 views)	-	-	**90.1%**	-
ECC	1024 points (12 views)	90.0%	90.8%	83.2%	87.4 ± 0.40%
PointNet	1024 points (1 view)	91.53%	91.74%	85.35%	88.37±0.33%
PointGCN (global pooling)	1024 points (1 view)	91.39%	91.77%	86.19%	89.27±0.20%
PointGCN (multi-resolution pooling)	1024 points (1 view)	91.57%	91.91%	86.05%	89.51±0.23%
Deep Sets	1000 points (1 view)	-	-	-	87 ± 1%
Deep Sets	5000 points (1 view)	-	-	-	**90 ± 0.3%**

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

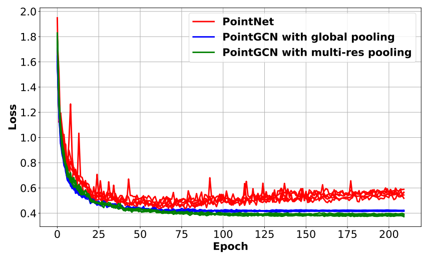

Fig. 2. Test set loss comparison for PointGCN and PointNet from five complete training process

rics.

Fluctuation of the performance during the training (i.e., sensitivity to initial weights) is one of the difficulties when training on ModelNet40. This phenomenon is especially severe for PointNet [18], demonstrated by the learning curve in Figure 2 and the model stability in Figure 3. Our proposed method has relatively faster convergence rate as well as improves the reliability of the model performance for both pooling approaches. One possible reason could be that PointNet only seek the latent representation in a global fashion and lack of exploration of local point features. And in our model, since we introduce the structure of the data by providing the local interconnection between points and explore graph features from different abstraction levels by the localized graph convolutional layers, it guarantees the exploration of more distinctive latent representations for each object class.

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. 4. Active points at the first layer appear to emphasize local patterns, while those at the second layer emphasize more global structural information, which is consistent with our assumption that the K-localized graph convolutional layer can explore features from different receptive fields.

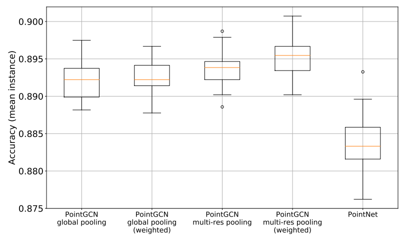

Fig. 3. Model stability comparison with PointNet and PointGCN from 50 trials (present results of our proposed model with and without weighting scheme for fair comparison).

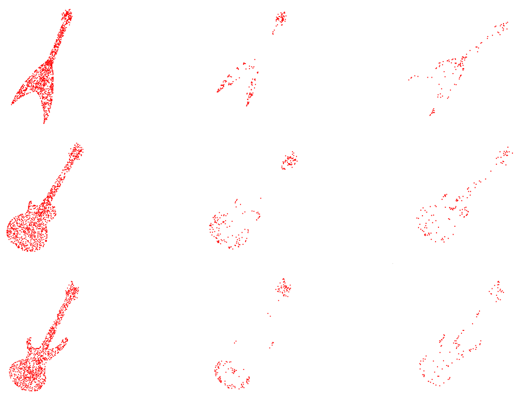

Fig. 4. Max pooling contribution points from different layers (Guitar); *Left:* Original point cloud; *Middle:* Layer one active points; *Right:* Layer two active points

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 [14].

The proposed method has a number of interesting properties. First, by leveraging geometric information encoded in the graph structure, we narrow the search space for the learned model, which makes the model converge faster 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 point order invariant. 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 order permutation preserves nearest neighbors, which are encoded in the constructed graph. At last, 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. This property guarantees that the learned model is robust under rotation transformation, which explains that our proposed model perform well in ModelNet 40 (objects orientation are not aligned) without providing the input point cloud data from different views.

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] Aliaksei Sandryhaila and José 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," in *Int. Conf. Learning Representations (ICLR)*, Banff, Canada, 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 *Advances in Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2016.

[6] James Atwood and Don Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2016.

[7] Risi Kondor and John D. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," in *Int. Conf. on Machine Learning (ICML)*, Sydney, Australia, 2002.

[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," in *Int. Conf. Learning Representations (ICLR)*, Toulon, France, 2017.

[10] Siheng Chen, Dong Tian, Chen Feng, Anthony Vetro, and Jelena Kovačević, "Contour-enhanced resampling of 3d point clouds via graphs," in *Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, United States, 2017.

[11] 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.

[12] 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.

[13] 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.

[14] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Boston, United States, 2015.

[15] Daniel Maturana and Sebastian Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Int. Conf. on Intelligent Robots (IROS)*, Hamburg, Germany, 2015.

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

[17] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Int. Conf. on Computer Vision (ICCV)*, Santiago, Chile, 2015.

[18] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Hawaii, United States, 2017.

[19] Martin Simonovsky and Nikos Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Hawaii, United States, 2017.

[20] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2017.

[21] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan R. Salakhutdinov, and Alexander J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2017.

[22] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[23] David I. Shuman, Pierre Vandergheynst, and Pascal Frossard, "Chebyshev polynomial approximation for distributed signal processing," in *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Barcelona, Spain, 2011.