TOWARDS 3D CONVOLUTIONAL NEURAL NETWORKS WITH MESHES

Miguel Dominguez, Felipe Petroski Such, Shagan Sah, and Raymond Ptucha

Rochester Institute of Technology
Machine Intelligence Laboratory
1 Lomb Memorial Drive, Rochester, USA

ABSTRACT

Voxels are an effective approach to 3D mesh and point cloud classification because they build upon mature Convolutional Neural Network concepts. We show however that their cubic increase in dimensionality is unsuitable for more challenging problems such as object detection in a complex point cloud scene. We observe that 3D meshes are analogous to graph data and can thus be treated with graph signal processing techniques. We propose a Graph Convolutional Neural Network (Graph-CNN), which enables mesh data to be represented exactly (not approximately as with voxels) with quadratic growth as the number of vertices increases. We apply Graph-CNN to the ModelNet10 classification dataset and demonstrate improved results over a previous graph convolution method.

Index Terms— deep learning, convolutional neural networks, graph convolution, voxels

1. INTRODUCTION

Convolutional Neural Networks (CNNs) have revolutionized the computer vision and pattern recognition community by jointly learning the feature extractor and classifier simultaneously. This approach has led to record-breaking performance in problems such as image recognition [1,2] and object detection [3,4]. These problems all depend on the ability of data to be modeled by cascades of Finite Impulse Response (FIR) filters. Unstructured data such as 3D point clouds, which cannot be represented as discrete gridded structures, cannot be filtered using the traditional CNN paradigm.

One approach to classify 3D data is to first convert it to voxels [5–7], then make use of existing CNN theory by switching from 2D to 3D convolution filters. The downsides of this are twofold: First, point clouds and 3D meshes do not convert well to dense 3D representations because they only encode surface information. This means that a 3D voxelized volume would mostly contain zero values. Second, the rapid $O(n^3)$ increase in dimensionality necessitates aggressive upfront downsampling of the data, operating on scales such as $32 \times 32 \times 32$ [5]. These factors limit the ability of these methods to generalize to larger or more complex 3D scenes such as those in the RueMonge [8] and Paris-rue-Madame [9] datasets.

We propose that inference on 3D meshes and point clouds can benefit from a convolution operation that does not assume a dense, gridded structure. 3D Meshes, which are ultimately a collection of vertices and edges, may more accurately be posed as graph structures. Several studies have proposed graph convolution operations [10–14]. Some methods, based on spectral graph theory, require a single graph structure. Other methods require careful preprocessing of the graph data. We introduce an improved variant called “Graph-CNN” that is built from graph signal processing theory [15] and enables flexible graph structures. We perform classification on two datasets: the Bosphorus point cloud dataset [16] and the ModelNet10 mesh dataset [7]. Figure 1 shows the proposed pipeline, which is valid for both mesh and point cloud datasets.

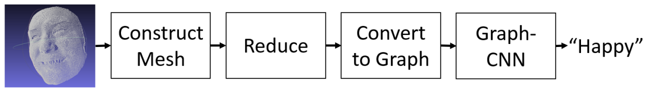

Fig. 1. The architecture proposed in this work. Point clouds are passed through preprocessing and a Graph-CNN architecture to classify the sample.

Section 2 summarizes the literature surrounding graph and 3D mesh classification with CNNs. Section 3 introduces Graph-CNN and our preprocessing techniques. Section 4 applies our method to two classification problems and compares against the literature. Section 5 is the conclusion.

2. RELATED WORK

2.1. Voxel Representations

Voxel based methods have been extremely effective in the ModelNet classification problem [5–7]. This is due to the well-understood feature extraction capability of learned convolutions. However, due to the dramatic increase in dimensionality that would be required, this approach does not scale well to more difficult point cloud data. The $O(n^3)$ memory requirements per sample are acceptable when $n = 32$, but the memory requirements grow much more quickly than the visual quality. Figure 2 illustrates this by comparing the relative quality of different representations of two samples from the ModelNet10 dataset.

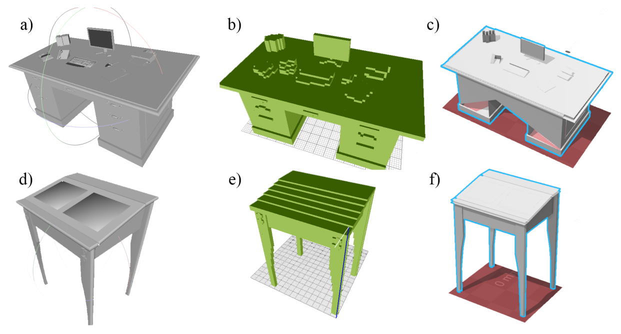

Fig. 2. Two samples from ModelNet10. Model **a)**, **b)**, and **c)** represent the same single sample, but in different representations. Likewise **d)**, **e)**, and **f)** are a single sample. Models **a)** and **d)** are original mesh samples from ModelNet10. Models **b)** and **e)** are $80 \times 80 \times 80$ voxel representations. Models **c)** and **f)** are lower-dimensional mesh representations. The camera angles are slightly different for each image. Voxel samples were created using Patrick Min's binvox program based on [17].

The top row of Figure 2 shows that even a relatively "high-dimensional" voxel representation in **b)** (512,000 total voxels) loses enough information about the objects on the table so that a human would be unable to tell what those objects are. The voxel representation of a night stand in **e)** adds misleading stair-step features to the diagonally-sloped surface of the model in **d)**. For a task like ModelNet, where each sample is clean and visually distinct, this generally does not matter. For recent point cloud datasets such as Paris-Rue-Madame [9] and RueMonge [8], where the data is noisy, large, and complex, voxel approximation would be impractical.

One reason voxels require such high dimensionality for real world applications is that naive voxels do not take into account the natural sparsity in 3D data. Sensors can only measure the surface of an object. Whether an object is stored as a point cloud, a mesh, or voxels, the entire interior of the object will likely be unknown and provide no useful information for any inferencing task. A significant amount of memory and convolution compute cost is spent on meaningless interior data. Octnet [18] approaches this by encoding voxels as octtrees, then encoding information at varying degrees of quality to mitigate redundancy. We propose using the structure of a mesh, which for hollow shapes does not encode information about the interior.

2.2. Graph Representations

3D meshes can be represented by vertices and edges. With this approach, we can take advantage of advances in graph signal processing to attempt a graph convolutional neural network that makes use of these advances. Graph data is often described by the tuple in (1).

$$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{A}) \tag{1}$$

where $\boldsymbol{V} \in \mathbb{R}^{N \times f}$ is a matrix describing N vertices with f features each. $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is an adjacency matrix, which encodes the connections between the vertices in a graph. Each entry in $\boldsymbol{A}$ can be described by (2).

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge between vertex } i \text{ and } j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The binary 1 in (2) can be replaced with any value, including differing edge weights for each edge. There is also no symmetry requirement (it is possible that $a_{ij} \neq a_{ji}$). For our work, we will assume the definition given in (2).

Storing each $\boldsymbol{A}$ costs $O(N^2)$, and each $\boldsymbol{V}$ costs $O(fN)$, where N is the number of vertices and f is a fixed constant indicating the number of features. The total memory cost of $O(N^2 + fN)$ is an asymptotic improvement over the voxels' cost of $O(D^3)$ (where D is the number of voxels along each axis). In all likelihood $N > D$, but if D is large enough, the savings will eventually be realized. This is a conservative estimate as well: Sparse $\boldsymbol{A}$ can be encoded much more efficiently than $O(N^2)$ (e.g. using linked lists). With this representation, we will define a convolution on $\boldsymbol{G}$.

Some approaches at graph convolutions rely on spectral graph theory [10–12]. These methods firstly define the Graph Laplacian for any given graph. This structure is a unitary matrix of eigenvectors $\boldsymbol{U}$ that can serve as an analogue to the Discrete Fourier Transform (DFT). This Eigenbasis encodes the structure of the graph. Hence a "graph signal" $\boldsymbol{x}$, that encodes only the vertex data, can be transformed into the spectral domain. As in classical DSP, convolutions can be performed by multiplying the signal by filter coefficients $\boldsymbol{h}$. The shortcoming of this approach is that it requires each graph to have the same structure, i.e., the structure is homogeneous. The solved Eigenbasis does not effectively model any other graph. This is insufficiently flexible for 3D meshes where two meshes will likely not share a graph representation.

Other recent approaches have abandoned spectral graph theory in favor of defining a spatial-domain convolution operation. PATCHY-SAN [13] linearizes a graph and subsequently learns filters, but the linearization step loses geometric knowledge inherent in the point cloud data. Diffusion-Convolutional Neural Networks (DCNNs) [14] maintain comparable tensor representations of graph data. DCNNs create several matrices modeling different numbers of vertex hops from a reference node. Creating multiple representations of our graph data is useful in our method below.

3. METHOD

3.1. Graph-CNNs

Our method, Graph-CNNs, is inspired by Sandryhaila and Moura's spatial filters on graph signals [15] and is discussed further in [19]. In their work, filter taps are defined as scalar parameters on a polynomial of $\boldsymbol{A}$ as in (3).

$$\boldsymbol{H} = h_0\boldsymbol{I} + h_1\boldsymbol{A} + h_2\boldsymbol{A}^2 \dots h_n\boldsymbol{A}^k, \boldsymbol{H} \in \mathbb{R}^{N\times N} \quad (3)$$

The resulting $\boldsymbol{H}$ matrix is multiplied with a "graph signal", which are values of all the vertices $\boldsymbol{V} \in \mathbb{R}^N$, resulting in $\boldsymbol{V}_{filtered} = \boldsymbol{HV}$.

We adjust this model for our own purposes in three ways. First, we only use the first two taps of $\boldsymbol{H}$ for any given filter, eschewing the exponentiated $\boldsymbol{A}$. This is done because a cascade of these filters in a neural network can approximate the exponentiation. Second, we allow a graph to be represented by ℓ adjacency matrices instead of one. This enables a network to learn more parameters from the same sample and to have different filter taps emphasize different aspects of the data. Finally, we allow a graph signal to have f channels or features, as in the definition in (1). As such, the total input data for a $C-$filter convolution layer is represented in (4).

$$\boldsymbol{V}_{in} \in \mathbb{R}^{N\times f}, \boldsymbol{\mathcal{A}} \in \mathbb{R}^{N\times N\times \ell}, \boldsymbol{\mathcal{H}} \in \mathbb{R}^{\ell\times f\times C}, \boldsymbol{b} \in \mathbb{R}^C \quad (4)$$

$\boldsymbol{V}_{in}$ is an input graph signal, $\boldsymbol{\mathcal{A}}$ is a tensor representing all ℓ of the adjacency matrices for a particular sample, $\boldsymbol{\mathcal{H}}$ is the total graph filter tensor, and $\boldsymbol{b}$ is the bias. The filtering operation is shown in tensor notation [20] in (5) and (6).

$$\boldsymbol{\Gamma} = \boldsymbol{\mathcal{A}} \times_1 \boldsymbol{V}_{in}^T, \boldsymbol{\Gamma} \in \mathbb{R}^{f\times N\times \ell} \quad (5)$$

$$\boldsymbol{V}_{out} = \boldsymbol{\Gamma}_{(2)}\boldsymbol{\mathcal{H}}_{(3)}^T + \boldsymbol{b}, \boldsymbol{V}_{out} \in \mathbb{R}^{N\times C} \quad (6)$$

Like other CNNs, this operation can be learned through back-propagation and it is compatible with methods such as ReLU and batch normalization.

There are two issues we must address. First, we do not propose a pooling method in this work. We believe however that the small increases in receptive field size through cascaded convolution layers will be enough for effective inference. The second detail is that each sample can be a different size. Fully-connected layers expect fixed inputs. To address this we learn one more N'-filter convolutional layer. The output of this layer is passed through a sigmoid to produce an embedding matrix $\boldsymbol{V}_{em} \in \mathbb{R}^{N\times N'}$ that will scale each sample to a fixed size before classification.

$$\boldsymbol{V}_{fixed} = \boldsymbol{V}_{em}^T\boldsymbol{V}, \boldsymbol{A}_{fixed} = \boldsymbol{V}_{em}^T\boldsymbol{A}\boldsymbol{V}_{em} \quad (7)$$

Equation (7) illustrates a single adjacency matrix, though the operation is applied to all ℓ adjacency matrices in the stacked tensor $\boldsymbol{\mathcal{A}}$.

3.2. Preprocessing

Meshlab [21] is a flexible tool used to preprocess point cloud and 3D mesh data. We use it to create high-quality meshes from point cloud data and to scale high-quality meshes to an appropriate dimensionality. Large meshes can encourage overfitting since more vertices mean more features.

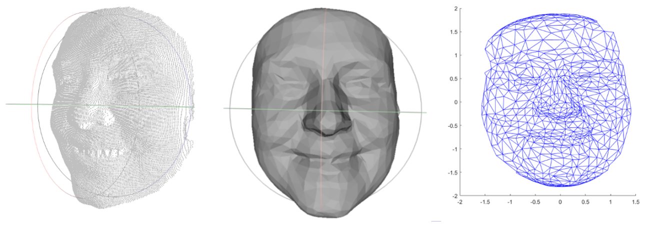

Fig. 3. Different stages of Bosphorus face preprocessing. **Left:** Original point cloud. **Center:**3D mesh. **Right:** 2D projection of the front of 3D mesh (optional).

To feed this data into a Graph-CNN, we create $\boldsymbol{V} \in \mathbb{R}^{N\times 3}$, where the features are the cartesian coordinates of each vertex. We create eight adjacency matrices for each mesh. The edges are partitioned by their geometric orientation. Each edge can be posed as the vector created by subtracting two 3D vertices. Each vector has X, Y, and Z components, each with a sign (positive or negative). One adjacency matrix is defined as the partition of edges such that $(X \geq 0, Y \geq 0, Z \geq 0)$, another is defined as the partition of edges such that $(X \geq 0, Y \geq 0, Z \leq 0)$, and so on for all eight combinations. This increases the number of parameters in the Graph-CNN model and encodes an understanding of angle. The adjacency matrices are assumed to be symmetric.

4. RESULTS

We defined the Graph-CNN networks as cascades of Graph Convolution$\rightarrow$ Batch Normalization$\rightarrow$ ReLU, ending in a single fully connected layer with Softmax loss. We trained the model with momentum and ℓ_2 normalization. The hyperparameters for each experiment are discussed in the results. In each of our investigations, our networks are listed in the form $d\times$GraphConv$C$, where d is the number of layers and C is the number of filters per layer.

4.1. Bosphorus

Bosphorus [16] is a dataset of point clouds of human faces. 453 samples are labeled with one of six facial expressions: Anger, Disgust, Fear, Happiness, Sadness, and Surprise. We convert this data into a 3D mesh and use Graph-CNNs for facial expression classification. The dataset is extremely small, so we apply aggressive dimensionality reduction to discourage overfitting. This includes reducing the mesh with Meshlab, cutting out vertices in the rear of the head, and projecting the data onto a 2D plane. as Figure 3 shows, the face is still discernable in 2D.

To train the networks, we used a minibatch size of 90 samples, an initial learning rate of 0.01, and momentum with a parameter of 0.9. The learning rate is decreased by a factor of 10 after every 100 iterations. We apply an ℓ_2 regularization parameter of 0.025. There is no pooling operation, simply cascaded convolutions and activations that end in a fully-connected layer.

We compare the results of different numbers of layers in Table 1. Results show our Graph-CNN networks can infer facial expression from these graph convolutions. As a quick comparison, we also created a similarly-sized 5-layer traditional CNN that takes the Bosphorus 2D color images and alternates convolution and max pooling (CNN+Images). The increased CNN accuracy is likely due to both increased information (for example, there is no color, light, or shadow in the mesh data) and the increased receptive field due to pooling.

Table 1. Bosphorus results, 5-fold cross validation.

Architecture	Accuracy	# Parameters
3× GraphConv16	54.8	8016
4× GraphConv16	70.0	10336
5× GraphConv16	67.9	12656
CNN+Images	**82.2%**	8032

4.2. ModelNet10

ModelNet [7] is a dataset of 3D meshes, separated into 10-class and 40-class sets and used for classification and object retrieval problems. Each of the classes in ModelNet is visually distinct from the others. Even a relatively low-dimensionality signal can represent the class well. This bears out in the literature, where $32 \times 32 \times 32$ voxels get up to 97.1% accuracy [5], with data augmentation and ensembles. We simplify each mesh using Meshlab to 500 faces (which translates to an average of 115×115 adjacency matrices in the entire dataset). We do not project the data onto a 2D plane as in Bosphorus.

We also apply data augmentation during training. Initially we rotated the samples in space as in [22], but found that it did not reasonably model the data. All of the samples in the dataset are facing the same direction. We observe that one of the major differences in examples are relative lengths. To that end, we apply stretching in ratios of $1:1$, $1.5:1$, and $2:1$ in the X, Y, and Z, axes, resulting in 27 total samples per sample from the original dataset.

We train using the same initial learning rate and momentum as the Bosphorus evaluation. The training is done for 6000 iterations, stepping down the learning rate by a factor of 10 every 2000 iterations (though the curves suggest the step down could be done in fewer iterations). Instead of a 5-fold cross-validation, we tuned hyperparameters on the training set by splitting it into roughly 75% training and 25% validation. The best network has an ℓ_2 regularization parameter of 0.05. The final network was trained on the entire training set and tested on the test set.

Table 2. ModelNet10 classification results.

Architecture	Accuracy
VRN Ensemble [5]	**97.14%**
ORION [6]	93.8%
4× GraphConv24 (ours)	74.3%
Ravanbakhsh, et al. Graph Method* [22]	58.0%

*Only has results on ModelNet40, not ModelNet10

The results from a selection of models are compared in Table 2. Graph-CNN does not get state of the art results compared to the best ensemble (VRN) nor the best single model (ORION), but we are highly encouraged that Graph-CNN can further mesh and point cloud understanding. First, we are competitive against a competing Graph-Convolution method that achieved 58% on ModelNet40. This is not a direct comparison with ModelNet10 as ModelNet40 scores understandably tend to be lower, but the disparity between ModelNet10 and ModelNet40 scores for a given model in the literature tend to be relatively close. Second, we are pushing forward with a fundamentally new method of convolution while the best-performing models are using mature techniques on visually distinct classes.

5. CONCLUSION

We show that voxel-based CNNs, while effective at simple 3D image classification, have difficulty scaling to larger data. Dimensionality grows quickly without a commensurate increase in quality. Graph convolutions provide potential for asymptotically more efficient 3D inference. We believe our Graph-CNN method of 3D point cloud and mesh processing will begin a race for efficient, effective geometric convolutions. As datasets get more complex, the more we will need analysis techniques such as Graph-CNN that can learn effectively from non-gridded data.

6. ACKNOWLEDGEMENTS

We would like to thank the National GEM Consortium for funding this research through the GEM Fellowship program.

7. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," *arXiv preprint arXiv:1506.02640*, 2015.

[5] Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston, "Generative and discriminative voxel modeling with convolutional neural networks," in *ArXiv e-prints*, vol. 1608.

[6] Nima Sedaghat Alvar, Mohammadreza Zolfaghari, and Thomas Brox, "Orientation-boosted voxel nets for 3d object recognition," *CoRR*, vol. abs/1604.03351, 2016.

[7] Wu Zhirong, S. Song, A. Khosla, Yu Fisher, Zhang Linguang, Tang Xiaoou, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1912–1920.

[8] Hayko Riemenschneider, András Bódis-Szomorú, Julien Weissenberg, and Luc Van Gool, *Learning Where to Classify in Multi-view Semantic Segmentation*, pp. 516–532, Springer International Publishing, Cham, 2014.

[9] Andres Serna, Beatriz Marcotegui, Francois Goulette, and Jean-Emmanuel Deschaud, "Paris-rue-madame database: a 3d mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods," in *3rd International Conference on Pattern Recognition, Applications and Method*.

[10] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[11] Oren Rippel, Jasper Snoek, and Ryan P Adams, "Spectral representations for convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2440–2448.

[12] Mikael Henaff, Joan Bruna, and Yann LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[13] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov, "Learning convolutional neural networks for graphs," in *Proceeding of the 33rd International Conference on Machine Learning*, 2016, pp. 2014–2023.

[14] James Atwood and Don Towsley, "Diffusion-convolutional neural networks," *arXiv preprint arXiv:1511.02136*, 2015.

[15] Aliaksei Sandryhaila and José MF Moura, "Discrete signal processing on graphs," *Signal Processing, IEEE Transactions on*, vol. 61, no. 7, pp. 1644–1656, 2013.

[16] Arman Savran, Neşe Alyüz, Hamdi Dibeklioğlu, Oya Çeliktutan, Berk Gökberk, Bülent Sankur, and Lale Akarun, "Biometrics and identity management," chapter Bosphorus Database for 3D Face Analysis, pp. 47–56. Springer-Verlag, Berlin, Heidelberg, 2008.

[17] F. S. Nooruddin and G. Turk, "Simplification and repair of polygonal models using volumetric techniques," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 191–205, April 2003.

[18] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger, "Octnet: Learning deep 3d representations at high resolutions," in *ArXiv e-prints*, vol. 1611.

[19] Felipe Petroski Such, Shagan Sah, Miguel Domínguez, Suhas Pillai, Chao Zhang, Andrew Michael, Nathan D. Cahill, and Raymond Ptucha, "Robust spatial filtering with graph convolutional neural networks," *arXiv preprint arXiv:1703.00792*, vol. abs/1703.00792, 2017.

[20] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and Phan H. A, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[21] Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia, "Meshlab: an open-source 3d mesh processing system," *ERCIM News*, , no. 73, pp. 45–46, April 2008.

[22] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos, "Deep learning with sets and point clouds," in *ArXiv e-prints*, vol. 1611.