General-Purpose Deep Point Cloud Feature Extractor

Conference Paper · March 2018

DOI: 10.1109/WACV.2018.00218

6 authors, including:

Miguel Dominguez
Rochester Institute of Technology
7 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Rohan Dhamdhere
Rochester Institute of Technology
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Atir Petkar
Rochester Institute of Technology
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Shagan Sah
Rochester Institute of Technology
19 PUBLICATIONS **17** CITATIONS
SEE PROFILE

General-Purpose Deep Point Cloud Feature Extractor

Miguel Dominguez Rohan Dhamdhere Atir Petkar
Saloni Jain Shagan Sah Raymond Ptucha
Machine Intelligence Lab
Rochester Institute of Technology

Abstract

Depth sensors used in autonomous driving and gaming systems often report back 3D point clouds. The lack of structure from these sensors does not allow these systems to take advantage of recent advances in convolutional neural networks which are dependent upon traditional filtering and pooling operations. Analogous to image based convolutional architectures, recently introduced graph based architectures afford similar filtering and pooling operations on arbitrary graphs. We adopt these graph based methods to 3D point clouds to introduce a generic vector representation of 3D graphs, we call graph 3D (G3D). We believe we are the first to use large scale transfer learning on 3D point cloud data and demonstrate the discriminant power of our salient latent representation of 3D point clouds on unforeseen test sets. By using our G3D network (G3DNet) as a feature extractor, and then pairing G3D feature vectors with a standard classifier, we achieve the best accuracy on ModelNet10 (93.1%) and ModelNet 40 (91.7%) for a graph network, and comparable performance on the Sydney Urban Objects dataset to other methods. This general-purpose feature extractor can be used as an off-the-shelf component in other 3D scene understanding or object tracking works.

1. Introduction

Interest in deep learning on 3D meshes and point clouds is accelerating, in part due to development of low cost sensors such as LIDAR, RADAR and IR for applications such as autonomous driving and advanced object and activity recognition. Traditional Convolutional Neural Networks (CNNs) can operate on voxel representations of 3D data [63, 30, 40, 61, 9, 55, 57], and recently have been adapted to operate directly on point clouds [34, 35, 28] and graph data [45, 18, 49, 58] formed from meshes and point clouds. The recent introduction of large, high-quality 3D datasets [17, 11, 63] enable successful training of deep learning approaches.

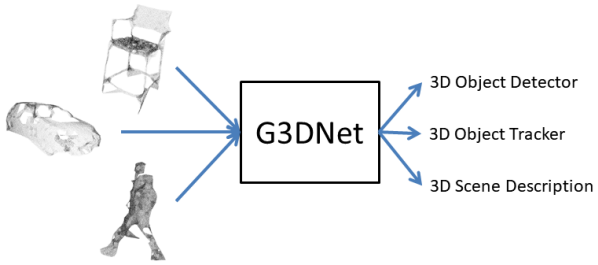


Figure 1: G3DNet is a general-purpose 3D feature extractor that can generate compact vector representations of point clouds for usage in other machine learning tasks. The samples in this figure from top to bottom are a chair, car and person.

The commonly used ImageNet [16] dataset, or more specifically the 1000 class image recognition challenge partition [37] is often used for training due to its sufficiently complex content which provides useful general-purpose features even on data not strictly within the 1000 specific classes. By passing unforeseen samples into such a pretrained CNN, the corresponding vector representations can feed subsequent machine learning tasks. For example, it has been shown that feature vectors composed of fully connected layers from deep CNNs form salient features for object detection [29], captioning [19, 13, 25, 56], and video understanding [50, 52].

There are multiple 3D mesh and point cloud datasets that provide tens of thousands of individual samples and hundreds of unique classes. Although developed for varied purposes, this research investigates combining these disparate datasets to train a single general purpose 3D feature extractor. We term our Graph 3D network feature extractor as G3DNet and the resulting vector representation as a G3D vector. We show these G3D features serve as a useful vector representation on unforeseen 3D data.

Our G3D models utilize recent advances in Graph Convolutional Neural Network theory which allows 3D meshes

and point clouds to be posed as graphs. Graph convolutions and graph clustering can replicate the convolution filtering and pooling in traditional image-based CNNs. G3D feature vectors are the vectors from the first fully connected layer of our deep graph networks. G3D vectors extracted from unforeseen datasets are used to train independent classifiers. Our results achieve comparable results with end-to-end systems that were trained specifically for their respective datasets. Our trained feature extractor and source code will be open source at https://github.com/WDot/G3DNet so that other researchers can incorporate it as an "off the shelf" component in their work.

The contributions of the paper are three fold. Firstly, we represent 3D point cloud datasets as graphs such that we can apply recent discoveries in graph CNNs. Secondly, we train a large graph CNN on 3D point clouds to create a general-purpose vector representation for graphs. Lastly, we show the applicability of our G3D representation on unforeseen 3D point cloud datasets, empirically giving evidence to the saliency and usefulness of this vector representation. The code for replicating the experiments will be released with publication.

The rest of this paper is organized as follows: Section 2 reviews related work associated with models using 3D point clouds and graph convolutions. Section 3 details our G3D contribution. Section 4 discusses our experiments along with the results. Concluding remarks are presented in Section 5.

2. Related Work

A number of 3D datasets have been released in recent years that can be used for classification tasks, and as a result there has been an explosion in the number of different methods for 3D object classification. Researchers have transformed the 3D data into voxels, image projections, raw point clouds, and graphs, and trained convolutional neural networks on each. Graph CNNs have the potential to be a superior method, containing more information than image projections and point clouds at a lower cost than voxels.

2.1. 3D Datasets

Table 1 is a list of 3D Datasets that focus on studying individual objects. There are other 3D datasets, such as those used for 3D estimation in cities [59, 36, 42]. Since VGGNet [46] has seen wide use as a feature extractor, and it was trained on a large classification problem, we will attempt to optimize our feature extractor based on its classification ability.

ModelNet [63] is the most popular dataset for researching 3D object classification. It contains two splits, ModelNet10 and ModelNet40, meant to classify household objects such as chairs and desks. There is a larger ModelNet (ModelNet Complete in Table 1) database that has not, to our knowledge, been applied to a classification task. We take a subset of that database as additional pretraining samples for our networks (Our Cut in Table 1.

Table 1: Popular 3D Datasets that focus on studying individual objects along with their number of Samples and Classes.

Dataset	# Samples	# Classes
ModelNet10	4,899	10
ModelNet40	12,311	40
ModelNet Complete	127,915	662
ModelNet "Open"	36,621	421
ShapeNetCore	51,300	55
Sydney Urban Objects	588	14

ShapeNet [11] is a large 3D database designed for 3D segmentation and reconstruction. To assist with these tasks it contains a large number of labeled samples of objects, which we can also use as pretraining for our network.

Sydney Urban Objects [17] is a small point cloud dataset that represents objects measured from a LIDAR sensor exploring an urban environment. Unlike the other datasets in Table 1, these samples are realistically noisy because they were not created in a lab.

2.2. Classification on 3D Data

Much of the work that has gone into 3D classification has focused on the ModelNet10 and ModelNet40 datasets. Many works transform the 3D meshes into voxels because they can be convolved just like any other tensor [63, 30, 40, 61, 9, 55, 57]. Estimating voxel orientation in addition to classifying can improve results on classification [40, 61]. Wu et al. [55] trains a generative adversarial network whose hidden representation is used as classification features. There are two downsides to voxels. First, 3D meshes and point clouds only encode the surface, so most of the voxels in a sample are zero valued. Second, the dimensionality of voxels increases cubically, which is why most works keep the dimensionality small (e.g. $32 \times 32 \times 32$), which eliminates most fine details.

Another avenue researchers frequently choose is to map the 3D data to 2D, either by creating images from multiple angles [48, 5, 24, 60], or by creating a mapping function from mesh to image such that depth information is preserved [44, 47]. This allows relatively high-quality 2D image representations.

There have been a handful of attempts at creating CNNs that operate directly on the 3D data rather than transforming it into a more convenient tensor format. One approach has been to try to define convolution and pooling operations directly on point clouds that are permutation invariant (or at least permutation equivariant) [35, 34]. Klokov and Lempitsky [28] eschew convolutions all together for a neural net-

work network based on KD-Trees. Some researchers create a graph from the data so that the points benefit from connections that encode structure, either by treating the mesh as a graph [49, 18], or drawing a k-nearest neighbor graph from the point clouds [45, 35].

2.3. Graph Convolutional Neural Network

A graph is composed of vertices and edges that connect them. This can be represented by the tuple $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{A})$. $\boldsymbol{V} \in \mathbb{R}^{N \times F}$ is a matrix that represents the vertex data, with N vertices and F features for each vertex. $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix, where each entry a_{ij} is a nonzero weight if there is a connection between vertices i and j, and zero if there is no connection.

A popular alternative to the graph adjacency is the Graph Laplacian $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$. The degree matrix $\boldsymbol{D}$ is a diagonal matrix whose elements are the row-wise sums of $\boldsymbol{A}$. The Laplacian $\boldsymbol{L}$ is often used because its eigenvectors can be treated as an analogue to the Discrete Fourier Transform (DFT) matrix. Spectral Graph theory exploits properties of the graph in this spectral domain for clustering, filtering, and dimensionality reduction. The eigenbasis computed from a given graph is unique to that graph, which means any spectral filters learned for a given graph do not generalize to other graphs.

The first Graph Convolutional Neural Networks (Graph CNNs) were proposed by Bruna et al. [10], who proposed both a "spatial" network and a "spectral" network. The spatial network filtered vertices based on local neighborhoods and the spectral network filtered by elementwise multiplication of the frequency components by filter taps. Edwards and Xie [20] created a network with spectral filtering and algebraic multigrid (AMG)-based pooling. Henaff [22] also proposed a spectral network, but used spectral embeddings [53] for pooling. Both works attempt to learn "smooth" filters, as smoothness in the spectral domain corresponds to locality in the spatial domain. Yi et al. [58] attempts to rectify the fact that spectral filters do not generalize to different samples by learning a mapping function between different graph samples.

Many other works attempt to define useful definitions of spatial graph convolution for their networks. The challenge in this effort is that unlike images, it is not reasonable to have a kernel of a fixed size and shape, since different vertices have different-sized neighborhoods. Some efforts include filtering based on a canonical linear representation of a graph [32], filtering based on the diffusion of a graph from a given vertex [4], and learning latent representations from random walks [33]. Several works filter based on a polynomial of the graph structure, either on the Laplacian [15, 27] or the adjacency matrix [38]. A k-th order polynomial filter in these works will only filter vertices within k hops of a given vertex. One consequence of this filter design is that every vertex a given number of hops away will be filtered the same way. These filters are isotropic, which means they have no sense of filtering differently depending on direction.

Different methods of anisotropic spatial filters that can take "direction" into account have recently been developed. Petroski Such et al. [49] proposed a variation on the polynomial of $\boldsymbol{A}$ where the adjacency is partitioned to create multiple matrices, each with their own filter taps. Simonovsky and Komodakis [45] learned multilayer perceptrons that dynamically generated filter taps based on edge features. Monti et al. [31] adapts their manifold work to graphs by mapping vertices to a "patch", then learning filters for that patch.

3. Method

To train a useful, generalizable, deep neural network, we need to define a data representation, a convolution operation, and a pooling operation. We can leverage graph theory concepts to create a complete graph-based CNN. In addition, we define two different potential feature extractors, whose architectures and training procedures are laid out in detail.

3.1. Graph Convolution

We use the graph convolution definition proposed by Petroski Such et al. [49]. It is a variation of the graph convolution described by Sandryhaila et al. [38], which proposes a shift-invariant convolutional filter as a polynomial of the adjacency matrix, as in (1).

$$\boldsymbol{H} = h_0\boldsymbol{I} + h_1\boldsymbol{A} + h_2\boldsymbol{A}^2 + ... + h_k\boldsymbol{A}^k, \boldsymbol{H} \in \mathbb{R}^{N \times N} \quad (1)$$

The filter matrix $\boldsymbol{H}$ is then multiplied by the vertex matrix: $\boldsymbol{V}_{out} = \boldsymbol{H}\boldsymbol{V}_{in}$. Each row (and column) in $\boldsymbol{A}$ represents the immediate "one-hop" neighbors of the corresponding vertex. $\boldsymbol{A}^2$ represents the two-hop neighbors, $\boldsymbol{A}^3$ represents the three-hop neighbors, and so on. Vertices of a given distance are filtered by a single learnable h_i filter tap. This is an isotropic filter, which means it filters the same amount in every direction.

[49] proposed changing $\boldsymbol{A}$ to an adjacency tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N \times N \times L}$. Each slice $\boldsymbol{A}_\ell \in \mathbb{R}^{N \times N}$ of the tensor can either encode separate edge features or partition an adjacency matrix to indicate direction. Like [49] and [27] we treat each layer as a linear equation (only the $\boldsymbol{I}$ and $\boldsymbol{A}$ terms are kept), because multiple convolution layers should approximate a higher-order polynomial.

3.2. Graph Pooling

Graph pooling, unlike typical image pooling, requires two operations: the reduction of the graph structure and the

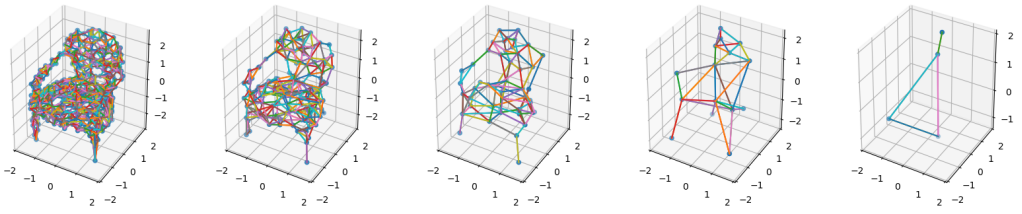


Figure 2: A chair graph sample (**left**), pooled to 33% of its original vertices four times going from left to right.

reduction of a graph signal. The former operates on the adjacency matrix, while the latter is applied to vertex values. For example, in an image, the underlying structure is implicit: a larger grid structure is reduced to a smaller grid structure of the desired size. In a graph, the structure needs to be solved for. In the case of point clouds and meshes, the structure will be different for each sample.

We are inspired by Algebraic Multigrid, specifically with the Galerkin operator [51] that reduces a Laplacian matrix $\boldsymbol{L} \in \mathbb{R}^{N_1 \times N_1}$ by multiplying on both sides with a a coarsening matrix $\boldsymbol{P} \in \mathbb{R}^{N_1 \times N_2}$ as in (2).

$$\boldsymbol{L}^{reduced} = \boldsymbol{P}^T \boldsymbol{L} \boldsymbol{P} \tag{2}$$

$\boldsymbol{P}$ is a (usually sparse) matrix where the nonzero entries in a column indicate that the corresponding vertices in the original graph's vertex set $(1, 2, \ldots, N_1)$ are clustered into a vertex in the coarsened graph's vertex set $(1, 2, \ldots, N_2, N_2 < N_1)$. This means some form of graph clustering algorithm must be employed to create this mapping. We use a variation of k-means clustering created by Bell that uses the Bellman-Ford algorithm as a distance metric, making it suitable for graph data [6, 7]. Only edge information is used for clustering, not vertex features. The resulting $\boldsymbol{P}$ matrix has 1 in a given entry if a given vertex is a member of a cluster, and 0 otherwise. We then column-normalize the matrix.

To produce the pooled outputs, we use (3) and (4). Equation (4) is equivalent to an average pooling due to the column-normalization. These operations can be included as layers in a deep neural network that can be back-propagated since they are matrix multiplications. The $\boldsymbol{P}$ matrices are pre-computed for each sample and directly placed into the desired location in the network.

$$\boldsymbol{A}_\ell^{reduced} = \boldsymbol{P}^T \boldsymbol{A}_\ell \boldsymbol{P} \text{ for each } \ell \in \{1, 2, \ldots L\} \tag{3}$$

$$\boldsymbol{V}^{reduced} = \boldsymbol{P}^T \boldsymbol{V} \tag{4}$$

(a) Original (b) Voronoi Filtered (c) Resampled

Figure 3: Low-dimensional point cloud samples are resampled to a high dimension by constructing a mesh from the data and sampling the mesh to the required dimensionality. The sample in this figure is a traffic sign.

3.3. Training Process

3.3.1 Preprocessing: Data Representation

We treat each sample in this work as a point cloud. We use Meshlab [14] to preprocess the raw data into a form our neural network expects. If the sample is originally a mesh, we resample 500 points on the surface of the mesh. If the sample is originally a point cloud with more than 500 points, we subsample it down to 500. If the point cloud sample is less than 500 points, we construct a mesh from those points using Voronoi Filters [1], then sample 500 points on that mesh. This is illustrated in Figure 3. The Meshlab scripts sometimes produce meshes that have slightly more or less than 500 vertices, so all graphs are zero padded to 516 vertices.

We follow the same method of [49, 18] in representing our graph. Each vertex corresponds to a point in the cloud. Each has three features: the X, Y, and Z coordinates of the point in 3D space. Edges are drawn to the $k = 6$ nearest neighbors and have a constant value of 1. We partition the adjacency matrix based on direction into eight bins. Each

edge is treated as a vector $V_j - V_i$. The vector is placed into one of the eight octants in 3D space. Each octant has its own layer in the adjacency tensor containing only edges within that octant.

ModelNet10 is designed so that every member of a class has the same orientation. For ModelNet40 (and the rest of ModelNet) this assumption does not hold true. Sedaghat et al. [39, 41] have created a method of automatically aligning objects per-class that we use to reduce the variance between ModelNet40 samples.

3.3.2 Preprocessing: Data Collection

The datasets utilized are: ModelNet10, ModelNet40, the full ModelNet dataset [63], ShapeNetCore [11], and Sydney Urban Objects [17]. We train on ShapeNetCore as well as entries in the full ModelNet not found in ModelNet10 or ModelNet40. The remainder of the datasets are used for evaluation.

ShapeNetCore is a neatly organized dataset and can simply be preprocessed and passed through the network. However, the full ModelNet dataset has not been curated for wider consumption. There are 662 classes, though some classes have many examples and some have very few. Some are only available in a proprietary Google Sketchup format. We remove these samples, along with any samples that have the same class label as a sample in ModelNet10 and ModelNet40. After screening, we have 36,621 samples in an open format which we call "ModelNet Open", representing 421 unique classes.

We treat the entirety of ModelNet Open as a training set, with no validation. We use it more as a tool for creating a wide variety of useful features than as a means to create a 421-class classifier. We use the ShapeNetCore training and validation sets for training and its test set for validation. Since we do not intend to evaluate our network on ShapeNetCore, there is no need for a held-out validation set.

This data will be used in a multistage training process to create deep networks that can understand objects from both datasets.

3.3.3 Architectures

We define two very deep network architectures, illustrated in Figure 4. Figure 4a we call G3DNet-18 and Figure 4b, the deeper network, we call G3DNet-26. Each convolutional layer is followed by batch normalization [23] and ReLU activation. Each network takes multiple weeks to train on a single Geforce GTX Titan X GPU. To speed up training we split the training into a stepwise procedure.

First, we train G3DNet-18 from scratch on ModelNet Open. We train for close to $90,000$ iterations. The initial learning rate is 0.01, which we reduce to 0.001 at $50,000$ iterations and 0.0001 near $68,000$ iterations. A final fully connected layer with 421 neurons is attached to attempt to classify the 421 classes in ModelNet Open. At the end of this training step, the final fully connected layer is switched out for a 55 neuron layer and the model is fine-tuned on ShapeNetCore for nearly $30,000$ iterations. The initial learning rate is 0.001, then divided by 10 after 4000 iterations. This is the final G3DNet-18 network.

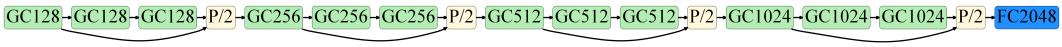
We initialize the G3DNet-26 network by loading the first twelve convolutional layers with the weights of the GENNet-18 convolutional layers, and randomly elsewhere. Then we train this network for roughly $36,600$ iterations on ShapeNetCore classification. We experiment with learning rates for the first $5,000$ iterations but settle on 0.001 until iteration $30,000$, when we divide by 10.

Each step of each network is trained with momentum with a parameter of 0.9. Data augmentation is performed on the training samples through random vertex dropouts and random flips, as used in [45].

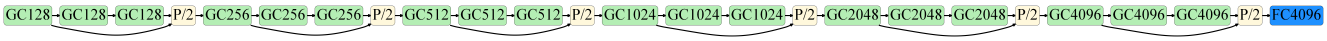
Figure 5 illustrates the networks' accuracies on these intermediate classification tasks. The left plot shows that G3DNet-18 does not have the capacity to overfit on the 421-class ModelNet Open. This is not an issue because no 3D object recognition task yet has this many classes. This is mostly an opportunity to learn a wide variety of features. The center plot Shows that when G3DNet-18 is fine-tuned on ShapeNetCore, it very quickly reaches its capacity due to the useful features learned from ModelNet Open. The rightmost plot shows the training of G3DNet-26. This network contains several fresh convolutional layers and a new fully connected layer that have not benefited from pretraining. It does benefit somewhat from the earlier pretrained layers, as the 55-class task very quickly reaches 20% accuracy. While this network has a harder time fitting to the ShapeNetCore data than G3DNet-18, we will see if the added depth and complexity give us more useful features in evaluation.

4. Results

We extract G3D features from ModelNet10, ModelNet40, and Sydney Urban Objects datasets, then train "shallow" classifiers on these extracted G3D feature vectors. Example ModelNet40 samples are in Figure 6 and example Sydney samples are in Figure 7. The classifiers we use are support vector machines (SVMs) and multilayer perceptrons (MLPs). We use our G3DNet models to collect 11 G3D feature vectors per sample per network we train, in order to account for the random augmentation that occurs in the network. This allows us to train each classifier with augmented G3D data and evaluate each classifier with a single-model voting scheme.



(a) G3DNet-18 Network



(b) G3DNet-26 Network

Figure 4: Network architectures for ModelNet experiments. GCX represents a graph convolution with X filters, P/2 means a pooling that reduces the vertices by half, and FCY is a fully-connected layer with Y neurons.

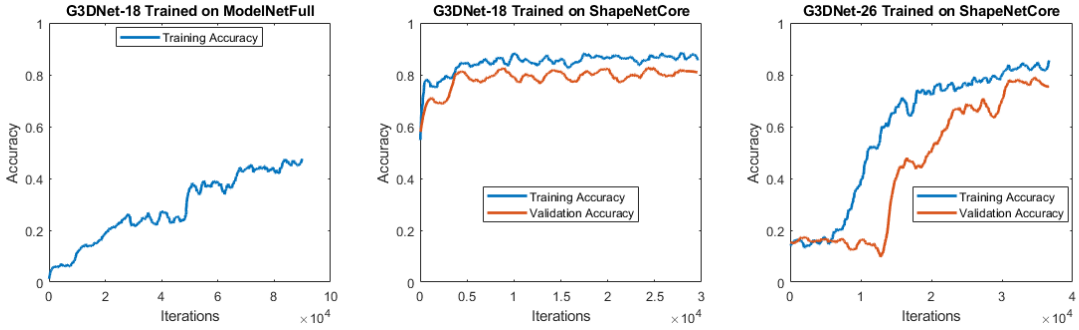


Figure 5: Training and Validation Accuracies on G3DNet Classification Tasks.

4.1. Modelnet

We perform a hyperparameter search on the ModelNet10 and ModelNet40 dataset, using 10% of the training set as validation for this purpose. We evaluate the best set of hyperparameters in a series of ablation experiments. We evaluate the held-out test set with our G3DNet-18 feature vectors, our G3DNet-26 feature vectors, a G3DNet-18 fine-tuned on the training set, and the fine-tuned feature vectors subjected to a vote over 10 predictions. This vote is performed by a single trained G3DNet model that applies the random data augmentations from training to each input sample. It is an ensemble of predictions, but only one model.

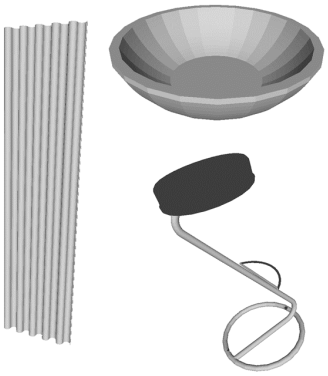


Figure 6: Example meshes in the ModelNet40 Dataset. Left: a curtain. Top: a bowl. Bottom: a stool.

The best ModelNet10 MLP contains two hidden layers (each with 2000 neurons), ReLU activations, Adam optimization [26], a batch size of 200, an initial learning rate of 1.01×10^{-3}, an exponential learning rate decay parameter of 0.7, and an ℓ_2 regularization parameter of 5×10^{-6}. Training continues until the network loss improvement is less than 0.0001. The best Modelnet40 MLP contains three hidden layers (1500, 1000, and 500 neurons), has a a batch size of 200, ReLU activations, momentum optimization, a learning rate of 1.01×10^{-3} an exponential learning rate decay parameter of 0.7, and an ℓ_2 regularization parameter of 5×10^{-5}.The best SVMs for both ModelNet10 and ModelNet40 have a C parameter of 1.0, a radial basis function kernel, and a γ parameter equal to the reciprocal of feature size. The results of our experiments are in Table 2

Table 2: ModelNet Classification Accuracy Overall.

	Modelnet	
Method	10 class	40 class
G3DNet-18 SVM	86.5%	76.0%
G3DNet-26 SVM	70.2%	58.3%
G3DNet-18 MLP	82.7%	82.5%
G3DNet-26 MLP	78.3%	70.5%
G3DNet-18 MLP Fine-Tuned	90.2%	88.8%
G3DNet-18 SVM Fine-Tuned	91.5%	85.7%
G3DNet-18 MLP, Fine-Tuned, Vote	92.8%	**91.7**%
G3DNet-18 SVM, Fine-Tuned, Vote	**93.1**%	91.13%

We see that G3DNet-26 provides weaker features than G3DNet-18. This may be because the G3DNet-26 feature

vector is larger (4096 elements) and thus may be harder to generalize over a relatively small dataset. However, it may also be because the G3DNet-26 had a more difficult time learning from the same training data, achieving a lower validation accuracy at convergence on ShapeNetCore. MLPs appear to have an advantage over SVMs until fine-tuning and voting are added, at which point SVMs start to perform better.

Our work is compared against the broader ModelNet literature in Table 3. The table is separated by non-graph methods (above the horizontal separator) and graph methods. Our work achieves the best results so far among graph methods, and is competitive among alternative methods. Our best model that was not fine-tuned is still only a few percentage points away from the previous best graph model on ModelNet10, and close in performance to several architectures on ModelNet40. This is with the G3DNet feature extractor never having seen a sample from ModelNet10 or ModelNet40!

Table 3: ModelNet Classification Accuracy Overall.

	Modelnet	
Method	10 class	40 class
set-convolution [35]	-	90.0%
Zanuttigh and Minto [60]	91.6%	87.8%
DS-CNN [54]	-	93.8%
PANORAMA-NN [43]	91.1%	90.7%
PointNet [34]	-	89.2%
ORION [40]	93.8%	-
Kd-Net [28]	94.0%	91.8%
VRN Ensemble [9]	**97.1**%	**95.5**%
LonchaNet [21]	94.37%	-
Arvind et al. [3]	-	86.5%
Soltani et al. [2]	-	82.1%
3DmFV-Net [8]	95.2%	91.6%
graph-convolution [35]	-	58.0%
Dominguez et al. [18]	74.3%	-
ECC [45]	90.0%	83.2%
G3DNet-18 MLP	86.2%	82.5%
G3DNet-18 MLP, Fine-Tuned, Vote	92.8%	91.7%
G3DNet-18 SVM, Fine-Tuned, Vote	93.1%	91.13%

4.2. Sydney Urban Objects

Sydney Urban Objects has a fixed set of four splits for 4-fold cross validation on fourteen of the most common classes. F1 Score weighted by class frequency is the standard metric for this dataset. We similarly perform hyperparameter searches to find the best SVM and MLP. The MLP is trained with one 64-neuron hidden layer, a batch size of 160, learning rate of 0.002 with exponential decay parameter of 0.1, momentum optimization, ReLU activation, and an ℓ_2 regularization parameter of 3.5. The SVM is trained with $C = 2.25 \times 10^{-12}$, a 13-degree polynomial kernel with a constant coefficient of 10, and a γ equal to the reciprocal of the feature size. When performing the fine tuning, we created four separate copies of G3DNet-18, one trained on the training set for each fold. When evaluating, the MLP and SVMs see training samples that are fine-tuned and test samples that are not fine-tuned for each fold. The results are in Table 4.

Table 4: Sydney Urban Objects 4-fold Cross Validation Mean F1 Score.

Method	Mean F1
G3DNet-18 MLP	60.3%
G3DNet-26 MLP	45.9%
G3DNet-18 SVM	59.1%
G3DNet-26 SVM	44.1%
G3DNet-18 MLP Fine-Tuned	67.8%
G3DNet-18 SVM Fine-Tuned	65.9%
G3DNet-18 MLP Fine-Tuned, Vote	**72.7**%
G3DNet-18 SVM Fine-Tuned, Vote	70.4%

G3DNet-18 is again a better fit than G3DNet-26, which makes sense as this is a small dataset. Fine-tuning and voting drastically improve results but even without these boosts the feature extractor is a competent baseline without having seen the Sydney Urban Objects samples. It's also worth considering that Sydney Urban Objects is the noisiest dataset, while the feature extractor was generally trained on clean data.

Our comparison with the literature is in Table 5. While

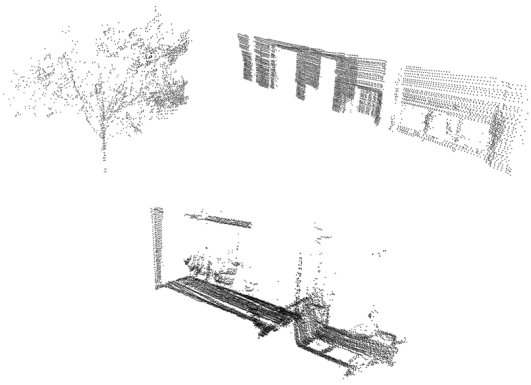


Figure 7: Example point clouds in the Sydney Urban Objects Dataset. From left to right: a tree, a truck, and a building. These samples have not been yet preprocessed into graphs.

we do not achieve state of the art, our results are comparable, and again reasonably close even without fine-tuning.

Table 5: Sydney Urban Objects 4-fold Cross Validation Mean F1 Score.

Method	Mean F1
Triangle+SVM[17]	67.1
GFH+SVM[12]	71.0
VoxNet[30]	73.0
ORION[40]	77.8
ECC[45]	78.4
LightNet[62]	**79.8**
G3DNet-18 MLP	60.3%
G3DNet-18 MLP Fine-Tuned, Vote	72.7%

5. Conclusion

The popularity of autonomous driving and gaming systems has been driving the growth of 3D point cloud sensors. Unlike image, video, and audio data which occurs on regular gridded structures, 3D point cloud data requires special purpose convolution and pooling operations. This research adopts graph based convolutions to 3D point clouds. We introduce G3DNet-18 and G3DNet-26, custom 3D point cloud deep feature extractors. By passing 3D point clouds into our G3DNet architectures, G3D feature vectors are created. We demonstrate the discriminant power of the G3D vector representation on unforeseen point cloud test sets. By using our G3D network (G3DNet) as a feature extractor, and then pairing G3D vector outputs with a standard classifier, we are able to achieve comparative quality as custom deep feature extractors on the ModelNet 10, ModelNet 40, and Sydney Urban Objects datasets. This tool is available as an off the shelf component to assist in more complex tasks such as 3D object tracking and 3D scene understanding, or as an additional set of features paired with image data.

References

[1] N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. *Discrete and Computational Geometry*, 22:481–504, 1998. 4

[2] A. Arsalan Soltani et al. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 7

[3] V. Arvind et al. Wide and deep volumetric residual networks for volumetric image classification. *ArXiv e-prints*, Sept. 2017. 7

[4] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. *Advances in Neural Information Processing Systems 29*, pages 1993–2001, 2016. 3

[5] S. Bai et al. Gift: A real-time and scalable 3d shape search engine. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5023–5032, June 2016. 2

[6] W. N. Bell. *Algebraic Multigrid For Discrete Differential Forms*. Thesis, 2008. 4

[7] W. N. Bell et al. PyAMG: Algebraic multigrid solvers in Python v3.0, 2015. Release 3.2. 4

[8] Y. Ben-Shabat, M. Lindenbaum, and A. Fischer. 3D Point Cloud Classification and Segmentation using 3D Modified Fisher Vector Representation for Convolutional Neural Networks. *ArXiv e-prints*, Nov. 2017. 7

[9] A. Brock et al. Generative and discriminative voxel modeling with convolutional neural networks. 1, 2, 7

[10] J. Bruna et al. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013. 3

[11] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 1, 2, 5

[12] T. Chen, B. Dai, D. Liu, and J. Song. Performance of global descriptors for velodyne-based urban object recognition. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 667–673, June 2014. 8

[13] X. Chen and C. L. Zitnick. Learning a recurrent visual representation for image caption generation. 2015. 1

[14] P. Cignoni et al. Meshlab: an open-source 3d mesh processing system. *ERCIM News*, (73):45–46, April 2008. 4

[15] M. Defferrard et al. Convolutional neural networks on graphs with fast localized spectral filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3844–3852. Curran Associates, Inc. 3

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 1

[17] M. D. Deuge, A. Quadros, C. Hung, and B. Douillard. Unsupervised feature learning for classification of outdoor 3d scans. In *Australasian Conference on Robotics and Automation*, volume 2, 2013. 1, 2, 5, 8

[18] M. Dominguez et al. Towards 3d convolutional neural networks with meshes. In J. Luo, editor, *International Conference on Image Processing*. IEEE, 2017. 1, 3, 4, 7

[19] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015. 1

[20] M. Edwards and X. Xie. Graph convolutional neural network. In R. C. Wilson, E. R. Hancock, and W. A. P. Smith, editors, *British Machine Vision Conference*. BMVA Press. 3

[21] F. Gomez-Donoso et al. Lonchanet: A sliced-based cnn architecture for real-time 3d object recognition. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 412–418, May 2017. 7

[22] M. Henaff et al. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015. 3

[23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning*, 37:448–456, 07–09 Jul 2015. 5

[24] E. Johns et al. Pairwise decomposition of image sequences for active multi-view recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[25] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE CVPR*, pages 3128–3137, 2015. 1

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 6

[27] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of ICLR 2017*. 3

[28] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 2, 7

[29] A. Krizhevsky et al. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[30] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, September 2015. 1, 2, 8

[31] F. Monti et al. Geometric deep learning on graphs and manifolds using mixture model cnns. *arXiv preprint arXiv:1611.08402*, 2016. 3

[32] M. Niepert et al. Learning convolutional neural networks for graphs. In *Proceeding of the 33rd International Conference on Machine Learning*, pages 2014–2023, 2016. 3

[33] B. Perozzi et al. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD*, pages 701–710. ACM, 2014. 3

[34] C. R. Qi et al. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1, 2, 7

[35] S. Ravanbakhsh et al. Deep learning with sets and point clouds. In *Proceedings of ICLR 2017*, volume 1611. 1, 2, 3, 7

[36] H. Riemenschneider, A. Bódis-Szomorú, J. Weissenberg, and L. Van Gool. *Learning Where to Classify in Multi-view Semantic Segmentation*, pages 516–532. Springer International Publishing, Cham, 2014. 2

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1

[38] A. Sandryhaila and J. M. Moura. Discrete signal processing on graphs. *Signal Processing, IEEE Transactions on*, 61(7):1644–1656, 2013. 3

[39] N. Sedaghat and T. Brox. Unsupervised generation of a viewpoint annotated car dataset from videos. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 5

[40] N. Sedaghat et al. Orientation-boosted voxel nets for 3d object recognition. *CoRR*, abs/1604.03351, 2016. 1, 2, 7, 8

[41] N. Sedaghat et al. Orientation-boosted voxel nets for 3d object recognition. In *British Machine Vision Conference (BMVC)*, 2017. 5

[42] A. Serna, B. Marcotegui, F. Goulette, and J.-E. 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*. 2

[43] K. Sfikas et al. Exploiting the PANORAMA Representation for Convolutional Neural Network Classification and Retrieval. In I. Pratikakis, F. Dupont, and M. Ovsjanikov, editors, *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2017. 7

[44] B. Shi et al. Deeppano: Deep panoramic representation for 3-d shape recognition. *IEEE Signal Processing Letters*, 22(12):2339–2343, Dec 2015. 2

[45] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1, 3, 5, 7, 8

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015. 2

[47] A. Sinha et al. *Deep Learning 3D Shape Surfaces Using Geometry Images*, pages 223–240. Springer International Publishing, Cham, 2016. 2

[48] H. Su et al. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015. 2

[49] F. P. Such, S. Sah, M. A. Dominguez, S. Pillai, C. Zhang, A. Michael, N. D. Cahill, and R. Ptucha. Robust spatial filtering with graph convolutional neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):884–896, Sept 2017. 1, 3, 4

[50] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015. 1

[51] U. Trottenberg et al. *Multigrid*. Academic press, 2000. 4

[52] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko. Sequence to sequence-video to text. In *Proceedings of the IEEE international conference on computer vision*, pages 4534–4542, 2015. 1

[53] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007. 3

[54] C. Wang et al. Dominant set clustering and pooling for multi-view 3d object recognition. In *British Machine Vision Conference*. British Machine Vision Association, 2017. 7

[55] J. Wu et al. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in Neural Information Processing Systems 29*, pages 82–90, 2016. 1, 2

[56] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France, 07–09 Jul 2015. PMLR. 1

[57] X. Xu and S. Todorovic. Beam search for learning a deep convolutional neural network of 3d shapes. pages 3506–3511, Dec 2016. 1, 2

[58] L. Yi et al. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1, 3

[59] A. R. Zamir, T. Wekel, P. Agrawal, C. Wei, J. Malik, and S. Savarese. Generic 3D representation via pose estimation and matching. In *European Conference on Computer Vision*, pages 535–553. Springer, 2016. 2

[60] P. Zanuttigh and L. Minto. Deep learning for 3d shape classification from multiple depth maps. In J. Luo, editor, *International Conference on Image Processing*. IEEE, 2017. 2, 7

[61] S. Zhi et al. LightNet: A Lightweight 3D Convolutional Neural Network for Real-Time 3D Object Recognition. In I. Pratikakis, F. Dupont, and M. Ovsjanikov, editors, *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2017. 1, 2

[62] S. Zhi, Y. Liu, X. Li, and Y. Guo. Toward real-time 3d object recognition: A lightweight volumetric CNN framework using multitask learning. *Computers & Graphics*, Nov. 2017. 8

[63] W. Zhirong et al. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920. 1, 2, 5