Mo-Net: Flavor the Moments in Learning to Classify Shapes

Mor Joseph-Rivlin\textsuperscript{1}  \quad Alon Zvirin\textsuperscript{2}  \quad Ron Kimmel\textsuperscript{2}

\textsuperscript{1} Technion EE department, Haifa, Israel  \quad \textsuperscript{2} Technion CS department, Haifa, Israel

Abstract

A fundamental question in learning to classify 3D shapes is how to treat the data in a way that would allow us to construct efficient and accurate geometric processing and analysis procedures. Here, we restrict ourselves to networks that operate on point clouds. There were several attempts to treat point clouds as non-structured data sets by which a neural network is trained to extract discriminative properties. The idea of using 3D coordinates as class identifiers motivated us to extend this line of thought to that of shape classification by comparing attributes that could easily account for the shape moments. Here, we propose to add polynomial functions of the coordinates allowing the network to account for higher order moments of a given shape. Experiments on two benchmarks show that the suggested network is able to provide more accurate results and at the same token learn more efficiently in terms of memory and computational complexity.

1. Introduction

In recent years the popularity and demand for 3D sensors has vastly increased. Applications using 3D sensors include robot navigation, stereo vision, and advanced driver assistance systems to name just a few. Recent studies attempt to adjust deep neural networks (DNN) to operate on 3D data representations for diverse geometric tasks. Motivated mostly by memory efficiency, our choice of 3D data representation is to process raw point clouds. One school of thought promoted feeding geometric features as input to deep neural networks that operate on point clouds for classification of rigid objects.

From a geometry processing point of view, it is well known that moments characterize a surface and can be useful for the classification task. To highlight the importance of moments as class identifiers, we first consider the case of a continuous surface. In this case, geometric moments uniquely characterize an object. Furthermore, a finite set of moments is often sufficient as a compact signature that defines the surface \cite{1}. This idea was classically used in finding surface similarities. For example, if all moments of two surfaces coincide, the surfaces are considered identical. Moreover, sampled surfaces, such as point clouds, can be identified by their estimated geometric moments, where it can be shown that the error introduced by the sampling is proportional to the sampling radius and uniformity. Our goal is to allow a neural network to simply lock onto variations of geometric moments. One of the main challenges of this approach is that training a neural network to approximate polynomial functions requires the network depth and complexity to be logarithmically inversely proportional to the approximation error \cite{2}. In practice, in order to approximate polynomial functions of the coordinates for the calculation of geometric moments the network requires a large number of weights and layers. Qi et al. \cite{3} proposed a network architecture which processes point clouds for object classification. The framework they suggested includes lifting the coordinates of each point into a high dimensional learned space, while ignoring the geometric structure. An additional pre-processing transformation network (T-Net) was supposed to canonize a given point cloud, yet it was somewhat surprising to discover that the T-Net results are not invariant to the given orientations of the point cloud. Learning to lift into polynomial spaces would have been a challenge using the architecture suggested in \cite{3}. At the other end, networks that attempt to process other representations of low dimensional geometric structures such as meshes, voxels (volumetric grids), and multi-view projec-

Figure 1. Illustration of the proposed object classification architecture. The input of the network includes the point cloud coordinates as well as second order polynomial functions of these coordinates. It enables the network to efficiently learn the shape moments.
tions are often less efficient when considering both computational and memory complexities.

In this paper, we propose a new network that favors geometric moments for point cloud object classification. The most prominent element of the network is supplementing the given point cloud coordinates together with polynomial functions of the coordinates, see Fig. 1. This simple operation allows the network to account for higher order moments of a given shape. The proposed network implementation is based on a simplified version of the pointNet architecture, with only one layer in the feature domain. Thereby, the suggested network requires relatively low computational resources in terms of floating point operations per second (FLOPs), and memory in sense of the number of network’s parameters. Experiments on two benchmarks show that the suggested scheme is able to learn more efficiently compared to pointNet in terms of memory and actual computational complexity while providing more accurate results. Lastly, it is easy to implement the proposed concept by just calculating the polynomial functions and concatenating them as an additional vector to the current input of point cloud coordinates.

2. Related Efforts

This section reviews some early results relevant to our discussion. First, we relate to methods used for data representation, spatial transformation canonization modules, and object classification that integrate features into neural networks. The second part describes early studies of higher order networks, in which each layer applies polynomial functions to its inputs, defined by the previous layer’s output. We provide evidence that similar simple lifting ideas were applied quite successfully to geometric object recognition and classification in the late 80’s.

2.1. Deep Learning of Geometric Structures

The most straightforward way to apply convolutional neural networks (CNNs) to 3D data is by transforming 3D models to grids of voxels, see for example [4][5][6]. A grid of occupancy voxels is produced and used as input to a 3D CNN. This approach has produced successful results, but has some disadvantages such as loss of spatial resolution, and the use of excessively large memory. For some geometric tasks that require the analysis of fine details, in some cases, implicit (voxel) representation would probably fail to capture fine features.

A desired geometric property of 3D rigid objects is invariance to transformations. Spatial Transformer Networks (STN) were suggested as a tool that learns such transformations from a given data in the context of deep learning [7]. It allows networks to learn a set of transformations that transforms the geometric input structure into some canonical configuration. STN is designed as a layer which can be integrated into other neural networks. For cloud of points in \( \mathbb{R}^3 \), the layer’s output is a nine elements vector that can be arranged as a 3 \( \times \) 3 matrix, that multiplies the coordinates of each point. The implementation of such a transformation is simple and does not require re-sampling like the case of voxels or most other implicit representations.

A deep neural network applied to point clouds known as pointNet was introduced in [3]. That architecture processes the points’ coordinates for classification and segmentation. The classification architecture is based on fully connected layers and symmetry functions, like max pooling, to compensate for potential permutations of the points. In addition, all Multi-Layer Perceptrons (MLPs) operations are performed per point, thus, interrelations between points are accomplished only by weight sharing. The architecture pipeline commences with MLPs to generate a per point feature vector, then, applies max pooling to generate global features that serve as a signature of the point cloud. Finally, fully connected layers produce output scores for each class. Part of the pointNet architecture is a transformer network based on the spatial transformer networks (STN) [7]. It is supposed to map the input point cloud to a canonical form. However, the part handling spatial context, the STN in pointNet, is sensitive to different orientations of the point cloud.

2.2. Higher-Order Neural Networks

Multi-layer perceptron (MLP) is a neural network with one or more hidden layers of perceptron units. The output \( \phi \) of such a unit with an activation function \( \sigma \), previous layer’s output \( \eta \) and vector of learned weights \( w \) is a first order perceptron, defined as \( \phi = \sigma(\sum_j w_j \eta_j) \). Where, \( \sigma \) is a sigmoid function, \( \sigma(a) = \frac{1}{1 + e^{-a}} \).

In the late 80’s, the early years of artificial intelligence, Giles et al. [8][9] proposed extended MLP networks called higher-order neural networks. Their idea was to extend all the perceptron units in the network to include also the sum of products between elements of the previous layer’s output \( \eta \). The extended perceptron unit named high-order unit is defined as

\[
\phi = \sigma \left( \sum_i w_i \eta_i + \sum_{i,j} w_{ij} \eta_i \eta_j + \sum_{i,j,k} w_{ijk} \eta_i \eta_j \eta_k + \ldots \right) \tag{1}
\]

These networks included some or all of the summation terms. Theoretically, an infinite term of single high order layer can perform any computation of a first order multilayer network [10]. Moreover, the convergence rate using a single high layer network is higher, usually by orders of magnitude, compared to the convergence rate of a multilayer first order network [2]. Therefore, higher-order networks are considered to be powerful, yet, at the cost of high memory complexity. The number of weights grow expo-
mentally with the number of inputs, which is a prohibitive factor in many applications.

A special case of high order networks is the square multi-layer perceptron proposed by Flake et al. [11]. They extend the perceptron unit with only the squared components of \( \eta \), given by

\[
\phi = \sigma \left( \sum_i w_i \eta_i + \sum_j w_j \eta_j^2 \right).
\]  

(2)

The authors have shown that with a single hidden unit the network has the ability to generate localized features in addition to spanning large volumes of the input space, while avoiding large memory requirements.

3. Methods

The main contribution of this paper is leveraging the network’s ability to operate on point clouds by adding polynomial functions to their coordinates. Such a design can allow the network to account for higher order moments and therefore achieve higher classification accuracy with lower time and memory consumption. Next, we show that it is indeed essential to add polynomial functions to the input, as learning to multiply inputs items is a challenge for neural networks.

3.1. Problem definition.

The goal of our network is to classify 3D objects given as point clouds embedded in \( \mathbb{R}^3 \). A given point cloud \( X \) is defined as a cloud of \( n \) points, where each point is described by its coordinates in \( \mathbb{R}^3 \). That is, \( X = \{ x_1, \ldots, x_n \} \), where each point \( x_j \), is given by its coordinates \( (x_j, y_j, z_j)^T \). The output of the network, should allow us to select the class \( Y \in L \), where \( L = \{ 1, \ldots, L \} \) is a set of \( L \) labels. For a neural network defined by the function \( g : \mathbb{R}^{n \times 3} \to \mathbb{R}^L \), the desired output is a score vector in \( \mathbb{R}^L \), such that \( Y = \arg \max_{i \in L} g(X; t) \).

3.2. Geometric Moments

Geometric moments of increasing order represent distinct spatial characteristics of the point cloud distribution, implying a strong support for construction of global shape descriptors. By definition, first order moments represent the extrinsic centroid; second order moments measure the co-variance and can also be thought of as moments of inertia. Second order moments of a set of points \( X \subseteq \mathbb{R}^3 \) can be compactly expressed in a \( 3 \times 3 \) symmetric matrix \( \Sigma \), Eq. (3), where \( x_j \in X \) defines a point given as a vector of its coordinates \( x_j = (x_j, y_j, z_j)^T \).

\[
\Sigma = \sum_j x_j x_j^T.
\]  

(3)

Roughly speaking, we propose to relate between point clouds by learning to implicitly correlate their moments. Explicitly, the functions \( (x^2, y^2, z^2, xy, xz, yz) \) of each point are given to a neural network as input features in order to obtain better accuracy.

Geometric transformations. A desired geometric property is invariance to rigid transformations. Any rigid transformation in \( \mathbb{R}^3 \) can be decomposed into rotation and translation transformations, each defined by three parameters [1]. A rigid Euclidean transformation \( T \) operating on a vector \( v \in \mathbb{R}^3 \) has the general form

\[
T(v) = R \cdot v + t,
\]  

where \( R \) is the rotation matrix and \( t \) is the translation vector.

Figure 2. The first and second geometric moments displayed on a point cloud. Using the first order moments (red disc), the translation ambiguity can be removed. The principal directions \( d_1, d_2, d_3 \) (blue arrows) are defined by the second order geometric moments. Adding these moments to the input helps the network to resolve the rotation ambiguity.

Once translation and rotation are resolved, a canonical form can be realized. The pre-processing procedure translates the origin to the center of mass given by the first order moments, and scales it into a unit sphere compensating for variations in scale. The rotation matrix, determined by three degrees of freedom, can be estimated by finding the principal directions of a given object, for example see Figure 2. The principal directions are defined by the eigenvectors of the second order moments matrix \( \Sigma \), see Eq. (5). They can be used to rotate and translate a given set of points into a canonical pose, where the axes align with directions of maximal variations of the given point cloud [12]. The first principal direction \( d_1 \), the eigenvector corresponding to the largest eigenvalue, is the axis along which the largest variance in the data is obtained. For a set of points \( X \subseteq \mathbb{R}^3 \), the \( k \)th direction can be found by

\[
d_k = \arg \max_{\|d\|=1} d^T (X_k X_k^T) d,
\]  

(5)

where

\[
X_k = X - \sum_{s=1}^{k-1} d_s d_s^T X.
\]  

(6)
3.3. Approximation of polynomial functions.

In the suggested architecture we added low order polynomial functions as part of the input. The question arises whether a network can learn polynomial functions, obviating the need to add them manually.

Here, we first provide experimental justification to the reason that one should take into account the ability of a network to learn such functions as a function of its complexity. Mathematically, we examined the ability of a network $f(w, x)$ to approximate $f(x)$, where $w$ denotes the network parameters, such that

$$\arg \min_{f(w, x)} \| f(w, x) - f(x) \|_{L_{\infty}} < \epsilon,$$  \hspace{1cm} (7)

for a given function $f : \mathbb{R} \rightarrow \mathbb{R}$. Theoretically, according to [13][2], there exists a ReLU network that can approximate polynomial functions up to the above accuracy on the interval $[0, 1]$, with network depth, number of weights, and computation units of $O(\epsilon^{-1})$ each.

In order to verify these theoretical claims, we performed experiments to check whether a network can learn the geometric moments from the point cloud coordinates. Figure 3 shows an example for $f(x) = x^2$ and it’s approximation by a simple ReLU network.

The pipeline can be described as follows. First we consider uniform sampling of the interval of $[0, 1]$; the number of samples was chosen experimentally to be 1,000 samples. Next, we arbitrarily chose the number of nodes in each layer to be 4, each with ReLU activation, using fully connected layers (in contrast to the suggested network where we perform MLP separately per point). Lastly, we set the weight initialization to be taken from a normal distribution.

Ideally, networks with a larger number of layers could better approximate a given function. However, it is still a challenge to train such networks. Our experiments show that although a two layer network has achieved the theoretical bound, the network had difficulty to achieve more accurate results than $L_{\infty} = 0.006$ even when we increased the number of layers above 6. Furthermore, we tried to add skip connections from the input to each node in the network; however, we did not observe a significant improvement.

Comparing two point clouds by comparing their moments is a well known method in the geometry processing literature. Yet, we have just shown that the approximation of polynomial functions is not trivial for a network. Therefore, adding polynomial functions of the coordinates as additional inputs could allow the network to learn the elements of $\Sigma$ in eq.[3] and, assuming consistent sampling, should better capture the geometric structure of the data.

3.4. Mo-Net Architecture

The baseline architecture of the suggested Mo-Net network is based on the pointNet architecture, with three main modifications. (1) we add polynomial functions as part of the input point cloud, (2) we reduce the number of MLP layers to only one layer with 512 features, and (3) we concatenate average pooling to the existing max pooling operation, which is also a symmetric operation. Justifications for these architectural changes are provided in the next paragraphs. The architecture of Mo-Net compared to the pointNet and to a plain network, that is the same as Mo-Net but without the polynomial functions as input, is shown in Figure 4. The Mo-Net architecture demonstrates improvement in terms of computational and memory efficiency. Compared to pointNet, computational complexity (measured in number of flops) was dropped by two orders of magnitude and Mo-Net memory requirement is 20% less that of pointNet. Classification accuracy evaluated on two benchmarks was also improved.

Our main contribution is the simple addition of powers of the coordinates of each point to the input. We implemented this by taking the first MLP, with previously 3 elements for each point, and extended the input to 9 elements. Now, each point has nine components that represent the elements required to construct the second geometric moments.
This simple lifting allows us to reduce the number of MLP layers to a single layer with 512 features.

Another aspect is that point cloud representations describe unordered collections of points. Therefore, an important property of the network is the requirement for invariance to permutations of the order of the points. The solution suggested by Qi et al. [3] was to generate features with shared weights per point on which a symmetry function operates.

We propose to generate features from the coordinates of each point and its polynomial expansions. Then, compute two symmetric operations on the features, max-pooling and average-pooling. The expression for moments contains a summation operator see Eq. (3). As a means to help the network exploit geometric moments, we added the average pooling operator in addition to the max pooling between all the points. The output of the symmetric operations is a global feature vector containing 1024 numbers. Next, similar to pointNet, we apply two fully connected layers of sizes (512, 256) to produce a score for each class.

4. Experimental Analysis

We compared the performance of the proposed model to that of pointNet [3], as their architectures are similar and operate directly on points in \( \mathbb{R}^3 \). We used the pointNet implementation provided by the authors. The following paragraphs describe the datasets and experimental results.

4.1. Datasets

4.1.1 ModelNet40

Evaluation and comparison of the results to previous efforts is performed on the ModelNet40 benchmark [14]. ModelNet40 is a synthetic dataset composed of Computer-Aided Design (CAD) models, containing 12,311 CAD models given as triangular meshes, split to 9,843 samples for training and 2,468 for testing. Pre-processing of each triangular mesh as proposed in [3] yields 1024 points sampled from each triangular mesh using the farthest point sampling (FPS) algorithm. Rotation by a random angle, about the \( y \) axis, and additive noise are used for data augmentation. The database contains samples of very similar categories, like the flower-pot, plant and vase, for which separation is subjective rather than objective and is a challenge even for a human observer.

4.1.2 S3DIS

[15] is an indoor scene dataset, unlike ModelNet40 which is made by 3D modeling tools. S3DIS contains 6,005 objects given as point clouds labeled to 13 different categories. 256 points are sampled from each point cloud using farthest point sampling (FPS) algorithm in a similar way to Mo-Net. The main challenge with this dataset is partiality. There is a high level of occlusion due to sensor noise and limited scanning time. We eliminated four categories, floor, wall, ceiling and clutter, from the dataset. The reason is that we would like our classifier to be robust to orientations, and with the right rotation a floor is nothing but a wall or a ceiling. If required, these classes could have been trivially classified using a normal direction and moments.

4.2. Classification Performance

For comparison, we trained two pointNet versions as published by the authors. The first is a vanilla version which we compare to our Mo-Net network (baseline). The second version incorporates transformer blocks, and we compare it to Mo-Net with the transformer blocks (STN) as well. We use the preprocessing advised by the authors. The idea of adding polynomial functions to the input domain is simple, induces low time and space complexity, and achieves better results compared to those realized by pointNet. Training on ModelNet40 takes about 3 hours to converge with Tensorflow [25] and Nvidia Titan X.
| Method            | Input    | Main Operator | Mean Class Accuracy | Overall Accuracy |
|-------------------|----------|---------------|---------------------|------------------|
| 3DShapeNets [14]  | voxels   | 3D conv       | 77.3                | 84.7             |
| VoxNet [16]       | voxels   | 3D conv       | 83.0                | 85.9             |
| Subvolume [17]    | voxels   | 3D conv       | 86.0                | 89.2             |
| MVCNN [18]        | multi-view | 2D conv   | -                   | 90.1             |
| RotationNet [19]  | multi-view | 2D conv   | -                   | 97.3             |
| ECC [20]          | point cloud | Local features | 83.2                | 87.4             |
| DGCNN [21]        | point cloud | Local features | 90.2                | 92.2             |
| Kd-Networks [22]  | point cloud | Local features | 88.5                | 91.8             |
| PointCNN [23]     | point cloud | Local features | -                   | 91.7             |
| PointNet++ [24]   | point cloud | Local features | -                   | 90.7             |
| PointNet [3]      | point cloud | Pointwise MLP | 85.9                | 88.9             |
| Mo-Net (Baseline) | point cloud | Pointwise MLP | 83.3                | 87.2             |
| Mo-Net            | point cloud | Pointwise MLP | 86.1                | 89.3             |

Table 1. Comparison of classification accuracy (%) on ModelNet40.

Table[1] shows the results of the ModelNet40 classification task for various methods that assume different representations of the data and with different core operators. Table[2] compares classification results when pointNet[3] and the suggested network Mo-Net are applied to ModelNet40 and S3DIS. The results on these two benchmark datasets confirm the superiority of the Mo-Net architecture. Other points based approaches [20, 21, 22, 23, 24, 26] report better results; however, they consider features that require a support larger than a single point, or a partitioning of the input set of points in addition to the point features. It should be noted that although classification rates above 90% were reported for example in [5, 16, 17], they did not use point-clouds as input, but different data representations such as meshes, voxel-grids or multi-view images.

We also tested the effects of input and feature transformations on the results. Using spatial transformers improved the Mo-Net performance by 2.1%. We conclude that the suggested approach achieves substantially better results than pointNet, with or without the transformer blocks.

### 4.3. Variations of the Architecture

We next explore the variations in architecture with respect to ModelNet40 accuracy as a function of the hidden layer size in the MLP layer, see Table[3]. The results show that the suggested Mo-Net yields better accuracy than the plain network for all sizes of hidden layer. The only difference between the plain and the suggested Mo-Net network is the polynomial expansion of the input coordinates. Therefore, we can conclude that there is a strong relation between accuracy and the additional inputs that simplify the realization of geometric moments by the network.

| Number of nodes | plain network | Mo-Net (Baseline) |
|-----------------|---------------|-------------------|
| 16              | 84.0          | 84.6              |
| 32              | 84.8          | 85.6              |
| 64              | 85.5          | 86.5              |
| 128             | 85.7          | 86.5              |
| 256             | 85.6          | 86.7              |
| 512             | 85.7          | 87.2              |

Table 3. Comparison of accuracy on ModelNet40 between plain network to the Mo-Net suggested network for different hidden layer sizes in the MLP layer.

### 4.4. Memory and Computational Efficiency

As a result of different representations such as meshes, multi-view and voxel-grids, 3D DNN architectures present a wide range of computational requirements. For example, the MVCNN DNN architecture [18] performs 62 billion floating-point arithmetic operations to classify a point cloud.

Table[4] presents computational requirements with respect to the number of the network’s parameters (memory) and with respect to the number of multiplication-addition operations per sample (FLOPs), required by the models during the inference phase. Our results show that adding polynomial expansions to the input leads to better classification performance, as well as computational and memory efficiency that can be exploited in order to train larger models and achieve better accuracy.
Table 4. Memory in terms of the size of the model and computation efficiency in terms of floating-point operations per sample (FLOPs) for the network architectures. Here, M stands for $10^6$. The Mo-Net implementation achieved a reduction of FLOPs by more than 90% and of model size by 20% compared to pointNet architecture.

| Method       | Memory | FLOPs |
|--------------|--------|-------|
| PointNet     | 0.8M   | 150M  |
| Mo-Net       | 0.6M   | 5.4M  |
| Mo-Net (Baseline) | 3.1M | 7.8M  |

5. Conclusions

In this paper, we combined a geometric understanding about the ingredients required to construct compact shape signatures with neural networks that operate on cloud of points to leverage the network’s abilities to cope with the problem of rigid objects classification. By lifting the shape coordinates into a small dimensional, moments-friendly, space, the suggested network, Mo-Net, is able to learn more efficiently in terms memory and computational complexity and provide more accurate classifications compared to related methods. Experimental results on two benchmark datasets confirm the benefits of such a design. We showed that lifting the input coordinates of points in $\mathbb{R}^3$ into $\mathbb{R}^9$ by simple second degree polynomial expansion, allowed the network to lock onto the required moments and classify the objects with better efficiency and accuracy compared to previous methods that operate in the same domain. We showed experimentally that it is beneficial to add these expansions as a pre-processing step. We believe that the ideas introduced in this paper could be applied in other fields where geometry analysis is involved, and that the simple cross product of the input point in homogeneous coordinates with itself could improve networks abilities to efficiently and accurately handle geometric structures.

6. Acknowledge

This research was partially supported by the Israel Innovation Authority, Omek Consortium.

References

[1] Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. Numerical geometry of non-rigid shapes. Springer Science & Business Media, 2008.

[2] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. Neural Networks, 94:103–114, 2017.

[3] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 1(2):4, 2017.

[4] Jing Huang and Suya You. Point cloud labeling using 3d convolutional neural network. In Pattern Recognition (ICPR), 2016 23rd International Conference on, pages 2670–2675. IEEE, 2016.

[5] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. arXiv preprint arXiv:1608.04236, 2016.

[6] Alon Shtern and Ron Kimmel. V-flow: Deep unsupervised volumetric next frame prediction.

[7] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In Advances in Neural Information Processing Systems 28, pages 2017–2025. Curran Associates, Inc., 2015.

[8] C Lee Giles and Tom Maxwell. Learning, invariance, and generalization in high-order neural networks. Applied optics, 26(23):4972–4978, 1987.

[9] C Lee Giles, RD Griffin, and T Maxwell. Encoding geometric invariances in higher-order neural networks. In Neural information processing systems, pages 301–309, 1988.

[10] YC Lee, Gary Doolen, HH Chen, GZ Sun, Tom Maxwell, and HY Lee. Machine learning using a higher order correlation network. Technical report, Los Alamos National Lab., NM (USA); Maryland Univ., College Park (USA), 1986.

[11] Gary William Flake. Square unit augmented radially extended multilayer perceptrons. In Neural Networks: Tricks of the Trade, pages 145–163. Springer, 1998.

[12] NA Campbell and William R Atchley. The geometry of canonical variate analysis. Systematic Biology, 30(3):268–280, 1981.

[13] Shiyu Liang and R Srikant. Why deep neural networks for function approximation? Proceedings of the International Conference on Learning Representations (ICLR), 2017.

[14] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaouo Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1912–1920, 2015.

[15] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. arXiv preprint arXiv:1702.01105, 2017.

[16] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, pages 922–928. IEEE, 2015.

[17] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5648–5656, 2016.

[18] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In Proceedings of the IEEE international conference on computer vision, pages 945–953, 2015.
[19] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[20] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In Proc. CVPR, 2017.

[21] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. arXiv preprint arXiv:1801.07829, 2018.

[22] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In Computer Vision (ICCV), 2017 IEEE International Conference on, pages 863–872. IEEE, 2017.

[23] Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. Pointcnn. arXiv preprint arXiv:1801.07791, 2018.

[24] 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, pages 5105–5114, 2017.

[25] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In OSDI, volume 16, pages 265–283, 2016.

[26] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3d point cloud classification and segmentation using 3d modified fisher vector representation for convolutional neural networks. arXiv preprint arXiv:1711.08241, 2017.