Geometric Learning

048865

Geometric Moments & Neural Shape Analysis

Final Project

By: Yuval Haitman

GitHub Repository

Abstract

3D point clouds are widely used in the evolving world of computer vision especially the industry of autonomous driving and self drone navigation. One basic but important task preformed on 3D data is classification. Classification based on 3D point cloud might be very challenging since no a-priori model is given. For this task DNN might be used applied just on the point cloud information, the most famous DNN for this task is PointNet [1] . In this project we compare the performance of PointNet against one of is flavors - Momen^{e}t [2] in the task of classification preformed on ModelNet40 dataset. We suggest some variation to Momen^{e}t using new input lifting and compared their results to the benchmark. Project implementation in PyTorch.

1 Models and Architecture

First we will present the basic architecture of each model as shown in the original papers.

1.1 PointNet

Since there is no order on a point cloud dataset, the authors of PointNet [1] used functions that will yield same values for different permutations of the point cloud. To do so, they used MLP layers implemented by 1D convolution, followed by MaxPolling layers and Fully connected layers. To learn a pose invariant geometry, they used Transformation Network (TNet), that learns a canonical pose of the object. Between each layer a BatchNorm and ReLu activation functions where used. Finally dropout is preformed on the FC output and softmax classification loss. A regularization term is used for the Tnet output $\boldsymbol{A}$ show in (1). The full model architecture is shown in Figure 1.

$$L_{reg} = \|I - \boldsymbol{A}\boldsymbol{A}^T\|_F^2 \tag{1}$$

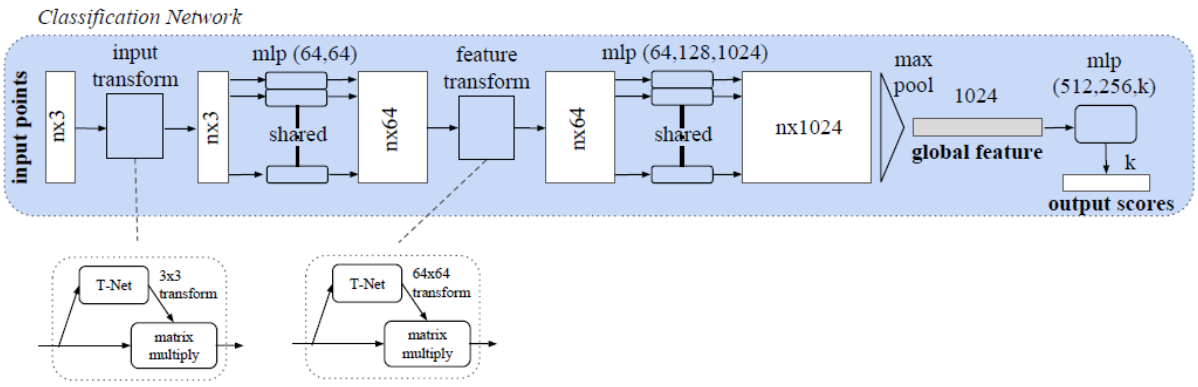

Figure 1: PointNet architecture

1.2 Momen^{e}t

The Momen^{e}t architecture [2] is a flavor of the original PointNet. Instead of using just the first order moments given by $\{x, y, z\}$ point coordinates, as use of higher order moments is done $\{x, y, z, x^2, y^2, z^2, xy, xz, yz\}$ (second order moments). As shown in the paper, approximation of polynomial functions is not trivial to the network and so, using second order moments as input lifting did improve the classification performance. Another flavor suggested in the paper is the use of k-nn component to learn local environment of each point as well. In our evaluation we sticked to the basic Momen^{e}t without the k-nn component as was asked in the exercise. Except for the second order layer, the network architecture is very similar to pointNet as shown in Figure 2.

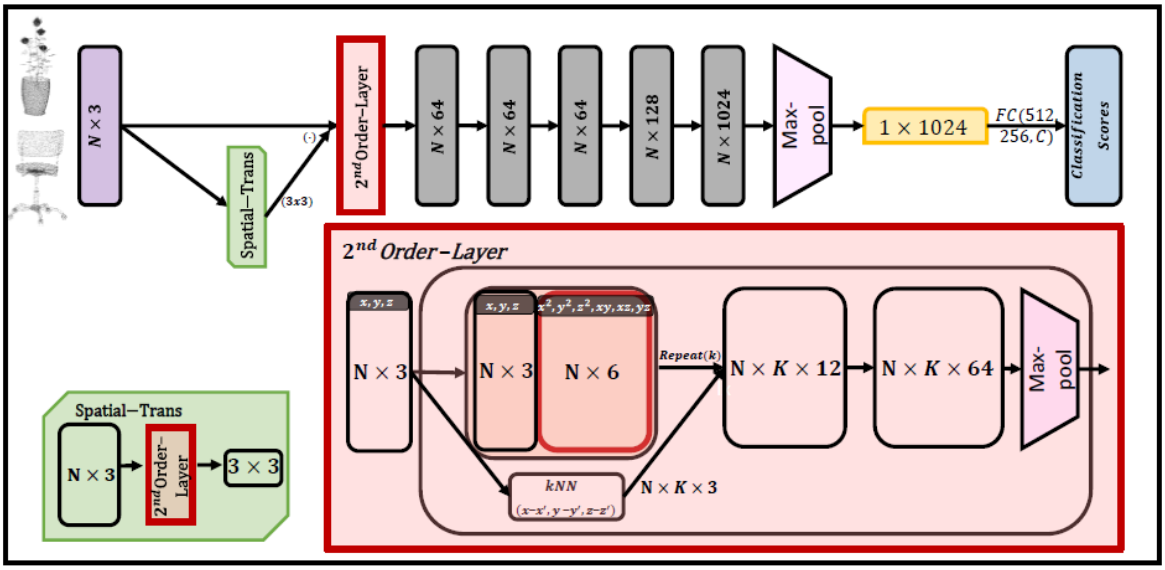

Figure 2: Momen^{e}t model architecture

2 Input Lifting

We have tested differnt input lifts: first to third moments, vertex normals and point curvature. In Figure 3 we can see a unified descriptor for the lifted input, in each test we used some of it fields therefore the descriptor size can be changed between 3-25.

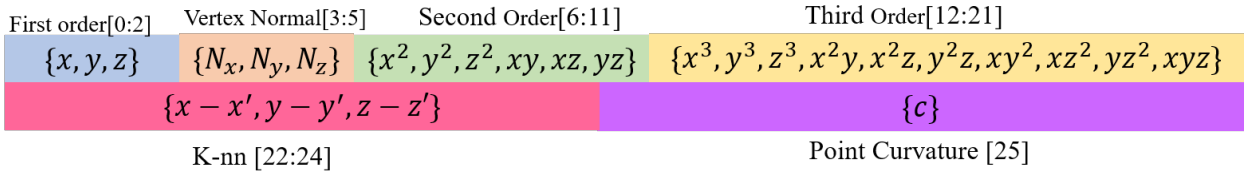

Figure 3: unified descriptor, can include some of its field, size varies between 3 to 25

2.1 Vertex Normals

In general, to find the vertex normal explicitly we will need the shape face information, but because the network doesn't have this information, the normals are needed to be estimated just from the mesh vertices. A common way to do so is by using a local patch estimation. Given a point and its neighborhood we can create a local estimator to the point covariance matrix by using (3), the normal estimator will be the eigenvector corresponds to the smallest eigenvector.

$$\widehat{\boldsymbol{\mu}}(\boldsymbol{p}) = \frac{1}{|N(\boldsymbol{p})|} \sum_{\boldsymbol{p}' \in N(\boldsymbol{p})} \boldsymbol{p}' \tag{2}$$

$$\widehat{\boldsymbol{\Sigma}}(\boldsymbol{p}) = \frac{1}{|N(\boldsymbol{p})|} \sum_{\boldsymbol{p}' \in N(\boldsymbol{p})} (\boldsymbol{p}' - \widehat{\boldsymbol{\mu}}(\boldsymbol{p}))(\boldsymbol{p}' - \widehat{\boldsymbol{\mu}}(\boldsymbol{p}))^T \tag{3}$$

2.2 Curvature

To estimate the curvature of the point cloud in each point, we used another well known estimator for the mean curvature which is given by a function of the eigenvalues of the local patch (as described in section 2.1). Denoting the eigenvalues by their order statistic $\lambda_1 \leq \lambda_2 \leq \lambda_3$, the curvature estimator will be:

$$\widehat{C}(\boldsymbol{p}) = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} \tag{4}$$

3 Training Pipeline

The traning procedure for all the described models is as follows:

1. Point Cloud Preprocessing - Each 3D CAD model from the ModelNet40 dataset was sampled randomly in 1024 points to generate a point cloud. Finally, each point cloud was resized to fit the unit sphere. As for the models using the information of the normals / curvature, we used the method described in section 2.1.

2. Augmentations - We decided to use two types of augmentations: rigid transformations and additive noise. As we used the Tnet, a canonical representation should be learned for each point clouds. Because we are dealing with point clouds of rigid objects, the use of rigid transformations on the input point clouds will make the network more robust. As for additive noise, the network will be more robust the input jittering. We used the two augmentations on the input batch with random probability of 0.5 (half of the time no augmentation is done).

3. Optimizer - we used the well known Adam optimizer since it achieved the best results. We use learning rate of 10^{-3}.

4. Hyper Parameters - We used the following hyper parameters:

Max Epoch	100
Batch Size (train)	32
Batch Size (validation)	64
Point Cloud Size	1024
Tnet Regularization	10^{-4}

Table 1: Train Parameters.

4 Results

First we will state that all models were implemented with PyTorch. We compared the classification performance using our implementation for the original PointNet, Momen^{e}t and the other asked implementations.

Base Model	K-nn	Normals	Max Moment Order	Curvature	Accuracy avg.class	Accuracy Overall
PointNet	X	X	1	X	0.844	0.875
Momen^{e}t	20	X	2	X	0.852	0.884
Momen^{e}t	20	✓	2	X	0.862	0.891
Momen^{e}t	20	X	3	X	0.842	0.885
Momen^{e}t	20	✓	3	X	0.853	0.886
Momen^{e}t	20	X	2	✓	0.850	0.886
Momen^{e}t	20	✓	2	✓	**0.865**	**0.892**
Momen^{e}t	20	✓	3	✓	0.857	0.889

Table 2: Models Comparison.

In Figure 4 we can see the train loss of all tested models. We can see that for each model we achieved a train convergence and for models that include more spatial information input we achieved even lower loss.

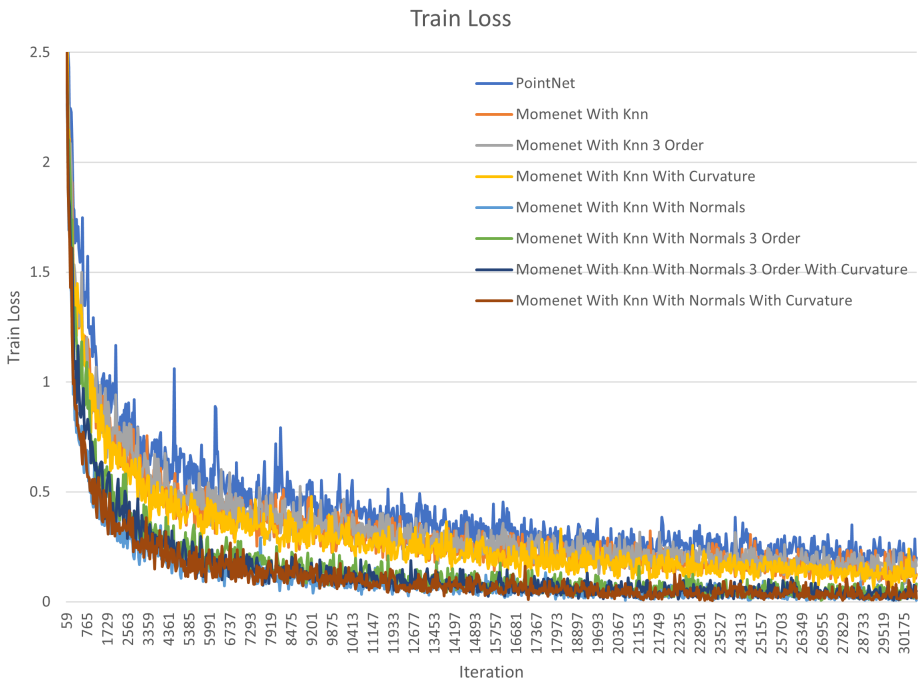

Figure 4: Train Loss

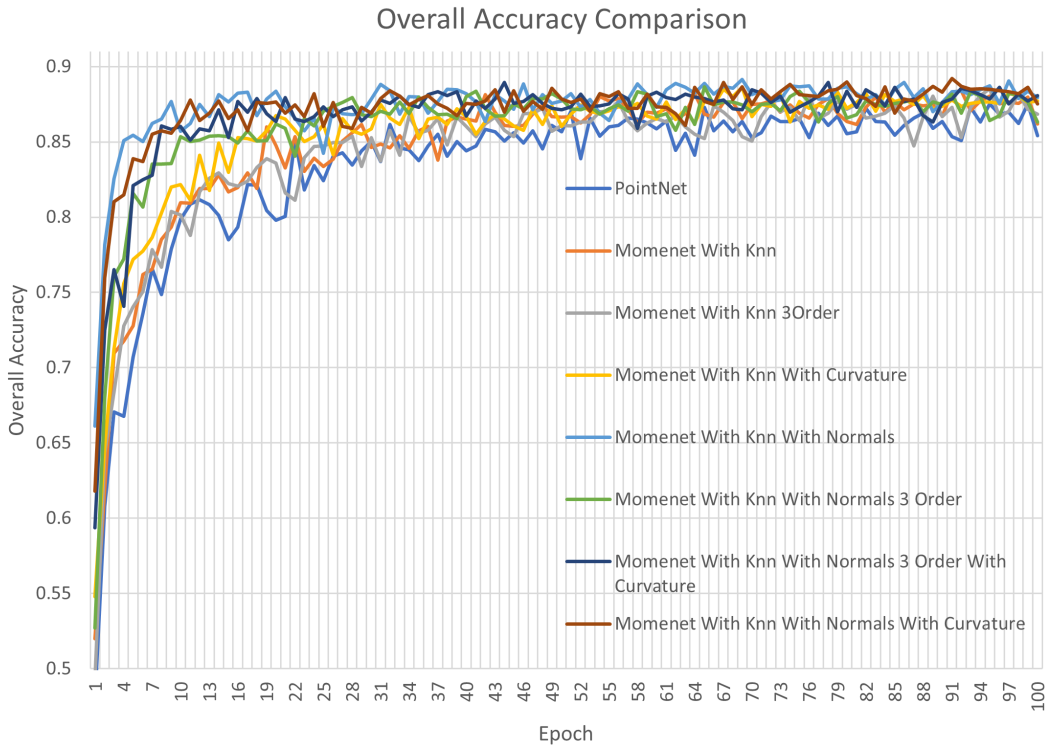

Figure 5: Overall Accuracy Comparison

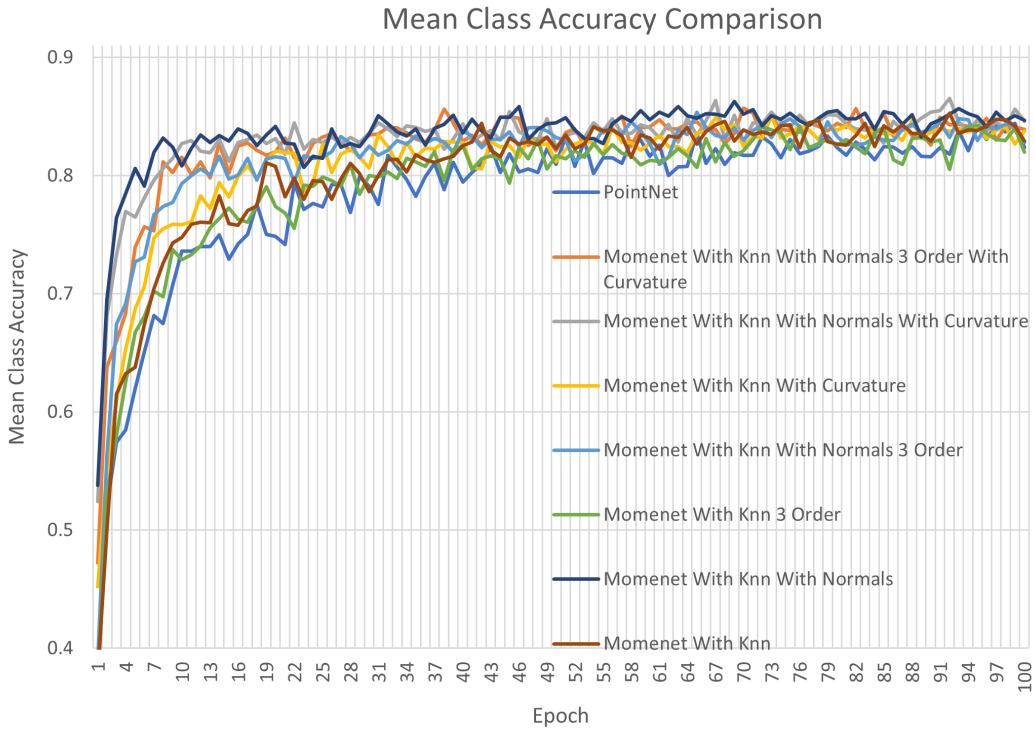

Figure 6: Mean Class Accuracy Comparison

From Table 2 and from Figures 5,6 we can see that as we add more information to the network input the accuracy results are improved. It seems that spatial geometric information such as normals and curvature significantly contribute to the accuracy performance. Indeed the best results were achieved when using Momen^{e}t with normals and points curvature. The use of 3 order moments did not improve the accuracy as we would expect, a possible reason is when we take high order moments we also significantly increase sampling noise. Although we did not reconstruct the reported result both on [1],[2], we can clearly see the trends in the accuracy values when we change the network architecture using different input lifting. Using some optimization tricks and more sophisticated augmentations my achieve better results than those reported in the papers.

References

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[2] M. Joseph-Rivlin, A. Zvirin, and R. Kimmel, "Momen (e) t: Flavor the moments in learning to classify shapes," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.