Deep Learning for Point Cloud Classification

Anirudh Patil
apaandka@unc.edu

Johnathan Leung
jleung18@unc.edu

Scott Merrill
smerrill@unc.edu

Luke Duan
lduan11@live.unc.edu

1 Introduction

As remote-sensing methods have become cheaper over recent years, the usage of three-dimensional representations for objects and scenes has quickly gained popularity. Applications range across a number of industries, from autonomous driving, to robotics, to medical imaging [2, 6, 15].

Point clouds are a common data format for storing the 3D information captured from sensors. While deep learning techniques have produced groundbreaking segmentation and classification results on 2D images [1, 12, 25, 7], there exist several challenges in applying these same techniques directly to point clouds [1].

One primary complication stems from the fact that point clouds inherently lack a prescribed order of points. This contrasts with the structure of 2D images, where pixels have a natural matrix ordering. Moreover, when modeling point clouds, it is important to enforce permutation invariance - when a model's predictions remain consistent regardless of the order in which its elements are arranged [1].

The fixed format of 2D images also makes construction of local features a more consistent operation. In point clouds, local features are often constructed by aggregating characteristics of points within a limited neighborhood or region. However, this approach presents several challenges, such as how to assign neighboring points, how to weigh points that are closer in geometric space, or how to handle the variable densities of point clouds, and how to do so efficiently.

Many works have proposed methods that improve local feature aggregation [25, 12, 11, 14, 19]. A promising approach is the Transformer self-attention architecture [18], which allows a model to draw on dependencies between different elements in the input data. The Transformer was originally applied to achieve state-of-the-art performance in natural language processing. However, transformer models have been applied to point clouds [4, 22] to aggregate local contexts and model the relationship between different points. There are still many potential areas for improvement in a transformer model, such as improvement to local feature aggregation and embedding modules, as well as efficiency enhancements in the actual attention layer.

In this report, we will investigate these challenges in greater detail and consider the state-of-the-art solutions that have been proposed. In particular, we look to explore different methods of gathering and using local features within a seminal point cloud model, the Point Cloud Transformer [4], to improve the accuracy of 3D point cloud classification while maintaining the model efficiency. [1]

2 Related Work

2.1 Deep Learning for Point Clouds

Techniques for point cloud segmentation and classification tasks can be crudely categorized into three classes: multi-view methods, volumetric methods and point-based methods [5].

[1]Our code is at https://github.com/smerrillunc/COMP775.

Multi-view methods project the point clouds to a series of 2D images and then perform typical 2D convolutions [16, 24, 11]. These methods are often the most computationally efficient, but suffer information loss and thus are sensitive to viewpoint selection. Volumetric based approaches convert point clouds to voxel grids and perform 3D convolutions on such volumetric representations [10, 11, 9]. These approaches are the most computationally demanding and become more data inefficient with increased point cloud sparsity. Finally, there are point-based methods, which work directly on point clouds [12, 1, 7]. These methods are of particular interest as they are more computationally feasible than volumetric approaches without forfeiting any explicit information loss that multi-view methods. Our work mainly addresses point-based methods.

Point-wise models. A pioneering point-based deep learning model on point clouds was PointNet [1]. This model processes a point cloud by passing point features through several fully connected layers. To enforce permutation invariance, a max pooling operation is applied in the final layer of the network. This process produces a global embedding for the point cloud, which is appended to each point vector, allowing the network to make predictions for each point with the added context of the entire point cloud. However, PointNet is unable to learn hierarchical features, as it only utilizes point-based features and a global mapping.

PointNet++ [12] augments PointNet with the ability to encode local neighborhood-based features. Instead of using a single layer like in PointNet, the model stacks multiple set abstraction layers that consist of a sampling and grouping layer along with a mini-PointNet layer. Each set abstraction layer successively subsamples the set of points while aggregating the feature vectors with a PointNet module. A visualization of this approach is shown in Figure 1.

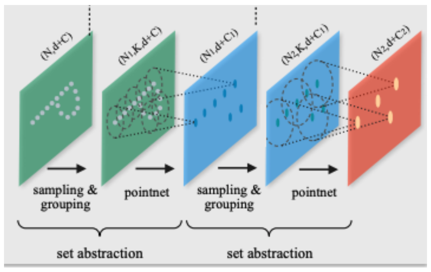

Figure 1: PointNet++ set abstraction layer from Qi et al.

To further improve local neighborhood features in PointNet++, they aggregate features at multiple scales in a scheme called Multi-Scale Grouping (MSG). When using a ball query to summarize neighborhoods, smaller radii enable more closer inspection of localized features. But, radii too small results in sample deficiencies that can corrupt local geometric patterns. To handle this, many different scales (or radii) are considered in each grouping layer and the produced embeddings are concatenated, thus enabling each centroid group to encode local features with varying receptive field sizes.

Other methods have been proposed to improve feature aggregation, such as different neighborhood sampling techniques [7] and enhancing modeling of dependencies between points and local structures [25, 3, 21].

Komachirev et al. [7] further improved the neighborhood selection process in PointNet++ with ring-based schemes, removing the overlap between mutiple scaled regions. Rather than relating each point in a particular grouping to the centroid, Zhao et al. [25] created an Adaptive Feature Adjustment (AFA) layer to summarize a local grouping by interconnecting all pairs of points in a fully connected graph. Their method uses a feature modulator that modifies the features of each vertex based on the information from the local neighborhood.

While the above works focus on methods to aggregate local features, they have not combined these with Transformer-based models, which could bring further improvements.

Local neighborhood approaches are also similar to other methods using graph neural models. In graph neural networks for point clouds [14, 19, 13], each point in the the point cloud is assigned to a vertex, edges connect vertices in a neighborhood, and vertices and/or edges have associated features. A graph neural network layer computes new features for each vertex based on aggregation

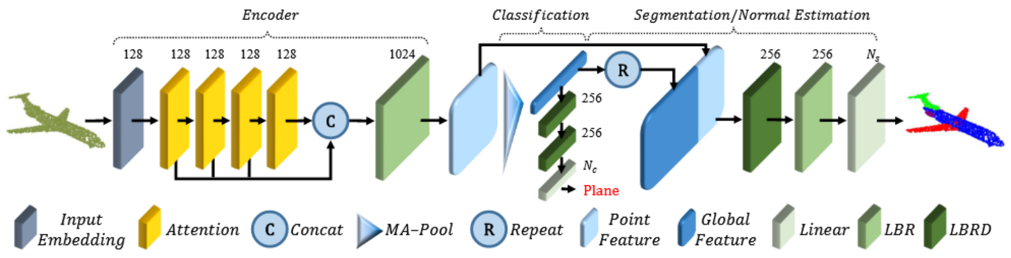

Figure 2: The Point Cloud Transformer from Guo et al. [4]. The points are passed through an input embedding layer (dark blue), then attention modules (yellow) and further processed by multiple LBR (or LBRD – LBR with dropout) layers in the classifier stage.

of information from its neighboring vertices or edges in the graph. While graph-based approaches are effective, the graph layers can become a bottleneck as the number of points and neighbors increase.

Transformers in point clouds. The transformer architecture [18] is a method for self-attention, which involves finding relationships in the form of attention weights between different elements in the input data. The Transformer was originally applied to achieve state-of-the-art performance in natural language processing. However, transformer architectures can model other types of data, as self-attention can be formulated as a set operator.

In the context of deep learning for point clouds, transformer models have been applied to point cloud embeddings to aggregate local contexts and model the relationship between different points [4, 22, 23]. This provided a robust way to calculate global relationships between a set of points. To cut down on compute time, the Induced Set Attention Block (ISAB) was introduced in the Set Transformer [8], which uses a fixed number of induced points that interact with all points in the input set, thereby reducing the number of pairwise interactions that need to be computed from quadratic to linear in the number of input points.

3 Method

We plan to borrow the base architecture of the Point Cloud Transformer (PCT) model [4] and modify it to incorporate many of the other techniques common in the point cloud literature. We first explain the model in detail and then discuss our proposed modifications.

3.1 Point Cloud Transformer (PCT)

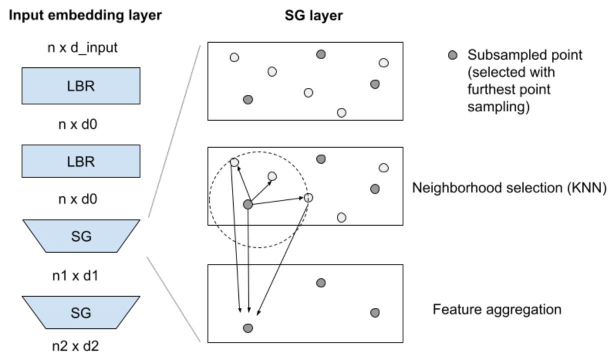

Figure 3: (Left) We illustrate the local embedding layer that encodes and subsamples points. (Right) We illustrate the operations in the Sampling and Grouping layer. The sampling layer selects a subset of points from the input point cloud using farthest point sampling, then aggregates the features within a neighborhood using KNN.

Input embedding. The model first transforms a point cloud into a new space by stacking two LBR layers. These layers apply a linear transformation independently on each input point, followed by a batch normalization (B) and ReLU (R) operation. This produces an n by d embedding of the initial point cloud. These processed points are fed into two Sample and Grouping (SG) layers, which together encode local hierarchical features. A diverse set of centroids are first sampled using the farthest point sampling (FPS) algorithm. For each neighborhood center, groups are constructed by identifying the k closets points based on squared distance. In each grouping, embeddings are expressed as relative coordinates to the center by taking the element wise difference of each feature vector to the centroid. We then concatenate this with the feature of the point p itself to get a matrix $\tilde{F}(p)$ of size $n \times 2d$. Finally two more LBR layers and a max pooling layer aggregate the features into a single vector of size $2d$. These operations are summarized below, where $F(p)$ is the input feature vector of a single point, RP denotes stacking the feature k times and MP denotes max pooling.

$$\Delta F(p) = \mathrm{concat}_{q \in \mathrm{knn}(p)}(F(q) - F(p)) \tag{1}$$

$$\tilde{F}(p) = \mathrm{concat}(\Delta F(q), \mathrm{RP}(F(p), k)) \tag{2}$$

$$F_s(p) = \mathrm{MP}(\mathrm{LBR}(\mathrm{LBR}(\tilde{F}(q))) \tag{3}$$

In the original implementation, the input number of points is 1024, the first layer subsamples 512 points (with 128 dimensional features), and the second layer subsamples 256 points with 256 dimensional features. The k in the k nearest neighbors step is set to 32 points.

Attention layer. These encoded points are then passed through multiple self-attention layers based on the Transformer architecture [18], which captures the relations between the point embeddings. The Transformer architecture is a method to compute self-attention, which calculates affinities between different items in an input data. The architecture is as follows. First, the model takes in a set of embeddings F of dimensions $n \times d$. It is multiplied by trainable weights to give query, key and values Q, K, V, which are $n \times d_a, n \times d_a, n \times d_e$ matrices, respectively. Specifically, $Q = W_q \cdot F, K = W_k \cdot F, V = W_v \cdot F$ for weights W_q, W_v, W_k. In the original model, d_a is 128 and d_e is 32. Finally, the Q and K matrices are multiplied together into QK^T and a scaled softmax is applied to the product. This softmaxed output is the dot product between every pair of query and key values. The output is subsequently multiplied with the value matrix. The final output is

$$F_{\mathrm{sa}} = \mathrm{Att}(\mathrm{Q}, \mathrm{K}, \mathrm{V}) = \mathrm{softmax}(QK^T/\sqrt{d})V \tag{4}$$

where d is the dimension of the vector.

The PCT [4] uses a slightly modified formulation of the Transformer. Instead of directly using the output of the self-attention layer, this model specifically calculates the element-wise subtraction between the input F_{in} and the attention output F_{sa} in an operation called offset attention. Thus, their output is:

$$F_{\mathrm{out}} = \mathrm{LBR}(F_{\mathrm{in}} - F_{\mathrm{sa}}) - F_{\mathrm{in}} \tag{5}$$

where F_{sa} is the output from the self-attention module and F_{in}. This idea is inspired by graph convolutional networks, where the adjacency matrix E is replaced with a graph Laplacian matrix $L = D - E$ to enhance the model. D represents the degree of the graph.

Classification. The output of the transformer is followed by a stack of linear layers to compute classification scores.

The modifications we make can be categorized in into 4 categories: sampling, grouping, feature construction, and the attention mechanism which are discussed in the following subsections.

3.2 Sampling

While the FPS algorithm provides good coverage, its runtime is $O(nk)$, where n is the number of points and k is the number of points subsampled. Furthermore, given the extreme sparsity inherent in many point cloud scans, this algorithm results in many of the sparse outlier points being selected as feature aggregation centers. From the literature, it is unclear whether such a strategy is preferred over simpler, more efficient strategies. Thus, we propose the use of random sampling and a voxel-based method, which can be seen as a hybrid of FPS and random sampling.

The voxel method evenly divides the point cloud into a regular grid of voxels and selects one representative point from each voxel to form our subset. It's not guaranteed that every grid cell will

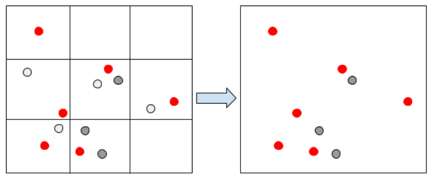

Figure 4: Voxel nearest neighbors. We select one representative point from each voxel to form our subset (red). Because not every grid cell will contain a point, but we still want enough points to feed into the subsequent layers, we randomly select enough points to fill up the rest of the set (gray).

contain a point yet we still want to create features for k neighborhoods. To resolve this issue, if a grid cell lacks any points we choose on point randomly from the point cloud to represent the neighborhood of that grid cell.

We considered the FPS algorithm using a single layer of voxel sampling with 256 points, as well as stacking two voxel sampling layers; the first of which sub-samples to 512 points with a $8 \times 8 \times 8$ voxel grid and the second subsamples to 256 points with a $6 \times 6 \times 6$ voxel grid.

3.3 Grouping

Grouping refers to the task of identifying good neighboring points with respect to a certain cluster center. Different neighborhood grouping strategies can result in vastly different local feature embeddings, which may have a substantial effect on the final classification accuracy of the PCT. Thus, to optimize the performance of PCT it is prudent to exhaustively try many grouping strategies.

The default PCT model uses the k-nearest neighbor (KNN) algorithim and squared distance to determine the closest points. To explore different grouping strategies, we first consider Minkowski Distance, which is defined as $(\sum_{i=1}^{3}(a_i - b_i)^p)^{1/p}$ for pairs of 3D point coordinates $\mathbf{a}, \mathbf{b}$. Note that when $p = 2$, the formula is equivalent to the Euclidean squared distance. We tried this with varying orders of p. Additionally, we tried modifying this distance function to cosine similarity, which geometrically can be interpreted as measuring the angle between two vectors.

Another strategy we took to identify local regions was to use a ball query. This algorithm will find the first k points that are within a radius r of the query point. This neighborhood search strategy is common in the point cloud literature and also plays nicely with the idea of Multi-Scale grouping (MSG), originally proposed by PointNet++ [12]. The idea of MSG is simple: in each grouping layer we perform a ball query over multiple radii $\{r, r + x, r + 2x, ...\}$ or $\{r, rx, rx^2, ...\}$ to identify neighborhoods within each receptive field. Each of these neighborhoods are concatenated into a single vector allowing the network to learn from features at multiple different scales. We implemented this idea and tested many combinations of 2 and 3 radii.

3.4 Feature Construction

The local feature embeddings are a function not only of the neighborhood groupings, but also how the neighborhood points are represented in each grouping. The default PCT model converts all points in a neighborhood to relative coordinates with respect to the neighborhood center. To test the robustness of PCT with respect to how features are encoded, we also tried absolute values and higher order exponents of these relative coordinates. Further we applied both Minkowski distance and cosine similarity to the feature vectors. Note that Minkowski distance and cosine similarity are functions from $\mathbb{R}^n \to \mathbb{R}$, whereas the element-wise difference is a function from $\mathbb{R}^n \to \mathbb{R}^n$. Moreover, these functionals will essentially summarize the difference vectors and reduce the cardinality of feature embeddings allowing for modest training improvements.

3.5 Attention Mechanism

We also considered a self-attention mechanism borrowed from the Set Transformer. The Set Transformer model introduces innovative modules for dealing with sets in neural networks, the Induced Set Attention Block (ISAB), an extension of a self-attention block. These modules are crucial for

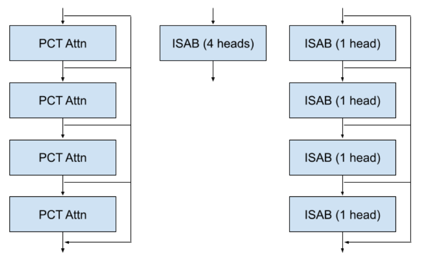

Figure 5: The configurations of transformers we want to test: (Left) the original transformer, (Middle) a single ISAB with 4 heads, (Right) four ISABs with 1 head each. We keep the residual connections as the original attention module contained residual connections.

modeling interactions within sets efficiently, providing a framework for attention-based, permutation-invariant neural networks.

The time complexity of a self-attention mechanism is $O(n^2)$, where n is the number of elements in the set. This complexity arises because self-attention needs to compute pairwise attention scores for all pairs of elements in the input set, which can be computationally intensive for large sets [8].

To address the computational challenges posed by self-attention, the Induced Set Attention Block (ISAB) was introduced. ISAB significantly reduces the time complexity from $O(n^2)$ to $O(mn)$, where m is the number of induced point vectors which are trainable parameters (denoted as I). As mentioned before, this reduction is achieved by using a fixed number m of induced points that interact with all points in the input set, thereby reducing the number of pairwise interactions that need to be computed. The ISAB is conceptually similar to low-rank projection or autoencoder, where the input is projected into low dimensions and then reconstructed into an output. The operation of ISAB is defined as:

$$\mathrm{ISAB}_m = \mathrm{MAB}(X, \mathrm{MAB}(I, X)) \tag{6}$$

where a Multi-head Attention Block (MAB) is applied twice: first between the induced points and the input, and then between the input and the output of the first MAB. This approach can be seen as performing a low-rank projection of inputs, allowing the module to efficiently capture global properties of the set [8].

In our experiments, we replaced the attention layer in the Point Cloud Transformer with the ISAB from the Set Transformer paper to measure if Set Transformers could enhance the learning of the network. In the original paper, the authors used four offset attention layers stacked with multiple skip connections. We specifically considered the following architectures: The first architecture (ISAB 1) contained a single ISAB with four heads. The second (ISAB 2) was to stack four ISAB layers with one head each in a similar way to the original model. See Figure 5.

4 Results

We implemented and tested several modifications on the model. Following the original paper, the model was tested on the ModelNet40 dataset for point cloud classification [20]. This dataset contains point clouds from CAD models with 40 class labels. The input point clouds contain 1024 points, following the original setup. The second dataset that we considered was the ScanObjectNN dataset [17]. In contrast to ModelNet40, this dataset contains real-world scans, grouped into 15 classes. Hence, there would be some noise and artifacts from the LiDAR scanning process. Our classification accuracies, and training time per batch of 16 can be seen in Tables 1 - 4.

4.1 Sampling

Table 2 shows results replacing the FPS algorithm with random sampling and the proposed voxel-based methods. While random subsampling falls below the accuracy of the original PCT model, we note substantial improvements in training time. Furthermore, the voxel-based method offers encouraging results as it offers substantial training improvements while maintaining high accuracies. This suggests that FPS may be an inefficient use of computation resources if similar results can be achieved with our proposed voxel method.

Table 1: We tested the model when (Top) using different distances for the KNN downsampling and (Bottom) using MSG with multiple ball query radii. The list of radii are specified in the left column. Training times are averaged per batch of 16 objects.

	ModelNet40		ScanObjectNN	
Model	Acc	Time	Acc	Time
Original	**0.914**	0.216s	0.801	0.163s
Mink $p = 1$	0.912	**0.210s**	**0.819**	**0.157s**
Mink $p = 3$	**0.913**	**0.212s**	**0.809**	**0.158s**
Cos sim	0.906	**0.212s**	0.722	**0.158s**
MSG				
0.1 0.2 0.3	0.906	0.267s	0.797	**0.206s**
0.05 0.1 0.2	0.915	0.259s	0.800	0.211s
0.1 0.25 0.5	**0.916**	0.248s	**0.809**	**0.209s**
0.2 0.3	**0.913**	**0.230s**	0.793	0.181s
0.1 0.2	0.910	**0.231s**	**0.802**	0.188s
0.25 0.5	0.908	0.234s	0.786	0.187s

Table 2: We tested the model after replacing the furthest-point sampling with different subsampling methods.

	ModelNet40		ScanObjectNN	
Model	Acc	Time	Acc	Time
Random	0.907	0.086s	0.798	0.062s
Voxel 2 layers	0.911	0.090s	**0.804**	0.069s
Voxel 1 layer	**0.916**	**0.061s**	0.768	**0.039s**

4.2 Grouping

Table 1 shows the results from modifying the KNN distance function. We see that using Minkowski Distance with $p = 3$ or $p = 1$ does not result in substantial changes in performance. However, we do notice that cosine similarity performs very poorly on ScanObjectNN. This large decrease in accuracy is quite encouraging as it indicates that the choice of neighbors in feature construction is critical to classification performance. Moreover, these results motivate additional research into alternative neighborhood identification strategies.

Combining features of different scales with MSG layers indicate modest improvements. In particular, our model which considers 3 radii with absolute units $r_1 = 0.1, r_2 = 0.25, r_3 = 0.5$ appears to offer small improvements over the original PCT in both datasets. These radii correspond to absolute units and were not tuned to the dataset. Moreover, setting both the number and scale of the radii in a more clever way agnostic to the scale of the data may be an interesting area of future work.

4.3 Attention Mechanism

For the ModelNet40 dataset, ISAB 1 achieved a classification accuracy of 0.914, while ISAB 2 scored slightly lower at 0.913. This marginal difference suggests that adding more ISAB layers does not significantly improve performance on this dataset. It could indicate that a single ISAB layer is sufficient to capture the necessary features for effective classification in this context, possibly due to the inherent structure or complexity of the ModelNet40 dataset.

On the ScanObjectNN dataset, the results are slightly different. ISAB 1 achieved an accuracy of 0.793, while ISAB 2 improved upon this with a score of 0.803. This improvement with additional ISAB layers implies that the ScanObjectNN dataset might be more complex or varied, requiring more layers to effectively capture the nuances in the data.

These results suggest that the utility of multiple ISAB layers in point cloud classification depends on the specific characteristics of the dataset. For simpler datasets like ModelNet40, a single ISAB layer may be adequate, while more complex datasets like ScanObjectNN can benefit from additional layers.

Table 3: We tested the model after replaced ISAB.

	ModelNet40		ScanObjectNN	
Model	Acc	Time	Acc	Time
ISAB 1 (one ISAB layer)	**0.914**	**0.208s**	0.793	**0.147s**
ISAB 2 (four ISAB layers)	0.913	0.215s	**0.803**	0.159s

Table 4: We tested the model on multiple combinations of modifications. All models have a 2 layer voxel model.

	ModelNet40				ScanObjectNN			
	1024 pts		256 pts		1024 pts		256 pts	
Model	Acc	Time	Acc	Time	Acc	Time	Acc	Time
Original	**0.914**	0.216s	**0.904**	0.116s	**0.801**	0.163s	**0.769**	0.046s
0.1 0.2, ISAB 1	0.907	**0.087s**	0.892	**0.079s**	**0.784**	**0.087s**	**0.763**	**0.030s**
0.1 0.25 0.5 ISAB 1	**0.913**	0.114s	0.895	**0.082s**	0.757	0.114s	0.750	**0.034s**
0.1 0.2, ISAB 2	0.899	**0.100s**	0.896	0.089s	0.759	**0.100s**	0.761	0.035s
0.1 0.25 0.5 ISAB 2	0.901	0.119s	**0.898**	0.093s	0.746	0.119s	0.746	0.041s

This observation underscores the importance of tailoring the depth of the model to the specific data being processed. Overall, ISAB's attention mechanism proves to be a robust approach for point cloud classification, adaptable to varying levels of data complexity.

4.4 Combination

Finally, we tried implementing multiple combinations of modifications. We first tested using Voxel method and ISAB 1, and Voxel and ISAB 2, with the best 2 radii and 3 radii configuration in MSG. We also tested robustness to fewer points by reducing the number of input points from 1024 to 256 points and corresponding reducing the width of the neural network model. There are more major drops in accuracy for some of the models but these models are more efficient than the original models. However, these models still performed worse than the model with 2 voxel layers without further modifications. We specifically note that the models with radii $r_1 = 0.1, r_2 = 0.2$ perform better than the models with radii $r_1 = 0.1, r_2 = 0.25, r_3 = 0.5$ on ScanObjectNN. Furthermore, the model with radii $r_1 = 0.1, r_2 = 0.2$, and ISAB 1 was the fastest model and had the highest accuracy on 256 points.

5 Conclusion

Overall, we saw that enhancing the local feature aggregation modules within transformer-based point cloud model led to promising improvements in performance. Furthermore, the results point to potential directions for further investigation.

The competitive results from adding a MSG using ball queries offer a compelling future area of study to integrate Annular Convolutions. As discussed previously, in a MSG architecture, ball queries result in the potential for the same point to be present in multiple scaling groups. Instead of using a ball query, the authors proposed two potential ring-based schemes to create neighborhood groupings - regular rings and dilated rings. Komarichev et al. [7] demonstrate how reducing this redundancy improves classification accuracies on their PointNet++ inspired network architecture. Applying the same Annular Convolution operation to our PCT inspired network may offer similar improvements.

Another exciting future direction is to borrow ideas from Zhao et al. [25] and incorporate Adaptive Feature Adjustment (AFA) layers. These feature transformation layers offer much more flexibility; they can be seen to connect all neighbors in a neighborhood in a fully connected graph rather than just connecting each neighbor with the query point. And, producing features based on all the neighbors in a region seems more intelligent and can possibly create better features embeddings that improve classification scores.

References

[1] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[2] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016.

[3] Yueqi Duan, Yu Zheng, Jiwen Lu, Jie Zhou, and Qi Tian. Structural relational reasoning of point clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 949–958, 2019.

[4] Meng-Hao Guo, Junxiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R. Martin, and Shi-Min Hu. PCT: point cloud transformer. *CoRR*, abs/2012.09688, 2020.

[5] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *CoRR*, abs/1912.12033, 2019.

[6] Pileun Kim, Jingdao Chen, and Yong K. Cho. Slam-driven robotic mapping and registration of 3d point clouds. *Automation in Construction*, 89:38–48, 2018.

[7] Artem Komarichev, Zichun Zhong, and Jing Hua. A-CNN: annularly convolutional neural networks on point clouds. *CoRR*, abs/1904.08017, 2019.

[8] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. 2019.

[9] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel CNN for efficient 3d deep learning. *CoRR*, abs/1907.03739, 2019.

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

[11] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. *CoRR*, abs/1604.03265, 2016.

[12] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

[13] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Neighbors do help: Deeply exploiting local structures of point clouds. *CoRR*, abs/1712.06760, 2017.

[14] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. *CoRR*, abs/1704.02901, 2017.

[15] Martin Sinko, Patrik Kamencay, Robert Hudec, and Miroslav Benco. 3d registration of the point cloud data using icp algorithm in medical image analysis. In *2018 ELEKTRO*, pages 1–6, 2018.

[16] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. *CoRR*, abs/1505.00880, 2015.

[17] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. *CoRR*, abs/1908.04616, 2019.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018.

[20] Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. *CoRR*, abs/1406.5670, 2014.

[21] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. *CoRR*, abs/2003.00492, 2020.

[22] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. *CoRR*, abs/1904.03375, 2019.

[23] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. *CoRR*, abs/1904.03375, 2019.

[24] Tan Yu, Jingjing Meng, and Junsong Yuan. Multi-view harmonized bilinear network for 3d object recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 186–194, 2018.

[25] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5560–5568, 2019.