Few-shot 3D Point Cloud Semantic Segmentation of Indoor Scenes

Mayur Mankar (IISER Bhopal)

1 Abstract

3D point cloud semantic segmentation aims to group all points into different semantic categories, which benefits important applications such as point cloud scene reconstruction and understanding. Existing supervised point cloud semantic segmentation methods usually require large-scale annotated point clouds for training and cannot handle new categories. Few-shot learning method addresses these two problems. To mitigate these limitations, we are trying to propose an improved version of attention-aware multi-prototype transductive few-shot point cloud semantic segmentation method to segment new classes given a few labeled examples. Experiments on two benchmark data sets: **S3DIS**[1] and **Scannet**[2] in various few-shot point cloud semantic segmentation scenarios are done to demonstrate significant and consistent improvements over baselines.

Figure 1: 3D Point Cloud

2 Introduction

3D point clouds play an important role in many computer vision and computer graphics applications. Given a complex scene, the captured point cloud may mix many different objects, structures and background. For better scene reconstruction and understanding, a critical step is point cloud semantic segmentation, which classifies each point into its underlying semantic category. Recent state-of-the-art methods for point cloud semantic segmentation are mainly based on supervised deep learning. However, these methods have two major limitations: 1) they rely on large-scale training data, which require costly and laborious manual annotation, and 2) the trained model cannot segment new categories that are not seen during the training process.

Few-shot learning is a promising direction that allows the model to generalize to new classes with only a few examples. In few-shot point cloud segmentation, our goal is to train a model to segment new classes given a few labeled point clouds, as illustrated in Figure 2. We propose an improved version of attention-aware multiprototype transductive inference method for few-shot point cloud semantic segmentation.

In attMPTI[21], they adopt DGCNN[17], a dynamic graph CNN architecture, as the backbone of feature extractor to respectively produce local geometric features (outputs of the first EdgeConv

layer) and semantic features (outputs of the feature extractor). But DGCNN provide limited segmentation performance due to its crude learning architecture or local aggregation method. Here we are trying to improve feature extraction of point clouds by introducing adversarial graph convolutional network (AGCN)[9] in place of dynamic graph CNN (DGCNN)[17] in attention-aware multi-prototype transductive inference (attMPTI)[21] method.

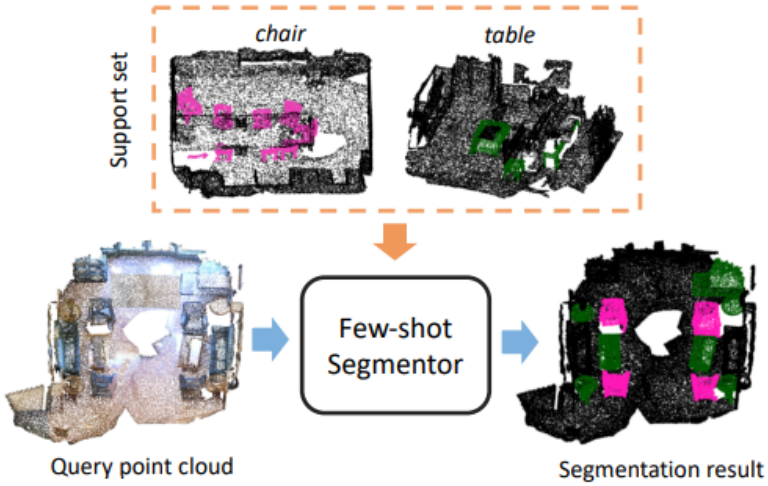


Figure 2: Few-shot point cloud semantic segmentation task is to learn a segmentor that segments the query point cloud in terms of new classes with learned knowledge from the support examples. This figure illustrates an example with 2-way 1-shot setting.

3 Previous Works

3.1 3D Semantic Segmentation

Many deep learning based approaches[6, 7, 10, 12, 13, 17, 18, 20] are proposed to tackle 3D semantic segmentation using full supervisions, i.e. point-wise ground truths. PointNet[13] is the first work that designs an end-to-end deep neural network to segment raw point clouds instead of their transformed representations, e.g. voxel grids and multi-view images. PointNet achieves permutation invariance of points by operating on each point independently and subsequently applying a symmetric function to accumulate features. This independence, however, neglects the geometric relationships among points, presenting a fundamental limitation that cannot capture local features.

DGCNN[17] addresses this issue by designing the EdgeConv module, which captures local geometric structure while maintaining permutation invariance. Instead of generating point features directly from their embeddings, EdgeConv generates edge features that describe the relationships between a point and its neighbors. EdgeConv is designed to be invariant to the ordering of neighbors, and thus is permutation invariant. Because EdgeConv explicitly constructs a local graph and learns the embeddings for the edges, the model is capable of grouping points both in Euclidean space and in semantic space.

Both PointNet and DGCNN have proposed different point convolutions, but most of them still provide limited segmentation performance due to their crude learning architectures or local aggregation methods. These works consist of a single segmentation network that predicts labels for each point independently, and thus it tends to produce inconsistent labels such as misclassification in boundaries, noisy label within a class or misclassification of a class. Moreover, their convolution techniques are not designed to aggregate sufficient local features to learn complex structures, resulting in poor segmentation accuracy.

AGCN[9] improves semantation accuracy without extra computational burden. Adversarial Graph Convolution Network (AGCN) trains two networks, a segmentation network and a discriminator network, in an adversarial manner where the discriminator network calculates a difference between two respective embedding features of ground truth map and predicted label map from the segmentation network in its last convolution layer to train the segmentation network. This adversarial training helps improve the segmentation accuracy as well as the training stability of segmentation network by enabling the network to learn high level features of ground truth labels that are smooth and consistent. Additionally, they propose a new graph convolution, which is a geometry-preserving edge convolution, abbreviated as GeoEdgeConv. It is designed to aggregate rich local geometric features such as geometric shape or structure by explicitly incorporating both edge features and relative positions between a point and neighbouring points. This allows the network to reduce noisy labels as well as improve segmentation accuracy in boundaries by enabling the network to learn fine-detailed geometry of complex structures.

Although these fully supervised approaches achieved promising segmentation performance, their requirement for large amounts of training data precludes their use in many real-world scenarios where training data is costly or hard to acquire. Moreover, these approaches can only segment a set of predefined classes that are seen during training. To alleviate these limitations, we explore the direction of few-shot learning for 3D semantic segmentation. This enables the model to segment new classes by seeing just a few labeled samples.

3.2 Generative Adversarial Networks

Since[5], There have been many studies applying GANs to different applications. While many studies[8, 14, 22] focused on improving image generations, some studies proposed hybrid loss functions where a discriminator network is used to improve segmentation performance. Although there has been very little research that uses GANs for point cloud segmentation, a few studies such as [4] and [11] proposed adversarial training for point cloud segmentation. [4] constructed a discriminator inspired from T-Net and fed statistical data calculated from each segmented tooth into the discriminator to reduce training time. However, their architectures are inspired from either [12] or [13], lacking in extracting fine local features. Additionally, as [3] stated, they used binary cross entropy (BCE) for adversarial training, which is not sufficient to train the network in stable and effective manners as it only provides a single binary prediction. Since then, there has not been any other work using different adversarial loss functions in 3D point cloud segmentation.

3.3 Few-shot Learning

The goal of few-shot learning is to develop a classifier that is able to generalize to new classes with very few examples (e.g. one example for the one-shot case). To address this challenging few-shot learning, several meta-learning approaches have proposed to learn transferable knowledge from a collection of learning tasks and made significant progress. In particular, metric-based method is notable because of its effectiveness in directly inferring labels for unseen classes during inference. The key idea in metric-based method is to learn a good metric function which is able to produce a similarity embedding space representing the relationship between labeled and unlabeled samples. *Matching network*[16] and *Prototypical Network*[15] are two representative metric-based methods. Both methods utilize deep neural network to map the support and query sets into an embedding space, and then apply a non-parametric method to predict classes for the query based on the support. Specifically, matching network leverages the weighted nearest neighbor method that represents a class by all its support samples, while prototypical network leverages the prototypical method that represents a class by the mean of its support samples. These two non-parametric methods become two extreme ends of the spectrum of complicated to simple data distribution modeling when applied to few-shot point cloud semantic segmentation. This is because the support samples for one class in the point cloud counterpart can contain a large number of points. In attMPTI, they represent each class in point clouds somewhere between the two extremes with multiple prototypes and perform segmentation in a transductive manner.

3.4 Few-shot Image Segmentation

Recently, several works started to study few-shot learning on image segmentation by extending these meta-learning techniques to pixel levels. Most existing approaches leverage on metric-based techniques to solve a one-to-many matching problem between the support and query branch, where the support sample(s) of each class is represented as one global vector. On the contrary, Zhang *et al.*][19] considers the problem as many-to-many matching, where the support branch is represented as a graph with each element in the feature map of the support sample(s) as a node. However, these few-shot image segmentation approaches learn image features by using convolution neural network (CNN) based architectures, which are not applicable to point cloud data due to the irregular structures of point clouds. Moreover, the properties of a good embedding space are different for point clouds and images. In view of the differences, in [21], they design an attention-aware multi-level feature learning network and propose a novel attention-aware multi-prototype transductive inference method for the task of few-shot 3D point cloud semantic segmentation.

3.5 PointNet

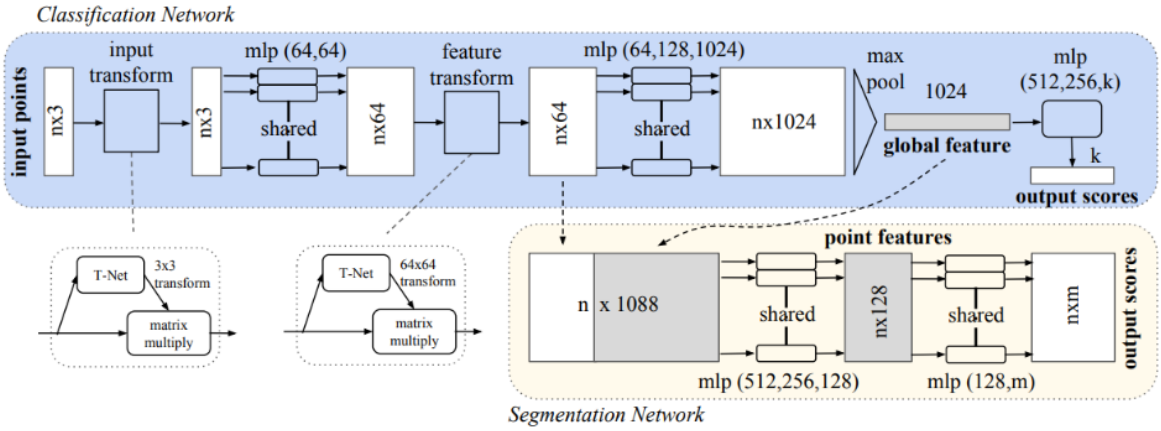


Figure 3: **PointNet Architecture**. The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores.

3.5.1 PointNet Architecture

This network has three key modules: the max pooling layer as a symmetric function to aggregate information from all the points, a local and global information combination structure, and two joint alignment networks that align both input points and point features.

Symmetry Function for Unordered Input In order to make a model invariant to input permutation, three strategies exist: 1) sort input into a canonical order; 2) treat the input as a sequence to train an RNN, but augment the training data by all kinds of permutations; 3) use a simple symmetric function to aggregate the information from each point. Here, a symmetric function takes n vectors as input and outputs a new vector that is invariant to the input order. For example, $+$ and $*$ operators are symmetric binary functions.

While sorting sounds like a simple solution, in high dimensional space there in fact does not exist an ordering that is stable w.r.t. point perturbations in the general sense. This can be easily shown by contradiction. If such an ordering strategy exists, it defines a bijection map between a high-dimensional space and a 1D real line. It is not hard to see, to require an ordering to be stable w.r.t point perturbations is equivalent to requiring that this map preserves spatial proximity as the dimension reduces, a task that cannot be achieved in the general case. Therefore, sorting does not fully resolve the ordering

issue, and it's hard for a network to learn a consistent mapping from input to output as the ordering issue persists. As shown in experiments (Figure 4), they find that applying a MLP directly on the sorted point set performs poorly, though slightly better than directly processing an unsorted input.

The idea to use RNN considers the point set as a sequential signal and hopes that by training the RNN with randomly permuted sequences, the RNN will become invariant to input order. However, order does matter and cannot be totally omitted. While RNN has relatively good robustness to input ordering for sequences with small length (dozens), it's hard to scale to thousands of input elements, which is the common size for point sets. Empirically, they have also shown that model based on RNN does not perform as well as the proposed method (Figure 4).

The idea is to approximate a general function defined on a point set by applying a symmetric function on transformed elements in the set:

$$f(x_1, ..., x_n) \approx g(h(x_1), ..., h(x_n) \tag{1}$$

where $f : 2^{\mathbb{R}^N} \to \mathbb{R}, h : \mathbb{R}^N \to \mathbb{R}^K$ and $g : \underbrace{R^N \times ... \times \mathbb{R}^K}_{n} \to \mathbb{R}$ is a symmetric equation.

Empirically, the basic module used in this paper is very simple: we approximate h by a multi-layer perceptron network and g by a composition of a single variable function and a max pooling function. This is found to work well by experiments. Through a collection of h, the model can learn a number of f's to capture different properties of the set. While the key module in this paper seems simple, it has interesting properties and can achieve strong performace in a few different applications like 3D object classification, object part segmentation and semantic scene segmentation.

Local and Global Information Aggregation The output from the above section forms a vector $[f_1, ..., f_K]$, which is a global signature of the input set. We can easily train a SVM or multi-layer perceptron classifier on the shape global features for classification. However, point segmentation requires a combination of local and global knowledge. We can achieve this by a simple yet highly effective manner.

The solution can be seen in Figure 3 (*Segmentation Network*). After computing the global point cloud feature vector, we feed it back to per point features by concatenating the global feature with each of the point features. Then we extract new per point features based on the combined point features - this time the per point feature is aware of both the local and global information.

With this modification the network is able to predict per point quantities that rely on both local geometry and global semantics. For example we can accurately predict per-point normals, validating that the network is able to summarize information from the point's local neighborhood.

Joint Alignment Network The semantic labeling of a point cloud has to be invariant if the point cloud undergoes certain geometric transformations, such as rigid transformation. We therefore expect that the learnt representation by our point set is invariant to these transformations.

A natural solution is to align all input set to a canonical space before feature extraction. We predict an affine transformation matrix by a mini-network (T-net in Figure 3) and directly apply this transformation to the coordinates of input points. The mini-network itself resembles the big network and is composed by basic modules of point independent feature extraction, max pooling and fully connected layers.

This idea can be further extended to the alignment of feature space, as well. We can insert another alignment network on point features and predict a feature transformation matrix to align features from different input point clouds. However, transformation matrix in the feature space has much higher dimension than the spatial transform matrix, which greatly increases the difficulty of optimization. They therefore add a regularization term to our softmax training loss. We constrain the feature transformation matrix to be close to orthogonal matrix:

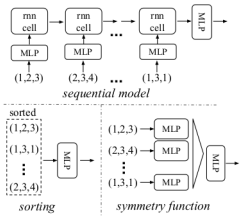
$$L_{reg} = \left\| I - AA^t \right\|_F^2 \tag{2}$$

where A is the feature alignment matrix predicted by a mini-network. An orthogonal transformation will not lose information in the input, thus is desired. they find that by adding the regularization term, the optimization becomes more stable and our model achieves better performance.

3.5.2 Comparison with Alternative Order-invariant Methods

There are at least three options for consuming unordered set inputs. They use the **ModelNet40** shape classification problem as a test bed for comparisons of those options, the following two control experiment will also use this task.

The baselines compared with include multi-layer perceptron on unsorted and sorted points as $n \times 3$ arrays, RNN model that considers input point as a sequence, and a model based on symmetry functions. The symmetry operation we experimented include max pooling, average pooling and an attention based weighted sum. Maxpooling operation achieves the best performance by a large winning margin, which validates the choice.



	accuracy
MLP (unsorted input)	24.2
MLP (sorted input)	45.0
LSTM	78.5
Attention sum	83.0
Average pooling	83.8
Max pooling	**87.1**

Figure 4: **Three approaches to achieve order invariance.** Multi- layer perceptron (MLP) applied on points consists of 5 hidden layers with neuron sizes 64, 64, 64, 128,1 024, all points share a single copy of MLP. The MLP close to the output consists of two layers with sizes 512, 256.

3.6 DGCNN

This approach is inspired by PointNet and convolution operations. Instead of working on individual points like PointNet, however, they exploit local geometric structures by constructing a local neighborhood graph and applying convolution-like operations on the edges connecting neighboring pairs of points, in the spirit of graph neural networks. It is shown in the following that such an operation, dubbed *edge convolution* (EdgeConv), has properties lying between translation-invariance and non-locality.

Unlike graph CNNs, the graph is not fixed but rather is dynamically updated after each layer of the network. That is, the set of k-nearest neighbors of a point changes from layer to layer of the network and is computed from the sequence of embeddings. Proximity in feature space differs from proximity in the input, leading to non-local diffusion of information throughout the point cloud.

3.6.1 Edge Convolution

Consider an F-dimensional point cloud with n points, denoted by $X = x_1, ..., x_n \subseteq \mathbb{R}^F$. In the simplest setting of $F = 3$, each point contains 3D coordinates $x_i = (x_i, y_i, z_i)$; it is also possible to include additional coordinates representing color, surface normal, and so on. In a deep neural network architecture, each subsequent layer operates on the output of the previous layer, so more generally the dimension F represents the feature dimensionality of a given layer.

We compute a directed graph $G = (V, \varepsilon)$ representing local point cloud structure, where $V = 1, ..., n$ and $\varepsilon \subseteq V \times V$ are the vertices and edges, respectively. In the simplest case, we construct G as the

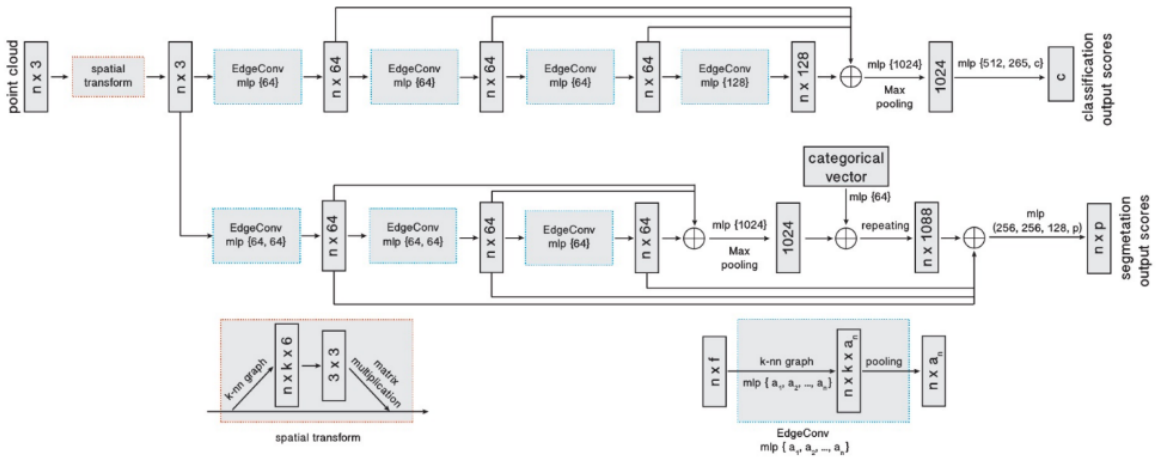


Figure 5: **Model Architectures**: The model architectures used for classification (top branch) and segmentation (bottom branch). The classification model takes as input n points, calculates an edge feature set of size k for each point at an EdgeConv layer, and aggregates features within each set to compute EdgeConv responses for corresponding points. The output features of the last EdgeConv layer are aggregated globally to form an $1D$ global descriptor, which is used to generate classification scores for c classes. The segmentation model extends the classification model by concatenating the $1D$ global descriptor and all the EdgeConv outputs (serving as local descriptors) for each point. It outputs per-point classification scores for p semantic labels. $\oplus$: concatenation. **Point cloud transform block:** The point cloud transform block is designed to align an input point set to a canonical space by applying an estimated 3×3 matrix. To estimate the 3×3 matrix, a tensor concatenating the coordinates of each point and the coordinate differences between its k neighboring points is used. **EdgeConv block:** The EdgeConv block takes as input a tensor of shape $n \times f$, computes edge features for each point by applying a multi-layer perceptron (mlp) with the number of layer neurons defined as $\{a_1, a_2, ..., a_n\}$, and generates a tensor of shape $n \times a_n$ after pooling among neighboring edge features.

k-nearest neighbor (k-NN) graph of X in $\mathbb{R}^F$. The graph includes self-loop, meaning each node also points to itself. We define *edge features* as $e_{ij} = h_\Theta(x_i, x_j)$, where $h_\Theta : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^{F'}$ is a nonlinear function with a set of learnable parameters Θ.

Finally, they define the EdgeConv operation by applying a channel-wise symmetric aggregation operation $\square$ (e.g.$\sum$, or max) on the edge features associated with all the edges emanating from each vertex. The output of EdgeConv at the i-th vertex is thus given by

$$x'_i = \underset{j:(i,j\epsilon)\varepsilon}{\square} h_\Theta(x_i, x_j) \tag{3}$$

Making analogy to convolution along images, they regard x_i as the central pixel and $x_j : (i, j)\epsilon\varepsilon$ as a patch around it (see Figure 6). Overall, given an F-dimensional point cloud with n points, EdgeConv produces an F'-dimensional point cloud with the same number of points.

Choice of h and $\square$. The choice of the edge function and the aggregation operation has a crucial influence on the properties of EdgeConv. For example, when $x_1, ..., x_n$ represent image pixels on a regular grid and the graph G has connectivity representing patches of fixed size around each pixel, the choice $\theta_m \cdot x_j$ as the edge function and sum as the aggregation operation yields standard convolution:

$$x'_{im} = \sum_{j:(i,j\epsilon\varepsilon)} \theta_m \cdot x_j \tag{4}$$

Here, $\Theta = (\theta_1, ..., \theta_M)$ encodes the weights of M different filters. Each θ_m has the same dimensionality as x, and $\cdot$ denotes the Euclidean inner product.

A second choice of h is

$$h_\Theta(x_i, x_j) = h_\Theta(x_i) \tag{5}$$

encoding only global shape information oblivious of the local neighborhood structure. This type of operation is used in PointNet, which can thus be regarded as a special case of EdgeConv. A third choice of h is

$$h_\Theta(x_i, x_j) = h_\Theta(x_j) \tag{6}$$

and

$$x'_{im} = \sum_{j \epsilon V} (h_{\theta(x_j)}) g(u(x_i, x_j)) \tag{7}$$

where g is a Gaussian kernel and u computes pairwise distance in Euclidean space.

A fourth option is

$$h_\Theta(x_i, x_j) = h_\Theta(x_j - x_i) \tag{8}$$

This encodes only local information, treating the shape as a collection of small patches and losing global structure.

Finally, a fifth option that they adopt in this paper is an asymmetric edge function

$$h_\Theta(x_i, x_j) = \bar{h}_\Theta(x_i, x_j - x_i) \tag{9}$$

This explicitly combines global shape structure, captured by the coordinates of the patch centers xi , with local neighborhood infor- mation, captured by $x_j - x_i$. In particular, they define the operator by notating

$$e'_{ijm} = ReLU(\theta_m \cdot (x_j - x_i) + \phi_m \cdot x_i) \tag{10}$$

which can be implemented as a shared MLP, and taking

$$x'_{im} = \max_{j:(i,j) \epsilon \varepsilon} e'_{ijm} \tag{11}$$

where $\Theta = (\theta_1, ..., \theta_M, \phi_1, ..., \phi_M)$

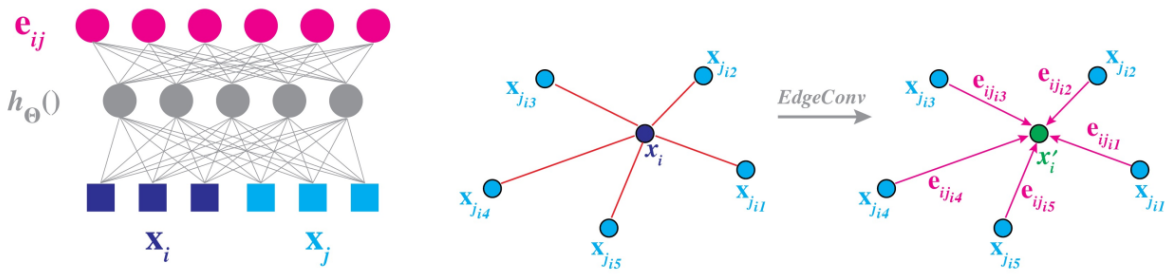


Figure 6: **Left:** Computing an edge feature, $e_{ij}(top)$, from a point pair, x_i and $x_j(bottom)$. In this example, $h_\Theta()$ is instantiated using a fully connected layer, and the learnable parameters are its associated weights. **Right:** The EdgeConv operation. The output of EdgeConv is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

3.6.2 Dynamic graph update

Experiments suggest that it is beneficial to recompute the graph using nearest neighbors in the feature space produced by each layer. This is a crucial distinction of the method from graph CNNs working on a fixed input graph. Such a dynamic graph update is the reason for the name of the architecture, the *Dynamic Graph CNN* (DGCNN). With dynamic graph updates, the receptive field is as large as the diameter of the point cloud, while being sparse.

At each layer we have a different graph , $G^{(l)} = (V^{(l)}, \varepsilon^{(l)})$ where the l-th layer edges are of the form $(i, j_{i1}), ..., (i, j_{ik_l})$ such that $x^{(l)}_{j_{i1}}, ..., x^{(l)}_{j_{ik_l}}$ are the k_l points closest to $x^{(l)}_i$. Put differently, the architecture learns how to construct the graph G used in each layer rather than taking it as a fixed constant constructed before the network is evaluated. In the implementation, they compute a pairwise distance matrix in feature space and then take the closest k points for each single point.

3.6.3 Properties

Permutation Invariance. Consider the output of a layer

$$x'_i = \max_{j:(i,j)\epsilon\varepsilon} h_\Theta(x_i, x_j) \tag{12}$$

and a permutation operator π. The output of the layer x'_i is invariant to permutation of the input x_j because max is a symmetric function (other symmetric functions also apply). The global max pooling operator to aggregate point features is also permutation-invariant.

Translation Invariance. The operator has a "partial" translation invariance property, in that the choice of edge functions (9) explicitly exposes the part of the function that can be translation-dependent and optionally can be disabled. Consider a translation applied to x_j and x_i ; they show that part of the edge feature is preserved when shifting by T. In particular, for the translated point cloud we have

$$\begin{aligned} e'_{ijm} &= \theta_m \cdot (x_j + T) - (x_i + T)) + \phi_m \cdot (x_i + T) \\ &= \theta_m \cdot (x_j - x_i) + \phi_m \cdot (x_i + T) \end{aligned} \tag{13}$$

If we only consider $x_j - x_i$ by taking $\phi_m = 0$, then the operator is fully invariant to translation. In this case, however, the model reduces to recognizing an object based on an unordered set of patches, ignoring the positions and orientations of patches. With both $x_j - x_i$ and x_i as input, the model takes account into the local geometry of patches while keeping global shape information.

3.6.4 Comparison of classification results on ModelNet40

Model	Overall Classification Accuaracy
PointNet (BASELINE)	87.1
PointNet	89.2
DGCNN (BASELINE)	91.7
DGCNN	92.9

Table 1: Overall classification accuracy on ModelNet40 test set

3.7 AGCN

3.7.1 Architecture

As shown in Figure 7, input point clouds are first fed into the segmentation network, which outputs a predicted label map. The one-hot encoded label map is concatenated with the input point clouds as an input for the discriminator. Being trained with BCE loss, the discriminator network discriminates whether the label map is real or fake. The segmentation network is trained to minimise a hybrid loss, its own point-wise cross entropy loss and an additional adversarial loss that is defined as L_2 difference

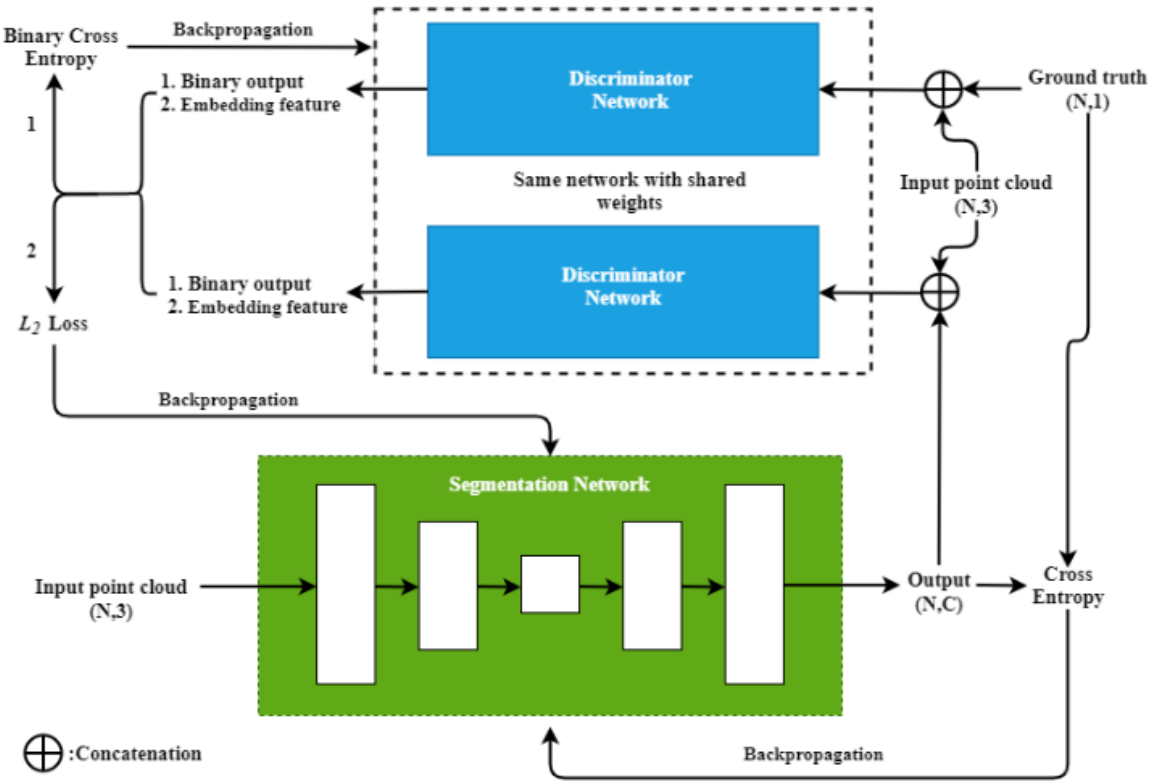


Figure 7: Overall architecture of proposed network

between two respective embedding feature vectors of the ground truth and the predicted label maps in the final convolution layer of the discriminator. In this way, the two networks are adversarial, and the segmentation network tries to deceive the discriminator network by predicting outputs that have the similar distribution as the ground truth.

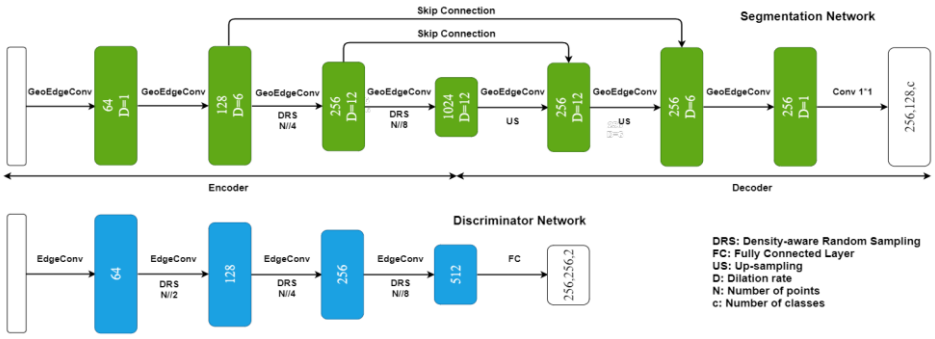


Figure 8: Proposed segmentation and discriminator networks

As shown in Figure 8, the segmentation network adopts the shape of U-Net. In the encoder, point clouds are down-sampled using Density-aware Random Sampling (DRS). Random sampling (RS) can sub-sample points quickly, but it may lose important geometric features over sparse region. In contrast, DRS estimates the density of each point by finding distances of neighbouring points and uses this distance to weigh each point during random sampling, which has a computational cost of $O(N)$. The

weight of a point can be computed as follows:

$$W_i = (\sum_{j=1}^{k} \|x_i - x_j\|_2)^n \tag{14}$$

where k is the number of neighbouring points in KNN, x_i and x_j are centre and neighbouring point respectively, and n decides how much to focus on the sparse regions to preserve important geometric information over sparse data. Furthermore, since we compute KNN in each convolution, there is no need to find the nearest neighbours again, reducing the com- putational cost significantly. The decoder consists of three components: an up-sampling by using Inverse Distance Weighted KNN Interpolation, a skip connection with the correspond- ing feature map from the encoder and GeoEdgeConv.

The discriminator network resembles the encoder part of the segmentation network. However, EdgeConv is used instead of GeoEdgeConv as point coordinates of the ground truth and the prediction are the same; using GeoEdgeConv could degrade the discriminating ability by diluting informative local point features with unnecessary point coordinates.

3.7.2 Adversarial Loss

Adversarial training helps segmentation network to improve its performance by trying to reduce the difference between two outputs of the ground truth and predicted label maps in the discriminator. Typical adversarial learning-based segmentation networks are trained with two losses, a point-wise loss L_{point} and an adversarial loss L_{adv}:

$$L_S = L_{point}(S(x;\theta_{seg}), y) + \lambda L_{adv}(S(x;\theta_{seg}); x, \theta_{disc}) \tag{15}$$

where x and y are an input and the corresponding label respectively, θ_{seg} and θ_{disc} are the parameters for the segmentation and the discriminator networks respectively, $S(x;\theta_{seg})$ indicates the output from the segmentation, and λ represents the relative importance of the adversarial loss term. The discriminator network is trained so that the ground truth and predicted label inputs can be classified as true and fake respectively.

Existing works proposed adversarial learning for point cloud segmentation, but they both used BCE loss between the two outputs of the discriminator as an adversarial loss, resulting in unstable training as well as insufficient segmentation accuracy with gradient feedback by a single binary prediction based on global average feature of in- put labels. To overcome this drawback, in this paper, they propose to use embedding features in the last convolution layer of the discriminator, which contain richer structural features than the final output of the input label map. Accordingly, our adversarial loss for the segmentation network is expressed as follows:

$$L_{adv}(\hat{y}, y; x, \theta_{disc}) = \frac{1}{B \times N \times P} \sum_{i=1}^{B} \sum_{j=1}^{N} \sum_{k=1}^{P} (D_{emb}^{ijk}(y; x, \theta_{disc}) - D_{emb}^{ijk}(\hat{y}; x, \theta_{disc}))^2 \tag{16}$$

where the first and second terms denote BCE for the predicted label map from the segmen- tation network and the ground truth label map respectively.

As the segmentation network is trained so that the high level features of its predicted labels resemble those of the ground truth labels, it will deceive the discriminator. By training the two networks adversarially, the discriminator will also discriminate the plausible prediction from the segmentation network from the ground truth. This model improves the training stability of the networks and the accuracy of label prediction by learning high level features of the ground truth labels that are smooth and consistent.

The optimal value for λ can be estimated automatically by calculating homoscedastic uncertainty during back-propagation. Hence, the final loss for our segmentation network becomes:

$$L_S = \frac{1}{\sigma_1^2} L_{point} + \frac{1}{2\sigma_2^2} L_{adv} + R(\sigma) \tag{17}$$

where each σ indicates the uncertainty of each task and $R(\sigma) = \log \sigma_1 \sigma_2$ is a regularisation term to prevent the uncertainty parameters from becoming too large.

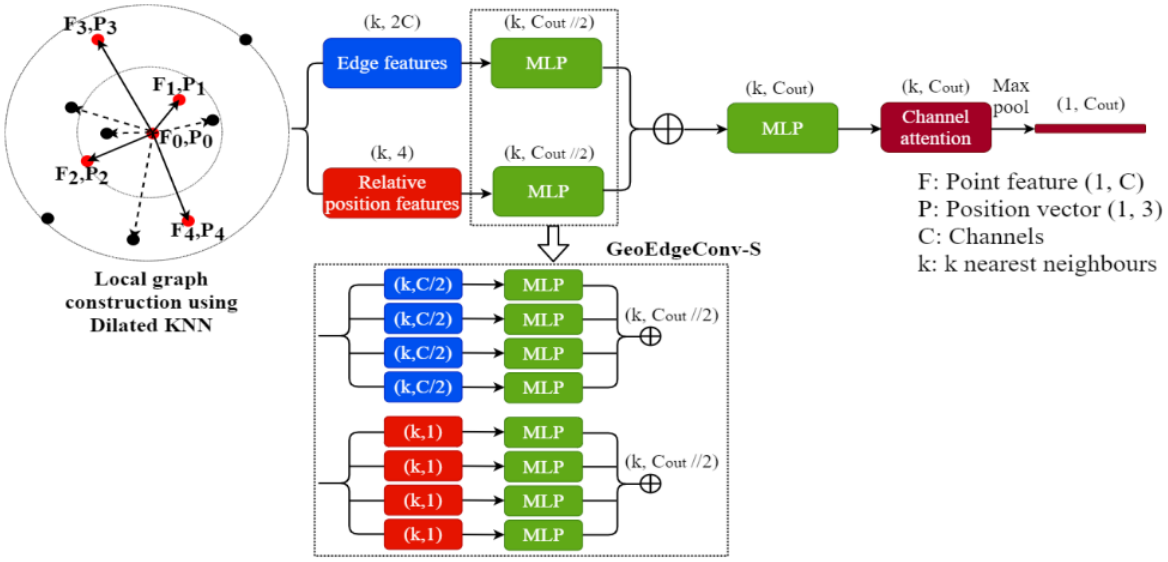


Figure 9: Proposed local feature extraction methods: GeoEdgeConv and GeoEdgeConv-S

3.7.3 GeoEdgeConv

To perform convolution onto raw point clouds, it is essential to satisfy permutation invariance and to extract local features. Previous works introduced various approaches to deal with these issues, but they still carry limitations in capturing rich local features.

Consider raw point clouds X with n points as $\{x_1...x_k...x_n\}$ where x_k has x, y, z coordinates. DGCNN proposed EdgeConv as a method of local feature aggregation, which can be expressed as follows:

$$x_{new} = \underset{j:(i,j)\subseteq E}{max} h_\Theta(f_i, f_j - f_i) \tag{18}$$

where E denotes a local neighbourhood point set for convolution, h_Θ is a nonlinear function with learnable parameters (e.g. ReLU with MLP) and $f_j - f_i$ is a relative point feature between a centre point i and a neighbouring point j.

However, EdgeConv is likely to lose local geometric information if points are down- sampled in intermediate layers, specially when a large receptive field is set. Eventually the drawback can result in poor segmentation accuracy such as misclassification in boundaries. This motivated the authors to propose GeoEdgeConv, which uses relative point coordinates explicitly in addition to edge features as point features. The GeoEdgeConv can be expressed as follows:

$$x_{new} = \underset{j:(i,j)\subseteq E}{max} CA(h_{\theta 3}(h_{\theta 2}(f_i, f_j - f_i), h_{\theta 1}(x_j - x_i, \|x_j - x_i\|_2))) \tag{19}$$

where CA denotes channel-wise attention, $x_j - x_i$ and $\|x_j - x_i\|$ are relative positions and the Euclidean distance respectively, and $h_{\theta 2}$, $h_{\theta 2}$ and $h_{\theta 3}$ are mish with shared MLP. By incorporating the two relative position features, the local geometry can be preserved over the convolution layers regardless of sub-sampling and the size of receptive field. This advantage improves segmentation accuracy significantly by enabling the network to learn fine-detailed geometry of complex structures.

GeoEdgeConv first constructs local graphs using dilated KNN as shown in Fig 9, enlarging the receptive field with no added computational cost. Then, each local point and relative position features are fed into shared MLPs, $h_{\theta 1}$ and $h_{\theta 2}$, to lift the features into the same number of channels to carry same importance. The outputs are then concatenated and fed into another MLP to fuse information. A channel-wise attention by squeeze-and-excitation is added to perform feature recalibration to suppresses unnecessary features and place more weights on important features, and thus improve the segmentation accuracy.

To design a lighter model, they propose GeoEdgeConv-S, where it uses group convolution for the shared MLPs, $h_{\theta 1}$ and $h_{\theta 2}$, with group size= 4 to reduce the model size and complexity as shown in

figure 9. The size of a kernel is also halved to $k = 8$ from 16 to reduce the computational cost, while the dilation rate is increased to maintain the same receptive field. The full procedure of GeoEdgeConv is summarised in Algorithm 1.

Algorithm 1 GeoEdgeConv

Input: x_{pos}, x, k, r ▷ position vectors, point features, size of neighbours, dilation rate
Output: F_{out}
F_c, F_n, P_c, P_n ▷ Centre and neighbouring point features and point coordinates
1: $idx \leftarrow DilatedKNN(x_{pos}, k, r)$
2: $F, P \leftarrow x[idx], x_{pos}[idx]$ ▷ Construct local neighbourhood graphs
3: $F_{edge}, P_{geo} \leftarrow [F_n - F_c, F_c], [P_n - P_c, \|P_n - P_c\|_2]$ ▷ Find edge and relative position features
4: $F_{edge}, F_{geo} \leftarrow MLP(F_{edge}), MLP(P_{geo})$ ▷ Lift each feature into same dimension
5: $F_{out} \leftarrow CA(MLP([F_{edge}, F_{geo}]))$ ▷ Fuse the features and feed to the channel attention
6: **return** F_{out}

3.8 Comparison of existing methods

Model	Adversarial	Feature	Learnable parameter	Graph	Sampling
PointNet	x	f_i	MLP, ReLU	x	x
DGCNN	BCE	$f_i, f_j - f_i$	MLP, LReLU	KNN	x
AGCN	Embedding L_2	f_i, $f_j f_i$, $x_j - x_i$, $\|x_j - x_i\|_2$	Group MLP, Mish, CA	DKNN	DRS

Table 2: LReLU denotes Leaky ReLU, CA denotes channel-wise attention and DKNN is dilated KNN

3.9 attMPTI

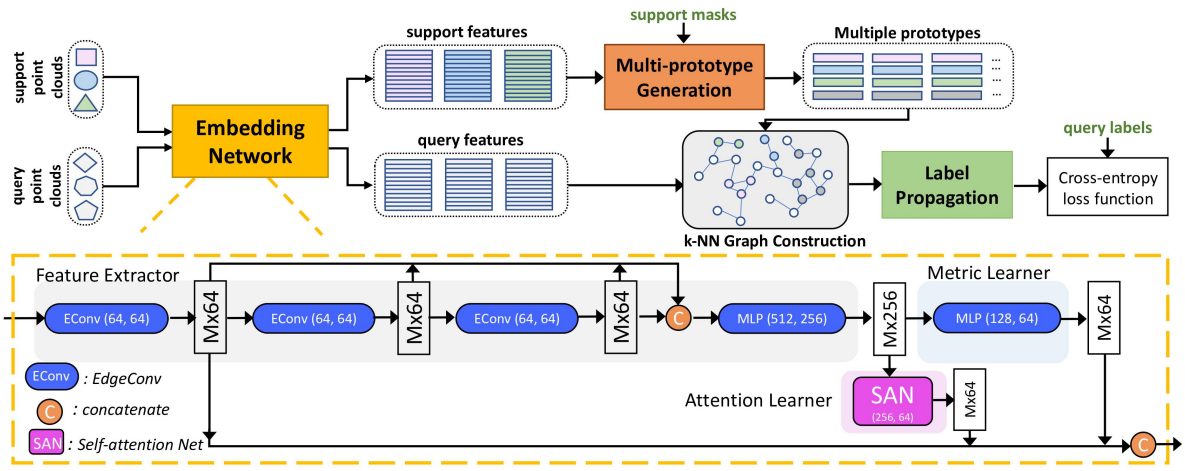


Figure 10: The architecture of our proposed method. This figure illustrates a 3-way 1-shot setting

3.9.1 Problem Definition

They align the training and testing of few-shot point cloud semantic segmentation with the episodic paradigm that is commonly used in few-shot learning. Specifically, they train the model on a group of few-shot tasks sampled from a data set with respect to a training class set C_{train}, and then they test the trained model by evaluating it on another group of tasks sampled from a different data set with respect to new classes C_{test}, where $C_{test} \cap C_{train} = \varnothing$. Each few-shot task, a.k.a. an episode,

is instantiated as an N-way K-shot point cloud semantic segmentation task. In each N-way K-shot episode, we are given a *support* set, denoted as $S = \{(\mathbf{P}_s^{1,k}, \mathbf{M}^{1,k})_{k=1}^{K}, ..., (\mathbf{P}_s^{N,k}, \mathbf{M}^{N,k})_{k=1}^{K}\}$, with K labeled pairs of support point cloud $\mathbf{P}_s^{n,k}$ and its corresponding binary mask $\mathbf{M}^{n,k}$ for each of the N unique classes. Each point cloud $\mathbf{P} \epsilon \mathbb{R}^{M \times (3+f_\circ)}$ contains M points associated with the coordinate information $\epsilon \mathbb{R}^3$ and an additional feature $\epsilon \mathbb{R}^{f_\circ}$, *e.g.* color. We are also given a *query* set, denoted as $Q = (\mathbf{P}_q^i, \mathbf{L}^i)_{i=1}^{T}$, which contains T pairs of query point cloud $\mathbf{P}_q^i$ and its corresponding label $\mathbf{L}^i \epsilon \mathbb{R}^{M \times 1}$. Note that the ground-truth label L is only available during training. The goal of N-way K-shot point cloud semantic segmentation is to learn a model $f_\phi(\mathbf{P}_q, S)$ that predicts the label distribution $\mathbf{H} \epsilon \mathbb{R}^{M \times (N+1)}$ for any *query* point cloud $\mathbf{p}_q$ based on S. Formally, our training objective is to find the optimal parameters Φ^* of $f_\phi(\mathbf{P}_q, S)$ by computing:

$$\Phi^* = \underset{\Phi}{argmin}\, \mathbb{E}_{S,Q \sim T_{train}} \left[\sum_{(\mathbf{P}_q^i, \mathbf{L}^i) \epsilon Q} J(\mathbf{L}^i, f_\Phi(\mathbf{P}_q^i, S)) \right] \tag{20}$$

where T_{train} denotes the training set containing all the episodes sampled from C_{train} , and $J(\cdot)$ is the loss function.

3.9.2 Attention-aware Multi-prototype Transductive Inference Method

Figure 10 illustrates the attention-aware multi-prototype transductive inference framework. It consists of five components:
1) **Embedding network** that learns the discriminative features for the support and query point clouds
2) **Multi-prototype generation** that produces multiple prototypes for each of the *N + 1* classes (*N* semantic classes and one background class)
3) ***k*-NN graph construction** that encodes both the cross-set (support-query) and intra-set (support-support, query-query) relationships within the embedding space
4) **Label propagation** that diffuses labels through the whole graph along high density areas formed by the unlabeled query points.
5) **Cross-entropy loss** function that computes the loss between the predicted labels and ground-truth labels of all the query points

3.9.3 Embedding Network

The embedding network is the most important part of the model since both multi-prototype generation and k-NN graph construction are dependent on the learned embedding space. We expect this space to possess three properties: it can 1) encode the geometric structures of points based on local context; 2) encode the semantic information of points and their semantic correlation based on global context; and 3) quickly adapt to different few-shot tasks. To this end, they design an attention-aware multi-level feature learning network that incorporates three levels of features: local geometric features, global semantic features, and metric-adaptive features. Specifically, the embedding network is composed of three modules: *feature extractor, attention learner, and metric learner*. They adopt DGCNN, a dynamic graph CNN architecture, as the backbone of our feature extractor to respectively produce local geometric features (outputs of the first EdgeConv layer) and semantic features (outputs of the feature extractor). To further explore semantic correlation between points in the global context, they apply a self-attention network (SAN) on the generated semantic features. SAN allows the point-wise feature to aggregate the global contextual information of the corresponding point cloud in a flexible and adaptive manner. The architecture of SAN is illustrated in Figure 11. In addition, they introduce the metric learner, i.e. a stack of multi-layer perceptrons (MLP) layers to enable faster adaptability of the embedding space to different few-shot tasks since the feature extractor is updated with a slower learning rate. The metric learner maps all point-wise features of support and query sets into a manifold space, where common distance functions (*e.g.* euclidean distance or cosine distance) can be directly used to measure proximity between points. Finally, they concatenate the three levels of learned features together as the output of the embedding network.

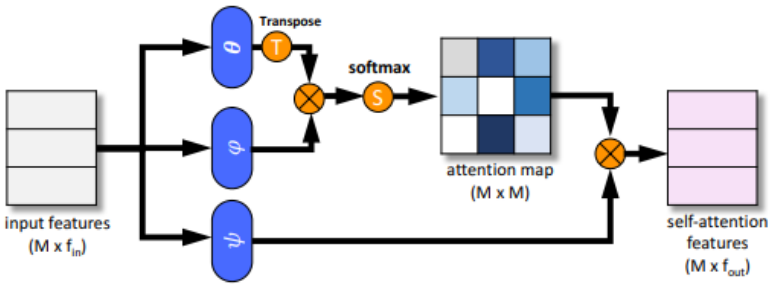


Figure 11: The architecture of Self Attention Network (SAN)
. θ, φ and ψ are linear embedding functions with trainable parameters.

3.9.4 Multi-prototype Generation

For each of the $N+1$ classes in the support set, we generate n prototypes to model the complex data distribution according to the few labeled samples in the episode. We cast the generation procedure as a clustering problem. While there can be different ways to cluster support points into multiple prototypes, we employ a simple strategy: sampling seed points and point-to-seed assignment based on the learned embedding space. Specifically, we sample a subset of n seed points from a set of support points in one class using the farthest point sampling based on the embedding space. Intuitively, the farthest points in this space can inherently represent different perspectives of one class if the embedding space is learned well. Let $\{s_i^c\}_{i=1}^{n}$ and $\{f_i^c\}_{i=1}^{m^c}$, where $\{s_i^c\}_{i=1}^{n} \subset \{f_i^c\}_{i=1}^{m^c}$ denote the sampled seeds and all the mc support points belonging to the class c, respectively. We compute the point-to-seed distance and take the index of the closest seed as the assignment of a point. Formally, the multi-prototypes μ^c of class c is given by:

$$\begin{aligned} \mu^c = \{\mu_1^c, ..., \mu_n^c \mid \mu_i^c = \frac{1}{|I_i^{c*}|} \sum_{f_j^c \, \epsilon \, I_i^{c*}} f_j^c \\ s.t. \;\; I^{c*} = \underset{I^c}{argmin} \sum_{i=1}^{n} \sum_{f_j^c \, \epsilon \, I_i^c} \left\| f_j^c - s_i^c \right\|_2 \end{aligned} \tag{21}$$

where $\{f_i^c\}_{i=1}^{m^c}$ is partition into n sets $I^{c*} = \{I_1^{c*}, ..., I_n^{c*}\}$ such that $f_j^c \, \epsilon \, I_i^{c*}$ is assigned to s_i^c.

3.9.5 Transductive Inference

In addition to the similarity relations between each unlabeled query point and the labeled multiprototypes, they also consider the similarity relations between pairs of unlabeled query points to exploit the "smoothness" constraints among neighboring query points in our few-shot point cloud semantic segmentation task. To this end, they leverage on transductive inference to reason cross-set and intra-set relationships based on the embedding space. Concretely, they propose the use of transductive label propagation to construct a graph on the labeled multi-prototypes and the unlabeled query points, and then propagate the labels in the graph with random walk.

3.9.6 k-NN graph construction

To mitigate the large number of query points, they construct a k Nearest Neighbor (NN) graph instead of a fully-connected graph for computational efficiency. Specifically, they take both the $n \times (N+1)$ multiprototypes and $T \times M$ query points as nodes of a graph with size $V = n \times (N+1) + T \times M$. We construct a sparse affinity matrix, denoted as $\mathbf{A} \, \epsilon \, \mathbb{R}^{V \times V}$, by computing the Gaussian similarity between each node and its k nearest neighbors in the embedding space:

$$\mathbf{A}_{ij} = exp\left(-\frac{\|\mathbf{v}_i - \mathbf{v}_j\|_2^2}{2\sigma^2} \right), \; for \; \mathbf{v}_j \, \epsilon \, N_k(\mathbf{v}_i) \tag{22}$$

where $\mathbf{v}_i$ represents the node feature and σ^2 is the variance of the distance between two nodes. Let $\mathbf{W} = \mathbf{A} + \mathbf{A}^\top$, this assures the adjacency matrix is non-negative and symmetric. Subsequently, they symmetrically normalize $\mathbf{W}$ to yield $\mathbf{S} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{D}$ is the diagonal degree matrix with its diagonal value to be the sum of the corresponding row of $\mathbf{W}$. Furthermore, they define a label matrix $\mathbf{Y} \epsilon \mathbb{R}^{V\times(N+1)}$, where the rows corresponding to labeled prototypes are one-hot ground truth labels and the rest are zero.

3.9.7 Label propagation

Given $\mathbf{S}$ and $\mathbf{Y}$, label propagation iteratively diffuses labels through the graph according to:

$$\mathbf{Z}_{t+1} = \alpha\mathbf{S}\mathbf{Z}_t + (1-\alpha)\mathbf{Y} \tag{23}$$

$\mathbf{Z}_t \epsilon \mathbb{R}^{V\times(N+1)}$ represents the predicted label distributions at iteration t, and $\alpha \epsilon (0,1)$ is a parameter that controls the relative probability of the information from its adjacency nodes or its initial labels. Sequence $\{\mathbf{Z}_t\}$ converges to a closed-form solution:

$$\mathbf{Z}^* = (\mathbf{I} - \alpha\mathbf{S})^{-1}\mathbf{Y} \tag{24}$$

They adopt the closed-form solution to directly compute the predictions $\mathbf{Z}^*$ of label propagation.

3.9.8 Loss Function

Once $\mathbf{Z}^*$ is obtained, they first take the predictions corresponding to the T query point clouds denoted as $\left\{\mathbf{z}^i\right\}_{i=1}^{T}$, where $\mathbf{z}^i \epsilon \mathbb{R}^{M\times(N+1)}$ represents the predictions of the point cloud $\mathbf{P}_i^q$. The prediction of each point in $\mathbf{z}^i$ is then normalized into a probability distribution using the softmax function:

$$\mathbf{H}_{m,n}^i = \frac{exp(\mathbf{z}_{m,n}^i)}{\sum_{j=1}^{N+1} exp(\mathbf{z}_{m,j}^i)} \tag{25}$$

Finally, they compute the $cross-entropy\,loss$ between $\left\{\mathbf{H}^i\right\}_{i=1}^{T}$ and the ground truth labels $\left\{\mathbf{L}^i\right\}_{i=1}^{T}$ as:

$$J_\Phi = -\frac{1}{T}\frac{1}{M}\sum_{i=1}^{T}\sum_{m=1}^{M}\sum_{n=1}^{N+1}\left[\mathbf{L}_m^i = n\right] log(\mathbf{H}_{m,n}^i) \tag{26}$$

where Φ is the set of parameters of our model $f_\Phi(\mathbf{P}_q, S)$. More precisely, $f_\Phi(\mathbf{P}_q, S) = h(g_\Phi(\mathbf{P}_q, \mathbf{P}_s), \mathbf{M})$ is a composite function of the embedding network $g_\Phi(.)$, and the multi-prototypes generation and transductive inference operations $h(.)$. It becomes apparent that the minimization of J over the parameters Φ is governed by the affinity properties among the labeled multi-prototypes and unlabeled query points since the gradients have to flow through the parameter-less $h(.)$ into the embedding network $g_\Phi(.)$.

4 Our Methodology

Here, we are trying to implement AGCN model in place on DGCNN model in Attention-aware Multi-prototype Transductive Inference Method and perform few-shot task for 3D semantic segmentation on two benchmark datasets **S3DIS** and **Scannet**.

We will compare the results of few-shot tasks on three state-of-the-art models, namely - PointNet, DGCNN and AGCN in different N-way K-shot settings. Our methodology is same as attMPTI[21] method.

5 Dataset and Setup

5.1 S3DIS

Path to S3DIS data set: `https://cvg-data.inf.ethz.ch/s3dis/`
The Stanford 3D Indoor Scene Dataset (S3DIS) dataset contains 6 large-scale indoor areas with 271 rooms. Each point in the scene point cloud is annotated with one of the 13 semantic categories.
Name of dataset file: Stanford3dDataset v1.2.zip
Size: 4.8 GB
Procedure to obtain the dataset:
1) Move the .zip file to the working directory (attMPTI). Now unzip the dataset.
2) Rename the folder as S3DIS.

5.2 Scannet

Path to Scannet data set: `http://www.scan-net.org/`
Scanet is an RGB-D video dataset containing 2.5 million views in more than 1500 scans, annotated with 3D camera poses, surface reconstructions, and instance-level semantic segmentations.
Name of dataset file: ScanNet v2
Size: 1.3 TB
Procedure to obtain the dataset:
1) Use the following script to download the ScanNet data: `http://kaldir.vc.in.tum.de/scannet/` download-scannet.py.
2) Move this script file in the working directory (attMPTI).
3) **Some useful info:**
Scan data is named by scene[spaceid] [scanid], or scene%04d %02d, where each space corresponds to a unique location (0-indexed).
Script usage
- To download the entire ScanNet release (1.3TB): download-scannet.py -o [directory in which to download]
- To download a specific scan (e.g., scene0000 00): download-scannet.py -o [directory in which to download] –id scene0000 00
- To download a specific file type (e.g., *.sens, valid file suffixes listed here): download-scannet.py -o [directory in which to download] –type .sens
- Train/test splits are given in the main ScanNet project repository: `https://github.com/ScanNet/ScanNet/tree/master/Tasks/Benchmark`
4) Here we are not downloading *.sens files, as these files take lot of memory space and are not used in our work.
5) Put all the files in a folder and rename it as Scannet.

6 Implementation

6.1 Making environment

1) Make virtual environment
```
python -m venv env
```
2) Activate virtual environment
```
source env/bin/activate
```

6.2 Installation

Installation of required libraries in the virtual environment
- `pip install torch`
- `pip install faiss-gpu`
- `pip install tensorboard h5py transforms3d`
-Install 'torch-cluster' with the corrreponding torch and cuda version

pip install pyg_lib torch_scatter torch_sparse torch_cluster torxh_spline conv -f http://data.pyg.org/whl.torch-2.0.0+cu117.html

6.3 Preprocessing

6.3.1 S3DIS

1) Re-organize raw data into .npy files by running
python preprocess/collect_s3dis_data.py –data_path $path_to_S3DIS_raw_data
The generated numpy files are stored in ./datasets/S3DIS/scenes/data by default.
2) To split rooms into blocks, run
python preprocess/room2blocks.py –data_path ./datasets/S3DIS/scenes/
One folder named blocks_bs1_s1 will be generated under ./datasets/S3DIS/ by default.

6.3.2 Scannet

1) Re-organize raw data into .npy files by running
python preprocess/collect_scannet_data.py –data_path $path_to_Scannet_raw_data
The generated numpy files are stored in ./datasets/Scannet/scenes/data by default.
2) To split rooms into blocks, run
python preprocess/room2blocks.py –data_path ./datasets/Scannet/scenes/
One folder named blocks_bs1_s1 will be generated under ./datasets/Scannet/ by default.

6.4 Pretraining

Pretrain the segmentor which includes feature extractor module on the available training set:
bash scripts/pretrain_segmentor.sh
Arguments for S3DIS dataset in scripts/pretrain_segmentor.sh file
DATASET='s3dis'
DATA_PATH='./datasets/S3DIS/blocks_bs1_s1'
SAVE_PATH='./log_s3dis/
Arguments for Scannet dataset in scripts/pretrain_segmentor.sh file
DATASET='scannet'
DATA_PATH='./datasets/Scannet/blocks_bs1_s1'
SAVE_PATH='./log_scannet/
Arguments for split = 0 in scripts/pretrain_segmentor.sh file
SPLIT=0
Arguments for split = 1 in scripts/pretrain_segmentor.sh file
SPLIT=1

6.5 Training

Train model:
bash scripts/train_attMPTI.sh
Arguments for S3DIS dataset in scripts/train_attMPTI.sh file
DATASET='s3dis'
DATA_PATH='./datasets/S3DIS/blocks_bs1_s1'
SAVE_PATH='./log_s3dis/'
Arguments for Scannet dataset in scripts/train_attMPTI.sh file
DATASET='scannet'
DATA_PATH='./datasets/Scannet/blocks_bs1_s1'
SAVE_PATH='./log_scannet/'

Arguments for split = 0 in scripts/train_attMPTI.sh file
SPLIT=0
PRETRAIN_CHECKPOINT='./log_s3dis/log_pretrain_s3dis_S0'
Arguments for split = 1 in scripts/train_attMPTI.sh file
SPLIT=1
PRETRAIN_CHECKPOINT='./log_s3dis/log_pretrain_s3dis_S1'
Arguments for N-way K-shot in scripts/train_attMPTI.sh file
N_WAY=2 | 3
K_SHOT=1 | 5

6.6 Evaluation

After training evaluate the model:
bash scripts/eval_attMPTI.sh
Arguments for Split N-way K-shot in scripts/eval_attMPTI.sh file
MODEL_CHECKPOINT='./log_s3dis/log_mpti_s3dis_S0_N2_K1_Att1'
# Here SPLIT = 0, N-WAY = 2, K-SHOT = 1

6.7 Results (Reproduced)

All the results of attMPTI for S3DIS and Scannet dataset are reproduced.

2 - way				3 - way			
1 - shot		5 - shot		1 - shot		5 - shot	
S^0	S^1	S^0	S^1	S^0	S^1	S^0	S^1
53.77	55.94	61.67	67.02	45.18	49.27	54.92	56.79
52.23	**56.71**	**66.78**	**65.98**	**46.47**	**51.22**	**53.79**	**55.12**

Table 3: Reproduced results on S3DIS dataset using mean-IoU metric (%). S^i denotes the split i is used for testing. Red-colored numbers indicates reproduced results.

2 - way				3 - way			
1 - shot		5 - shot		1 - shot		5 - shot	
S^0	S^1	S^0	S^1	S^0	S^1	S^0	S^1
42.55	40.83	54.00	50.32	35.23	30.72	46.74	40.80
43.10	**39.50**	**53.04**	**47.55**	**34.28**	**29.91**	**45.68**	**38.21**

Table 4: Reproduced results on Scannet dataset using mean-IoU metric (%). S^i denotes the split i is used for testing. Red-colored numbers indicates reproduced results.

7 Conclusion

We are investigating important few-shot point cloud semantic segmentation problem. Furthermore, offering several key insights on few-shot 3D point cloud semantic segmentation. Through experiments and ablation studies on popular **S3DIS** dataset and **Scannet** dataset we are trying to propose an effective method that can outperform current state-of-the-art on few-shot 3D semantic segmentation methods.

8 Supplementary Material

8.1 Dataset Split

Table 5 lists the class names in each split of the **S3DIS** and **ScanNet** datasets.

	split = 0	split = 1
S3DIS	beam, board, bookcase, ceiling, chair, column	door, floor, sofa, table, wall, window
ScanNet	bathtub, bed, bookshelf, cabinet, chair, counter, curtain, desk, door, floor	otherfurniture, picture, refrigerator, show curtain, sink, sofa, table, toilet, wall, window

Table 5: Test class names for each split of S3DIS and Scan-Net.

References

[1] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[2] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[3] Mohsen Ghafoorian, Cedric Nugteren, Nóra Baka, Olaf Booij, and Michael Hofmann. El-gan: Embedding loss driven generative adversarial networks for lane detection, 2018.

[4] Farhad Ghazvinian Zanjani, David Anssari Moin, Bas Verheij, Frank Claessen, Teo Cherici, Tao Tan, and Peter H. N. de With. Deep learning approach to semantic segmentation in 3d point cloud intra-oral scans of teeth. In M. Jorge Cardoso, Aasa Feragen, Ben Glocker, Ender Konukoglu, Ipek Oguz, Gozde Unal, and Tom Vercauteren, editors, *Proceedings of The 2nd International Conference on Medical Imaging with Deep Learning*, volume 102 of *Proceedings of Machine Learning Research*, pages 557–571. PMLR, 08–10 Jul 2019.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[6] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds, 2020.

[7] Qiangui Huang, Weiyue Wang, and Ulrich Neumann. Recurrent slice networks for 3d segmentation of point clouds, 2018.

[8] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[9] Seung Gu Kim and Daniel C. Alexander. Agcn: Adversarial graph convolutional network for 3d point cloud segmentation. In *British Machine Vision Conference*, 2021.

[10] Loïc Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. *CoRR*, abs/1711.09869, 2017.

[11] Hongyan Li, Zhengxing Sun, Yunjie Wu, and Bo Li. Ppsan: Perceptual-aware 3d point cloud segmentation via adversarial learning. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4025–4029, 2019.

[12] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on $\mathcal{X}$-transformed points, 2018.

[13] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017.

[14] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[15] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning, 2017.

[16] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning, 2017.

[17] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019.

[18] Xiaoqing Ye, Jiamao Li, Hexiao Huang, Liang Du, and Xiaolin Zhang. 3d recurrent neural networks with context fusion for point cloud semantic segmentation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 415–430, Cham, 2018. Springer International Publishing.

[19] Chi Zhang, Guosheng Lin, Fayao Liu, Jiushuang Guo, Qingyao Wu, and Rui Yao. Pyramid graph networks with connection attentions for region-based one-shot semantic segmentation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9586–9594, 2019.

[20] Na Zhao, Tat-Seng Chua, and Gim Hee Lee. Psˆ2-net: A locally and globally aware network for point-based semantic segmentation. In *Proceedings of the 25th International Conference on Pattern Recognition (ICPR)*, pages 723–730, 2020.

[21] Na Zhao, Tat-Seng Chua, and Gim Hee Lee. Few-shot 3d point cloud semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[22] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.