IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON GEOMETRIC ALGEBRA IN SIGNAL PROCESSING

Received October 7, 2020, accepted October 13, 2020, date of publication October 16, 2020, date of current version October 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3031812

DCG-Net: Dynamic Capsule Graph Convolutional Network for Point Clouds

DENA BAZAZIAN[1] **AND DHANANJAY NAHATA**[2]
[1]Geomatics Department, Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), 08860 Barcelona, Spain
[2]Computer Science Department, Universitat Autònoma de Barcelona (UAB), 08193 Barcelona, Spain

Corresponding author: Dena Bazazian (dena.bazazian@cttc.es)

ABSTRACT This article introduces DCG-Net (Dynamic Capsule Graph Network) to analyze point clouds for the tasks of classification and segmentation. DCG-Net aggregates point cloud features to build and update the graphs based on the dynamic routing mechanism of capsule networks at each layer of a convolutional network. The first layer of DGC-Net exploits the geometrical attributes of the point cloud to build a graph by neighborhood aggregation while the deeper layers of the network dynamically update the graph based on the feature space of convolutions. We conduct extensive experiments on public datasets, ModelNet40, ShapeNet-Part. Our experimental results demonstrate that DCG-Net achieves state-of-the-art performance on public datasets, 93.4% accuracy on ModelNet40, and 85.4% instance mIoU (mean Intersection over Union) on ShapeNet-Part.

INDEX TERMS Convolutional networks, dynamic capsule graph, point clouds, segmentation and classification.

I. INTRODUCTION

Point clouds are a rich and fundamental representation of 3D raw data structures broadly employed as a storage format by 3D sensing devices. The importance of point clouds is due to their geometrical information and wide range of applications such as robotics, remote sensing, photogrammetry, autonomous driving and virtual/augmented reality [1], [2]. Principally two tasks of classification and segmentation are performed on point clouds in order to apply them for these aforementioned applications. Segmentation is a process of grouping point clouds into multiple homogeneous regions with identical properties whereas classification is a task that labels these regions. The regions can be considered as a part of an object or a whole object. The main aim of this work is to tackle the problems of classification and segmentation on unorganized point clouds data. Point clouds obtained by 3D scanner devices such as LiDAR scanners or line scanners are unorganized, noisy and sparse. These devices scan a scene line by line in order to render streams of depth information. The information cannot be represented in a single lattice when data is combined from several sensors types. Therefore, 3D point clouds data from multiple sensors is combined in an unorganized (non-ordered) format with color or intensity information of each point. This non-uniformity attribute of 3D point clouds causes challenges of using them in comparison with other types of data (namely 2D images). Despite the recent tremendous progress of deep learning approaches, there are still various challenges in applying CNNs (Convolutional Neural Networks) for point clouds because of unstructured and unorganized attributes of point clouds. Pioneering techniques for employing point clouds directly based on deep learning algorithms were introduced in [3], [4], whereas the previous techniques were based on pre-processing steps such as converting a point cloud to a 3D mesh or mapping it to a 2D space. These methods were time-consuming and with a possibility of missing some essential data [5], [6]. Graph-based approaches have been recently applied for point clouds processing by CNN techniques [7]–[11] due to their robustness for combining features on local and global surfaces. Dynamic graphs unlike graph CNNs are not fixed and dynamically updated after each layer of the network [7]. In DGCNN [7], a neighborhood aggregation graph is built based on the point's nearest neighbors to contribute to its convolution. Contrary to [7], in this work, we do not select the neighbors based on L^2 norm to calculate the euclidean distances between points. The selection of the neighbors of each point is based on performing a dynamic routing mechanism among all the points of a point cloud. The features of the points are represented in capsules. The most similar points of each point are explored based on the agreement that can be inferred from dynamic routing between capsules. The idea of

The associate editor coordinating the review of this manuscript and approving it for publication was Wenming Cao.

dynamic routing between capsules was initiated in [12], but to sthe best of our knowledge, we are the first to apply dynamic routing between capsules to select the neighbor points from raw point cloud data in order to build graphs by neighborhood aggregation. Each capsule is a set of units that represents a specific type of entity such as an object or a part of an object. Furthermore, we construct the capsules at the first layer of the network, based on the feature similarities of the points within the point cloud in a combination of euclidean, eigenvalues and geometrical spaces. The geometrical features that characterize each point, i.e. omnivariance, planarity, sphericity, sharp edges, etc., are determined by the computation of the eigenvalues of their covariance matrix of each point which in turn depends on the points selected to be in the neighborhood of each point. In contrast to [8] where neighbors are defined as nearest neighbors in euclidean and eigenvalues spaces, we consider the geometrical features additionally to determine the similarities between points, and we define a capsule graph at each layer in order to disentangle the geometrical features of point clouds to build the neighborhood aggregation graph. We also update the graph dynamically at each layer of the network.

The robustness of this work stems from the essence of the dynamic capsule graph itself. Unlike CapsGNN [13] and GCAPS-CNN [14] approaches where the capsules are fixed, our proposed capsule graph is dynamically updated at each layer of the network. Our capsule graph is initialized by the geometrical similarities of the points and the features of the capsule graph evolving from layer to layer of the network and arising from the sequence of voting and agreement between the latent capsules. Moreover, the inputs of the networks in [13], [14] are based on graph structure data. However, the input of our proposed network are a raw point cloud data; and we employed the capsules for solving the issues of building graphs from raw data, whereas [13], [14] applied the capsules to solve the problems regarding the graph classification and graph embeddings. In GCAPS-CNN [14] scalar features of graphs are extended to vector-valued capsules without any routing mechanism. CapsGNN [13] applied a routing mechanism between the network layers in order to maintain multiple information properties of graphs for generating graph embeddings. However, the key idea of this work is to apply a routing mechanism to find the similarities between all the points of a point cloud in order to build a graph by neighborhood aggregation from a raw data. We apply the routing mechanism at each layer to update the graph dynamically through the network. Therefore, contrary to CapsGNN [13], we define an individual routing mechanism module at each layer of the network, whereas in CapsGNN [13] the routing mechanism is applied between the layers of the network.

In this article, we tackle the drawbacks of DGCNN [7] and GS-Net [8] regarding: 1) neglecting the geometrical features; 2) selecting the nearest neighbors based on L^2 norm. To pave the way for these issues, we consider geometrical features as one of the inputs of the network in order to improve the training process of our proposed model by capturing features in both local and global spaces. Furthermore, we apply a dynamic routing mechanism instead of L^2 norm in order to select the nearest neighbors through a voting agreement process. The neighborhood aggregation graphs built by this process are more accurate because of the certain number of iterations that is considered for voting agreement in the dynamic routing mechanism between the capsules. Moreover, we tackle the drawbacks of Caps-GNN [13], GCAPS-CNN [14] and GS-Net [8] regarding neglecting to update the graph at each layer of the network. Our proposed model dynamically updates the graphs based on feature space through the network. It has the advantage of exploring the nearest neighbors based on the similarities in the convolutional feature space. In addition, unlike GS-Net [8] that feeds down-sampled features of point cloud at each layer of the network, our model after the first layer updates the graphs merely based on the feature space. Therefore, it leads to a significant improvement in the aspect of computational time at each forward pass of the network.

Our proposed model similar to other graph convolutional models [7], [8], [13], [14] is designed to be invariant to the ordering of neighbors, therefore it is invariant to permutation of each point of a point cloud. Furthermore, DCG-Net is differentiable and can be plugged into existing architectures for point cloud processing.

In a nutshell, the novelties of this work are to provide a dynamic capsule graph network that is initializes by geometrical features and builds neighborhood aggregation graphs based on the dynamic routing between capsules; it also updates the capsule graphs dynamically through the network. The main contributions of this article can be summarized as follows:

- We present a novel Dynamic Capsule Graph Network as DCG-Net by constructing a capsule graph based on geometrical features initialization and updating the capsule graph dynamically at each layer of the network.
- We formulate a novel technique to construct graphs by neighborhood aggregation through a dynamic routing mechanism between capsules based on the features similarities.
- We prove the robustness of our proposed DCG-Net by experimentally demonstrating the capabilities of the network with achieving the state-of-the-art in classification and part segmentation tasks on major public datasets.

The remainder of this article is organized as follows; Section II describes the background and literature review. In Section III, DCG-Net is introduced. The experimental results are shown in Section IV. Finally, we conclude this article in Section V.

II. RELATED WORK

Point cloud processing has been applied in a broad range of applications such as autonomous driving [15], robotics [16], CAD (Computer Aided Design) and sharp edge extraction [17], [18], photogrametry [19], and remote sensing [20].

Although CNNs have achieved remarkable success for various computer vision tasks, it still is an open challenge to integrate CNNs with 3D point clouds due to their unorganized and non-uniformity attributes. Deep learning on raw point clouds is currently attracting a lot of attention. Since the proposal of PointNet and PointNet++ [3], [4], many state-of-the-art methods have been developed [21]–[23]. The problems of point cloud analysis commonly concentrate on the tasks of classification and segmentation. Classification models are for finding the label of a given point cloud object. Segmentation models are for extracting parts of a point cloud into segments which are conceptually meaningful or simple for further analysis. A complete review of deep learning techniques for point clouds can be found in [6].

1) GRAPH-BASED METHODS

Graphs and manifolds are non-euclidean domains. Recently, various studies are devoted to making CNNs applicable for learning on graphs and manifolds. Geometric deep learning field focuses on generalizing the definition of convolution to functions on manifolds or graphs [24]. The challenging issue in geometric deep learning is because of the complexity of applying convolution when the space does not contain a group action, and when the input data is comprised of various shapes or graphs, where it is challenging to determine an alternative for convolutional filters [25]. To address this drawback, DGCNN [7] proposed a model by grouping points both in euclidean space and in semantic space. In this case the model learns the features on edge relationships of graph instead of the relative positions of points. KPConv [26] combines features locally according to the 3D geometry in order to be capable of capturing the deformations of the surfaces instead of combining features on local surface patches in graph convolution models while being invariant to the deformations of those patches in euclidean space. SPG [11] is built based on partitioning the points into geometrically homogeneous elements, which is then fed into a graph neural network. PointWeb [27] concentrates on the interaction between points in each local neighborhood region by exploring the context information between all point pairs. MHNet [28] constructed a non-directional probability graph from input points in order to mitigate the influence of noise points in the segmentation task of point clouds. GAC [29] is a graph attention convolution network that dynamically assigns adequate attentional weights to different neighboring points by merging their spatial positions and feature attributes. DPC [9] tackled the problem of small receptive field size of point cloud convolutional networks and proposed a dilated point convolution network based on a k-nearest neighbor graph. Following the same idea, PointAtrousGraph [10] proposed to enlarge receptive fields of filters by presenting a sampling rate in order to equivalently sparsely sample the neighboring point features and to preserve multiscale local geometrical details hierarchically. GS-Net [8] introduced a connection modules which exploit to group the points with similar and relevant geometrical data, and collect features from neighbors in both the euclidean and eigenvalues spaces. GS-Net [8] feeds the network just based on euclidean and eigenvalues features, however our model considers geometrical features (i.e. planarity, sphericity, sharp edges, change of curvature) in addition to euclidean and eigenvalues features. Accordingly, we build graphs based on the feature similarities in euclidean, eigenvalues and geometrical spaces. Furthermore, unlike GS-Net [8], we maintain the dynamic strategy to update the graph dynamically at each layer of the network. In addition, we build the graphs based on dynamic routing between capsules instead of L^2 norm.

2) CAPSULE NETWORKS

The concept of capsule networks was initially mentioned in [30] as a local component of artificial neural networks in which each capsule learns to recognize an implicitly defined visual entity over a limited domain of viewing conditions. This idea of capsules was later developed into a dynamic routing mechanism between capsules in [12] as CapsNet (capsule networks). In this model, the information is stored at the vector level instead of scalar and capsules are groups of neurons that act together. CapsNet [12] used the concept of iterative routing-by-agreement and squashing function on the output vector to get the activation capsules. Capsule network recently has been applied in various lines of research; a thorough review of capsule networks can be found in [31]. Furthermore, this concept has been applied in point clouds processing research [32]–[35]. 3D Point Capsule Networks [32] proposed an auto-encoder designed to process sparse 3D point clouds while maintaining spatial arrangements of the input data. Geometric capsules introduced in [33] to learn object representations from 3D point clouds by bundles of geometrically interpretable hidden units based on pose and features. Quaternion equivariant capsule networks [34] proposed to learn a pose-equivariant representation of objects by building a hierarchy of local reference frames where each frame is modeled as a quaternion. 3DCapsule [35] replaced the common fully connected classifier with a 3D capsule architecture in order to determine the spatial relationship between feature vectors by mapping feature vectors to capsules. GCAPS-CNN [14] is designed to improve the performance of graph classification task by considering a vector-valued capsule function in lieu of scalar function of graph structure without applying dynamic routing between capsules. CapsGNN [13] is a capsule graph neural network architecture and uses node features extracted from GNN (Graph Neural Network) to generate graph embeddings. In this model, basic capsules are from extracted node features based on GNN, and then high-level graph capsules and class capsules are generated through a routing mechanism. Contrary to [13], our DCG-Net is not fixed but rather is dynamically updated after each layer of the network. Furthermore, we initialized it based on the geometrical features and adapt it for employing in point cloud processing tasks.

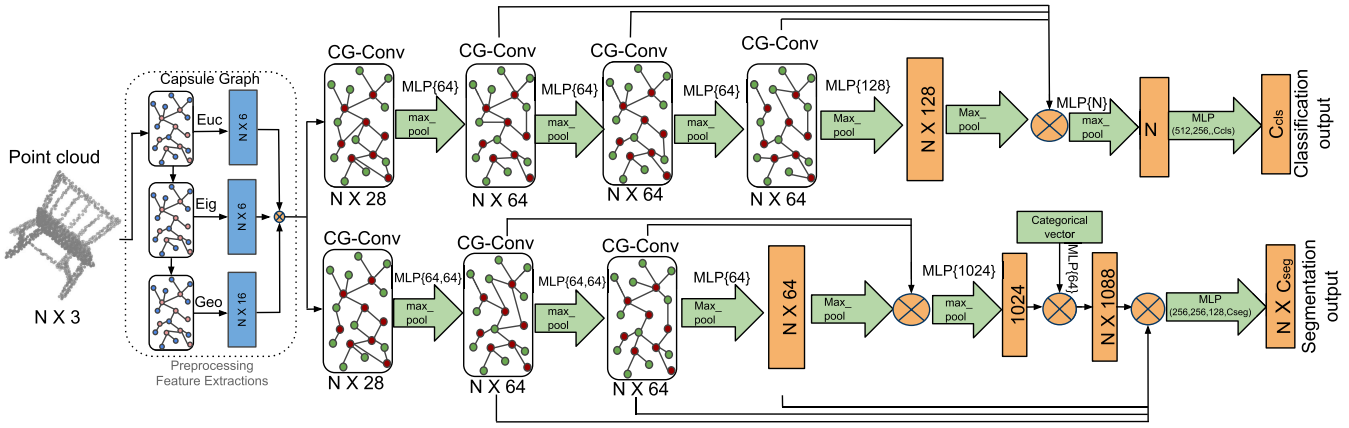

FIGURE 1. Dynamic Capsule Graph Network (DCG-Net) architecture. The architecture of classification model is illustrated in the top branch and the architecture of segmentation model is depicted in the bottom branch. ⊗ stands for concatenation. In the preprocessing step, we concatenate the features from euclidean, eigenvalues and geometrical spaces. The input features of CG-Conv module at the first layer of the main network are based on the combination of features from euclidean, eigenvalues and geometrical spaces. In the next layers, CG-Conv performs merely on the feature space based on convolutions. The classification model takes N points as input, calculates neighborhood aggregation graph features for each point at an CG-Conv layer. The output features of the last CG-Conv layer are stacked up to form an $1D$ global descriptor, which is employed to calculate classification scores for C_{cls} classes. The segmentation model expands the classification model by inserting the $1D$ global descriptor and for each point all the CG-Conv outputs are considered as local descriptors. The segmentation scores are calculated as per-point scores for C_{seg} semantic labels.

III. PROPOSED METHOD

In this work we introduce DCG-Net for two tasks of classification and part segmentation as shown in Figure 1. The details of DCG-Net are explained in the following sections.

A. CAPSULE GRAPH

In this work instead of operating on each point to build a neighborhood aggregation graph, we exploit the local features based on the dynamic routing between capsules. We incorporate the concept of dynamic routing inspired from [12] to select the k nearest neighbors. In this work, we do not exploit the neighbors based on the L^2 norm, but we select the neighbors based on the similarity between the features of each point, and we extract the similarities by dynamic routing between capsules. The difference of our proposed technique with [12] is that we omit the usage of the convolutional layers for computing the features of the child capsule layer. We directly compute the features of the child capsule layer from the features extracted at each stage of the model and apply the dynamic routing mechanism to search for the k nearest neighbor features. This dynamic routing mechanism is based on the child and parent capsule approach as shown in Figure 2.

Initially, all the features are present in the layer of child capsules, and for each point, we search to find the neighboring points with similar features. The points with similar features are present in the parent capsule. We focus on designing the routing mechanism between the child and parent capsules in such a way that those features which are similar to the parent capsule, will contribute more and those points which are not similar will contribute less from the child capsule.

The first step is to compute the features of the child capsules f_i. The child capsule layer consists of the features $f = \{f_1, f_2, \ldots, f_n\}$ where n is the number of capsules.

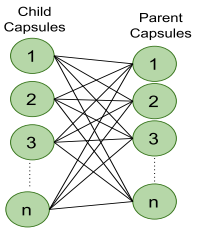

FIGURE 2. Child and parent capsule approach for dynamic routing mechanism between capsules in order to search for k nearest neighbors based on the vector level similarities between the points of a point cloud.

Here, we consider the number of capsules equal to the number of points in the point cloud. The features (f) are computed by $f = \mathcal{X}^T\mathcal{X}$, where $\mathcal{X} = \{x_1, \ldots, x_n\} \subset \mathbb{R}^{\mathcal{F}}$, $\mathcal{F}$ represents the feature dimensionality of a given input. The fact of multiplying each input ($\mathcal{X}$) with its transposed ($\mathcal{X}^T$) is to reproduce the features of each point by multiplying with features of the other points, including itself. Then, after computing f, we apply the dynamic routing mechanism between the child and parent capsule.

In the dynamic routing mechanism, there are coupling coefficients denoted by C between the parent and child capsule layers, where $C = \{C_{11}, C_{12}, \ldots, C_{ij}, \ldots, C_{nn}\}$, where i stands for child capsules and j stands for parent capsules, $1 \leq i \leq n$, $1 \leq j \leq n$, and the summation of all of them is equal to one as shown in Figure 3. The coefficient values are computed by a *routing softmax* which is initially defined by b_{ij}. The b_{ij} represents the logit of i-th index of the child capsule and j-th index of the parent capsule. The b_{ij} is initialized with 0 at the first iteration of dynamic routing mechanism, then it is rectified by evaluating the agreement between the current output of each capsule iteratively. Hence, coupling coefficient values are computed as $C_{ij} = \frac{exp(b_{ij})}{\sum_{z=1}^{n} exp(b_{iz})}$, where

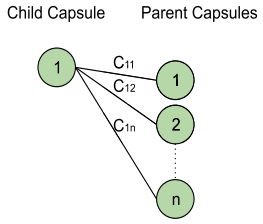

FIGURE 3. The coefficient values between capsules, where $\sum_{j=1}^{n} C_{1j} = 1$ and $1 \leq j \leq n$. The number of capsules is n which is equal to the number of points in the given point cloud.

z represents the iterator through all the parent capsules. The coupling coefficients are the log prior probabilities that the child capsule layer i should be coupled to parent capsule layer j, and they are updated by the iterative routing process.

Afterward, by multiplication of all the features of the child capsules f with the routing coefficient C, we obtain the list of prediction vectors $\overrightarrow{P} = \{\overrightarrow{P}_{11}, \overrightarrow{P}_{21}, \ldots, \overrightarrow{P}_{ij}, \ldots, \overrightarrow{P}_{nn}\}$ for each vector of child capsule $\overrightarrow{P}_{ij} = C_{ij} f_i$. Each vector in parent capsules layers consists of a weighted sum (S_j) of all prediction vectors in order to connect child capsules layers with the parent capsules $S_j = \sum_{i=1}^{n} \overrightarrow{P}_{ij}$. The list of prediction vectors $\overrightarrow{P}$ represents the probability of similarities of features with respect to the features of each point. Hence, in order to define more similar prediction vectors close to 1 and less similar prediction vectors close to 0, we apply a non-linear squashing function [12], $V_j = \frac{\|S_j\|^2}{1+\|S_j\|^2} \frac{S_j}{\|S_j\|}$, where V_j is the vector output of capsule j and S_j is its total input.

The output vector after performing the routing mechanism represents a probability to indicate the similarity of the neighboring points with the individual point. The points with similar features tend to have a higher probability in comparison with the points with fewer feature similarities. Therefore, at each iteration of the routing mechanism, the vector length is reduced close to 0 for dissimilar features and is increased close to 1 for similar features. This process is performed by updating the routing coefficients as $b_{ij} = b_{ij} + V_j.f_i$ (i.e. we make a dot product of V_j and f_i). After the routing mechanism, we get the most similar features for each point and thus getting the most similar neighbors for constructing the local graph. An overview of the dynamic routing mechanism is illustrated in Algorithm 1.

For simplicity, an example is shown in Figure 4. Here, we consider three features f_1, f_2 and f_3 in three child capsules contributing to the parent capsule. The coupling coefficients C_{11}, C_{21} and C_{31} are updated using dynamic routing, where we search to find the similarity of the resultant vector with each of the individual vectors. Prediction vectors are denoted by $\overrightarrow{P}_{11}$, $\overrightarrow{P}_{21}$ and $\overrightarrow{P}_{31}$. Each prediction vector $\overrightarrow{P}_{ij}$ from a child capsule to a parent capsule is computed by multiplication of the correspondent feature f_i with the coupling coefficient C_{ij}, hence, $\overrightarrow{P}_{i1} = C_{i1} f_i$, where $1 \leq i \leq 3$. The output of the parent capsule is denoted by S_1, and is generated by the summation of prediction vectors $S_1 = \sum_{i=1}^{3} \overrightarrow{P}_{i1}$. The squashed output of the parent capsule is denoted by V_1, where

Algorithm 1 Dynamic Routing of Capsule Graph

Result: Neighborhood aggregation graphs of point clouds.
Initialization: Features from euclidean, eigenvalues, geometrical spaces or feature space from convolutions.
Procedure: Routing (f_i, r, l)
for child capsule i in layer l and parent capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$
for r iterations **do**
  for all capsule i in layer l: $C_{ij} \leftarrow softmax(b_{ij})$
  for all capsule i in layer l: $\overrightarrow{P}_{ij} \leftarrow C_{ij} f_i$
  for all capsule j in layer $(l+1)$: $S_j \leftarrow \sum_{i=1}^{n} \overrightarrow{P}_{ij}$
  for all capsule j in layer $(l+1)$: $V_j \leftarrow squash(S_j)$
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + V_j.f_i$
return V_j

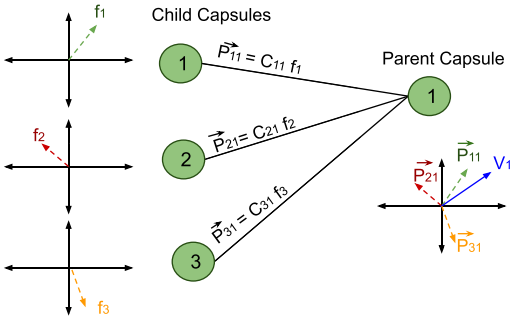

FIGURE 4. The visualization of computing the vectors of capsules in dynamic routing mechanism. f_1, f_2 and f_3 are features for child capsules. C_{11}, C_{21} and C_{31} are coupling coefficients between child and parent capsules. $\overrightarrow{P}_{11}$, $\overrightarrow{P}_{21}$ and $\overrightarrow{P}_{31}$ are prediction vectors. V_1 is the squashed output of the parent capsule. Output of the parent capsule is $S_1 = \sum_{i=1}^{3} \overrightarrow{P}_{i1}$ and its squashed is $V_1 = \frac{\|S_1\|^2}{1+\|S_1\|^2} \frac{S_1}{\|S_1\|}$. The benefit of squashed function is to divert the short vectors to almost zero length and long vectors to a length slightly below one.

$V_1 = \frac{\|S_1\|^2}{1+\|S_1\|^2} \frac{S_1}{\|S_1\|}$. After calculating the output vector S_1, the aim is to find the similarity of the resultant vector with each of the individual vectors, and this process is performed by applying dot product of each of individual vectors with the resultant vectors. The resultant dot product will be high for the $\overrightarrow{P}_{11}$, and the resultant dot product will be less for the $\overrightarrow{P}_{31}$ in comparison with the $\overrightarrow{P}_{21}$. Thus, the resulting vector comparison will be $\overrightarrow{P}_{11} > \overrightarrow{P}_{21} > \overrightarrow{P}_{31}$, which demonstrates the contribution of all these three prediction vectors in the resultant vector V_1 at the parent capsule. The agreement process is performed according to a log likelihood and it is added to the initial logit, b_{ij} before calculating each new value of each coupling coefficient.

After the dynamic routing mechanism, we obtain the vectors to exploit the similarities between all the points of a point cloud. Then, for each point, we select the k points with the most similar features to build the *Neighbors Graph* as shown in Figure 5 inside the *Capsule Graph* block. The *Neighbors Graph* applies to define the structure and topology of the

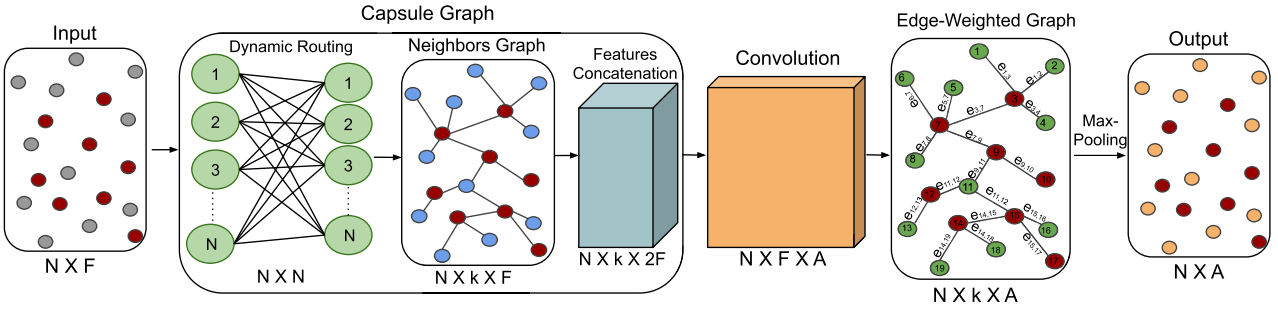

FIGURE 5. Capsule Graph Convolution (CG-Conv) module. The input of this module is a point cloud of shape $N \times F$. By a dynamic routing mechanism the k most similar points of each point are exploited, and a neighbors graph is built of a shape $N \times k \times F$. The Features Concatenation function gives an output containing all the features and differences of features of the neighbors of each point and in a shape of $N \times k \times 2F$. The edge features of the graphs are computed by a convolution of shape $N \times F \times A$ in order to obtain an edge-weighted graph in a shape of $N \times k \times A$. Afterward, a max-pooling applies on edge-weighted graph in order to obtain a point cloud of $N \times A$. In DCG-Net pipeline, the output of each CG-Conv module is the input of the next CG-Conv module. Hence, the structure of the input and output are the same, but they are with different dimensionalities. The parameters of F and A are feature dimensionalities at each layer of the network, and they are variable at different layers of the network.

graph without the weights of the graph's edges. Afterward, from the k nearest neighbors of each point, we compute the features of those similar points by a *Features Concatenation* function shown in Figure 5. In this function, we compute the differences between features as it was proven by [8]. Afterward, we concatenate the features of each anchor point with the difference in feature values for the neighbor points. This concatenation gives an output shape of $N \times k \times 2\mathcal{F}$, where N is the number of points in a point cloud, k is the number of neighbors and $\mathcal{F}$ is the feature dimensionality of the given input. The obtained $2\mathcal{F}$ dimensionality is because of concatenating the features and their differences, some examples are illustrated in Table 1. Afterward, the output of *Features Concatenation* function serves as the input of a convolutional layer to create an edge-weighted graph data structure as described in III-B.

B. CG-CONV: CAPSULE GRAPH CONVOLUTION

In the CG-Conv (Capsule Graph Convolution) module, we apply the convolutional operation on the features of the capsule graph module that we obtained based on the dynamic routing between capsules and feature concatenation function as explained in III-A. Afterward, the output from the convolutional operation is an edge-weighted graph that passes through a max-pooling layer, thus we receive a final output in a point cloud structure. The input and output of each CG-Conv module are in a point cloud structure but with different dimensionalities. This is because of the fact that the output of each CG-Conv serves as the input of the next CG-Conv layer. The CG-Conv operation acts on the edge features, which correspond to the weights of the edges, emerging from each vertex. The CG-Conv operation is illustrated in Figure 5.

The convolutions for each point of the capsule graph can be formulated by the convention definition of continuous convolutions [9] in a $\mathcal{F}$-dimensional space. For a $\mathcal{F}$-dimensional point cloud $\mathcal{X}$ with N points the dimension $\mathcal{F}$ represents the feature dimensionality of a given layer. Continuous convolutions are defined as:

$$(f * g)(x_i) = \int_{-\infty}^{\infty} f(x_j) \odot g(x_i - x_j)\, dx_j \tag{1}$$

where $\odot$ is an element-wise-product of the continuous feature function $f : \mathbb{R}^{\mathcal{F}} \rightarrow \mathbb{R}^{\mathcal{A}}$ assigning a feature-vector $f(x_j) \in \mathbb{R}^{\mathcal{A}}$ to each position $x_j \in \mathbb{R}^{\mathcal{F}}$ and the continuous kernel function $g : \mathbb{R}^{\mathcal{F}} \rightarrow \mathbb{R}^{\mathcal{A}}$ mapping a relative position to a kernel weight. The edge function is defined as e_{ij}, where x_i is an anchor point and x_j is a neighbor point. Then, for this convolution operation we have: $e_{ij} = g(x_i - x_j)$, $1 \leq i, j \leq N$, where recent methods [7], [9] implement the kernel function $g(\cdot)$ as a learned parametric function based on a shared MLP (Multi Layer Perceptron) as: $g(x; \Omega) = \text{MLP}(x; \Omega)$, where x is the correspondent position between two points and Ω is a set of learned parameters.

From the convolutional layer of CG-Conv, we compute a directed edge-weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing point cloud structure based on its features, where the vertices are $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_N\}$ and the edges are $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. We construct $\mathcal{G}$ as a k-NN edge-weighted graph in $\mathbb{R}^{\mathcal{A}}$, where the k nearest neighbors are based on the similarities of the points in $\mathcal{X}$, and the similarities are exploited by dynamic routing mechanism between capsules. The edge-weighted graph $\mathcal{G}$ is constructed by connecting all the nearest neighbours in shape of $N \times k \times \mathcal{A}$. The graph $\mathcal{G}$ includes self-loops, meaning each vertex also points to itself.

Afterward, the max pooling operator applies on the edge-weighted graph to collect and output the resulting point features in shape of $N \times \mathcal{A}$. This operation is invariant with respect to the permutation of the features, since it is a channel-wise symmetric function.

CG-Conv is designed independently of the ordering of the neighbours, this follows the property of permutation invariance.

In DCG-Net architecture, the output of each CG-Conv module is the input of the next CG-Conv module. Hence,

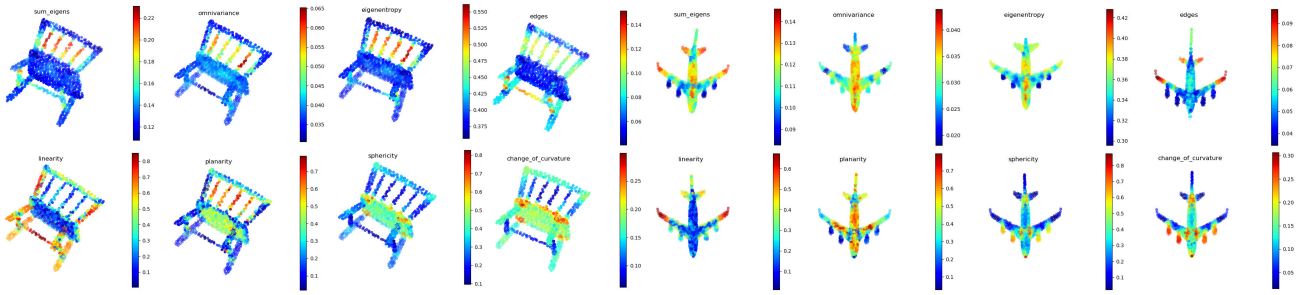

FIGURE 6. The range of the values from the geometrical features of a point cloud with N points. The variety of each of the geometrical features in $\gamma \in \mathbb{R}^8$ (Sum of eigenvalues (γ_i^1), Omnivariance (γ_i^2), Eigenentropy (γ_i^3), Linearity(γ_i^4), Planarity(γ_i^5), Sphericity (γ_i^6), Change of curvature (γ_i^7) and Sharp Edges (γ_i^8), where $1 \leq i \leq N$) illustrates the capability of building a graph beyond euclidean distances between the points and relying on the shape similarities between the points of a point cloud.

the structure of the input and output of CG-Conv are the same, but they are with different dimensionalities. The parameters of $\mathcal{F}$ and $\mathcal{A}$ are feature dimensionalities at each layer of the network, and they are variable at different layers of the DCG-Net.

In this approach we update the graphs by a dynamic routing mechanism after each layer of the network. At each layer the nearest neighbours of each point are updated, thus the graph is dynamically updated at each layer of the network. The advantage of this approach is due to the differences of the proximity in the nearest neighbors of input point cloud and feature space.

C. INPUT FEATURES

We consider a S-dimensional point cloud with N points, denoted by $\mathcal{X} = \{x_1, \ldots, x_N\} \subset \mathbb{R}^S$. Each point of a point cloud commonly contains 3 coordinates $x_i = (x_i^1, x_i^2, x_i^3)$, which means that $S = 3$. Furthermore, it is possible to include further coordinates such as representing color information, normal vectors.

In order to build a neighbors graph, we select the k neighbors of each point based on a dynamic routing mechanism between capsules as explained in III-A. We build the first *Capsule Graph*, by extracting the k nearest neighbors of each point of a raw point cloud in a preprocessing step before feeding the features into the main blocks of the network as it shown in Figure 1. Then, we exploit a combination of euclidean, eigenvalues and geometrical features of input point clouds before applying the convolutional operation. To this end, we compute the covariance matrix to extract the eigenvalues. Covariance is a measurement that explores the variance of each dimension from the mean with respect to each other. From a point cloud with N points, for a 3-dimensional sample point $x_i = (x_i^1, x_i^2, x_i^3)$, where $1 \leq i \leq N$, the 3×3 covariance matrix C_i is given by:

$$C_i = \begin{bmatrix} Cov(x_i^1, x_i^1) & Cov(x_i^1, x_i^2) & Cov(x_i^1, x_i^3) \\ Cov(x_i^2, x_i^1) & Cov(x_i^2, x_i^2) & Cov(x_i^2, x_i^3) \\ Cov(x_i^3, x_i^1) & Cov(x_i^3, x_i^2) & Cov(x_i^3, x_i^3) \end{bmatrix} \quad (2)$$

where, for instance $Cov(x_i^1, x_i^2)$ is the covariance of x_i^1, x_i^2 by k nearest neighbor points is computed as:

$$Cov\left(x_i^1, x_i^2\right) = \frac{\sum_{j=1}^{k} \left(x_j^1 - \bar{x}_i^1\right)\left(x_j^2 - \bar{x}_i^2\right)}{k-1}, \quad (3)$$

where $\bar{x}_i^1$ is the average of the neighbors of x_i over the first dimension and $\bar{x}_i^2$ is the average over the second dimension. The eigenvalues and eigenvectors of x_i are computed as: $C_i V_i = \lambda_i V_i$, where C_i is a 3×3 covariance matrix, V_i is the eigenvectors and λ_i is the eigenvalues. Then, the eigenvalues of the covariance matrix of the point cloud are defined as $\lambda = \{\lambda_1, \ldots, \lambda_N\} \subset \mathbb{R}^3$, each point of a point cloud contains three values in eigenvalues space denoted by $\lambda_i = (\lambda_i^1, \lambda_i^2, \lambda_i^3)$ and ordered as: $\lambda_i^1 \leq \lambda_i^2 \leq \lambda_i^3$.

In the next step, we define the values of each point in the geometrical space by extracting eight geometrical features.

We define geometrical features of each point of a point cloud in $\gamma = \{\gamma_1, \ldots, \gamma_N\} \subset \mathbb{R}^8$. Hence, each point of a point cloud contains 8 values in geometrical space as $\gamma_i = (\gamma_i^1, \gamma_i^2, \ldots, \gamma_i^7, \gamma_i^8)$. Each one of the geometrical features is stated in (4), namely: Sum of eigenvalues (γ_i^1), Omnivariance (γ_i^2), Eigenentropy (γ_i^3), Linearity(γ_i^4), Planarity(γ_i^5), Sphericity (γ_i^6), Change of curvature (γ_i^7) as explained in [36], [37] and Sharp Edges (γ_i^8) as identified in [18], [38].

$$\begin{aligned} \gamma_i^1 &= \sum_{\iota=1}^{3} \lambda_i^\iota, \qquad \gamma_i^2 = \left(\prod_{\iota=1}^{3} \lambda_i^\iota\right)^{\frac{1}{3}}, \\ \gamma_i^3 &= \sum_{\iota=1}^{3} \lambda_i^\iota ln(\lambda_i^\iota), \quad \gamma_i^4 = \frac{\lambda_i^2 - \lambda_i^1}{\lambda_i^2}, \\ \gamma_i^5 &= \frac{\lambda_i^1 - \lambda_i^0}{\lambda_i^2}, \quad \gamma_i^6 = \frac{\lambda_i^0}{\lambda_i^2}, \\ \gamma_i^7 &= \frac{\lambda_i^0}{\lambda_i^0 + \lambda_i^1 + \lambda_i^2}, \quad \gamma_i^8 = \frac{\lambda_i^0}{\lambda_i^2 - \lambda_i^0}. \end{aligned} \quad (4)$$

The geometrical behavior of each one of these features shown in Figure 6.

To initialize the network, we exploit the k most similar points of each point in a combination of the features from

euclidean, eigenvalues and geometrical spaces by the *Capsule Graph* module as explained in section III-A.

The concatenation of all the features from the euclidean, eigenvalues, and geometrical spaces gives us the capability of capturing both local and holistic geometrical features, like symmetry, curvature, convexity, and connectivity. The points in eigenvalues space provide more information about the geometry of the whole point cloud in comparison with the euclidean space which depends just on the euclidean distances between points. Eigenvalues features associate the anchor points with points having similar local geometry, even though these points are far apart from each other in euclidean space. Features in geometrical space are capable of capturing the similarities of the local and global features in the whole shape.

The process of *Capsule Graph* in the first layer of DCG-Net is different from the other layers. The dynamic routing mechanism of *Capsule Graph* module at the first layer is based on the the concatenation of features from euclidean, eigenvalues and geometrical spaces. However, for the next layers, the dynamic routing mechanism is merely based on the features from the previous convolution layer. Therefore, the graphs are updated dynamically based on CG-Conv module by feature space information which is learned by the preceding convolution layers.

D. DYNAMIC CAPSULE GRAPH UPDATE

The advantages of a dynamic graph were shown in DGCNN [7] to recompute the graph by nearest neighbors in the feature space produced at each layer. This is a remarkable privilege of dynamic graph CNN methods from graph CNNs based on a fixed input graph. Updating the graphs dynamically yields the benefit of achieving a receptive field as large as the diameter of the given point cloud. The key difference of our method with DGCNN [7] is due to the dynamic routing between capsules for building the graphs at each layer of the network. Moreover, the other difference of our technique is due to the distinct initialization approach for building the graph based on the geometrical consistency at the first layer of the network.

DCG-Net architecture learns how to build the graph $\mathcal{G}$ at each layer instead of feeding it as a fixed constant constructed. At each layer we have a different graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$, where $\mathcal{G}^{(l)}$ denotes the graph at the l-th layer of the network. We initialize the construction of the graph based on the similarities in euclidean, eigenvalues, and geometrical spaces to extract the closest point by voting and agreement of the capsules. Afterward, for the next layers, the closest points for every single point are taken by voting and agreement of capsules based on the feature space. The robustness of this approach stems from the differences in the similarities between the nearest neighbors at the first layer and the next layers of the network. The nearest neighbors at the first layer are selected based on the similarities of the points due to euclidean, eigenvalues, and geometrical spaces of the input point clouds, but at the next layers, the nearest neighbors are based on the similarities in the convolutional feature space.

E. LOSS FUNCTION

Each task is supervised by applying a one-hot encoding cross-entropy loss function:

$$\mathcal{L}oss = -\sum_{i=1}^{n_c} \Big(h(t_i) \log(p_i) + \big(1 - h(t_i)\big) \log(1 - p_i) \Big), \quad (5)$$

where h is a one-hot encoding function of target labels (t), hence, $h(t)$ refers to one-hot encoded classes which can be considered as labels; and p refers to softmax applied on prediction probabilities, log is a natural logarithm in (5), n_c is the number of classes, referring either to segmentation classes (C_{seg}) or to classification classes (C_{cls}), depending on each task.

IV. EXPERIMENTAL RESULTS

We apply DCG-Net on two tasks of classification and part segmentation.

A. CLASSIFICATION

1) DATASETS

We evaluate our proposed method for the classification task on ModelNet40 [40] dataset which contains 12311 CAD models from 40 categories. 9843 models are used for training and 2468 models are for testing. We have made experiments by extracting and sampling 1024 and 2048 points from the mesh faces, and re-scale it to form a unit sphere, i.e. we feed only the (x, y, z) coordinates of each of the point to the model. We followed the configuration in PointNet [3] to sample points uniformly from the mesh models.

2) ARCHITECTURE

The model used for the classification task is depicted in the top branch of Figure 1. We applied four CG-Conv layers for extracting the features at different scales at the main blocks of the classification model. For the preprocessing CG-Conv layer, we extract features of euclidean, eigenvalues, and geometrical spaces. For each one of these features, we consider the differences to the input features inspired by [8]. Therefore, by concatenating all the features from these three spaces and their differences, we have 28 channels in the first layer of the model. The computation of channels is described in Table 1. For the rest of the layers, we build the graph by the features of the CG-Conv at each layer and feed it as the input for the next CG-Conv layer. Each layer consists of Batch-Normalization layer [43] and Leaky-ReLU [44] activation function with a negative slope of 0.2. After these four CG-Conv Layers, we use a fully connected layer to accumulate all the features which are computed by these four CG-Conv layers (i.e. $64 + 64 + 128 + 256$) to form a 512 dimensional point cloud. These multi-scale features are concatenated using the skip-connections. We also employ the global max/sum pooling function for collecting the global features and two-fully

TABLE 1. Number of channels for each input features. *i* denotes index of anchor point and *j* denotes its neighbors' indices. *x*: coordinates in euclidean space, λ: eigenvalues, γ: values in geometry space. Euc: features in euclidean space and their differences, Eig: features in eigenvalues space and their differences, Geo: features in geometrical space and their differences.

Input features		Channels
Euclidean coordinates	x_i	3
Eigenvalues	λ_i	3
Geometry values	γ_i	8
Euc	$x_i - x_j, x_i$	6
Eig	$\lambda_i - \lambda_j, \lambda_i$	6
Geo	$\gamma_i - \gamma_j, \gamma_i$	16
Euc+Eig	$x_i - x_j, x_i, \lambda_i - \lambda_j, \lambda_i$	12
Euc+Eig+Geo	$x_i - x_j, x_i, \lambda_i - \lambda_j, \lambda_i, \gamma_i - \gamma_j, \gamma_i$	28

connected layers at the end for transforming these features. The dropouts [45] are applied with these fully-connected layers, with a keep probability of 0.5.

3) TRAINING AND RESULTS

We trained the classification model by ADAM optimizer [46] with a learning rate of 0.001, momentum of 0.9, and reduce the learning rate by Cosine annealing untill 0.0001. Cosine Annealing is a learning rate scheduler that has the effect of starting with a large learning rate that is relatively rapidly decreased to a minimum value before being increased rapidly again [47]. We kept the batch-size as 32 and 16 during training and testing respectively, and made the training for 250 epochs. We have initialized the classification network with different features of euclidean, eigenvalues, geometrical, and combinations of them. In all experiments, the number of neighbors (k) is either equal to 20 or 40 when we respectively sampled either 1024 or 2048 points. Furthermore, we have performed an experiment on GS-Net [8] architecture by aggregating features from geometrical space in GS-Net [8] model as well as the features from euclidean and eigenvalues spaces. In this experiment, in the Eigen-Graph module of [8], we obtained the k nearest neighbors of geometrical space along with the k nearest neighbors of euclidean and eigenvalues spaces. Afterward, in GSC (Geometry Similarity Connection) module of [8], we concatenated the features from the three spaces through GroupLayer function of [8]. We followed the exact architecture and setup of GS-Net [8] model for classification by considering the above mentioned changes in Eigen-Graph and GSC modules. The results are summarized in Table 2. Our model achieves state-of-the-art performance (93.4%).

B. PART SEGMENTATION

1) DATASETS

Part segmentation task is a challenging task for shape analysis. We evaluate our method for this task on ShapeNet-Part benchmark [42]. ShapeNet-Part consists of 16880 models from 16 shape categories and 50 different parts in total, with 14006 models for training and 2874 models for testing split. We sampled 2048 points from each model. Each point cloud is annotated with 2 to 6 parts.

TABLE 2. Classification results (%) on ModelNet40 dataset. In our experiments, we initialize the network in euclidean space [Euc] eigenvalues space [Eig], geometrical space [Geo], euclidean and eigenvalue spaces (Euc+Eig) and euclidean, eigenvalue and geometrical spaces [Euc+Eig+Geo]. In GS-Net [8] [Euc+Eig+Geo] experiments, we preserved the architecture of GS-Net by the modification of concatenating the features and neighbors from all the three spaces at each layer.

Method	Input	Accuracy
VRN Single [48]	voxels	91.3
ECC [49]	graphs	87.4
PointNet [3]	1024 points	89.2
SO-Net [39]	2048 points	90.9
PointNet++ [4]	1024 points	90.7
PointNet++ [4]	5000 points+norm	91.9
DGCNN [7]	1024 points	92.2
SpiderCNN [25]	1024 points+norm	92.4
SpiderCNN [25]	1024 points	92.2
3DCapsules [35]	1024 points	92.7
GS-Net [8]	1024 points	92.9
GS-Net [8]	2048 points	93.3
DCG-Net [Euc] (Ours)	1024 points	92.1
DCG-Net [Eig] (Ours)	1024 points	92.4
DCG-Net [Geo] (Ours)	1024 points	92.6
DCG-Net [Euc+Eig] (Ours)	1024 points	92.9
GS-Net [8] [Euc+Eig+Geo]	1024 points	93.0
DCG-Net [Euc+Eig+Geo] (Ours)	1024 points	**93.1**
DCG-Net [Euc](Ours)	2048 points	92.6
DCG-Net [Eig] (Ours)	2048 points	92.8
DCG-Net [Geo] (Ours)	2048 points	93.0
DCG-Net [Euc+Eig] (Ours)	2048 points	93.2
GS-Net [8] [Euc+Eig+Geo]	2048 points	93.3
DCG-Net [Euc+Eig+Geo](Ours)	2048 points	**93.4**

2) ARCHITECTURE

The architecture of the part-segmentation network is illustrated in Figure 1 in the bottom branch. In this model, we employed three CG-Conv layers for features extraction at multiple stages, and a fully-connected layer to aggregate all the information from the previous layers using the skip-connections. Batch-Normalization, dropouts, and Leaky-ReLU activation function are used similarly as in the classification architecture model. There is a categorical vector in this architecture that provides a categorical label based on each part of the point cloud for each point in 64 dimensions. Then, by concatenating it with $1D$ tensor of the point cloud size as 1024, we have an output of size 1088. Then, we concatenate it with the outputs of each one of the layers of this model and pass it to a fully-connected layer to obtain a point-wise score for each part of the point cloud.

3) TRAINING AND RESULTS

The training setup for part segmentation is the same as the classification. We trained our model for 200 epochs. The number of neighbors (k) is set to 40 for 2048 sampled of each model at all the experiments. We choose mIoU (mean Intersection over Union) as the evaluation metric which is averaged across all categories and instances. The results are summarized in Table 3. DCG-Net achieves 82.3% on categories mIoU and 85.4% on instances mIoU. Our method can effectively deal with point clouds with geometric characteristics. Some qualitative results are shown in Figure 7.

TABLE 3. Part segmentation results (%) on ShapeNet-Part dataset. Mean IoU over categories (Cat.) and instances (Ins.).

Method	Cat. mIoU	Ins. mIoU	aero plane	bag	cap	car	chair	earp hon	guitar	knife	lamp	lap top	motor	mug	pistol	rocket	skate board	table
PointNet [3]	80.4	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
SO-Net [39]	81.0	84.6	81.9	83.5	84.8	78.1	**90.8**	72.2	90.1	83.6	82.3	95.2	69.3	94.2	80.0	51.6	72.1	82.6
PointNet++ [4]	81.9	85.1	82.4	79.0	87.7	77.3	**90.8**	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
DGCNN [7]	**82.3**	85.2	**84.0**	83.4	86.7	77.8	90.6	74.7	91.2	**87.5**	82.8	95.7	66.3	94.9	81.1	**63.5**	74.5	82.6
SpiderCNN [25]	81.7	85.3	83.5	81.0	87.2	77.5	90.7	76.8	91.1	87.3	**83.3**	95.8	70.2	93.5	82.7	59.7	75.8	82.8
GS-Net [8]	-	85.3	82.9	**84.3**	**88.6**	78.4	89.7	78.3	**91.7**	86.7	81.2	95.6	**72.8**	94.7	**83.1**	62.3	**81.5**	**83.8**
DCG-Net (Ours)	**82.3**	**85.4**	83.1	84.0	88.2	**78.5**	90.7	**78.7**	91.6	**87.5**	82.5	**95.9**	72.1	**94.8**	82.8	62.1	79.9	82.8

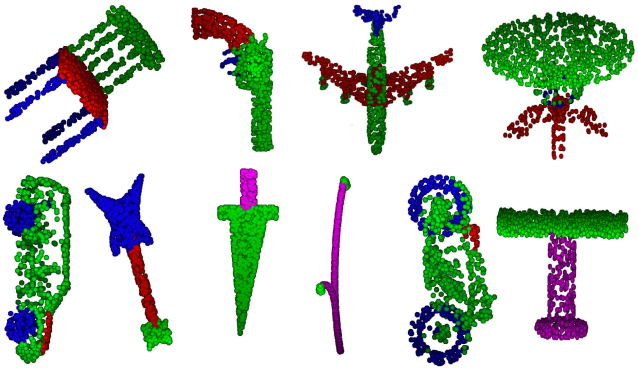

FIGURE 7. Part segmentation examples of ShapeNet dataset.

C. ABLATION STUDY

We perform an ablation study to analyze the effectiveness of our method's components on ModelNet40 bench-mark for the classification task. The results are summarized in Table 2. All experiments in the ablation study are conducted using 1024 points for each point clouds. In Table 4, we summarized the behavior of accuracy for different numbers of neighbors (k) and different initialization metric for selecting the neighbors.

TABLE 4. The behaviour of DCG-Net classification accuracy (%) with different initialization parameters and number of neighbors.

Model	Accuracy (%)				
	k=10	k=15	k=20	k=25	k=30
Euc	88.9	90.3	92.1	90.5	89.6
Eig	89.5	90.8	92.4	90.7	89.8
Geo	90.3	91.3	92.6	90.9	90.1
Euc+Eig	91.1	92.2	92.9	91.7	90.6
Euc+Eig+Geo	92.1	92.5	**93.1**	92.4	91.3

Furthermore, another ablation study of our proposed model is based on determining the adequate number of the applied CG-Conv module in the classification architecture. These experiments are performed on ModelNet40 bench-mark for the classification task. All the experiments are conducted using 1024 points for each point cloud, with 20 being the number of neighbors and initialized by a combined parameters from euclidean, eigenvalue and geometrical spaces. The results are summarized in Table 5.

D. COMPLEXITY ANALYSIS

In our approach, we do not employ any convolutional layer in dynamic routing mechanism for finding similar features. Hence, it leads to a decrease in the computational time.

TABLE 5. Classification accuracy (%) based on the number of CG-Conv modules at the corresponded architecture of DCG-Net.

Number of CG-Conv Modules	1	2	3	4	5
Classification Accuracy (%)	83.7	88.2	90.8	**93.1**	92.5

Our model uses a few parameters which prove that DCG-Net is cost-effective in terms of both time and model size in comparison with 3DCapsules [35]. Moreover, we achieved a better forward-time in comparison with GS-Net [8] because of dynamically updating the graphs and avoid inserting down-sampled features of point clouds at each layer of the network. We evaluate the model complexity in terms of model size and forward time in Table 6. This experiment is conducted on a system with a single GeForce RTX 2080 Ti GPU and implemented by Pytorch. As illustrated, our method made a competitive performance in terms of accuracy by maintaining a reasonable speed and model size.

TABLE 6. Complexity analysis of DGC-Net in classification.

Method	Model size (MB)	Forward time (ms)	Accuracy (%)
PointNet [3]	13.4	**30**	89.2
PointNet++ [4]	7.0	603	91.9
DGCNN [7]	7.2	73	92.2
3DCapsules [35]	52	154	92.7
GS-Net [8]	**6.0**	126	92.9
DCG-Net (ours)	7.2	97	**93.1**

V. CONCLUSION

In this article, we presented DCG-Net as a novel, robust, and efficient approach for point cloud analysis. In DCG-Net, we applied a dynamic routing mechanism between capsules to find similar points based on the features from three different spaces to construct the graphs at the first layer, and then we updated the graphs dynamically at each layer of the network based on the feature space and by dynamic routing between capsules. We have evaluated our proposed technique on ModelNet40 and ShapeNet-Part datasets for classification and part-segmentation tasks respectively. We have identified that the concatenation of features from euclidean, eigenvalues, and geometrical spaces along with the proposed capsule graph convolution module yields a superior performance on classification and part-segmentation tasks of point clouds. We have proven that DCG-Net is computationally efficient in terms of time and it also achieved state-of-the-art results on both classification and part-segmentation tasks. In future work, we plan to extend this approach to semantic segmentation applications in large scale point clouds.

Furthermore, we plan to employ CG-Conv module to design a network architecture to be applied on bioinformatics and social network datasets.

REFERENCES

[1] W. Liu, J. Sun, W. Li, T. Hu, and P. Wang, "Deep learning on point clouds and its application: A survey," *Sensors*, vol. 19, no. 19, pp. 4188–4210, 2019.
[2] E. Grilli, F. Menna, and F. Remondino, "A review of point clouds segmentation and classification algorithms," *ISPRS-Int. Arch. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. 42, pp. 339–344, Feb. 2017.
[3] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 652–660.
[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5099–5108.
[5] D. Griffiths and J. Boehm, "A review on deep learning techniques for 3D sensed data classification," *Remote Sens.*, vol. 11, no. 12, pp. 1499–1528, 2019.
[6] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3D point clouds: A survey," 2019, *arXiv:1912.12033*. [Online]. Available: http://arxiv.org/abs/1912.12033
[7] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 1–12, Nov. 2019.
[8] M. Xu, Z. Zhou, and Y. Qiao, "Geometry sharing network for 3D point cloud classification and segmentation," 2019, *arXiv:1912.10644*. [Online]. Available: http://arxiv.org/abs/1912.10644
[9] F. Engelmann, T. Kontogianni, and B. Leibe, "Dilated point convolutions: On the receptive field size of point convolutions on 3D point clouds," in *Proc. Int. Conf. Robot. Automat. (ICRA)*, May 2020. [Online]. Available: https://francisengelmann.github.io/DPC/
[10] L. Pan, C.-M. Chew, and G. H. Lee, "PointAtrousGraph: Deep hierarchical encoder-decoder with point atrous convolution for unorganized 3D points," 2019, *arXiv:1907.09798*. [Online]. Available: http://arxiv.org/abs/1907.09798
[11] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4558–4567.
[12] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3856–3866.
[13] Z. Xinyi and L. Chen, "Capsule graph neural network," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–16.
[14] S. Verma and Z. Zhang, "Graph capsule convolutional neural networks," in *Proc. Joint ICML IJCAI Workshop Comput. Biol.*, 2018, pp. 1–12.
[15] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "RandLA-Net: Efficient semantic segmentation of large-scale point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11108–11117.
[16] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Sci. Robot.*, vol. 4, no. 26, pp. 1–11, 2019.
[17] P. Raina, S. Mudur, and T. Popa, "Sharpness fields in point clouds using deep learning," *Comput. Graph.*, vol. 78, pp. 37–53, Feb. 2019.
[18] D. Bazazian, J. R. Casas, and J. Ruiz-Hidalgo, "Fast and robust edge extraction in unorganized point clouds," in *Proc. Int. Conf. Digit. Image Comput., Techn. Appl. (DICTA)*, Nov. 2015, pp. 1–8.
[19] D. Moon, S. Chung, S. Kwon, J. Seo, and J. Shin, "Comparison and utilization of point cloud generated from photogrammetry and laser scanning: 3D world model for smart heavy equipment planning," *Autom. Construct.*, vol. 98, pp. 322–331, Feb. 2019.
[20] S. Puttagunta, F. Chraim, A. Gupta, S. Harvey, J. Creadore, and G. Mills, "Real time machine vision and point-cloud analysis for remote sensing and vehicle control," U.S. Patent 10 549 768, Feb. 4, 2020.
[21] W. Wang, R. Yu, Q. Huang, and U. Neumann, "SGPN: Similarity group proposal network for 3D point cloud instance segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2569–2578.
[22] M. Jaritz, J. Gu, and H. Su, "Multi-view PointNet for 3D scene understanding," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 1–9.
[23] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su, "PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 909–918.
[24] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.
[25] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "SpiderCNN: Deep learning on point sets with parameterized convolutional filters," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 87–102.
[26] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. Guibas, "KPConv: Flexible and deformable convolution for point clouds," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6411–6420.
[27] H. Zhao, L. Jiang, C.-W. Fu, and J. Jia, "PointWeb: Enhancing local neighborhood features for point cloud processing," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5565–5573.
[28] X. Liang and Z. Fu, "MHNet: Multiscale hierarchical network for 3D point cloud semantic segmentation," *IEEE Access*, vol. 7, pp. 173999–174012, 2019.
[29] L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan, "Graph attention convolution for point cloud semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10296–10305.
[30] G. Hinton, A. Krizhevsky, and S. Wang, "Transforming auto-encoders," in *Proc. Int. Conf. Artif. Neural Netw.*, 2011, pp. 44–51.
[31] M. K. Patrick, A. F. Adekoya, A. A. Mighty, and B. Y. Edward, "Capsule networks—A survey," *J. King Saud Univ.-Comput. Inf. Sci.*, to be published. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157819309322
[32] Y. Zhao, T. Birdal, H. Deng, and F. Tombari, "3D point capsule networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1009–1018.
[33] N. Srivastava, H. Goh, and R. Salakhutdinov, "Geometric capsule autoencoders for 3D point clouds," 2019, *arXiv:1912.03310*. [Online]. Available: http://arxiv.org/abs/1912.03310
[34] Y. Zhao, T. Birdal, J. E. Lenssen, E. Menegatti, L. Guibas, and F. Tombari, "Quaternion equivariant capsule networks for 3D point clouds," 2019, *arXiv:1912.12098*. [Online]. Available: http://arxiv.org/abs/1912.12098
[35] A. Cheraghian and L. Petersson, "3DCapsule: Extending the capsule architecture to classify 3D point clouds," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2019, pp. 1194–1202.
[36] H. Thomas, F. Goulette, J.-E. Deschaud, B. Marcotegui, and Y. LeGall, "Semantic classification of 3D point clouds with multiscale spherical neighborhoods," in *Proc. Int. Conf. 3D Vis. (3DV)*, Sep. 2018, pp. 390–398.
[37] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3D point clouds with strongly varying density," *ISPRS Ann. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. 3, pp. 177–184, Jun. 2016.
[38] D. Bazazian, J. R. Casas, and J. Ruiz-Hidalgo, "Segmentation-based multi-scale edge extraction to measure the persistence of features in unorganized point clouds," in *Proc. 12th Int. Joint Conf. Comput. Vis., Imag. Comput. Graph. Theory Appl.*, 2017, pp. 317–325.
[39] J. Li, B. M. Chen, and G. H. Lee, "SO-Net: Self-organizing network for point cloud analysis," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9397–9406.
[40] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1912–1920.
[41] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on X-transformed points," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 820–830.
[42] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas, "A scalable active framework for region annotation in 3D shape collections," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 1–12, Nov. 2016.
[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[44] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 2015, *arXiv:1505.00853*. [Online]. Available: http://arxiv.org/abs/1505.00853
[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[46] D. Kingma and J. Ba, "ADAM: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.
[47] I. Loshchilov and H. Frank, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–16.
[48] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," 2016, *arXiv:1608.04236*. [Online]. Available: http://arxiv.org/abs/1608.04236
[49] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3693–3702.

DENA BAZAZIAN received the Ph.D. degree from the Computer Vision Center (CVC), Autonomous University of Barcelona (UAB), in 2018. She was a Postdoctoral Researcher at the CVC, UAB. She had long-term research visits at NAVER LABS Europe, Grenoble, France, and the Media Integration and Communication Center (MICC), University of Florence, Italy. She was with the Universitat Politècnica de Catalunya (UPC), from 2013 to 2015, working on sharp edge extraction and feature description of unorganized point cloud. She is currently a Research Scientist at the Centre Tecnològic de Telecomunicacions de Catalunya (CTTC). Her research interests include computer vision and geometric deep learning algorithms to analyze 3D point clouds. She was one of the organizers of Deep Learning for Geometric Computing (DLGC) Workshop at CVPR2020, Women in Computer Vision (WiCV) Workshops at CVPR2018 and ECCV2018, and Robust Reading Challenge on Omnidirectional Video at ICDAR 2017.

DHANANJAY NAHATA received the bachelor's degree in engineering from the Birla Institute of Technology and Science, Pilani, India, in 2019. He is currently pursuing the master's degree with the Universitat Autònoma de Barcelona (UAB), Barcelona, Spain, in the field of computer vision. Before that, he was with the Research and Development Department, SONY Corporation, Tokyo, Japan, working in the field of generative adversarial networks (GANs). He is currently one of the Program Committee members of the Special Session of "Games for Sustainable Development Goals" at IEEE COG 2020. His research interests include applications of deep learning and computer vision for post-earthquake damage assessment.