DeepGCNs: Making GCNs Go as Deep as CNNs

Guohao Li, Matthias Müller, Guocheng Qian, Itzel C. Delgadillo, Abdulallah Abualshour, Ali Thabet, and Bernard Ghanem

Abstract—Convolutional neural networks (CNNs) have been very successful at solving a variety of computer vision tasks such as object classification and detection, semantic segmentation, activity understanding, to name just a few. One key enabling factor for their great performance has been the ability to train very deep networks. Despite their huge success in many tasks, CNNs do not work well with non-Euclidean data, which is prevalent in many real-world applications. Graph Convolutional Networks (GCNs) offer an alternative that allows for non-Euclidean data input to a neural network. While GCNs already achieve encouraging results, they are currently limited to architectures with a relatively small number of layers, primarily due to vanishing gradients during training. This work transfers concepts such as residual/dense connections and dilated convolutions from CNNs to GCNs in order to successfully train very deep GCNs. We show the benefit of using deep GCNs (with as many as 112 layers) experimentally across various datasets and tasks. Specifically, we achieve very promising performance in part segmentation and semantic segmentation on point clouds and in node classification of protein functions across biological protein-protein interaction (PPI) graphs. We believe that the insights in this work will open avenues for future research on GCNs and their application to further tasks not explored in this paper. The source code for this work is available at https://github.com/lightaime/deep_gcns_torch and https://github.com/lightaime/deep_gcns for PyTorch and TensorFlow implementations respectively.

Index Terms—Graph convolutional network, non-Euclidean data, 3D semantic segmentation, node classification, deep learning

1 INTRODUCTION

GCNs have become a prominent research topic in recent years. There are several reasons for this trend, but above all, GCNs promise a natural extension of CNNs to non-Euclidean data. While CNNs are very powerful when dealing with grid-like structured data, e.g., images, their performance on more irregular data, e.g., point clouds, graphs, etc., is sub-par. Since many real-world applications need to leverage such data, GCNs are a very natural fit. There has already been some success in using GCNs to predict individual relations in social networks [1], model proteins for drug discovery [2], [3], enhance predictions of recommendation engines [4], [5], and efficiently segment large point clouds [6]. While these works show promising results, they rely on simple and shallow network architectures.

In the case of CNNs, the primary reason for their continued success and state-of-the-art performance on many computer vision tasks, is the ability to reliably train very deep network architectures. Surprisingly, it is not clear how to train deep GCN architectures and many existing works have investigated this limitation along with other shortcomings of GCNs [7], [8], [9]. Similar to CNNs, stacking multiple layers in GCNs leads to the vanishing gradient problem. The over-smoothing problem can also occur when repeatedly applying many GCN layers [7]. In this case, it was observed that the features of vertices within each connected component will converge to the same value and thus become indistinguishable from each other. As a result, most state-of-the-art GCNs are limited to shallow network architectures, usually no deeper than 4 layers [9].

The vanishing gradient problem is well-known and well-studied in the realm of CNNs. As a matter of fact, it was the key limitation for deep convolutional networks before ResNet [10] proposed a simple, yet effective solution. The introduction of residual connections [10] between consecutive layers addressed the vanishing gradient problem by providing additional paths for the gradient. This enabled deep residual networks with more than a hundred layers (e.g., ResNet-152) to be trained reliably. The idea was further extended by DenseNet [11], where additional connections are added across layers.

Training deep networks reveals another bottleneck, which is especially relevant for tasks that rely on the spatial composition of the input image, e.g., object detection, semantic segmentation, depth estimation, etc. With increased network depth, more spatial information can potentially be lost during pooling. Ideally, the receptive field should increase with network depth without loss of resolution. For CNNs, dilated convolutions [12] were introduced to tackle this issue. The idea is again simple but effective. Essentially, the convolutions are performed across more distant neighbors as the network depth increases. In this way, multiple resolutions can be seamlessly encoded in deeper CNNs. Several innovations, in particular residual/dense connections and dilated convolutions, have enabled reliable training of very deep CNNs achieving state-of-the-art performance on many tasks. This yields the following question: do these innovations have a counterpart in the realm of GCNs?
Contributions. The contributions of this work are threefold. (1) We adapt residual connections, dense connections, and dilated convolutions, which were introduced for CNNs to enable deep GCN architectures, denoted DeepGCNs. (2) We present extensive experiments on point cloud and biological graph data, showing the effect of each component to the stability and performance of training deep GCNs. We use semantic segmentation and part segmentation on point clouds, as well as, node classification of biological networks as our experimental testbeds. (3) We show how these new concepts enable successful training of a 112-layer GCN, the deepest GCN architecture by a large margin. With only 28 layers, we already improve the previous best performance by almost 4 percent in terms of mIOU on the S3DIS dataset [13]; we also outperform previous methods in the task of part segmentation for many classes of PartNet [14]. Similarly, we achieve superior results on the PPI dataset [2] in the task of node classification in biological networks.

A preliminary version of this work was published in [15]. This journal manuscript extends the initial version in several aspects. First, we investigate even deeper DeepGCN architectures with more than 100 layers. Interestingly, we find that divergence occurs when training a PlainGCN with 112 layers, while our proposed counterpart, ResGCN with skip connections and dilated convolutions, converges without problem (see Fig. 1). Second, to investigate the generality of our DeepGCN framework, we perform extensive additional experiments on the tasks of part segmentation on PartNet and node classification on PPI. Third, we examine the performance and efficiency of MRGCN, a memory-efficient GCN aggregator we propose, with thorough experiments on the PPI dataset. Our results show that DeepMRGCN models are able to outperform previous methods. We also demonstrate that MRGCN is very memory-efficient compared to other GCN operators via GPU memory usage experiments. Finally, to ensure the reproducibility of our experiments and contribute to the graph learning research community, we have published code for training, testing, and visualization along with several pretrained models in both TensorFlow and PyTorch. To the best of our knowledge, our work is the first to successfully train deep GCNs beyond 100 layers and achieves superb results on both point cloud and biological graph data.

2 RELATED WORK

A large number of real-world applications deal with non-euclidean data, which cannot be systematically and reliably processed by CNNs in general. To overcome the shortcomings of CNNs, GCNs provide well-suited solutions for non-euclidean data processing, leading to greatly increasing interest in using GCNs for a variety of applications. In social networks [1], graphs represent connections between individuals based on mutual interests/relations. These connections are non-euclidean and highly irregular. GCNs help better estimate edge strengths between the vertices of social network graphs, thus leading to more accurate connections between individuals. Graphs are also used to model chemical molecule structures [2], [3]. Understanding the bio-activities of these molecules can have substantial impact on drug discovery. Another popular use of graphs is in recommendation engines [4], [5], where accurate modelling of user interactions leads to improved product recommendations. Graphs are also popular modes of representation in natural language processing [16], [17], where they are used to represent complex relations between large text units.

GCNs also find many applications in computer vision. In scene graph generation, semantic relations between objects are modelled using a graph. This graph is used to detect and segment objects in images, and also to predict semantic relations between object pairs [18], [19], [20], [21]. Scene
graphs facilitate the inverse process as well, where an image is reconstructed given a graph representation of the scene [22]. Graphs are also used to model human joints for action recognition in video [23], [24]. Moreover, GCNs are a perfect candidate for 3D point cloud processing, especially since the unstructured nature of point clouds poses a representational challenge for systematic research. Several attempts in creating structure from 3D data exist by either representing it with multiple 2D views [25], [26], [27], [28], or by voxelization [29], [30], [31], [32]. More recent work focuses on directly processing unordered point cloud representations [33], [34], [35], [36], [37]. The recent EdgeConv method by Wang et al. [6] applies GCNs to point clouds. In particular, they propose a dynamic edge convolution algorithm for semantic segmentation of point clouds. The algorithm dynamically computes node adjacency at each graph layer using the distance between point features. This work demonstrates the potential of GCNs for point cloud related applications and beats the state-of-the-art in the task of point cloud segmentation. Unlike most other works, EdgeConv does not rely on RNNs or complex point aggregation methods.

Current GCN algorithms including EdgeConv are limited to shallow depths. Recent works have attempted to train deeper GCNs. For instance, Kipf et al. trained a semi-supervised GCN model for node classification and showed how performance degrades when using more than 3 layers [38]. Pham et al. [39] proposed Column Network (CLN) for collective classification in relational learning, where they showed peak performance at 10 layers and degrading performance when using more than 9 layers. Inspired by CNNs, GCNs do not rely on RNNs or complex point aggregation methods.

3 Methodology
3.1 Representation Learning on Graphs

Graph Definition. A graph G is represented by a tuple \( G = (\mathcal{V}, \mathcal{E}) \) where \( \mathcal{V} \) is the set of unordered vertices and \( \mathcal{E} \) is the set of edges representing the connectivity between vertices \( v \in \mathcal{V} \). If \( e_{i,j} \in \mathcal{E} \), then vertices \( v_i \) and \( v_j \) are connected to each other with an edge \( e_{i,j} \).

Graph Convolution Networks. Inspired by CNNs, GCNs intend to extract richer features at a vertex by aggregating features of vertices from its neighborhood. GCNs represent vertices by associating each vertex \( v \) with a feature vector \( h_v \in \mathbb{R}^D \), where \( D \) is the feature dimension. Therefore, the graph \( G \) as a whole can be represented by concatenating the features of all the unordered vertices, i.e., \( h_G = [h_{v_1}, h_{v_2}, \ldots, h_{v_{|\mathcal{V}|}}]^T \in \mathbb{R}^{N \times D} \), where \( N \) is the cardinality of set \( \mathcal{V} \). A general graph convolution operation \( F \) at the \( l \)th layer can be formulated as the following aggregation and update operations

\[
G_{l+1} = F(G_l, W_l) = Update(Aggregate(G_l, W_l^{agg}), W_l^{update}).
\]

where \( G_l = (\mathcal{V}_l, \mathcal{E}_l) \) and \( G_{l+1} = (\mathcal{V}_{l+1}, \mathcal{E}_{l+1}) \) are the input and output graphs at the \( l \)th layer, respectively. \( W_l^{agg} \) and \( W_l^{update} \) are the learnable weights of the aggregation and update functions respectively, and they are the essential components of GCNs. In most GCN frameworks, aggregation functions are used to compile information from the neighborhood of vertices, while update functions perform a non-linear transform on the aggregated information to compute new vertex representations. There are different variants of these two functions. For example, the aggregation function can be a mean aggregator [38], a max-pooling aggregator [6], [33], [42], an attention aggregator [43], or an LSTM aggregator [44]. The update function can be a multi-layer perceptron [42], [45], a gated network [46], etc. More concretely, the representation of vertices is computed at each layer by aggregating features of neighbor vertices for all \( v_{l+1} \in \mathcal{V}_{l+1} \) as follows:

\[
h_{v_{l+1}} = \phi(h_{v_l}, \rho([ h_{v_l} | u \in \mathcal{N}(v_l) ], h_{v_l}, W_{\rho}), W_\phi),
\]

where \( \rho \) is a vertex feature aggregation function and \( \phi \) is a vertex feature update function, \( h_{v_l} \) and \( h_{v_{l+1}} \) are the vertex features at the \( l \)th layer and \((l+1)\)th layer, respectively. \( \mathcal{N}(v_l) \) is the set of neighbor vertices of \( v_l \) at the \( l \)th layer, and \( h_{v_l} \) is the feature of those neighbor vertices parametrized by \( W_{\rho} \). \( W_\phi \) contains the learnable parameters of these functions. For simplicity and without loss of generality, we use a max-pooling vertex feature aggregator, without learnable
parameters, to pool the difference of features between vertex \( v_l \) and all of its neighbors: 
\[
\rho(\cdot) = \max(h_{u_i} - h_{u_j} | u_i \in \mathcal{N}(v_l))
\]
We then model the vertex feature updater \( \phi \) as a multi-layer perceptron (MLP) with batch normalization [47] and a ReLU as an activation function. This MLP concatenates \( h_{u_i} \) with its aggregate features from \( \rho(\cdot) \) to form its input.

**Dynamic Edges.** As mentioned earlier, most GCNs have fixed graph structures and only update the vertex features at each iteration. Recent work [6], [48], [49] demonstrates that dynamic graph convolution, where the graph structure is allowed to change in each layer, leads to a better graph representations compared to GCNs with fixed graph structure. For instance, ECC (Edge-Conditioned Convolution) [48] uses dynamic edge-conditional filters to learn an edge-specific weight matrix. Moreover, EdgeConv [6] finds the nearest neighbors in the current feature space to reconstruct the graph after every EdgeConv layer. In order to learn to generate point clouds, Graph-Convolution GAN (Generative Adversarial Network) [49] also applies \( k \)-NN graphs to construct the neighborhood of each vertex in every layer.

We find that dynamically changing neighbors in GCNs results in an effectively larger receptive field, when deeper GCNs are considered. In our framework, we propose to re-compute edges between vertices via a Dilated \( k \)-NN function in the feature space of each layer to further increase the receptive field.

Designing deep GCN architectures [8], [9] is an open problem in the graph learning domain. Recent work [7], [8], [9] suggests that GCNs do not scale well to deep architectures, since stacking multiple layers of graph convolutions leads to high complexity in back-propagation. As such, most state-of-the-art GCN models are usually quite shallow [9]. Inspired by the huge success of ResNet [10], DenseNet [11], and Dilated Convolutions [12], we transfer these ideas to GCNs to unleash their full potential. This enables much deeper GCNs that reliably converge in training and achieve superior performance in inference. In what follows, we provide a detailed description of three operations that can enable much deeper GCNs: residual connections, dense connections, and dilated aggregation.

### 3.2 Residual Connections for GCNs

In the original graph learning framework, the underlying mapping \( \mathcal{F} \), which takes a graph as an input and outputs a new graph representation (see Equation (1)), is learned. Here, we propose a graph residual learning framework that learns an underlying mapping \( \mathcal{H} \) by fitting another mapping \( \mathcal{F} \). After \( \mathcal{G}_l \) is transformed by \( \mathcal{F} \), vertex-wise addition is performed to obtain \( \mathcal{G}_{l+1} \). The residual mapping \( \mathcal{F} \) learns to take a graph as input and outputs a residual graph representation \( \mathcal{G}^{res}_{l+1} \) for the next layer. \( \mathcal{W}_l \) is the set of learnable parameters at layer \( l \). In our experiments, we refer to our residual model as ResGCN.

\[
\mathcal{G}_{l+1} = \mathcal{H} (\mathcal{G}_l; \mathcal{W}_l) = \mathcal{F} (\mathcal{G}_l; \mathcal{W}_l) + \mathcal{G}_l = \mathcal{G}^{res}_{l+1} + \mathcal{G}_l. \tag{3}
\]

**Dense Connections for GCNs.** DenseNet [11] was proposed to exploit dense connectivity among layers, which improves information flow in the network and enables efficient reuse of features among layers. Inspired by DenseNet, we adapt a similar idea to GCNs so as to exploit information flow from different GCN layers. In particular, we have

\[
\mathcal{G}_{l+1} = \mathcal{H} (\mathcal{G}_l; \mathcal{W}_l) = \mathcal{T} (\mathcal{F} (\mathcal{G}_l; \mathcal{W}_l), \mathcal{G}_l) = \mathcal{T} (\mathcal{F} (\mathcal{G}_l; \mathcal{W}_l), \ldots, \mathcal{F} (\mathcal{G}_l; \mathcal{W}_0), \mathcal{G}_0). \tag{4}
\]

The operator \( \mathcal{T} \) is a vertex-wise concatenation function that densely fuses the input graph \( \mathcal{G}_0 \) with all the intermediate GCN layer outputs. To this end, \( \mathcal{G}_{l+1} \) consists of all the GCN transitions from previous layers. Since we fuse GCN representations densely, we refer to our dense model as DenseGCN. The growth rate of DenseGCN is equal to the dimension \( D \) of the output graph (similar to DenseNet for CNNs [11]). For example, if \( \mathcal{F} \) produces a \( D \) dimensional vertex feature, where the vertices of the input graph \( \mathcal{G}_0 \) are \( D_0 \) dimensional, the dimension of each vertex feature of \( \mathcal{G}_{l+1} \) is \( D_0 + D \times (l+1) \).

### 3.3 Dilated Aggregation for GCNs

Dilated wavelet convolution is an algorithm originating from the wavelet processing domain [50], [51]. To alleviate spatial information loss caused by pooling operations, Yu et al. [12] propose dilated convolutions as an alternative to applying consecutive pooling layers for dense prediction tasks, e.g., semantic image segmentation. Their experiments demonstrate that aggregating multi-scale contextual information using dilated convolutions can significantly increase the accuracy on the semantic segmentation task. The reason behind this is the fact that dilation enlarges the receptive field without loss of resolution. We believe that dilation can also help with the receptive field of DeepGCNs.

Therefore, we introduce dilated aggregation to GCNs. There are many possible ways to construct a dilated neighborhood. We use a Dilated \( k \)-NN to find dilated neighbors after every GCN layer and construct a Dilated Graph. In particular, for an input graph \( \mathcal{G} = (V, E) \) with Dilated \( k \)-NN and \( d \) as the dilation rate, the Dilated \( k \)-NN operation returns the \( k \) nearest neighbors within the \( k \times d \) neighborhood region by skipping every \( d \) neighbors. The nearest neighbors are determined based on a pre-defined distance metric. In our experiments, we use the \( l_2 \) distance in the feature space of the current layer. Let \( \mathcal{N}^{(d)}(v) \) denote the \( d \)-dilated neighborhood of vertex \( v \). If \( (u_1, u_2, \ldots, u_{k \times d}) \) are the first sorted \( k \times d \) nearest vertices, vertices \( (u_1, u_{1+d}, u_{1+2d}, \ldots, u_{1+(k-1)d}) \) are the \( d \)-dilated neighbors of vertex \( v \) (see Fig. 3), i.e.,

\[
\mathcal{N}^{(d)}(v) = \{ u_1, u_{1+d}, u_{1+2d}, \ldots, u_{1+(k-1)d} \}. \tag{5}
\]

Hence, the edges \( \mathcal{E}^{(d)} \) of the output graph are defined on the set of \( d \)-dilated vertex neighbors \( \mathcal{N}^{(d)}(v) \). Specifically, there exists a directed edge \( e \in \mathcal{E}^{(d)} \) from vertex \( v \) to every vertex \( u \in \mathcal{N}^{(d)}(v) \). The GCN aggregation and update functions are applied, as in Equation (1), by using the edges \( \mathcal{E}^{(d)} \) created by the Dilated \( k \)-NN, so as to generate the feature \( h^{(d)}_v \) of each output vertex in \( v^{(d)} \). We denote this layer operation as a dilated graph convolution with dilation rate \( d \), or more formally:

\[
\mathcal{G}^{(d)} = (v^{(d)}, \mathcal{E}^{(d)}). \tag{6}
\]

We visualize and compare it to a conventional dilated convolution used in CNNs in...
Instead of aggregating neighborhood features \( h_v \); \( h_u \) max \( h_v + h_u \); \( h_v \) sum \( + \) leaving neighbors from the set of \( \{v_1, v_2, \ldots, v_{\text{K}}\} \). Inference time, we perform deterministic aggregation without stochasticity, i.e., the dilated neighbors are sampled based on Equation (5) with probability 1.

### 3.4 Deep GCN Variants

In our experiments in the paper, we mostly work with a GCN based on EdgeConv [6] to show how deep GCNs can be trained. However, it is straightforward to build other deep GCNs with the same aforementioned concepts (e.g., residual/dense graph connections and dilated graph convolutions). To show that these concepts are universal operators and can be used for general GCNs, we perform additional experiments. In particular, we build ResGCNs based on GraphSAGE [42] and Graph Isomorphism Network (GIN) [52]. In practice, we find that EdgeConv learns a better representation than the other implementations. However, it is less efficient in terms of memory and computation. Therefore, we also propose a simple GCN operation combining the advantages of both, which we refer to as MRGCN (Max-Relative GCN). In the following, we discuss each GCN operator in detail.

**EdgeConv.** Instead of aggregating neighborhood features directly, EdgeConv [6] proposes to first get local neighborhood information for each neighbor by subtracting the feature of the central vertex from its own feature. In order to train deeper GCNs, we add residual/dense graph connections and dilated graph convolutions to EdgeConv

\[
\begin{align*}
    h_{v_{l+1}}^{\text{res}} &= \max \left( \{\text{mlp}(\text{concat}(h_v, h_u - h_v)) | u_l \in \mathcal{N}^{(d)}(v_l)\} \right), \\
    h_{v_{l+1}} &= h_{v_{l+1}}^{\text{res}} + h_v.
\end{align*}
\]

**GraphSAGE.** GraphSAGE [42] proposes different types of aggregator functions including a Mean aggregator, an LSTM aggregator, and a Pooling aggregator. Their experiments show that the Pooling aggregator outperforms the others. We adapt GraphSAGE with the max-pooling aggregator to obtain ResGraphSAGE

\[
\begin{align*}
    h_{v_{l+1}}^{\text{res}} &= \max \left( \{\text{mlp}(h_v) | u_l \in \mathcal{N}^{(d)}(v_l)\} \right), \\
    h_{v_{l+1}} &= \text{mlp}(\text{concat}(h_v, h_{v_{l+1}}^{\text{res}}(v_l))), \\
    h_{v_{l+1}} &= h_{v_{l+1}}^{\text{res}} + h_v.
\end{align*}
\]

In the original GraphSAGE paper, the vertex features are normalized after aggregation. We implement two variants, one without normalization (see Equation (7)), another with normalization \( h_{v_{l+1}}^{\text{res}} = h_{v_{l+1}}^{\text{res}} / \|h_{v_{l+1}}^{\text{res}}\|_2 \).

**GIN.** The main difference between GIN [52] and other GCNs is that an \( \epsilon \) is learned at each GCN layer to give the central vertex and aggregated neighborhood features different weights. Hence ResGIN is formulated as follows:

\[
\begin{align*}
    h_{v_{l+1}}^{\text{res}} &= \text{mlp}\left( (1 + \epsilon) \cdot h_v + \text{sum}(\{h_u | u_l \in \mathcal{N}^{(d)}(v_l)\}) \right), \\
    h_{v_{l+1}} &= h_{v_{l+1}}^{\text{res}} + h_v.
\end{align*}
\]

**MRGCN.** We find that first using a max aggregator to aggregate neighborhood relative features \( h_u - h_v \), \( u_l \in \mathcal{N}^{(d)}(v_l) \)
is more effective and efficient than aggregating raw neighborhood features $h_{v_l}$, $u_l \in \mathcal{N}(v_l)$ or aggregating features after non-linear transforms. We refer to this simple GCN as MRGCN (Max-Relative GCN). The residual version of MRGCN is as such

$$h_{v_l}^{res} = \max\left(h_{v_l} - h_{v_l}|u_l \in \mathcal{N}(v_l)\right),$$

$$h_{v_l+1}^{res} = mlp\left(concat\left(h_{v_l}, h_{v_l}^{res}\right)\right),$$

$$h_{v_l+1} = h_{v_l+1}^{res} + h_{v_l}.$$  

Here, $h_{v_l+1}$ and $h_{v_l}$ are the hidden states of vertex $v$ at layers $l$ and $l+1$, and $h_{v_l}^{res}$ is the hidden state of the residual graph. All the $mlp$ (multilayer perceptron) functions use a ReLU as activation function; all the $\max$ and $\text{sum}$ functions above are vertex-wise feature operators; $\text{concat}$ functions concatenate features of two vertices into one feature vector. $\mathcal{N}(v_l)$ denotes the neighborhood of vertex $v_l$ obtained from Dilated $k$-NN. MRGCN is more efficient than EdgeConv and GraphSAGE. MRGCN only applies an $mlp$ transform once for each vertex, while EdgeConv needs to apply this transform for every neighbor of each vertex. In contrast to GraphSAGE, MRGCN does not need to apply $mlp$ transforms to the neighborhood features before $\max$ aggregation and the additional computation of subtraction is negligible.

4 EXPERIMENTS ON 3D POINT CLOUDS

We propose ResGCN and DenseGCN to handle the vanishing gradient problem of GCNs. To enlarge the receptive field, we define a dilated graph convolution operator for GCNs. To evaluate our framework, we conduct extensive experiments on the tasks of semantic segmentation and part segmentation on large-scale 3D point cloud datasets and demonstrate that our methods significantly improve performance. In addition, we perform a comprehensive ablation study to show the effect of different components of our framework.

4.1 Graph Learning on 3D Point Clouds

Point cloud segmentation is a challenging task because of the unordered and irregular structure of 3D point clouds. Normally, each point in a point cloud is represented by its 3D spatial coordinates and possibly auxiliary features such as color and/or surface normal. We treat each point as a vertex $v$ in a directed graph $\mathcal{G}$ and we use $k$-NN to construct the directed dynamic edges between points at every GCN layer (refer to Section 3.1). In the first layer, we construct the input graph $\mathcal{G}_0$ by applying a dilated $k$-NN search to find the nearest neighbors in 3D coordinate space. At subsequent layers, we dynamically build the edges using dilated $k$-NN in feature space. For the segmentation task, we predict the categories of all the vertices at the output layer.

4.2 Experimental Setup

We use the overall accuracy (OA) and mean intersection over union (mIoU) across all classes as evaluation metrics. For each class, the IoU is computed as $\frac{TP}{TP + FP + FN}$, where $TP$ is the number of true positive points, $T$ is the number of ground truth points of that class, and $P$ is the number of predicted positive points. We perform the majority of our experiments on semantic segmentation of point clouds on the S3DIS dataset. To motivate the use of DeepGCNs, we do a thorough ablation study on area 5 of this dataset to analyze each component and provide insights. We then evaluate our proposed reference model ResGCN-28 (backbone of 28 layers with residual graph connections and stochastic dilated graph convolutions) on all 6 areas and compare it to the shallow DGCNN baseline [6] and other state-of-the-art methods. In order to validate that our method is general and does not depend on a specific dataset, we also show results on PartNet for the task of part segmentation of point clouds.

4.3 Network Architectures

As shown in Fig. 2, all the network architectures in our experiments have three blocks: a GCN backbone block, a fusion block and an MLP prediction block. The GCN backbone block is the only part that differs between experiments. For example, the only difference between PlainGCN and ResGCN is the use of residual skip connections for all GCN layers in ResGCN. Both have the same number of parameters. We linearly increase the dilation rate $d$ of dilated $k$-NN with network depth. For fair comparison, we keep the fusion and MLP prediction blocks the same for all architectures. The GCN backbone block takes as input a point cloud with 4096 points, extracts features by applying consecutive GCN layers to aggregate local information, and outputs a learned graph representation with 4096 vertices. The fusion and MLP prediction blocks follow a similar architecture as PointNet [33] and DGCNN [6]. The fusion block is used to fuse the global and multi-scale local features. It takes as input the extracted vertex features from the GCN backbone block at every GCN layer and concatenates those features, then passes them through a $1 \times 1$ convolution layer followed by max pooling. The latter layer aggregates the vertex features of the whole graph into a single global feature vector, which in return is concatenated with the feature of each vertex from all previous GCN layers (fusion of global and local information). The MLP prediction block applies three MLP layers to the fused features of each vertex/point to predict its category. In practice, these layers are $1 \times 1$ convolutions.

PlainGCN. This baseline model consists of a PlainGCN backbone block, a fusion block, and an MLP prediction block. The backbone stacks EdgeConv [6] layers with dynamic $k$-NN, each of which is similar to the one used in DGCNN [6]. No skip connections are used here.

ResGCN. We construct ResGCN by adding dynamic dilated $k$-NN and residual graph connections to PlainGCN. The connections between all GCN layers in the GCN backbone block do not increase the number of parameters.

DenseGCN. Similarly, DenseGCN is built by adding dynamic dilated $k$-NN and dense graph connections to the PlainGCN. As described in Section 3.2, dense graph connections are created by concatenating all the intermediate graph representations from previous layers. The dilation rate schedule of our DenseGCN is the same as for ResGCN.

4.4 Implementation

For semantic segmentation on S3DIS [13], we implement our models using TensorFlow, and for part segmentation...
on PartNet [14], we implement them using PyTorch. For fair comparison, we use the Adam optimizer with the same initial learning rate 0.001 and the same learning rate schedule for all experiments; the learning rate decays 50 percent every $3 \times 10^5$ gradient decent steps. Batch normalization is applied to every layer. Dropout with a rate of 0.3 is used at the second MLP layer of the MLP prediction block. As mentioned in Section 3.3, we use dilated $k$-NN with a random uniform sampling probability $\epsilon = 0.2$ for GCNs with dilations. In order to isolate the effect of the proposed DeepGCN architectures, we do not use any data augmentation or post processing techniques. We train our models end-to-end from scratch for 100 epochs. We evaluate every 10th epoch on the test set and report the best result for each model. The networks are trained with two NVIDIA Tesla V100 GPUs using data parallelism in semantic segmentation on S3DIS, and the batch size is set to 8 for each GPU. For part segmentation on PartNet, we set the batch size to 7 and the networks are trained with one NVIDIA Tesla V100 GPU.

4.5 Results

For convenient referencing, we use the naming convention BackboneBlock-#Layers to denote the key models in our analysis. We focus on residual graph connections for our analysis, since ResGCN-28 is easier and faster to train, but we expect that our observations also hold for dense graph connections.

4.5.1 Semantic Segmentation on S3DIS

In order to thoroughly evaluate the ideas proposed in this paper, we conduct extensive experiments on the Stanford large-scale 3D Indoor Spaces Dataset (S3DIS), a large-scale indoor dataset for 3D semantic segmentation of point clouds. S3DIS covers an area of more than $6,000m^2$ with semantically annotated 3D meshes and point clouds. In particular, the dataset contains 6 large-scale indoor areas represented as colored 3D point clouds with a total of 695,878,620 points. As is common practice, we train networks on 5 out of the 6 areas and evaluate them on the left out area.

We begin with the extensive ablation study, where we evaluate the trained models on area 5, after training them on the other areas. Our aim is to shed light on the contribution of each component of our novel network architectures. To this end, we investigate the performance of different ResGCN architectures, e.g., with dynamic dilated $k$-NN, with regular dynamic $k$-NN (without dilation), and with fixed edges. We also study the effect of different parameters, e.g., number of $k$-NN neighbors (4, 8, 16, 32), number of filters (32, 64, 128), and number of layers (7, 14, 28, 56). To ensure that our contributions (residual/dense connections and dilated graph convolutions) are general, we apply them to multiple GCN variants that have been proposed in the literature. Overall, we conduct 25 experiments and report detailed results in Table 1. We also summarize the most important insights of the ablation study in Fig. 4. In the following, we discuss each block of experiments.

Effect of Residual Graph Connections. Our experiments in Table 1 (Reference) show that residual graph connections play an essential role in training deeper networks, as they tend to result in more stable gradients. This is analogous to the insight from CNNs [10]. ResNet [10] adds skip connections between every two convolutional blocks; we conduct experiments with two skip connection variants, ResGCN-28 and ResGCN-28'. ResGCN-28 adds skip connections between every GCN layer while ResGCN-28' adds skip connections between every two GCN layers. We find that ResGCN-28 outperforms ResGCN-28' by 1.63 percent mIOU (refer to Table 1). Thus, we adopt skip connections between every layer for all other ResGCN architectures. When the residual graph connections between layers are removed (i.e., in PlainGCN-28), performance dramatically degrades (-12.18 percent mIoU). Fig. 5 shows the importance of residual graph connections very clearly. As network depth increases, skip connections become critical for convergence. We also show similar performance gains by combining residual graph connections and dilated graph convolutions with other types of GCN layers. These results can be seen in the ablation study Table 1 (GCN Variants) and are further discussed later in this section.

Effect of Dilation. Results in Table 1 (Dilation) [12] show that dilated graph convolutions account for a 2.85 percent improvement in mean IoU (row 3), justified primarily by the expansion of the network’s receptive field. We find that adding stochasticity to the dilated $k$-NN does help performance but not to a significant extent. We also investigate the effect of $\epsilon$ for stochastic dilation and report the results in Table 1 (stochastic). We find that models are robust to different values of $\epsilon$. The mIOUs vary in the range of about 1 percent for different $\epsilon$ values. It is possible to achieve better performance by carefully tuning $\epsilon$. For instance, ResGCN-28 with $\epsilon$ set to 0.4 increases the mIOU by 0.49 percent compared the Reference model with $\epsilon$ equal to 0.2. Interestingly, our results in Table 1 also indicate that dilation especially helps deep networks when combined with residual graph connections (rows 1,13). Without such connections, performance can actually degrade with dilated graph convolutions. The reason for this is probably that these varying neighbors result in ‘worse’ gradients, which further hinder convergence when residual graph connections are not used.

Dilation versus Downsampling. Our proposed dilated graph convolution enlarges the receptive field without the need of downsampling. To show the advantage of dilated graph convolution compared to downsampling, we conduct experiments with U-Net-style [53] ResGCN models and report the results in Table 1 (downsampling). For the U-Net variants, we insert downsampling modules into the backbone of our ResGCN-28. The number of vertices is downsampled by one half after every 7 layers. The final features are fed into the sequential interpolation layers to gradually upsample the features back to their original resolution. We experiment with two popular downsampling methods in the point cloud literature, i.e., random down sampling [54] and farthest point sampling (FPS) [34]. We denote the architectures as ResGCN-28-random and ResGCN-28-FPS respectively. We observe that ResGCN-28 with dilated graph convolutions outperforms ResGCN-28-random by 4.51 percent and ResGCN-28-FPS by 9.06 percent respectively, which clearly shows the advantage of dilation. However, it is worth mentioning that U-Net like structures with downsampling are used in recent SOTA methods such as KPCConv [55] and RandLA-Net [54] and achieve very promising results. The results of ResGCN-28-random and ResGCN-28-
could be sub-optimal. We conjecture that with more sophisticated architectural designs and hyper-parameter twists, U-Net like GCN architectures could reach more reasonable performance.  

**Effect of Dynamic $k$-NN.** While we observe an improvement when updating the $k$ nearest neighbors after every layer, we would also like to point out that it comes at a relatively high computational cost. We show different variants without dynamic edges in Table 1 (Fixed $k$-NN).  

**Effect of Dense Graph Connections.** We observe similar performance gains with dense graph connections (DenseGCN-28) in Table 1 (Connections). However, with a naive implementation, the memory cost is prohibitive. Hence, the largest model we can fit into GPU memory uses only 32 filters and 8 nearest neighbors, as compared to 64 filters and 16 neighbors in the case of its residual counterpart ResGCN-28. Since the performance of these two deep GCN variants is similar, residual connections are more practical for most use cases and hence we focus on them in our ablation study. Yet, we do expect the same insights to transfer to the case of dense graph connections.  

**Effect of Nearest Neighbors.** Results in Table 1 (Neighbors) show that a larger number of neighbors helps in general. As the number of neighbors is decreased by a factor of 2 and 4, the performance drops by 2.51 percent and 3.27 percent respectively. However, a large number of neighbors only results in a performance boost, if the network capacity is sufficiently large. This becomes apparent when we increase the number of neighbors by a factor of 2 and decrease the number of filters by a factor of 2 (refer to row 3 in Neighbors).

![Ablation study on area 5 of S3DIS](image)

**TABLE 1:** Ablation Study on Area 5 of S3DIS

| Ablation | Model | Operator | mIoU | $\Delta$ mIoU | dynamic | connection | dilation | sto. eps. | # NNs | # filters | # layers |
|----------|-------|----------|------|---------------|---------|------------|---------|----------|-------|-----------|---------|
| Reference | ResGCN-28 | EdgeConv | 52.49 | 0.00 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |
| **Dilation** | | | | | | | | | | | |
| | EdgeConv | 51.98 | -0.51 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.0 | 16 | 64 | 28 |
| | EdgeConv | 52.98 | 0.49 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.4 | 16 | 64 | 28 |
| | EdgeConv | 51.92 | -0.57 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.6 | 16 | 64 | 28 |
| | EdgeConv | 52.35 | -0.14 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.8 | 16 | 64 | 28 |
| | EdgeConv | 51.40 | -1.08 | $\checkmark$ | $\oplus$ | $\checkmark$ | 1.0 | 16 | 64 | 28 |
| | EdgeConv | 49.64 | -2.85 | $\checkmark$ | $\oplus$ | $\checkmark$ | 16 | 64 | 28 |
| | EdgeConv | 40.31 | -12.18 | $\checkmark$ | $\oplus$ | $\checkmark$ | 16 | 64 | 28 |
| **Downsampling** | | | | | | | | | | | |
| | EdgeConv | 47.98 | -4.51 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |
| | EdgeConv | 44.30 | -8.19 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |
| **Fixed $k$-NN** | | | | | | | | | | | |
| | EdgeConv | 48.38 | -4.11 | $\checkmark$ | $\oplus$ | $\checkmark$ | 16 | 64 | 28 |
| | EdgeConv | 43.43 | -9.06 | $\checkmark$ | $\oplus$ | $\checkmark$ | 16 | 64 | 28 |
| **Connections** | DenseGCN-28 | EdgeConv | 51.27 | -1.22 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 8 | 32 | 28 |
| | ResGCN-28 & 50.86 | -1.63 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |
| | EdgeConv | 40.47 | -12.02 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |
| | EdgeConv | 38.79 | -13.70 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 8 | 64 | 56 |
| | EdgeConv | 49.23 | -3.26 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 14 |
| | EdgeConv | 47.92 | -4.57 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 7 |
| **Neighbors** | | | | | | | | | | | |
| | EdgeConv | 49.98 | -2.51 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 8 | 64 | 28 |
| | EdgeConv | 49.22 | -3.27 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 4 | 64 | 28 |
| | EdgeConv | 49.18 | -3.31 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 32 | 32 | 28 |
| **Depth** | ResGCN-112 | EdgeConv | 51.97 | -0.52 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 4 | 64 | 112 |
| | ResGCN-56 | EdgeConv | 53.64 | 1.15 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 8 | 64 | 56 |
| | ResGCN-14 | EdgeConv | 49.90 | -2.59 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 14 |
| | ResGCN-7 | EdgeConv | 48.95 | -3.53 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 7 |
| **Width** | ResGCN-28W | EdgeConv | 53.78 | 1.29 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 8 | 128 | 28 |
| | EdgeConv | 48.80 | -3.69 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 32 | 28 |
| | EdgeConv | 45.62 | -6.87 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 16 | 28 |
| **GCN Variants** | GraphSAGE | 49.20 | -3.29 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |
| | GraphSAGE-N | 49.02 | -3.47 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |
| | GIN-e | 42.81 | -9.68 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |
| | ResMRGCN-28 | 51.17 | -1.32 | $\checkmark$ | $\oplus$ | $\checkmark$ | 0.2 | 16 | 64 | 28 |

We compare our reference network (ResGCN-28) with 28 layers, residual graph connections, and dilated graph convolutions to several ablated variants. All models were trained with the same hyper-parameters for 100 epochs on all areas except for area 5, which is used for evaluation. We denote residual and dense connections with the $\oplus$ and $\triangleright$ symbols respectively. $\checkmark$ denotes that residual connections are added between every two GCN layers. We highlight the most important results in bold. $\Delta$ mIoU denotes the difference in mIoU with respect to the reference model ResGCN-28.

FPS could be sub-optimal. We conjecture that with more sophisticated architectural designs and hyper-parameter twists, U-Net like GCN architectures could reach more reasonable performance.

Fig. 4. Ablation study on area 5 of S3DIS. We compare our reference network (ResGCN-28) with 28 layers, residual graph connections, and dilated graph convolutions with several ablated variants. All models were trained with the same hyper-parameters for 100 epochs on all areas except for area 5 of the S3DIS dataset.
Table 1 shows how the performance of different models varies with the increase in the number of layers. All models were trained for 100 epochs on all areas except for area 5 with the same hyper-parameters. Only when residual graph connections are used do the results improve with increasing depth.

Effect of Network Depth. Table 1 (Depth) shows that increasing the number of layers improves network performance, but only if residual graph connections and dilated graph convolutions are used, as is clearly shown in Table 1 (Connections).

Effect of Network Width. Results in Table 1 (Width) show that increasing the number of filters leads to a similar increase in performance as increasing the number of layers. In general, a higher network capacity enables learning nuances necessary for succeeding in corner cases.

GCN Variants. Our experiments in Table 1 (GCN Variants) show the effect of using different GCN operators. The results clearly show that different deep GCN variants with residual graph connections and dilated graph convolutions converge better than PlainGCN. Using our proposed MRGCN operator achieves comparable performance to the ResGCN reference model, which relies on EdgeConv, while only using half the GPU memory. The GraphSAGE operator performs slightly worse and our results also show that using normalization (i.e., GraphSAGE-N) is not essential. Interestingly, when using the GIN-ε operator, the network converges well during the training phase and has a high training accuracy but fails to generalize to the test set. This phenomenon is also observed in the original paper [52], in which the best performance is achieved when ε is set to 0.

Table 2 shows the effect of using different GCN operators and different network widths. All models were trained for 100 epochs on all areas except for area 5 with the same hyper-parameters. Residual connections and dilated graph convolutions are used to alleviate the over-smoothing issue.

Effect of Batch Normalization. Batch normalization (BN) [47] is a common technique used to reduce internal covariant shift when training deep CNNs. We adopt BN in every GCN block of all the models shown in Table 1 by default. To show the effect of BN, we conduct further experiments on area 5 where we remove BN from ResGCN-28 and ResGCN-7. The results in Fig. 6 show that the performance of ResGCN-28 and ResGCN-7 drops by 6.1 and 4.0 percent respectively when BN is removed. ResGCN-28 w/o BN is even outperformed by ResGCN-7 w/o BN. This indicates that BN is essential for training deep GCNs.

Analysis of Over-Smoothing. To study the effect of residual graph connections and dilated graph convolutions on the over-smoothing issue, we adopt two quantitative metrics proposed in [56], i.e., Group Distance Ratio and Instance Information Gain, to measure the over-smoothness of the learned node representation of the last GCN layer. The group distance ratio measures the ratio of the inter-group distance and the intra-group distance in the euclidean space of the final representation. The instance information gain measures the mutual information between a input node feature and the final representation. A higher group distance ratio or instance information gain implies less over-smoothing. We measure the group distance ratio and instance information gain of the trained reference model (ResGCN-28) and the corresponding ablated models without dilation (w/o dilation) or without residual connections (w/o connection). As shown in Table 2, ResGCN-28 has a significantly higher group distance ratio (1.73 versus 1.12) and instance information gain (0.46 versus 0.01) than the non-residual counterpart. A similar observation can be made with respect to dilation albeit the impact is less pronounced. This indicates that ResGCN-28 learns sharper features and suffers less from over-smoothing. However, the current over-smoothing metrics can not isolate the effects caused by the difficulty of optimization. We believe disentangling the effects of over-smoothing and vanishing gradient is a promising direction to better understand deep GCN architectures.

Qualitative Results. Fig. 7 shows qualitative results on area 5 of S3DIS [13]. As expected from the results in Table 1, our ResGCN-28 and DenseGCN-28 perform particularly well on difficult classes such as board, beam, bookcase and door. Rows 1-4 clearly show how ResGCN-28 and DenseGCN-28 are able to segment the board, beam, bookcase and door respectively, while PlainGCN-28 completely fails. Please refer to the supplementary material, which can be found on the Computer Society Digital Library at http://doi.
Finally, we compare our reference network (ResGCN-28), which incorporates the ideas put forward in the methodology, to several state-of-the-art baselines in Table 3. The results clearly show the effectiveness of deeper models with residual graph connections and dilated graph convolutions. ResGCN-28 outperforms DGCNN [6] by 3.9 percent (absolute) in mean IoU. DGCNN has the same fusion and MLP prediction blocks as ResGCN-28 but a shallower PlainGCN-like backbone block. Furthermore, we outperform all baselines in 9 out of 13 classes. We perform particularly well in the difficult object classes such as board, where we achieve 51.1 percent, and sofa, where we improve the state-of-the-art by about 10 percent mIoU.

This significant performance improvement on the difficult classes is probably due to the increased network capacity, which allows the network to learn subtle details necessary to distinguish between a board and a wall for example. The first row in Fig. 7 is a representative example for this occurrence. Our performance gains are solely due to our innovation in the network architecture, since we use the same hyper-parameters and even learning rate schedule as the baseline DGCNN [6] and only decrease the number of nearest neighbors from 20 to 16 and the batch size from 24.

Fig. 7. Qualitative Results on S3DIS Semantic Segmentation. We show here the effect of adding residual and dense graph connections to deep GCNs. PlainGCN-28, ResGCN-28, and DenseGCN-28 are identical except for the presence of residual graph connections in ResGCN-28 and dense graph connections in DenseGCN-28. We note how both residual and dense graph connections have a substantial effect on hard classes like board, bookcase, and sofa. These are lost in the results of PlainGCN-28.

### TABLE 3
Comparison of ResGCN-28 With the State-of-the-Art on S3DIS Semantic Segmentation

| Method            | OA  | mIoU | ceiling | floor | wall | beam | column | window | door | table | chair | sofa | bookcase | board | clutter |
|-------------------|-----|------|---------|-------|------|------|--------|--------|------|-------|-------|------|----------|-------|---------|
| PointNet [33]     | 78.5| 47.6 | 88.0    | 88.7  | 69.3 | 42.4 | 23.1   | 47.5   | 51.6 | 54.1  | 42.0  | 9.6  | 38.2     | 29.4  | 35.2    |
| MS+CU [35]        | 79.2| 47.8 | 88.6    | 95.8  | 67.3 | 36.9 | 24.9   | 48.6   | 52.3 | 51.9  | 45.1  | 10.6 | 36.8     | 24.7  | 37.5    |
| G+RCU [35]        | 81.1| 49.7 | 90.3    | 92.1  | 67.9 | 44.7 | 24.2   | 52.3   | 51.2 | 58.1  | 47.4  | 6.9  | 39.0     | 30.0  | 41.9    |
| PointNet++ [34]   | -   | 53.2 | 90.2    | 91.7  | 73.1 | 42.7 | 21.2   | 49.7   | 42.3 | 62.7  | 59.0  | 19.6 | 45.8     | 48.2  | 45.6    |
| 3DRNN+CF [37]     | 86.9| 56.3 | 92.9    | 93.8  | 73.1 | 42.5 | 25.9   | 47.6   | 59.2 | 60.4  | 66.7  | 24.8 | 57.0     | 36.7  | 51.6    |
| ResGCN-28 (Ours)  | 85.9| 60.0 | 93.1    | 95.3  | 78.2 | 33.9 | 37.4   | 56.1   | 68.2 | 64.9  | 61.0  | 34.6 | 51.5     | 51.1  | 54.4    |
| Deep LPN [57]     | 85.7| 60.0 | 91.0    | 95.6  | 76.1 | 50.3 | 25.9   | 55.1   | 56.8 | 66.3  | 74.3  | 25.8 | 54.0     | 52.3  | 55.3    |
| ShellNet [58]     | 87.1| 66.8 | 90.2    | 93.6  | 79.9 | 60.4 | 44.1   | 64.9   | 52.9 | 71.6  | 84.7  | 53.8 | 64.6     | 48.6  | 59.4    |
| RandLA-Net [54]   | 88.0| 70.0 | 93.1    | 96.1  | 80.6 | 62.4 | 48.0   | 64.4   | 69.4 | 69.4  | 76.4  | 60.0 | 64.2     | 65.9  | 60.1    |
| KPConv [55]       | -   | 70.6 | 93.6    | 92.4  | 83.1 | 63.9 | 54.3   | 66.1   | 76.6 | 64.0  | 57.8  | 74.9 | 69.3     | 61.3  | 60.3    |

We report average per-class results across all areas for our reference model ResGCN-28 and state-of-the-art baselines. ResGCN-28 which has 28 GCN layers, residual graph connections, and dilated graph convolutions outperforms the previous state-of-the-art by almost 4 percent. It also outperforms all baselines in 9 out of 13 classes. The metrics shown are overall point accuracy (OA) and mean IoU (mIoU). '-' denotes not reported and **bold** denotes best performance. The recent related works [57], [58], [54], [55] are not in gray.
PlainGCN-28 and ResGCN-28 are identical except for the presence of residual connections in ResGCN-28. Many important parts of the objects are classified incorrectly using PlainGCN-28.

Comparison to State-of-the-Art. We summarize the results of our ResGCN-28 and compare it to PlainGCN-28 and other state-of-the-art methods in Table 4. ResGCN-28 outperforms PlainGCN-28 by a large margin, which illustrates how our proposed residual connections and stochastic dilated convolutions enable training deep GCN architectures. ResGCN-28 also substantially outperforms PointNet [33], PointNet++ [34], and SpiderCNN [59]. Note that PointCNN designs a point convolution specialized for point clouds, whereas ResGCN-28 is a general framework for any graph (e.g., point clouds, biological networks, citation networks, etc.). Despite this huge disadvantage, ResGCN-28 still achieves results on par with PointCNN [60] and even outperforms PointCNN on some categories including bottle, knife, lamp, and table. We also include the most recent SOTA results from Deep LPN [57] and PosPool [61] for reference. These methods employ different data pre-processing and augmentation pipelines and were published after the short version of our paper. We color these results in gray.

4.5.2 Part Segmentation on PartNet
We further experiment with our architecture on the task of part segmentation and evaluate it on the recently proposed large-scale PartNet [14] dataset. PartNet consists of over 26,671 3D models from 24 object categories with 573,585 annotated part instances. The dataset establishes three benchmarking tasks for part segmentation on 3D objects: fine-grained semantic segmentation, hierarchical semantic segmentation, and instance segmentation. For the following experiments, we focus on the fine-grained level of semantic segmentation, which includes 17 out of the 24 object categories present in the PartNet dataset.

We use the same reference architecture, ResGCN-28, that we used for the experiments on the S3IDS dataset. We compare its performance to the baseline architecture PlainGCN-28 to show the impact of our residual connections and stochastic dilated convolutions. We also compare against the state-of-the-art reported in the PartNet paper [14], namely PointNet [33], PointNet++ [34], SpiderCNN [59], and PointCNN [60]. As suggested in PartNet [14], we use 10,000 sampled points as input. We train a separate network for each category and optimize them with Adam for 500 epochs without weight decay. The initial learning rate 0.005 is decayed by a factor of 0.9 every 50 epochs. We report our results using the best pretrained models on the validation sets.

Qualitative Results. Fig. 8 shows qualitative results on 4 categories of PartNet [14]: bottle, bed, microwave, and refrigerator. As expected from the results in Table 4, ResGCN-28 performs very well compared to the baseline PlainGCN-28, where there are no residual connections between layers. Although ResGCN-28 produces some incorrect outputs compared to the ground truth in categories like microwave and bed, it still outperforms PlainGCN-28 and segments the important parts of the object. We provide more qualitative results in the supplementary material, available online.

4.5.3 Object Classification on ModelNet40
We also apply our architecture to the task of object classification and evaluate it on the popular object classification dataset ModelNet40 [62]. The dataset contains 12,311 CAD models from 40 different categories. Although saturated, ModelNet40 is considered an important benchmark for the task of 3D object classification. The results in Table 5 show the superiority of our ResGCN and DenseGCN architectures over the baseline PlainGCN and state-of-the-art methods. When compared to the baseline PlainGCN, one observes the positive effect of our residual connections, dense connections, and dilated convolutions. In addition, the ResGCN and DenseGCN architectures outperform all state-of-the-art approaches in overall test accuracy. Particularly, ResGCN-14 achieves 93.6 percent on ModelNet-40 in terms of overall accuracy. Here, we point out that the recent work of...
KPCConv [55] uses a much larger number of trainable parameters compared to our reference ResGCN-28, as shown in Table 5. Yet, even our shallower variant ResGCN-14 outperforms KPCConv, while using almost one order of magnitude less trainable parameters.

5 EXPERIMENTS ON BIOLOGICAL NETWORKS

In order to demonstrate the generality of our contributions and specifically our deep ResGCN architecture, we conduct further experiments on general graph data. We choose the popular task of node classification on biological graph data, which is quite different from point cloud data. In the following experiments, we mainly study the effects of skip connections, the number of GCN layers (i.e., depth), number of filters per layer (i.e., width) and different graph convolutional operators.

### Table 4

Comparison of Our GCN Variants With the State-of-the-Art on ModelNet40 Point Cloud Classification

| Model                  | O.A. (%) | C.A. (%) | Param (M) |
|------------------------|----------|----------|-----------|
| 3DmFV-Net [62]         | 91.6     | -        | 45.77     |
| PointNet [33]          | 89.2     | 86.2     | 3.5       |
| PointNet++ [34]        | 90.7     | -        | 1.48      |
| SO-Net [63]            | 90.9     | 87.3     | -         |
| PCNN by Ext [64]       | 92.3     | -        | 8.2       |
| SpecGCN [65]           | 91.5     | -        | 2.05      |
| SpiderCNN [59]         | 90.5     | -        | -         |
| MCConv [66]            | 90.9     | -        | -         |
| FlexConv [67]          | 90.2     | -        | 0.35      |
| PointCNN [60]          | 92.2     | 88.1     | 0.6       |
| DGCNN [6]              | 92.9     | 90.2     | 1.84      |
| KPCConv rigid [55]     | 92.9     | -        | 14.3      |
| KPCConv deform [55]    | 92.7     | -        | 15.2      |
| PlainGCN-14            | 90.5     | 85.1     | 2.2       |
| PlainGCN-28            | 91.4     | 86.8     | 3.3       |
| ResGCN-14 (Ours)       | 93.6     | 90.9     | 2.2       |
| ResGCN-28 (Ours)       | 93.3     | 89.5     | 3.3       |
| DenseGCN-14 (Ours)     | 93.3     | 89.4     | 8.8       |
| DenseGCN-28 (Ours)     | 93.2     | 90.3     | 30.9      |

We report the overall classification test accuracy (O.A.) and mean per-class accuracy (C.A.) of our models and state-of-the-art models in addition to the total number of parameters for each model. One can observe the superiority of our ResGCN and DenseGCN architectures over state-of-the-art.

5.1 Graph Learning on Biological Networks

The main difference between biological networks and point cloud data is that biological networks have inherent edge information and high dimensional input features. For the graph learning task on biological networks, we use the PPI [2] dataset to evaluate our architectures. PPI is a popular dataset for multi-label node classification, containing 24 graphs with 20 in the training set, 2 in the validation set, and 2 in the testing set. Each graph in PPI corresponds to a different human tissue, each node in a graph represents a protein and edges represent the interaction between proteins. Each node has positional gene sets, motif gene sets, and immunological signatures as input features (50 in total) and 121 gene ontology sets as labels. The input of the task is a graph that contains 2373 nodes on average, and the goal is to predict which labels are contained in each node.

We use essentially the same reference architecture as for point cloud segmentation described in Section 4.1, but we predict multiple labels. The number of filters of the first and last layers are changed to adapt to this task. Instead of constructing edges by means of $k$-NN, we use the edges provided by PPI directly. If we were to construct edges dynamically, there is a chance to lose the rich information provided by the initial edges.

5.2 Experimental Setup

Following common procedure, we use the micro-average F1 (m-F1) score as the evaluation metric. For each graph, the F1 score is computed as follows:

$$F1\text{-score} = 2 \times \frac{(\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})},$$

(10)

where $\text{precision} = \frac{TP}{P'}$, $\text{recall} = \frac{TP}{T'}$, $TP$ is the total number of true positive points of all the classes, $T'$ is the number of ground truth positive points, and $P'$ is the number of predicted positive points. We find the best model, i.e., the one with the highest accuracy (m-F1 score) on the validation set in the training phase, and then calculate the m-F1 score across all the graphs in the test set.

We show the performance and GPU memory usage of our proposed MRGCN and compare them with other graph convolutions, e.g., EdgeConv [6], GATConv [43], SemiGCN [38] and GINConv [52]. We conduct an extensive ablation study on the number of filters and the
number of GCN layers to show their effect in the backbone network. Our ablation study also includes experiments to show the importance of residual graph connections and dense graph connections in our DeepGCN framework. To ensure a fair comparison, all networks in our ablation study share the same architecture. Finally, we compare our best models to several state-of-the-art methods for this task.

### 5.3 Implementation

For this biological network node classification task, we implement all our models based on PyTorch Geometric [68]. We use the Adam optimizer with the same initial learning rate 0.0002 and learning rate schedule with learning rate decay of 80 percent every 2,000 gradient descent steps for all the experiments. The networks are trained with one NVIDIA Tesla V100 GPU with a batch size of 1. Dropout with a rate of 0.2 is used at the first and second MLP layers of the prediction block. For fair comparison, we do not use any data augmentation or post processing techniques. Our models are trained end-to-end from scratch.

### 5.4 Results

We study the effect of residual and dense graph connections on multi-label node classification performance. We also investigate the influence of different parameters, e.g., the number of filters (32, 64, 128, 256) and layers (3, 7, 14, 28, 56, 112). To show the generality of our framework, we also apply the proposed residual connections to multiple GCN variants.

**Effect of Graph Connections.** Results in Table 6 show that both residual and dense graph connections help train deeper networks. When the network is shallow, models with graph connections achieve similar performance as models without them. However, as the network grows deeper, the performance of models without graph connections drops dramatically, while the performance of models with graph connections is stable or even improves further. For example, when the number of filters is 32 and the depth is 112, the performance of ResMRGCN-112 is nearly 37.66 percent higher than PlainMRGCN-112 in terms of the m-F1 score. We note that DenseMRGCN enjoys a slight performance gain as the network depth increases from 3 to 14. For depths beyond 14 layers, the performance of models without graph connections decreases significantly. Clearly, using graph connections achieves better performance than ResMRGCN with the same network depth and width.

**Effect of Network Depth.** Results in Table 6 show that increasing the number of layers improves network performance if residual or dense graph connections are used. Although ResMRGCN has a slight performance drop when the number of layers reaches 112, the m-F1 score is still much higher than the corresponding PlainMRGCN. The performance of DenseMRGCN increases reliably as the network grows deeper; however, DenseMRGCN consumes more memory than ResMRGCN due to concatenations of feature maps. Due to this memory issue, we are unable to train some models and denote them with '-' in Table 6. Meanwhile, PlainMRGCN, which has no graph connections, only enjoys a slight performance gain as the network depth increases from 3 to 14. For depths beyond 14 layers, the performance drops significantly. Clearly, using graph connections improves performance, especially for deeper networks where it becomes essential.

**Effect of Network Width.** Results of each row in Table 6 show that increasing the number of filters can consistently increase performance. A higher number of filters can also help convergence for deeper networks. However, a large number of filters is very memory consuming. Hence, we only consider networks with up to 256 filters in our experiments.

**Effect of GCN Variants.** Table 7 shows the effect of using different GCN operators with different model depths. Residual graph connections and the GCN operators are the only difference when the number of layers is kept the same. The results clearly show that residual graph connections in
Fig. 9. Memory usage for different GCNs on PPI node classification. We keep the same parameters for different models and compare their m-F1 scores with their total memory usages. All models in the comparison have 256 filters per layer and 56 layers. We notice that our model ResMRGCN uses approximately 1/6 of the total memory used by ResEdgeConv and gives a better m-F1 score.

deep networks can help different GCN operators achieve better performance than PlainGCN. Interestingly, when the network grows deeper, the performance of PlainSemiGCN, PlainGAT, and PlainGIN decreases dramatically; meanwhile, PlainEdgeConv and PlainMRGCN only observe a relatively small performance drop. In comparison, our proposed MRGCN operator achieves the best performance among all models.

Memory Usage of GCN Variants. In Fig. 9, we compare the total memory usage and performance of different GCN operators. All these models share the same architecture except for the GCN operations. They all use residual graph connections with 56 layers and 256 filters. We implement all the models with PyTorch Geometric and train each using one NVIDIA Tesla V100. The GPU memory usage is measured when the memory usage is stable. Our proposed ResMRGCN achieves the best performance, while only using around 15 percent GPU memory compared to ResEdgeConv.

Comparison to State-of-the-Art. Finally, we compare our DenseMRGCN-14 and ResMRGCN-28 to several state-of-the-art baselines in Table 8. Results clearly show the effectiveness of deeper models with residual and dense graph connections. DenseMRGCN-14 and ResMRGCN-28 outperform the previous state-of-the-art Cluster-GCN [69] by 0.07 and 0.05 percent respectively. It is worth mentioning that a total of ten models in Table 6 surpass Cluster-GCN.

7 Future Work

Our results show that after solving the vanishing gradient problem plaguing deep GCNs, we can either make GCNs deeper or wider to get better performance. We expect GCNs to become a powerful tool for processing graph-structured data in computer vision, natural language processing, and data mining. We show successful cases for adapting concepts from CNNs to GCNs (i.e., skip connections and dilated convolutions). In the future, it will be worthwhile to explore how to transfer other operators (e.g., deformable convolutions [74]), other architectures (e.g., feature pyramid architectures [75]), etc. It will also be interesting to study different distance measures to compute dilated k-NN, constructing graphs with different k at each layer, better dilation rate schedules [73], [76] for GCNs, and combining residual and dense connections.

We also point out that, for the specific task of point cloud semantic segmentation, the common approach of processing the data in $1m \times 1m$ columns is sub-optimal for graph representation. A more suitable sampling approach should lead to further performance gains on this task. For the task of node classification, the existing datasets are relatively

| Model               | m-F1 score (%) |
|---------------------|----------------|
| Cluster-GCN         | 99.36          |
| GeniePath           | 98.50          |
| GaAN                | 98.71          |
| VR-GCN              | 97.80          |
| GraphSAGE[42]       | 97.30          |
| GATConv [43]        | 97.00          |
| VR-GCN [70]         | 98.71          |
| GeniePath [72]      | 98.50          |
| Cluster-GCN [69]    | 99.36          |
| ResMRGCN-28(Ours)   | 99.41          |
| DenseMRGCN-14(Ours) | 99.43          |

We follow convention and compare models based on the m-F1 score. Our model DenseMRGCN-14, which has 14 MRGCN layers, 256 filters in the first layer, and dense graph connections outperforms all baselines. Our ResMRGCN-28, which has 28 MRGCN layers, 256 filters per layer, and residual graph connections also outperforms previous state-of-the-art.

6 Conclusion

This work shows how proven concepts from CNNs (i.e., residual/dense connections and dilated convolutions) can be transferred to GCNs in order to make GCNs go as deep as CNNs. Adding skip connections and dilated convolutions to GCNs alleviates the training difficulty, which was impeding GCNs to go deeper and thus impeding further progress. A large number of experiments on semantic segmentation and part segmentation of 3D point clouds, as well as node classification on biological graphs show the benefit of deeper architectures, as they achieve state-of-the-art performance. We also show that our approach generalizes across several GCN operators.
small. We expect that experimenting on larger datasets will further unleash the full potential of DeepGCNs.

**Acknowledgments**

The authors would like to thank Adel Bibi and Hani Itani for their help with the project. This work was supported by the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research through the Visual Computing Center (VCC) funding. Guohao Li, Matthias Müller, and Guocheng Qian contributed equally to this work.

**References**

[1] L. Tang and H. Liu, “Relational learning via latent social dimensions,” in Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2009, pp. 817–826.

[2] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, “Graph R-CNN for scene graph generation,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 670–685.

[3] Y. Li, W. Ouyang, B. Zhou, J. Shi, C. Zhang, and X. Wang, “Factorizable net: An efficient subgraph-based framework for scene graph generation,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 335–351.

[4] J. Johnson, A. Gupta, and L. Fei-Fei , “Image generation from scene graphs,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 1219–1228.

[5] S. Yan, Y. Xiong, and D. Lin, “Spatial temporal graph convolutional networks for skeleton-based action recognition,” in Proc. 32nd AAAI Conf. Artif. Intell., 2018, pp. 7444–7452.

[6] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, “Structural-RNN: Deep learning on spatio-temporal graphs,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 5308–5317.

[7] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller , “Multi-view convolutional neural networks for 3D shape recognition,” in Proc. IEEE Int. Conf. Comput. Vis., 2015, pp. 945–953.

[8] J. Guerra, A. Boulch, B. Le Saux , J. Moras, A. Pleyer, and D. Filliat, “ViTPhRNN: Consistent multi-view semantic labeling for robotics,” in Proc. IEEE Int. Conf. Comput. Vis. Workshops, 2017, pp. 669–678.

[9] A. Boulch, B. Le Saux , and N. Audebert, “Unstructured point cloud semantic labeling using deep segmentation networks,” in Proc. Workshop 3D Object Retrieval, 2017, pp. 17–24.

[10] Z. Li, Y. Cao, X. Liang, Y. Yu, H. Cheng, and L. Lin, “LSTM-CGC: Unifying context modeling and fusion with LSTMs for RGB-D scene labeling,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 541–557.

[11] A. Dai, A. X. Chang, M. Savva, M. Palter, T. A. Funkhouser, and M. Nießner , “ScanNet: Richly-annotated 3D reconstructions of indoor scenes,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, Art. no. 10.

[12] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, “Volumetric and multi-view CNNs for object classification on 3D data,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 5648–5656.

[13] G. Riegler, A. O. Ulusoy, and A. Geiger, “OctNet: Learning deep 3D representations at high resolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 6620–6629.

[14] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese, “SEGCNet: Semantic segmentation of 3D point clouds,” in Proc. Int. Conf. 3D Vis., 2017, pp. 537–547.

[15] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas , “PointNet: Deep learning on point sets for 3D classification and segmentation,” in Proc. Comput. Vis. Pattern Recognit., 2017, pp. 77–85.

[16] R. Q. Charles, L. Yi, H. Su, and L. J. Guibas , “PointNet++: Deep hierarchical feature learning on point sets in a metric space,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2017, pp. 5099–5108.

[17] F. Engelmann, T. Kontogianni, A. Hermans, and B. Leibe, “Exploring spatial context for 3D semantic segmentation of point clouds,” in Proc. IEEE Int. Conf. Comput. Vis., 2017, pp. 716–724.

[18] Q. Huang, W. Wang, and U. Neumann, “Recurrent slice networks for 3D segmentation of point clouds,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 2626–2635.

[19] X. Ye, J. Li, H. Huang, L. Du, and X. Zhang, “3D recurrent neural networks with context fusion for point cloud semantic segmentation,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 415–430.

[20] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. 5th Int. Conf. Learn. Representations, 2017.

[21] T. Pham, T. Tran, D. Phung, and S. Venkatesh, “Column networks for collective classification,” in Proc. 31st AAAI Conf. Artif. Intell., 2017, pp. 2485–2491.

[22] A. Rahimi, T. Cohn, and T. Baldwin, “Semi-supervised user geolocation via graph convolutional networks,” in Proc. 56th Ann. Meeting Assoc. Comput. Linguistics, 2018, vol. 1, pp. 2009–2019, doi: 10.18653/v1/P18-1187.

[23] H. Fu, C. Li, Y. Tian, T. Sonobe, K.-I. Kawarabayashi, and S. Jegelka, “Representation learning on graphs with jumping knowledge networks,” in Proc. 35th Int. Conf. Mach. Learn., 2018, vol. 80, pp. 5453–5462.

[24] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2017, pp. 1024–1034.
[43] P. Velicković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in Proc. Int. Conf. Learn. Representations, 2018.

[44] N. Peng, H. Poon, C. Quirk, K. Toutanova, and W.-T. Yih, “Cross-sentence N-ary relation extraction with graph LSTMs,” Trans. Assoc. Comput. Linguistics, vol. 5, pp. 101–115, 2017.

[45] D. K. Duvenaud et al. “Convolutional networks on graphs for learning molecular networks,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2015, pp. 2224–2232.

[46] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, “Gated graph sequence neural networks,” in Proc. Int. Conf. Learn. Representations, 2016.

[47] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in Proc. 32nd Int. Conf. Mach. Learn., 2015, vol. 37, pp. 448–456.

[48] M. Simonovsky and N. Komodakis, “Dynamic edge-conditioned filters in convolutional neural networks on graphs,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 3693–3702.

[49] D. Valsesia, G. Fracastoro, and E. Magli, “Learning localized generative models for 3D point clouds via graph convolution,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2018, pp. 6873–6882.

[50] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, “A real-time algorithm for signal analysis with the help of the wavelet transform,” in Wavelets. Berlin, Germany: Springer, 1990, pp. 286–297.

[51] M. J. Shensa, “The discrete wavelet transform: Wedding the a trous and Mallat algorithms,” IEEE Trans. Signal Process., vol. 40, no. 10, pp. 2464–2482, Oct. 1992.

[52] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in Proc. Int. Conf. Learn. Representations, 2019.

[53] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional networks for biomedical image segmentation,” in Proc. Int. Conf. Med. Image Comput. Assist. Intervention, 2015, pp. 234–241.

[54] Q. Hu et al. “RandLA-Net: Efficient semantic segmentation of large-scale point clouds,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 11 105–1114.

[55] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, “K-PointConv: Flexible and deformable convolution for point clouds,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2019, pp. 6410–6419.

[56] Z. Zhang, B.-S. Hua, and S.-K. Yeung, “ShellNet: Efficient point convolutional neural networks,” in Proc. Adv. Neural Inform. Process. Syst., vol. 33, pp. 4917–4928, 2020.

[57] E.-T. Le, I. Kokkinos, and N. Mitra, “Going deeper with lean point networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 9500–9509.

[58] Z. Zhang, B.-S. Hua, and S.-K. Yeung, “ShellNet: Efficient point cloud convolutional neural networks using concentric shells statistics,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2019, pp. 1607–1616.

[59] 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.

[60] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, “PointCNN: Convolution on X-treme Points,” in Proc. 32nd Int. Conf. Neural Inf. Process. Syst., 2018, pp. 828–838.

[61] Z. Liu, H. Hu, Y. Cao, Z. Zhang, and X. Tong, “A closer look at local aggregation operators in point cloud analysis,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 326–342.

[62] Y. Ben-Shabat, M. Lindenbaum, and A. Fischer, “3DmFV: Three-dimensional point cloud classification in real-time using convolutional neural networks,” IEEE Robot. Autom. Lett., vol. 3, no. 4, pp. 3145–3152, Oct. 2018.

[63] 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., 2018, pp. 9397–9406.

[64] M. Atzmon, H. Marion, and Y. Lipman, “Point convolutional neural networks by extension operators,” ACM Trans. Graph., vol. 37, no. 12, pp. 1–12, 2018.

[65] C. Wang, B. Samari, and K. Siddiqi, “Local spectral graph convolution for point set feature learning,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 56–71.
Itzel C. Delgadillo received the bachelor’s degree in artificial intelligence from the Panamenican University, Mexico, in 2019. She is currently a visiting student research intern at the King Abdullah University of Science and Technology. Her research interests include computer vision and neural architecture search.

Abdulellah Abualshour received the BS degree in computer science from Rutgers, The State University of New Jersey, New Brunswick, New Jersey, in 2018. He is currently working toward the MS degree and a member of the Image and Video Understanding Lab (IVUL), Visual Computing Center (VCC), King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia. He was a recipient of the KAUST Gifted Student Program (KGSP) scholarship award. His research interests include computer vision and deep learning.

Ali Thabet is currently a research scientist at the Visual Computing Center (VCC), King Abdullah University of Science and Technology (KAUST), working in the Image and Video Understanding Laboratory (IVUL). His research interests include problems related to 3D computer vision. In general, he’s interested in algorithmic applications of machine learning, and deep learning specifically, to understand the 3D world with the help of sensors like RGB-D cameras, LiDAR, and others. Also, he is interested in applying deep learning to image reconstruction, 3D object detection and generation, and autonomous vehicles.

Bernard Ghanem is currently an associate professor in the CEMSE division, a theme leader at the Visual Computing Center (VCC), and the Interim Lead of the AI Initiative at King Abdullah University of Science and Technology (KAUST). His research interests include computer vision and machine learning with emphasis on topics in video understanding, 3D recognition, and theoretical foundations of deep learning. He received the bachelor’s degree from the American University of Beirut (AUB) in 2005 and his MS/PhD from the University of Illinois at Urbana-Champaign (UIUC) in 2010. His work has received several awards and honors, including four best paper awards for workshops in CVPR 2013 and 2019 and ECCV 2018 and 2020, a Google Faculty Research Award in 2015 (first in MENA for Machine Perception), and an Abdul Hameed Shoman Arab Researchers Award for Big Data and Machine Learning in 2020. He has coauthored more than 120 peer reviewed conference and journal papers in his field as well as three issued patents. He serves as an Associate Editor for the IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) and has served as an Area Chair (AC) for CVPR 2018/2021, ICCV 2019/2021, ICLR 2021, and AAAI 2021.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.