DeepGCNs: Making GCNs Go as Deep as CNNs

https://www.deepgcns.org

Guohao Li* Matthias Müller* Guocheng Qian Itzel C. Delgadillo Abdulellah Abualshour
Ali Thabet Bernard Ghanem
Visual Computing Center, KAUST, Thuwal, Saudi Arabia
{guohao.li, matthias.mueller.2, guocheng.qian, itzel.delgadilloperez, abdulellah.abualshour, ali.thabet, bernard.ghanem}@kaust.edu.sa

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 CNNs. 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-Eucledian data as input to a neural network similar to CNNs. While GCNs already achieve encouraging results, they are currently limited to shallow architectures with $2-4$ layers 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 deep GCNs with as many as 112 layers experimentally across various datasets and tasks. Specifically, we achieve state-of-the-art 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 a lot of avenues for future research on GCNs and transfer to further tasks not explored in this work. The source code for this work is available for Pytorch and Tensorflow at https://github.com/lightaime/deep_gcns_torch and https://github.com/lightaime/deep_gcns respectively.

Index Terms—Graph Convolution 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, it turns out that 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 train very deep network architectures reliably. 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 as for CNNs, stacking multiple layers in GCNs leads to the vanishing gradient problem. Over-smoothing problem occurs when repeatedly applying GCN many layers [7]. They observed the features of vertices within each connected component will converge to the same value and become indistinguishable. As a result most state-of-the-art GCNs are limited to very shallow network architecture, usually no deeper than 4 layers [9].

The vanishing gradient problem is well known and 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 and 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 that could be trained reliably, *e.g.* ResNet-152. 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 are 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?

In this work, we present an extensive study of methodologies that allow training very deep GCNs. We adapt concepts that were successful in training deep CNNs, in particular residual connections, dense connections, and dilated convolutions. We show how

**equal contribution*

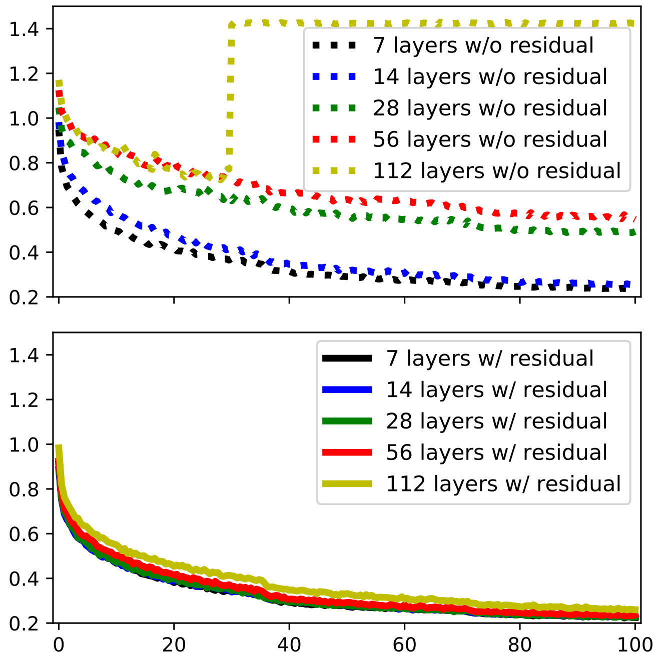

Fig. 1. **Training Deep GCNs**. (*left*) We show the square root of the training loss for GCNs with 7, 14, 28, 56 and 112 layers, with and without residual connections for 100 epochs. We note that adding more layers without residual connections translates to a substantially higher loss and for very deep networks *e.g.* 112 layers even to divergence. (*right*) In contrast, training GCNs with residual connections results in consistent training stability across all depths.

these concepts can be incorporated into a graph framework. In order to quantify the effect of these additions, we conduct an extensive analysis of each component and its impact on accuracy and stability of deep GCNs. To showcase the potential of these concepts in the context of GCNs, we apply them to the popular tasks of semantic segmentation and part segmentation of point clouds as well as node classification of biological graphs. Adding either residual or dense connections in combination with dilated convolutions, enables successful training of GCNs with a depth of 112 layers (refer to Figure 1). The proposed deep GCNs improve the state-of-the-art on the challenging point cloud dataset S3DIS [13] by 3.9% mIOU and outperform previous methods in many classes of PartNet [14]. The same deep GCN architecture also achieves an F1 score of 99.43 on the very different PPI dataset [2].

Contributions. The contributions of this work comprise 3 aspects. **(1)** We adapt residual connections, dense connections, and dilated convolutions which were introduced for CNNs to GCNs. **(2)** We present extensive experiments on point cloud data 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% 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 set a new state-of-art for the PPI dataset [2] in the task of node classification in biological networks.

A preliminary version of this work was published in ICCV 2019 [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 when training a *PlainGCN* with 112 layers it diverges while our proposed counterpart, *ResGCN* with skip connections and dilated convolutions, converges without problem (see Figure 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 current state-of-the-art methods. We also demonstrate that *MRGCN* is very memory-efficient compared to other GCN operators via GPU memory usage experiments. *Finally*, to ensure 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 method is the first approach that successfully trains deep GCNs beyond 100 layers and achieves state-of-the-art results on both point cloud data 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]. 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 and showed peak performance for 10 layers and degrading performance for deeper graphs. Rahimi *et al.* [40] developed a Highway GCN for user geo-location in social media graphs, where they add "highway" gates between layers to facilitate gradient flow. Even with these gates, the authors demonstrate performance degradation after 6 layers of depth. Xu *et al.* [41] developed a *Jump Knowledge Network* for representation learning and devised an alternative strategy to select graph neighbors for each node based on graph structure. As with other works, their network is limited to a small number of layers (6). Recently, Li *et al.* [7] have studied the depth limitations of GCNs and showed that deep GCNs can cause over-smoothing, which results in features at vertices within each connected component converging to the same value. Other works [8], [9] also show the limitations of stacking multiple GCN layers, which leads to highly complex back-propagation and the common vanishing gradient problem.

Many difficulties facing GCNs nowadays (*e.g.* vanishing gradients and limited receptive field) were also present in the early days of CNNs [10], [12]. We bridge this gap and show that the majority of these drawbacks can be remedied by borrowing several orthogonal tricks from CNNs. Deep CNNs achieved a huge boost in performance with the introduction of ResNet [10]. By adding residual connections between inputs and outputs of layers, ResNet tends to alleviate the vanishing gradient problem. DenseNet [11] takes this idea a step further and adds connections across layers as well. Dilated Convolutions [12] are another recent approach that has lead to significant performance gains, specifically in image-to-image translation tasks such as semantic segmentation [12], by increasing the receptive field without loss of resolution. In this work, we show how one can benefit from concepts introduced for CNNs, mainly residual/dense connections and dilated convolutions, to train very deep GCNs. We support our claim by extending different GCN variants to deeper versions by adapting these concepts, and therefore significantly increasing their performance. Extensive experiments on the tasks of semantic segmentation and part segmentation of point clouds and node classification in biological graphs validate these ideas for general graph scenarios.

3 Methodology

3.1 Representation Learning on Graphs

Graph Definition. A graph $\mathcal{G}$ is represented by a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ 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 $\mathbf{h}_v \in \mathbb{R}^D$, where D is the feature dimension. Therefore, the graph $\mathcal{G}$ as a whole can be represented by concatenating the features of all the unordered vertices, *i.e.* $\mathbf{h}_{\mathcal{G}} = [\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, ..., \mathbf{h}_{v_N}]^\top \in \mathbb{R}^{N \times D}$, where N is the cardinality of set $\mathcal{V}$. A general graph convolution operation $\mathcal{F}$ at the l-th layer can be formulated as the following aggregation and update operations,

$$\begin{aligned} \mathcal{G}_{l+1} &= \mathcal{F}(\mathcal{G}_l, \mathcal{W}_l) \\ &= Update(Aggregate(\mathcal{G}_l, \mathcal{W}_l^{agg}), \mathcal{W}_l^{update}). \end{aligned} \tag{1}$$

$\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$ and $\mathcal{G}_{l+1} = (\mathcal{V}_{l+1}, \mathcal{E}_{l+1})$ are the input and output graphs at the l-th layer, respectively. $\mathcal{W}_l^{agg}$ and $\mathcal{W}_l^{update}$ 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 those 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,

$$\mathbf{h}_{v_{l+1}} = \phi\left(\mathbf{h}_{v_l}, \rho(\{\mathbf{h}_{u_l} | u_l \in \mathcal{N}(v_l)\}, \mathbf{h}_{v_l}, \mathcal{W}_\rho), \mathcal{W}_\phi\right), \tag{2}$$

where ρ is a vertex feature aggregation function and ϕ is a vertex feature update function, $\mathbf{h}_{v_l}$ and $\mathbf{h}_{v_{l+1}}$ 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 at the l-th layer, and $\mathbf{h}_{u_l}$ is the feature of those neighbor vertices parametrized by $\mathcal{W}_\rho$. $\mathcal{W}_\phi$ 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: $\rho(.) = \max(\mathbf{h}_{u_l} - \mathbf{h}_{v_l} |\ u_l \in \mathcal{N}(v_l))$. We then model the vertex feature updater ϕ as a multi-layer perceptron (MLP) with batch normalization [47] and a ReLU as an activation function. This MLP concatenates $\mathbf{h}_{v_l}$ with its aggregate features from $\rho(.)$ 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, can learn 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 neighbourhood of each vertex in every layer. We find that dynamically changing neighbors in GCNs helps alleviate the over-smoothing problem and results in an effectively larger receptive

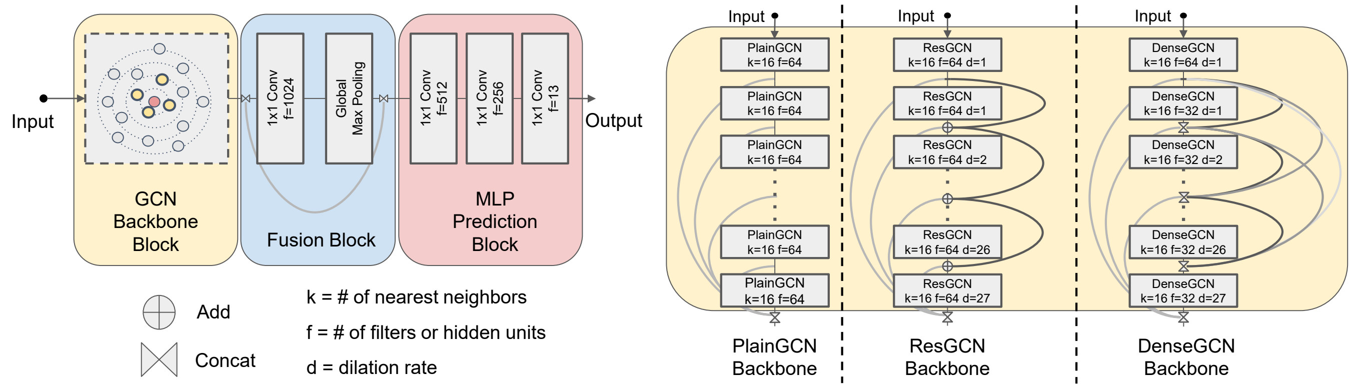

Fig. 2. **Proposed GCN architecture for point cloud semantic segmentation**. *(left)* Our framework consists of three blocks: a GCN Backbone Block (feature transformation of input point cloud), a Fusion Block (global feature generation and fusion), and an MLP Prediction Block (point-wise label prediction). *(right)* We study three types of GCN Backbone Block (*PlainGCN*, *ResGCN* and *DenseGCN*) and use two kinds of layer connection (vertex-wise addition used in *ResGCN* or vertex-wise concatenation used in *DenseGCN*).

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 no more than three layers deep [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 to be trained: 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 $\mathcal{G}_{l+1}^{res}$ 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*.

$$\begin{aligned}\mathcal{G}_{l+1} &= \mathcal{H}(\mathcal{G}_l, \mathcal{W}_l) \\ &= \mathcal{F}(\mathcal{G}_l, \mathcal{W}_l) + \mathcal{G}_l = \mathcal{G}_{l+1}^{res} + \mathcal{G}_l.\end{aligned} \tag{3}$$

3.3 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:

$$\begin{aligned}\mathcal{G}_{l+1} &= \mathcal{H}(\mathcal{G}_l, \mathcal{W}_l) \\ &= \mathcal{T}(\mathcal{F}(\mathcal{G}_l, \mathcal{W}_l), \mathcal{G}_l) \\ &= \mathcal{T}(\mathcal{F}(\mathcal{G}_l, \mathcal{W}_l), ..., \mathcal{F}(\mathcal{G}_0, \mathcal{W}_0), \mathcal{G}_0).\end{aligned} \tag{4}$$

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 $D_0 + D \times (l + 1)$.

3.4 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 deep GCNs.

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 $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with *Dilated k-NN* and d as the dilation rate, the *Dilated k-NN* 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 $(u_1, u_2, ..., u_{k\times d})$ are the first sorted $k \times d$ nearest neighbors, vertices $(u_1, u_{1+d}, u_{1+2d}, ..., u_{1+(k-1)d})$ are the d-dilated neighbors of vertex v (see Figure 3), *i.e.*

$$\mathcal{N}^{(d)}(v) = \{u_1, u_{1+d}, u_{1+2d}, ..., u_{1+(k-1)d}\}.$$

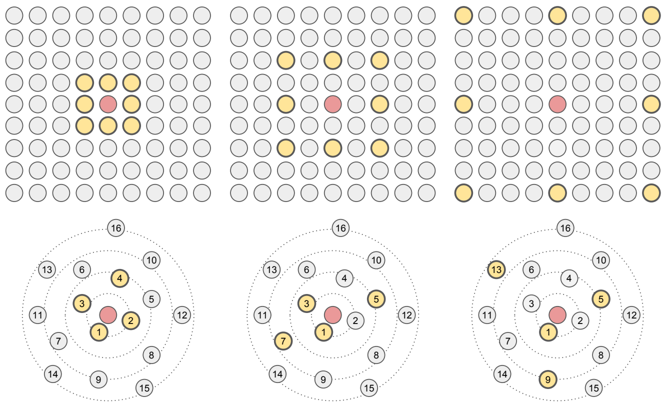

Fig. 3. **Dilated Convolution in GCNs**. Visualization of dilated convolution on a structured graph arranged in a grid (*e.g.* 2D image) and on a general structured graph. (*top*) 2D convolution with kernel size 3 and dilation rate 1, 2, 4 (left to right). (*bottom*) Dynamic graph convolution with dilation rate 1, 2, 4 (left to right).

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 $\mathbf{h}_v^{(d)}$ of each output vertex in $\mathcal{V}^{(d)}$. We denote this layer operation as a *dilated graph convolution* with dilation rate d, or more formally: $\mathcal{G}^{(d)} = (\mathcal{V}^{(d)}, \mathcal{E}^{(d)})$. We visualize and compare it to a conventional dilated convolution used in CNNs in Figure 3. To improve generalization, we use *stochastic dilation* in practice. During training, we perform the aforementioned dilated aggregations with a high probability $(1-\epsilon)$ leaving a small probability ϵ to perform random aggregation by uniformly sampling k neighbors from the set of $k \times d$ neighbors $\{u_1, u_2, ..., u_{k\times d}\}$. At inference time, we perform deterministic dilated aggregation without stochasticity.

3.5 Deep GCN Variants

In our experiments in the paper, we mostly work with a GCN based on EdgeConv [6] to show how very deep GCNs can be trained. However, it is straightforward to build other deep GCNs with the same concepts we propose (*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:

$$\begin{aligned}
\mathbf{h}_{v_{l+1}}^{res} &= \mathit{max}\left(\{\mathit{mlp}(\mathit{concat}(\mathbf{h}_{v_l}, \mathbf{h}_{u_l} - \mathbf{h}_{v_l}))|u_l \in \mathcal{N}^{(d)}(v_l)\}\right), \\
\mathbf{h}_{v_{l+1}} &= \mathbf{h}_{v_{l+1}}^{res} + \mathbf{h}_{v_l}.
\end{aligned} \tag{5}$$

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

$$\begin{aligned}
\mathbf{h}_{\mathcal{N}^{(d)}(v_l)}^{res} &= \mathit{max}\left(\{\mathit{mlp}(\mathbf{h}_{u_l})|u_l \in \mathcal{N}^{(d)}(v_l)\}\right), \\
\mathbf{h}_{v_{l+1}}^{res} &= \mathit{mlp}\left(\mathit{concat}\Big(\mathbf{h}_{v_l}, \mathbf{h}_{\mathcal{N}^{(d)}(v_l)}^{res}\Big)\right), \\
\mathbf{h}_{v_{l+1}} &= \mathbf{h}_{v_{l+1}}^{res} + \mathbf{h}_{v_l},
\end{aligned} \tag{6}$$

In the original GraphSAGE paper, the vertex features are normalized after aggregation. We implement two variants, one without normalization (see Equation (6)), the other one with normalization $\mathbf{h}_{v_{l+1}}^{res} = \mathbf{h}_{v_{l+1}}^{res} / \left\|\mathbf{h}_{v_{l+1}}^{res}\right\|_2$.

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

$$\begin{aligned}
\mathbf{h}_{v_{l+1}}^{res} &= \mathit{mlp}\left((1+\epsilon)\cdot \mathbf{h}_{v_l} + \mathit{sum}(\{\mathbf{h}_{u_l}|u_l \in \mathcal{N}^{(d)}(v_l)\})\right), \\
\mathbf{h}_{v_{l+1}} &= \mathbf{h}_{v_{l+1}}^{res} + \mathbf{h}_{v_l}.
\end{aligned} \tag{7}$$

MRGCN. We find that first using a max aggregator to aggregate neighborhood relative features $(\mathbf{h}_{u_l} - \mathbf{h}_{v_l})$, $u_l \in \mathcal{N}(v_l)$ is more efficient than aggregating raw neighborhood features $\mathbf{h}_{v_l}$, $u_l \in \mathcal{N}(v_l)$ 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:

$$\begin{aligned}
\mathbf{h}_{\mathcal{N}^{(d)}(v_l)}^{res} &= \mathit{max}\left(\{\mathbf{h}_{u_l} - \mathbf{h}_{v_l}|u_l \in \mathcal{N}^{(d)}(v_l)\}\right), \\
\mathbf{h}_{v_{l+1}}^{res} &= \mathit{mlp}\left(\mathit{concat}\Big(\mathbf{h}_{v_l}, \mathbf{h}_{\mathcal{N}^{(d)}(v_l)}^{res}\Big)\right), \\
\mathbf{h}_{v_{l+1}} &= \mathbf{h}_{v_{l+1}}^{res} + \mathbf{h}_{v_l}.
\end{aligned} \tag{8}$$

Here $\mathbf{h}_{v_{l+1}}$ and $\mathbf{h}_{v_l}$ are the hidden state of vertex v at $l+1$ and $\mathbf{h}_{v_{l+1}}^{res}$ is the hidden state of the residual graph. All the *mlp* (multilayer perceptron) functions use a ReLU as activation function; all the *max* and *sum* functions above are vertex-wise feature operators; *concat* functions concatenate features of two vertices into one feature vector. $\mathcal{N}^{(d)}(v_l)$ denotes the neighborhood of vertex v_l obtained from *Dilated k-NN*.

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 point cloud datasets and demonstrate that our methods significantly improve performance. In addition, we also 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 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 executing a dilated k-NN search to find the nearest neighbor 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 $\frac{TP}{TP+T-P}$, 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 deep GCNs, we do a thorough ablation study on area 5 to analyze each component and provide insights. We then evaluate our proposed reference model (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 Figure 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 1×1 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 1×1 convolutions.

PlainGCN. This baseline model consists of a *PlainGCN* backbone block, a fusion block, and a 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*. These 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 3.3, 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% every 3×10^5 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 3.4, 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 deep GCN 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 10^{th} 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.

Ablation	Model	Operator	mIoU	ΔmIoU	dynamic	connection	dilation	stochastic	# NNs	# filters	# layers
Reference	*ResGCN-28*	EdgeConv	**52.49**	0.00	✓	$\oplus$	✓	✓	16	64	28
Dilation		EdgeConv	51.98	-0.51	✓	$\oplus$	✓		16	64	28
		EdgeConv	49.64	-2.85	✓	$\oplus$			16	64	28
	PlainGCN-28	EdgeConv	**40.31**	-12.18	✓				16	64	28
Fixed k-NN		EdgeConv	48.38	-4.11		$\oplus$			16	64	28
		EdgeConv	43.43	-9.06					16	64	28
Connections	*DenseGCN-28*	EdgeConv	**51.27**	-1.22	✓	$\bowtie$	✓	✓	8	32	28
		EdgeConv	40.47	-12.02	✓		✓	✓	16	64	28
		EdgeConv	38.79	-13.70	✓		✓	✓	8	64	56
		EdgeConv	49.23	-3.26	✓		✓	✓	16	64	14
		EdgeConv	47.92	-4.57	✓		✓	✓	16	64	7
Neighbors		EdgeConv	49.98	-2.51	✓	$\oplus$	✓	✓	8	64	28
		EdgeConv	49.22	-3.27	✓	$\oplus$	✓	✓	4	64	28
		EdgeConv	49.18	-3.31	✓	$\oplus$	✓	✓	32	32	28
Depth	*ResGCN-112*	EdgeConv	51.97	-0.52	✓	$\oplus$	✓	✓	4	64	112
	ResGCN-56	EdgeConv	**53.64**	1.15	✓	$\oplus$	✓	✓	8	64	56
	ResGCN-14	EdgeConv	49.90	-2.59	✓	$\oplus$	✓	✓	16	64	14
	ResGCN-7	EdgeConv	48.95	-3.53	✓	$\oplus$	✓	✓	16	64	7
Width	*ResGCN-28W*	EdgeConv	**53.78**	1.29	✓	$\oplus$	✓	✓	8	128	28
		EdgeConv	48.80	-3.69	✓	$\oplus$	✓	✓	16	32	28
		EdgeConv	45.62	-6.87	✓	$\oplus$	✓	✓	16	16	28
GCN Variants		GraphSAGE	49.20	-3.29	✓	$\oplus$	✓	✓	16	64	28
		GraphSAGE-N	49.02	-3.47	✓	$\oplus$	✓	✓	16	64	28
		GIN-ϵ	42.81	-9.68	✓	$\oplus$	✓	✓	16	64	28
	ResMRGCN-28	MRGCN	**51.17**	-1.32	✓	$\oplus$	✓	✓	16	64	28

TABLE 1
Ablation study on area 5 of S3DIS. 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 $\bowtie$ symbols respectively. We highlight the most important results in bold. ΔmIoU denotes the difference in mIoU with respect to the reference model *ResGCN-28*.

We begin with the extensive ablation study where we evaluate the trained models on area 5, after training them on the remaining ones. 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, 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 Figure 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]. When the residual graph connections between layers are removed (*i.e.* in *PlainGCN-28*), performance dramatically degrades (-12% mIoU). Figure 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 on in this section.

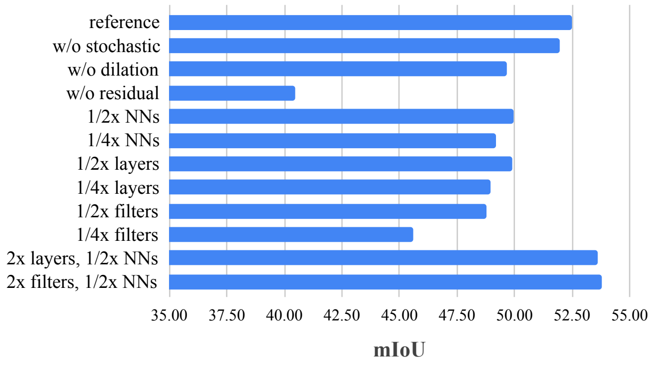

Fig. 4. **Ablation study on area 5 of S3DIS**. We compare our reference network (*ResGCN-28*) with 28 layers, *residual graph connections* and *dilated graph convolutions* to several ablated variants. All models were trained for 100 epochs on all areas except for area 5 with the same hyper-parameters.

Effect of dilation. Results in Table 1 (*Dilation*) [12] show that dilated graph convolutions account for a 2.85% improvement in mean IoU (*row 3*), motivated 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. Interestingly, our results in Table 1 also indicate that dilation especially helps deep networks when combined with residual graph connections (*rows 1,8*). Without such connections, performance can actually degrade with dilated graph convolutions. The reason for this is probably that these varying neighbors result in 'worse'

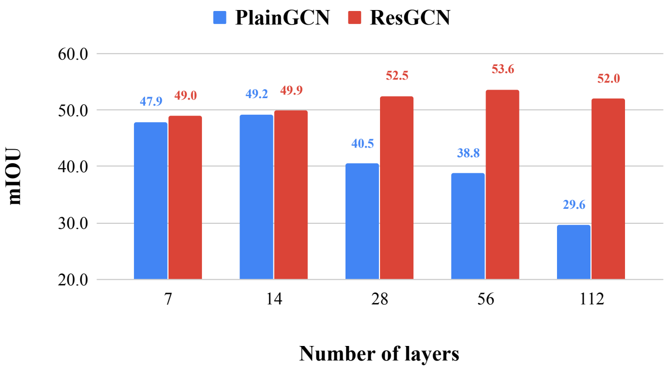

Fig. 5. **PlainGCN vs. ResGCN on area 5 of S3DIS**. We compare networks of different depths with and without *residual graph connections*. All models were trained for 100 epochs on all areas except for area 5 with the same hyper-parameters. Results only improve as depth increases with *residual graph connections*.

gradients, which further hinder convergence when residual graph connections are not used.

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 2.5% and 3.3% 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*).

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 the *PlainGCN*. Using our proposed *MRGCN* operator achieves almost the same performance as the *ResGCN* reference model which relies on *EdgeConv* while only using half of 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 they find setting ϵ to 0 leads to the best performance.

Qualitative Results. Figure 6 shows qualitative results on S3DIS [13], area 5. 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** for more qualitative results and further results.

Comparison to state-of-the-art. 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 2. 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% (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%, and sofa, where we improve state-of-the-art by about 10% 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 Figure 6 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 to 16 due to memory constraints. We outperform state-of-the art methods by a significant margin and expect further improvement from tweaking the hyper-parameters, especially the learning schedule.

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.

As is common practice, we use 10,000 sampled points as input to our network architecture. We use the same reference architecture, *ResGCN-28*, as for the S3IDS dataset. We compare it to several state-of-the-art baseline architectures with default training hyper-parameters as reported in the original papers, namely

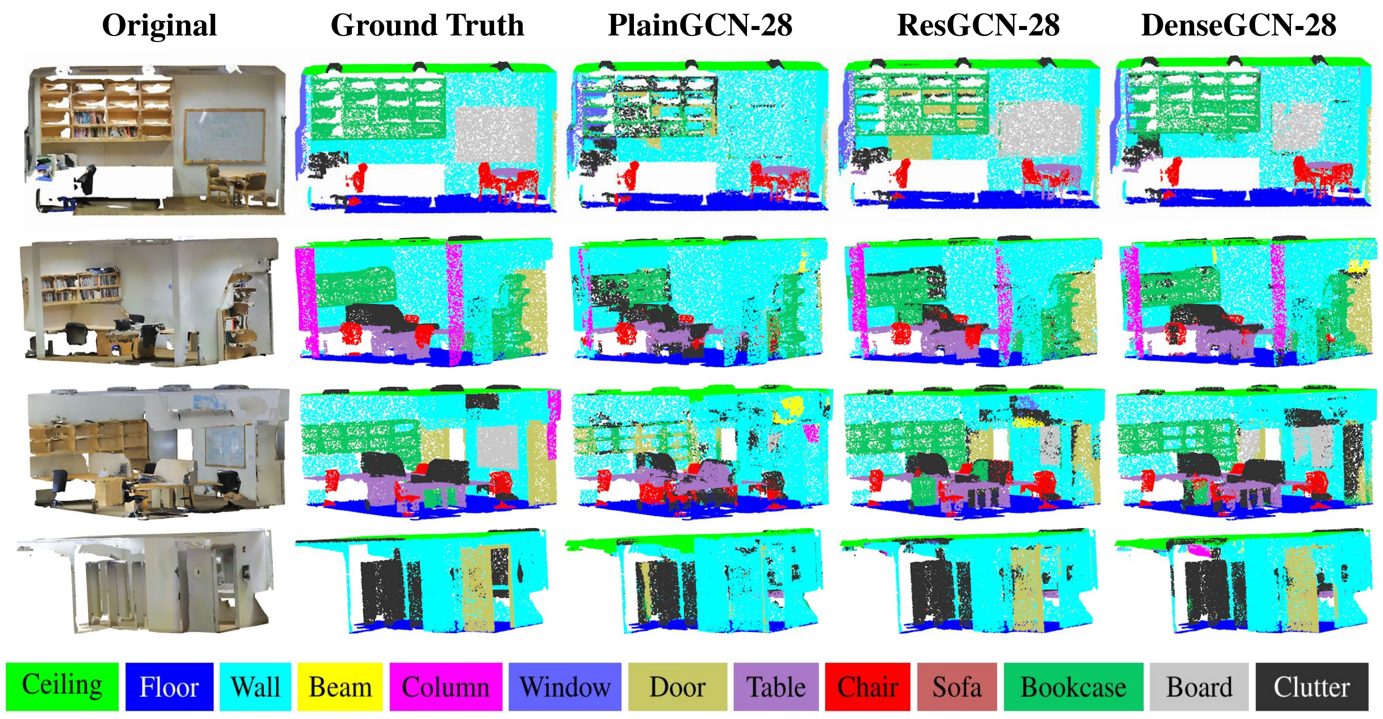

Fig. 6. **Qualitative Results on S3DIS Semantic Segmentation**. We show here the effect of adding residual and dense graph connections to deep GCNs. *PlainGCN-28*, *ResGCN-28*, and *DenseGCN-28* are identical except for the presence of residual graph connections in *ResGCN-28* and dense graph connections 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*.

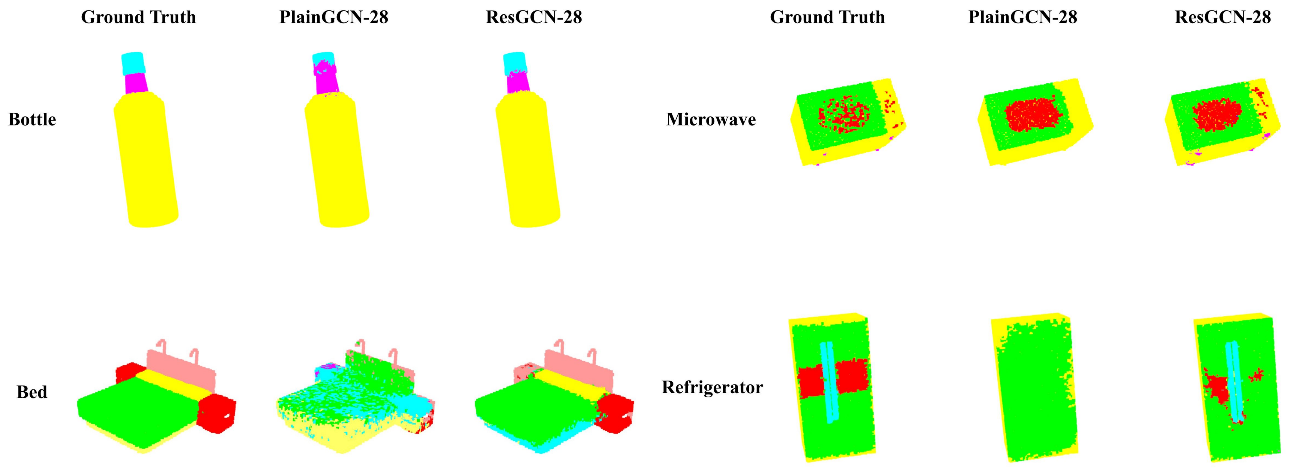

Fig. 7. **Qualitative Results on PartNet Part Segmentation**. We show here the effect of adding residual connections to deep GCNs. *PlainGCN-28* and *ResGCN-28* are identical except for the presence of residual connections in *ResGCN-28*. We note how residual connections have a positive effect on part segmentation compared to *PlainGCN-28*. Many important parts of the objects are classified incorrectly using *PlainGCN-28*.

PointNet [33], PointNet++ [34], SpiderCNN [53] and PointCNN [54].

Qualitative results. Figure 7 shows qualitative results on 4 categories of PartNet [14]: bottle, bed, microwave, and refrigerator. As expected from the results in Table 3, *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 refrigerator, it still outperforms *PlainGCN-28* and segments the important parts of the object. We provide further qualitative results in the **supplementary material**.

Comparison to state-of-the-art. We summarize the results of our performance compared to other state-of-the-art methods in Table 3. Our model *ResGCN-28* performs very well in comparison to current state-of-the-art methods. In particular, *ResGCN-28* achieves better results than previous methods in the categories dishwasher, display, door, earphone, knife, microwave, refrigerator, trash can and vase.

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
DGCNN [6]	84.1	56.1	-	-	-	-	-	-	-	-	-	-	-	-	-
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**

TABLE 2
Comparison of *ResGCN-28* with state-of-the-art on S3DIS Semantic Segmentation. 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%. 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.

Method	bed	bottle	chair	clock	dishw.	disp.	door	earph.	fauc.	knife	lamp	micro.	fridge	st. furn.	table	tr. can	vase
PointNet [33]	13.4	29.5	27.8	28.4	48.9	76.5	30.4	33.4	47.6	32.9	18.9	37.2	33.5	38.0	29.0	34.8	44.4
PointNet++ [34]	30.3	**41.4**	**39.2**	**41.6**	50.1	80.7	32.6	38.4	**52.4**	34.1	**25.3**	48.5	36.4	40.5	**33.9**	46.7	49.8
SpiderCNN [53]	**36.2**	32.2	30.0	24.8	50.0	80.1	30.5	37.2	44.1	22.2	19.6	43.9	39.1	**44.6**	20.1	42.4	32.4
ResGCN-28 (*Ours*)	35.2	36.8	33.8	32.6	**52.7**	**84.4***	**42.5***	**41.9**	49.7	**35.4***	20.0	**54.3**	**46.1***	42.5	14.8	**49.7**	**50.8**
PointCNN [54]	41.9	41.8	43.9	36.3	58.7	82.5	37.8	48.9	60.5	34.1	20.1	58.2	42.9	49.4	21.3	53.1	58.9

TABLE 3
Comparison of *ResGCN-28* with other methods on PartNet Part Segmentation. We report the part-category mean IoU (mIoU) on the fine-grained level of segmentation. *ResGCN-28* which has 28 GCN layers, residual graph connections, and dilated graph convolutions outperforms other methods on 9 out of 17 fine-grained level categories, with substantial improvement of 6.5% on the Door category. PointCNN uses heavy data augmentation and hyper-parameter tuning and therefore it outperforms *ResGCN-28* on some categories. However, *ResGCN-28* outperforms PointCNN on some categories including door, display, knife and refrigerator. '*' denotes *ResGCN-28* results that outperform PointCNN. **bold** denotes best performance among *ResGCN-28*, PointNet, PointNet++, and SpiderCNN.

ResGCN-28 performs substantially better than all previous methods including PointCNN [54], in the difficult door category with an improvement of 4.7% in terms of part-category mIoU. Note that PointCNN uses heavy data augmentation and hyper-parameter tuning unlike our *ResGCN-28*. For this reason, *ResGCN-28* is outperformed by PointCNN in many categories. However, despite this huge disadvantage, *ResGCN-28* outperforms PointCNN on some categories including door, display, knife and refrigerator where PointCNN achieves 37.8%, 82.5%, 34.1% and 42.9% respectively.

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 very 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.

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 which 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 predict multiple labels. The number of filters of the first layer and last layer 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 edges.

5.2 Experimental Setup

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

$$\text{F1-score} = 2 \times \frac{(precision \times recall)}{(precision + recall)} \tag{9}$$

where $precision = \frac{TP'}{P'}$, $recall = \frac{TP'}{T'}$, 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 dataset.

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 *DeepGCNs*. 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.

5.3 Implementation

On this biological network node classification task, we implement all our models based on Pytorch Geometric [55]. We use the Adam optimizer with the same initial learning rate 0.0002 and learning rate schedule with learning rate decay of 80% every $2,000$ gradient decent 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 graph connections* and *dense graph connections* on the performance of multi-label node classification. We also investigate the influence of different parameters, *e.g.* the number of filters $(32, 64, 128, 256)$ and the number of layers $(3, 7, 14, 28, 56, 112)$. To show the generality of our framework, we also apply the proposed *residual graph connections* to multiple GCN variants.

Effect of graph connections. Our experiments in Table 4 show that both *residual graph connections* and *dense graph connections* help train deeper networks. When the network is shallow, models with graph connections achieve similar performance as models without graph connections. However, as the network goes 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% higher than *PlainMRGCN-112* in terms of the m-F1 score. We note that *DenseMRGCN* achieves slightly better performance than *ResMRGCN* with the same network depth and width.

Effect of network depth. The results in Table 4 show that increasing the number of layers improves network performance if *residual graph connections* 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 goes 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 4. 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 leads to better performance, especially for deeper networks where it becomes essential.

Effect of network width. Results of each row in Table 4 show that increasing the number of filters can increase performance consistently. 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 5 shows the effect of using different GCN operators with different model depths. *Residual graph connections* and GCN operators are the only difference when the number of layers is kept the same. The results clearly show that *residual graph connections* in deep networks can help different GCN operators achieve better performance than the *PlainGCN*. Interestingly, when the network goes deeper, the performance of *PlainSemiGCN*, *PlainGAT* and *PlainGIN* decreases dramatically; meanwhile, *PlainEdgeConv* and *PlainMRGCN* only observe a relatively small performance drop. Besides, our proposed *MRGCN* operator achieves the best performance among all the models.

Memory usage of GCN variants. In Figure 8 we compare the total memory usage and performance of different GCN operators. All these models share the the same architecture except for the GCN operations. They all have 56 layers, 256 filters and use *residual graph connections*. We implement all the models with Pytorch geometric and they are all trained on one NVIDIA Tesla V100. The GPU memory usage is measured when the memory usage is stable. Results show that our proposed *ResMRGCN* achieves the best performance and only uses around 15% 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 6. The results clearly show the effectiveness of deeper models with *residual graph connections* and *dense graph connections*. *DenseMRGCN-14* and *ResMRGCN-28* outperform the previous state-of-the-art Cluster-GCN [56] by 0.07% and 0.05% respectively. It is worth mentioning that there are a total of ten models in Table 4 which surpass Cluster-GCN and achieve the new state-of-the-art.

Number of filters	32	64	128	256
PlainMRGCN-3	95.84	97.60	98.58	99.13
PlainMRGCN-7	97.35	98.69	99.22	**99.38**
PlainMRGCN-14	97.55	99.02	99.31	99.34
PlainMRGCN-28	98.09	99.00	99.02	99.31
PlainMRGCN-56	92.70	97.43	97.31	97.61
PlainMRGCN-112	60.75	71.97	89.69	91.50
ResMRGCN-3	96.04	97.60	98.53	99.09
ResMRGCN-7	97.00	98.43	99.19	99.30
ResMRGCN-14	97.75	98.88	99.26	**99.38**
ResMRGCN-28	98.50	99.16	99.29	**99.41**
ResMRGCN-56	98.62	99.27	**99.36**	**99.40**
ResMRGCN-112	98.41	99.34	**99.38**	**99.39**
DenseMRGCN-3	95.96	97.85	98.66	99.11
DenseMRGCN-7	97.87	98.47	99.31	**99.36**
DenseMRGCN-14	98.93	99.00	99.01	**99.43**
DenseMRGCN-28	99.16	99.29	**99.42**	-
DenseMRGCN-56	99.22	-	-	-

TABLE 4
Ablation study on skip graph connections, network width and network depth. m-F1 score is used as the evaluation metric. The unit is %. As expected, we find that residual connections and dense connections can help deep network converge much better compared to the same model without any connection. Also, network width is positively correlated with network performance. Note that '-' denotes that the model is not applicable due to memory limitation and **bold** highlights the models that outperform all state-of-the-art baselines.

Number of layers	3	7	14	28	56
PlainSemiGCN	97.82	90.40	80.55	41.00	50.75
ResSemiGCN	97.88	95.05	93.50	90.60	90.54
PlainGAT	98.52	80.92	56.88	42.40	48.95
ResGAT	98.63	97.86	98.99	99.06	63.25
PlainGIN	97.86	57.78	40.79	35.82	0.26
ResGIN	97.80	96.44	98.22	97.44	97.18
PlainEdgeConv	99.16	99.27	99.30	99.33	98.99
ResEdgeConv	99.03	99.19	99.26	99.30	99.04
PlainMRGCN	99.13	**99.38**	99.34	99.31	97.61
ResMRGCN	99.09	99.30	**99.38**	**99.41**	**99.40**

TABLE 5
Ablation study on network depth and GCN variants. m-F1 score is used as the evaluation metric. The unit is %. We set the number of filters per layer in the backbone network to 256 and vary the number of layers. We find that *residual graph connections* can generally help different GCN operators achieve better performance than the *PlainGCN* when the network goes deep. Note that **bold** highlights the models that outperform all state-of-the-art baselines.

Model	m-F1 score (%)
GraphSAGE [42]	61.20
GATConv [43]	97.30
VR-GCN [57]	97.80
GaAN [58]	98.71
GeniePath [59]	98.50
Cluster-GCN [56]	99.36
***ResMRGCN-28* (Ours)**	**99.41**
***DenseMRGCN-14* (Ours)**	**99.43**

TABLE 6
Comparison of *DenseMRGCN-14* with state-of-the-art on PPI node classfication. 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.

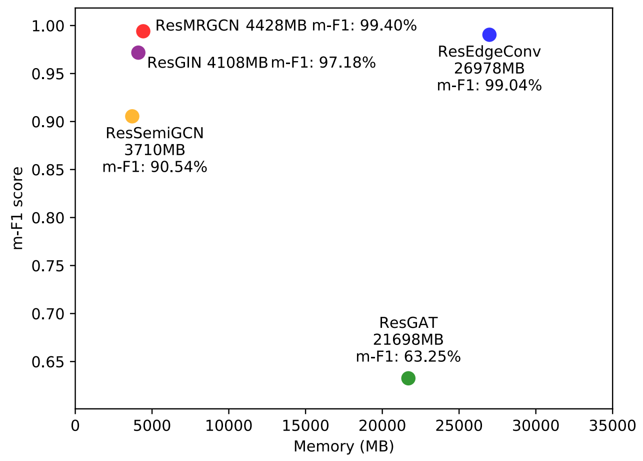

Fig. 8. **Memory usage for different GCNs on PPI node classification.** We keep the same parameters for different models and compare their m-F1 score with their total memory usage. All models in the comparison have 256 filters per layer and 56 layers. We notice that our model *ResMRGCN* uses approximately 1/6 of the total memory used by *ResEdgeConv* and gives a better m-F1 score.

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 impeding further progress. A large number of experiments on semantic segmentation and part segmentation of 3D point clouds and node classification on biological graphs shows the benefit of deeper architectures by state-of-the-art performance. We also show that our approach generalizes across several GCN operators.

On the point cloud data we achieve the best results using EdgeConv [60] as GCN operators for our backbone networks. Moreover, we find that dilated graph convolutions help to gain a larger receptive field without loss of resolution. Even with a small amount of nearest neighbors, deep GCNs can achieve high performance on point cloud semantic segmentation. *ResGCN-112* and *ResGCN-56* perform very well on this task, although they only use 4 and 8 nearest neighbors respectively compared to 16 for *ResGCN-28*. On the biological graph data we achieve the best results using the MRGCN operator which we propose as a novel memory-effiencent alternative to EdgeConv. We successfully trained *ResMRGCN-112* and *DenseMRGCN-56*; both networks converged very well and achieved state-of-the-art results on the PPI dataset.

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 connection and dilated convolutions). In the future, it will be worthwhile to explore how to transfer other operators (*e.g.* deformable convolutions [61]), other architectures (*e.g.* feature pyramid architectures [62]), *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 [60], [63] 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 small. We expect that experimenting on larger datasets will further unleash the full potential of *DeepGCNs*.

Acknowledgments

The authors thank Adel Bibi for his 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.

References

[1] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 817–826.

[2] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[3] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[4] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 3697–3707.

[5] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 974–983.

[6] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *arXiv preprint arXiv:1801.07829*, 2018.

[7] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[9] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[12] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[13] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese, "Joint 2D-3D-Semantic Data for Indoor Scene Understanding," *ArXiv e-prints*, Feb. 2017.

[14] 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 *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[15] G. Li, M. Mller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[16] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Simaan, "Graph convolutional encoders for syntax-aware neural machine translation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1957–1967.

[17] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1506–1515.

[18] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for rgbd semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5199–5208.

[19] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5410–5419.

[20] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, "Graph r-cnn for scene graph generation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 670–685.

[21] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 335–351.

[22] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1219–1228.

[23] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[24] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5308–5317.

[25] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[26] J. Guerry, A. Boulch, B. Le Saux, J. Moras, A. Plyer, and D. Filliat, "Snapnet-r: Consistent 3d multi-view semantic labeling for robotics," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 669–678.

[27] A. Boulch, B. Le Saux, and N. Audebert, "Unstructured point cloud semantic labeling using deep segmentation networks." in *3DOR*, 2017.

[28] Z. Li, Y. Gan, X. Liang, Y. Yu, H. Cheng, and L. Lin, "Lstm-cf: Unifying context modeling and fusion with lstms for rgb-d scene labeling," in *European Conference on Computer Vision*. Springer, 2016, pp. 541–557.

[29] A. Dai, A. X. Chang, M. Savva, M. Halber, T. A. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes." in *CVPR*, vol. 2, 2017, p. 10.

[30] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5648–5656.

[31] G. Riegler, A. O. Ulusoy, and A. Geiger, "Octnet: Learning deep 3d representations at high resolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 3, 2017.

[32] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese, "Segcloud: Semantic segmentation of 3d point clouds," in *3D Vision (3DV), 2017 International Conference on*. IEEE, 2017, pp. 537–547.

[33] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, vol. 1, no. 2, p. 4, 2017.

[34] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems*, 2017, pp. 5099–5108.

[35] F. Engelmann, T. Kontogianni, A. Hermans, and B. Leibe, "Exploring spatial context for 3d semantic segmentation of point clouds," in *IEEE International Conference on Computer Vision, 3DRMS Workshop, ICCV*, 2017.

[36] Q. Huang, W. Wang, and U. Neumann, "Recurrent slice networks for 3d segmentation of point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2626–2635.

[37] X. Ye, J. Li, H. Huang, L. Du, and X. Zhang, "3d recurrent neural networks with context fusion for point cloud semantic segmentation," in *European Conference on Computer Vision*. Springer, 2018, pp. 415–430.

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[39] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Column networks for collective classification," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[40] A. Rahimi, T. Cohn, and T. Baldwin, "Semi-supervised user geolocation via graph convolutional networks," *arXiv preprint arXiv:1804.08049*, 2018.

[41] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *arXiv preprint arXiv:1806.03536*, 2018.

[42] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[43] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[44] N. Peng, H. Poon, C. Quirk, K. Toutanova, and W.-t. Yih, "Cross-sentence n-ary relation extraction with graph lstms," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 101–115, 2017.

[45] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[46] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[47] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[48] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE*

Conference on Computer Vision and Pattern Recognition, 2017, pp. 3693–3702.

[49] D. Valsesia, G. Fracastoro, and E. Magli, "Learning localized generative models for 3d point clouds via graph convolution," 2018.

[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*. Springer, 1990, pp. 286–297.

[51] M. J. Shensa, "The discrete wavelet transform: wedding the a trous and mallat algorithms," *IEEE Transactions on signal processing*, vol. 40, no. 10, pp. 2464–2482, 1992.

[52] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[53] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "Spidercnn: Deep learning on point sets with parameterized convolutional filters," *arXiv preprint arXiv:1803.11527*, 2018.

[54] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," in *NeurIPS*, 2018.

[55] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[56] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," *arXiv preprint arXiv:1905.07953*, 2019.

[57] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," *arXiv preprint arXiv:1710.10568*, 2017.

[58] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," *arXiv preprint arXiv:1803.07294*, 2018.

[59] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4424–4431.

[60] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 1451–1460.

[61] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 764–773.

[62] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

[63] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

Guohao Li obtained the BEng degree in Communication Engineering from Harbin Institute of Technology in 2015. In 2018, he received his Master Degree in Communication and Information Systems from Chinese Academy of Science. He was a research intern at SenseTime. He is currently a Computer Science PhD student at King Abdullah University of Science and Technology. His primary research interests are Computer Vision, Robotics and Deep Learning.

Matthias Müller received his PhD in computer vision from KAUST in 2019. He now works as a research scientist at the Intelligent Systems Lab at Intel. His research interests lie in the fields of computer vision, robotics and machine learning where he has contributed to more than 10 publications in top tier conferences and journals. Matthias was recognized as an outstanding reviewer for CVPR'18 and won the best paper award at the ECCV'18 workshop UAVision.

Guocheng Qian received the BEng degree with first class honors from Xi'an Jiaotong University, China in 2018. He is working towards the MSc degree currently in the Department of Computer Science at King Abdullah University of Science and Technology. His research interests include computer vision, computational photography and neural architecture search.

Itzel C. Delgadillo received the Bachelor in Artificial Intelligence from the Panamerican University, Mexico in 2019. She is currently a Visiting Student Research Intern in 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 in 2018. He was a recipient of the KAUST Gifted Student Program (KGSP) scholarship award. He is currently an MS student and a member of the Image and Video Understanding Lab (IVUL) at the Visual Computing Center (VCC) at King Abdullah University of Science and Technology (KAUST). His research interests include computer vision and deep learning.

Ali Thabet is a Research Scientist at the Visual Computing Center (VCC) in King Abdullah University of Science and Technology (KAUST), working in the Image and Video Understanding Laboratory (IVUL). His research focuses on problems related to 3D computer vision. In general, hes 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, Ali is interested in applying deep learning to image reconstruction, 3D object detection and generation, and autonomous vehicles.

Bernard Ghanem is currently an associate professor with the King Abdullah University of Science and Technology (KAUST), in the Visual Computing Center (VCC). He leads the Image and Video Understanding Lab (IVUL), KAUST. He is involved in several interesting projects that focus on exploiting techniques in computer vision and machine learning for real-world applications including semantic sports video analysis, large-scale activity recognition/detection, and real-time crowd analysis. He has published more than 100 peer-papers in peer-reviewed venues including the IEEE Transactions on Pattern Analysis and Machine Intelligence, the International Journal of Computer Vision, CVPR, ICCV, ECCV, etc. He is a member of the IEEE.

Appendix A
Qualitative Results for DeepGCNs

Figures 9, 10, 11, 12, 13 show qualitative results for DeepGCNs on S3DIS [13] and Figure 14 shows qualitative results for DeepGCNs on PartNet [14].

Appendix B
Run-time Overhead of Dynamic k-NN

We conduct a run-time experiment comparing the inference time of the reference model *ResGCN-28* (28 layers, k=16) with dynamic k-NN and fixed k-NN. The inference time with fixed k-NN is 45.63ms. Computing the dynamic k-NN increases the inference time by 150.88ms. It is possible to reduce computation by updating the k-NN less frequently (*e.g.* computing the dynamic k-NN every 3 layers).

Appendix C
Comparison with DGCNN over All Classes

To showcase the consistent improvement of our framework over the baseline DGCNN [6], we reproduce the results of DGCNN[1] in Table 7 and find our method outperforms DGCNN in all classes.

Class	**DGCNN [6]**	**ResGCN-28 (*Ours*)**
ceiling	92.7	**93.1**
floor	93.6	**95.3**
wall	77.5	**78.2**
beam	32.0	**33.9**
column	36.3	**37.4**
window	52.5	**56.1**
door	63.7	**68.2**
table	61.1	**64.9**
chair	60.2	**61.0**
sofa	20.5	**34.6**
bookcase	47.7	**51.5**
board	42.7	**51.1**
clutter	51.5	**54.4**
mIOU	56.3	**60.0**

TABLE 7
Comparison of *ResGCN-28* with DGCNN. Average per-class results across all areas for our reference network with 28 layers, *residual graph connections* and *dilated graph convolutions* compared to DGCNN baseline. *ResGCN-28* outperforms DGCNN across all the classes. Metric shown is IoU.

1. The results across all classes were not provided in the DGCNN paper.

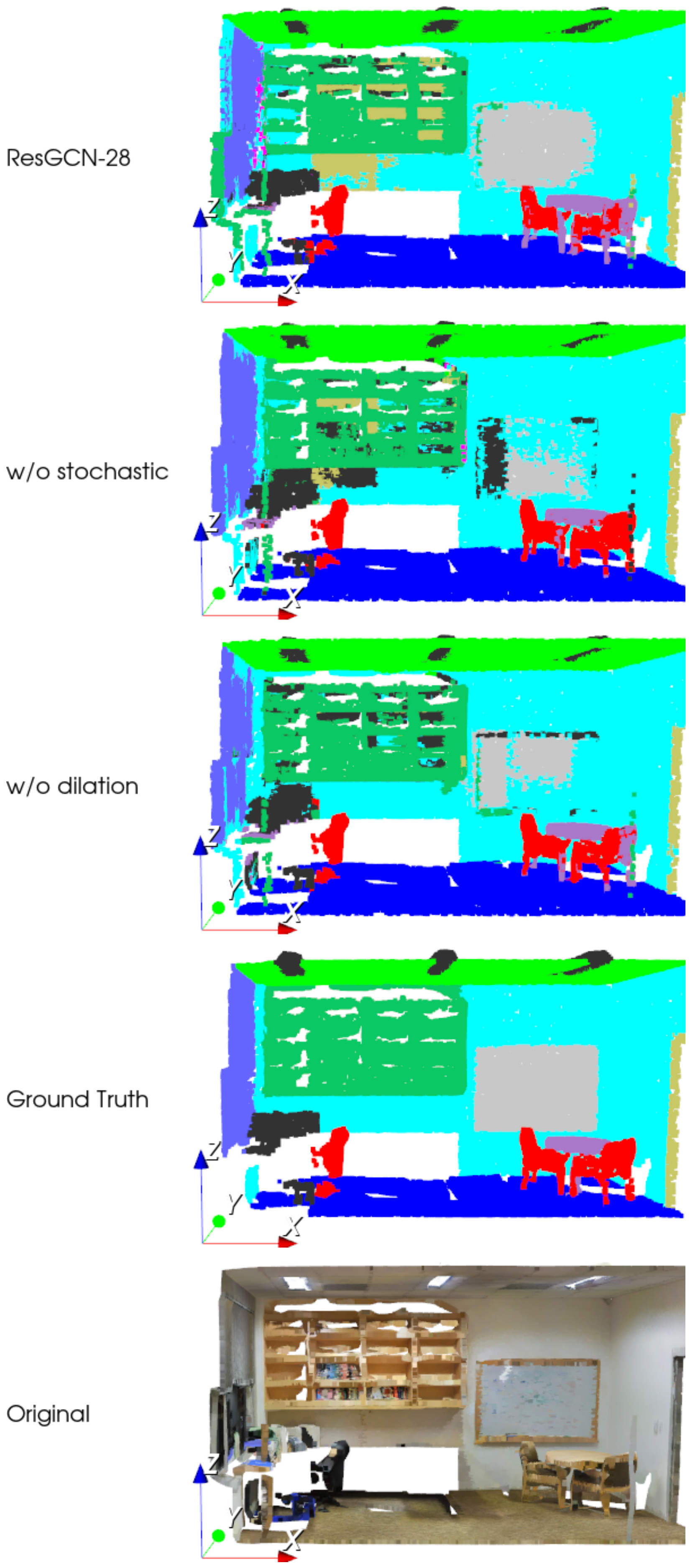

Fig. 9. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of stochastic dilated convolutions.

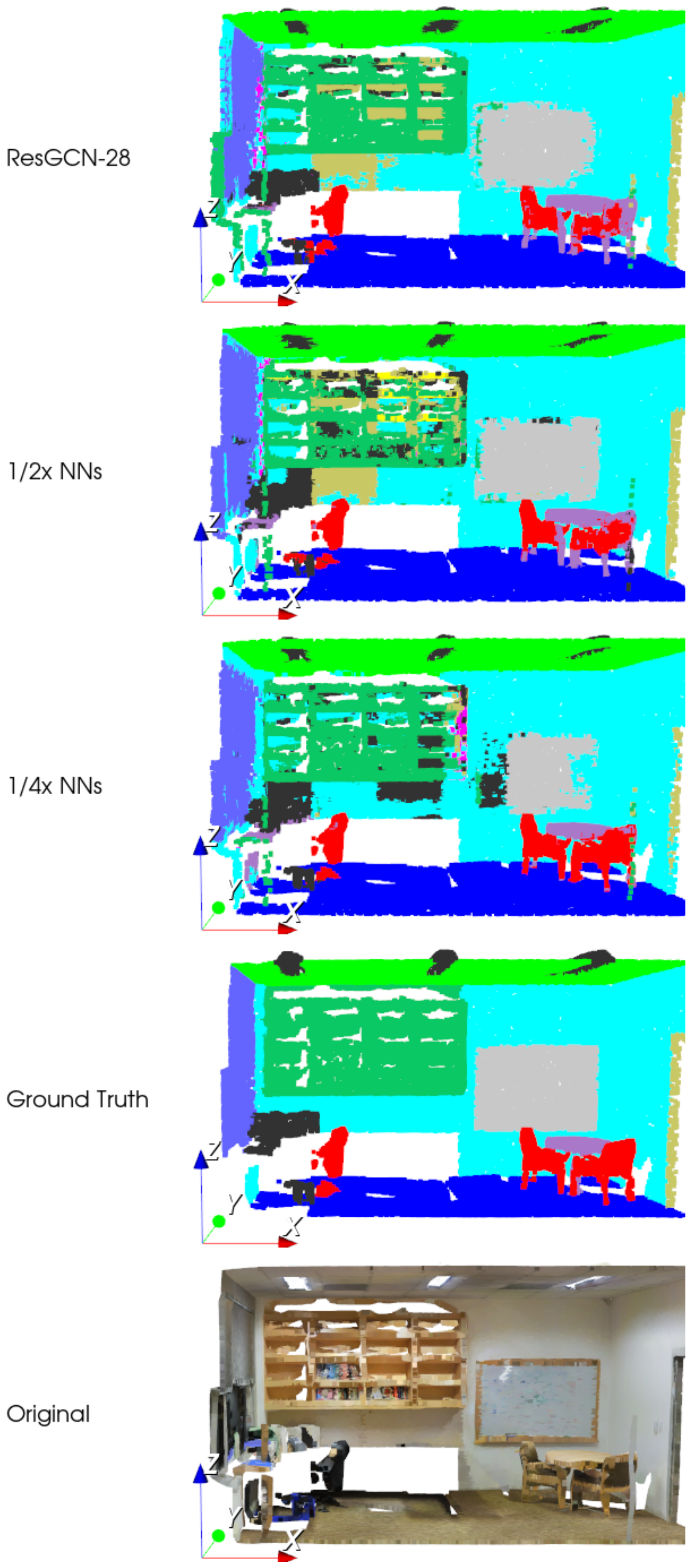

Fig. 10. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of the number of nearest neighbors used in the convolutions.

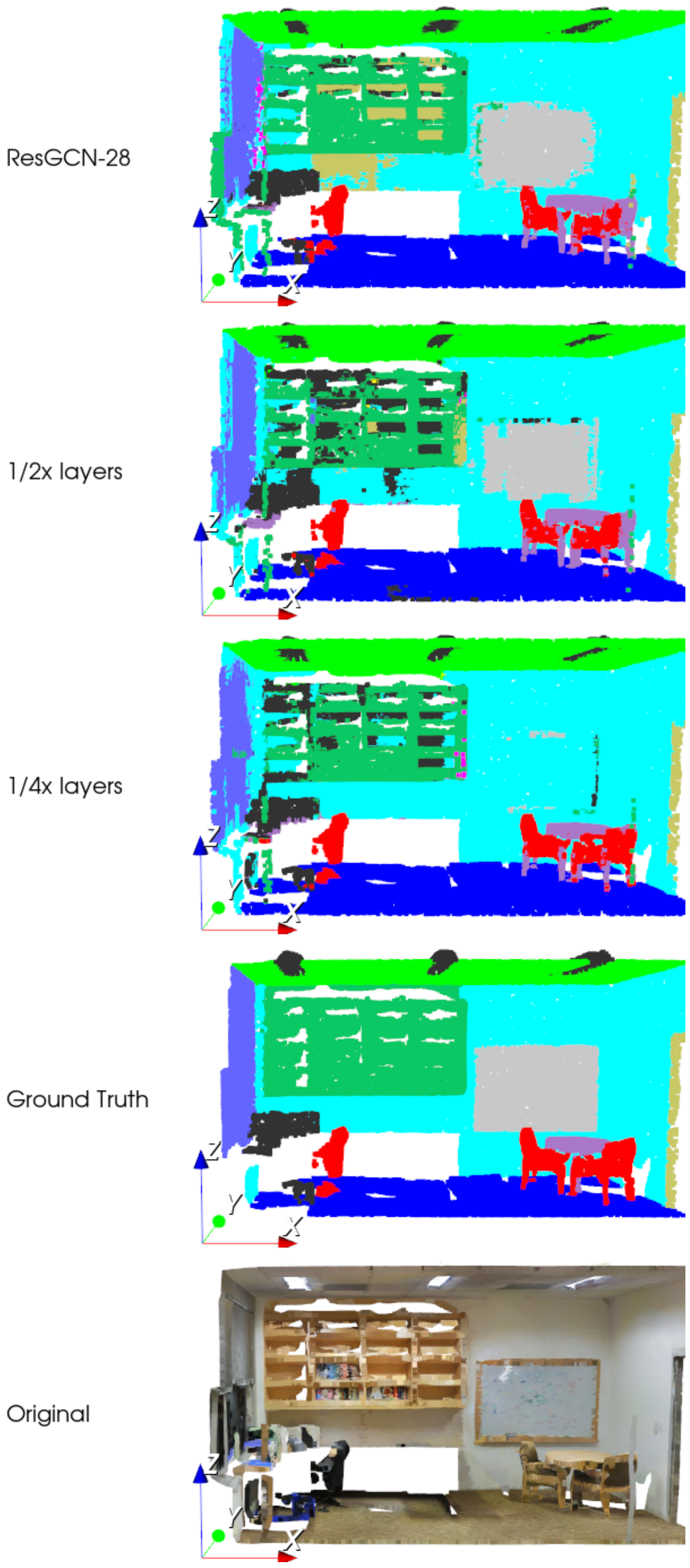

Fig. 11. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of network depth (number of layers).

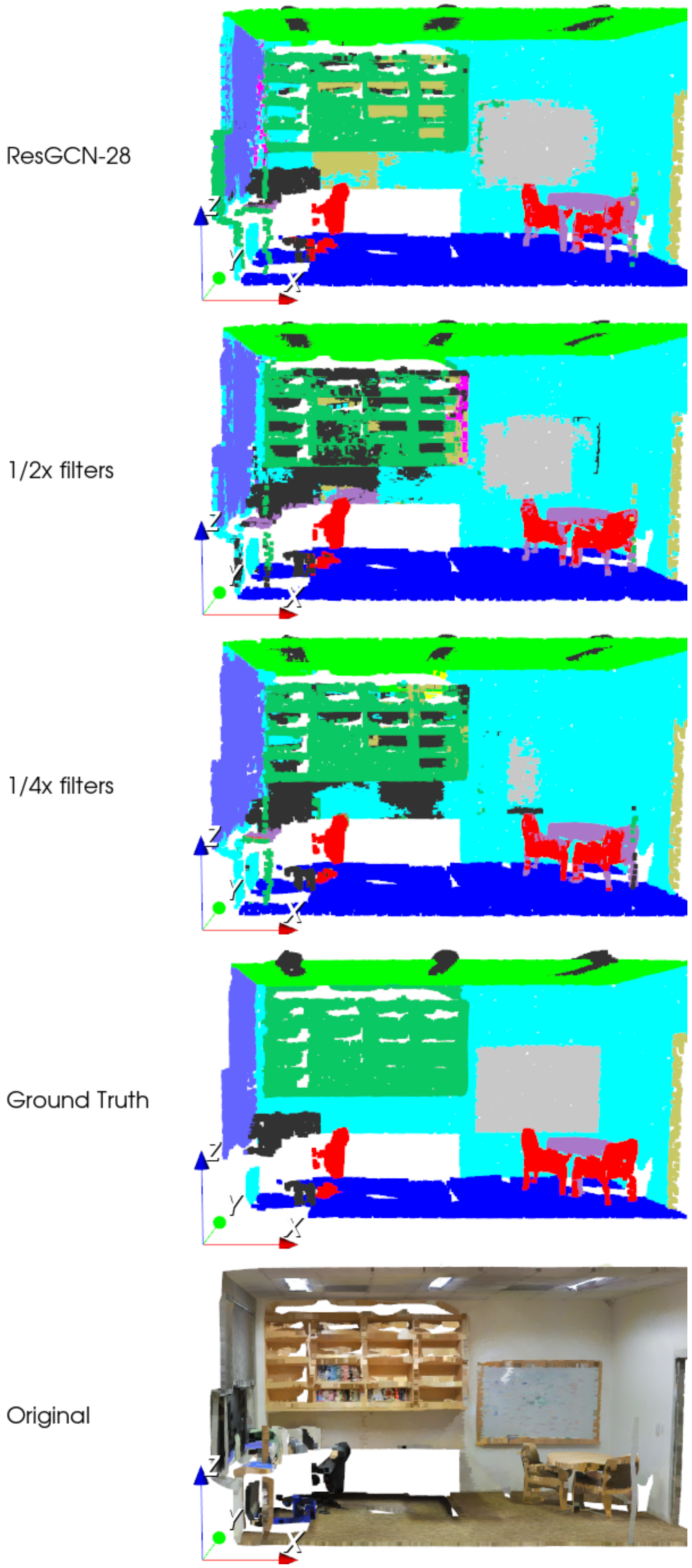

Fig. 12. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of network width (number of filters per layer).

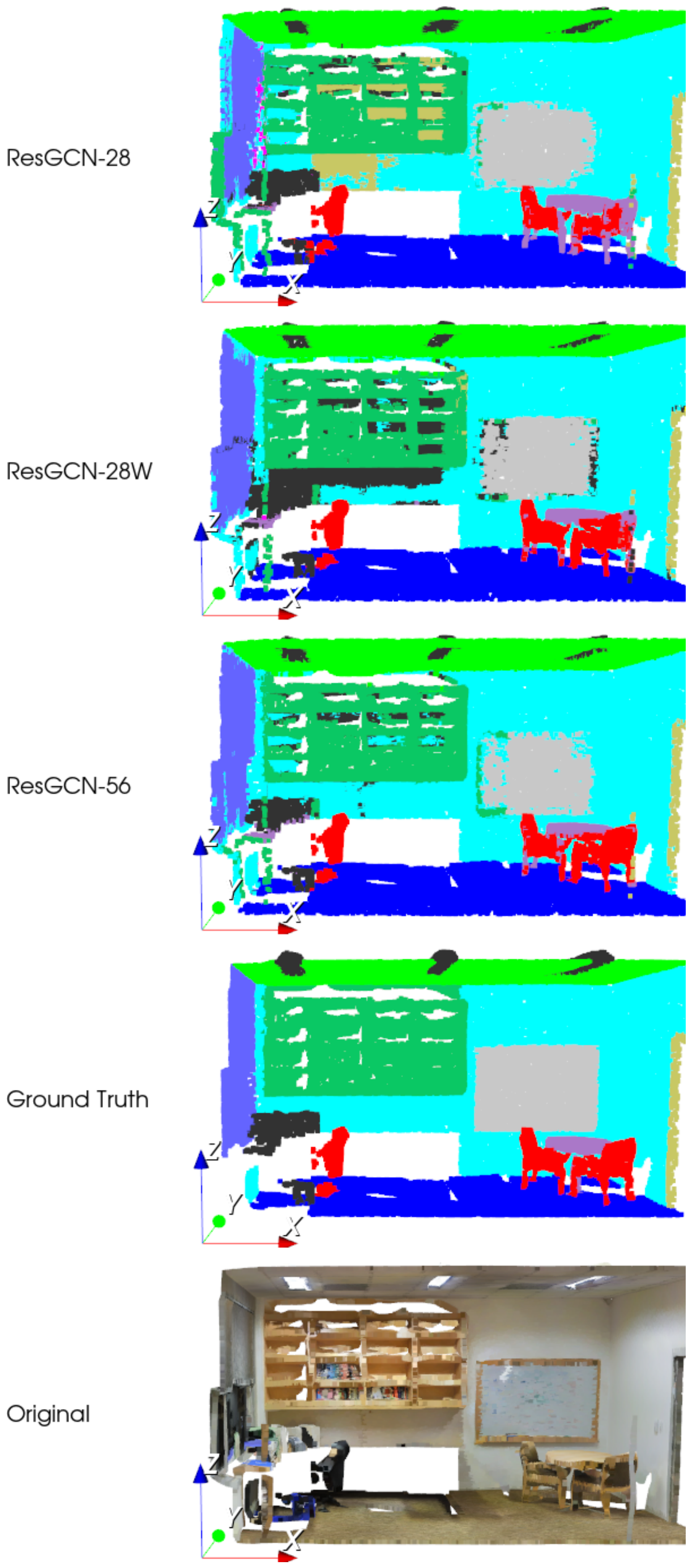

Fig. 13. **Qualitative Results for S3DIS Semantic Segmentation**. We show the benefit of a wider and deeper network even with only half the number of nearest neighbors.

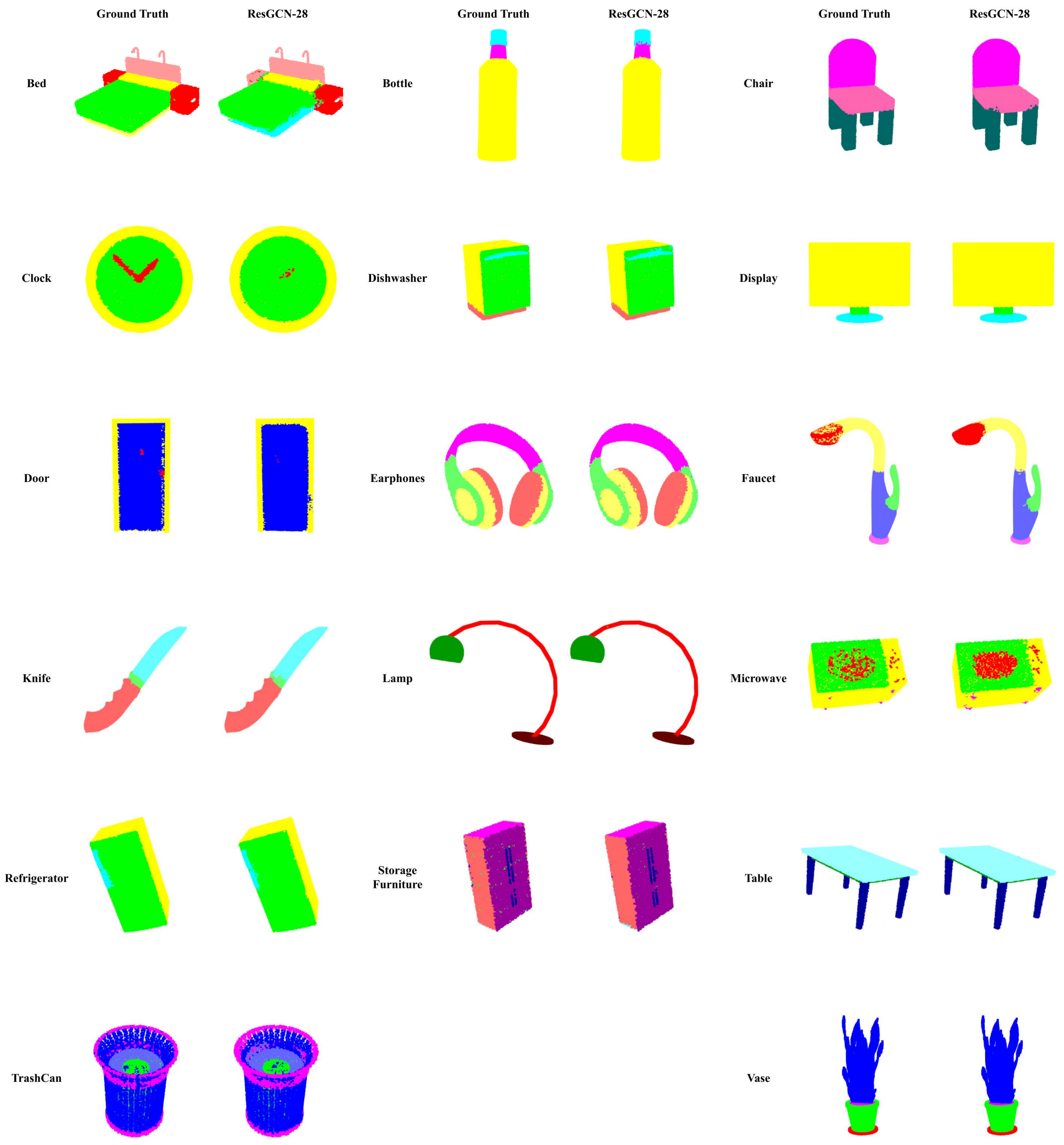

Fig. 14. **Qualitative Results on PartNet Part Segmentation**. We illustrate our performance compared to the ground truth on PartNet.