DeepGCNs: Can GCNs Go as Deep as CNNs?

https://sites.google.com/view/deep-gcns

Guohao Li* Matthias Müller* Ali Thabet Bernard Ghanem
Visual Computing Center, KAUST, Thuwal, Saudi Arabia
{guohao.li, matthias.mueller.2, ali.thabet, bernard.ghanem}@kaust.edu.sa

Abstract

Convolutional Neural Networks (CNNs) achieve impressive performance in a wide variety of fields. Their success benefited from a massive boost when very deep CNN models were able to be reliably trained. Despite their merits, CNNs fail to properly address problems with non-Euclidean data. To overcome this challenge, Graph Convolutional Networks (GCNs) build graphs to represent non-Euclidean data, borrow concepts from CNNs, and apply them in training. GCNs show promising results, but they are usually limited to very shallow models due to the vanishing gradient problem (see Figure 1). As a result, most state-of-the-art GCN models are no deeper than 3 or 4 layers. In this work, we present new ways to successfully train very deep GCNs. We do this by borrowing concepts from CNNs, specifically residual/dense connections and dilated convolutions, and adapting them to GCN architectures. Extensive experiments show the positive effect of these deep GCN frameworks. Finally, we use these new concepts to build a very deep 56-layer GCN, and show how it significantly boosts performance (+3.7% mIoU over state-of-the-art) in the task of point cloud semantic segmentation. We believe that the community can greatly benefit from this work, as it opens up many opportunities for advancing GCN-based research.

1. Introduction

GCNs have been gaining a lot of momentum in the last few years. This increased interest is attributed to two main factors: the increasing proliferation of non-Euclidean data in real-world applications, and the limited performance of CNNs when dealing with such data. GCNs operate directly on non-Euclidean data and are very promising for applications that depend on this information modality. GCNs are currently used to predict individual relations in social networks [36], model proteins for drug discovery [54, 40], enhance predictions of recommendation engines [24, 50], efficiently segment large point clouds [42], among other fields.

*equal contribution

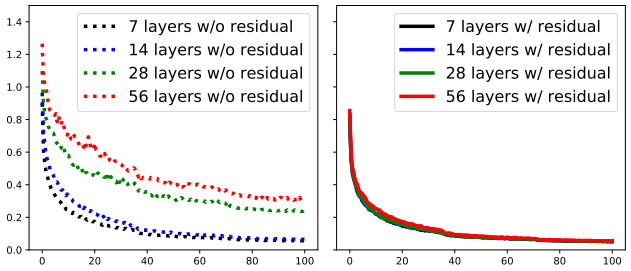

Figure 1. **Training Deep GCNs**. (*left*) We show the training loss for GCNs with 7, 14, 28, and 56 layers, with and without residual connections. We note how adding more layers without residual connections translates to substantially higher loss. (*right*) In contrast, training GCNs with residual connections results in consistent stability across all depths.

A key reason behind the success of CNNs is the ability to design and reliably train very deep CNN models. In contrast, it is not yet clear how to properly train deep GCN architectures, where several works have studied their limitations [19, 43, 53]. Stacking more layers into a GCN leads to the common vanishing gradient problem. This means that back-propagating through these networks causes over-smoothing, eventually leading to features of graph vertices converging to the same value [19]. Due to these limitations, most state-of-the-art GCNs are no deeper than 4 layers [53].

Vanishing gradients is not a foreign phenomenon in the world of CNNs. It also posed limitations on the depth growth of these types of networks. ResNet [11] provided a big step forward in the pursuit of very deep CNNs when it introduced residual connections between input and output layers. These connections massively alleviated the vanishing gradient problem. Today, ResNets can reach 152 layers and beyond. Further extension came with DenseNet [13], where more connections are introduced across layers. More layers could potentially mean more spatial information loss due to pooling. This issue was also addressed, with Dilated Convolutions [51]. The introductions of these key concepts had substantial impact on the progress of CNNs, and we believe they can have a similar effect if well adapted to GCNs.

In this work, we present an extensive study of methodologies that allow for training very deep GCNs. We adapt concepts that were successful in training deep CNNs, mainly residual connections, dense connections, and dilated convolutions. We show how we can incorporate these layers into a graph framework, and present an extensive analysis of the effect of these additions to the accuracy and stability of deep GCNs. To showcase these layer adaptations, we apply them to the popular task of point cloud semantic segmentation. We show that adding a combination of residual and dense connections, and dilated convolutions, enables successful training of GCNs up to 56 layers deep (refer to Figure 1). This very deep GCN improves the state-of-the-art on the challenging S3DIS [1] point cloud dataset by 3.7%.

Contributions. We summarize our contributions as three fold. **(1)** We adapt residual/dense connections, and dilated convolutions to GCNs. **(2)** We present extensive experiments on point cloud data, showing the effect of each of these new layers to the stability and performance of training deep GCNs. We use point cloud semantic segmentation as our experimental testbed. **(3)** We show how these new concepts help build a 56-layer GCN, the deepest GCN architecture by a large margin, and achieve close to 4% boost in state-of-the-art performance on the S3DIS dataset [1].

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 [36], 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 [54, 40]. Understanding the bioactivities of these molecules can have substantial impact on drug discovery. Another popular use of graphs is in recommendation engines [24, 50], where accurate modelling of user interactions leads to improved product recommendations. Graphs are also popular modes of representation in natural language processing [2, 23], 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 [30, 44, 48, 20]. Scene graphs also facilitate the inverse process, where an image is reconstructed given a graph representation of the scene [17]. Graphs are also used to model human joints for action recognition in video [47, 16]. 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 [35, 9, 3, 22], or by voxelization [5, 28, 32, 37]. More recent work focuses on directly processing unordered point cloud representations [27, 29, 8, 14, 49]. The recent *EdgeConv* method by Wang *et al*. [42] 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 attempt 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 [18]. Pham *et al*. [26] proposed Column Network (CLN) for collective classification in relational learning and showed peak performance with 10 layers with the performance degrading for deeper graphs. Rahimi *et al*. [31] 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*. [46] 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*. [19] 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 [43, 53] also show the limitations of stacking multiple GCN layers, which lead 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 [11, 51]. 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 [11]. By adding residual connections between inputs and outputs of layers, ResNet tends to alleviate the vanishing gradient problem. DenseNet [13] takes

this idea a step further and adds connections across layers as well. Dilated Convolutions [51] are a more recent approach that has lead to significant performance gains, specifically in image-to-image translation tasks such as semantic segmentation [51], 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 the work of Wang *et al*. [42] to a much deeper GCN, and therefore significantly increasing its performance. Extensive experiments on the task of point cloud semantic segmentation 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 [18], a max-pooling aggregator [27, 10, 42], an attention aggregator [39] or an LSTM aggregator [25]. The update function can be a multi-layer perceptron [10, 7], a gated network [21], *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 [15] 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 [34, 42, 38] 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) [34] uses dynamic edge-conditional filters to learn an edge-specific weight matrix. Moreover, EdgeConv [42] 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) [38] 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 field, when deeper GCNs are considered. In our framework, we propose to recompute edges between vertices via a *Dilated k-NN* function in the feature space of each layer to further increase the receptive field. In what follows, we provide detailed description of three operations that can enable much deeper GCNs to be trained: residual connections, dense connections, and dilated aggregation.

3.2. Residual Learning for GCNs

Designing deep GCN architectures [43, 53] is an open problem in the graph learning space. Recent work [19, 43, 53] 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 3 layers deep [53]. Inspired by the huge success of ResNet [11], DenseNet [13] and Dilated Convolutions [51],

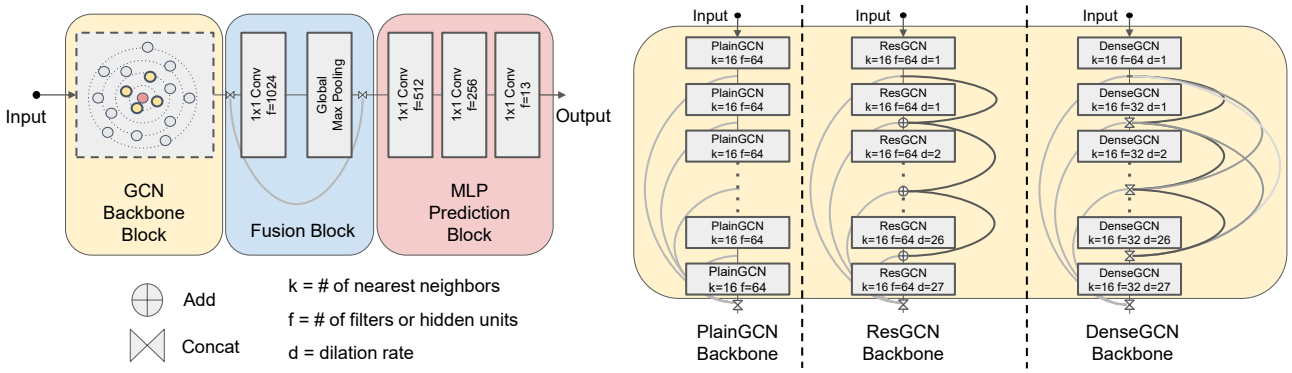

Figure 2. **Proposed GCN architecture for point cloud semantic segmentation**. *(left)* Our framework consists of three blocks: a GCN Backbone Block (feature transformation of input point cloud), a Fusion Block (global feature generation and fusion), and an MLP Prediction Block (point-wise label prediction). *(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*).

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 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*.

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

3.3. Dense Connections in GCNs

DenseNet [13] 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 [13]). 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 in GCNs

Dilated wavelet convolution is an algorithm originating from the wavelet processing domain [12, 33]. To alleviate spatial information loss caused by pooling operations, Yu *et al*. [51] 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 of semantic segmentation tasks. 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 fields 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*.

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

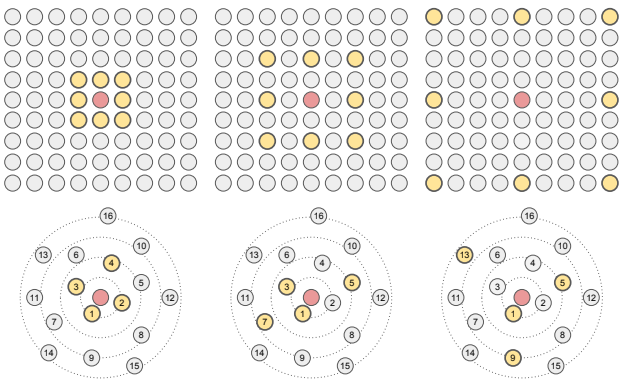

Figure 3. **Dilated Convolution in GCNs**. Visualization of dilated convolution on a structured graph arranged in a grid (*e.g.* 2D image) and on a general structured graph. (*top*) 2D convolution with kernel size 3 and dilation rate 1, 2, 4 (left to right). (*bottom*) Dynamic graph convolution with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

Therefore, 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: $\mathcal{G}^{(d)} = (\mathcal{V}^{(d)}, \mathcal{E}^{(d)})$. 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.

4. Experiments

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 task of large-scale point cloud segmentation 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. 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 [42] and other state-of-the-art methods.

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. In the S3DIS semantic segmentation task, 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 [27] and DGCNN [42]. 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 28 EdgeConv [42] layers with dynamic k-NN, each of which is similar to the one used in DGCNN [42]. 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 *ResGCN*.

4.4. Implementation

We implement all our models using Tensorflow. For fair comparison, we use the Adam optimizer with the same initial learning rate 0.001 and the same learning rate schedule; the learning rate decays 50% every 3×10^5 gradient decent steps. The networks are trained with two NVIDIA Tesla V100 GPUs using data parallelism. The batch size is set to 8 for each GPU. 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.

4.5. Results

For convenient referencing, we use the naming convention *BackboneBlock-#Layers* to denote the key models in our analysis and we provide all names in Table 1. 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.

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). Overall, we conduct 20 experiments and show their results in Table 1.

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 [11]. When the residual graph connections between layers are removed (*i.e.* in *PlainGCN-28*), performance dramatically degrades (-12% mIoU). In Appendices A and B, we show similar performance gains by combining residual graph connections and dilated graph convolutions with other types of GCN layers.

Effect of dilation. Results in Table 1 (*Dilation*) [51] 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' 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.

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 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.

Ablation	Model	mIoU	ΔmIoU	dynamic	connection	dilation	stochastic	# NNs	# filters	# layers
Reference	*ResGCN-28*	**52.49**	0.00	✓	⊕	✓	✓	16	64	28
Dilation		51.98	-0.51	✓	⊕	✓		16	64	28
		49.64	-2.85	✓	⊕			16	64	28
	PlainGCN-28	**40.31**	-12.18	✓				16	64	28
Fixed k-NN		48.38	-4.11		⊕			16	64	28
		43.43	-9.06					16	64	28
Connections	*DenseGCN-28*	**51.27**	-1.22	✓	⋈	✓	✓	8	32	28
		40.47	-12.02	✓		✓	✓	16	64	28
		38.79	-13.70	✓		✓	✓	8	64	56
		49.23	-3.26	✓		✓	✓	16	64	14
		47.92	-4.57	✓		✓	✓	16	64	7
Neighbors		49.98	-2.51	✓	⊕	✓	✓	8	64	28
		49.22	-3.27	✓	⊕	✓	✓	4	64	28
Depth	*ResGCN-56*	**53.64**	1.15	✓	⊕	✓	✓	8	64	56
	ResGCN-14	49.90	-2.59	✓	⊕	✓	✓	16	64	14
	ResGCN-7	48.95	-3.53	✓	⊕	✓	✓	16	64	7
Width	*ResGCN-28W*	**53.78**	1.29	✓	⊕	✓	✓	8	128	28
		49.18	-3.31	✓	⊕	✓	✓	32	32	28
		48.80	-3.69	✓	⊕	✓	✓	16	32	28
		45.62	-6.87	✓	⊕	✓	✓	16	16	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 ⊕ and ⋈ symbols respectively. We highlight the most important results in bold. ΔmIoU denotes the difference in mIoU with respect to the reference model *ResGCN-28*.

Qualitative Results. Figure 4 shows qualitative results on area 5 of S3DIS [1]. 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 Appendices C, D and E for more qualitative results and other ablation studies.

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 [42] by 3.9% (absolute) in mean IoU, even though DGCNN has the same fusion and MLP prediction blocks as *ResGCN-28* but with a shallower *PlainGCN* 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%.

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 4 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 [42] 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.

5. Conclusion and Future Work

In this work, we investigate how to bring proven useful concepts (residual connections, dense connections and dilated convolutions) from CNNs to GCNs and answer the question: how can GCNs be made deeper? Extensive experiments show that by adding skip connections to GCNs, we can alleviate the difficulty of training, which is the primary problem impeding GCNs to go deeper. Moreover, 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-56* performs very well on this task, although it uses only 8 nearest neighbors compared to 16 for *ResGCN-28*. We were also able to train *ResGCN-151* for 80 epochs; the network converged very well and achieved similar results as *ResGCN-28* and *ResGCN-56* but with only 3 nearest neighbors. Due to com-

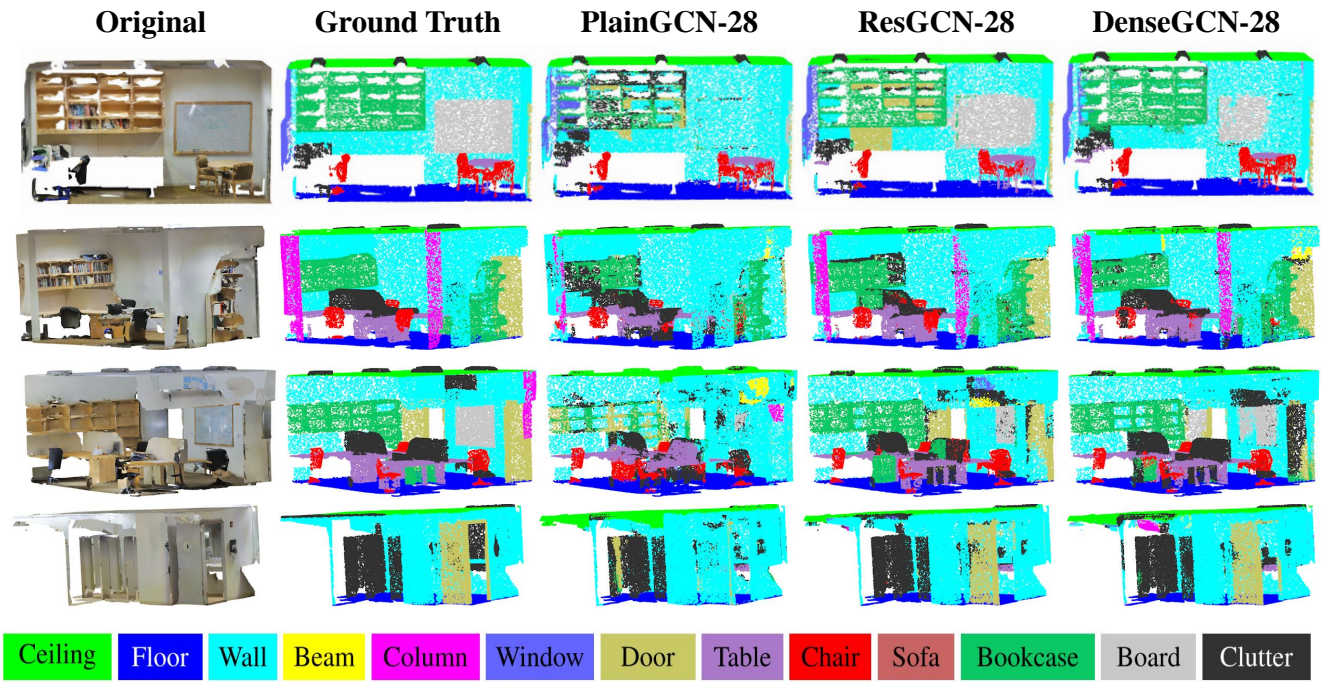

Figure 4. **Qualitative Results on S3DIS Semantic Segmentation**. We show here the effect of adding residual and dense graph connections to deep GCNs. *PlainGCN-28*, *ResGCN-28*, and *DenseGCN-28* are identical except for the presence of residual graph connections in *ResGCN-28* and dense graph connections 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*.

Method	OA	mIOU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
PointNet [27]	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 [8]	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 [8]	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++ [29]	-	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 [49]	**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 [42]	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*, which has 28 GCN layers, residual graph connections, and dilated graph convolutions, and state-of-the-art baselines. *ResGCN-28* outperforms 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.

putational constraints, we were unable to investigate such deep architectures in detail and leave it for future work.

Our results show that after solving the vanishing gradient problem plaguing deep GCNs, we can either make GCNs deeper or wider (*e.g. ResGCN-28W*) to get better performance. We expect GCNs to become a powerful tool for processing non-Euclidean data in computer vision, natural language processing, and data mining. We show successful cases for adapting concepts from CNNs to GCNs. In the future, it will be worthwhile to explore how to transfer other operators, *e.g.* deformable convolutions [6], other architectures, *e.g.* feature pyramid architectures [52], *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 [4, 41] 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.

Acknowledgments. The authors thank Adel Bibi and Guocheng Qian 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.

References

[1] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017. 2, 7

[2] 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*, pages 1957–1967, 2017. 2

[3] A. Boulch, B. Le Saux, and N. Audebert. Unstructured point cloud semantic labeling using deep segmentation networks. In *3DOR*, 2017. 2

[4] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 8

[5] 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*, volume 2, page 10, 2017. 2

[6] 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*, pages 764–773, 2017. 8

[7] 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*, pages 2224–2232, 2015. 3

[8] 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. 2, 8

[9] 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)*, pages 669–678, 2017. 2

[10] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017. 3, 11

[11] 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*, pages 770–778, 2016. 1, 2, 4, 6

[12] 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*, pages 286–297. Springer, 1990. 4

[13] 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*, pages 4700–4708, 2017. 1, 3, 4

[14] 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*, pages 2626–2635, 2018. 2

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 3

[16] 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*, pages 5308–5317, 2016. 2

[17] 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*, pages 1219–1228, 2018. 2

[18] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2, 3

[19] 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. 1, 2, 3

[20] 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)*, pages 335–351, 2018. 2

[21] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015. 3

[22] 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*, pages 541–557. Springer, 2016. 2

[23] 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*, pages 1506–1515, 2017. 2

[24] F. Monti, M. Bronstein, and X. Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pages 3697–3707, 2017. 1, 2

[25] 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*, 5:101–115, 2017. 3

[26] T. Pham, T. Tran, D. Phung, and S. Venkatesh. Column networks for collective classification. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 2

[27] 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*, 1(2):4, 2017. 2, 3, 5, 8

[28] 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*, pages 5648–5656, 2016. 2

[29] 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*, pages 5099–5108, 2017. 2, 8

[30] 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*, pages 5199–5208, 2017. 2

[31] A. Rahimi, T. Cohn, and T. Baldwin. Semi-supervised user geolocation via graph convolutional networks. *arXiv preprint arXiv:1804.08049*, 2018. 2

[32] 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*, volume 3, 2017. 2

[33] M. J. Shensa. The discrete wavelet transform: wedding the a trous and mallat algorithms. *IEEE Transactions on signal processing*, 40(10):2464–2482, 1992. 4

[34] 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*, pages 3693–3702, 2017. 3

[35] 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*, pages 945–953, 2015. 2

[36] 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*, pages 817–826. ACM, 2009. 1, 2

[37] 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*, pages 537–547. IEEE, 2017. 2

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

[39] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 3

[40] N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008. 1, 2

[41] 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)*, pages 1451–1460. IEEE, 2018. 8

[42] 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. 1, 2, 3, 5, 6, 7, 8, 11, 12

[43] 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. 1, 2, 3

[44] 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*, pages 5410–5419, 2017. 2

[45] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 11, 12

[46] 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. 2

[47] 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. 2

[48] 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)*, pages 670–685, 2018. 2

[49] 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*, pages 415–430. Springer, 2018. 2, 8

[50] 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*, pages 974–983. ACM, 2018. 1, 2

[51] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 1, 2, 3, 4, 6

[52] 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*, pages 2881–2890, 2017. 8

[53] 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. 1, 2, 3, 4

[54] M. Zitnik and J. Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017. 1, 2

A. Deep GCN Variants

In our experiments in the paper, we work with a GCN based on EdgeConv [42] to show how very deep GCNs can be trained. However, it is straightforward to build other deep GCNs with the same concepts we proposed (*e.g. residual/dense graph connections*, *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 [10], Graph Isomorphism Network (GIN) [45] and *MRGCN* (Max-Relative GCN) which is a new GCN operation we proposed. In practice, we find that EdgeConv learns a better representation than the other implementations. However, it is less memory and computation efficient. Therefore, we propose a simple GCN combining the advantages of them all.

All of the ResGCNs have the same components (*e.g.* dynamic $k-NN$, residual connections, stochastic dilation) and parameters (*e.g.* #NNs, #filters and #layers) as *ResGCN-28* in Table **Ablation Study** of the paper except for the internal GCN operations. To simplify, we refer to these models as *ResEdgeConv*, *ResGraphSAGE*, *ResGIN* and *NewResGCN* respectively. Note that *ResEdgeConv* is an alias for *ResGCN* in our paper. We refer to it as *ResEdgeConv* to distinguish it from the other GCN operations.

ResEdgeConv. Instead of aggregating neighborhood features directly, EdgeConv [42] 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:

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

ResGraphSAGE. GraphSAGE [10] 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*:

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

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 $\mathbf{h}^{res}_{v_{l+1}} = \mathbf{h}^{res}_{v_{l+1}} / \left\|\mathbf{h}^{res}_{v_{l+1}}\right\|_2$.

ResGIN. The main difference between GIN [45] 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:

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

ResMRGCN. 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 nonlinear transforms. We refer to this simple GCN as *MRGCN* (Max-Relative GCN). The residual version of *MRGCN* is as such:

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

Where $\mathbf{h}_{v_{l+1}}$ and $\mathbf{h}_{v_l}$ are the hidden state of vertex v at $l+1$; $\mathbf{h}^{res}_{v_{l+1}}$ 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*.

B. Results for Deep GCN Variants

Table 3 shows a comparison of different deep residual GCNs variants on the task of semantic segmentation; we report the mIOU for area 5 of S3DIS. All deep GCN variants are 28 layers deep and we denote them as *ResEdgeConv-28*, *ResGraphSAGE-28*, *ResGraphSAGE-N-28*, *ResGIN-ϵ-28* and *ResMRGCN-28*; *ResGraphSAGE-28* is GraphSAGE without normalization, *ResGraphSAGE-N-28* is the version with normalization. The results clearly show that different deep GCN variants with *residual graph connections* and *dilated graph convolutions* converge better than the *PlainGCN*. *ResMRGCN-28* achieves almost the same performance as *ResEdgeConv-28* while only using half of the GPU memory. *ResGraphSAGE-28* and *ResGraphSAGE-N-28* are slightly worse than *ResEdgeConv-28* and *ResMRGCN-28*. The results also show that using normalization for *ResGraphSAGE* is not essential. Interestingly, we find that *ResGIN-ϵ-28* converges

Model	mIoU	ΔmIoU	dynamic	connection	dilation	stochastic	# NNs	# filters	# layers
ResEdgeConv-28	**52.49**	0.00	✓	⊕	✓	✓	16	64	28
PlainGCN-28	**40.31**	-12.18	✓				16	64	28
ResGraphSAGE-28	**49.20**	-3.29	✓	⊕	✓	✓	16	64	28
ResGraphSAGE-N-28	**49.02**	-3.47	✓	⊕	✓	✓	16	64	28
ResGIN-ϵ-28	**42.81**	-9.68	✓	⊕	✓	✓	16	64	28
ResMRGCN-28	**51.17**	-1.32	✓	⊕	✓	✓	16	64	28

Table 3. **Comparisons of Deep GCNs variants on area 5 of S3DIS**. We compare our different types of ResGCN (*ResEdgeConv*, *ResGraphSAGE*, *ResGIN* and *ResMRGCN*) with 28 layers. *Residual graph connections* and *Dilated graph convolutions* are added to all the GCN 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 with the ⊕ symbols.

well during the training phase and has a high training accuracy. However, it fails to generalize to the test set. This phenomenon is also observed in the original paper [45] in which they find setting ϵ to 0 can get the best performance. Therefore, we can draw the conclusion that the concepts we proposed (*e.g. residual/dense graph connections* and *dilated graph convolutions*) generalize well to different types of GCNs and enable training very deep GCNs.

C. Qualitative Results for the Ablation Study

We summarize the most important insights of the ablation study in Figure 5. Figures 6, 7, 8, 9, 10 show qualitative results for the ablation study presented in the paper.

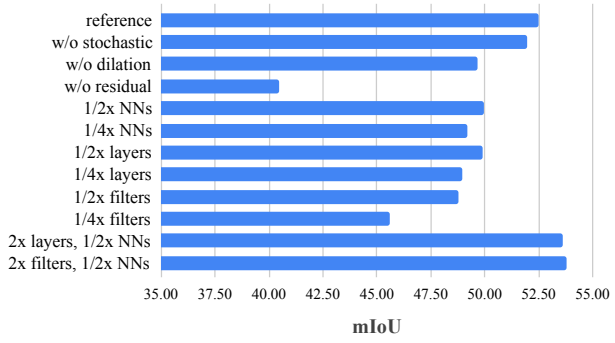

Figure 5. **Ablation study on area 5 of S3DIS**. We compare our reference network (*ResGCN-28*) with 28 layers, *residual graph connections* and *dilated graph convolutions* to several ablated variants. All models were trained for 100 epochs on all areas except for area 5 with the same hyper-parameters.

D. Run-time Overhead of Dynamic k-NN

We conduct a run-time experiment comparing the inference time of the reference model (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).

E. Comparison with DGCNN over All Classes

To showcase the consistent improvement of our framework over the baseline DGCNN [42], we reproduce the results of DGCNN[1] in Table 4 and find our method outperforms DGCNN in all classes.

Class	DGCNN [42]	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 4. **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 over all classes were not provided in the original DGCNN paper

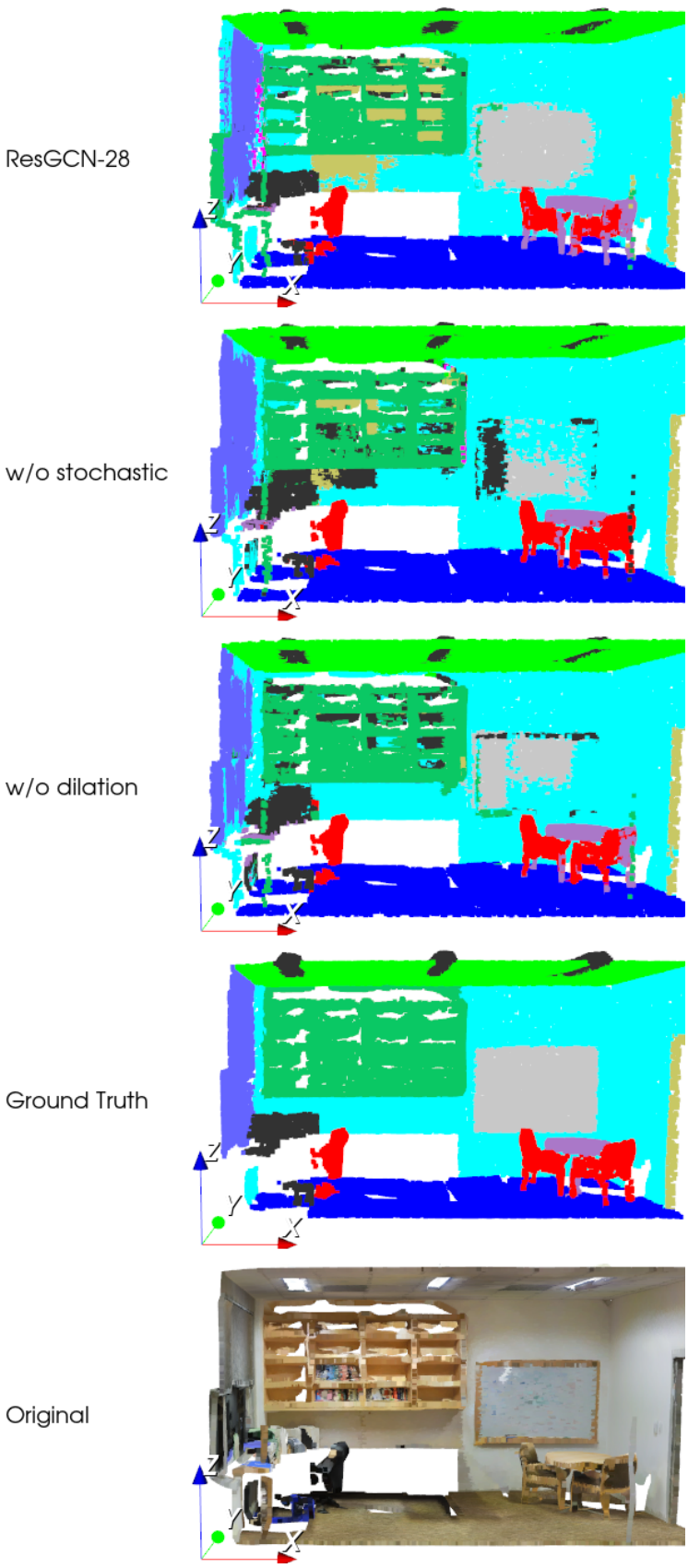

Figure 6. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of stochastic dilated convolutions.

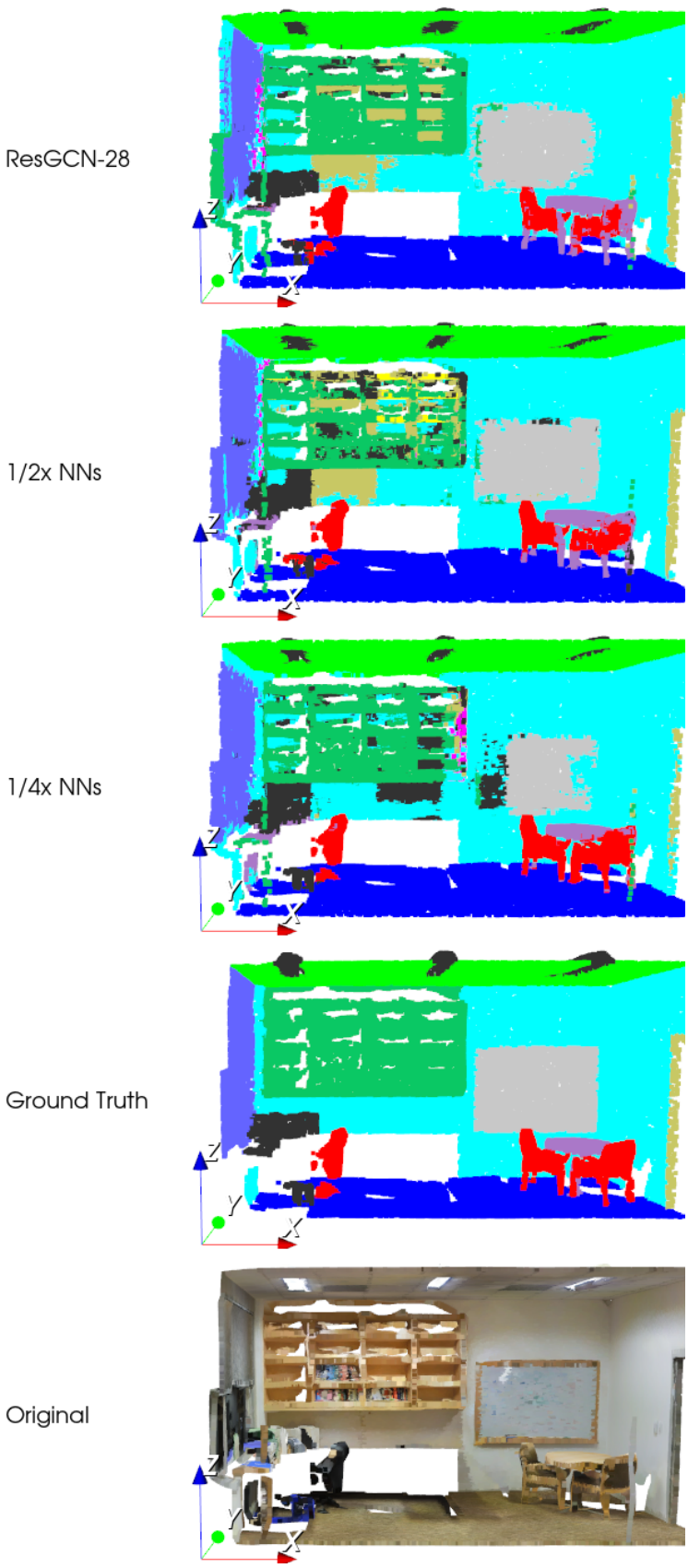

Figure 7. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of the number of nearest neighbors used in the convolutions.

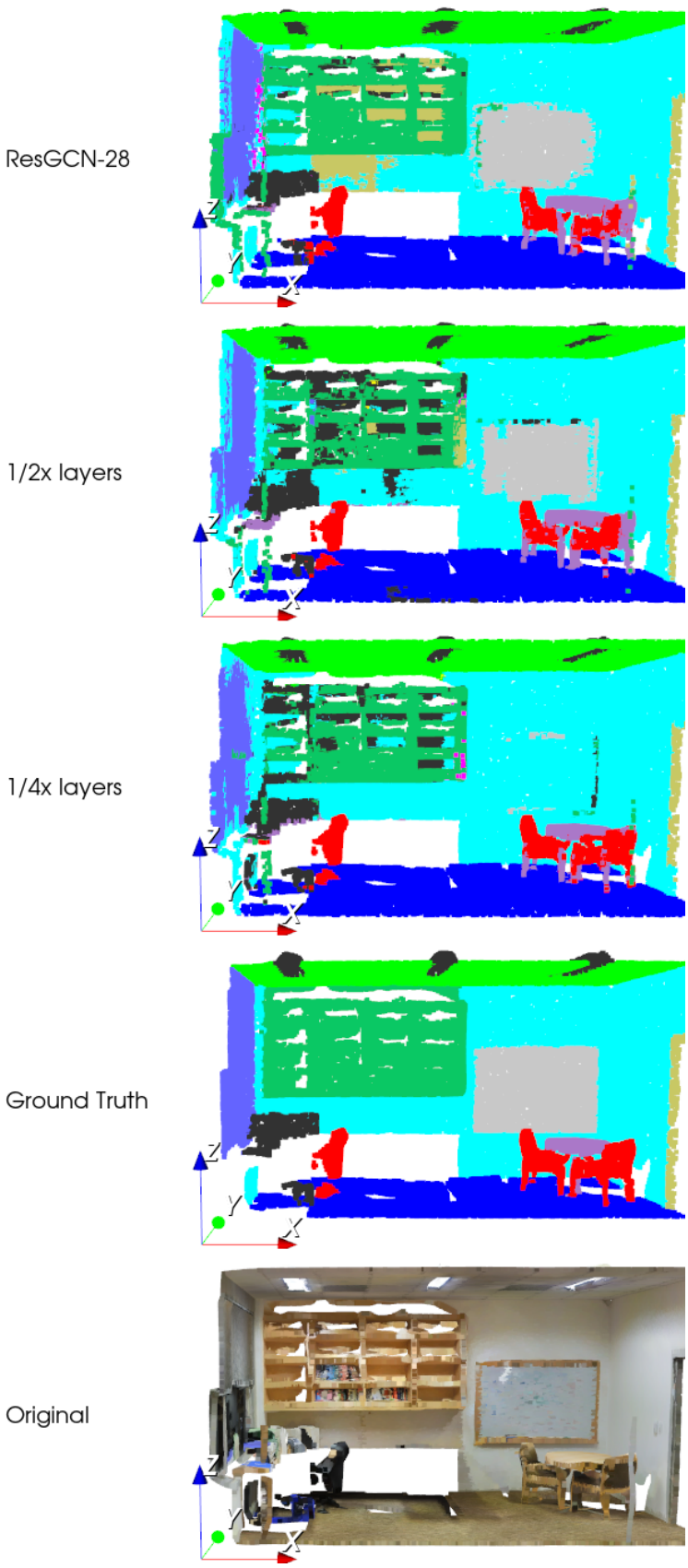

Figure 8. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of network depth (number of layers).

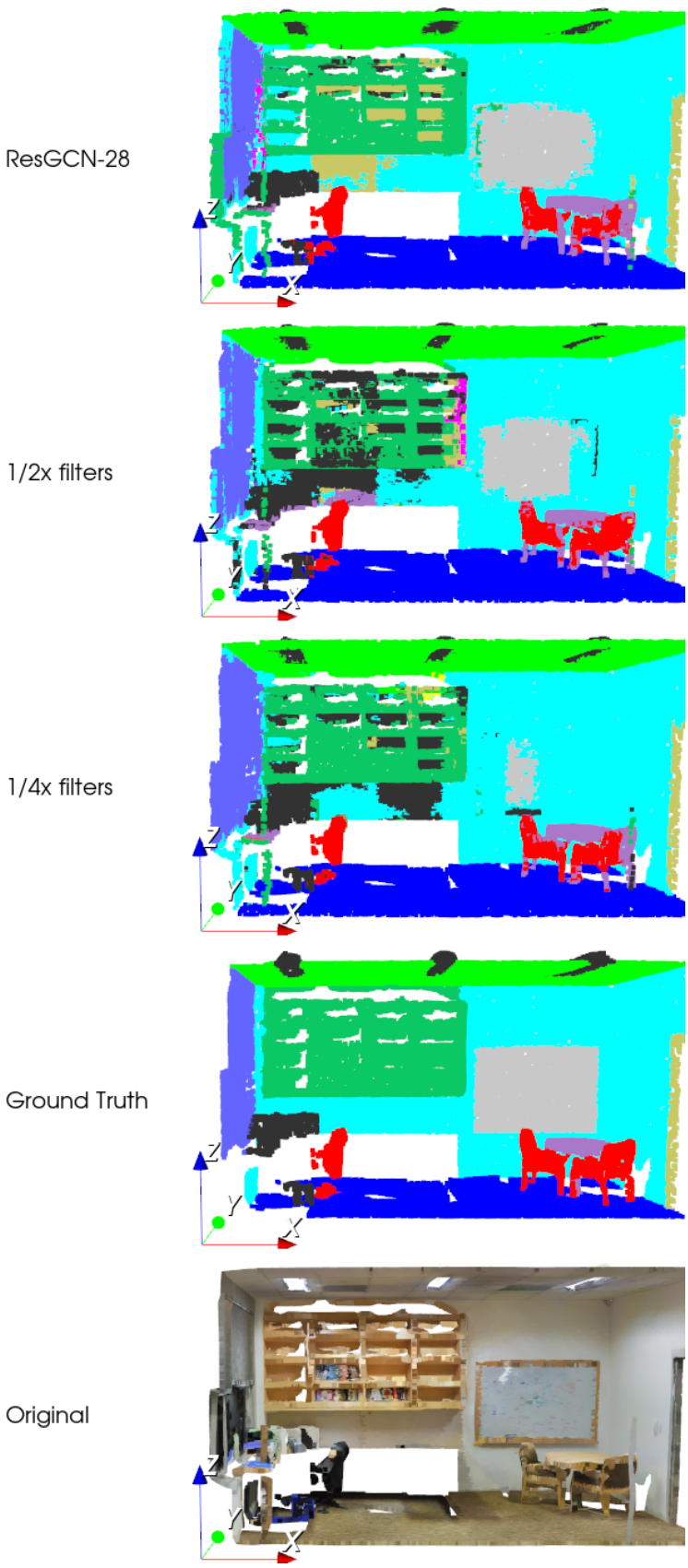

Figure 9. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of network width (number of filters per layer).

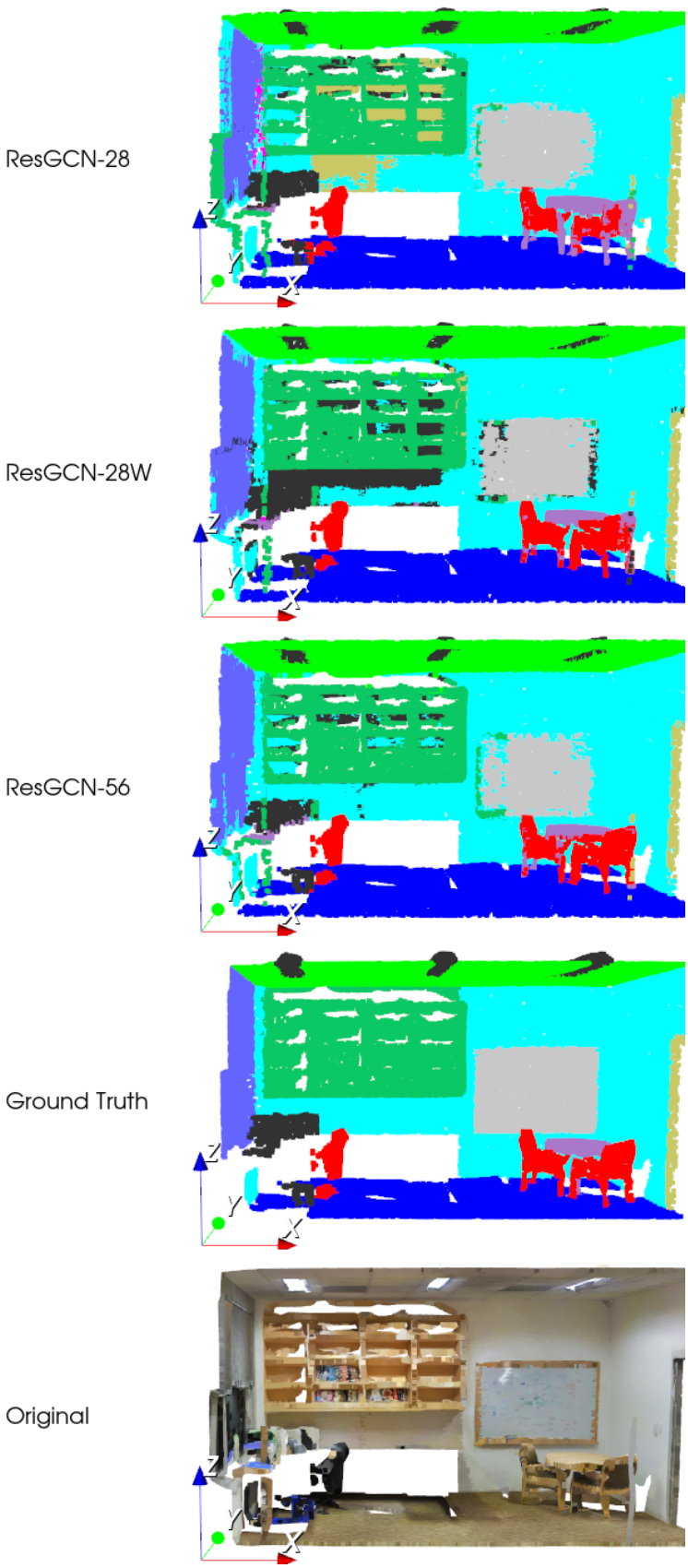

Figure 10. **Qualitative Results for S3DIS Semantic Segmentation**. We show the benefit of a wider and deeper network even with only half the number of nearest neighbors.