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 results in a wide variety of fields. Their success benefited from a massive boost with the ability to train very deep CNN models. Despite their positive results, 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, and borrow concepts from CNNs and apply them to train these models. GCNs show promising results, but they are limited to very shallow models due to the vanishing gradient problem (see Figure 1). As a result most state-of-the-art GCN algorithms are no deeper than 3 or 4 layers. In this work, we present new ways to successfully train very deep GCNs. We borrow concepts from CNNs, mainly residual/dense connections and dilated convolutions, and adapt them to GCN architectures. Through extensive experiments, we 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.

1. Introduction

GCNs are gaining fast momentum in the last few years. This 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 [37], model proteins for drug discovery [56, 41], enhance predictions of recommendation engines [25, 52], efficiently segment large point clouds [43], among other fields.

A key factor to the success of CNNs is the ability to design and train very deep models. In contrast, it is not yet clear how to properly train deep GCN architectures. Several works have studied the limiting factors of deep GCNs [20, 44, 55]. 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 vertices converging to the same value [20]. Due to these limitations, most state-of-the-art GCNs are no deeper than 4 layers [55].

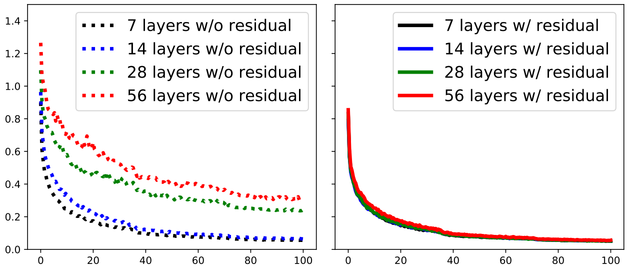

Figure 1. **Training Deep GCNs**. *Left:* We present here the training loss for GCNs with 7, 14, 28, 56 layers, with and without residual connections. We note how adding more layers without residual connections translates to substantially higher loss. *Right:* In contrast, training GCNs with residual connections results in consistent stability for all depths.

Vanishing gradients is not a foreign phenomenon in the world of CNNs. They too posed limitations on the depth growth of these kinds of networks. ResNet [12] provided a big step forward in the pursuit of very deep CNNs, since 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. An extension came with DenseNet [14], where connections are also introduced across layers. More layers could potentially mean more spatial information loss due to pooling. This issue was also addressed with Dilated Convolutions [53]. The introductions of these key concepts had a substantial effect in 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

[*] equal contribution

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 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 framework. **(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.

In what follows, we first review GCN algorithms in Section 2; we talk about their applications and the attempts done at training deeper GCN models, and also mention specific applications to point clouds. Section 3 presents the adaptation of residual/dense connections and dilated convolutions to GCNs. We show our experiments and results in Section 4. Finally, we draw conclusions in Section 5.

2. Related Work

A large number of real-world applications deal with non-Euclidean data. To overcome the shortcomings of CNNs, GCNs provide solutions for non-Euclidean data processing. These solutions have found an increasing interest in a variety of applications. In social networks [37], 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 nodes of social network graphs, thus leading to more accurate connections between individuals. Graphs are also used to model chemical molecule structures [56, 41]. Understanding the bio-activities of these molecules can have substantial impact on drug discovery. Another popular use of graphs is in recommendation engines [25, 52], where accurate modelling of user interactions leads to improved product recommendations. Graphs are also popular modes of representation in language processing [2, 24], to represent complex relations between words, sentences, and larger 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 [45, 49, 21]. Scene graphs also facilitate the inverse process, where an image is reconstructed given a graph representation of the scene [18]. Graphs are also used to model human joints for action recognition in video [48, 17].

GCNs are a perfect candidate for point cloud processing. Point cloud data poses a representational challenge, given its unstructured nature. Several attempts in creating structure from 3D data exist by either representing the 3D data with multiple 2D views [36, 10, 3, 23], or by voxelization [5, 30, 33, 38]. More recent work focuses on directly processing unordered point cloud representations [28, 31, 9, 15, 50]. The recent *EdgeConv* method by Wang *et al.* [43] 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.* train a semi-supervised GCN model for node classification [19]. The authors show how performance degrades when using more than 3 layers. Pham *et al.* [27] proposed Column Network (CLN) for collective classification in relational learning. The authors show peak performance with 10 layers; however, the performance degrades for deeper graphs. Rahimi *et al.* [32] developed a Highway GCN for user geolocation in social media graphs, where the authors add "highway" gates between layers to facilitate gradient flow. Even with these gates, authors show performance drop after 6 layers of depth are reached. Xu *et al.* [47] developed a *Jump Knowledge Network* for representation learning. The authors devise an alternative strategy to select neighbors for each node based on graph structure. As with other works, their network is limited to a small number of layers (6). To this end, Li *et al.* [20] studied the depth limitations of GCNs. They show that deep GCNs can cause over-smoothing, which results into features at vertices within each connected component converging to the same value. Other works [44, 55] also show the limitations of stacking multiple GCN layers, which lead to high complexity of back-propagation and the common vanishing gradient problem.

A lot of the difficulties that face GCNs nowadays (*e.g.* vanishing gradients, limited receptive field, *etc.*) were also present in the early days of CNNs [12, 53]. 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 [12]. By adding residual connections between inputs and outputs of layers, ResNet alleviates the vanishing gradient problem. DenseNet [14] takes this idea a step further and adds connections across layers as well. Dilated Convolutions [53] are a more recent approach that has lead to significant performance gains, specifically in image-to-image translation tasks such as semantic segmentation [53], by increasing the receptive field without loss of resolution. We show how we can benefit from these 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.* [43] to a much deeper GCN, and therefore significantly increasing its performance. Extensive experiments on the task of point cloud semantic segmentation validate that these ideas could be used in any general graph scenario.

3. Methodology

Graph Convolution Networks. Inspired by CNNs, GCNs intend to extract high level features by aggregating features from vertex neighborhoods. GCNs represent vertices by associating each vertex v with a feature vector $h_v \in \mathbb{R}^D$, where D is the feature dimension. Therefore, the graph $\mathcal{G}$ as a whole can be represented by concatenating the features of all the vertices, *i.e.* $h_{\mathcal{G}} = [h_{v_1}, h_{v_2}, ..., h_{v_N}]^\top \in \mathbb{R}^{N \times D}$, 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 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, which are the essential components of GCNs. In most GCN frameworks, aggregation functions are used to compile information from the neighborhood of vertices; update functions perform a non-linear transform on aggregated information to compute new representations of vertices. There are different variants of those two functions. For example, the aggregation function can be a mean aggregator [19], a max-pooling aggregator [29, 11, 43], an attention aggregator [40] or an LSTM aggregator [26]. The update function can be a multi-layer perceptron [11, 7], a gated network [22], *etc*. More concretely, the representation of vertices is computed at each layer by aggregating features of neighbor vertices for all $v_{l+1} \in \mathcal{V}_{l+1}$ as follows,

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

where ρ is a vertex feature aggregation function and ϕ is a vertex feature update function, h_{v_l} and $h_{v_{l+1}}$ are the vertex features at the l-th layer and $l+1$-th layer respectively. $\mathcal{N}(v_l)$ is the set of neighbor vertices of v at the l-th layer, 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. We use a simple max-pooling vertex feature aggregator, without learnable parameters, to aggregate the difference of features between the central vertex and all of its neighbors. The updater we used is a multi-layer perceptron (MLP) with batch normalization [16] and a ReLU as an activation function. We also do analyses for different GCN variants. (See **Appendix** A for more details.)

Dynamic Edges. As mentioned earlier, most GCNs only update the vertex features at each iteration. Recent works [35, 43, 39] show that dynamic graph convolution can learn better graph representations compared to GCNs with fixed graph structures. For instance, ECC (Edge-Conditioned Convolution) [35] uses dynamic edge-conditional filters to learn an edge-specific weight matrix. EdgeConv [43] finds the nearest neighbors in the feature space to reconstruct the graph after every EdgeConv layer. In order to learn to generate point clouds, Graph-Convolution GAN (Generative Adversarial Network) [39] also applies k-NN graphs to construct the neighbourhood for each vertex in every layer. We find that dynamically changing neighbors of GCNs helps to alleviate the over-smoothing problem and results in an effectively larger receptive field. In our framework, we propose to re-compute edges between vertices via a *Dilated* k*-NN* in the feature space at each layer to further increase the receptive field.

3.1. Residual Learning for GCNs

Designing deep GCN architectures [44, 55] is an open problem in the graph learning space. Recent works [20, 44, 55] suggest 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 no more than 3 layers deep [55]. Inspired by the huge success of ResNet [12], DenseNet [14] and Dilated Convolutions [53], we transfer those ideas to GCNs in order to unleash their full potential. This enables much deeper GCNs that converge well and achieve superior performance.

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. We propose a graph residual learning framework that learns a desired underlying mapping $\mathcal{H}$ by fitting another mapping

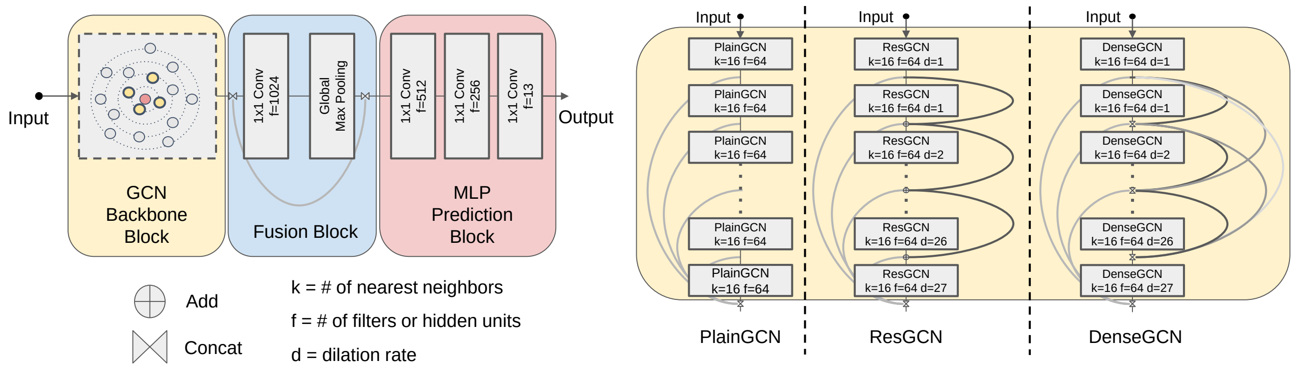

Figure 2. **Our GCNs Network architecture for point clouds semantic segmentation**. *Left:* Our framework consits of three blocks (one GCN Backbone Block, one Fusion Block and one MLP Prediction Block). *Right:* We mainly study three types of GCN Backbone Blocks *i.e*. *PlainGCN*, *ResGCN* and *DenseGCN*. There are two kinds of GCN skip connections - vertex-wise additions and vertex-wise concatenations. k is the number of nearest neighbors in GCN layers. f is the number of the filters or hidden units. d is the dilation rate.

$\mathcal{F}$. After $\mathcal{G}_l$ is transformed by mapping $\mathcal{F}$, vertex-wise addition is performed to obtain $\mathcal{G}_{l+1}$.

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

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*.

$$\mathcal{G}_{l+1}^{res} = \mathcal{F}(\mathcal{G}_l, \mathcal{W}_l) := \mathcal{G}_{l+1} - \mathcal{G}_l \tag{4}$$

3.2. Dense Connections in GCNs

DenseNet [14] proposes a more efficient way to improve information flow and reuse features between layers by dense connectivity. Inspired by DenseNet, we adapt a similar idea to GCNs so as to exploit information flow from different GCN layers. In particular, we have:

$$\begin{aligned} \mathcal{G}_{l+1} &= \mathcal{H}(\mathcal{G}_l, \mathcal{W}_l) \\ &= \mathcal{T}(\mathcal{F}(\mathcal{G}_l, \mathcal{W}_l), \mathcal{G}_l) \\ &= \mathcal{T}(\mathcal{F}(\mathcal{G}_l, \mathcal{W}_l), ..., \mathcal{F}(\mathcal{G}_0, \mathcal{W}_0), \mathcal{G}_0). \end{aligned} \tag{5}$$

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. 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 the features of each vertex of $\mathcal{G}_{l+1}$ is $D_0 + D \times (l + 1)$.

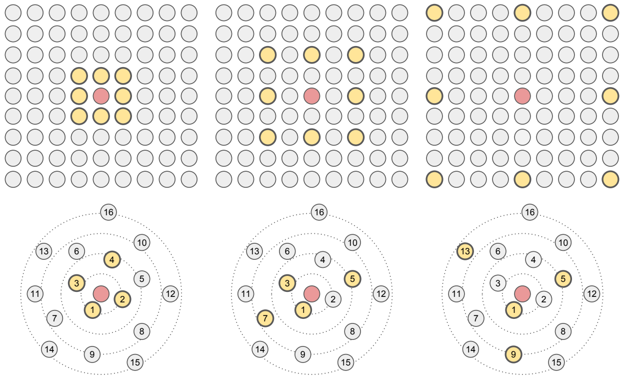

Figure 3. **Dilated Convolutions in GCNs**. Visualization of dilated convolution on a structured graph arranged in a grid (*e.g*. 2D image) and on a general structured graph. *Top:* 2D convolution with kernel size 3 and dilation rate 1, 2, 4 (left to right). *Bottom:* Dynamic graph convolution with dilation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

3.3. Dilated Aggregation in GCNs

Dilated wavelet convolution is an algorithm originating from the wavelet processing domain [13, 34]. To alleviate spatial information loss caused by pooling operations, Yu *et al*. [53] 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 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; we use the ℓ_2 distance in the feature space.

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.*

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

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 $u \in \mathcal{N}^{(d)}(v)$, where the GCN aggregation and update functions are applied (see Equation (1)) based on the edges $\mathcal{E}^{(d)}$ created by the *Dilated k-NN* to get the features of output vertices $h_v^{(d)}$ for all $v \in \mathcal{V}$. Denote the updated vertices as $\mathcal{V}^{(d)}$, then the output graph generated by the *dilated graph convolution* with dilation rate d is $\mathcal{G}^{(d)} = (\mathcal{V}^{(d)}, \mathcal{E}^{(d)})$. To gain better generalization, we use *stochastic dilation* in practice. Namely, at training time we perform dilated aggregations with a small probability ϵ to do a random aggregation by uniformly sampling k neighbors from $\{u_1, u_2, ..., u_{k \times d}\}$. At test time, we do regular dilated aggregation without stochasticity.

4. Experiments

We propose *ResGCN* and *DenseGCN* to handle the vanishing gradient problem of GCNs. In order to gain a large receptive field, we also define a *dilated graph convolution* for GCNs. To verify our framework, we conduct extensive experiments on the task of large-scale point cloud segmentation and demonstrate that our methods improve performance significantly. 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

Because of the unordered and irregular structure of point clouds, point cloud segmentation is a challenging task. Normally, each point in a point cloud is represented by its 3D coordinates and extra features such as color, surface normal and so on. 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 (Section 3). At the first layer, we construct an initial input graph $\mathcal{G}_0$ by executing a *dilated k-NN* search to find the nearest neighbor in the space of 3D coordinates. 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 investigate the practical usefulness of the components discussed in Section 3 by designing a deep GCN that incorporates all of them. We choose semantic segmentation of 3D point clouds as an application and show extensive results on the Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [1], which contains 3D point clouds of six areas from three different buildings. This dataset contains a total of 695,878,620 points across an area of 6,000m^2 and is annotated with thirteen semantic classes. As is common practice, we use the following evaluation method: we train a model on 5 of the 6 areas and then evaluate it on the area that was left out. We use the overall accuracy (OA) and mean intersection over union across all classes (mIoU) as evaluation metrics. The IoU per class 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 first 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 baseline model and other state-of-the-art methods, so as to motivate the use of deep GCNs. We then do a thorough ablation study on area 5 to analyse each component and present numerous insights.

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 that we add residual skip connections to all the GCN layers in *ResGCN*. They have the same number of parameters. We keep the fusion block and the MLP prediction block the same for all the architectures to compare fairly. For 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 output a learned graph representation with 4096 vertices. The fusion block and the MLP prediction block follow a similar architecture as PointNet [29] and EdgeConv [43]. 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, and a max pooling layer. The max pooling layer aggregates the vertex features of the whole graph to a global feature vector. Then the global feature vector is repeated and concatenated with all the vertex features from all previous GCN layers to fuse global and local information. The MLP prediction block takes as input the fused features and applies three MLP layers to predict the categories of all points. In practice, the MLP layers are implemented as 1×1 convolution layers.

PlainGCN. As shown in Figure 2, the *PlainGCN* has a GCN backbone block, a fusion block, and an MLP prediction block. Its GCN backbone block consists of 28 graph convolution network layers. The GCN layer we used is similar to the one of our baseline - EdgeConv [43]. We stack 28-layer GCNs without any skip connections between backbone layers.

ResGCN. We construct our *ResGCN* by adding *residual graph connections* to *PlainGCN*. We add such connections between all GCN layers in the GCN backbone block without increasing the number of parameters. We study 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). Detailed results are reported in Table 2. The dilation rate d of *dilated k-NN* is increased linearly as GCN layers go deeper (see Figure 2).

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

4.4. Implementation

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 3.3, we use a *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 architechtures, we do not use any data augmentation and post processing techniques. We train all our models end-to-end from scratch.

4.5. Results

Note that we focus on *residual graph connections* for our analysis since *ResGCN-28* is easier and faster to train.

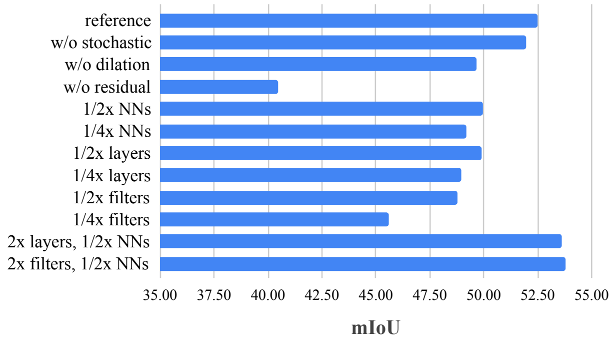

Figure 4. **Ablation study on area 5 of S3DIS**. We compare our reference network (*ResGCN-28*) with 28 layers, *residual graph connections* and *dilated graph convolutions* to several ablated variants. All models were trained for 100 epochs on all areas except for area 5 with the same hyper-parameters.

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 1. The results clearly show the effectiveness of deeper models with *residual graph connections* and *dilated graph convolutions*. Compared to DGCNN [43], which our *ResGCN-28* architecture builds upon, *ResGCN-28* gives a 7.0% (relative) or 3.9% (absolute) boost in performance in terms of mean IoU. 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%. The good performance on the difficult classes is probably due to the higher network capacity allowing the network to learn subtle details necessary to distinguish between a board and the wall for example. The first row in Figure 5 is a representative example for this occurrence. Our performance gains are solely due to our innovation in the network architecture; we use the same hyper-parameters and even learning rate schedule as DGCNN [43] 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 margin and expect further improvement from tweaking the hyper-parameters, especially the learning schedule.

We conduct an extensive ablation study to shed light on the contribution of each component of our novel network architectures. In total, we conduct 20 experiments and show all the results in Table 2. We also summarize the most important insights in Figure 4 and provide some further details about each block of experiments in the following.

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 dy-

Method	OA	mIOU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
PointNet [29]	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++ [31]	-	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 [51]	**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 [43]	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 1. **Comparison of *ResGCN-28* with state-of-the-art**. Average per-class results across all areas for our reference network with 28 layers, *residual graph connections* and *dilated graph convolutions* compared to state-of-the-art baselines. *ResGCN-28* beats state-of-the-art by almost 4%. Moreover, our network outperforms all baselines in 9 out of 13 classes. Metric shown is overall point accuracy (OA) and mean IoU (mIoU). '-' denotes not reported.

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
Dynamic k-NN		48.38	-4.11		⊕			16	64	28
		43.43	-9.06					16	64	28
Connections	*DenseGCN-28*	**51.96**	-0.53	✓	⋈	✓	✓	8	32	28
		47.92	-4.57	✓		✓	✓	16	64	7
		49.23	-3.26	✓		✓	✓	16	64	14
		40.47	-12.02	✓		✓	✓	16	64	28
		38.79	-13.70	✓		✓	✓	8	64	56
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 2. **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.

namic edges in Table 2 (*Dynamic k-NN*).

Residual graph connections. Our experiments (see Table 2: *Connections*) show that the *residual graph connections* are essential to train deeper networks. It is analogous to the insight for CNNs [12]. We expect that the connections result in more stable gradients. When the *residual graph connections* between the layers are removed the performance degrades dramatically (-12% mIoU).

Dense graph connections. We observe similar performance gains with *dense graph connections* (see Table 2: *Connections*). However, with a naive implementation the memory cost is prohibitive. Hence, the model we are able to fit into GPU memory uses only 32 filters and 8 nearest neighbors compared to 64 and 16 for the case of the residual counterpart. Since performance is similar, *residual connections* seem more practical for most use cases. Hence, we focus on *residual graph connections* in our ablation study but expect the insights to transfer to *dense graph connections*.

Dilation. Our results show that *dilated graph convolutions* account for about 3% in terms of mean IoU. Similar as in 2D convolutions, *dilated graph convolutions* increase the receptive field of the network (see Figure 3 for a visualization). We find that adding stochasticity to the *Dilated k-NN* helps a little bit but is not essential. Interestingly, our results in Table 4 (*Ablation*), also indicate that dilation helps especially for deep networks in combination with *residual graph connections*. Without *residual graph 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 without *residual graph connections*.

Nearest neighbors. Our experiments (see Table 2: *Neighbors*) show that in general a larger number of neighbors helps. 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 sufficient. This becomes apparent when we increase the number of neighbors by a factor of 2 while decreasing the number of filters by a factor of 2.

Network width. Our experiments (see Table 2: *Width*) show that increasing the number of filters improves network performance. This makes sense since the network has a higher capacity to learn nuances necessary for succeeding in the corner cases.

Network Depth. Table 2 (*Depth*) shows that increasing the number of layers leads to a similar increase in performance as increasing the number of filters. However, this only holds if *residual graph connections* and *dilated graph convolutions* are used as is apparent from Table 2 (*Connections*).

Qualitative results. We show qualitative results on S3DIS [1] area 5 in Figure 5. As expected from the results in Table 1, our *ResGCN* and *DenseGCN* perform particularly well on difficult classes such as board, bookcase, door and sofa. The first row very clearly shows how *ResGCN* is able to segment the board, while *PlainGCN* completely fails. Please refer to the **Appendix** for more qualitative results and other ablation studies.

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 performances on point cloud 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 a GCN with 151 layers 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 computational 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 of 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, data mining and so on. We show successful cases to adapt 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 [54], and so on. It will be also interesting to study different distance measures to compute dilated k-nn, constructing graphs using different k at each layer, better dilation rate schedules [4, 42] for GCNs, and combining residual and dense connections.

We would also like to point out that for the specific task of point cloud semantic segmentation the common approach of processing the data in $1m$ x $1m$ columns is sub-optimal for graph representation. A more suitable sampling approach should lead to further performance gains.

Acknowledgments. 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, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017. 2, 5, 8

[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. 7

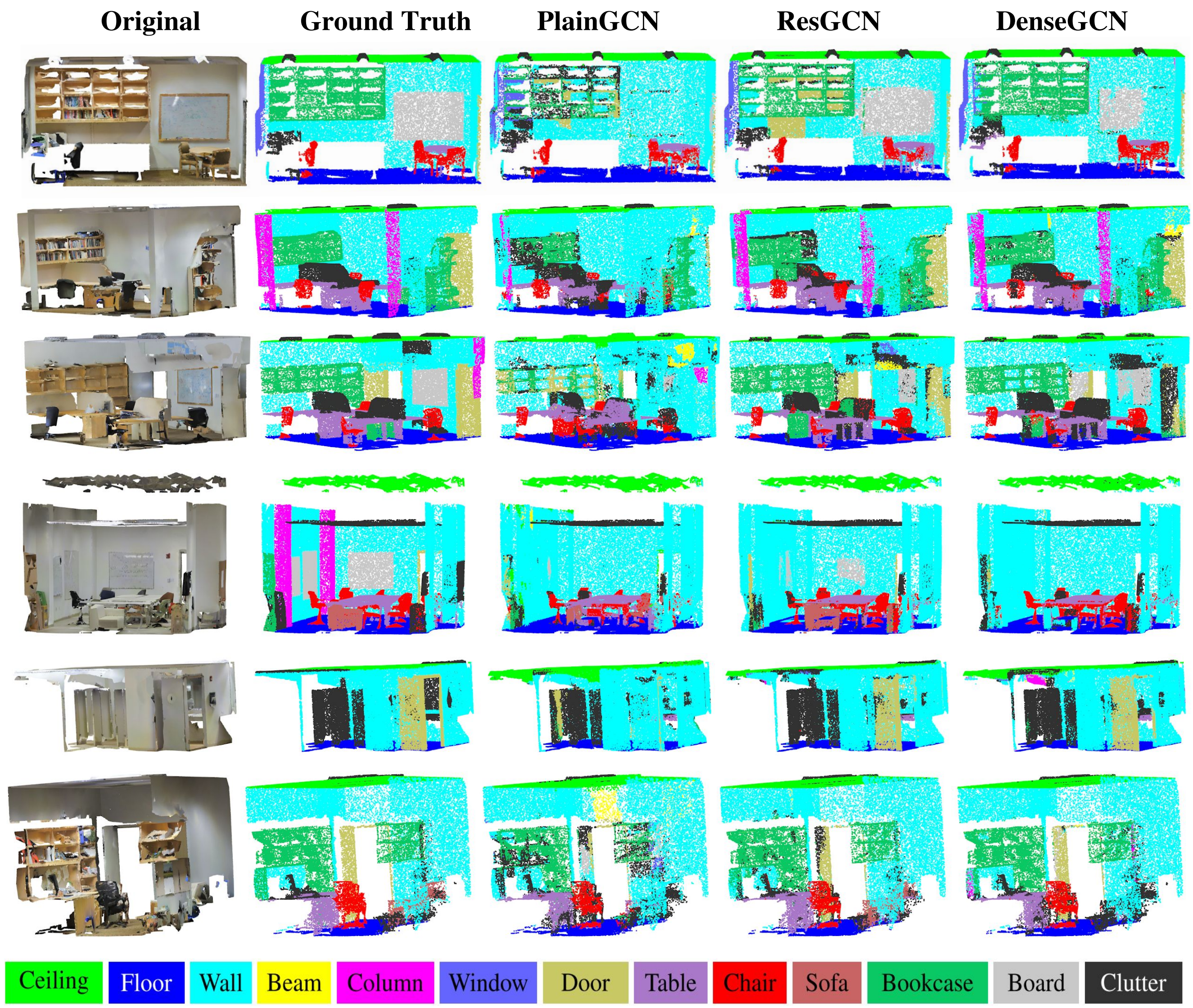

Figure 5. **Qualitative Results for S3DIS Semantic Segmentation**. We show here the effect of adding *residual and dense graph connections* to deep GCNs. *PlainGCN*, *ResGCN*, and *DenseGCN* are identical except for the presence of *residual graph connections* in *ResGCN* and *dense graph connections* in *DenseGCN*. We note how both *residual graph connections* and *dense graph connections* have a substantial effect for hard classes like board, bookcase, and sofa; these are lost in the results of *PlainGCN*.

[9] F. Engelmann, T. Kontogianni, A. Hermans, and B. Leibe. Exploring spatial context for 3d semantic segmentation of point clouds. feb 2018. 2

[10] 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

[11] 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, 12

[12] 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, 3, 7

[13] 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

[14] 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

[15] 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

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

[17] 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

[18] 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

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

[20] 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

[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)*, pages 335–351, 2018. 2

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

[23] 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

[24] 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

[25] 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

[26] 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

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

[28] 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

[29] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 3, 5, 7

[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*, pages 5648–5656, 2016. 2

[31] 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, 7

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

[33] 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

[34] 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

[35] 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

[36] 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

[37] 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

[38] 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

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

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

[41] 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

[42] 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

[43] 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, 12

[44] 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

[45] 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

[46] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 12, 13

[47] 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

[48] 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

[49] 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

[50] 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

[51] X. Ye, J. Li, H. Huang, L. Du, and X. Zhang. 3d recurrent neural networks with context fusion for point cloud semantic segmentation. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, pages 415–430, Cham, 2018. Springer International Publishing. 7

[52] 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

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

[54] 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

[55] 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

[56] 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 [43] 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 [11], Graph Isomorphism Network (GIN) [46] 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, connection, dilation stochastic) 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 the *ResGCN* in our paper. We refer as *ResEdgeConv* to distinguish it from other GCN operations.

ResEdgeConv. Instead of aggregating neighborhood features directly, EdgeConv [43] 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:

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

ResGraphSAGE. GraphSAGE [11] 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*:

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

In the original GraphSAGE paper, the vertex feature are normalized after aggregation. We implement two variants, one without normalization (see Equation (7)), the other one with normalization $h^{res}_{v_{l+1}} = h^{res}_{v_{l+1}} / \left\| h^{res}_{v_{l+1}} \right\|_2$.

ResGIN. The main difference between GIN [46] 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:

$$\begin{aligned} h^{res}_{v_{l+1}} &= \mathit{mlp}\left((1+\epsilon) \cdot h_{v_l} + \mathit{sum}(\{h_{u_l} | u_l \in \mathcal{N}^{(d)}(v_l)\})\right), \\ h_{v_{l+1}} &= h^{res}_{v_{l+1}} + h_{v_l}. \end{aligned} \tag{8}$$

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

$$\begin{aligned} h^{res}_{\mathcal{N}^{(d)}(v_l)} &= \mathit{max}\left(\{h_{u_l} - h_{v_l} | u_l \in \mathcal{N}^{(d)}(v_l)\}\right), \\ h^{res}_{v_{l+1}} &= \mathit{mlp}\left(\mathit{concat}\left(h_{v_l}, h^{res}_{\mathcal{N}^{(d)}(v_l)}\right)\right), \\ h_{v_{l+1}} &= h^{res}_{v_{l+1}} + h_{v_l}. \end{aligned} \tag{9}$$

Where $h_{v_{l+1}}$ and h_{v_l} are the hidden state of vertex v at $l+1$; $h^{res}_{v_{l+1}}$ is the hidden state of the residual graph. All the *mlp* (multilayer perceptron) functions use a ReLU as activation function; all the *max*, *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 GCNs 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 well during the training phase and has a high training accuracy. However, it fails to generalize to the test set. This

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

phenomenon is also observed in the original paper [46] 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. skip 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

Figures 6, 7, 8, 9, 10 show qualitative results for the ablation study presented in the paper.

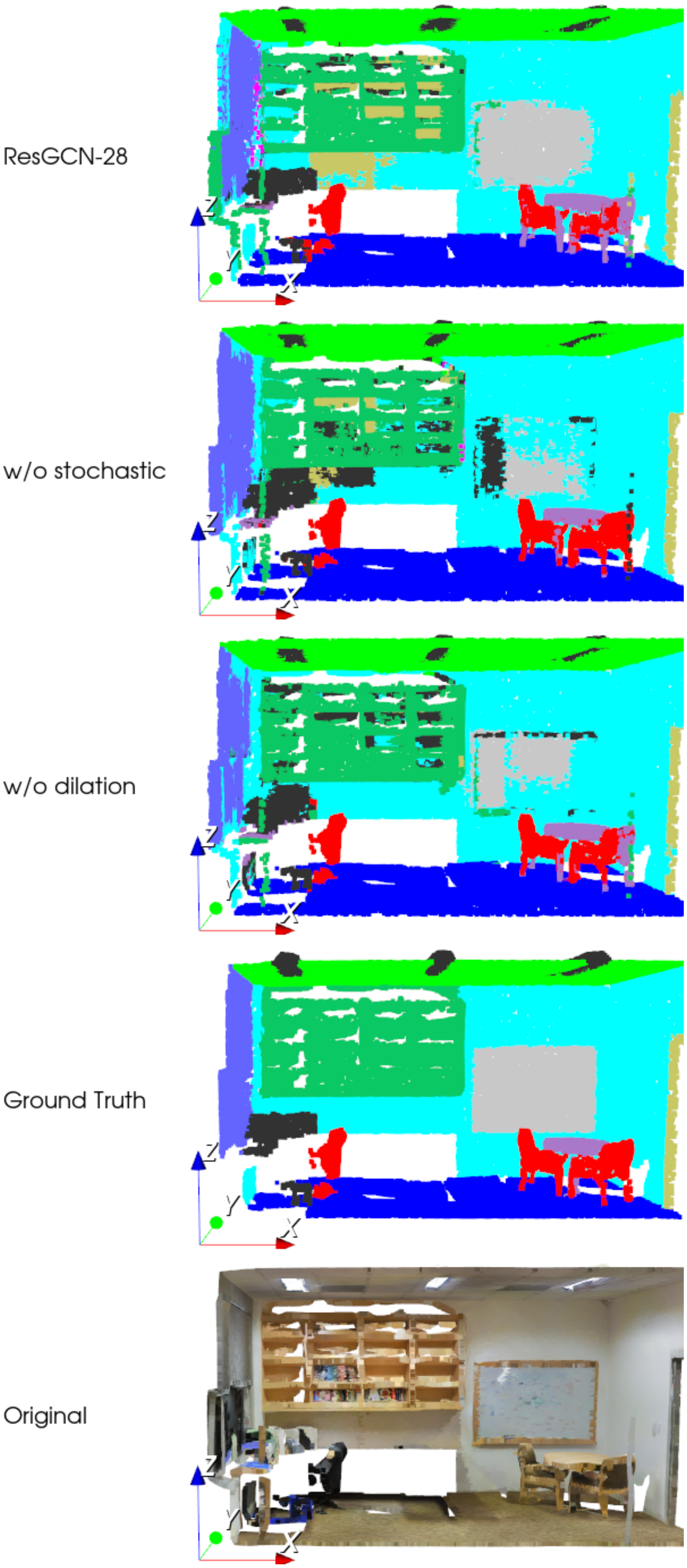

Figure 6. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of stochastic dilated convolutions.

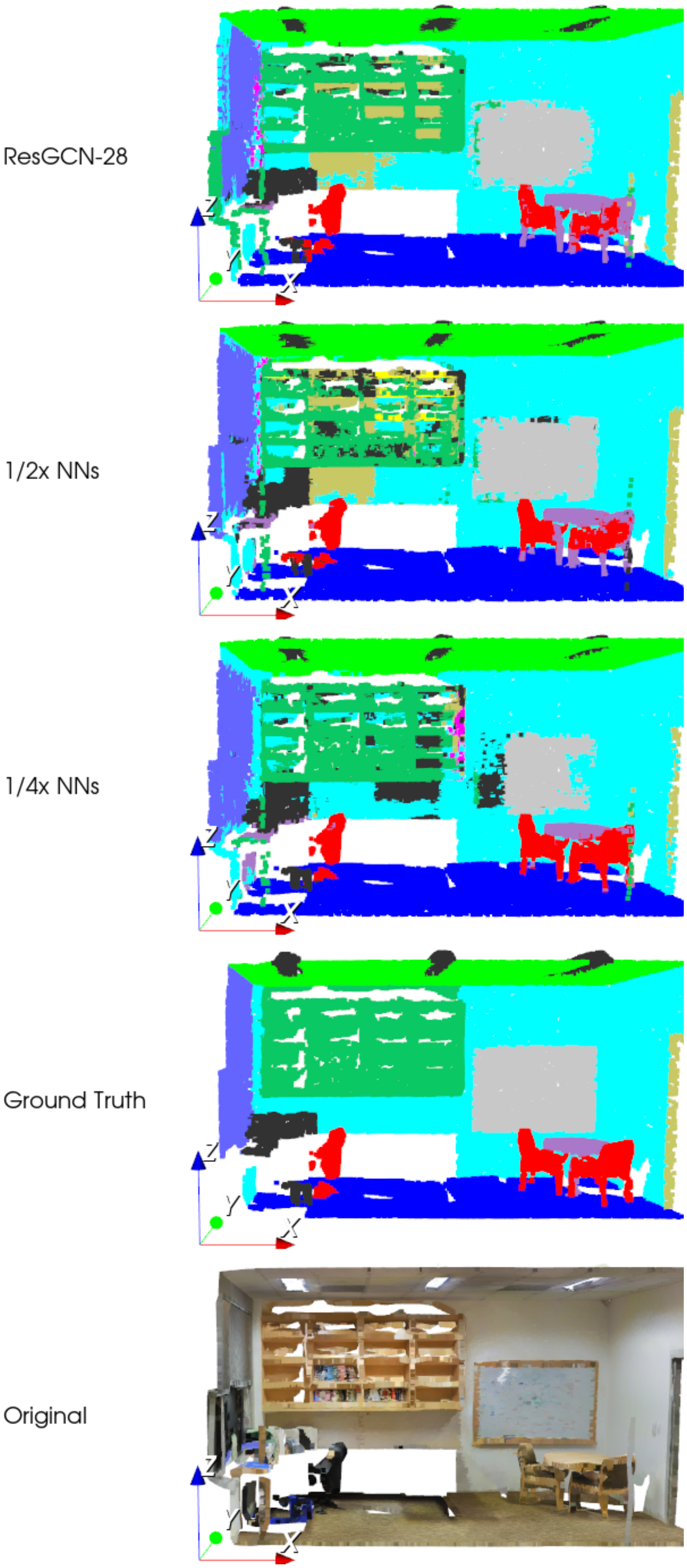

Figure 7. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of the number of nearest neighbors used in the convolutions.

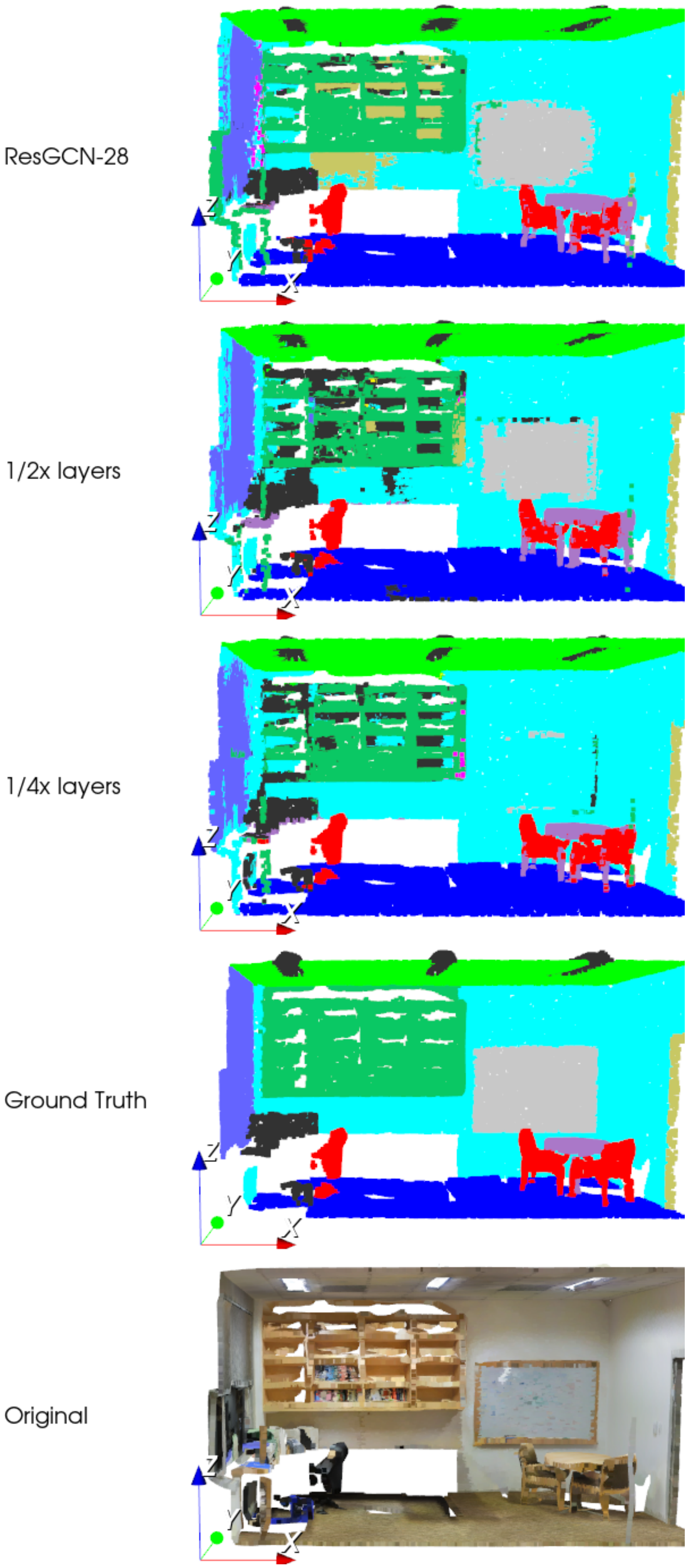

Figure 8. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of network depth (number of layers).

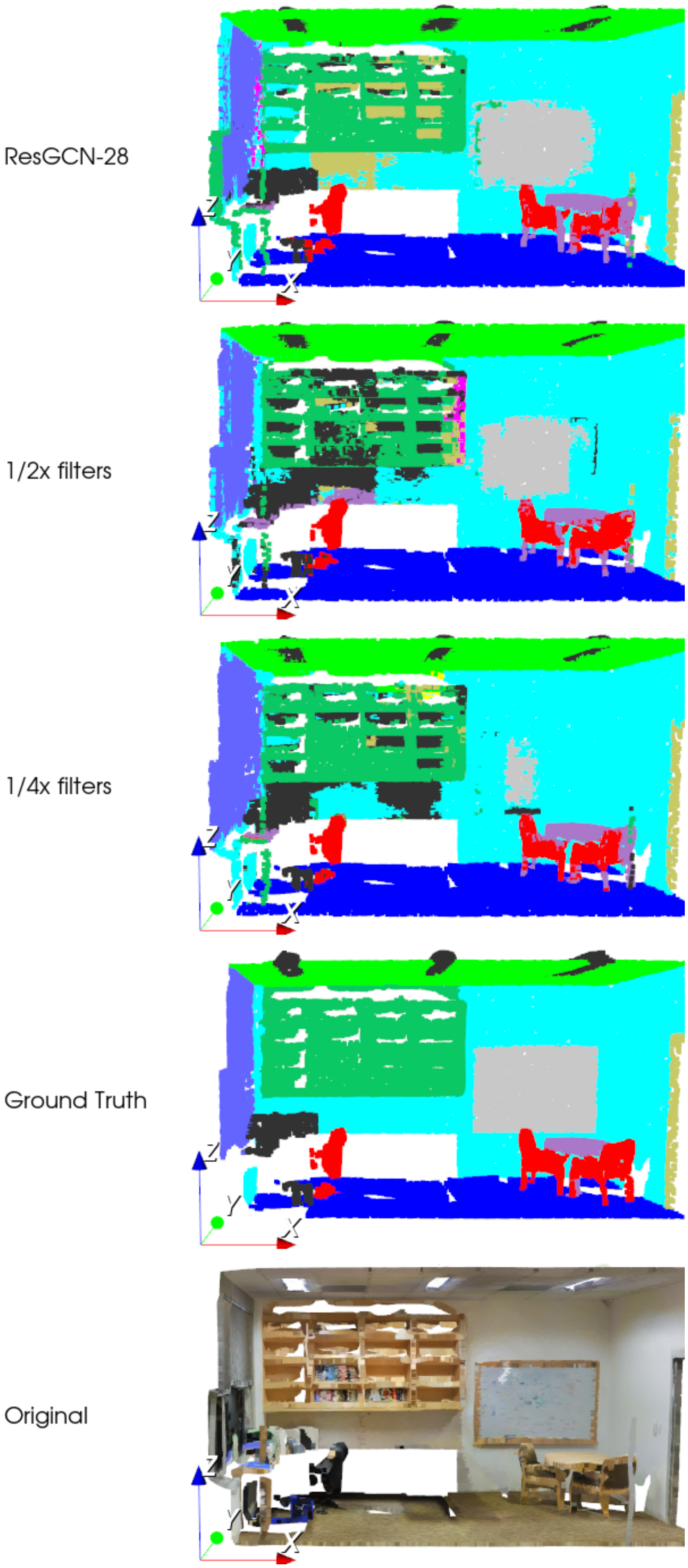

Figure 9. **Qualitative Results for S3DIS Semantic Segmentation**. We show the importance of network width (number of filters per layer).

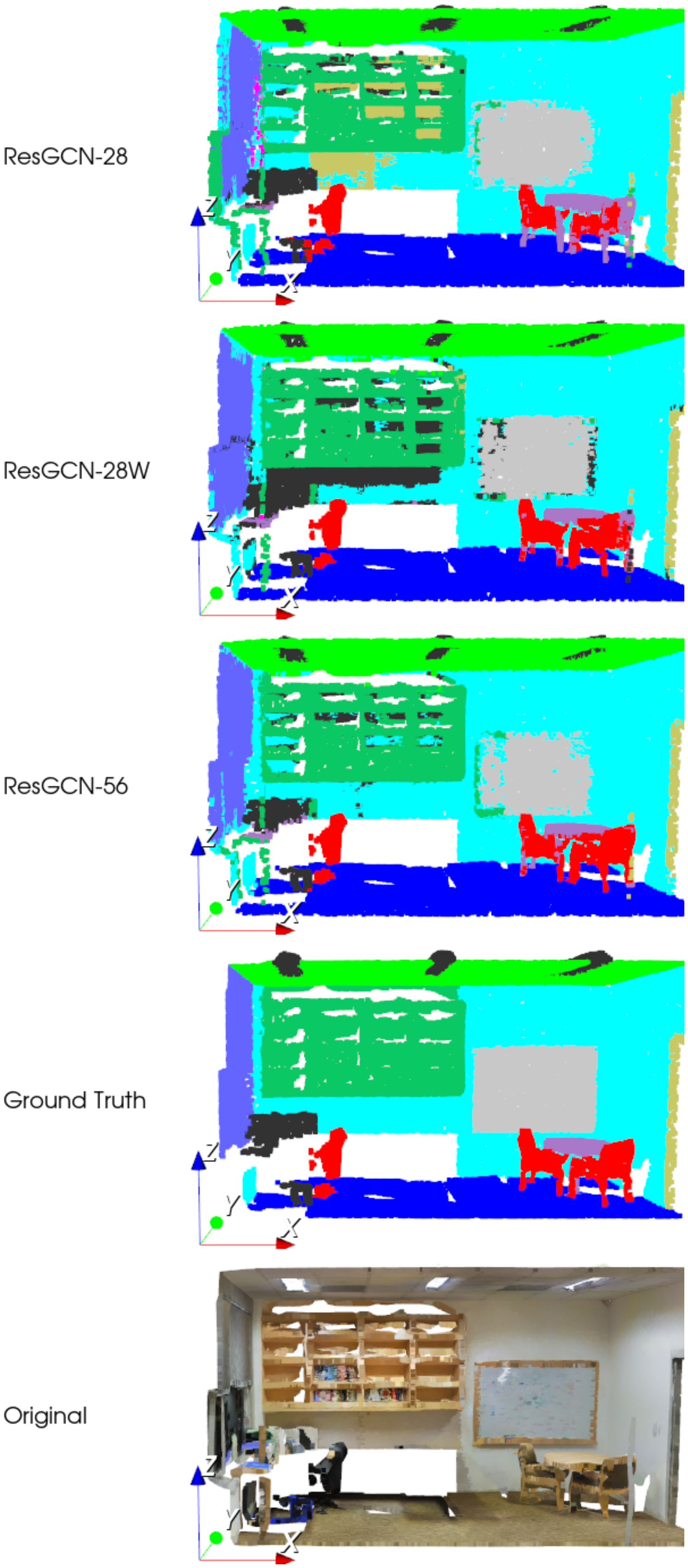

Figure 10. **Qualitative Results for S3DIS Semantic Segmentation**. We show the benefit of a wider and deeper network even with only half the number of nearest neighbors.