Applications of Graph Convolutional Networks and DeepGCNs in Point Cloud Part Segmentation and Upsampling

Thesis by
Abdulellah H. Abualshour

In Partial Fulfillment of the Requirements
For the Degree of
Masters of Science
in Computer Science

King Abdullah University of Science and Technology
Thuwal, Kingdom of Saudi Arabia

April, 2020
EXAMINATION COMMITTEE PAGE

The thesis of Abdulellah H. Abualshour is approved by the examination committee

Committee Chairperson: Bernard Ghanem
Committee Members: Peter Wonka and Markus Hadwiger
Applications of Graph Convolutional Networks and DeepGCNs in Point Cloud Part Segmentation and Upsampling

Abdulellah H. Abualshour

Graph convolutional networks (GCNs) showed promising results in learning from point cloud data. Applications of GCNs include point cloud classification, point cloud segmentation, point cloud upsampling, and more. Recently, the introduction of Deep Graph Convolutional Networks (DeepGCNs) allowed GCNs to go deeper, and thus resulted in better graph learning while avoiding the vanishing gradient problem in GCNs. By adapting impactful methods from convolutional neural networks (CNNs) such as residual connections, dense connections, and dilated convolutions, DeepGCNs allowed GCNs to learn better from non-Euclidean data. In addition, deep learning methods proved very effective in the task of point cloud upsampling. Unlike traditional optimization-based methods, deep learning-based methods to point cloud upsampling does not rely on priors nor hand-crafted features to learn how to upsample point clouds. In this thesis, I discuss the impact and show the performance results of DeepGCNs in the task of point cloud part segmentation on PartNet dataset. I also illustrate the significance of using GCNs as upsampling modules in the task of point cloud upsampling by introducing two novel upsampling modules: Multi-branch GCN and Clone GCN. I show quantitatively and qualitatively the performance results of our novel and versatile upsampling modules when evaluated on a new proposed standardized dataset: PU600, which is the largest and most diverse point cloud upsampling dataset currently in the literature.
ACKNOWLEDGEMENTS

I thank Professor Bernard Ghanem and Professor Peter Wonka for their supervision and mentorship over my studies at King Abdullah University of Science and Technology (KAUST). Their continued support proved invaluable throughout my research and academic career, and I deeply appreciate it.

I also thank my colleagues in KAUST’s Image and Video Understanding Lab (IVUL) and KAUST’s Visual Computing Center (VCC) for their help and support in research, especially Gordon Qian, Ali Thabet, Guohao Li, Motasim Alfarra, Silvio Giancola, Modar Alfadly and Wamiq Reyaz.

Additionally, I express my gratitude to Prof. Bernard Ghanem, Prof. Peter Wonka, and Prof. Markus Hadwiger for taking the time to attend and evaluate my thesis defense.

I thank KAUST Gifted Student Program (KGSP) for their continued and ongoing support that started when I first left for my undergraduate degree in the United States. KGSP and its management helped, mentored, and guided my academic career from the beginning, and I will forever be proud to be part of the program.

Finally, I would like to thank my mother Mona, my step-father Mahmoud and my only brother Abdullah for their encouragement and support throughout every step I decide to take in my academic career. I cannot imagine myself reaching and achieving any of my goals and dreams without them standing by my side. I dedicate this work to my family, and I aspire to make them proud in the next milestone and all the milestones to come.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Concluding Remarks</td>
<td>33</td>
</tr>
<tr>
<td>References</td>
<td>34</td>
</tr>
<tr>
<td>Appendices</td>
<td>40</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.1 **DeepGCN part segmentation architecture.** DeepGCNs for part segmentation constitute a GCN Backbone, a Fusion Block, and an MLP Prediction Block. The GCN Backbone Block uses ResGCN which utilizes residual connections. 21

2.2 **Qualitative Results on PartNet Part Segmentation.** We show here the effect of adding residual connections to deep GCNs. PlainGCN-28 and ResGCN-28 are identical except for the presence of residual connections in ResGCN-28. We note how residual connections have a positive effect on part segmentation compared to PlainGCN-28. Many important parts of the objects are classified incorrectly using PlainGCN-28. 24

3.1 **The proposed upsampling modules.** (a) Multi-branch GCN: We apply \( r \) different GCN layers to the input nodes. The outputs from each GCN are concatenated node-wise. (b) Clone GCN: We pass the input through \( r \) GCN layers with shared weights, and concatenate their outputs. \( \gg \) denotes node-wise concatenation. 26

3.2 **Qualitative results of upsampling \((\times 4)\) \((1/3)\).** We show the upsampled point clouds of input (a) when processed using state-of-the-art methods (PU-Net and 3PU) while replacing their upsampling modules with our proposed Clone GCN and Multi-branch GCN. (a) Input point cloud. (b) PU-Net upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (c) 3PU upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (d) Ground Truth. 30

3.3 **Qualitative results of upsampling \((\times 4)\) \((2/3)\).** (a) Input point cloud. (b) PU-Net upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (c) 3PU upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (d) Ground Truth. 31
3.4 Qualitative results of upsampling (×4) (3/3). (a) Input point cloud. (b) PU-Net upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (c) 3PU upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (d) Ground Truth.

A.1 Training samples from PU600. ........................................ 41
A.2 Testing samples from PU600. .......................................... 42
A.3 Training samples from PU-GAN’s Dataset. ......................... 43
A.4 Testing samples from PU-GAN’s Dataset. .......................... 43
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of ResGCN-28 with other methods on PartNet Part Segmentation for the categories: bed, bottle, chair, clock, dishwasher, display, door, earphone, faucet, and knife. We report the part-category mean IoU (mIoU) on the fine-grained level of segmentation.</td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison of ResGCN-28 with other methods on PartNet Part Segmentation for categories: lamp, microphone, fridge, storage furniture, table, trash can, and vase. We report the part-category mean IoU (mIoU) on the fine-grained level of segmentation. We also report the average mIOU for all categories including the ones from Table 2.1</td>
</tr>
<tr>
<td>3.1</td>
<td>Ablation study on upsampling modules. Results show that our upsampling modules (Multi-Branch GCN and Clone GCN) can transfer well to different upsampling architectures in the literature. Replacing the original upsampling module with one of the proposed ones improves upsampling performance overall. <strong>Bold</strong> denotes the best performance for each architecture.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Recently, GCNs have been a topic of interest for research in the field of graph processing. By extending convolutional neural networks (CNNs) to the non-euclidean data domain, GCNs are considered a powerful tool in learning from graph and point cloud data. While CNNs illustrated great performance on grid structured data (e.g. images), their performance on non-euclidean data (e.g. point clouds, graphs) is not measuring up to the standards of performance.

Non-euclidean data is everywhere, and GCNs proved to be effective in many real-world applications such as predicting social network relations \[1\], predicting multicellular function in proteins \[2\], improving recommender systems \[3\], and point cloud segmentation \[4\]. GCNs applied in these domains illustrated an improvement in performance. However, they are limited by their shallow and simplistic design. What makes CNNs very successful in computer vision is their continued ability to achieve state-of-the-art performances by training very deep and non-trivially designed networks. On the other hand, how to train deep GCNs is not very clear in the literature and was investigated by recent research \[5\] \[6\] \[7\]. The main disadvantage of going deeper using GCNs is the vanishing gradient problem, which is similar to what happens using CNNs. Another problem that faces deeper GCNs is the over-smoothing problem, in which the features of the graph vertices converge to the same value. These disadvantages proved to affect the performance of GCNs negatively when going deeper, and limit the networks to be very shallow with a usual maximum of 4 layers \[5\].
The vanishing gradient problem is an active and important topic of research in CNNs. It was the main limitation of CNNs before the introduction of ResNet [8] where the authors introduced a solution to the problem. Residual connections between layers alleviated the vanishing gradient problem by introducing additional connections for the flow in the network. This enabled ResNets deeper than a hundred layers to be trained effectively, while alleviating the vanishing gradient problem. Later on, DenseNet [9] introduced another good strategy in CNNs where every layer was connected to all other layers in a feed-forward fashion. Another problem arises with deeper network architectures on tasks like object detection and semantic segmentation. By increasing the number of layers in the network, more information can be lost throughout the pipeline due to pooling. To tackle this problem and aggregate as much spatial information as possible, we can increase the receptive field the deeper we go in a network while minimizing the loss of spatial information. Dilated convolutions [10] were introduced to provide a reliable way of aggregating spatial information and increasing the receptive field in CNNs. The convolutions applied aggregate more distant neighbors with the increased depth. All these problems were tackled by innovative solutions in the domain of CNNs, and the question is, can we extend these best practices from CNNs to GCNs?

The recent work of DeepGCNs [11, 12], which I contributed to, introduces a detailed and comprehensive study of strategies that allow deeper GCNs to be trained effectively. DeepGCNs adapt the successful strategies of the CNN domain into the GCN domain: residual connections, dense connections, and dilated convolutions. DeepGCNs show how these strategies can be applied on non-euclidean data using graph frameworks, and detailed analysis of DeepGCNs proves their effectiveness [11]. In this thesis, I discuss and show the effect of using DeepGCNs, namely the architecture of ResGCN which utilizes residual connections between GCN layers, in the tasks of point cloud part segmentation on PartNet [13] and point cloud upsampling
1.1 Non-Euclidean Data

CNNs are not that effective when considering many real-world applications that deal with non-Euclidean data. GCNs were innovated to provide a reliable tool to learn from non-Euclidean data that can form graph structures. Edges between nodes in social network graphs indicate the connections between persons on mutual interests and relationships. Given these edges, one can see that they are non-Euclidean and irregular. Using GCNs, we can learn from such edge structures to better estimate the importance of some edges over others in social network graphs. This results in a better understanding of connections between individuals. Graphs are also useful in predicting multi-cellular function of proteins. Understanding the molecular structure of proteins can be impactful for the drug industry. One important use of graphs is in recommender systems, where user interactions need to be accurately modelled to improve product recommendations.

1.2 Computer Vision and GCN Architecture Depth

Meaningful relations between objects in a given image can be detected and modelled in the task of scene graph generation. GCNs proved useful in this task. Given an image as input, one can generate a scene graph that models the semantic relations between the different objects contained in the input image. Scene graphs can also be used to reconstruct an image when used as input. In addition, graphs are used in video action recognition by modeling body structures as graphs. GCNs can be utilized in various tasks in 3D point cloud processing. Given the unstructured nature of 3D point clouds, their representation imposed a challenging topic of research. Many works showed different ways...
of representing point clouds. Some use voxelization \cite{22, 23, 24, 25}, and others generate the structure from multiple 2D perspectives \cite{26, 27, 28, 29}. By building a graph in point clouds using an algorithm like $k$-NN, one can utilize point cloud local and global neighborhood information for point cloud processing tasks including point cloud classification, segmentation, and upsampling. Recent work \cite{30, 31, 32, 33, 34} processes order-invariant point clouds. For example, the $EdgeConv$ method introduced by Wang \textit{et al.} \cite{4} uses GCNs to process point clouds and dynamically build neighborhoods of points using $k$-NN. To be more specific, their dynamic edge convolution operator for semantic segmentation of point clouds proved very effective. It illustrates the effectiveness of GCNs for point cloud applications, and outperforms state-of-the-art results in many point cloud tasks. $EdgeConv$ does not rely on complicated architectures for point aggregations such as RNNs.

\textbf{GCN Architecture Depth.} Shalowness of the architecture is one of issues that face GCNs, including $EdgeConv$. The work of Kipf \textit{et al.} shows that when training semi-supervised GCN models for node classification the performance goes down \cite{35}, specifically when going deeper than 3 layers. The Highway GCN introduced by Rahimi \textit{et al.} \cite{36} utilized user geo-location in social media graphs. By adding “highway” gates between layers, it improves gradient flow but is still limited to 6-layer deep networks. \textit{Jump Knowledge Network} introduced by Xu \textit{et al.} \cite{37} innovated an alternative strategy to select graph neighbors for representation learning depending on graph structure. However, their network also reaches a depth of 6-layers only. The work of Pham \textit{et al.} \cite{38} showed degradation of performance when going deeper than 10 layers as well in their proposed Column Network (CLN) for collective classification. Li \textit{et al.} \cite{5} studied this limitation of GCNs and showed that deep GCNs can result in over-smoothing, which causes the features at individual nodes within connected components to converge to the same value. The vanishing gradient problem also arises with deeper networks as discussed in other works \cite{7, 6}.
Problems such as the vanishing gradient problem and the limited receptive field were also faced by CNNs [8, 10]. DeepGCNs bridge the gap between GCNs and CNNs and show that the majority of these problems can be alleviated by using successful strategies from the CNN domain. ResNet [8] boosted the performance of deep CNNs by adding residual connections between inputs and outputs of layers, resulting in alleviating the vanishing gradient problem. DenseNet [9] introduces a new type of connection, in which each layer is connected to all other layers in a feed-forward fashion. Also, Dilated Convolutions introduced by Yu et al. [10] lead to great performance gains in image-to-image translation tasks such as semantic segmentation [10] by increasing the receptive field with no loss of resolution. DeepGCNs show how the successful strategies originally introduced to CNNs can be implemented in GCNs and help go deeper using GCNs while alleviating the vanishing gradient problem and limited receptive field. DeepGCNs extend several GCN variants to deeper versions by adapting these successful strategies from the CNN domain, and show significant performance gains. In this thesis, I will focus on the application of DeepGCNs in the task of point cloud part segmentation, showing how DeepGCNs can outperform state-of-the-art in this task.

1.3 Point Cloud Upsampling

Optimization-based approaches to upsampling proved very effective in the early works [39, 40, 41, 41, 42]. However, all these methods essentially rely on human hand-crafted priors.

**Deep learning-based upsampling methods.** The data-driven nature of deep learning-based approaches proved very effective in recent works. Works like PointNet [43], PointNet++ [44], DGCNN [45], KPConv [46] allowed learning from point clouds using neural networks and proved effective in point cloud processing tasks. PU-Net was introduced by Yu et al. [47] to learn multi-scale features per point based
on the PointNet architecture, and it expands the point set in feature space using CNNs. However, it also contains a downsampling step to obtain and learn multi-scale features, which results in resolution loss. EC-Net was also introduced by Yu et al. [48], an edge-aware network for point set consolidation. By using an edge-aware joint loss, it encourages the network to learn to consolidate points for edges. Nevertheless, edge-notation for the training data is very expensive and inefficient computationally. 3PU was introduced by Wang et al. [49] and proposed a progressive network that upsamples the input point patches in several steps using a duplicate-based approach. Unlike PU-Net, 3PU learns multi-scale features with no resolution loss. However, it is progressive and computationally inefficient due to its progressive nature. Li et al. [50] recently introduced PU-GAN, a Generative Adversarial Network (GAN) designed for learning upsampled point distributions in point clouds from latent space. They first upsample more points than the required target (e.g. \( \times 6 \)) using their GAN architecture and then use farthest upsampling to sample the target number of points (e.g. \( \times 4 \)) to produce the final output. However, the inherent GAN architecture is unstable and is hard to reproduce their results using the code made available by the authors. PU-GAN also does not focus on designing a better upsampling module. In the proposed PU-GCN [14], upsampling is improved using novel GCN upsampling modules to capture local point information, and by encoding multi-scale features using a proposed Inception DenseGCN. In my thesis, I will focus on two of these upsampling modules in addition to their qualitative and quantitative performance on a new standardized dataset: PU600, which is the largest and most diverse point cloud upsampling dataset to this date.

1.4 Contributions

I experiment with DeepGCNs [11], namely ResGCN, in the task of part segmentation on the recently proposed PartNet dataset [13]. I show quantitative and qualitative
results of these two DeepGCN architecture and discuss the performance and illustrate the superiority of DeepGCNs over state-of-the-art methods.

In addition, I contribute to the work of PU-GCN [14] by compiling and presenting a new standardized benchmark dataset for point cloud upsampling named PU600, and show the quantitative and qualitative performance of our proposed upsampling modules (Multi-branch GCN and Clone GCN) on this dataset.

**Disclaimer.** The methodology mentioned in Section 2.1.2 is introduced by the original ICCV 2019 paper by Li et al. [11].
Chapter 2

Part Segmentation using DeepGCNs

2.1 Methodology

2.1.1 Point Cloud Representation

Before applying deep learning to 3D data, it showed superior performance in the 2D domain when used for tasks such as image semantic segmentation [51, 52], image classification [53, 54], and object detection [55, 56]. This continued success in the 2D domain inspired the emergence of many methods to process point cloud data.

Multi-view 2D images. Representing point clouds as 2D images and using a 2D CNN to process the data was one of the earliest solutions to the problem of 3D processing. Multi-view based methods such as MVCNN [57] tackled the problem by representing the 3D point cloud as multiple 2D images from multiple views. This allowed processing of point clouds using CNNs, but presented other problems as well. Since the 2D multi-views generated are mere approximations, many geometric details can be lost during the generation of the 2D views. Another problem is how complex point clouds can be. A geometrically complex and large point cloud may need many 2D representations to be processed in the correct way. These problems tend to show that multi-view 2D representations of 3D data may not be the ideal way to go when processing 3D data.

Voxelization. Another early solution to solving the problem of processing 3D data is voxelization. By converting point clouds into 3D voxels, it was possible to finally process 3D voxelized point clouds using 3D CNNs [58]. However, voxelization is
extremely inefficient in terms of memory and computation, and therefore not ideal when considering deep networks.

**Processing point clouds directly.** The introduction of PointNet [44] allowed processing points directly without the need of an extra preprocessing step separate from the deep architecture. PointNet allows deep neural training on points directly. It processes a point set as follows:

\[
f(x_1, ..., x_n) \approx g(h(x_1), ..., h(x_n)) \tag{2.1}
\]

The function \( h() \) uses Multi-layer Perceptrons (MLPs) to approximate the per-point local features in the point cloud, and the function \( g() \) is a symmetric function that represents the aggregation of the global features using max pooling. The PointNet architecture, while seems simple, proved very effective in point cloud classification and segmentation. However, it does not take neighborhood information into account when processing points.

### 2.1.2 Graphs and GCNs

We represent a graph \( G \) using a tuple \( G = (V, E) \) where \( V \) is the set of unordered nodes and \( E \) is the set of edges representing the connectivity between the nodes \( v \in V \). If \( e_{i,j} \in E \), then the nodes \( v_i \) and \( v_j \) are connected with an edge \( e_{i,j} \).

**Graph Convolutional Networks.** Nodes are represented using GCNs by associating a node \( v \) with a feature vector \( f_v \in \mathbb{R}^D \), where \( D \) is the dimension of the features. So graph \( G \) can be represented by concatenating the features of all the nodes, \( i.e. \)

\[
f_G = [f_{v_1}, f_{v_2}, ..., f_{v_N}]^T \in \mathbb{R}^{N \times D}, \quad \text{where } N \text{ is the cardinality of } V.
\]

A general graph convolution operation \( F \) in the \( l \)-th layer can be represented as follows:

\[
G_{l+1} = F(G_l, W_l)
= \text{Update}(\text{Aggregate}(G_l, W_l^{\text{aggregate}}), W_l^{\text{update}}). \tag{2.2}
\]
where \( \text{Aggregate}(\cdot) \) and \( \text{Update}(\cdot) \) are the aggregation and update functions respectively, and \( G_l = (V_l, E_l) \), \( G_{l+1} = (V_{l+1}, E_{l+1}) \) are the input and output graphs of the \( l \)-th layer, respectively. \( W_l^{\text{aggregate}} \) and \( W_l^{\text{update}} \) are the weights of the aggregation and update functions respectively, and these two are learnable. They are also the most important components in the structure of GCNs. The representation of nodes is calculated in each layer by aggregating the features of neighborhood nodes for all \( v_{l+1} \in V_{l+1} \) as follows,

\[
f_{v_{l+1}} = \omega(f_{v_l}, \phi(\{f_{u_l} | u_l \in N(v_l)\}, f_{v_l}, W_\phi), W_\omega),
\]

where \( \phi \) is a node feature aggregator and \( \omega \) is a node feature updater. \( f_{v_l} \) and \( f_{v_{l+1}} \) are the node features in the \( l \)-th layer and \( l + 1 \)-th layer respectively. \( N(v_l) \) is the set of neighborhood nodes of \( v \) in the \( l \)-th layer, and \( f_{u_l} \) is the feature of the neighborhood nodes modeled by \( W_\phi \). \( W_\omega \) represents the weights of these functions. We use a max-pooling node feature aggregator with no learnable parameters to aggregate the difference of features between node \( v_l \) and its neighborhood. The updater \( \omega \) is represented as a multi-layer perceptron (MLP) with batch normalization [59] and a ReLU.

**Dynamic Edges.** Dynamic graph convolution allows the graph structure to change in every layer in GCNs. Recent work [60, 4, 61] shows the effectiveness of this strategy and how it results in better learning of graph representations compared to static graph structures. ECC (Edge-Conditioned Convolution) [60] uses dynamic edge-conditional filters to learn an edge-specific weight matrix. In addition, EdgeConv [4] uses \( k \)-NN to compute the nearest neighbors in feature space and construct a different graph after every EdgeConv layer. For point cloud generation, Graph-Convolution GAN [61] takes a similar approach to EdgeConv. Dynamic reconstruction of graph structure helps alleviate over-smoothing and results in a larger receptive field for deeper GCN
Figure 2.1: DeepGCN part segmentation architecture. DeepGCNs for part segmentation constitute a GCN Backbone, a Fusion Block, and an MLP Prediction Block. The GCN Backbone Block uses ResGCN) which utilizes residual connections.

architectures. In DeepGCNs, we propose to construct edges between nodes using a Dilated k-NN function in the feature space of each layer.

2.1.3 DeepGCNs

The architecture of DeepGCN shown in Figure 2.1 and proposed by [11] is constructed of 3 main blocks: the GCN Backbone, the Fusion block, and the MLP Prediction Block. The GCN backbone contains 28 blocks of graph convolution. The Fusion Block fuses the features produced by the GCN backbone with the aggregation output of the Global Max Pooling layer. Finally, the MLP Prediction Block applies 3 fully-connected operators to produce the final output class scores. For the GCN Backbone, we report results of using 28 ResGCN blocks as graph convolution blocks.

is applied after transforming $G_l$ by $F$ to get $G_{l+1}$. The residual mapping $F$ takes a graph as input and outputs a residual graph representation $G_{l+1}^{\text{res}}$ for the following layer. The learnable weights at a given layer $l$ is represented by $W_l$. In the experiments presented in this thesis, we refer to this model as $\text{ResGCN}$.

\[
G_{l+1} = H(G_l, W_l)
= F(G_l, W_l) + G_l = G_{l+1}^{\text{res}} + G_l.
\]  

(2.4)

\begin{flushleft}
\textbf{Dilated Convolution in GCNs.} Neighbors are constructed differently for each node given a dilation rate and neighborhood size. With a larger dilation rate, we increase the receptive field of the graph convolution. Dilation in graphs proved very useful as illustrated in the ablation study of DeepGCNs [11].
\end{flushleft}

2.2 Experiments

We evaluate DeepGCN on the recently proposed large-scale PartNet [13] dataset. PartNet consists of over 26,671 3D models from 24 object categories with 573,585 annotated part instances. The dataset establishes three benchmarking tasks for part segmentation on 3D objects: fine-grained semantic segmentation, hierarchical semantic segmentation, and instance segmentation. For the following experiments, we focus on the fine-grained level of semantic segmentation which includes 17 out of the 24 object categories present in the PartNet dataset. As is common practice, we use 10,000 sampled points as input to our network architecture. We use the reference architecture, $\text{ResGCN-28}$ in the experiments. We compare it to several state-of-the-art baseline architectures with default training hyper-parameters as reported in the original papers, namely PointNet [30], PointNet++ [31], SpiderCNN [62] and PointCNN [63].
2.2.1 Results

Qualitative results. Fig. 2.2 shows qualitative results on 4 categories of PartNet [13]: bottle, bed, microwave, and refrigerator. As expected from the results in Tables 2.1 and 2.2, ResGCN-28 performs very well compared to the baseline PlainGCN-28, where there are no residual connections between layers. Although ResGCN-28 produces some incorrect outputs compared to the ground truth in categories like microwave and refrigerator, it still outperforms PlainGCN-28 and segments the important parts of the object.

Comparison to state-of-the-art. We summarize the results of our performance compared to other state-of-the-art methods in Tables 2.1 and 2.2. ResGCN-28 which has 28 GCN layers, residual graph connections, and dilated graph convolutions outperforms other methods w.r.t the average mIoU overall. ResGCN-28 also performs better than other methods on 6 out of 17 fine-grained level categories and achieves similar performance as the state-of-the-art on other categories. In particular, ResGCN-28 achieves better results than previous methods in the categories bottle, dishwasher, earphone, knife, refrigerator, and vase. In addition, ResGCN-28 outperforms all other methods on average as reported in Table 2.2.

Note that PointCNN uses heavy data augmentation and hyper-parameter tuning unlike our ResGCN-28. For this reason, ResGCN-28 is outperformed by PointCNN in some categories. However, despite this disadvantage, ResGCN-28 outperforms PointCNN on some categories including bottle, clock, dishwasher, earphone, knife, lamp, refrigerator, table, and vase.
Table 2.1: Comparison of ResGCN-28 with other methods on PartNet Part Segmentation for the categories: bed, bottle, chair, clock, dishwasher, display, door, earphone, faucet, and knife. We report the part-category mean IoU (mIoU) on the fine-grained level of segmentation.

<table>
<thead>
<tr>
<th>Method</th>
<th>bed</th>
<th>bottle</th>
<th>chair</th>
<th>clock</th>
<th>dishw.</th>
<th>disp.</th>
<th>door</th>
<th>earph.</th>
<th>fauc.</th>
<th>knife</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>13.4</td>
<td>29.5</td>
<td>27.8</td>
<td>28.4</td>
<td>48.9</td>
<td>76.5</td>
<td>30.4</td>
<td>33.4</td>
<td>47.6</td>
<td>32.9</td>
</tr>
<tr>
<td>PointNet++</td>
<td>30.3</td>
<td>41.4</td>
<td>39.2</td>
<td><strong>41.6</strong></td>
<td>50.1</td>
<td>80.7</td>
<td>32.6</td>
<td>38.4</td>
<td>52.4</td>
<td>34.1</td>
</tr>
<tr>
<td>SpiderCNN</td>
<td>36.2</td>
<td>32.2</td>
<td>30.0</td>
<td>24.8</td>
<td>50.0</td>
<td>80.1</td>
<td>30.5</td>
<td>37.2</td>
<td>44.1</td>
<td>22.2</td>
</tr>
<tr>
<td>PointCNN</td>
<td><strong>41.9</strong></td>
<td>41.8</td>
<td><strong>43.9</strong></td>
<td>36.3</td>
<td>58.7</td>
<td><strong>82.5</strong></td>
<td>37.8</td>
<td>48.9</td>
<td><strong>60.5</strong></td>
<td>34.1</td>
</tr>
<tr>
<td>ResGCN-28 (Ours)</td>
<td>39.0</td>
<td><strong>54.7</strong></td>
<td>41.1</td>
<td>41.3</td>
<td><strong>59.7</strong></td>
<td>80.6</td>
<td>36.3</td>
<td>49.0</td>
<td>54.8</td>
<td><strong>55.5</strong></td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of ResGCN-28 with other methods on PartNet Part Segmentation for categories: lamp, microphone, fridge, storage furniture, table, trash can, and vase. We report the part-category mean IoU (mIoU) on the fine-grained level of segmentation. We also report the average mIoU for all categories including the ones from Table 2.1.

<table>
<thead>
<tr>
<th>Method</th>
<th>lamp</th>
<th>micro.</th>
<th>fridge</th>
<th>st. furn.</th>
<th>table</th>
<th>tr.</th>
<th>can</th>
<th>vase</th>
<th>Average mIOU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>18.9</td>
<td>37.2</td>
<td>33.5</td>
<td>38.0</td>
<td>29.0</td>
<td>34.8</td>
<td>44.4</td>
<td>35.6</td>
<td></td>
</tr>
<tr>
<td>PointNet++</td>
<td><strong>25.3</strong></td>
<td>48.5</td>
<td>36.4</td>
<td>40.5</td>
<td><strong>33.9</strong></td>
<td>46.7</td>
<td>49.8</td>
<td>42.5</td>
<td></td>
</tr>
<tr>
<td>SpiderCNN</td>
<td>19.6</td>
<td>43.9</td>
<td>39.1</td>
<td>44.6</td>
<td>20.1</td>
<td>42.4</td>
<td>32.4</td>
<td>37.0</td>
<td></td>
</tr>
<tr>
<td>PointCNN</td>
<td>20.1</td>
<td><strong>58.2</strong></td>
<td>42.9</td>
<td><strong>49.4</strong></td>
<td>21.3</td>
<td><strong>53.1</strong></td>
<td>58.9</td>
<td>46.5</td>
<td></td>
</tr>
<tr>
<td>ResGCN-28 (Ours)</td>
<td>22.8</td>
<td>55.9</td>
<td><strong>43.2</strong></td>
<td>46.9</td>
<td>31.2</td>
<td>52.5</td>
<td><strong>59.7</strong></td>
<td><strong>48.5</strong></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2: Qualitative Results on PartNet Part Segmentation. We show here the effect of adding residual connections to deep GCNs. PlainGCN-28 and ResGCN-28 are identical except for the presence of residual connections in ResGCN-28. We note how residual connections have a positive effect on part segmentation compared to PlainGCN-28. Many important parts of the objects are classified incorrectly using PlainGCN-28.
Chapter 3

Point Cloud Upsampling using GCNs

3.1 Methodology

3.1.1 Upsampling Modules

To effectively upsample point clouds, we propose two different upsampling modules: *Multi-branch GCN* and *Clone GCN* illustrated in Fig. 3.1.

**Multi-branch GCN**

Fig. 3.1a shows our Multi-branch GCN module. For an upsampling factor $r$, we pass the input point cloud through $r$ branches of graph convolutions. The outputs are concatenated node-wise to create the final output. Formally, for input $V_l \in \mathbb{R}^{N \times C}$, we obtain output $V_{l+1}^{UP} \in \mathbb{R}^{rN \times C}$ as follows:

$$V_{l+1}^{UP} = T(F_1(V_l), F_2(V_l), ..., F_r(V_l)),$$

(3.1)

where $T$ is a node-wise concatenation operator that fuses the outputs from different GCN branches. $F_i$ denotes the $i$-th GCN branch.

**Clone GCN**

Multi-branch GCNs are effective but parameter-heavy, since we need $r$ different GCN modules. To address this capacity issue, we propose Clone GCN (illustrated in Fig. 3.1b). Instead of using multiple branch convolutions in parallel, Clone GCN applies
Figure 3.1: The proposed upsampling modules. (a) Multi-branch GCN: We apply \( r \) different GCN layers to the input nodes. The outputs from each GCN are concatenated node-wise. (b) Clone GCN: We pass the input through \( r \) GCN layers with shared weights, and concatenate their outputs. \( \ll\ll \) denotes node-wise concatenation.

layers of a shared GCN to the output progressively. Then, the outputs are concatenated to generate the upsampled point cloud. The upsampling function performed by Clone GCN is defined as:

\[
\mathcal{V}_{l+1}^{UP} = \mathcal{T}(\mathcal{F}(\mathcal{V}_l), \mathcal{F}(\mathcal{V}_l), \mathcal{F}(\mathcal{V}_l), ..., \mathcal{F}^r(\mathcal{V}_l)),
\]

where \( \mathcal{F} \) denotes the shared GCN. \( \mathcal{F}^r(\mathcal{V}_l) \) is obtained by applying the same GCN to \( \mathcal{V}_l \) \( r \) times using shared weights.

In contrast to the previous upsampling modules like multi-branch CNNs [47] or duplicate-based upsampling [49, 50], our Multi-branch GCN and Clone GCN take advantage of graph convolutions instead of CNNs. GCNs enable our upsampler to encode spatial information from point neighborhoods and learn new points from the latent space rather than simply duplicate the original points (as done in 3PU [49] and PU-GAN [50]).

Comparison of Multi-branch CNN with Multi-branch GCN. Multi-branch CNN uses shared \( 1 \times 1 \) convolutions to learn per-point features. In our proposed multi-branch GCN, we find the local neighborhood for each point first, then the
graph convolutions (EdgeConv [45]) are used to learn the point neighborhood features. Multi-branch CNN is a special case of Multi-branch GCN when the number of neighbors $k$ equals to 0, and only the center point is used. In our GCN-based upsampling, $k$NN is used to find the neighborhood and build the graph. Different GCN operators, like EdgeConv [45] (which is used in our work by default), GAT [64], MRConv [11], and GraphSAGE [65] can be used as the graph convolution operation.

Comparison of duplicate-based approaches with Clone GCN. Duplicate-based approaches simply concatenate the same features $r$ times to upsample the points. Our proposed Clone GCN, applies graph convolutions to the point features $r$ times sequentially. The $(r - 1) \times$ new points are different cloned versions of the original points, instead of the duplicates of the original points.

3.2 Experiments

3.2.1 Dataset

We propose a new dataset for point cloud upsampling, denoted as $PU600$. It is four times bigger than the largest publicly available dataset collected by PU-GAN [50]. $PU600$ consists of 620 3D models split into 551 training samples and 69 testing samples. The training set contains 171 3D models compiled from the datasets used by PU-Net [47], 3PU [49], and PU-GAN [50], in addition to 380 different models collected from ShapeNet [66]. The testing set contains 39 models compiled from the datasets used by PU-Net, 3PU, and PU-GAN and 30 more models from ShapeNet. The models from ShapeNet were randomly chosen from 10 different categories and 450 different models with various complexities to encourage diversity. Overall, $PU600$ covers a great semantic range of 3D objects and includes simple, as well as complex shapes. More details of $PU600$ and a comparison with PU-GAN’s dataset can be found in Appendix A.
Evaluation Metrics. Following previous work, we use the Chamfer distance (CD), Hausdorff distance (HD), and point-to-surface distance (P2F) with respect to ground truth meshes as evaluation metrics. The smaller the metrics, the better the performance.

3.2.2 Implementation Details

For training and testing, we use Poisson disk sampling from the original meshes to generate input and ground truth point clouds. In training, we use farthest point sampling to crop 200 patches from each 3D model as the input to the network. In total, we obtain 110,200 training patches in PU600. Each patch consists of 256 input points sampled from the original model. We use the same method to sample 1024 points per patch as the ground truth. As for testing, we use farthest point sampling to sample overlapped patches (patch size is 256) of the input point cloud and ensure coverage of the entire input, then we implement patch-based upsampling. The final result is obtained by first merging the overlapping patch outputs and then resampling with farthest point sampling.

In all our experiments, we train the architectures for 100 epochs with batch size 28 on an NVIDIA Tesla V100 (16GB) GPU. We optimize using Adam with learning rate of 0.0005 and beta 0.9. We perform point cloud normalization and augmentation (rotation, scaling, and random perturbations). To show the effectiveness of our upsampling modules, we replace the upsampling modules in PU-Net and 3PU with the two upsampling modules proposed in this work. We train these networks on PU600 and compare their performance against the original architectures. All models converge before the maximum number of epochs. As suggested by PU-Net and 3PU, we use the model from the last epoch to evaluate performance. For all experiments, we report results using a ×4 upsampling rate.
3.2.3 Results

Quantitative results. Results in Table 3.1 show clear performance gains by replacing the upsampling modules in different architectures with our Multi-branch GCN and Clone GCN. The proposed upsampling modules have different impact depending on the architecture. For PU-Net, Clone GCN performs best on CD and P2F. For 3PU, Multi-branch GCN gives the best results in all metrics.

<table>
<thead>
<tr>
<th>Network</th>
<th>CD 10^{-3}</th>
<th>HD 10^{-3}</th>
<th>P2F 10^{-3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU-Net (Original) [47]</td>
<td>0.903</td>
<td>11.556</td>
<td>8.012</td>
</tr>
<tr>
<td>PU-Net (Multi-branch GCN)</td>
<td>0.731</td>
<td>11.855</td>
<td>7.332</td>
</tr>
<tr>
<td>PU-Net (Clone GCN)</td>
<td>0.717</td>
<td>13.930</td>
<td>6.520</td>
</tr>
<tr>
<td>3PU (Original) [49]</td>
<td>0.508</td>
<td>5.624</td>
<td>4.171</td>
</tr>
<tr>
<td>3PU (Multi-branch GCN)</td>
<td>0.501</td>
<td>5.516</td>
<td>3.916</td>
</tr>
<tr>
<td>3PU (Clone GCN)</td>
<td>0.702</td>
<td>5.552</td>
<td>3.930</td>
</tr>
</tbody>
</table>

Table 3.1: Ablation study on upsampling modules. Results show that our upsampling modules (Multi-Branch GCN and Clone GCN) can transfer well to different upsampling architectures in the literature. Replacing the original upsampling module with one of the proposed ones improves upsampling performance overall. Bold denotes the best performance for each architecture.

Qualitative results. Figures 3.2, 3.3 and 3.4 show qualitative results of using Multi-branch GCN and Clone GCN as upsampling modules replacing the ones in the original architectures of PU-Net [47] and 3PU [49]. The improvement is more obvious when integrating the upsampling modules in PU-Net compared to 3PU. Looking at the results of Figure 3.2 we can see how the intricate structure of the chair object is preserved. Specifically, looking at the chairs legs, one can observe how the new points are generated more consistently when using our upsampling modules. In Figure 3.3 we can also see how PU-Net’s original architecture tends to flatten the car’s tires when upsampling, while our upsampling modules greatly alleviate the problem. The tiger in Figure 3.4 illustrates our upsampling modules’ ability to avoid merging different parts of the objects when upsampling.
Figure 3.2: **Qualitative results of upsampling** (×4) (1/3). We show the upsampled point clouds of input (a) when processed using state-of-the-art methods (PU-Net and 3PU) while replacing their upsampling modules with our proposed Clone GCN and Multi-branch GCN. (a) Input point cloud. (b) PU-Net upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (c) 3PU upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (d) Ground Truth.
Figure 3.3: **Qualitative results of upsampling** ($\times4$) (2/3). (a) Input point cloud. (b) PU-Net upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (c) 3PU upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (d) Ground Truth.
Figure 3.4: **Qualitative results of upsampling** ($\times 4$) (3/3). (a) Input point cloud. (b) PU-Net upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (c) 3PU upsampling results. From left to right: Original, Clone GCN, Multi-branch GCN. (d) Ground Truth.
Chapter 4

Concluding Remarks

In this thesis, I discuss Graph Convolutional Networks (GCNs) and their effectiveness and present an application of DeepGCNs [11] in the task of part segmentation on PartNet [13], and show qualitatively and quantitatively the effective performance of the architecture of DeepGCN, which utilizes GCNs with a deep structure, residual connections, and dynamic dilated $k$-NN graph building. I also present two GCN-based upsampling modules (Multi-branch GCN and Clone GCN) which improve point cloud upsampling architectures when integrated into their pipelines. I show their effective performance on a new proposed standardized dataset $PU600$ which 4 times bigger than the largest publicly available dataset [50] and is rich in shape variety.

Future work

Integrating the upsampling modules into upsampling architectures proved effective. Our planned next step in this field is to test if these upsampling methods help when integrated into downstream tasks such as segmentation and detection. We also plan to experiment with GCNs and its effectiveness in real-world applications such as computational fluid dynamics (CFD).
REFERENCES


APPENDICES

A Details of Proposed Dataset PU600

We compile *PU600*, a new large-scale dataset with various levels of shape complexity. Our PU600 is four times bigger than the largest publicly available point upsampling dataset (PU-GAN’s dataset [50]). Fig. A.1 and Fig. A.2 show some of the training and testing samples from our PU600 dataset. We also show some examples from PU-GAN’s dataset in Fig. A.3 and Fig. A.4. We sample 1024 points from the original meshes for visualization. One can observe that our PU600 dataset contains models from more categories and with more shape diversity.
Figure A.1: Training samples from PU600.
Figure A.2: Testing samples from PU600.
Figure A.3: Training samples from PU-GAN’s Dataset.

Figure A.4: Testing samples from PU-GAN’s Dataset.
B Papers Submitted and Under Preparation