arXiv:1912.03264v2 [cs.CV] 28 Mar 2020

PU-GCN: Point Cloud Upsampling using Graph Convolutional Networks

https://sites.google.com/kaust.edu.sa/pugcn

Guocheng Qian*, Abdulellah Abualshour*, Guohao Li, Ali Thabet, and Bernard Ghanem

Visual Computing Center, King Abdullah University of Science and Technology
{guocheng.qian,abdulellah.abualshour,guohao.li,ali.thabet,bernard.ghanem}@kaust.edu.sa

Abstract. The effectiveness of learning-based point cloud upsampling pipelines heavily relies on the upsampling modules and feature extractors used therein. We propose three novel point upsampling modules: Multi-branch GCN, Clone GCN, and NodeShuffle. Our modules use Graph Convolutional Networks (GCNs) to better encode *local point information* from the point neighborhood. These upsampling modules are *versatile* and can be incorporated into any point cloud upsampling pipeline. Extensive experiments show how these modules consistently improve state-of-the-art upsampling methods. We also propose a new *multi-scale point feature extractor*, called Inception DenseGCN. By aggregating features at multiple scales, this feature extractor enables further performance gain in the final upsampled point clouds. We combine Inception DenseGCN with one of our upsampling modules (NodeShuffle) into a new point upsampling pipeline: PU-GCN. We show qualitatively and quantitatively the significant advantages of PU-GCN over the state-of-the-art. Source code of this work is available at https://github.com/guochengqian/PU-GCN.

Keywords: point cloud, upsampling, multi-scale feature extraction, graph convolutional networks

1 Introduction

Point clouds are a popular way to represent 3D data. This increasing popularity stems from the increased availability of 3D sensors like LiDAR. Such sensors are now a critical part of important application in robotics and self-driving cars. However, due to hardware and computational constraints, these 3D sensors often produce sparse and noisy point clouds, which show evident deficiencies especially for small objects or those far away from the camera. Therefore, point cloud upsampling, the task of converting sparse, incomplete and noisy point clouds into dense, complete and clean ones, is attracting much attention.

Following the success in image super-resolution [4, 10, 15, 19], deep learning methods now achieve state-of-the-art results in point cloud upsampling [14, 27–29]. Most deep upsampling pipelines comprise two major components: feature

* Equal Contribution

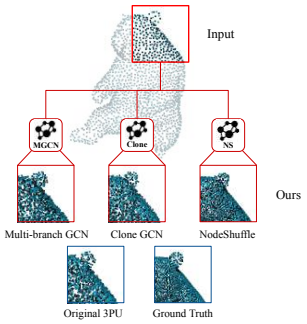


Fig. 1: **Proposed GCN-based point upsampling modules.** We develop three new upsampling modules (*Multi-branch GCN*, *Clone GCN*, and *NodeShuffle*) using a Graph Convolutional Network (GCN). When integrated into the 3PU [27] upsampling pipeline, they generate dense point clouds with fine-grained details, as compared to state-of-the-art point upsampling. The original 3PU uses duplicate-based upsampling.

extraction and point upsampling. The performance of the point upsampling component tends to define the effectiveness of the final network. Current methods use either a multi-branch CNN (PU-Net [29]) or a duplicate-based approach (3PU [27] and PU-GAN [14]) to upsample 3D points. Multi-branch CNNs operate on each point separately, ignoring any neighborhood information, while duplicate upsampling methods tend to generate point patches similar to the original point clouds. More details of these upsampling modules can be found in the **supplementary material**. These shortcomings lead to upsampled point clouds that lack local details (see Fig. 5). To better represent locality, we leverage the power of graphs and specifically Graph Convolutional Networks (GCNs). GCNs are considered a tool to process non-Euclidean data, and recent research on point cloud semantic and part segmentation shows their power in encoding local and global information [12,13,23,25]. In this paper, we use GCNs to design novel, versatile point cloud upsampling modules (see Fig. 1 and Fig. 2), which are better equipped at encoding local point information and learning to generate new point patches instead of merely replicating parts of the input.

Point clouds often represent objects with variable part sizes. Using multi-scale features is an effective way to encode this property, and is essential for obtaining point clouds of high quality. Recent works like PU-Net [29] extract point features at different downsampling levels. Such an architecture can encode multi-scale features, however, downsampling leads to loss of fine-grained details. In contrast, 3PU [27] proposes a progressive upsampling network using different numbers of neighbors in subsequent upsampling units. This achieves different receptive fields and encodes multi-scale information. However, 3PU is computationally expensive due to its progressive nature. In this paper, we tackle this feature learning problem using a densely connected GCN architecture we call Inception DenseGCN. Following its prevalent usage in image recognition [20–22] for the merits of efficient extraction of multi-scale image information, we adopt the Inception architecture to encode multi-scale point features, after it is modified to use densely connected GCNs instead of CNNs.

Contributions. We summarize our contributions as three-fold. **(1)** We propose three novel point cloud upsampling modules using graph convolutions: *Multi-branch GCN*, *Clone GCN*, and *NodeShuffle*. We show how these modules can be

seamlessly integrated into current point upsampling methods, and consistently improve their performance. **(2)** We design *Inception DenseGCN*, a feature extraction block that encodes multi-scale information effectively. We combine Inception DenseGCN and one of our proposed upsampling modules (NodeShuffle) into a new architecture named *PU-GCN*. Through extensive quantitative and qualitative results and for both synthetic and real data, we show the superior performance of PU-GCN to the state-of-the-art. **(3)** We compile *PU600*, a new large-scale dataset with variable levels of shape diversity. We show the advantages of our new dataset using our approach as well as other state-of-the-art methods. Our reproducible code and PU600 dataset will be publicly available.

2 Related Work

Graph convolutional networks (GCNs). To cope with the increasing amount of non-Euclidean data in real-world scenarios, a surge of graph convolutional networks [7,11,17,24,25] have been proposed in recent years. Kipf *et al.* [11] simplified spectral graph convolutions with a first-order approximation. DGCNN [25] proposed EdgeConv to conduct dynamic graph convolution on point clouds. Recently, Li *et al.* [12,13] introduced residual/dense connections and dilated convolutions to GCNs, and successfully trained deep GCN architectures. Previous GCN works mainly investigate basic modules of discriminative models. However, due to the unordered and irregular nature of graph data, generative tasks for this modality remain elusive. In particular, point upsampling techniques, which are indispensable components for generative models, are under-explored.

Multi-scale feature extraction. Inception architectures [20–22] enable very good performance in image recognition at relatively low computational cost. They extract multi-scale information by using different kernel sizes in different paths of the architecture. Inspired by the success of the Inception architecture for CNNs, Kazi *et al.* [9] proposed InceptionGCN, in which feature maps are passed into multiple branches, then each branch applies one graph convolution with a different kernel size. The outputs of these branches are aggregated by concatenation. We also adopt the Inception concept in our work to propose Inception DenseGCN, a GCN architecture that improves upon InceptionGCN by leveraging dense connections and global pooling.

Optimization-based upsampling methods. Alexa *et al.* [1] introduced one of the earliest optimization-based methods for point cloud upsampling. The idea behind their work is to insert new points at the vertices of the Voronoi diagram computed on the moving least squares surface. Lipman *et al.* [16] introduced a locally optimal projection operator to resample points based on an L_1 norm. Another optimization-based method was introduced by Huang *et al.* [8], in which they proposed a method of resampling to process noisy and outlier-ridden point clouds in an edge-aware manner. A consolidation approach was introduced by Wu *et al.* [26], in which they augment surface points into *deep points* by associating them with other points that lie on the meso-skeleton of the shape. All these optimization-based methods rely on the quality of human crafted priors.

Deep learning-based upsampling methods. Deep learning methods illustrate a promising improvement over optimization-based algorithms due to their data-driven nature and the learning capacity of neural networks. Learning features directly from point clouds was made possible by deep neural networks, such as PointNet [5], PointNet++ [18], DGCNN [25], KPConv [23], *etc.* Yu *et al.* [29] introduced the PU-Net that learns multi-scale features per point, expands the point set via a multi-branch CNN in feature space. However, PU-Net needs to downsample the input first to learn multi-scale features, which causes unnecessary resolution loss. Yu *et al.* [28] also proposed EC-Net, an edge-aware network for point set consolidation. It uses an edge-aware joint loss to encourage the network to learn to consolidate points for edges. However, it requires a very expensive edge-notation for training. Wang *et al.* [27] proposed 3PU, a progressive network that duplicates the input point patches over multiple steps. By using different kernel sizes in different upsampling units, 3PU learns multi-scale features with no resolution loss, unlike PU-Net. However, 3PU is computationally expensive due to its progressive nature, and it requires more data to supervise the middle stage output of the network. Recently, Li *et al.* [14] proposed PU-GAN, a Generative Adverserial Network (GAN) designed to learn upsampled point distributions in point clouds from latent space. Different from previous work, they first upsample more points than the required target (*e.g.*$\times 6$) and then use farthest upsampling to sample the target number of points (*e.g.*$\times 4$) to produce the final output. While PU-GAN focuses on improving uniformity of upsampled point clouds by introducing a novel uniformity loss function, it does not specifically design upsampling modules or feature extractors. Our work tackles upsampling using novel GCN upsampling modules to capture local point information, and by encoding multi-scale features using Inception DenseGCN.

3 Methodology

3.1 Upsampling Modules

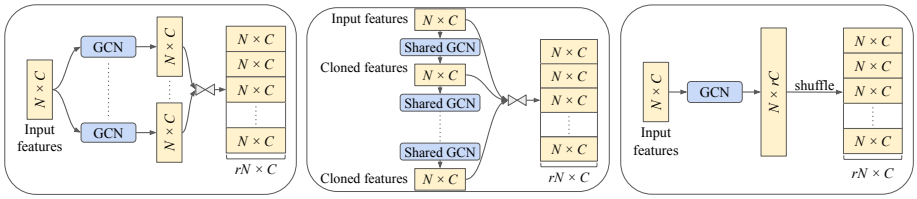
To effectively upsample point clouds, we propose three different upsampling modules: *Multi-branch GCN*, *Clone GCN*, and *NodeShuffle* illustrated in Fig. 2.

Multi-branch GCN. Fig. 2a 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 $\mathcal{V}_l \in \mathbb{R}^{N\times C}$, we obtain output $\mathcal{V}_{l+1}^{UP} \in \mathbb{R}^{rN\times C}$ as follows:

$$\mathcal{V}_{l+1}^{UP} = \mathcal{T}(\mathcal{F}_1(\mathcal{V}_l), \mathcal{F}_2(\mathcal{V}_l)..., \mathcal{F}_r(\mathcal{V}_l)), \tag{1}$$

where $\mathcal{T}$ is a node-wise concatenation operator that fuses the outputs from different GCN branches. $\mathcal{F}_i$ 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. 2b). Instead of using multiple branch convolutions in



(a) Multi-branch GCN (b) Clone GCN (c) NodeShuffle

Fig. 2: **Our 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. (c) NodeShuffle: We expand the number of input features using a GCN layer. We later apply a shuffle operation to rearrange the feature map. $\bowtie$ denotes node-wise concatenation.

parallel, Clone GCN applies 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:

$$\mathcal{V}_{l+1}^{UP} = \mathcal{T}(\mathcal{F}(\mathcal{V}_l), \mathcal{F}(\mathcal{F}(\mathcal{V}_l)), \mathcal{F}^3(\mathcal{V}_l), ..., \mathcal{F}^r(\mathcal{V}_l)), \tag{2}$$

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.

NodeShuffle. Both Multi-branch GCN and Clone GCN run r graph convolutions. To reduce the computation burden further, we propose NodeShuffle (illustrated in Fig. 2c). Inspired by PixelShuffle [19] from the image super-resolution literature, NodeShuffle is an efficient graph convolutional upsampling layer. Given node features $\mathcal{V}_l$ with shape $N \times C$, NodeShuffle operation will output the new node features $\mathcal{V}_{l+1}^{UP}$ with shape $rN \times C$ as follows:

$$\mathcal{V}_{l+1}^{UP} = \mathcal{PS}(\mathcal{W}_{l+1} * \mathcal{V}_l + b_{l+1}), \tag{3}$$

where $\mathcal{PS}$ is a periodic shuffling operator that rearranges the graph ($e.g.$point features) of shape $N \times rC$ to $rN \times C$. The NodeShuffle operation can be divided into two steps. (1) Channel expansion: we use a 1 layer GCN to expand node features $\mathcal{V}_l$ to shape $N \times rC$, using learnable parameters $\mathcal{W}_{l+1}$ and b_{l+1}. (2) Periodic shuffling: we rearrange the output of channel expansion to shape $rN \times C$. NodeShuffle runs only 1 GCN operation, as opposed to r such operations in Multi-branch and Clone GCN.

In contrast to the previous upsampling modules like multi-branch CNNs [29] or duplicate-based upsampling [14, 27], our Multi-branch GCN, Clone GCN, and NodeShuffle leverage 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 [27] and PU-GAN [14]).

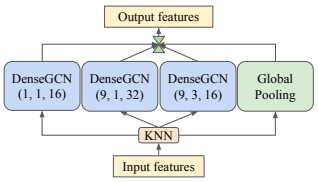


Fig. 3: **Inception DenseGCN.** We use the parameters (k, d, c) to define a DenseGCN block. k is the number of neighbors (kernel size), d is the dilation rate, and c is the number of output channels. The green $\bowtie$ denotes feature-wise concatenation.

3.2 Multi-scale Feature Extractor: Inception DenseGCN

Point clouds scanned using 3D sensors often include objects of various sizes and point resolutions. In order to encode the multi-scale nature of point clouds, we propose a new *Inception DenseGCN* feature extractor, which effectively integrates the DenseGCN module from DeepGCN [13] into the Inception module from GoogLeNet [21]. Residual and dense connections have been proven to be useful to increase the performance by Li *et al.* [13]. We favor dense over residual connections here, since the former utilizes features from previous layers, as well as different inception paths.

Fig. 3 shows our Inception DenseGCN block. In our experiments, we use three DenseGCN blocks per Inception block. Each DenseGCN block is defined by the number of node neighbors k, dilation rate d, and number of filters c. Note that the dilated graph convolution operator was introduced in DeepGCN [13]. Similar to the 2D case, this operator efficiently increases the receptive field using the same k without reducing spatial resolution. K nearest neighbor algorithm (KNN) is used at the first layer of the Inception DenseGCN block to build the graph and obtain the node neighborhoods. Experiments show that making the graph dynamic in the later layers inside Inception DenseGCN does not actually impact performance much, but it does increase the model's computational footprint. So, we decide to share the graph structure and not update it in subsequent layers inside Inception DenseGCN. Additionally, we add a global pooling layer to extract global contextual information. Each Inception block outputs a concatenation of the DenseGCN blocks and the global pooling layer.

3.3 PU-GCN Architecture

We combine the proposed Inception DenseGCN extractor with our novel upsampling modules, and followed by a coordinate reconstructor propose a new upsampling model called PU-GCN. Its architecture is illustrated in Fig. 4. Given a point cloud of size $N \times 3$, we compute dense features of size $N \times C$ using our dense feature extractor. Then, we upsample the $N \times C$ features to $rN \times C'$ using one of the proposed upsamplers. Finally, the coordinate reconstructor generates the $rN \times 3$ upsampled point cloud.

Dense feature extractor. We use 1 GCN layer followed by 1 DenseGCN layer at the beginning of PU-GCN to embed the 3D coordinates into latent space and extract higher-level spatial information. We do not use dilated graph convolutions in these two layers in order to preserve more local information. More

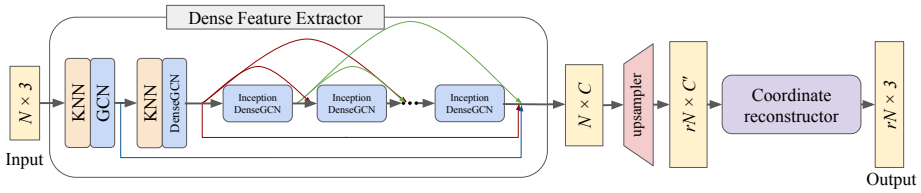


Fig. 4: **PU-GCN architecture.** PU-GCN uses a dense feature extractor consisting of 1 or more densely connected Inception DenseGCN blocks, followed by the upsampler and coordinate reconstructor.

details about the influence of dilation rate in the first layers are discussed in the **supplementary material**. The output of DenseGCN will be passed through several densely connected Inception DenseGCN blocks. In our experiments, we only use two such Inception DenseGCN blocks. We experiment with the number of Inception DenseGCN blocks in Section 4.6. The outputs of the first GCN layer, the DenseGCN block, and the Inception DenseGCN blocks are concatenated together and passed to our upsampler module.

Upsampler. Our upsampler consists of two stages: upsampling and feature compression. Given input features $N \times C$, we use our proposed upsampling modules to generate denser features of size $rN \times C$. Then, we use two sets of MLPs to compress features to $rN \times C'$. By default, we use NodeShuffle in PU-GCN. We experiment with the different upsampling modules in Section 4.6.

Coordinate reconstructor. We reconstruct points from latent space to 3D coordinate space, resulting in the desired denser point cloud of size $rN \times 3$. We use the same coordinate reconstruction approach as 3PU [27] and PU-GAN [14], in which 3D coordinates are regressed using two sets of MLPs.

4 Experiments

4.1 Datasets

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 [14]. 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 [29], 3PU [27], and PU-GAN [14], in addition to 380 different models collected from ShapeNet [3]. 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. To show the value of our proposed dataset, we compare our PU-GCN with previous approaches on both PU600 and the latest dataset proposed by PU-GAN, which contains only 120 3D models for

training and 27 models for testing. More details of PU600 and a comparison with PU-GAN's dataset can be found in the **supplementary material**.

4.2 Loss Function and Evaluation Metrics

Loss Function. We use the modified Chamfer distance with weighted repulsion loss introduced in 3PU [27]:

$$L = C(P, Q) + \lambda L_{rep}, \tag{4}$$

where $C(P, Q)$ is the modified Chamfer distance loss between the predicted point cloud P and the ground truth point cloud Q. L_{rep} denotes the repulsion loss, which encourages generated points to be far from the original ones. λ is a trade-off coefficient. As suggested by 3PU [27], we use $\lambda = 0.02$ in all our experiments. The influence of repulsion loss on PU-GCN is studied in **supplementary material**.

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.

4.3 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. The data preprocessing mentioned above is the same as 3PU [27]. To measure performance w.r.t. the size of the input point cloud, we sample two sets of test samples: a sparse input with only 256 points and a dense input with 2048 points. We will open source the original meshes of PU600 and the sampled point clouds to standardize the dataset.

In all our experiments, we train PU-GCN 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. Similar to previous works, we perform point cloud normalization and augmentation (rotation, scaling, and random perturbations). To fairly compare PU-GCN against PU-Net [29] and 3PU [27], we train all these upsampling pipelines using our PU600 dataset, as well as PU-GAN's dataset. Here, we note that we were unable to reproduce PU-GAN's [14] results from the code made available by its authors, most probably because of the unstable nature

Table 1: **Performance comparison of PU-GCN vs. the state-of-the-art on PU-GAN's dataset.** Our PU-GCN outperforms PU-Net, 3PU and PU-GAN in most cases, regardless of the input point cloud size. **Bold** denotes the best performance.

Network	**PU-GAN's Dataset**					
	Sparse (256) input			**Dense (2048) input**		
	CD 10^{-3}	**HD** 10^{-3}	**P2F** 10^{-3}	**CD** 10^{-3}	**HD** 10^{-3}	**P2F** 10^{-3}
PU-Net [29]	2.461	14.370	13.099	0.720	8.940	6.840
3PU [27]	2.177	12.672	10.328	0.490	6.110	3.960
PU-GAN [14]	2.072	16.592	**8.055**	0.280	4.640	**2.330**
PU-GCN	**1.973**	**11.786**	9.591	**0.254**	**1.821**	2.935

of the inherent GAN architecture in PU-GAN. Therefore, we only compare with PU-GAN on PU-GAN's dataset using the pretrained model made available by its authors. To show the effectiveness of our upsampling modules, we also replace the upsampling modules in PU-Net and 3PU with the three upsampling modules proposed in this paper. We train these networks on PU600 and compare their performance against PU-GCN. 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 $\times 4$ upsampling rate.

4.4 Quantitative and Qualitative Results

Quantitative results on PU-GAN's dataset. Table 1 reports the performance of our PU-GCN compared to PU-Net, 3PU and PU-GAN on PU-GAN's dataset. PU-GCN maintains significant improvement over 3PU and PU-Net on all metrics using sparse and dense inputs, showing the importance of the Inception DenseGCN feature extractor and the NodeShuffle upsampling module. We also outperform PU-GAN on both CD and HD metrics. For the P2F metric, we do not outperform PU-GAN quantitatively. However, the qualitative results in Fig. 5 (and others in the **supplementary material**) show that PU-GCN generates less outliers and higher quality local and fine-grained details (*e.g.* the back of the chair in row 3 and the legs of the bird in row 4) than PU-GAN in general, even though these upsampled point clouds have higher P2F values than those of PU-GAN. This discrepancy indicates that P2F might not be as reliable a metric as CD and HD for point cloud upsampling, especially since the latter distances are meant for point clouds (the natural form of the input/output in training) and not meshes (as is the case for P2F). More examples and an analysis of this discrepancy can be found in the **supplementary material**.

Quantitative results on PU600 dataset. Table 2 compares PU-GCN against PU-Net and 3PU on PU600. We do not compare against PU-GAN on this dataset for the reason mentioned in Section 4.3. We observe all methods achieve lower performance overall when trained and evaluated on PU600 compared to PU-GAN's dataset. As such, PU600 presents a challenge to state-of-the-art methods compared to the much smaller and less diverse PU-GAN's dataset.

Table 2: **Performance comparison of PU-GCN vs. the state-of-the-art on PU600.** PU-GCN outperforms PU-Net and 3PU. We do not compare against PU-GAN on this dataset, since we were unable to reproduce PU-GAN results from the publicly available code. **Bold** denotes the best performance.

Network	PU600					
	Sparse (256) input			Dense (2048) input		
	CD 10^{-3}	HD 10^{-3}	P2F 10^{-3}	CD 10^{-3}	HD 10^{-3}	P2F 10^{-3}
PU-Net [29]	3.260	23.504	10.059	0.903	11.556	8.012
3PU [27]	3.250	18.002	11.230	0.508	5.624	4.171
PU-GCN	**2.325**	**13.752**	**4.900**	**0.333**	**2.529**	**1.529**

Qualitative results. Fig. 5 shows qualitative upsampling results generated by PU-GCN and the state-of-the-art methods. We note that all models used here are trained on PU-GAN's dataset for fair comparison with PU-GAN. Upsampled point clouds and their close-ups show that PU-GCN produces less outliers, while preserving more fine-grained details. Particularly, close-ups of the bed point cloud (top row) show that PU-GCN successfully upsamples intricate structures of the bed. In addition, the Saint Lambert statue in the second row shows how PU-GCN succeeds in upsampling small fine-grained details, while other methods fail to do so. We also observe that other methods tend to merge originally separate structures as shown in the example of the clock in the fifth row, while our method preserves this separation with high quality. The qualitative results clearly show the effectiveness of our multi-scale feature extractor (Inception DenseGCN) and our GCN based upsampling module (NodeShuffle) in capturing detailed local information. More qualitative results and a discussion of some failure cases can be found in the **supplementary material**.

4.5 Upsampling Real-Scanned Point Clouds

Fig. 6 compares PU-GCN against its two most competitive upsampling methods (3PU and PU-GAN) on real-scanned data from the KITTI dataset [6]. KITTI contains challenging real-world scenes and objects. All the models are trained on PU-GAN's dataset for fair comparison with PU-GAN. To ease visualization and since it consistently produces worse results than the other methods, PU-Net [29] is not included here. Looking at the top row, we observe that the other methods tend to produce more outliers and overfill holes (see the window of the truck), while PU-GCN encodes local information well and preserves these fine-grained details. Our method also does very well when upsampling objects of interest, such as people and cyclists as shown in the other examples. A higher level of detail and structure is produced using PU-GCN.More examples of real-scanned point clouds from different datasets can be found in the **supplementary material**.

4.6 Ablation Study

To justify the design choices of our architecture, we conduct ablation studies on the proposed upsampling modules and Inception DenseGCN. Here, we use

Table 3: **Effect of Inception DenseGCN and global pooling**. Using a single Inception DenseGCN block performs better than using the dynamic GCN feature extractor in 3PU [27]. Global pooling further improves performance. Increasing the number of Inception DenseGCN blocks tends to improve PU-GCN performance as well. Here, NodeShuffle is used as the upsampling module, the dataset is PU600, the upsampling ratio is ×4, and the input size is 2048 points.

Ablation	**CD** 10^{-3}	**HD** 10^{-3}	**P2F** 10^{-3}
Dynamic GCN feature extraction [27]	0.340	2.663	1.666
No global pooling in Inception DenseGCN	0.340	2.618	1.627
PU-GCN† (1 Inception DenseGCN)	0.335	2.549	1.576
PU-GCN (2 Inception DenseGCN)	**0.333**	**2.529**	**1.529**

PU-GCN† as our baseline architecture, which uses a single Inception block as its dense feature extractor and NodeShuffle as the upsampling module. All the models are trained on PU600. All inputs in these experiments have 2048 points.

Inception Modules. We validate the effectiveness of our Inception DenseGCN by replacing it with the dynamic GCN used in 3PU [27], which has slightly more parameters. The results in Table 3 show that our Inception DenseGCN outperforms the dynamic GCN feature extractor in all metrics. Additionally, we show that using global pooling inside our Inception DenseGCN improves performance. We also study the effect of the number of Inception DenseGCN blocks. PU-GCN (2 Inception blocks) outperforms PU-GCN† (1 Inception block), thus showing that inserting one more Inception DenseGCN block can further improve upsampling performance. We also experimented with using more blocks, but performance did not improve while computational complexity did. Therefore, we use 2 Inception DenseGCN blocks in PU-GCN.

Upsampling Modules. Results in Table 4 show clear performance gains by replacing the upsampling modules in different architectures with our Multi-branch GCN, Clone GCN, and NodeShuffle. The proposed upsampling modules have different impact depending on the architecture. For PU-Net, Clone GCN performs best on CD and P2F. For 3PU, NodeShuffle gives the best results in all cases, while Multi-branch GCN performs second best. A qualitative comparison of our different upsampling modules integrated in 3PU is shown in Fig. 1. NodeShuffle shows clear improvement in generating fine-grained details compared to the other upsampling methods. For PU-GCN†, NodeShuffle also outperforms the other upsampling modules. This is the main reason why it is used as the default upsampling method in the previous experiments.

4.7 Effects of Additive Noise and Input Size on Point Upsampling

Upsampling point clouds with additive noise. To show the robustness of PU-GCN, we perturb the input point cloud with additive Gaussian noise at varying noise levels. We show qualitative results in Fig. 7a and compare our PU-GCN against the state-of-the-art PU-GAN. Both models were trained using the same augmentation strategy of point cloud perturbation. Results show that

Table 4: **Ablation study on upsampling modules.** Results show that our upsampling modules (Multi-Branch GCN, Clone GCN, and NodeShuffle) 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. PU-GCN† uses a single Inception DenseGCN block in its feature extractor. **Bold** denotes the best performance for each architecture.

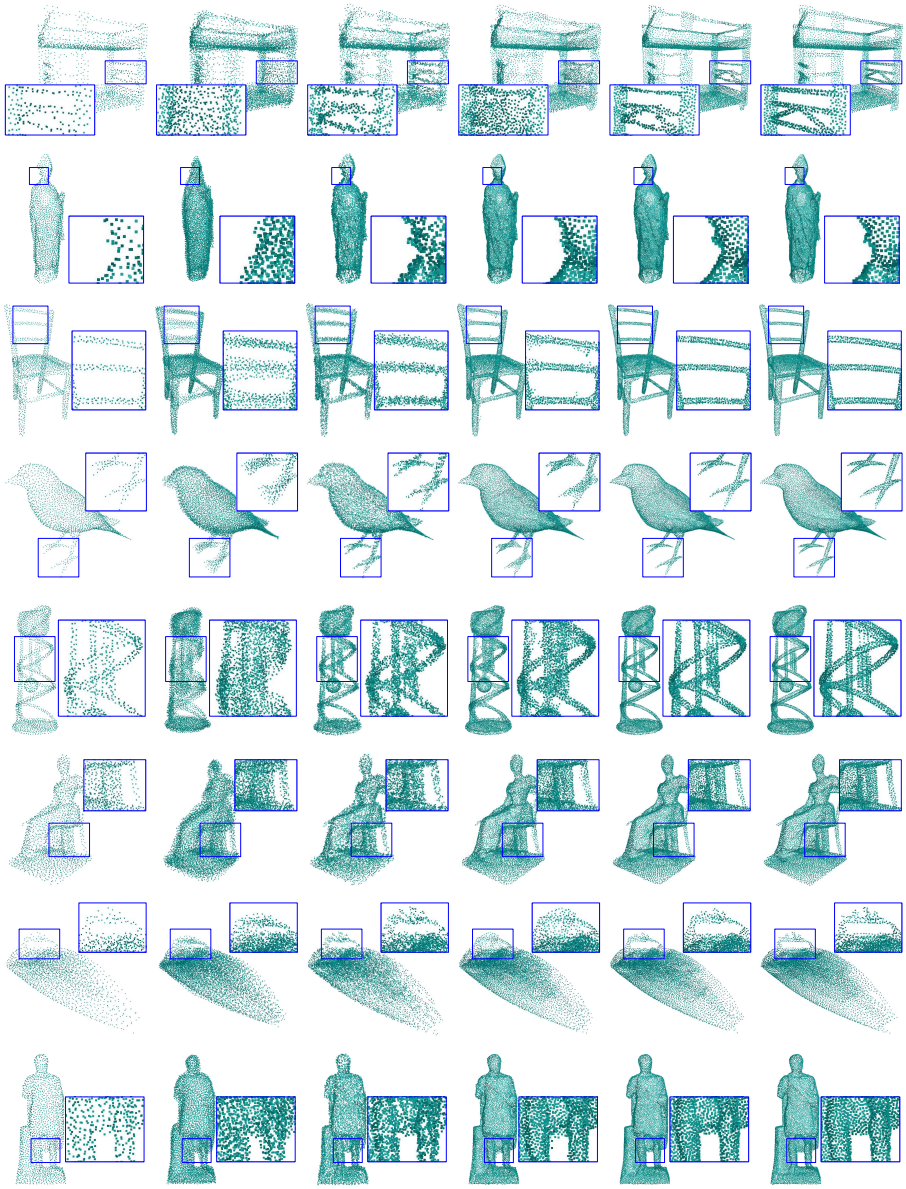
Network	**CD** 10^{-3}	**HD** 10^{-3}	**P2F** 10^{-3}
PU-Net (Original) [29]	0.903	**11.556**	8.012
PU-Net (Multi-branch GCN)	0.731	11.855	7.332
PU-Net (Clone GCN)	**0.717**	13.930	**6.520**
PU-Net (NodeShuffle)	0.888	18.33	8.934
3PU (Original) [27]	0.508	5.624	4.171
3PU (Multi-branch GCN)	0.501	5.516	3.916
3PU (Clone GCN)	0.702	5.552	3.930
3PU (NodeShuffle)	**0.369**	**3.760**	**2.290**
PU-GCN† (Multi-branch GCN)	0.845	6.706	3.1558
PU-GCN† (Clone GCN)	0.803	6.283	3.056
PU-GCN† (NodeShuffle)	**0.335**	**2.549**	**1.576**

PU-GCN can preserve fine-grained details with very few outliers, even in the presence of additive noise, while PU-GAN tends to produce more outliers.

Upsampling point clouds of varying sizes. Fig. 7b shows qualitative examples of upsampling point clouds with PU-GCN for different input sizes. Our architecture always produces high quality point clouds regardless of input cloud size. This indicates that even if PU-GCN is trained on patches with only 256 points, it can generalize to point clouds with different sizes. As expected, PU-GCN generates better quality results when the input point cloud is denser.

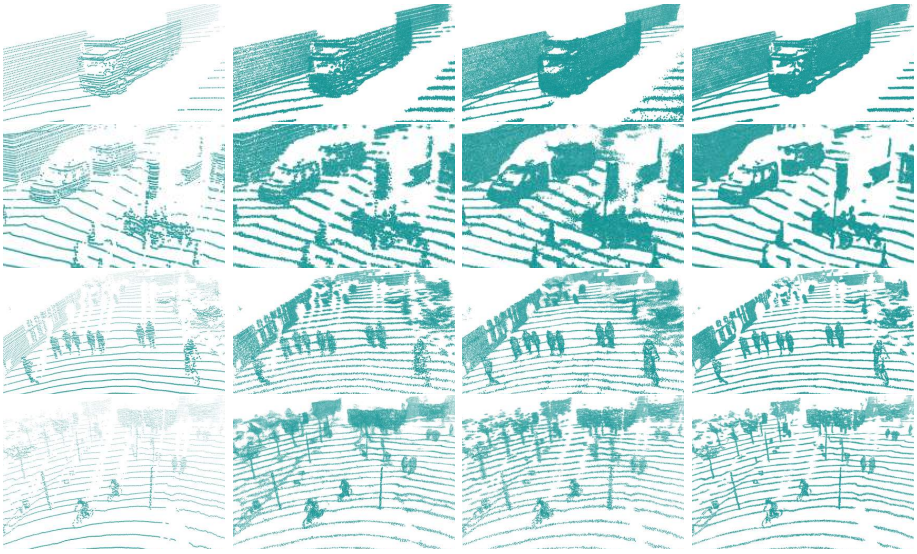
5 Conclusion

We propose three novel GCN based point cloud upsampling modules Multi-branch GCN, Clone GCN, and NodeShuffle, which improve state-of-the-art upsampling pipelines when they are used in place of the original upsampling. We also introduce the Inception DenseGCN feature extractor to encode multi-scale information from input point clouds. We further compile and introduce a new large-scale dataset PU600 for point cloud upsampling. Our PU-GCN, which integrates Inception DenseGCN and NodeShuffle, outperforms the state-of-the-art on previous upsampling datasets and our PU600. We show that PU-GCN has better performance on the real-scanned point clouds compared to the state-of-the-art. PU-GCN also performs well on synthesized point clouds with different levels of noise and with different input sizes. Since image super-resolution has been shown useful in some 2D object detection tasks [2, 30], we believe that our proposed versatile upsampling modules and multi-scale feature extractor can be used in point cloud segmentation/detection tasks as well, especially if the inputs are sparse. We intend to investigate the effect of upsampling and multi-scale feature extraction when we integrate them into downstream pipelines.



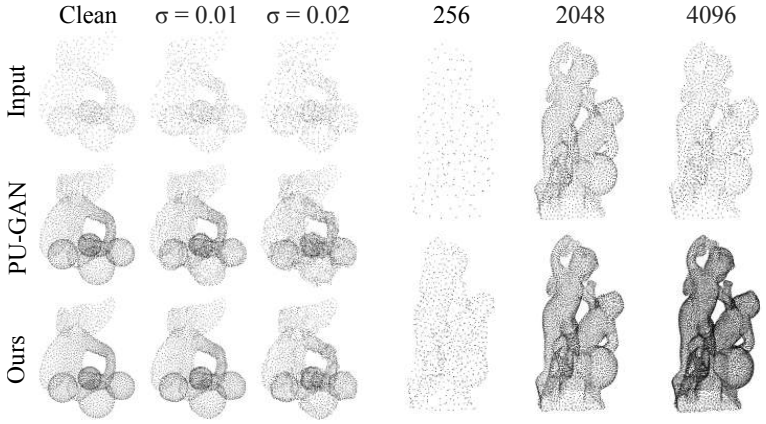
(a) Input (b) PU-Net (c) 3PU (d) PU-GAN (e) Ours (f) GT

Fig. 5: **Qualitative upsampling (×4) results.** We show the upsampled point clouds of input (a) when processed by different upsampling methods (b)-(d) and our proposed PU-GCN (e). The ground truth upsampled point clouds are in (f) for reference. All the inputs are taken from the PU600 test set and have 2048 points. PU-GCN produces the best results overall, while preserving fine-grained local details (refer to close-ups).



(a) Input (b) 3PU (c) PU-GAN (d) PU-GCN (Ours)

Fig. 6: **Upsampling real-scanned point clouds from KITTI dataset [6]**. PU-GCN preserves intricate structures in objects in real-scanned data and generates fine-grained details (the window of the truck, the pedestrians, the cyclist, *etc*). Please zoom in to see details.



(a) (b)

Fig. 7: **More qualitative results using PU-GCN.** The top row of (a) shows the input point clouds with different additive noise levels: 0, 0.01, and 0.02 from left to right, respectively. The middle row of (a) shows the ×4 upsampled point clouds using PU-GAN. The bottom row of (a) shows the upsampled point clouds using PU-GCN (Ours). The top row of (b) shows the input point clouds of different sizes, and the bottom row of (b) shows the ×4 upsampled outputs.

References

1. Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C.T.: Computing and rendering point set surfaces. IEEE Trans. Vis. Comput. Graph. **9**, 3–15 (2003)
2. Cao, L., Wang, C., Li, J.: Vehicle detection from highway satellite images via transfer learning. Information sciences **366**, 177–187 (2016)
3. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An Information-Rich 3D Model Repository. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015)
4. Dong, C., Loy, C.C., He, K., Tang, X.: Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **38**, 295–307 (2014)
5. Garcia-Garcia, A., Gomez-Donoso, F., Rodriguez, J.A., Orts, S., Cazorla, M., López, J.A.: Pointnet: A 3d convolutional neural network for real-time object class recognition. 2016 International Joint Conference on Neural Networks (IJCNN) pp. 1578–1584 (2016)
6. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2012)
7. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems. pp. 1024–1034 (2017)
8. Huang, H., Wu, S., Gong, M., Cohen-Or, D., Ascher, U.M., Zhang, H.: Edge-aware point set resampling. ACM Trans. Graph. **32**, 9:1–9:12 (2013)
9. Kazi, A., Shekarforoush, S., Krishna, S.A., Burwinkel, H., Vivar, G., Kortüm, K., Ahmadi, S.A., Albarqouni, S., Navab, N.: Inceptiongcn: Receptive field aware graph convolutional network for disease prediction. In: International Conference on Information Processing in Medical Imaging. pp. 73–85. Springer (2019)
10. Kim, J., Lee, J.K., Lee, K.M.: Accurate image super-resolution using very deep convolutional networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 1646–1654 (2015)
11. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
12. Li, G., Müller, M., Qian, G., Delgadillo, I.C., Abualshour, A., Thabet, A.K., Ghanem, B.: Deepgcns: Making gcns go as deep as cnns. ArXiv **abs/1910.06849** (2019)
13. Li, G., Mller, M., Thabet, A., Ghanem, B.: Deepgcns: Can gcns go as deep as cnns? In: The IEEE International Conference on Computer Vision (ICCV) (2019)
14. Li, R., Li, X., Fu, C.W., Cohen-Or, D., Heng, P.A.: Pu-gan: a point cloud upsampling adversarial network. In: IEEE International Conference on Computer Vision (ICCV) (2019)
15. Lim, B., Son, S., Kim, H., Nah, S., Lee, K.M.: Enhanced deep residual networks for single image super-resolution. 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) pp. 1132–1140 (2017)
16. Lipman, Y., Cohen-Or, D., Levin, D., Tal-Ezer, H.: Parameterization-free projection for geometry reconstruction. In: ACM SIGGRAPH 2007 Papers. SIGGRAPH '07, ACM, New York, NY, USA (2007). https://doi.org/10.1145/1275808.1276405, `http://doi.acm.org/10.1145/1275808.1276405`

17. Pham, T., Tran, T., Phung, D., Venkatesh, S.: Column networks for collective classification. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
18. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: NIPS (2017)
19. Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A.P., Bishop, R., Rueckert, D., Wang, Z.: Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 1874–1883 (2016)
20. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: AAAI (2016)
21. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 1–9 (2014)
22. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 2818–2826 (2015)
23. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: Flexible and deformable convolution for point clouds. Proceedings of the IEEE International Conference on Computer Vision (2019)
24. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
25. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph cnn for learning on point clouds. ACM Transactions on Graphics (TOG) (2019)
26. Wu, S., Huang, H., Gong, M., Zwicker, M., Cohen-Or, D.: Deep points consolidation. ACM Trans. Graph. **34**(6), 176:1–176:13 (Oct 2015). https://doi.org/10.1145/2816795.2818073, `http://doi.acm.org/10.1145/2816795.2818073`
27. Yifan, W., Wu, S., Huang, H., Cohen-Or, D., Sorkine-Hornung, O.: Patch-based progressive 3d point set upsampling. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
28. Yu, L., Li, X., Fu, C.W., Cohen-Or, D., Heng, P.A.: Ec-net: an edge-aware point set consolidation network. In: ECCV (2018)
29. Yu, L., Li, X., Fu, C.W., Cohen-Or, D., Heng, P.A.: Pu-net: Point cloud upsampling network. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
30. Zhang, Y., Ding, M., Bai, Y., Ghanem, B.: Detecting small faces in the wild based on generative adversarial network and contextual information. Pattern Recognition **94**, 74–86 (2019)

PU-GCN: Point Cloud Upsampling using Graph Convolutional Networks Supplementary Material

Guocheng Qian*, Abdulellah Abualshour*, Guohao Li, Ali Thabet, and Bernard Ghanem

Visual Computing Center, King Abdullah University of Science and Technology
{guocheng.qian,abdulellah.abualshour,guohao.li,ali.thabet,bernard.ghanem}@kaust.edu.sa

In this supplementary material, we provide additional content to complement the paper. First, we present detailed configurations of our network PU-GCN and experiment the influence of dilation rate on the quantitative performance. We provide details of the loss function we use, and conduct ablation study for the repulsion loss term. We further compare our proposed Multi-GCN, Clone-GCN and NodeShuffle with previous upsampling methods. Then, we provide the cross dataset quantitative results and additional qualitative results on PU600 dataset. We show qualitative experimental results on a new real-scanned dataset. We also discuss the discrepancy between the point-to-surface (P2F) metric and the qualitative performance. In addition, we discuss the failure case of our PU-GCN. Finally, we visualize our proposed large-scale dataset PU600 in addition to PU-GAN's dataset.

A Configuration of PU-GCN

Our proposed PU-GCN consists of three sub-network: (1) Dense feature extractor; (2) Upsampler; (3) Coordinator reconstructor. The detailed architecture of our PU-GCN is shown in Table 1. We use EdgeConv [25] as our basic GCN layer. The output of the $(l+1)$-th layer is represented as follows:

$$\mathbf{h}_{v_{l+1}} = mlp(concat(\mathbf{h}_{v_l}, \mathbf{h}_{u_l} - \mathbf{h}_{v_l}))|u_l \in \mathcal{N}^{(d)}(v_l) \tag{1}$$

where $\mathbf{h}$ denotes the node feature. Node u_l is one of the neighbors of node v_l in the l-th layer. We perform a max aggregation operation only in the last layer of the GCN block and the DenseGCN block.

Ablation study on dilation rate. We use dilated graph convolutions to enlarge the receptive fields without the need to increase the kernel size. However, one possible drawback of using dilated graph convolutions is that dilated graph convolutions in the low level features may encode information from the neighbors far away from the center points. Aggregating information from a large receptive field in the shallow layers will fail to learn good local representations. To verify this, we vary the dilation rate of the first layer of PU-GCN and measure the network's performance. Table 2 shows how dilated graph convolutions in the first

Table 1: **Detailed architecture of our PU-GCN.** The feature size is indicated by (N, k, C), which represents the number of points, the number of neighbors (kernel size) and the number of channels, respectively. Each path in the Inception DenseGCN is a DenseGCN block. We use the input patch with size 256×3 as an example to show the architecture of PU-GCN.

Module	Block	Layer	Input size	Output size	Kernel size	Dilation rate
Dense feature extrator	Dynamic GCN	KNN	(256, 1, 3)	-	16	1
		GCN	(256, 16, 3)	(256, 16, 24)	16	1
		Max Aggregation	(256, 16, 24)	(256, 1, 24)	16	1
	DenseGCN	KNN	(256, 1, 24)	-	16	1
		Concatenation Layer	-	(256, 16, 48)	-	-
		GCN	(256, 16, 48)	(256, 16, 24)	16	1
		Concatenation Layer	-	(256, 16, 72)	-	-
		GCN	(256, 16, 72)	(256, 16, 24)	16	1
		Concatenation Layer	-	(256, 16, 96)	-	-
		GCN	(256, 16, 96)	(256, 16, 24)	16	1
		Concatenation Layer	-	(256, 16, 96)	16	1
		Max Aggregation	(256, 16, 96)	(256, 1, 96)	16	1
	Inception DenseGCN (1)	KNN	(256, 1, 96)	-	9	3
		Inception (1a)	(256, 1, 96)	(256, 1, 48)	1	1
		Inception (1b)	(256, 1, 96)	(256, 1, 96)	9	1
		Inception (1c)	(256, 1, 96)	(256, 1, 48)	9	3
		Global Pooling	(256, 1, 96)	(256, 1, 1)	1	1
		Concatenation Layer	-	(256, 1, 289)	-	-
	Inception DenseGCN (2)	KNN	(256, 1, 289)	-	16	1
		Inception (2a)	(256, 1, 289)	(256, 1, 48)	1	1
		Inception (2b)	(256, 1, 289)	(256, 1, 96)	9	1
		Inception (2c)	(256, 1, 289)	(256, 1, 48)	9	3
		Global Pooling	(256, 1, 289)	(256, 1, 1)	1	1
		Concatenation Layer	-	(256, 1, 482)	-	-
Upsampler	NodeShuffle	-	(256, 1, 482)	(1024, 1, 482)	16	1
	Feature Compression	MLP (1)	(1024, 1, 482)	(1024, 1, 482)	1	1
		MLP (2)	(1024, 1, 256)	(1024, 1, 128)	1	1
Coordinate Reconstructor	Feature Reconstruction	MLP (1)	(1024, 1, 128)	(1024, 1, 64)	1	1
		MLP (2)	(1024, 1, 64)	(1024, 1, 3)	1	1

layer lowers the performance of PU-GCN. In our implementation, we do not use dilated graph convolutions in the first layer as well as the first DenseGCN block in PU-GCN, in order to preserve more local information.

Table 2: **Influence of dilation rate in the first layer of PU-GCN† on PU600.** Dilated convolutions in the first layer may encode information from neighbors that are far away from the center points and therefore lower the performance. PU-GCN† denotes the architecture that has 1 Inception block in the dense feature extractor and uses NodeShuffle as the upsampling module. d denotes the dilation rate in the first layer. **Bold** denotes the best performance. As expected, increasing d in the first layer deteriorates overall performance.

Dilation Rate	CD 10^{-3}	HD 10^{-3}	P2F 10^{-3}
PU-GCN† ($d = 7$)	0.341	2.556	1.605
PU-GCN† ($d = 5$)	0.341	2.568	1.584
PU-GCN† ($d = 1$)	**0.335**	**2.549**	**1.576**

B Loss function

Loss function. We use the modified Chamfer distance with weighted repulsion loss introduced in 3PU [27]. The Chamfer distance is for generating the point sets similar to the ground-truth point sets. The similarity is based on the Chamfer distance, which is also one of the metrics we use in the evaluation.

We also add a repulsion loss to encourage the local uniformity. The repulsion loss helps with distributing generated points rather than having them close to the original ones:

$$L_{rep} = \sum_{i=0}^{rN} \sum_{i' \in K(i)} \eta(||q_{i'} - p_i||_2)\omega(||q_{i'} - p_i||_2), \tag{2}$$

where rN is the number of output upsampled points, $K(i)$ is the index set of the k-nearest neighbors of point p_i, η is a decreasing function that penalizes p_i if it is too close to points in $K(i)$, and ω is a fast-decaying weight function that penalizes p_i only if it is too close to its neighboring points. The operator $||\cdot||_2$ denotes the Euclidean norm. Our final joint loss function is as follows:

$$L = C(P, Q) + \lambda L_{rep}, \tag{3}$$

where λ is a balancing term.

Ablation study on repulsion loss. In the main paper, we use $\lambda = 0.02$ in PU-GCN by default, which is the same as 3PU [27]. We do not perform any hyperparameter tuning and outperform the state-of-the-art. Here, we conduct ablation study with the influence of the parameter λ in repulsion loss. Table 3 shows that the quantitative results by using different λ in PU-GCN. Results indicate that choosing a suitable λ can further improve the quantitative performance of PU-GCN.

Table 3: **Influence of repulsion loss on PU600.** Choosing a suitable the parameter λ in the loss function can further improve the quantitative performance of PU-GCN.

Model	**CD** 10^{-3}	**HD** 10^{-3}	**P2F** 10^{-3}
PU-GCN ($\lambda = 1$)	0.335	2.612	1.568
PU-GCN ($\lambda = 0.1$)	**0.327**	2.615	1.516
PU-GCN ($\lambda = 0.02$)	0.333	**2.529**	1.529
PU-GCN ($\lambda = 0.002$)	0.331	**2.529**	**1.507**
PU-GCN ($\lambda = 0$)	0.330	2.555	1.527

C Comparison of Upsampling Modules

PU-Net [29] concatenates the outputs from a multi-branch CNN to upsample the point features. Multi-branch CNNs operate on each point separately using 1×1 convolutions, ignoring any neighborhood information. 3PU [27] and PU-GAN [14] merely duplicate the point features and append with additional feature vectors to distinguish the copied features from the original ones.

In contrast, our proposed upsampling modules Multi-branch GCN, Clone GCN, and NodeShuffle leverage graph convolution networks (GCNs). GCNs enable our upsampler to encode spatial information from point neighborhoods. The spatial information equips our upsampling modules with the ablity to capture local geometry. In addition, our upsampling modules learn new points from the latent space rather than simply duplicating the original points (unlike 3PU [27] and PU-GAN [14]). This alleviates the possibility of outputting similar points.

Comparison of Multi-branch CNN with Multi-branch GCN. Multi-branch CNN uses shared 1×1 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 [25]) 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, kNN (used in PU-GCN by default) or ball querying algorithms can be used to find the neighborhood and build the graph. Different GCN operators, like EdgeConv [25] (used in PU-GCN by default), GAT [24], MRConv [13], and GraphSAGE [7] 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.

Comparison of NodeShuffle with PointShuffle [19]. PointShuffle is introduced by Shi *et al.* [19], where the channel expansion and shuffle operation are combined to learn the sub-pixel information in 2D grid data. In our paper, we bring this efficient upsampling technique to non-Euclidean data. We adopt GCNs to learn the local geometry in the channel expansion part. Our NodeShuffle aims to upsample point clouds by learning the sub-point information.

D Additional Cross Dataset Results

D.1 Cross dataset quantitative result

In the main paper, we mention that we were unable to reproduce PU-GAN's [14] results from the code made available by its authors, because of the unstable nature of the inherent GAN architecture in PU-GAN. Therefore, to make fair comparison, we trained all models on PU-GAN's dataset (for PU-GAN we use their provided pretrained model) and here we show their qualitative comparison on our PU600 dataset. Table 4 provides the corresponding cross dataset quantitative results. Our PU-GCN still outperforms PU-Net and 3PU by a great extent. PU-GCN also outperforms PU-GAN w.r.t the CD and HD.

Table 4: **Cross dataset quantitative performance comparison of PU-GCN and the state-of-the-art.** The models are trained on PU-GAN;s dataset and evaluated on PU600. PU-GCN outperforms PU-Net and 3PU. PU-GCN also outperforms PU-GAN w.r.t CD and HD. **Bold** denotes the best performance.

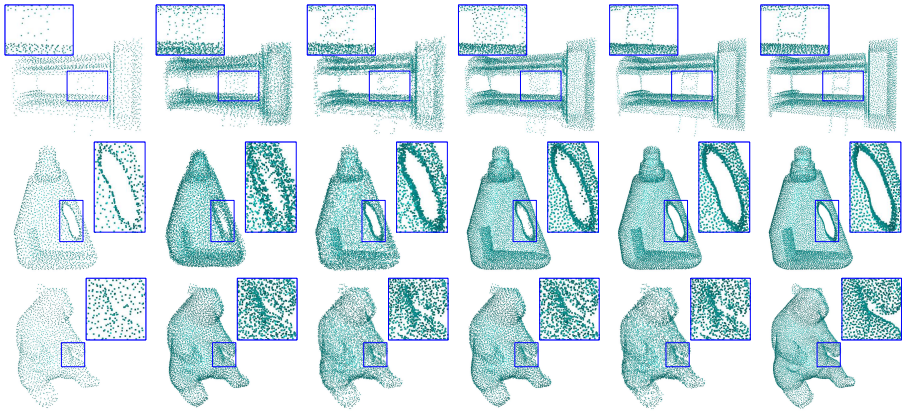
Network	**PU600**					
	Sparse (256) input			**Dense (2048) input**		
	CD 10^{-3}	**HD** 10^{-3}	**P2F** 10^{-3}	**CD** 10^{-3}	**HD** 10^{-3}	**P2F** 10^{-3}
PU-Net [29]	3.010	18.651	11.590	0.815	8.820	8.878
3PU [27]	3.363	20.138	10.171	0.901	7.903	4.026
PU-GAN [14]	2.546	19.769	**6.101**	0.370	4.749	**1.945**
PU-GCN	**2.457**	**14.481**	6.876	**0.351**	**2.763**	2.227

D.2 More qualitative results on PU600

Fig. 1 shows more qualitative results of PU-GCN compared to PU-Net [29], 3PU [27] and PU-GAN [14]. All models are trained on PU-GAN's dataset for fair comparison. Our PU-GCN clearly outperforms the state-of-the-art. PU-GCN successfully generates points with much less outliers (refer to the handle of the bottle in the second row, and the hand of the bear in the last row) and more fine-grained details (the ladder of the bed in the first row).

E More real-scanned results

In the main paper, we provide the results of PU-GCN when used to upsample real-scanned data from the KITTI dataset [6]. Here, we also experiment with the ScanObjectNN dataset [**?**]. ScanObjectNN contains real-scanned data of various indoor scenes and objects. Fig. 2 shows PU-GCN upsampling results on the original point clouds from the ScanObjectNN dataset. Our PU-GCN trained on PU600 dataset successfully produces high quality upsampled point clouds.



(a) Input (b) PU-Net (c) 3PU (d) PU-GAN (e) PU-GCN (f) GT

Fig. 1: **More qualitative results on PU600.** All models are trained on PU-GAN's dataset for fair comparison. Our PU-GCN outperforms the state-of-the-art with less outliers and more fine-grained details.

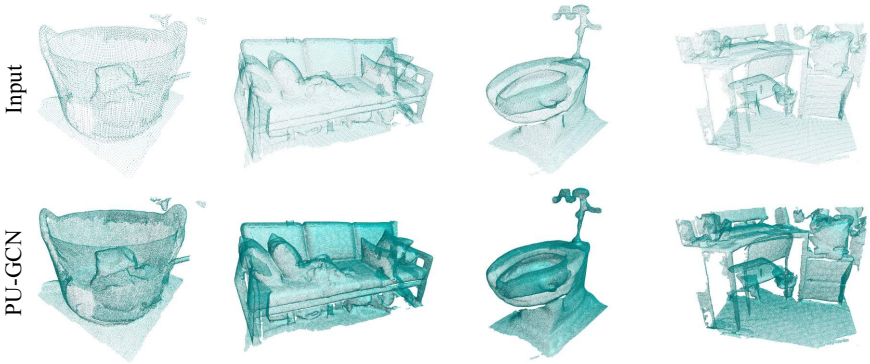
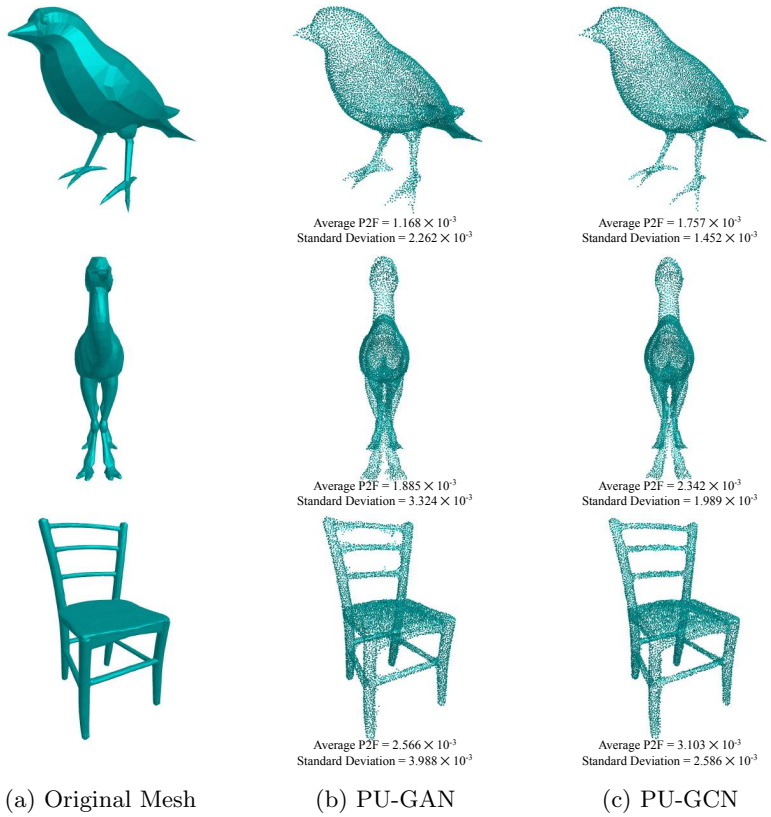


Fig. 2: **Upsampling real-scanned point clouds from ScanObjectNN dataset [?].** PU-GCN trained on PU600 produces high quality ×4 upsampled point clouds.

F Discrepancy between P2F and Qualitative Performance

In the main paper, we show that there exists a discrepancy between the point-to-surface (P2F) metric and qualitative performance. Fig. 3 below shows the discrepancy. The average of P2F values of our PU-GCN's results are higher than PU-GAN. However, one can observe that PU-GCN always generates less outliers and higher quality fine-grained details compared to PU-GAN (*e.g.*legs of the bird, between the legs of the camel, and the back of the chair). P2F is not as reliable as CD and HD for point cloud upsampling. We note that only point clouds are considered in the training pipeline. The P2F metric uses an unseen input form (mesh) in training may cause the discrepancy in the evaluation for point cloud upsampling. PU-GCN scores higher P2F than PU-GAN, however PU-GCN is much better in CD and HD as well as the qualitative performance.



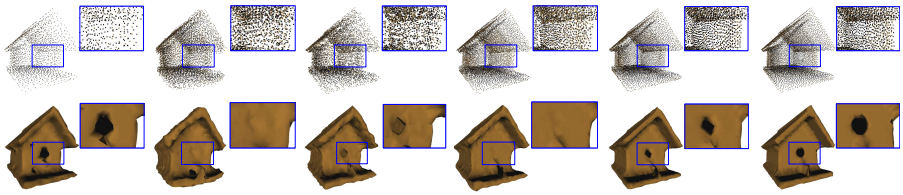
(a) Original Mesh (b) PU-GAN (c) PU-GCN

Fig. 3: **Visualization of original meshes and upsampling results of PU-GAN and PU-GCN**. PU-GCN generates points with better qualitative performance although PU-GCN scores higher P2F than PU-GAN.

G Failure Case

If the input point cloud has an inherent hole, all existing methods: PU-Net [29], 3PU [27], PU-GAN [14], and our PU-GCN, tend to overfill it (see Fig. 4 below). This happens because all the methods are designed to generate a (locally) uniformly distributed point cloud rather than distributing the points according to the density of the entire input point cloud. Better upsampling based on the density of the input can be researched to improve the quality.

Nonetheless, our PU-GCN maintains the intrinsic structure better than the state-of-the-art. We believe this is due to our feature extractor Inception DenseGCN being able to extract better multi-scale local information and our GCN-based upsampling moduels being able to capture a better local geometry.



(a) Input (b) PU-Net (c) 3PU (d) PU-GAN (e) PU-GCN (f) GT

Fig. 4: **Failure case** (the birdhouse from our PU600 testing dataset, the upsampling ratio is $\times 4$). All the existing methods, PU-Net [29], 3PU [27], PU-GAN [14] and our PU-GCN, tend to overfill the inherent holes.

H Details of Proposed Dataset PU600

We compile *PU600*, a new large-scale dataset with various levels of shape complexity. Our PU600 is is four times bigger than the largest publicly available point upsampling dataset (PU-GAN's dataset [14]). Fig. 5 and Fig. 6 show some of the training and testing samples from our PU600 dataset. We also show some examples from PU-GAN's dataset in Fig. 7 and Fig. 8. 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.

Fig. 5: **Training samples from PU600.**

Fig. 6: **Testing samples from PU600.**

Fig. 7: **Training samples from PU-GAN's Dataset.**

Fig. 8: **Testing samples from PU-GAN's Dataset.**