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 Bernard Ghanem
KAUST, Thuwal, Saudi Arabia
{guocheng.qian, abdulellah.abualshour, guohao.li, ali.thabet, bernard.ghanem}@kaust.edu.sa

Abstract

Upsampling sparse, noisy, and non-uniform point clouds is a challenging task. In this paper, we propose 3 novel point upsampling modules: Multi-branch GCN, Clone GCN, and NodeShuffle. Our modules use Graph Convolutional Networks (GCNs) to better encode local point information. Our upsampling modules are versatile and can be incorporated into any point cloud upsampling pipeline. We show how our 3 modules consistently improve state-of-the-art methods in all point upsampling metrics. We also propose a new multi-scale point feature extractor, called Inception DenseGCN. We modify current Inception GCN algorithms by introducing DenseGCN blocks. By aggregating data at multiple scales, our new feature extractor is more resilient to density changes along point cloud surfaces. We combine Inception DenseGCN with one of our upsampling modules (NodeShuffle) into a new point upsampling pipeline: PU-GCN. We show both qualitatively and quantitatively the advantages of PU-GCN against the state-of-the-art in terms of fine-grained upsampling quality and point cloud uniformity. The source code of this work is available at https://github.com/guochengqian/PU-GCN.

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 applications in robotics and self-driving cars. Due to hardware and computational constraints, 3D scanning sensors often produce sparse, noisy, and non-uniformly dense point clouds. These deficiencies are more obvious for objects that are small or far away from the camera. Overcoming these disadvantages is challenging, but can greatly enhance the performance of point cloud based methods. Consequently, point cloud upsampling, which is the task of converting sparse incomplete point sets into clean, complete, dense, and locally uniform ones, is attracting much attention recently, and it is the focus of this paper.

Early optimization based methods [1, 16, 7, 26] attempt to tackle point cloud upsampling by using various handcrafted shape priors. More recently and inspired by their success in image super-resolution [4, 15, 9, 19], deep learning methods now achieve state-of-the-art results in point cloud upsampling [30, 31, 28, 14]. Most deep upsampling pipelines comprise two major modules, feature extraction and point upsampling. The performance of the point upsampling module defines the effectiveness of the final network. Current methods use either a multi-branch CNN [31] or a duplicate-based approach [28, 14] to upsample points. Multi-branch CNNs operate on each point separately, ignoring any neighborhood information, while duplicate upsam-
pling methods tend to generate point patches similar to the original point clouds. These shortcomings lead to upsampled point clouds that lack local details. To better represent locality, we leverage the power of graphs and specifically Graph Convolutional Networks (GCNs). GCNs are considered a versatile 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 [25, 13, 12]. In this paper, we use GCNs to design novel point cloud upsampling modules (refer to Figure 1), which are better equipped at encoding local information and learn to generate new point patches instead of merely replicating parts of the input.

Point clouds are often non-uniformly distributed, and represent objects with varying part sizes. Multi-scale features are an effective way to encode these characteristics, and are essential for obtaining uniform and dense point clouds. Recent works like PU-Net [31], extract point features at different downsampling levels. This architecture can encode multi-scale features, however, downsampling leads to loss of fine-grained details. In 3PU [28], the authors proposed a progressive upsampling network using different number of neighbors (kernel sizes) in subsequent upsampling units. This effectively achieves different receptive fields and encodes multi-scale information. However, 3PU is computationally expensive due to its progressive nature. We tackle the feature learning problem using GCNs. And, following its prevalent usage in image recognition [21, 22, 20] 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 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 to significantly improve their performance. (2) We design Inception DenseGCN, a feature extraction block that encodes multi-scale information effectively and efficiently. 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, we show the superior performance of PU-GCN against the state-of-the-art. (3) We compile PU660, a new large-scale dataset with varying levels of shape complexity. PU660 consists of 660 3D models with 210 models compiled from the datasets of previous point clouds upsampling methods [31, 28, 14] and 450 more models we collect from ShapeNet [3]. Our PU660 is 4 times bigger than the largest publicly available point upsampling dataset. We show the advantages of our new dataset using our approach as well as other state-of-the-art methods.

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 [10, 6, 24, 17, 25] have been proposed in recent years. Kipf et al. [10] simplify spectral graph convolutions with a first-order approximation. Since each node in a GCN layer can encode information from its neighbors (e.g. defined by a fixed adjacency matrix or a dynamic one that can change from layer to layer), there exist many ways to perform this encoding. For this purpose, Hamilton et al. [6] proposed different aggregators (i.e. mean, LSTM, and pooling aggregators) to effectively encode features from a node’s neighborhood. Velickovic et al. [24] adapted a self-attention mechanism to GCNs, which attends different weights to different neighbors. To learn better hierarchical feature representation, graph pooling methods such as DIFFPool [29] and SAGPooling [11] are proposed. Recently, Li et al. [13, 12] introduced residual/skip connections and dilated convolutions to GCNs, and successfully trained high capacity GCN architectures over 100 layers in depth. Previous GCN works mainly investigate discriminative models for node classification or graph classification tasks. However, due to the unordered and irregular nature of graph data, generative tasks are still considered difficult to perform. In particular, upsampling techniques as indispensable components for generative models are under-explored.

Multi-scale feature extraction. Inception architectures [21, 22, 20] 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. Moreover, dense or residual connections are also used to aggregate information and make it possible for the network to go deeper. Inspired by success of the Inception architecture for CNNs, Kazi et al. [8] propose 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 or max pooling. We adopt the Inception concept in our work, thus improving 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. [7], in which they propose 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 methods 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], SpiderCNN [27], DGCNN [25], KPConv [23], etc. Yu et al. [31] introduced the neural network PU-Net that learns multi-scale features per point, expands the point set via a multi-branch CNN in feature space, and reconstructs an upsampled point set from those features. However, PU-Net needs to downsample the input first to learn multi-scale features, which causes unnecessary resolution loss. Yu et al. [30] also proposed EC-Net, which is 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-notaation for training. Wang et al. [28] presented 3PU, which is 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] presented PU-GAN, which is a Generative Adversarial Network (GAN) designed to learn upsampled point distributions in point clouds from latent space. Different from previous work, they upsample ×6 at first then use farthest upsampling to sample ×4 points to produce the final output. While PU-GAN focuses on improving the quantitative performance and uniformity of upsampled point clouds, it does not specifically focus on designing an effective upsampling module.

3. Methodology

3.1. Upsampling Modules

To effectively upsample point clouds, we propose three different upsampling modules: Multi-branch GCN, Clone GCN, and NodeShuffle, whose details are provided next.

Multi-branch GCN. Figure 2a illustrates 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 \( \mathcal{V}_i \in \mathbb{R}^{N \times C} \), we obtain output \( \mathcal{V}^{UP}_{i+1} \in \mathbb{R}^{rN \times C} \) as follows:

\[
\mathcal{V}^{UP}_{i+1} = \mathcal{T}(\mathcal{F}_1(\mathcal{V}_i), \mathcal{F}_2(\mathcal{V}_i), ..., \mathcal{F}_r(\mathcal{V}_i)),
\]

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.

Contrary to Multi-branch CNNs, our Multi-branch GCN uses graph instead of regular convolutions, enabling it to encode spatial information from point neighborhoods.

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 Figure 2b). Instead of using multiple branch convolutions, 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:

\[
\mathcal{V}^{UP}_{i+1} = \mathcal{T}(\mathcal{F}(\mathcal{V}_i), \mathcal{F}(\mathcal{F}(\mathcal{V}_i)), \mathcal{F}^3(\mathcal{V}_i), ..., \mathcal{F}^r(\mathcal{V}_i)),
\]

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

NodeShuffle. Both Multi-branch GCN and Clone GCN run \( r \) graph convolutions. To alleviate this computational burden, we propose NodeShuffle (NS), which is illustrated in Figure 2c. Inspired by PixelShuffle [19] from the image
Figure 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. \(KNN\) is applied at the first layer to build the graph and the node neighborhoods. The green \(\gg\) denotes feature-wise concatenation.

super-resolution literature, \(\mathcal{NS}\) is an efficient graph convolutional upsampling layer. Given node features \(\mathcal{V}_l\) with shape \(N \times C\), \(\mathcal{NS}\) will output the new node features \(\mathcal{V}^{U\text{p}}_{l+1}\) with shape \(rN \times C\) as follows:

\[
\mathcal{V}^{U\text{p}}_{l+1} = \mathcal{NS}(\mathcal{V}_l) = \mathcal{PS}(\mathcal{W}_l \ast \mathcal{V}_l + b_l),
\]

(3)

where \(\mathcal{PS}\) is a periodic shuffling operator that rearranges the graph of shape \(N \times rC\) to \(rN \times C\). The \(\mathcal{NS}\) 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 \(\mathcal{W}_l\) and \(b_L\). (2) Periodic shuffling: we rearrange the output of channel expansion to shape \(rN \times C\). \(\mathcal{NS}\) runs only 1 GCN operation, as opposed to \(r\) such operations in Multi-branch and Clone GCN.

3.2. Feature Extractor: Inception DenseGCN

Point clouds scanned using 3D sensors are sparse and non-uniform. They also often include objects of various sizes and point resolutions. In order to encode the multiscale nature of point clouds, we propose a new Inception DenseGCN feature extractor, which effectively integrates the DenseGCN module from [13] into the Inception GCN module from [8]. We favor dense over residual connections here, since the former utilizes features from previous layers, as well as different inception paths.

Figure 3 shows an Inception DenseGCN block. In our experiments, we use 3 DenseGCN blocks per Inception block. Each DenseGCN block is defined by a number of node neighbors \(k\), dilation rate \(d\), and number of filters \(c\). Note that the dilated graph convolution operator was introduced in [13], from which we adopt the DenseGCN module. Similar to the 2D case, this operator increases the receptive field without reducing spatial resolution. Additionally, we add a global pooling layer to extract global contextual in-
formation. Each Inception block outputs a concatenation of the 3 DenseGCN blocks and the global pooling layer.

3.3. PU-GCN Architecture

We combine our Inception DenseGCN extractor with our novel upsampling modules, followed by a coordinate reconstructor. We name this new network PU-GCN. Our architecture is illustrated in Figure 4. Given a point cloud of size \(N \times 3\), we compute dense features of size \(rN \times C\) using our dense feature extractor. We then upsample the \(N \times C\) features to \(rN \times C'\) using our upsampler. Finally, the coordinate reconstructor generates the \(rN \times 3\) upscaled point cloud. Below, we explain each component in more detail.

**Dense feature extractor.** We use 1 GCN layer followed by 1 DenseGCN layer 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 details about the influence of dilation rate in the first layers are discussed in the supplementary material. The output of DenseGCN will be passed into several densely connected Inception DenseGCN blocks. In our implementation, we only use two such Inception DenseGCN blocks in PU-GCN. We experiment with the number of Inception DenseGCN blocks in Section 4.5. 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 2 sets of MLPs to compress features to \(rN \times C'\). We experiment with the different upsampling modules in Section 4.5.

**Coordinate reconstructor.** We reconstruct nodes from latent space to 3D coordinate space, resulting in the desired denser point cloud of size \(rN \times 3\). We use the same coordi-
nate reconstruction approach as 3PU [28], in which a set of 3D coordinates is regressed using 2 sets of MLPs.

4. Experiments

4.1. Datasets

We propose a new dataset for point cloud upsampling, denoted as PU660. It consists of 660 3D models split into 551 training samples and 109 testing samples. The training set contains 171 3D models compiled from the datasets used by PU-Net [31], 3PU [28], and PU-GAN [14], in addition to 380 different models collected from ShapeNet [3]. The test set contains 39 models compiled from the datasets used by PU-Net [31], 3PU [28], and PU-GAN [14] and 70 more models from ShapeNet. The models from ShapeNet were randomly chosen from 10 different categories and 450 different shapes with various complexities to encourage diversity. Overall, PU660 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 PU660 and the latest dataset proposed by PU-GAN [14], which contains only 147 3D models for training and 13 models for testing. More details of PU660 and a comparison with PU-GAN’s dataset can be found in the supplementary material.

4.2. Loss Function and Evaluation Metrics

We use the modified Chamfer distance with weighted repulsion loss introduced in 3PU [28]:

\[ L = C(P, Q) + \lambda L_{\text{rep}}, \]  

(4)

where \( C(P, Q) \) is the modified Chamfer distance loss, \( P \) is the predicted point cloud, and \( Q \) is the ground truth point cloud. \( L_{\text{rep}} \) denotes the repulsion loss, which encourages generated points to be far from the original ones. Adding \( L_{\text{rep}} \) tends to generate more uniformly distributed point clouds. \( \lambda \) is a tradeoff coefficient.

**Evaluation Metrics.** Following previous works, we use the Chamfer distance (CD), Hausdorff distance (HD), and point-to-surface distance (P2F) with respect to ground truth meshes as evaluation metrics. For testing, we use Poisson disk sampling to sample input and ground truth point clouds from the original meshes (refer to Figure 5a and 5f). More details about the loss function and evaluation metrics can be found in the supplementary material.

4.3. Implementation Details

We use TensorFlow for all our experiments. In training, we use the same farthest point sampling strategy as 3PU to crop 200 patches from each 3D model as the input for the network. In total, we obtain 110,200 training patches in PU660. Each patch consists of 256 input points sampled from the original model. We use the same method to sample 1024 points to form the ground truth upsampled point cloud. We employ the same data pre-processing and augmentation techniques suggested by 3PU, which include point cloud normalization, random rotation, scaling, and point perturbation with Gaussian noise. Adam is used as the optimizer during training with a learning rate of 0.0005 and beta 0.9. In all the experiments, we train PU-GCN for 100 epochs with batch size 28 on NVIDIA Tesla V100 (16GB) GPU.

We compare PU-GCN against PU-Net [31], 3PU [28], and PU-GAN [14]. We train all these upsampling methods using our P660 dataset, as well as PU-GAN’s dataset. We test all methods on sparse and dense input resolutions, each containing 256 and 4096 points respectively. We report results using a \( \times 4 \) upsampling rate. We note that PU-GAN upsamples \( \times 6 \) points then uses farthest point sampling to obtain the \( \times 4 \) output. For fair comparison, we also report its results without the farthest sampling module. We refer to this architecture as PU-GAN∗. We train PU-GAN∗ using the same hyperparameters as PU-GAN on both datasets. To show the effectiveness of our upsampling modules, we replace the upsampling modules of previous architectures with our three new modules. We train these networks on PU660 and compare their performance against PU-GCN. All models converge before the maximum epochs. We use the model from the last epoch to evaluate the performance as suggested by PU-Net, 3PU, and PU-GAN.

4.4. Quantitative and Qualitative Results

**Quantitative results.** Table 1 reports the performance results of our architecture compared to PU-Net, 3PU, PU-GAN, and PU-GAN∗ on PU-GAN’s dataset and PU660. We observe that our PU-GCN integrating Inception DenseGCN outperforms other methods. For instance, PU-GCN maintains significant improvement over 3PU on the CD metric using sparse and dense inputs on both datasets, showing the importance of the Inception DenseGCN feature extractor and the NodeShuffle upsampling module. For fair comparison, we compare our performance to PU-GAN excluding the farthest sampling strategy (i.e. PU-GAN∗) as discussed earlier. We see that PU-GCN outperforms PU-GAN∗ in all metrics on both datasets. Although PU-GAN uses the farthest point sampling strategy, PU-GCN outperforms their method on all metrics for sparse inputs and most metrics for dense inputs in both datasets. We note that all methods achieve lower performance when trained and evaluated on PU660 compared to PU-GAN’s dataset, which illustrates the complexity and diversity of our proposed PU660. The dataset presents a challenge to state-of-the-art methods and alleviates the potential issue of overfitting, as compared to PU-GAN’s dataset that is much smaller and less diverse.

**Qualitative results.** Figure 5 shows qualitative results of the performance of our PU-GCN method compared to state-of-the-art on the PU660 dataset. We show point cloud up-
### 4.5. Ablation Study

**Upsampling Modules.** We evaluate the performance of our proposed upsampling modules compared to the original upsampling units used in state-of-the-art architectures. We use PU-GCN\* as our baseline architecture, which uses a single Inception block in its dense feature extractor and NodeShuffle as the upsampling module. Table 2 shows 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 varying impact depending on the architecture. For PU-Net, Clone GCN performs best on CD and P2F. Interestingly, we find that Multi-branch CNN, which is used by PU-Net, performs better than Multi-branch GCN overall. It is possible that Multi-branch GCN tends to include some far away points as neighbors using $KNN$ in sparse point clouds, and therefore local information aggregation is sacrificed. For 3PU, NodeShuffle gives the best results in all the cases, while Multi-branch GCN performs second best. A qualitative comparison of our different upsampling modules integrated in 3PU is illustrated in Figure 1. NodeShuffle shows smoother and more uniform outputs and clear improvement in generating fine-grained details of the model compared to other upsampling methods. For our 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.

| Network | Sparse (256) input | Dense (4096) input |
|---------|-------------------|-------------------|
|         | CD $10^{-3}$ | HD $10^{-3}$ | P2F $10^{-3}$ | CD $10^{-3}$ | HD $10^{-3}$ | P2F $10^{-3}$ |
| PU-Net [31] | 3.213 | 25.235 | 9.099 | 0.914 | 24.328 | 8.919 |
| 3PU [28] | 3.111 | 17.154 | 7.630 | 0.309 | 2.709 | 3.782 |
| PU-GAN [14] | 2.403 | 17.668 | 8.005 | **0.182** | **2.275** | **1.179** |
| PU-GCN | 2.504 | 16.565 | 5.514 | 0.279 | 2.850 | 2.697 |
| **PU660** | **2.249** | **12.628** | **4.043** | **0.209** | **1.719** | **1.405** |

Table 2: Ablation study on upsampling modules on PU660 using sparse (256) input. Experiments show that our upsampling modules can transfer/generalize to different architectures. By simply replacing the original upsampling module with one of our proposed ones, the performance of different architectures all improve. PU-GCN\* uses a single Inception DenseGCN block in its feature extractor. **Bold** denotes the best performance for each architecture: PU-Net, 3PU, PU-GAN and our PU-GCN.

**Inception Modules.** We conduct an ablation study on our Inception DenseGCN and report the results in Table 3. We validate the effectiveness of our Inception DenseGCN by replacing it with the dynamic GCN used in 3PU [28], which has slightly more parameters than our In-
Figure 5: Comparing point cloud upsampling (×4) and surface reconstruction results produced by different methods (b-e) from inputs (a). Our PU-GCN produces the best results overall, with uniform and fine-grained detailed upsampled point clouds. The reconstructed surfaces are smoother with less wrinkles or bulges and maintain the intricate structures of the original shape.

ception DenseGCN. Experiments show that our Inception DenseGCN outperforms the dynamic GCN feature extractor in all metrics. As expected, using global pooling inside our Inception DenseGCN improves the performance. PU-GCN outperforms PU-GCN†, clearly showing that inserting more Inception DenseGCN blocks can further improve upsampling performance.

4.6. More Experiments

Upsampling point clouds of varying sizes. Figure 7 shows qualitative examples of upsampling point clouds with PU-GCN for different input sizes. Our architecture always produces uniform upsampled point clouds regardless of input cloud size. This indicates that even if PU-GCN is trained on patches with 256 points, our model can be generalized to point clouds with different sizes. As expected, PU-GCN generates better quality results when the input point cloud is denser.

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 the qualitative results of this experiment in Figure 6.
and compare our PU-GCN method against PU-GAN. Both models were trained using the same augmentation strategy of point cloud perturbation. Results show that our PU-GCN can preserve fine-grained details with very few outliers, even in the presence of additive noise, while PU-GAN tends to generate many outliers.

Figure 6: Upsampling point clouds with additive Gaussian noise. The top row shows the input point clouds with different additive noise levels: 0, 0.01, and 0.02 from left to right, respectively. The middle row shows the \( \times 4 \) upsampled point clouds using PU-GAN. The bottom row shows the upsampled point clouds using PU-GCN (Ours).

Table 3: Ablation study on the effect of Inception DenseGCN and global pooling using sparse (256) input on PU660. Using a single Inception DenseGCN block in PU-GCN\(^\dagger\) outperforms the architecture integrating the dynamic GCN feature extractor used in 3PU [28]. The global pooling layer in our feature extractor improves performance. By increasing the number of Inception DenseGCN blocks, we observe further improvement in PU-GCN. Bold denotes the best performance.

| Ablation                              | CD       | HD       | P2F      |
|---------------------------------------|----------|----------|----------|
| Dynamic GCN feature extraction [28]   | 2.297    | 13.081   | 4.463    |
| No global pooling                     | 2.276    | 12.730   | 4.120    |
| PU-GCN\(^\dagger\)                    | 2.263    | 12.874   | 4.079    |
| PU-GCN                                | 2.249    | 12.628   | 4.043    |

Figure 7: Upsampling point clouds of different sizes using PU-GCN. The top row shows the input point clouds, and the bottom row shows the \( \times 4 \) upsampled outputs.

5. Conclusion

We introduce three novel point cloud upsampling modules Multi-branch GCN, Clone GCN, and NodeShuffle, which improve state-of-the-art upsampling pipelines when they are used instead of the original upsampling. We validate the performance of these upsampling modules quantitatively and qualitatively and illustrate the value they add when used in other network architectures. We also introduce the Inception DenseGCN feature extractor, which is an improvement upon Inception GCN. We use densely concatenated graph convolutions (DenseGCNs) and global pooling to encode multi-scale information and allow for more efficient learning from input point clouds. Since the datasets used by state-of-the-art methods are small and saturated, we further compile and introduce a new large-scale dataset for point cloud upsampling, called PU660. We evaluate our architecture in addition to state-of-the-art approaches on this dataset. Our architecture that integrates Inception DenseGCN and NodeShuffle upsampling, denoted as PU-GCN, outperforms the state-of-the-art on previous upsampling datasets and our PU660.

Acknowledgments

The authors thank Silvio Giancola for his help with the project. This work was supported by the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research through the Visual Computing Center (VCC) funding.
References

[1] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Cláudio T. Silva. Computing and rendering point set surfaces. *IEEE Trans. Vis. Comput. Graph.*, 9:3–15, 2003. 1, 2

[2] Fausto Bernardini, Joshua Mittleman, Holly E. Rushmeier, Cláudio T. Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5:349–359, 1999. 6

[3] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savaresi, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 2, 5

[4] Chao Dong, Chen Change Loy, Kaiming He, and Xiaou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:295–307, 2014. 1

[5] Alberto Garcia-Garcia, Francisco Gomez-Donoso, J. A. Rodríguez, Sergio Orts, Miguel Cazorla, and Jorge Azorín López. Pointnet: A 3d convolutional neural network for real-time object class recognition. 2016 *International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1584, 2016. 3

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017. 2

[7] Hui Huang, Shihao Wu, Minglung Gong, Daniel Cohen-Or, Uri M. Ascher, and Hao Zhang. Edge-aware point set resampling. *ACM Trans. Graph.*, 32(9):1–9:12, 2013. 1, 2

[8] Anees Kazi, Shayan Shekarforoush, S Arvind Krishna, Henrik Burwinkel, Gerome Vivar, Karsten Kortüm, Seyed-Ahmad Ahmadi, Shadi Albarqouni, and Nassir Navab. Inceptiongcn: Receptive field aware graph convolutional network for disease prediction. In *International Conference on Information Processing in Medical Imaging*, pages 73–85. Springer, 2019. 2, 4

[9] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1646–1654, 2015. 1

[10] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2

[11] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019. 2

[12] Guohao Li, Matthias Müller, Guocheng Qian, Itzel C. Delgadillo, Abdulellah Abualshour, Ali K. Thabet, and Bernard Ghanem. Deepgcn: Making gcns go as deep as cnns. *ArXiv*, abs/1910.06849, 2019. 2

[13] Guohao Li, Matthias Miller, Ali Thabet, and Bernard Ghanem. Deepgcn: Can gcns go as deep as cnns? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019. 2, 4

[14] Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-gan: a point cloud upsampling adversarial network. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. 1, 2, 3, 5, 6

[15] Bee Lim, Sanghyun Son, Heewon Kim, Seunghun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1132–1140, 2017. 1

[16] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH ’07, New York, NY, USA, 2007. ACM. 1, 2

[17] Trang Pham, Truyen Tran, Dinh Phung, and Svetla Venkatesh. Column networks for collective classification. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 2

[18] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 3

[19] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zeheh Wang. 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)*, pages 1874–1883, 2016. 1, 3

[20] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2016. 2

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2014. 2

[22] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2015. 2

[23] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *Proceedings of the IEEE International Conference on Computer Vision*, 2019. 3

[24] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 2

[25] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019. 2, 3

[26] Shihao Wu, Hui Huang, Minglung Gong, Matthias Zwicker, and Daniel Cohen-Or. Deep points consolidation. *ACM Trans. Graph.*, 34(6):176:1–176:13, Oct. 2015. 1, 3

[27] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. *arXiv preprint arXiv:1803.11527*, 2018. 3
[28] Wang Yifan, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. Patch-based progressive 3d point set upsampling. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019. 1, 2, 3, 5, 6, 8

[29] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In Advances in Neural Information Processing Systems, pages 4800–4810, 2018. 2

[30] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Ec-net: an edge-aware point set consolidation network. In ECCV. 2018. 1, 3

[31] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018. 1, 2, 3, 5, 6